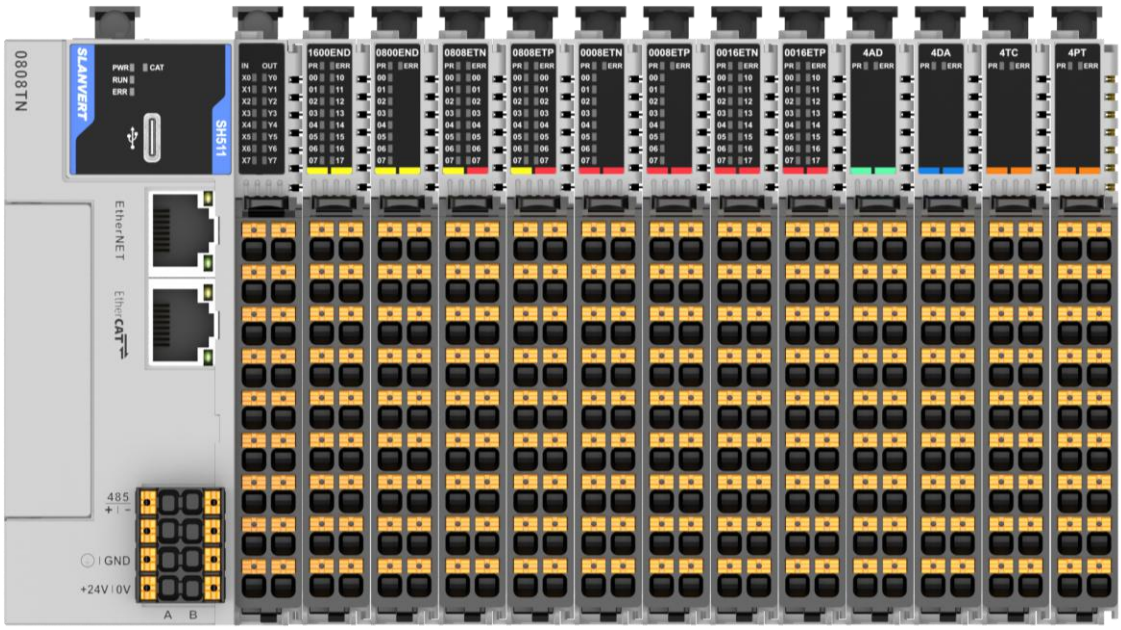
SLANVERT

SH100/300/SH500 PLC Programming and Application Manual



Hope Senlan Science and Technology Holding Corp., Ltd.

Preface

Brief

- The SH Series PLC, independently developed by SLANVERT, is a new-generation compact controller featuring EtherCAT communication for real-time data transmission and robust motion control capabilities, and its flexible I/O extension enables adaptable system configuration.
- FB/FC encapsulation improves engineering efficiency by reducing repetitive development.
- With multi-protocol interfaces (RS485, CAN, Ethernet, EtherCAT), the SH series adapts to diverse network architectures.
- Available in four models (SH100/SH300/SH500), the SH series mid-sized controllers address automation demands requiring compact dimensions, multi-axis control, temperature regulation, and industrial networking.
- This manual details programming fundamentals, quick setup, bus communication, motion control, and high-speed counter implementation for SH300/SH500 series PLCs.

Version Change Log

Date	Version	Content
202502	V1.0	First version issued
202506	V1.1	Update SH100

Manual Acquisition

This manual is not shipped with products. To obtain the PDF file, please:

- Log on to the official website of SLANVERT (https://www.SLANVERT.com.cn), Download", search for keywords and download the PDF file.

Catalog

1 General

1.1 Introduction

1.1.1 Product Introduction

The SH series is SLANVERT’s next-gen compact PLC. It supports EtherCAT bus communication and multi-level networking through RS485, CAN, Ethernet, and EtherCAT interfaces. With robust motion control, distributed I/O control, and FB/FC functions for process encapsulation and reuse, it is widely applicable in industrial automation to ensure efficient operation, precise control, and seamless communication for equipment.

The SH series compact controllers include four models (SH100/SH200/SH300/SH500), addressing diverse requirements for small-to-medium-sized automation equipment. Optimized for space-constrained environments requiring multi-axis motion control, precision temperature regulation, and industrial networking, the SH series controllers deliver excellent performance in providing robust control for efficient and stable operation of various automated systems.

1.1.2 Software Introduction

AutoSoft, SLANVERT's programming software for small PLCs, features an intuitive programming and debugging interface. It enables flexible communication and control implementation and is compatible with multiple programming languages like Ladder Diagram (LD), Sequential Function Chart (SFC), and Structured Text (ST), catering to different users' programming habits and needs.

AutoSoft's Features:

- Flexible Communication: Supports PLC communication via COM, USB, and Ethernet interfaces, enabling efficient data exchange while facilitating remote operation and collaborative debugging;
- Comprehensive Network Support: Configurable for Modbus and CANopen protocols, simplifying multi-protocol integration and enhancing workflow efficiency.
- Precision Motion Control: Offers a wide range of motion control instructions, such as axis positioning, electronic cam, interpolation, and flying/chasing shear, meeting intricate automation requirements.
- Diverse Debugging: Provides multiple debugging functions, including motion trajectory graph, monitoring, online modification, oscilloscope, and fault diagnosis, shortening debugging and troubleshooting time.
- IP Protection System: Implements password-protected upload/download protocols, device authentication, and access restrictions to prevent unauthorized acquisition and operations and secure user's intellectual property.

1.1.3 SH Series Specifications

(I) SH100 Series Specifications

Name		Specification/Description	
I/O Configuration	Max. I/O points	16(8 input/8 output)	
	Extension module quantity	None	
User File Capacity	User program capacity	16K steps	
	Data capacity	2K Bytes are saved at power outage;	
	Data block size	8000 D components, 32K R components	
Command Processing Speed	Basic commands	0.2μs/each	
	Application commands	A few μs to several hundred μs/each	
Soft Component Resources	I/O points	8input/8 output (X0~X7 input, Y0~Y7 output)	
	Auxiliary relay	2048 (M0~M2047)	
	Local auxiliary relay	64 (LM0~LM63)	
	Special auxiliary relay	512(SM0~SM511)	
	Status relay	1024 (S0~S1023)	
	Timer	255(T0~T255) (1) 100ms accuracy: T0~T209 (2) 10ms accuracy: T210~T251 (3) 1ms accuracy: T252~T255 (4) 1ms accuracy: DTON/DTOF/DTACR (function blocks)	
	Counter	264 (C0~C263) (1) 16-bit increment counter: C0~C199 (2) 32-bit increment/decrement counter: C200~C235 (3) 32-bit high-speed counter: C236~C263	
	Data register	8000 (D0~D7999),	
	Local data register	64 (V0~V63)	
	Variable addressing register	16 (Z0~Z15)	
	Special data register	512 (SD0~SD511)	
Interrupt Resources	External input interrupt	16 (trigger edge settable, for X0~X7 rising/falling edges)	
	High-speed counter interrupt	8	
	Internal timing interrupt	3	
	Serial interrupt	6	
	PTO output completion interrupt	3	
	High-speed comparison interrupt	8	
Communication	Ports	1 RS232(COM0) 1 RS485(COM1)	
	Protocol	Modbus, free port,	
Special Functions	High-speed counter	X0~X7	X1/X2: 50kHz，X2-X7:10kHz Total input frequency: <60kHz
	High-speed pulse output	Y0~Y7	Y0/Y1/Y2: mx. 100kHz Y4-Y7: normal output
	Digital filtering	X0~X7 use digital filtering, 0-60ms	
	Subroutine call	Up to 64 user subroutines, with 6-level nesting. Local variables support, max. 16 parameter passes per subroutine, and variable aliases supported	
Special Functions	User program protection	Upload password	3 formats, ≤8 alphanumeric characters, case-sensitive
		Download password	
		Clock password	
		Subroutine encryption	≤16 alphanumeric characters, case-sensitive
		Other protection	Format and upload inhibition supported
	Programming	AutoSoft programming software	Run on IBM PC or compatible PC

(II) SH300 Series Specifications

Table 1-1 SH300 Series Product Specifications

Name		Specification/Description	
I/O Configuration	Max. I/O points	512 (256 input/256 output)	
	Extension module qty.	Total I/O and special modules ≤ 16	
User File Capacity	User program capacity	200K steps	
	Data capacity	128K Bytes for user soft components, ~84K Bytes battery-backed; 1M Bytes for custom variables, 128K Bytes battery-backed	
	Data block size	8000 D components, 32K R components	
Command Processing Speed	Basic commands	0.039μs/each	
	Application commands	A few μs to several hundred μs/each	
Soft Component Resources	I/O points	512 input/512 output (X0~X777 input, Y0~Y777 output)	
	Auxiliary relay	10240 (M0~M10239)	
	Local auxiliary relay	64 (LM0~LM63)	
	Special auxiliary relay	1024 (SM0~SM1023)	
	Status relay	4096 (S0~S4095)	
	Timer	512 (T0~T511) (1) 100ms accuracy: T0~T209 (2) 10ms accuracy: T210~T255 (3) 1ms accuracy: T256~T511 (4) 1ms accuracy: DTON/DTOF/DTACR (function blocks)	
	Counter	264 (C0~C263) (1) 16-bit increment counter: C0~C199 (2) 32-bit increment/decrement counter: C200~C235 (3) 32-bit high-speed counter: C236~C263	
	Data register	8000 (D0~D7999), 32768 (R0~R32767), 32768 (W0~W32767)	
	Local data register	64 (V0~V63)	
	Variable addressing register	16 (Z0~Z15)	
	Special data register	1024 (SD0~SD1023)	
Interrupt Resources	External input interrupt	16 (trigger edge settable, for X0~X7 rising/falling edges)	
	High-speed counter interrupt	8	
	Internal timing interrupt	3	
	Serial interrupt	6	
	PTO output completion interrupt	8	
	High-speed comparison interrupt	16 (for local counter commands)	
Communication	Ports	1×asynchronous serial port (0): RS485, 1×USB port, 1×CAN port, 2×Ethernet ports, 2×extended serial ports (RS-485)	
	Protocol	Modbus, free port, N:N (SLANVERT dedicated), supports 1:N and N:N networks	
Special Functions	High-speed counter	X0~X7	200kHz * 8 channels
	High-speed pulse output	Y0~Y7	200kHz * 8 independent channels (transistor output type only)
	Digital filtering	X0~X7 use digital filtering, others use hardware filtering	
	Subroutine call	Up to 64 user subroutines, with 6-level nesting. Local variables support, max. 16 parameter passes per subroutine, and variable aliases supported	
Special Functions	User program protection	Upload password	3 formats, ≤8 alphanumeric characters, case-sensitive
		Download password	
		Clock password	
		Subroutine encryption	≤16 alphanumeric characters, case-sensitive
		Other protection	Format and upload inhibition supported
	Programming	AutoSoft programming software	Run on IBM PC or compatible PC

(Ⅲ) SH500 Series Specifications

Table 1-2 SH500 Series Product Specifications

Name		Specification/Description	
I/O Configuration	Max. I/O points	512 (256 input/256 output)	
	Extension module qty.	Total I/O and special modules ≤ 16	
User File Capacity	User program capacity	200K steps	
	Data capacity	128K Bytes for user soft components, ~84K Bytes battery-backed; 1M Bytes for custom variables, 128K Bytes battery-backed	
	Data block size	8000 D components, 32K R components	
Command Processing Speed	Basic commands	0.039μs/each	
	Application commands	A few μs to several hundred μs/each	
Soft Component Resources	I/O points	512 input/512 output (X0~X777 input, Y0~Y777 output)	
	Auxiliary relay	10240 (M0~M10239)	
	Local auxiliary relay	64 (LM0~LM63)	
	Special auxiliary relay	1024 (SM0~SM1023)	
	Status relay	4096 (S0~S4095)	
	Timer	512 (T0~T511) (1) 100ms accuracy: T0~T209 (2) 10ms accuracy: T210~T255 (3) 1ms accuracy: T256~T511 (4) 1ms accuracy: DTON/DTOF/DTACR (function blocks)	
	Counter	264 (C0~C263) (1) 16-bit increment counter: C0~C199 (2) 32-bit increment/decrement counter: C200~C235 (3) 32-bit high-speed counter: C236~C263	
	Data register	8000 (D0~D7999), 32768 (R0~R32767), 32768 (W0~W32767)	
	Local data register	64 (V0~V63)	
	Variable addressing register	16 (Z0~Z15)	
	Special data register	1024 (SD0~SD1023)	
Interrupt Resources	External input interrupt	16 (trigger edge settable, for X0~X7 rising/falling edges)	
	High-speed counter interrupt	8	
	Internal timing interrupt	3	
	Serial interrupt	6	
	PTO output completion interrupt	8	
	High-speed comparison interrupt	16 (for local counter commands)	
Communication	Ports	1×asynchronous serial port (0): RS485, 1×USB port, 1×CAN port, 2×Ethernet ports, 2×extended serial ports (RS-485) 1×EtherCAT communication port, supporting up to 48 real axes, 64 total real and virtual axes, and up to 72 EtherCAT slaves.	
	Protocol	Modbus, free port, N:N (SLANVERT dedicated), supports 1:N and N:N networks	
Special Functions	High-speed counter	X0~X7	200kHz * 8 channels
	High-speed pulse output	Y0~Y7	200kHz * 8 independent channels (transistor output type only)
	Digital filtering	X0~X7 use digital filtering, others use hardware filtering	
	Subroutine call	Up to 64 user subroutines, with 6-level nesting. Local variables support, max. 16 parameter passes per subroutine, and variable aliases supported	
Special Functions	User program protection	Upload password	3 formats, ≤8 alphanumeric characters, case-sensitive
		Download password	
		Clock password	
		Subroutine encryption	≤16 alphanumeric characters, case-sensitive
		Other protection	Format and upload inhibition supported
	Programming	AutoSoft programming software	Run on IBM PC or compatible PC

1.1.4 Networking Solutions

(I) SH300 Series Networking Solutions

The SH300 series features CAN, Ethernet, and RS485 interfaces for multilevel network communication, supporting diverse applications. It integrates 4 high-speed inputs and 4 high-speed outputs, enabling 4-axis pulse output control and 4-axis encoder counting, as shown in Figure 1-3.

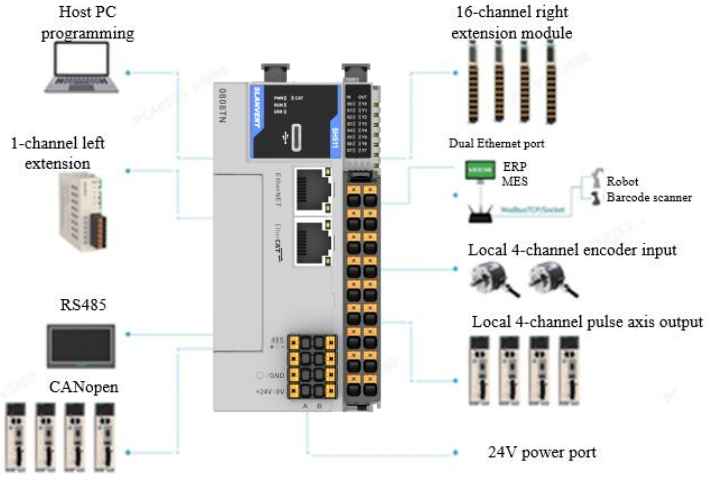


Figure 1-3 Typical Application Topology

(II) SH500 Series Networking Solutions

The SH500 series features EtherCAT, CAN, Ethernet, and RS485 interfaces for multilevel network communication, supporting diverse applications. It integrates 4 high-speed inputs and 4 high-speed outputs, enabling 4-axis pulse output control and 4-axis encoder counting, as shown in Figure 1-4.

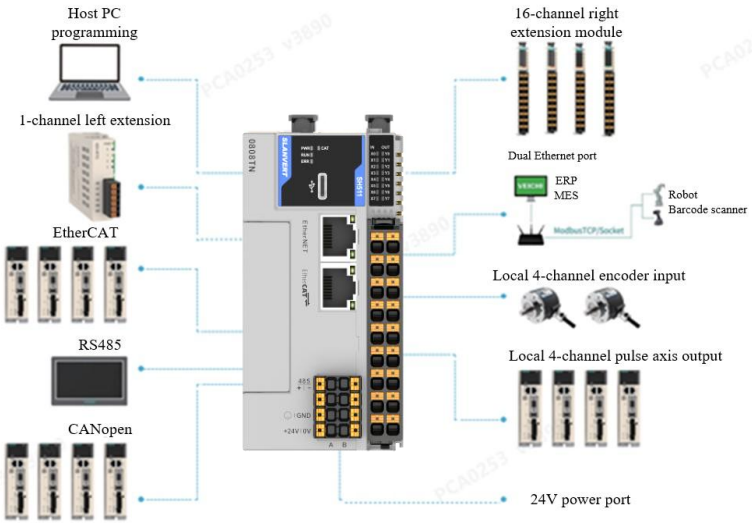


Figure 1-4 Typical Application Topology

1.2 Software Acquisition and Installation

1.2.1 Acquisition

AutoSoft software is available free of charge, please log on to the official website of SLANVERT Electric (https://www.SLANVERT.com/en/), "Services and Support- Software Download", search for keywords and download the installation package.

Note

- SLANVERT continuously improves its products and updates technical data. For optimal application, please maintain current software versions and consulting the latest documentation as needed.

1.2.2 Installation Requirements

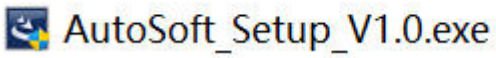
The PC for installing AutoSoft should meet the following specification:

Item	Specification
OS	Windows 7/10 (64-bit recommended)
CPU Processor Speed	4GHz minimum
Memory	4GB minimum
HDD	5GB minimum

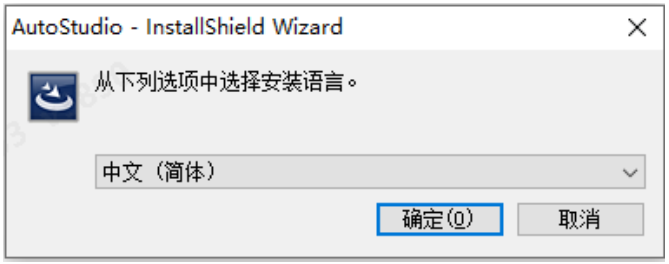
1.2.3 Installation Steps

Get the latest installation package from SLANVERT's official website. Here take installing VCPLC_Setup_V1.12.10.1 on Windows 10 as an example.

(1) Unzip the "VCPLC_Setup_V1.12.10.1.zip" package.



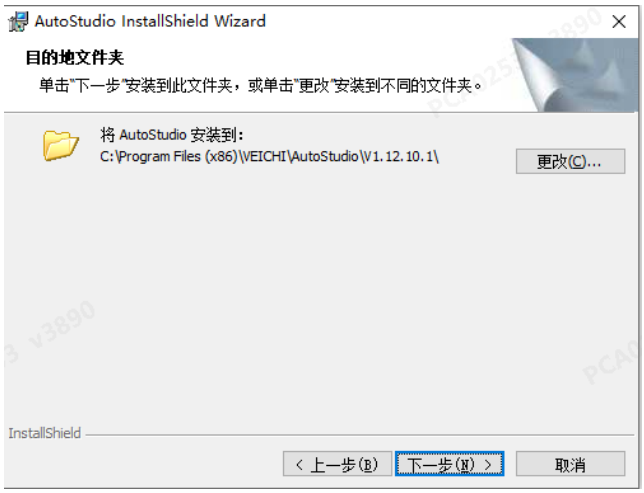
(2) Double-click "AutoSOft_Setup_V1.0.exe", select language in the pop-up box, and then click “OK”.



(3) Click the “Next”.

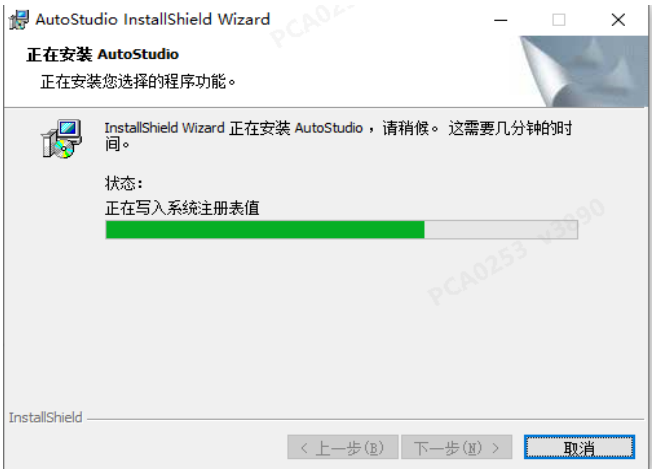
(4) Click the “Next”.

(5) Click "Change" to select the installation path if needed (default recommended), and then click "Next".

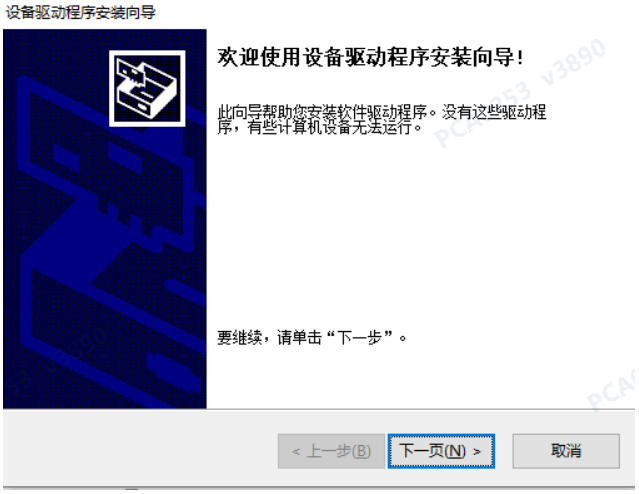


(6) Click the “Install”.

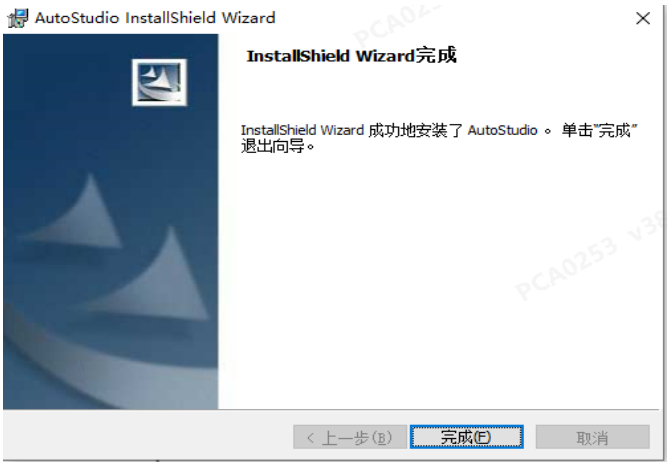
(7) Wait for the installation.



(8) Then, a "USB Driver" dialog box will pop up, and click the “Next”.



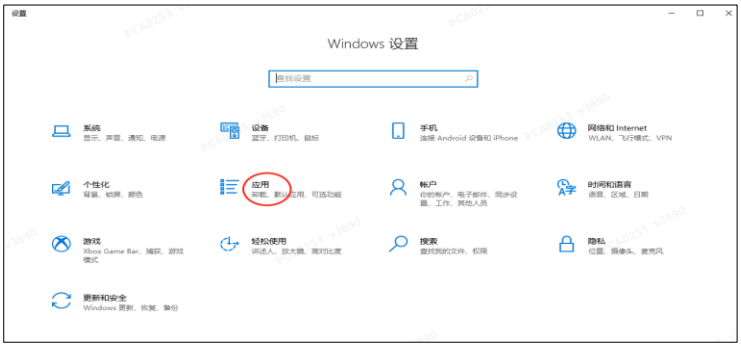
(9) After the "USB Driver" is installed, click "Finish" to complete the software installation.



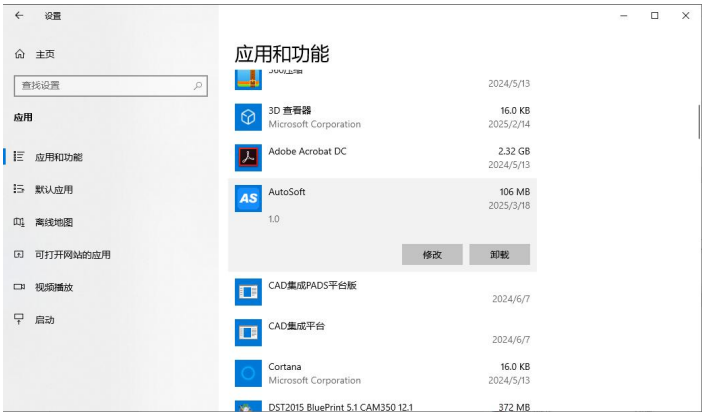
1.2.4 Uninstallation Steps

Take uninstalling VCPLC_Setup_V1.12.10.1 on Windows 10 as an example:

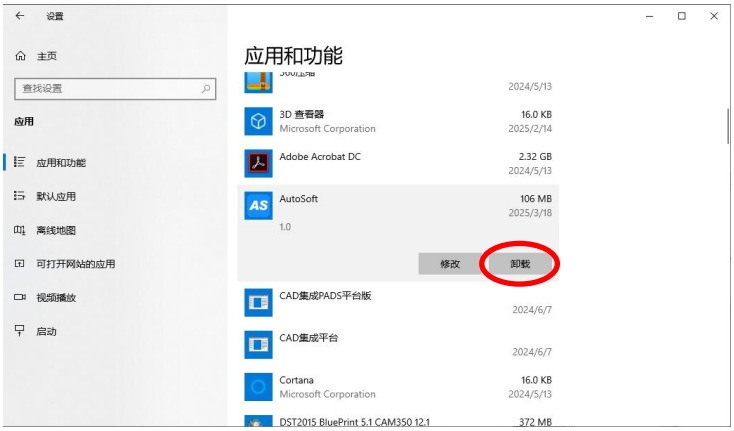
(1) Click on the desktop, then click in the opened interface to access the "Windows Settings" interface.



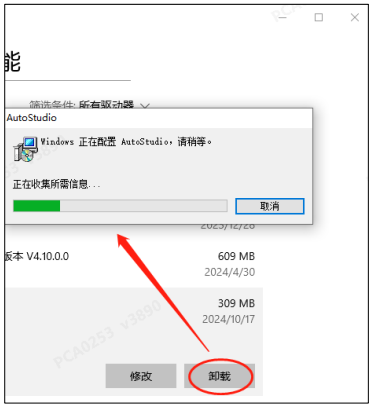
(2) Click "Apps" to open the "Settings" dialog.



(3) Click "Uninstall" to open the "AutoSoft Uninstall Wizard" dialog.

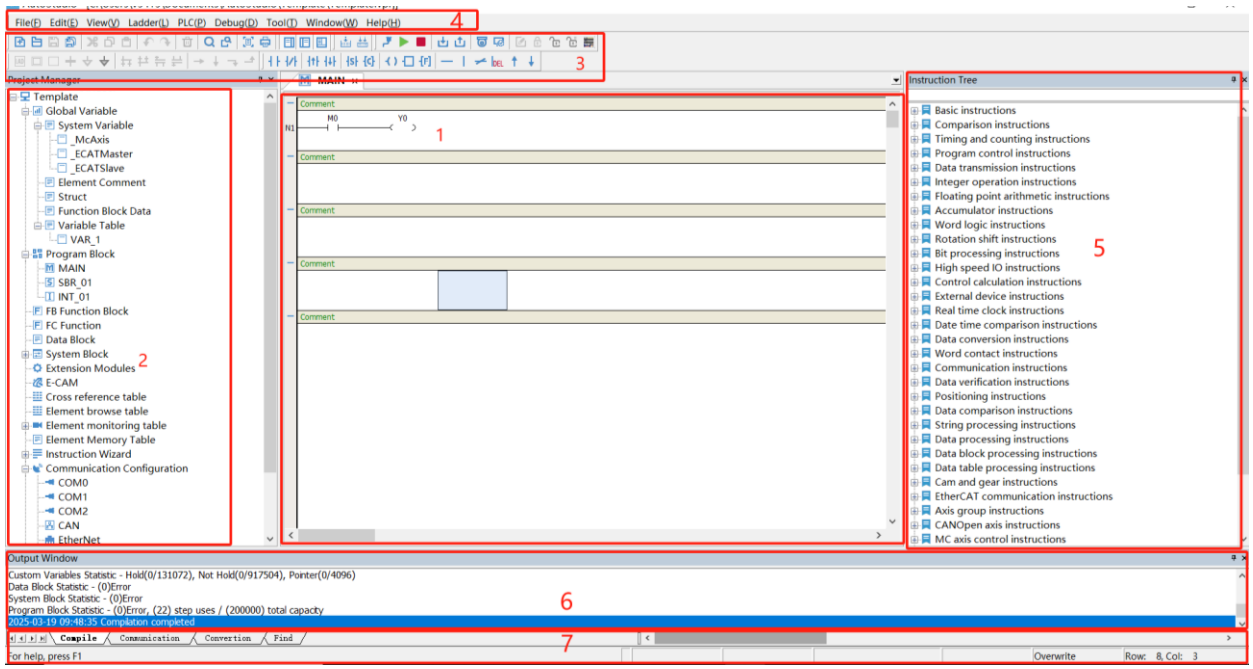


(4) Locate and click "AutoSoft Vx.x.x.x" in the "Apps & Features" interface, then click "Uninstall", and wait for the uninstallation to complete.



1.2.5 AutoSoft Main Interface

The main interface consists of: Menu Bar, Toolbar, Project Manager, Program Editing Area, Instruction Tree Window, Output Window, Status Bar.



Description of software main interface:

No.	Function	Description
①	Program Editing Area	Edits user programs.
②	Project Manager	Includes parameter management, variable management, program management,

	Area	and configuration management for PLC projects.
③	Toolbar	Provides shortcuts for file management and programming/debugging tools.
④	Menu Bar	Contains settings for programming, debugging, communication, etc.
⑤	Instruction Tree	Displays loaded slaves and instruction sets supported by the selected PLC. It includes Ladder Diagram and ST toolboxes, which switch based on the current editing interface. The toolboxes differ only in their supported instruction sets.
⑥	Output Window	Displays compilation status, communication status, find results, etc.
⑦	Status Bar	Shows PLC status, fault status, PLC firmware version, scan cycle, etc.

2 QuickStart

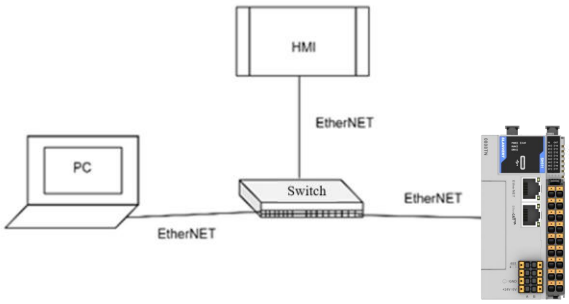
2.1 General

This section helps users quickly grasp programming and debugging through simple examples and instructions of common programming and debugging functions.

2.2 PC-PLC Communication Connection

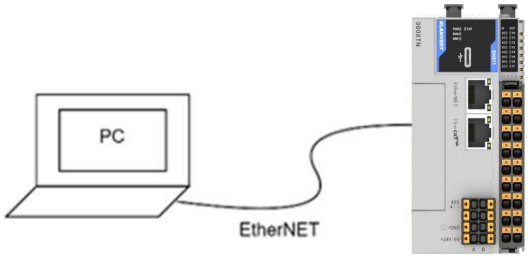
2.2.1 Overview

A PC with AutoSoft can communicate with a PLC via USB or Ethernet for program upload/download, monitoring, and debugging. Ethernet-based connection supports hub/switch connections or direct links, enabling both multidrop networks and dedicated point-to-point configurations.



2.2.2 Ethernet-based Connection

Ethernet-based PC-PLC communication requires target PLC selection and IP/device name configuration.

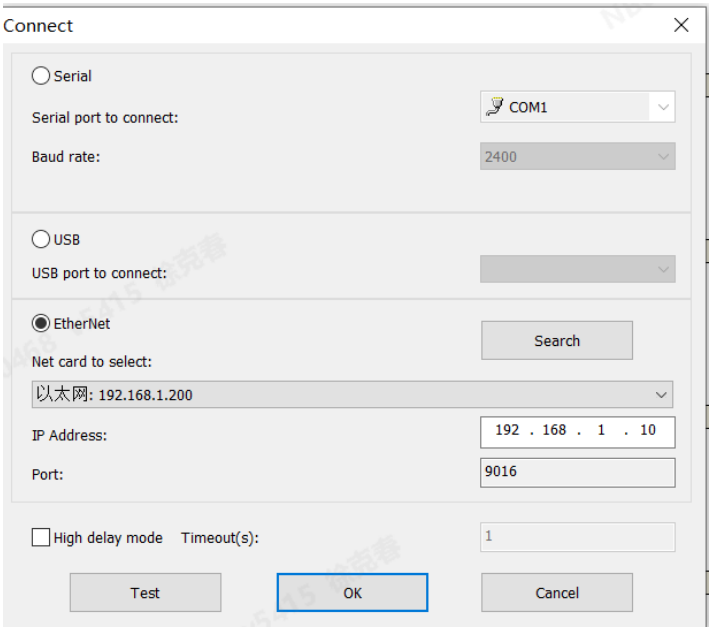


Target PLC Connection

Target PLC connection via Ethernet involves two configuration scenarios.

(1) If the PLC IP is known, configure the communication type and device IP, then connect.

1. Connect the PLC to the PC with an Ethernet cable.
2. Double-click the “AutoSoft” shortcut to open it.
3. Navigate to Tools > PLC Communication > Connection Settings in the menu bar, or click in the toolbar to open the “Connect” box.



4. Configure parameters in the “Connect” box:

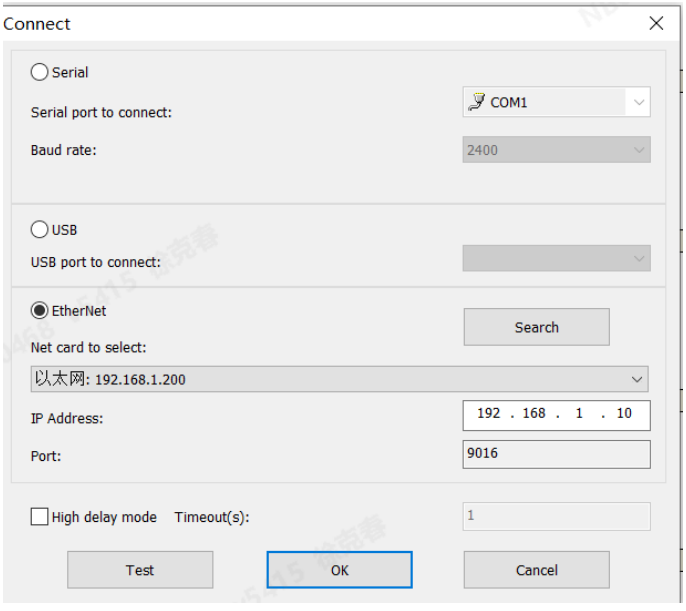
■Select “Ethernet” and set the connected net card;
■Enter the PLC’s IP address in “Device IP”.
■Click “OK” to check connection success.

Note:

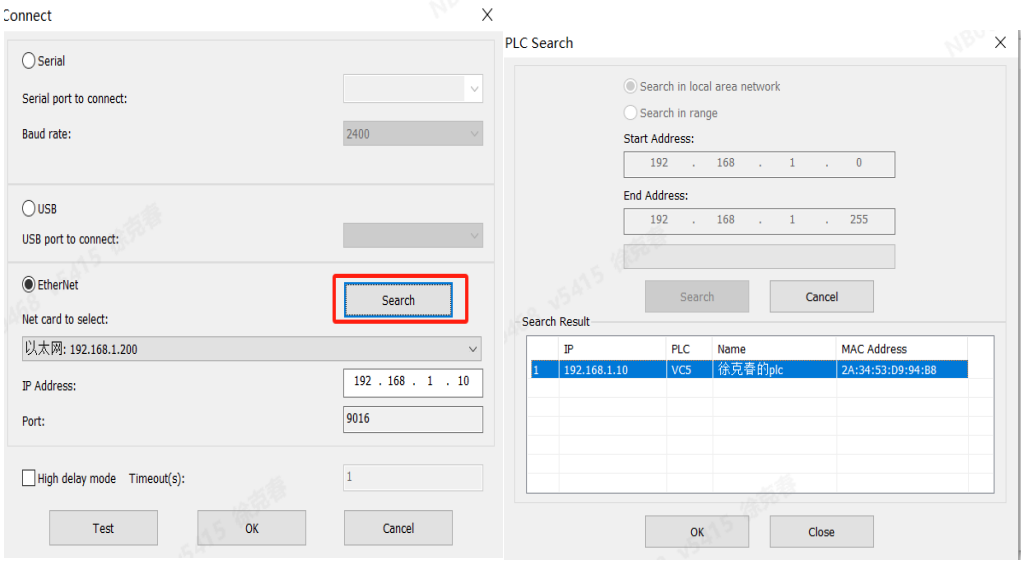
- Click “PING” to test the network connection between the PC and PLC.
- Confirm the PC's IP resides in the same subnet.
- Select the correct network address if multiple exist.

(2) If the PLC IP is unknown, use the search function to find the target PLC device.

1. Connect the PLC to the PC with an Ethernet cable.
2. Double-click the “AutoSoft” shortcut to open it.
3. Navigate to Tools > PLC Communication in the menu bar, or click in the toolbar to open the “Connect” box.



4. Select “Ethernet” in “Connect” box and click “Search” to find PLCs on the local network.



■When the PLC and PC are connected through a switch, only PLCs in the same subnet as the PC can be found.
■When the PLC and PC are directly connected via an Ethernet cable, PLCs in the same or different subnets can be found.

5. Select the target IP from the search results and click “OK” to auto-fill it to the “Remote IP” field. Click “OK” again to establish the connection.

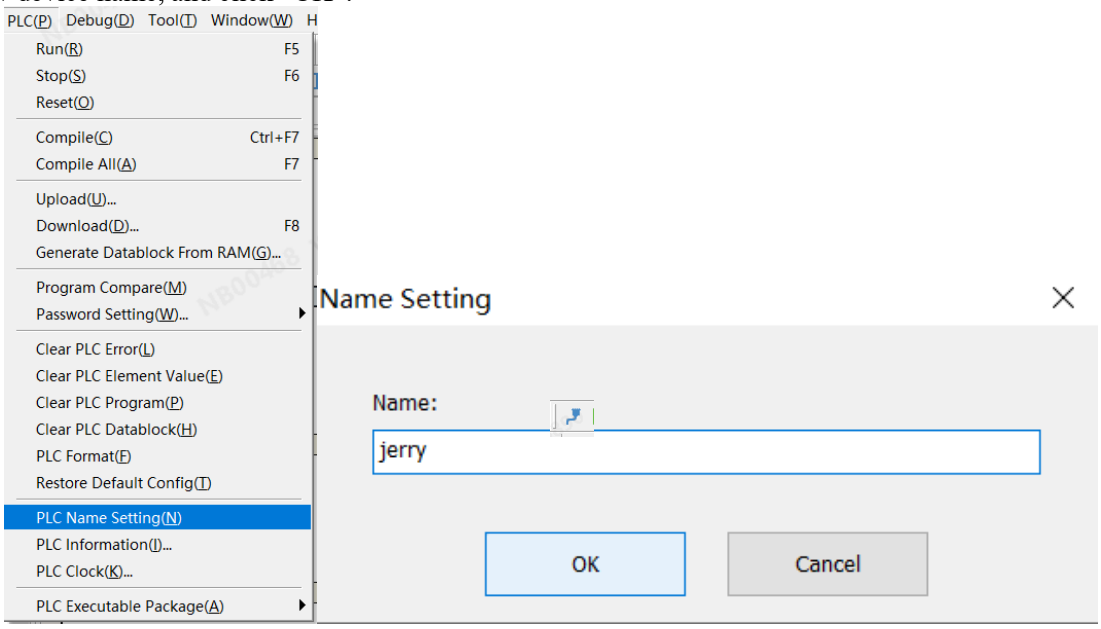
Note:

- Click “PING” to test the network connection between the PC and PLC.

PLC IP Address/Name Modification

Users can modify the PLCs’ IP address and name as needed to differentiate them.

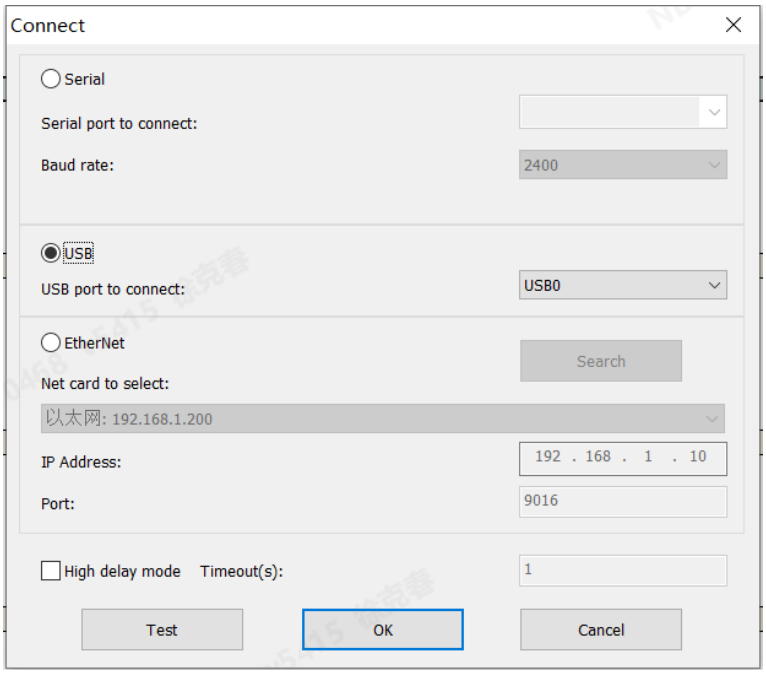
1. After connecting to the target PLC, click the “PLC(P)” in the menu bar, then click “PLC Name Setting”, enter the new device name, and click “OK”.



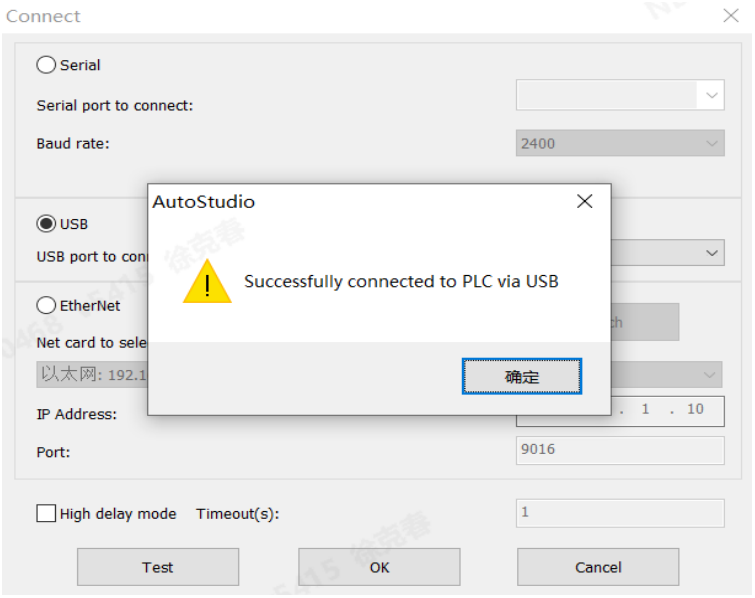
2.2.3 USB-based Connection

Target PLC Connection

1. Connect the PLC to the PC with an USB cable.
2. Double-click the “AutoSoft” icon to open it.
3. Navigate to Tools > PLC Communication in the menu bar, or click in the toolbar to open the “Connect” box.

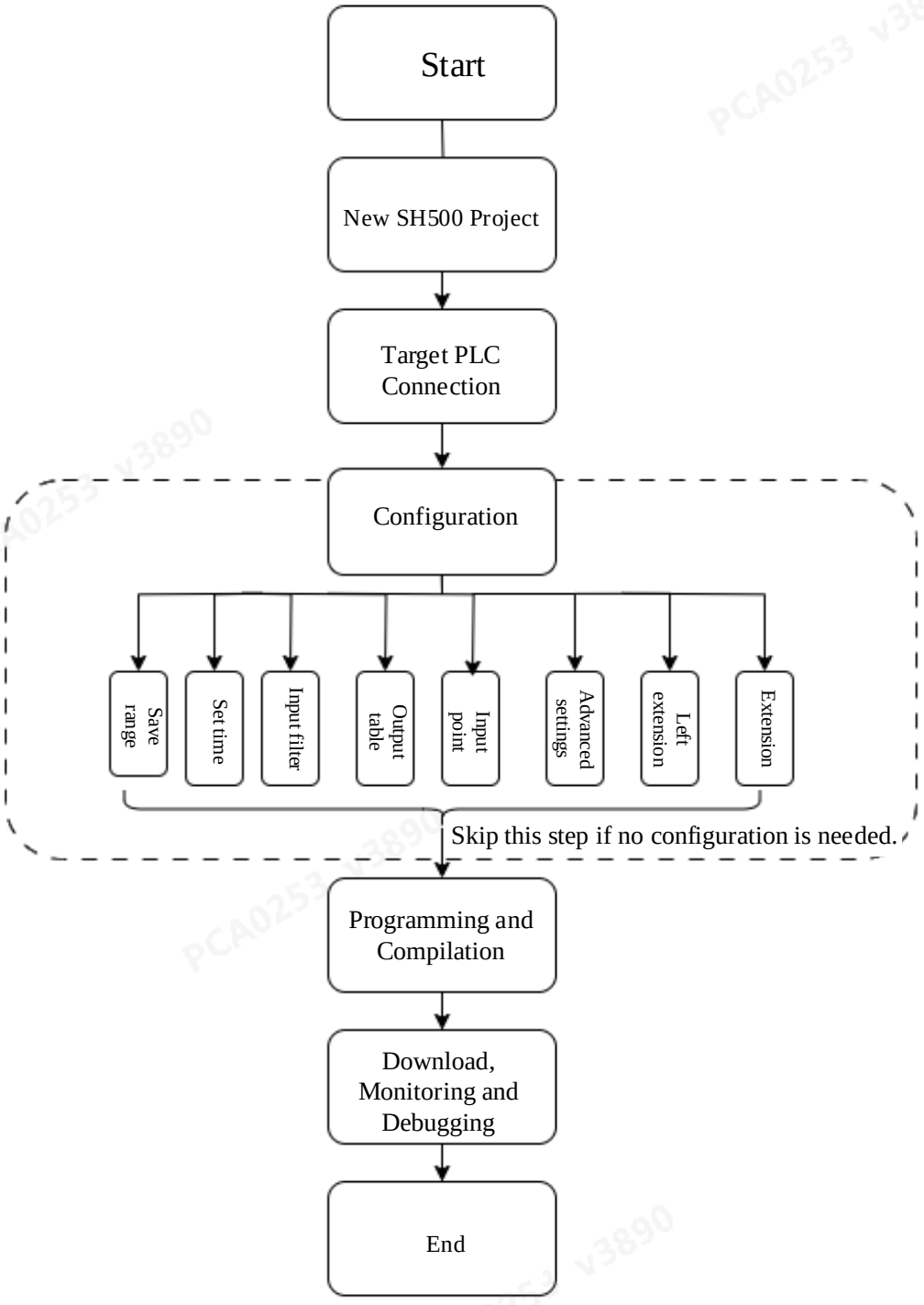


4. Select “USB” in “Connect” box and click “OK” to check connection success.



2.3 Programming Process

The figure below illustrates the programming and debugging workflow for a typical SH500 series PLC application.



Note:

- *Configuration steps are optional when operating with the SH host unit alone.

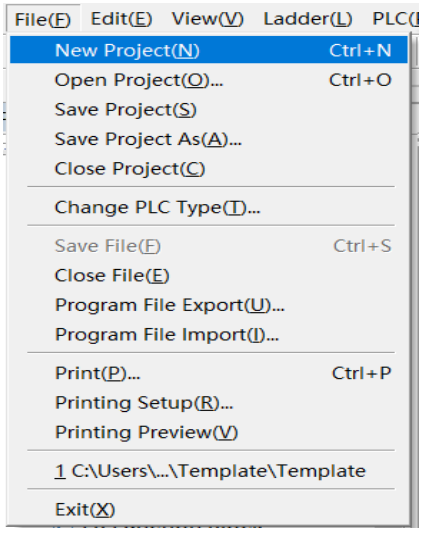
2.4 Programming Example

2.4.1 Requirements

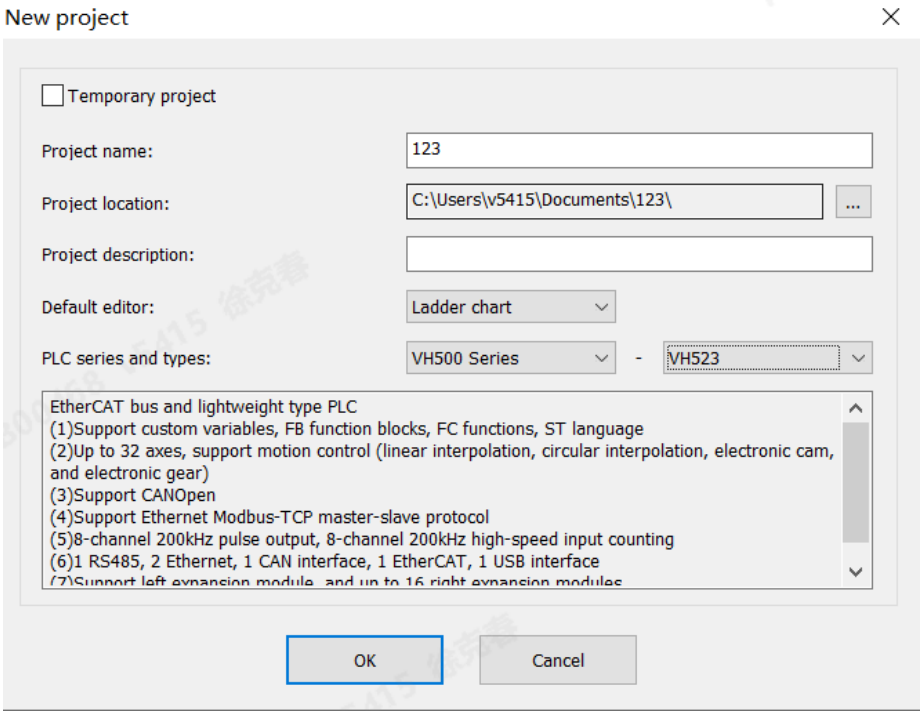
Design two control buttons (X0 and X1): When X0 is pressed, Y0 output is ON; when X1 is pressed, Y0 output is OFF.

2.4.2 New Project

1. Double-click the AutoSoft shortcut to open it. Click “File>New Project” in the menu bar or click in the toolbar.



2. Select the default editor and select SH523 as the PLC type in the pop-up box. Enter the project name and select the project location, then click “OK” to create the project and enter the main project interface.



Note:

- Projects created and saved with higher AutoSoft versions cannot be opened with lower versions.
- Projects created with lower versions can be opened with higher versions, but their version will be upgraded and cannot be reopened with lower versions.

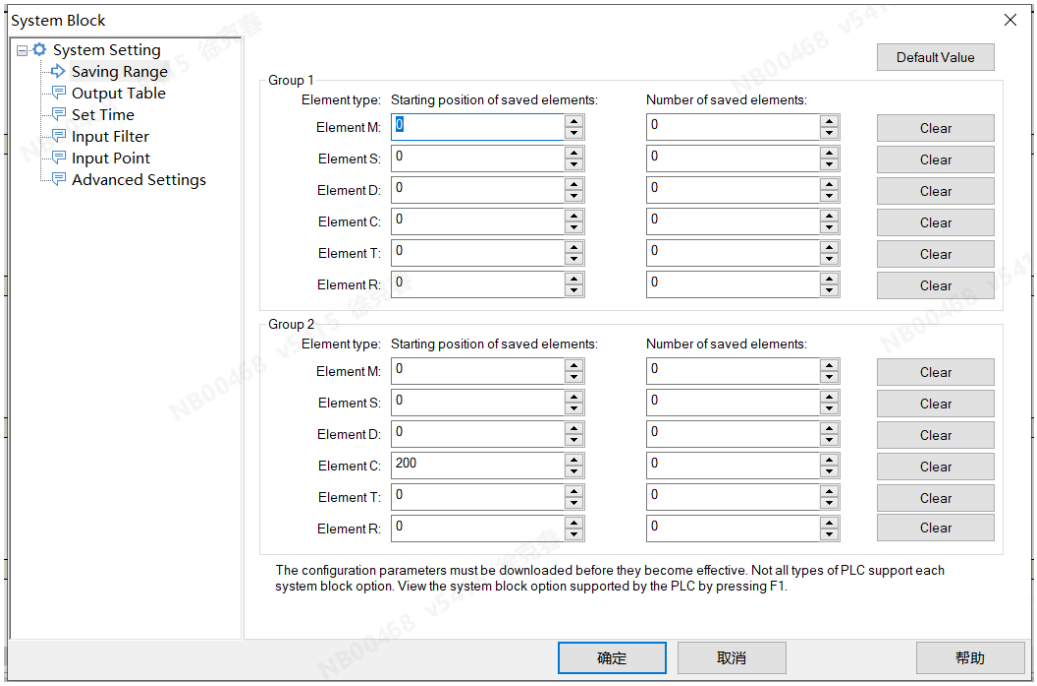
2.4.3 Target PLC Connection

Here take Ethernet-based connection as an example (refer to 2.2.2 Ethernet-based Connection for details).

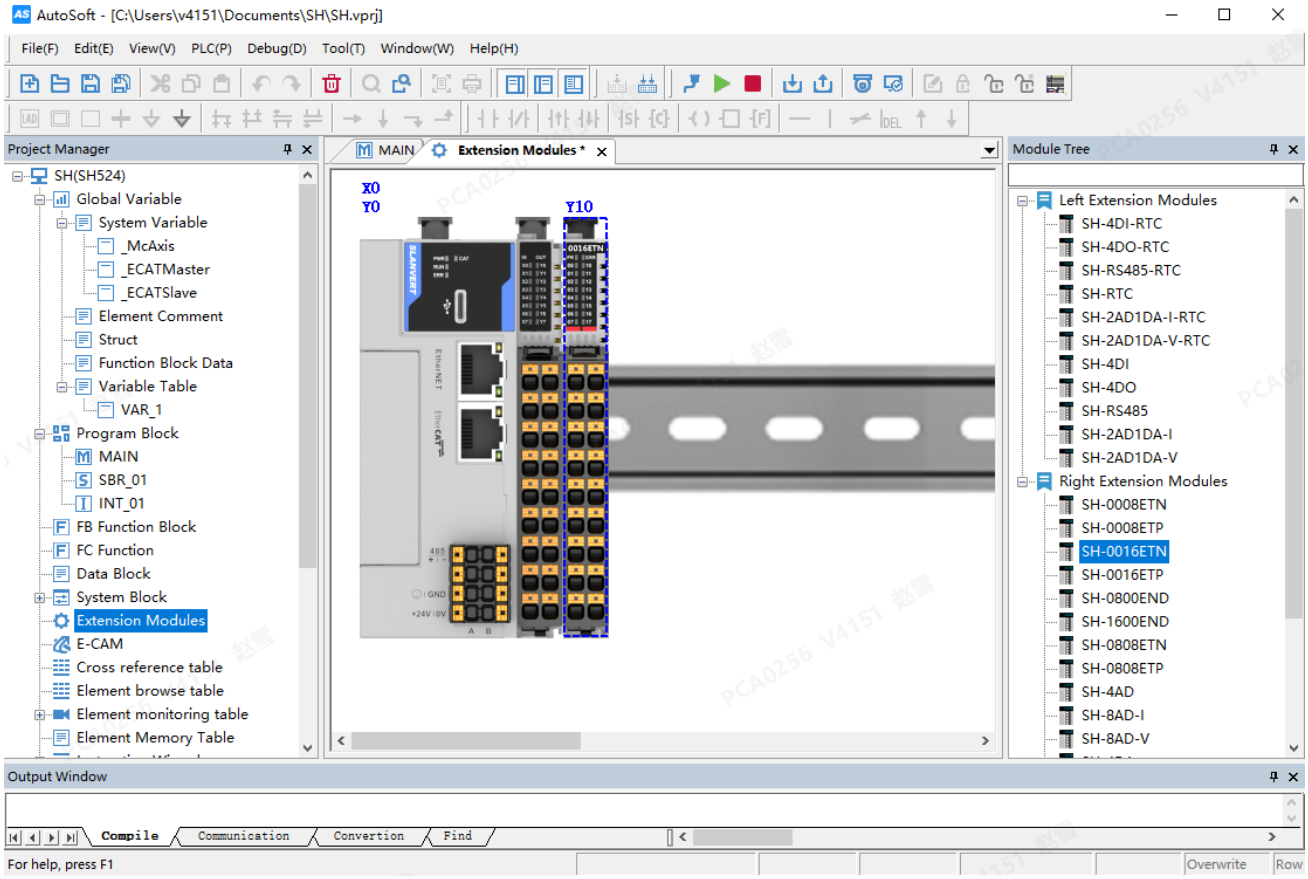
2.4.4 System Configuration (Optional)

For SH523 PLC configuration (Saving Range, Constant Scanning Time Setting, Data Block Validity, Left/Right Extension Modules), modify settings in [Project Manager]>[System Block] and download to apply. Configuration of left/right extension modules requires auto-scan or manual module addition via [Project Manager]>[Extension Modules]. Skip this step if no configuration is needed.

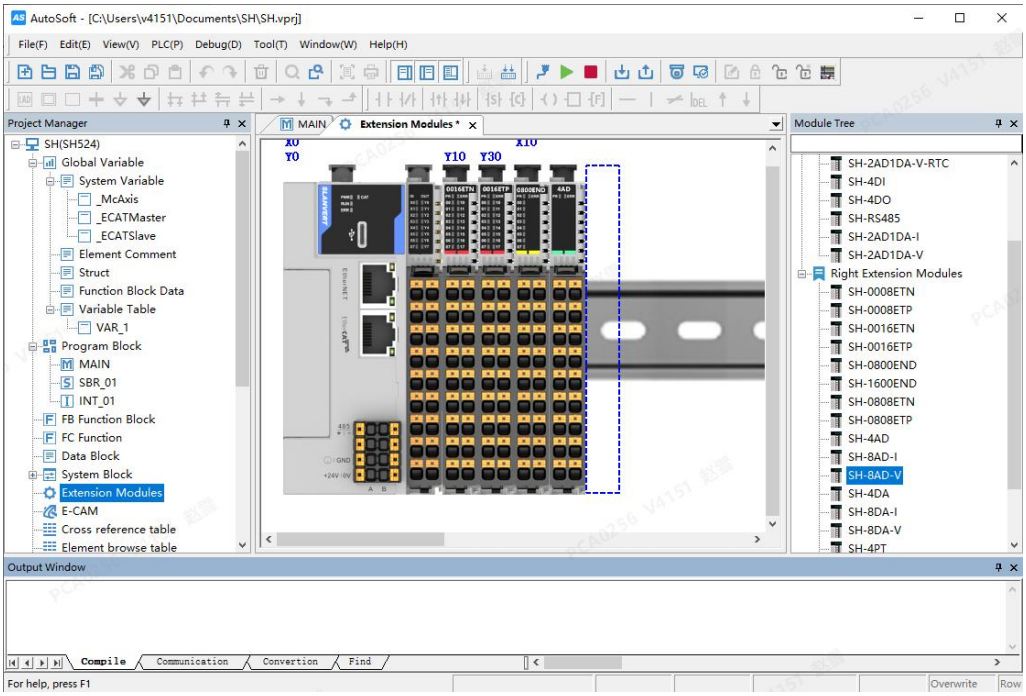
1. Unfold “System Block” in the Project Manager and configure as needed.



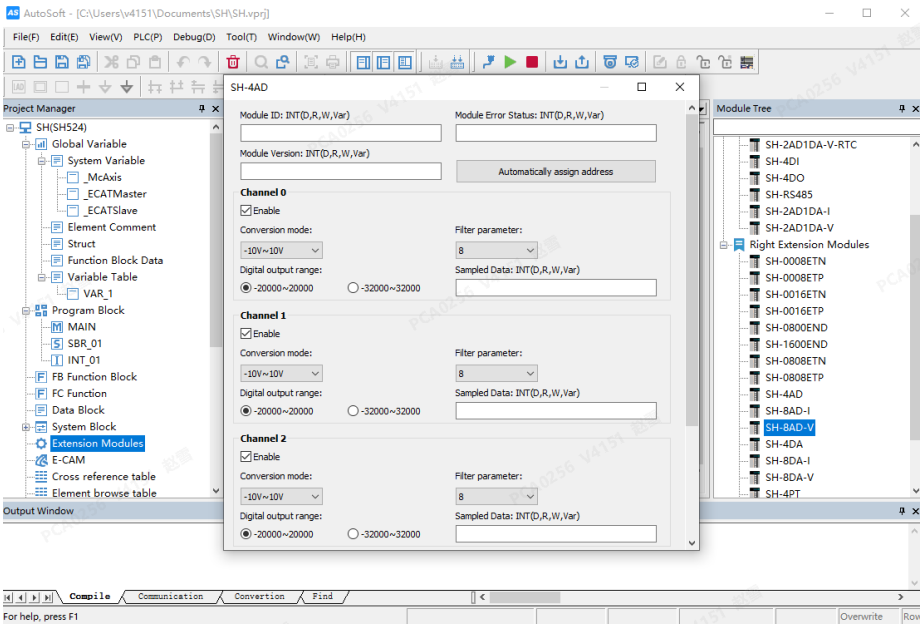
2. Double-click “Extension Modules” in the Project Manager.



3. Add modules by double-clicking it in the right “Module Tree” area according to the actual installation order.



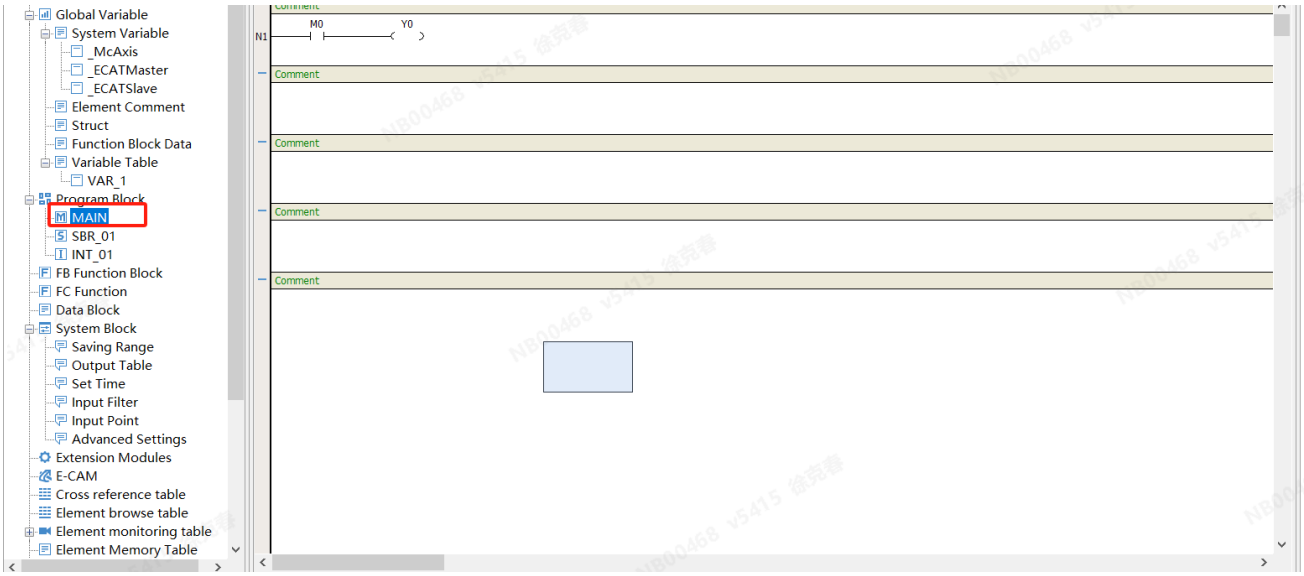
4. Double-click the added module to configure it.



Exit the interface after configuration.

2.4.5 Programming and Compilation

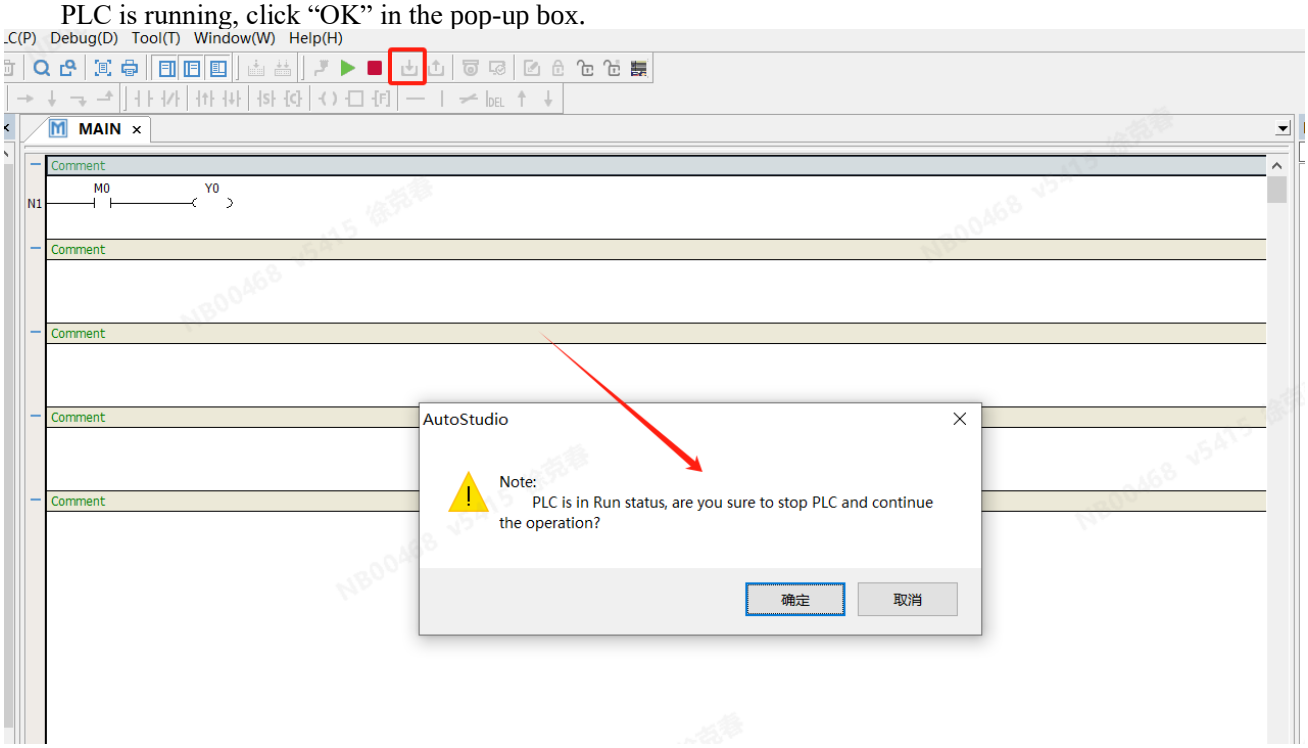
1. Unfold the “Program Block” in the Project Manager, double-click “MAIN”, and write the program.



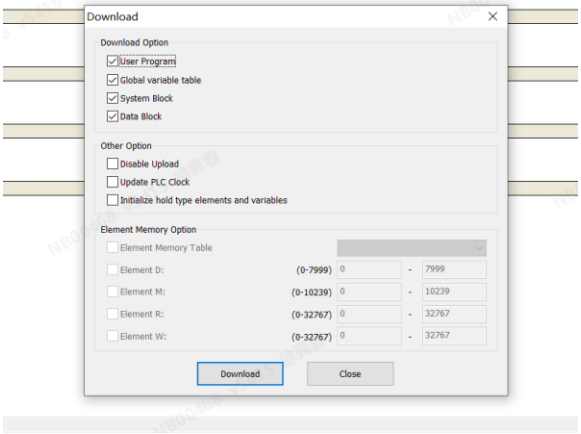
Compilation information will be displayed at the bottom of the main interface after compilation.

2.4.6 Program Download and Debugging Monitoring

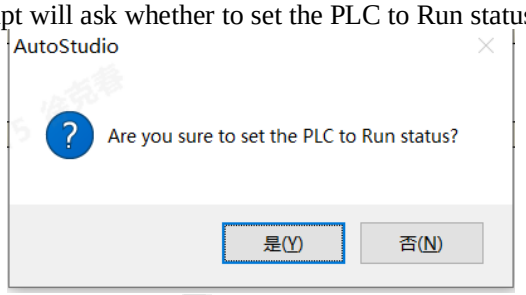
1. Navigate to “PLC>Download” in the menu bar or click in the toolbar to download the program. If the PLC is running, click “OK” in the pop-up box.



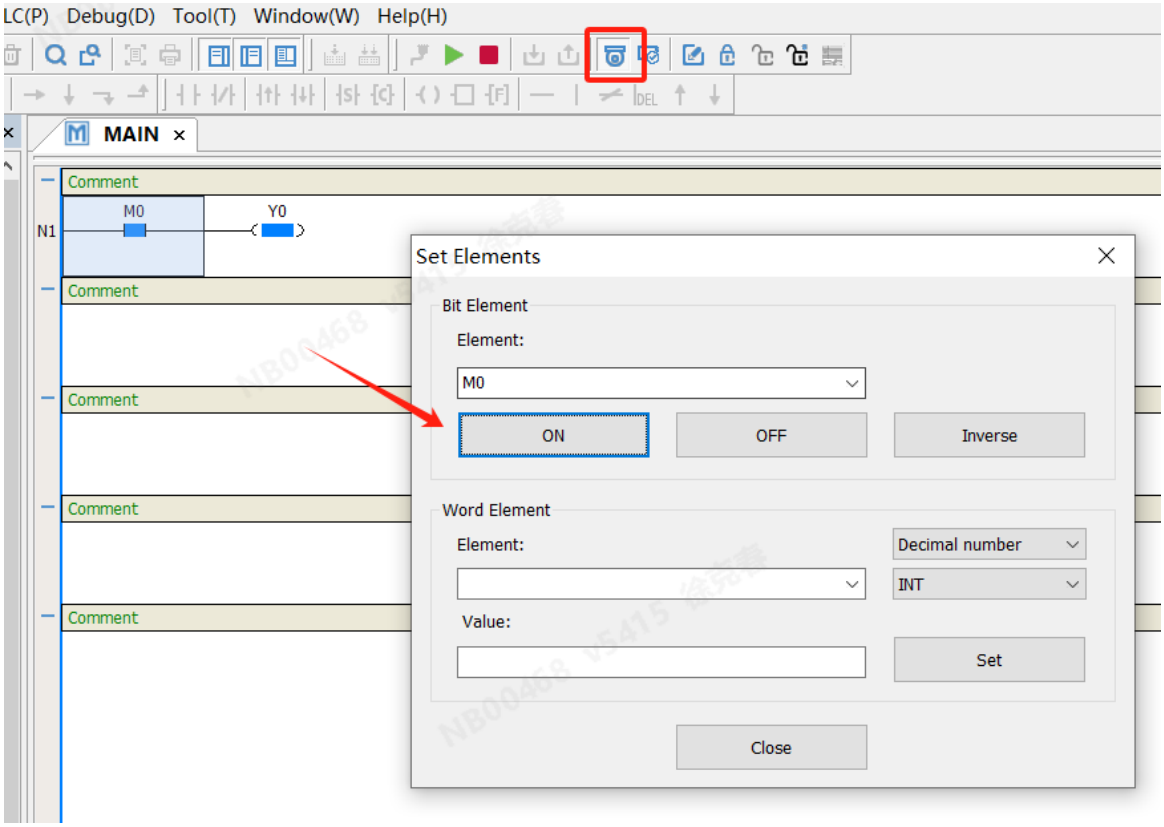
2. Then the Download box will pop up.



3. After downloading, a prompt will ask whether to set the PLC to Run status; click “Yes”.



4. Click “Debug>Monitor” in the menu bar or in the toolbar to monitoring program debugging.


LC(P) Debug(D) Tool(T) Window(W) Help(H)
MAIN
Comment
N1
M0
Y0
Comment
Comment
Comment
Comment
Set Elements
Bit Element
Element:
M0
ON
OFF
Inverse
Word Element
Element:
Decimal number
INT
Value:
Set
Close

3 Software Model

3.1 PLC Operation Mechanism

The SH Series PLC operates on a scan cycle model, sequentially executing four cyclic tasks:

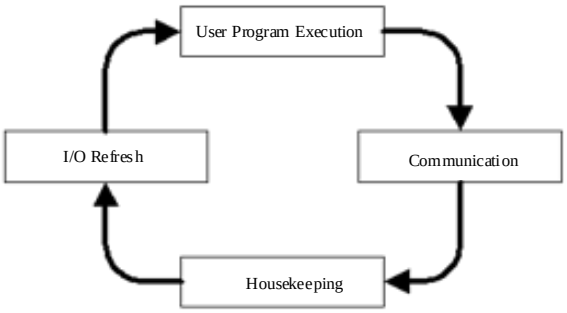


Figure 2-1 PLC Operation Mechanism

1) User Program Execution

Executes the user program's command sequence sequentially, starting from the first main program command until the end command is executed.

2) Communication

Process programming commands (download/run/stop) from software.

3) Housekeeping

Handle system housekeeping, such as refreshing panel indicators, updating software timer values, and refreshing special auxiliary relays and data registers.

4) I/O Refresh

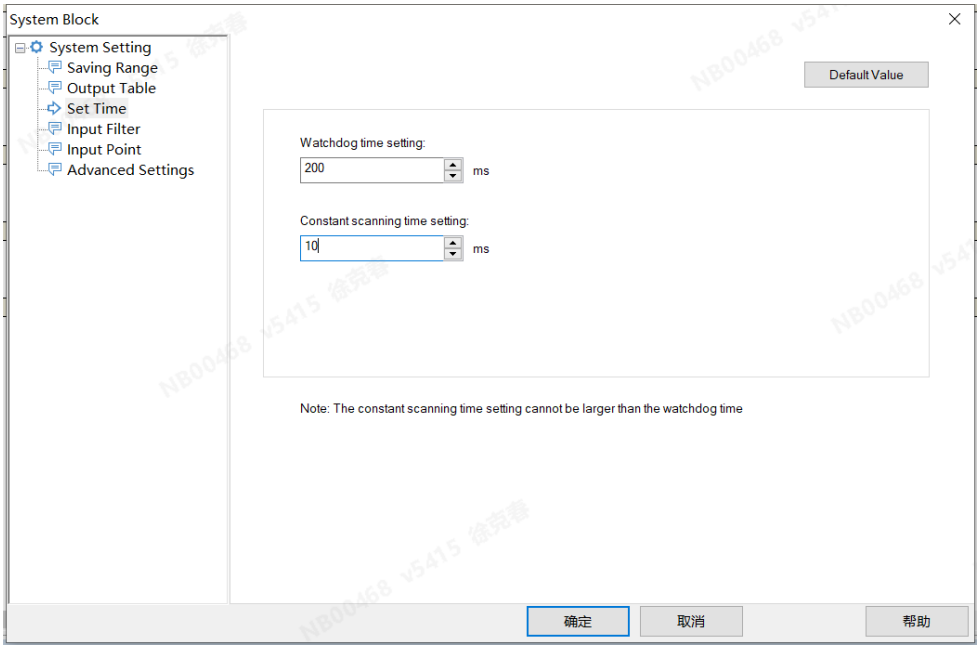
Include output and input stages.

Output stage: Turn on or off hardware outputs based on Y-element states (ON/OFF).

Input stage: Update X-element states (ON/OFF) from hardware input on/off status

3.2 Constant Scanning Time Setting

The constant scanning mode ensures fixed-duration scan cycles during PLC operation. Activate this mode and set the constant scanning time in the "Set Time" page under "System Block" in AutoSoft. The default value is 0 (disabled); users can set it to 10ms, as shown below.

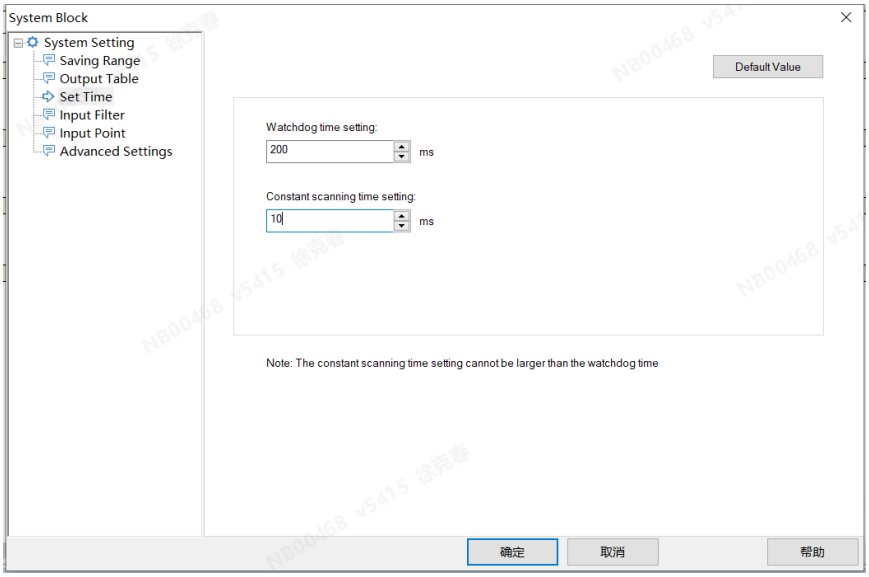


Note:

- The constant scanning time setting cannot be larger than the watchdog time.
- If the actual scan time exceeds the set value, the program will run according to the actual scan time.

3.3 User Program Watchdog

The watchdog timer monitors program execution time per scan cycle. If it exceeds the set value, the system stops and then restarts the user program. Set the watchdog time in the “Set Time” page under “System Block” in AutoSoft.

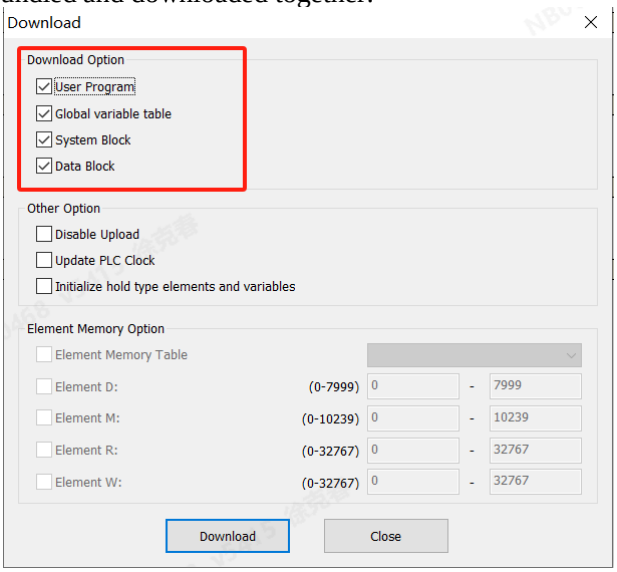


3.4 User File Download/Storage

Users can program and control the main module by downloading specific user files.

(1) These files include four types: Program Block files, Data Block files, System Block files, and user auxiliary information files (containing global variable tables and user data source files).

(2) When selecting Program/Data/System Block files for download, related user auxiliary information files will be automatically bundled and downloaded together.



3.5 Element Value Initialization

During the STOP→RUN transition, PLC initializes soft elements using the following data sources in priority order: Table 3-1 Initialization Priority Order for PLC RUN Mode

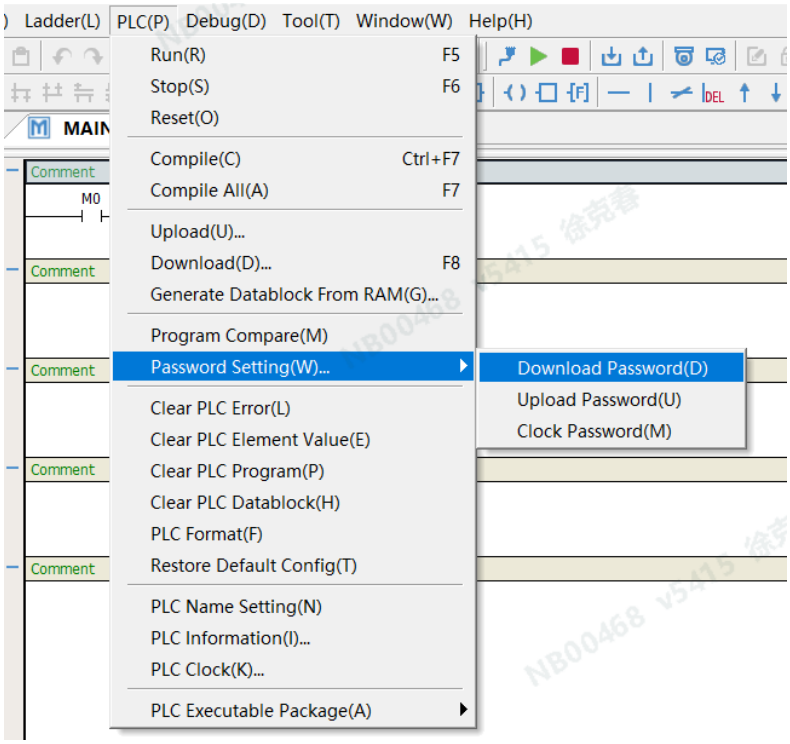
Memory Type	Power OFF→ON	STOP→RUN
Power-down save value	Highest	Highest

Data blocks*	Medium	Medium
Element value retained*	—	Low

*Requires corresponding options enabled in System Block > Advanced Settings.

3.6 User Program Security

PLC implements multi-level security strategies for program protection:



Description of user program security settings:

Function	Description
PLC Format Prohibited	Enabling and downloading "PLC Format Prohibited" in the System Block prevents deletion of user programs, system blocks, or data blocks via formatting. To disable this function, re-download a new system block without this restriction.
Download Password	Restricts access to the download function.
Upload Prohibited	Enabling "Upload Prohibited" during download blocks program upload even with a valid password. To disable this function, re-download user data and select "Allow Upload" in the download box.
Upload Password	Restricts access to the upload function.
Clock Password	Restricts access to the clock setting.
Program Password	Encrypts main/sub/interrupt programs. Encrypted programs cannot be viewed/edited without the correct password. Encryption: Right-click the program → Encrypt/Decrypt → Set password. Decryption: Right-click the program → Encrypt/Decrypt → Enter password.

Note:

- 5 consecutive failed password attempts will trigger a 5-minute lockout on SH-series PLCs.

3.7 System Configuration

Power Failure Data Retention Settings

1) Trigger Condition

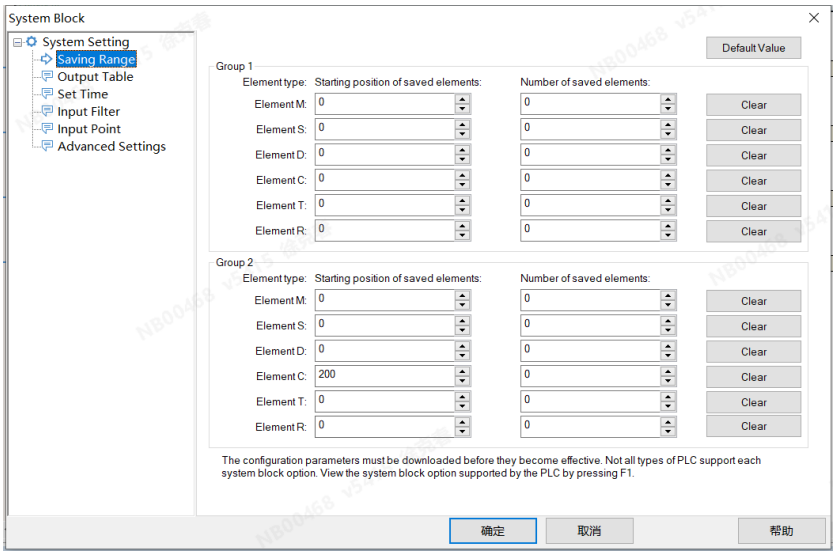
Upon detecting a power failure, the system stops the user program and saves the element data within the specified range in the System Block to the power failure backup file.

2) Data Recovery on Power-Up

If the backup file is valid after power-on, the specified elements' values are restored to their last saved state. Elements outside the save range are cleared. If the backup file is missing or corrupted, all elements are cleared.

3) Range Configuration

Configure element saving range in System Block > Saving Range, as shown below.

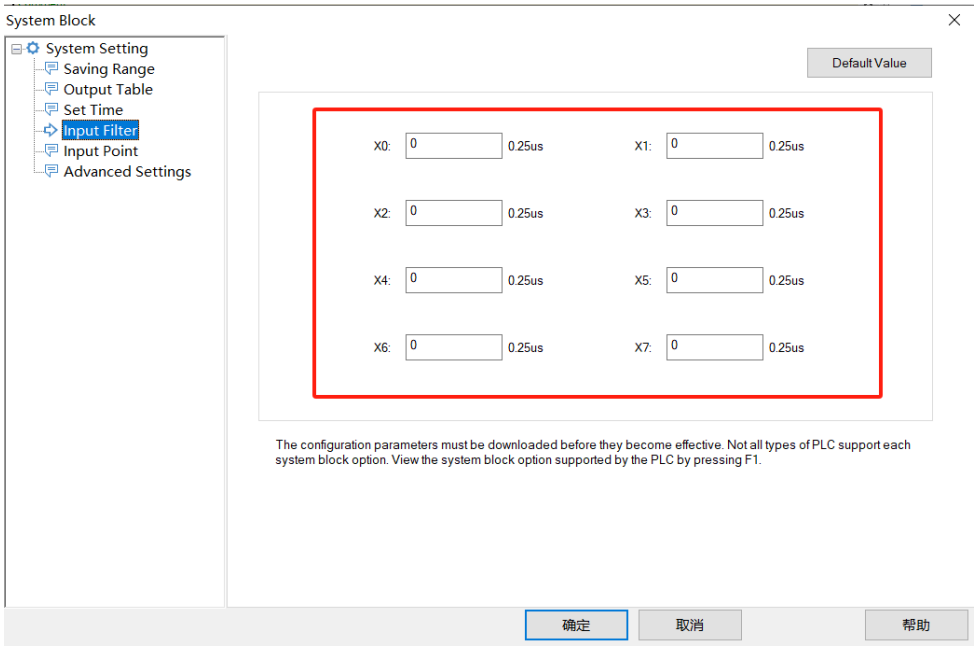


Note:

- For the SH100/300/500 series PLCs, element data is saved in non-volatile memory.

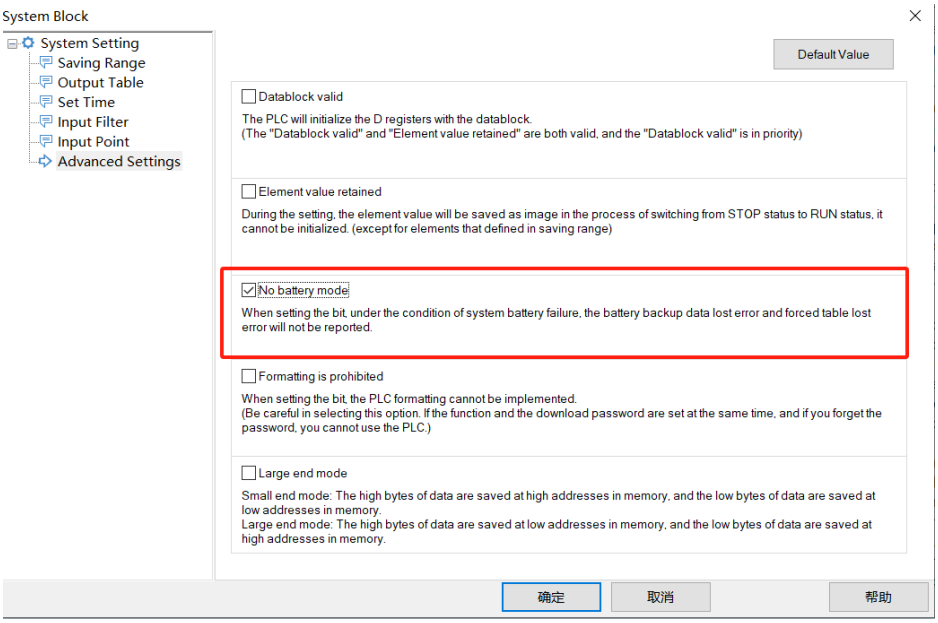
3.7.1 Input Filter

The main module's input points (X0~X7) feature digital filtering to remove interference signals. Adjust the input filter constant via the System Block> Input Filter, as shown below.



3.7.2 No Battery Mode

The SH series main module support operation without a battery or under low battery voltage. In this mode, battery-related errors will not be reported.

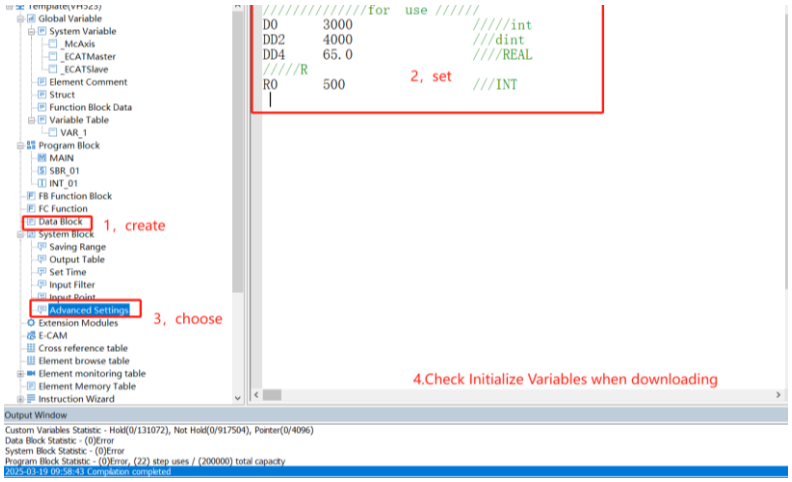


3.7.3 Data Block Configuration

A data block sets default values for D/R elements. After compilation and download to PLC, PLC initializes D/R elements using these values during startup.

Configuration Steps:

1. Click Data Block in the Project Manager to create a data block.
2. Assign initial values to D/R registers (data memory) in the data block editor. Users can assign values to words or double words, but not bytes. To add comments, put double-slash before a string.
3. Click System Block>Advanced Settings, and check Datablock valid.
4. Compile and check Initialize Variables in the download box.



Note:

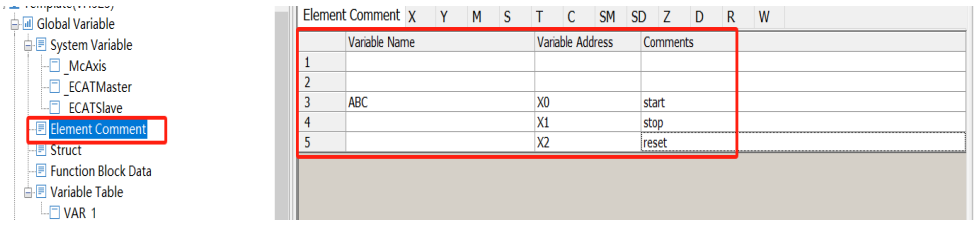
- Follow steps 1-4 to activate the data block.
- Values are assigned only during download.
- No register address overlap in data blocks; or compilation errors will be reported.

3.7.4 Element Comment

Element comments assign meaningful symbolic names to PLC addresses, accessible project-wide as global variables. These symbols are defined in a global variable table containing three attributes: Variable Name, Variable Address, and Comments.

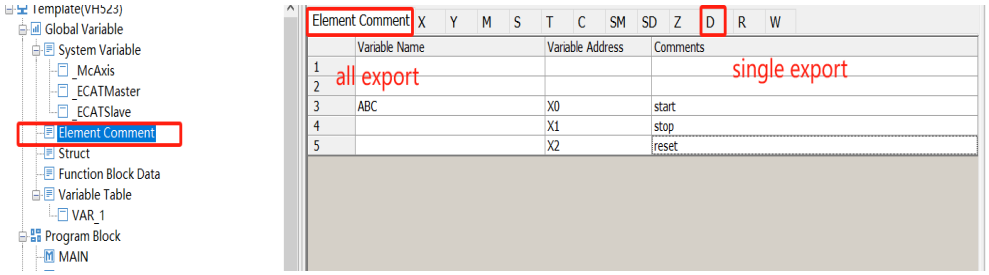
1) Comment Definition Rules:
   Combine letters (A~Z, a~z), numbers (0~9), underscores, or Chinese characters;
   Cannot start with a number or be numeric-only; Case-insensitive, ≤8 bytes;

Avoid element type letters+numbers; avoid reserved keywords: e.g., data types, command names, or operators;
No spaces allowed in variable names.

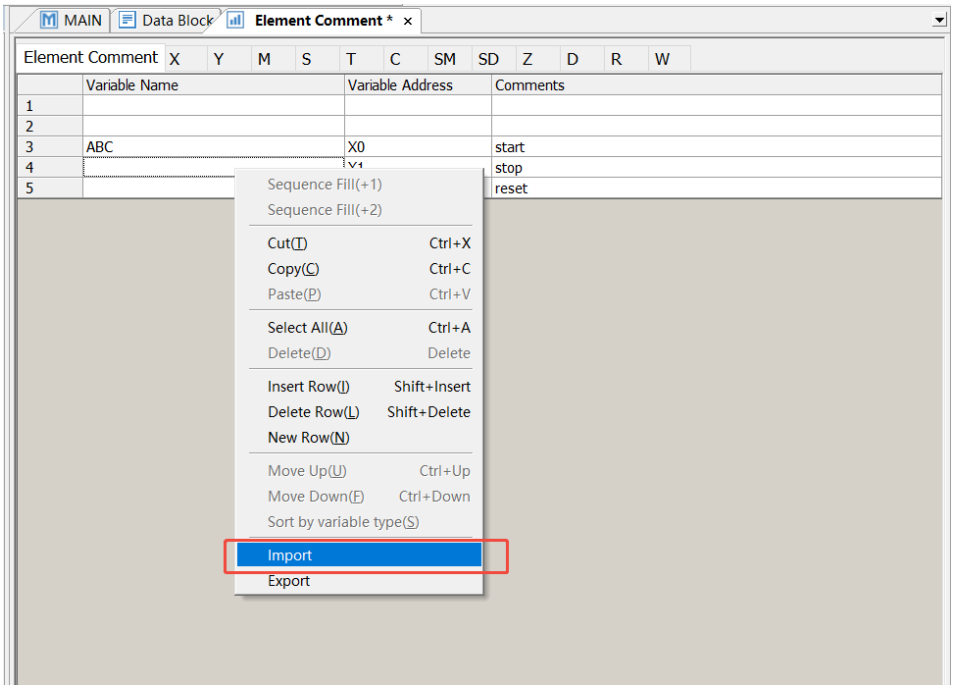


2) Element Comment Import/Export

Users can export all element comments or selected element-type comments, as shown below.



3) For batch element commenting: Right-click and select "Export" to export a .csv file, edit it, save, and then close it. Then right-click and select "Import" to import the edited file.



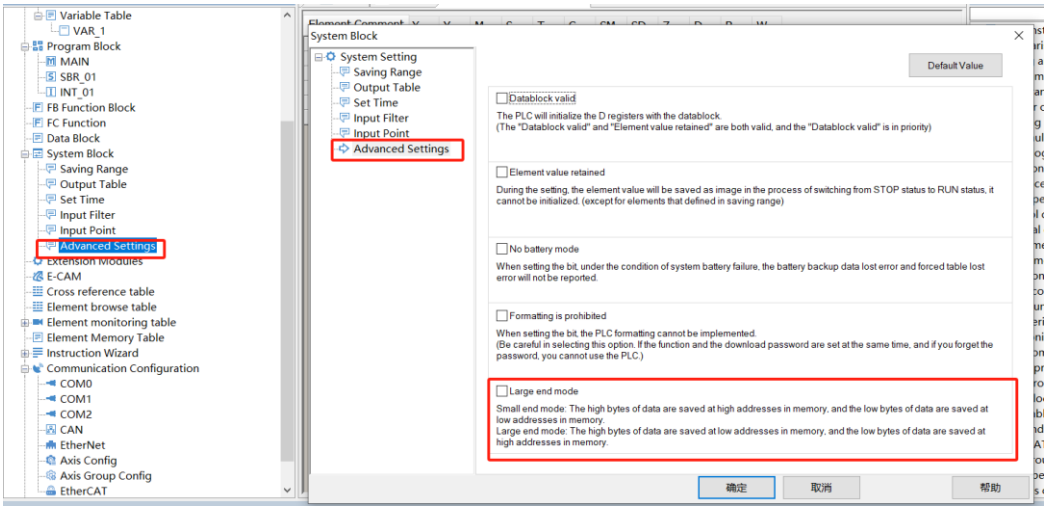
Note:

- SH300/SH500 series support a maximum of 20K global variables.

3.7.5 Large/Small End Mode

Data storage supports large end mode and small end mode. SH300/500 series allow mode selection via system configuration, with small end mode defaulted. Users can set it by "Project Manager > System Block > Advanced

Settings" and check "Large end mode", as shown below.



- Large end mode: The high bytes of data are saved at low addresses in memory, and the low bytes of data are saved at high addresses in memory.
- Small end mode: The high bytes of data are saved at high addresses in memory, and the low bytes of data are saved at low addresses in memory.

D0D1=16#12345678		
Memory address	D0	D1
Large end mode	D0=16#1234	D1=16#5678
Small end mode	D0=16#5678	D1=16#1234

Note:

➢ For SH300/500 series, 16-bit data byte retains small end mode by default. Enabling large end mode affects 32-bit data byte order only, not 16-bit data byte.

3.8 Operation and Status Control

The PLC can enter or exit the RUN state through three methods:

1. DIP switch;
2. Designated terminals (X00~X17), by setting System Block>Input Point>Input point On mode>Input point;
3. Programming software, if the DIP switch is set at ON position.

3.8.1 Run/Stop States

1) RUN

When the main module is in the running state, the whole scan cycle is executed: Execute user program → Communication → Housekeeping → I/O refresh.

2) STOP

When the main module is in the stop state, the system does not execute user program, but Communication → Housekeeping → I/O refresh are still executed.

3.8.2 State Transition

1) STOP→RUN

1. Reset

Set the DIP switch to ON position. After reset (including power-on reset), the system runs automatically.

Note:

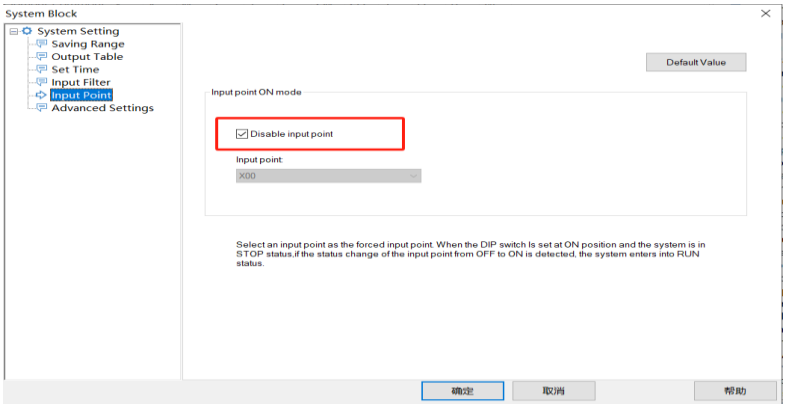
➢ If input point control is enabled in the system block, the designated input point must be ON; otherwise, the transition fails.

2. Manual operation

Toggle the DIP switch from OFF to ON during STOP.

3. Input point ON mode

When this function is valid and the system is in STOP status, if the status change of the input point (X0~X17) from OFF to ON is detected, the system enters into RUN status.



Note:

- The DIP switch must be set at ON position for this method to work.

2) RUN→STOP

1. Reset

Set the DIP switch to OFF position. After reset (including power-on reset), the system stops automatically.

Note:

- If input point control is enabled and the designated input point is OFF, the system enters STOP after reset even if the DIP switch at ON position.

2. Manual operation

Toggle the DIP switch from ON to OFF during RUN.

3. Command control

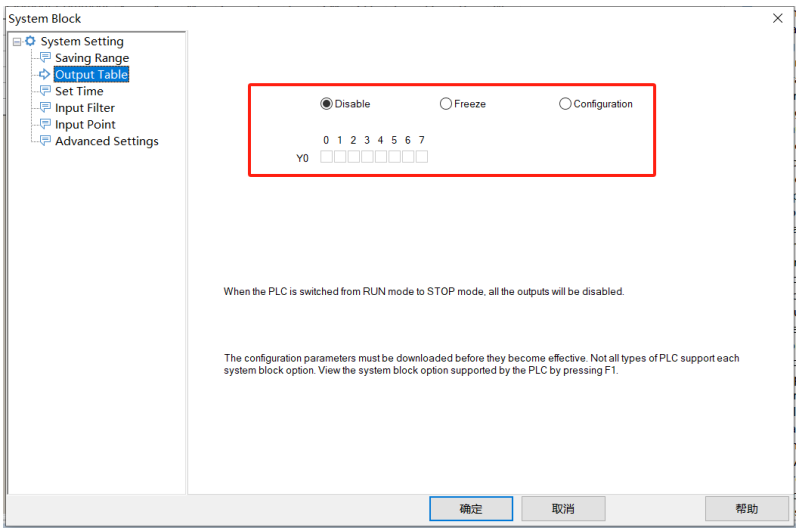
Execute the STOP command in the user program.

4. Error-induced stop

The system stops automatically upon critical errors (e.g., user program errors, user program timeout) detected.

3.8.3 Output Point Status in Stop State

The user can set the output status of output points (Y0–Y17) during STOP, with three modes selectable:



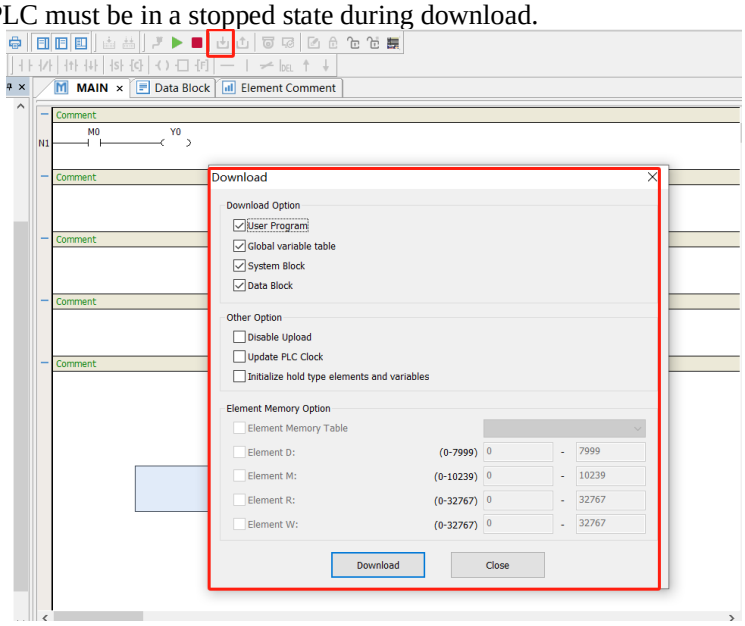
1. Disable - All output points are OFF in the STOP state.
2. Freeze - All output points remain their pre-STOP values in the STOP state.
3. Configuration - Users can set the output values of output points as needed in the System Block>Output Table.

3.9 System Debugging

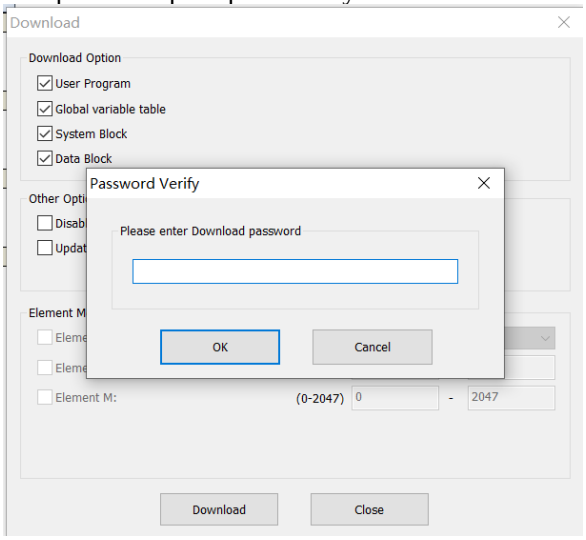
3.9.1 Program Upload/Download

1) Download

Download the system block, data block, and user program generated by AutoSoft software to the PLC via serial/USB/Ethernet. The PLC must be in a stopped state during download.

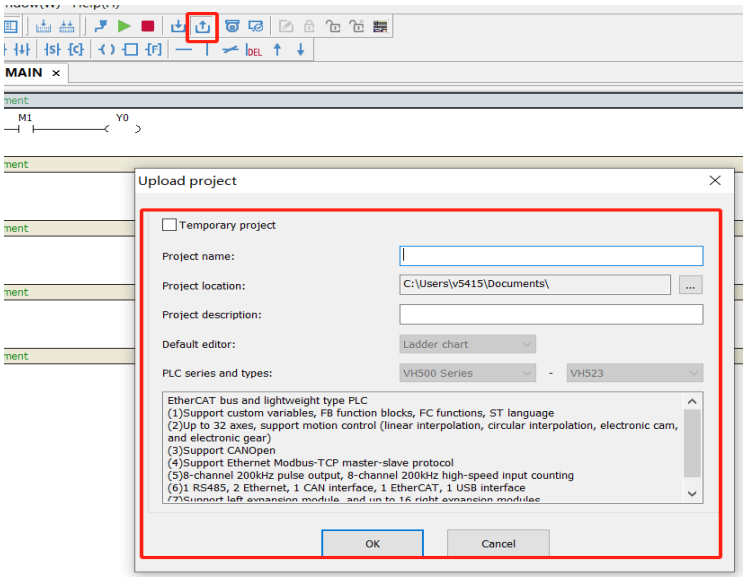


If a download password is set and not entered after software launch, a password window will pop up. Correct password initiates download; incorrect password prompts re-entry. Click the Cancel to exit the download.



2) Upload

Upload the PLC's system block, data block, and user program to the computer via serial/USB/Ethernet, and create a new project to storage it. When battery-backed data is valid, relevant user auxiliary info files are also uploaded, as shown below.



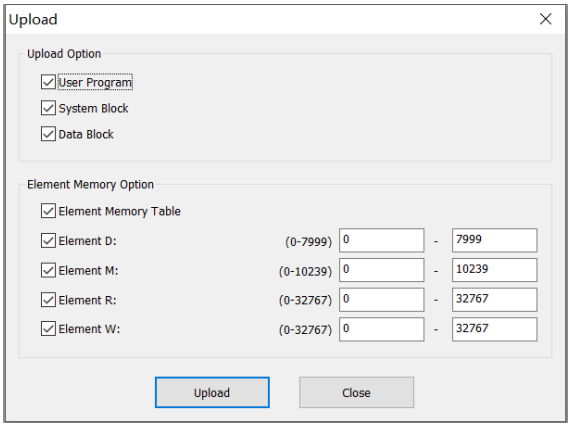
When uploading, if no password is set, the program can be directly uploaded. If a password is set and not entered

after software launch, a password window will pop up. Correct password starts upload; incorrect password returns to the upload dialog box.

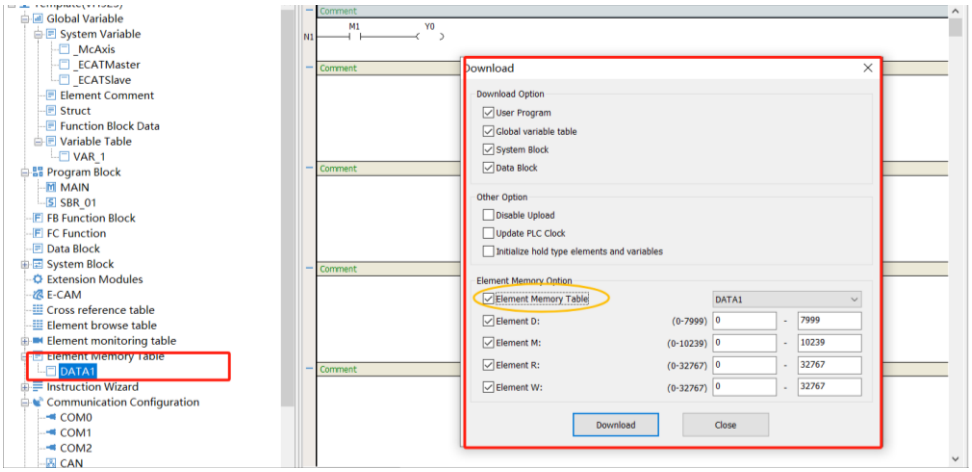
If "Upload Prohibited" is enabled during program download, the PLC cannot upload programs afterwards unless the correct password is entered to disable this restriction.

3.9.2 Element Memory Table

Upload element memory table: Upload the current D/R/W/M element values from the PLC. Upon successful upload, a DATA1 table will be automatically generated under "Project Manager > Element Memory Table", recording the uploaded element data values, as shown below.

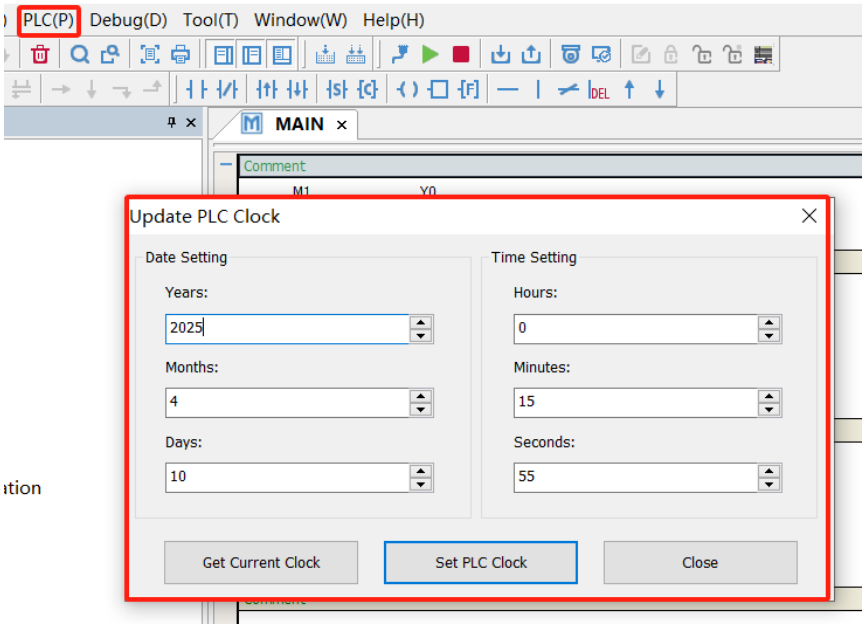


Download element memory table: When downloading the program, check the "Element Memory Table" and select required D/R/W/M element memory tables. The selected tables will be automatically written to the PLC upon successful download, as shown below.



3.9.3 Online Clock Settings

Access PLC > PLC Clock in the menu bar to synchronize the PLC's SD time register with the local computer time. Refer to the illustration below for details.



3.9.4 Online Modification

This function allows online modification to the PLC program during operation.

Warning:

- Authorize only qualified personnel to perform online modifications, adhering to safety protocols to prevent injury and equipment damage.

1) Steps

Ensure the software maintains active communication with the PLC hardware, and the PLC is in RUN mode. Click Debug->Online Modification to switch to the online modification state.

Then users can modify the main program, subroutine, and interrupt subroutine contents directly. After modifications, click PLC->Download. The software will compile all programs in the current project and automatically download them to the PLC hardware. The PLC will then execute the revised program.

2) Modification Restrictions

Unsupported Operations

①Program file management:

- Adding/deleting files
- Renaming files
- Attribute modification

②Subroutine encryption/decryption

③Configuration changes

Global Variables:

- Supported: Variable additions/deletions; modification of variable names or comments
- Unsupported: Modification of attributes other than variable names or comments

FB/FC Local Variables:

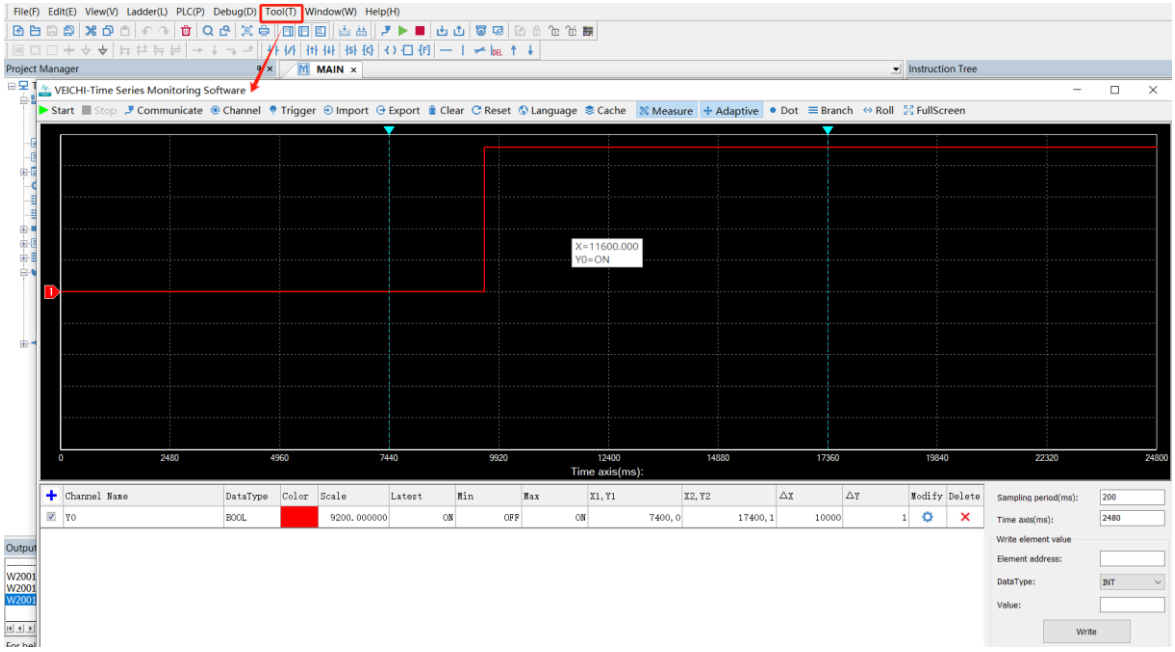
- Unsupported: IN/INOUT/OUT-type variables additions/modifications/deletions
- Supported: VAR-type variable additions/deletions

VAR Variables:

- Added before current cycle:
  - Name/comment modifications supported only
- Added during current cycle:
  - Full attribute modifications enabled

3.9.5 Sequence Monitor

This oscilloscope-like tool captures variable value changes in real time for PLC debugging and operational analysis. Access via Tool>Sequence Monitor in the menu bar, as shown below.



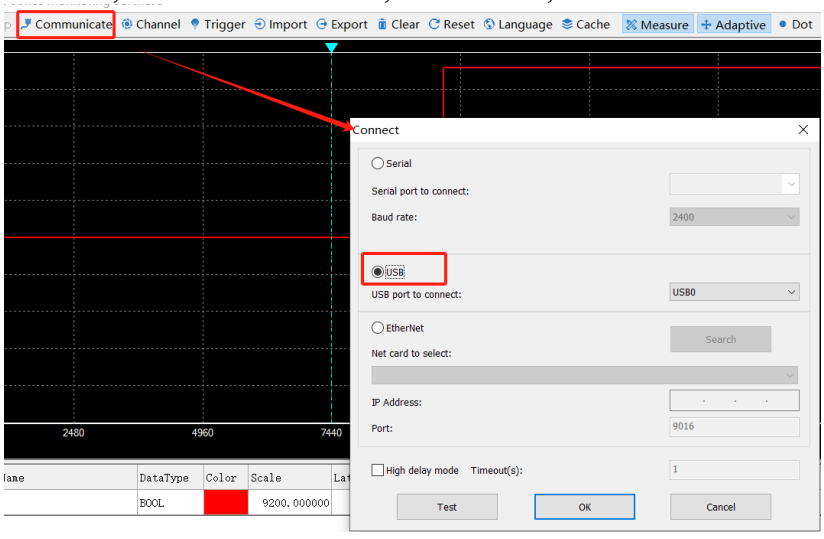
(I) Features

(1) USB/Ethernet connection support
(2) Exclusive communication protocol (incompatible with AutoSoft) (! CAUTION)
(3) Up to 6-channel concurrent monitoring, with a minimum sampling cycle of 1ms
(4) Waveform import/export capability
(5) Independent/overlapping channel display (via “Branch”)
(6) Axis scaling: Ctrl+roll (vertical), roll (horizontal)
(7) Sampling trigger functionality

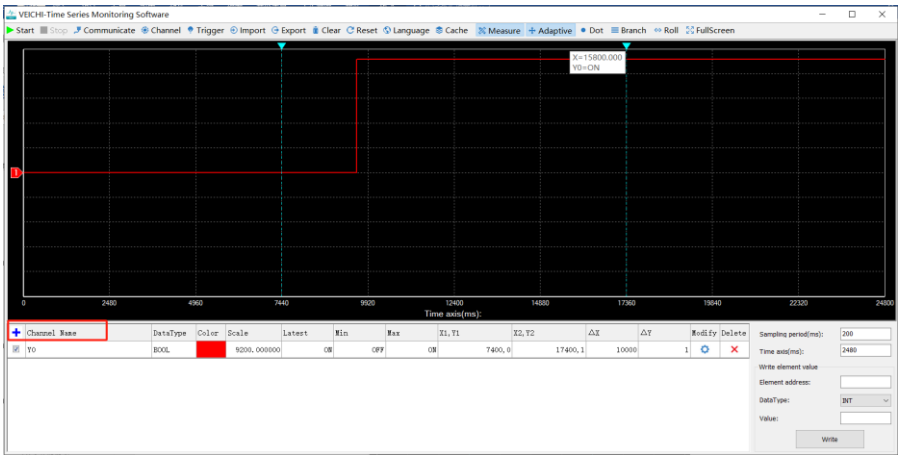
(II) Setup Workflow

Click the Tool > Sequence Monitor in the menu bar to open the sequence monitor chart.

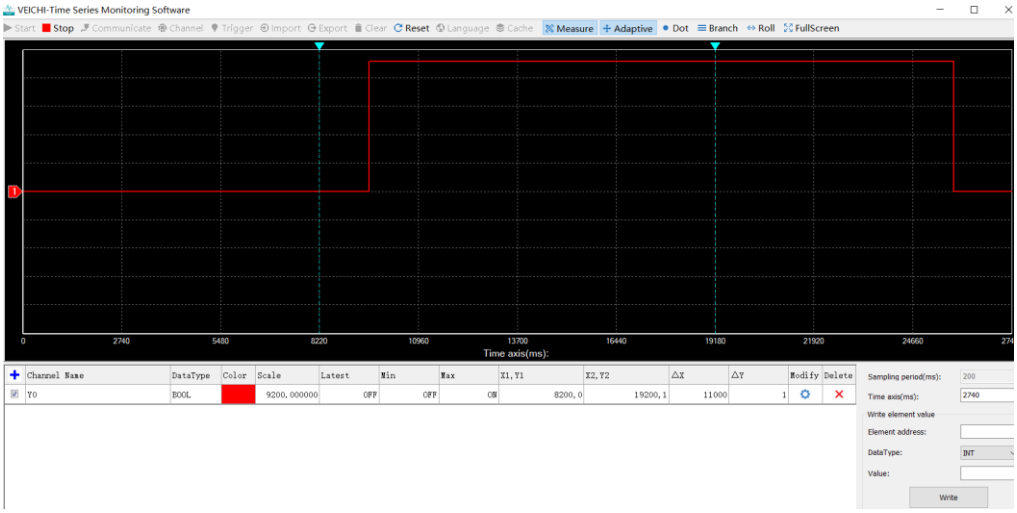
1. Click the Communicate, select USB or Ethernet, and click OK, as shown below.



2. After a successful connection, manually input target variables and select the data type, as shown below.

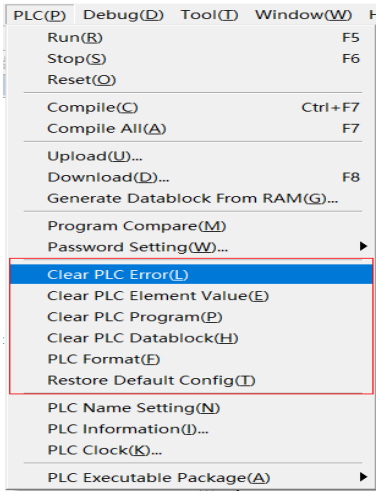


3. Then run it.



3.9.6 Program Clear/Format

This function includes: Clear PLC Error, Clear PLC Element Value, Clear PLC Program, Clear PLC Datablock, and PLC Format, as shown below.



Description of program clear/format

Function	Description
Clear PLC Element Value	Clears all element values in the PLC; requires PLC in STOP mode. This may cause operation errors or data loss; please use with caution. To prevent accidental operation, the software will pop up a confirmation window before execution.
Clear PLC Program	Clears user programs in the PLC; requires PLC in STOP mode. After this, the PLC won't execute any user program; please use with caution. To prevent accidental operation, the software will pop up a confirmation window before execution.
Clear PLC Datablock	Clears all data block settings in the PLC; requires PLC in STOP mode. After this, the PLC won't use preset data block values to initialize D elements; please use with caution. To prevent accidental operation, the software will pop up a confirmation window before execution.

PLC Format	Formats all PLC data, including clearing user program, restoring default settings, and clearing data blocks; requires PLC in STOP mode. All downloaded and set data will be lost after this; please use with caution. To prevent accidental operation, the software will pop up a confirmation window before execution.
Clear PLC Error	Clears current PLC error information.

3.9.7 Fault Diagnostics

The system detects and reports the system error and user program execution error.

1. System error is caused by abnormal system operation,
2. and user program execution error is triggered by abnormal program execution.

All errors are uniquely numbered, with each error code representing a specific fault. For details, refer to 16 Error Codes.

1) System Error Report

When a system error is detected, the error code is recorded in special data register SD3 and special relay SM3 is triggered. Read the error code from SD3 to identify the error.

In case of concurrent errors, the highest priority error is stored in SD3.

Critical system errors will halt user program execution and cause the ERR indicator on the main module ON.

2) Program Execution Error Report

When a user program execution error occurs, special relay SM20 is triggered, and the error code is stored in special data register SD20.

SM20 resets after successful execution of the next command, while SD20 retains the last error code.

The system logs execution errors using a stack structure: Special data registers SD20~SD24 form a 5-level error stack storing the last five errors.

When a new error occurs and its code differs from the one in SD20, it is pushed into the stack, as illustrated below.

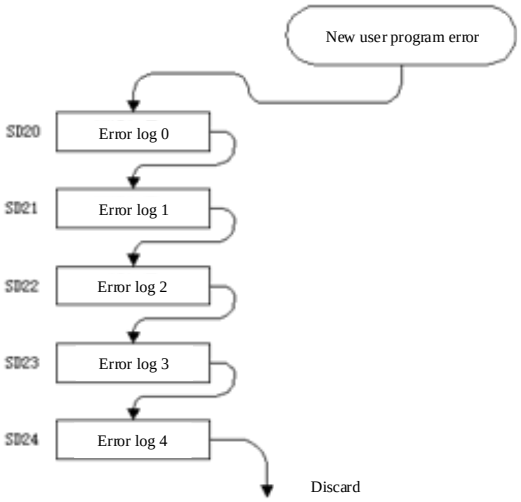
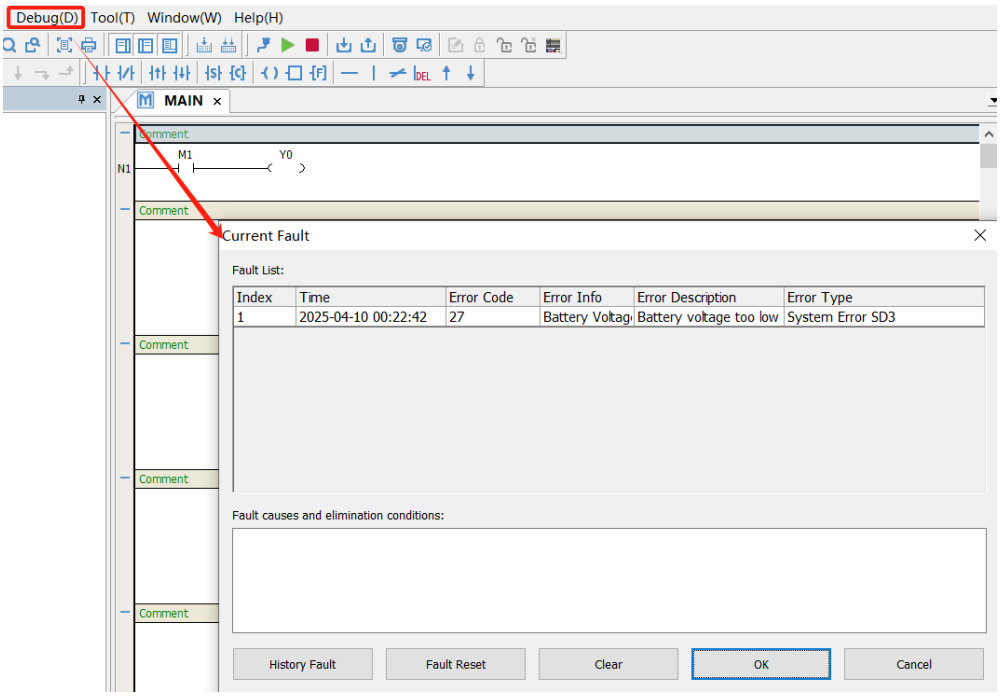


Figure 2-12 Error Code Stack Operation

Critical execution errors will halt program execution and cause the ERR indicator on the main module ON, while non-critical errors do not trigger the indicator.

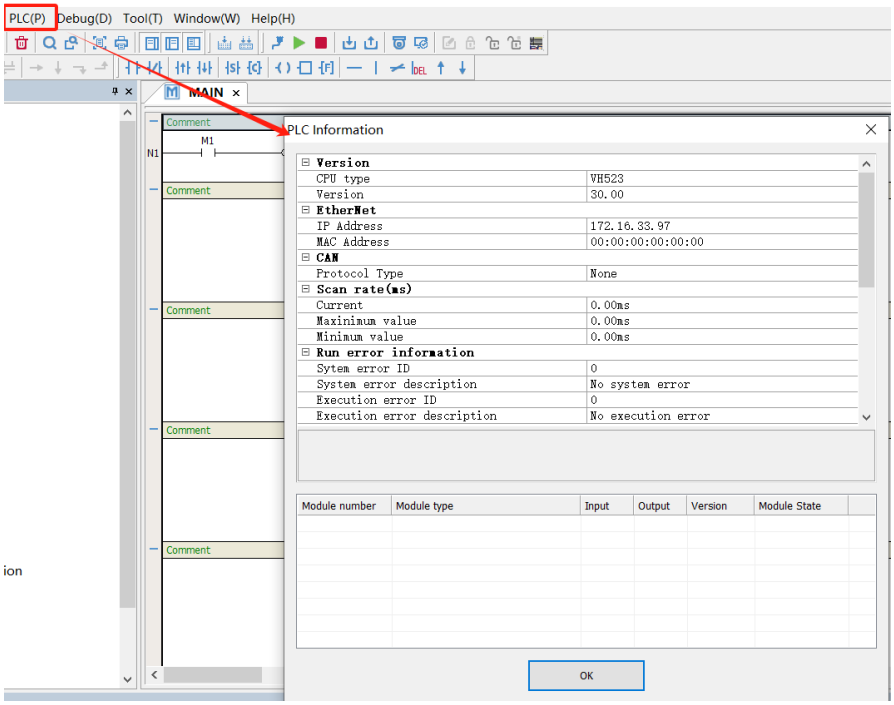
3) View of Error Information

When the PLC is online, view PLC fault diagnostics in AutoSoft via “Current Fault” or “History Fault” menus with error codes and descriptions. Click Debug > Current Fault/History Fault in the menu bar. Double-clicking any entry will navigate to the error location in the program. This is shown in the figure below:



3.9.8 PLC Information

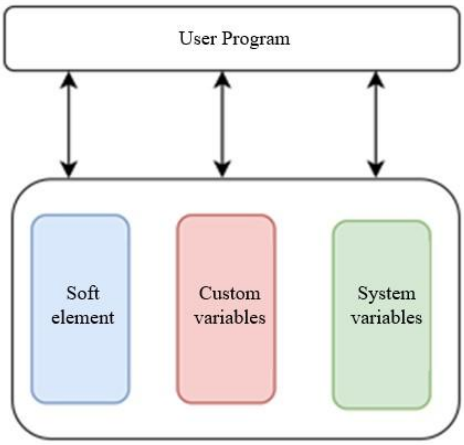
Navigate to PLC > PLC Information in the menu bar to view current PLC software specifications: PLC model, MCU firmware version, FPGA version, high-speed output channels, MAC address, CAN baud rate, current scan cycle, runtime error status, program capacity, operational status, battery voltage, local I/O points, extended I/O points, and left/right extension module types, as shown below.



4 Programming Basics

4.1 Overview

In system development, PLC supports three variable memory structures including soft elements, custom variables, and system variables. The operation rules are summarized as follows:



SH300/SH500 Memory Architecture

Memory Space	Description	Operation Rule
Soft elements (~1.2MB)	Supports X/Y/M/SM/L/S/T/C/D/R/W/SD/Z/V element types	No definition required; directly accessible in user programs
Custom variables (16MB)	Retentive: 2MB (131,072 words) Non-retentive: 14MB (917,504 words)	Requires definition in Variable Table; Supported types: BOOL, INT, DINT, RELA, Array, Pointer, Struct, String
Pointer variables (128KB)	Max. 4,096 pointers	Requires definition in Variable Table prior to use.
System variables	/	For accessing axis data structures, etc.

4.2 Soft Elements

4.2.1 Bit Soft Elements

PLC programming supports bit-soft elements. Specifications including types, ranges, points, and related descriptions are shown below:

Type	Range	Points	Data Type	Description
X	X0~X1777	1024 (octal)	BOOL	Input
Y	Y0~Y1777	1024 (octal)	BOOL	Output
M	M0~M10247	10248	BOOL	Auxiliary relay
S	S0~S4095	4096	BOOL	Status relay
LM	LM0~LM63	64	BOOL	Local auxiliary relay
SM	SM0~SM1023	1024	BOOL	Special auxiliary relay
C	C0~C263	264	BOOL	Counter

T	T0~T512	512	BOOL	Timer

Note:
- X points reflect physical input terminal states, while Y points represent physical output terminal states;
- All bit-soft elements operate as BOOL types for input/output parameters in BOOL-type instructions

4.2.2 Word Soft Elements

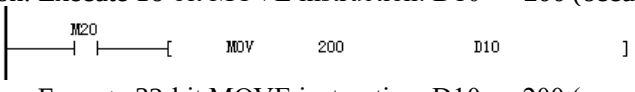
PLC programming supports word-soft elements. Specifications including types, ranges, points, and related descriptions are shown below:

Type	Range	Points	Data Type	Description
D	D0~D7999	8000	BOOL/INT/DINT/REAL	Configurable power-down save range
R	R0~R32767	32768	BOOL/INT/DINT/REAL	Configurable power-failure retention range
W	W0~W32767	32768	BOOL/INT/DINT/REAL	No power-down save
C	C0~C263	264	BOOL/INT/DINT	C0~C199: 16-bit counter; C200~C235: 32-bit bidirectional counter; C236~C263: 32-bit high-speed counter
T	T0~T511	512	BOOL/INT	T0~T209: 100ms resolution T210~T255: 10ms resolution T256~T511: 1ms resolution
V	V0~V63	64	INT	Local variable register
Z	Z0~Z15	16	INT	Variable addressing register
SD	SD0~SD1023	1024	INT/DINT/REAL	Special auxiliary register
KnXX*	Up to XX	Up to XX	INT/DINT	Example: K4M0: 16-bit address (M0~M15); K8M0: 32-bit address (M0~M31) KnXX denotes bit address
*: XX: X/Y/M/SM/L/S bit addresses Kn: Bit address grouping notation				

Element Usage Descriptions
- The range of power-down save can be configured via the System Block.
- Word soft elements can be used as integers or floating-point numbers. They lack inherent data type attribute and resolve based on parameter attributes of the instruction.
- When used as integers, word soft elements can be 16-bit (occupies single soft element) or 32-bit (occupies two soft elements). As floating-point numbers, they occupie 2 elements.

Example:

1) 16-bit integer operation: Execute 16-bit MOVE instruction: D10 ← 200 (occupies D10)

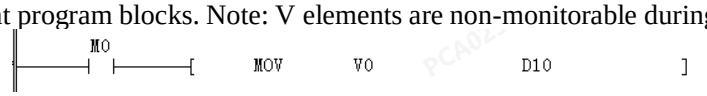


2) 32-bit integer operation: Execute 32-bit MOVE instruction: D10 ← 200 (occupies D10 [low-order] + D11 [high-order])

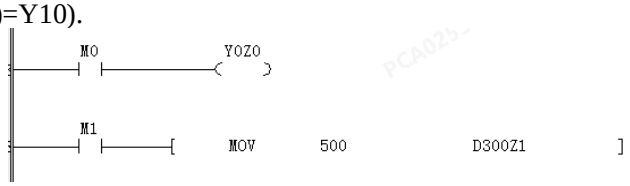
3) Floating-point number operation: Execute REAL MOVE instruction: D100 ← 100.0 (occupies D100+D101)

4) Word elements support direct bit manipulation via . notation. For example, programming D0.5 sets bit 5 of the D0 word element to ON.

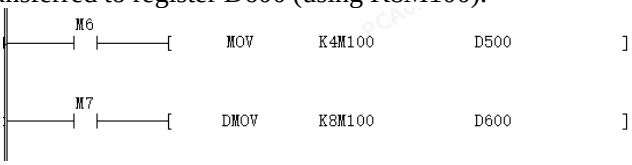
5) The V soft elements can be used in the main program and subroutines as locally scoped variables valid within independent program blocks. Note: V elements are non-monitorable during program execution.



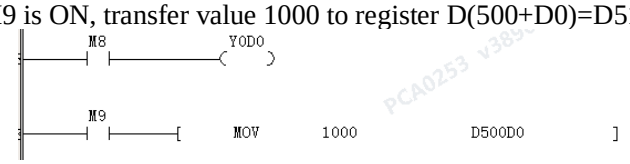
6) The Z elements enable indirect addressing of bit or word elements. Note: When using indexing, avoid address out-of-bounds (e.g., maximum address for D devices is D7999). Example: If Z1=10, when M1 turns ON, the value 500 is transferred to register D(300+Z1)=D310. (Note: X and Y are octal. If Z0=8 and M0 is ON, Y(0+Z0)=Y10).



7) The KnXX format combines bit elements into 16-bit or 32-bit words. Example: When M6 is ON, the 16-bit value of M100~M115 is transferred to register D500 (using K4M100); When M7 is ON, the 32-bit value of M100~M131 is transferred to register D600 (using K8M100).



8) D0~D255 can serve as index registers for indirect addressing, functionally identical to Z registers. Example: When D1=10 and M9 is ON, transfer value 1000 to register D(500+D0)=D510.



4.3 Data Type

All instruction operands possess defined data type attributes. The four supported data types are specified below.

(I) Operand Data Types

Data Type	Description	Width	Range
BOOL	Bit	1	1 (ON)/0 (OFF)
INT	Integer	16	-32768~32767
DINT	Double-length integer	32	-2147483648~2147483647
REAL	Floating point	32	±1.175494E-38~±3.402823E＋38

(II) Element-Data Type Matching

Instruction operands must maintain compatibility between element types and data types. The matching relationships are specified in the following table.

Element-Data Type Matching Relationships

Data Type	Soft Elements														
BOOL	X	Y	M	S	LM	SM				C	T				

Data Type	Soft Elements														
INT	Constant	KnX	KnY	KnM	KnS	KnLM	KnSM	D	SD	C	T	V	Z	R	W
DINT	Constant	KnX	KnY	KnM	KnS	KnLM	KnSM	D	SD	C		V		R	W
REAL	Constant							D				V		R	W

When an instruction's operand type conflicts with its data type requirement, the instruction becomes invalid. For example, "MOV 10 X0" is invalid as MOV is an integer-type operand while X0 is a BOOL-type element.

Note:

- INT-type operands accept: KnX, KnY, KnM, KnS, KnLM, KnSM (1≤n≤4)
- DINT-type operands accept: KnX, KnY, KnM, KnS, KnLM, KnSM (5≤n≤8)
- INT-type counters: C0~C199
- DINT-type counters: C200~C263

4.3.1 Constant

Constants may be used as instruction operands. The SH-series PLC supports multiple constant formats:

Table 4-1 Constant Formats

Constant Type	Example Format	Valid Range	Comments
16-bit signed decimal integer	8949	-32768~32767	/
32-bit signed decimal integer	2147483646	-2147483648~2147483647	/
16-bit hexadecimal	16#1FE9	16#0~16#FFFF	Sign interpretation depends on operand data type. Applies to hex/octal/binary constants.
32-bit hexadecimal	16#FD1EAFE9	16#0~16#FFFFFFFF	
16-bit octal	8#7173	8#0~8#177777	
32-bit octal	8#71732	8#0~8#37777777777	
16-bit binary	2#10111001	2#0~2#1111111111111111	
32-bit binary	2#101110011111	2#0~2#1111111111111111 1111111111111111	
Single-precision float point	-3.1415E-16 3.1415E＋30.016	±1.175494E-38~±3.402823E ＋38	IEEE 754 compliant. Precision loss may occur (e.g., 1234567.89 stored as 1234567.9 in D0).

4.3.2 Variables

AutoSoft programming supports both direct addressing (X, Y, M, D, R, W elements) and custom variables for implementing control logic or complex control sequences, improving code maintainability and readability.

Table 4-2 Supported Custom Variables

Type	Capacity	Data Type	Description
BOOL	Custom variables (16MB)	BOOL, INT, DINT, RELA, Array, Pointer, Struct, String	Retentive: 2MB (131,072 words) Non-retentive: 14MB (917,504 words)
INT			
DINT			
REAL			
STRING			
Pointer	4096	BOOL/INT/DINT/REAL	Non-retentive

4.4 Custom Variables

Users can define global variables for programming. Global variable naming must comply with the following rules:

- May contain underscores (_), letters, numbers, or Chinese characters, but cannot start with underscores or

numbers.

- Must not conflict with reserved elements (soft elements, constants, standard data types, instructions, subroutines, interrupt subroutines, or MC instructions).
- Prohibited keywords include: ARRAY, TRUE, FALSE, ON, OFF, and NULL.

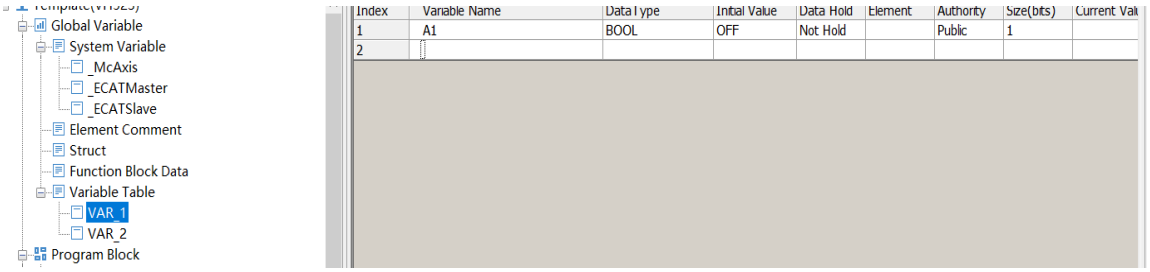
(I) Variable Data Types

Variables support structs and arrays. Supported data types are:

Data Type	Description
BOOL	Boolean
INT	Single-word integer
DINT	Double-word integer
REAL	Real number
STRING	String

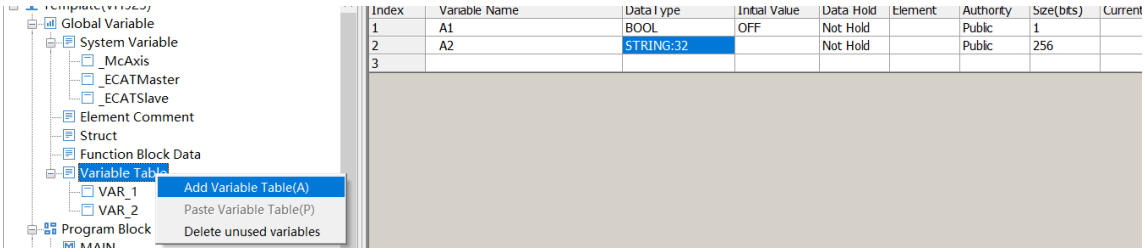
(II) Global Variables

The "Global Variable" in the Project Manager area contains: System Variable, Element Comment, Struct, Function Block Data, and Variable Table. Variable Table manages custom variables and support multiple variable creation, modification, and deletion operations, as shown below.



(III) Custom Variables

Create a new Variable Table (VAR_1), double-click text fields and enter variable names, and select data types for it.

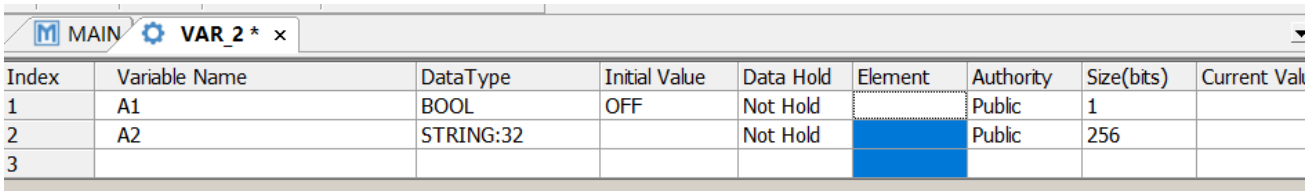


1. Double-click text fields or use dropdown menus to edit it.
2. Right-click and select "Insert Row" to add variables.
3. Right-click target row and select "Delete Row" to delete variables.
4. Export Variable Table for batch editing, and then import the Table. Close files before re-importing to avoid failures.
5. Compile project before exporting HMI variables.

Note:

- Single table: Right-click target table and select "Export HMI Variable".
- All variables: Right-click "Global Variable" and select "Export HMI Variable" to generate a unified HMI tag table.

Custom Variable Table is shown in the figure below.



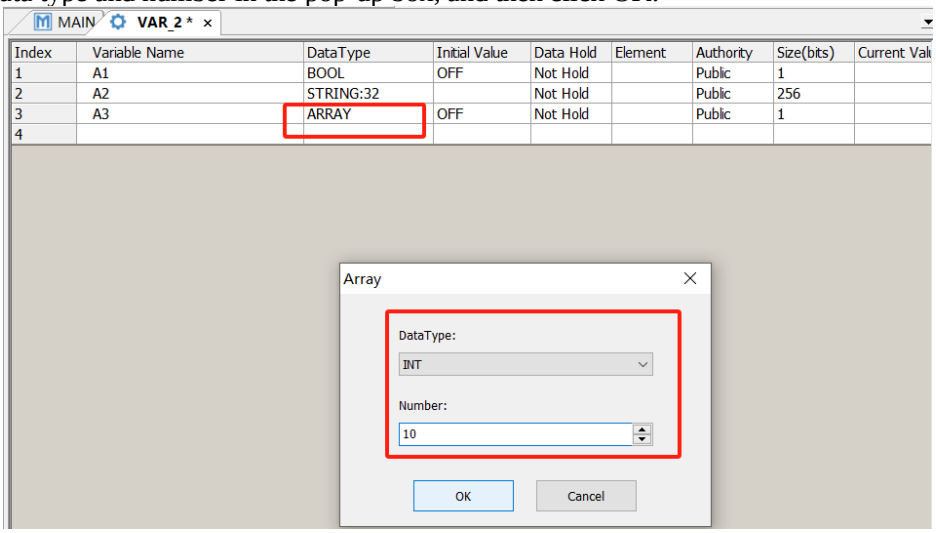
Description of Custom Variables

Item	Description
Variable Name	Name directly referenced in programming, following naming rules.
Data Type	BOOL/INT/DINT/REAL/STRING, including BOOL/INT/DINT/REAL/ DI arrays BOOL/INT/DINT/REAL/ structs For arrays, specify type and length via pop-up dialog box. For structs, use predefined struct variables.
Initial Value	Default value assignment. Supports element-specific initialization for arrays/structs.
Data Hold	Options: Hold/not hold. Initial values apply only to not hold variables. Note: Binding to element addresses inherits their retention properties.
Element Address	Bind to X/Y/M/S/D/R/W element addresses for shared address access.
Authority	Private: HMI variable export disabled. Public: Available for HMI tag communication.
Size (bits)	Displays allocated memory size based on data type.
Note	Optional descriptive notes for the variable.

4.5 Array Variable

An array groups variables of the same type into an ordered collection. In programming, select the ARRAY as data type can define it to an array.

1. Set the data type and number in the pop-up box, and then click OK.



2. Double-click the "…" under "Initial Value" to modify initial values and note for individual variables, as shown below.

MAIN VAR_2 * ×

Index	Variable Name	DataType	Initial Value	Data Hold	Element	Authority	Size(bits)	Current Val
1	A1	BOOL	OFF	Not Hold		Public	1	
2	A2	STRING:32		Not Hold		Public	256	
3	A3	INT[10]	...	Not Hold		Public	160	
4	A3[0]	INT	0					
5	A3[1]	INT	0					
6	A3[2]	INT	0					
7	A3[3]	INT	0					
8	A3[4]	INT	0					
9	A3[5]	INT	0					
10	A3[6]	INT	0					
11	A3[7]	INT	0					
12	A3[8]	INT	0					
13	A3[9]	INT	0					
14								

When an instruction accesses an array without specifying an index, it starts at the first element. If an explicit index is provided, the instruction accesses the element at the specified index. (Note: Array indices begin at 0.)

For example:

1) When M10 is ON: Assign 10 consecutive elements from Array_0[0] to Array_0[9] to addresses D300~D309.

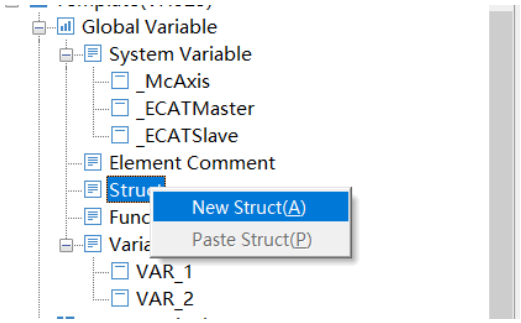
2) When M11 is ON: Assign 5 consecutive elements from Array_0[5] to Array_0[9] to addresses D305~D309.

4.6 Struct Variable

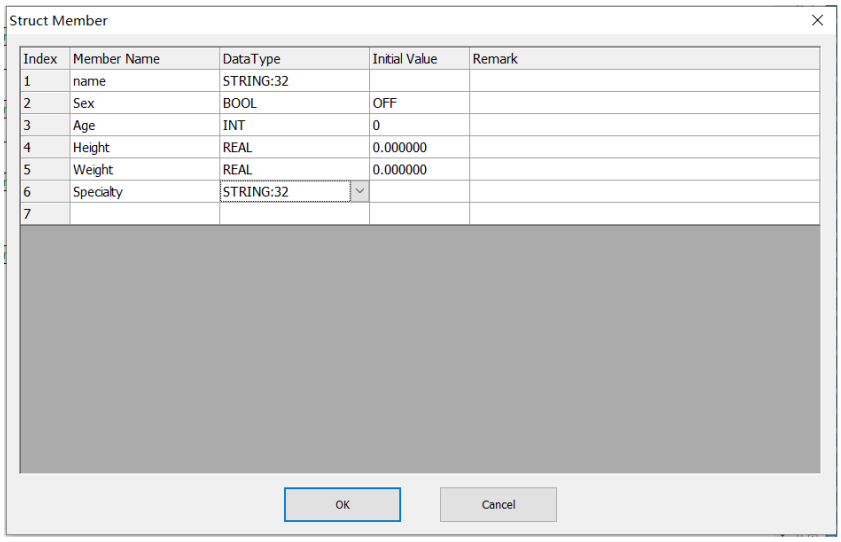
A struct is a composite data type that groups multiple data types into a single variable. Each data unit within a struct is termed a member, which can be basic data types, arrays, pointers, or nested structs.

Struct Creation

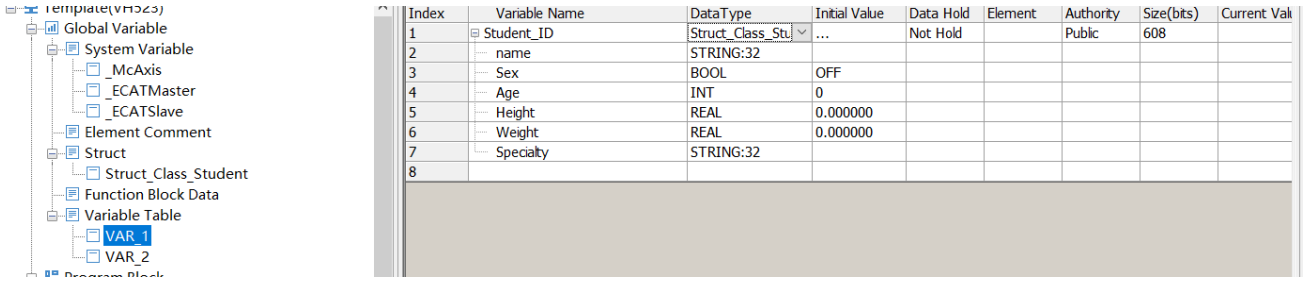
1. In the Project Manager, right-click "Struct" and select "New Struct" to create a struct.



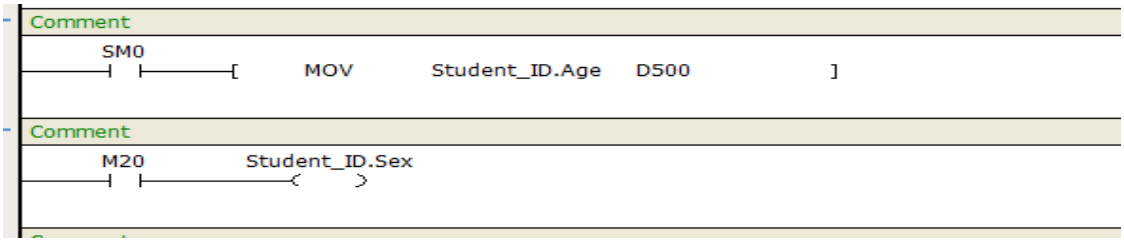
2. Right-click the new struct to rename it (e.g., "Class_Student"), and double-click "Struct_Class_Student" to add its members.



3. Then select the “Struct” as the data type. Double-click the “…” to set initial values to its members, as shown in the figure below:



4. Access struct members in programs follows the format: “<variable_name>.<member_name>”, as shown in the figure below.

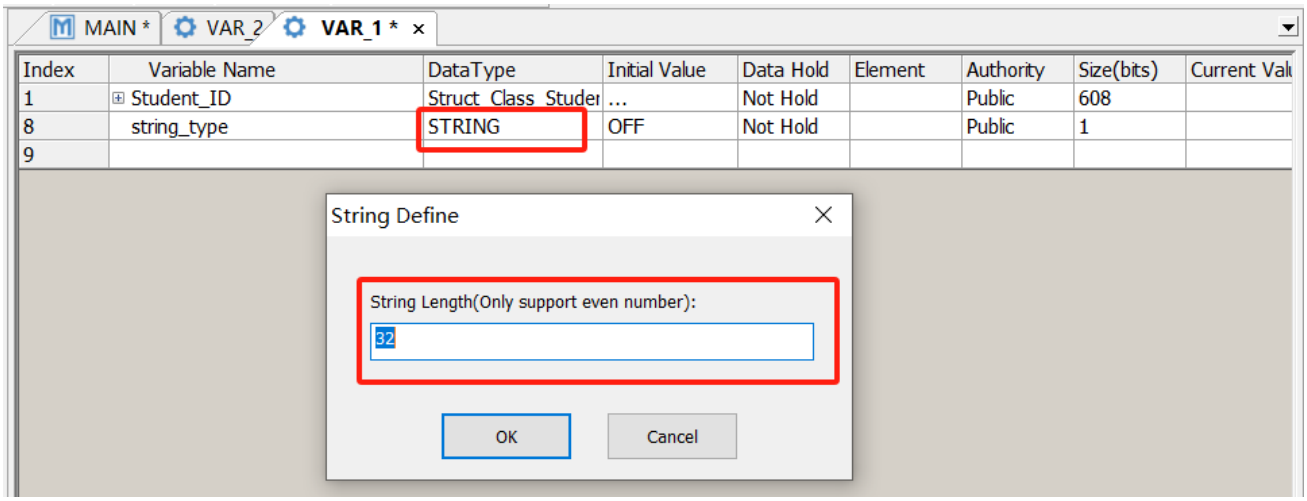


4.7 String Variable

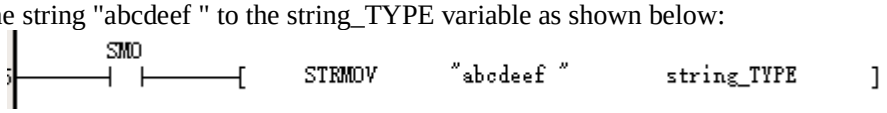
A string is a sequence of one or more characters (including numbers, letters, spaces, etc.), such as “This is an array of strings”. Note: Enclosing double quotes are not part of the string but indicate its boundaries to the compiler.

String Creation

1. String variables can be defined in either the Variable Table or Program interface. Select "STRING" from the Data Type dropdown menu, then specify the string length in the pop-up dialog box. The default length is 32 bytes (maximum 256 bytes), with the final byte reserved for the termination character.



2. Assign the string "abcdeef " to the string_TYPE variable as shown below:



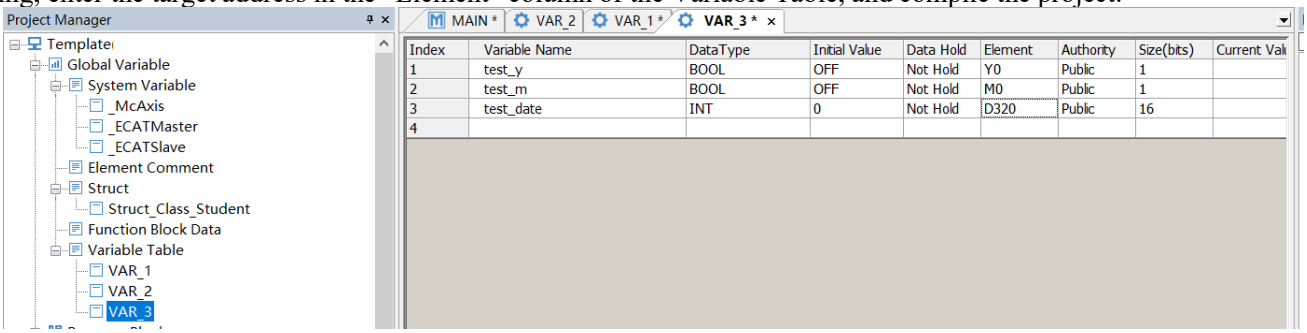
Guidelines:

- Once declared, variables can be referenced by name directly in code without reallocating soft element addresses.
- For standard variables, reference by name directly.
- For array variables, access elements using [index] (indexing starts at 0).
- For struct variables, access members via “struct_variable_name.member_variable”.

4.8 Variable Address Binding

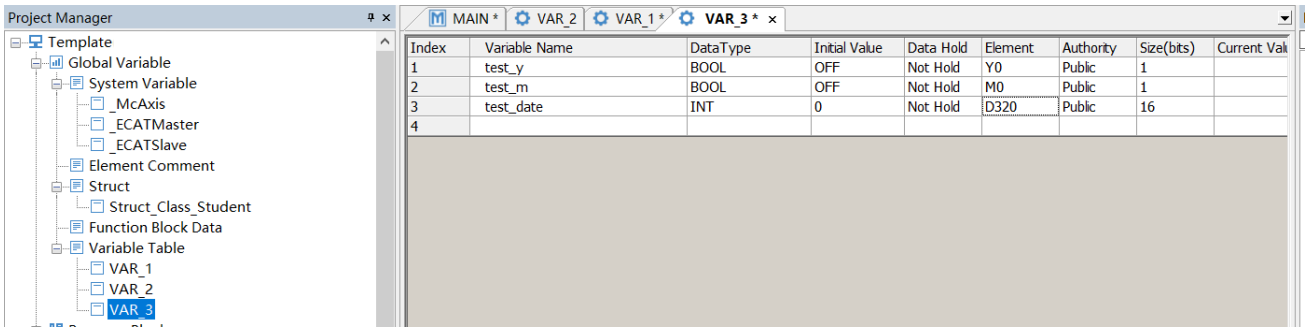
4.8.1 Overview

Custom variables can be bound to soft element addresses to establish shared memory locations. To implement binding, enter the target address in the “Element” column of the Variable Table, and compile the project.



4.8.2 Variable Properties

When bound to soft elements, custom variables inherit their “Data Hold” property. For example, as shown in the figure below, M0 is “Hold”, so the variable Test_M bound to it inherits the “Hold” property. Conversely, Test_Data bound to D320 inherits the “Not Hold” property. The variable's “Data Hold” property dynamically adjusts based on the bound soft element, determined by the user-configuration.

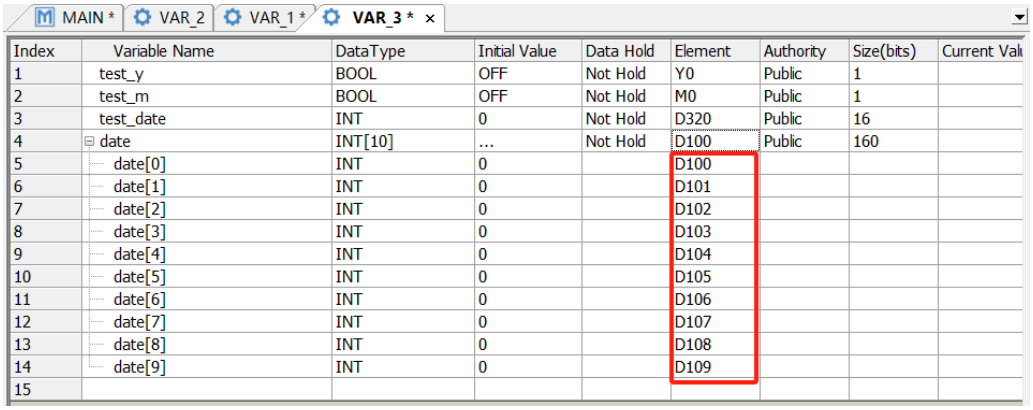


4.8.3 Basic Variables Binding

- BOOL variables (1-bit length) can only be bound to bit elements; INT (16-bit) can be bound to a single word element; DINT and REAL (32-bit) can be bound to two consecutive word elements.
- STRING variables have user-defined lengths aligned to 2-byte boundaries. For example, a length of 5 occupies 6 bytes. After bound to DO, it occupies D0, D1, and D2.

4.8.4 Array Variables Binding

To bind an array variable to soft elements, enter the address in the "Initial Value>Element" column of the variable table, as shown in the figure below



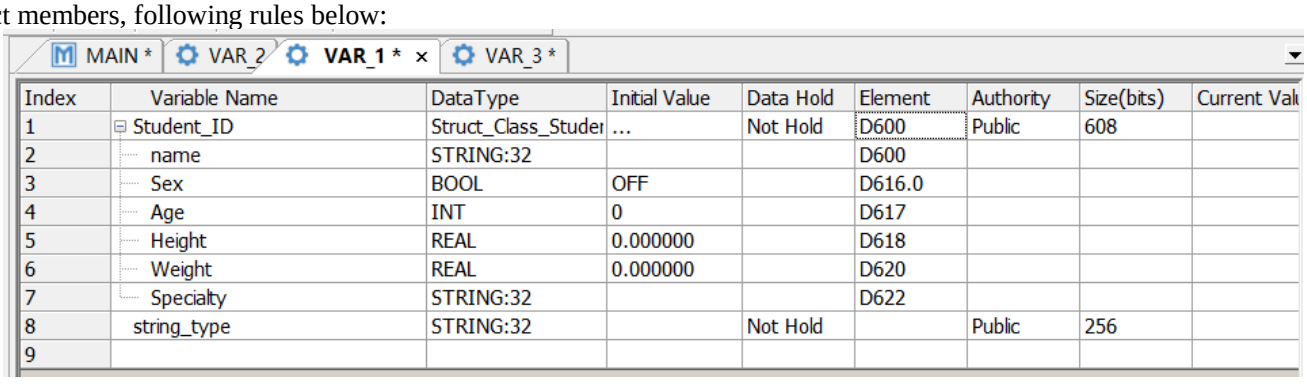
- Word-type variables occupy corresponding number of word elements:
  An INT variable (16-bit) occupies one 16-bit word element; a DINT or a REAL variable (32-bit) occupies two consecutive 16-bit word elements.
- STRING array variables maintain 2-byte alignment, occupying sequential word elements proportional to their length.
- BOOL array variables occupy the corresponding number of bit elements.
- Array variables can only bind to elements of matching types (word variables to word elements, bit variables to bit elements).

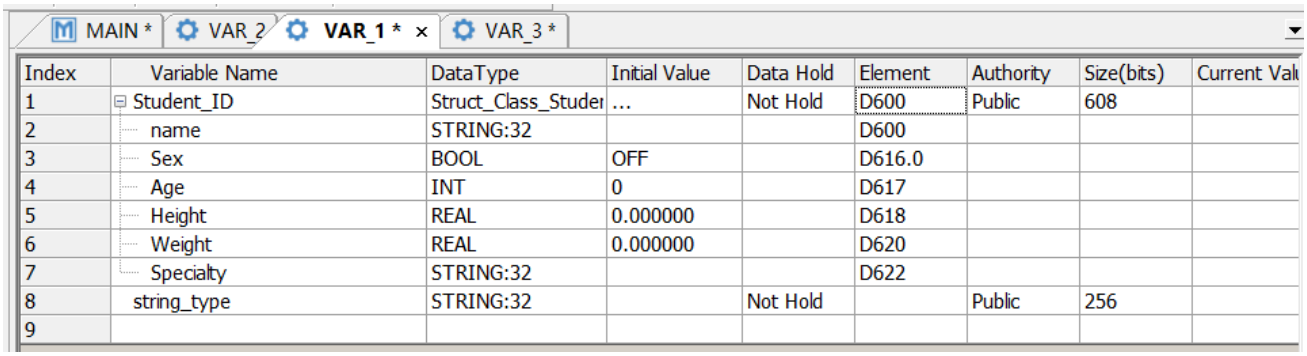
  Example:

A BOOL array variable Array_0 (length 10) bound to M0 occupies M0–M9.
An INT array variable Array_1 (length 10) bound to D0 occupies D0–D9.

4.8.5 Struct Variables Binding

To bind a struct variable to soft elements, enter the address (bit devices are invalid) in the "Initial Value>Element" column of the variable table. Then click "OK" to let AutoSoft automatically generate addresses for struct members, following rules below:





MAIN * | VAR_2 | VAR_1 * | VAR_3 *

Index	Variable Name	DataType	Initial Value	Data Hold	Element	Authority	Size(bits)	Current Val
1	Student_ID	Struct_Class_Studer	...	Not Hold	D600	Public	608	
2	name	STRING:32			D600			
3	Sex	BOOL	OFF		D616.0			
4	Age	INT	0		D617			
5	Height	REAL	0.000000		D618			
6	Weight	REAL	0.000000		D620			
7	Specialty	STRING:32			D622			
8	string_type	STRING:32		Not Hold		Public	256	
9								

(1) INT variables occupy one 16-bit word element; REAL and DINT variables occupy two consecutive 16-bit word elements.

(2) Continuous BOOL members are aligned to 16-bit, starting from bit 0, with addresses incrementing by 1 bit; non-continuous BOOL members are independently aligned to 16-bit boundaries.

(3) Array and struct variables are aligned to 16-bit boundaries as a whole.

For example: When a struct variable Stru_Student_ID is bound to D600, the binding addresses are as follows:

No.	Variable Name	Data Type	Element
1	Name	STRING (32 bytes)	D600~D615
2	Gender	BOOL	D616.0
3	Age	INT	D617
4	Height	REAL	D618
5	Weight	REAL	D620
6	Major	STRING (32 bytes)	D622~D637

4.9 Notes on Arrays

4.9.1 Indexing Rules

Only one variable can act as an array index within a single expression. Valid formats include array[index] or stru[index].var, where:

array: An array or struct array;

index, var, i: Variables;

stru: A struct.

4.9.2 Basic Indexing

● For array variables, only bit, word, double-word, or floating-point arrays are supported, and pointer arrays are not supported.

● For index variables (used as array index), only INT (16-bit) or DINT (32-bit) variables are allowed. Soft elements, bit variables, floating-point variables, and pointers are invalid. Index variables can be specific elements of an array or specific members of a struct (e.g., array[index[5]], array[stru.index]); not be array elements or struct members with variable indexes (e.g., array[index[i]], array[stru[i].index]).

4.9.3 Complex Indexing

● Using array elements as operands is supported, with the index variable placed last (e.g., array[index], stru.array[index], stru1[3].stru2.array[index], stru1.stru2.stru3.array[index]).

● Using struct array members as operands is supported, with the index variable placed in the middle (e.g., stru[index].var, stru1[index].stru2.var, stru1.stru2[5].stru3[index].array[3]).

● Struct arrays with dual/multiple nested variables are not supported (e.g., stru[index1].array[index2]).

● Dual/multi dimensional arrays are not supported (e.g., array[index1][index2]).

Notes:

➢ The operands of ZSET/ZRST, PTxxx, and SFC instructions do not support arrays with variable indexing.

➢ For instructions requiring consecutive array-type operands (e.g., BMOV), arr[index] is allowed, but stru[index].var (non-consecutive struct array elements) is invalid. Use loop instructions for non-consecutive

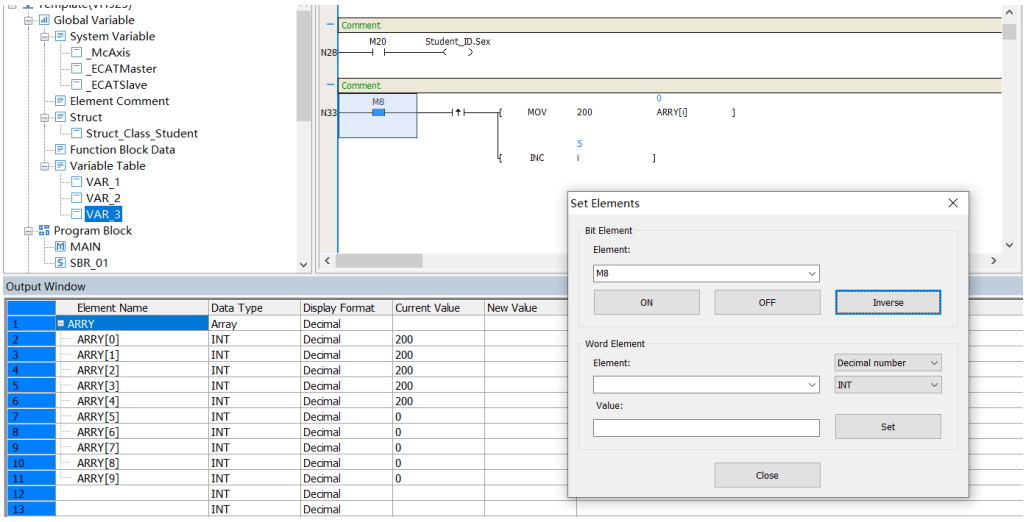
batch assignments.

- Variable-indexed arrays are recommended for single-cycle instructions only. Avoid using them in multi-cycle instructions. If unavoidable, ensure strict control over logic and timing to prevent execution conflicts (e.g., pulse output axes during value transitions).

4.9.4 Programming Example

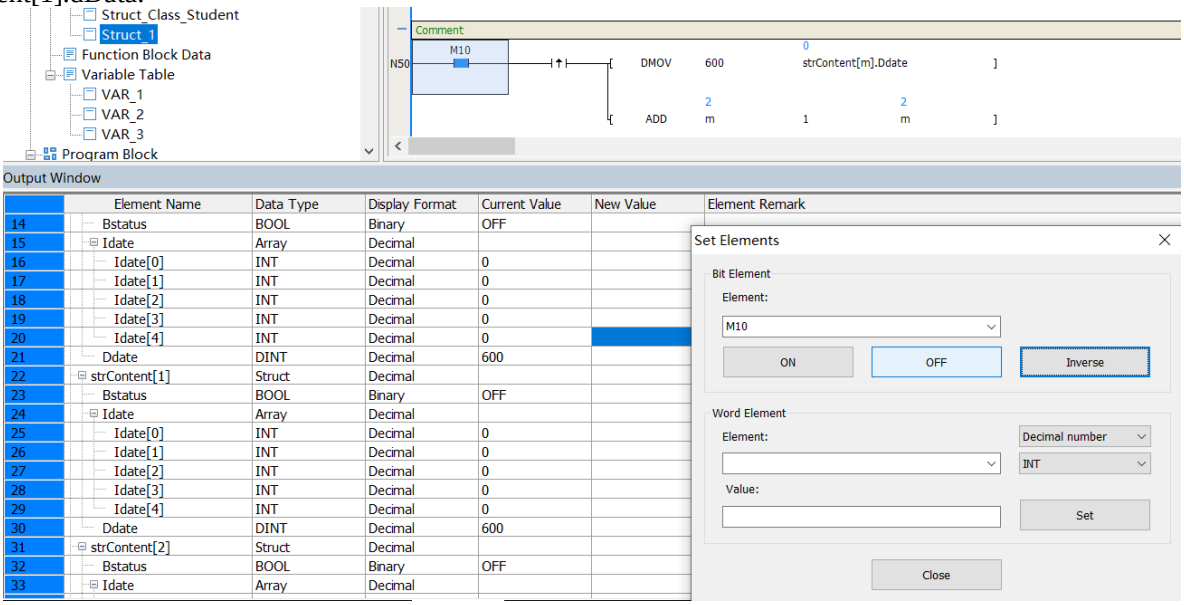
(I) Assigning Values to Array Elements

When M8 turns ON, assign 200 to ARRY_INT[0] and set i = 1. On the next M8 trigger, assign 200 to ARRY_INT[1]. Execute the assigning for five times.



(II) Modifying Struct Array Members

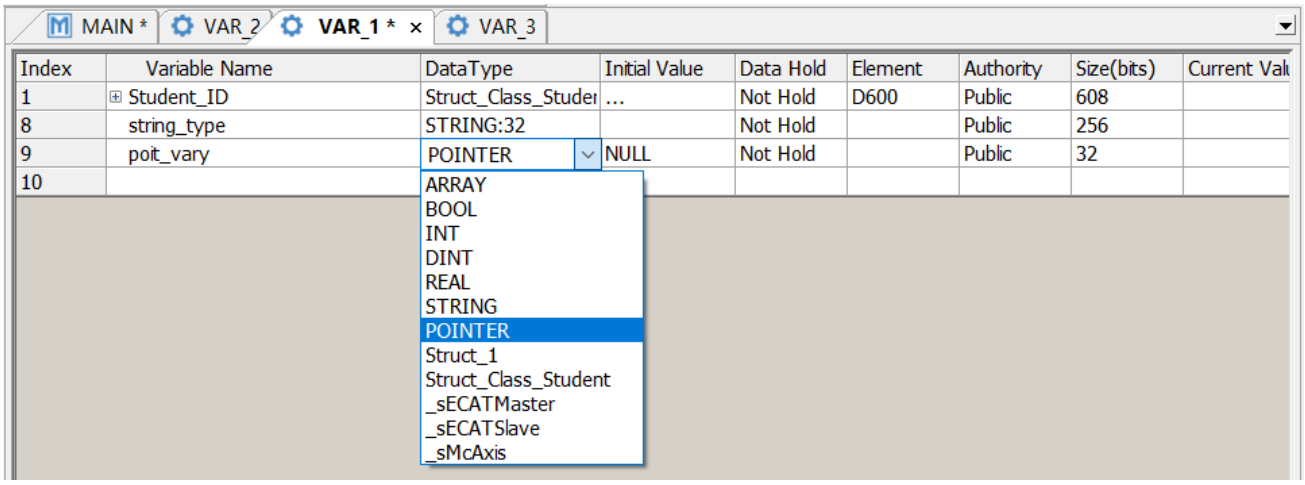
When M10 turns ON, assign 600 to strContent[0].dData and set m = 1. On the next M10 trigger, assign 600 to strContent[1].dData.



4.10 Pointer-Type Variables

4.10.1 Definition

A pointer represents a memory address, and pointer variables store memory addresses. They must be declared before storing addresses of other variables. To declare a pointer variable: Enter the variable name in the variable table, and select "POINTER" as the data type. The initial value is NULL, and data is “Not Hold”, as shown below.



Pointer-type variables support address operations and indirect addressing operations. Address Operations Instructions for pointer address operations are listed in the table below, enabling functions like address acquisition, offset calculation, and address comparison.

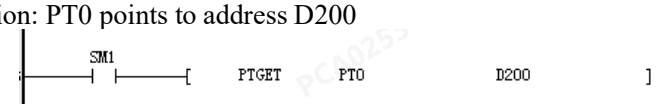
Table 3-2 Address Operation Instructions

Instruction	Description
PTGET	Get pointer address (Single-word)
DPTGET	Get pointer address (Dual word)
RPTGET	Get pointer address (Floating point)
PTINC	Increment pointer address by 1
PTDEC	Decrement pointer address by 1
PTADD	Add offset to pointer address
PTSUB	Subtract offset from pointer address
PT>, PT>=, PT<, PT<=, PT=, PT<>	Pointer contact comparison (>, ≥, <, ≤, =, ≠)
PTMOV	Assign values to pointer variables

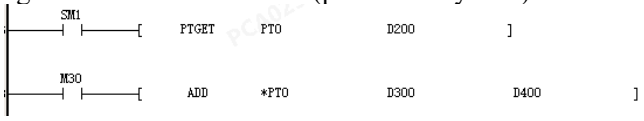
Indirect Addressing: When pointer variables are used in instructions beyond explicit address operations (listed in dedicated tables), they perform indirect addressing on referenced elements or array variable values. In programming, this is denoted as "*pointer_var".

Examples:

- Pointer address operation: PT0 points to address D200



- Pointer indirect addressing: Adds the value at D200 (pointed to by PT0) to D300 and stores the result in D400



Note:

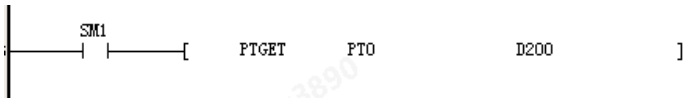
➢ The programming software automatically prepends "*" to pointer variables used in non-address operation instructions. Users may also manually add the "*" prefix.

4.10.2 Address Pointed by Pointer Variables

The address pointed to by a pointer variable can be acquired using the instruction PTGET.

Example

1. When the instruction's power flow is active, pointer variable PT0 points to D200, assigning the address of the D200 soft element to PT0.

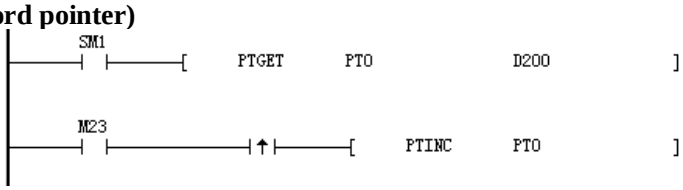


2. Pointer variables can target bit elements (X, Y, M, S), word elements (D, R, W), or user-defined array variables.

4.10.3 PT Pointer Address Operations

After obtaining the pointer address, arithmetic operations (addition/subtraction) can be performed on the pointer variable address to offset the address of the targeted element.

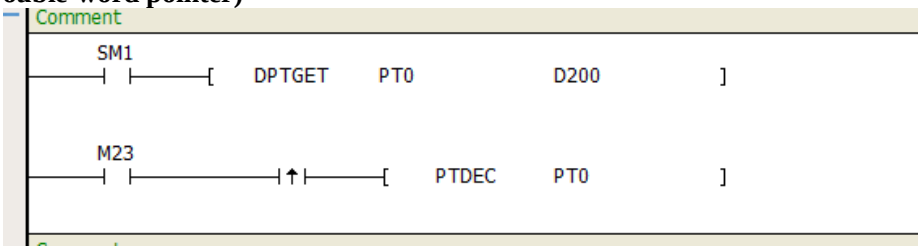
Example 1 (Single-word pointer)



When M23 is ON, the address of the soft element pointed to by pointer variable PT0 is incremented by 1. For instance: If PT0 originally points to D200, after executing the PTINC instruction, PT0 points to D201. The system automatically adjusts the offset increment based on the type of the targeted element or array variable.

Current PT0 Pointer	After PTDEC Execution
D200	D201
M200	M201
diVal[0]	diVal[1]

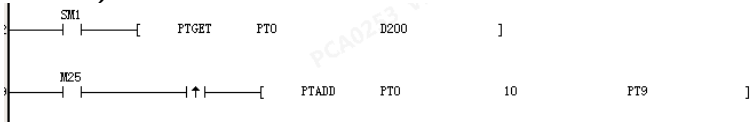
Example 2 (Double-word pointer)



When M23 is ON, the address of the soft element pointed to by pointer variable PT0 is decremented by 1. For instance: If PT0 originally points to D200, after executing the PTDEC instruction, PT0 points to D198. The system automatically adjusts the offset decrement based on the type of the targeted element or array variable.

Current PT0 Pointer	After PTDEC Execution
D200	D198
M200	M199
diVal[5]	diVal[4]

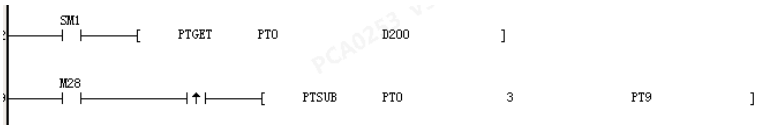
Example 3 (Pointer address)



When M25 is ON, the PTADD instruction assigns the address of the soft element pointed to by PT0, incremented by 10, to PT9. For example: If PT0 originally points to D200, after executing PTADD, PT9 points to D210.

Current PT0 Pointer	After PTADD Execution
D200	D210
M200	M210
diVal[5]	diVal[15]

Example 4



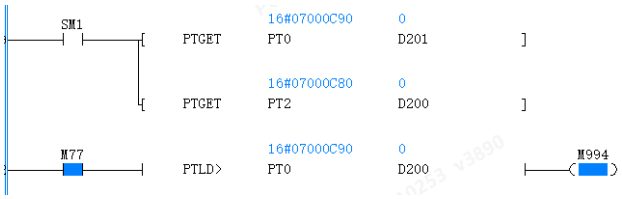
When M28 is ON, the PTSUB instruction assigns the address of the soft element pointed to by PT0, decremented by 3, to PT9. For example:

Current PT0 Pointer	After PTSUB Execution
D200	D197
M200	M197
diVal[10]	diVal[7]

Note:

- All the above examples require the PTGET instruction to first acquire the pointer address. When a pointer targets an array variable, ensure bounds checking is performed during address operations to prevent out-of-range access.

Example 5

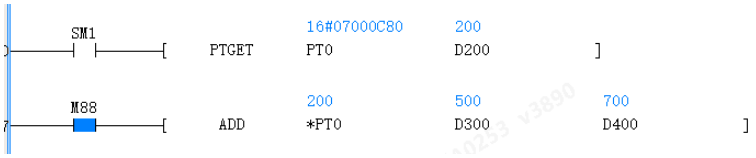


When M77 is ON, the PTLD> instruction checks if the address of pointer variable PT0 exceeds D200. If PT0 points to D201, output M0 is set to ON. Similar comparison instructions (PT>=, PT<, PT<=, PT=, PT<>) enable pointer address checks.

4.10.4 Indirect Addressing

Once a pointer variable acquires an address through address operation instructions, it can be used in other instructions to perform indirect addressing on the soft element or array variable it points to.

Example



When the instruction's power flow is active, the value of the soft element pointed to by pointer variable PT0 is added to D300. For instance, if PT0 points to D200, the result is stored in D400 as D10+D100.

Note:

- To use a pointer variable for indirect addressing, a valid pointer address must first be obtained via pointer address operation instructions.

4.11 System Variables

4.11.1 Overview

System Variables function enables monitoring of master status, slave status, and axis parameter information.

System Variables	Description
_McAxis	Data for motion control axes
_ECATMaster	EtherCAT master status
_ECATSlave	EtherCAT slave status

4.11.2_McAxis Parameters

Table 4-4 Axis Operation Status

Name	Data	Description	R/W

	Type		
dPulsesPreCycle	DINT	Pulses per cycle (motor/encoder)	W/R
fDistancePreCycle	REAL	Distance per cycle (worktable)	W/R
dNumerator	DINT	Gear ratio numerator	W/R
dDenorminator	DINT	Gear ratio denominator	W/R
bDirection	BOOL	Direction	W/R
bSoftLimitEnable	BOOL	Software limit enable	W/R
fPLimit	REAL	Positive limit in linear mode	W/R
fNLimit	REAL	Negative limit in linear mode	W/R
iLineRotateMode	INT	Linear/rotational mode selection 0: linear; 1: rotational	W/R
fRotation	REAL	Cycle in rotational mode	W/R
EncodeMode	INT	Encoder mode (valid in bus servo axis) 1: absolute, 0: incremental	W/R
iHomeMethod	INT	Homing method	W/R
fHomeVelocity	REAL	Homing velocity	W/R
fHomeApproachVelocity	REAL	Homing approach velocity	W/R
fHomeAcceleration	REAL	Homing acceleration	W/R
iHomeTimeOut	INT	Homing timeout	W/R
bPLimitTerminalPolarity	BOOL	Positive limit terminal polarity (valid in local pulse axis)	W/R
bNLimitTerminalPolarity	BOOL	Negative limit terminal polarity (valid in local pulse axis)	W/R
bHomeTerminaPolarity	BOOL	Home terminal polarity (valid in local pulse axis)	W/R
iPLimitType	INT	Positive limit input type (valid in local pulse axis) 0: X, 1: M, 2: S	W/R
iPLimitID	INT	Positive limit input number (valid in local pulse axis) X0~7/M0~M/S0~S	W/R
iNLimitType	INT	Negative limit input type (valid in local pulse axis) 0: X, 1: M, 2: S	W/R
iNLimitID	INT	Negative limit input number (valid in local pulse axis)	W/R
iHomeInType	INT	Home input type (valid in local pulse axis) 0: X, 1: M, 2: S	W/R
iHomeInID	INT	Home input number (valid in local pulse axis)	W/R
iEncoderInType	INT	Local encoder input type (valid in local encoder)	W/R
iEncoderRstInEn	INT	Local encoder reset input enable (valid in local encoder)	W/R
iEncoderRstInID	INT	Local encoder reset input ID (valid in local encoder)	W/R
iEncoderEnInEn	INT	Local encoder enable input enable (valid in local encoder)	W/R
iEncoderEnInID	INT	Local encoder enable input ID (valid in local encoder)	W/R
iEncoderPreSetInEn	INT	Local encoder preset input enable (valid in local encoder)	W/R
iEncoderPreSetInID	INT	Local encoder preset input ID (valid in local encoder)	W/R
iPluseMethod	INT	Pulse output method (valid in local pulse axis)	W/R
bTouchProbeEn0	BOOL	Touch probe 0 input enable (valid in local pulse axis)	W/R
iTouchProbeID0	INT	Touch probe 0 ID (valid in local pulse axis) 0~7X0~7	W/R
bTouchProbeEn1	BOOL	Touch probe 1 input enable (valid in local pulse axis)	W/R
iTouchProbeID1	INT	Touch probe 1 ID (valid in local pulse axis) 0~7X0~7	W/R
bCmpEnable	BOOL	Comparison output enable (valid in local pulse axis)	W/R
iCmpOutID	INT	Comparison output port ID (valid in local pulse axis) 0~7: Y0~7	W/R
iCmpOutUnit	INT	Comparison output unit (valid in local pulse axis)	W/R
iCmpOutWidth	INT	Comparison output width (valid in local pulse axis)	W/R
fErrorStopDeceleration	REAL	Axis error stop deceleration	W/R
fFollowErrorWindow	REAL	Follow error window	W/R
fMaxVelocity	REAL	Maximum velocity	W/R
fMaxJerkVelocity	REAL	Maximum jerk velocity	W/R
fMaxAcc	REAL	Maximum acceleration	W/R

iMaxTorque	INT	Maximum torque	W/R
dConfigReserved[16]	Struct	Reserved	W/R
iMapped	INT	Axis parameter mapping flag	R/O
iType	INT	Axis type	R/O
iSlave	INT	Axis mapping ID	R/O
iVirtualAxis	INT	Virtual axis flag	R/O
iEnableStatus	INT	Axis enable status	R/O
iAlmStatus	INT	Axis alarm status	R/O
iAxisOprationStatus	INT	Axis operation status	R/O
iCheckDoingStatus	INT	Axis motion status 0: stopped, 1: running	R/O
iInterpNum	INT	Interpolation channel number	R/O
iInterpBitType	INT	Control mode flags bit0: P_Task, bit1: S_Task, bit2: F_Task, bit3: cam	R/O
dCommandPulse	DINT	Current command position	R/O
hEncoderCounter	DINT	Feedback-to-command position deviation	R/O
dEncoderPos	DINT	Encoder feedback position	R/O
dStatusReserved[16]	DINT	Reserved	R/O

4.11.3_ECATMaster Parameters

The EtherCAT master parameters include the master operation status and maximum cycle time, as shown in Table 4-2.

Table 4-2 Master Information

Name	Data Type	Description	R/W
bMasterEnableState	BOOL	Master enable status	RO
bLinkState	BOOL	Master link state	RO
dCycleTime	DINT	Master cycle time	RO
dTaskExeTime	DINT	Master task execution time	RO
iMasterState	INT	Master bus status	RO
iSlaveNumber	INT	Number of connected slaves	RO
iDcSlaveNumber	INT	Number of slaves supporting DC sync	RO
iLossPackeCounter	INT	Master packet loss cumulative counter	RO
dPdoInLength	DINT	Master PDO input length	RO
dPdoOutLength	DINT	Master PDO output length	RO
iCycleJitter	INT	Master sync cycle jitter	RO
iEthercatRun	INT	Master running flag 0: stopped, 1: initializing, 3: running	RO
iScanRaedy	INT	Master scan ready 0: not ready, 1: ready	RO
dReserved[31]	DINT	Reserved	RO

4.11.4_ECATSlave Parameters

Table 4-3 Slave Information

Name	Data Type	Description	R/W
_ ECATSlave	Array	EtherCAT slave operational status	RO
_ ECASlave[0]	Sruct	Slave 0 axis	RO
iState	INT	Slave current bus status	RO
iALstatescode	INT	AL states code	RO
iConfigaddr	INT	Configuration address	RO
iAliasaddr	INT	Slave alias address	RO
dEep_man	DINT	Slave equipment manufacturer ID	RO
dEep_id	DINT	Slave equipment ID	RO
iItype	INT	Interface type	RO
iDtype	INT	Device type	RO
iObits	INT	PDO output bits	RO
iObytes	INT	PDO output bytes	RO
iOstartbit	INT	PDO output start bit	RO
iIbits	INT	PDO input bits	RO
iIbytes	INT	PDO input bytes	RO

iIstartbit	INT	PDO input start bit	RO
iHasdc	INT	DC support	RO
iPtype	INT	PHY interface type	RO
iTopology	INT	Topology	RO
iActiveports	INT	Active ports	RO
iParent	INT	Parent slave ID	RO
iParentport	INT	Parent port ID	RO
iEntryport	INT	Entry port	RO
iSlotConfig	INT	Valid slot configuration	RO
iRdSlotsNum	INT	Read slot numbers	RO
dRdSlotsIds[31	ARRY	Read slot ID list	RO
dEep_rev	DINT	Slave equipment version	RO
iPackLossCounter	INT	Cumulative packet loss counter	RO
iPackLossSign	INT	Packet loss sign	RO

Note:

- _ECATSlave is a struct array with a length of 72. _ECASlave[0] data structure represents the information for Slave 0, including its operational status, packet loss count, and other relevant details. Subsequent axes follow the same structure for their respective slaves.

4.12 Timer

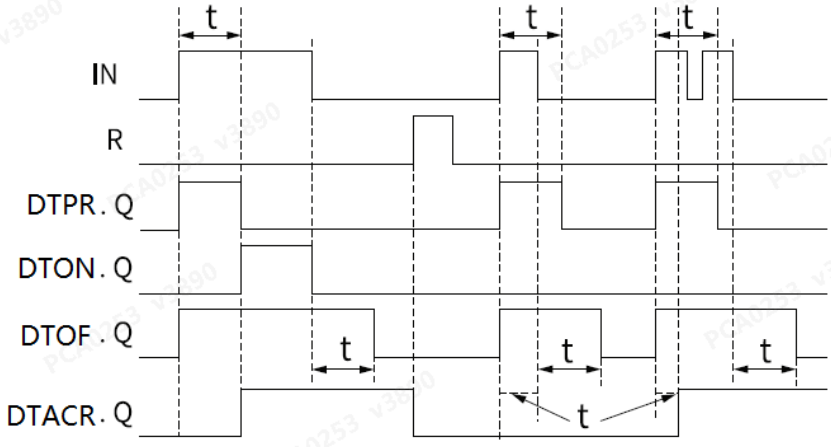
4.12.1 Overview

Based on the IEC 61131-3 standard timer specifications, four types of timer with enhanced reset functionality are defined as follows: Pulse Timer (DTPR), On-Delay Timer (DTON), Off-Delay Timer (DTOF), and Time Accumulation Timer (DTACR). They feature a 1ms time base with real-time updates to current values and status during execution. The program supports up to 4096 timer instructions. All timer types share identical instruction parameters:

Table 4-4 Timer Instruction Parameters

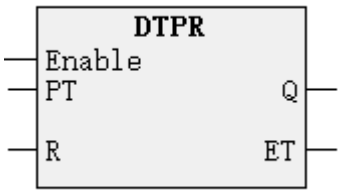
Name	Definition	Data Type	Description
Enable	Enable flow	/	Start input
PT	Input variable	DINT	Delay time
R	Input variable	BOOL	Reset input
Q	Output variable	BOOL	Timer output
ET	Output variable	DINT	Current elapsed time

Timer timing sequence operation:



4.12.2 Pulse Timer (DTPR)

When the Enable input flow transitions from OFF to ON, the timer starts timing, and output Q turns ON. Regardless of subsequent changes to the Enable input flow, Q remains ON for the duration specified by the PT parameter. Upon reaching the PT-specified time, Q turns OFF. During timing, ET outputs the current elapsed time. If the Enable input flow remains ON after the timer completes, the ET value is retained; if Enable is OFF, ET resets to 0.

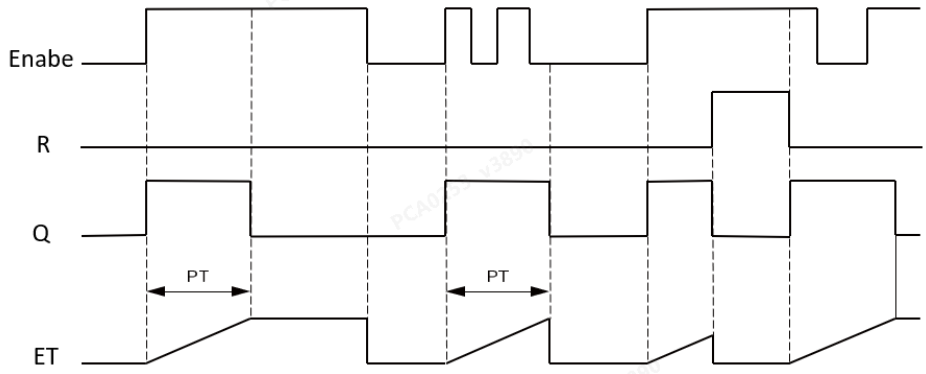


If the reset input R transitions from OFF to ON during timing, the TPR timer resets to 0 and output Q turns OFF. After the reset input R returns to OFF, the timer resumes timing if the Enable input flow is active.

PT: Setting range of 0ms~2147483647ms (~24 days maximum); if PT ≤ 0, it is handled as 0.

Timing Diagram

The timing diagram for parameters Enable, R, Q, and ET is as follows:



Note:

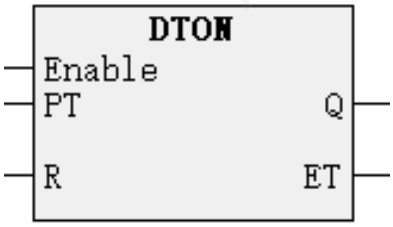
- Output parameters "ET" and "Q" are updated when the instruction is executed. Thus, the state change of "Q" occurs not at the exact moment when the elapsed time equals "PT" but at the first execution of the instruction after the elapsed time reaches "PT." This introduces a maximum delay of one scan cycle for the output parameters.

4.12.3 On-Delay Timer (DTON)

When the Enable input flow transitions from OFF to ON, the timer starts timing, and output Q remains OFF. While Enable remains ON, the timer runs for the duration specified by the PT parameter. Once the PT-specified time is reached, Q turns ON. If Enable turns OFF during or after timing, the timer stops, and Q resets to OFF.

During timing (with Enable ON), ET outputs the current elapsed time. If the timer completes and Enable remains ON, the ET value is retained; if Enable turns OFF, ET resets to 0.

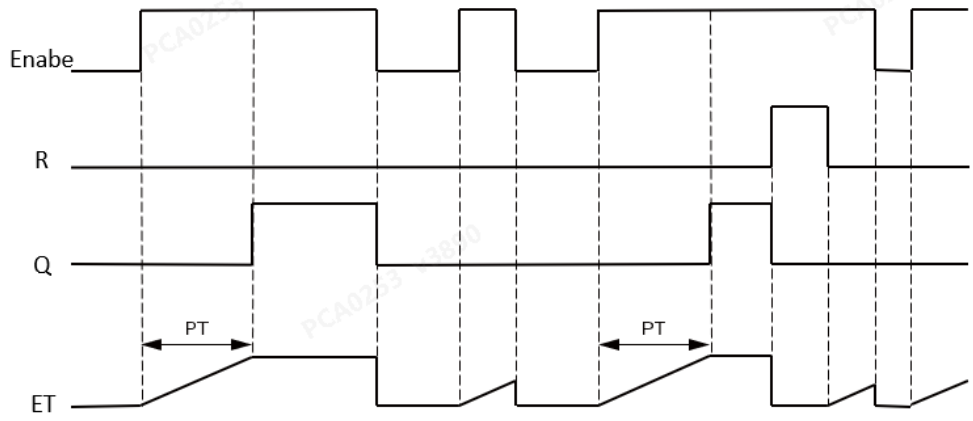
If the reset input R transitions from OFF to ON during timing, the DTON timer resets to 0 and output Q turns OFF. After R returns to OFF, the timer resumes counting only if the Enable input flow transitions from OFF to ON again.



PT: Setting range of 0ms~2147483647ms (~24 days maximum); if PT ≤ 0, it is handled as 0.

Timing Diagram

The timing diagram for parameters Enable, R, Q, and ET is as follows:



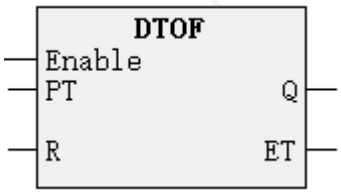
Note:

- Output parameters "ET" and "Q" are updated when the instruction is executed. Thus, the state change of "Q" occurs not at the exact moment when the elapsed time equals "PT" but at the first execution of the instruction after the elapsed time reaches "PT." This introduces a maximum delay of one scan cycle for the output parameters.

4.12.4 Off-Delay Timer (DTOF)

When the Enable input flow transitions from OFF to ON, the timer starts timing and output Q turns ON. When Enable transitions from ON to OFF, the timer runs for the duration specified by the PT parameter while Enable remains OFF. Q turns OFF once the PT-specified time is reached.

While Enable is ON, ET outputs 0. When Enable transitions from ON to OFF, ET outputs the current elapsed time during timing. After the timer completes, the ET value is retained.

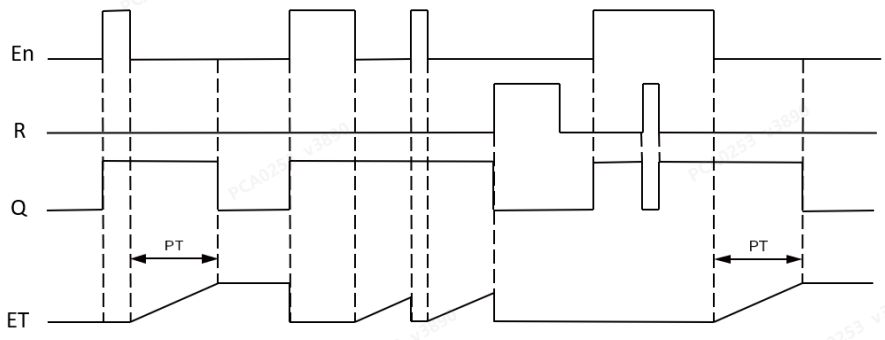


When Enable is ON and the reset input R transitions from OFF to ON, output Q turns OFF. If R returns to OFF, Q resumes ON. When Enable transitions from ON to OFF during or after timing, if the reset input R transitions from OFF to ON, output Q turns OFF and ET resets to 0. After R returns to OFF, the timer resumes counting only if the Enable input flow transitions from ON to OFF again.

PT: Setting range of 0ms~2147483647ms (~24 days maximum); if PT ≤ 0, it is handled as 0.

Timing Diagram

The timing diagram for parameters Enable, R, Q, and ET is as follows:



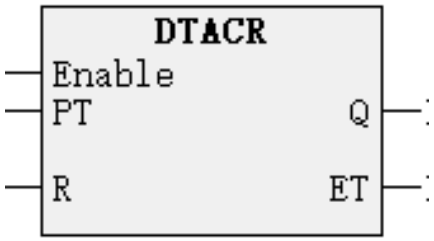
Note:

- Output parameters "ET" and "Q" are updated when the instruction is executed. Thus, the state change of "Q" occurs not at the exact moment when the elapsed time equals "PT" but at the first execution of the instruction after the elapsed time reaches "PT." This introduces a maximum delay of one scan cycle for the output parameters.

4.12.5 Time Accumulation Timer (DTACR)

When the Enable input flow is ON, the timer continues counting if its elapsed time has not reached the PT-specified time, keeping output Q OFF. Q turns ON once the PT-specified time is attained. If Enable transitions to OFF during timing (while previously ON), the timer retains its current count. When Enable returns to ON, the timer resumes counting from the retained value until PT is reached, at which point Q turns ON.

While Enable is ON, ET outputs the current elapsed time. After reaching the PT-specified time, ET retains its value. If Enable is OFF, the ET value remains unchanged.

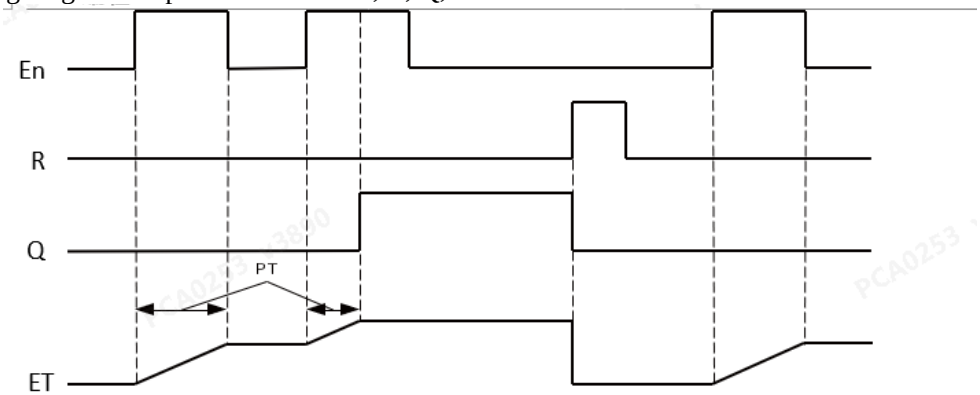


If the reset input R transitions from OFF to ON during or after timing, output Q turns OFF and ET resets to 0. After the reset input R returns to OFF, the timer resumes timing if DTACR input flow is active.

PT: Setting range of 0ms~2147483647ms (~24 days maximum); if PT ≤ 0, it is handled as 0.

Timing Diagram

The timing diagram for parameters Enable, R, Q, and ET is as follows:



Note:

- Output parameters "ET" and "Q" are updated when the instruction is executed. Thus, the state change of "Q" occurs not at the exact moment when the elapsed time equals "PT" but at the first execution of the instruction after the elapsed time reaches "PT." This introduces a maximum delay of one scan cycle for the output parameters.

Timer Usage Notes

1. For standard timers T0~T511, the time base values are as follows:

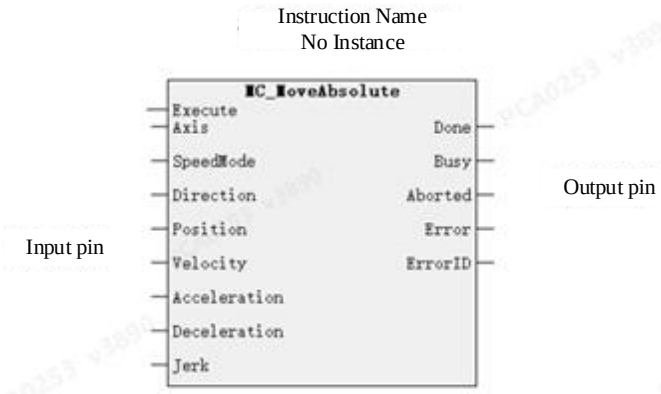
Timer Type	Base Value
T0~T209	100ms
T210~T255	10ms
T256~T511	1ms
DTPR/DTON/DTOF/DTACR	1ms

2. Avoid using T0~T511 timers (fixed addresses) in FB/FC function block encapsulation. Repeated calls to FB may cause double-coil issues for T timers. Use DTPR/DTON/DTOF/DTACR timers instead, which are automatically instantiated during multiple calls.

4.13 Graphical Block Instructions

4.13.1 Instruction Structure

Selected instructions support graphical block programming. A graphical block instruction consists of an instruction name, input terminals, and output terminals. The schematic below demonstrates this structure using an axis motion control block as a representative example:

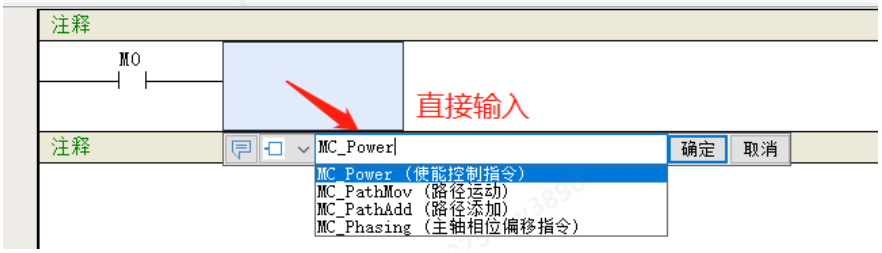


Floating-point values (e.g., target position and velocity) in the instructions use the single-precision floating-point (FLOAT) type. Therefore, values in PLC programs must adhere to the range and precision of the FLOAT type: a range of -3.4E38 to 3.4E38 and 7 significant digits. If a value exceeds 7 significant digits, the excess digits are automatically rounded.

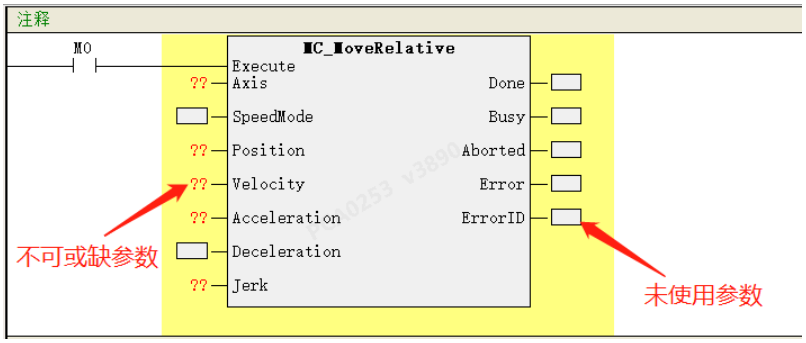
4.13.2 Implementation Workflow

When programming, entering the graphical block instruction name and pressing Enter inserts the instruction into the network. Parameters can be directly edited within the graphical block.

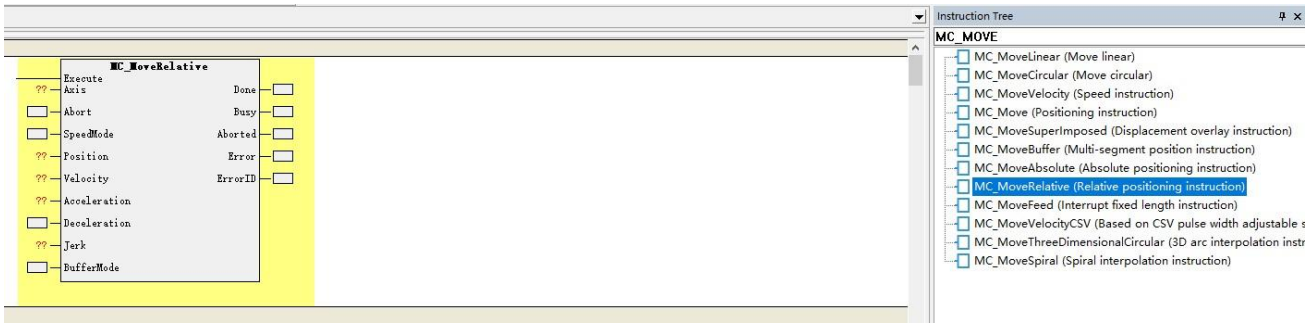
1. In ladder diagram editing, type the instruction name or select it from the prompt list to add the graphical block instruction, as shown below.



2. After inserting the instruction, press Enter to auto-invoke it. Parameters marked with "??" are mandatory, while "□" indicates optional parameters. If optional parameters are unused, default values are automatically assigned. Outputs cannot retrieve instruction status during program execution or monitoring/debugging if parameters are unused.

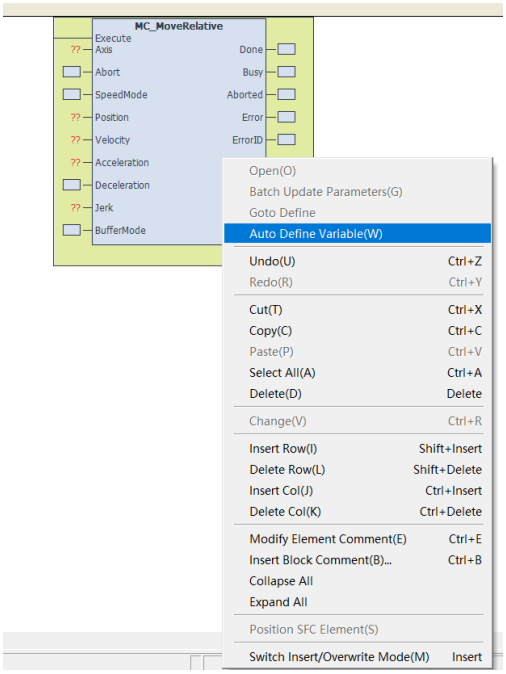


3. Search the instruction name in the Instruction Tree, and then double-click the desired instruction from the results, as shown below:

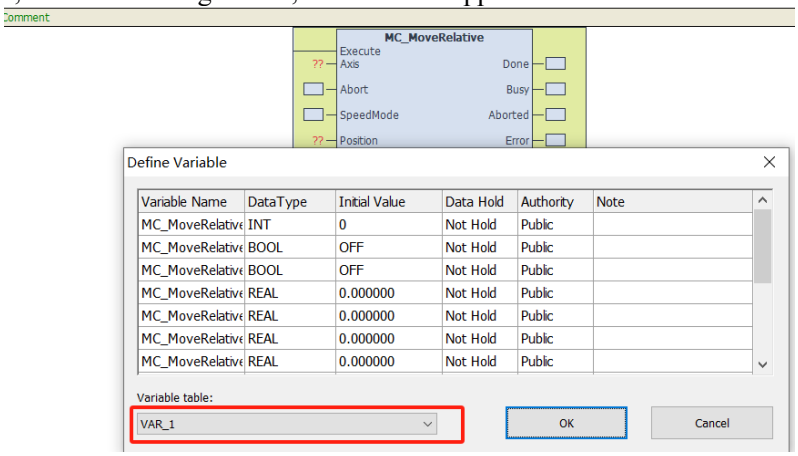


4.13.3 Quick Variable Addition

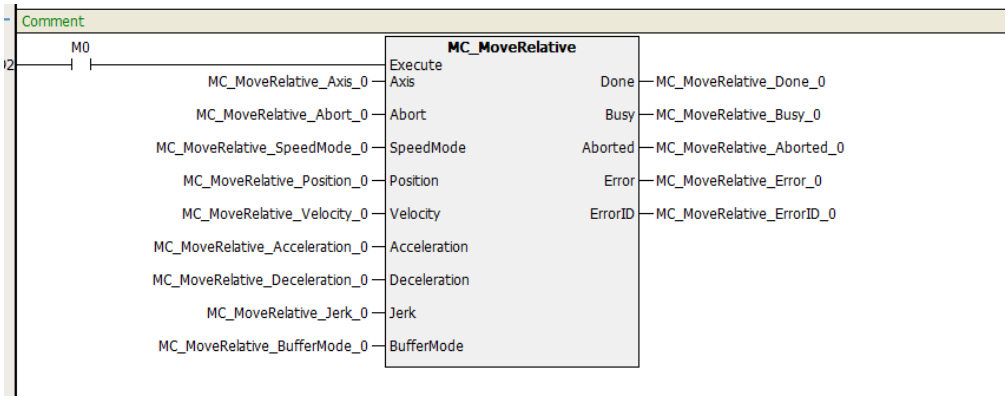
1. In ladder diagram programming, right-click the instruction and select "Auto Define Variables", as shown below.



2. The "Define Variable" dialog will open, allowing configuration of “Initial Value”, “Variable Table”, “Data Hold” attributes, etc. After configuration, the interface appears as shown below:



3. Click "OK", and the variables are automatically generated. They can be viewed and modified in the variable table, as illustrated below:



4.14 Subroutines

4.14.1 Overview

A subroutine constitutes a self-contained code module executable by the main program or other subroutines, functioning as an optional component within user programs.

Using subroutines in user programming offers the following advantages:

1. Reduces program size. Repetitive code segments with identical functionality can be encapsulated into a subroutine for repeated calls.
2. Simplifies program structure, particularly by streamlining the main program.
3. Improves cross-project portability.

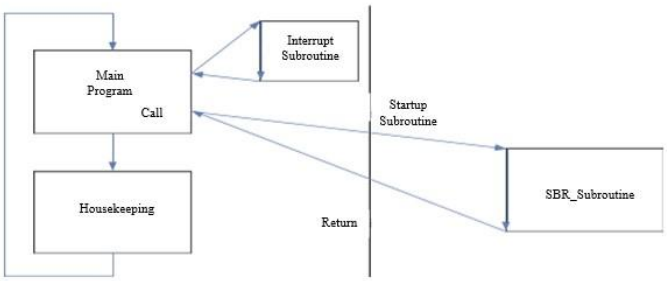
4.14.2 Subroutine Concepts

Subroutines are categorized as follows:

Flag	Name	Description
SBR	Subroutine	Supports up to 256 subroutines, including Standard Subroutine or Encrypted Subroutine. Encrypted and standard subroutines share the system's 200K-step capacity without restrictions.
INT	Interrupt Subroutine (Max. 68)	External interrupts: X0~X7 input interrupts, supporting rising edge, falling edge, and dual-edge detection.
		Timed interrupts: 3 points, with a configurable timebase of 1ms~32767ms.
		High-speed counter interrupts: 7 points.
		High-speed output completion interrupts: Y0~Y7.
		Comparison interrupts: 16 points (1~16).

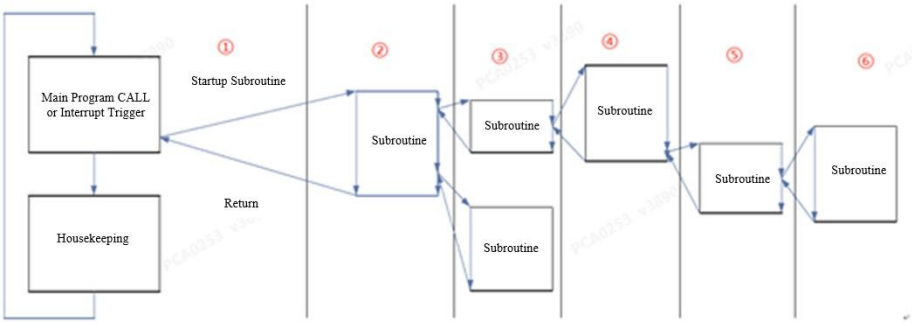
4.14.3 Subroutine Execution Mechanism

The execution logic of the main program and subroutines, along with the cyclic scan method, is illustrated below:



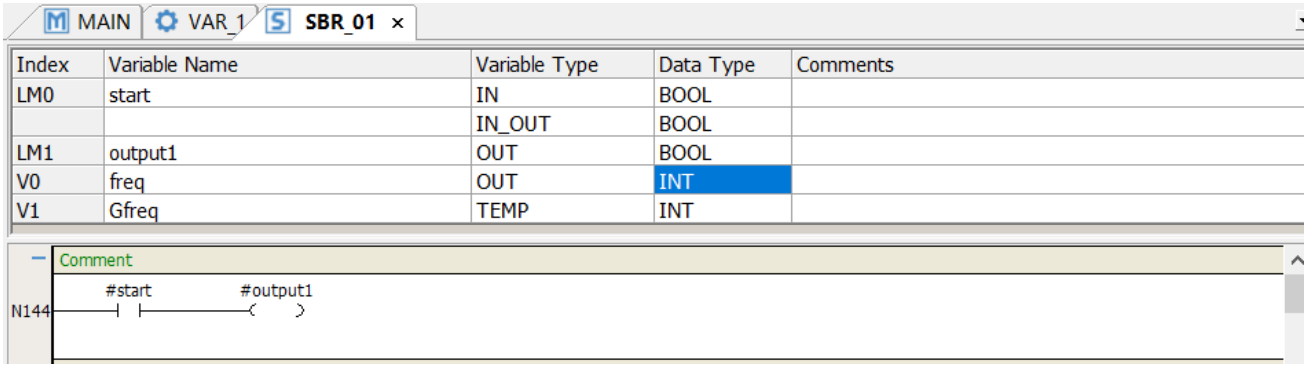
4.14.4 Subroutine Nesting Levels

Subroutines support up to 6 nesting levels, with the main program calling a subroutine counted as Level 1. Each subsequent call increases the nesting level by one. If a nested call returns, the nesting level is not incremented. The structure is shown below:



4.14.5 Subroutine Variable Table Definition

The subroutine variable table declares the subroutine's interface parameters and local variables (collectively referred to as variables) and defines their usage properties.



Subroutine Variable Attributes

The interface parameters and local variables (collectively referred to as variables) of a subroutine have the following attributes:

1. Variable Address

Each subroutine variable is assigned a fixed LM or V element address. The address is automatically allocated by the programming software based on the variable's data type, following contiguous address allocation principles.

2. Variable Name

A variable name (alias) can be assigned to interface parameters or local variables. Variables can be referenced by their names in the program.

3. Variable Type

Subroutine variables are categorized as IN, OUT, IN_OUT, or TEMP:

①IN-type: Passes input values to the subroutine upon invocation.

②OUT-type: Passes return values from the subroutine upon completion.

③IN_OUT-type: Passes input values upon invocation and return values upon completion.

④TEMP-type: Acts as a local variable valid only within the subroutine scope.

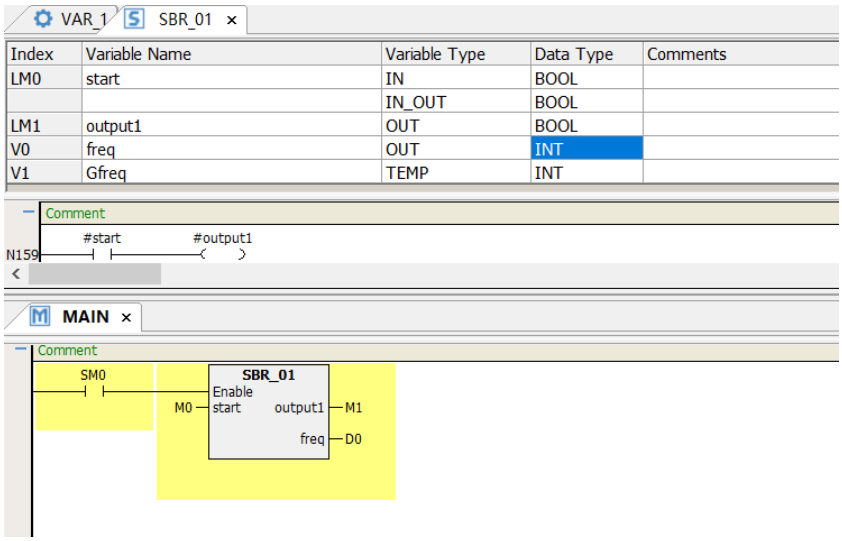
4. Variable Data Type

The variable data type defines the data width and range. The following table lists the available variable data types:

Data Type	Description	Occupied LM/V Addresses
BOOL	Bit-type variable	1 LM element address
INT	Signed integer variable	1 V element address
DINT	Signed long integer variable	2 consecutive V element addresses
REAL	Floating-point variable	2 consecutive V element addresses

4.14.6 Subroutine Parameter Passing

When calling a subroutine in the main program, if the subroutine defines local input/output variables, the corresponding values or global/temporary variable elements must be specified in the interface parameters. Ensure data type consistency between local variables and interface parameters.



4.14.7 Subroutine Usage Example

The following example illustrates how to create and call a subroutine:

1) Overview

Call subroutine SBR_1 from the main program MAIN to perform the addition of two integer constants (10 + 5), and assign the result (15) to D2.

2) Steps

1. Create a subroutine named SBR_1 in the project.

2. Program SBR_1.

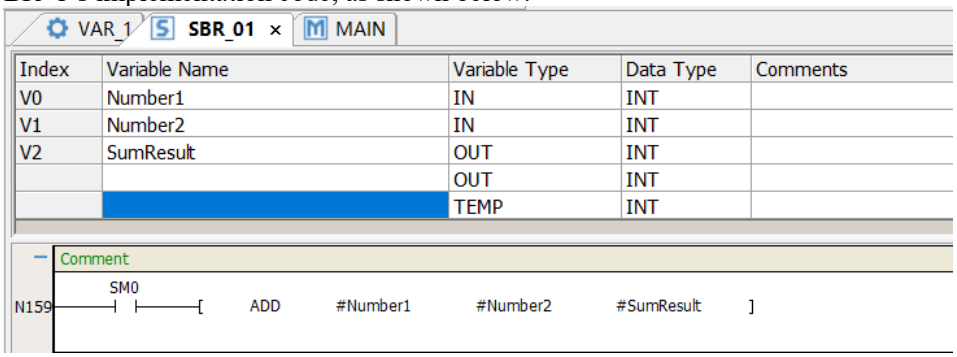
Define the subroutine's operand interface in its variable table:

Variable 1: Name = Number1 (IN-type parameter, INT data type), sequentially assigned V element address V0.

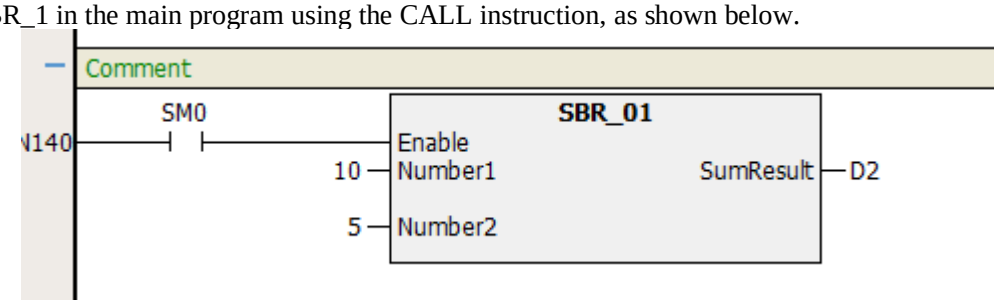
Variable 2: Name = Number2 (IN-type parameter, INT data type), sequentially assigned V element address V1.

Variable 3: Name = SumResult (OUT-type parameter, INT data type), sequentially assigned V element address V2.

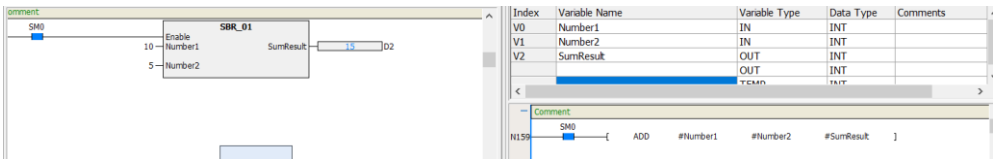
3. Program SBR_1's implementation code, as shown below.



4. Call SBR_1 in the main program using the CALL instruction, as shown below.



5. Compile, download, execute the user program, and perform online debugging and monitoring. Execution results are shown below.



4.14.8 Subroutine Usage Notes

The following considerations apply when writing or calling subroutines:

1. Nested Calls: Supports nested subroutine calls up to 6 levels.

Example of valid nesting:

MAIN → SBR1 → SBR2 → SBR3 → SBR4 → SBR5 → SBR6 (→ denotes subroutine calling via CALL instruction).

2. Prohibited Calls:

The following call structures are invalid.

Recursive Calls: MAIN → SBR0 → SBR0 (illegal).

Cyclic Calls: MAIN → SBR0 → SBR1 → SBR0 (illegal).

3. A subroutine variable table supports up to 16 BOOL-type and 16 word-type variables.
4. Ensure operand attributes in the CALL instruction match the variable attributes defined in the subroutine's variable table. The compiler verifies this matching.
5. Subroutine calls are not allowed within interrupt routines.
6. In the call editing area, type "CALL + Spacebar", then select the target subroutine from the pop-up list.

4.15 Interrupt Subroutine

4.15.1 Interrupt Overview

Interrupts execute immediately upon triggering, independent of the main program's scan cycle. This mitigates delays or timing deviations caused by scan cycles in high-speed signal processing, improving mechanical operation accuracy.

(I) Interrupt Handling Mechanism

1. When an interrupt event occurs and is enabled, its event number is added to the interrupt request queue, a FIFO (First-In-First-Out) queue with a depth of 8.

(II) System Handling of Interrupt Requests

1. If the interrupt request queue is not empty, the system interrupts normal user program execution.
2. The system checks the head entry of the queue (the earliest pending interrupt event number) and executes the corresponding user-defined interrupt subroutine.
3. After the interrupt subroutine completes (via a return instruction), the head entry is removed. Subsequent entries shift forward, and the next entry becomes the new head.
4. The system repeats the above steps until the queue is empty.
5. When the queue is empty, the system resumes the interrupted main program.

(III) The system processes one interrupt request at a time. New interrupts during processing are queued at the tail and handled sequentially after prior requests are resolved.

(IV) When the queue reaches its maximum capacity (8 entries), the system blocks new interrupt events until all queued requests are processed and the main program resumes execution.

Note

1. Avoid excessive execution time in interrupt routines, as this may lead to blocked interrupt events (lost requests), prolonged system scan cycles, and reduced main program efficiency.
2. Subroutine calls are prohibited within interrupt routines.
3. Use the immediate refresh instruction (REF) to refresh I/O instantly during interrupts. Note that REF execution time depends on the number of I/O points refreshed.
4. To trigger an interrupt request, enable the corresponding interrupt event via its SM flag (each interrupt type has dedicated SM enable/disable controls). Ensure the global interrupt enable flag is ON.
5. If an interrupt request occurs without a corresponding interrupt program in the user program, the system will still respond but execute a no-operation (NOP).

4.15.2 Timed Interrupts

Overview

Timed interrupts execute an interrupt subroutine once at a preset interval, independent of the scan cycle.

Applications

Timed interrupts are ideal for scenarios requiring periodic processing with strict timing, such as periodic sampling of analog inputs or waveform-based refreshing of analog outputs.

Timed Interrupt Resources for SH Series PLC

Timed Interrupts	Event Number	Timer Setting (SD)	Enable Control (SM)
0	22	SD47 (1~32767ms)	SM47
1	23	SD48 (1~32767ms)	SM48
2	24	SD49 (1~32767ms)	SM49

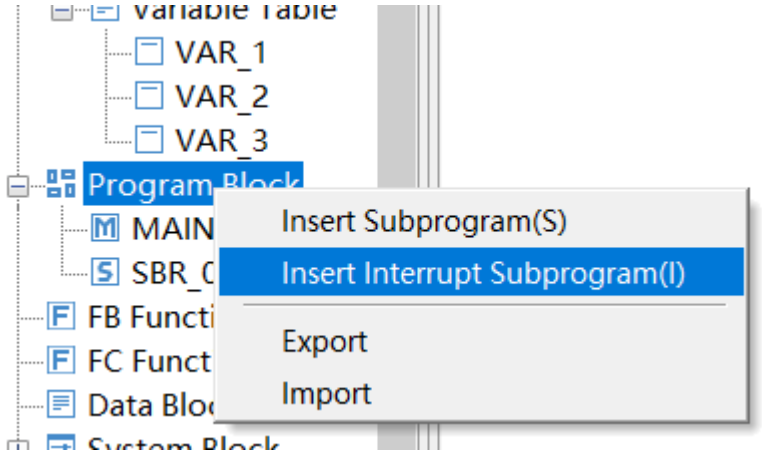
CAUTION

- When the timer interrupt is disabled, pending timer interrupts in the queue will still be executed.
- If the timer interrupt is re-enabled after being disabled, the timer resets and starts counting from zero.
- To modify the timer value setting during program execution, follow these steps: disable the timer interrupt, adjust the timer value, and then re-enable the interrupt.

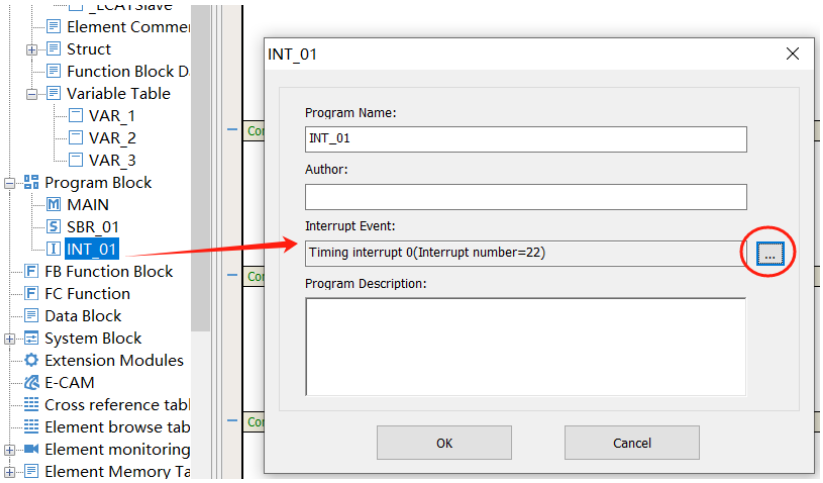
Example

Requirements: Use Timer Interrupt 0 to trigger an interrupt every 1 ms. Each time the interrupt is entered, the D200 register increments by 1.

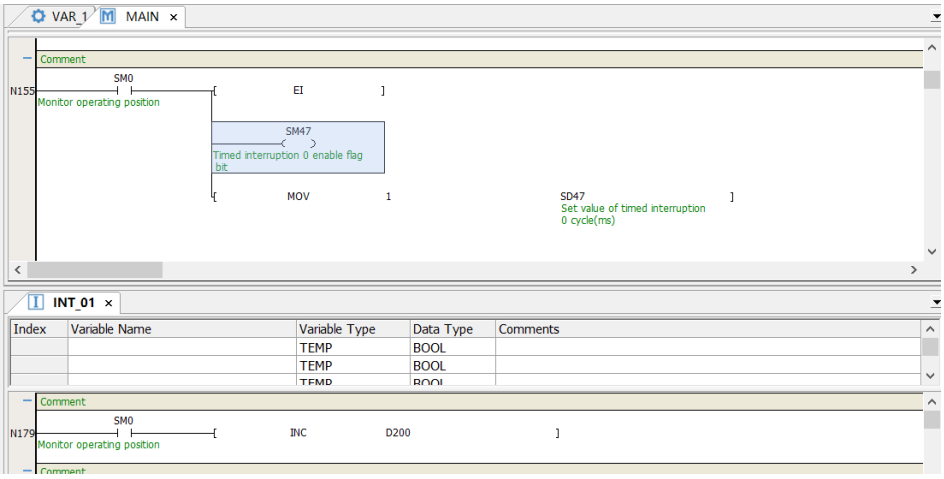
1. Right click the "Program Block" in the Project Manager area, and select "Insert Interrupt Subprogram" to create an interrupt subprogram (INT_01), as shown below.



2. Right-click the newly created INT_01 subprogram and select “Property”, and then set its property to “Timing interrupt 0”, as shown below.



3. Program in the Main program and INT_01 as shown below:



4.15.3 External Interrupt

(1) Description

Uses input signals from X0~X7 to execute interrupt subprograms.

(2) Application

Suitable for high-speed control or capturing short-duration pulses, as it processes external input signals independently of the PLC’s scan cycle.

(3) Precautions

i. The system’s maximum response frequency to external signals is 1 kHz. Events exceeding this frequency may be missed.

ii. Both rising-edge and falling-edge interrupts can be enabled on the same input port. All external interrupts are

active only when the global interrupt control (EI) is enabled and the corresponding interrupt enable SM bit is active.

iii. For SH300 and SH500 series, the maximum input pulse frequency for X0~X7 is <200 kHz.

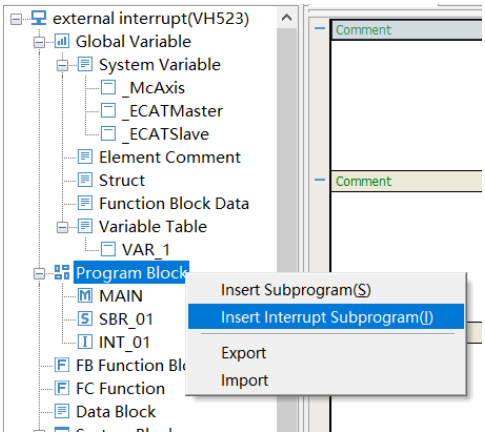
External Interrupt Number Assignment:

Rising-edge Interrupt				Falling-edge Interrupt			
Port	Interrupt No.	Interrupt Source	Interrupt Enable SM	Port	Interrupt No.	Interrupt Source	Interrupt Enable SM
X0	0	X0 Rising-edge Interrupt	SM25	X0	8	X0 Trailing-edge Interrupt	SM33
X1	1	X1 Rising-edge Interrupt	SM26	X1	9	X1 Trailing-edge Interrupt	SM34
X2	2	X2 Rising-edge Interrupt	SM27	X2	10	X2 Trailing-edge Interrupt	SM35
X3	3	X3 Rising-edge Interrupt	SM28	X3	11	X3 Trailing-edge Interrupt	SM36
X4	4	X4 Rising-edge Interrupt	SM29	X4	12	X4 Trailing-edge Interrupt	SM37
X5	5	X5 Rising-edge Interrupt	SM30	X5	13	X5 Trailing-edge Interrupt	SM38
X6	6	X6 Rising-edge Interrupt	SM31	X6	14	X6 Trailing-edge Interrupt	SM39
X7	7	X7 Rising-edge Interrupt	SM32	X7	15	X7 Trailing-edge Interrupt	SM40

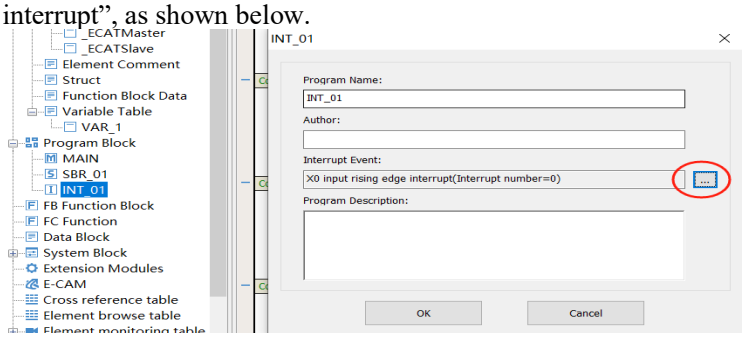
Example

Requirements: Trigger an interrupt on the rising-edge signal of X0 to increment the D0 register by 1 in the interrupt subprogram.

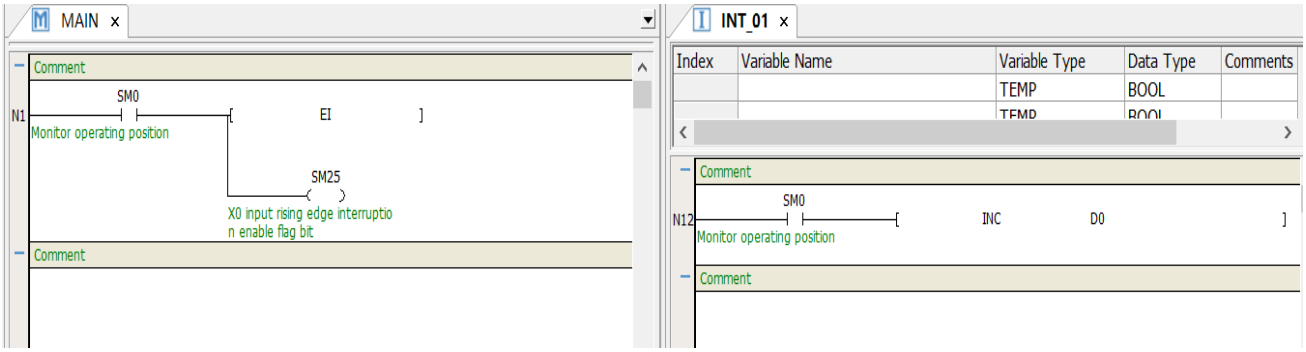
1. Right click the "Program Block" in the Project Manager area, and select “Insert Interrupt Subprogram”, as shown below.



2. Right-click the newly created INT_01 subprogram and select “Property”, and then set its property to “X0 input rising edge interrupt”, as shown below.



3. Program in the Main program and INT_01 as shown below:



4.15.4 High-speed Counter Interrupt

(1) Description

Triggers an interrupt based on the current value of the high-speed counter using the HCNT instruction. Used in conjunction with the DHSCI instruction, the interrupt subprogram executes when the counter's current value matches the threshold defined by DHSCI.

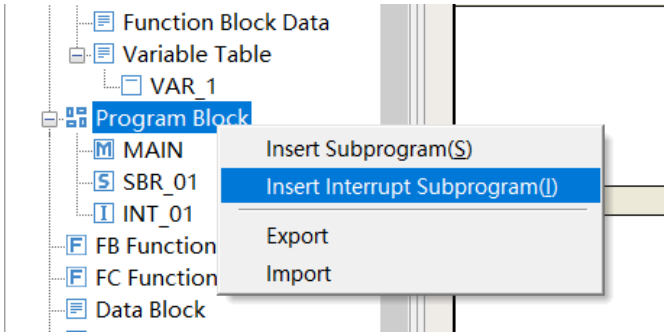
(2) Usage Conditions

High-speed counter interrupts are valid only when paired with the HCNT or DHSCI instruction, based on the counter's value. Users can program logic related to external pulse inputs in the high-speed interrupt program. All high-speed counter interrupts (33~40) are active only when the global interrupt control (EI) and the corresponding interrupt enable flag are enabled. The interrupt assignments are listed below:

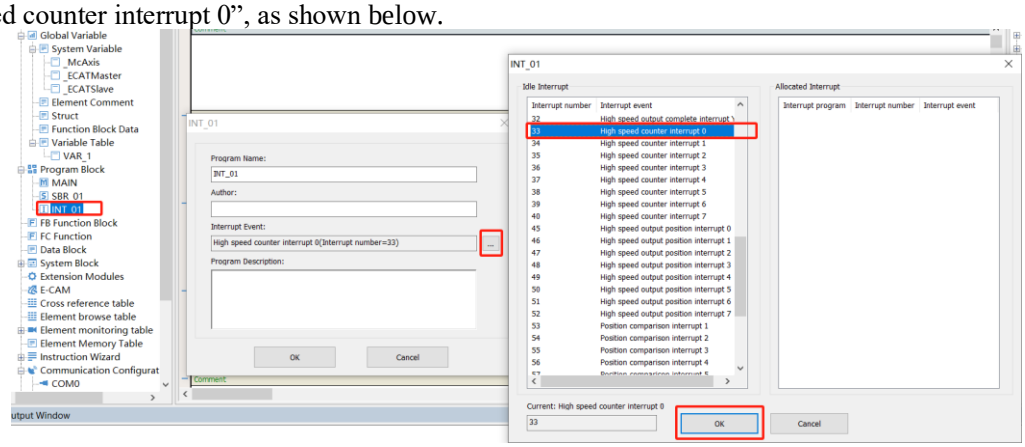
Event No.	Interrupt Event	Interrupt Enable SM
33	High-speed counter interrupt 0	SM58
34	High-speed counter interrupt 1	SM58
35	High-speed counter interrupt 2	SM58
36	High-speed counter interrupt 3	SM58
37	High-speed counter interrupt 4	SM58
38	High-speed counter interrupt 5	SM58
39	High-speed counter interrupt 6	SM58
40	High-speed counter interrupt 7	SM58

Example

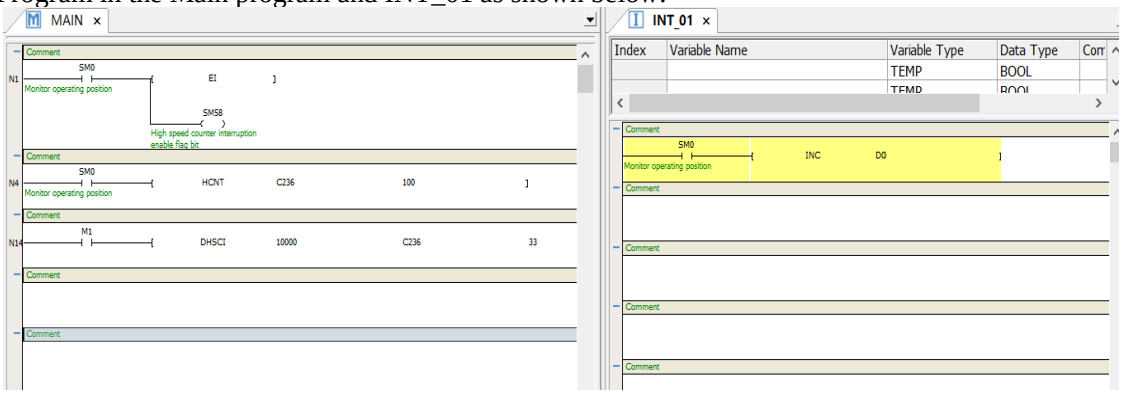
1. Right click the "Program Block" in the Project Manager area, and select "Insert Interrupt Subprogram", as shown below.



2. Right-click the newly created INT_01 subprogram and select "Property", and then set its property to "High-speed counter interrupt 0", as shown below.



3. Program in the Main program and INT_01 as shown below:



4. Compile, download, and run the program. When M1 is ON, an interrupt is triggered once the current value of high-speed counter C236 reaches 10,000. At this point, register D0 in the interrupt program is set to 1.

4.15.5 Pulse Output Complete Interrupt

(1) Description

i. For SH100 series: When enable flags SM50, SM51, SM52 (corresponding to Y0~Y2, respectively) are ON, the Pulse Output Complete Interrupt can be triggered with positioning instructions PLSY, PLSR, DRVA, DRVI. Users can perform related operations within the interrupt program.

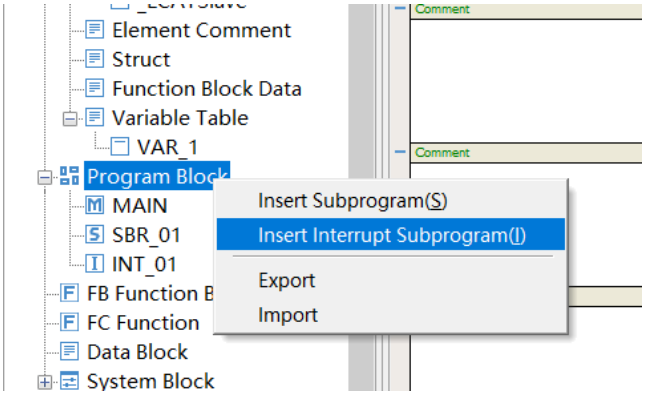
ii. For SH300/SH500 series: When enable flags SM50, SM51, SM52, SM53, SM54, SM56, SM57 (corresponding to Y0~Y7, respectively) are ON, the Pulse Output Complete Interrupt can be triggered with positioning instructions PLSY, PLSR, DRVA, DRVI. Users can perform related operations within the interrupt subroutine.

(2) Interrupt Enable Mapping

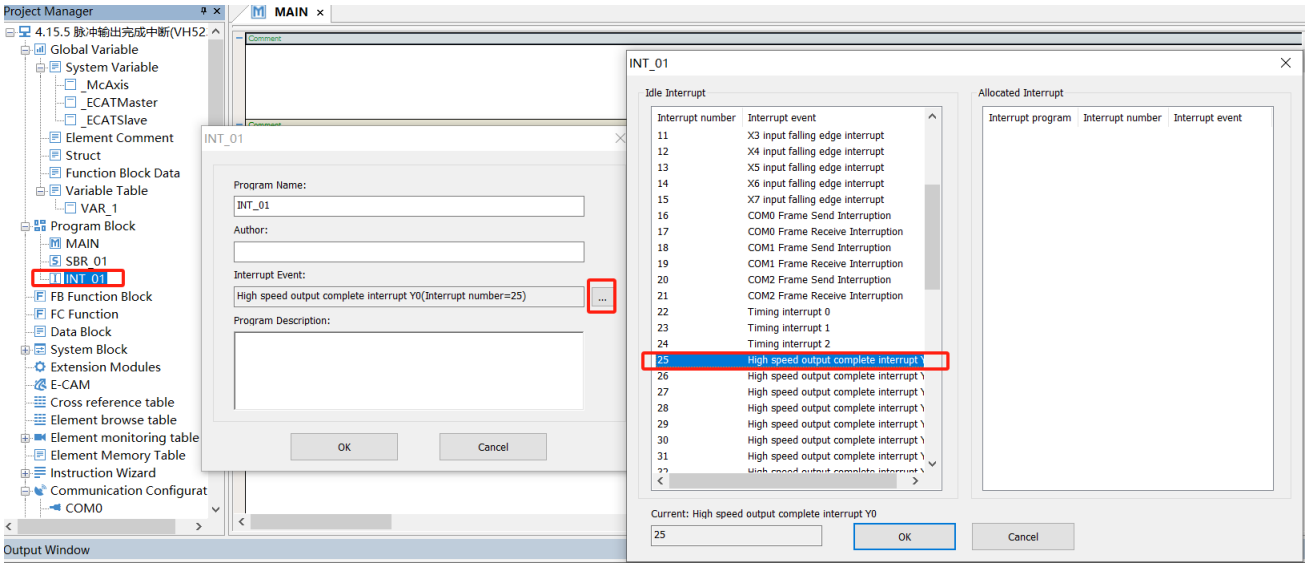
Port	Event No.	Interrupt Event	Interrupt Enable SM
Y0	25	High-speed output complete interrupt 0	SM50
Y1	26	High-speed output complete interrupt 1	SM51
Y2	27	High-speed output complete interrupt 2	SM52
Y3	28	High-speed output complete interrupt 3	SM53
Y4	29	High-speed output complete interrupt 4	SM54
Y5	30	High-speed output complete interrupt 5	SM55
Y6	31	High-speed output complete interrupt 6	SM56
Y7	32	High-speed output complete interrupt 7	SM57

Example

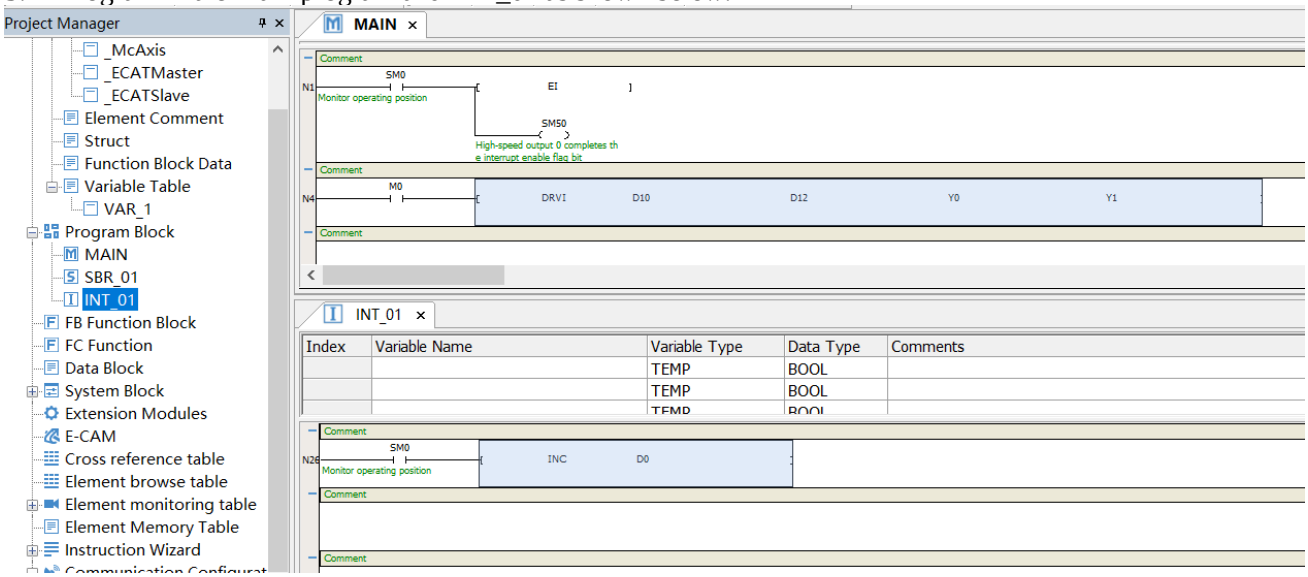
1. Right click the "Program Block" in the Project Manager area, and select "Insert Interrupt Subprogram", as shown below.



2. Right-click the newly created INT_01 subprogram and select "Property", and then set its property to "High-speed counter interrupt 0", as shown below.



3. Program in the Main program and INT_01 as shown below:



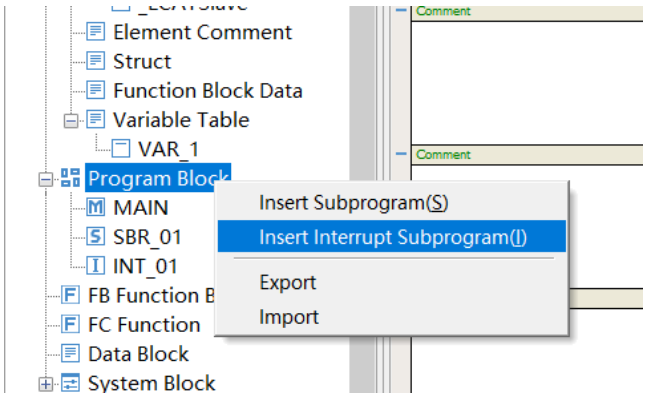
4. Compile, download, and run the program. Manually set registers D10=1000 and D12=1000. When M0 is ON, an interrupt is triggered upon completion of the pulse instruction output. At this point, register D0 in the interrupt program is set to 1.

4.15.6 Axis High-speed Counter Comparison Interrupt

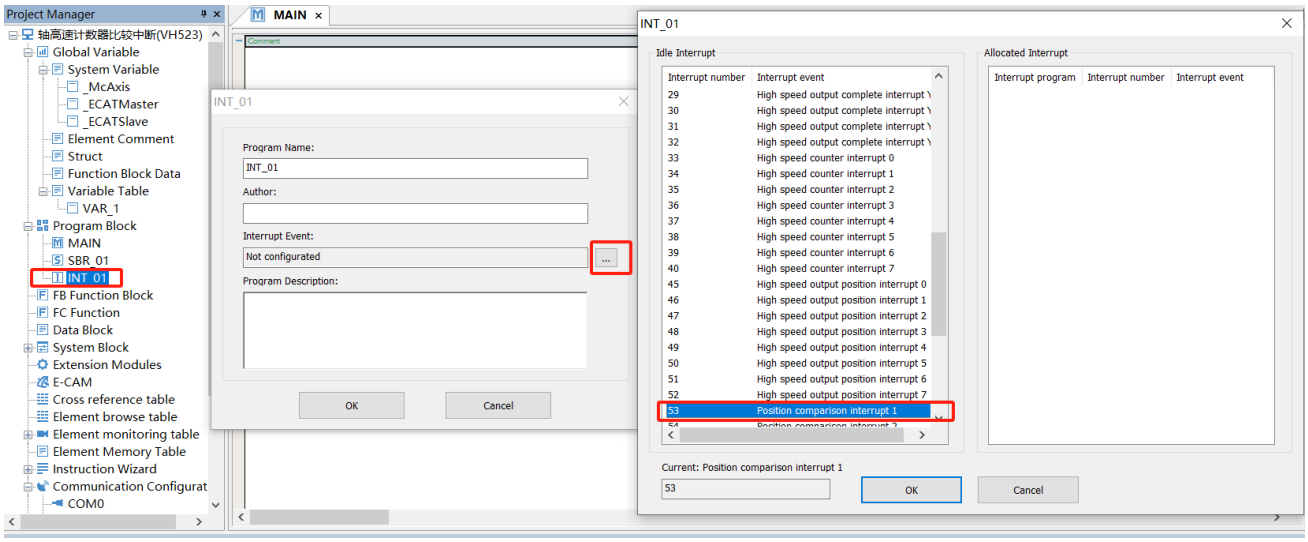
When using a local counter axis, the axis's high-speed counter comparison instruction can be employed. Enable the corresponding interrupt by setting the enable flag SM71, with total support for 16 sets of comparison interrupt instructions.

Example

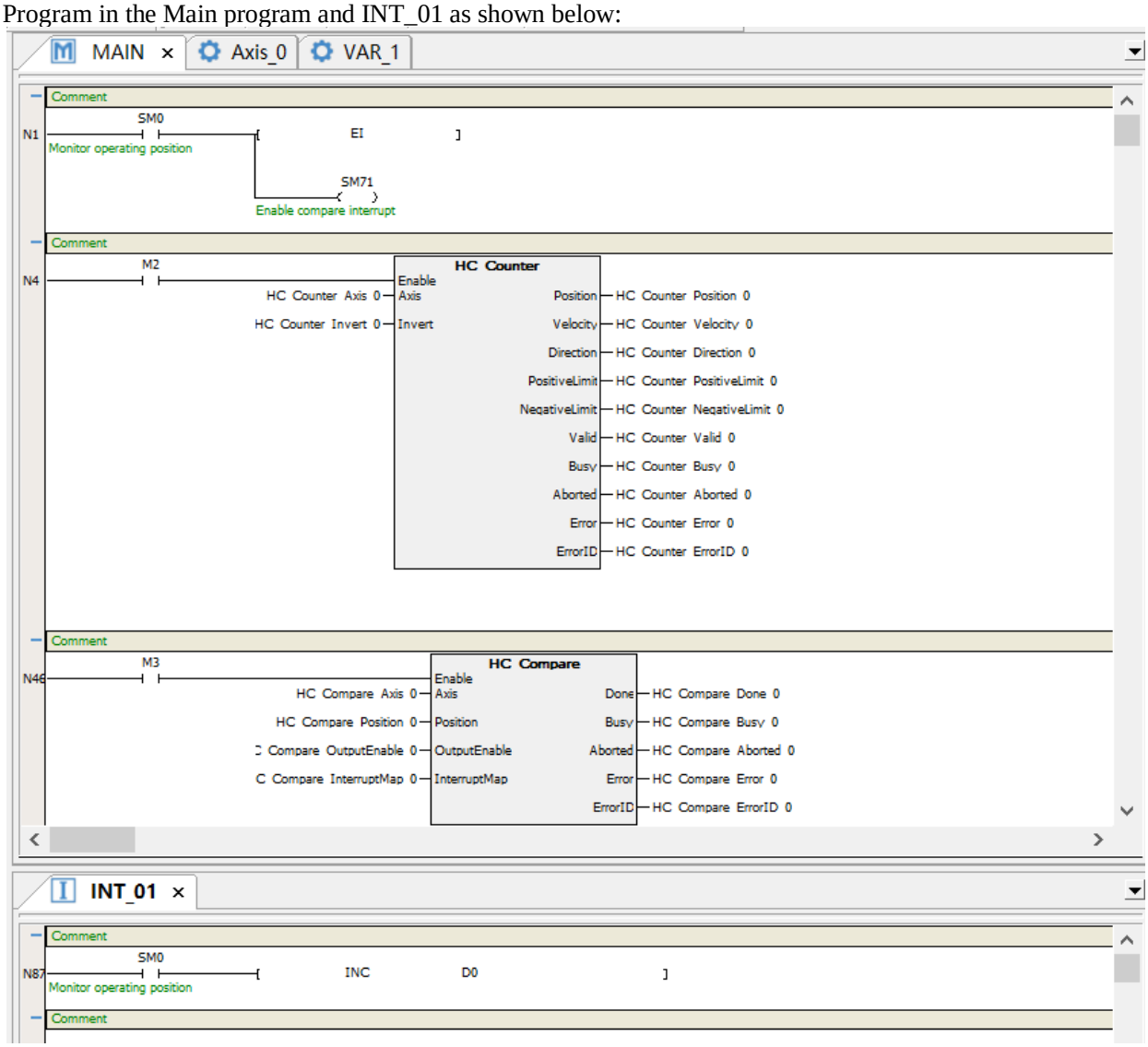
1. Right click the "Program Block" in the Project Manager area, and select "Insert Interrupt Subprogram", as shown below.



2. Right-click the newly created INT_01 subprogram and select “Property”, and then set its property to “Position comparison interrupt 1”, as shown below.



3. Program in the Main program and INT_01 as shown below:



4. Compile, download, and run the program. Manually set HC_Compare_Position_0=30 and HC_Compare_InterruptMap_0=53. When M2 and M3 are turned ON, an interrupt signal is triggered once the counter value reaches 30, and the register D0 is set to 1.

4.15.7 Serial Port Interrupt

(I) Description

In Freeport mode, the system generates interrupt events based on serial port send/receive operations.

(II) Application

Each serial port is allocated 2 interrupt resources. Serial port interrupts are used for scenarios requiring special processing or real-time handling of frame send/receive operations, ensuring immediate response independent of the scan cycle.

(III) Precautions

Enable/disable serial port interrupts by setting corresponding SM elements ON/OFF. Disabling interrupts does not cancel queued interrupt tasks. Avoid calling the XMT instruction under continuously enabled conditions within a send interrupt service subprogram, as this may cause nested interrupts and block user program execution.

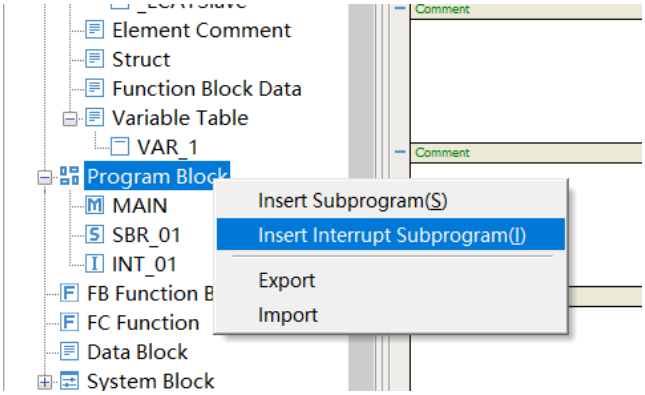
(IV) Frame Send/Receive Interrupt. Refers to interrupt events triggered upon completion of XMT (send) or RCV (receive) instructions.

Serial Port Interrupt Resource Table

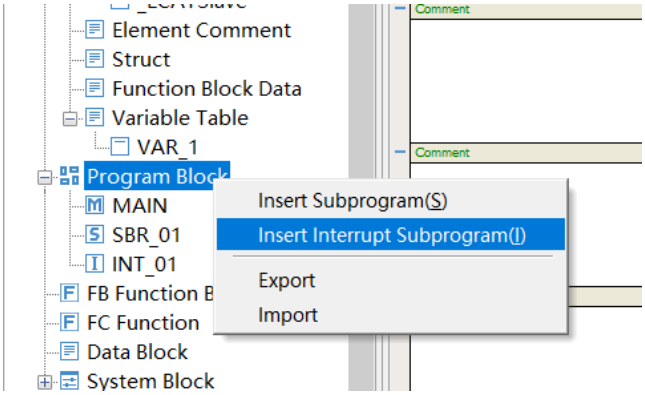
Event No.	Interrupt Event	Interrupt Enable SM
16	COM0 Frame Send Interruption	SM41
17	COM0 Frame Receive Interruption	SM42
18	COM1 Frame Send Interruption	SM43
19	COM1 Frame Receive Interruption	SM44
20	COM2 Frame Send Interruption	SM45
21	COM2 Frame Receive Interruption	SM46

Example

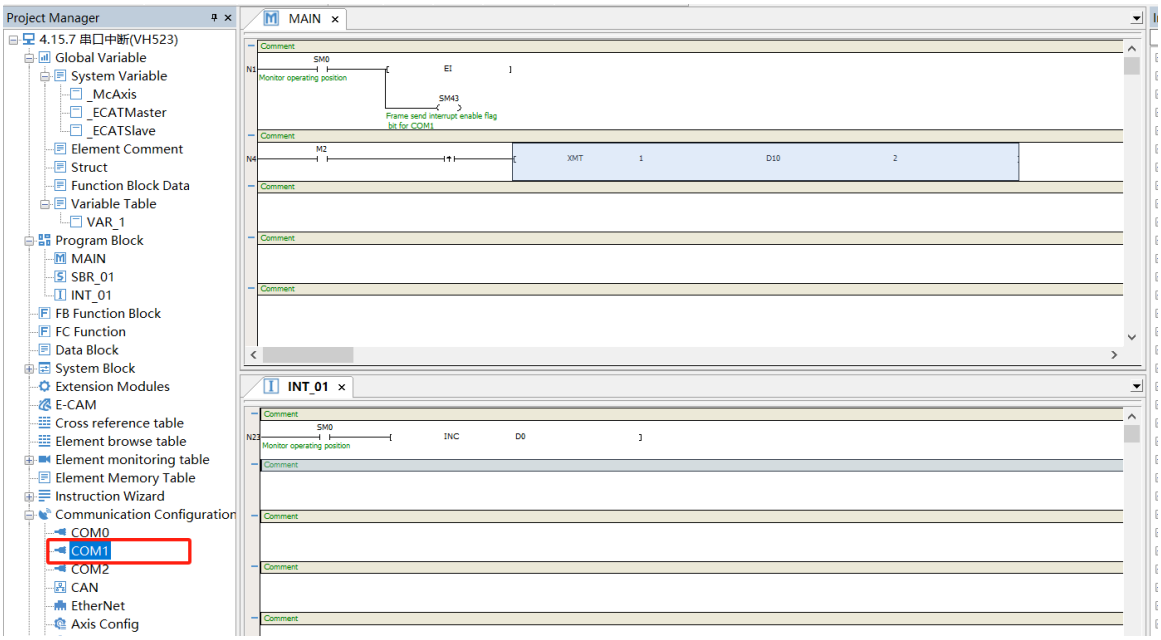
1. Right click the "Program Block" in the Project Manager area, and select “Insert Interrupt Subprogram”, as shown below.



2. Right-click the newly created INT_01 subprogram and select “Property”, and then set its property to “Position comparison interrupt 1”, as shown below.



3. Program in the Main program and INT_01 as shown below:



4. Compile, download, and run the program. Assign D10=10. When M2 is activated, an interrupt will be triggered, setting D0=1 in the interrupt subprogram.

4.16 Functions and Function Blocks (FB&FC)

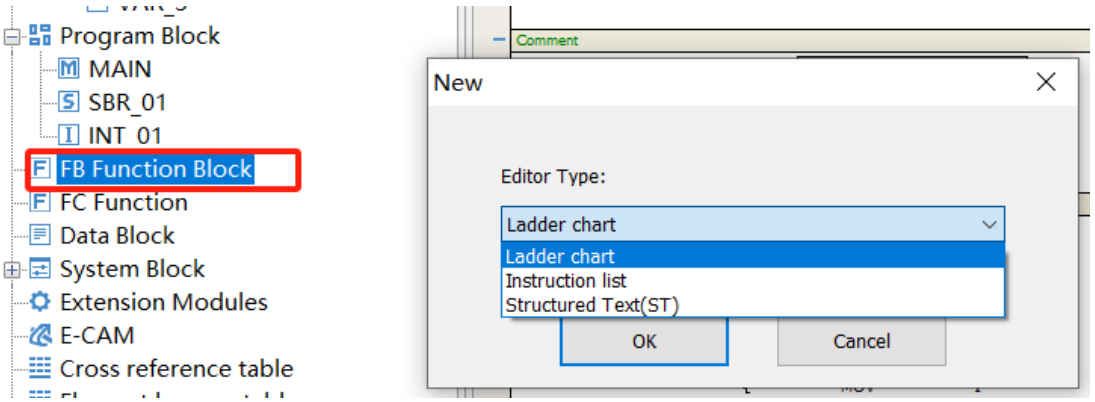
4.16.1 Function Block (FB)

A Function Block (FB) abstracts and encapsulates reusable program segments into a generic block that can be repeatedly called. Utilizing FB improves development efficiency, reduces programming errors, and enhances program quality. During execution, an FB generates one or more output values and retains unique internal variables. The controller's runtime system allocates memory for these variables, which define the state characteristics. For identical input values, varying internal states may yield different computational results.

The basic workflow for using FB: Create → Program → Instantiate → Execute→ Encapsulate → Import.

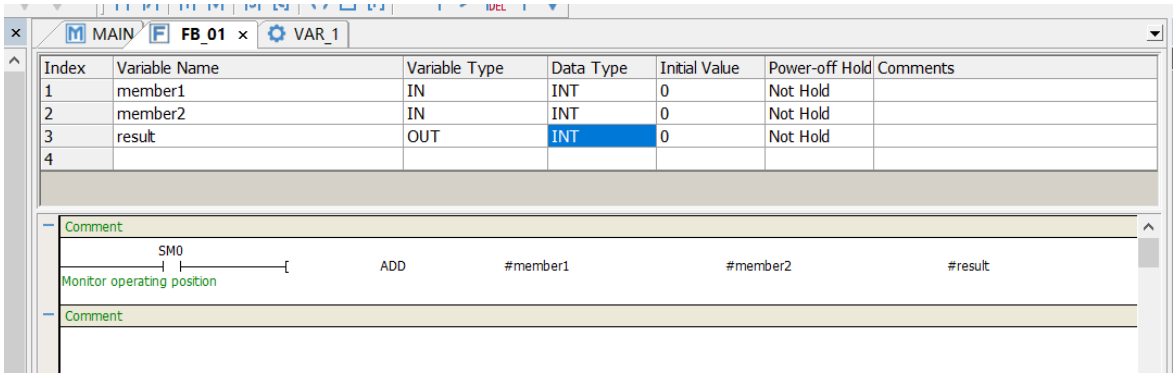
Create a FB

1. In the Project Manager area, right-click "FB Function Block" and select New. In the pop-up dialog box, configure the FB editor and click OK to complete creation (FB_01). Right-click "FB_01" to rename, encrypt, import/export, or delete the FB.



Program a FB

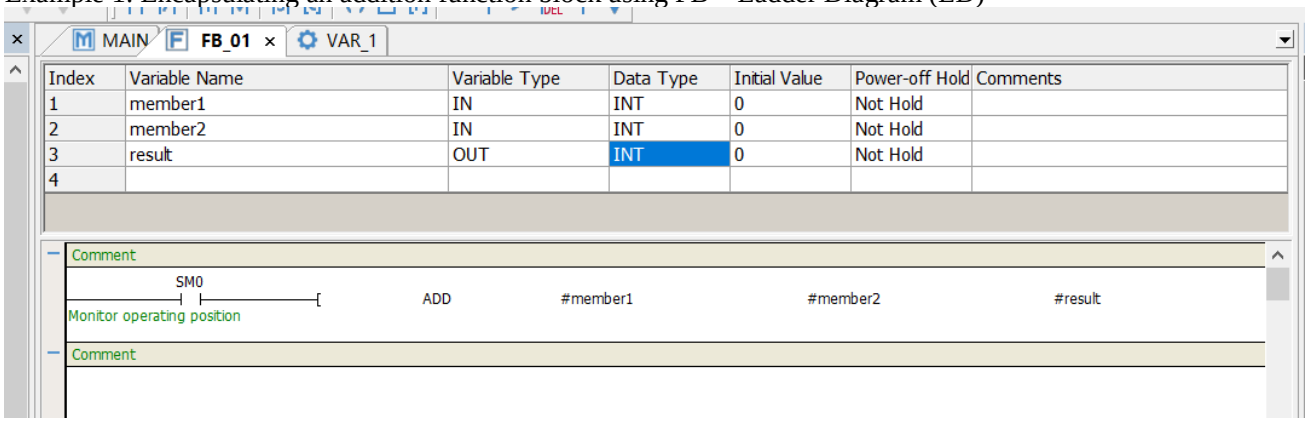
Double-click "FB_01" to enter the editing interface. Define variable properties such as Variable Name, Variable Type, Data Type, Initial Value, and Power-off Hold before use, as shown below.



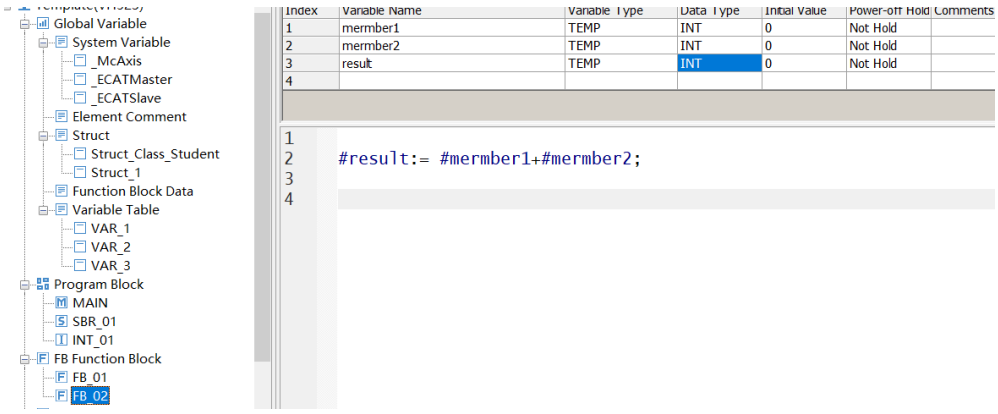
Description of FB properties

Property	Description
Variable Name	User-defined variable identifier
Variable Type	<table><tr><th>Variable Type</th><th>Type</th><th>Description</th></tr><tr><td>INT</td><td>Input variable</td><td>Parameter passed into the FB</td></tr><tr><td>OUT</td><td>Output variable</td><td>Parameter returned from the FB</td></tr><tr><td>INTOUT</td><td>Input/Output variable</td><td>Parameter passed into and out of the FB</td></tr><tr><td>TEMP</td><td>Local variable</td><td>Accessible only within the FB</td></tr></table>
Data Type	Support BOOL, INT, DINT, REAL, STRING, ARRAY, struct, pointers, etc.
Initial Value	Assign a default value to the variable.
Power-off Hold	Hold: If "Initialize Variables Hold" is selected during download, variables reset to initial values; otherwise, retain last runtime values. Not Hold: Variables reset to initial values on power-up.
Comments	Add comments to variables or soft elements.
Note	1. Within an FB program, functions (FC) or function blocks (FB) can be called, supporting up to 8 levels of nested calls. 2. Supported soft elements (e.g., D0) can be used as global variables in addition to FB variables.

Example 1: Encapsulating an addition function block using FB – Ladder Diagram (LD)



Example 2: Encapsulating an addition function block using FB – Structured Text (ST)

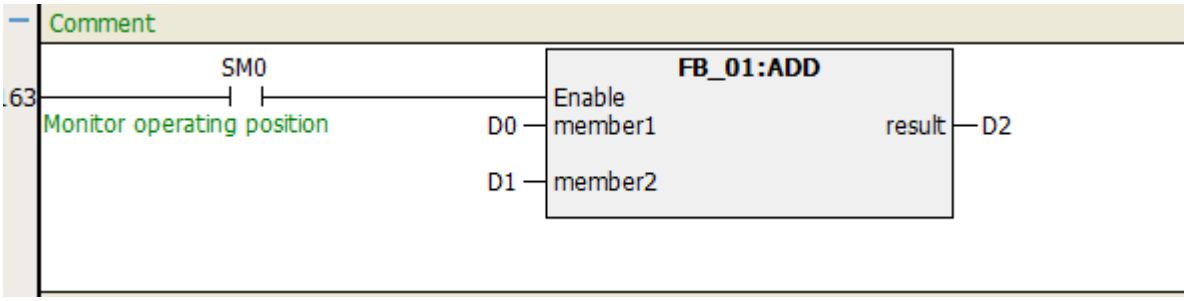


Instantiate a FB

After programming an FB, instantiate it in ladder diagram (LD) or structured text (ST) applications.

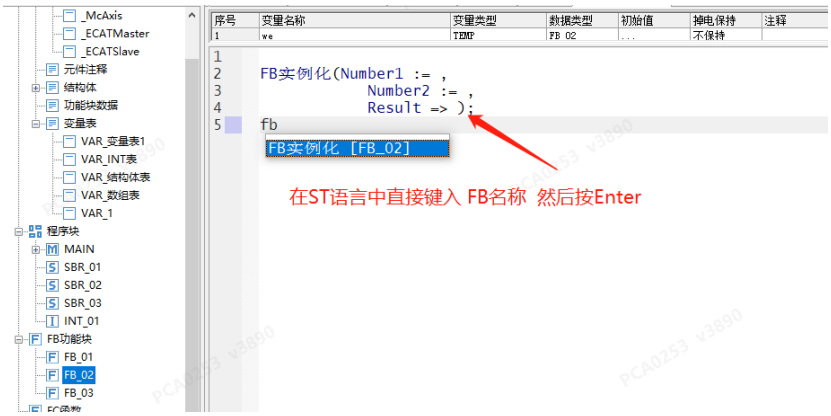
Ladder Diagram (LD)

● Method 1: In the LD application, select FB_02 and drag it to the editing area. Fill in parameter values in the dialog box to complete instantiation, as shown below.

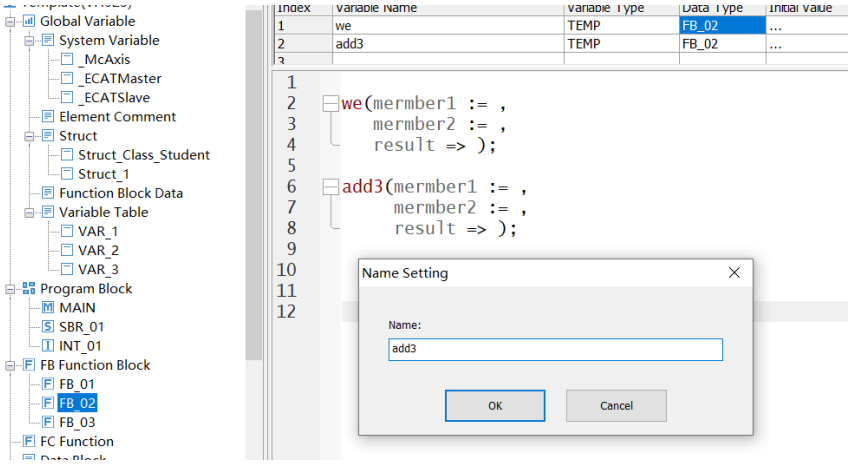


Structured Text (ST)

● Method 1: Declare the FB instance in the instance table, then directly input the FB instance name in the ST application and press Enter, as shown below.



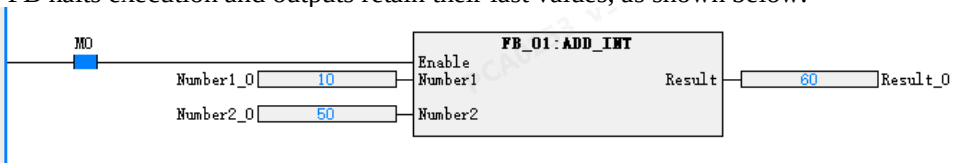
● Method 2: In ST programming, select the FB and drag it to the programming area. Enter the FB name in the pop-up window to automatically instantiate it, as shown below.



Execute a FB

Ladder Diagram (LD)

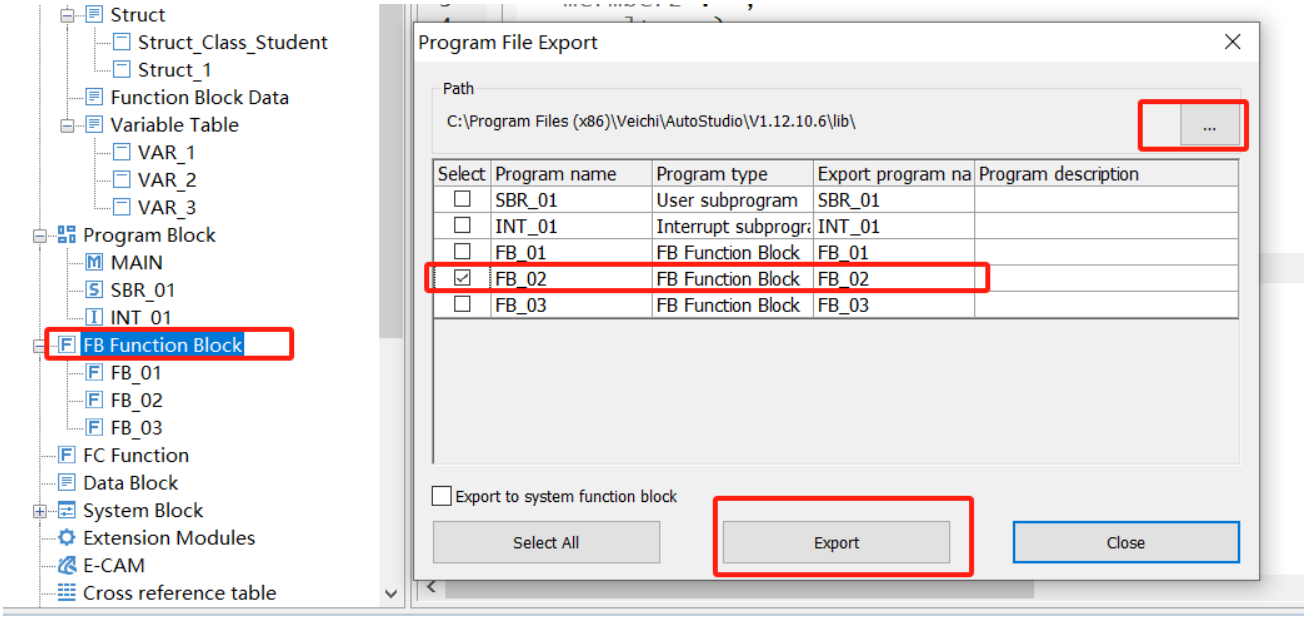
After instantiation, the FB's Enable pin is connected to the ladder logic network. When the Enable power flow is ON, the FB executes, and its outputs update based on input states and internal variables. When the Enable power flow is OFF, the FB halts execution and outputs retain their last values, as shown below.



Encapsulate a FB

A tested FB can be encapsulated into a library for reuse across projects via AutoSoft's library management.

1. Right-click the function block and select Export, as shown below.

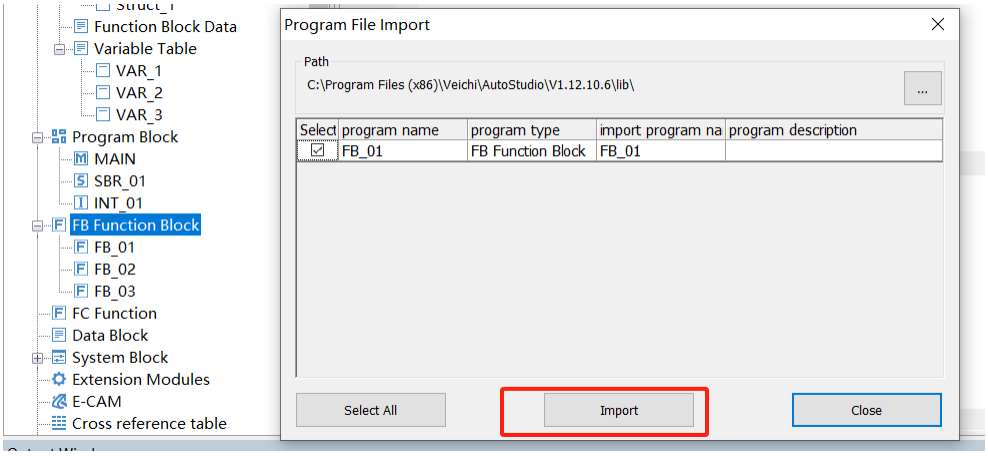


2. In the pop-up Program File Export dialog box, select the target FB, set the export path, and click Export to generate the FB as an .EXP file.

Import a FB

Exported FB libraries can be imported into other programs for reuse via the following steps:

1. Right-click the "FB Function Block" in the Project Manager area and select "Import" to add it to the project, as shown below.



2. The imported FB can be opened by double-clicking for editing or debugging. If the FB is encrypted, a password must be entered to modify it.

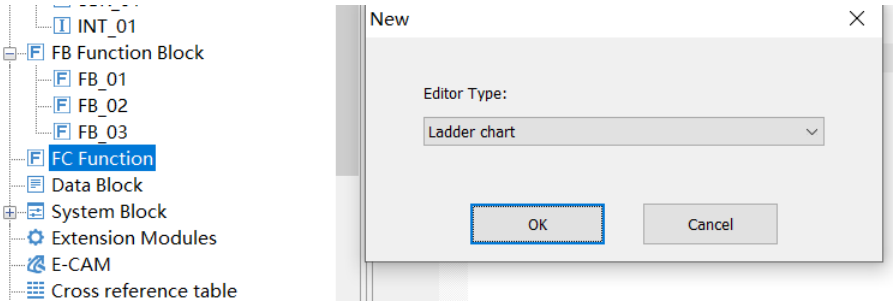
4.16.2 Function (FC)

A Function (FC) is an independently encapsulated program block that can define input/output parameters and non-static internal variables. Calling a function with identical input parameters always yields the same output results. A key characteristic of functions is their static internal variables and lack of internal state storage, distinguishing them from Function Blocks (FB). As a fundamental algorithm unit, FC is commonly used for mathematical operations (e.g., sin(x), sqrt(x)).

The basic workflow for using FC: Create → Program → Call → Execute → Encapsulate.

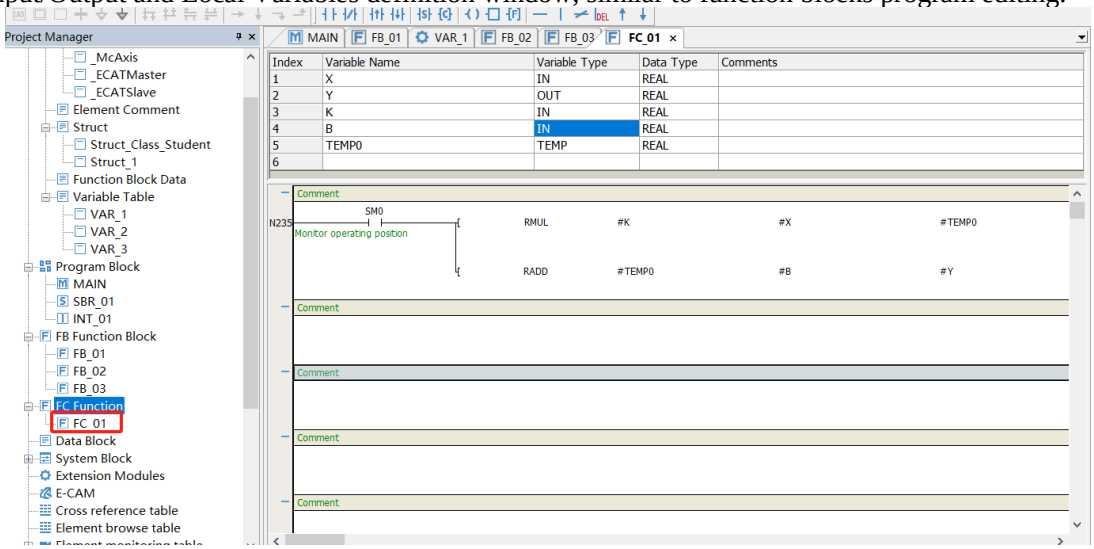
Create a Function

1. In the Project Manager area, right-click the "FC Function", select New, choose the editor in the dialog box, and click OK to create a Function.



Program the Function

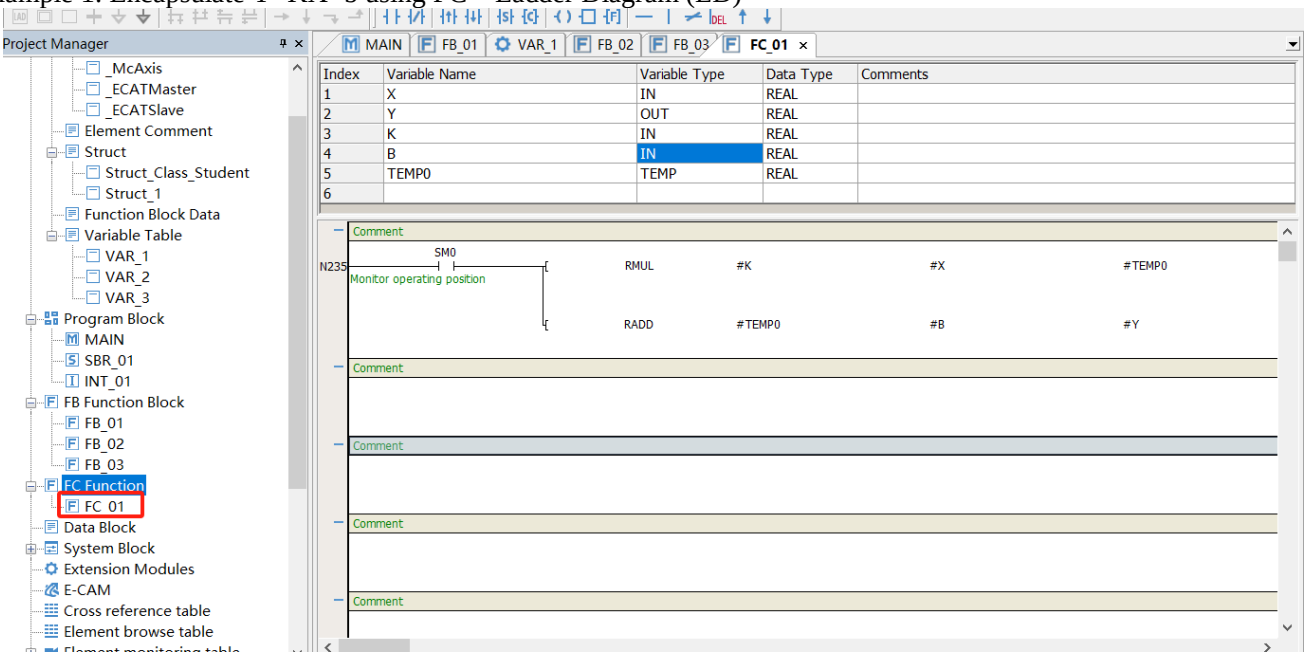
1. Double-click the newly created function to enter the editing interface. This interface includes an Input/Output and Local Variables definition window, similar to function blocks program editing.



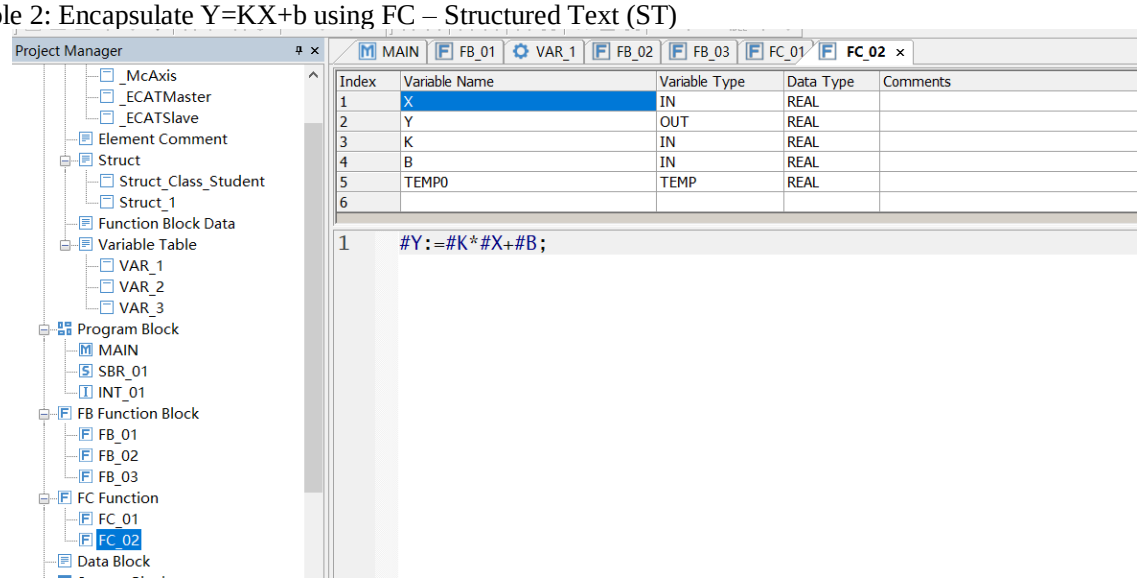
Users can define input (IN), output (OUT), input/output (INOUT), and local variables (VAR). Supported data types include BOOL, INT, DINT, REAL, STRING, arrays, and struct. For struct variables, define members in the global variable structure is required.

- Unlike function blocks, function variables cannot have initial values, and all local variables are “Not Hold”.
- Use ladder diagram (LD) to program the function. Functions (FC) can be called within the function and by other functions, function blocks, or programs.
- Avoid state-dependent or multi-cycle execution instructions (e.g., LDP, motion control commands).
- Functions support encryption, and import/export.

Example 1: Encapsulate Y=KX+b using FC – Ladder Diagram (LD)

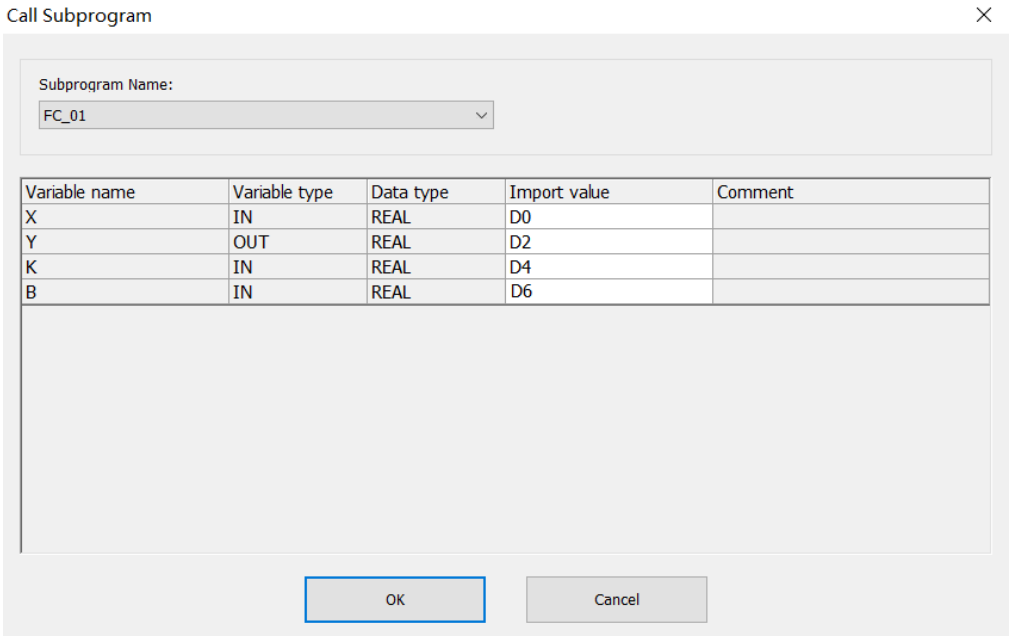


Example 2: Encapsulate Y=KX+b using FC – Structured Text (ST)



Call a Function

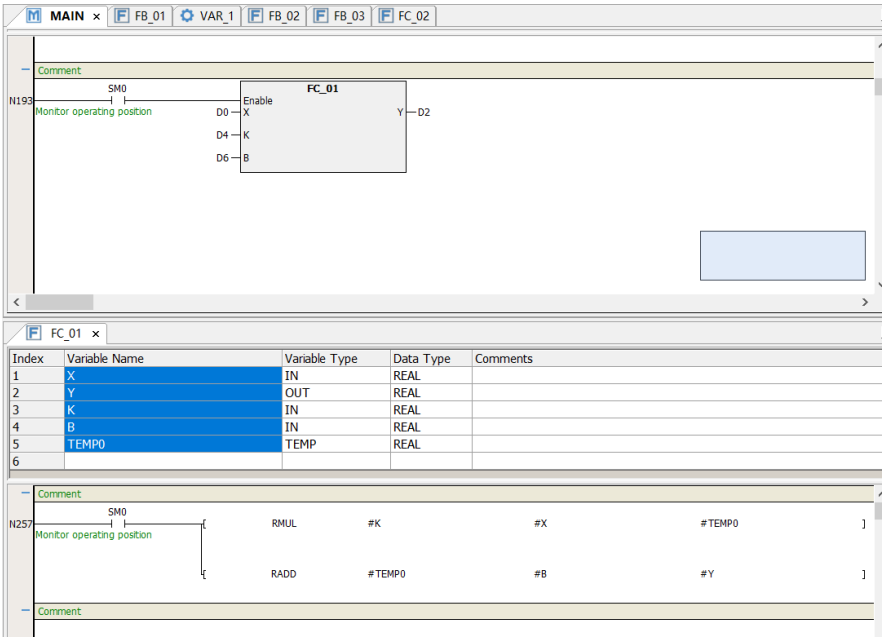
After programming the FC, drag it to ladder diagram (LD) or structured text (ST) applications to call it, assign addresses or variables in the “Input Value”, and click OK.



Execute a Function

Ladder Diagram (LD)

Once called, the function's Enable pin connects to the ladder logic network. When the Enable power flow is ON, the function executes, updating outputs based on input states. When the Enable power flow is OFF, the function halts execution, and outputs retain their last values.



Set Initial Values for FB

Initial values for FB can be modified via either the FB type or FB instance.

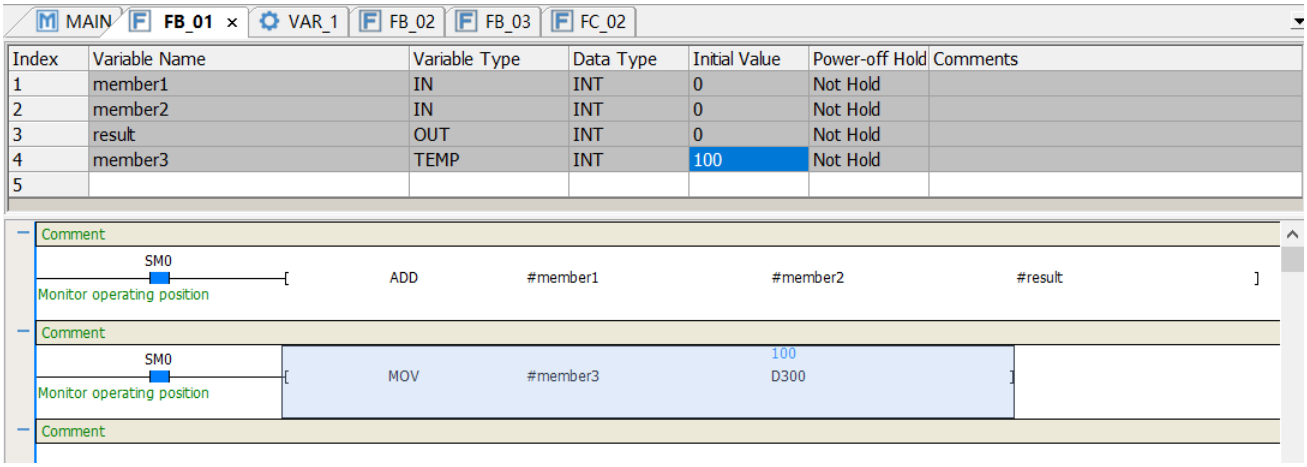
- Modifying the initial value through the FB type updates the type's default value.
- Modifying the initial value through the FB instance updates the instance-specific value.
- If an instance's initial value is modified, its member variables display the updated value with a yellow cell background.

If unmodified, instance member variables show the default value with a white cell background.

The FB type's initial value serves as the default for instances. Reset the instance's modified value to the default.

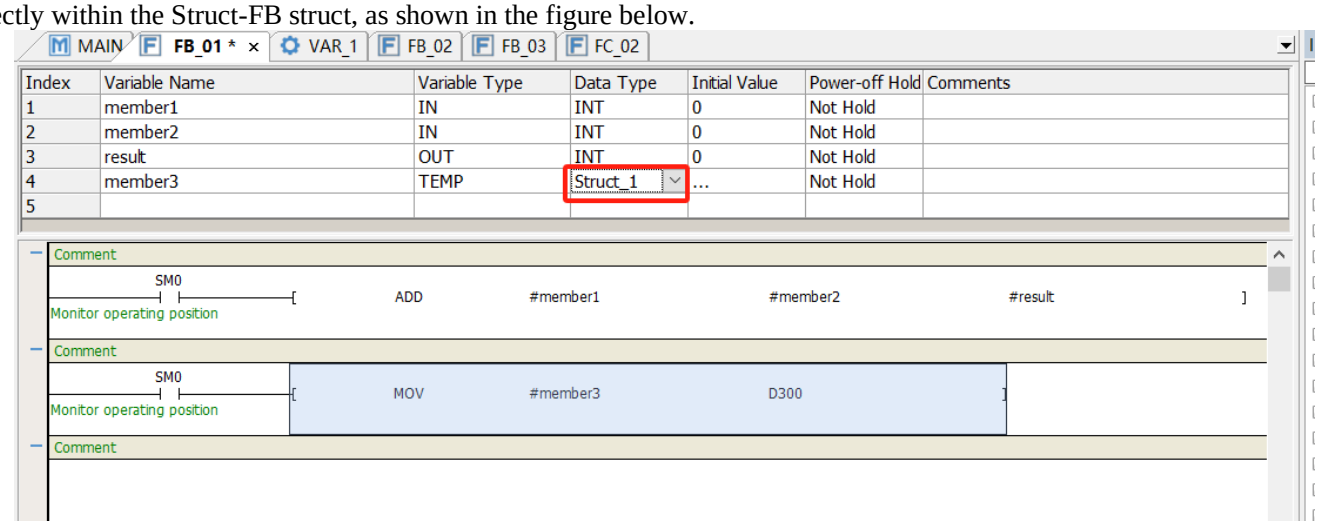
Modifying Initial Values in Non-Nested FBs

When modifying the FB type's initial value (e.g., from 0 to 10), the FB instance adopts the default value (FB type's initial value), as shown below.



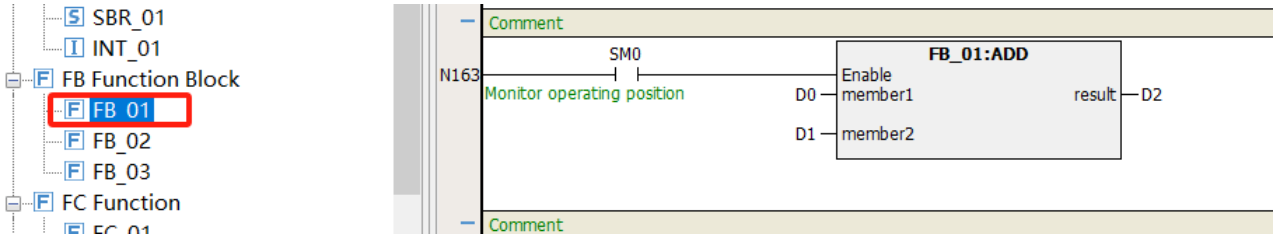
Modifying Initial Values in Nested FBs

If the variable Number3 (data type: Struct-FB1) references a nested struct Struct-FB, assign initial values directly within the Struct-FB struct, as shown in the figure below.



FB View Label

The displayed label consists of three components from left to right: Node Name, Instance Name, and Unsaved Indicator. Node Name refers to the name of the project tree node; Instance Name indicates the instance name within square brackets.

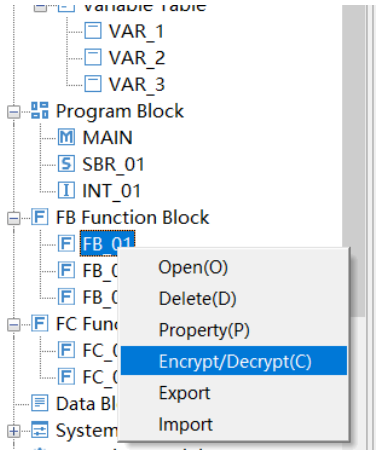


As shown above, FB is the Node Name, FB_01:ADD_INT is the Instance Name, and * denotes unsaved changes. To ensure proper label parsing, the characters ., *, (,) are prohibited when renaming FB or struct.

4.16.3 Encrypt Function Blocks/Functions

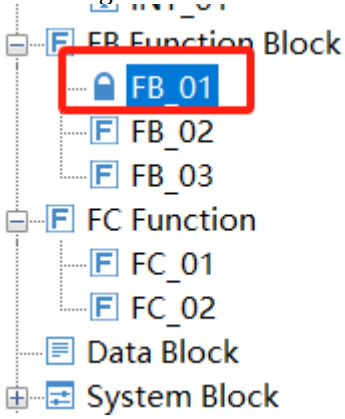
Here using Function Blocks (FB) as an example. (The encryption procedure for Functions (FC) is identical.) Encrypted FB/FC retain the same invocation method as standard FB/FC.

1. In the Project Manager area, unfold FB Function Block, right-click the target FB, and select Encrypt/Decrypt.



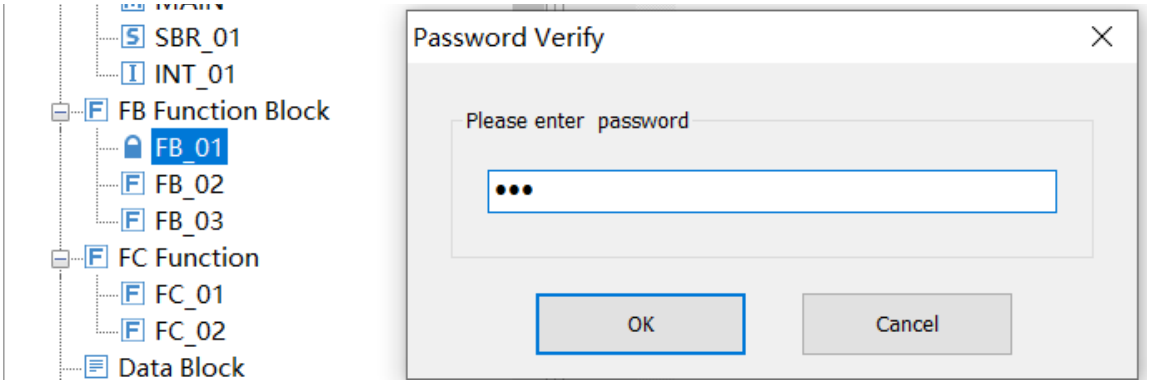
2. In the “Program Encrypt” dialog box, enter and confirm the password.

3. The encrypted FB is displayed as the figure below.



4. Repeat the same steps to decrypt the FB, restoring it to an unencrypted state.

To access an encrypted FB: Double-click the encrypted node, or right-click the encrypted node and select Password Verify, and enter the correct password in the pop-up dialog box.



5 Programming Language

5.1 Structured Text (ST)

5.1.1Overview

ST (Structured Text) is one of the five standard languages specified in IEC 61131-3, featuring a syntax similar to PASCAL. It excels in conciseness, efficiency, and portability, enabling developers to write PLC programs using structured code akin to high-level programming languages. Compared to traditional Ladder Diagram (LD), ST demonstrates superior capabilities:

- Conciseness: ST features a clear code structure, enhancing readability and maintainability.
- Efficiency: Leveraging high-level language features (e.g., loops, conditional statements) significantly boosts programming productivity.
- Portability: ST code is executable across various PLC platforms, ensuring strong portability.

For example:
IF bool_expression_1 THEN
<logic_statement>
ELSEIF bool_expression_2 THEN//Optional lines
<logic_statement>
ELSE//Optional lines
<logic_statement>
ENDIF;

Note:

- Supported by SH300/SH500 series. For VC5 series, firmware v2.29+ and AutoSoft 12.10.3+ are required.
- Maximum 1,000 lines per ST file.

5.1.2 Basic Rules of ST Language

(1) Variable declaration before use

Variables in ST require declaration before use. PLC programming tools prompt undeclared variables, with some even automatically creating them. Declared variables generally need only data type and attributes, requiring no initialization unless specified.

(2) English input

ST programming mandates English input mode for compilation success. While Chinese variables are supported, their usage demands careful input method management: either English mode (default half-width/punctuation) or Chinese input with half-width characters and English punctuation.

(3) Comments

Comments are ignored during compilation and support any text/symbols as long as the PLC permits. Comments are categorized into single-line comment and multi-line comment, which is enclosed within (**) delimiters, with an example shown in figure below.

```
1   #Y:=#K*#X+#B;
2
3   /*1234567890     */     //multiline comment
4
5   ////AAAAA     Single line comment
6
7
8
9
10
11
```

5.1.3 ST Expressions

Expressions are fundamental elements in ST, composed of operators and operands. Operands may be constants, variables, function calls, or nested expressions and operators define computations between variables.

Example:

- Constants: 10, 10.5, 16#10
- Variables: dVar, D0:R
- Function calls: Fun1(2,8,9)
- Compound expressions:
  10+3, var1 OR var2, (x+y)/z, iVar1:=iVar2+22

Expressions are evaluated based on operator precedence. Operators with the highest precedence are evaluated first, followed by those with lower precedence in descending order. Operators of equal precedence are evaluated left to right as written.

Examples

Given INT variables A=1, B=2, C=3, D=4:

A+B-C*ABS(D) evaluates to -9

(A+B-C)*ABS(D) evaluates to 0

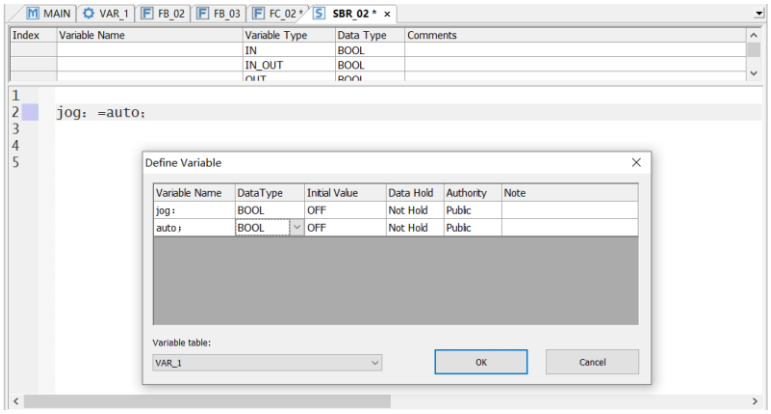
For operators with two operands, left operand is evaluated first. Example: For SIN(X)*COS(Y), SIN(X) is evaluated first, followed by COS(Y), and finally their multiplication.

Table 4-1 ST Language Operators

Operation	Symbol	Example	Priority
Parentheses	(Expression)	(A+B/C), (A+B)/C, A/(B+C)	9 (Highest)
Function call	Function name (parameters separated by commas)	LN(A), MAX(X,Y)	8
Negation	-	-A	7
Unary plus	+	+B	7
Bitwise NOT	NOT	NOTC	7
Multiply	*	A*B	6
Divide	/	A/B	6
Modulus	MOD	AMODB	6
Add	+	A+B	5
Subtract	-	A-B	5
Comparison	<,>,<=,>=	A<B	4
Equality	=	A=B	4
Inequality	<>	A<>B	4
Logical AND	AND	AANDB	3
Logical XOR	XOR	AXORB	2
Logical OR	OR	AORB	1 (Lowest)

5.1.4 Variables

ST programming requires pre-declared variables. Define variables in the variable table and reference them in ST, or directly write variables in the ST editor; pressing ENTER or clicking outside the program block to auto-trigger batch declaration dialog box. The default variable type is INT, Data types are automatically recognized during function/instruction calls.



ST programming supports soft elements (e.g., D, R, W) but requires data type declarations.

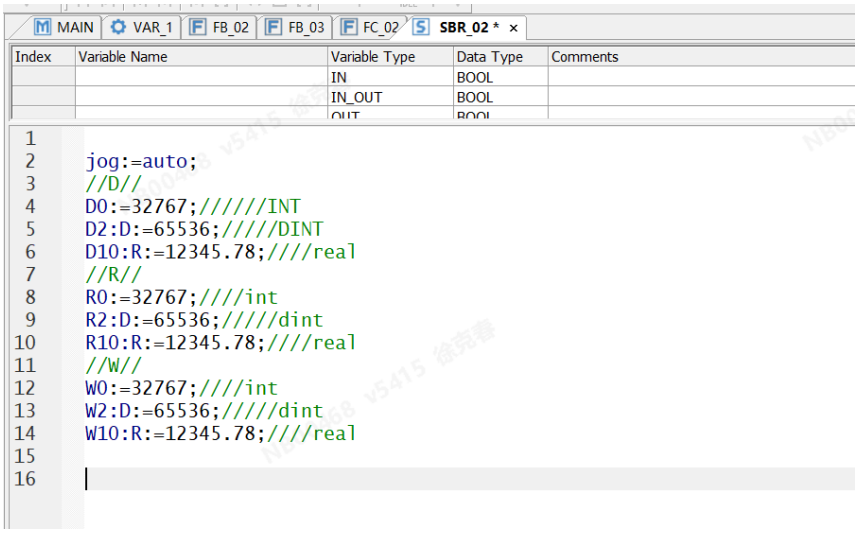
Examples:

D0: Defined as INT.

D0:D: Defined as DINT (occupies D0 and D1 registers).

D10:R: Defined as REAL (occupies D10 and D11 registers).

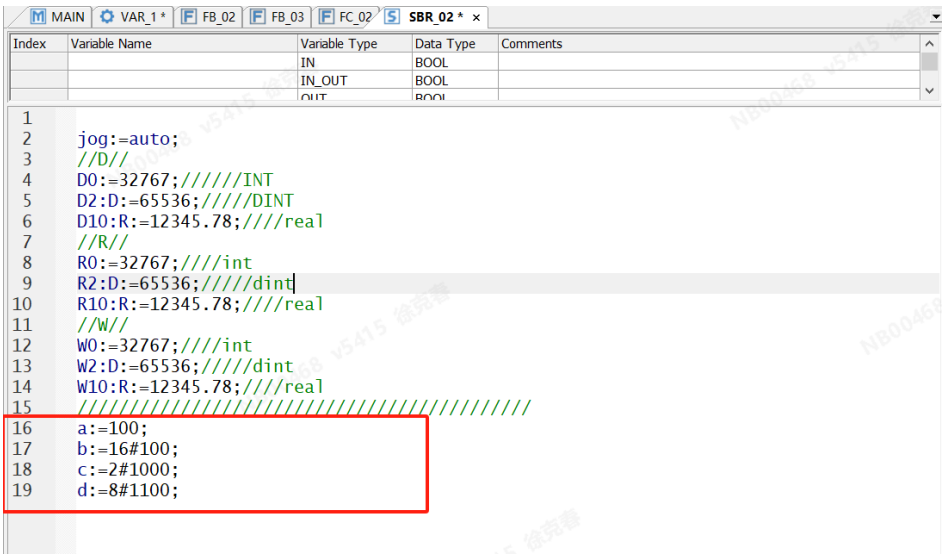
Refer to the figure below for illustration. Refer to the figure below for illustration.



5.1.5 Constants

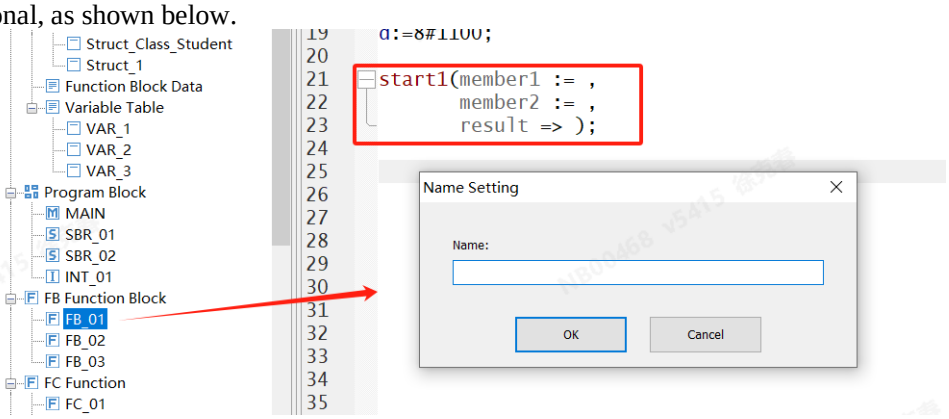
Constants can be defined in multiple formats:

1. Default decimal: E.g., a:= 100;
2. Ladder Diagram (LD)-style constants: 100 (decimal), 16# for hexadecimal, 8# for octal, floating-point numbers (e.g., 12.234).



5.1.6 FB/FC/SBR/Interrupt Invocation

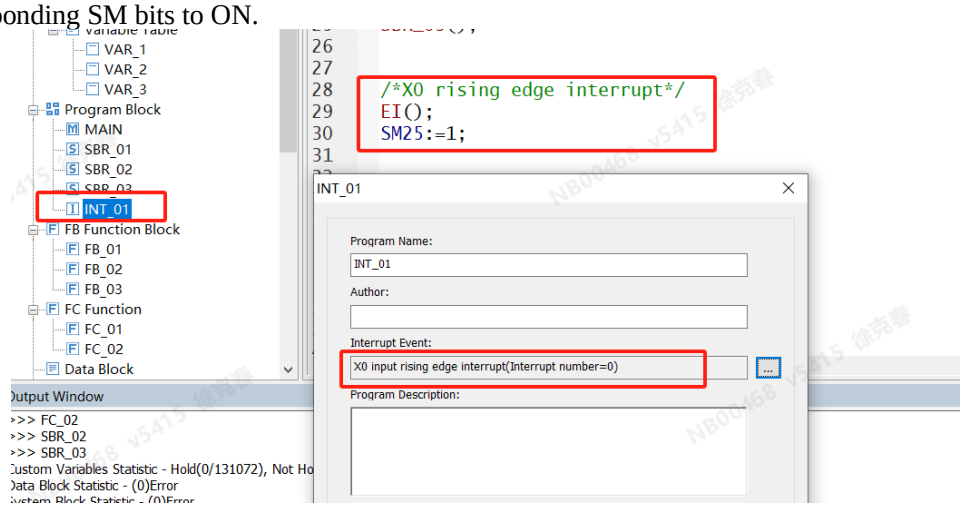
(1) FB Block in ST: Input parameters (except axis parameters, which are mandatory) and output parameters are optional, as shown below.



(2) FC Block in ST: All input parameters are mandatory; output parameters can be omitted. Missing inputs will cause compilation errors.

(3) Subprogram in ST: Called as a parameterless function, e.g., SBR_03();

(4) Interrupt in ST: No manual invocation required. Enable interrupts by calling EI(); and setting the corresponding SM bits to ON.



Note:

- FB/FC nested calls must not exceed 8 levels.
- Subprogram (SBR) nesting must not exceed 6 levels.

5.2 Syntax Instruction

An ST program is composed of instructions, which are separated by semicolons ";".

Table 4-3 ST Syntax Instruction List

Instruction	Function	Example
:=	Assignment	A:=B
FB call	Function block call and output	TONR (In:=b0, PT:=dVar, R:=b0, Q=>, ET=>);

IF	Selection	IF A>0 THEN
		X:=10;
		ELSE
		X:=0;
		END_IF;

CASE	Multi-branch selection	CASEAOF 1:X:=1; 2:X:=2; 3:X:=3; ELSEX:=0; END_CASE;
WHILE	While loop	A:=0; WHILE A<=1000DO A:=A+7; END_WHILE;
		A:=1; TOTAL:=0; REPEAT

REPEAT	Repeat loop	TOTAL:=TOTAL+A; A:=A+1; UNTILA>10 END_REPEAT;
FOR	For loop	FORi:=0TO100DO X[i]:=0; END_FOR;
EXIT	Exit loop	EXIT;
CONTINUE	Break current loop	CONTINUE;
RETURN	Return	RETURN;
(*text*)	Comment	(* Comment out multiple lines IFA=3THENA:=5; END_IF; *)
// text	Single-line comment	//A:=5;
;	Empty statement	;

5.2.1 Assignment Statement

The assignment statement is one of the most frequently used constructs in Structured Text (ST). It assigns the value of the right-hand expression to the left-hand operand (variable or address) using the operator :=.

Syntax:

<Variable> := <Expression>;

Example:

A := B * 5;

After execution, the value of A becomes five times the value of B.

5.2.2 Function Block Invocation

Syntax:

FB_Instance Name(FB_Input := Value, FB_Output => Value, ...);

Example:

Call an instance of an addition function block (ADD_Function), assign input parameters D0 and D1, map the output to D2, and assign the result to i_Sum:

ADD_Function (Number1:=D0,
Number2:=D2,
Result=>D10);
i_Sum:=ADD_Function.Result;

Note:

ADD_Function is the function block instance.

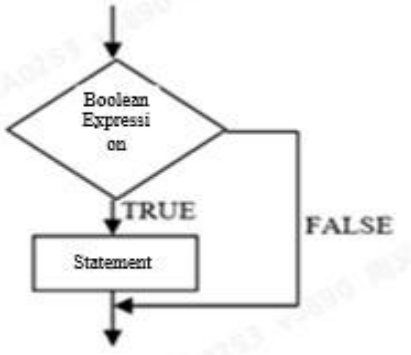
5.2.3 IF Statement

The IF statement implements a single-branch conditional structure.

Syntax:

IF <Boolean_Expression> THEN
<Statements>;
END_IF;

The <Statements> execute only if the <Boolean_Expression> evaluates to TRUE. <Statements> may contain a single statement, multiple statements, or be empty.



Example: Check if the current temperature exceeds 60°C. If true, turn on the fan:

```
// Variable declaration
nTemp: INT;  (* Current temperature signal *)
bFan: BOOL;  (* Fan ON/OFF signal *)
// Program
nTemp:=80;
IF nTemp>60 THEN
bFan:=TRUE;
END_IF
```

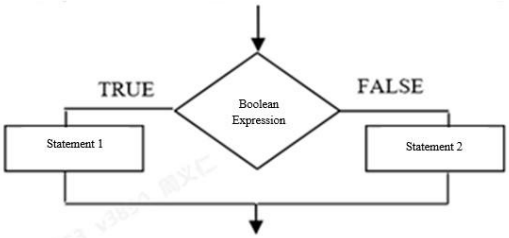
5.2.4 IF-ELSE Statement

The IF-ELSE statement implements a dual-branch conditional structure.

Syntax:

```
IF <Boolean_Expression> THEN
<Statements 1>;
ELSE
<Statements 2>;
END_IF
```

If <Boolean_Expression> is TRUE, <Statements1> executes; otherwise, <Statements2> executes.



Example: Check if temperature is below 20°C to enable heating; otherwise, disable it:

```
// Variable declaration
nTemp: INT;  (* Current temperature signal *)
bHeating: BOOL;  (* Heater ON/OFF signal *)
// Program
IF nTemp<20 THEN
bHeating:=TRUE;
ELSE
bHeating:=FALSE;
END_IF
```

Nested IF-ELSE (Multi-Branch):

```
IF <Boolean_Expression1> THEN
IF <Boolean_Expression 2> THEN
<Statements 1>;
ELSE
<Statements 2>;
END_IF
ELSE
<Statements 3>;
END_IF
```

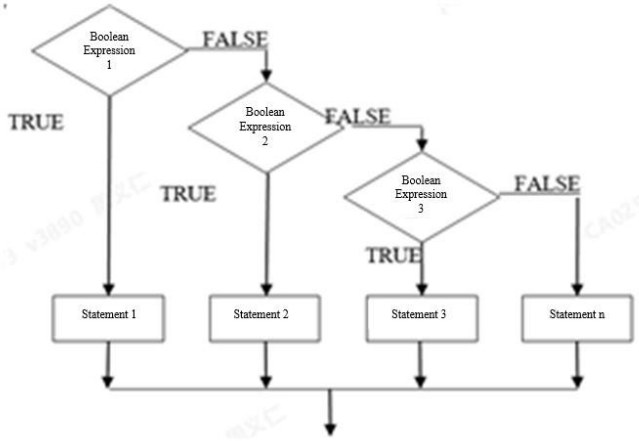
3) IF..ELSIF..ELSE (Multi-Branch):

Syntax:

```
IF <Boolean_Expression 1> THEN
<Statements 1>;
ELSIF <Boolean_Expression 2> THEN
<Statements 2>;
ELSIF <Boolean_Expression 3> THEN
<Statements 3>;
...
...
ELSE
<Statements n>;
```

END_IF

Executes <Statements1> if <Boolean_Expression1> is TRUE. If false, evaluates subsequent ELSIF conditions sequentially. Executes <Statements n> only if all conditions are FALSE.



Notes:

- An IF statement must include at least one IF, THEN, and END_IF.
- Keyword: Use ELSIF (not ELSEIF).
- Conditions are terminated by THEN.

5.2.5 CASE Statement

The CASE statement enables multi-branch selection based on the value of an expression.

Syntax:

```
CASE <Condition_Variable> OF
<Value1>: <Statements1>;
<Value2>: <Statements2>;
<Value3, Value4, Value5>: <Statements3>;
<Value6..Value10>: <Statements4>;
<Value n>: <Statements n>;
ELSE
<Else_Statements>;
END_CASE;
```

Execution Logic:

- If the value of the <Condition_Variable> equals <Value_i>, execute <Statements_i>.
- If no specified value matches the conditional variable, execute the <Else_Statements>.
- When multiple values require identical operations, list them consecutively separated by commas to execute shared instructions, as shown in line 4 of the example.
- For value ranges requiring identical operations, specify the start and end values separated by two dots (..) to execute shared instructions, as shown in line 5 of the example.

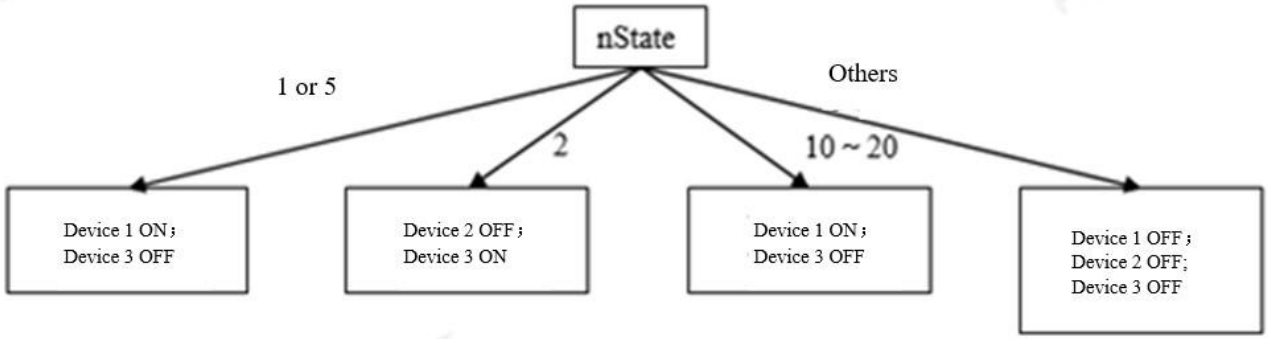
Example: When the state is 1 or 5, Device 1 operates and Device 3 stops; for state 2, Device 2 stops and Device 3 operates; for states 10 to 20 (inclusive), Devices 1 and 3 operate simultaneously. In all other cases, Devices 1, 2, and 3 stop. Implementation code is as follows:

```
// Variable declaration
nDevice1, nDevice2, nDevice3:BOOL; (*Device 1..3 switch control signals*)
nState:INT; (*Current state signal*)
// Program
CASE nStateOF
1.5:
nDevice1:=TRUE;
nDevice3:=FALSE;
2:
nDevice2:=FALSE;
nDevice3:=TRUE;
10..20:
nDevice1:=TRUE;
```

```
nDevice3:=TRUE;
ELSE
nDevice1:=FALSE;
nDevice2:=FALSE;
nDevice3:=FALSE;
END_CASE;
```

Execution of the CASE statement:

- When nState is 1 or 5: Device 1 turns on and Device 3 turns off.
- When nState is 2: Device 2 turns off and Device 3 turns on.
- When nState is 10~20: Device 1 and Device 3 turn on.
- In all other cases: Devices 1, 2, and 3 turn off.



Notes:

- The expression in a CASE statement must evaluate to an integer value.
- The ELSE clause is optional; some implementations may use only the CASE...OF…END_CASE structure.

5.2.6 FOR Loop Statement

The FOR loop initializes a variable, repeats nested statements while a condition is TRUE, evaluates iteration expressions, and terminates when the condition becomes FALSE.

Syntax:

```
FOR <Variable> := <Initial_Value> TO <Target_Value> {BY <Step>} DO
<Statements>
END_FOR;
```

Execution sequence:

Check if the <Variable> lies within the range from <Initial_Value> to <Target_Value>.

If <Variable> is less than <Target_Value>, execute <Statements>.

If <Variable> exceeds <Target_Value>, skip the loop.

After each iteration, increment <Variable> by the specified <Step> The step can be any integer value (default: 1 if unspecified). Terminate the loop when <Variable> exceeds <Target_Value>.

Conceptually, a FOR loop operates like a copier: a preset number of iterations (copies) is defined, and the loop stops once the count is reached.

The FOR loop is the most common iteration structure, designed for fixed-count repetition. However, its flexibility allows implementation of other loop patterns through varying coding approaches. A practical example is provided below to demonstrate its usage.

Example:

Calculating 2^5 using a FOR loop:

```
// Variable declaration
Counter:INT; (*Loop counter*)
Var1:INT; (*Output result*)
// Program
FOR Counter:= 1 TO 5 BY 1 DO
Var1:= Var1 * 2;
END_FOR;
```

If Var1's initial value is 1, its final value becomes 32 after the loop.

Note:

If the <Target_Value> equals the <Variable> limit value, the loop becomes infinite. For example, if the counter variable (e.g., Counter of type SINT, range: -128 to 127) is assigned a <Target_Value> of 127, the controller enters

an infinite loop. Thus, avoid setting <Target_Value> to limit values.

5.2.7 WHILE Loop Statement

The WHILE loop operates similarly to the FOR loop but differs in its termination condition: it uses an arbitrary Boolean expression to determine iteration.

Syntax:

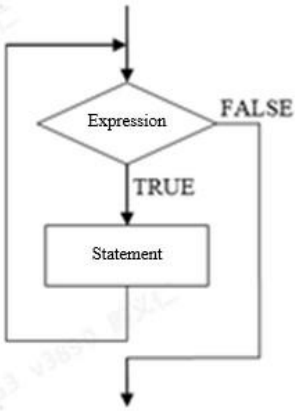
```
WHILE <Boolean_Expression>
<Statement>;
END_WHILE;
```

Execution sequence:

Evaluate the <Boolean_Expression>.

If the result is TRUE, execute <Statement> and repeat.

If the initial evaluation is FALSE, skip <Statement> and exit the loop immediately. The execution flow is as shown below.



Note:

- If the <Boolean_Expression> remains TRUE indefinitely, an infinite loop occurs. Avoid this by modifying loop conditions, e.g., using increment/decrement counters.

A WHILE loop behaves like controlling a motor in an industrial setting: the motor runs continuously while the "Start" button is pressed (Boolean = TRUE) and stops immediately when the "Stop" button is pressed (Boolean = FALSE).

Example 7.X: Execute the loop body as long as the counter is non-zero.

```
// Variable declaration
Counter:INT;
Var1:INT; (*Counter*)
// Program
WHILE Counter <> 0 DO
Var1:= Var1 * 2;
Counter := Counter - 1;
END_WHILE
```

In a certain sense, WHILE loops are more powerful than FOR loops because they do not require prior knowledge of the iteration count before execution. Consequently, only these two loop types are necessary in some scenarios. However, when the exact number of iterations is predetermined, FOR loops are preferable as their fixed structure inherently prevents infinite loops.

5.2.8 Repeat Loop

The REPEAT loop differs from the WHILE loop because it checks the termination condition after executing the loop. This ensures the loop executes at least once, regardless of the initial condition.

Syntax:

```
REPEAT
<Statements>
UNTIL
<Boolean_Expression>
END_REPEAT;
```

Execution sequence:
The <Statements> are executed if the <Boolean_Expression> evaluates to FALSE.
The loop terminates when the <Boolean_Expression> becomes TRUE.
If the <Boolean_expression> is TRUE after the first iteration, the <Statements> execute exactly once.

Note:
If the <Boolean_Expression> remains TRUE indefinitely, an infinite loop occurs. Avoid this by modifying loop conditions, e.g., using increment/decrement counters.

Example: Demonstrating a REPEAT loop that terminates when the counter reaches 0.

```
// Variable declaration
Counter:INT;
// Program
REPEAT
Counter:=Counter+1;
UNTIL
Counter=0
END_REPEAT;
```

In this case, the REPEAT loop executes at least once per program cycle. The Counter, defined as INT (0~32767), performs 32767 increment-by-1 operations per cycle. Due to the post-condition check, the is incremented once (to 1) before evaluation. The variable continues accumulating until it overflows to 0, thereby exiting the loop.

5.2.9 EXIT Statement

The EXIT statement immediately terminates the FOR, WHILE, or REPEAT loop, regardless of its termination condition.

Syntax:

```
EXIT;
```

Example: Using EXIT to avoid division by zero in an iterative loop.

```
FOR Counter:=1 TO 5 BY 1 DO
INT1:=INT1/2;
IFINT1= 0THEN
EXIT; (*Prevent division by zero*)
END_IF
Var1:=Var1/INT1;
END_FOR
```

When INT1 equals 0, the EXIT statement terminates the FOR loop immediately.

5.2.10 CONTINUE Statement

The CONTINUE statement, an extension based on the IEC 61131-3 standard, can be used within FOR, WHILE, or REPEAT loops. It skips the remaining code in the current iteration and immediately starts a new loop iteration. In nested loops, CONTINUE affects only the innermost loop containing it.

Syntax:

```
CONTINUE;
```

Example: Using CONTINUE to avoid division by zero in an iterative loop.

```
// Variable declaration
Counter:INT; (*Loop counter*)
INT1, Var1:INT; (*Intermediate variables*)
Erg:INT; (*Output result*)
// Program
FOR Counter:=1 TO 5 BY 1 DO
INT1:=INT1/2;
IF INT1=0 THEN
CONTINUE; (*To avoid division by zero*)
END_IF
Var1:= Var1 / INT1; (*Executed only if INT1 ≠ 0*)
END_FOR;
Erg:=Var1;
```

5.2.11 RETURN Statement

The RETURN statement terminates the execution of the current main program, subroutine, function block (FB), or function (FC).

Syntax:

```
RETURN;
```

Example: Using RETURN to exit immediately when a condition is met.

```
// Variable declaration
nCounter:INT;
bSwitch:BOOL; (*Switch signal*)
// Program
IF bSwitch=TRUE THEN
RETURN;
END_IF;
nCounter:=nCounter+1;
```

If bSwitch is FALSE, nCounter increments by 1 each cycle; if bSwitch is TRUE, nCounter retains its value from the previous cycle, and the RETURN statement exits the current program immediately.

5.2.12 GOTO and LABEL Statements

The GOTO statement enables unconditional jumps to a labeled code line.

Syntax:

```
GOTO <identifier>;
LABEL <identifier>;
```

<identifier> is placed at the start of a program line. When GOTO is executed, control jumps to the line marked by the specified <identifier>.

Note:

- Avoid infinite loops by controlling jumps with conditional statements like IF.

Example: Using GOTO to cycle a counter within 0..10.

```
IF(D2100<10)THEN
GOTO 1;  // Jump to label 1
END_IF
LABEL 0;  // Label 0
D2100:=0;
LABEL 1;  // Label 1
D2100:=D2100+1;
```

LABEL0 and LABEL1 are not variables, and require no declaration.

The IF statement checks if the counter is within the 0~10. If true, the GOTO1 statement directs the program to jump to LABEL1 in the next cycle, executing D2100:=D2100+1 to increment the counter. Otherwise, it jumps to Label1 and executes nCounter:=0 to reset the counter. This functionality can also be implemented with FOR, WHILE, or REPEAT loops. GOTO statement should generally be avoided, as it reduces code readability and maintainability.

5.2.13 Comment

Comments are critical in programs, enhancing readability without affecting execution. In the ST editor, comments can be added anywhere in the declaration or execution sections.

The ST language supports two comment formats:

Multi-line comments enclosed between (* and *), allowing block annotations.

Single-line comments starting with //, extending to the end of the line.

5.3 ST Language Instructions

5.3.1 Program Control Instructions

Instruction	Description	Example
EI	Enable interrupt	EI();
DI	Disable interrupt	DI();

5.3.2 Bit Processing Instructions

Instruction	Description	Example
ZRST	Batch bit reset	ZRST(Address, Number);
ZSET	Batch bit set	ZSET(Address, Number);

5.3.3 Pointer Instructions

Instruction	Description	Example
PTGET	Pointer assignment	PTGET (Pointer, Target);
PTINC	Pointer address increment by 1	PTINC (Pointer);
PTDEC	Pointer address decrement by 1	PTDEC (Pointer);
PTADD	Pointer address addition	PTADD(Source, Offset, Target);
PTSUB	Pointer address subtraction	PTSUB(Source, Offset, Target);
PTMOV	Pointer assignment between variables	PTMOV(Source, Target);

5.3.4 Numeric Conversion Instructions

Instruction	Description	Example
TO_BOOL	Convert variable to BOOL type	TO_BOOL(Source);
TO_INT	Convert variable to INT type	TO_INT(Source);
TO_DINT	Convert variable to DINT type	TO_DINT(Source);
TO_REAL	Convert variable to REAL type	TO_REAL(Source);

5.3.5 Mathematical Function Instructions

Instruction	Description	Example
ABS	Single-word absolute value	ABS(Source);
SQT	Single-word square root	SQT(Source);
LN	Natural logarithm	LN(Source);
LOG	Common logarithm	LOG(Source);
EXP	Natural exponent	EXP(Source);
POWER	Exponentiation	POWER(Source1, Source2);
SIN	Sine function	SIN(Source);
COS	Cosine function	COS(Source);
TAN	Tangent function	TAN(Source);
ASIN	Arcsine function	ASIN(Source);
ACOS	Arccosine function	ACOS(Source);
ATAN	Tangent function	ATAN(Source);

5.3.6 Data Processing Instructions

Instruction	Description	Example
SHL	Word left shift	SHL(Source, Number);
SHR	Word right shift	SHR(Source, Number);
MAX	Take maximum value of words	MAX (Start, Number);
MIN	Take minimum value of words	MIN (Start, Number);
SEL	Select between two words based on condition	SEL (Condition, Data1, Data2);

5.3.7 String Processing Instructions

Instruction	Description	Example
STRCLEAR	Clear string	STRCLEAR (String);
STRDELET	Delete string	STRDELET(Source, DeleteLength, DeletePos, Result);
STRADD	Concatenate strings	STRADD(String1,String2,Result);
STRLEN	Get string length	STRLEN(Source, Length);
STRRIGHT	Read string from right side	STRRIGHT(Source, Result, Length);
STRLEFT	Read string from left side	STRLEFT(Source, Result, Length);
STRMIDR	Read string from specified position	STRMIDR(Source, Result, StartPos, ReadLength);
STRMIDW	Replace string at specified position	STRMIDW(Replace, Source, StartPos, ReplaceLength);
STRINSERT	Insert string at specified position	STRINSERT(Source, Insert, InsertPos, Result);
STRINSTR	Search for string	STRINSTR(Search, Source, ReturnPos, StartPos);
STRCMP	Compare strings	STRCMP(String1, String2, Result);

5.3.8 Timer Instruction

Instruction	Description	Example
DTPR	Pulse timer instruction	DTPR(Enable:=, PT:=, R:=, Q=>, ET=>);
DTON	On-delay timer instruction	DTON(Enable:=, PT:=, R:=, Q=>, ET=>);
DTOF	Off-delay timer instruction	DTOF(Enable:=, PT:=, R:=, Q=>, ET=>);
DTACR	Accumulation timer instruction	DTACR(Enable:=, PT:=, R:=, Q=>, ET=>);

5.3.9 MC Axis Control Instructions

Instruction	Description	Comment
MC_Power	Enable axis control	
MC_Reset	Reset fault	
MC_Home	Homing	
MC_Stop	Stop	
MC_MoveVelocity	Velocity control	
MC_Jog	Jog	
MC_Move	Positioning	
MC_ReadAxisError	Read axis error	
MC_ReadPosition	Read actual position	
MC_ReadStatus	Read axis status	
MC_TorqueControl	Torque control	
MC_SetPosition	Set position	
MC_MoveSuperImposed	Superimposed motion	
MC_TouchProbe	Touch probe	
MC_Linear	Linear interpolation	
MC_Circle_CW	Clockwise circular interpolation	
MC_Circle_CCW	Counter-clockwise circular interpolation	
MC_MoveBuffer	Multi-segment move	
MC_MoveAbsolute	Absolute move	
MC_MoveRelative	Relative move	
MC_ReadVelocity	Read actual velocity	
MC_MoveFeed	Feed-interrupt move	
MC_Halt	Motion halt	
MC_SyncTorqueControl	Synchronized torque control	
MC_ReadActualTorque	Read actual torque	
MC_FollowVelocity	Velocity superposition control	
MC_ReadDigitalInput	Read digital inputs	
MC_MoveVelocityCSV	CSV-based variable pulse width velocity control	
MC_SyncMoveVelocity	CSV-based synchronized velocity control	
MC_MoveThreeDimensionalCircular	3D circular interpolation	
MC_MoveSpiral	Spiral interpolation	
MC_SetAxisConfigPara	Axis parameter configuration	

5.3.10 Axis Group Instructions

Instruction	Description	Comment
MC_MoveLinear	Linear interpolation	

MC_MoveCircular	Circular interpolation	
MC_GroupStop	Stop axis group motion	
MC_GroupPause	Pause axis group motion	
MC_AddAxisToGroup	Add axis to group	
MC_RemoveAxisFromGroup	Remove axis from group	
MC_PathAdd	Path addition	
MC_PathMov	Path motion execution	
MC_SetForwardLookingPara	Set forward-looking parameters	

5.3.11 Electronic Cam Instructions

Instruction	Description	Comment
MC_GearIn	Engage gear synchronization	
MC_GearOut	Disengage gear synchronization	
MC_CombineAxes	Dual-spindle gear synchronization	
MC_CamIn	Engage cam profile	
MC_CamOut	Disengage cam profile	
MC_GenerateCamTable	Generate/update cam table	
MC_Phasing	Master axis phase offset	
MC_GetCamTablePhase	Get cam table phase value	
MC_GetCamTableDistance	Get cam table displacement	
MC_GenerateTappet	Generate/update tappet profile	
MC_GetCamTable	Get cam table data	
MC_RotaryCut	Rotary cutting	
MC_ChasingCut	Chasing cutting	
MC_GetCamTableVelocityRatio	Get cam table velocity ratio	

5.3.12 EtherCAT Communication Instructions

Instruction	Description	Comment
E_WriteParameter_CoE	Write CoE slave axis parameters	
E_ReadParameter_CoE	Read CoE slave axis parameters	

5.3.13 Local High-Speed Counter Instructions

Instruction	Description	Comment
HC_Preset	Preset high-speed counter value	
HC_Counter	Enable high-speed counter	
HC_TouchProbe	Enable high-speed counter touch probe	
HC_Compare	Configure high-speed counter comparison	
HC_ArrayCompare	Configure high-speed counter array comparison	
HC_StepCompare	Configure high-speed counter step comparison	

5.4 Smart Input and Tooltips

5.4.1 Engineering Example

MC_Power usage example (ST language):

```
MC_Power(Enable := bEnable, // Enable pin
Axis := dAxis, // Axis number
Status => bStatus, // Axis enable flag
Busy => bBusy, // Busy flag
Error => dError, // Instruction error flag
ErrorID => dErrorID); // Error code
```

5.4.2 Quick Input

After typing an instruction name, press the TAB key to auto-complete its parameters. If a parameter's default value is "???", it must be explicitly assigned. Other parameters are optional based on requirements.

```
MC_Jog(Enable:=??,
Axis:=??,
SpeedMode:=,
Forward:=??,
Backward:=??,
Velocity:=??,
Acceleration:=??,
Deceleration:=,
Jerk:=??,
Busy=>,
Aborted=>,
Error=>,
ErrorID=>);
```

5.4.3 Mouse Hover Tooltips

Variables: Displays variable name, data type, and comment. FB/FC/Instructions: Displays function name, function type (FB/FC), comment, input/output parameters, and parameter descriptions (name, type, comment).

5.5 Programming Language (LD)

Ladder Diagram (LD) is a graphical programming language with a structure resembling electrical circuits. It consists of multiple networks (also called rungs), each starting with a vertical power rail (energy flow line) on the left. A network comprises contacts, coils, function blocks (functions, function blocks, programs, execution blocks, actions, methods), jumps, labels, and connecting wires.

Key elements in LD include contacts, coils, function blocks, branches, and comments. These elements are added to networks through inserting, dragging, wiring, copying, and pasting operations to form execution logic.

LD supports online debugging for monitoring, writing values, forcing values, and breakpoints.

For detailed LD specifications, please refer to AutoSoft.chm. In AutoSoft's menu bar, select Help > Help Manual to open the required manual.

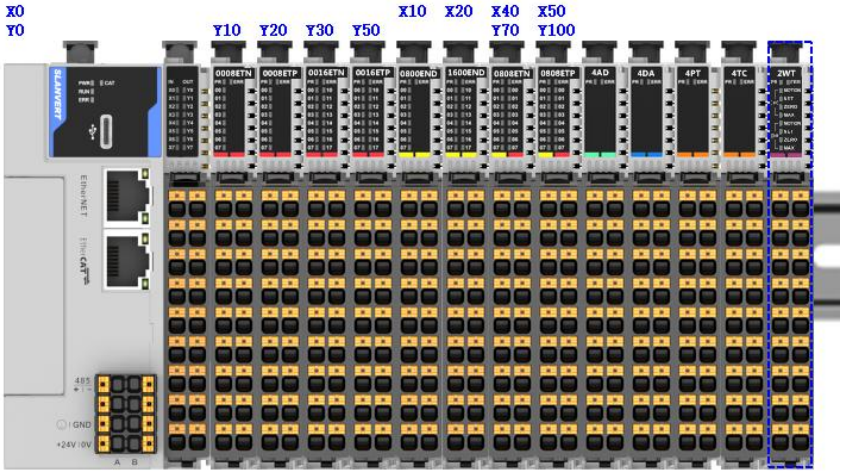
5.6 Sequential Function Chart (SFC)

The SFC is an intuitive graphical programming language designed for process-oriented programming. All SFC elements can be accessed via corresponding icons or menu items in the SFC toolbar and menu. Users can click specific icons to insert desired elements and define their properties in the SFC element properties dialog. The SFC toolbar and menu also provide shortcuts for adding connection lines, allowing users to establish connections between SFC elements as needed. For detailed SFC specifications, refer to AutoSoft.chm. In AutoSoft's menu bar, select Help > Help Manual to open the required manual.

6 Extension Module Configuration

6.1 SH Local Right Extension Module Configuration

The SH controller can support up to 16 local extension modules. Access to these local extensions is implemented through module configuration. The hardware configuration diagram for connecting local extension modules to the SH is shown below.



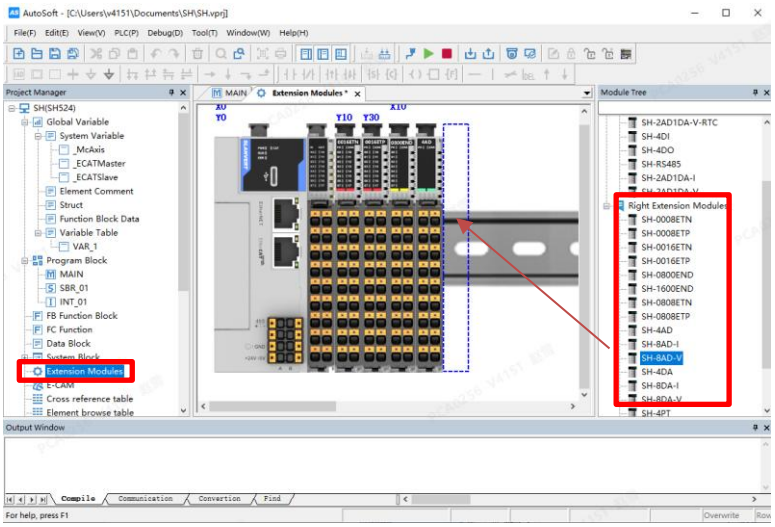
The supported local extension module models are listed in the table below.

I/O Module	Module Description
SH-1600END	16-channel digital input module
SH-0800END	8-channel digital input module
SH-0016ETN	16-channel NPN digital output module
SH-0016ETP	16-channel PNP digital output module
SH-0808ETN	8-channel digital input module and 8-channel NPN digital output module
SH-0808ETP	8-channel digital input module and 8-channel PNP digital output module
SH-0008ETN	8-channel NPN digital output module
SH-0008ETP	8-channel PNP digital output module
SH-4AD	4-channel analog input module
SH-4DA	4-channel analog output module
SH-4PT	4-channel thermal resistance temperature detection input module
SH-4TC	4-channel thermocouple temperature detection input module
SH-2WT	2-channel input weighing module

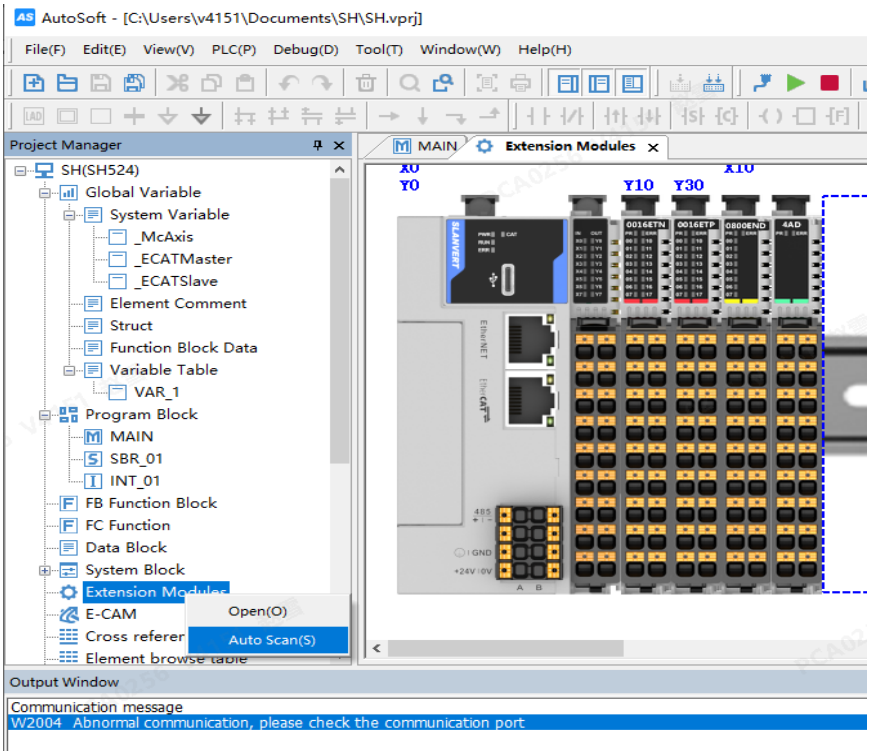
6.1.1 Extension Module Auto-Scan

Module configuration provides two methods:

(1) Method 1: Manually configure each module one by one based on the actual connection order, as shown in the figure below. (Note: The configuration order must match the actual connection sequence; otherwise, a Special Module Error will be reported.)



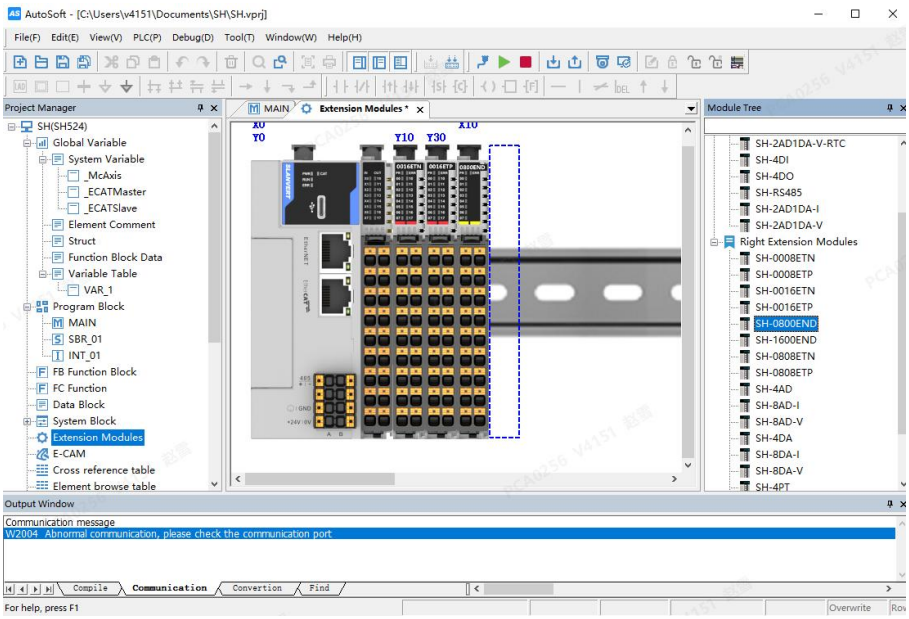
(2) Method 2: Right-click on "Extension Modules", select "Auto-Scan", and verify whether the scanned modules match the actual connections. Then click "Update" to automatically add the scanned modules to the configuration, as shown in the figure below:



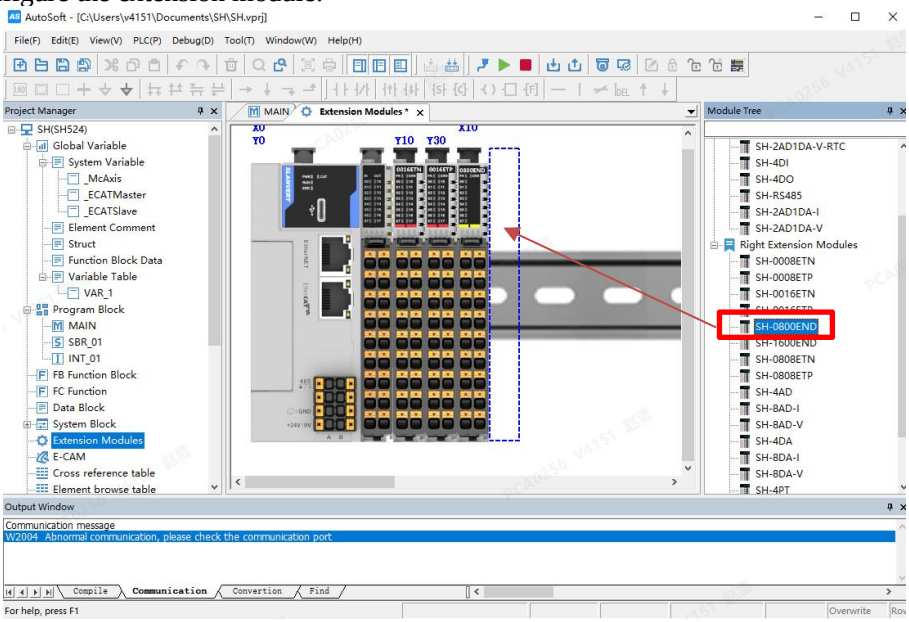
6.1.2 IO Module Configuration

Take SH-0808ETP/SH0016ENT as examples:

1. Double-click "Extension Modules" in the Project Manager area to enter the interface as below.



2. Click the position number on the rail corresponding to the actual installation location of the extension module, and double-click the module on the right or hold the left mouse button to drag the module onto the rail to configure the extension module.



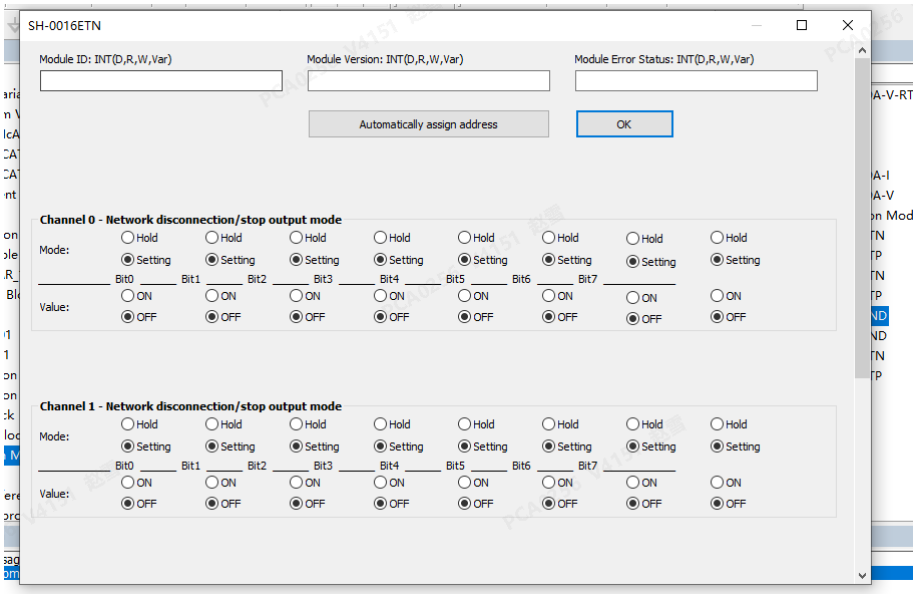
When connected to the main module, the extension module's X ports and Y ports are numbered sequentially following the X and Y port numbering of the main module. For example: If the main module is a SH universal type and a SH-0808ETP extension module is connected, the X ports start from X10–X16 and Y ports from Y10–Y16. If a SH0016ENT module is connected next, its output ports will be Y20–Y27 and Y30–Y37. Subsequent modules follow this numbering pattern accordingly.

Note:

- X and Y use octal numbering.
- After adding an IO extension module, the system automatically assigns addresses to X/Y ports, displayed in order above the module. Manual address assignment is currently not supported.

IO Module Function Parameter Settings

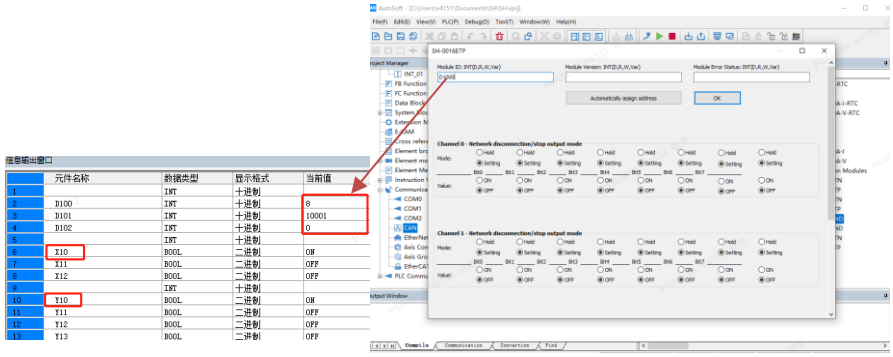
Double-click the added module to configure parameters such as digital I/O input filter settings, output configurations, and module version information, as shown in the figure below.



Enter the starting address D100 in "Module ID" and click "Automatically assign address" for addresses allocation.

- For input channels, set input filter time of 0ms~63.75ms.
- For output channels, configure the output settings.
- Output Hold: Set the ON/OFF of output channels.
- Output Set Value: Determines the output channel states based on set values.
- Offline Output Value: When the output mode is set to "Output Set Value," the output channels follow this parameter.

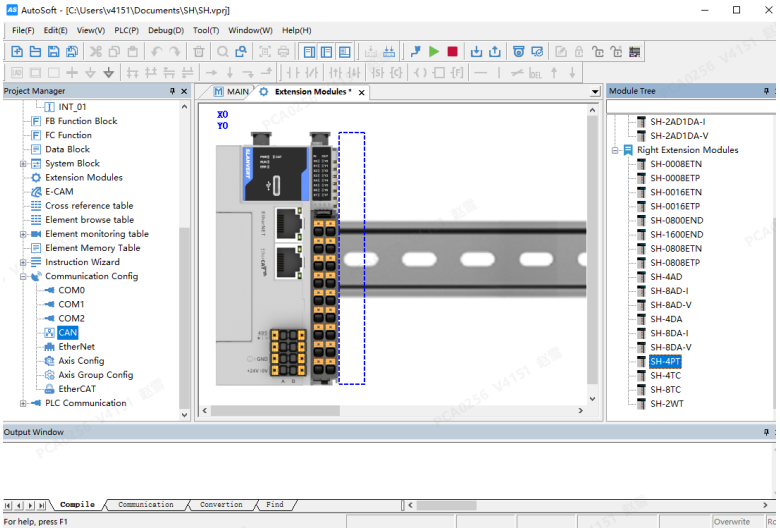
After completing the above configurations, compile, download, and debug as shown in the figure below:



6.1.3 4PT Module Configuration

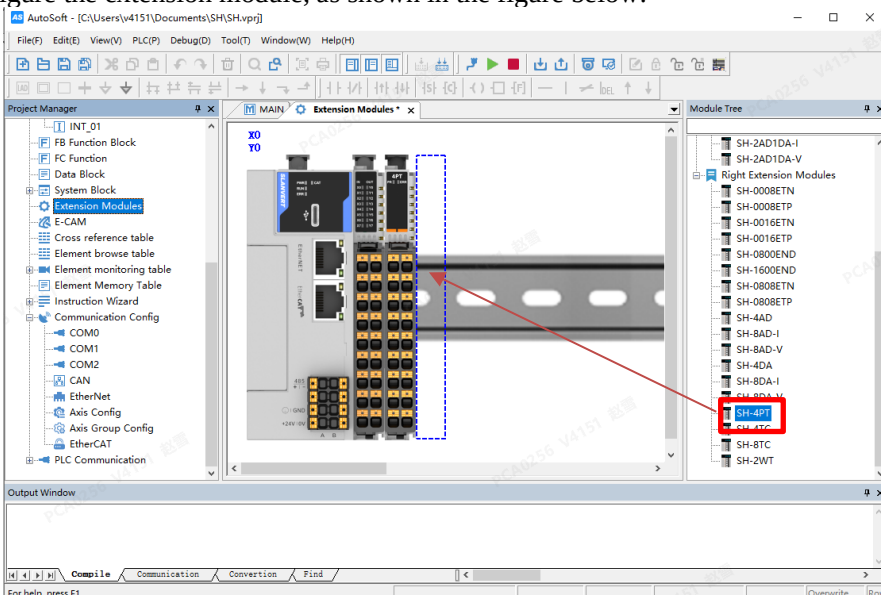
Taking the first channel of SH-4PT as an example, connect a PT100 sensor to read its current temperature:

1. Double-click "Extension Modules" in the Project Manager area to enter the interface as below.

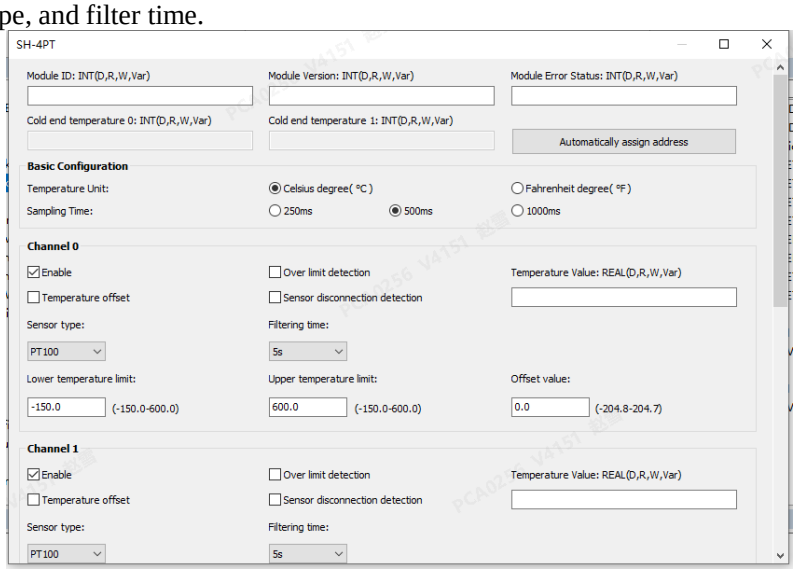


2. Click the position number on the rail corresponding to the actual installation location of the extension module, and double-click the module on the right or hold the left mouse button to drag the module onto the

rail to configure the extension module, as shown in the figure below:



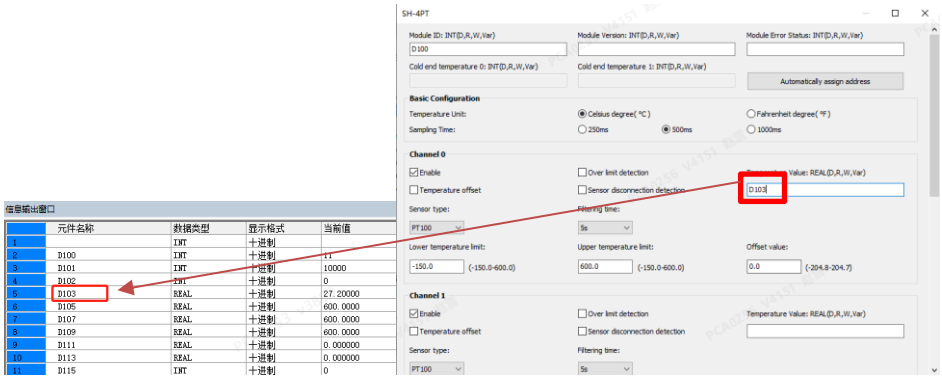
3. Double-click the added 4PT module to configure parameters such as temperature unit (°C/°F), sampling time, sensor type, and filter time.



Channel Parameters Description

- Module Information: Module ID, module version, module error status.
- Basic Configuration: Temperature unit (°C/°F), sampling time (default to 500ms).
- Channel Enable: Enabled by default. When enabled, channel signals are detected.
- Temperature Offset: When enabled, the measured temperature value + offset value is displayed as the final value. Disabled by default.
- Limit Detection: When enabled, an alarm is triggered if the temperature exceeds the upper limit or drops below the lower limit.
- Sensor Disconnection Detection: When enabled, an alarm is triggered if the sensor disconnection is detected.
- Sensor Type: PT100, PT500, PT1000, CU50, CU100, KTY84, NTC-5K, NTC-10K, etc.
- Filter Time: Default to 5s.
- Temperature Value: Current filtered temperature value.
- Quick Address Assignment: Enter a register address (e.g., D100) to "Module ID" and click "Automatically assign address" to allocate addresses for all channels.

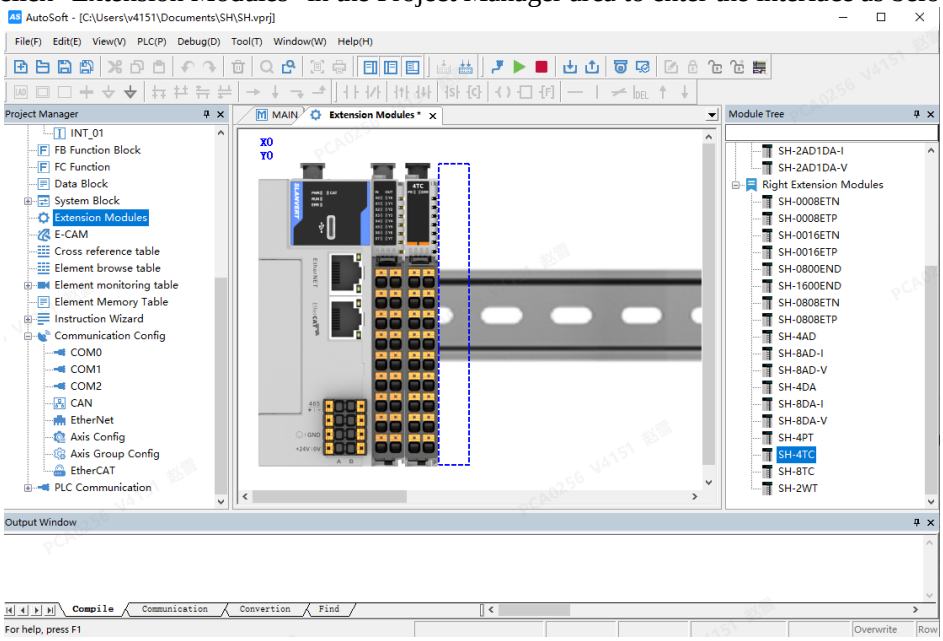
4. After configuration, compile, download, and monitor. The D103 register displays the current sensor temperature value, as shown in the figure below:



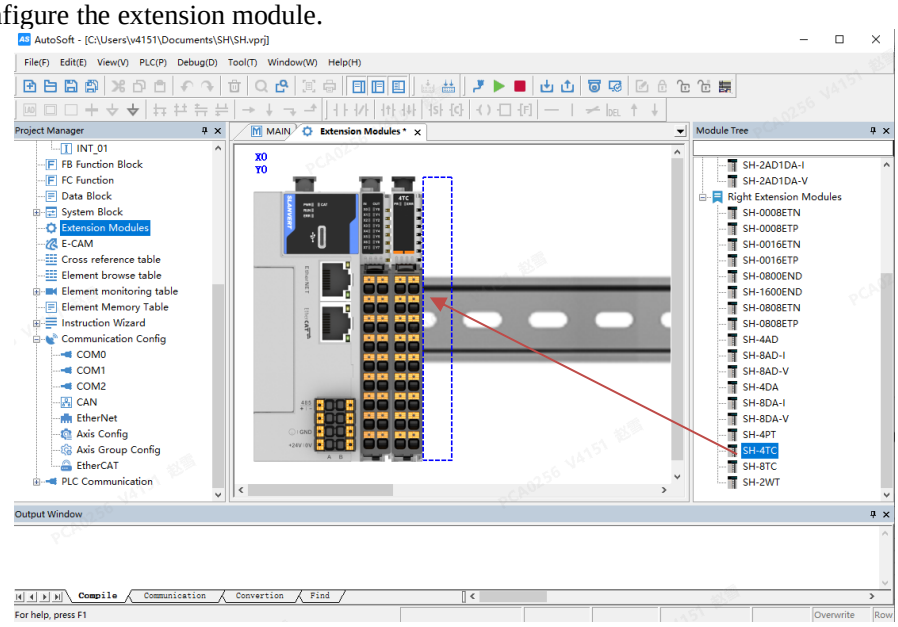
6.1.4 4TC Module Configuration

Taking the first channel of SH-4TC as an example, connect a K-type sensor to read its current temperature:

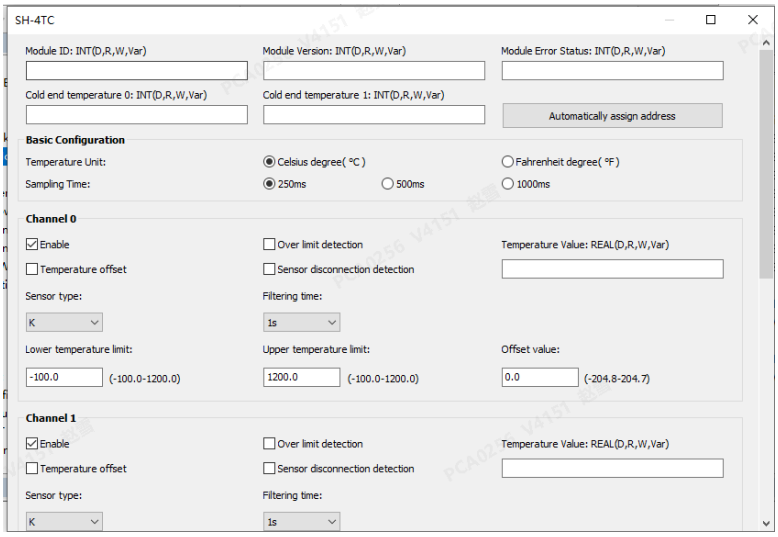
1. Double-click "Extension Modules" in the Project Manager area to enter the interface as below.



2. Click the position number on the rail corresponding to the actual installation location of the extension module, and double-click the module on the right or hold the left mouse button to drag the module onto the rail to configure the extension module.



3. Double-click the added 4TC module to configure parameters such as temperature unit (°C/°F), sampling time, sensor type, and filter time.


SH-4TC
Module ID: INT(D,R,W,Var)
Module Version: INT(D,R,W,Var)
Module Error Status: INT(D,R,W,Var)
Cold end temperature 0: INT(D,R,W,Var)
Cold end temperature 1: INT(D,R,W,Var)
Automatically assign address
Basic Configuration
Temperature Unit:
Celsius degree(°C)
Fahrenheit degree(°F)
Sampling Time:
250ms
500ms
1000ms
Channel 0
Enable
Over limit detection
Temperature Value: REAL(D,R,W,Var)
Temperature offset
Sensor disconnection detection
Sensor type:
Filtering time:
K
1s
Lower temperature limit:
Upper temperature limit:
Offset value:
-100.0
(-100.0-1200.0)
1200.0
(-100.0-1200.0)
0.0
(-204.8-204.7)
Channel 1
Enable
Over limit detection
Temperature Value: REAL(D,R,W,Var)
Temperature offset
Sensor disconnection detection
Sensor type:
Filtering time:
K
1s

Channel Parameters Description

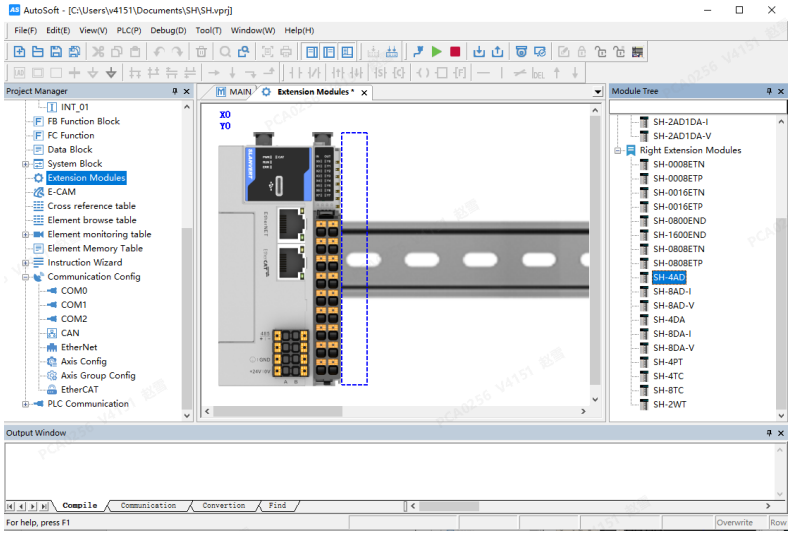
- Module Information: Module ID, module version, module error status, and cold spot temperature.
- Basic Configuration: Temperature unit (°C/°F), sampling time (default to 250ms).
- Channel Enable: Enabled by default. When enabled, channel signals are detected.
- Temperature Offset: When enabled, the measured temperature value + offset value is displayed as the final value. Disabled by default.
- Limit Detection: When enabled, an alarm is triggered if the temperature exceeds the upper limit or drops below the lower limit.
- Sensor Disconnection Detection: When enabled, an alarm is triggered if the sensor disconnection is detected.
- Sensor Type: K, J, E, B, N, R, S, T-type.
- Filter Time: Default to 5s.
- Temperature Value: Current filtered temperature value.
- Quick Address Assignment: Enter a register address (e.g., D100) to "Module ID" and click "Automatically assign address" to allocate addresses for all channels.

4. After configuration, compile, download, and monitor. The R105 register displays the current sensor temperature value, as shown below.

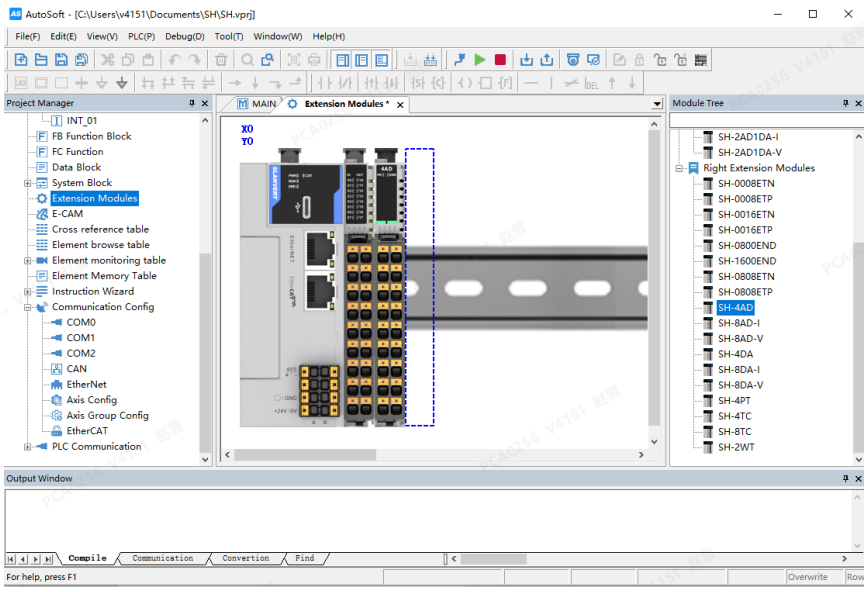
6.1.5 4AD Module Configuration

Taking the first channel of SH-4AD as an example, with a port input voltage of 5V, to read the analog-to-digital converted value:

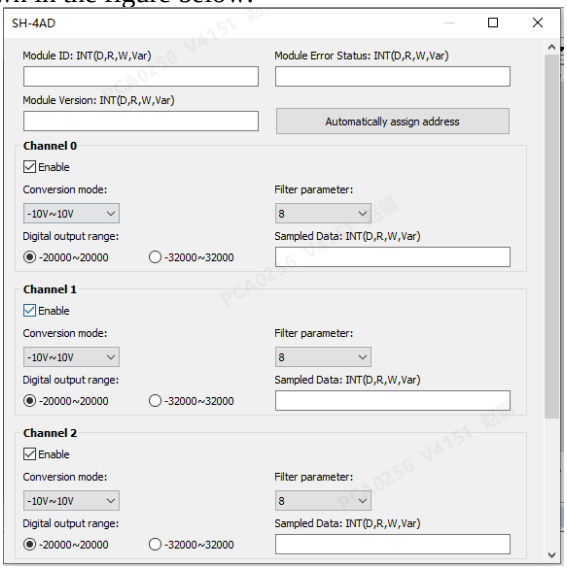
1. Double-click "Extension Modules" in the Project Manager area to enter the interface as below.



2. Click the position number on the rail corresponding to the actual installation location of the extension module, and double-click the module on the right or hold the left mouse button to drag the module onto the rail to configure the extension module.



3. Double-click the added 4AD module to configure parameters such as input mode, filter settings, and conversion range, as shown in the figure below.



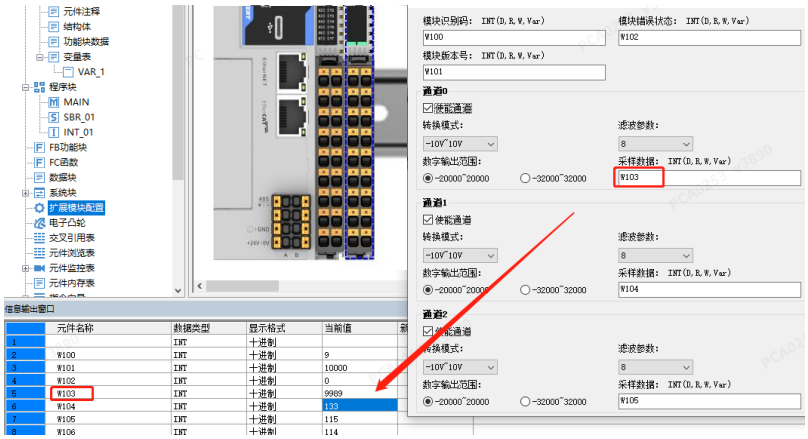
Channel Parameters Description

- Module Information: Module ID, module version, and module error status.
- Channel Enable: Enabled by default. When enabled, channel signals are detected.
- Filter Time: Default to 8 (adjustable range: 0~255).
- Input Mode: Three modes available.

Input Mode	Output Range
-10V~10V	-20000~20000 or -32000~32000
-20mA~20mA	-20000~20000 or -32000~32000
4mA~20mA	-20000~20000 or -32000~32000

- Sampled Data: Current filtered digital value converted from the analog input.
- DO Range: Analog input values are converted to digital values within -20000~20000 or -32000~32000.

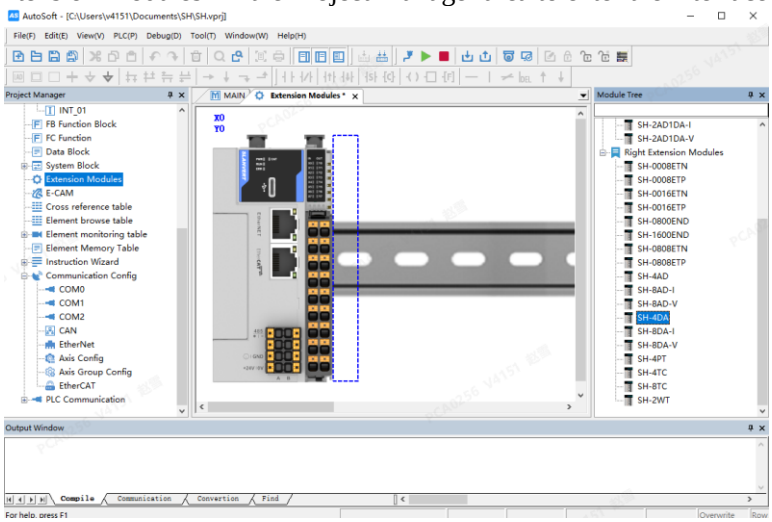
4. After configuration, compile, download, and monitor. W103=9989 displays the converted analog value, as shown in the figure below: This is shown in the figure below:



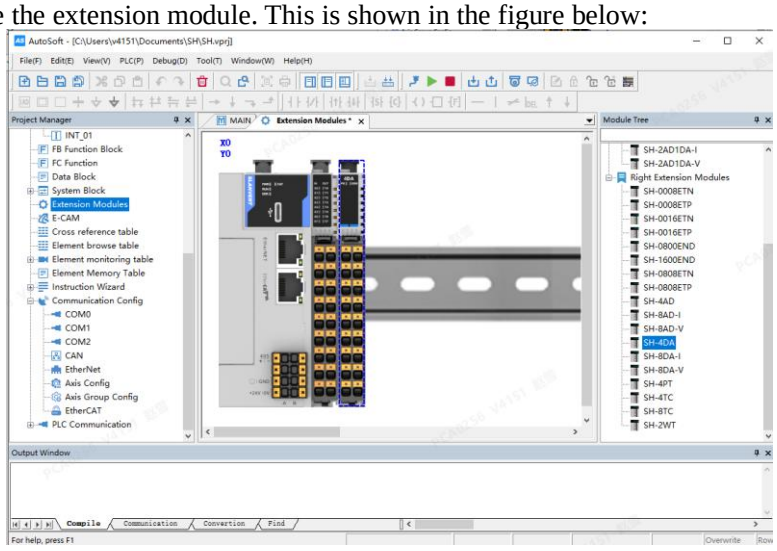
6.1.6 4DA Module Configuration

Taking the first channel of SH-4DA as an example, set the mode value to read the analog-to-digital converted value:

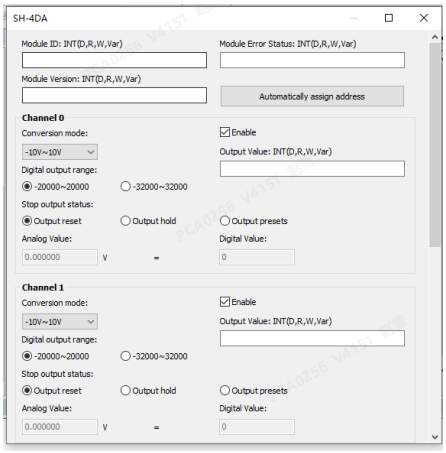
1. Double-click "Extension Modules" in the Project Manager area to enter the interface as below.



2. Click the position number on the rail corresponding to the actual installation location of the extension module, and double-click the module on the right or hold the left mouse button to drag the module onto the rail to configure the extension module. This is shown in the figure below:



3. Double-click the added 4DA module to configure parameters such as output mode, filter settings, and conversion range, as shown in the figure below.



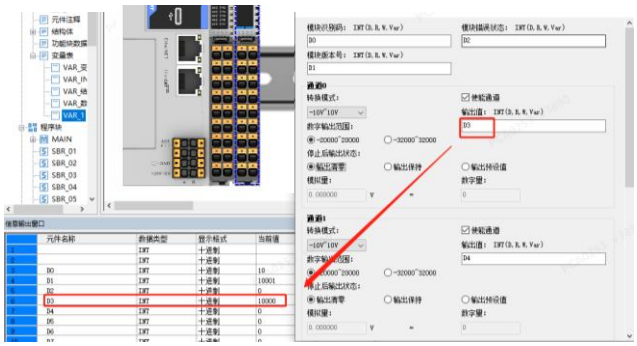
Channel Parameters Description

- Module Information: Module ID, module version, and module error status.
- Channel Enable: Enabled by default. When enabled, channel signals are detected.
- Conversion Mode: Supports voltage and current output.

Output Mode	Range
-10V~10V	-20000~20000 or -32000~32000
-20mA~20mA	-20000~20000 or -32000~32000
4mA~20mA	-20000~20000 or -32000~32000

- Output Mode on Stop: When the host stops or a module error occurs, the output ports can be configured with three stop modes:
- Zero Output: Output is set to 0.
- Hold Output: Output retains the last value.
- Preset Output: Output follows a predefined value.

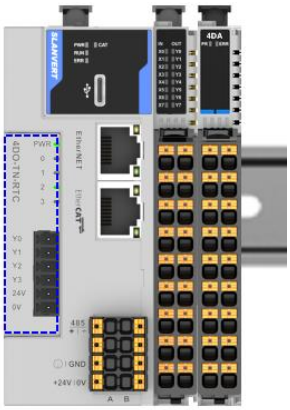
4. After configuration, compile, download, and monitor. Setting D3=10000 will output 5V on Channel 1, as shown below.



6.2 SH Local Left Extension Module Configuration

6.2.1 SH Extension Hardware Configuration

The connection diagram for the left extension modules is shown below.



6.2.2 SH Left Extension Supported Types

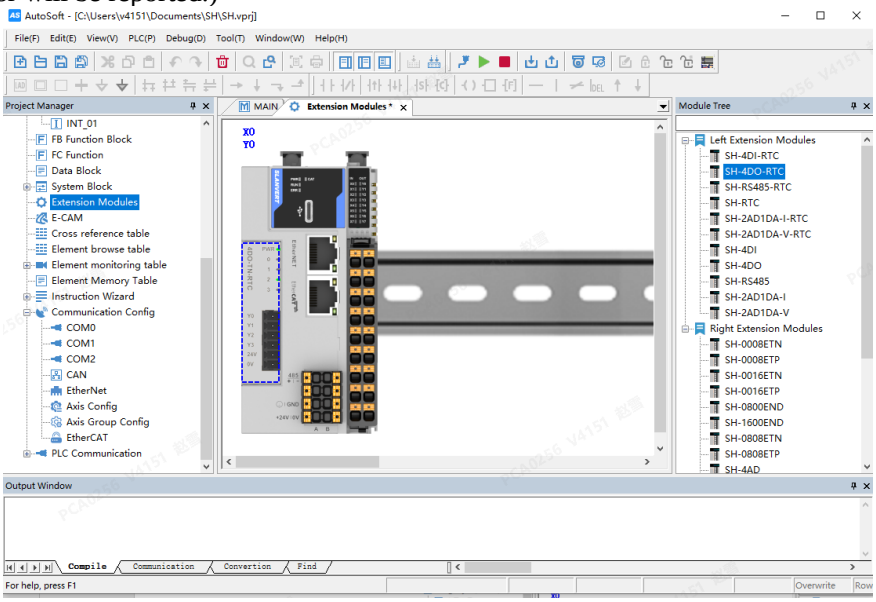
The SH series controller can be extended with up to 1 extension. There are 11 types of extension available, each supporting different functions, as detailed in the table below.

No.	Model	Specification
1	SH-RS485	2×RS485
2	SH-RS485-RTC	2×RS485+real-time clock
3	SH-4DI	4-channel digital input
4	SH-4DI-RTC	4-channel digital input+real-time clock
5	SH-4DO-TN	4-channel drain type digital output
6	SH-4DO-TN-RTC	4-channel drain type digital output+real-time clock
7	SH-2AD1DA-I	2-channel analog input, 1-channel analog output, current mode
8	SH-2AD1DA-I-RTC	2-channel analog input, 1-channel analog output (current mode) + real-time clock
9	SH-2AD1DA-V	2-channel analog input, 1-channel analog output, voltage mode
10	SH-2AD1DA-V-RTC	2-channel analog input, 1-channel analog output (voltage mode) + real-time clock
11	SH-RTC	Real-time clock

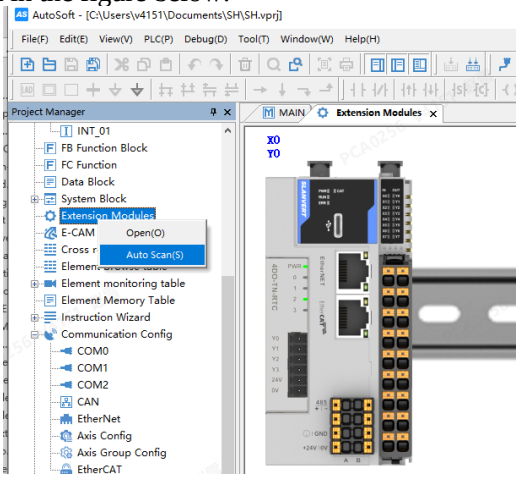
6.2.3 Left Extension Auto-Scan

There are two methods for configuring extensions:

(1) Method 1: Configure manually based on the actual physical connection order of modules, as shown in the figure below. (Note: The configuration order must match the actual connection sequence; otherwise, a module error will be reported.)



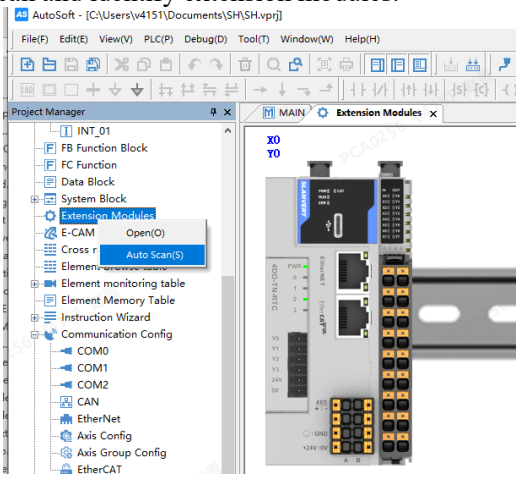
(2) Method 2: Right-click on "Extension Modules", select "Auto-Scan", and verify whether the scanned modules match the actual connections. Then click "Update" to automatically add the scanned modules to the configuration, as shown in the figure below:



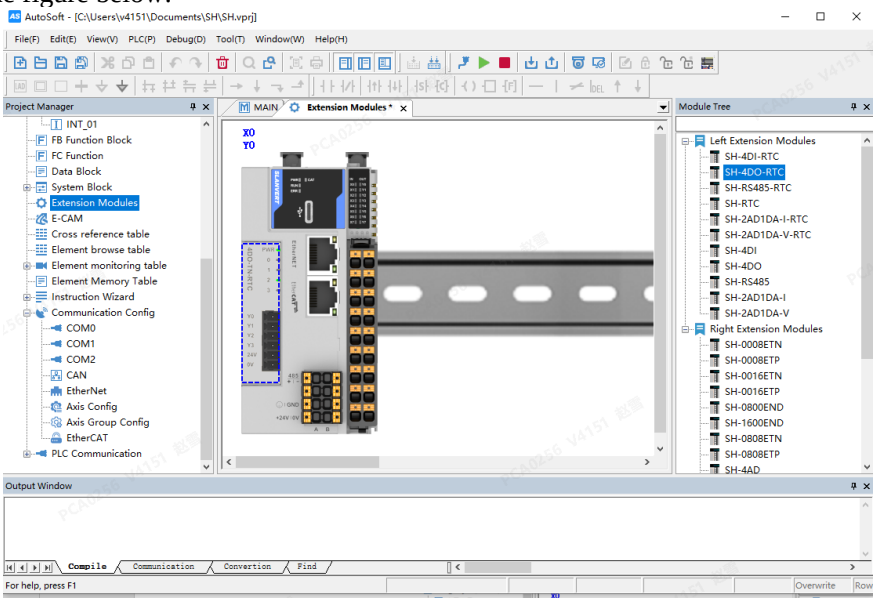
6.2.4 Extension Configuration Example

Using the SH-4DO-RTC module as an example:

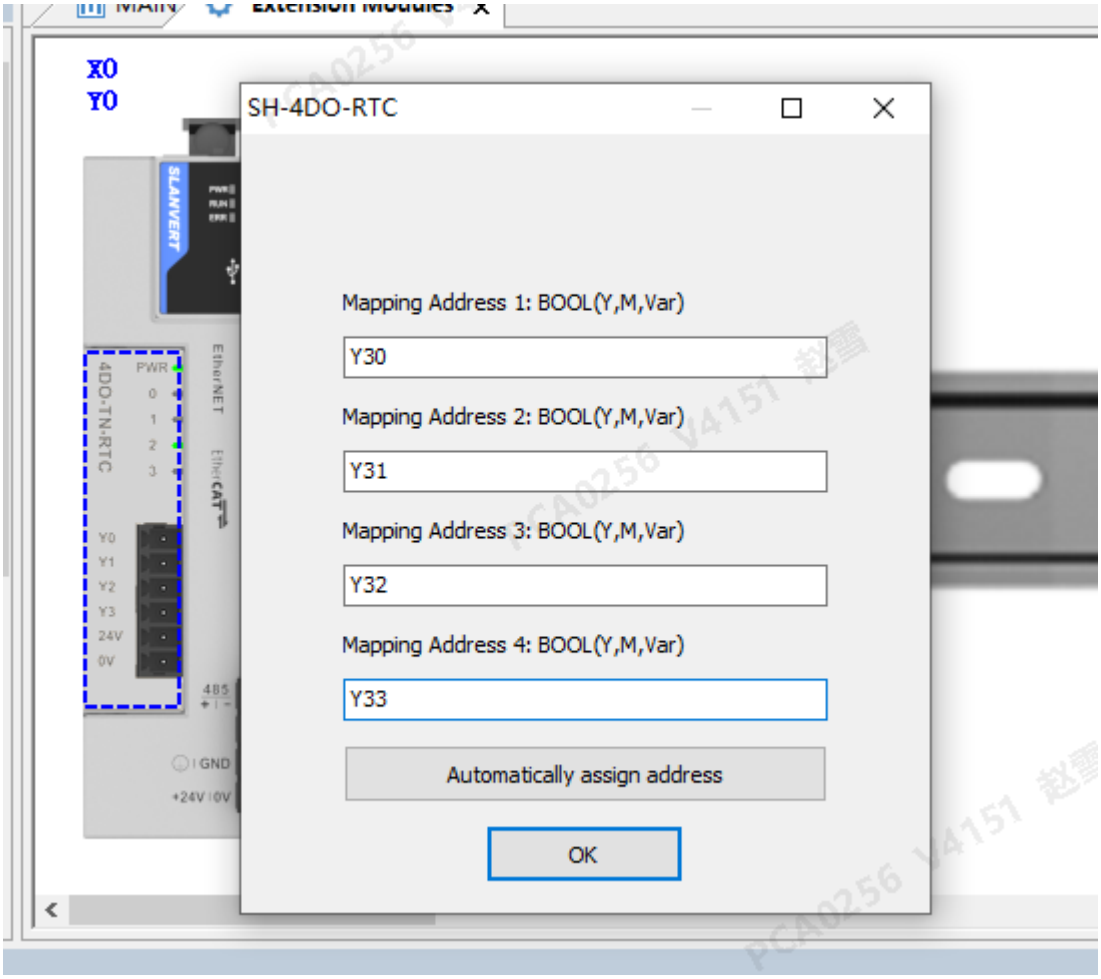
(1) After installing the extension, right-click "Extension Modules" under Project Manager in the software and select the "Auto-Scan" to scan and identify extension modules.



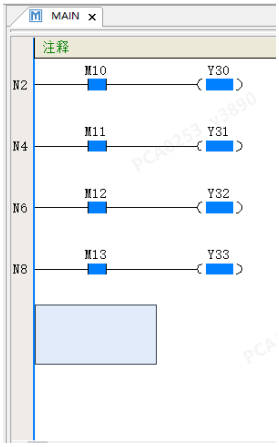
(2) Click "Update Configuration" to automatically add the scanned module to the hardware configuration, as shown in the figure below.



(3) Double-click the added extension to assign addresses, such as Y, M, or custom variables. (Note: Avoid address conflicts with the main unit or right extension modules.) Here, the Y port output addresses are mapped to Y30~Y33, as shown below:



(4) After completing the configuration, compile, download, and invoke the program. When M10 is turned ON, Y30 outputs, as illustrated below:



Extension Type Descriptions

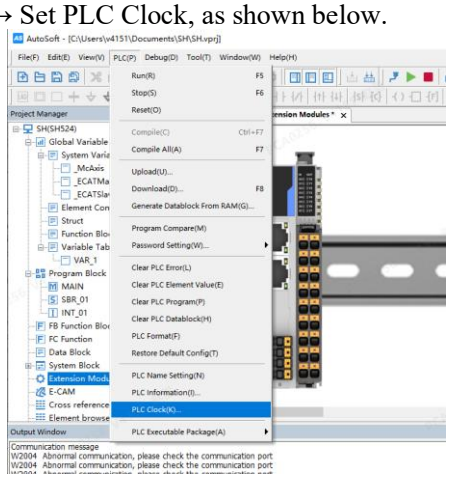
(1) SH-RS485 Extension: The SH supports 2×RS-485, compatible with Freeport protocol, Modbus RTU, and N:N protocol. The communication port configuration is as follows:

No.	Extension Channel	Software Port Configuration
1	1A\1B-	COM1
2	2A\2B-	COM2

(2) SH-RTC Extension: The SH series supports extending 1×RTC. The clock values can be modified via clock instructions. No configuration is required; the extension is automatically recognized by the software. The relevant SD special registers are as follows:

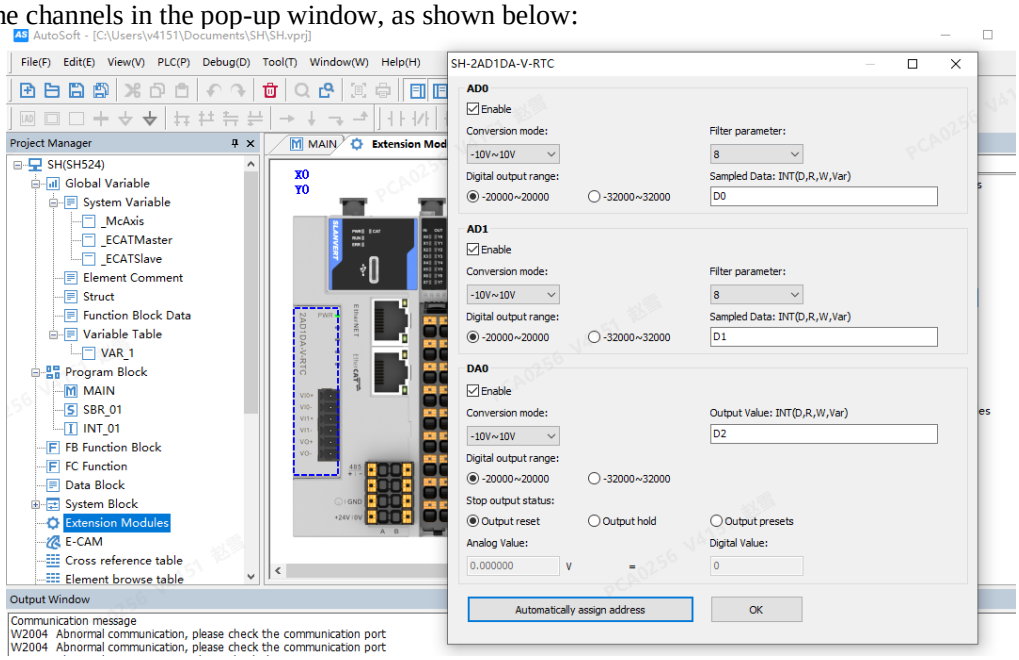
Year	Month	Day	Hour	Minute	Second	Week
SD60	SD61	SD62	SD63	SD64	SD65	SD66

1. Quick clock calibration: When the PC is connected to the PLC, navigate to the menu bar: PLC → PLC Clock → Get Current Clock → Set PLC Clock, as shown below.



(3) Analog Extension

1. The SH controller supports 1-channel left analog extension (SH-2AD1DA-I, SH-2AD1DA-V, or an integrated RTC module). Users can select based on requirements. Double-click the added module, and configure parameters such as conversion mode, filter parameters, and analog-to-digital conversion range for the channels in the pop-up window, as shown below:



6.3 SH-RTU-ETC Coupler

The SH-RTU-ETC coupler can support up to 16 local extension modules. Access to these local extensions is implemented through module configuration. The hardware configuration diagram for connecting local extension modules to the SH is shown below.

The supported local extension module models are listed in the table below.

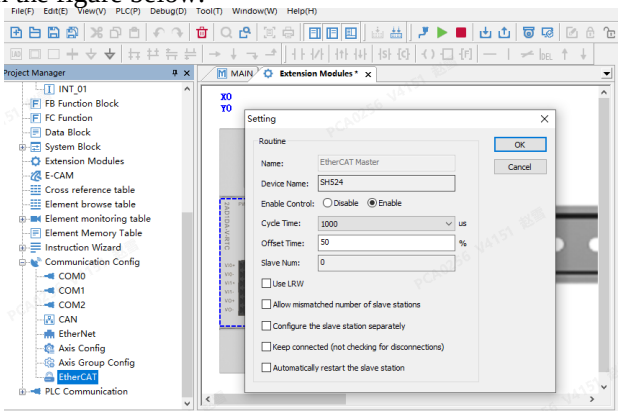
I/O Module	Description
SH-1600END	16-channel digital input module
SH-0800END	8-channel digital input module
SH-0016ETN	16-channel NPN digital output module
SH-0016ETP	16-channel PNP digital output module
SH-0808ETN	8-channel digital input module and 8-channel NPN digital output module
SH-0808ETP	8-channel digital input module and 8-channel PNP digital output module
SH-0008ETN	8-channel NPN digital output module
SH-0008ETP	8-channel PNP digital output module
SH-4AD	4-channel analog input module
SH-4DA	4-channel analog output module
SH-4PT	4-channel thermal resistance temperature detection input module
SH-4TC	4-channel thermocouple temperature detection input module
SH-2WT	2-channel input weighing module

6.3.1 Module Auto-Scan Configuration

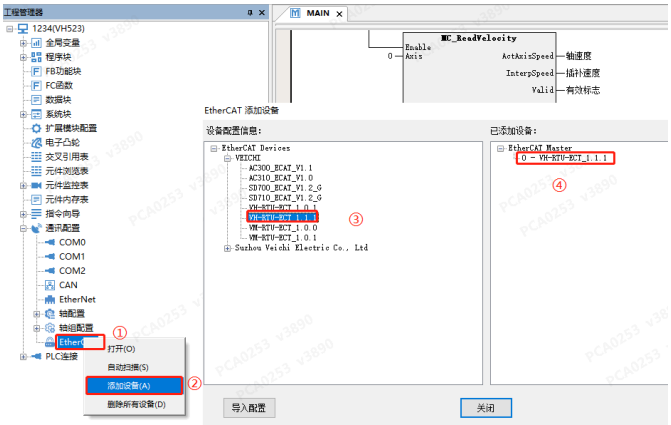
Module configuration provides two methods:

(1) Method 1: Manually configure each module one by one based on the actual connection order, as shown in the figure below. (Note: The configuration order must match the actual connection sequence; otherwise, the coupler will trigger an alarm.)

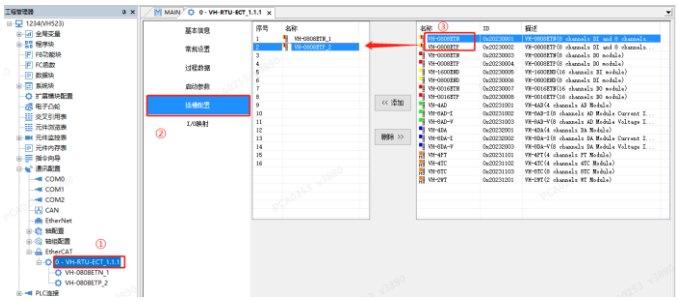
1. In Project Manager area, double-click EtherCAT → check Enable for "Enable Control" to activate EtherCAT, as shown in the figure below.



2. Right-click EtherCAT → Add Device → select SH-RTU-ETC and double-click to add it to the network, as shown in the figure below.



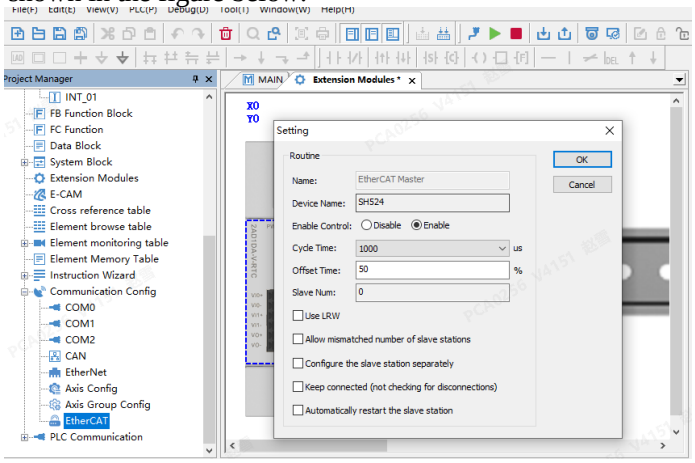
3. Double-click the added SH-RTU-ETC, select Slot Configuration in the pop-up window, and double-click to add the required modules, as shown in the figure below.



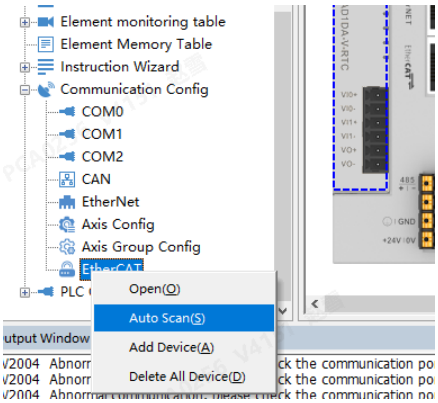
Note:

- The physical positions of connected modules must align with the configuration order; otherwise, an alarm will be generated.

(2) Method 2: In Project Manager area, double-click EtherCAT → Enable to activate EtherCAT. After completion, download the project, as shown in the figure below.



1. Right-click and select the Auto-Scan. Verify whether the scanned modules match the physical connections, then click Update Configuration to automatically add the scanned modules to the configuration, as shown in the figure below.



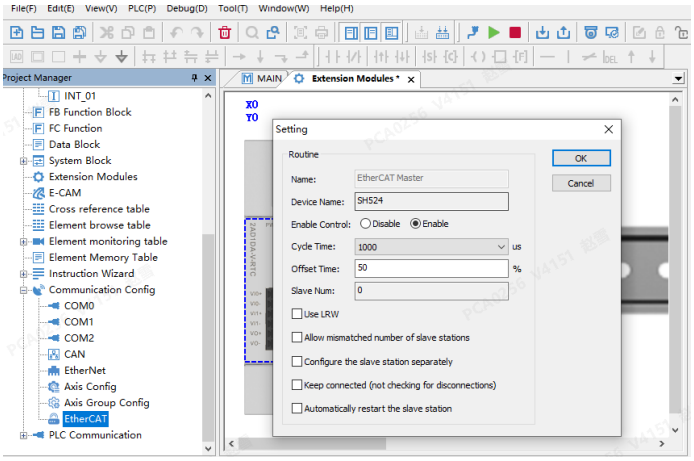
Note:

- To use the ECAT auto-scan function, ensure the XML files of the slave devices are correctly imported first. Then, check the Enable to confirm the PLC's current running program has EtherCAT configuration enabled. If Enable Control is disabled, the auto-scan function cannot be used.

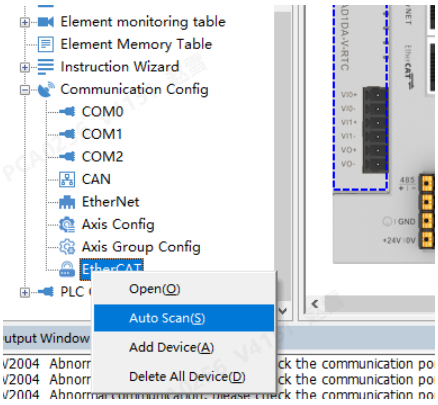
6.3.2 IO Module Configuration Example

Using SH0016ENT as an example:

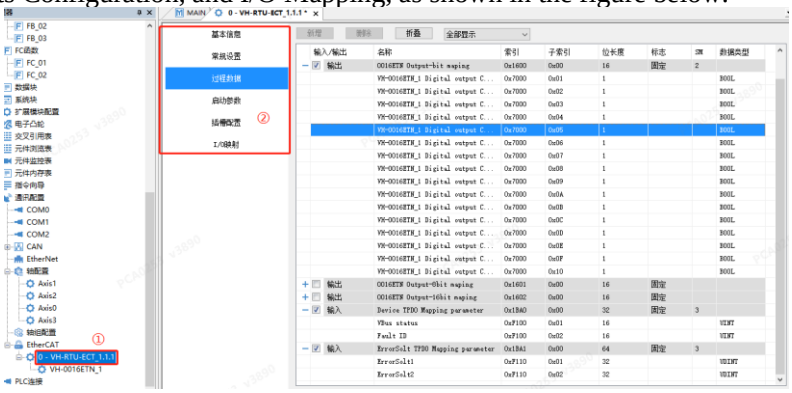
1. In Project Manager area, double-click EtherCAT and select Enable. After configuration, if the currently running PLC project does not have this function activated, download the project once, as shown in the figure below.



2. Right-click and select the Auto-Scan. Verify whether the scanned modules match the physical connections, then click Update Configuration to automatically add the scanned modules to the configuration, as shown in the figure below.



3. Double-click SH-RTU-ETC. In the pop-up window, configure parameters such as Process Data, Startup Parameter, Slots Configuration, and I/O Mapping, as shown in the figure below.

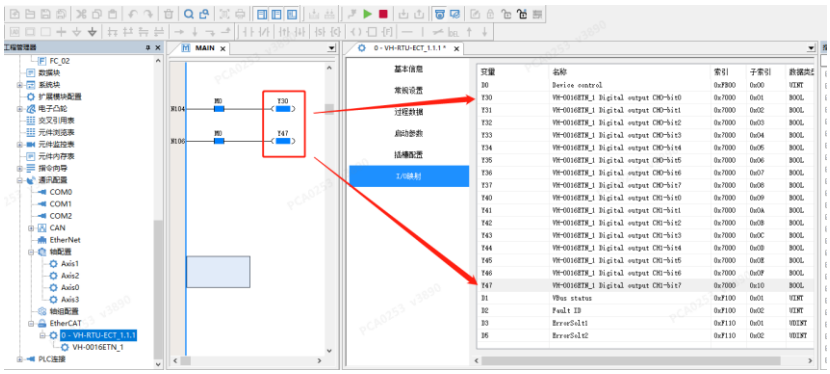


Coupler Parameter Descriptions

- Basic Information: Displays device name, manufacturer name, ID, version, etc.
- General Setting: Supports SM or DC synchronous mode. (Recommended to keep default.)
- Process Data: Configures extended I/O modules for BOOL operations or word operations, and reads the communication status and error information of the SH-RTU-ETC coupler.
- Startup Parameter: Initializes module parameters. (Recommended to keep default.)
- Slots Configuration: Supports 16 slot positions. Modules can be added to slots by double-clicking. Ensure the module configuration order matches physical connections.
- I/O Mapping: Binds module addresses to corresponding soft element addresses (supports D/R/W word elements and X/Y/M/S bit elements). Right-click Auto-Fill on the starting address to quickly bind addresses, as shown in the figure below.

4. Double-click the added module to configure the digital I/O mode as needed, as shown below:

5. After configuration, invoke the program, compile, download, and debug, as shown below:



Note:

- Configuration for other modules of the SH-RTU-ETC coupler follows the right extension modules; details are not repeated here.
- For details, please refer to the SH-RTU-ETC Coupler Manual.

7 Serial Communication

7.1 Overview

The SH controller integrates a built-in serial communication interface supporting baud rates of 9600, 19200, 38400, 57600, and 115200 bps. The serial port configurations for SH series models are listed in the table below.

Model	SH100 Series	SH300 Series		SH500 Series			
	/	SH301	SH311	SH511	SH522	SH523	SH524
Serial Port	Built-in 1×RS232 Built-in 1×RS485	Built-in 1×RS485 Expandable 2×RS485	Built-in 1×RS485 Expandable 2×RS485	Built-in 1×RS485 Expandable 2×RS485	Built-in 1×RS485 Expandable 2×RS485	Built-in 1×RS485 Expandable 2×RS485	Built-in 1×RS485 Expandable 2×RS485

7.1.1 Communication Protocol

Protocol	Description
Free protocol	Enables unrestricted data transmission/reception using XMT/RCV instructions.
Modbus-RTU master	Standard Modbus-RTU master for reading/writing data from/to slave devices via Modbus configuration.
Modbus-RTU slave	Standard Modbus-RTU slave.
N:N protocol	Proprietary protocol for the SH series, enabling data sharing between controllers.

7.1.2 Port Mapping

Host Software	Communication Channel	SH100 Series	SH300 Series	SH500 Series
COM0	0*	RS-232 port	Built-in RS-485 port	Built-in RS-485 port
COM1	1*	RS-485 port	Extension port 1A/1B	Extension port 1A/1B
COM2	2*	/	Extension port 2A/2B	Extension port 2A/2B
*: The first operand of the MODRW instruction specifies the communication channel.				

7.1.3 Serial Port Transmission Medium

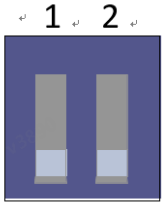
We recommend to use shielded twisted pairs on RS485 bus, and twisted paird on 485 +, 485-; The two ends of the bus are respectively connected with 120 Ω terminal resistors to prevent signal reflection; The reference ground GND of all Node 485 signals is connected together, 31 nodes max, and the distance between branches of each node should be within 3m.

Communication termination resistor DIP switch:

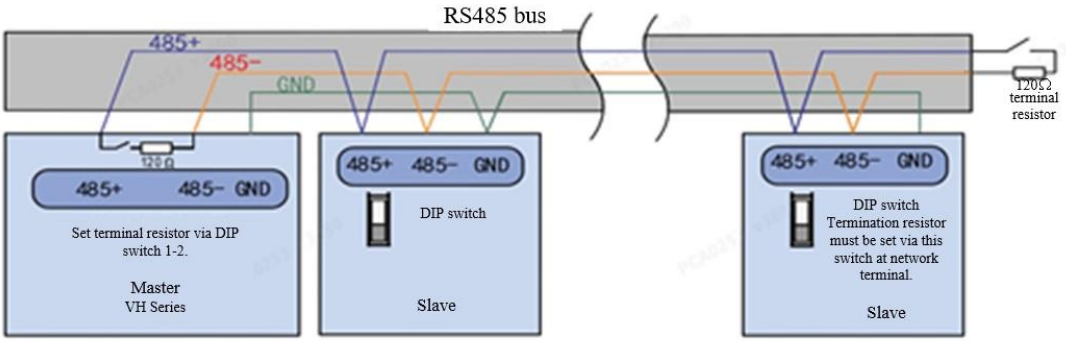
Located below the left extension interface.

ON: Resistor enabled (factory default: ON).

Switch configuration: 1 for RS485 communication; 2 for CAN communication.



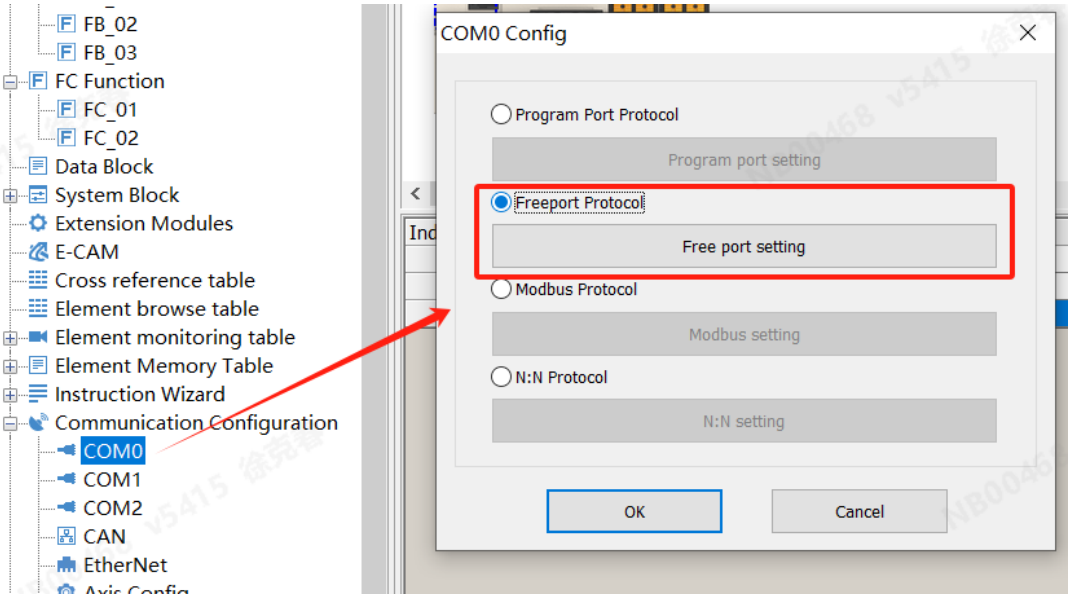
7.1.4 RS485 Serial Communication Networking



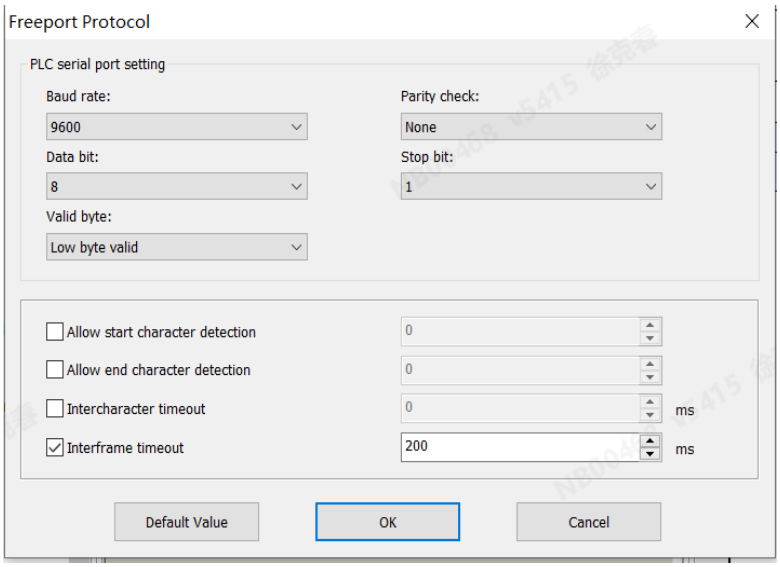
7.2 Freeport Communication

7.2.1 Freeport Protocol Configuration

1. Freeport Protocol is a user-defined communication method for custom data formats. It enables data transmission and reception via dedicated instructions and supports both ASCII and binary formats. Freeport communication is only available when the PLC is in RUN mode.
2. COM0/COM1/COM2 support the Freeport Protocol. The associated instructions include XMT (transmit) and RCV (receive).
3. Double-click "COM" in the left Project Manager area, then select "Freeport Protocol" in the dialog box, as shown below.



4. Configure the serial port settings and click "OK". The RCV and XMT instructions can then be used in the user program for data transmission/reception.



Configurable Parameters

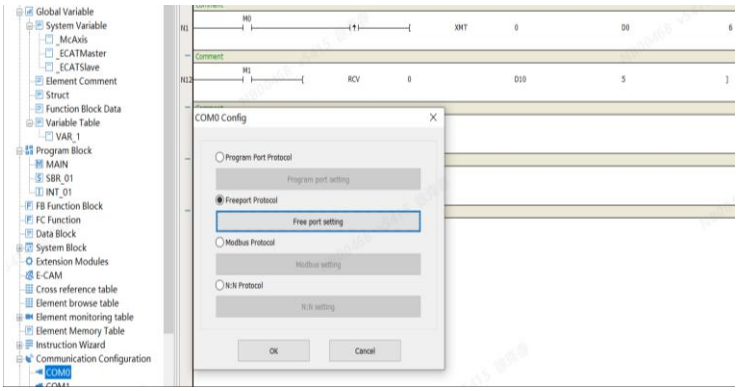
Item	Setting	Description
Baud rate	115200, 57600, 38400, 19200, 9600, 4800, 2400, 1200 (Default: 9600)	-
Data bit	7 or 8 (Default: 8)	-
Parity	None/Odd/Even (Default: None)	-
Stop bit	1 or 2 (Default: 1)	-
Valid byte	Low byte or High/Low bytes	Data processing mode: Low byte: Only the low byte is transmitted/received; high byte is discarded. High/Low bytes: Both bytes are transmitted/received.
Allow start character detection	Enable/Disable (Default: Disable)	-
Start character detection	0~255 (hex: 00~FF)	Reception starts when the specified start character is detected. The character (including the start character) is stored in the buffer.
Allow end character detection	Enable/Disable (Default: Disable)	-
End character detection	0~255 (hex: 00~FF)	Reception stops when the specified end character is detected. The end character is stored in the buffer.
Allow Intercharacter timeout	Enable/Disable (Default: Disable)	-
Intercharacter timeout	0ms~65535ms	Reception aborts if the interval between two received characters exceeds the set time.
Allow interframe timeout	Enable/Disable (Default: Disable)	-
Interframe timeout	0ms~65535ms	Starts timing when RCV is activated and communication conditions are met. Reception aborts if the frame is not fully received within the time period.

7.2.2 Program Example

Example 1: Two PLC devices are configured with COM0 in Freeport communication mode. COM0 transmits 5 bytes of data and then receives 6 bytes. Valid byte: Low byte.

Transmit data: 55778899aa

Receive data: 3489cd446688



Notes:

- Before transmitting data, ensure the receiving device has RCV enabled.
- XMT (transmit) and RCV (receive) instructions cannot be executed simultaneously. Use SM flags to implement polling control for alternating transmission and reception.
- Retrieve received data promptly to avoid overwriting by subsequent frames.

7.3 Modbus Communication Protocol

7.3.1 Overview

The SH series PLC supports the Modbus-RTU protocol via serial ports, configurable as a master or slave.

Link Characteristics

1. Physical layer: RS232 or RS485.
2. Link layer: Asynchronous transmission.
   - Data bit: 8 bits (RTU).
   - Baud rate: 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200.
   - Parity: Even, odd, or none.
   - Stop bit: 1 or 2 bits.
3. Network: Up to 31 devices, with addresses ranging from 1 to 247. Broadcast is supported.
   - RTU transmission mode.
   - Data in hexadecimal format.
   - Intercharacter interval must be <1.5 character times.
   - No frame header/trailer; interframe interval must be ≥3.5 character times.
4. CRC16 error checking.
5. RTU frame structure (max. 256 bytes):

Frame Field	Address	Funcode	Data	CRC
Bytes	1	1	0~252	2

6. Intercharacter interval calculation:

- Example (19200 baud):
  1.5 character times = (1/19200) × 11 × 1.5 × 1000 = 0.86 ms.
  3.5 character times = (1/19200) × 11 × 3.5 × 1000 = 2 ms.

7.3.2 Modbus Function Codes

(1) When the SH series PLC operates as a slave, it supports the following Modbus function codes: 01, 02, 03, 04, 05, 06, 15, 16.

Function Code (Decimal)	Name	Modbus Data Address	Operable Element Type	Description
01	Read coils	0[1]: xxxx	Y, X, M, SM, S, T, C	Read bits
02	Read discrete inputs	1[2]: xxxx	X	Read bits
03	Read registers	4[3]: xxxx[4]	D, SD, Z, T, C, R	Read words
05	Write a single coil	0: xxxx	Y, M, SM, S, T, C	Write bit
06	Write a single register	4: xxxx	D, SD, Z, T, C, R	Write word
15	Write multiple	0: xxxx	Y, M, SM, S, T, C	Write

	coils			multiple bits
16	Write multiple registers	4: xxxx	D, SD, Z, T, C, R	Write multiple words

Note:
1) "0" for coil
2) "1" for discrete input
3) "4" for register
4) xxxx for logical address range: 1~9999 (protocol addresses start from 0). Each element type has an independent logical address range (1~9999).
5) "0", "1", "4" do not have physical significance and are not involved in actual addressing.
6) Do not use function code 05 or 15 to write to X elements. Writing to X elements will not trigger errors but will be ignored by the system.

(2) Modbus Frame Format (Modbus-RTU)

i. Function Code: 0x01(01) (Read Coils)

Request frame format: slave address + 0x01 + coil start address + coil quantity + CRC parity.

No.	Data Byte	Bytes	Description
1	Slave address	1 byte	1~247 (configured in communication settings)
2	0X01 (function code)	1 byte	Read coils
3	Coil start address	2 bytes	High byte first, low byte last.
4	Coil quantity	2 bytes	High byte first, low byte last (N: Number of coils).
5	CRC parity	2 bytes	High byte first, low byte last.

Response frame format: slave address + 0x01 + byte count + coil status + CRC parity.

No.	Data Byte	Bytes	Description
1	Slave address	1 byte	1~247 (configured in communication settings)
2	0X01 (function code)	1 byte	Read coils
3	Byte count	1 byte	Value: [(N/7)/8], where N is the number of coils read.
4	Coil status	[(N/7)/8]	Each byte represents 8 coils. Unused bits in the last byte are padded with 0s. The least significant bit (LSB) corresponds to the lowest address coil.
5	CRC parity	2 bytes	High byte first, low byte last.

ii. Function Code: 0x02(02) (Read Coils)

Request frame format: slave address + 0x02 + coil start address + coil quantity + CRC parity.

No.	Data Byte	Bytes	Description
1	Slave address	1 byte	1~247 (configured in communication settings)
2	0X02 (function code)	1 byte	Read coils
3	Coil start address	2 bytes	High byte first, low byte last.
4	Coil quantity	2 bytes	High byte first, low byte last (N: Number of coils).
5	CRC parity	2 bytes	High byte first, low byte last.

Response frame format: slave address + 0x02 + byte count + coil status + CRC parity.

No.	Data Byte	Bytes	Description
1	Slave address	1 byte	1~247 (configured in communication settings)
2	0X02 (function code)	1 byte	Read coils
3	Byte count	1 byte	Value: [(N/7)/8], where N is the number of coils read.
4	Coil status	[(N/7)/8]	Each byte represents 8 coils. Unused bits in the last byte are padded with 0s. The least significant bit (LSB) corresponds to the lowest address coil.
5	CRC parity	2 bytes	High byte first, low byte last.

iii. Function Code: 0x03(03) (Read Registers)

Request frame format: slave address + 0x03 + register start address + register quantity + CRC parity.

No.	Data Byte	Bytes	Description
1	Slave address	1 byte	1~247 (configured in communication settings)
2	0X03 (function code)	1 byte	Read registers

3	Register start address	2 bytes	High byte first, low byte last.
4	Number of registers	2 bytes	High byte first, low byte last (N: Number of coils).
5	CRC parity	2 bytes	High byte first, low byte last.

Response frame format: slave address + 0x03 + byte count + register value + CRC parity.

No.	Data Byte	Bytes	Description
1	Slave address	1 byte	1~247 (configured in communication settings)
2	0X03 (function code)	1 byte	Read registers
3	Byte count	1 byte	Value: N*2
4	Register value	N*2 bytes	Each register value is represented by two bytes, with the high byte first and low byte last. Registers are arranged in ascending address order.
5	CRC parity	2 bytes	High byte first, low byte last.

iv. Function Code: 0x04(04) (Read Registers)

Request frame format: slave address + 0x04 + register start address + register quantity + CRC parity.

No.	Data Byte	Bytes	Description
1	Slave address	1 byte	1~247 (configured in communication settings)
2	0X04 (function code)	1 byte	Read registers
3	Register start address	2 bytes	High byte first, low byte last.
4	Number of registers	2 bytes	High byte first, low byte last (N: Number of coils).
5	CRC parity	2 bytes	High byte first, low byte last.

Response frame format: slave address + 0x04 + byte count + register value + CRC parity.

No.	Data Byte	Bytes	Description
1	Slave address	1 byte	1~247 (configured in communication settings)
2	0X04 (function code)	1 byte	Read registers
3	Byte count	1 byte	Value: N*2
4	Register value	N*2 bytes	Each register value is represented by two bytes, with the high byte first and low byte last. Registers are arranged in ascending address order.
5	CRC parity	2 bytes	High byte first, low byte last.

v. Function Code: 0x05(05) (Write a Single Coil)

Request frame format: slave address + 0x05 + coil address + coil status + CRC parity.

No.	Data Byte	Bytes	Description
1	Slave address	1 byte	1~247 (configured in communication settings)
2	0X05 (function code)	1 byte	Write a single coil
3	Coil address	2 bytes	High byte first, low byte last.
4	Coil status	2 bytes	The valid write values are 0xFF00 (ON, 1) or 0x0000 (OFF, 0), with the high byte first and low byte last.
5	CRC parity	2 bytes	High byte first, low byte last.

Response frame format: slave address + 0x05 + coil address + coil status + CRC parity.

No.	Data Byte	Bytes	Description
1	Slave address	1 byte	1~247 (configured in communication settings)
2	0X05 (function code)	1 byte	Write a single coil
3	Coil address	2 bytes	High byte first, low byte last.
4	Coil status	2 bytes	The valid value is 0xFF00, with the high byte first and low byte last.
5	CRC parity	2 bytes	High byte first, low byte last.

vi. Function Code: 0x06(06) (Write a Single Register)

Request frame format: slave address + 0x06 + register address + register value + CRC parity.

No.	Data Byte	Bytes	Description
1	Slave address	1 byte	1~247 (configured in communication settings)
2	0X06 (function code)	1 byte	Write a single register

3	Register address	2 bytes	High byte first, low byte last.
4	Register value	2 bytes	High byte first, low byte last.
5	CRC parity	2 bytes	High byte first, low byte last.

Response frame format: slave address + 0x06 + register address + register value + CRC parity.

No.	Data Byte	Bytes	Description
1	Slave address	1 byte	1~247 (configured in communication settings)
2	0X06 (function code)	1 byte	Write a single register
3	Register address	2 bytes	High byte first, low byte last.
4	Register value	2 bytes	High byte first, low byte last.
5	CRC parity	2 bytes	High byte first, low byte last.

vii. Function Code: 0x0F(15) (Write Multiple Coils)

Request frame format: slave address + 0x0F(15) + coil start address + coil quantity + byte count + coil status + CRC parity.

No.	Data Byte	Bytes	Description
1	Slave address	1 byte	1~247 (configured in communication settings)
2	0X0F (function code)	1 byte	Write multiple single coils
3	Coil start address	2 bytes	High byte first, low byte last.
4	Coil quantity	2 bytes	High byte first, low byte last (N: Number of coils).
5	Byte count	1 byte	Value: [(N/7)/8], where N is the number of write.
6	Coil status	[(N/7)/8]	Each byte represents 8 coils. Unused bits in the last byte are padded with 0s. The least significant bit (LSB) corresponds to the lowest address coil.
7	CRC parity	2 bytes	High byte first, low byte last.

Response frame format: slave address + 0x0F(15) + coil start address + coil quantity + CRC parity.

No.	Data Byte	Bytes	Description
1	Slave address	1 byte	1~247 (configured in communication settings)
2	0X0F (function code)	1 byte	Write multiple single coils
3	Coil start address	2 bytes	High byte first, low byte last.
4	Coil quantity	2 bytes	High byte first, low byte last.
5	CRC parity	2 bytes	High byte first, low byte last.

viii. Function Code: 0x10(16) (Write Multiple Registers)

Request frame format: slave address + 0x10(16) + register start address + register quantity + byte count + register value + CRC parity.

No.	Data Byte	Bytes	Description
1	Slave address	1 byte	1~247 (configured in communication settings)
2	0X10 (function code)	1 byte	Write multiple registers
3	Register start address	2 bytes	High byte first, low byte last.
4	Number of registers	2 bytes	High byte first, low byte last (N: Number of coils).
5	Byte count	1 byte	Value: [(N/7)/8], where N is the number of write.
6	Register value	N*2 or (N*4)	
7	CRC parity	2 bytes	High byte first, low byte last.

Response frame format: slave address + 0x10 + register start address + register quantity + CRC parity.

ix. Error Response Frame 0

Error response format: slave address + (function code + 0x80) + error code + CRC parity.

No.	Data Byte	Bytes	Description
1	Slave address	1 byte	1~247 (configured in communication settings)
2	0×80+ function code	1 byte	Error function code**
3	Error code	1 byte	
4	CRC parity	2 bytes	High byte first, low byte last.
Note: Error function code** = Requested function code＋0x80.			

Notes:

1. Only soft elements of the same type can be read in a single frame. For example, users cannot combine X and Y elements in the same frame.
2. The address range and data length for reading must not exceed the protocol-defined limits. Example:

For Y elements with protocol address range of 0000~0255, corresponding to Y0~Y377:

① Starting address = 1, element count = 256 → Returns address error (error code 02). Only 255 Y elements exist starting from address 1.

② Starting address = 0, element count = 257 → Returns data error (error code 03). The protocol defines only 256 Y elements (exceeding the maximum count).

③ Starting address = 0, element count = 256 → Returns valid status for all 256 elements. Ensure the requested elements are defined in the protocol (within the valid range). Applies to both word elements and bit elements.

7.3.3 Modbus Slave Address

(1) PLC as Modbus Slave: The correspondence between its soft elements and Modbus addresses is as follows:

Element	Type	Physical Element	Protocol Address	Supported FCs	Comment
Y	Bit	Y0~Y777 (octal) 512 bits	0000~0511	01, 05, 15	Output status, with element addresses Y0–Y7, Y10–Y17, and so on.
X	Bit	X0~X777 (octal) 512 bits	1200~01711	01, 05, 15, 02	Input status, with dual addressing supported. The element addresses are the same as above.
M	Bit	M0~M2047 M2048~M10239	2000~4047 12000~20191	01, 05, 15	
SM	Bit	SM0~SM255 SM256~SM1023	4400~4655 30000~30767	01, 05, 15	
S	Bit	S0~S1023 S1024~S4095	6000~7023 31000~34071	01, 05, 15	
T	Bit	T0~T255 T256~T511	8000~8255 11000-11255	01, 05, 15	T element status
C	Bit	C0~C255 C256~C511	9200~9455 10000~10255	01, 05, 15	C element status
D	Word	D0~D7999	0000~7999	03, 06, 16	
SD	Word	SD0~SD255 SD256~SD1023	8000~8255 12000~12767	03, 06, 16	
Z	Word	Z0~Z15	8500~8515	03, 06, 16	
T	Word	T0~T255 T256~T511	9000~9255 11000~11255	03, 06, 16	T element current value
C	Word	C0~C199	9500~9699	03, 06, 16	C element (INT) current value
C	Dual-word	C200~C255	9700~9811	03, 16	C element (DINT) current value
C	Dual-word	C256~C263	10000~10101	03, 16	C element (DINT) current value
R	Word	R0~R32767	13000~45767	03, 06, 16	

Communication Diagnostic Function Codes

The diagnostic function codes provide communication testing between the master and slave, or report internal error states of the slave. Supported function codes are listed below:

Function Code	Subfunction Code	Subfunction Name		Funcode	Subfunction Code	Subfunction Name
08	00	Return query data		08	12	Return bus communication error count
08	01	Restart communication options		08	13	Return bus exception error count
08	04	Force listen-only mode		08	14	Return slave message count
08	10	Clear counters		08	15	Return slave no-response count
08	11	Return bus message count		08	18	Return bus character overrun count

Error Code

In a normal response, the slave returns data or statistic in the data field. In an error response, the slave returns an error code as follows:

Error Code	Description
0x01	Illegal function code
0x02	Illegal register address
0x03	Illegal data

No-response scenarios:

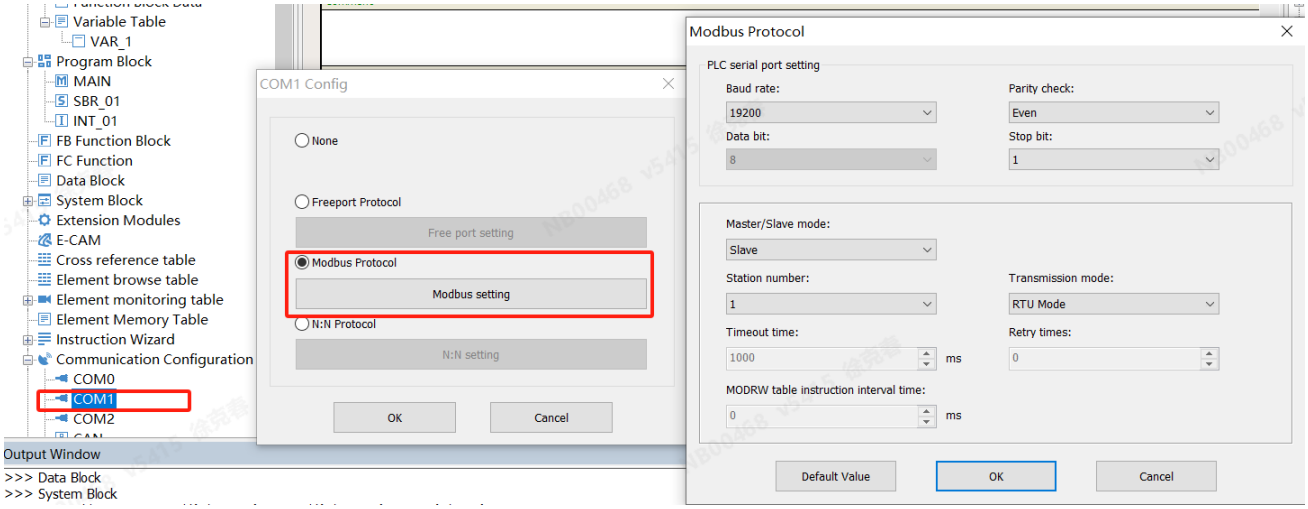
- Errors in broadcast frames (e.g., data/address errors).
- Character length exceeds protocol limit (e.g., RTU frames >256 bytes).
- Intercharacter timeout in RTU mode (treated as invalid frame).
- Slave in listen-only mode.
- Invalid ASCII frames (e.g., incorrect end characters, invalid character ranges).

7.3.4 Modbus Slave Communication Configuration

When the PLC operates as a Modbus slave, it does not initiate communication and responds to the master only upon receiving a locally addressed frame. The slave supports Modbus function codes 01, 02, 03, 05, 06, 08, 15, 16; all others (except broadcast frames) return an Illegal Function Code.

Slave Configuration via Software

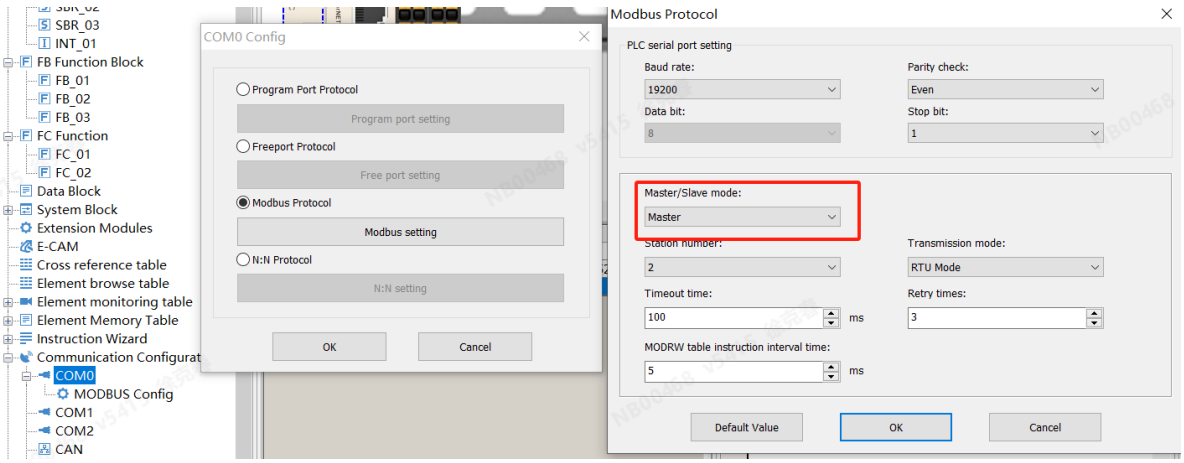
1. Navigate to Project Manager > Communication Config, double-click COM0 to access the configuration interface. Set parameters such as communication protocol, station number, and baud rate, then click OK to save settings.



7.3.5 Modbus Master Communication Configuration

Maste Configuration via Software

1. Navigate to Project Manager > Communication Config, double-click COM0 to access the configuration interface. Set parameters such as communication protocol, station number, and baud rate, then click OK to save settings.



Parameters Description

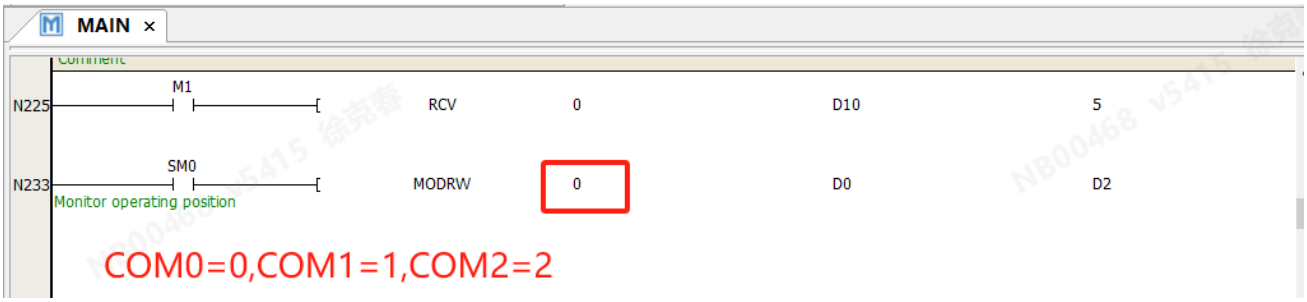
Item	Settings
Station number	0~247
Baud rate	115200, 57600, 38400, 19200, 9600, 4800, 2400, 1200
Data bit	7 or 8 (7 for ASCII mode, 8 for RTU mode)
Parity check	None/Odd/Even
Stop bit	1 or 2 (1 if parity enabled, 2 if parity disabled)
Master/Slave mode	Configure as Master or Slave
Transmission mode	RTU or ASCII
Timeout time	Time limit for master waiting for slave response
MODRW table instruction interval time	Minimum delay between sending the next instruction after completion of the previous one.

Note:

- When the PLC is configured as a Master, its station number must differ from slave numbers to avoid communication conflicts.
- Configuration parameters set in the system block take effect only after a full PLC runtime cycle.

7.3.6 MODRW Instruction Description

1. When the PLC operates as a Modbus master, use the MODRW/Modbus instruction (provided by the system) to communicate with slave devices.



2. Example program: Write value 200 to address 0 of Station 1, read data from address 0 of Station 1 and store it in register D18, and then write value 16#00FF to address 0 of Station 1.

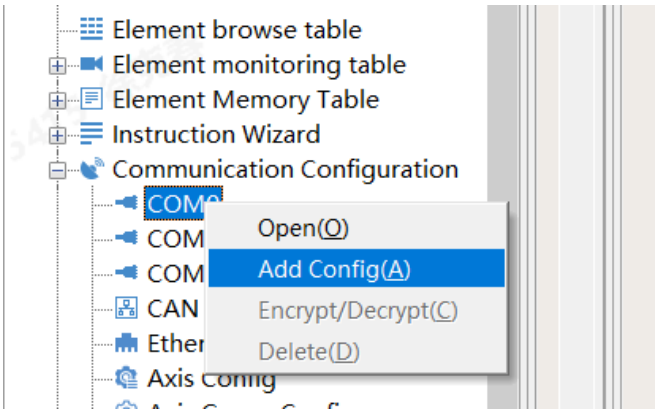
Key Features:

- MODRW can be invoked to send/receive data to/from slaves when COM0/COM1/COM2 is configured as the master.
- Supports up to 255 concurrent MODRW instructions.
- Instructions are executed sequentially per scan cycle.
- The instruction continuously executes as long as the enabling condition (EN) is active.
- Channel parameter: COM0=0, COM1=1, COM2=2.
- Monitor communication faults via SM/SD registers of the corresponding port.

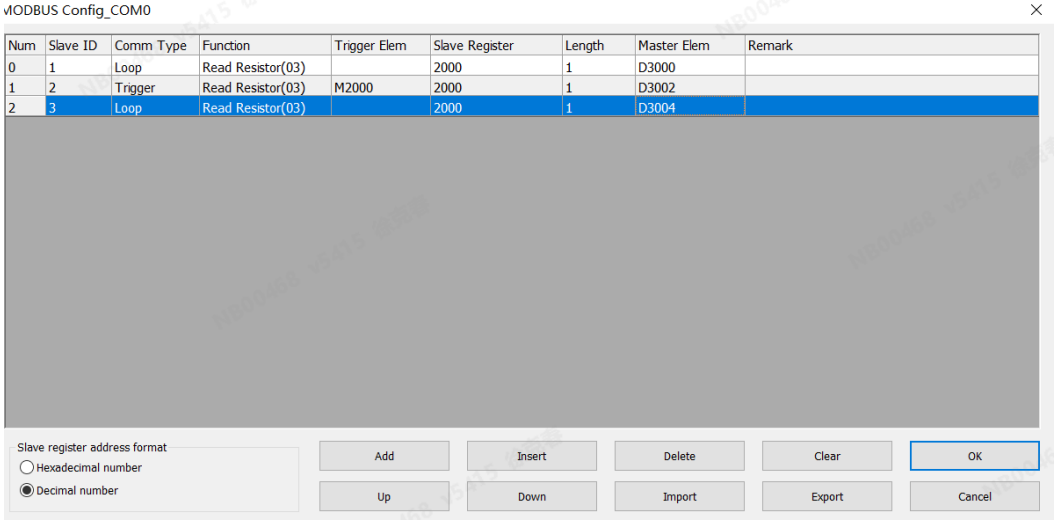
7.3.7Modbus Configuration Table

When the PLC is configured as a Modbus master, communication with slaves can be managed through the Modbus configuration table, eliminating the need for explicit instruction calls in the program. Configuration steps:

1. Navigate to Project Manager → Communication Config → COM0, and right-click "Add Config", as shown below.



2. Double-click "MODBUS Config". In the dialog box, set parameters including slave ID, communication type, function code, read/write slave address, length, and remarks, as illustrated below.



Modbus Configuration Table Description

Item	Setting	Description
Slave ID	Supports 31 slaves	Station ID range: 1~255.
Communication Type	Loop mode	PLC repeatedly executes all "Loop" configurations during program scan.
	Trigger mode	Triggers communication when the trigger flag is set in the program. Each flag set initiates one communication. Use a timer to periodically set the flag for desired frequency.
Function	Supported function codes: 01, 02, 03, 04, 05, 06, 15, 16.	**Write register (16):** Uses FC06 (Data Length=1) or FC16 (Data Length>1). **Write coil (15):** Uses FC05 (Data Length=1) or FC16 (Data Length>1).
Trigger Element	M0~M10247 elements can be selected as trigger flags.	In Trigger mode, communication is initiated when the flag (e.g., M flag) is ON. Upon successful completion, the system automatically clears this trigger flag, allowing the M flag to also serve as an indicator of successful communication. When configuring the communication table, avoid reusing the same M flag for multiple entries, as the system's clearance operation may interfere with other communication operations.
Slave Register	Slave communication address register	Addresses can be displayed in decimal or hexadecimal.
Length	Length of communication addresses (max. 120 INT-type registers)	Specifies the number of communication register addresses.
Master Element	Master send/receive addresses (supports D and R	Specifies the address for data transmission or reception.

	elements).	
Max. Table Rows	Supports up to 512 entries.	
Remark	-	Describes register/element addresses.

Notes:

Configure parameters based on refresh requirements to optimize communication performance. Avoid setting all entries to Loop mode, as excessive loop entries may reduce responsiveness. Use Trigger mode for non-critical data to improve real-time performance. For RS485 Modbus (common rate: 9600bps), limit loop entries to ≤10 and trigger entries to ~10 per second for optimal performance.

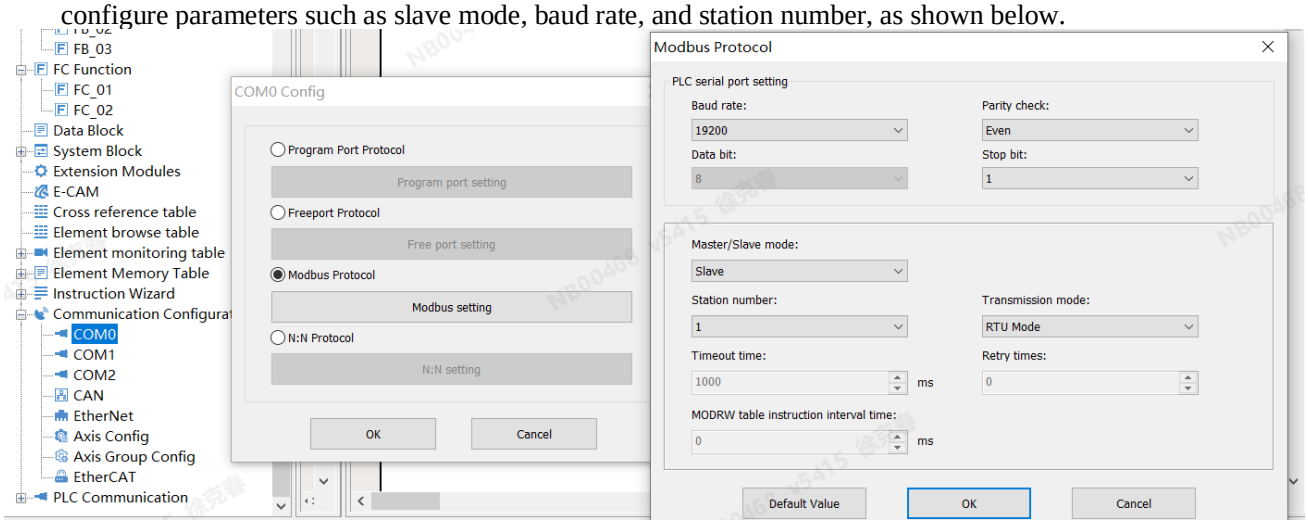
7.3.8 Modbus-RTU Communication Example

Program Requirements:

Establish serial communication between two devices (SH311 and SH523) using the Modbus-RTU protocol. The master PLC reads the value of register D100 from the slave PLC.

Slave Configuration:

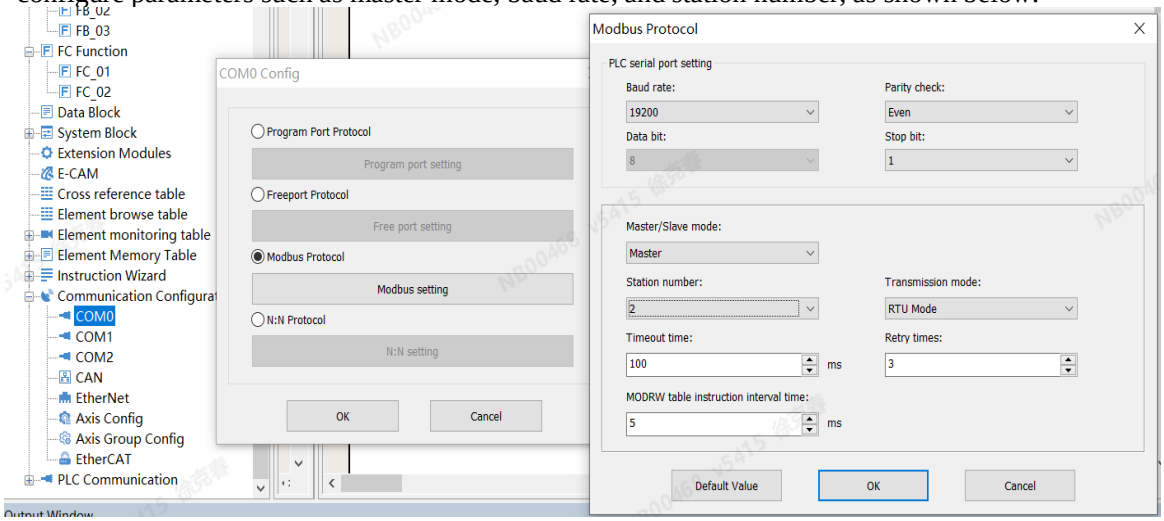
1. Navigate to Project Manager → Communication Config, double-click "COM0", select Modbus setting, and configure parameters such as slave mode, baud rate, and station number, as shown below.



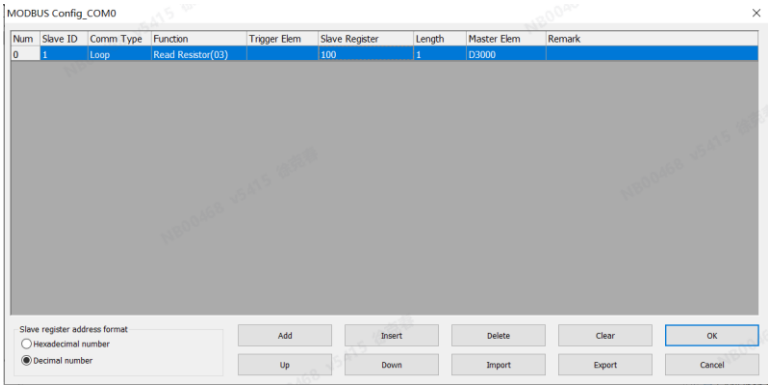
After configuration, download the project to the PLC.

Master Configuration:

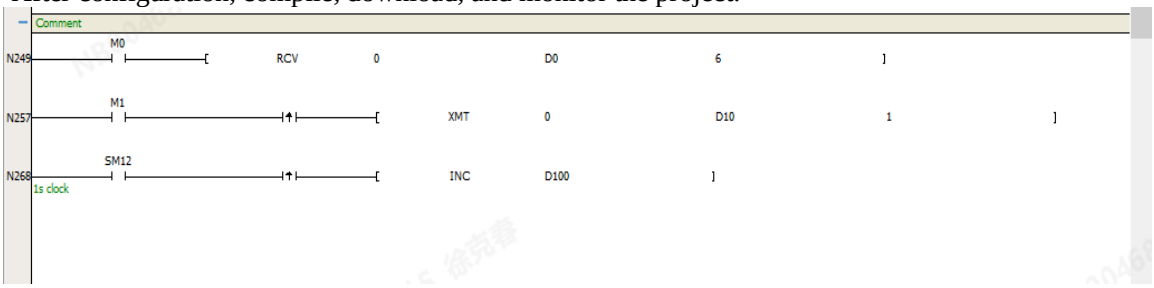
1. Navigate to Project Manager → Communication Config, double-click "COM0", select Modbus setting, and configure parameters such as master mode, baud rate, and station number, as shown below.



2. Right-click "COM0", select "Add Config", and then double-click "MODBUS Config". In the dialog box, set parameters including slave ID, communication type, function code, read/write slave address, length, and remarks, as shown below.



3. After configuration, compile, download, and monitor the project.



Program Description: The master reads the value of register D100 from the slave PLC and stores it in D3000.

7.3.9 Slave Address Modification

When Modbus master/slave protocol is configured for COM0/COM1/COM2, the communication station number can be modified via SD registers, and changes take effect immediately.

Port	Register	Effective Mechanism
COM0	SD109	Take effect immediately
COM1	SD129	Take effect immediately
COM2	SD149	Take effect immediately

7.4 N:N Communication Protocol

7.4.1 Overview

N:N is a compact PLC network developed by Suzhou SLANVERT Electric Co., Ltd. It operates on the RS485 physical layer, enabling direct PLC connection via Communication Port 1. PLCs in the N:N network automatically exchange values of D registers and M registers, allowing peer-to-peer access to network PLC registers as if accessing local registers.

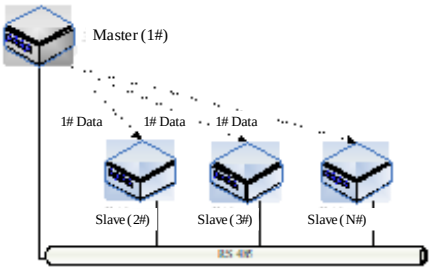
Key Features: Most parameters only require configuration on the #0 PLC; Online parameter modification is supported; Automatically detects newly added PLCs in the network; If a PLC disconnects, other PLCs continue data exchange uninterrupted; Communication status can be monitored via SM registers on any PLC in the network.

7.4.2 N:N Network Data Transmission

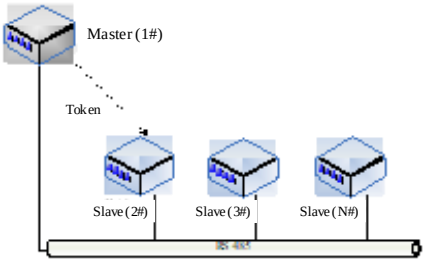
The N:N network employs two types of messages: token distribution from the master and data broadcasts from individual PLCs.

Token Distribution: The master initially holds the token. After broadcasting data, the master sequentially passes the token to slaves in a cyclic manner. Only the slave holding the token can broadcast data to other PLCs (including the master).

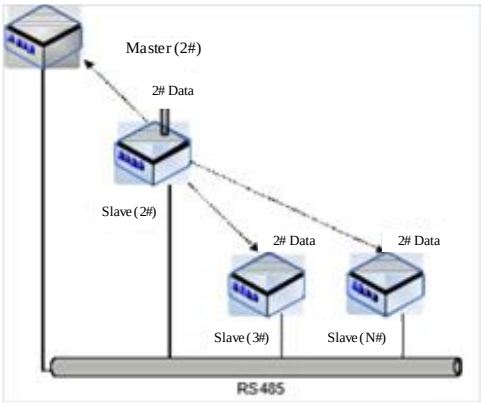
Figures 10-1 to 10-5 illustrate the primary processes of network communication. In these diagrams, Station 1# serves as the master. Note that by default, Station 0# is designated as the master, while Station 1# acts as a backup master (automatically activated if the primary master experiences communication failure or power loss).



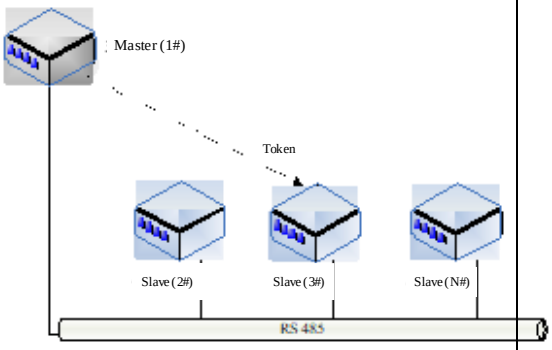
Master broadcasts



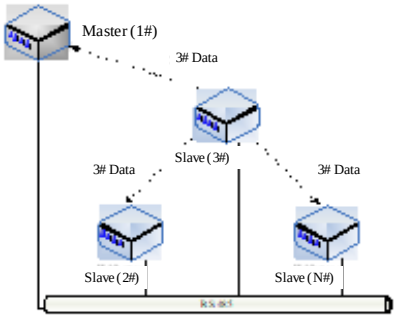
Master passes the token to 2# Slave



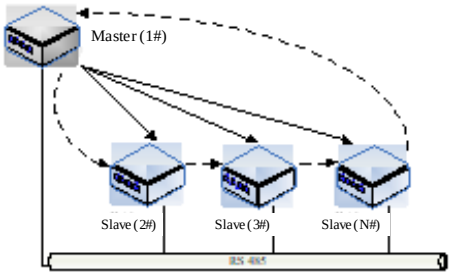
2# Slave broadcasts



Master passes the token to 3# Slave



3# Slave broadcasts

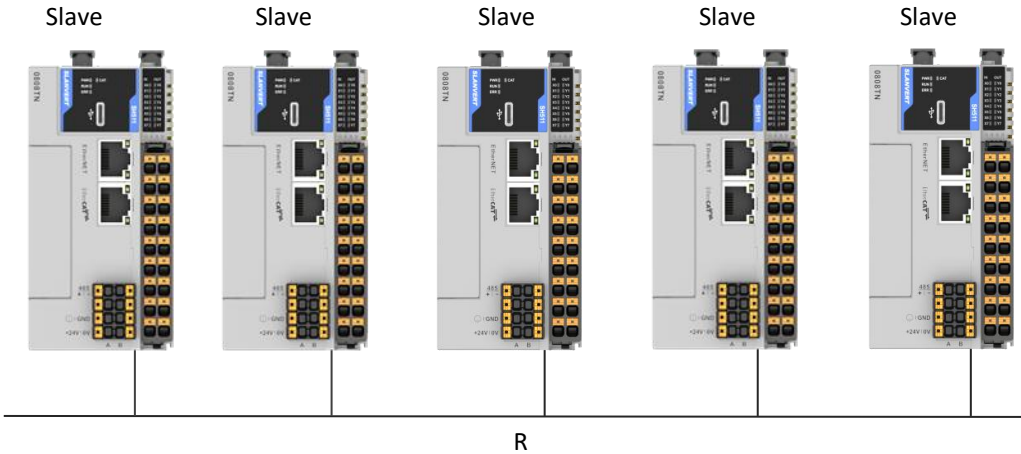


Token passing sequence

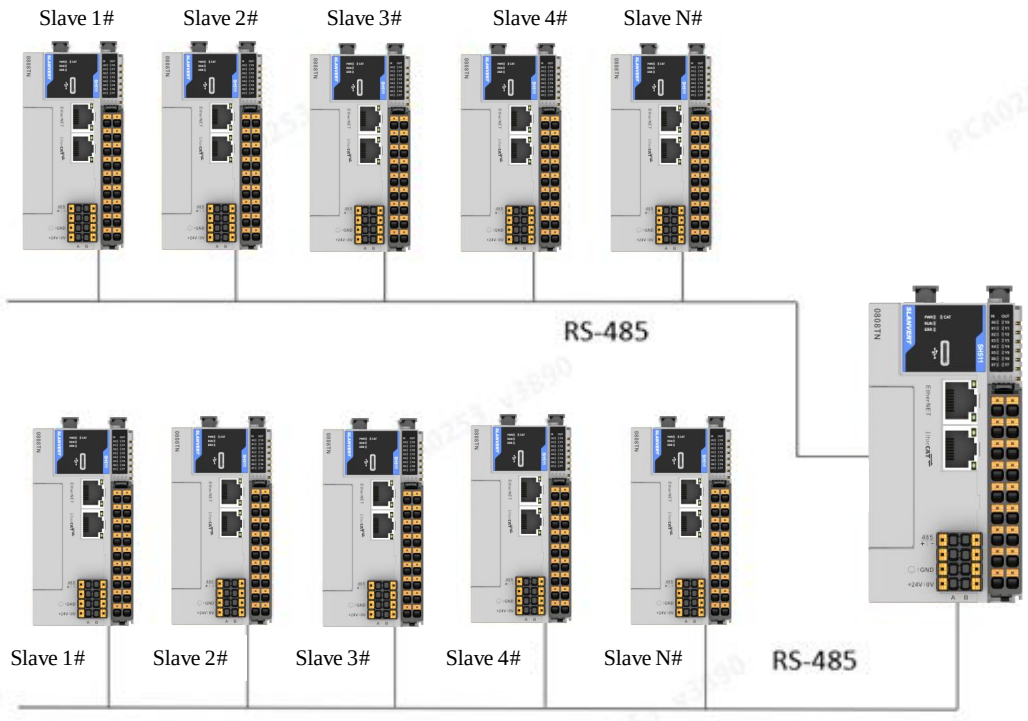
Figure 10-6 depicts the token-passing sequence. The solid line indicates the actual token passing path, while the dashed line represents the order of stations holding and broadcasting the token. Note that the token is not transferred directly between slaves (e.g., from 2# PLC to 3# PLC). Instead, the master issues the token to 2# PLC first, then subsequently to 3# PLC.

7.4.3 N:N Network Architecture

The N:N network supports two types of topologies: single-layer and multi-layer, as illustrated below:



Single-layer Network

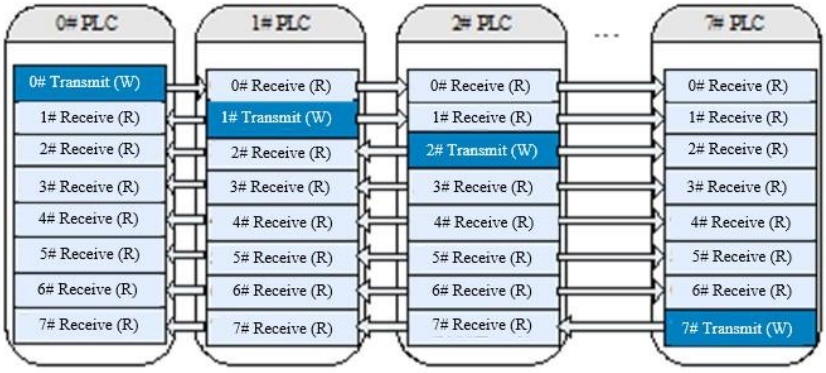


Single-layer Network

In a single-layer network, each PLC connects to the N:N network through one communication port, supporting a maximum of 32 PLCs per layer. In a multi-layer network, intermediate node PLCs act as inter-layer bridges, with their two communication ports linked to different layers. Each layer in a multi-layer configuration supports up to 16 PLCs.

7.4.4 N:N Refresh Mode

PLCs in the N:N network automatically exchange values of D registers and M registers within a fixed-range "shared register area". Once N:N is enabled, these shared registers are continuously refreshed to ensure consistency across all PLCs.



In "shared register area", each PLC has a writable transmit area, and N:N automatically broadcasts the content of this transmit area (specific D/M register values) to other PLCs. Simultaneously, each PLC receives data from other PLCs and stores it in a read-only receive area.

The shared component area has a fixed capacity (64 D registers and 512 M registers available for sharing), which are distributed across connected PLCs. The fewer PLCs in the network, the more registers each PLC is allocated. This mapping relationship is defined by the N:N Refresh Mode Table.

D Register Allocation in N:N Single-Layer Network

D Register Allocation	Mode 1	Mode 2	Mode 3	Mode 4	Mode 5
D7700~D7701	#0	#0	#0	#0	#0
D7702~D7703	#1				
D7704~D7705	#2	#1			
D7706~D7707	#3				
D7708~D7709	#4	#2	#1		
D7710~D7711	#5				
D7712~D7713	#6	#3			

D Register Allocation	Mode 1	Mode 2	Mode 3	Mode 4	Mode 5
D7714~D7715	#7				
D7716~D7717	#8	#4	#2	#1	
D7718~D7719	#9				
D7720~D7721	#10	#5			
D7722~D7723	#11				
D7724~D7725	#12	#6	#3		
D7726~D7727	#13				
D7728~D7729	#14	#7			
D7730~D7731	#15				
D7732~D7733	#16	#8	#4	#2	#1
D7734~D7735	#17				
D7736~D7737	#18	#9			
D7738~D7739	#19				
D7740~D7741	#20	#10	#5		
D7742~D7743	#21				
D7744~D7745	#22	#11			
D7746~D7747	#23				
D7748~D7749	#24	#12	#6	#3	
D7750~D7751	#25				
D7752~D7753	#26	#13			
D7754~D7755	#27				
D7756~D7757	#28	#14	#7		
D7758~D7759	#29				
D7760~D7761	#30	#15			
D7762~D7763	#31				

Example:

(10) In Mode 1, Station 0# is allocated D registers D7700~D7701. The PLC at Station 0# can write values to D7700 and D7701, while other stations (1#~31#) can directly read values from these registers.

(11) In Mode 2, Station 0# is allocated D registers D7700~D7703. The PLC at Station 0# can write values to D7700~D7703, while other stations (1#~15#) can directly read values from these registers.

M Register Allocation in N:N Single-Layer Network

M Register Allocation	Mode 1	Mode 2	Mode 3	Mode 4	Mode 5
M1400~M1415	#0	#0	#0	#0	#0
M1416~M1431	#1				
M1432~M1447	#2	#1			
M1448~M1463	#3				
M1464~M1479	#4	#2	#1		
M1480~M1495	#5				
M1496~M1511	#6	#3			
M1512~M1527	#7				
M1528~M1543	#8	#4	#2	#1	
M1544~M1559	#9				
M1560~M1575	#10	#5			
M1576~M1591	#11				
M1592~M1607	#12	#6	#3		#0
M1608~M1623	#13				
M1624~M1639	#14	#7			
M1640~M1655	#15				
M1656~M1671	#16	#8	#4	#2	#1
M1672~M1687	#17				
M1688~M1703	#18	#9			
M1704~M1719	#19				
M1720~M1735	#20	#10	#5		
M1736~M1751	#21				
M1752~M1767	#22	#11			
M1768~M1783	#23				
M1784~M1799	#24	#12	#6	#3	
M1800~M1815	#25				
M1816~M1831	#26	#13			

M Register Allocation	Mode 1	Mode 2	Mode 3	Mode 4	Mode 5
M1832~M1847	#27				
M1848~M1863	#28	#14	#7		
M1864~M1879	#29				
M1880~M1895	#30	#15			
M1896~M1911	#31				

Note:

(12) In Mode 1, Station 0# is allocated M registers M1400~M1415. The PLC at Station 0# can write values to M1400~M1415, while other stations (1#~31#) can directly read values from these registers.

(13) In Mode 2, Station 0# is allocated M registers M1400~M1431. The PLC at Station 0# can write values to M1400~M1431, while other stations (1#~31#) can directly read values from these registers.

D Register Allocation in N:N Multiple-Layer Network (Layer 0)

D Register Allocation	Mode 6	Mode 7	Mode 8	Mode 9
D7700~D7701	#0	#0	#0	#0
D7702~D7703	#1			
D7704~D7705	#2	#1		
D7706~D7707	#3			
D7708~D7709	#4	#2	#1	
D7710~D7711	#5			
D7712~D7713	#6	#3		
D7714~D7715	#7			
D7716~D7717	#8	#4	#2	#1
D7718~D7719	#9			
D7720~D7721	#10	#5		
D7722~D7723	#11			
D7724~D7725	#12	#6	#3	
D7726~D7727	#13			
D7728~D7729	#14	#7		
D7730~D7731	#15			

Note:

In Mode 6, Station 0# (Layer 0) is allocated D registers D7700~D7701. The PLC at Station 0# can write values to D7700~D7701, while other stations (1#~15#) can directly read values from these registers.

D Register Allocation in N:N Multiple-Layer Network (Layer 1)

D Register Allocation	Mode 10	Mode 11	Mode 12	Mode 13
D7732~D7733	#0	#0	#0	#0
D7734~D7735	#1			
D7736~D7737	#2	#1		
D7738~D7739	#3			
D7740~D7741	#4	#2	#1	
D7742~D7743	#5			
D7744~D7745	#6	#3		
D7746~D7747	#7			
D7748~D7749	#8	#4	#2	#1
D7750~D7751	#9			
D7752~D7753	#10	#5		
D7754~D7755	#11			
D7756~D7757	#12	#6	#3	
D7758~D7759	#13			
D7760~D7761	#14	#7		
D7762~D7763	#15			

Note:

In Mode 10, Station 0# (Layer 1) is allocated D registers D7732~D7733. The PLC at Station 0# can write values to D7732~D7733, while other stations (1#~15#) can directly read values from these registers.

M Register Allocation in N:N Multiple-Layer Network (Layer 0)

M Register Allocation	Mode 6	Mode 7	Mode 8	Mode 9
M1400~M1415	#0	#0	#0	#0
M1416~M1431	#1			
M1432~M1447	#2	#1		
M1448~M1463	#3			
M1464~M1479	#4	#2	#1	

M Register Allocation	Mode 6	Mode 7	Mode 8	Mode 9
M1480~M1495	#5			
M1496~M1511	#6	#3		
M1512~M1527	#7			
M1528~M1543	#8	#4	#2	#1
M1544~M1559	#9			
M1560~M1575	#10	#5		
M1576~M1591	#11			
M1592~M1607	#12	#6	#3	
M1608~M1623	#13			
M1624~M1639	#14	#7		
M1640~M1655	#15			

Note:

In Mode 6, Station 0# (Layer 0) is allocated M registers M1400~M1415. The PLC at Station 0# can write values to M1400~M1415, while other stations (1#~15#) can directly read values from these registers.

M Register Allocation in N:N Multiple-Layer Network (Layer 1)

M Register Allocation	Mode 10	Mode 11	Mode 12	Mode 13
M1656~M1671	#0	#0	#0	#0
M1672~M1687	#1			
M1688~M1703	#2	#1		
M1704~M1719	#3			
M1720~M1735	#4	#2	#1	
M1736~M1751	#5			
M1752~M1767	#6	#3		
M1768~M1783	#7			
M1784~M1799	#8	#4	#2	#1
M1800~M1815	#9			
M1816~M1831	#10	#5		
M1832~M1847	#11			
M1848~M1863	#12	#6	#3	
M1864~M1879	#13			
M1880~M1895	#14	#7		
M1896~M1911	#15			

Note:

In Mode 10, Station 0# (Layer 1) is allocated M registers M1656~M1671. The PLC at Station 0# can write values to M1656~M1671, while other stations (1#~15#) can directly read values from these registers.

CAUTION

- When the N:N communication protocol is configured, D registers D7700~D7763 and M registers M1400~M1911 are reserved as shared resources for network data exchange. Exercise caution when using these registers in your program!

7.4.5 Enhanced Refresh Modes

The SH-series PLCs offer Modes 14~18 to support expanded shared registers, applicable only to single-layer networks requiring extensive data exchange. The M and D register ranges are extended to M1400~M1911 and D7500~D7755, respectively.

M register allocation (512 registers) is shown in the table below:

M Register Allocation	Mode 14	Mode 15	Mode 16	Mode 17	Mode 18
M1400~M1415	#0	#0	#0	#0	#0
M1416~M1431	#1				
M1432~M1447	#2	#1			
M1448~M1463	#3				
M1464~M1479	#4	#2	#1		
M1480~M1495	#5				
M1496~M1511	#6	#3			
M1512~M1527	#7				
M1528~M1543	#8	#4	#2	#1	
M1544~M1559	#9				
M1560~M1575	#10	#5			
M1576~M1591	#11				
M1592~M1607	#12	#6	#3		
M1608~M1623	#13				
M1624~M1639	#14	#7			
M1640~M1655	#15				
M1656~M1671	#16	#8	#4	#2	#1
M1672~M1687	#17				
M1688~M1703	#18	#9			
M1704~M1719	#19				
M1720~M1735	#20	#10	#5		
M1736~M1751	#21				
M1752~M1767	#22	#11			
M1768~M1783	#23				
M1784~M1799	#24	#12	#6	#3	
M1800~M1815	#25				
M1816~M1831	#26	#13			
M1832~M1847	#27				
M1848~M1863	#28	#14	#7		
M1864~M1879	#29				
M1880~M1895	#30	#15			
M1896~M1911	#31				

D register allocation (256 registers) is shown in the table below:

D Register Allocation	Mode 14	Mode 15	Mode 16	Mode 17	Mode 18
D7500~D7507	#0	#0	#0	#0	#0
D7508~D7515	#1				
D7516~D7523	#2	#1			
D7524~D7531	#3				
D7532~D7539	#4	#2	#1		
D7540~D7547	#5				
D7548~D7555	#6	#3			
D7556~D7563	#7				
D7564~D7571	#8	#4	#2	#1	
D7572~D7579	#9				
D7580~D7587	#10	#5			
D7588~D7595	#11				
D7596~D7603	#12	#6	#3		
D7604~D7611	#13				
D7612~D7619	#14	#7			
D7620~D7627	#15				
D7628~D7635	#16	#8	#4	#2	#1
D7636~D7643	#17				
D7644~D7651	#18	#9			
D7652~D7659	#19				
D7660~D7667	#20	#10	#5		
D7668~D7675	#21				
D7676~D7683	#22	#11			
D7684~D7691	#23				
D7692~D7699	#24	#12	#6	#3	
D7700~D7707	#25				
D7708~D7715	#26	#13			
D7716~D7723	#27				
D7724~D7731	#28	#14	#7		
D7732~D7739	#29				
D7740~D7747	#30	#15			
D7748~D7755	#31				

N:N Control Strategy

Master Determination

Station 0 is the default master. Only Station 0 can initialize and start the network. N:N parameters (refresh mode, delay time, retry count, etc.) can only be configured via Station 0#. During online configuration updates or system block downloads on Station 0#, the backup master temporarily assumes control. After completion, Station 0# reclaims master status.

Master selection rule: The lowest station number in the network acts as the master.

Maximum Polling Station Count

Set this value equal to the actual number of PLCs in the network, with stations numbered sequentially from 0#. If the set value (N) is less than the actual PLC count, stations with numbers ≥ N cannot broadcast data but can receive broadcasts from stations < N.

Multi-Master-Slave (M:N)

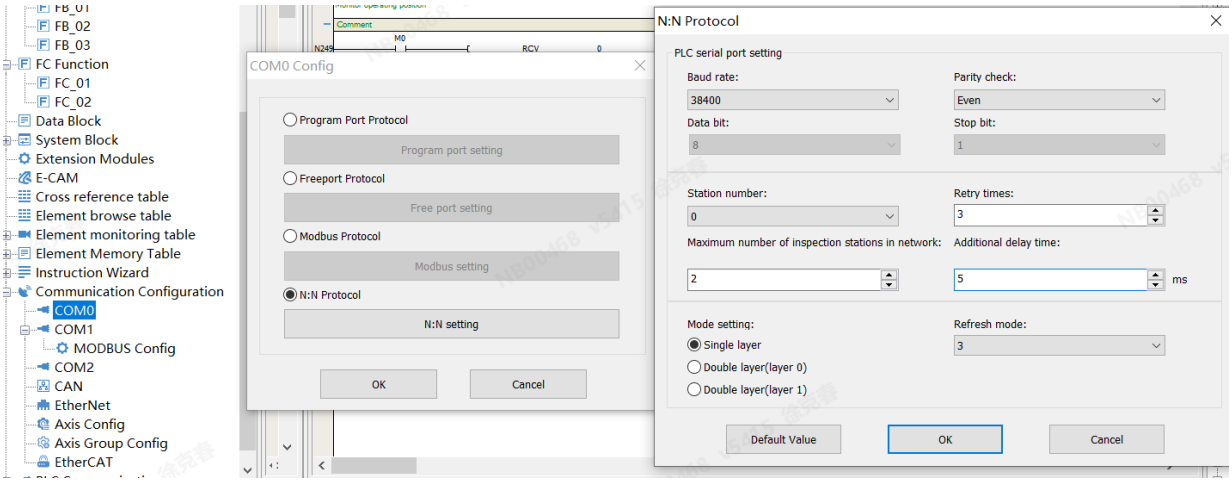
N:N supports multi-master-slave networks. Master here refers to PLCs that can both write their own M/D registers and read others', while slave refers to PLCs that can only read others' M/D registers. Within the configured maximum polling station count (limited by refresh mode), stations <N act as masters, while stations ≥N are slaves. Slaves access master registers based on the masters' refresh modes and mappings defined in the N:N Shared M/D Register Table. Slaves have no corresponding entries in these tables.

7.4.6 N:N Protocol Usage Example

Two PLCs communicating via N:N protocol in a single-layer network configured to Mode 3.

Master Configuration

1. Navigate to Project Manager → Communication Config, double-click COM0, and select N:N Protocol. Double-click the N:N setting, and set communication parameters as shown below:

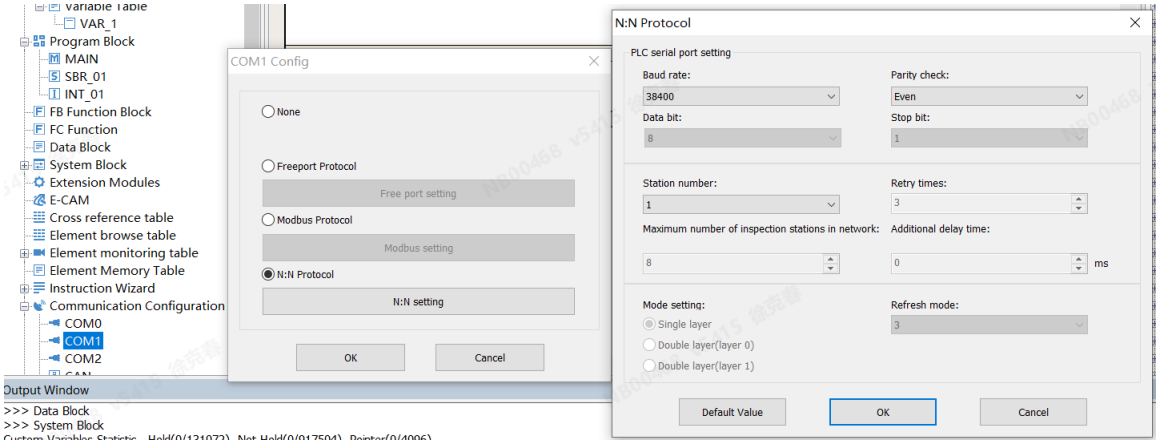


Parameters Description

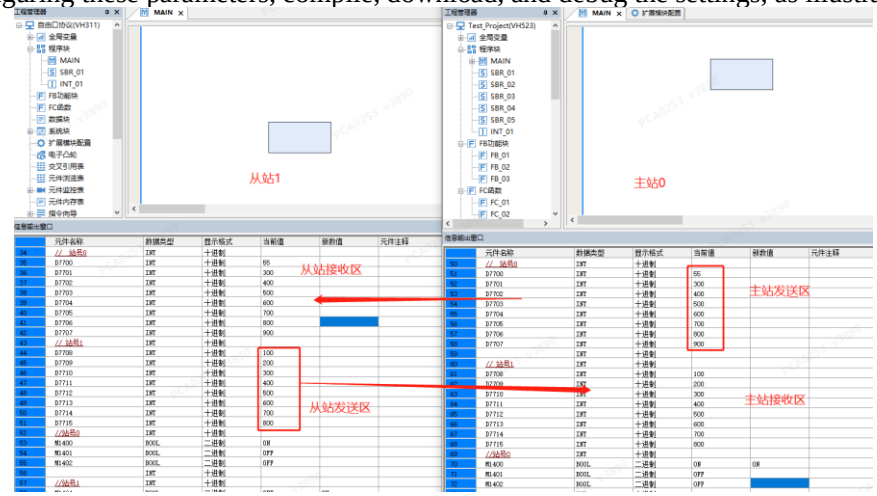
- Station number: Assign unique sequential station numbers starting from 0. Station 0# initiates and configures the network.
- Maximum polling station count: Total number of PLCs in the network (up to 32). Configure this parameter, along with additional delay time, retry times, and mode, only on Station 0#.
- Refresh mode: 1~18 modes are available, determining the shared ranges of D and M elements.
- For a current network with 6 PLCs, set the maximum polling station count to 6 and assign station numbers 0~5. To allow future extension without interrupting the network (e.g., adding 2 PLCs), set the count to 8. New PLCs assigned station numbers 6 and 7 will be automatically detected and integrated into data exchange with stations 0~5 within 1 second.

Slave Configuration

2. For stations other than Station 0#, set only their station number, ensuring baud rate and parity match Station 0#.



3. After configuring these parameters, compile, download, and debug the settings, as illustrated below:



4. Program Description:

Master Station 0#: Inputs data into register D7700~D7707 in transmit area, and Slave Station 1# receives from D7700~D7707.

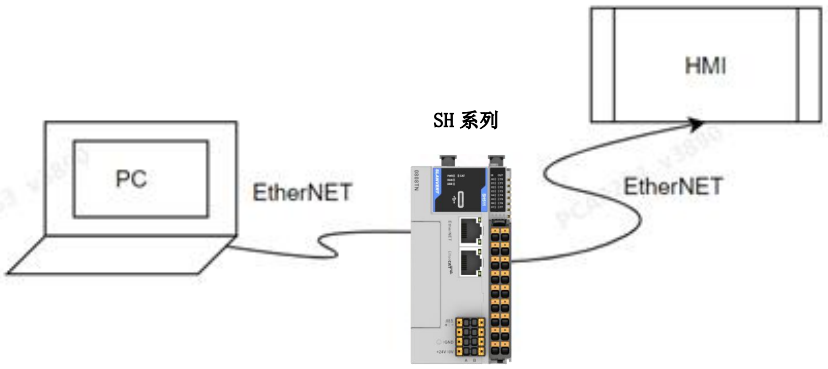
Slave Station 1#: Inputs data into register D7708~D7715 in transmit area, and Master Station 0# receives from

D7708~D7715.

8 Ethernet Communication

8.1 Overview

The SH300 and SH500 series integrate the ModbusTCP protocol (server and client), enabling seamless communication and data exchange with ModbusTCP-compatible devices. For devices not supporting ModbusTCP, socket instructions are provided to implement custom TCP/UDP-based application protocols. AutoSoft supports Ethernet-based monitoring, downloading, uploading, and debugging of PLCs with high efficiency.



8.2 Hardware Interface Specifications

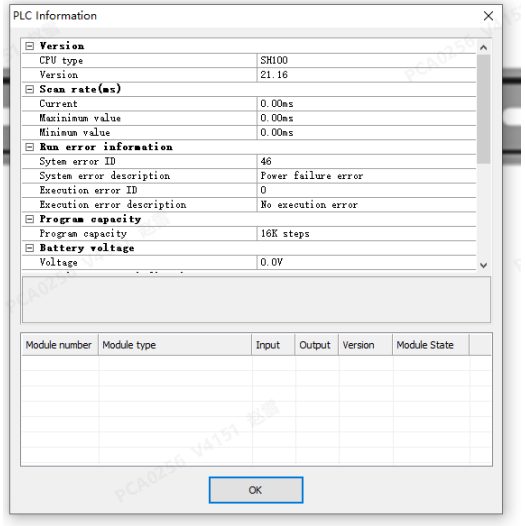
Item	Ethernet Interface
Transmission speed	10Mbps: 10BASE-T 100Mbps: 100BASE-TX 10Mbps/100Mbps auto-tuning
Modulation	Baseband
Topology	Star
Transmission medium	Shielded twisted pair cables of Category 5 and above with aluminum foil and braided mesh
Transmission distance	Node-to-node distance: ≤100 m
Connection capacity	Supports up to 16 slaves and 4 masters

8.3 IP Address Configuration/Viewing

The SH series has a factory default IP address of "192.168.1.10". The IP address can be modified via the AutoSoft software when connected via USB or Ethernet.

IP Address Viewing

1. With a USB or Ethernet connection established, view the IP address by navigating to the menu bar -> PLC -> PLC Information, as shown below.



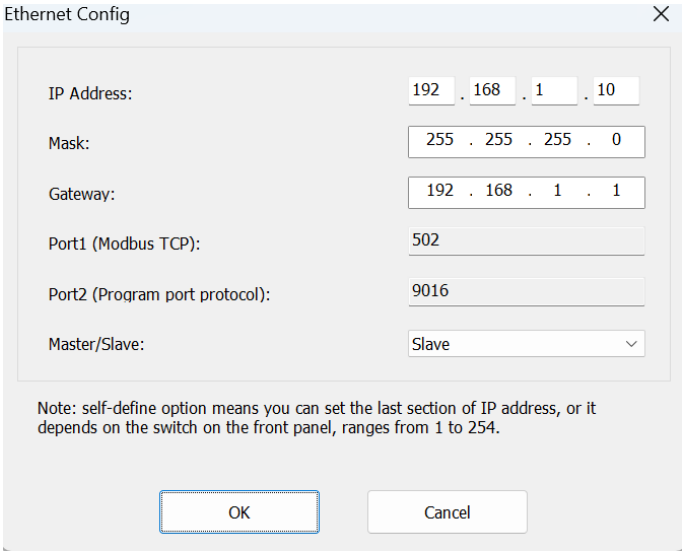
2. Alternatively, monitor the IP address through SD registers in the monitoring table, as illustrated below.

Output Window

	Element Name	Data Type	Display Format	Current Value	New Value	Element Remark
1	SD470	INT	Decimal	192		IP0
2	SD471	INT	Decimal	168		IP1
3	SD472	INT	Decimal	1		IP2
4	SD473	INT	Decimal	10		IP3
5	SD474	INT	Decimal	502		Ethernet slave listening port

IP Address Configuration

1. In the Project Manager, double-click “EtherNet”, and configure the IP address in the pop-up window, as shown below.



Function Description

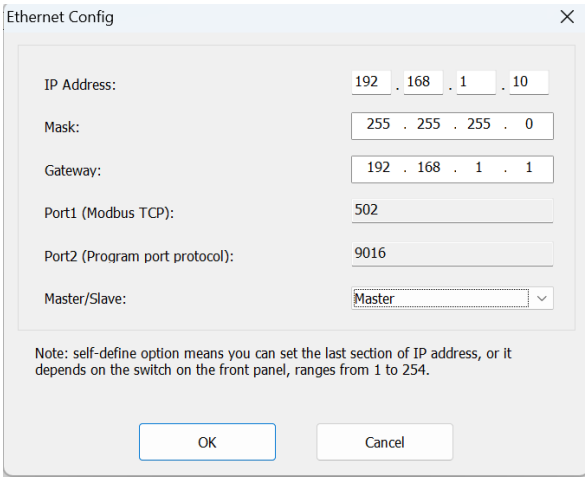
- The default IP address for SH series communication is 192.168.1.10. After restoring the PLC to factory settings, this IP is retained, enabling direct communication with host computers and Modbus TCP clients.
- IP Address: A unique identifier for the device in network communication. Each device must have a distinct IP address. Otherwise, the device cannot connect to the network.
- Subnet Mask: Enables addressing of multiple physical networks under the same network address. The mask divides the subnet address and the host ID. The subnet address is derived by retaining bits in the IP address corresponding to "1" positions in the mask and replacing others with "0". Default subnet mask: 255.255.255.0 (unless specified otherwise).
- Gateway Address: Routes messages to devices outside the current network. If no gateway exists, set to 0.0.0.0.
- Port 1: Reserved for Modbus TCP communication via TCP 502. Not configurable.
- Port 2: Used for AutoSoft host communication via Port 9016. Not configurable.
- Master/Slave Mode: Configures the device as a master or slave.

8.4 Master Configuration

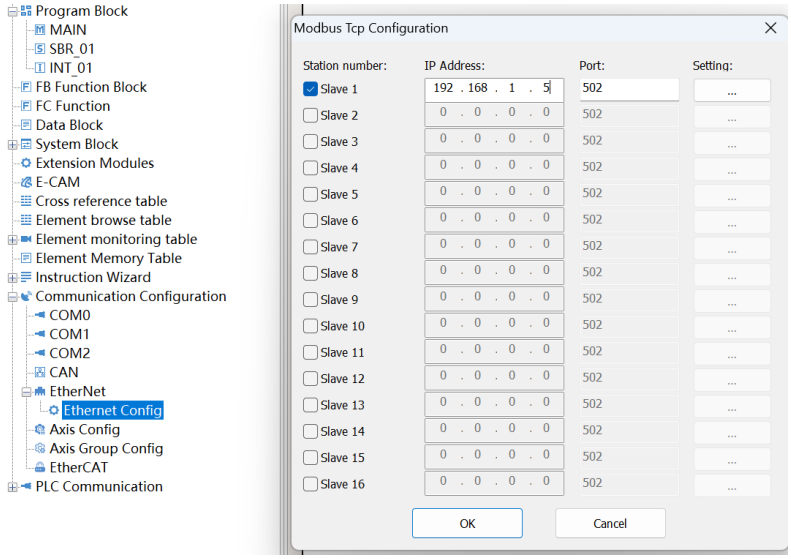
The Modbus TCP Master (Modbus TCP client) supports simultaneous communication with up to 16 Modbus TCP Slaves (Modbus TCP servers).

Steps:

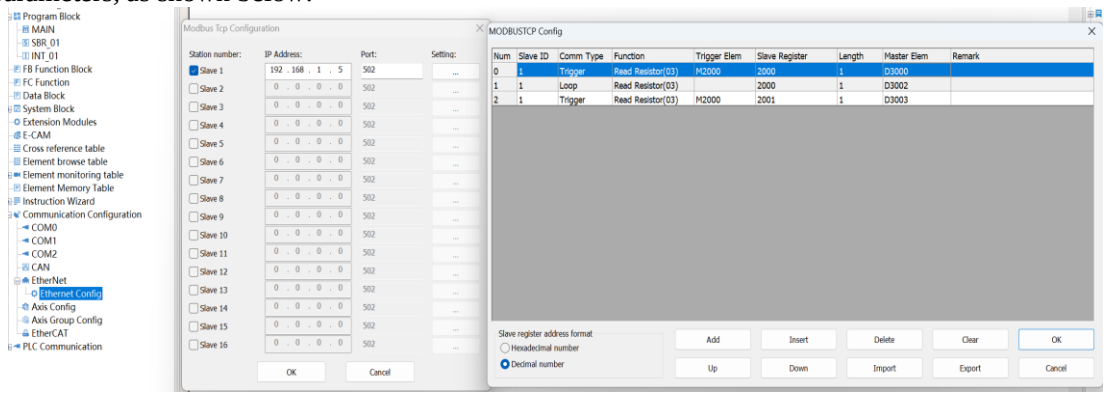
1. In the Project Manager area, double-click “EtherNet”, and configure the IP address and Master mode in the pop-up window, as shown below.



2. Right-click EtherNet -> Add Config -> Double-click Ethernet Config to configure the Modbus TCP Slave's IP address, as shown below.



3. Double-click "Setting" to add communication channels for the Modbus TCP Slave and configure relevant parameters, as shown below.



ModbusTCP Configuration Table Description

Parameter	Description
Slave ID	No setting required; reserved.
Communication Type	Loop mode: Polls slaves cyclically. Trigger mode: Requires a trigger condition (specified via trigger element). Accesses the slave when the element is ON; automatically turns OFF after completion.
Function	Read coils, write coils, read registers, write registers; supported function codes: 01, 02, 03, 04, 05, 06, 15, 16.
Trigger Element	Supports M elements (M0~M10247) for trigger configuration.
Slave Register	Address of the coil/register to access (decimal or hexadecimal)
Length	Number of data elements to access. Example: Accessing M10~M20 requires a length of 11.

Master Element	Address for transmitting/receiving data in the master buffer.

4. Click OK to confirm, then compile, download, and debug the program.

Program Example: Reads data from addresses 2000~2003 of the Slave (IP: 192.168.1.15) and stores it in registers D3000~D3003.

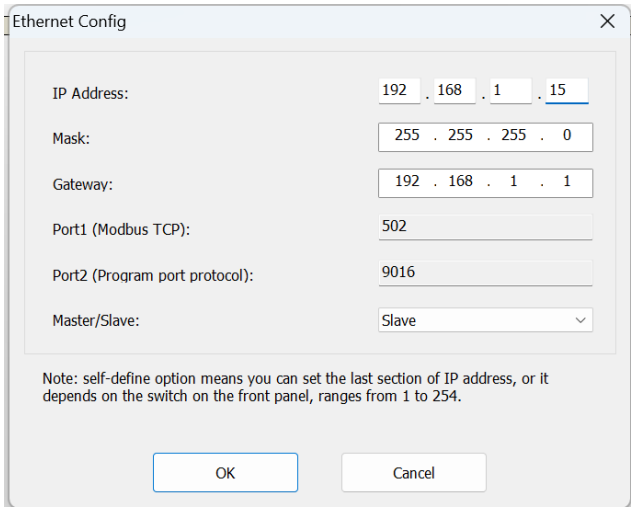
Note:

- To add multiple Modbus TCP slaves, repeat Steps 2~3.

8.5 Slave Configuration

ModbusTCP Slave

1. In the Project Manager area, double-click “EtherNet”, and configure the IP address and Slave mode in the pop-up window, as shown below.



8.6 ModbusTCP Function Codes

Supported ModbusTCP function codes:

Funcode	Function	Data Length
0x01 (01)	Read coils	>=1
0x02 (02)	Read coils	>=1
0x03 (03)	Read registers	>=1
0x04 (04)	Read registers	>=1
0x05 (05)	Write a single coil	=1
0x06 (06)	Write a single register	=1
0x0F (15)	Write multiple coils	>1
0x10 (16)	Write multiple registers	>1

8.7 ModbusTCP Communication Address

When SH series operates as a ModbusTCP slave, the address mapping for elements is as follows:

Element	Type	Physical Element	Protocol Address	Supported Function Codes	Comment
Y	Bit	Y0~Y777 (octal) 512 bits	0000~0511	01*05*15	Output status, with element addresses Y0–Y7, Y10–Y17, and so on.
X	Bit	X0~X777 (octal) 512 bits	1200~01711	01*05*15 02	Input status, with dual addressing supported. The element addresses are the same as above.

Element	Type	Physical Element	Protocol Address	Supported Function Codes	Comment
M	Bit	M0~M2047 M2048~M10239	2000~4047 12000-20191	01*05*15	
SM	Bit	SM0~SM255 SM256~SM1023	4400~4655 30000-30767	01*05*15	
S	Bit	S0~S1023 S1024~S4095	6000-7023 31000-34071	01*05*15	
T	Bit	T0~T255 T256~T511	8000~8255 11000-11255	01*05*15	T element status
C	Bit	C0~C255 C256~C511	9200~9455 10000-10255	01*05*15	C element status
D	Word	D0~D7999	0000~7999	03*06*16	
SD	Word	SD0~SD255 SD256~SD1023	8000~8255 12000-12767	03*06*16	
Z	Word	Z0~Z15	8500~8515	03*06*16	
T	Word	T0~T255 T256~T511	9000~9255 11000-11255	03*06*16	T element current value
C	Word	C0~C199	9500~9699	03*06*16	C element (INT) current value
C	Dual-word	C200~C255	9700~9811	03, 16	C element (DINT) current value
C	Dual-word	C256~C263	10000-10101	03, 16	C element (DINT) current value
R	Word	R0~R32767	13000-45767	03*06*16	

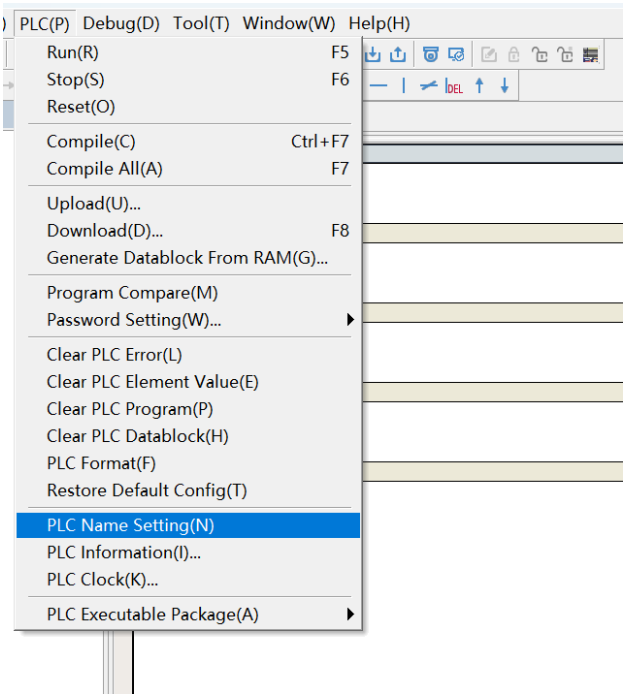
- Query quantity of coils/discrete inputs/registers

Funcode	Name	Max. Qty
0x01 (01)	Read coils	1920
0x02 (02)	Read discrete inputs	255
0x03 (03)	Read registers	120
0x04 (04)	Read inputs registers	255
0x05 (05)	Single coil	1
0x06 (06)	Single register	1
0x0f (15)	Write multiple coils	1920
0x10 (16)	Write multiple registers	120

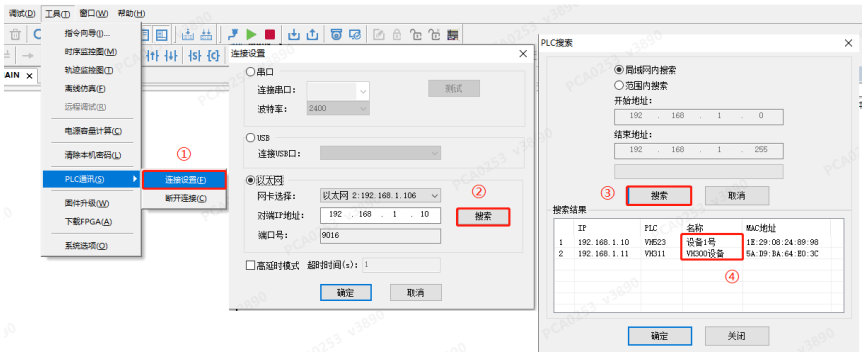
8.8 Device Name Modification

When multiple PLCs are connected within a LAN, assigning unique names helps distinguish devices efficiently.

1. Navigate to the menu bar: PLC > PLC Name Setting, as shown below.



2. Navigate to Tool> PLC Communication> Connect, select “Ethernet” and click “Search” to search in local area network. Double-click the target device to establish communication with the host software, as shown in the figure below:



8.9 Ethernet-related SD Registers

When using Ethernet connection, users can monitor or modify IP addresses through SD special registers.

Address	Function	Property	Description
SD470	IP Address 0	R/W	IP Address 0
SD471	IP Address 1	R/W	IP Address 1
SD472	IP Address 2	R/W	IP Address 2
SD473	IP Address 3	R/W	IP Address 3
SD474	Ethernet Slave Listening Port 502	R	Port 502
SD475	MAC Address 0	R	MAC Address 0
SD476	MAC Address 1	R	MAC Address 1
SD477	MAC Address 2	R	MAC Address 2
SD478	MAC Address 3	R	MAC Address 3
SD479	MAC Address 4	R	MAC Address 4
SD480	MAC Address 5	R	MAC Address 5
SD481	Slave IP3 Address Number for Communication Errors	R	Indicates network communication errors
SM470	IP Address Modification Enable	R/W	Set SM470 to ON after modifying IP addresses to activate new IP immediately.

8.10 Ethernet Free Port Protocol

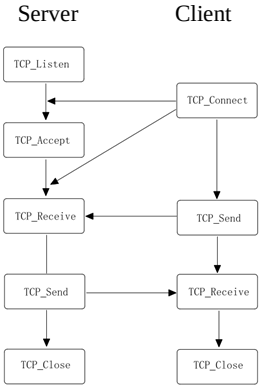
8.10.1 Overview

TCP Free Port (Socket) communication is a two-way method where both parties must implement programs: one acts as the active party to send data, and the other as the passive party to receive data. Hosts on the network transmit data through interfaces provided by sockets. The SH series offers Ethernet socket interfaces. By utilizing sockets, users can conveniently achieve communication between different devices over a TCP/IP network.

8.10.2 Transmission Control Protocol

Transmission Control Protocol (TCP) is a connection-oriented, reliable, byte-stream-based transport layer communication protocol. The Internet differs significantly from individual networks, as its various segments may have distinct topologies, bandwidths, latencies, packet sizes, and other parameters. TCP is designed to dynamically adapt to these characteristics of the Internet and maintain robustness against diverse failures.

The SH series provides a connection-oriented socket TCP communication interface, with the workflow illustrated in the figure below.



8.10.3 Free Port Protocol High/Low Byte

The data transmission and reception formats can be controlled via SM special registers, effective only for Ethernet Free Port Protocol instructions.

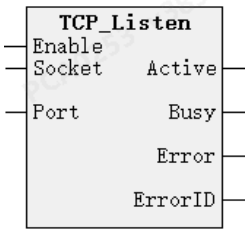
Mode Configuration	SM Special Register
High/Low Byte Validity	SM471=OFF: Both high and low bytes are valid. Example: Sending data 16#1234 will transmit both high byte (16#12) and low byte (16#34).
	SM471=ON: Only the low byte is valid. Example: Sending data 16#1234 will transmit only the low byte (16#34), omitting the high byte (16#12).

8.10.4 Freeport Protocol Instruction List

Instruction Name	Function
TCP_Listen	Establish listening state (server-side)
TCP_Accept	Establish connection (server-side)
TCP_Connect	Establish connection (client-side)
TCP_Close	Close Ethernet connection
TCP_Send	Send data via Ethernet
TCP_Recive	Receive data via Ethernet

8.11 Freeport Protocol Instruction Descriptions

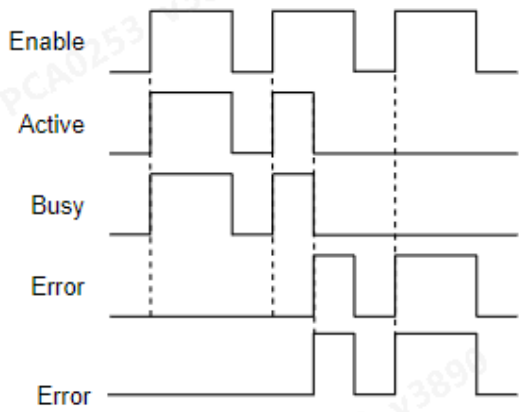
8.11.1 TCP_Listen Instruction (Establish Listening State - Server)



I/O	Name	Data Type	Applicable (Soft Element)	Range	Description
IN	Enable	BOOL	M/S	TRUE, FALSE	Enabled
IN	Socket	WORD	Constant/D/R/W	0~4095	User-defined socket
IN	Port	WORD	Constant/D/R/W	0x0~0xffff	Local port number
OUT	Active	BOOL	M/S	TRUE, FALSE	Active flag
OUT	Busy	BOOL	M/S	TRUE, FALSE	Executing flag
OUT	Error	BOOL	M/S	TRUE, FALSE	Error flag
OUT	ErrorID	WORD	D/R/W	0x0~0xffff	Error ID

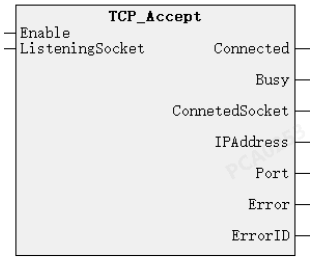
Function and Instruction Description

- The server must wait for connection requests from clients. The TCP_Listen instruction is used to listen on a specified local port for client requests. Once a connection request is received, the TCP_Accept instruction must be used to establish communication with the client.



8.11.2 TCP_Accept Instruction (Establish Connection - Server)

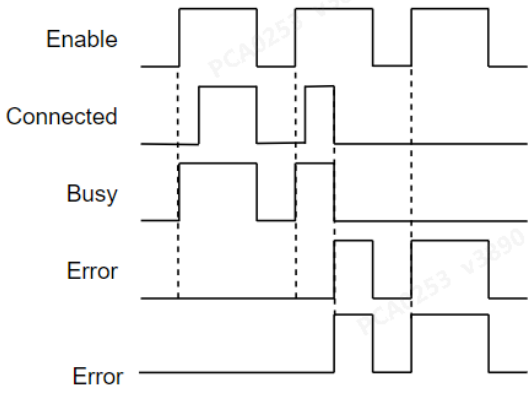
The TCP_Accept instruction accepts connection requests. When a server in listening state receives a client's connection request, it places the request into a waiting queue. When operating as a server, use the TCP_Accept instruction to accept client connection requests.



I/O	Name	Data Type	Applicable (Soft Element)	Range	Description
INT	Enable	BOOL	M/S	TRUE, FALSE	Enabled
IN	ListeningSocket	WORD	Constant/D/R/W	0~4095	User-defined socket
OUT	Connected	BOOL	M/S	TRUE, FALSE	Connection established flag
OUT	Busy	BOOL	M/S	TRUE, FALSE	Executing flag
OUT	ConnetedSocket	WORD	D/R/W	4096~65535	Connected socket
OUT	IPAddress	DWORD	D/R/W	-	Remote host IP address
OUT	Port	WORD	D/R/W	-	Remote host port
OUT	Error	BOOL	M/S	TRUE, FALSE	Error flag
OUT	ErrorID	WORD	D/R/W	0x0~0xffff	Error ID

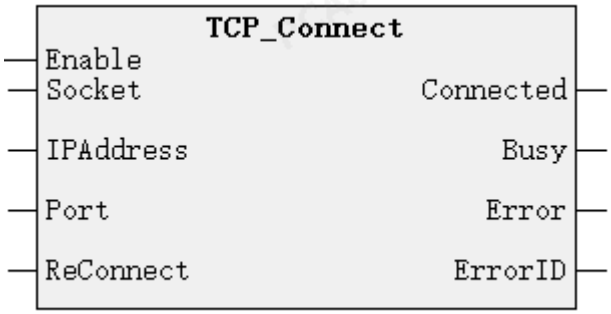
Function and Instruction Description

- A server in listening state must use the TCP_Accept instruction to establish communication with a client after receiving its connection request. After successful communication establishment, data transmission or reception can be performed via TCP_Send or TCP_Receive.
- Multiple TCP_Accept instructions enable the same local port to establish communication connections with multiple clients.
- The maximum total number of simultaneous connections supported between clients and the server is four.



8.11.3 TCP_Connect Instruction (Establish Connection - Client)

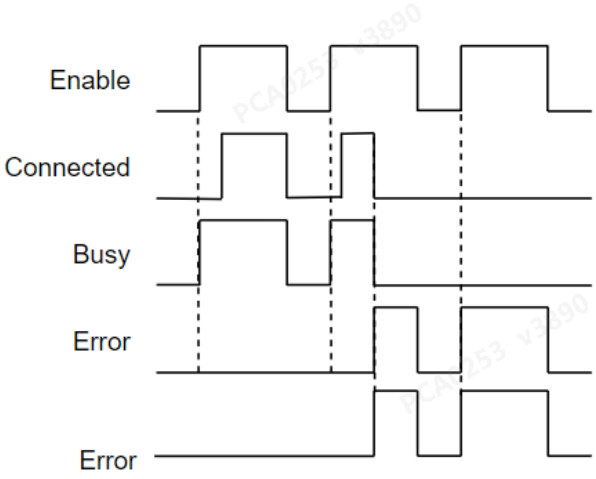
For a client to communicate with a server, it must send a connection request to the server. When operating as a client, use the TCP_Connect instruction to initiate connection requests.



I/O	Name	Data Type	Applicable (Soft Element)	Range	Description
IN	Enable	BOOL	M/S	TRUE, FALSE	Enabled
IN	Socket	WORD	Constant/D/R/W	0~4095	User-defined socket
IN	IPAddress	DWORD	IP address	-	Target server IP address
IN	Port	WORD	Constant/D/R/W	0x0~0xffff	Target server port number
IN	ReConnect	BOOL	M/S	TRUE, FALSE	Auto-reconnect when ON
OUT	Connected	BOOL	M/S	TRUE, FALSE	Connection status with the server
OUT	Busy	BOOL	M/S	TRUE, FALSE	Executing flag
OUT	Error	BOOL	M/S	TRUE, FALSE	Error flag
OUT	ErrorID	WORD	D/R/W	0x0~0xffff	Error ID

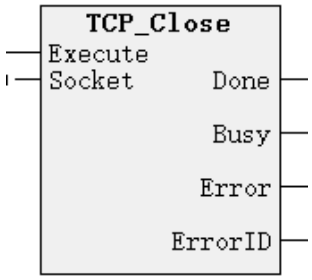
Function and Instruction Description

- When operating as a client, use the TCP_Connect instruction to connect to a specified server port for communication. After the server accepts the connection, data transmission or reception can be performed via TCP_Send or TCP_Receive.
- After initiating a connection request via TCP_Connect, the client waits up to 127 seconds. If the server does not respond, the connection fails.



8.11.4 TCP_Close Instruction (Close Ethernet Connection)

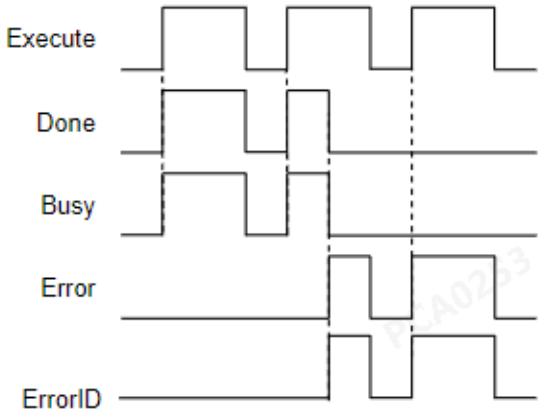
The TCP_Close instruction is used to close connections or terminate listening after communication completion.



I/O	Name	Data Type	Applicable (Soft Element)	Range	Description
IN	Execute	BOOL	M/S	TRUE, FALSE	Rising edge trigger
IN	Socket	WORD	Constant/D/R/W	-	User-defined socket
OUT	Done	BOOL	M/S	TRUE, FALSE	Execution completed
OUT	Busy	BOOL	M/S	TRUE, FALSE	Executing flag
OUT	Error	BOOL	M/S	TRUE, FALSE	Error flag
OUT	ErrorID	WORD	D/R/W	0x0~0xffff	Error ID

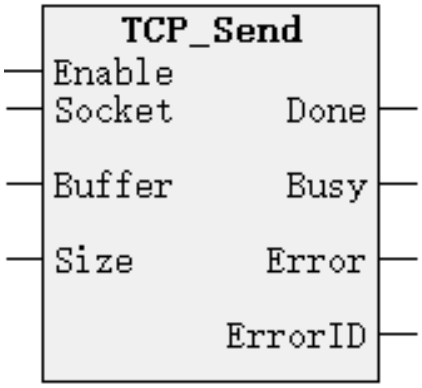
Function and Instruction Description

- After completing communication, use the TCP_Close instruction to close connections, stop listening, or terminate active sockets.
- The TCP_Send instruction transmits data to a remote host after a successful connection is established between the server and client.



8.11.5 TCP_Send Instruction (Ethernet Data Transmission)

The TCP_Send instruction transmits data to a remote host after a successful connection is established between the server and client.

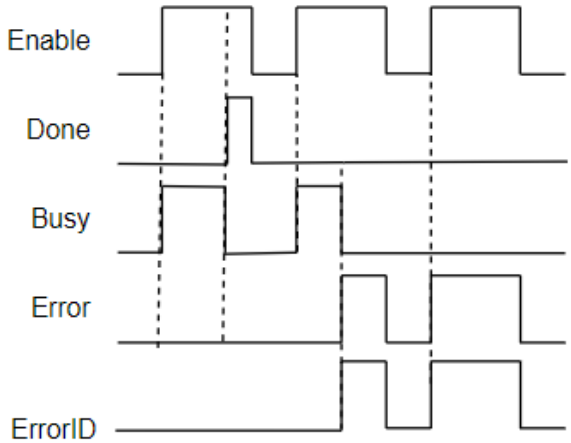


I/O	Name	Data Type	Applicable (Soft Element)	Range	Description
IN	Enable	BOOL	M/S	TRUE, FALSE	Enabled
IN	Socket	WORD	Constant/D/R/W	0~65535	User-defined socket
IN	Buffer	WORD	Constant/D/R/W	Software address	Start address for sending data
IN	Size[note]	WORD	Constant/D/R/W	0~256	Data length and byte order configuration (low/high byte).
OUT	Done	BOOL	M/S	TRUE, FALSE	Transmission completion flag
OUT	Busy	BOOL	M/S	TRUE, FALSE	Transmission in progress flag
OUT	Error	BOOL	M/S	TRUE, FALSE	Error flag
OUT	ErrorID	WORD	D/R/W	0x0~0xffff	Error ID
Note:	The Size parameter has two functions: 1. Low byte (bits 0~7): Specifies the data length to be sent. 2. Highest bit of high byte (bit 15): Controls the byte order (high/low). Example: When using TCP_Send to transmit Buffer data 16#1234 with a length of 10 bytes: If Size = 16#000A, the sent data is 16#1234. If Size = 16#800A, the sent data is 16#3412.				

Function and Instruction Description

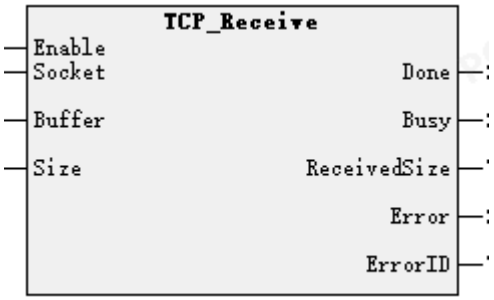
- After a successful connection is established between the server and client, the TCP_Send instruction

transmits data from the buffer to the remote host with the specified length. The Size parameter must be less than or equal to the actual size of the data buffer (Buffer parameter); otherwise, there is a risk of out-of-bounds data access.



8.11.6 TCP_Receive Instruction (Ethernet Data Reception)

The TCP_Receive instruction retrieves message data sent from a remote host via a specified socket after a successful connection is established between the server and client.

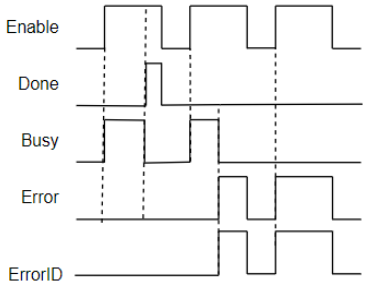


I/O	Name	Data Type	Applicable (Soft Element)	Range	Description
IN	Enable	BOOL	M/S	TRUE, FALSE	Enabled
IN	Socket	WORD	Constant/D/R/W	0~65535	User-defined socket
IN	Buffer	WORD	Constant/D/R/W	Software address	Start address for receiving data
IN	Size[note]	WORD	Constant/D/R/W	0~256	Buffer length and byte order configuration
OUT	Done	BOOL	M/S	TRUE, FALSE	Reception completion flag
OUT	Busy	BOOL	M/S	TRUE, FALSE	Reception in progress flag
OUT	RxSize	WORD	D/R/W	0~256	Received data length
OUT	Error	BOOL	M/S	TRUE, FALSE	Error flag
OUT	ErrorID	WORD	D/R/W	0x0~0xffff	Error ID
Note:	The Size parameter has two functions: 1. Low byte (bits 0~7): Sets the receive data length. 2. Highest bit of high byte (bit 15): Configures the byte order (high/low). Example: When receiving data 16#1234 with a length of 10 bytes: If Size = 16#000A, the received data is 16#1234. If Size = 16#800A, the received data is 16#3412.				

Function and Instruction Description

- When the Size parameter is set to 0, data is transmitted in string format, i.e., sending data from the buffer (Buffer parameter) starting at the first byte up to (but excluding) the terminator (ASCII code 0).

- The Size parameter must be less than or equal to the actual size of the data buffer (Buffer parameter); otherwise, there is a risk of out-of-bounds data access. After a successful connection is established between the server and client, message data sent from the remote host is stored in the socket buffer. Use the TCP_Receive instruction to retrieve received message data from the specified socket buffer.



8.12 TCP Server Communication Example

SH311 operates as a TCP server using the TCP_Listen instruction for listening, with Port=100 (configurable 0-4095). After accepting client connections, it can receive data from clients and send data back.

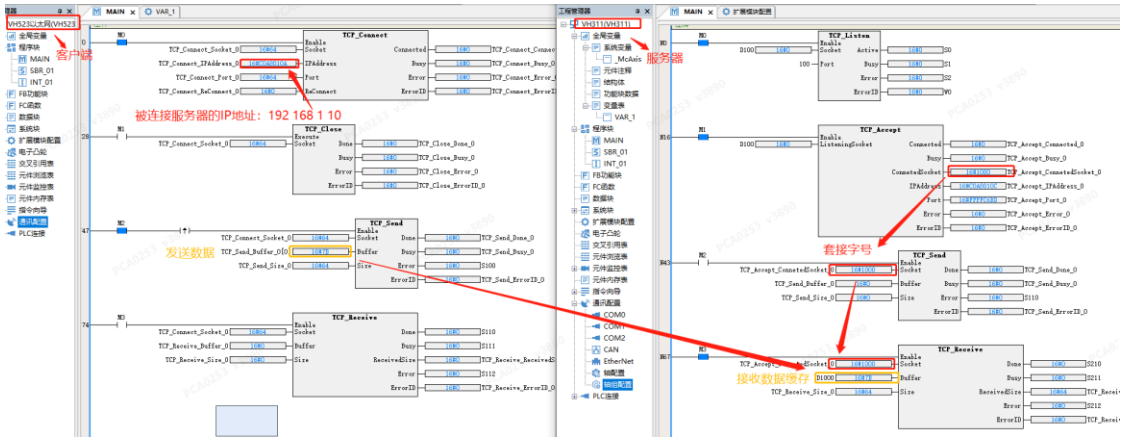
Software Configuration:

SH523: IP 192.168.1.12 (Client)

SH311: IP 192.168.1.10 (Server)

TCP Server Function Implementation:

1. The server receives data 16#7B sent from the client and stores it in register D1000. as shown in the figure below:



Note:

- When the PLC acts as a server, the TCP_Listen instruction automatically generates a socket ID upon successful connection. The Socket parameter for both server-side receive and send instructions must be identical; otherwise, error code 1809 will occur.
- IPAddress parameter configuration in TCP_Connect: For example, the server IP address 192.168.1.10 corresponds to 16#C0A8010A.

8.13 TCP Client Communication Example

SH523 operates as a TCP client with Port=100, sends a connection request to the server at 192.168.1.10, and transmits 20 bytes after establishing the connection.

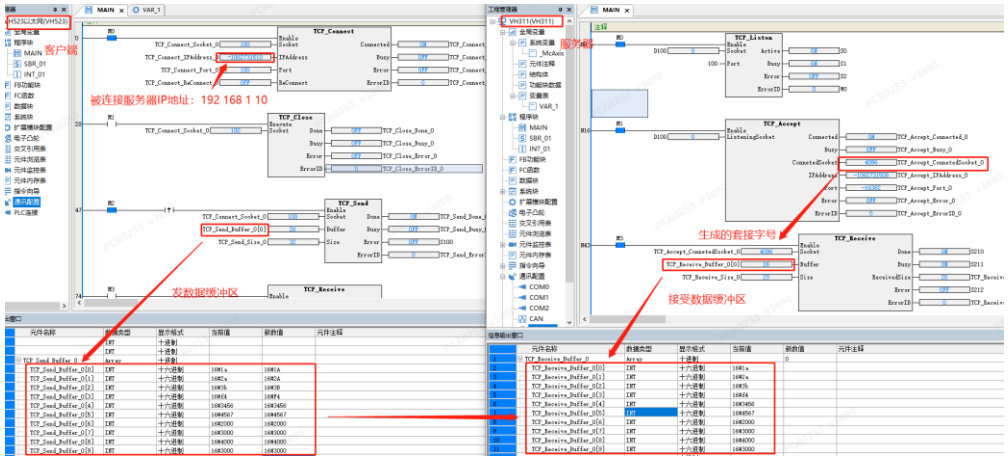
Software Configuration:

SH523: IP 192.168.1.12 (Client)

SH311: IP 192.168.1.10 (Server)

TCP Server Function Implementation:

1. The client sends data 16#7B, which the server receives and stores in register D1000.



8.14 TCP Freeport Error Codes

When using the Ethernet freeport protocol, incorrect instruction parameters will trigger the following errors:

Error Code	Description
1800	Socket ID error
1801	Socket port error
1802	Socket port or ID already exists
1803	Failed to create Socket listener
1804	Failed to bind Socket port
1805	Max. number of socket ports exceeded
1806	Socket pointer error
1807	Socket listener port already closed
1808	Socket connection port already closed
1809	Socket closed
1810	Socket data reception error
1811	Data sent via unconnected Socket
1812	Multiple consecutive Socket data transmission errors
1813	Socket data size exceeds limit
1814	Host disconnected

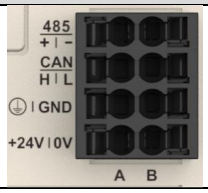
9 CAN Communication

9.1 Overview

The SH series (SH311/SH522/SH523/SH524) features a built-in CAN communication interface that supports the CANopen communication protocol. Using CANopen, the system supports the extension of up to 31 slaves (with 16 recommended).

9.2 Hardware interface

The CANopen hardware interfaces are described in the table below:

Terminal Interface	Terminal A (Left Side)	Terminal B (Right Side)
	RS-485+	RS-485-
	CAN (H)	CAN (L)
	⏚	GND
	+24V	0V

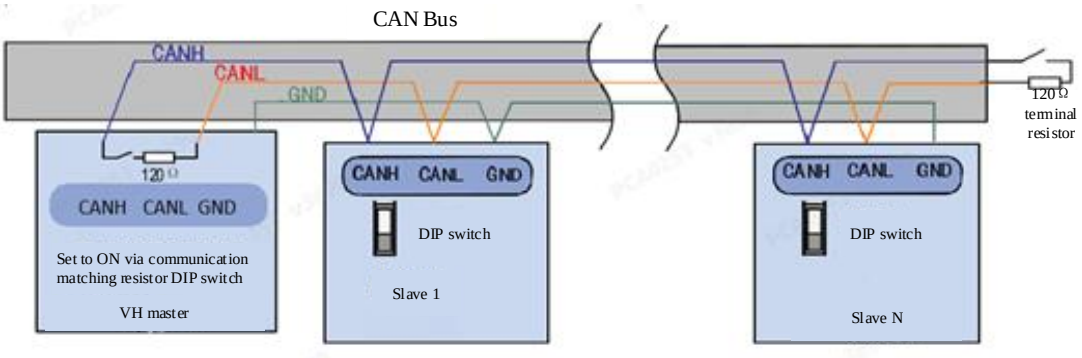
Termination Resistor DIP Switch

The termination resistor DIP switch is located below the left extension interface. ON indicates the termination resistor is connected (defaulted). The switch diagram is as follows: Switch 1 for RS485 communication, and switch 2 for CAN communication.

9.3 CAN Communication Networking

When configuring a CAN network, all three wires (CAN_H, CAN_L, and GND) of each device must be connected in a one-to-one correspondence. Both ends of the bus must be terminated with a 120Ω CAN bus termination resistors. The SH master has a built-in resistor, which can be enabled/disabled via the DIP switch, defaulting to ON.

The CAN wiring diagram for a multi-device network is as follows:



Note:

- The GND of all devices must be connected together.

9.3.1 Relationship Between Distance and Baud Rate

The relationship between supported baud rates and communication distances for CAN communication is shown in the table below:

Baud Rate (kbit/s)	Distance (m)	Min Wire Diameter (mm^2)	Max Nodes
1000	20	0.3	18
500	80	0.3	31
250	150	0.3	31
125	300	0.5	31
100	500	0.5	31
50	1000	0.7	31

9.4 CANopen Protocol

The SH series supports the CANopen communication standard DS301.

Software Function	Master	Slave
Supported Protocol	DS301V4.02	DS301V4.02
Max TPDO	64	4
Max RPTO	64	4
Slave Nodes	30	/
Data exchange elements	D0~D7999 (configurable)	SD400~SD415 (receive area) SD432~SD447 (transmit area)

9.4.1 CANopen Indicators

During CANopen communication, the operation status can be determined via the CANopen indicators, as shown in the table below:

LED State	CAN (Green)	ERR (Red)
OFF	Unconfigured	No error
ON	Operation status	System error
Flash	Communication error	CAN communication error or system error

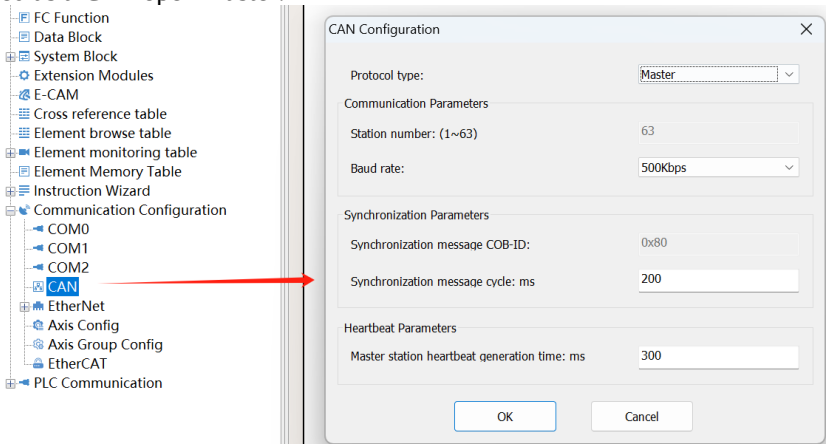
9.4.2 CANopen Terms

- NMT (Network Management): Manages application layers, network states, and node ID allocation, and operates in a master-slave mode: only one NMT master and multiple slaves are allowed in a CAN network. It is primarily used to control slave states.
- SDO (Service Data Object): Accesses data in a slave device's object dictionary via index and sub-index, mainly used for slave configuration. Each SDO frame requires an acknowledgment reply.
- PDO (Process Data Object): Transmits real-time data, limited to 1 to 8 bytes. PDO transmission includes synchronous and asynchronous modes. PDO frames are the primary data exchange frames after slave initialization.
- SYNC (Synchronous Service): Uses master-slave mode. The SYNC master periodically sends SYNC objects, and SYNC slaves synchronize task execution upon receipt. It is primarily used for synchronous PDO transmission.
- COB-ID (Communication Object Identifier): Each CANopen frame starts with a COB-ID, which acts as the communication object identifier in the CAN frame. The COB-ID is not equal to the slave ID but is typically initialized by default to correlate with the slave ID.

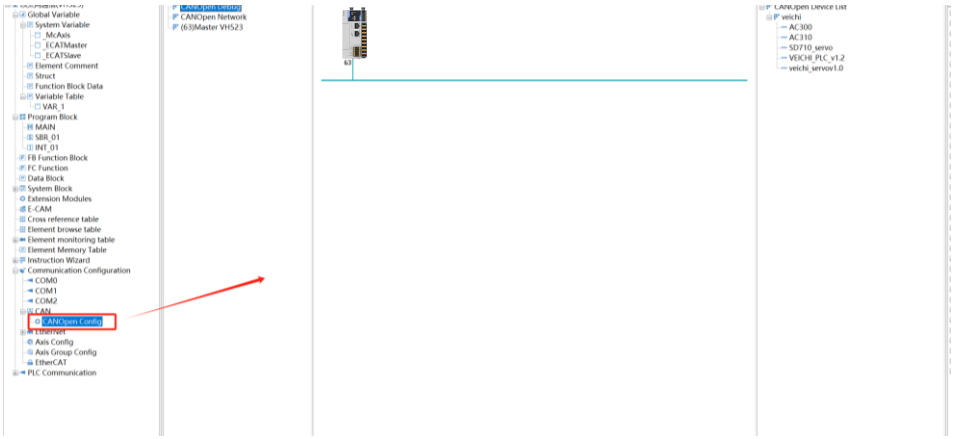
9.5 CANopen Configuration

9.5.1 Master Configuration

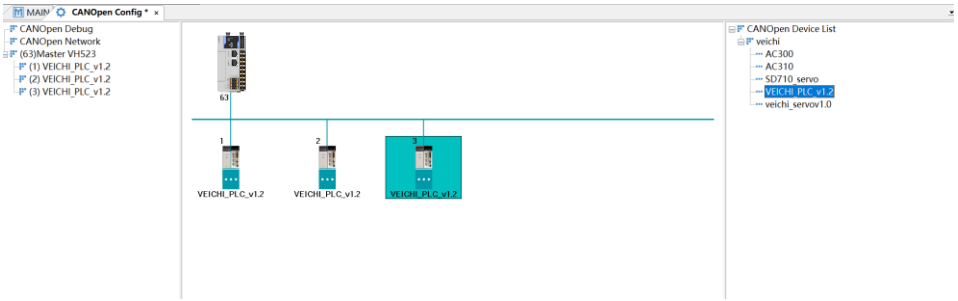
1. Open AutoSoft software, double-click "CAN" under the Communication Config, select "Master" as the protocol type, set the Station Number and Baud Rate as required, and click "OK". The CAN interface is then configured as a CANopen master.



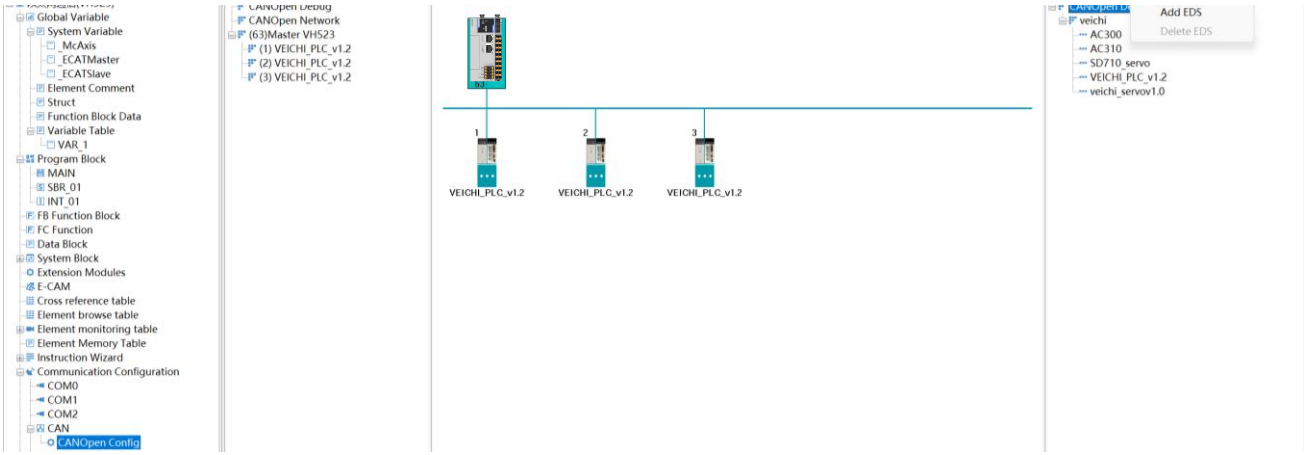
2. Right-click CAN -> Add Config. After adding, double-click CANopen Config, as shown below.



3. Add CANopen slaves by double-clicking or dragging them in the CANopen Device List, as shown below.

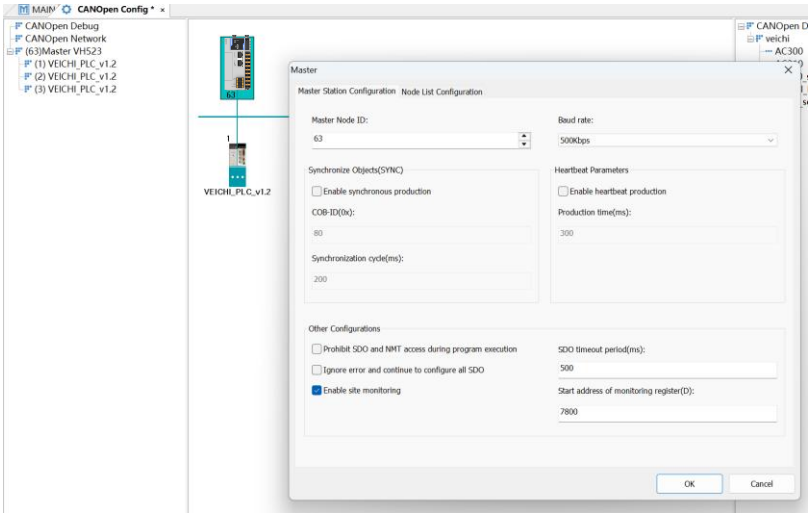


4. If a slave is not listed, right-click on CANopen Device List, select Add EDS (EDS files can be obtained from the device supplier), or manually copy the EDS file to Installation Path > Config Folder and restart the software.



Master Configuration

1. Set master parameters by double-clicking the SH master in the network. The following window will appear:



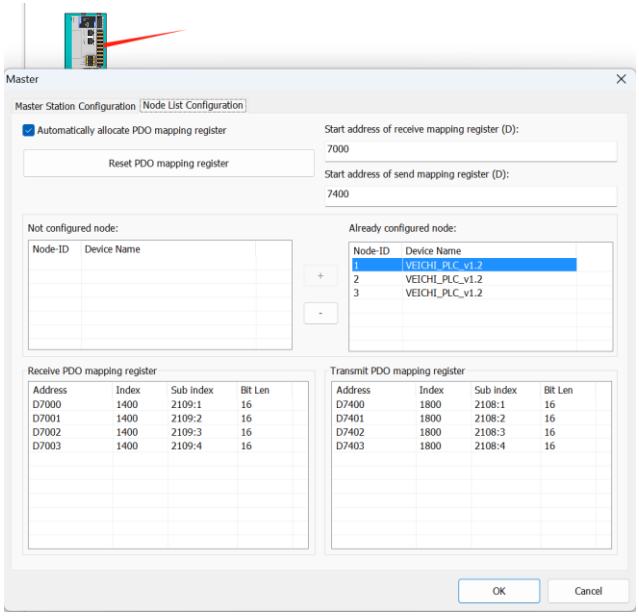
Master Parameters Description

Item	Description
Master Node ID	Set the master node ID. When this ID matches the PLC's own station number, the PLC is initialized as a CANopen master.
Baud Rate	Effective communication baud rate for the master.
Prohibit SDO and NMT access during program execution	When enabled, online debugging functions are disabled during program execution. This restriction applies only to background software.
Ignore error and continue to configure all SDO	When enabled, configuration proceeds even if errors occur (except validation errors). This applies to all slaves. If disabled, the master broadcasts a reset to slaves upon SDO errors.
Enable sync production	When enabled, the master sends Sync frames cyclically at the interval set in Sync Cycle (ms). (Note: Only one Sync frame transmitter is allowed in the network.)
COB-ID	Sync frame COB-ID. Default: 0x80 (non-configurable). Sync Cycle (ms): SYNC frame transmission interval. Default: 200 ms. Window Length (ms): Default: 0 (non-configurable).
Enable heartbeat production	When enabled, the master sends heartbeat frames cyclically at the interval set in Production Time (ms). Production Time (ms): Heartbeat transmission interval. Default: 300 ms. (Note: The master's default heartbeat consumption time is 2.5× the heartbeat production time.)
SDO Timeout Period	SDO response wait time. Default: 500 ms. SDO frames are primarily used for network configuration. If no response is received after 3 retries, the master declares a timeout. This value defines the wait interval per frame.
Start address of monitoring register	The online status of nodes will be updated to the designated register (default: D7800 (configurable)).

Value	Status
0	Initialization
4	Stop
5	Operation
127	Pre-run
255	Offline

Master Node List Configuration

1. Node configuration is primarily used to map slave receive and transmit addresses, as shown below.

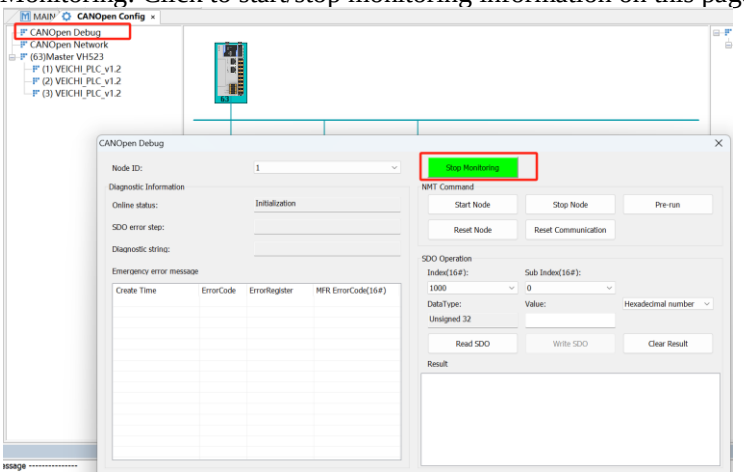


Description

- Automatically allocate PDO mapping register: When enabled, register addresses for master-slave data exchange are automatically assigned; when disabled, manually set the start address for each PDO (default: enabled).
- Start address of receive mapping register: Auto-assigned start address for data transmitted by the master (meaningful only when “Automatically allocate PDO mapping register” is enabled).
- Slave address of send mapping register: Auto-assigned start address for data received by the master (meaningful only when “Automatically allocate PDO mapping register” is enabled).

Network Status

Start Monitoring/Stop Monitoring: Click to start/stop monitoring information on this page.



Parameter Descriptions:

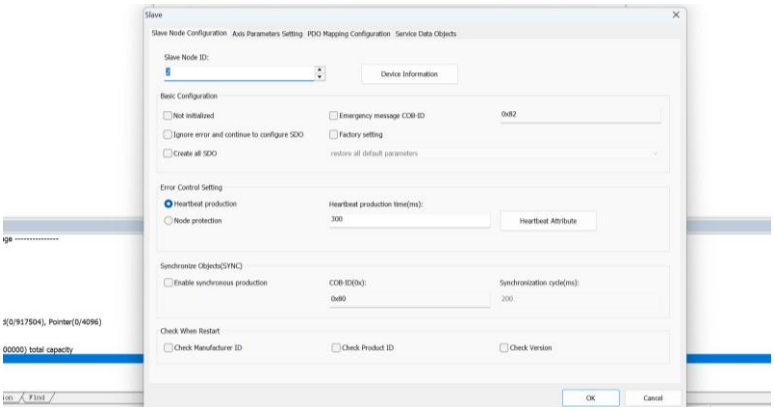
- Network Load: Monitors network load in real time.
- Network Status: Displays the operation status of current network nodes. Monitoring is meaningful only for the master. Status values are sourced from the node status monitoring register.
- Emergency Error Message: Displays emergency error messages in the network. Monitoring is meaningful only for the master. The PLC master caches only the latest error message. Up to 5 messages are cached if the background remains open.
- SDO Configuration Node Number: Node ID with SDO configuration errors.
- Error Step: ID of the SDO error. Refer to the "Service Data Object" tab of the corresponding slave for details.
- Error Code: SDO error codes. (CANopen standard error codes).

9.5.2 Slave Configuration

This section describes the CANopen slave configuration process and parameters using the SD710 slave as an example.

(1) Slave Node Configuration

1. Drag the slave into the configuration and double-click the slave in the network to open the dialog box, as shown below:



(2) Parameter Description

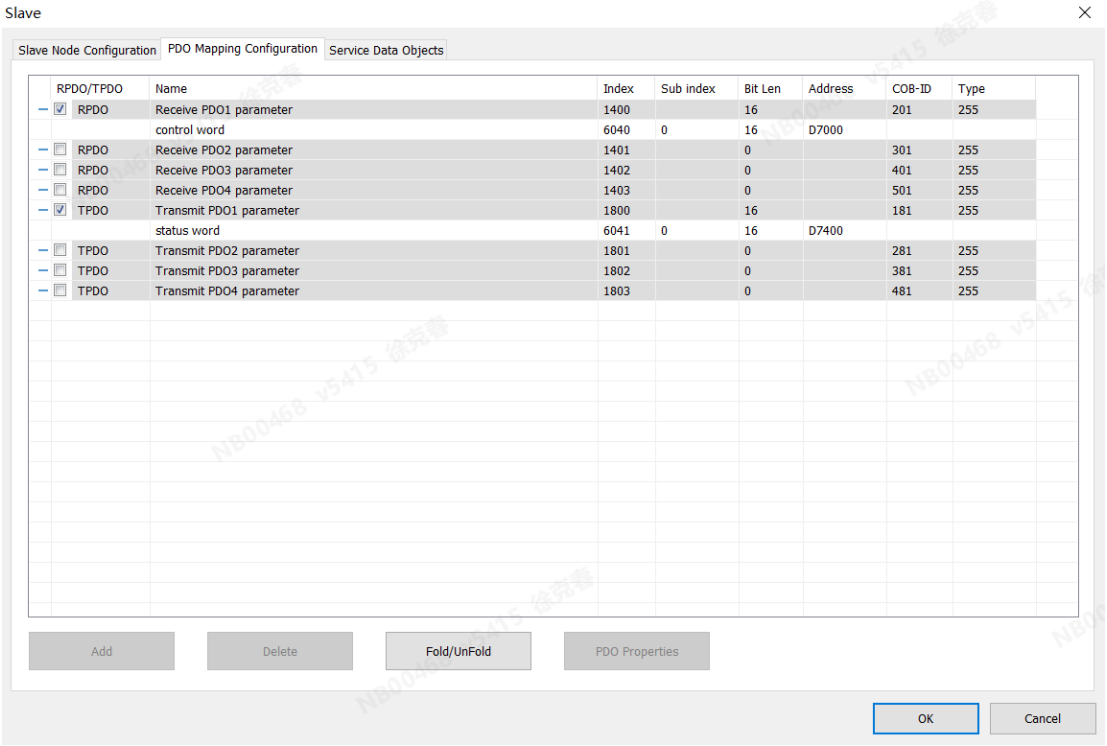
Item	Description
Slave Node ID	Node ID of the slave to be configured.
Ignore error and continue to configure SDO	Enabled: Configuration continues on errors (except validation errors). Disabled: Configuration halts on errors, and the master stops the entire network if it is running. This function is disabled by default.
Create all SDO	When enabled, all writable object dictionary entries in the EDS are added and initialized during configuration. This function is disabled by default.
Not initialized	When enabled, the slave skips initialization (selectable only when using default configurations). This function is disabled by default.
Factory Setting	When enabled, additional operations can be selected. (Default: disabled; this option requires support from the selected slave device).

(3) Error Control Setting

Item		Description
Error Control	Node Protection	When enabled, node protection is activated for the slave (default: disabled). Node protection timeout = protection time × lifecycle factor. Node protection is a mutual monitoring mechanism between master and slave via acknowledgment frames. Heartbeat and node protection are mutually exclusive. Protection time (ms): Node protection interval, defaulted to 200 ms. Lifecycle factor: Node protection factor, defaulted to 3.
	Heartbeat Production	When enabled, the slave generates heartbeat frames (default: enabled). The master monitors the slave's heartbeat status by default. Heartbeat Production Time (ms): Heartbeat transmission interval. Heartbeat Attribute: Configures the slave to monitor heartbeats from other nodes (default: disabled; requires slave support for heartbeat monitoring).
	Synchronize Objects	When enabled, the slave sends Sync frames cyclically at the interval set in Sync Cycle (ms). COB-ID: Sync frame COB-ID, defaulted to 0x80 (non-configurable). Sync Cycle (ms): Sync frame transmission interval, defaulted to 200 ms. Window Length (ms): Default: 0 (non-configurable). Note: Only one Sync frame transmitter is allowed in the network.
	Emergency Message	When enabled, the emergency message COB-ID is configured during configuration (default: disabled).
	Check When Restart	Vendor ID Check, Product ID Check, Version Check: When enabled, the corresponding validations are performed before slave configuration begins. If validation fails, the network cannot start.

9.5.3 PDO Mapping Configuration

Receive PDO (RPDO): Data transmitted from the master to the slave.
Transmit PDO (TPDO): Data transmitted from the slave to the master.

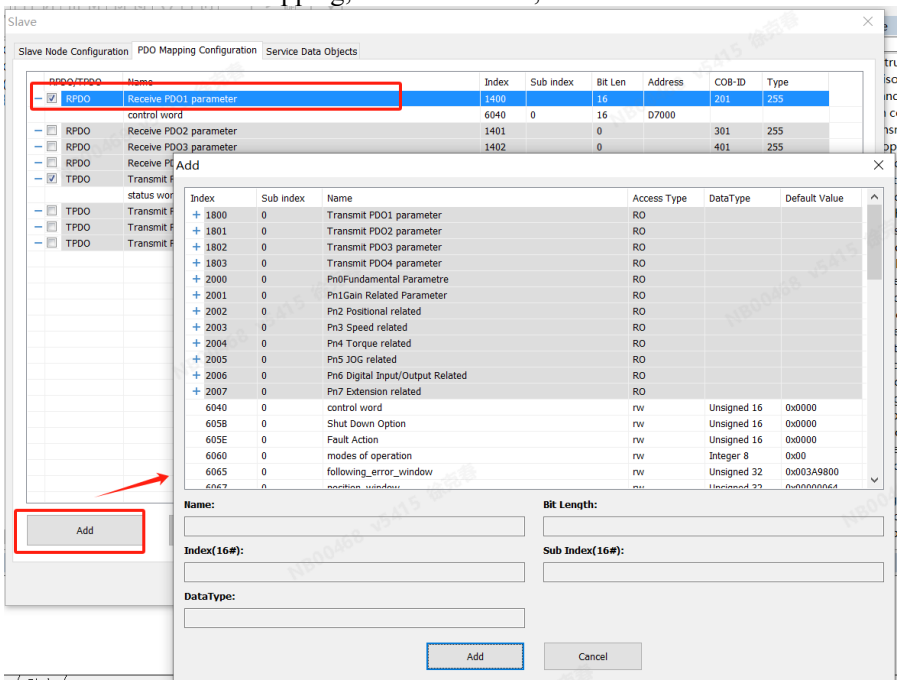


(1) PDO Enable

Enable PDO: Check the checkbox to enable the PDO. Default-enabled PDOs from the slave's EDS file are pre-checked.

(2) PDO Mapping Edit

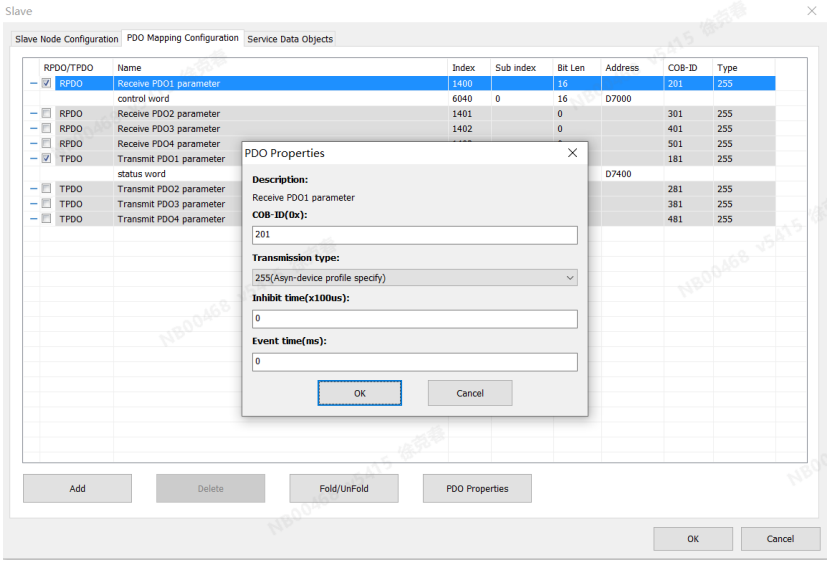
Click "Add", select the desired PDO mapping, and click "OK", as shown below.



To delete a PDO mapping, check the target PDO and click "Delete".

9.5.4 PDO Property

Double-click a PDO to access its property settings, as shown below:



(1) COB-ID

The ID used for PDO transmission. According to CANopen DS301, the first four PDOs have default COB-ID initial values; others require manual configuration (if supported by the slave). Configuration rules: No duplicate COB-IDs are allowed in the network, with valid range of 0x180~0x57F.

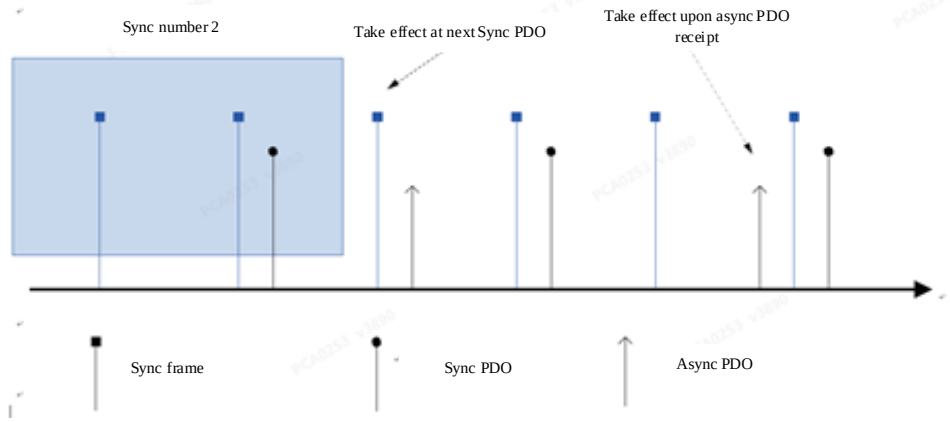
Transmission Type	Transmission Trigger	Activation Condition
Loop-Sync (Type 0)	Data changes + a Sync frame received	Activated after receiving a Sync frame.
Loop-Sync (Type 1~240)	After specified Sync counter reached	Activated after receiving a Sync frame.
Async-RTR only (Type 252)	N/A	N/A
Async-RTR only (Type 253)	N/A	N/A
Async-vendor specific (Type 254)	Vendor-defined	Vendor-defined
Async-device profile specify (Type 255)	Async-device profile specify (Type 255)	Immediately effective

Note:

➢ When using synchronous transmission types, ensure sync production is enabled on a node (typically the master).

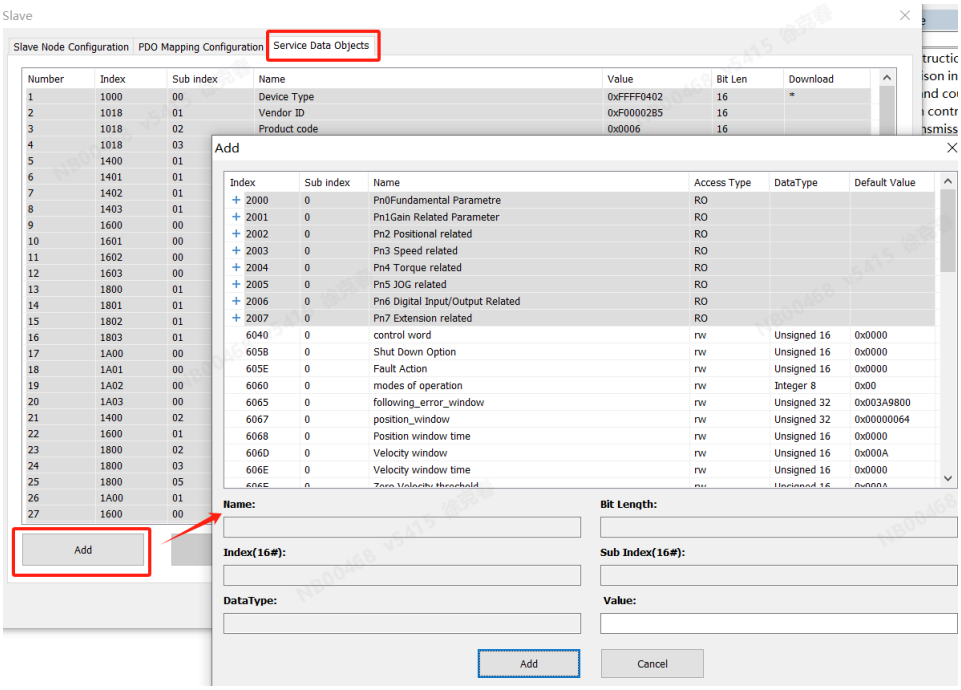
- Sync Counter
  Valid when Loop-Sync (Type 1~240) is selected. Set the Sync counter value.
- Inhibit Time (100μs)
  Configurable for Async-device profile specify (Type 255). 0: Disabled; Non-zero: Minimum interval between frame transmissions.
- Event Time (ms)
  Configurable for Async-device profile specify (Type 255). 0: Disabled; Non-zero: Periodic transmission interval, subject to the inhibit time.

Example: The figure below illustrates settings for Loop-Sync (Type 1~240).



9.5.5 Service Data Objects (SDO)

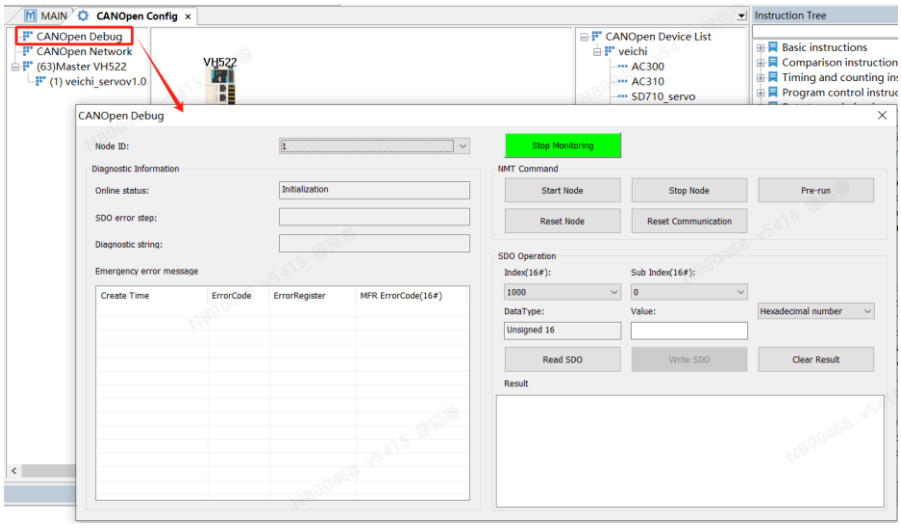
1. Click “Service Data Objects” to access the SDO interface. This interface displays SDO configuration data automatically generated based on user settings.



2. Click “Add” to create a user-defined configuration, primarily used to assign initial values to the slave’s object dictionary.
   Edit: Modify user-defined configurations.
   Delete: Remove user-defined configurations.

9.5.6 Online Debugging

1. Click “CANopen Debug” to access the debugging interface, as shown below:



CANopen Debugging

Item		Description
NMT Command	Start Node	Sends a Start Node command to the slave.
	Stop Node	Sends a Stop Node command to the slave.
	Pre-run	Sends a Pre-run command to the slave.
	Reset Node	Sends a Reset Node command to the slave.
	Reset Communication	Sends a Reset Communication command to the slave.
Service Data Objects (SDO)	Index & Sub-index	Selectable only from the object dictionary entries provided in the slave’s EDS.
	Value	Data to send or received data.
	Bit Length	Automatically generated based on the object dictionary in the EDS (non-editable).
	Result	Error information.
	Read SDO/Write	Executes read/write operations on the object dictionary.

	SDO	
Diagnostic Information	Online Status	Current slave state (based on heartbeat or node protection feedback).
	SDO Error Step	Step number of SDO errors encountered during configuration (corresponds to entries in the SDO tab).
	Diagnostic String	Current error message (SDO error).
	Emergency Error Messages	Displays emergency error frames in the network (real-time monitoring; up to 5 cached in background, but PLC retains only the latest error).

Note:

- If "Prohibit SDO and NMT access during program execution" is enabled in the master, this feature becomes unavailable.

9.6 CANopen Troubleshooting

9.6.1 Troubleshooting Methods

(I) Check Termination Resistors

Power off all devices. Use a multimeter to measure the resistance between CAN_H and CAN_L at either end of the network. The value should be approximately 60 Ω. Too low value indicates multiple termination resistors are incorrectly connected beyond the two ends. Locate and remove the extra resistors. 120 Ω value indicates that only one termination resistor is connected, leading to poor communication quality. No termination resistors will lead to communication fails. Ensure termination resistors are enabled at the first and last nodes in the network.

(II) Check Baud Rate

Verify that all nodes in the network are configured with the same baud rate. Baud rate settings take effect only after power cycling or restarting the devices.

Refer to 9.3.1 Relationship Between Distance and Baud Rate for compatibility guidelines.

(III) Check Wiring

Connect the GND terminals of all CAN devices to ensure a common ground reference.

Inspect for short circuits between communication lines, shielded cables, and power supplies.

(IV) Other Considerations

If severe environmental interference persists, attempt to reduce the baud rate to improve communication stability.

9.6.2 EMCY Error Code

Table 9-1 (Hexadecimal)

EMCY Error Code	Description	EMCY Error Code	Description
00xx	No error	50xx	Device hardware
10xx	General error	60xx	Device software
20xx	Current	61xx	Internal software
21xx	Device input current	62xx	User software
22xx	Device internal current	63xx	Data settings
23xx	Device output current	70xx	Additional modules
30xx	Voltage	80xx	Monitoring
31xx	Power supply voltage	81xx	Communication
32xx	Device internal voltage	82xx	Protocol error
33xx	Output voltage	90**	External error
40xx	Temperature	F0**	Additional functions
41xx	Ambient temperature	FF**	Device-specific
42xx	Device temperature		

Table 9-2 (Hexadecimal)

EMCY Error Code	Description	EMCY Error Code	Description
0000	Error reset or no error	6300	Data settings
1000	General error	7000	Additional module error
2000	Current error	8000	Monitoring error
2100	Device input current	8100	General communication error
2200	Device internal current	8110	CAN communication overload
2300	Device output current	8120	CAN passive mode error
3000	Voltage error	8130	Node protection or heartbeat

			error
3100	Power supply voltage	8140	Bus-off recovery
3200	Device internal voltage	8150	CAN-ID collision
3300	Output voltage	8200	Protocol error
4000	Temperature error	8210	PDO length error
4100	Ambient temperature	8220	Oversize PDO length
4200	Device temperature	8240	Unrecognized SYNC data length
5000	Device hardware error	8250	RPDO timeout
6000	Device software error	9000	External error
6100	Internal software	F000	Additional function error
6200	User software	FF00	Device-specific error

9.7 CANopen Axis Control Instructions

9.7.1 Axis Control Instructions List

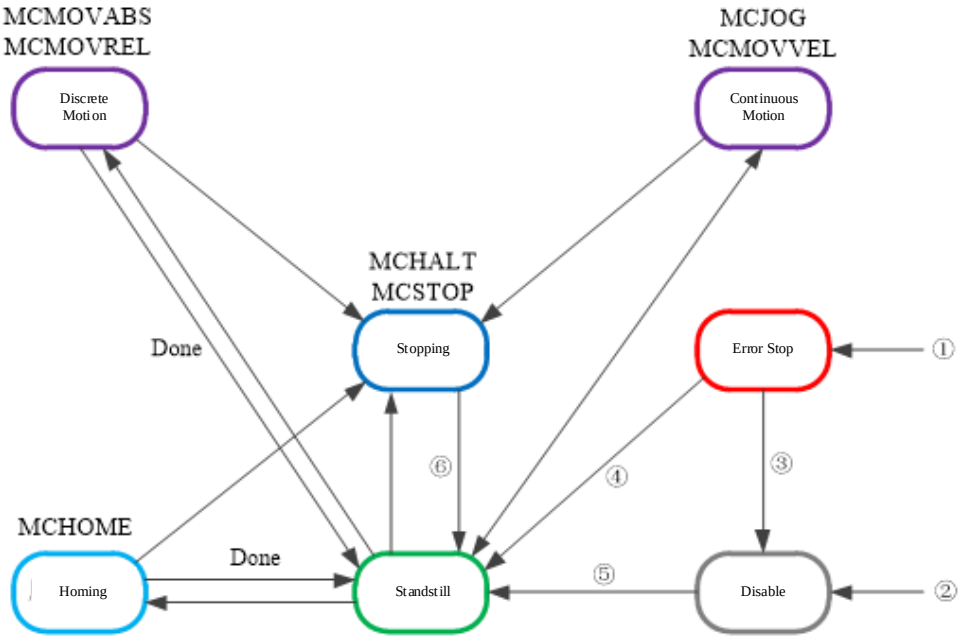
The CANopen axis control module includes two instruction sets: CANopen Integer Axis Control Instructions and CANopen Floating-Point Axis Control Instructions. The integer instructions do not support the [Axis Parameter Setting] function, and homing modes or torque functions require manual SDO parameter configuration.

Instructions	Description	Effective
MCPOWER	Communication-controlled drive enable	/
MCRESET	Communication-controlled drive reset	/
MCSTOP	Communication-controlled drive stop	/
MCHALT	Communication-controlled drive halt	/
MCRDPOS	Communication-read actual position	No
MCRDVEL	Communication-read actual velocity	
MCRDPAR	Communication-read drive parameter	
MCWRPAR	Communication-write drive parameter	
MCHOME	Communication-controlled drive homing	
MCMOVABS	Communication-controlled absolute positioning	
MCMOVREL	Communication-controlled relative positioning	
MCMOVVEL	Communication-controlled velocity mode	
MCJOG	Communication-controlled jogging (velocity mode)	
MCRDPOS_R	Communication-read actual position	Yes Note: When using CANopen Floating-Point Axis Control Instructions, the drive must support uint unit conversion.
MCRDVEL_R	Communication-read actual velocity	
MCRDPAR_R	Communication-read drive parameter	
MCWRPAR_R	Communication-write drive parameter	
MCHOME_R	Communication-controlled drive homing	
MCMOVABS_R	Communication-controlled absolute positioning	
MCMOVREL_R	Communication-controlled relative positioning	
MCMOVVEL_R	Communication-controlled velocity mode	
MCJOG_R	Communication-controlled jogging (velocity mode)	

9.7.2 Axis Control Command State Machine

Axis State Machine

Each servo actuator is treated as a motion control axis. Axis control follows the state machine below.



Axis State Descriptions

State	Description
Disable	Initial state after power-up. Motion commands are invalid, and the servo actuator is disabled. Transition ⑤: Upon valid MCPOWER command, master sends 0x06, 0x07, 0x0F to servo object dictionary (0x6040). Upon completion, servo enters enabled state. Transition ②: For non-fault states, invalid MCPOWER command triggers master to send 0x00 to 0x6040. Transition completes when servo feedback (0x6041) confirms non-operational state. Transition ③: In fault state, MCRESET triggers master to send 0x80 to 0x6040. Transition completes when servo feedback confirms fault reset and disabled state.
Error Stop	Highest priority. Transition ①: Activated when a fault occurs in the axis or servo (402 state machine transitions to fault state) in any state. Some faults do not halt the servo.
Standstill	Servo actuator is enabled and fault-free. No active commands. Transition ④: Executed when MCRESET is applied in fault state while servo remains enabled. Transition ⑥: Activated upon completion of MCSTOP (busy flag cleared).
Stopping	Executing a stop command with configured deceleration.
Discrete Motion	Executing MCMOVABS or MCMOVREL. Master sends 0x0F and 0x1F to 0x6040. The servo operates in PP control mode.
Continuous Motion	Executing MCMOVVEL or MCJOG. Master sends 0x0F and 0x1F to 0x6040. The servo operates in PV control mode.
Homing	Executing MCHOME. Master sends 0x0F and 0x1F to 0x6040. The servo operates in HM control mode.

9.7.3 Error Code Descriptions

Code	Description
0	No error
1	Axis number error. Axis number does not exist in CANopen configuration or PDO configuration error.
2	Instruction parameter error.

	Acceleration/deceleration ≤ 0 for MCMOVABS, MCMOVREL, MCMOVVEL, MCJOG; velocity ≤ 0 for MCMOVABS, MCMOVREL.
3	Parameter (position, home offset) value out of range. ※1
4	Parameter (velocity) value out of range. ※2
5	Parameter (acceleration) value out of range. ※2
6	Parameter (deceleration) value out of range. ※2
8	Instruction interrupted by another instruction, enable loss, or disconnection during execution.
9	Forward overtravel caused instruction termination. ※3
10	Reverse overtravel caused instruction termination. ※3
11	Homing failed.
16	Axis not enabled; thus instruction cannot execute.
17	MCRESET cannot execute unless in "fault stop" state.
18	Axis in "stop" state; instruction cannot execute.
19	Axis is homing; instruction cannot execute.
20	Axis in continuous motion; instruction cannot execute.
21	Axis is positioning; instruction cannot execute.
31	Axis in "fault stop" state; instruction cannot execute. ※3
33	Axis remains in "stop" state or drive disconnected during execution; instruction cannot execute. ※4
250	Axis enable timeout.
251	Servo/motor drive error. ※3
255	Servo/motor drive disconnected. ※3

10 EtherCAT Communication

10.1 Overview

EtherCAT is an open industrial Ethernet-based fieldbus technology featuring short communication refresh cycles, low synchronization jitter, and cost-effective hardware. For EtherCAT principles and technical details, refer to the Design and Application of Industrial Ethernet Fieldbus EtherCAT Drivers or visit the EtherCAT Technology Group official website: https://www.EtherCAT.org.cn. The SH and SH500 series support standard EtherCAT interfaces (1×RJ45 port), accommodate up to 72 EtherCAT slave nodes in a linear topology, and enable a minimum bus cycle time of 1ms.

10.2 EtherCAT Interface Specifications

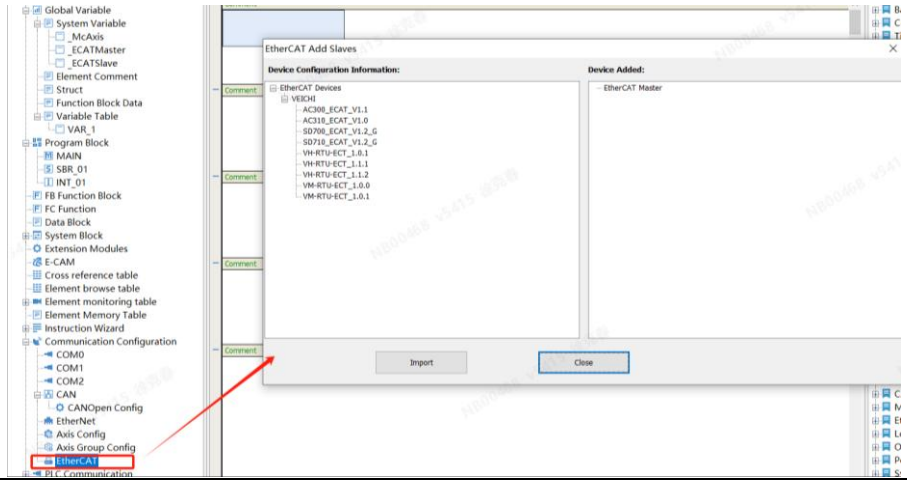
Item	Specification
Transmission speed	100Mbps: 100BASE-TX
Modulation	Baseband
Topology	Line, Daisy-chain
Transmission medium	Shielded twisted pair cables of Category 5 and above with aluminum foil and braided mesh
Transmission distance	Node-to-node distance: ≤100 m
Connection capacity	72

10.3 Master Configuration

10.3.1 Importing Device XML

Importing device XML refers to loading ETG (EtherCAT Technology Group) compliant device description files (.xml) into AutoSoft programming software. After parsing, these files generate configurable EtherCAT devices for user operations like addition or deletion. AutoSoft integrates common SLANVERT EtherCAT slave devices, requiring no separate installation. Third-party EtherCAT devices must have their description files installed first. Example: Importing SLANVERT SD710 drive.

1. Create a new project. In Project Manager area, right-click "EtherCAT"-> Add Device -> Import, as shown below:

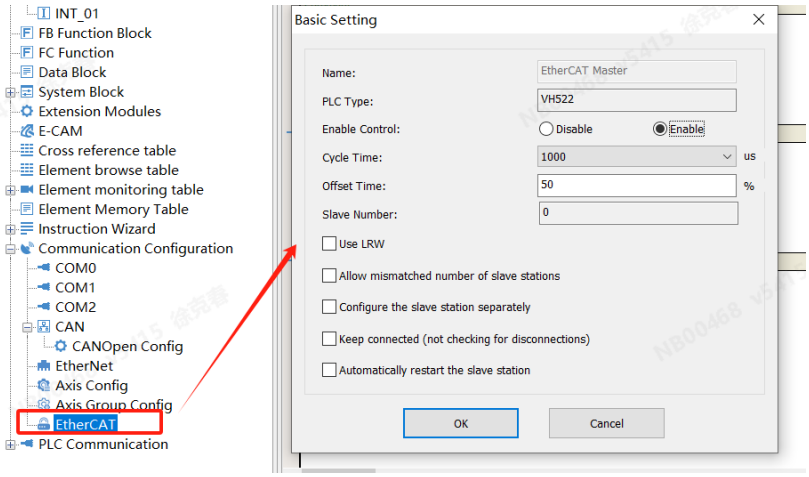


Note:

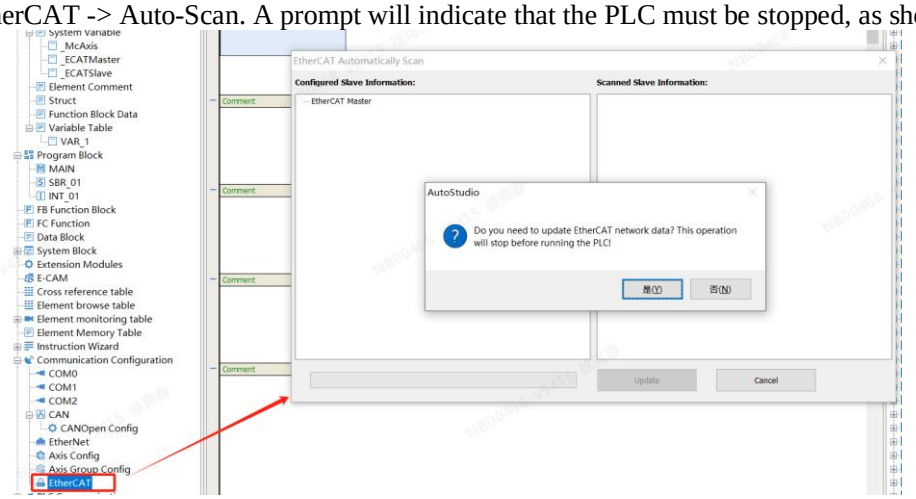
- If XML import fails due to system permission restrictions, manually copy the XML file to the installation directory \Config\xml, then reopen the software.

10.3.2 Scanning Slaves

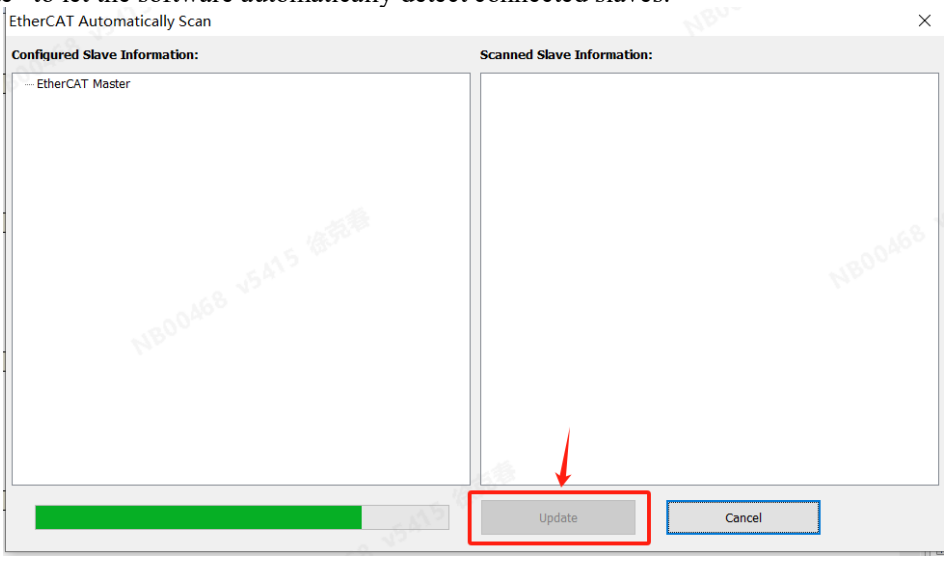
1. Ensure the "Enable" control for the EtherCAT device in the current running PLC project is selected. If unchecked, check it, then compile and download the project once, as shown below:



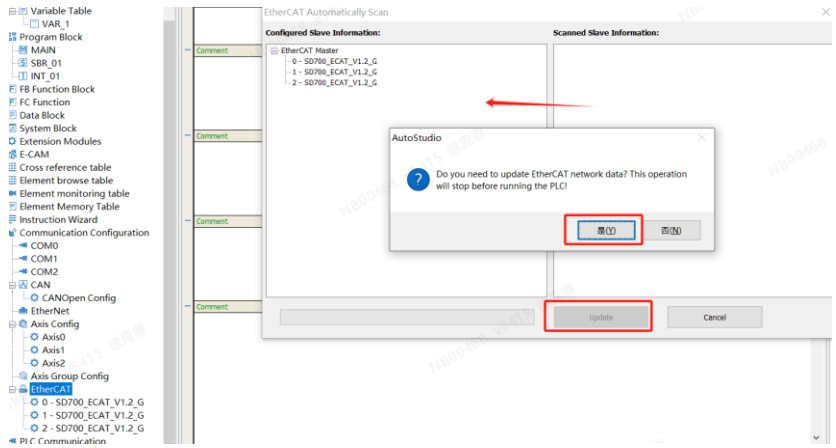
2. Right-click EtherCAT -> Auto-Scan. A prompt will indicate that the PLC must be stopped, as shown below:



3. Click “Update” to let the software automatically detect connected slaves.



4. After completion, the configuration updates as shown below:

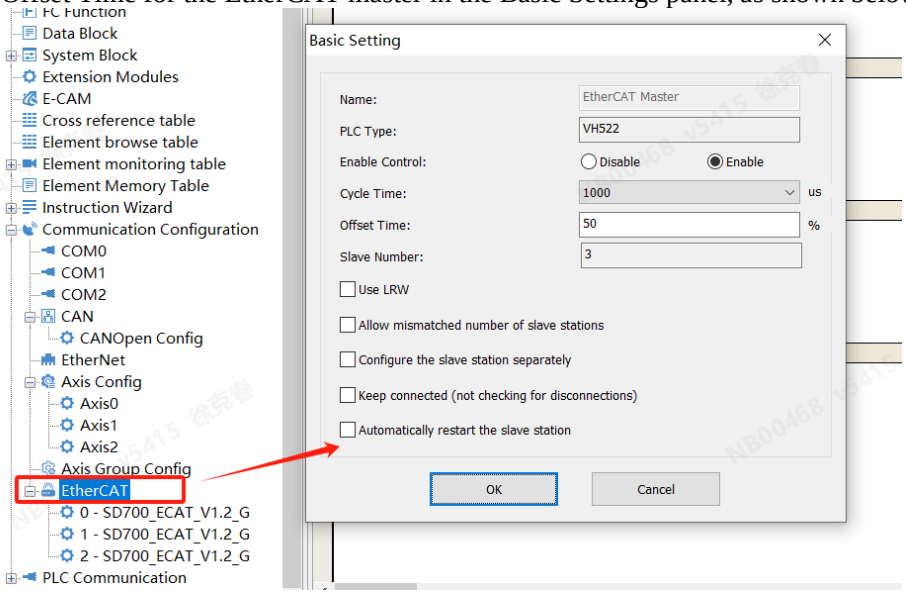


Note:

- EtherCAT slave scanning is only permitted when the PLC is in Stop mode.

10.3.3 Configuring the Master

1. Double-click EtherCAT in Project Manager area. Configure parameters such as Enable Control, Cycle Time, and Offset Time for the EtherCAT master in the Basic Settings panel, as shown below.



Parameter Descriptions:

Parameter	Description
Enable Control	Enable: Activates EtherCAT functionality; Disable: Stops EtherCAT operation.
Cycle Time	Interval for EtherCAT data frame transmission and task cycle time.
Offset Time	Percentage offset of the EtherCAT task relative to the Sync0 interrupt of slaves.
Use LRW instead of LWR/LRD	Default: Unchecked (separate Logical Read/Write). Check to use combined Logical Read-Write. Note: For some slaves only support separate modes, leave unchecked.
Allow mismatched number of slave stations	Check to permit slave initialization when configured slave count mismatched with physically connected slaves. Unchecked: Configured slave counts must match the physically connected slaves. Note: Excess slaves are treated as virtual axes.
Configure the slave station separately	Check to assign separate PDO packets per slave. Unchecked: All slaves share PDO packets. Note: If unchecked, one slave disconnection will disrupt the entire network. If checked, disconnected slaves do not affect others; monitor offline slaves via system parameter _ECATSlave[x].ipackLossSign. X indicates 0~71 station number.
Keep connected (not checking for disconnections)	Check to trigger an alarm (without disconnection handling) for offline slaves. Offline status flags are not updated, preventing specific slave monitoring.
Automatically	Check to auto-restart slaves upon reconnection.

restart the slave station	Note: Enabling this restarts all slaves. To restart only offline slaves, also enable "Configure the slave station separately."

10.3.4 Start/Stop/Disable/Enable

(1) Start/Stop Control

Supports starting and stopping the entire EtherCAT bus but not individual slaves. Operations include:

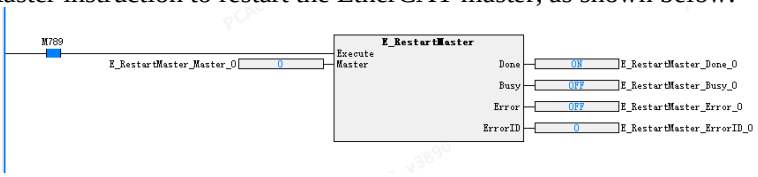
- When the PLC transitions from STOP to RUN mode, the EtherCAT master starts automatically.
- When the PLC transitions from RUN to STOP mode, the EtherCAT master stops automatically.
- The EtherCAT master can be restarted via instructions during PLC operation.

Note:

➢ Only the entire EtherCAT bus can be started/stopped; individual slave control is unsupported.

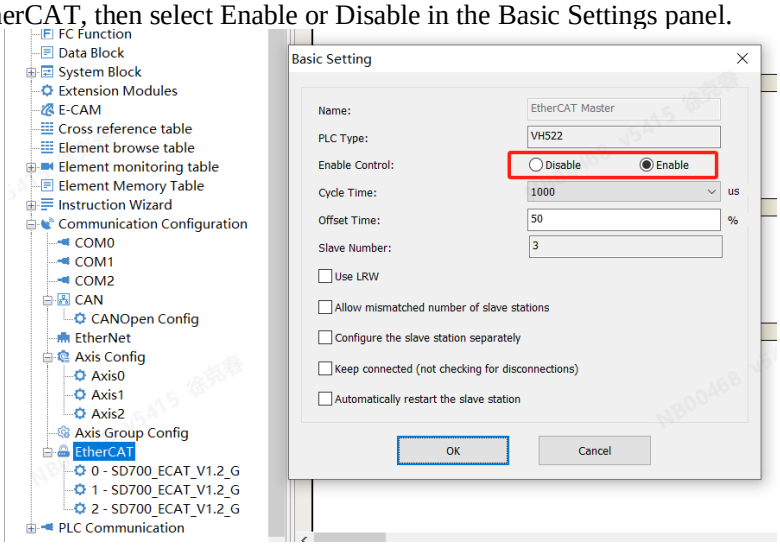
(2) Auto-Restart

Call the E_RestartMaster instruction to restart the EtherCAT master, as shown below:



(3) Disable/Enable

Double-click the EtherCAT, then select Enable or Disable in the Basic Settings panel.



10.3.5 Monitoring Master System Variables

Master status, such as running, stopped, connection state, connected slave count, and current sync cycle time, can be monitored via the system variable _ECATMaster.

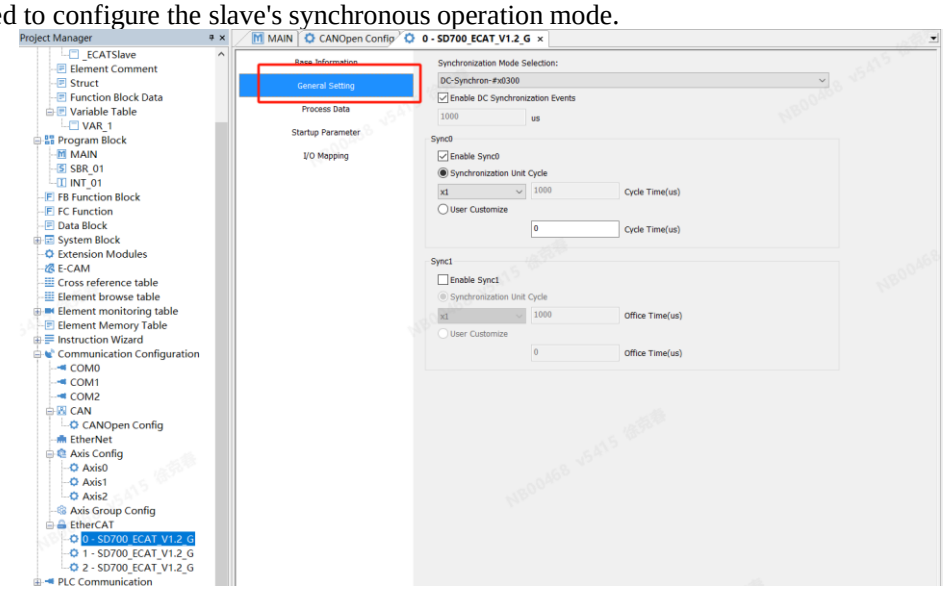
System Variables	Data Type	Function
bMasterEnableState	BOOL	Master enable status. Becomes TRUE when the EtherCAT master receives a run command and all slaves are activated. Note: Remains TRUE even if partial slaves disconnect during operation.
bLinkState	BOOL	Master connection status. Turns ON if at least one slave maintains physical connection; otherwise OFF.
dCycleTime	DINT	Master cycle time (ns). Current configured sync cycle time.
dTaskExeTime	DINT	Master task execution time (ns). Current sync cycle time of the EtherCAT bus.
iMasterState	INT	Master bus state. Initialization states 1/2/4/8; state 8 indicates initialization complete.
iSlaveNumber	INT	Number of connected slaves.
iDcSlaveNumber	INT	Number of DC-supported slaves connected to the master.
iLossPackeCounter	INT	Master packet loss cumulative counter
dPdoInLength	DINT	Master PDO input length (bytes).
dPdoOutLength	DINT	Master PDO output length (bytes).
iCycleJitter	INT	Master PDO output length

iEthercatRun	INT	Master run flag: 0=stop, 1=initialization, 3=running.
iScanReady	INT	Master scan status: 0=not ready, 1=ready.

10.4 Slave Configuration

10.4.1 Distributed Clock

1. It is used to configure the slave's synchronous operation mode.

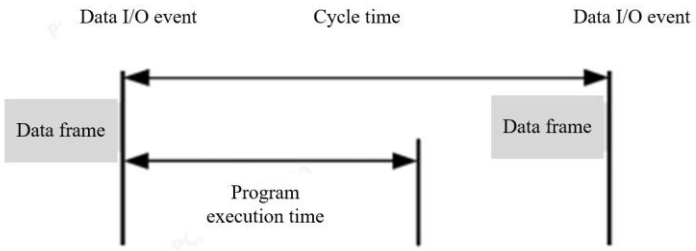


Function Parameter Description:

Sync Mode Selection: Options typically include FreeRun (free-running), SM-Synchron (synchronized to I/O events), and DC-Synchron (synchronized to Distributed Clock). Supported modes vary by slave.

(1) Take SH-ECT coupler as an example:

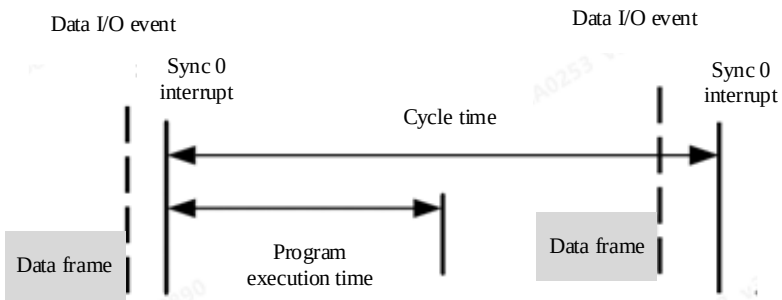
It supports SM-Synchron mode and SYNC0/SYNC1 configurations are unavailable. The slave's internal clock triggers interrupts only for data I/O events. Internal logic is as shown below:



(2) Take SD710 drive as an example:

It supports DC-Synchron mode only and SYNC interrupts are configurable.

Default settings: DC sync event enabled, SYNC0 interrupt enabled (cycle matches master cycle time), and SYNC1 interrupt disabled.



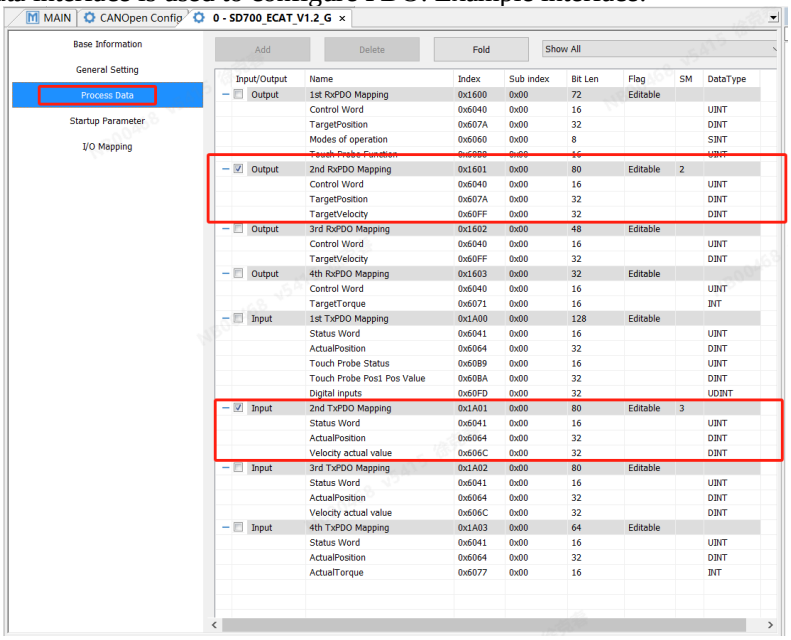
Note:

- In DC-Synchron mode, avoid modifying default configurations unless fully understanding EtherCAT

communication principles.

10.4.2 Process Data

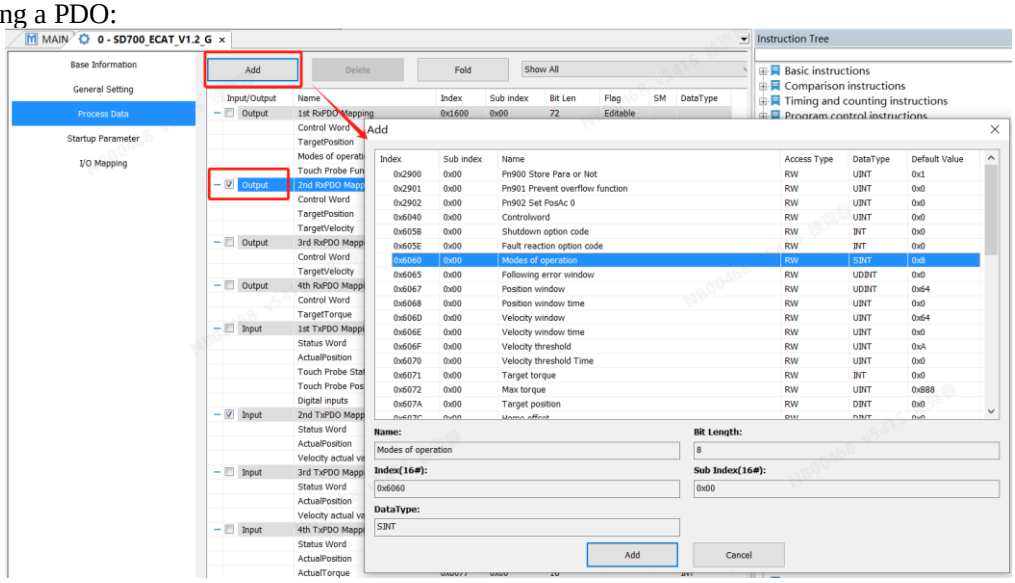
1. The Process Data interface is used to configure PDO. Example interface:



PDOs are categorized by data direction:

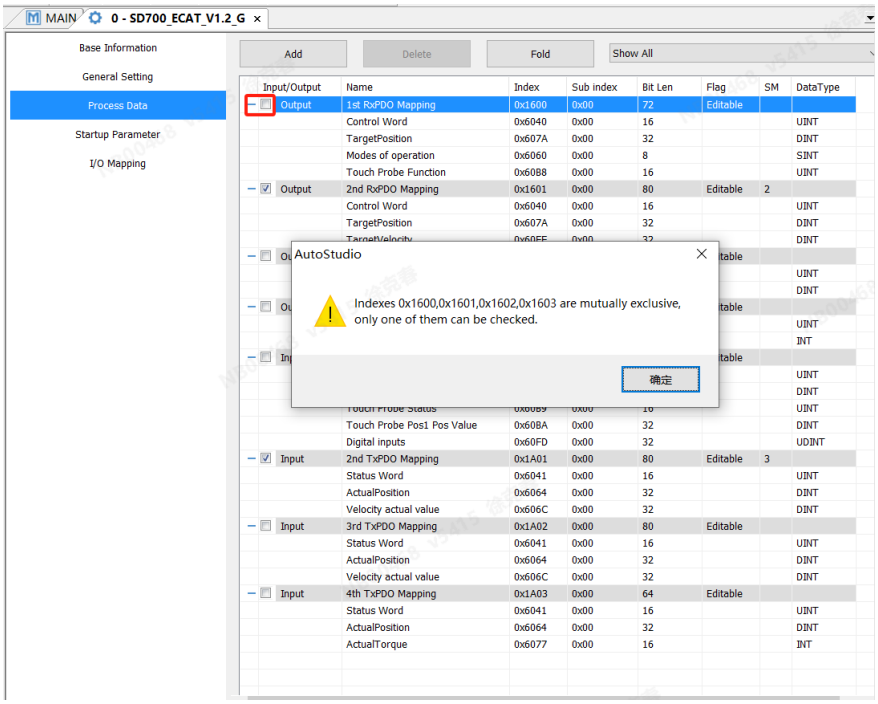
- Output PDO: Data sent from EtherCAT master to slave (e.g., control word 0x6040).
- Input PDO: Data sent from EtherCAT slave to master (e.g., status word, feedback speed/position).
- A slave may have single or multiple PDO groups. As shown above, the second input/output PDO groups can be edited (add/modify/delete).

2. Adding a PDO:



① Select a PDO in the second group.
② Click Add.
③ Select 0×6060.
④ Click OK.

3. When a slave has multiple PDO groups, mutual exclusion relationships may exist between groups (e.g., SD710) where only one group can be selected at a time. These mutual exclusion relationships vary depending on the slave.

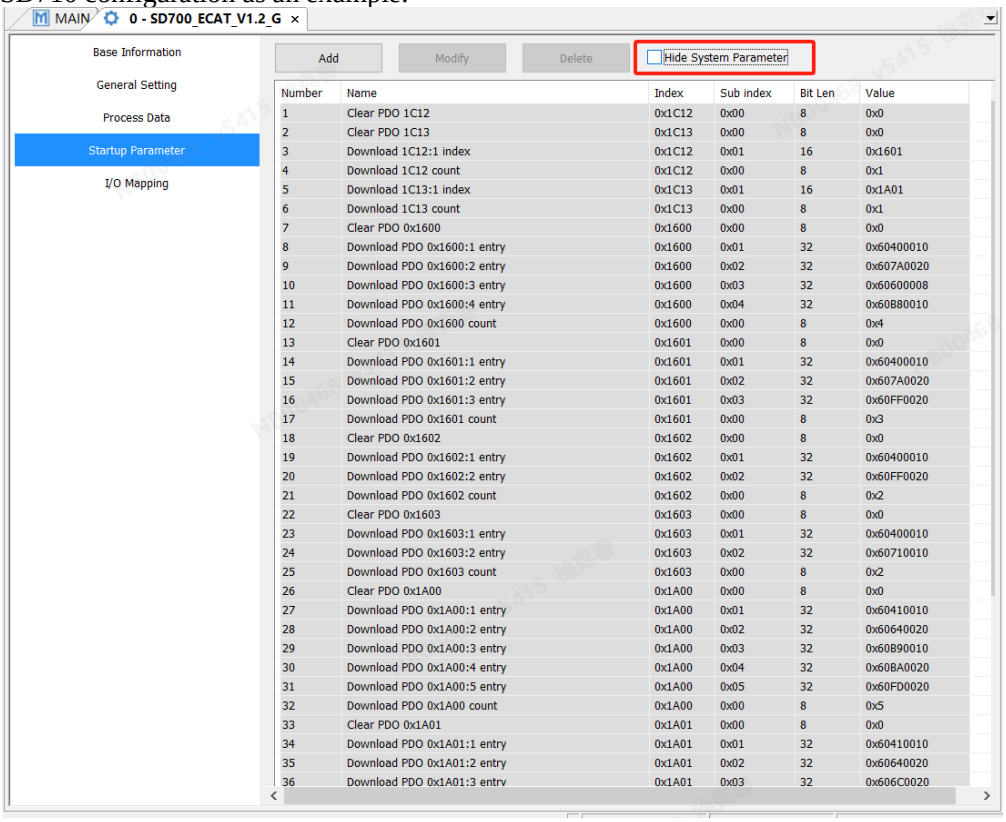


The master configures PDO into slaves through PDO Assignment and PDO Mapping by downloading these settings as startup parameters.

"PDO Assignment" selects specific PDO group numbers for download to the slave, while "PDO Mapping" configures editable internal PDO entries within the selected group. Failure to enable both options during PDO modifications may cause slave startup failures. These configurations can be verified in the startup parameters list.

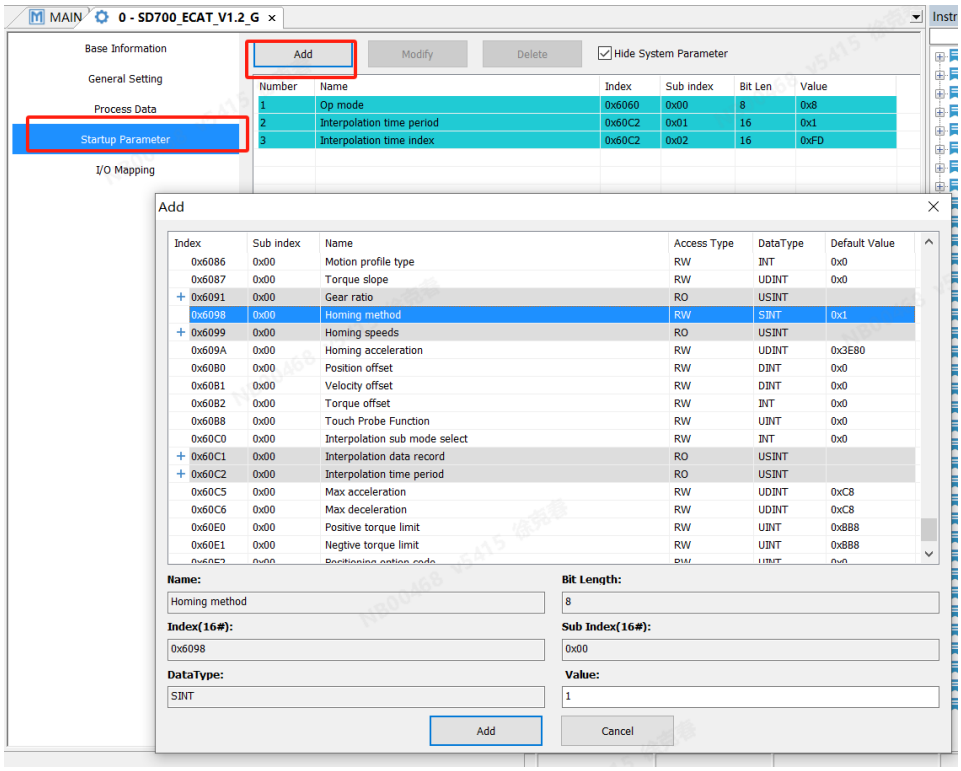
10.4.3 Startup Parameters

1. Startup parameters are used to configure PDO settings, manufacturer-specific parameters, and protocol-defined parameters (e.g., 402 protocol) into the slave via SDO writes while the slave is in PreOP state. Take SD710 configuration as an example:



Number	Name	Index	Sub index	Bit Len	Value
1	Clear PDO 1C12	0x1C12	0x00	8	0x0
2	Clear PDO 1C13	0x1C13	0x00	8	0x0
3	Download 1C12:1 index	0x1C12	0x01	16	0x1601
4	Download 1C12 count	0x1C12	0x00	8	0x1
5	Download 1C13:1 index	0x1C13	0x01	16	0x1A01
6	Download 1C13 count	0x1C13	0x00	8	0x1
7	Clear PDO 0x1600	0x1600	0x00	8	0x0
8	Download PDO 0x1600:1 entry	0x1600	0x01	32	0x60400010
9	Download PDO 0x1600:2 entry	0x1600	0x02	32	0x607A0020
10	Download PDO 0x1600:3 entry	0x1600	0x03	32	0x60600008
11	Download PDO 0x1600:4 entry	0x1600	0x04	32	0x60B80010
12	Download PDO 0x1600 count	0x1600	0x00	8	0x4
13	Clear PDO 0x1601	0x1601	0x00	8	0x0
14	Download PDO 0x1601:1 entry	0x1601	0x01	32	0x60400010
15	Download PDO 0x1601:2 entry	0x1601	0x02	32	0x607A0020
16	Download PDO 0x1601:3 entry	0x1601	0x03	32	0x60FF0020
17	Download PDO 0x1601 count	0x1601	0x00	8	0x3
18	Clear PDO 0x1602	0x1602	0x00	8	0x0
19	Download PDO 0x1602:1 entry	0x1602	0x01	32	0x60400010
20	Download PDO 0x1602:2 entry	0x1602	0x02	32	0x60FF0020
21	Download PDO 0x1602 count	0x1602	0x00	8	0x2
22	Clear PDO 0x1603	0x1603	0x00	8	0x0
23	Download PDO 0x1603:1 entry	0x1603	0x01	32	0x60400010
24	Download PDO 0x1603:2 entry	0x1603	0x02	32	0x60710010
25	Download PDO 0x1603 count	0x1603	0x00	8	0x2
26	Clear PDO 0x1A00	0x1A00	0x00	8	0x0
27	Download PDO 0x1A00:1 entry	0x1A00	0x01	32	0x60410010
28	Download PDO 0x1A00:2 entry	0x1A00	0x02	32	0x60640020
29	Download PDO 0x1A00:3 entry	0x1A00	0x03	32	0x60B90010
30	Download PDO 0x1A00:4 entry	0x1A00	0x04	32	0x60BA0020
31	Download PDO 0x1A00:5 entry	0x1A00	0x05	32	0x60FD0020
32	Download PDO 0x1A00 count	0x1A00	0x00	8	0x5
33	Clear PDO 0x1A01	0x1A01	0x00	8	0x0
34	Download PDO 0x1A01:1 entry	0x1A01	0x01	32	0x60410010
35	Download PDO 0x1A01:2 entry	0x1A01	0x02	32	0x60640020
36	Download PDO 0x1A01:3 entry	0x1A01	0x03	32	0x606C0020

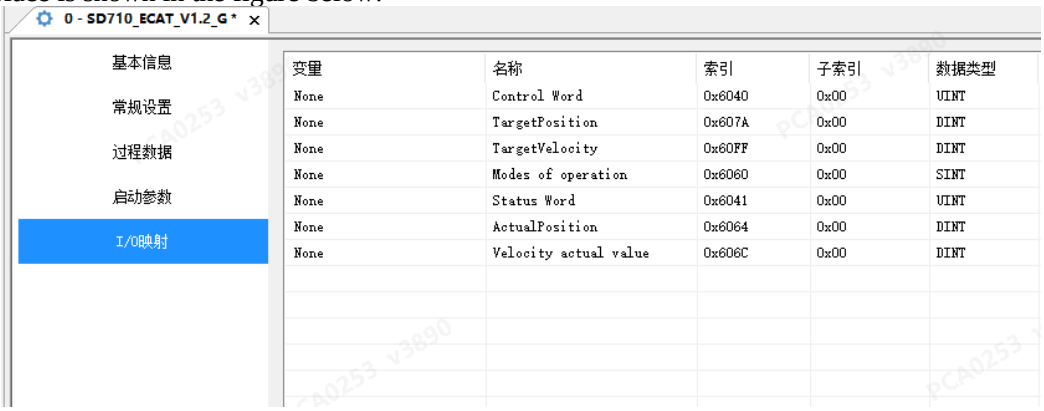
2. Add object dictionary entry 0x6098 to the "Startup Parameters" and set its value to 35, as illustrated below:



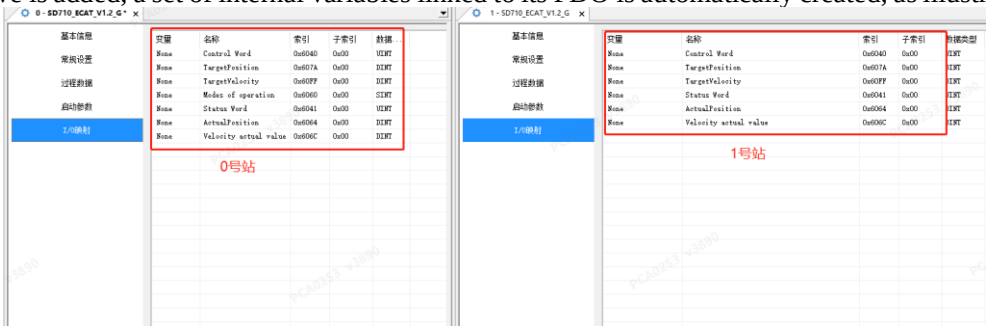
① Click Add
② Select 0x6098
③ Modify the value to 35
④ Click OK

10.4.4 I/O Mapping

PDO data must be mapped to PLC internal variables to control EtherCAT slaves via these variables. The I/O Mapping interface is shown in the figure below:



When a slave is added, a set of internal variables linked to its PDO is automatically created, as illustrated below:



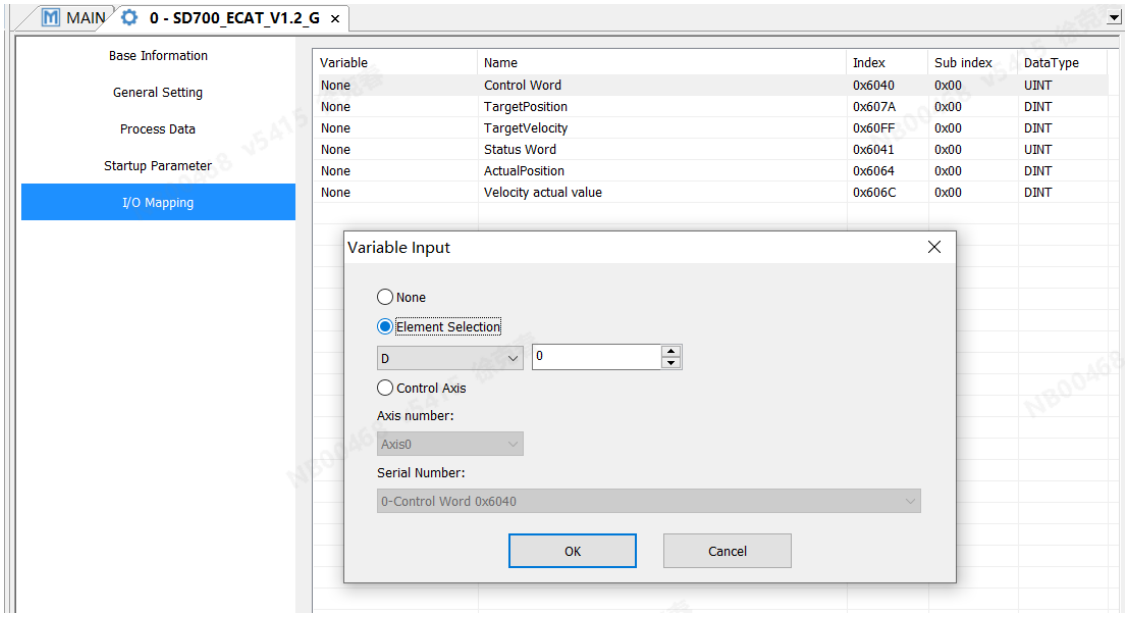
Note:

- If a slave is associated with a motion control axis (e.g., SD710), its variables can only be controlled via axis commands. The mapped registers in I/O Mapping serve solely as monitoring addresses.

➢ For drives, data exchange with the slave is performed through the mapped register addresses in I/O Mapping.

Associated Variables

Double-click "None", select a D/R/W variable type and enter the address in the pop-up dialog, and click OK, as shown below:



10.4.5 Slave System Variables

System variables related to EtherCAT slaves are listed below:

System Variables	Data Type	Description
iState	INT	Slave current bus state 1:INT 2:PreOP 4:SafeOP 8:OP Note: In OP state (8), this value remains unchanged even if the slave disconnects.
iALstatescode	INT	AL states code. Indicates state machine transition failure (see slave manual).
iConfigaddr	INT	Configuration address. Default: starts at 30000 + connection sequence number (e.g., 30001 for the first slave).
iAliasaddr	INT	Slave alias address. Default: 0 (currently unsupported).
dEep_man	DINT	Slave equipment manufacturer ID
dEep_id	DINT	Slave equipment ID
iItype	INT	Interface type
iDtype	INT	Device type
iObits	INT	PDO output bits
iObytes	INT	PDO output bytes
iOstartbit	INT	PDO output start bit
iIbits	INT	PDO input bits
iIbytes	INT	PDO input bytes
iIstartbit	INT	PDO input start bit
iHasdc	INT	DC support
iPtype	INT	PHY interface type
iTopology	INT	Topology
iActiveports	INT	Active ports
iParent	INT	Parent slave ID
iParentport	INT	Parent port ID
iEntryport	INT	Entry port
iSlotConfig	INT	Slot configuration flag. Note: Set to 1 for SH-ECT couplers or multi-drop slaves; 0 for single slaves.
iRdSlotsNum	INT	Read slot number. Displays the number of connected slots.
dRdSlotsIds[31]	Array	Read slot ID. Displays module IDs in connected slots in order.

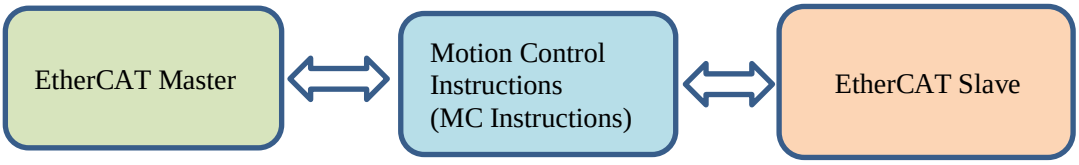
dEep_rev	DINT	Slave equipment version
iPackLossCounter	INT	Cumulative packet loss counter. Increments continuously after slave disconnection.
iPackLossSign	INT	Pack loss sign. Set to 1 if the slave is offline.

11 EtherCAT Motion Control

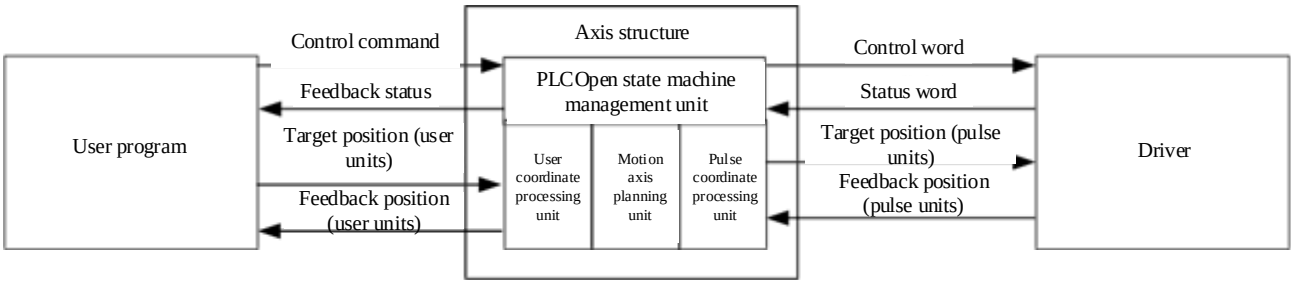
11.1 Overview

11.1.1 Basic Structure and Control Logic

In motion control systems, the controlled object is referred to as an axis. An axis acts as a bridge between the drive and PLC instructions. Motion control axes are used to control EtherCAT bus drives compliant with the 402 protocol, as well as local high-speed pulse outputs and inputs.



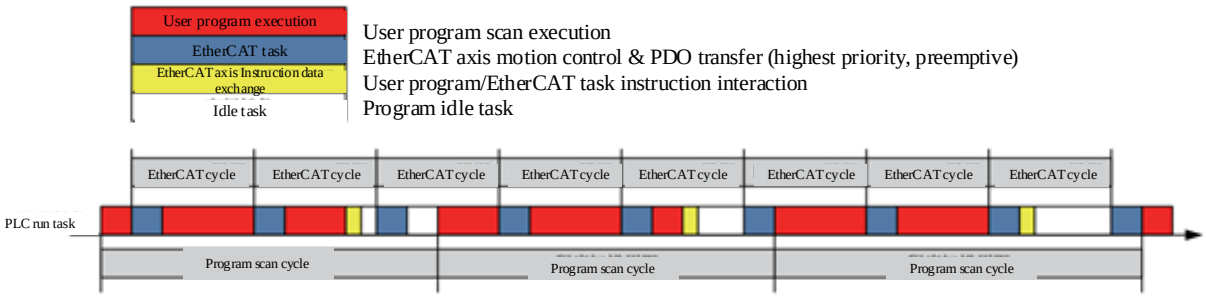
Within the PLC, the basic structure and processing logic of an axis are as follows:



11.1.2 Motion Instruction Scheduling Mechanism

The Main program, subroutines, and interrupt routines are available for user programming. However, motion control instructions can only be executed in the Main program or subroutines and must not be called within interrupt routines.

The EtherCAT task is a hidden task and not accessible to users. Therefore, programming within the EtherCAT task is not supported.



11.1.3 Axis Type Configuration

Supported axis types include bus servo axis, local pulse axis, bus encoder axis (not supported), and local encoder axis.

Axis Type	Description
Bus servo axis	Axis controlled via EtherCAT slave servo drives. Supports torque, position, velocity, homing, and other basic control modes. Virtual axis mode is supported.
Local pulse axis	Supports 4×local pulse axes: Y0/Y1, Y2/Y3, Y4/Y5, Y6/Y7.

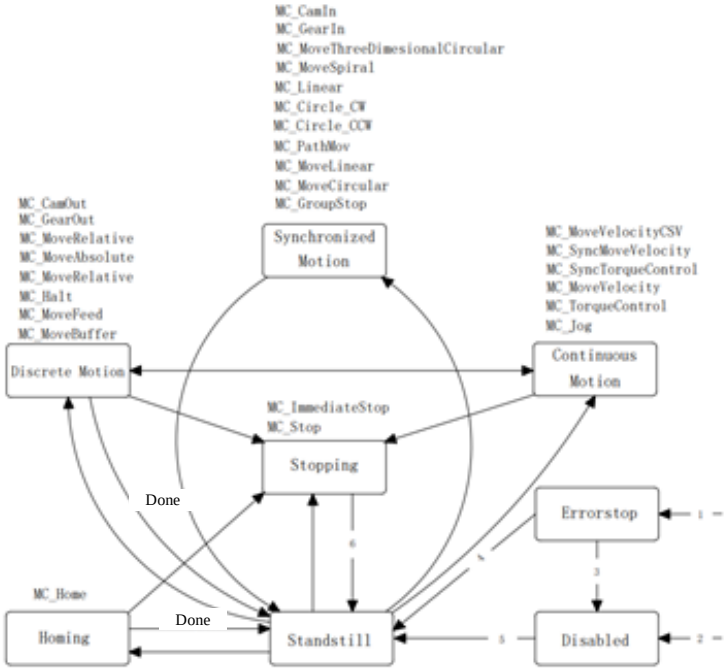
	● Controls pulse-driven axes using local high-speed I/O. Pulse output modes: Pulse + direction, CW/CCW, AB phase. Probe: 2×probe inputs per pulse axis. Motion control: Supports positioning, linear/circular interpolation, and homing functions.
Bus encoder axis	Reserved
Local encoder axis	Supports 8×single-phase (200 kHz) or 4×AB-phase high-speed (200 kHz) inputs. Counting modes: Linear axis or rotary axis. Probe: 8×probe inputs total. Note: Probe and counter ports are shared; only one function can be used per port.

A properly configured axis involves three components:

Component	Function
Axis configuration parameters	Configures axis settings, such as mode, gear ratio, homing type, and encoder mode.
Axis system variables	Monitors axis status and errors (e.g., current position, fault codes).
Axis control instructions	Executes motion control via MC instructions in user programs. MC instructions are categorized into: Management (e.g., MC_Power), Motion (e.g., MC_Jog), and Status (e.g., MC_ReadStatus).

11.1.4 PLCOpen State Machine

The PLCOpen State Machine manages axis states and motion by executing specific functions in each state. The state transition diagram is shown below:



State Descriptions

State Value	State	Description
0	Disabled	Axis is inactive.
1	ErrorStop	Axis stopped due to an error.
2	Stopping	Axis is decelerating to a standstill.
3	Standstill	Axis is enabled and stationary.
4	DiscreteMotion	Axis is in discrete motion.
5	ContinuousMotion	Axis is in continuous motion.
7	Homing	Axis is performing homing.
8	SynchronizedMotion	Axis is synchronized with other axes or processes.

11.1.5 Axis Parameter Description

Axis parameter configuration consists of three parts: User Units & Axis Configuration Parameters, Axis System Variables, and Axis Control Instructions. These components are described below:

User Units

Two units are used in the axis structure: User Unit and Pulse Unit.

Unit	Description
User unit	Measurement units, such as millimeters, centimeters, and degrees, used in instruction operations, denoted as Uint. User coordinate systems are categorized into Linear Coordinate and Rotational Coordinate based on application requirements: Linear coordinate: Includes a zero point. Increasing the target position represents positive movement; decreasing it represents negative movement. Supports positive/negative soft limits. Rotational coordinate: Includes a zero point and a cycle period. Within one cycle, increasing the target position represents clockwise motion; decreasing it represents counterclockwise motion.
Pulse unit	Unit based on pulse count used on the drive side, denoted as pluse. Drive parameters typically include a pulse zero point and encoder pulses per motor rotation.

Axis Configuration Parameters

Configure motion control axis properties to meet application requirements. The parameters are summarized below:

Type	Parameter	Bus Servo Axis	Local Pulse Axis
Basic Settings	Axis number	✓	✓
	Axis type	✓	✓
	Associated device (Not associated = virtual axis)	✓	✓
	Auto mapping	✓	×
	PDO	✓	×
Unit Conversion	REV	✓	✓
	Pulses per motor/encoder revolution	✓	✓
	Workpiece movement per revolution (Background)	✓	✓
	Gear ratio numerator	✓	✓
	Gear ratio denominator	✓	✓
Mode/Parameters	Encoder mode	✓	×
	Linear/Rotary mode	✓	✓
	Software limits	✓	✓
	Follow error	✓	✓
	Axis velocity	✓	✓
	Torque limit	✓	×
	Probe	×	✓
	Output	×	✓

The homing mode parameters are configured as follows:

Type	Parameter	Bus Servo Axis	Local Pulse Axis
Homing Settings	Home signal	✓	✓
	Positive limit	✓	✓
	Negative limit	✓	✓
	Z-signal	✓	×
	Homing direction	✓	✓
	Home input polarity	✓	✓
	Homing method	✓	✓
	Homing velocity	✓	✓
	Homing approach velocity	✓	✓
	Homing acceleration	✓	✓
	Homing timeout	✓	✓
	Negative limit input	×	✓
	Positive limit input	×	✓
	Home signal configuration	×	✓

Axis System Variables

In the program, the current status of an axis can be monitored through its system variables. The system variables for bus servo axes/local pulse axes are listed in the table below:

System Variables	Data Type	Description
dPulsesPreCycle	DINT	Pulses per revolution (motor/encoder)
fDistancePreCycle	REAL	Workpiece movement per revolution
dNumerator	DINT	Gear ratio numerator
dDenorminator	DINT	Gear ratio denominator
bDirection	BOOL	Direction
bSoftLimitEnable	BOOL	Software limit enable
fPLimit	REAL	Positive limit in linear mode
fNLimit	REAL	Negative limit in linear mode
iLineRotateMode	INT	Linear/rotational mode selection 0: linear; 1: rotational
fRotation	REAL	Cycle in rotational mode
EncodeMode	INT	Encoder mode (valid in bus servo axis) 1: incremental; 0: absolute
iHomeMethod	INT	Homing method
fHomeVelocity	REAL	Homing velocity
fHomeApproachVelocity	REAL	Homing approach velocity
fHomeAcceleration	REAL	Homing acceleration
iHomeTimeOut	INT	Homing timeout
bPLimitTerminalPolarity	BOOL	Positive limit terminal polarity (valid in local pulse axis)
bNLimitTerminalPolarity	BOOL	Negative limit terminal polarity (valid in local pulse axis)
bHomeTerminaPolarity	BOOL	Home terminal polarity (valid in local pulse axis)
iPLimitType	INT	Positive limit input type (valid in local pulse axis) 0: X, 1: M, 2: S
iPLimitID	INT	Positive limit input number (valid in local pulse axis) X0~7/M0~M/S0~S
iNLimitType	INT	Negative limit input type (valid in local pulse axis) 0: X, 1: M, 2: S
iNLimitID	INT	Negative limit input number (valid in local pulse axis)
iHomeInType	INT	Home input type (valid in local pulse axis) 0: X, 1: M, 2: S
iHomeInID	INT	Home input number (valid in local pulse axis)
iEncoderInType	INT	Local encoder input type (valid in local encoder)
iEncoderRstInEn	INT	Local encoder reset input enable (valid in local encoder)
iEncoderRstInID	INT	Local encoder reset input ID (valid in local encoder)
iEncoderEnInEn	INT	Local encoder enable input enable (valid in local encoder)
iEncoderEnInID	INT	Local encoder enable input ID (valid in local encoder)
iEncoderPreSetInEn	INT	Local encoder preset input enable (valid in local encoder)
iEncoderPreSetInID	INT	Local encoder preset input ID (valid in local encoder)
iPluseMethod	INT	Pulse output method (valid in local pulse axis)
bTouchProbeEn0	BOOL	Touch probe 0 input enable (valid in local pulse axis)
iTouchProbeID0	INT	Touch probe 0 ID (valid in local pulse axis) 0~7: X0~7
bTouchProbeEn1	BOOL	Touch probe 1 input enable (valid in local pulse axis)
iTouchProbeID1	INT	Touch probe 1 ID (valid in local pulse axis) 0~7: X0~7
bCmpEnable	BOOL	Comparison output enable (valid in local pulse axis)
iCmpOutID	INT	Comparison output port ID (valid in local pulse axis) 0~7: Y0~7
iCmpOutUnit	INT	Comparison output unit (valid in local pulse axis)
iCmpOutWidth	INT	Comparison output width (valid in local pulse axis)
fErrorStopDeceleration	REAL	Axis error stop deceleration
fFollowErrorWindow	REAL	Follow error window
fMaxVelocity	REAL	Maximum velocity
fMaxJerkVelocity	REAL	Maximum jerk velocity
fMaxAcc	REAL	Maximum acceleration
iMaxTorque	INT	Maximum torque
iMapped	INT	Axis parameter mapping flag

iType	INT	Axis type
iSlave	INT	Axis mapping ID
iVirtualAxis	INT	Virtual axis flag
iEnableStatus	INT	Axis enable status
iAlmStatus	INT	Axis alarm status
iAxisOprationStatus	INT	Axis operation status
iCheckDoingStatus	INT	Axis motion status 0: stopped 1: running
iInterpNum	INT	Interpolation channel number
iInterpBitType	INT	Control mode flag bits bit0: P_Task, bit1: S_Task, bit2: F_Task, bit3: cam
dCommandPulse	DINT	Current command position
hEncoderCounter	DINT	Encoder high-order counter
dEncoderPos	DINT	Encoder feedback position

11.1.6 Axis Control Instructions

Single-axis control instructions are listed below. For detailed usage, refer to the SH Series PLC Instruction Manual.

Instruction	Name
MC_Power	Enable axis control
MC_Reset	Reset fault
MC_Home	Homing
MC_Homing	Axis homing
MC_Stop	Stop
MC_MoveVelocity	Velocity control
MC_Jog	Jog
MC_Move	Positioning
MC_ReadAxisError	Read axis error
MC_ReadPosistion	Read actual position
MC_ReadStatus	Read axis status
MC_TorqueControl	Torque control
MC_SetPosition	Set position
MC_MoveSuperImposed	Superimposed motion
MC_TouchProbe	Touch probe
MC_Linear	Linear interpolation
MC_Circle_CW	Clockwise circular interpolation
MC_Circle_CCW	Counter-clockwise circular interpolation
MC_MoveBuffer	Multi-segment move
MC_MoveAbsolute	Absolute move
MC_MoveRelative	Relative move
MC_ReadVelocity	Read actual velocity
MC_MoveFeed	Feed-interrupt move
MC_Halt	Motion halt
MC_SyncTorqueControl	Synchronized torque control
MC_ReadActualTorque	Read actual torque
MC_FollowVelocity	Velocity superposition control
MC_ReadDigitalInput	Read digital inputs
MC_MoveVelocityCSV	CSV-based variable pulse width velocity control
MC_SyncMoveVelocity	CSV-based synchronized velocity control
MC_MoveThreeDimensionalCircular	Three dimensional circular interpolation
MC_MoveSpiral	Spiral interpolation
MC_SetAxisConfigPara	Axis parameter configuration
MC_SetAxisBackLash	Set backlash compensation
MC_GetAxisBackLash	Get backlash compensation status

11.1.7 Online Modification of Axis Configuration Parameters

- Users can modify configuration parameters of individual axes via the PLC program to meet different

application requirements, such as software limits and rotation period in ring mode.

- When modifying axis configuration parameters in the PLC program, use the _McAxis system structure variable to set parameters. This structure data is not retained after power loss. Upon PLC restart, axis configuration parameters from the software backend will initialize this structure variable.
- Modify the initialized variable values as needed and execute the MC_SetAxisConfigPara instruction to validate the settings.

Note:

- Instruction execution requires completion of EtherCAT bus initialization. Modifications made before initialization may result in parameter configuration failures.
- When modifying parameters during operation, set the MC_Power instruction's power flow to OFF.
- Axis configuration parameters are non-retentive variables, thus previously set values will be lost after PLC reboot. Reconfigure parameters after each PLC startup.

Structure _McAxis

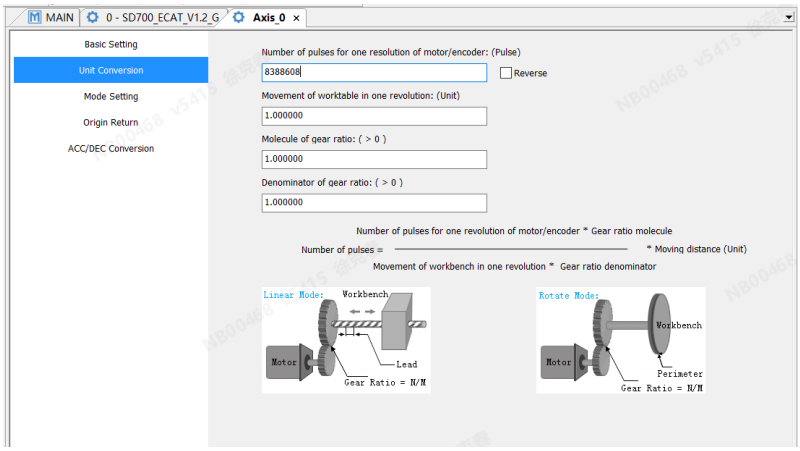
Name	Data Type	R/W	Description	Index
_McAxis[0]	Struct	/	/	/
_McAxis[0].dPulsesPreCycle	DINT	WR	Pulses per cycle (motor/encoder)	1
_McAxis[0].fDistancePreCycle	REAL	WR	Workpiece movement per cycle	
_McAxis[0].dNumerator	DINT	WR	Gear ratio numerator	
_McAxis[0].dDenorminator	DINT	WR	Gear ratio denominator	
_McAxis[0].bDirection	BOOL	WR	Direction	
_McAxis[0].bSoftLimitEnable	BOOL	WR	Software limit enable	2
_McAxis[0].fPLimit	REAL	WR	Positive limit in linear mode	
_McAxis[0].fNLimit	REAL	WR	Negative limit in linear mode	
_McAxis[0].iLineRotateMode	INT	WR	Linear/Rotary mode 0: linear; 1: rotational	
_McAxis[0].fRotation	REAL	WR	Cycle in rotational mode	
_McAxis[0].EncodeMode	INT	WR	Encoder mode (valid in bus servo axis) 1: absolute 0: incremental	3
_McAxis[0].iHomeMethod	INT	WR	Homing method	4
_McAxis[0].fHomeVelocity	REAL	WR	Homing velocity	
_McAxis[0].fHomeApproachVelocity	REAL	WR	Homing approach velocity	
_McAxis[0].fHomeAcceleration	REAL	WR	Homing acceleration	
_McAxis[0].iHomeTimeOut	INT	WR	Homing timeout	
_McAxis[0].bPLimitTerminalPolarity	BOOL	WR	Positive limit terminal polarity (valid in local pulse axis)	5
_McAxis[0].bNLimitTerminalPolarity	BOOL	WR	Negative limit terminal polarity (valid in local pulse axis)	
_McAxis[0].bHomeTerminaPolarity	BOOL	WR	Home terminal polarity (valid in local pulse axis)	
_McAxis[0].iPLimitType	INT	WR	Positive limit input type 0: X, 1: M, 2: S (valid in local pulse axis)	
_McAxis[0].iPLimitID	INT	WR	Positive limit input number X0~7/M0~Mxx/S0~Sxx (valid in local pulse axis)	
_McAxis[0].iNLimitType	INT	WR	Negative limit input type 0: X, 1: M, 2: S (valid in local pulse axis)	
_McAxis[0].iNLimitID	INT	WR	Negative limit input number (valid in local pulse axis)	
_McAxis[0].iHomeInType	INT	WR	Home input type 0: X, 1: M, 2: S (valid in local pulse axis)	
_McAxis[0].iHomeInID	INT	WR	Home input number (valid in local pulse axis)	

_McAxis[0].iEncoderInType	INT	WR	Encoder input type (valid in local encoder)	
_McAxis[0].iEncoderRstInEn	INT	WR	Local encoder reset input enable (valid in local encoder)	
_McAxis[0].iEncoderRstInID	INT	WR	Local encoder reset input number (valid in local encoder)	
_McAxis[0].iEncoderEnInEn	INT	WR	Local encoder enable input enable (valid in local encoder)	
_McAxis[0].iEncoderEnInID	INT	WR	Local encoder enable input number (valid in local encoder)	
_McAxis[0].iEncoderPreSetInEn	INT	WR	Local encoder pre-set input enable (valid in local encoder)	
_McAxis[0].iEncoderPreSetInID	INT	WR	Local encoder pre-set input number (valid in local encoder)	
_iPluseMethod	INT	WR	Pulse output method (valid in local pulse axis)	6
_McAxis[0].iPluseMethod	INT	WR	Pulse output method (valid in local pulse axis)	7
_McAxis[0].bTouchProbeEn0	BOOL	WR	Touch probe 0 input enable (valid in local pulse axis)	
_McAxis[0].iTouchProbeID0	INT	WR	Touch probe 0 input number 0~7: X0~7 (valid in local pulse axis)	
_McAxis[0].bTouchProbeEn1	BOOL	WR	Touch probe 1 input enable (valid in local pulse axis)	
_McAxis[0].iTouchProbeID1	INT	WR	Touch probe 1 input number 0~7: X0~7 (valid in local pulse axis)	
_McAxis[0].bCmpEnable	BOOL	WR	Comparison output enable (valid in local pulse axis)	8
_McAxis[0].iCmpOutID	INT	WR	Comparison output port number 0~7: Y0~7 (valid in local pulse axis)	
_McAxis[0].iCmpOutUnit	INT	WR	Comparison output unit (valid in local pulse axis)	
_McAxis[0].iCmpOutWidth	INT	WR	Comparison output width (valid in local pulse axis)	
_McAxis[0].fErrorStopDeceleration	REAL	WR	Axis error stop deceleration	9
_McAxis[0].fFollowErrorWindow	REAL	WR	Follow error window	
_McAxis[0].fMaxVelocity	REAL	WR	Maximum velocity	
_McAxis[0].fMaxJerkVelocity	REAL	WR	Maximum jerk velocity	
_McAxis[0].fMaxAcc	REAL	WR	Maximum acceleration	
_McAxis[0].iMaxTorque	INT	WR	Maximum torque	
_McAxis[0].dConfigReserved[16]	DINT	WR	Reserved	
_McAxis[0].iMapped	INT	RO	Axis parameter mapping flag	/
_McAxis[0].iType	INT	RO	Axis type	/
_McAxis[0].iSlave	INT	RO	Axis mapping ID	/
_McAxis[0].iVirtualAxis	INT	RO	Virtual axis flag	/
_McAxis[0].iEnableStatus	INT	RO	Axis enable status	/
_McAxis[0].iAlmStatus	INT	RO	Axis alarm status 0x0001: Hardware positive limit 0x0002: Hardware negative limit 0x0004: Software positive limit	/

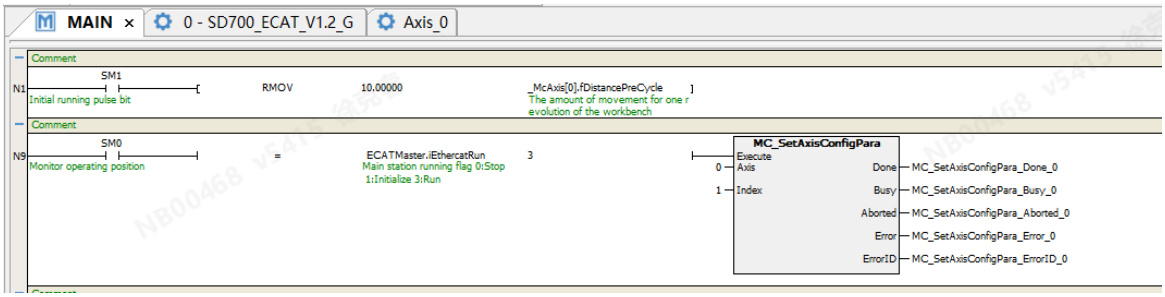
			0x0008: Software negative limit 0x0010: Position deviation error	
_McAxis[0].iAxisOprationStatus	INT	RO	Axis operation status	/
_McAxis[0].iCheckDoingStatus	INT	RO	Axis motion status 0: stop, 1: running	/
_McAxis[0].iInterpNum	INT	RO	Interpolation channel number	/
_McAxis[0].iInterpBitType	INT	RO	Control mode flags bit0: P_Task, bit1: S_Task, bit2: F_Task, bit3: cam	/
_McAxis[0].dCommandPulse	DINT	RO	Current command position	/
_McAxis[0].dDeltaCommand	DINT	RO	Feedback-to-command position deviation	/
_McAxis[0].dEncoderPos	DINT	RO	Encoder feedback position	/
_McAxis[0].dStatusReserved[16]	DINT	RO	Reserved	/

Axis Parameter Modification Example

1. Original axis configuration parameters use default values. Modify the axis configuration parameter from 1 Unit to 10 Unit via command.



2. Develop a PLC program to modify the travel distance per revolution of Axis 0's worktable to 10 Units. Execute the MC_SetAxisConfigPara command to validate the modifications. Trigger the parameter modification command after EtherCAT bus initialization is completed. (Note: For the relationship between Index values and axis parameters, refer to the "Structure _McAxis" table above.)



11.2 Motion Control Axis Configuration

11.2.1 Axis Type

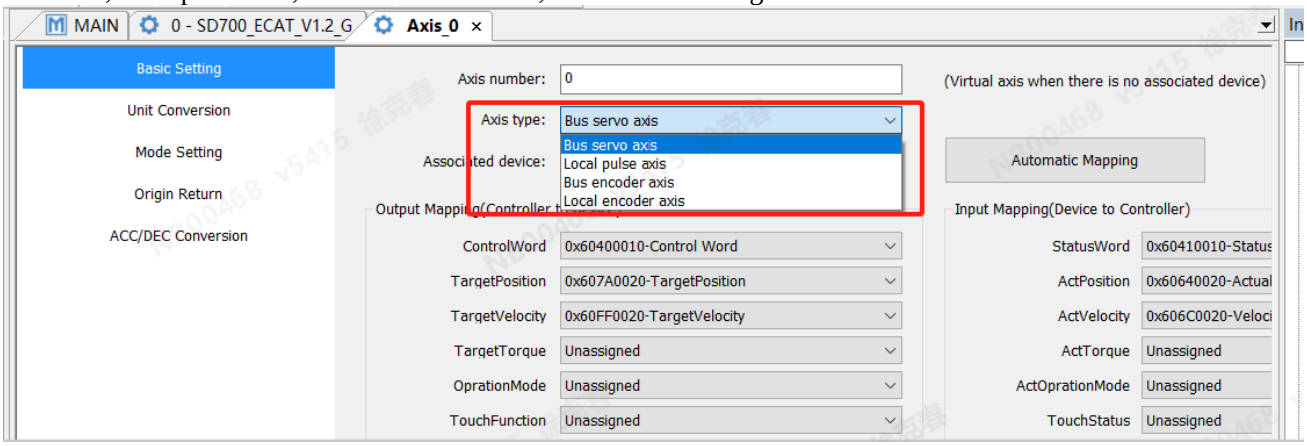
Both bus servo axes and local pulse axes are controlled using the same set of MC instructions and share an identical axis structure design. The following outlines the differences between them.

Item	Local Pulse Axis	Bus Drive
Axis type	Select local pulse axis	Select bus servo axis
Associated devices	Assign output ports Y0/Y1, Y2/Y3, Y4/Y5, Y6/Y7	Configure PDO mapping to relevant variables
Pulse output form	Pulse+direction, CW/CCW, AB phase	\
Probe	Supports 2 channels of probe	Configure probe terminals according to EtherCAT drive manual

Homing	Supports homing methods specified in 402 protocol (excluding Z signal)	Supports homing methods 1~35 specified in 402 protocol Limit and home signals must be connected to the drive

11.2.2 Basic Settings

1. The Basic Setting interface is used to set the axis type, confirming whether the axis functions as a bus servo axis, local pulse axis, or local encoder axis, as shown in the figure below:



Parameter Descriptions

- Axis Number: Each axis is assigned a unique identifier (0~71) and cannot be manually modified. The axis number serves as the input parameter for MC instructions to access the axis.
- Axis Type: Options include bus servo axis, local pulse axis, bus encoder axis (not supported), and local encoder axis.
- Associated Device: For bus servo axis, select the EtherCAT servo drive. For local pulse axis, assign local high-speed output terminals (Y0/Y1, Y2/Y3, etc.). For local encoder axis, assign local high-speed input terminals. If unassigned, the axis operates as a virtual axis.
- Auto Mapping: Valid only for bus servo axis. EtherCAT slaves use PDO-based cyclic communication, where the axis connects to the slave's object dictionary via cyclic variables. Enabling auto-mapping assigns mappings automatically; manual configuration is disabled.

For details on the object dictionary parameters of bus servo axes, please refer to the standard 402 protocol.

Loop Variable	Object Directory	Function
Controlword	0x6040	Control word
TargetPosition	0x607A	Target position value
TargetVelocity	0x60FF	Target velocity value
Settorque	0x6071	Target torque
Modesofoperation	0x6060	Control mode
Touchprobefunction	0x60b8	Probe mode
Physicaloutputs	0x60fe:1	Physical outputs
StatusWord	0x6041	Status word
ActPosition	0x6064	Actual position
ActVelocity	0x606C	Actual velocity
Torqueactualvalue	0x6077	Actual torque value
Modesofoperationdisplay	0x6061	Mode of operation display
Digitalinputs	0x60fd	Digital input status
TouchProbeStatus	0x60b9	Touch probe status
TouchProbePos1PosValue	0x60ba	Touch probe 1 rising edge position
TouchProbePos1NegValue	0x60bb	Touch probe 1 falling edge position
TouchProbePos2PosValue	0x60bc	Touch probe 2 rising edge position
Touchprobe2fallingedge	0x60bd	Touch probe 2 falling edge position
Errorcode	0x603f	Error code

11.2.3 Unit Conversion

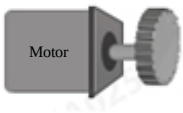
Item	Function
Pulses per motor/encoder revolution	Sets the pulse count per motor/encoder revolution based on encoder resolution.
Workpiece movement per	Defines the workpiece movement per revolution.

revolution	
Gear ratio numerator	Numerator of the gear reduction ratio.
Gear ratio denominator	Denominator of the gear reduction ratio.

For bus drives (local pulse axes), motor control uses pulse units, while motion control instructions operate in common measurement units (e.g., millimeters, degrees, inches), referred to as user units (Unit). Based on configuration parameters, the axis internally converts between these units. The conversion modes are categorized as follows:

(I) Linear Axis Mode

(1) Without a gear reduction mechanism, the conversion formula from user units to pulse units is as follows:



$$\text{Pulse} = \frac{\text{Pulses per motor/encoder revolution[DINT]*Gear ratio numerator[DINT]}}{\text{Workpiece movement per revolution[REAL]*Gear ratio denominator[DINT]}} * \text{Displacement[Uint]}$$

Example with a 17-bit encoder drive:

Set the parameters as:

Pulses per motor/encoder revolution = 131072

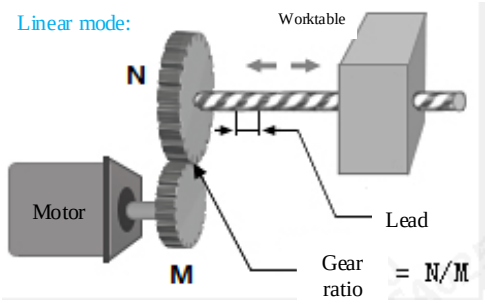
Workpiece movement per revolution = 1

Gear ratio numerator = 1

Gear ratio denominator = 1

When a relative position command specifies a target displacement of 10 user units, the motion control axis sends 1310720 pulses, resulting in 10 motor revolutions.

(2) With a gear reduction mechanism, the conversion formula from user units to pulse units is as follows:



$$\text{Pulse} = \frac{\text{Pulses per motor/encoder revolution[DINT]*Gear ratio numerator[DINT]}}{\text{Workpiece movement per revolution[REAL]*Gear ratio denominator[DINT]}} * \text{Displacement[Uint]}$$

Example with a 17-bit encoder drive:

Set the parameters as:

Pulses per motor/encoder revolution = 131072

Workpiece movement per revolution = 1

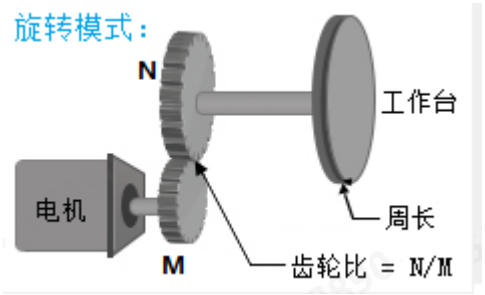
Gear ratio numerator = 3 (N)

Gear ratio denominator = 1 (M)

When a relative position command specifies a target displacement of 1 user unit, the motion control axis sends 393216 pulses, resulting in 3 motor revolutions and 1 workpiece revolution (1 lead).

(II) Rotary Axis Mode

The conversion formula from user units to pulse units is as follows:



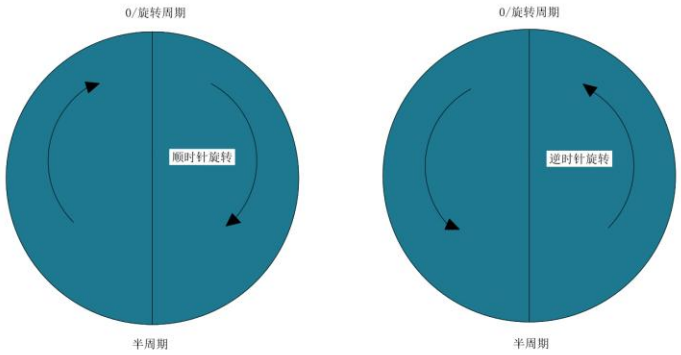
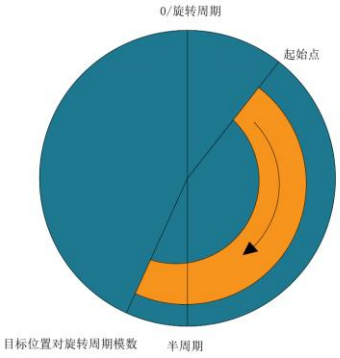
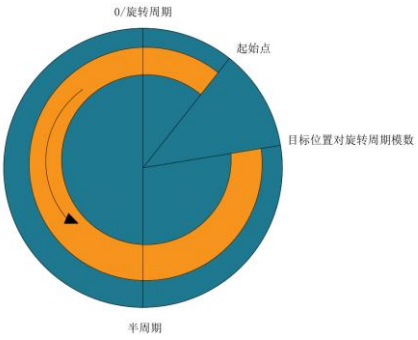
$$\text{Pulse} = \frac{\text{Pulses per motor/encoder revolution[DINT]*Gear ratio numerator[DINT]}}{\text{Workpiece movement per revolution[REAL]*Gear ratio denominator[DINT]}} * \text{Displacement[Uint]}$$

11.2.4 Mode Setting

1. Basic settings interface varies depending on selected axis type. Visible parameter lists differ accordingly.

Parameter Description

Item	Description
Encoder Mode	Encoder mode is valid only in bus servo axis mode, used with incremental or absolute encoder servo drives. Select based on the actual servo drive type. PLC processing modes: **Incremental mode:** The PLC ignores overflow-triggered revolution increments in the servo drive's 32-bit encoder counter. The PLC does not retain the encoder's current position during power cycles. Upon restart, the axis calculates its current position solely using the single-turn position feedback from the servo drive. **Absolute mode:** The PLC tracks overflow-induced revolution increments in the 32-bit encoder counter. The PLC stores the encoder's current position during power cycles. Upon restart, the axis calculates the absolute position by reconciling the stored encoder position with real-time servo feedback. After the axis is enabled, the drive position (6064h) is converted and synchronized to the command position.
	Linear mode 1. Linear mode is typically used in devices with mechanical motion ranges in X-Y Cartesian coordinate systems. 2. A zero point is usually defined in linear mode. 3. Increasing feedback position indicates forward motion; decreasing feedback position indicates reverse motion. 4. Forward and reverse software limits can be configured. When enabled, the axis can only operate within the defined range. Absolute positioning: If the target position > starting position, move forward by the distance (target – starting position). If the target position < starting position, move reverse

Mode Setting	by the distance (starting – target position). Relative positioning: For target displacement > 0: move forward by the displacement distance. For target displacement < 0: move reverse by the absolute value of the displacement. Velocity command in linear mode: For target velocity > 0: move forward. For target velocity < 0: move reverse.
	Ring mode ● Ring mode uses a cyclic counter that repeats infinitely within a defined range, commonly applied in turntables or spools. ● A zero point and rotation period are typically defined. The feedback position range is 0 ≤ feedback position < rotation period. ● In this mode, increasing feedback position indicates clockwise (CW) motion; decreasing feedback position indicates counterclockwise (CCW) motion.  **Software limits are disabled in ring mode.** Relative positioning: For target displacement > 0: CW motion by the displacement distance. For target displacement < 0: CCW motion by the absolute value of the displacement. Absolute positioning: Forward: Apply modulo operation on the target position relative to the rotation period, and move CW from the starting position to the target.  Reverse: Apply modulo operation on the target position, and move CCW from the starting position to the target.  Shortest path: Compute the target position modulo the rotation period, and calculate the CW displacement from the starting position. If displacement ≤ half the rotation period, move CW; for else, move CCW. Current direction: Follow the axis's last motion direction. For first-time power-on, move CW to the target. Velocity instruction in ring mode: For target velocity > 0, move CW; For target velocity < 0, move CCW.

11.2.5 Software Limit

When the soft limit is enabled, the system continuously monitors the axis' absolute position during motion. If the projected stopping position (calculated by T-curve deceleration from the current speed using the specified limit deceleration) exceeds the limit range, the axis will trigger the soft limit deceleration algorithm and interrupt ongoing positioning/speed commands. Soft limits are inactive during homing mode and torque mode.

11.2.6 Error Deceleration

If a motion command's internal logic failure during axis operation necessitates switching to the error stop state, the axis will perform T-curve deceleration using the configured error deceleration until it decelerates to zero.

11.2.7 Following Error Threshold

During positioning/speed commands execution, the servo drive operates in CSP (Cyclic Synchronous Position) mode with trajectory planning performed on the PLC side. The PLC sends target positions via 0x607A to the servo drive, while the motor encoder's actual position is fed back through 0x6064. The discrepancy between 0x607A and 0x6064, converted into user units, is defined as the following error. The SH500 configures the maximum allowable following error. If the absolute following error exceeds this threshold, the axis triggers an excessive following error fault and enters the errorstop state.

11.2.8 Axis Velocity

Set the maximum velocity, maximum acceleration, and maximum jerk. If any parameter (target velocity, acceleration, or deceleration) in positioning or velocity commands exceeds these limits, the command triggers a fault and the axis enters the ErrorStop state.

11.2.9 Maximum Torque

Maximum torque setting is exclusively applicable to bus-servo axes.

If the target torque in torque commands exceeds the maximum torque limit, the command triggers a fault and the axis enters ErrorStop state.

Positive/negative torque limit values are configured via startup parameters in the servo drive's object dictionary (0x60E0 and 0x60E1).

11.2.10 Probe

Local pulse axes can enable probe terminals through probe configuration. Bus axes require configuring relevant PDO.

For local pulse axes, each axis supports up to two probe terminals. Probe terminal sources can be selected from X0~X7.

After enabling probe terminals, local pulse axes can utilize probe instructions and interrupt fixed-length instructions.

11.2.11 Pulse Output Mode Configuration

Local pulse axes can be configured as four channels using Y0/Y1, Y2/Y3, Y4/Y5, or Y6/Y7.

They support output in Pulse+Direction, CW/CCW, or AB-phase formats.

For channels configured as pulse axes: In Pulse+Direction mode: Y0, Y2, Y4, Y6 are designated as pulse terminals; Y1, Y3, Y5, Y7 are direction terminals. In CW/CCW mode: Y0, Y2, Y4, Y6 are CW pulse terminals; Y1, Y3, Y5, Y7 are CCW terminals.

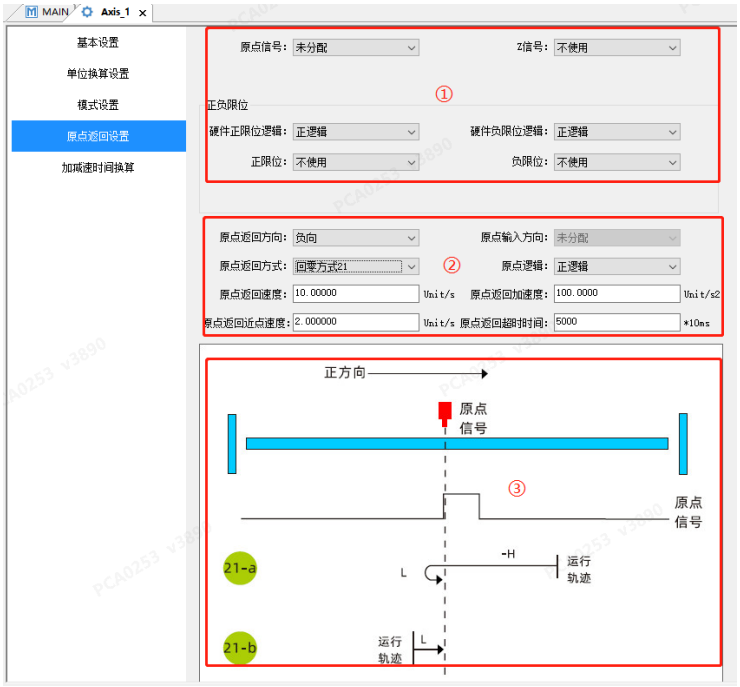
11.2.12 Hardware Limit

The axis system variables include bphlimit (hardware positive limit) and bnhlimit (hardware negative limit) indicating hardware limit states. When set to positive logic, these variables correspond to bits 1 and 0 of object dictionary 0x60FD, respectively. When configured as negative logic, their values are inverted from bits 1 and 0 of 0x60FD.

Hardware limit logic settings only affect these variables and have no impact on the servo's limit-triggered stop behavior.

11.2.13 Origin Return

The SH500 supports CiA402-compliant origin return methods 1~35, as shown below.



Origin return configuration interface:

Parameter	Description
Origin signal	Enables/disables use of the origin signal. Unassigned: Not enforced as a mandatory filter. Not Used: Excludes returning methods requiring a origin signal. Use: Excludes returning methods incompatible with a origin signal.
Z Signal	Enables/disables use of the motor Z signal. Unassigned: Not enforced as a mandatory filter. Not Used: Excludes returning methods requiring a Z signal. Use: Excludes returning methods incompatible with a Z signal.
Positive Limit	Enables/disables use of the hardware right limit signal. Unassigned: Not enforced as a mandatory filter. Not Used: Excludes returning methods requiring a positive limit signal. Use: Excludes returning methods incompatible with a negative limit signal.
Negative Limit	Enables/disables use of the hardware left limit signal. Unassigned: Not enforced as a mandatory filter. Not Used: Excludes returning methods requiring a negative limit signal. Use: Excludes returning methods incompatible with a negative limit signal.
Origin Return Direction	Initial movement direction during origin return. Positive direction: Moves positive if limit/origin signal is inactive; otherwise, moves negative. Negative direction: Moves negative if limit/origin signal is inactive; otherwise, moves positive.
Origin Input Direction	Direction when triggering origin signal. Positive direction: Stops upon triggering origin signal edge during positive motion. Negative direction: Stops upon triggering origin signal edge during negative motion.
Origin Logic	Positive logic (high level)/Negative logic (low level).
Origin Return Method	Range: Method 1~35. Written to object dictionary 0x6098 via startup parameters.
Origin Return Speed	User unit converted to pulse units, written to sub-index 1 of object dictionary 0x6099 via startup parameters.
Origin Return Acceleration	User unit converted to pulse units, written to object dictionary 0x609A via startup parameters.
Near Point Speed	User unit converted to pulse units, written to sub-index 2 of object dictionary 0x6099 via startup parameters.
Origin Return Timeout	Unit: 10ms.

In practice, origin return methods are filtered by parameters including origin signal, positive/negative limits, Z signal, origin return direction, and origin input direction, then selected via the origin return method option.

Note that after applying filter criteria, multiple origin return methods may remain available. Select the

appropriate method from the origin return method list. For example, the settings below filter two origin return methods:

Signal	Selection
Origin Signal	Use
Negative Limit	Not Used
Positive Limit	Use
Z Signal	Not Used
Origin Return Direction	FWD
Origin Input Direction	FWD

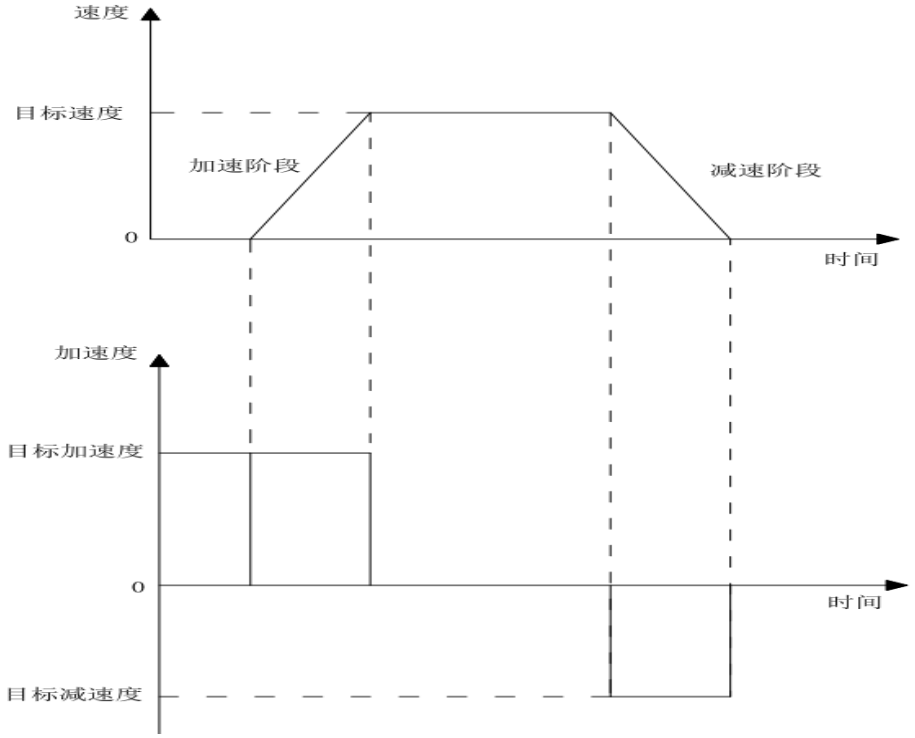
Choose the required method (20, 24, or 26) based on field application requirements.

11.2.14 Curve Type

The system supports two velocity profiles: T-curve acceleration/deceleration and 5-segment S-curve acceleration/deceleration, determined by the SpeedMode parameter in instructions. Additionally, when axes encounter limits or require error-triggered deceleration to enter Errorstop state, T-curve deceleration is applied.

(I) T-curve Velocity Profile

When SpeedMode=0, axes execute T-curve acceleration/deceleration. The motion profile is planned based on target position, velocity, acceleration, and deceleration. During acceleration/deceleration phases, the actual acceleration/deceleration rates remain constant, as shown below.



- Target position: Final absolute position in user units (Unit) for absolute positioning commands.
- Target velocity: Maximum achievable velocity during motion in user units per second (Unit/s).
- Target acceleration: Velocity change rate during acceleration in user units per second squared (Unit/t²).
- Target deceleration: Velocity change rate during deceleration in user units per second squared (Unit/t²).
- Acceleration phase: If initial velocity = Vs, target velocity = Vt, and acceleration =Acc, then the acceleration time:

Tacc=(Vt-Vs)/Acc

- Deceleration phase: If initial velocity = Vs, target end velocity = Ve, and deceleration = Dec, then the deceleration time:

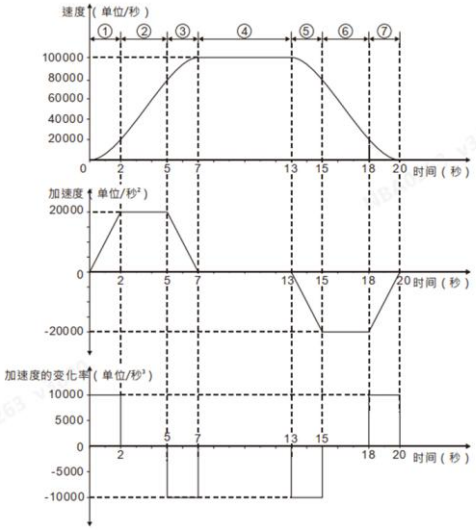
Tdec=(Vs-Ve)/Dec

(II) S-Curve Velocity Profile

When SpeedMode=1, the axis follows an S-curve acceleration/deceleration motion. This profile reduces mechanical shock by shaping the velocity waveform into an S-curve. The motion control instruction must specify at least velocity (v), acceleration (Acc) or deceleration (Dec), and jerk (Jerk):

- Velocity: Maximum axis speed during operation, in "units/s".
- Acceleration: Maximum acceleration during operation, in "units/s²".

- Jerk: Maximum rate of acceleration/deceleration change, in "units/s³". The specified Jerk value in instruction applies to both acceleration and deceleration phases. Adjusting Jerk can improve velocity smoothness.



The relationship between velocity, acceleration, and jerk is illustrated in the following table:

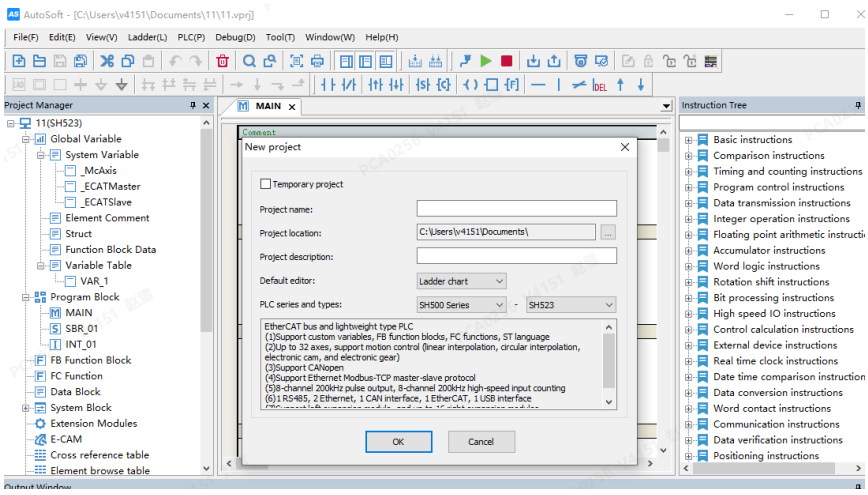
Phase	Time (s)	Jerk (units/s³)	ACC/DEC (units/s²)	Velocity (units/s)	Motion Type
1	0~2	Fixed at 10000	Acceleration increases to 20000	Velocity increases	Acceleration-increasing accelerated motion
2	2~5	Fixed at 0	Acceleration remains at 20000	Velocity increases	Accelerated motion with constant acceleration
3	5~7	Fixed at -10000	Acceleration decreases to 0	Increases to 100000	Acceleration-decreasing accelerated motion
4	7~13	Fixed at 0	Acceleration remains at 0	Constant at 100000	Uniform motion
5	13~15	Fixed at -10000	Deceleration increases to 20000	Velocity decreases	Deceleration-increasing decelerated motion
6	15~18	Fixed at 0	Deceleration remains at 20000	Velocity decreases	Decelerated motion with constant deceleration
7	18~20	Fixed at 10000	Deceleration decreases to 0	Decreases to 0	Deceleration-decreasing decelerated motion

11.3 Quick Setup Example for EtherCAT Axis

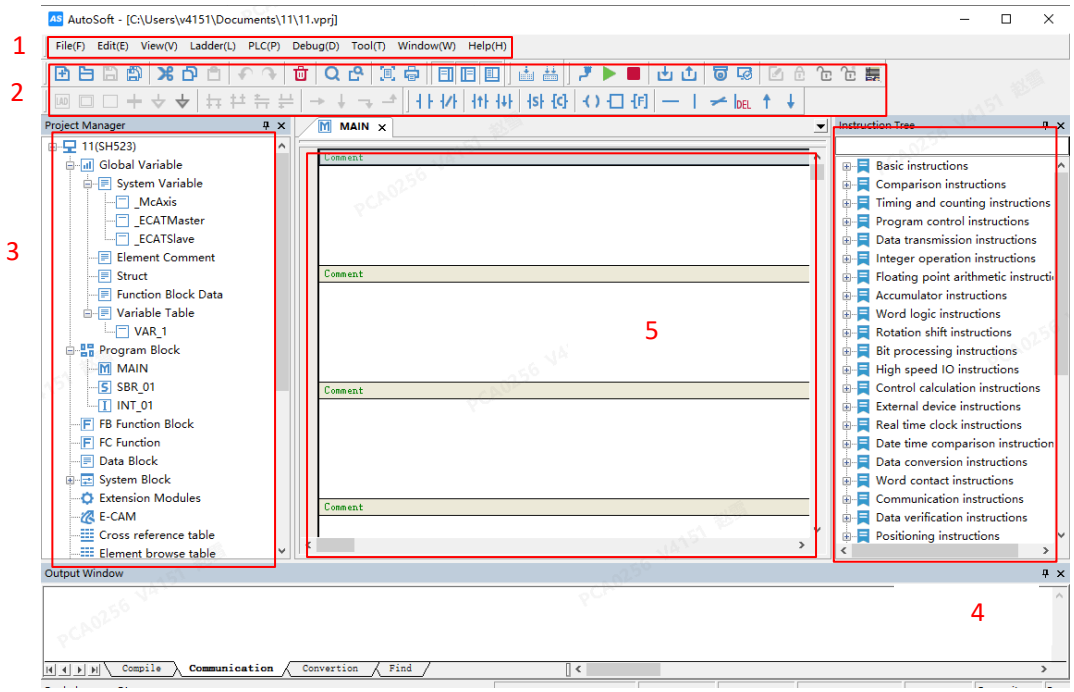
Taking the SH523 as an example, this demonstrates controlling a drive via EtherCAT communication to execute a positioning movement with the following motion profile: acceleration time 1 s, deceleration time 1 s, velocity 300 r/min, and displacement of 10 units. Steps:

New Project

1. Open AutoSoft software and create a new project. Select SH523 as the PLC series and type, as shown below:



2. After creating the project, enter the main interface.



Interface Descriptions

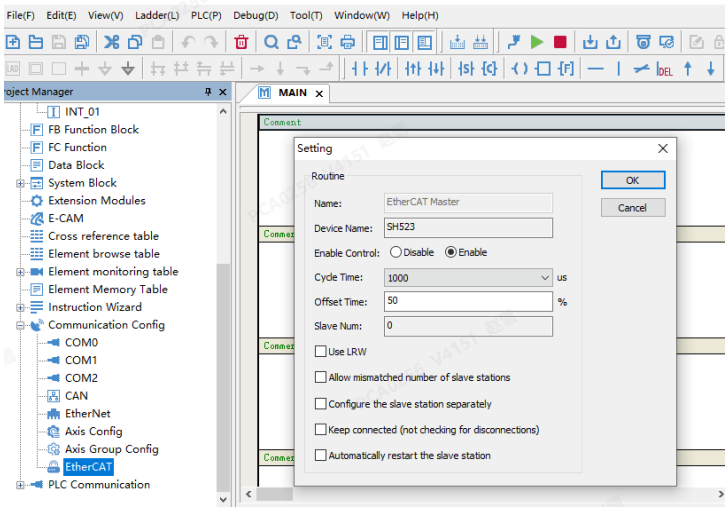
① Menu Bar ② Toolbar ③ Project Manager ④ Instruction Tree ⑤ Programming Area

11.3.1 Bus/Local Pulse Axis Configuration

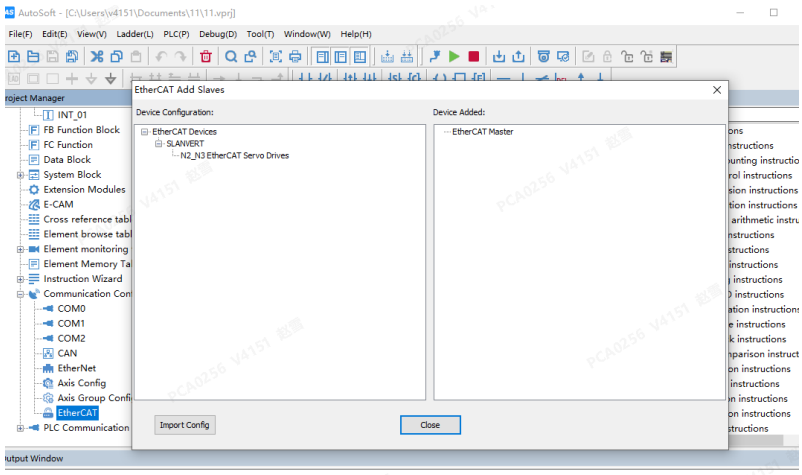
To properly control a bus drive, the programming software requires configuring a servo drive and a bus servo axis, then establishing their connection. Two configuration methods are available: Auto-Scan and Manual Add. Auto-Scan is only for adding bus servo axes; and local pulse axes must be added manually.

(I) Auto-Scan

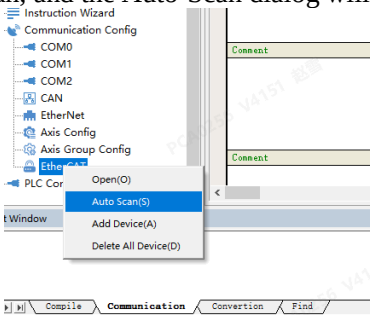
1. Double-click EtherCAT under the Project Manager. In the dialog box, check Enable, click OK, then compile and download the project. (Note: The main program must not be empty).



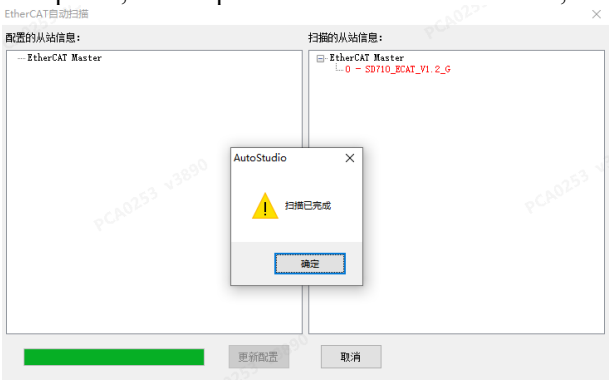
2. Ensure the PLC's EtherCAT port is properly connected to the servo drive. Verify that the drive's XML file exists in the device library. If not, add the corresponding XML file, as shown below.



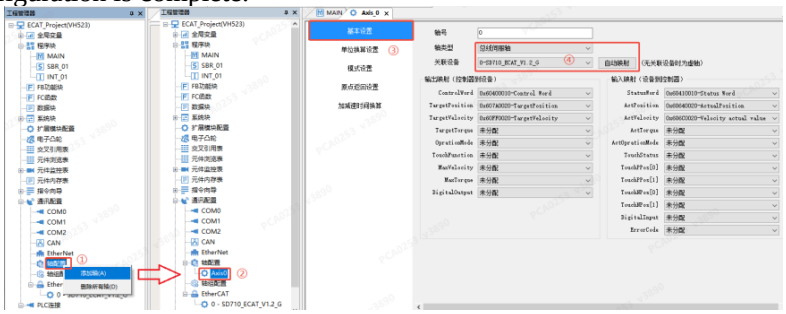
3. Right-click EtherCAT, select Auto-Scan, and the Auto-Scan dialog will open, as shown below.



4. Click “Yes”. After scanning completes, click Update to add the bus servo axis, as shown below.



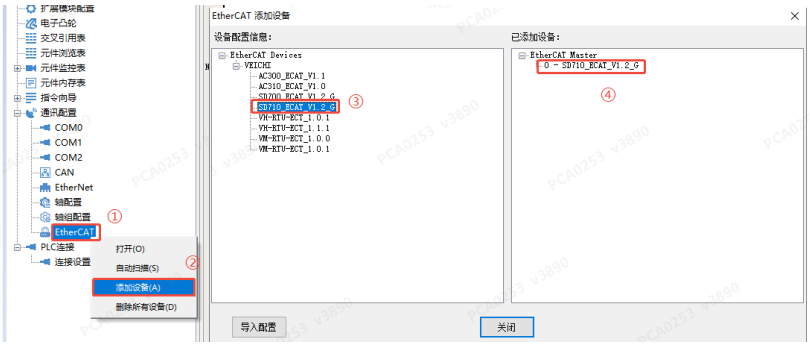
5. After scanning, right-click Axis Configuration to configure it, and link it to the scanned slaves. At this point, the axis association configuration is complete.



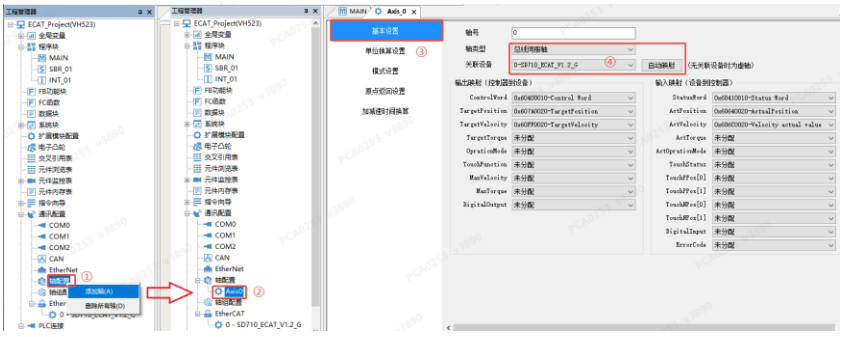
(II) Manual Add

1. Right-click EtherCAT -> Add Device -> Import to manually import the drive’s XML file, as shown below.

2. Right-click EtherCAT -> Add Device, then double-click the target slave device to add, as shown below.



3. After manual add, right-click Axis Configuration to configure it, and link it to the scanned slaves, as shown below.



Note:

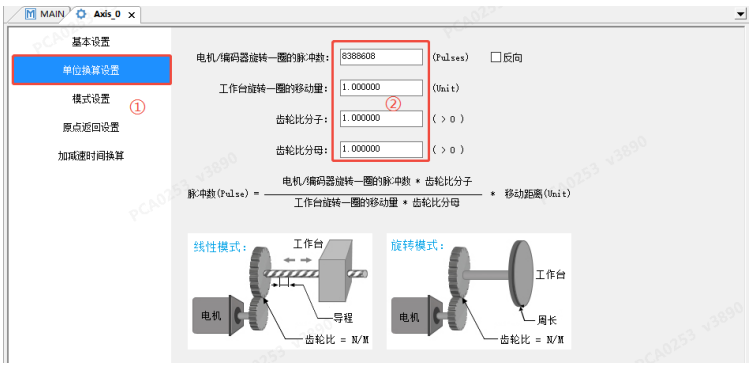
For manually added slaves, ensure Enable under EtherCAT is checked to activate functionality.

11.3.2 Axis Parameter Settings

Bus Servo Axis

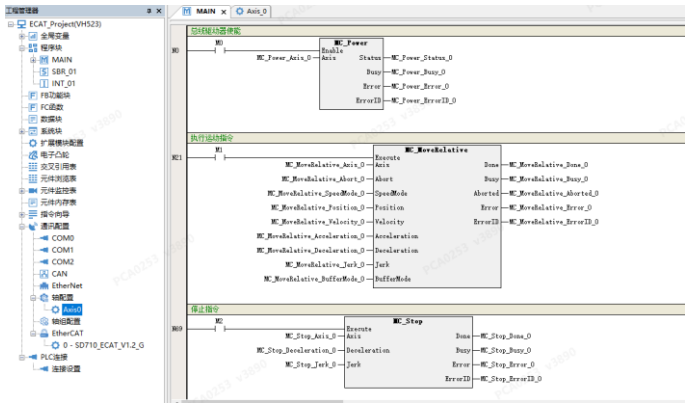
Configure axis parameters based on the actual device. For this example, set the following parameters:

- Number of pulses for one resolution of motor/encoder: 8388608
- Gear ratio: 1:1
- Axis mode: Linear
- Movement of worktable in one revolution: 1 Unit



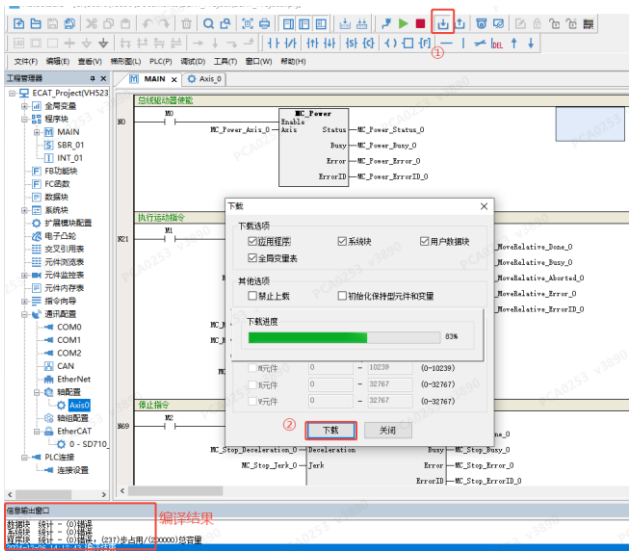
11.3.3 Programming

1. Directly call MC_Power, MC_MoveRelative, and MC_Stop instructions in the program without instantiation, as shown below:



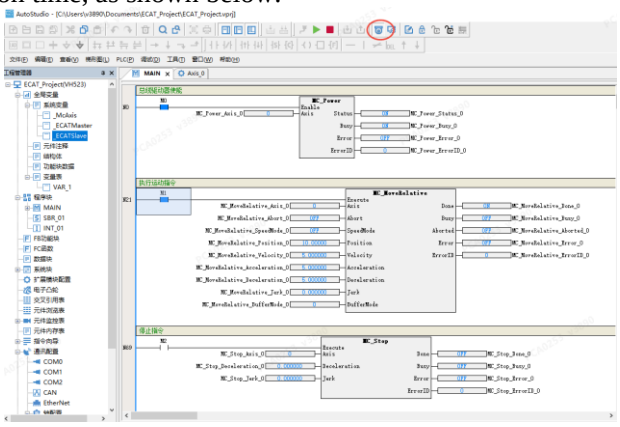
11.3.4 Compilation/Download

1. After completing the project, click ① to download it. The system will first compile the project. Upon successful compilation, a download dialog box will appear. Click Download to proceed. A progress bar will display, and after completion, a prompt will ask whether to set the PLC to Run mode. Select Yes, as shown below:



11.3.5 Online Monitoring

1. Click the Monitor button. Assign parameters to the MC_MoveRelative instruction and set M0 and M1 to ON sequentially. The drive will then execute a 10-uint displacement at 300 r/min with 1s acceleration/deceleration time, as shown below.



11.3.6 Fault Types

Axis faults are categorized into instruction faults, axis faults, and drive faults.

- Instruction faults: Errors generated by MC motion control instructions, such as invalid parameters and PLCOpen state machine changes during operation. Check the fault code via the ErrorID of the faulty instruction (refer to

the Appendix).

- Axis faults: Errors reported by the axis, e.g., excessive following error. Axis fault codes can be viewed through: Current faults monitoring; Historical faults records; or System variable iAlmStatus.
- Drive faults: Errors from EtherCAT bus drives or local pulse output axes. To retrieve faults from the EtherCAT bus drive, 0x603F must be configured in the PDO mapping and linked to the axis. Drive faults can be viewed through: Servo Error section in the drive's backend interface; or ServoErrorID in the MC_ReadAxisError instruction.

12 High-Speed Counter

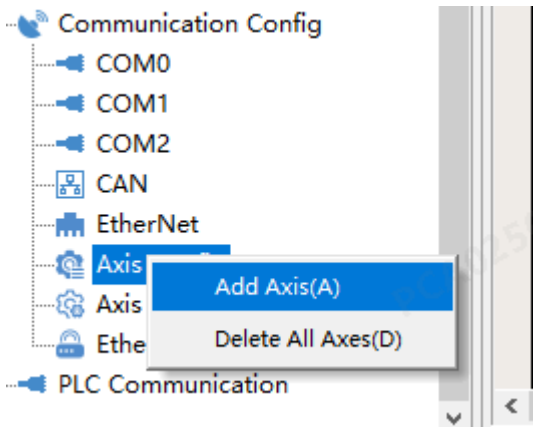
12.1 Overview

Counters are managed as encoder axes in AutoSoft software and engineering applications. When associated with an axis, they are collectively referred to as local encoder axes. The system supports 4-axis 32-bit high-speed counters capable of AB-phase 1/2/4× frequency multiplication, CW/CCW, pulse+direction, and single-phase counting. It also enables counter preset and high-speed counter position latch functions.

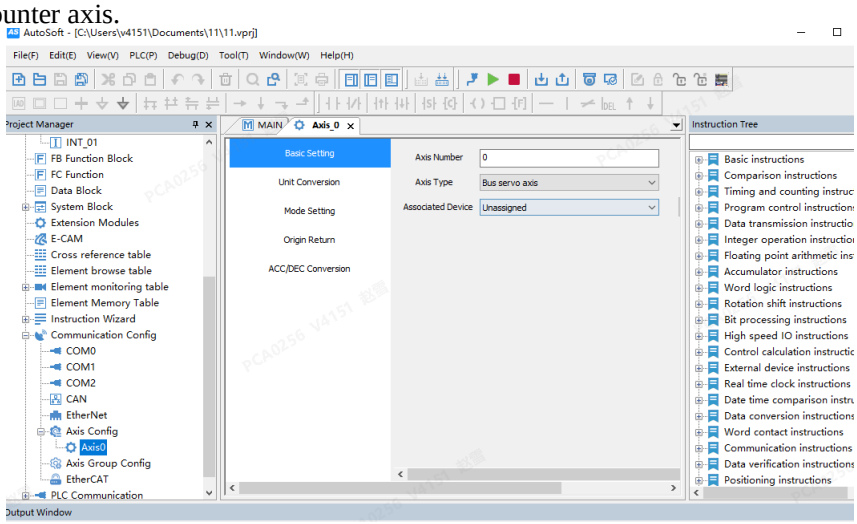
12.2 Creating a Counter Axis

The counter must first be associated with an axis before using.

1. In the Project Manager area, right-click Axis Configuration, and select Add Axis to create a motion control axis.



2. Double-click the newly added axis (e.g., Axis0) to open its configuration page. Under the Basic Settings interface, select “Local encoder axis” as the axis type and “High-speed counter” as the associated device to associate the axis with the counter. The axis number serves as the identifier for programmatic control of the corresponding counter axis.



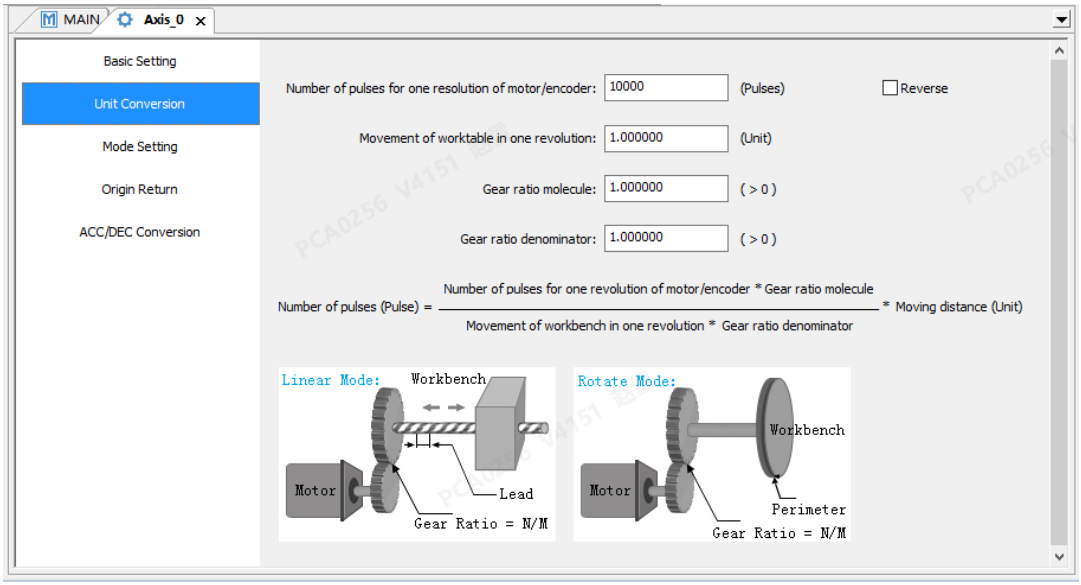
12.3 Unit Conversion for Counter Axis

The high-speed counter uses pulse units to decode encoder signals, while counter instructions employ common measurement units (e.g., millimeters, degrees, or inches), referred to as user units. Unit conversion translates pulse counts into user units (Uint). User units can be defined as application-specific units (e.g., millimeters, revolutions) based on actual requirements.

The parameters required for unit conversion are listed below.

Item	Function
Pulses per motor/encoder revolution	Sets the pulse count per motor/encoder revolution based on encoder resolution.
Workpiece movement per revolution	Defines the workpiece displacement per worktable revolution with transmission.
Gear ratio numerator	Sets the gear ratio on the workpiece side.
Gear ratio denominator	Sets the gear ratio on the motor side.

For example, when a servo motor drives a worktable via a gear reducer and lead screw, the PLC controller monitors the encoder pulses (in pulse units) to determine the worktable position. The counter axis, representing the worktable position, uses millimeters as its unit. Thus, user units are uniformly applied in the program. The conversion relationship between user units (e.g., degrees) and pulses is illustrated in the figure below.



In unit conversion settings, parameters must be configured according to the actual device.

$$\text{Pulse} = \frac{\text{Pulses per motor/encoder revolution[DINT]*Gear ratio numerator[DINT]}}{\text{Workpiece movement per revolution[REAL]*Gear ratio denominator[DINT]}} * \text{Displacement[Uint]}$$

For example, if one encoder revolution corresponds to one revolution of the work axis and the user unit (Uint) is defined as "revolution," set the travel distance per motor/encoder revolution to 1.

Example for a 2000-line encoder:

Pulses per motor/encoder revolution = 2000

Travel distance per motor/encoder revolution = 1

When executing the HC_Counter high-speed counter instruction, the counter value will be 1 after the encoder completes one full revolution.

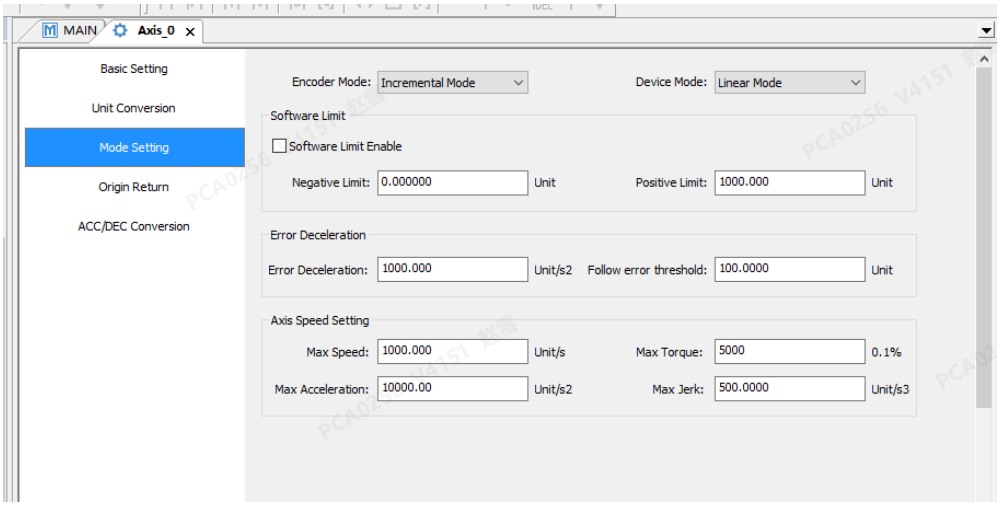
12.4 Operating Mode Configuration

12.4.1 Linear Mode

In linear mode, the counter axis position varies between the negative and positive limit values. If the counter axis reaches a limit and continues to receive pulses in the same direction, the HC_Counter instruction reports an

overflow, but the counter continues counting in the same direction. When reverse pulses are input after an overflow, the counter decrements and the overflow error is cleared.

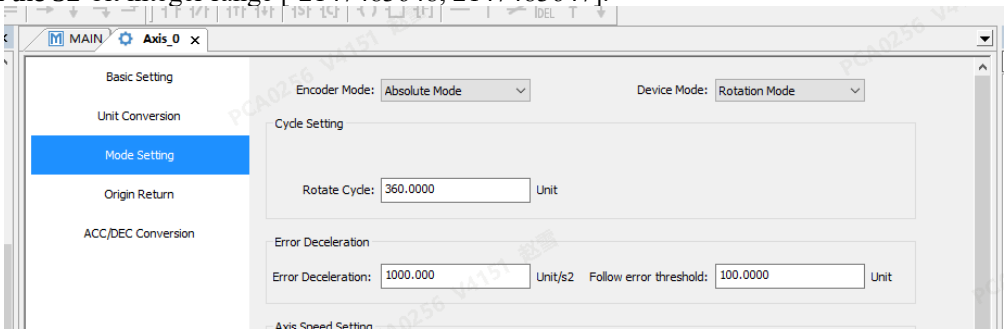
Since the high-speed counter is a 32-bit device, the rotational period converted to pulse units must fall within the 32-bit integer range [-2147483648, 2147483647].

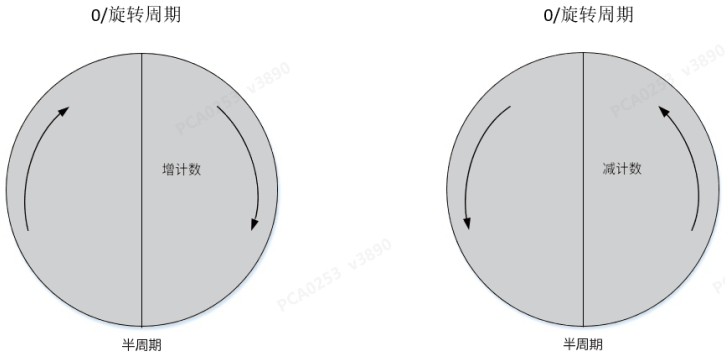


12.4.2 Rotation Mode

In rotation mode, the counter axis position cycles within the rotational period. During incremental counting, the position resets to 0 upon reaching the maximum rotational period value. During decremental counting, the position jumps to the maximum value when decremented from 0.

In this mode, the rotational period of the counter axis can be configured in the interface, with the period unit defined in user units. Since the high-speed counter is a 32-bit device, the rotational period converted to pulse units must fall within the 32-bit integer range [-2147483648, 2147483647].

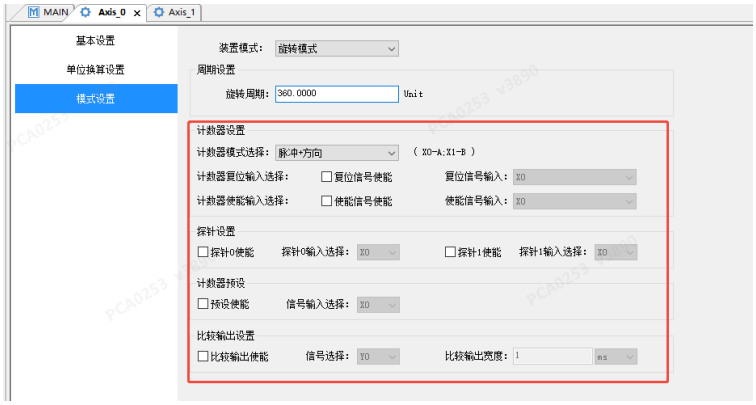




12.5 Counter Parameter Configuration

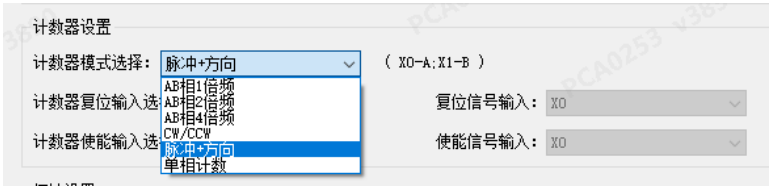
12.5.1 Overview

Parameter configuration primarily includes counter mode, reset, probe, preset, and comparison output functions.



12.5.2 Counter Mode

Local encoder axes support multiple signal counter modes: A/B phase (×1/×2/×4 multiplication), CW/CCW, pulse+direction, and single-phase counting. Signal sources can be selected based on the counter mode.



The input ports supported for each counter mode are listed below. Different local encoder axes may share the same signal source input ports.

Counter Mode	X0	X1	X2	X3	X4	X5	X6	X7
A/B Phase	A Phase	B Phase	A Phase	B Phase	A Phase	B Phase	A Phase	B Phase
CW/CCW	CW	CCW	CW	CCW	CW	CCW	CW	CCW
Pulse+Direction	Pulse	Direction	Pulse	Direction	Pulse	Direction	Pulse	Direction
Single Phase	Pulse	Pulse	Pulse	Pulse	Pulse	Pulse	Pulse	Pulse

Note:

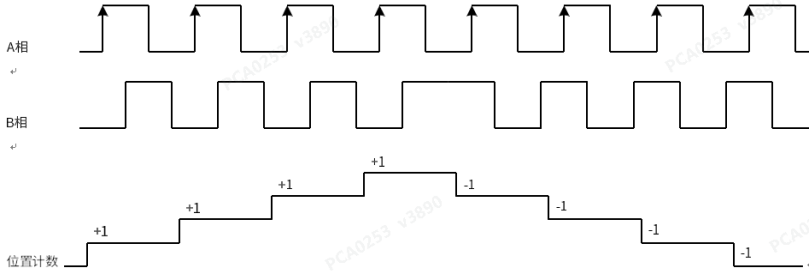
- For modes requiring two input signals: X0/X1, X2/X3, X4/X5, and X6/X7 are grouped as four pairs.
- All four counters can freely select counter modes and signal sources, allowing mode or source reuse across different counters.

A/B Phase Mode

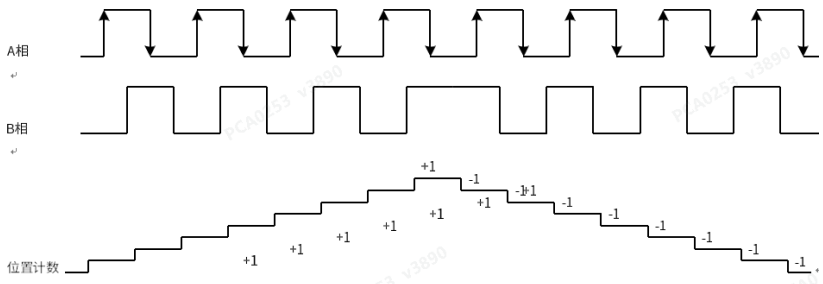
In A/B Phase Mode, the encoder generates two quadrature phase pulse signals (Phase A and Phase B) with a 90° phase shift. The counter increments when Phase A leads Phase B and decrements when Phase B leads Phase A.

A/B phase pulses can operate in ×1, ×2, or ×4 multiplication modes:

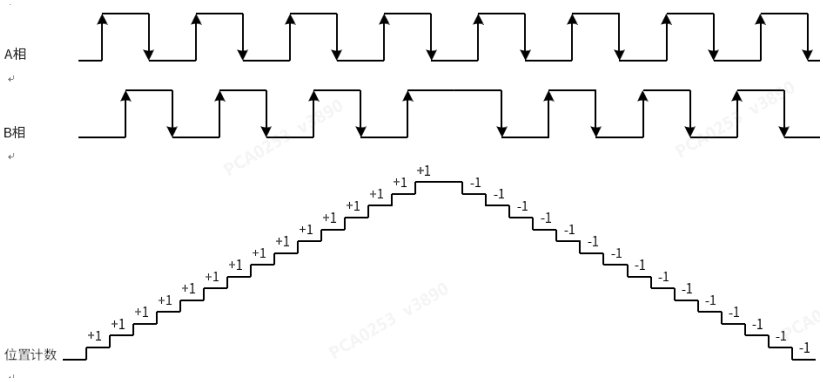
- ×1 multiplication: Counts only on the rising edge of Phase A pulses, as shown in the figure below.



- ×2 multiplication: Counts on both rising and trailing edges of Phase A pulses, as shown in the figure below.

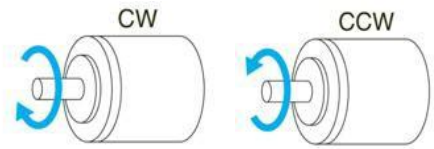


- ×4 multiplication: Counts on rising and trailing edges of both Phase A and Phase B pulses, as shown in the figure below.

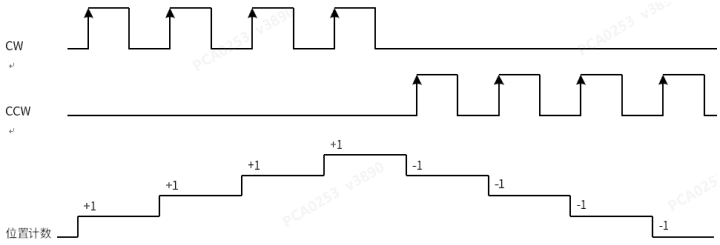


CW/CCW Mode

In CW (Clock Wise) and CCW (Counter Clock Wise) modes, the encoder outputs CW pulses during forward rotation and CCW pulses during reverse rotation.

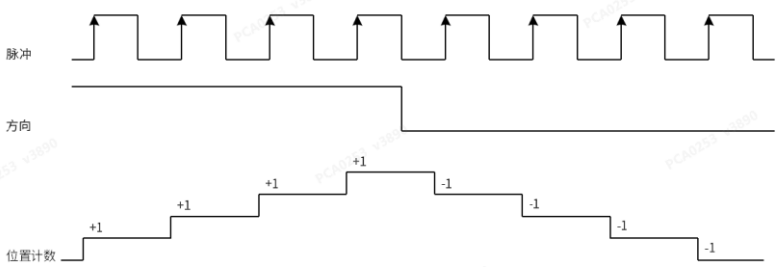


When the local encoder axis operates in this counter mode, the high-speed counter increments on CW signals and decrements on CCW signals, as shown in the figure below.



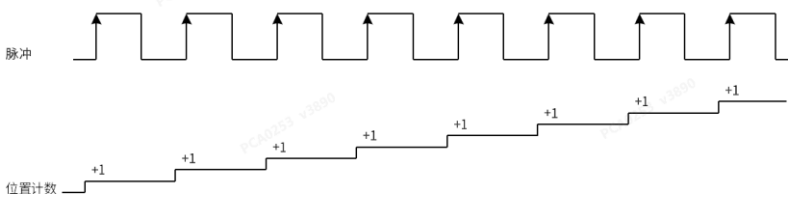
Pulse+Direction Mode

In this mode, the high-speed counter increments on pulse signals when the direction signal is ON and decrements when the direction signal is OFF, as shown in the figure below.



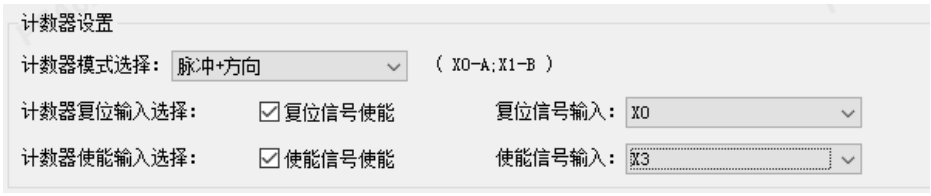
Single Phase

In this mode, the high-speed counter increments on pulse signals. The position count increases by 1 at the rising edge of each input pulse.



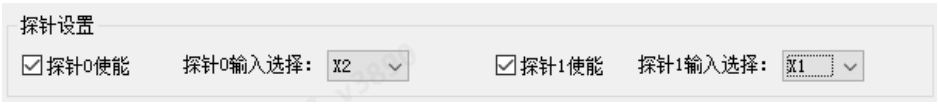
12.5.3 Reset Configuration

(1) Check the "Counter reset signal enable" to use an external input (X0~X7 terminals) for encoder axis position reset. The trigger condition (rising/trailing edge) is configurable.
(2) When enabled, the encoder axis position is cleared to zero via the external input signal.



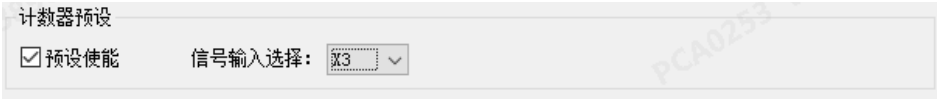
12.5.4 Probe Terminal

(1) Each counter supports two external inputs to latch the current counter value for probe functionality. Check the "Probe0 Enable" to trigger position latching via an external input (X0~X7 terminals).
(2) Once enabled, use the HC_TouchProbe function block to read the latched position of the counter axis.



12.5.5 Preset Terminal

(1) Check the "Preset Enable" to use an external input (X0~X7 terminals) for counter value preset. The trigger condition (rising/trailing edge) is configurable.
(2) When enabled, use the HC_Preset function block to preset the encoder axis position via external input.



12.5.6 Compare Output Terminal

Enabling Compare Output allows hardware-based output activation upon comparison match without software intervention, ensuring high real-time performance.

- When enabled, the HC_Compare, HC_ArrayCompare, or HC_StepCompare function blocks trigger hardware-controlled output (ON state) upon position match. Output terminals (Y0~Y7) and pulse width (time or user units) are configurable.
- Each local encoder axis supports one comparison output channel, configurable with input terminals and pulse width.
- When the unit is set to ms, the configurable time range is 1ms~6553.5ms. When the unit is set to pulses, ensure the value converted to pulse units falls within 1~65535.

Compare Output is hardware-controlled and cannot be monitored via Y soft elements. The Y soft elements and comparison output share an OR-logic control over the output port. If a Y soft element keeps the output ON, the actual port remains ON regardless of comparison results.

12.6 Counter Axis Instruction Applications

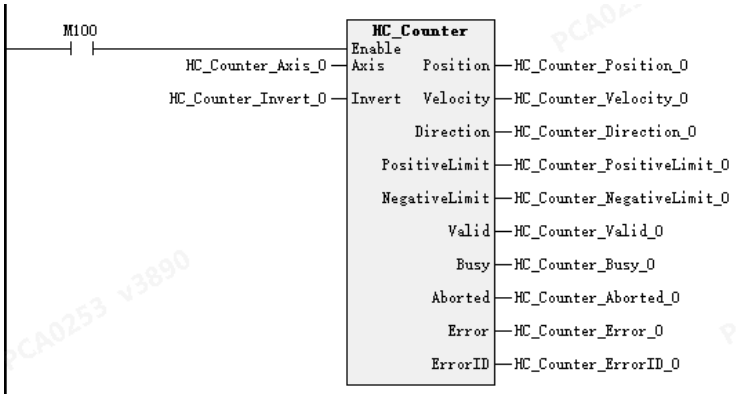
12.6.1 Overview

In AutoSoft software, configuring a counter axis combined with function block instructions enables axis position counting/speed measurement, position presetting, position latching, and comparison functions.

12.6.2 Position Counting/Speed Measurement Instructions

The HC_Counter instruction enables position counting and speed measurement for a counter axis. The counter axis position value varies within the range defined by the counter axis mode, with its unit specified as user units.

The counter axis velocity reflects real-time velocity in user units/s. The minimum measurable velocity corresponds to one pulse per second (e.g., if one pulse equals 0.01 user units, the minimum measurable speed is 0.01 user units/s).



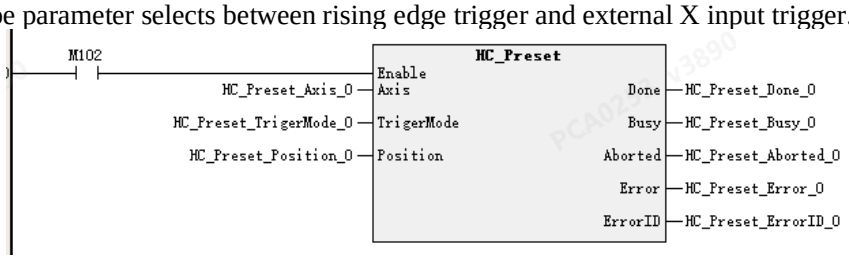
The counting direction can be adjusted using the Invert parameter. After modifying the Invert setting, re-enable the function block instruction to apply the change. The relationship between Invert settings and counting directions is shown in the following table:

Invert	A/B Phase	Pulse+Direction	CW/CCW	Single Phase
0	A leads B: Increment counting; B leads A: Decrement counting	Direction signal low: Decrement counting; Direction signal high: Increment counting	A: Increment counting; B: Decrement counting	Increment counting
1	A leads B: Decrement counting; B leads A: Increment counting	Direction signal low: Increment counting; Direction signal high: Decrement counting	A: Decrement counting; B: Increment counting	Decrement counting

12.6.3Position Preset Instruction

The HC_Preset instruction assigns a position value to the counter axis based on preset conditions.

The TriggerType parameter selects between rising edge trigger and external X input trigger.



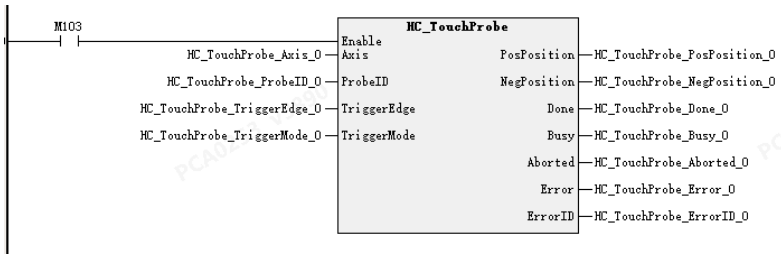
TriggerType	Definition
0	Element/variable rising edge trigger
1	External X rising edge trigger

When external X input trigger is selected, enable the preset function in counter parameter settings, select an input terminal (X0~X7), and set the trigger condition (rising edge).

12.6.4Probe Instruction

The HC_TouchProbe function block instruction latches the counter axis position value when external input trigger conditions are met.

1. Each counter axis supports 2×probe. To use this feature, enable the corresponding probe function in counter parameter settings, select an input terminal (X0~X7), and configure the trigger condition.



2. The ProbeID parameter specifies the probe number:

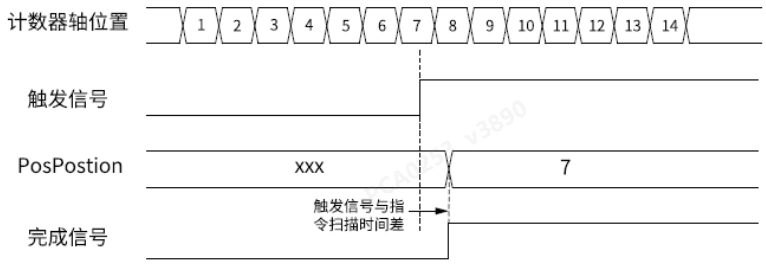
ProbeID	Definition
0	Probe 1
1	Probe 2

3. The TriggerEdge parameter defines the trigger edge type: Rising edge triggers latch the position in the PosPosition output parameter; Trailing edge triggers latch the position in the NegPosition output parameter.

TriggerEdge	Definition
0	External X rising edge trigger
1	External X trailing edge trigger
2	Both rising and trailing edges

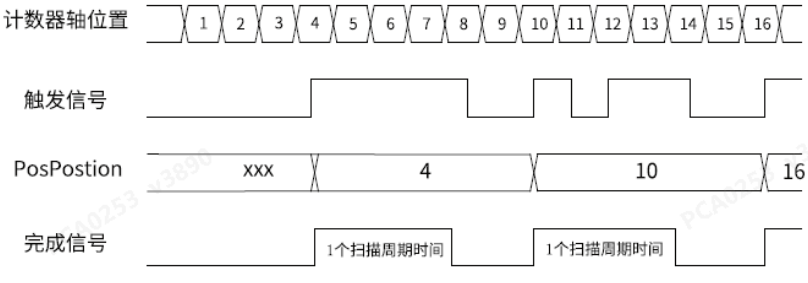
The TriggerMode parameter defines single trigger or continuous trigger modes:

- Single trigger mode: When the function block instruction's power flow is active and the external trigger condition is met, the counter axis position is latched once, and a Done signal is output. The probe position latches the axis position in real-time based on the trigger edge, unaffected by program execution. The latched position is updated to the instruction's output parameters when the program scan cycle processes the latch instruction.



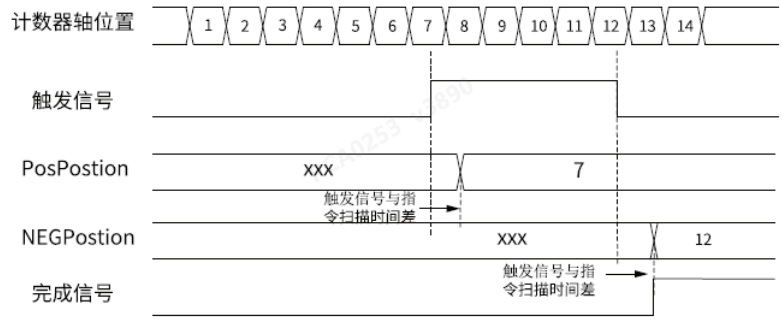
上升沿单次触发模式

- Continuous trigger mode: When the power flow is active and the trigger condition is met, the axis position is latched, and a Done signal (valid for one scan cycle) is output. After the Done signal turns OFF, subsequent trigger conditions will continue to latch positions and generate Done signals. During the Done signal's active scan cycle, new triggers are ignored.



上升沿连续触发模式

- Dual-edge trigger mode: After both rising and trailing edges trigger latching, a Done signal is output. In single trigger mode, the Done signal persists until the instruction completes; in continuous trigger mode, the Done signal lasts one scan cycle, during which new triggers are ignored.



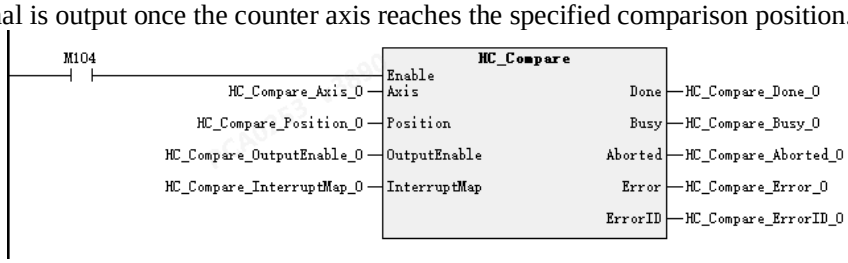
上升下降沿单次触发模式

12.7 Compare Instructions

The HC_Compare, HC_StepCompare, and HC_ArrayCompare instructions enable single-position comparison, equidistant continuous comparison, and multi-position continuous comparison for counter axes.

12.7.1 HC_Compare Instruction

This instruction performs a single-position comparison for the counter axis. When the instruction's power flow is active, a Done signal is output once the counter axis reaches the specified comparison position.



12.7.2 HC_StepCompare Instruction

This instruction enables equidistant continuous position comparison for the counter axis. When the power flow is active, the counter axis position is compared starting from StartPosition. After a match, the comparison position increments/decrements by the Step interval and continues. A Done signal is output only after the last comparison position is reached, not after each individual match.

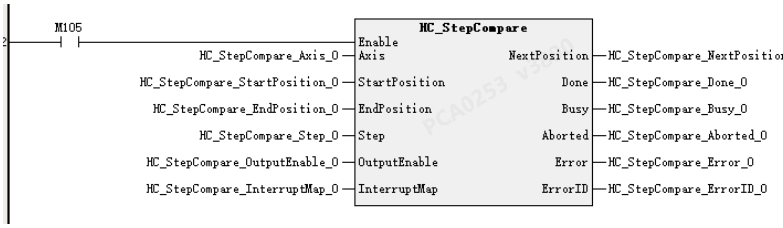
When StartPosition < EndPosition:

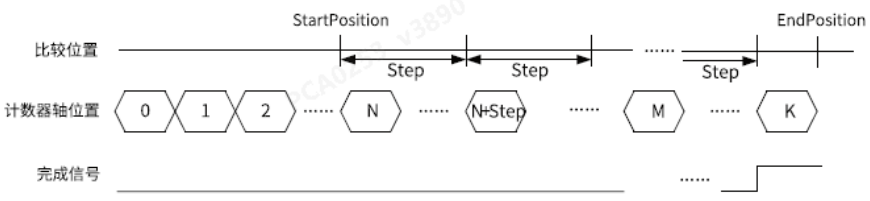
The comparison position increases by Step after each match. The last comparison occurs when the incremented position exceeds EndPosition.

When StartPosition > EndPosition:

The comparison position decreases by Step after each match. The last comparison occurs when the decremented position falls below EndPosition.

● The output parameter NextIndex indicates the next comparison point index (i.e., the count of completed matches).



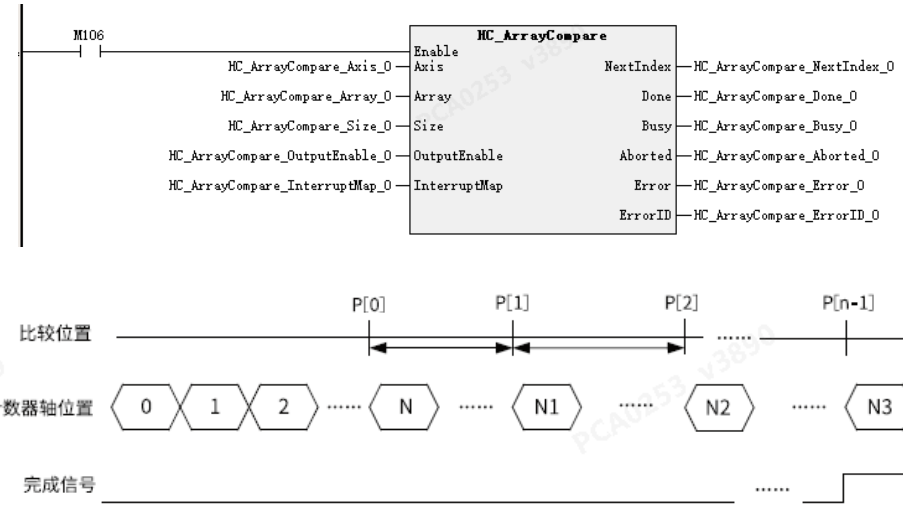


12.7.3 HC_ArrayCompare Instruction

This instruction enables multi-position continuous comparison between the counter axis and an array. When the power flow is active, the counter axis position is compared sequentially starting from the first array element. After a match, it proceeds to the next array element. A Done signal is output after the last array position is compared.

● ArrayLength sets the array length. Once all positions defined by the array length are compared, the Done signal remains active, completing the multi-position comparison.

● The output parameter NextIndex indicates the next comparison point index (i.e., the count of completed matches).



12.8 High-Speed Hardware Compare Output

The counter axis enables position comparison hardware output. When the counter axis matches the comparison position, the hardware circuitry directly turns the output ON with a delay of less than 1ms.

● Configuring comparison output for the counter axis:

In the counter axis parameter settings, check the “Compare output enable”, select an output terminal (Y0~Y3), and set the pulse width (time or user units) for the ON state.

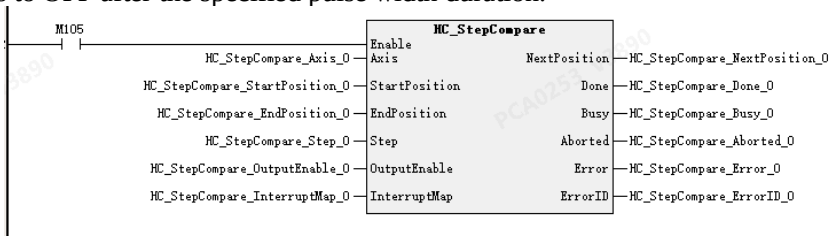
Note:

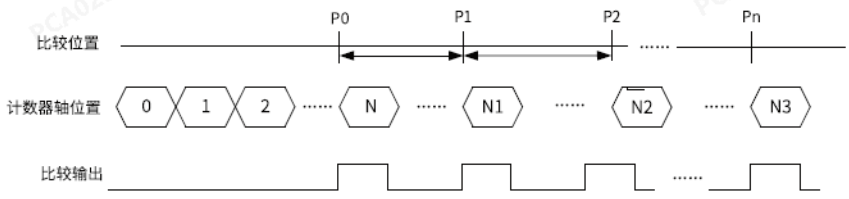
➢ When pulse width is set in time units, the precision is 1ms, with a maximum width of 6500ms; When set in pulse units, the maximum width corresponds to 65535 pulses.

Enabling the OutputEnable Parameter in Compare Instructions

For HC_Compare, HC_StepCompare, and HC_ArrayCompare instructions, setting OutputEnable to 1 links the comparison match to hardware output.

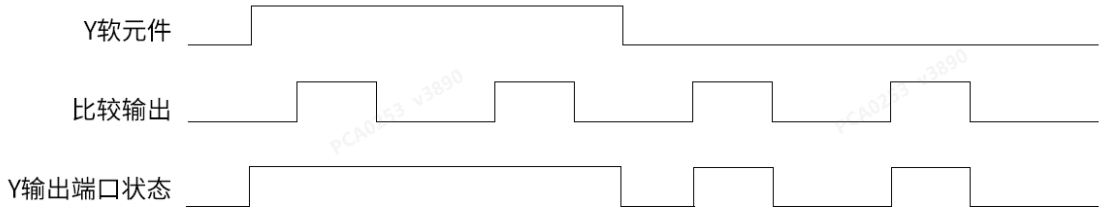
When a comparison match occurs, the hardware circuitry directly turns the configured output terminal (Y0~Y3) ON. The output reverts to OFF after the specified pulse width duration.





Note:

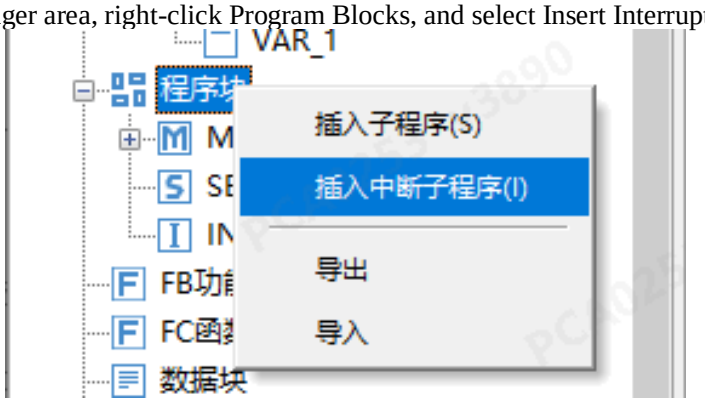
- High-speed comparison hardware output is controlled directly via hardware circuits. The output state cannot be monitored through Y soft elements in the program. The Y soft elements and comparison output share an OR-logic control over the output port. If a Y soft element keeps the output ON, the actual port remains ON regardless of comparison results.



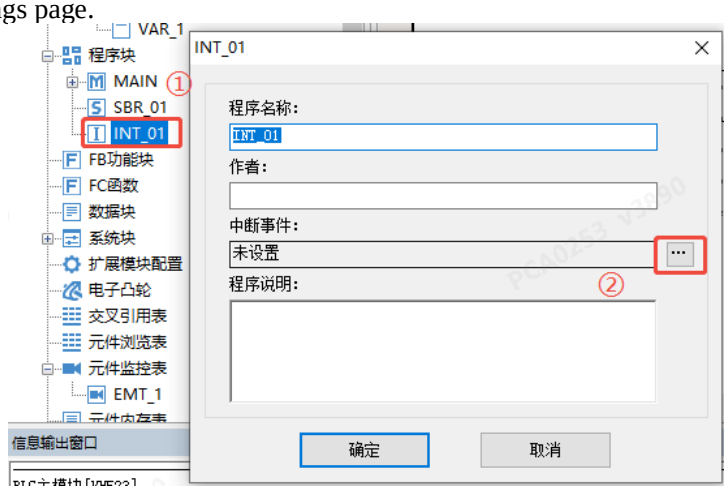
12.9 Compare Interrupt

When the counter axis matches a comparison position, it can trigger a comparison interrupt to execute an interrupt subprogram, with configuration steps as below:

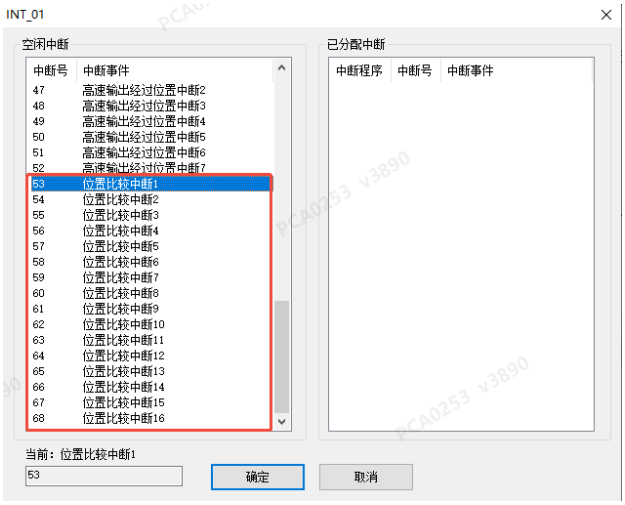
1. In the Project Manager area, right-click Program Blocks, and select Insert Interrupt Subprogram.



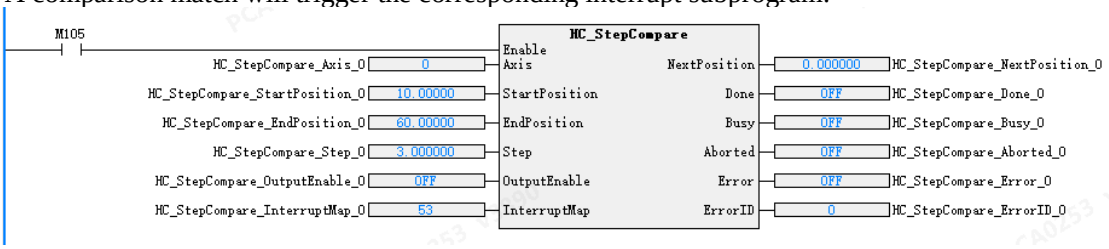
2. Right-click the inserted interrupt subprogram (e.g., INT_01) and select Property to open the interrupt subprogram settings page.



3. Click "...", select Position comparison interrupt 1 under Interrupt Event, then click OK. Write the interrupt logic in INT_01.



4. In the main or subprogram, call HC_Compare, HC_StepCompare, or HC_ArrayCompare instructions, and link the InterruptMap parameter to the comparison interrupt number. Enable EI and SM71 in the program. A comparison match will trigger the corresponding interrupt subprogram.



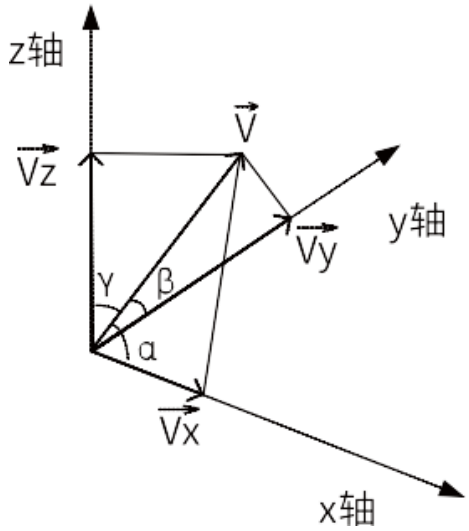
13 Interpolation

13.1 Introduction to Interpolation

13.1.1 Overview

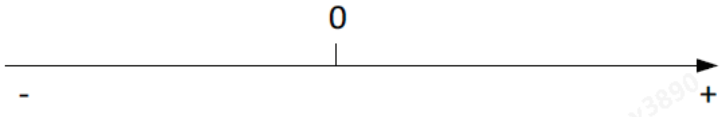
Interpolation operates in a spatial Cartesian coordinate system, supporting linear interpolation, circular interpolation, helical interpolation, and 3D circular interpolation. The interpolation function is implemented through axis groups.

- Each axis group can control up to 8 motion axes, including three coordinate axes (X, Y, Z) and auxiliary axes (A, B, C, H, W).
- The SH series supports up to 8 axis groups. Each group can be configured as 2-axis (XY), 3-axis (XYZ), or 4-axis (XYZ + one auxiliary axis).
- Linear and circular interpolation support buffer mode. Each axis group can buffer up to 8 curves, with individually configurable transition modes between curves (for buffer and transition details, refer to Interrupt + No Transition).
- When an axis group executes interpolation, single-axis instructions (e.g., MC_MoveAbsolute, MC_Stop) are prohibited if the PLCopen state machine of any axis is in the SynchronizedMotion state. Single-axis motion instructions (e.g., MC_MoveAbsolute, MC_MoveRelative, MC_Jog, MC_Home) can be executed only when the axis is in the StandStill state.



Spatial Cartesian Coordinate System

In the diagram, Vx, Vy, Vz represent the component velocities of the three coordinate axes, corresponding to the actual speeds of the servo axes. V denotes the real-time speed of the interpolated curve. α, β, and γ represent the angles between the velocity vector and the coordinate axes.



Linear Coordinate System for Auxiliary Axes

● During linear interpolation, the motion control axes (X, Y, Z) move along their respective coordinate axes, while auxiliary axes travel linearly from their starting to ending positions.

● During circular interpolation, a plane (XY, YZ, or XZ) is selected for the circular motion. If additional axes are configured in the axis group, these axes move linearly from their starting to ending positions.

13.1.2 Axis Group Instruction List

The axis group control instructions are listed below. For details, refer to the SH Series PLC Instruction Manual.

Instruction	Name
MC_MoveLinear	Linear interpolation

MC_MoveCircular	Circular interpolation
MC_GroupStop	Stop axis group motion
MC_GroupPause	Pause axis group motion

13.1.3 Configuration Interface

In Project Manager area, right-click Axis Group Configuration>Add Group, and double-click to open its configuration interface.

The axis group configuration interface includes two sections: Basic Settings and Parameter Settings.

Basic Settings

● Group ID: Assigns a unique identifier to the axis group.

● Coordinate Axis: Selects coordinate axes. X and Y axes are mandatory; Z and auxiliary axes are optional. Axes can be shared across different axis groups.

Parameter Settings

● Maximum Speed: In linear interpolation mode, specifies the maximum interpolation speed for spatial linear motion; in circular interpolation mode, defines the maximum linear velocity for circular motion.

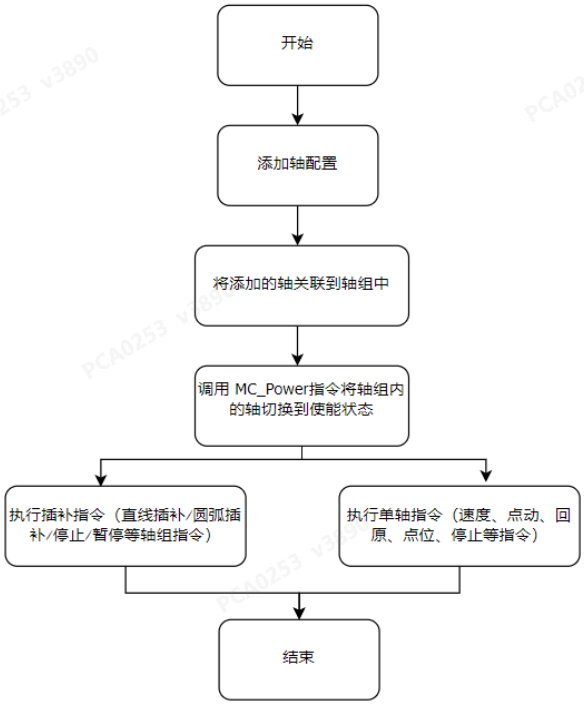
● Maximum Acceleration: In linear interpolation mode, sets the maximum interpolation acceleration for spatial linear motion; in circular interpolation mode, defines the maximum acceleration for circular motion.

● Stop Method: Determines the stopping method when a fault occurs in the axis group.

13.2 Axis Group Interpolation Example

13.2.1 Overview

To execute interpolation instructions correctly, first create an axis group and enable the axes within the group. The basic workflow is shown below:



Axis Group Interpolation Operation Flowchart

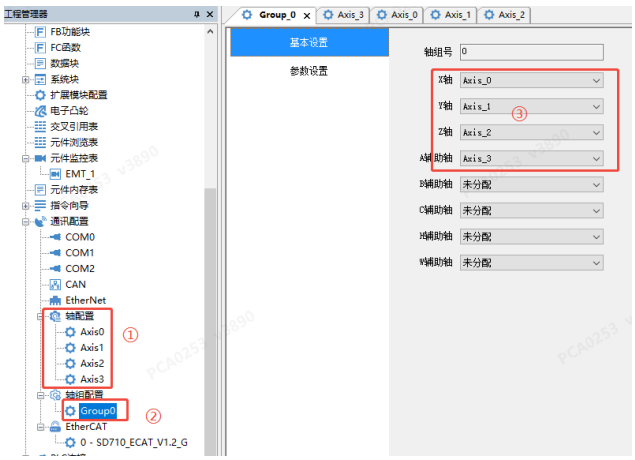
Note

- Even after creating an axis group, single-axis motion and control instructions can still be executed for axes within the group. However, single-axis motion instructions and axis group interpolation instructions are mutually exclusive; they cannot be executed simultaneously or interrupt each other.

This section provides an example to demonstrate the basic workflow for configuring Axis_0, Axis_1, Axis_2, and Axis_3 into an axis group and executing interpolation actions.

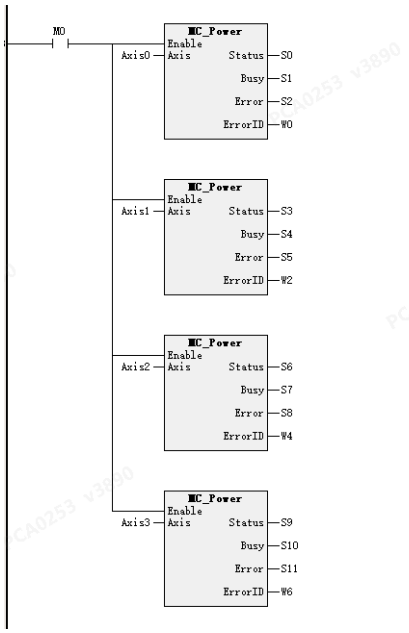
13.2.2 Creating an Axis Group

1. In Project Manager area, right-click Axis Group Configuration>Add Group, and double-click to open its configuration interface. Configure coordinate axes, auxiliary axes, and set relevant parameters.



13.2.3 Enabling the Axis Group

Each individual axis in the group is enabled or disabled via the MC_Power instruction. Axis group instructions can be executed only when all axes in the group are enabled.

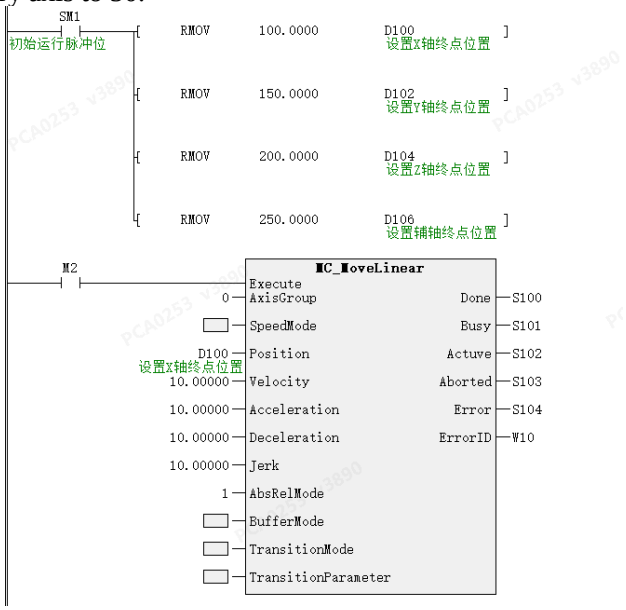


13.2.4 Linear Interpolation

1. The axis group's linear interpolation is executed via the MC_MoveLinear instruction. When all axes in the group are in the StandStill state, triggering Execute starts the linear interpolation, transitioning all axes to the SynchronizedMotion state. Single-axis motion instructions (e.g., MC_MoveAbsolute, MC_Stop) are prohibited during this state.
2. After completing the interpolation, all axes return to the StandStill state, allowing single-axis instructions to resume.

Example

1. This example uses absolute positioning to move the X, Y, and Z axes to position (100, 100, 100), while positioning the auxiliary axis to 50.



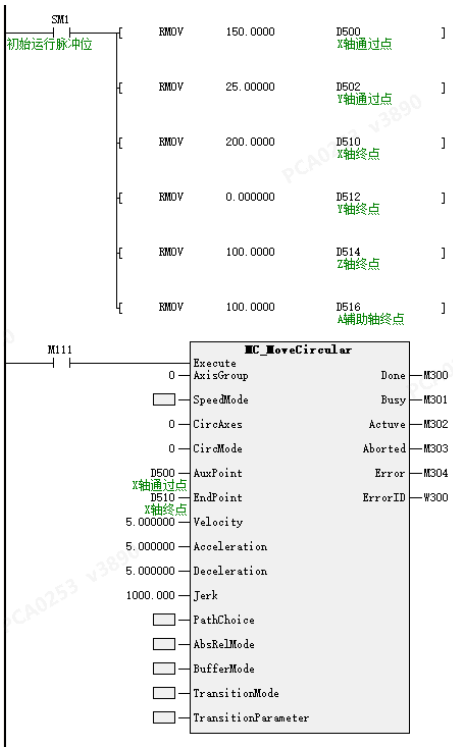
13.2.5 Circular Interpolation

The axis group's circular interpolation is executed via the MC_MoveCircular instruction. The state transitions in the PLCopen state machine follow the same rules as linear interpolation.

Example

This example performs circular interpolation in the XY plane, while the Z axis and auxiliary axis execute synchronized linear motion. The circular interpolation uses via-point mode with absolute positioning, passing through (150, 25) and ending at (200, 0). The Z axis and auxiliary axis are positioned to 100.

For detailed parameters of circular interpolation, please refer to the SH Series PLC Instruction Manual.

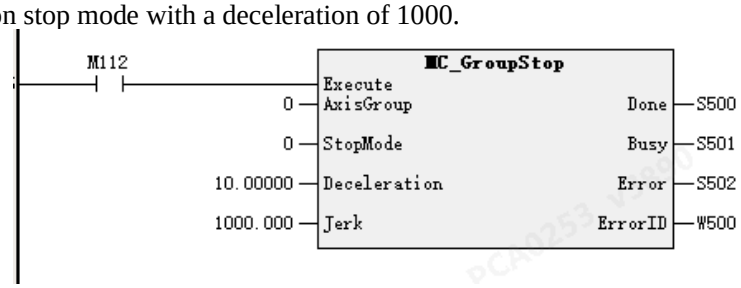


13.2.6 Axis Group Stop

1. The MC_GroupStop command stops the execution of interpolation curves. It interrupts the interpolation on the rising edge of Execute, and the CommandAborted output is activated.
2. Triggering the interpolation command while Execute=TRUE is invalid. Execute must be set to FALSE to restart a new interpolation command.
3. This command can be called only when all axes in the group are in the StandStill or SynchronizedMotion state. During the Execute phase, axes remain in SynchronizedMotion.
4. MC_GroupStop stops only interpolation curves; it does not affect single-axis motion commands (e.g., MC_MoveAbsolute).

Example

In this example, the following command is called during linear or circular interpolation to halt the interpolation curve using deceleration stop mode with a deceleration of 1000.



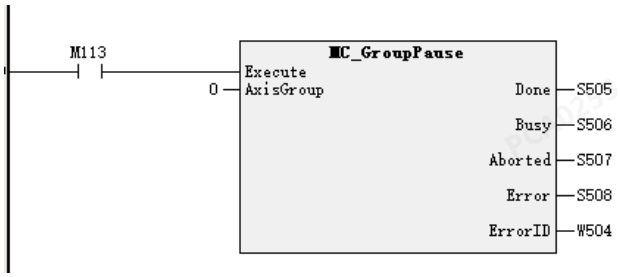
13.2.7 Axis Group Pause

The MC_GroupPause command pauses or resumes interpolation curves. Setting Enable=TRUE pauses the interpolation, while Enable=FALSE resumes execution.

MC_GroupPause pauses only interpolation curves; it does not affect single-axis motion commands (e.g., MC_MoveAbsolute).

Example

In this example, the following command is called during linear or circular interpolation to pause the interpolation curve with a deceleration of 5000.



13.3 Buffer and Transition

13.3.1 Overview

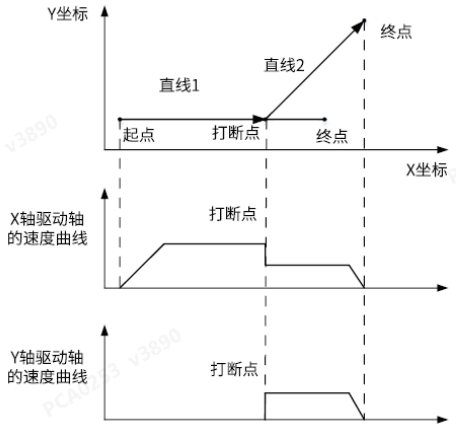
- Buffer mode defines the execution flow when multiple interpolation instructions are activated simultaneously.
- Transition mode specifies the method for switching between curve segments.
- Four buffering and transition combination modes are supported:

No.	Buffer Mode	Description
0	Break + No transition	Immediately switch to the next block without transition curve.
1	Buffer + No transition	Execute the buffered block after deceleration of the first segment completes, no transition curve.
2	Merge with previous velocity + No transition	Move to the end of the first segment at current velocity and execute the second segment at the first segment's rate.
3	Angular transition	With transition curve; deceleration of the first segment overlaps with acceleration of the second segment.

13.3.2 Interrupt + No transition

First, execute the first interpolation instruction. If the second interpolation instruction with "Interrupt + No transition" buffer mode is triggered before the first linear segment completes, the second instruction will immediately interrupt the first one and execute the new interpolation curve.

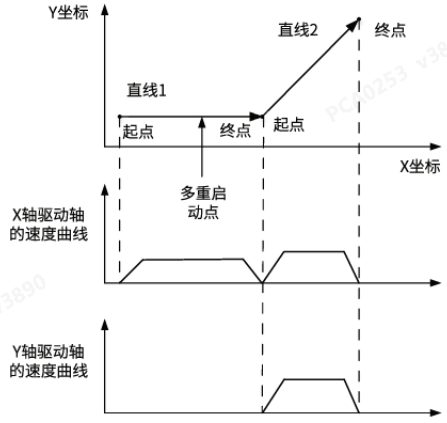
At the interruption point, the new curve maintains the original speed and recalculates component velocities for axes (e.g., X, Y) as shown in the figure below:



13.3.3 Buffer + No transition

First, execute the first interpolation instruction. If the second interpolation instruction with "Buffer + No transition" buffer mode is triggered before the first linear segment completes, the interpolator will continue executing the first instruction.

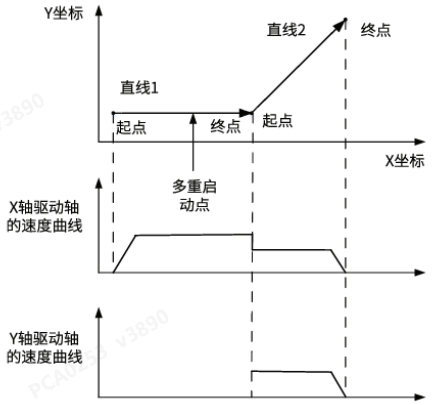
The second instruction starts execution only after the first instruction completes and outputs a valid Done signal, as shown in the figure below:



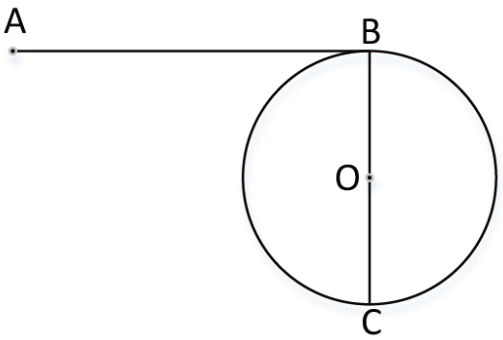
13.3.4 Merge with previous velocity + No transition

First, execute the first interpolation instruction. If the second interpolation instruction is triggered before the first linear segment completes and its buffer mode is set to "Merge with previous velocity + No transition," the interpolator will maintain the target speed of the first instruction until the current linear segment finishes.

The second instruction starts execution only after the first instruction completes and outputs a valid Done signal. At the transition point, the velocity remains unchanged, and the axis velocity components (e.g., X, Y) are reallocated, as shown in the figure below:

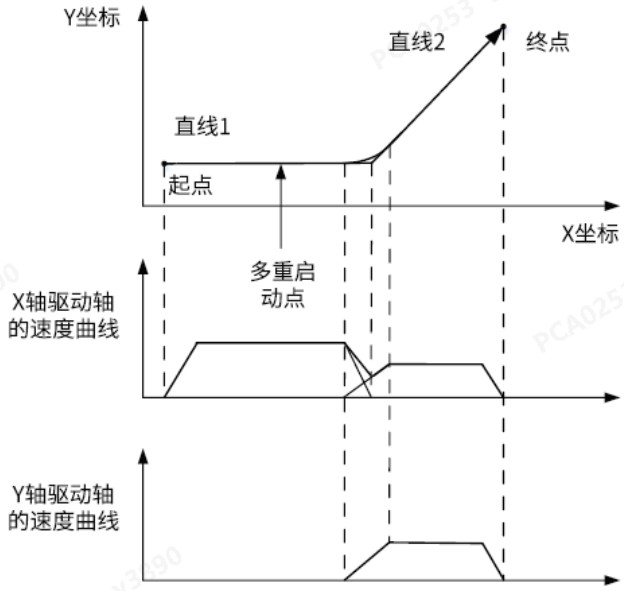


This mode is particularly suitable for transitions between linear and circular paths where the linear segment lies tangent to the circular arc, ensuring continuous interpolation speed.



13.3.5 Angular transition

First, execute the first interpolation instruction. If the second interpolation instruction is triggered before the first linear segment completes and its buffer mode is set to "Additional corner transition," the interpolator initiates the second instruction when the first segment begins deceleration. The final component velocities of each axis are the sum of the velocities from both instructions, as shown in the figure below:


Y坐标
直线2
终点
直线1
起点
X坐标
多重启
动点
X轴驱动轴
的速度曲线
Y轴驱动轴
的速度曲线

14 Electronic Cam

14.1 Introduction to Electronic Cam

An electronic cam essentially describes the motion relationship where a slave axis follows a master axis, defined either by cam table data or electronic gear ratio.

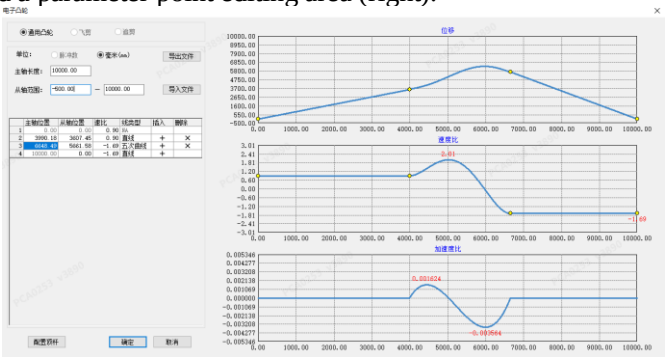
- Cam tables support up to 361 key points. Electronic gear ratios maintain a fixed proportional relationship between master and slave axes.
- Electronic gears require only numerator/denominator ratio settings without cam table configuration. Electronic cams require predefined cam table data.
- Programming software supports 16 configurable cam tables, with 8 simultaneously active. Each cam table allows a maximum of 361 key points.
- Cam tables permit dynamic addition, deletion, or modification of key points during operation. Modified cam tables take effect in the next cam cycle.

14.2 Software Configuration

14.2.1 Overview

1. In the Project Manager area, right-click "E-Cam", and select "Add Configuration" to create a new cam table (CAM0).

2. Double-click "CAM0" to access its configuration interface. The cam table interface includes a graphical editing area (left) and a parameter point editing area (right).



14.2.2 Cam Table Specifications

When creating a cam table, adhere to the following specifications:

Item	Description
Total key points per cam table	361
Total configurable cam tables	16
Number of cam tables executable simultaneously in PLC	16
Cam table switching during operation	Modify the cam table using the MC_CamIn instruction, taking effect in the next cam cycle.
Cam data read/write	Cam table data, such as phase and displacement, can be read via instructions. Key point data can be directly edited and applied using the MC_GenerateCamTable instruction.

14.2.3 Cam Node Configuration

Users can configure cam nodes in the parameter point editing area based on application requirements. Click "+" to add a new cam node data row for editing. Select a node row and click "×" to delete it.

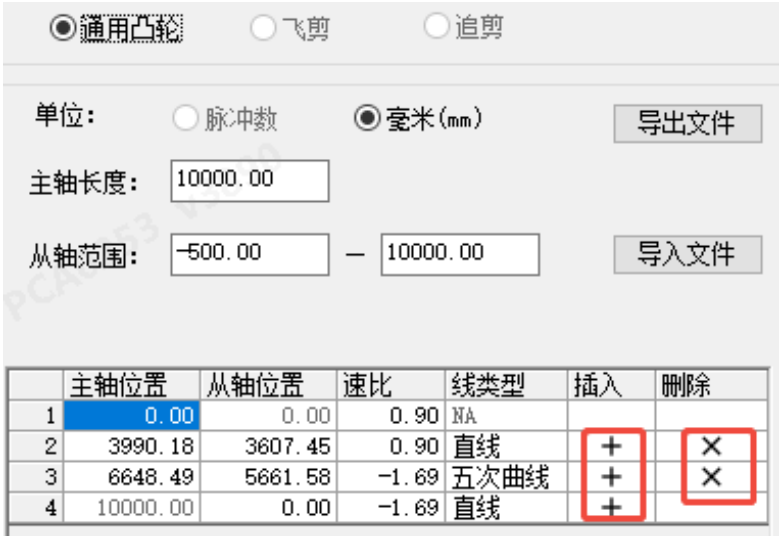


Table 17–1 Cam Node Parameter Descriptions

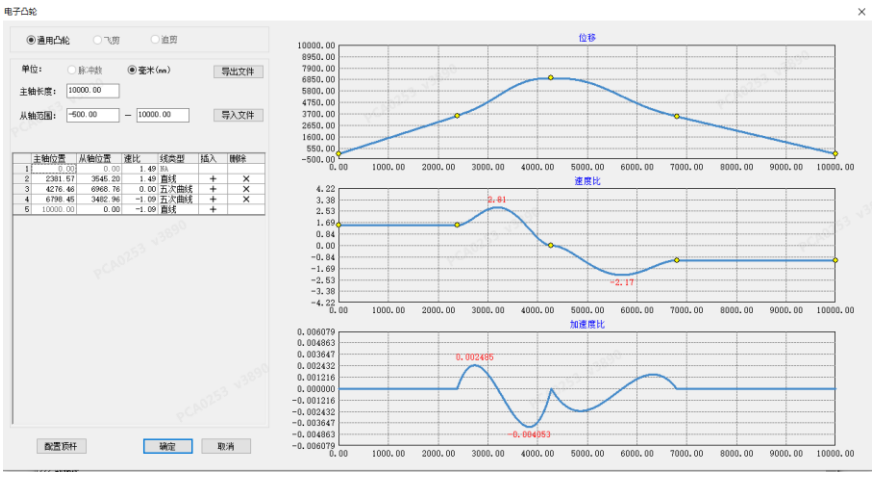
Parameter	Description
M-Pos	Master axis phase (relative mode).
S-Pos	Slave axis displacement (relative mode).
PU-Speed	Connection speed (auto-generated for line curves; manually set for 5th-order splines).
Type	Curve type selection Line: linear; Spline: 5th-order curve

Note

- The first node's M-Pos and S-Pos default to 0 and cannot be modified.
- M-Pos values must be in ascending order.
- The last node's M-Pos defines the master axis cycle length; no separate cycle setting is required.

14.2.4 Cam Curve Configuration

Users can configure cam curves (position, speed ratio, and acceleration ratio curves) in the graphical editing area based on application requirements.



Notes

- For position curves, key points can be dragged vertically/horizontally; for speed ratio curves, key points can only be dragged vertically; for acceleration ratio curves, editing is disabled.
- The last key point can only be vertically dragged; horizontal dragging is disabled. To adjust its horizontal position, manually modify the data in the right-side toolbar.
- Hovering the mouse over any coordinate area displays precise position values.
- Right-click to insert or delete key points.

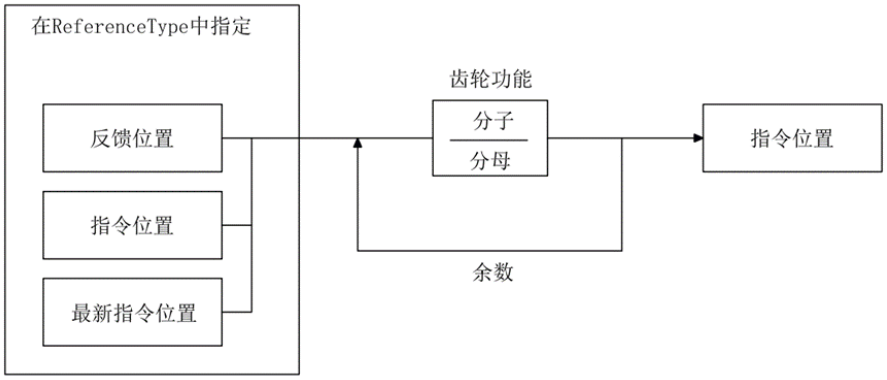
14.2.5 Import/Export

Individual cam tables can be exported/imported. Double-click the target electronic cam table, then select "Import" or "Export" to import/export the cam table in CSV format.

14.3 Electronic Cam Operations

14.3.1 Gear Motion

Functional Diagram



Function Description:

Supported master/slave axis types for gear motion:

- Master axis: Bus servo axis, local pulse axis, local encoder axis, bus encoder axis (currently unsupported).
- Slave axes: Bus servo axis and local pulse axis.

(1) Gear motion is initiated via the MC_GearIn and terminated using MC_GearOut or MC_Stop (forced stop).

(2) After initiation, the slave axis accelerates/decelerates to match the target speed (master axis speed × gear ratio). Catching phase: Before reaching the target speed. InGear phase: After synchronization.

(3) A positive gear ratio drives the Slave axis in the same direction as the Master; a negative ratio drives it in the opposite direction.

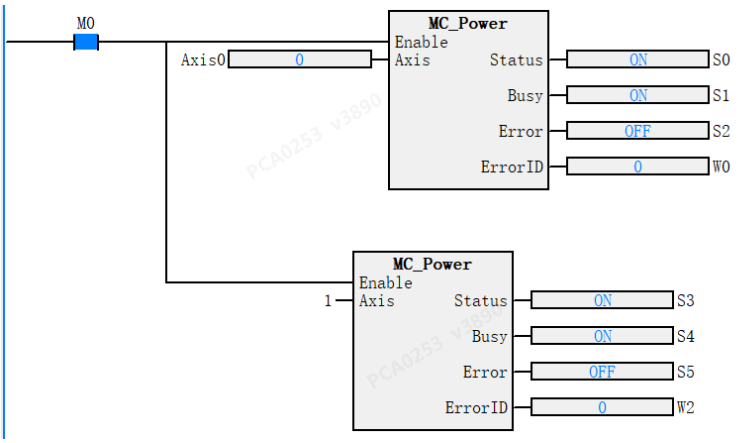
Example

Description: Configure two bus servo axes, with the second axis following the first in a 1:1 gear ratio.

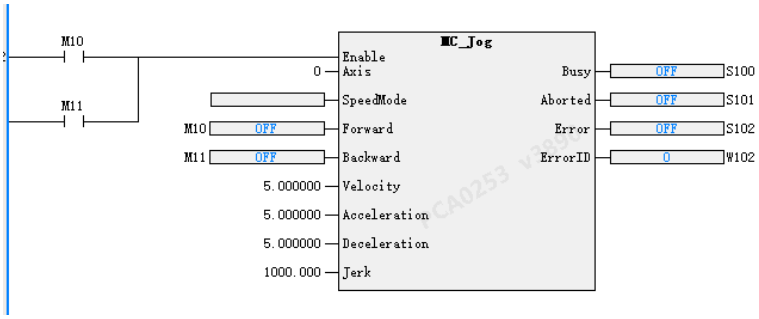
Steps:

1. Create a project with two bus servo axes, with Axis_0 as master axis (bound to SD710 drive) and Axis_1 as slave axis (bound to SD710 drive).

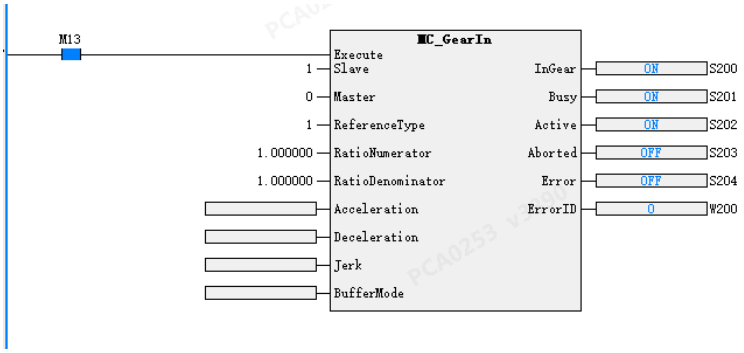
2. Use the MC_Power instruction to enable the master and slave axes.



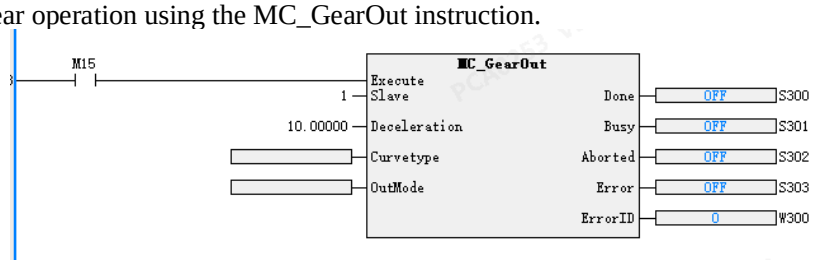
3. Use the MC_Jog instruction to control the master axis's forward/reverse motion.



4. Execute gear synchronization via the MC_GearIn instruction with a gear ratio of 1:1.



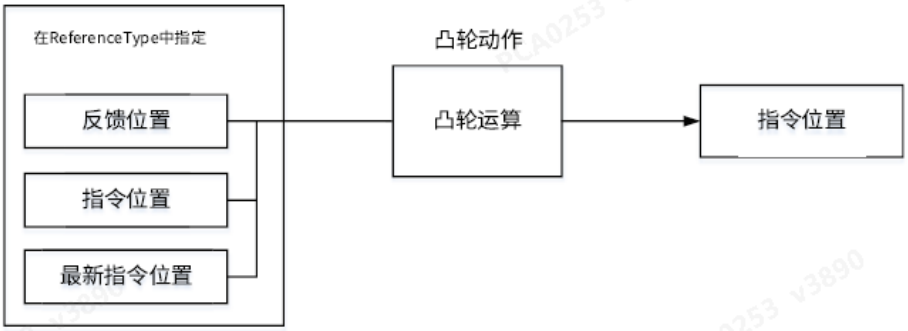
5. Terminate the gear operation using the MC_GearOut instruction.



14.3.2 Cam Motion

Functional Diagram

Cam motion synchronizes the slave axis with the master axis position based on a predefined cam table.



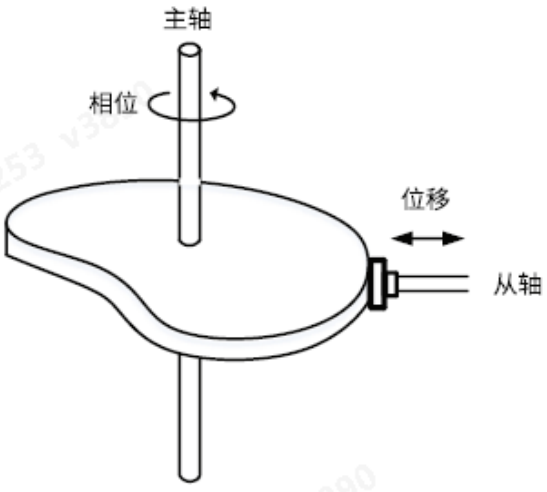
Function Description:

Supported master/slave axis types for cam motion:

- Master axis: Bus servo axis, local pulse axis, local encoder axis, and virtual axis.
- Slave axis: Bus servo axis and local pulse axis.

Cam motion is initiated or modified (cam table switching) via the MC_CamIn instruction, and terminated using MC_CamOut or MC_Stop (forced stop).

Typical cam structure is as follows: The master axis performs periodic rotational motion, while the slave axis executes reciprocating motion under the master axis's control.

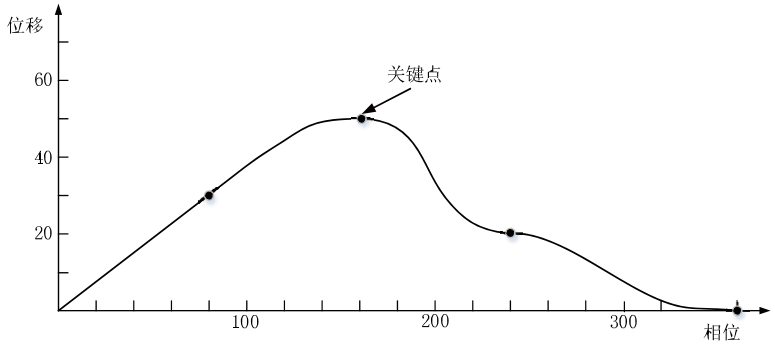


Electronic cams emulate this structure: A master axis (bus servo, local pulse, local encoder, or remote encoder

axis) and a slave axis (bus servo or local pulse axis) synchronize under a configured cam curve.

14.3.3 Cam Curve

A cam curve is defined in a 2D coordinate system where the X-axis represents the master axis phase and the Y-axis represents the slave axis displacement. Key points are set in the system, connected by specified curves (e.g., linear or 5th-order splines), forming the cam curve.



Cam Table

	Phase	Displacement
Start point	0	0
	80	30
	160	50
	240	20
End point	360	0

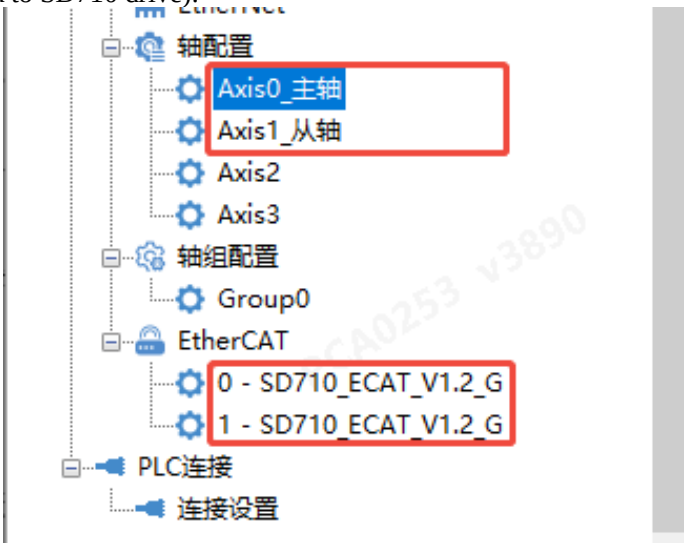
For detailed functionality, refer to the MC_CamIn and MC_CamOut instructions in the "Electronic Cam Instructions" section of the instruction manual.

Example

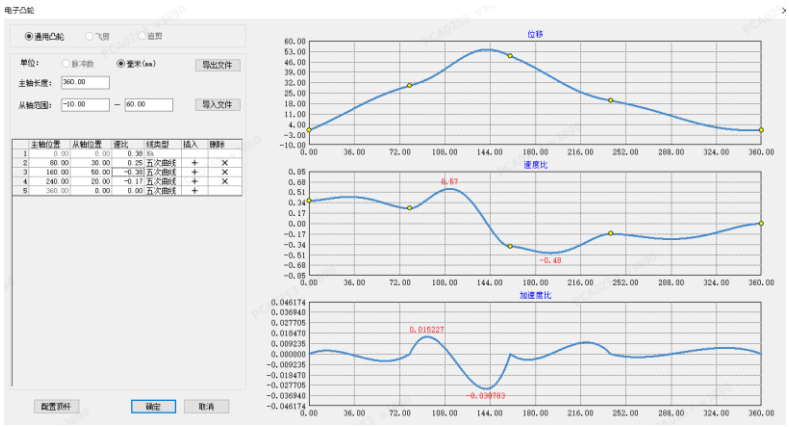
Axis_0 serves as the master cam axis, and Axis_1 follows Axis_0 in cam motion.

Steps:

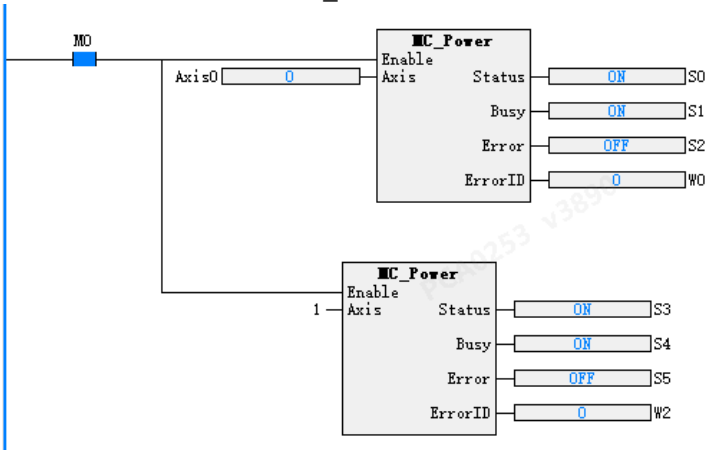
1. Create a project with two bus servo axes, with Axis_0 as master axis (bound to SD710 drive) and Axis_1 as slave axis (bound to SD710 drive).



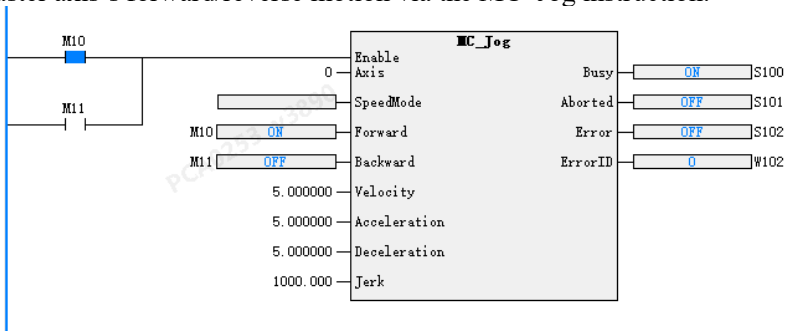
2. Create a cam table.



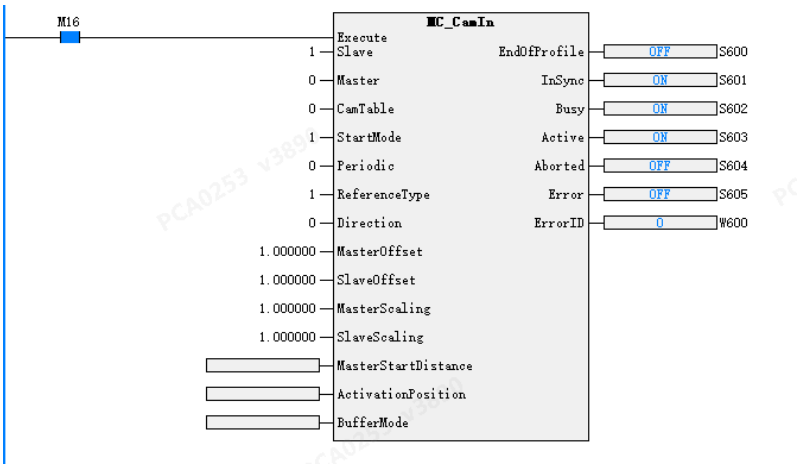
3. Enable the master and slave axes via the MC_Power instruction.



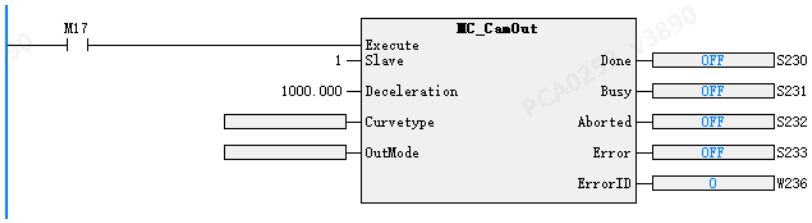
4. Control the master axis's forward/reverse motion via the MC Jog instruction.



5. Execute cam synchronization via the MC_CamIn instruction.



6. Terminate the cam operation using the MC_CamOut instruction.



14.3.4 Cam Table

(I) Cam Table Overview

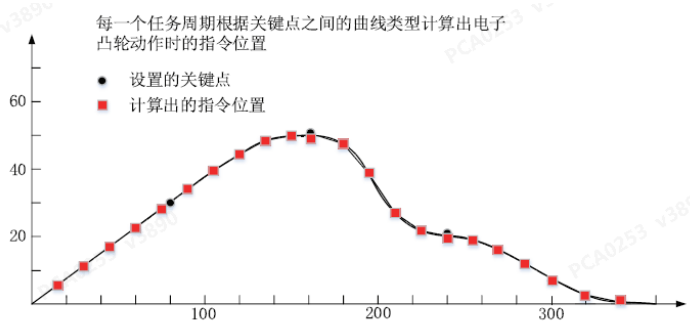
In the cam module, cam data is defined as paired values consisting of master axis phase and slave axis displacement. A cam table comprises multiple sets of cam data.

Phase and displacement values in the cam table are expressed as relative quantities starting from the start point "0.0".

During cam operation, slave axis displacement is calculated based on master axis phase and configured curve type to control slave axis motion.

凸轮表

	相位	位移
起点	0	0
	80	30
	160	50
	240	20
终点	360	0



Cam data within cam tables can be modified through user programs.

14.3.5 Cam Table Specifications

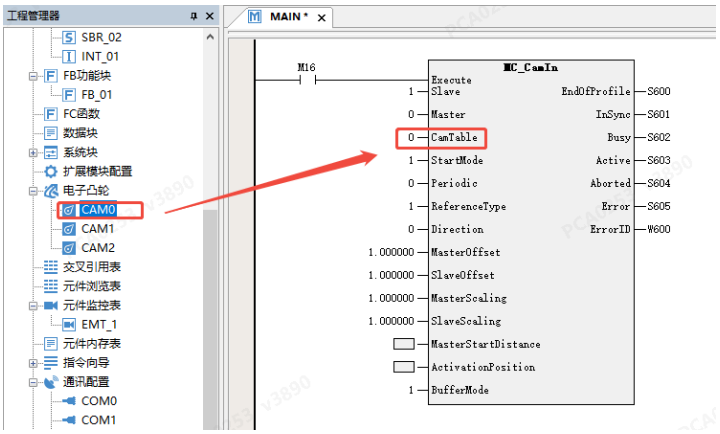
The following specifications apply when creating cam tables:

Table 7-1 Cam Table Specifications

Item	Description
Total key points per cam table	361
Total configurable cam tables	16
Number of cam tables executable simultaneously in PLC	8
Cam table switching during operation	Use MC_CamIn instruction to switch cam tables. Current cam operation must be terminated before reactivating cam coupling.
Cam data read/write	Directly modify cam key points in tables. Changes take effect via MC_GenerateCamTable and apply in the next cam cycle.
Cam table save	Parameters modified through MC_GenerateCamTable are not hold. Original cam table data from the project must be reloaded after power cycle.

14.3.6 Create Cam Tables

Cam table variables can only be created via the background system. Each added cam table automatically generates a cam table variable. CAM0~CAM16 correspond to numbers 0~16 and serve as input parameters for cam instructions (CamTable).



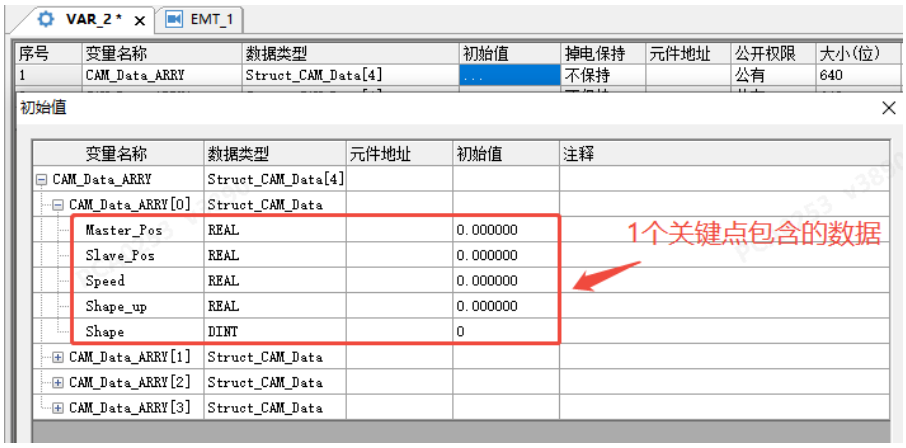
14.3.7 Modify Cam Data

Cam data can be modified through the following methods: Adjust the number of key points in the cam table. Changes take effect via the MC_GenerateCamTable instruction and apply in the next cam cycle.

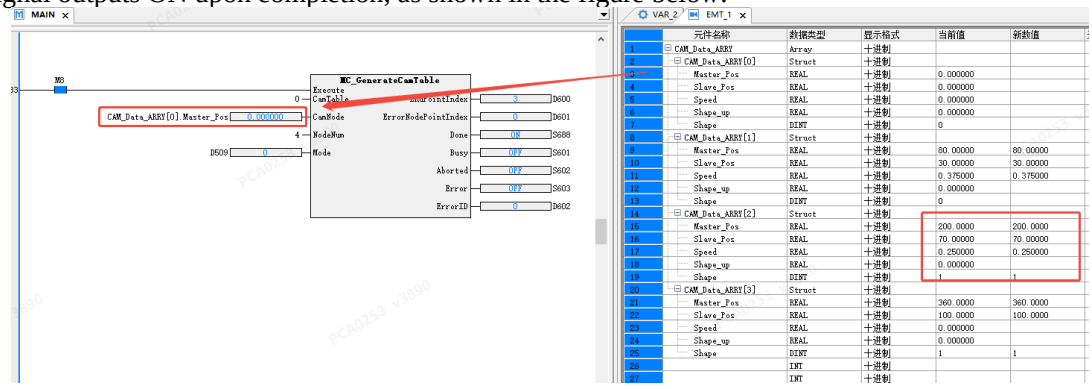
Original				Modified		
Key Point	Master Position	Slave Position		Key Point	Master Position	Slave Position
1	80	30		1	80	30
2	160	50		2	200	70
3	360	100		3	360	100

Program Example:

(1) Manually create a cam table structure array in the program, as shown below.



(2) Execute the MC_GenerateCamTable instruction. Input the starting address of the structure array into the CamNode parameter, modify key point values at corresponding addresses, and set M8 to ON. The Done signal outputs ON upon completion, as shown in the figure below:



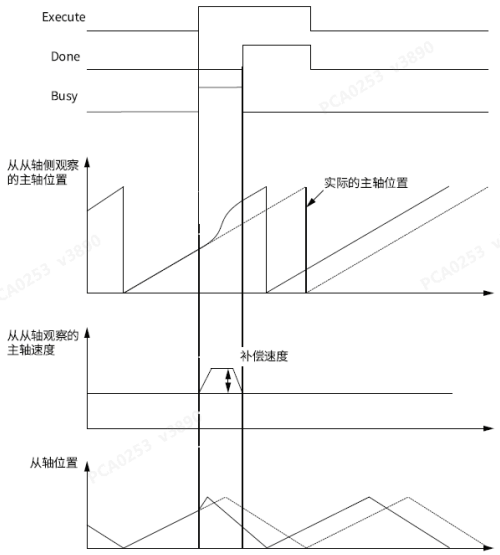
14.3.8 Master Axis Phase Compensation

This function compensates the phase of the master axis (observed from the slave axis) during operation.

The MC_Phasing (master axis relative phase compensation) instruction enables phase compensation for

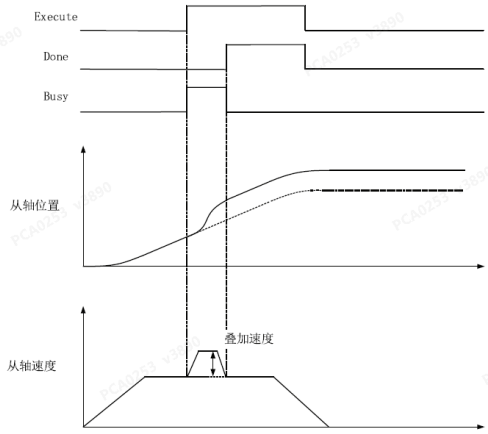
synchronization control,

and configuration for parameters such as phase compensation amount, target velocity, acceleration, and deceleration.



14.3.9 Motion Superposition

The MC_MoveSuperImposed instruction enables motion superposition for motion-controlled axes, and configuration for parameters such as displacement compensation amount, target velocity, acceleration, and deceleration.



15 Offline Simulation

15.1 Overview

AutoSoft V1.12.10.3 and above versions support offline debugging.

- Users can debug program logic offline through this function.
- The offline debugging includes module configuration, communication, online modifications, and a status monitoring interface for real-time tracking of PLC and module I/O channel states. It also integrates with HMI online simulation for hardware-free debugging.

The supported and unsupported functions are listed below.

Supported/Unsupported	Category	Description
Supported	Program	Main program, subprogram, FB/FC
		Timed interrupt subprogram
		Online modification
	Communication	Ethernet Modbus TCP protocol
		HMI online simulation via specified ports
	Instruction	Basic instructions (some are excluded; see details below)
	Extension	Virtual extension module I/O, local DI/DO
	Other functions	RTC clock (uses Windows system clock; PLC time setting not supported).
		Consistent with the PLC
Unsupported	Basic	Time setting
		Entering/exiting offline debugging during compilation, download, or firmware upgrade
	Program	Hardware/edge/comparison interrupts
	Motion control	Bus encoder axis
		Local pulse axis, EtherCAT bus servo axis
		Local encoder axis
	Communication	Serial communication
		CANopen
		EtherCAT communication/instructions
	Instruction	High-speed I/O, control calculation, communication, positioning, electronic cam, EtherCAT, axis group, CANopen axis, MC axis, Ethernet free protocol, local high-speed counter, etc.

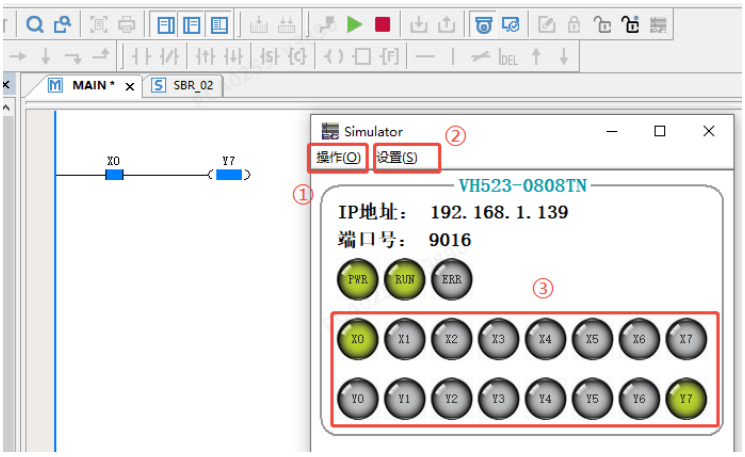
Note

- Unsupported features will remain non-functional but not interfere with offline debugging, requiring no special program modifications.

15.2 Offline Simulation Starting

Prerequisites: A new or existing project requiring offline debugging is opened.

1. Click  in the toolbar to activate the Simulator, and run the PLC program, as shown below:



Interface Description

No.	Description
① Operation	Set PLC state to Run or Stop during offline operation.
② Settings	PLC Type: Select the PLC model for offline simulation. The I/O display quantity will match the selected model. Language: Supports Chinese or English. Top Most: When checked, the simulation panel remains visible over other software interfaces.
③ Panel	I/O Panel: Click X (input) or Y (output) points to toggle their states.
IP address	Displays the IP address and port for the current offline simulation.

Note:
After starting offline debugging, users can:
- Monitor program execution status and PLC I/O states online.
- Modify or download PLC programs.
- Perform online program modifications.

15.3 Digital Terminals Debugging

● Manually toggle the ON/OFF state of uncontrolled I/O point by directly clicking it. If an I/O point is already controlled by the program, its state follows the program logic.

15.4 Simulation Debugging

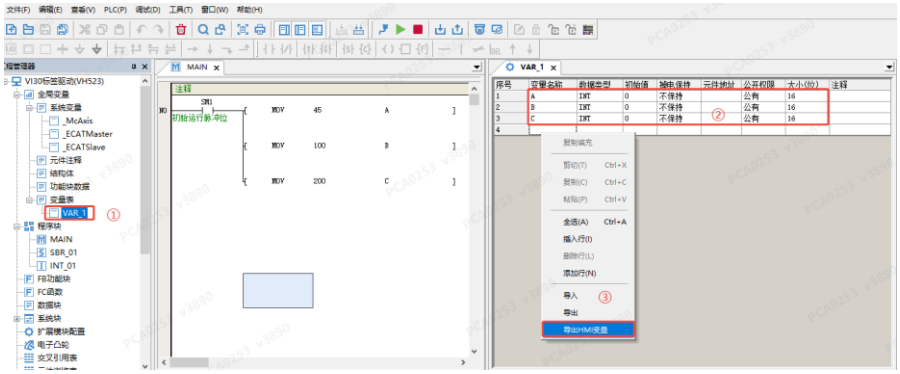
15.4.1 Overview

SH series PLCs and VI20/VI30 can perform hardware-free simulation debugging using AutoSoft, VI20Studio3, or VI30Studio. PLC and HMI communicate via the TCP monitoring protocol, supporting read/write operations for custom variables and soft elements.

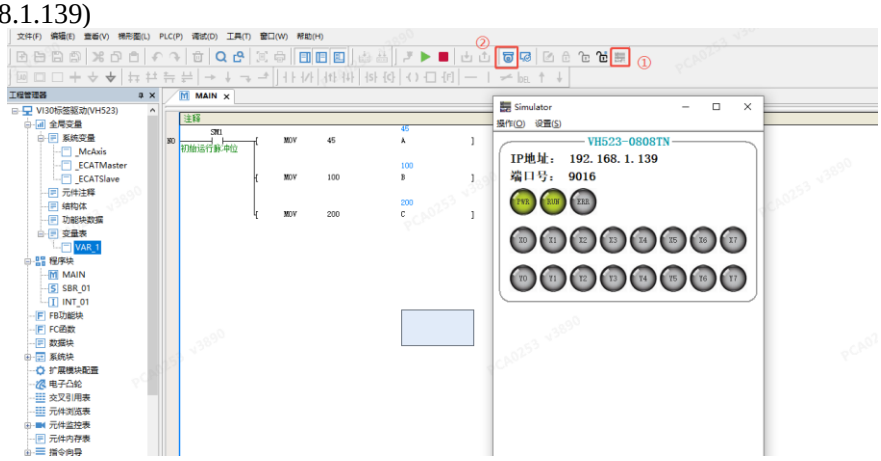
15.4.2 PLC Side Configuration

Internal communication eliminates PLC IP configuration for hardware-free debugging, requiring only PLC variables export to HMI monitoring variable table.

1. After adding variables and compiling the user program, open the VAR_1 variable table. Right-click any area in the table and select Export HMI Variable.



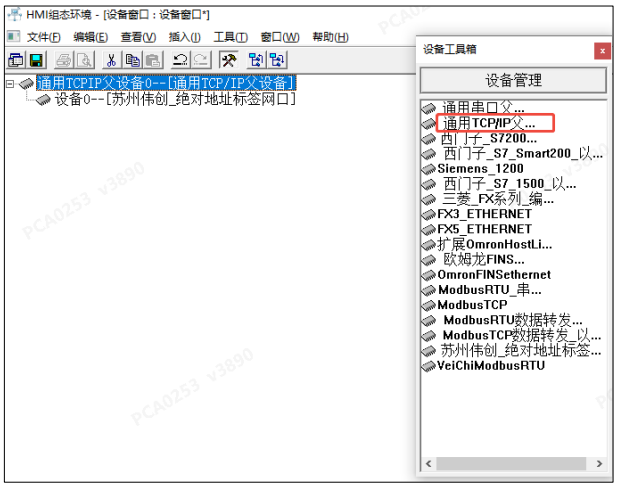
2. Choose the export path, enter a file name, and click Save to generate a .csv file.
3. Click “Offline Simulation”. In the pop-up dialog, click Download, as shown below. (Offline simulation IP address: 192.168.1.139)



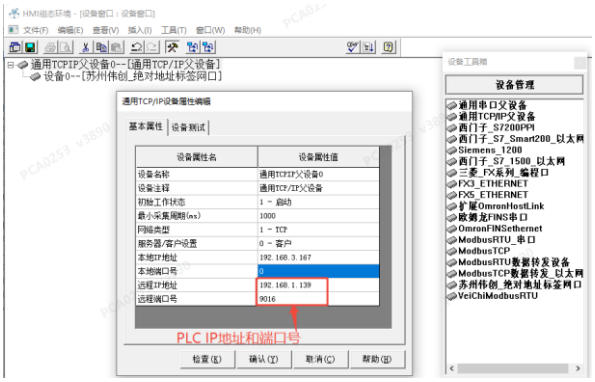
15.4.3 HMI Side Configuration

For hardware-free debugging, add a PLC connection and import the variable table on the HMI side. The steps for VI30 are as follows:

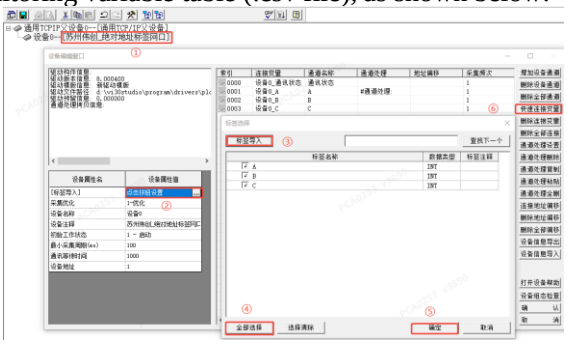
1. Create a new HMI project. Add a TCP/IP Parent Device 0, then click Device Manager to add a net tag, as shown below:



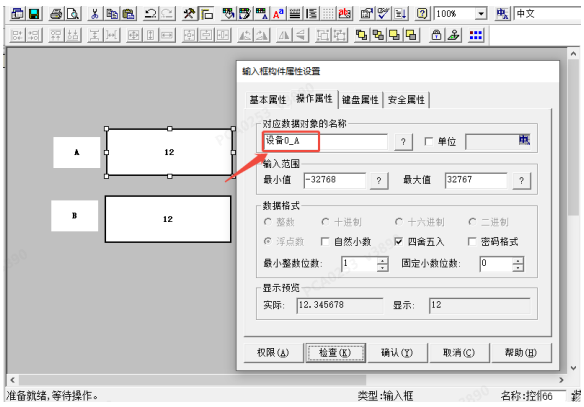
2. Double-click TCP/IP Parent Device 0 > Basic Properties > Set the PLC’s IP address and port address (HMI IP does not require configuration), as shown below:



3. Double-click Net Tag, click "…" in the Import Label row, select target tag, and click Import Label to import them to HMI monitoring variable table (.csv file), as shown below:

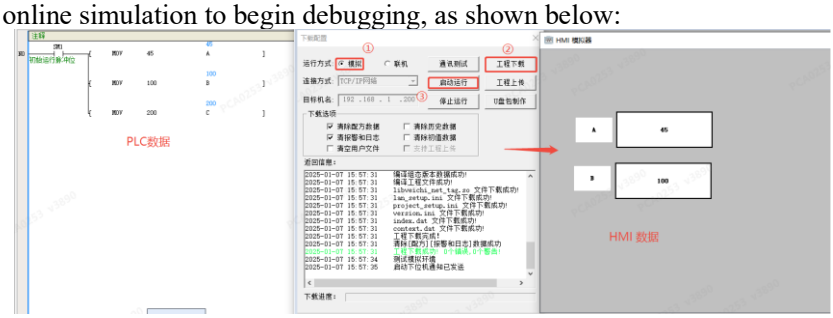


4. Configure the Input Box Properties Setting and bind tag addresses. Compile the project after completion, as shown below.



15.4.4 Debug Starting

After completing HMI and PLC program editing and importing the variable table, launch AutoSoft's offline debugging and VI30's online simulation to begin debugging, as shown below:



16 Troubleshooting

16.1 Hardware Indicators

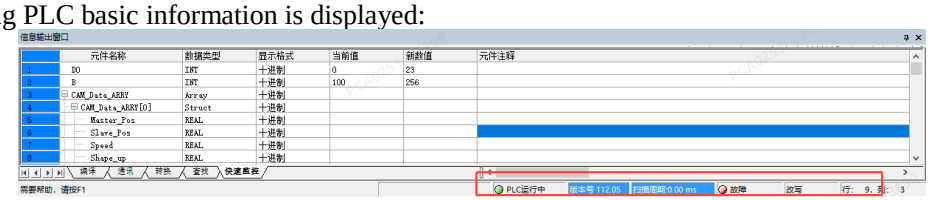
The status of SH series panel indicators are defined as follows:

Type	Mark	Definition	Indicator	Description
IO indicator	IN/OUT	IO status display	Yellow-green	● ON: Input/output is active ● OFF: Input/output is inactive
Operation indicator	PWR	Normal power supply	Yellow-green	● ON: Power supply is normal ● OFF: Power supply is abnormal
	RUN	Normal operation	Yellow-green	● Flash: User program is running ● OFF: User program is stopped
	ERR	Operation error	Red	● OFF: No serious errors ● Flash: A serious error has occurred
	CAT	EtherCAT operation	Yellow-green	● OFF: Communication is unsuccessful ● EtherCAT communication is successful (remains flashing after disconnection)
	CAN	CANopen operation	Yellow-green	● ON: Normal CAN communication ● Flash: Communication is setting up
	EtherNET1	Ethernet indicator	Yellow-green	● ON: Communication is successful ● Flash: Communication is setting up ● OFF: Communication is unsuccessful
	EtherNET2	Ethernet indicator	Yellow-green	● ON: Communication is successful ● Flash: Communication is setting up ● OFF: Communication is unsuccessful

16.2 Software Diagnosis

16.2.1 PLC Basic Information

1. Click the icon on the toolbar to enter monitoring mode. In the bottom-right corner of the interface, the following PLC basic information is displayed:



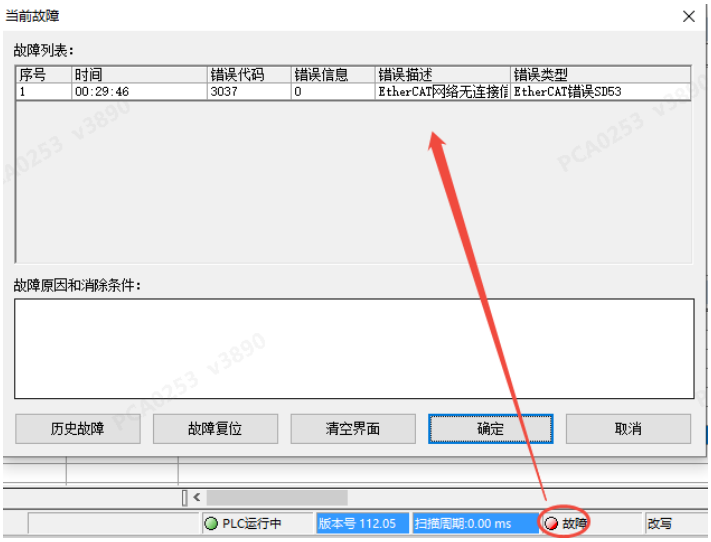
(1) — PLC status indicator and fault indicator

(2) — Current firmware version and program scan cycle

Table 21-1 Description of Indicator Status

Current Status		Fault Status	
Green	PLC is running	Green	No error
Red	PLC is stopped	Red	Error
-	-		

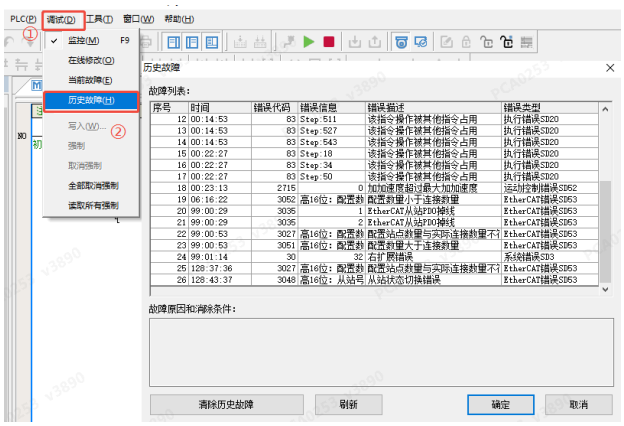
2. Double-click the fault indicator to access the fault diagnosis page and obtain detailed fault information.



16.2.2 Historical Operation Faults

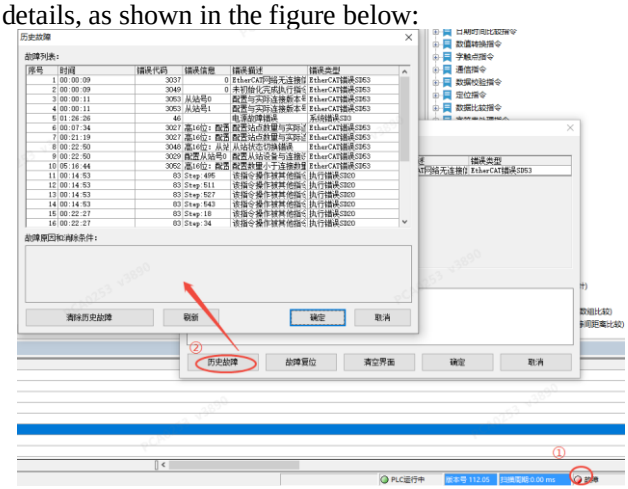
History faults log error messages that occurred during PLC operation. To view history faults:

1. Select Debug > History Fault from the menu bar to access the history fault page, as shown in the figure below.



History faults include: Time, Error Code, Error Info, Error Description, and Error Type.

2. Double-click the fault indicator in the bottom-right corner. A Current Fault dialog will pop up. Select History Fault to view details, as shown in the figure below:



16.3 Error Code

When programming errors occur, the background software displays error codes by category. The table below lists error code categories and corresponding resolutions:

16.3.1 System Errors SD3 (0-59)

Code	Category	Information	Description
10		SRAM error	User program is stopped; error indicator on. Resolution: Download new system configuration file/Format.
11		FLASH error	User program is stopped; error indicator on. Resolution: Power off and check hardware.
12		Communication port error	User program is stopped; error indicator on. Resolution: Power off and check hardware.
13		RTC error	User program is stopped; error indicator on. Resolution: Power off and check hardware.
14		I2C error	User program is stopped; error indicator on. Resolution: Power off and check hardware.
15		FPGA configuration error	User program is stopped; error indicator on. Resolution: Power off and check hardware.
20		Local I/O critical error	User program is stopped; error indicator on. Resolution: Power off and check hardware.
21	Faulty module ID	Extension I/O critical error	Error indicator flashes; auto-cleared when error resolves.
22	Faulty module ID	Special module critical error	Error indicator flashes; auto-cleared when error resolves.
23		RTC refresh error	Error indicator flashes; auto-cleared when error resolves.
24		EEPROM read/write error	Error indicator flashes; auto-cleared when error resolves.
25		Local analog input error	Error indicator flashes; auto-cleared when error resolves.
26		System special module config error	Error indicator flashes; auto-cleared when error resolves.
27	Battery voltage	Low battery voltage	Error indicator flashes; auto-cleared when error resolves.
28		CANOPEN operation error	Error indicator flashes; auto-cleared when error resolves.
29		Left extension config error	Error indicator flashes; auto-cleared when error resolves.
30	0x20: Excessive communication errors	Right extension error	

	0x40~0x4F: Module 0~15 parameter transmission errors 0x60~0x6F: Module 0~15 parameter reception errors 0x80~0x8F: Module 0~15 ID errors 0xA0~0xAF: Module 0~15 parameter polling errors		
40		User program file error	User program is stopped; error indicator on. Resolution: Download new program/Format.
41		System config file error	User program is stopped; error indicator on. Resolution: Download new system configuration file/Format.
42		Data block file error	User program is stopped; error indicator on. Resolution: Download new data block file/Format.
43		Retentive data loss error	User program continues; error indicator flashes. Resolution: Clear data/Format/Reset and no error detected.
44		Force table loss error	User program continues; error indicator flashes. Resolution: Clear data/Format/Reset and no error detected.
45		User info file error	User program continues; error indicator flashes. Resolution: Clear data/Format/Reset and no error detected.
46		Power supply error	User program is stopped; error indicator on. Resolution: Restore power supply.

16.3.2 Execution Errors SD20 (60~255)

Code	Category	Information	Description
60		User program compilation error	User program is stopped; error indicator on.

61		User program execution timeout	User program is stopped; error indicator on.
62	Instruction step number	Illegal user program instruction execution	User program is stopped; error indicator on.
63	Instruction step number	Illegal operand type for instruction	User program is stopped; error indicator on.
64	Instruction step number	Illegal operand value	User program continues; error indicator off, but error type code is recorded in SD20.
65	Instruction step number	Operand address out of range	
66	Instruction step number	Subroutine stack overflow	
67		User interrupt request queue overflow	
68	Instruction step number	Illegal label jump or subprogram call	
69	Instruction step number	Division by zero	
70	Instruction step number	Illegal stack definition	Occurs if stack size/element count is negative, or element count exceeds stack size limit.
71		Reserved	
72	Instruction step number	Undefined user subprogram or interrupt subprogram	
73	Instruction step number	Invalid special module address	
74	Instruction step number	Special module access error	
75	Instruction step number	Immediate I/O refresh error	
76		Clock setting error	
77	Instruction step number	PLSR instruction parameter error	
78	Instruction step number	Special module BFM address out of bounds	
81	Instruction step number	DSZR instruction in abnormal state	
82	Instruction step number	CANOPEN axis control instruction execution error	
83	Instruction step number	Instruction operation occupied by others	
84	Instruction step number	TPID instruction concurrent auto-tuning limit exceeded	
85	Instruction step number	Pointer accesses non-existent memory	
86	Instruction step number	Pointer access out of bounds	
87		Left extension communication error	
88	Instruction step number	User program compilation error	User program is stopped; error indicator on.

16.3.3 Serial Communication Errors SD50 (1000~1499)

Code	Category	Information	Description
1001	Table instruction number	PORT0 illegal function code	
1002	Table instruction number	PORT0 illegal register address	
1003	Table instruction number	PORT0 data count error	

1016	Table instruction number	PORT0 communication timeout; exceeds user-defined maximum communication time.	
1017	Table instruction number	PORT0 received data frame error	
1018	Table instruction number	PORT0 parameter error; invalid mode or master/slave configuration.	
1019	Table instruction number	PORT0 station ID conflict; local station ID matches instruction-set ID.	
1020	Table instruction number	PORT0 address overflow; Received/sent data exceeds element storage space.	
1021	Table instruction number	PORT0 command execution failed	
1022	Table instruction number	PORT0 received address conflicts with requested address; stored in error code element	
1023	Table instruction number	PORT0 received function code conflicts with requested code; stored in error code element	
1024	Table instruction number	PORT0 frame error: Element start address mismatch; stored in error code element	
1025	Table instruction number	PORT0 data length/element count exceeds protocol/function code limits.	
1032	Table instruction number	PORT0 CRC/LRC check error	
1034	Table instruction number	PORT0 element start address error	
1035	Table instruction number	PORT0 unsupported/illegal function code	
1036	Table instruction number	PORT0 element count error	
1038	Table instruction number	PORT0 parameter modification prohibited during operation	
1039	Table instruction number	PORT0 parameter password-protected	
1040	Reserved		
1101	Table instruction number	PORT1 illegal function code	
1102	Table instruction number	PORT1 illegal register address	
1103	Table instruction number	PORT1 data count error	
1116	Table instruction number	PORT1 communication timeout; exceeds user-defined maximum communication time.	
1117	Table instruction number	PORT1 received data frame error	
1118	Table instruction number	PORT1 parameter error; invalid mode or master/slave configuration.	
1119	Table instruction number	PORT1 station ID conflict; local station ID matches instruction-set ID.	
1120	Table instruction number	PORT1 address overflow; Received/sent data exceeds element storage space.	
1121	Table instruction number	PORT1 command execution failed	
1122	Table instruction number	PORT1 received address conflicts with requested address; stored in error code element	
1123	Table instruction number	PORT1 received function code conflicts with requested code; stored in error code element	

1124	Table instruction number	PORT1 frame error: Element start address mismatch; stored in error code element	
1125	Table instruction number	PORT1 data length/element count exceeds protocol/function code limits.	
1132	Table instruction number	PORT1 CRC/LRC check error	
1134	Table instruction number	PORT1 element start address error	
1135	Table instruction number	PORT1 unsupported/illegal function code	
1136	Table instruction number	PORT1 element count error	
1138	Table instruction number	PORT1 parameter modification prohibited during operation	
1139	Table instruction number	PORT1 parameter password-protected	
1201	Table instruction number	PORT2 illegal function code	
1202	Table instruction number	PORT2 illegal register address	
1203	Table instruction number	PORT2 data count error	
1216	Table instruction number	PORT2 communication timeout; exceeds user-defined maximum communication time.	
1217	Table instruction number	PORT2 received data frame error	
1218	Table instruction number	PORT2 parameter error; invalid mode or master/slave configuration.	
1219	Table instruction number	PORT2 station ID conflict; local station ID matches instruction-set ID.	
1220	Table instruction number	PORT2 address overflow; Received/sent data exceeds element storage space.	
1221	Table instruction number	PORT2 command execution failed	
1222	Table instruction number	PORT2 received address conflicts with requested address; stored in error code element	
1223	Table instruction number	PORT2 received function code conflicts with requested code; stored in error code element	
1224	Table instruction number	PORT2 frame error: Element start address mismatch; stored in error code element	
1225	Table instruction number	PORT2 data length/element count exceeds protocol/function code limits.	
1232	Table instruction number	PORT2 CRC/LRC check error	
1234	Table instruction number	PORT2 element start address error	
1235	Table instruction number	PORT2 unsupported/illegal function code	
1236	Table instruction number	PORT2 element count error	
1238	Table instruction number	PORT2 parameter modification prohibited during operation	
1239	Table instruction number	PORT2 parameter password-protected	

16.3.4 Ethernet-based CAN Communication Errors SD51 (1500~1999)

Code	Category	Information	Description
1501	Faulty slave IP last byte	Ethernet master receive timeout	
1502	Faulty slave IP last byte	Ethernet slave response error	
1503	Faulty slave IP last byte	Ethernet master connection failure	
1504		Ethernet slave exceeds maximum connections	
1601		RAM allocation error	
1602		Invalid data or out of range	
1603		Incomplete CAN program	
1604		CANopen configuration error	
1605		CAN data download error	
1606		CAN unknown error	
1607		CAN transmit buffer overflow	
1608		CAN receive buffer overflow	
1609		CAN general error	
1610		CAN passive error	
1611		CAN bus off	
1612		CAN heartbeat error	
1613		CAN protocol error	
1614		CANPDO length error	
1615		CANRPDO timeout	
1616		CAN overload	
1617		CANPDO transmit/receive error	
1618		CANPDO transmission type error	
1619		CAN received invalid message	
1620		CAN received emergency message	
1621		CAN slave count exceeds limit	
1622		CANSDO invalid command code	
1623		CAN download error	
1624		CANSDO write data error	
1625		CAN Sync frame timing error	
1626		SDO read timeout	
1627		SDO write timeout	
1628		CAN instruction/protocol mismatch	
1629		CAN free protocol instruction exceeds limit	
1630		Axis communication error	
1631		Servo fault	
1632		CAN axis control instruction: axis not enabled	
1633		CAN axis control instruction: axis busy	
1637		Forward overtravel	
1638		Reverse overtravel	
1640		Axis number out of range	
1641		CANopen not configured	
1642		CANopen monitor address setting error	

1643		CAN receive overflow	
1644		CAN origin return error	
1645		Module error; not specified motion module	
1649		SDO read/write aborted (See SD393 for object dictionary index)	
1650		Instruction count exceeds limit	
1651	_	Invalid data type	
1800	_	Socket ID error	
1801	_	Socket port error	
1802	_	Socket port or ID already exists	
1803	_	Failed to create Socket listener	
1804	_	Failed to bind Socket port	
1805	_	Max. number of socket ports exceeds limit	
1806	_	Socket pointer error	
1807	_	Socket listener port already closed	
1808	_	Socket connection port already closed	
1809	_	Socket closed	
1810	User-defined socket	Socket data reception error	
1811	User-defined socket	Data sent via unconnected Socket	
1812	User-defined socket	Multiple consecutive Socket data transmission errors	
1813	_	Socket data size exceeds limit	
1814	User-defined socket	Host disconnected	
1815	_	Ethernet initialization incomplete	

16.3.5 EtherCAT Error Codes (SD53)

Code	Category	Information
2001	Axis No.	EtherCAT servo axis configured but EtherCAT task not enabled
2002	Axis No.	EtherCAT servo axis mapped to an invalid EtherCAT slave number
2003	Axis No.	Axis operation state error
2004	Axis No.	Axis position deviation alarm
2005	Axis No.	Hardware positive limit triggered
2006	Axis No.	Hardware negative limit triggered
2007	Axis No.	Software positive limit triggered
2008	Axis No.	Software negative limit triggered
2009	Axis No.	Target velocity exceeds maximum limit
2010	Axis No.	Acceleration exceeds maximum limit
2011	Axis No.	Deceleration exceeds maximum limit
2012	Axis No.	Acceleration step value exceeds maximum limit
2013	Axis No.	Torque exceeds maximum limit
2014	Axis No.	Necessary PDO not mapped for the axis
2501	-	Cam channel error (exceeds maximum cam count)
2502	-	Invalid cam table number setting
2503	-	Cam direction/tracking distance/start position mismatch
2504	-	Invalid cam point count
2505	-	Invalid activation mode

2506	-	Cam point data error
2507	-	Invalid follower count
2508	-	Follower data error
2509	-	Invalid cam axis number setting
2510	-	Cam start/end point calculation error
2511	-	Master axis phase exceeds 100 cycles
2512	-	CAM stop mode switch failed
2513	-	Axis already in coupling state (repeated)
2514	-	Master/slave scaling ratio is zero
2515		Invalid cam table data retrieval mode
2516		No data in cam table buffer
2517		No data during cam table operation
2520		Flying shear mode setting error
2521		Flying shear data setting error
2522		Tracking shear mode setting error
2523		Tracking shear data setting error
2524		Flying shear return coefficient error
2551	-	Gear axis number setting error
2552	-	Gear denominator setting error
2553	-	Gear acceleration setting error
2554	-	Gear deceleration setting error
2555	-	Gear jerk setting error
2601	Axis No.	Master axis position change exceeds limit
2602	Axis No.	Slave axis position change exceeds limit
2603	Axis No.	Slave axis not enabled
2604	Axis No.	Slave axis acceleration exceeds maximum limit
2605	Axis No.	Slave axis deceleration exceeds maximum limit
2606	Axis No.	Slave axis jerk exceeds maximum limit
2607	Axis No.	CAM mode switch failed during axis operation
2700	Axis Group No.	Axis group count exceeds limit
2701	Axis Group No.	Invalid axis group number configuration
2702	-	Axis group number does not exist or is unconfigured.
2703	Axis Group No.	Axis number out of range
2704	-	Invalid axis position within group
2705	-	Axis group or axis not in stopped state
2706	-	Duplicate activation of axis group instruction
2707	-	Velocity mode setting error
2708	-	Velocity range setting error
2709	-	Acceleration range setting error
2710	-	Deceleration range setting error
2711	-	Jerk range setting error
2712	-	Velocity exceeds maximum limit
2713	-	Acceleration exceeds maximum limit
2714	-	Deceleration exceeds maximum limit
2715	-	Jerk exceeds maximum limit
2716	-	Absolute/relative mode setting error

2717	-	Buffer mode setting error
2718	-	Transition mode setting error
2719	-	Transition parameter setting error
2720	-	Axis group instruction exceeds buffer limit
2721	-	Stop instruction in progress
2722	-	Stop mode setting error
2723	-	Not all axes in group are enabled
2724	-	Circular interpolation plane setting error
2725	-	Circular interpolation mode setting error
2726	-	Circular interpolation path setting error
2727	-	Specified interpolation axis does not exist
2728	-	No interpolation axis in group
2729	-	Circular interpolation start/end/midpoint collinear.
2730	-	Radius setting error
2731		Interpolation instruction type error
2732		Look-ahead calculation delayed
2733		Look-ahead buffer overflow
2734		Reloading prohibited during look-ahead operation
2735		Look-ahead line number error
2736		Look-ahead auxiliary parameter error
2737		Invalid look-ahead start line
2738	-	Look-ahead interpolation parameter error
2800		Axis number out of range
2801	-	Instruction activated while axis is running
2802	-	Axis not enabled
2803	-	60FF_PDO not mapped
2804	-	Pulse width out of range

16.3.6 MC Axis Instruction Error Codes

Code	Category	Information
16	Instruction error	Instruction interrupted
17	Instruction error	SDO parameter write error
18	Instruction error	SDO parameter read error
256	Instruction error	ECAT not initialized or slave connection failed
257	Instruction error	Configuration error
258	Instruction error	Axis not in running state
259	Instruction error	Invalid instruction
260	Instruction error	Bus servo axis alarm detected
261	Instruction error	SDO write error
262	Instruction error	SDO read error
263	Instruction error	Origin return error
264	Instruction error	Origin return timeout
265	Instruction error	Abnormal instruction execution
266	Instruction error	Axis in motion
267	Instruction error	Axis not enabled
268	Instruction error	Axis already in use
269	Instruction error	Axis stop error
270	Instruction error	Axis in stopped state

271	Instruction error	Non-bus servo axis or local pulse axis
272	Instruction error	PDO not configured
273	Instruction error	Reserved
274	Instruction error	Axis not running
275	Instruction error	JOG motion paused
276	Instruction error	Non-bus servo axis
277	Instruction error	PDO unsupported
278	Instruction error	Reserved
279	Instruction error	Reserved
280	Instruction error	Reserved
281	Instruction error	Negative limit input signal not used
282	Instruction error	Invalid axis type
288	Instruction error	Positive limit input signal not used
289	Instruction error	Zero-point input signal not used
290	Instruction error	Negative limit input signal out of range
291	Instruction error	Positive limit input signal out of range
292	Instruction error	Zero-point input signal out of range
293	Instruction error	Z-pulse input out of range
294	Instruction error	Counter type error
295	Instruction error	Probe enable input missing
296	Instruction error	Invalid counter number
297	Instruction error	Comparison channel occupied
298	Instruction error	Comparison exception
299	Instruction error	Comparison output not set
300	Instruction error	Illegal exit from JOG mode
301	Instruction error	Velocity too low
302	Instruction error	SDO access busy
303	Instruction error	Not in position mode
304	Instruction error	Parameter change incomplete
305	Instruction error	Data out of range
306	Instruction error	Enable input preset missing
307	Instruction error	Counter interrupt mapping error
308	Instruction error	Comparison count exceeds limit
309	Instruction error	Positive/negative direction active simultaneously in JOG mode
310	Instruction error	Circular interpolation start/end/midpoint collinear
311	Instruction error	Operation prohibited while axis is enabled
312	Instruction error	PDO input mapping error
313	Instruction error	PDO output mapping error
314	Instruction error	Invalid axis number
315	Instruction error	Positive hardware limit triggered
316	Instruction error	Negative hardware limit triggered
317	Instruction error	Positive software limit triggered
318	Instruction error	Negative software limit triggered
319	Instruction error	Position deviation alarm
320	Instruction error	No error detected during axis reset

17 Firmware Upgrade

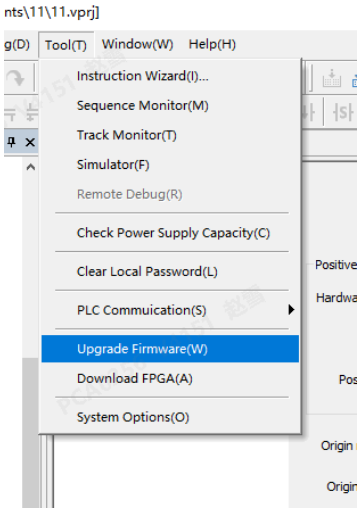
To update device functionality and meet evolving requirements, customers can upgrade the firmware via two methods: using AutoSoft programming software or an SD card.

17.1 Firmware Upgrade via Host Computer

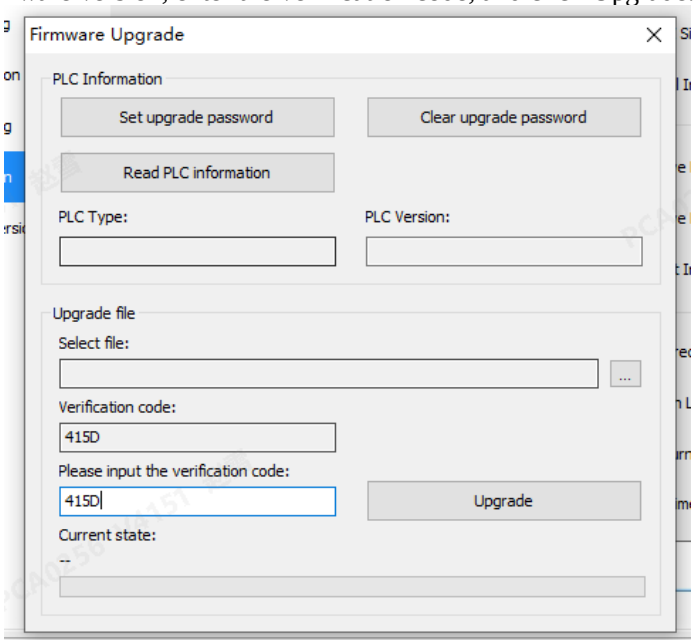
17.1.1 MCU Firmware Upgrade

SH series PLCs support firmware upgrades via Ethernet or USB using AutoSoft. The existing application program will be retained after the upgrade.

1. Navigate to Tools > Upgrade Firmware in the menu bar to open the firmware upgrade dialog.



2. Select the target firmware version, enter the verification code, and click Upgrade.



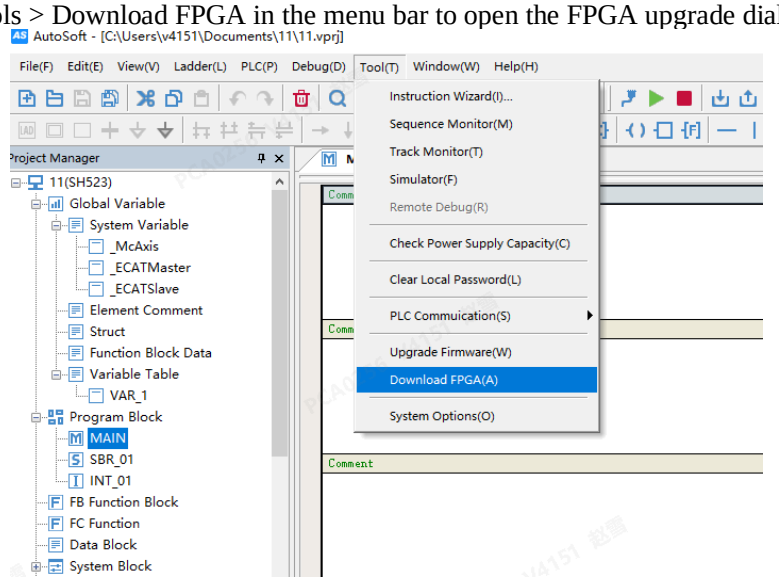
3. When prompted to place the PLC in STOP mode, click OK to start the upgrade. The process completes when the "Upgrade Successful" message appears. Power cycle the PLC to finalize.

Notes

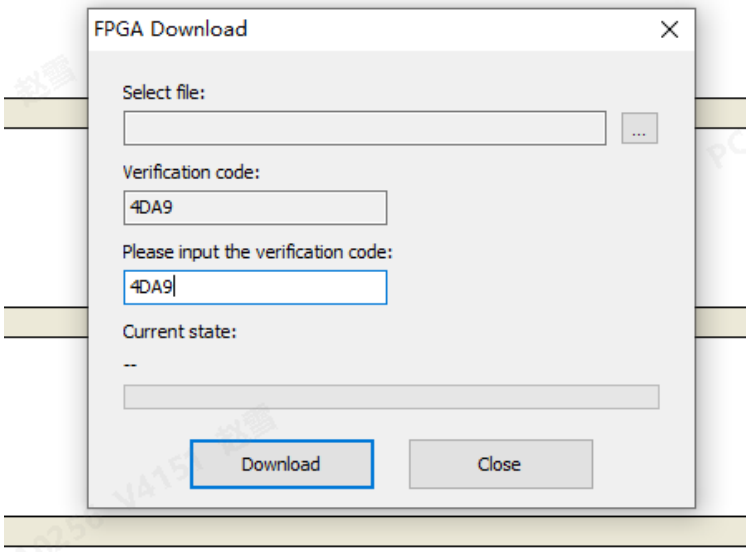
- Firmware versions must be obtained from the manufacturer or authorized distributors.
- Firmware versions must be obtained from the manufacturer or authorized distributors.
- Maintain stable power supply during the upgrade. Power loss may render the PLC inoperable (indicated by a slowly flashing RUN indicator). If this occurs, reattempt the upgrade. If unsuccessful, return the device for repair.
- Always power cycle the device after upgrading. For USB upgrades, disconnect the USB cable before restarting.

17.1.2 FPGA Upgrade

1. Navigate to Tools > Download FPGA in the menu bar to open the FPGA upgrade dialog.



2. Select the target FPGA version, enter the verification code, and click Upgrade.



3. When prompted to place the PLC in STOP mode after clicking Download, click OK to start the upgrade. The process completes when the "Upgrade Successful" message appears.

Notes

- FPGA versions must be obtained from the manufacturer or authorized distributors.
- Ensure the PC is connected to the SH device before upgrading.
- Maintain stable power supply during the upgrade. Power loss may render the PLC inoperable (indicated by a slowly flashing RUN indicator). If this occurs, reattempt the FPGA upgrade. If unsuccessful, return the device for repair.
- Always power cycle the device after upgrading. For USB upgrades, disconnect the USB cable before restarting.

17.2 Firmware Upgrade via SD Card

17.2.1 MCU Firmware Upgrade

Firmware upgrades can be performed through either host computer connection or SD card. Follow these steps:

1. Prepare an SD card formatted as FAT32, with capacity ≤32GB (as shown in the diagram below).

2. Insert the SD card into a card reader and connect it to a computer via USB port. Verify the file format is FAT32 by Right-click>Properties.

3. Copy UpCfg.ini and System.bin to the SD card, insert the SD card into the PLC device and power it on. Monitor the upgrade status via panel indicators. (Note: Source files available from manufacturer/authorized agents)

Update Progress	Indicator Status
Updating...	PWR/ERR/CAT/CAN/RUN indicators ON (Note: No CAT indicator in SH300 series)
Update succeeded	After completion: - Only PWR remains steady on. - RUN flashes during system operation. Remove the SD card after successful program upgrade to prevent automatic re-update on reboot. (ERR may stay on if connected via USB/Ethernet without user program)
Update error	ERR flashes slowly + RUN displays 1 long + 2 short flashes. Host communication via USB/Ethernet is disabled.

17.2.2 FPGA Upgrade

1. Copy both UpCfg.ini and FPGA.bin to the SD card, insert the SD card into the PLC device and power it on. Monitor the upgrade status via panel indicators. (Note: Source files available from manufacturer/authorized agents)

Update Progress	Indicator Status
Updating...	PWR/ERR/CAT/CAN/RUN indicators ON (Note: No CAT indicator in SH300 series)
Update succeeded	After completion: - Only PWR remains steady on. - RUN flashes during system operation. Remove the SD card after successful program upgrade to prevent automatic re-update on reboot. (ERR may stay on if connected via USB/Ethernet without user program)
Update error	ERR flashes slowly + RUN displays 1 long + 2 short flashes. Host communication via USB/Ethernet is disabled.

17.2.3 Simultaneous MCU&FPGA Upgrade

1. Copy System.bin, FPGA.bin, and UpCfg.ini to the SD card, insert the SD card into the PLC device and power it on. Monitor the upgrade status via panel indicators. (Note: Source files available from manufacturer/authorized agents)

Update Progress	Indicator Status
Updating...	PWR/ERR/CAT/CAN/RUN indicators ON (Note: No CAT indicator in SH300 series)
Update succeeded	After completion: - Only PWR remains steady on.

	- RUN flashes during system operation. Remove the SD card after successful program upgrade to prevent automatic re-update on reboot. (ERR may stay on if connected via USB/Ethernet without user program)
Update error	ERR flashes slowly + RUN displays 1 long + 2 short flashes. Host communication via USB/Ethernet is disabled.

警 告

● Power interruption is strictly prohibited during SD card burning to avoid critical failures such as PLC malfunction.

● Existing application programs remain preserved after firmware burning.

17.3 Application Download

17.3.1 Overview

The application download function allows compiling a PLC project into a .cmf file, enabling users to download programs without accessing the original project files. This function includes:

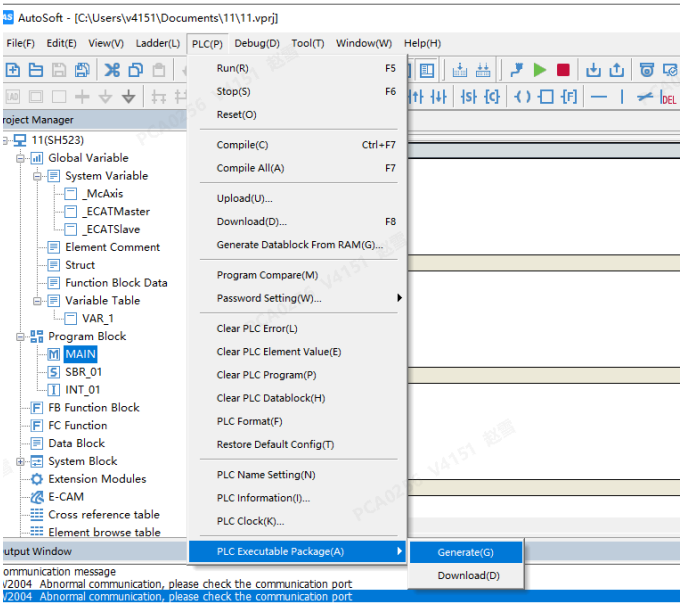
(1) Batch PLC project updates/upgrades using an SD card;

(2) PLC project updates via the AutoSoft background software.

17.3.2 Generate .cmf File

To generate a .cmf file using AutoSoft, follow these steps:

1. Open the PLC project, and navigate to Menu Bar > PLC > PLC Executable Package (A)> Generate to generate the file as shown below.



2. In the Save As dialog, select the output path, name the file, and click Save to generate the .cmf file, as shown below.

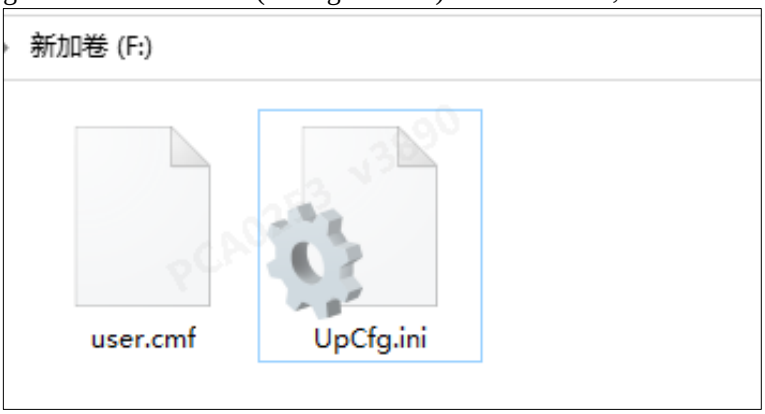
17.3.3 PLC Project Update Example via SD Card

Example: Update the application project via SD card. (In this case, the generated file is named user.cmf. Refer to 17.3.2 for .cmf file generation steps.)

1. Prepare an SD card formatted as FAT32.
2. Open the UpCfg.ini file in Notepad, enter “user.cmf[SPACE]4”, append “//” on a new line as as the end

marker, and then save the modification. This is shown in the figure below:

3. Copy both UpCfg.ini and user.cmf file (PLC-generated) to the SD card, as shown below.



4. Safely eject the SD card, insert it into the PLC and power it on. The upgrade progress indicator will display the status.

Update Progress	Indicator Status
Updating...	PWR/ERR/CAT/CAN/RUN indicators ON (Note: May not activate for small projects.)
Update succeeded	After completion: - Only PWR remains steady on. - RUN flashes during system operation. Remove the SD card after successful program upgrade to prevent automatic re-update on reboot. (ERR may stay on if connected via USB/Ethernet without user program)
Update error	ERR flashes slowly + RUN displays 1 long + 2 short flashes. Host communication via USB/Ethernet is disabled.

CAUTION

- UpCfg.ini format requirements:

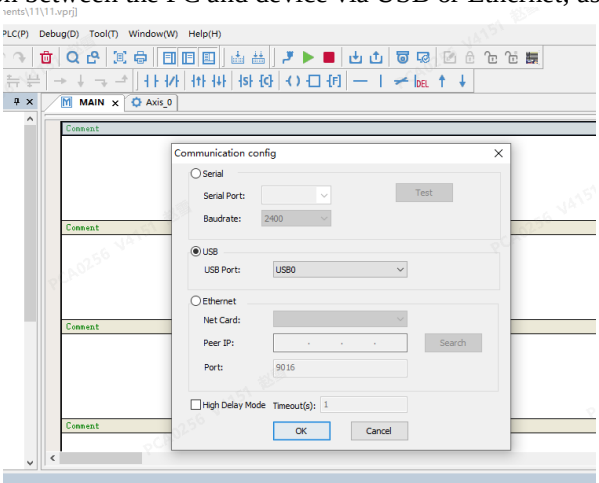
Single-space delimiter only appears between filename and number; No extra spaces or empty lines allowed elsewhere.>

- // marks configuration end (subsequent text ignored); Maximum file size: 500 characters.
- Project upload are disabled by default during SD card program updates.

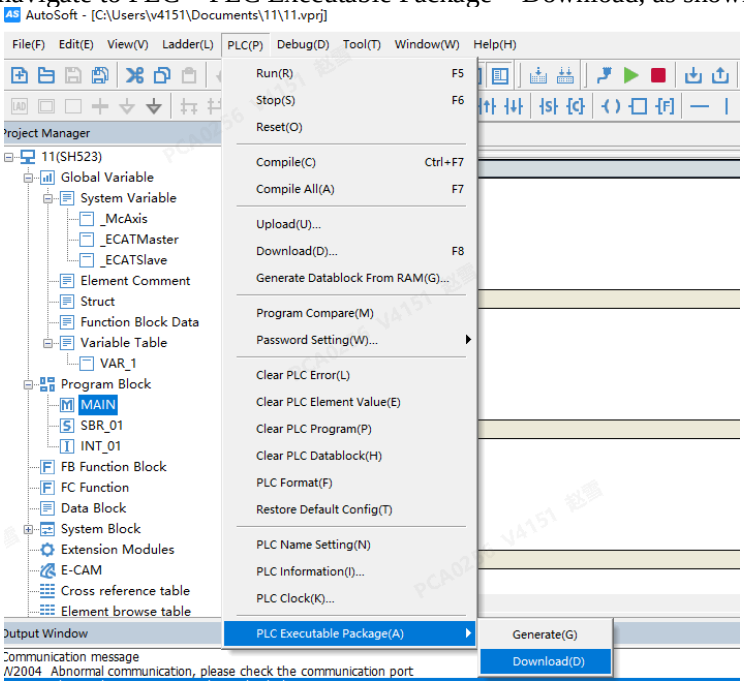
17.3.4 PLC Project Update Example via PC

Example: Download the generated .cmf file via AutoSoft.

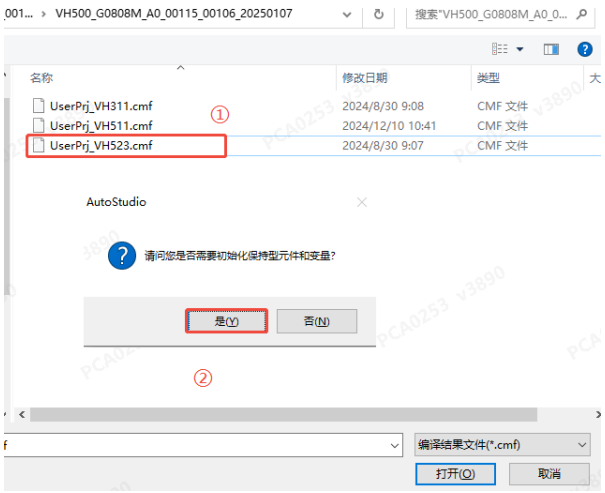
1. Establish communication between the PC and device via USB or Ethernet, as shown in the figure below:



2. In the menu bar, navigate to PLC > PLC Executable Package > Download, as shown below.



3. Select the target .cmf file. A dialog will prompt: "Load initial values for retentive elements and variables?" Choose Yes to load initial values or No to skip.



4. A progress bar displays during the download. Upon completion, a prompt asks: "Set the PLC to RUN mode?" Click Yes to start the PLC and finalize the upgrade.