TM240

Ladder Diagram (LD)

PERFECTION IN AUTOMATION
www.br-automation.com

Requirements

Training modules:	TM210 – The Basics of Automation Studio TM223 – Automation Studio Diagnostics
Software	Automation Studio 4.2.5 or higher
Hardware	None

Table of contents

1 Introduction

Ladder Diagram is a visual programming language that was originally developed as a way to replace programming hard-wired relay logic. Ladder Diagram is commonly used and included in the IEC standard[1].

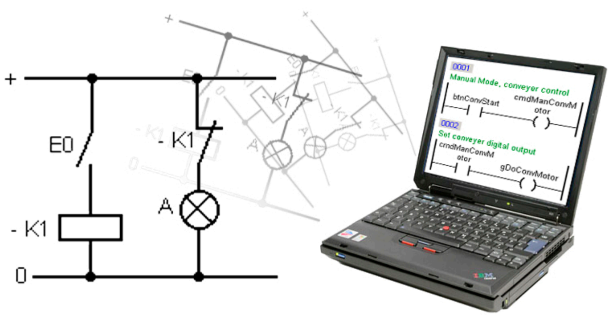


Figure 1: Ladder Diagram

In the following chapters, you will be provided an overview of the features of programming with Ladder Diagram. Individual functions will be explained using examples.

1.1 Learning objectives

This training module uses selected examples illustrating typical application tasks to help you learn how to work with Ladder Diagram (LD).

- You will learn the operating principle of the LD programming language.
- You will learn how to use the LD editor and embedded diagnostic tools.
- You will learn the difference between an action, a function and a function block, as well as how to use them.
- You will learn how to integrate digital and analog signals into LD programs in addition to the basics of logic programming.
- You will learn the elements used to control the flow of an LD program.

[1] The IEC 61131-3 standard is the only valid international standard for programming languages used on programmable logic controllers. This standard also includes Instruction List, Structured Text and Function Block Diagram.

2 Ladder Diagram

2.1 Interesting information about Ladder Diagram

The original concept of the PLC (programmable logic controller) was developed in the USA in 1968. The PLC concept was developed as a microprocessor-based, programmable replacement for hard-wired systems.

The PLC itself was based around the ladder diagram, which is a schematic representation of a logical control system based on relay circuitry. At the time, the concept became a very fast way of quickly setting up and programming a simple logical control system with relatively little training.

Many manufacturers based their programming systems on ladder diagrams. Unfortunately, the lack of an open standard meant that each vendor's system was slightly different. Many manufacturers often added special commands in order to increase functionality.

By the beginning of the 1990s, there were literally thousands of PLC manufacturers, each with their own programming interfaces and command sets. Although the programs developed on different systems were similar, their structure and the commands they used often varied greatly.

In 1979, a working group was set up by the International Electrotechnical Commission (IEC) to create a common standard for PLCs. This working group decided to develop a new standard (what became IEC 61131).

Part III, "Programming Languages for PLCs", was published in 1993 and included the specification for PLC software. Part III covers PLC configuration, programming and data storage.

2.2 Features and options

Ladder Diagram is a graphics-based programming language. Symbolic representations of electrical circuits are used that coincide with the schematic symbols used in conventional circuit diagrams. These symbols and connecting lines are used to program the necessary logic.

Ladder Diagram has the following features:

- Visual programming
- Circuit diagram rotated 90°
- Simple, clear programming
- Self-explanatory
- Easy to diagnose

The Ladder Diagram editor allows you to:

- Use digital inputs / outputs and internal boolean variables
- Use analog inputs / outputs
- Use function blocks, functions and actions
- Control the program flow (jumps, program abort)
- Use tools for diagnostics

2.3 The Ladder Diagram editor

All Ladder Diagram functions can be implemented using the Ladder Diagram Catalog, the editor's toolbar and the keyboard.

Toolbar

Basic Ladder Diagram symbols can be inserted using the toolbar in the Ladder Diagram editor. The icons in the toolbar are enabled or disabled depending on the position of the cursor.

Figure 2: The toolbar in the Ladder Diagram editor

Toolbox - Ladder Diagram Catalog

When you open the Ladder Diagram editor, the Toolbox shows the Ladder Diagram Catalog. It shows a list of all available functions and function blocks. The list can be narrowed down by setting filters. Items from the Ladder Diagram Catalog can be added to the program using drag-and-drop.

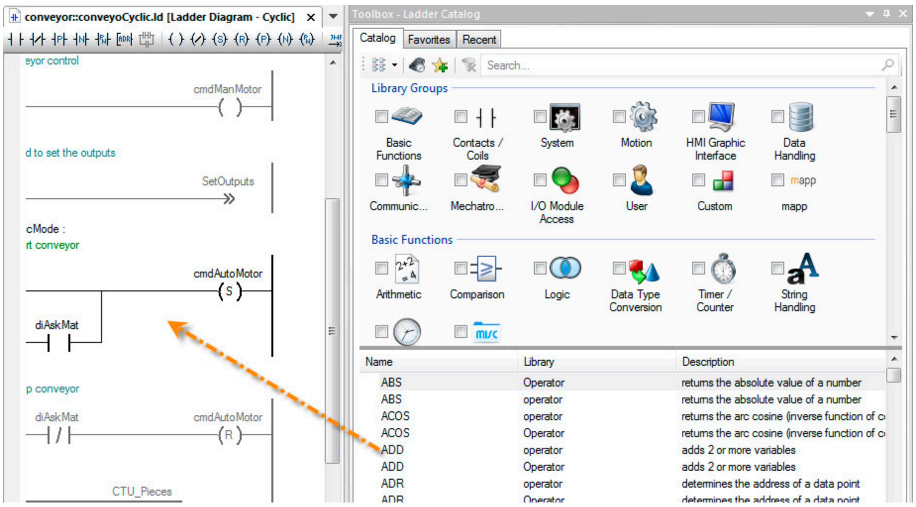


Figure 3: Adding Ladder Diagram functions using the Ladder Diagram Catalog

Viewing options

How Ladder Diagram icons and the editor are displayed can be customized. For example, it is possible in the Ladder Diagram editor's shortcut menu or from the **View** menu to show or hide data types, comments and the scope of connections and variables.

It is also possible to configure the size of networks and ladder diagram symbols with the **Tools/Options** menu.

The standard width of networks can be configured with the "Minimum count of columns" setting. As long as the network has the same number of columns as this value or less, then the outputs of all networks will align perfectly with one another.

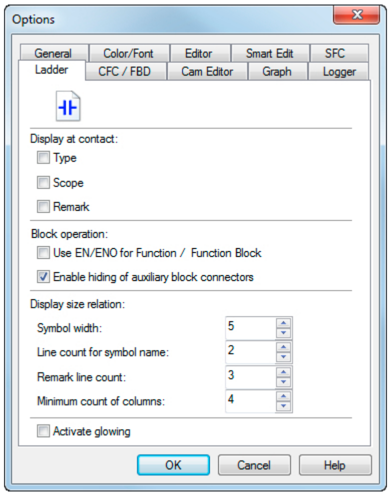


Figure 4: Editor-specific Ladder Diagram settings

Variable assignment

Variables can be assigned to highlighted contacts using the **<space bar>** or dragging and dropping them from the logical view.

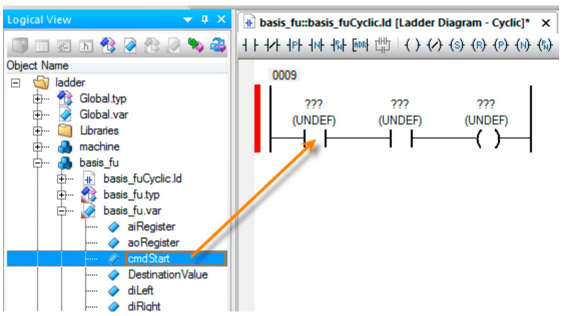


Figure 5: Assigning variables to contacts with drag and drop

All the functions of the Ladder Diagram editor can be operated with the mouse or keyboard (10.1 "Keyboard shortcuts in the editor").

Programming \ Editors \ Graphic editors \ Ladder Diagram editor

- Toolbar
- Ladder Diagram Catalog

Project management \ The workspace \ AS Settings \ Ladder Diagram editor settings

Diagnostics

Monitor mode and Powerflow can be used for diagnostics in Ladder Diagram. All logic paths that are TRUE will then be shown in color. In addition, a variable's tooltip indicates its process value and data type. The Automation Studio variable watch feature rounds out the range of diagnostic functions.

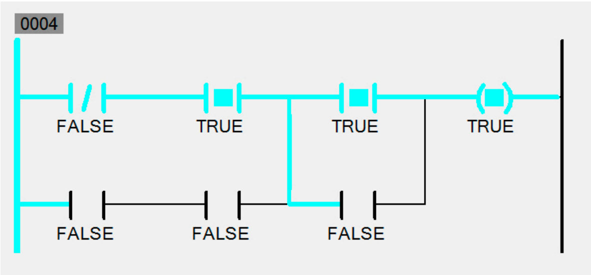


Figure 6: Ladder diagram with Powerflow enabled

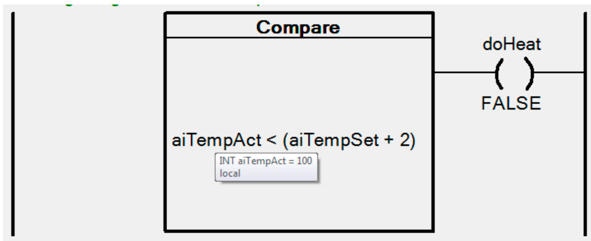


Figure 7: Display of variable tooltip

Diagnostics and Service \ Diagnostics Tool \ Monitors \ Programming languages in monitor mode \ Powerflow

Diagnostics and Service \ Diagnostic Tool \ Variable watch

3 Basic elements of Ladder Diagram

The following illustrations show the basic elements of a ladder diagram. On the left side is the permanent "current-carrying" vertical power rail (1). To the right is a normally open contact (2), on top of which is the process variable (3) that is being used to store the value of the contact on the controller. Command lines (4) lead off to the right where they connect to other contacts or coils.

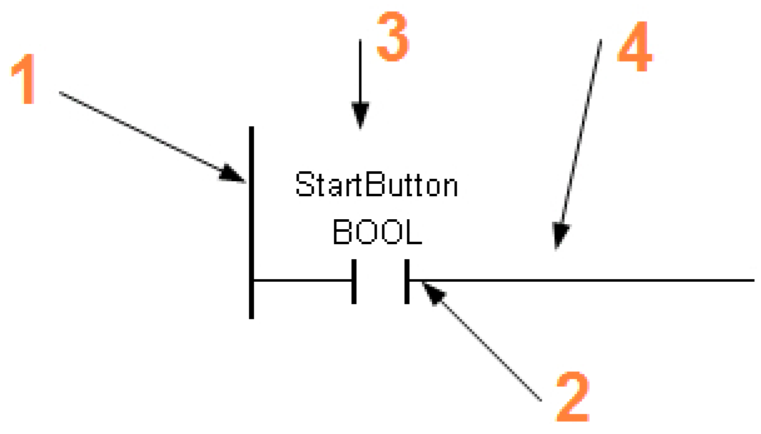


Figure 8: Basic elements of Ladder Diagram

A ladder diagram essentially consists of two parts. The left side contains the logic that results in the outputs on the right side. Elements on the far right are called coils. The value of the coil can be used as a digital output, for example.

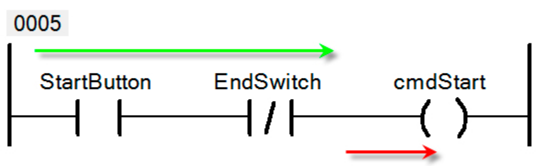


Figure 9: Logic (green), output or switching command (red)

3.1 Networks

A Ladder Diagram program is divided into smaller program units. These are referred to as networks.

A network consists of contacts, which can be connected in parallel or in series, and coils. The power supply is at the far left, with the reference potential located to the far right.

A network can consist of 50 rows and 50 columns. It is only complete if at least one coil or result value has been configured on the far right.

A ladder diagram can consist of multiple networks, with each assigned its own network number in ascending order by the system.

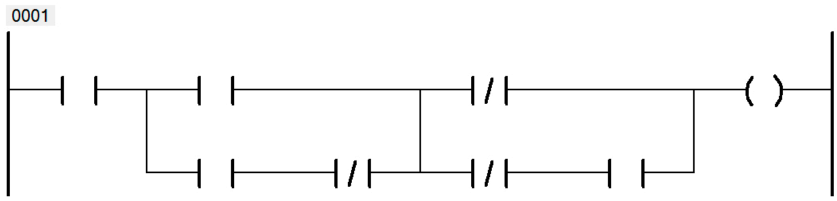


Figure 10: Network structure

Comments can be added to each network. One can be inserted using the editor toolbar or by pressing the **<D>** key.

Programming \ Programs \ Ladder Diagram (LD) \ Network

Programming \ Editors \ Graphic editors \ Ladder Diagram editor \ Working with networks

3.2 Order of execution

In the Ladder Diagram program

Networks in a ladder diagram are executed one after the other in ascending order according to the network number. The order of execution can also be manipulated with jumps that direct to a certain destination.

In the network

The network is executed from left to right. Explicit signal feedback is prevented by the editor. Signal flow in the reverse direction is not possible.

Programming \ Programs \ Ladder Diagram (LD) \ Execution order

4 Ladder Diagram symbols

4.1 Contacts

Contacts with various functions are available. They can be added to the left side of the ladder diagram and connected to other contacts. They cannot be added to the far right, however, since this area is reserved for coils. Contacts are of data type BOOL and can be connected to digital inputs/outputs or function block parameters that have a matching data type.

The result of the logical connective of contacts within a network can be assigned to one or more coils. Each contact is represented by a variable name, which is defined in the variable declaration window.

The connection between contacts depends on the required control logic. They can be connected in series, parallel or a combination of the two in order to energize a coil.

Types of contact	Symbol
Normally open contact	─┤ ├─
Normally closed contact	─┤/├─
Positive edge	─┤P├─
Negative edge	─┤N├─
Both edges	─┤PN├─

Table 1: Overview of contacts

Programming \ Programs \ Ladder Diagram (LD) \ Contacts and coils

4.1.1 Difference between normally closed and normally open contacts

In the industrial environment, we are confronted with the terms "normally closed contact" and "normally open contact". Both terms belong in the category of contacts, inputs and outputs.

A normally closed contact conducts current as long as it is not being actuated.

A normally open contact conducts current only when it is being actuated.

If a normally closed contact is chosen, a doorbell will ring until someone presses the doorbell button. Pressing the button opens up the contact, which interrupts the flow of electricity. If using a normally open contact, the behavior is exactly the opposite.

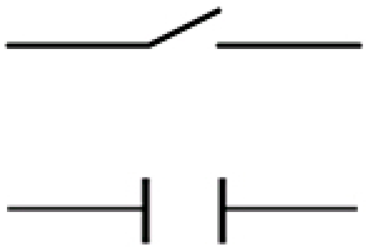


Figure 11: Normally open contact

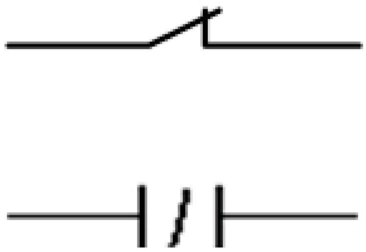


Figure 12: Normally closed contact

4.1.2 Normally open contact

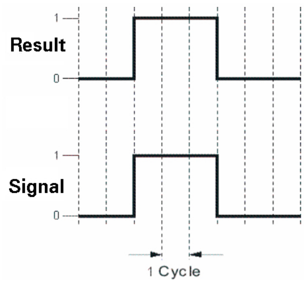


Figure 13: Relationship between the input signal and result

Figure 14: Normally open contact

As long as the contact is not being actuated, current doesn't flow and the logic state is FALSE.
When actuated, the physical state changes to "ON" and the result becomes TRUE.

4.1.3 Normally closed contact

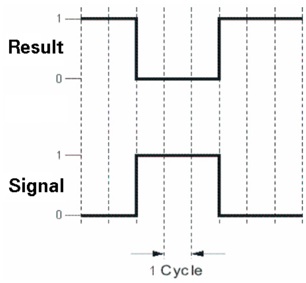


Figure 15: Relationship between the input signal and result

Figure 16: Normally closed contact

This contact inverts the status of a variable.
It is used when an input signal does NOT need to be present for the output to be set.
The state of the output is set to FALSE if the input is set to TRUE.

4.1.4 Contacts for edges

In programming, it is always helpful when rising and falling edges of signal levels can be evaluated.

Positive edges

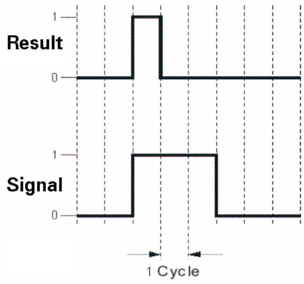


Figure 17: Relationship between the input signal and result

Figure 18: Positive edge

This contact is used to detect a positive edge of a signal.
When the value of a variable changes from FALSE to TRUE, i.e. a positive edge occurs, this contact returns TRUE for one cycle. It is used to set or reset conditions as well as to count the number of positive edges.

Negative edges

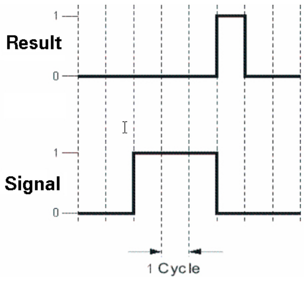


Figure 19: Relationship between the input signal and result

Figure 20: Negative edge

This contact is used to detect a negative edge of a signal.
If the value of a variable is switched from TRUE to FALSE, the result becomes TRUE for one cycle. This can be done, for example to set or reset outputs or to count the number of negative edges.

Both edges

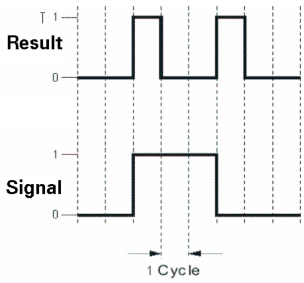


Figure 21: Relationship between the input signal and result

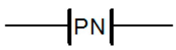


Figure 22: Positive and negative edge

This contact can be used to form a positive and negative edge of a digital signal.
This behavior corresponds to a parallel connection of the positive and negative edge.

4.2 Coils

Coils are basic elements of a ladder diagram. They are always placed on the right-hand side of the ladder diagram as an output. Coils can be connected to the right of contacts or to function block outputs. A network must have at least one coil. It is also possible to use several coils arranged in parallel.

Each coil can be used for digital outputs or internal variables that will be used later in the program as an input for another network. All contacts are constantly queried while the program is running. If a logical pathway is found, then the coil becomes TRUE.

Only Boolean variables can be assigned to coils.

Type of coil	Symbol
Coil	—( )—┤
Negated coil	—(/)—┤
Set coil	—(S)—┤
Reset coil	—(R)—┤
Positive transition coil	—(P)—┤
Negative transition coil	—(N)—┤
Both edges	—(PN)—┤

Table 2: Overview of coils

> ? Programming \ Programs \ Ladder Diagram (LD) \ Contacts and coils

4.2.1 Types of coils

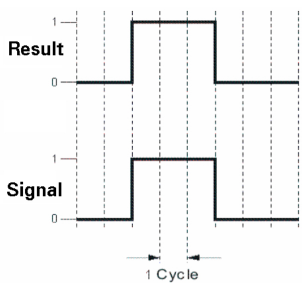


Figure 23: Relationship between the input signal and result

—( )—┤

Figure 24: Normally open coil

If a signal has the value TRUE, then the coil is switched on.

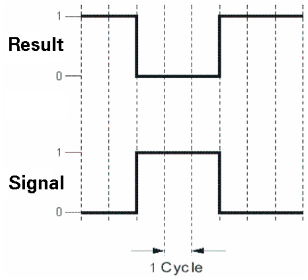


Figure 25: Relationship between the input signal and result

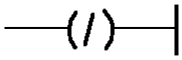


Figure 26: Normally closed coil

If a signal has the value TRUE, then the coil is switched off. At all other times, it is on.

4.2.2 Set and reset

Set coil

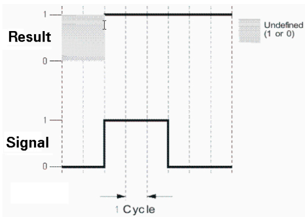


Figure 27: Relationship between the input signal and result

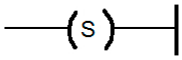


Figure 28: Set coil

This coil sets a variable to TRUE when a signal is present.
This state remains until the variable is reset. For this reason, this coil is conditional.

Reset coil

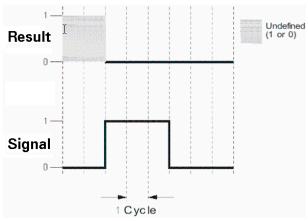


Figure 29: Relationship between the input signal and result

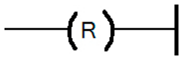


Figure 30: Reset coil

This coil sets a variable to FALSE when a signal is present with the value TRUE.

4.2.3 Edge outputs

Positive transition coil

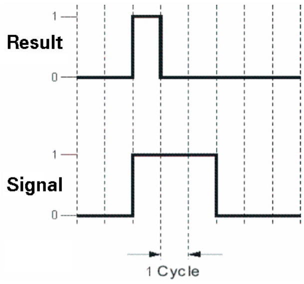


Figure 31: Relationship between the input signal and result

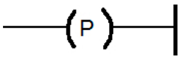


Figure 32: Positive transition coil

This coil sets a variable to TRUE for one cycle when a signal is present with the value TRUE. For all subsequent cycles with the same signal, the output stays FALSE.

Negative transition coil

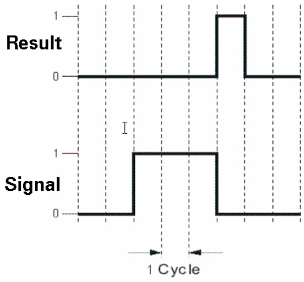


Figure 33: Relationship between the input signal and result

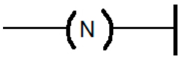


Figure 34: Negative transition coil

This coil sets a variable to TRUE for one cycle when a signal is present with the value FALSE. For all subsequent cycles with the same signal, the value of the variable stays FALSE.

Positive and negative transition coil

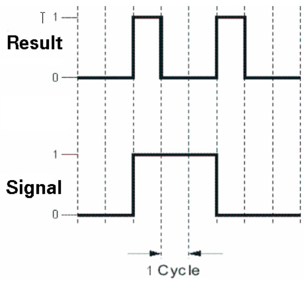


Figure 35: Relationship between the input signal and result

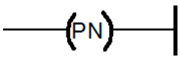


Figure 36: Output for positive and negative edge

This coil unites the function of the positive and negative edge output.

Exercise: Create your first ladder diagram

You will now create your first Ladder Diagram program. When a button is pressed, a lamp should light up until the button is released.

Variables	Data types	Description
diSwitch	BOOL	Input used for switching the light on/off
doLight	BOOL	Output used for energizing the light

Table 3: Overview of input and output variables

Exercise: Using positive and negative edges

Modify the previous example so that the lamp is turned on at a positive edge of the input and turned off at a negative edge of the input.

Exercise: Ladder programming using the keyboard

Create the following Ladder Diagram program using only the keyboard. First, find the keyboard shortcuts for inserting the Ladder Diagram symbols and creating the connection lines.

Then, an actuator should be switched using two signals. When **"bntSwitch1"** is present, the output is set and remains so until **"bntSwitch2"** arrives.

Variables	Data types	Description
bntSwitch1	BOOL	Input used to switch on the light
bntSwitch2	BOOL	Input used to switch off the light
doLight	BOOL	Actuator used for energizing the lamp

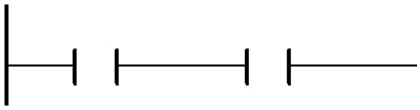
Table 4: Overview of input and output variables

5 Logic programming

Ladder Diagram is not only used for simple switching operations; it can also be used to implement binary logic.

5.1 Binary logic

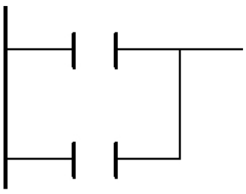
AND connective



If two or more contacts are switched in series, the result is a logical AND connective.
When all of the conditions have been met, the output is set to TRUE.

Figure 37: AND connective

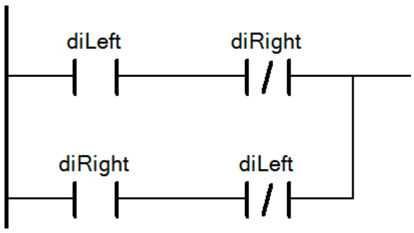
OR connective



A parallel block is equivalent to an OR connective.
If at least one of these parallel branches is TRUE, then the output is also TRUE.

Figure 38: OR connective

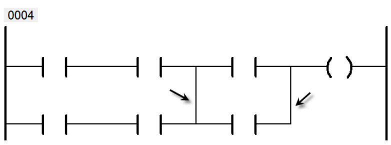
Exclusive OR operation



The Exclusive OR connective is a combination of the logical AND and OR connectives.
If either of the inputs is TRUE, then the output is TRUE. If both inputs are TRUE, then the output remains FALSE.

Figure 39: XOR connective

Branching and merging logic paths



The logic path can be modified through branching. This allows parallel paths to be taken. A branch needs to be merged again in order for the logic path to be closed.
Merging can also be branching from the next parallel path (see image).

Figure 40: Branching and merging

> A branch can be created in the editor using the arrow icons in the toolbar or by pressing **<ALT> + <↓>**.

Programming \ Programs \ Ladder Diagram (LD) \ Simple logic structures
Programming \ Editors \ Graphic editors \ Ladder Diagram editor \ Connection lines

Exercise: Programming a flip-flop

The following example combines some of the possibilities available in logic programming. In addition, the order of execution of this Ladder Diagram program is critical for the application to function correctly. Several solutions are possible.

Desired program behavior:

- When the user switches the input on, the output should be switched on.
- When the input is switched back off, the output should remain in the same state.
- The next time the input is switched on, the output should be switched off.

Variable name	Data type	Description
diSwitch	BOOL	Input that results in a change in status on the output at each positive edge
doFlipFlop	BOOL	Output controlled by the input

Table 5: Overview of input and output variables

6 Controlling program flow

6.1 Conditional jump

In addition to the network number, each network can also be given a unique jump label. A conditional jump can then be placed somewhere in the program sequence to any network with a jump label.

If the condition at the jump is TRUE, then the jump is executed.

Jumps are used to skip over networks in the program. This allows for greater control over program flow.

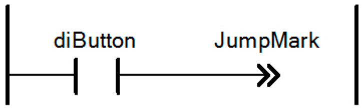


Figure 41: Conditional jump to the "JumpMark" network

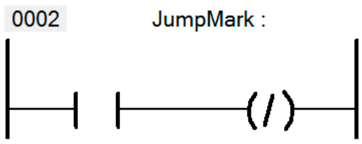


Figure 42: Network with the symbolic name "JumpMark"

If the jump label doesn't exist, then an error is output in the message window when the program is compiled.

`Error 1490: Label 'JumpMark' not defined.`

?

Programming \ Programs \ Ladder Diagram (LD) \ Jump / Jump return

6.2 Return

The Return command is used to interrupt the ladder diagram at a certain point. Any subsequent networks are no longer executed. In the next program cycle, the program executes from the first network until the return point (if active) or the end of the program.

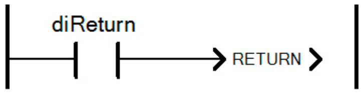


Figure 43: Program interruption with Return

?

Programming \ Programs \ Ladder Diagram (LD) \ Jump / Jump return

7 Functions, function blocks and actions

Using functions, function blocks and actions extends the capabilities of a programming language. Functions and function blocks contain program sections that are used more than once.

Functions

... have several parameters and only one return value. The result is always returned immediately after the function is called.

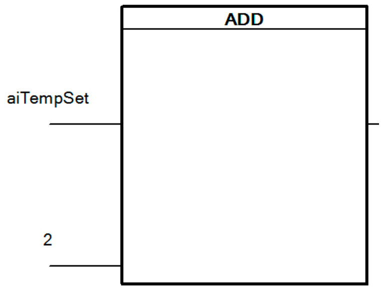


Figure 44: Function example

Function blocks

... usually have several return values and one instance variable. The instance variable is needed since function blocks can be spread out between tasks over a longer period of time, i.e. several cycles. In addition, the same function block will return different results when different input parameters are specified. The instance variable represents the "local memory" of the function block.

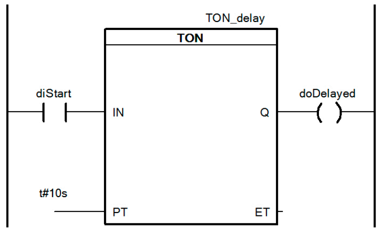


Figure 45: Function block example

Actions

... are subroutines or binary activities that can be called. Qualifiers specify the nature, timing and duration of the call. (7.3 "Using IEC actions")

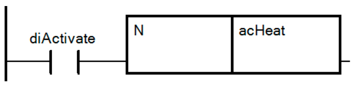


Figure 46: Calling an action with the "N" qualifier

Programming \ Programs \ Ladder Diagram (LD) \ Blocks

7.1 Working with function blocks

Functions and function blocks are managed in libraries. If a function block is inserted, its instance variable must be declared.

Inserting via editor's toolbar

Function blocks can be inserted into the program from the editor's toolbar.

Figure 47: "Insert function / function block" icon

Only libraries used in Automation Studio are part of the project. If a function or a function block from another library should be used, then the option "Show external libraries" needs to be enabled in the selection dialog box.

Inserting from Ladder Diagram Catalog

The list of available function blocks in the Ladder Diagram Catalog can be narrowed down using a full-text search or by setting filters. A function block can be added to the Ladder Diagram program using drag-and-drop. Then the instance variable must be declared.

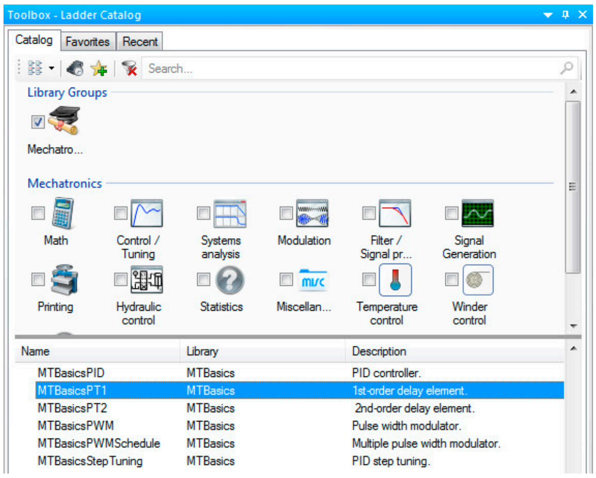


Figure 48: Ladder Diagram Catalog – Filter category: Mechatronic functions

Programming \ Programs \ Ladder Diagram (LD) \ Blocks

Programming \ Editors \ Graphic editors \ Ladder Diagram editor \ Functions

- Functions
- Ladder Diagram Catalog

Programming \ Editors \ General operations \ Dialog boxes for input support

7.1.1 Using analog values

For values that do not have data type BOOL, i.e. analog values, there are no ladder diagram symbols. These values are connected directly to the function or function block. They can be entered using the toolbar, the space bar, or by double-clicking on the contact.

Figure 49: Connecting an analog value using the toolbar

Bit addressing of analog values

If analog values should be associated with contacts and coils, then individual bits of analog values can be connected. To do so, a period "." is placed after the name of the analog value variable. Bits are numbered in ascending order starting with 0. For example, the second bit of an analog value can be accessed using `aiTemperature.1`.

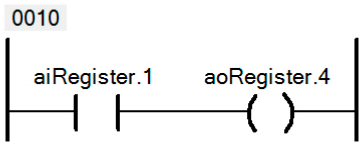


Figure 50: Assigning Bit 2 of "aiRegister" to Bit 5 of "aoRegister"

Programming \ Programs \ Ladder Diagram (LD) \ Analog value

Programming \ Editors \ Graphic editors \ Ladder Diagram editor \ Analog values

Programming \ Variables and data types \ Variables \ Bit addressing

7.1.2 Extensible functions

Some functions can be extended by the user. An additional input can be added in the function's properties, for example. The following functions can be expanded in Ladder Diagram:

ADD, AND, SUB, DIV, EQ, GE, GT, LE, LT, MAX, MIN, MOVE, MUL, MUX, OR, XOR

In addition to the above functions, the MOVE function can also be extended. For each extension, an input and an output are added. The assignments are executed in order by row as shown in the image.

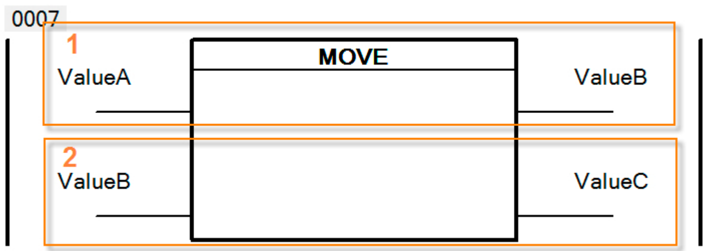


Figure 51: Assignments are executed in order, with ValueC = ValueA at the end of the program.

Programming \ Programs \ Ladder Diagram (LD) \ Blocks

Programming \ Editors \ Graphic editors \ Ladder Diagram editor \ Functions

Programming \ Libraries \ IEC 61131-3 functions \ OPERATOR

7.1.3 Blocks with EN / ENO

To simplify Ladder Diagram programming, function blocks can be enabled or disabled using a bit. This option is referred to as "EN / ENO" and can be turned on in the properties of each function block individually.

An EN signal with a value of TRUE enables the function block. Only the value of the EN signal is passed on to the ENO output. This allows functional blocks to be connected in series and enabled or disabled using a bit.

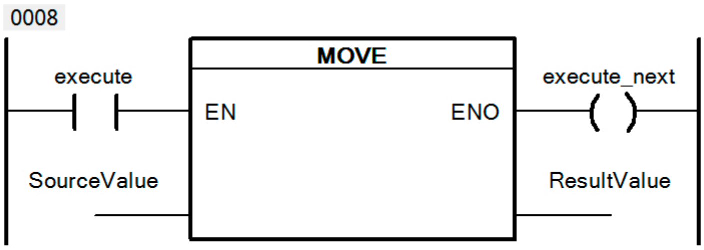


Figure 52: MOVE is executed only when "execute" is TRUE.

By default, the EN / ENO signal option is turned on for function blocks when they are inserted. It can be turned off using the **Tools / Options** menu item.

Programming \ Programs \ Ladder Diagram (LD) \ Blocks with EN/ENO
Project management \ The workspace \ AS settings \ Ladder Diagram editor settings

Exercise: Using function blocks

Several function blocks must be called in this exercise.

The state of an input is to be recorded several times. The collected data will then be used in a visualization applications. You can manipulate the visualization variables in the variable watch window.

Implement the following program behavior:

- The output is to be switched on 3 seconds after the input is enabled.
- After the input has been switched on three times, a warning appears on the visualization device.
- After acknowledging the warning on the visualization device, it disappears.
- After restarting the CPU, the count value for the number of times the input has been switched on should remain.

Variable	Data type	Description
diSwitch	BOOL	Monitored input
doMotor	BOOL	Time-delayed output
visButtonReset	BOOL	Acknowledgment button on the visualization device

Table 6: Overview of input and output variables

Variable	Data type	Description
visWarning	BOOL	Warning to be displayed on the visualization device

Table 6: Overview of input and output variables

Exercise: Configuring parameters using the visualization device

Expand the previous task to include the following functions:

- Number for the limit value until the warning is displayed should be configured.
- The default value for this limit value should be three (variable watch).
- The revised limit values should also remain after the CPU is restarted (RETAIN).
- Display how often the output has been switched on.

Variable	Data type	Description
visParamLimit	UINT	Input field for the limit value used for displaying the warning
visActivationCount	UINT	Output field for how often the output has been enabled

Table 7: Overview of the additional variables

7.1.4 Creating user functions

Program sections that are used frequently in Automation Studio can be stored in user functions and function blocks.

These functions and function blocks can be assigned directly to the Ladder Diagram program. This functionality can then be used several times in the program. In addition, it is also possible to create your own user library. It can be used throughout the project as often as necessary. In both cases, the implementation language may be different than the program doing the calling.

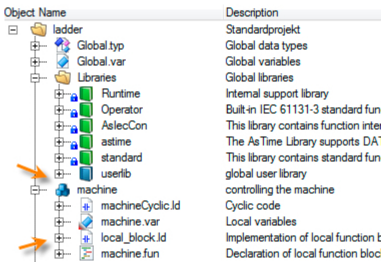


Figure 53: User function block in the program, user library in the project

Programming \ Libraries \ Example: Creating a user library

7.2 Compute and Compare blocks

Logic can be programmed with contacts and the functions available in the OPERATOR library. More complex calculations and comparisons usually require more than one function, however. This makes the networks more complex.

The Compute and Compare blocks can be used to enter expressions in a standardized format.

To handle even larger expressions, the Compute and Compare blocks can be extended just like the extensible functions (7.1.2 "Extensible functions").

Both blocks can access all local and global variables and constants. In addition, the expression can include any functions. Your input takes place in "Structured Text format".

7.2.1 Compute block

The Compute Block can be used to calculate an expression. The result is then passed to the block's output. It is possible to use all variables and constants as well as function calls.

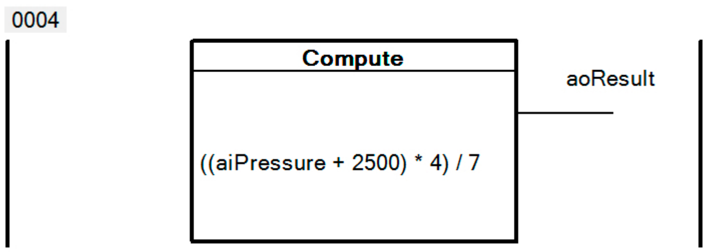


Figure 54: Calculating an expression with the Compute block

Programming \ Programs \ Ladder Diagram (LD) \ Compute

Exercise: Calculate the average value.

The temperature of a room is measured at three different places (**aiTemp1**, **aiTemp2**, **aiTemp3**). The average temperature (**aoTempAvg**) should be determined.

7.2.2 Compare block

The Compare block makes it possible to use logical comparison expressions. If the expression is TRUE, then the output of the Compare block is set until the expression becomes FALSE again.

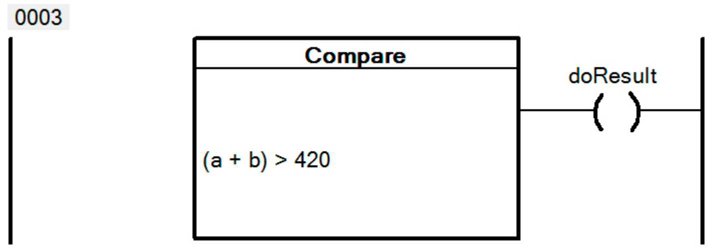


Figure 55: Using the Compare block

Programming \ Programs \ Ladder Diagram (LD) \ Compare

Exercise: Controlling room temperature

The set temperature and actual temperature in a room should be compared. If the actual temperature (**aiTempAct**) is less than the set temperature (**aiTempSet**), then the heater (**doHeat**) should be activated.

To prevent the heater from constantly switching on and off in the target range, it should continue to heat until the actual temperature is 2°C more than the set temperature.

1) Declare the variables.
2) Insert the Compare block.
3) Enter the comparison expression.
4) Test the application.

7.3 Using IEC actions

Actions can be used to implement subroutines and binary actions. The logic activates the action block and then calls the associated action with consideration of the qualifier. Qualifiers can be used to specify whether an action is delayed or limited in duration, for example.

Usage example

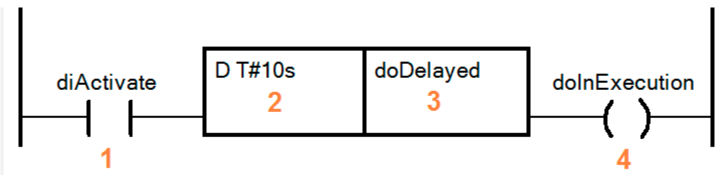


Figure 56: Delayed switching of "doDelayed"

1	"diActivate" calls the action block.
2	The "D T#10s" qualifier specifies that the action is delayed by 10 s.

Table 8: Description of the figure above

3	The binary action "doDelayed" is set with a time delay and remains set as long as the action block is being called.
4	"doInExecution" corresponds to the passed-on "diActivate" signal.

Table 8: Description of the figure above

Overview of important qualifiers

Some of the most important qualifiers are shown in the table below. A complete overview can be found in the Automation Studio help documentation.

Symbol	Description
D	Action delayed, time literal specification necessary
L	Action limited in time, time literal specification necessary
S	Action set and remains active until the R qualifier
R	Action reset
N	Action invoked as long as the action block is active

Table 9: Important qualifiers

Programming \ Programs \ Ladder Diagram (LD) \ Blocks

Programming \ Actions

Programming \ Actions \ Action block - Qualifiers

Exercise: Create a user function block

All of the basic functions and possibilities of Ladder Diagram programming have now been described. In this exercise, you will be creating a function block.

To do so, you will need to include the contents of the last exercise in the function block.

IN / OUT	Name	Data type	Description
IN	Switch	BOOL	Input for activating the motor
IN	vis ParamLimit	UINT	Configurable limit where a warning is output
IN	oldActivation-Count	USINT	Old counter for number of activations
IN	visButtonRe-set	BOOL	Acknowledges the warning from the visualization de-vice
OUT	visWarning	BOOL	Warning for the visualization device
OUT	visActivation-Count	UINT	Counter for the total number of activations
OUT	Motor	BOOL	Output for the motor

Table 10: Overview of function block inputs and outputs

8 Exercises

8.1 Exercise - Conveyor belt

Conveyor belt exercise

A conveyor belt should be driven using a Ladder Diagram program. The application itself has a manual and automatic mode. A detailed description of function as well as a variable list of inputs and outputs follows.

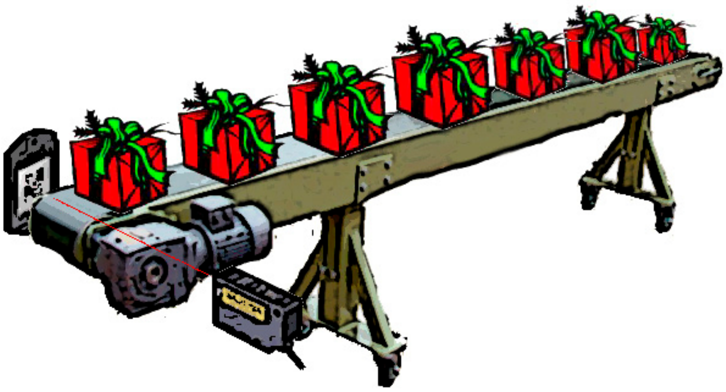


Figure 57: Conveyor belt

Exercise: Conveyor belt

The following functions are to be implemented for control purposes:

Manual mode:

- Automatic mode is inactive (**"diAutoMode"**).
- The conveyor belt runs as long as **"diManualStart"** is active.

Automatic mode:

- Start the conveyor belt if:
  - Automatic mode "**diAutoMode**" is active
  - Limit switch "**diConvEnd**" is not active
  - The end switch **"diConvEnd"** and material request **"diMachAskMat"** is active
- Stop the conveyor belt if:
  - Limit switch "**diConvEnd**" is active and material request "**diMachAskMat**" is inactive

Program structure:

- In manual mode, the networks that handle automatic operation should be skipped.

Batch counter:

- The CTU function block should be used to count the number of items moved on the conveyor.

Variable	Data type	Description
diAutoMode	BOOL	Switches between manual and automatic operation
diManualStart	BOOL	Starts manual movement of the conveyor belt in manual mode
diConvEnd	BOOL	Conveyor belt end switch
diMachAskMat	BOOL	Material request from the machine
doConvMotor	BOOL	Motor output that drives the conveyor belt

Table 11: Overview of inputs and outputs

8.2 Exercise - Concrete filling system

Exercise: Concrete filling system

In a concrete mixing system, concrete is loaded into the truck via a conveyor.

This filling operation is begun by pressing the On button **(btnOn)**.

However, the hydraulic system controlled by a solenoid valve **(doValve)** cannot be opened until the conveyor has been running for 5 seconds and a truck is located beneath the belt **(diTruck)**.

The solenoid valve is shut off as soon as the total permissible weight of the truck has been reached **(diPressure)**. The conveyor belt should continue to run for an additional 5 seconds, however.

The entire system is immediately shut down if the Off button **(btnOff)** is pressed.

If there is a disturbance in the conveyor system **(diConveyorMotorProtection)**, then the solenoid valve and the conveyor belt **(doConveyor)** should be shut off immediately. If there is a disturbance in the solenoid valve **(diValveProtection)**, then it is closed immediately, but the belt should continue running for an additional 5 seconds.

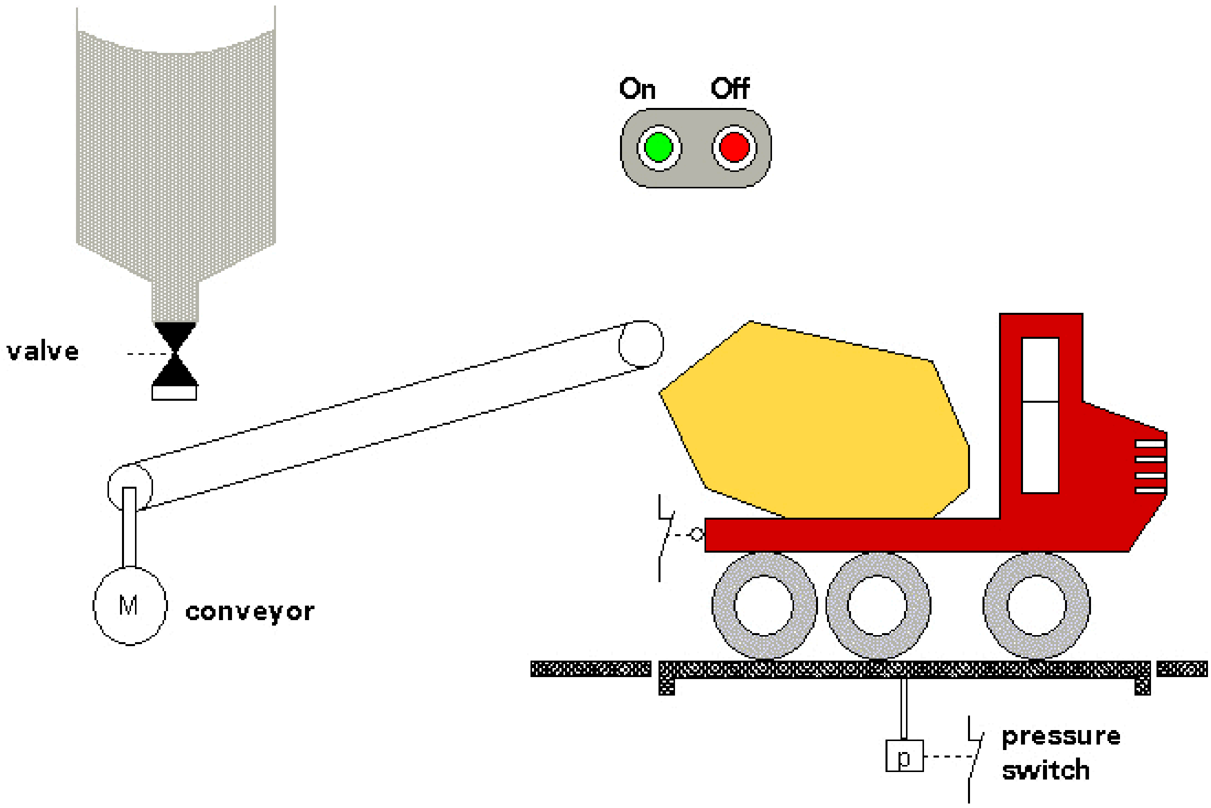


Figure 58: Schematic representation of the "Concrete filling system" exercise

9 Summary

Programming with Ladder Diagram is still very popular. It was developed to program logical switches as a replacement for hard-wired relay logic.

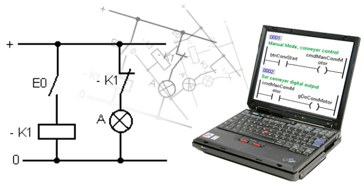


Figure 59: Ladder Diagram

Using analog signals and function blocks makes it possible to create high-powered applications using Ladder Diagram. Additional elements for controlling program flow extend the range of functions. In Automation Studio, program execution can be traced using Powerflow. Colors are used to display the status of lines that are conducting electricity.

10 Appendix

10.1 Keyboard shortcuts in the editor

Figure 60: Ladder Diagram editor toolbar

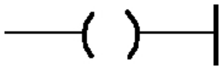
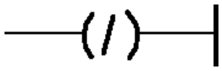
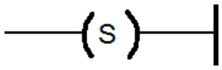
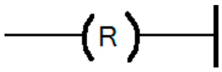
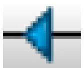
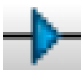
Symbol	Keyboard shortcut	Symbol	Keyboard shortcut
 Figure 61: Normally open contact	C	 Figure 62: Coil	Shift + C
 Figure 63: Normally closed contact	L	 Figure 64: Negated coil	Shift + L
 Figure 65: Positive edge	P	 Figure 66: Set coil	Shift + S
 Figure 67: Negative edge	N	 Figure 68: Reset coil	Shift + R
 Figure 69: Insert / Delete connection line to the left	ALT + ←	 Figure 70: Insert function block	F
 Figure 71: Insert / Delete connection line to the right	ALT + →	 Figure 72: Connect analog value (number, string, etc.)	Space bar

Table 12: Overview of keyboard shortcuts in the editor

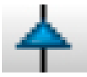
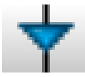
Symbol	Keyboard shortcut	Symbol	Keyboard shortcut
 Figure 73: Insert / Delete upwards connection line	ALT + ↑	 Figure 74: Address contact	A
 Figure 75: Insert / Delete downwards connection line	ALT + ↓	Complete and verify network	Enter
Insert new column, insert in-between contact	INS	 Figure 76: Add description	D

Table 12: Overview of keyboard shortcuts in the editor

How can I add a contact between existing ones?

To add contacts between existing ones, use the **<INS>** key to add a new column. Selecting a contact with a keyboard shortcut will then add it to the open position.

Is there an easy way to change the type of contact?

If a contact is highlighted or the cursor is placed directly in front of it, it can be changed using the keyboard shortcut for a different contact type.

How can I connect a (different) variable to a contact?

If a contact is selected or the cursor placed directly in front of it, you can press the **<space bar>** to activate the field for connecting the variable to the contact. Pressing the **<space bar>** again will open up the variable list where an existing variable can be selected.

Solutions

Creating your first ladder diagram

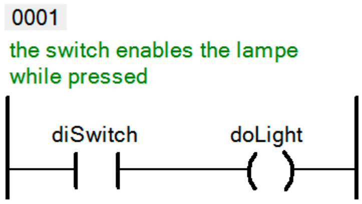


Figure 77: The output remains TRUE as long as the input is set.

Using the positive and negative edge

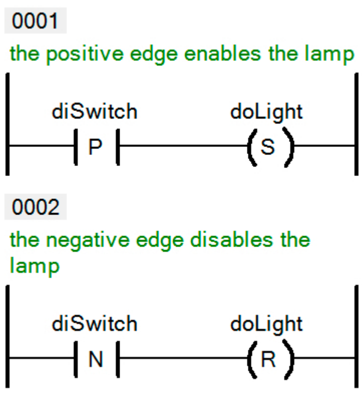


Figure 78: The positive edge is used to set; the negative edge is used to reset.

Ladder programming using the keyboard

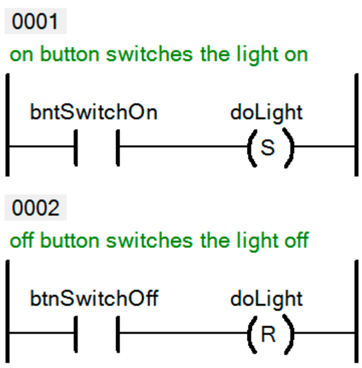


Figure 79: Separate on and off switches for the light

Programming a flip-flop

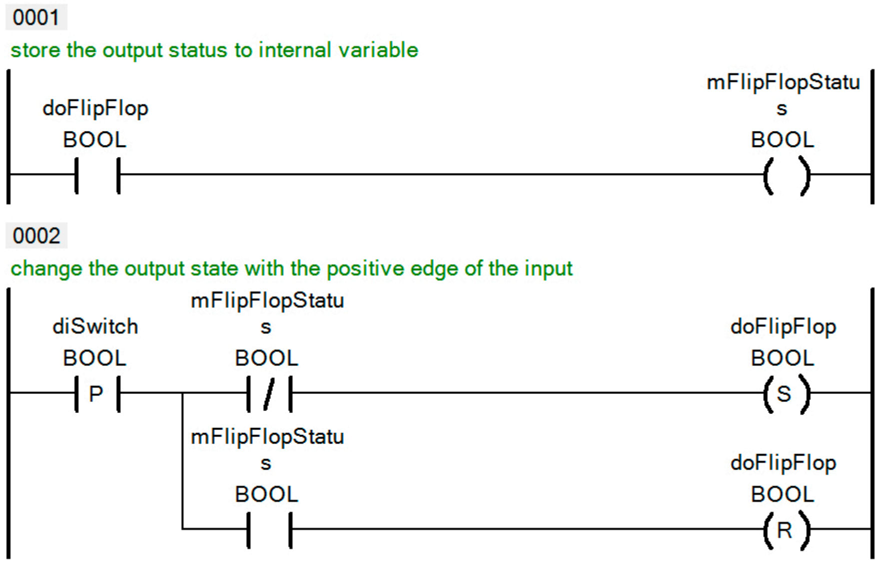


Figure 80: Network 1 stores the initial state, Network 2 changes the output state with the edge of the input

Using function blocks

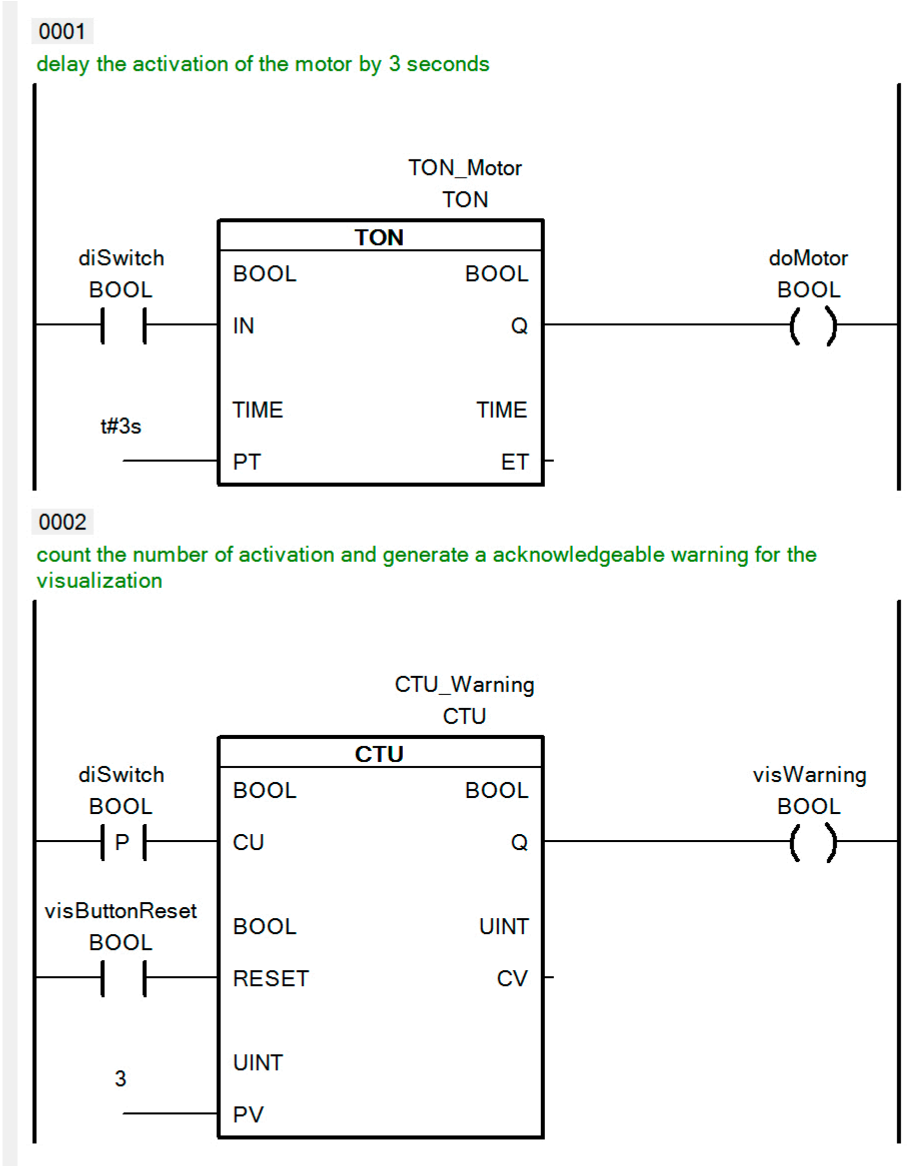


Figure 81: TON for turn-on delay, CTU with reference value PV to output warning

Configuring parameters using the visualization device

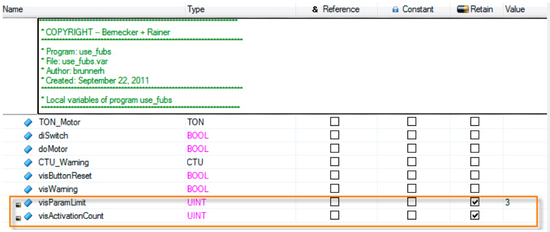


Figure 82: Pre-initialization of variables in the declaration window. With the RETAIN option these can be kept after restarting.

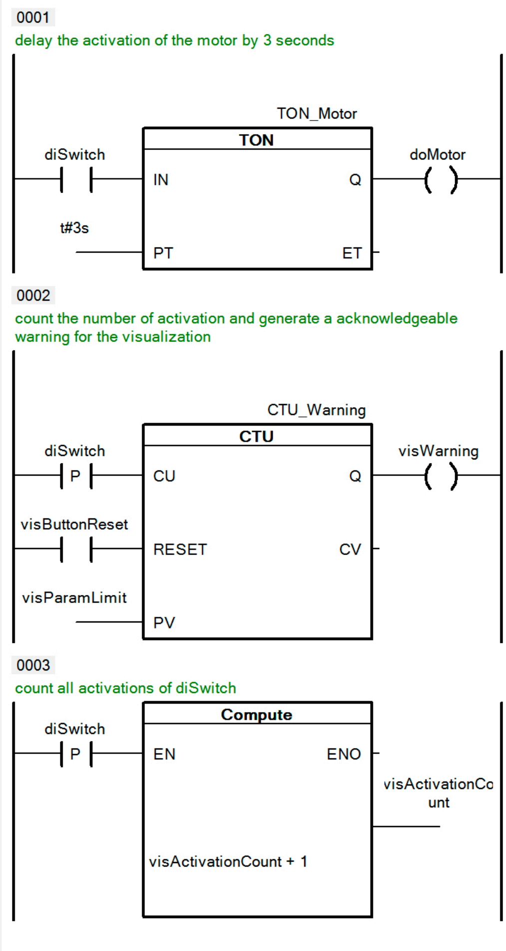


Figure 83: Configurable limit connected to CTU, total count calculated with Compute block

Creating a user function block

alarming			
Switch	BOOL	☐	VAR_INPUT
visParaLimit	UINT	☐	VAR_INPUT
visButtonReset	BOOL	☐	VAR_INPUT
oldActivationCount	UINT	☐	VAR_INPUT
visWarning	BOOL	☐	VAR_OUTPUT
visActivationCount	UINT	☐	VAR_OUTPUT
Motor	BOOL	☐	VAR_OUTPUT
TON_0	TON	☐	VAR
TON_Motor	TON	☐	VAR
CTU_Alarm	CTU	☐	VAR
zzInternalMemory	USINT[0..9]	☐	VAR

Figure 84: Declaration of the function block instance in the .fun file

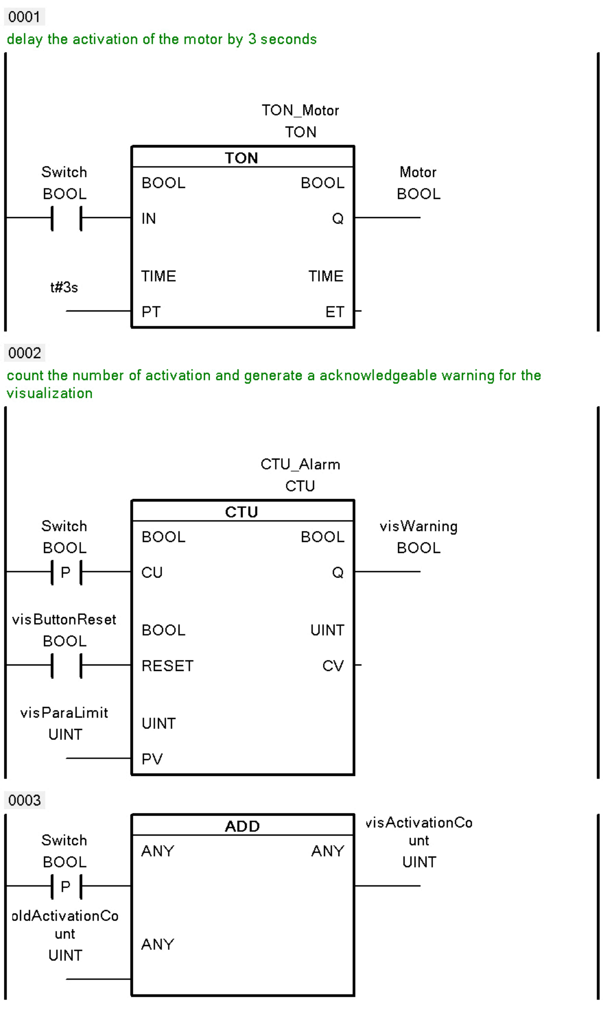


Figure 85: Possible solution for implementing the function block

Name	Type	Reference	Constant	Retain	Value
** * COPYRIGHT -- Bernecker + Rainer ** * Programm: use_userfub * Datei: use_userfub.var * Autor: tregguser * Erstellt: 22. September 2011 ** * Lokale Variablen des Programms use_userfub **					
alarming_0	alarming	☐	☐	☐	
diSwitch	BOOL	☐	☐	☐	
visLimit	UINT	☐	☐	☑	3
ActivationCounter	UINT	☐	☐	☑	
visReset	BOOL	☐	☐	☐	
Warning	BOOL	☐	☐	☐	
doMotor	BOOL	☐	☐	☐	

Figure 86: Variable declaration for the program calling the function block

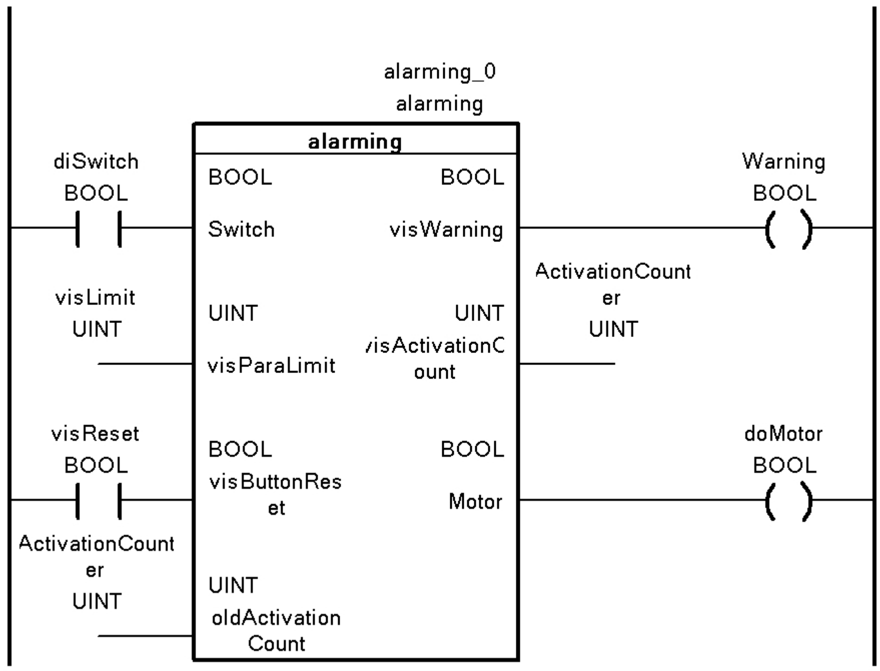


Figure 87: Calling the user function block

Compare block

With Compare block

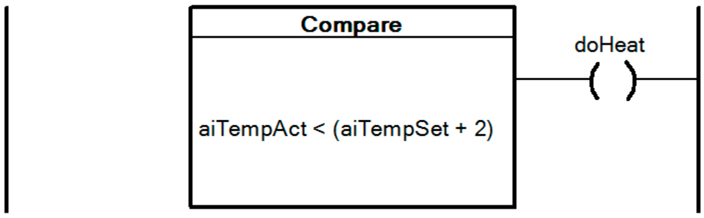


Figure 88: Using the Compare block

Without Compare block

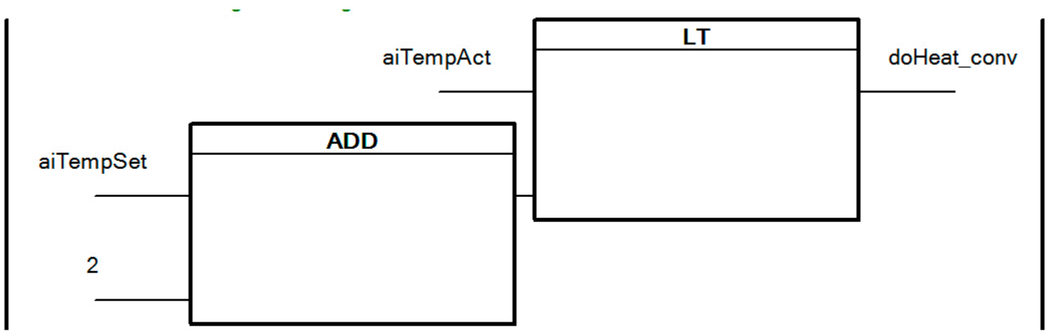


Figure 89: Same result, but without the Compare block

Compute block

With Compute block

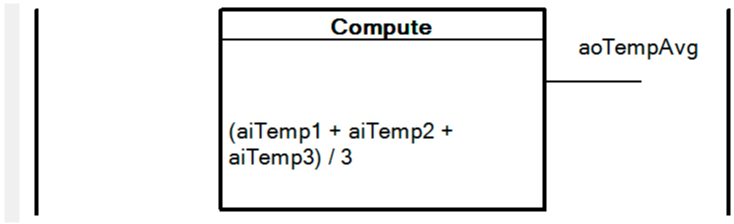


Figure 90: Solution with Compute block

Without Compute block

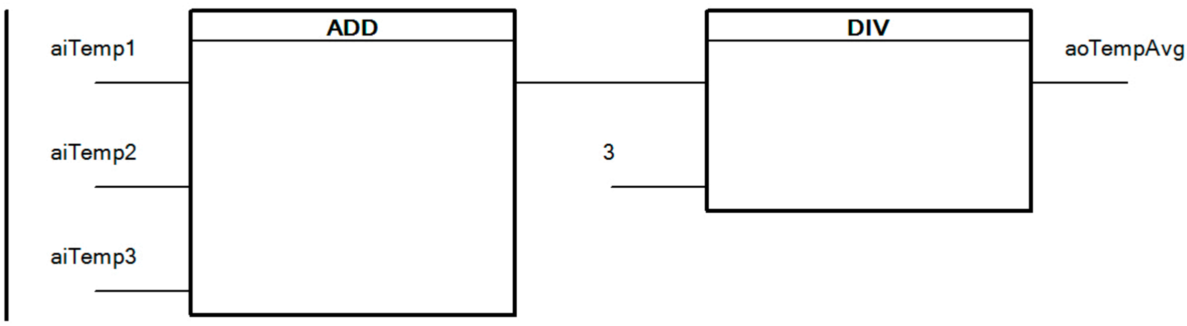


Figure 91: Solution without Compute block

Conveyor belt

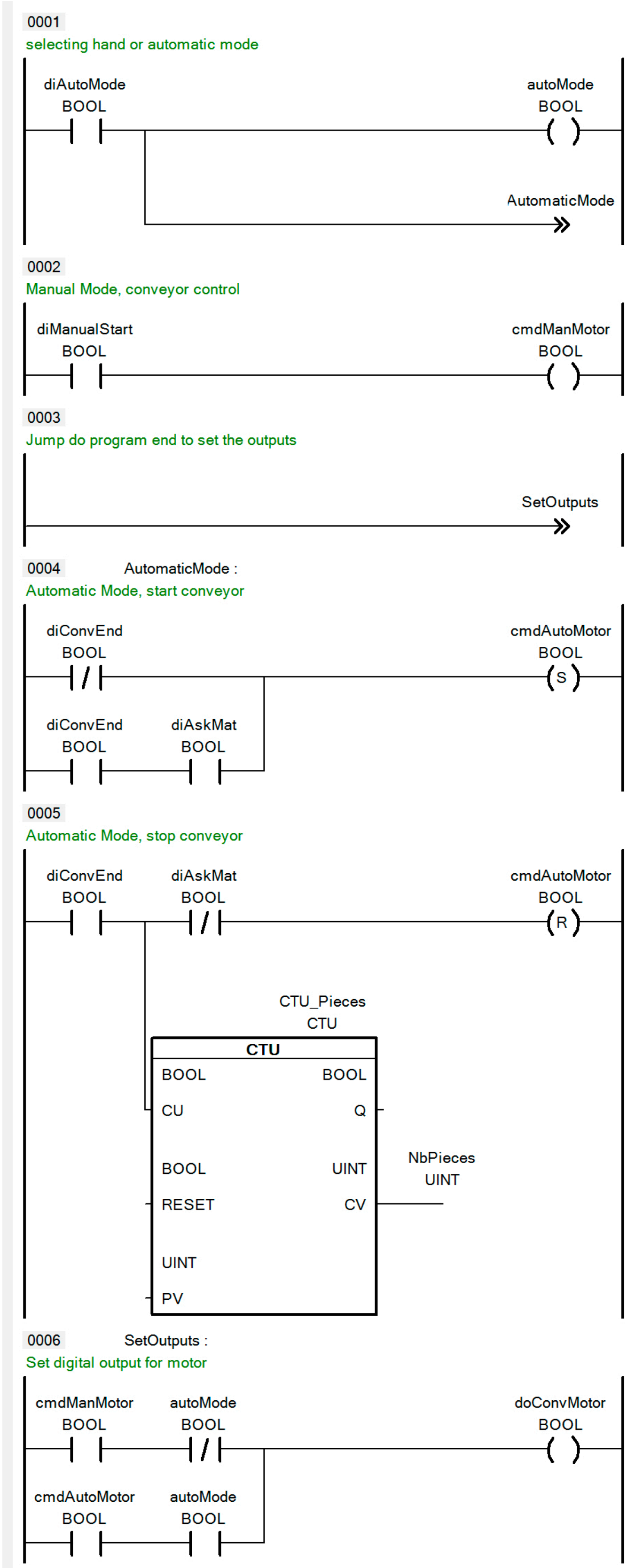


Figure 92: Possible solution for conveyor belt

Concrete filling system

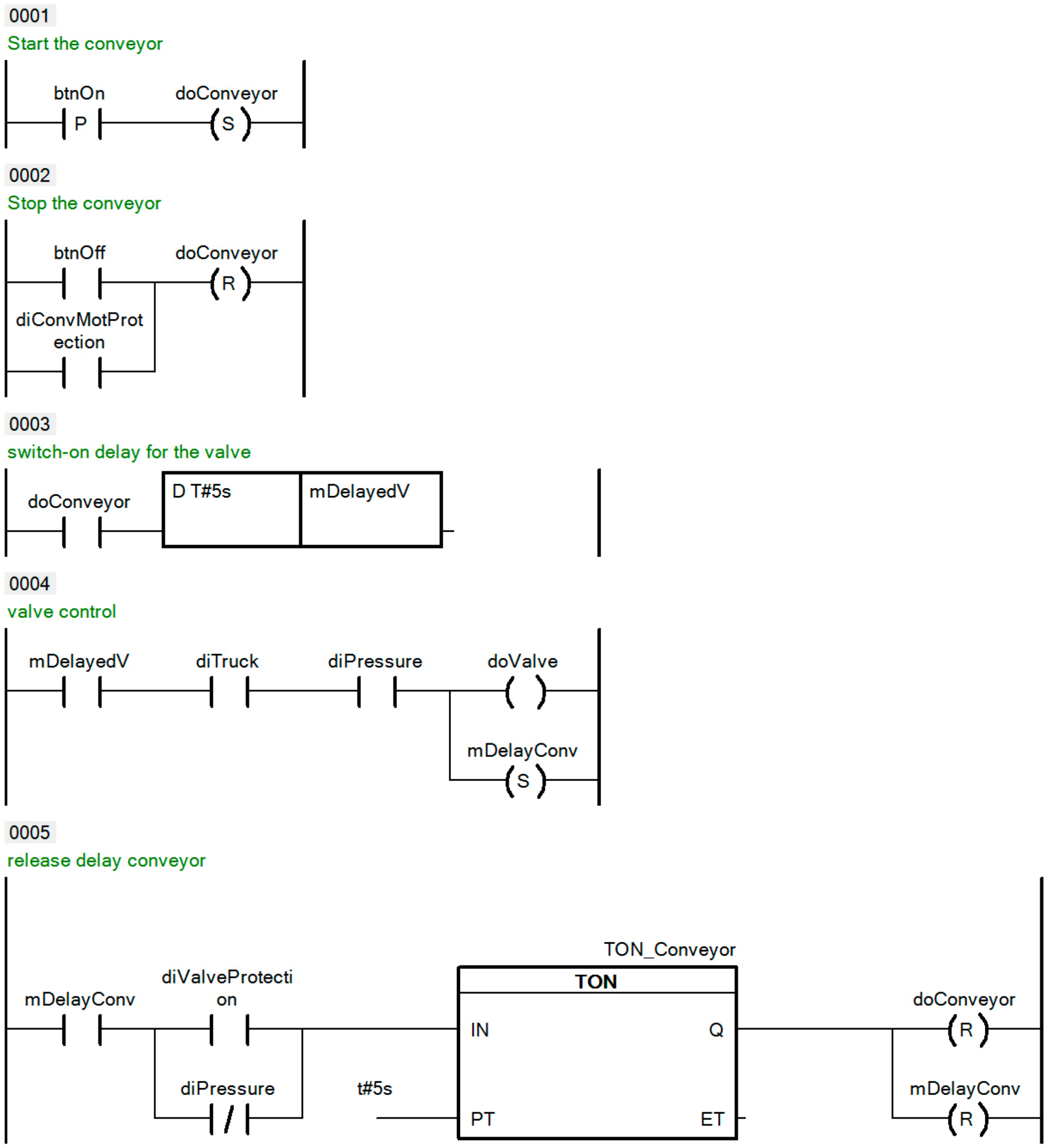


Figure 93: Possible solution for concrete mixer

Seminars and training modules

At the Automation Academy, you'll develop the skills you need in no time!
Our seminars make it possible for you to improve your knowledge in the field of automation engineering.

Automation Studio seminars and training modules

Programming and configuration

SEM210 – Basics
SEM246 – IEC 61131-3 programming language ST*
SEM250 – Memory management and data storage

SEM410 – Integrated motion control*
SEM441 – Motion control: Electronic gears and cams**
SEM480 – Hydraulics**
SEM1110 – Axis groups and path-controlled movements**

SEM510 – Integrated safety technology*
SEM540 – Safe motion control***

SEM610 – Integrated visualization*

Diagnostics and service

SEM920 – Diagnostics and service for end users
SEM920 – Diagnostics and service with Automation Studio
SEM950 – POWERLINK configuration and diagnostics*

If you don't happen to find a seminar on our website that suits your needs, keep in mind that we also offer customized seminars that we can set up in coordination with your sales representatives:
SEM099 – Individual training day

Please visit our website for more information****.****:
www.br-automation.com/academy

Overview of training modules

TM210 – Working with Automation Studio
TM213 – Automation Runtime
TM223 – Automation Studio Diagnostics
TM230 – Structured Software Development
TM240 – Ladder Diagram (LD)
TM241 – Function Block Diagram (FBD)
TM242 – Sequential Function Chart (SFC)
TM246 – Structured Text (ST)
TM250 – Memory Management and Data Storage

TM400 – Introduction to Motion Control
TM410 – Working with Integrated Motion Control
TM440 – Motion Control: Basic Functions
TM441 – Motion control: Electronic gears and cams
TM1110 – Integrated Motion Control (Axis Groups)
TM1111 – Integrated Motion Control (Path Controlled Movements)
TM450 – Motion Control Concept and Configuration
TM460 – Initial Commissioning of Motors

TM500 – Introduction to Integrated Safety
TM510 – Working with SafeDESIGNER
TM540 – Integrated Safe Motion Control

TM600 – Introduction to Visualization
TM610 – Working with Integrated Visualization
TM611 – Working with mapp View
TM630 – Visualization Programming Guide
TM640 – Alarm System, Trends and Diagnostics
TM670 – Advanced Visual Components

TM920 – Diagnostics and service
TM923 – Diagnostics and Service with Automation Studio
TM950 – POWERLINK Configuration and Diagnostics

TM280 – Condition Monitoring for Vibration Measurement
TM480 – The Basics of Hydraulics
TM481 – Valve-based Hydraulic Drives
TM482 – Hydraulic Servo Pump Drives
TM490 – Printing Machine Technology

In addition to the printed version, our training modules are also available on our website for download as electronic documents (login required):

Visit our website for more information:
www.br-automation.com/academy

Process control seminars and training modules

Process control standard seminars

SEM841 – Process control training: Basic 1
SEM842 – Process control training: Basic 2
SEM890 – Advanced Process Control Solutions

Process control training modules

TM800 – APROL System Concept
TM810 – APROL Setup, Configuration and Recovery
TM811 – APROL Runtime System
TM812 – APROL Operator Management
TM813 – APROL web portal
TM820 – APROL solutions
TM830 – APROL Project Engineering
TM835 – APROL ST-SFC Configuration
TM840 – APROL Parameter Management and Recipes
TM850 – APROL Controller Configuration and INA
TM860 – APROL Library Engineering
TM865 – APROL Library Guide Book
TM870 – APROL Python Programming
TM880 – APROL reporting
TM890 – The Basics of LINUX

* SEM210 - Basics is a prerequisite for this seminar.
** SEM410 - Integrated motion control is a prerequisite for this seminar.
*** SEM410 - Integrated motion control and SEM510 - Integrated safety technology are prerequisites for this seminar.
****Our seminars are listed in the Academy\Seminars area of the website.
****Seminar titles may vary by country. Not all seminars are available in every country.

TM240TRE.00-ENG