

MCR7 Series PLC

Programming Software MCR Master

User Manual

Version: 01

Publication Date: 2026-05-18

Revision History

Date	Version	Revision
2026-05-18	01	First release

Content

Preface.....	1
1 Product Overview	1
1.1 Product Profile.....	1
1.2 Product Technology Advantages	1
1.3 Applicable PLC Models.....	1
2 Quick Start	3
2.1 Install MCR Master.....	3
2.1.1 Pre-installation Preparation	3
2.2.2 Installation Procedure	4
2.2 Introduction to the Software Configuration Interface	4
2.3 Main Operating Procedures.....	5
3 Programming Concept.....	7
3.1 IEC 61131 standard and Implementation of MCR Master.....	7
3.1.1 Introduction to IEC61131-3 standard	7
3.1.2 Characteristics of IEC61131-3 standard.....	7
3.2 Programming languages that MCR7 Series PLCs Support.....	8
3.2.1 Ladder Diagram Programming Language	8
3.2.2 Structured Text Programming language	12
3.3 Variable.....	29
3.3.1 Overview	29
3.3.2 Variable Name	29
3.3.3 Data Type of Variable	30
3.3.4 Variable Property	35
3.3.5 Initialization of Variable	37
3.3.6 Mapping of Variable	37

3.3.7	Variable Declaration	38
3.4	Adjust Instruction Pin.....	39
3.5	Switch Instruction Trigger State.....	39
4	MCR Master Built-in Instruction	41
4.1	Basic element	41
4.1.1	Contact.....	41
4.1.2	Coil	42
4.1.3	SR Set-Rest Flip-Flop.....	44
4.1.4	RS Reset-set Flip Flop	45
4.1.5	R_TRIG Rising Edge Detection Trigger	46
4.1.6	F_TRIG Falling Edge Detection Trigger	47
4.1.7	PID Algorithm Control	49
4.2	Timer	53
4.2.1	TP Timer Pulse.....	53
4.2.2	TON Timer On-Delay	55
4.2.3	TOF Timer Off-Delay	57
4.3	Couter.....	59
4.3.1	CTU Cout-up Counter	59
4.3.2	CTD Cout-down Couter	62
4.3.3	CTUD Count up/down Counter.....	64
4.4	Type conversion	68
4.4.1	TRUNC Truncation	68
4.4.2	*_TO_* Data Type Conversion	69
4.5	Mathematical Function.....	70
4.5.1	ABS Absolute Value	70
4.5.2	SQRT Square Root	71

4.5.3	LN Natural Logarithm	73
4.5.4	LOG Logarithm with Base 10	74
4.5.5	EXP Exponential Instruction	75
4.5.6	SIN Sine.....	76
4.5.7	COS Cosine	77
4.5.8	TAN Tangent.....	79
4.5.9	ASIN Arc Sine	80
4.5.10	ACOS Arc Cosine.....	81
4.5.11	ATAN Arc Tangent	82
4.5.12	RANDOM_NUM Getting a Random Positive Integer.....	83
4.6	Arithmetic Function	84
4.6.1	ADD Addition.....	84
4.6.2	MUL Multiplication.....	86
4.6.3	SUB Subtraction	88
4.6.4	DIV Division	90
4.6.5	MOD Modulus.....	92
4.6.6	EXPT Exponent.....	94
4.6.7	MOVE Assignment.....	95
4.7	Left/Right Bitwise Shift	97
4.7.1	SHL Shift Left	97
4.7.2	SHR Shift Right.....	99
4.7.3	ROR Rotation on Right	101
4.7.4	ROL Rotation on Left.....	103
4.7.5	AND Bitwise AND.....	105
4.7.6	OR Bitwise OR.....	106
4.7.7	XOR Bitwise XOR	108

4.7.8	NOT Bitwise NOT.....	110
4.8	Selection.....	111
4.8.1	SEL Either or	112
4.8.2	MAX Maximum	114
4.8.3	MIN Minimum	116
4.8.4	LIMIT Limit Value	118
4.8.5	MUX Multiplexer	120
4.9	Comparison	122
4.9.1	GT Greater Than.....	123
4.9.2	GE Greater Than or Equal to	124
4.9.3	EQ Equal to	126
4.9.4	LT Less Than	127
4.9.5	LE Less Than or Equal to	129
4.9.6	NE Not Equal to	131
4.10	String Operation	132
4.10.1	LEN String Length	133
4.10.2	LEFT Extract String from the Left	134
4.10.3	RIGHT Extract String from the Right	135
4.10.4	MID Extract String from the Middle	137
4.10.5	CONCAT Concatenate string.....	138
4.10.6	INSERT Insert String.....	139
4.10.7	DELETE Delete String.....	141
4.10.8	REPLACE Replace String.....	142
4.10.9	FIND Find String.....	143
4.11	Time.....	145
4.11.1	*_ADD_* Data and Time Addition	145

4.11.2	*_SUB_** Date and Time Subtraction.....	146
4.11.3	*_MUL_** Date and Time Multiplication	148
4.11.4	*_DIV_** Date and Time Division.....	149
4.11.5	RTC_RD Read Real-time Clock.....	151
4.11.6	RTC_WR Write Real-time Clock	153
4.11.7	RTC_RD_SPLIT Separated RTC Reading	155
4.11.8	RTC_WR_SPLIT Separated RTC Writing	157
4.12	Communication.....	160
4.12.1	SEN_MSG Raw Serial Port Send Data	160
4.12.2	RECV_MSG Raw Serial Port Receive Data	163
4.12.3	ARRAY_ADDR_BOOL Get the Address of BOOL Array	167
4.12.4	ARRAY_ADDR_WORD Get the Address of WORD Array	169
4.12.5	MODBUS_RW_BIT Serial (Ethernet) Port Send/ Receive Bit Data	170
4.12.6	MODBUS_RW_WORD Serial(Ethernet) Port Send/Receive WORD Data	174
4.12.7	MODBUS_WRITE_STRING Write String Data in Modbus Communication	178
4.12.8	MODBUS_READ_STRING Read String Data in Modbus Communication.....	183
4.12.9	STRING_TO_BYTEARRAY Convert String to BYTE Array.....	187
4.12.10	BYTEARRAY_TO_STRING Convert BYTE Array to String.....	188
4.12.11	SET_SERIAL_PARAMETER Set Serial Port Parameter	190
4.12.12	SET_ETH_PARAMETER Set Ethernet Port Parameter	194
4.13	Motion Control.....	197
4.13.1	MC_Power.....	198
4.13.2	MC_Home	199
4.13.3	MC_Stop.....	203
4.13.4	MC_Halt	205
4.13.5	MC_Reset	208

4.13.6	MC_SetPosition.....	210
4.13.7	MC_Jog	211
4.13.8	MC_MoveAbsolute	214
4.13.9	MC_MoveRelative	220
4.13.10	MC_MoveVelocity.....	225
4.13.11	MC_ReadActualVelocity	229
4.13.12	MC_ReadActualPosition	230
4.13.13	MC_ReadStatus	231
4.13.14	MC_ReadAxisInfo.....	233
4.13.15	ENC_Counter.....	235
4.13.16	ENC_Preset.....	238
4.13.17	ENC_TouchProbe	242
4.13.18	PWM.....	246
5	File.....	248
5.1	Create a New Project.....	248
5.2	Open the Project	249
5.3	View the Start Page	250
5.4	Close the Project.....	250
5.5	Save the Project.....	250
5.6	Save the Project as.....	250
5.7	Exit the Program.....	251
6	View.....	252
6.1	Fullscreen	252
6.2	Restore Default Workspace Layout.....	252
6.3	Cross Reference.....	252
6.4	Problem	253

6.5	All Variables	253
6.6	Monitoring.....	254
6.7	Output.....	254
7	PLC Hardware Configuration	256
7.1	Configure the CPU Unit.....	256
7.1.1	Basic Information	256
7.1.2	RTC Configuration	257
7.1.3	Task Execution Time	258
7.1.4	Built-in Digital Input	259
7.1.5	Built-in Digital Output.....	260
7.1.6	Ethernet Configuration	261
7.1.7	Serial Communication Configuration.....	267
7.1.8	Configure Modbus Slave Network Address	270
7.2	Configure The Extension I /O Bus	274
7.2.1	Analog Input	274
7.2.2	Analog Output	275
7.2.3	Digital Input.....	276
7.2.4	Digital Output	277
7.2.5	Temperature Acquisition Module	277
7.3	Read The Hardware Configuration of PLC.....	280
7.4	Hardware address of PLC	281
8	Task	285
8.1	POU View	285
8.1.1	Program Organizational Unit.....	285
8.1.2	Global Variable	310
8.1.3	System Variable	315

8.1.4	Custom Data Type	316
8.1.5	Task Configuration	323
8.1.6	Trace	325
8.2	Machine View.....	328
9	PLC	329
9.1	Connect and Debug	329
9.1.1	Compile	329
9.1.2	Connect PLC.....	329
9.1.3	Disconnect PLC	330
9.1.4	Upload	330
9.1.5	Download Project to PLC	331
9.1.6	Start Running.....	332
9.1.7	Stop Running	332
9.1.8	Online Monitoring	333
9.1.9	Offline.....	334
9.1.10	View ST Code.....	335
9.2	Print Project Information	335
9.3	Project Comparison	336
9.4	Modify project property	338
9.5	Restart PLC	338
9.6	Restore Factory Settings	339
10	Setup	340
10.1	Switch System Interface Language	340
10.2	Display Setting	340
11	Upgrade PLC Firmware.....	341
12	Help.....	342

12.1	Logs.....	342
12.1.1	Set the Save Path of Logs	342
12.1.2	Set the Level of collected logs.....	342
12.2	PLC Logs.....	342
12.2.1	Collect PLC logs.....	343
12.2.2	Clear PLC logs.....	343
12.3	Disk Management	343
12.4	View the Help Document of MCR Master.....	344
12.5	Check for Updates.....	344
12.6	View Software Version Information.....	344
13	Motion axis.....	345
13.1	Overview	345
13.1.1	The Composition and Control Logic of the Motion Axis.....	345
13.1.2	Type of Motion Axis.....	345
13.2	Operating Mode.....	346
13.3	Buffering Mode	347
13.4	High-speed I /O Hardware	348
13.5	Create Motion Axis	352
13.5.1	Pulse Axis	352
13.5.2	PWM.....	360
13.5.3	Counting axis.....	361
14	Typical Configuration Cases.....	365
14.1	Modbus RTU Communication Between MX On CORPORATION PLC and Schneider ATV32 Frequency Inverter.....	365
14.1.1	Wiring method	365
14.1.2	Configure ATV32.....	366

14.1.3	Configure PLC.....	368
14.2	Modbus RTU Communication between MX On CORPORATION PLC and ABB ACS510 Frequency Inverter.....	371
14.2.1	Wiring method.....	372
14.2.2	Configure ACS510.....	372
14.2.3	Configure PLC.....	376
14.3	Common Programming Cases.....	379
14.3.1	Starting, Retaining, and Stopping Circuit.....	379
14.3.2	Call Counter.....	384
14.3.3	Call Timer	386
14.3.4	Configure Interrupt Tasks	388
15	Appendix A - Communication Code Description	394
16	Appendix B - Address Area Relationship Description.....	396
17	Appendix C - Motion Control Error Codes	397
18	Appendix D- Status Indicator.....	398
18.1	RUN Status Indicator	398
18.2	ERR Status Indicator.....	398
19	Appendix E - Abbreviations	400

Preface

Overview

This manual details the installation and configuration methods of MCR Master (hereinafter referred to as “MCR Master”) and guides users in visualization programming of the MCR7 Series PLCs.

The content provided in this manual only serves as general guidance and does not guarantee coverage of all usage scenarios for all product models. Due to reasons such as version upgrades, different device models and configuration files, the content provided in the manual may not match the actual device interface used by the user. Please refer to the actual information displayed on the user's device interface. The manual will not provide a detailed explanation of the differences caused by the aforementioned situations.

For the purpose of functional introduction and configuration examples, the manual may use IP addresses, URLs, domain names, etc. Unless otherwise specified, the aforementioned content is for illustration only and does not represent any actual significance.

Target Reader

This document is intended for readers who want to know how to program the MCR7 Series PLCs via MCR Master, including system administrators, PLC programmers, etc. This document assumes that readers have a certain level of knowledge in the following areas:

- ◆ Basic network communication protocols such as TCP/IP
- ◆ Modbus protocol
- ◆ How PLC works
- ◆ Basics of PLC programming

Format Convention

This manual follows the following content formatting conventions.

Content	Description
Bold	Bold represents the names and contents of various controls on the software interface. For example, “Select Window/Current Window Properties from the menu bar to enter the Modify Window page, and select the Timer tab.”
/	When describing the operation steps on the software interface, slash is used to isolate the clicked objects (menu item, sub-menu, button, etc.). For example, “Select Component/Switch/Bit Set from the menu bar, and create a new bit set switch component”.

Content	Description
<i>Italic</i>	Variables, must be replaced by actual values accordingly. For example, “Enter ‘ftp:// <i>the IP of HMI</i> ’ in the browser address bar, and press Enter to enter the file directory interface of the HMI.”

This manual follows the icon formatting conventions below.

Icon	Description
	Tips, operation tips for users to solve problems.
	Description, supplementary and explanatory information for the main text.
	Caution, reminders for operation precautions, improper operation may cause potential device damage or data loss.
	Warning, the content following this icon requires special attention, otherwise it may result in personal injury.

1 Product Overview

1.1 Product Profile

MCR Master provides a configuration and programming environment for MCR7 series PLCs, providing users with a platform to develop, edit and monitor control applications. The software strictly comply with international standard IEC-61131-3 and supports both Ladder Diagram and Structured Text (ST) programming languages. Users can program with standard Ladder Diagram, Structured Text language, or a mix of both, as required.

1.2 Product Technology Advantages

- ◆ supports Ladder Diagram and Structured Text programming languages: Program MCR7 Series PLCs using MCR Master, support Ladder Diagram and Structured Text programming language, Ladder Diagram for logic control, Structured Text for complex computing, easy to deal with a variety of scenarios.
- ◆ Data model: Modeling for equipment, create modular project applications based on models, and high degree of project reuse.
- ◆ Graphical configuration interface, WYSIWYG: MCR Master provides graphical configuration interface, allows a clear view of hardware configuration , one-click acquisition of hardware architecture; project setup, variable mapping is convenient and quick, abandoning the traditional address concept, create project fastly and efficiently.

1.3 Applicable PLC Models

For PLC Models which is applicable to MCR Master, please refer to the table below.

Series	Model	Output type
MCR7 (Main Module)	MCR7-DN(P)16	Transistor output
	MCR7-DR14	Relay output
	MCR7-DN(P)16 01	Transistor output
	MCR7-DR14 01	Relay output
MCR7 (Extension Module)	MCR-DI16	Digital input
	MCR-DO16-N(P)	Transistor output
	MCR-DO06-R	Relay output
	MCR-DX16-N(P)	Digital input + Transistor output

Series	Model	Output type
	MCR-AI04	Analog input
	MCR-AO04	Analog output
	MCR-AX04	Analog input + output
	MCR-TI04-K(P)	Temperature input
	MCR-ECAT	Ethercat communication

2 Quick Start

2.1 Install MCR Master

2.1.1 Pre-installation Preparation

2.1.1.1 Get Installation File

MCR Master software installation package is available on MX On CORPORATION's official website.

2.1.1.2 Provide Installation Environment

Installation of MCR Master requires the necessary installation environment, including the hardware installation environment and the software installation environment.

2.1.1.2.1 Hardware Installation Environment

To ensure that MCR Master runs smoothly, it is recommended that the PC in which MCR Master is installed meets the following minimum hardware configuration requirements.

Hardware	Minimum Requirement
CPU	Dual core, clock speed 2.4GHz
Memory	8GB
Installation Disk	50GB
Network Interface	One gigabit Ethernet interface
USB Interface	One USB interface

2.1.1.2.2 Software Installation Environment

It is necessary to ensure that the operating system of the PC on which FStudio will be installed is one of the following:

Windows 7 SP1 and above.



If you run MCR Master in the PC whose operating system is Windows7 SP1, and cannot create a project or start page display blank, try downloading and installing the system patch:

2.2.2 Installation Procedure

Step1. Double-click the installation file *MCR Master Installer.exe* (software version should be consistent with actual situation).

Step2. Enter the MCR Master Installation Wizard interface and select the system interface language.

Step3. Click **Customize**.

Click **Install**, to install MCR Master using the default path (C :\Program Files\MCR Master\).

Step4. Click **Browse**, set installation directory, click **Install**.

Step5. Click **I Accept**.

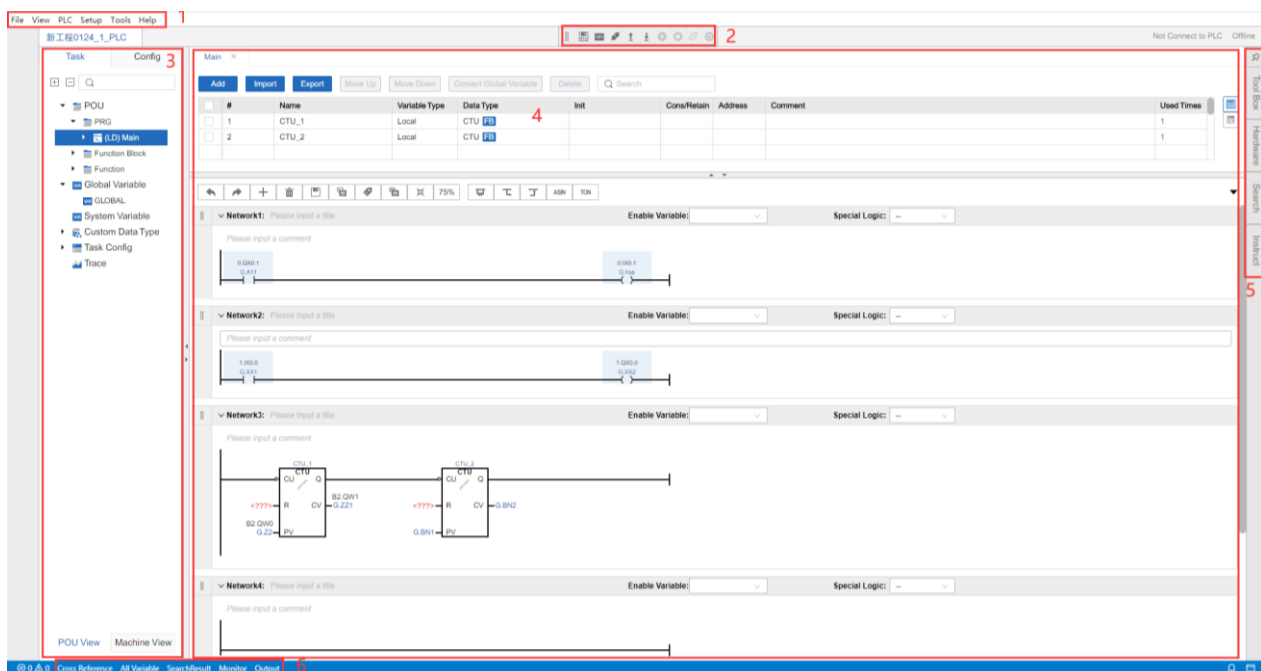
Step6. Wait for the program to install automatically.

Step7. After installation is complete, click **Finish**.

2.2 Introduction to the Software Configuration Interface

System Provides easy and convenient configuration interface, divided into program and hardware configurations by configuration type.

MCR Master provides easy and convenient program configuration interface for PLC project, as shown in the following figure.

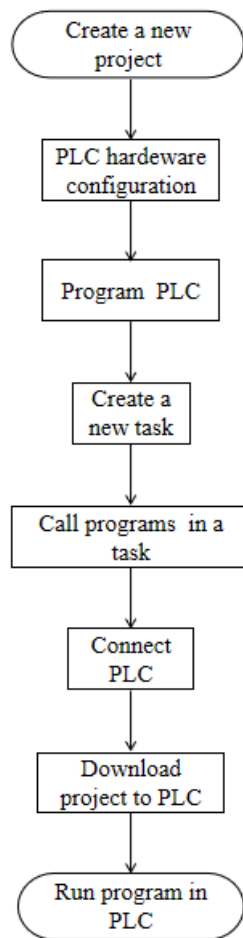


Please refer to the following table for detailed description of each area.

Number	Area	Description
1	Menu bar	Provides configuration entry for each feature, making it convenient for users to switch according to their actual needs.
2	Program debugging toolbar	Provides common debugging functions, including save, compile, connect PLC, upload, download, etc.
3	Task/Config	◆ Task: Show program task information. ◆ Config: Show hardware configuration of PLC.
4	Operating area	◆ Editing Variables, Programs in Task view. ◆ Configure hardware of PLC in Config view.
5	auxiliary function	Includes Toolbox, Hardware, Search, and Instruction
6	Information output	Project information output, including Cross References, All Variables, Search Results, Monitor, Output.

2.3 Main Operating Procedures

The main operating procedures of MCR Master are shown in the following figure.



Step1. [Create a New Project](#).

Step2. [PLC Hardware Configuration](#).

Step3. Program the PLC, for detailed information, please refer to [Program Organizational Unit](#).

Step4. [Create a New Task](#).

Step5. [Call Programs in a Task](#).

Step6. [Connect PLC](#).

Step7. [Download Project to PLC](#).

Step8. Run program in the PLC, for detailed information, please refer to [Start Running](#).

3 Programming Concept

3.1 IEC 61131 standard and Implementation of MCR Master

3.1.1 Introduction to IEC61131-3 standard

IEC 61131-3 is part 3 of the IEC 61131 standard developed by the International Electrotechnical Commission (IEC) in December 1993 to standardize programming systems for programmable logic controller (PLC), DCS, IPC, CNC and SCADA, and the application of IEC 61131-3 has become a trend in the field of industrial control. IEC 61131-3 specifies the syntax and semantics of the programmable controller language, defines five programming languages, and defines the basic software model through formal definitions, syntax and (partial) semantic descriptions, and examples. In the case of PLC, the editing software only needs to comply with the IEC 61131-3 international standard specification, so that the language structure in compliance with each standard can be used to create programs that can be understood by anyone.

3.1.2 Characteristics of IEC61131-3 standard

◆ Diversity

There are five different programming languages, which are divided into two categories: graphical programming language and text-based programming language. Especially when dealing with large projects, users can combine and integrate multiple programming languages in a project according to the actual needs, so as to realize the optimization of programming and provide a good operating environment for the application of programmable controllers.

◆ Compatibility

As the international programming language standards are adopted, it applies to programmable controllers, distributed control system, Fieldbus Control System, data acquisition and visual systems, motion control systems, etc. And the software model applies to industrial applications in different industries and structures. As a result, the standard programming language embodies the important feature of being more independent of the hardware used and less dependence on the specified hardware vendor for the user.

◆ Openness

The standardization of programming languages brings another benefit of making systems open. Any product produced by a supplier that conforms to a standard programming language can be programmed using the standard programming language, essentially cut off the dependency between the software and the specific hardware. However, at present, it is difficult to make the software completely unmodified when applies to different hardware. e.g., the I/O addresses of the external terminals of different programmable controllers may

be different, and the corresponding address definitions need to be re-entered during porting.

◆ Readability

A large number of expressions in the PLC programming language are similar to those in common computer programming languages. For example, select statements such as IF and CASE, and loop statements such as FOR, WHILE, and REPEAT, are similar to computer programming languages, which greatly facilitates the user's understanding of standard language usage and improves the readability of programs.

◆ Ease of operation and Security

In general, ease of operation and security are contradictory, but in this standard, they have been combined organically. The standard programming language is a continuation, improvement and extension of common computer programming languages, so it retains the advantages of these programming languages and overcomes the disadvantages, making it easier to operate; at the same time, because these languages are standard, the probability of error has been controlled to a minimum, making the programming language more secure.

3.2 Programming languages that MCR7 Series PLCs Support

MCR7 Series PLCs support Ladder Diagram and Structured Text Programming languages.

3.2.1 Ladder Diagram Programming Language

3.2.1.1 Ladder Diagram Overview

Ladder diagram originated in the USA, it is based on a graphic representation of relay logic and is a widely used graphical language in PLC programming. The ladder diagram program has two vertical power rails (namely bus-bar) on the left and right side.

The power rail on the left side nominally supplies the energy flow from left to right along a horizontal staircase through the individual contacts, functions, function blocks, coils etc. The energy flow ends at the power rail on the right side.

Each contact represents the state of a Boolean variable and each coil represents the state of an actual device. If the contact is “TRUE”, the coil of the corresponding soft relay in the ladder diagram is “energised”, its normally open contact is on and its normally closed contact is off, this state is called the “TRUE” or “ON” state of the soft relay. If the contact is “FALSE” and the state of the coil and contact of the corresponding soft relay is the opposite of the above, then the state of soft relay is called “FALSE” or “OFF”.

The common concept of Ladder Diagram are as follows:

◆ Bus-bar

Ladder Diagram use a network structure, a network of ladder diagrams is bounded by the left bus-bar. When analyzing the logic of a ladder diagram, in order to borrow the analysis method of a relay circuit diagram, it is possible to imagine that there is a left positive and right negative DC supply voltage between the left bus-bar and right bus-bar, and that there is an “energy flow” from left to right between the bus-bars.

◆ Section

Section is the smallest unit of the ladder diagram network structure and is called a program segment in MCR Master. The relevant logic network starting from the input conditions to a coil is called a section. In the configuration interface, the sections are arranged vertically and run in top-down order.

◆ Energy flow

Energy flow can be considered as a hypothetical “current” flowing from left to right, a direction that is consistent to the order of the logical operations performed by the user program. The energy flow can only flow from left to right. The concept of energy flow can be used to help us understand and analyze ladder diagrams better.

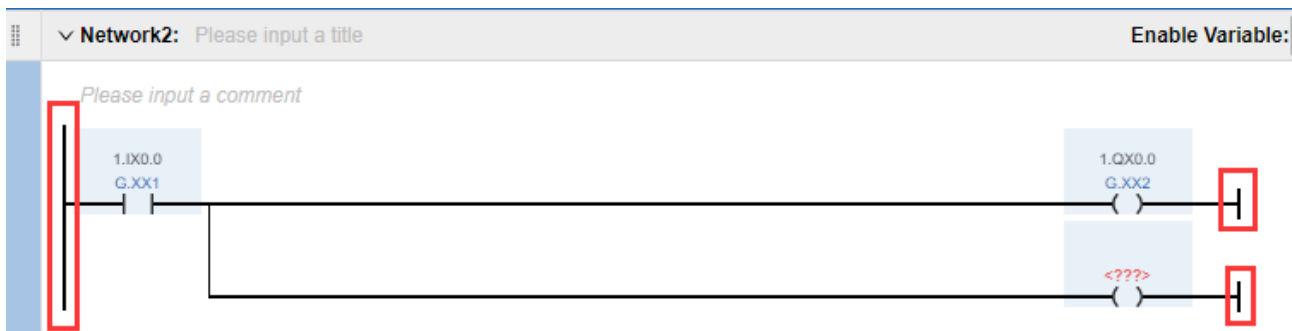
◆ Branch

When there are branches in the Ladder Diagram, the state of each graphic element is also analyzed according to the order of execution from top to bottom and left to right. For vertically connected elements, the state of the connected element on the right side is determined in accordance with the above-mentioned rules, and the process of evaluation from left to right and from top to bottom is carried out one by one.

3.2.1.2 Element in Ladder Diagram

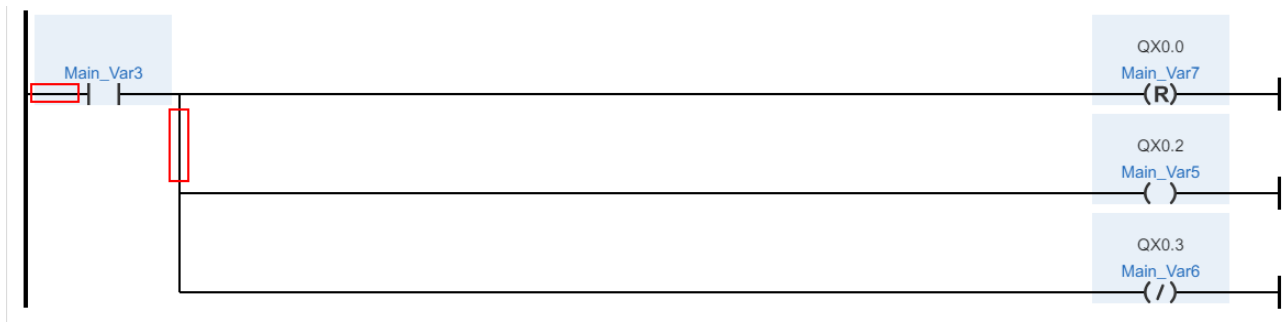
3.2.1.2.1 Bus-bar

The graphic element of the Power Rail of the ladder diagram is also known as the bus-bar. The bus-bar located on the left side of the ladder diagram is left bus-bar, the bus-bar located on the right side of Ladder Diagram is right bus-bar.



3.2.1.2.2 Connection line

In a Ladder Diagram, the graphic elements are connected by connection lines, which are divided into horizontal and vertical connection lines, as shown in the following Figure, in the order of horizontal and vertical connection lines.



3.2.1.2.3 Contact

For more information about contact, please refer to [Contact](#).

3.2.1.2.4 Coil

For more information about coil, please refer to [Coil](#).

3.2.1.2.5 Function Block and Function

Function blocks and functions can be called in Ladder Diagram. Includes MCR Master built-in Function Blocks (such as timers, counters, etc.), built-in Functions (such as mathematical functions), and user-defined Function Blocks, Functions. For more information about built-in Function Blocks and built-in Functions, please refer to [MCR Master Built-in Instruction](#).

As with contact and coil, users can also insert Function blocks and Functions in the program. In a network, they must have a Boolean type input variable and an Boolean type output variable, connected like contacts, on the left side of the Ladder Diagram.

Ladder Diagram and Structured Text language support calling Function and Function Block, when calling Functions and Function Blocks, need to note the following:

- ◆ Function and Function Block are represented by a rectangular box. The Function can have more than one input parameter, but only one return parameter. Function Block can have multiple input parameters and multiple output parameters.
- ◆ The input parameter is listed on the left side of the rectangular box and the output parameter on the right side of the rectangular box. The input parameter and output parameter can bind variable or constant.
- ◆ The name of the Function and Function Block appears in the top center of the box, the Function Block needs to instantiate it, and the instance name is in the top center outside the box. Use the instance name of the Function Block as its unique identity in the project.
- ◆ To ensure that the energy flow can pass through a Function or Function Block, each called Function or Function Block should have at least one input and one output parameter. In order for the connected function block or function to be executed, at least one Boolean type input pin should be connected to the left bus-bar by a horizontal cascade.
- ◆ When calling a Function Block, you can fill in the actual parameter value directly at the outer connector of the Function Block with the name of the internal formal parameter.
- ◆ Without dedicated input/output parameters(EN and ENO) , Functions and Function Blocks are executed automatically and the state is passed downstream.

3.2.1.3 Energy Flow State Transfer Rules for Connection Elements

The state of the connecting element is passed from left to right, the flow of energy flows is achieved, and the transmission of the state follows the following rules:

- ◆ The connecting element connected to the left power rail , whose state is TRUE at any moment, indicates that the left power rail is the starting point of the energy flow. The right power rail is similar to the zero potential in the electrical diagram.
- ◆ A horizontal connecting element is represented by a horizontal connection line, and the horizontal connecting element passes the state of the graphic element to the graphic element to its right, starting with the graphic element on its left.
- ◆ A vertical connecting element is always connected to one or more horizontal connecting elements, that is, it consists of one or more horizontal connecting elements intersecting a vertical line on one side (left or right). The state of the vertical connection element is determined by an “or” operation in accordance with the state of each left-side horizontal connection element to which it is connected.

The state of vertical connecting element is therefore determined according to the following rules:

- ◆ If the state of all connected horizontal connecting elements on the left is FALSE, the state of a vertical connecting element is FALSE.
- ◆ If one or more horizontal connecting elements on the left are TRUE, the vertical connecting element is TRUE.
- ◆ The state of the vertical connecting element is passed to all horizontal connecting elements connected to its right, but not to all horizontal connecting elements connected to its left.

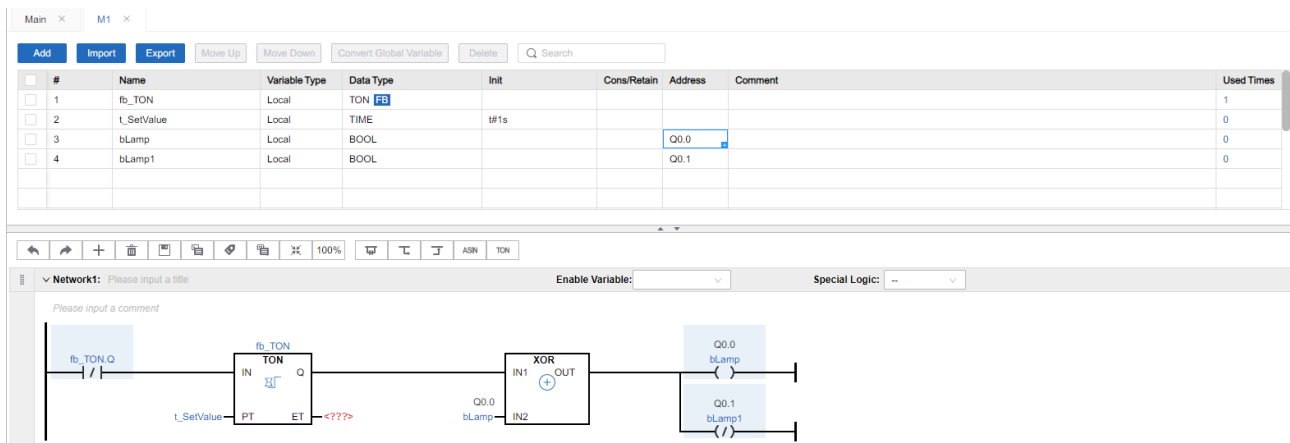
3.2.1.4 Programming Examples

3.2.1.4.1 Control Requirement

Use timers and logical functions to form a signal flashing light system. The line output enables the light to turn on (ON) and off (OFF) at the specified period.

3.2.1.4.2 Programming

The program implements the signal flashing light control requirements by alternately turning bLamp and bLamp1 on (ON) and off (OFF). The user can set the time for on (ON) and OFF (OFF) switches by t_SetValue, which is set to 1s in the program. The detailed design method of the program is shown in the figure below.



When this program is running, the outputs of bLamp and bLamp1 are exactly opposite to each other, and the state switching time is exactly 2s.

3.2.2 Structured Text Programming language

3.2.2.1 Introduction to Structured Text

Structured Text is a high-level text language that can be used to describe the behavior of Functions, Function Blocks, and programs, as well as the behavior of steps, actions, and transitions in sequential function diagrams.

Structured text programming language is a high-level language, similar to Pascal, developed especially for industrial control applications. For those familiar with high-level computer language development, structured text languages are easier to learn and use, enabling functions such as selection, iteration, and jump statements. In addition, structured text languages are easy to read and understand, especially when commented with meaningful identifiers and annotations. In complex control systems, Structured Text can greatly reduce the amount of code and make complex system problems simple, but the disadvantage is to make debugging not visualized and compilation is relatively slow.

The order of execution of programs using Structured Text is based on “line numbers” from top to bottom.

An expression includes an operator and an operand. The operand operates according to the rules specified by the operator to obtain the result and return it. Operands can be variables, constants, register addresses, Functions, etc.

3.2.2.2 Instruction Statement

There are five main types of structured text statements, namely assignment statement, function and function block control statement, selection statement, iteration (loop) statement, and jump statement. Following table lists all the statements used in structured text.

Statement type	Expression	Example
Assignment statement	:=	bFan:=TRUE;
function and function block control statement	Function/function block ()	fb_TON(IN := NOT(fb_TON.Q), PT := t_SetValue);
Selection statement	IF	IF< Boolean expression >THEN <>; END_IF;
	CASE	CASE< condition variable >OF <value1>:<statement content1>; <value2>:< statement content 2>; ...; <value n>:< statement content n>; ELSE <ELSE statement content >; END_CASE;
Iteration (Loop) statement	FOR	FOR<variable>:=< initial value >TO<target value>{BY<step-size>} DO <statement content>; END_FOR;
	WHILE	WHILE< Boolean expression >; <statement content>; END_WHILE;
	REPEAT	REPEAT; < statement content>; UNTIL < Boolean expression >; END_REPEAT;
Jump statement	EXIT	EXIT;
	RETURN	RETURN;

3.2.2.2.1 Assignment statement

The assignment statement is one of the most commonly used statements in Structured Text, the value generated by the expression on its right side is assigned to the operand (variable or address) on the left side, using “:=”, in the following format:

```
<variable>:=<expression>;
```

For example:

```
bFan:=TRUE;
```

```
bFan1:=FALSE;
```



- ◆ For data type matching, if there different data types on two sides, you should use the data type conversion function to convert them to the same type before assigning the value.
- ◆ There can be more than one statement in a line, such as “Test_1:=TRUE;Test_2:=FALSE;”, which are written in one line.
- ◆ The function return value is used as the value of the expression when the function is called, and it should be the latest result of the evaluation.

3.2.2.2.2 Function and Function Block Control Statement

Function and function block statements are used to call functions and function blocks.

3.2.2.2.2.1 Function Control Statement

After the function is called, the return value is assigned to the variable as the value of the expression.

The format of function control statement is as follows:

```
variable:=Function name ( parameter table ) ;
```

For example:

```
Test_ADD:=ADD(Test_int1,Test_int2);
```

3.2.2.2.2.2 Function Block Control Statement

When function block is called, the call is implemented by instantiating the function block name, e.g. Timer is the instance name of the TON function block. The format of the function block control statement is as follows:

```
function block instance name ( function block parameters ) ;
```

For example:

```
TON1(IN:=S1,Q=>Q1,ET=>et1);
```

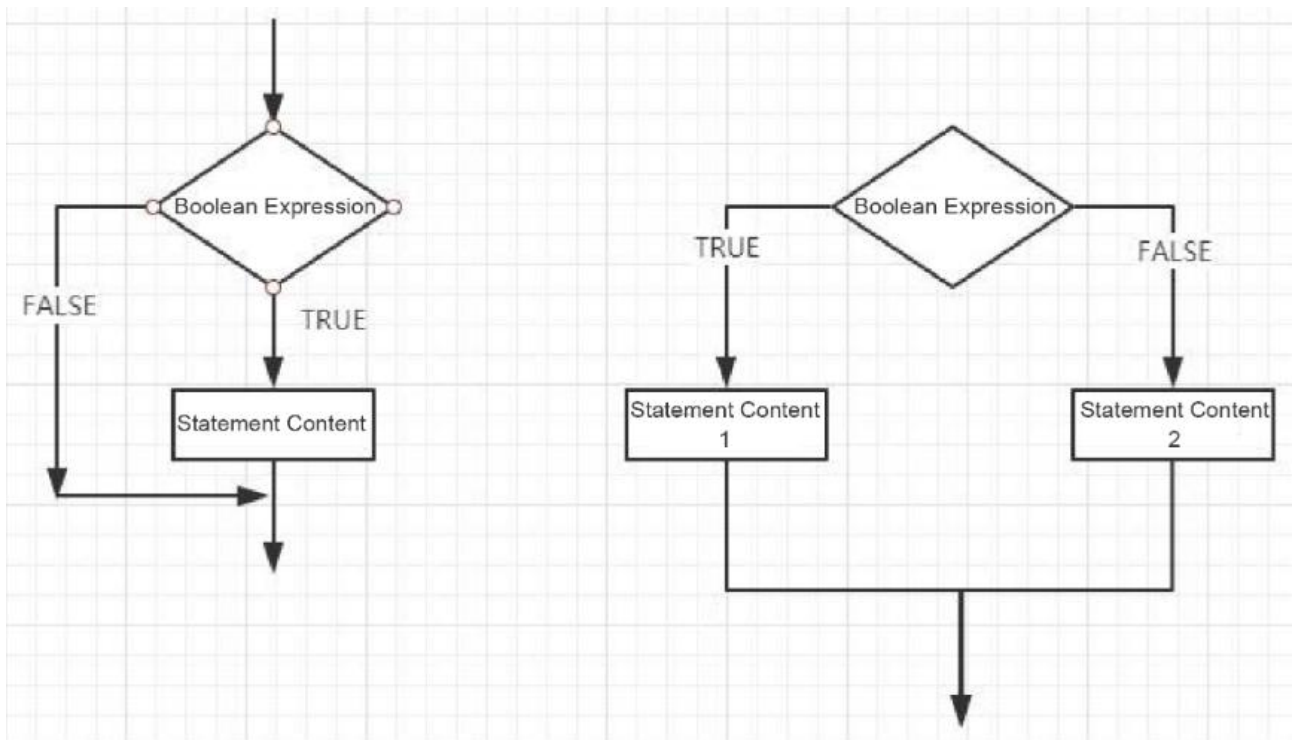

3.2.2.2.3 Selection Statement

3.2.2.2.3.1 IF Statement

Implement a single branch selection structure with an IF statement, the basic format of IF statement is as follows.

```
IF<Boolean expression> THEN  
  
<statement content>;  
  
END_IF;
```

When using the above format, 'statement content' is executed only if the result of 'Boolean expression' is TRUE, otherwise 'statement content' of IF statement is not executed. The 'statement content' can be a single statement, multiple statements in parallel, or even an empty statement. As shown in following Figure.



Note that IF statements can also be used in Ladder Diagram. The following figure indicates that executing program segment 1 when the variable **start** takes a value of TRUE.



3.2.2.2.3.2 IF...ELSE Statement

Implement the double branch selection structure with the IF... ELSE statement in the following basic format

```
IF <Boolean expression> THEN  
  
<statement content 1>;  
  
ELSE  
  
<statement content 2>;  
  
END_IF;
```

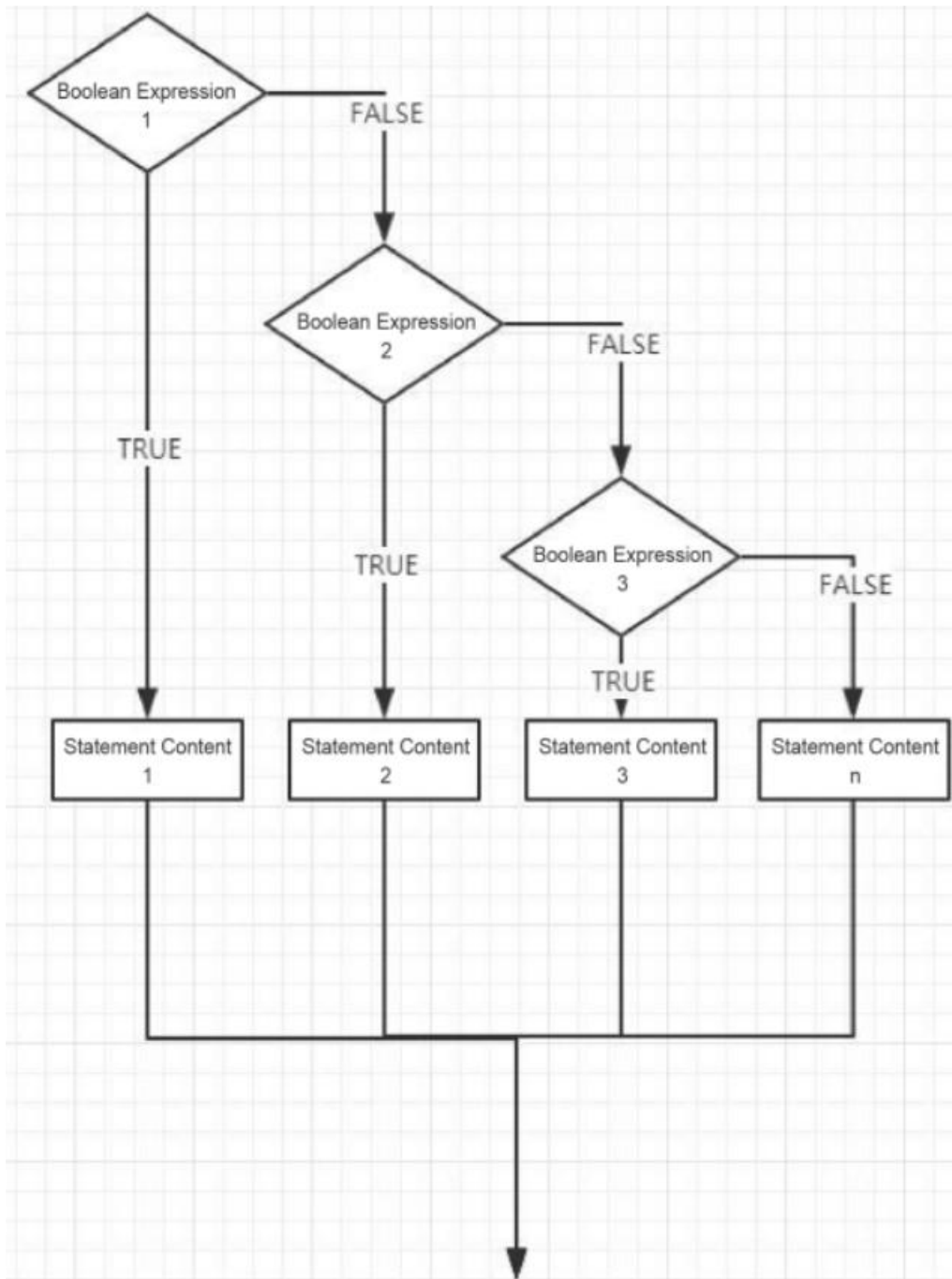
The above statement structure firstly determines the value of 'Boolean expression' and executes 'statement content1' if it is TRUE and 'statement content 2' if it is FALSE.

3.2.2.2.3.3 IF...ELSEIF...ELSE Statement

In addition, the multi-branch selection structure can be presented in the following format.

```
IF <Boolean expression 1> THEN  
  
<statement content 1>;  
  
ELSIF< Boolean expression 2> THEN  
  
< statement content 2>;  
  
ELSIF< Boolean expression 3> THEN  
  
< statement content 3>;  
  
...  
  
ELSE  
  
< statement content n>;  
  
END_IF;
```

If the expression 'Boolean expression 1' is TRUE, then only 'statement content 1' is executed, and no other commands are executed. Otherwise, it starts from 'Boolean expression 2' until one of the Boolean expressions is TRUE, and then executes the statement content corresponding to it. If the Boolean expressions are all FALSE, only 'statement content n' is executed. The execution flow of the IF...ELSEIF...ELSE statement is shown in following figure.



3.2.2.2.3.4 CASE Statement

The CASE statement is also a multi-branch selection statement, it enables the program to select one from multiple branches for execution based on the value of expression, with the following basic format:

```

CASE <condition variable>    OF
<value 1>:    <statement content 1>;
<value 2>:    <statement content 2>;
<value 3,value 4,value 5>:    <statement content 3>;
  
```

```

<value 6...value 10>:      <statement content 4>;
...
<value n>:      <statement content n>;
ELSE
<ELSE statement content>;
END_CASE;

```

The CASE statement is executed according to the following pattern:

- ◆ If the value of 'condition variable' is 'value i', then the corresponding command 'statement content i' is executed.
- ◆ If 'conditional variable' does not have any specified value, then the command 'ELSE statement content' is executed.
- ◆ If several values of a condition variable are required to execute the same instruction, then the values can be written one after another and separated by instructions so that the common instruction is executed, as in line 4 of the basic format above.
- ◆ If you need the condition variable to execute the same instruction within a certain range, then you can write the initial and final values separately, separated by two dots, so that the common instruction is executed, as in line 5 of the basic format above.

3.2.2.2.4 Iteration (Loop) Statement

Iteration (loop) statements are mainly used in programs that are repeatedly executed. In MCR Master, the common iteration (loop) statements are FOR, REPEAT and WHILE. The following is a detailed description of such statements.

3.2.2.2.4.1 FOR

The FOR loop statement is used to compute an initialization sequence that repeats the nested statements and computes a sequence of iterative expressions when a condition is TRUE, or terminates the loop if it is FALSE, in the following format.

```

FOR <variable>:=<initial value> TO <target value> {BY<step-size>} DO
<statement content>
END_FOR;

```

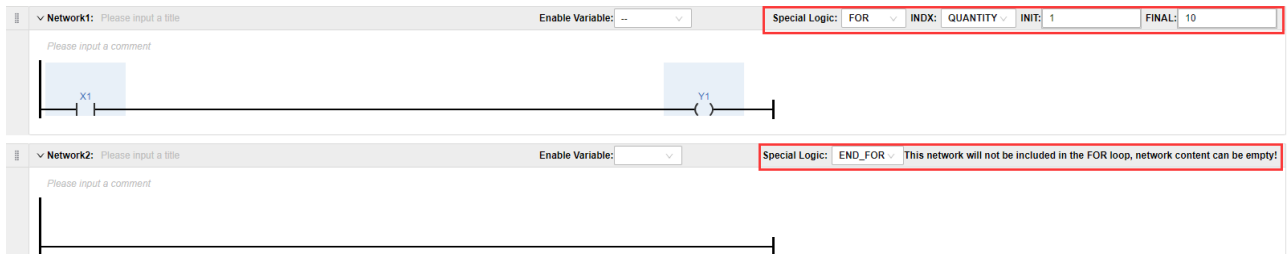
The order of executing the FOR loop is as follows:

Step1. Determine if 'variable' is within the range of 'initial value' and 'target value'.

Step2. Execute 'statement content' when 'variable' is within the range of 'initial value' and 'target value'. When 'variable' is not within the range of 'initial value' and 'target value', then 'statement content' is not executed.

Step3. Each time 'statement content' is executed, 'variable' always increases its value by the specified step-size. The step-size can be any integer value. If no step-size is specified, the default value is 1. When 'variable' is greater than 'target value', exits loop.

It should be noted that FOR loop can also be used in Ladder Diagram. The following figure shows that when the variable QUANTITY is incremented from 1 to 10, the program segment 1 is repeated 10 times.



3.2.2.2.4.2 WHILE

The WHILE loop is similar to the FOR loop, except that the end condition of the WHILE loop can be an arbitrary logical expression. The WHILE loop can specify a condition that, when satisfied, executes the loop in the following format.

```
WHILE <Boolean expression>
<statement content>;
END_WHILE;
```

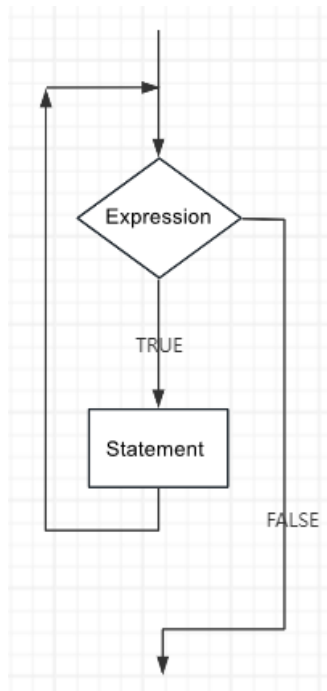
The execution sequence of the WHILE loop is as follows:

Step1. Calculate the return value of 'Boolean expression'.

Step2. Repeat 'statement content' when the value of 'Boolean expression' is TRUE.

Step3. When the value of 'Boolean expression' is FALSE, 'statement content' will not be executed and will jump to the end of the WHILE statement.

The flow chart of the WHILE loop is shown in following Figure.



In a certain sense, WHILE loop is more powerful than FOR loop because WHILE loop does not need to know the number of loops before it is executed. Therefore, in some cases, it is sufficient to use only WHILE loops. However, if the number of loops is clearly known, then the FOR loop is better because the FOR loop avoids creating a “dead loop”.

3.2.2.2.4.3 REPEAT

The REPEAT loop differs from WHILE loop in that the REPEAT loop checks for an end condition only after the instruction is executed, which means that the loop is executed at least once, regardless of the end condition. Its specific format is as follows:

```

REPEAT
<statement content>;
UNTIL
<Boolean expression>
END_REPEAT;
  
```

The execution sequence of the REPEAT loop is as follows:

- ◆ When the value of 'Boolean expression' is FALSE, execute 'statement content'.
- ◆ When the value of 'Boolean expression' is TRUE , stop executing 'statement content'.
- ◆ After the first execution of 'statement content', if the value of 'Boolean expression' is TRUE, then 'statement content' is executed only once.



If the value of 'Boolean expression' is always FALSE, then a “dead loop” will be generated, which should be avoided by changing the condition in the loop instruction section. For example, use a count up/down counter to avoid “dead loops”.

3.2.2.2.5 Jump Statement

3.2.2.2.5.1 EXIT

If the EXIT instruction is used in the FOR, WHILE, and REPEAT loops, the inner loop stops immediately regardless of the end condition, the format of EXIT statement is as follows:

```
FOR TEST_INT:=0 TO 5 BY 1 DO
Test_Counter:=ADD(Test_Counter,1);
IF Test_exit=1 THEN
EXIT;
END_IF;
END_FOR;
```

3.2.2.2.5.2 RETURN

The RETURN instruction is a return instruction used to exit the Program Organization Unit (POU).The format of it is as follows:

```
IF bSwitch=TURE THEN
RETURN;
END_IF;
nCounter:=nCounter+1;
```

3.2.2.2.6 Comment

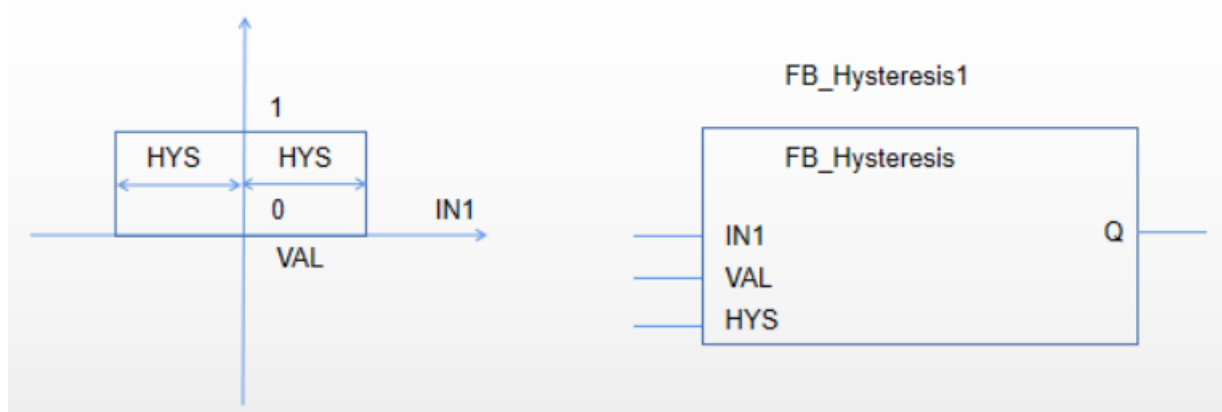
Comment is a very important part of a program, making it more readable without affecting its operation. Comments can be added anywhere in the declaration or execution sections of the Structured Text editor.

- ◆ Multi-line comments, starting with “(/*” and ending with “*)”. This commenting method allows comments to appear on multiple lines.
- ◆ A single line comment, starting with “//” and continuing to the end of the line.

3.2.2.3 Programming Example

3.2.2.3.1 Control Requirement

This function block has three input signals: the current real-time value input signal, the comparison set value input signal and the deviation value input signal. In addition, an output value is required. When the output signal is TRUE, the output switches to FALSE only if the input signal IN1 is smaller than $VAL - HYS$; when the output signal is FALSE, the output switches to TRUE only if the input signal IN1 is larger than $VAL + HYS$.



3.2.2.3.2 Function Block Programming

The program used to determine the input signal is as follows.

```
IF Q THEN
IF IN1<(VAL-HYS) THEN
Q:=FALSE;
END_IF;
ELSIF IN1>(VAL+HYS) THEN
Q:=TRUE;
END_IF;
```

3.2.2.3.3 Call Function Block

The **FB_Hysteresis** function block can be used for bit signal control, where **IN1** is connected to the process variable **rActualValue**, **VAL** is connected to the process set value **rSetValue**, and **rTolerance** is the desired control deviation. The body of the program is as follows:

```
fbHysersis(IN1:=rActualValue,VLA:=rSetValue,HYS:=rTolerance, Q=>bOutput);
```

The above part of the program can also be represented by the following program, which has the same result.


```
fbHystersis(IN1:=rActuallyValue,VAL:=rSetValue,HYS:=rTolerance);

bOutput:=fbHystersis.Q;
```

3.2.2.4 Elements of Structured Text programming language

Structured Text are composed of a certain number of basic language elements (minimal units) that are grouped together to form instructions or statements, including delimiters, keywords, identifiers, comments, etc.

3.2.2.4.1 Delimiter

A Delimiter is a character or combination of characters used to separate elements of a programming language. It is a special character, and different delimiters have different meanings; Following table shows application examples of various delimiters.

Delimiter	Application scenario	Comment or example
Space	Allow insertion of one or more spaces in the PLC program	Do not allow direct insertion of spaces in keywords, texts, identifiers and enumerated values
TAB	Allow insertion of TAB in the PLC program	Do not allow direct insertion of TAB in keywords, texts, identifiers and enumerated values
(*	ST Start comment	User-defined comments, which can be entered anywhere the program allows spaces
*)	ST End comment	
+	Prefix notation for decimal numbers (positive number)	+456、+1.23
	Add operator	23+11
-	Prefix notation for decimal numbers (negative number)	-789
	Separator for year, month and date	D#2023-07-31
	Subtract operator	55-23
#	Base numbers separator	2#1101、16#EF
	Data type separator	SINT#123
	Time/text separator	T#200ms、TOD#05:30:20、t#12m45s
.	Separator of integers and decimals	3.14
	Classful addressing separator	%IX0.3
	Structure element separator	Channel[0].Type、abc.number

Delimiter	Application scenario	Comment or example
	Function block structure separator	TON1.Q、SR_3.S1
E/e	Real/exponent separator	10e+6、3.14E6
‘	String start/end mark	‘Hello Word’
\$	Start of special characters in a string	‘\$L’ means line feed; ‘\$R’ means carriage return
:	Moment text separator	TOD#12:14:11
	Separator between variables and types	Test:INT
:=	Initialization operator	VAL1:INT:=1
	Input variable connection operator	INT_2(SINGLE:=Z2,PRIORETY:=1)
	Assignment operator	Val2:=45
()	Enumeration table separator	V:(B1_10V,UP_10V,IP_15V):=UP_10V
	Subrange separator	DATA_INT(-32768..32767)
	Initialize iteration factor	ARRAY(1..2,1..3) OF INT:=1,2,3(4),6
	Function independent variable	Var2*LIMIT(Var1)
	Sub-expression grading	(A*(B-C)+D)
	Function block input table separator	TON_1(IN:=#IX5.1,PT:=T#500ms);
[]	Array subscript separator	MOD_5_CFG.CH[5].Range:=BI_10V;
,	Enumerated value separator	V:(B1_10V,UP_10V,IP_15V):=UP_10V
	Initial value separator	ARRAY(1..2,1..3) OF INT:=1,2,3(4),6
	Array subscript separator	ARRAY(1..2,1..3) OF INT:=1,2,3(4),6
	Declared variable separator	VAR_INPUT A,B,C:REAL;
	Function block initial value separator	TON_1(IN:=#IX5.1,PT:=T#500ms);
	Function block input table separator	SR_1(S1:=%IX1.1,RESET:=IX2.2);
	Operand table separator	ARRAY(1..2,1..3) OF INT:=1,2,3(4),6
	Function independent variable separator	LIMIT(MN:=4,IN:=%IW0,MX:=20);
	CASE value table separator	CASE STEP OF 1,5:DISPLAY:=FALSE;
;	Type separator	TYPE R:REAL;END_TYPE
	Statement separator	A:=B+C;D:=A+B;

Delimiter	Application scenario	Comment or example
..	Subrange separator	ARRAY(1..2,1..3);
	CASE range separator	CASE STEP OF (1..5):TEST_X:=FALSE;
%	Prefixes that directly represent variables	%IW0
=>	Output connection operator	C10(CU:=bInput,Q=>Out);

3.2.2.4.2 Keyword

Keywords are lexical units for the characterization of language elements. In the IEC 61131-3 standard, keywords are used as programming language words to define different structures or specific software elements.

Some keywords are used in pairs, like “FUNCTION” and “END_FUNCTION”, and some keywords can be used separately, like “ABS”. Keywords cannot be used for any other purpose, such as variable names or extension names, i.e. neither TON as a variable name nor VAR as an extension name.

As keywords are standard identifiers, they cannot contain spaces. The keywords and descriptions commonly used in MCR Master are listed in following table.

Keyword	Description
PROGRAM/END_PROGRAM	Start /end of program segment
FUNCTION/END_FUNCTION	Start /end of function segment
FUNCTION_BLOCK/END_FUNCTION_BLOCK	Start /end of function block segment
VAR/END_VAR	Start /end of internal variable segment
VAR_INPUT/END_VAR	Start /end of input variables segment
VAR_OUTPUT/END_VAR	Start /end of output variables segment
VAR_IN_OUT/END_VAR	Start/end of input variables segment/output variable segment
VAR_GLOBAL/END_VAR	Start/end of global variable segment
ARRAY OF	Array
AT	Direct address
EN/ENO	Enable input/Enable output
TRUE/FALSE	logic “1”/logic “0”
TYPE/END_TYPE	Start/end of data type segment

Keyword	Description
STRUCT/END_STRUCT	Start/end of Struct
IF THEN ELSIF ELSE END_IF	IF statement
CASE OF ELSE END_CASE	CASE statement
FOR TO BY DO END_FOR	FOR loop statement
REPEAT UNTIL END_REPEAT	REPEAT loop statement
WHILE DO END_WHILE	WHILE loop statement
RETURN	Return operator
NOT、AND、OR、XOR	Logic operator

3.2.2.4.3 Constant

Numeric Literal is used to define a numeric value, which can be a number in decimal or other positional notations. There are two types of Numeric Literal: Integer Literal and Real Literal, which are written in the following format.

<Type>#<numerical value>

<Type>:Specify the desired data type. The supported types are BOOL, SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT, DWORD, REAL and LREAL.

< numerical value >:Specify a constant. The input data must match the specified data type <Type>.

3.2.2.4.3.1 Numerical Constant

Numerical constant can be expressed in binary, decimal and hexadecimal.

- ◆ The binary expression is 2#'value';
- ◆ The 10# before the decimal value does not need to be entered;
- ◆ The hexadecimal expression is 16#'value'.

The details are shown in the following table.

Numeric literal type	Expression	Example
Integer	Integer type name# Decimal integer	INT#-34;INT#21
	2#Binary integer	2#00100010
	8#Octal integer	8#77
	16# Hexadecimal integer	16#E5
Real number	Real type name#signed integer.integer[exponent]	REAL#4.94,6.3e-7

Numeric literal type	Expression	Example
	Signed integer.integer	3.1415926
Boolean number	[Boolean type name#] 0 or 1, False or True	0,1,TURE,BOOL#1

3.2.2.4.3.2 BOOL Constant

BOOL constants are logical values TRUE and FALSE, and can also be expressed as 1 (true) and 0 (false), which have the same meaning.

3.2.2.4.3.3 TIME Constant

The TIME constant is expressed by a start character T or t and the numeric identifier #, plus the actual time, but don't need to contain all units of length of time; care should be taken when using it that the order of days, hours, minutes and seconds is not misplaced and the units of time must be consecutive.

For example:T#14ms, T#100s12ms, t#12h34m15s

3.2.2.4.3.4 DATE Constant

These constants can be used to represent dates. When declaring a DATE constant, the starting character is d, D, followed by a # sign, and then any date can be represented in the “year-month-day” format.

For example:D#2023-08-02, d#2023-01-12

3.2.2.4.3.5 TIME_OF_DAY Constant

Different times of the day can be saved using this type of constant. The declaration of the TIME_OF_DAY constant uses the start character tod#, TOD# followed by a time in the format “hour:minute:second”. The seconds value can be an integer or a decimal.

For example:TOD#15:30:36.123, tod#00:01:15

3.2.2.4.3.6 DATE_AND_TIME Constant

DATE constants and TIME_OF_DAY constants can be combined to form a so-called DATE_AND_TIME constant. dt#, DT# are the starting characters for DATE_AND_TIME constants.

For example:DT#2023-08-03-15:33:02、 dt#2023-08-01-00:03:34

3.2.2.4.3.7 REAL/LREAL Constant

REAL and LREAL constants can be expressed in decimal fraction and exponential form, using the US format with a decimal point for real number (REAL/LREAL).For example:

7.4 replaces 7,4、

1.64e+009 replaces 1,64e+009



In MCR Master, the values of REAL and LREAL must have decimal places, e.g. 1.0, 3.4.

3.2.2.4.3.8 STRING Constant

A string is a queue of characters. STRING constants use a single quote as their prefix and suffix. Blank spaces and special characters can also be entered and these will be treated like all other characters, examples of expressions for strings are as follows.

For example: ‘MCR Master!!’, ‘MX On CORPORATION’, ‘-_-’

In the character queue, the combination of the “\$” sign and the two hexadecimal numbers that follow it is interpreted as a hexadecimal expression of the 8-bit character code. In addition, the meaning of the combination of two characters with \$ as the start character is shown in following table.

Character combinations	Description
\$ < two hexadecimal numbers >	Hexadecimal expression of ASCII code
\$\$	Dollar sign
\$’	Single quote
\$L or \$l	Line feed
\$N or \$n	New line
\$P or \$p	Input page
\$R or \$r	Enter
\$T or \$t	<Tab> key

3.2.2.4.4 Space and Comment

3.2.2.4.4.1 Space

One or more spaces are allowed to be inserted anywhere in the program text. Spaces are not allowed inside keywords, identifiers, delimiters, etc.

3.2.2.4.4.2 Comment

It is common to add comments to areas with strong logical of a program to show how the logic of the program works and to make it easier for you and others to understand later. Proper comments can improve the readability of the code.

Comments are allowed in all text editors, declaration editors, statement tables, Structured Text languages and user-define data types. If the project uses a template for printout, comments entered during variable declaration appear in the text-based programming component after each variable. The software has different commenting methods for different programming languages and editors, which are explained in detail below.

◆ Comments of Structured Text

- Comments in brackets: start with “(*)”, end with “*)”, e.g.

```
(*Test*)
```

- Single line comments: line comments, using the symbol “//”, e.g.

```
//Test
```

◆ Comments of Ladder Diagram

In Ladder Diagram, the user can add comments to the variables in the variable table.

#	Name	Variable Type	Data Type	Init	Cons/Retain	Address	Comment
1	start	Local	BOOL				click to start pump
2	X1	Local	BOOL				input
3	Y1	Local	BOOL				output
4	QUANTITY	Local	INT				

3.3 Variable

3.3.1 Overview

Variables are abstract data stored in memory to be processed and are names used to identify the inputs/outputs of the PLC, the storage area inside the PLC can be used in the program instead of the physical address.

The value of the data stored in a variable can be changed at any time as required. The value of a variable can change during the execution of a program. Variables must be declared before they can be used, and their type and name must be specified. Variables have a name, a type and a value. The data type of a variable determines the size and type of memory it represents. The variable name is the identifier in the program source code.

Variables can be used to represent a numerical value, a string value, or an array, etc. MCR Master divides the data types of variables into standard data types, extended data types, and custom data types.

3.3.2 Variable Name

The identifier is the name of the variable. When defining an identifier, the MCR Master can use Chinese. When using English, numeric definitions, which must consist of letters, numbers, and underscore according to the IEC 61131-3 standard. In addition, the following rules should be followed:

- ◆ The first character of the identifier must be a letter or underscore, the last character must be a letter or a number, and letters, numbers, and underscores are allowed in between.

- ◆ Case-insensitive in identifiers.
- ◆ Underscores are part of the identifier, but two or more consecutive underscores are not allowed in the identifier.
- ◆ Variable names must not exceed 32 characters in length.
- ◆ No spaces are allowed.

3.3.3 Data Type of Variable

The data types of variables in the MCR Master mainly include basic data types, custom data types, function blocks, and arrays.

3.3.3.1 Basic Data Type

The data types that MCR Master support are listed in the following table.

Data type	Type code	Data length (bit)	Range
Boolean	BOOL	1	0 or 1
Bit string of length 8	BYTE	8	0~255
Bit string of length 16	WORD	16	0~65535
Bit string of length 32	DWORD	32	0~ ($2^{32}-1$)
Bit string of length 64	LWORD	64	0~ ($2^{64}-1$)
Integer	INT	16	-32678~32677
Double integer	DINT	16	$-2^{31} \sim (2^{31}-1)$
Short integer	SINT	8	-128~127
Long integer	LINT	64	$-2^{63} \sim (2^{63}-1)$
Unsigned integer	UINT	16	0~65535
Unsigned double integer	UDINT	32	0~ ($2^{32}-1$)
Unsigned long integer	ULINT	64	0~ ($2^{64}-1$)
Unsigned short integer	USINT	8	0~255
Single-precision real number	REAL	32	positive number ($1.4\text{E}-45 \sim 3.4\text{E}+38$) ; negative number ($-3.4\text{E}+38 \sim -1.4\text{E}-45$)
Double-precision real number	LREAL	64	positive number ($4.9\text{E}-324 \sim 1.798\text{E}+308$) ; negative number ($-1.798\text{E}+308 \sim -4.9\text{E}-324$)

Data type	Type code	Data length (bit)	Range
Duration	TIME	32	T#-24d20h31m23s648ms ~T#24d20h31m23s647ms
Date(only)	DATE	16	D#1970-01-01~D#2038-01-19
Time of day(only)	TOD	32	TOD#00:00:00.000~TOD#23:59:59.999
Date and Time of Day	DT	64	DT#1970-01-01- 00:00:00.000~DT#2038-01-19- 00:00:00.000

3.3.3.2 Custom Data Type

For more information about custom data type, please refer to [Custom Data Type](#).

3.3.3.3 Function Block

Function blocks include custom function block and built-in system function block. For more information about custom function blocks, refer to [Function Block](#). Refer to [MCR Master Built-in Instruction](#) for the system built-in function blocks.

3.3.3.4 Array

An array is a union of ordered data, where each element has the same data type. For example, if a device has 20 points to measure temperature, without using an array, you would declare nTemp1, nTemp2.... nTemp20, 20 variables in total as the specific temperature values of the measurement points, but if you use an array, you only need to define an array nTemp with 20 members to complete the definition of these 20 temperature variables. The specific ST instruction is as follows:

```
nTemp :ARRAY [1..20 ] OF REAL;
```

The data in the square bracket indicate the 20 defined measurement points from 1 to 20, e.g. nTemp[17] indicates the temperature value of the 17th measurement point.

In MCR Master, 1D, 2D and 3D arrays can be defined directly as data types. The user can define arrays in custom data type with the interface as shown in following figure.

Data Type :

StartIndex : EndIndex :

3.3.3.4.1 One-dimensional Array

One-dimensional arrays are one of the common data types, and the number inside [] is the length of the array.

3.3.3.4.2 Two-dimensional Array

A two-dimensional array can be considered as a special one-dimensional array whose own elements can be understood as a new one-dimensional array. For example, a can be considered as a one-dimensional array with three elements: a[0], a[1], and a[2], each of which contains a one-dimensional array of four elements. As shown in the following figure.

Data Type :

StartIndex : EndIndex :

StartIndex : EndIndex :

3.3.3.4.3 Three-dimensional Array

Similar to the way a two-dimensional array is understood, a three-dimensional array is obtained by considering each element of a two-dimensional array as a one-dimensional array.

Data Type :

StartIndex : EndIndex :

StartIndex : EndIndex :

StartIndex : EndIndex :

3.3.3.4.4 Initialization of Array

Initial values are assigned to array elements when defining the array, either all or only some of them. For duplicate

dispositions, you can define them in bulk by simply prefixing the parentheses with the number, for example:

```
arr1 : ARRAY [1..5] OF INT :=[1,2(3)];
```

For 2D/3D arrays, you can write all the data in the square bracket and assign initial values to the elements in the order in which the arrays are arranged, for example:

```
arr2: ARRAY[1..2,3..4] OF INT :=[1,3(7)];
```

Output of the above assignments is:

```
arr2[1,3]=1,arr2[1,4]=7,arr2[2,3]=7,arr2[2,4]=7
```

3.3.3.4.5 References to Array

Array must be defined before it is referenced. The data reference form is as follows:

```
<Array name>[Index1,Index2,Index3]
```

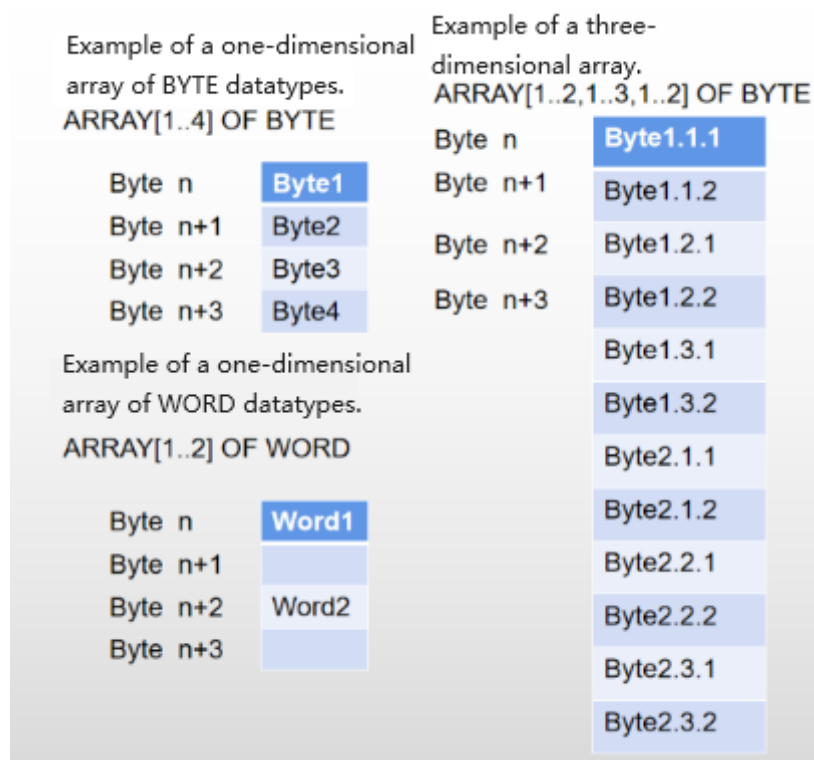
3.3.3.4.6 Storage Structure of Array Variables

When a program is running, to access array elements, it is important to understand how array variables are stored in memory first.

Array variables start at word boundaries, i.e., the starting address is an even BYTE address. Then each structure element is stored in memory in the order in which it was declared. Structure elements of data types BOOL and BYTE are stored starting from even bytes, and array elements of other data types are stored starting from word addresses, as shown in the one-dimensional arrays in the following figure.

The order of the elements in a two-dimensional array is by row, i.e., the elements of the first row are stored in memory in order, followed by the elements of the second row, and so on.

The order in which the three-dimensional arrays are arranged in memory: the subscripts in the first dimension change the slowest, and the rightmost subscripts change the fastest.



3.3.3.5 ANY Data Type

Many built-in instructions in MCR Master provide pins that support multiple data types to facilitate convenient input, where the prompt box will be displayed as ANY_XXX when a major data type is supported, where XXX is the name of a data type, as shown in the following table.

I	II	III	IV
ANY	ANY_NUM	ANY_INT	INT
			UINT
			SINT
			DINT
			UDINT
			LINT
			ULINT
		ANY_REAL	REAL
			LREAL
	ANY_BIT	BIT	
		BYTE	

I	II	III	IV
		WORD	
		DWORD	
		LWORD	

The first row in the table is the priority, if it is level I it means that any type can be used, if it is level II it means that data types of level III and IV under its type can be used, but no other data types of level I and II can be used.

3.3.4 Variable Property

The MCR Master defines properties for variables according to the IEC 61131-3 standard, assigning their related performance to variables by setting the variable properties. Variables can be categorized according to their scope of application, and in addition to variable data types, MCR Master provides additional properties for variables.

Variables in the MCR Master can be divided into different variable types depending on the scope. Please refer to the table below for details.

Variable type keyword	Variable type	External R/W	Internal R/W
VAR	Local variable	–	R/W
VAR_INPUT	Input variable, provided externally	R/W	R
VAR_OUTPUT	Output variable, provided by internal variable to external	W	R/W
VAR_IN_OUT	Input-output variable	R/W	R/W
VAR_GLOBAL	Global variables that can be used within all resources	R/W	R/W
VAR_TEMP	Temporary variable, variable used for internal storage of programs and function blocks	-	R
VAR_EXTERNAL	External variable, which can be modified inside the program, but need to be provided by global	R/W	R/W

Variable type keyword	Variable type	External R/W	Internal R/W
	variable		

In MCR Master, the additional property of the variables are shown in the table below.

Variable additional property keyword	Description
RETAIN	Retain variables for power-off retain
CONSTANT	Constant

◆ RETAIN

The variable is declared with the keyword RETAIN, which retains its value after the controller has been shut down normally, opened (or received an online command to “stop”), or even shut down unexpectedly. The stored value will continue to function as the program starts running again. RETAIN variables are declared in the following format.

VAR RETAIN

<identifier>:<data type>;

END_VAR

However, the RETAIN variable will be reinitialized after the “initial value reset”, “cold reset”, and the program download. RETAIN variables are stored in a separate memory area.

◆ CONSTANT

A quantity that can only be read, but not modified, during program operation is called a constant, with the keyword CONSTANT. Constants can be declared either as local constants or as global constants.

Constants are declared in the following format.

VAR CONSTANT

<identifier>:<data type>:=<initial value>;

END_VAR

In practice, it is often possible to set important parameters or coefficients as constants, so that they are not modified by other variables, which would ultimately affect the overall stability and security of the system.

MCR Master provides a more convenient way to set up variables that you can specify as constants when editing them.

#	Name	Variable Type	Data Type	Init	Cons/Retain	Address	Comment
1	start	Local	BOOL		Constant		click to start pump
2	X1	Local	BOOL				input
3	Y1	Local	BOOL				output

3.3.5 Initialization of Variable

In programs, sometimes it is necessary to preassign initial values to some variables, MCR Master allows variables to be initialized when they are defined, with the variable and its type to the left of the assignment operator “:=”, and the value of the specified address or expression on to the right.

Multiple variables of the same type can be assigned initial values simultaneously, e.g.

```
Test_BOOL1,Test_BOOL2,Test_BOOL3:BOOL:=TRUE;
```

Assigning initial values to arrays, structs, etc. is relatively complex, with the following example of assigning initial values to arrays.

```
VAR  
  
ARR1:ARRAY[1..5] OF INT:=[1,2,3,4,5];  
  
ARR2:ARRAY[1..2,3..4] OF INT:=[1,3(7)];  
  
ARR3:ARRAY[1..2,2..3,3..4] OF INT:=[2(0),4(4),2,3];  
  
END_VAR
```

Structs can be initialized with only some of their members, as in the following example:

```
VAR  
  
MOTOR_1:MOTOR:=(VENDOR:='FESTO',BRAKE:=TRUE);  
  
END_VAR
```

The initial value of the variable can be entered directly in the variable table.

Main * ×

Array ×

Add

Import

Export

Move Up

Move Down

Convert Global Variable

Delete

🔍 Search

☐	#	Name	Variable Type	Data Type	Init
☐	1	start	Local	BOOL	TRUE
☐	2	X1	Local	BOOL	FALSE
☐	3	Y1	Local	BOOL	TRUE

3.3.6 Mapping of Variable

Click **configuration** at the upper left corner of the MCR Master to view all the hardware configurations of the PLC, which includes the hardware I/O addresses. The user can bind the I/O addresses of the inputs/outputs to the required variables according to their needs, and when the I/Os have external inputs and outputs, the value of that bound

variable will also change, and similarly, when the bound value changes, the corresponding I/Os will also have inputs/outputs, which is the mapping of variables.

Used	Address	Variable	Owner	Ordinary Filter(...)	Hardware Filter
	0.IX0.0			8	1440ns
	0.IX0.1			8	1440ns
	0.IX0.2			8	1440ns



MCR Master does not support two or more variables mapped to the same address. To achieve the same functionality, a global variable can be used to map the corresponding address and then called in the required program or function block.

3.3.7 Variable Declaration

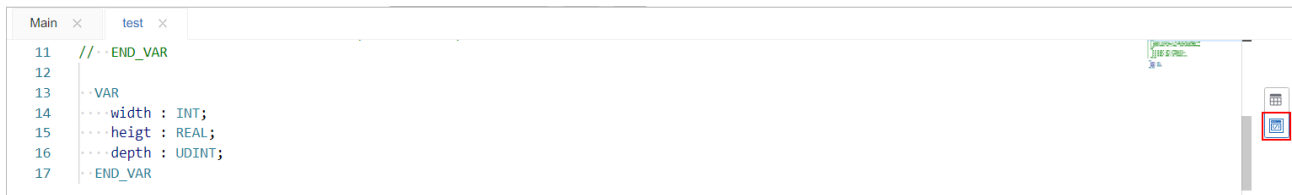
Variable declaration is the process of specifying the name and data type of a variable and assigning an initial value. The declaration of variables is very important and undeclared variables cannot be compiled or used in the program. Variables can be declared in the Program Organization Unit (POU), in the Table of Global Variables and in the automatic declaration dialog. There are two types of declaration: normal variable declarations and direct variable declarations, the latter require the variable to be linked to the correct address.

As shown below, you can create variables in two ways: using tables and using the ST language.

- ◆ Click icon, Create variables in tables, drag and drop variables in order of variables.

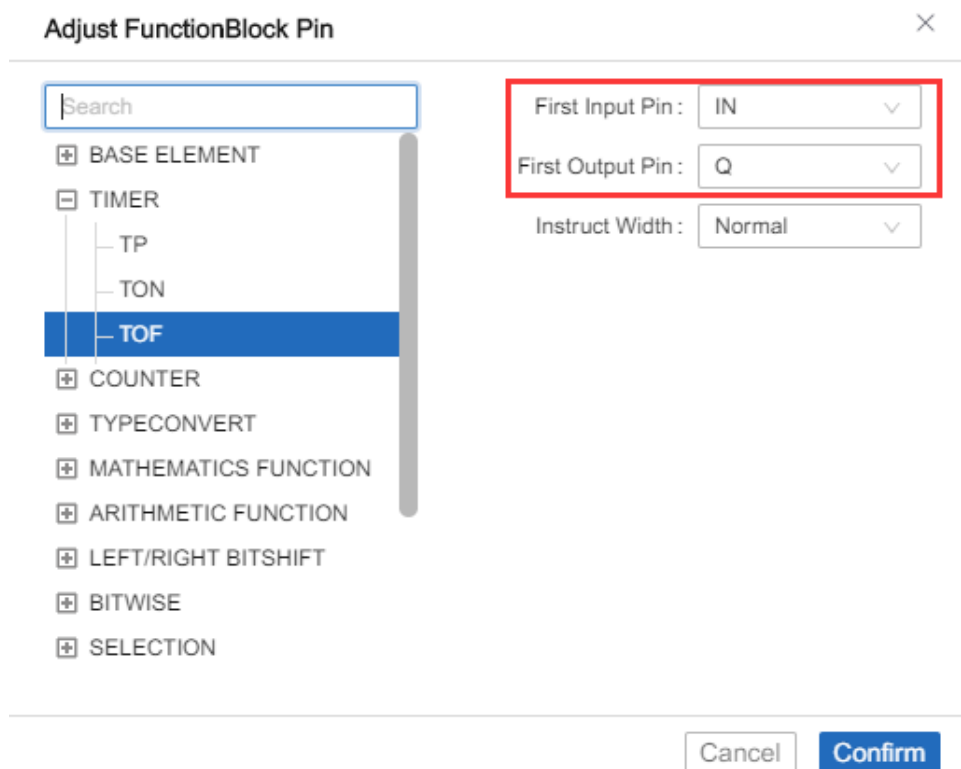
#	Name	Variable Type	Data Type	Init	Cons/Retain	Address	Comment	Used Times
1	start	Local	BOOL	TRUE	Constant		click to start pump	0
2	X1	Local	BOOL	FALSE			input	1
3	Y1	Local	BOOL	TRUE			output	1

- ◆ Click icon, create variable using ST language.



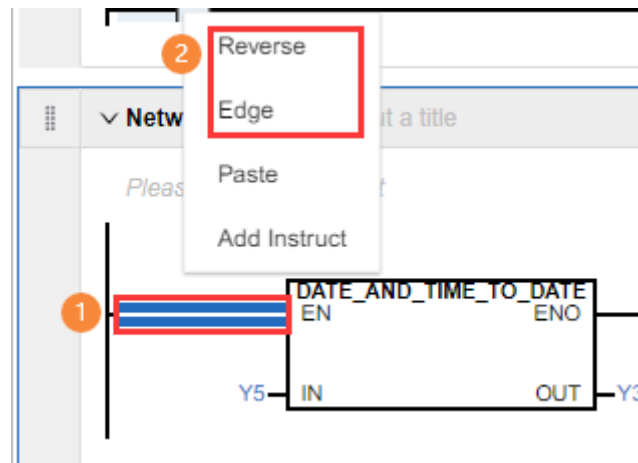
3.4 Adjust Instruction Pin

When using a instruction (built-in Function or Function Block) , modify the enable (EN/ENO) pin, just double-click the instruction block you want to modify, in the pop-up dialog box, select whether to use the enable pin (If you do not use an EN/ENO pin, you need to specify the first input pin and first output pin) .



3.5 Switch Instruction Trigger State

If you need to switch the instruction trigger state, you can modify the instruction trigger state by clicking on the horizontal connection line connected to the instruction trigger pin, right-clicking, and selecting the appropriate option (“Reverse” or “Edge “) in the pop-up interface.



For detailed configuration method, please refer to the table below.

Parameter	Description
Reverse	The trigger state of the instruction is reversed() icon), if it is currently triggered at a high level, after being reversed it becomes triggered at a low level.
Edge	Rise edge trigger() icon) Fall edge trigger () icon)

4 MCR Master Built-in Instruction

The MCR Master has built-in instructions including basic element, functions, and function blocks that can be called directly to reduce programming effort.

4.1 Basic element

4.1.1 Contact

A contact is a graphical element of a ladder diagram. The contact of Ladder Diagram follows the contact terminology of electrical logic diagrams and is used to represent a change in state of a Boolean variable. A contact is a element that passes a state to the horizontally connected element to its right.

The contacts can be divided into Normally Open Contact (NO) and Normally Closed Contact (NC). Normally open contact means the contact is open under normal operating conditions, and its state is FALSE. Normally closed contact means the contact is closed under normal operating conditions, and its state is TRUE. The following table lists the common contact types.

Type	Graphic symbol	Description
Normally Open Contact		If the contact corresponds to the current Boolean variable with a value of TRUE, the contact is pulled-in. If the state of the connected element on the left side of the contact is TRUE, the state TRUE is passed to the right side of the contact, making the state of the connected element on the right side is TRUE. Conversely, when the Boolean variable value is FALSE, the state of the right connection element is FALSE.
Normally Closed Contact		If this contact corresponds to the current Boolean variable with a value of FALSE, the normally closed contact is pulled-in, and if the state of the connected element on the left side of the contact is TRUE, the state TRUE is passed to the right side of the contact, making the state of the connected element on the right side TRUE. Conversely, when the Boolean variable value is TRUE, the contact is disconnected and the state of the connected element on the right side is FALSE.
Rising Edge Contact		Only if the contact corresponds to the moment when the value of the current Boolean variable changes from False to True, it is pulled-in, so that the state of the right connected element is TRUE.
Falling Edge Contact		Only if the contact corresponds to the moment when the value of the

Type	Graphic symbol	Description
Contact		current Boolean variable changes from False to True, it is pulled-in, so that the state of the right connected element is TRUE.

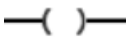
Depending on the state of the contact and the state of the left connection element to which the contact is connected, the state of its right graphic symbol is determined according to the following rules.

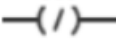
- ◆ Only when the state of the graphical element on the left side of the contact is TRUE, its state can be passed to the graphical element on the right side of the contact, pass in accordance to the following passing rules.
 - If the contact state is TRUE, the state of the graphical element to the right of the contact is TRUE.
 - If the contact state is FALSE, the state of the graphical element to the right of the contact is FALSE.
- ◆ When the state of the left graphic element of the contact is FALSE, the state of the contact cannot be passed to the right graphic element, regardless of the state of the contact, that is, the state of the right graphic element is FALSE.
- ◆ When the left graphical element state of the contact changes from FALSE to TRUE, its related variable also changes from FALSE to TRUE, then the state of graphic element in the right of the contact changes from FALSE to TRUE. It will hold for one cycle, then changes to FALSE, which is called rising edge trigger.
- ◆ When the state of the graphic element in the left of contact changes from FALSE to TRUE, variable related to the contact also changes from FALSE to TRUE, then the state of graphic element in the right of the contact changes from FALSE to TRUE. It will hold for one cycle, then changes to FALSE, which is called rising edge trigger.
- ◆ When the state of the graphic element in the left of contact changes from TRUE to FALSE, variable related to the contact also changes from TRUE to FALSE, then the state of graphic element in the right of the contact changes from TRUE to FALSE. It will hold for one cycle, then changes to FALSE, which is called falling edge trigger.

4.1.2 Coil

4.1.2.1 Coil Type

Coils are the graphical elements of ladder diagrams. Coils in ladder diagrams follow the terminology of coils in electrical logic diagrams and are used to represent changes in state of Boolean variables. Depending on the characteristics of the coils, they can be divided into instantaneous coils and latching coils, where the latching coils are divided into setting coils and resetting coils. The following table lists the common coil symbols and descriptions for graphical diagrams.

Type	Graphical symbol	Description
Instantaneous coil		The state of the left connected element is passed to the relevant Boolean variable and to the right connected element. If the state of the left connected element of the coil is TRUE, the Boolean type variable

Type	Graphical symbol	Description
		for the coil is TRUE, and vice versa, the Boolean type variable for the coil is FALSE.
Set coil		There is an S in the coil. When the state of the left connected element is TRUE, the Boolean variable for that coil is set and held until reset by the reset coil.
Reset coil		There is an R in the coil. When the state of the left connected element is TRUE, the Boolean variable for that coil is reset and held until set by set coil.。
Negation coil		There is a “/” in the coil. The state of the Boolean variable of this coil is False when the left connected element is TRUE, and True vice versa.

4.1.2.2 Passing rule of the coil

The coil is a Ladder Diagram element that passes the state of the horizontally or vertically connected element on the left to the horizontally connected element on its right without any change. During the transfer, the state of the relevant variables and direct addresses of the left connection are stored in the appropriate Boolean variables. Conversely, the negation coil is the Ladder Diagram element that passes the negation state of its left-hand horizontally or vertically connected element to the right-hand horizontally connected element.

The set and reset coils hold the state of the moment when the left side horizontally connected element changes from FALSE to TRUE and from TRUE to FALSE for one evaluation cycle, and pass its left side horizontally connected element state to the right side horizontally connected element at other times.

No rule stipulates that on the right side only one element can be connected, so the user can expand on the right side, thus simplifying the process.

4.1.2.3 Double coil

A double coil is a phenomenon where the same coil is used twice or more in a user program, and it is called a double coil output. In the same scan cycle, the logic of the two coils may have opposite results, i.e., one coil may be “powered on” and the other may be “powered off”.

Such a double coil output is allowed as long as only the logic operation corresponding to one of the coils is executed in the same scan cycle.

The following two cases allow double coil outputs.

- ◆ Double coil outputs are allowed in two program segments with opposite judgment conditions (e.g., automatic and manual programs), i.e., coils of the same variable can appear once in each of the two program segments.

In practice, the PLC executes only one coil output instruction of the double coil element in the program segment being processed.

- ◆ Double coil phenomenon is allowed in two subroutines with opposite call conditions (e.g., automatic and manual programs), i.e., coils of the same variable can appear once in each of the two subroutines. Commands in a subroutine are executed only when the program is called, not when it is not called

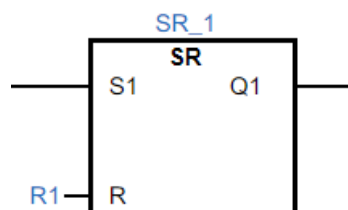
To avoid double-coil output, the set/reset instruction can be used multiple times for the same variable.

4.1.3 SR Set-Reset Flip-Flop

4.1.3.1 Function Description

This function block is used to set the variable priorly when it is triggered by the set and reset instruction at the same time. The operator is SR.

4.1.3.2 Graphic



4.1.3.3 Input Parameter

Parameter	Data type	Description
S1	BOOL	Set, set signal input, output parameter Q1 is set to TRUE when both S1 and R or only S1 change from FALSE to TRUE.
R	BOOL	Reset, reset signal input, output parameter Q1 is set to FALSE when only R change from FALSE to TRUE.

4.1.3.4 Output Parameter

Q1: Operand state output, data type of it is BOOL, determines the final output set or reset state result based on the input signal state.

4.1.3.5 Variable Declaration

VAR

set:BOOL;

```

reset:BOOL;

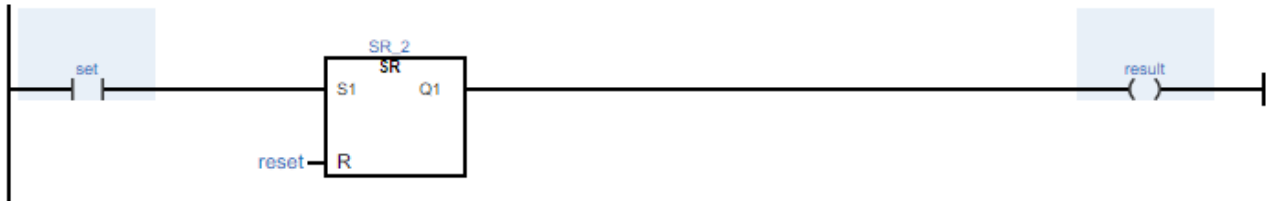
result : BOOL;

SR_2 : SR;

END_VAR

```

4.1.3.6 Expression in LD



4.1.3.7 Expression in ST

```

SR_2(S1 := set, R := reset);

result := SR_2.Q1;

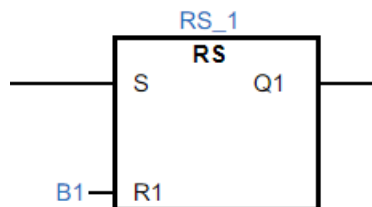
```

4.1.4 RS Reset-set Flip Flop

4.1.4.1 Function Description

This function block is used to reset the variable priorly when it is triggered by the set and reset instruction at the same time. The operator is RS.

4.1.4.2 Graphic



4.1.4.3 Input Parameter

Parameter	Data type	Description
S	BOOL	Set, set signal input, output parameter Q1 is set to TRUE when only S pin changes from FLASE to TRUE.

Parameter	Data type	Description
R1	BOOL	Reset, reset signal input, output parameter Q1 is set to FALSE when both S and R1 or only S change from FALSE to TRUE.

4.1.4.4 Output Parameter

Q1: Operand state output, data type is BOOL, determines the final output set or reset state result based on the input signal state.

4.1.4.5 Variable Declaration

```

VAR

  RS_1 : RS;

  set : BOOL;

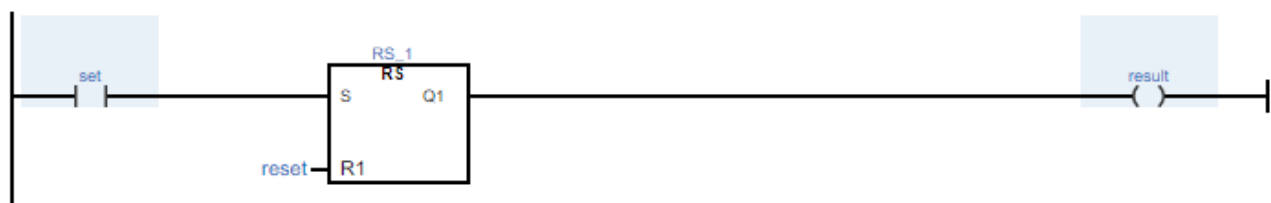
  reset : BOOL;

  result : BOOL;

END_VAR

```

4.1.4.6 Expression in LD



4.1.4.7 Expression in ST

```

RS_1(S := set, R1 := reset);

result := RS_1.Q1;

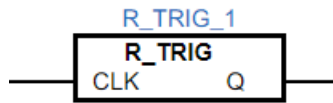
```

4.1.5 R_TRIG Rising Edge Detection Trigger

4.1.5.1 Function Description

This function block is used to detect the rising edge in input signal. Operator is R_TRIG.

4.1.5.2 Graphic



4.1.5.3 Input Parameter

CLK: Signal to be detected, data type is BOOL, Output is TRUE when CLK change from FLASE to TRUE.

4.1.5.4 Output Parameter

Q: Output value, data type is BOOL. Q is TURE when CLK changes from FALSE to TRUE.

4.1.5.5 Variable Declaration

```
VAR
set:BOOL;
Z1:BOOL;
R_TRIG_1:R_TRIG;
END_VAR
```

4.1.5.6 Expression in LD



4.1.5.7 Expression in ST

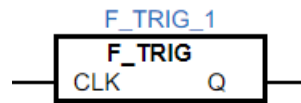
```
R_TRIG_1(CLK := set);
Z1 := R_TRIG_1.Q;
```

4.1.6 F_TRIG Falling Edge Detection Trigger

4.1.6.1 Function Description

This function block is used to detect falling edge in input signal. Operator is F_TRIG.

4.1.6.2 Graphic



4.1.6.3 Input Parameter

CLK: Input signal to be detected, data type is BOOL. Output value is TRUE when CLK changes from TRUE to FALSE.

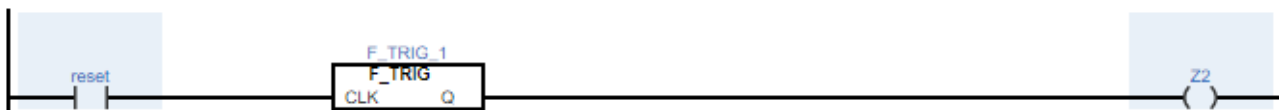
4.1.6.4 Output Parameter

Q: Output value, data type is BOOL, Q is TRUE when CLK changes from TRUE to FALSE.

4.1.6.5 Variable Declaration

```
VAR
reset:BOOL;
Z2:BOOL;
F_TRIG_1:F_TRIG;
END_VAR
```

4.1.6.6 Expression in LD



4.1.6.7 Expression in ST

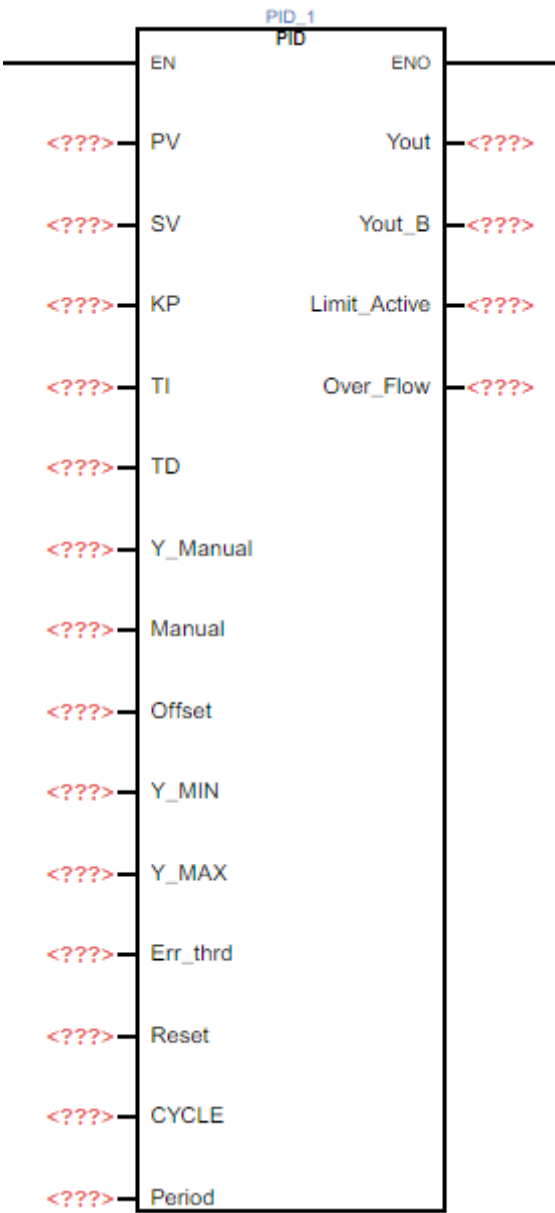
```
F_TRIG_1(CLK := reset);
Z2 := F_TRIG_1.Q;
```

4.1.7 PID Algorithm Control

4.1.7.1 Function Description

This function block is a PID algorithm control instruction. Operator is PID.

4.1.7.2 Graphic



4.1.7.3 Input Parameter

Parameter	Data type	Initial value	Description
EN	BOOL	FALSE	Enable input, when EN is TRUE, this function block is activated.

Parameter	Data type	Initial value	Description
PV	REAL	-	Process variable, calculating result of PID algorithm control is running, actual value.
SV	REAL	-	Set value, target value of the operation.
KP	REAL	-	Percentage, coefficients during PID operations.
TI	REAL	-	Time of Integration.
TD	REAL	-	Time of differentiation.
Y_Manul	REAL	-	PID output value when manual signal MANUAL is TRUE.
Manual	BOOL	FALSE	When the value of Manul is TRUE, the output value is Y_Manul.
Offset	REAL	-	Output offset when not manual.
Y_Min	REAL	-	Limit output minimum.
Y_Max	REAL	-	Limit output maximum.
Err_thrd	REAL	-	Deviation threshold for integral limits, anti-saturation.
Reset	BOOL	FALSE	When it is TRUE, reset function block output Y to Offset, and reset the integral item at the same time..
CYCLE	REAL	-	Sampling cycle time, the unit of which is second.
Period	REAL	-	Output cycle time of Yout, the unit of which is second.

4.1.7.4 Output Parameter

Parameter	Data type	Initial value	Description
ENO	BOOL	FALSE	Enable output, once EN is TRUE, ENO is TRUE.
Yout	REAL	-	PID calculation output value.
Yout_B	BOOL	FALSE	PID calculated output value, duty cycle output signal.
Limit_Active	BOOL	FALSE	It is TRUE when output value is less than minimum or greater than maximum.
Overflow	BOOL	FALSE	Points overflow flag bit, the value is TRUE, it represents an integral overflow.

4.1.7.5 Variable Declaration

```
VAR

PID_1 : PID;

OUT1 : REAL;

OUT2 : REAL;

Manual : BOOL;

Reset : BOOL;

SV : REAL := 10000.0;

KP : REAL := 0.1;

Y_Manual : REAL := 500.0;

offset : REAL := -50.0;

Ymin : REAL := 0.0;

Ymax : REAL := 10000.0;

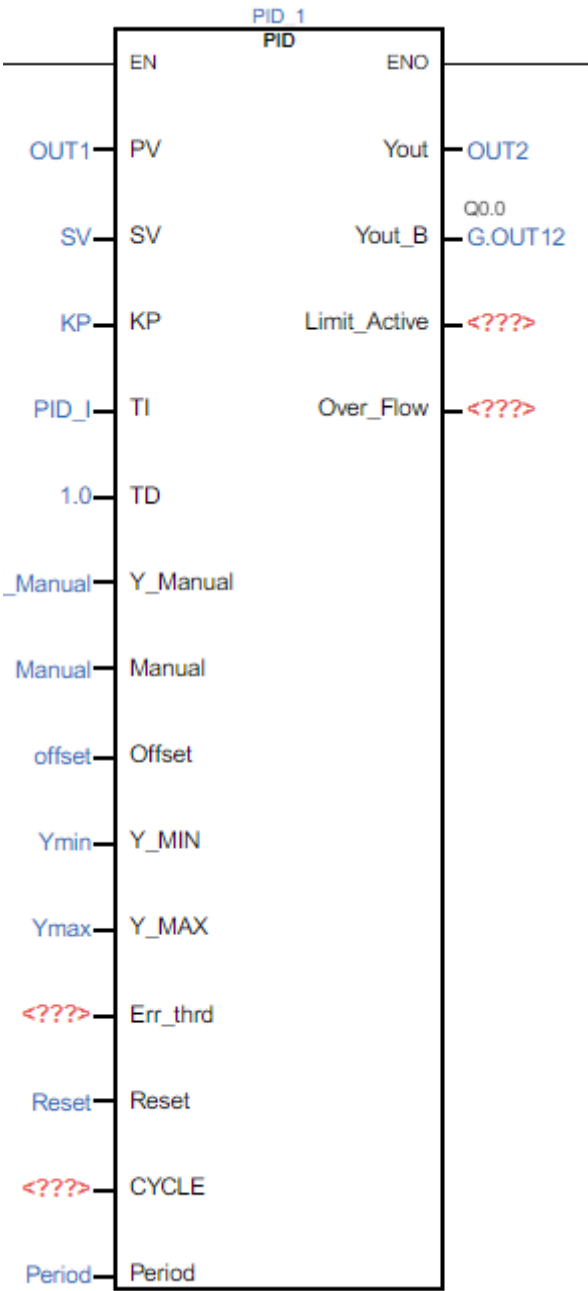
Period : REAL := 1.0;

START : BOOL;

PID_I : REAL;

END_VAR
```

4.1.7.6 Expression in LD



4.1.7.7 Expression in ST

```
PID_1(EN := START, PV := OUT1, SV := SV, KP := KP, TI := PID_I, TD := 1.0, Y_Manual := Y_Manual, Manu
al := Manual, Offset := offset, Y_MIN := Ymin, Y_MAX := Ymax, Reset := Reset, Period := Period);

IF PID_1.ENO THEN

  OUT2 := PID_1.Yout;

  G.OUT12 := PID_1.Yout_B;
```

END_IF;

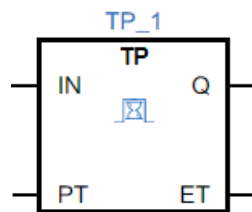
4.2 Timer

4.2.1 TP Timer Pulse

4.2.1.1 Function Description

This function block is used as a timer. Operator is TP.

4.2.1.2 Graphic



4.2.1.3 Definition of Pin

When an rising edge in IN is detected, whether or not the IN remains TRUE, The output ET starts timing with millisecond accuracy. Q outputs TRUE before ET timing reaches the PT timing limit. As long as the ET clock reaches the timing limit PT, Q output is FALSE.

Q output is FALSE if a rising edge is not detected in IN.

4.2.1.4 Input Parameter

Parameter	Data type	Description
IN	BOOL	Detects the signal input of the rising edge to trigger the timer.
PT	TIME	A time quantity used to set the upper limit of the output ET timing.

4.2.1.5 Output Parameter

Parameter	Data type	Description
Q	BOOL	Timer status output, Q outputs TRUE until the ET timing reaches the set PT timing limit.
ET	TIME	The real-time value of the timing, starting from the rising edge of IN.

4.2.1.6 Variable Declaration

VAR

TP_1 : TP;

START : BOOL;

PT_1 : TIME;

Result : BOOL;

ET_1 : TIME;

END_VAR

4.2.1.7 Expression in LD



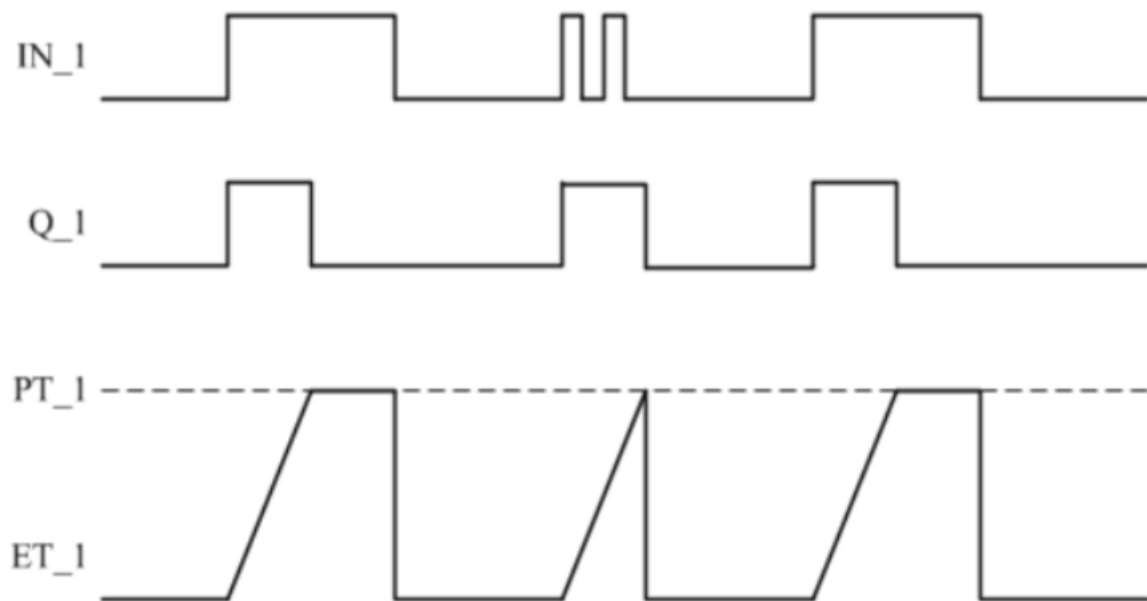
4.2.1.8 Expression in ST

TP_1(IN := Start, PT := PT_1);

ET_1 := TP_1.ET;

Result := TP_1.Q;

4.2.1.9 Sequency Diagram

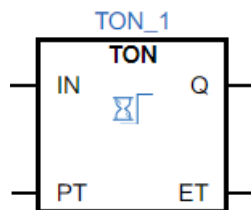


4.2.2 TON Timer On-Delay

4.2.2.1 Function Description

A Timer on-delay usually applies to the following scenarios: After the drive is powered up for a certain amount of time, the output control current. The operator is TON.

4.2.2.2 Graphic



4.2.2.3 Definition of Pins

When the rising edge in IN is detected, output ET starts timing, and the timer state output Q is TRUE only when the input IN remains TRUE and the timing reaches the set time PT.

If the input IN changes from TRUE to FALSE before the timer reaches the set time PT, the timer state output Q is still FALSE.

4.2.2.4 Input Parameter

Parameter	Data type	Description
IN	BOOL	Trigger the timer on-delay when a rising edge is detected in input signal.
PT	TIME	A time quantity used to set the power-on delay time.

4.2.2.5 Output Parameter

Parameter	Data type	Description
Q	BOOL	Timer state output. Q is TRUE when ET reaches the delay time ET.
ET	TIME	This output is a delayed real-time value, time value that starting from the moment that rising edge is detected in IN.

4.2.2.6 Variable Declaration

VAR

TON_1 : TON;

START : BOOL;

PT_1 : TIME;

Result : BOOL;

ET_1 : TIME;

END_VAR

4.2.2.7 Expression in LD



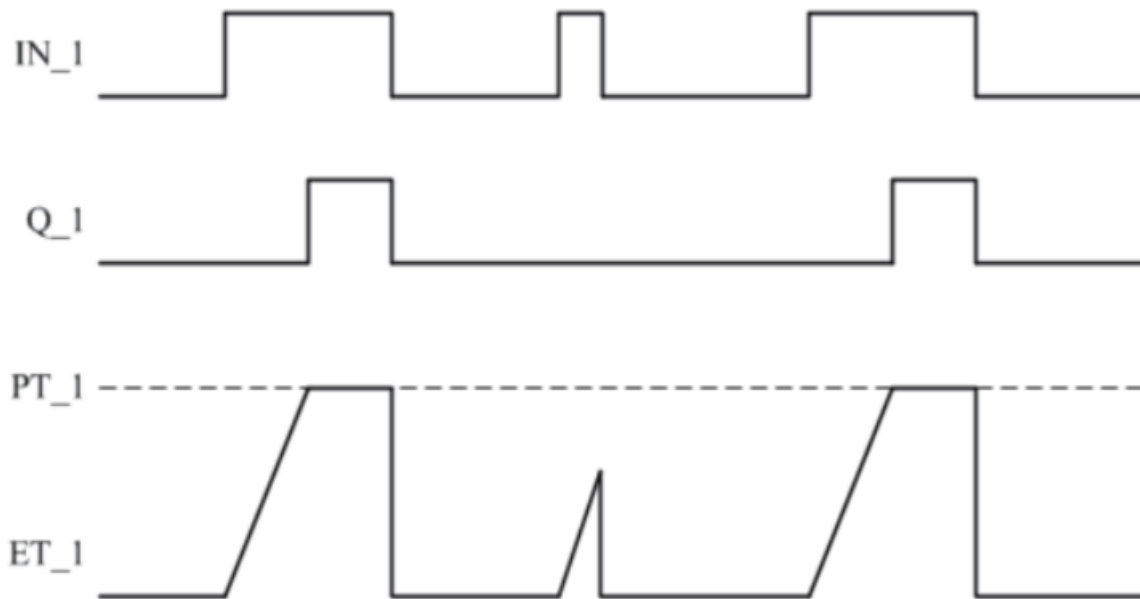
4.2.2.8 Expression in ST

TON_1(IN := Start, PT := PT_1);

ET_1 := TON_1.ET;

Result := TON_1.Q;

4.2.2.9 Sequence Diagram

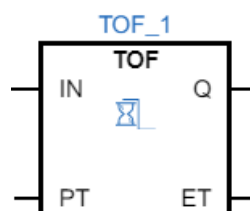


4.2.3 TOF Timer Off-Delay

4.2.3.1 Function Description

A Timer Off-Delay (TOF) is used to extend a certain interval after an OFF (or FALSE) condition and then power off. e.g. the delay of cooling motor. The operator is TOF.

4.2.3.2 Graphic



4.2.3.3 Definition of Pins

When the falling edge in IN is detected, output ET starts timing, and the timer state output Q is FALSE only when the input IN remains FALSE and the timing reaches the set time PT.

If the input IN changes from FALSE to TRUE before the timer reaches the set time PT, the timer state output Q is still TRUE.

4.2.3.4 Input Parameter

Parameter	Data Type	Description
IN	BOOL	Trigger the Timer Off-Delay when a falling edge is detected in IN.
PT	TIME	A time Quantity to set power-off delay time.

4.2.3.5 Output Parameter

Parameter	Data Type	Description
Q	BOOL	Timer status output, Q is FALSE when ET reaches the limit delay time PT.
ET	TIME	This output is a delayed real-time value, time value that starting from the moment that a falling edge is detected in IN.

4.2.3.6 Variable Declaration

VAR

TOF_1 : TOF;

START : BOOL;

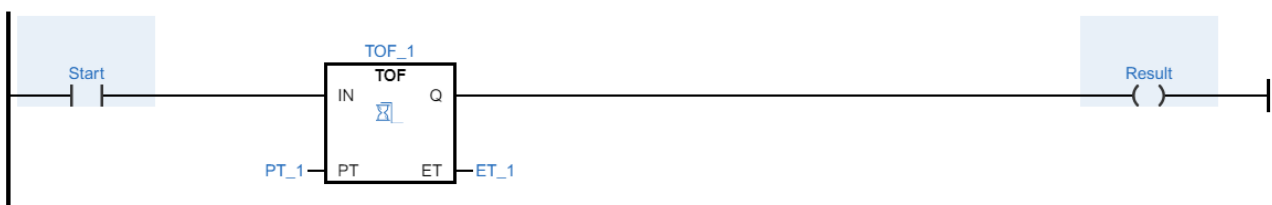
PT_1 : TIME;

Result : BOOL;

ET_1 : TIME;

END_VAR

4.2.3.7 Expression in LD



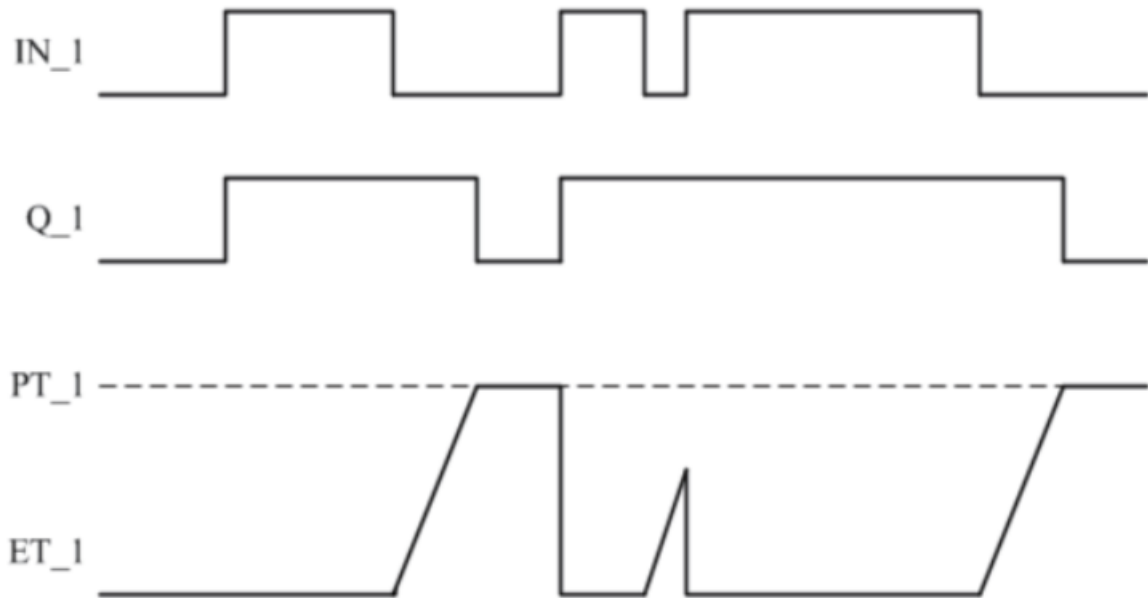
4.2.3.8 Expression in ST

TOF_1(IN := Start, PT := PT_1);

ET_1 := TOF_1.ET;

Result := TOF_1.Q;

4.2.3.9 Sequence Diagram



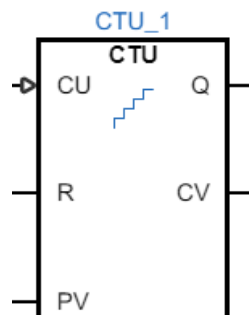
4.3 Couter

4.3.1 CTU Cout-up Counter

4.3.1.1 Function Description

This function clock is used as a Cout-up counter. The operator is CTU.

4.3.1.2 Graphic



4.3.1.3 Definition of Pins

When R is TRUE, regardless of whether the CU detects a rising edge, the count-up counter cannot be triggered and the CV remains 0. When R is FALSE, CV will be added 1 when there is a rising edge from FALSE to TRUE at CU,

and CV will be incremented to the upper limit PV. Meanwhile, Q is TRUE.

4.3.1.4 Input Parameter

Parameter	Data type	Description
CU	BOOL	Trigger the output CV increment when a rising edge is detected in IN.
R	BOOL	Reset CV to 0 when R is TRUE.
PV	ANY-INT	A value to set the upper limit of the output CV.

4.3.1.5 Output Parameter

Parameter	Data type	Description
Q	BOOL	Counter state output, output CV is incremented to the upper count limit PV, then Q output TRUE.
ET	BOOL	This output is the incremental count real-time value, displaying the incremental value from 0 to PV in order according to the rising edge of CU.

4.3.1.6 Variable Declaration

```
VAR
```

```
    CTU_1 : CTU;
```

```
    Start: BOOL;
```

```
    RESET_1 : BOOL;
```

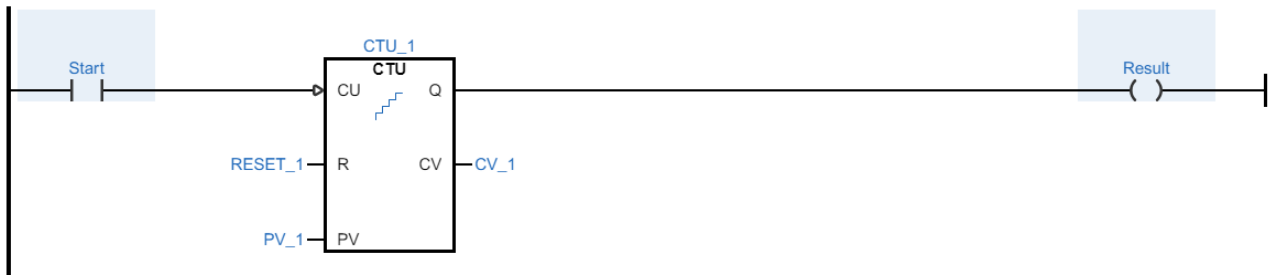
```
    PV_1 : INT;
```

```
    Result : BOOL;
```

```
    CV_1 : INT;
```

```
END_VAR
```

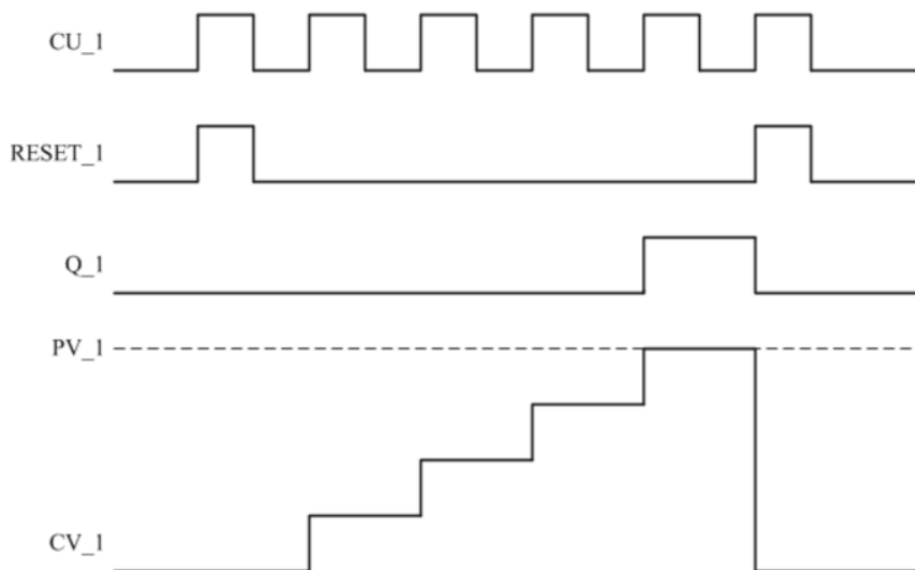
4.3.1.7 Expression in LD



4.3.1.8 Expression in ST

```
R_TRIG1(CLK := Start);  
  
CTU_1(CU := R_TRIG1.Q, R := RESET_1, PV := PV_1);  
  
CV_1 := CTU_1.CV;  
  
Result := CTU_1.Q;
```

4.3.1.9 Sequence Diagram



4.3.1.10 Usage Restriction

Variable data types of V and CV must be consistent and available data types are as follows:

- ◆ USINT
- ◆ UINT
- ◆ UDINT
- ◆ ULINT
- ◆ SINT

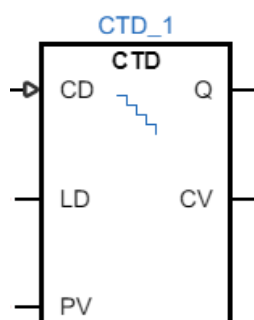
- ◆ INT
- ◆ DINT
- ◆ LINT

4.3.2 CTD Cout-down Couter

4.3.2.1 Function Description

This function block is used as a count-down counter. The operator is CTD.

4.3.2.2 Graphic



4.3.2.3 Definition of Pins

When LD is TRUE, regardless of whether CD detects a rising edge, it cannot trigger the decrement count, and CV maintains the upper PV value of decrement count; when LD is FALSE, when there is a rising edge from FALSE to TRUE at CD, if CV is greater than 0, CV will be decremented by 1, and when CV is equal to 0, Q output will be TRUE.

4.3.2.4 Input Parameter

Parameter	Data type	Description
CD	BOOL	Trigger the output CV decrement when a rising edge is detected.
LD	BOOL	To reset the counter, if TRUE the CV is reset to 0.
PV	ANY_INT	A value to set the upper limit of the output CV decrement count.

4.3.2.5 Output Parameter

Parameter	Data type	Description
Q	BOOL	Q is TRUE when output parameter CV decreases to 0.

Parameter	Data type	Description
CV	ANY_INT	This output is the decremental count real time value, in accordance with the rising edge of the CD in order to display the decreasing value from PV to 0.

4.3.2.6 Variable Declaration

VAR

CTD_1 : CTD;

Start: BOOL;

RESET_1 : BOOL;

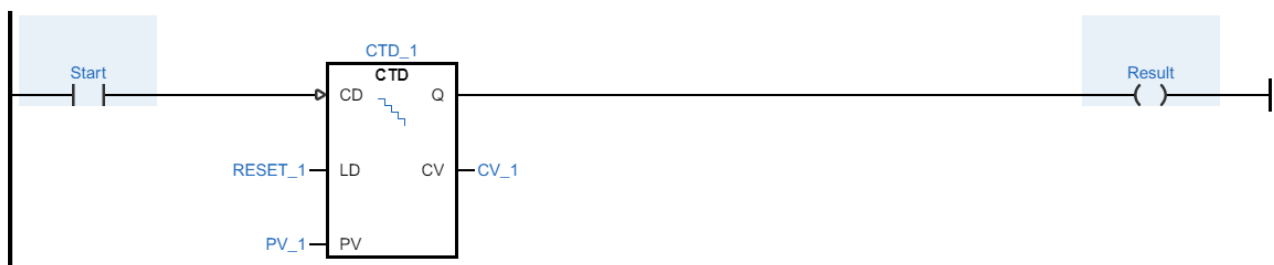
PV_1 : INT;

Result : BOOL;

CV_1 : INT;

END_VAR

4.3.2.7 Expression in LD



4.3.2.8 Expression in ST

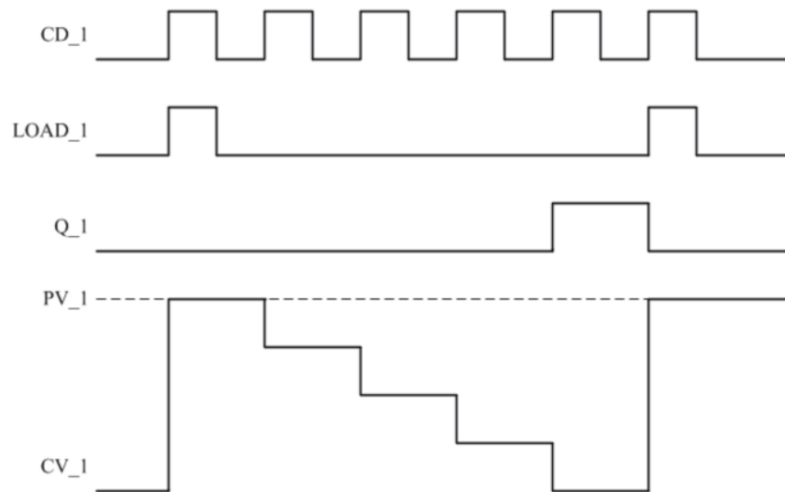
R_TRIG1(CLK := Start);

CTD_1(CD := R_TRIG1.Q, LD := RESET_1, PV := PV_1);

CV_1 := CTD_1.CV;

Result := CTD_1.Q;

4.3.2.9 Sequence Diagram



4.3.2.10 Usage Restriction

Data types of PV and CV must be consistent and available data types are as follows:

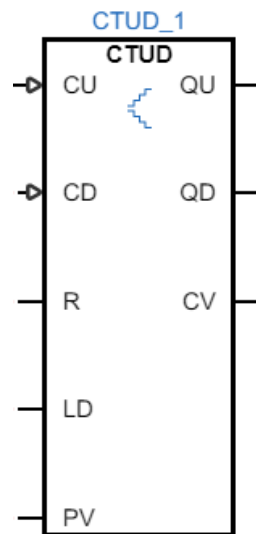
- ◆ USINT
- ◆ UINT
- ◆ UDINT
- ◆ ULINT
- ◆ SINT
- ◆ INT
- ◆ DINT
- ◆ LINT

4.3.3 CTUD Count up/down Counter

4.3.3.1 Function Description

This function block is used as an count up/down counter. The operator is CTUD.

4.3.3.2 Graphic



4.3.3.3 Definition of Pins

When R is TRUE, the counter cannot be triggered regardless of whether the CU or CD detects a rising edge, and the CV remains 0.

When LD is TRUE, the counter cannot be triggered regardless of whether CU or CD detects a rising edge, and CV keeps the decreasing count upper limit PV value.

4.3.3.4 Input Parameter

Parameter	Data type	Description
CU	BOOL	Trigger CV increment when a rising edge is detected in CU.
CD	BOOL	Trigger CV decrement when a rising edge is detected in CD.
R	BOOL	To reset count-up counter, if R is TRUE, reset CV to 0.
LD	BOOL	To reset count-down counter, if LD is TRUE, reset CV to the upper limit.
PV	ANY_INT	A value that sets the upper count limit for the decreasing or decreasing output CV.

4.3.3.5 Output Parameter

Parameter	Data type	Description
QU	BOOL	Counter state output, QU is TRUE when CV increases to the upper counter limit PV.

Parameter	Data type	Description
QD	BOOL	Couter state output, QD is TRUE when CV decreases to 0.
CV	ANY_INT	Count real-time value, adds 1 when detects a rising edge in CD, decreases 1 when detects a rising edge in CU.

4.3.3.6 Variable Declaration

VAR

CTUD_1: CTUD;

Start: BOOL;

CD_1: BOOL;

RESET_1: BOOL;

LOAD_1: BOOL;

PV_1: INT;

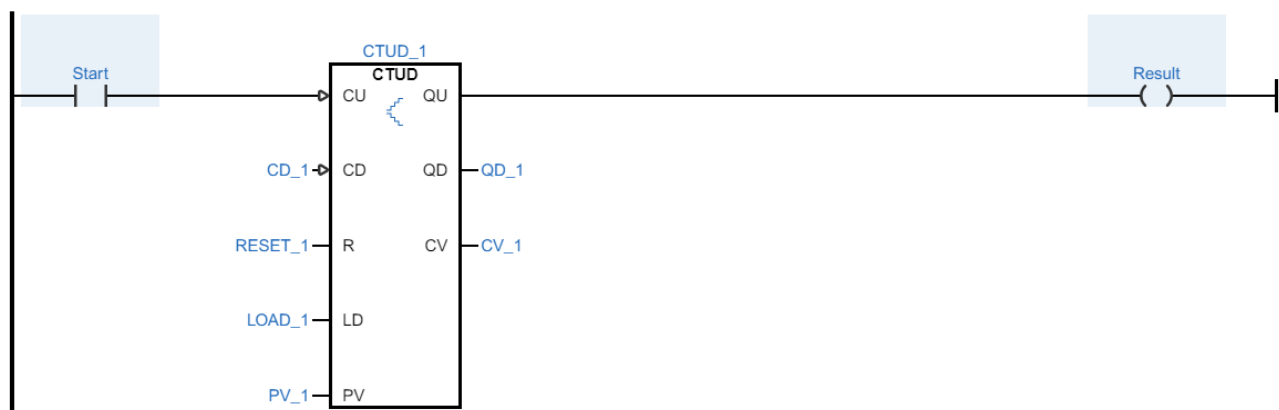
Result: BOOL;

QD_1: BOOL;

CV_1: INT;

END_VAR

4.3.3.7 Expression in LD



4.3.3.8 Expression in ST

R_TRIG1(CLK := Start);

```

R_TRIG2(CLK := CD_1);

CTUD_1(CU := R_TRIG1.Q, CD := R_TRIG2.Q, R := RESET_1, LD := LOAD_1, PV := PV_1);

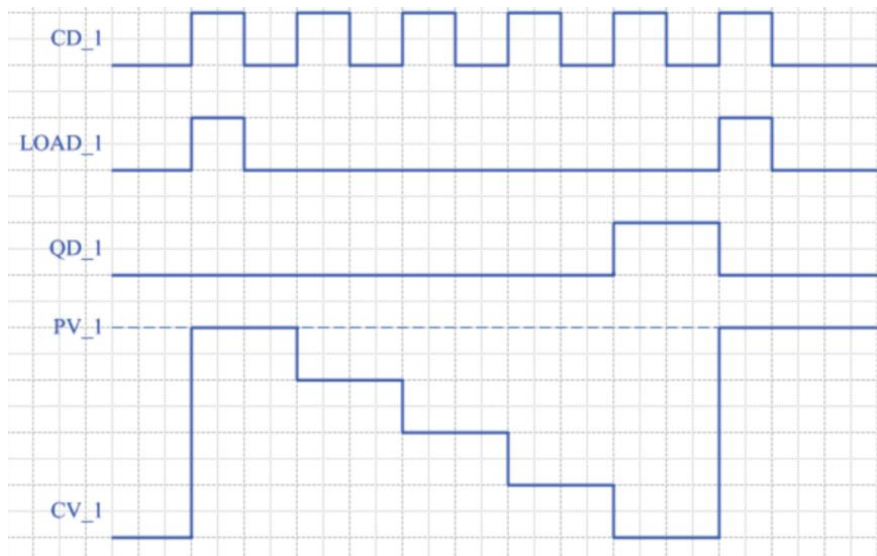
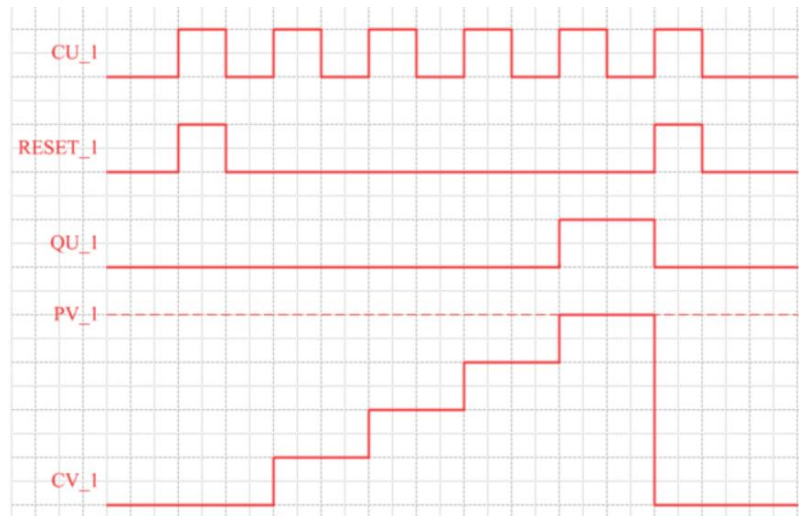
QD_1 := CTUD_1.QD;

CV_1 := CTUD_1.CV;

Result := CTUD_1.QU;

```

4.3.3.9 Sequence Diagram



4.3.3.10 Usage Restriction

The data type of PV and CV must be consistent, available data types is as follows:

- ◆ USINT

- ◆ UINT
- ◆ UDINT
- ◆ ULINT
- ◆ SINT
- ◆ INT
- ◆ DINT
- ◆ LINT

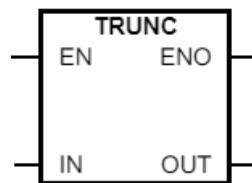
4.4 Type conversion

4.4.1 TRUNC Truncation

4.4.1.1 Function Description

This function is used to convert the REAL type to INT type, which will use the integer part of the value. The operator is TRUNC.

4.4.1.2 Graphic



4.4.1.3 Input Parameter

Parameter	Data type	Description
EN	BOOL	Enable input.
IN	ANY_REAL	Input value.

4.4.1.4 Output Parameter

Parameter	Data type	Description
ENO	BOOL	Enable output.
OUT	ANY_INT	Converted integer value.

4.4.1.5 Variable Declaration

VAR

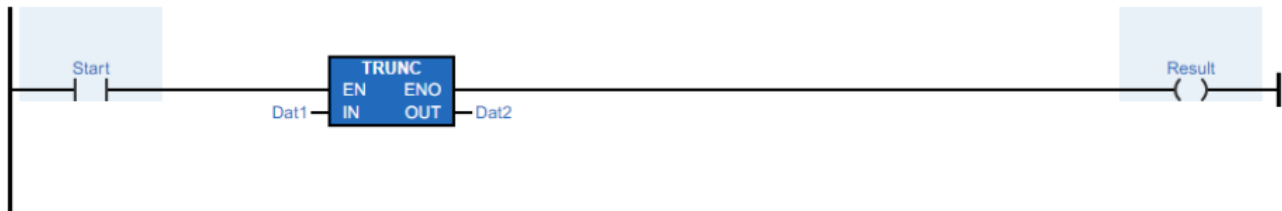
```
Start : BOOL;
```

```
Dat1 : REAL;
```

```
Dat2 : INT;
```

```
END_VAR
```

4.4.1.6 Expression in LD



4.4.1.7 Expression in ST

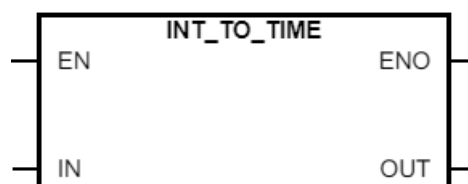
```
Dat2:=TRUNC(Dat1);
```

4.4.2 *_TO_* Data Type Conversion

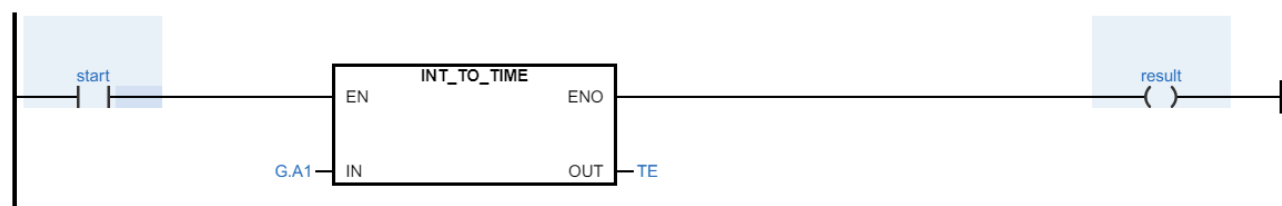
4.4.2.1 Function Description

This function is used to convert data of one type to data of the other type. The operator is `*_TO_*` (e.g. `INT_TO_TIME`).

4.4.2.2 Graphic



4.4.2.3 Expression in LD



When this Function is called, **Adjust FunctionBlock Pin** dialog-box pops up, select the **Input Type** and **Output**

Type to be converted.

Adjust FunctionBlock Pin

Search

⊕ BASE ELEMENT

⊕ TIMER

⊕ COUNTER

⊖ TYPECONVERT

— TRUNC

— *_TO_*

⊕ MATHEMATICS FUNCTION

⊕ ARITHMETIC FUNCTION

⊕ LEFT/RIGHT BITSHIFT

⊕ BITWISE

⊕ SELECTION

⊕ COMPARISON

First Input Pin : EN

First Output Pin : ENO

InputType : INT

* OutputType : TIME

Instruct Width : Widen

Cancel

Confirm

4.4.2.4 Expression in ST

```
TE:=INT_TO_TIME(G.A1);
```

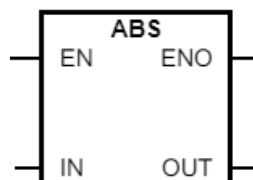
4.5 Mathematical Function

4.5.1 ABS Absolute Value

4.5.1.1 Function Description

This function calculates the absolute value of the input value and assigns the result to the output. The operator is ABS.

4.5.1.2 Graphic



4.5.1.3 Input Parameter

IN: Input value(ANY_NUM).

4.5.1.4 Output Parameter

OUT: Output value, consistent with the data type of the input value.

4.5.1.5 Variable Declaration

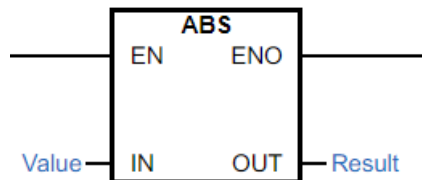
```
VAR
```

```
    Value: INT;
```

```
    Result: INT;
```

```
END_VAR
```

4.5.1.6 Expression in LD



4.5.1.7 Expression in ST

```
Result := ABS(Value);
```

4.5.1.8 Usage Restriction

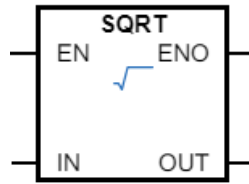
- ◆ Available data type includes REAL, LREAL, SINT, INT, DINT, LINT, USINT, UINT, UDINT, and ULINT.
- ◆ The data type of output parameter must be consistent with the data type of input parameter.

4.5.2 SQRT Square Root

4.5.2.1 Function Description

This function calculates the square root value of the input value and assigns the result to the output. Standard IEC operator SQRT.

4.5.2.2 Graphic



4.5.2.3 Input Parameter

IN: Input value, data type of it is REAL or LREAL.

4.5.2.4 Output Parameter

OUT: Output value, data type of it is REAL or LREAL.

4.5.2.5 Variable Declaration

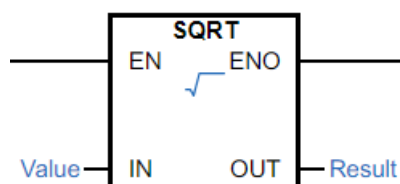
```
VAR
```

```
    Value: REAL;
```

```
    Result: REAL;
```

```
END_VAR
```

4.5.2.6 Expression in LD



4.5.2.7 Expression in ST

```
Result := Sqrt(Value);
```

4.5.2.8 Usage Restriction

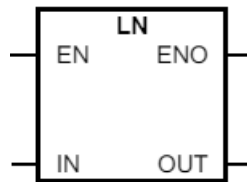
- ◆ Available data type includes REAL and LREAL.
- ◆ Input value can not be less than 0.
- ◆ The data type of input value must be consistent with the data type of output value.

4.5.3 LN Natural Logarithm

4.5.3.1 Function Description

This function calculates the natural logarithm of the input value (with the constant e as the base). Standard IEC operator LN.

4.5.3.2 Graphic



4.5.3.3 Input Parameter

IN: Input value, data type of it is REAL or LREAL.

4.5.3.4 Output Parameter

OUT: Output value, data type of it is REAL or LREAL.

4.5.3.5 Variable Declaration

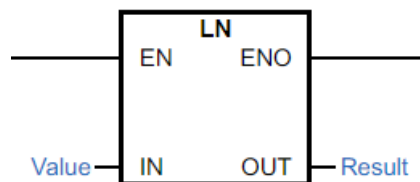
```
VAR
```

```
    Value: REAL
```

```
    Result: REAL;
```

```
END_VAR
```

4.5.3.6 Expression in LD



4.5.3.7 Expression in ST

```
Result:=LN(Value);
```

4.5.3.8 Usage Restriction

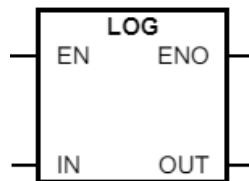
- ◆ Available data type includes REAL and LREAL.
- ◆ Input value can not be less than 0.
- ◆ The data type of output value must be consistent with the data type of input value.

4.5.4 LOG Logarithm with Base 10

4.5.4.1 Function Description

The function calculates the common logarithm of the input value (with base of 10) and assigns the result to the output variable. IT is a standard IEC operator LOG.

4.5.4.2 Graphic



4.5.4.3 Input Parameter

IN: Input value, data type of which is REAL OR LREAL.

4.5.4.4 Output Parameter

OUT: Output value, data type of which is REAL or LREAL.

4.5.4.5 Variable Declaration

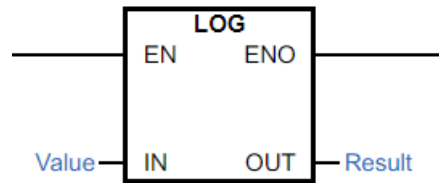
```
VAR
```

```
    Value: REAL
```

```
    Result: REAL;
```

```
END_VAR
```

4.5.4.6 Expression in LD



4.5.4.7 Expression in ST

```
Result:=LOG(Value);
```

4.5.4.8 Usage Restriction

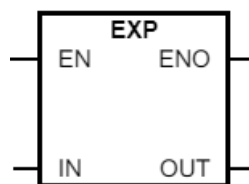
- ◆ Available data type includes REAL and LREAL.
- ◆ Input value can not be less than 0.
- ◆ The data type of output value must be consistent with the data type of the input value.

4.5.5 EXP Exponential Instruction

4.5.5.1 Function Description

This function calculates the power using natural constant e as the base and input Value as exponent. It is a standard IEC operator EXP.

4.5.5.2 Graphic



4.5.5.3 Input Parameter

IN: Input value, data type of which is REAL or LREAL.

4.5.5.4 Output Parameter

OUT: Output value, data type of which is REAL or LREAL.

4.5.5.5 Variable Declaration

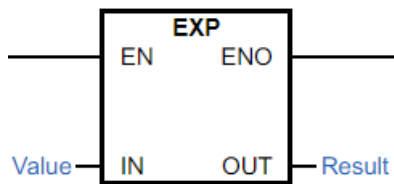
VAR

Value: REAL;

Result: REAL;

END_VAR

4.5.5.6 Expression in LD



4.5.5.7 Expression in ST

Result:=EXP(Value);

4.5.5.8 Usage Restriction

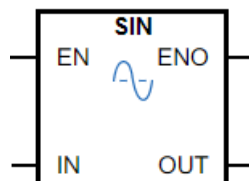
- ◆ Available data type includes REAL and LREAL.
- ◆ Input value can not be less than 0.
- ◆ The data type of output value must be consistent with the data type of input value.

4.5.6 SIN Sine

4.5.6.1 Function Description

This function calculates the sine of the input value; expressed in radians. Standard IEC operator SIN.

4.5.6.2 Graphic



4.5.6.3 Input Parameter

IN: Input value, data type of which is REAL or LREAL.

4.5.6.4 Output Parameter

OUT: Output value, data type of which is REAL or LREAL.

4.5.6.5 Variable Declaration

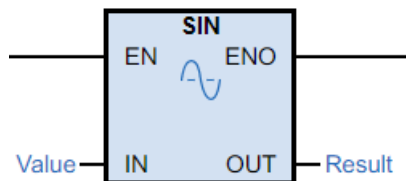
VAR

Value: REAL

Result: REAL;

END_VAR

4.5.6.6 Expression in LD



4.5.6.7 Expression in ST

```
Result:=SIN(Value);
```

4.5.6.8 Usage Restriction

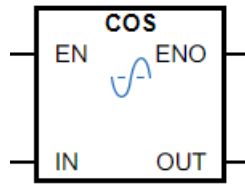
- ◆ Available data type includes REAL and LREAL.
- ◆ The data type of output value must be consistent with the data type of input value.

4.5.7 COS Cosine

4.5.7.1 Function Description

This function calculates the cosine of the input value; expressed in radians. It is a standard IEC operator COS.

4.5.7.2 Graphic



4.5.7.3 Input Parameter

IN: Input value, data type of which is REAL or LREAL.

4.5.7.4 Output Parameter

OUT: Output value, data type of which is REAL or LREAL.

4.5.7.5 Variable Declaration

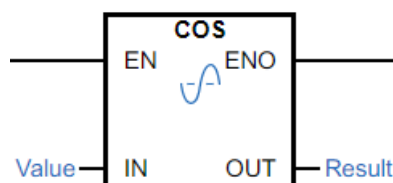
VAR

Value: REAL

Result: REAL;

END_VAR

4.5.7.6 Expression in LD



4.5.7.7 Expression in ST

Result:=COS(Value);

4.5.7.8 Usage Restriction

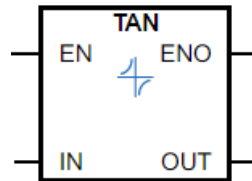
- ◆ Available data type includes REAL and LREAL.
- ◆ The data type of output value must be consistent with the data type of the input value.

4.5.8 TAN Tangent

4.5.8.1 Function Description

This function calculates the tangent of the input value; expressed in radians. Standard IEC operator TAN.

4.5.8.2 Graphic



4.5.8.3 Input Parameter

IN: Input value, of which the data type is REAL or LREAL.

4.5.8.4 Output Parameter

OUT: Output value, of which the data type is REAL or LREAL.

4.5.8.5 Variable Declaration

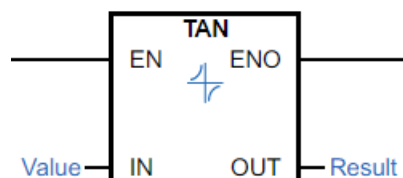
```
VAR
```

```
    Value: REAL
```

```
    Result: REAL;
```

```
END_VAR
```

4.5.8.6 Expression in LD



4.5.8.7 Expression in ST

```
Result:=TAN(Value);
```

4.5.8.8 Usage Restriction

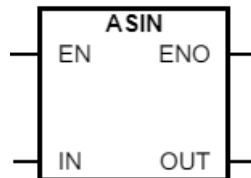
- ◆ Available data type includes REAL and LREAL.
- ◆ Input value cannot be $(\pi/2+k*\pi)$.
- ◆ The data type of output value must be consistent with the data type of input value.

4.5.9 ASIN Arc Sine

4.5.9.1 Function Description

This function calculates the arc sine of the input value, the inverse function of the sine function; expressed in radians. It is a standard IEC operator ASIN.

4.5.9.2 Graphic



4.5.9.3 Input Parameter

IN: Input value, of which the data type is REAL or LREAL.

4.5.9.4 Output Parameter

OUT: Output value, of which the data type is REAL or LREAL.

4.5.9.5 Variable Declaration

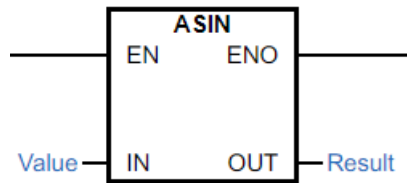
```
VAR
```

```
    Value: REAL
```

```
    Result: REAL;
```

```
END_VAR
```

4.5.9.6 Expression in LD



4.5.9.7 Expression in ST

```
Result:=ASIN(Value);
```

4.5.9.8 Usage Restriction

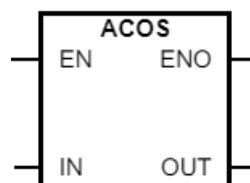
- ◆ Available data type includes REAL and LREAL.
- ◆ The range of input value is [-1,1].
- ◆ The data type of output value must be consistent with the data type of input value.

4.5.10 ACOS Arc Cosine

4.5.10.1 Function Description

This function calculates the arc cosine of the input value, the inverse function of the cosine function; expressed in radians. It is a standard IEC operator ACOS.

4.5.10.2 Graphic



4.5.10.3 Input Parameter

IN: Input value, of which the data type is REAL or LREAL.

4.5.10.4 Output Parameter

OUT: Output value, of which the data type is REAL or LREAL.

4.5.10.5 Variable Declaration

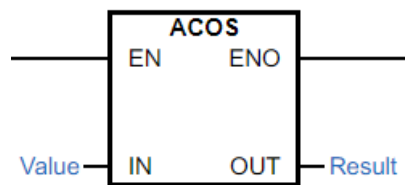
VAR

Value: REAL

Result: REAL;

END_VAR

4.5.10.6 Expression in LD



4.5.10.7 Expression in ST

```
Result:=ACOS(Value);
```

4.5.10.8 Usage Restriction

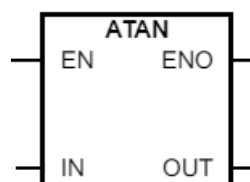
- ◆ Available data type includes REAL and LREAL.
- ◆ The range of input value is $[-1,1]$.
- ◆ The data type of output value must be consistent with the data type of input value.

4.5.11 ATAN Arc Tangent

4.5.11.1 Function Description

This function calculates the arc tangent of the input value, the inverse function of the tangent function; expressed in radians. Standard IEC operator ATAN.

4.5.11.2 Graphic



4.5.11.3 Input Parameter

IN: Input value, of which the data type is REAL or LREAL.

4.5.11.4 Output Parameter

OUT: Output value, of which the data type is REAL or LREAL.

4.5.11.5 Variable Declaration

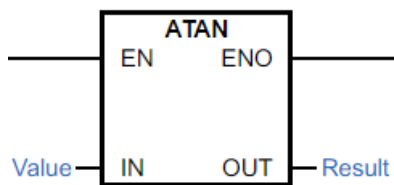
```
VAR
```

```
    Value: REAL
```

```
    Result: REAL;
```

```
END_VAR
```

4.5.11.6 Expression in LD



4.5.11.7 Expression in ST

```
Result:=ATAN(Value);
```

4.5.11.8 Usage Restriction

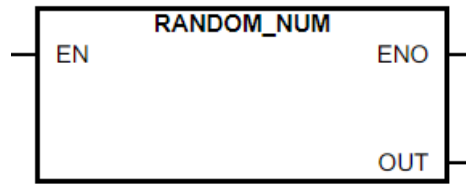
- ◆ Available data type includes REAL and LREAL.
- ◆ The data type of output value must be consistent with the data type of input value.

4.5.12 RANDOM_NUM Getting a Random Positive Integer

4.5.12.1 Function Description

This function is used to get a random positive integer.

4.5.12.2 Graphic



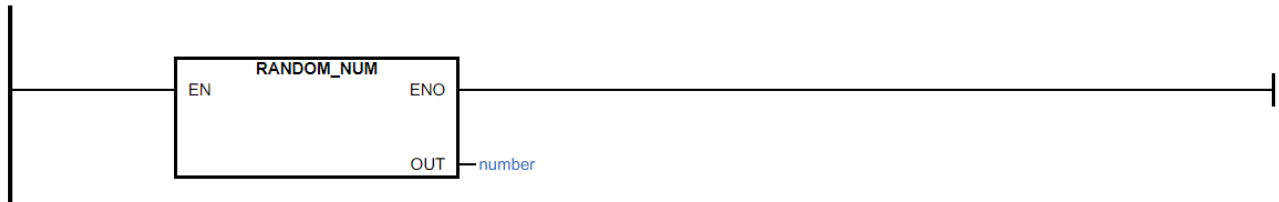
4.5.12.3 Output Parameter

OUT: Output value, of which the data type is UDINT.

4.5.12.4 Variable Declaration

```
VAR  
number:UDINT;  
END_VAR
```

4.5.12.5 Expression in LD



4.5.12.6 Expression in ST

```
number:=RANDOM_NUM;
```

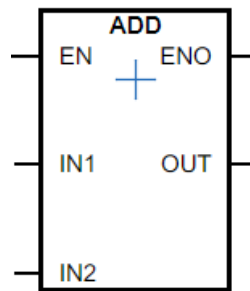
4.6 Arithmetic Function

4.6.1 ADD Addition

4.6.1.1 Function Description

This function is used as an addition of variables or constants. The operator is ADD.

4.6.1.2 Graphic



4.6.1.3 Definition of Pins

When EN is TRUE, the ADD function is activated and ENO output is TRUE; after adding IN1 and IN2, the result is assigned to OUT.

4.6.1.4 Input Parameter

Parameter	Data type	Description
EN	BOOL	Enable input, when it is TRUE, this function is activated.
IN1	Refer to Usage Restriction	Addend 1
IN2	Refer to Usage Restriction	Addend 2

4.6.1.5 Output Parameter

Parameter	Data type	Description
ENO	BOOL	Enable output.
OUT	Refer to Usage Restriction	sum, i.e., the result of summing the input values.

4.6.1.6 Variable Declaration

VAR

```
Start : BOOL;
```

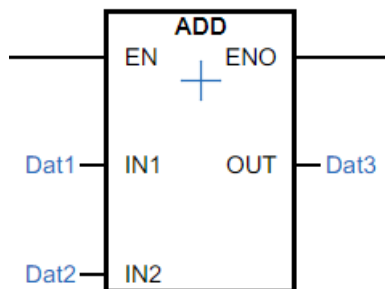
```
Dat1 : INT;
```

```
Dat2 : INT;
```

```
Dat3 : INT;
```

```
END_VAR
```

4.6.1.7 Expression in LD



4.6.1.8 Expression in ST

```
Dat3 := ADD(Dat1, Dat2);
```

4.6.1.9 Usage Restriction

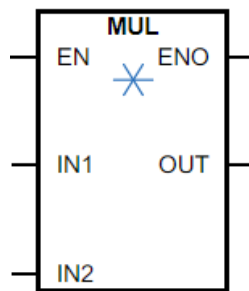
- ◆ Available data type for IN1, IN2 and OUT includes REAL, LREAL, SINT, INT, DINT, LINT, USINT, UINT, UDINT, and ULINT.
- ◆ The data type of IN2 must be consistent with the data type of IN1.

4.6.2 MUL Multiplication

4.6.2.1 Function Description

This function block is used as a multiplication of variables or constants. The operator is MUL.

4.6.2.2 Graphic



4.6.2.3 Definition of Pins

When EN is TRUE, the MUL function is activated and ENO output is TRUE; after multiply IN1, IN2, the result is assigned to OUT.

4.6.2.4 Input Parameter

Parameter	Data type	Description
EN	BOOL	Enable input, when EN is TURE, this function is activated.
IN1	Refer to Usage Restriction	Multiplier 1.
IN2	Refer to Usage Restriction	Multiplier 2.

4.6.2.5 Output Parameter

Parameter	Data type	Description
ENO	BOOL	Enable output, once EN is TURE, ENO is TRUE.
OUT	Refer to Usage Restriction	Product, i.e., the result of multiplying the input variables.

4.6.2.6 Variable Declaration

VAR

Start : BOOL;

Dat1 : INT;

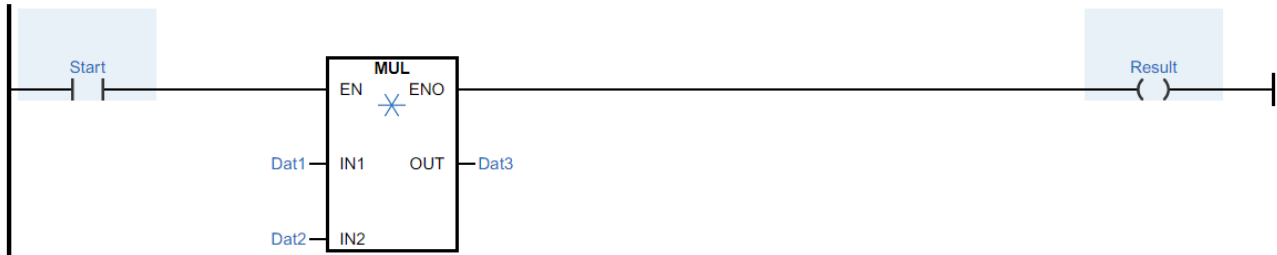
Dat2 : INT;

```
Dat3 : INT;

Result : BOOL;
```

```
END_VAR
```

4.6.2.7 Expression in LD



4.6.2.8 Expression in ST

```
Dat3 := MUL(Dat1, Dat2);
```

4.6.2.9 Usage Restriction

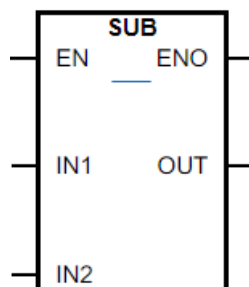
- ◆ Available data type for IN1, IN2, and OUT includes REAL, LREAL, SINT, INT, DINT, LINT, USIN, UINT, UDINT, and ULINT.
- ◆ Please set IN1, IN2 to the same data type, otherwise the compilation will generate a data type mismatch error.

4.6.3 SUB Subtraction

4.6.3.1 Function Description

This function is used as a subtraction of variables or constants. The operator is SUB.

4.6.3.2 Graphic



4.6.3.3 Definition of Pins

When EN is TRUE, the SUB function is activated and ENO output is TRUE; after subtracting IN2 from IN1, the result is assigned to OUT.

4.6.3.4 Input Parameter

Parameter	Data type	Description
EN	BOOL	Enable input, when EN is TRUE, this function is activated.
IN1	Refer to Usage Restriction .	Minuend
IN2	Refer to Usage Restriction .	Subtrahend

4.6.3.5 Output Parameter

Parameter	Data type	Description
ENO	BOOL	Enable output, once EN is TRUE, ENO is TRUE.
OUT	Refer to Usage Restriction .	Difference, i.e., the result of subtracting the subtrahend from the minuend.

4.6.3.6 Variable Declaration

VAR

Start : BOOL;

Dat1 : INT;

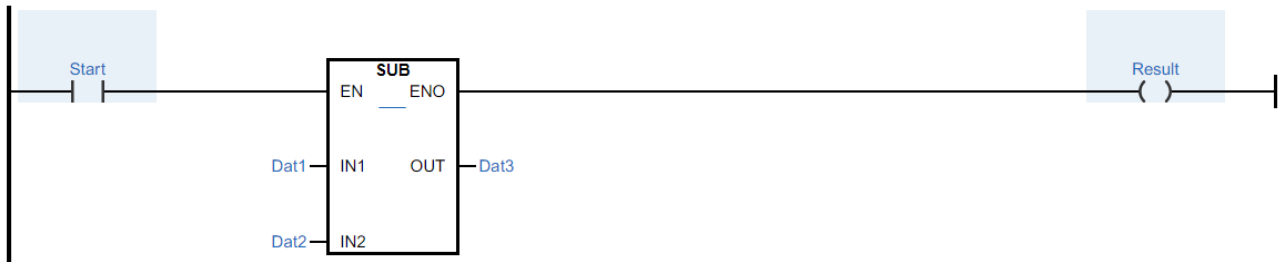
Dat2 : INT;

Dat3 : INT;

Result : BOOL;

END_VAR

4.6.3.7 Expression in LD



4.6.3.8 Expression in ST

```
Dat3 := SUB(Dat1, Dat2);
```

4.6.3.9 Usage Restriction

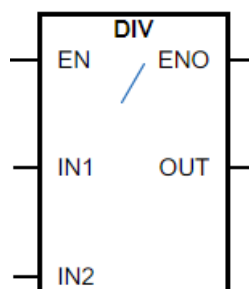
- ◆ Available data type for IN1, IN2, and OUT includes REAL, LREAL, SINT, INT, DINT, LINT, USINT, UINT, UDINT, and ULINT.
- ◆ Please set IN1, IN2 to the same data type, otherwise the compilation will generate a data type mismatch error.

4.6.4 DIV Division

4.6.4.1 Function Description

This function block is used to divide variables or constants. Its operator is DIV.

4.6.4.2 Graphic



4.6.4.3 Definition of Pins

When EN is TRUE, the DIV function is activated and the ENO is TRUE; IN1 divided by IN2, assigning its results to OUT.

4.6.4.4 Input Parameter

Parameter	Data type	Description
EN	BOOL	Enable input, when EN is TRUE, this function is activated.
IN1	Refer to Usage Restriction	Dividend
IN2	Refer to Usage Restriction	Divisor

4.6.4.5 Output Parameter

Parameter	Data type	Description
ENO	BOOL	Enable output, once EN is TRUE, ENO is TRUE.
OUT	Refer to Usage Restriction	Quotients, i.e., the result of dividing the dividend by the divisor.

4.6.4.6 Variable Declaration

VAR

Start : BOOL;

Dat1 : INT;

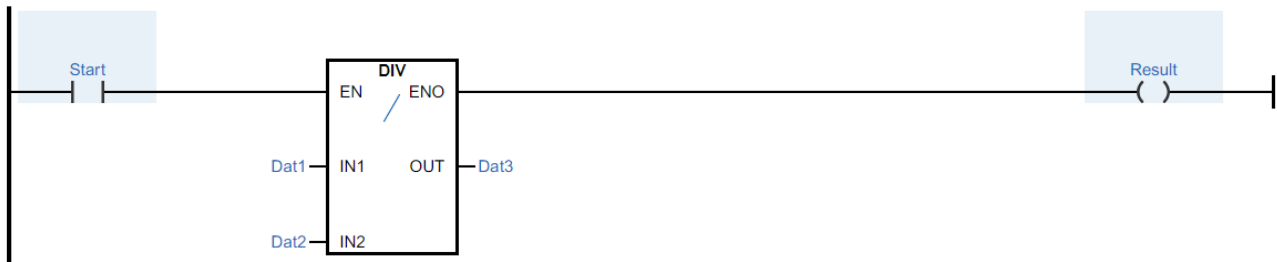
Dat2 : INT

Dat3 : INT

Result : BOOL;

END_VAR

4.6.4.7 Expression in LD



4.6.4.8 Expression in ST

```
Dat3 := DIV(Dat1, Dat2);
```

4.6.4.9 Usage Restriction

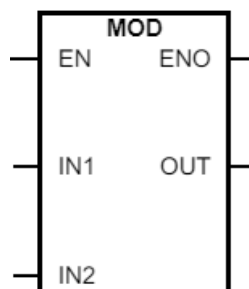
- ◆ Available data type for IN1, IN2, and OUT includes REAL, LREAL, SINT, INT, DINT, LINT, USINT, UINT, UDINT, and ULINT.
- ◆ Please set IN1, IN2 to the same data type, otherwise the compilation will generate a data type mismatch error.
- ◆ IN2 can not be 0.

4.6.5 MOD Modulus

4.6.5.1 Function Description

This function block is used to divide the value of IN1 by the value of IN2 to get the remainder. Its operator is MOD.

4.6.5.2 Graphic



4.6.5.3 Definition of Pins

When EN is TRUE, the MOD function is activated and ENO output is TRUE; EN should be kept TRUE when MOD function is executed continuously; after dividing the dividend IN1 by the divisor IN2, the remainder of the result is assigned to OUT.

4.6.5.4 Input Parameter

Parameter	Data type	Description
EN	BOOL	Enable input, when EN is TRUE, this function is activated.
IN1	Refer to Usage Restriction	Dividend
IN2	Refer to Usage Restriction	Divisor

4.6.5.5 Output Parameter

Parameter	Data type	Description
ENO	BOOL	Enable output, once EN is TRUE, ENO is TRUE.
OUT	Refer to Usage Restriction	Modulus, i.e., the remainder after division.

4.6.5.6 Variable Declaration

VAR

Start : BOOL;

Dat1 : INT;

Dat2 : INT;

Dat3 : INT;

Result : BOOL;

END_VAR

4.6.5.7 Expression in LD



4.6.5.8 Expression in ST

```
Dat3 := MOD(Dat1, Dat2);
```

4.6.5.9 Usage Restriction

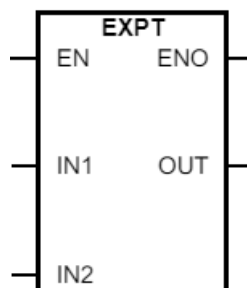
- ◆ Available data type for IN1, IN2, and OUT includes SINT, INT, DINT, LINT, USINT, UINT, UDINT, and ULINT.
- ◆ Please set IN1, IN2, OUT to the same data type, otherwise the compilation will generate a data type mismatch error.

4.6.6 EXPT Exponent

4.6.6.1 Function Description

This function is a power calculation with Value1 as the base and Value2 as the exponent. IT is a standard IEC operator EXPT.

4.6.6.2 Graphic



4.6.6.3 Input Parameter

Parameter	Data type	Description
EN	BOOL	Enable input, when EN is TRUE, this function is activated.
IN1	REAL	Base

Parameter	Data type	Description
IN2	REAL	Exponent

4.6.6.4 Output Parameter

Parameter	Data type	Description
ENO	BOOL	Enable output, once EN is TURE, ENO is TRUE.
OUT	REAL	Power

4.6.6.5 Variable Declaration

VAR

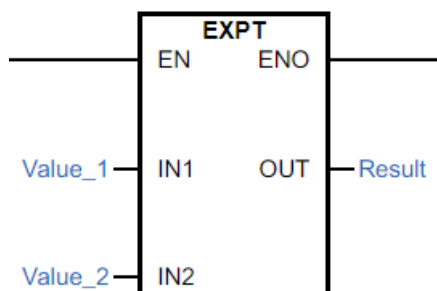
Value_1: REAL;

Value_2: INT;

Result: REAL;

END_VAR

4.6.6.6 Expression in LD



4.6.6.7 Expression in ST

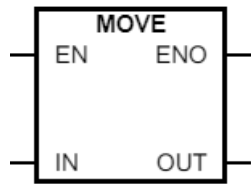
Result := EXPT(Value_1, Value_2);

4.6.7 MOVE Assignment

4.6.7.1 Function Description

This function is used to assign the value of a constant or variable to another variable. The operator is MOVE.

4.6.7.2 Graphic



4.6.7.3 Definition of Pins

When EN is TRUE, the MOVE function is activated and ENO output is TRUE ; transfer data from IN to OUT.

4.6.7.3.1 Input Parameter

Parameter	Data type	Description
EN	BOOL	Enable input, when EN is TRUE, this function is activated.
IN	Refer to Usage Restriction	Variable 1

4.6.7.3.2 Output Parameter

Parameter	Data type	Description
ENO	BOOL	Enable output, once EN is TRUE, ENO is TRUE.
OUT	Consistent with the data type of IN	Variable 2

4.6.7.4 Variable Declaration

```
VAR  
  
    START : BOOL;  
  
    Dat1 : INT;  
  
    Dat2 : INT;  
  
END_VAR
```

4.6.7.5 Expression in LD



4.6.7.6 Expression in ST

Dat2 := Dat1;

4.6.7.7 Usage Restriction

- ◆ Available data type for IN and OUT includes BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL, TIME, DT, BOOL, STRING, and ARRAY.
- ◆ The data type of OUT must be consistent with the data type of IN.

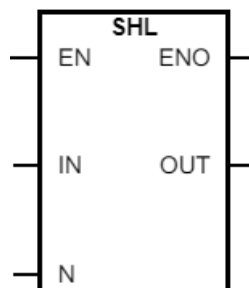
4.7 Left/Right Bitwise Shift

4.7.1 SHL Shift Left

4.7.1.1 Function Description

This function block is used to shift the operand by bit to the left, the bit shifted out on the left is not processed and the right is automatically filled with 0.

4.7.1.2 Graphic



4.7.1.3 Definition of Pins

When EN is TRUE, function SHL is activated, ENO output is TRUE; The result of shifting the operand IN left by the bits of N is assigned to OUT.

4.7.1.3.1 Input Parameter

Parameter	Data type	Description
EN	BOOL	Enable input, when EN is TURE, this function is activated.
IN	BYTE, WORD, LWORD, or DWORD	Operand
N	ANY_INT	Number of bitwise left shift

4.7.1.3.2 Output Parameter

Parameter	Data type	Description
ENO	BOOL	Enable output, once EN is TRUE, ENO is TRUE.
OUT	Same data type as IN	Result of a bitwise left shift of the operand

4.7.1.4 Variable Declaration

VAR

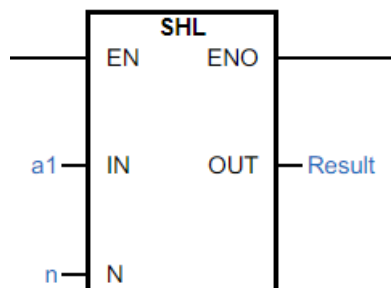
a1:WORD;

n:INT;

Result:WORD;

END_VAR

4.7.1.5 Expression in LD



4.7.1.6 Expression in ST

```
Result:=SHL( a1, n);
```

4.7.1.7 Usage Restriction

The data type of OUT must be consistent with the data type of IN.

4.7.1.8 Cautions

In the above example, although the value of the input variable is the same, the result obtained is different because the size of the output data type is different. for example:

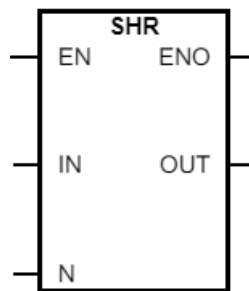
```
Var3:=SHL(16#45,2); (*Var3: BYTE; result Var3 is 16#14*) Var3:=SHL(16#45,2); (*Var3: WORD;  
result Var3 is 16#0114*)
```

4.7.2 SHR Shift Right

4.7.2.1 Function Description

This function is used to shift the operand by bit to the right, the bit shifted out on the right is not processed and the left is automatically filled with 0.

4.7.2.2 Graphic



4.7.2.3 Definition of Pins

When EN is TRUE, function SHR is activated, ENO output is TRUE; The result of shifting the operand IN right by the bits of N is assigned to OUT.

4.7.2.3.1 Input Parameter

Parameter	Data type	Description
EN	BOOL	Enable input, when EN is TRUE, this function is activated.
IN	BYTE, WORD, DWORD, or LWORD	Operand
N	ANY_INT	Number of bitwise right shift.

4.7.2.3.2 Output Parameter

Parameter	Data type	Description
ENO	BOOL	Enable output, once EN is TRUE, ENO is TRUE.
OUT	same data type as IN	Result of a bitwise right shift of the operand

4.7.2.4 Variable Declaration

VAR

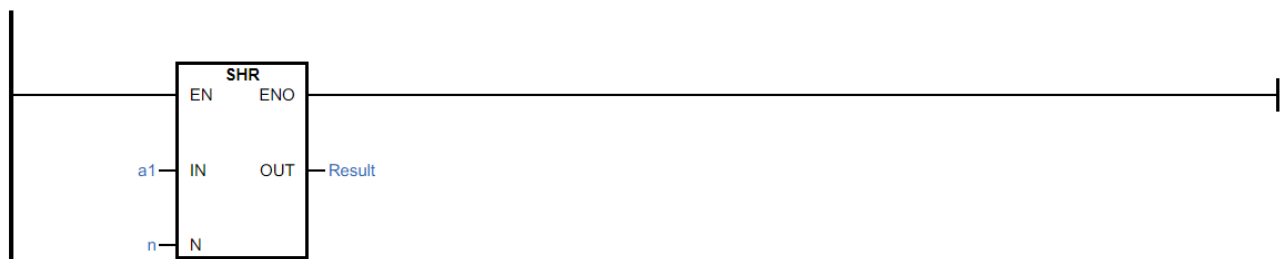
a1:WORD;

n:INT;

Result:WORD;

END_VAR

4.7.2.5 Expression in LD



4.7.2.6 Expression in ST

Result:=SHR(a1,n);

4.7.2.7 Usage Restriction

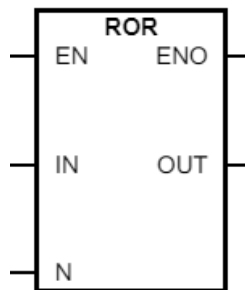
The data type of OUT must be consistent with the data type of IN.

4.7.3 ROR Rotation on Right

4.7.3.1 Function Description

This function is used to shift the operand by bitwise left rotation , with the bit shifted out on the right being added directly to the highest bit on the left. The operator is ROR.

4.7.3.2 Graphic



4.7.3.3 Definition of Pins

When EN is TRUE, function ROR is activated, ENO output is TRUE; The result of rotating the operand IN right by the bits of N is assigned to OUT.

4.7.3.3.1 Input Parameter

Parameter	Data type	Description
EN	BOOL	Enable input, when EN is TRUE, this function is activated.
IN	BYTE, WORD, DWORD, or LWORD	Operand
N	ANY_INT	Bitwise right rotation of the operand

4.7.3.3.2 Output Parameter

Parameter	Data type	Description
ENO	BOOL	Enable output, once EN is TRUE, ENO is TURE.

Parameter	Data type	Description
OUT	Same data type as IN	Result of a bitwise right rotation of the operand

4.7.3.4 Variable Declaration

VAR

EN_1 : BOOL;

L1_Var1:WORD;

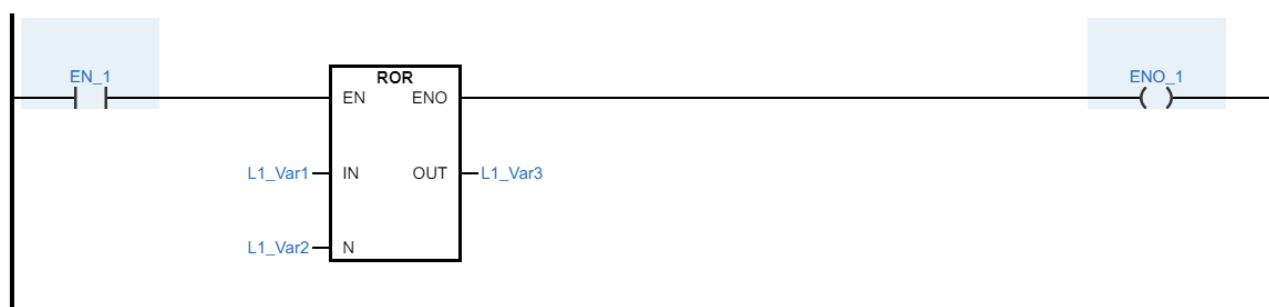
L1_Var2:INT;

ENO_1:BOOL;

L1_Var3: WORD;

END_VAR

4.7.3.5 Expression in LD



4.7.3.6 Expression in ST

```
L1_Var3:=ROR(EN:=EN_1,IN:=L1_Var1,N:=L1_Var2,ENO=>ENO_1);
```

4.7.3.7 Cautions

Although the value of the input variable is the same, the result obtained is different because the size of the output data type is different. for example:

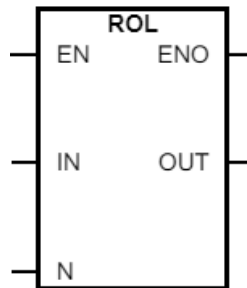
```
Var3:=ROR(16#45,2); (*Var3 : BYTE; result of Var3 is 16#51*) Var3:=ROR(16#45,2);
(*Var3: WORD; result of Var3 is 16#4011*)
```


4.7.4 ROL Rotation on Left

4.7.4.1 Function Description

This function block is used to shift the operand by bitwise left rotation, with the bit shifted out on the left being added directly to the lowest bit on the right. The operator is ROL.

4.7.4.2 Graphic



4.7.4.3 Definition of Pins

When EN is TRUE, function ROL is activated, ENO output is TRUE; The result of rotating the operand IN left by the bits of N is assigned to OUT.

4.7.4.3.1 Input Parameter

Parameter	Data type	Description
EN	BOOL	Enable input, when EN is TRUE, this function is activated.
IN	BYTE, WORD, DWORD, or LWORD	Operand
N	ANY_INT	Bitwise left rotation of the operand

- ◆ EN: Function enable, of which the data type is BOOL, the function ROL is activated when EN is TRUE.
- ◆ IN: Operand, of which the data type is BYTE, WORD, DWORD, or LWORD.
- ◆ N: bitwise left rotation of the operand, of which the data type is ANY_INT(such as INT, SINT, UINT etc.).

4.7.4.3.2 Output Parameter

Parameter	Data type	Description
ENO	BOOL	Enable output, once EN is TRUE, ENO is TRUE.

Parameter	Data type	Description
OUT	Same data type as IN	Result of a bitwise left rotation of the operand

4.7.4.4 Variable Declaration

VAR

EN_1 : BOOL;

L1_Var1:WORD;

L1_Var2:INT;

ENO_1:BOOL;

L1_Var3:WORD;

END_VAR

4.7.4.5 Expression in LD



4.7.4.6 Expression in ST

```
L1_Var3:=ROL(EN:=EN_1,IN:=L1_Var1,N:=L1_Var2,ENO=>ENO_1);
```

4.7.4.7 Cautions

Although the value of the input variable is the same, the result obtained is different because the size of the output data type is different. For example:

```
Var3:=ROL(16#45,2); (*Var3: BYTE; result of Var3 is 16#15*)
```

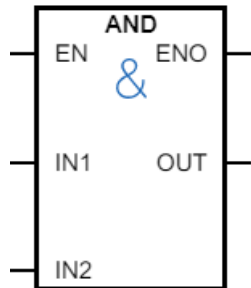
```
Var3:=ROL(16#45,2); (*Var3: WORD; result of Var3 is 16#0114*)
```

4.7.5 AND Bitwise AND

4.7.5.1 Function Description

This function is used as an “AND” operation of variables or constants. When the corresponding input bits of both operands are all 1, the output bits are also 1, otherwise 0. The operator is AND.

4.7.5.2 Graphic



4.7.5.3 Definition of Pins

When EN is TRUE, the AND function is activated and ENO outputs TRUE; IN1 and IN2 perform the AND operation and assign the result to OUT.

4.7.5.3.1 Input Parameter

Parameter	Data type	Description
EN	BOOL	Enable input, when EN is TRUE, this function is activated.
IN1	BOOL, BYTE, WORD, DWORD, or LWORD	Operand 1
IN2	the same data type as IN1	Operand 2

4.7.5.3.2 Output Parameter

Parameter	Data type	Description
ENO	BOOL	Enable output, once EN is TRUE, ENO is TRUE.
OUT	the same data	The result of the AND operation of IN1 and IN2

Parameter	Data type	Description
	type as IN1	

4.7.5.3 Example

IN1 is True, IN2 is False, OUT is False;

IN1 is 2#10010011, IN2 is 2#10001010, OUT is 2#10000010.

4.7.5.4 Variable Declaration

VAR

START : BOOL;

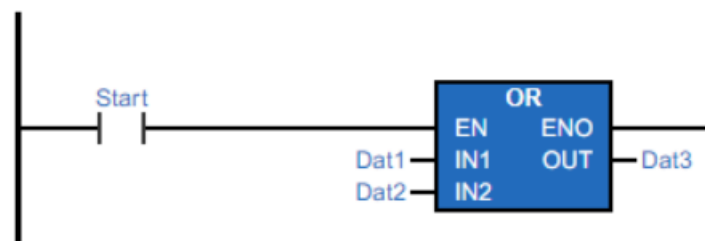
Dat1 : BOOL;

Dat2 : BOOL;

Dat3 : BOOL;

END_VAR

4.7.5.5 Expression in LD



4.7.5.6 Expression in ST

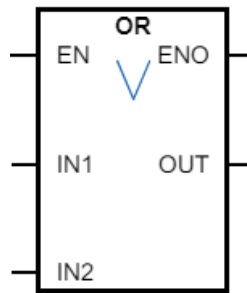
Dat3 := Dat1 OR Dat2;

4.7.6 OR Bitwise OR

4.7.6.1 Function Description

This function block is used as an “OR” operation of variables or constants. The operator is OR.

4.7.6.2 Graphic



4.7.6.3 Definition of Pins

When EN is TRUE, the OR function is activated and ENO outputs TRUE; IN1 and IN2 perform the OR operation and assign the result to OUT.

4.7.6.3.1 Input Parameter

Parameter	Data type	Description
EN	BOOL	Enable input, when EN is TRUE, this function is activated.
IN1	BOOL, BYTE, WORD, DWORD, or LWORD	Operand 1
IN2	The same data type as IN1	Operand 2

4.7.6.3.2 Output Parameter

Parameter	Data type	Description
ENO	BOOL	Enable output, once EN is TRUE, ENO is TRUE.
OUT	The same data type as IN1	The result of the OR operation of IN1 and IN2

4.7.6.3.3 Example

IN1 is True, IN2 is False, OUT is True;

IN1 is 2#10010011, IN2 is 2#10001010, OUT is 2#10011011.

4.7.6.4 Variable Declaration

VAR

START : BOOL;

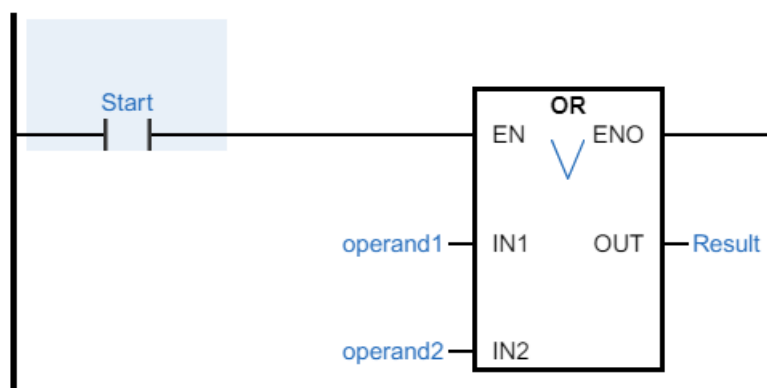
operand1 : BOOL;

operand2 : BOOL;

Result : BOOL;

END_VAR

4.7.6.5 Expression in LD



4.7.6.6 Expression in ST

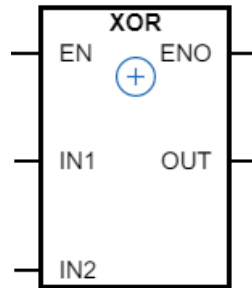
```
Result := operand1 OR operand2;
```

4.7.7 XOR Bitwise XOR

4.7.7.1 Function Description

This function is used as an “XOR” operation of variables or constants. The operator is XOR.

4.7.7.2 Graphic



4.7.7.3 Definition of Pins

When EN is TRUE, the XOR function is activated and ENO outputs TRUE; IN1 and IN2 perform the XOR operation and assign the result to OUT.

4.7.7.3.1 Input Parameter

Parameter	Data type	Description
EN	BOOL	Enable input, when EN is TRUE, this function is activated.
IN1	BOOL, BYTE, WORD, DWORD, or LWORD	Operand 1
IN2	The same data type as IN1	Operand 2

4.7.7.3.2 Output Parameter

Parameter	Data type	Description
ENO	BOOL	Enable output, once EN is TRUE, ENO is TRUE.
OUT	The same data type as IN1	The result of the XOR operation of IN1 and IN2

4.7.7.3.3 Example

- ◆ IN1 is TRUE, IN2 is FALSE, OUT is TRUE; IN1 is TRUE, IN2 is TRUE, OUT is FALSE.
- ◆ IN1 is 2#10010011, IN2 is 2#10001010, OUT is 2#00011001.

4.7.7.4 Variable Declaration

VAR

START : BOOL;

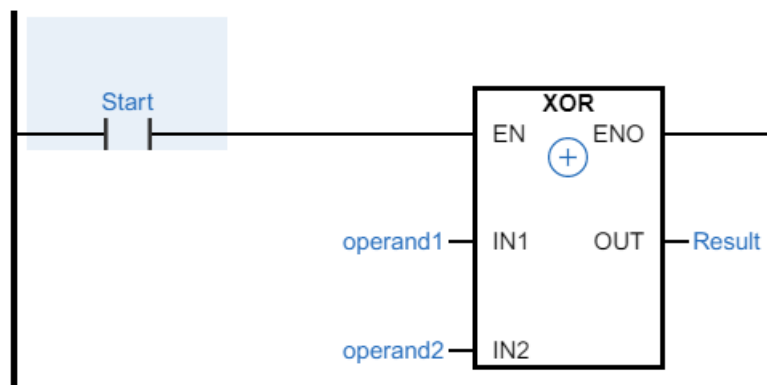
operand1 : BOOL;

operand2 : BOOL;

Result : BOOL;

END_VAR

4.7.7.5 Expression in LD



4.7.7.6 Expression in ST

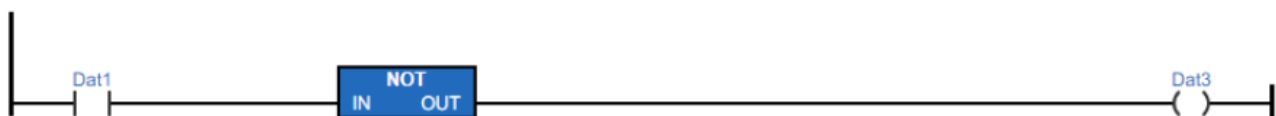
```
Result := operand1 XOR operand2;
```

4.7.8 NOT Bitwise NOT

4.7.8.1 Function Description

This function is used as a “NOT” operation of variables or constants. The operator is NOT.

4.7.8.2 Graphic



4.7.8.3 Definition of Pins

IN performs NOT operation and assigns its result to OUT.

4.7.8.3.1 Input Parameter

IN: operand, of which the data type is BOOL, BYTE, WORD, DWORD, or LWORD. Double-click the function, set First Input Pin/ First Output Pin as EN/ENO to use the numeric type variable.

4.7.8.3.2 Output Parameter

OUT: Result of IN after “NOT” operation, same data type as IN.

4.7.8.3.3 Example

IN is TRUE, OUT is FALSE.

IN is 2#10010011, OUT is 2#01101100.

4.7.8.4 Variable Declaration

```
VAR  
  
    Dat1:BOOL;  
  
    Dat3:BOOL;  
  
END_VAR;
```

4.7.8.5 Expression in LD



4.7.8.6 Expression in ST

```
Dat3 := NOT(Dat1);
```

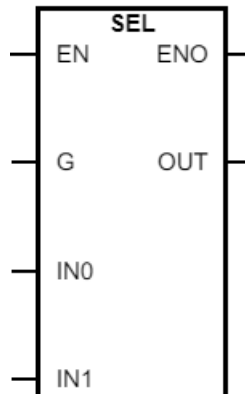
4.8 Selection

4.8.1 SEL Either or

4.8.1.1 Function Description

This function is used to select an input variable as an output from two input variables. The operator is SEL.

4.8.1.2 Graphic



4.8.1.3 Input Parameter

Parameter	Data type	Description
EN	BOOL	Enable input, when EN is TRUE, this function is activated.
G	BOOL	Selection signal, when it is TRUE, OUT is IN0;when it is FALSE, OUT is IN1.
IN0	Any basic data type	Candidate variable 1 for the either or choice
IN1	Any basic data type	Candidate variable 2 for the either or choice

4.8.1.4 Output Parameter

Parameter	Data type	Description
ENO	BOOL	Enable output, once EN is TRUE, ENO is TRUE.
OUT	Any basic data type	Selection result, one of the two candidate variables will be selected as output.

4.8.1.5 Variable Declaration

```
VAR  
  
  G_1: BOOL;  
  
  Dat_IN0: INT;  
  
  Dat_IN1: INT;  
  
  Dat_OUT: INT;  
  
END_VAR
```

4.8.1.6 Expression in LD



4.8.1.7 Expression in ST

```
Dat_OUT :=SEL(G_1,Dat_IN0,Dat_IN1);
```

4.8.1.8 Cautions

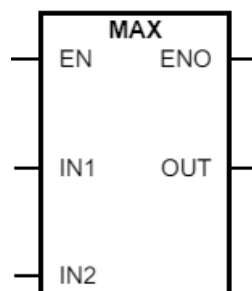
- ◆ It is recommended to set IN0, IN1, OUT to the same data type to avoid errors in data conversion.
- ◆ The variable types of IN0 and IN1 on the Input side may be different, but must be a comparable data type, e.g. IN0 is BYTE, IN1 is Real, Out is Real is allowed; e.g. IN0 is BOOL (its value is False or True), IN1 is Real, Out is Real is not allowed, because Bool types can not be compared with integers or floating point numbers. In addition, the variable type on the output side must be one that covers the length of the variable on the input side, e.g. if the variables on the Input side are BYTE and INT respectively, then the variable type on the output side must be at least INT, and it can not be BYTE (refer to the [Data Type of Variable](#) for a description of variable data type lengths). Whether the value of the parameter G on the input side is selected as False or True, the output side will have the output.

4.8.2 MAX Maximum

4.8.2.1 Function Description

This function is used to select the largest number from the values of two or more input-side variables as the output value. The operator is MAX.

4.8.2.2 Graphic



4.8.2.3 Input Parameter

Parameter	Data type	Description
EN	BOOL	Enable input, when EN is TRUE, this function is activated.
IN1	Any basic data type	Comparison value 1
IN0	Any basic data type	Comparison value 2

4.8.2.4 Output Parameter

Parameter	Data type	Description
ENO	BOOL	Enable output, once EN is TRUE, ENO is TRUE.
OUT	Any basic data type	The maximum in the values being compared

4.8.2.5 Variable Declaration

VAR

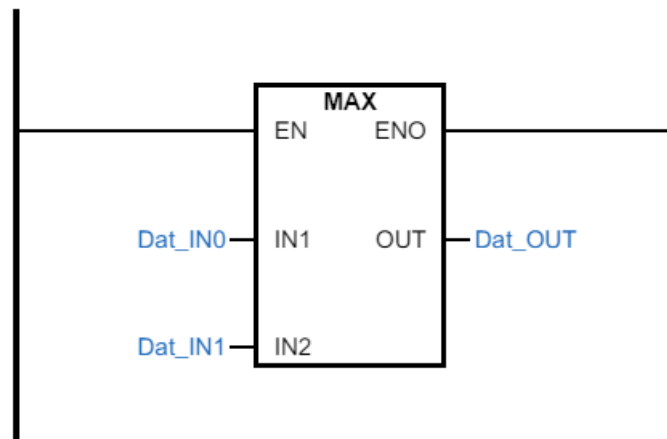
Dat_IN0: INT;

```
Dat_IN1: INT;

Dat_OUT: INT;

END_VAR
```

4.8.2.6 Expression in LD



4.8.2.7 Expression in ST

```
Dat_OUT :=MAX(Dat_IN0,Dat_IN1);
```

4.8.2.8 Usage Restriction

- ◆ The maximum number of inputs should not exceed 480, otherwise it will cause the software to exit abnormally during compilation. To ensure the normal operation of the program, it is recommended that the number of data involved in the comparison should not exceed 300.
- ◆ It is recommended that IN0, IN1, OUT be set to the same data type to avoid errors and mistakes in data conversion.

4.8.2.9 Cautions

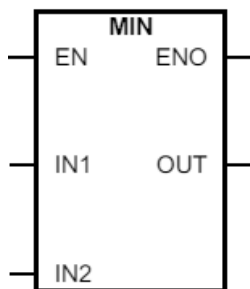
- ◆ The number of data to be compared can be more than two, in the LD language, right-click the function, select **Add Pin**, you can add pins, increase the number of variables to be compared.
- ◆ Multiple input variables can be of different types, but they must be comparable data types, e.g. one input pin is BYTE, another input pin is Real and Out is Real is allowed; e.g. one Input pin is BOOL (its value is FALSE or TRUE) and another input pin is Real, Output variable is Real is not allowed, because Bool type is not comparable with integers or floating point numbers. If the input types are different, the output variable type must be one that can cover the length of the input variable, e.g. if the inputs are BYTE and INT respectively, the output variable type must be at least INT and it can not be BYTE (refer to [Data Type of Variable](#) for a description of variable data type lengths).

4.8.3 MIN Minimum

4.8.3.1 Function Description

The function is used to select the minimum as the output value from the values of two or more input variables. The operator is MIN.

4.8.3.2 Graphic



4.8.3.3 Input Parameter

Parameter	Data type	Description
EN	BOOL	Enable input, when EN is TRUE, this function is activated.
IN1	Any basic data type	Comparison value 1
IN0	Any basic data type	Comparison value 2

4.8.3.4 Output Parameter

Parameter	Data type	Description
ENO	BOOL	Enable output, once EN is TRUE, ENO is TRUE.
OUT	Any basic data type	The minimum in the values being compared

4.8.3.5 Variable Declaration

VAR

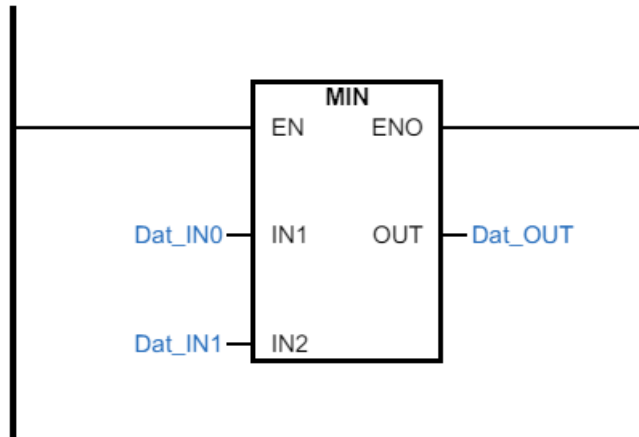
Dat_IN0: INT;

```
Dat_IN1: INT;

Dat_OUT: INT;

END_VAR
```

4.8.3.6 Expression in LD



4.8.3.7 Expression in ST

```
Dat_OUT := MIN(Dat_IN0, Dat_IN1);
```

4.8.3.8 Usage Restriction

- ◆ The maximum number of inputs should not exceed 480, otherwise it will cause the software to exit abnormally during compilation. To ensure the normal operation of the program, it is recommended that the number of data involved in the comparison should not exceed 300.
- ◆ It is recommended that IN0, IN1, OUT be set to the same data type to avoid errors and mistakes in data conversion.

4.8.3.9 Cautions

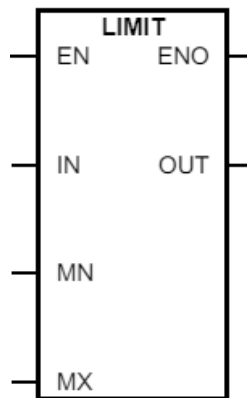
- ◆ The number of data to be compared can be more than two, in the LD language, right-click the function, select **Add Pin**, you can add pins, increase the number of variables to be compared.
- ◆ Multiple input variables can be of different types, but they must be comparable data types, e.g. one input pin is BYTE, another input pin is Real and Out is Real is allowed; e.g. one Input pin is BOOL (its value is FALSE or TRUE) and another input pin is Real, Output variable is Real is not allowed, because Bool type is not comparable with integers or floating point numbers. If the input types are different, the output variable type must be one that can cover the length of the input variable, e.g. if the inputs are BYTE and INT respectively, the output variable type must be at least INT and it can not be BYTE (refer to [Data Type of Variable](#) for a description of variable data type lengths).

4.8.4 LIMIT Limit Value

4.8.4.1 Function Description

This function is used to limit the output to the value of the input variable, so that only the values between MN and MX are output, regardless of the change in the input variable IN. The operator is IMIT.

4.8.4.2 Graphic



4.8.4.3 Input Parameter

Parameter	Data type	Description
EN	BOOL	Enable input, when EN is TRUE, this function is activated.
IN	Any basic data type	Input value
MN	Any basic data type	Minimum, output is greater than or equal to this value.
MX	Any basic data type	Maximum, output is less than or equal to this value.

4.8.4.4 Output Parameter

Parameter	Data type	Description
ENO	BOOL	Enable output, once EN is TRUE, ENO is TRUE.
OUT	Any basic data type	Output value. If $MX \geq IN \geq MN$, the output value is equal to IN; if $IN \leq MN$, the output value is equal to the value of MN; if $IN \geq MX$, the Output value is equal to the value of MX. For example: Dat_OUT:=LIMIT(4, 8, 10),

Parameter	Data type	Description
		Dat_OUT is 8; Dat_OUT:=LIMIT(4 , 3 , 10), Dat_OUT is 4; Dat_OUT:=LIMIT(4, 11, 10), Dat_OUT is 10.

4.8.4.5 Variable Declaration

VAR

Dat_Min: INT;

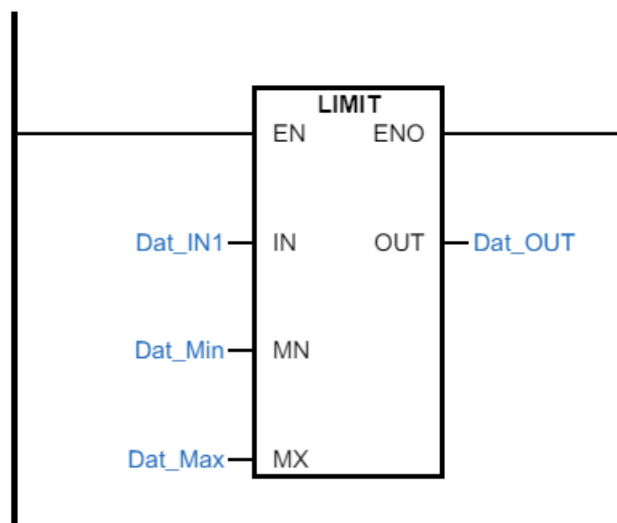
Dat_IN1: INT;

Dat_Max: INT;

Dat_OUT: INT;

END_VAR

4.8.4.6 Expression in LD



4.8.4.7 Expression in ST

```
Dat_OUT :=LIMIT(Dat_IN1,Dat_Min,Dat_Max);
```

4.8.4.8 Usage Restriction

It is recommended that MN, IN, MX and OUT be set to the same data type to avoid errors and mistakes in data conversion.

4.8.4.9 Cautions

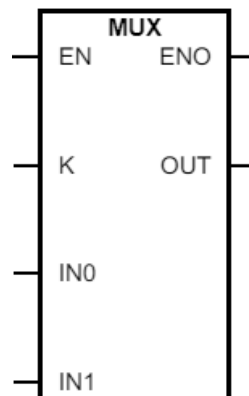
- ◆ Multiple input variables can be of different types, but they must be comparable data types, e.g. one input pin is BYTE, another input pin is Real and Out is Real is allowed; e.g. one input pin is BOOL (its value is False or True) and another input pin is Real, output variable is Real is not allowed, because BOOL type is not comparable with integers or floating point numbers.
- ◆ If the input types are different, the output variable type must be one that can cover the length of the input variable, e.g. if the inputs are BYTE and INT respectively, the output variable type must be at least INT and it can not be BYTE (refer to [Data Type of Variable](#) for a description of variable data type lengths).
- ◆ If the value of MN is greater than the value of the MX, OUT outputs the value of MX, regardless of the value of IN.

4.8.5 MUX Multiplexer

4.8.5.1 Function Description

This function is used to select a variable output from multiple input pins. The operator is MUX.

4.8.5.2 Graphic



4.8.5.3 Input Parameter

Parameter	Data type	Description
EN	BOOL	Enable input, when EN is TRUE, this function is activated.
K	Any_INT	Data selection number; the value of K will determine which Input pin is selected as the Output, e.g. K is 0, outputs the value of IN0.
IN0	Any basic data type	The first input value.
IN1	Any basic	The second input value.

Parameter	Data type	Description
	data type	

4.8.5.4 Output Parameter

Parameter	Data type	Description
ENO	BOOL	Enable output, once EN is TRUE, ENO is TRUE.
OUT	Any basic data type	◆ Output after selecting one of the multiple candidate variables. For example: Dat_OUT:=MUX(0,30,40,50,60,70,80); (* 0 means that the first data is selected, result Dat_OUT=30 *);

4.8.5.5 Variable Declaration

VAR

K_1: BYTE;

Dat_IN0: INT;

Dat_IN1: INT;

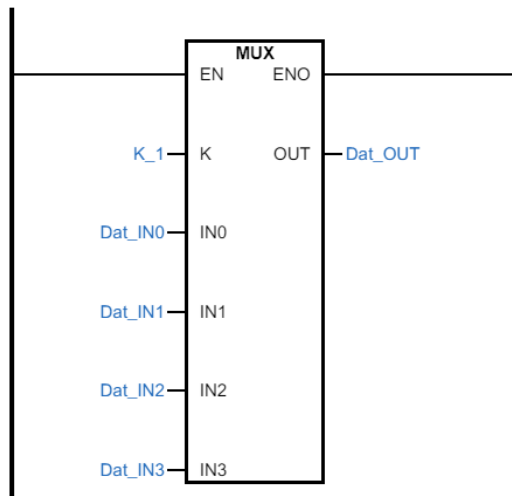
Dat_IN2: INT;

Dat_IN3: INT;

Dat_OUT: INT;

END_VAR

4.8.5.6 Expression in LD



4.8.5.7 Expression in ST

```
Dat_OUT := MUX(K_1, Dat_IN0, Dat_IN1, Dat_IN2, Dat_IN3);
```

4.8.5.8 Usage Restriction

- ◆ Available data type for K is ANY_INT (such as INT, SINT, DINT, etc.).
- ◆ Any data type is applicable for the selected number and output, but it is recommended that the same data types are used so that errors and mistakes in data conversion can be avoided.
- ◆ The maximum number of inputs should not exceed 480, otherwise it will cause the software to exit abnormally during compilation. To ensure the normal operation of the program, it is recommended that the number of data not involved in the comparison should not exceed 300.

4.8.5.9 Cautions

- ◆ The number of data entries can be more than two. In LD language, by right-clicking the function, select **Add Pin**, Add input value.
- ◆ If the input types are different, the output variable type must be one that can cover the length of the input variable, e.g. if the inputs are BYTE and INT respectively, the output variable type must be at least INT and it can not be BYTE (refer to [Data Type of Variable](#) for a description of variable data type lengths).
- ◆ If the value of the selected input K is greater than the number of selected input variables, it will output the last input variable.

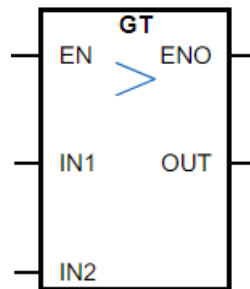
4.9 Comparison

4.9.1 GT Greater Than

4.9.1.1 Function Description

This function is used for data comparison. The operator is GT.

4.9.1.2 Graphic



4.9.1.3 Input Parameter

Parameter	Data type	Description
EN	BOOL	Enable input, when EN is TRUE, this function is activated.
IN1	Any basic data type	Input value 1
IN2	Any basic data type	Input value 2

4.9.1.4 Output Parameter

Parameter	Data type	Description
ENO	BOOL	Enable output, once EN is TRUE, ENO is TRUE.
OUT	BOOL	Comparison result, OUT is TRUE when IN1 is greater than IN2, otherwise, OUT is FALSE.

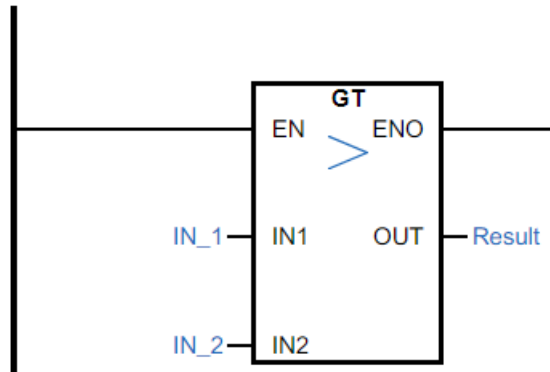
4.9.1.5 Variable Declaration

```
VAR
    IN_1: INT;
    IN_2: INT;
```

Result: BOOL;

END_VAR

4.9.1.6 Expression in LD



4.9.1.7 Expression in ST

```
GT_OUT := GT(EN := TRUE, IN1 := IN1, IN2 := IN2);
```

```
Result := GT_OUT;
```

4.9.1.8 Usage Restriction

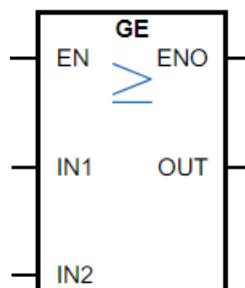
- ◆ Please set IN1, IN2 to the same data type, otherwise complication fails.
- ◆ data type of IN1 and IN2 cannot be BOOL.

4.9.2 GE Greater Than or Equal to

4.9.2.1 Function Description

This function is used for data comparison. The operator is GE.

4.9.2.2 Graphic



4.9.2.3 Input Parameter

Parameter	Data type	Description
EN	BOOL	Enable input, when EN is TRUE, this function is activated.
IN1	Any basic data type	Comparison value 1
IN2	Any basic data type	Comparison value 2

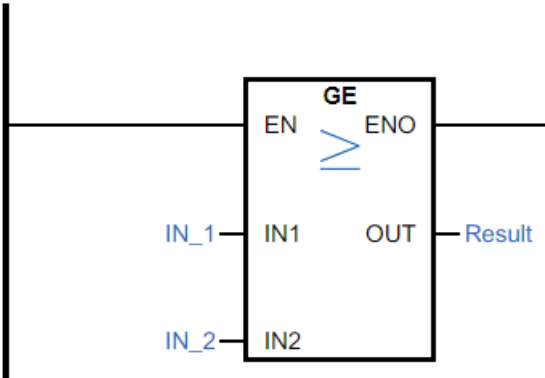
4.9.2.4 Output Parameter

Parameter	Data type	Description
ENO	BOOL	Enable output, once EN is TRUE, ENO is TRUE.
OUT	BOOL	Comparison result, OUT is TRUE when IN1 is greater than or equal to IN2, otherwise, OUT is FALSE.

4.9.2.5 Variable Declaration

```
VAR
  IN_1: INT;
  IN_2: INT;
  Result: BOOL;
END_VAR
```

4.9.2.6 Expression in LD



4.9.2.7 Expression in ST

```
GE_OUT := GE(EN := TRUE, IN1 := IN_1, IN2 := IN_2);  
  
Result := GE_OUT;
```

4.9.2.8 Usage Restriction

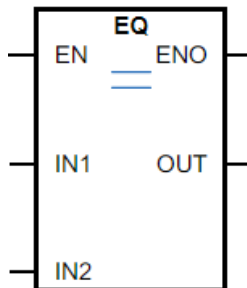
The data type of IN1 and IN2 can not be BOOL.

4.9.3 EQ Equal to

4.9.3.1 Function Description

This function is used to determine whether two data values are equal. The operator is EQ.

4.9.3.2 Graphic



4.9.3.3 Input Parameter

Parameter	Data type	Description
EN	BOOL	Enable input, when EN is TRUE, this function is activated.
IN1	Any basic data type	Comparison value 1
IN2	Any basic data type	Comparison value 2

4.9.3.4 Output Parameter

Parameter	Data type	Description
ENO	BOOL	Enable output, once EN is TRUE, ENO is TRUE.

Parameter	Data type	Description
OUT	BOOL	Comparison result, OUT is TRUE when IN1 is equal to IN2, otherwise, OUT is FALSE.

4.9.3.5 Variable Declaration

VAR

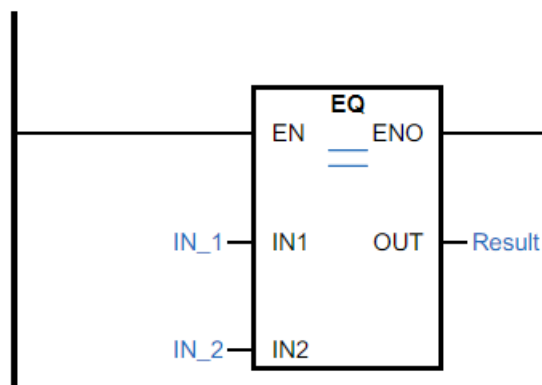
IN_1: INT;

IN_2: INT;

Result: BOOL;

END_VAR

4.9.3.6 Expression in LD



4.9.3.7 Expression in ST

EQ_OUT := EQ(EN := TRUE, IN1 := IN_1, IN2 := IN_2);

Result := EQ_OUT;

4.9.3.8 Usage Restriction

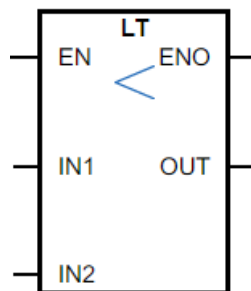
The data type of IN1 and IN2 can not be BOOL.

4.9.4 LT Less Than

4.9.4.1 Function Description

This function is used for data comparison. The operator is LT.

4.9.4.2 Graphic



4.9.4.3 Input Parameter

Parameter	Data type	Description
EN	BOOL	Enable input, when EN is TRUE, this function is activated.
IN1	Any basic data type	Comparison value 1
IN2	Any basic data type	Comparison value 2

4.9.4.4 Output Parameter

Parameter	Data type	Description
ENO	BOOL	Enable output, once EN is TRUE, ENO is TRUE.
OUT	BOOL	Comparison result, OUT is TRUE when IN1 is less than IN2, otherwise, OUT is FALSE.

4.9.4.5 Variable Declaration

VAR

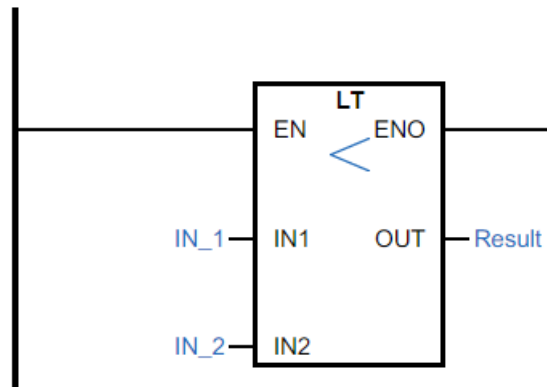
IN_1: INT;

IN_2: INT;

Result: BOOL;

END_VAR

4.9.4.6 Expression in LD



4.9.4.7 Expression in ST

```
LT_OUT := LT(EN := TRUE, IN1 := IN1, IN2 := IN2);
```

```
Result := LT_OUT;
```

4.9.4.8 Usage Restriction

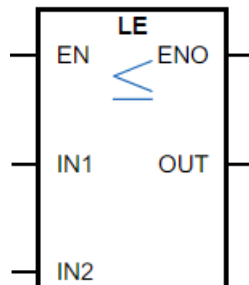
The data type of IN1 and IN2 can not be BOOL.

4.9.5 LE Less Than or Equal to

4.9.5.1 Function Description

This function is used for data comparison. The operator is LE.

4.9.5.2 Graphic



4.9.5.3 Input Parameter

Parameter	Data type	Description
EN	BOOL	Enable input, when EN is TRUE, this function is activated.
IN1	Any basic data type	Comparison value 1
IN2	Any basic data type	Comparison value 2

4.9.5.4 Output Parameter

Parameter	Data type	Description
ENO	BOOL	Enable output, once EN is TRUE, ENO is TRUE.
OUT	BOOL	Comparison result, OUT is TRUE when IN1 is less than or equal to IN2, otherwise OUT is FALSE.

4.9.5.5 Variable Declaration

VAR

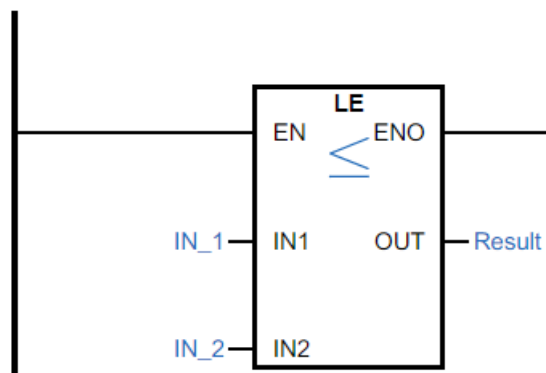
IN_1: INT;

IN_2: INT;

Result: BOOL;

END_VAR

4.9.5.6 Expression in LD



4.9.5.7 Expression in ST

```
LE_OUT := LT(EN := TRUE, IN1 := IN_1, IN2 := IN_2);
```

```
Result := LE_OUT;
```

4.9.5.8 Usage Restriction

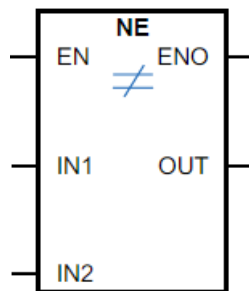
The data type of IN1 and IN2 can not be BOOL.

4.9.6 NE Not Equal to

4.9.6.1 Function Description

This function is used to determine whether two data values are equal. The operator is NE.

4.9.6.2 Graphic



4.9.6.3 Input Parameter

Parameter	Data type	Description
EN	BOOL	Enable input, when EN is TRUE, this function is activated.
IN1	Any basic data type	Comparison value 1
IN2	Any basic data type	Comparison value 2

4.9.6.4 Output Parameter

Parameter	Data type	Description
ENO	BOOL	Enable output, once EN is TRUE, ENO is TRUE.

Parameter	Data type	Description
OUT	BOOL	Comparison result, OUT is TRUE when IN1 is not equal to IN2, otherwise OUT is FALSE.

4.9.6.5 Variable Declaration

VAR

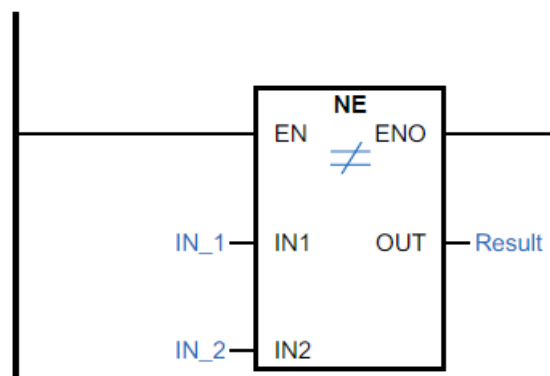
IN_1: INT;

IN_2: INT;

Result: BOOL;

END_VAR

4.9.6.6 Expression in LD



4.9.6.7 Expression in ST

```
NE_OUT := NE(EN := TRUE, IN1 := IN_1, IN2 := IN_2);
```

```
Result := NE_OUT;
```

4.9.6.8 Usage Restriction

The data type of IN1 and IN2 can not be BOOL.

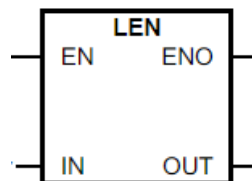
4.10 String Operation

4.10.1 LEN String Length

4.10.1.1 Function Description

This function is used to calculate the length of a string. Its operator is LEN.

4.10.1.2 Graphic



4.10.1.3 Input Parameter

Parameter	Data type	Description
EN	BOOL	Enable input, when EN is TRUE, this function is activated.
IN	STRING	Input string

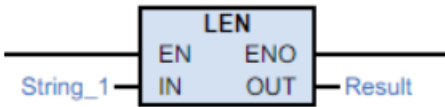
4.10.1.4 Output Parameter

Parameter	Data type	Description
ENO	BOOL	Enable output, once EN is TRUE, ENO is TRUE.
OUT	INT	String length

4.10.1.5 Variable Declaration

```
VAR
    String_1: STRING;
    Result: INT;
END_VAR
```

4.10.1.6 Expression in LD



4.10.1.7 Expression in ST

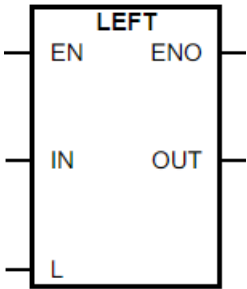
```
Result:=LEN(String_1);
```

4.10.2 LEFT Extract String from the Left

4.10.2.1 Function Description

Extracting a specific number of characters from the leftmost part of a string to form a new string. Its operator is LEFT.

4.10.2.2 Graphic



4.10.2.3 Input Parameter

Parameter	Data type	Description
EN	BOOL	Enable input, when EN is TRUE, this function is activated.
IN	STRING	Input string
L	ANY_INT	Number of characters to extract

4.10.2.4 Output Parameter

Parameter	Data type	Description
ENO	BOOL	Enable output, once EN is TRUE, ENO is TRUE.

Parameter	Data type	Description
OUT	STRING	Output string

4.10.2.5 Variable Declaration

VAR

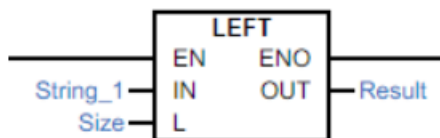
String_1: STRING;

Size: INT;

Result: String;

END_VAR

4.10.2.6 Expression in LD



4.10.2.7 Expression in ST

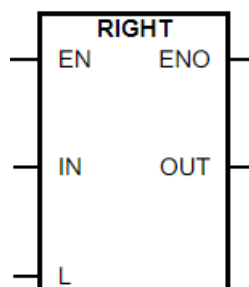
Result:=LEFT(String_1, Size);

4.10.3 RIGHT Extract String from the Right

4.10.3.1 Function Description

Extracting a specific number of characters from the rightmost part of a string to form a new string. Its operator is RIGHT.

4.10.3.2 Graphic



4.10.3.3 Input Parameter

Parameter	Data type	Description
EN	BOOL	Enable input, when EN is TRUE, this function is activated.
IN	STRING	Input string
L	ANY_INT	Number of characters to extract

4.10.3.4 Output Parameter

Parameter	Data type	Description
ENO	BOOL	Enable output, once EN is TRUE, ENO is TRUE.
OUT	STRING	Output string

4.10.3.5 Variable Declaration

VAR

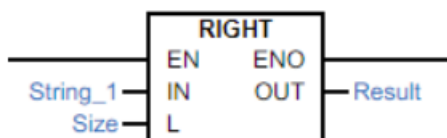
String_1: STRING;

Size:INT;

Result: STRING;

END_VAR

4.10.3.6 Expression in LD



4.10.3.7 Expression in ST

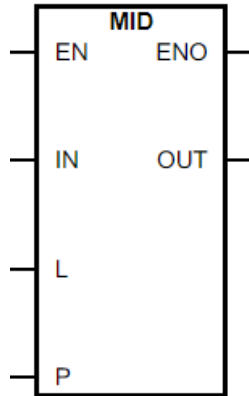
Result:=RIGHT(String_1, Size);

4.10.4 MID Extract String from the Middle

4.10.4.1 Function Description

Extracting a specific number of characters from the middle of a string to form a new string. Its operator is MID.

4.10.4.2 Graphic



4.10.4.3 Input Parameter

Parameter	Data type	Description
EN	BOOL	Enable input, when EN is TRUE, this function is activated.
IN	STRING	Input string
L	ANY_INT	Number of characters to extract
P	ANY_INT	The starting position of extraction, counted from the left.

4.10.4.4 Output Parameter

Parameter	Data type	Description
ENO	BOOL	Enable output, once EN is TRUE, ENO is TRUE.
OUT	STRING	Output string

4.10.4.5 Variable Declaration

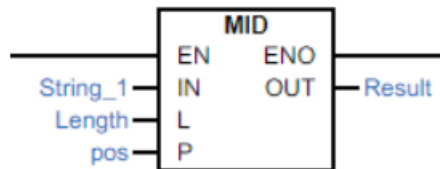
```
VAR  
  
String_1: STRING;  
  
Length:INT;
```

Pos:INT;

Result: STRING;

END_VAR

4.10.4.6 Expression in LD



4.10.4.7 Expression in ST

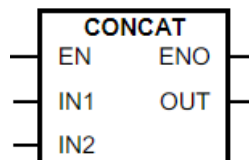
Result:=MID(String_1, Length, Pos);

4.10.5 CONCAT Concatenate string

4.10.5.1 Function Description

Concatenate two strings to form a new string. Its operator is CONCAT.

4.10.5.2 Graphic



4.10.5.3 Input Parameter

Parameter	Data type	Description
EN	BOOL	Enable input, when EN is TRUE, this function is activated.
IN1	STRING	Head of the concatenated string
IN2	STRING	Tail of the concatenated string

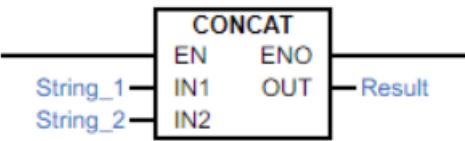
4.10.5.4 Output Parameter

Parameter	Data type	Description
ENO	BOOL	Enable output, once EN is TRUE, ENO is TRUE.
OUT	STRING	Concatenated string

4.10.5.5 Variable Declaration

```
VAR
String_1: STRING;
String_2: STRING;
Result: STRING;
END_VAR
```

4.10.5.6 Expression in LD



4.10.5.7 Expression in ST

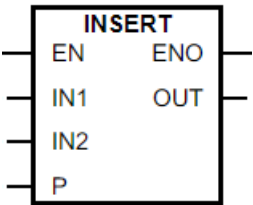
```
Result:=CONCAT(String_1, String_2);
```

4.10.6 INSERT Insert String

4.10.6.1 Function Description

Insert another string into a string to form a new string. Its operator is INSERT.

4.10.6.2 Graphic



4.10.6.3 Input Parameter

Parameter	Data type	Description
EN	BOOL	Enable input, when EN is TRUE, this function is activated.
IN1	STRING	Initial string
IN2	STRING	String to be inserted into IN1
P	ANY_INT	Position of insertion, count starts from the left of the string corresponding to IN1 (starting from 1), and string IN2 is inserted at this position.

4.10.6.4 Output Parameter

Parameter	Data type	Description
ENO	BOOL	Enable output, once EN is TRUE, ENO is TRUE.
OUT	STRING	New string after inserting specific characters.

4.10.6.5 Variable Declaration

VAR

String_1: STRING;

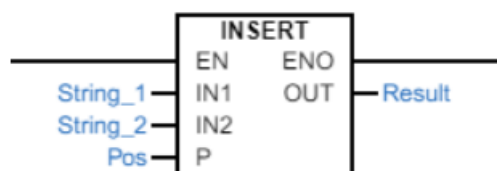
String_2: STRING;

Pos:INT;

Result: STRING;

END_VAR

4.10.6.6 Expression in LD



4.10.6.7 Expression in ST

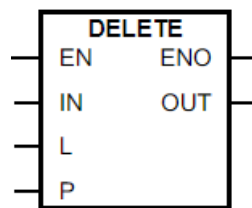
```
Result:=INSERT(String_1, String_2, Pos);
```

4.10.7 DELETE Delete String

4.10.7.1 Function Description

Delete a specific continuous part of a string and forms a new string with the remaining characters. Its operator is DELETE.

4.10.7.2 Graphic



4.10.7.3 Input Parameter

Parameter	Data type	Description
EN	BOOL	Enable input, when EN is TRUE, this function is activated.
IN	STRING	Initial string
L	STRING	Length of the characters to be deleted
P	ANY_INT	Starting position of the characters to be deleted, counting from the left (starting from 1).

4.10.7.4 Output Parameter

Parameter	Data type	Description
ENO	BOOL	Enable output, once EN is TRUE, ENO is TRUE.
OUT	STRING	New string after deleting specific characters.

4.10.7.5 Variable Declaration

```
VAR
String_1: STRING;

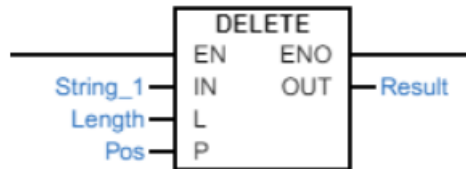
Length: INT;

Pos:INT;
```

```
Result:STRING;
```

```
END_VAR
```

4.10.7.6 Expression in LD



4.10.7.7 Expression in ST

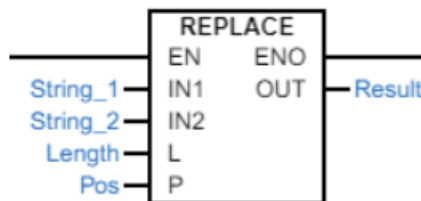
```
Result:=DELETE(String_1, Legth, Pos);
```

4.10.8 REPLACE Replace String

4.10.8.1 Function Description

Replace a part of characters in a string with a shorter string, to form a new string. Its operator is REPLACE.

4.10.8.2 Graphic



4.10.8.3 Input Parameter

Parameter	Data type	Description
EN	BOOL	Enable input, when EN is TRUE, this function is activated.
IN	STRING	Initial string
L	STRING	Length of the characters to be replaced
P	ANY_INT	Starting position of replacement (counting from the left)

4.10.8.4 Output Parameter

Parameter	Data type	Description
ENO	BOOL	Enable output, once EN is TRUE, ENO is TRUE.
OUT	STRING	New string after replacing specific characters.

4.10.8.5 Variable Declaration

VAR

String_1: STRING;

String_2: STRING;

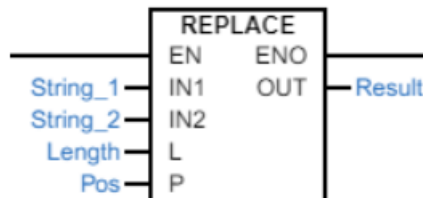
Length:INT;

Pos:INT;

Result:STRING;

END_VAR

4.10.8.6 Expression ins LD



4.10.8.7 Expression in ST

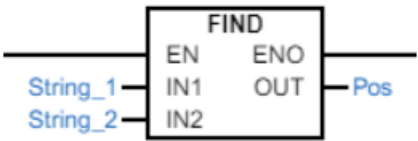
```
Result:=REPLACE(String_1, String_2, Legth, Pos);
```

4.10.9 FIND Find String

4.10.9.1 Function Description

Find the starting position of a specific substring within a string. Its operator is FIND.

4.10.9.2 Graphic



4.10.9.3 Input Parameter

Parameter	Data type	Description
EN	BOOL	Enable input, when EN is TRUE, this function is activated.
IN1	STRING	Initial string
IN2	STRING	String to be found

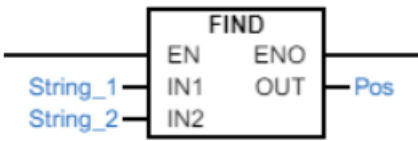
4.10.9.4 Output Parameter

Parameter	Data type	Description
ENO	BOOL	Enable output, once EN is TRUE, ENO is TRUE.
OUT	STRING	Position of the first character to be found.

4.10.9.5 Variable Declaration

```
VAR
String_1: STRING;
String_2: STRING;
Pos:INT;
END_VAR
```

4.10.9.6 Expression in LD



4.10.9.7 Expression in ST

```
Pos:=FIND(String_1, String_2);
```

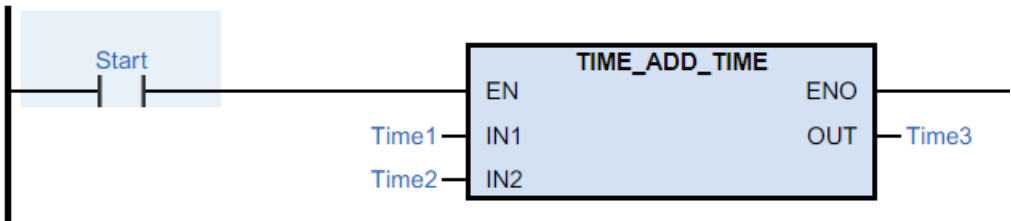
4.11 Time

4.11.1 *_ADD_* Data and Time Addition

4.11.1.1 Function Description

This function is used for adding time or date. Its operator is *_ADD_* (e.g., TIME_ADD_TIME).

4.11.1.2 Graphic



4.11.1.3 Input Parameter

Parameter	Data type	Description
EN	BOOL	Enable input, when EN is TRUE, this function is activated.
IN1	TIME, TOD, DT	Addend 1
IN2	TIME	Addend 2

4.11.1.4 Output Parameter

Parameter	Data type	Description
ENO	BOOL	Enable output, once EN is TRUE, ENO is TRUE.
OUT	the same data type as IN1	Sum, the result after addition.

4.11.1.5 Variable Declaration

VAR

Start : BOOL ;

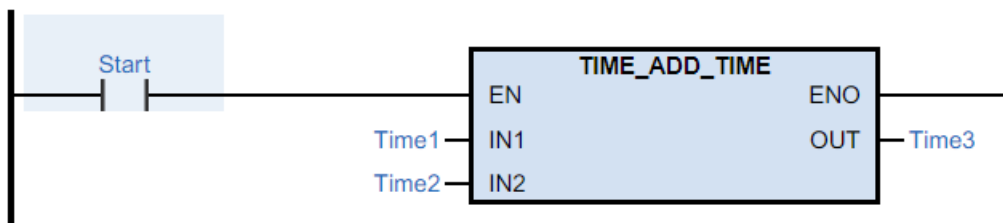
Time1:TIME;

Time2:TIME;

Time3:TIME;

END_VAR

4.11.1.6 Expression in LD



4.11.1.7 Expression in ST

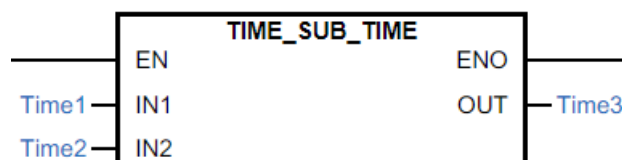
```
Time3:=TIME_ADD_TIME(EN:=Start,IN1:=Time1,IN2:=Time2);
```

4.11.2 *_SUB_* Date and Time Subtraction

4.11.2.1 Function Description

This function is used for subtracting time or date. Its operator is *_SUB_* (e.g., TIME_SUB_TIME).

4.11.2.2 Graphic



4.11.2.3 Input Parameter

Parameter	Data type	Description
EN	BOOL	Enable input, when EN is TRUE, this function is activated.
IN1	TIME, DATE, TOD, DT	Minuend
IN2	The data type of IN2 depends on the data type of IN1. If IN1 is TIME, then IN2 can only be TIME; if IN1 is DATE, then IN2 can only be DATE; if IN1 is TOD, then IN2 can be TOD or TIME; if IN1 is DT, then IN2 can be DT or TIME.	Subtrahend

4.11.2.4 Output Parameter

Parameter	Data type	Description
ENO	BOOL	Enable output, once EN is TRUE, ENO is TRUE.
OUT	The data type of OUT depends on the data type of IN1. If IN1 is TIME, then OUT can only be TIME; if IN1 is DATE, then OUT can only be DATE; if IN1 is TOD, then OUT can be TOD or TIME; if IN1 is DT, then OUT can be DT or TIME	Difference, result of subtracting IN2 from IN1.

4.11.2.5 Variable Declaration

VAR

Start : BOOL ;

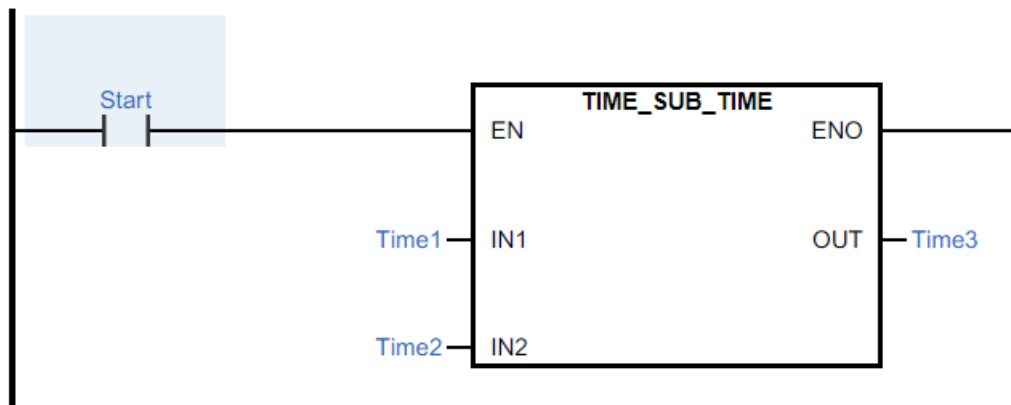
Time1: TIME;

Time2:: TIME;

Time3: TIME;

END_VAR

4.11.2.6 Expression in LD



4.11.2.7 Expression in ST

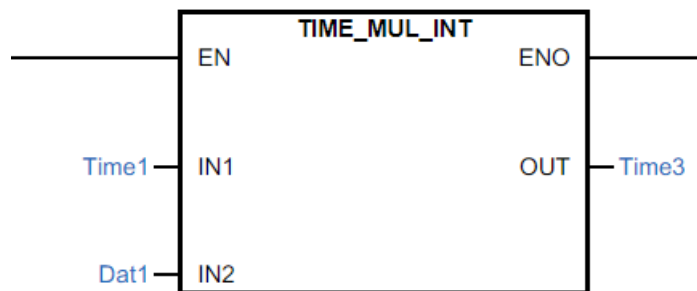
```
Time3:=TIME_SUB_TIME(EN:=Start,IN1:=Time1,IN2:=Time2);
```

4.11.3 *_MUL_* Date and Time Multiplication

4.11.3.1 Function Description

This function is used to perform multiplication operations on date or time. Its operator is *_MUL_* (e.g., TIME_MUL_INT).

4.11.3.2 Graphic



4.11.3.3 Input Parameter

Parameter	Data type	Description
EN	BOOL	Enable input, when EN is TRUE, this function is activated.
IN1	TIME	Multiplicand
IN2	ANY_INT or ANY_REAL	Multiplier

4.11.3.4 Output Parameter

Parameter	Data type	Description
ENO	BOOL	Enable output, once EN is TRUE, ENO is TRUE.
OUT	TIME	Result of multiplying the input values.

4.11.3.5 Variable Declaration

VAR

Start : BOOL ;

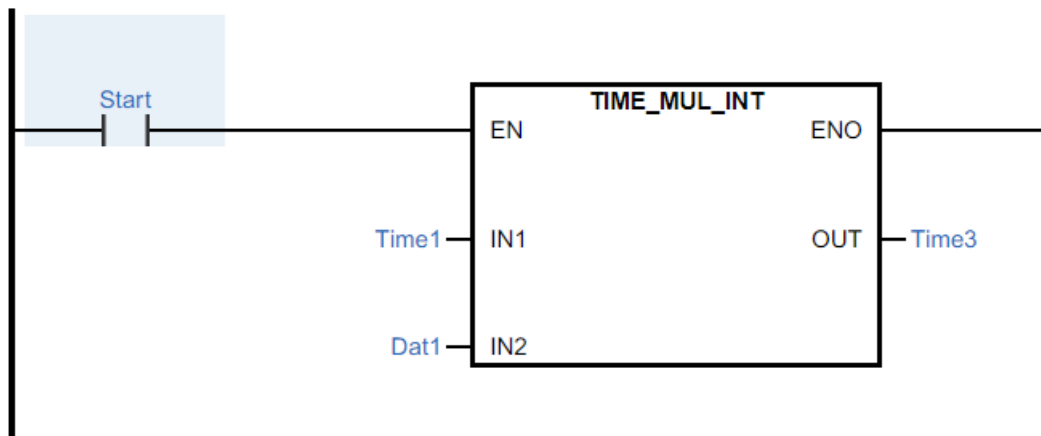
Time1:TIME;

Data1:INT;

Time3:TIME;

END_VAR

4.11.3.6 Expression in LD



4.11.3.7 Expression in ST

```
Time3:=TIME_MUL_INT(EN:=Start,IN1:=Time1,IN2:=Dat1);
```

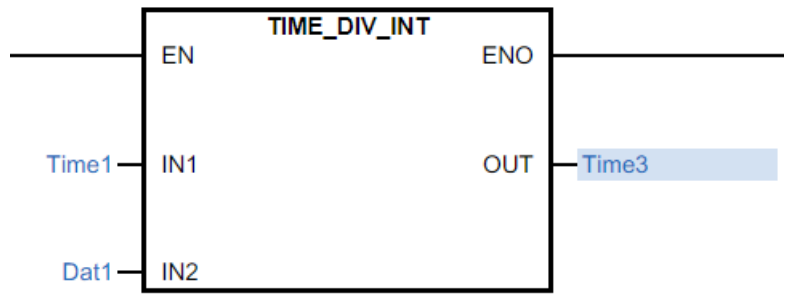
4.11.4 *_DIV_** Date and Time Division

4.11.4.1 Function Description

This function is used to perform division operations on date or time. Its operator is `*_DIV_**` (e.g.,

TIME_DIV_INT).

4.11.4.2 Graphic



4.11.4.3 Input Parameter

Parameter	Data type	Description
EN	BOOL	Enable input, when EN is TRUE, this function is activated.
IN1	TIME	Input time value, used as the dividend.
IN2	ANY_INT or ANY_REAL	Divisor

4.11.4.4 Output Parameter

Parameter	Data type	Description
ENO	BOOL	Enable output, once EN is TRUE, ENO is TRUE.
OUT	TIME	Result of division of input values.

4.11.4.5 Variable Declaration

```
VAR

    Start : BOOL ;

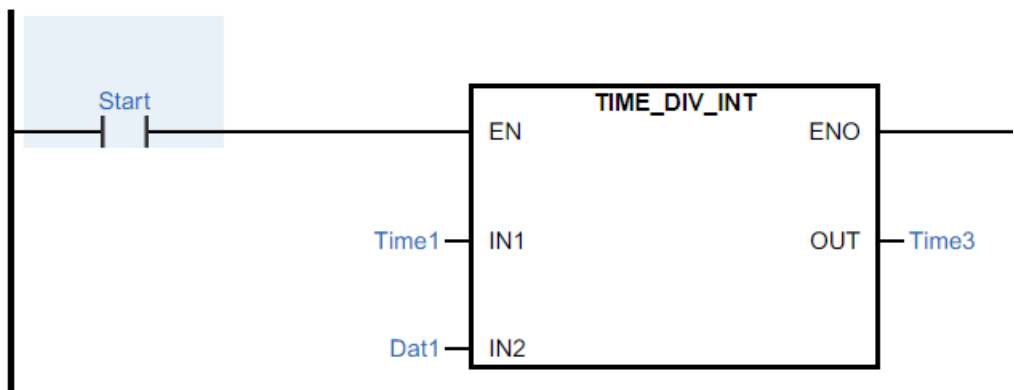
    Time1:TIME;

    Dat1:INT;

    Time3:TIME;

END_VAR
```


4.11.4.6 Expression in LD



4.11.4.7 Expression in ST

```
Time3:=TIME_DIV_INT(EN:=Start,IN1:=Time1,IN2:=Dat1);
```

4.11.4.8 Usage Restriction

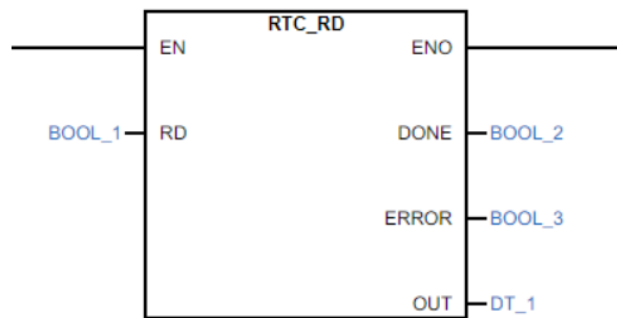
IN2 can not be 0.

4.11.5 RTC_RD Read Real-time Clock

4.11.5.1 Function Description

This function is used to read the real-time value of the PLC clock. Its operator is RTC_RD.

4.11.5.2 Graphic



4.11.5.3 Input Parameter

Parameter	Data type	Description
EN	BOOL	Enable input, when EN is TRUE, this function is activated.

Parameter	Data type	Description
RD	BOOL	Enable signal to read Real-time clock, When RD is TRUE, it reads the real-time clock information; otherwise, it does not read the real-time clock information.

4.11.5.4 Output Parameter

Parameter	Data type	Description
ENO	BOOL	Enable output, once EN is TRUE, ENO is TRUE.
DONE	BOOL	Success flag for real-time clock reading operation. If successful, DONE is set to TRUE.
ERROR	BOOL	Error flag for real-time clock reading operation. If an error occurs, ERROR is set to TRUE.
OUT	DT	Current RTC time in the PLC

4.11.5.5 Variable Declaration

```
VAR
```

```
RD_1:BOOL;
```

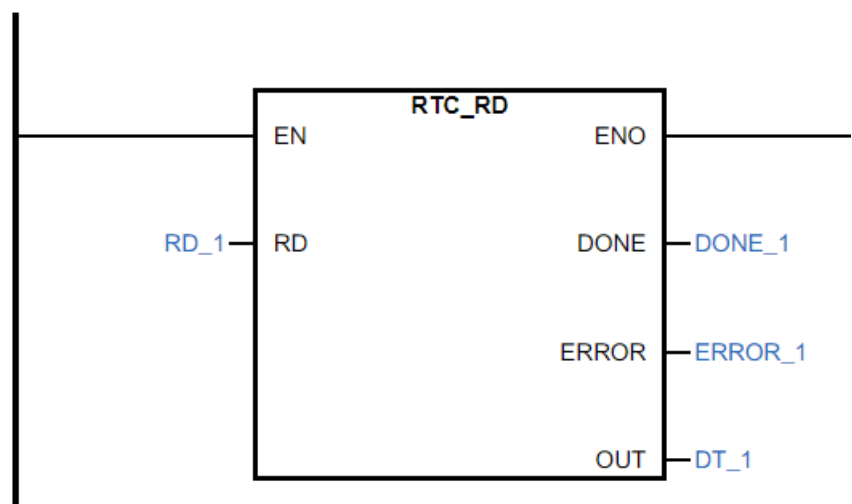
```
DONE_1:BOOL;
```

```
ERROR_1:BOOL;
```

```
DT_1:DT;
```

```
END_VAR
```

4.11.5.6 Expression in LD



4.11.5.7 Expression in ST

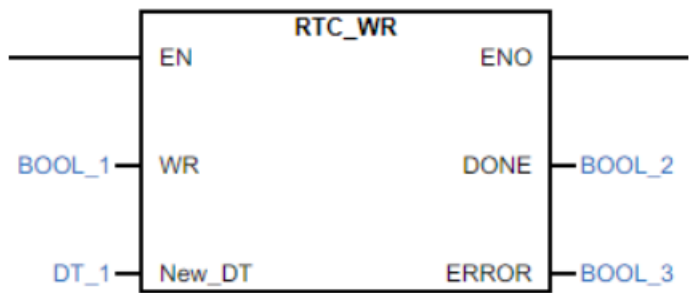
```
DT_1:=RTC_RD(EN:=TRUE,RD:=RD_1,DONE=>DONE_1,ERROR=>ERROR_1);
```

4.11.6 RTC_WR Write Real-time Clock

4.11.6.1 Function Description

This function is used to write the values to real-time clock time of the PLC. Its operator is **RTC_WR**.

4.11.6.2 Graphic



4.11.6.3 Input Parameter

Parameter	Data type	Description
EN	BOOL	Enable input, when EN is TRUE, this function is activated.
WR	BOOL	Enable signal to write Real-time clock, When WR is TRUE, write the value of

Parameter	Data type	Description
		the real-time clock.
New_DT	DT	Date and Time value to be written.

4.11.6.4 Output Parameter

Parameter	Data type	Description
ENO	BOOL	Enable output, once EN is TRUE, ENO is TRUE.
DONE	BOOL	Success flag for real-time clock writing operation. If successful, DONE is set to TRUE.
ERROR	BOOL	Error flag for real-time clock writing operation. If an error occurs, ERROR is set to TRUE.
OUT	DT	Current RTC time in the PLC

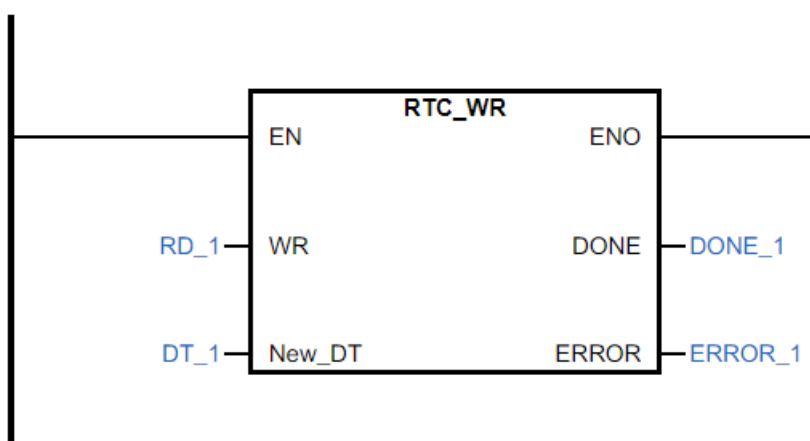
4.11.6.5 Variable Declaration

```

VAR
WR_1:BOOL;
DONE_1:BOOL;
ERROR_1:BOOL;
DT_1:DT;
END_VAR

```

4.11.6.6 Expression in LD



4.11.6.7 Expression in ST

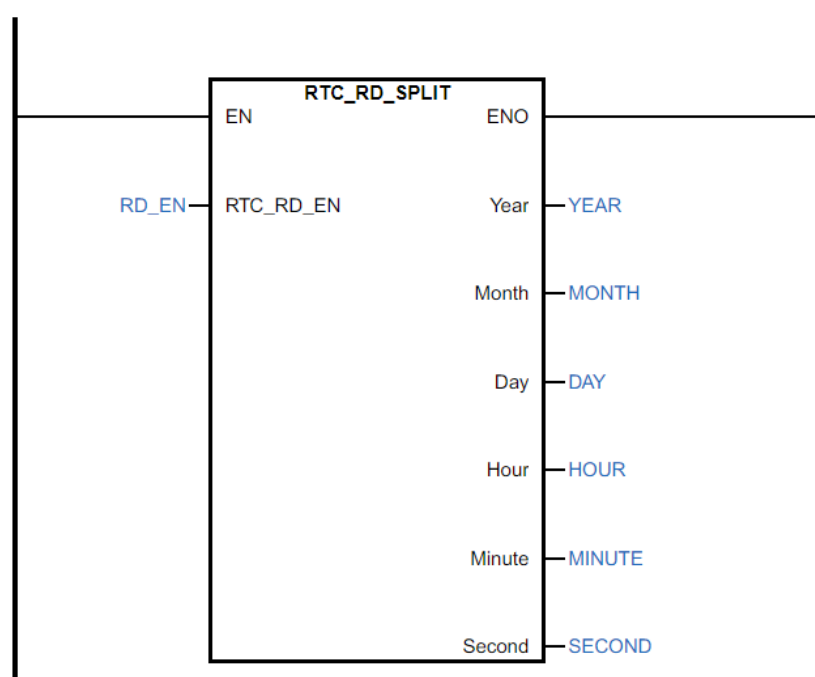
```
RTC_WR(WR:=WR_1,New_DT:=DT_1,DONE=>DONE_1,ERROR=>ERROR_1);
```

4.11.7 RTC_RD_SPLIT Separated RTC Reading

4.11.7.1 Function Description

This function is used to read the real-time value of the clock and output the values of year, month, day, hour, minute, and second separately.

4.11.7.2 Graphic



4.11.7.3 Input Parameter

Parameter	Data type	Description
EN	BOOL	Enable input, when EN is TRUE, this function is activated.
RTC_RD_EN	BOOL	Enable signal to read PLC RTC, When RTC_RE_EN is TRUE, perform the operation to read the RTC, getting year, month, day, hour, minute, second separately.

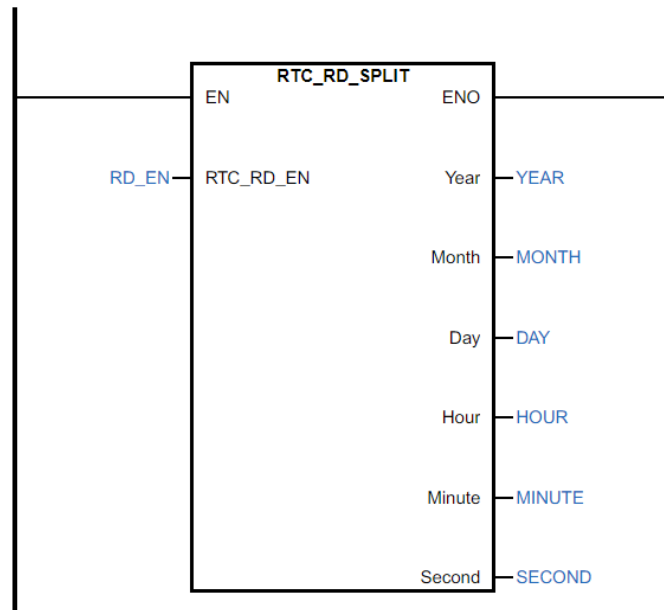
4.11.7.4 Output Parameter

Parameter	Data type	Description
ENO	BOOL	Enable output, once EN is TRUE, ENO is TRUE.
Year	UINT	The year corresponding to the RTC value
Month	UINT	The month corresponding to the RTC value
Day	UINT	The day corresponding to the RTC value
Hour	UINT	The hour corresponding to the RTC value
Minute	UINT	The minute corresponding to the RTC value
Second	UINT	The second corresponding to the RTC value

4.11.7.5 Variable Declaration

```
VAR  
  
    RD_EN:BOOL;  
  
    YEAR:UINT;  
MONTH:UINT;  
DAY:UINT;  
HOUR:UINT;  
MINUTE:UINT;  
SECOND:UINT;  
END_VAR
```

4.11.7.6 Expression in LD



4.11.7.7 Expression in ST

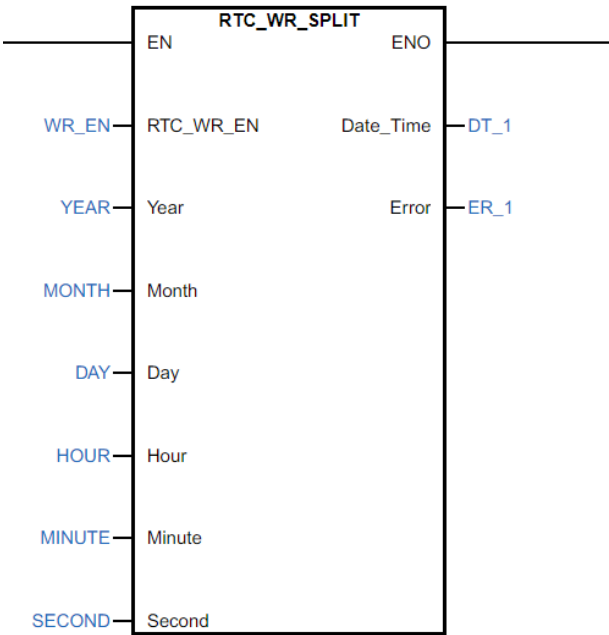
```
RTC_RD_SPLIT(RTC_RD_EN:=RD_EN,Year=>YEAR,Month=>MONTH,Day=>DAY,Hour=>HOUR,Minute=>MINUTE,Second=>SECOND);
```

4.11.8 RTC_WR_SPLIT Separated RTC Writing

4.11.8.1 Function Description

This function is used to write the values to PLC RTC in the forms of year, month, day, hour, minute, and second separately.

4.11.8.2 Graphic



4.11.8.3 Input Parameter

Parameter	Data type	Description
EN	BOOL	Enable input, when EN is TRUE, this function is activated.
RTC_WR_EN	BOOL	Enable signal to write PLC RTC , When RTC_WR_EN is TRUE, perform the operation to write the RTC, writing year, month, day, hour, minute, second separately.
Year	UINT	The year corresponding to the value to be written to the RTC
Month	UINT	The month corresponding to the value to be written to the RTC
Day	UINT	The day corresponding to the value to be written to the RTC
Hour	UINT	The hour corresponding to the value to be written to the RTC
Minute	UINT	The minute corresponding to the value to be written to the RTC
Second	UINT	The second corresponding to the value to be written to the RTC

4.11.8.4 Output Parameter

Parameter	Data type	Description
ENO	BOOL	Enable output, once EN is TRUE, ENO is TURE.

Parameter	Data type	Description
Date_Time	DT	Date and time to be written to the RTC.
Error	BOOL	Error flag for RTC writing operation, if an error occurs during the writing RTC operation, set Error to TRUE.

4.11.8.5 Variable Declaration

```

VAR

    WR_EN:BOOL;

YEAR:UINT;

MONTH:UINT;

DAY:UINT;

HOUR:UINT;

MINUTE:UINT;

SECOND:UINT;

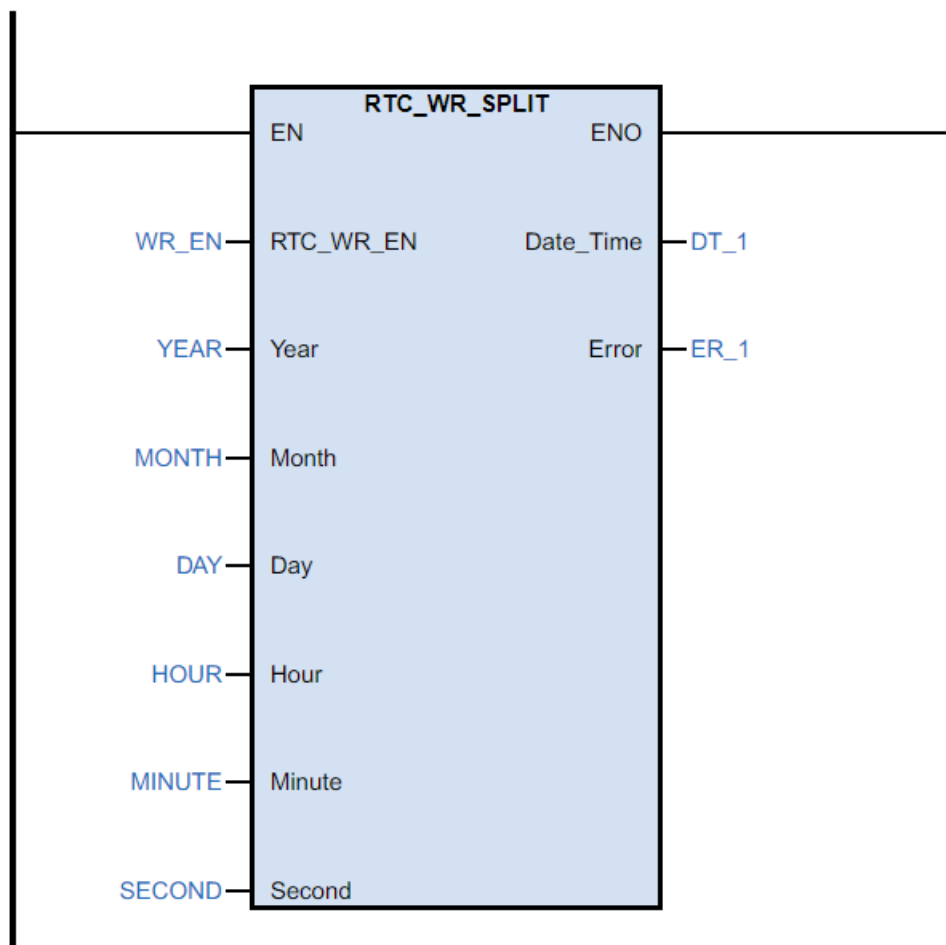
DT_1:DT;

ER_1:BOOL;

END_VAR

```

4.11.8.6 Expression in LD



4.11.8.7 Expression in ST

```
RTC_WR_SPLIT(RTC_WR_EN:=WR_EN,Year:=YEAR,Month:=MONTH,Day:=DAY,Hour:=HOUR,Minute:=MINUTE,Second:=SECOND,Date_Time=>DT_1,Error=>ER_1);
```

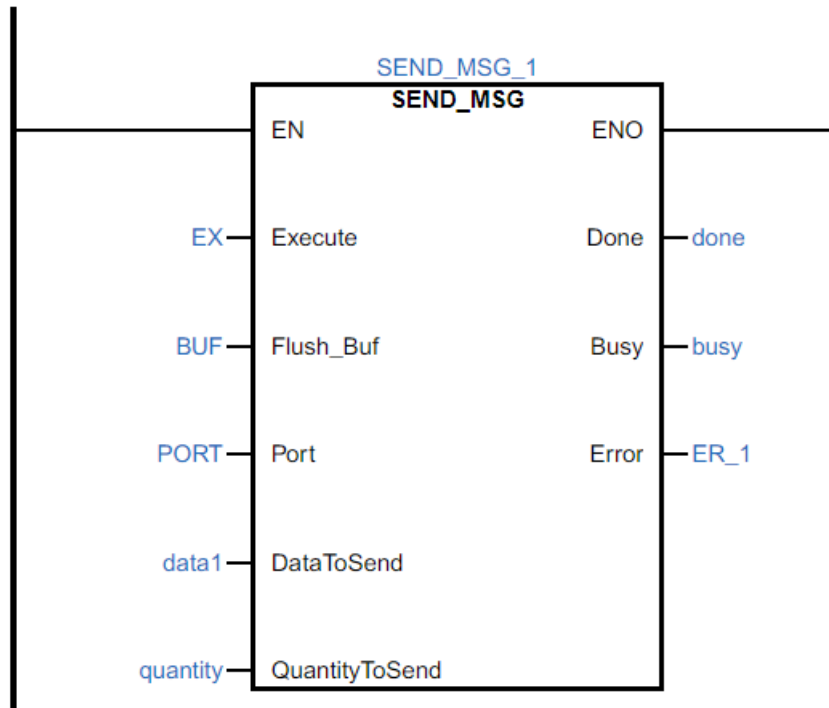
4.12 Communication

4.12.1 SEN_MSG Raw Serial Port Send Data

4.12.1.1 Function Description

This function block is used for sending data via a raw serial port. Its operator is SEND_MSG.

4.12.1.2 Graphic



4.12.1.3 Input Parameter

Parameter	Data type	Initial value	Description
EN	BOOL	FALSE	Enable input, when EN is TRUE, this function block is activated.
Execute	BOOL	FALSE	Execution switch, When It is TRUE, the operation of sending data via raw serial port is executed.
Flush_Buf	BOOL	FALSE	Indicate whether to clear the buffer data before sending. When It is TRUE, buffer data is cleared before sending.
Port	UINT	-	Serial port number to be used (e.g. COM1). Valid values are 1, 2, 3, and 4. 1 and 2 represent COM1 and COM2 on the CPU, 3 represents the serial port on extension board 1, and 4 represents the serial port on extension board 2.
DataToSend	ARRAY [0..255] OF BYTE	-	Data to be sent
QuantityToSend	UINT	-	Length of the data to be sent in bytes

4.12.1.4 Output Parameter

Parameter	Data type	Initial value	Description
ENO	BOOL	FALSE	Enable output, once EN is TRUE, ENO is TRUE.
Done	BOOL	FALSE	Success flag of sending data. If successful, Done is set to TRUE.
Busy	BOOL	FALSE	Whether the serial port is currently reading or writing data. it is TRUE if the port is reading data.
Error	BOOL	FALSE	Whether the buffer has overflowed. It is TRUE if the buffer overflows.

4.12.1.5 Variable Declaration

```
VAR

    EX:BOOL;

    BUF:BOOL;

    PORT:UINT;

    data1: ARRAY[0..255] OF BYTE;

    quantity:UINT;

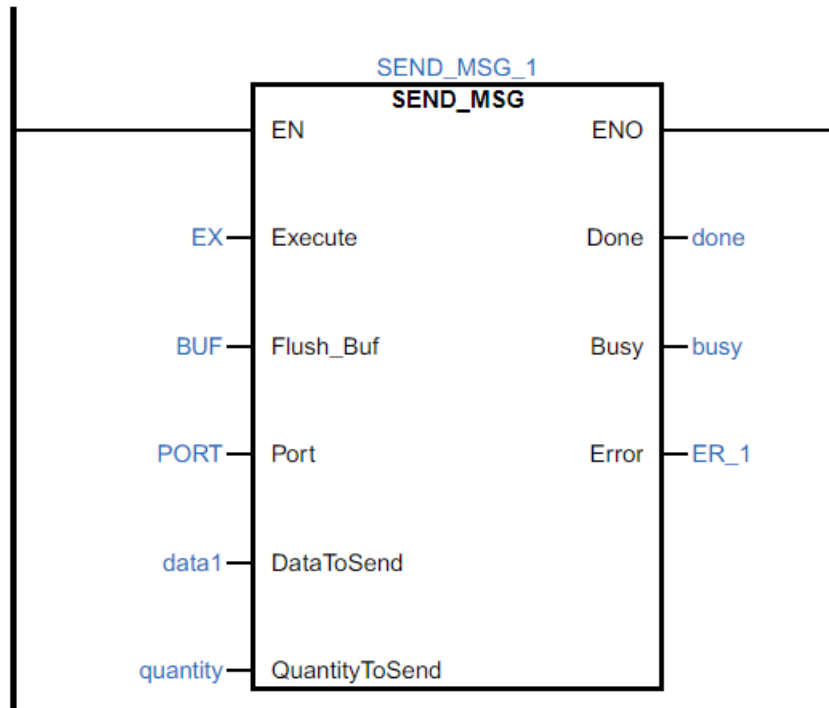
    done:BOOL;

    busy:BOOL;

    ER_1:BOOL;

END_VAR
```

4.12.1.6 Expression in LD



4.12.1.7 Expression in ST

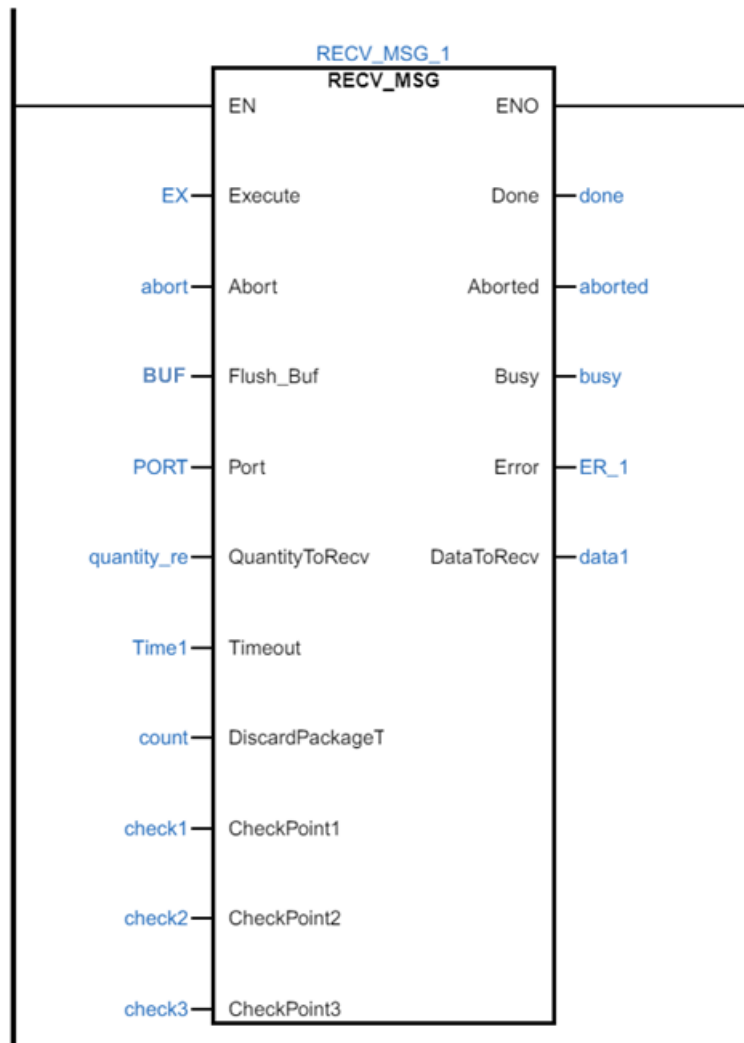
```
SEND_MSG_1(EN:=TRUE,Execut:=EX,Flush_Buf:=BUF,Port:=PORT,DataToSend:=data1,QuantityToSend:=quantity,Done=>done,Busy=>busy,Erro=>ER_1);
```

4.12.2 RECV_MSG Raw Serial Port Receive Data

4.12.2.1 Function Description

This function block is used for receiving data via a raw serial port. Its operator is SEND_MSG.

4.12.2.2 Graphic



4.12.2.3 Input Parameter

Parameter	Data type	Initial value	Description
EN	BOOL	FALSE	Enable input, when EN is TRUE, this function block is activated.
Execute	BOOL	FALSE	Execution switch, When It is TRUE, the operation of receiving data via raw serial port is executed.
Abort	BOOL	FALSE	Whether to abort data reception. When it is TRUE, the data reception is aborted(due to invalid data sending configuration).
Flush_Bu	BOOL	FALSE	Whether to clear the buffer data after receiving. When it is TRUE, buffer data is cleared after receiving.

Parameter	Data type	Initial value	Description
Port	UINT	-	Serial port number to be used (e.g. COM1). Valid values are 1, 2, 3, and 4. 1 and 2 represent COM1 and COM2 on the CPU, 3 represents the serial port on extension board 1, and 4 represents the serial port on extension board 2.
QuantityToRecv	UINT	-	Length of the data to be received in byte. The received data length cannot be greater than the length of the sent data.
TimeOut	TIME	-	Connection timeout duration
DiscardPackageTimeout	UDINT	-	Timeout duration of receiving a single data package in milliseconds
CheckPoint1	STRING	-	Checkpoint expression 1. The expression cannot contain spaces and does not support enforcement. For example, to verify that the first element of received array has a value of hexadecimal 10, the expression would be 0=16#10.
CheckPoint2	STRING	-	Checkpoint expression 2. The expression cannot contain spaces and does not support enforcement. For example, to verify that the first element of received array has a value of hexadecimal 10, the expression would be 0=16#10.
CheckPoint3	STRING	-	Checkpoint expression 3. The expression cannot contain spaces and does not support enforcement. For example, to verify that the first element of received array has a value of hexadecimal 10, the expression would be 0=16#10.

4.12.2.4 Output Parameter

Parameter	Data type	Initial value	Description
ENO	BOOL	FALSE	Enable output, once EN is TRUE, ENO is TRUE.
Done	BOOL	FALSE	Whether the data is received successfully. It is TRUE if the receiving is successful.

Parameter	Data type	Initial value	Description
Aborted	BOOL	FALSE	Whether data reception is aborted. It is TRUE if data reception is aborted
Busy	BOOL	FALSE	Whether the serial port is currently reading or writing data. It is TRUE if the port is writing data, it is FALSE if the port is reading data.
Error	BOOL	FALSE	Whether data reception encountered an error. It is TRUE if an error occurs in data reception (including buffer overflow, connection timeout, and data reception error)
DataToRecv	ARRAY[0..255] OF BYTE	-	Received data

4.12.2.5 Variable Declaration

VAR

```

EX:BOOL;

abort:BOOL;

BUF:BOOL;

PORT:UINT;

quantity_re:UINT;

Time1:TIME;

count:UDINT;

check1:STRING;

check2:STRING;

checke3:STRING;

done:BOOL;

aborted:BOOL;

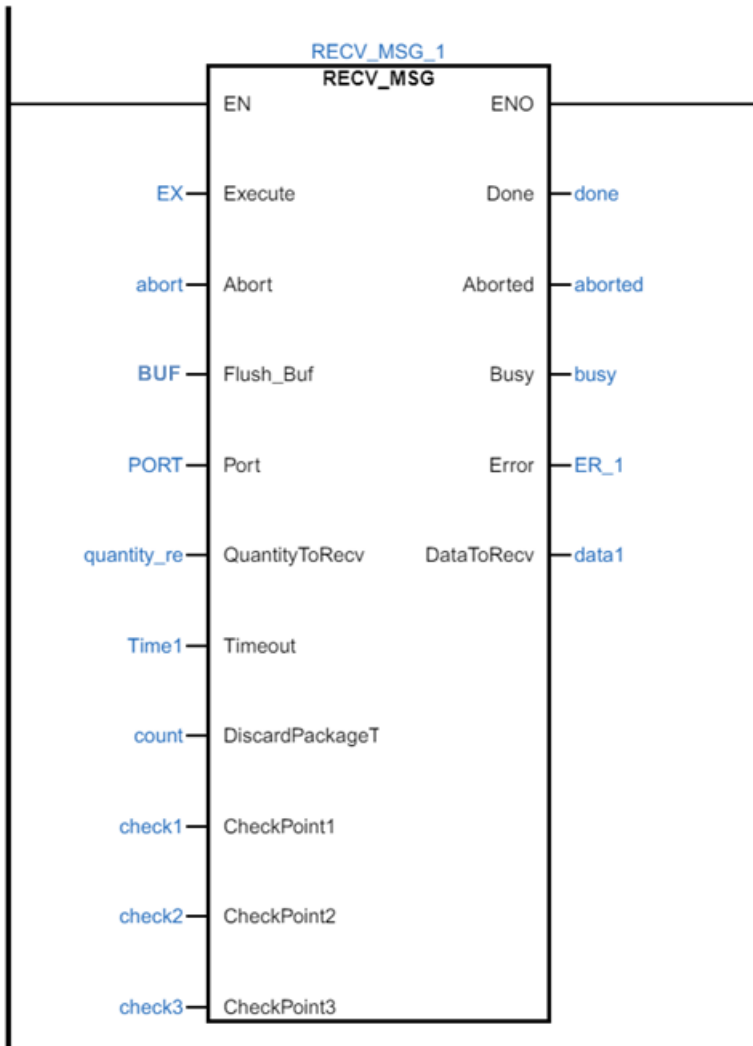
ER_1:BOOL;

data1:ARRAY[0..255] OF BYTE;

```

END_VAR

4.12.2.6 Expression in LD



4.12.2.7 Expression in ST

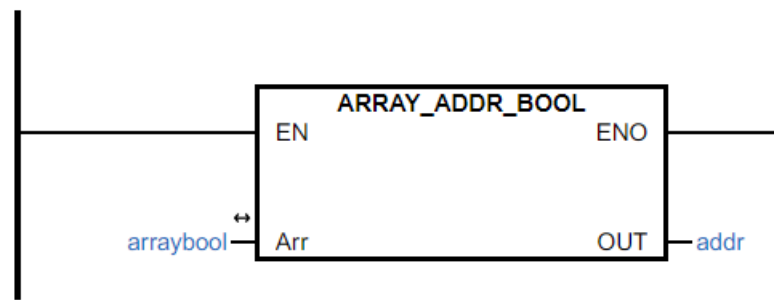
```
RECV_MSG_1(Execute:=EX,Abort:=abort,Flush_Buf:=BUF,Port:=PORT,QuantityToRecv:=quantity_re,Timeout:=Time1,DiscardPackageTimeout:=count,CheckPoint1:=check1,CheckPoint2:=check2,CheckPoint3:=check3,Done=>done,Aborted=>aborted,Busy=>busy,Error=>ER_1,DataToRecv=>data1);
```

4.12.3 ARRAY_ADDR_BOOL Get the Address of BOOL Array

4.12.3.1 Function Description

This function is used to get the address of a BOOL array and is used in tandem with the MODBUS_RW_BIT instruction.

4.12.3.2 Graphic



4.12.3.3 Input Parameter

Parameter	Data type	Initial value	Description
EN	BOOL	FALSE	Enable input, when EN is TRUE, this function is activated.
Arr	ARRAY[0..1999] OF BOOL	-	Data for serial port receiving/sending

4.12.3.4 Output Parameter

Parameter	Data type	Initial value	Description
ENO	BOOL	FALSE	Enable output, once EN is TURE, ENO is TRUE.
OUT	ARRAY_ADDR	-	Address of BOOL array

4.12.3.5 Variable Declaration

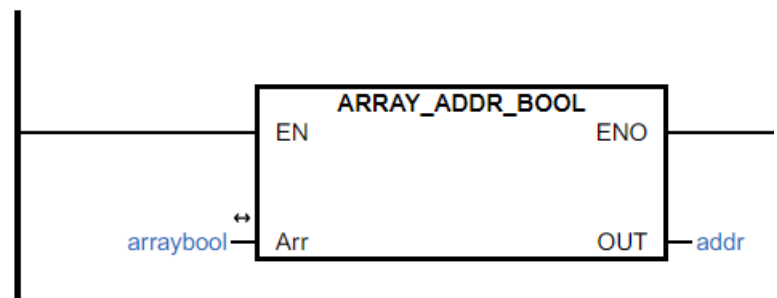
```
VAR

    arraybool : ARRAY [0..1999] OF BOOL;

    addr : ARRAY_ADDR;

END_VAR
```

4.12.3.6 Expression in LD



4.12.3.7 Expression in ST

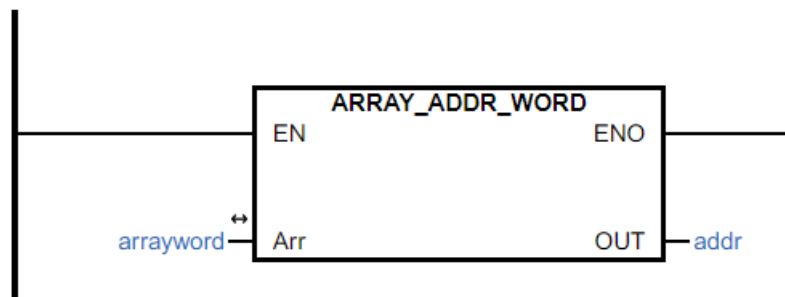
```
addr:=ARRAY_ADDR_BOOL(EN:=TRUE,Arr:=arraybool);
```

4.12.4 ARRAY_ADDR_WORD Get the Address of WORD Array

4.12.4.1 Function Description

This function is used to get the address of a WORD array and is used in tandem with the MODBUS_RW_WORD instruction. Its operator is ARRAU_ADDR_WORD.

4.12.4.2 Graphic



4.12.4.3 Input Parameter

Parameter	Data type	Initial value	Description
EN	BOOL	FALSE	Enable input, when EN is TRUE, this function is activated.
Arr	ARRAY[0..1999] OF WORD	-	Data for serial port receiving/sending

4.12.4.4 Output Parameter

Parameter	Data type	Initial value	Description
ENO	BOOL	FALSE	Enable output, once EN is TRUE, ENO is TRUE.
OUT	ARRAY_ADDR	-	Address of WORD array

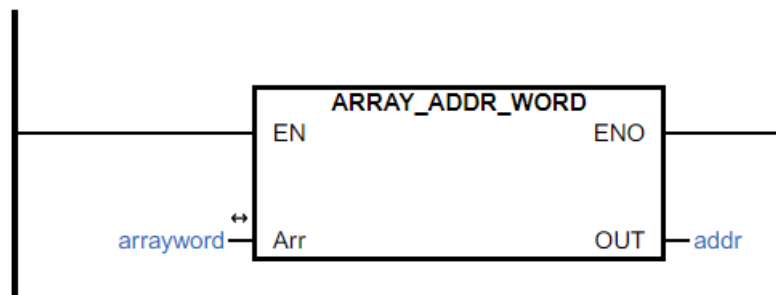
4.12.4.5 Variable Declaration

VAR

```
arrayword:ARRAY[0..119] OF WORD;
```

```
addr:ARRAY_ADDR;
```

4.12.4.6 Expression in LD



4.12.4.7 Expression in ST

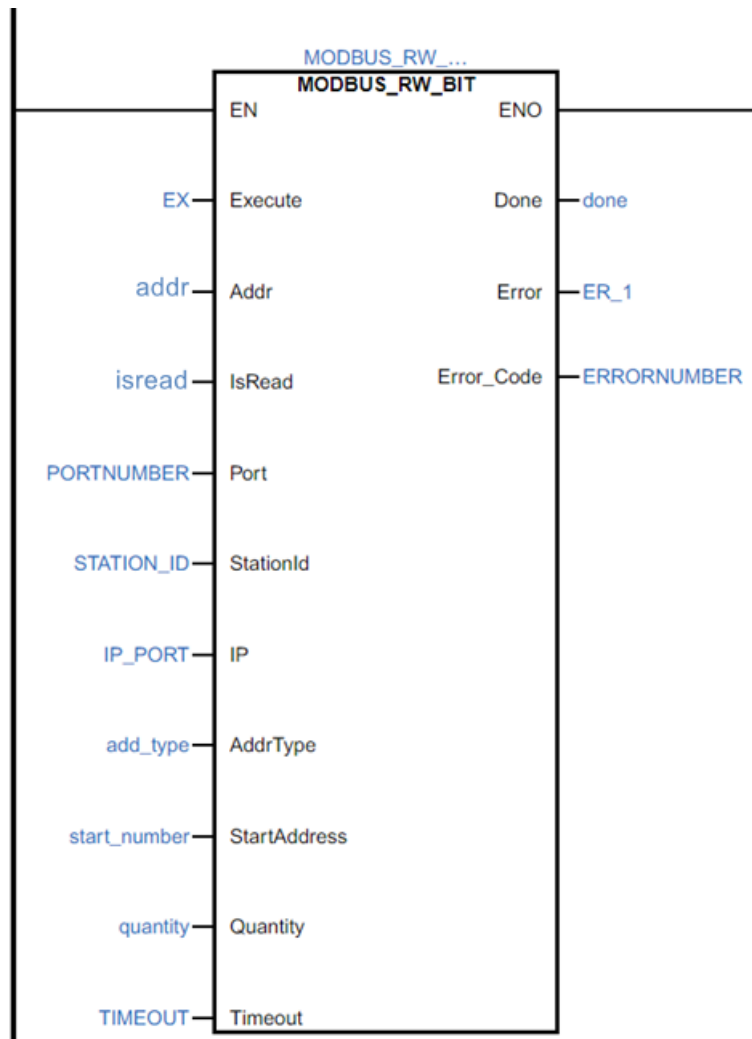
```
addr:=ARRAY_ADD_WORD(EN:=TRUE,Arr:=arrayword);
```

4.12.5 MODBUS_RW_BIT Serial (Ethernet) Port Send/ Receive Bit Data

4.12.5.1 Function Description

Receive/send bit data via the PLC's serial port or ethernet port. Its operator is MODBUS_RW_BIT.

4.12.5.2 Graphic



4.12.5.3 Input Parameter

Parameter	Data type	Initial value	Description
EN	BOOL	FALSE	Enable input, when EN is TRUE, this function clock is activated.
Execute	BOOL	FALSE	When it changed from FALSE to TRUE, the operation of receiving/sending data via serial port (Ethernet port) is executed.
Addr	ARRAY_ADDR	-	BOOL array address to be sent/received, converted using the ARRAY_ADDR_BOOL instruction.
IsRead	BOOL	FALSE	Is read or write Modbus data, It is TRUE when the port is reading data; FALSE when the port is writing

Parameter	Data type	Initial value	Description
			data.
Port	UINT	-	Valid values are 1, 2, 3, 4, and 10. 1 and 2 represent COM1 and COM2 on the CPU, 3 represents the serial port on extension board 1, 4 represents the serial port on extension board 2, and 10 represents the Ethernet port.
StationId	BYTE	0	Slave ID of serial communication. For Ethernet communication, please use the actual server setting. Default value is 0, for broadcast communication.
IP	STRING	-	IP and port of the Ethernet communication server, for example, 192.168.1.2:3442.
AddrType	BYTE	-	PLC address type, including 0X, 1X.
StartAddress	UINT	-	Modbus starting address of communication server, starting from 1.
Quantity	UINT	-	Length of communication data in bytes, ranging from 1 to 1920. Out-of-range values result in an error.
Timeout	UINT	0	Response timeout of the other party (device communicating with the PLC) in milliseconds

4.12.5.4 Output Parameter

Parameter	Data type	Initial value	Description
ENO	BOOL	FALSE	Enable output, once EN is TRUE, ENO is TRUE.
Done	BOOL	FALSE	Instruction execution result, It is TRUE if the instruction is executed successfully.
Error_Code	UINT	-	Communication error code, for detailed information about communication codes, please refer to Appendix A - Communication Code Description .
Error	BOOL	FALSE	Whether the instruction execution encounter an error. It is TRUE if an error occurs in the instruction execution.

4.12.5.5 Variable Declaration

VAR

EX:BOOL;

addr:ARRAY_ADDR;

isread:BOOL;

PORTNUMBER:BYTE;

STATION_ID:BYTE;

IP_PORT:STRING;

add_type:BYTE;

start_number:UINT;

quantity:UINT;

TIMEOUT:UDINT;

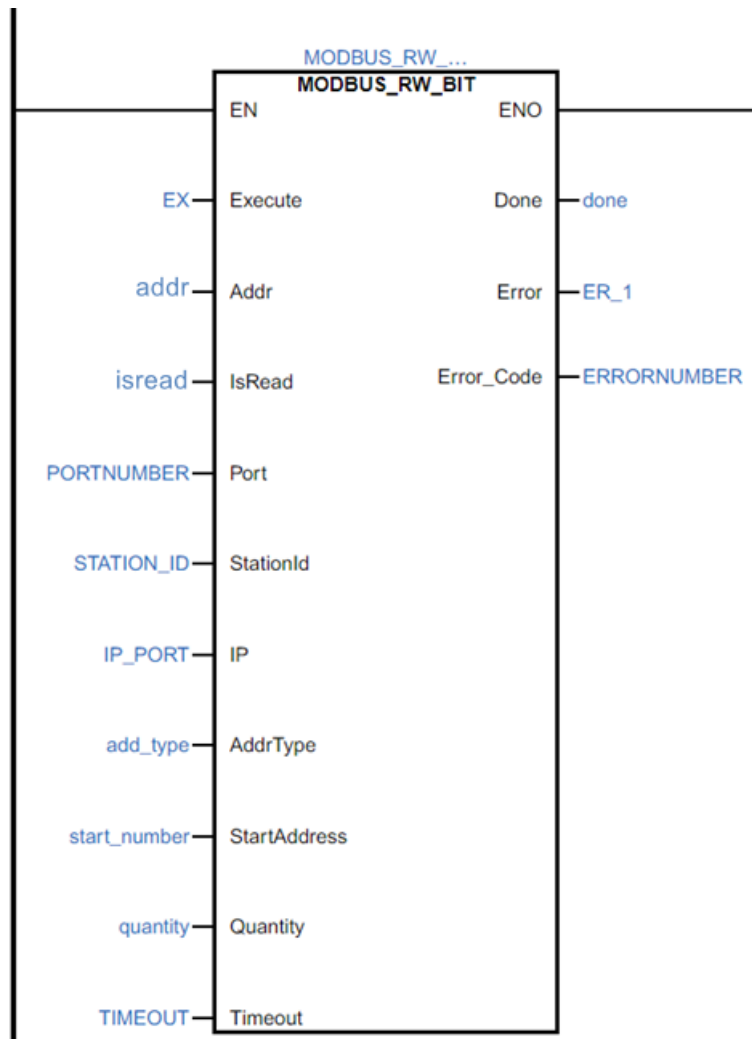
done:BOOL;

ERRORNUMBER:UINT;

ER_1:BOOL;

END_VAR

4.12.5.6 Expression in LD



4.12.5.7 Expression in ST

```
MODBUS_RW_BIT_1(EN:=TRUE,Execute:=EX,Addr:=addr,IsRead:=isread,Port:=PORTNUMBER,StationId:=
STATION_ID,IP:=IP_PORT,AddrType:=add_type,StartAddress:=start_number,Quantity:=quantity,Timeout:=TIM
EOUT, Done=>done,Error_Code=>ERRORNUMBER,Error=>ER_1);
```

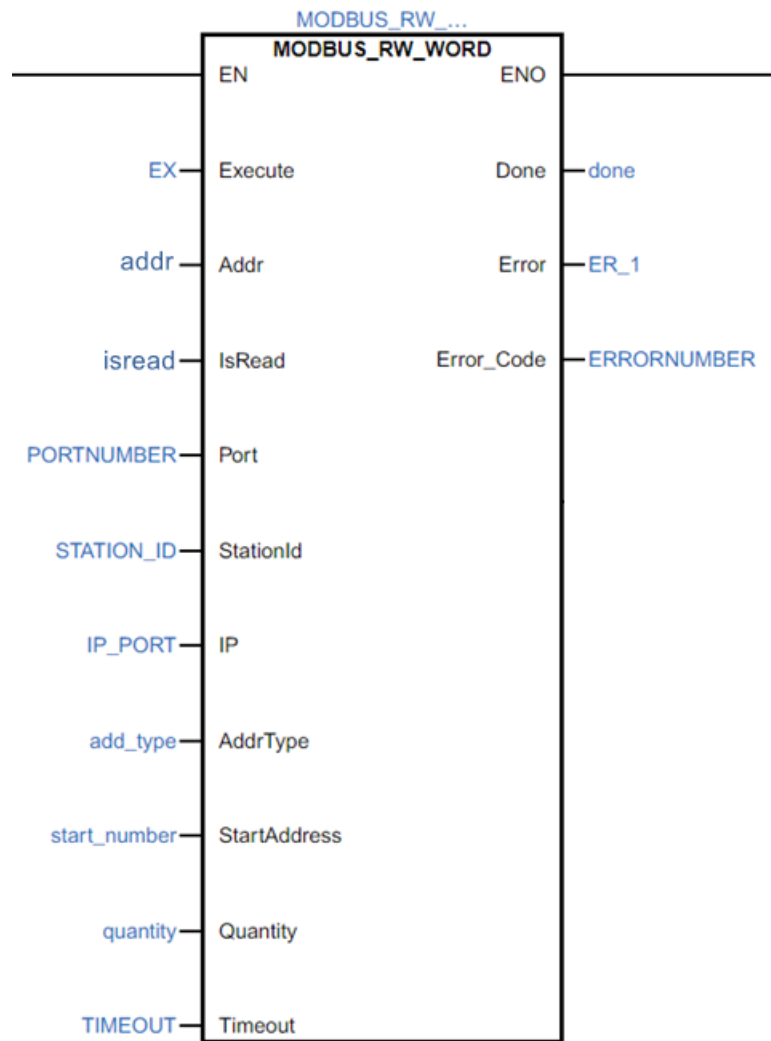
4.12.6 MODBUS_RW_WORD Serial(Ethernet) Port Send/Receive

WORD Data

4.12.6.1 Function Description

Receiving/sending WORD data via the PLC's serial port or ethernet port. Its operator is MODBUS_RW_WORD.

4.12.6.2 Graphic



4.12.6.3 Input Parameter

Parameter	Data type	Initial value	Description
EN	BOOL	FALSE	Enable input, when EN is TRUE, this function clock is activated.
Execute	BOOL	FALSE	When it changes from FALSE to TRUE, the operation of receiving/sending data via serial port (Ethernet port) is executed.
Addr	ARRAY_ADDR	-	WORD array address to be sent/received, converted using the ARRAY_ADDR_WORD instruction.
IsRead	BOOL	FALSE	Is read or write Modbus data, It is TRUE when the port is reading data; FALSE when the port is writing

Parameter	Data type	Initial value	Description
			data.
Port	UINT	-	Valid values are 1, 2, 3, 4, and 10. 1 and 2 represent COM1 and COM2 on the CPU, 3 represents the serial port on extension board 1, 4 represents the serial port on extension board 2, and 10 represents the Ethernet port.
StationId	BYTE	0	Slave ID of serial communication. For Ethernet communication, please use the actual server setting. Default value is 0, for broadcast communication.
IP	STRING	-	IP and port of the Ethernet communication server, for example, 192.168.1.2:3442.
AddrType	BYTE	-	PLC address type, including 3X, 4X.
StartAddress	UINT	-	Modbus starting address of communication server, starting from 1.
Quantity	UINT	-	Length of communication data in bytes, ranging from 1 to 1920. Out-of-range values result in an error.
Timeout	UINT	0	Response timeout of the other party (device communicating with the PLC) in milliseconds

4.12.6.4 Output Parameter

Parameter	Data type	Initial value	Description
ENO	BOOL	FALSE	Enable output, once EN is TRUE, ENO is TRUE.
Done	BOOL	FALSE	Instruction execution result, It is TRUE if the instruction is executed successfully.
Error_Code	UINT	-	Communication error code, for detailed information about communication codes, please refer to Appendix A - Communication Code Description .
Error	BOOL	FALSE	Whether the instruction execution encounter an error. It is TRUE if an error occurs in the instruction execution.

4.12.6.5 Variable Declaration

VAR

EX:BOOL;

addr:ARRAY_ADDR;

isread:BOOL;

PORTNUMBER:BYTE;

STATION_ID:BYTE;

IP_PORT:STRING;

add_type:BYTE;

start_number:UINT;

quantity:UINT;

TIMEOUT:UDINT;

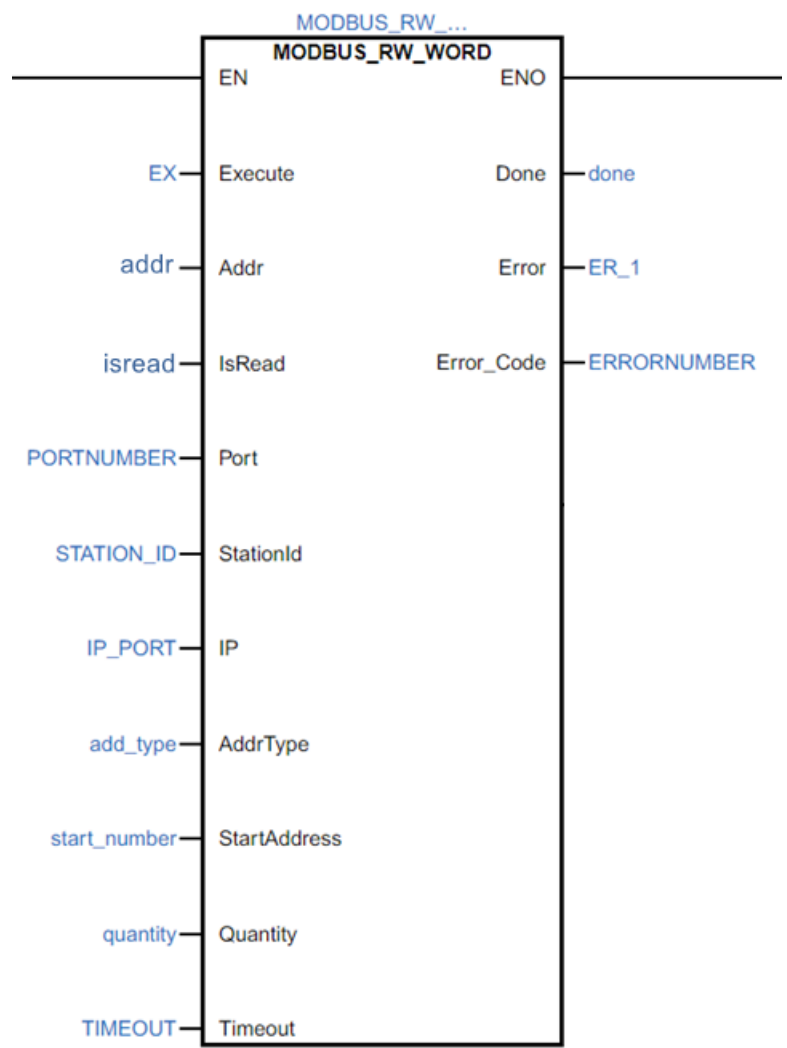
done:BOOL;

ERRORNUMBER:UINT;

ER_1:BOOL;

END_VAR

4.12.6.6 Expression in LD



4.12.6.7 Expression in ST

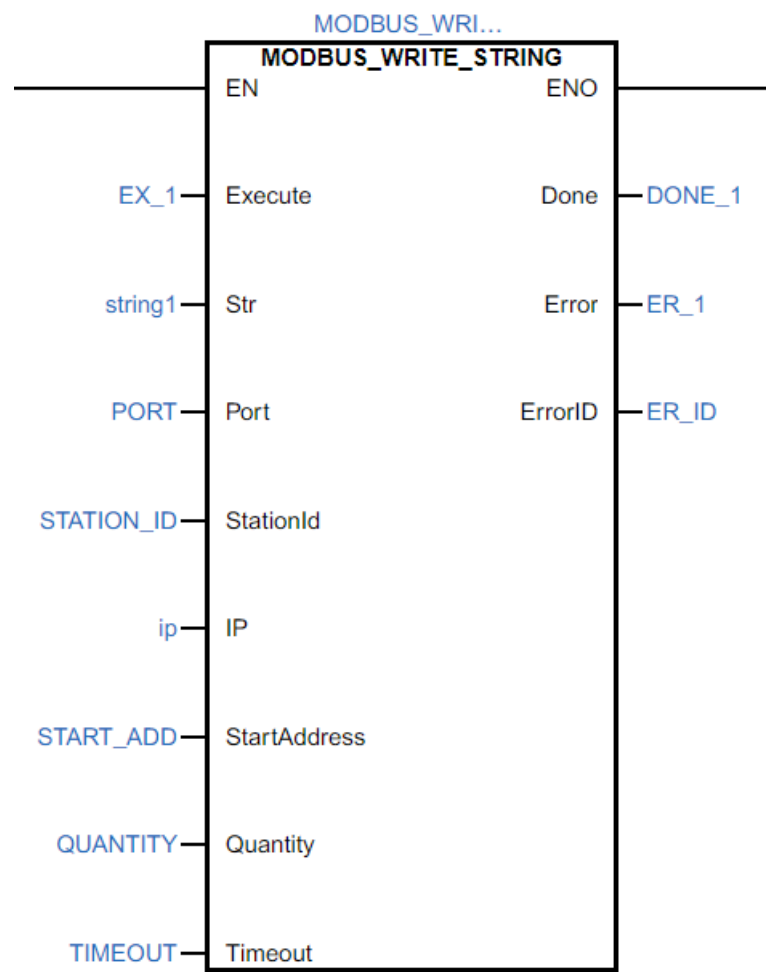
```
MODBUS_RW_WORD_1(EN:=TRUE,Execute:=EX,Addr:=addr,IsRead:=isread,Port:=PORTNUMBER,StationId:=STATION_ID,IP:=IP_PORT,AddrType:=add_type,StartAddress:=start_number,Quantity:=quantity,Timeout:=TIMEOUT,Done=>done,Error_Code=>ERRORNUMBER,Error=>ER_1);
```

4.12.7 MODBUS_WRITE_STRING Write String Data in Modbus Communication

4.12.7.1 Function Description

This function block is used for string type data writing in Modbus Communication. Its operator is MODBUS_WRITE_STRING.

4.12.7.2 Graphic



4.12.7.3 Input Parameter

Parameter	Data type	Initial value	Description
EN	BOOL	FALSE	Enable input, when EN is TRUE, this function block is activated.
Execute	BOOL	FALSE	When it changes from FALSE to TRUE, the operation of writing a string via serial port (Ethernet port) is executed.
Str	STRING	-	The string to be written.
Port	UINT	-	Valid values are 1, 2, 3, 4, and 10. 1 and 2 represent COM1 and COM2 on the CPU, 3 represents the serial port on extension board 1, 4 represents the serial port on extension board 2, and 10 represents

Parameter	Data type	Initial value	Description
			the Ethernet port.
StationId	BYTE	0	Slave ID of serial communication. For Ethernet communication, please use the actual server setting. Default value is 0, for broadcast communication.
IP	STRING	-	IP and port of the Ethernet communication server, for example, 192.168.1.2:3442.
StartAddress	UINT	-	Modbus starting address of communication server, starting from 1.
Quantity	UINT	-	Length of communication data in bytes, ranging from 1 to 120. Out-of-range values result in an error.
Timeout	UINT	0	Response timeout of the other party (device communicating with the PLC) in milliseconds

4.12.7.4 Output Parameter

Parameter	Data type	Initial value	Description
ENO	BOOL	FALSE	Enable output, once EN is TRUE, ENO is TRUE.
Done	BOOL	FALSE	Instruction execution result, It is TRUE if the instruction is executed successfully.
Error	BOOL	FALSE	Whether the instruction execution encounter an error. It is TRUE if an error occurs in the instruction execution.
ErrorID	UINT	-	Communication error code, for detailed information about communication codes, please refer to Appendix A - Communication Code Description .

4.12.7.5 Variable Declaration

VAR

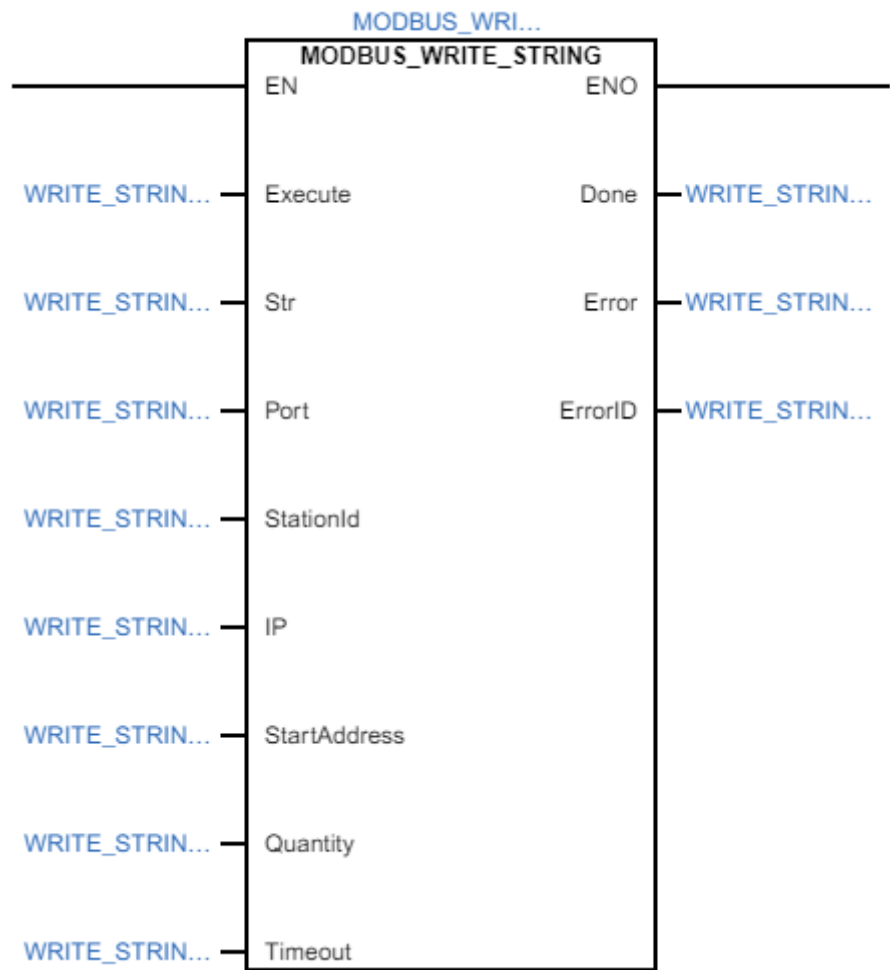
MODBUS_WRITE_STRING_1 : MODBUS_WRITE_STRING;

WRITE_STRING_Execute : BOOL;

WRITE_STRING_Str : STRING ;

```
WRITE_STRING_Port : BYTE;  
  
WRITE_STRING_StationId : BYTE;  
  
WRITE_STRING_IP : STRING;  
  
WRITE_STRING_StartAddress : UINT;  
  
WRITE_STRING_Quantity : UINT;  
  
WRITE_STRING_Timeout : UDINT;  
  
WRITE_STRING_Done : BOOL;  
  
WRITE_STRING_Error : BOOL;  
  
WRITE_STRING_ErrorID : UINT;  
  
END_VAR
```

4.12.7.6 Expression in LD



4.12.7.7 Expression in ST

```
MODBUS_WRITE_STRING_1(EN := TRUE (*BOOL*), Execute := WRITE_STRING_Execute (*BOOL*),  
Str := WRITE_STRING_Str (*STRING*), Port := WRITE_STRING_Port (*BYTE*), StationId :=  
WRITE_STRING_StationId (*BYTE*), IP := WRITE_STRING_IP (*STRING*), StartAddress :=  
WRITE_STRING_StartAddress (*UINT*), Quantity := WRITE_STRING_Quantity (*UINT*), Timeout :=  
WRITE_STRING_Timeout (*UDINT*), ENO => WRITE_STRING_ENO (*BOOL*), Done =>  
WRITE_STRING_Done (*BOOL*), Error => WRITE_STRING_Error (*BOOL*), ErrorID =>  
WRITE_STRING_ErrorID (*UINT*));
```

4.12.7.8 Usage Restriction

When using this instruction, the port corresponding to the Port parameter in the **configuration** needs to be enabled. Set the mode to “Master” and select “Function Block” for the communication address configuration method.

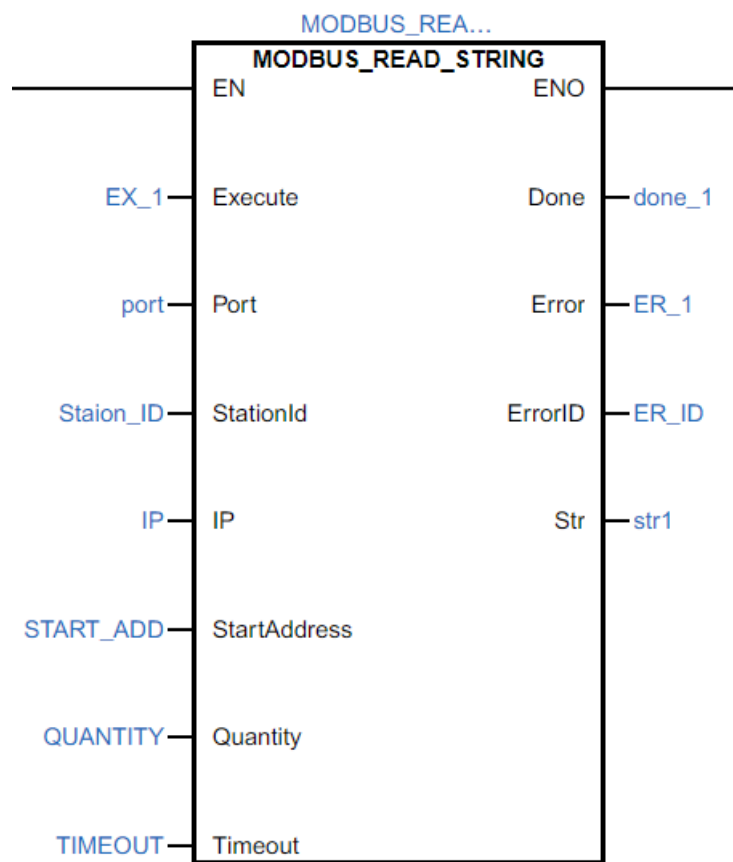
4.12.8 MODBUS_READ_STRING Read String Data in Modbus

Communication

4.12.8.1 Function Description

This Function block is used for reading String data in Modbus communication. Its operator is MODBUS_READ_STRING.

4.12.8.2 Graphic



4.12.8.3 Input Parameter

Parameter	Data type	Initial value	Description
EN	BOOL	FALSE	Enable input, when EN is TRUE, this function clock is activated.
Execute	BOOL	FALSE	When it changes from FALSE to TRUE, the operation of reading a string via serial port

Parameter	Data type	Initial value	Description
			(Ethernet port) is executed.
Port	UINT	-	Valid values are 1, 2, 3, 4, and 10. 1 and 2 represent COM1 and COM2 on the CPU, 3 represents the serial port on extension board 1, 4 represents the serial port on extension board 2, and 10 represents the Ethernet port.
StationId	BYTE	0	Slave ID of serial communication. For Ethernet communication, please use the actual server setting. Default value is 0, for broadcast communication.
IP	STRING	-	IP and port of the Ethernet communication server, for example, 192.168.1.2:3442.
StartAddress	UINT	-	Modbus starting address of communication server, starting from 1.
Quantity	UINT	-	Length of communication data in bytes, ranging from 1 to 120. Out-of-range values result in an error.
Timeout	UINT	0	Response timeout of the other party (device communicating with the PLC) in milliseconds

4.12.8.4 Output Parameter

Parameter	Data type	Initial value	Description
ENO	BOOL	FALSE	Enable output, once EN is TRUE, ENO is TRUE.
Done	BOOL	FALSE	Instruction execution result, It is TRUE if the instruction is executed successfully.
Error	BOOL	FALSE	Whether the instruction execution encounter an error. It is TRUE if an error occurs in the instruction execution.
ErrorID	UINT	-	Communication error code, for detailed information about communication codes, please refer to Appendix A - Communication Code Description .
Str	STRING	-	The string read.

4.12.8.5 Variable Description

VAR

MODBUS_READ_STRING_1 : MODBUS_READ_STRING;

READ_STRING_Execute : BOOL;

READ_STRING_Port : BYTE;

READ_STRING_Stationid : BYTE;

READ_STRING_ip : STRING;

READ_STRING_StartAddress : UINT;

READ_STRING_Quantity : UINT;

READ_STRING_Tiemout : UDINT;

READ_STRING_Done : BOOL;

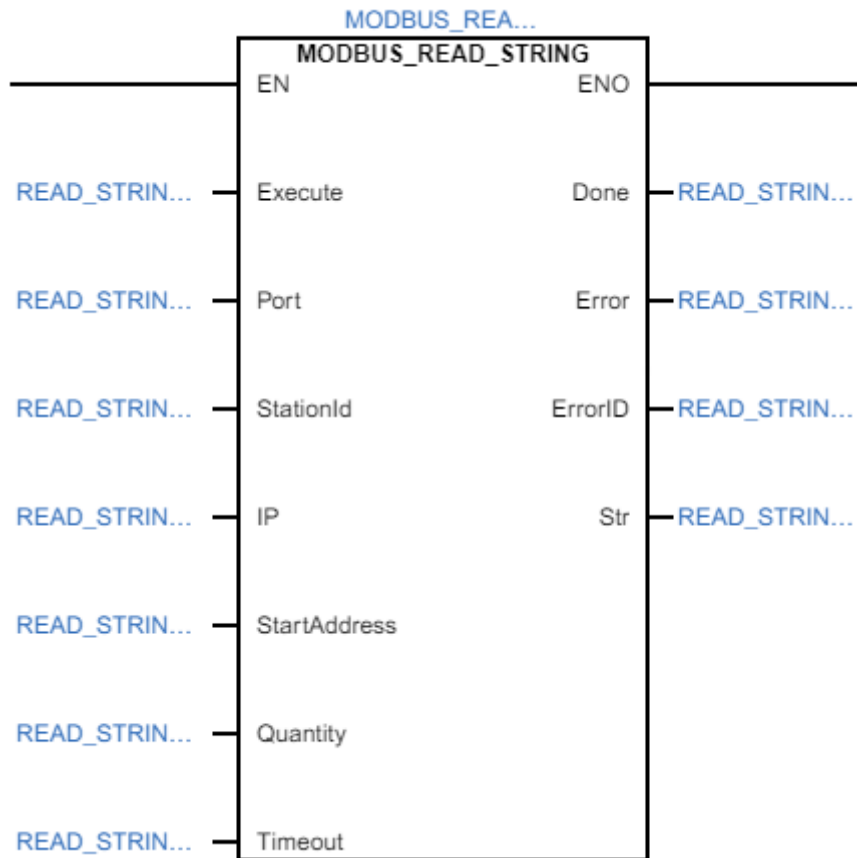
READ_STRING_Error : BOOL;

READ_STRING_ErrorID : UINT;

READ_STRING_Str : STRING;

END_VAR

4.12.8.6 Expression in LD



4.12.8.7 Expression in ST

```
MODBUS_READ_STRING_1(EN := TRUE (*BOOL*), Execute := READ_STRING_Execute (*BOOL*), Port :=
READ_STRING_Port (*BYTE*), StationId := READ_STRING_Stationid (*BYTE*), IP := READ_STRING_ip
(*STRING*), StartAddress := READ_STRING_StartAddress (*UINT*), Quantity := READ_STRING_Quantity
(*UINT*), Timeout := READ_STRING_Tiemout (*UDINT*), ENO => READ_STRING_ENO (*BOOL*), Done
=> READ_STRING_Done (*BOOL*), Error => READ_STRING_Error (*BOOL*), ErrorID =>
READ_STRING_ErrorID (*UINT*), Str => READ_STRING_Str (*STRING*));
```

4.12.8.8 Usage Restriction

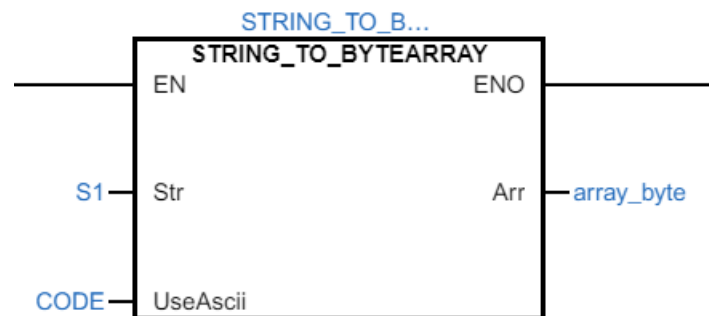
When using this instruction, the port corresponding to the Port parameter in the **configuration** needs to be enabled. Set the mode to “Master” and select “Function Block” for the communication address configuration method.

4.12.9 STRING_TO_BYTEARRAY Convert String to BYTE Array

4.12.9.1 Function Description

Convert a string to a BYTE array. Its operator is STRING_TO_BYTEARRAY.

4.12.9.2 Graphic



4.12.9.3 Input Parameter

Parameter	Data type	Description
EN	BOOL	Enable input, when EN is TRUE, this function block is activated.
Str	STRING	Input string
Use Ascii	BOOL	The conversion method of the string. When it is TRUE, the input string is converted to an ASCII string before being converted into an array. When it is FALSE, the input string is converted to a hexadecimal string before being converted into an array.

4.12.9.4 Output Parameter

Parameter	Data type	Description
ENO	BOOL	Enable output, once EN is TRUE, ENO is TRUE.
Arr	ARRAY[0..255] of BYTE	Output BYTE array

4.12.9.5 Variable Declaration

```
VAR
    STRING_TO_BYTEARRAY_1:STRING_TO_BYTEARRAY;
```

```

S1:STRING;

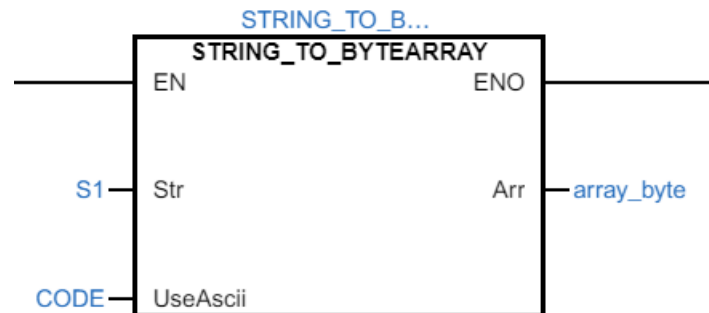
CODE:BOOL;

array_byte:ARRAY[0..255] OF BYTE;

END_VAR

```

4.12.9.6 Expression in LD



4.12.9.7 Expression in ST

```

Array_byte:=STRING_TO_BYTEARRAY_1(EN:=TRUE,Str:=S1,UseAscii:=CODE);

```

4.12.9.8 Usage Restriction

The BYTE array supports a maximum of 256 elements.

4.12.9.9 Cautions

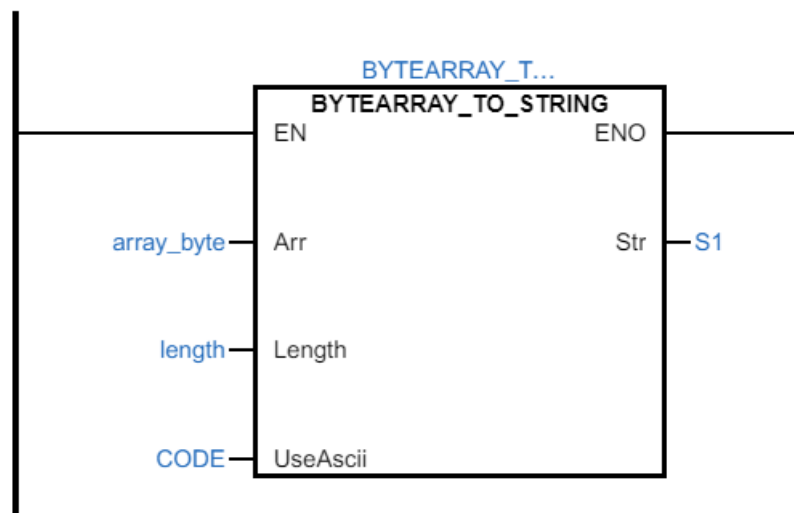
There is only one set of data receiving actions in a program running at the same time to take effect, so as to avoid data conflicts.

4.12.10 BYTEARRAY_TO_STRING Convert BYTE Array to String

4.12.10.1 Function Description

Convert a BYTE array to a string. Its operator is **BYTEARRAY_TO_STRING**.

4.12.10.2 Graphic



4.12.10.3 Input Parameter

Parameter	Data type	Description
EN	BOOL	Enable input, when EN is TRUE, this function block is activated.
Arr	ARRAY[0..255] OF BYTE	BYTE array
Length	UINT	Length of the string to be converted.
UseAscii	BOOL	Conversion method. When it is TRUE, convert BYTE array to ASCII string; when it is FALSE, convert BYTE array to hexadecimal string.

4.12.10.4 Output Parameter

Parameter	Data type	Description
ENO	BOOL	Enable output, once EN is TRUE, ENO is TRUE.
Str	STRING	Converted string

4.12.10.5 Variable Declaration

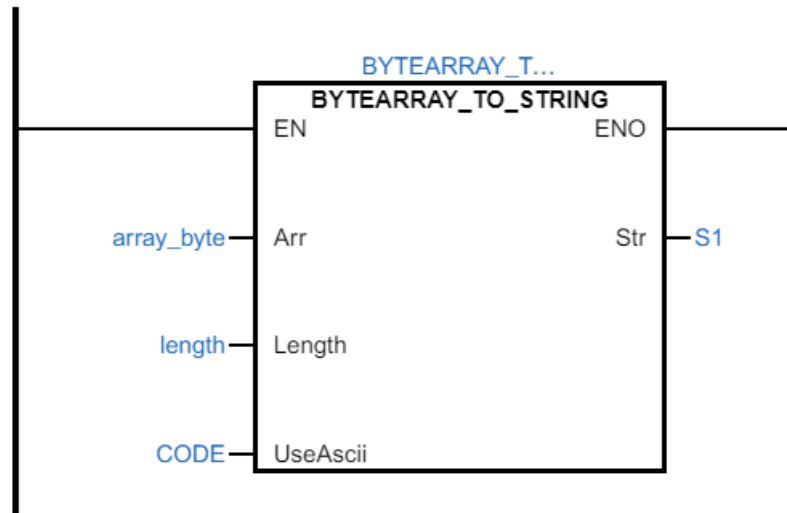
```
VAR
    BYTEARRAY_TO_STRING_1:BYTEARRAY_TO_STRING;
    array_byte:ARRAY[0..255] OF BYTE;
    length:UINT;
```

```
CODE:BOOL;
```

```
S1:STRING;
```

```
END_VAR
```

4.12.10.6 Expression in LD



4.12.10.7 Expression in ST

```
BYTEARRAY_TO_STRING_1(EN:=TRUE,Arr:=array_byte,Length:=length,UseAscii:=CODE,Str=>S1);
```

4.12.10.8 Usage Restriction

The BYTE array supports a maximum of 256 elements.

4.12.10.9 Cautions

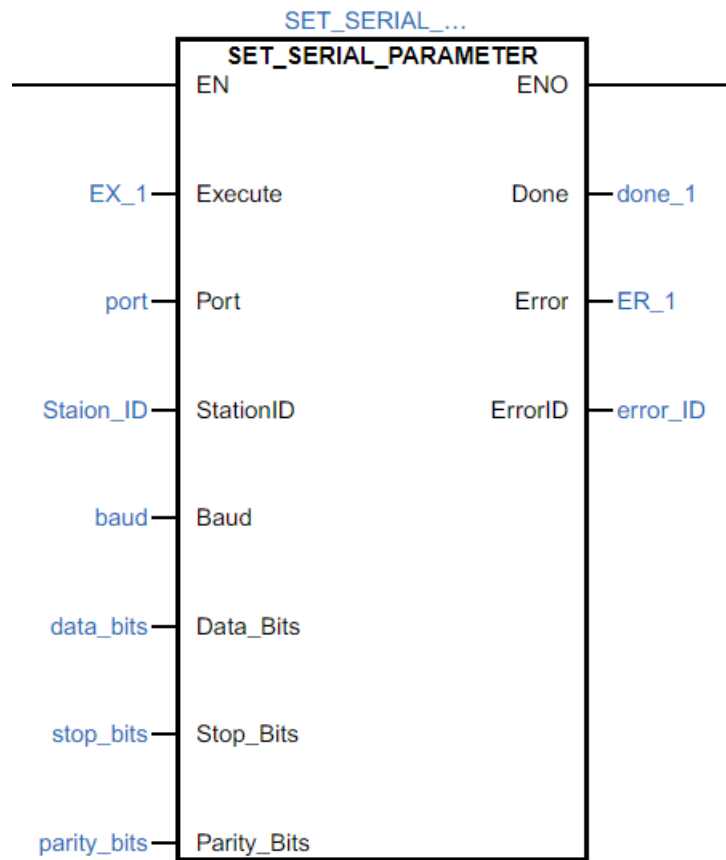
There is only one set of data receiving actions in a program running at the same time to take effect, so as to avoid data conflicts.

4.12.11 SET_SERIAL_PARAMETER Set Serial Port Parameter

4.12.11.1 Function Description

Used for serial port parameter configuration. Its operator is SET_SERIAL_PARAMETER.

4.12.11.2 Graphic



4.12.11.3 Input Parameter

Parameter	Data type	Initial value	Description
EN	BOOL	FALSE	Enable input, when EN is TURE, this function block is activated.
Execute	BOOL	FALSE	Execution switch. When it changes from FALSE to TRUE, the operation of modifying the serial port parameter configuration is executed.
Port	BYTE	-	Used serial port. Legal values are 1, 2, 3, and 4. 1 and 2 represent COM1 and COM2 on the CPU, 3 represents the serial port on extension board 1, and 4 represents the serial port on extension board 2.
StationId	BYTE	-	Slave station number of serial communication. Ranges from 1 to 247. Leave it empty if no modification is needed.
Baud	UINT	-	Serial port baud rate, optional parameter. Leave it empty if

Parameter	Data type	Initial value	Description
			no modification is needed.
Data_Bits	BYTE	-	Data bits, optional value is 8. Optional parameter, leave it empty if no modification is needed.
Stop_Bits	BYTE	-	Stop bits, optional values are 1 or 2. Optional parameter. Leave it empty if no modification is needed
Parity_Bits	BYTE	-	Parity check, where 0 is no check, 1 is odd check, 2 is even check, 3 is mark, and 4 is space.

4.12.11.4 Output Parameter

Parameter	Data type	Initial value	Description
ENO	BOOL	FALSE	Enable output, once EN is TRUE, ENO is TRUE.
Done	BOOL	FALSE	Instruction execution result. It is TRUE if the instruction is executed successfully.
Error	BOOL	FALSE	Whether the instruction execution encounter an error. It is TRUE when an error occurs in the instruction execution.
ErrorID	BYTE	-	Error code (1: Incorrect serial port number or the serial port is not enabled; 2: Incorrect slave station number; 3: Incorrect data bits; 4: Incorrect stop bits; 5: Parity bit error; 6: Execution error)

4.12.11.5 Variable Declaration

```

VAR

  SET_SERIAL_PARAMETER_1 : SET_SERIAL_PARAMETER;

  EX_1 : BOOL;

  port : BYTE;

  Station_ID : BYTE;

  baud : UDINT;

  data_bits : BYTE;

  stop_bits : BYTE;

```

```

parity_bits : BYTE;

done_1 : BOOL;

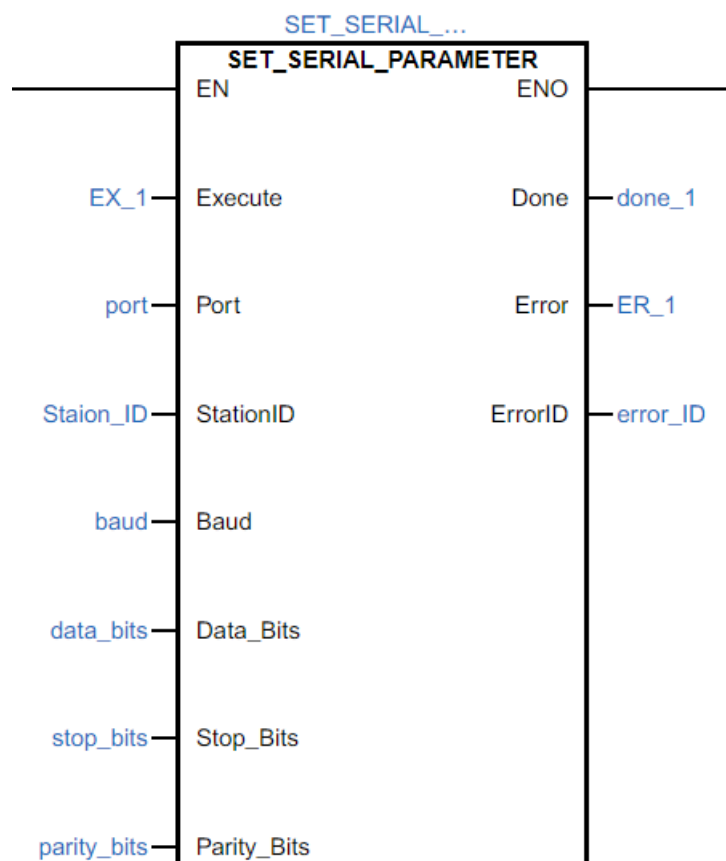
ER_1 : BOOL;

error_ID : BYTE;

END_VAR

```

4.12.11.6 Expression in LD



4.12.11.7 Expression in ST

```

SET_SERIAL_PARAMETER_1(EN := TRUE, Execute := EX_1, Port :=port, StationID := Station_ID,
Baud :=baud, Data_Bits := data_bits, Stop_Bits := stop_bits, Parity_Bits := parity_bits ,
Done=>done_1,Error=>ER_1,ErrorID=>error_ID);

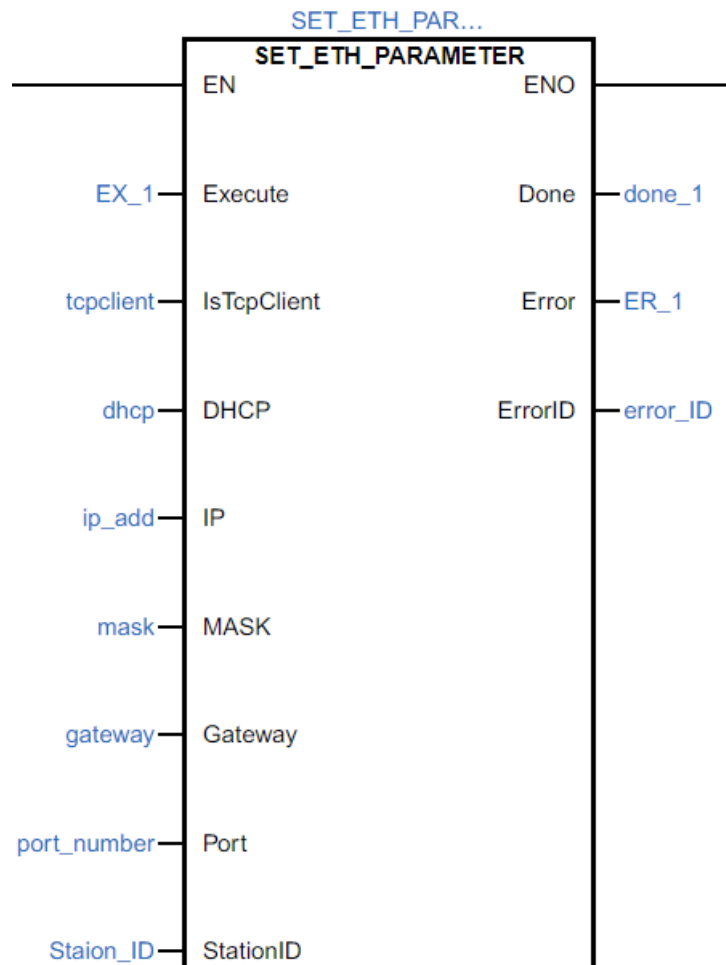
```

4.12.12 SET_ETH_PARAMETER Set Ethernet Port Parameter

4.12.12.1 Function Description

Used for Ethernet port parameter configuration. Its operator is SET_ETH_PARAMETER.

4.12.12.2 Graphic



4.12.12.3 Input Parameter

Parameter	Data type	Initial value	Description
EN	BOOL	FALSE	Enable input, when EN is TURE, this function block is activated.
Execute	BOOL	FALSE	Execution switch. When it changes from FALSE to TRUE, the operation of modifying the Ethernet port parameter configuration is executed.

Parameter	Data type	Initial value	Description
IsTcpClient	BOOL	FALSE	Differentiate between updating master settings and slave settings. When it is TRUE, update Tcp Client settings; when it is FALSE, update Tcp Server settings.
DHCP	BOOL	FALSE	Whether to enable DHCP functionality. If it is TRUE, DHCP is enabled.
IP	ARRAY[0..3] OF BYTE	-	Updated IP address. A one-dimensional array of 4 bytes. Optional parameter, leave it empty if no modification is needed.
MASK	ARRAY[0..3] OF BYTE	-	Updated subnet mask. A one-dimensional array of 4 bytes. Optional parameter, leave it empty if no modification is needed.
Gateway	ARRAY[0..3] OF BYTE	-	Updated gateway. A one-dimensional array of 4 bytes. Optional parameter, leave it empty if no modification is needed.
Port	UINT	-	The port number of the Modbus server. Optional parameter, leave it empty if no modification is needed.
StationID	BYTE	-	The station number of Modbus server. Valid values are 1 to 247. Leave it empty if no modification is needed.

4.12.12.4 Output Parameter

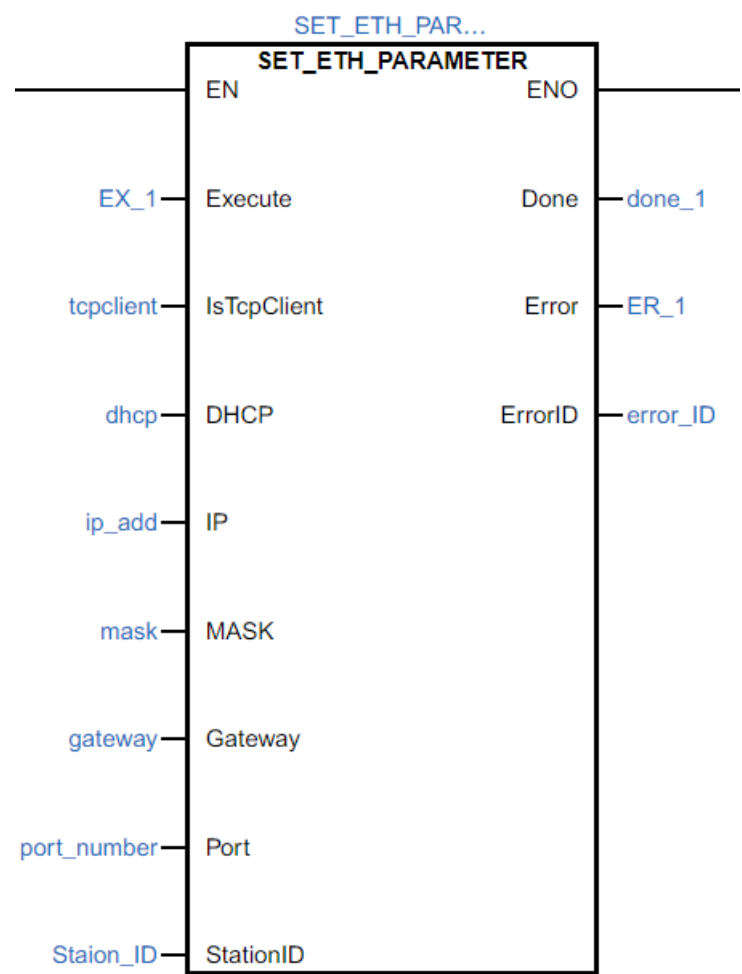
Parameter	Data type	Initial value	Description
ENO	BOOL	FALSE	Enable output, once EN is TRUE, ENO is TRUE.
Done	BOOL	FALSE	Instruction execution result. It is TRUE if the instruction is executed successfully
Error	BOOL	FALSE	Whether the instruction execution encounter an error. It is TRUE when an error occurs in the instruction execution.
ErrorID	BYTE	-	Error code (1: TCP Client/Server not enabled; 2: Incorrect slave station number; 3: Execution error)

4.12.12.5 Variable Declaration

VAR

```
SET_ETH_PARAMETER_1 : SET_ETH_PARAMETER;  
  
EX_1: BOOL;  
  
tcpclient : BOOL;  
  
dhcp : BOOL;  
  
ia_add : ARRAY [0..3] OF BYTE;  
  
mask : ARRAY [0..3] OF BYTE;  
  
gateway : ARRAY [0..3] OF BYTE;  
  
port_number : UDINT;  
  
Station_ID : BYTE;  
  
done_1 : BOOL;  
  
ER_1 : BOOL;  
  
error_ID : BYTE;  
  
END_VAR
```

4.12.12.6 Expression in LD

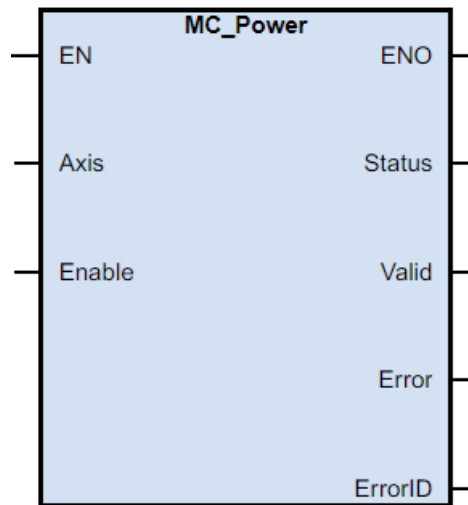


4.12.12.7 Expression in ST

```
SET_ETH_PARAMETER_1(EN := TRUE, Execute := EX_1, IsTcpClient := tcpclient, DHCP := dhcp, IP := ip_add,  
MASK := mask, Gateway := gateway, Port := port_number, StationID :=  
Station_ID,Done=>done_1,Error=>ER_1,ErrorID=>error_ID);
```

4.13 Motion Control

4.13.1 MC_Power



When EN is set to ON, the axis enters the enabled state, and the Status signal becomes effective. The axis's PLCOpen state machine transitions from the Disabled state to the StandStill state. Only after enabling, motion commands such as MC_MoveRelative can be executed.

After setting EN to OFF, the axis's enable state can be released, and interrupting motion commands (such as MC_MoveAbsolute) can be executed. After disabling the axis, it does not accept motion commands and cannot be controlled. However, non-motion commands such as MC_Power, MC_Reset, MC_SetPosition can still be executed.

When the axis enters the ErrorStop state due to a fault, re-enabling with MC_Power alone cannot transition the axis to the Standstill state. The fault must be recovered by calling the MC_Reset command at first.

Only one call to the MC_Power function block is allowed for each axis.

4.13.1.1 Input Parameter

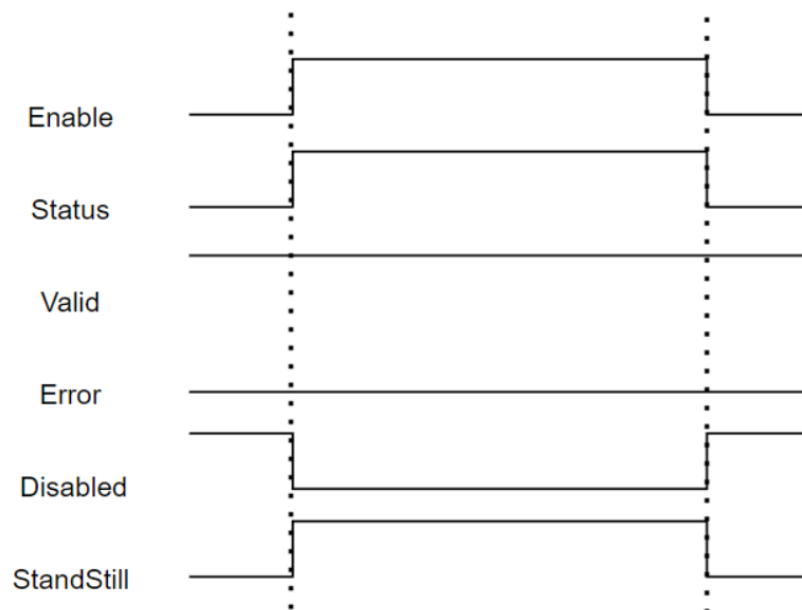
Parameter	Type	Initial value	Description
Axis	AXIS_REF	-	The name of the axis (instance) for executing the function block, customizable when adding the commands.
Enable	BOOL	FALSE	When the value is TRUE, it enables the axis specified by Axis; when the value is FALSE, it releases the enable.

4.13.1.2 Output Parameter

Parameter	Type	Initial value	Description
Status	BOOL	FALSE	When the value is TRUE, the axis specified by Axis is enabled.
Valid	BOOL	FALSE	When the value is TRUE, it indicates that there is a set of valid outputs on this function block.
Error	BOOL	FALSE	When the value is TRUE, it indicates that an error has been detected, and the function block execution is terminated.
ErrorID	WORD	-	For motion control error codes, please refer to Appendix C - Motion Control Error Codes for details.

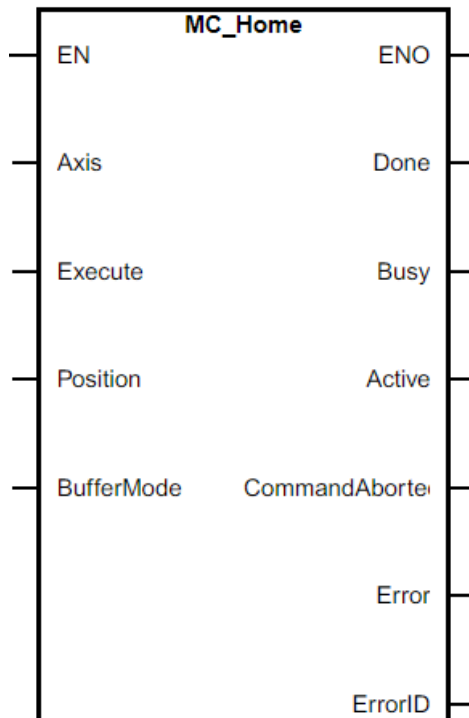
4.13.1.3 Sequency Diagram

When the MC_Power command is executed and Enable is True, the axis status is updated from Disabled to StandStill.



4.13.2 MC_Home

This function block initiates the axis to perform a homing operation, and transitions the axis state to Homing on the rising edge of Execute.



4.13.2.1 Input Parameter

Parameter	Type	Initial Value	Description
Axis	AXIS_REF	-	The name of the axis (instance) for executing the function block, customizable when adding the commands.
Execute	BOOL	FALSE	On the rising edge, initiate the execution of the function block. On the falling edge, reset the outputs of the function block when the execution terminates.
Position	REAL	0.0	After the homing operation is completed, set the current position to the specified absolute position.
BufferMode	MC_BUFFER_MODE	mcAborting	Buffer mode settings, refer to Buffering Mode for detailed information.

4.13.2.2 Output Parameter

Parameter	Type	Initial value	Description
Done	BOOL	FALSE	When the value is TRUE, the axis specified by Axis has completed homing.

Parameter	Type	Initial value	Description
Busy	BOOL	FALSE	When the value is TRUE, it indicates that this function block is being executed.
Active	BOOL	FALSE	The current function block controls Axis. For a defined Axis, only one function block can set Active to TRUE at a time.
CommandAborted	BOOL	FALSE	When the value is TRUE, it indicates that another motion command has been aborted or an error has been detected, terminating the execution of the function block.
Error	BOOL	FALSE	When the value is TRUE, it indicates that an error has been detected, and the function block execution is terminated.
ErrorID	WORD	-	For motion control error codes, please refer to Appendix C - Motion Control Error Codes for details.

At the rising edge of Execute, the function block latches the Position input parameter, and the axis is in the Homing state, performing a homing motion.

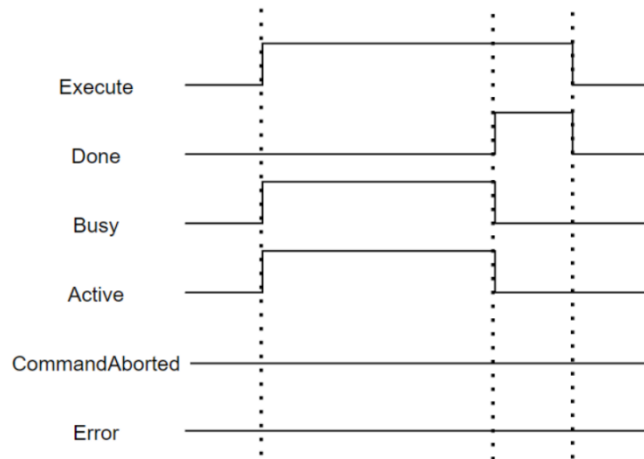
When calling this command in virtual axis mode, it will follow the absolute mode homing as specified in the CIA402 protocol, number 35.

In PLCOpen, the axis will be in the Homing state when the command is valid, and the MC_Stop command that can make the axis in the Stopping state can interrupt the command, and when the command is interrupted, CommandAbowart will be set to ON.

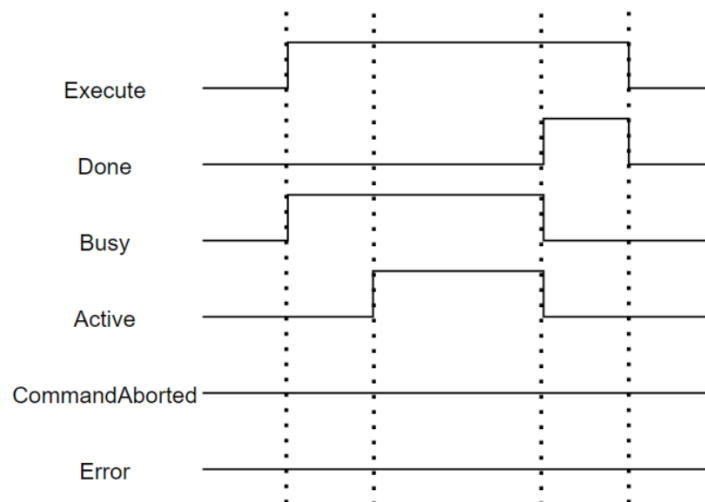
The MC_Power command is invalid when Execute=ON and the Done signal is invalid, which causes CommandAboarted to work when the axis is de-enabled.

4.13.2.3 Sequency Diagram

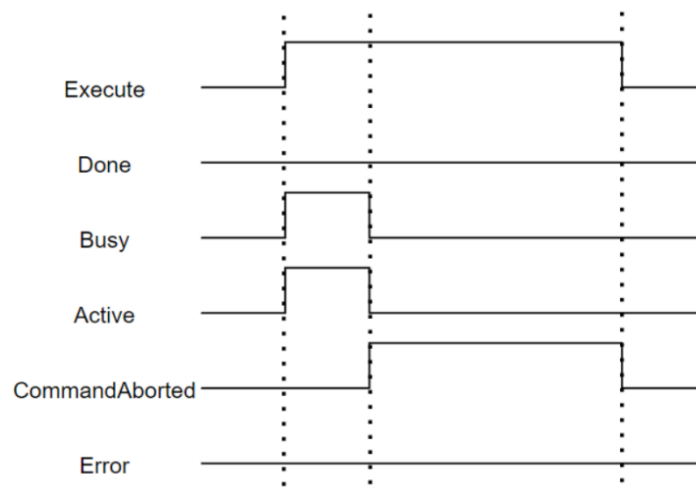
- ◆ Scenario 1: No other commands are executed, and the MC_Home command is executed, which ends normally



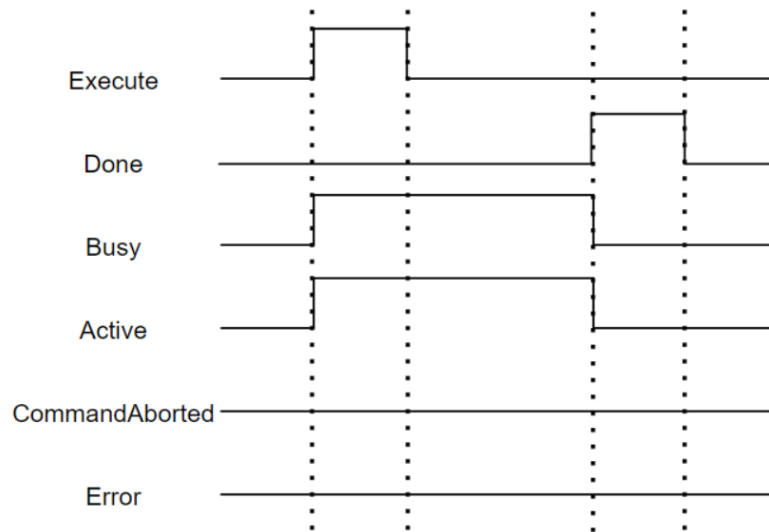
- ◆ Scenario 2: In the process of executing other commands, the motion command MC_Home is executed in mcBuffered mode, and the axis control needs to be obtained until the normal end is completed after the previous command is executed.



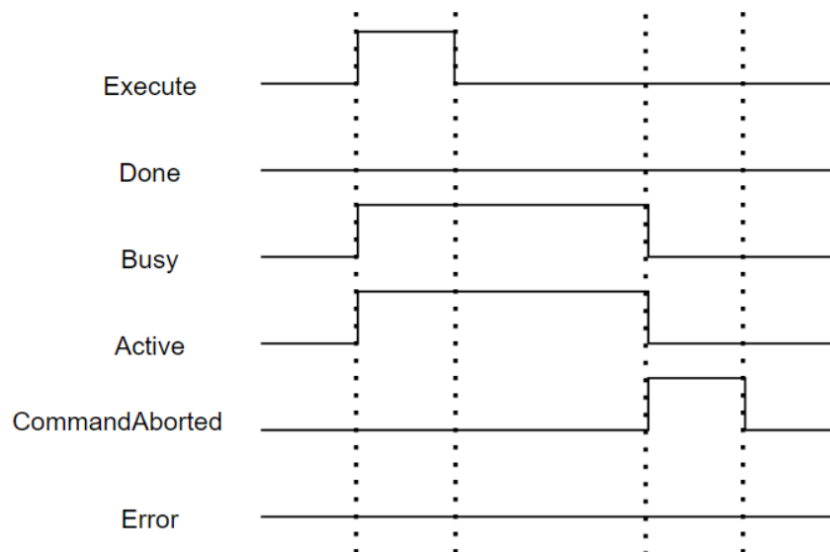
- ◆ Scenario 3: During the execution of the MC_Home command, it is preempted by other commands in the mode of mcAborting, so that the current command loses axis control, and the output pin of MC_Home CommandAborted is set to True.



- ◆ Scenario 4: During the execution of the MC_Home command, when the Execute is set to False during the movement, the axis continues to move to complete the current command, and the output pin Done of the MC_Home is set to TRUE for a scan cycle.



- ◆ Scenario 5: During the execution of the MC_Home command, when Execute is set to False during the movement, the axis is preempted by other commands in mcAborting when the axis continues to move, and the output pin CommandAborted of the MC_Home is set to True for a scan cycle.



4.13.3 MC_Stop

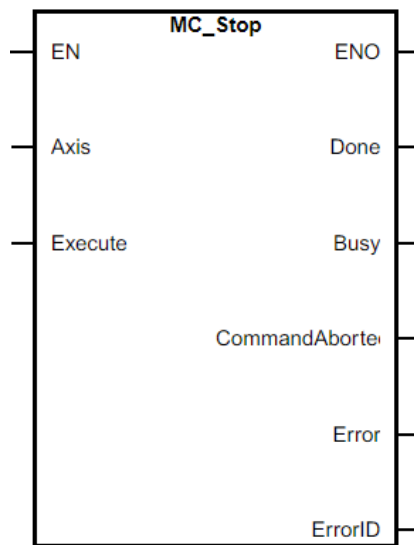
This function block commands the control of the motion to stop and converts the state of the axis to Stopping, with the Execute rising edge valid. It will abort any move execution in progress. When the axis is in the Stopping state, no other function block can perform any motion on the same axis.

On the rising edge of the Execute input, the function block latches the Deceleration input parameter and decelerates the motion, and the axis is in the Stopping state and decelerates the motion.

After the deceleration is complete, the Done signal is valid and remains in the Stopping state during Execute=ON.

When Execute=OFF and Done=ON, the axis switches from Stopping state to Standstill state.

This function block is mainly used to handle exceptions, or as a quick stop function.



4.13.3.1 Input Parameter

Parameter	Type	Initial value	Description
Axis	AXIS_REF	-	The name of the axis (instance) for executing the function block, customizable when adding the commands.
Execute	BOOL	FALSE	On the rising edge, initiate the execution of the function block. On the falling edge, reset the outputs of the function block when the execution ends.

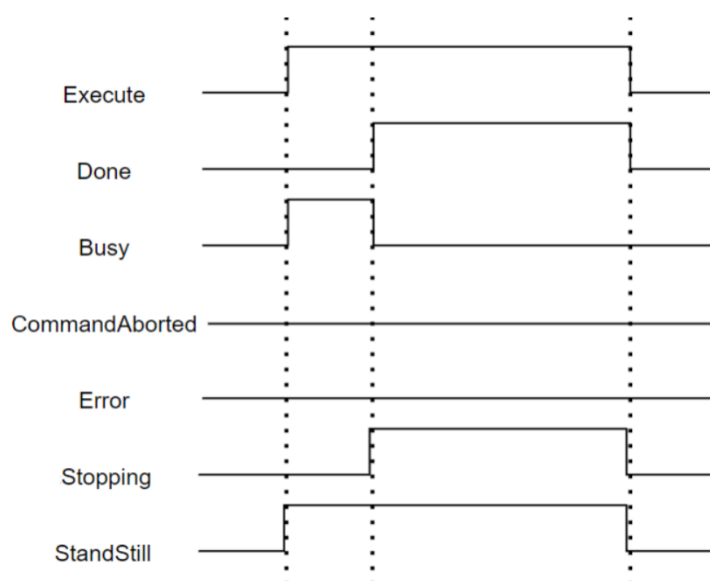
4.13.3.2 Output Parameter

Parameter	Type	Initial value	Description
Done	BOOL	-	When the value is TRUE, it indicates that the function block execution is completed, and no errors were detected.
Busy	BOOL	FALSE	When the value is TRUE, it indicates that this function block is being executed.
CommandAborted	BOOL	FALSE	When the value is TRUE, it indicates that another motion command has been aborted or an error has been detected, terminating the execution of the function block.
Error	BOOL	FALSE	When the value is TRUE, it indicates that an error has been

Parameter	Type	Initial value	Description
			detected, and the function block execution is terminated.
ErrorID	WORD	-	For motion control error codes, please refer to Appendix C - Motion Control Error Codes for details.

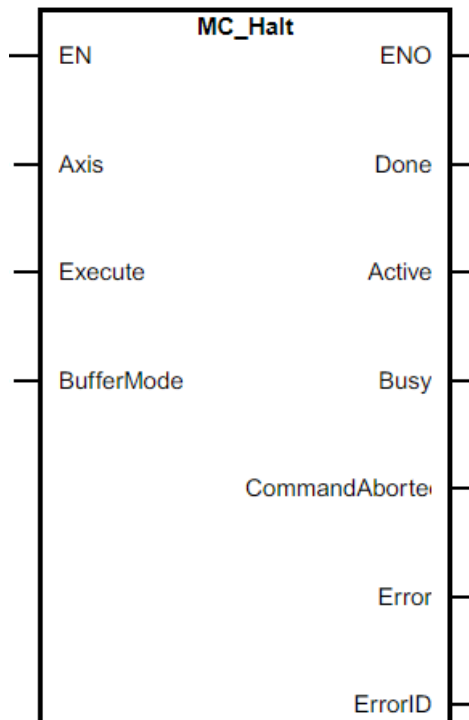
4.13.3 Sequency Diagram

Execute the MC_Stop command, triggered by the rising edge of Execute, updates the axis state to Stopping. After the execution is complete, set Execute back to FALSE, and the axis state is updated to StandStill.



4.13.4 MC_Halt

This function block commands the controlled movement to stop until the speed reaches zero and converts the state of the axis to Discrete. It takes effective at rising edge of Execute.



4.13.4.1 Input Parameter

Parameter	Type	Initial value	Description
Axis	AXIS_REF	-	The name of the axis (instance) for executing the function block, customizable when adding the commands.
Execute	BOOL	FALSE	On the rising edge, initiate the execution of the function block. On the falling edge, reset the outputs of the function block when the execution is terminated.
BufferMode	MC_BUFFER_MODE	mcAborting	Buffer mode setting, refer to Buffering Mode for detailed information.

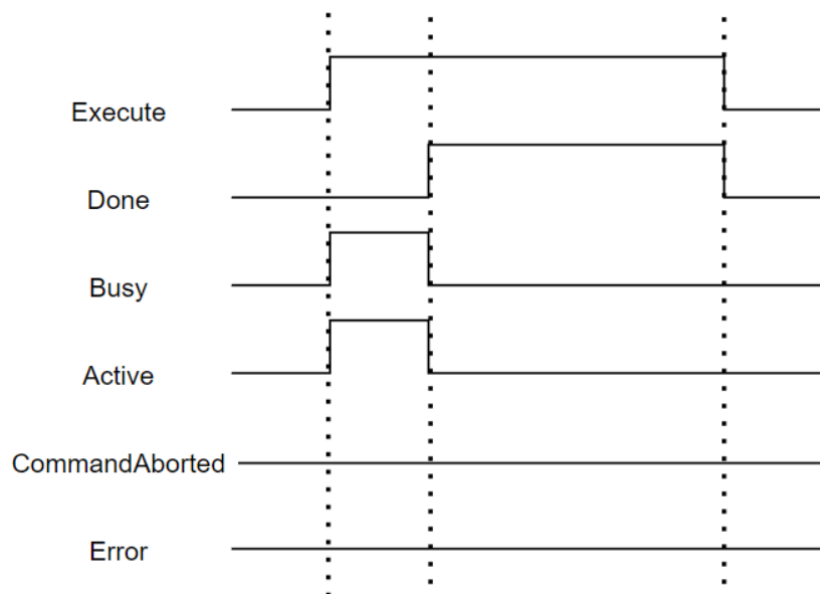
4.13.4.2 Output Parameter

Parameter	Type	Initial value	Description
Done	BOOL	-	When the value is TRUE, it indicates that the function block execution is completed, and no errors were detected.
Busy	BOOL	FALSE	When the value is TRUE, it indicates that this function

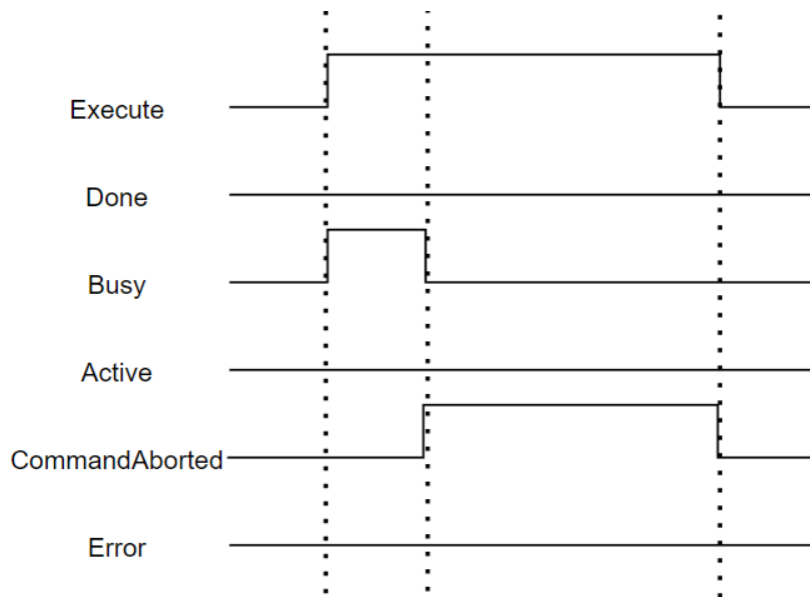
Parameter	Type	Initial value	Description
			block is being executed.
CommandAborted	BOOL	FALSE	When the value is TRUE, it indicates that another motion command has been aborted or an error has been detected, terminating the execution of the function block.
Error	BOOL	FALSE	When the value is TRUE, it indicates that an error has been detected, and the function block execution is terminated.
ErrorID	WORD	-	For motion control error codes, please refer to Appendix C - Motion Control Error Codes for details.

4.13.4.3 Sequency Diagram

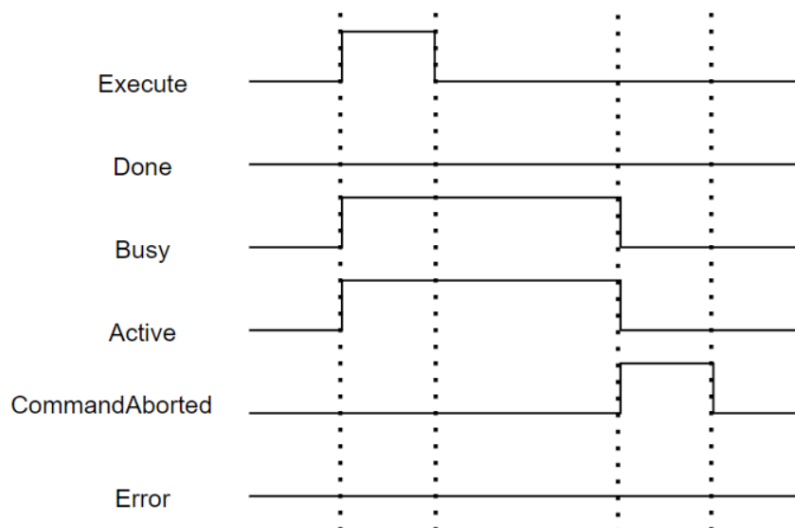
- ◆ Scenario 1: Currently, there are no other commands being executed. Execute the MC_Halt command, and it completes successfully.



- ◆ Scenario 2: Currently, there are other commands in progress. Execute the MC_Halt command using the mcBuffered mode, but before execution, it is preempted by another command using mcAborting. The MC_Halt output pin CommandAborted is set to TRUE.

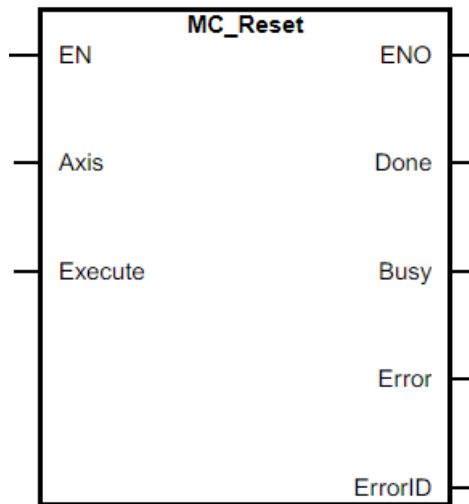


- ◆ Scenario 3: During the execution of the motion command MC_Halt, with Execute set to FALSE, the axis continues moving but is preempted by another command using mcAborting. The MC_Halt output pin CommandAborted is set to TRUE for a scan cycle.



4.13.5 MC_Reset

This function block resets all detected errors related to the axis. If the reset is successful, the Done signal is set to ON, and the state transitions from ErrorStop to Standstill. If the reset fails, the Error signal is set to ON. IT takes effective at rising edge of Execute.



4.13.5.1 Input Parameter

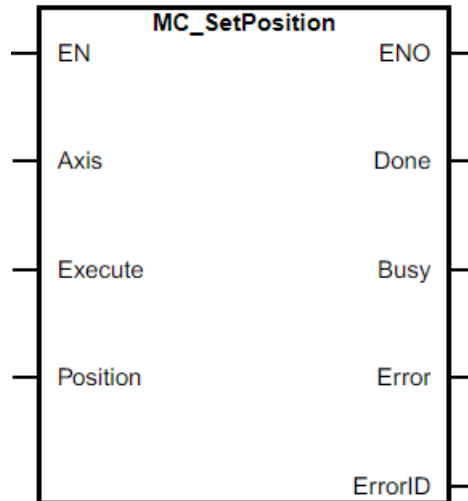
Parameter	Type	Initial value	Description
Axis	AXIS_REF	-	The name of the axis (instance) for executing the function block, customizable when adding the commands.
Execute	BOOL	FALSE	At the rising edge, initiate the execution of the function block. At the falling edge, reset the outputs of the function block when the execution is terminated.

4.13.5.2 Output Parameter

Parameter	Type	Initial value	Description
Done	BOOL	-	When the value is TRUE, it indicates that the function block execution is completed, and no errors were detected.
Busy	BOOL	FALSE	When the value is TRUE, it indicates that this function block is being executed.
Error	BOOL	FALSE	When the value is TRUE, it indicates that an error has been detected, and the function block execution is terminated.
ErrorID	WORD	-	For motion control error codes, please refer to Appendix C - Motion Control Error Codes for details.

4.13.6 MC_SetPosition

This function block is used to change the actual position of the axis without causing any motion. It cannot be used during motion. IT takes effective when Execute is at rising edge.



4.13.6.1 Input Parameter

Parameter	Type	Initial value	Description
Axis	AXIS_REF	-	The name of the axis (instance) for executing the function block, customizable when adding the commands.
Execute	BOOL	FALSE	At the rising edge, initiate the execution of the function block. At the falling edge, reset the outputs of the function block when the execution is terminated.
Position	REAL	-	The target position is expressed in u, where u is the user-defined project unit.

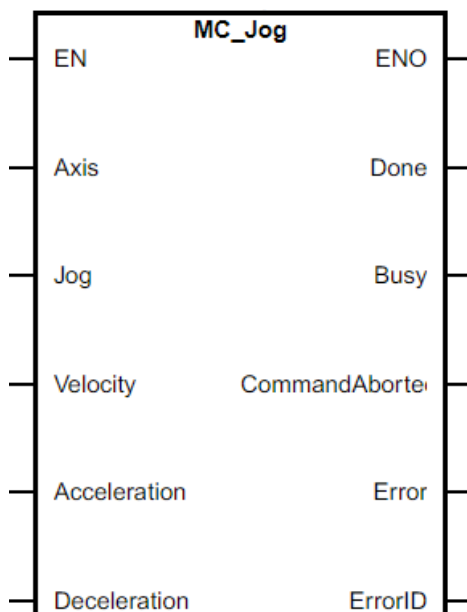
4.13.6.2 Output Parameter

Parameter	Type	Initial value	Description
Done	BOOL	-	When the value is TRUE, it indicates that the function block execution is completed, and no errors were detected.
Busy	BOOL	FALSE	When the value is TRUE, it indicates that this function block is being executed.
Error	BOOL	FALSE	When the value is TRUE, it indicates that an error has been

Parameter	Type	Initial value	Description
			detected, and the function block execution is terminated.
ErrorID	WORD	-	For motion control error codes, please refer to Appendix C - Motion Control Error Codes for details.

4.13.7 MC_Jog

This function block moves the axis at a specified speed and switches the axis to Continuous Motion state.



4.13.7.1 Input Parameter

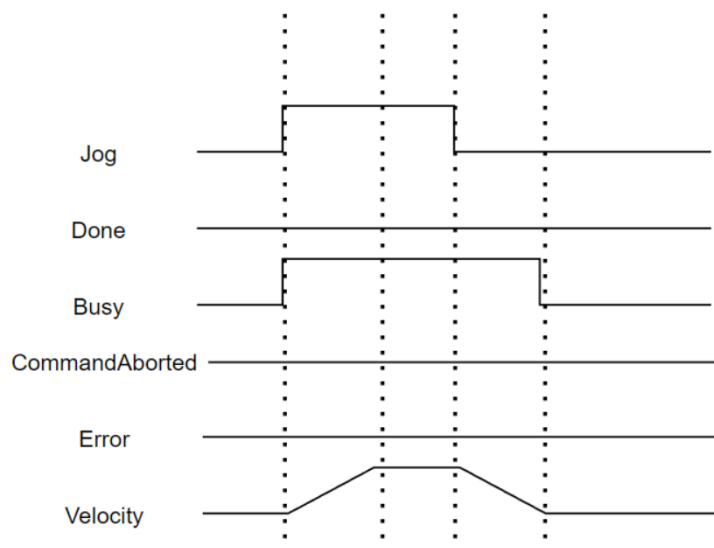
Parameter	Type	Initial value	Description
Axis	AXIS_REF	-	The name of the axis (instance) for executing the function block, customizable when adding the commands.
Jog	BOOL	FALSE	Start the jog motion of the axis when Jog is TRUE.
Velocity	REAL	-	The target speed is expressed in u/s, where u is the user-defined project unit.
Acceleration	REAL	0.0	The acceleration is expressed in u/s ² , where u is the user-defined project unit.
Deceleration	REAL	0.0	The deceleration is expressed in u/s ² , where u is the user-defined project unit.

4.13.7.2 Output Parameter

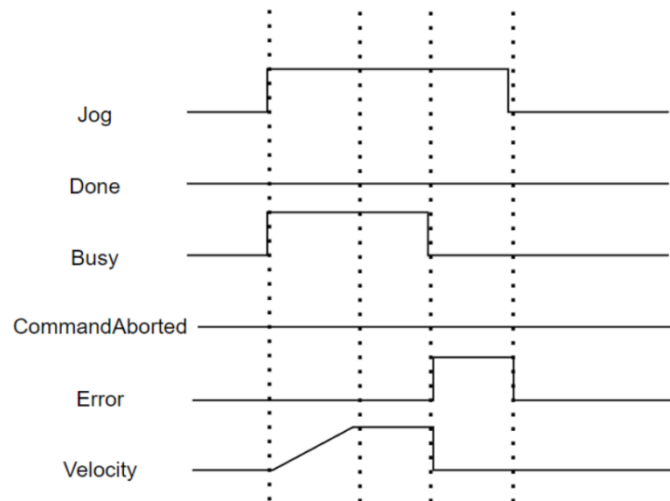
Parameter	Type	Initial value	Description
Done	BOOL	FALSE	When the value is TRUE, it indicates that the function block execution is completed, and no errors were detected.
Busy	BOOL	FALSE	When the value is TRUE, it indicates that this function block is being executed.
CommandAborted	BOOL	FALSE	When the value is TRUE, it indicates that another motion command has been aborted or an error has been detected, terminating the execution of the function block.
Error	BOOL	FALSE	When the value is TRUE, it indicates that an error has been detected, and the function block execution is terminated.
ErrorID	WORD	-	For motion control error codes, please refer to Appendix C - Motion Control Error Codes for details.

4.13.7.3 Sequency Diagram

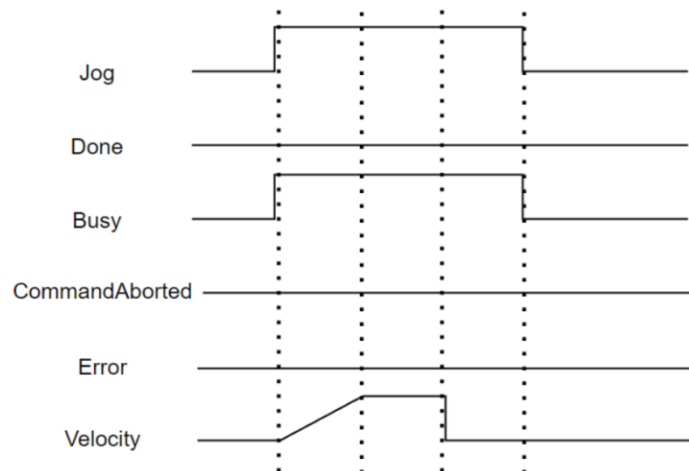
- ◆ Scenario 1: Currently, there are no other commands being executed. Execute the MC_Jog motion command, accelerate to the preset Velocity, and continue moving. When Jog is set to FALSE, it begins decelerating to stop at 0.



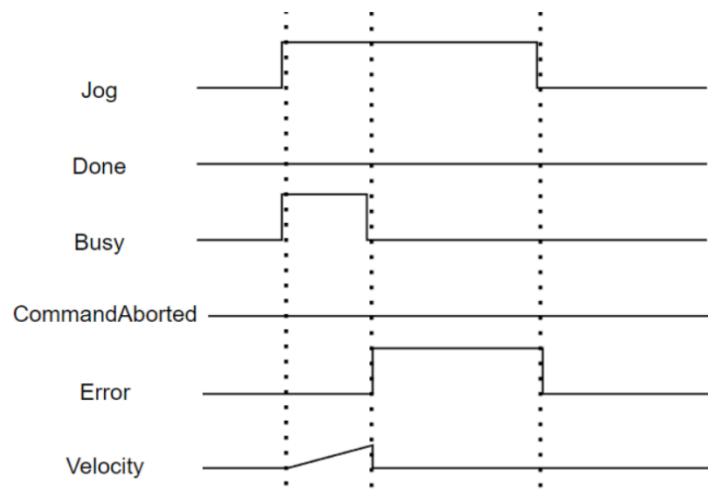
- ◆ Scenario 2: Currently, there are no other commands being executed. Execute the MC_Jog motion command, accelerate to the preset Velocity, and continue moving. When reaching the positive or negative hardware limits or the positive or negative software limits, the axis state is updated to ErrorStop, and the current command's output pin Error is set to TRUE.



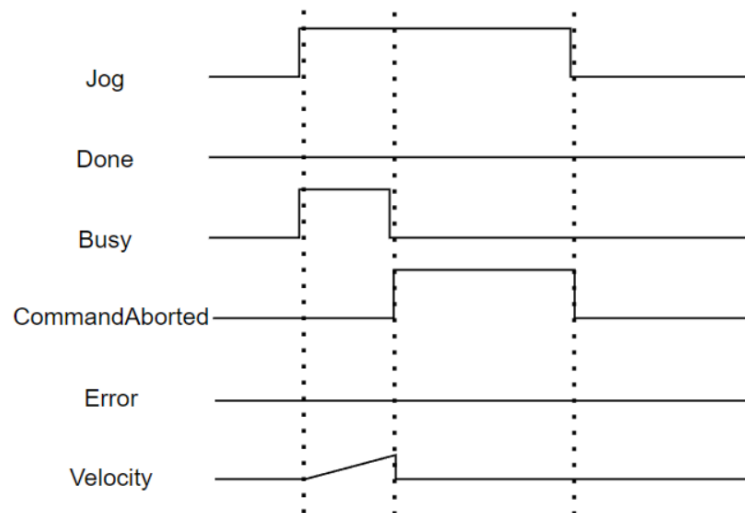
- ◆ Scenario 3: Currently, there are no other commands being executed. Execute the MC_Jog motion command, accelerate to the preset Velocity. If preempted by another motion command using mcAborting, CommandAborted remains FALSE.



- ◆ Scenario 4: Currently, there are no other commands being executed. Execute the MC_Jog motion command, but acceleration does not reach the preset Velocity. When reaching the positive or negative hardware limits or the positive or negative software limits, the axis state is updated to ErrorStop, and the MC_Jog output pin Error is set to TRUE.



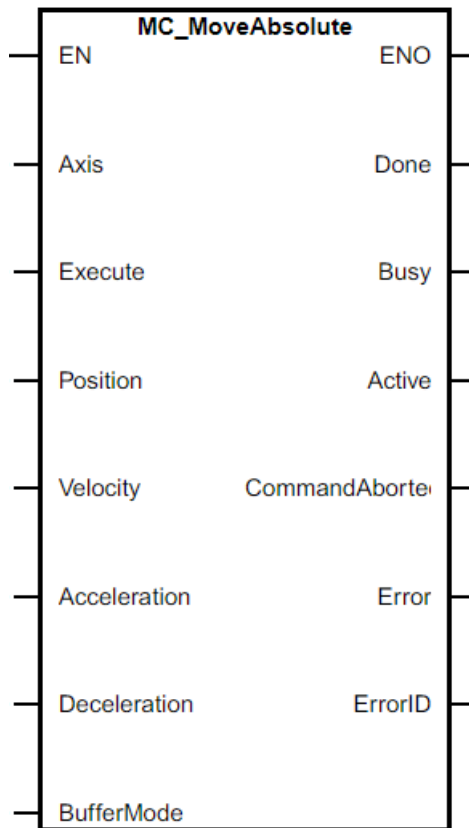
- ◆ Scenario 5: Currently, there are no other commands being executed. Execute the MC_Jog motion command, but acceleration does not reach the preset Velocity. If preempted by another motion command using mcAborting, the MC_Jog output pin CommandAborted is set to TRUE.



4.13.8 MC_MoveAbsolute

This function block moves the axis towards the target position at a specified speed and transitions the axis's state to Discrete. If the axis is not homed (i.e., the absolute reference position is not defined), the function block will terminate execution and set Error to ON.

The 'Position' parameter is used to set the target position for positioning. If the current position is less than the target position, the axis will move in the positive direction until it reaches the position set by 'Position.' If the current position is greater than the target position, the axis will move in the negative direction until it reaches the position set by 'Position'.



4.13.8.1 Input Parameter

Parameter	Type	Initial value	Description
Axis	AXIS_REF	-	The name of the axis (instance) for executing the functionality block, customizable when adding the commands.
Execute	BOOL	FALSE	At the rising edge, start the execution of the function block. At the falling edge, reset the output of the function block at the end of its execution.
Position	REAL	-	Absolute position expressed in u , where u is the user-defined project unit.
Velocity	REAL	-	Target velocity expressed in u/s , where u is the user-defined project unit.
Acceleration	REAL	0.0	Acceleration expressed in u/s ² , where u is the user-defined project unit.
Deceleration	REAL	0.0	Deceleration expressed in u/s ² , where u is the user-defined project unit.

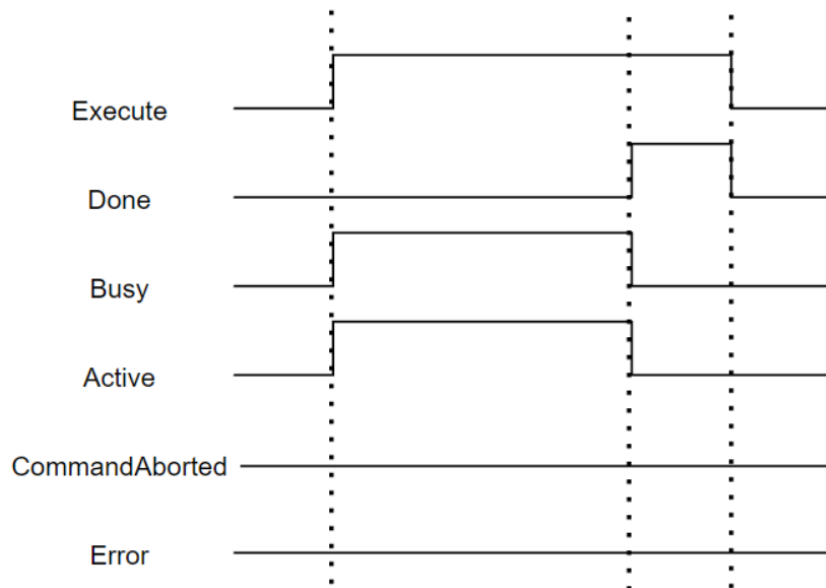
Parameter	Type	Initial value	Description
BufferMode	MC_BUFFER_MODE	mcAborting	Buffering mode, for more information, please refer to Buffering Mode .

4.13.8.2 Output Parameter

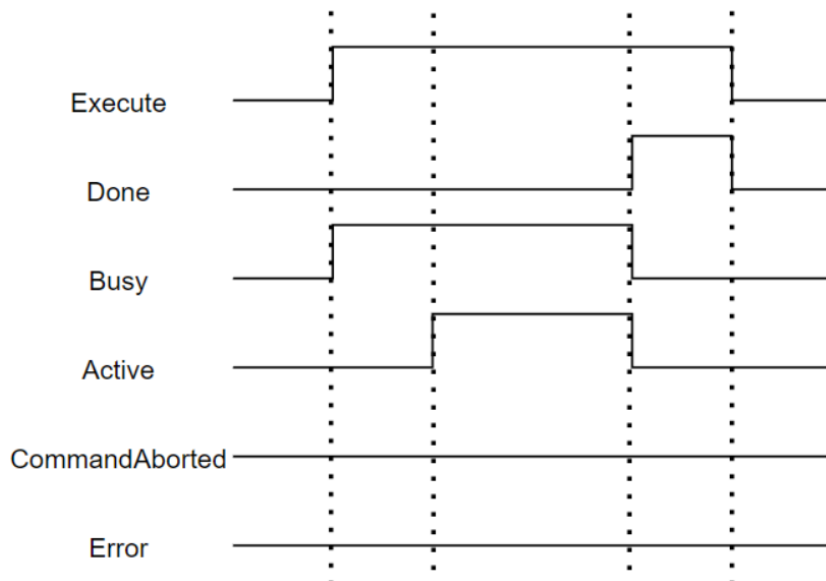
Parameter	Type	Initial value	Description
Done	BOOL	FALSE	When the value is TRUE, it indicates that the function block execution is completed, and no errors were detected.
Busy	BOOL	FALSE	When the value is TRUE, it indicates that this function block is being executed.
Active	BOOL	FALSE	When the value is TRUE, it signifies that the current function block controls the Axis. For a given Axis, only one function block can set Active to TRUE at a time.
CommandAborted	BOOL	FALSE	When the value is TRUE, it indicates that another motion command has been aborted or an error has been detected, terminating the execution of the function block.
Error	BOOL	FALSE	When the value is TRUE, it indicates that an error has been detected, and the function block execution is terminated.
ErrorID	WORD	-	For motion control error codes, please refer to Appendix C - Motion Control Error Codes for details.

4.13.8.3 Sequency Diagram

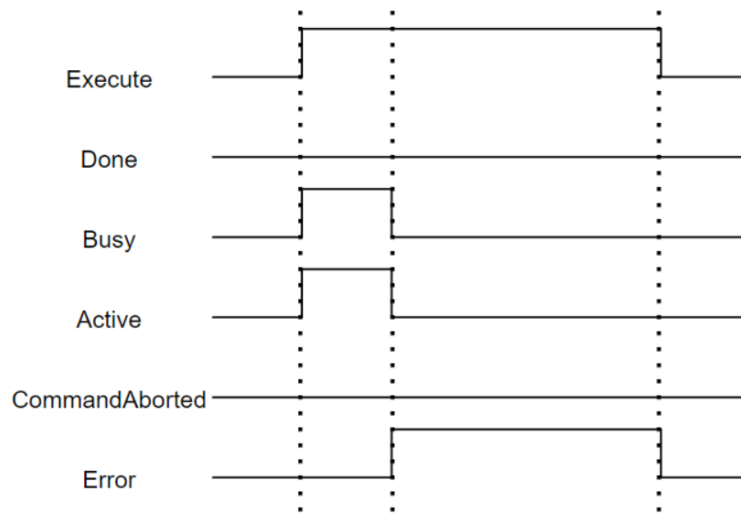
- ◆ Scenario 1: Currently, there are no other commands being executed. Execute the motion command MC_MoveAbsolute, and it completes successfully.



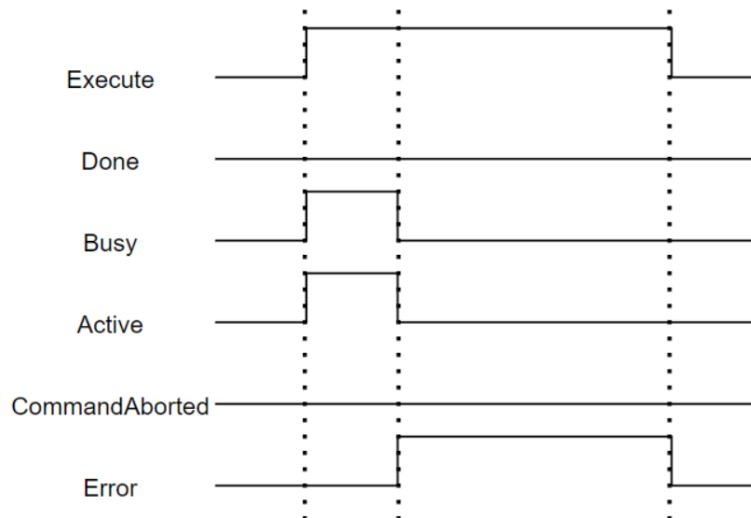
- ◆ Scenario 2: Currently, there are other commands in progress. Execute the motion command MC_MoveAbsolute using the mcBuffered mode. It is necessary to wait for the completion of the previous command before obtaining control of the axis and allowing the MC_MoveAbsolute command to execute until it completes successfully.



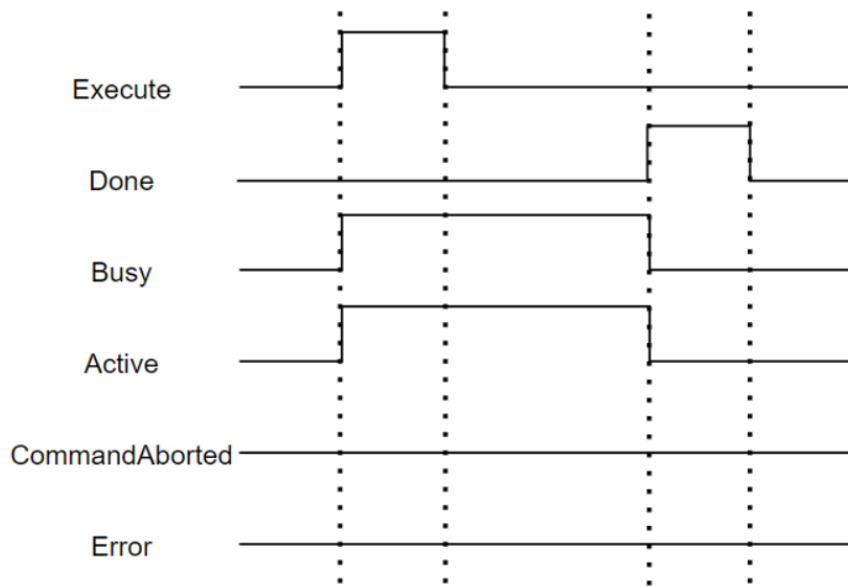
- ◆ Scenario 3: During the execution of the motion command MC_MoveAbsolute, it is preempted by another command using mcAborting mode. As a result, the current command loses control of the axis, and the current command's output pin CommandAborted is set to TRUE.



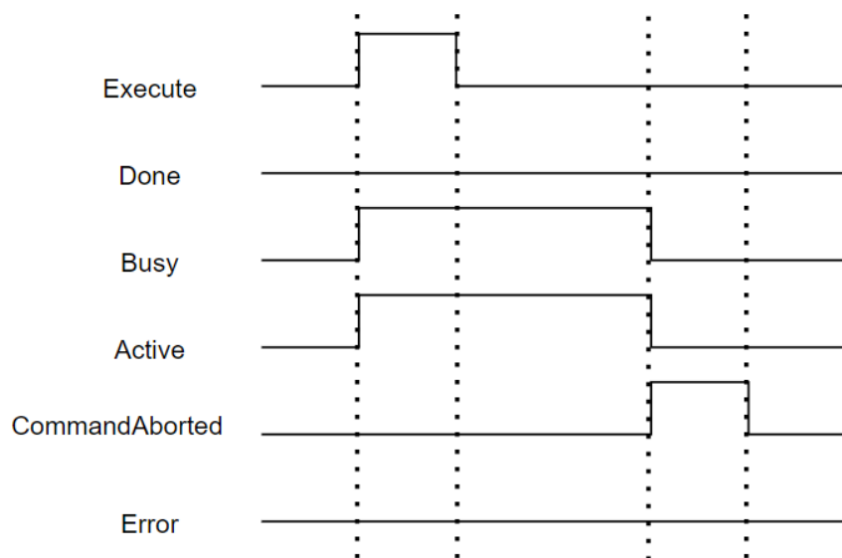
- ◆ Scenario 4: During the execution of the motion command MC_MoveAbsolute, reaching the positive or negative hardware limits or the positive or negative software limits results in the axis state being updated to ErrorStop. The current command's output pin Error is set to TRUE.



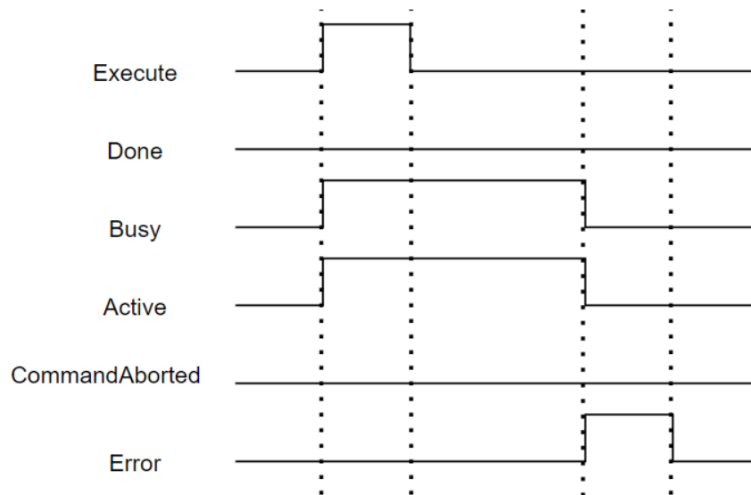
- ◆ Scenario 5: During the execution of the motion command MC_MoveAbsolute, setting Execute to FALSE allows the axis to continue moving and complete the current command. The output pin Done is set to TRUE (for a duration of one scan cycle).



- ◆ Scenario 6: During the execution of the motion command MC_MoveAbsolute, setting Execute to FALSE allows the axis to continue moving. If preempted by another command using mcAborting, the current command's output pin CommandAborted is set to TRUE (for a duration of one scan cycle).



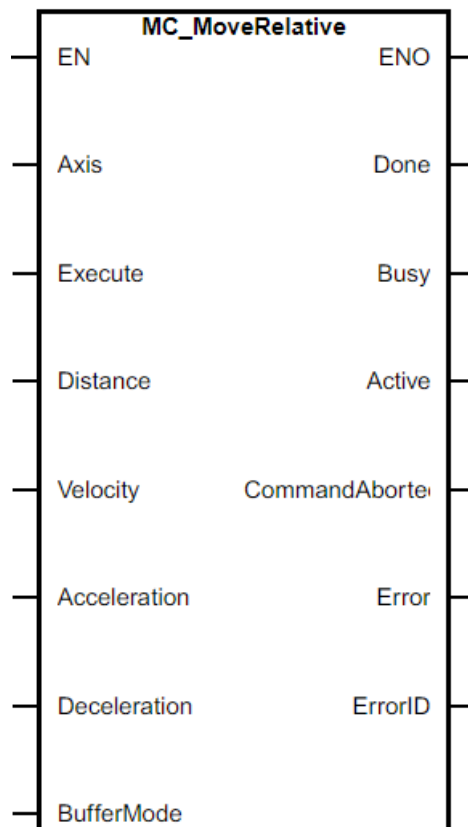
- ◆ Scenario 7: During the execution of the motion command MC_MoveAbsolute, setting Execute to FALSE allows the axis to continue moving until reaching the positive or negative hardware limits or the positive or negative software limits. The axis state is updated to ErrorStop, and the current command's output pin Error is set to TRUE (for a duration of one scan cycle).



4.13.9 MC_MoveRelative

This function block moves the axis a relative distance and transitions the axis's state to Discrete. When executing this command, the target position is set by taking the current position as a reference and adding a relative distance. It takes effect at rising edge of Execute.

Distance is used to set the distance for relative positioning. In both linear and circular motion modes, if Distance is a positive value, the axis moves in the positive direction for the distance specified by Distance. If Distance is negative, the axis moves in the negative direction for the distance specified by |Distance|.



4.13.9.1 Input Parameter

Parameter	Type	Initial value	Description
Axis	AXIS_REF	-	The name of the axis (instance) for executing the functionality block, customizable when adding the commands.
Execute	BOOL	FALSE	At the rising edge, start the execution of the function block. At the falling edge, reset the output of the function block at the end of its execution.
Distance	REAL	-	The relative motion distance is expressed in units (u), where u is the user-defined project unit.
Velocity	REAL	-	The target velocity is expressed in units (u/s), where u is the user-defined project unit.
Acceleration	REAL	0.0	The acceleration is expressed in units (u/s ²), where u is the user-defined project unit.
Deceleration	REAL	0.0	The deceleration is expressed in units (u/s ²), where u is the user-defined project unit.
BufferMode	MC_BUFFER_MODE	mcAborting	Buffering mode, for details please refer to Buffering Mode .

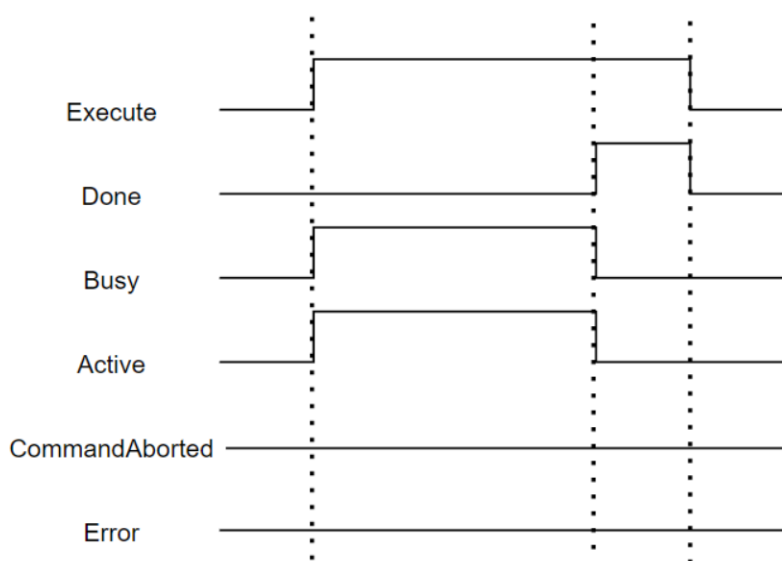
4.13.9.2 Output Parameter

Parameter	Type	Initial value	Description
Done	BOOL	FALSE	When the value is TRUE, it indicates that the function block execution is completed, and no errors were detected.
Busy	BOOL	FALSE	When the value is TRUE, it indicates that this function block is being executed.
Active	BOOL	FALSE	When the value is TRUE, it signifies that the current function block controls the Axis. For a given Axis, only one function block can set Active to TRUE at a time.
CommandAborted	BOOL	FALSE	When the value is TRUE, it indicates that another motion command has been aborted or an error has been detected, terminating the execution of the function block.

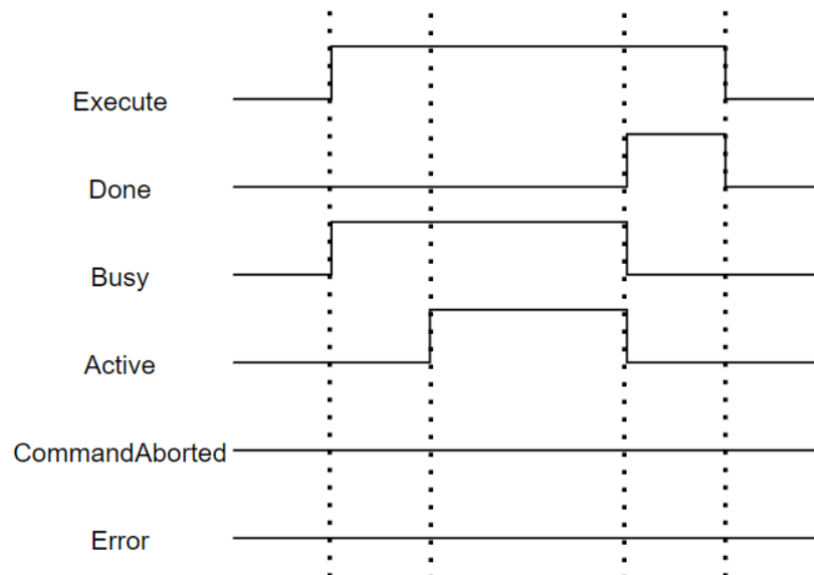
Parameter	Type	Initial value	Description
Error	BOOL	FALSE	When the value is TRUE, it indicates that an error has been detected, and the function block execution is terminated.
ErrorID	WORD	-	Error code, for detailed information, please refer to Appendix A - Communication Code Description .

4.13.9.3 Sequency Diagram

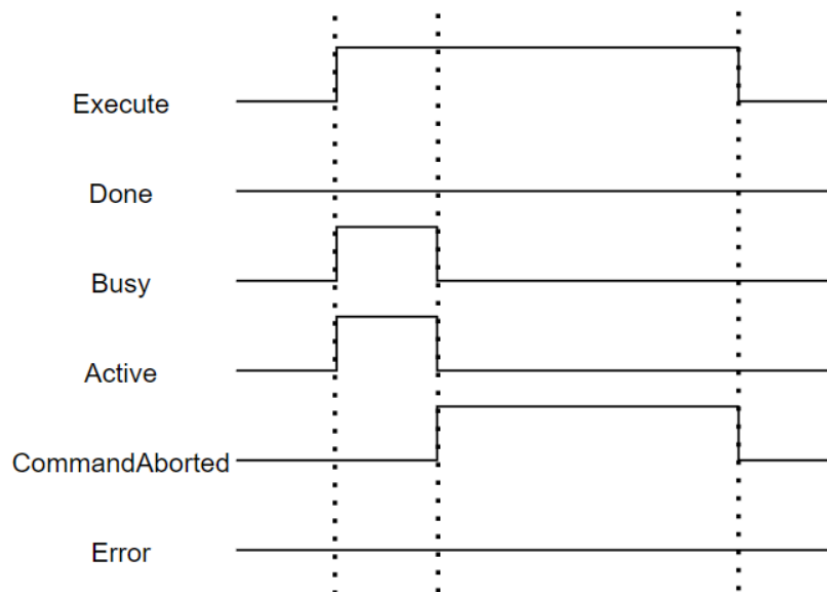
- ◆ Scenario 1: Currently, there are no other commands being executed. Execute the motion command MC_MoveRelative, and it completes successfully.



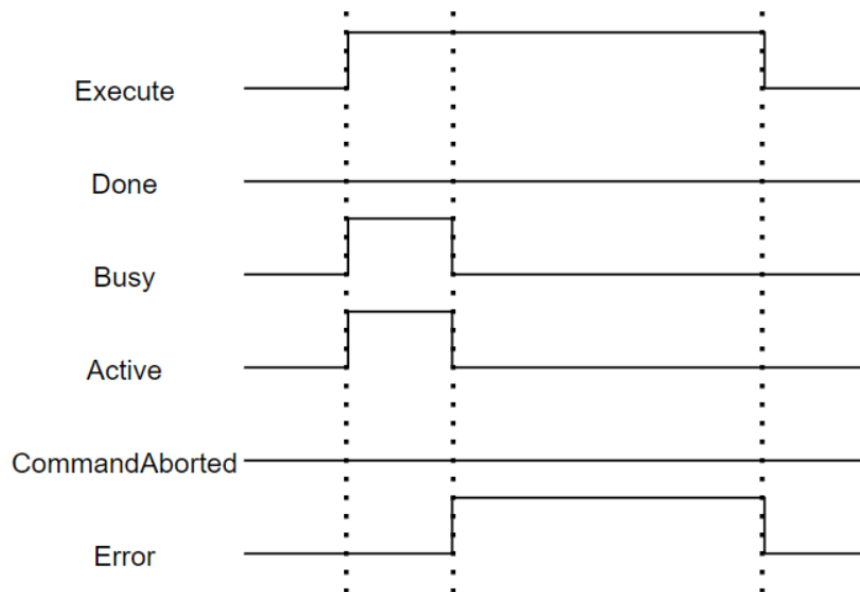
- ◆ Scenario 2: Currently, there are other commands in progress. Execute the motion command MC_MoveRelative using the mcBuffered mode. It is necessary to wait for the completion of the previous command before obtaining control of the axis and allowing the MC_MoveRelative command to execute until it completes successfully.



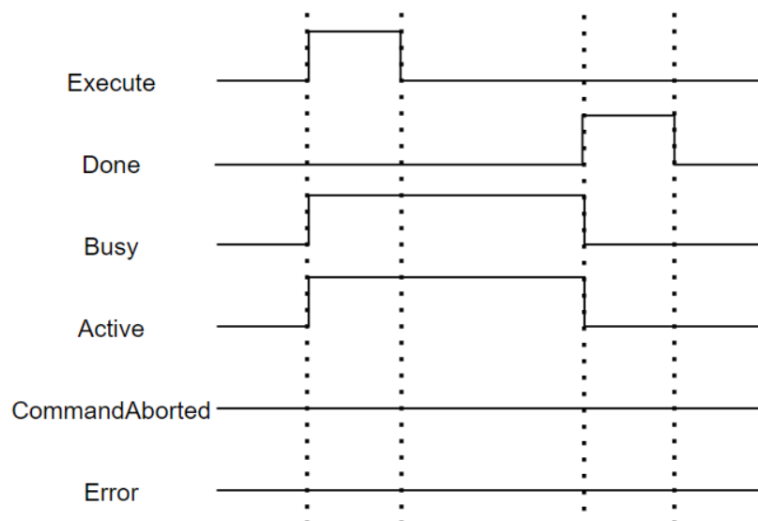
- ◆ Scenario 3: During the execution of the MC_MoveRelative command, it is preempted by other commands in the form of mcAborting, so that the current command loses axis control, and the output pin of MC_MoveRelative CommandAborted is set to True.



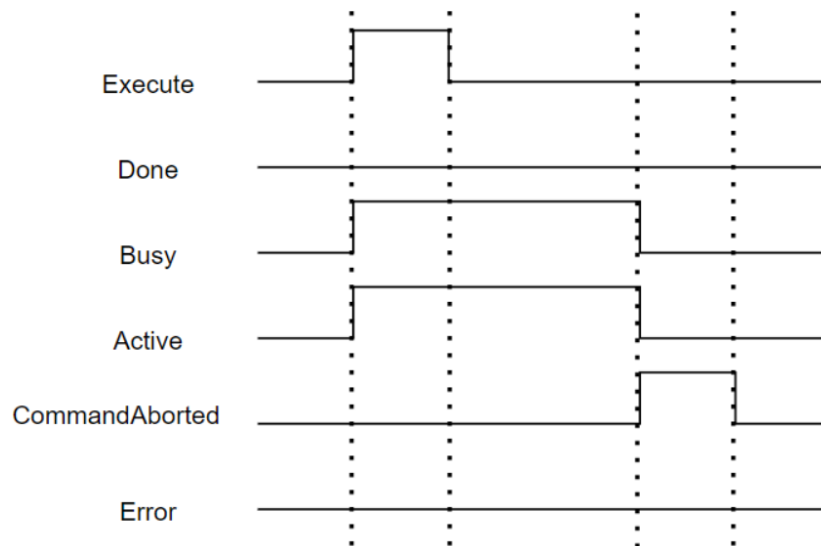
- ◆ Scenario 4: During the execution of the motion command MC_MoveRelative, reaching the positive or negative hardware limits or the positive or negative software limits, the axis status is updated to ErrorStop, and the output pin Error of the MC_MoveRelative is set to True.



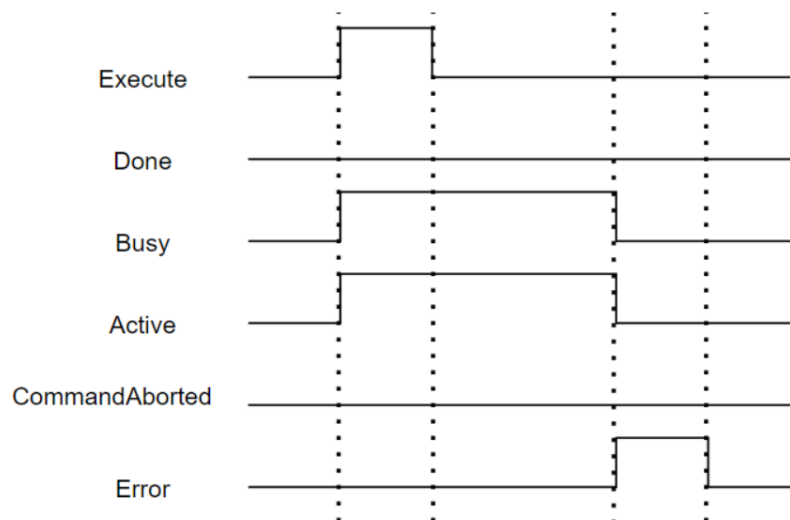
- ◆ Scenario 5: During the execution of the MC_MoveRelative command, when Execute is set to False during the motion, the axis continues to move to complete the current command, and the output pin Done is set to True for a scan cycle.



- ◆ Scenario 6: During the execution of the MC_MoveRelative command, when Execute is set to False during the movement, the axis is preempted by other commands in mcAborting mode when it continues to move, and the current command output pin CommandAborted is set to True for a scan cycle.

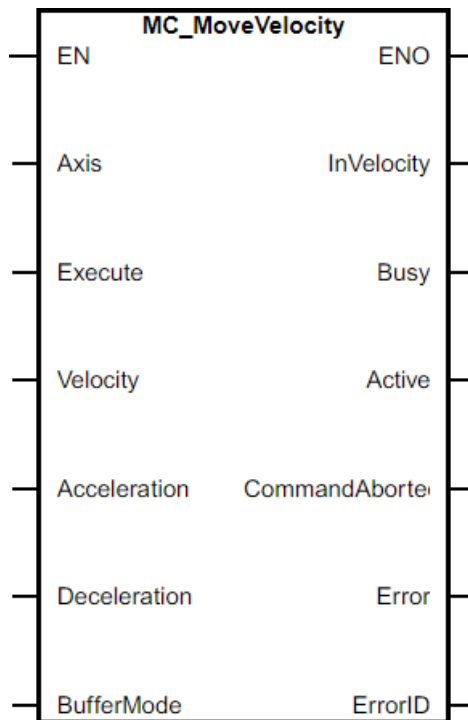


- ◆ Scenario 7: During the execution of the MC_MoveRelative motion command, when Execute is set to False during the movement, the axis continues to move and reaches the positive and negative hardware or the positive and negative soft limit, the axis status is updated to ErrorStop, and the current command output pin Error is set to True for a scan cycle.



4.13.10 MC_MoveVelocity

This function block moves the axis at the specified speed and transitions the state of the axis to Continuous. This continuous motion will be maintained until the software limit is reached, the movement is triggered to abort, or a transition to an ErrorStop state is detected. It takes effect at the rising edge of Execute.



4.13.10.1 Input Parameter

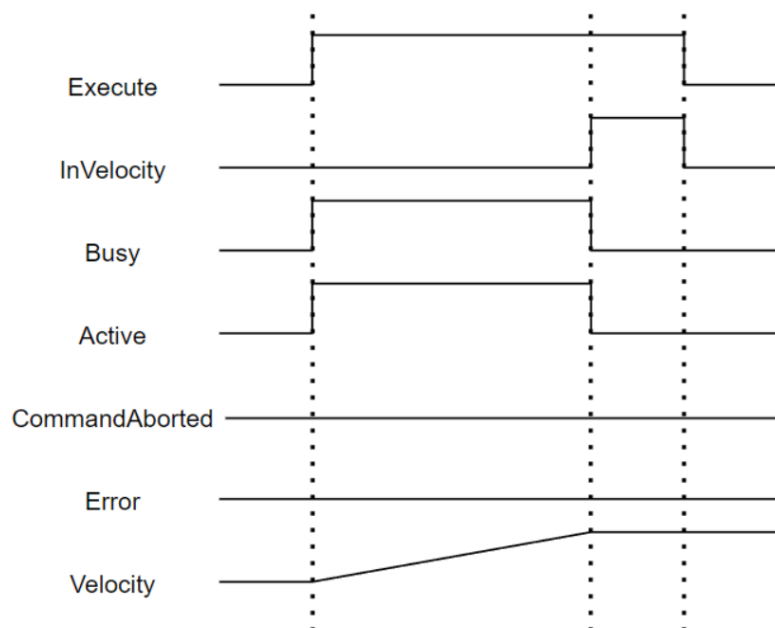
Parameter	Type	Initial value	Description
Axis	AXIS_REF	-	The name of the axis (instance) for executing the functionality block, customizable when adding the commands.
Execute	BOOL	FALSE	At the rising edge, start the execution of the function block. At the falling edge, reset the output of the function block at the end of its execution.
Velocity	REAL	-	Target velocity expressed in u/s, where u is the user-defined project unit.
Acceleration	REAL	0.0	Acceleration expressed in u/s ² , where u is the user-defined project unit.
Deceleration	REAL	0.0	Deceleration expressed in u/s ² , where u is the user-defined project unit.
BufferMode	MC_BUFFER_MODE	mcAborting	Buffering mode, for details please refer to Buffering Mode .

4.13.10.2 Output Parameter

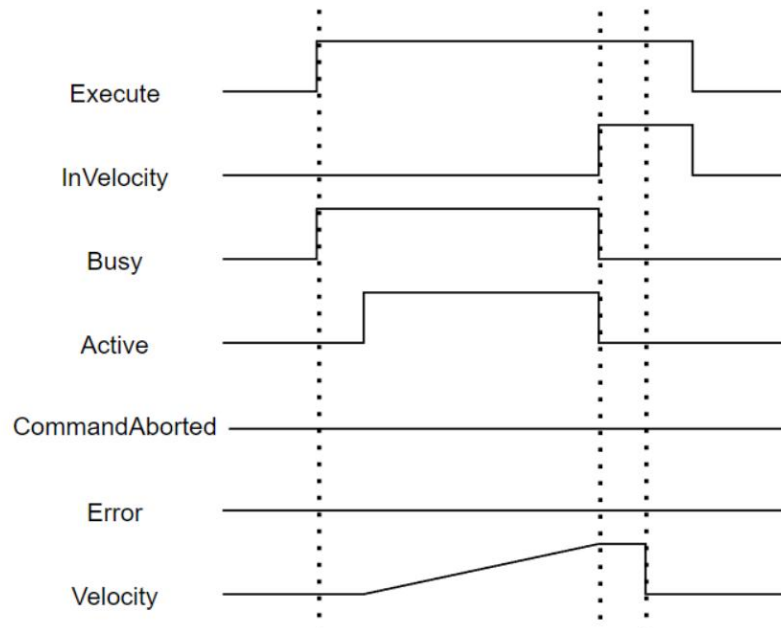
Parameter	Type	Initial value	Description
InVelocity	BOOL	FALSE	When the value is TRUE, it indicates that the target velocity has been reached.
Busy	BOOL	FALSE	When the value is TRUE, it indicates that this function block is currently being executed.
Active	BOOL	FALSE	When the value is TRUE, it signifies that the current function block controls the Axis. For a given Axis, only one function block can set Active to TRUE at a time.
CommandAborted	BOOL	FALSE	When the value is TRUE, it indicates that another motion command has been aborted or an error has been detected, terminating the execution of the function block.
Error	BOOL	FALSE	When the value is TRUE, it indicates that an error has been detected, and the function block execution is terminated.
ErrorID	WORD	-	Error code, for detailed information, please refer to Appendix A - Communication Code Description .

4.13.10.3 Sequence Diagram

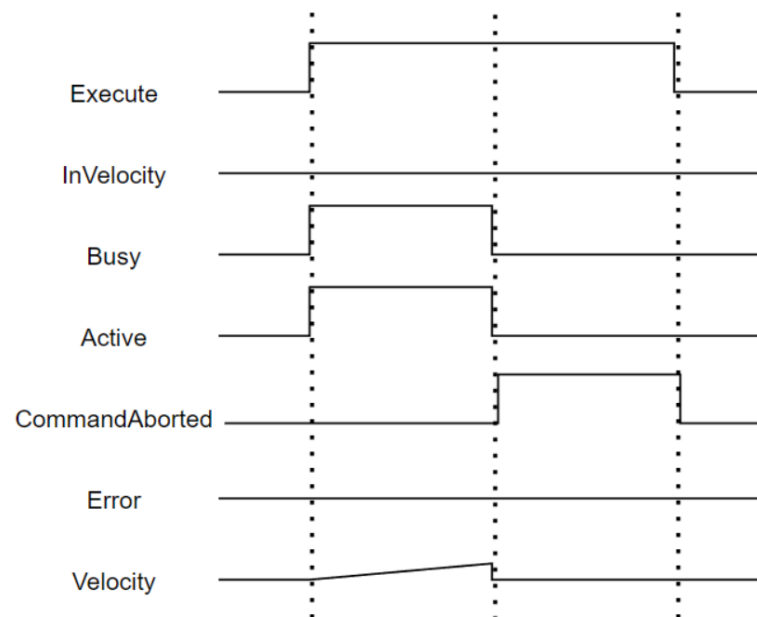
- ◆ Scenario 1: Currently, no other commands are being executed. Execute the motion command MC_MoveVelocity, accelerating to the preset Velocity. InVelocity is set to TRUE.



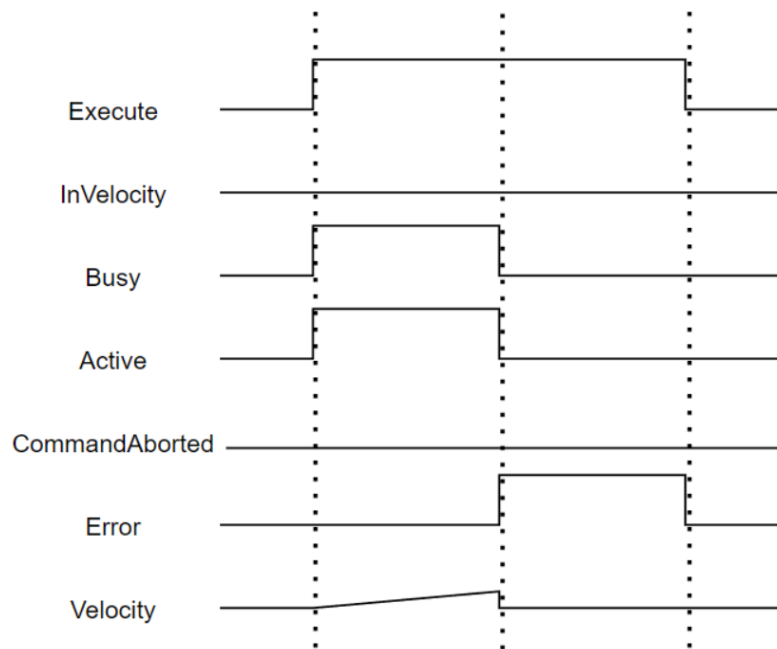
- ◆ Scenario 2: Currently, other commands are being executed. Execute the motion command MC_MoveVelocity in mcBuffered mode, accelerating to the preset Velocity. After reaching the positive or negative limits or being preempted by another command using mcAborting, Error/CommandAborted remains FALSE.



- ◆ Scenario 3: Currently, no other commands are being executed. Execute the motion command MC_MoveVelocity, accelerating but not reaching the preset Velocity. Being preempted by another command using mcAborting, CommandAborted is set to TRUE.

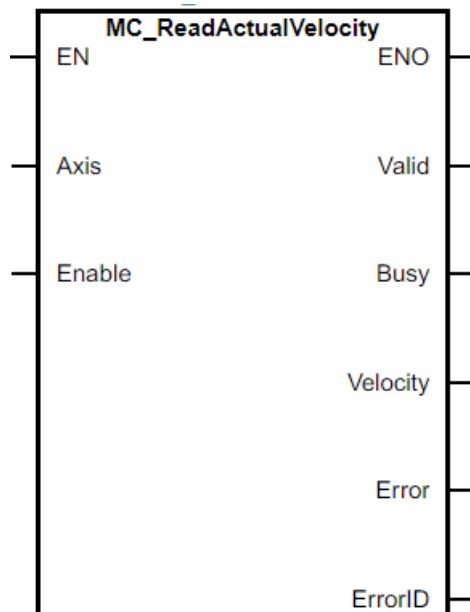


- ◆ Scenario 4: Currently, no other commands are being executed. Execute the motion command MC_MoveVelocity, accelerating but not reaching the preset Velocity. Upon reaching the positive or negative hardware limits or software limits, the axis state is updated to ErrorStop, and MC_MoveVelocity outputs the Error pin, which is set to TRUE.



4.13.11 MC_ReadActualVelocity

This function block reads the actual velocity of the axis.



4.13.11.1 Input Parameter

Parameter	Type	Initial value	Description
Axis	AXIS_REF	-	The name of the axis (instance) for executing the function block, customizable when adding the commands.

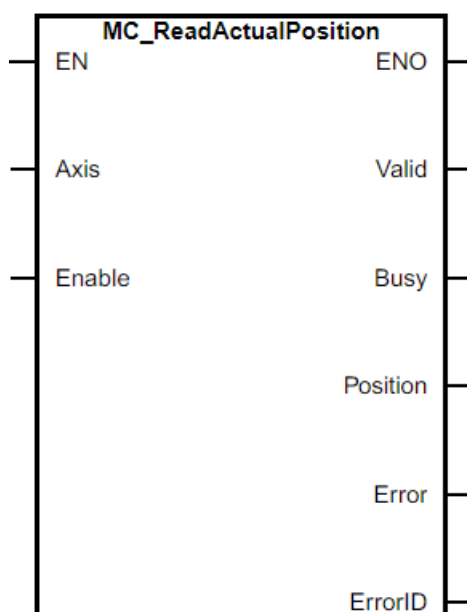
Parameter	Type	Initial value	Description
Enable	BOOL	FALSE	When the value is TRUE, the function block is executed.

4.13.11.2 Output Parameter

Parameter	Type	Initial value	Description
Valid	BOOL	FALSE	When the value is TRUE, it indicates that the output result of this function block is valid.
Busy	BOOL	FALSE	When the value is TRUE, it signifies that this function block is currently being executed.
Velocity	REAL	-	The actual velocity is expressed in u/s, with u being the user-defined project unit.
Error	BOOL	FALSE	When the value is TRUE, it indicates that an error has been detected, and the function block execution is terminated.
ErrorID	WORD	-	motion control error codes, please refer to Appendix C - Motion Control Error Codes for details.

4.13.12 MC_ReadActualPosition

This function block reads the actual position of the axis.



4.13.12.1 Input Parameter

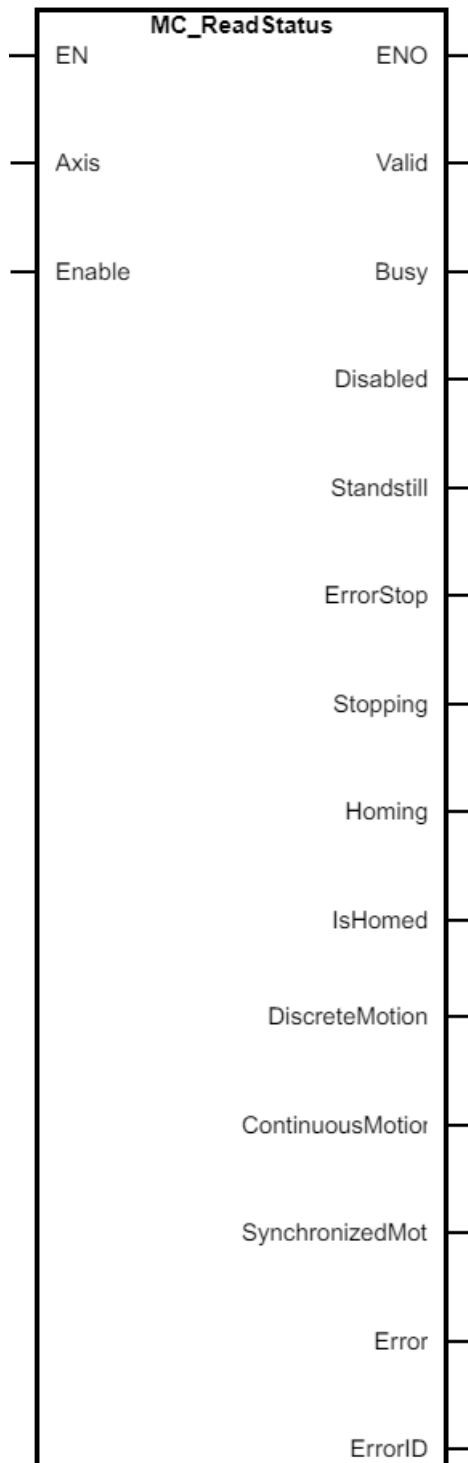
Parameter	Type	Initial value	Description
Axis	AXIS_REF	-	The name of the axis (instance) for executing the function block, customizable when adding the command.
Enable	BOOL	FALSE	When the value is TRUE, the function block is executed.

4.13.12.2 Output Parameter

Parameter	Type	Initial value	Description
Valid	BOOL	FALSE	When the value is TRUE, it indicates that the output result of this function block is valid.
Busy	BOOL	FALSE	When the value is TRUE, it signifies that this function block is currently being executed.
Position	REAL	-	The actual position is expressed in u, with u being the user-defined project unit.
Error	BOOL	FALSE	When the value is TRUE, it indicates that an error has been detected, and the function block execution is terminated.
ErrorID	WORD	-	Error code, for detailed information, please refer to Appendix C - Motion Control Error Codes .

4.13.13 MC_ReadStatus

This function block is used to read the status of the axis.



4.13.13.1 Input Parameter

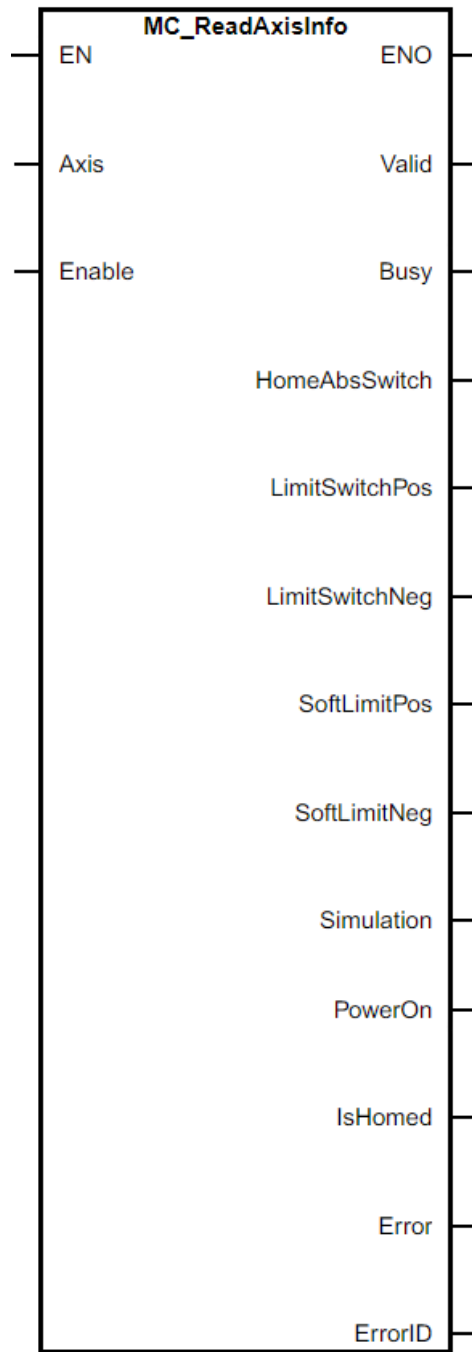
Parameter	Type	Initial value	Description
Axis	AXIS_REF	-	The name of the axis (instance) for executing the function block, customizable when adding the commands.
Enable	BOOL	FALSE	When the value is TRUE, the function block is executed.

4.13.13.2 Output Parameter

Parameter	Type	Initial value	Description
Valid	BOOL	FALSE	When the value is TRUE, it indicates that the output result of this function block is valid.
Busy	BOOL	FALSE	When the value is TRUE, it signifies that this function block is currently being executed.
Disabled	BOOL	FALSE	When the value is TRUE, the axis is in the current state.
Standstill	BOOL	FALSE	When the value is TRUE, the axis is in Standstill state.
ErrorStop	BOOL	FALSE	When the value is TRUE, the axis is in ErrorStop state.
Stopping	BOOL	FALSE	When the value is TRUE, the axis is in Stopping state.
Homing	BOOL	FALSE	When the value is TRUE, the axis is in Homing state.
IsHomed	BOOL	FALSE	When the value is TRUE, the zero point is valid, allowing for absolute motion.
DiscreteMotion	BOOL	FALSE	When the value is TRUE, the axis is in Discrete Motion state.
ContinuousMotion	BOOL	FALSE	When the value is TRUE, the axis is in Continuous Motion state.
SynchronizedMotion	BOOL	FALSE	When the value is TRUE, the axis is in Synchronized Motion state.
Error	BOOL	FALSE	When the value is TRUE, it indicates that an error has been detected, and the function block execution is terminated.
ErrorID	WORD	-	Error code, for detailed information, please refer to Appendix C - Motion Control Error Codes .

4.13.14 MC_ReadAxisInfo

This function block is used to read information about the axis, including whether it has reached positive or negative hardware limits and whether the origin coordinates have been established.



4.13.14.1 Input Parameter

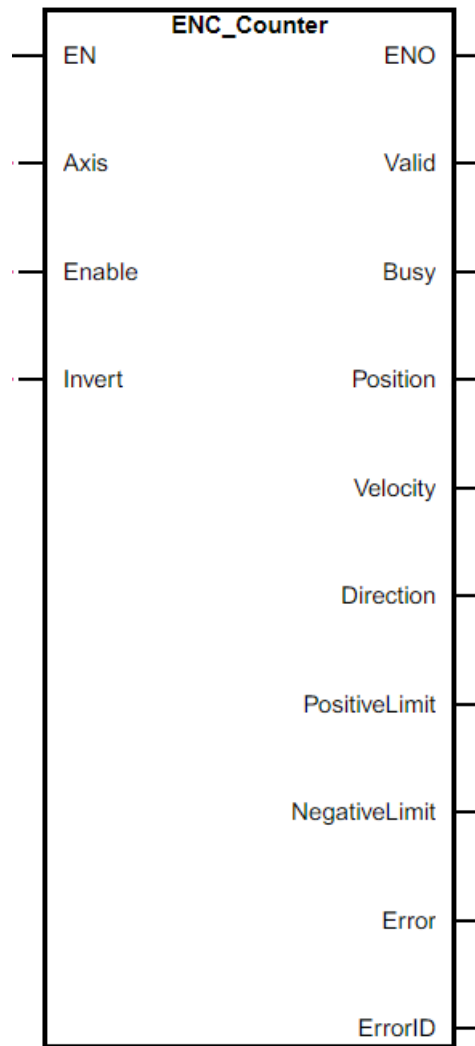
Parameter	Type	Initial value	Description
Axis	AXIS_REF	-	The name of the axis (instance) for executing the function block, customizable when adding the commands.
Enable	BOOL	FALSE	The function block is executed when the value is TRUE.

4.13.14.2 Output Parameter

Parameter	Type	Initial value	Description
Valid	BOOL	FALSE	When the value is TRUE, it indicates that the output result of this function block is valid.
Busy	BOOL	FALSE	When the value is TRUE, it indicates that this function block is currently being executed.
HomeAbsSwitch	BOOL	FALSE	When the value is TRUE, it indicates the triggering of the homing signal input.
LimitSwitchPos	BOOL	FALSE	When the value is TRUE, it indicates reaching the positive hardware limit.
LimitSwitchNeg	BOOL	FALSE	When the value is TRUE, it indicates reaching the negative hardware limit.
SoftLimitPos	BOOL	FALSE	When the value is TRUE, it indicates reaching the positive software limit.
SoftLimitNeg	BOOL	FALSE	When the value is TRUE, it indicates reaching the negative software limit.
Simulation	BOOL	FALSE	When the value is TRUE, it signifies enabling the virtual axis mode.
PowerON	BOOL	FALSE	When the value is TRUE, it signifies that the axis has been enabled.
IsHomed	BOOL	FALSE	When the value is TRUE, it signifies the creation of the origin coordinate.
Error	BOOL	FALSE	When the value is TRUE, it indicates the detection of an error, and the execution of the function block is terminated.
ErrorID	WORD	-	Error code, for detailed information, please refer to Appendix C - Motion Control Error Codes .

4.13.15 ENC_Counter

This function block is used to enable the counter and provide feedback on signals such as current position value, velocity, direction, etc.



4.13.15.1 Input Parameter

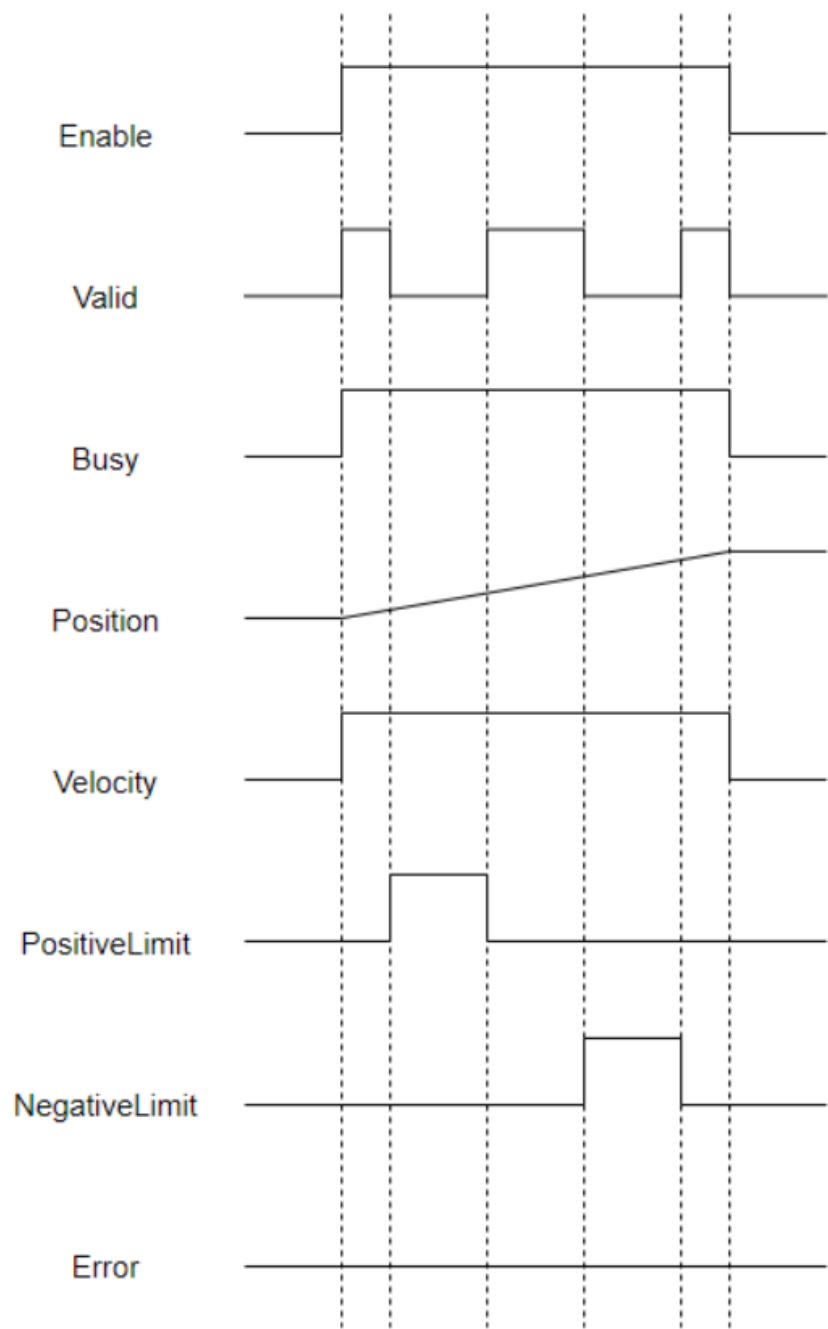
Parameter	Type	Initial value	Description
Axis	AXIS_REF	-	The name of the axis (instance) for executing the function block, customizable when adding the commands.
Enable	BOOL	FALSE	When the value is TRUE, start counting on the specified axis; when the value is FALSE, stop counting on the axis.
Invert	BOOL	FALSE	When the value is FALSE, counting up; when the value is TRUE, counting down.

4.13.15.2 Output Parameter

Parameter	Type	Initial value	Description
Valid	BOOL	FALSE	When the value is TRUE, it indicates a set of valid outputs in

Parameter	Type	Initial value	Description
			this function block.
Busy	BOOL	FALSE	When the value is TRUE, it indicates that this function block is currently being executed.
Position	REAL	-	The current value of the counting axis.
Velocity	REAL	-	The current velocity of the counting axis.
Direction	BOOL	FALSE	The current direction of the counting axis. TRUE indicates positive, FALSE indicates negative.
PositiveLimit	BOOL	FALSE	Whether the position of the counting axis has reached the positive software limit value. TRUE indicates reached, FALSE indicates not reached.
NegativeLimit	BOOL	FALSE	Whether the position of the counting axis has reached the negative software limit value. TRUE indicates reached, FALSE indicates not reached.
Error	BOOL	FALSE	When the value is TRUE, it indicates that an error is detected, and the execution of the function block ends.
ErrorID	WORD	-	Error code, for detailed information, please refer to Appendix C - Motion Control Error Codes .

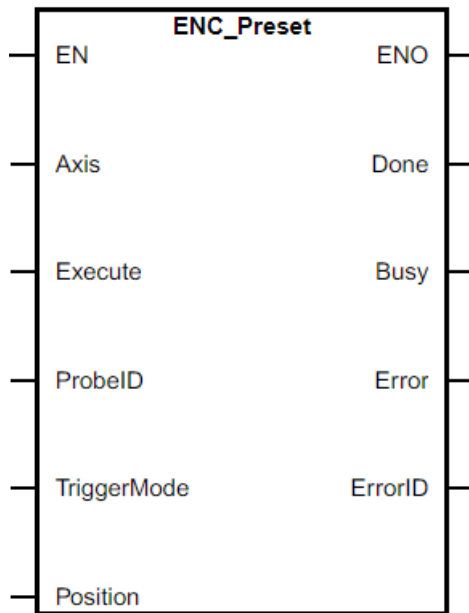
4.13.15.3 Sequency Diagram



When the Invert pin is TRUE, the values of Position/Velocity are negated, and Direction/PositiveLimit/NegativeLimit are inverted.

4.13.16 ENC_Preset

This function block changes the position of the counting axis to the user's specified value.



4.13.16.1 Input Parameter

Parameter	Type	Initial value	Description
Axis	AXIS_REF	-	Execute the function block on the specified axis (instance) with a customizable name.
Execute	BOOL	FALSE	When the value changes from FALSE to TRUE, trigger at the rising edge to initiate the execution.
Probe ID	INT	0	Probe selection: 0 corresponds to Probe 1, and 1 corresponds to Probe 2.
TriggerMode	INT	0	Trigger mode: 0: Enable rising edge trigger; 1: Probe rising edge trigger; 2: Probe falling edge trigger; 3: Z-phase rising edge signal trigger.
Position	REAL	0.0	Pre-set position value.

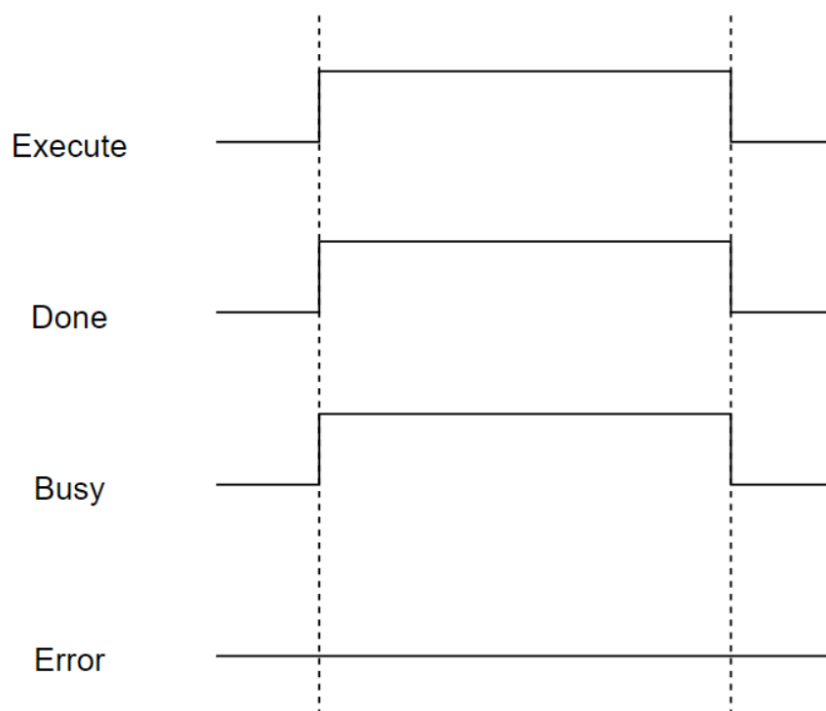
4.13.16.2 Output Parameter

Parameter	Type	Initial value	Description
Done	BOOL	FALSE	When the value is TRUE, it indicates that the function block has completed execution without detecting any errors.
Busy	BOOL	FALSE	When the value is TRUE, it indicates that the function block is currently being executed.
Error	BOOL	FALSE	When the value is TRUE, it indicates that an error has been

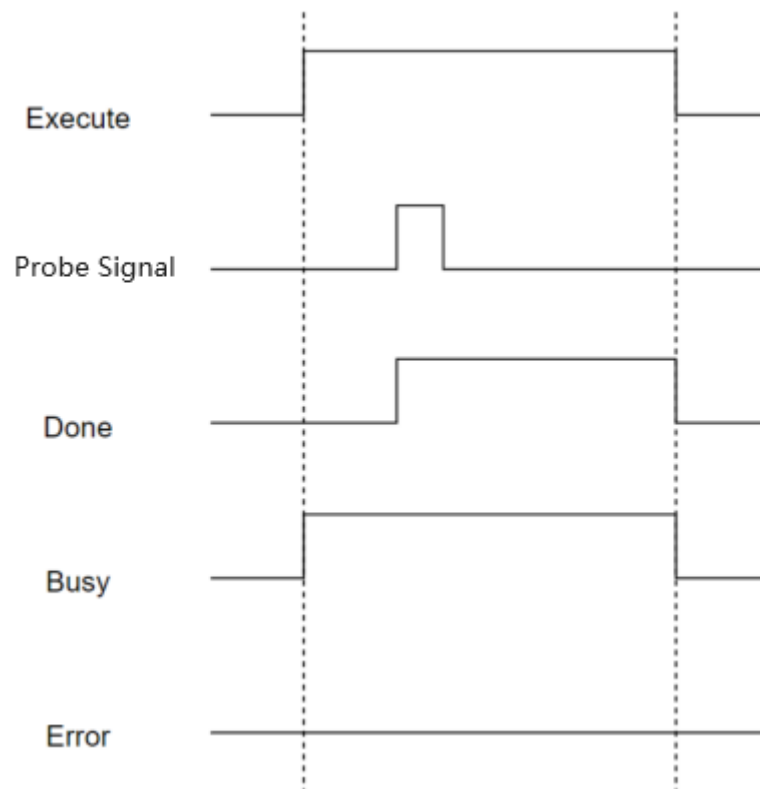
Parameter	Type	Initial value	Description
			detected, and the function block execution has been terminated.
ErrorID	WORD	-	Error code, for detailed information, please refer to Appendix C - Motion Control Error Codes .

4.13.16.3 Sequency Diagram

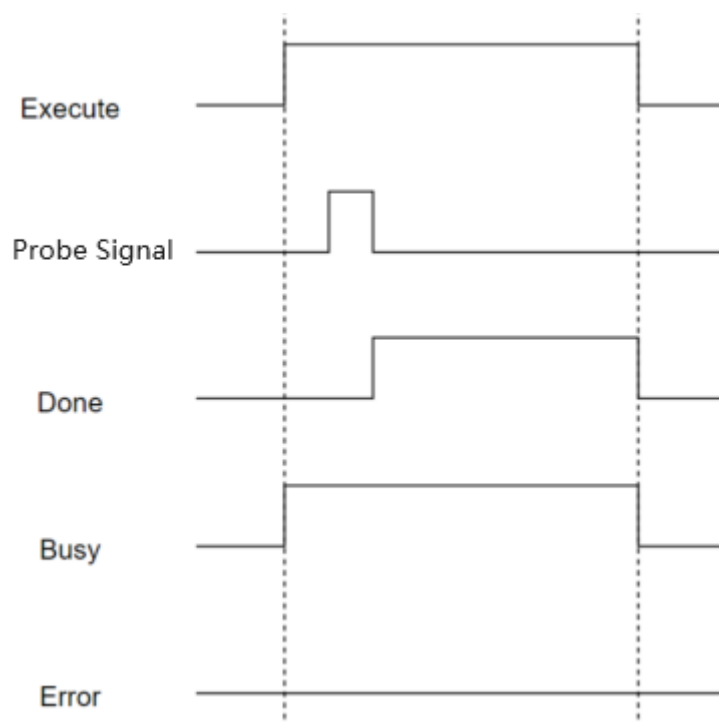
- ◆ When TriggerMode is 0, it is triggered by the rising edge of Execute.



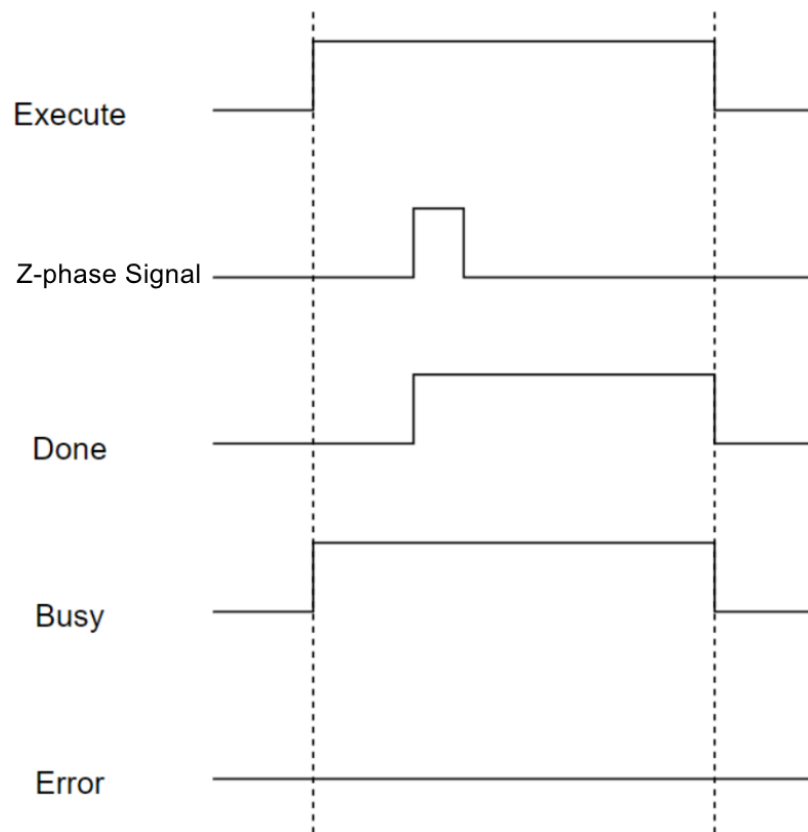
- ◆ When TriggerMode is 1, it is triggered by the rising edge of the probe.



- ◆ When TriggerMode is 2, it is triggered by the falling edge of the probe.

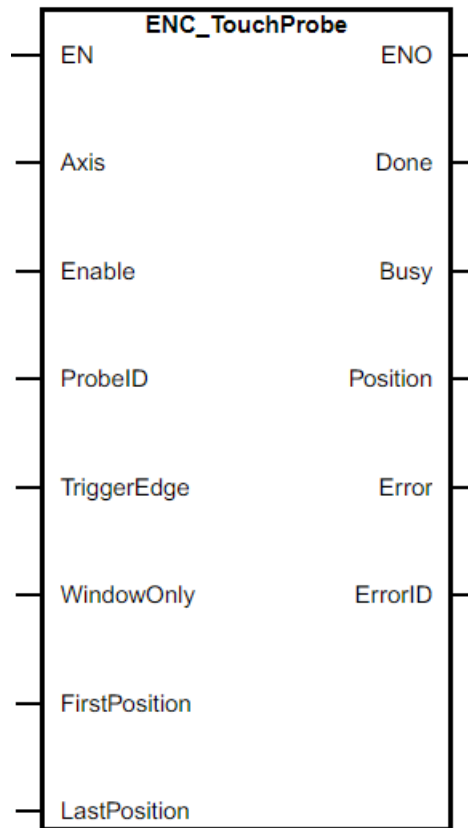


- ◆ When TriggerMode is 3, it is triggered by the rising edge of the Z-phase signal.



4.13.17 ENC_TouchProbe

This function block is used to obtain the position of the counting axis when the probe is triggered.



4.13.17.1 Input Parameter

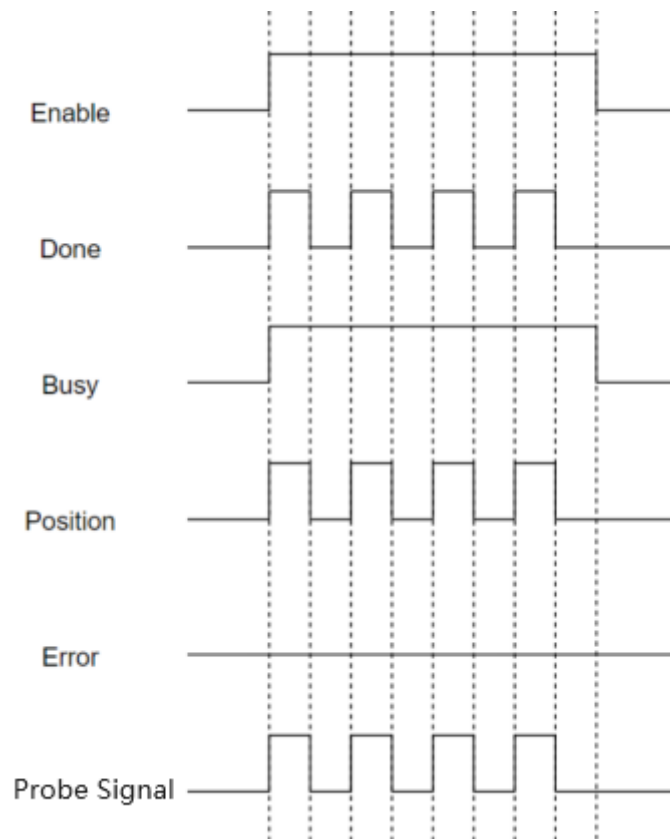
Parameter	Type	Initial value	Description
Axis	AXIS_REF	-	The name of the axis (instance) for executing the function block, customizable when adding the commands.
Enable	BOOL	FALSE	Start the function block when the value is TRUE.
ProbeID	INT	0	0 represents probe 1; 1 represents probe 2.
TriggerEdge	INT	0	0 represents rising edge triggering; 1 represents falling edge triggering.
WindowsOnly	BOOL	0	0 represents disabling the use of window functionality, detecting probes in all position ranges; 1 represents enabling window functionality, detecting probes only when the condition FirstPosition is less than or equal to the current position and the current position is less than LastPosition.
FirstPosition	REAL	0.0	Probe window start position.
LastPosition	REAL	0.0	Probe window end position.

4.13.17.2 Output Parameter

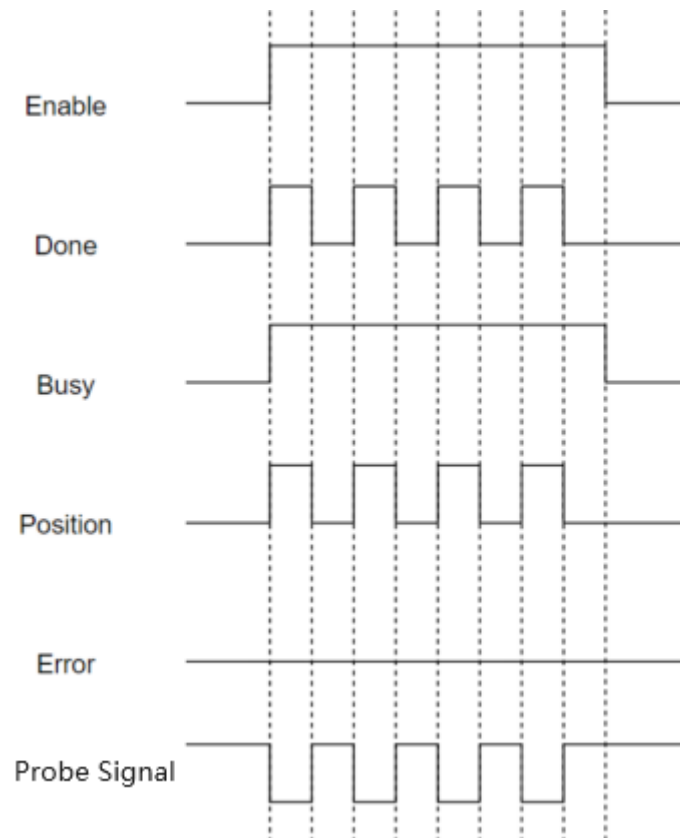
Parameter	Type	Initial value	Description
Done	BOOL	FALSE	When the value is TRUE, it indicates that the function block has completed execution without detecting any errors.
Busy	BOOL	FALSE	When the value is TRUE, it indicates that the function block is currently being executed.
Position	REAL	-	The latched count axis position when the probe is triggered.
Error	BOOL	FALSE	When the value is TRUE, it indicates that an error has been detected, and the function block execution has ended.
ErrorID	WORD	-	Error code, for detailed information, please refer to Appendix C - Motion Control Error Codes .

4.13.17.3 Sequency Diagram

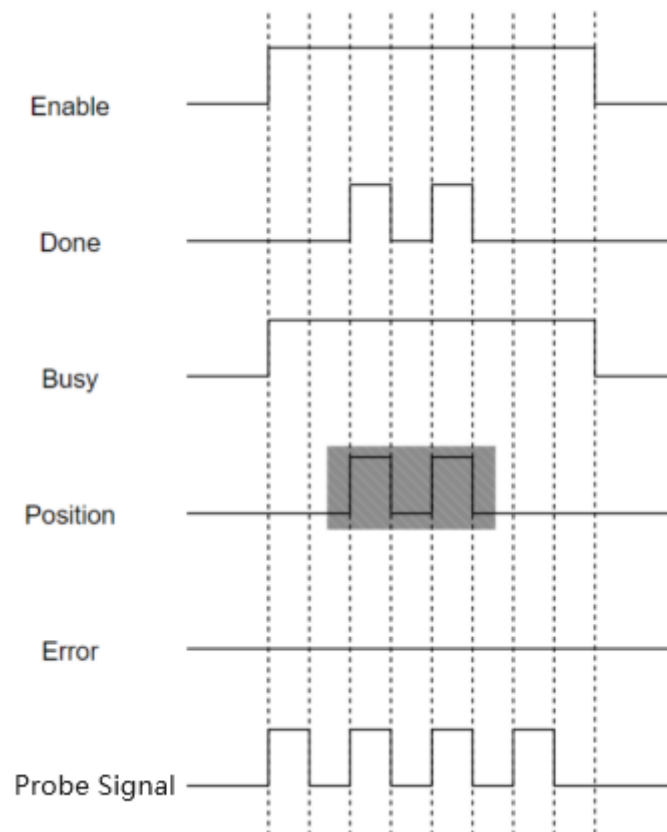
- ◆ When TriggerEdge is 0 and the probe window function is not enabled.



- ◆ When TriggerEdge is 1 and the probe window function is not enabled.

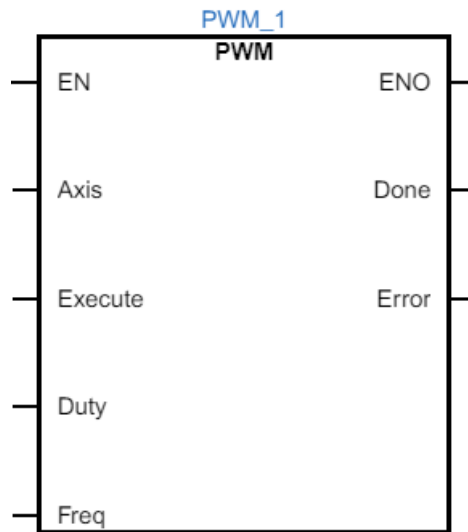


- ◆ When TriggerEdge is 0 and the probe window function is enabled, the area of shaded area in the following figure represents the range of FirstPosition to LastPosition, that is, the range of Window.



4.13.18 PWM

This function block is used to output PWM pulses.



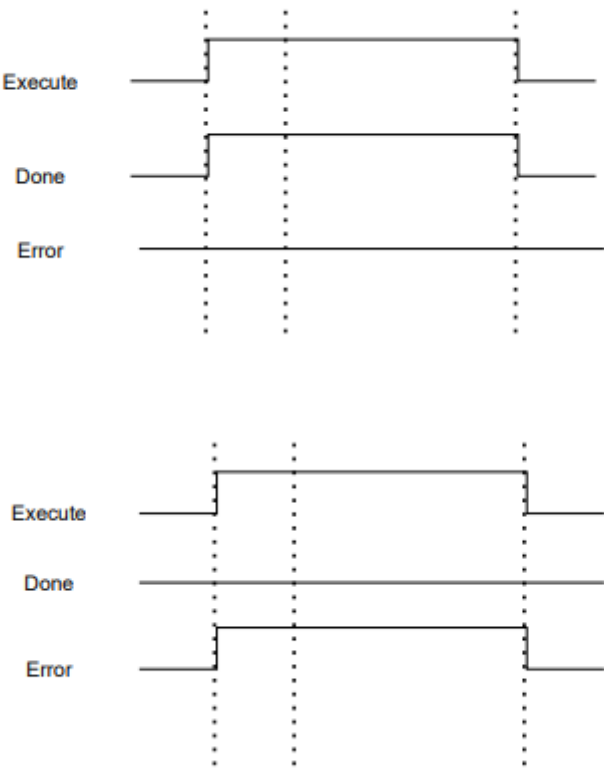
4.13.18.1 Input Parameter

Parameter	Type	Initial value	Description
Axis	AXIS_REF	-	The name of the axis (instance) for executing the function block, customizable when adding the commands.
Execute	BOOL	FALSE	Start the function block when the value is TRUE.
Duty	WORD	-	Duty cycle of the square wave, with a range of 0 to 1000, where the duty cycle is the set value divided by 1000.
Freq	UDINT	-	Pulse frequency, with a range of 2000Hz to 50000Hz.

4.13.18.2 Output Parameter

Parameter	Type	Initial value	Description
Done	BOOL	FALSE	When the value is TRUE, it indicates that the function block has completed execution without detecting any errors.
Error	BOOL	FALSE	When Error is TRUE, it indicates that an error has been detected, and the function block execution has ended.

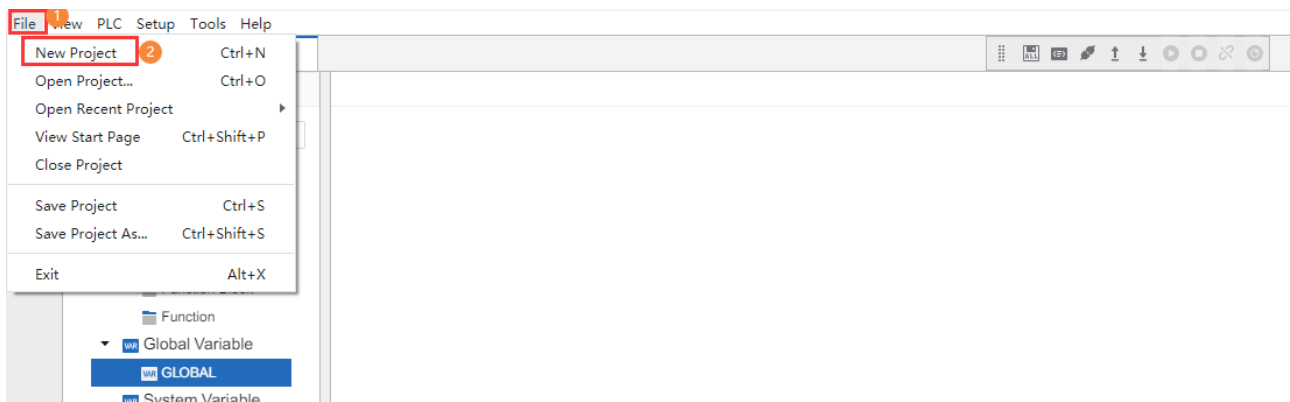
4.13.18.3 Sequency Diagram



5.1 Create a New Project

A project is the program and hardware configuration information for PLC operation, defining the control functions of the PLC. The procedure for creating a new project is as follows:

Step1. Select **File/New Project** in the menu bar (shortcut key “Ctrl+N”).



Step2. Configure the relevant information in the pop-up dialog box and click **Confirm**.

New Project

* Name :

NewProject

Location :

C:\Users\Ivan\Documents\MCR Master\Projects

PLC Model/Hardware Version

MCR7-DP16

V3.X

Project Version/Template :

1.20.0

Nonuse

☐ Set as default

Reset Defalut Setting

Author :

Desc :

Please input...

Confirm

Cancel

For detailed configuration method, please refer to the table below.

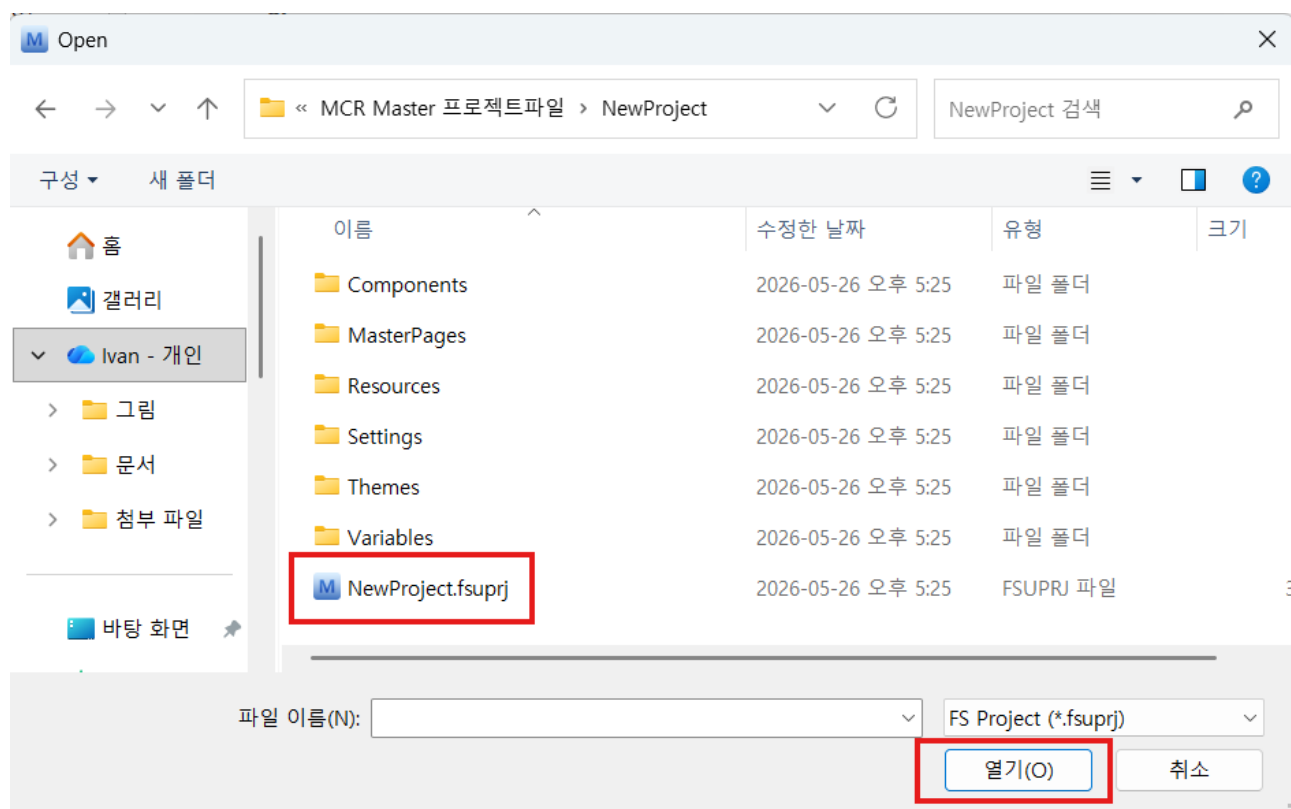
Parameters	Description
Name	Not exceeding 64 characters. Only supports Chinese, English, numbers, and the special character “_”.

Parameters	Description
Location	The save location of the project file.
Project Type	Select PLC .
PLC Model/Firmware Version	Select the PLC model and firmware version. The firmware version must match the MCR Master version.
Address Template	Use an address template for easy import of variables from third-party PLCs: ◆ Mitsubishi FX Series Address Template ◆ Omron CP Series Address Template ◆ Siemens S7-200 Series Address Template By default, the address template is not used.
Author	The author of the project.
Description	Description information for the project.

5.2 Open the Project

Step1. In the menu bar, select **File/Open Project** (shortcut key “Ctrl+O”).

Step2. In the pop-up dialog box, navigate to the directory containing the project file, select a file with the .fsuprj extension, and click **Open**.

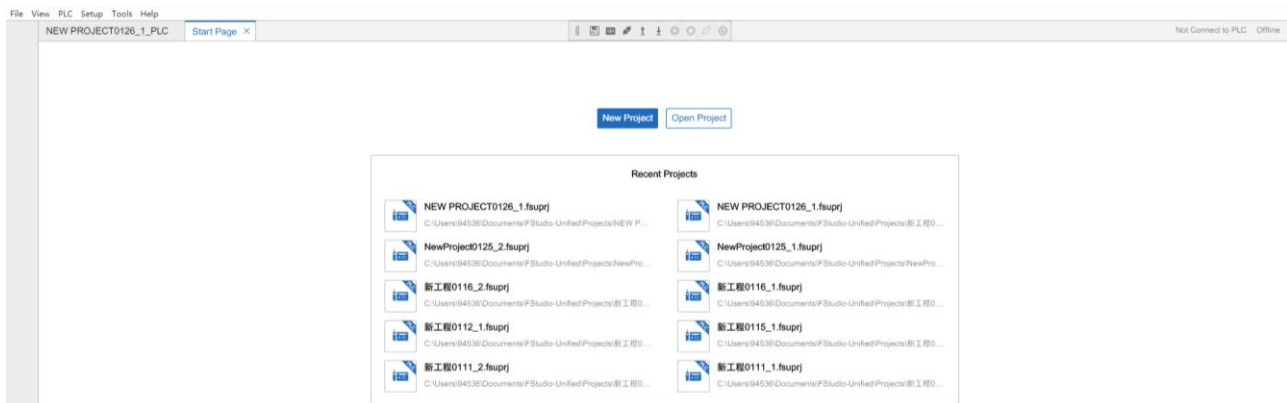




You can directly navigate to the project file directory, double-click the project file with the .fsuprj extension, and open the project directly.

5.3 View the Start Page

In the menu bar, select **File/View Start Page** to enter the **Start Page** interface and view recently used projects. Click **New Project** to create a new project and click **Open Project** to open a specific project.



5.4 Close the Project

Close the current project without exiting the current program, used for switching projects. The operation is as follows:

In the menu bar, select **File/Close Project** to close the current project and return to the **Start Page** interface.

5.5 Save the Project

Select **File/Save Project** in the menu bar (shortcut key “Ctrl+S”) to save the current project.

5.6 Save the Project as

The “Save Project As” operation is convenient for modifying a project based on the existing one while preserving the original project. Follow these steps:

Step1. Select **File/Save Project As** in the menu bar.

Step2. In the pop-up dialog box, set the name, select the save location, and click **confirm**.

Save Project As...

×

* Name :

FLSHINGLIGHT

Location :

C:\Users\74152\Desktop\233



Confirm

Cancel

5.7 Exit the Program

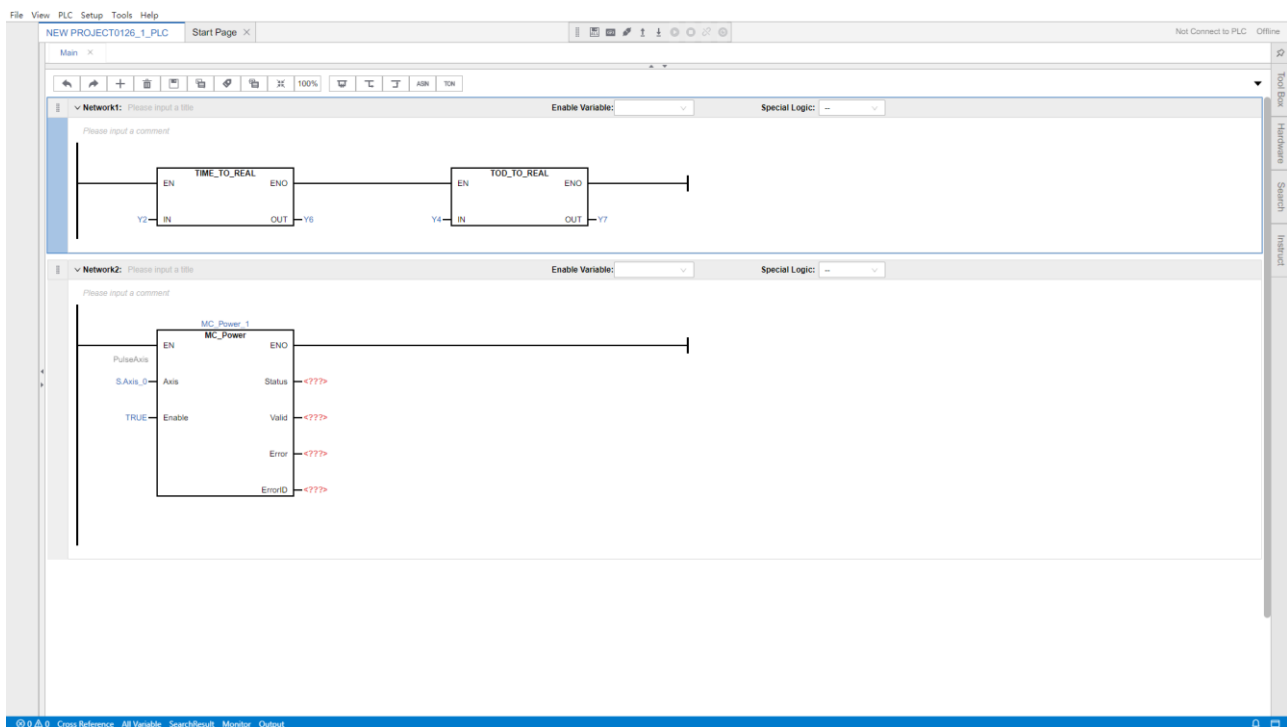
Select **File/Exit** in the menu bar (shortcut key “Alt+X”) to close the current project and exit the program.

6 View

The View is used to adjust the software interface layout and can restore the default workspace layout.

6.1 Fullscreen

Select **View/Fullscreen** in the menu bar (shortcut key “F11”) to enter full-screen view, as shown in the figure below. To exit full-screen view, select **View /Full Screen** in the menu bar.



6.2 Restore Default Workspace Layout

Select **View /Reset Workbench Layout** in the menu bar to restore the default workspace layout.

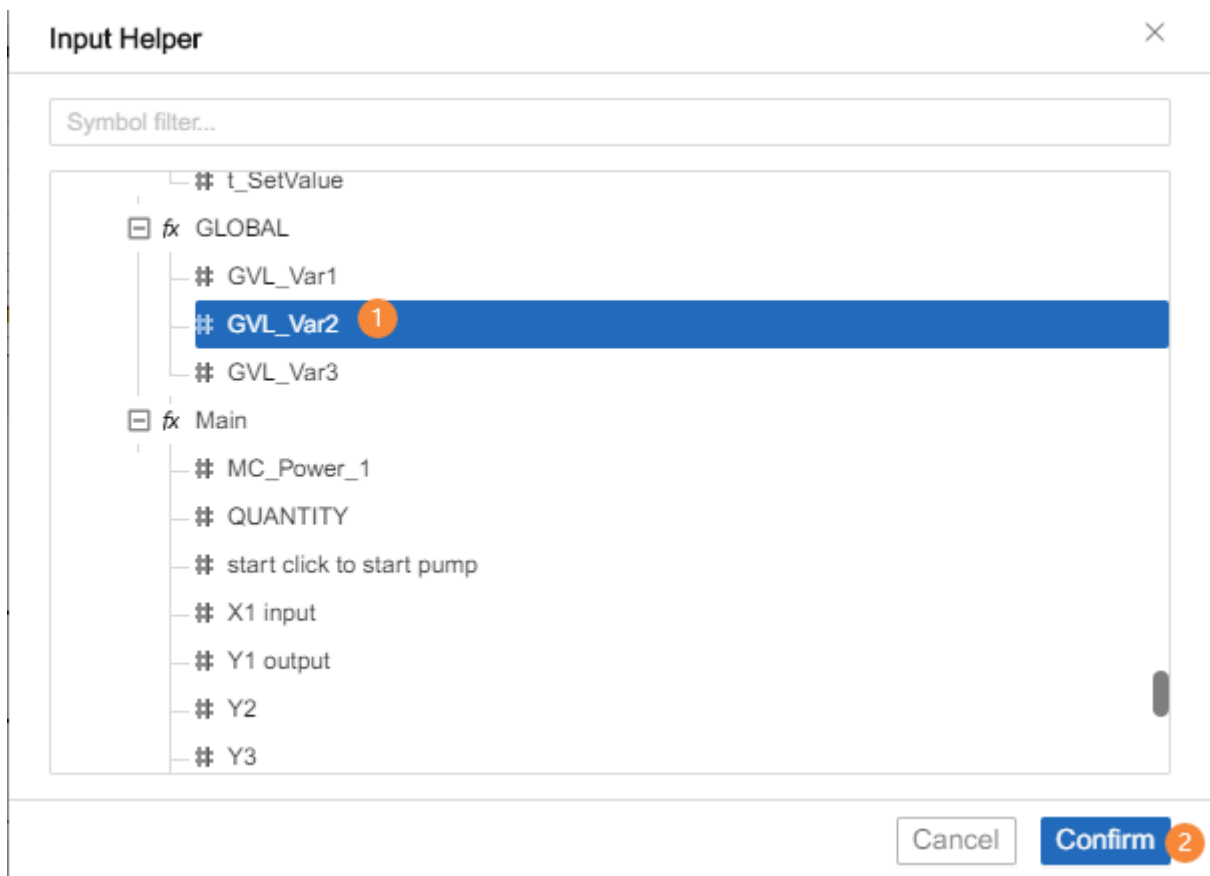
6.3 Cross Reference

The cross reference shows the usage of variables. Follow these steps to view the cross reference:

Step3. Select **View /Cross Reference** in the menu bar. The **Cross Reference** list box will appear at the bottom left of the software interface. Click the  icon.



Step4. In the pop-up dialog box, select the variable and click **Confirm**.

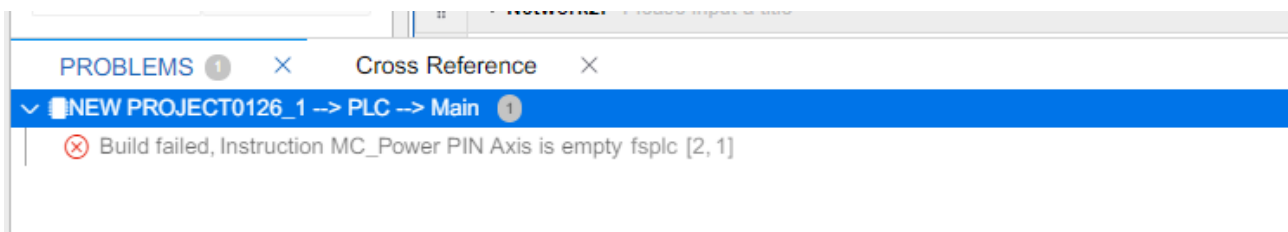


Step5. You can now view the references for that variable, such as the called POU, access type, calling location, and effective range.

PROBLEMS x Cross Reference x					
G.OVL_Var2					
Variable:GVL_Var2 Type:INT Address:0x0 GLOBAL					
Symbol	POU	Access	Position	Range	Comment
GVL_Var2	GLOBAL	DECLARE		GLOBAL	

6.4 Problem

Problem refer to errors found during the project compilation. To view the error messages for the project, select **View/PROBLEMS** from the menu bar.



6.5 All Variables

To view all variable information for the project, select **Window/All Variable** from the menu bar. The window with **All Variable** in the project will appear at the bottom of the software interface.

POU	Variable	Type	Address	Range	Used Times	Comment
SYSTEM	Always_On	BOOL		System	0	Always connected
SYSTEM	Axis_0	AXIS_REF		System	1	Axis_0_Hsp
SYSTEM	Clock_100ms	BOOL		System	0	This bit provides a clock pulse. The bit is FALSE
SYSTEM	Clock_10ms	BOOL		System	0	This bit provides a clock pulse. The bit is FALSE
SYSTEM	Clock_10s	BOOL		System	0	This bit provides a clock pulse. The bit is FALSE
SYSTEM	Clock_1s	BOOL		System	0	This bit provides a clock pulse. The bit is FALSE
SYSTEM	Clock_2s	BOOL		System	0	This bit provides a clock pulse. The bit is FALSE
SYSTEM	Clock_5s	BOOL		System	0	This bit provides a clock pulse. The bit is FALSE
SYSTEM	Clock_90s	BOOL		System	0	This bit provides a clock pulse. The bit is FALSE
SYSTEM	CPU_ID	DWORD		System	0	An identify to distinguish CPU model
SYSTEM	ExBus_Err	BOOL		System	0	Expansion bus is abnormal
SYSTEM	First_Scan_On	BOOL		System	0	The CPU sets this bit to TRUE for the first scan

6.6 Monitoring

In the online monitoring state, users can monitor real-time status information for specific variables. The operation method is as follows:

Step6. In the online monitoring state, click **Monitor**.

Step7. In the pop-up monitoring list, double-click on the cell under the **Expression** column, right-click on the cell, select the variable you want to monitor, and you can view the current status information of the monitored variable.

The screenshot shows the software interface with the 'Monitor' window open. The window displays a table with columns: Expression, Data Type, Current Value, Prepare Value, Address, and Comment. The table lists variables like start, X1, Y1, QUANTITY, and Y2. Below the table, there is a ladder logic diagram showing a network with a timer T100ms and a variable Y6. The bottom status bar shows the 'Monitor' tab is active.

6.7 Output

The **Output** interface allows you to view the compilation output results of the project, including errors, warnings, and other information. If the project compiles successfully, it will output information about the memory usage.

Click on **Output** in the bottom left corner of the software interface, and in the pop-up **Output** window, you can view the compiler's output information.

NEW PROJECT0126_1_PLC

Task

Config

+

-

Q

POU

PRG

(LD) Main

(LD) M1

Function Block

Function

Global Variable

GLOBAL

System Variable

Custom Data Type

Task Config

Trace

POU View

Machine View

Main

×

Add

Import

E

	#	Name
☐	1	start
☐	2	X1
☐	3	Y1

↶

↷

+

🗑

⋮

Network1: Please inp

Please input a comm

Y2

⋮

Network2: Please inp

Please input a comm

Output

×

Start to prepare project to build structured text...

Structured text build successfully

Start to pre-compile, prepare project file for compiling...

Start to generate PLC config file...

Generate PLC config file successfully

Finish preparing the project file, pre-compile successfully

Start to compile and build project files into executable binary file...

Memory region	Used Size	Region Size	%age Used
app_rom:	354748 B	2 MB	16.92%
app_ram:	3348 B	192 KB	1.70%

Finish compiling the project, build executable binary file successfully

0

Cross Reference

All Variable

SearchResult

Monitor

Output

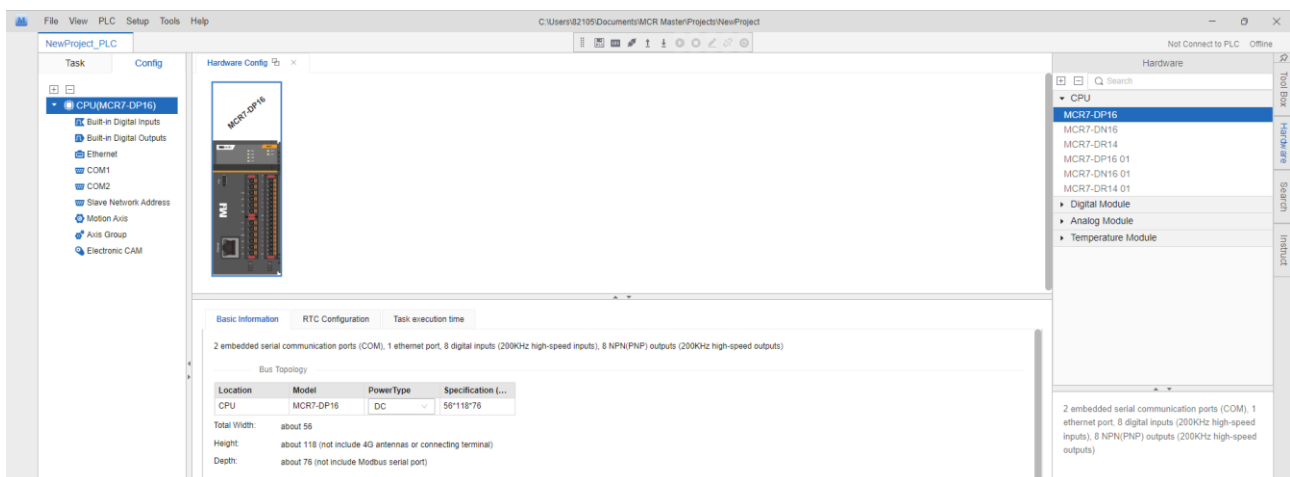
7 PLC Hardware Configuration

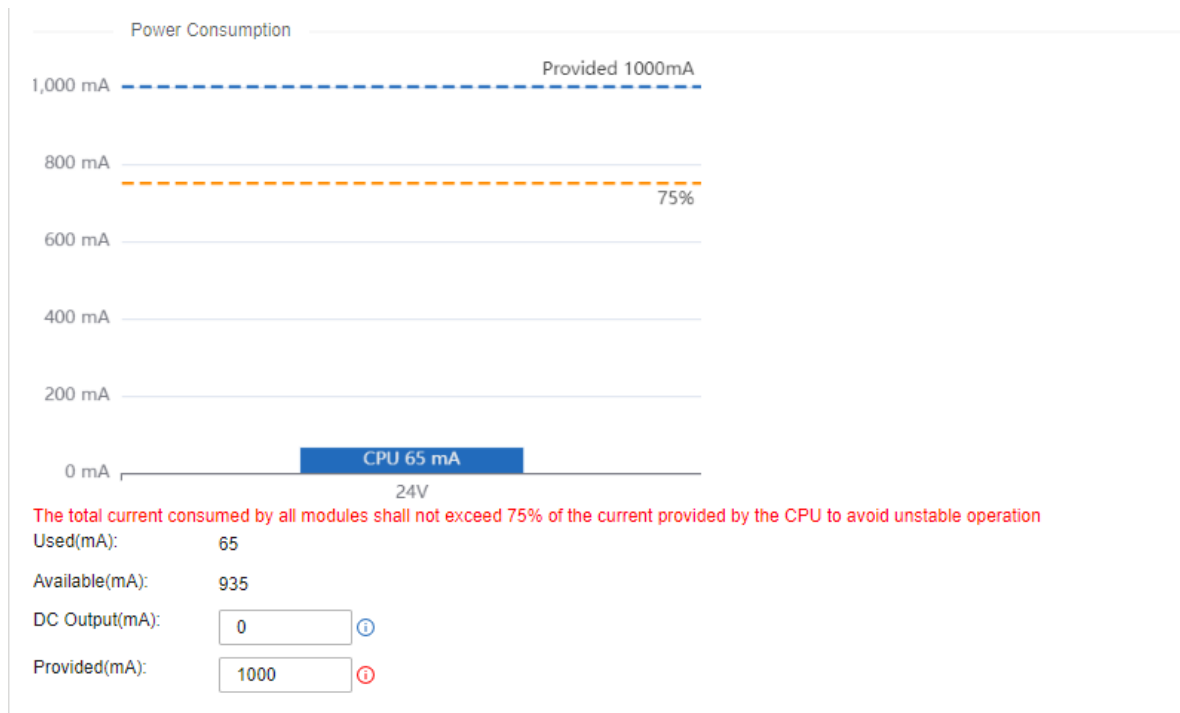
MCR Master provides a visual way to configure PLC hardware.

7.1 Configure the CPU Unit

7.1.1 Basic Information

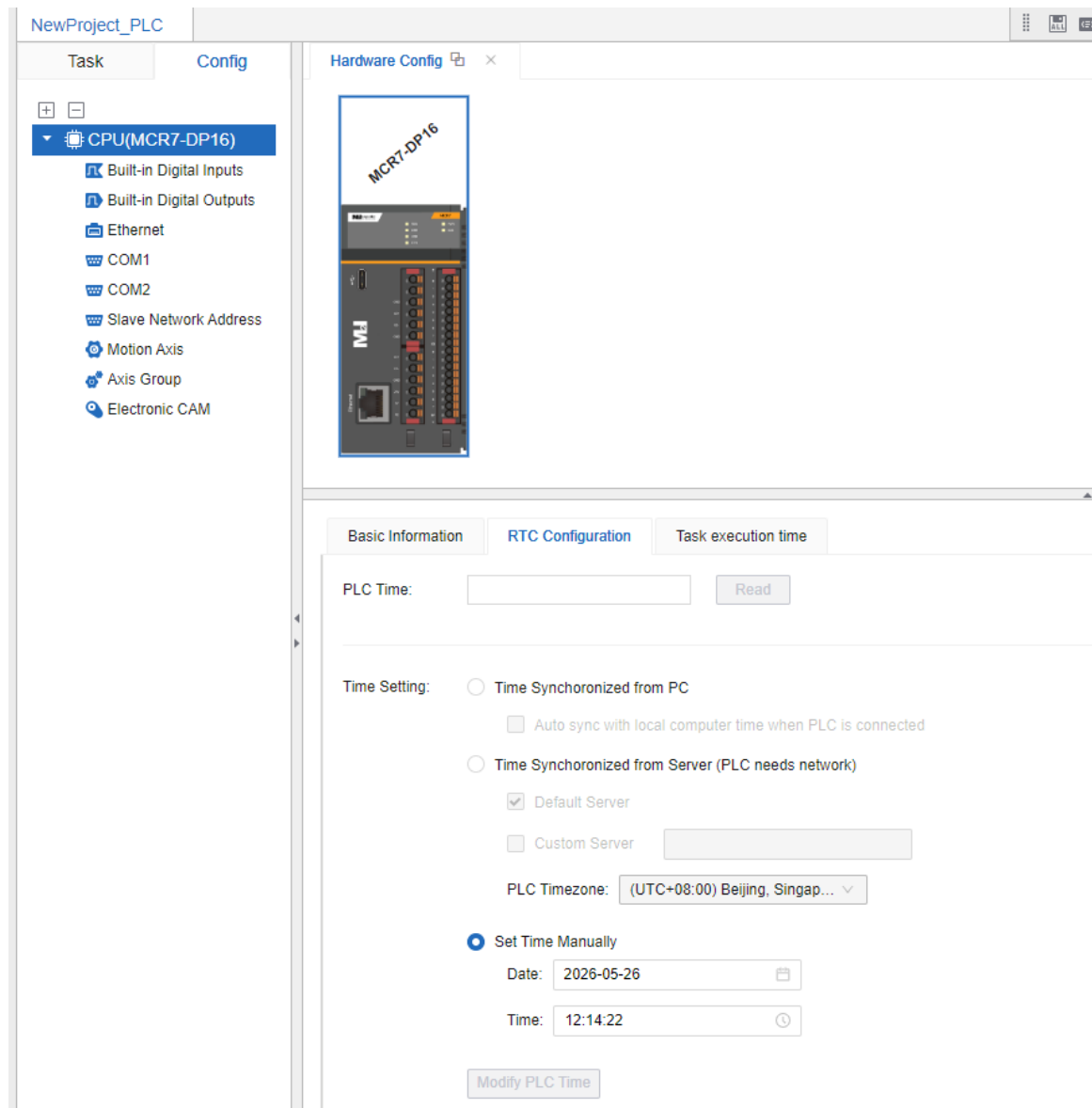
Select **Configuration** in the **Task/Configuration** area, click **Hardware** on the right side of the software interface to expand hardware module information. Select the CPU unit and drag it to the hardware diagram. In the **Basic Information** section, configure the CPU's power type and DC output current.





7.1.2 RTC Configuration

Select the **RTC Configuration** tab, click **Read** to retrieve the system time of the PLC.

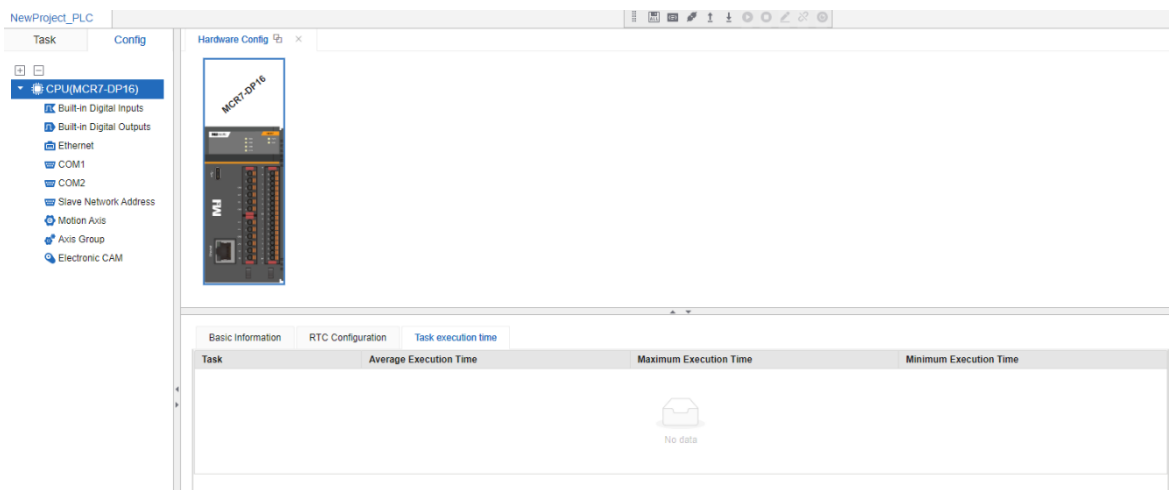


Setting the PLC system time supports three methods.

- ◆ Time Synchronized from PC : Check the option to **Auto Synchronize with local computer time when PLC is connected**. When connected to the PLC, the PLC's system time will be synchronized with the PC's system time.
- ◆ Time Synchronized from Server : Get the time from the server.
- ◆ Set Time Manually: Select **Write Manual Time to PLC**, which writes the PC's time to the PLC. Make sure the PLC is connected. Alternatively, Select **Write Bellow Time to PLC**, manually set the system time, and write it to the PLC.

7.1.3 Task Execution Time

Select the **Task execution time** tab to view information about task execution time.

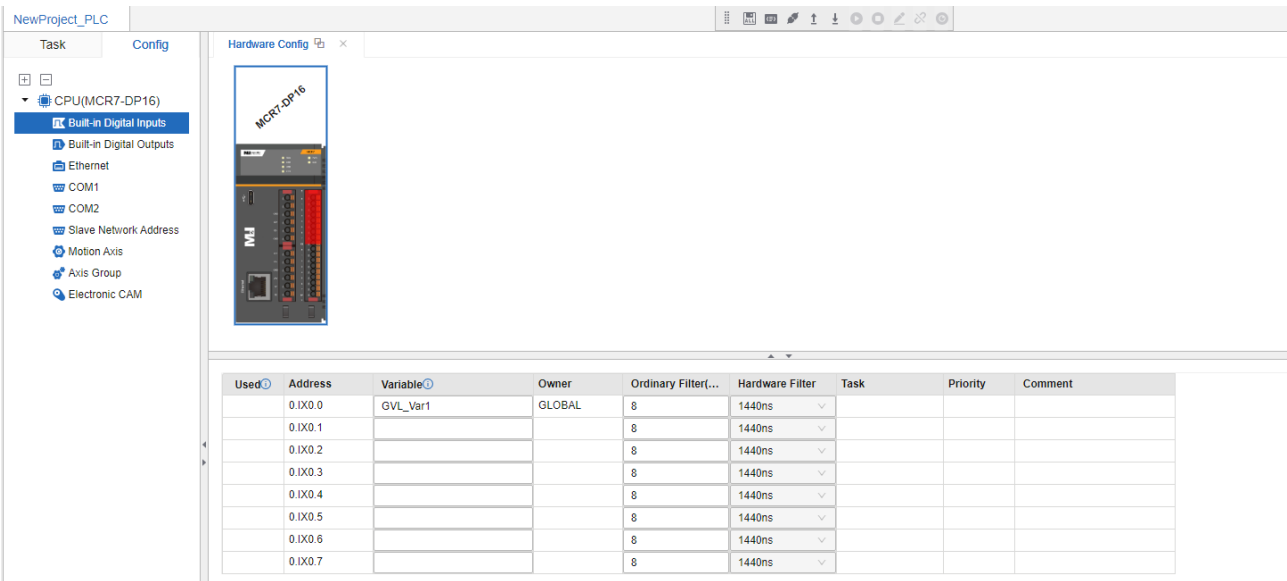


7.1.4 Built-in Digital Input

Built-in digital input is the process of converting various switch signals into signals required by the CPU for logical control, commonly used for button signals.

To configure built-in digital input, follow these steps:

In the **Configuration** area, select **Built-in Digital Inputs** to configure the channel information for built-in digital input.



For detailed configuration method, please refer to the table below.

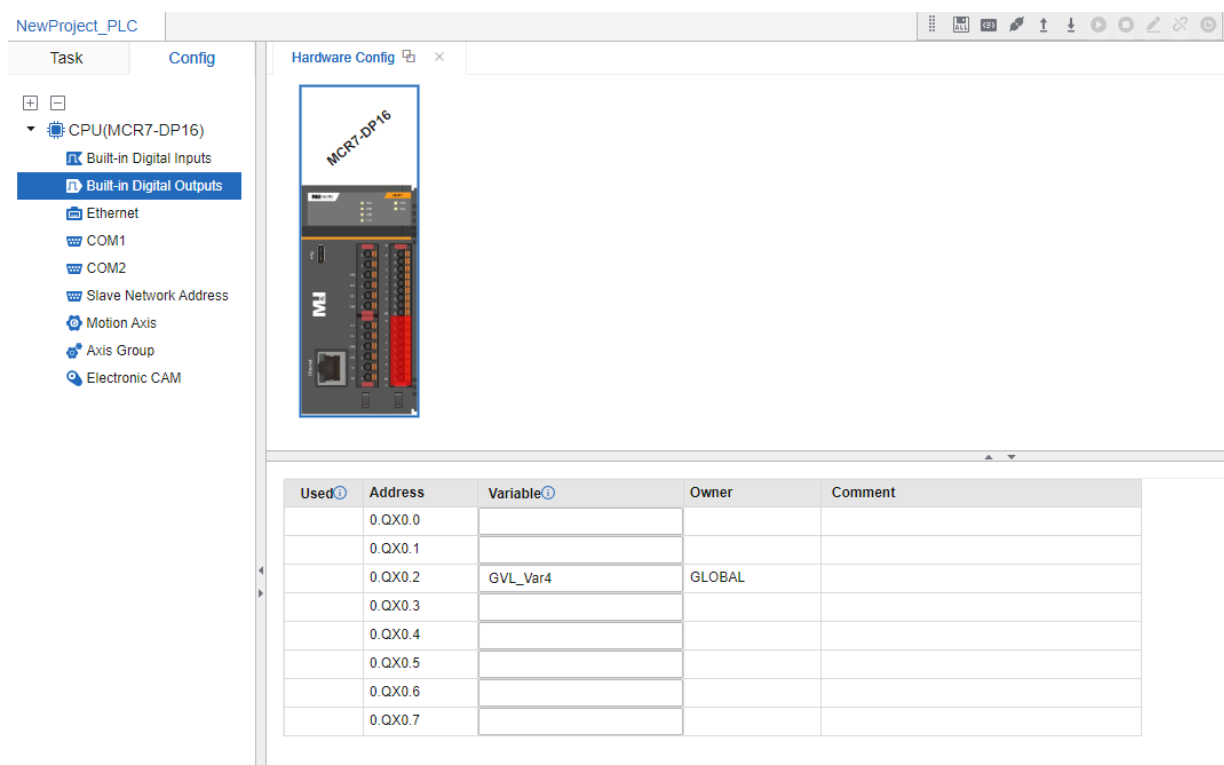
Parameters	Description
Used	If it is bound by global variables/local variables or used in the motion axis configuration area, the icon ☒ will be displayed.
Address	The register address corresponding to the variable.

Parameters	Description
Variable	The variable bound to the address. Supports Chinese, English letters, numbers, and the special character “_”, and cannot start with a number.
Owner	Indicates where the variable is used. It can be the name of the global variable table or the name of the motion axis. It is empty when used as a local variable.
Filter(ms)	Signals are recognized only when they persist, mainly to reduce interference. The value can be adjusted based on the actual situation; the default is 8.
Latch	Latches the input signal in the current logical state (e.g., high or low level) and stores it in the current state until the next refresh or change.
Task	The event-type task that calls the variable. For detailed information about tasks, please refer to Task Configuration .
Priority	The priority of the task being called.

7.1.5 Built-in Digital Output

Built-in digital outputs are used to output information obtained via PLC operations, and control various industrial field devices via external connections to the PLC.

In the **Configuration** area, select **Built-in Digital Outputs** and set the channel information for the built-in digital output.



Used	Address	Variable	Owner	Comment
	0.QX0.0			
	0.QX0.1			
	0.QX0.2	GVL_Var4	GLOBAL	
	0.QX0.3			
	0.QX0.4			
	0.QX0.5			
	0.QX0.6			
	0.QX0.7			

Please refer to the table below for detailed configuration methods.

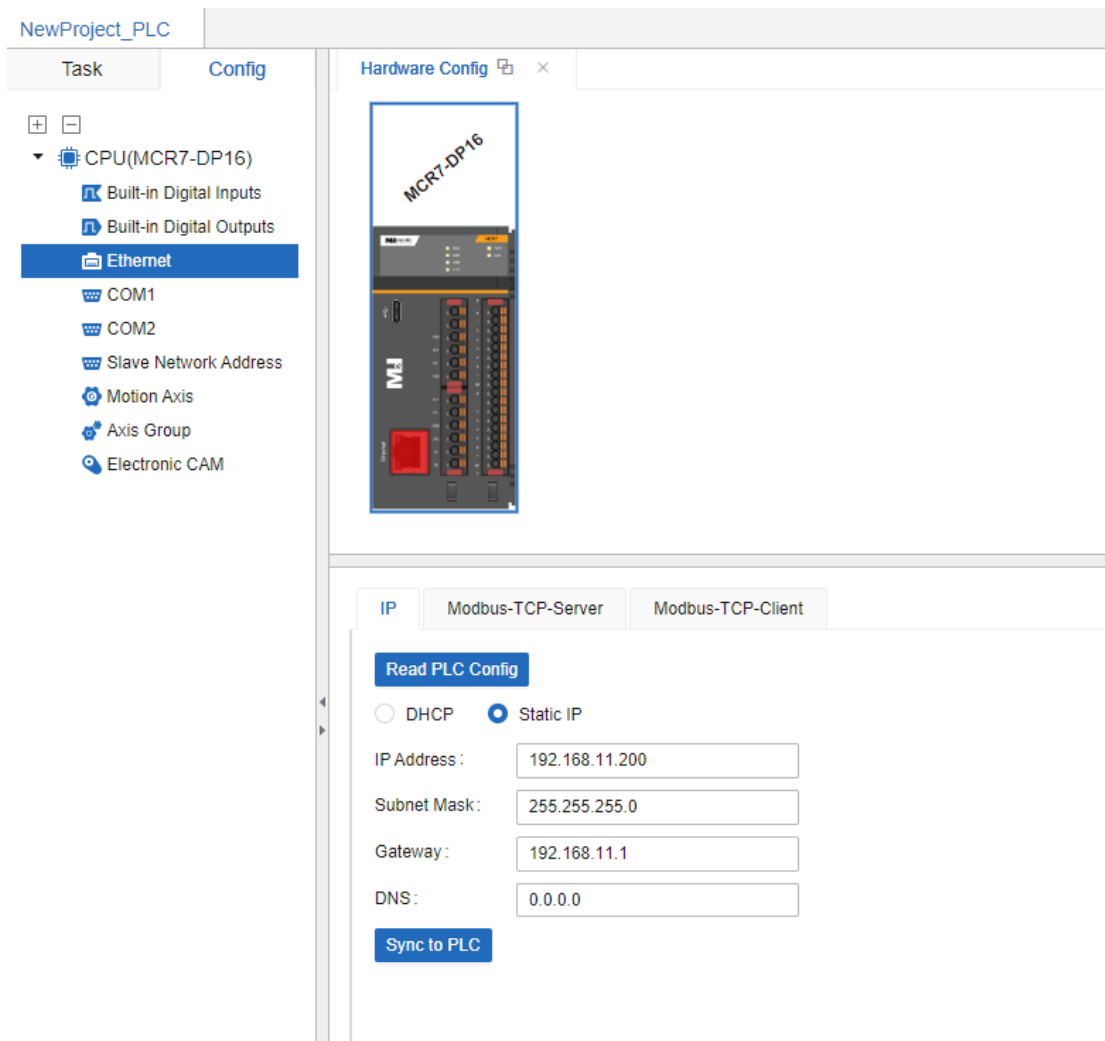
Parameters	Description
Used	If the address is bound by global variables/local variables or used in the motion axis configuration area, the icon  will be displayed.
Address	The register address corresponding to the variable.
Variable	The variable bound to the address. Supports Chinese, English letters, numbers, and special characters “_” but cannot start with a number.
Owner	Indicates where the variable is used. It can be the name of the global variable group or the name of the motion axis. It is empty when used as a local variable.

7.1.6 Ethernet Configuration

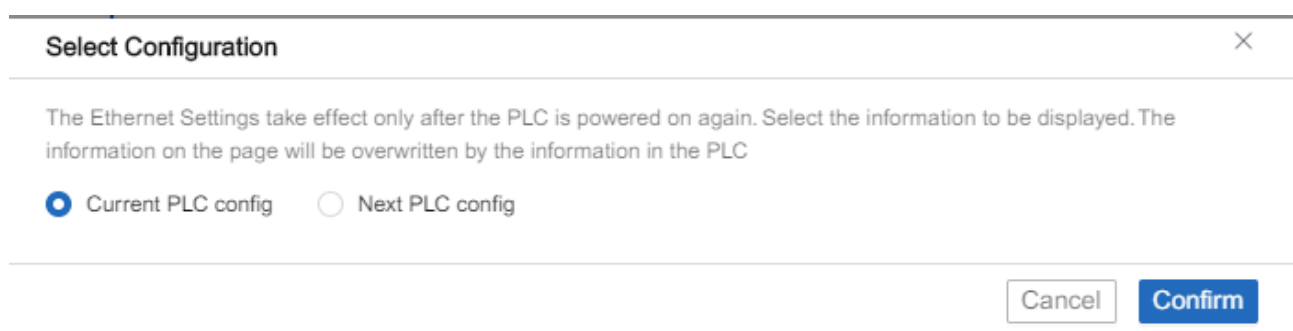
Ethernet parameters of the PLC are configurable, mainly including IP, Modbus-TCP Server, and Modbus-TCP Client.

7.1.6.1 Setting the PLC's IP Address

Step8. In the **Configuration** area, select **Ethernet**, select the **IP** tab, and click **Read PLC Config**.



Step9. In the pop-up dialog, select either **Current PLC config** or **Next PLC config**. Click **Confirm** to display the PLC's current configured IP address or the IP address that will take effect after a power cycle.



Step10. Select the IP address acquisition method (DHCP for automatic IP address assignment or Static IP for a static IP address). Click **Sync to PLC** to set the PLC's IP address.

Read PLC Config

☐ DHCP ☒ Static IP

IP Address : 192.168.101.116

Subnet Mask : 255.255.255.0

Gateway : 192.168.101.1

DNS : 192.168.101.1

Sync to PLC



After modifying the IP address of the PLC, power off and power on to restart the PLC to take effect.

7.1.6.2 Configure Modbus-TCP-Server

PLC acts as a Modbus TCP server, and the configuration method is as follows:

Select the **Modbus-TCP-Server** tab, toggle the **Enable** Switch to the “ON” state, set the port number, and choose whether to hide the station ID.

NewProject_PLC

Task Config

Hardware Config

MCR7-DP16

Modbus-TCP-Server

Enable: ☒ The maximum connections count of Modbus TCP Server and Client at the same time is 12 while PLC running, the PLC will reject new connection requests after reaching the limit.

Port: 502 Hide StationId: ☐ Station ID: 1

Restrict Communication: ☐

Modbus Slave Network Address Configuration

Export

Please refer to the table below for detailed configuration methods.

Parameters	Description
Enable	Whether to enable PLC as a Modbus/TCP server.

Parameters	Description
Port	The port number occupied by PLC's Modbus/TCP service; default is 502.
Hide StationID	Whether to hide the station ID of PLC.
Station ID	A number used to identify the Modbus/TCP server of PLC. The value ranges from 0 to 255, with 0 representing the broadcast address.

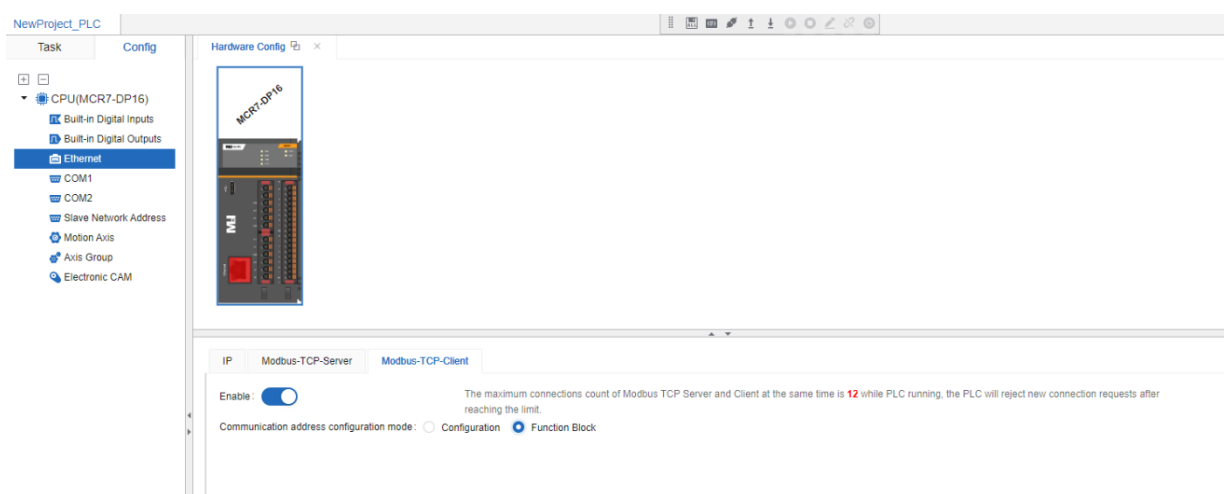
7.1.6.3 Configure Modbus-TCP-Client

When PLC acts as a Modbus TCP client and communicates with a Modbus TCP server, follow these steps:

Select the **Modbus-TCP-Client** tab and toggle the **Enable** switch to the “ON” state. Select the communication address configuration method.

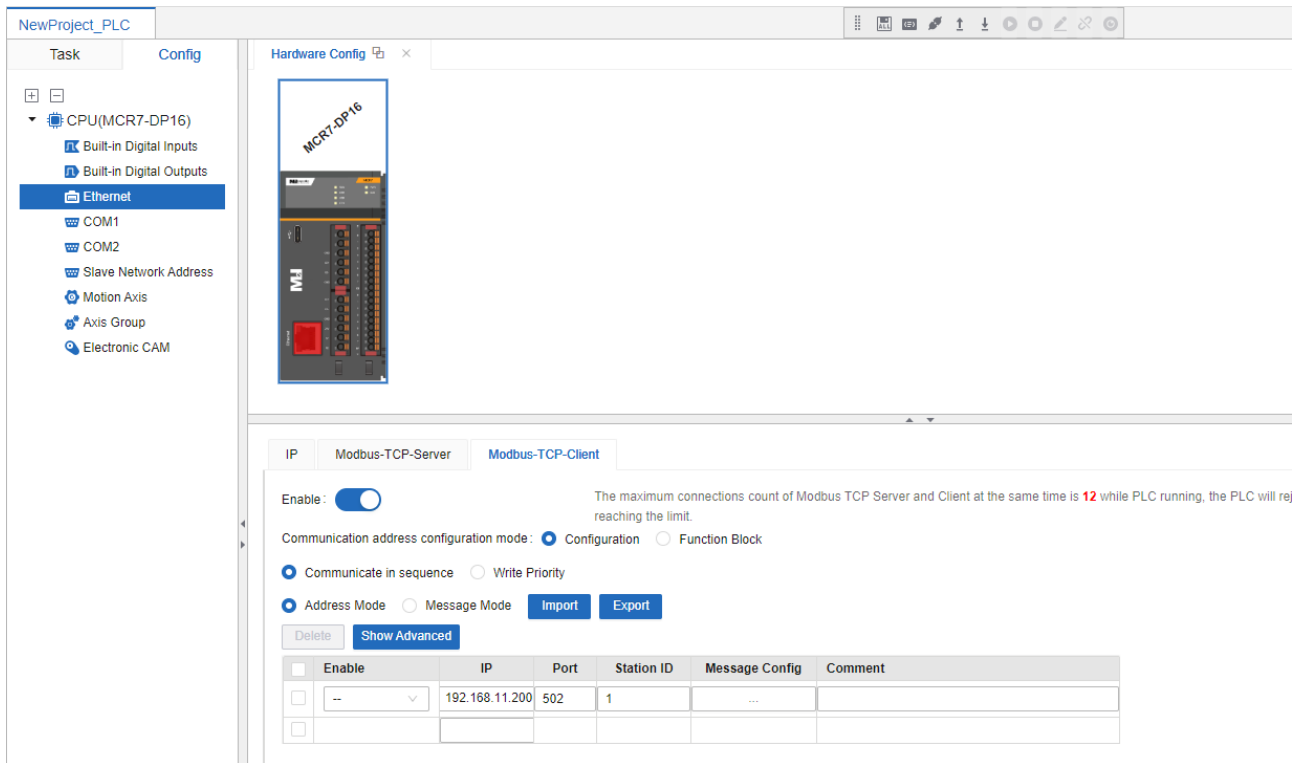
◆ Select Function Block

Based on this configuration, in PLC programming (ladder diagram and ST programming), use relevant instructions such as ARRAY_ADDR_BOOL (get BOOL array address), ARRAY_ADDR_WORD (get WORD array address), MODBUS_RW_BIT (serial port receive/send bit data instruction), MODBUS_RW_WORD (serial port receive/send WORD data over Modbus Communication), MODBUS_WRITE_STRING (write STRING type data over Modbus communication), MODBUS_READ_STRING (read STRING type data over Modbus Communication) for Modbus communication.



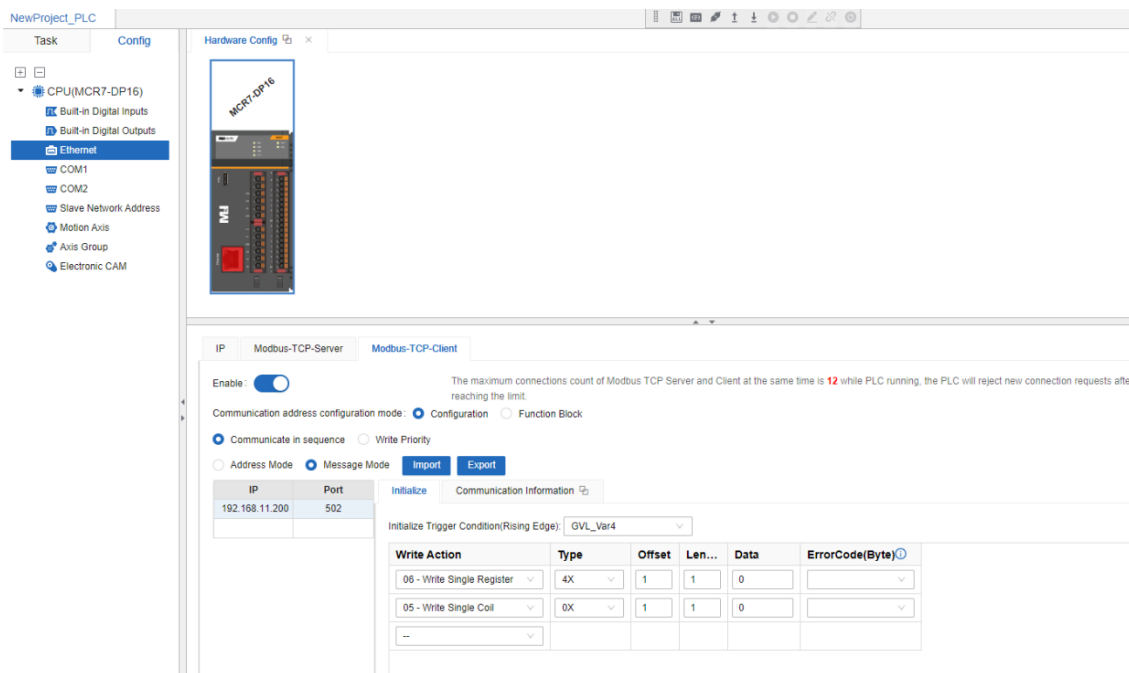
◆ Select Configuration

- When selecting the **Address mode**, select the enable method, configure IP, port, and station ID.



- When selecting the **Message Mode**.

1) Set communication information.

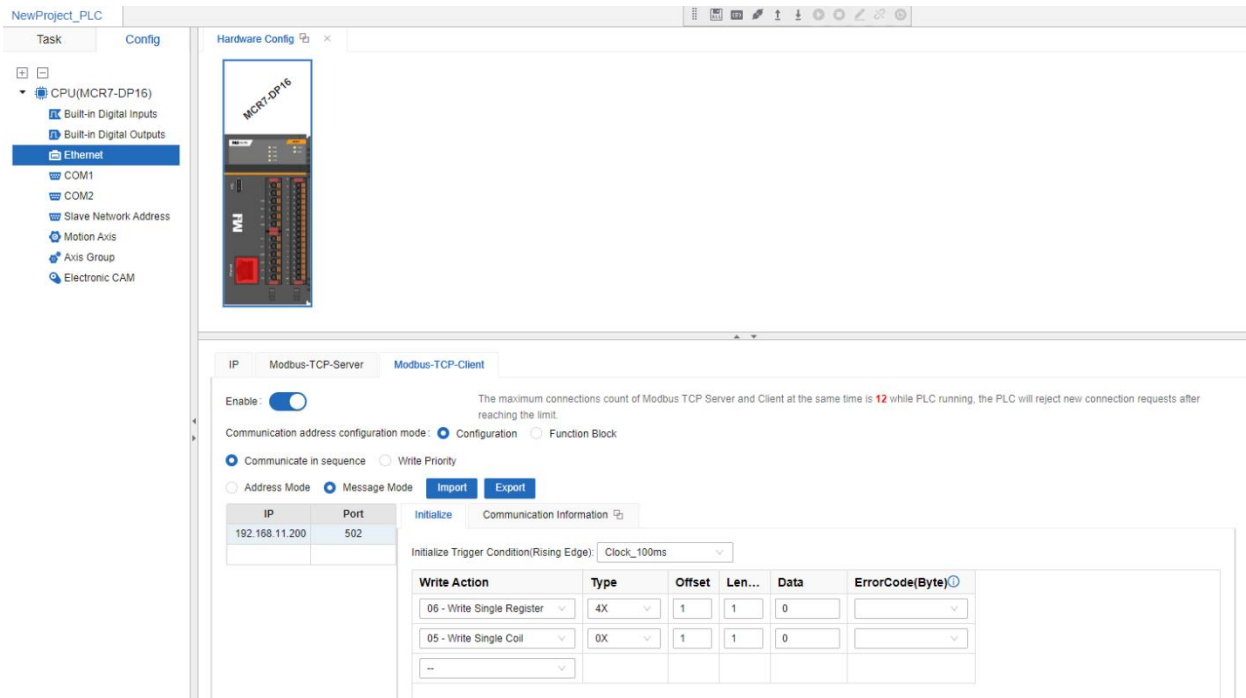


Please refer to the table below for detailed configuration methods.

Parameters	Description
Read/Write	◆ Read Coils: Read the value from slave 0X address and assign it to a global variable. ◆ Read Input Discrete: Read the value from slave 1X address and assign it to a global variable.

Parameters	Description
	◆ Read Holding Registers: Read the value from slave 4X address and assign it to a global variable. ◆ Read Input Registers: Read the value from slave 3X address and assign it to a global variable. ◆ Write Single Coil: Write the value at slave 0X address, the value changes with the global variable of the master. ◆ Write Single Register: Write the value at slave 4X address, the value changes with the global variable of the master. ◆ Write Multiple Coils: Write the status of multiple 0X addresses at the slave, corresponding one-to-one with the global variable status of the master.
Data Type	Please refer to the actual situation.
Global Variable	Add global variables of the specified data type to read or write values.
Type	◆ 0X: Bit address type, read-write. Typically refers to coil status. ◆ 1X: Bit address type, read-only. Typically refers to discrete input status. ◆ 3X: Word address type, read-only. Typically refers to holding registers. ◆ 4X: Word address type, read-write. Typically refers to input registers.
Offset	Address offset.
Length	Data length in bytes.
Trigger Type	◆ Circular: Perform read-write operations within a specified period. ◆ Variable: Execute read-write operations when a specified variable meets the trigger condition.
Trigger Condition	◆ When the trigger type is “Circular”, set the task cycle. ◆ When the trigger type is “Variable”, select the variable and trigger condition (including High Level, Low Level, Rising Edge, Falling Edge).
Error Code	Communication status corresponding code, please refer to Appendix A - Communication Code Description for details.
Comment	Description information for read/write operations.

2) Set the initialization parameters.



Please refer to the table below for detailed configuration methods.

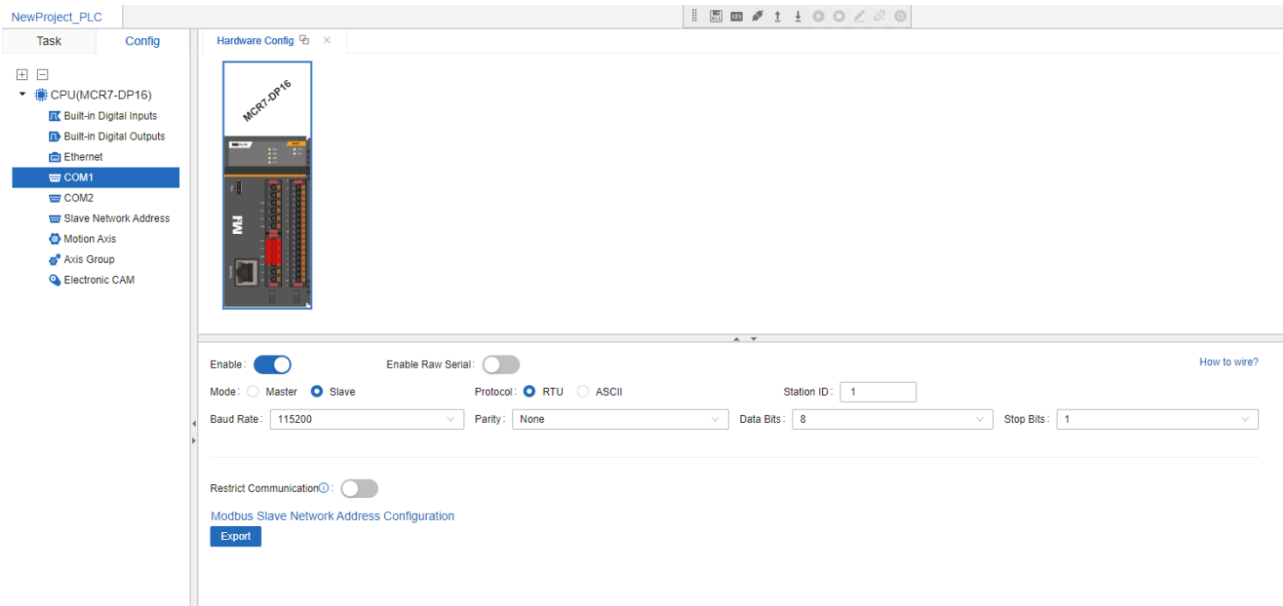
Parameters	Description
Initialize Trigger Condition (Rising Edge)	To initialize the data for the slave addresses 0X/4X, you can bind a global variable or a system variable and trigger it by the rising edge of the variable.
Write Action	◆ Write Single Coil: When the initialization condition is triggered, write the value in the Data column to the slave address 0X. ◆ Write Single Register: When the initialization condition is triggered, write the value in the Data column to the slave address 4X.
Type	Address Types: ◆ 0X: Bit address type, readable and writable. Operates on PLC output points. ◆ 4X: Word address type, readable and writable. Operates on PLC data registers.
Offset	Address Offset.
Length	Data Length, measured in Words.
Data	The data to be written.
Error Code	Communication status corresponding code, please refer to Appendix A - Communication Code Description for details.

7.1.7 Serial Communication Configuration

The MCR7 series PLCs support serial communication with RS232 and RS485 interfaces.

- ◆ When the PLC acts as a slave (slave station), the configuration method is as follows:

Step11. In the **Configuration** interface, select **COM 1** or **COM 2**, set the **Enable** Switch to the “ON” state, set the mode to **Slave**, and configure the relevant information.



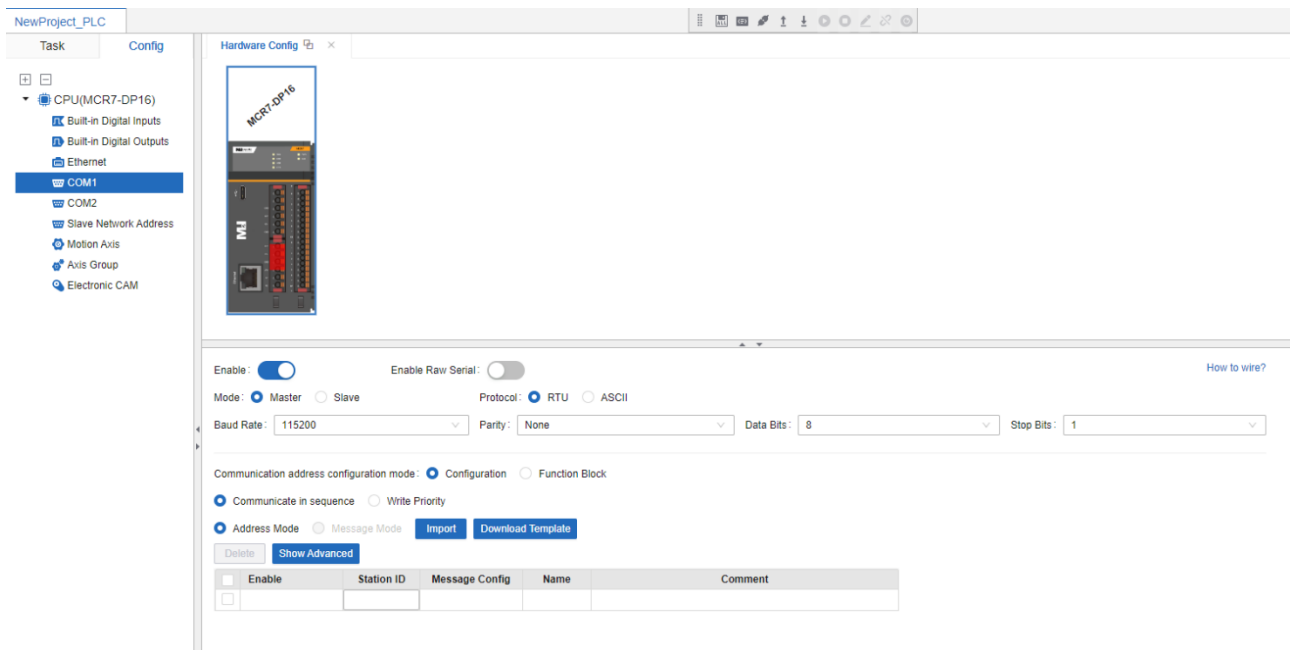
Please refer to the table below for detailed configuration methods.

Parameters	Description
Enable Raw Serial	After enabling raw serial, the PLC serial port can communicate with the other party using a negotiated protocol.
RTU	Use Modbus RTU protocol for communication.
Station ID	Modbus slave number, ranging from 1 to 247.
Baud Rate	Data transfer rate, please refer to the actual situation.
Parity	◆ None: No parity check. ◆ Odd Check: The number of “1” bits in the data plus the check bit is odd. ◆ Even Check: The number of “1” bits in the data plus the check bit is even. ◆ Mark: The check bit is fixed to “1”. ◆ Space: The check bit is fixed to “0”.
Data Bits	The number of bits in each byte that represent actual transmitted data, supporting only 8 bits.
Stop Bits	An ending delimiter for a data transmission, which can be 1 or 2 bits of “1”.

Step12. Click **Modbus Slave Network Address Configuration** to set the Modbus slave network address. For detailed operating instructions, please refer to [Configure Modbus Slave Network Address](#).

◆ When the PLC is set as the master , the configuration method is as follows:

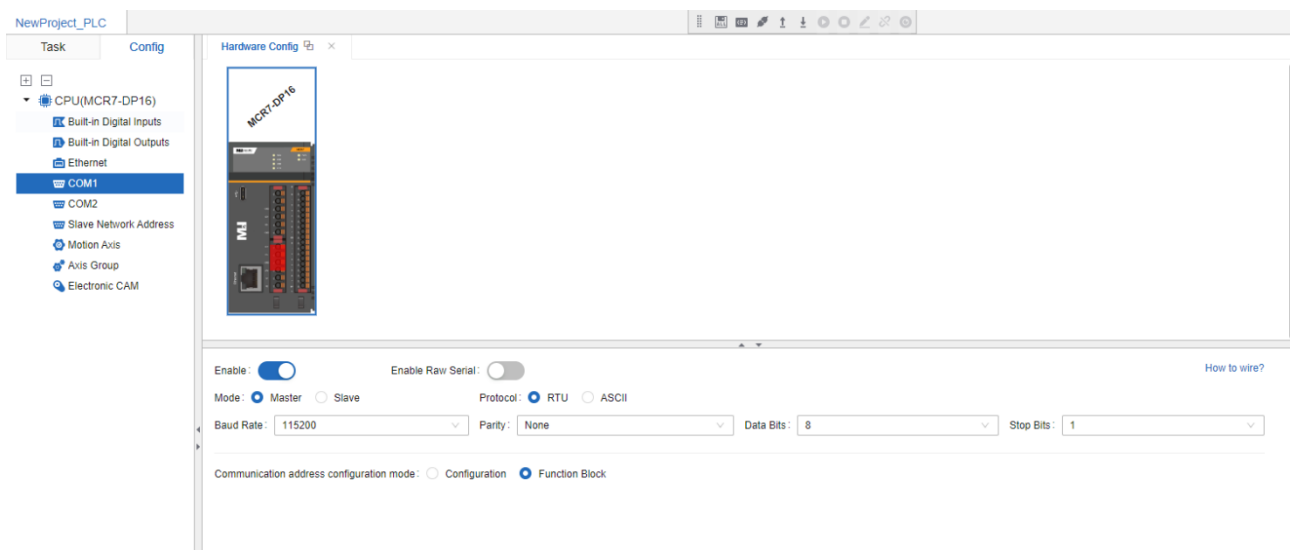
Step13. Select **COM 1** or **COM 2** in the **Configuration** interface, turn on the **Enable** Switch, set the **Mode** to “Master,” and configure the baud rate, parity, data bits, and stop bits.



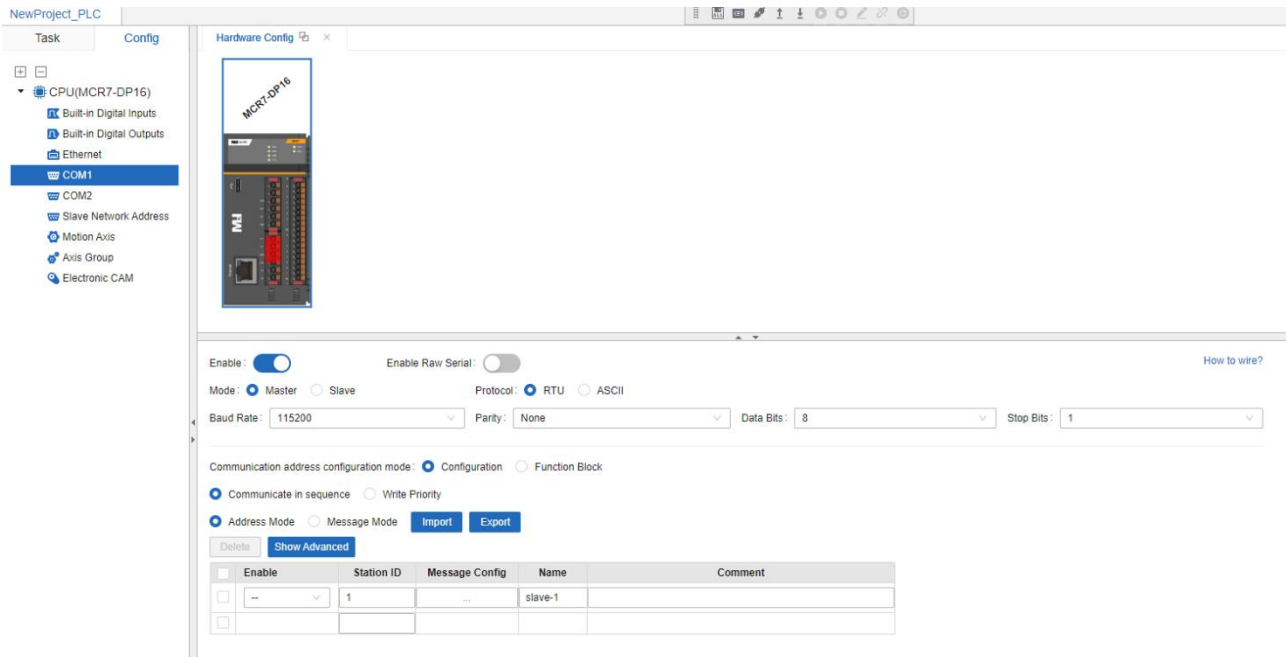
Step14. Configure the communication address.

- Use Function Block

Based on this configuration, in PLC programming (Ladder Diagram and ST programming), use the “Communication” instructions such as ARRAY_ADDR_BOOL (get BOOL array address), ARRAY_ADDR_WORD (get WORD array address), MODBUS_RW_BIT (serial port receive/send bit data instruction), MODBUS_RW_WORD (serial port receive/send word data instruction), MODBUS_WRITE_STRING (write STRING type data over Modbus communication), MODBUS_READ_STRING (read STRING type data over Modbus communication) for Modbus communication.



- Use configuration. The configuration method is similar to the address configuration method of Modbus-TCP-Client. For detailed operations, please refer to [Configure Modbus-TCP-Client](#).

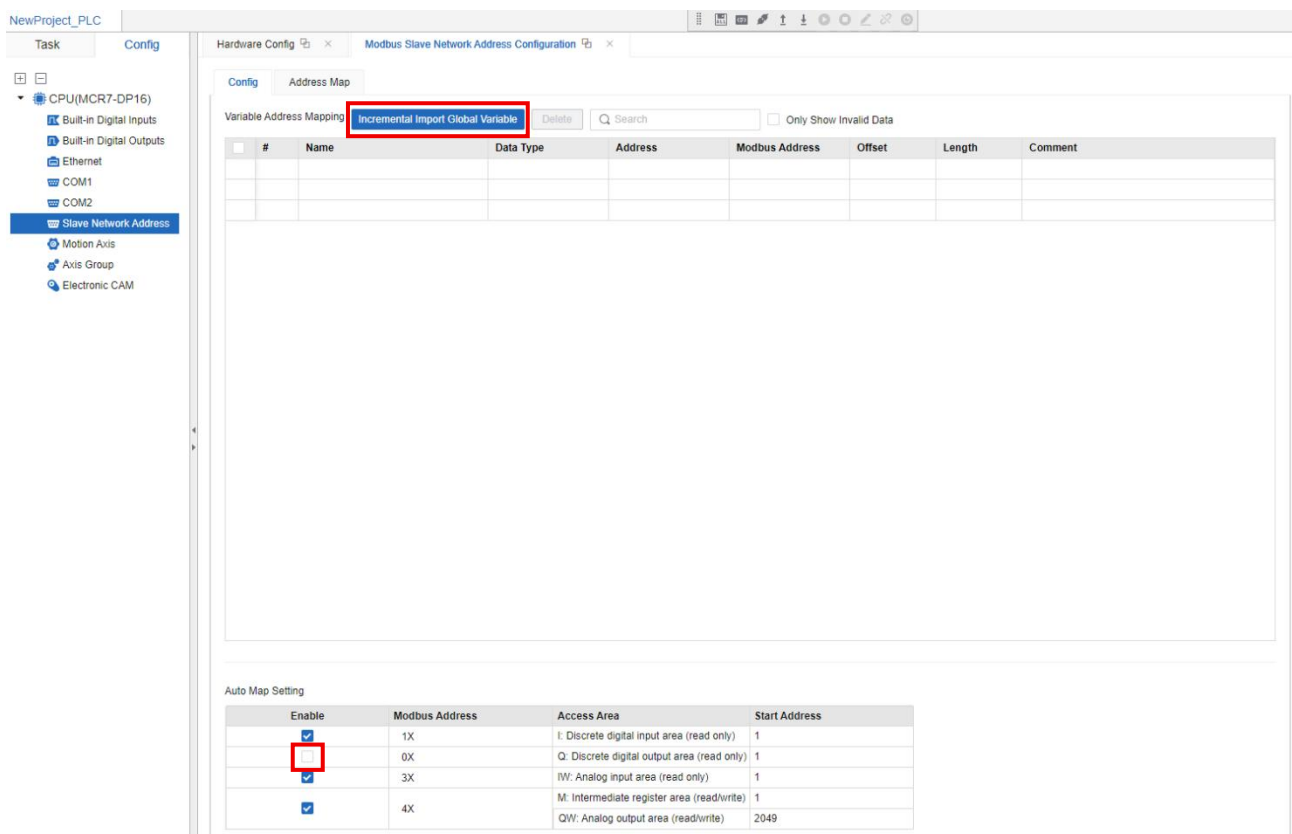


7.1.8 Configure Modbus Slave Network Address

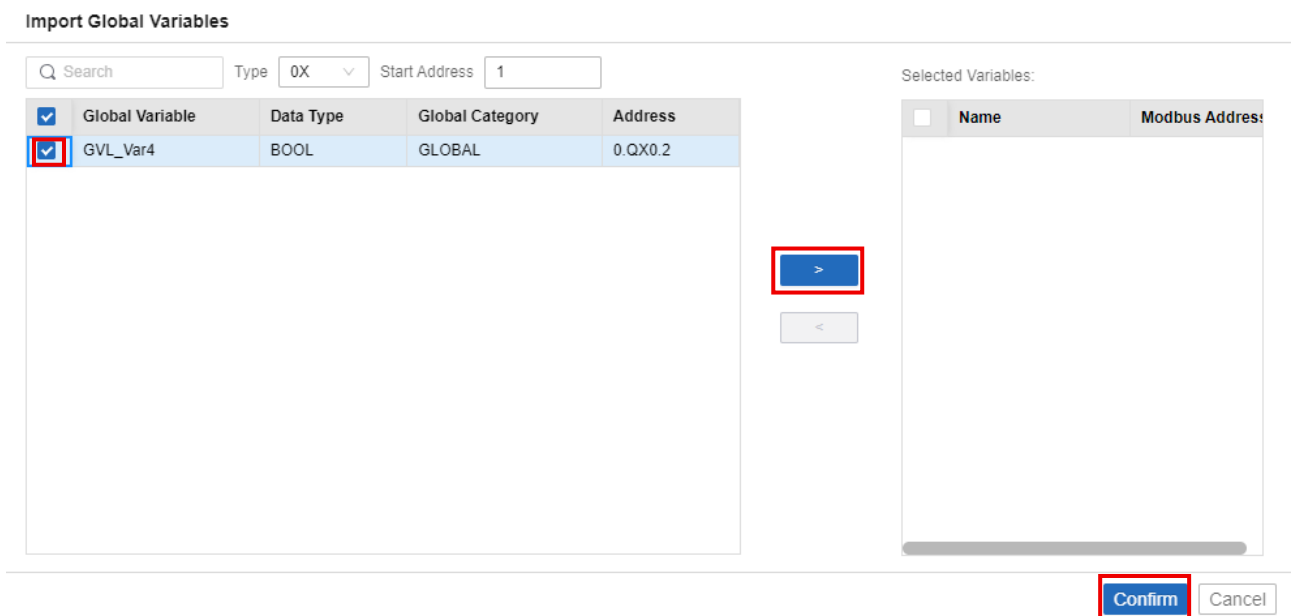
The Modbus slave (PLC as a slave) address information can be configured using two methods: manual setting and automatic mapping.

◆ Manual Setting

Step15. Select **Slave Network Address** in the **Configuration** interface, uncheck the enable option for the corresponding address type in the **Auto Mapping Settings** area, and click **Incremental Import Global Variable**.



Step16. Check the global variables that need to be imported in the pop-up dialog, click the  icon, and then click **Confirm**.



Step17. Set the Modbus address mapped to the global variables.

Config

Address Map

Variable Address Mapping

Incremental Import Global Variable

Delete

Q Search

☐ Only Show Invalid Data

☐	Name	Data Type	Address	Modbus Address	Offset	Length	Comment
☐	GVL_Var4	BOOL		0X1	0	1	
☐							

Auto Map Setting

Enable	Modbus Address	Access Area	Start Address
☒	1X	I: Discrete digital input area (read only)	1
☐	0X	Q: Discrete digital output area (read only)	1
☒	3X	IW: Analog input area (read only)	1
☒	4X	M: Intermediate register area (read/write)	1
		QW: Analog output area (read/write)	2049

◆ Automatic Mapping:

Check the **Auto Mapping** for the corresponding address types in the **auto map setting** area, and it will automatically map the addresses from the corresponding hardware access area to the corresponding Modbus addresses.

For example: If you check “Q”: Discrete Digital Output Area (Readable and Writable)” and “0X,” it will automatically map the address QX0.0 to 0x1.

NewProject_PLC

Task Config

CPU(MCR7-DP16)

- Built-in Digital Inputs
- Built-in Digital Outputs
- Ethernet
- COM1
- COM2
- Slave Network Address
- Motion Axis
- Axis Group
- Electronic CAM

Hardware Config x Modbus Slave Network Address Configuration x

MCR7-DP16

Enable: ☒ Enable Raw Serial: ☐ [How to wire?](#)

Mode: ☒ Master ☐ Slave Protocol: ☒ RTU ☐ ASCII

Baud Rate: 115200 Parity: None Data Bits: 8 Stop Bits: 1

Communication address configuration mode: ☒ Configuration ☐ Function Block

☒ Communicate in sequence ☐ Write Priority

☒ Address Mode ☐ Message Mode [Import](#) [Export](#)

[Delete](#) [Show Advanced](#)

Enable	Station ID	Message Config	Name	Comment
☐	1	...	slave-1	
☐				

NewProject_PLC

Task Config

CPU(MCR7-DP16)

- Built-in Digital Inputs
- Built-in Digital Outputs
- Ethernet
- COM1
- COM2
- Slave Network Address
- Motion Axis
- Axis Group
- Electronic CAM

Hardware Config x Modbus Slave Network Address Configuration x

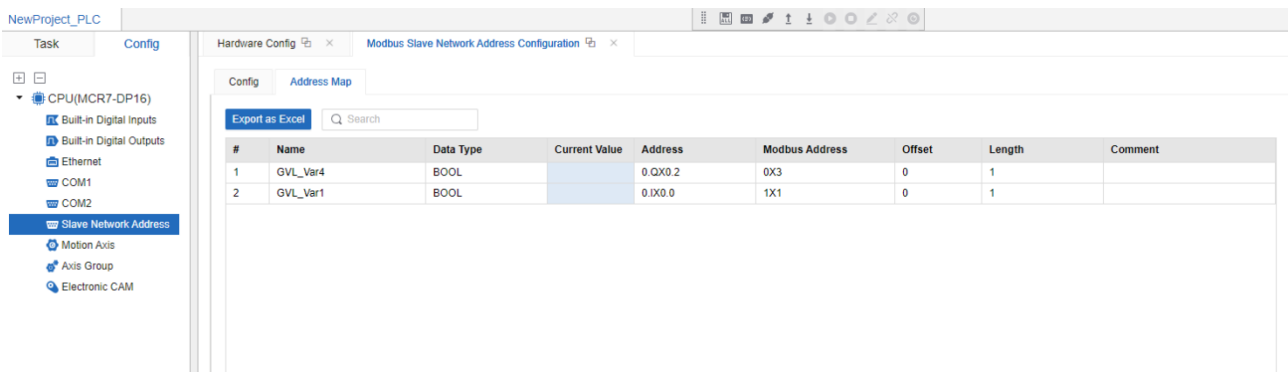
Config Address Map

Variable Address Mapping [Incremental Import Global Variable](#) [Delete](#) ☐ Only Show Invalid Data

#	Name	Data Type	Address	Modbus Address	Offset	Length	Comment

Auto Map Setting

Enable	Modbus Address	Access Area	Start Address
☒	1X	I: Discrete digital input area (read only)	1
☒	0X	Q: Discrete digital output area (read only)	1
☒	3X	IW: Analog input area (read only)	1
☒		M: Intermediate register area (read/write)	1
☒	4X	QW: Analog output area (read/write)	2049



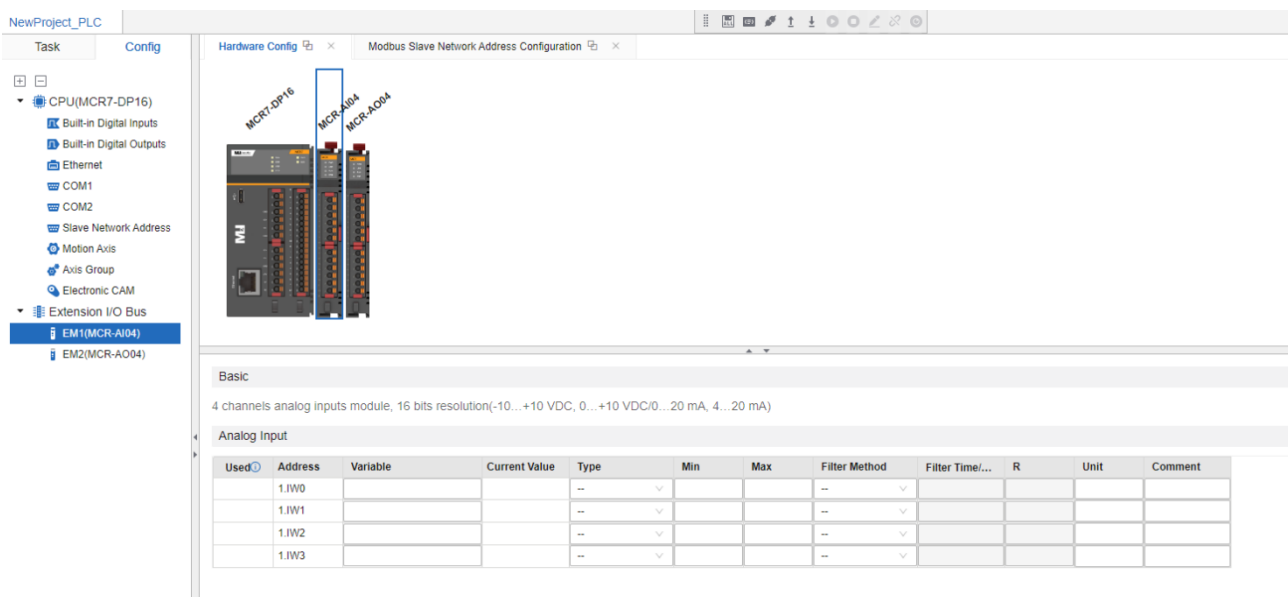
7.2 Configure The Extension I/O Bus

The extended I/O bus includes analog input modules, analog output modules, digital input modules, digital output modules and temperature acquisition modules.

7.2.1 Analog Input

The operation method of adding analog input module is as follows:

Click **Config** on the right side of the software interface to expand the hardware information, drag the analog input module to the hardware diagram, and configure the analog input channel information.



For detailed configuration methods, please refer to the table below.

parameter	Description
Variable	Variable bound to a specified address. The variable name supports Chinese, English letters, numbers and the special character “_”, and cannot start with a number.
Current Value	The current value of the variable.

parameter	Description
Type	Includes voltage and current.
Min	The minimum value of the variable.
Max	The maximum value of the variable.
Filter Method	Filter analog, including low-pass filtering and Kalman filtering.
Filter Time	Low-pass filtering time.
Q	The Kalman filter parameter Q, represents the process noise variance.
Unit	The unit of the variable, such as mA , mV , etc.
Comment	Description of the variable.

7.2.2 Analog Output

The operation method of adding analog output module is as follows:

Click **Config** on the right side of the software interface to expand the hardware information, drag the analog output module to the hardware diagram, and configure the analog output channel information.

Used	Address	Variable	Current Value	Type	Min	Max	Excepti...	Unit	Comment
	1.QW0			--					
	1.QW1			--					
	1.QW2			--					
	1.QW3			--					

For detailed configuration methods, please refer to the table below.

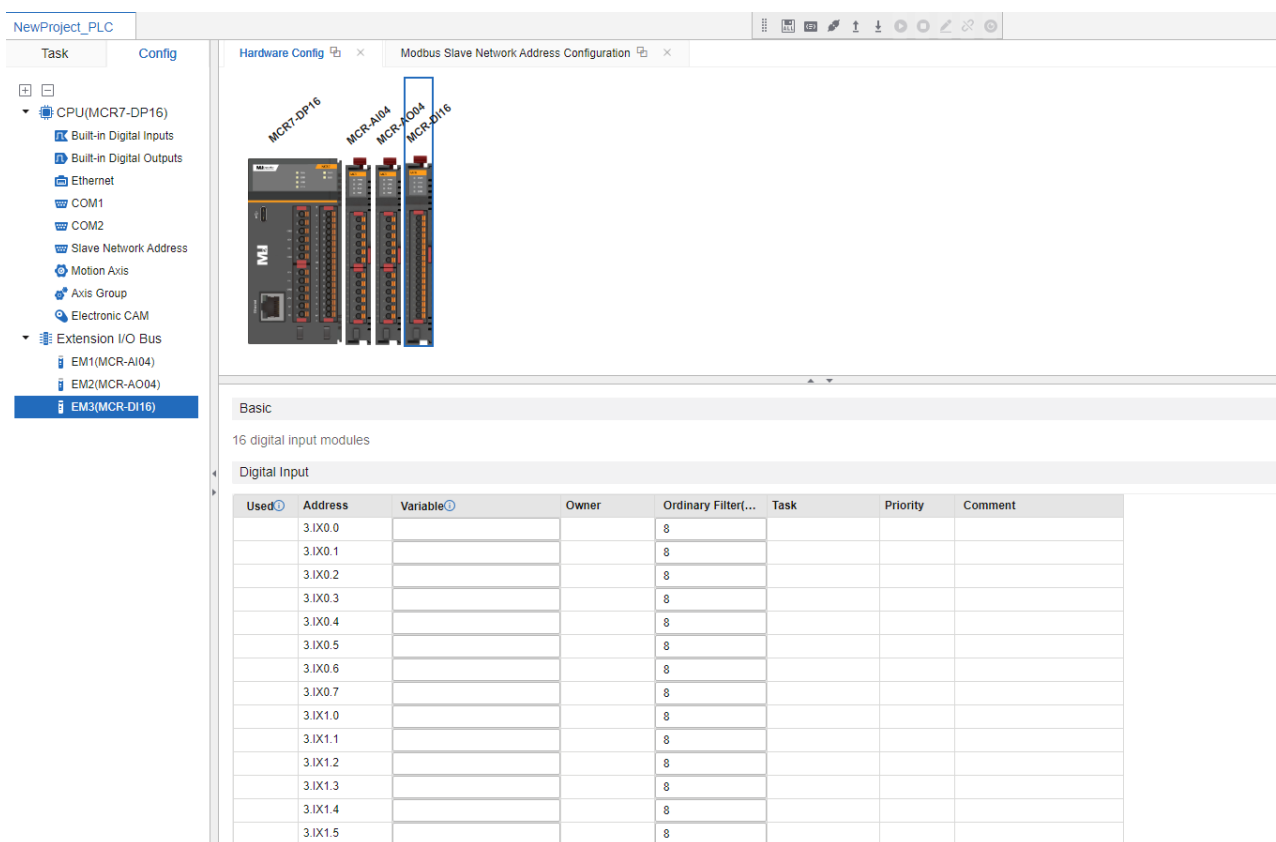
Parameter	Description
Variable	Variable bound to a specified address. The variable name supports Chinese, English letters, numbers and the special character “_”, and cannot start with a number.

Parameter	Description
Current Value	The current value of the variable.
Type	Includes current and voltage.
Min	The minimum value of the variable.
Max	The maximum value of the variable.
Exceptional Default	Analog output value corresponding to fault status.
Unit	The unit of the variable, such as mA , mV , etc.
Comment	Description of the variable.

7.2.3 Digital Input

The operation method of adding digital input module is as follows:

Click **Config** on the right side of the software interface to expand the hardware information, drag the digital input module to the hardware diagram, and configure the digital input channel information.



The screenshot shows the software interface for configuring hardware. The left sidebar lists various hardware components, including CPU(MCR7-DP16), Built-in Digital Inputs, Built-in Digital Outputs, Ethernet, COM1, COM2, Slave Network Address, Motion Axis, Axis Group, Electronic CAM, and Extension I/O Bus. The main area displays a hardware diagram with three modules: MCR7-DP16, MCR-A04, and MCR-DI16. The bottom panel shows the 'Basic' configuration for the digital input modules, indicating 16 digital input modules are available. A table lists the configuration for each module, including the address, variable, owner, ordinary filter, task, priority, and comment.

Used	Address	Variable	Owner	Ordinary Filter(...)	Task	Priority	Comment
	3.IX0.0			8			
	3.IX0.1			8			
	3.IX0.2			8			
	3.IX0.3			8			
	3.IX0.4			8			
	3.IX0.5			8			
	3.IX0.6			8			
	3.IX0.7			8			
	3.IX1.0			8			
	3.IX1.1			8			
	3.IX1.2			8			
	3.IX1.3			8			
	3.IX1.4			8			
	3.IX1.5			8			

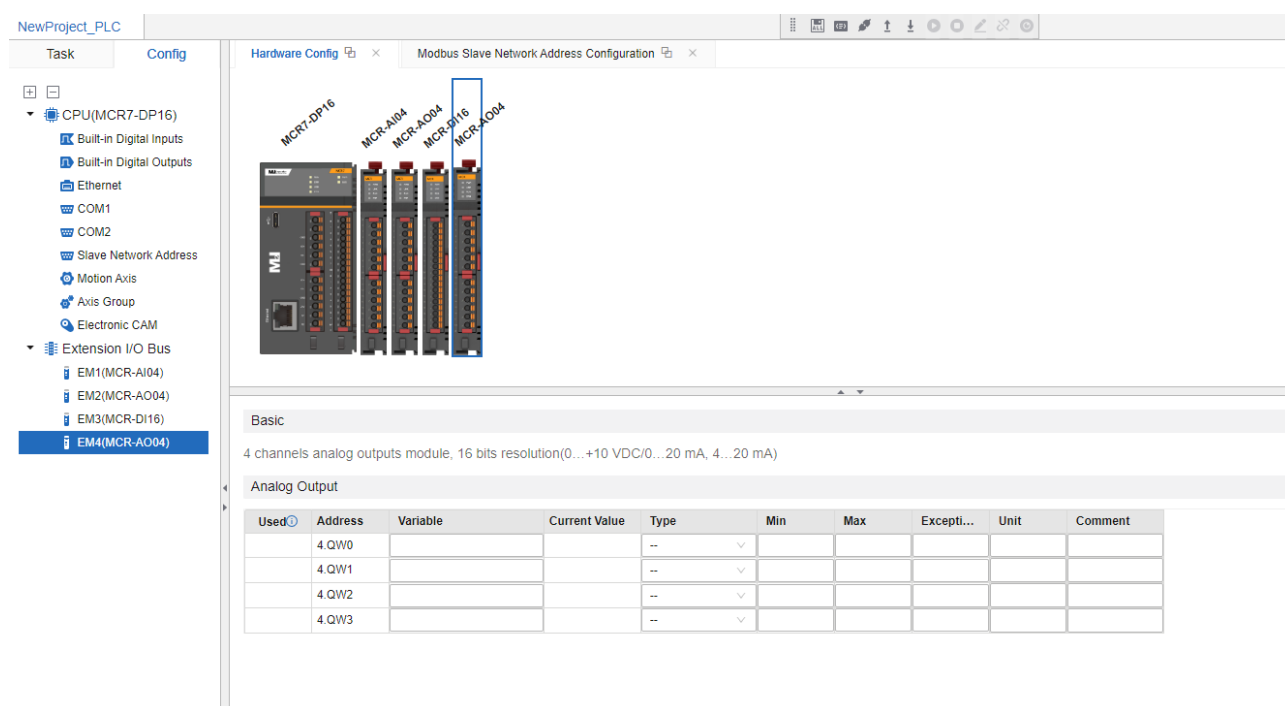
For detailed configuration methods, please see the table below.

parameter	Description
Variable	Variable bound to a specified address. The variable name supports Chines, English letters, numbers and the special character “_”, and cannot start with a number.
Owner	Indicates where this variable is used. It can be the name of the global variable table or the name of the motion axis. It is empty when using local variables.
Latch	Refers to locking the input signal in the current logic state (such as high or low level) and storing it in the current state until the next time it is refreshed or changed.
Task	This variable is called when executing the task.
Priority	The priority of the task being called.
Comment	Description of the variable.

7.2.4 Digital Output

The operation method of adding digital output module is as follows:

Click **Config** on the right side of the software interface to expand the hardware information, drag the digital output module to the hardware diagram, and configure the variables.

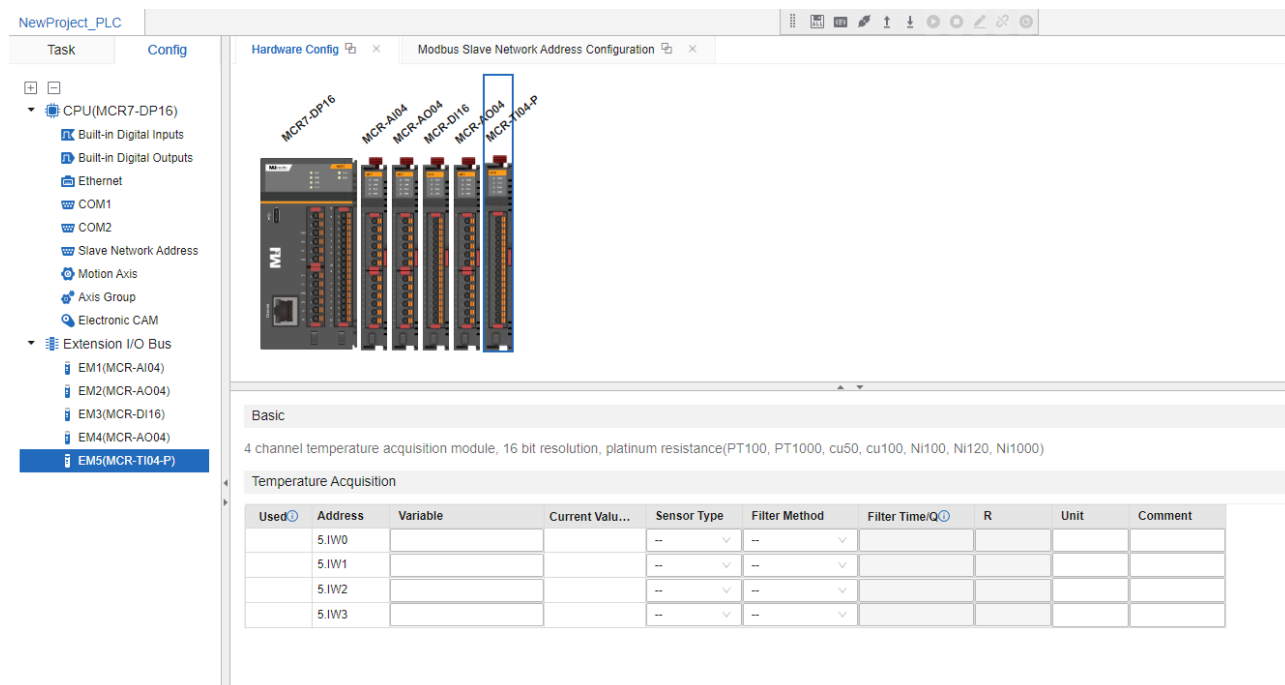


7.2.5 Temperature Acquisition Module

The temperature acquisition module collects temperature information through built-in sensors. The operation method of adding temperature acquisition module is as follows:

◆ Steps of adding P-type temperature acquisition module (FR-T0400P) are as follows:

Click **Config** on the right side of the software interface to expand the hardware information, drag the temperature acquisition module (FR-T0400P) to the hardware diagram, and configure the temperature acquisition channel information.



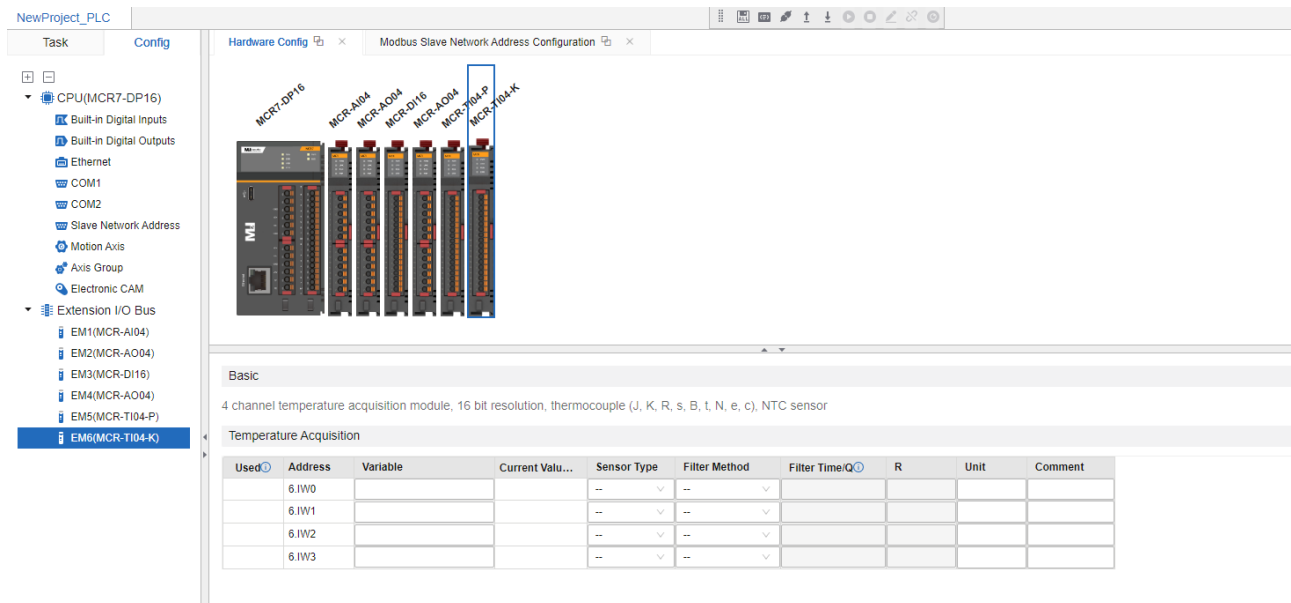
For detailed configuration methods, please refer to the table below.

Parameter	Description
Variable	Variable bound to a specified address. The variable name supports Chines, English letters, numbers and the special character “_”, and cannot start with a number.
Current Value	The current value of the variable.
Sensor Type	Classification based on sensor material: ◆ PT100 ◆ PT1000 ◆ Cu 50 ◆ Cu 100 ◆ Ni 1000
Filter Method	Including low-pass filtering and Kalman filtering.
Filter Time	Low-pass filtering time.
Q	The parameter Q of the Kalman filter represents the process noise variance.

◆ Steps of adding K- type temperature acquisition module (FR-T0440K) are as follows:

Click **Config** on the right side of the software interface to expand the hardware information, drag the K-type

temperature acquisition module (FR-T0440K) to the hardware diagram, and configure the temperature acquisition channel information.



For detailed configuration methods, please refer to the table below.

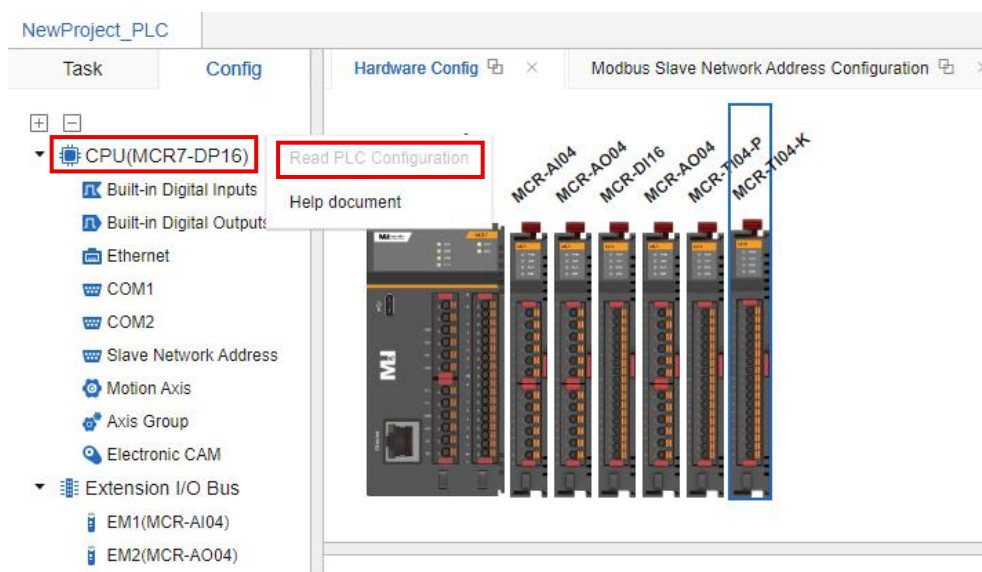
Parameter	Description
Variable	Variable bound to a specified address. The variable name supports Chinese, English letters, numbers and the special character “_”, and cannot start with a number.
Current Value	The current value of the variable.
Sensor type	◆ Type R thermocouple: The nominal chemical composition of the positive electrode (RP) is platinum-rhodium alloy, containing 13% rhodium and 87% platinum. The negative electrode (RN) is pure platinum. ◆ Type S thermocouple: The nominal chemical composition of the positive electrode (SP) is platinum-rhodium alloy, which contains 10% rhodium and 90% platinum. The negative electrode (SN) is pure platinum. ◆ Type B thermocouple: The nominal chemical composition of the positive electrode (BP) is platinum-rhodium alloy, which contains 30% rhodium and 70% platinum. The negative electrode (BN) is a platinum-rhodium alloy containing 6% rhodium. ◆ Type T thermocouple: the positive electrode (TP) is pure copper, and the negative electrode (TN) is copper-nickel alloy. ◆ Type N thermocouple: The nominal chemical composition of the positive electrode (NP) is: Ni:Cr:Si =84.4:14.2:1.4. The nominal chemical composition of the negative electrode (NN) is: Ni:Si:Mg =95.5:4.4:0.1. ◆ Type E thermocouple: The positive electrode (EP) is: nickel-chromium 10 alloy, the chemical composition is the same as KP. The negative electrode (EN) is copper-nickel alloy. ◆ Type C thermocouple: positive electrode material: Tungsten-5% Rhenium. Negative material: Tungsten-26% Rhenium.

Parameter	Description
	◆ NTC: NTC temperature sensor is a thermistor and probe. Its principle is: the resistance value decreases rapidly as the temperature rises. It usually consists of 2 or 3 metal oxides.
Filter Method	Including low-pass filtering and Kalman filtering.
Filter Time	Low-pass filtering time.
Q	The Kalman filter parameter Q represents the process noise variance.
R	The Kalman filter parameter R represents the measurement noise variance.
Unit	Fixed at 0.1 °C.
Comment	Description of the variable.

7.3 Read The Hardware Configuration of PLC

MCR Master supports one-click identification of the hardware configuration of the currently connected PLC. The operation method is as follows:

Step18. Right-click on the CPU and select **Read PLC Configuration** .



Step19. You can view the hardware configuration information of current project and the hardware configuration information of the actual connected PLC .

Current Project Hardware Configuration



Current project hardware configuration is inconsistent with the actual PLC hardware configuration!

[Update Project With Actual PLC Configuration](#)

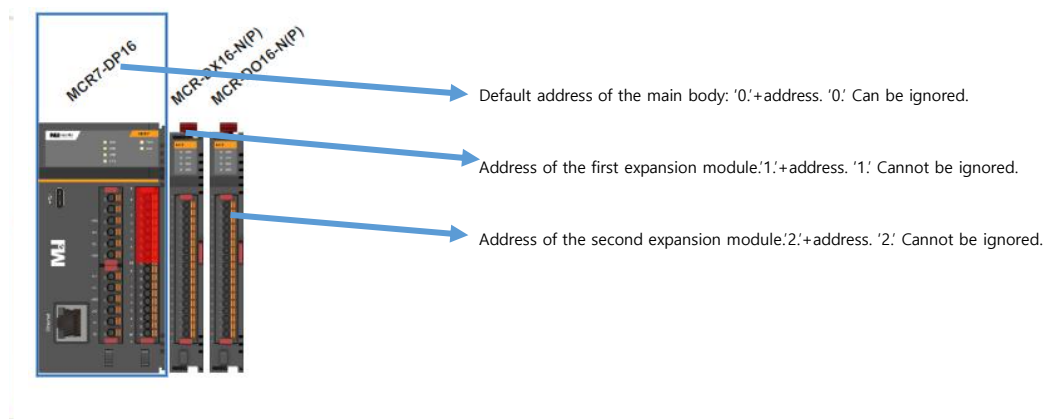
Actual PLC Hardware Configuration



Step20. If it is detected that the current project configuration is inconsistent with the actual PLC hardware configuration, MCR Master will give a prompt message, click to **update project with Actual PLC configuration**, and update the hardware configuration information of the project using the hardware configuration of the actual connected PLC .

7.4 Hardware address of PLC

PLC has default hardware address format and allocation rules. When calling the address on the newly added expansion module, you need to refer to the actual address allocation information in the hardware configuration interface to ensure that the calling address is accurate and can be compiled successfully.



the CPU body is as shown in the figure below.

NewProject_PLC

Task Config Hardware Config Modbus Slave Network Address Configuration

+ -
 CPU(MCR7-DP16)
 Built-in Digital Inputs
 Built-in Digital Outputs
 Ethernet
 COM1
 COM2
 Slave Network Address
 Motion Axis
 Axis Group
 Electronic CAM
 Extension I/O Bus
 EM1(MCR-DX16-N(P))
 EM2(MCR-DO16-N(P))

MCR7-DP16
 MCR-DX16-N(P)
 MCR-DO16-N(P)

Used	Address	Variable	Owner	Ordinary Filter(...)	Hardware Filter	Task	Priority	Comment
	0.IX0.0	GVL_Var1	GLOBAL	8	1440ns			
	0.IX0.1			8	1440ns			
	0.IX0.2			8	1440ns			
	0.IX0.3			8	1440ns			
	0.IX0.4			8	1440ns			
	0.IX0.5			8	1440ns			
	0.IX0.6			8	1440ns			
	0.IX0.7			8	1440ns			

The address format of the Digital Expansion Module is shown below.

NewProject_PLC

Task Config

Hardware Config Modbus Slave Network Address Configuration Motion Axis Axis Group

CPU(MCR7-DP16)
 Built-in Digital Inputs
 Built-in Digital Outputs
 Ethernet
 COM1
 COM2
 Slave Network Address
 Motion Axis
 Axis Group
 Electronic CAM
 Extension I/O Bus
 EM1(MCR-DX16-N(P))
 EM2(MCR-DO16-N(P))

MCR7-DP16
 MCR-DX16-N(P)
 MCR-DO16-N(P)

Basic

8-point bipolar input, with 8-point NPN(PNP) output modules, 0.5A/Point

Digital Input

Used	Address	Variable	Owner	Ordinary Filter(...)	Task	Priority	Comment
	1.IX0.0			8			
	1.IX0.1			8			
	1.IX0.2			8			
	1.IX0.3			8			
	1.IX0.4			8			
	1.IX0.5			8			
	1.IX0.6			8			
	1.IX0.7			8			

Digital Output

Used	Address	Variable	Owner	Comment
	1.QX0.0			
	1.QX0.1			
	1.QX0.2			
	1.QX0.3			
	1.QX0.4			
	1.QX0.5			

The address format of the Analog Expansion Module is shown below.



“.” cannot be added after the IW/QW type address.

Hardware Config

Modbus Slave Network Address Configuration

Motion Axis

Axis Group





Basic

2 channels analog inputs module, with 2 channels analog outputs module, 16 bits resolution(input: -10...+10 VDC, 0...+10 VDC /0...20 mA, 4...20 mA, output: 0...+10 VDC /0...20 mA, 4...20 mA)

Analog Input

Used	Address	Variable	Current Value	Type	Min	Max	Filter Method	Filter Time/...	R	Unit	Comment
	2.IW0			--			--				
	2.IW1			--			--				

Analog Output

Used	Address	Variable	Current Value	Type	Min	Max	Excepti...	Unit	Comment
	2.QW0			--					
	2.QW1			--					

For more information about hardware address types, refer to the table below.

Modbus address type	Hardware access area	initial address
0X	I: Discrete digital input area (read only)	1
1X	Q: Discrete digital output area (read and write)	1
3X	IW: Analog input area (read only)	1
4X	M: Intermediate register area (read and write)	1
	QW: Analog output area (read and write)	513 (changes depending on the memory allocated in the M area)

8 Task

Task is the sequence in which PLC executes programs. A task can contain multiple programs, and tasks are also divided into circular and event types.

MCR Master provides POU view and Machine view, to display task information.

8.1 POU View

POU view is to display different program organization unit (POU) from the perspective of the program.

8.1.1 Program Organizational Unit

Program Organization Unit (POU) is the basic unit of PLC program, including variable declaration area and code area.

- ◆ The variable declaration area is used to specify the name, type, and initial value of the variable. The variable declaration area is used to declare POU variables and data types. All variables that will be used in this POU need to be declared in the declaration area of the POU. These variables include: input variables, output variables, input / output variables, local variables and constants. In the Ladder Diagram editing (LD) mode, declarations are made directly in the table. In the Structured Text language (ST) editing mode, the declaration format is based on the IEC 61131-3 standard. The declaration of variables adopts the following format:

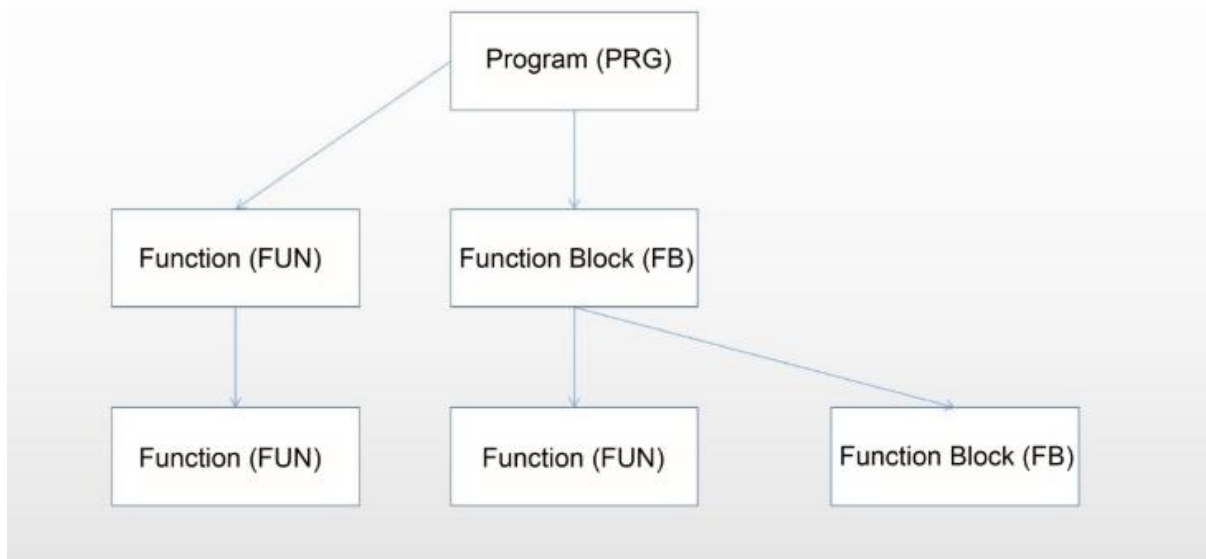
```
<identifier>{AT<Address>}.<data type>{:=<initialization>}; // The part in {} is optional.
```

- ◆ The code area is program code written in the five programming languages recommended by IEC 61131-3. MCR Master supports only two programming languages: Ladder Diagram (LD) and Structured Text (ST).

Program Organizational Unit includes programs, function blocks, and functions.

8.1.1.1 Program

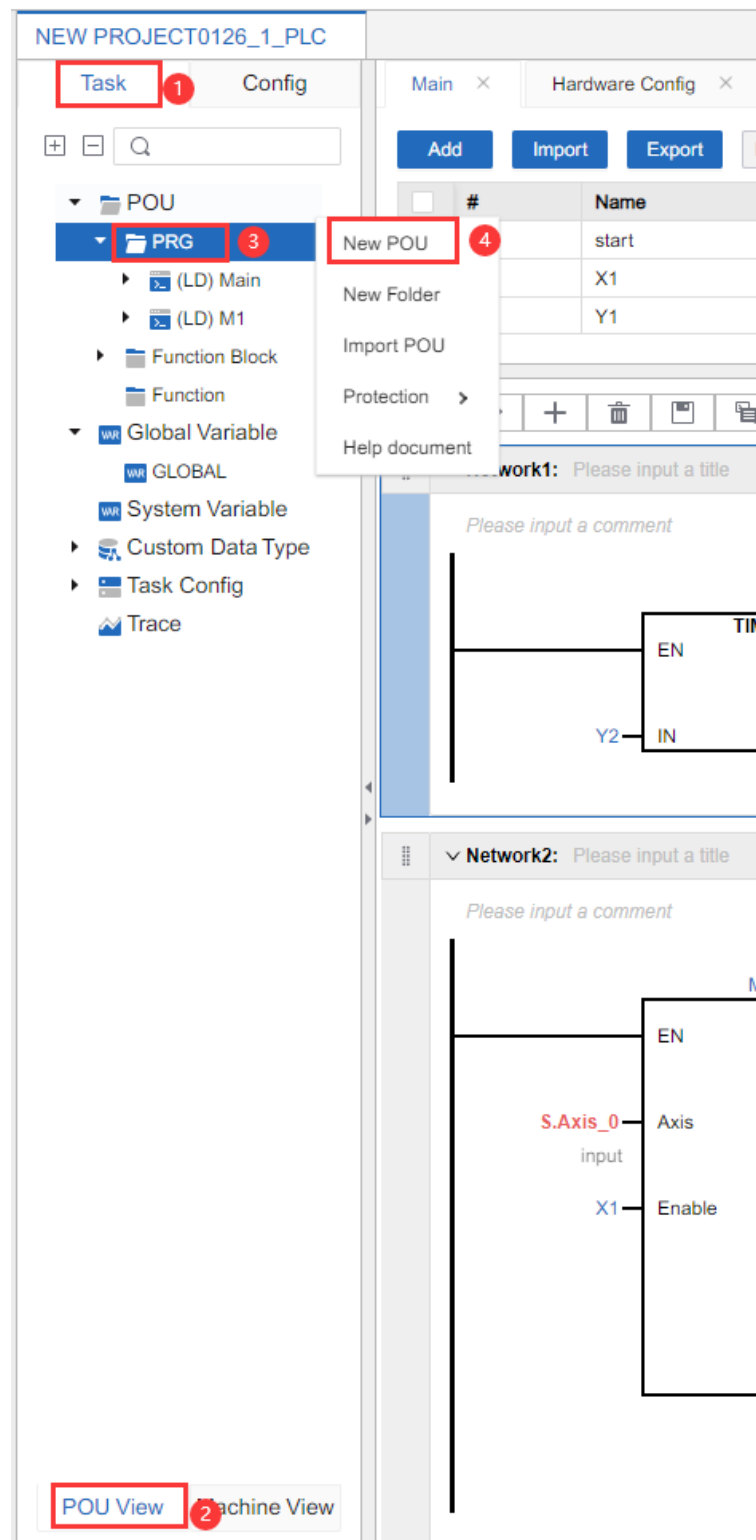
Program is the main core of planning a task. It has the greatest calling rights to call function blocks and functions. In a broad sense, a program also includes hardware configuration, task configuration, and communication configuration. Common global variables, mapped hardware address global variables, and local variables are usually defined in the program. Application logic is implemented via program calls. Function block instances and functions can be called by programs, programs are considered global, are the largest form in the program organization unit. Similarly, function blocks are allowed to call other function blocks and functions, but functions can only call other functions. The program call relationship is shown in the following figure.



8.1.1.1.1 Create a New Program

MCR Master has the Main program by default . The operation method of creating a new program is as follows:

Step21. Select **POU View** in the task area , right-click **PRG**, and select **New POU** in the pop-up menu .



Step22. Configure the relevant parameters in the pop-up dialog box and click **Confirm** to save the configuration.

New POU

* Name:

sum

* Type:

☒ PRG
☐ Function
☐ Function Block

* Language:

☒ LD
☐ ST

Desc:

Cancel

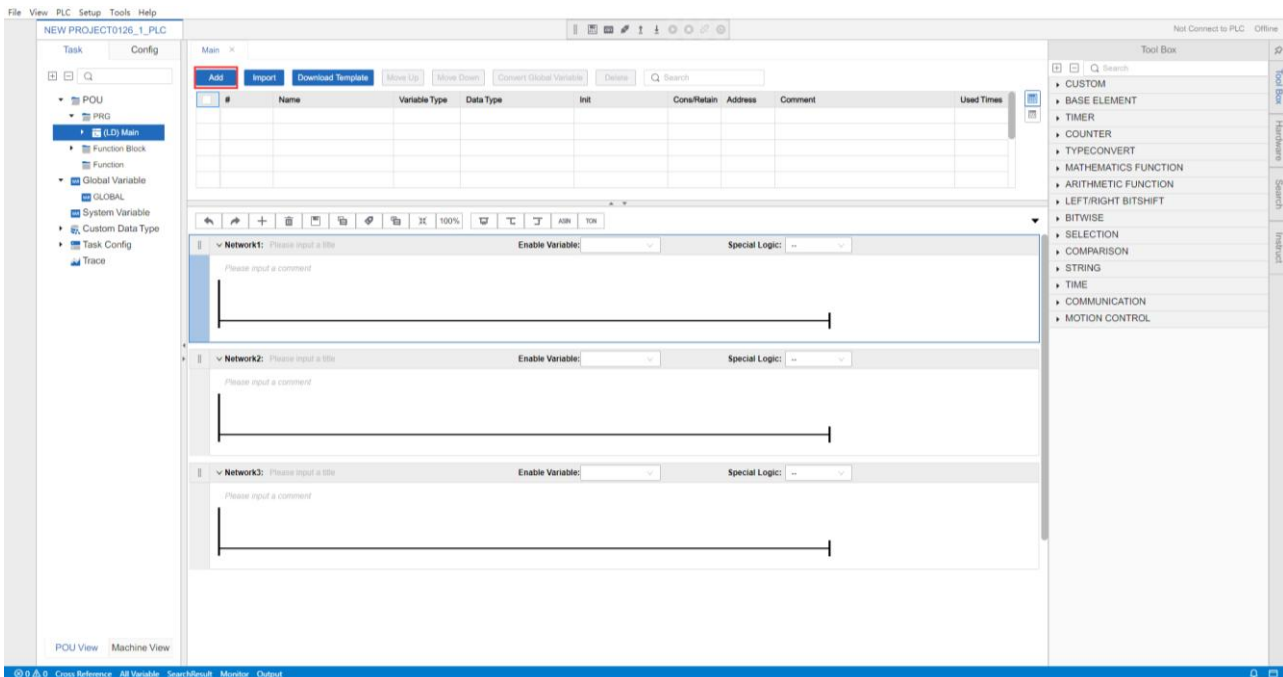
Confirm

Please refer to the table below for detailed configuration methods.

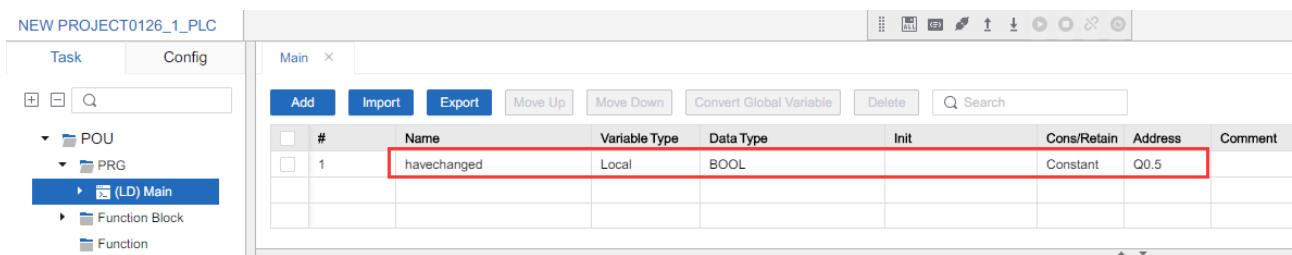
parameter	Description
Name	The program name only supports English letters, numbers and the special character “_”, and must start with an English letter.
Type	Select “Program”.
Language	Supports LD (Ladder Diagram) or ST (Structured Text).
Description	Description of the program.

Step23. Declare variables.

3) Click Add.



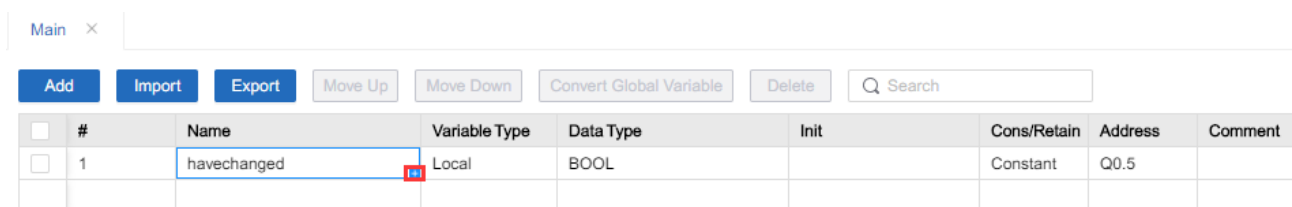
4) Configure variables.



Please refer to the table below for detailed configuration methods.

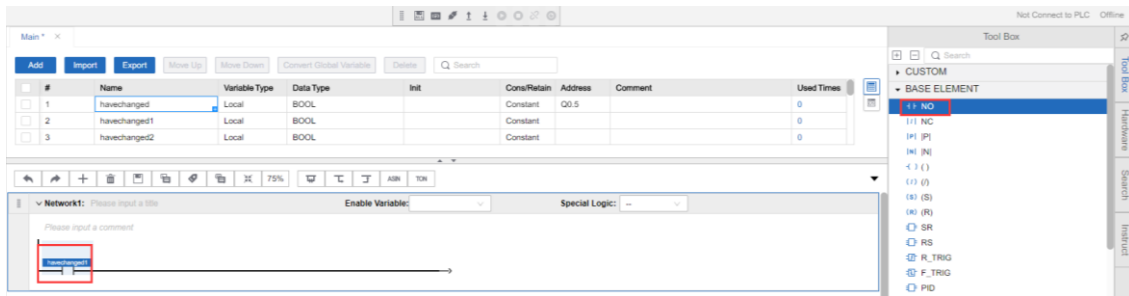
Parameter	Description
Name	The variable name supports Chinese, English letters, numbers and the special character “_”, and cannot start with a number.
Variable Type	◆ Local variables: Variables declared within a specific scope in a program. They are also called internal variables, which are defined and accessed inside a function or compound statement and can only be used within the declared POU. ◆ Temporary variable: As a kind of instrumental variable, it stores intermediate results or is used to exchange the values of two variables, etc. The scope of temporary variables is usually limited to the block of code in which they are declared; once outside of that block, the variables are destroyed. Therefore, temporary variables are temporary and local.
Data Type	Please refer to the actual situation.
Init	Initial value of variables.
Constant/ Retain	Constant: The variable is a constant. Retain: After forcing variables in online monitoring state, check Retain the force state and value of variables when switch to offline state, and the variables will retain the forced results.
Address	The address to which the variable is bound.

- 5) Select the cell where the variable name is located and an refresh icon  will appear. Hover the cursor to the refresh icon  until a black cross appears. Hold down the left mouse button and scroll down to create variables in batches.

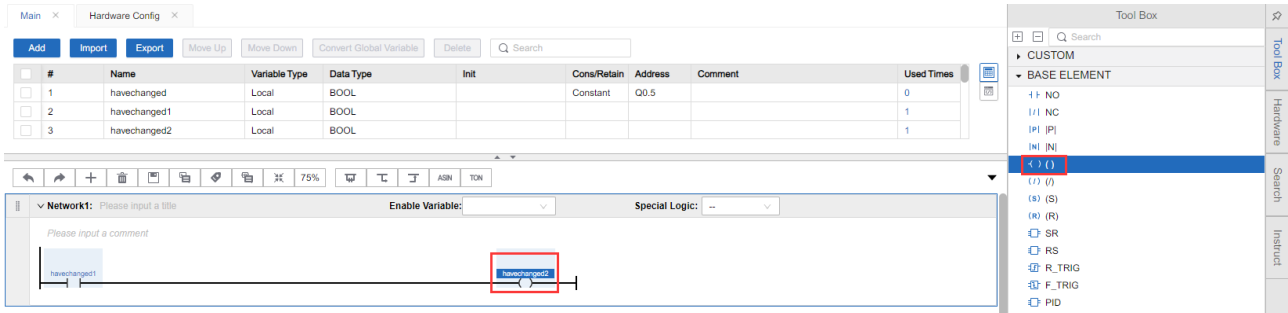


Step24. Edit program (takes Ladder Diagram language as an example).

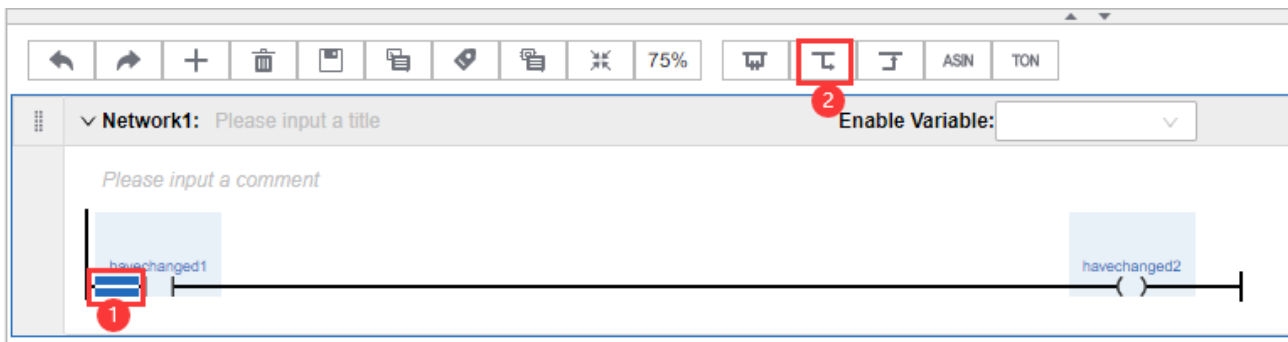
- 6) Click **Tool Box** on the right side of the program interface to display the calculation component information, drag the component (e.g. normally open contact) to the connection line in the program segment, and bind variables.



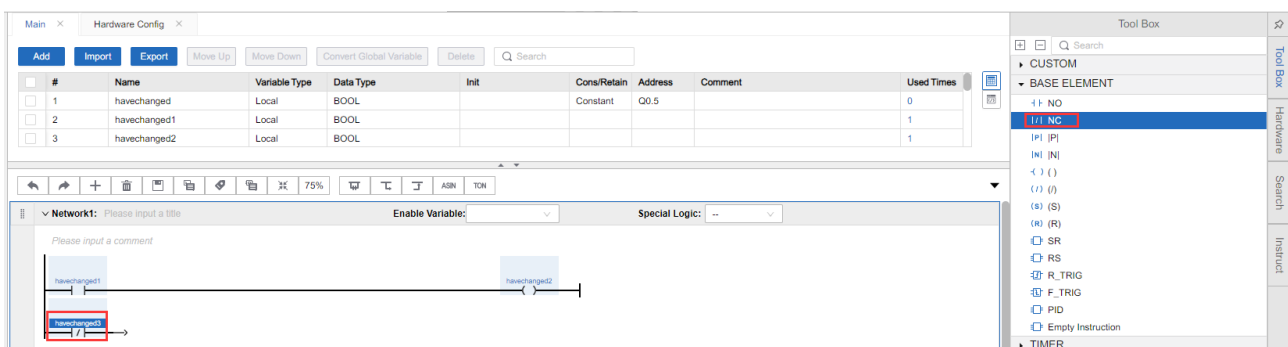
7) Drag the coil to the right side of the program segment and bind the variable.



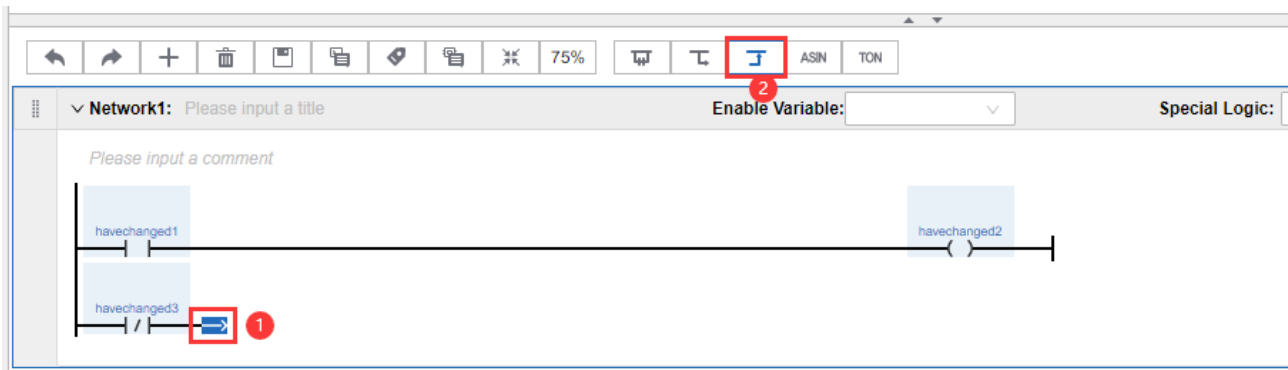
8) Click the location on the branch and click the  icon to create a new branch.



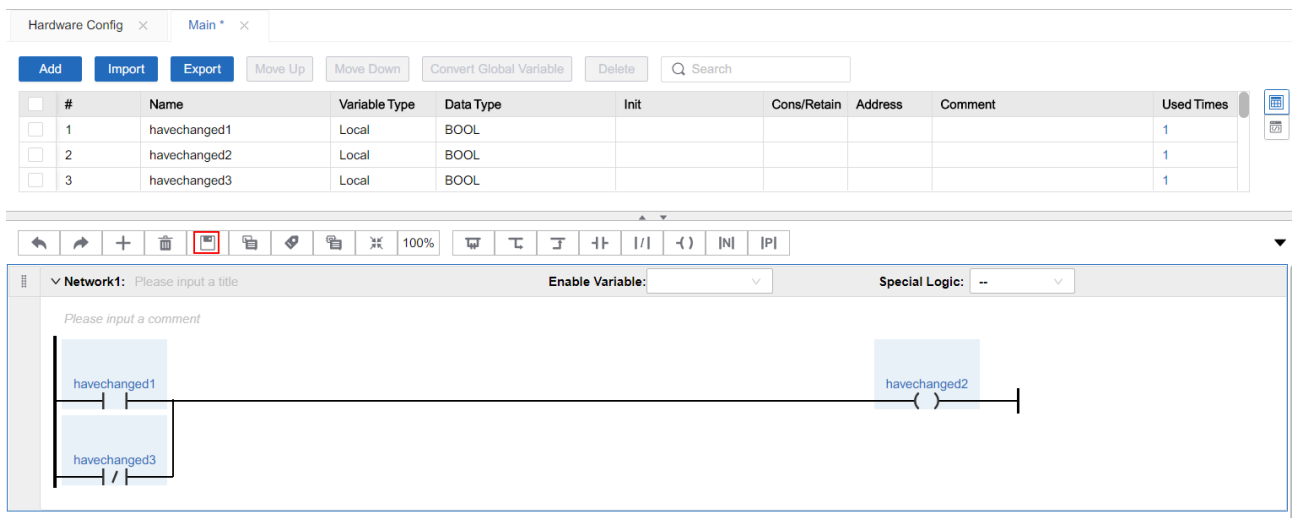
9) Drag components (e.g. normally closed contacts) to the branch, add components on the branch, and bind variables.



10) Click the branch arrow and click the  icon to merge branches.

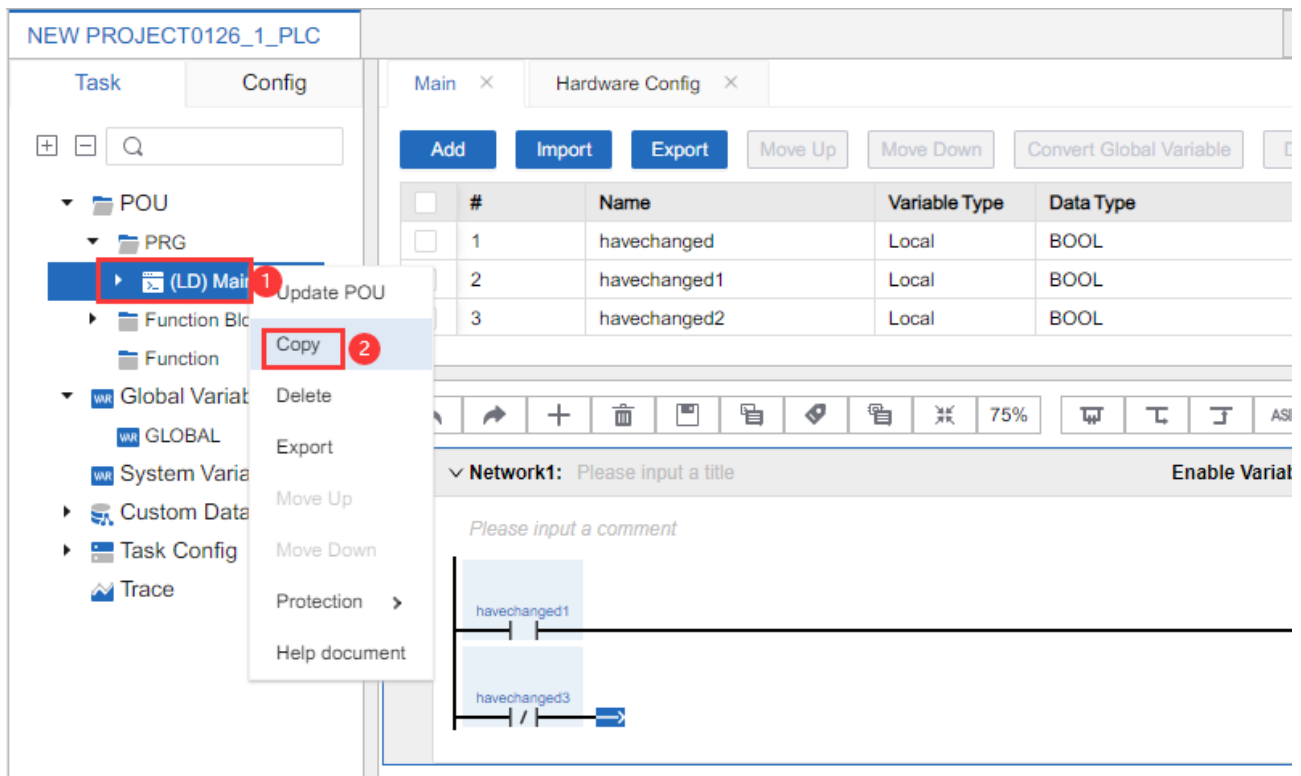


Step25. After the program design is completed, click the  icon to save the program.



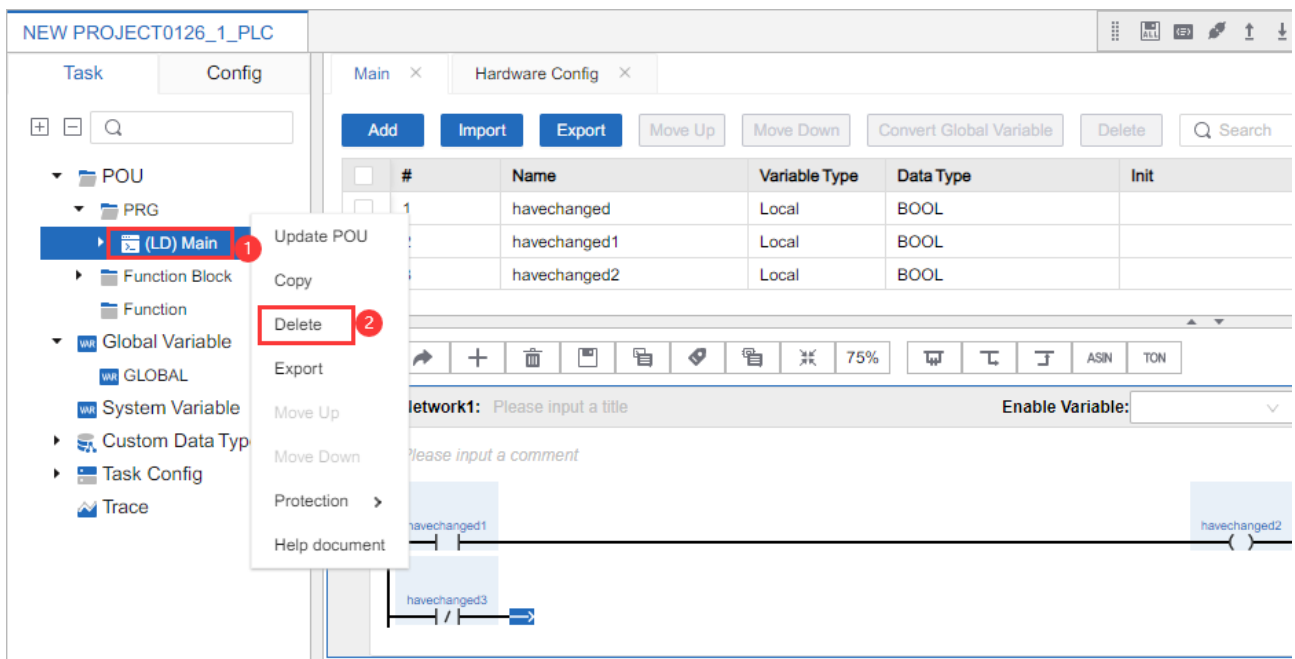
8.1.1.1.2 Copy Program

Right-click on the program and select **Copy** in the pop-up menu to copy the program. Modifications can be made to the original program and the original program can be retained.



8.1.1.1.3 Delete Program

Right-click on the program name, select **Delete** in the pop-up menu, and click **Confirm** in the pop-up dialog box to delete the program.



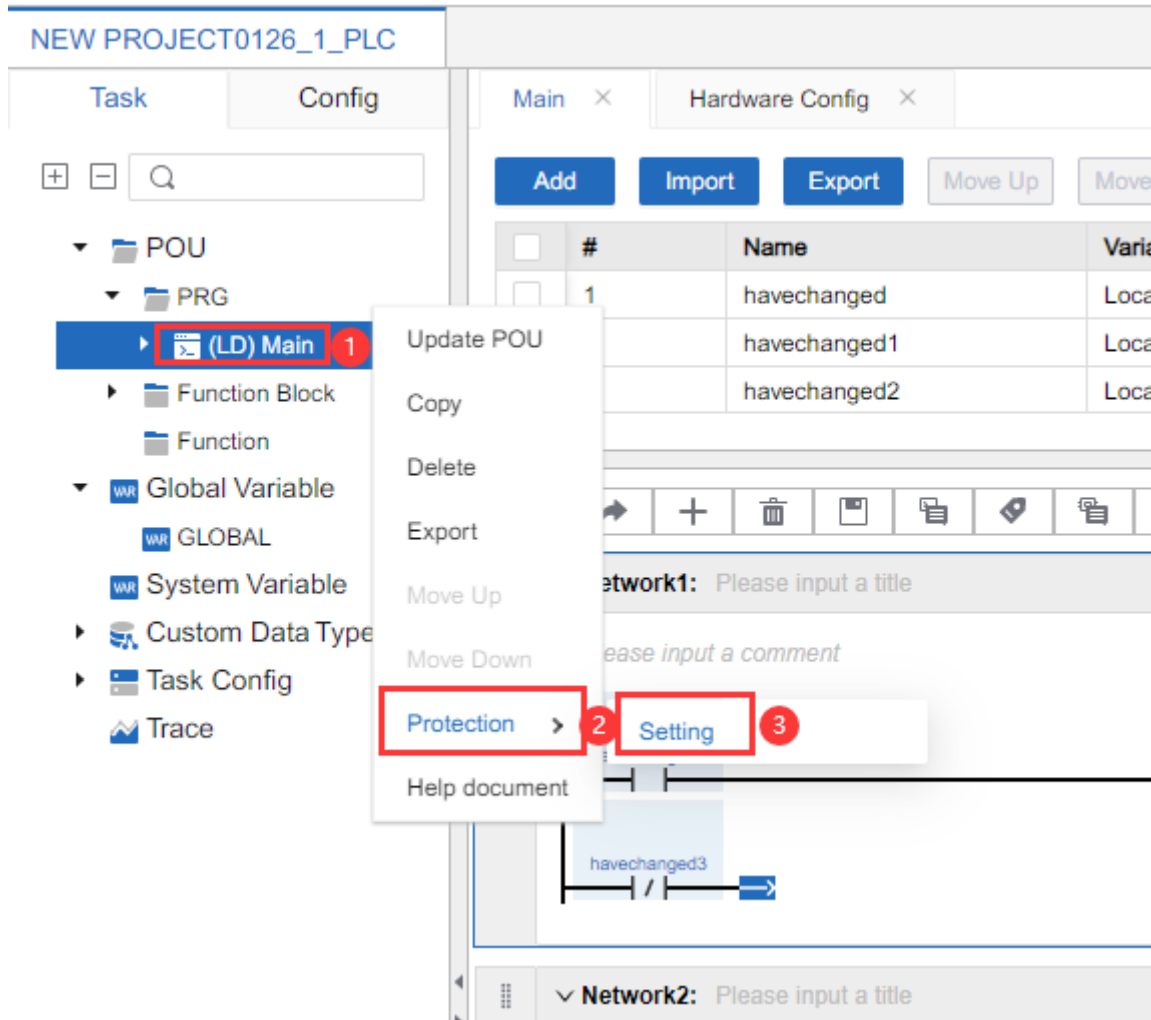
8.1.1.1.4 Set Program Password

After setting a password for the program, you need to enter the password for verification when viewing and modifying the program. It can ensure the privacy and security of the program and prevent unauthorized users from

stealing and modifying the program.

Steps to set the program password are as follows:

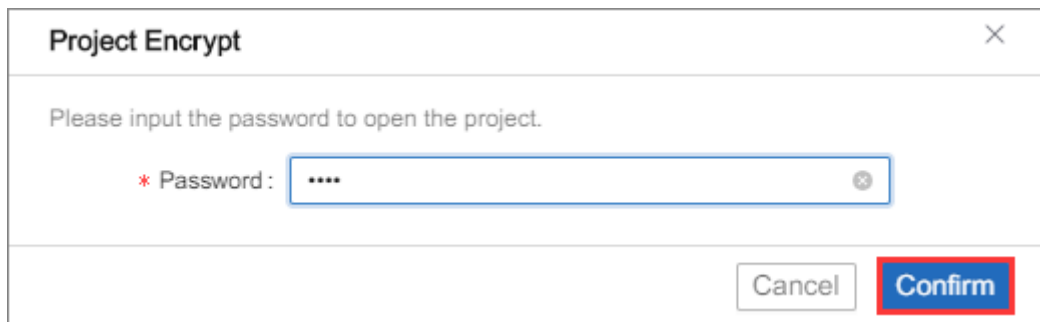
Step26. Right-click on the program name and select **Protection/Setting** from the pop-up menu.



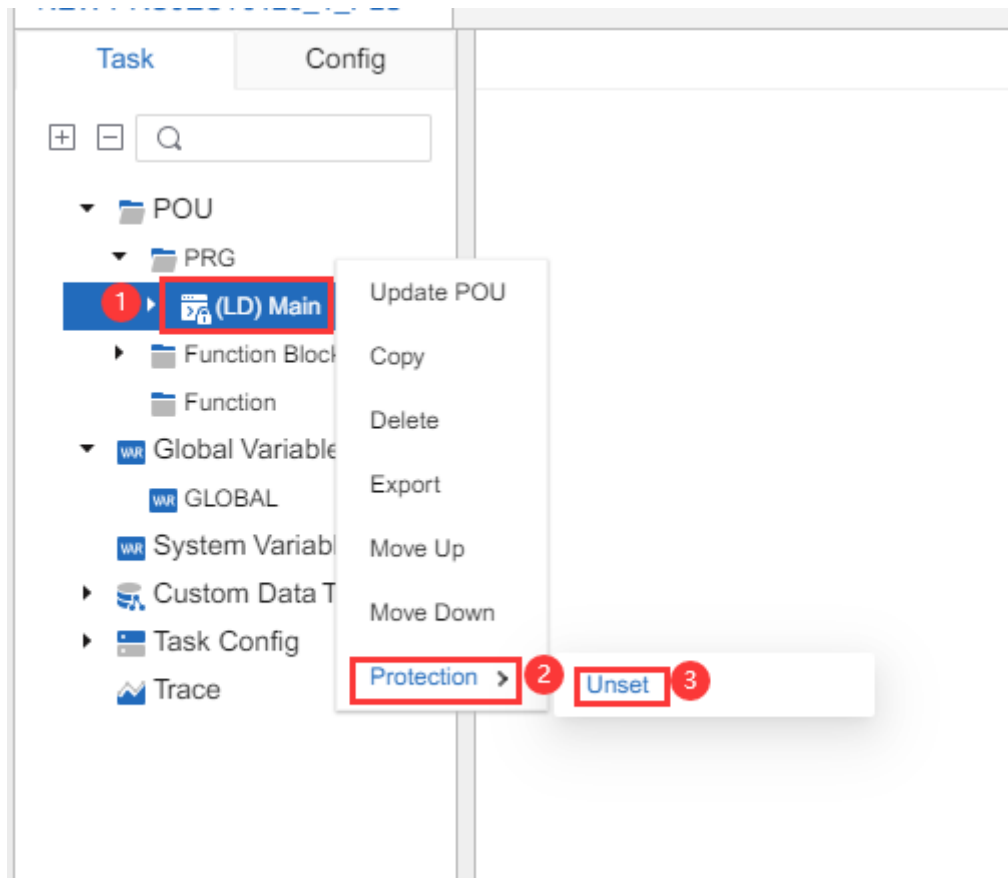
Step27. Set the password and re-enter the password in the pop-up dialog box, and click **Confirm**.



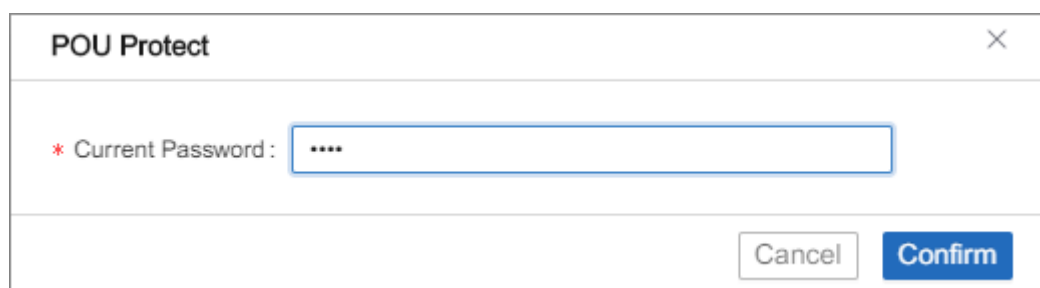
Step28. After closing the project, when editing the program again, you need to enter the password and click **Confirm** for verification.



Step29. Right-click on the encrypted program and select **Protection/Unset**.



Step30. Enter the password in the pop-up dialog box and click **Confirm** to cancel the program password.



8.1.1.2 Function Block

Function Block converts repeatedly used part of the program block into a universal component. It can be called by

any programming language in the program and used repeatedly. It not only improves the development efficiency of the program, but also reduces the programming time and errors, thereby improving program quality.

A function block is a kind of program organization unit that produces one or more values when executed. The function block retains its own special internal variables. The controller target execution system must allocate memory to the internal state variables of the function block. These internal variables constitute its own state characteristics. The execution logic of the function block constitutes its own object behavior characteristics. Therefore, for input variable values of the same parameters, different calculation results may be obtained because there may be different internal state variables. In the control system, the function block can be a certain control algorithm, such as the PID function block for closed-loop control, and can also be used for counters, ramps, and filtering.

Properties of the function block:

◆ **Instantiate**

According to the standard of IEC 61131-3 , the type of function block is the definition of an abstract structural type, rather than a real data entity. If it is not defined and instantiated, it cannot be called and executed by the program. Therefore, function blocks need to be instantiated before they can be used. The instantiated function block is a private data, fully encapsulated, independent structural variable that can complete specific functions according to established logic, thereby converting the previous abstract type definition into a data entity.

◆ **EN and ENO**

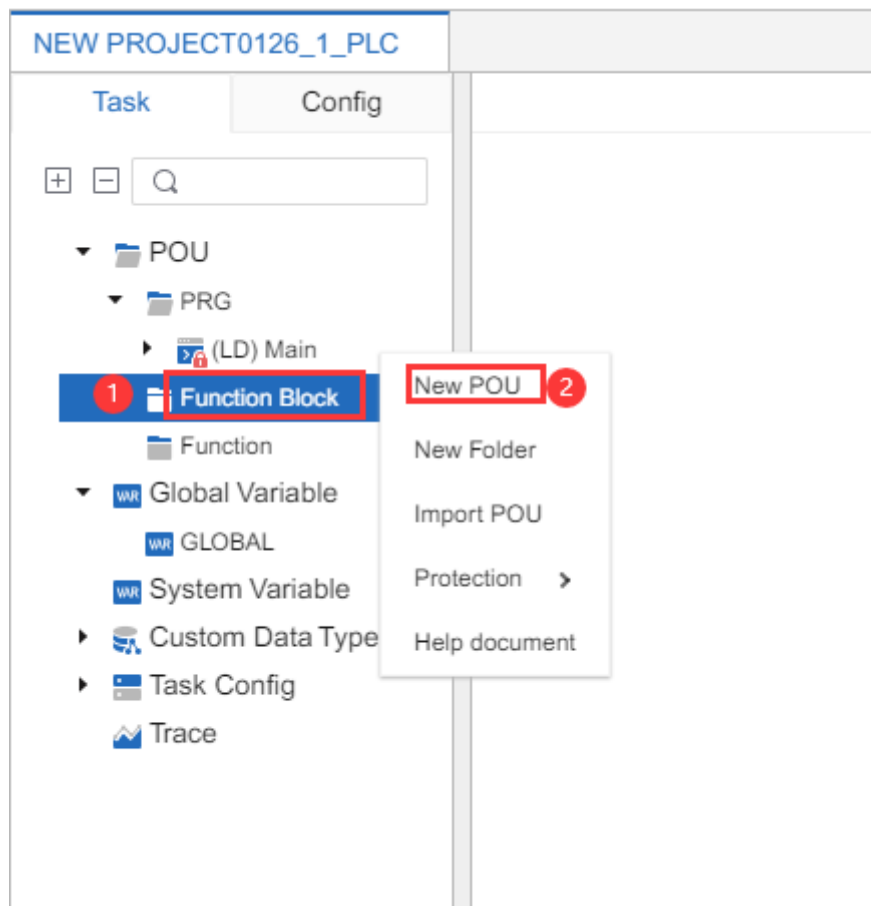
Function blocks have additional attributes of EN and ENO , which are similar to the usage of EN and ENO in functions . EN and ENO are mainly used as the input and output of the detection function block. Simply put, if the function block has energy flow at EN (input is 1) and executes without errors, ENO outputs energy flow to the next element (i.e. output is 1).

Refer to the table below for the differences between functions and function blocks.

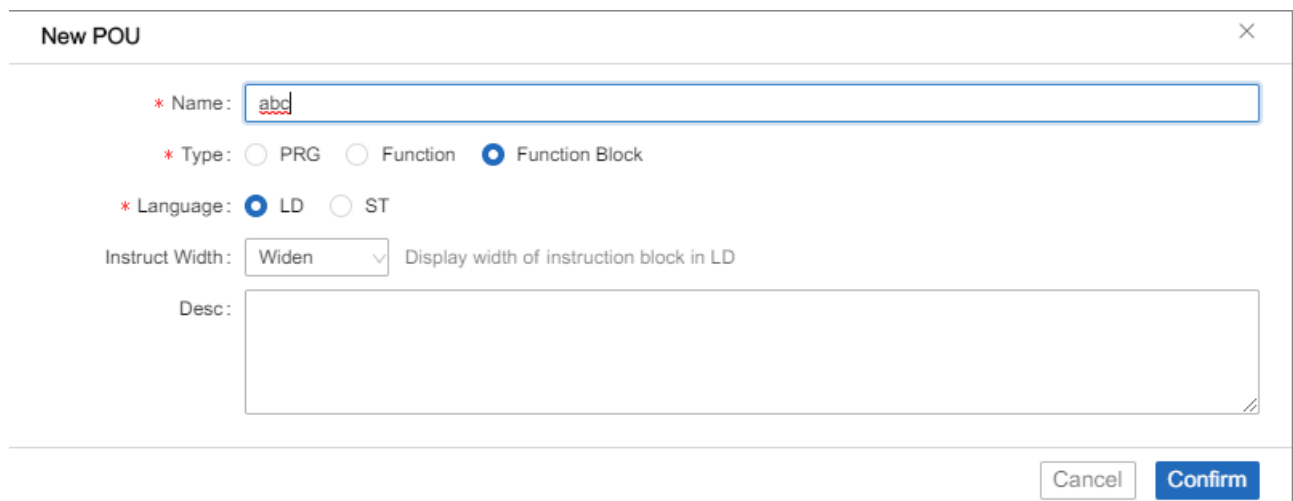
Item	Function (the result is consistent every time it is called)	Function blocks (results may be inconsistent for each call)
Memory allocation	No memory allocation address specified	All data allocated memory address
Input/output variables	By default, there will be a return value variable with the same name as the function name, and the type can be specified.	Multiple output variables or no output variables
Calling relationship	Can call functions, can not call function blocks	Can call function blocks or functions

The operation method of creating a new function block is as follows:

Step31. Right-click on **Function Block** and select **New POU** in the pop-up menu.



Step32. Configure the relevant parameters in the pop-up dialog box and click **confirm**.



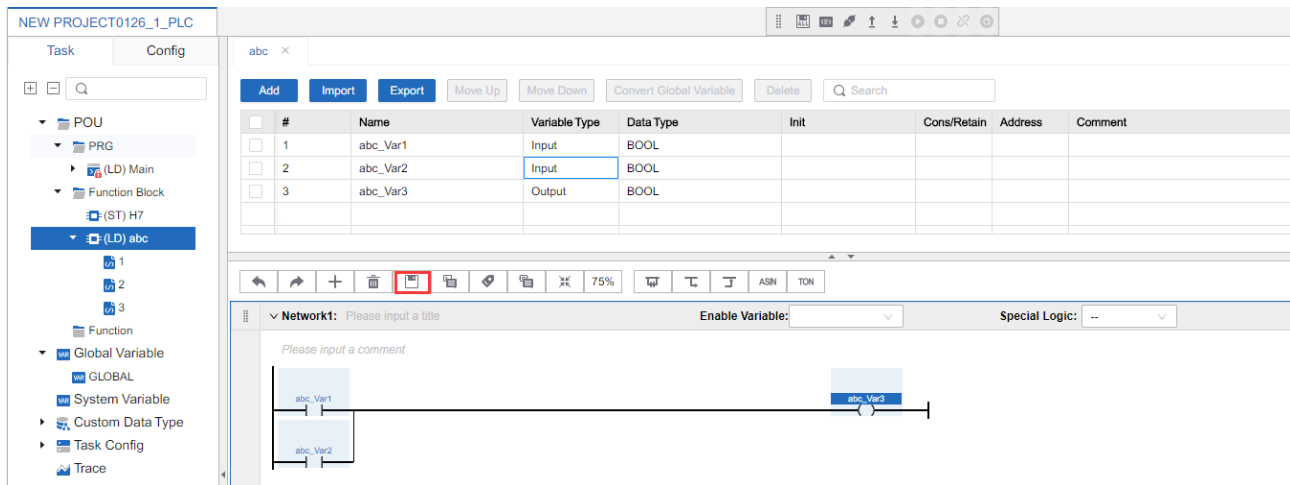
Step33. Declare variables.

Please refer to [Step23](#) for specific operations.

Step34. Edit program.

Please refer to [Step24](#) for specific operations.

Step35. After the function block design is completed, click the  icon to save the configurations.



#	Name	Variable Type	Data Type	Init	Cons/Retain	Address	Comment
1	abc_Var1	Input	BOOL				
2	abc_Var2	Input	BOOL				
3	abc_Var3	Output	BOOL				

Network1: Please input a title

Enable Variable: Special Logic: --

Please input a comment

```

graph LR
    abc_Var1[abc_Var1] --- abc_Var2[abc_Var2]
    abc_Var2 --- abc_Var3[abc_Var3]
  
```

8.1.1.3 Function

A function is also defined as a program organization unit. It is a program organization unit that can be assigned parameters but has no static variables. That is, when a function is called with the same input parameters, the function can always generate the same result as the function value (return value). An important feature of functions is that they cannot use internal variables to store values, which is completely different from function blocks.

Functions are basic units of algorithms that have no internal state (no memory is allocated at runtime). In other words, as long as the same input parameters are given, calling the function will definitely get the same operation result, and there is absolutely no ambiguity. Various mathematical operation functions commonly used, such as $\sin(x)$, $\sqrt{x}$, etc., are typical function types.

A function is a basic algorithm unit with at least one input variable, no private data, and only one return value. Functions can be called by functions, function blocks, and programs.

The properties of the function are as follows:

◆ Overloadability

For a certain function, if its input is described by a generic data type, it is called an overloaded function. This means that the input variable of the function is not limited to a single data type, but can be used for different data types. If a function is only applicable to a certain data type, it needs to be declared in the function name. This is called typing of the function.

◆ Scalability

The attribute that the number of input variables of a function can be expanded is called the scalability of a function. For example, the input variables of the ADD function are not limited to two, and it can implement the addition operation of multiple input variables. Therefore, the ADD function can be said to be scalable. Not all standard functions are scalable. The scalability is limited by the functions of the PLC, the height limit of

the box in the graphic programming language, or the function definition of the function itself. For example, the DIV function has no scalability. Extensible functions simplify programs and reduce the storage space required.

◆ EN and ENO

This attribute is only valid in Ladder Diagram. EN and ENO are the input enablement and output enablement of the function respectively. Its application principles are as follows:

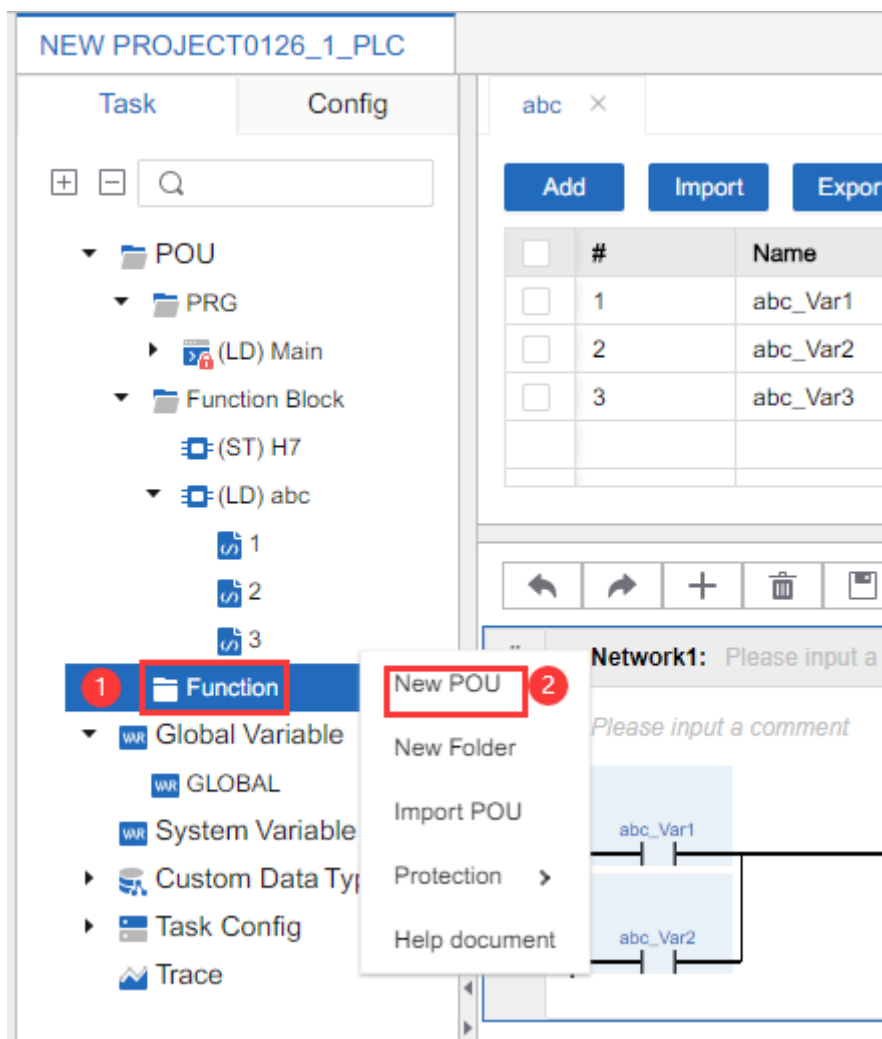
When the function is called and the value of EN is FALSE , the operations defined by the function body will not be executed by the program, and the status value of ENO is FALSE.

EN is TRUE , the function is called, the operation defined in the function body is performed, and the value of ENO is TRUE .

The EN and ENO attributes are additional attribute that can be used or disabled according to actual needs.

The operation method of creating a new function is as follows:

Step36. Right-click **Function** and select **New POU** in the pop-up menu .



Step37. Configure the relevant parameters in the pop-up dialog box and click **confirm** .

New POU

*

Name:

ddd

*

Type:

PRG

Function

Function Block

*

Language:

LD

ST

*

Return Type:

BOOL

▼

Instruct Width:

Widen

▼

Display width of instruction block in LD

Desc:

Cancel

Confirm

Please refer to the table below for detailed configuration methods.

Parameter	Description
Name	Supports English letters, numbers and the special character “_”, and must start with an English letter.
Type	Please select “Function”.
Language	Supports LD (Ladder Diagram) and ST (Structured Text).
Return type	Please refer to the actual data type of the function return value.

Step38. Declare variables.

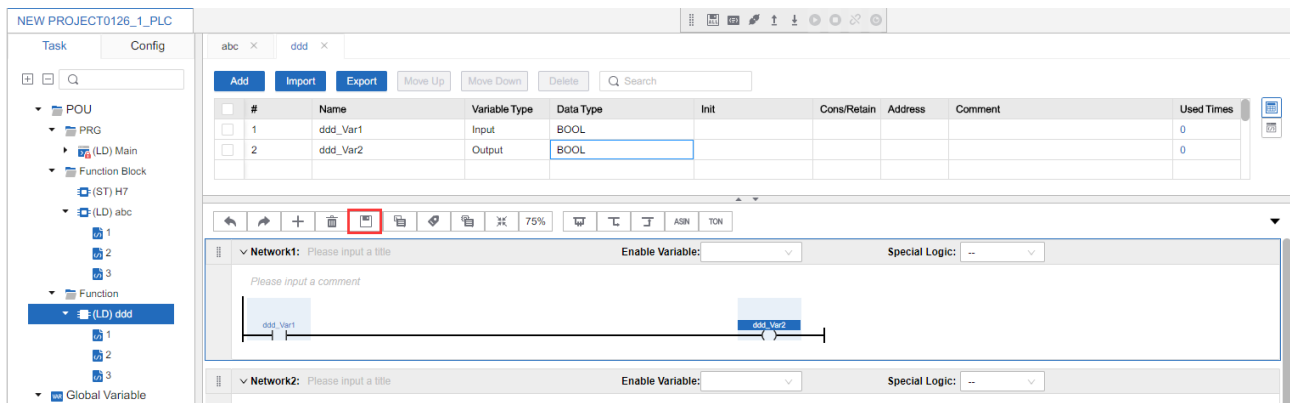
Please refer to [Step23](#) for specific operations.

Step39. Step 4. Design program.

Please refer to [Step24](#) for specific operations.

Step40. After the function design is completed, click  icon to save the configuration.

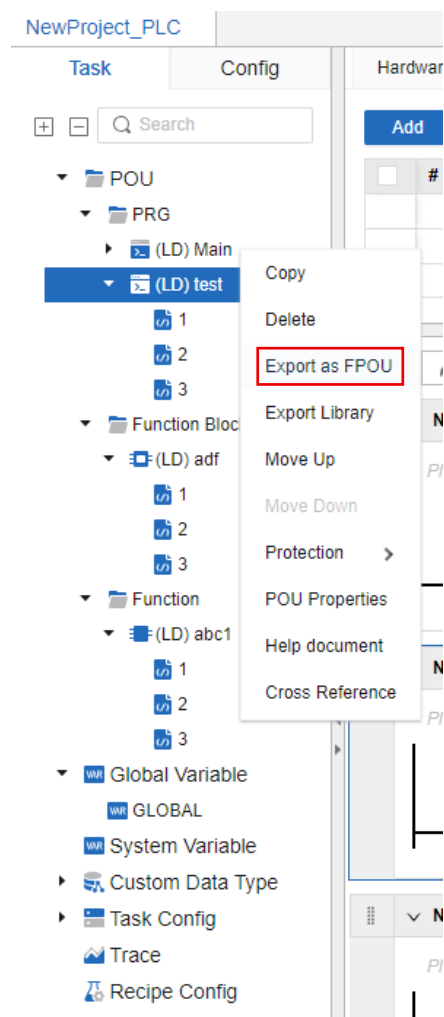
299



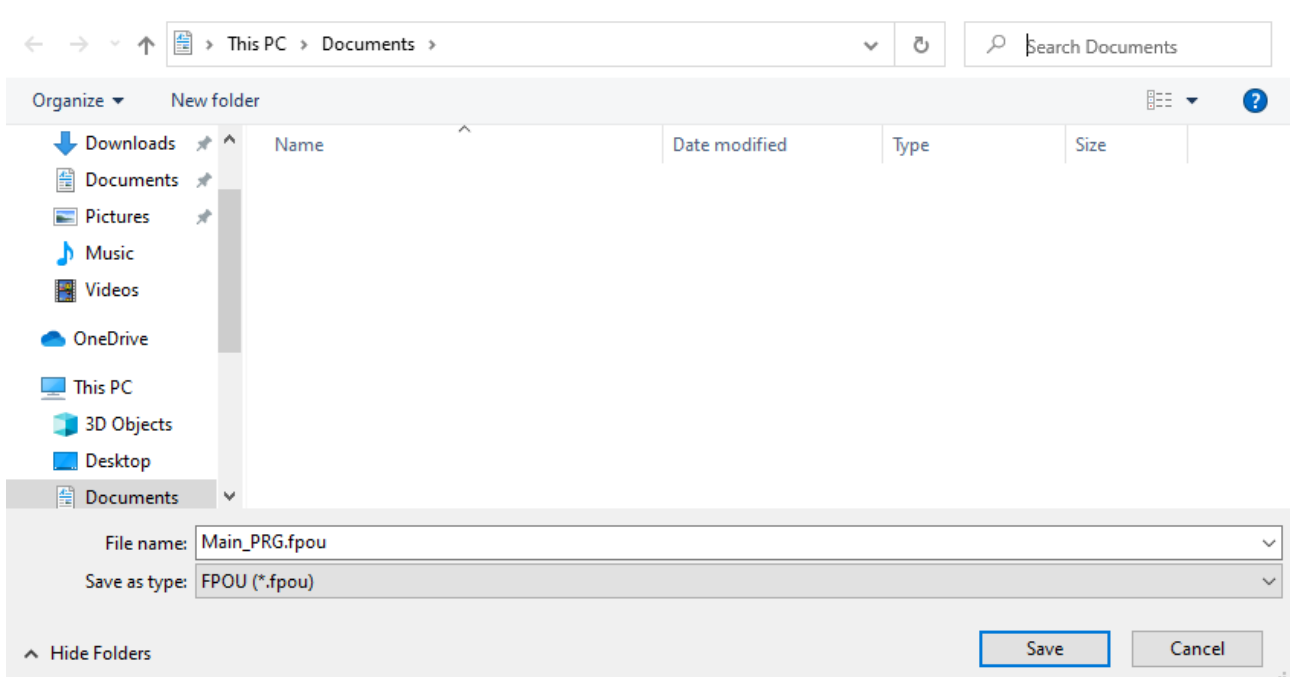
8.1.1.4 Export Program Organization Unit

MCR Master supports exporting program organization units to facilitate program modularization and sharing with others. The operation method of exporting program organizational units is as follows:

Step41. Right-click the program organization unit and select **Export** from the pop-up menu.



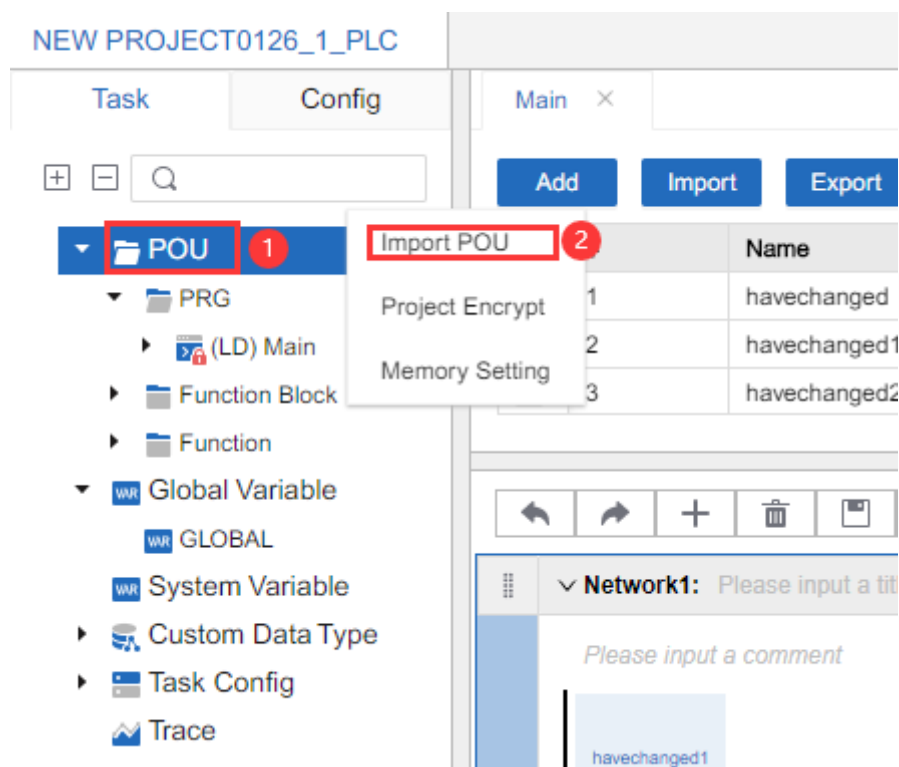
Step42. Set the save path and file name in the pop-up dialog box, and click **Save**.



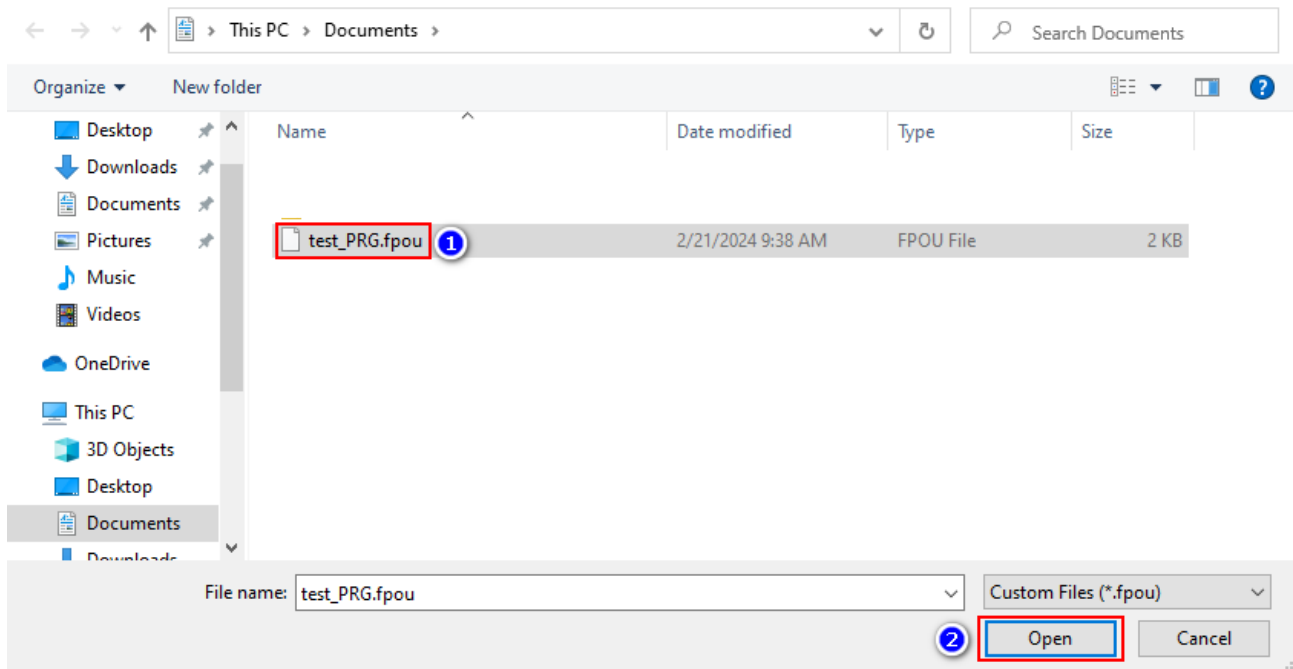
8.1.1.5 Import Program Organization Unit

MCR Master supports the import of program organization units, which facilitates the reuse of POU in other projects, modularizes programs, and simplifies programming difficulty. The operation method of importing program organizational units is as follows:

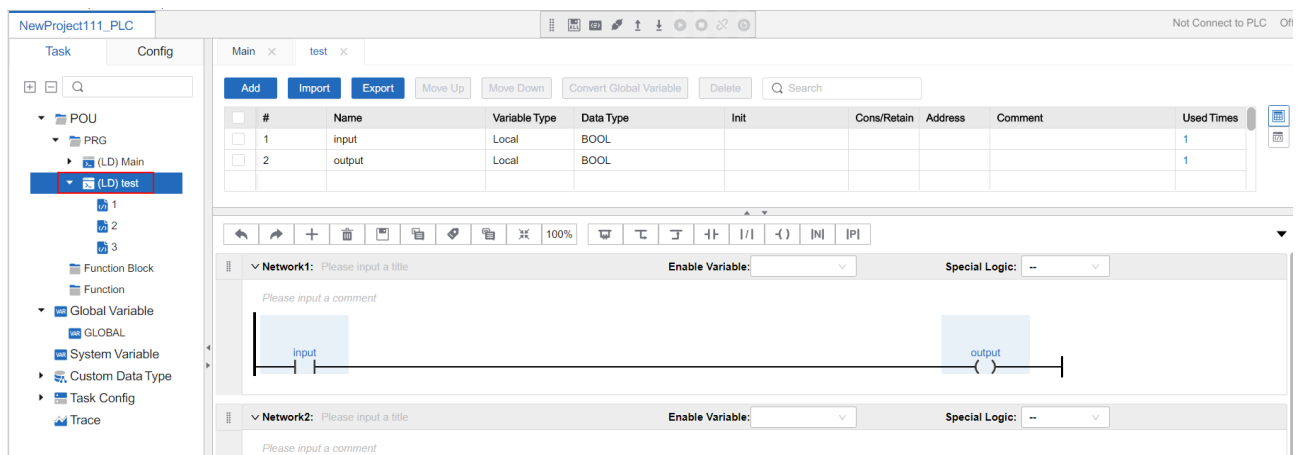
Step43. Right-click **POU** (program organization unit) (or program, function block, function) and select **Import POU**.



Step44. Select the file (the exported POU file) in the pop-up dialog box and click **Open** .



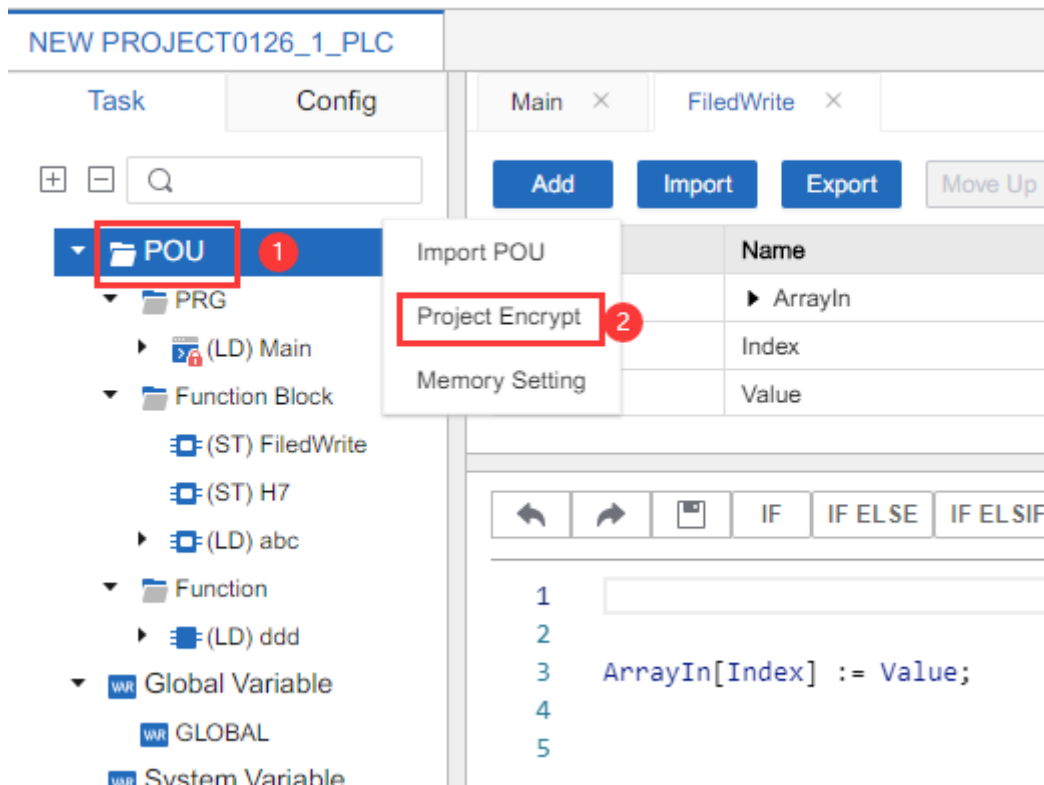
Step45. After importing a program organizational unit, you can view and modify the imported program organizational unit.



8.1.1.6 Project Encryption

Set a password for the project to prevent unauthorized users from using the project.

Step46. Right-click **POU** and select **Project Encrypt**.



Step47. Set the password and confirm the password in the pop-up dialog box, and click **Confirm**.

The 'Project Encrypt' dialog box is shown. It has a title bar with a close button (X). The main text reads: 'Please set a password to protect this project. After set it, password-protected project cannot be opened without providing this password. Please keep properly. If the password is lost, it cannot be retrieved.' Below this text are two input fields: '* New Password :' and 'Confirm Password :'. Both fields have a password icon (a dot) and a clear button (an X). At the bottom right, there are two buttons: 'Cancel' and 'Confirm'.

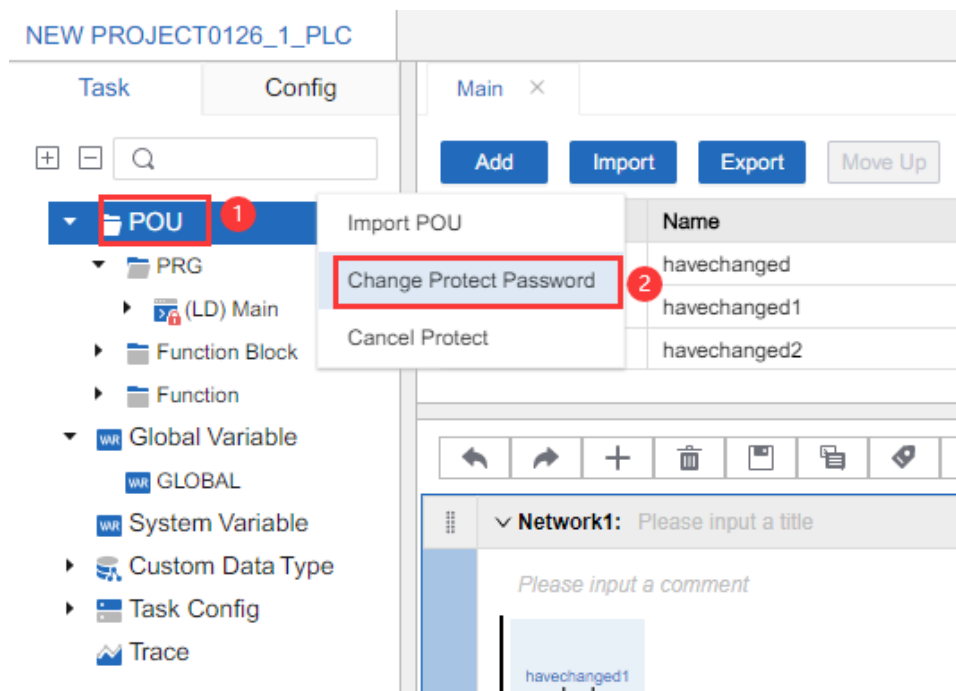
Step48. When re-opening the project, you need to enter the password for verification.

The 'Project Encrypt' dialog box is shown for password verification. It has a title bar with a close button (X). The main text reads: 'Please input the password to open the project.' Below this text is a single input field: '* Password :'. The field has a password icon (a dot) and a clear button (an X). At the bottom right, there are two buttons: 'Cancel' and 'Confirm'.

Other operations

- ◆ Change project protection password

Step49. Right-click **POU** and select **Change Protect Password** in the pop-up menu .



Step50. Enter the current password in the pop-up dialog box, set a new password, confirm the new password, and click **Confirm**.

Project Encrypt ×

Please input the current password and new one. This project will be protected by the new password if passing the verification.

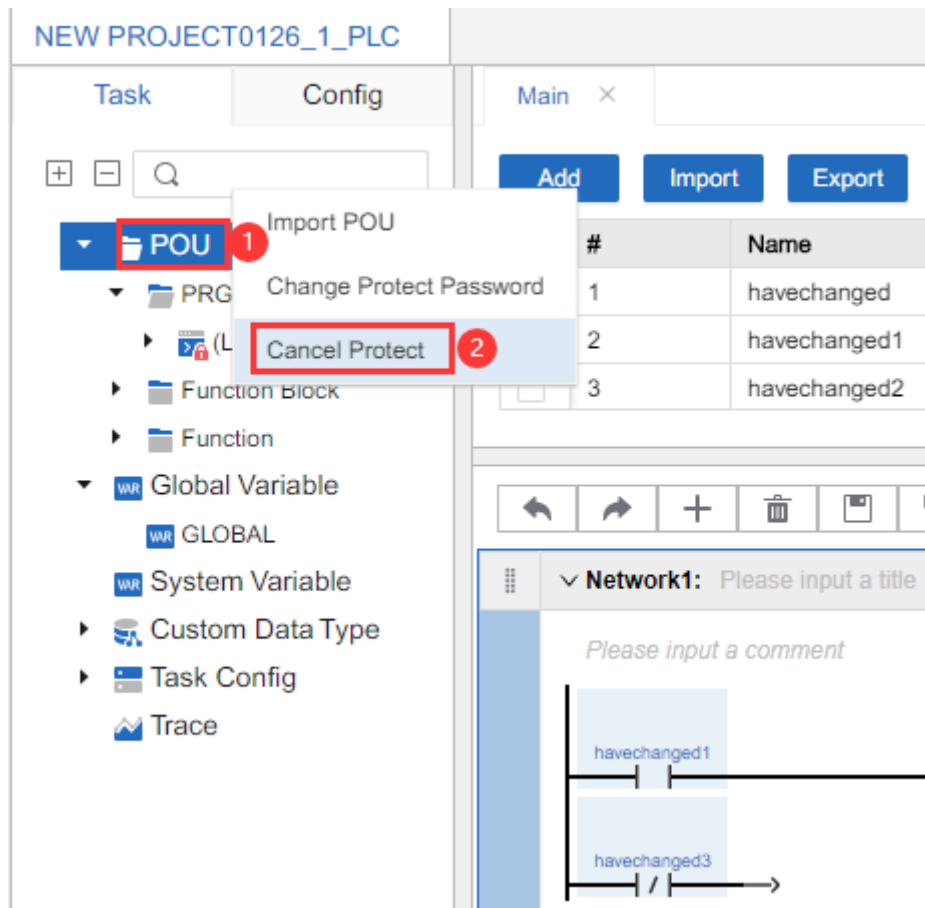
* Current Password :

* New Password :

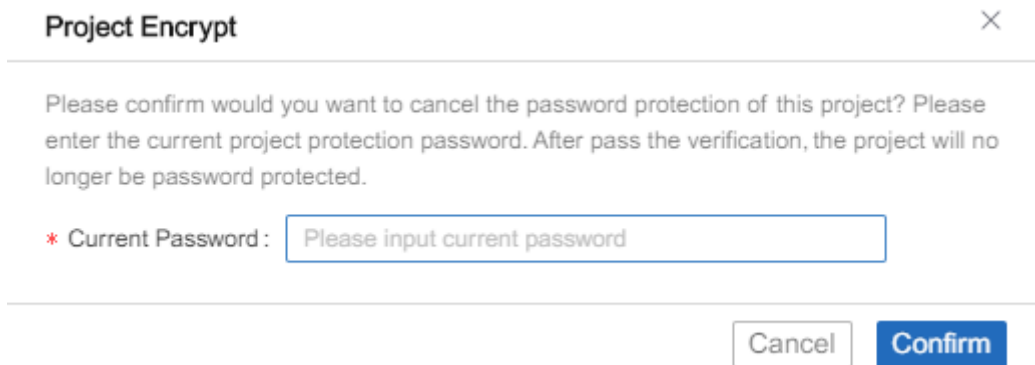
Confirm Password :

◆ Cancel project protection password

Step51. Right-click **POU** and select **Cancel Protect** in the pop-up menu.



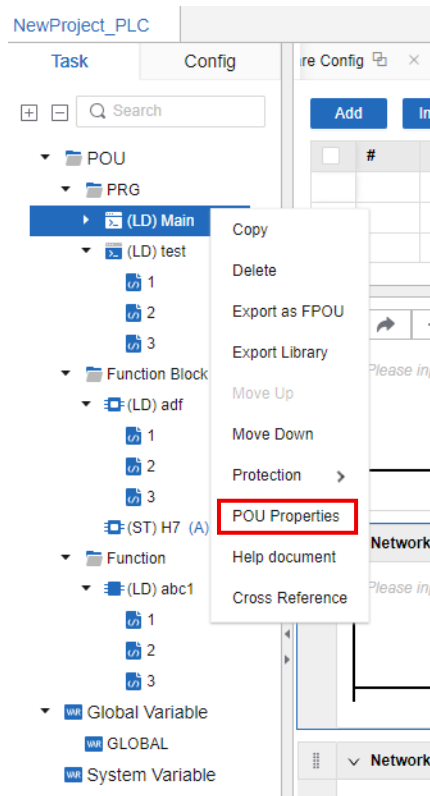
Step52. Enter the current password in the pop-up dialog box and click **Confirm**. After passing the verification, the project is no longer protected by the password.



8.1.1.7 Modify the Properties of The Program Organizational Unit

You can modify the properties of the program organizational unit. The operation method is as follows:

Step53. Right-click the program organizational unit and select **POU Properties**.



Step54. Modify the name, type and programming language in the pop-up dialog box and click **Confirm** .

Update POU

* Name:

Main

* Type:

☒ PRG
 ☐ Function
 ☐ Function Block

* Language:

☒ LD
 ☐ ST

Desc:

Cancel

Confirm

8.1.1.7.1 Modify the POU Type

The conditions for POU type conversion are as follows:

◆ Convert function to function block

The return value of the function is discarded directly in the converted function block. If the function has been called by the POU, modifications need to be checked manually. For other issues, please pay attention to the compilation results.

◆ Convert function to program

The return value of the function is discarded directly in the converted program. If the data types of “input

variable”, “output variable”, and “input/output” are used in the function variable declaration, they need to be manually modified to the appropriate data type in the program. If the function has been called by POU, you need to manually check the modification. For other issues, please pay attention to the compilation results.

◆ Convert function block to program

If the data types of “input variables”, “output variables”, and “input/output” variables are used in the function block variable declaration, they need to be manually modified to the appropriate data types in the program. If the function block has been called by the POU, modifications need to be checked manually. For other issues, please pay attention to the compilation results.

◆ Convert function block to function

If a “temporary variable” is used in the function block variable declaration, it will be converted into a “local variable” in the function, and the “retain/constant” configuration information in the variable declaration will be directly discarded. If global variables (G.xx) or system variables (S.xx) are used in the function block , they need to be modified manually in the converted function. If the function block has been called by the POU, modifications need to be checked manually. For other issues, please pay attention to the compilation results.

◆ Convert program to function

If the variable uses an address in the program variable declaration, or uses the “retain/constant” configuration, it will be directly discarded after being converted into a function. If a global variable (G.xx) or a system variable (S.xx) is used in the program , you need to manually modify it in the converted function; if the program has been called in the task, you need to manually check the modification; for other issues, please pay attention to the compilation results.

◆ Convert program to function block

If the variable address is used in the program variable declaration, it will be directly discarded after being converted into a function block. If the program has already been called in a task, modifications need to be checked manually. For other issues, please pay attention to the compilation results.

8.1.1.7.2 Modify the Programming Language of POU

Only converting Ladder Diagram into Structured Text is supported, and this operation is irreversible. It is recommended to backup the original POU before operation.

Update POU

×

* Name:

Main

* Type:

☒ PRG ☐ Function ☐ Function Block

* Language:

☐ LD ☒ ST

This operation is irreversible, please use caution!

Backup POU:

☒

Backup Location:

Desc:

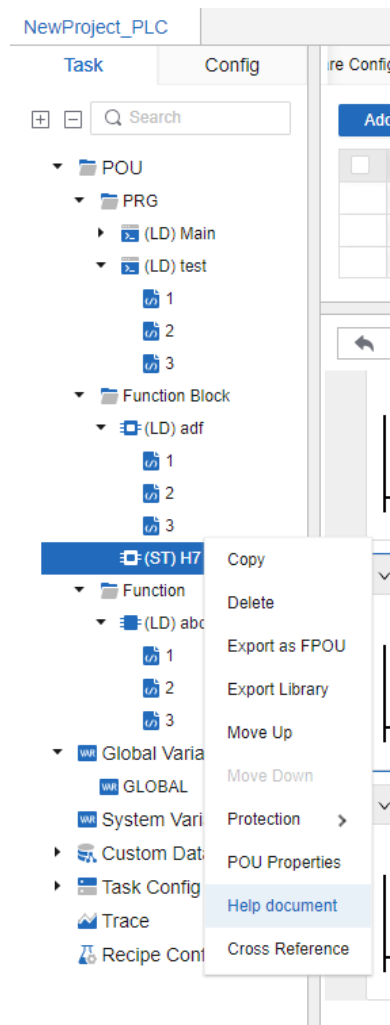
Cancel

Confirm

8.1.1.8 Adjust the Order of POU

There're two methods to adjust the order of POU.

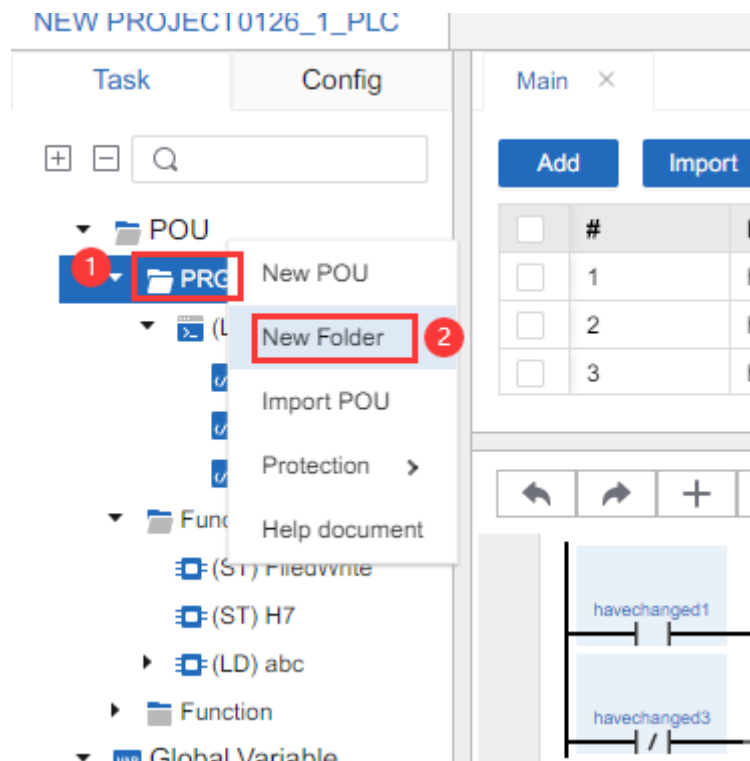
- ◆ Method 1: Right-click a program organization unit (such as a program) and select **Move Up** to move the program organization unit up one position; select **Move Down** to move the program organization unit down one position.
- ◆ Method 2: Adjust the position by dragging the program organization unit.



8.1.1.9 Create a New Folder

In order to manage POU's in groups, creating a new folder is supported. The operation method is as follows:

Step55. Right-click **RPG (Function Block or Function)** and select **New Folder** in the pop-up menu.



Step56. Set the folder name in the pop-up **New Folder** dialog box and click **Confirm**.



You can move a POU to a folder, or you can move a POU out of a folder.

8.1.2 Global Variable

Global variables are variables that can be referenced by all objects or functions in this project.

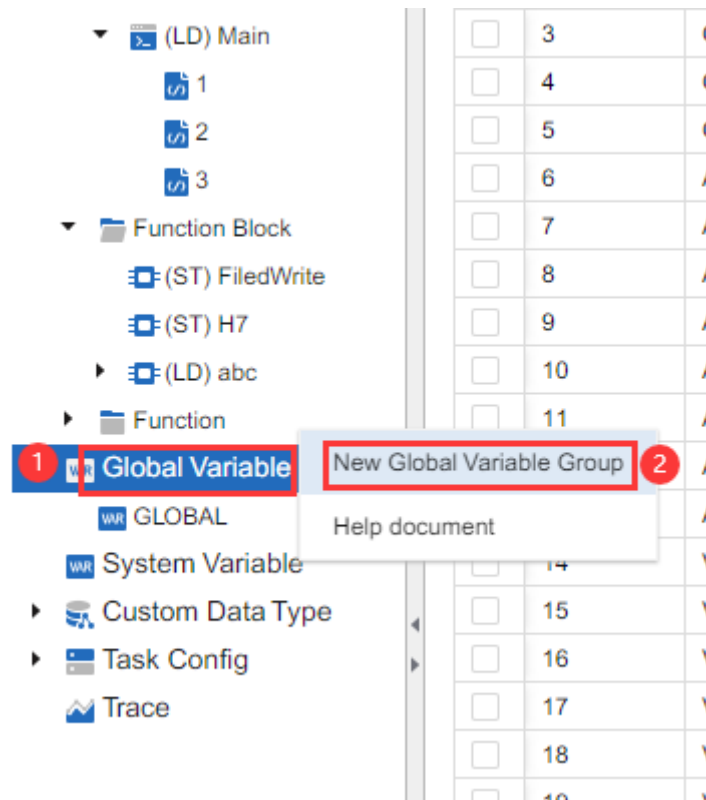
8.1.2.1 Global Variable Group

A global variable group is a collection of specific global variables that facilitates users to classify and manage global variables. MCR Master has built-in global variable group GLOBAL.

8.1.2.1.1 Create a New Global Variable Group

The operation method of creating a new global variable group is as follows:

Step57. Right-click **Global Variable** and select **New Global Variable Group** in the pop-up menu .



Step58. Configure the name in the pop-up dialog box (no more than 20 characters, supports Chinese, English, numbers and special characters), and click **Confirm**.

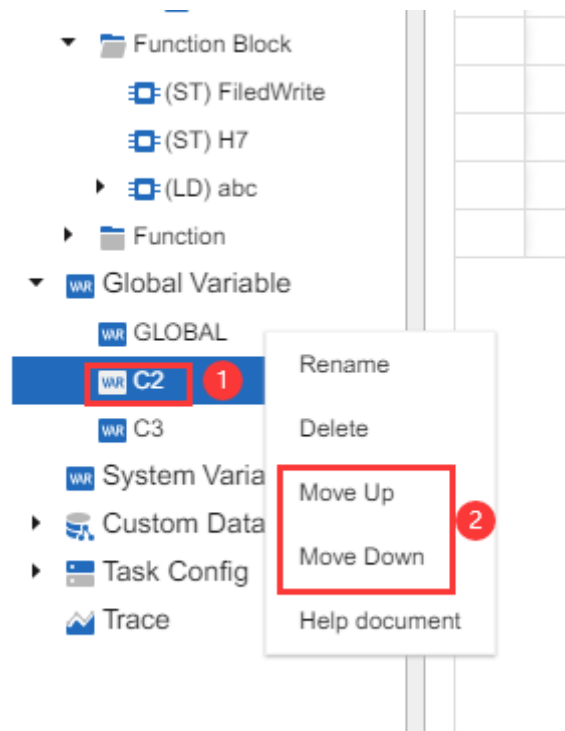
New Global Variable Group
×

* Name :

Cancel
Confirm

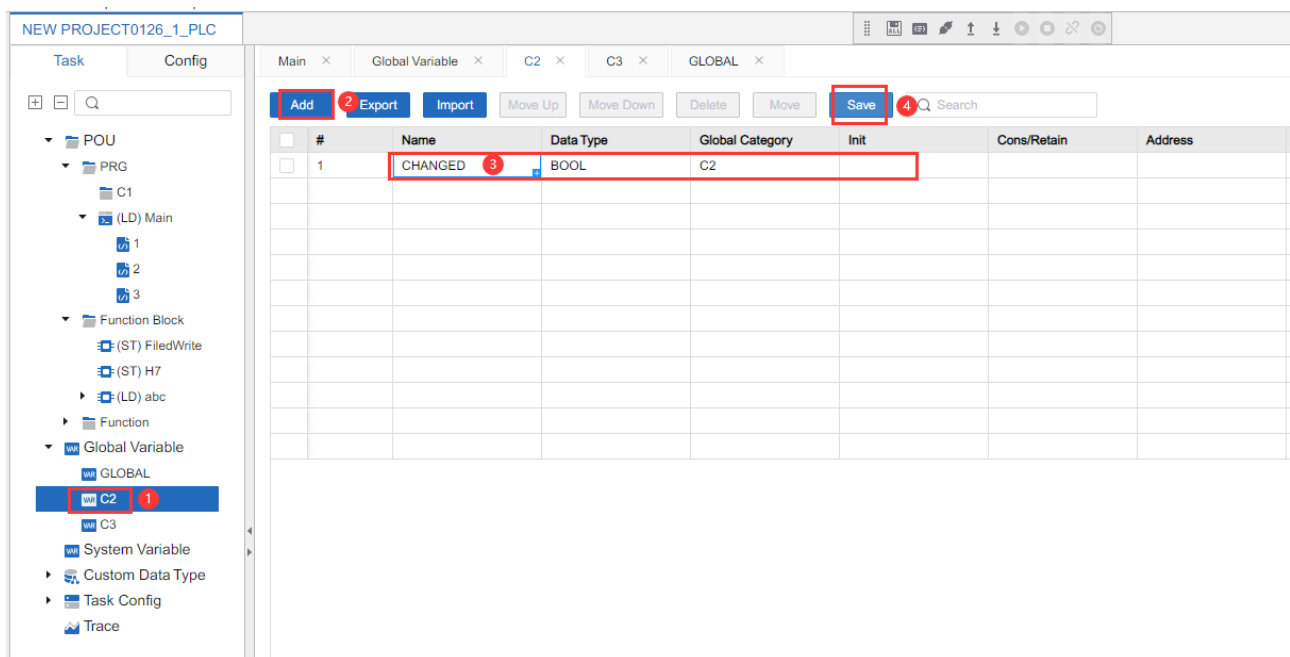
8.1.2.1.2 Adjust the Order of Global Variable Groups

Right-click the global variable group and select **Move Up** to move the global variable group up one position; select **Move Down** to move the global variable group down one position.



8.1.2.2 Add Global Variable

Select the global variable group, click **Add**, configure relevant parameters, and click **Save**.



Please refer to the table below for detailed configuration methods.

Parameter	Description
Name	Variable name, support Chinese, English letters, numbers and special character “_”, can not start with a number.

Parameter	Description
Data Type	Please refer to the actual situation. For more information about data types, refer to Data Type of Variable .
Global Category	Global variable group. For detailed information about global variable group, please refer to Global Variable Group .
Init	The initial value of the variable. For more information about variable initialization, refer to Initialization of Variable .
Constant/Retain	◆ Constant: The value of a variable is a constant. ◆ Retain: Force variables in the online monitoring state. When switching to offline state, check Retain the force state and value of variables, and the variables will be the result of force state.
Address	Address to which the variable is bound.
Comment	Description of the variable.

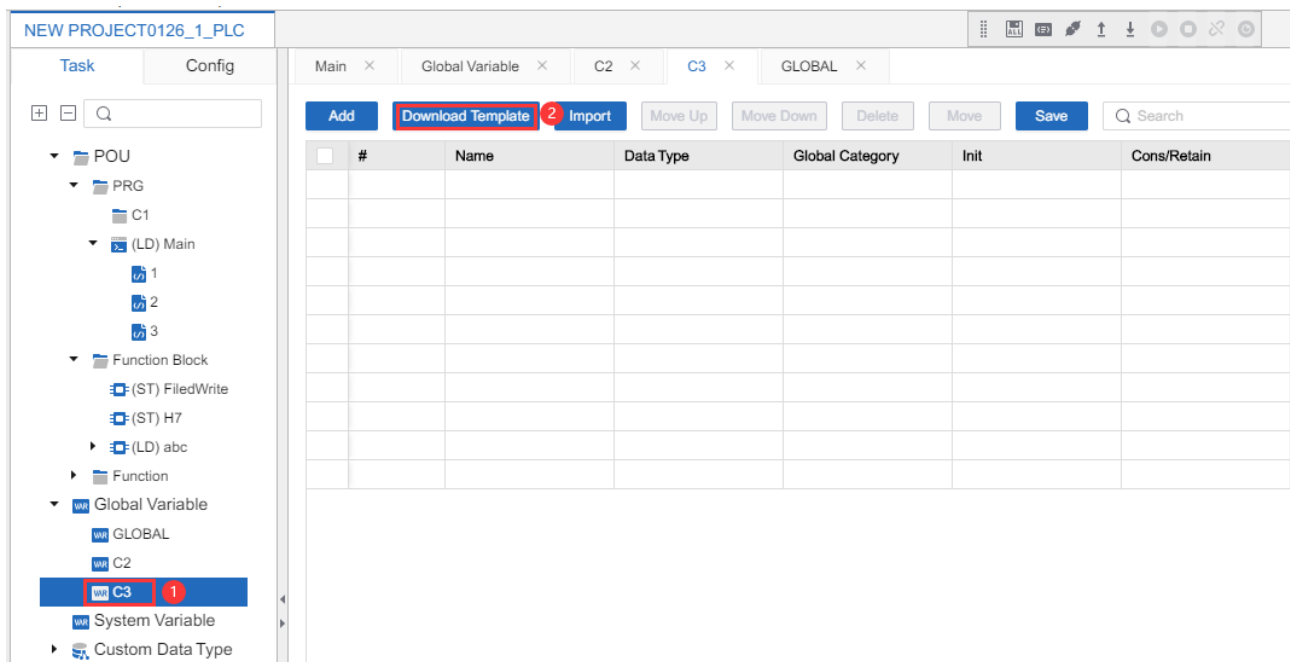
8.1.2.3 Import Global Variable

MCR Master provides operation methods for importing global variables to improve the efficiency of creating new global variables.



MCR Master supports importing variables of third-party PLCs (including Mitsubishi FX series PLCs, Omron CP series PLCs and Siemens S7-200 series PLCs), and it is necessary to set the address template to the address template of the third-party PLCs when you create a new project. Please refer to [Create a New Project](#) for details .

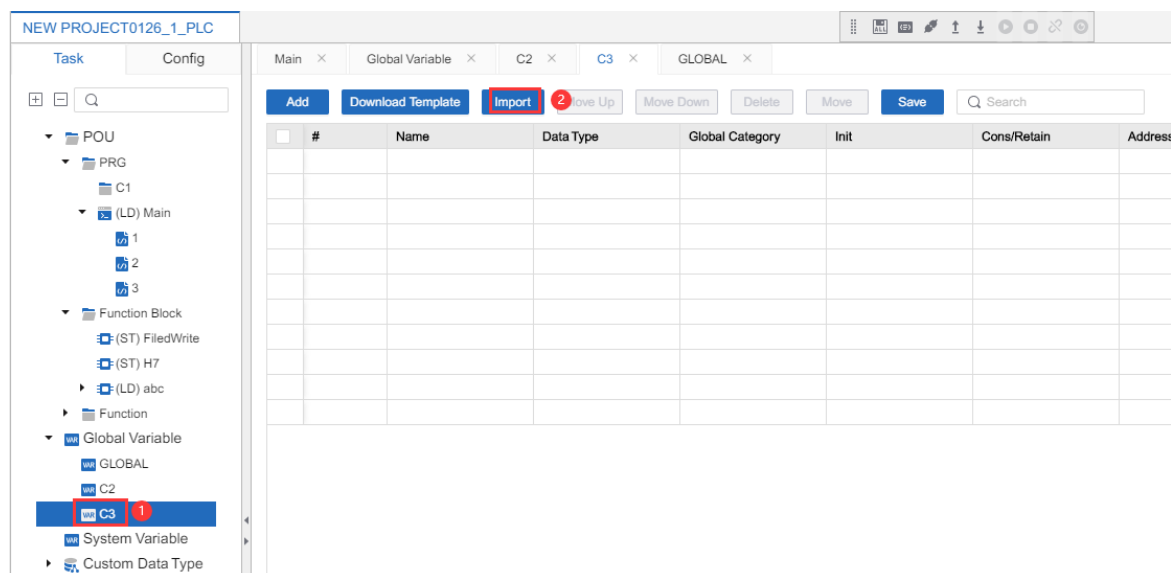
Step59. Select the global variable group, click **Download Template** to download the template file locally.



Step60. Edit the template file and save it.

	A	B	C	D	E	F	G
1	Name	Type	Initial	Cons/Retain	Address	Documentation	Category
2	HEIGHT	REAL					GLOBAL
3	LENGTH	REAL					GLOBAL
4	WIDTH	REAL					GLOBAL
5	Temperature	INT					environment

Step61. Select the global variable group and click **Import**.



Step62. Click **Select** in the pop-up dialog box, select the edited template file, and click **Confirm**.

Import Variable

After consting variables successfully, the new variable table will overwrite the existing variable table

Select

GVL_NEW+PROJECT0126_1_2024-01-30-17-59-06.xlsx

Cancel

Confirm

Step63. After importing the global variables, click **Save**.

NEW PROJECT0126_1_PLC

Task Config

Main GLOBAL C3 System Variable

#	Name	Data Type	Access Level	Comment
1	Always_On	BOOL	Read Only	Always connected
2	CPU_ID	DWORD	Read Only	An identify to distinguish CPU model
3	Clock_100ms	BOOL	Read Only	This bit provides a clock pulse. The bit is FALSE for 50 milliseconds and then TRUE for 50 milliseconds for a cycle time of 100 milliseconds
4	Clock_10ms	BOOL	Read Only	This bit provides a clock pulse. The bit is FALSE for 5 milliseconds and then TRUE for 5 milliseconds for a cycle time of 10 milliseconds
5	Clock_10s	BOOL	Read Only	This bit provides a clock pulse. The bit is FALSE for 5 seconds and then TRUE for 5 seconds for a cycle time of 10 seconds
6	Clock_1s	BOOL	Read Only	This bit provides a clock pulse. The bit is FALSE for 0.5 seconds and then TRUE for 0.5 seconds for a cycle time of 1 second
7	Clock_2s	BOOL	Read Only	This bit provides a clock pulse. The bit is FALSE for 1 seconds and then TRUE for 1 seconds for a cycle time of 2 seconds
8	Clock_5s	BOOL	Read Only	This bit provides a clock pulse. The bit is FALSE for 2.5 seconds and then TRUE for 2.5 seconds for a cycle time of 5 seconds
9	Clock_60s	BOOL	Read Only	This bit provides a clock pulse. The bit is FALSE for 30 seconds and then TRUE for 30 seconds for a cycle time of 60 seconds
10	EM_1_Disconnect	BOOL	Read Only	Extension module 1 disconnected
11	EM_1_Not_Exists	BOOL	Read Only	Extension module 1 does not exist when cables are connected
12	EM_2_Disconnect	BOOL	Read Only	Extension module 2 disconnected
13	EM_2_Not_Exists	BOOL	Read Only	Extension module 2 does not exist when cables are connected
14	ExBus_Err	BOOL	Read Only	Expansion bus is abnormal
15	First_Scan_On	BOOL	Read Only	The CPU sets this bit to TRUE, for the first scan cycle, and sets it to FALSE thereafter
16	IO_Err	BOOL	Read Only	This bit is set ON if any I/O errors are present
17	RTC_Lost	BOOL	Read Only	The CPU sets this bit to TRUE for one scan cycle if the time on the real time clock device was reset or lost at power-up
18	RUN_Power_Up	BOOL	Read Only	The CPU sets this bit to TRUE for one scan cycle when RUN mode is entered from a power-up or warm restart condition
19	Retentive_Lost	BOOL	Read Only	This bit can be used as either an error memory bit or as a mechanism to invoke a special start-up sequence
20	TimeStamp	DWORD	Read Only	Time stamp

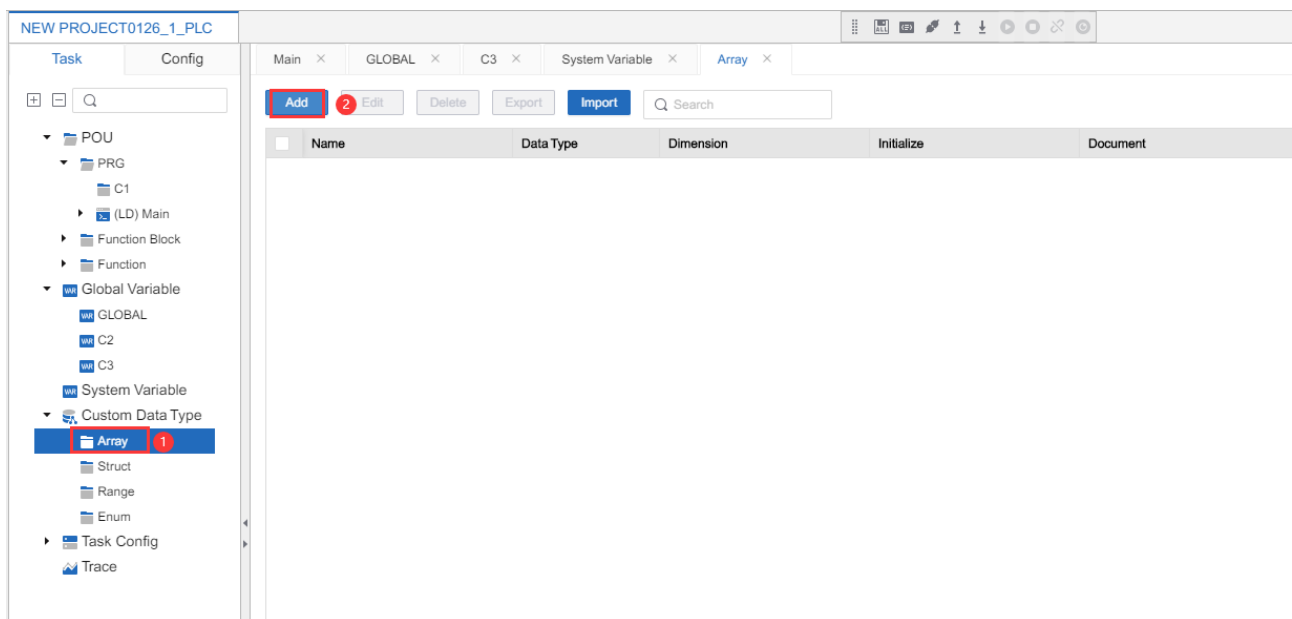
8.1.4 Custom Data Type

Custom data types are data types other than basic data types (such as integers, Boolean, strings, etc.), including array, structure, range, and enumeration.

8.1.4.1 Array

The operation method of adding a new array is as follows:

Step64. Select **Custom data type/array** and click **Add**.



Step65. Configure the relevant information in the pop-up dialog box and click **Confirm**.

✕

Add Array

* Name :

Desc :

Data Type : ▼

StartIndex : EndIndex :

+ Add Dimension

Please refer to the table below for detailed configuration methods.

Parameter	Description
Name	Array name. Support Chinese, English letters, numbers and the special character “_”. Cannot start with a number. Cannot be the same as the program organizational unit name, instruction name, or other custom data type name.
Desc	Description of the array.
Data Type	Please refer to the actual situation.
StartIndex	The starting index of the array dimension.
EndIndex	The end index of the array dimension.
Add dimension	Click Add Dimension to add the array dimension. The default is a one-dimensional array.

Step66. Click the icon  in the **Initialize** column.

NEW PROJECT0126_1_PLC

Task
Config
Main
GLOBAL
C3
System Variable
Array

Add
Edit
Delete
Export
Import
Q Search

	Name	Data Type	Dimension	Initialize	Initialize	Document
☐	newer	TIME	[0...5]			

POU
PRG
C1
(LD) Main
Function Block
Function
Global Variable
GLOBAL
C2
C3
System Variable
Custom Data Type
Array
Struct
Range
Enum
Task Config
Trace

Step67. Set the initial value of the array in the pop-up dialog box and click **Confirm**.

Initial Value

Data Type:TIME

Index	Data Type	Initialize
[0]	TIME	TIME#1ms
[1]	TIME	TIME#2ms
[2]	TIME	TIME#0ms
[3]	TIME	TIME#0ms
[4]	TIME	TIME#0ms
[5]	TIME	TIME#0ms

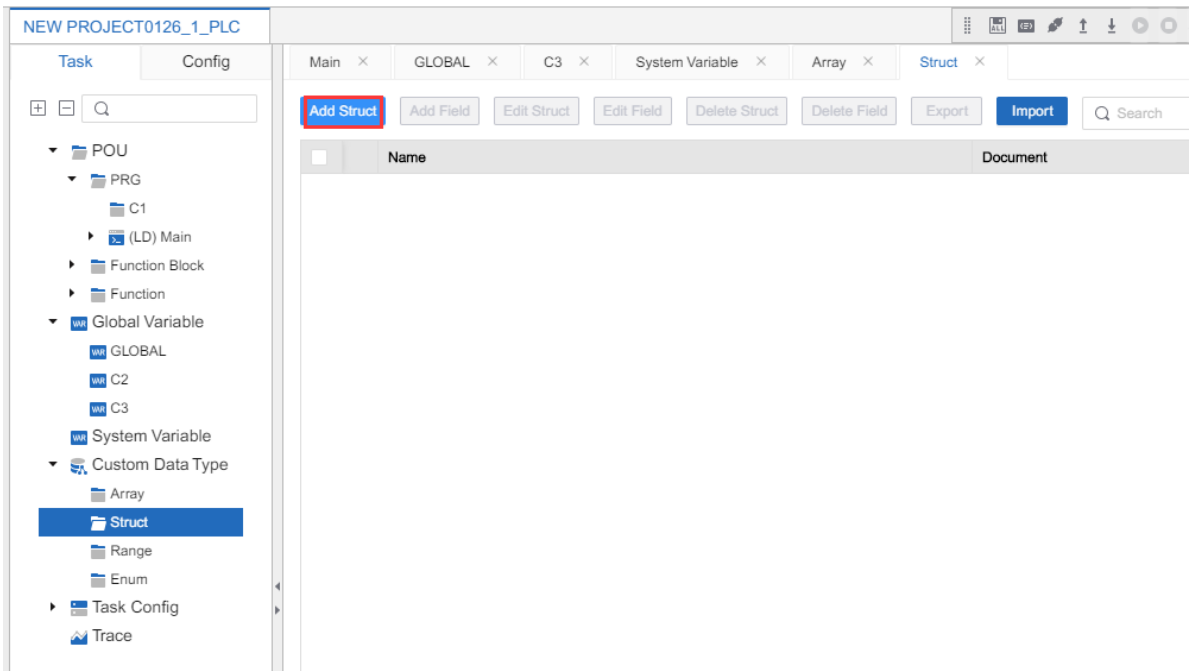
Cancel

Confirm

8.1.4.2 Structure

The operation method of adding a new structure is as follows:

Step68. Select **Custom Data Type/Structure** and click **Add Structure** .



Step69. Configure the name and description information in the pop-up dialog box, and click **Confirm**.

Add Struct
✕

* Name :

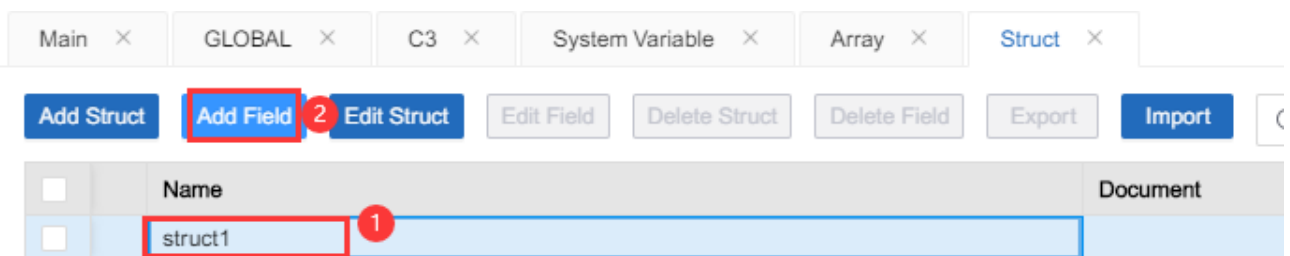
Desc :

Cancel
Confirm

Please refer to the table below for detailed configuration methods.

Parameter	Description
Name	Structure name. Supports Chinese, English letters, numbers and the special character “_”. Cannot start with a number. Cannot be the same as the program organizational unit name, instruction name, or other custom data type name.
Description	Description information of the structure.

Step70. Select the structure and click **Add Field**.



Step71. Edit the name, select the data type In the pop-up dialog box, and click **Confirm**.

Add Field to Struct struct1

* Name :

I1

Data Type :

INT

Cancel

Confirm

Please refer to the table below for detailed configuration methods.

parameter	Description
Name	Field name. Supports Chinese, English letters, numbers and the special character “_”, and cannot start with a number.
Data Type	Includes basic data types and custom data types.

Step72. Set initialization values. (Optional)

Main * ×GLOBAL ×C3 ×System Variable ×Array ×Struct ×

Add StructAdd FieldEdit StructEdit FieldDelete StructDelete FieldExportImport

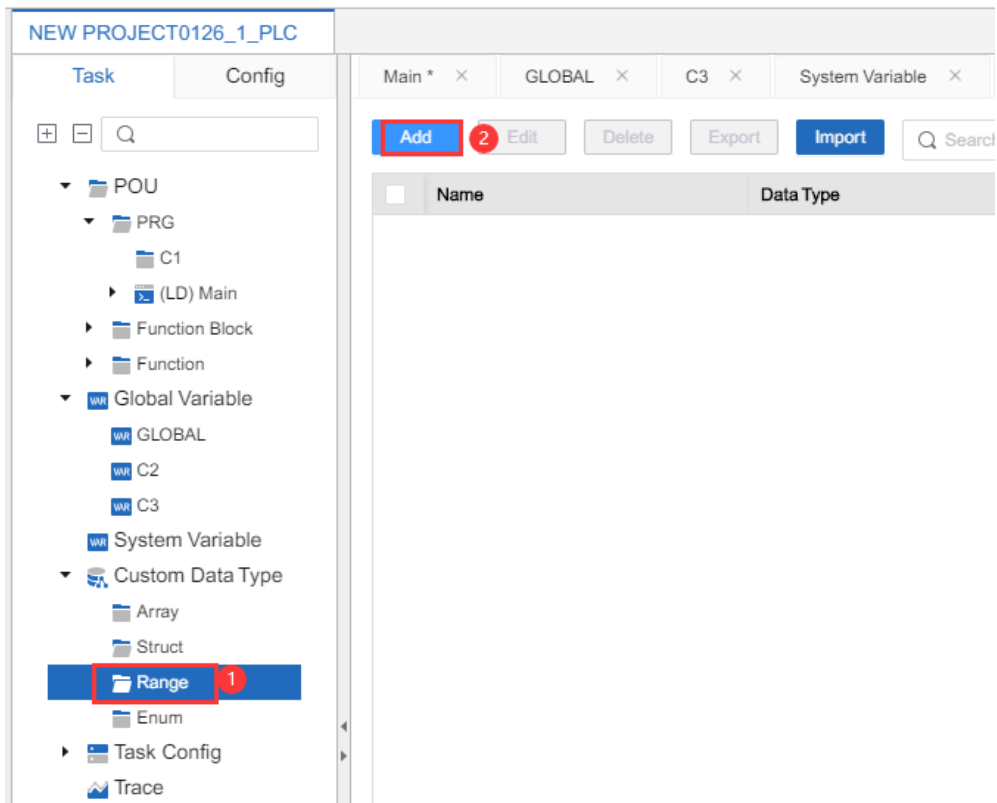
Q Search

	Name	Document	
	▼ struct1		
	Field	Data Type	Initialize
	I1	INT	15

8.1.4.3 Range

The operation method of adding a new range is as follows:

Step73. Select **Custom Data Type/Range** and click **Add**.



Step74. Configure the relevant parameters in the pop-up dialog box and click **Confirm**.

Add Range

* Name :

TEMP

Value Type :

INT

Min :

-10

Max :

40

Init :

25

Cancel

Confirm

Please refer to the table below for detailed configuration methods.

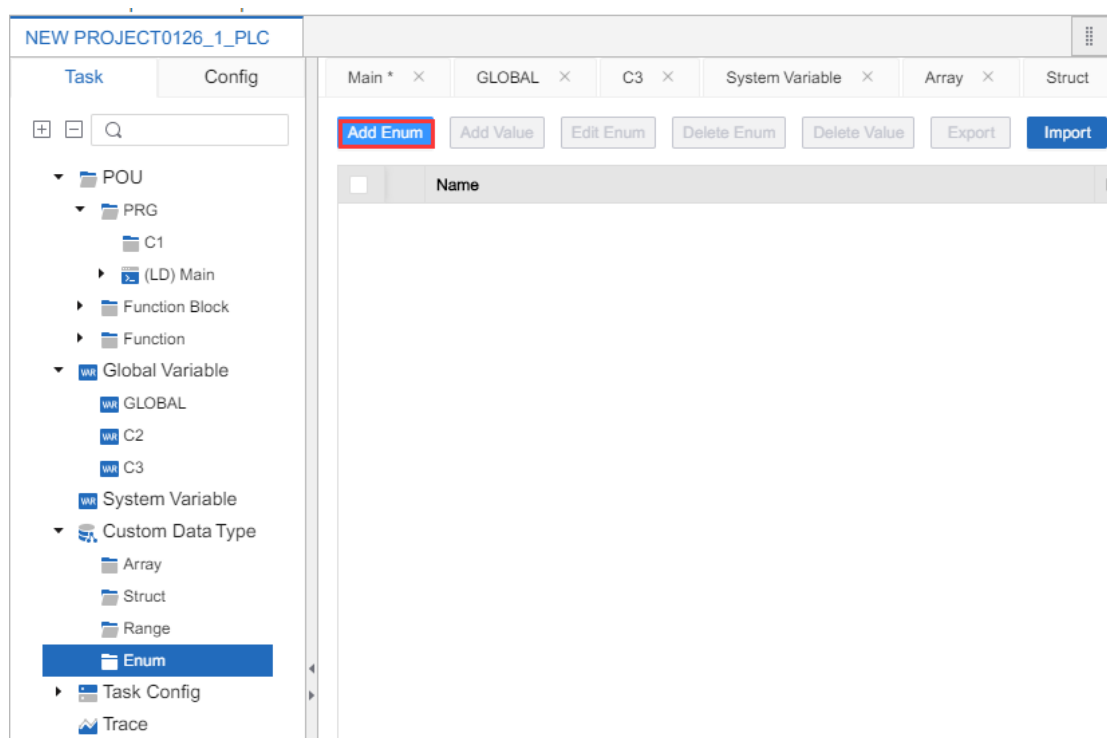
Parameter	Description
Name	Supports Chinese, English letters, numbers and the special character “_”, and cannot start with a number. Cannot be the same as the program organizational unit name, instruction name, or other custom data type name.
Value type	◆ INT: Integer , value range: - 32678~32677. ◆ SINT : Short integer , value range: - 128~127 . ◆ DINT: Double integer , value range: - 2147483648~2147483647.

Parameter	Description
	◆ USINT : Unsigned short integer, value range: 0 ~255. ◆ ULINT : Unsigned Long integer, value range: 0 ~ (2⁶⁴-1). ◆ UDINT : Unsigned double integer type, value range: 0 ~2147483647 .
Min	The minimum value of the range.
Max	The maximum value of the range.
Init	Initial value of range.

8.1.4.4 Enum

The operation method of adding a new enum is as follows:

Step75. Select **Custom Data Type/Enum** and click **Add Enum** .



Step76. Configure the relevant parameters in the pop-up dialog box and click **Confirm** .

Add Enum

×

* Name :

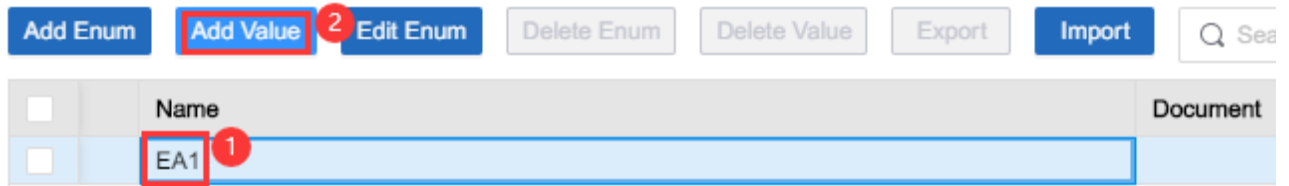
EA1

Desc :

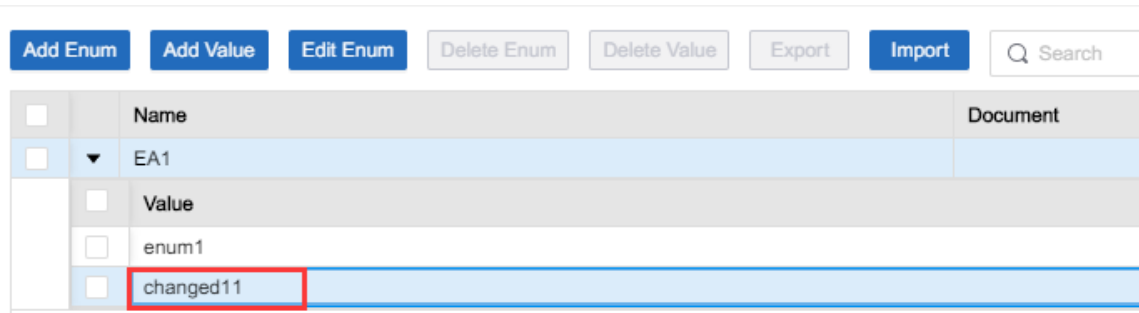
Cancel

Confirm

Step77. Select the enumeration and click **Add Value** .



Step78. Edit the enumeration value in the pop-up dialog box.

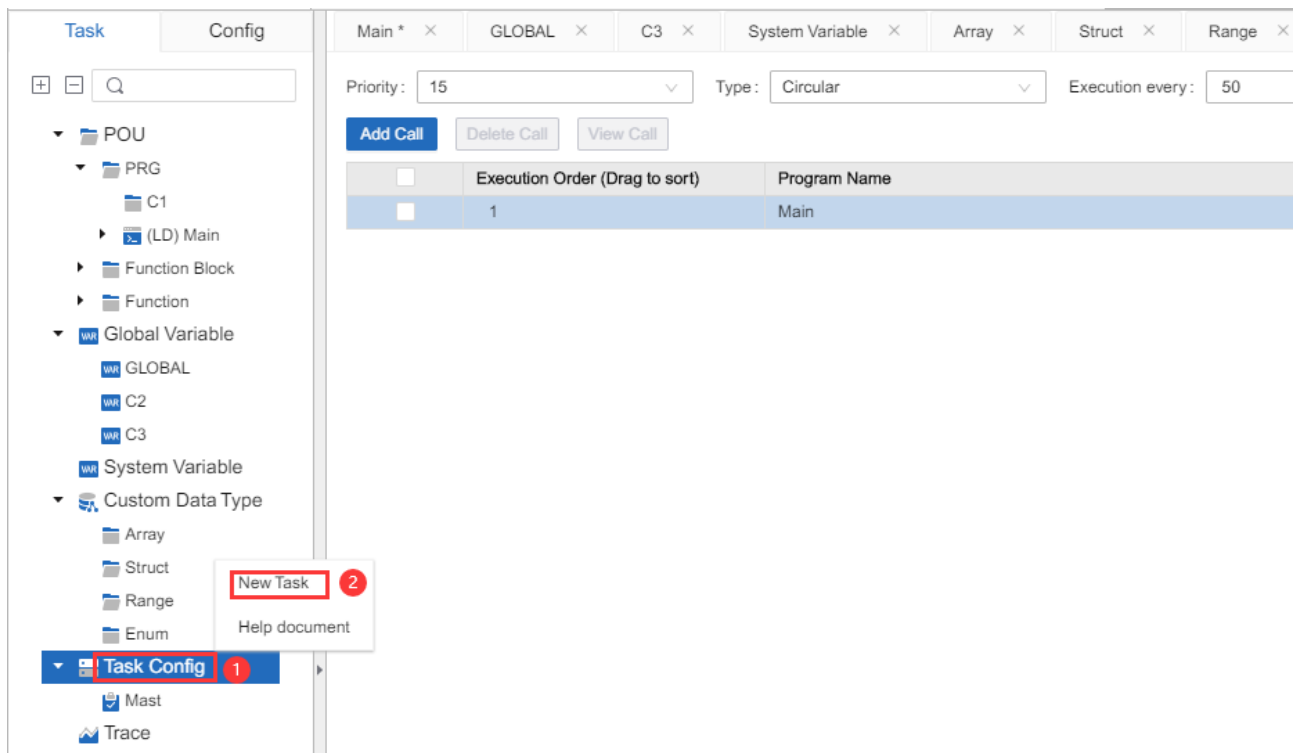


8.1.5 Task Configuration

Task refers to the sequence in which PLC executes programs. A task can contain multiple programs. Tasks include two types: circular and event. The built-in task of the project is Mast , which executes the Main program.

8.1.5.1 Create a New Task

Step79. Right-click **Task configuration** and select **New Task** in the pop-up menu.



Step80. Configure the relevant parameters in the pop-up dialog box and click **OK**.

New Task
✕

* Name :

* Priority :

Cancel
Confirm

Please refer to the table below for detailed configuration methods.

Parameter	Description
Name	Support English letters, numbers and the special character “_”, and cannot start with a number.
Priority	The value range is 0 ~31 . The smaller the value, the higher the priority.

8.1.5.2 Call Programs in a Task

Step81. Select the task and click **Add Call** .

Step82. Check the programs in the pop-up dialog box and click **Confirm** .

Add Call

	Name	Desc
☒	Main	
☒	AAS	

Step83. Drag a program in the list to adjust the order of execution of the program.

Main × GLOBAL × C3 × System Variable × Array × Struct × Range × Enum × Mast × first × AAS ×

Priority: 1 Type: Circular Execution every: milliseconds

	Execution Order (Drag to sort)	Program Name	Comment
☒	1	AAS ^{Main}	
☒	2	Main	

Step84. Set trigger conditions for tasks.

Priority: 1 Type: Circular Execution every: 55 milliseconds

	Execution Order (Drag to sort)	Program Name	Comment
☒	1	AAS	
☒	2	Main	

Please refer to the table below for detailed configuration methods.

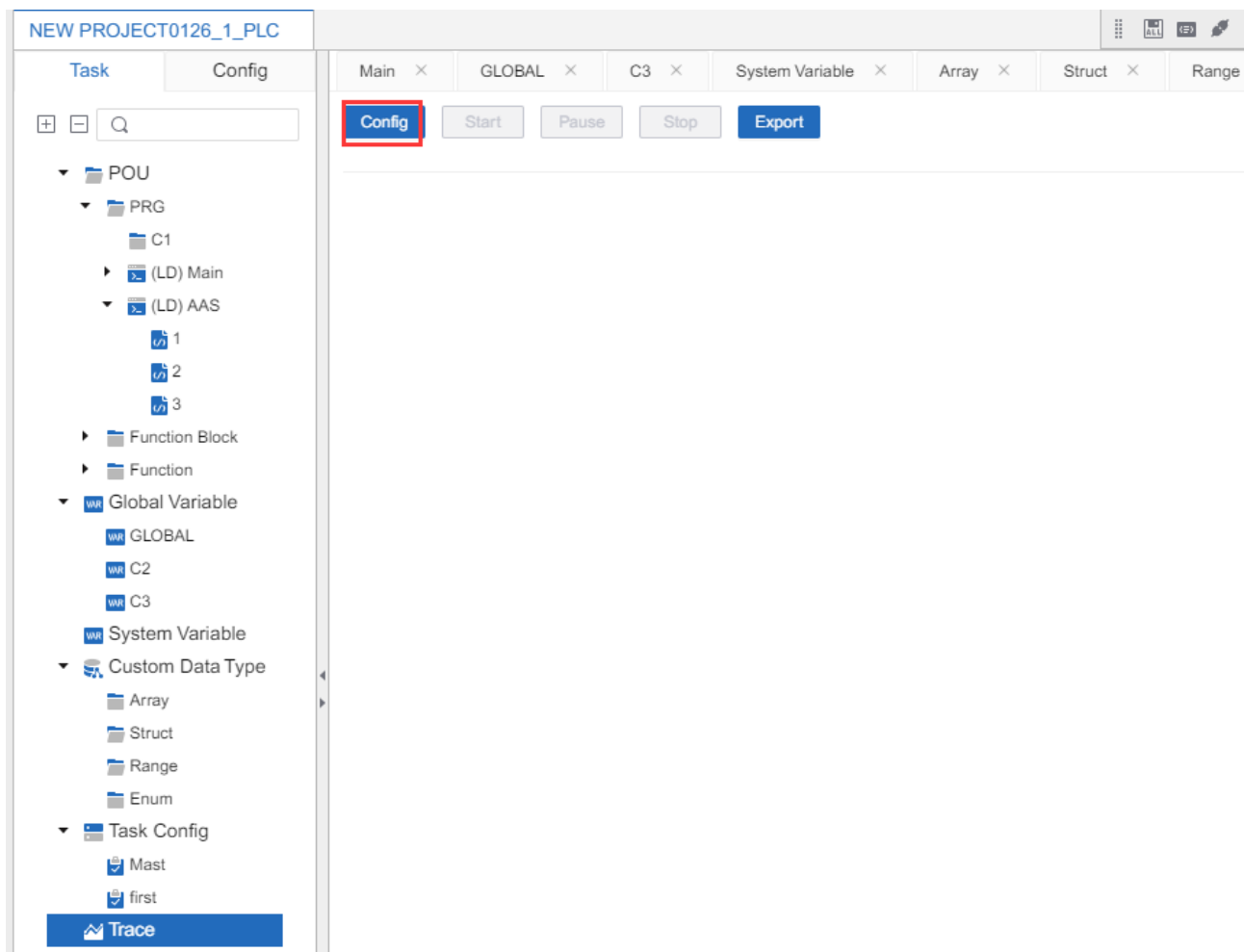
Parameter	Description
Type	◆ Circular: execute tasks cyclically and need to set the execution cycle. ◆ Event: Execute the task when the condition is met (when the rising/falling edge of the specified variable jumps).

8.1.6 Trace

The Trace chart displays the real-time values of variables in the form of a line chart. Before monitoring variables, the PLC needs to be in the online monitoring state (please refer to [Online Monitoring](#) for details).

The operation method of viewing the trace chart is as follows:

Step85. Select **Trace** and click **Config**.



Step86. Set the monitored variables in the pop-up dialog box, configure the parameters such as **Interval**, **Buffer Area Reserved Samples**, etc., and click **Confirm**.

Config

Delete

Import

Export

Monitor Variables

☐	Name	Data Type	Color	Y-axis minimum	Y-axis maximum	Comment
☐	GVL_Var4	BOOL	Orange			
☐	GVL_Var6	BOOL	Green			
☐						

Setting

* Interval(ms): 100

* Buffer Area Reserved Samples: 100

☐ Stop sampling when buffer full

☐ Multi-Y-axis display

> Advanced

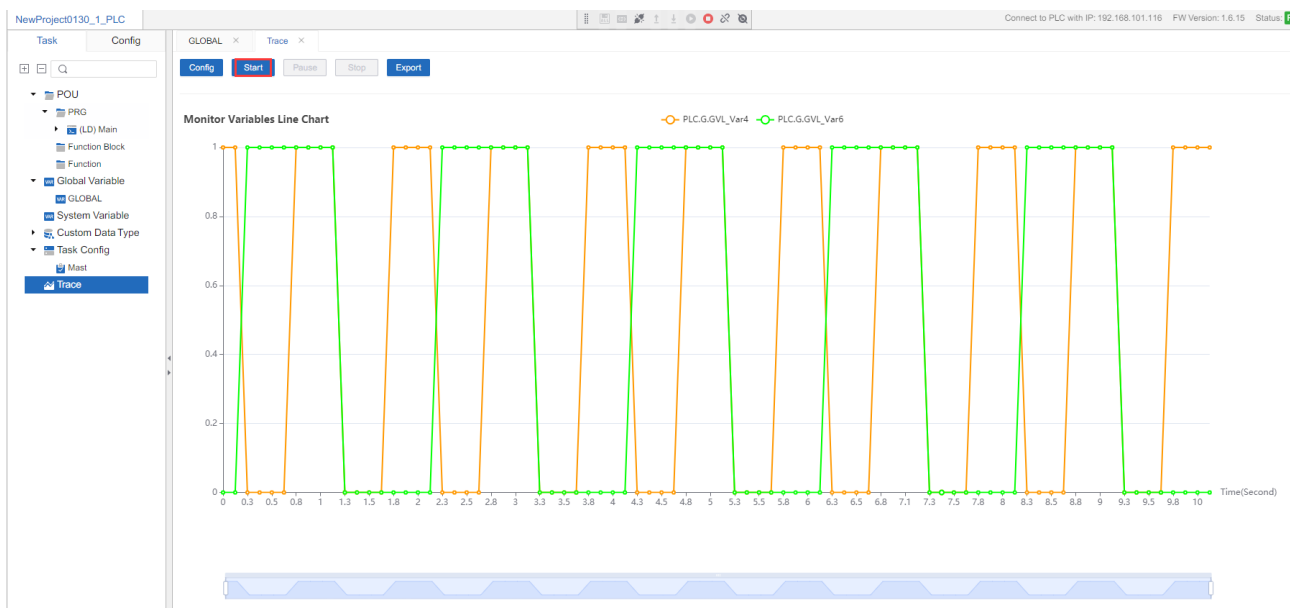
Cancel

Confirm

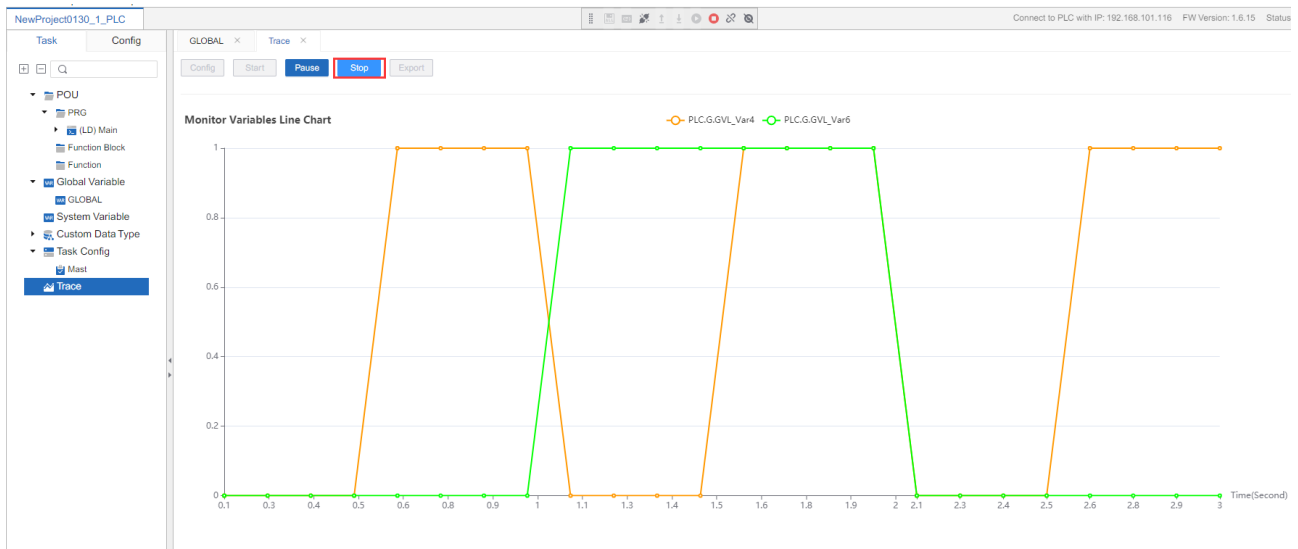
Please refer to the table below for detailed configuration methods.

Parameter	Description
Name	The name of the variable being monitored.
Color	The color of the monitored variable line chart.
Interval	Variable sampling interval. The range is 1~9007199254740991.
Buffer Area Reversed Samples	The number of sampled values saved. The range is 1~9007199254740991.
Stop sampling when buffer is full	When the number of samples in the buffer reaches the set number of retained samples, sampling is stopped.
Multiple Y- axis display	Separate Y -axes are used for different variables.
Trigger settings	Select the trigger variable and set the trigger condition (for example, the value of the trigger variable is greater than 10).

Step87. Click **Start** to start monitoring the value of the variable.

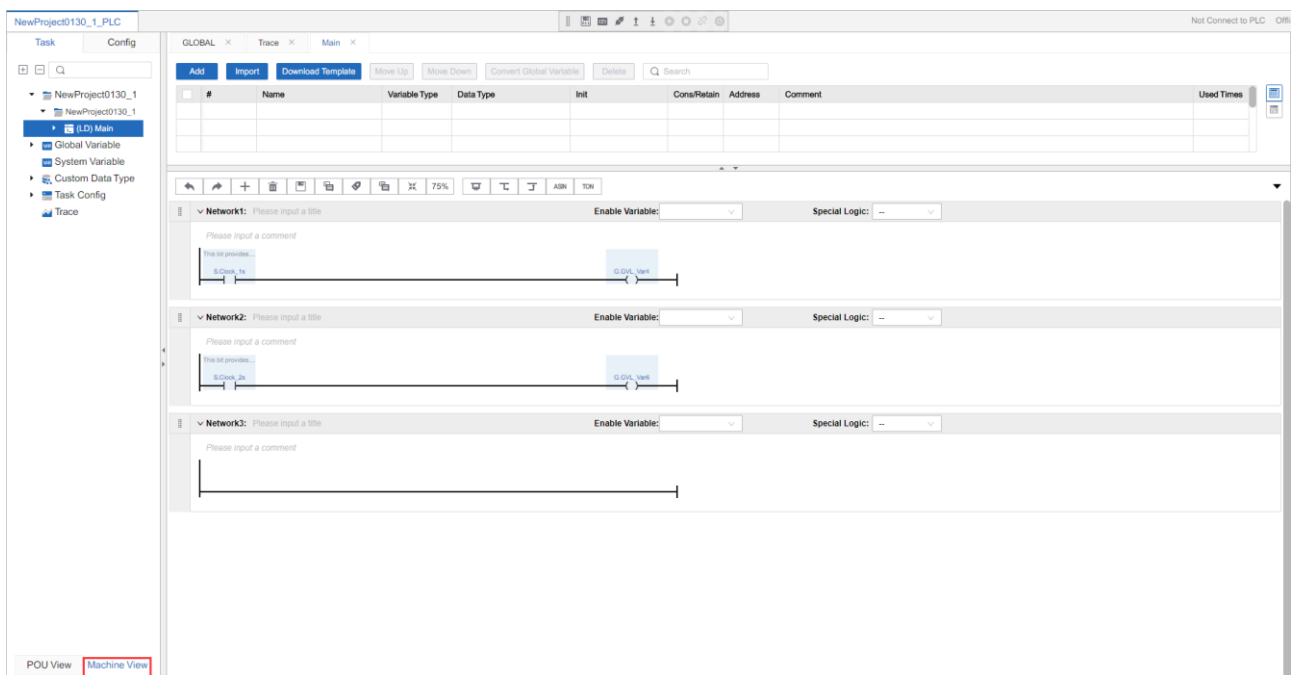


Step88. Click **Stop** to stop monitoring the value of the variable.



8.2 Machine View

Display PLC program and variable information in the Machine dimension, including the device's program organization unit, global variables, system variables, custom data types, task configurations, create folder classifications and monitoring trend charts. The operation method is the same as that in the POU View, please refer to [POU View](#) for details.



The **PLC** module is used to debug and maintain PLC. Including connection and debugging, project archiving, project comparison, project properties, restarting PLC, and restoring factory settings.

9.1 Connect and Debug

9.1.1 Compile

Compilation refers to converting project files into executable binary files to facilitate PLC recognition and execution of programs.

Select **PLC/Connect and Debug/Build** in the menu bar to compile the project file and output the compilation result information.

9.1.2 Connect PLC

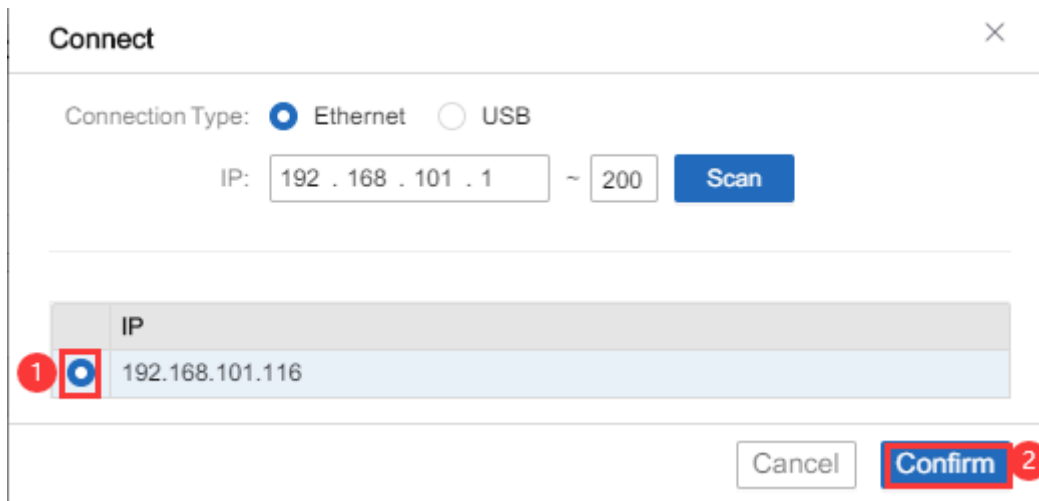
Step89. Use an Ethernet cable or USB data cable to connect the PC and PLC.



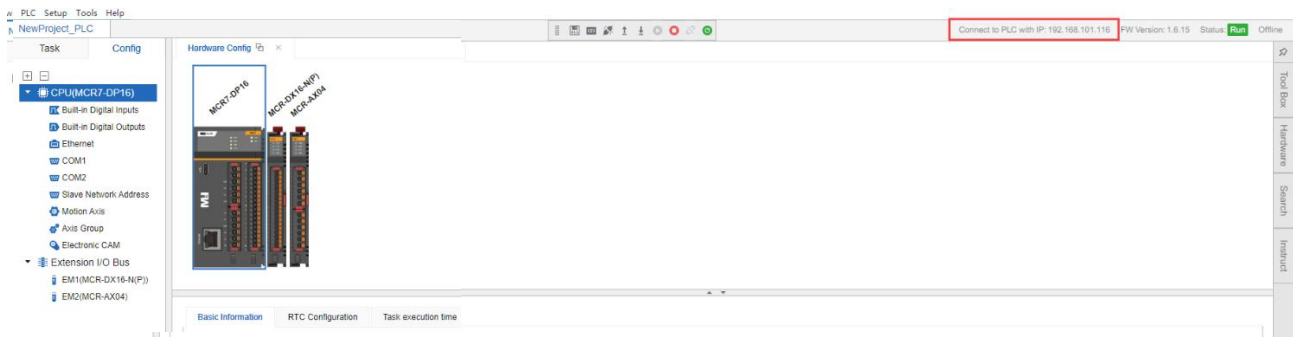
When using an Ethernet cable to connect the PC and PLC, please ensure that the IP addresses of the PC and PLC are in the same network segment.

Step90. Select **PLC/Connect and Debug/ Connect** in the menu bar, select the connection type (Ethernet or USB , this manual takes Ethernet as an example) in the pop-up dialog box, set the IP address range, and click **Scan** .

Step91. Select the Scanned PLC and click **Confirm** .



Step92. The result of successful PLC connection is shown in the figure below .



9.1.3 Disconnect PLC

You can disconnect the PLC after connecting it. The operation methods are as follows:

Select **PLC/ Connect and Debug/Disconnect** in the menu bar to disconnect the PLC .

9.1.4 Upload



Before uploading PLC files to PC, you need to ensure that the state of PLC is “Connected”.

PLC project files can be uploaded to PC. The operation methods are as follows:

Select **PLC/Connect and Debug/Upload** in the menu bar, configure the relevant parameters in the pop-up dialog box, and click **Confirm** .

Upload

×

Upload File Type:

☒ Project
☐ Config
☐ User File

Location:

C:\Users\94536\Desktop

...

☐ Automatically open the project after uploading completed

Cancel

Confirm

Please refer to the table below for detailed configuration methods.

Parameter	Description
Upload file type	◆ Project: PLC Project file. ◆ Configuration: PLC configuration file, mainly hardware configuration information.
Location	Set the path to upload files.

9.1.5 Download Project to PLC

PLC project source code or PLC system configuration (Ethernet) can be downloaded to PLC. The operate methods are as follows:

Step93. Select **PLC/ Connect and Debug/Download** in the menu bar, configure the relevant parameters in the pop-up dialog box, and click **Confirm** .

Download

×

Download Project Source code:

☐

Sync PLC Config(Ethernet):

☐

Project Encrypt:

☐

Project Pack Password:

Project Pack Password

🔑

Confirm Password:

Confirm Password

🔑

Remember Operate:

☐

Cancel

Confirm

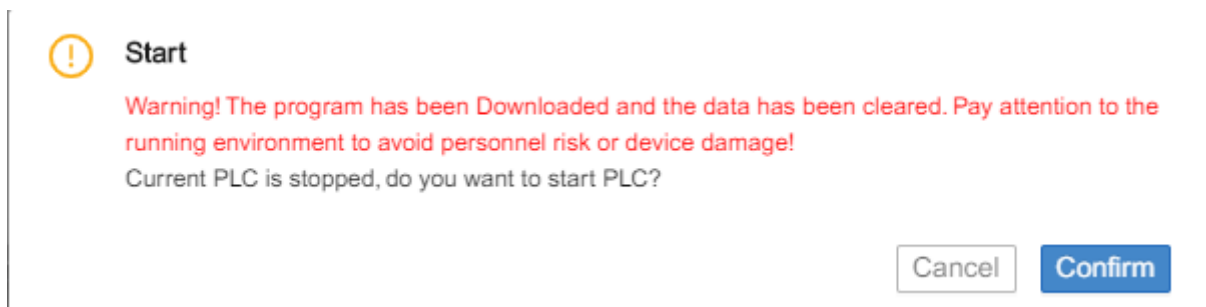
Please refer to the table below for detailed configuration methods.

Parameter	Description
Download project source code	Check this parameter to download the project source code to PLC.
Synchronize PLC configuration (Ethernet)	Check this parameter to download the PLC's Ethernet configuration information of the PLC.

Parameter	Description
Project Encrypt	Check this parameter to set a password for the project source code downloaded to the PLC. When uploading the project source code from the PLC to the PC and opening the project file, you need to enter the project packaging password for decryption. If you forget your password, you will not be able to open the project file.
Project Pack Password	Project file packaging password.
Remember Operate	Check Remember Operate to make this setting as the default parameter, and this configuration will be used by default when you perform the download operation in the future.

Step94. MCR Master will compile the project and prompt the download progress. After the download is completed.

- Click **Confirm** to switch the PLC to the running state.
- Click **Cancel** to switch the PLC to the stopped state.



9.1.6 Start Running

The running state of a PLC is that the PLC is running the program (that is, executing the program in the project file).

When the PLC is in the stopped state, select **PLC/Connect and Debug/Start** in the menu bar to switch the PLC to running state.

9.1.7 Stop Running

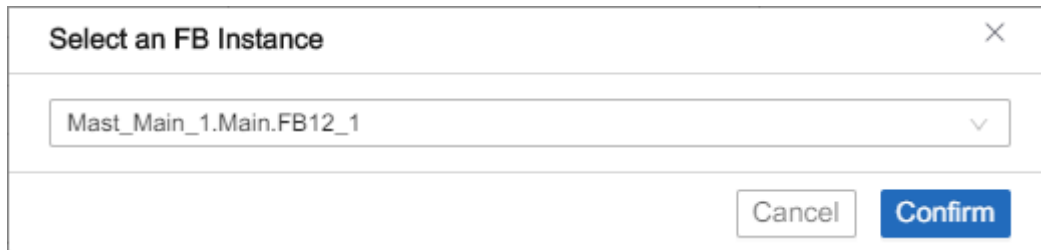
When the PLC is in running state, select **PLC/ Connect and Debug/Stop** in the menu bar to switch the PLC to Stop state. The PLC program will not be executed in Stop state.

9.1.8 Online Monitoring

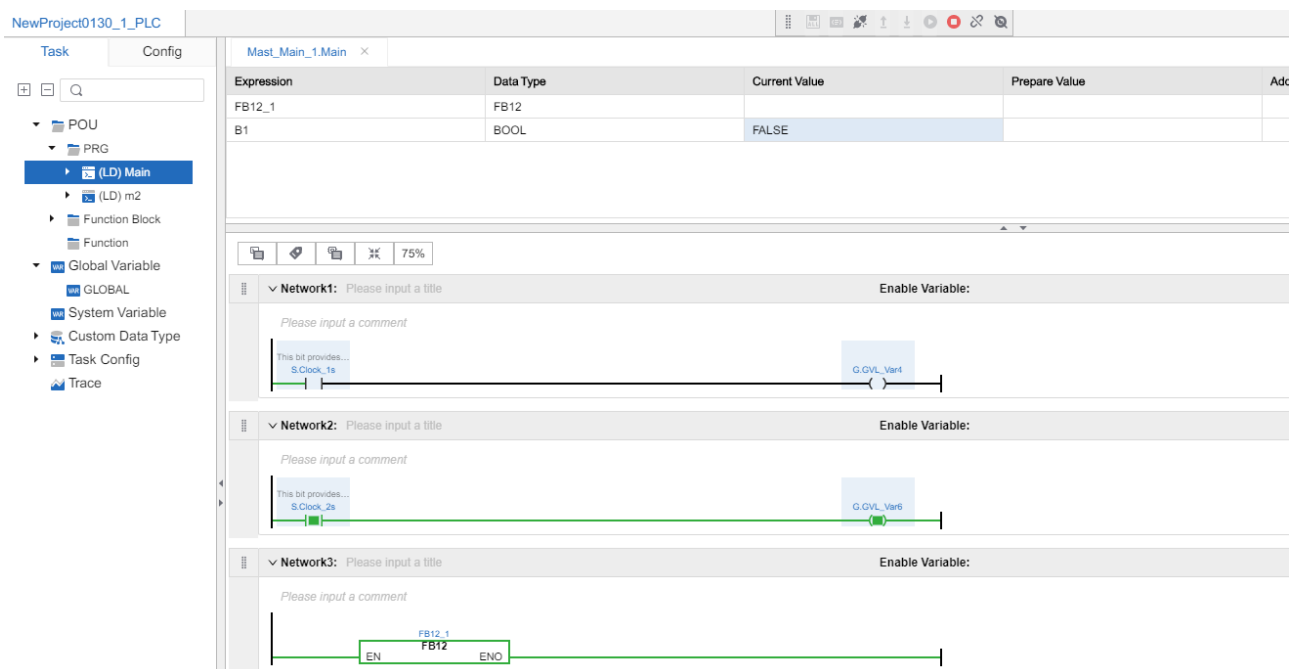
Online monitoring refers to real-time monitoring of the program running status and variable status of the PLC . Only when the status of the PLC is “running” can online monitoring be performed.

Step95. Select **PLC / Connect and Debug/Online** in the menu bar.

Step96. Select the task instance in the pop-up dialog box and click **Confirm**.

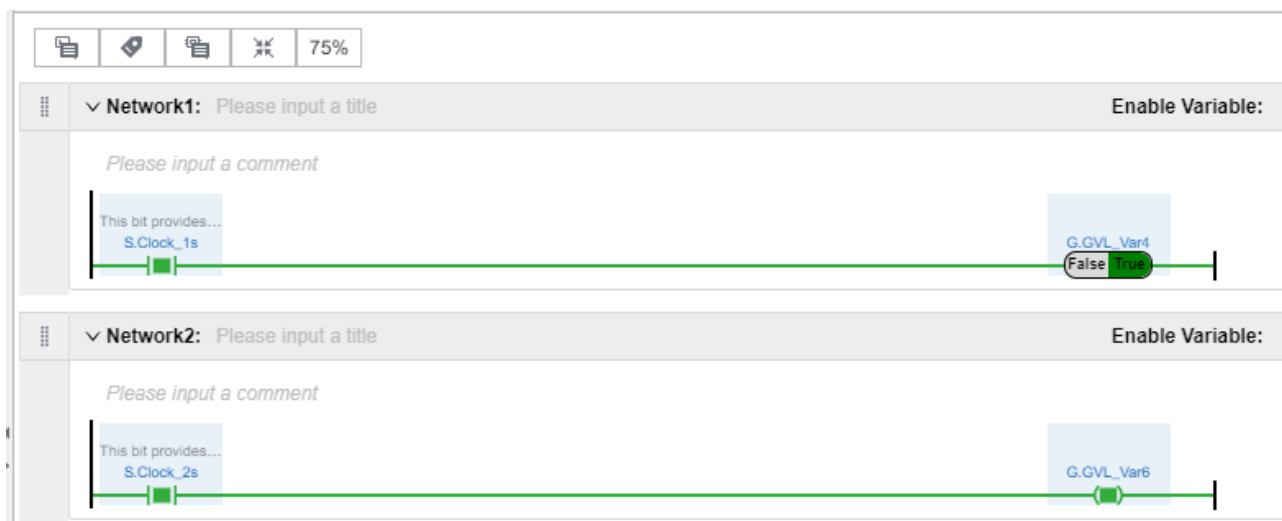


Step97. After entering the online monitoring state, you can view the real-time values of variables, contacts, and coils.



Expression	Data Type	Current Value	Prepare Value	Adc
FB12_1	FB12			
B1	BOOL	FALSE		

Step 4. Double-click the current value of a variable to force modify the value of the variable.



9.1.8.1 Prepare Value

In the online monitoring state, there is a **Prepare Value** column in the **variable (global/local)** table and monitor list, where you can prepare a value for the next forced variable. These values can be **forced** to be written by the shortcut key “F7”, or reset force by the shortcut key “Ctrl+F7” (specifically, forced to be written first, then unforced). Right-click on the value in the forced state and select **Reset Force** to unforce it, or you can unforce all forced variables with the shortcut key “Alt+F7”.

Expression	Data Type	Current Value	Prepare Value
FB12_1	FB12		
B1	BOOL	FALSE	TRUE

9.1.9 Offline

When the PLC is in the online monitoring state, select **PLC/ Connect and Debug/Offline** in the menu bar, set the options in the pop-up dialog box, and click **Confirm**. To enter offline editing state.

Project Data Option

☐ Backup the value of data type variables
 ☐ Retain the force state and value of variables

Option Description

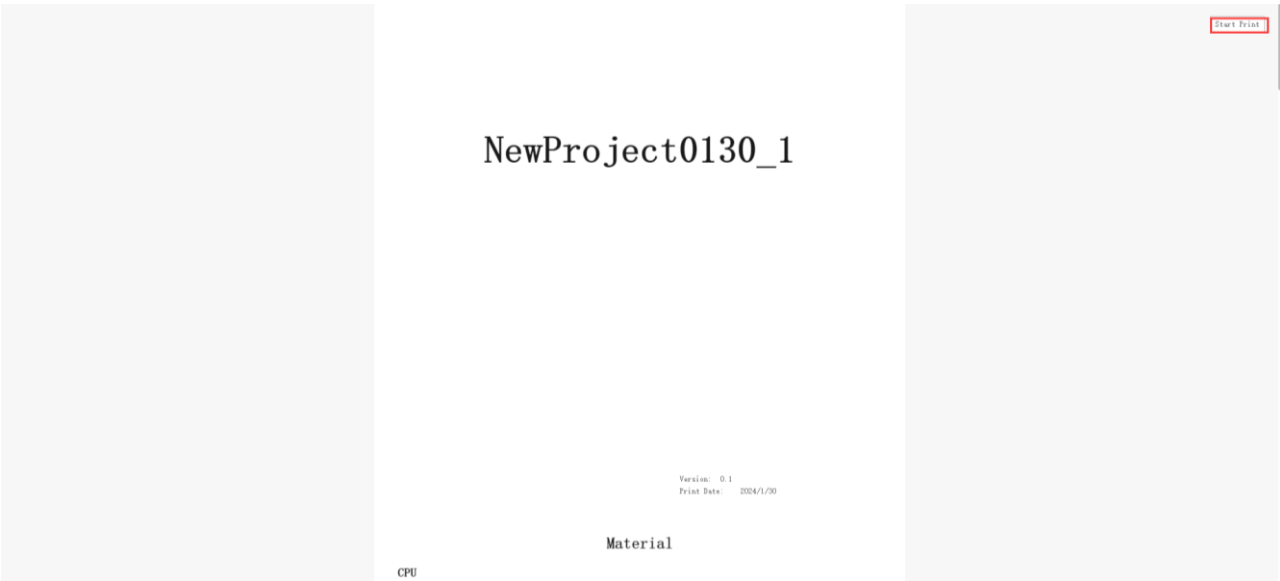
Backup the value of data type variables:
When exiting the monitoring state, the value of the data type variable is retained, which can be used for data recovery after power failure and restart or update program; The state of a Boolean variable is not retained.

Retain the force state and value of variables:
When exiting the monitoring state, the force state of the Boolean variable and the force value of the data variable are retained until the PLC power is restarted or the program is updated, and the force state disappears.

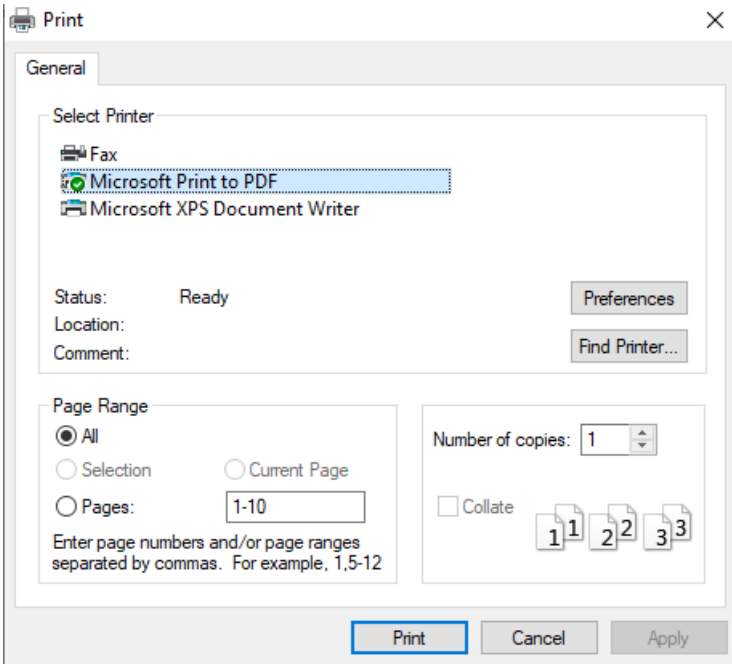
Cancel

Confirm

Refer to the table below for detailed configuration method.



Step99. Configure the relevant information in the pop-up dialog box and click **Print**.



Please refer to the table below for detailed configuration methods.

Parameter	Description
Select Printer	Select the printer to use or click Find Printer to add a new printer. It is necessary to ensure that the PC and printer are connected to the network.
Page Range	◆ All pages. ◆ Custom page range. For example: 1-2, 5.
Preferences	Click Preferences to set the paper size and orientation, that is, the print quality.

9.3Project Comparison

Project Comparison means that you can compare the differences between the current project and the target project, and the difference information mainly includes version, POU, task, hardware configuration, etc. The operation method is as follows:

Step100. Select **PLC/ Comparison** in the menu bar, configure relevant information in the pop-up dialog box, and click **Comparison**.

Comparison

Type: ☒ Local Project ☐ PLC Project

Project:

Projects with large version differences cannot be compared

Please refer to the table below for detailed configuration methods.

Parameter	Description
Type	◆ Local project: The project of the local PC . ◆ PLC project : MCR Master connected PLC project .
Project	Select the path where the local project file is located.

Step101. View comparison information.

Comparison

×

Type : ☒ Local Project ☐ PLC Project

Project :

Projects with large version differences cannot be compared

	Comparison Items	Comparison Results									
+	Version										
-	POU										
	<table> <thead> <tr> <th>Comparison Items</th> <th>Source Project</th> <th>Target Project</th> <th>Comparison Results</th> </tr> </thead> <tbody> <tr> <td>Main</td> <td>Exist</td> <td>Exist</td> <td>Inequality</td> </tr> </tbody> </table>	Comparison Items	Source Project	Target Project	Comparison Results	Main	Exist	Exist	Inequality		
Comparison Items	Source Project	Target Project	Comparison Results								
Main	Exist	Exist	Inequality								
	Global Variable	Equality									
	Custom Type List	Equality									
-	Task										
	<table> <thead> <tr> <th>Comparison Items</th> <th>Source Project</th> <th>Target Project</th> <th>Comparison Results</th> </tr> </thead> <tbody> <tr> <td>Mast</td> <td>Exist</td> <td>Exist</td> <td>Inequality</td> </tr> </tbody> </table>	Comparison Items	Source Project	Target Project	Comparison Results	Mast	Exist	Exist	Inequality		
Comparison Items	Source Project	Target Project	Comparison Results								
Mast	Exist	Exist	Inequality								
[-]	Hardware Orchestration										
	<table> <thead> <tr> <th>Comparison Items</th> <th>Source Project</th> <th>Target Project</th> <th>Comparison Results</th> </tr> </thead> <tbody> </tbody> </table>	Comparison Items	Source Project	Target Project	Comparison Results						
Comparison Items	Source Project	Target Project	Comparison Results								

9.4 Modify project property

Select **PLC/Project Property** in the menu bar, modify the relevant information in the pop-up dialog box, and click **Confirm** to modify the project property.

Project Property

* Name :

Location :

PLC Model/Hardware Version/Project Version :

Author :

Desc :

9.5 Restart PLC



Service is unavailable during PLC restart, please operate with caution.

Select **PLC/Restart PLC** in the menu bar and click **Confirm** in the pop -up dialog box to restart PLC.

9.6 Restore Factory Settings



Restoring the factory settings will delete the project and configuration data in the PLC and restore it to the factory state. Please operate with caution.

Select **PLC/Restore Factory Settings** in the menu bar and click **Confirm** in the pop-up dialog box to restore PLC to factory settings.

10 Setup

Setup refers to the configuration of the MCR Master's operating parameters to meet the user's individual needs.

10.1 Switch System Interface Language

Select **Setup/Switch Language/English** in the menu bar and click **Confirm** in the pop-up dialog box to switch the system interface language to English. Select **Setup /Switch Language / Chinese** on the menu click **Confirm** in the pop-up dialog box to switch the system interface language to Chinese.

10.2 Display Setting

Select **Setup/Display Settings** in the menu bar, configure the relevant parameters in the pop-up dialog box, and click **Confirm**.

The screenshot shows a 'Display Setting' dialog box with a close button (X) in the top right corner. It contains two settings:

- LD Cell Width :** A slider with three positions: 1X, 2X, and 3X. The slider is currently positioned at 1X.
- LD Variable Font-Size :** A slider with five positions: 8, 10, 12, 14, and 16. The slider is currently positioned at 12.

At the bottom right, there are two buttons: 'Cancel' and 'Confirm'. The 'Confirm' button is highlighted in blue.

11 Upgrade PLC Firmware

PLC firmware is the software responsible for the most basic and bottom-level work of PLC, which is equivalent to the operating system of PLC. Therefore, the firmware also determines the functions and performance of PLC equipment.



Before upgrading the PLC firmware, you need to ensure that the PLC is in “Offline” state.

The operation method of upgrading PLC firmware is as follows:

Step102. Select **Tools/Update Firmware** in the menu bar, configure the relevant parameters in the pop-up dialog box, and click **Confirm**.

Update Firmware

Model :

Current Firmware Version : 1.5.3

SN Number :

Once fireware is updated, the PLC will be restarted automatically, please reconnect PLC manually

☐ Update

☒ Online Upgrade

☐ Download To Local

☐ Upgrade extend modules

1.6.15

1.Fix the issue of inaccurate PLC RTC in certain situations 2.Optimize the sampling logic of the temperature module

Cancel

Confirm

Please refer to the table below for detailed configuration methods.

Parameter	Description
Update	Upgrade PLC's firmware by offline files. Please contact MX On CORPORATION technical support engineer to obtain the PLC firmware file.
Online upgrade	Connect to the cloud server to upgrade the PLC firmware. Only versions higher than the current firmware version are displayed.
Download to Local	Download the PLC firmware file from the cloud server to the local computer.

- ◆ After updating the firmware, the system will prompt “Please reconnect the PLC” and the PLC needs to be reconnected .



Firmware update succeeded, PLC has been restarted, please reconnect PLC

OK

12 Help

Help is used to guide users to configure and maintain PLC. Including logs, PLC logs, disk management, viewing MCR Master help documents, checking for updates, and viewing software version information.

12.1 Logs

Log refers to recording information of MCR Master event information, such as project compilation errors, normal operating processes, etc.

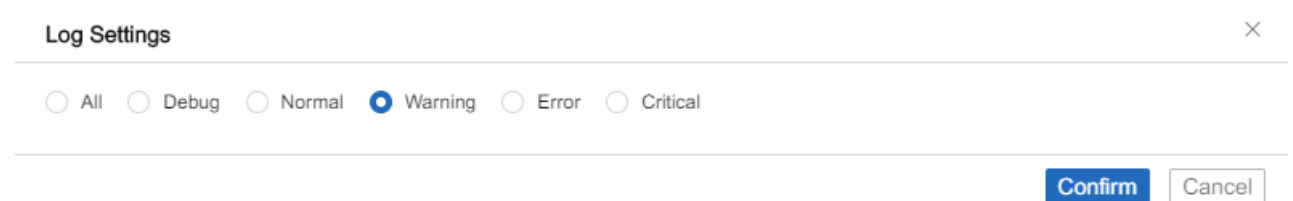
12.1.1 Set the Save Path of Logs

Select **Help/Logs/Collect Logs** in the menu bar, click  icon in the pop-up dialog box, select the location, and click **Confirm** to set the log save path.



12.1.2 Set the Level of collected logs

Select **Help/Log/Log Settings** in the menu bar, select the log level in the pop-up dialog box, and click **Confirm**. You can set the log information that needs to be recorded at the corresponding level.



The levels of the logs, from lowest to highest, are Debug, Normal, Warning, Error, and Critical. When a specific level is selected, the system saves logs of the corresponding level and higher (e.g., when Warning is selected, log messages with levels of Warning, Error, and Critical are saved). Select **All** to save log messages for all levels.

12.2 PLC Logs

The PLC logs is used to record PLC event information, mainly including hardware failures, firmware failures, etc.

12.2.1 Collect PLC logs

Select **Help/ PLC Logs/Collect PLC Logs** in the menu bar, set the PLC log saving path in the pop-up dialog box, and click **Confirm** .



The dialog box titled "Collect PLC Logs" has a close button (X) in the top right corner. Below the title bar, there is a label "Location :" followed by a text input field and a button with three dots "...". At the bottom right, there are two buttons: "Cancel" and "Confirm".

12.2.2 Clear PLC logs



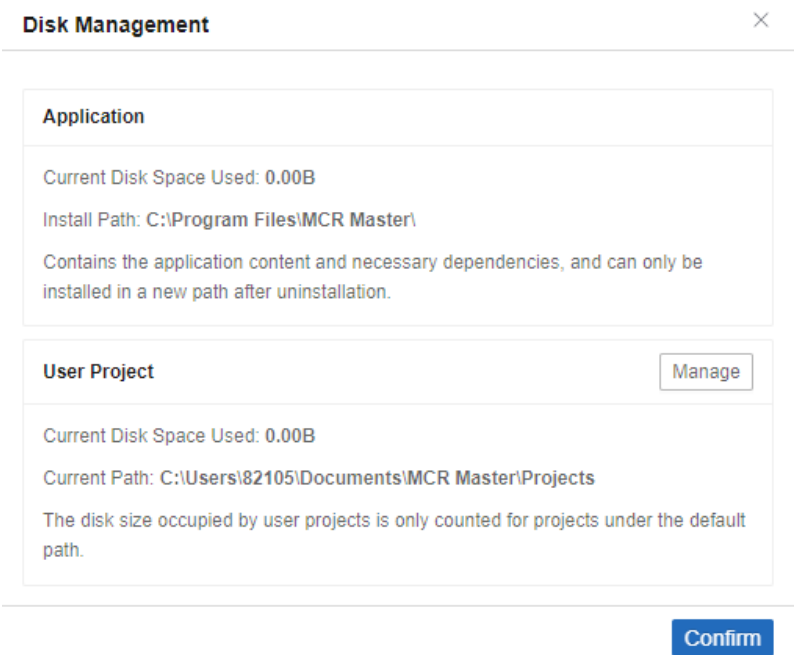
After clearing the PLC log, the log data cannot be recovered, so please operate with caution.

Select **Help/ PLC Log/Clear PLC Logs** in the menu bar and click **Confirm** in the pop-up dialog box to clear the PLC logs .

12.3 Disk Management

The operation method of disk management is as follows:

Step103. Select **Help/Disk Management** from the menu bar.



The dialog box titled "Disk Management" has a close button (X) in the top right corner. It contains two main sections: "Application" and "User Project".

Application

Current Disk Space Used: 0.00B

Install Path: C:\Program Files\MCR Master\

Contains the application content and necessary dependencies, and can only be installed in a new path after uninstallation.

User Project Manage

Current Disk Space Used: 0.00B

Current Path: C:\Users\82105\Documents\MCR Master\Projects

The disk size occupied by user projects is only counted for projects under the default path.

At the bottom right, there is a "Confirm" button.

Step104. Click **Manage** behind **User Project**, set a new location in the pop-up dialog box, and click **Edit** to modify the default storage location of the project. Existed projects will not be affected.

User ProjectManage

Current Location

C:\Users\82105\Documents\MCR Master\Projects

New Location

C:\Users\82105\Documents\MCR Master\Projects

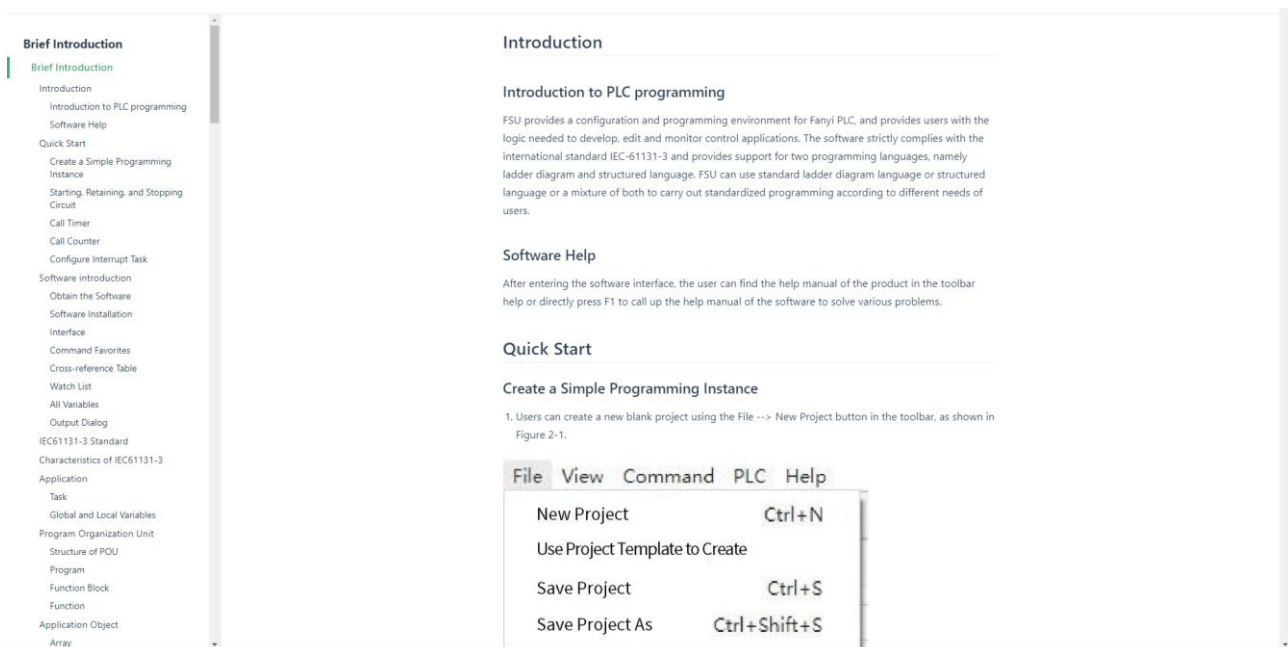
Note: Only modify the default storage location of the project, existing projects will not be affected.

Edit

Cancel

12.4 View the Help Document of MCR Master

Select **Help/Help** on the menu bar to view the MCR Master help document and get the MCR Master configuration method.



12.5 Check for Updates

Select **Help/Check for Updates** in the menu bar, and MCR Master will automatically check whether there is a new version of software on the cloud server. If there is a new version of software, you can update the software version, otherwise it will prompt “You are currently using the latest version”.

12.6 View Software Version Information

Select **Help/About** in the menu bar to view the current software version information.

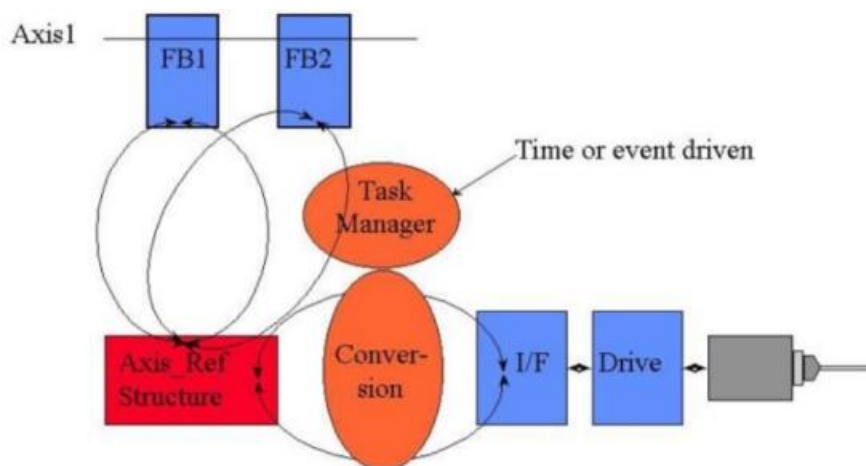
13 Motion axis

13.1 Overview

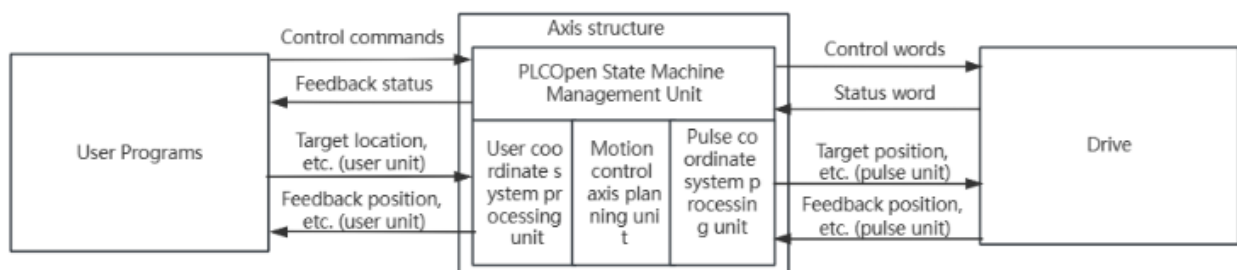
MCR7 series PLCs support motion control function. Motion functions operate according to the axis state diagram and are used to modify the motion of the axis. These function blocks can return state to the application while in motion. Applications use these status bits to determine movement state (Done, Busy, Active, CommandAborted , and Error). The current status of the axis can be read and determined using the MC_ReadStatus instruction.

13.1.1 The Composition and Control Logic of the Motion Axis

In a motion control system, objects that are controlled in motion are referred to as axis. The axis is the bridge between the driver and PLC instructions. The motion control axis is used to control local high-speed pulse output and high-speed pulse input in compliance with the CiA402 protocol.



Inside the PLC, the basic structure and processing logic of the axis are shown in the figure below.



13.1.2 Type of Motion Axis

The axis types supported by MCR7-DN(P)16 include local pulse axis and local encoder axis.

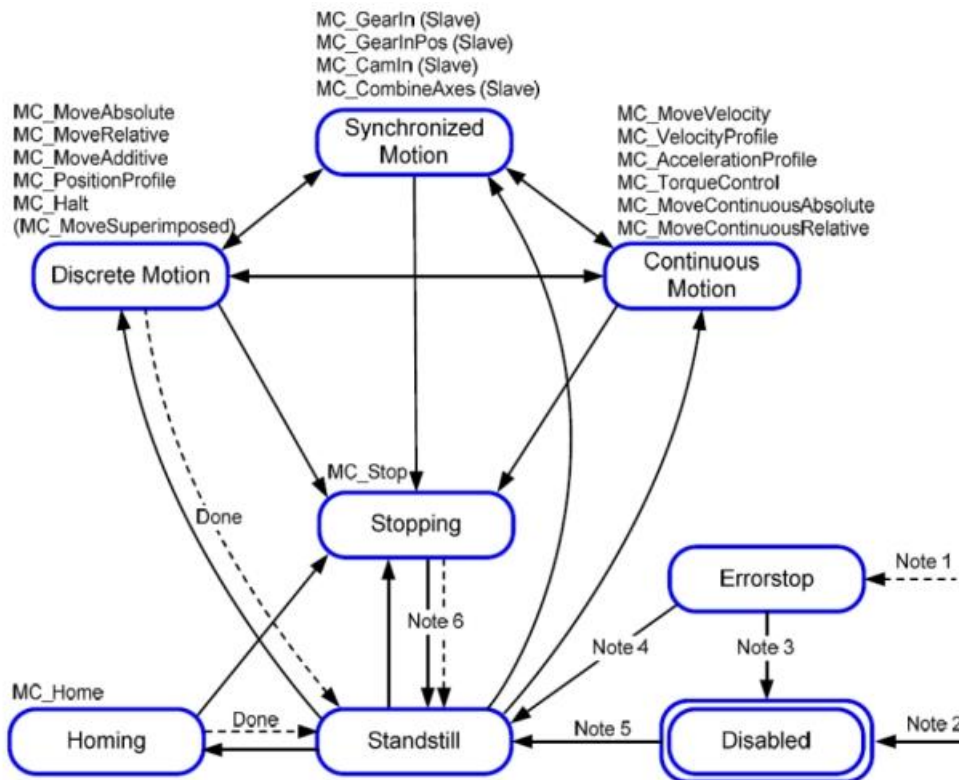
- ◆ Local pulse axis: An axis controlled by a pulse driver using local high-speed output control. Each pulse output channel can be optionally set to “pulse + direction”. The local pulse axis supports relative motion/absolute

motion, speed movement, home return and other basic modes of control, but does not support torque mode. Including pulse axis and PWM two modes.

- ◆ Local encoder axis: Pulse axis collected using local high-speed input.

13.2 Operating Mode

Based on the PLCOpen state machine, the state and motion of the motion axis are managed, and different functions are completed in each different state. The state transition is shown in the figure below.



Please refer to the table below for the meaning of each state.

Parameter	Description
Disabled	The initial disable state does not allow any motion instructions to be executed.
Errorstop	Fault shutdown state, highest priority, applies when an error is detected on the axis or in the controller. Any movement in progress will be aborted and decelerated to a stop according to the Quick Stop deceleration configuration parameter. The Error output is set on the used function block. The ErrorStop state remains as long as the error is pending . No other motion instructions are accepted until reset with MC_Reset is complete.
Standstill	Enabled state, power is on, no errors are detected, and no motion instructions are active on the axis. Allows execution of motion instructions.
Homing	Return to zero state.

Parameter	Description
Stopping	Stopped state.
Discrete Motion	Discrete motion states.
Continuous Motion	Continuous motion state.
Synchronized Motion	Synchronized motion state.

Please refer to the table below for the conditions for state transition.

Item	Transition conditions
Note 1	When the fault detection logic of the axis detects a fault, it immediately enters the Errorstop state.
Note 2	When the axis has no fault and MC_Power.Enable =FALSE, it enters the Disabled state.
Note 3	The current state is the Errorstop state. When MC_Reset is called to reset the axis fault and MC_Power.Status =FALSE, it enters the Disabled state.
Note 4	The current state is Errorstop state. When MC_Reset is called to reset the axis fault and MC_Power.Status =TRUE, it enters Standstill state.
Note 5	The current state is Disabled state. When MC_Power.Enable =TRUE and MC_Power.Status =TRUE, it enters Standstill state.
Note 6	The current state is Stopping state. When MC_Stop.Done =TRUE and MC_Stop.Execute =FALSE, it enters Standstill state.

13.3 Buffering Mode

Some motion function blocks have an input called BufferMode. Using this input, the function block can start immediately or enter a buffer to start.

The available options are defined in the enumeration of type MC_BUFFER_MODE:

- ◆ An aborted movement (mcAborting) will start immediately, aborting any movement in progress and clearing the movement queue.
- ◆ Buffered movements (mcBuffered) are queued, that is, attached to any movement currently executing or waiting to be executed, and will start after the previous movement completes.

The buffer can only contain 1 motion function block.

The execution conditions of the motion function blocks existing in the buffer are: when the current continuous motion is InVelocity , or when the current discontinuous motion stops.

Clears the motion queue (delete all buffered motions) when:

- ◆ When an aborted move is triggered (mcAborting), the CommandAborted pin is set in the buffered function block .
- ◆ Executing the MC_Stop function, the Error pin is set in the cleared buffer function block.
- ◆ A transition to the ErrorStop state is detected, the Error pin is set in the buffered function block.



- ◆ Only valid motion can be queued. If the execution of the function block terminates because the Error output is set, the motion is not enqueued, any motion currently executing is not affected, and the queue is not cleared.
- ◆ When the queue is full, the Error output is set in the corresponding function block .

The pulse output channel can respond to new commands when executing the current command (the current command has not yet been executed). Please see the table below for details. “allow” means that the new command will start executing even if the previous command has not yet completed execution. “deny” means the new command will be ignored with an “Error detected” prompt.

Current command	Next command					
	Home	MoveVelocity	MoveRelative	MoveAbsolute	Halt	Stop
Stand still	allow	allow*	allow*	allow*	allow	allow
Home	deny	deny	deny	deny	deny	allow
MoveVelocity	deny	allow	allow	allow	allow	allow
MoveRelative	deny	allow	allow	allow	allow	allow
MoveAbsolute	deny	allow	allow	allow	allow	allow
Halt	deny	allow	allow	allow	allow	allow
Stop	deny	deny	deny	deny	deny	deny



- ◆ Meaning of the “*” after “allow”: Buffering mode mcAborting / mcBuffered behaves the same way (starts movement immediately) if the axis is stationary.
- ◆ When an error is detected during motion conversion, the axis will enter the ErrorStop state and Error is set to TRUE .

13.4 High-speed I /O Hardware

The MCR7 series PLC (transistor output models only) features high-speed input points and output points. The MCR7-DN(P)16 integrates 8 high-speed input points and 8 high-speed output points. These high-speed inputs and outputs support the following dedicated functions:

- ◆ High-speed counter: it can perform fast counting of pulses from sensors, switches, etc. connected to fast inputs. For details about high-speed counters, refer to [Counting axis](#).

- ◆ Pulse output: it is divided into PTO mode and PWM mode. In open loop mode, the PTO function generates a pulse train output to control a stepper driver or servo driver for applications such as positioning. The PWM function can generate a square wave signal with a variable duty cycle on a dedicated output channel for applications such as heating control.

High-speed I/O hardware address mapping is as follows:

- ◆ High speed input

	Input address	Counting mode				
		AB	ABZ		CW/CCW	pulse + direction
Counting channel	I X0.0	A	A		CW	pulse
	IX0.1	B	B		C CW	direction
Counting channel	I X0.2	A		A	CW	pulse
	IX0.3	B		B	C CW	direction
Counting channel	I X0.4		Z	Z		Pulse (direction can be specified as any input point)
Counting channel	I X0.5					Pulse (direction can be specified as any input point)
Counting channel	I X0.6					Pulse (direction can be specified as any input point)

- ◆ High speed output

	Output address	Output mode	
		pulse + direction	PWM
Pulse channel	Q X0.0	Pulse (the direction can be specified as any output point)	PWM pulse
Pulse channel	Q X0.1	Pulse (the direction can be specified as any output point)	PWM pulse
Pulse channel	Q X0.2	Pulse (the direction can be specified as any output point)	PWM pulse

		point)	
Pulse channel	Q X0.3	Pulse (the direction can be specified as any output point)	PWM pulse
Pulse channel	Q X0.4	Pulse (the direction can be specified as any output point)	PWM pulse
Pulse channel	Q X0.5	Pulse (the direction can be specified as any output point)	PWM pulse

◆ MCR7 high speed input

	Input address	Counting mode			
		AB	ABZ	CW/CCW	Pulse+direction
Counting Channel	IX0.0	A	A	CW	Pulse
Counting Channel	IX0.1	B	B	CCW	direction
Counting Channel	IX0.2	A	Z	CW	Pulse
Counting Channel	IX0.3	B		CCW	direction
Counting Channel	IX0.4	A	A	CW	Pulse
Counting Channel	IX0.5	B	B	CCW	direction
Counting Channel	IX0.6	A	Z	CW	Pulse
Counting	IX0.7	B		CCW	direction

	Input address	Counting mode			
		AB	ABZ	CW/CCW	Pulse+direction
Channel1					

◆ MCR7 high speed output (only MCR7-DN(P)16 support high speed output)

	Output address	Output mode	
		Pulse+Direction	PWM
Pulse channel	QX0.0	Pulse (Direction can be assigned to any output point)	PWM pulse
Pulse channel	QX0.1	Pulse (Direction can be assigned to any output point)	PWM pulse
Pulse channel	QX0.2	Pulse (Direction can be assigned to any output point)	PWM pulse
Pulse channel	QX0.3	Pulse (Direction can be assigned to any output point)	PWM pulse
Pulse channel	QX0.4	Pulse (Direction can be assigned to any output point)	PWM pulse
Pulse channel	QX0.5	Pulse (Direction can be assigned to any output point)	PWM pulse
Pulse channel	QX0.6	Pulse (Direction can be assigned to any output point)	PWM pulse
Pulse channel	QX0.7	Pulse (Direction can be	PWM pulse

		assigned to any output point)	
--	--	-------------------------------	--

When the high-speed input and output points are configured as dedicated functions, they cannot be used as ordinary input and output points. If configured as ordinary input and output points, MCR Master will prompt “The current hardware port is already occupied”.

Refer to the table below for the characteristics of high-speed I/O .

Characteristic	Description
Range	-2,147,483,648 ~ 2,147,483,647 (32 bits)
Maximum frequency	100kHz
Minimum step size	1Hz
Maximum value of acceleration and deceleration	100kHz/ms
Minimum value of acceleration and deceleration	1Hz/ms
Speed accuracy	0.5 %

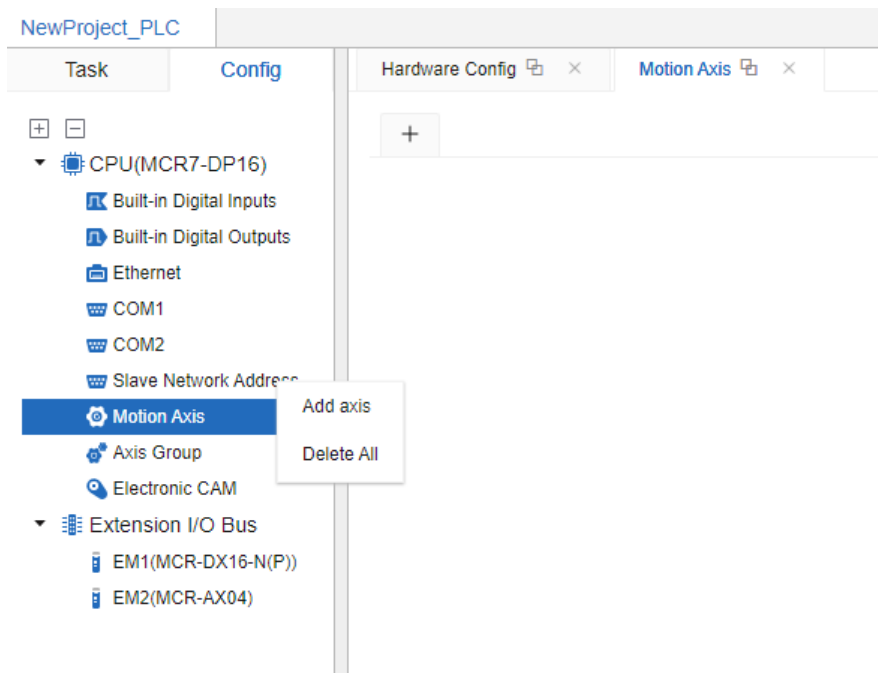
13.5 Create Motion Axis

13.5.1 Pulse Axis

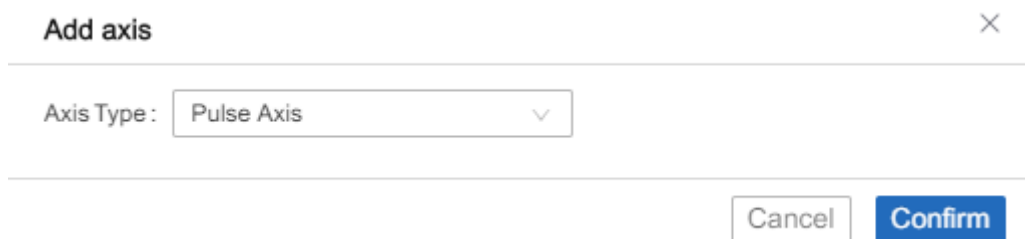
The working principle of the pulse axis is mainly to control the rotation angle and rotation speed of the motor through the pulse signal of the controller. Specifically, the controller controls the motor to rotate a certain angle or number of revolutions by sending a certain number of pulse signals to the motor, thereby achieving the movement of the shaft.

Pulse axes are used in a wide scope of applications, such as in robots, automation equipment, printers, CNC machine tools and other fields. In short, the pulse axis is widely used in various automation equipment and robots, and various precise motion controls can be achieved by controlling the pulse axis .

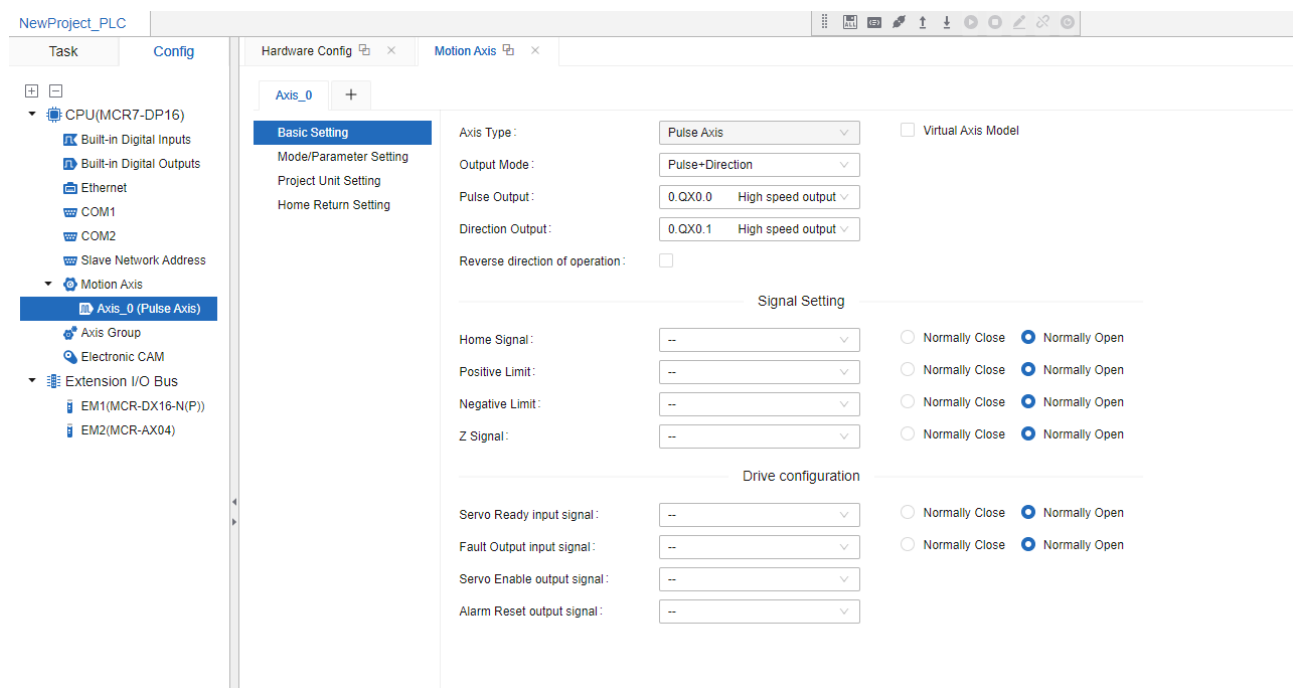
Step105. Right-click the **Motion Axis** and select **Add Axis** in the pop-up menu .



Step106. Select **Pulse Axis** in the pop-up dialog box and click **Confirm**.



Step107. After creating the pulse axis, the axis will be displayed and the **basic setting** parameters can be configured.



Please refer to the table below for detailed configuration methods.

Parameter	Description
Output Mode	MCR7-DN(P)16 PLCs only support 'Pulse + Direction' mode.
Pulse Output	The MCR7-DN(P)16 provides 8 high-speed output points (QX0.0 – QX0.7). Any of these points can be selected as the pulse output signal for the current axis
Direction Output	Any output point of the PLC can be selected as the direction output signal of the current axis. If a high-speed output point is selected as the direction output signal, this point cannot be used as the pulse signal of other pulse axes.
Reverse	When you find that the servo's movement direction is not as expected, check Reverse to adjust the servo's movement direction without adjusting external wiring.
Home Signal	The starting point of motion, this signal can be configured and selected according to the system hardware conditions or application scenarios.
Positive Limit	The limit position of the machine in the positive direction. Beyond this position, the machine will stop moving or issue a warning. This signal can be configured and selected according to the system's hardware conditions or application scenarios.
Negative Limit	The machine is at the limit position in the negative direction. If it exceeds this position, the machine will stop moving or issue a warning. This signal can be configured and selected according to the system's hardware conditions or application scenarios.
Z signal	Z phase is the zero signal. The encoder outputs a pulse for each shaft revolution. It is generally used when the machine returns to zero. This signal can be configured and selected according to the system's hardware conditions or application scenarios.
Servo ready input signal	The power-on self-test is completed normally, the servo driver outputs a Ready signal to the controller. After receiving this signal, the controller will output the servo enable signal to the driver to avoid forcing the enable signal to cause more serious servo failure in the event of servo failure. This signal can be configured and selected according to the system's hardware conditions or application scenarios.
Servo fault input signal	If the servo driver encounters a problem during power-on or operation, the driver will output an ERROR signal to the controller. After receiving this signal, the controller will know that the servo has failed and output a warning. This signal can be configured and selected according to the system's hardware conditions or application scenarios.
Servo enable output signal	The output signal is used to enable the servo driver. The servo driver enters the enable state after receiving this signal. Generally connected to the Servo_On or SON terminal of the servo driver . This signal can be configured and selected according to the system's hardware conditions or application scenarios.

Parameter	Description
Alarm reset output signal	Output signal for fault reset of servo driver. After receiving this signal, the servo driver will try to perform fault reset operation to restore normal operation. Generally connected to the Reset terminal of the servo driver. This signal can be configured and selected according to the system's hardware conditions or application scenarios.

Step108. Set project unit.

- ◆ Relative pulse: The distance in the project is in the unit of “number of pulses”, the speed is in the unit of “number of pulses per second”, and the acceleration and deceleration is in the unit of “number of pulses per second squared”.

- ◆ User unit

It is necessary to configure the number of pulses in one revolution of motor/encoder , and the amount of movement of the worktable in a circle.

As shown in the above figure, the number of pulses in one revolution of motor/encoder is 2000, and the amount of movement of the worktable in a circle is 2000mm, i.e. the number of pulses required for a corresponding worktable movement of 2000mm is 2000.

The distance in the project is set by the user in unit (e.g. millimeters), the speed is expressed in u/second (e.g. millimeters/second), and the acceleration and deceleration are expressed in u/quadratic second (e.g. millimeters/quadratic second), where u is user unit.

Use gearbox: In practical applications, a gearbox may be used to connect the driving motor to the load. At this point, one revolution of the motor no longer corresponds to one revolution of the worktable. The speed ratio of the gearbox needs to be calculated, as shown in the following application.

Axis_0

+

Basic Setting

Project Unit Setting

Mode/Parameter Setting

Home Return Setting

All speeds and distances will be specified by the following measurement system

Choose measurement system : ☐ Relative pulse ☒ User unit

All speeds and distances will be specified by the following measurement system

Number of pulses in one turn by motor/encoder : 2000 Pulse ☐ Hexadecimal

☐ Use gearbox

The amount of movement of the worktable in a circle : 2000 mm v

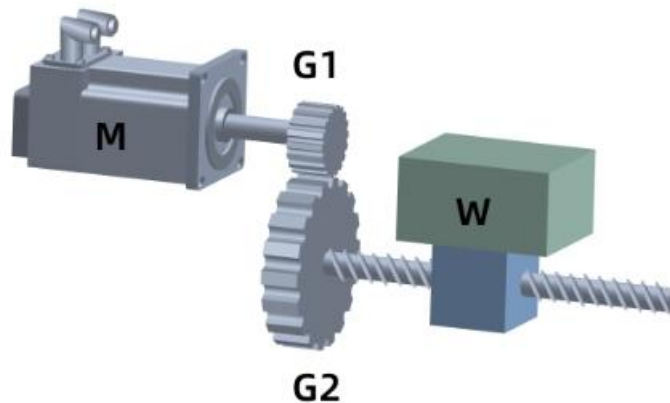
Pulse number = $\frac{\text{Number of pulses rotated by motor/encoder}[DINT]}{\text{Moving amount of worktable rotation}[REAL]} * \text{Moving distance}(UINT)$

■ When the axis type is linear model:

Number of pulses =

$$\frac{\text{Number of pulses rotated by mortor or encoder [DINT]} * \text{numerator of Gear Ratio [DINT]}}{\text{Moving amount of workbench rotation[REAL]} * \text{denominator of Gera Ratio[DIINT]}} *$$

moving distance [UINT]



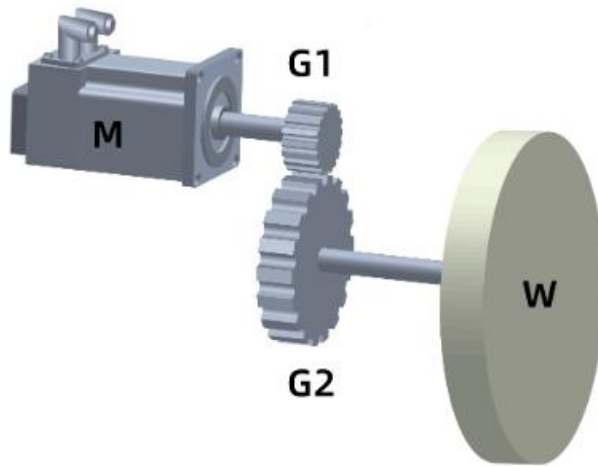
As shown in the figure above, M is the motor, W is the workbench, G1 is the numerator of the gear ratio, and G2 is the denominator of the gear ratio.

■ When the axis type is rotation mode:

Number of pulses

$$= \frac{\text{Number of pulses rotated by mortor or encoder [DINT]} * \text{numerator of Gear Ratio [DINT]}}{\text{Moving amount of workbench rotation[REAL]} * \text{denominator of Gera Ratio[DIINT]}}$$

* moving distance [UINT]



Step109. Set mode/parameters.

Axis_0 +

Basic Setting

Project Unit Setting

Mode/Parameter Setting

Home Return Setting

Software limits :

☒ Enable

Negative limit : mm

Forward limit : mm

Please refer to the table below for detailed configuration methods.

parameter	Description
Enable	Whether to enable the software limit function.
Negative limit	Software limit, the maximum distance of negative movement. Beyond this distance, the machine will stop moving and issue a warning.
Forward limit	Software limit, the maximum distance of forward movement. Beyond this distance, the machine will stop moving and issue a warning.

Step110. Set homing mode.

Axis_0 +

Basic Setting

Project Unit Setting

Mode/Parameter Setting

Home Return Setting

Home Signal : (0..IX1.2)

Z Signal :

Positive Limit :

Negative Limit :

Home Return Direction :

Home Input Detection Direction :

Home Return Mode :

Home Return High Speed(H): mm/s

Home Return Acceleration : mm/s²

Home Return Low Speed(L): mm/s

357

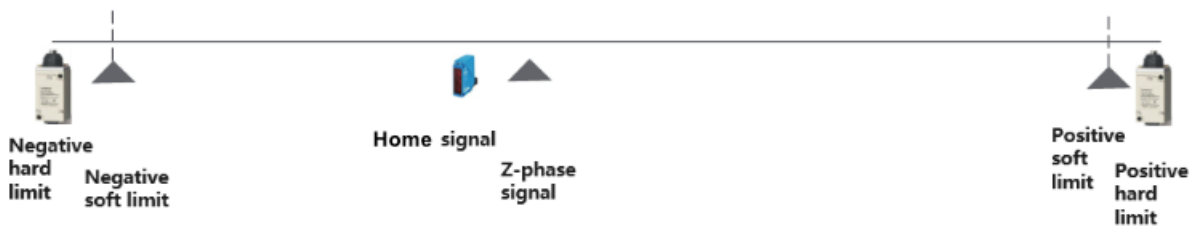
Please refer to the table below for detailed configuration methods.

Parameter	Description
Home Signal	◆ Use: Use the home signal. ◆ Do not use: Do not use the home signal. ◆ Unassigned: The home signal is not assigned.
Z signal	◆ Use: Use Z phase signal. ◆ Do not use: Z-phase signal is not used. ◆ Unassigned: The Z-phase signal is not assigned.
Positive Limit	◆ Use: Use positive limit signal. ◆ Do not use: Do not use the positive limit signal. ◆ Unassigned: The positive limit signal is not assigned. Only one of the positive limit and the negative limit can be used at most.
Negative Limit	◆ Use: Use negative limit signal. ◆ Do not use: Do not use the negative limit signal. ◆ Unassigned: The negative limit signal is not assigned.
Home Return Direction	◆ Forward: When returning to zero, the initial movement direction is positive. ◆ Negative : When returning to zero, the initial movement direction is negative. ◆ Not assigned: The home direction signal is not assigned.
Home Input Detection Direction	◆ Forward: Stop moving when it encounters the edge of the zero signal during forward motion. ◆ Negative direction: Stop moving when encountering the edge of the zero signal during negative movement. ◆ Unassigned: The zero point input detection direction signal is not assigned.
Home Return Mode	the homing mode based on the current system's hardware facilities , and the software will automatically filter out the corresponding homing mode based on the user's selection.
Home Return High Sped(H)	The maximum speed to return to the starting point.
Home Return Low Sped(L)	Minimum speed to return to starting point.
Home Return Acceleration	The acceleration back to the starting point.

Return to zero means moving to a defined position. This position is defined as the reference point (or starting point), which serves as the reference point for absolute movement. All absolute movements based on the coordinate system (such as MC_MoveAbsolute) must be based on the establishment of a reference point, otherwise the axis will enter the ErrorStop state.

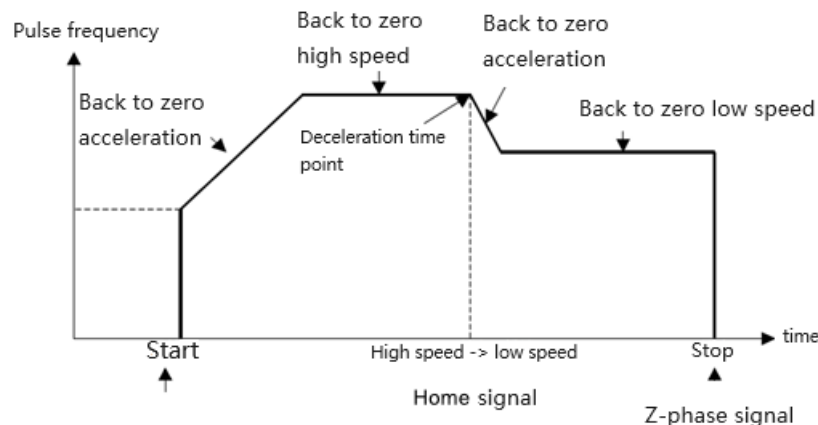
MCR7 series PLCs support homing methods No. 1 ~ 35 defined by Ci A 402 protocol.

A linear motion system generally consists of the following elements.



- ◆ **Home signal:** In a system with Z-phase signal, it is generally used as a forward-looking signal for Z-phase signal monitoring. After the system detects the Home signal, it will control the zero-return speed from homing high speed to homing low speed, making it easier to obtain the cycle. The shorter Z-phase signal ensures the accuracy and consistency of the zero point; in systems without Z-phase signal, it is generally directly used as the zero-point signal of the system.
- ◆ **Positive/ negative hard limit:** When the axis moves linearly, the limit switches set at both ends of the movement play a role in system protection.
- ◆ **Positive/ negative software limit:** After the axis coordinate system is established, by setting the software limit, when the axis moves linearly, the system can enter the early warning state in advance to avoid actually triggering the hardware limit switch, which also plays a role in system protection; when in return-to-zero motion, the soft limit has no effect. In addition to basic hardware facilities, users can specify whether the return-to-zero direction is positive or negative by setting **Home Return Direction**.
- ◆ **Zero input detection direction:** Set the movement direction when reaching the zero signal. Forward: Stop moving when it encounters the edge of the zero signal during forward motion. Negative direction: Stop moving when encountering the edge of the zero signal during negative movement.
- ◆ **Homing high speed:** In the homing mode, the homing high speed setting will shorten the homing time and improve the system efficiency.
- ◆ **Homing low speed:** After the system detects the home signal, it switches to a lower zero return low speed operation to ensure that the system can accurately capture the Z-phase signal and successfully complete the homing movement.
- ◆ **Homing acceleration:** The switching slope between the homing high speed and the homing low speed, which reduces the impact on the system during speed switching.

The homing curve is shown in the figure below.



After completing the creation of the motion axis, you can program the motion axis in the **task** area.

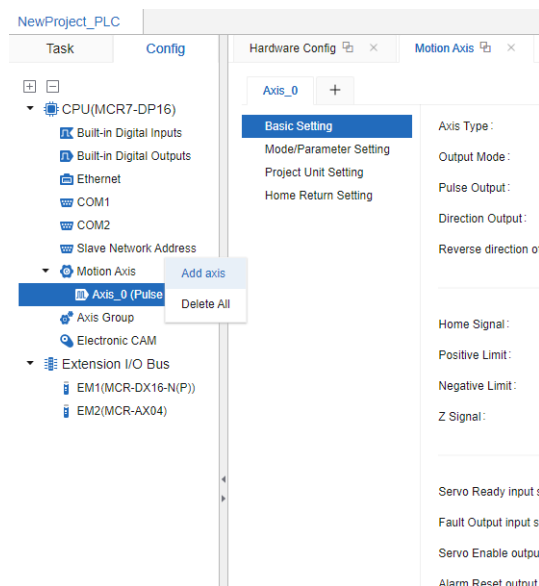
13.5.2 PWM

PWM, the full name is Pulse Width Modulation. The principle is to obtain an analog voltage signal by changing the duty cycle of the square wave, that is, the time ratio of high level and low level in one cycle. At a certain frequency, output analog voltages of different sizes can be obtained through different duty cycles.

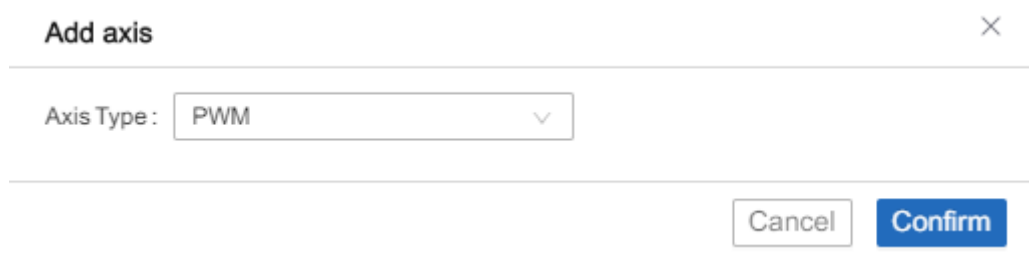
PWM has wide applications in many fields. For example, in motor control, PWM can be used to control the speed of the motor to achieve precise control of the motor. In addition, PWM is often used to control LED lighting. By adjusting the duty cycle of the PWM signal, the brightness of the LED can be changed to achieve precise lighting control in different environments, save energy and avoid unnecessary optical effects.

Creating a new P W M axis is as follows:

Step111. Right-click **Motion Axis** and select **Add Axis** in the pop-up menu.



Step112. Select **PWM** in the pop-up dialog box and click **Confirm**.



Step113. Set the basic properties of the motion axis.

Axis_0

Axis_1

+

Basic Setting

Axis Type :

PWM

Output Mode :

PWM

PWM Output :

0.QX0.2

High speed output

Output Frequency :

10000

Hz

Please refer to the table below for detailed configuration methods.

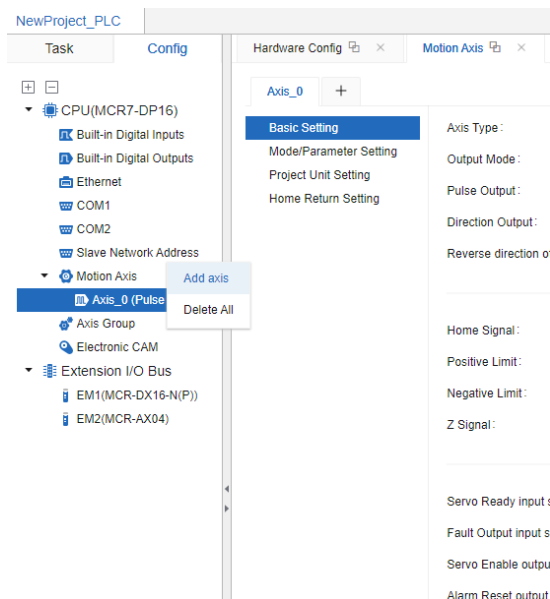
Parameter	Description
Output Mode	Only PWM is supported .
PWM Output	Select the high-speed output address.
Output Frequency	PWM frequency, value range 2000~50000Hz .

13.5.3 Counting axis

Counters are implemented in software and engineering applications in the form of encoder shafts for management applications. After the counter is associated with the axis, it is called a counter axis. MCR7 series PLCs support AB phase 1 frequency multiplication, AB phase 4x frequency, CW/CCW, pulse + direction and single-phase counting modes. With other input signals, it can realize counter preset and latch functions.

The operation method of creating a new counting axis is as follows:

Step114. Right-click **Motion Axis** and select **Add Axis** in the pop-up menu.



Step115. Select **Encoder Axis** in the pop-up dialog box and click **Confirm**.

Add axis
✕

Axis Type : Encoder Axis ▼

Cancel
Confirm

Step116. Set basic properties.

Axis_0
Axis_1
Axis_2
+

Basic Setting

Project Unit Setting

Mode/Parameter Setting

Axis Type : Encoder Axis ▼

Counter Mode : Pulse+Direction ▼

Pulse input : 0.IX0.0 High speed input ▼

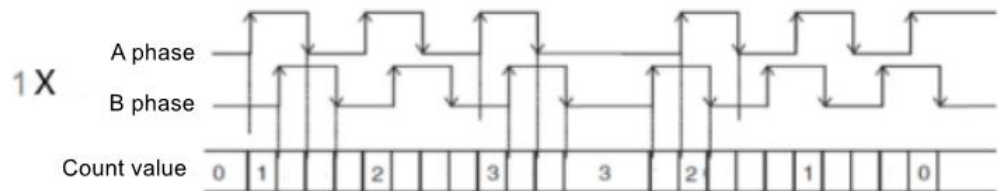
Direction input : 0.IX0.1 High speed input ▼

Please refer to the table below for detailed configuration methods.

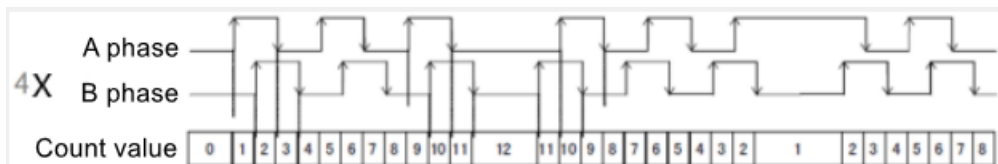
Parameter	Description
Conter Mode	◆ Pulse + Direction: Define the pulse using a pulse signal and a direction signal. It is necessary to define the address of the pulse input and the address of the direction input. ◆ CW /CCW : Uses CW pulses and CCW pulses . It is necessary to define the address of the CW pulse and the address of the CCW pulse. CW pulse is a forward pulse signal (looking

from the motor shaft to the motor, the motor rotates clockwise); CCW pulse is a reverse pulse signal (looking from the motor shaft to the motor, the motor rotates counterclockwise).

- ◆ AB phase 1x: Common encoder output signals include three phases: A, B, and Z. Phase A / B is the pulse output signal, and phase Z is the number of revolutions. The phase difference between A and B is 90° . According to A leading B still lags behind B to determine the direction of rotation. One frequency multiplication count means that one cycle of A and B square waves is completely detected and a pulse is output.



- ◆ AB phase 4x: 4 times frequency is to detect 1 rising edge and 1 falling edge of phase A and 1 rising edge and 1 falling edge of phase B in one cycle. Each rising edge or falling edge outputs a pulse, so 4 pulses are output in one cycle, so the resolution is improved and the positioning is accurate.



Step117. Set project unit.

Please refer to [Step108](#).

Axis_0

Axis_1

Axis_2

+

Basic Setting

Project Unit Setting

Mode/Parameter Setting

All speeds and distances will be specified by the following measurement system

Choose measurement system : ☐ Relative pulse ☒ User unit

All speeds and distances will be specified by the following measurement system

Number of pulses in one turn by motor/encoder : Pulse ☐ Hexadecimal

☐ Use gearbox

The amount of movement of the worktable in a circle :

Pulse number = $\frac{\text{Number of pulses rotated by motor/encoder[DINT]}}{\text{Moving amount of worktable rotation[REAL]}} \times \text{Moving distance(Uint)}$

Step118. Set Mode/Parameters.

Axis_0
Axis_1
Axis_2
+

Basic Setting
Project Unit Setting
Mode/Parameter Setting

Software limits :
☒ Enable
Negative limit:
0 mm
Forward limit:
1000 mm

Negative Limit Value
0
Positive Limit Value

Incremental Count
Minus Count

Underflow
Overflow

Prob settings :
☒ Prob1 enable
☒ Prob2 enable
Prob1:
0.IX1.0 Low speed input
Prob2:
0.IX0.5 High speed input

Please refer to the table below for detailed configuration methods.

Parameter	Description
Software limit	check Enable , the software limit is enabled. After the counting axis reaches the positive limit value/negative limit value, the system will generate an alarm output and stop counting.
Negative limit value	Software limit, the maximum distance for negative counting. Stop counting when it exceeds the limit value.
Positive limit value	Software limit, the maximum distance for forward counting, and stops counting when it exceeds the limit value.
Probe 1/Probe 2	Each counter supports 2 external inputs to latch the current value of the counter to implement the probe function. Check the probe enable to enable the external input counter axis position latch, and the input address can be selected arbitrarily.

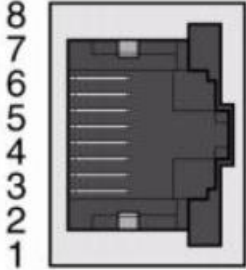
14 Typical Configuration Cases

14.1 Modbus RTU Communication Between MX On CORPORATION PLC and Schneider ATV32 Frequency Inverter

14.1.1 Wiring method

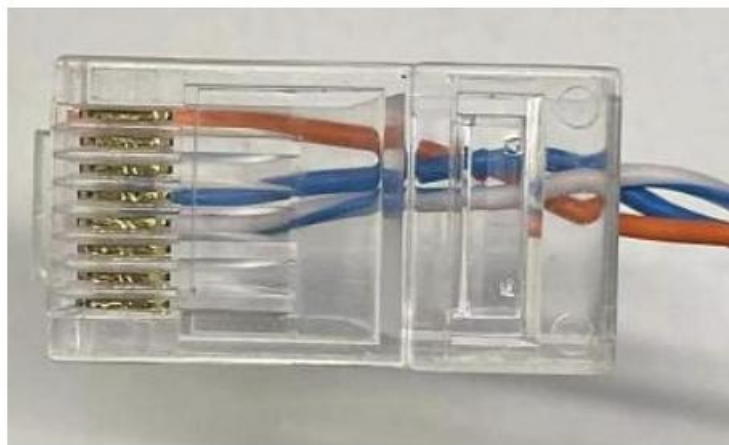
Since ATV32 series frequency inverter only supports RS485 communication with PLC via the network port, you need to wire the network port according to the following figure.

Pin	Signal
1	Reserved for CANopen ⁽³⁾
2	
3	
4	D1 ⁽¹⁾
5	D0 ⁽¹⁾
6	-
7	VP, 10 Vdc ⁽²⁾
8	Common

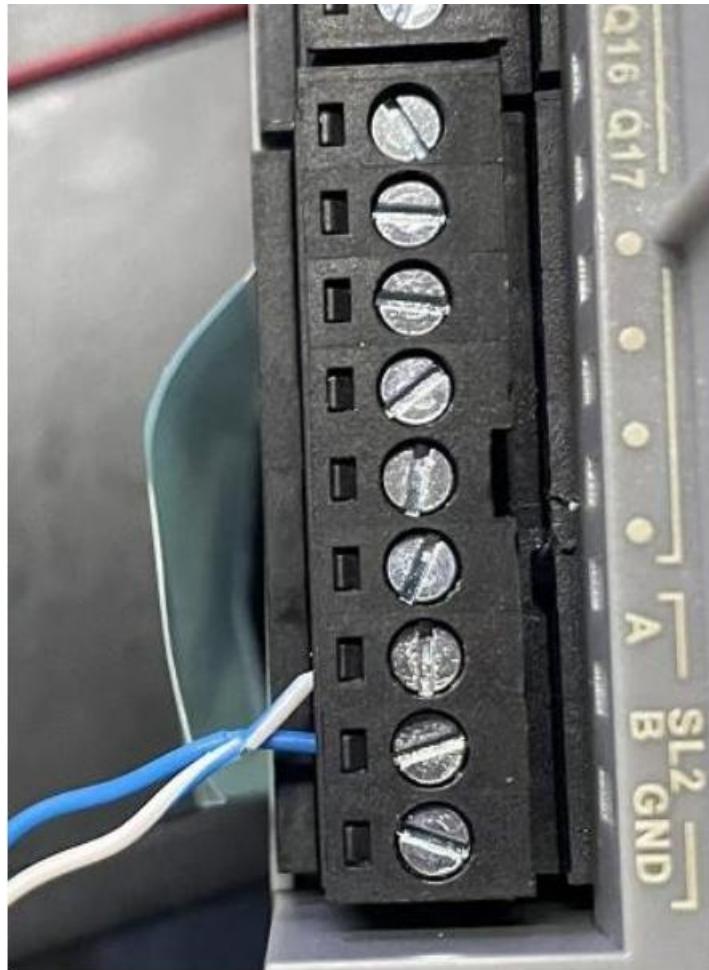


Only three wires, 4, 5 and 8, are required for RS485 communication. The actual wiring is shown below.

◆ ATV32 terminal



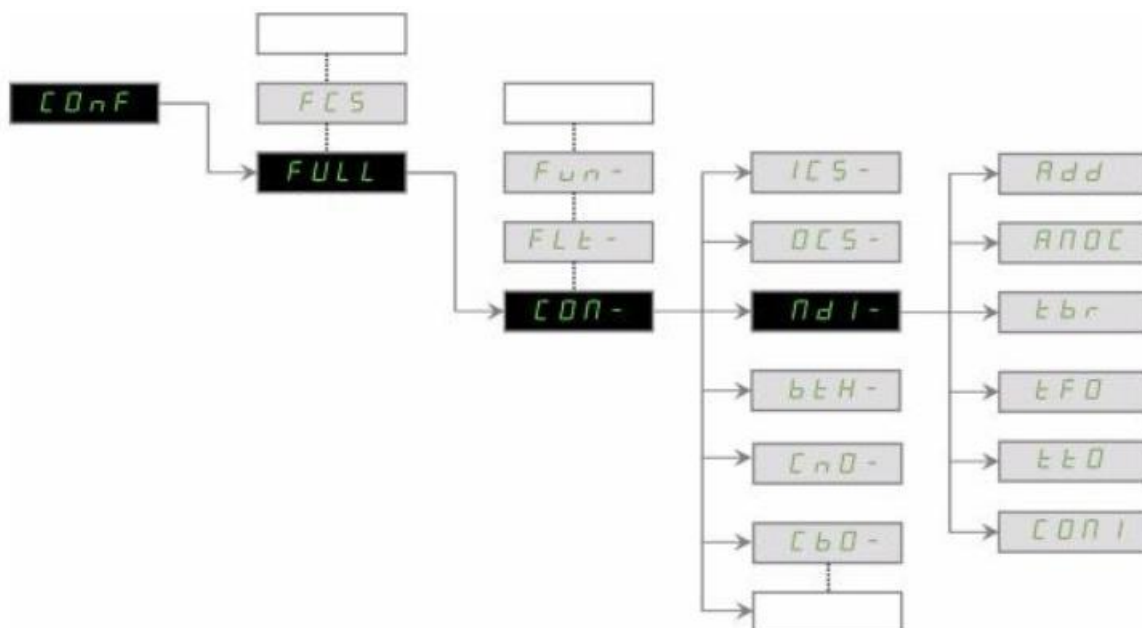
◆ MCR7 series PLC terminal



14.1.2 Configure ATV32

According to the ATV32 frequency inverter Modbus communication manual, enter the following parameters by a controller to set the basic communication parameters and modes.

The parameter used for this test (Reference) is: Add=1; tbr =9.6, tfd =8n1, ttd =10.



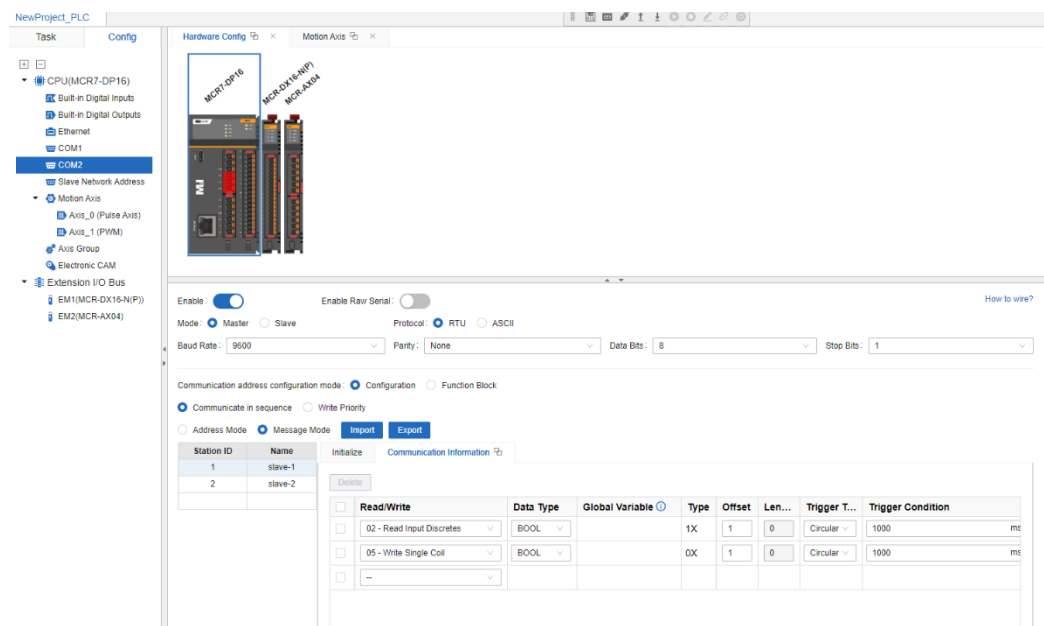
Parameter description	Range or listed values	Default	Possible value	Modbus address
[Modbus Address] (Add)	1 to 247 0: OFF (broadcast only, see "Modbus Protocol" on page 57)	[OFF] (OFF)	[OFF; 1 to 247] (OFF; 1 . . . 247)	16#1771 = 6001
[Modbus baud rate] (tbr)	4,8 kbps 9,6 kbps 19,6 kbps ⁽¹⁾ 38,4 kbps	[19.2 kbps] (19.2)	[4.8] (4.8) [9.6] (9.6) [19.2] (19.2) [38.4] (38.4)	16#1773 = 6003
[Modbus format] (tFD)	8 data bits, odd parity, 1 stop bit 8 data bits, even parity, 1 stop bit ⁽¹⁾ 8 data bits, no parity, 1 stop bit 8 data bits, no parity, 2 stop bits	[8E1] (8E1)	[8O1] (8O1) [8E1] (8E1) [8N1] (8N1) [8N2] (8N2)	16#1774 = 6004
[Modbus time out] (tED)	Adjustable from 0.1 to 30s	[10 s] (10)	[0.1 to 30.0] (0.1 . . . 30.0)	16#1775 = 6005

[Ref.1 channel] (FrI)	[Modbus] (NdIb)
[Cmd switching] (CCS)	Default
[Cmd channel 1] (CdI)	[Modbus] (NdIb)
[Forward] (Fr d)	[Cd00] (CdDD)
[Reverse assign.] (rrS)	[Cd01] (CdDI)
[Fault reset] (rSF)	[Cd07] (CdDT)
[Profile] (CHCF)	[Separate] (SE P)

14.1.3 Configure PLC

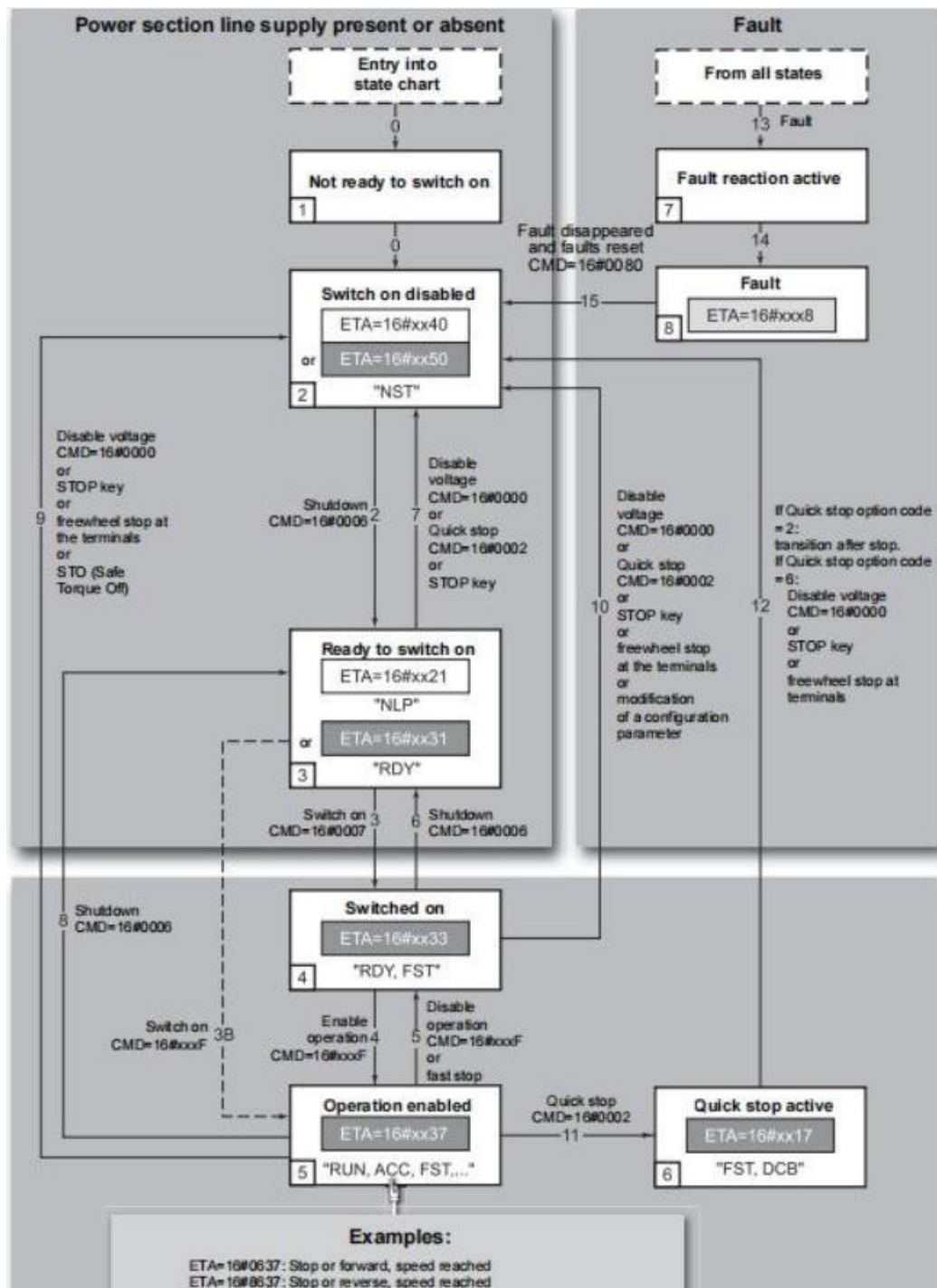
14.1.3.1 Configure Serial Communication

Run the MCR Master software and click **Config** to enter the **configuration** interface. Select the actually connected serial port (such as **COM2**) and configure it according to the figure below.



14.1.3.2 Program PLC

Step119. According to the ATV32 frequency inverter Modbus communication manual, the following states need to be given in order to start and run the motor.



Step120. According to the ATV32 frequency inverter programming manual, the command word (CMD) is 4X8501, the set speed is 4X8602, the status word (ETA) is 4X3201, and the current speed is 4X8604. Since the Modbus communication address in the PLC starts from 1 instead of 0, the Modbus address created in MCR Master needs to be increased by 1.

The following figure shows that Create variable and bind Modbus address.

NewProject0130_1_PLC

Task Config

POU

PRG

(LD) Main

(LD) m2

Function Block

Function

Global Variable

GLOBAL

System Variable

Custom Data Type

Task Config

Trace

Main Hardware Config Motion Axis GLOBAL

Add Export Import Move Up Move Down Delete Move Save Search

#	Name	Data Type	Global Category	Init
1	GVL_Var4	BOOL	GLOBAL	
2	GVL_Var5	BOOL	GLOBAL	
3	CMD	WORD	GLOBAL	
4	SET_SPEED	WORD	GLOBAL	
5	READ_SPEED	WORD	GLOBAL	
6	ETA	WORD	GLOBAL	

Communication address configuration mode: ☒ Configuration ☐ Function Block

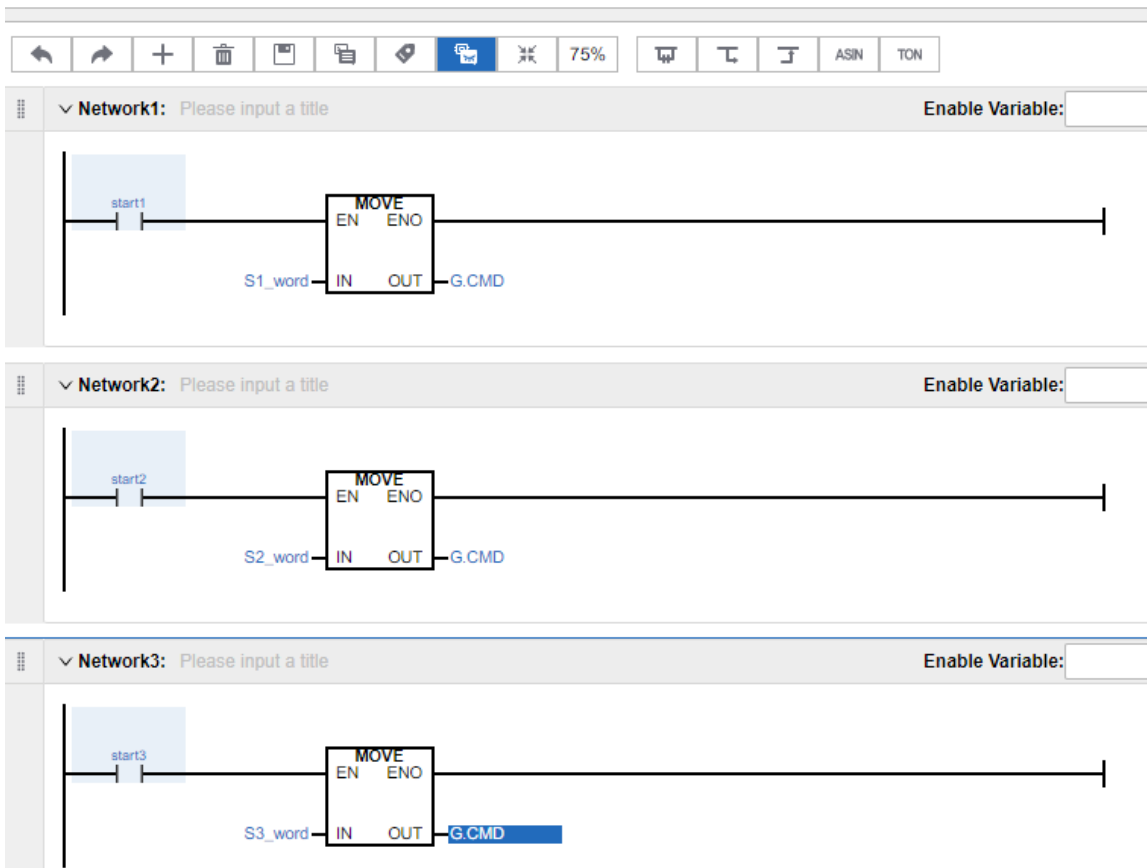
☐ Address Mode ☒ Message Mode **Import** **Export**

Station ID	Name
1	slave-1

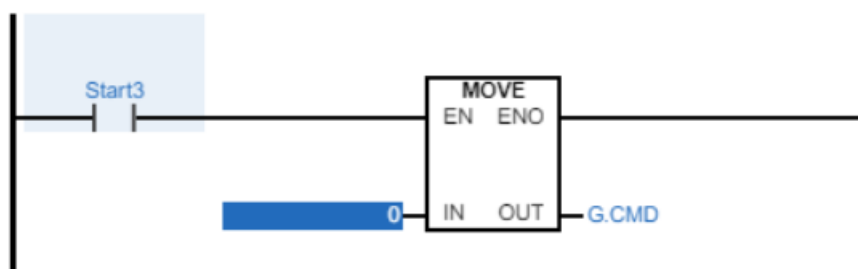
Read/Write	Data Type	Global Variable	Type	Offset	Length	Trigger Type	Trigger Condition	ErrorCode(Byte)	Comme
06 - Write Single Register	WORD	CMD	4X	8502	1	Circular	200	ms	
06 - Write Single Register	WORD	SET_SPEED	4X	8603	1	Circular	200	ms	
03 - Read Holding Registers	WORD	READ_SPEED	4X	8605	1	Circular	200	ms	
03 - Read Holding Registers	WORD	ETA	4X	3202	1	Circular	200	ms	

Step121. Write ladder diagram programs in MCR Master .

☐	#	Name	Variable Type	Data Type	Init	Cons/F
☐	1	start1	Local	BOOL		
☐	2	start2	Local	BOOL		
☐	3	start3	Local	BOOL		
☐	4	S1_word	Local	WORD		
☐	5	S2_word	Local	WORD		
☐	6	S3_word	Local	WORD		

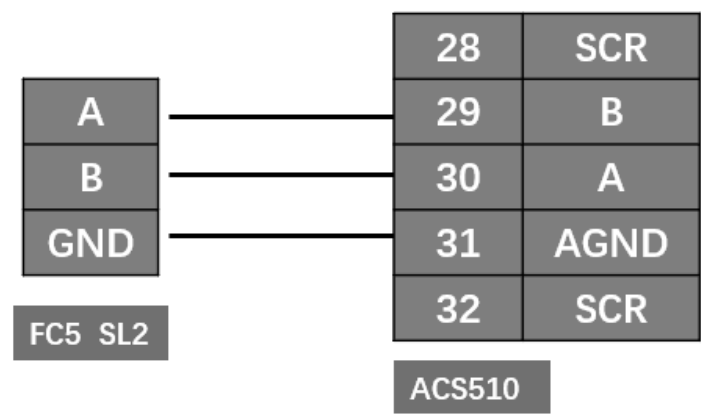


Step122. If you want to stop the motor from running, set CMD to 0.



14.2 Modbus RTU Communication between MX On CORPORATION PLC and ABB ACS510 Frequency Inverter

14.2.1 Wiring method



14.2.2 Configure ACS510

Step123. Configure the communication protocol and station address parameters (in this case, set the 5302 parameter to 1).

Parameter code	Description	Parameter configuration method (when selecting Modbus protocol)
5301	EFB PROTOCOL ID, communication protocol ID and program version.	Read only. Setting parameter 9802 COMM PROT SEL (communication protocol selection) to a non-zero value will automatically set this parameter. The format is: XXYY, where XX is the protocol ID and YY is the program version.
5302	EFB STATION ID, the station address of the RS485 link.	Use a unique value to represent the transmission of the network. When the protocol is Modbus, the value defaults to 1.



For a new station address to take effect, the drive must be powered off and on again, or parameter 5302 must be set to 0 before selecting a new station address. Parameter 5302 is 0, which means the RS485 channel is reset and communication is prohibited.

Step124. Configure Baud rate, parity and other parameters (in this case, set the 5033 parameter to 9.6, the 5034 parameter to 0, and the 5305 parameter to 0).

Parameter code	Description	Parameter configuration method (when selecting Modbus protocol)
5303	EFB BAUD RATE, RS485 network communication rate, unit is kbit/s. The following parameters are supported: 1.2, 2.4, 4.8, 9.6, 19.2, 38.4, 57.6, 76.8.	The default value is 9.6 .
5304	EFB PARITY, data bits, parity, and stop bits of RS485 communication. The settings must be consistent across all stations in the network. The following parameters are supported: 0: Indicates 8-bit data, no check, and 1 stop bit. 1: Indicates 8-bit data, no check, and 2 stop bits. 2: Indicates 8-bit data, even check, and 1 stop bit. 3: Indicates 8-bit data, odd check, and 1 stop bit.	The default value is 1.
5305	Select the communication profile used by the EFB protocol. The following parameters are supported: 0: Indicates ABB DRV LIM . The operation of the control word/status word must comply with the requirements of the ABB drive configuration file. The typical application is the same as the ACS400 drive. 1: Indicates DCU PROFILE. The operation of control word/status word must comply with the requirements of the 32-bit DCU configuration file. 2: Indicates ABB DRV FULL. The operation of the control word/status word must comply with the requirements of the ABB drive	The default value is 0.

Parameter code	Description	Parameter configuration method (when selecting Modbus protocol)
	configuration file. The typical application is the same as the ACS600/800 drive.	

Step125. Set the start/stop control and direction (in this case, set the 1001 parameter to 10 and the 1003 parameter to 3).

Drive parameters		Value	Description	Modbus protocol provisions	
				ABB drives	DCU settings _
1001	External 1 command	10 (communication)	Start/stop control via fieldbus Select via Ext1.	40001 bits 0~3	40031 bits 0,1
1002	External 2 command	10 (communication)	Start/stop control via fieldbus Select via Ext2.	40001 bits 0~3	40031 bits 0,1
1003	direction	3 (bidirectional)	Direction is controlled by fieldbus .	40002/40003	40031 bit 3

Step126. Select the input given (in this case, set the 1102 parameter to 8 and the 1103 parameter to 8).

Drive parameters		Value	Description	Modbus protocol provisions	
				ABB drives	DCU settings _
1102	External 1/2 selection	8 (Communication)	Reference is selected via fieldbus .	40001 bit 11	40031 bit 5
1103	Given 1 choice	8 (Communication)	Input reference 1 comes from fieldbus.	40002	
1106	Given 2 choices	8 (bidirectional)	Input reference 2 comes from fieldbus.	40003	

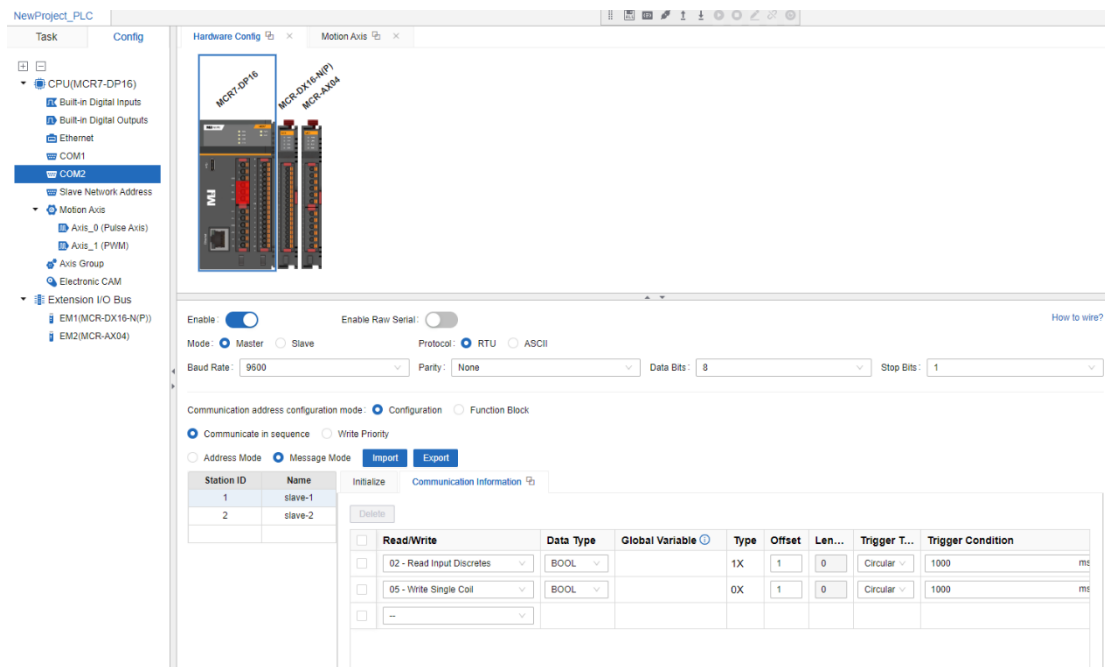
Step127. Set the signal source (in this case, set the 1601 parameter to 7, the 1604 parameter to 8, the 1606 parameter to 8 , and the 1607 parameter to 1).

Drive parameters		Value	Description	Modbus protocol provisions	
				ABB drives	DCU settings
1601	RUN ENABLE run allowed	7 (Communication)	The run enable signal comes from the fieldbus.	40001 bit 3	40031 bit 6 (anti-logic)
1604	FAULT RESET SEL Fault reset selection	8 (Communication)	The fault reset signal comes from the fieldbus.	40001 bit 7	40031 bit 4
1606	LOCAL LOCK local lock	8 (Communication)	The local control lock selection signal comes from the fieldbus.	-	40031 bit 14
1607	PARAM SAVE Parameter storage	1 (storage)	Save the changed parameters to memory (return value is 0).	4 1607	
1608	START ENABLE 1 Start allowed 1	7 (Communication)	The signal source of start enable 1 is the fieldbus command word.	-	40032 bit 2
1609	START ENABLE 2 Start allowed 2	7 (Communication)	The signal source of start enable 2 is the fieldbus command word.	-	40032 bit 3
2 201	ACC/DEC 1/2 SEL Acceleration and deceleration 1/2 selection	7 (Communication)	The signal source for the acceleration/deceleration ramp pair is the fieldbus.	-	40031 bit 10

14.2.3 Configure PLC

14.2.3.1 Set Communication Parameters

Run the MCR Master software, select **Configuration** , click on the actually connected serial port (such as COM2), and set it as shown in the figure below.



14.2.3.2 Program PLC

Step128. According to the ACS510 frequency inverter manual, the following states need to be given in order to start and run the motor.

Step	Control word value	Description
1	CW=0000 0000 0000 0110	This value puts the drive into READY TO SWITCH ON state.
2	-	100 milliseconds before proceeding to the next step .
3	CW=0000 0000 0000 0111	This value puts the drive into READY TO OPERATE state.
4	CW=0000 0000 0000 1111	This value puts the drive into OPERATION ENABLED state. The drive starts to fire up but will not accelerate.
5	CW=0000 0000 0010 1111	This value releases the RFG output and the drive enters the RFG: ACCELERATOR ENABLED state.
6	CW=0000 0000 0110 1111	This value releases the integral function generator (RFG) output and the drive enters the OPERATING state. The drive accelerates

Step	Control word value	Description
		to the specified value and operates at the specified value.

Step129. Run the MCR Master software, create corresponding variables and map the M-area address to facilitate addressing operations, and configure the Modbus communication address.

Start_1 to Start_5 are the addressing bit variables corresponding to the control word variable Control. Speed is the current speed that needs to be set for communication transmission, and Read_Speed is the current speed that is read and stored in this variable.

Add

Export

Import

Move Up

Move Down

Delete

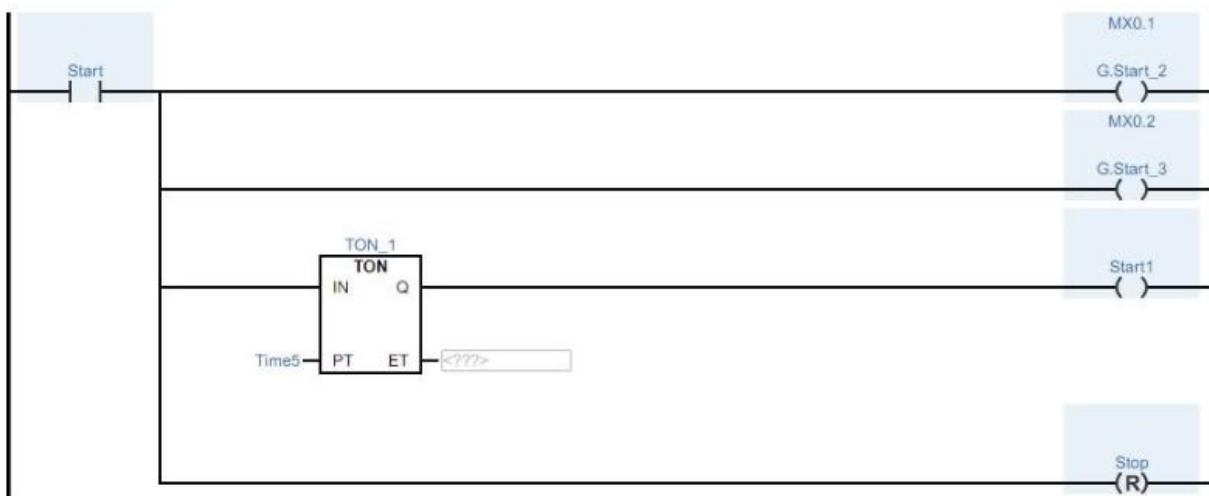
Move

Save

☐	#	Name	Data Type	Global Category	Init	Cons/Retain	Address
☐	1	READ_SPEED	WORD	G1			
☐	2	CONTROL	WORD	G1			MW0
☐	3	SPEED	WORD	G1			MW1
☐	4	START_0	BOOL	G1			MX0.3
☐	5	START_01	BOOL	G1			MX0.0
☐	6	START_02	BOOL	G1			MX0.1
☐	7	START_03	BOOL	G1			MX0.2
☐	8	START_04	BOOL	G1			MX0.5
☐	9	START_05	BOOL	G1			MX0.6
☐	10	ERR_RESTART	BOOL	G1			MX0.7
☐	11	SWITCH_1	BOOL	G1			MX1.3

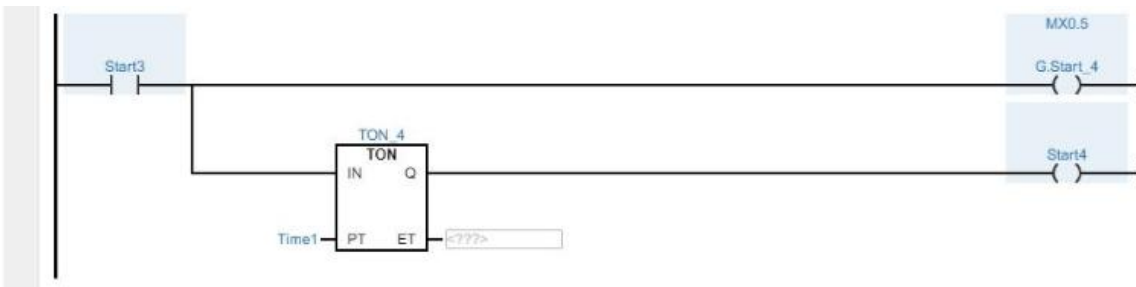
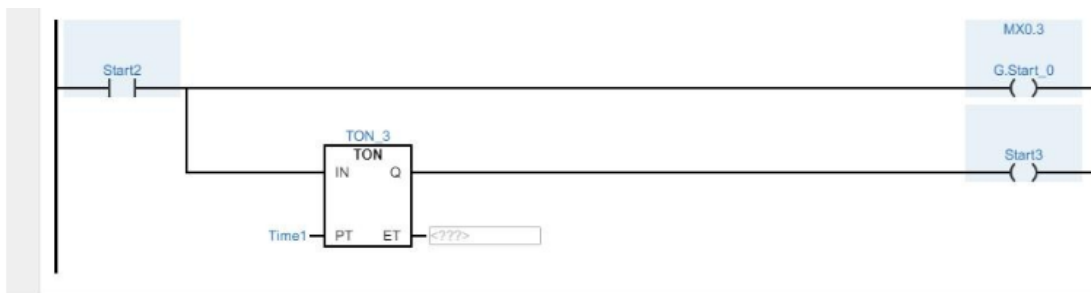
☐	Read/Write	Data Type	Global Variable	Type	Offset	Length	Trigger Type	Trigger Condition	ErrorCode(Byte)
☐	06 - Write Single Register	WORD	CONTROL	4X	1	1	Circular	1000	ms
☐	06 - Write Single Register	WORD	SPEED	4X	2	1	Circular	1000	ms
☐	03 - Read Holding Registers	WORD	READ_SPEED	4X	102	1	Circular	1000	ms

Step130. Please follow the figure below to write the ladder diagram of steps 1 to 3 of the state machine.

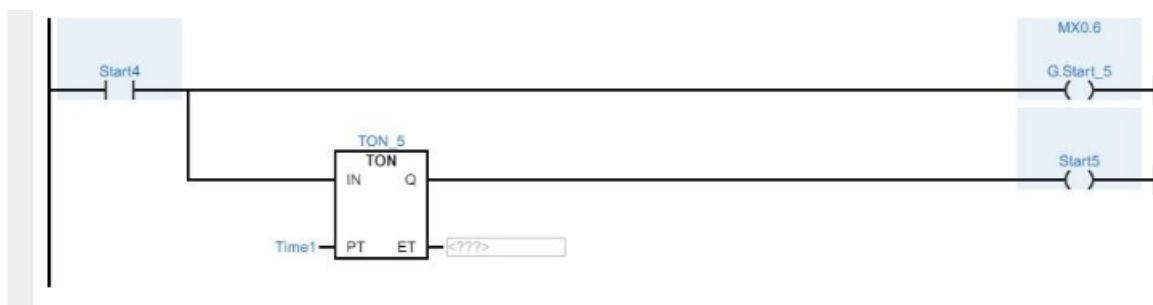


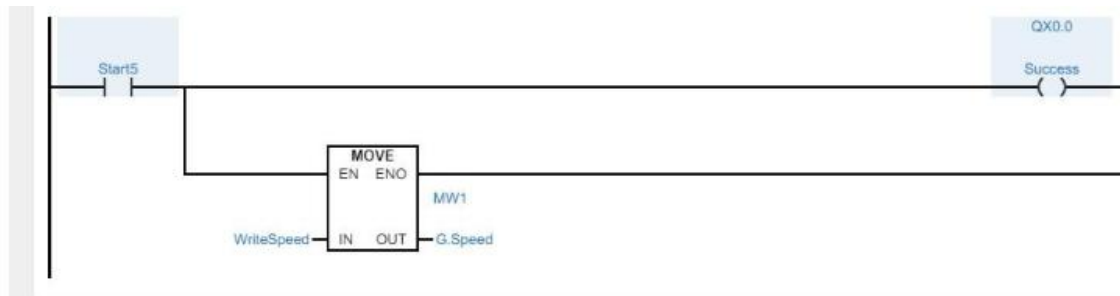
As shown in the figure above, Start is the startup condition. When Start is turned on, the global variables Start_2 and Start_3 that map the MX0.1 and MX0.2 addresses are set. At this time, because the MW0

address is mapped, the variable Control is also changed. Configuration The Modbus address is configured as a master station's "write single register" action, and the corresponding slave station address 4X0001 is written.



Step131. 4X0001 is the control word of ACS510 , and the above ladder diagram completes the conversion of steps 1 to 3 of the state machine. In the same way, the following is the conversion from steps 3 to 6, including the final speed writing. The speed is first written to WriteSpeed and assigned to the global variable Speed and then communicated via ModbusRTU.





Step132. To stop the motor from running, you need to reset the variable Start_1 mapped to the MX0.0 address. In this experiment, the Stop variable is used to reset it, as shown in the figure below.



Step133. After completing the above program, download the project to MCR7 series PLC and run it, then you can achieve a simple serial Modbus RTU protocol communication program to control the operation of inverter motor.

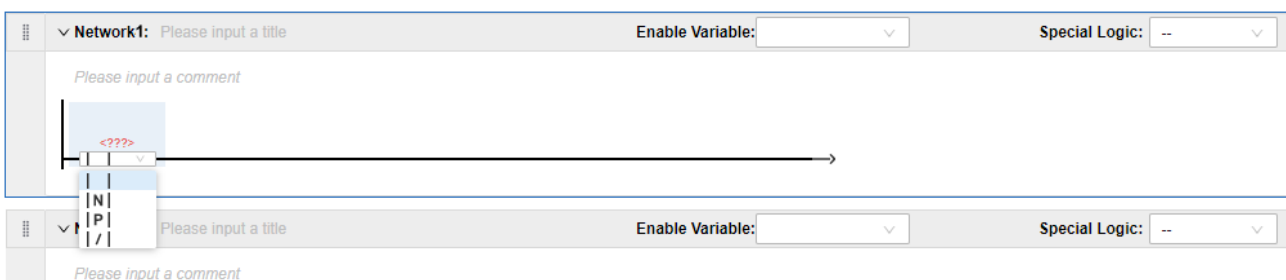
14.3 Common Programming Cases

14.3.1 Starting, Retaining, and Stopping Circuit

The starting, retaining, and stopping circuit, as the name implies, has three functions: start, retain, and stop.

The steps of creating a starting, retaining, and stopping circuit program are as follows:

Step134. Select a program (e.g., MAIN program) in the program organization unit, edit the program (using the Ladder Diagram), and add a normally open contact by double-clicking on the main line in program segment 1.



Step135. Input the variable name (e.g. X1) into the address input box of the normally open contact, enter to bring up the auto-declaration dialog box (you can also add new variables in the variable declaration area at the top of the interface), configure the relevant parameters, and click **Confirm**.

Automatic declaration
✕

* Variable Type : Local ▼

* Name : X1

* Data Type : BOOL ▼

Init :

Cons/Retain : None ▼

Address :

Comment :

Cancel
Confirm

Please refer to the table below for detailed configuration methods.

Parameter	Description
Variable Type	◆ Local variables: Variables declared within a specific scope in a program. They are also called internal variables, which are defined and accessed inside a function or compound statement and can only be used within the declared POU. ◆ Temporary variable: As a kind of instrumental variable, it stores intermediate results or is used to exchange the values of two variables, etc. The scope of temporary variables is usually limited to the block of code in which they are declared; once outside of that block, the variables are destroyed. Therefore, temporary variables are temporary and local. ◆ Global variable: variables that are effective in all POUs in the current project .
Name	Variable name. Please refer to Variable Name for details.
Data Type	For more information about data types, please refer to Data Type of Variable . In this case, it needs to be set to BOOL .
Init	The initial value of the variable.
Constant/Retain	◆ None: The data value is a changing value. ◆ Constant: The data is a fixed value. ◆ Retain: Force variables during online monitoring. Check Retain the force state and value of variables when switching to offline state , and the variables will retain the forced results.
Address	The register address to which the variable is bound.

Parameter	Description
Comment	Description for the variable.

Step136. The results of adding a normally open contact are as follows.

Add
Import
Export
Move Up
Move Down
Convert Global Variable
Delete
Q Search

#	Name	Variable Type	Data Type	Init	Cons/Retain	Address	Comment	Used Times
1	X1	Local	BOOL					0

75%
ASIN
TON

Network1: Please input a title
Enable Variable:
Special Logic: --

Please input a comment

X1

Step137. Double-click the arrow at the right end of the main line, add a coil and bind the variable Y1 .

Add
Import
Export
Move Up
Move Down
Convert Global Variable
Delete
Q Search

#	Name	Variable Type	Data Type	Init	Cons/Retain	Address	Comment	Used Times
1	X1	Local	BOOL					1
2	Y1	Local	BOOL					0

75%
ASIN
TON

Network1: Please input a title
Enable Variable:
Special Logic: --

Please input a comment

X1
Y1

Step138. Use a similar method to add a normally closed contact to the right of the normally open contact and bind the variable X2.

Add
Import
Export
Move Up
Move Down
Convert Global Variable
Delete
Q Search

#	Name	Variable Type	Data Type	Init	Cons/Retain	Address	Comment	Used Times
1	X1	Local	BOOL					1
2	Y1	Local	BOOL					1
3	X2	Local	BOOL					0

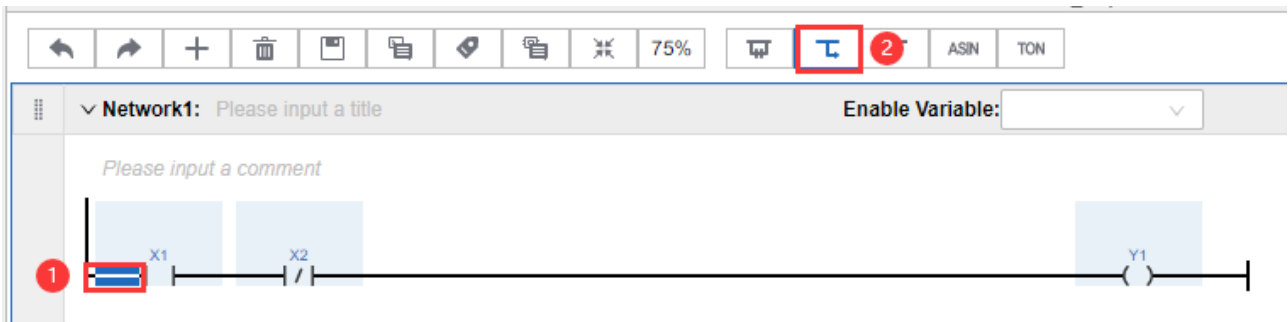
75%
ASIN
TON

Network1: Please input a title
Enable Variable:
Special Logic: --

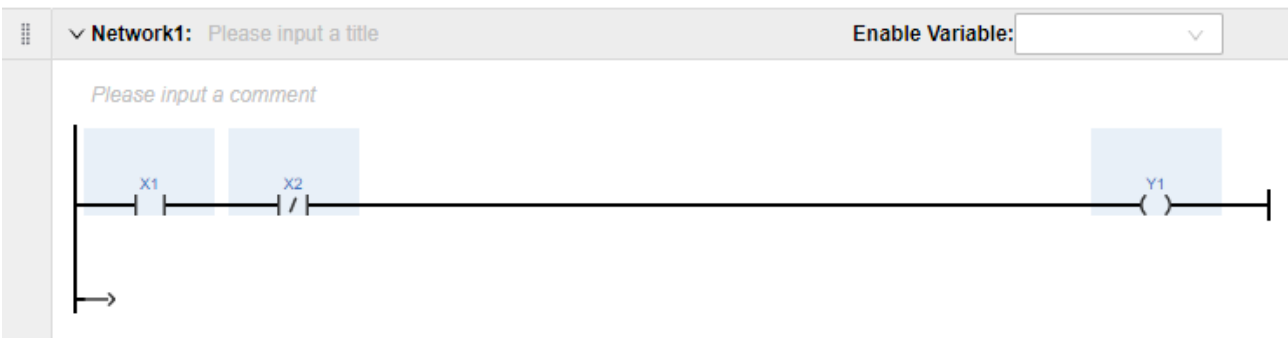
Please input a comment

X1
X2
Y1

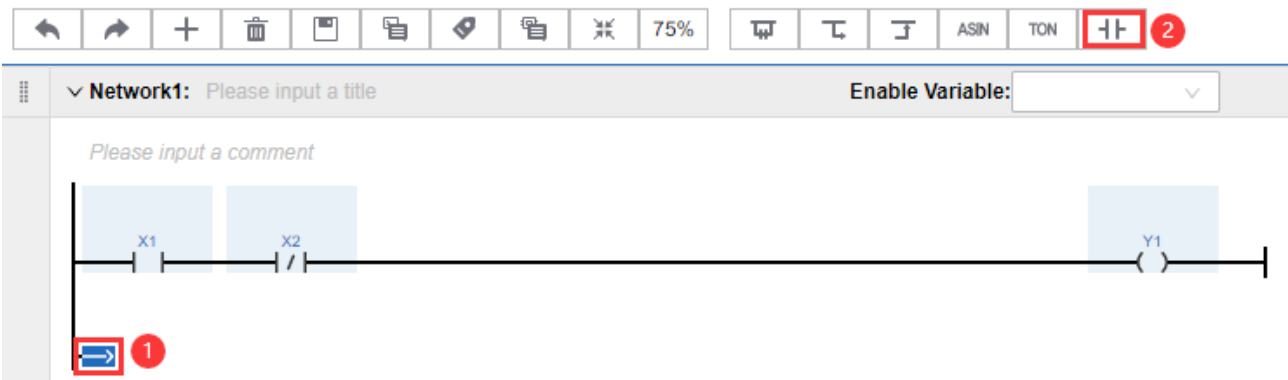
Step139. Select the line segment to the left of the normally open contact, click  icon, and create a new branch.



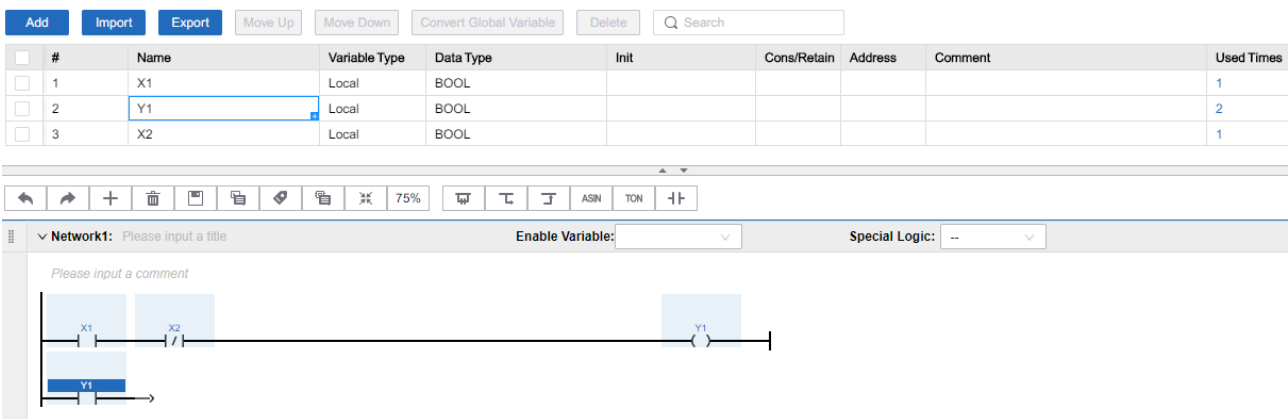
Step140. The result of the new branch is shown in the figure below.



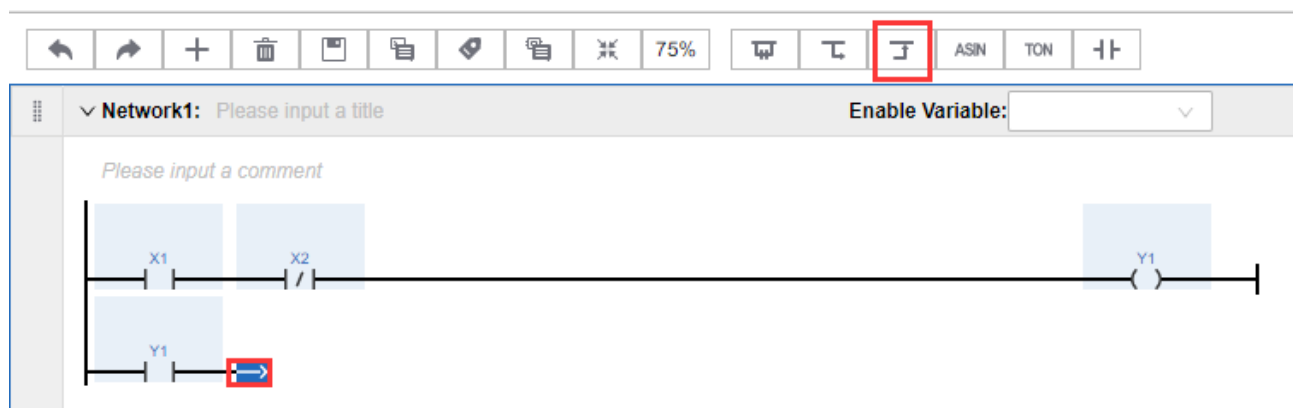
Step141. Select the arrow in the branch, click  icon, add a normally open contact, and bind the variable Y1 (Y1 has been declared).



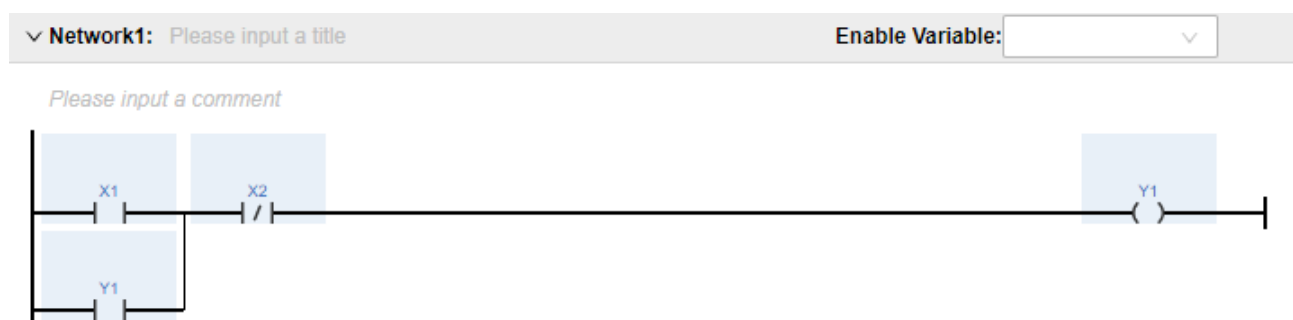
Step142. The result of adding the normally open contact Y1 is as follows.



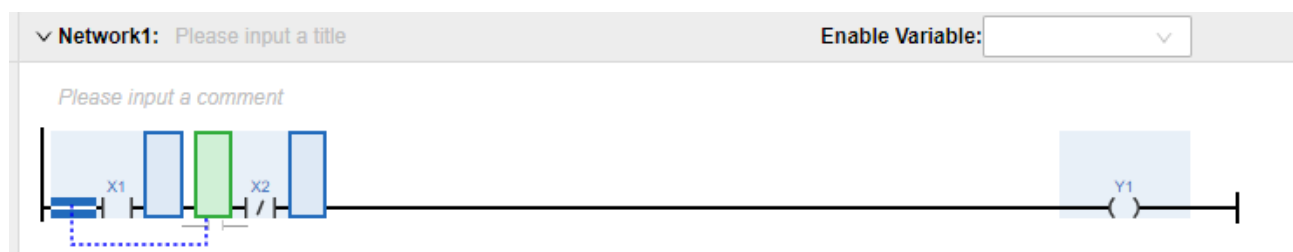
Step143. Select the arrow in the branch and click the icon  to merge the branch.



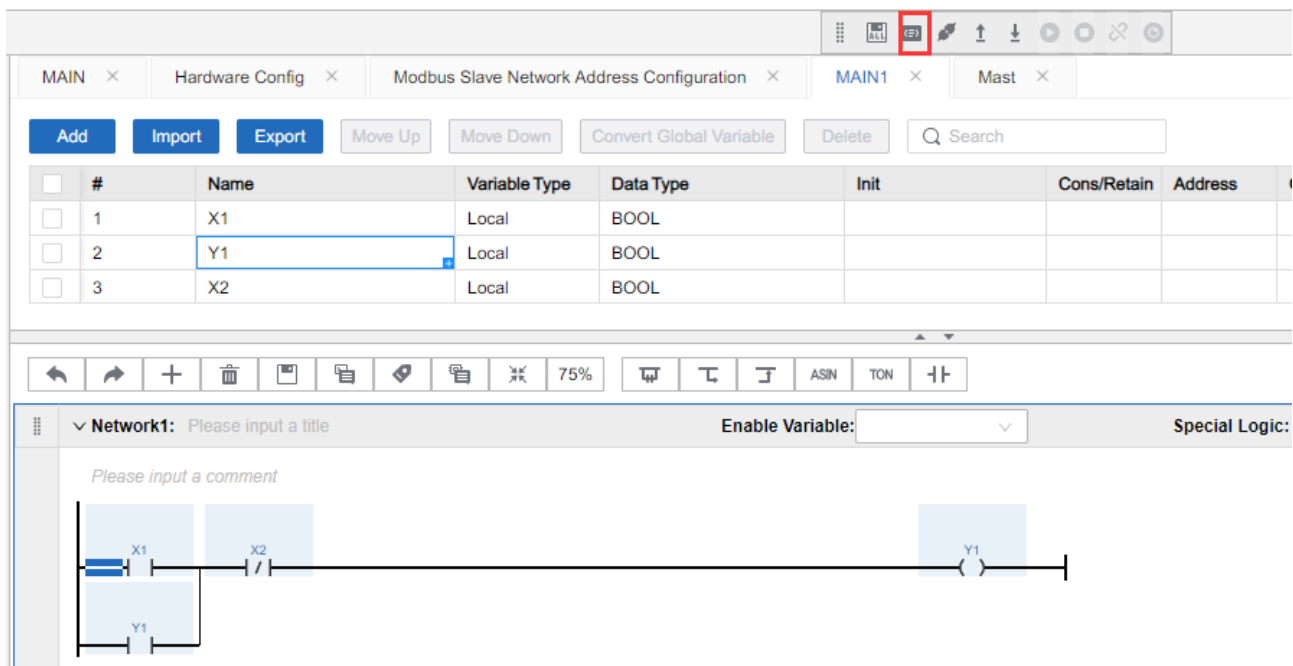
Step144. The result after merging branches is shown in the figure below.



MCR Master provides an easier parallel operation: simply hold down the left mouse button at the start point of the desired connection and drag it to the end point to complete the parallel operation of the Ladder Diagram.



Step145. Click the icon  in the upper toolbar to compile. If the compilation is successful, a simple starting, retaining, and stopping circuit program is created.



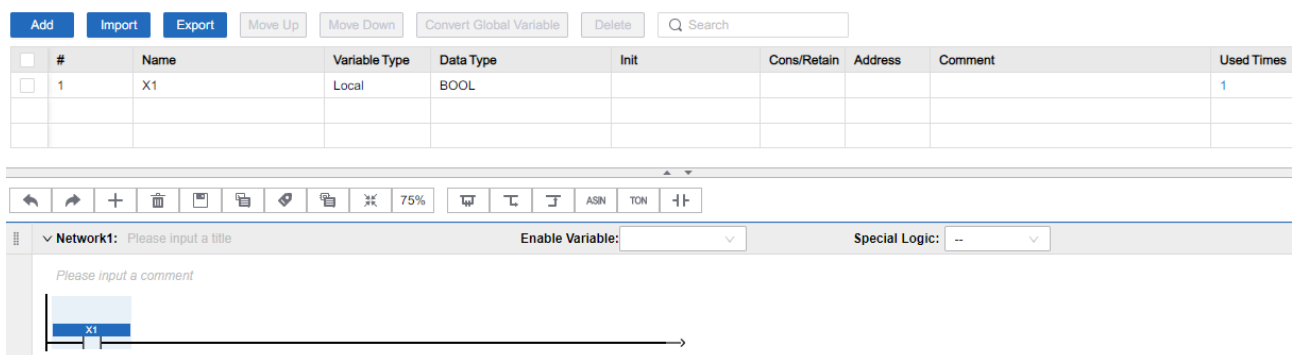
14.3.2 Call Counter

The counter is one of the basic components of the PLC system and is used to count and control equipment that generates pulse signals. The counter can count according to the input pulse signal and has a variety of counting modes, including up counting, down counting, up/down bidirectional counting, etc. In industrial automation production processes, PLC counters are widely used in control fields such as counting, positioning, and timing. By programming the PLC counter, a variety of complex counting control functions can be realized to achieve efficient optimization and fine control of the automated production process.

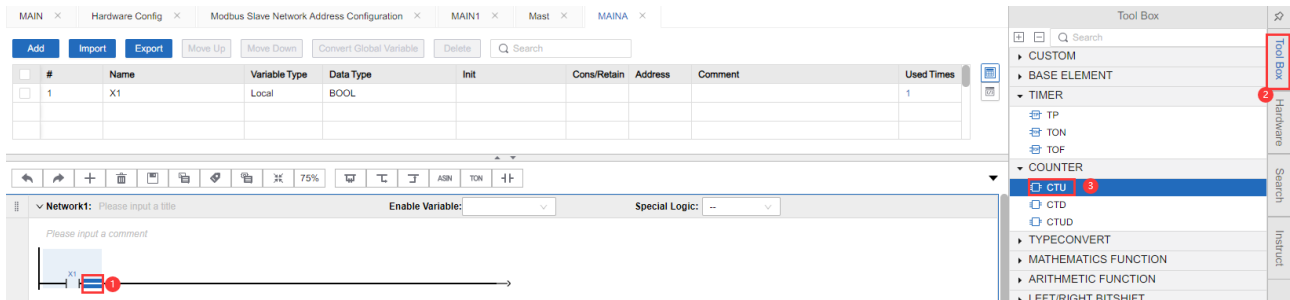
MCR Master provides a built-in counter function block, please refer to [Counter](#) for details.

The method to create a counter program is as follows:

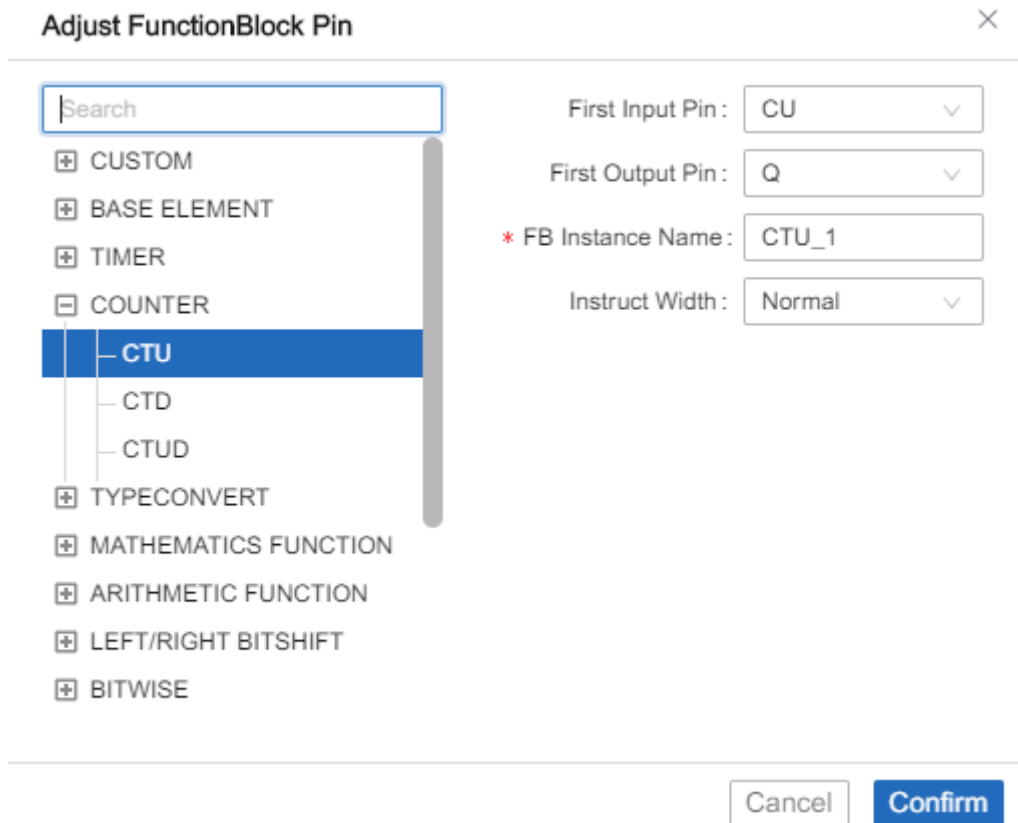
Step146. Add a normal open contact to the program segment and bind the variable X1.



Step147. Select the line segment on the right side of the normally open contact, click **Tool Box**, and drag the CTU counter in the toolbox to the selected line segment.



Step148. Configure the relevant parameters in the pop-up dialog box and click **Confirm**.



Please refer to the table below for detailed configuration methods.

Parameter	Description
First Input Pin	Select the input pin connected to the energy flow line.
First Output Pin	Select the output pin connected to the energy flow line.
FB Instance Name	The instance name of the function block is used to identify the function block. Supports English letters, numbers and underscores, and must start with an English letter.
Instruct Width	Normal: Normal width. Widen: The width of the graphic corresponding to the instruction is widened.

Step149. Bind variables to the counter pins (variables need to be declared, the variables bound to PV can be integer constants).

Add
Import
Export
Move Up
Move Down
Convert Global Variable
Delete
Search

#	Name	Variable Type	Data Type	Init	Cons
1	X1	Local	BOOL		
2	CTU_1	Local	CTU FB		
3	Reset	Local	BOOL		

75%
ASIN
TON

Network1: Please input a title
Enable Variable:

Please input a comment

Step150. Add a coil to the right of the counter and bind the variable Y1 .

Network1: Please input a title
Enable Variable:

Please input a comment

Step151. Click the icon in the toolbar to compile the program. If the compilation is successful, a program to call the count-up counter is created.

14.3.3 Call Timer

The timer is a function block built into the MCR Master and is used for timing. MCR Master provides three types of timers: pulse timer, Timer on-Delay, and Timer off-Delay. For more information about timers, refer to [Timer](#).

The operation method of creating a new timer program is as follows:

Step152. Add a normally open contact and a coil to program segment 1 in the Main program, declare variables X1 and Y1, and bind variable X1 to the normally open contact and variable Y1 to the coil.

Add Import Export Move Up Move Down Convert Global Variable Delete Search

#	Name	Variable Type	Data Type	Init	Cons.
1	X1	Local	BOOL		
2	Y1	Local	BOOL		

Please input a title: Network1 Enable Variable:

Please input a comment

Step153. Click **Tool Box** on the right side of the interface, and drag the TON in the expanded toolbox between the normal open contact and the coil.

Add Import Export Move Up Move Down Convert Global Variable Delete Search

#	Name	Variable Type	Data Type	Init	Cons/Retain	Address	Comment	Used Times
1	X1	Local	BOOL					0
2	Y1	Local	BOOL					0

Please input a title: Network1 Enable Variable: Special Logic: --

Tool Box Hardware Search Instruct

- CUSTOM
- BASE ELEMENT
- TIMER
 - TP
 - TON
 - TOF
- COUNTER
- TYPECONVERT
- MATHEMATICS FUNCTION
- ARITHMETIC FUNCTION
- LEFT/RIGHT BITSHIFT
- BITWISE
- SELECTION
- COMPARISON

Step154. Configure the function block instantiation name in the pop-up dialog box and click **Confirm**.

Adjust FunctionBlock Pin

✕

Search

- ✚ CUSTOM
- ✚ BASE ELEMENT
- ✚ TIMER
 - TP
 - TON
 - TOF
- ✚ COUNTER
- ✚ TYPECONVERT
- ✚ MATHEMATICS FUNCTION
- ✚ ARITHMETIC FUNCTION
- ✚ LEFT/RIGHT BITSHIFT
- ✚ BITWISE

First Input Pin:

First Output Pin:

* FB Instance Name:

Instruct Width:

Cancel

Confirm

Step155. Click at the PT pin on the left side of the TON_1 instance to add a TIME type variable (in this case add a Test_time1 variable) or a constant (e.g. T#2s or T#200ms). Add a Test_time2 variable at the ET pin.

Add Import Export Move Up Move Down Convert Global Variable Delete

#	Name	Variable Type	Data Type	Init	Cons/Retain	Address	Comment	Used Times
1	X1	Local	BOOL					1
2	Y1	Local	BOOL					1
3	TON_1	Local	TON FB					1
4	Test_time1	Local	TIME					1
5	Test_time2	Local	TIME					1

↶ ↷ + ✕ 📄 📁 🔍 75%

🔧 📏 📐 ASIN TON ↕

▼ Network1: Please input a title

Enable Variable:

Special Logic:

Please input a comment

Step156. Click the icon in the toolbar to compile. If the compilation is successful, a program that calls the timer is created.

14.3.4 Configure Interrupt Tasks

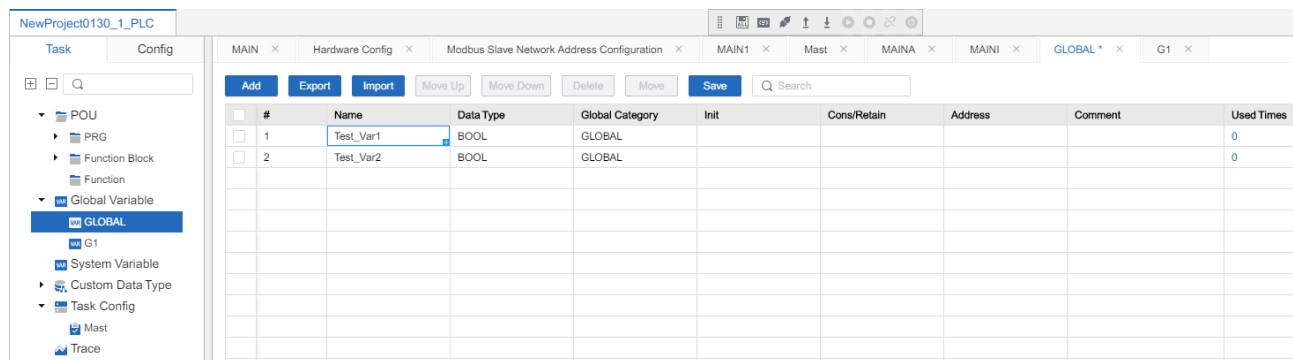
In the daily work, in accordance with the normal process to work, such as the occurrence of an emergency or a certain period of time apart to take care of another job, it is necessary to stop the work at hand to deal with emergencies or another job, which stops the current work is called an interruption, deal with emergencies or another

job, it is called an interruption of the program to deal with the problem and then come back to continue with the original work.

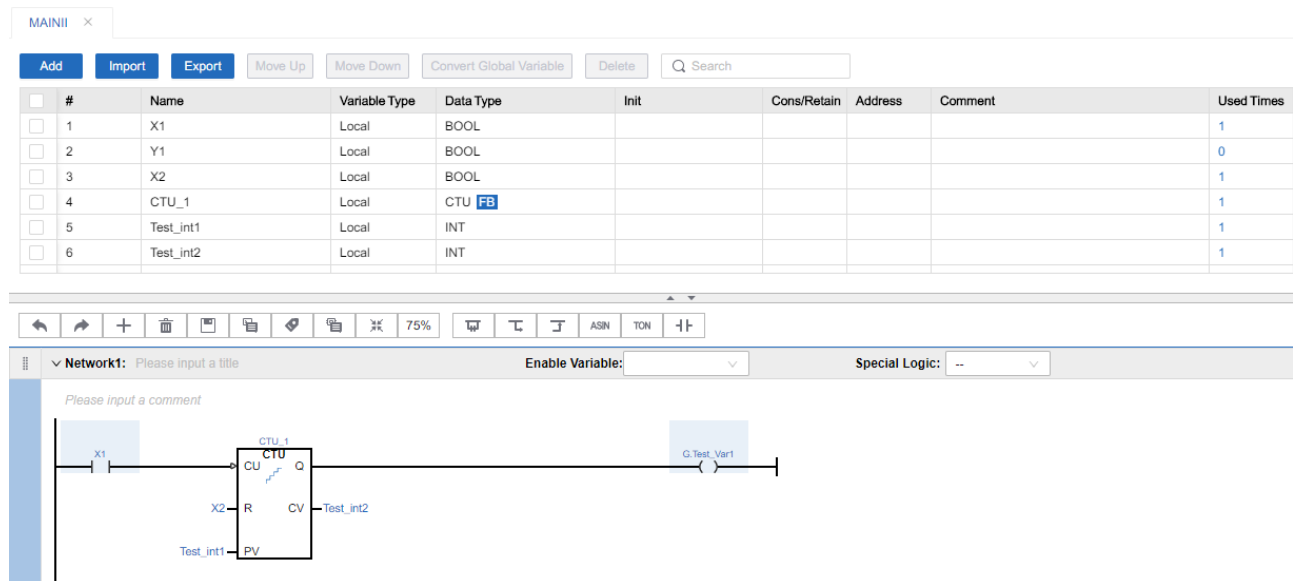
PLC also has the same interrupt, CPU normal execution of the program, such as the occurrence of the system identified in the middle of the action or parameter to meet the set requirements, then interrupt the program execution, to deal with the program set, and after the completion of program set, continue to execute the original program.

The operation method of creating a new interrupt task program is as follows:

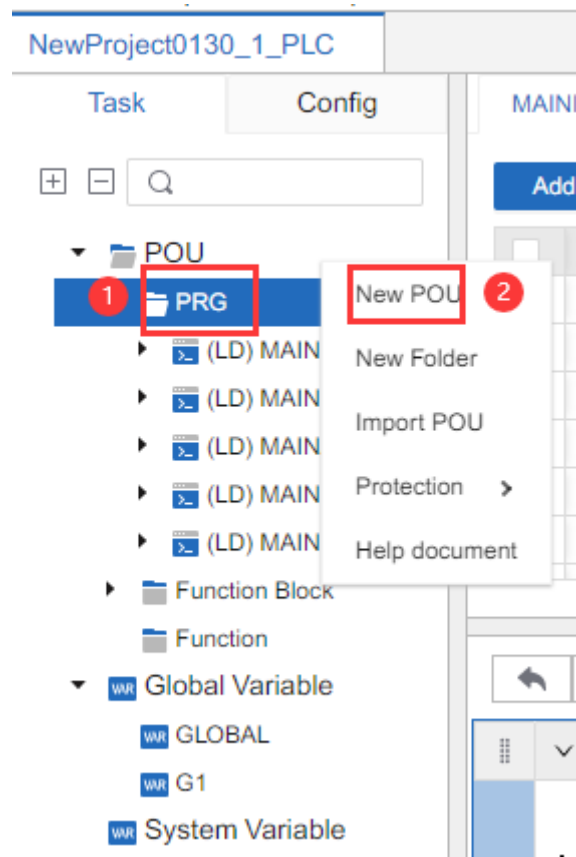
Step157. Click **GLOBAL** in the POU tree, click **Add**, and create two global variables Test_Var1 and Test_Var2.



Step158. Select the Main program and edit the program as shown below.



Step159. Right-click the **PRG** in the POU tree on the left and select **New POU** in the pop-up menu.



Step160. Configure the relevant parameters in the pop-up dialog box (refer to the figure below for the configuration method) and click **Confirm** .

New POU
✕

* Name:

* Type: ☒ PRG ☐ Function ☐ Function Block

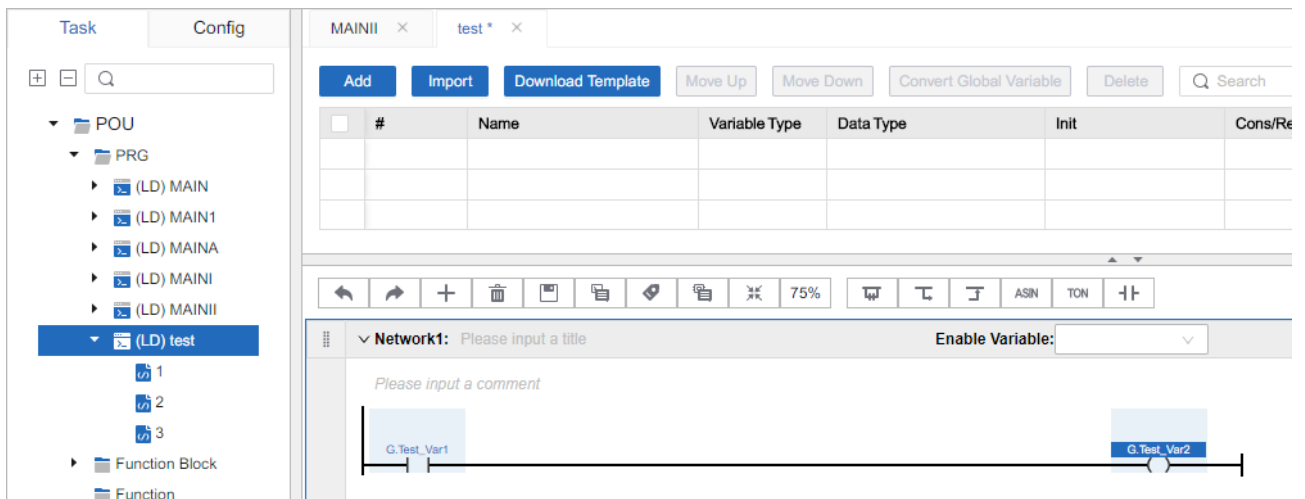
* Language: ☒ LD ☐ ST

Desc:

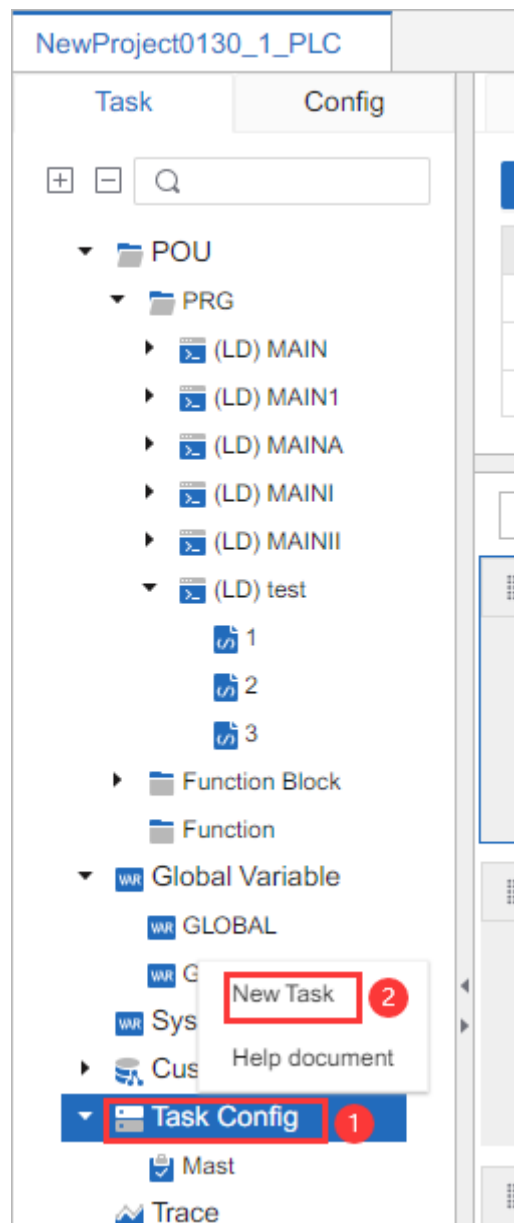
Cancel

Confirm

Step161. Create the following program in the Test program (the bound variables are all global variables, so there are no local variables).



Step162. Right-click **Task Configuration** in the left POU tree and select **New Task** from the pop-up menu.



Step163. Set the name to Test_1, the priority to 0 in the pop-up dialog box, and click **Confirm** .

New Task

*

Name :

Test_1

*

Priority :

0

Highest ▾

Cancel

Confirm

Step164. Click **Add Call** .

Task

Config

MAINII ×

test ×

Test_1 ×

Priority : 0

Highest ▾

Type : Circular ▾

Execution every : milliseconds

Add Call

Delete Call

View Call

☐	Execution Order (Drag to sort)	Program Name	Comment
No Data			

POU

PRG

(LD) MAIN

(LD) MAIN1

(LD) MAINA

(LD) MAINI

(LD) MAINII

(LD) test

1

2

3

Function Block

Function

Global Variable

GLOBAL

G1

System Variable

Custom Data Type

Task Config

Mast

Test_1

Trace

Step165. Check the Test program in the pop-up dialog box and click **Confirm**.

Add Call

☒	Name	Desc
☐	MAIN	
☐	MAIN1	
☐	MAINA	
☐	MAINI	
☐	MAINII	
1 ☒	test	

Cancel

2

Confirm

Step166. Set the **type** to “Event” and select “Test_Var1” and “R_TRIG” as the **trigger condition** .

Priority: 0 Highest ▾

Type: Event ▾

Trigger condition when variable: Test_Var1 ▾ R_TRIG ▾ edge

Add Call

Delete Call

View Call

☐	Execution Order (Drag to sort)	Program Name	Comment
☐	1	test	

Step167. Click the icon  to compile. If the compilation is successful, an interrupt task is configured. When the Main program is executed and the value of the Test_Var1 variable is TRUE, the Test_1 task will be called. The Test program configured in the Test_1 task will run, making the value of Test_Var2 also become TRUE.

15 Appendix A - Communication Code Description

Communication code	Description
01	Illegal function. The function code received in the query is an impermissible operation for the server (or slave), possibly because the function code is only valid for new devices and is not possible in the selected unit. It also indicates that the server (or slave) is handling this request in an error state, for example: it is unconfigured and a register value is required to be returned.
02	Illegal data address. The data address received in the query is an unallowable address for the server (or slave), in particular the combination of reference number and transmission length is invalid. For a controller with 100 registers, a request with offset 96 and length 4 will succeed, while a request with offset 96 and length 5 will generate exception code "02".
03	Illegal data value. The value included in the query is not allowed for the server (or slave). This value indicates a failure in the remaining structure of the combined request. For example: the implied length is incorrect. The Modbus protocol is not aware of the significance of any special value in any special register, and the data item submitted for storage in the register has a value other than that expected by the application.
04	The slave device failure. An unrecoverable error occurs while the server (or slave) is trying to perform the requested operation.
05	Confirm. Used in conjunction with programming commands, the server (or slave) has accepted the request and is processing the request, but it takes a long time to perform these operations. Returning this response prevents a timeout error from occurring in the client (or master). The client (or master) can continue to send polling program completion messages to confirm whether the processing is completed.
06	The slave device is busy. Used in conjunction with programming commands. The server (or slave) is processing a long duration program command. When the server (or slave) is idle, the master should retransmit the message later.
08	Stores parity errors. Used in conjunction with function codes 20 and 21 and reference type 6, it indicates that the extended file area cannot pass the consistency check. The server (or slave) tried to read the log file, but found a parity error in memory. The client (or master) can resend the request, but can request service on the server (or slave) device.

Communication code	Description
10	Unavailable Gateway Path. Used in conjunction with a gateway to indicate that the gateway cannot allocate an internal communication path from the input port to the output port for processing requests. Usually means that the gateway is misconfigured or overloaded.
11	Gateway target device response failed. Used in conjunction with a gateway to indicate that no response was obtained from the target device. Usually means that the device is not in the network.
238	An abnormal or invalid response, generally occurring when the Modbus-TCP Server side is not started properly.
239	The response is not supported and usually occurs when the other party's serial port is not opened.
240	Disconnect.
241	Time out.
1000	For Modbus Instruction only, Port is not enable.
1001	For Modbus Instruction only, illegal port value (legal values are 1, 2, 3, 4, 10).
1002	For Modbus Instruction only, illegal starting address(less than 1) (the starting address is less than 1).
1003	For Modbus Instruction only, Illegal communication data length, less then 1 or more then max value (1920 for BOOL or 120 for WORD).
1004	For Modbus Instruction only, 1X address does not allow writing (1X address is a read-only address).
1005	For Modbus Instruction only, 3X address does not allow writing (3X address is a read-only address).
1006	For Modbus Instruction only, the address type is illegal (types other than 0X, 1X, 3X, 4X).
1007	For Modbus Instruction only, the IP address is illegal when using the network port for communication.

16 Appendix B - Address Area Relationship Description

DWORD	WORD	BYTE	Bit							
MD0	MW0	MB0	MX0.7	MX0.6	MX0.5	MX0.4	MX0.3	MX0.2	MX0.1	MX0.0
		MB1	MX1.7	MX1.6	MX1.5	MX1.4	MX1.3	MX1.2	MX1.1	MX1.0
	MW1	MB2	MX2.7	MX2.6	MX2.5	MX2.4	MX2.3	MX2.2	MX2.1	MX2.0
		MB3	MX3.7	MX3.6	MX3.5	MX3.4	MX3.3	MX3.2	MX3.1	MX3.0

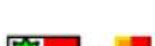
17 Appendix C - Motion Control Error Codes

Error code	Description
1	Invalid axis.
2	There is no return to the initial position. For example: Absolute motion is performed before executing the Home instruction.
3	Invalid window range. For example: when executing the ENC_Touch Probe instruction to open the window and first is greater than last, this error will be reported.
1000	Invalid axis.
1001	Internal anomalies.
1002	Unsupported motion instructions.
1003	The limit switch is triggered. For example: when the pulse axis reaches the soft/hard limit, execute the motion instruction.
1004	Not return to the homing position.
1005	Unsupported homing mode.
1006	Axis error.

18 Appendix D- Status Indicator

18.1 RUN Status Indicator

The RUN status indicator is on the front of the CPU body. Please refer to the table below for the meanings of different statuses.

Status		Meaning	
	OFF	normal	Out of service status
	Steady green		Running status
	Green flashing slowly (0.6Hz)		Pause status
	Green flashing fastly (5 Hz)		Initializing system or user program, need to wait
	Red flashing slowly (0.6Hz)	abnormal	User-level exceptions (configuration errors, abnormal operating status), can be hotfixed.
	Red flashing fastly (5 Hz)		System level error, need to restart to recover.
	Red ON		Hardware initialization error, needs to be returned to factory for repair.
	Orange ON		BOOT state. If it remains in this status, the firmware is damaged and needs to be returned to the factory for repair.
	Orange-red fast flashing (5 Hz)		Device certification failure. needs to be returned to the factory for repair.
	Orange-green slow flashing (0.6H z)		Upgrading firmware, need to wait.

18.2 ERR Status Indicator

The ERR status indicator is on the front of the expansion module. Please refer to the table below for the meanings of different statuses.

Status	Meaning
Steady red	Hardware error.
Steady green	Hardware function ready.
Green flashing	Receiving or sending data.
Steady orange	Factory mode (no IO board connected or unrecognized IO board connected or forced into factory mode).
Orange flashing	Authentication failed.
Orange-green flashing	There are errors (such as loss of analog calibration data).

19 Appendix E - Abbreviations

Abbreviation	Description
CNC	CNC (Computer Numerical Control, computer numerical control machine tool) is an automated machine tool controlled by a program. The control system can logically process programs with control codes or other symbolic instructions, and decode them via the computer, so that the machine tool can perform prescribed actions and process the blank into semi-finished products or finished parts via tool cutting.
CPU	CPU (Central Processing Unit), as the computing and control core of the computer system, is the final execution unit for information processing and program execution. Since the birth of CPU, CPU has made great progress in logical structure, operating efficiency and function extension. CPU is one of the main equipment of electronic computers and the core accessory in the computer. Its function is mainly to interpret computer instructions and process data in computer software. The CPU is the core component of the computer that reads instructions, decodes them, and executes them. CPU mainly consists of two parts, namely the controller and the arithmetic unit, which also include cache memory and the data and control bus that realize the connection between them. The three core components of an electronic computer are the CPU, internal memory, and input/output devices. The main functions of the central processing unit are to process instructions, perform operations, control time, and process data.
DCS	DCS (Distributed Control System) is a new generation of instrument control system based on microprocessors and adopts the design principles of decentralized control functions, centralized display operations, and taking into account separate autonomy and comprehensive coordination. It adopts the basic design ideas of decentralized control, centralized operation and management, and adopts a multi-layer hierarchical, cooperative and autonomous structure. Its main features are its centralized management and decentralized control. DCS has been widely used in various industries such as electric power, metallurgy, and petrochemicals.
IPC	IPC (Interprocess communication) is a set of programming interfaces that allow programmers to coordinate different processes so that they can run simultaneously in an operating system and transfer and exchange information with each other. This enables a program to handle many user requests at the same time. Because even if only one user issues a request, it may cause multiple processes to run in an operating system, and the processes must communicate with each other. The IPC interface provides this possibility. Each IPC method has its own advantages and limitations, and generally it is uncommon for a single program to use all IPC methods.

Abbreviation	Description
	IPC methods include pipe, message queuing, semaphores, shared memory, and sockets.
Modbus RTU	Modbus RTU protocol is an open communication protocol mainly based on serial link (RS232C or RS485). RTU is the abbreviation of “Remote Terminal Unit” in English, which is “remote terminal equipment”. It supports a variety of electrical interfaces, such as RS-232, RS-485, etc., and can also be transmitted on various media, such as twisted pair cables , fiber optic, wireless, etc. In the Modbus RTU protocol, the two parties of communication are called “master station” and “slave station”. The master station will issue a query or write command to the slave station, and then the slave station passively receives the command, feeds back the corresponding data result or executes the write command according to the function code and register number. An RS 485 network can theoretically have up to 254 slave stations. In practical applications, considering line loss and interference, the number will generally not exceed 100. Otherwise, it is recommended to use Ethernet for communication.
PLC	PLC (Programmable Logic Controller, Programmable Logic Controller) is a digital computing operation electronic system specially designed for application in industrial environments. It uses a programmable memory to store instructions for performing operations such as logical operations, sequence control, timing, counting and arithmetic operations, and controls various types of mechanical equipment or production process via digital or analog input and output.
POU	POU (Programming Organization Unit) is an important and widely used basic IEC programming unit in the IEC61131-3 standard. POU is composed of three types of basic units: Program, Function Block, and Function.
PTO	PTO (Pulse Train Output) is to output a specified number of square wave pulse trains with a duty cycle of 50%. In high-speed pulse train output mode, only the pulse period value and pulse number can be changed.
PWM	PWM (Pulse Width Modulation) is an analog control method that modulates the bias of the transistor base or MOS tube gate according to changes in the corresponding load to change the conduction time of the transistor or MOS tube, thereby achieving Changes in the output of a switching regulated power supply. This method can keep the output voltage of the power supply constant when the working conditions change. It is a very effective technology for using the digital signal of the microprocessor to control the analog circuit. Widely used in many fields such as measurement, communication, power control and conversion
RS485	Typical serial communication standard. The RS-485 bus standard stipulates the electrical characteristic standards of the bus interface, that is, the definition of two logical states: the

Abbreviation	Description
	positive level is between +2V and +6V, indicating a logical state; the negative level is between -2V and -6V, it represents another logical state; the digital signal adopts differential transmission method, which can effectively reduce the interference of noise signals.
RTC	RTC (Real Time Clock) is an integrated circuit, often called a clock chip. Real-time clock chip is one of the most widely used consumer electronic products in daily life. It provides people with accurate real-time time, or provides an accurate time reference for electronic systems. Most real-time clock chips use a high-precision crystal oscillator as the clock source. Some clock chips need to be powered by an external battery in order to still work when the main power supply is cut off.
SCADA	SCADA (Supervisory Control And Data Acquisition, data acquisition and monitoring control system) has a wide range of applications. It can be used in discrete process manufacturing plants, as well as in data acquisition and process control in electric power, metallurgy, petroleum, chemical industry, gas, railway and other fields. MX On CORPORATION SCADA system provides a web access interface, supports cross-platform deployment, and provides unlimited data access capabilities. Perform aggregate statistics and analysis on raw data to deeply explore the value of the data. Supports MQTT and OPC UA data push capabilities, and provides an open API interface to support docking with IT and OT devices . Based on data processing capabilities, it provides a variety of applications that meet the needs of the factory to improve the operation and management level of the factory.