InoProShop(V1.3.50.0)

User Guide

Medium-Sized PLC Programming Software

AM400/AM600/AC800

User Guide

Contents

Chapter 1 Product Information

1. Product Information

1.1 Overview

1.1.1 Product Information

Inovance medium-sized programmable logic controllers (PLCs), including AM400, AM600, and AC800 series, provide intelligent automation solutions for users. These PLCs use the IEC61131-3 programming language, and supports six programming languages of the PLCopen standard. The AM400 and AM600 use the rack structure, with each rack supporting 16 local expansion modules and also remote expansion modules through multiple field buses such as PROFIBUS DP, EtherCAT and CANopen. The AM600 local expansion modules, that is, I/O modules, are connected through the internal bus protocol, including digital input (DI), digital output (DO), analog input (AI), analog output (AO), and temperature modules. The AM600 has the following functions:

(1) High-performance motion control function through the EtherCAT bus

(2) Single-axis acceleration/deceleration control, electronic gear, and electronic cam, and basic single-axis positioning with the maximum frequency of 200 kHz through the high-speed I/O

(3) Communication functions through the RS485, Ethernet, and USB ports

The AC800 adopts the booksize structure and excellent motion control of up to 256 axes, meeting various application requirements of users.

The medium-sized PLC has the following features:

1) Multiple motion control functions: bus motion control and pulse motion control

2) Abundant I/O channels, up to ten thousand channels

3) Large program capacity and data storage

4) Fast instruction execution speed

5) Various high-end field buses including PROFIBUS DP, EtherCAT, and CANopen

6) Easy-to-use software

7) Online debugging mode

8) Online editing mode

The following table lists the CPU module types and their function differences.

Table 1-1 Functional characteristics of different CPU module types

Product Model	Number of Local Expansion Modules	Program Storage Space	Data Storage Space	Data Stored at Power Failure	Motion Control Axes	High-speed I/O	Soft Element	Output Type
AM401-CPU1608TP	8	10 MB	20 MB	512 KB	4 servo axes, 4 positioning axes	16 inputs and 8 outputs	$\sqrt{}$	Source
AM402-CPU1608TP	8	10 MB	20 MB	512 KB	8 servo axes, 4 positioning axes	16 inputs and 8 outputs	$\sqrt{}$	Source

Product Model	Number of Local Expansion Modules	Program Storage Space	Data Storage Space	Data Stored at Power Failure	Motion Control Axes	High-speed I/O	Soft Element	Output Type
AM401-CPU1608TN	8	10 MB	20 MB	512 KB	4 servo axes, 4 positioning axes	16 inputs and 8 outputs	√	Sink
AM402-CPU1608TN	8	10 MB	20 MB	512 KB	8 servo axes, 4 positioning axes	16 inputs and 8 outputs	√	Sink
AM600-CPU1608TP	16	10 MB	20 MB	512 KB	32	16 inputs and 8 outputs	√	Source
AM600-CPU1608TN	16	10 MB	20 MB	512 KB	32	16 inputs and 8 outputs	√	Sink
AM610-CPU1608TP	16	10 MB	20 MB	512 KB	X	16 inputs and 8 outputs	√	Source
AC802-0222-U0R0	X	128 MB	128 MB	5 MB (external UPS required)	128	X	X	X
AC810-0122-U0R0	X	128 MB	128 MB	5 MB (external UPS required)	256	X	X	X

Table 1-2 Communication characteristics of the CPU module

Product Model	Communication				
	EtherCAT	PROFIBUS DP	CANopen/CANlink	Modbus TCP	Modbus (Serial Port)
AM401-CPU1608TP	1 channel (maximum of 128 slave stations)	X	1 channel (maximum of 8 slave stations)	1 channel (maximum of 63 slave stations)	1 channel (maximum of 31 slave stations)
AM402-CPU1608TP	1 channel (maximum of 128 slave stations)	X	1 channel (maximum of 8 slave stations)	1 channel (maximum of 63 slave stations)	1 channel (maximum of 31 slave stations)
AM401-CPU1608TN	1 channel (maximum of 128 slave stations)	X	1 channel (maximum of 8 slave stations)	1 channel (maximum of 63 slave stations)	1 channel (maximum of 31 slave stations)
AM402-CPU1608TN	1 channel (maximum of 128 slave stations)	X	1 channel (maximum of 8 slave stations)	1 channel (maximum of 63 slave stations)	1 channel (maximum of 31 slave stations)
AM600-CPU1608TP	1 channel (maximum of 128 slave stations)	X	1 channel (maximum of 63 slave stations)	1 channel (maximum of 63 slave stations)	2 channels (maximum of 31 slave stations for each)
AM600-CPU1608TN	1 channel (maximum of 128 slave stations)	X	1 channel (maximum of 63 slave stations)	1 channel (maximum of 63 slave stations)	2 channels (maximum of 31 slave stations for each)

Product Model	Communication				
	EtherCAT	PROFIBUS DP	CANopen/CANlink	Modbus TCP	Modbus (Serial Port)
AM610-CPU1608TP	X	1 channel (maximum of 124 slave stations)	X	1 channel (maximum of 63 slave stations)	2 channels (maximum of 31 slave stations for each)
AC802-0222-U0R0	2 channels (maximum of 64 slave stations)	X	X	2 channels (maximum of 63 slave stations)	2 channels (maximum of 31 slave stations for each)
AC810-0122-U0R0	2 channels (maximum of 128 slave stations)	X	X	2 channels (maximum of 63 slave stations)	2 channels (maximum of 31 slave stations for each)

Note: The number of slave stations supported does not include the power module and stopper plate.

1.1.2 Product Configuration and Module Description

Inovance provides a rich range of medium-sized PLC products for users to select the required product configuration according to the applications. Take the AM600 series PLC as an example. The AM600 includes two structures: AM600-CPU1608TP (EtherCAT+CANopen) and AM610-CPU1608TP (PROFIBUS DP). The following are the typical system integration diagrams of the two structures.
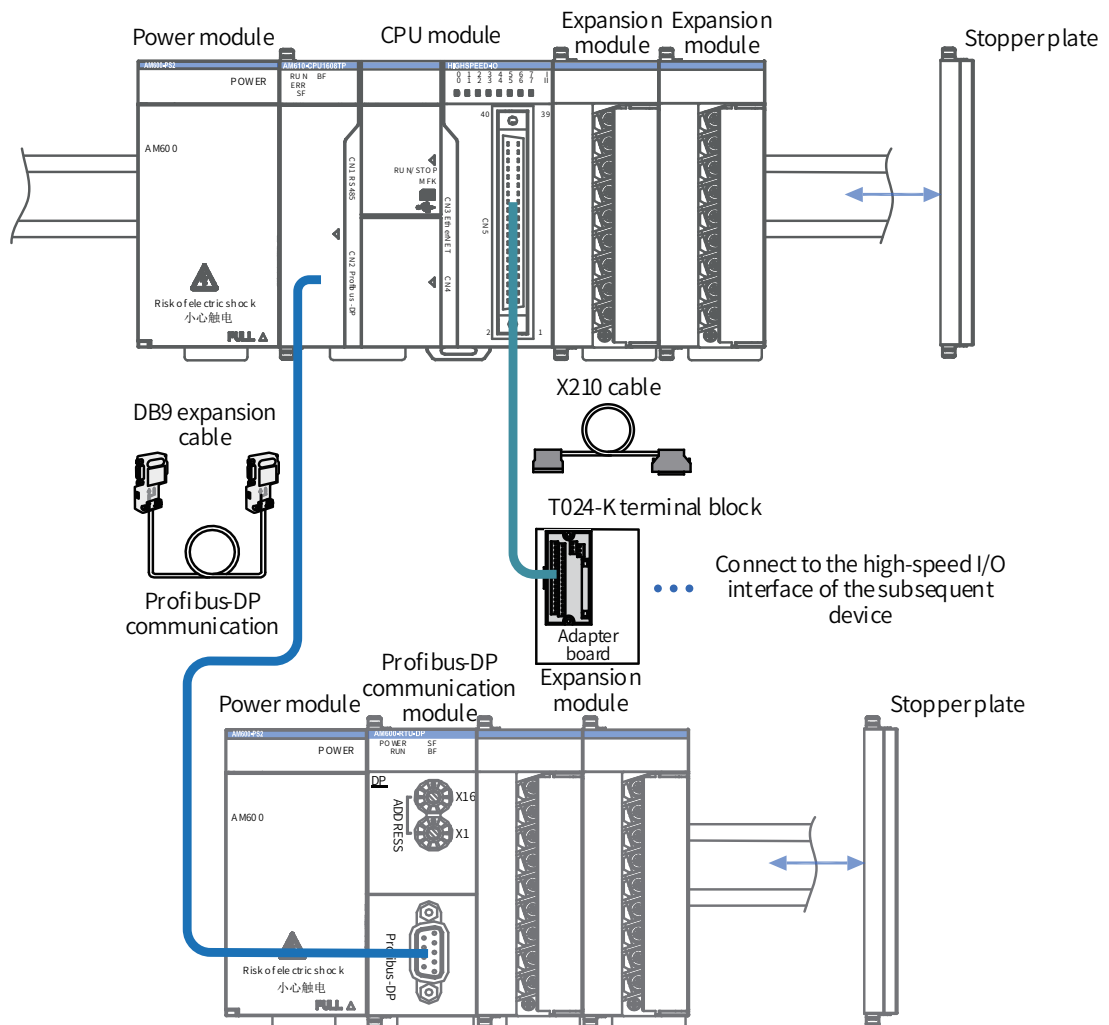
Figure 1-1 AM610-CPU1608TP typical system integration diagram
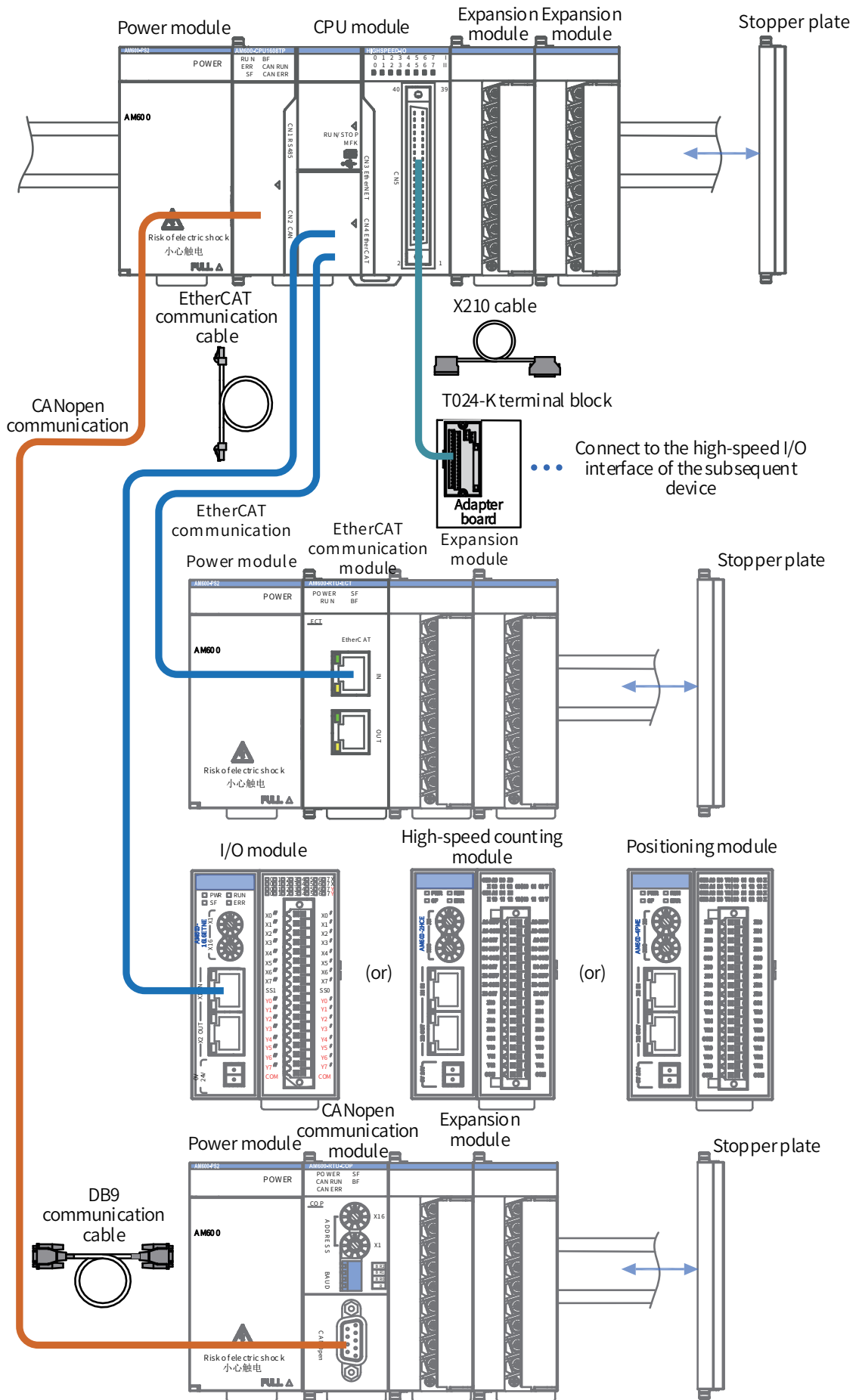
Figure 1-2   AM600-CPU1608TP system integration diagram

The AM600 modules are classified into the power module, CPU module, remote communication modules and local expansion modules based on functions, as described as follows.

Ordering Code	Model	Category	Descriptions
01440010	AM600-PS2	Power module	220 V voltage input, 24 V/2 A output
01440028	AM401-CPU1608TP	CPU module	1 x RS485, 1 x CANopen/CANlink, 1 x LAN 4-axis motion control, EtherCAT Built-in *16 high-speed inputs, 8 high-speed outputs* Source output
01440029	AM402-CPU1608TP	CPU module	1 x RS485, 1 x CANopen/CANlink, 1 x LAN 8-axis motion control, EtherCAT Built-in 16 high-speed inputs, 8 high-speed outputs Source output
01440032	AM401-CPU1608TN		1 x RS485, 1 x CANopen/CANlink, 1 x LAN 4-axis motion control, EtherCAT Built-in 16 high-speed inputs, 8 high-speed outputs Sink output
01440031	AM402-CPU1608TN		1 x RS485, 1 x CANopen/CANlink, 1 x LAN 8-axis motion control, EtherCAT Built-in 16 high-speed inputs, 8 high-speed outputs Sink output
01440014	AM600-CPU1608TP	CPU module	2 x RS485, 1 x CANopen/CANlink, 1 x LAN Basic motion control, EtherCAT Built-in 16 high-speed inputs, 8 high-speed outputs Source output
01440016	AM610-CPU1608TP		2 x RS485, 1 x LAN Basic motion control, PROFIBUS DP Built-in 16 high-speed inputs, 8 high-speed outputs Source output
01440038	AC810-0122-U0R0	Booksize controller	2 x USB2.0, 2 x USB3.0 1 x RS485/RS232, 4 x LAN Up to motion control of 256 axes, EtherCAT Multi-functional expansion slot, built-in Mini-PCIE expansion slot
01440101	AC802-0222-U0R0		2 x USB2.0, 2 x USB3.0 1 x RS485/RS232, 4 x LAN Up to motion control of 256 axes, EtherCAT Built-in Mini-PCIE expansion slot
01440005	AM600-1600END	DI module	16-channel DI module, 24 VDC input (source/sink)
01440017	AM600-0016ER	DO module	16-channel DO module, relay output
01440003	AM600-0016ETP		16-channel DO module, transistor output (source)
01440018	AM600-0016ETN		16-channel DO module, transistor output (sink)
01440006	AM600-4AD	AI module	4-channel AD module, voltage/current analog input
01440007	AM600-4DA	AO module	4-channel DA module, voltage/current analog output
01440012	AM600-RTU-DP	PROFIBUS DP communication module	PROFIBUS DP communication module

Ordering Code	Model	Category	Descriptions
01440013	AM600-RTU-ECT	EtherCAT communication module	EtherCAT communication module
01440011	AM600-RTU-COP	CANopen communication module	CANopen communication module

1.1.3 System Application Process

The system application process is shown in the following figure. For module installation and wiring, see the *AM600 Series PLC Hardware Manual* (CN, EN) and *AC810 Series PLC Hardware Manual* (only CN currently).

Start

Confirmation

Install the modules

Install the power, CPU, and expansion modules.

Carry out wiring

Finish wiring of the power module.
Finish wiring of the CPU and expansion modules.

Connect the system power

Confirm the wiring is correct and the power supply meets with required specifications, and the CPU module is in STOP state. Then, switch on the power

Connect to the PC

Connect the PC where the programming tool is installed to the CPU module.

For installation of the programming tool, see section 1.2

Write the program and parameters into the CPU

Write the program created with the programming tool and the related parameters into the CPU module.

Locate the error

Ensure that the LEDs of the CPU module and ERR, SF and BF indicators of other modules are off.
Use the programming tool to locate the error when an indicator is on or blinks.

Run the program

Set the CPU module to RUN state and verify that the RUN indicator is on.

Figure 1-3 Basic application flowchart of medium-sized PLCs

1.2 Overview of InoProShop

1.2.1 Brief Introduction

InoProShop is a programming configuration software for medium-sized PLCs. Developed based on CoDeSys V3 (shorted as "CoDeSys"), InoProShop provides a complete configuration, programming, commissioning, and monitoring environment for medium-sized PLCs with flexible processing on IEC languages.

InoProShop is used to manage projects and devices, and supports the following configurations on medium-sized PLCs:

9) CPU configuration

10) I/O module configuration

11) EtherCAT bus

12) PROFIBUS DP bus

13) CANopen/CANlink bus

14) Modbus/Modbus TCP bus

15) High-speed I/O

Besides programming, downloading, and commissioning, this software also provides the following functions:

■ Standard programming (IEC 61131-3 compliant)

The software supports multiple programming languages, including structured test (ST), function block diagram (FBD), instruction list (IL), ladder diagram (LD), sequential function chart (SFC), and continuous function chart (CFC) of IEC61131-3 extended programming language.

■ Flexible function block (FB) library

The software supports full FB library and user-defined library.

■ Offline simulation

Users can complete program commissioning simulation without connecting the PLC hardware.

■ Intelligent error locating

The software quickly locates errors during pre-programming and programming and provides diagnostics and logs.

■ Sampling tracking

The software can establish the timing diagram of process variables.

1.2.2 Connection Between InoProShop and Hardware

Connect the programming device to the PLC through Ethernet (such as hub or switch) or USB, write user programs in the InoProShop, and download the program to the PLC to monitor the program and control the PLC.
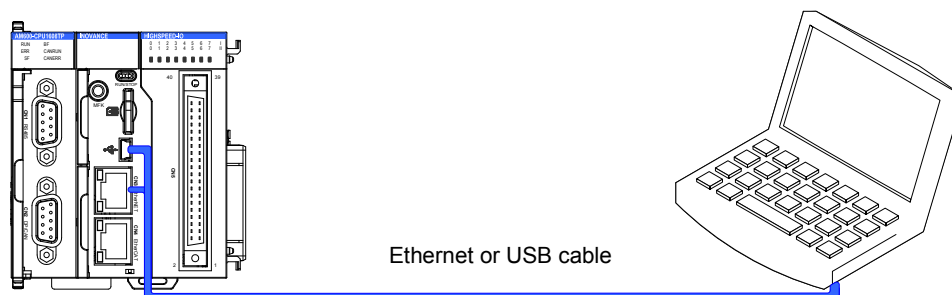
Figure 1-4 Connection between InoProShop and hardware

1.2.3 Acquisition and Installation Requirements

1 Software acquisition

Inovance InoProShop is available for free. You can obtain the installation file and related reference using the following methods:

■   Obtain the software installation CD from Inovance distributors at all levels.

■   Access www.inovance.com, and download the software installation package for free.

Inovance is committed to continuous improvement of its products and documents. You are advised to update software versions in time and refer to the latest documents when designing your applications.

2 Installation environment requirements

Use a desktop or laptop PC that meets the following requirements:

■   Operating system: Windows 7 or Windows 10; 64-bit operating system recommended

■   Memory: 4 GB or above

■   Available hard disk space: 5 GB or above

Connect the PC and the PLC as follows:

Connection Mode	Cable	Remarks
LAN network cable (recommended)	One idle LAN port and one network cable required on the local network	Long-distance connection between the PC and the PLC is supported, and the interaction communication is fast. For example, you can sit in the office and program the PLC in the workshop.
USB cable	One USB cable, of which the end connecting the PLC must be a Mini USB connector	Currently, the AM400 and AM600 series support this connection mode.

1.2.4 Installation Procedure

1 Preparation

If you install InoProShop for the first time, ensure that the available space in the destination partition of the hard disk is above 5 GB; in this case, you can directly install the software.

If you need to upgrade InoProShop, back up your work files, uninstall the original InoProShop, restart the PC, and then install the new InoProShop version.

2 Installation

Locate the directory where the installation package is stored in the Windows Explorer, and double-click the InoProShop (V*.*.*.*).exe file (V*.*.*.* is the InoProShop version. Ensure that you are installing the latest version.)

The following preparation interface is displayed:
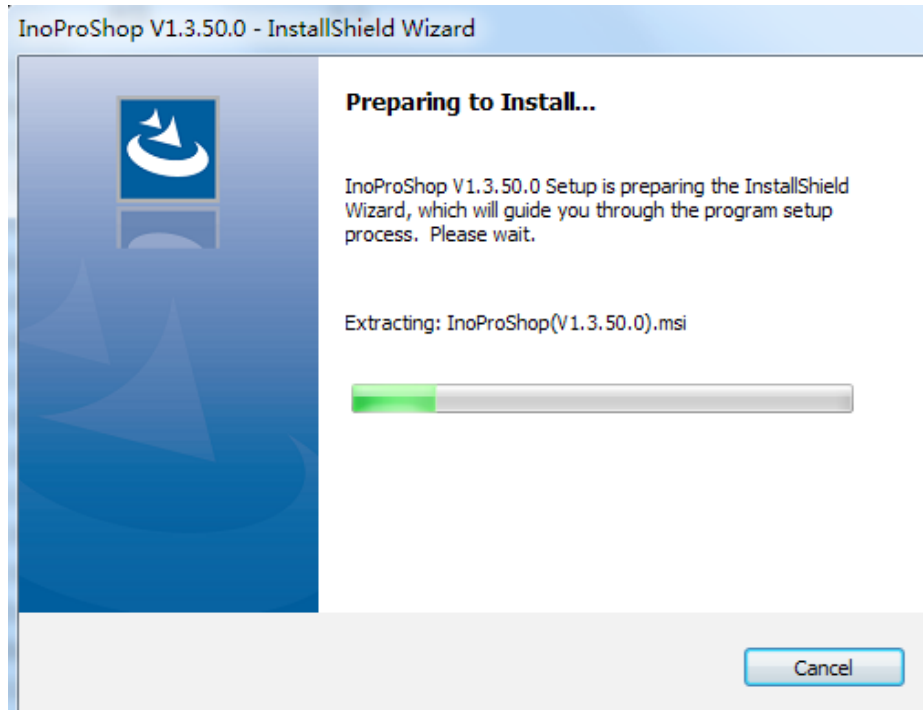
On the displayed interface, click **Next**.
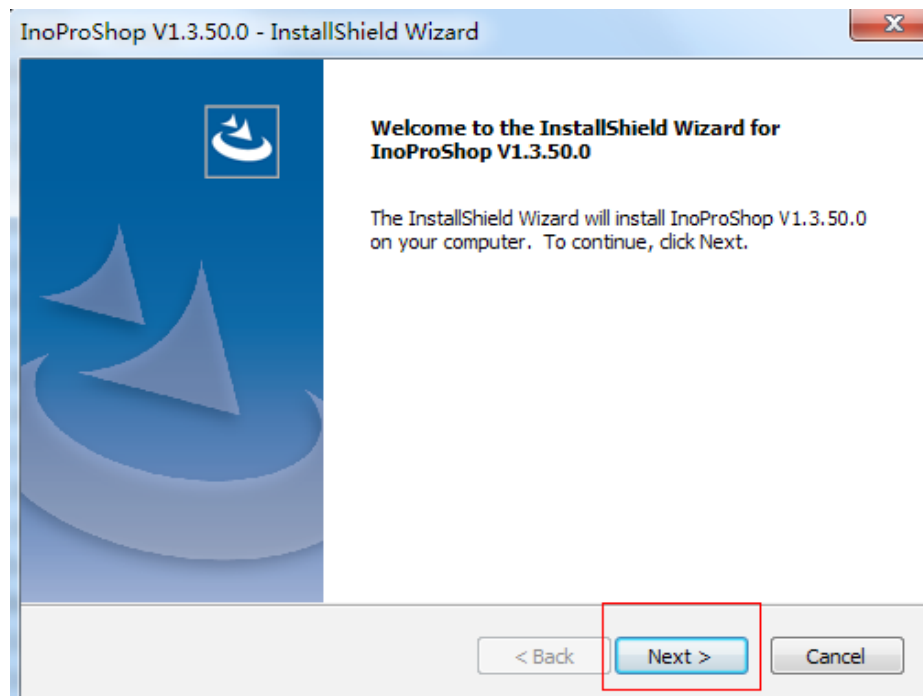
Set the installation path, and click **Next**.
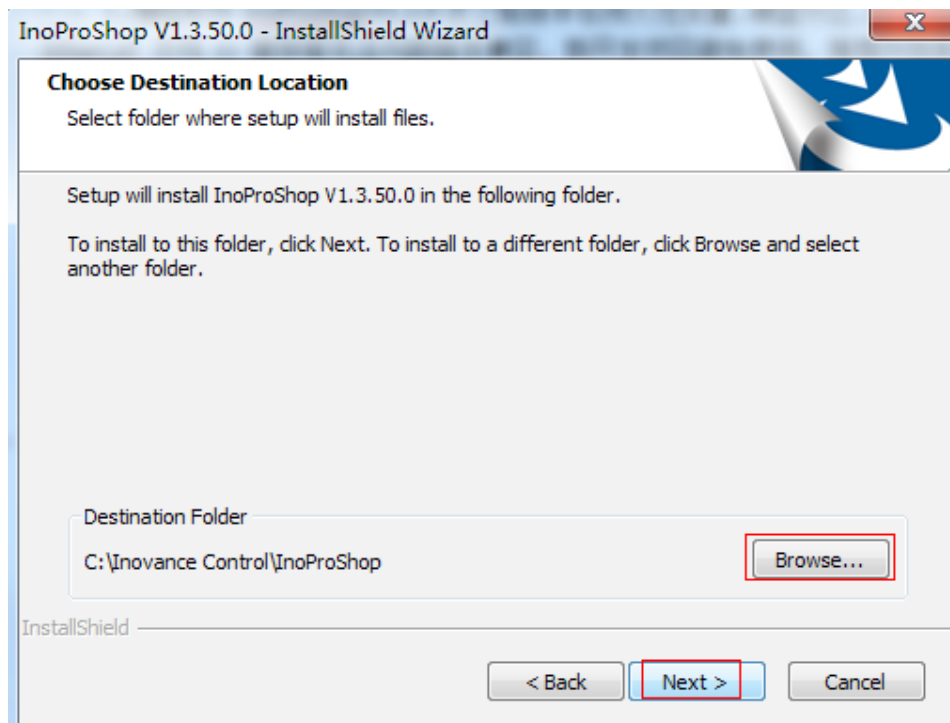
Select the features to be installed on the displayed interface and click **Next**. Keep the default selection if you do not have special requirements.
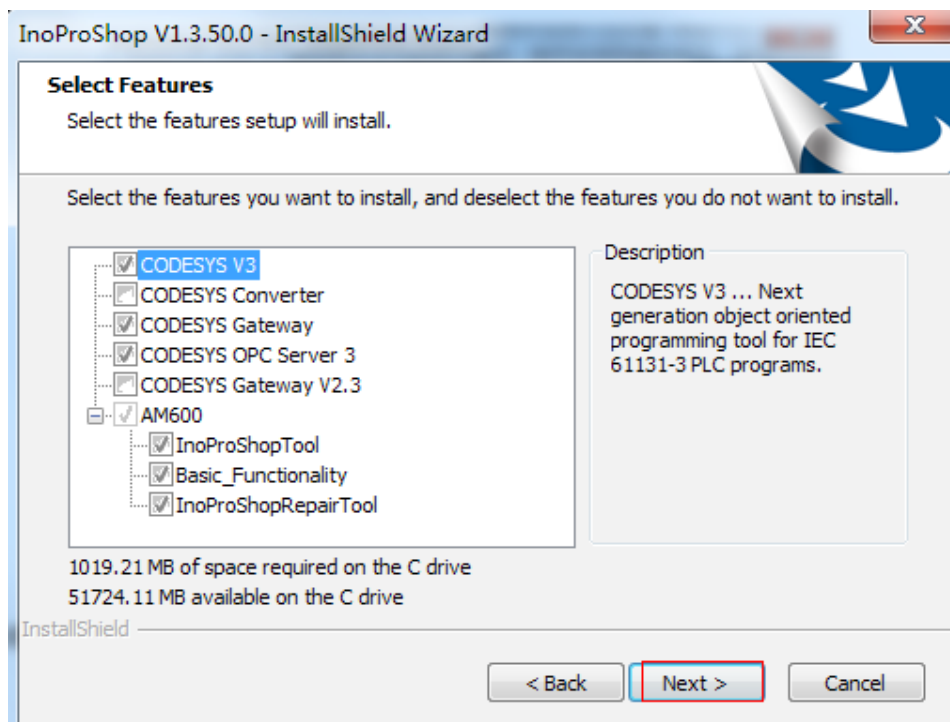
On the displayed interface, click **Next**.
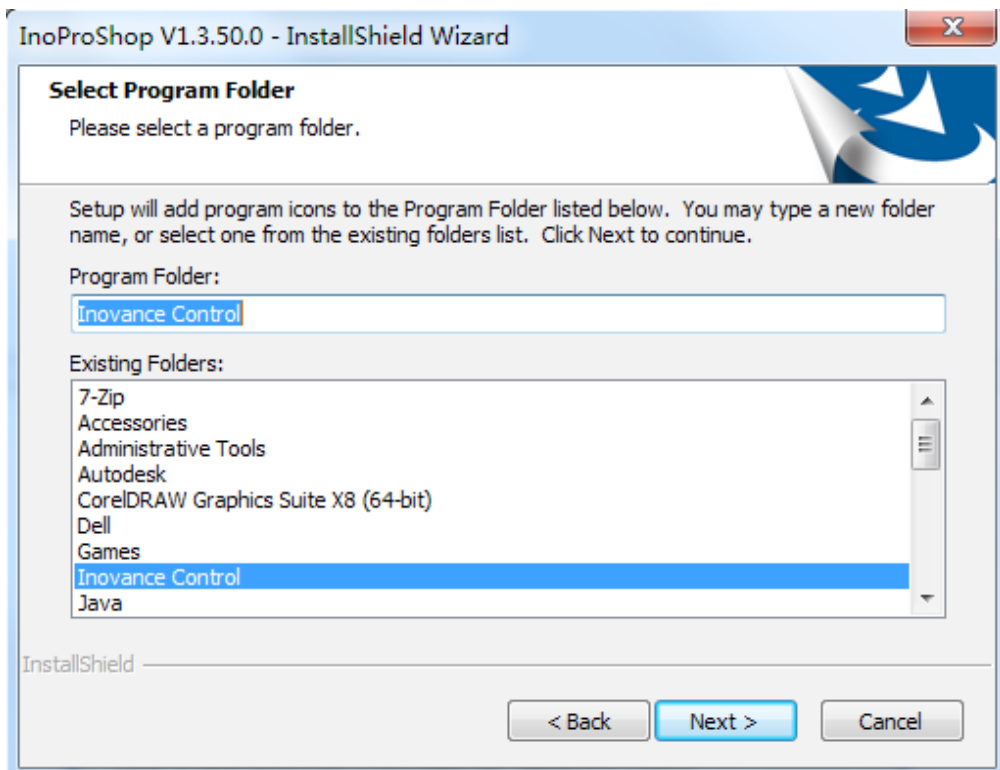
On the displayed interface, click **Next**.
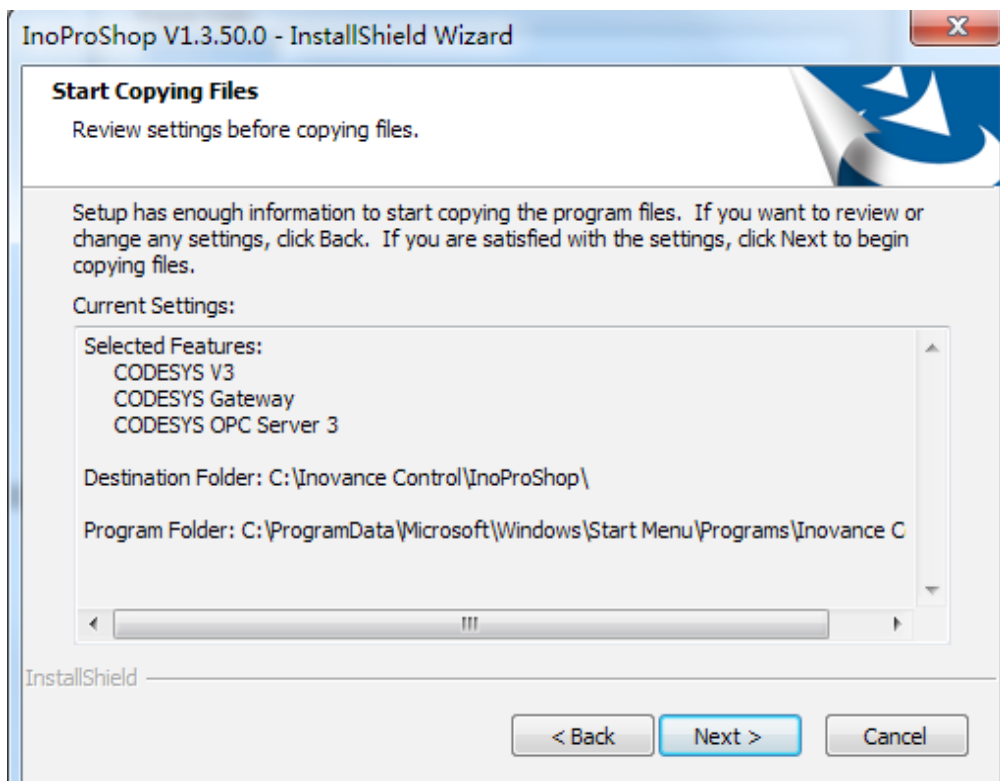
The installation progress is displayed. Wait and click **Finish** when the following interface is displayed.
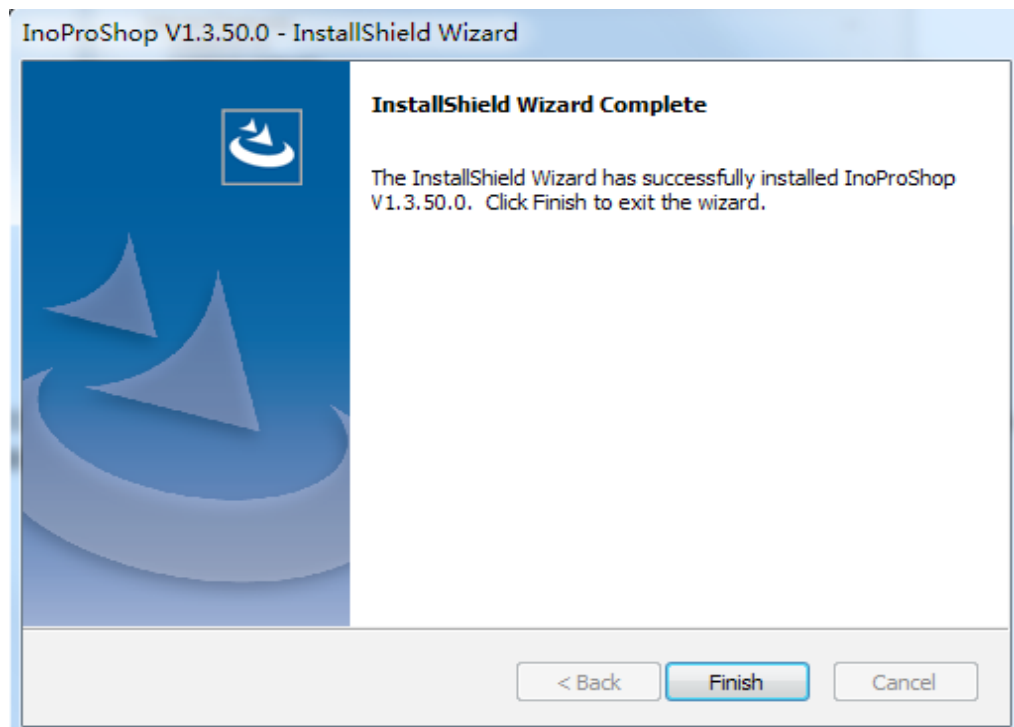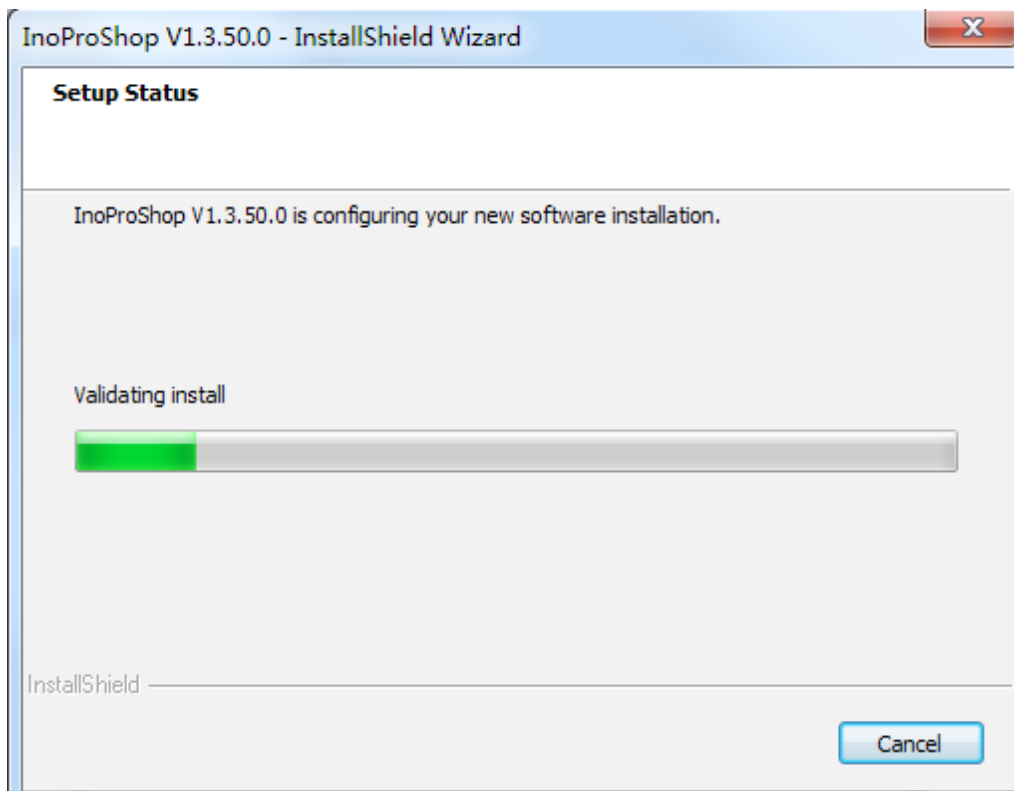
3 Language setting

Set the language you require from **Main Menu > Options > International Settings > User interface language**.
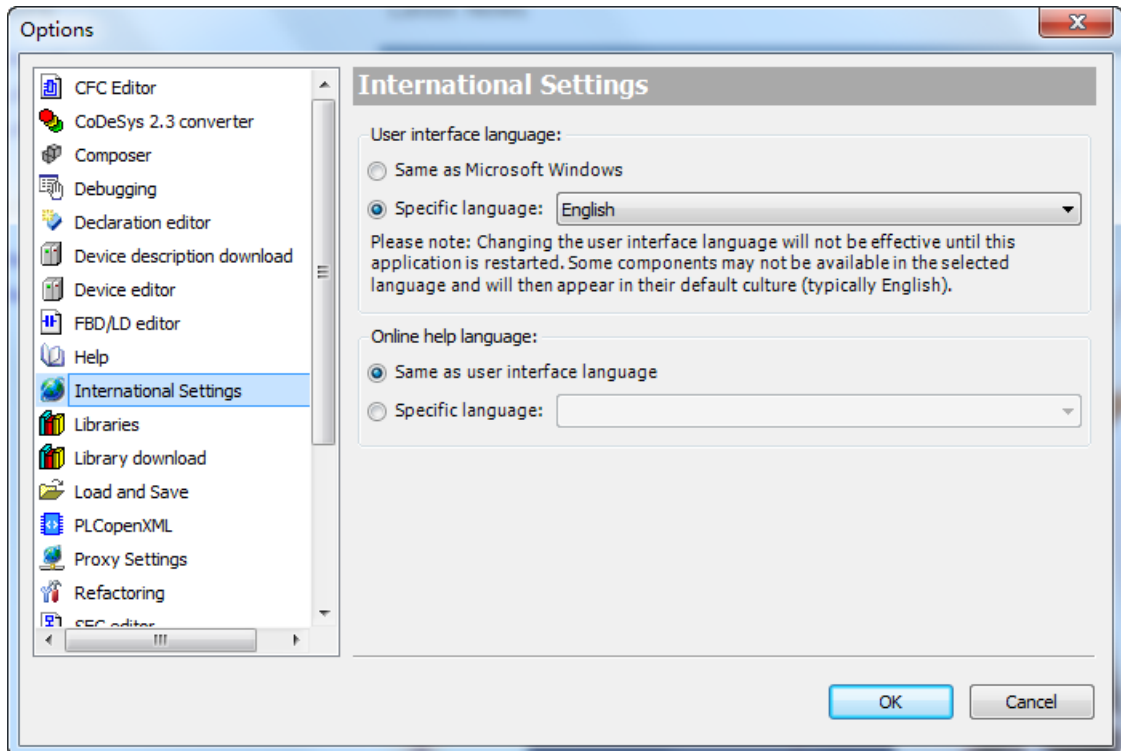
1.2.5 Uninstallation

Use the typical method of uninstalling software in a Window system to uninstall InoProShop. The procedure is as follows:

■ Exit InoProShop and ensure that the gateway is closed.

■ If the CoDeSys icon exists in the task bar of the operating system, right-click the icon and choose **Exit** to close Gateway.

■ Choose **Start** > **Control Panel**.

■ Click **Programs and Features**.

■ Select **InoProShop**.

■ Right-click, and select **Uninstall**.

Chapter 2 Quick Start

2. Quick Start

2.1 Programming Environment Launching

1) Double-click the programming software icon [icon] on the desktop to launch the InoProShop programming environment. The launch interface is as follows:
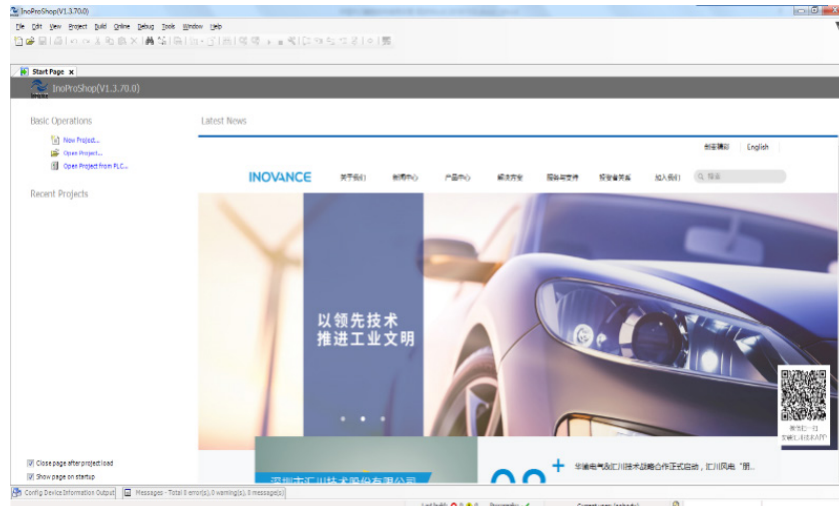
2) Click [icon] on the top left corner of the menu bar or choose **File** > **New Project** to create a project. Select the project type and specify the project file name as well as storage path, shown as follows:
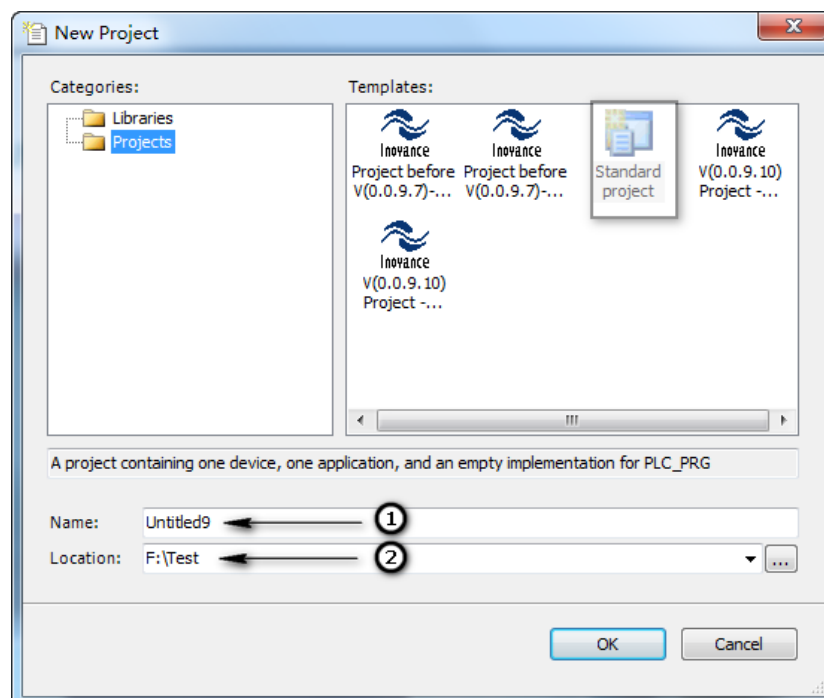
1 - Enter the project name; 2 - Select the storage path.

Click **OK** to open the standard project interface. You can select device type and programming language, shown as follows:
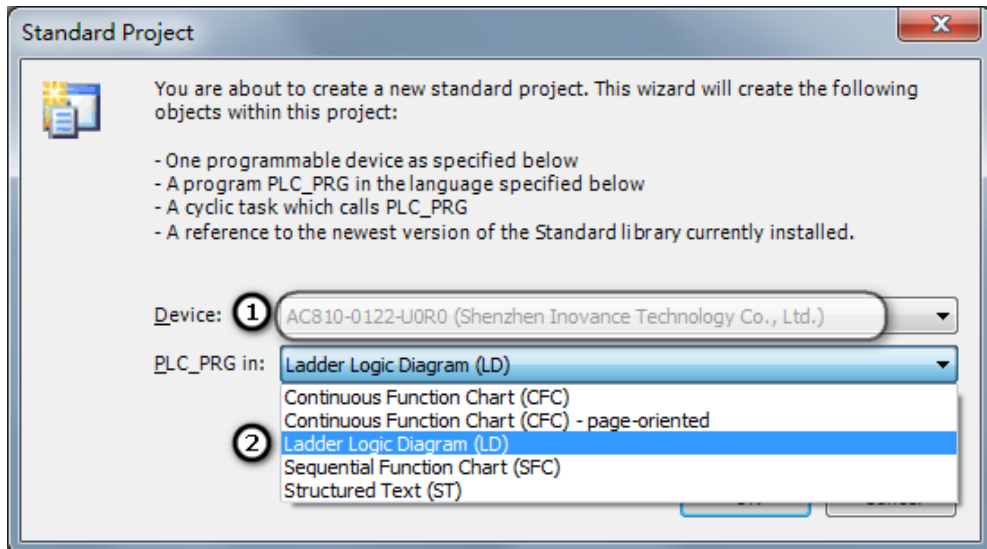
1 - Select the main module type; 2 - Select a familiar programming language.

Click **OK** to open the system configuration and programming interface. The commonly used buttons and window layout are as follows:
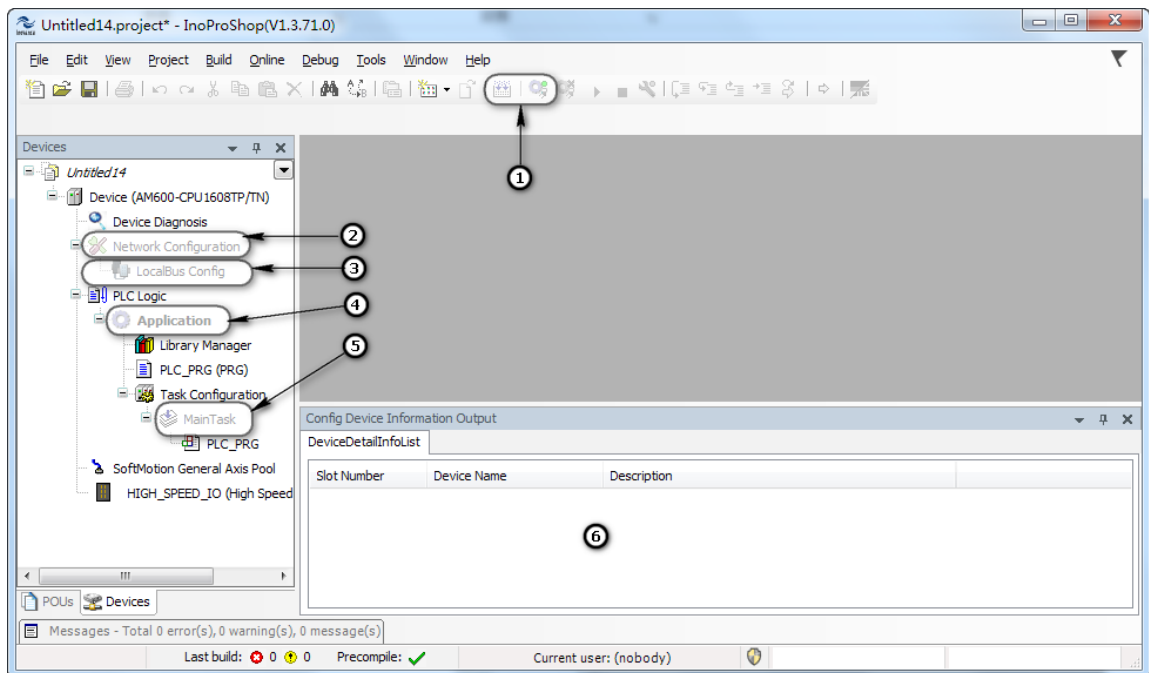
1 - Compiling, login, and commissioning; 2 - Network configuration and hardware configuration; 3 - Port and program variable linkage; 4 - Add user program unit; 5 - Configure task execution mode and period; 6 - Device information area

2.2 Typical Procedure for Writing a User Program

If this is the first time you use Inovance medium-sized PLC, note that 5 steps are required to write and commission a complete user program.

1) Configure the hardware system based on the hardware connection structure of the medium-sized PLC application system.

   ■ If only the CPU main module and I/O expansion module are used, you only need to configure the hardware: Place the "elements" of selected module type and model and installation order into the "rack" on the InoProShop hardware configuration interface.

   ■ If the expansion racks are used, configure the bus first. Then, add a certain number of network expansion modules according to the number of expansion racks, and add expansion modules into each rack.

2) Write the user program according to the control procedure of the application system. During programming, the variables are customized based on the data storage width and use scope, which may be independent of hardware configuration.

3) Link the input port variable (I), output status (Q), or value (M) of each hardware port in the system structure with the variables in the user program.

4) Configure the synchronization period of network communication (for example, the EtherCAT bus). Configure the execution periods of user program units according to the instantaneity requirements of tasks.

5) Log in to the medium-sized PLC in the InoProShop programming environment. Download the user program, carry out simulated commissioning, and rectify faults until the program runs normally.

2.2.1 User System Configuration Operations

On the InoProShop main interface, double-click **LocalBus Config** in the left device tree to open the PLC rack hardware configuration interface:
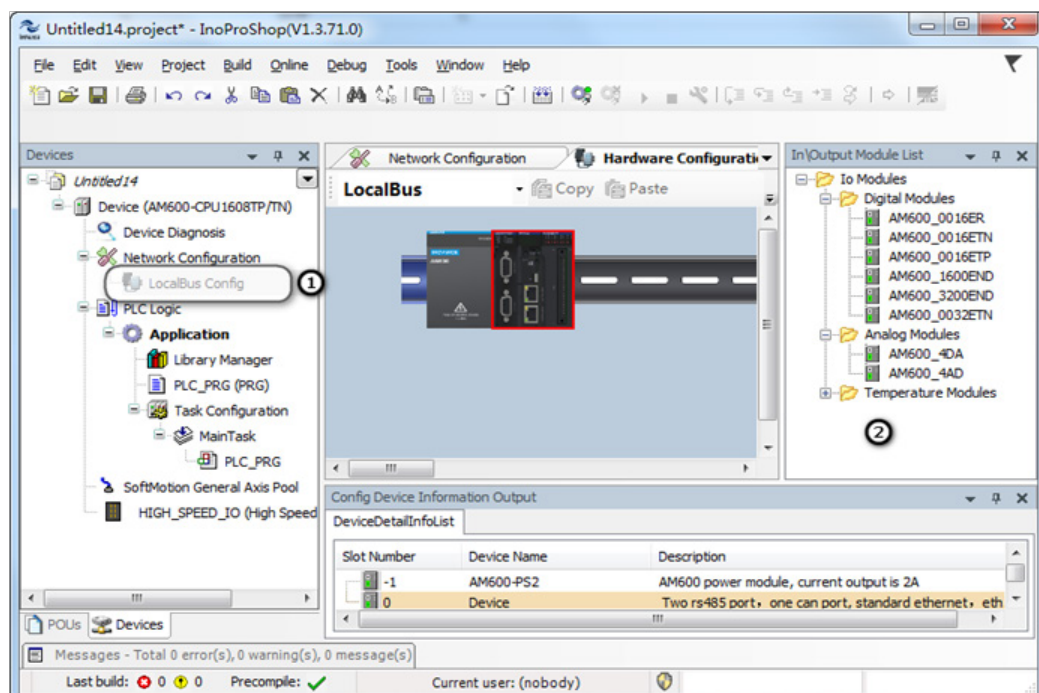
1 - Double-click to open the local expansion module configuration interface; 2 - Element library of the expansion module.

Double-click the required the modules in turn in the right expansion module library according to the required module models and installation order, and drag the modules to the rack. To delete a module, select it and press Del.

Take AM600 for example. A maximum of 16 expansion modules can be mounted to the rack, including 8 analog modules.
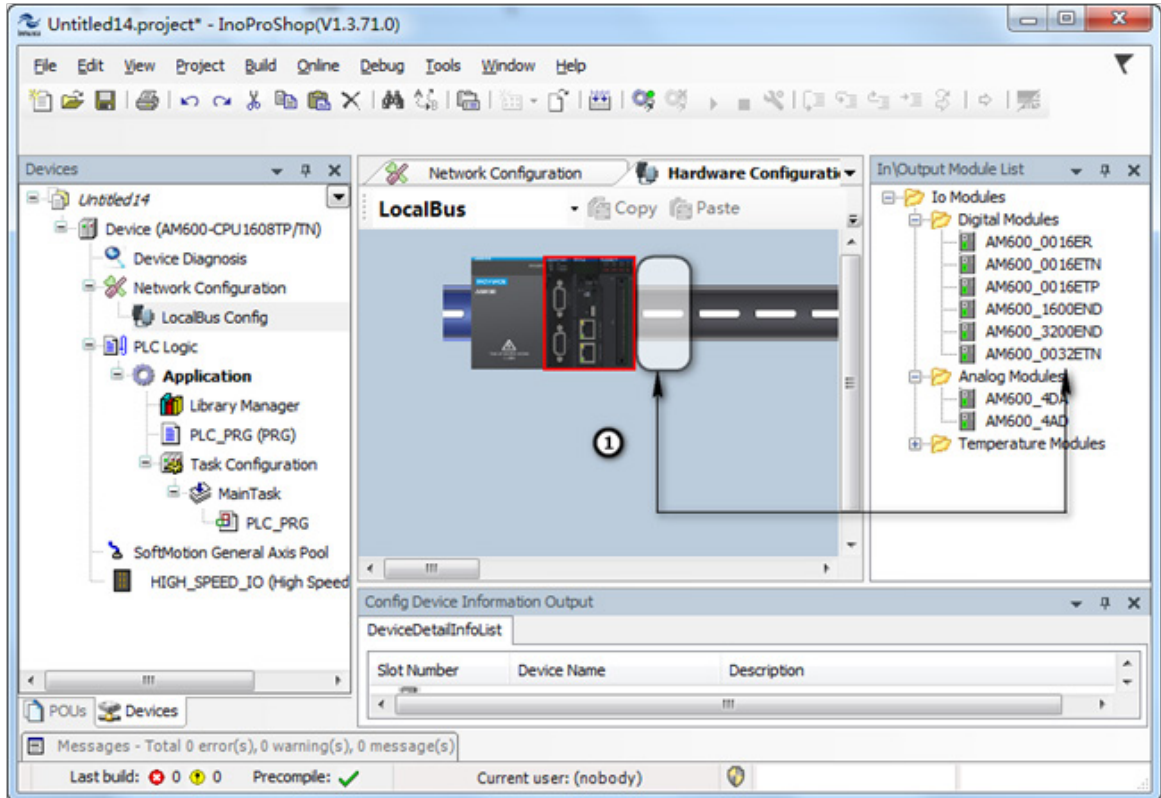
1. Click on the right side of the CPU unit in installation slot. Double-click the required I/O module in the expansion module library window. Place the modules in turn.

2.2.2 User Program Writing Operations

Double-click **PLC_PRG(PRG)** in the left device tree to open the user programming interface. The programming language is ST (selected in projection creation), shown as follows: Similar to C language, every variable can be used only after declaration. After a program statement is written, when you press **Enter**, the programming environment automatically displays a declaration prompt. When you press **OK**, the variable declaration window automatically adds this statement. This simplifies the program procedure.
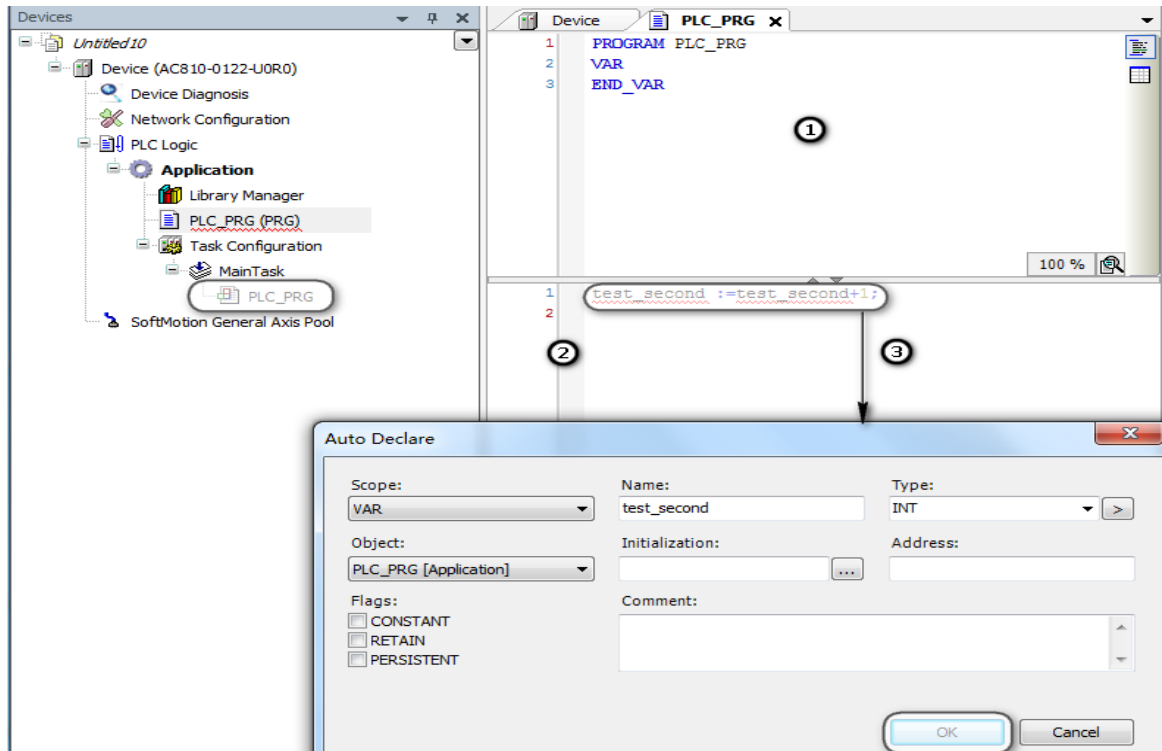
1- Variable declaration window; 2 - Programming window; 3 - Programming with ST language, and prompt for confirming variable type after pressing **Enter.**

A programming example: Assign the value of the second variable to the first variable and progressively increase the value.
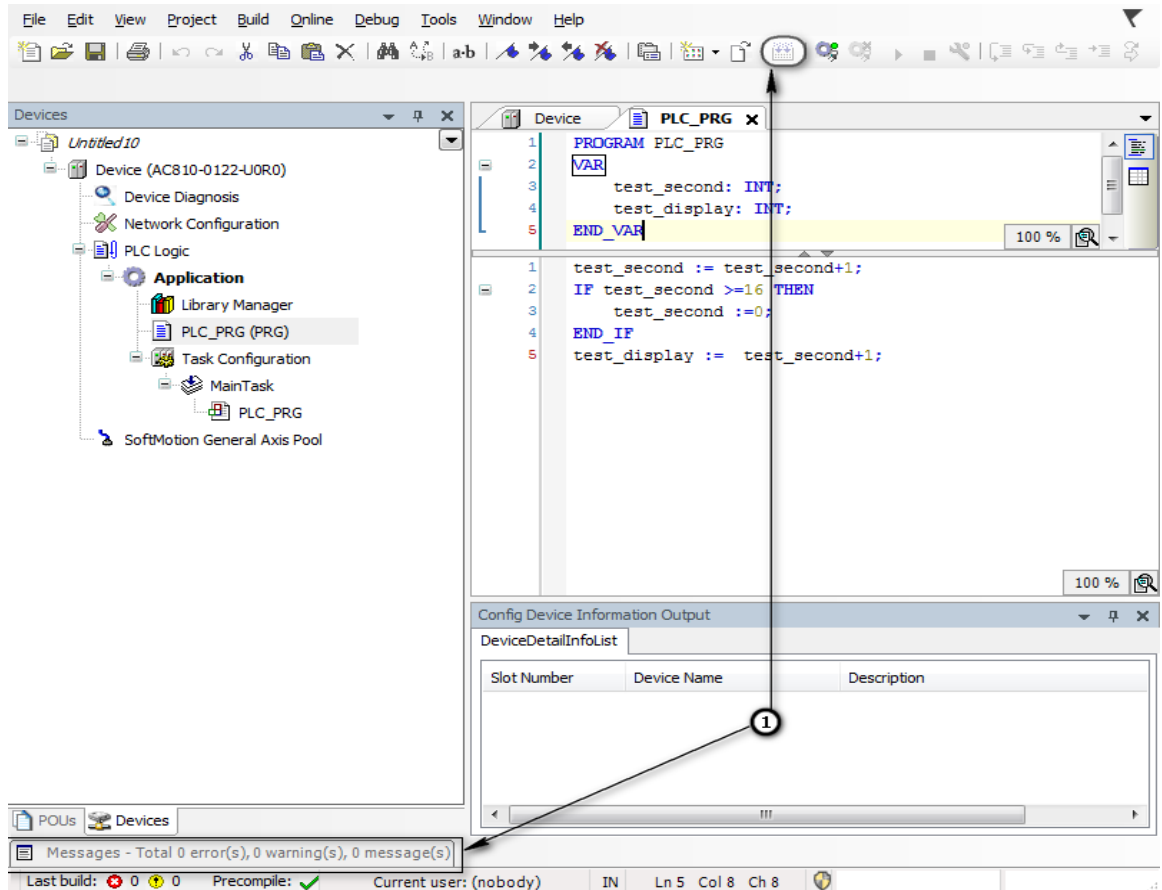
1 - Examine whether compiling is correct in the information output window.

2.2.3 Linkage Configuration Between User Program Variables and Ports

On the local bus configuration interface, link the hardware ports with variables in the user program. As shown in the following figure, link the test_display variable with the output port of the first DO module. The configuration procedure is as follows:
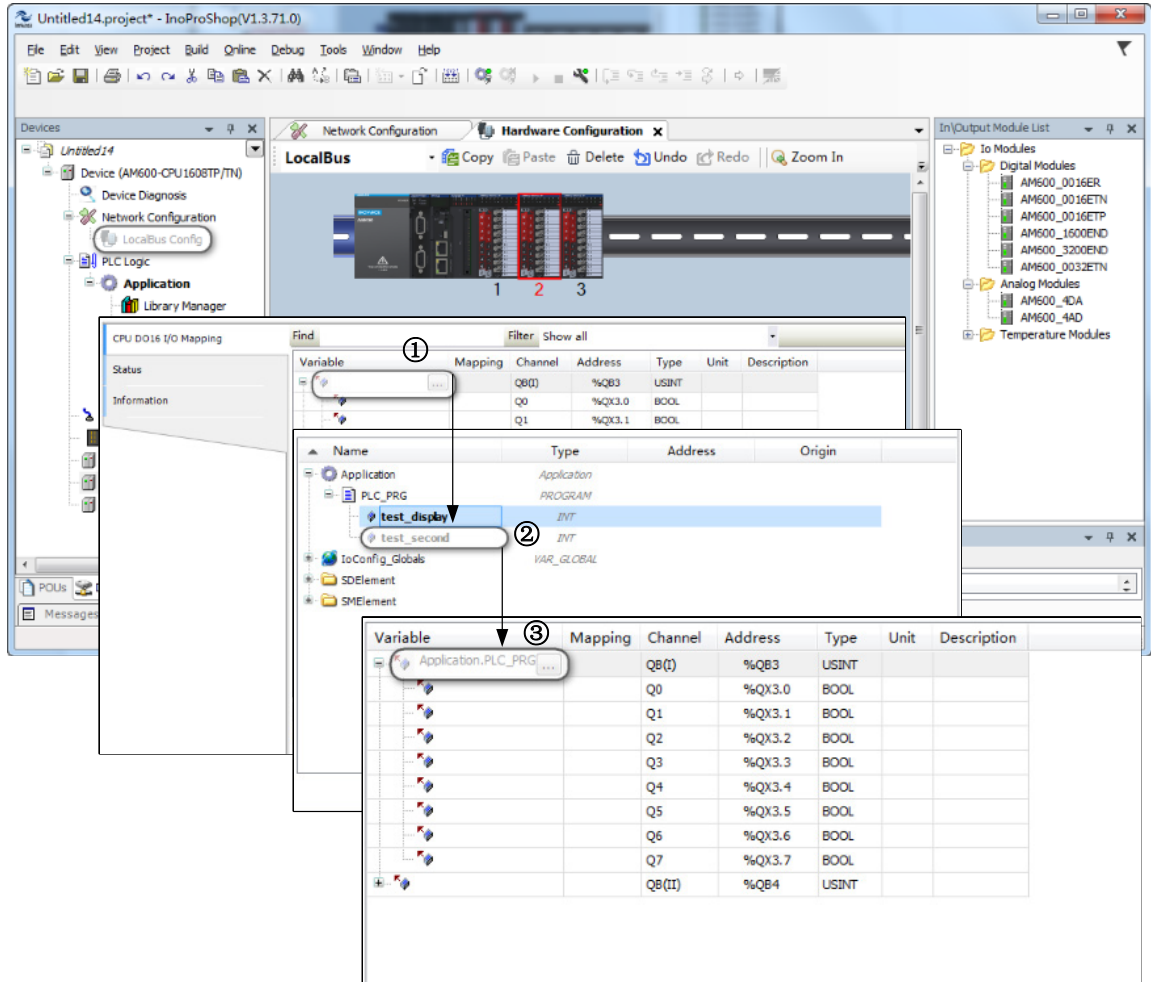
2.2.4 Configuring the Execution Mode and Running Period of User Program

In the previous example, the sub-program is executed every 20 ms by default. If you need to change the execution mode, for example, change it to repeated execution, scheduled execution, or execution at a specified interval, perform the following operations:
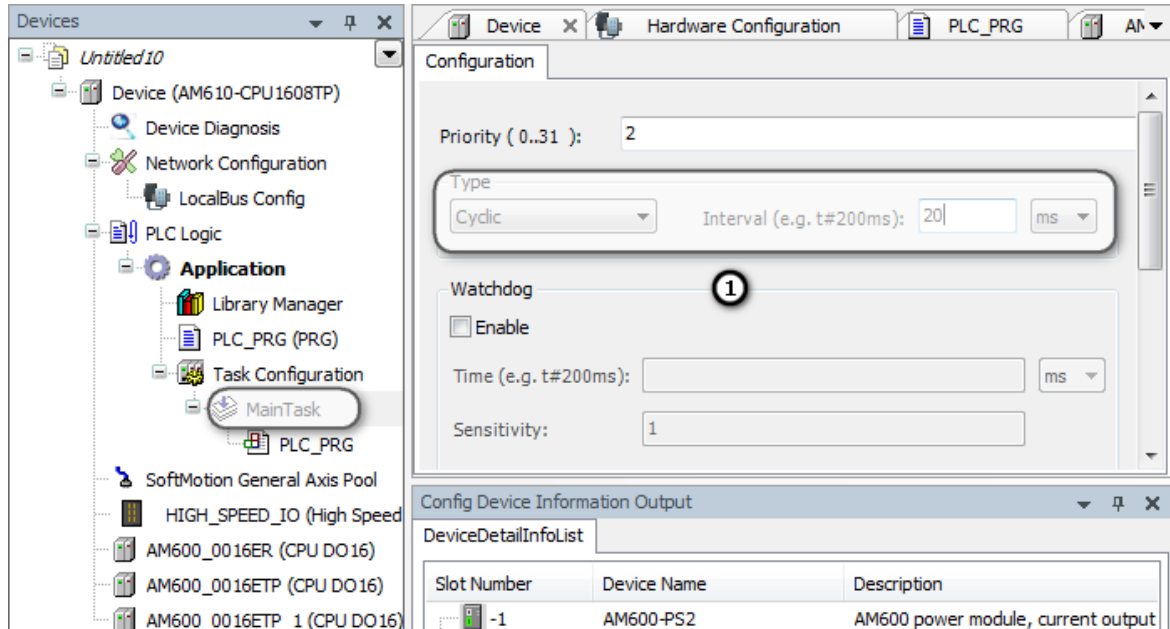
1 - Execution mode, scanning interval

2.2.5 User Program Compiling and Login Download

After programming, the program needs to be complied. Verify the compiling operation and locate errors based on the compiling information, and then repeat the operation until there is no error.

The compiling information is displayed in the following compiling information box:
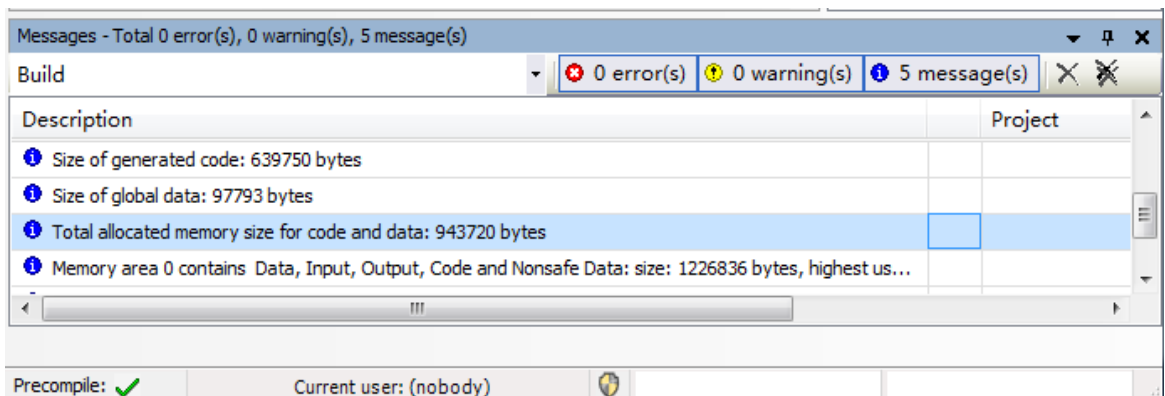
After confirming that no compiling error exists, click **Online** and the login button, shown as follows:
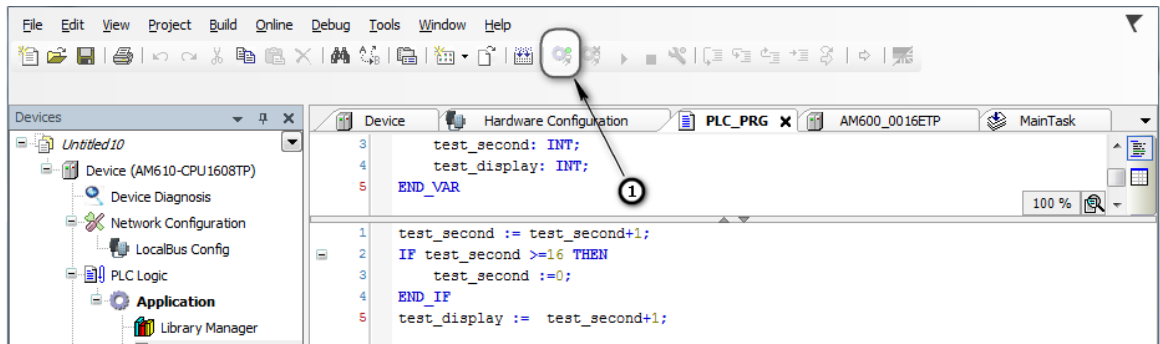
1 - Click the login button to log in to the AM600 to download and debug the programs

The following dialog box is displayed. Choose whether to create a project and continue download:
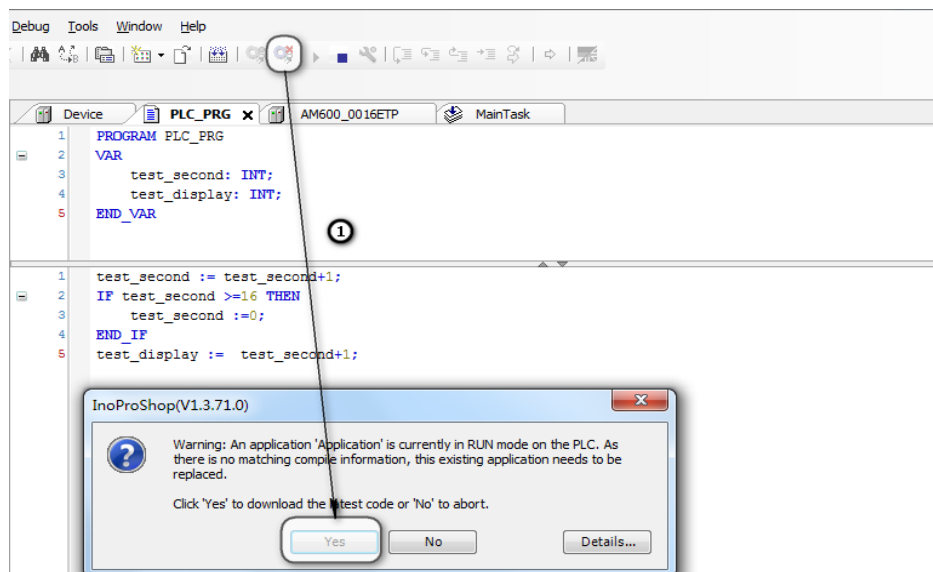
1 - After you click the login button, InoProShop displays a prompt to avoid misoperation.

Click **Yes** to connect the host computer to the device and retain the connection. The initial status is **Stop**, shown as follows:
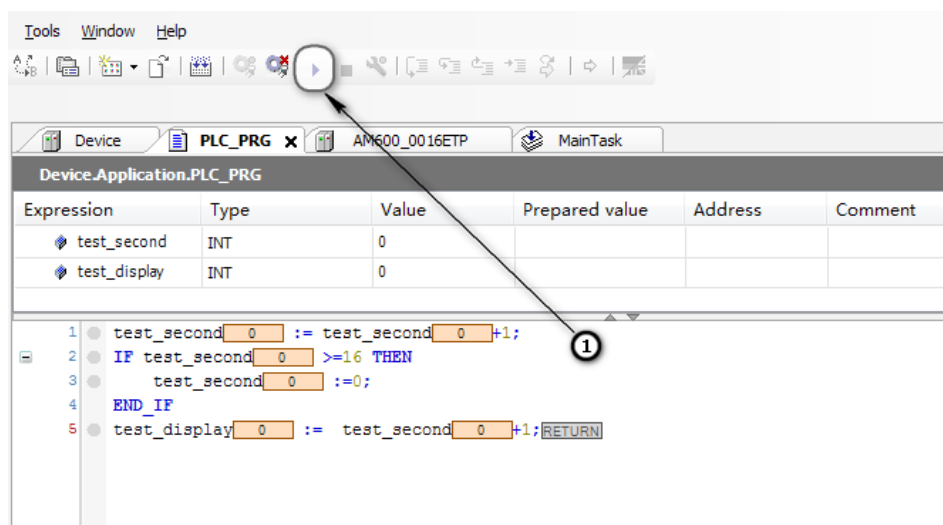
1 - Click the run icon to start the running of user program.

Choose **Debug** > **Start**. The device enters the running state and starts to run the user program.
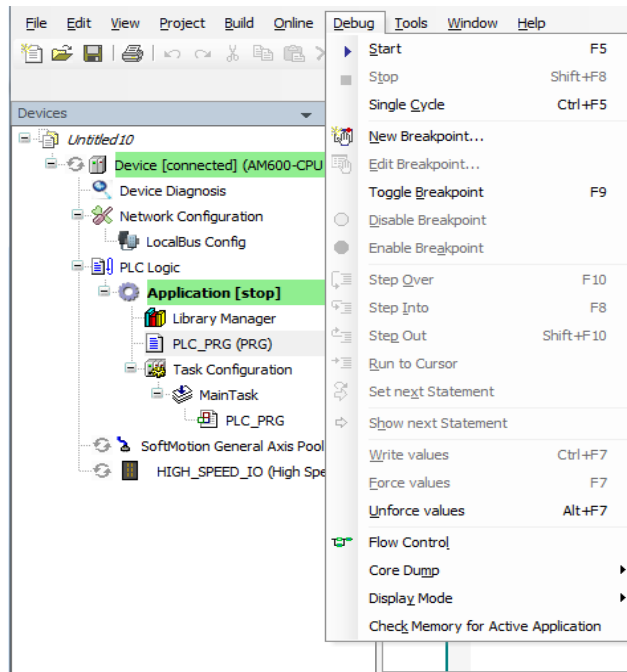
The following figure shows the monitoring interface of a running user program:
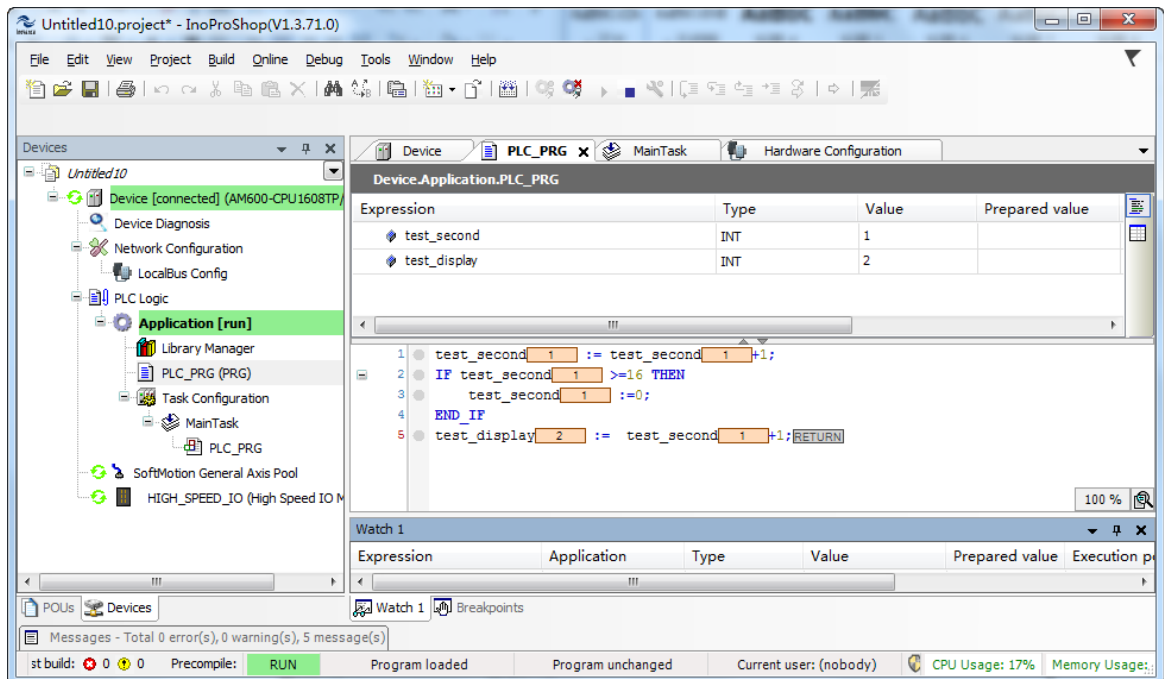
Check the first DO module behind AM600. You can see that the output status indicator cyclically counts in a binary mode.

2.3 Writing a Marquee Sample Project with InoProShop

1 Launch the InoPro Programming Environment

Create a project:

Click 📋 on the top left corner of the menu bar or choose **File** > **New Project**. Select the project type and specify the project file name as well as storage path, shown as follows:
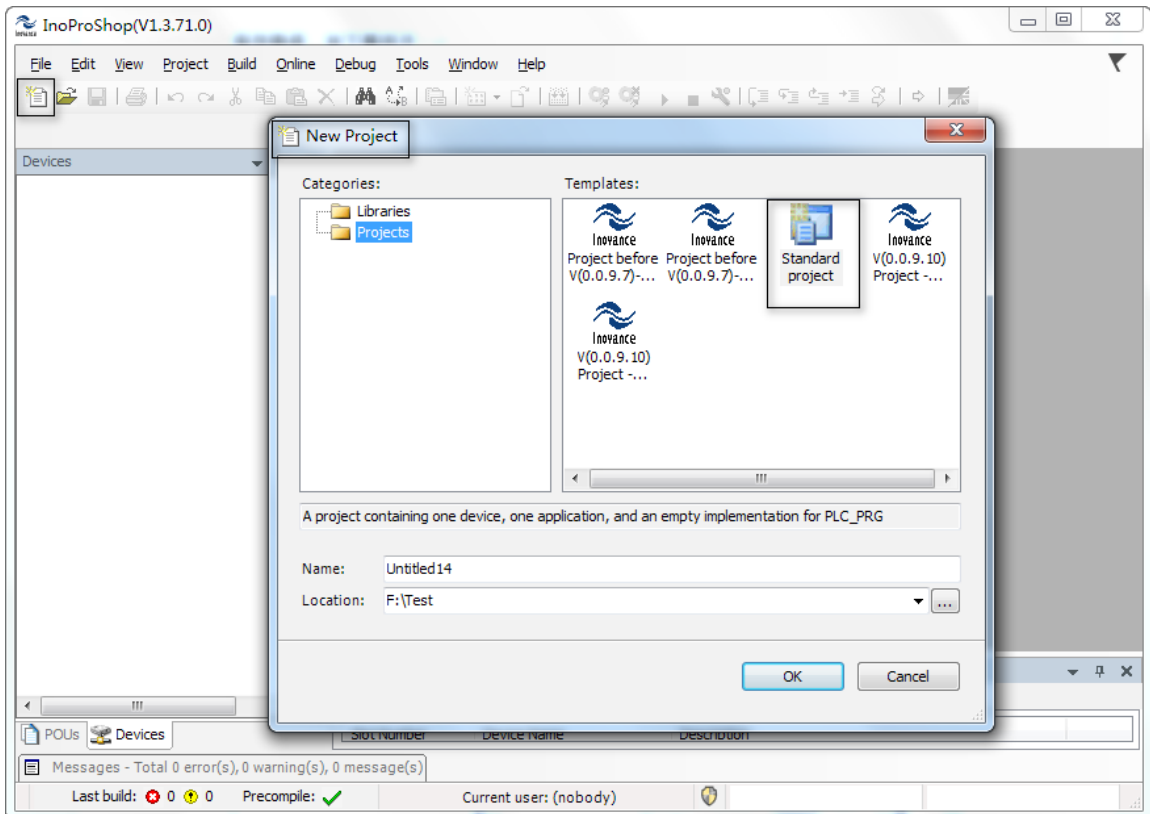
2 Select a Device Type and Programming Language

Open the standard project interface. You can select the device type and programming language, shown as follows:
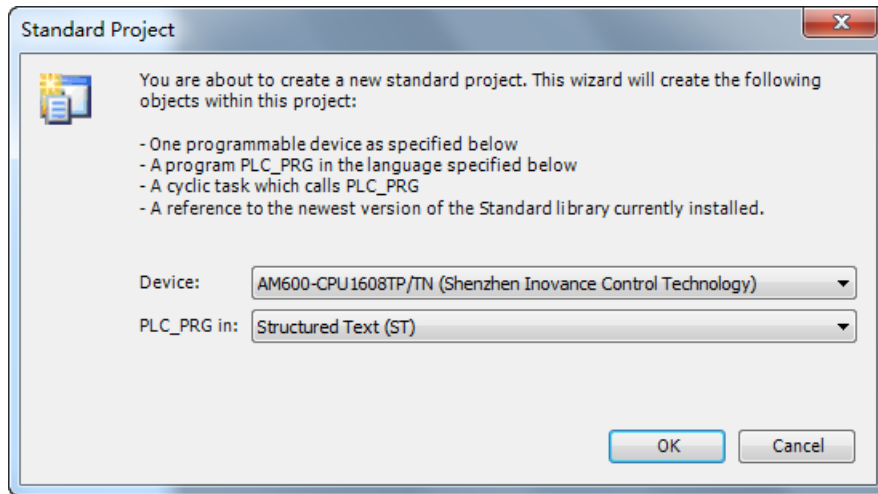
Device: Select the main module model.

PLC_PRG in: Structured Text (ST)

3 System Configuration and Programming Interface
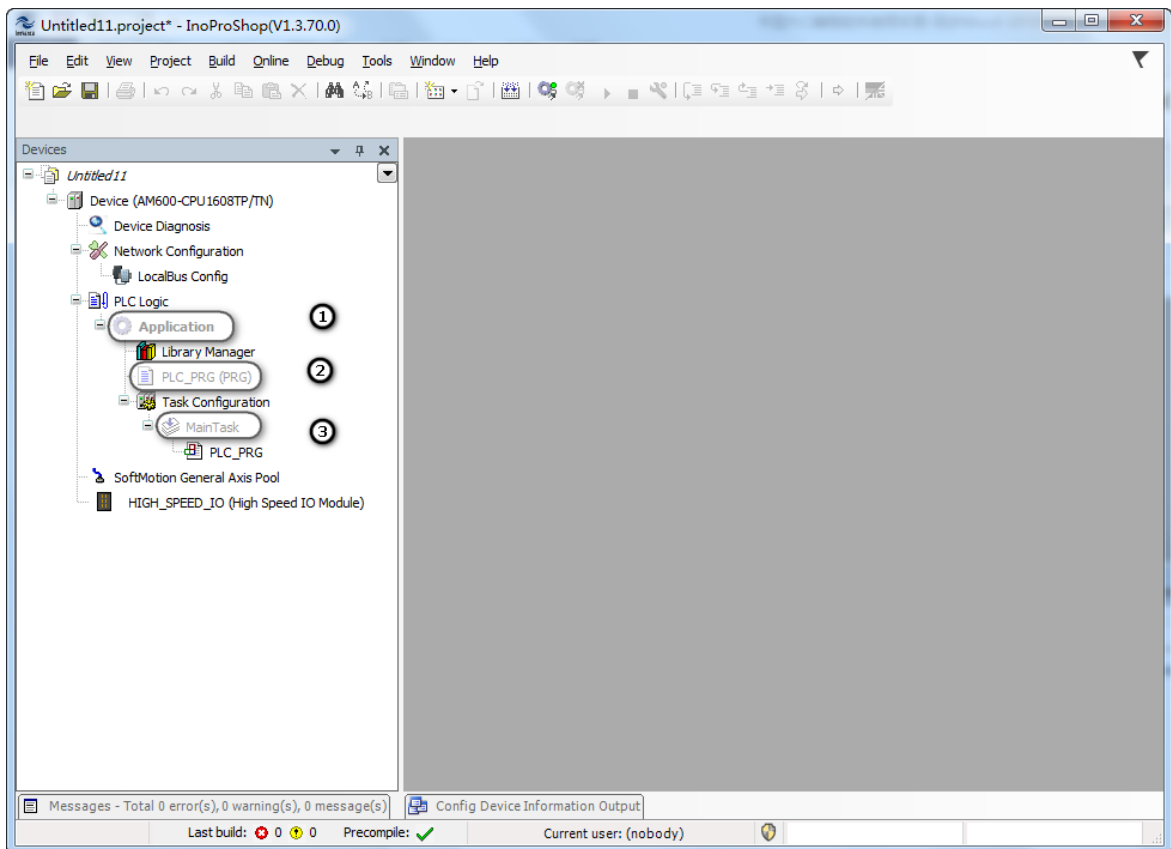
1 - Add user program unit; 2 - User program; 3 - Task configuration and program call

4 Write the marquee sample program with ST

Double-click to launch the PLC_PRG.
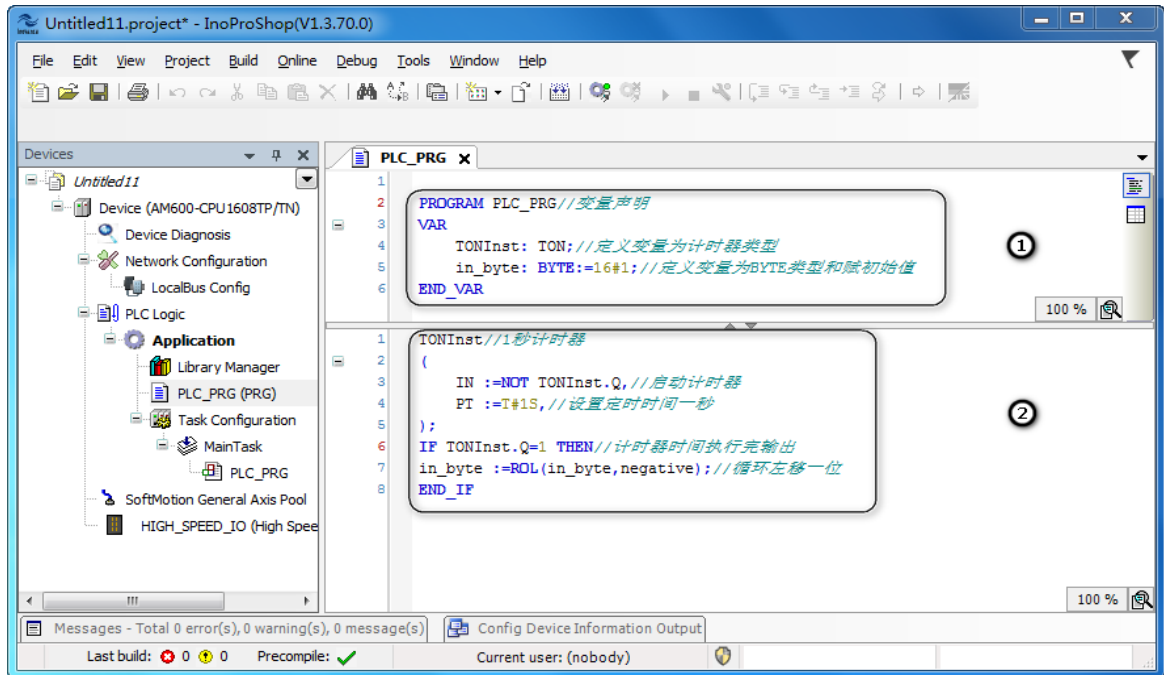
1 - Program variable declaration; 2 - Circular 1-bit left shift program with ST

5 Linked PLC Output I/O

Left shift in_byte variable and eight output port links (Bit 0-Bit 7) of PLC. Observe the output indicator status change.
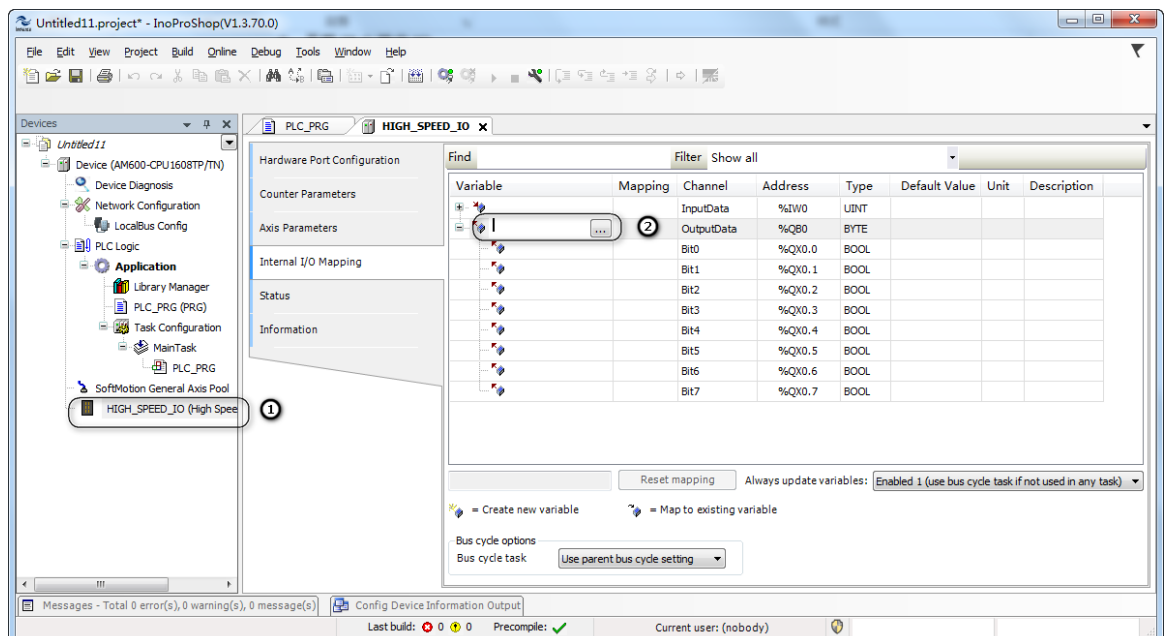
1 - High-speed I/O definition; 2 - Output variable link
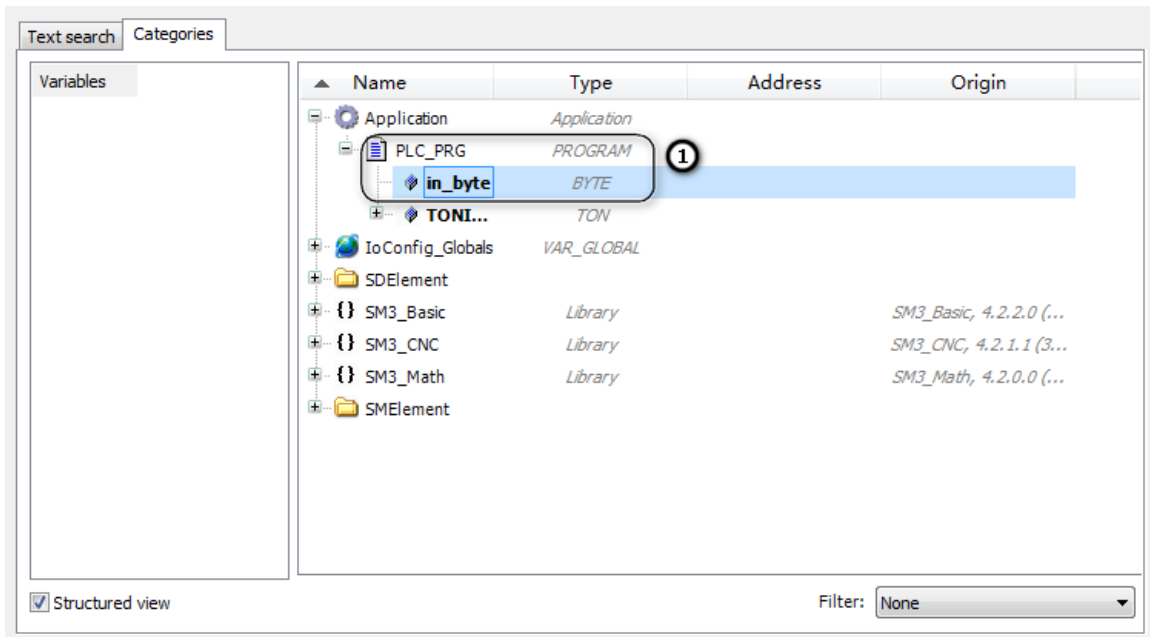
1 - Linked I/O output

6 Simulation Debugging
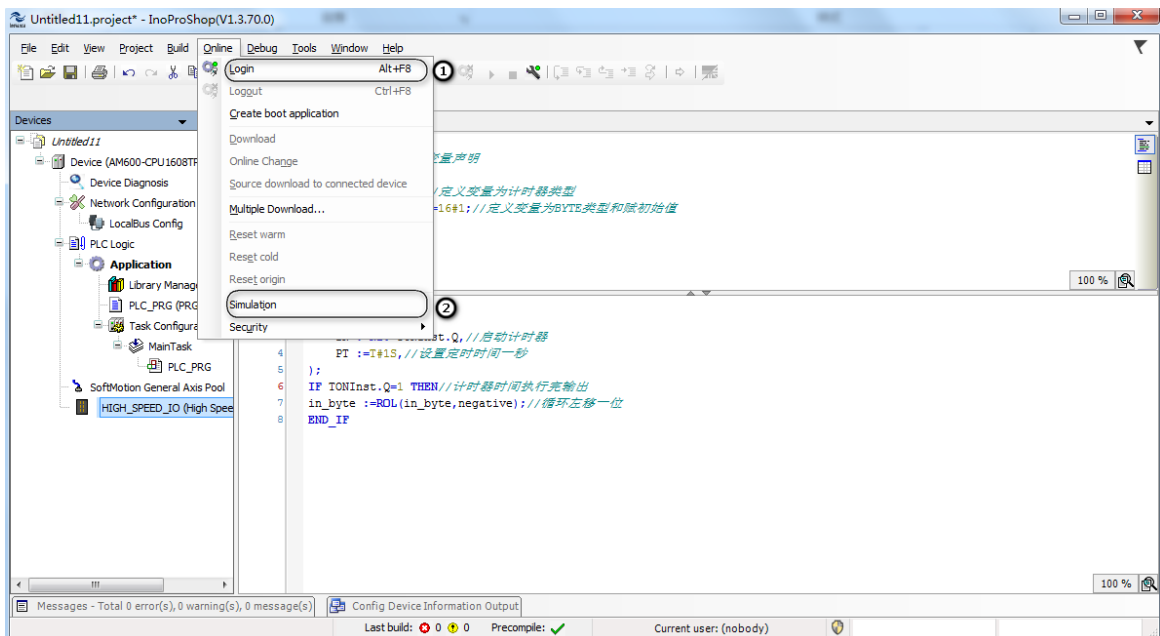
1 - Log in to the download program; 2 - Start the simulation function.

Click **Simulation** to start the simulation function. You can view the IO shifting status without linking to PLC.

7 Download Program in Simulation Mode

Click **Login** to download program in simulation mode.
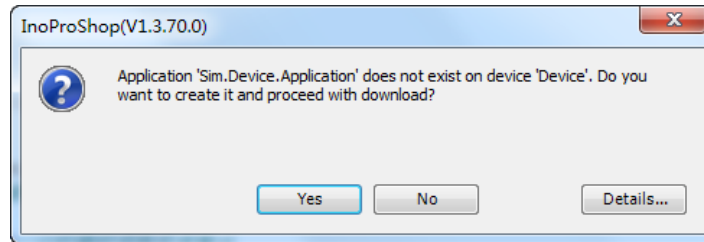
8 Run the PLC After Downloading
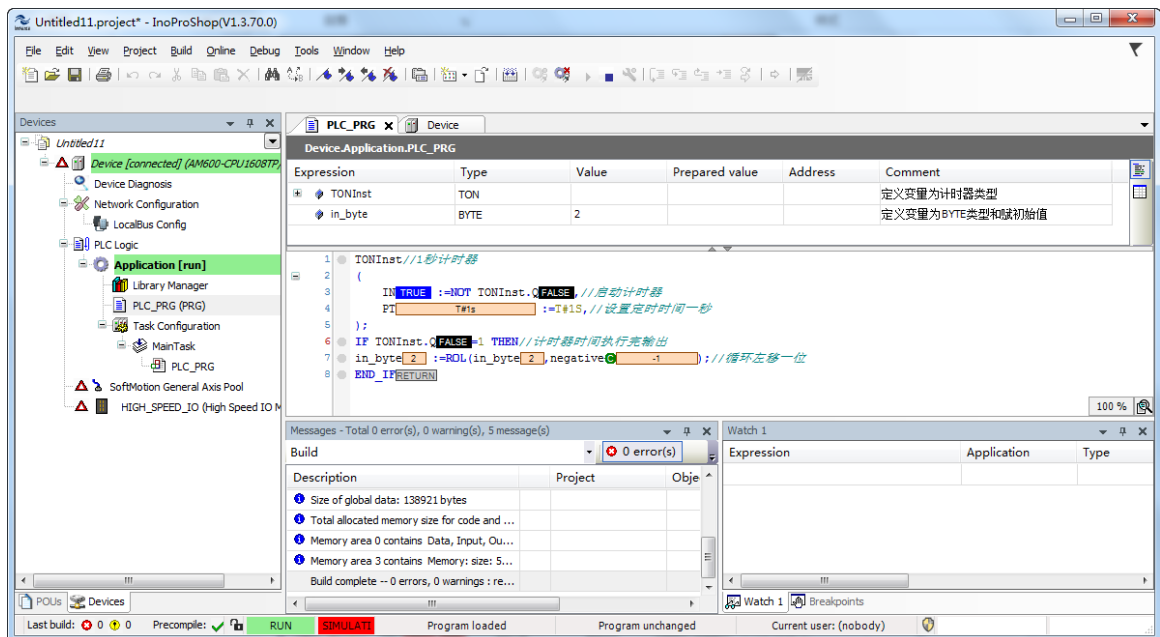
9 Monitor I/O change
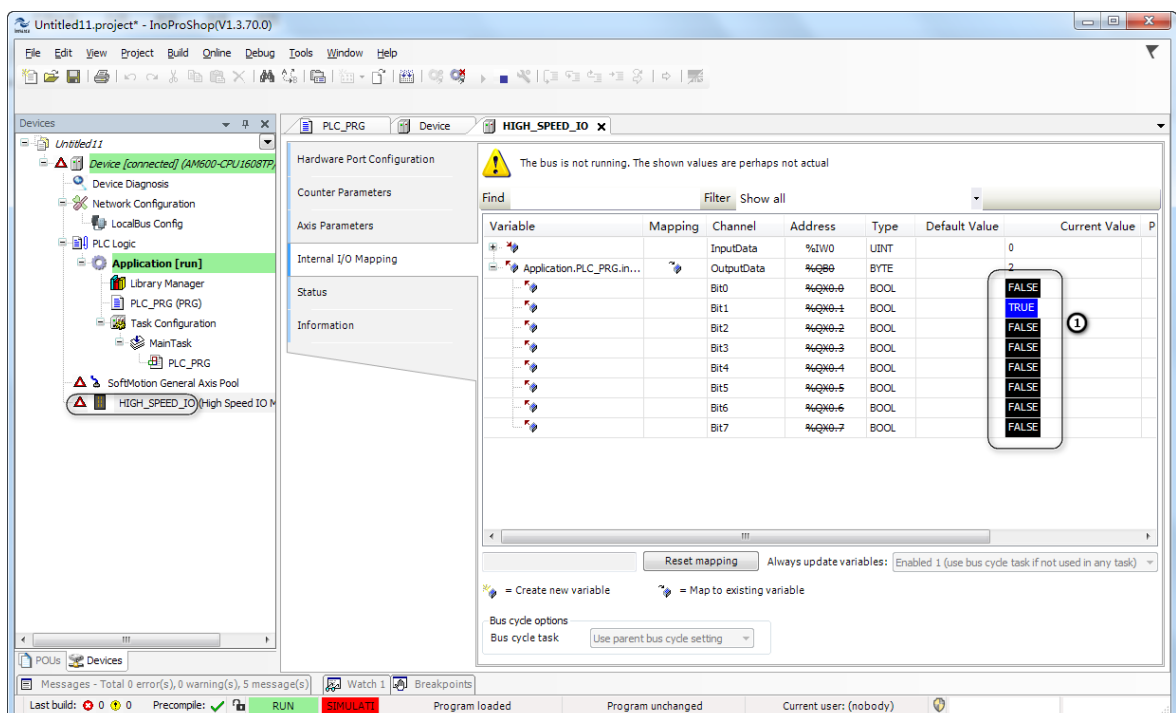
1 - Circular 1-bit shift

2.4 How to Log In to the Main Module

2.4.1 Prerequisites and Operations of Main Module Login

Main Module Login means that the InoProShop running on PC sets up communication with the medium-sized PLC main module, so that user program can be run, downloaded, started/stopped, and monitored. In addition, you can check and modify program parameters.

■ You can log in to a medium-sized PLC: through the LAN or USB.

■ The PC can be connected to the medium-sized PLC through a network cable in peer-to-peer mode, or connected to multiple medium-sized PLCs through router or hub. Multiple PCs can also access the same medium-sized PLC.

■ A PC can log in to the medium-sized PLC only when their IP addresses are in the same network segment; otherwise, the InoProShop cannot detect the medium-sized PLC. For example, the default IP address of AM600 is 192.168.1.88. If a PC's IP address is 192.168.1.xxx (xxx ranges from 1 to 254, but is different from that in the IP address of AM600), the InoProShop can detect the AM600 and exchange data with it. Then, you can download and monitor the user program. If the IP address of AM600 has been changed to another network segment, the PC and AM600 cannot set up communication. In this situation, restore the default IP address 192.168.1.88 of AM600, and change the PC's IP address to 192.168.1.xxx. When a peer-to-peer connection is set up, change the IP address of AM600 to the desired one.

■ To log in to the PLC through USB, connect the MiniUSB port. Wait for 20s to 60s until the device can be detected.

Precautions of USB connection:

1) The USB drive is automatically installed during software installation. If not, you can find the file named "<PLC series> user drive" in the **Common** folder under the installation directory.

   Then update the drive in the Windows **Device Manager**. The drive is installed from the installation directory. After the USB connection is successfully set up, the Windows **Device Manager** displays the drive program installed.

2) If both USB connection and network connection are available, the network connection is used because its network scanning speed is faster.

2.4.2 Scanning Medium-Sized PLC in InoProShop

The PC can log in to the medium-sized PLC through LAN. Taking AM600 for example, the connection is as follows:
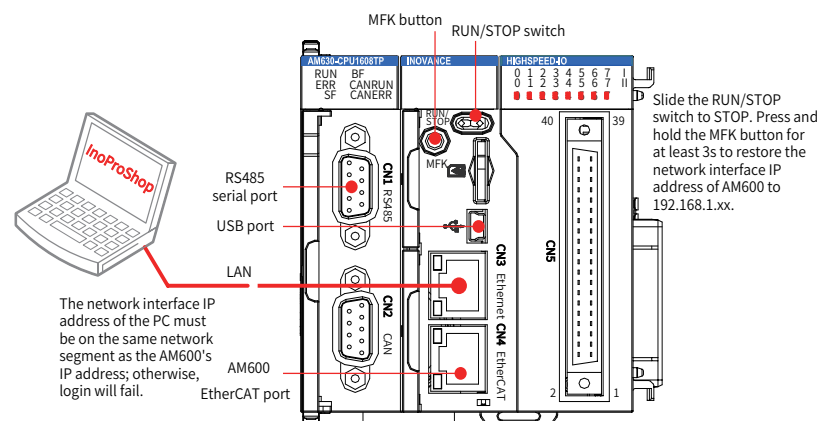
In the InoProShop, double-click Device (AM600-CPU-1608TP/TN). The following interface is displayed:
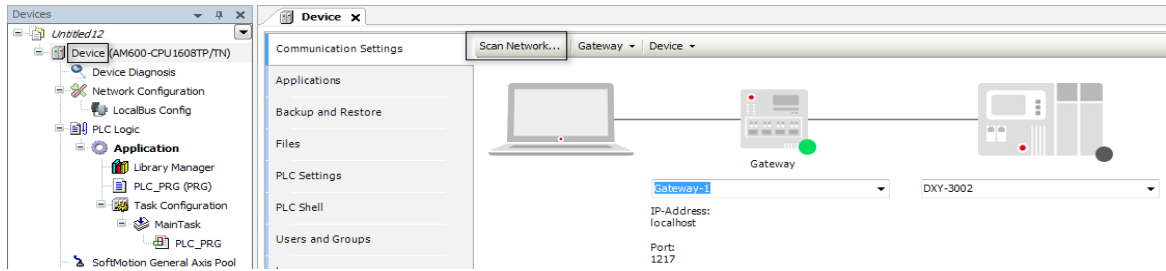
Click **Scan Network…**. The following interface is displayed. In the left part of the window, click AM600-CPU to show its introduction on the right part of the window.
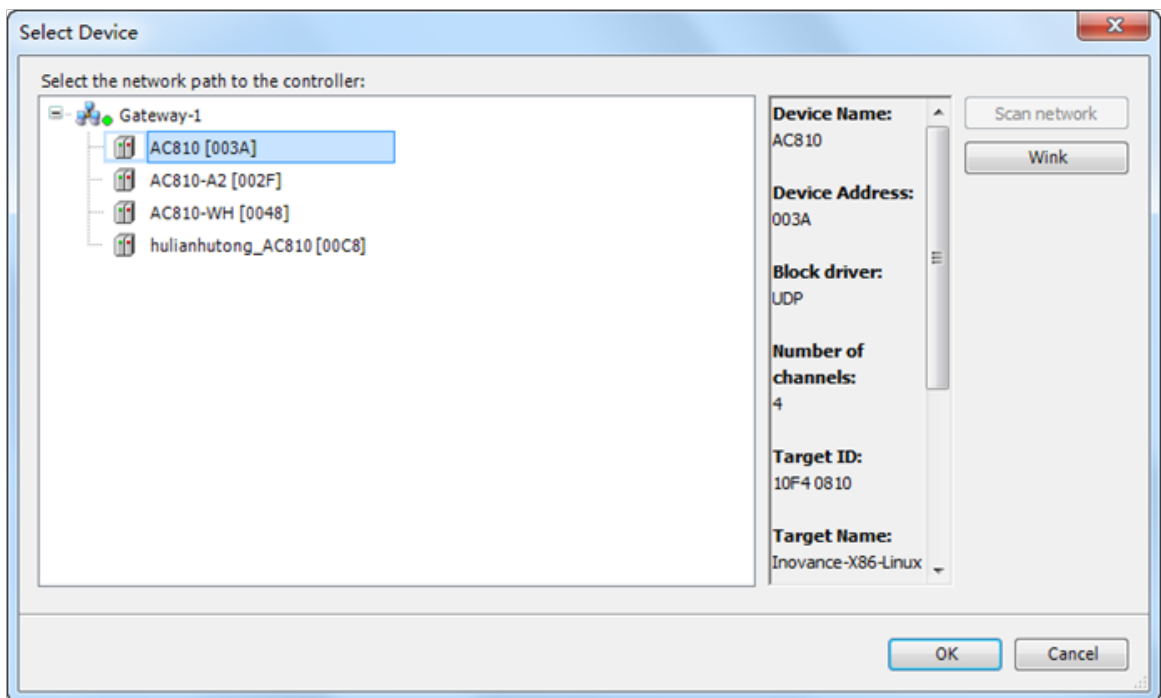
The previous figure shows four controllers, which are displayed in two rows:

The first AC810 [003A]: It is in the network segment and named AC810. The last two digits "3A" in brackets is the fourth bits in the IP address of AC810. "3A" is in hexadecimal format, and its decimal notation is 58.

The second AC810-A2 [002F]: It is another device in the network segment and named AC810-A2. After logging in, you can modify the device name so that you can easily identify the devices when there are multiple controllers.

Double-click the selected device, or select a device and click **OK**. The host computer is connected to the device.

If the controller ID recorded in the project is different from the selected controller ID, the following information may be displayed. To connect to the network, click **Yes**.
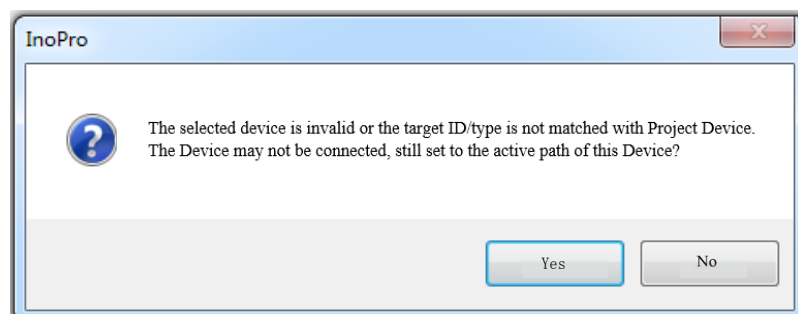
2.4.3 Solution to AM600 Scanning Failure

If the InoProShop cannot detect AM600, the possible reasons and solutions are as follows:

1)   The CoDeSys gateway is not started.

Verify that the gateway has been started and run scanning again:
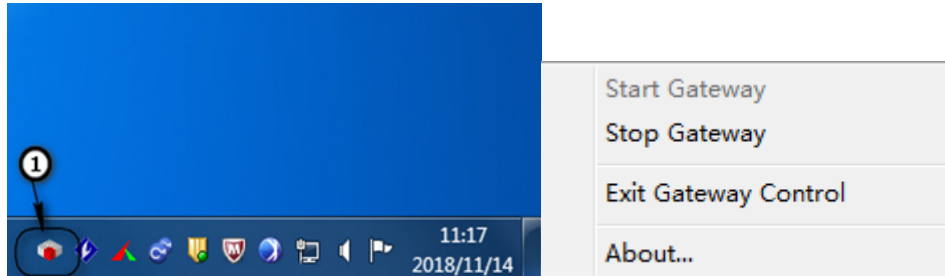
1 - Check whether the CoDeSys gateway is running in the lower right task bar on PC (colorful display). If the gateway is in stop state, click to start it.

2)   The IP addresses of PC and AM600 are in different network segments.

Solution: Check whether the PC's IP address is in network segment 192.168.1.xxx. If not, modify the IP address settings. Before modification, record the PC's IP address settings for later restoration.

■   Open the Resource Manager on the PC. Click **Local Connection** to check and modify the IP address settings:
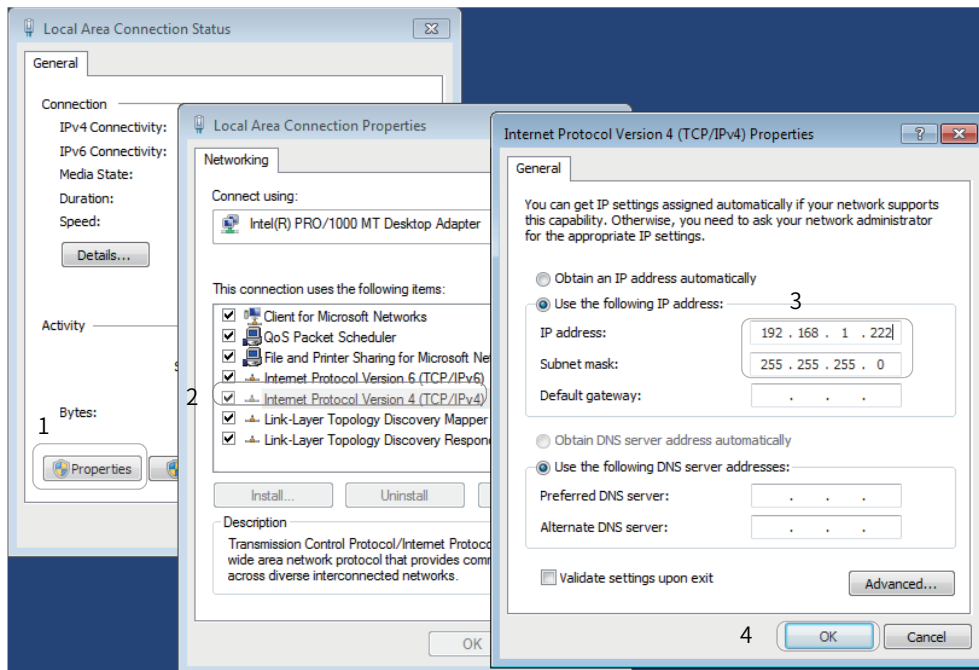
1 - Click Properties; 2 - Click Internet Protocol Version 4 (TCP/IPv4); 3 - View or change the IP address; 4 - Click OK.

■   Restore the default IP address 192.168.1.88 of AM600. After AM600 is powered on, slide the RUN/STOP switch to STOP. Hold down the MFK button for at least 3s. AM600 restores the IP address of the network interface to 192.168.1.88. Before the restoration, there are countdown reminder of such information as "I.P." and numbers "10" to "0". During the countdown, you can press MKF again to cancel the restoration.

The modified IP address takes effect immediately no matter whether you restore the default IP address of AM600 or change the IP address using the InoProShop software tool.
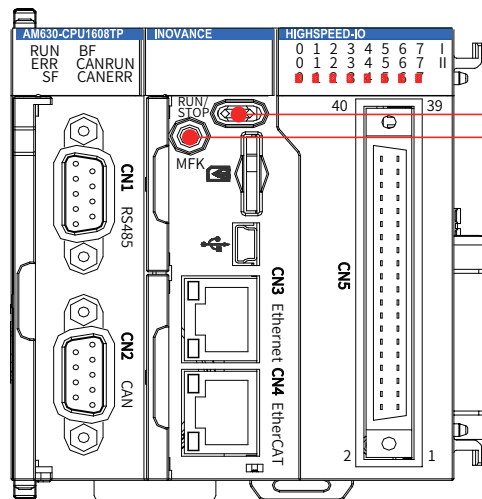
Slide the RUN/STOP switch to STOP. Press and hold the MFK button for at least 3s to restore the network interface IP address of AM600 to 192.168.1.xx.

Once the scanning and network connection are successful, the following network status information is displayed on the device scanning interface:
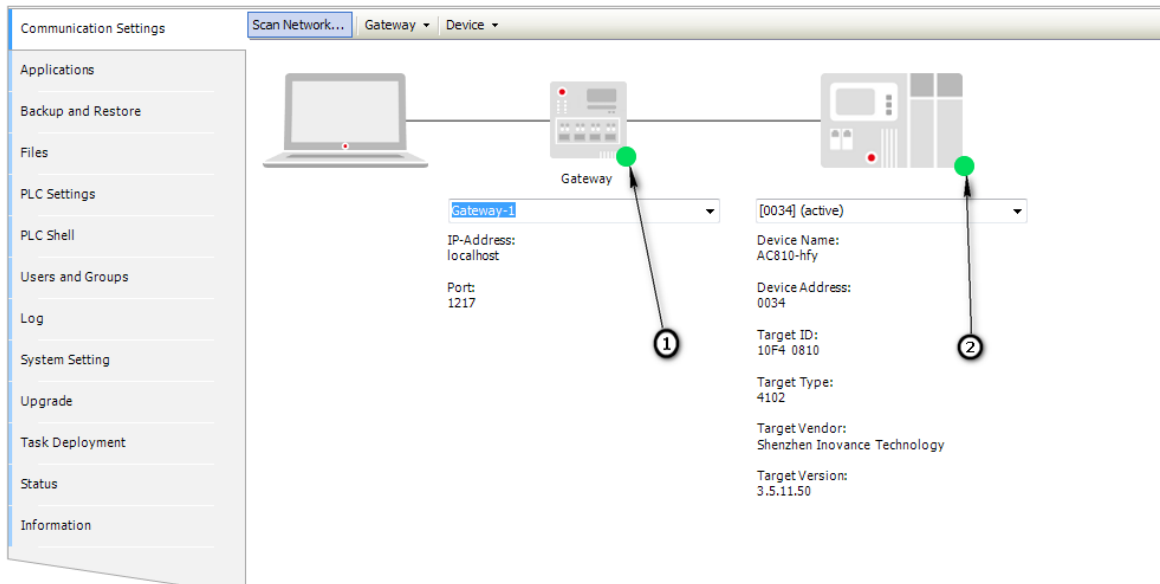
1 - Green indicates that the gateway is normally running; 2 - Green indicates that the controller has been scanned and network connection is normal.

3) When a LAN contains multiple AM600s and you have logged in to one controller, you may need to verify whether the selected controller is correct:
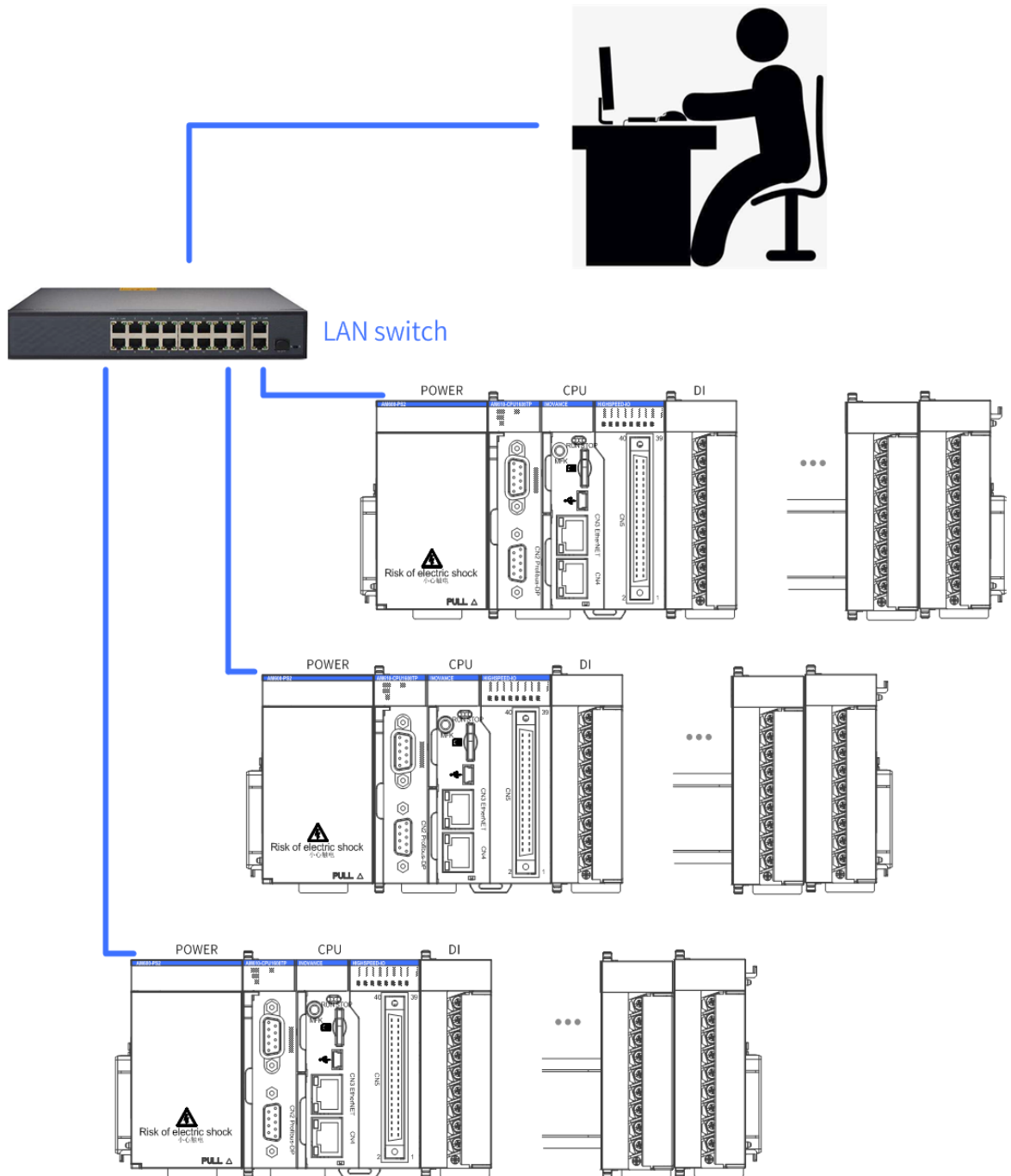
On the Device page of InoProShop, click the **System Setting** tab and click the **Identify Device** button, shown as follows:
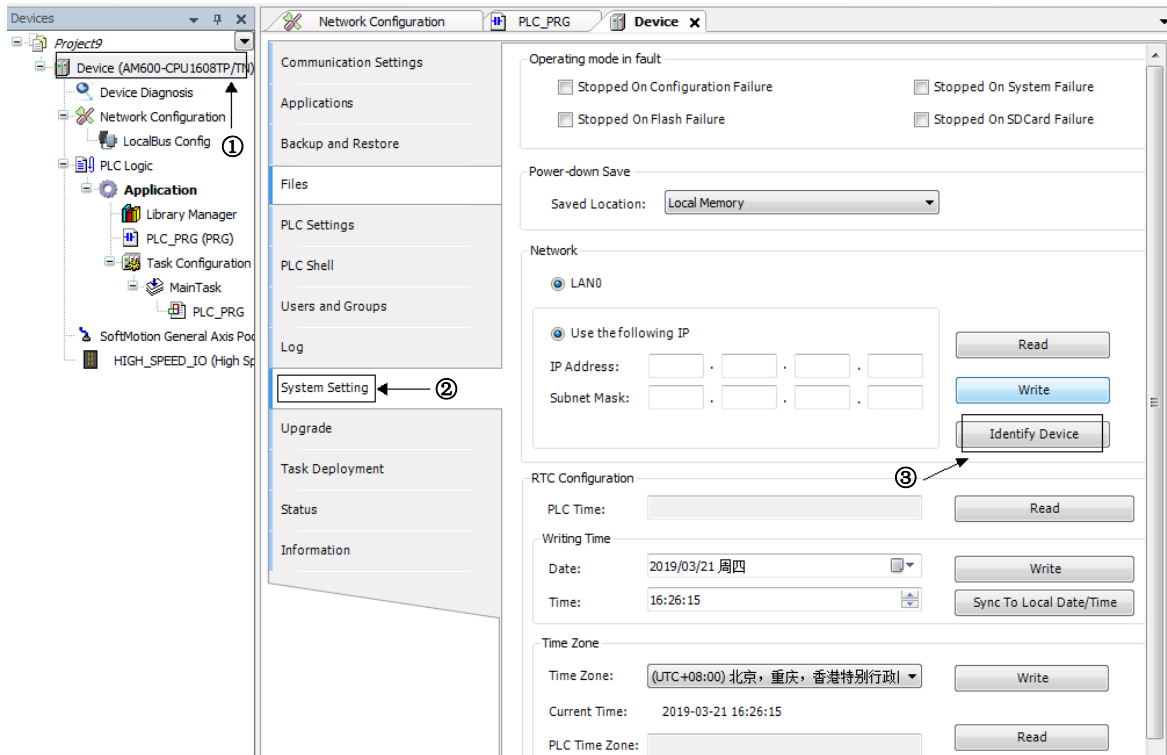
1 - Double-click **Device**. 2 - Switch to the **System Setting** tab. 3 - Click **Sync To Local Date/Time**. The 7-segment LED of PLC selected on the **Communication Setting** tab page alternately blinks.

The two bits of 7-segment LED on the AM600 or AM610 you have logged in to alternately display 0, shown as follows:

The two bits alternatively display 0 at a frequency of 1 Hz, indicating that this controller is being verified for login.
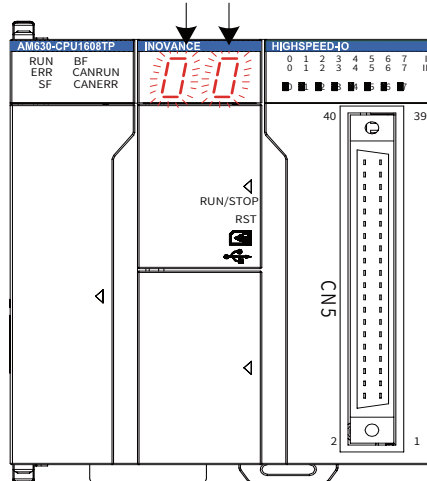
The LED stops blinking only when you click **OK** in the InoProShop pop-up dialog box, and the original information is restored:

Memo No. _____

Date   /   /
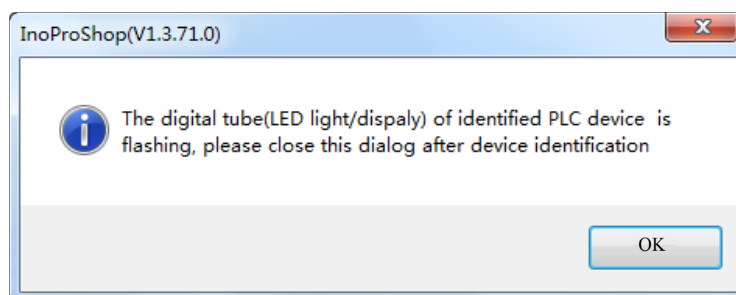
Chapter 3 Network Settings

3 Network Settings

3.1 Device Configuration

Device configuration is the first step of PLC programming. It involves two functions: network configuration and hardware configuration. You can use the two functions to deploy the device.

■ Network Configuration

It is designed from the perspective of bus-type network topology, and is the entrance of device configuration.

■ Hardware Configuration

It is used to add the expansion I/O modules of medium-sized PLC.

3.1.1 Network Configuration

After creating an InoProShop project, double-click the **Network Configuration** node in the left device tree, shown as follows:
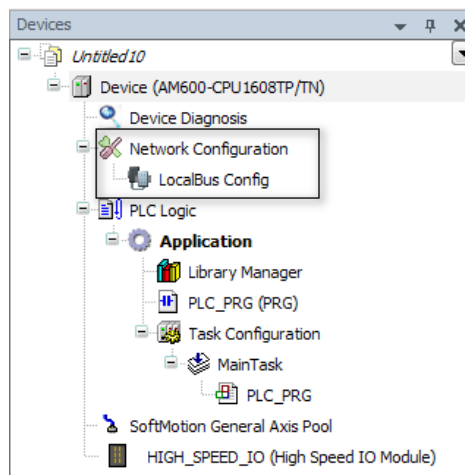
Figure 3-1 Network Configuration node

Double-click this node to open the **Network Configuration** interface and device list (as shown in Figure 3-1). The network configuration interface displays the PLC currently used by the user program, and the device list includes all the devices supported by PLC.
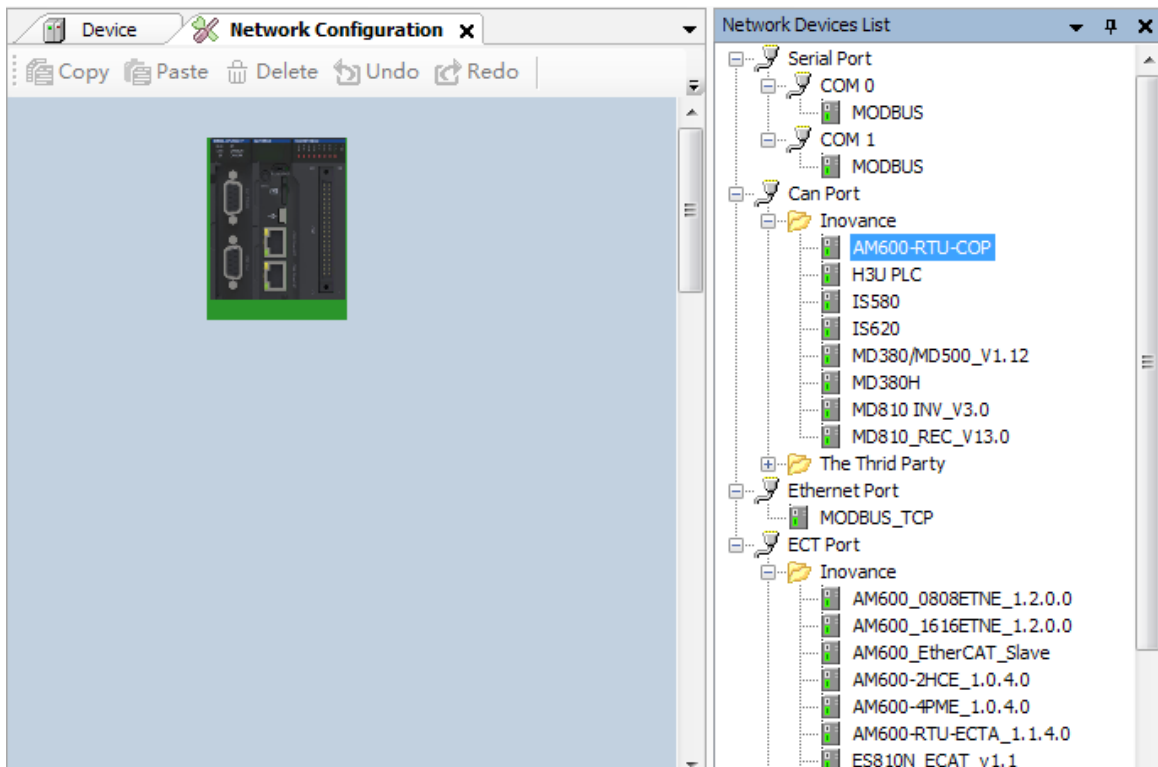
Figure 3-2 Network Configuration

1 Configuring PLC as Master or Slave Device

When you click the PLC in network configuration, the master/slave stations supported by PLC are displayed, shown as follows. Click the desired checkbox to enable the master/slave stations supported by the CPU.
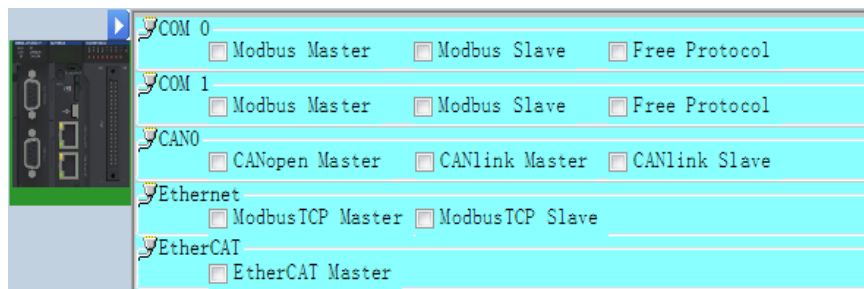
When a master station (except the CANlink master station) function is enabled for the CPU, the bus-type topology is displayed. For example, the following figure shows the enabled EtherCAT master station:
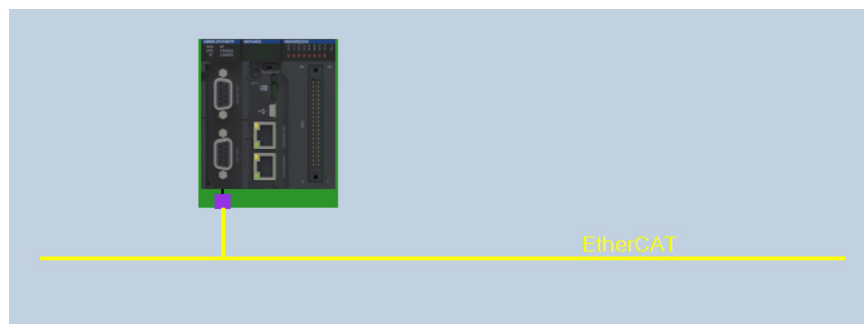
■ Add slave station.

After a master station in CPU is enabled, you can add the slave station under the corresponding bus. The slave station can be added by three methods (taking EtherCAT bus for example):

1) Enable the EtherCAT master station function, and then select a slave station node under the EtherCAT port node in Figure 3-1. Drag the node to the network configuration interface.

2) Enable the EtherCAT master station function, and then double-click a slave station node under the EtherCAT port node in Figure 3-1.

3) Double-click a slave station node under the EtherCAT port node in the device list. If you use this method, the internal master station function of the CPU will be enabled first.

To add I/O modules for an added slave station (slave station for AM600), double-click the device to open the Hardware Configuration interface.
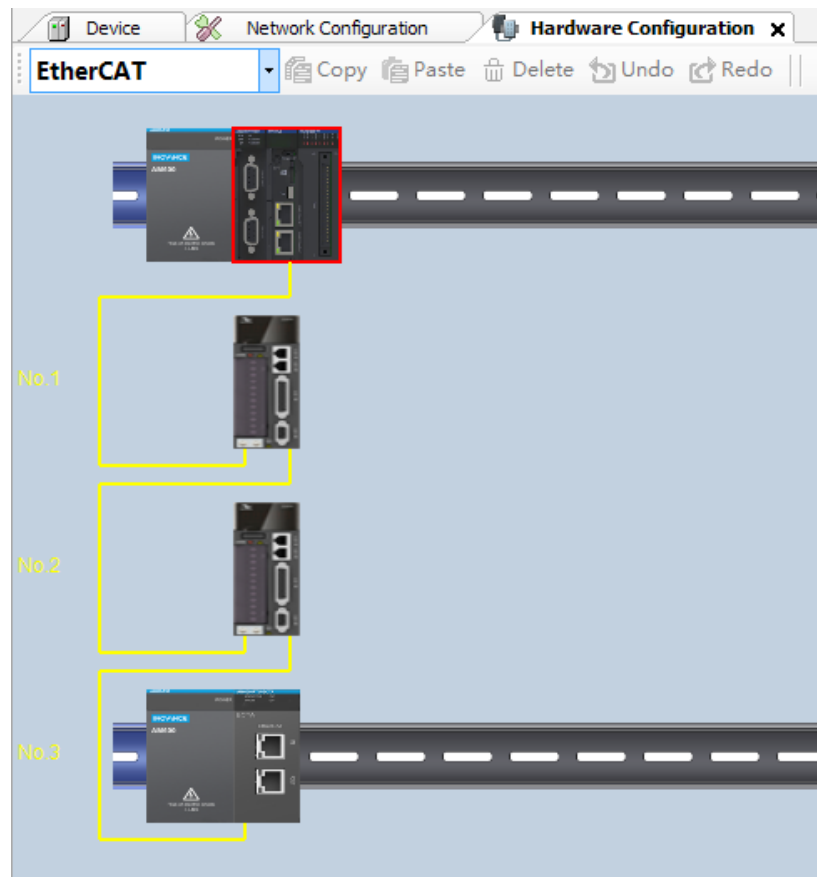
■ View basic device information.

Click a device on the Network Configuration interface, and you can see the basic device information in **Config Device Information Output** > **DeviceDefaultInfoList**.
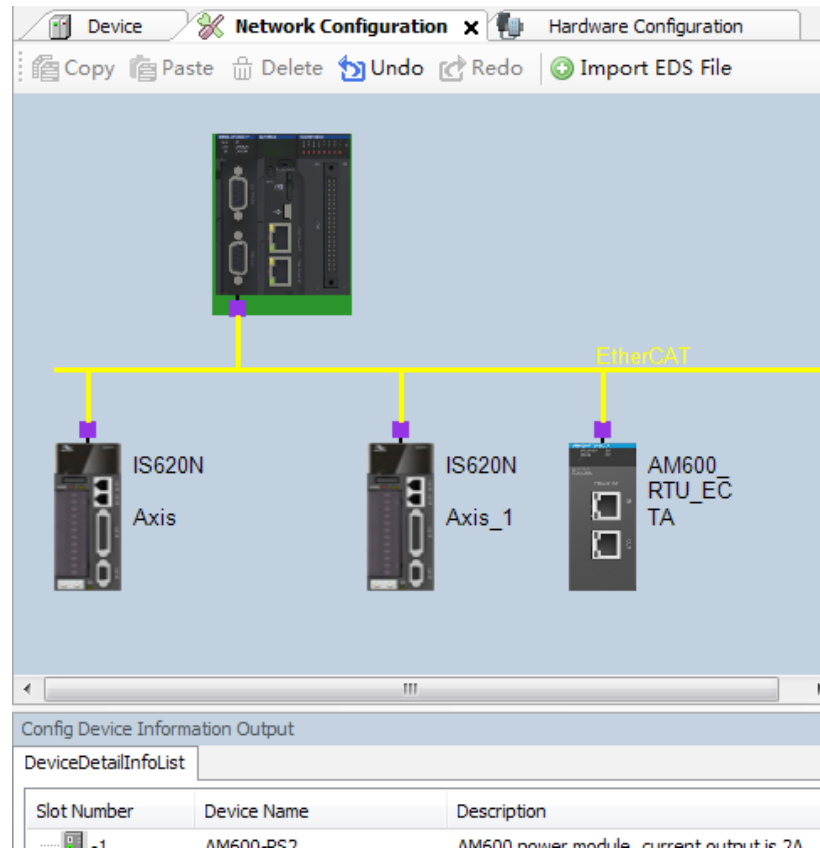
■ Open the device configuration interface.

Right-click the EtherCAT slave station in network configuration, and choose **Open Editor Page** from the shortcut menu, shown as follows:
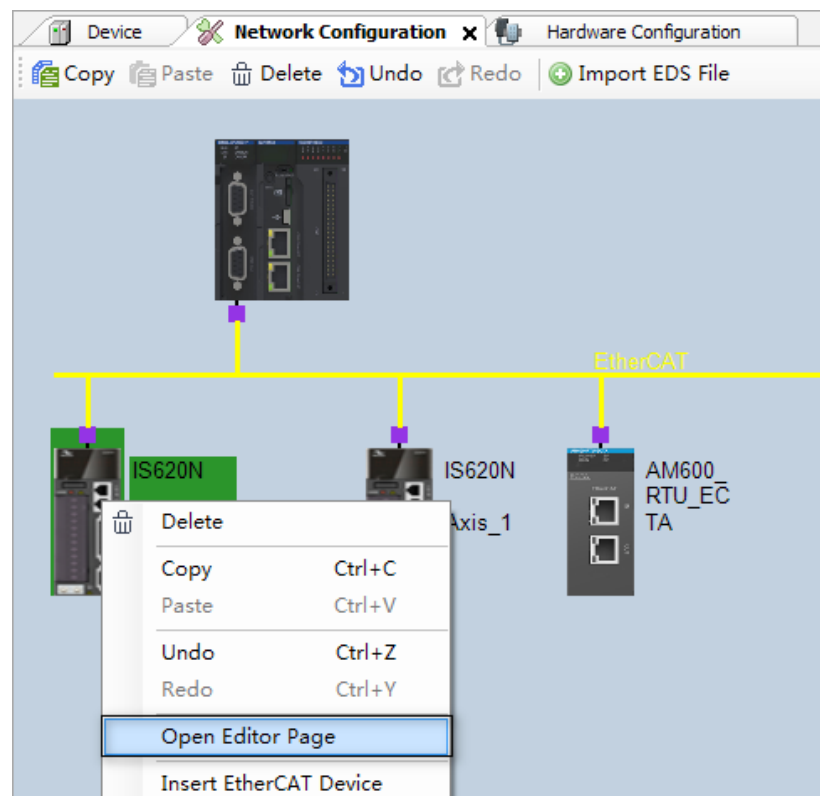
■ Insert the EtherCAT slave station.

Right-click the EtherCAT slave station in network configuration, and choose **Insert EtherCAT Device** from the shortcut menu, shown as follows:
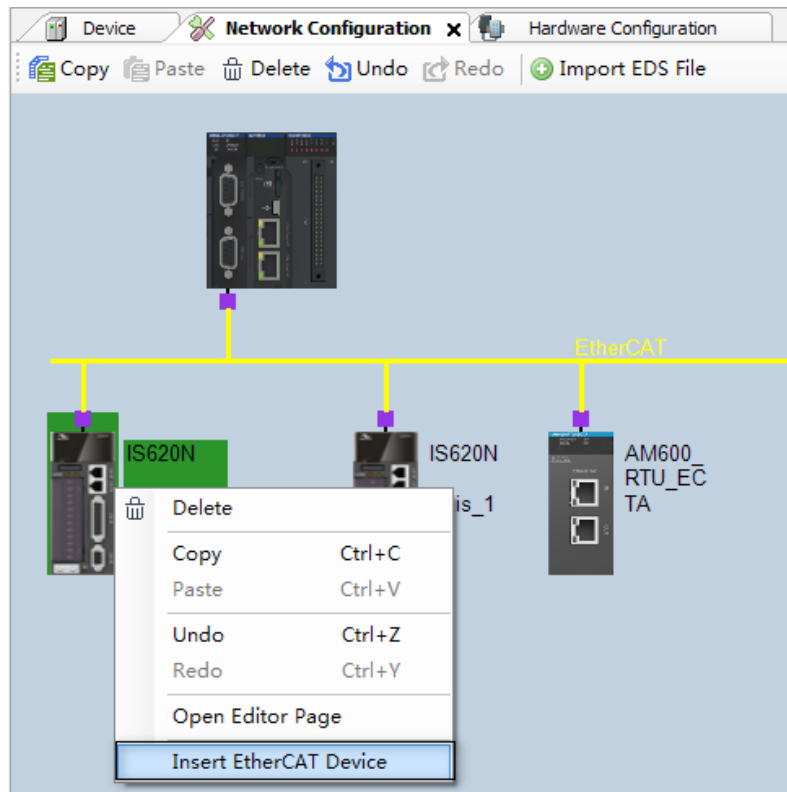
■ Config devices can be copied, deleted, and added. For details, see Config Device Common Operations.

■ Edit configuration.

If the network configuration includes repeated Modbus slave station addresses or ModubsTCP slave station IP addresses, the repeat information is displayed in the output box during project compiling. For details, see Config Compiling Error Locating.

2 Device Information List

To open the device information list, choose **View** > **Config Device Information View**. The config device basic information is displayed, including slot number, device name, and description. The information is minimized at the bottom of the interface by default ( ). You need to open the list manually by clicking it.

Figure 3-3 Device Information List

■ Slot number

The slot numbers match the slot numbers in Hardware Configuration. The numbers of the slots on rack and on the communication slave station both start from 1. Slot -1 matches the AM600 power module, and slot 0 matches the CPU.

■ Device name

The device names are the same as the device names in the left device view.

■ Description

The basic description of devices, including operating indicators and functions.

To locate a device in the configuration interface, click a row in the device list. To open the configuration interface of a device, double-click a row.

3 Config Device Common Operations

The common operations of config devices include copy, paste, cancel, restore, delete, import EDS, GSD, and ECT files, zoom in, and zoom out.
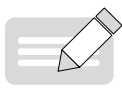
> **NOTE**
>
> 1) The copy, paste, delete, cancel, and restore operations only apply to I/O modules on the Hardware Configuration interface and to slave stations on the Network Configuration interface.
> 2) If you perform copy, paste, or delete operation on the slave station on the network configuration interface, the same operation is also performed on its modules.
> 3) The CPU of AM600-CPU1608TP supports the import of EDS and ECT files, but does not support the import of GSD files. The CPU of AM610-CPU1608TP supports only the import of GSD files.

■ Import EDS files: The network device list contains some CANopen devices by default. If you need to add other CANopen devices, import the corresponding standard EDS file. After the file is imported, the device is added to the CAN port node in the network device list. If the imported device is from Inovance, it will be displayed under the Inovance node; otherwise, it is displayed under the third-party vendor node.

■ Import GSD files: The network device list contains some DP devices by default. If you need to add other DP devices, import the corresponding standard GSD file. After the file is imported, the device is added to the DP port node in the network device list. If the imported device is from Inovance, it will be displayed under the Inovance node; otherwise, it is displayed under the third-party vendor node.

■ Import ECT files: The network device list contains some EtherCAT devices by default. If you need to add other EtherCAT devices, import the corresponding standard EtherCAT xml (*.xml) file. After the file is imported, the device is added to the EtherCAT port node in the network device list. If the imported device is from Inovance, it will be displayed under the Inovance node; otherwise, it is displayed under the third-party vendor node.

3.1.2 Hardware Configuration

The hardware configuration uses the rack and slot used in device configuration to simulate the field device modular configuration. Hardware configuration applies to the I/O modules of medium-sized PLC products.

From the aspect of configuration procedure, to add a remote I/O module, you need to configure the communication module in Network Configuration first, and then configure the I/O module in Hardware Configuration; to add a local I/O module, directly open the Hardware Configuration to perform operations. Hardware Configuration supports multi-bus I/O configuration, depending on the used CPU model.

1 Opening the Hardware Configuration Interface

Except Modbus and Modbus TCP devices, other bus-type devices should have matching Hardware Configuration interfaces.

You can access the Hardware Configuration interface in two ways:

1) Double-click a device on the Network Configuration interface.

2) Double-click a bus node under the **Network Configuration** node in the left device tree, shown as follows.
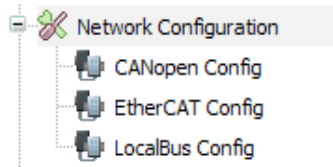
By default, the LocalBus Config, namely, local bus configuration node, is available. Double-click it to perform the local module configuration:
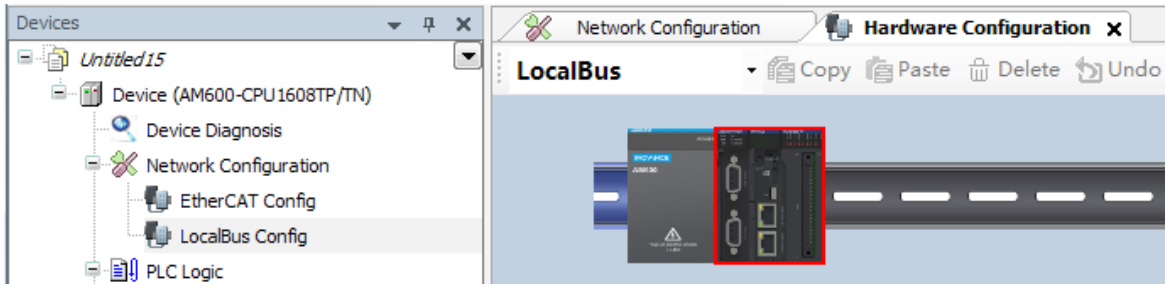
In addition, the **Input/Output Module List** is displayed on the right.
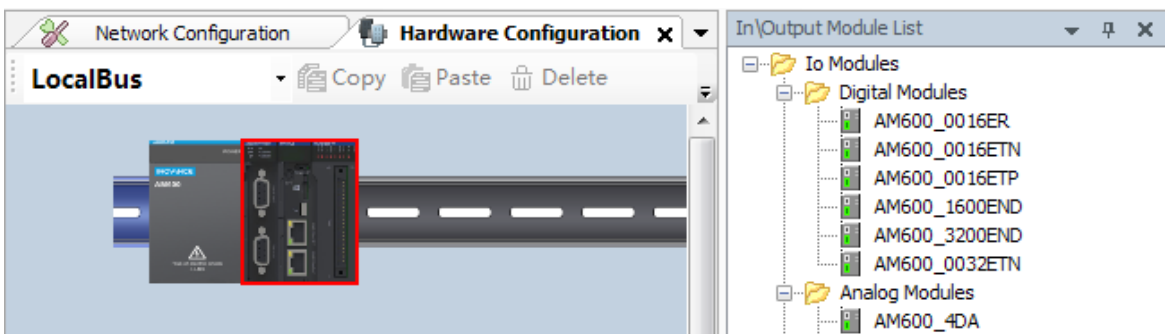
2 Switching Bus

You can switch the bus of hardware configuration in two ways:

1) Double-click a bus node under the **Network Configuration** node in the left device tree.

2) Select another bus type in the current Hardware Configuration interface, shown as follows:
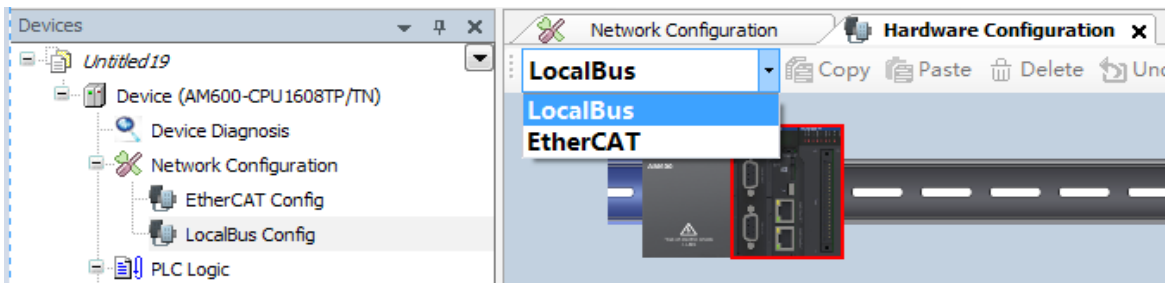
3 Adding Module

You can add I/O modules in three ways:

1) Double-click an idle slot on the rack. In the pop-up module list, double-click a module to add it.

2) Select a node from the right module list, and drag it to an idle slot.

3) Select a rack (blue part in the figure) or device, and double-click a device in the right module list.

Then you can add the devices to the idle slots in order. If you click an idle slot, the selected device is added to this slot.
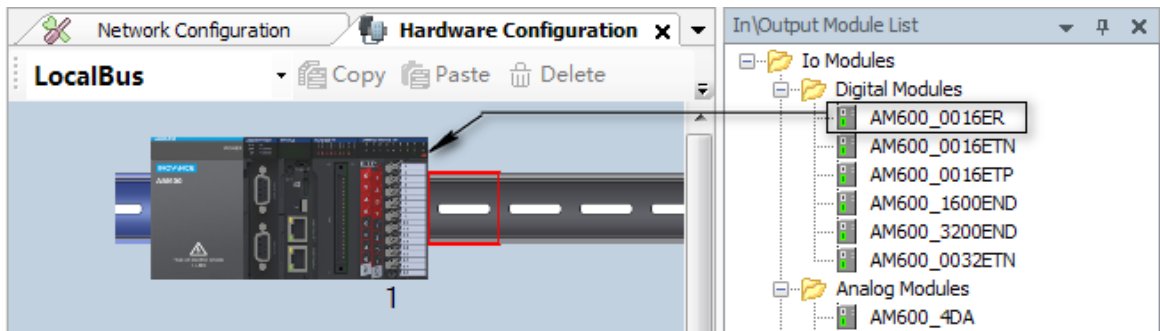
4 Dragging Module

Select a module and drag it to the target slot. By using the dragging function, you can exchange the locations of two modules or move a module to an idle slot. However, the modules in the main rack and expansion rack cannot be interchanged.

3.1.3 Configuration Compiling Error Locating

The config device defines configuration rules and error detection mechanism, for example, repeated station addresses of MODBUS devices or IP addresses of TCP devices in Network Configuration. If the slave station in the expansion rack of Hardware Configuration is not connected to I/O module, a configuration compiling error will be reported.

If a configuration error occurs during compiling, the InoProShop message box will display the error. Double-click the error list to go to the corresponding configuration interface and the red rectangular box blinks three times, shown as follows:
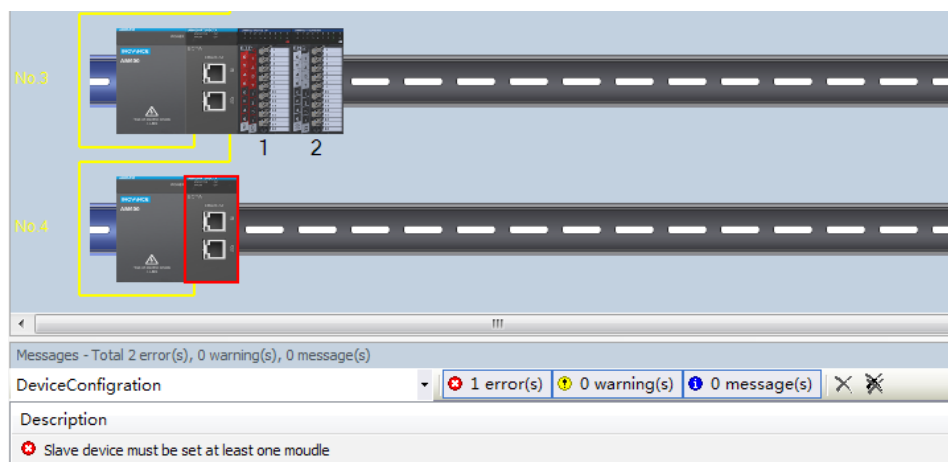
3.2 CPU Configuration

The CPU module is the main module of a medium-sized PLC. The CPU is configured based on the control system requirements of the PLC hardware, to complete the configurations of PLC and its control system. A medium-sized PLC supports EtherCAT bus, PROFIBUS DP, Modbus RTU, CAN bus, and Modbus TCP, and also supports high-speed I/O. Therefore, to complete all CPU configurations, you need to set the bus parameters according to the PLC hardware network configuration.

For example, the CPU module of AM600 has built-in high-speed I/O, and local I/O modules can be configured. In addition, on the CPU configuration interface, you can set the CPU system parameters and CPU firmware upgrade.

3.2.1 General CPU Configuration Procedure

1) Design the entire CPU hardware network structure.

2) Activate the corresponding bus in Network Configuration and add the slave stations corresponding to the bus. Currently, the CPU supports EtherCAT bus, DP bus, CANopen, CANlink, Modbus RTU, and Modbus TCP.

3) For the EtherCAT AM600 slave station, CANopen AM600 slave station, or DP AM600 slave station, add I/O modules to Hardware Configuration.

4) Configure the master station, slave station, and module configuration parameters corresponding to the bus.

By default, the AM600 CPU has the high-speed I/O function. Each CPU can be configured with up to 16 local I/O modules. In addition, you need to configure the CPU system parameters, PLC I/O update, PLC bus task, PLC user management, log, upgrade, and tasks according to the actual requirements.

To know the parameter settings of buses and their corresponding slave stations, see the chapters of the buses. For the built-in functions of Codesys, such as PLC I/O update, PLC bus task, PLC user management, log, and tasks, see the Codesys software help. In this document, the CPU configuration mainly involves the functions of medium-sized PLC: CPU parameter configuration, I/O module configuration, and high-speed I/O configuration.

3.2.2 CPU Parameter Configurations

1 System Configurations

System settings include the configurations of downtime caused by CPU fault, location retaining upon outage, network address, and system time, shown as follows:

Figure 3-4 System settings dialog box

Operating mode in fault

■ Stopped On Configuration Failure: Whether the CPU stops running when configurations are inconsistent, for example, when the configured I/O module mounted to the CPU does not match the physically connected I/O module.

■ Stopped On System Failure: Whether the CPU stops running when an system error occurs, for example, when interrupt error or stack overflow occurs.

■ Stopped on Flash Failure: Whether the CPU stops running when a Flash error occurs. This function is unavailable currently.

■ Stopped on SD Card Failure: Whether the CPU stops running when an SD card error occurs, for example, when the SD card memory is used up or SD card is lost. This function is unavailable currently.

Power-down Save

■ Saved Location: Sets the data saving location upon power failure, including local memory and SD card. A maximum of 512 KB data can be stored upon power failure.

> **NOTE**
> When you set the saved location as SD card, ensure that an SD card is available; otherwise, data will be lost upon power failure.

Network

■ LAN0: indicates the network interface name used by the PLC for Ethernet communication. AM600 and AM400 have only one Ethernet interface, and AC800 has two Ethernet interfaces. The network names vary with PLCs. You can configure differentiated network information according to network interface configurations.

■ Use the following IP: The IP address of PLC can be manually modified, or automatically obtained. This configuration is used to manually modify the PLC network information.

■ Automatically obtained IP address: The IP address of the PLC is assigned by a router or switch. Note: This function is available only for AC800.

■ IP Address: indicates the IP address of PLC.

■ Subnet Mask: indicates the subnet mask of PLC.

■ Gateway: sets the gateway for PLC. Note: This function is available only for AC800.

■ Read: reads the IP address and subnet mask of PLC, which are displayed in the IP address and subnet mask edit boxes.

■ Write: writes the IP address and subnet mask in the edit boxes into PLC. If you have logged in and connected to the network, you will be logged out. In this situation, you need to re-log in. If the USB connection is used, you do not need to re-connect the device.
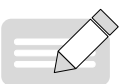
> **NOTE**
> When the IP address needs to be read and written, select the PLC to be read and written in the **Communication Settings** tab. In addition, the IP address and subnet mask to be written must comply with the related standards.

■ Identify Device: identifies the PLC to be connected. After you configure PLC scanning on the Communication Settings tab, multiple PLCs may be detected. After selecting one PLC, click this button. Then the two digits of the 7-segment LED on the PLC panel will alternately display 0, as shown in Figure 3-5.

Two zeros blink alternately at the frequency of 1 Hz, indicating that the controller is being identified.
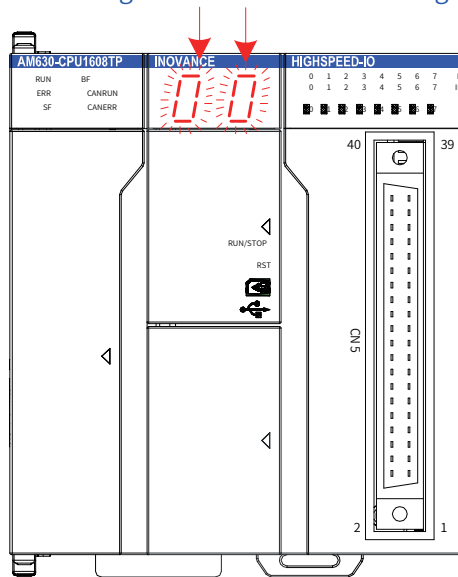
Figure 3-5   PLC 7-segment LED in Identifying state

Moreover, the software tool InoProShop displays the dialog box shown in Figure 3-6. Close the dialog box to complete the identification, and then the 7-segment LED restores the default state.
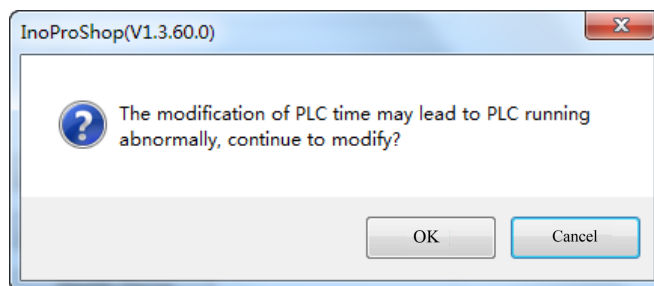
Figure 3-6   Identifying device

RTC configuration

■   PLC Time: displays the current PLC time.

■   Read: reads the PLC time.

■   Write: writes the current date and time set on PLC. The current date and time are displayed in the left edit boxes.

■   Sync To Local Date/Time: writes the current date and time from the PC to PLC.

> **NOTE**
>
> When the system writing PLC time or synchronizing local date/time to PLC, the PLC may be affected, for example, the bus synchronization may be affected. Therefore, before writing the PLC time, ensure that the PLC is in Stop state. It is recommended to hot reset the PLC after the time is written.

Time Zone

■   Time Zone: reads the PLC time zone. For example, the time zone of Beijing is UTC+8.

■   Read: reads the time zone of PLC.

■   Write: writes the time zone selected for PLC.

2 Upgrade

The upgrade interface is used for the upgrade of PLC firmware, as shown in Figure 3-7. The PLC firmware upgrade package provides the software data for upgrade, which may include UBOOT, Device Tree, kernel, and system program. Generally, the upgrade package includes only the system program.

NOTE

The upgrade function is unavailable. Use the InoProShop tool to perform an upgrade.

Figure 3-7 Upgrade dialog box

PLC Information

■ PLC Model: AM600 or AM610

■ Firmware Version: firmware version of PLC, for example, 1.2.3.0

■ Detailed Version: detailed version information about the PLC, which may include UBOOT, Device Tree, kernel, and system program, as shown in Figure 3-8. If the PLC firmware has not been upgraded, the version information may not be included.

■ Get PLC Information: gets PLC model and firmware information. If the PLC firmware has not been upgraded, PLC information may not be obtained.

Firmware Upgrade

■ Firmware Package: sets the firmware upgrade package, with the file name extension **.upgrade**.

■ Compatible Device: displays the devices compatible with the firmware upgrade package. The upgrade can be performed only when the device is compatible with the PLC model.

■ Firmware Version: displays the firmware version of the upgrade package.

■ Firmware In Details: gets detailed information about the firmware upgrade package, shown as follows:
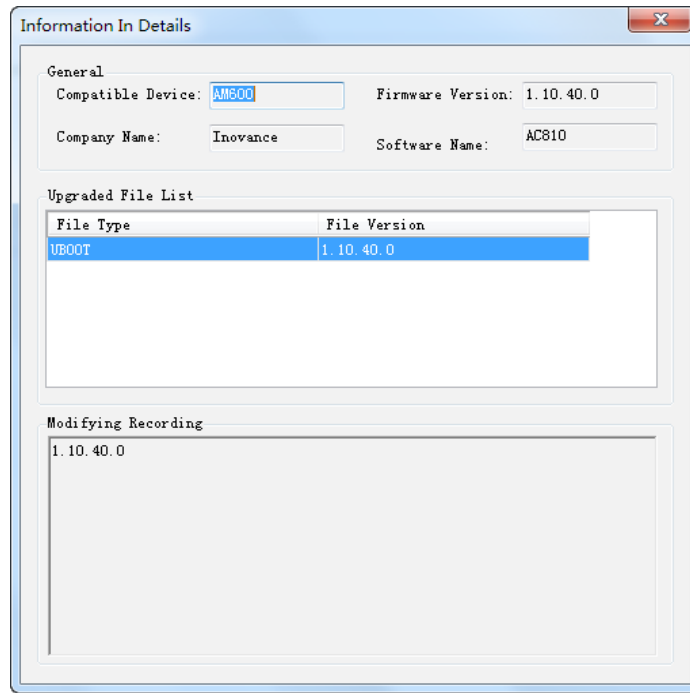
Figure 3-8 Firmware detailed information dialog box

■ Upgrade: starts the firmware upgrade. Before performing the upgrade, the system checks the device type and upgrade firmware file version. If the upgrade firmware version is later than the PLC firmware version, the upgrade is performed. If the versions are the same, the upgrade is not performed. If the upgrade firmware version is earlier than the PLC firmware version, the upgrade is performed only after confirmation.

NOTE

◆ Before the upgrade, scan the device to be upgraded in the **Communication Settings** interface and select the PLC to be upgraded.

◆ Do not power off the device during the upgrade; otherwise, unrecoverable system faults may occur.

◆ The upgrade will last about two minutes. After the upgrade, the device automatically restarts.

◆ After the restart (upgrade completed), the 7-segment LED displays 00 or dynamically changing digits.

◆ After the upgrade, the PLC device name may be changed. Scan the device again.

After the upgrade, display and verify that the PLC information and detailed version information are consistent with the firmware version and detailed information, as shown in Figure 3-9:
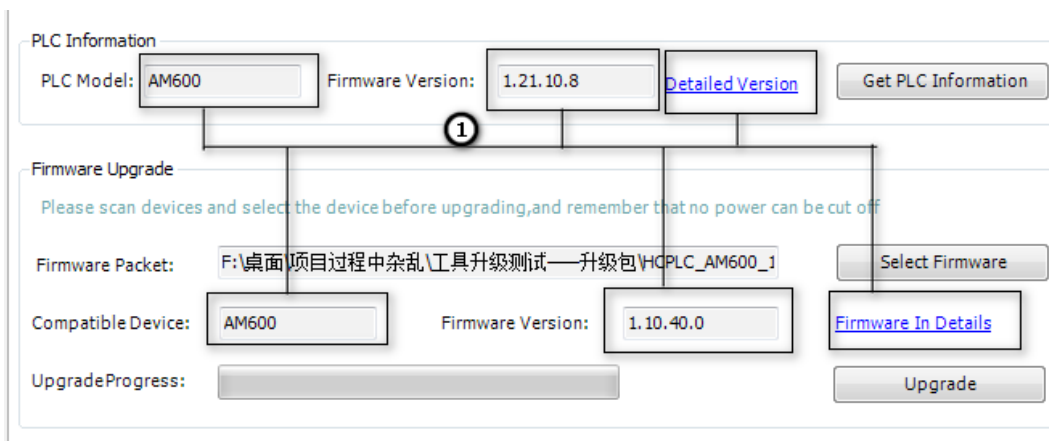
1 - Verify that the firmware information and PLC information are consistent.

Figure 3-9 Verify upgrade

3 Information

The PLC basic information is displayed, including Name, Vendor, Categories, Type, ID, Version, Order Number, Description, and Image, as shown in Figure 3-9.

Moreover, after you log in, the Information interface displays the PCB software version of the CPU and logic software version. The PCB software version is the CPU system program version, and the logic software version is the FPGA software version within the CPU.
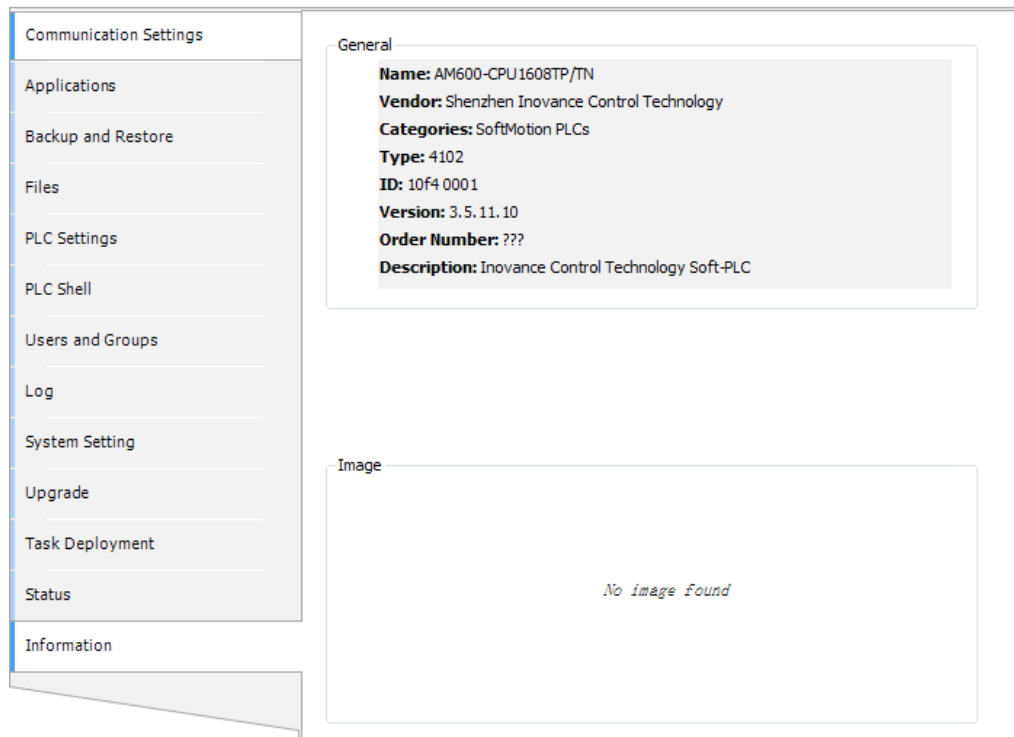
Figure 3-10 CPU information

3.2.3 I/O Module Configurations

Five types of I/O modules are available:

Digital input (DI), digital output (DO), analog input (AD), digital output (DA), and temperature.

Digital output modules are classified into three types: relay output (ERN type), NPN output (ETN type), and PNP output (ETP type).

Digital input and output include 16-channel and 32-channel.

The temperature module includes 4TC (4-channel temperature detection module, supporting thermocouple), 8TC (8-channel temperature detection module, supporting thermocouple), and 4PT (4-channel temperature detection module, supporting thermocouple).

1 Digital Input Module

There is no module parameter setting for digital input module. Only the I/O mapping, status, and information interfaces are available. You only need to map the I/O variables on the I/O mapping interface to obtain the digital input values. The 16-channel digital input module is used as an example here.

1)   DI16 I/O mapping

DI16 is a 16-bit digital input module. As shown in Figure 3-11, on the I/O mapping interface, each bit or every eight bits can be mapped to one variable to obtain the input value. For details, click the I/O Mapping link.
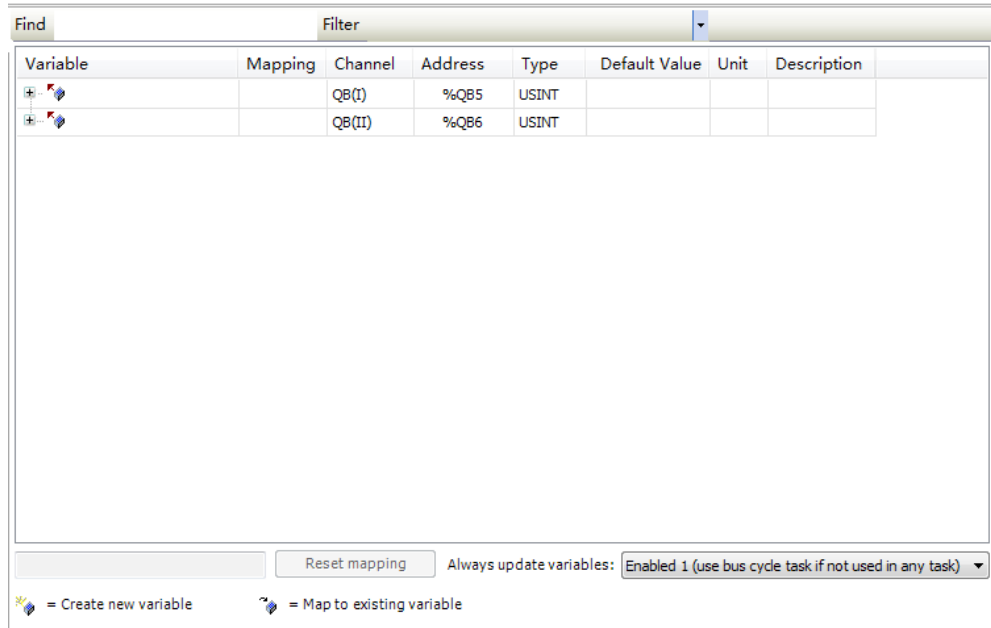
Figure 3-11 DI16 I/O mapping dialog box

2)   Information

The DI16 module basic information is displayed, including Name, Vendor, Categories, Type, ID, Version, Order Number, Description, and Image, as shown in Figure 3-12.

Moreover, after you log in, the Information interface displays the DI module logic software version, which is the FPGA software version within the DI module.

Figure 3-12 DI information

2 Digital Output Module

There is no module parameter setting for digital output module. Only the I/O mapping, Status, and Information interfaces are available. You only need to map the I/O variables on the I/O mapping interface and output the mapped variable values to the digital output module. The digital output includes 16-channel and 32-channel. They have similar I/O mapping interfaces. The 16-channel digital output module is used as an example here.

1) DO16 I/O mapping

DO16 is a 16-bit digital input module. As shown in Figure 3-13, on the I/O mapping interface, each bit or every eight bits can be mapped to one variable to output the variable values. For details, click the I/O Mapping link.

Variable	Mapping	Channel	Address	Type	Default Value	Unit	Description
IB(I)		IB(I)	%IB2	USINT			
		I0	%IX2.0	BOOL			
		I1	%IX2.1	BOOL			
		I2	%IX2.2	BOOL			
		I3	%IX2.3	BOOL			
		I4	%IX2.4	BOOL			
		I5	%IX2.5	BOOL			
		I6	%IX2.6	BOOL			
		I7	%IX2.7	BOOL			
		IB(II)	%IB3	USINT			

Find   Filter Show all

Reset mapping   Always update variables: Enabled 1 (use bus cycle task if not used in any task)

= Create new variable   = Map to existing variable
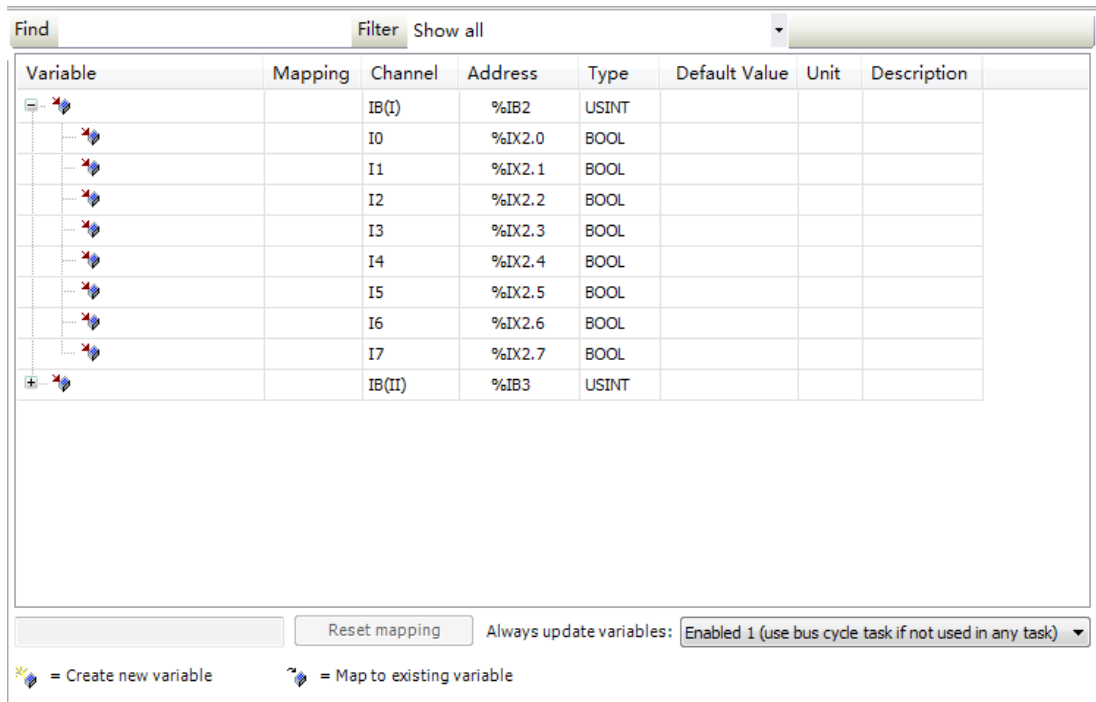
Figure 3-13 DO16 I/O mapping dialog box

2) Information

The DO16 module basic information is displayed, including Name, Vendor, Categories, Type, ID, Version, Module ID, Description, Order Number, and Image, as shown in Figure 3-14.

Moreover, after you log in, the Information interface displays the DO module logic software version, which is the FPGA software version within the DO module.

General

Name: CPU DI16
Vendor: Inovance
Categories:
Type: 40010
ID: 10F4 0010
Version: 00.00.00.10
Order Number: 14400005
Description: CPU DI16 Module hardware description

Image

No image found
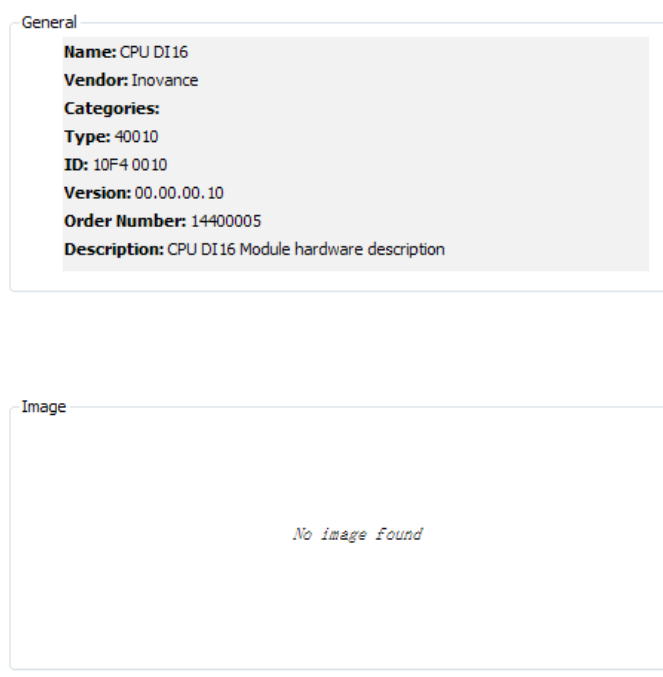
Figure 3-14 DO information

3 Analog Input Module

1) General settings

Analog input module includes four channels, each of which has independent parameter settings and I/O mapping register (16-bit) settings. The following is the description of one channel in each module.
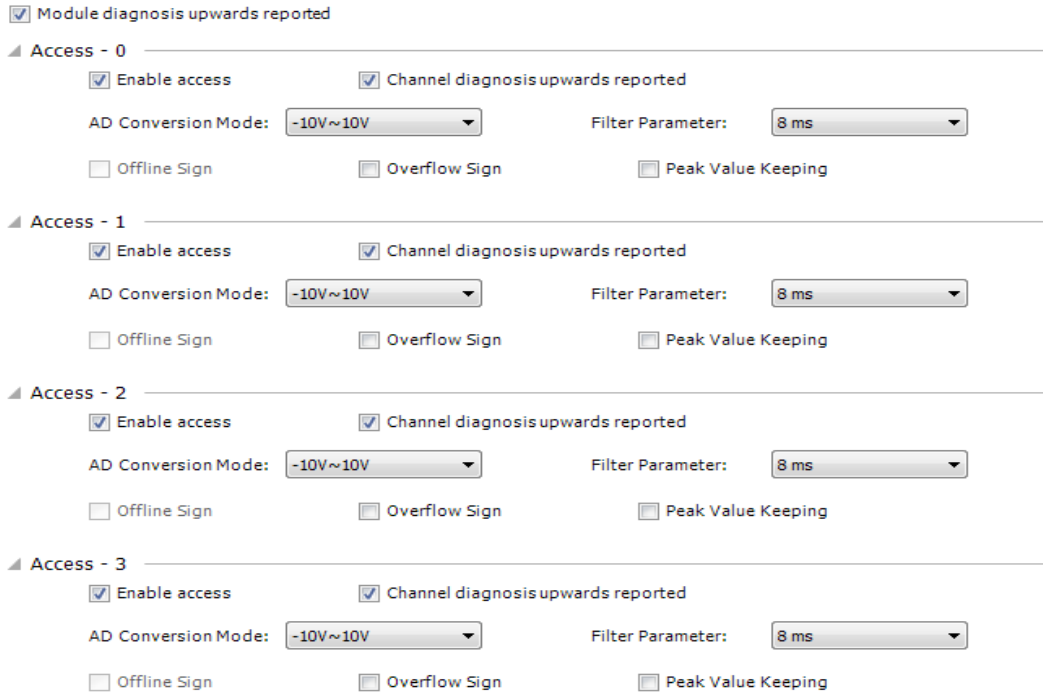
Figure 3-15 General settings of analog input module

- Module diagnosis upwards reported: Whether to report the module faults to the parent device (such as CPU and remote module slave station). If the faults are configured to be reported and the parent device is configured with Stopped On Failure, the parent device will stop the running of the device.

- Enable access: Whether to activate the channel. The channel is available only after it is activated.

- Channel diagnosis upwards reported: Whether to report the channel faults to the parent device (such as CPU and remote module slave station). If the faults are configured to be reported and the parent device is configured with Stopped On Failure, the parent device will stop the running of the device.

- AD Conversion Mode: Conversion mode of the analog input. This setting specifies the channel input conversion type, conversion value range, and the mappings between conversion types and digital values, shown as follows:

Table 3-1 Mappings between analog values of analog input and digital values

	Rated Input Range	Rated Digital Value	Input Limit Range	Limiting Digital Value
Analog voltage input	-10 V to +10 V	-20000 to 20000	-11 V to 11 V	-22,000 to +22,000
	0 V to +10 V	0 to 20000	-0.5 V to 10.5 V	-1000 to +21,000
	-5 V to 5 V	-20000 to 20000	-5.5 V to 5.5 V	-22,000 to +22,000
	0 V to 5 V	0 to 20000	-0.25 V to 5.25 V	-1000 to +21,000
	1 V to 5 V	0 to 20000	0.8 V to 5.2 V	-1000 to +21,000
Analog current input	-20 mA to 20 mA	-20000 to 20000	-22 mA to 22 mA	-22,000 to +22,000
	0 mA to 20 mA	0 to 20000	-1 mA to 21 mA	-1000 to +21,000
	4 mA to 20 mA	0 to 20000	3.2 mA to 20.8 mA	-1000 to +21,000

■ Filter Parameter: Filter time of the analog input channel, ranging from 1 ms to 255 ms.

■ Offline Sign: Whether to detect the offline state of analog input channel. The system cannot distinguish the input value 0 and offline state of the analog input module, so all values in the conversion mode range, including 0, cannot activate the offline sign.

■ Overflow Sign: Whether to detect overflow of the analog input channel.

■ Peak Value Keeping: Whether to keep the peak value input of the analog input channel.

2) AI4 I/O mapping

AI4 is 4-channel analog input. Each channel matches a 16-digit integer. For the mappings between analog values and digital values, see General Settings of Analog Input. On this interface, each 16-digit integer can be mapped to a variable to obtain the digital value matching an analog value of input channel. For details, click I/O Mapping.

Figure 3-16 AI4 I/O mapping dialog box

3) Information

The AI4 module basic information is displayed, including Name, Vendor, Categories, Type, ID, Version, Module ID, Description, Order Number, and Image.

Moreover, after you log in, the Information interface displays the PCB software version and logic software version of the AI4 module. The PCB software version is the embedded software version of AI4 and the logic software version is the FPGA software version within the AI4 module.

General

Name: CPU AI4
Vendor: Inovance
Categories:
Type: 40030
ID: 10F4 0030
Version: 00.00.00.10
Order Number: 01440006
Description: CPU AI4 Module hardware description

Image

No image found

Figure 3-17 AI4 information

4 Analog Output Module

1) General settings

Analog output module includes four channels, each of which has independent parameter settings and I/O mapping register (16-bit) settings. The following is the description of one channel in each module.

Figure 3-18 General settings of analog output module

■ Module diagnosis upwards reported: Whether to report the module faults to the parent device (such as CPU and remote module slave station). If the faults are configured to be reported and the parent device is configured with Stopped On Failure, the parent device will stop the running of the device.

■ Enable access: Whether to activate the channel. The channel is available only after it is activated.

■ Channel diagnosis upwards reported: Whether to report the channel faults to the parent device (such as CPU and remote module slave station). If the faults are configured to be reported and the parent device is configured with Stopped On Failure, the parent device will stop the running of the device.

■ Conversion Mode: Conversion mode of the analog output. This setting specifies the channel output conversion type, conversion value range, and the mappings between conversion types and digital values, shown as follows:

Table 3-2 Mappings between analog values of analog output and digital values

	Rated Output Range	Rated Digital Value	Output Limit Range	Limiting Digital Value
Analog Voltage Output	-10 V to +10 V	-20000 to 20000	-11 V to 11 V	-22,000 to +22,000
	0 V to 10 V	0 to 20000	-0.5 V to 10.5 V	-1000 to +21,000
	-5 V to 5 V	-20000 to 20000	-5.5 V to 5.5 V	-22,000 to +22,000
	0 V to 5 V	0 to 20000	-0.25 V to 5.25 V	-1000 to +21,000
	1 V to 5 V	0 to 20000	0.8 V to 5.2 V	-1000 to +21,000
Analog current output	0 mA to 20 mA	0 to 20000	0 mA to 21 mA	0 to +21,000
	4 mA to 20 mA	0 to 20000	3.2 mA to 20.8 mA	-1000 to +21,000

■ State Output After Stop: Sets the output retaining value after the module stops running.

■ Output zero: Always outputs 0 after the module stops running.

■ Output last value: Always outputs the last value after the module stops running.

■ Output preset value: Always outputs the preset value after the module stops running. The preset value can be an analog value or a digital value. The analog values and digital values have the mapping relationship. If the analog or digital value is changed, its corresponding digital or analog value is also changed. The preset value range depends on the conversion mode. For details, see the description of conversion mode.

2) AO4 I/O mapping

AO4 is 4-channel analog input. Each channel matches a 16-digit integer. For the mappings between analog values and digital values, see General Settings of Analog Output. On this interface, each 16-bit integer can be mapped to a variable, and this variable is output to the current channel. Then the analog output module converts the variable into the analog value for output. For details, click I/O Mapping.
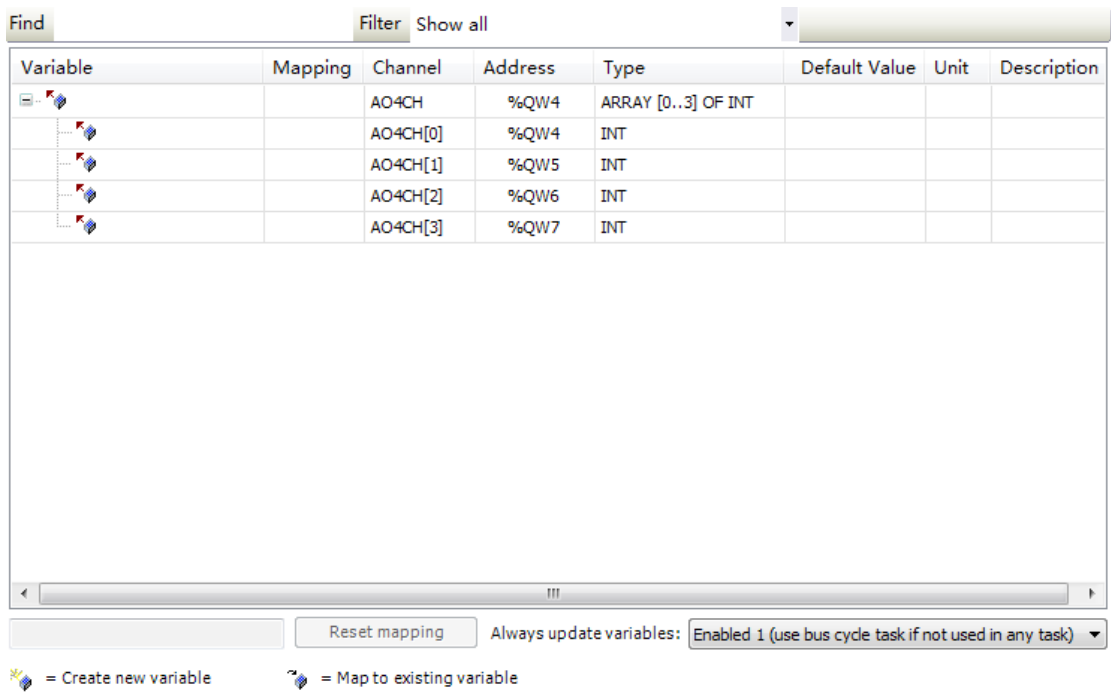
Figure 3-19 AO4 I/O mapping dialog box

3) Information

The AO4 module basic information is displayed, including Name, Vendor, Categories, Type, ID, Version, Module ID, Description, Order Number, and Image.

Moreover, after you log in, the Information interface displays the PCB software version and logic software version of the AO4 module. The PCB software version is the embedded software version of AO4 and the logic software version is the FPGA software version within the AO4 module.

General

Name: CPU AO4
Vendor: Inovance
Categories:
Type: 40040
ID: 10F4 0040
Version: 00.00.00.10
Order Number: 01440007
Description: CPU AO4 Module hardware description

Image

No image found

Figure 3-20 AO4 information

5 Temperature Module

The temperature module includes 4TC (4-channel temperature detection module, supporting thermocouple), 8TC (8-channel temperature detection module, supporting thermocouple), and 4PT (4-channel temperature detection module, supporting thermocouple). It has the corresponding general settings and channel settings.

General settings include the unit type and sample cycle of the temperature module. Channel settings include the sensor type, filter time, overflow, and temperature offset of each channel.

1) General settings

The temperature module settings vary with the module type. The 4TC and 8TC modules support the cold junction compensation function, but 4PT does not. In addition, 8TC supports outer cold junction compensation, but 4TC does not. The following figure shows the configuration interface of 8TC.
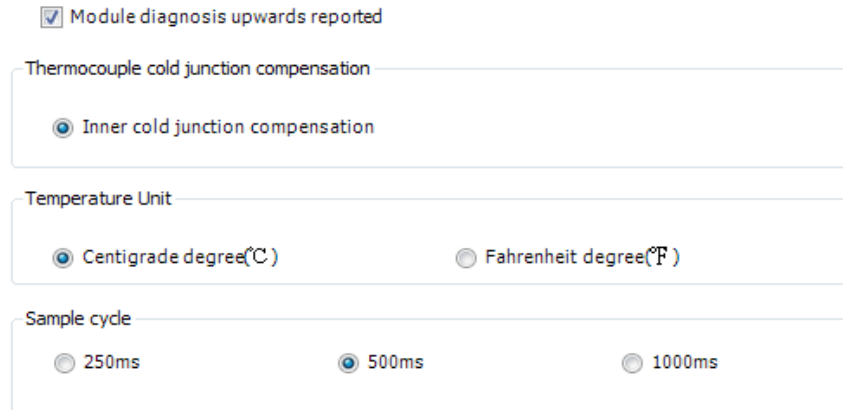
Figure 3-21 8TC general settings dialog box

- Module diagnosis upwards reported: Whether to report the module faults to the parent device (such as CPU and remote module slave station). If the faults are configured to be reported and the parent device is configured with Stopped On Failure, the parent device will stop the running of the device.

- Cold Junction Compensation: Selects the cold junction compensation mode. Only the 8TC supports outer cold junction compensation, and 8TC uses channel 7 (the last channel) for the input of outer cold junction compensation.

- Temperature Unit: Sets the input unit used by temperature module, including Celsius and Fahrenheit.

- Sample Cycle: Sets the sample cycle used by the temperature module, including 250 ms, 500 ms, and 1000 ms.

2) Channel settings

Different types of modules support different numbers of channels. 4TC and 4PT support 4 channels, and 8TC supports 8 channels. The channels have similar parameter settings. The following is the description for one channel. The following figure shows the channel setting interface of 8TC.
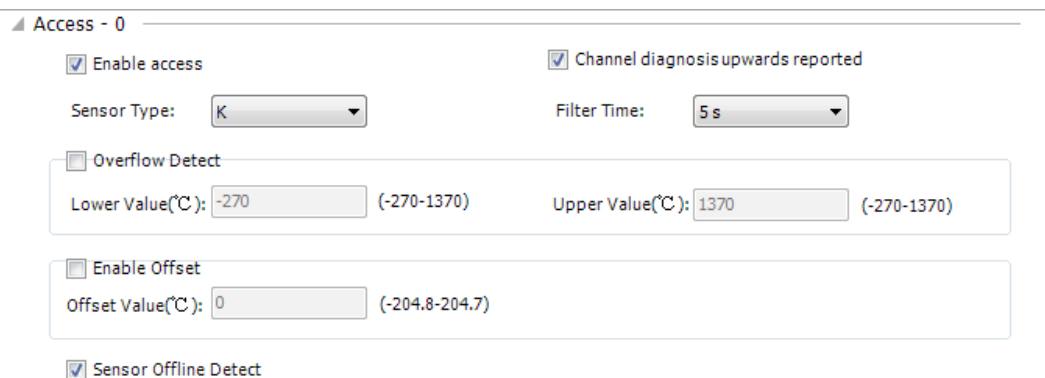
Figure 3-22 Temperature module channel settings dialog box

- Enable access: Whether to activate the channel. The channel is available only after it is activated.

- Channel diagnosis upwards reported: Whether to report the channel faults to the parent device (such as CPU and remote module slave station). If the faults are configured to be reported and the parent device is configured with Stopped On Failure, the parent device will stop the running of the device.

- Default: Restores the default settings of the channel.

■ Sensor Type: The sensor type and specifications of 8TC and 4TC are listed in the following table. By default, the K sensor is used.

Table 3-3 Sensor type and specifications of 8TC and 4TC

Item	Sensor Name	Temperature Range (ºC)	Temperature Range (ºF)
Thermocouple	B	250ºC to 1800ºC	482ºF to 3272ºF
	E	-270ºC to 1000ºC	-454ºF to 1832ºF
	N	-200ºC to 1300ºC	-328ºF to 2372ºF
	J	-210ºC to 1200ºC	-346ºF to 2192ºF
	K	-270ºC to 1372ºC	-454ºF to 2502ºF
	R	-50ºC to 1768ºC	-58ºF to 3214ºF
	S	-50ºC to 1768ºC	-58ºF to 3214ºF
	T	-270ºC to 400ºC	-454ºF to 752ºF

Table 3-4 4PT specifications

Item	Sensor Name	Temperature Range (ºC)	Temperature Range (ºF)
Thermal Resistance	Pt100	-200ºC to 850ºC	-328ºF to 1562ºF
	Pt500	-200ºC to 850ºC	-328ºF to 1562ºF
	Pt1000	-200ºC to 850ºC	-328ºF to 1562ºF
	Cu100	-50ºC to 150ºC	-58ºF to 302ºF

■ Filter Time: Filter time of the temperature module when using this channel, ranging from 0 ms to 100 ms. The default value is 5 ms.

■ Overflow Detect: Enables overflow detection for the channel. If the temperature is beyond the specified range, an overflow fault is reported. For the temperature range, see Table 3-3.

■ Enable Offset: Sets the offset compensation for the temperature module, ranging from -204.8 to 204.7.

■ Sensor Offline Detect: Enables offset alarming of the sensor.

3) I/O mapping

The numbers of included channels and I/O mappings vary with the temperature module type. The following figure shows the I/O mapping interface of 4PT. The parameter value of each channel is the temperature value. For details, *click I/O Mapping.*

Figure 3-23 Temperature I/O mapping dialog box

4)   Information

The temperature module basic information is displayed, including Name, Vendor, Categories, Type, ID, Version, Module ID, Description, Order Number, and Image.

Moreover, after you log in, the Information interface displays the PCB software version and logic software version of the temperature module. The PCB software version is the embedded software version of the temperature module and the logic software version is the FPGA software version within the temperature module.

General

Name: CPU TC4
Vendor: Inovance
Categories:
Type: 40070
ID: 10F4 0070
Version: 01.00.00.00
Order Number: ***
Description: CPU TC4 Module hardware description

Image

No image found
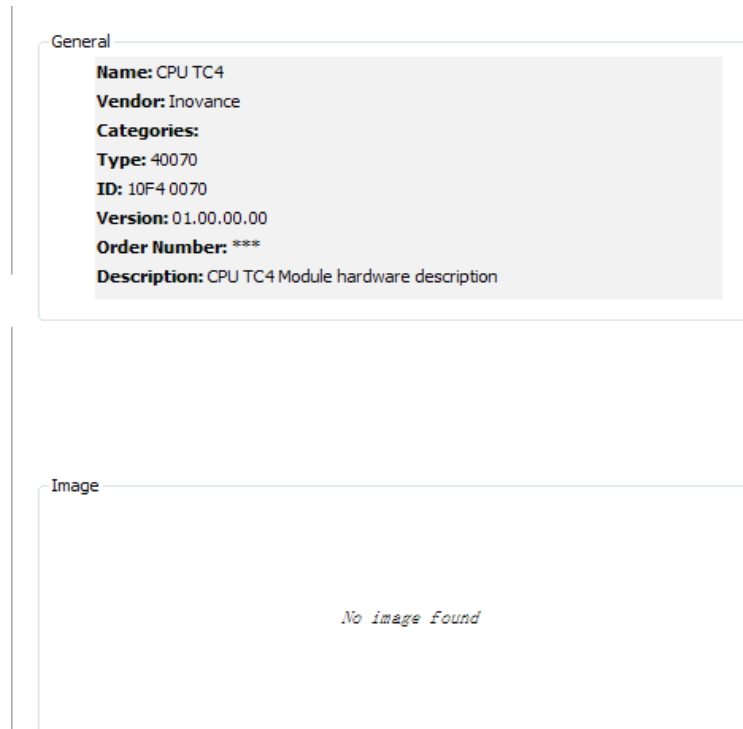
Figure 3-24 Temperature module information interface

3.2.4 High-Speed I/O Configuration

Click the HIGH_SPEED_IO option in the Device interface to display the HSIO parameter settings interface. On this interface, you can configure the high-speed I/O function and its parameters, including:

(1) High-speed counter

(2) High-speed output

(3) High-speed input interrupt

1 High-Speed Counter
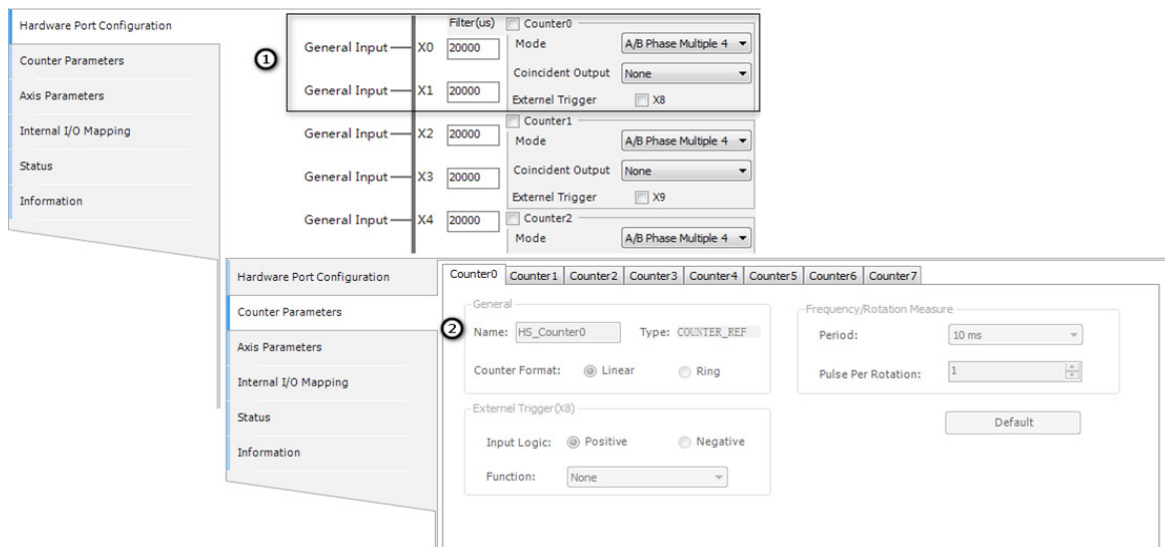
1. The high-speed counter function parameters include **Mode**, **Coincident Output**, and **External Trigger**. The counter mode includes single phase counter, AB phase counter, CW/CCW counter, and internal clock counter.

Taking Counter 0 for example (counter numbers range from 0 to 7), the configuration procedure is as follows:

1) Select Counter 0.

2) Set the high-speed counter mode:

Single phase counter: receives pulse signals from external single-phase encoders. It occupies only the hardware port X0.
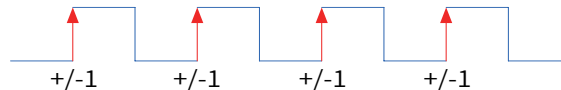
AB phase counter: receives pulse signals from external AB phase. AB can also be set to multiple 4. It occupies hardware ports X0 and X1.

Phase A advance B Increment

Phase B advance A Decrement

Quadruple increment

Quadruple decrement
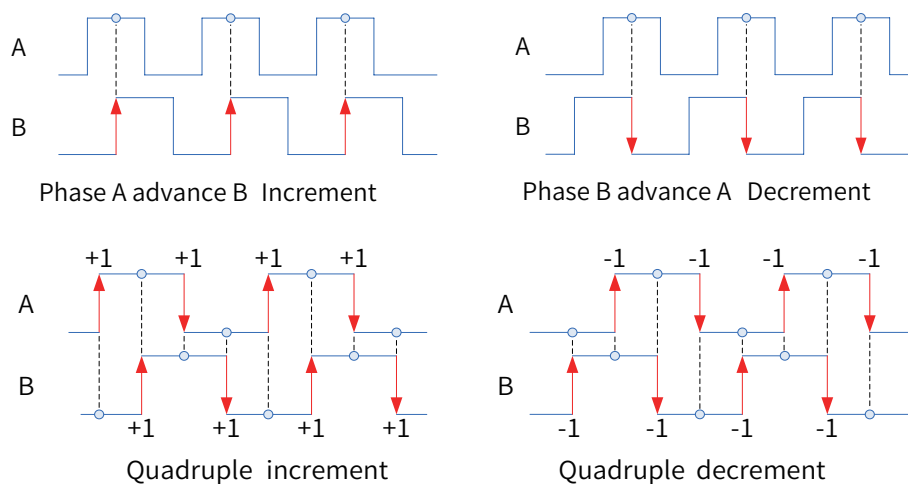
CW/CCW phase counter: receives pulse signals from external CW/CCW phase. It occupies hardware ports X0 and X1.

CW/CCW counter
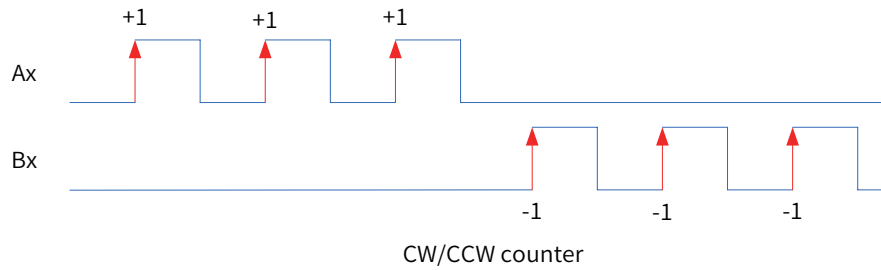
Internal clock: uses the internal pulse signals generated periodically by the software as the pulse signals of counter 0, supporting 1 μs, 10 μs, 100 μs, and 1 ms.

3) Coincident Output:

When the counter of high-speed I/O reaches the specified value, the corresponding hardware output port will output the matching level signals. When this function is enabled, the HC_EnableInterrupt and HC_SetCompare/HC_SetCompareM functions need to be invoked.

4) External Trigger:

When the external trigger pin is enabled, the high-speed counter latch and pulse width measurement functions are available. The counter latch function needs to invoke HC_TouchProbe and the pulse width measurement function needs to invoke HC_MeasurePulseWidth.

5) Filter Time:

Set the filter time of the high-speed counter interface. The default value is 2 μs.

2. Instantiation high-speed counter, with the data type COUNTER_REF

Take counter 0 as an example:

1) Default name of counter 0 instantiation: HS_Counter0

2) Counter mode:

■ Linear counter: Count between the maximum and minimum values. The counter stops when the count up reaches the maximum or the count down reaches the minimum, and the overflow sign takes effect.
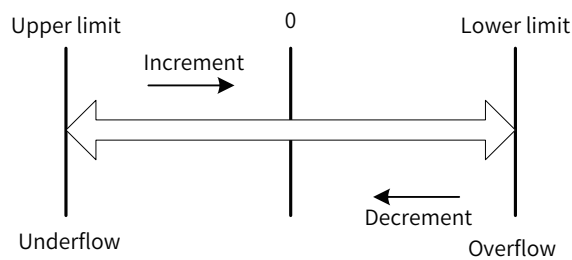
■ Ring count: Used together with HC_SetRing.

Count between the maximum and minimum values. When count up exceeds the maximum, the value skips to the minimum. When count down exceeds the minimum, the value skips to the maximum.
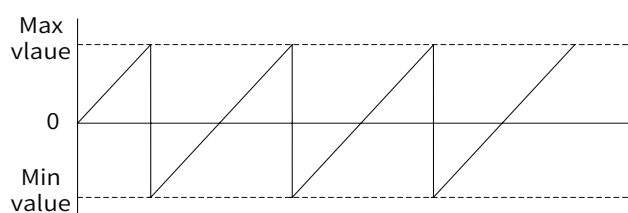
3)   External Trigger:
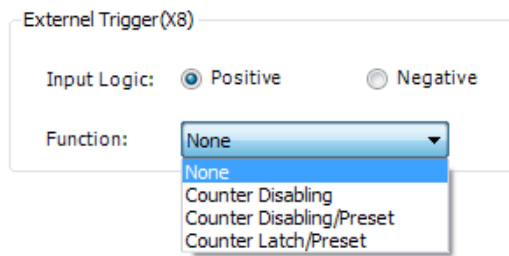
When the input level of X8 is effective, you can set the counter function. For example, configure X8 as the signal of disabling counter 0. You can also disable the preset and counter latch functions of counter 0.

2 High-Speed Pulse Output

1. Configure the high-speed pulse output function, including output pulse mode (pulse + direction, CW/CCW) and home mode.

Take axis 0 as an example (axis numbers range from 0 to 3):

1)   Check Axis 0.

2)   Configure the high-speed pulse output mode:

Pulse Command Format	Pulse + direction	
	Forward	Inverse
Positive	PULS ⎍⎍⎍ SIGN _/‾	PULS ⎍⎍⎍ SIGN ‾_
Negative	PULS ⎍⎍⎍⎍ SIGN ‾_	PULS ⎍⎍⎍⎍ SIGN _/‾

Pulse Command Format	CW/CCW	
	Forward	Inverse
Positive	PULS ‾_ SIGN _⎍⎍⎍	PULS _⎍⎍⎍ SIGN ‾_
	PULS _⎍‾ SIGN ‾_⎍⎍⎍	PULS _⎍⎍⎍⎍ SIGN _/‾

3) Home method

Methods 0 to 3 are supported. For details, see the home diagram.

2. Axis instantiation, with data type HS_AXIS_REF

Take axis 0 as an example:

■ Default name of axis 0 instantiation: HS_Axis0

■ Stroke Limit: soft limit

■ Speed Limit: limits the maximum speed

■ Bias Speed: baseline speed when pulse is started

■ Acc Method: trapezoid and S curve

3. Home parameter settings, including home speed and creep speed

Used together with the MC_Home_P function.

4. Home method diagram

3 High-Speed Input Interrupt
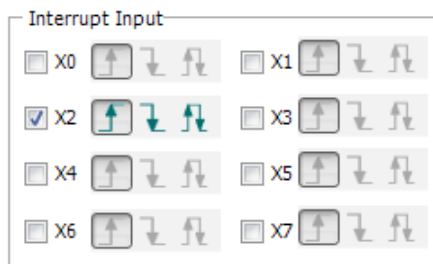
For example:

1) Select high-speed input interrupt X2.

Select X2 edge interrupt: raising edge, falling edge, and both raising and falling edges.

2) Configure interrupt task:
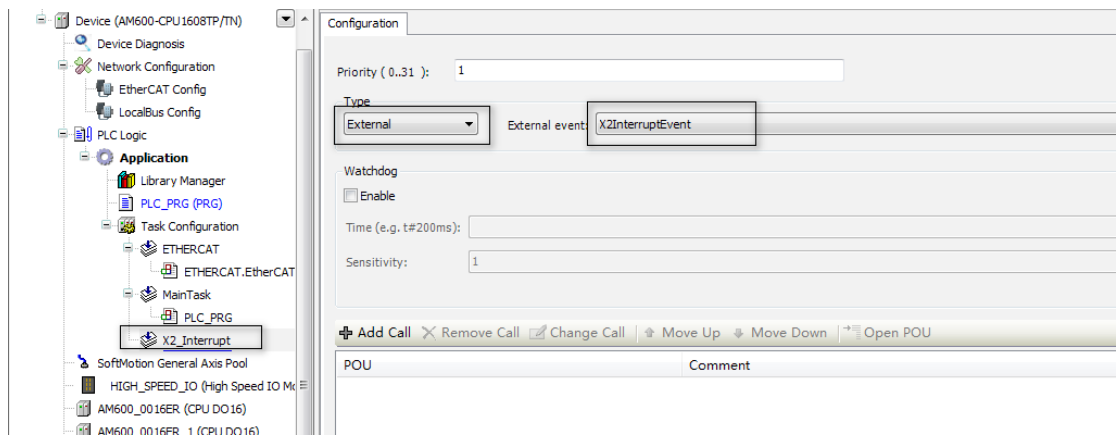
Launch the HC_EnableInterrupt function to add the X2 interrupt task. When the raising edge signal of X2 takes effect, the program in the interrupt task starts to run.

4 High-Speed I/O Library
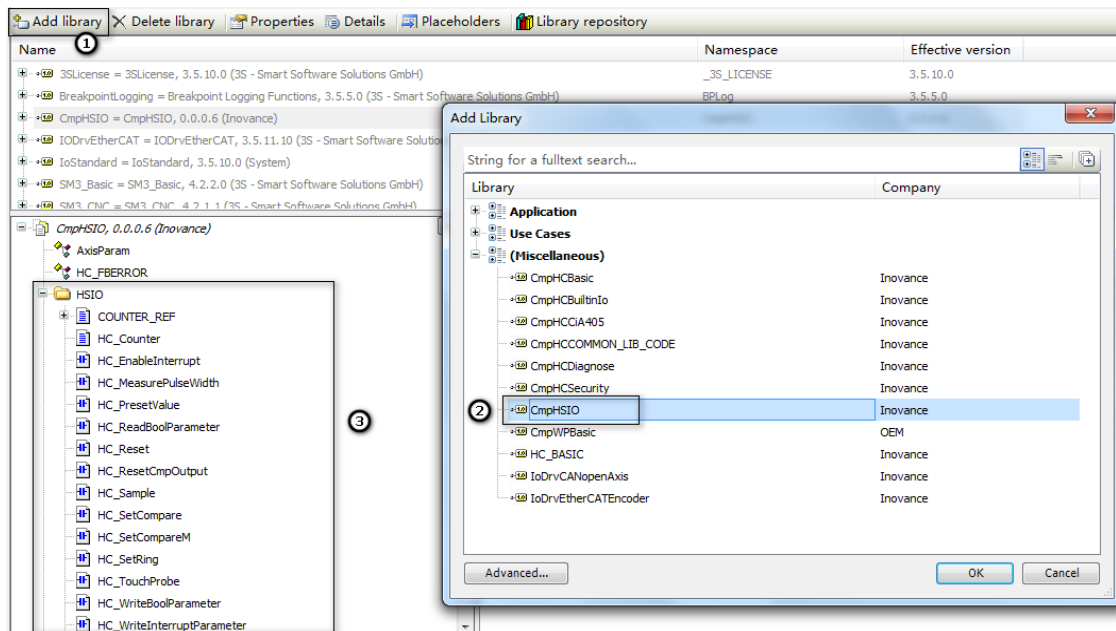
1 - Add a library in the library manager; 2 - Add a new high-speed I/O library; 3 - The functions of the new high-speed I/O library are displayed.

3.3 EtherCAT Configuration

3.3.1 Overview

EtherCAT is an open industrial field technology over the Ethernet. It features short communication update interval, low synchronization jitter, and low hardware cost. EtherCAT supports the linear, tree, start, and hybrid topologies. EtherCAT slave stations must use dedicated communication chipset ESC, and EtherCAT master stations can use a standard Ethernet controller.

For more information about EtherCAT working mechanism and technologies, see EtherCAT - Industrial Ethernet Fieldbus and Its Drive Design or log in to the official website of EtherCAT Technical Committee at https://www.EtherCAT.org.

3.3.2 Common Functions

1 Installing Device

EtherCAT device installation is to import the device description file (with file name extension .XML) in compliance with ETG (EtherCAT Technical Committee) standards into the programming software InoProShop. After the software parses and processes the file, it generates the EtherCAT config devices that can be added and deleted by users. The InoProShop integrates all EtherCAT slave stations of Inovance, and you do not need to install them. If you need to use third-party EtherCAT devices, install the device description files provided by the third-party vendors.

Two installation methods are available: installation on the Network Configuration interface and installation by using menu tools. The procedures are as follows:

■   Installation on the Network Configuration interface

1)   Click **Import ECT file**. The following dialog box is displayed:
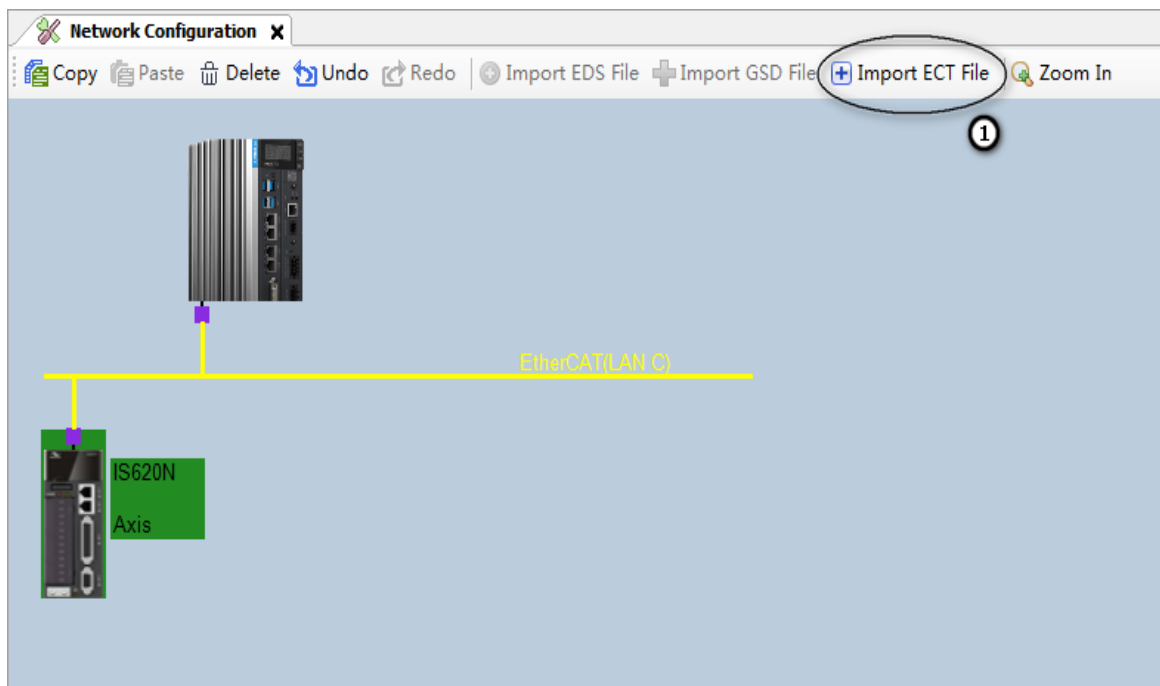
2) Select an XML file and click **Open**.

■ Installation by using menu tools

1) Choose **Tools** > **Device Repository**.
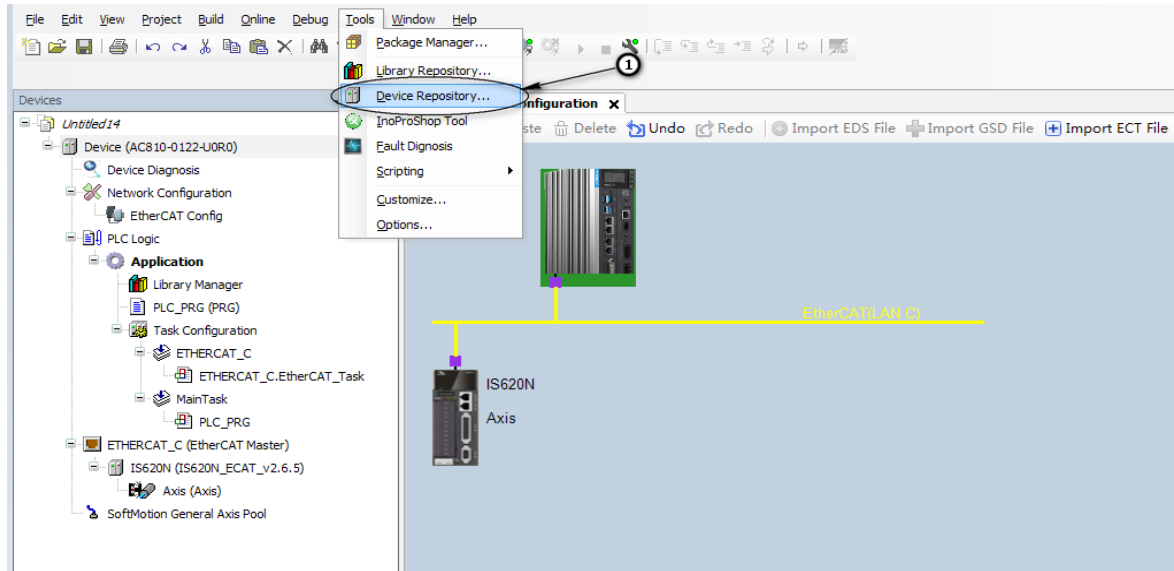
2) In the pop-up dialog box, click **Install...**.
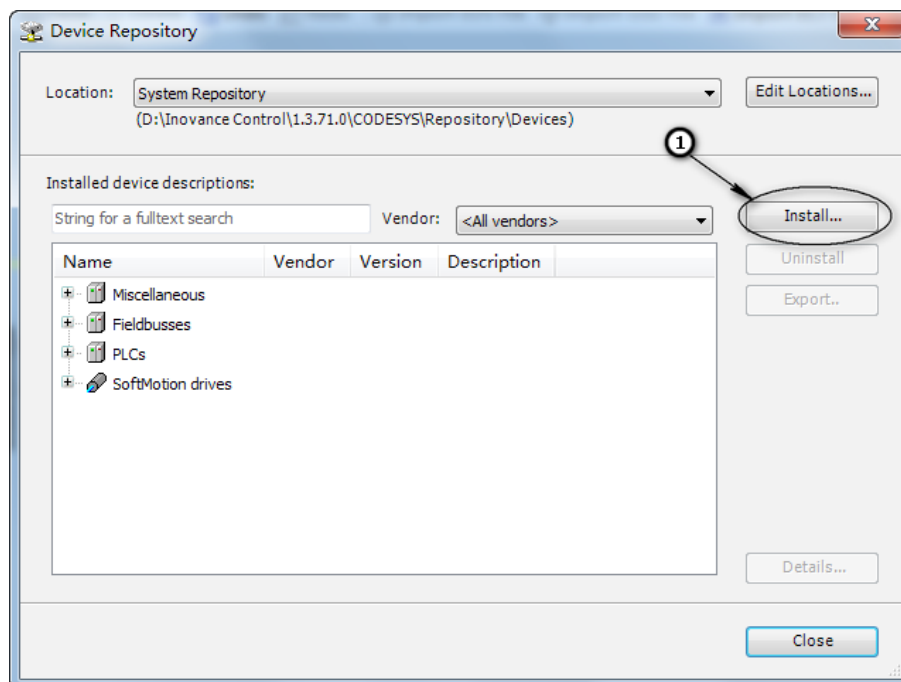
3) In the pop-up **Installed device descriptions** dialog box, select a device description file in EtherCAT XML format. Select a slave station device description file saved locally and open its XML file.

2 Scanning Device

The scanning function is recommended. The procedure is hot reset -> logout -> device scanning.

■ Preparations

The prerequisites of using device scanning commands are as follows:

1) PC is properly connected to the PLC through a gateway, shown as follows:
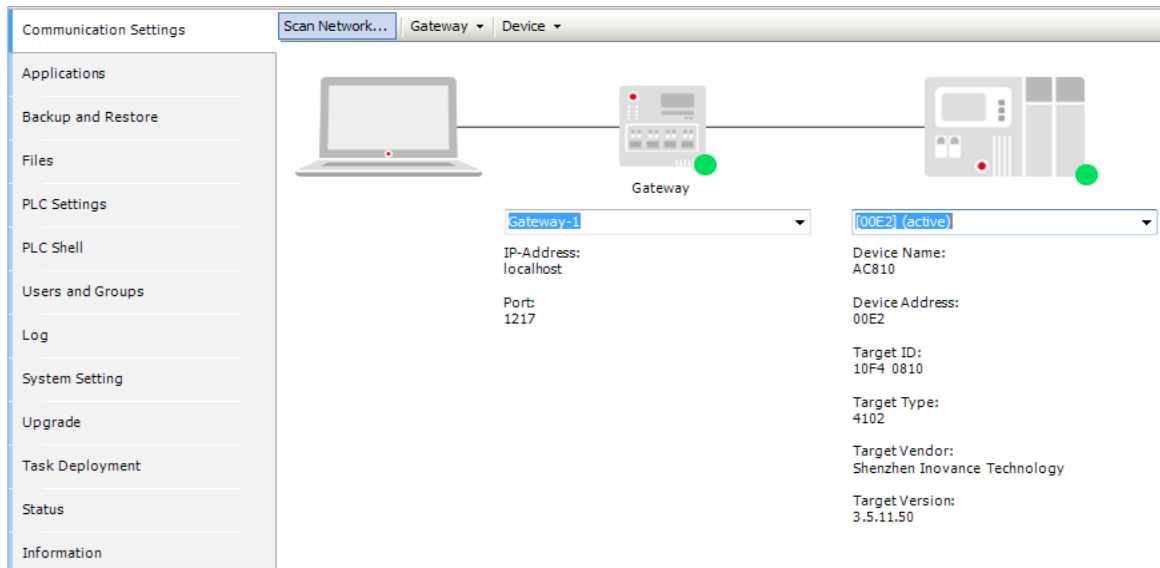
2) The PLC is networked with the slave station.

3) The port configuration of PC is consistent with that of PLC, shown as follows:
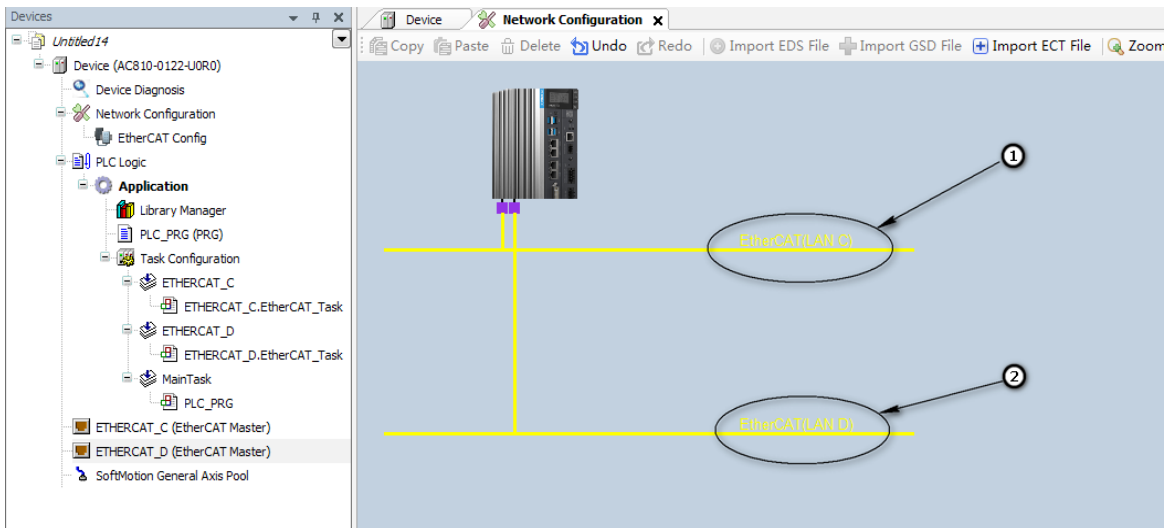
The EtherCAT of PC includes the EtherCAT_C(1) and EtherCAT_D(2), so the PLC must also have the EtherCAT_C(1) and EtherCAT_D(2). To ensure port consistency, download the port configuration before using the scanning command.

■ Scanning Operations

1) In normal situations, the following device scanning interface is displayed when you scan the device.
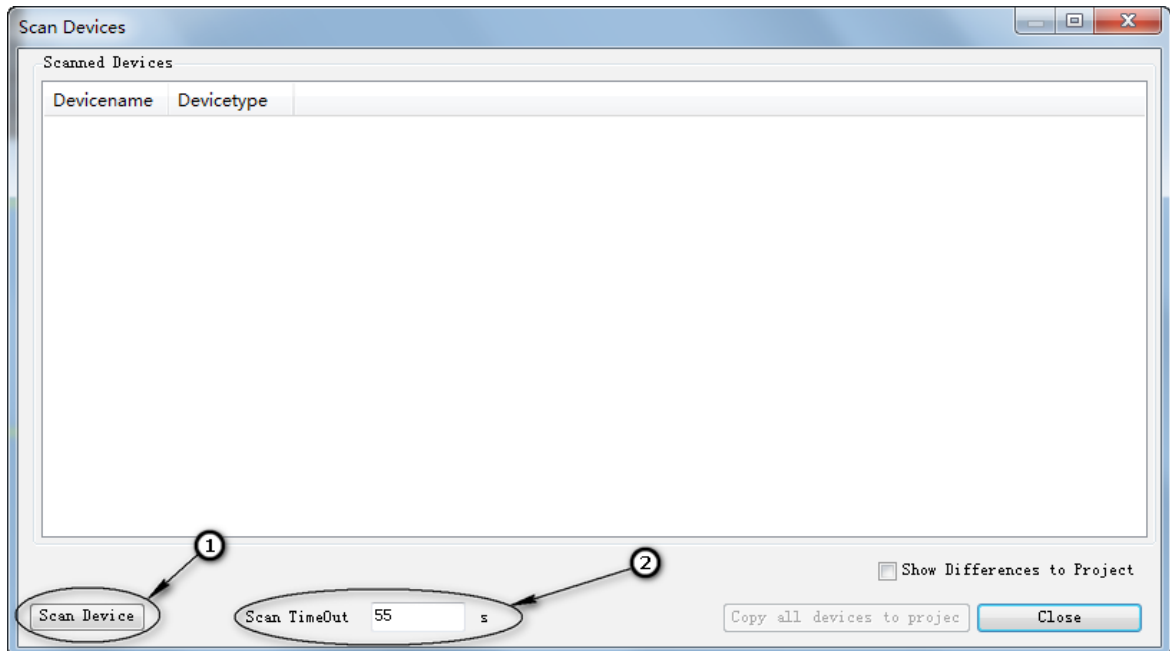
2) The options in the figure are as follows:

No.	Option	Description
1	Scan Device	Scans the device.
2	Scan TimeOut	Indicates the maximum timeout interval of each scan operation. If no result is detected, the timeout interval can be prolonged. By default, the minimum value is 20s.

■ **Operations on Scanning Result**

In normal situations, the scanning results are as follows:

1) Copy all devices
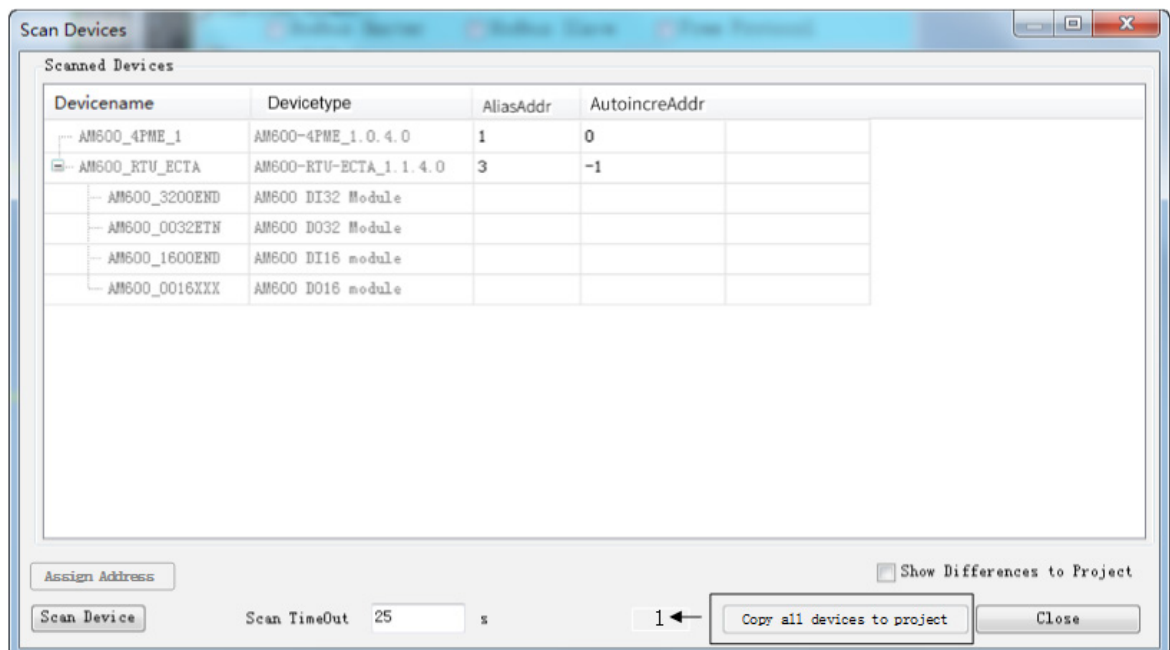
If you choose to copy all devices to the project, the scanning result is added to the device tree and configuration.

2) Copy some devices

If you need to copy only some of the devices, press Ctrl+, or press Shift+ and click the desired devices. In this situation, the copy all devices button is as follows:
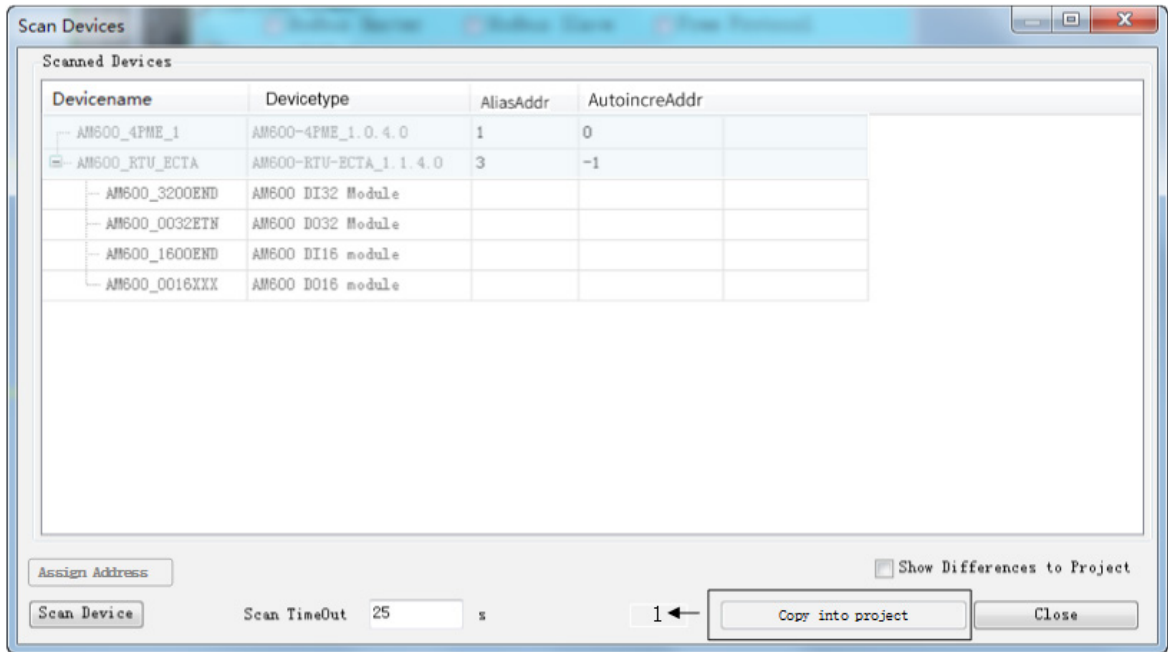
Then you click **Copy into project** to complete the device copy operation.

3) Allocate alias address

In the first row of the **AutoincreAddr** column, you can double-click an alias address to edit it. After the modification, the text color is changed, shown as follows:
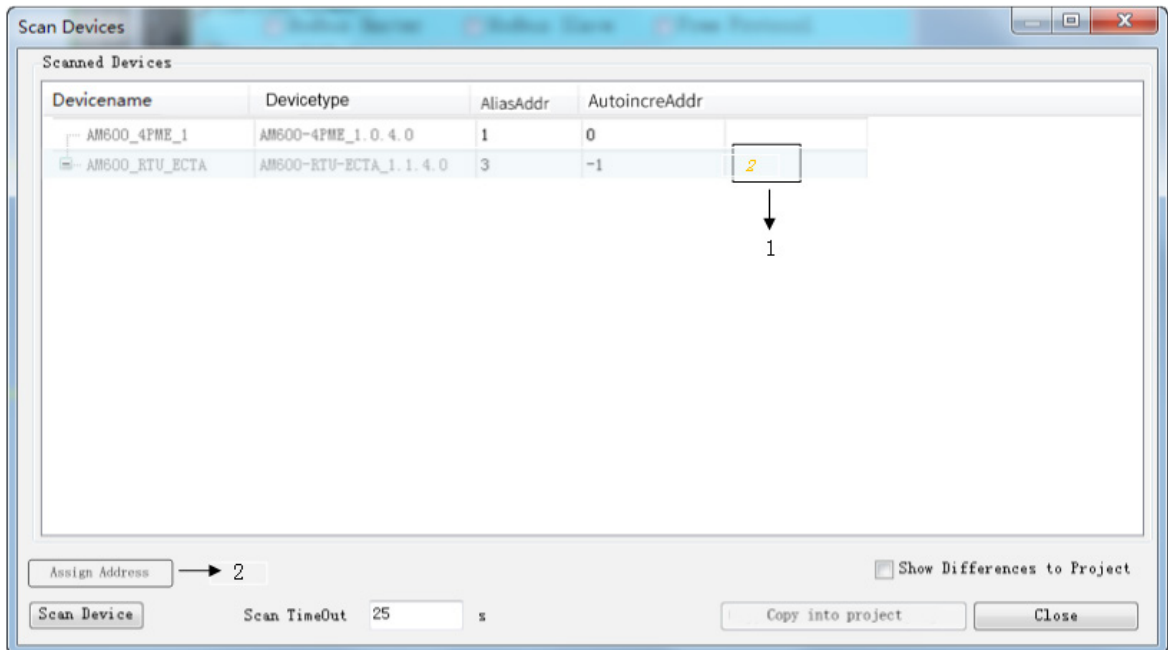
After editing the alias address, click the **Assign Address** button in 2 to make the alias address take effect.

If the modification is successful, the system displays the following prompt:
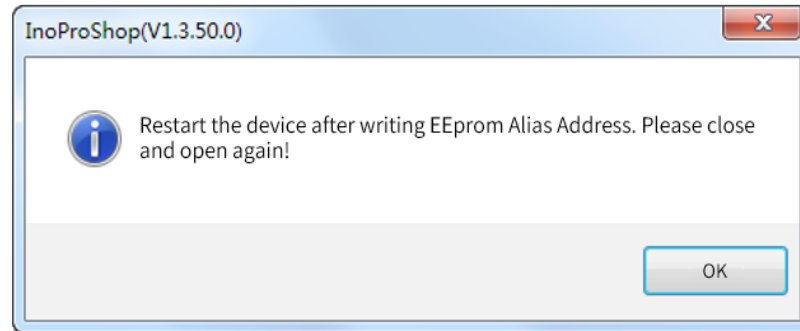
If the modification is failed, the system displays the following prompt:
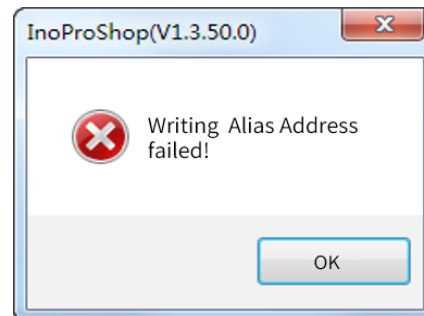
4)　Display the project differences

Display the differences between the configuration device and scanned device, shown as follows:
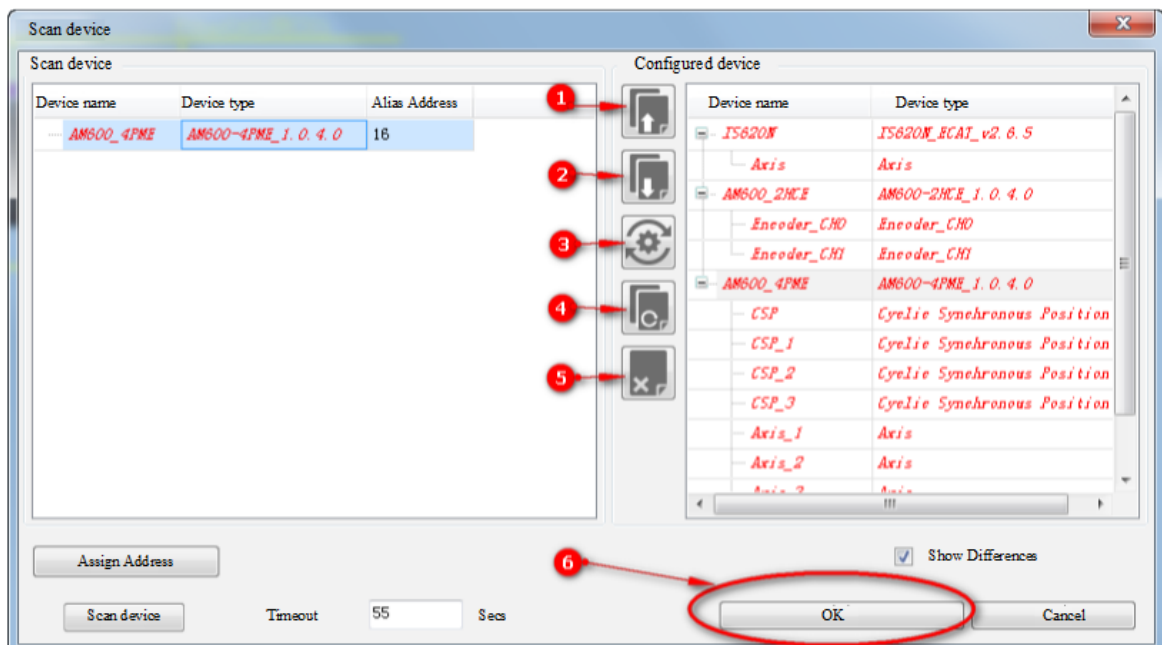
The numbers in the figure are described as follows:

No.	Name	Description
1	Before copy	Select the scanned device on the left and the current device on the right, and select 1. The scanned device is inserted in front of the selected device on the right. This command can also be used between modules.
2	After copy	Select the scanned device on the left and the current device on the right, and select 2. The scanned device is inserted behind the selected device on the right. This command can also be used between modules.
3	Changed to	Select the scanned device on the left and the current device on the right (the left device and the right device must be of the same type), and select 3. The right device is replaced with the scanned device.

No.	Name	Description
4	Copy all	All the devices on the right are cleared, and all the scanned devices are copied to the right side.
5	Delete	Select a device on the right to delete it. This command can also be used on modules.
6	Confirm	Confirm the preceding operations, and copy the devices to the device tree and configuration.

■ **Scanning Abnormalities**

1) The PC is not connected to the PLC. The following information is displayed:
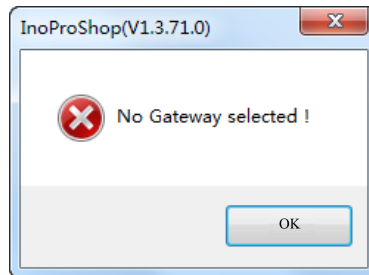
2) The PC is connected to the PLC, but the PLC is not connected to the slave station. The following information is displayed when the slave station cannot be detected:
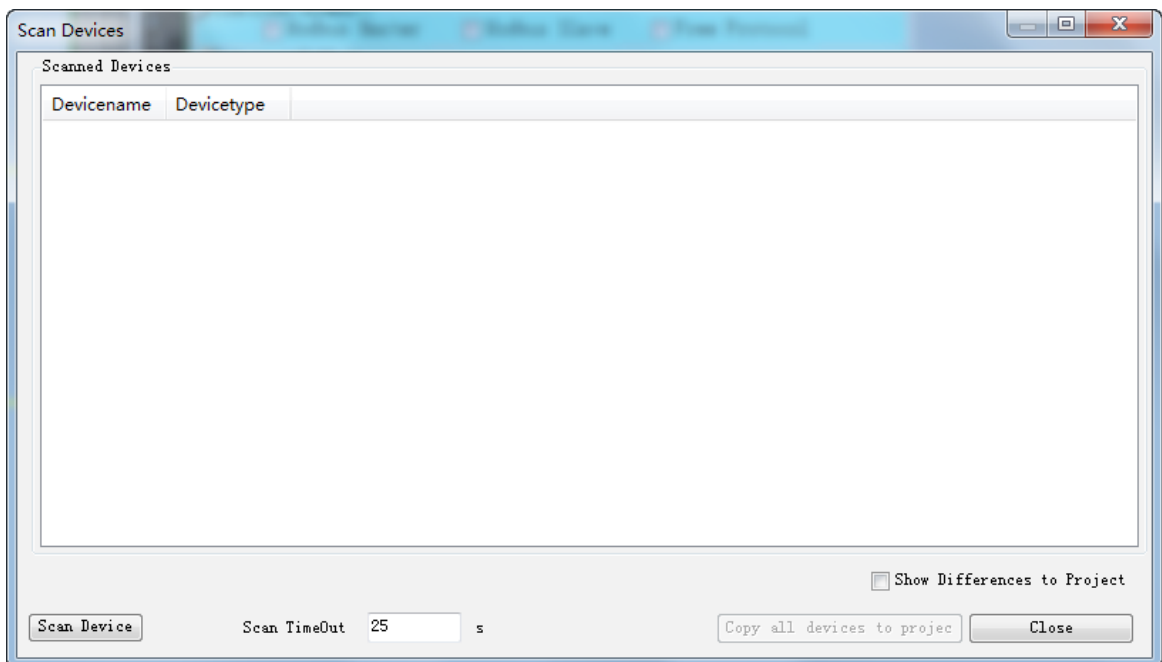
3) The port information in programming software configuration is different from the PLC information. The following information is displayed:
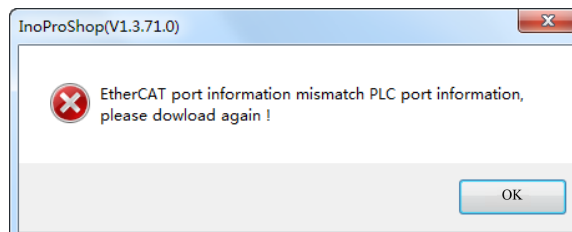
In this situation, you need to download the port information in the programming software again.
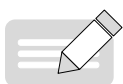
NOTE As shown in the following figure, EtherCAT_C is selected in configuration, but the actual MAC address is changed to EtherCAT_D. The EtherCAT_D port information may be displayed when EtherCAT_C is scanned.
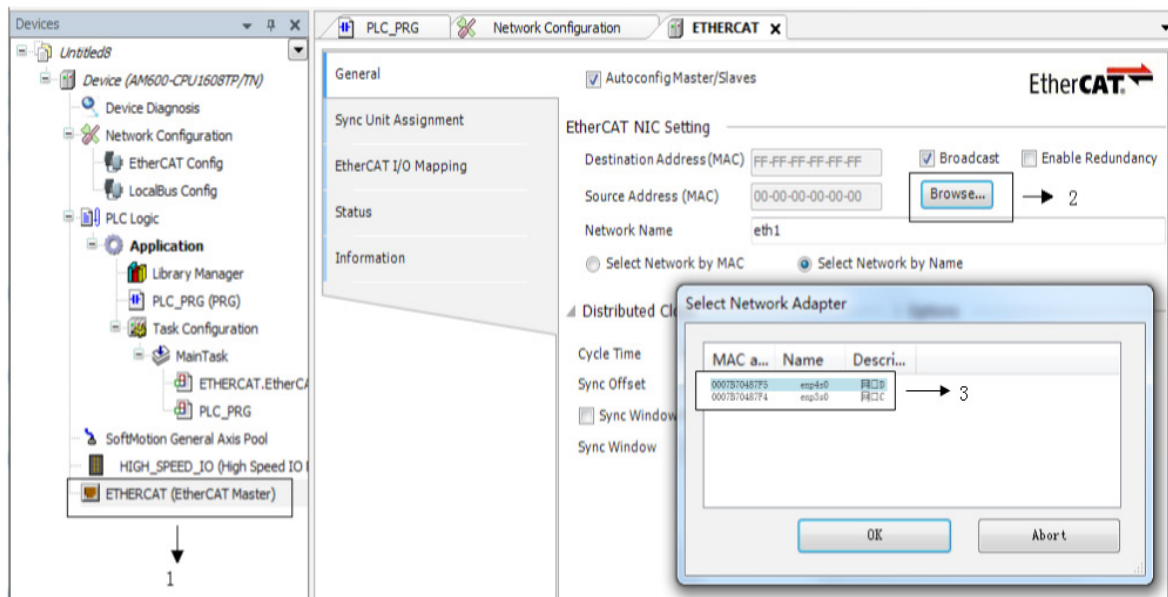
If you log out after the EtherCAT bus starts and perform the device scanning operation, the scanning operation may fail or return an inaccurate result. The reason is that reading the slave station information (for example, object directory 0xF050) through SDO communication is timed out, so the scanning result is affected. (SDO communication is asynchronous, so CPU loading, bus cycle period, and user program execution time will affect the SDO communication.)

3 Updating Device

If the master station version does not match or needs to be upgraded, you can run the device update command.

Procedure: Right-click the EtherCAT master station, and choose **Update Device** from the shortcut menu. The following dialog box is displayed. Click **Display all versions (for experts only)**, select the desired version, and click **Update Device**.
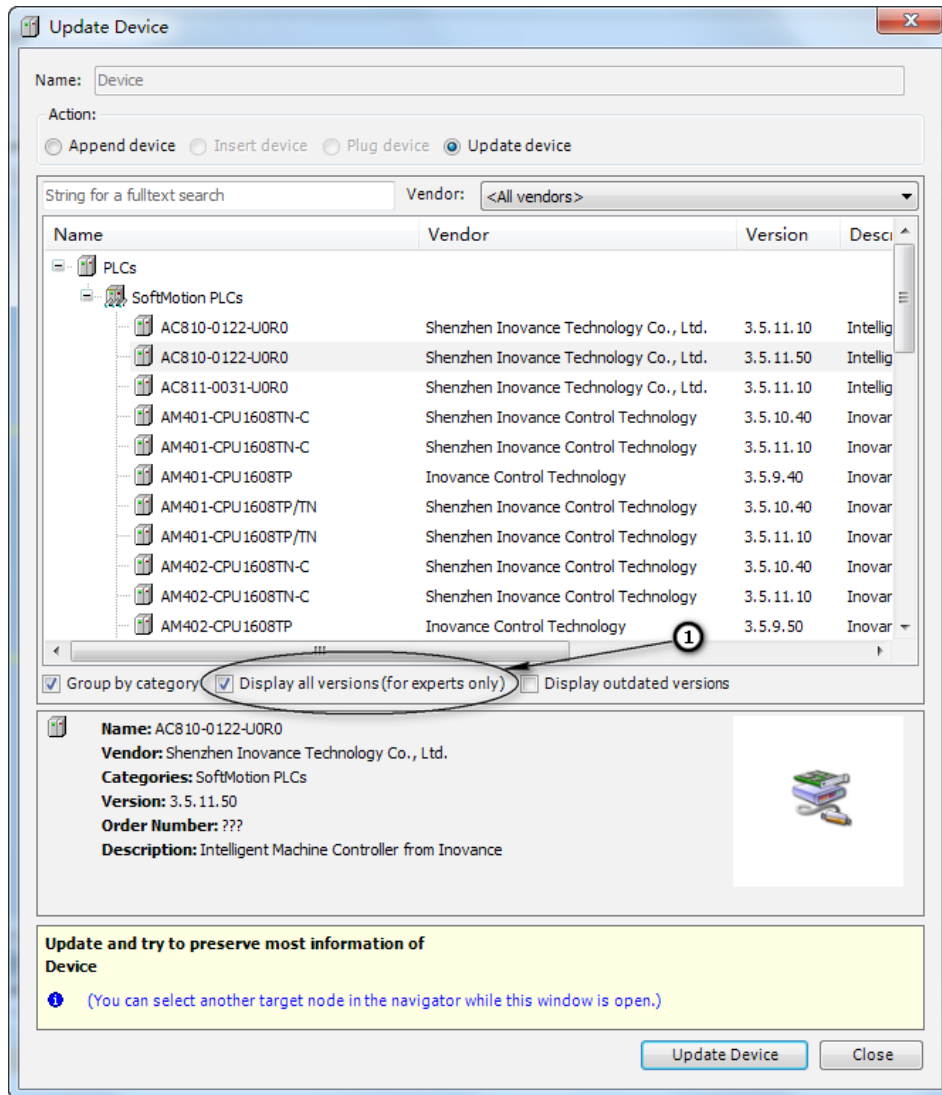
4 Editing I/O Mappings

When you choose to edit I/O mapping, the following interface is displayed:
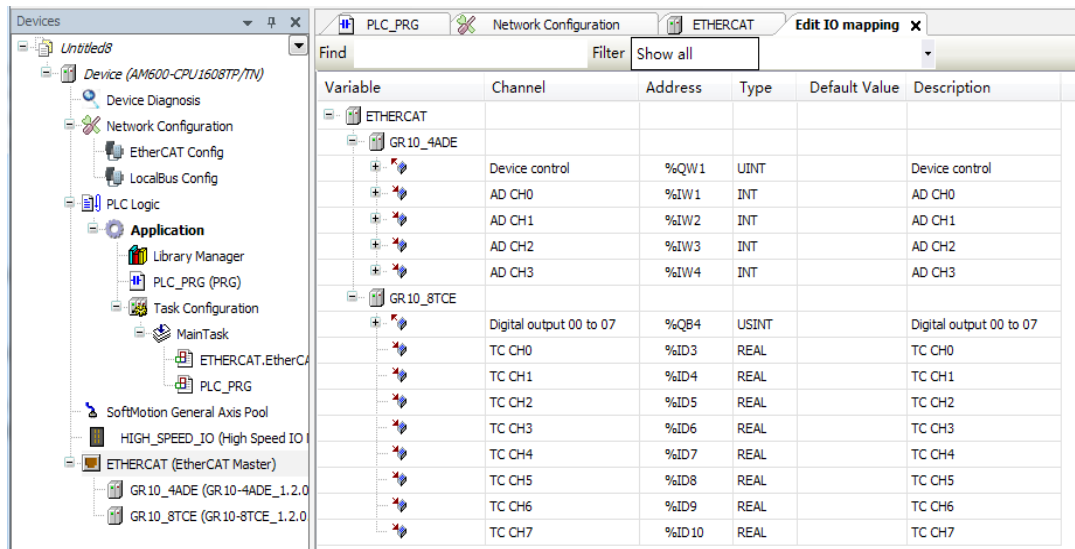
Select a proper filter to filter out unneeded options.

5 Bus Tasks

All IEC tasks of PLC are scanned and executed cyclically strictly based on the same logical sequence. The logical sequence includes four steps: input update (1), IEC task execution (2), output update (3), bus cycle execution (4), shown as follows:
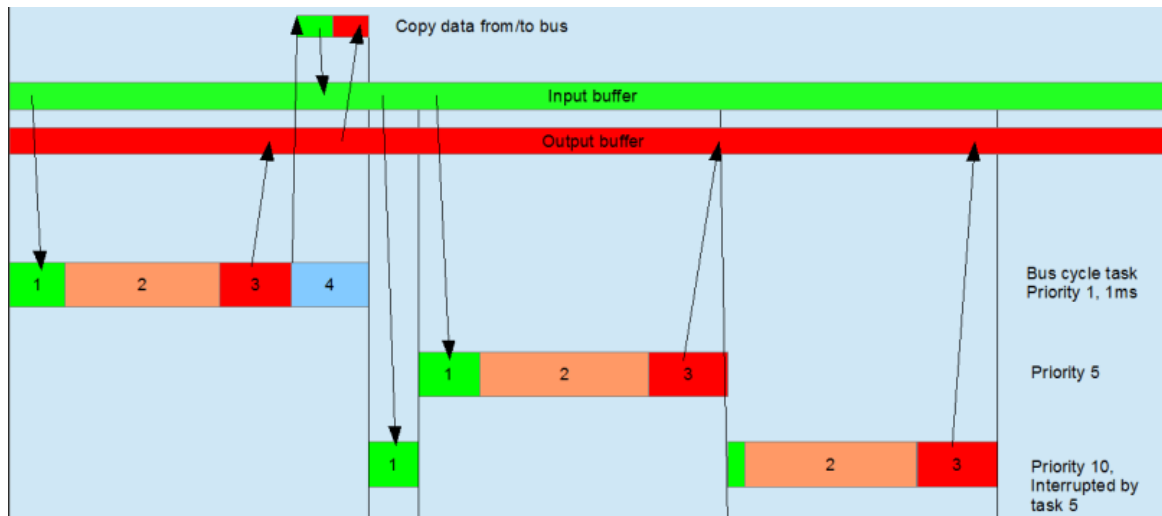
Figure 3-25 Bus cyclic tasks

Each step is described as follows:

No.	Step	Color	Description
1	Input update	Green	Before the IEC task starts, data is read from the bus input buffer, and copied to the task-related input variable.
2	IEC task execution	Orange	Scan and execute the POU under the bus task.
3	Output update	Red	Before the IEC task is finished, the bus-related output variables in the task are copied to the bus output buffer.
4	Bus cyclic	Blue	It is the bus communication execution program implemented by bottom-layer I/O drive. It includes two functions: (1) Transfer the data from bus output buffer to the receive buffer of the remote slave station. (2) Transfer the data in the send buffer of the remote slave station to the bus input buffer.

◆ Warning!

1) If an output variable is used by multiple tasks, the variable value is uncertain (the output variable value may be changed or overwritten by other tasks).

2) If a task is interrupted by another task with a higher priority, the high-priority task reads data from the input buffer, and synchronizes the data to the input variable of the current task. Therefore, the input variables within a scan period may be different. To avoid this problem, copy the input variable value before the task starts so that the task will invoke the copied input variable.

NOTE

Special Bus Cycle Action of EtherCAT

The bus data in the last period is copied before IEC input.
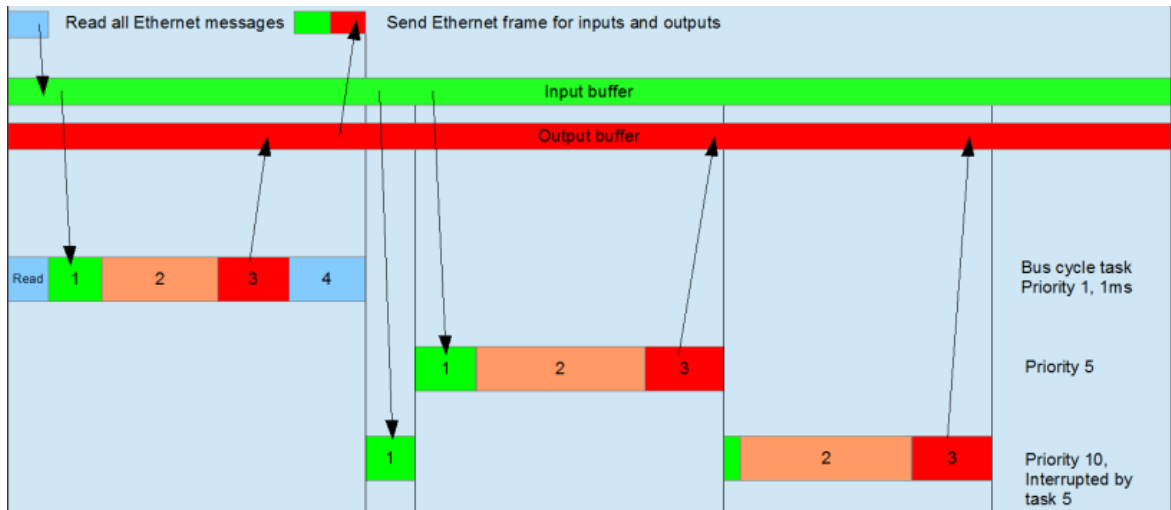
Figure 3-26   EtherCAT bus cycle table

The **Enable each task message** option in EtherCAT Master Station Configuration is available. After this option is activated, the additional information of each task will be sent to the device. In this situation, bus communication can be executed under multiple tasks, to reduce the bus load.

EtherCAT bus cycle table when the **Enable each task message** option is activated
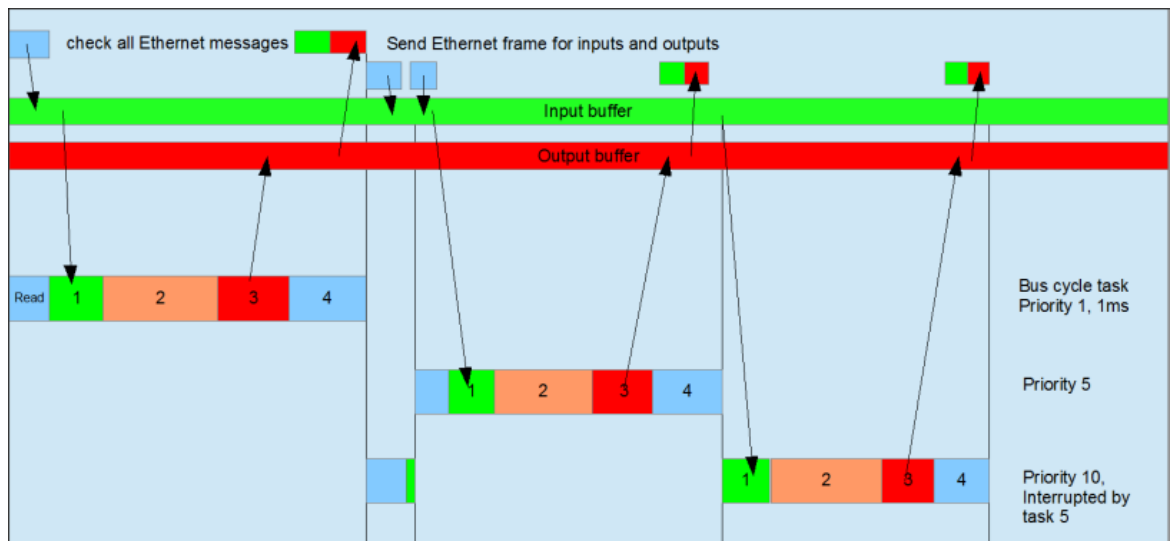
NOTE

◆ After the EtherCAT master station is automatically inserted, the EtherCAT_*** task is also inserted to the current task configuration.

◆ The bus cycle task of EtherCAT master station must be executed in the same task as the EtherCAT_***.EtherCAT_Task.

◆ The input and output of EtherCAT master station are executed in the same task as EtherCAT_***.EtherCAT_Task. Therefore, it is recommended to execute the device control program (such as PLCOpen axis control command) under this task.
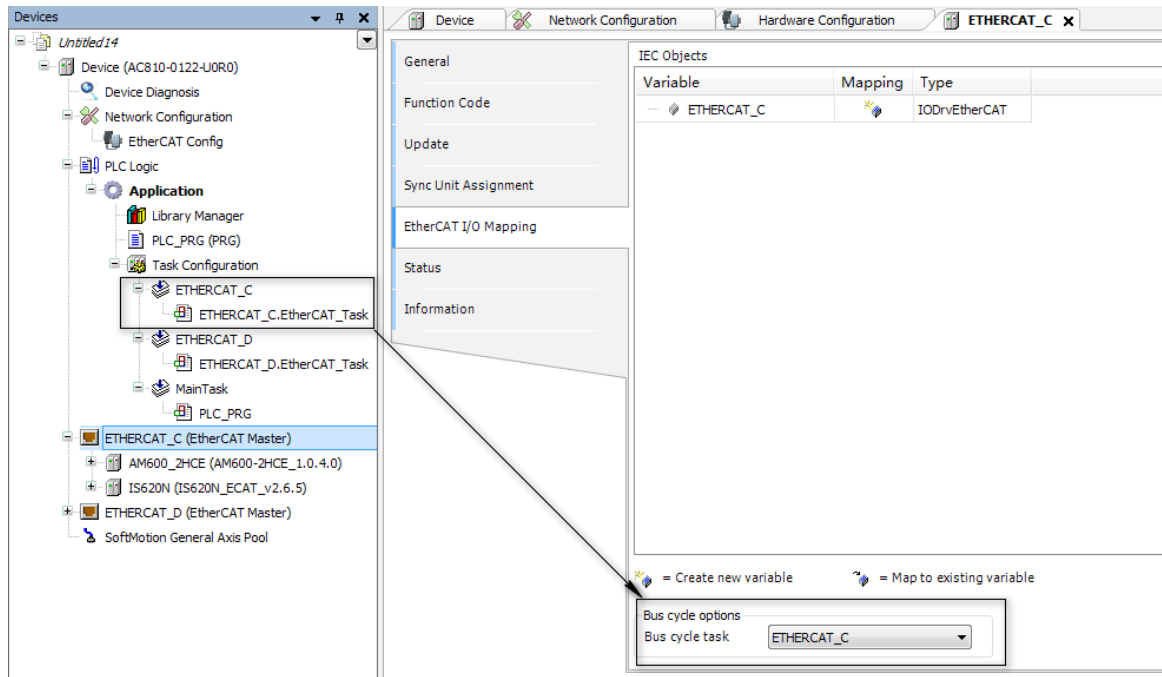
3.3.3 EtherCAT Master Station

1 General Settings

The EtherCAT master station configuration dialog box provides the main settings of the master station.
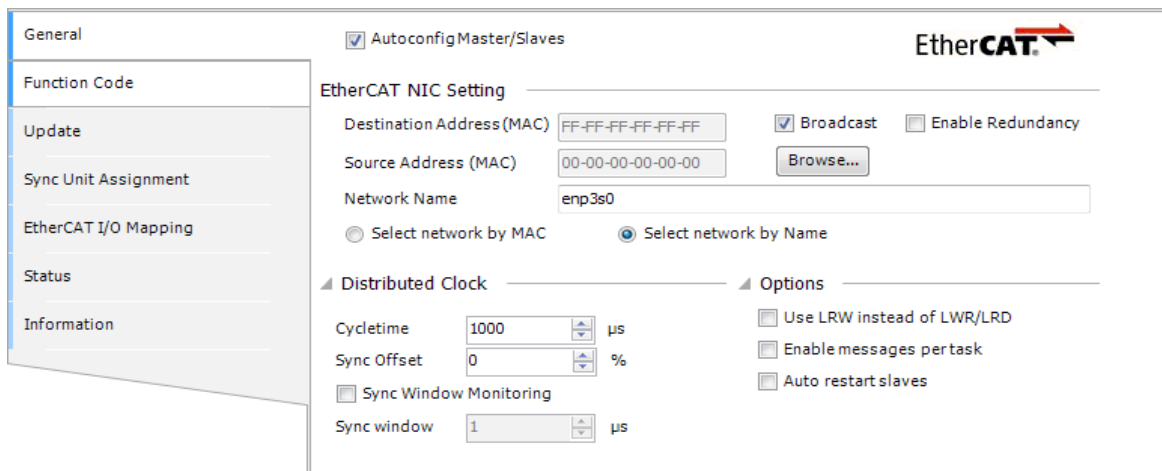
Autoconfig Master/Slave

If you click this option, the main settings of master and slave stations will be automatically completed. In this situation, all the editors of slave stations will not display the FMMU/Sync tab.

	◆ Prompt
NOTE	1) The automatic settings are default settings, which are strongly recommended to standard applications.
	2) If this option is not selected, you must manually complete all settings of master and slave stations. Therefore, you must have professional skills.
	3) For the communication settings between slave stations, do not select auto configuration.

EtherCAT NIC Setting

■ Destination Address (MAC)

MAC address of the EtherCAT network member that receives packages. If **Broadcast** is selected, use a broadcast address (FF FF FF FF FF FF).

■ Enable Redundancy

If ring topology is selected, the redundancy option needs to be enabled. When this option is enabled, if a single point failure occurs in network connection, the EtherCAT network can still work normally. After this option is enabled, you also need to define the second EtherCAT NIC.
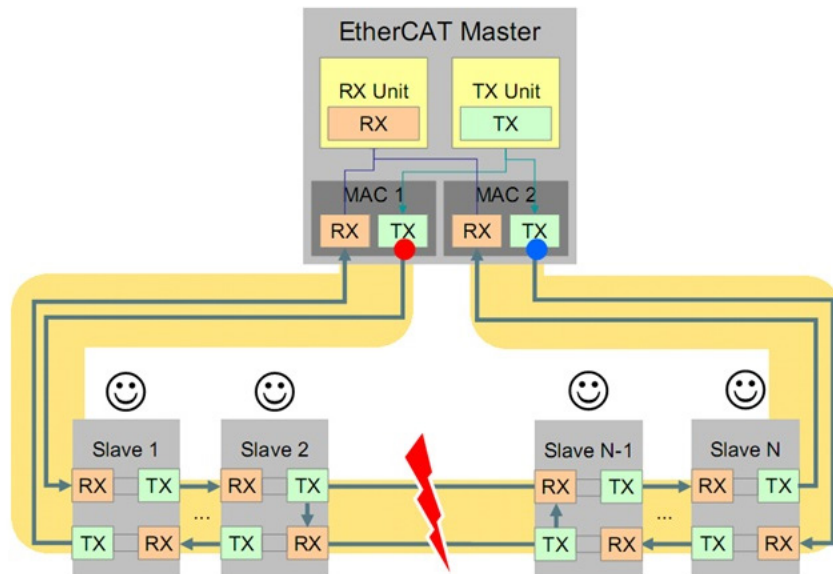
Figure 3-27 EtherCAT ring topology (redundancy)

■ Source Address (MAC)

PLC's MAC address.

■ Network Name

NIC name, including the following options:

■ Select network by MAC/Select network by Name

Each EtherCAT NIC has a unique MAC address. Therefore, if you click **Select network by MAC**, this project cannot be used on other devices.

If you need the project to be independent of device, click **Select network by Name**. In each option, you can click Browse... to display the MAC addresses of available target devices and their names, shown as follows:
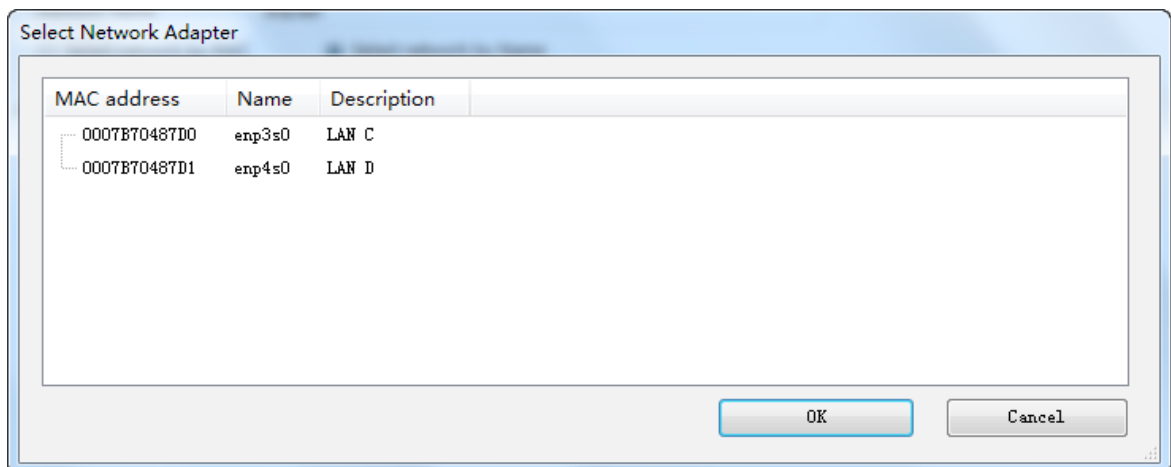
■ Redundant EtherCAT NIC Setting

If the **Enable Redundancy** option is clicked, this setting is available. You can set the options of redundant EtherCAT NIC.

Distributed Clock

■ Cycle time [μs]

Execution cycle time of EtherCAT master station function. It must be the same as the cycle time of the IEC task bound to the EtherCAT master station. If the distributed clock function of the slave station is enabled, the cycle time is synchronized with the **Distributed Clock** settings in the slave station editor.

■ Sync Offset [%]

Ratio of the cycle time of the EtherCAT master station's IEC task (PLC task) to the reference distributed clock (generally, SYNC0 interrupt), ranging from -50% to +50%, with the default value 0%.

Sync offset [%] = (SYNC0 interrupt time - PLC task cycle start time)/ PLC task cycle time.
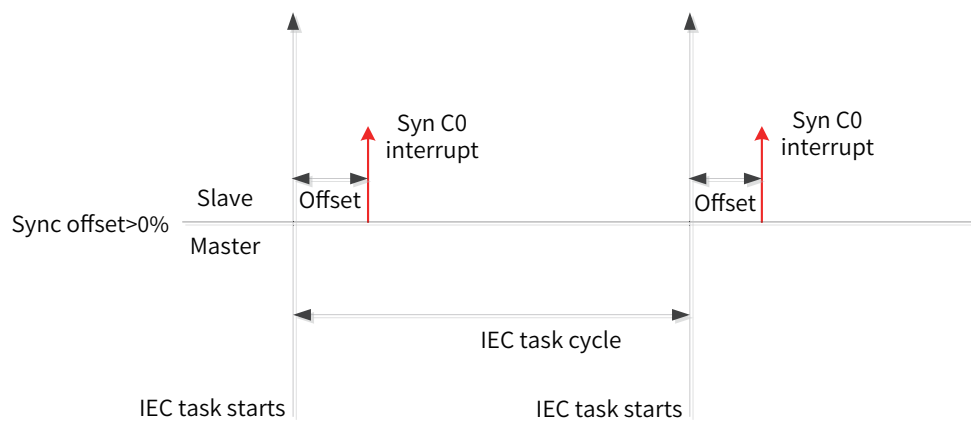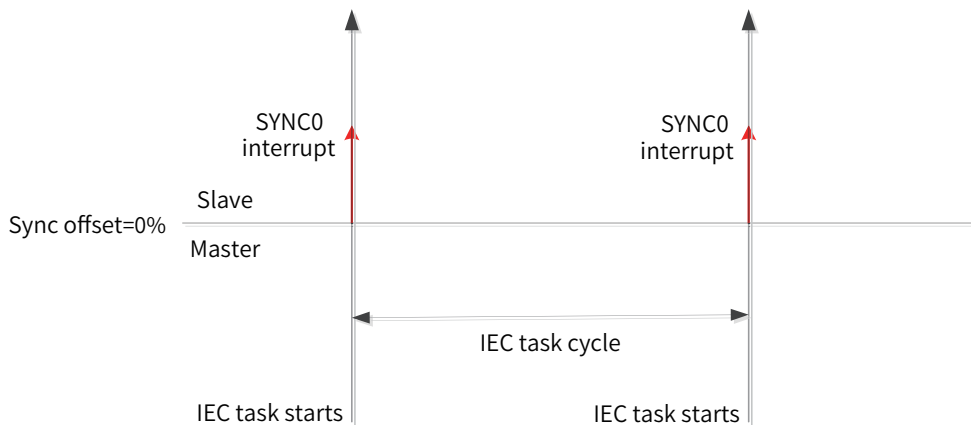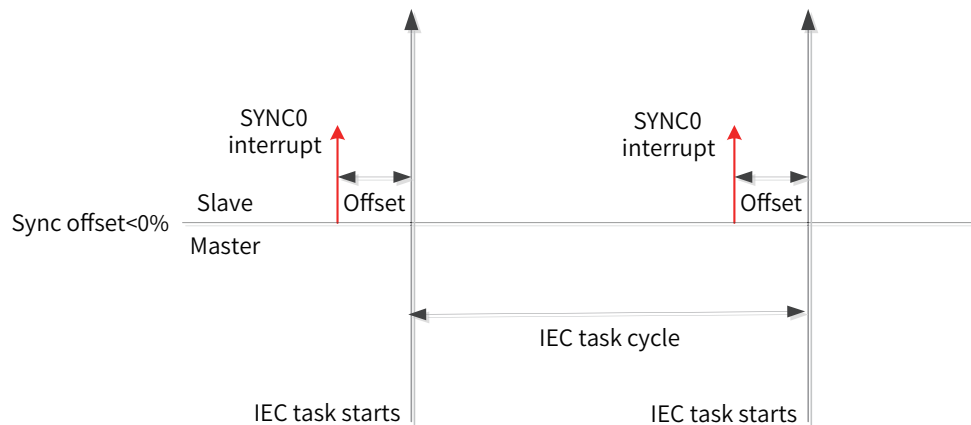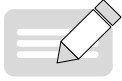
> **NOTE**
>
> ◆ By default, the PLC task cycle time is the same as the distributed clock cycle time of the slave station.
>
> ◆ In actual settings, consider the clock jitter of the controller master station (system instantaneity), PLC task execution time, PLC task cycle time, and number of slave stations.

■ Sync Window Monitoring

After this option is enabled, synchronization of the slave station is monitored.

■ Sync Window

Synchronization window monitoring time. If the synchronization information of all slave stations is included in this window, the xSyncInWindow (IoDrvEtherCAT) variable is set to TRUE; otherwise, set it to FALSE.

■ Diagnostic Info

In the online mode, the diagnosis information includes the information about EtherCAT master station startup and running.

Options

■ Use LRW instead of LWR/LRD

This option enables the slave station - slave station communication. EhterCAT master station logical addressing will use the combination of read/ write command (LRW) to replace the read-only (LRD) and write-only (LWR) commands.

■ Enable messages per task

After this option is selected, the read and write commands of input and output information will be completed by different tasks.

■ Auto restart slaves

After this option is selected, the master station will restart the slave station upon communication error.

Master Station Settings

The master station settings can be completed only when the auto mode is disabled (see the following description); otherwise, these settings are automatically completed and hidden in the dialog box.

■ Image In Address: inputs the first logical address of the first slave station.

■ Image Out Address: outputs the first logical address of the first slave station.

2 Function Code

Function codes refer to the vendor-specific parameters of Inovance servo products. By using the master station options, you can read/write, import, and export vendor-specific parameters of multiple products, for commissioning and maintenance.

Slaves
⊟ ☐ 🖿 IS620N
⊟ ☐ 🖉 Axis
☐ 🔵 H00 Servo Motor Parameters
☐ 🔵 H01 Servo Drive Parameters
☐ 🔵 H02 Basic control
☐ 🔵 H03 Input Terminal Paramet...
☐ 🔵 H04 Output terminal Parame...
☐ 🔵 H05 Position Control Parame...
☐ 🔵 H06 Speed control
☐ 🔵 H07 Torque control
☐ 🔵 H08 Gain Parameters
☐ 🔵 H09 Auto-adjusting Paramet...
☐ 🔵 H0A Error and Protection Pa...
☐ 🔵 H0B Display Parameters
☐ 🔵 H0C Communication
☐ 🔵 H0D Auxiliary Function Para...
☐ 🔵 H0F Closed Loop Function
☐ 🔵 H17 VDI VDO
☐ 🔵 H30 Servo Related Variables...
☐ 🔵 H31 Servo Related Variables...
☐ 🔵 H3F Manufacturor Error code

FunCode	Name	CurrentValue	WriteValue	DefaultValue	Range	Access
⊟ 🔵 H02	Basic control					
🔵 H02-00	Control mode			9	0-9	RO
🔵 H02-01	Absolute Encoder M...			0	0-3	RW
🔵 H02-02	Rotating direction			0	0-1	RW
🔵 H02-03	Direction of output p...			0	0-1	RW
🔵 H02-05	Stop mode at servo ...			0	0-1	RW
🔵 H02-06	Stop mode at fault 2			1	0-2	RW
🔵 H02-07	Stop mode at overtr...			1	0-2	RW
🔵 H02-08	Stop mode at fault 1			0	0-0	RW
🔵 H02-09	Brake release comm...			250	0-500	RW
🔵 H02-10	Servo drive disable ...			150	1-1000	RW
🔵 H02-11	Output speed limit o...			30	0-3000	RW
🔵 H02-12	Waiting time from se...			500	1-1000	RW
🔵 H02-15	Display of keypad w...			0	0-1	RW
🔵 H02-21	Allowed minimum br...			40	1-1000	RO
🔵 H02-22	Power of built-in br...			40	1-65535	RO
🔵 H02-23	Resistance of built-i...			50	1-1000	RO
🔵 H02-24	Resistor heat dissipa...			30	10-100	RW
🔵 H02-25	braking resistor type			0	0-3	RW
🔵 H02-26	Power of external d...			40	1-65535	RW
🔵 H02-27	Resistance of exter...			50	1-1000	RW
🔵 H02-31	Parameter initialization			0	0-2	RW
🔵 H02-32	Default keypad display			50	0-99	RW
🔵 H02-35	Display frequency of...			0	0-20	RW

■ Select All: selects all slave stations, axes, and servo function codes under the axes.

■ Cancel All: cancels all selected options.

■ Read the Parameter: reads the servo function codes carrying the RO, RW or R attribute when the servo is running.

■ Write the Parameter: writes the servo function codes carrying the W or RW attributes when the servo is running.

■ Export FunCode: exports all servo function codes of slave stations, in Excel format.

■ Import FunCode: imports all function codes of slave stations from an Excel file. If the configuration is inconsistent with the file, an error is reported.

3 Upgrade

General
Function Code
Update
Sync Unit Assignment
EtherCAT I/O Mapping
Status
Information

Slave Name	Update
AM600_2HCE	☑
IS620N	☑

Select All Cancel All Download EtherCAT XML file

■ Select All: selects all slave stations.

■ Cancel All: cancels all selected slave stations.

■ Download EtherCAT XML file: downloads the EtherCAT slave station's XML file from InoProShop to the E2PROM of the slave station. To perform batch download, select multiple slave stations.

4 Sync Unit Assignment

■ Sync unit: groups slave stations. If any slave station in the group is lost, the entire group is lost, but other groups are not affected.

■ Add: adds a group. Then you can select the slave station group.

5 EtherCAT I/O Mapping

This is a tab on the EtherCAT master station configuration editor, in which the instance (variable) of IODrvEtherCAT type is specified for EtherCAT I/O, so that the PLC connected to EtherCAT can be controlled by user program. For the description of mapping, see I/O Mapping.

The automatically created master station instance is displayed at the bottom of the IEC Objects dialog box.
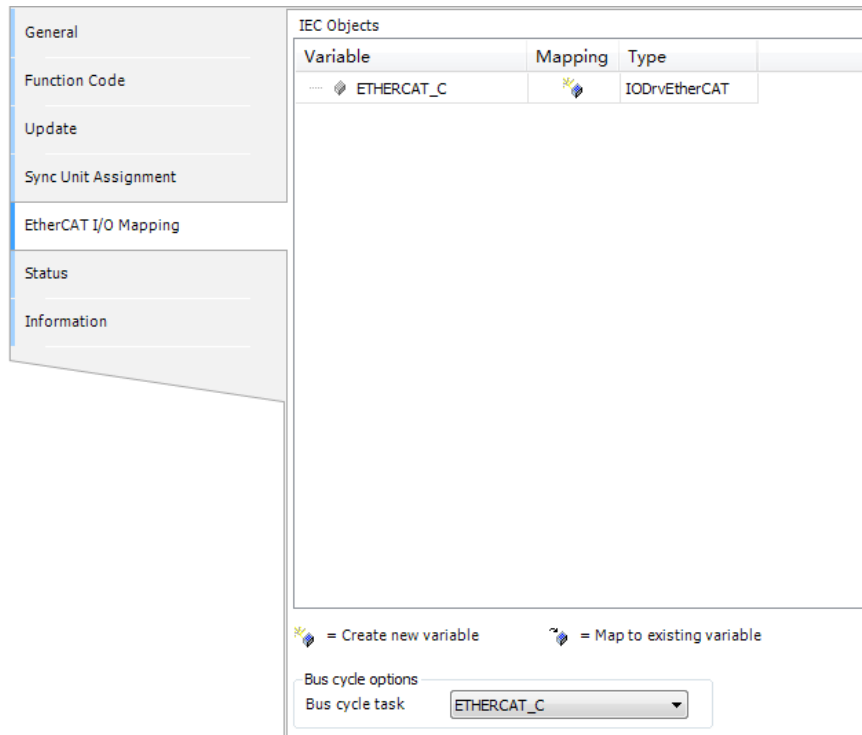
NOTE

Note: The variables and types of mappings must be consistent.

6 Information

This dialog box displays the following information about the current module: name, vendor, group, category, ID, and version.
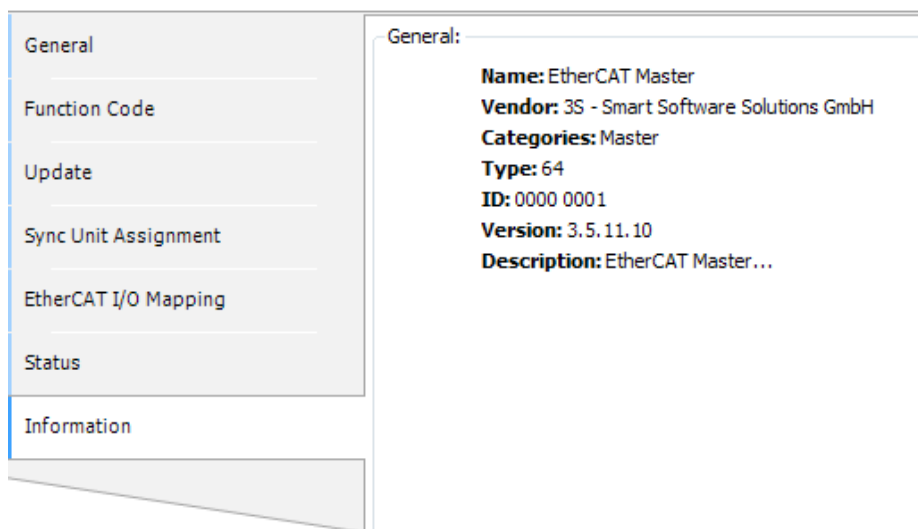
3.3.4 EtherCAT Slave Station

1 General Settings

The general settings interface of the EtherCAT slave station is as follows. It provides the basic settings of the slave stations.

Address

If automatic configuration mode in the master station editor is not enabled, the following options are available:

- AutoInc Address: indicates the automatically incremental address (16 bits), which is determined by the physical topology location of the slave station. This address is used only when the EtherCAT is started. The EtherCAT slave station address is allocated to the slave station at the corresponding physical topology location by using the sequential addressing method.

During sequential addressing, according to the EtherCAT protocol, the AutoInc address of the slave station is determined by its connection location in the physical topology network, and is represented by a negative number. Sequential addressing sends the subframe, in which the AutoInc address is increased by 1 every time the subframe passes a slave station. When the physical slave station receives the frame, it determines whether the frame belongs to itself by checking whether the AutoInc address in the frame is 0. This mechanism is called sequential addressing or automatically incremental addressing.

- EtherCAT Address: indicates the final address (formal address) of the slave station. It is assigned by the master slave when the master station starts. This address is independent of the actual location on network. The slave station address is irrelevant to the connection order on the network segment.

- Enable Expert Settings: If this option is enabled, the settings of distributed clock, auto check upon startup, timeout, cycle unit control, and watchdog can be performed.

Distributed Clock

- If the enable option is selected, the distributed clock function is enabled.

- Sync Unit Cycle (μs): If the distributed clock function is enabled, the syn unit cycle value is the same as the cycle time of EtherCAT master station.

- Select DC: This option provides all distributed clock settings in the device description file, including AutoRun, SM Event Synchron, DC Synchron. The options are described as follows:

No.	Option	Function
1	AutoRun	In this mode, the local control cycle is generated by a local timer interrupt. The cycle time is determined by the master station. This is an optional function of the slave station.
2	SM Event Synchron	Triggered by the data input or output event within the local cycle. The master station can write the sending period of the procedure frames to the slave station, and the slave station checks whether this cycle time is supported or optimizes the cycle time. This is an optional function of the slave station. The synchronization is usually triggered by data output event. If the slave station has only the data input event, the synchronization is triggered by data input.
3	DC Synchron	Triggered by the SYNC event within the local cycle. The master station must complete data frame sending before the SYNC event. Therefore, the master station clock must be synchronized with the reference clock.
4	SYNC0: SYNC1:	Indicates the slave station synchronization signal 0/1. The distributed clock control unit (internal function of the EtherCAT dedicated communication chipset) of the slave station can generate two synchronization signals: SYNC 0 and SYNC 1, which provide the interrupt sign to the application layer programs of the slave station, or directly trigger the output data update.
5	Enable SYNC 0: Enable SYNC 1:	After this option is selected, SYNC0/SYNC1 synchronization signals are started. Sync Unit Cycle: If this option is selected, the synchronization cycle time of the slave station is the master cycle time x selected coefficient. The **Cycle time (μs)** field displays the current cycle time.

- User Defined: If this option is selected, you can enter the desired cycle time in μs in the **Cycle time (μs)** field.

Diagnosis

This part is available only in the online mode.

- Current Status: displays the current communication state machine of the slave station. The possible values include initialization, pre-operation, safe operation, operation, and BootStrap (not supported by Inovance servo currently). If the status is running, the configuration of slave station is completed.

Additional

Options: If a slave station device is set as optional, no error message is generated, so that the device will not be included in the bus system. To enable this option, you must save a station address in the slave station device. Therefore, define and write the station alias into E2PROM. In addition, this option is effective only when **Autoconfig Master/Slave** in EtherCAT master station settings is selected and this option is supported by the EtherCAT slave station.

Startup checking

By default, the system automatically checks the vendor ID or product ID after startup. If the IDs do not match, the bus stops running and does not perform the subsequential operations. This avoids downloading wrong configuration.

Timeouts

By default, the following operations are not defined as timeout. If you need to know whether the operations exceed the specified time, set the time here:

■ SDO Access: Timeout interval in EtherCATzwhich the service data object (SDO) of the EtherCAT master station accesses the slave station.

■ I -> P: The communication state machine of slave station changes from initialization to pre-operation.

■ P -> S / S-> O: The communication state machine of slave station changes from pre-operation to safe operation, or from safe operation to operation.

■ DC cyclic unit control: Allocated to the local microprocessor to set the distributed clock options. This function is completed in register 0x980 of the EtherCAT slave station. The possible values include cyclic unit, latch unit 0, and latch unit 1.

Watchdog

■ Set multiplier: Set the frequency multiplication ratio of the watchdog timer to determine the minimum increment unit. The default value is 2498. The minimum increment unit is 100 μs.

■ Set PID watchdog: If PDI watchdog is enabled, when the process data interface (PDI) communication time of EtherCAT slave station exceeds the specified value, the watchdog is triggered.

■ Set SM watchdog: If synchronization management (SM) watchdog is enabled, when the process data communication time of EtherCAT slave station exceeds the specified value, the watchdog is triggered.

Slave Station Alias

The settings are effective only when **Options** is selected and the slave station supports alias address (defined in the device description file). If the alias address of the slave station has been configured, the slave station can normally run without the need of modifying the user program configuration when you adjust its location on the physical topology network.

Note: After the alias address of slave station is changed, you must download the user program again to make the alias effective. In addition, the aliases of some slave stations can take effect only after power cycle. For details, see the slave station manuals.

■ Disable: If this option is selected, the slave station will not detect the alias address.

■ Configure Station Alias (ADO 0x0012): When this option is supported by the slave station and **Options** is selected, the alias address can be written in online running state.

■ Write into E2PROM: This option is available only in online mode. It writes the defined address into the slave station E2PROM. If the slave station does not support this option, this option is invalid and the slave station cannot work with the alias.

■ Real Address: This column is available only in online ode. It displays the actual address of the slave

station. It is used to check whether the E2PROM writing command is successful.

■ Explicit Device Identification ADO(0X0134): Reserved.

■ Data Word (2 Bytes): Reserved.

2 FMMU/Sync

If the autoconfig mode of the master station is not enabled, this dialog box is only provided by the EtherCAT slave station configuration editor. It displays the slave station's fieldbus memory management units (FMMUs) defined in the device description file and the synchronization manager (Sync). These settings can be modified, for example, the communication between slave stations.
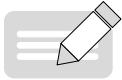
NOTE

These are advanced settings, and are unnecessary for standard applications.

FMMU

This tab configures the fieldbus memory management unit of the slave station used to process the process data, including the logical address (Ph. Start Address) mapping to each physical address (Ph. GlobStartAddr). You can add a new unit by clicking **Add...** or **Edit...** In the FMMU dialog box.

Sync Manager

This tab displays the synchronization manager of the slave station, including the synchronization manager type (Mailbox In, Mailbox Out, Inputs, or Outputs), physical start address, access type, buffer, and physical address to be accessed by the interrupt. In the synchronization manager editor, you can click **Add** or **Edit** to add or modify synchronization management.

FMMU

Add · Edit · Delete

GlobStartAddr	Length/By...	Start...	End...	Ph. Start Add...	Ph. StartBit	Flags
16#02000000	20	0	7	16#1100	0	WE
16#01000000	32	0	7	16#1400	0	RE

Sync Manager

Add · Edit · Delete

Start Address	Length/Byte	Flags	Type
16#1000	128	16#00010026 (1WPE)	Mailbox Out
16#1080	128	16#00010022 (1RPE)	Mailbox In
16#1100	20	16#00010064 (3WPE)	Outputs
16#1400	32	16#00010020 (3RPE)	Inputs

3 Process Data

In the automated control system, application programs exchange data in two modes: time-critical and non-time-critical. Time-critical means that the specified action must be completed within the specified time window. If the action cannot be completed in the specified time window, the control is ineffective. The procedure for periodically sending time-critical data is called PDO. Non-time-critical data does not need to be periodically sent, and uses MailBox data communication SDO in EtherCAT.

The first row of process data is the PDO edit function key and displays PDO information, shown as follows:

- Add: Adds PDO based on the PDO group attributes (only editable attributes). You can add one group or multiple groups of data. To add multiple groups of data, press **Ctrl+** and **Shift+**, and click the desired groups. Note that the object dictionary indexes of the options to be added are correct. Note: The number of PDOs to be added cannot exceeds the limitation described in the servo manual.

- Edit: Edits the PDO option based on the PDO group attributes (only editable attributes).

- Delete: Deletes the PDO option based on the PDO group attributes (only editable attributes). You can delete one or multiple options. To delete multiple options, press **Ctrl+** and **Shift+**, click the desired groups, and click **Delete**. Or right-click the selected options and choose **Delete** from the shortcut menu.

- Collapse: Collapses all PDO groups.

- Filter: Includes the display all, display output PDO, display input PDO, display all, display input and output PDO, display output PDO, display only output PDO, display input PDO, and display only input PDO group functions.

- Load PDO: The PDO group data of slave station can be uploaded to the programming software only when the slave station is running.

- PDO Assign: After this option is selected, click the **System Parameters Options** in the Startup Parameters interface. Then the input and output PDO group assignment information is added to the startup parameter group, shown as follows:

Sidebar: General | FMMU/Sync | Process Data | Startup parameters(SDO) | Online | CoE Online | Servo Function Code | EtherCAT I/O Mapping | Status | Information

Toolbar: Add | Edit | Delete | Move Up | Move Down | DownLoadAll(SDO) | CancelAllDownload(SDO) | ☑ DisplaySystemParameter

Line	Index:Subindex	Name	Value	Bitlength	IsDownload	Abort if error	Jump to line if err...	Next line	Comment
1	16#1C12:16#00	clear pdo 1C12	0	8				0	
2	16#1C13:16#00	clear pdo 1C13	0	8				0	
3	16#1C12:16#01	download pdo 1C12:1 index	5889	16				0	
4	16#1C12:16#00	download pdo 1C12 count	1	8				0	
5	16#1C13:16#01	download pdo 1C13:1 index	6913	16				0	
6	16#1C13:16#00	download pdo 1C13 count	1	8				0	
7	16#6060:16#00	Modes of operation	8	8	☑			0	Modes of operation

■ PDO Config: Select PDO configuration and click the System Parameters Options on the Startup Parameters interface. Then the input and output PDO group information is added to the startup parameter group, as show below:

Sidebar: General | FMMU/Sync | Process Data | Startup parameters(SDO) | Online | CoE Online | Servo Function Code | EtherCAT I/O Mapping | Status | Information

Toolbar: Add | Edit | Delete | Move Up | Move Down | DownLoadAll(SDO) | CancelAllDownload(SDO) | ☑ DisplaySystemParameter

Line	Index:Subindex	Name	Value	Bitlength	IsDownload	Abort if error	Jump to line if err...	Next line	Comment
1	16#1C12:16#00	clear pdo 1C12	0	8				0	
2	16#1C13:16#00	clear pdo 1C13	0	8				0	
3	16#1C12:16#01	download pdo 1C12:1 index	5889	16				0	
4	16#1C12:16#00	download pdo 1C12 count	1	8				0	
5	16#1C13:16#01	download pdo 1C13:1 index	6913	16				0	
6	16#1C13:16#00	download pdo 1C13 count	1	8				0	
7	16#1600:16#00	clear pdo 1600	0	8				0	
8	16#1600:16#01	download pdo 1600:1 entry	1614807056	32				0	
9	16#1600:16#02	download pdo 1600:2 entry	1618608160	32				0	
10	16#1600:16#03	download pdo 1600:3 entry	1622671376	32				0	
11	16#1600:16#00	download pdo 1600 count	3	8				0	
12	16#1A00:16#00	clear pdo 1A00	0	8				0	
13	16#1A00:16#01	download pdo 1A00:1 entry	1614872592	32				0	
14	16#1A00:16#02	download pdo 1A00:2 entry	1617166368	32				0	
15	16#1A00:16#03	download pdo 1A00:3 entry	1622736912	32				0	
16	16#1A00:16#04	download pdo 1A00:4 entry	1622802464	32				0	
17	16#1A00:16#05	download pdo 1A00:5 entry	1622933536	32				0	
18	16#1A00:16#06	download pdo 1A00:6 entry	1614741520	32				0	
19	16#1A00:16#07	download pdo 1A00:7 entry	1627193376	32				0	
20	16#1A00:16#00	download pdo 1A00 count	7	8				0	
21	16#6060:16#00	Modes of operation	8	8	☑			0	Modes of operation

■ PDO Len: The data size includes the total input and output PDO length.

4 Startup Parameters

The startup parameters can be transmitted to the slave station through the service data object (SDO). The startup parameters include the basic configuration parameters used for the startup of the slave station. The interface is as follows:

Sidebar: General | FMMU/Sync | Process Data | Startup parameters(SDO) | Online | CoE Online | ESC Register | EtherCAT I/O Mapping | Status | Information

Toolbar: Add | Edit | Delete | Move Up | Move Down | DownLoadAll(SDO) | CancelAllDownload(SDO) | ☐ DisplaySystemParameter

Line	Index:Subindex	Name	Value	Bitlength	IsDownload	Abort if error	Jump to line if err...	Next line	Comment
1	16#3100:16#00	Counter type	8481	16	☑			0	Counter type
2	16#3103:16#01	Ch0 maximum counter value	2880000	32	☑			0	Ch0 maximum counter value
3	16#3103:16#02	Ch1 maximum counter value	288000	32	☑			0	Ch1 maximum counter value
4	16#3104:16#01	Ch0 minimum counter value	0	32	☑			0	Ch0 minimum counter value
5	16#3104:16#02	Ch1 minimum counter value	0	32	☑			0	Ch1 minimum counter value
6	16#3105:16#01	CH0 external input function selection	513	32	☑			0	CH0 external input function sel
7	16#3105:16#02	CH1 external input function selection	513	32	☑			0	CH1 external input function sel
8	16#3106:16#00	External input polarity selection	0	16	☑			0	External input polarity selection
9	16#3107:16#01	Ch0 external output function selection	513	32	☑			0	Ch0 external output function s
10	16#3107:16#02	Ch1 external output function selection	513	32	☑			0	Ch1 external output function s
11	16#3108:16#00	External output polarity selection	0	16	☑			0	External output polarity selecti
12	16#3109:16#01	Ch0 filter	4	16	☑			0	Ch0 filter
13	16#3109:16#02	Ch1 filter	4	16	☑			0	Ch1 filter
14	16#310A:16#01	Ch0 pluse rate test time	10	16	☑			0	Ch0 pluse rate test time
15	16#310A:16#02	Ch1 pluse rate test time	10	16	☑			0	Ch1 pluse rate test time

■ Add: Adds an SDO project to the startup parameter list. The pop-up object dictionary dialog box is as follows:
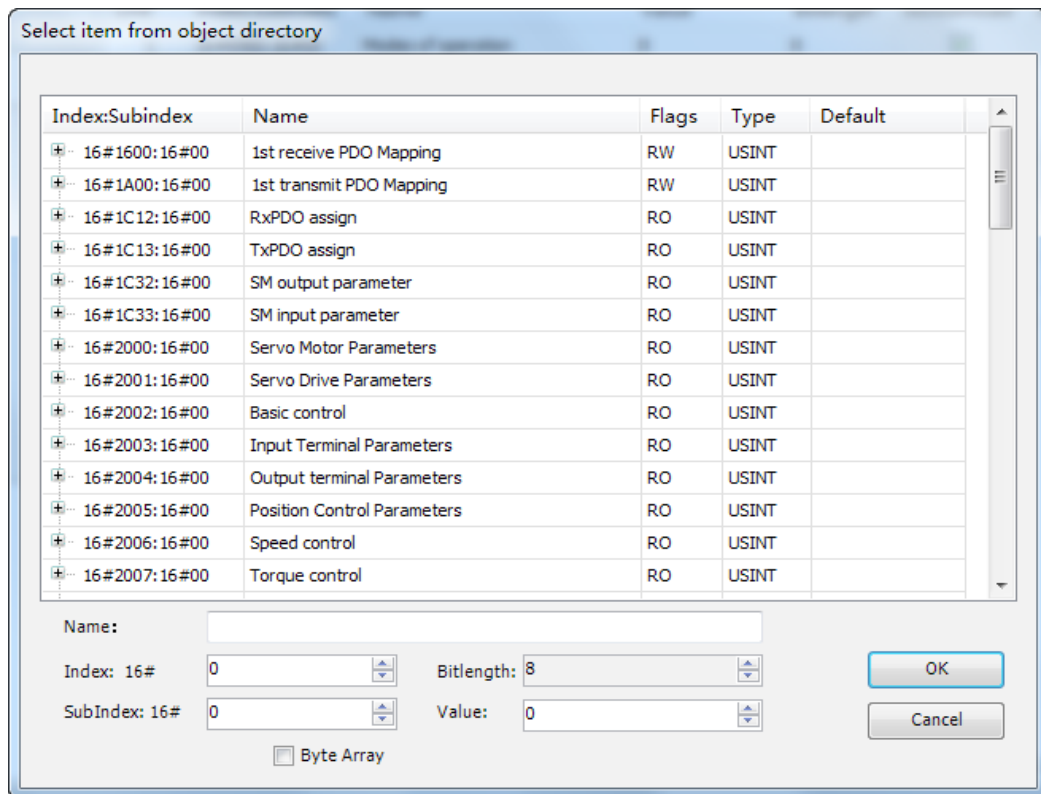
Before adding the SDO, you can modify its parameters below the edit bar, including the index, sub-index, bit length, and value, to form a new startup parameter. To add multiple groups of startup parameters, you can press **Ctrl+** and **Shift+**, and click multiple groups of startup parameters.

■ Edit: Edits the options. The read-only options cannot be edited, such as system parameters.

■ Delete: Deletes the options. To delete multiple groups of startup parameters, press **Ctrl+** and **Shift+**, click multiple groups of startup parameters, and click **Del** or press the Del key.

■ Move Up, move Down

The SDO list order (from top to bottom) indicates the order in which the startup parameters are transmitted to module. By clicking the Move Up and Move Down buttons, you can change the parameter transmission order.

■ DownLoadAll, CancelAllDownload

After SDO is downloaded, you do not need to download it again. By clicking **CancelAllDownload** (system parameters cannot be canceled), you can cancel the attribute download. You can also select some attributes to be downloaded. By clicking **DownLoadAll**, you can download all attributes.

■ DisplaySystemParameter

After you select PDO distribution and PDO configuration, the parameters added to SDO are only for comparison.

■ To edit the **Value** and **Comment** columns of SDO, press **Space** or click the blank space.
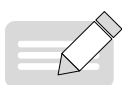
> **NOTE**
>
> If an error occurs in SDO transmission, the following operations are supported:
>
> 1) Abort if error: If an error is detected, SDO transmission is stopped.
>
> 2) Jump to line if error: If an error is detected, SDO transmission skips to the line of which the line number is specified. (Line number is displayed in the line column.)

Figure 3-28 Error handling

5 Slot Configuration

■ Slot configuration supports the slave stations in compliance with ETG5001.1. It is used to configure the modules or functions.

As shown in the following figure, the current mode is CSP_CSV:

The following modes are available:

No.	Mode	Description
1	CSP	Cyclic Synchronous Position
2	PP	Profile Position
3	CSV	Cyclic Synchronous Velocity
4	PV	Profile Velocity
5	CST	Cyclic Synchronous
6	PT	Profile Torque

If the location mode does not meet requirements, change it to another mode, for example, CST synchronization torque.

The default CSP/CSV is applicable to most application scenarios. If the field drive axis needs to use synchronization location and synchronization torque, select CSP/CST.
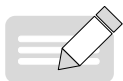
NOTE

◆ Change: switches to a different mode.
◆ Delete: deletes the current mode.
◆ To change the mode to another mode, delete the current slot mode.

After the CSP_CSV mode is changed to the CST mode the process data and I/O mapping are also changed:

Before the change to CST:

Output	Outputs		16#1600	16#00	7.0		Editable	2
Ou...	CSP Target position		16#607A	16#00	4.0	DINT		
Ou...	CSP Modes of Operation		16#6060	16#00	1.0	SINT		
Ou...	CSP ControlWord		16#6040	16#00	2.0	UINT		
Output	Outputs		16#1610	16#00	7.0		Editable	2
Ou...	CSP_1 Target position		16#687A	16#00	4.0	DINT		
Ou...	CSP_1 Modes of Operation		16#6860	16#00	1.0	SINT		
Ou...	CSP_1 ControlWord		16#6840	16#00	2.0	UINT		
Output	Outputs		16#1620	16#00	7.0		Editable	2
Ou...	CSP_2 Target position		16#707A	16#00	4.0	DINT		
Ou...	CSP_2 Modes of Operation		16#7060	16#00	1.0	SINT		
Ou...	CSP_2 ControlWord		16#7040	16#00	2.0	UINT		
Output	Outputs		16#1630	16#00	7.0		Editable	2
Ou...	CSP_3 Target position		16#787A	16#00	4.0	DINT		
Ou...	CSP_3 Modes of Operation		16#7860	16#00	1.0	SINT		
Ou...	CSP_3 ControlWord		16#7840	16#00	2.0	UINT		
Input	Inputs		16#1A00	16#00	21.0		Editable	3
Input	CSP Digital inputs		16#60FD	16#00	4.0	UDINT		
Input	CSP Following error actual value		16#60F4	16#00	4.0	DINT		
Input	CSP ActualVelocity		16#606C	16#00	4.0	DINT		
Input	CSP Position actual value		16#6064	16#00	4.0	DINT		
Input	CSP Modes of Operation Display		16#6061	16#00	1.0	SINT		
Input	CSP StatusWord		16#6041	16#00	2.0	UINT		
Input	CSP Error code		16#603F	16#00	2.0	UINT		
Input	Inputs		16#1A10	16#00	21.0		Editable	3
Input	CSP_1 Digital inputs		16#68FD	16#00	4.0	UDINT		
Input	CSP_1 Following error actual value		16#68F4	16#00	4.0	DINT		

After the change to CST:

■ Download slot configuration: After the module is configured, the configured slot information needs to be downloaded to the device. After this option is selected, the following parameters are added to the startup parameters:

Line	Index:Subindex	Name	Value	Bitlength	IsDownload	Abort if error	Jump to line if err...	Next line	Comment
19	16#609A:16#00	Homing acceleration	1000000	32	☑	☐	☐	0	
20	16#6099:16#02	Speed during search for zero	20000	32	☑	☐	☐	0	
21	16#310A:16#01	Time of home searching	50000	16	☑	☐	☐	0	
22	16#607F:16#00	Max profile velocity	1000000	32	☑	☐	☐	0	
23	16#6898:16#00	Homing method	26	8	☑	☐	☐	0	
24	16#6899:16#01	Speed during search for switch	100000	32	☑	☐	☐	0	
25	16#689A:16#00	Homing acceleration	1000000	32	☑	☐	☐	0	
26	16#6899:16#02	Speed during search for zero	20000	32	☑	☐	☐	0	
27	16#310A:16#02	Time of home searching	50000	16	☑	☐	☐	0	
28	16#687F:16#00	Max profile velocity	1000000	32	☑	☐	☐	0	
29	16#7098:16#00	Homing method	26	8	☑	☐	☐	0	
30	16#7099:16#01	Speed during search for switch	100000	32	☑	☐	☐	0	
31	16#709A:16#00	Homing acceleration	1000000	32	☑	☐	☐	0	
32	16#7099:16#02	Speed during search for zero	20000	32	☑	☐	☐	0	
33	16#310A:16#03	Time of home searching	50000	16	☑	☐	☐	0	
34	16#707F:16#00	Max profile velocity	1000000	32	☑	☐	☐	0	
35	16#7898:16#00	Homing method	26	8	☑	☐	☐	0	
36	16#7899:16#01	Speed during search for switch	100000	32	☑	☐	☐	0	
37	16#789A:16#00	Homing acceleration	1000000	32	☑	☐	☐	0	
38	16#7899:16#02	Speed during search for zero	20000	32	☑	☐	☐	0	
39	16#310A:16#04	Time of home searching	50000	16	☑	☐	☐	0	
40	16#787F:16#00	Max profile velocity	1000000	32	☑	☐	☐	0	
41	16#F030:16#00	clear slot cfg 0xf030 entries	0	8	☑	☐	☐	0	
42	16#F030:16#01	download slot cfg 0xf030 entry	2201600	32	☑	☐	☐	0	①
43	16#F030:16#02	download slot cfg 0xf030 entry	2201600	32	☑	☐	☐	0	
44	16#F030:16#03	download slot cfg 0xf030 entry	2201600	32	☑	☐	☐	0	
45	16#F030:16#04	download slot cfg 0xf030 entry	2201600	32	☑	☐	☐	0	
46	16#F030:16#00	download slot cfg 0xf030 entry count	4	8	☑	☐	☐	0	②

1 - indicates the mode ID of the current module.

2 - indicates the number of modes of the module to be downloaded.

6 Online

You can use the slave station online editor only after logging in to the . This interface is used to switch the slave station state machine manually, read/write the E2PROM of the slave station, perform FoE upload/download, and slave station firmware upgrade.

EtherCAT state machine coordinates the status of master and slave stations during initialization and running, shown as follows:
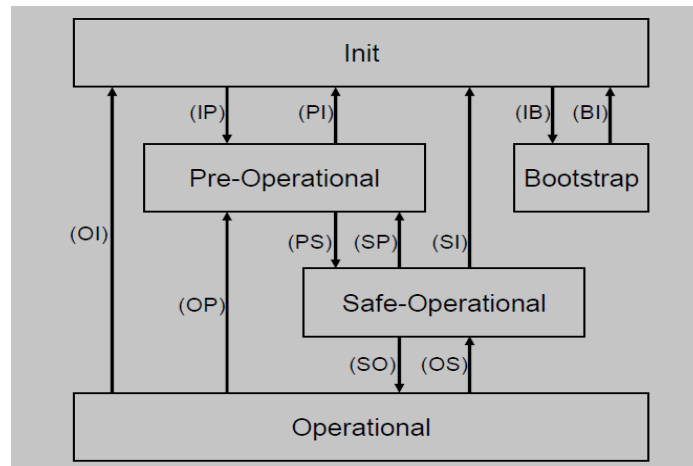
EtherCAT status conversion order: initialization -> pre-operation -> safe operation -> operation.

Status changing process:

Status Transition	Local Management Service
IP	Start Mailbox Communication
PI	Stop Mailbox Communication
PS	Start Input Update
SP	Stop Input Update
SO	Start Output Update
OS	Stop Output Update
OP	Stop Output Update, Stop Input Update
SI	Stop Input Update, Stop Mailbox Communication
OI	Stop Output Update, Stop Input Update, Stop Mailbox Communication
IB	Start Bootstrap Mode
BI	Restart Device

Initialization, pre-operation, safe operation, operation, and clear error

EtherCAT file access

To transmit firmware file to the slave station for firmware upgrade, click Bootstrap to convert the slave station into Bootstrap mode.

By clicking the corresponding button, you can download and upload the firmware file. Then a save or firmware file selection dialog box is displayed. File is transmitted by using character string and password. The information is provided by the slave station, and recorded in the data table of the slave station. During firmware upgrade, do not power off the device or switch status. Perform the operations after the upgrade is complete.

E2PROM Access

Slave station configuration can be read from E2PROM and written into E2PROM. Similar to firmware file transmission, this operation also displays a file save or open dialog box.

You can run the write E2PROM XML command to write the slave station configuration to the device through an XML file. This command is valid only when the XML file contains configuration data (in configuration data part).

Clear errors

When the current status has an error, click this button to clear the error.

7 Online COE

The online COE values can be read only when the bus is normal and you have logged in to the PLC, shown as follows:

Index:Subindex	Name	Flags	Type	Value
16#1000:16#00	Device type	RO	UDINT	
16#1001:16#00	Error Register	RO	USINT	
16#1008:16#00	Device name	RO	STRING(31)	
16#1009:16#00	Hardware version	RO	STRING(4)	
16#100A:16#00	Software version	RO	STRING(4)	
16#1018:16#00	Identity	RO	USINT	
16#1600:16#00	1st receive PDO Mapping	RW	USINT	
16#1701:16#00	258th receive PDO Mapping	RO	USINT	
16#1702:16#00	259th receive PDO Mapping	RO	USINT	
16#1703:16#00	260th receive PDO Mapping	RO	USINT	
16#1704:16#00	261th receive PDO Mapping	RO	USINT	
16#1705:16#00	262th receive PDO Mapping	RO	USINT	
16#1A00:16#00	1st transmit PDO Mapping	RW	USINT	
16#1B01:16#00	258th transmit PDO Mapping	RO	USINT	
16#1B02:16#00	259th transmit PDO Mapping	RO	USINT	
16#1B03:16#00	260th transmit PDO Mapping	RO	USINT	
16#1B04:16#00	261th transmit PDO Mapping	RO	USINT	
16#1C00:16#00	Sync manager type	RO	USINT	
16#1C12:16#00	RxPDO assign	RO	USINT	
16#1C13:16#00	TxPDO assign	RO	USINT	
16#1C32:16#00	SM output parameter	RO	USINT	
16#1C33:16#00	SM input parameter	RO	USINT	

8 EOE Settings

Ethernet over EtherCAT allows any Ethernet device to connect to the EtherCAT through a conversion terminal, without affecting the instantaneity of EtherCAT. Similar to popular Internet protocols (such as TCP/IP, VPN, and PPPoE(DSL)), Ethernet frames are transmitted using the EtherCAT protocol. This allows the standard network devices to connect to the terminals, such as printer and PC, through switches.

For the slave stations supporting Ethernet over EtherCAT (EOE), you can configure communication function. The previous dialog box is available only when the device supports Ethernet over EtherCAT:

■ Virtual Ethernet Port: Enables the EOE function for the slave station. If this option is activated, a special virtual MAC address must be specified.

■ Switch Port: Uses this device as an IP port. The Ethernet communication parameters must be set.

■ IP Settings: Uses this device as an IP port. The Ethernet communication parameters must be set.

The Ethernet communication parameters must be set based on the virtual Ethernet adapter parameters. Four bytes are assigned to each of IP Address, Subnet Mask, and Default Gateway to identify the slave station on the network. When you place the cursor on the edit area, you can modify the default settings.
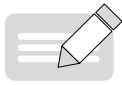
NOTE	The IP port must be in the same network segment as the virtual Ethernet adapter. For example, if the Ethernet adapter's IP address is 192.168.1.1 and subnet mask is 255.255.255.0, the IP port must be within the range from 192.168.1.2 to 192.168.1.254.

■ DNS Server: IP address of the DNS server.

■ DNS Name: DNS server name.

9 Servo Function Codes

Function codes refer to the vendor-specific parameters of Inovance servo products. On the Function Code tab of the slave station, you can read, write, import, and export vendor-specific parameters for commissioning and maintenance, as shown in the following figure:

■ Select All: Selects all servo function codes or cancels all selections.

■ Select Page: Selects all or cancels all selections on the current page.

■ Read: Read only when the slave station is in operation state and the project attribute is read.

■ Write: Written only when the slave station is in operation state and the project attribute is write.

■ Export: Exports the selected function codes.

■ Import: Imports function codes.

10   ESC Register

The ESC register is available only when the **Enable Expert Settings** option is selected. It is used to read the ESC chipset register address in advanced commissioning, shown as follows:

Selected	Address	Value	Write Value	Flags	Length(Byte)	Description
☐	16#0000			R	2	Revision/Type
☐	16#0002			R	2	Build
☐	16#0004			R	2	SM/FMMU channels
☐	16#0006			R	2	Ports Config/DPRAM Size
☐	16#0008			R	2	Features
☐	16#0010			RW	2	Configured Station Address
☐	16#0012			R	2	Configured Station Alias
☐	16#0020			RW	2	Register Write Protection/Enable
☐	16#0030			RW	2	ESC Write Protection/Enable
☐	16#0040			RW	1	ESC Reset ECAT
☐	16#0041			R	1	ESC Reset PDI
☐	16#0100			RW	2	ESC DL Control_L
☐	16#0102			RW	2	ESC DL Control_H
☐	16#0108			RW	2	Physical RW offset
☐	16#0110			R	2	ESC DL Status
☐	16#0120			RW	2	AL Control
☐	16#0130			R	2	AL Status
☐	16#0134			R	2	AL Status Code
☐	16#0140			R	1	PDI Control
☐	16#0141			R	1	ESC Config
☐	16#0150			R	2	PDI Config_1
☐	16#0152			R	2	PDI Config_2

■ SelectAll: Selects all items.

■ CancelAll: Cancels all selections.

■ Read: Reads option values in operation state.

■ Write: Writes the values carrying write attributes in operation state.

■ Export: Exports the selected items in XML format.

■ Import: Imports a valid XML file. Only the exported XML items are displayed.

■ Shortcut menu: Implements conversion among hexadecimal, decimal, and binary formats.

11　EtherCAT I/O Mapping

EtherCAT I/O Mapping is a tab in EtherCAT slave station configuration editor, in which the ETCSlave instance (variable) and I/O variable defined by slave station are specified for EtherCAT I/O. Therefore, the EtherCAT slave station connected to the PLC can be controlled by user program.

For the description of mapping, see I/O Mapping.

The automatically created slave station instance is displayed in IEC Objects at the bottom of the dialog box.
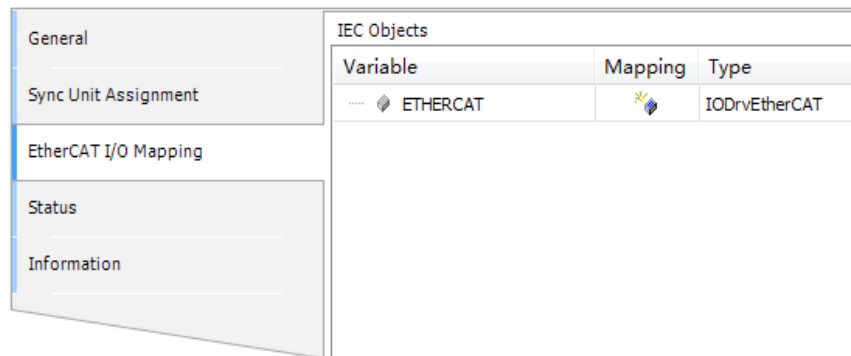
Note: The variables and types of mappings must be consistent.

12　Status

This configuration editor is used to configure the EtherCAT slave station, including the NIC and internal bus system status information (such as startup and stop), and diagnosis information of the specified device.

13　Information

This dialog box is provided in EtherCAT master or slave station configuration. The configuration of a module includes: Name, Vendor, Categories, Version, Type, Order ID, Description, and Image.

3.3.5 CiA402

After adding a servo slave station, double-click an axis. The axis configuration interface is displayed. The following is the description of the options on the axis configuration interface from top to bottom.

1 General Setting

Axis general settings include virtual axis and physical axis. The definitions are as follows:

Type	Function
Virtual mode	Virtual mode means the operation without physical servo or motor. It obtains desired parameters by simulated operation. This mode is not affected by external environment.
Physical mode	Physical mode means the operation with servo and motor. Some parameters can be obtained only in physical axis mode, for example, online COE. This mode may be affected by external environment, for example, the trace operation.

Settings of virtual and physical axes
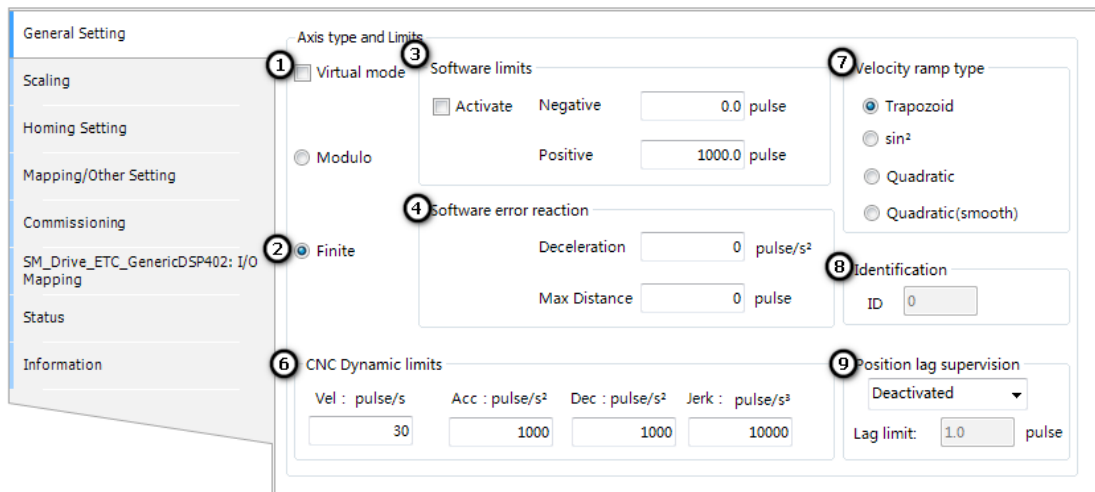
Settings of modulo and finite modes

The functions in the figure are as follows:

No.	Name	Function
1	Virtual mode	If this option is selected, the virtual axis mode is used; otherwise, the physical axis mode is used.
2	Axis type	Modulo mode: Axis locations are added or reduced in modulo way. Finite mode: Axis locations are added or reduced within a specified range.
3	Software limits	After this option is selected, the negative and positive location limits on axis are applied to the modulo mode.
4	Software error reaction	It is relevant to software limits, and is effective only when the software limits function is enabled. After this option is selected, if the axis location exceeds the software limit, the software reacts in response to the error. That is, it decelerates to the maximum distance.
5	Modulo settings	It applies to the finite mode to limit the finite cycle. This parameter is associated with the **Command pulse count per motor rotation** parameter on the Scaling interface, the homing parameters on the Homing Setting interface, the **Maximum Velocity** parameter on the Mapping/Other Setting interface. When setting this parameter, note the associated parameter settings. If the associated parameter settings do not match this parameter, the error will be prompted and the correct parameter values will be displayed.
6	CNC Dynamic limits	Applied to the settings of CNC function axis.
7	Velocity ramp type	Applied to the axis velocity change track.

No.	Name	Function
8	Identification	Axis's external ID.
9	Position lag supervision	Axis operation mode when the position lags.

The following figure shows the axis operation in modulo mode after the servo starts, including the real-time position, velocity, accelerated velocity, torque, and communication status. The error information can be displayed.

2 Scaling

This interface is used to calculate the number of pulse by setting the related parameters.

Unit in application

① ⦿ pulse   ○ mm   ○ um   ○ nm   ○ degree   ○ inch

Travel Distance

② ☐ Invert Direction

③ Command pulse count per motor rotation   `16#100000`   pulse/rev

④ ⦿ Do not use gearbox

     Work travel distance per motor rotation   `1`   pulse/rev

Reference: Unit conversion formula

$$\text{Number of pulses [pulse]} = \frac{\text{Command pulse count per motor rotation [DINT]}}{\text{Work travel distance per motor rotation [LREAL]}} * \text{Travel distance [Unit in application]}$$

⑤ ○ Use gearbox

     Work travel distance per work rotation   `1`   pulse/rev

     (Please refer to the Modulo value in General Setting if the Axis type is Modulo mode)

     Numerator of the gear ratio (the number of teeth (5) in the following picture)   `1`

     Denominator of the gear ratio (the number of teeth (4) in the following picture)   `1`

The Axis type is Linear mode

Reference: Unit conversion formula

$$\text{Number of pulses [pulse]} = \frac{\text{Command pulse count per motor rotation [DINT]}}{\text{Work travel distance per work rotation [LREAL]}} * \frac{\text{Numerator of the gear ratio [DINT]}}{\text{Denominator of the gear ratio [DINT]}} * \text{Travel distance [Unit in application]}$$

M: Motor , W: Work

The Axis type is Modulo mode

Reference: Unit conversion formula

$$\text{Number of pulses [pulse]} = \frac{\text{Command pulse count per motor rotation [DINT]}}{\text{Work travel distance per work rotation [LREAL]}} * \frac{\text{Numerator of the gear ratio [DINT]}}{\text{Denominator of the gear ratio [DINT]}} * \text{Travel distance [Unit in application]}$$

M: Motor , W: Work

The options and functions in the previous figure are as follows:

No.	Name	Function
1	Unit in application	Sets the length unit according to your requirement. After the unit is selected, the unit settings in axis general settings, scaling, homing settings, and mapping/other settings modules are also changed.
2	Invert Direction	Enables the axis to run in the invert direction.
3	Command pulse count per motor cycle	Sets the encoder resolution ratio of the motor, namely, the number of pulses required for a motor rotation. The default value is 1048576 (Inovance 20-bit encoder).
4	Do not use gearbox	Sets the work travel distance per motor rotation according to the device situation. Work travel distance per work rotation: travel distance (unit in application) of the table (such as belt pulley, gear, and reducer) from the end of machinery per rotation. The number of pulses can be calculated by referencing the unit conversion formula.
5	Use gearbox	Sets the work travel distance, numerator of gear ratio, and denominator of gear ratio per motor rotation according to the device situation. Work travel distance per work rotation: travel distance (unit in application) of the table from the end of machinery per rotation; numerator of gear ratio: number of teeth of the table; denominator of gear ratio: number of teeth of gear. The number of pulses can be calculated based on the axis type selected in axis basic settings and the corresponding reference unit. Note: The denominator and numerator of gear ratio can be scaled up and down by ratio.

Application Example

1) The motor directly drives the screw rod to move, and the motor moves 10 mm per circle around the screw rod. The motor is IS620N incremental motor (20-bit encoder resolution ratio) of Inovance. The configuration is as follows:

2) The motors are connected by driving turnplate. The reduction gear ratio between motor and turnplate is 30:1. (If the number of motor gear teeth is 1, the number of table teeth is 30. That is, when the work gear rotates 1 round, the motor gear rotates 30 rounds.) The travel distance of turnplate is 0-360 degree. The motor is the IS620N absolute motor (23-bit encoder resolution ratio) of Inovance. The configuration is as follows:

Unit in application

○ pulse   ○ mm   ○ um   ○ nm   ● degree   ○ inch

Travel Distance

☐ Invert Direction

Command pulse count per motor rotation   16#800000   pulse/rev

○ Do not use gearbox

Work travel distance per motor rotation   10   degree/rev

Reference: Unit conversion formula

$$\text{Number of pulses [pulse]} = \frac{\text{Command pulse count per motor rotation [DINT]}}{\text{Work travel distance per motor rotation [LREAL]}} * \text{Travel distance [Unit in application]}$$

● Use gearbox

Work travel distance per work rotation   1   degree/rev

(Please refer to the Modulo value in General Setting if the Axis type is Modulo mode)

Numerator of the gear ratio (the number of teeth (5) in the following picture)   30

Denominator of the gear ratio (the number of teeth (4) in the following picture)   1

NOTE

After the numerator of gear ratio, denominator of gear ratio, and work travel distance per work rotation are modified, the axis parameters on other interfaces are affected; therefore, you need to modify them accordingly.

3 Homing Setting

Homing settings include the graphic parameter settings for axis homing (shown as follows). The interface provides graphic setting instruction, so you can select the homing mode from the drop-down list without reading the servo manual. You visibly and easily complete parameter settings, as shown in ① in the following figure.

The options and functions in the figure are as follows:

No.	Name	Description
1	Homing Methods	Homing method of the drive. A total of 35 options are supported (the actual homing method is determined by the drive). The diagrams below vary with the homing method. Select a homing method according to your needs.
2	Homing Vel	High velocity of the axis when searching for the deceleration point, for example, H in the figure.
3	Acceleration	Acceleration of the axis when searching for the velocity change.
4	Homing Crawl Vel	Low velocity of the axis when searching for the homing, for example, L in the figure.
5	Time Limit	Total homing time. If timeout, an alarm is reported. It is the maximum time allowed for the axis performing the homing operation. If homing times out, the axis homing operation fails.

4 Mapping/Other Setting

The options and functions in the figure are as follows:

No.	Name	Description
1	Other Setting	Sets the maximum values and velocities of positive and negative torques.
		Max Positive Torques/Max Negative Torques: Torque command limits set for protecting the drive. When the drive torque value is larger than the limit, the actual drive torque value is changed to be consistent with the limit.
		Maximum Velocity: Limits the velocity within the specified limit. If the torque value is larger than the mechanical load torque, the motor keeps speeding up, and overspeed may occur to damage the mechanical device.
		Note: The maximum velocity cannot be set to 0. If it is set to 0, the axis may have a running error.
2	Mapping	When mapping is selected, the slave station is associated with axis. The slave station data is directly mapped to the axis. If mapping is not selected, you can manually modify the address in axis mapping, in which:
		Input format is %I+ Type letters + Arabic numbers
		Output format is %Q+ Type letters + Arabic numbers
		Type letters (bytes occupied by type) include SINT-B, UINT-W, DINT-D, and UDINT-D.
		When a compiling error is reported during manual address input, the input address needs to be deleted, and you need to enter the correct address.
		Note: The input address must be quoted by single quotes. If a compiling error is reported by the display is normal, delete the displayed result and enter the address in correct format.

5 Information

This interface displays the axis basic settings, including Name, Vendor, Group, Categories, ID, Version, Order Number, and Description.
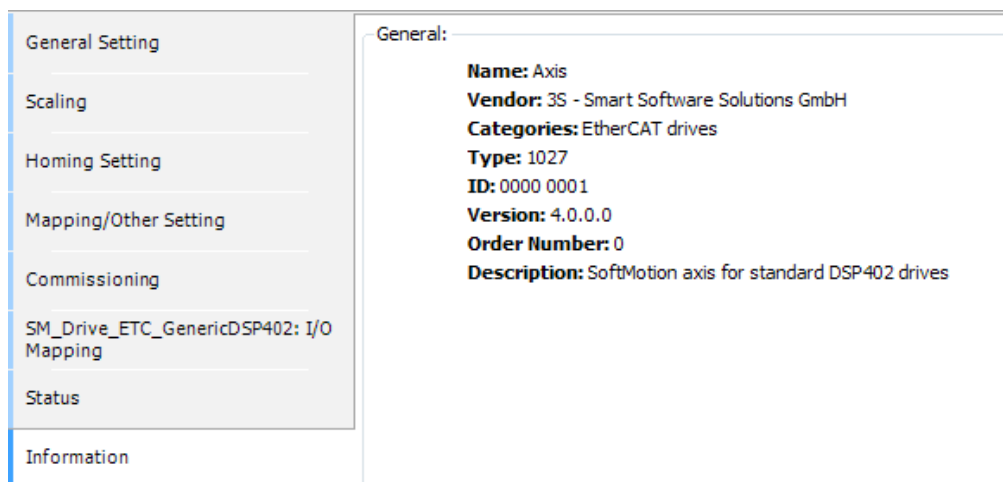
NOTE

Ensure that the storage is carried out in the specified ambient environment range, the battery has a good contact and sufficient power; otherwise the encoder position may be lost.

If the axis parameter settings are not modified, the default parameter settings of SDO are also retained. If the axis parameter settings have been modified, the default parameter settings of SDO are also modified, shown as follows:

3.3.6 Virtual Axis

The virtual axis interface is similar to the EtherCAT configuration interface. It should be noted that when multiple EtherCAT master stations are enabled and virtual axis is enabled (by choosing **Axis Pool** > **Virtual Axis**), the bus cycle task must be bound to the invoked EtherCAT task (The default setting is **Use parent bus cycle setting**. This only applies to the single-master scenarios.) One virtual axis cannot be invoked by multiple EtherCAT tasks; otherwise, a running error will occur.

The settings in red box are bus cycle task settings.
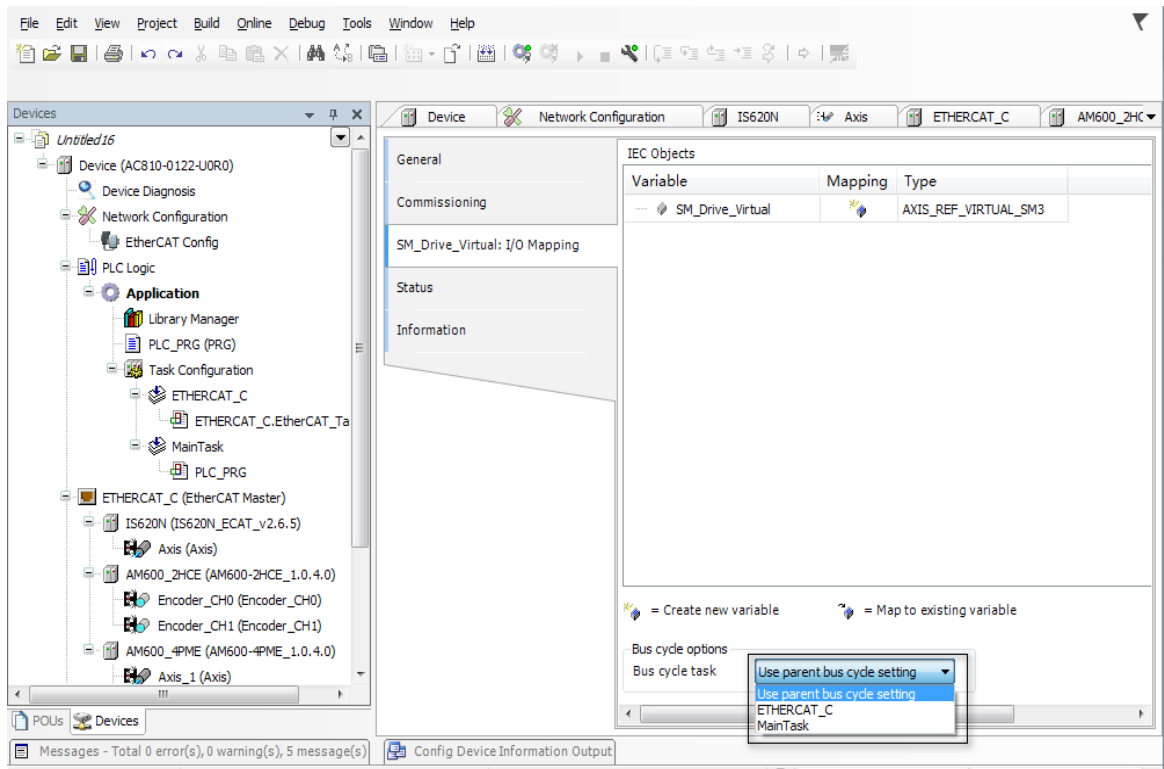
3.3.7 AM600-4PME Position Module

The position module interfaces are the same as CiA402 axis settings interfaces, except that the following interface is added:
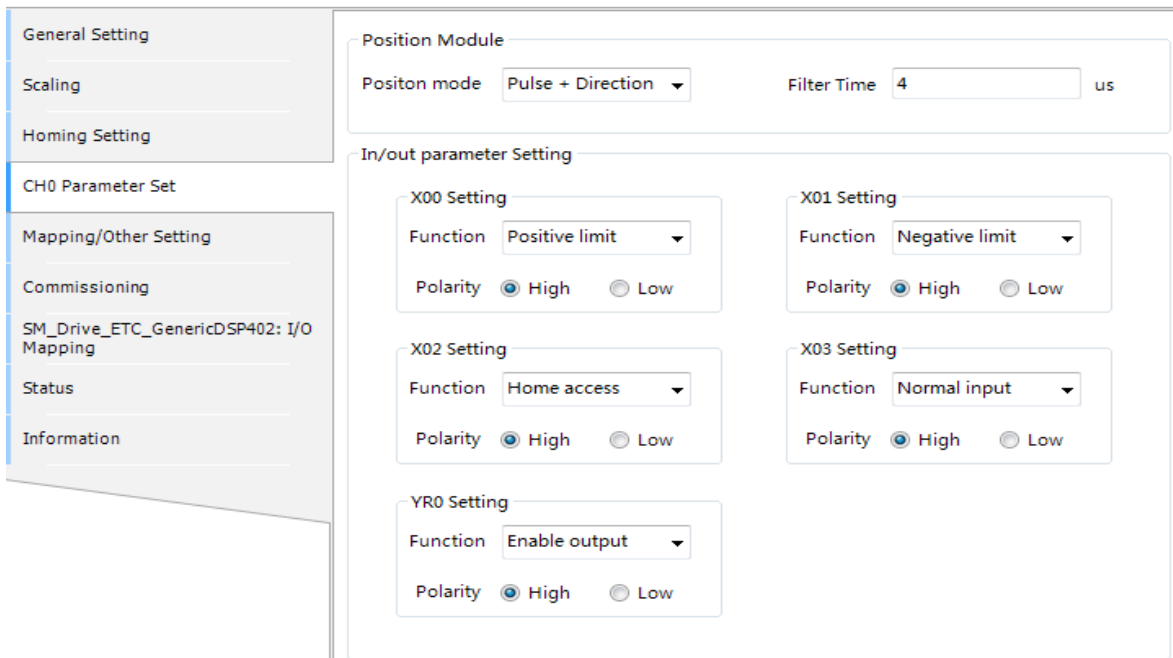
The AM600-4PME module is a pulse position module that has four high-speed output channels. It implements speed and position control for the pulse and stepping drives that use pulse as signals. This interface sets the parameters of the AM600-4PME module. Taking the first channel of AM600-4PME as an example, the interface configures the following functions:

Name		Description	Default Setting
Position Mode		Indicates the output pulse type on the high-speed pulse output interface. AB phase 1 multiplier Pulse + direction CW/CCW	Pulse + direction
Filter Time		Indicates the pulse input filter time of the digital input terminal.	4 μs
X00 Setting	Function	Selects the functions of X00 digital input terminal. Normal input Emergency stop switch Positive limit	Positive limit
	Polarity	Selects the effective polarity of X00 digital input. High - Input high level is effective. Low - Input low level is effective.	High
X01 Setting Polarity	Function	Selects the functions of X01 digital input terminal. Normal input Emergency stop switch Negative limit	Negative limit
	Selects the effective polarity of X01 digital input. High - Input high level is effective. Low - Input low level is effective.	High	
X02 Setting Polarity	Function	Selects the functions of X02 digital input terminal. Normal input Emergency stop switch Home access	Home access
	Selects the effective polarity of X02 digital input. High - Input high level is effective. Low - Input low level is effective.	High	
X03 Setting Polarity	Function	Selects the functions of X03 digital input terminal. Normal input Emergency stop switch	Normal input
	Selects the effective polarity of X03 digital input. High - Input high level is effective. Low - Input low level is effective.	High	
YR0 Setting Polarity	Function	Sets the YR0 function of digital output terminal. Normal digital output terminal Enable servo (output signal)	Enable servo (output signal)
	Energizes or disables the digital output terminal YR0. High - Energization when control is set to 1. Low - Energization when control is set to 0.	High	

For the applications of the position module, see EtherCAT Remote Communication Application Manual.

3.3.8 AM600-2HCE Counter Module

The counter module interfaces are the same as CiA402 axis settings interfaces, except that the following interface is added:

The AM600-2HCE module is a pulse counter module that has two high-speed input channels. It implements pulse counter and frequency measurement in AB phase pulse, pulse + direction, and CW/CCW modes. This interface sets the parameters of the AM600-2HCE module. Taking the first channel of AM600-2HCE as an example, the interface configures the following functions:

Name	Description	Default Setting
Mode	Selects the input mode of channel input pulse. AB phase 1 multiplier AB phase 2 multiplier AB phase 4 multiplier Pulse + direction CW/CCW	AB phase 4 multiplier
Sampling cycle	Calculates sampling cycle by input filter frequency.	10 ms
Filter Time	Sets the sampling filter of pulse input channel and digital input channel.	2 μs

Name		Description	Default Setting
Direction	Phase A ahead B	A/B-phase The counter is increased when phase A is ahead of phase B. Pulse + direction The counter is increased when phase B input level is high. CW/CCW The counter is increased when phase A has counts.	Phase A ahead B
	Phase B ahead A	A/B-phase The counter is increased when phase B is ahead of phase A. Pulse + direction The counter is increased when phase B input level is low. CW/CCW The counter is increased when phase B has counts.	
X00 Setting	Function	Selects the functions of X00 digital input terminal. Normal input Touch probe 1 Reset counter to 0 Preset counter Door control	Touch probe 1
	Polarity	Selects the effective polarity of X00 digital input. High - Input high level is effective. Low - Input low level is effective.	High
X01 Setting Polarity	Function	Selects the functions of X01 digital input terminal. Normal input Touch probe 2 Reset counter to 0 Preset counter Door control	Touch probe 2
	Selects the effective polarity of X01 digital input. High - Input high level is effective. Low - Input low level is effective.	High	
X02 Setting Polarity	Function	Selects the functions of X02 digital input terminal. Normal input Reset counter to 0 Preset counter Door control	Normal input
	Selects the effective polarity of X02 digital input. High - Input high level is effective. Low - Input low level is effective.	High	

Name		Description	Default Setting
X03 Setting Polarity	Function	Selects the functions of X03 digital input terminal. Normal input Reset counter to 0 Preset counter Door control	Normal input
	Selects the effective polarity of X03 digital input. High - Input high level is effective. Low - Input low level is effective.	High	
Y00 Setting Polarity	Function	Sets the Y00 function of digital output terminal. Normal output Compare output 1	Compare output 1
	Energizes or disables the digital output terminal Y00. High - Energization when control is set to 1. Low - Energization when control is set to 0.	High	
Y01 Setting Polarity	Function	Sets the Y01 function of digital output terminal. Normal output Compare output 2	Compare output 2
	Energizes or disables the digital output terminal Y01. High - Energization when control is set to 1. Low - Energization when control is set to 0.	High	
Y02 Setting Polarity	Function	Sets the Y02 function of digital output terminal. Normal output	Normal output
	Energizes or disables the digital output terminal Y02. High - Energization when control is set to 1. Low - Energization when control is set to 0.	High	

For the applications of the position module, see EtherCAT Remote Communication Application Manual.

3.3.9 I/O Module

The procedure for adding the Inovance AM600-RTU-ECTA module is as follows:

1) Double-click **Network Configuration**. In the pop-up graphic configuration interface, click the EtherCAT bus.
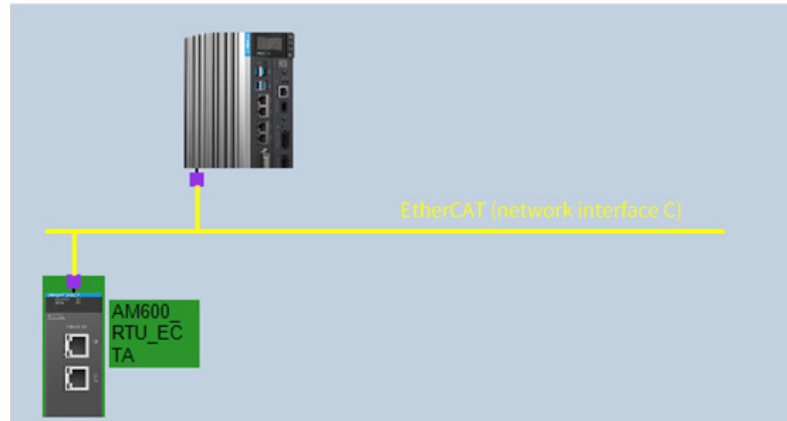
2) In the network device list, double-click the AM600-RTU-ECTA slave station to add it. (Note: AM600_EtherCAT_Slave is not recommended.)
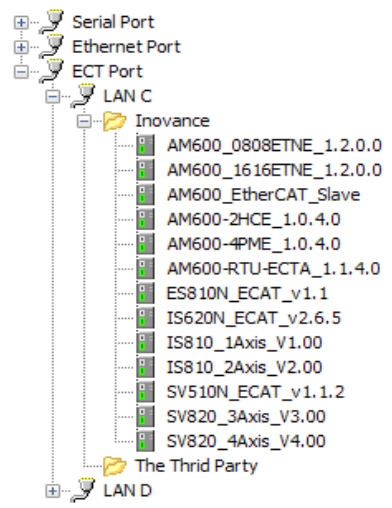
3) Double-click the AM600-RTU-ECTA slave station. In the following red box, add the desired I/O module.
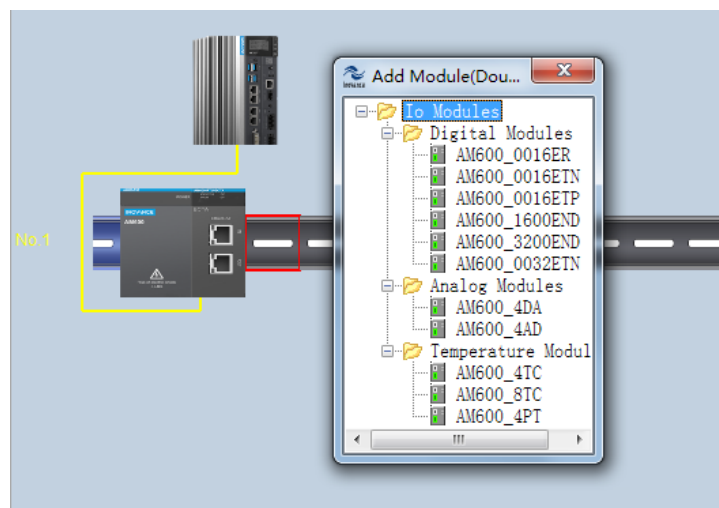
4) For the specific functions of each module, see EtherCAT Remote Communication Application Manual.

3.3.10 Library (Implicit Variables)

1 Implicit Instance of Master Station

An IoDrvEtherCAT implicit instance is created as long as the EtherCAT master station is inserted into the device list. The instance name is the same as the device name in the device list.

NOTE	An implicit instance is automatically created, and cannot be defined in program; otherwise, the PLC will encounter a running error.

The definition of IoDrvEtherCAT implicit instance is as follows:

IoDrvEtherCAT implicit instance

```
                        IoDrvEtherCAT
  —|xRestart  BOOL              BOOL  xConfigFinished|—
  —|xStopBus  BOOL              BOOL  xDistributedClockInSync|—
                               BOOL  xError|—
```

Input parameter	Description
xRestart	Bus restart: In rising edge, the master station restarts, and all configuration parameters are reloaded.
xStopBus	Bus stop: When the input value is true, the EtherCAT bus communication stops, and a communication error occurs. To resume the communication, run xRestart to restart the EtherCAT communication.
Output parameter	Description
xConfigFinished	If the parameter value is true, the transmission of all configuration parameters is complete. The communication is on-going.
xDistributedClockInSync	If distributed clock (DC) is configured for the EtherCAT slave station, the EtherCAT slave station parameters are set first when the bus starts. When parameter settings are complete (xConfigFinished is changed to true), the clocks of slave and master stations are adjusted. When the synchronization between master and slave station clocks is successful, the value true is output. If loss of synchronization occurs due to a bus fault during running, the value false is output. In the DC mode, the motion control module can be started only when this parameter is changed to true; otherwise, the position of motion axis may jump.
xError	If an error is detected during the start of EtherCAT master station or communication is interrupted when the slave station communication state machine enters operation, true is output because EtherCAT master station cannot receive any message (for example, the connection is torn down). In this situation, the error reason can be located in the diagnosis information or log of the master station.

IoDrvEtherCAT Example

EtherCAT_Master.xRestart := xRestart;	Use the xRestart parameter to restart the master station.
EtherCAT_Master.xStopBus := xStop;	Use the xStop parameter to stop the bus communication.
EtherCAT_Master(); xFinish:= EtherCAT_Master.xConfigFinished;	When the configuration parameters are successfully downloaded, invoke the master station to obtain information.

Master Station Attributes

Attribute	Description
AutoSetOperational	If this attribute is set to true, the master station always tries to restart the slave station upon communication interruption. Default value: FALSE

Attribute	Description
ConfigRead	If this attribute returns true, the configuration reading is completed. You can edit the configuration. For example, you can add customized SDOs.
DCInSyncWindow	Time window condition for setting XDistributedClockInSync to true. The value of XDistributedClockInSync is true only when the master station synchronization jitter is within this window. Default value: 50 μs
DCIntegralDivider	Integral divider of DC used for circuit control. Default value: 20
DCPropFactor	Proportion factor of DC used for circuit control. Default value: 25
DCSyncToMaster	Synchronization between DC and master station. If it is set to true, all slave stations are synchronized with the master station, rather than the first slave station synchronizing with PLC. Default value: FALSE
DCSyncToMasterWithSysTime	Synchronization with the master station DC. If it is set to true, all slave stations are synchronized with the master station system clock. The time read by SysTimeRtcHighResGet can also be used to synchronize PLC with all EtherCAT slave stations. Default value: FALSE
EnableTaskOutputMessage	Generally, EtherCAT messages are sent by bus cycle task, while some messages may be sent by each slave station output task. All outputs are written into the bus cycle task, and all inputs will be read. In other tasks, outputs are sent one more time so that they can be written into the corresponding slave stations immediately. Therefore, the deadline should be shortened to ensure a fast writing. When DC is available, some slave stations may encounter problems. For example, synchronization between servo controller and synchronization interrupt is lost, but the written time is used for internal synchronizer. In this situation, multiple write access operations may exist in a cycle. If EnableTaskOutputMessage is set to false, only the bus cycle task is used, but addition tasks will not affect messages. Default value: TRUE.
FirstSlave	Pointer of the first slave station under the master station.
FrameAtTaskStart	If FrameAtTaskStart is set to true, the frame content to the slave station will be sent when the task starts, to ensure the minimum jitter. This command ensures the smooth motion of the servo drive. If this flag is set to true, the output buffer frame is written into the next cycle. Default value: FALSE
LastInstance	Pointer associated with the master station list -> previous master station.
LastMessage	This attribute with the EtherCAT latest message together return a character string. If the startup is successful, "all slave stations completed" is returned. The function of character string is the same as the diagnosis information displayed in the EtherCAT master station editor in online mode.
NextInstance	Pointer associated with the master station list -> next master station.
NumberActiveSlaves	This attribute returns the number of connected slave stations. If StartConfigWithLessDevice is set to true, the number of devices can be determined.
OpenTimeout	Timeout of opening the NMS. Default value: 4s
StartConfigWithLessDevice	This attribute can affect stack start action. If five servo controllers are configured, but only three are connected, the EtherCAT stack stops immediately. However, if StartConfigWithLessDevice in the first cycle is set to true, the stack still starts. In the following scenario, if one mismatch is detected, the stack stops: Ten servo controllers are configured, the number of connected controllers is changeable, and the vendor ID and product ID of each slave station will be checked.

If supported by the device, the interface provided by IIoDrvBusControl.library can be used to access the EtherCAT device from external applications.

2 Implicit Instance of Slave Station

An ETCSlave implicit instance is created as long as the EtherCAT slave station is inserted into the device list. The instance name is the same as the device name in the device list.

The input and output parameters of instantiation objects are used for special purposes. For example, during application running, the slave station status is obtained, switched, and checked by using the slave station instances.

The definition of ETCSlave implicit instance is as follows:

ETCSlave implicit instance	
Input parameter	Description
xSetOperational	In rising edge, the slave station communication state machine is attempted to be set as ETC_ SLAVE_ OPERATIONAL.
Output parameter	Description
wState	Return the current status of the slave station. The possible values include: 0: ETC_SLAVE_BOOT 1: ETC_SLAVE_INIT 2: ETC_SLAVE_PREOPERATIONAL 4: ETC_SLAVE_SAVEOPERATIONAL 8: ETC_SLAVE_OPERATIONAL

NOTE The ETC_SLAVE_OPERATIONAL state indicates that configuration is completed. If a configuration error occurs, the device may return to the previous state. The following is an example of IS620N slave station.

ETCSlave Example

Taking 620N as an example, add the instance name 620N. Definition: nSlaveState: ETC_SLAVE_STATE;

Programming	Description
IS620N(xSetOperational:= , wState=> nSlaveState);	By invoking the slave station IS620N implicit instance, the slave station status is output to the nSlaveState variable.

Slave Station Attributes

Attribute	Description
VendorID	After the EtherCAT master station starts, this attribute returns the vendor ID read from the device.
ConfigVendorID	This attribute reads the vendor ID from configuration.
ProductID	After the EtherCAT master station starts, this attribute returns the product ID read from the device.
ConfigProductID	This attribute reads the product ID from configuration.
SerialID	After the EtherCAT master station starts, this attribute carries the device SN.
LastEmergency	If a message is received, the message is stored in the slave station. This attribute can be used to read information from application. In addition, a log message is added.

If supported by the device, the interface provided by IIoDrvBusControl.library can be used to access the EtherCAT device from external applications. After the vendor and product IDs are activated in advanced settings, if the vendor ID does not match the configured vendor ID or the product ID does not match the configured product ID, the master is stopped.

3 Checking All Slave Station Link Tables

A function block instance is created between each pair of EtherCAT master station and EtherCAT slave station in implicit way. The instance monitors the status of each slave station. Therefore, this instance must be invoked in application program. The slave station status is read by using wState. To simplify the programming, all master and slave stations can be found in link tables. Therefore, all slave stations can be checked cyclically by a simple WHILE.

The master and slave stations correspond to attributes NextInstance and LastInstance, respectively, returning the pointers to next and previous stations. In addition, the FirstSlave attribute of master station is effective. It provides the pointer to the first slave station.

Link Table Function Example

Check all slave station status. Definition: pSlave: POINTER TO ETCSlave;

Programming	Description

`pSlave := EtherCAT_Master.FirstSlave;`
`WHILE pSlave <> 0 DO`
`pSlave^();`
`IF pSlave^.wState = ETC_SLAVE_STATE.ETC_SLAVE_`
`OPERATIONAL THEN ;`
`END_IF`
`pSlave := pSlave^.NextInstance;`
`END_WHILE`
	The first slave station of the master station is found by using EtherCAT_Master. FirstSlave.
	
	Instances are invoked in WHILE loop, to determine wState. Then the status is checked.
	
	The pointer to the next slave station is found by using pSlave^.NextInstance.
	
	The pointer at the end of table is null, and the loop is finished.

4 CoE IODrvEtherCAT Function Library

CoE function block: CANOPEN over EtherCAT

After EtherCAT configuration is enabled for IODrvEtherCAT.library of EtherCAT, the library is automatically added to project, including the read/write function block. Therefore, the special parameters can be checked and modified in online mode. When the CANOPEN over EtherCAT function block is used, multiple functional modules can be invoked. Internal requests are processed in queue.

The CANOPEN over EtherCAT function block includes the following function blocks:

■ ETC_CO_SdoRead (retrieve parameter, of which the length may exceed four bytes)

■ ETC_CO_SdoRead4 (read parameter, of which the length does not exceed four bytes)

■ ETC_CO_SdoReadDword (read parameter, of which the value is stored in DWORD)

■ ETC_CO_SdoRead_Access (read all records)

■ ETC_CO_SdoRead_Channel (read slave station parameters)

■ ETC_CO_SdoWrite (write parameter, of which the length may exceed four bytes)

■ ETC_CO_SdoWrite4 (write parameter, of which the length does not exceed four bytes)

■ ETC_CO_SdoWriteDWord (value is written into DWORD)

■ ETC_CO_SdoWriteAccess (write slave station parameters)

ETC_CO_SdoRead

This functional module is provided by IODrvEtherCAT.library to read EtherCAT slave station parameters. Different from ETC_CO_SdoRead4, this module supports the parameters longer than four bytes. The read parameters are specified by object dictionary indexes and sub-indexes.

ETC_CO_SdoRead Function Block

Input parameter	Description
xExecute	In the input rising edge, read slave station parameters. To obtain the storage unit of internal channel, this instance must be invoked at least once by 'xExecute:= FALSE'.
xAbort	If this parameter is true, the read process is aborted.
usiCom	Number of EtherCAT master stations: If only one EtherCAT master station is used, usiCom is 1. If multiple master stations exist, the value of the first master station is 1, the value of the second one is 2, and so on.
uiDevice	Slave station's physical address. If the automatic configuration mode of the master station is disabled, you can configure special address for the slave station. The address selected arbitrarily must be entered. If the automatic configuration mode is enabled, the first slave station obtains address 1001. The current slave station address can be checked in the **Slave station configuration** dialog box of device editor.
usiChannel	Reserved for expansion.
wIndex	Parameter index in the object dictionary.
bySubIndex	Parameter sub-index in the object dictionary.
udiTimeout	You can set the timeout interval in millisecond. If the read parameter is not executed within the timeout, an error is prompted.
pBuffer	Data buffer pointer. Data buffer area refers to the storage area where the successfully transmitted parameters are stored.
szSize	Data buffer (see pBuffer) size, in bytes.
Output parameter	Description
xDone	The value is true when parameter reading is completed.
xBusy	The value is true when parameter reading has not been completed.
xError	The value is true if an error occurs. The eError parameter displays the error reason.
eError	The output (ETC_CO_ERROR) displays the error reason specified by xError. For example, ETC_CO_TIMEOUT indicates timeout error.
udiSdoAbort	When an error occurs in device checking, this output provides more error information.
szDataRead	Number of read bytes, namely, the maximum szSize (see input parameters).

ENUM ETC_CO_ERROR

Errors	Code	Description
ETC_CO_NO_ERROR	0	No error
ETC_CO_FIRST_ERROR	5750	The error reason is stored in udiSdoAbort.
ETC_CO_OTHER_ERROR	5751	No master station is found.
ETC_CO_DATA_OVERFLOW	5752	ETC_CO_Expedited, of which the length is larger than 4.
ETC_CO_TIME_OUT	5753	Timeout interval.
ETC_CO_FIRST_MF	5770	Not in use.
ETC_CO_LAST_ERROR	5799	Not in use.

ETC_CO_SdoRead4

This functional module is provided by IODrvEtherCAT.library to read EtherCAT slave station parameters. Different from ETC_CO_SdoRead, this functional module reads only the parameters smaller than four bytes. The read parameters are specified by object dictionary indexes and sub-indexes.

ETC_CO_SdoRead4 Function Block

ETC_CO_SdoRead4

Input parameter	Description
xExecute	In the input rising edge, read slave station parameters. To obtain the storage unit of internal channel, this instance must be invoked at least once by 'xExecute:= FALSE'.
xAbort	If this parameter is true, the read process is aborted.
usiCom	Number of EtherCAT master stations: If only one EtherCAT master station is used, usiCom is 1. If multiple master stations exist, the value of the first master station is 1, the value of the second one is 2, and so on.
uiDevice	Slave station's physical address.
	If the automatic configuration mode of the master station is disabled, you can configure special address for the slave station. The address selected arbitrarily must be entered.
	If the automatic configuration mode is enabled, the first slave station obtains address 1001. The current slave station address can be checked in the **Slave station configuration** dialog box of device editor.
usiChannel	Reserved for expansion.
wIndex	Parameter index in the object dictionary.
bySubIndex	Parameter sub-index in the object dictionary.
udiTimeout	You can set the timeout interval in millisecond. If the read parameter is not executed within the timeout, an error is prompted.
Output parameter	Description
wState	Return the current status of the slave station. The possible values include:
	0: ETC_SLAVE_BOOT
	1: ETC_SLAVE_INIT
	2: ETC_SLAVE_PREOPERATIONAL
	4: ETC_SLAVE_SAVEOPERATIONAL
	8: ETC_SLAVE_OPERATIONAL

ETC_CO_SdoRead4 Function Block	
xDone	The value is true when parameter reading is completed.
xBusy	The value is true when parameter reading has not been completed.
xError	The value is true if an error occurs. The eError parameter displays the error reason.
eError	The output (ETC_CO_ERROR) displays the error reason specified by xError. For example, ETC_CO_TIMEOUT indicates timeout error.
abyData	The read parameter data is copied to this 4-byte array. If the first byte has been read, it is stored in the first index of the array. The 2 or 4-byte data is copied to this array in Intel byte order.
usiDataLength	Number of read bytes (1, 2, or 4).

ENUM ETC_CO_ERROR

Errors	Code	Description
ETC_CO_NO_ERROR	0	No error
ETC_CO_FIRST_ERROR	5750	The error reason is stored in udiSdoAbort.
ETC_CO_OTHER_ERROR	5751	No master station is found.
ETC_CO_DATA_OVERFLOW	5752	ETC_CO_Expedited, of which the length is larger than 4.
ETC_CO_TIME_OUT	5753	Timeout interval.
ETC_CO_FIRST_MF	5770	Not in use.
ETC_CO_LAST_ERROR	5799	Not in use.

ETC_CO_SdoReadDword

This function block is provided by IODrvEtherCAT.library. Similar to ETC_CO_SdoRead4, it reads EtherCAT slave station parameters. However, the read data is copied to DWORD (dwData), rather than array. Byte exchange is automatically carried out. Therefore, the read data can be used by subsequential processes.

ETC_CO_SdoRead_Access

This function block is provided by IODrvEtherCAT.library. Similar to ETC_CO_SdoRead4, it reads EtherCAT slave station parameters. All record indexes can be read by inputting xCompleteAccess (BOOL). Therefore, xCompleteAccess must be set to true, and bySubIndex must be 0.

ETC_CO_SdoRead_Channel

The EtherCAT programming interface in EtherCAT configuration editor is used by the ETC_CO_SdoRead_Channel function block of IODrvEtherCAT in CAN over EtherCAT.

This function block is provided by IODrvEtherCAT.library. Similar to ETC_CO_SdoRead_Access, it reads EtherCAT slave station parameters.

However, it has an additional input byChannelPriority (BYTE) that specifies the channel and priority in CoE email box message. The first 6 bits specify the channel, and the later 2 bits specify the priority.

ETC_CO_SdoWrite

This functional module is provided by IODrvEtherCAT.library to read EtherCAT slave station parameters. Different from ETC_CO_SdoWrite 4, this functional module reads only the parameters larger than four bytes. The written parameters are specified by object dictionary indexes and sub-indexes.

ETC_CO_SdoWrite Function Block

Input parameter	Description
xExecute	In the input rising edge, write slave station parameters. To obtain the storage unit of internal channel, this instance must be invoked at least once by 'xExecute:= FALSE'.
xAbort	If this parameter is true, the write process is aborted.
usiCom	Number of EtherCAT master stations: If only one EtherCAT master station is used, usiCom is 1. If multiple master stations exist, the value of the first master station is 1, the value of the second one is 2, and so on.
uiDevice	Slave station's physical address. If the automatic configuration mode of the master station is disabled, you can configure special address for the slave station. The address selected arbitrarily must be entered. If the automatic configuration mode is enabled, the first slave station obtains address 1001. The current slave station address can be checked in the **Slave station configuration** dialog box.
usiChannel	Reserved for expansion.
wIndex	Parameter index in the object dictionary.
bySubIndex	Parameter sub-index in the object dictionary.
udiTimeout	You can set the timeout interval in millisecond. If the write parameter is not executed within the timeout, an error is prompted.
pBuffer	Data buffer pointer. Data buffer area refers to the storage area where the successfully transmitted parameters are stored.
szSize	Data buffer (see pBuffer) size, in bytes.
eMode	This input (enumeration: ETC_CO_MODE) defines the number of written bytes. The possible values include ETC_CO_AUTO (automatic),ETC_CO_EXPEDITED (acceleration), and ETC_CO_SEGMENTED (segmented). Generally, the ETC_CO_AUTO mode is used because the data length is automatically matched in this mode.
Output parameter	Description
xDone	The value is true when parameter writing is completed.
xBusy	The value is true when parameter writing has not been completed.
xError	The value is true if an error occurs. The eError parameter displays the error reason.
eError	The output (ETC_CO_ERROR) displays the error reason specified by xError. For example, ETC_CO_TIMEOUT indicates timeout error.
udiSdoAbort	When the device has an error, this output provides more error information.
szDataWritten	Number of written bytes. After bytes are successfully written, it is set to szSize.

ENUM ETC_CO_MODE

Mode	No.	Description
AUTO	0	Automatic mode is selected.
EXPEDITED	1	Acceleration protocol is used.

Mode	No.	Description
SEGMENTED	2	Segmented transmission protocol is used.

ETC_CO_SdoWrite4

This functional module is provided by IODrvEtherCAT.library to read EtherCAT slave station parameters. Different from ETC_CO_SdoWrite, this functional module reads only the parameters smaller than four bytes. The written parameters are specified by object dictionary indexes and sub-indexes.

ETC_CO_SdoWrite4 Function Block

Input parameter	Description
xExecute	In the input rising edge, write slave station parameters. To obtain the storage unit of internal channel, this instance must be invoked at least once by 'xExecute:= FALSE'.
xAbort	If this parameter is true, the write process is aborted.
usiCom	Number of EtherCAT master stations: If only one EtherCAT master station is used, usiCom is 1. If multiple master stations exist, the value of the first master station is 1, the value of the second one is 2, and so on.
uiDevice	Slave station's physical address.
	If the automatic configuration mode of the master station is disabled, you can configure special address for the slave station. The address selected arbitrarily must be entered.
	If the automatic configuration mode is enabled, the first slave station obtains address 1001. The current slave station address can be checked in the **Slave station configuration** dialog box.
usiChannel	Reserved for expansion.
wIndex	Parameter index in the object dictionary.
bySubIndex	Parameter sub-index in the object dictionary.
udiTimeout	You can set the timeout interval in millisecond. If the write parameter is not executed within the timeout, an error is prompted.
abyData	This array includes four written data records. The data must be stored in the Intel byte order.
usiDataLength	Number of written bytes (1, 2, or 4).
Output parameter	Description
xDone	This output is set to true when parameter writing is completed.
xBusy	The output is true if the write operation is not completed.
xError	If an error occurs, this output is set to true. eError displays the error reason.
eError	This output (ETC_CO_ERROR) displays the error reason, identified as xError. For example, ETC_CO_TIMEOUT indicates timeout.
udiSdoAbort	If the device has an error, this output provides more error information.

ENUM ETC_CO_MODE

Mode	No.	Description
AUTO	0	Automatic mode is selected.
EXPEDITED	1	Acceleration protocol is used.
SEGMENTED	2	Segmented transmission protocol is used.

ETC_CO_SdoWriteDWord

This function block is provided by IODrvEtherCAT.library. Similar to ETC_CO_SdoWrite4, it writes EtherCAT slave station parameters. However, the data to be written is output in DWORD format (dwData), rather than array. Byte exchange is automatically carried out. The value to be written can be specified.

ETC_CO_SdoWriteAccess

This function block is provided by IODrvEtherCAT.library. Similar to ETC_CO_SdoWrite, it writes EtherCAT slave station parameters.

All record indexes can be written by inputting xCompleteAccess (BOOL). Therefore, xCompleteAccess must be set to true, and bySubIndex must be 0. However, it has an additional input byChannelPriority (BYTE) that specifies the channel and priority in CoE email box message.
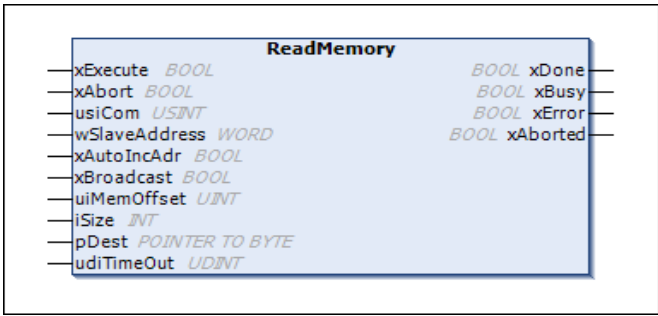
5 Direct Access to EtherCAT Slave Station Memory

Choose EtherCAT configuration editor > EtherCAT programming interface to directly access the EtherCAT slave station memory.

Access EtherCAT slave station memory: ReadMemory and WriteMemory.

■　ReadMemory

This function block is in IODrvEtherCAT.library to read data in EtherCAT slave station memory.

ReadMemory Function Block

Input parameter	Type	Description
xExecute	BOOL	Rising edge: action starts. Falling edge: reset output. If there is a falling edge before the function block completes action, the output operation is performed in a normal way and a reset is performed when the action is completed or aborted (xAbort). In this situation, the related output values (xDone, xError, and iError) are output within a cycle.
xAbort	BOOL	If the input is true, the action is aborted immediately and all outputs are reset to the initial values.
usiCom	USINT	Master index 1: The first EtherCAT master station.
wSlaveAddress	WORD	Automatically created address or device physical address.
xAutoIncAdr	BOOL	Flag confirming that the address mode is used.
xBroadcast	BOOL	If the card mode is used and the value is true, wSlaveAddress and bAutoIncAdr will not be used.

ReadMemory Function Block		
uiMemOffset	UINT	Memory address offset.
iSize	INT	Read byte.
pDest	POINTER TO BYTE	Data storage and retrieve buffer.
udiTimeOut	UDINT	Operation timeout (ms)
Output parameter	Type	Description
xDone	BOOL	Action completed successfully.
xBusy	BOOL	Function block activated.
xError	BOOL	TRUE: An error is generated and the function block aborts action; FALSE: No error.
xAborted	BOOL	Abort action.

Example: Read register 0x130 (current state)

```
PROGRAM PLC_PRG
VAR
    etcreadmemory : ReadMemory;
    wStatus : WORD;
    xRead : BOOL;
END_VAR


etcreadmemory(xExecute := xRead, usiCom:=1, wSlaveAddress := 1002,
        xAutoIncAdr := FALSE, xBroadcast := FALSE, uiMemOffset := 16#130,
        iSize := 2, pDest := ADR(wStatus), udiTimeout := 500);
```

■　WriteMemory

This function block is in   IODrvEtherCAT.library to write data into EtherCAT slave station memory.

WriteMemory Function Block		
Input parameter	Type	Description
xExecute	BOOL	Rising edge: action starts. Falling edge: reset output. If there is a falling edge before the function block completes action, the output operation is performed in a normal way and a reset is performed when the action is completed or aborted (xAbort). In this situation, the related output values (xDone, xError, and iError) are output within a cycle.
xAbort	BOOL	If the input is true, the action is aborted immediately and all outputs are reset to the initial values.
usiCom	USINT	Master index 1: The first EtherCAT master station.
wSlaveAddress	WORD	Automatically created address or device physical address.
xAutoIncAdr	BOOL	Flag confirming that the address mode is used.

WriteMemory Function Block		
xBroadcast	BOOL	If the card mode is used and the value is true, wSlaveAddress and bAutoIncAdr will not be used.
uiMemOffset	UINT	Memory address offset.
iSize	INT	Write byte.
pDest	POINTER TO BYTE	Read data in and retrieve data from data buffer.
udiTimeOut	UDINT	Operation timeout (ms)
Output parameter	Type	Description
xDone	BOOL	Action completed successfully.
xBusy	BOOL	Function block activated.
xError	BOOL	TRUE: An error is generated and the function block aborts action; FALSE: No error.
xAborted	BOOL	Abort action.

Example: Write register 0x120 (control register)

```
PLC_PRG
VAR
    etcwritememory : WriteMemory;
    xWrite : BOOL;
    wCommand : WORD := 4; // set to safe operational
END_VAR
etcwritememory(xExecute := xWrite, usiCom:=1, wSlaveAddress := 1002,
          xAutoIncAdr := FALSE, xBroadcast := FALSE, uiMemOffset := 16#120,
          iSize := 2, pSrc := ADR(wCommand), udiTimeout := 500);
```

3.4 Modbus Editor

Click the PLC in Network Configuration. The master and slave station windows supported by PLC are displayed. As shown in the following figure, select the check box before the master or slave you want to enable, and double-click **MODBUS** in **Network Devices List** on the right to add the slave to the network.
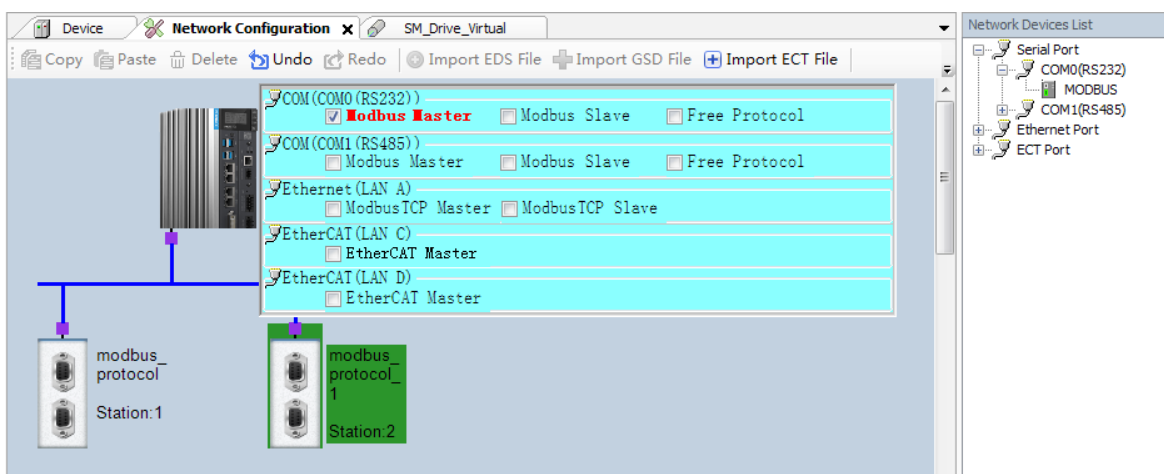
Figure 3-29 Modbus Configuration Example

The device tree corresponding to the Modbus configuration is displayed on the left of the interface, shown as follows:
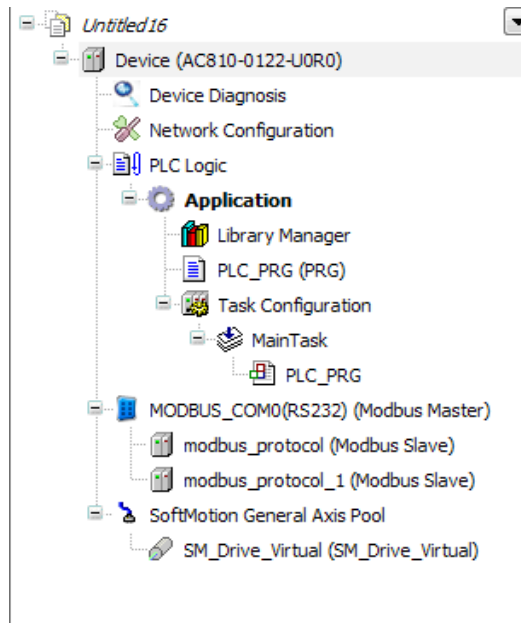
Figure 3-30 Device tree corresponding to Modbus configuration

AM600 supports two channels of Modbus serial port communication, matching serial port 0 and serial port 1. Both ports support standard Modbus RTU protocol, and can be configured as master or slave station. They support 6 baud rates, including 4800, 9600, 19200, 38400, 57600, and 115200.

The variable ranges that can be accessed by the master station are as follows:

1) All bit variable operations (01 02 05 0f) can read and write 65535 bit variables from %QX0.0 to %QX8191.7.

2) All register variable operations (03 04 06 10) can read and write 65536 register variables from MW0 to MW65535.

3) Inovance HMI can access AM600 system variables SM0-SM7999 and register variables SD0-SD7999.

3.4.1 Modbus Master Station Configuration

When a serial port is used as master station, configure Modbus master station and TCP of Modbus master station in the Modbus device editor, including the following parameters:

Modbus master configuration
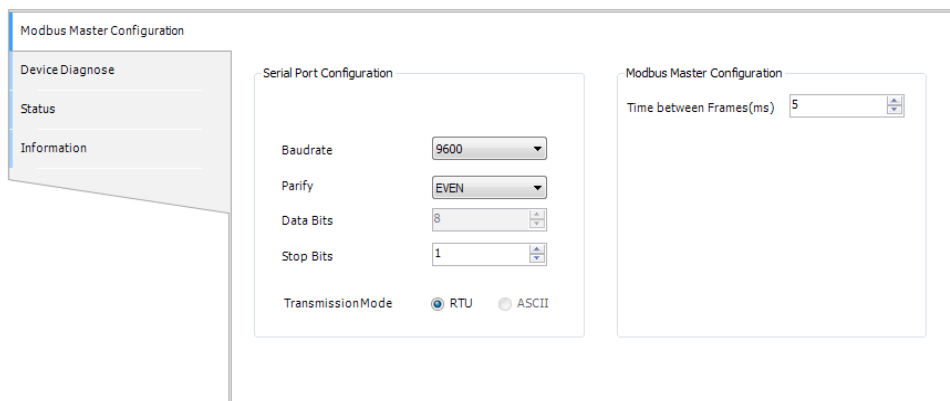
Figure 3-31   Modbus master configuration

Master configuration parameters

Parameter	Function
COM Port	COM port 0 or 1, which is used to establish a physical connection to the master
Baudrate	Communication rate
Purify	Method of verifying communication frames
Data Bit	Actual data bits included in communication frames
Stop Bit	Last bit in a single packet during communication
TransmissionMode	RTU
Time Between Frames	The time for the master to wait for the next request data frame after receiving a response data frame

Example:

Parameter	Value
COM Port	0
Baudrate	115200
Purify	EVEN
Data Bit	8
Stop Bit	1
TransmissionMode	RTU
Time Between Frames	2 ms

3.4.2 Modbus Master-Slave Connection Configuration

When a serial port functions as the master station, the Modbus slave station is configured in Modbus device editor, involving the following parameters:
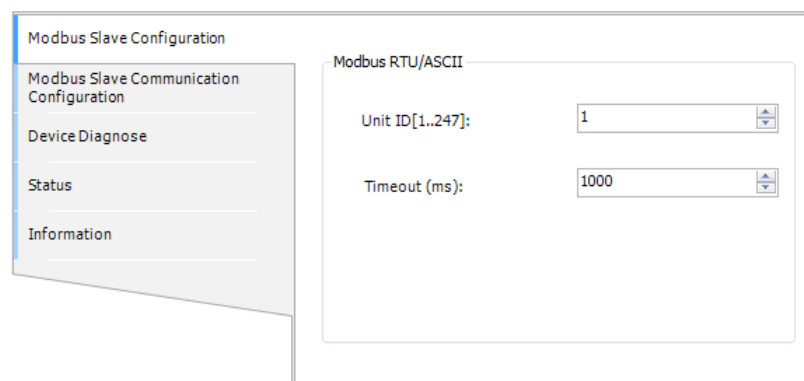
Figure 3-32 Modbus slave station configuration when port functions as master

Slave configuration parameters

Parameter	Function
Unit ID	ID of the slave station, ranging from 1 to 247
Timeout	After sending data, the master reports receiving timeout if no data is received within this timeout period
Slave Enable Variable	Enables the slave station by programming and starts to send frame to the slave station. It is effective when it is set to ON.

Example:

Parameter	Value
Unit ID	11
Timeout	1000 ms
Slave Enable Variable	1001

Modbus slave station communication configuration when port functions as master
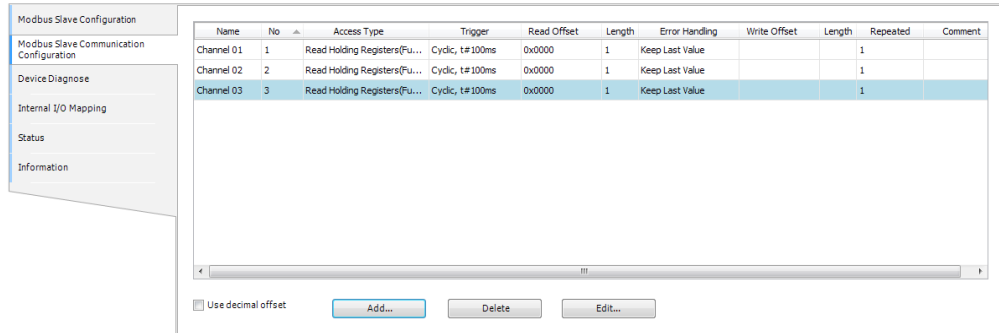
Figure 3-33 Modbus slave station communication configuration when port functions as master

In the previous figure, each channel represents an independent Modbus request. The third column defines the cyclic operation on a write-single register (function code: 06) to write a word to the register with an offset of 0x0006.

After you click **Add**, a dialog box for adding a channel for the Modbus slave station is displayed. Click **OK** to create a channel.

Select a channel from the Modbus slave channel list and click **Edit...** The **Modbus Channel Set** dialog box is displayed. Change the values of parameters to modify the channel settings. Click **OK** to update the channel settings. You can set the following parameters to add or edit a channel:
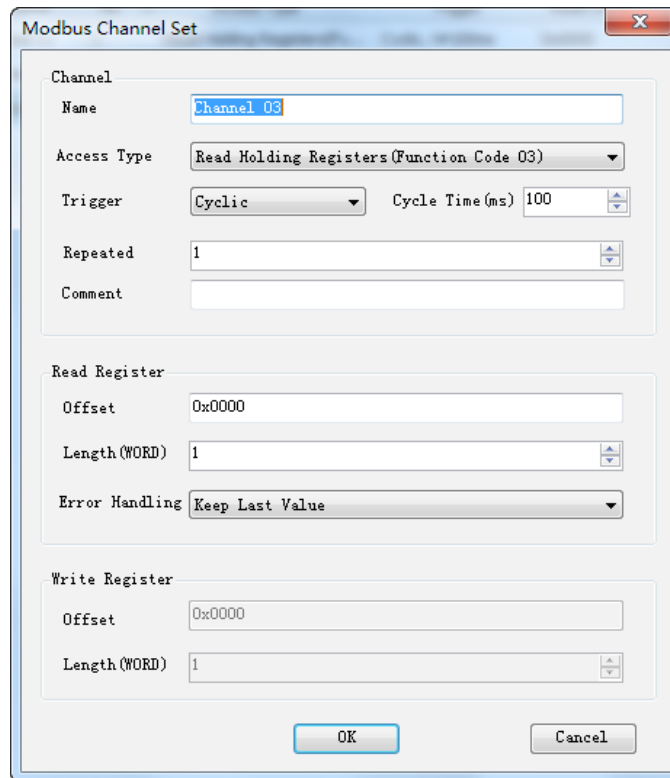
Figure 3-34 Modbus slave station communication configuration when port functions as master

Modbus communication parameter settings

Parameter	Function	
Name	Channel name, in the string format.	
Access Type	Read Coils (Function Code 01). Read Discrete Inputs (Function Code 02). Read Holding Registers (Function Code 03). Read Input Registers (Function Code 04). Write Single Coil (Function Code 05). Write Single Register (Function Code 06). Write Multiple Coils (Function Code 15). Write Multiple Registers (Function Code 16).	
Trigger	**Cyclic**: Requests are triggered periodically.	**Cycle Time**: time for re-execution.
	Level Trigger: Requests are triggered when a change is made during programming.	**Trigger Variable (SM)**: SM element that implements trigger. After trigger is successful, the element is reset automatically.
Repeated	A request is resent for the specified times when no response frame is received from the slave upon a communication error.	
Comment	Brief text description about data.	
Read Register		
Offset	Head address of the registers to be read.	
Length	Number of registers to be read.	
Error Handling	**Keep Last Value**: The last valid value is kept. **0**: All the values are zeroed.	
Maximum number of registers to be written		
Offset	Head address of the registers to be written.	
Length	Number of registers to be written.	

The valid range of the **Length** parameter depends on the following parameters:

Function Code	Access Type	Register Count
01	Read Coils.	1 to 2000
02	Read Discrete Inputs.	1 to 2000
03	Read Holding Registers.	1 to 125
04	Read Input Registers.	1 to 125
05	Writes one coil.	1
06	Writes one register.	1
15	Writes multiple coils.	1 to 1968
16	Write Multiple Registers.	1 to 123

Modbus Slave Internal I/O Mapping
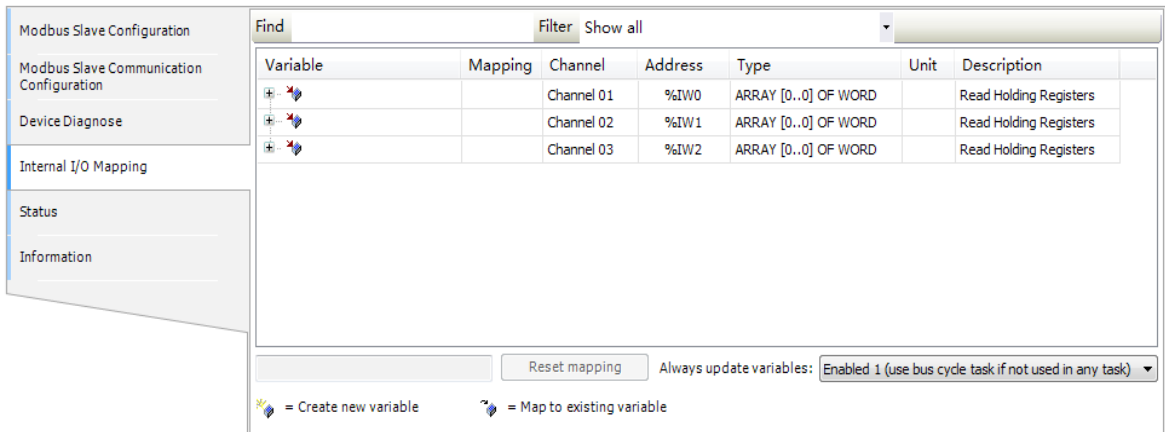
Figure 3-35 Modbus slave internal I/O mapping when port functions as master station

Map variables to I/Q addresses by using the input assistant or entering an instance variable path.

3.4.3 Modbus Master Station Broadcast Configuration

When the Modbus master station is connected to multiple Modbus slave stations, and all Modbus slave stations receive write operation, the Modbus master station needs to carry out broadcast.
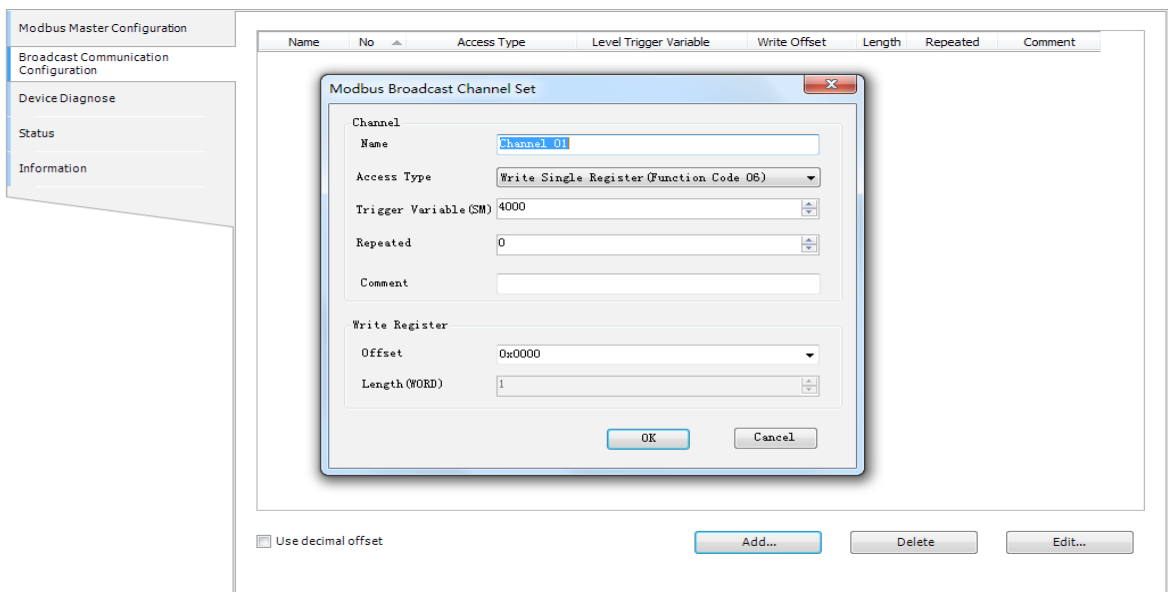
Figure 3-36 Broadcast configuration when port functions as master

By clicking **Add** on the broadcast communication setting tab, the **Modbus Broadcast Channel Set** dialog box is displayed, including the parameters of Name, Access Type, Trigger Variable (SM), Repeated, Comment, and Write Register (Offset and Length).

The access type includes multiple function codes, including Write Single Coil (Function Code 05), Write Single Register (Function Code 06), Write Multiple Coils (Function Code 15), and Write Multiple Registers (Function Code 16).

■ **Trigger Variable**: Condition that triggers the Modbus master to start communication. The Modbus master can perform broadcast communication only when the trigger variable is true. The trigger variable needs to be reset during programming.

■ **Repeated**: Number of resend times after a send is completed. The number of send failures can be reduced by setting the number of resend times.

3.4.4 Modbus Slave Station Configuration
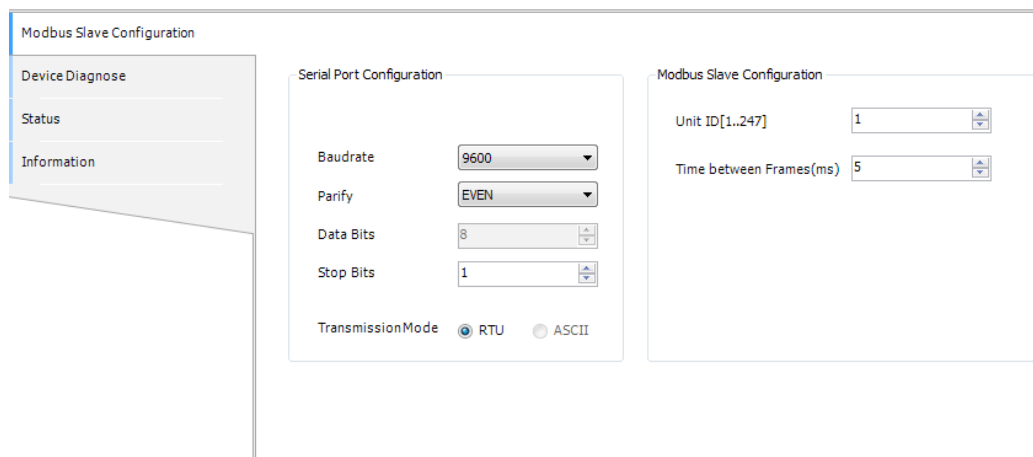
Figure 3-37 Configuration when port functions as slave

Among the Modbus slave station configuration parameters, the serial port configuration parameters have the same meanings as those of Modbus master station. The Modbus slave station number refers to the local device station number. Time between frame indicates the delay of responding to the master after the frame from master is received.

3.4.5 Modbus Device Diagnosis

The Modbus master station diagnosis information includes the communication configurations of faulty slave station and the fault.
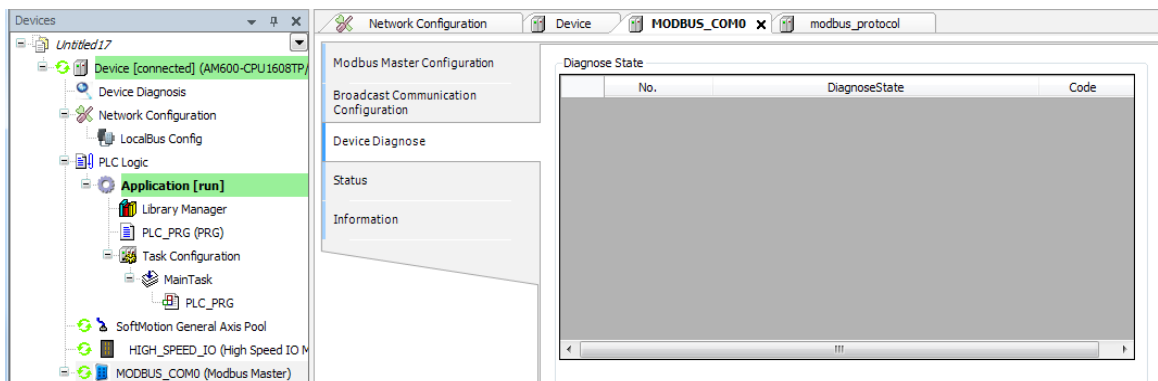
Figure 3-38 Modbus master diagnosis
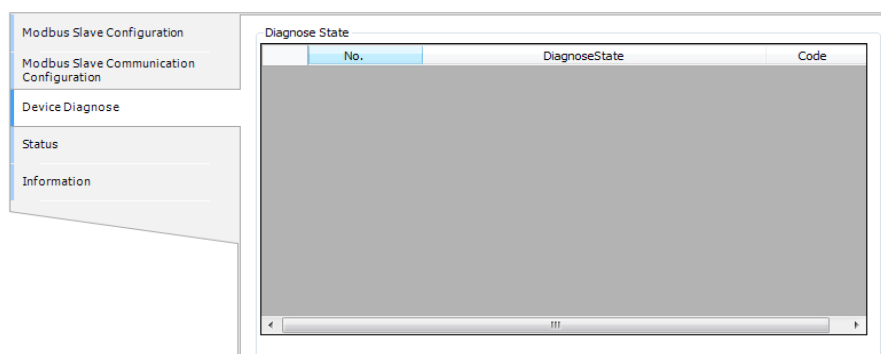
Figure 3-39 Modbus slave station diagnosis when port functions as master
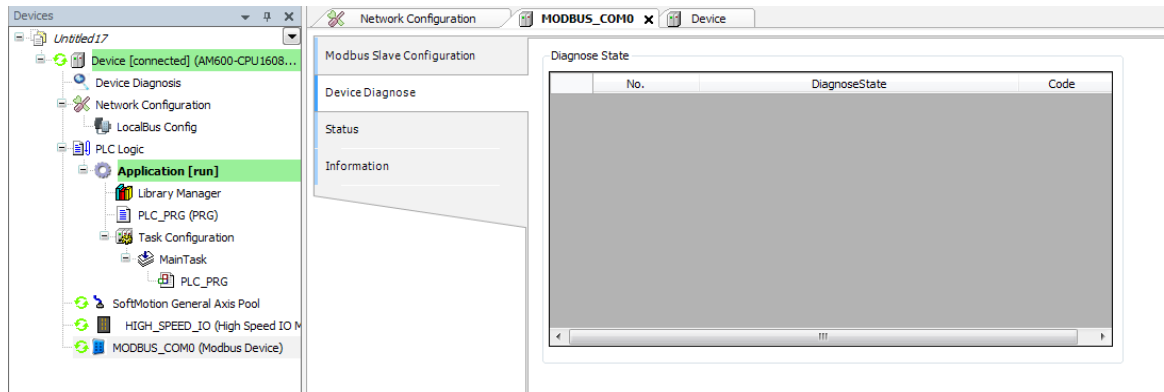
Figure 3-40　Device diagnosis when port functions as slave

3.4.6 Common Errors of Modbus

The following errors are frequently encountered during Modbus master-slave connection:

1. The configurations of Modbus master and Modbus slave are inconsistent, causing a communication failure between master and slave.

2. An error response is returned when the Modbus master accesses a Modbus slave through an invalid address.

3. The Modbus master receives an error response from the Modbus slave when it attempts to write a register of the Modbus slave that only supports the read operation.

Incorrect response frame

An error response consists of a transaction metadata identifier, protocol identifier, length, slave address, command code+0x80, error code, and cyclic redundancy check (CRC).

The preceding error frame is applicable to all command frames.

No.	Definition of Data (Byte)	Number of Bytes	Description
1	Transaction metadata identifier	2	Identifier of Modbus request/response transaction processing.
2	Protocol identifier	2	Modbus protocol if the value is **0**.
3	Length	2	Number of the following bytes.
4	Slave address	1	Value range: 1 to 247.
5	Command code+0x80	1	Command error code
6	Error code	1	The value ranges from 1 to 4.

3.4.7 Modbus Variable Addressing

Coil: Bit variables, indicated by 0 or 1. The PLC includes Q and SM area variables.

Variable	Command Code	Offset	Number of Coils	Description
QW0-QW511	0X01, 0x05, 0x0f	0	8192	Accessible by standard Modbus protocols.
SM0-SM7999	0X31, 0x35, 0x3f	0	8000	Use different function codes from Inovance HMI dedicated protocol.

Register: 16-bit (words) variable. The PLC includes M and SD area variables.

Variable	Command Code	Offset	Number of Registers	Description
MW0-MW65535	0X03, 0x06, 0x10	0	65536	Accessible by standard Modbus protocols.
SD0-SD7999	0X33, 0x36, 0x40	0	8000	Use different function codes from Inovance HMI dedicated protocol.

Note:

Inovance HMI dedicated protocol uses different function codes: For the access to SM, use 0x31, 0x35 and 0x3f (0x30 added based on bit variable access). For the access to SD, use 0x33, 0x36, and 0x40 (0x30 added based on register variable access).

AM600 soft elements include Q, I, and M areas, which can be accessed by bit, byte word, and dual-word. For example, %QX, %QB, %QW, and %QD are converted as follows:

QB0 = (QX0.0-QX0.7)

QW0 = (QB0-QB1) = ((QX0.0-QX0.7) + (QX1.0-QX1.7));

QD0 = (QW0-QW1) = (QB0-QB4) = ((QX0.0-QX0.7) + (QX1.0-QX1.7)+(QX2.0-QX2.7)+(QX3.0-QX3.7))

Register Addressing Rule

Addressing by bit	Addressing by byte	Addressing by word	Addressing by Dword	Addressing by bit	Addressing by byte	Addressing by word	Addressing by Dword
QX0.0	QB0	QW0	QD0	MX0.0	MB0	MW0	MD0
QX0.1				MX0.1			
QX0.2				MX0.2			
QX0.3				MX0.3			
QX0.4				MX0.4			
QX0.5				MX0.5			
QX0.6				MX0.6			
QX0.7				MX0.7			
QX1.0	QB1			MX1.0	MB1		
QX1.1				MX1.1			
QX1.2				MX1.2			
QX1.3				MX1.3			
QX1.4				MX1.4			
QX1.5				MX1.5			
QX1.6				MX1.6			
QX1.7				MX1.7			
QX2.0	QB2	QW1		MX2.0	MB2	MW1	
QX2.1				MX2.1			
QX2.2				MX2.2			
QX2.3				MX2.3			
QX2.4				MX2.4			
QX2.5				MX2.5			
QX2.6				MX2.6			
QX2.7				MX2.7			
QX3.0	QB3			MX3.0	MB3		
QX3.1				MX3.1			
QX3.2				MX3.2			
QX3.3				MX3.3			
QX3.4				MX3.4			
QX3.5				MX3.5			
QX3.6				MX3.6			
QX3.7				MX3.7			
QX4.0	QB4	QW2	QD1	MX4.0	MB4	MW2	MD1
QX4.1				MX4.1			

The head address of AM600's Word register contains an even number of bytes. The head address of DWord register contains an even number of words. The index number is 2 times, facilitating address calculation.

3.4.8 Modbus Communication Frame Format

The 0x01 command code is used to read the Q variable.

The 0x31 command code is used to read the SM variable.

Request frame format: slave address + 0x01 + head address of coils + number of coils + CRC code

No.	Definition of Data (Byte)	Number of Bytes	Description
1	Slave address	1	Value range: 1 to 247.
2	0x01/0x31 (command code)	1	The instruction is to read coils.
3	Head address of coils	2	Big-endian. See "Coil addressing".
4	Number of coils	2	Big-endian (N).
5	CRC code	2	Big-endian.

Response frame format: slave address + 0x01 + number of bytes + state of coils + CRC code

No.	Definition of Data (Byte)	Number of Bytes	Description
1	Slave address	1	Value range: 1 to 247.
2	0x01/0x31 (command code)	1	The instruction is to read coils.
3	Number of bytes	1	The value is (N + 7)/8.
4	Coil status	(N + 7)/8	Eight coils are indicated by one byte. If the last byte has less than eight bits, enter 0 for undefined bits. The first eight coils are indicated by the first byte, and the coil with the smallest address is indicated by the least significant bit.
5	CRC code	2	Big-endian.

The 0x03 command code is used to read the M variable.

The 0x33 command code is used to read the SD variable.

Request frame format: slave address + 0x03 + head address of registers + number of registers + CRC code

No.	Definition of Data (Byte)	Number of Bytes	Description
1	Slave address	1	Value range: 1 to 247.
2	0x03/0x33 (command code)	1	The instruction is to read registers.
3	Head address of registers	2	Big-endian. See "Register addressing".
4	Number of registers	2	Big-endian (N).
5	CRC code	2	Big-endian.

Response frame format: slave address + 0x03 + number of bytes + register value + CRC code

No.	Definition of Data (Byte)	Number of Bytes	Description
1	Slave address	1	Value range: 1 to 247.
2	0x03/0x33 (command code)	1	The instruction is to read registers.
3	Number of bytes	1	The value is N x 2
4	Register value	N x 2	One register value is indicated by two bytes. It is big-endian. The register with a smaller address is indicated by the first byte.
5	CRC code	2	Big-endian.

The 0x05 command code is used to read the Q variable.

The 0x35 command code is used to read the SM variable.

Request frame format: slave address + 0x05 + head address of the coil + state of the coil + CRC code

No.	Definition of Data (Byte)	Number of Bytes	Description
1	Slave address	1	Value range: 1 to 247.
2	0x05/0x35 (command code)	1	The instruction is to write a single coil.
3	Address of the coil	2	Big-endian. See "Coil addressing".
4	Coil status	2	Big-endian. A non-zero value is valid.
5	CRC code	2	Big-endian.

Response frame format: slave address + 0x05 + head address of the coil + state of the coil + CRC code

No.	Definition of Data (Byte)	Number of Bytes	Description
1	Slave address	1	Value range: 1 to 247.
2	0x05/0x35 (command code)	1	The instruction is to write a single coil.
3	Address of the coil	2	Big-endian. See "Coil addressing".
4	Coil status	2	Big-endian. A non-zero value is valid.
5	CRC code	2	Big-endian.

The 0x06 command code is used to read the M variable.

The 0x36 command code is used to read the SD variable.

Request frame format: slave address + 0x06 + head address of the register + register value + CRC code

No.	Definition of Data (Byte)	Number of Bytes	Description
1	Slave address	1	Value range: 1 to 247.
2	0x06/0x36 (command code)	1	The instruction is to write a single register.
3	Address of the register	2	Big-endian. See "Register addressing".
4	Register value	2	Big-endian. A non-zero value is valid.
5	CRC code	2	Big-endian.

Response frame format: slave address + 0x06 + head address of the register + register value + CRC code

No.	Definition of Data (Byte)	Number of Bytes	Description
1	Slave address	1	Value range: 1 to 247.
2	0x06/0x36 (command code)	1	The instruction is to write a single register.
3	Address of the register	2	Big-endian. See "Register addressing".
4	Register value	2	Big-endian. A non-zero value is valid.
5	CRC code	2	Big-endian.

The 0x0f command code is used to write multiple consecutive Q variables.

The 0x3f command code is used to write multiple consecutive Q variables.

Request frame format: slave address + 0x0f + head address of coils + number of coils + number of bytes + state of coils + CRC code

No.	Definition of Data (Byte)	Number of Bytes	Description
1	Slave address	1	Value range: 1 to 247.
2	0x0f/0x3f (command code)	1	The instruction is to write multiple coils.
3	Head address of coils	2	Big-endian. See "Coil addressing".
4	Number of coils	2	Big-endian. The maximum of N is 1968.
5	Number of bytes	1	The value is (N + 7)/8.
6	Coil status	(N + 7)/8	Eight coils are indicated by one byte. If the last byte has less than eight bits, enter 0 for undefined bits. The first eight coils are indicated by the first byte, and the coil with the smallest address is indicated by the least significant bit.
7	CRC code	2	Big-endian.

Response frame format: slave address + 0x05 + head address of coils + number of coils + CRC code

No.	Definition of Data (Byte)	Number of Bytes	Description
1	Slave address	1	Value range: 1 to 247.
2	0x0f/0x3f (command code)	1	The instruction is to write multiple coils.
3	Head address of coils	2	Big-endian. See "Coil addressing".
4	Number of coils	2	Big-endian.
5	CRC code	2	Big-endian.

The 0x10 command code is used to write multiple consecutive M variables.

The 0x40 command code is used to write multiple consecutive SD variables.

Request frame format: slave address + 0x10 + head address of registers + number of registers + number of bytes + register value + CRC code

No.	Definition of Data (Byte)	Number of Bytes	Description
1	Slave address	1	Value range: 1 to 247.
2	0x10/0x40 (command code)	1	The instruction is to write multiple registers.
3	Head address of registers	2	Big-endian. See "Register addressing".
4	Number of registers	2	Big-endian. The maximum of N is 120.
5	Number of bytes	1	The value is N x 2
6	Register value	N x 2 (N x 4)	
7	CRC code	2	Big-endian.

Response frame format: slave address + 0x05 + head address of coils + number of coils + CRC code

No.	Definition of Data (Byte)	Number of Bytes	Description
1	Slave address	1	Value range: 1 to 247.
2	0x10/0x40 (command code)	1	The instruction is to write multiple registers.
3	Head address of registers	2	Big-endian. See "Register addressing".
4	Number of registers	2	Big-endian. The maximum of N is 120.
5	CRC code	2	Big-endian.

3.5 Using Free Protocols on COM Ports
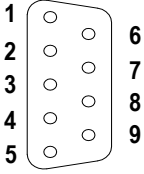
3.5.1 Overview

This section describes communication using free protocols on the COM ports of a medium-sized PLC. The applicable versions are as follows:
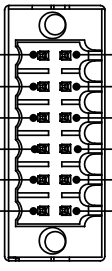
Device Name	Version
InoProShop	1.2.0 and later
PLC firmware	1.19.70 and later

3.5.2 Serial Hardware Port

AM600 supports communication using two RS485 COM ports, which are port 0 and port 1, with support for free protocols.

Port	Channel	Pin	Definition
	COM0 (RS485)	1	RS485-
		2	RS485+
		5	GND
	COM1 (RS485)	6	RS485-
		9	RS485+
		3	GND

AC810 supports RS485 and RS232 communication. For I/O COM ports, pins 1, 3, and 5 output DI signals, and pins 2 and 4 output DO signals.

Description	Function	Signal	No.	I/O COM Port	No.	Signal	Function	Description
ON: power-on; OFF: power-off	Power-on signal	⏻	1		2	P_STATUS	Indicator turn-on signal upon power-on	Output after the controller is powered on
Retentive at power failure upon ON-OFF switch	Power failure detection signal	P_OK	3		4	P_STATUS	Running status signal	Output after the controller is powered on
OFF: RUN; ON: STOP	RUN/STOP	RUN	5		6	0 V	Output common	--
--	Input common	0 V	7		8	GND	Communication reference ground	--
--	RS485+	485+	9		10	232R	RS232 receiving	--
--	RS485-	485-	11		12	232T	RS232 sending	--

3.5.3 COM Port Configuration

This section shows how to configure COM1 of the PLC to use free protocols.
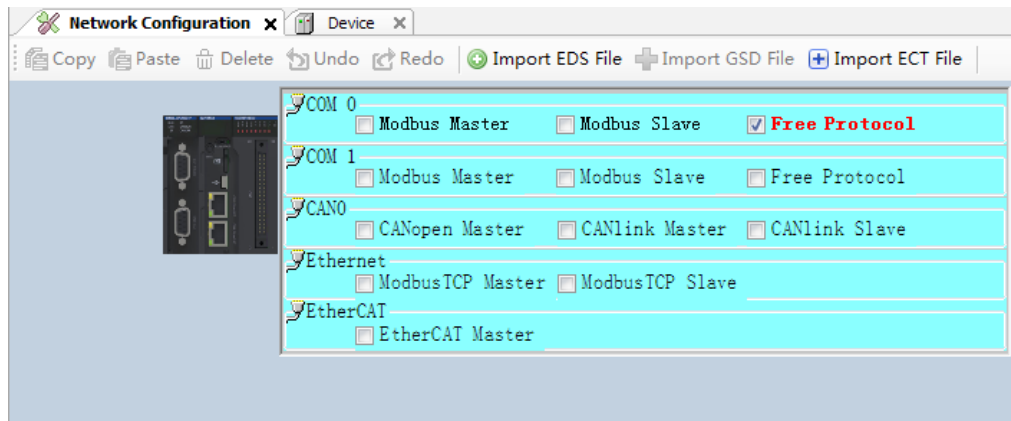
3.5.4 Communication Configuration
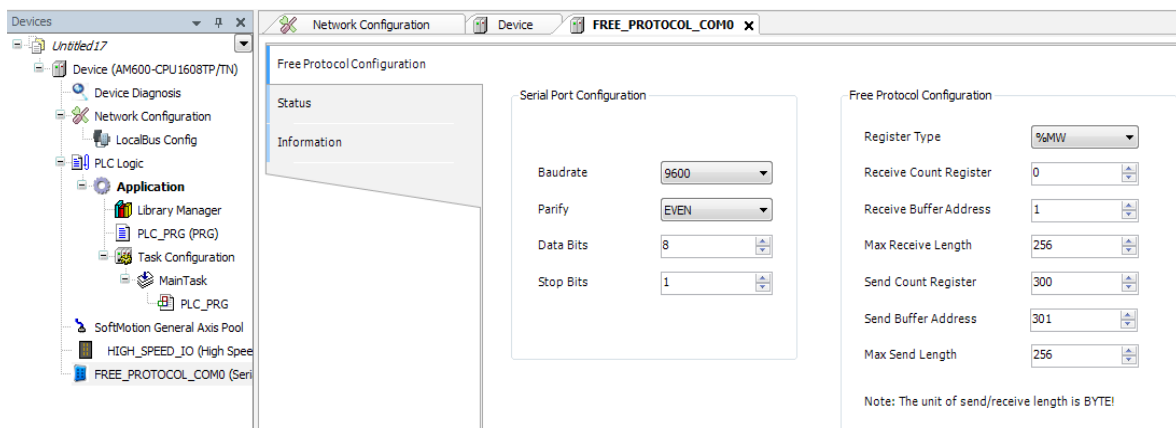
The following table lists the parameters that need to be set.

Parameter	Definition
COM Port Number	COM port 0 or 1, which is used to establish a physical connection to the master
Baudrate	Communication rate
Parify	Method of verifying communication frames
Data Bits	Actual data bits included in communication frames
Stop Bits	Last bit in a single packet during communication

The communication format used by the COM port with **Free Protocol** selected must be consistent with that of the connected slave, for example, 9600 8 E 1.

3.5.5 Registers for Data Sending and Receiving

■ **Register Type**: The options are **%MW** and **SD**. The **SD** option is only applicable to the AM600 series.

■ **Receive Count Register**: records the length (in bytes) of data frames received from external devices.

In Figure 3-41, the **%MW0** setting indicates the length of data frames received from external devices. MW0 needs to be cleared manually; otherwise, its value keeps increasing until it is cleared when the value of **Max Receive Length** is reached. In this case, the receive buffer is overwritten from the start.

■ **Receive Buffer Address**: records the head address (in bytes) of the buffer that receives data from external devices.

For example, if the receive length is **%MW0** = **10**, the receive buffer ranges from %MW1 to %MW5. One %MW occupies 2 bytes.

■ **Send Count Register**: records the length (in bytes) of data that the PLC sends to external devices. Here, the setting is **%MW300**.

■ **Send Buffer Address**: records the head address (in bytes) of the buffer that sends data.

For example, if the send length is **%MW300** = **8**, the send buffer ranges from %MW301 to %MW304. Data is automatically sent when **%MW300** is not **0**. **%MW300** is automatically cleared after data is sent.

Figure 3-41 shows the free protocol configuration for a COM port.

Figure 3-41 Free protocol configuration

Parameter	Definition
Register Type	The options are **%MW** and **SD**.
Receive Count Register	This register displays the number of received bytes when data is received.
Receive Buffer Address	This parameter indicates the head address of the buffer that receives data.
Max Receive Length	This parameter indicates the maximum number of buffered bytes.
Send Count Register	Data is sent when this parameter is not set to **0**. Its value is automatically reset after data is sent.
Send Buffer Address	This parameter indicates the head address of the buffer that sends data.
Max Send Length	This parameter indicates the maximum number of data bytes sent at a time.

The following is an example.
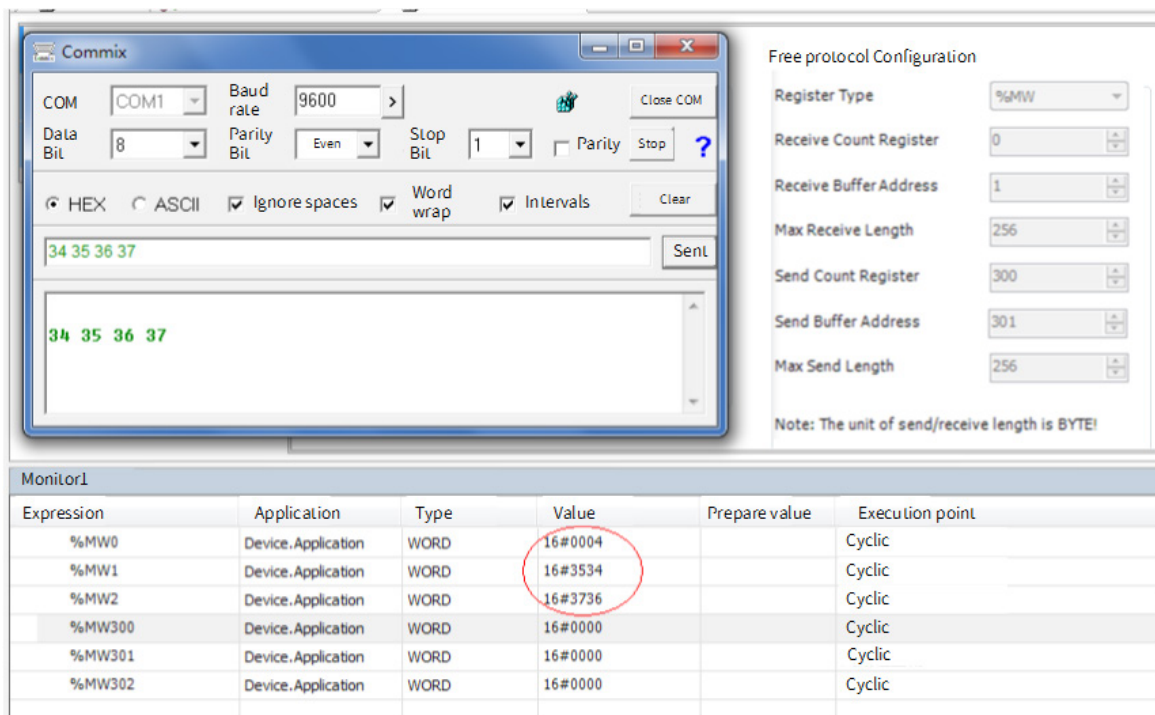
The process based on the preceding settings is as follows:

1) When data is received, **%MW0** displays the number of received data bytes, and the received data is stored in the registers starting from the head address %MW1.

2) Each time after data is received, **%MW0** must be cleared manually so that data is buffered all over again starting from %MW1. If **%MW0** is not cleared, data is buffered in sequence.

3) When the length of received data exceeds the value **256** (bytes) of **Max Receive Length**, **%MW0** starts counting again and the received data is stored from the head address %MW1.

4) Before data is sent, the data is written to the registers starting from the head address %MW301.

5) After the data (bytes) to be sent is written to %MW300, data starting from %MW301 is sent.

6) %MW300 is automatically cleared after data is sent.

7) When the number of bytes written to %MW300 exceeds the value **256** of **Max Send Length**, data is sent based on **Max Send Length**.

3.5.6 Data Send/Receive Tests Through the COM Port Debugging Assistant

1) Use the COM port debugging assistant to send "34 35 36 37".

The length %MW0 of the data received by the PLC is 4 bytes.

The buffer that receives data ranges from %MW1 to %MW2.

%MW0 must be manually cleared before data is received again.

2) Send data from the PLC.

When the length %MW300 of data received by the PLC is greater than 0, for example, 6 bytes, the send buffer ranges from %MW301 to %MW303. **%MW300** is automatically cleared after data is successfully sent.

3.6 Modbus TCP Device Editor

Click a PLC on the **Network Configuration** tab page. A window is displayed, allowing you to enable the master and slaves supported by the CPU, as shown in Figure 3-42. Select the check box before the master or slave you want to enable, and double-click **MODBUS_TCP** in **Network Devices List** on the right to add the slave to the network.
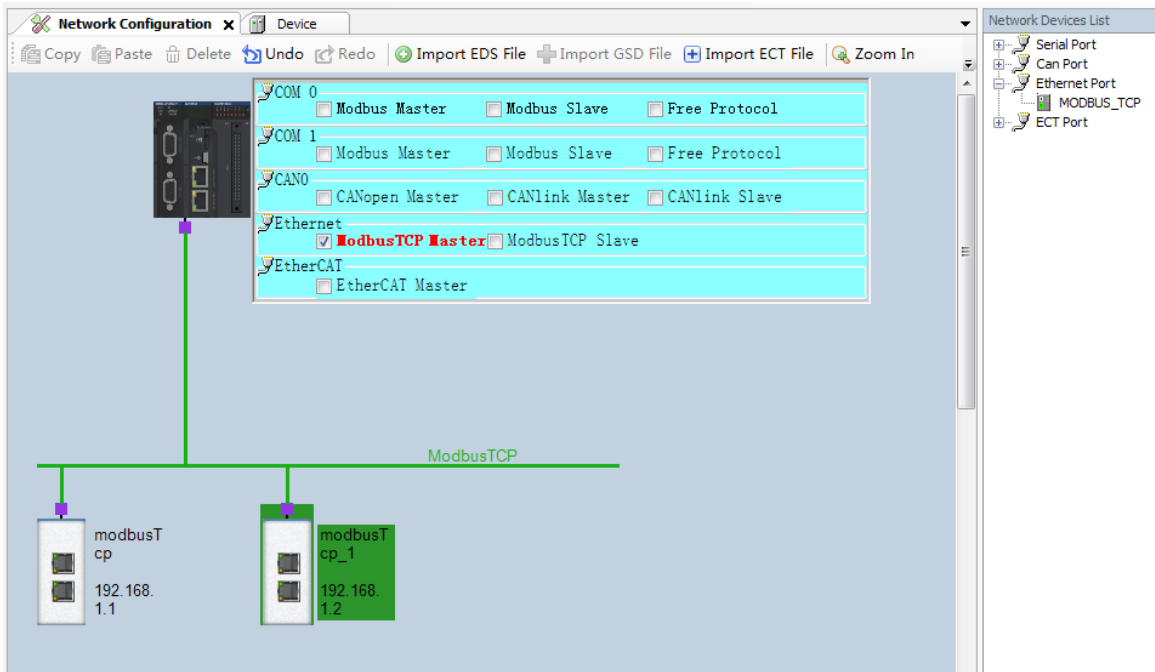
Figure 3-42 Modbus TCP configuration example

The device tree corresponding to Modbus TCP configuration is displayed on the left, as shown in Figure 3-43.
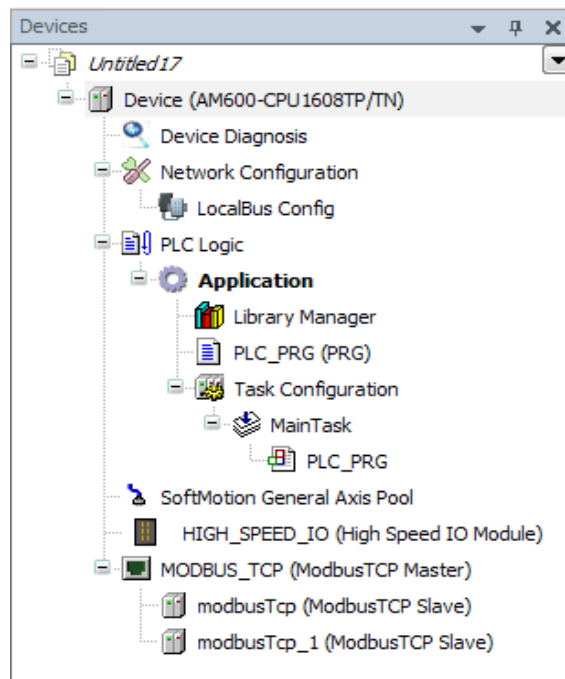
Figure 3-43 Device tree corresponding to Modbus TCP configuration

AM600 supports Modbus TCP communication. You can configure the Modbus TCP master and slaves.

3.6.1 Configuring a Modbus TCP Master

Modbus TCP master configuration
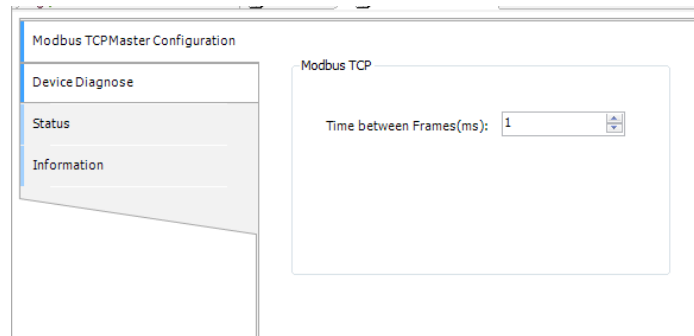
Figure 3-44 Modbus TCP master configuration

Time between Frames indicates the time for the master to wait for the next request data frame after receiving a response data frame. This parameter can be used to adjust the data exchange rate.

3.6.2 Configuring Modbus TCP Master-Slave Connection

Modbus TCP master-slave connection configuration
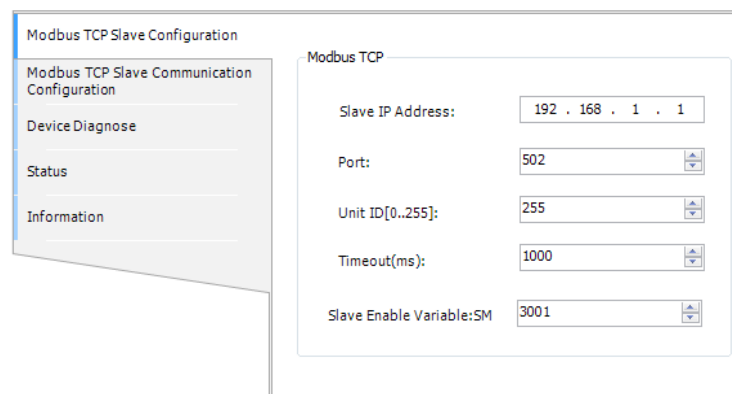
Figure 3-45 Modbus TCP master-slave connection configuration

Parameters:

Parameter	Definition
Slave IP Address	IP address of the Modbus TCP slave used to connect to the master.
Port	TCP port of the Modbus TCP slave used to connect to the master.
Unit ID	Protocol address of the Modbus TCP slave used to connect to the master.
Timeout	Enable the timeout feature and specify the timeout period, in milliseconds.
Slave Enable Variable	The SM element controls the communication initiated to the slave.

Example:

Parameter	Value
Slave IP Address	192.168.10.16
Port	502
Unit ID	05
Timeout	1000
Slave Enable Variable	3001

Configuring Modbus TCP master-slave communication

Figure 3-46 Modbus TCP master-slave communication configuration

In Figure 3-46, each channel represents an independent Modbus TCP request. The **Access Type** column defines the cyclic operation on a read holding register (parameter number: 03) to read the register value with a length of 7 from the register with an offset of 0x0012.

Click **Add...**. The **Modbus Channel Set** dialog box is displayed, allowing you to add a channel to the Modbus TCP slave. Complete settings and click **OK** to create a channel.

Select a channel from the Modbus TCP slave channel list and click **Edit...** The **Modbus Channel Set** dialog box is displayed. Change the values of parameters to modify the channel settings. Click **OK** to update the channel settings.

You can set the following parameters to add or edit a channel:

Figure 3-47 Dialog box for Modbus TCP master-slave communication configuration

Modbus communication parameter settings

Parameter	Definition	
Name	Channel name, in the string format.	
Access Type	Read Coils (Function Code 01). Read Discrete Inputs (Function Code 02). Read Holding Registers (Function Code 03). Read Input Registers (Function Code 04). Write Single Coil (Function Code 05). Write Single Register (Function Code 06). Write Multiple Coils (Function Code 15). Write Multiple Registers (Function Code 16).	
Trigger	**Cyclic**: Requests are triggered periodically.	**Cycle Time**: time for re-execution.
	Trigger by level: Requests are triggered when a change is made during programming.	**Trigger variable (SM)**: SM element that implements trigger.
Repeated	A request is resent for the specified times when no response frame is received from the slave upon a communication error.	
Comment	Brief text description about data.	
Read Register		
Offset	Head address of the registers to be read.	
Length	Number of registers to be read.	
Error Handling	**Keep Last Value**: The last valid value is kept. **0**: All the values are zeroed.	
Write Register		
Offset	Head address of the registers to be written.	
Length	Number of registers to be written.	

The valid range of the **Length** parameter depends on the following parameters:

Parameter Number	Access Type	Register Count
01	Read Coils	1 to 2000
02	Read Discrete Inputs	1 to 2000
03	Read Holding Registers	1 to 125
04	Read Input Registers	1 to 125
05	Write Single Coil	1
06	Write Single Register	1
15	Write Multiple Coils	1 to 1968
16	Write Multiple Registers	1 to 123

Modbus TCP slave internal I/O mapping
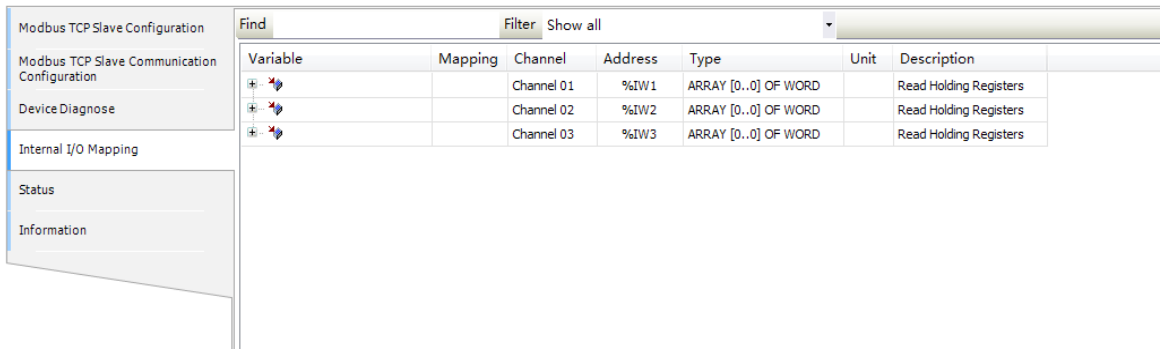
Figure 3-48 Internal I/O mapping of Modbus TCP master-slave connection

Map variables to I/Q addresses by using the input assistant or entering an instance variable path.

3.6.3 Configuring a Modbus TCP Slave

Modbus TCP slave configuration

Figure 3-49 Modbus TCP slave configuration

Parameters:

Parameter	Definition
Slave Port	TCP port of a Modbus TCP slave.
Time Between Frames	Delay for the Modbus TCP slave to return a response frame after receiving a communication frame.

Example:

Parameter	Value
Slave Port	502
Time Between Frames	1

3.6.4 Diagnosing Modbus TCP Devices

Modbus TCP master diagnosis

Figure 3-50 Modbus TCP master diagnosis
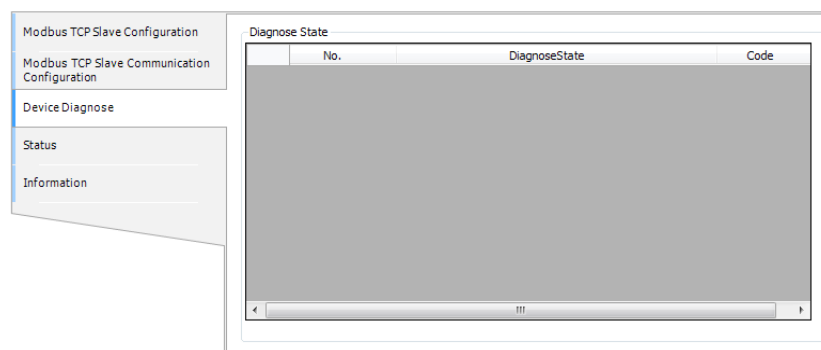
Figure 3-51 Modbus TCP master-slave connection diagnosis

Modbus TCP slave diagnosis

Figure 3-52 Modbus TCP slave diagnosis

3.6.5 Common Errors of Modbus TCP

The following errors are frequently encountered during Modbus TCP master-slave connection:

1. The IP addresses configured for Modbus TCP master-slave connection are incorrect, causing a communication failure.

2. An error response is returned when the Modbus TCP master accesses a slave through an invalid address.

3. The Modbus TCP master receives an error response from the slave when it attempts to write a register of the slave that only supports the read operation.

An error frame consists of a transaction metadata identifier, protocol identifier, length, slave address, command code+0x80, error code, and cyclic redundancy check (CRC).

The preceding error frame is applicable to all command frames.

No.	Definition of Data (Byte)	Number of Bytes	Description
1	Transaction metadata identifier	2	Identifier of Modbus request/response transaction processing
2	Protocol identifier	2	Modbus protocol if the value is **0**
3	Length	2	Number of the following bytes
4	Slave address	1	Value range: 1 to 247
5	Command code+0x80	1	Command error code
6	Error code	1	Value range: 1 to 4

3.6.6 Modbus TCP Communication Frame Format

The 0x01 command code is used to read the Q variable.

The 0x31 command code is used to read the SM variable.

A request frame consists of a transaction metadata identifier, protocol identifier, length, slave address, 0x01, head address of coils, and number of coils.

No.	Definition of Data (Byte)	Number of Bytes	Description
1	Transaction metadata identifier	2	Identifier of Modbus request/response transaction processing.
2	Protocol identifier	2	Modbus protocol if the value is **0**.
3	Length	2	Number of the following bytes.
4	Slave address	1	Value range: 1 to 247.
5	0x01/0x31 (command code)	1	The command is to read coils.
6	Head address of coils	2	Big-endian. See "Coil addressing".
7	Number of coils	2	Big-endian (N).

A response frame consists of a transaction metadata identifier, protocol identifier, length, slave address, 0x01, number of bytes, and coil status.

No.	Definition of Data (Byte)	Number of Bytes	Description
1	Transaction metadata identifier	2	Identifier of Modbus request/response transaction processing.
2	Protocol identifier	2	Modbus protocol if the value is **0**.
3	Length	2	Number of the following bytes.
4	Slave address	1	Value range: 1 to 247.
5	0x01/0x31 (command code)	1	The command is to read coils.
6	Number of bytes	1	The value is (N + 7)/8.
7	Coil status	(N + 7)/8	Eight coils are indicated by one byte. If the last byte has less than eight bits, enter 0 for undefined bits. The first eight coils are indicated by the first byte, and the coil with the smallest address is indicated by the least significant bit.

The 0x03 command code is used to read the M variable.

The 0x33 command code is used to read the SD variable.

A request frame consists of a transaction metadata identifier, protocol identifier, length, slave address, 0x03, head address of registers, and number of registers.

No.	Definition of Data (Byte)	Number of Bytes	Description
1	Transaction metadata identifier	2	Identifier of Modbus request/response transaction processing.
2	Protocol identifier	2	Modbus protocol if the value is **0**.
3	Length	2	Number of the following bytes.
4	Slave address	1	Value range: 1 to 247.
5	0x03/0x33 (command code)	1	The command is to read registers.
6	Head address of registers	2	Big-endian. See "Register addressing".
7	Number of registers	2	Big-endian (N).

A response frame consists of a transaction metadata identifier, protocol identifier, length, slave address, 0x03, number of bytes, and register value.

No.	Definition of Data (Byte)	Number of Bytes	Description
1	Transaction metadata identifier	2	Identifier of Modbus request/response transaction processing.
2	Protocol identifier	2	Modbus protocol if the value is **0**.
3	Length	2	Number of the following bytes.
4	Slave address	1	Value range: 1 to 247.
5	0x03/0x33 (command code)	1	The command is to read registers.
6	Number of bytes	1	The value is N x 2
7	Register value	N x 2	One register value is indicated by two bytes. It is big-endian. The register with a smaller address is indicated by the first byte.

The 0x05 command code is used to read the Q variable.

The 0x35 command code is used to read the SM variable.

A request frame consists of a transaction metadata identifier, protocol identifier, length, slave address, 0x05, coil address, and coil status.

No.	Definition of Data (Byte)	Number of Bytes	Description
1	Transaction metadata identifier	2	Identifier of Modbus request/response transaction processing.
2	Protocol identifier	2	Modbus protocol if the value is **0**.
3	Length	2	Number of the following bytes.
4	Slave address	1	Value range: 1 to 247.
5	0x05/0x35 (command code)	1	The command is to write a single coil.
6	Address of the coil	2	Big-endian. See "Coil addressing".
7	Coil status	2	Big-endian. A non-zero value is valid.

A response frame consists of a transaction metadata identifier, protocol identifier, length, slave address, 0x05, coil address, and coil status.

No.	Definition of Data (Byte)	Number of Bytes	Description
1	Transaction metadata identifier	2	Identifier of Modbus request/response transaction processing.
2	Protocol identifier	2	Modbus protocol if the value is **0**.

No.	Definition of Data (Byte)	Number of Bytes	Description
3	Length	2	Number of the following bytes.
4	Slave address	1	Value range: 1 to 247.
5	0x05/0x35 (command code)	1	The command is to write a single coil.
6	Address of the coil	2	Big-endian. See "Coil addressing".
7	Coil status	2	Big-endian. A non-zero value is valid.

The 0x06 command code is used to read the M variable.

The 0x36 command code is used to read the SD variable.

A request frame consists of a transaction metadata identifier, protocol identifier, length, slave address, 0x06, register address, and register value.

No.	Definition of Data (Byte)	Number of Bytes	Description
1	Transaction metadata identifier	2	Identifier of Modbus request/response transaction processing.
2	Protocol identifier	2	Modbus protocol if the value is **0**.
3	Length	2	Number of the following bytes.
4	Slave address	1	Value range: 1 to 247.
5	0x06/0x36 (command code)	1	The command is to write a single register.
6	Address of the register	2	Big-endian. See "Register addressing".
7	Register value	2	Big-endian. A non-zero value is valid.

A response frame consists of a transaction metadata identifier, protocol identifier, length, slave address, 0x06, register address, and register value.

No.	Definition of Data (Byte)	Number of Bytes	Description
1	Transaction metadata identifier	2	Identifier of Modbus request/response transaction processing.
2	Protocol identifier	2	Modbus protocol if the value is **0**.
3	Length	2	Number of the following bytes.
4	Slave address	1	Value range: 1 to 247.
5	0x06/0x36 (command code)	1	The command is to write a single register.
6	Address of the register	2	Big-endian. See "Register addressing".
7	Register value	2	Big-endian. A non-zero value is valid.

The 0x0f command code is used to write multiple consecutive Q variables.

The 0x3f command code is used to write multiple consecutive Q variables.

A request frame consists of a transaction metadata identifier, protocol identifier, length, slave address, 0x0f, head address of coils, number of coils, number of bytes, and coil status.

No.	Definition of Data (Byte)	Number of Bytes	Description
1	Transaction metadata identifier	2	Identifier of Modbus request/response transaction processing.
2	Protocol identifier	2	Modbus protocol if the value is **0**.

No.	Definition of Data (Byte)	Number of Bytes	Description
3	Length	2	Number of the following bytes.
4	Slave address	1	Value range: 1 to 247.
5	0x0f/0x3f (command code)	1	The command is to write multiple coils.
6	Head address of coils	2	Big-endian. See "Coil addressing".
7	Number of coils	2	Big-endian. The maximum of N is 1968.
8	Number of bytes	1	The value is (N + 7)/8.
9	Coil status	(N + 7)/8	Eight coils are indicated by one byte. If the last byte has less than eight bits, enter 0 for undefined bits. The first eight coils are indicated by the first byte, and the coil with the smallest address is indicated by the least significant bit.

A response frame consists of a transaction metadata identifier, protocol identifier, length, slave address, 0x05, head address of coils, and number of coils.

No.	Definition of Data (Byte)	Number of Bytes	Description
1	Transaction metadata identifier	2	Identifier of Modbus request/response transaction processing.
2	Protocol identifier	2	Modbus protocol if the value is **0**.
3	Length	2	Number of the following bytes.
4	Slave address	1	Value range: 1 to 247.
5	0x0f/0x3f (command code)	1	The command is to write multiple coils.
6	Head address of coils	2	Big-endian. See "Coil addressing".
7	Number of coils	2	Big-endian.

The 0x10 command code is used to write multiple consecutive M variables.

The 0x40 command code is used to write multiple consecutive SD variables.

A request frame consists of a transaction metadata identifier, protocol identifier, length, slave address, 0x10, head address of registers, number of registers, number of bytes, and register value.

No.	Definition of Data (Byte)	Number of Bytes	Description
1	Transaction metadata identifier	2	Identifier of Modbus request/response transaction processing.
2	Protocol identifier	2	Modbus protocol if the value is **0**.
3	Length	2	Number of the following bytes.
4	Slave address	1	Value range: 1 to 247.
5	0x10/0x40 (command code)	1	The command is to write multiple registers.
6	Head address of registers	2	Big-endian. See "Register addressing".
7	Number of registers	2	Big-endian. The maximum of N is 120.
8	Number of bytes	1	The value is N x 2
9	Register value	N x 2 (N x 4)	

A response frame consists of a transaction metadata identifier, protocol identifier, length, slave address, 0x05, head address of coils, and number of coils.

No.	Definition of Data (Byte)	Number of Bytes	Description
1	Transaction metadata identifier	2	Identifier of Modbus request/response transaction processing.
2	Protocol identifier	2	Modbus protocol if the value is **0**.
3	Length	2	Number of the following bytes.
4	Slave address	1	Value range: 1 to 247.
5	0x10/0x40 (command code)	1	The command is to write multiple registers.
6	Head address of registers	2	Big-endian. See "Register addressing".
7	Number of registers	2	Big-endian. The maximum of N is 120.

3.7 CANopen Network

CANopen Bus Overview

CANopen is the industrial communication protocol family for distributed automatic control devices based on the CAN bus. CANopen is promoted by CAN in Automation (CiA) and was standardized at the end of 2002, with a standard number of CENELEC EN 50325-4. CANopen defines application-layer protocols, communication-layer protocols, and multiple application protocols. Figure 3-53 shows the overall architecture of CANopen.
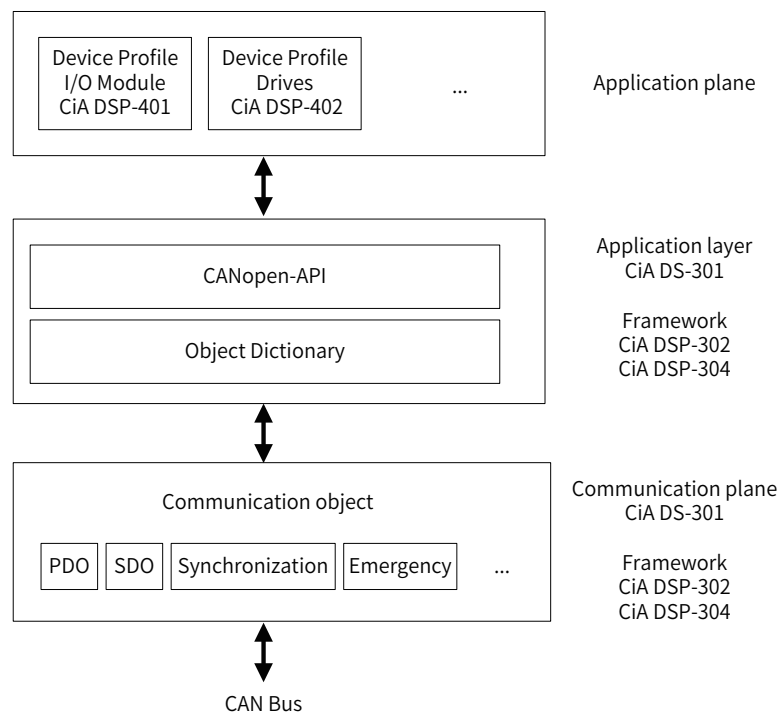
Figure 3-53   Overall architecture of CANopen

The application layer provides applications with acknowledged and unacknowledged services and defines communication objects. Those services can be used to request data from a server.

Communication objects are used during data exchange. They are used to exchange process data and service data, manage processes or synchronize the system clock, manage error statuses, and control and monitor node statuses. A communication object is defined by a structure, transmission type, and CAN identifier. The typical parameters of communication objects include the CAN identifier for data transmission, message transmission type, and disable time or event time. The parameters are defined in communication protocols.

For each CANopen device, the main data structure is an object dictionary, which is the primary data exchange medium for the communication between the application layer and CAN bus. Object dictionary portals can be accessed from the application layer and CAN bus through special messages. Those portals can be considered as a variable or a programmer-defined area.

Each portal has an index or a subindex. An object dictionary portal can be accurately located by using the index structure. The CANopen protocol stack provides standard APIs to define object dictionary portals, including their read and write attributes. Object dictionaries can be accessed by using communication objects through the CAN bus.

In object dictionaries, the attributes of each portal must be defined, including the data type, access permission, data transmission for process data objects (PDOs), and variable range.

■ PDOs are used to transmit the real-time data of processes.

■ Service data objects (SDOs) are used to access the object dictionary of a device, set parameters, or transmit non-real-time data.

■ SYNC messages do not contain data. They are sent by the producer to the CAN network periodically to trigger transmission of PDO data on nodes.

■ Emergency messages are triggered by the internal critical errors of devices. They consist of a fault code, fault register, and manufacturer specific error.

CiA DS-301 defines the application layer and communication protocols. CiA DSP-302 defines the framework of programmable CANopen devices. CiA DSP-304 defines the framework of secure redundant data transmission, whereas the data description of specific devices is defined in the application protocol consisting of respective device protocols. For example, CiA DSP-401 defines the data format of the I/O module, and CiA DSP-402 defines the data format for drive control.

3.7.1 General Process of Using CANopen

The general process of using CANopen is as follows:

1) Design the CANopen hardware network structure.

2) On the **Network Configuration** tab page, activate the CANopen bus. After the CANopen bus is activated, the CANopen master is automatically added, and the CANopen bus task named CANopen is also added. By default, the CANopen bus uses the task to refresh I/O.

3) On the **Network Configuration** tab page, add CANopen slaves and modules based on the hardware structure. Before adding a third-party slave, import an EDS file to import the third-party slave on the **Network Configuration** tab page.
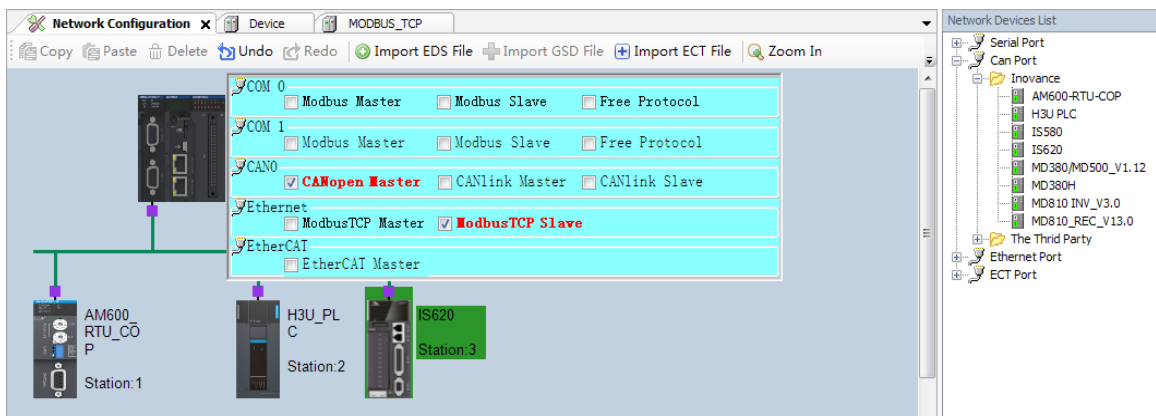
4) To add an AM600 slave, you need to add an I/O module on the **Hardware Configuration** tab page. Double-click the module in **In\Output Module List** on the right, as shown in the following figure. The CANopen slave is a CANopen remote device.
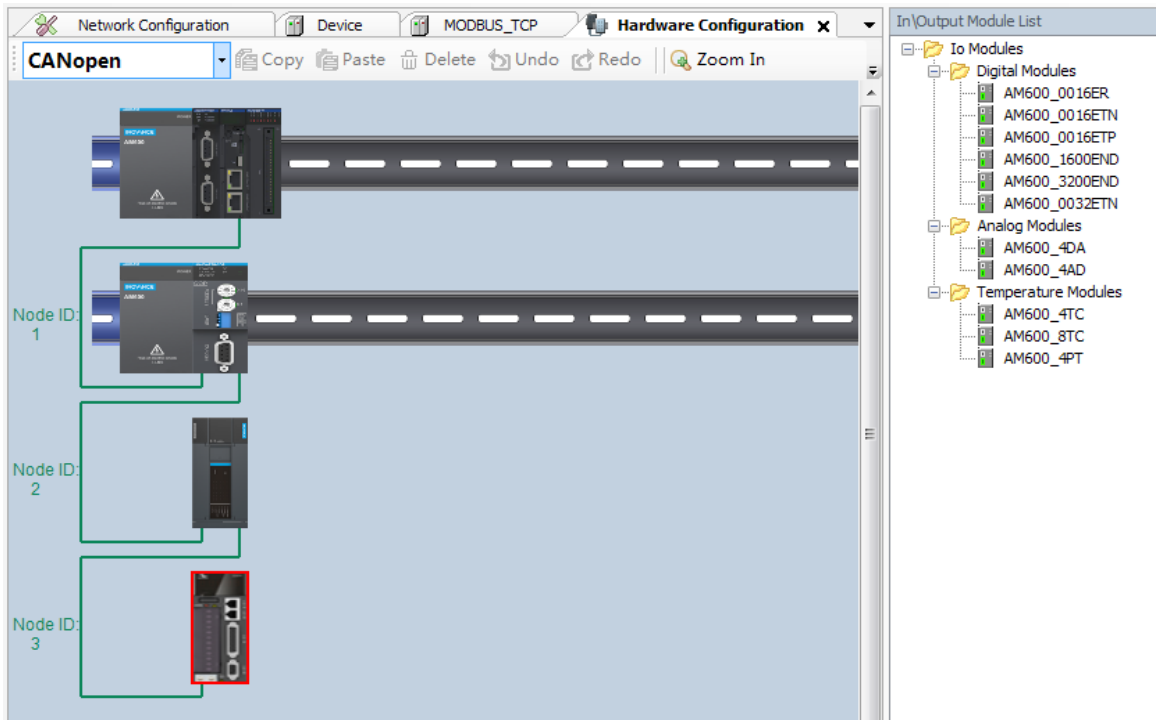
5) Set the master parameters, slave parameters, and module parameters properly. In normal cases, the slave node ID is automatically generated, PDOs and mapping are automatically generated based on the EDS file, and some special settings need to be modified manually.

When setting the parameters of the master and slave, ensure that the master baud rate and slave node ID match with the slave baud rate and slave node ID DIP switch, respectively.
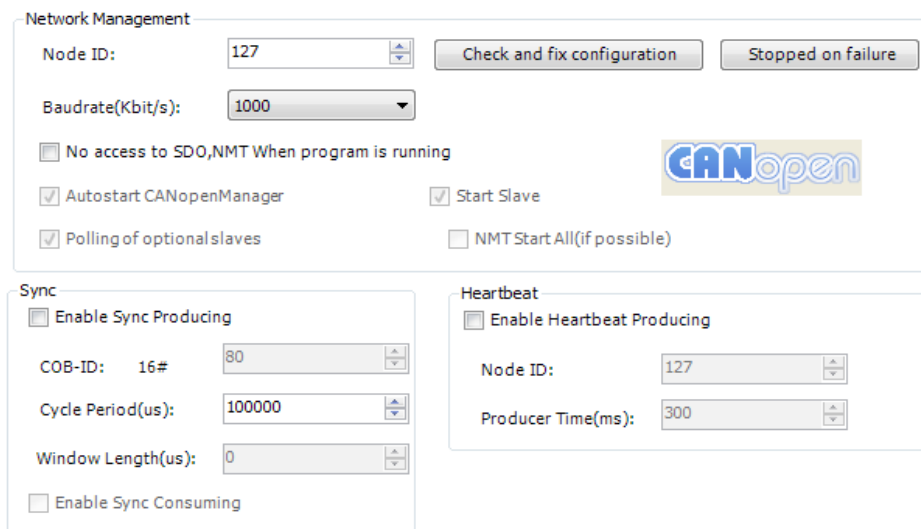
Figure 3-54 Master parameter settings

Figure 3-55 Slave parameter settings

Figure 3-56   Module parameter settings

The software provides soft elements to obtain the CANopen slave status and the CiA-DSP405 library for slave management and operation.

3.7.2 Configuring a CANopen Master

1 Master configuration

Figure 3-57 CANopen master configuration page

Network management

■ **Node ID**: unique identifier of the master in the CANopen network. The default value is **127**, and the value range is **1** to **127**, in the decimal format.

■ **Check and fix configuration**: See "Check and fix configuration."

■ **Stopped on failure**: See "Stopped on failure."

■ **Baudrate**: baud rate of transmission along the bus. The unit is bit/s. The optional values are 10000, 20000, 50000, 100000, 125000, 250000, 500000, 800000, and 1000000. The default value is 1000000.

> **NOTE**
> If the CPU module is at the head end or tail end of the network, turn the build-out resistor of the CANopen port to ON. Set a proper baud rate because the communication distance is related to the baud rate.

■ **No access to SDO, NMT When program is running**: If it is selected, the slave cannot be accessed through SDO and NMT in the user program or on the slave debugging page when the program is running.

■ **Network load**: real-time load of the CANopen network when the bus is running. The network load is displayed only after you log in to the PLC.

Sync

■ **Enable Sync Producing**: If it is selected, the master sends synchronization information. It is deselected by default. This function can be enabled on only one station of the CANopen bus system. The PDO indicating the synchronization type sends information based on the preset type after synchronization information is sent.

■ **COB-ID**: indicates the communication object identifier, which is also the synchronization message ID. It is invariably set to **16#80** and cannot be changed. This COB-ID is also used if **Enable Sync Producing** is selected on the slave.

■ **Cycle Period (μs)**: indicates the interval at which synchronization information is sent. The value ranges from 2000 to 4,294,967,000, in microseconds. It is an integral multiple of the bus task time.

■ **Window Length(μs)**: used for PDO synchronization. The unit is microseconds. It is invariably set to **0** and cannot be changed.

Heartbeat

Heartbeat is a node guarding mechanism. Different from node daemon, heartbeat can be triggered by the master or slave. In normal cases, the master sends heartbeat information to the slave, which is configured with the master node ID for consumption so that the slave can monitor the master.

- **Enable Heartbeat Producing**: If it is selected, the master sends heartbeat information. It is deselected by default.

- **Node ID**: unique identifier of the sent heartbeat information. By default, it is set to the master node ID. The value ranges from 1 to 127.

- **Producer Time (ms)**: interval at which heartbeat information is sent. The value ranges from 2 to 32,767, in milliseconds. It is an integral multiple of the bus task time.

- Window Length (μs): used for PDO synchronization. The unit is microseconds. It is invariably set to 0 and cannot be changed.

2 Check and fix configuration

When multiple slaves are added to the CANopen system, the master or slave node IDs may be repeated or the COB-IDs may conflict with each other because the EDS files of different slaves may contain a configured COB-ID by default or the slave node ID is modified. Click **Check and fix configuration** to solve the problem of repeated node IDs or conflicting COB-IDs. You can enter the **Check and fix configuration** page from the master configuration page.
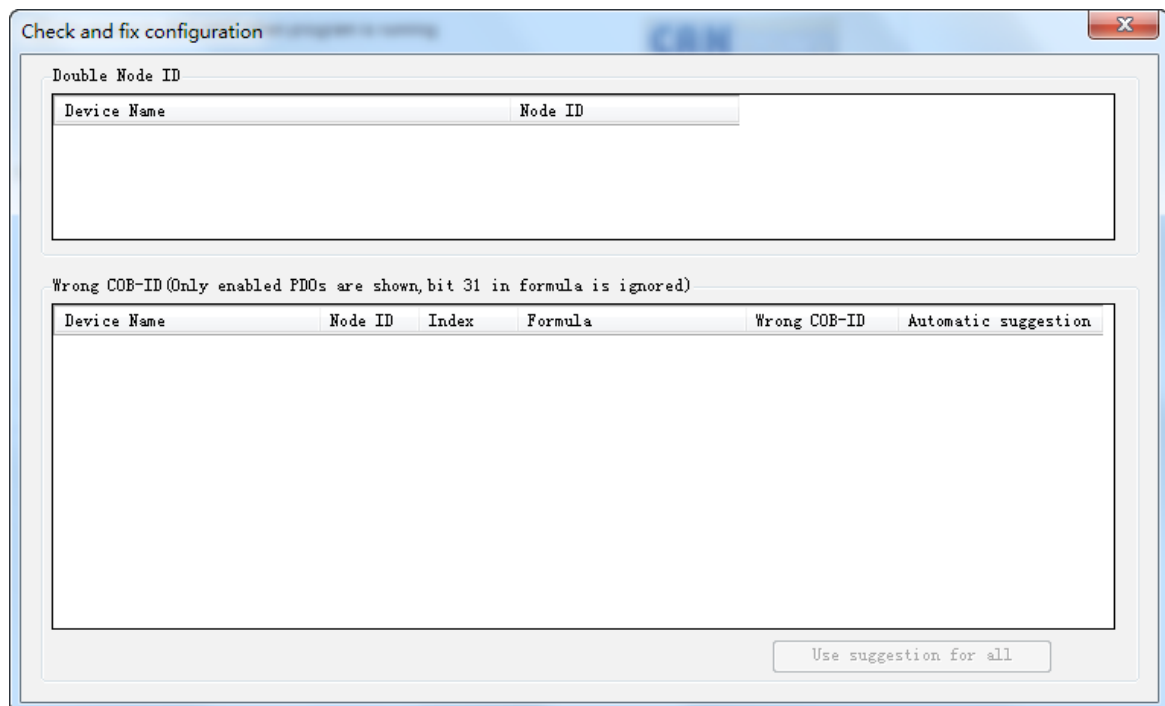
Figure 3-58   **Check and fix configuration** page

Double Node ID

The **Double Node ID** area lists the slaves with the same node ID. You can edit the **Node ID** column to re-allocate node IDs. Then, repeated node IDs are automatically canceled.

Wrong COB-ID

The **Wrong COB-ID** area lists the slaves with conflicting and invalid COB-IDs. You can modify COB-IDs in one of the following three ways:

■ Edit the **Wrong COB-ID** column to manually modify the COB-ID corresponding to the current slave index.

■ Click the button in the **Automatic suggestion** column and modify the COB-ID corresponding to the current slave index based on the displayed value.

■ Click **Use suggestion for all** and modify all the incorrect COB-IDs based on the displayed values.

After modification, slaves with correct COB-IDs disappear from the page.

3 Stop setting on failure

The Stop Setting on Failure function determines whether to stop slave operation when a slave or module is faulty or the configuration is inconsistent. This function is only applicable to AM600 CANopen slaves.
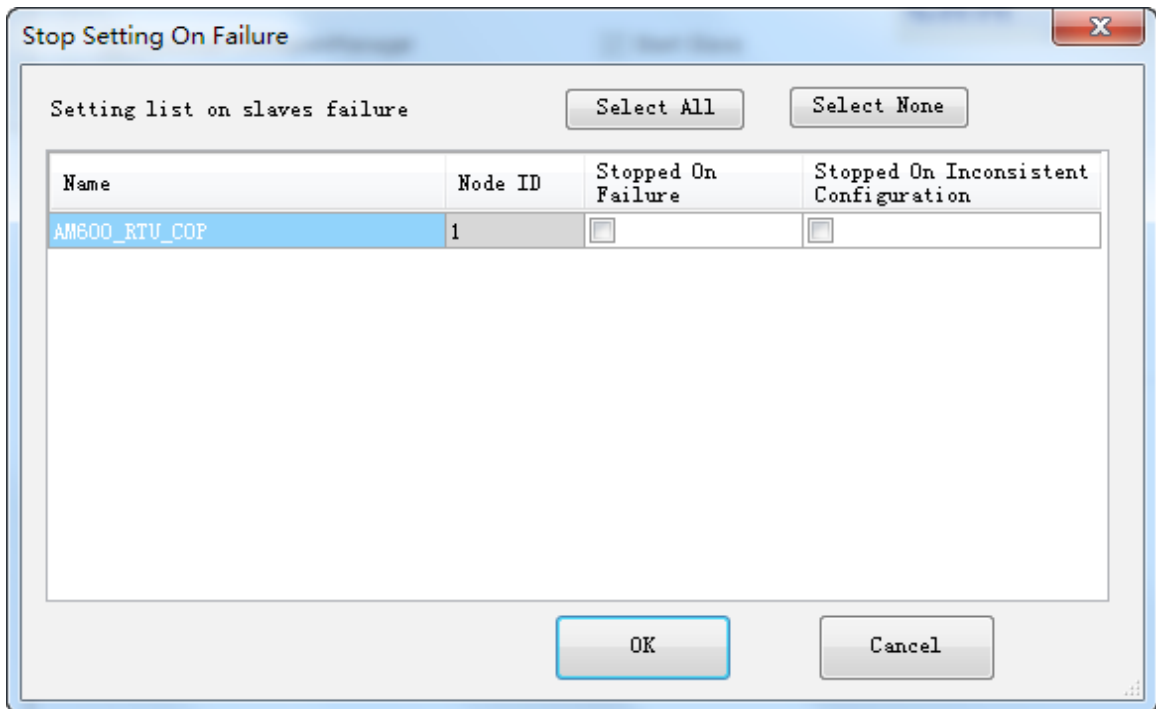
Figure 3-59   **Stop Setting on Failure** page

■ **Setting list on slaves failure**: You can view and set whether to stop operation upon slave failure or inconsistent configuration.

■ In the **Stopped On Failure** column, you can set whether to stop slave operation when the specified slave or module is faulty. If the check box under **Stopped On Failure** is selected, the slave stops running when it is faulty or when the I/O module with the diagnosis and report function enabled is faulty.

■ In the **Stopped on Inconsistent Configuration** column, you can set whether to stop slave operation when the I/O module of the slave has inconsistent configuration. If the check box is selected, the slave stops running when the I/O type does not match or the number of modules is more or less than the actual quantity.

■ Click **Select All** or **Select None** to activate or deactivate all the slave settings in **Setting list on slaves failure**.

■ Click **OK** or **Cancel** to save or cancel the settings on the **Stop Setting on Failure** page.

4 CANopen master I/O mapping

For the general description of I/O mapping and instructions on this dialog box, see "I/O mapping."

5 State

The state configuration editor for the CANopen bus devices or modules displays state information (such as Running and Stopped) and the status of the internal bus system.

6 Information

The following basic information about the currently available device is displayed: name, vendor, type, version, module number, and description.

3.7.3 Configuring a CANopen Slave

The main items of CANopen slave configuration include the basic parameters, PDO settings, SDO settings, and debugging.

1 Slave parameter setting

Figure 3-60 Slave parameter setting

General

- **Node ID**: unique identifier of a slave in the CANopen network. The value ranges from 1 to 127, in the decimal format. The node ID must be consistent with the slave identifier (such as the DIP switch).

- **SDO Channels**: not supported for the moment.

- **Enable Expert Settings**: If it is selected, you can set expert parameters, such as slave node protection, heartbeat generation, emergency packet, check restart, PDO mapping operation, system SDO display, and SDO abnormal jump.

- **Optional Device**: not supported for the moment.

- **Create all SDOs**: Select it to create writable SDOs in the object dictionary. For example, the object access attributes are RW, WO, RWR, and RWW. The created SDOs are displayed on the **Service Data Object** page.

- **NO Initialisation**: not supported for the moment.

- **Enable Sync Producing**: If it is selected, the slave sends synchronization information. It is deselected by default. This function can be enabled on only one station of the CANopen bus system. Synchronous sending adopts the synchronization parameter settings of the master.

- **Factory Settings**: If it is selected, the slave parameter settings are restored before you download configuration or configure the slave. The type of parameter restoration depends on the option selected from the restoration type list. The options are as follows:

1) **sub:001**: restores all the parameters.

2) **sub:002**: restores communication-related parameters (manufacturer-specified communication parameters indexed from 1000h to 1FFFh).

3) **sub:003**: restores application-related parameters (manufacturer-specified application parameters indexed from 6000h to 9FFFh).

4) **sub:004** to **sub:127**: restores manufacturer-defined parameters.

5) **sub:128** to **sub254**: reserved.

The options in the parameter restoration type list are based on the current object dictionary (EDS file) and come from the parsing results of EDS file index 1011. The subindexes have one-to-one correspondence with the preceding options.

Error Control

In the **Error Control** area, you can configure to monitor the node online status. The configuration items include node guarding and heartbeat.

The node guarding function enables the master to monitor the online status of the slave. The master sends slave daemon information periodically, and the slave is supposed to return a response to the master. If the slave fails to respond within the node daemon time (equal to the guard time multiplied by the life time factor), the master considers that the slave is lost.

Heartbeats can be produced by the master or slave. The producer broadcasts heartbeat packets to the CAN bus, and the heartbeat consumer consumes the heartbeats. If a node is configured with heartbeat consumption and no heartbeats corresponding to the node ID are detected within the configured time, the node is considered lost. Generally, the slave consumes the heartbeats of the master to monitor the online status of the master.

- **Enable Nodeguarding**: Select this check box to enable the node guarding function. Node guarding and heartbeat are mutually exclusive. The master sends a node guard matrix periodically within the guard time. If the slave fails to return a response containing a specific COB-ID (communication object identifier) within the node daemon time (which is equal to the guard time multiplied by the life time factor), the slave is considered offline.

- **Guard Time**: interval at which the master sends node guard frames periodically. The value is an integral multiple of the bus task time and ranges from 10 to 65,535, in ms.

- **Life Time Factor**: used with **Guard Time**. If the slave does not return a response within the node daemon time (equal to the guard time multiplied by the life time factor), the master considers that the slave is lost. The value ranges from 1 to 255.

- **Enable Heartbeat**: Select this check box to enable the heartbeat function on the slave so that the slave sends heartbeat frames periodically at an interval indicated by **Producer Time**. Heartbeat and node guarding are mutually exclusive.

- **Producer Time**: interval at which the slave sends heartbeat frames. The value is an integral multiple of the bus task time and ranges from 10 to 32,767, in ms.

- **Change Heartbeat Consumer Properties**: Click this button to configure a heartbeat producer for the slave. You can configure heartbeat consumption for a slave so that the slave monitors the online status of the slave that produces heartbeats. Generally, the slave consumes the heartbeats of the master.
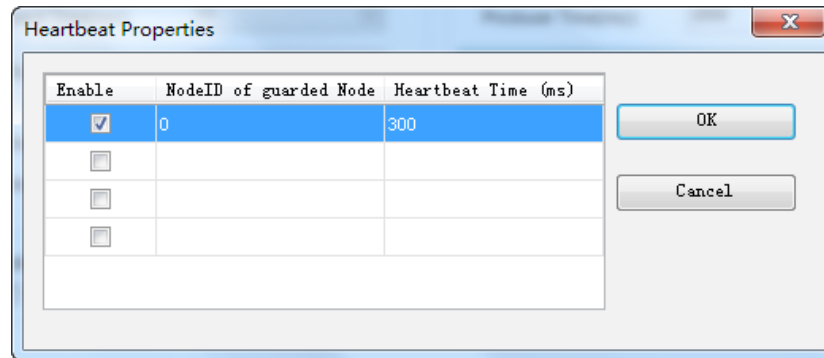
Figure 3-61   **Heartbeat Properties** dialog box

You can configure a heartbeat producer after selecting the **Enable** check box.

By default, **NodeID of guarded Node** is set to the ID of the heartbeating node of the master. If the heartbeat function is not enabled on the master, this parameter is set to 0. The value ranges from 1 to 127. By default, **Heartbeat Time** is equal to the master heartbeating time multiplied by 1.5. The value ranges from 1 to 65535.

Emergency

- **Enable Emergency**: Select this check box so that the slave sends emergency messages through the emergency COB-ID. The emergency messages can be obtained through the functions provided by the CiA405 library (RECV_EMCY_DEF, RECV_EMCY) function library.

- **Emergency COB-ID**: COB-ID for the slave to send emergency messages. The default value is **$NODEID+16#80**, where **NODEID** is the node ID of the slave. The COB-ID is in the format $NODEID+16#+hexadecimal number, 16#+hexadecimal number, or decimal number. (Example)

Checks At Startup

- **Check Vendor ID**: Select this check box to enable vendor ID checking. The slave checks whether the vendor ID (index: 1018; subindex: 01) in the object dictionary matches with the vendor ID of the slave. The slave may not run properly if they do not match.

- **Check Product Number**: Select this check box to enable product number checking. The slave checks whether the product number (index: 1018; subindex: 02) in the object dictionary matches with the product number of the slave. The slave may not run properly if they do not match.

- **Check Revision**: Select this check box to enable version checking. The slave checks whether the version (index: 1018; subindex: 03) in the object dictionary matches with the version of the slave. The slave may not run properly if they do not match.

2 Receive PDO

PDO is used by real-time data transmission between master and slave. Receive PDO is the real-time data that the master sends to the slave.

A PDO includes communication parameters and mapping parameters. The communication parameters include the unique communication identifier COB-ID, transmission type, and transmission control. The PDO mapping parameters indicate the indexes and subindexes in the object dictionary which the PDO transmitted data come from.

Receive PDO comes from the objects starting from index 1400 to index 1600 in the object dictionary. The default communication parameter values of each PDO come from the corresponding subindexes. The objects mapped to the receive PDO come from the writable objects in the object dictionary, such as the RW, RWW, and WO access permissions.

NOTE	1) In non-expert mode, you can only change the values of the PDO communication parameters, but cannot add or delete PDOs and PDO mappings.
	2) AM600 slaves can only change the values of the PDO communication parameters, but cannot add or delete PDOs and PDO mappings. PDO mappings increase as AM600 I/Os are added.

The receive PDO is mapped to the AM600 output module. Each module corresponds to an invariable index, as shown in the following table.

Module Type	Index (Hexadecimal)	Subindex (Hexadecimal)	Data Type
DO16	6300	01-10	Unsigned short int
DA4	6411	01-10	Short int

Figure 3-62 **Receive PDO** page

■ Click **Add PDO** to add a PDO. The new PDO appears at the end of the PDO list. The maximum number of receive PDOs of the slave is determined by the number of indexes from 1400 to 1600 in the object dictionary. No more PDOs can be added when the maximum number is exceeded. The added PDO name and index are automatically obtained from the object dictionary in the usage sequence, and they cannot be modified.

After you click **Add PDO**, a dialog box is displayed, where you can set PDO attributes. For details, see "PDO attributes."

■ Click **Add Mapping** to add a PDO mapping to the selected PDO. The new PDO mapping appears after the current PDO. A PDO mapping contains a maximum of 64 bits. If this limit is exceeded, the PDO mapping cannot be added. PDO mappings come from the object dictionary. Receive PDOs are mapped to the writable objects in the object dictionary, such as the RW, RWW, and WO access permissions. For non-AM600 slaves, when you click **Add Mapping** to add a receive PDO mapping, the **Select Item From Object Dictionary** dialog box is displayed. For details, see "Adding an object."

■ Click **Edit** to change the value of the selected PDO communication or mapping parameter. If a PDO is selected, you can change the values of PDO communication parameters. If a PDO mapping is selected, you can modify the PDO mapping. For AM600 slaves, you can only change the values of communication parameters.

■ Click **Delete** to delete the selected PDO or PDO mapping. If a PDO is selected, this PDO is deleted. If a PDO mapping is selected, you can modify the PDO mapping. AM600 slaves do not support the delete operation.

3 Send PDO

PDO is used by real-time data transmission between master and slave. Send PDO is the real-time data that the slave sends to the master.

Send PDO comes from the objects starting from index 1800 to index 1A00 in the object dictionary. The default communication parameter values of each PDO come from the corresponding subindexes. The objects mapped to the send PDO come from the readable objects in the object dictionary, such as the RW, RWR, RO, and CONST access permissions.

NOTE

1) In non-expert mode, you can only change the values of the PDO communication parameters, but cannot add or delete PDOs and PDO mappings.

2) AM600 slaves can only change the values of the PDO communication parameters, but cannot add or delete PDOs and PDO mappings. PDO mappings increase as AM600 I/Os are added.

The send PDO is mapped to the AM600 input module. Each module corresponds to an invariable index, as shown in the following table.

Module Type	Index (Hexadecimal)	Subindex (Hexadecimal)	Data Type
DI16	6100	01-10	unsigned short int
AD4	6401	01-10	short int

Figure 3-63 Send PDO page

■ Click Add PDO to add a PDO. The new PDO appears at the end of the PDO list. The maximum number of send PDOs of the slave is determined by the number of indexes from 1800 to 1A00 in the object dictionary. No more PDOs can be added when the maximum number is exceeded. The added PDO name and index are automatically obtained from the object dictionary in the usage sequence, and they cannot be modified.

After you click **Add PDO**, a dialog box is displayed, where you can set PDO attributes. For details, see "PDO attributes."

■ Click Add Mapping to add a PDO mapping to the selected PDO. The new PDO mapping appears after the current PDO. A PDO mapping contains a maximum of 64 bits. If this limit is exceeded, the PDO mapping cannot be added. PDO mappings come from the object dictionary. Send PDOs are mapped to the readable objects in the object dictionary, such as the RW, RWR, RO, and CONST access permissions.

When you click **Add Mapping** to add a PDO mapping, the **Select Item From Object Dictionary**

dialog box is displayed. For details, see "Adding an object."

■ Click Edit to change the value of the selected PDO communication or mapping parameter. If a PDO is selected, you can change the values of PDO communication parameters. If a PDO mapping is selected, you can modify the PDO mapping. For AM600 slaves, you can only change the values of communication parameters.

■ Click Delete to delete the selected PDO or PDO mapping. If a PDO is selected, this PDO is deleted. If a PDO mapping is selected, you can modify the PDO mapping. AM600 slaves do not support the delete operation.

4 SDO

SDOs are used to transmit data during slave initialization and operation. The settings on the **Service Data Object** page are written to the slave during slave initialization.

On the **Service Data Object** interface, you can configure the selected SDO, modify the SDO transmission sequence, and define an error handling method to be applied during SDO transmission.
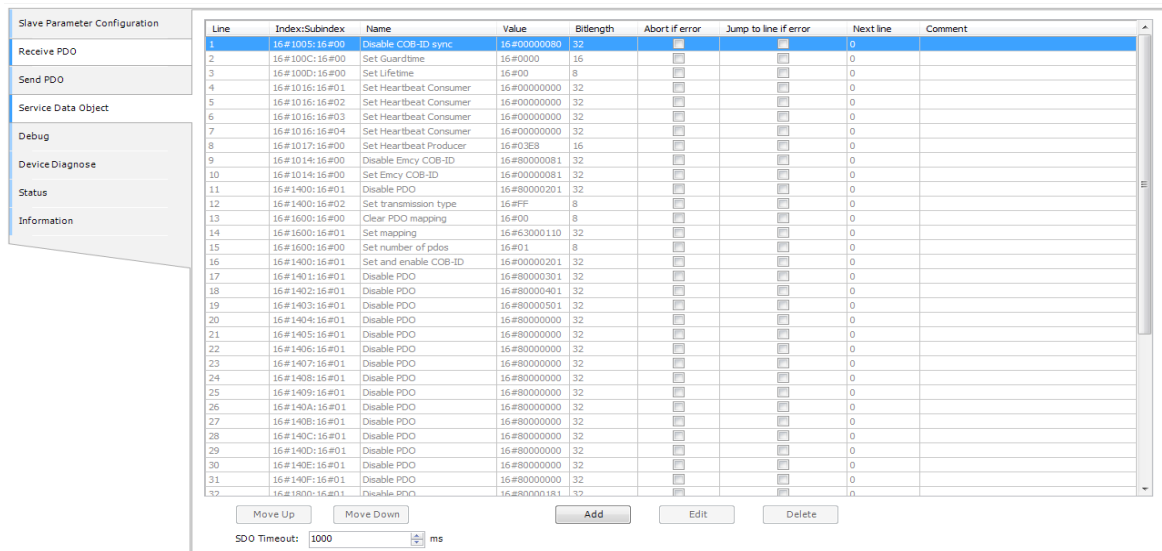
Figure 3-64   **Service Data Object** page

The SDO list displays all the SDOs written to the slave during slave initialization. The SDOs in gray are automatically added and displayed on top of the list. They are configured preferentially and automatically generated based on the parameter settings on the slave configuration page, such as heartbeat, node daemon, emergency information, PDO, and PDO mapping. You can click **Add** to add user-defined SDOs. User-defined SDOs can be modified and their positions in the list can be changed.

Double-click a value in the **Value** column to change the value of the corresponding SDO.

You can define an error handling method to be applied during SDO configuration. If the **Abort if error** check box is selected, SDO configuration is aborted in the case of an error, and the following SDOs are not configured. If the **Jump to line if error** check box is selected, the system jumps to the specified line and you can configure the following SDOs. If **Abort if error** and **Jump to line if error** are not selected, the default error handling method applies, where the next SDO is configured.
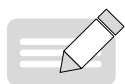
NOTE	1) SDOs and the error handling method are displayed only in expert mode. 2) Be cautious when selecting **Jump to line if error**. SDO configuration may encounter infinite loop if the system jumps to a previous line.

■ Click **Move Up** to move a user-defined SDO to the previous line.

■ Click **Move Down** to move a user-defined SDO to the next line.

■ Click **Add** to add an SDO before the selected SDO. When you click **Add**, the **Select Item From Object**

Dictionary dialog box is displayed. In the **Select Item From Object Dictionary** dialog box, you can modify the SDO value and comment.

■ Click **Edit** to modify the selected SDO. In the displayed **Select Item From Object Dictionary** dialog box, modify the SDO information. Only user-defined SDOs can be modified.

■ Click **Delete** to delete the selected SDO. Only user-defined SDOs can be deleted.

■ **SDO Timeout**: Set the timeout period of an SDO. The value ranges from 0 to 4,294,967, in ms. The default value is 1000 ms.

5 Debugging

The **Debug** page provides the functions of slave NMT control, SDO read and write, and diagnosis information acquisition.

Figure 3-65 **Debug** page

NMT

NMT provides network management services, such as initialization, node start/stop, and failed node detection. The network management services adopt the master-slave communication mode, in which only one NMT master node or station exists.

Figure 3-66 shows the state transition of a slave during startup.
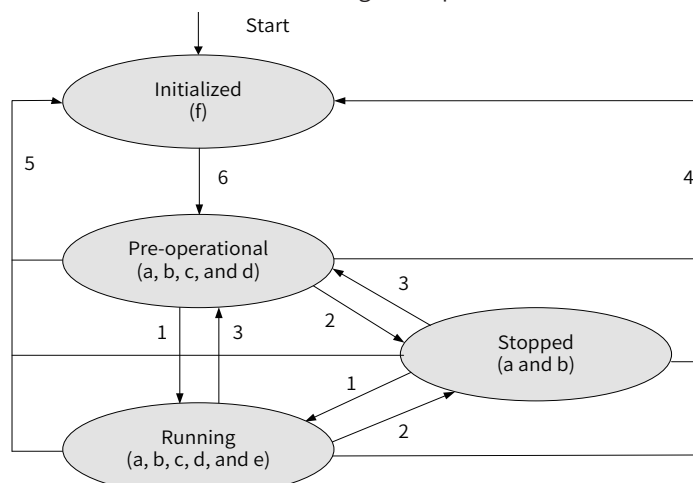
Figure 3-66 State transition of a slave during startup

Note:

a. NMT; b. Node Guard; c. SDO; d. Emergency; e. PDO; f. Boot-up

State transition (1 to 5 are initiated by NMT) sequence, NMT command words (enclosed by brackets):

1: Start_Remote_node (0x01, Start Node)

2: Stop_Remote_Node (0x02, Stop Node)

3: Enter_Pre-Operational_State (0x80, Enter Preoperational State)

4: Reset_Node (0x81, Reset Node)

5: Reset_Communication (0x82, Reset Communication)

6. The device completes initialization, enters the Pre_Operational state, and sends a Boot-up message.

Initialization includes application data initialization and communication initialization. During node reset, all the data of slave nodes is reset. During communication reset, only communication data is reset.

NMT can enable all or some nodes to enter different working statuses at any time.

- **Start Node**: Click this button to run slave nodes. PDO communication can be implemented only when slave nodes are running. When the slave is in the preoperational or stopped state, **Start Node** sets the slave to the running state (state 1 in Figure 3-66).

- **Stop Node**: Click this button to stop slave nodes from running, and all communication except node daemon and heartbeat stops. When the slave is in the preoperational or running state, **Stop Node** sets the slave to the stopped state (state 2 in Figure 3-66).

- **Enter Preoperational State**: Click this button to enable the slave to enter the preoperational state, in which the slave can initiate SDO communication, but not PDO communication. The slave enters the preoperational state after initialization. When the slave is in the running or stopped state, **Enter Preoperational State** sets the slave to the preoperational state (state 3 in Figure 3-66).

- **Reset Node**: Click this button to reset the configuration of the slave. The application configuration and communication configuration are reset in sequence. The slave enters the preoperational state (state 4 in Figure 3-66).

- **Reset Communication**: Click this button to reset the communication configuration of the slave. Only the communication configuration is reset. The slave enters the preoperational state (state 5 in Figure 3-66).

SDO

SDO is used to transmit a large volume of low-priority data between devices. A typical usage of SDO is to configure devices in a CANopen network. On this page, you can read or write SDO object values when slave nodes are running. When reading or writing an SDO object, you need to determine the index, subindex, and bit length of the SDO object. Also, specify the value to be written.

- **Index**: SDO read/write index, ranging from 16#0 to 16#FFFF.

- **Subindex**: SDO read/write index, ranging from 16#0 to 16#FF.

- **Bitlength**: SDO read/write bit length. The optional values are 8, 16, 24, and 32.

- **Data**: SDO value read or written. Enter a hexadecimal value in the first text box, and enter a decimal value after the equal sign (=). The range of the written SDO value is related to the bit length. The minimum value is 0, and the maximum value is indicated by the bit length.

- **Result**: SDO read/write result. An error message is displayed when read/write is abnormal.

Diagnosis

The **Device Diagnosis** page displays the running status and emergency information about slave nodes.

■ **Online Status**: indicates whether the slave is online. If the slave is online, a green icon with the text "Online" is displayed. If the slave is offline, a red icon with the text "Offline" is displayed.

■ **Run Status**: indicates the running status of the slave with an icon, followed by descriptive text. See the following figure.

State	Icon	Descriptive Text
Running	🟢	The slave is running.
Stopped	🔴	The slave is stopped.
Preoperational	🟡	The slave is in the preoperational state.
Initialized	⚫	The slave has been initialized.
Not connected	⚫	The slave is disconnected.
Connecting	⚫	The slave is being connected.
Preparing	⚫	The slave is in the preparation state.
Reset node	⚫	A node is being reset.
Reset communication	⚫	Communication is being reset.
Scan node	⚫	The slave is being scanned.
Configure node	⚫	The slave is being configured.
Start node	⚫	The slave is being started.
Unknown state	⚫	The status of the slave is unknown.

■ **Diag String**: displays the current diagnosis information of the slave.

■ **The Latest Emergency Information**: displays the first unconfirmed emergency message. Incoming emergency messages are not displayed until the current emergency message is confirmed.

An emergency message contains 8 bytes and is in the following format:

sender receiver(s) COB-ID	Byte 0-1	Byte 2	Byte 3-7
0x080+Node_ID	Fault code	Fault register (Object 0x1001)	Manufacturer specific code

The following table lists hexadecimal fault codes. The xx part is defined by the corresponding device subprotocol.

Fault Code (Hexadecimal)	Function
00xx	Error Reset or No Error
10xx	Generic Error
20xx	Current
21xx	Current, device input side
22xx	Current, inside the device
23xx	Current, device output side
30xx	Voltage
31xx	Mains voltage
32xx	Voltage inside the device

Fault Code (Hexadecimal)	Function
33xx	Output voltage
40xx	Temperature
41xx	Ambient temperature
42xx	Device temperature
50xx	Device hardware
60xx	Device software
61xx	Internal software
62xx	User software
63xx	Data set
70xx	Additional modules
80xx	Monitoring
81xx	communication
8110	CAN overrun
8120	Error Passive
8130	Life Guard Error Or Heartbeat Error
8140	Recovered from Bus-Off
82xx	Protocol Error
8210	PDO no processed Due to length error
8220	Length exceeded
90xx	External error
F0xx	Additional functions
FFxx	Device specific

The emergency information includes the time, fault code, fault register, and manufacturer specific code.

■ Time: time when the emergency information is obtained, rather than the time when a fault occurs.

■ Fault code: Place the cursor over a fault code to view the emergency information.

■ Fault register: register that stores emergency information. See the following table.

Fault Register Definition (Bit)	Fault Type
0	Generic
1	Current
2	Voltage
3	Temperature
4	Communication
5	Device profile specific
6	Reserved (=0)
7	Manufacturer

■ Manufacturer specific code: code of the emergency information manufacturer.

■ **Confirm**: Click this button to confirm the emergency information. Only one emergency message is retained. Incoming emergency messages are not displayed until the current emergency message is confirmed.

6 PDO attributes

PDO attributes are used to set PDO communication parameters, including **COB-ID** (communication object identifier), **Transmission Type**, **Inhibit Time**, and **Event Time**. The **Send PDO Properties** dialog box is displayed when you modify or add a PDO.
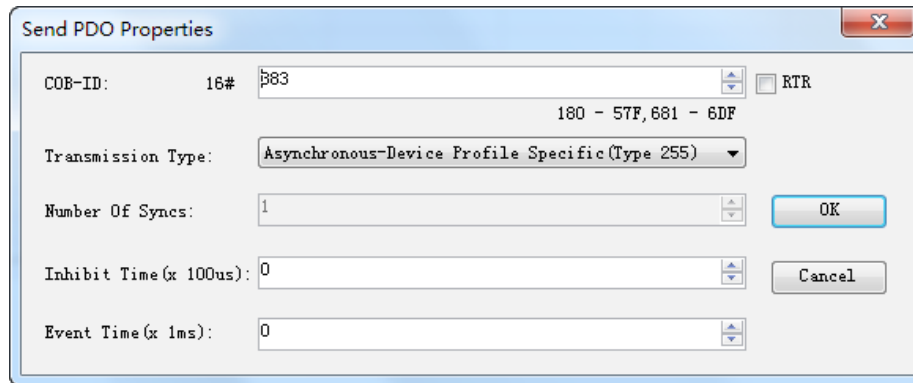
Figure 3-67 **Send PDO Properties** dialog box

■ **COB-ID**: communication object identifier of the PDO. Each COB-ID is unique within the CANopen bus and cannot be the same as other PDO COB-IDs, emergency COB-IDs, and heartbeat COB-IDs. The value range of PDO COB-ID is 16#180-57F, 681-6DF. If it is invalid, you can modify it manually or by using the check and fix configuration function on the master. The default COB-ID of each PDO comes from the object of subindex 01 of the corresponding PDO in the object dictionary. If the object dictionary format is $NodeID+value, the COB-ID changes with the slave node ID. After the COB-ID is changed manually, the original emergency code does not change with the new ID. The COB-ID cannot be changed if the object dictionary corresponding to the COB-ID does not have the write permission.

■ RTR: whether to enable remote frame reception. PDO sending is triggered when remote frames are received. Only sent PDOs are displayed.

■ **Transmission Type**: transmission mode to be applied during PDO communication.

PDO supports the following transmission modes:

1) Synchronous (synchronization is implemented by receiving SYNC objects)

Aperiodic: Sending is triggered by remote frames or object-specific events specified in the device subprotocol.

Periodic: Sending is triggered after 1 to 240 SYNC messages.

2) Asynchronous

Sending is triggered by remote frames or object-specific events specified in the device subprotocol.

The following table lists the different PDO transmission modes defined by transmission types, which are part of the PDO communication parameter objects and defined by an eight-digit unsigned integer.

Transmission Type	PDO Communication Trigger Condition (B = both needed; O = one or both)			PDO Transmission
	SYNC	RTR	Event	
0	B	-	B	Synchronous and non-cyclic
1-240	O	-	-	Synchronous and cyclic
241-251	-	-	-	Reserved
252	B	B	-	Synchronous, after RTR
253	-	O	-	Asynchronous, after RTR

Transmission Type	PDO Communication Trigger Condition (B = both needed; O = one or both)			PDO Transmission
	SYNC	RTR	Event	
254	-	O	O	Asynchronous, manufacturer specific event
255	-	O	O	Asynchronous, specific event defined in the device subprotocol
Note: SYNC: SYNC-object received. RTR: remote frame received. Event: Value changed or timer interrupted. Transmission type: A value ranging from 1 to 240 indicates the number of SYNC objects between two PDOs.				

The default transmission type of each PDO comes from the object of subindex 02 of the corresponding PDO in the object dictionary. If the object dictionary does not have the write permission, the transmission type cannot be changed.

- **Number Of Syncs**: This parameter is related to **Transmission Type** and can be modified only when the value of **Transmission Type** is within the range from 1 to 240. The slave starts processing transmitted PDO data after receiving the number of synchronization frames specified by this parameter.

- **Inhibit Time**: The value of this parameter can be changed only when the value is equal to the product of the minimum interval at which the same PDO transmits two data records and 100 µs. This parameter prevents frequent PDO sending when a value is changed. The value ranges from **0** to **65535**. The default value is **0**. This parameter can be set only when PDOs are sent and **Transmission Type** is set to **254** or **255**. The default value of **Inhibit Time** of each PDO comes from the object of subindex 03 of the corresponding PDO in the object dictionary. The value of **Inhibit Time** cannot be changed if the object dictionary corresponding to **Inhibit Time** does not have the write permission.

- **Event Time**: interval at which the same PDO transmits two data records. The value ranges from **0** to **65535**, in ms. The default value is **0**. This parameter can be set only when PDOs are sent and **Transmission Type** is set to **254** or **255**. The default value of **Event Time** of each PDO comes from the object of subindex 05 of the corresponding PDO in the object dictionary. The value of **Event Time** cannot be changed if the object dictionary corresponding to **Event Time** does not have the write permission.

7 Adding an object

In the **Select Item From Object Dictionary** dialog box, you can add and modify receive PDO mappings, send SDO mappings, or SDOs. During SDO operation, this dialog box adds the SDO **Value** text box and the SDO **Comment** text box.
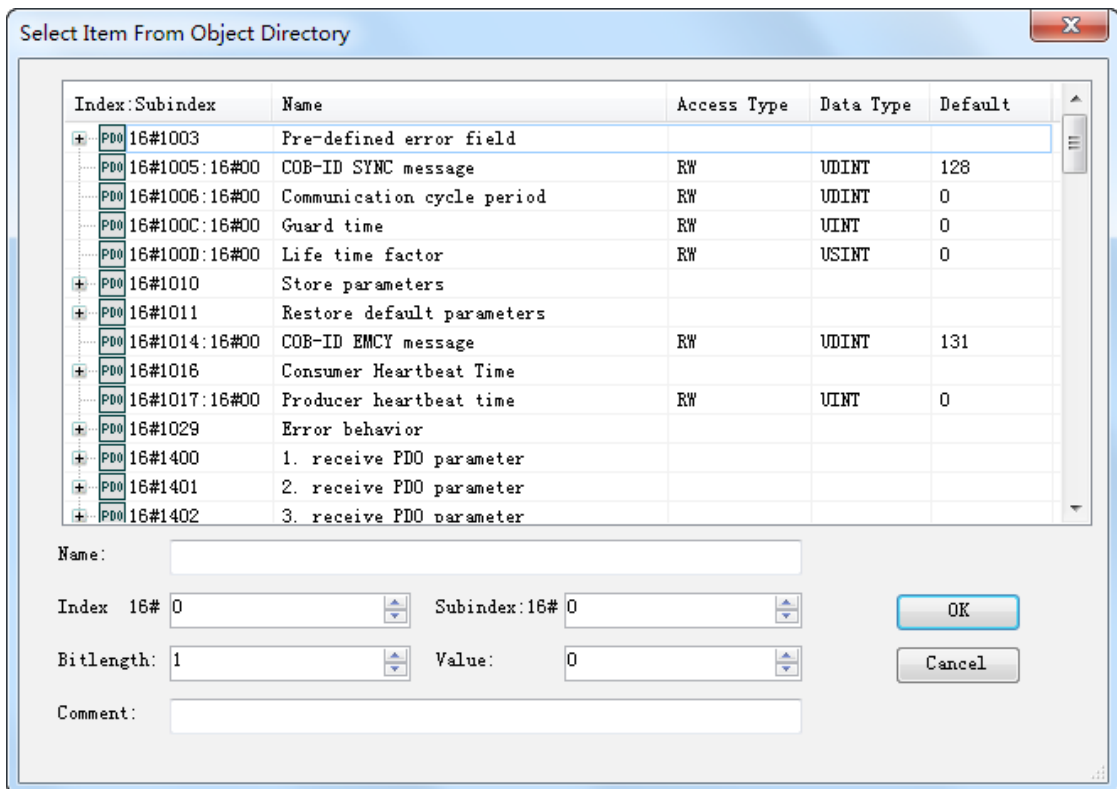
Figure 3-68 Select Item From Object Dictionary dialog box

Object list: lists the objects in the EDS file. When receive PDO mappings are modified, only the objects with the RW, RWW, or WO access permission and with an index greater than 16#2000 are displayed. When send PDO mappings are modified, only the objects with the RW, RWR, RO, or CONST access permission and with an index greater than 16#2000 are displayed. When SDOs are modified, only the objects with the RW, RWW, RWR, or WO access permission are displayed.

> **NOTE**
> When the SDOs of AM600 slaves are modified, the objects with an index within the range from 16#2000 to 16#40df cannot be displayed. These parameters are used by module configuration and set in the module.

■ **Index**: index of an object, ranging from 16#0 to 16#FFFF. After an object in the object list is selected, its index is displayed.

■ **Subindex**: subindex of an object, ranging from 16#0 to 16#FF. After an object in the object list is selected, its subindex is displayed.

■ **Bitlength**: bit length of an object, ranging from 0 to 32. After an object in the object list is selected, its bit length is displayed.

■ **Value**: SDO value. It is displayed only when SDOs are modified. Its value range is related to the data type of the selected object. After an object in the object list is selected, its value is displayed.

■ **Comment**: SDO comment. It is displayed only when SDOs are modified. The value contains a maximum of 50 characters.

8 CANopen slave I/O mapping

This page is displayed only when the slave is not of the AM600 series. The I/O mappings of AM600 slaves correspond to the AM600 I/O modules and are not displayed here. For the general description of I/O mapping and instructions on this page, see "I/O mapping."

9 State

The state configuration editor for the CANopen bus or modules displays state information (such as Running and Stopped) and the status of the internal bus system.

10 Information

The following basic information about the currently available device is displayed: name, vendor, type, version, module number, description, and image.

3.7.4 CANopen Module

1 Modular device and non-modular device

In CANopen slave configuration, you can connect CANopen slave nodes to the following two types of modular devices:

Modular device: It is connected to a CANopen slave node and provides an I/O mapping list. The **CANopen Slave I/O Mapping** dialog box is not required. The PDO mappings of slave nodes increase as modules are added. Currently, the AM600 I/O module is a type of modular device.

Non-modular device: The slave node dialog box includes the I/O mapping dialog box. PDO mappings cannot be configured automatically.

2 AM600 CANopen I/O module

The AM600 CANopen I/O module is added in hardware configuration. A PDO mapping is automatically added when an I/O is added. For details about the relationship with PDO mappings, see "Receive PDO and Send PDO." After an I/O is added, you can set I/O parameters and add mappings to refresh data. For details, see "CPU-specific I/O module."

3.7.5 Programming Interface

CiA-405 library. For details, see section "6.3.1 CiA405".

3.8 CANlink 3.0 Configuration Editor

The CANlink protocol is a real-time CAN bus application-layer protocol developed by Inovance based on the CAN 2.0 bus protocol. CANlink is mainly used for high-speed and real-time data exchange between Inovance products, such as PLCs, AC drives, servo controllers, and remote extension modules. Read this section carefully before using the CANlink function of the PLC of the AM600 series.

CANlink 3.0 adopts the master/slave mode. A network must have a single master and may have 1 to 62 slaves. The master and slave numbers range from 1 to 63, and each number must be unique.

1) Master/Slave running status monitoring through heartbeat

2) Bus usage warning and real-time bus usage monitoring

3) Reconnection upon disconnection

4) Hot access

5) 256 configuration records (including time trigger, event trigger, and synchronization trigger) sent by the master

6) 16 configuration records (including time trigger, event trigger, and synchronization trigger) sent by a single slave, amounting to 256 configuration records sent by all slaves

7) Point-to-multipoint data received by each station from other eight stations

8) Master-slave data exchange and slave-slave data exchange

9) Up to 128 data records written synchronously to the master

3.8.1 CANlink 3.0 Network Structure

1 Communication distance

A CANlink 3.0 network consists of 1 master and 1 to 62 slaves. The specific number of slaves is related to the baud rate.

Baud Rate	Maximum Communication Distance	Communication Cable Diameter	Number of Connected Stations
1000 kbit/s	20 m	$\geqslant 0.3$ mm^2	18
500 kbit/s	80 m	$\geqslant 0.3$ mm^2	32
250 kbit/s	150 m	$\geqslant 0.3$ mm^2	63
125 kbit/s	300 m	$\geqslant 0.5$ mm^2	63
100 kbit/s	500 m	$\geqslant 0.5$ mm^2	63
50 kbit/s	1000 m	$\geqslant 0.7$ mm^2	63

The preceding data is obtained under the premise of using standard shielded twisted pairs. The maximum number of connected stations (master and slaves) is determined based on the current baud rate.

2 Supported CANlink 3.0 devices

A CANlink 3.0 network must have a single master, which is a PLC of the AM400, AM600, or AC800 series. The network may have 1 to 62 slaves, including AM400, AM600, or AC800 (300 of D8280 indicates support), remote extension modules (51210 or versions later than 52210), 214 non-standard IS500 servos (H00-02=214.xx), IS620P (H01-00=6.0 or greater), IS700 (H01-00=301.05), MD310 (F7-11=u37.18 or greater), and MD380 (F7-11=4.71.06 or greater). Some products must be configured with a CANlink communication extension card to use the CANlink function. For details, see the user manual of the specific product.

3 Special elements of CANlink 3.0 supported by AM600
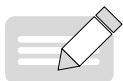
NOTE: The CANlink function of AM600 uses the SD and SM soft elements, which are similar to the D and M elements of small-sized PLCs. However, they do not have a mapping relationship.

SD Range	Function	SM Range	Function
0 to 7000	User-specific word soft element area (available range in the CANlink configuration table)	0 to 3071	User-specific bit soft element area (available range in the CANlink configuration table)
7000 to 7999	User-specific word soft element area	3072 to 7999	User-specific bit soft element area
8000 to 8999	System-defined special register element area	8000 to 8999	System-defined special bit element area, which is currently only used by CANlink

SD Range	Function	SM Range	Function
9000 to 9999	System-defined special register element area which is currently only used by high-speed I/O	9000 to 9999	System-defined special register element area which is currently only used by high-speed I/O

The following table lists the special soft elements involved in the AM600 CANlink function (for details, see the CANlink 3.0 standard).

Special Soft Element	Attribute
SD8100 to SD8163	SD8100 indicates the current status of the local node. SD81xx indicates the current status of the node with the station number xx. For example, SD8101 indicates the current status of the node with station number 1. Meanings of register values: 0: unconfigured; 1: configured; 2: online; 5: offline
SD8164 to SD8239	Reserved
SD8240	Bus loading capacity (monitoring in the programming software)
SD8241	CAN 3.0 slave status
SD8242	CAN 3.0 slave status
SD8243	CAN 3.0 slave status
SD8244	CAN 3.0 slave status
SD8245	Number of received frames
SD8246	Receive error count
SD8247	Send error count
SD8287	Manufacturer code
SD8288	Product series
SD8289	Product model
SD8290	Firmware version
SD8307	CAN 3.0 error (command error code)
SD8308	CAN 3.0 error (configuration error code)

3.8.2 General CANlink Use Process

The general process of using CANlink is as follows:

1) Design the CANlink hardware network structure.

2) On the **Network Configuration** tab page, activate the CANlink bus. The AM600 CPU can function as the CANlink master or a CANlink local slave. Bus devices are automatically added after the bus is activated.

3) If the AM600 CPU functions as the CANlink master, in the CANlink network configuration wizard, you can set the master parameters and add or delete slaves. Slaves refer to remote slaves. If the AM600 CPU functions as a CANlink local slave, you can set other parameters.

4) Set the send parameters, receive parameters, and synchronization parameters properly.

5) Control CANlink transmitted data through soft elements in programs.

6) Control master and slave start/stop and monitor the master/slave running status on the **Network Management** page.

3.8.3 CANlink Network Configuration

Activate the CANlink bus in network configuration before configuring a CANlink network. After the CANlink master is activated, the CANlink master node is added to the device tree. When configuring the first CANlink network, double-click the node to display the network configuration wizard. After a CANlink slave is activated, the CANlink slave node is added to the device tree. Double-click the node to display the local slave configuration page.

1 CANlink 3.0 network configuration wizard

The network configuration wizard is displayed when you configure the CANlink bus for the first time or click **Site Management** on the **Network Management** page. Set the master and slave parameters in the network configuration wizard.

2 Master configuration

Figure 3-69 Master parameter settings

■ **Master Site No.**: identifies a station as the master in the network. The master manages and monitors the entire CANlink 3.0 network. **Master Site No.** must be consistent with the number of the configured PLC.

Network information

■ **Baudrate**: Select a baud rate to be used by the network. The baud rate of each station must be consistent with the selected one. If no baud rate is selected here, the baud rates of the connected stations must be consistent.

■ **Network Heartbeat**: The value ranges from 10 to 20,000, in ms. The master and slaves send heartbeats to the network periodically to monitor each other. When a communication error occurs, the master or slave triggers an alarm and handles the error. When **Network Heartbeat** is set to a smaller value, monitoring is more sensitive and the network usage of heartbeats is higher. An alarm is automatically triggered when the network usage of heartbeats exceeds 10%.

NOTE | If you deselect this parameter, the heartbeat function is disabled and the system cannot monitor the network.

Network management

■ Network Start/Stop Element(SM): Control the CANlink network to start and stop by using the SM8290 soft element. When SM8290 is set to **TRUE**, the CANlink network starts; otherwise, the CANlink network stops.

■ Syn Send Trigger Element (SM): Control the synchronous write function for the configuration sent by all stations by using the SM8291 soft element. For details, see "Configuring synchronous trigger."

Synchronous write trigger element

The synchronous write trigger element is used by the master to synchronize configuration. Up to eight synchronous write trigger elements can be configured. You can leave this parameter unspecified when the master does not need to synchronize configuration. For details, see "Synchronous write by the master."

3 Adding and deleting a slave

This page allows you to add, modify, and delete CANlink slaves.
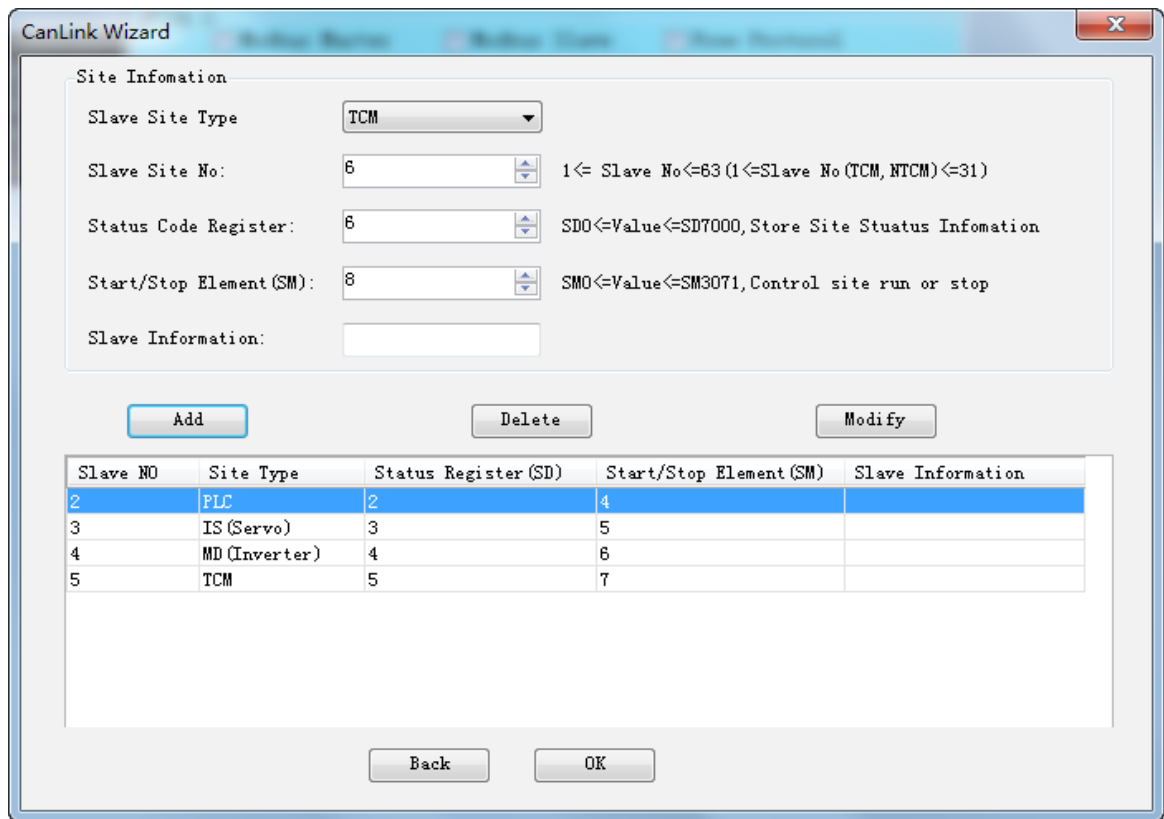
Figure 3-70   Page for adding and deleting a slave

Site information

■ **Slave Site Type**: supported type of the slave station, which can be the PLC, servo, AC drive, temperature control module, and non-temperature control module.

■ **Slave Site No.**: identifies a slave in the network. It must be different from the number of the master or any other slave.

■ **Status Code Register**: SD element that indicates the running status of the corresponding slave. These elements cannot be the same as other slave status register elements. The status may be Running or Faulty (excluding offline).

- **Start/Stop Element**: SM element used by the master to start and stop slaves. These SM elements can be used to start and stop slave communication during network runtime. These elements cannot be the same as other slave start/stop elements and synchronous write trigger elements.

- **Slave information**: comment about a slave, containing up to 32 characters.

- **Add**: Click this button to add the slave with the configured information to the slave list. The system checks whether **Status Code Register** and **Start/Stop Element** are unique.

- **Delete**: Click this button to delete the selected slave from the slave list.

- **Modify**: Click this button to modify the settings of the selected slave. The system checks whether **Status Code Register** and **Start/Stop Element** are unique.

After the slave settings are complete, click **OK** to enter the **Network Management** page.

3.8.4 Network Management

On the **Network Management** tab page, you can start and stop the network, trigger synchronous sending, start and stop monitoring, start and stop slaves, enter the station configuration wizard, modify the network information of stations, and clear all the configuration.

Figure 3-71 **Network Management** page

Network information

- **Baudrate**: Select a baud rate to be used by the network. The baud rate of each station must be consistent with the selected one. If no baud rate is selected here, the baud rates of the connected stations must be consistent.

- **Network Heartbeat**: The value ranges from 10 to 20,000, in ms. The master and slaves send heartbeats to the network periodically to monitor each other. When a communication error occurs, the master or slave triggers an alarm and handles the error. When **Network Heartbeat** is set to a smaller value, monitoring is more sensitive and the network usage of heartbeats is higher. An alarm is automatically triggered when the network usage of heartbeats exceeds 10%.

> **NOTE**
> If you deselect this parameter, the heartbeat function is disabled and the system cannot monitor the network.

- **Cycle Period**: used to estimate the loading capacity of CANlink.

- **Network Load**: CANlink network load, including CANlink receiving and sending, all heartbeat loads, real-time load of the CANlink bus obtained during monitoring (SD8240 register value), and estimated network load under non-monitoring conditions.

- Background color of **Network Load**:

Color	Range (%)	Load Description
Green	0 to 50	Load percentage, such as 10%
Yellow	51 to 75	Load percentage, such as 55%
Red	76 to 90	Load percentage, such as 78%
Red	Greater than 90	ERR

■ **Heartbeat Load**: heartbeat load of CANlink, which is calculated through estimation. An estimated value is obtained after login.

■ Background color of **Heartbeat Load**:

Color	Range (%)	Load Description
Green	0 to 10	Load percentage, such as 10%
Red	Greater than 10	ERR

Network management

The network management function is available only after login to the PLC.

■ **Start Network**: Click this button to start and stop the CANlink network by using the SM8290 element.

■ **Start Monitor**: Click this button to start CANlink network monitoring and obtain the CANlink master/ slave running status periodically. The online status of stations is updated in the station list.

■ **Sync Send Trigger**: Click this button to trigger a synchronous write operation.

Station configuration

■ **Device Type**: Select the device type to be displayed in the station list.

■ **Slave Start/Stop**: Set the slave to the running state by setting the slave start/stop element to **TRUE**.

■ **Site Management**: Click this button to display the network configuration wizard dialog box, where you can configure a CANlink network.

■ **Clear Settings**: Click this button to clear all the CANlink configuration and restore the master configuration to the default.

■ Station list: displays the information and status of CANlink stations. The following columns are displayed: **Site**, **Device Type**, **Online Status**, **Status Register**, **Status Code**, **Start/Stop Element**, and **Slave Information**. Double-click a slave in the list to display the the page for send configuration, receive configuration, or synchronous master write.

■ **Site**: unique identifier of a station.

■ **Device Type**: device type of the station, which can be the PLC, servo, AC drive, temperature control module, and non-temperature control module.

■ **Online Status**: Click **Start Monitor** after login to the PLC to display the slave status. The online status of a slave is obtained through the SD8240, SD8241, SD8242, and SD8243 online status registers. If the slave is online, the system determines whether the slave is running or stopped by obtaining the value of the slave start/stop element. The online status register is read-only and cannot be modified. The following table lists the mapping relationships between each bit of the online status register and the station number.

Soft Element	Bit	Slave Number
SD8243	Bit 15 to bit 0	32-47
SD8242	Bit 15 to bit 0	48 to 63
SD8241	Bit 14 to bit 0	1 to 16
SD8240	Bit 15 to bit 0	17 to 31

■ **Status Register**: register that stores the slave running status. The running status register is read-only and cannot be modified.

The following table lists each bit of the running status register.

Bit	Description
Bit 0	Indicates the fault status. **1** indicates that the node is faulty, and **0** indicates that the node is normal.
Bit 1	Indicates the running status. **1** indicates that the node is running, and **0** indicates that the node is stopped.
Bit 2	Indicates whether the device is ready. **1** indicates that the device is ready, and **0** indicates that the device is not ready. This bit is only applicable to servos.
...	Reserved
Bit 15	Reserved

■ **Status Code**: value of the slave status register, which indicates the status code defined by the slave.

■ **Start/Stop Element**: Start or stop the element.

■ **Slave Information**: comment about the slave.

3.8.5 Send Configuration

CANlink send configuration is divided into time trigger configuration, event trigger configuration, and synchronous trigger configuration. Event trigger configuration is divided into PLC event trigger and non-PLC event trigger.

1 Time trigger configuration

Time trigger configuration is a common configuration. The send station writes the source address to the target address of the target station based on the configured time cycle. That is, the target station reads the source address of the send station cyclically and saves the source address to the target address.

If the send station and receive station are the same, it is a point-to-multipoint configuration. All the stations configured to receive point-to-multipoint data from this station can receive such data. For details, see "Receive configuration."
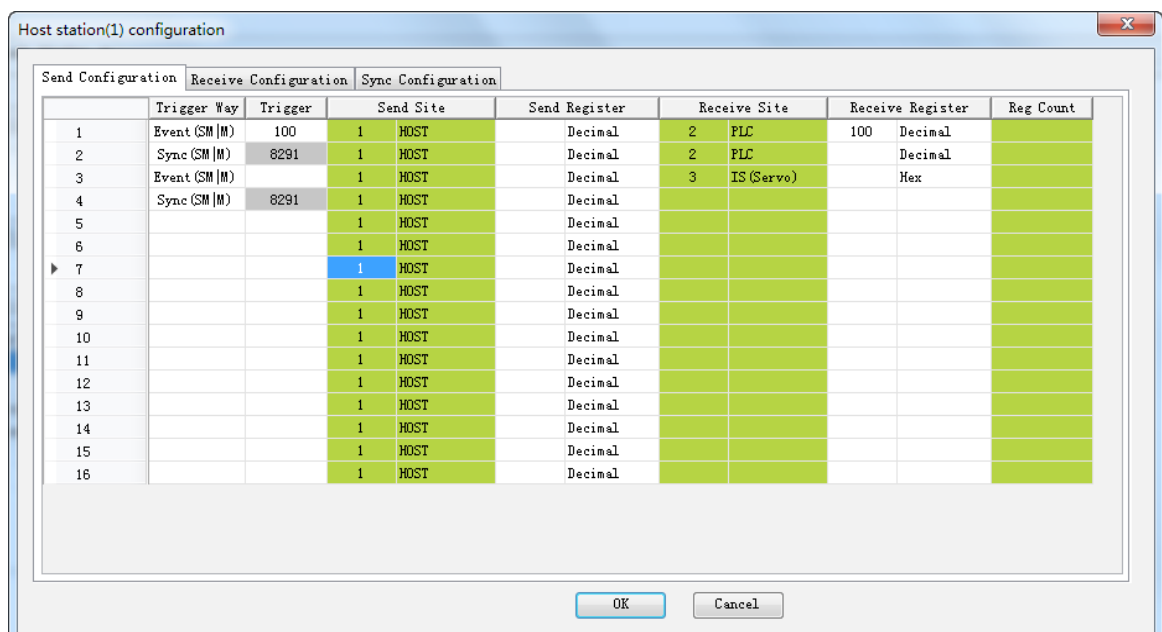
Figure 3-72 Time trigger configuration page

Station 1 writes the values of the neighboring registers SD102 and SD103 to H0200 and H0201 of station 3 every 100 ms.

In the fourth line of Figure 3-73, the send station and receive station are both station 1, so the configuration is a point-to-multipoint configuration. Station 1 sends the data of SD110 to the network in point-to-multipoint mode every 100 ms. All the stations configured to receive point-to-multipoint data from station 1 can receive the frame.

The time ranges from 1 to 30,000, in ms. The shorter the time, the higher the data update speed is.

2 Event trigger configuration

Event trigger is a real-time trigger configuration. Sending is triggered immediately when conditions are met. If conditions are not met, sending is not triggered, regardless of the interval from the last sending.

The event trigger of PLCs is slightly different from that of other products based on product features.

PLC event trigger configuration

The corresponding event is triggered when SM used as the trigger condition is set to ON. The event trigger range is SM0 to 3071. A prompt is displayed when the range is exceeded.
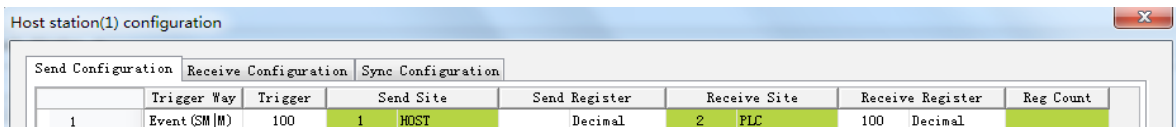
Figure 3-73 PLC event trigger configuration

As shown in Figure 3-74, when SM100 is set to 1, station 1 sends the value of SD100 to SD100 of the PLC of station 2. Then, SM100 is automatically reset.

Non-PLC event trigger configuration

Except PLCs, the event trigger of AC drives, servos, and extension modules is based on register value change and the minimum interval from the last sending.
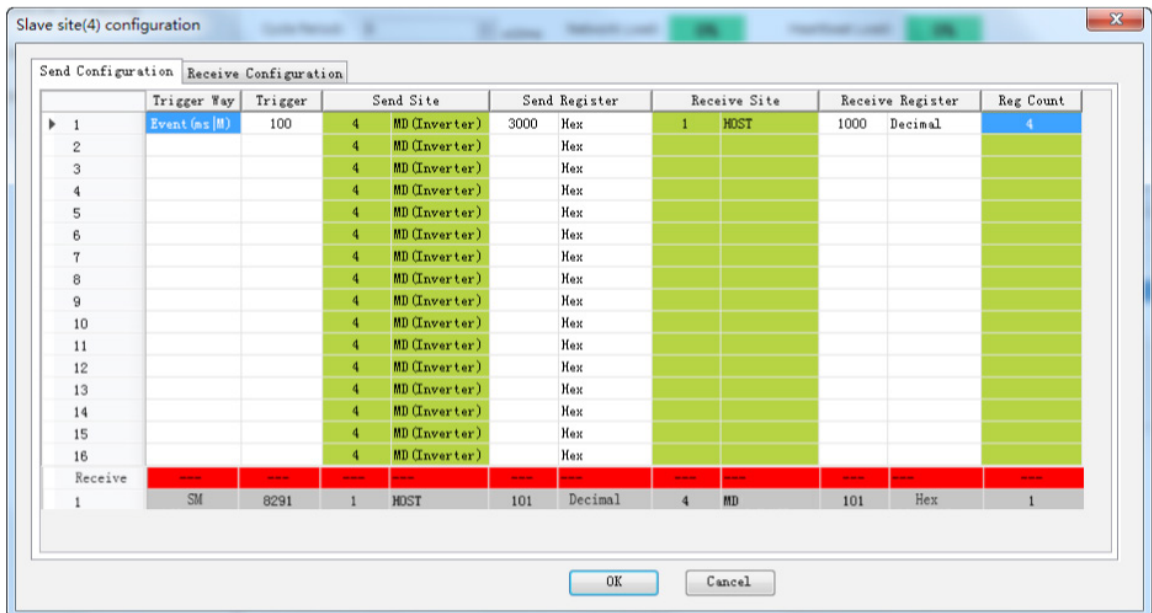
Figure 3-74   Non-PLC event trigger configuration

As shown in Figure 3-75, when H3000 of the AC drive of station 4 is changed, sending is triggered immediately if the interval from the last sending of the configuration reaches 100 ms. If the interval from the last sending is less than 100 ms, sending is not triggered until the specified time is reached. A shorter interval results in better real-time effect but has greater impact on the network.

The minimum interval ranges from 1 to 30,000, in ms. A prompt is displayed when the range is exceeded.

3 Synchronous trigger configuration

When the master configured with synchronous trigger detects that the synchronous sending trigger element SM8291 is reset, the master broadcasts commands to the network to require all the stations in the network to trigger synchronous sending in sequence. The trigger condition is configured by SM8291. The master resets SM8291 after sending a command frame. As shown in Figure 3-76, master 1, PLC slave 2, and MD380 slave 4 are configured with synchronous trigger. If SM8291 of the master is reset, the three stations send synchronous trigger data.
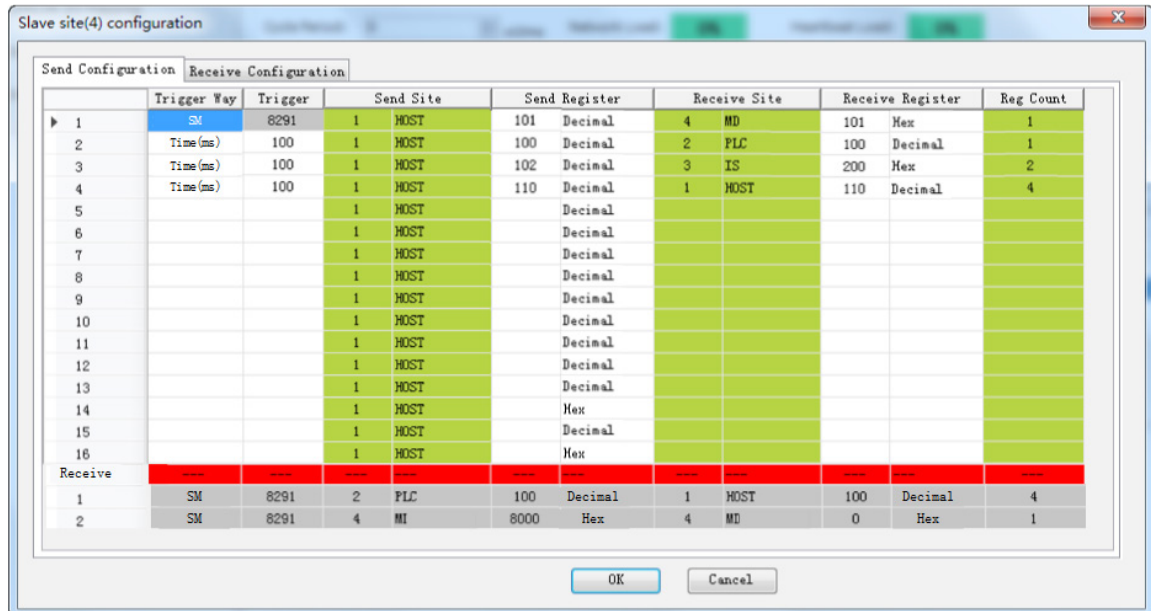
Figure 3-75   Synchronous trigger configuration

SM8291 is used in the same way as the SM element of PLC event trigger. You can perform the corresponding operation by clicking **Sync Send Trigger** on the CANlink 3.0 main interface.

Only SM8291 of the master can operate SM8291 of the slave PLC without impact.

In essence, synchronous trigger is a type of event trigger in which SM8291 of the master is used as an event by all the stations in the network.

4 Differences among time trigger, event trigger, and synchronous trigger

Item	Time Trigger	Event Trigger	Synchronous Trigger
Sending mode	Scheduled and cyclic sending.	Sending is triggered in the case of an event or data change. The event is automatically cleared after sending.	Sending is enabled by the master. The event is automatically cleared after sending.
Real-time effect	The real-time effect is related to the configured time. The shorter the time, the better the real-time effect is.	Sending is triggered in the case of an event, with good real-time effect.	Sending is triggered when M8291 of the master is reset. The real-time effect is better than that of event trigger.
Execution times	Scheduled sending.	Triggered once in the case of an event.	Triggered once in the case of an event.
Programming in PLC	Scheduled sending, without program design.	Sequential logic design in program.	Sequential logic design in program.
Network usage	High.	Low.	Low.

Item	Time Trigger	Event Trigger	Synchronous Trigger
Application scenario	There are no special time sequence requirements. Data can be written continuously.	There are time sequence requirements. Data is written once, or continuous writing may cause a fault.	The master requires data returned by multiple slaves in special scenarios.

5 Send configuration editor
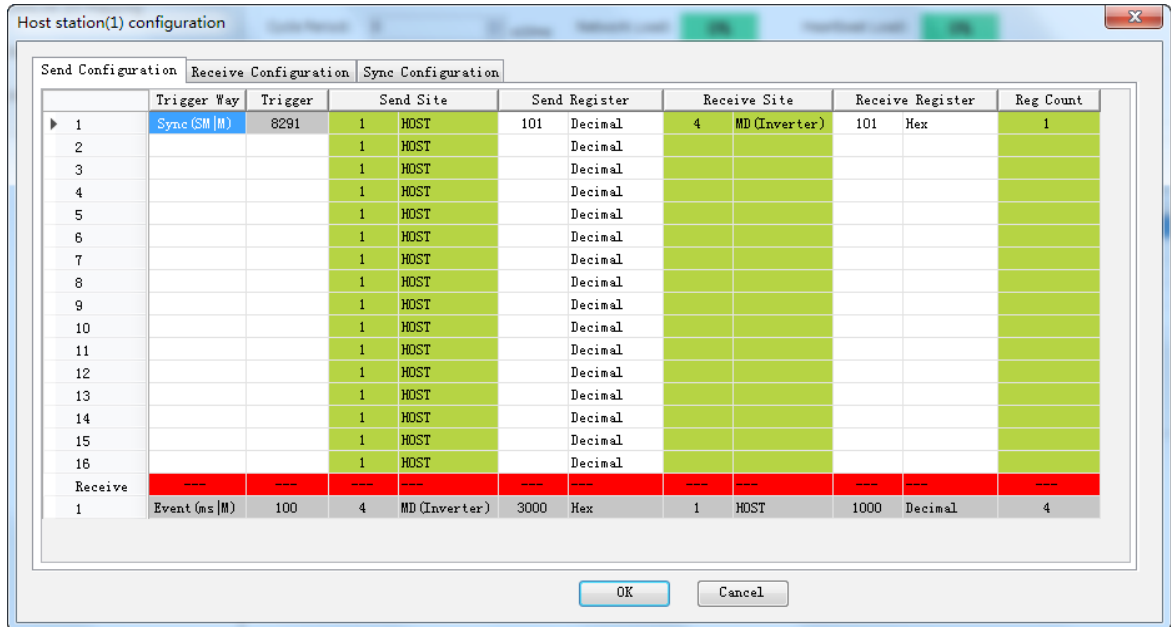
Figure 3-76 Send configuration editor

On the **Send Configuration** interface, you can configure sending for a station. The send list is divided into two parts. The first part displays the configuration that the local station sends to other stations, whereas the second part displays the configuration that other stations send to the local station.

For the master, up to 256 data records can be configured for sending; for slaves, up to 16 data records can be configured for sending.

The send list contains the following columns: **Trigger Way**, **Trigger**, **Send Site**, **Send Register**, **Receive Site**, **Receive Register**, and **Reg Count**.

■ **Trigger Way**: trigger type of data sending by stations. The options are time trigger, event trigger, and synchronous trigger.

■ **Trigger**: trigger condition of data sending by stations. Configuration is sent when the trigger condition is met. When the trigger type is time trigger and the range is 0 to 30,000, the station sends configuration periodically based on the trigger value. When the trigger type is event trigger and the trigger condition is TRUE, the station sends configuration if it belongs to the host or PLC type, the trigger is an SM soft element, and the range is 0 to 3071. The station sends configuration periodically if it is a servo, AC drive, temperature control module, or non-temperature control module, and the range is 0 to 30,000. Sending is executed when the trigger mode is synchronous trigger and soft element SM8291 is set to TRUE.

■ **Send Site**: station that sends configuration. It cannot be changed.

■ **Send Register**: register corresponding to the send station. If the send station belongs to the PLC type, the range is 0 to 7000 (which indicates the register value plus the number of registers). If the send station is a temperature control module, the range is 0 to 499 or 700 to 722. If the send station is a non-temperature control module, the range is 0 to 63 or 700 to 722. If the send station is a servo or AC drive, the range is 0 to 65,535.

- **Receive Site**: station that receives configuration. It must be an existing station.

- **Receive Register**: register of the station that receives configuration. For point-to-multipoint communication (the send station and receive station are the same), the range is 0 to 65,535. If the receive station belongs to the PLC type, the range is 0 to 7000. If the receive station is a temperature control module, the range is 0 to 499 or 721 to 722. If the receive station is a non-temperature control module, the range is 0 to 63 or 721 to 722. If the receive station is a servo or AC drive, the range is 0 or 65,535.

- **Reg Count**: number of send or receive registers. The value range is 1 to 4. The total length of **Reg Count** and **Send Register** or **Receive Register** cannot exceed the specified limit.

3.8.6 Receive Configuration

On the Receive Configuration interface, you can configure stations to receive point-to-multipoint data from the network.
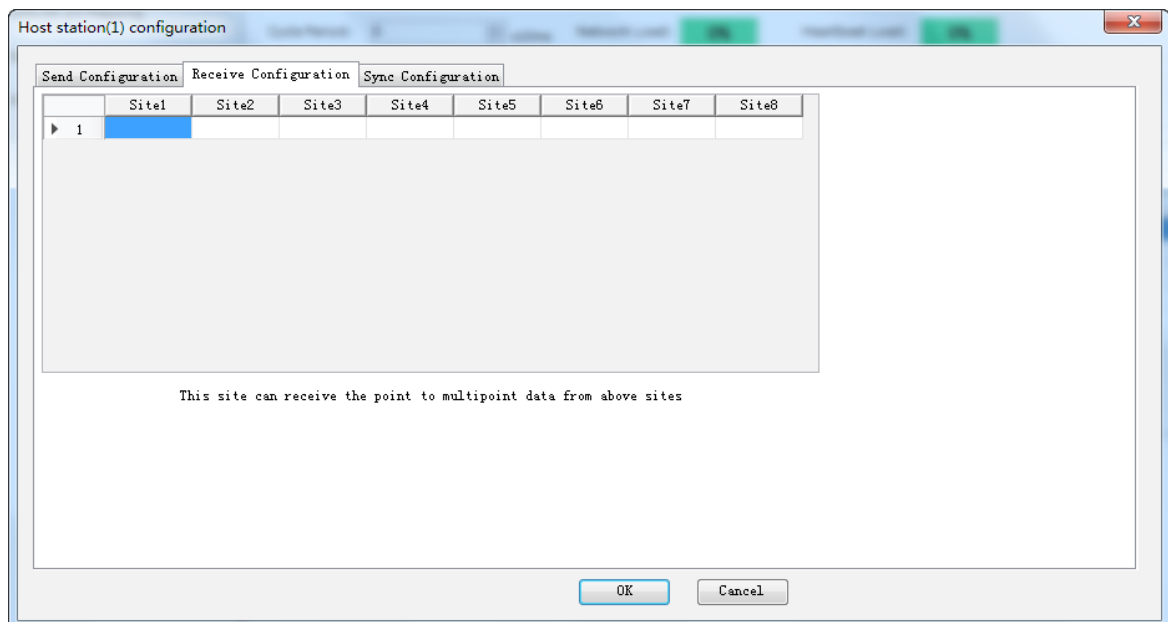
Figure 3-77   **Receive Configuration** page

As shown in Figure 3-78, the receive configuration of slave 2 contains stations 1 and 3, indicating that slave 2 only receives point-to-multipoint data frames from stations 1 and 3. If the destination register address of the received point-to-multipoint data meets the definition of station 2, the received data takes effect; otherwise, the received data is discarded.

Each station can receive point-to-multipoint data from up to eight stations. Each send station does not limit the number of stations that receive point-to-multipoint data.

The point-to-multipoint sending function allows a station to modify the same parameter number on multiple stations and implement synchronization.

3.8.7 Synchronous Write by the Master

Synchronization configuration is specific to the master and allows the master to write the multiple registers or parameter numbers of one or more slaves. For example, the master can control multiple servos to start or stop simultaneously by writing the H31-00 parameter number.

1 Trigger element

For example, after you enter a trigger element in Host Syn Write Trigger Element (SM) in the configuration wizard, double-click the master on the main interface to enter master configuration. On the Sync Configuration interface, select a configuration condition from the **Trigger Condition** drop-down list. If a trigger element is not entered in the configuration wizard, you can still open the **Sync Configuration** page but the **Trigger Condition** drop-down list is unavailable. In this case, click **Site Management** on the main interface to open the configuration wizard again and add or delete trigger elements.
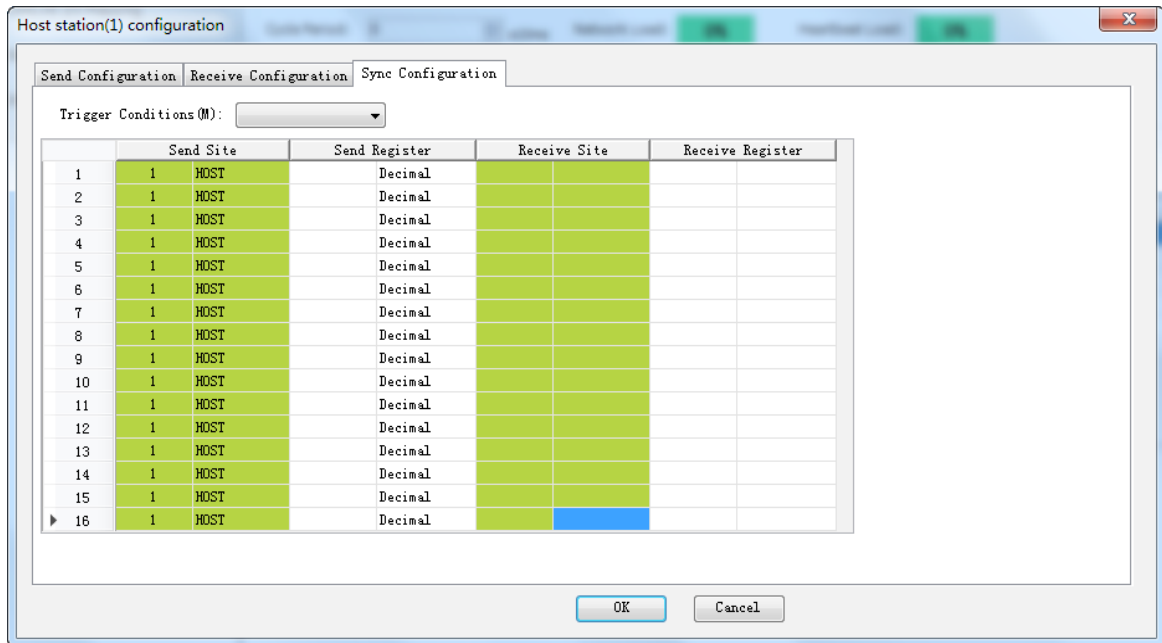
Figure 3-78   **Sync Configuration** page for the master

When a synchronous write trigger element of the master is set to ON, all the configuration data under the trigger element is sent in sequence. The slaves receive the data and store it in the buffer. When the master detects that all the synchronous writes under the trigger condition are successfully sent, the master broadcasts an effectiveness command so that all the receive slaves retrieve data from the buffer and make the data effective.

The master supports synchronous write based on up to 8 different trigger conditions, each of which can be configured with 16 synchronous writes. A single slave can receive up to 8 synchronous writes. A prompt is displayed when this limit is exceeded.

2 Operation on the 32-bit register of the servo

CANlink 3.0 supports operations on 16-bit registers and parameters. To operate 32-bit registers or parameters, use the following method:

Write the upper and lower 16-bit addresses of 32-bit registers or parameters through the synchronous write function of the master.

For example, for parameters H11-12 (1-segment displacement) of the servo, the following operation writes SD1000 and SD1001 of the master to H11-12 of servo 3 as lower 16 bits and upper 16 bits, respectively.

Figure 3-79 Two 16-bit registers combined in the 32-bit format

It is recommended that you use the SET statement of the upper- and lower-edge conducting trigger element of the M element. When operating the parameters of 32-bit addresses, you cannot only operate the addresses of the lower and upper 16 bits or split the addresses of the lower and upper 16 bits of 32-bit parameters to different trigger conditions.

You can use general send configuration to write the values of two consecutive SD elements to the lower address bits of the 32-bit parameters of the servo, as shown in Figure Figure 3-18.

Figure 3-80   Writing a 32-bit value to two 16-bit registers

3.8.8 Local Slave Configuration

A local slave is a PLC working as a CANlink slave. A remote slave is attached to the master, such as a servo, AC drive, temperature control module, or PLC. A PLC can work as a master or local slave.

Figure 3-81   Dialog box for local slave configuration

Site No.: identifies a local slave. It cannot be the same as the number of any other station in the CANlink network. The value ranges from 1 to 63.

Baudrate: baud rate of data communication of the local slave. It must be consistent with the baud rate of the master.

3.8.9 Device Access to the CANlink 3.0 Network

1 Connecting an AM600 series PLC to the CANlink 3.0 network

The station number and baud rate of AM600 can be set only in software. To make the settings take effect, you need to restart the machine or download the running program again.

Each station number in the same network must be unique. If station numbers are repeated, subsequently connected stations disable communication. Station number 0 is not allowed in the network.
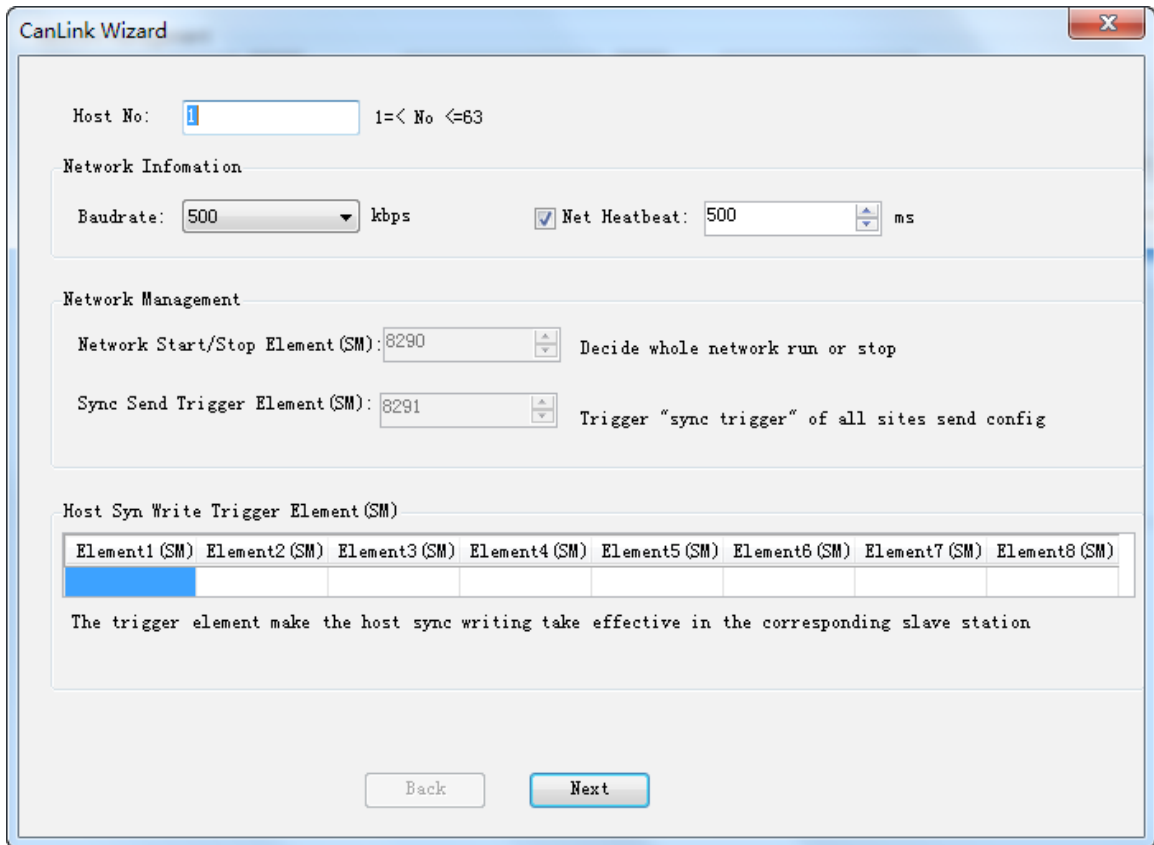
2 Connecting the MD380/500 AC drive to the CANlink 3.0 network

Installing the MD38CAN1 extension card of the AC drive

Insert the MD38CAN1 extension card into the Inovance AC drive. The extension card (CANlink) cannot be installed or uninstalled in the power-on state. Power off the AC drive before installation, and wait 10 minutes until the power indicator of the AC drive is off. Install the extension card in accordance with the following figure.

Insert the MD38CAN1 extension card into the AC drive and fasten screws.
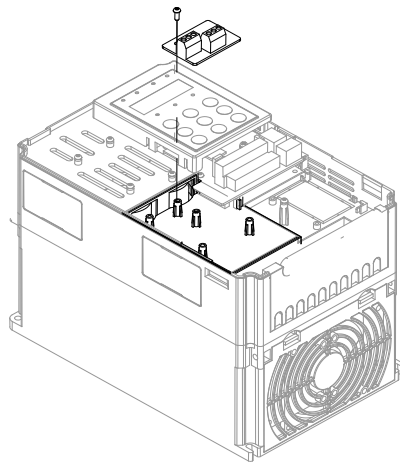
Configuring the MD380/500 AC drive

To configure start/stop control of the AC drive through the CANlink network, set the command source of the AC drive to be the communication command channel. That is, set F0-02 to 2. If start/stop control is not required, set F0-02 based on the actual situation.

Setting the station number

Set the station number through the Fd-02 parameter. The value ranges from 1 to 63. A value beyond the range may result in access failure of CANlink 3.0. The modified value takes effect immediately. Each station number in the network must be unique. If station numbers are repeated, subsequently connected stations may disable communication.

Setting the baud rate

Set the baud rate through the Fd-00 parameter. The thousands place indicates the baud rate of CANlink. The following table lists the mapping relationships.

Parameter Address	Name	Setting Range	Default Value
HFd-00	Baud rate	Ones place: Modbus Tens place: PROFIBUS DP Hundreds place: reserved Thousands place: CANlink 0: 20 kbit/s 1: 50 kbit/s 2: 100 kbit/s 3: 125 kbit/s 4: 250 kbit/s 5: 500 kbit/s 6: 1000 kbit/s	5005

The baud rates of all the stations in the CANlink 3.0 network must be consistent; otherwise, communication may be abnormal.

NOTE

MD380/500 AC drives do not support the 800 kbit/s baud rate.

3 Connecting the MD310 AC drive to the CANlink 3.0 network

Installing the MD310-CANL card

Insert the MD310-CANL card into the Inovance AC drive. Power off the AC drive before installation, and wait 10 minutes until the power indicator of the AC drive is off. Install the extension card in accordance with the following figure.

1. Push out the hooks on both sides by using a tool.

Housing

2. Grab the edges of the cover to lift it.

Hook slot

3. Lift the cabling plate.

4. Lift the baffle plate away from the keypad cover

Cabling plate

5. Press the extension card between the two buckles on the left, split the right-side wall of the middle enclosure, and press the extension card in position on the right.

6. Ensure that the extension card is fastened properly by the four buckles.

Buckle

Buckle
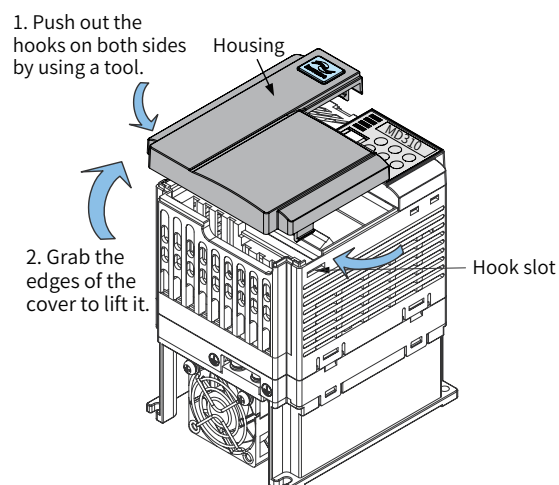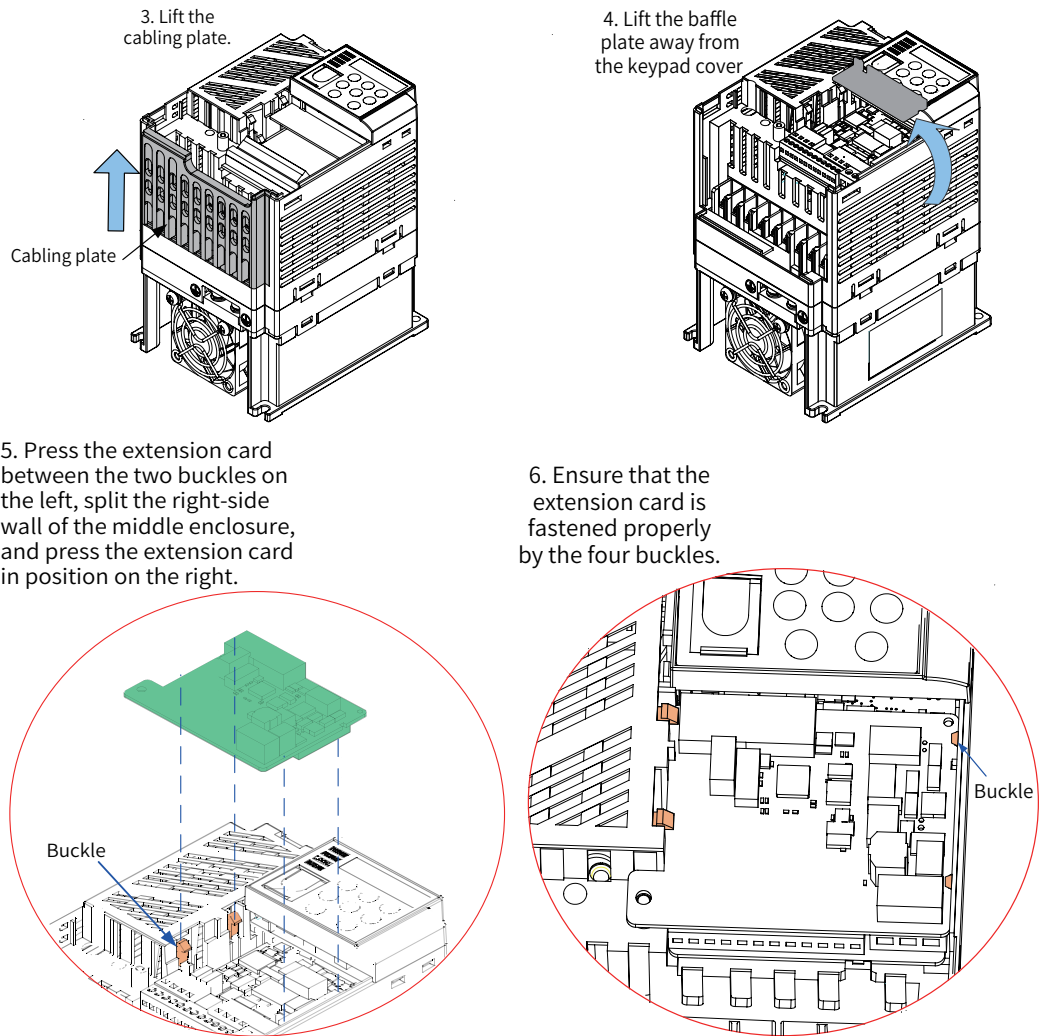
Configuring the MD310

When the MD310 uses CANlink 3.0, the station number and baud rate are set by the DIP switch of the CAN extension card. The S1 and S2 of the MD310-CANL DIP switch form a 10-bit DIP switch used to set the communication baud rate of the CAN bus and the communication device address. Figure 3-83 shows the numbers on the DIP switch. Bd1, 2, and 3 are used to set the baud rate, and Adr1 to 7 are used to set the CANlink address. The ON position of the DIP switch indicates 1, and the lower position indicates 0. The modified baud rate and station number take effect immediately.

ON

Bd1 2 3 Adr1 2   3 4 5 6 7

Baud rate   CANlink address
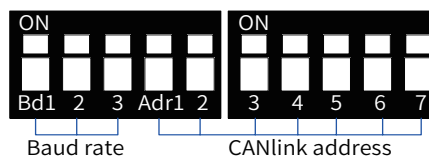
Figure 3-82 MD310-CANL DIP switch

Setting the baud rate

The following table lists the mapping relationships between the DIP switch and the baud rate. Up to eight baud rates can be set.

MD310-CANL baud rates

DIP Switch Bd			Baud rate
1	2	3	
0	0	0	20 kbit/s
0	0	1	50 kbit/s
0	1	0	100 kbit/s
0	1	1	125 kbit/s
1	0	0	250 kbit/s
1	0	1	500 kbit/s
1	1	0	800 kbit/s
1	1	1	1 Mbit/s

CANlink device address

MD310-CANL provides a 7-bit DIP switch for setting the CANlink communication address. Adr1 of the DIP switch indicates the highest bit, and Adr7 indicates the lowest bit. Adr1 to 7 correspond to b6 to b0 of a station number. The valid address setting range of the DIP switch is 1 to 63, as shown in the following table. 0 and 64 to 127 are reserved addresses and cannot be used. The MD310-CANL card is not functional when a reserved address is set.

MD310-CANL DIP switch addresses

DIP Switch Adr							Address
1	2	3	4	5	6	7	
0	0	0	0	0	0	0	Reserved
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	2
0	0	0	0	0	1	1	3
......							...
0	1	1	1	1	1	1	63
1	x	x	x	x	x	x	Reserved

For the usage instructions of the MD310-CANL extension card, see the MD310-CANL the related descriptions of the MD310-CANL expansion card.

4 Connecting the servo to the CANlink 3.0 network

Setting the station number

Modify the station number of the servo through the H0C-00 parameter. The value ranges from 1 to 63. A value beyond the range may result in access failure of CANlink 3.0. The modified value takes effect immediately. Each station number in the network must be unique. If station numbers are repeated, subsequently connected stations may disable communication.

Setting the baud rate

Modify the baud rate through the H0C-08 parameter. See the following table.

Parameter Address	Setting Range	Default Value
H0C-08	0: 20 kbit/s 1: 50 kbit/s 2: 100 kbit/s 3: 125 kbit/s 4: 250 kbit/s 5: 500 kbit/s 6: 800 kbit/s [1] 7: 1000 kbit/s	5

[1] IS620P does not support the 800 kbit/s baud rate. When 6 is selected, the 1000 kbit/s baud rate is used.

The baud rate takes effect immediately after setting. The baud rates of all the stations in the network must be consistent; otherwise, communication may be abnormal.

3.9 PROFIBUS DP Bus

Bus overview

PROFIBUS is an international and open fieldbus standard independent of device manufacturers. PROFIBUS is widely used by automation in the manufacturing, process, building, traffic, and electricity sectors. It becomes an European industrial standard in 1996, became an international standard in 1999, and was approved to be the only fieldbus standard of industrial automation in the People's Republic of China in 2001.

PROFIBUS adopts existing international standards and is based on the Open Systems Interconnection (OSI) model, as shown in Figure 3-83. Therefore, PROFIBUS meets the open and standard requirements. In the OSI models, predefined tasks are implemented accurately during transmission. The physical layer (first layer) defines the physical transmission features. The data link layer (second layer) defines the bus access protocol. The application layer (seventh layer) defines application functions.

PROFIBUS DP uses the first layer, second layer, and user interfaces. The third to seventh layers are not described. This flow structure ensures fast and effective data transmission. The direct data link mapper (DDLM) provides a service user interface for easy access to the second layer. The user interface defines the application functions called by users, the system, and different devices, describes the behaviors of different PROFIBUS DP devices, and provides the RS-485 transmission technique or optical fibers.
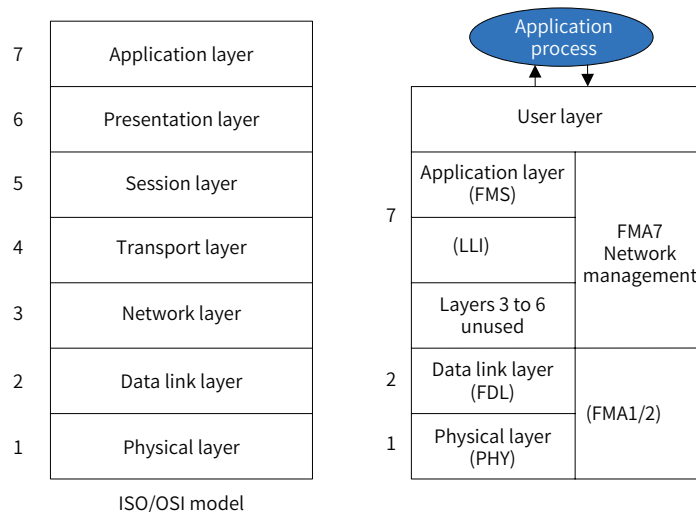
Figure 3-83   PROFIBUS DP fieldbus model

PROFIBUS DP is used for field high-speed data transmission. The master reads the input information of slaves and sends output information to slaves periodically. In addition to periodic transmission of user data, PROFIBUS DP also provides aperiodic communication required by smart field devices for configuration, diagnosis, and alarm handling.

3.9.1 General Process of Using PROFIBUS DP

The general process of using PROFIBUS DP is as follows:

1) Design the hardware network structure of PROFIBUS DP.

2) Activate the PROFIBUS DP bus in network configuration. The PROFIBUS DP master is automatically added after the bus is activated.

3) On the **Network Configuration** tab page, add PROFIBUS DP slaves and modules based on the hardware structure. Before adding a third-party slave, import a GSD file to import the third-party slave on the **Network Configuration** tab page. Before adding an AM600 slave, add an I/O module in hardware configuration. A PROFIBUS DP slave is a remote DP device.

4) Set the master parameters, slave parameters, and module parameters properly. In normal cases, the slave node ID is automatically generated, I/O mapping is automatically generated based on the GSD file, and some special settings need to be modified manually.

When setting the parameters of the master and slave, ensure that the baud rates of the master and slave are adaptive and that the configured slave node ID matches with the DIP switch setting of the actual slave node ID.

3.9.2 PROFIBUS DP Master Configuration

1 Master parameter settings

Figure 3-84 PROFIBUS DP master configuration page

Basic parameters

■ **Station Address**: unique identifier of the master in the PROFIBUS DP network. The default value is **1**, and the value range is **1** to **126**, in the decimal format.

■ **Highest station address**: maximum station address for token transfer. The default value is **126**.

■ **Watch Dog Control**: watch dog time transferred to the slave, used to determine the master-slave connection.

Bus parameters

■ **Baud rate**: baud rate of transmission along the bus. The unit is kbit/s,. The optional values are 9.6, 19.2, 45.45, 93.75, 187.5, 500, 1500, 3000, 6000, and 12000. The default value is 1500.

> **NOTE**
>
> Set the baud rate properly based on different communication distances and the number of communicating stations. The PROFIBUS subnet works properly only with matching parameters of the bus configuration file. You can change the default settings only if you are familiar with the parameter allocation of the PROFIBUS bus configuration file. It is recommended that the default bus parameter settings be used.

2 Stop setting on failure

The Stop Setting on Failure function determines whether to stop slave operation when a slave or module is faulty or the configuration is inconsistent. This function is only applicable to AM600 PROFIBUS DP slaves.

■ **Setting list on slaves failure**: You can view and set whether to stop operation upon slave failure or inconsistent configuration.

In the **Stopped On Failure** column, you can set whether to stop slave operation when the specified slave or module is faulty. If the check box under **Stopped On Failure** is selected, the slave stops running when it is faulty or when the I/O module with the diagnosis and report function enabled is faulty.

■ Click **OK** or **Cancel** to save or cancel the settings on the **Stop Setting on Failure** page.

3 PROFIBUS DP master I/O mapping

For the general description of I/O mapping and instructions on this page, see "I/O mapping."

4 State

The state configuration editor for the PROFIBUS DP bus or modules displays state information (such as Running and Stopped) and the status of the internal bus system.

5 Information

The following basic information about the currently available device is displayed: name, vendor, type, version, module number, and description.

3.9.3 PROFIBUS DP Slave Configuration

To configure a PROFIBUS DP slave, you mainly need to set the basic slave parameters and the slave parameters declared in GSD.

1 Slave parameter setting

Figure 3-85   PROFIBUS DP slave parameter setting

Basic parameters

■ **Station address**: unique identifier of a slave in the PROFIBUS DP network. The value range is 1 to 125, and the default value is 2. It must be a decimal value and consist with the slave identifier (DIP switch setting).

■ **ID number**: unique identifier of the slave, which is determined by the GSD file.

■ **T_SDR**: minimum response interval of the slave, that is, the minimum interval for the slave to return a response after receiving data from the master.

■ **Lock/Unlock**: current status of the slave.

User parameters

User parameters are defined by the GSD file and can be displayed in the decimal or hexadecimal format. For details, see the manual provided by the device vendor.

2 Slave diagnosis

The diagnosis function indicates the running status of slave nodes.

Diagnosis explanation:
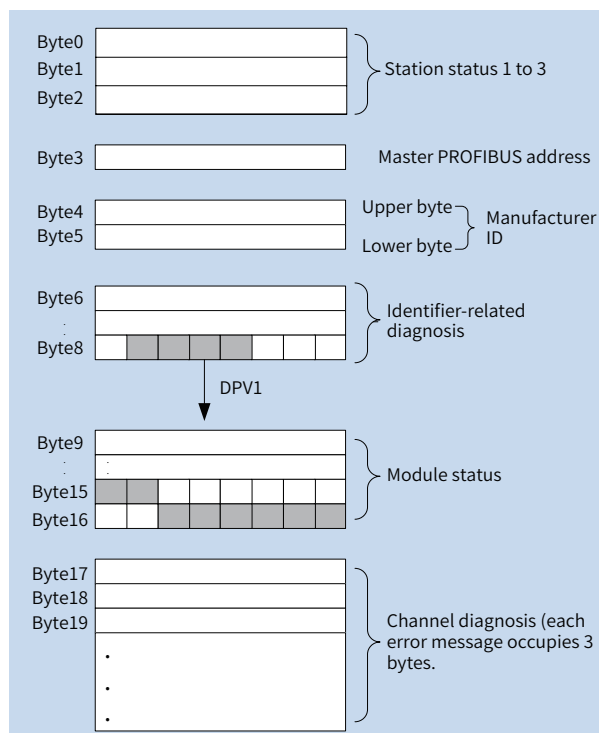
Figure 3-86 Diagnosis packet structure

Table 3-5   Meaning of station status 1

Bit		Definition	Cause and Solution
0	0:	The DP master cannot address the DP slave, and this bit of the DP slave is always 0.	Check whether the PROFIBUS address of the DP slave is correct. Check whether the bus connector and FOC are connected. Check the voltage of the DP slave. Check whether the settings of the RS485 relay are correct. Check whether the DP slave is reset (enabled or disabled).
1	1:	The DP slave is not ready for data exchange.	Wait until the DP slave is started.
2	1:	The configuration data that the DP master sends to the DP slave does not match with the actual configuration of the DP slave.	Check whether the station type entered in the configuration software or the DP slave configuration is correct.
3	1:	External diagnosis is available.	Evaluate the identifier-related diagnosis, module status, and/or channel-related diagnosis. Bit 3 is reset after all the errors are fixed. This bit is reset when a new diagnosis message is generated in the preceding diagnosis bytes.
4	1:	The slave does not support the requested function.	Check the configuration.
5	1:	The DP master fails to parse the response of the DP slave.	Check the bus configuration.
6	1:	The DP slave type does not match with the software configuration.	Check whether the configuration software of the station type is configured correctly.
7	1:	Other DP masters (not the DP master currently accessing the DP slave) have been configured for the DP slave.	This bit is always 1 when the DP slave is accessed through a programming device or other DP masters. The PROFIBUS address of the DP master configured for the DP slave is located in the "master PROFIBUS address" diagnosis byte.

Table 3-6 Meaning of station status 2

Bit		Definition
0	1:	The DP slave must be reconfigured.
1	1:	The slave is in the startup phase.
2	1:	This bit of the DP slave is always 1.
3	1:	Response monitoring is enabled for the DP slave.
4	1:	The DP slave has received the **FREEZE** control command.
5	1:	The DP slave has received the **SYNC** control command.
6	0:	This bit is always 0.
7	1:	Activation of the DP slave is canceled. That is, the DP slave is removed from current processing.

Table 3-7   Meaning of station status 3

Bit		Definition
0 to 6	0:	This bit is always 0.
7	1:	The number of channel-specific diagnosis messages exceeds the number of messages allowed by diagnosis frames.

■ The master PROFIBUS address indicates that the PROFIBUS DP master has been configured for the DP slave and has the read and write permissions on the slave. If the value is FF, the DP master is not configured for the DP slave.

■ The manufacturer ID indicates the DP slave type. It is declared by the device manufacturer and is reflected in GSD.

3.9.4 PROFIBUS DP Module

1 Modular device and non-modular device

In PROFIBUS DP slave configuration, you can connect DP slave nodes to the following two types of devices:

Modular device: It is connected to a DP slave node and provides an I/O mapping list. The **PROFIBUS DP Slave I/O Mapping** dialog box is not required. The data of slave nodes increases as modules are added. Currently, the AM600 I/O module is a type of modular device.

Non-modular device: The slave node dialog box includes the I/O mapping dialog box. Data cannot be configured automatically.

2 AM600 PROFIBUS DP I/O module

Add the AM600 PROFIBUS DP I/O module in hardware configuration, and set related I/O parameters and add I/O mappings to refresh data. For details, see "CPU > I/O module."

3.10 HMI Communication Configuration

3.10.1 Communication Configuration

This drive component reads and writes the data of various registers of Inovance medium-sized PLCs based on the Modbus TCP/IP protocol through the InoTouch Editor software configuration.

The HMI supports the 01, 31, 03, 33, 05, 35, 06, 36, 0F, 3F, 10, and 40 parameters. For details about the parameters, see the HMI user manual.

Drive type	Desciption
Communication protocol	The Modbus TCP/IP protocol of the Inovance AM600 series PLC is used.
Communication mode	One master and one slave; one master and multiple slaves. The drive component is the master, and devices are the slaves.
Communication device	Ethernet subdevices must be attached to the general TCP/IP parent devices to work properly.

1 Hardware connection

Ensure that hardware is connected correctly before configuring communication between the InoTouch Editor software and devices.

Connection method: Use an RJ-45 network cable to directly connect the HMI to the PLC (use a straight-through network cable or hub switch).

2 Device communication parameters

Set the client communication parameters of InoTouch Editor as follows:
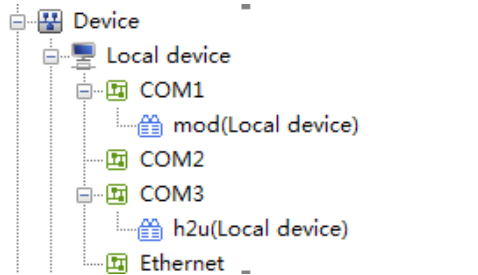
When communication setup is in progress, select **Ethernet** and add a device as follows:
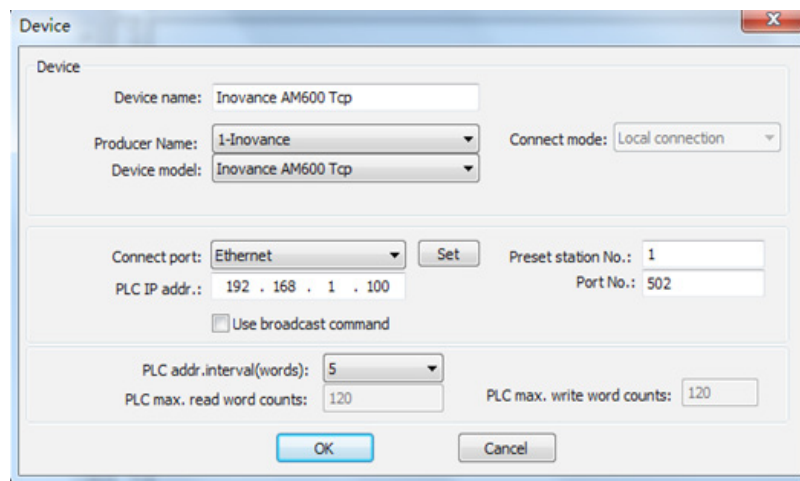
- **PLC IP addr.**: Enter the IP address of the PLC based on the actual setting.

- **Port No.**: port used to send and receive data frames by the host computer and lower computer. The default value is **502**. It is recommended that the default value be used.

Retain the default settings of other parameters.

3 Communication configuration

Set the parameters of the Inovance AM600-ModbusTCP subdevice as follows:
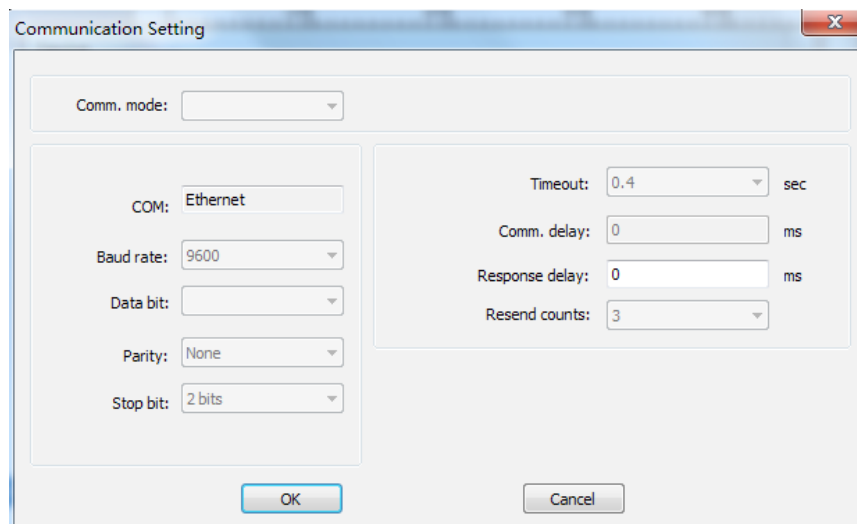
- **Response delay**: interval between frame sending and reception startup. The default value is 0 ms.

4 Collection channel

Communication status

Communication Status Value	Meaning
Communication normal	The current communication is normal.
Command failed	The read and write commands of the device fail to be executed.
Check failed	An error occurred while checking collected data.
Communication timeout	No collected data is returned.

5 Internal attributes

You can add a channel by using internal attributes. This drive component supports the Modbus TCP registers of Inovance AM600 series PLC. The following table lists the parameters.

Register	Data Type	Read Parameter	Write Parameter	Operation	Channel Example
[SM area] Input coil	BT	31	35 and 3F	Read and write	"SM read-only 0000" indicates SM area address 0.
[Q area] output coil	BT	01	05 and 0F	Read and write	"Q read-write 0001" indicates Q area address 1.
[SD area] Input register	16-bit-BCD 32-bit-BCD 16-bit-unsigned 16-bit-signed 32-bit-unsigned 32-bit-unsigned 32-bit-float	33	36 and 40	Read and write	"SD read-only 0002" indicates SD area address 2.
[M area] Output register	16-bit-BCD 32-bit-BCD 16-bit-unsigned 16-bit-signed 32-bit-unsigned 32-bit-unsigned 32-bit-float	03	06 and 10	Read and write	"M read and write 0003" indicates M area address 3.

Parameter: [SD area] uses the 40 parameter when double-word (32-bit) data is written or multiple data records are written in batches.

[M area] uses the 10 parameter when double-word (32-bit) data is written or multiple data records are written in batches.

> **NOTE**
>
> When a channel is added by using internal attributes, the start address is 0 (protocol address), which complies with the Modbus TCP protocol of Inovance AM600.

3.10.2 Communication Example

1 Bit variable read and write

A bit status indicator and a bit status switch are provided.
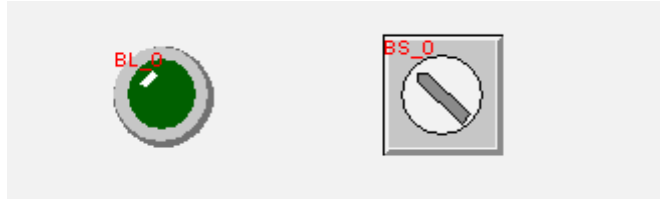
Modify the general attributes of the bit status indicator. Set the address to Q_bit(0:0).
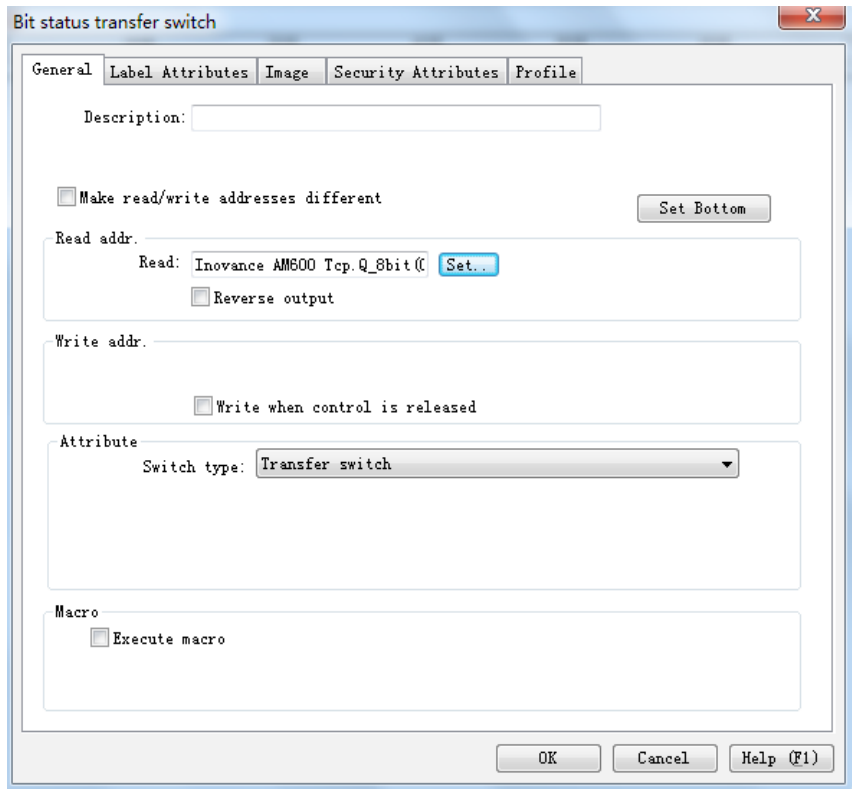
Modify the general attributes of the bit status indicator. Set the address to Q_bit(0:0).

2 Word variable read and write

Select the numeric value input control, as shown in the following figure.
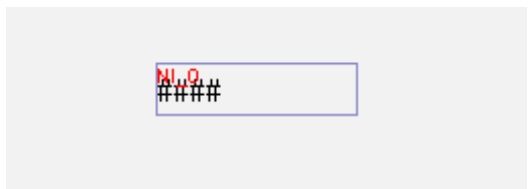
Modify the general attributes of the numeric value input control. Set the address to MW0.
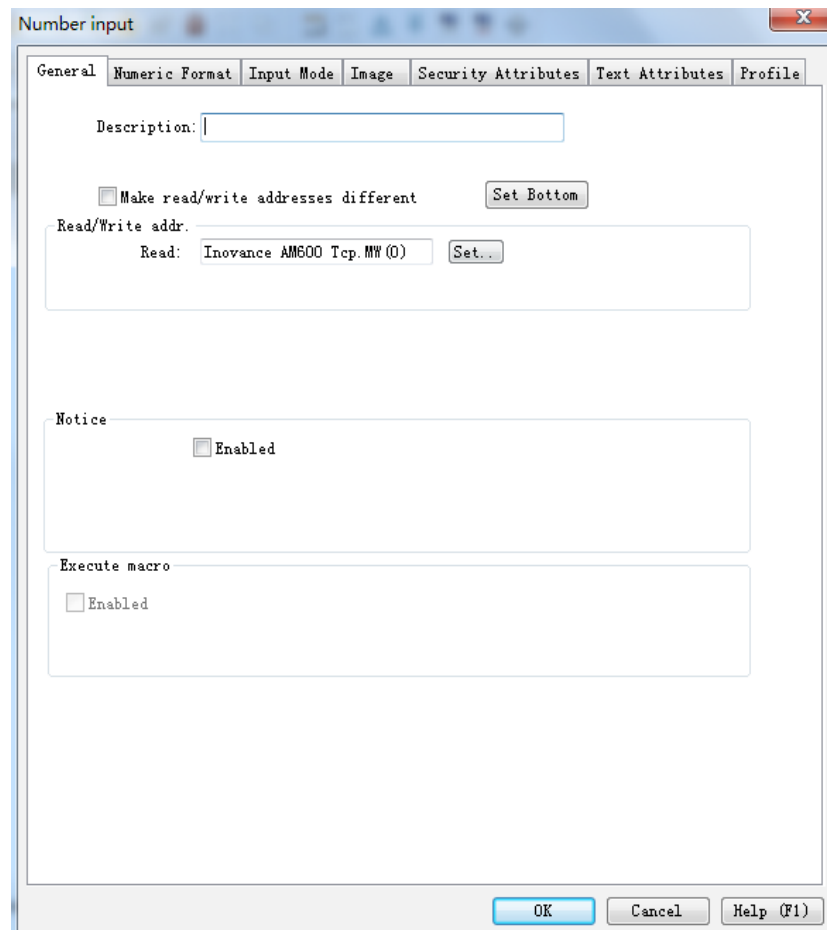
3.10.3 Fault Analysis

Fault Symptom	Analysis	Solution
Communication timeout	An error occurs during collection initialization.	1. Check whether the parameter settings of network devices are correct.
	No collected data is returned. (Incorrect communication hardware connection and parameter settings)	2. Check whether the COM port is occupied by other programs.
		3. Check whether the communication cable is connected correctly.
		4. Check whether the data read address exceeds the specified range.
Command failed	The read and write operations fail.	1. Check whether the data read address exceeds the specified range.
		2. Check whether the communication cable is too long, and perform short-distance testing.
		3. Check whether onsite interferences are excessive, and prevent interferences in the surrounding environment.

1 Registers and parameters supported by the drive component

Register	Read Parameter	Write Parameter	Parameter
[SM area] Input coil	31	35 3F	31: Read the input coil status 35: Enforce a single input coil 3F: Enforce multiple input coils
[Q area] output coil	01	05 0F	01: Read the output coil status 05: Enforce a single output coil 0F: Enforce multiple output coils

Register	Read Parameter	Write Parameter	Parameter
[SD area] Input register	33	36 40	33: Read the input register 36: Preconfigure a single register 40: Preconfigure multiple registers
[M area] Output register	03	06 10	03: Read the holding register 06: Preconfigure a single register 10: Preconfigure multiple registers

Note:

1) This drive component supports the 01, 31, 03, 33, 05, 35, 06, 36, 0F, 3F, 10, and 40 parameters. Other parameters not used by data communication are not supported.

2) The preceding parameters adopt the hexadecimal format. Parameters 0x0F and 0x10 correspond to 15 and 16 in the decimal format.

3) The [SM area] input coil uses the 3F parameter when writing multiple relays in batches.

4) The [Q area] output coil uses the 0F parameter when writing multiple relays in batches.

5) The [SD area] output register uses the 40 parameter when double-word (32-bit) data is written or multiple data records are written in batches.

6) The [M area] output register uses the 10 parameter when double-word (32-bit) data is written or multiple data records are written in batches.

NOTE

When a register channel is added, the start address is 0 (protocol address), which complies with the Modbus protocol of Inovance AM600.

2 Data type table

BTdd	Bit (dd range: 00 to 15)
16-bit-BCD	16-bit BCD
32-bit-BCD	32-bit BCD
16-bit-unsigned	16-bit unsigned
16-bit-signed	16-bit signed
32-bit-unsigned	32-bit unsigned
32-bit-unsigned	32-bit signed
32-bit-float	32-bit floating point number

3 32-bit data storage in registers

1) Mapping relationship between the variable name and register address in the PLC project (16-bit register)

■ Variable %MW5 corresponds to register address 5.

■ Variable %MD5 corresponds to register address 10.

Variable name prefix: W - word (16 bits); D - double word (32 bits).

2) The addresses of drive device commands and channel collection registers start from 0. 0 indicates that the register address is 0.

3) When 32-bit data is read or written, two registers (32 bits) starting from the corresponding register address are occupied.

Chapter 4 Programming Basics

4 Programming Basics

Operands are the objects in user programs related to operators, functions, function blocks, or program operations. They can be used as input, output, and intermediate stored results. The common operands of CoDeSys include direct addresses, constants, and variables.

Similar to other advanced languages, CoDeSys also provides constants and variables. Constants are unchanged numeric values. Variables are user-defined identifiers. Variables are stored in user-specified addresses of the %I, %Q, and %M areas. If addresses are not specified, variables are stored in system-allocated addresses. You do not need to concern the variable storage location.

4.1 Direct Address

A direct address, also called fixed address or direct variable, has a direct mapping to a specific address of the PLC. The address information includes the variable storage location in the CPU, storage size, and storage position offset.

4.1.1 Syntax

Syntax: %<store area prefix><size prefix><number>|.<number>

■ The programming system supports the following three storage area prefixes:

1) I: input, physical input, sensor

2) Q: output, physical output, executor

3) M: storage location

■ The programming system supports the following size prefixes:

1) X: bit, 1 bit

2) B: byte, 1 byte

3) W: word, 1 word

4) D: double word, 4 bytes (double byte)

■ The first number indicates the offset address of the memory prefix corresponding to the variable. The number following "." indicates the specific bit after the address offset when the variable is of the boolean type.

Example:

%QX7.5 output area with 7-byte offset, sixth bit (bit 5)

%QX17 output area with 17-byte offset

%IW215 input area with 215-word offset

%MD48 memory area with 48-double word offset

iVar AT %IW10: WORD;//The variable iVar is of the word type and maps to the location with 10-word offset in the input area.
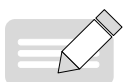
NOTE

◆ When a variable with the X-type size prefix indicates the boolean data type, the offset address needs to be precise to bits.
◆ The size prefix matches with the data type. A variable with the B-type size prefix must be declared as a 1-byte data type, such as BYTE, SINT, and USINT. A variable with the W-type size prefix must be declared as a 1-word data type, such as WORD, INT, and UINT. A variable with the D-type size prefix must be declared as a double-word data type, such as DWORD, DINT, and UDINT.

4.1.2 PLC Direct Address Storage Area

The direct address storage area varies depending on different PLCs. PLC data is not retained upon power failure for the %I and %Q areas, but is retained for the %M area.

The AM600, AM610, AM401, and AM402 programming systems provide the 128 KB (byte) input area (I area), 128 KB (byte) output area (Q area), and 512 KB storage area (M area). The first 480 KB of the storage area can be used directly, whereas the last 32 KB are used by the system, mainly as soft elements, and cannot be used directly by users. During programming, you can directly access addresses, or define variables and map the variables to addresses for indirect access. The following table lists the storage areas and address ranges.

Area	Function	Size	Address Range
I area (%I), 128 KB	Area used by users	64 Kwords	%IW0 to %IW65535
Q area (%Q), 128 KB	Area used by users	64 Kwords	%QW0 to %QW65535
M area (%M), 512 KB	Area used by users	240 Kwords	%MW0 to %MW245759
	SD element	10,000 words	%MW245760 to %MW255759
	SM element	10,000 bytes	%MB511520 to %MB521519
	Reserved	2768 bytes	%MB521520 to %MB524287

The AC800 and AC811 programming systems provide the 128 KB (byte) input area (I area), 128 KB (byte) output area (Q area), and 5 MB storage area (M area). The AC800 and AC811 series do not support soft elements. The %M area address can be used as needed. The following table lists the storage areas and address ranges.

Area	Function	Size	Address Range
I area (%I), 128 KB	Area used by users	64 Kwords	%IW0 to %IW65535
Q area (%Q), 128 KB	Area used by users	64 Kwords	%QW0 to %QW65535
M area (%M), 5MB	Area used by users	2.5 Mwords	%MW0 to %MW2321439

4.2 Variable

Variables can be defined by the POU, automatic declaration dialog box, and the DUT or GVL editor. Variable types are identified through variable type keywords. For example, VAR and END_VAR identify local variables.

Variable types include local variable (VAR), input variable (VAR_INPUT), output variable (VAR_OUTPUT), I/O variable (VAR_IN_OUT), global variable (VAR_GLOBAL), temporary variable (VAR_TEMP), static variable (VAR_STAT), and configuration variable (VAR_CONFIG).

4.2.1 Variable Definition

Variables can be edited in the declaration editor. The declaration editor is displayed in the text view or table view. Figure 4-1 shows the declaration editor of POU in the text view.

1 -Switch between the text view and table view

Figure 4-1   Declaration editor in the text view

Figure 4-2 shows the declaration editor in the table view.
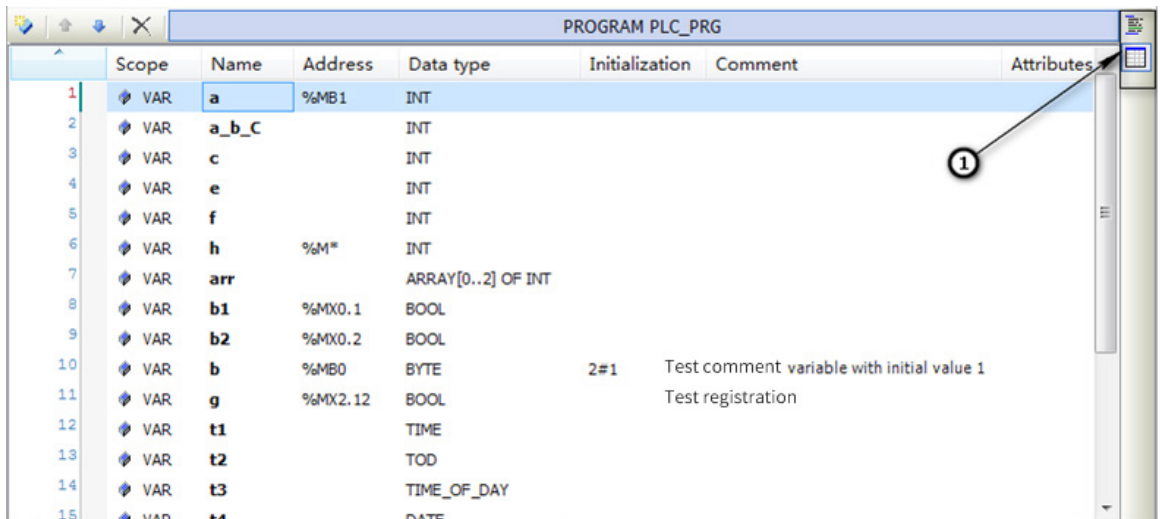
1 -Switch between the text view and table view

Figure 4-2 Declaration editor in the table view

Syntax: <identifier> {AT <address>}:<data type> {:=<initial value>};

The parts enclosed by braces ({}) are optional.

1 Identifier

An identifier is the name of a variable. The variable naming conventions are as follows:

1) The name cannot contain spaces or special characters.

2) The name cannot contain predefined keywords.

3) The name is case-insensitive.

4) The name length is unlimited.

5) The name cannot be defined repeatedly.

A local variable name can be the same as a global variable name. By default, the local variable is used and can indicate a global variable. A specific variable can also be indicated by a full path variable name. Example: local variable iVar: = 1; global variable .iVar: = 2; full path variable globlist1.iVar: = 3;

Consider naming suggestions when you name a variable. For example, a variable name must accurately indicate the meaning and data type of the variable, and the Hungarian naming method

(variable name = attribute + type + object description) is recommended.

2 AT address

An AT address is a direct address. For details, see section "5.1 Programming Languages Supported by InoProShop".

3 Data type

Data types are classified into standard data type and user-defined data type.

1)   Standard data type

Standard data types are classified into boolean, integer, floating point, string, and time.

Type	Keyword	Range	Memory Usage
Boolean	BOOL	TRUE, FALSE, 0, and 1	8 bits
Bit type	BIT	TRUE, FALSE, 0, and 1, only used in structures or function blocks	1 bit
Integer	BYTE	0 to 255	8 bits
	WORD	0 to 65535	16 bits
	DWORD	0 to 4294967295	32 bits
	LWORD	0 to ($2^{64} - 1$)	64 bits
	SINT	-128 to +127	8 bits
	USINT	0 to 255	8 bits
	INT	-32768 to +32767	16 bits
	UINT	0 to 65535	16 bits
	DINT	-2147483648 to +2147483647	32 bits
	UDINT	0 to 4294967295	32 bits
	LINT	-2^{63} to ($2^{63} - 1$)	64 bits
	ULINT	0 to ($2^{64} - 1$)	64 bits
Floating point	REAL	(1.401e – 45) to (3.403e + 38)	32 bits
	LREAL	(2.2250738585072014e – 308) to (1.7976931348623158e + 308)	64 bits
String	STRING	Only ASCII characters are supported. Chinese characters are not supported. By default, the maximum length is 80 characters. The part exceeding the maximum length is truncated. The maximum character length can be declared, for example, str:STRING(35):='This is a String'. A string function supports up to 255 characters.	Strings are stored in the ASCII format. The terminator is stored as 1 byte.
	WSTRING	Only Unicode characters (including Chinese characters) are supported. By default, the maximum length is 80 characters. The part exceeding the maximum length is truncated. The maximum character length can be declared, for example, wstr:WSTRING(35):="This is a WString";.	Strings are stored in the Unicode format. The terminator is stored as 2 bytes.
Time	TIME		
	TIME_OF_DAY(TOD)	Time range within one day	
	DATE	Starting from January 1, 1970	
	DATE_ADN_TIME(DT)	Starting from January 1, 1970	

2)   User-defined data type

User-defined data types include array, structure, enumeration, association, alias, subset, reference, and pointer. In AM600 programming software InoProShop, right-click an application and choose **Add Object** > **DUT** from the context menu to add the following four user-defined data types: structure, enumeration, association, and alias.

■   Array

Syntax: <Array_Name>:ARRAY [<ll1>..<ul1>,<ll2>..<ul2>,<ll3>..<ul3>] OF <elem. Type>

ll1, ll2, and ll3 define the lower limit of the area, whereas ul1, ul2, and ul3 define the upper limit. The numeric value must be an integer. elem. Type indicates the data type of each array element.

Initialization and example

```
Card_game: ARRAY [1..13, 1..4] OF INT;
```

```
arr1 : ARRAY [1..5] OF INT := [1,2,3,4,5];
```

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

```
arr3 : ARRAY [1..2,2..3,3..4] OF INT := [2(0),4(4),2,3];    (*array value
0,0,4,4,4,4,2,3*)
```

```
arr1 : ARRAY [1..10] OF INT := [1,2]; (*Array initialization. Uninitialized elements adopt the
default value 0.*)
```

Example of array structure initialization

Structure definition:

```
TYPE STRUCT1
```

```
STRUCT
```

```
 p1:int;
```

```
 p2:int;
```

```
 p3:dword;
```

```
END_STRUCT
```

```
END_TYPE
```

Array structure initialization:

```
arr1:ARRAY[1..3] OF STRUCT1:= [(p1:=1,p2:=10,p3:=4723),(p1:=2,p2:=0,p3:=299),(p1
:=14,p2:=5,p3:=112)];
```

Syntax of access association elements:

```
<Array-Name>[Index1,Index2].
```

Example:

```
Card_game [9,2]
```

■   Structure

Syntax:

```
TYPE <structurename> | EXTENDS DUTTYPE:
```

```
STRUCT
```

```
    <declaration of variables 1>
```

```
    ...

    <declaration of variables n>

END_STRUCT

END_TYPE
```

<structurename> is a type and can be used as a data type. EXTENDS DUTTYPE is optional and indicates inheritance from the members of DUTTYPE. Variables of the structurename type can be used to access the members of DUTTYPE. DUTTYPE is of the structure, association, or alias type.

Initialization and example

Polygonline structure definition:

```
TYPE Polygonline:

STRUCT

    Start:ARRAY [1..2] OF INT;

    Point1:ARRAY [1..2] OF INT;

    Point2:ARRAY [1..2] OF INT;

    Point3:ARRAY [1..2] OF INT;

    Point4:ARRAY [1..2] OF INT;

    End:ARRAY [1..2] OF INT;

END_STRUCT

END_TYPE
```

Initialization:

```
Poly_1:polygonline := ( Start:=[3,3], Point1:=[5,2], Point2:=[7,3],
Point3:=[8,5], Point4:=[5,7], End:= [3,5]);
```

Syntax of access structure elements:

```
<structurename>.<variable>
```

Example:

Poly_1.Start

■   Enumeration

An enumerated value consists of several constants.

Syntax:

```
TYPE <identifier>:(<enum_0> ,<enum_1>, ...,<enum_n>) |<base data type>;

END_TYPE
```

identifier: user-defined enumeration type; enum_n: constant value of the enumeration type. Each constant can declare its value. If no value is declared, the default value is used. The data type of enumerated constants is base data type. The value may not be declared and is an integer by default.

Initialization and example

```
TYPE TRAFFIC_SIGNAL: (red, yellow, green:=10);  (* red indicates the initial value 0,
yellow indicates the initial value 1, and green indicates the initial value 10.  *)
```

```
END_TYPE

TRAFFIC_SIGNAL1 : TRAFFIC_SIGNAL;

TRAFFIC_SIGNAL1:=0; (* The value of the enumerated variable is red. *)

FOR i:= red TO green DO

 i := i + 1;

END_FOR;
```

■   Association

Syntax:

```
TYPE <unionname>:UNION

    <declaration of variables 1>

    ...

    <declaration of variables n>

END_UNION

END_TYPE
```

< unionname > is a type and can be used as a data type. All variables of the association type share the same storage location and are allocated with space the same as that of the variable that occupies the largest space.

Example

```
TYPE union1: UNION

a : LREAL;

b : LINT;

END_UNION

END_TYPE
```

Syntax of access array elements:

```
< unionname >.<variable>
```

Example

```
union1.a
```

■   Alias

A data type can be expressed by an alias.

Syntax:

```
TYPE <aliasname>:basetype END_TYPE
```

aliasname indicates the alias type and is used as a data type. basetype is a standard or user-defined data type.

Example

```
TYPE alias1 : ARRAY[0..200] of byte; END_TYPE
```

The initialization and access mode are consistent with the basic type.

■ Subset

The subset data type is a subset of the defined basic data type. A subset type can be added by adding a DUT. A variable can be directly declared as a subset type.

Syntax of DUT objects:

```
TYPE <name> : <Inttype> (<ug>..<og>) END_TYPE;
```

name: valid IEC identifier.

Inttype: a data type, such as SINT, USINT, INT, UINT, DINT, UDINT, BYTE, WORD, and DWORD (LINT, ULINT, and LWORD).

ug: a constant, which must be compatible with the basic type and sets the lower boundary of the range types. The lower boundary itself is included in this range.

og: a constant, which must be compatible with the basic type and sets the upper boundary of the range types. The upper boundary itself is included in this basic type.

Example of DUT object declaration

```
TYPE

SubInt : INT (-4095..4095);

END_TYPE
```

Example of direct variable declaration

```
VAR

 i : INT (-4095..4095);

 ui : UINT (0..10000);

END_VAR
```

■ Reference

Reference is the alias of an object. Operating references is equivalent to operating objects.

Syntax:

```
<identifier> : REFERENCE TO <data type>
```

identifier: reference identifier. data type: data type of the referenced object.

Example and initialization

```
ref_int : REFERENCE TO INT;

a : INT;

b : INT;

ref_int REF= a;    (* ref_int references a. *)

ref_int := 12;     (* a is set to 12. *)

b := ref_int * 2; (* b is set to 24. *)

ref_int REF= b;    (* ref_int references b. *)

ref_int := a / 2; (* b is set to 6. *)
```
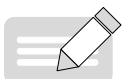
NOTE   The bit type cannot be referenced. That is, ref1:REFERENCE TO BIT. cannot be defined.

■ Pointer

A pointer stores the address of an object and can point to any data type (except the bit type).

Syntax:

```
<identifier>: POINTER TO <data type>;
```

identifier: pointer identifier. data type: data type pointed to by a pointer.

Pointers are operated by using address operators. Address operators include ADR (variable address acquisition) and ^ (value of a variable address).

Example and initialization

```
VAR

    pt:POINTER TO INT;   (* Declares the pointer pt of the INT type.*)

    var_int1:INT := 5;

    var_int2:INT;

END_VAR

pt := ADR(var_int1); (* Allocates the address of the varint1 variable to the pointer pt. *)

var_int2:= pt^;         (* Uses the ^ address operator to obtain the value of the pointer.*)

pt^:=33;    (*Assigns a value to the var_int1 variable corresponding to the pointer.*)
```

4 Initial value

By default, the initial value of a variable is 0. You can add user-defined initial values by using the valuation operator ":=" during variable declaration. An initial value is a valid ST expression. An ST expression consists of operators, operands, and a valuation expression. Operators mainly include addition (+), subtraction (–), multiplication (*), and division (/). Operands mainly include constants, variables, and functions. A valuation expression is the operator in the ST expression used to assign values to variables. Therefore, constants, variables, or functions can be initialized. Ensure that the used variables have been initialized.

Example:

```
VAR

var1:INT := 12;   (* The initial value of the integer variable is 12.*)

x : INT := 13 + 8;    (*Defines the initial value of the constant expression.*)

y : INT := x + fun(4);   (*Includes function call in the initial value.*)

z : POINTER TO INT := ADR(y);   (*Initializes the pointer by using the address function ADR.*)

END_VAR
```

NOTE
◆ The global variable table (GVL) is initialized before POU local variables are defined.
◆ If the default value is modified online, the pointer is not initialized during definition and still points to the variable before online modification.

4.2.2 Variable Type

The main variable types include local variable (VAR), input variable (VAR_INPUT), output variable (VAR_OUT), I/O variable (VAR_IN_OUT), global variable (VAR_GLOBAL), temporary variable (VAR_TEMP), static variable (VAR_STAT), and configuration variable (VAR_CONFIG).

Declaration syntax of the variable type: `<type_key> |atribute_key`

`variable1;`

`variable2;`

`...`

`END_VAR`

type_key: type keyword, which may be VAR (local variable), VAR_INPUT (input variable), VAR_OUTPUT (output variable), VAR_IN_OUT (I/O variable), VAR_GLOBAL (global variable), VAR_TEMP (temporary variable), VAR_STAT (static variable), and VAR_CONFIG (configuration variable).

atribute_key: attribute keyword, which may be RETAIN, PERSISTENT, or CONSTANT. It defines the range of a variable. The following details attribute keywords.

■ RETAIN variable (reserved)

The RETAIN variable retains the original value when the PLC restarts upon power failure or is reset in hot mode. RETAIN is stored in a specified RETAIN storage area. An example is the counter of a production machine. The counter starts counting again from the breakpoint upon power failure.

Example

Define the RETAIN variable in programs:

`PROGRAM PLC_PRG`

`VAR RETAIN`

`  iRem1 : INT;`

`END_VAR`

Define the RETAIN variable in the global variable table:

`VAR_GLOBAL RETAIN`

`  gvarRem1 : INT;`

`END_VAR`
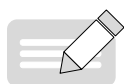
NOTE	◆ If the RETAIN variable is declared in programs, only this variable is stored in the RETAIN storage area.
	◆ If the RETAIN variable is declared in function blocks, the entire function sample data is stored in the RETAIN storage area, but only the RETAIN variable is processed as a reserved variable.
	◆ If the RETAIN variable is declared in functions, the declaration does not take effect.

■ PERSISTENT variable

The definition of VAR PERSISTENT is always the same as that of VAR PERSISTENT RETAIN or VAR RETAIN PERSISTENT. That is, the PERSISTENT variable supports value retention upon cold reset and program download, in addition to the features (value retention upon power failure and hot reset) of the RETAIN variable. The PERSISTENT variable is initialized only when the initial value is reset. A program runtime counter is a common PERSISTENT variable. The counter continues counting when the power supply fails or the program is redownloaded.

Example

Example of the PERSISTENT variable table:

`VAR_GLOBAL PERSISTENT RETAIN`

`  iVarPers1 : DINT;  bVarPers : BOOL;`

//Right-click the PERSISTENT variable table editor to add the path of a PERSISTENT variable instance.

```
PLC_PRG.PERS: INT; (* Defines the PERSISTENT variable named PERS in the PLC_PRG
program. *)

END_VAR
```

NOTE

◆ An application has only one PERSISTENT variable table. The only method to add a PERSISTENT variable table is to right-click an application and choose **Add Object** > **Persistent Variable** from the context menu.

◆ You can add a PERSISTENT variable by using the PERSISTENT attribute in a program. In the PERSISTENT variable editor, right-click and choose **Add all instance paths** from the context menu to add the PERSISTENT variables in all the programs to the PERSISTENT variable table.

The following table lists the situations where a variable retains the original value or is initialized in the cases of reset and power failure.

x = retain the original value - = initialize the value

Action	VAR	VAR RETAIN	VAR PERSITENT or VAR PERSITENT RETAIN or VAR RETAIN PERSITENT
Power failure	-	x	x
Hot reset	-	x	x
Cold reset	-	-	x
Initial value reset	-	-	-
Program download	-	-	x
Online modification	x	x	x

Note:

1) The RETAIN variable and PERSISTENT variable are reserved variables and are stored in the same reserved variable area of the programming system.

2) The direct variables mapped to the %M address can be declared as reserved variables, whereas the direct variables mapped to %I and %Q cannot be declared as reserved variables. (Reserved variables cannot be declared as direct variables during automatic declaration. Therefore, %M direct variables only support manual input.)

3) The specific reserved variable area of the programming system is 512 KB in size. This area does not contain the reserved variables mapped to the %M address (the available size of the %M address is 480 KB and can be used by reserved variables). That is, the maximum size of available reserved variables is 992 KB (512 KB + 480 KB).

■ CONSTANT

For details about constant declaration, see section "5.2 Structured Text (ST)".

1 Local variable (VAR)

The variables between VAR and END_VAR within POU are local variables and cannot be accessed externally.

Valuation format:

Local variable:=Value

Example

```
VAR
  iLoc1:INT; (* Local variable*)
END_VAR
```

2 Input variable (VAR_INPUT)

The variables between VAR_INPUT and END_VAR within POU are input variables and assigned values in the call location.

POU call format:

Local variable:=Value input by the caller

Example

```
VAR_INPUT
  iIn1:INT; (* Input variable*)
END_VAR
```

NOTE

Input variables can be modified within POU, even when the CONSTANT attribute is added.

3 Output variable (VAR_OUTPUT)

The variables between VAR_OUTPUT and END_VAR within POU are output variables. Output variables can be returned to the caller during the call process for further processing.

POU call format:

Output variable=>variable of the caller-matched type

Example

```
VAR_INPUT
  iOut1:INT; (* Output variable*)
END_VAR
```
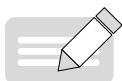
NOTE

◆ For FUNCTION and METHOD, return values and output variables are supported, but a caller needs to be allocated for receiving the variables during the call process. Example: fun(iIn1 := 1, iIn2 := 2, iOut1 => iLoc1, iOut2 => iLoc2);
◆ The output variables of function blocks can be assigned to the caller after the call process.

4 I/O variable (VAR_IN_OUT)

The variables between VAR_IN_OUT and END_VAR within POU are I/O variables. I/O variables can be transferred to the called POU and modified within the called POU. In the actual situation, the variables transferred to the called POU are referenced by the caller.

Example

```
VAR_IN_OUT
  iInOut1:INT; (* I/O variable*)
END_VAR
```

◆ Because the variables transferred to the called POU are referenced by the caller, the I/O variables in function block instances cannot be accessed directly. That is, <FBinstance>.<InOutVariable> cannot be used directly. The reason is that input variables are referenced by the caller and have been changed.

◆ I/O variables cannot be constants or direct variables of the bit type, such as xBit0 AT %I2.0:BOOL. Add the CONSTANT attribute (VAR_IN_OUT CONSTANT) to declare I/O variables. To use direct variables of the bit type, you need to add an intermediate variable as an I/O variable and assign the value of the intermediate variable to the direct variable of the bit type.

NOTE

Example of a direct variable of the bit type:

```
VAR_GLOBAL
    xBit0 AT %MX0.1 : BOOL;(*Declare a direct variable of the bit type.*)
    xTemp : BOOL; (*Intermediate variable*)
END_VAR
```

//Function block with an I/O variable (xInOut)

```
FUNCTION_BLOCK FB_Test
VAR_INPUT
    xIn    : BOOL;
END_VAR
VAR_IN_OUT
    xInOut : BOOL;
END_VAR
IF xIn THEN
    xInOut := TRUE;
END_IF
```

//Calls the function block in the program.

```
PROGRAM Main
VAR
    xIn : BOOL;
    I1  : FB_Test;
    I2  : FB_Test;
END_VAR
```

//A compiling error is returned when a direct address variable of the bit type is used.

```
//I1(xIn:=xIn, xInOut:=xBit0);
```

//Uses the intermediate variable xTemp to transfer the value of xBit0 to the function block and assigns the value of the intermediate variable to xBit0.

```
xTemp := xBit0;
I2(xIn:=xIn, xInOut:=xTemp);
xBit0 := xTemp;
```

I/O constants (VAR_IN_OUT CONSTANT) are read-only. Input variables can be modified in the current version, even when the constant attribute is added. Therefore, the variable attribute can be changed to non-modifiable by using an I/O constant.

I/O constant example:

```
PROGRAM PLC_PRG

VAR

 sVarFits : STRING(16);

 sValFits : STRING(16) := '1234567890123456';

 iVar: DWORD;

END_VAR

POU(sReadWrite:='1234567890123456', scReadOnly:='1234567890123456', iVar-
ReadWrite:=iVar);

//POU(sReadWrite:=sVarFits, scReadOnly:=sVarFits, iVarReadWrite:=iVar);

//POU(sReadWrite:=sValFits, scReadOnly:=sValFits, iVarReadWrite:=iVar);

//POU(sReadWrite:=sVarFits, scReadOnly:='23', iVarReadWrite:=iVar);

FUNCTION POU : BOOL

VAR_IN_OUT

 sReadWrite  : STRING(16);      (*  The string can be read and written within the POU.  *)

 iVarReadWrite : DWORD;   (*The word and variable can be read and written within the POU.*)

END_VAR

VAR_IN_OUT CONSTANT

 scReadOnly : STRING(16);    (*The string is read-only within the POU.*)

END_VAR

sReadWrite := 'string_from_POU';

iVarInPOU := STRING_TO_DWORD(scReadOnly);
```

5 Global variable (VAR_GLOBAL)

The variables between VAR_GLOBAL and END_VAR are global variables. Common variables, constants, and reserved variables can be declared as global variables. In the AM600 programming software InoProShop, right-click an application and choose **Add Object** > **Add Global Variable** from the context menu to add a global variable table, and add global variables to the table.

Example

```
VAR_GLOBAL

 iGlobVar1:INT; (*  Global variable*)

END_VAR
```
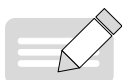
> **NOTE**
> ◆ If a local variable has the same name as a global variable, the local variable is operated when an operation is performed on the variable name. You can add the global range operator (.) before the variable name to operate the global variable, such as . iGlobVar1.
> ◆ Global variables are always initialized before local variables.

6 Temporary variable (VAR_TEMP)

The variables between VAR_TEMP and END_VAR are temporary variables, which are initialized when being called.

Example

```
VAR_TEMP

 iTemp1:INT; (*Temporary variable*)

END_VAR
```

> **NOTE**
> ◆ Temporary variables are declared only in programs and function blocks.
> ◆ Temporary variables are used only in declared programs or function blocks.

7 Static variable (VAR_STAT)

The variables between VAR_STAT and END_VAR are static variables. Static variables are initialized when being called for the first time. The variable values are returned after the POU is called.

Example

```
VAR_STAT

 iStat1:INT; (*Static variable*)

END_VAR
```

> **NOTE**
> ◆ Static variables are declared only in function blocks, functions, and methods, but cannot be declared in programs.
> ◆ Static variables are used only in the declared POU.

8 Configuration variable (VAR_CONFIG)

The variables between VAR_CONFIG and END_VAR are configuration variables. Configuration variables are direct variables that are mapped to the direct variables with indefinite addresses in function blocks. A variable with an indefinite address can be defined in a function block. The indefinite address (arbitrary address) is indicated by "*". Add a configuration variable table (by adding a global variable table) to add the variables with indefinite addresses in all the function block instances to the configuration variable table, which defines all the indefinite addresses. This allows you to manage the variables with indefinite addresses in all the function blocks.

Syntax for variables with indefinite addresses in function blocks:

<identifier> AT %<I|Q|M>* : <data type>

Addresses are finally defined in the variable configuration of the global variable list.

Example

```
FUNCTION_BLOCK locio

VAR

 xLocIn AT %I*: BOOL := TRUE;

 xLocOut AT %Q*: BOOL;

END_VAR
```

Two I/O variables, a local input variable (%I*), and a local output variable (%Q*) are defined.

A global variable list (GVL) is added. The specific addresses declared by instance variables are entered between the keywords VAR_CONFIG and END_VAR. The instance variables include the complete instance path of the POU. The specific addresses correspond to indefinite addresses (%I*, %Q*) in function blocks. The data type must be consistent with that declared by function blocks.

Syntax of configuration variables:

<instance variable path> AT %<I|Q|M><location> : <data type>;.

Example

```
PROGRAM PLC_PRG

VAR

locioVar1: locio;

locioVar2: locio;

END_VAR

VAR_CONFIG (*Correct variable configuration table*)

PLC_PRG.locioVar1.xLocIn AT %IX1.0 : BOOL;

PLC_PRG.locioVar1.xLocOut AT %QX0.0 : BOOL;

PLC_PRG.locioVar2.xLocIn AT %IX1.0 : BOOL;

PLC_PRG.locioVar2.xLocOut AT %QX0.3 : BOOL;

END_VAR
```
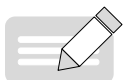
NOTE

◆ Configuration variables are not required in normal cases. The reason is that for I/O address input and output, variables can be mapped to I/O addresses by using an input assistant or directly entering the instance variable path on the I/O mapping page of the corresponding module.
◆ Configuration variables are mapped to variables with indefinite addresses in function blocks or programs.
◆ A compiling error is returned when only variables with indefinite addresses or configuration variables exist. The two types of variables must be used in combination.

4.3 Constant

In PLC programming, constants are parameters with unchanged values, such as timer time and conversion ratio.

Constant declaration syntax:

```
VAR CONSTANT

<identifier>:<type> := <initialization>;

END_VAR
```

Example

```
VAR CONSTANT

c_iCon1:INT:=12;

END_VAR
```

CoDeSys supports constants of multiple data types, such as boolean, integer, time, and string. The following table lists specific constants.

Type	Description	Example
Boolean	The optional values are TRUE and FALSE (or 1 and 0). 1 indicates TRUE, and 0 indicates FALSE.	TRUE, FALSE, and 1
Bit type	Similar to the boolean type, the bit type is used only in structures (number of occupied bits) or function blocks (direct addresses of the boolean type).	TRUE, FALSE, and 0

Type	Description	Example
Integer	Integer constants support values in the binary, decimal, octal, and hexadecimal formats. If an integer value is not in the decimal format, add the format number and the # symbol before the value. 10 to 15 in the decimal format are indicated by A to F in the hexadecimal format.	Decimal format: 66 Binary format: 2#101 Octal format: 8#72 Hexadecimal format: 16#3A Type constants: INT#22 BYTE#204
Floating point	Floating point constants are expressed by decimal numbers and exponents in scientific notation.	7.4 2.3e+9 REAL#3.12
ASCII string	An ASCII string constant is located between two single quotation marks and can include spaces and special characters. A character is expressed by a byte. Only ASCII characters are supported. Chinese characters are not supported. By default, the maximum length is 80 characters. The part exceeding the maximum length is truncated. The maximum character length can be declared, for example, str:STRING(35):=' This is a String' . A string function supports up to 255 characters.	Example of $ used as an escape character: '$30': 0, character 0, ASCII character corresponding to 30 in the hexadecimal format $$: $, US dollar character $' : ' , single quotation mark
Unicode string	A Unicode string constant is located between two double quotation marks. A character occupies two bytes. Only Unicode characters (including Chinese characters) are supported. By default, the maximum length is 80 characters. The part exceeding the maximum length is truncated. The maximum character length can be declared, for example, wstr:WSTRING(35):="This is a WString";.	"Unicode string"
Time	Time constants are generally used by time-related operations and consist of "T#" (or "t#") and a time value, in the units of days (d), hours (h), minutes (m), seconds (s), and milliseconds (ms).	t#12h34m15s;
Time	Time range within a day; syntax: TOD#time value.	TOD#15:36:30.123
Date	Starting from January 1, 1970; syntax: d#date.	d#2015-02-12
Date and time	Date constants and time constants are collectively called date and time constant, which starts from January 1, 1970; syntax: dt#date.	dt#2004-03-29-11:00:00

NOTE

Constants not of the boolean, bit, and string types are indicated in the format "keyword#constant value".

Chapter 5 Programming Language

5 Programming Languages

5.1 Programming Languages Supported by InoProShop

The programming software supports the following six PLC programming languages:

- Ladder diagram (LD)

- Function block diagram (FBD)

- Instruction list (IL)

- Structured text (ST)

- Sequential function chart (SFC)

- Continuous function chart (CFC)

The LD, FBD, ST, IL, and SFC are based on the IEC 61131-3 standard, and CFC is an extension of the IEC 61131-3 standard.

The basic edit methods on the programming interface are applicable regardless of the selected language, which greatly facilitates programming.

- ■ Common functions of the Windows text editor are supported, such as the copy **(Ctrl**+**C**), paste (**Ctrl**+**V**), and delete (**Del**) shortcut keys.

- ■ The Windows standard **Ctrl** and **Shift** keys can be used to select multiple options.

- ■ The function key **F2** can be used to start the input assistant. The system provides input tips or options based on the specific environment.

> **NOTE** By default, the FBD and IL are not supported for the moment. For details about the SFC and CFC, see the help document.

5.2 Structured Text (ST)

The ST is a text-based advanced language and similar to PASCAL or C. The program code consists of instructions comprising keywords and expressions. Different from the IL, the ST can include multiple statements during a statement cycle, allowing for complex structure development.

Example:

```
IF value < 7 THEN

    WHILE value < 8 DO

        value := value +1;

    END_WHILE;

END_IF;
```

5.2.1 Expression

An expression is a type of structure. Its calculated value can be used in instructions.

An expression consists of operators and operands. An operand can be a constant, variable, function call, or expression. Example:

- Constant, such as 20, t#20s, and '22231 test'

- Variable, such as iVar and Var1[2,3]

- Function call, with a call return value, such as Fun1(1,2,4)

- Other expressions, such as 10+3, var1 OR var2, (x+y)/z, and iVar1:=iVar2+22

An expression is calculated by calculating the values of operands based on the operator priority. Value calculation is based on a descending order of operator priorities. Operators with the same priority are executed from left to right based on their positions in the expression.

For example, A, B, C, and D are of the INT type and their values are 1, 2, 3, and 4, respectively. Then A+B-C*ABS(D) is equal to -9, and (A+B-C)*ABS(D) is equal to 0.

When an operator has two operands, the value of the left operand is calculated first. For example, in the expression SIN(A)*COS(B), the values of SIN(A), COS(B), and the product are calculated in sequence.

The following table lists the operators of the ST language.

Operation	Symbol	Priority
Parentheses	(Expression)	Highest
Function call	Function name (parameter list, separated by commas)	
Exponentiation	EXPT	
Negation value calculation	-	
Complementing	NOT	
Multiply	*	
Divide	/	
Mod	MOD	
Add	+	
Subtract	-	
Compare	<, >, <=, and >=	
Equal to	=	
Not equal to	<>	
Logical AND	AND	
Exclusive OR	XOR	
Logical OR	OR	Lowest

5.2.2 ST Instruction

The ST program consists of instructions, which are separated by semicolons (;). The instructions consist of keywords and expressions. The following table lists the ST instructions.

Keyword	Description	Example
:=, S=, and R=	Assign value, set, and reset	`A:=B; C:=SIN(X); b1 R=cond1;`
	Function block call and output	`CMD_TMR(IN := %IX5, PT := 300); A:=CMD_TMR.Q`
RETURN	Return (exit the current POU)	`RETURN;`

Keyword	Description	Example

IF	Select	```D:=B*B;```
		```IF D<0.0 THEN```
		```C:=A;```
		```ELSIF D=0.0 THEN```
		```C:=B;```
		```ELSE```
		```C:=D;```
		```END_IF;```

CASE	Multiple selection	```CASE INT1 OF```
		```1: BOOL1 := TRUE;```
		```2: BOOL2 := TRUE;```
		```ELSE```
		``` BOOL1 := FALSE;```
		``` BOOL2 := FALSE;```
		```END_CASE;```

FOR	FOR loop	```J:=101;```
		```FOR I:=1 TO 100 BY 2 DO```
		```IF ARR[I] = 70 THEN```
		```J:=I;```
		```EXIT;```
		```END_IF;```
		```END_FOR;```

WHILE	WHILE loop	```J:=1;```
		```WHILE J<= 100 AND ARR[J] <> 70 DO```
		```J:=J+2;```
		```END_WHILE;```

REPEAT	REPEAT loop	```J:=-1;```
		```REPEAT```
		```J:=J+2;```
		```UNTIL J= 101 OR ARR[J] = 70```
		```END_REPEAT;```

EXIT	Exit loop	```EXIT;```
CONTINUE	Continue the next execution loop	```CONTINUE;```
JMP	Jump	```label: i:=i+1;``` ```JMP label;```
;	Empty statement	```;```

1 Valuation

A valuation instruction assigns values to variables. In a valuation keyword, the left part is a variable and the right part is the value to be assigned by the keyword.

Example: Var1 := Var2 * 10;

After execution, the value of Var1 is 10 times that of Var2. Three types of valuation keywords are provided: ":=", "S=", and "R=".

■ ":=" indicates general valuation, in which the right-side value is directly assigned to the left-side value and the two values are equal.

■ "S=" indicates set valuation, in which the left-side variable is changed to TRUE (set) if the right-side value is TRUE, until it is initialized using the R= instruction.

■ "R=" indicates reset valuation, in which the left-side variable is changed to FALSE (reset) if the right-side value is TRUE. It is used to reset the variables set by the S= instruction,

for example, a S= b;.

Once the value of b changes to TRUE, the value of a remains TRUE, even after the value of b changes to FALSE.

2 Function block call

Syntax: <FB instance name>(FB input variable:=<value and address>|, <more FB input variables:=<value and address>|...more FB input variables);

In the following example, a delay function block (TON) is called, and the **IN** and **PT** parameters are allocated. The result variable Q is allocated to variable A. Delay FB is instantiated through "TMR:TON".

Syntax:

```
<FB instance name>, <FB variable>:
TMR(IN := %IX5, PT := 300);
A:=TMR.Q;
```

3 RETURN instruction

The RETURN instruction indicates exiting the POU when the predefined condition is TRUE.

Syntax:

```
RETURN;
```

Example

```
IF b=TRUE THEN
RETURN;
END_IF;
a:=a+1;
```

If b is TRUE, the a:=a+1; statement is not executed, and the POU is returned immediately.

4 IF instruction

The IF keyword is used to determine the condition for executing instructions.

Syntax:

```
IF <boolean expression1> THEN
<IF_instruction>
{ELSIF <boolean expression2> THEN
<ELSIF_instruction1>
ELSIF <boolean expression n> THEN
<ELSIF_instruction-1>
```

```
ELSE

<ELSE_instruction>}

END_IF;
```

The content inside {} is optional.

If <boolean expression 1> is TRUE, only <IF_instruction> is executed whereas other instructions are not; otherwise, the boolean condition expressions starting from <boolean expression 2> are calculated one by one until the value of an expression is TRUE. Then, the instructions of the expression are executed. If no expression has the value TRUE, the instruction corresponding to <ELSE_instruction> is executed.

Example

```
IF temp<17

THEN heating_on := TRUE;

ELSE heating_on := FALSE;

END_IF;
```

Here, heating starts when the temperature is below 17°C; otherwise, heating remains disabled.

5 CASE instruction

The CASE instruction lists and processes the instructions corresponding to the multiple values of a conditional variable. Conditional variables must be integers.

Syntax:

```
CASE <Var1> OF

<value1>: <Instruction 1>

<value2>: <Instruction 2>

<value3, value4, value5>: <Instruction 3>

<value6 .. value10>: <Instruction4>

...

<value n>: <Instruction n>

ELSE <ELSE Instruction>

END_CASE;
```

The CASE instruction implements the following processing:

■ If the value of the <Var1> variable is <valueI>, <Instruction I> is executed.

■ If no value matches with <Var1>, <ELSE Instruction> is executed.

■ If the same instruction is executed in multiple variable values, you can write the values one by one and separate them with commas (,) so that they are executed simultaneously.

■ If the same instruction is executed within a variable range, you can write the start and end values and separate them with two periods (.). The preceding two conditions can be combined.

Example

```
CASE iVar1 OF
    2:c1:=c1+1;
    1,6:c1:=c1-7;
    7..20:c1:=c1+5;
    ELSE
        c1:=c1-1;
END_CASE
```

6 FOR loop

You can write a repeated processing logic through FOR loop.

Syntax:

```
FOR <INT_Var> := <INIT_VALUE> TO <END_VALUE> {BY <Step size>} DO

<instructions>

END_FOR;
```

The content inside {} is optional.

INT_Var is a counter of the integer type. <Instructions> is executed as long as <INT_Var> is not greater than <END_VALUE>. Check the condition before executing <Instructions>. <Instructions> is not executed if <INIT_VALUE> is greater than <END_VALUE>.

<INT_Var> increases by <Step size> each time after <Instructions> is executed. <Step size> can be any integer. The default value **1** applies if this parameter is not set. Loop stops when <INT_Var> is greater than <END_VALUE>.

Example

```
FOR Counter:=1 TO 5 BY 1 DO

Var1:=Var1*2;

END_FOR;

Erg:=Var1;
```

Assume that the default value of Var1 is **2**. The value changes to **32** after FOR loop.

7 WHILE loop

WHILE loop can be used by loop processing. Different from FOR loop, WHILE loop supports loop conditions of arbitrary boolean expressions. The loop is executed once the loop condition is met; otherwise, it is exited.

Syntax:

```
WHILE <boolean expression> DO

<instructions>

END_WHILE;
```

When <Boolean_expression> is TRUE, <Instructions> is executed, until <Boolean_expression> changes to FALSE. <Instructions> is never executed if the initial value of <Boolean_expression> is FALSE. If <Boolean_expression> is never set to FALSE, <Instructions> is executed without stop. This results in infinite loop, which is not allowed during programming.

Example

```
WHILE Counter<>0 DO

Var1:= Var1*2;

Counter := Counter-1;

END_WHILE
```

WHILE loop and REPEAT loop are more powerful than FOR loop because the former do not calculate the loop times before loop is executed. Therefore, WHILE loop and REPEAT loop are sufficient in some cases. FOR loop is more efficient when the loop times are known.

8 REPEAT loop

Different from WHILE loop, REPEAT loop checks the loop condition after the loop instruction is executed. This means the loop is executed at least once, regardless of the loop condition.

Syntax:

```
REPEAT

<instructions>

UNTIL <Boolean expression>

END_REPEAT;
```

The execution logic is that <Instructions> is executed until <Boolean expression> is TRUE. <Instructions> is executed only once if the initial value of <Boolean expression> is TRUE. If the value of <Boolean_expression> is never TRUE, <Instructions> is executed permanently, resulting in infinite loop.

Example

```
REPEAT

Var1:=Var1*2;

Counter:=Counter-1;

UNTIL Counter=0;

END_REPEAT;
```

9 CONTINUE statement

The CONTINUE instruction is used in FOR, WHILE, and REPEAT loops. It terminates the current loop in advance and starts the next loop.

Example

```
FOR Counter:=1 TO 5 BY DO

INT1:=INT1/2;

IF INT1=0 THEN

CONTINUE;

END_JF

Var:=Var1/UBT1L

END_FOR;

Erg:=Var1;
```

10 EXIT statement

The EXIT instruction is used to exit the FOR, WHILE, or REPEAT loop.

11 JMP statement

The JMP instruction is used to jump to the code line marked by the specified label.

Syntax:

```
<label>:

JMP <label>;
```

<label> is at the start of the program line. The JMP instruction requires a jump target, that is, a predefined label. When the JMP instruction is reached, the program jumps to the code line marked by the specified label for execution.

Example

```
aaa :=0;

_label11:aaa:=aaa+1;

(*instructions*)

IF (aaa < 10) THEN

JMP _label1;

END_IF;
```

The initial value of the variable aaa is 0. As long as the value is less than 10, the program jumps to the code line marked by label 1 for execution. This affects the repeated execution of the program between the JMP instruction and label.

Such a function can also be implemented by the WHILE or REPEAT loop. Be cautious when using the JMP instruction because it reduces code readability.

12 Comment

The ST provides two comment write methods.

- ■ "(*"start,"*)"end. This allows for cross-line commenting, for example, "(*This is a comment.*)".

- ■ Single-line commenting, which uses "//" to indicate the start of a comment extending to the end of the current line, for example, "// This is a comment."

A comment can be inserted into the declaration or implementation part of the ST editor.

Comment nesting: A comment can be inserted into another comment.

Example

```
(*

a:=inst.out; (*to be checked*)

b:=b+1;

*)
```

5.3 Ladder Diagram (LD)

5.3.1 Overview

The LD is a graphic programming language and similar to the structure of a circuit diagram. The LD consists of a series of networks (also called segments), and each network starts from the left-side vertical line (power rail and energy flow line). A network consists of contacts, coils, optional POUs (functions, function blocks, and programs), jump, labels, and connection lines.

The bus on the left is the energy flow line and is always TRUE. Contacts, operation blocks, and coils are connected after the bus. Each contact is allocated with a boolean variable. If the variable is set to TRUE, the switch is turned on and the condition is transferred along the connection line from left to right. If the variable is not TRUE, the switch is turned off. The coil on the right receives the On or Off signal transmitted from the left side. TRUE or FALSE is written to the boolean variable associated with the coil. The following figure shows the LD edit page.
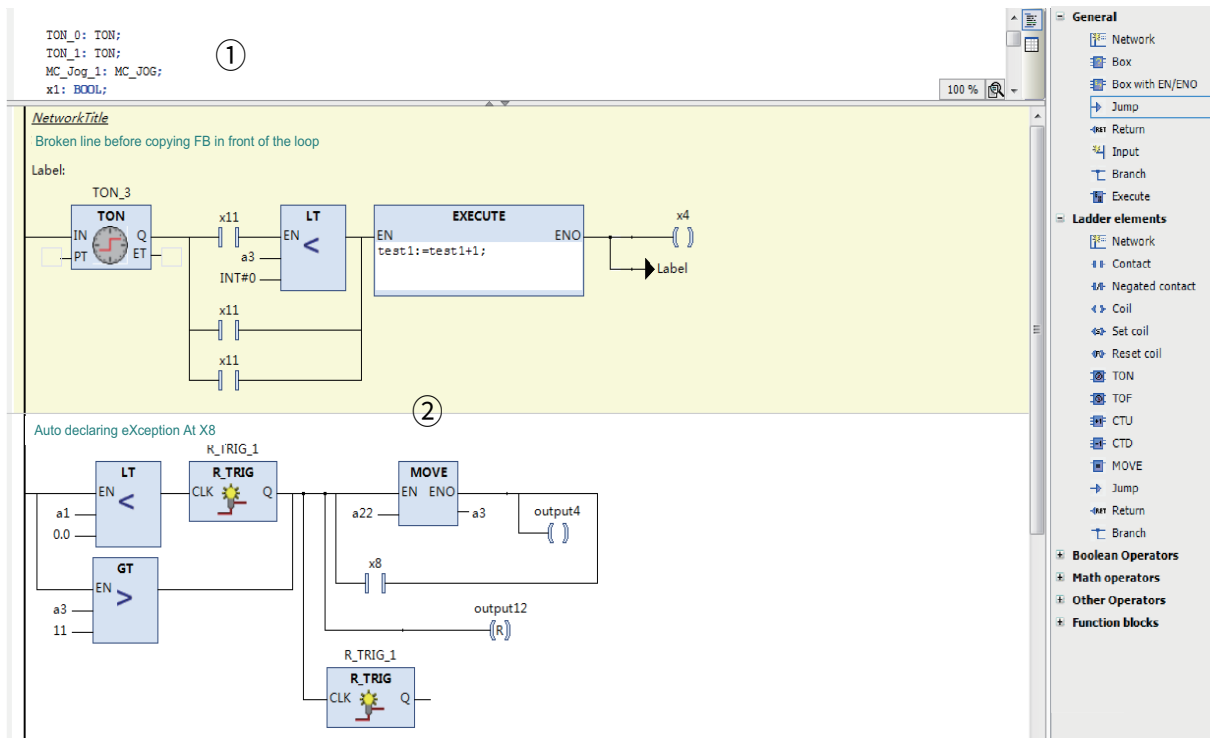
1: variable definition area; 2: LD programming area; 3: tool box.

The LD contains the following main elements: contact, coil, operation block, branch, and comment. You can add the elements to the network to form an LD execution logic through the insert, drag and drop, underline, and copy and paste operations. To set the page font, operands, and comment display of the LD, choose **Tools** > **Options** > **FBD/LD editor**.

The LD supports online debugging of the monitoring function, written values, mandatory values, and breakpoints.

Branches are classified into closed and non-closed branches. Closed branches are called parallel branches, whereas non-closed branches are called branches.
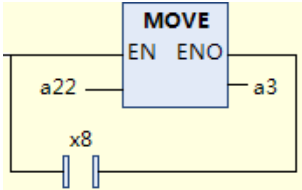
5.3.2 LD Elements

The LD elements include network, contact, coil, operation block, execution block, branch, jump, label, and return.

The input and output of contacts, coils, and operation blocks are related to operands, which can be variables, constants (TRUE or FALSE; 1 or 2), and addresses. For details, see the variable definition.

The LD elements are displayed under **ToolBox** (choose **View** > **ToolBox**), as shown in Figure 5-1. The toolbox includes general elements, LD elements, IEC standard operators (such as boolean operators and match operators), function blocks, and POU defined in the current program.
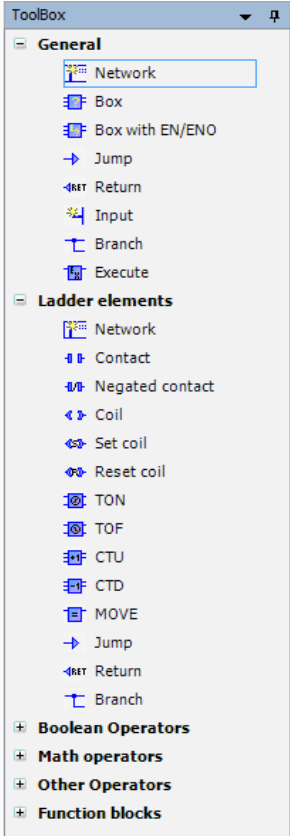
Figure 5-1 LD toolbox

1 Network

Icon -

The LD consists of a series of networks. All the other LD elements are within networks. Each network is indicated by a serial number on the left.

You can insert a network title (summary about the network) and comment (detailed description about the network). Choose **Tools** > **Options** > **FBD/LD editor** > **General** to show or hide the network title and comment.

You can insert a label below the network title and comment to indicate the jump target.

Use the menu command **Toogle network comment state** to enable or disable the network.

The network decoration area between the network serial number and network content displays the breakpoint mark and bookmark position.
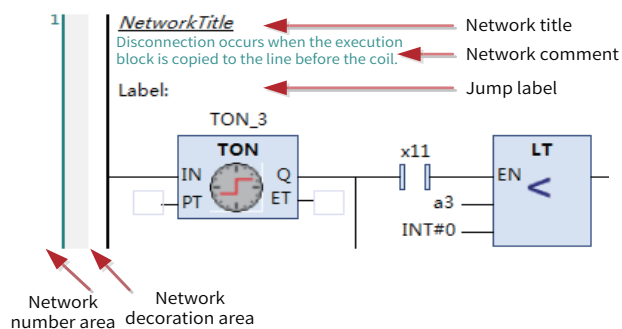
2 Contact

Icon -

Contacts are classified into normally open (NO) and normally closed (NC) contacts. Contacts are boolean variables and used to transfer ON (TRUE) and OFF (FALSE) values. If the variable value is TRUE, the NO contact transfers ON (TRUE) to the right; otherwise, it transfers OFF (FALSE). The NC contact transfers reverse values.

You can add the edge signal function to contacts. Right-click a contact and choose **Edge Detect** from the context menu to change the contact to a rising-edge trigger contact or a falling-edge trigger contact. The rising-edge trigger contact transfers ON to the right when the variable value of the contact changes from FALSE to TRUE. The falling-edge trigger contact transfers ON to the right when the variable value of the contact changes from TRUE to FALSE.

3 Coil

Icon -

Coils are located at the end of the network. The logical operation results on the left are assigned to coil variables. Coil variables are only of the boolean type. TRUE indicates ON and FALSE indicates OFF. Parallel coils can only be inserted upward or downward.

Coils are classified into coils, negated coils, set coils, and reset coils. You can switch between the four coil types by using the right-click menu command or shortcut.

■ Coil: assigns the logical operation results on the left to coil variables directly.

■ Negated coil: negates the logical operation results on the left and assigns the values to coil variables.

■ Set coil: sets the value of the coil variable to ON (TRUE) if the left-side status value of the coil is ON (TRUE), and keeps the value until the variable is reset to OFF (FALSE) by the reset coil.

■ Reset coil: resets the set coil.

4 Box (operation block)

Icon -

An operation block can be an operator, function, function block, program, action, or method. If the operation block is of the function block type, a text box is displayed above the operation block box to display the function block instance.

An operation block includes at least one input and one output. The following figure shows the components of an operation block.
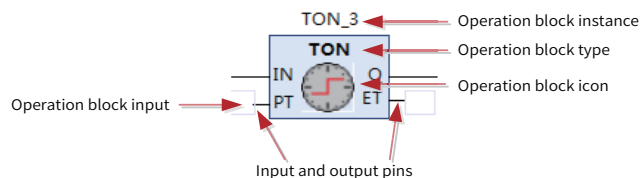
Operation blocks are classified into common operation blocks and EN/ENO operation blocks.

■ EN/ENO operation block: contains EN input and ENO output, in addition to the input and output provided by an operation block. The execution logic of the EN/ENO operation block is as follows: The operation block logic is executed when EN is TRUE. ENO is TRUE after execution. The operation block is not executed if EN is FALSE. In this case, ENO is FALSE. Note: The input line of the EN/ENO operation bock can only be connected to the EN pin, and the output line can only be connected to the ENO pin.

■ Input and output pins of the operation block: The negation, rising edge, and falling edge signals can be added to the input pin of the boolean type. The negation signal can be added to the output pin of the operation block.

■ Multi-input line operation block: The operation block contains multiple inputs that are connected to the energy flow line. Figure 5-2 shows an operation block with two input lines. Because a multi-input line operation block provides multiple lines connected to the energy flow line, it is only located in the first branch and cannot be connected to branches in parallel.
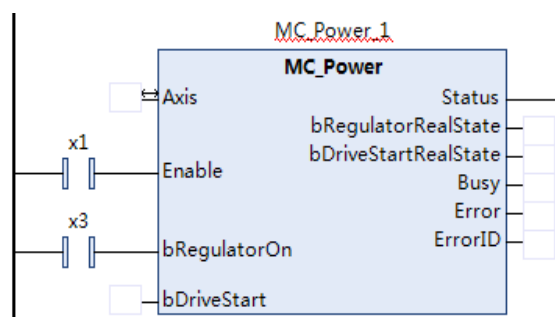
Figure 5-2 Multi-input line operation block

5 Execute (execution block)

Icon -

An execution block can be inserted into the embedded ST. ST statements can be edited within the block. The execution block can be zoomed in or out. The maximum size is 1000*400.

6 Branch

Icon -

Branches form a non-closed parallel logic.

7 Label

Icon -

A label indicates a jump location and is at the head of the network. The system jumps to the label position through jump elements. A jump label is a string and must comply with the naming conventions.

8 Jump

Icon - →

When the input on the left of a jump element is TRUE, the system jumps to the specified label position for execution. The jump element is on the rightmost of the network.

9 Return

Icon - ◀RET I

When the input on the left of the return element is TRUE, the current program exits execution immediately. The return element is on the leftmost of the network.

5.3.3 LD Editor Options

The LD editor options are used to control the LD screen display, single-key command setting, and print display mode. Access the LD editor options by choosing **Tools** > **Options** > **FBD/LD editor**. The LD editor provides three tabs: **General**, **LD**, and **Print**.

1 General setting

The following figure shows the **General** tab page.
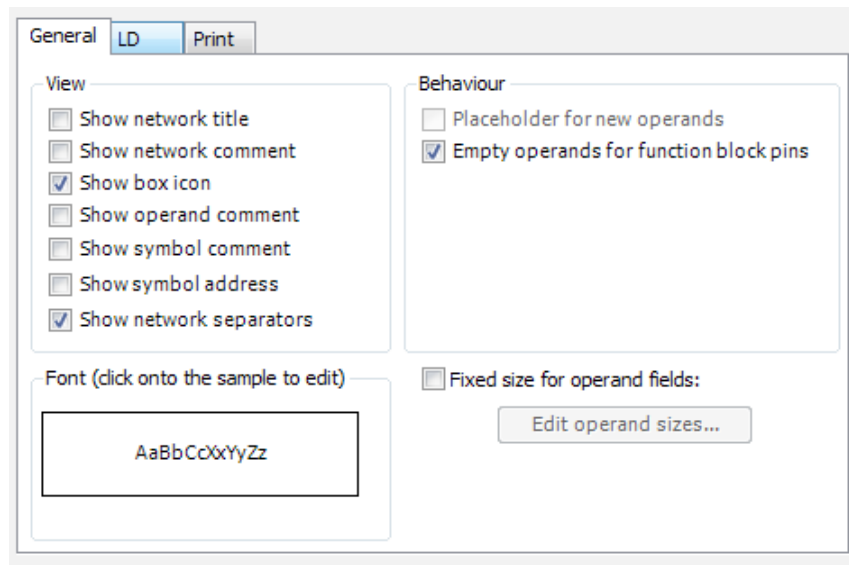
General tab page of the LD editor

View

- ■ **Show network title**: If this option is selected, you can insert and edit the title of each network of the LD. The inserted title is displayed above the current network. If no title exists, the title line is not displayed. A title is inserted by using a menu command.

- ■ **Show network comment**: If this option is selected, you can edit the comment of each network of the LD. If a network comment is added, it is displayed below the network title. If no network comment exists, the comment line is not displayed. A network comment is edited by using a menu command.

- ■ **Show box icon**: If this option is selected, the icon defined for the operation block is displayed in the middle of the block. Icons are defined for standard operands (such as ADD and SUB) and function blocks (such as TON and TOF). To add images as operation block icons to user-defined functions,

function blocks, or programs, right-click an object and choose **Properties** > **Bitmap** > **Click** to select project-related bitmap.

■ **Show operand comment**: If this option is selected, you can edit and display the comment of each operand on the LD page. An operand is a programming concept. Variables, constants, and addresses are operands. Because the LD does not necessarily use variables, when constants or addresses are used, you can describe them by using operand comments. When editing an operand comment, select an operand string and right-click it.

■ **Show symbol comment**: If this option is selected, the comments during variable declaration are displayed for the variables on the LD page. Variable comments come from variable declaration and cannot be edited.
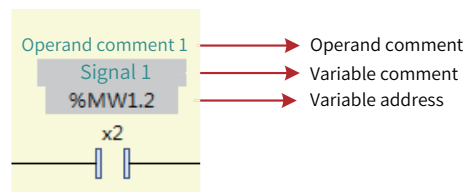
Font

Click the sample text. The font selection box is displayed, where you can set the font of the LD. The default font is Microsoft YaHei and the default font size is 9. The font range mainly covers operands and comments. The execution block font uses the text editor font (ST text and variable declaration text).

Behavior

■ **Placeholder for new operands**: not implemented.

■ **Empty operands for function block pins**: If this option is selected, the input and output pins of a new operation block use empty characters. If this option is not selected, the input and output pins of the operation block use "???".

Operand Fixed Size Settings

If this option is selected, you can set **Operand width**, **Operand comment height**, and **Symbol comment height**. See the following figure.
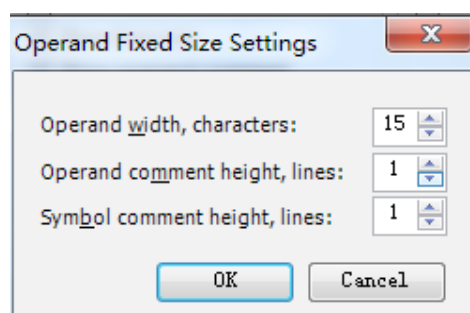
Operand Fixed Size Settings

■ **Operand width**: Set the number of fixed characters of an operand. The default value is **15**.

■ **Operand comment height**: Set the number of operand comment lines of fixed length. The default value is 1.

■ **Symbol comment height**: Set the number of variable comment lines of fixed length. The default value is 1.

2 LD setting

The following figure shows the **LD** tab page.
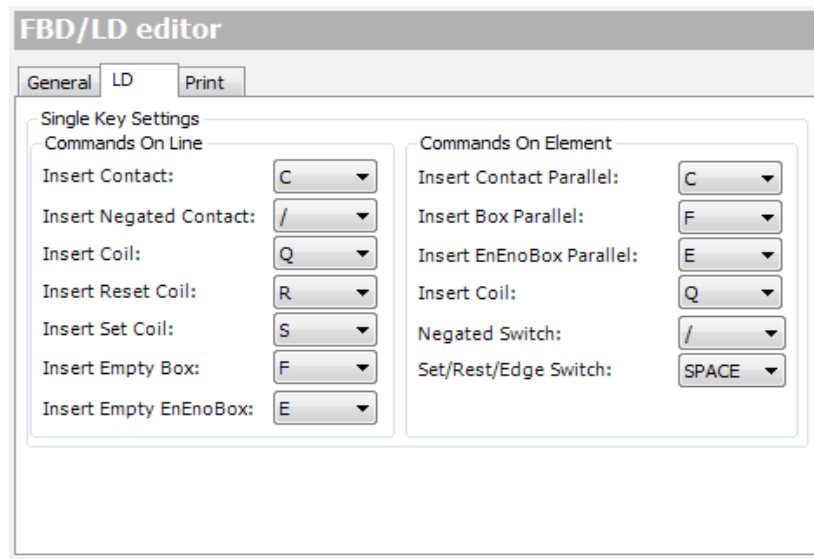
Single key settings

The single key settings function enables the edit operation through a single key, including single keys executed on lines and those executed on elements. The single-key commands executed on lines insert serial elements, whereas the single-key commands executed on elements insert parallel elements.

You can switch element functions when selecting elements, such as negation switching, edge signal switching, and set/reset switching. Negation switching is applied to contacts and coils and uses the **/** key. Edge signal switching is applied to contacts and uses space. Set/Reset switching is applied to coils and uses space.

You can set a single key for implementing a function. Each function has a default single-key character. You can set keys as needed.
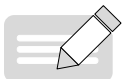
> **NOTE**
>
> The same key cannot be used repeatedly under **Commands on Line** or **Commands on Element**, but **Commands on Line** and **Commands on Element** may share the same key.

3 Print

The following figure shows the **Print** tab page.
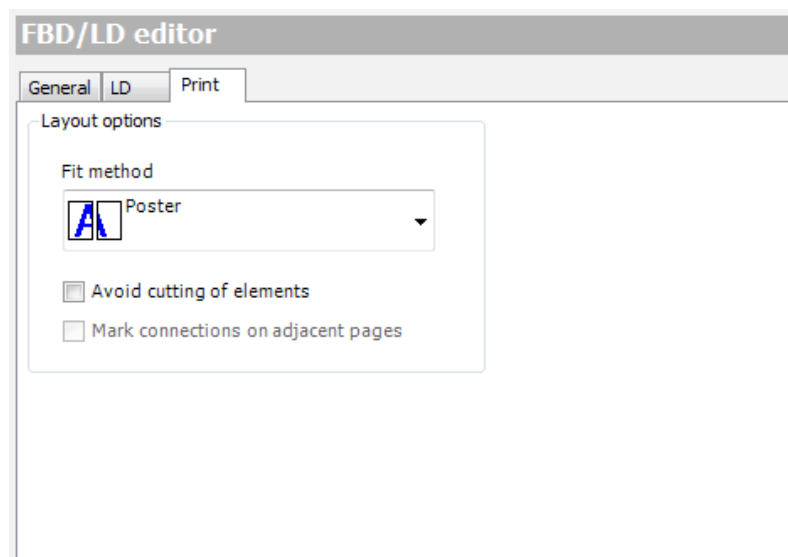
Layout options

Fit method:

- **Poster**: Print based on the normal proportion. If the current page is not high enough to display the entire network, the next page is printed. If the page is wide enough to display the entire network, the remaining part is printed on the next page.

- **Shrink to fit the widest network**: The displayed content is compressed during printing so that all the networks are displayed within the width of the page. If a page is not high enough to display the entire network, the remaining part of the network is displayed on the next page.

- **Avoid cutting of elements**: If this option is selected, when an element is displayed between two pages, the element is placed on the next page for display. This option is available only when **Poster** is selected for **Fit method**.

- **Mark connections on adjacent pages**: If this option is selected, the connection between two pages is marked. This option is available only when **Avoid cutting of elements** is selected.

5.3.4 Element Selection

Selection is the basis of the edit operation. You can select elements or lines, select one object at a time or select multiple objects simultaneously by pressing **Ctrl** or **Shift**, and select consecutive or nonconsecutive objects.

A selected element is highlighted. For example, a selected contact is displayed as ┤│ │├ . The external dotted box indicates that the contact is focused. You can select an element and paste it to another element in parallel mode, or drag and drop it to another element in serial or parallel mode. Because multiple selected elements may not be consecutive, you need to describe the result logic formed by the selected elements. The selection result logic is intended to keep the original logic consistent. You can select an element through box select, and select multiple or all elements by pressing **Ctrl** or **Shift**.

1 Result logic formed by selected elements

- Single-element selection: When you select a contact or coil, only the selected contact or coil is included. When you select an operation block, the operation block and the line-free input and output operands are included. As shown in the following figure, when the ADD operation block is selected, the ADD operation block, inputs a and b, and output c are pasted.
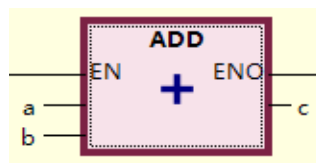
- Multi-element selection: Select the serial connection on a line (including nonconsecutive selection) and select elements on a parallel line (including nonconsecutive selection) to form parallel results.
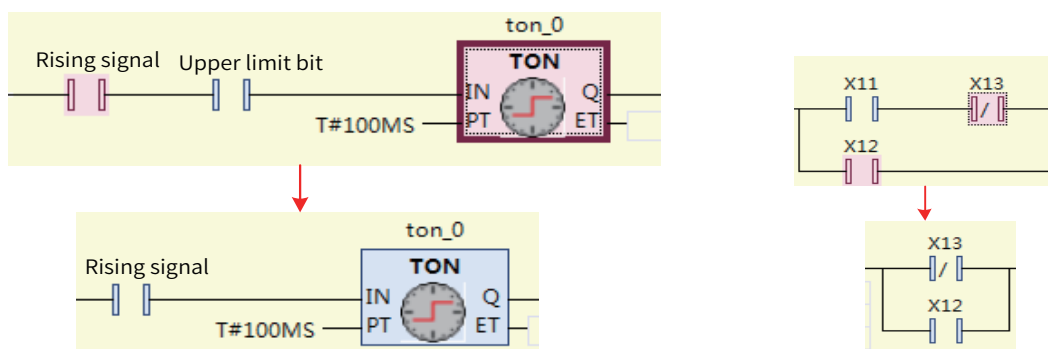
Parallel logic formation

■ For multi-element selection: If the selected elements span multiple branches, the results also span branches, and the original logic remains unchanged. If the selected elements span two branches, the elements are connected serially.
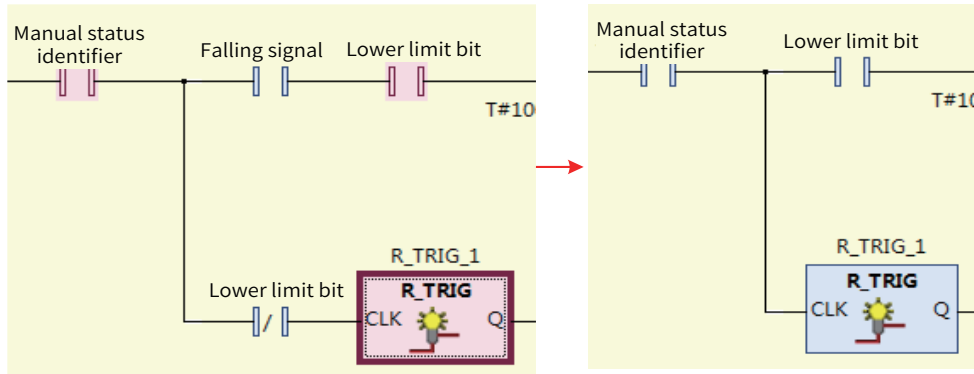
The results span multiple branches if the selected elements come from more than two branches.
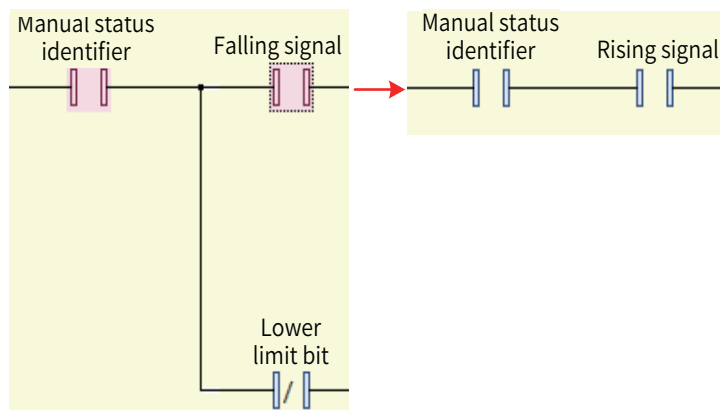
If the selected elements come from two branches, the results of the two branches are connected serially.

■ For multi-element selection: If you select the multi-input line operation block and the elements on multiple input lines, the results are the same as the selected ones. If you only select the elements on multiple input lines but do not select the operation block, the elements on the input lines are connected in parallel to form results.
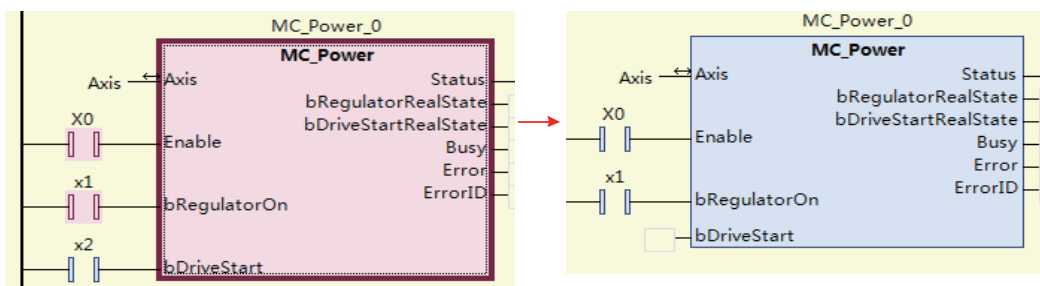
If you select the operation block and input elements, the results are the same as the selected ones.
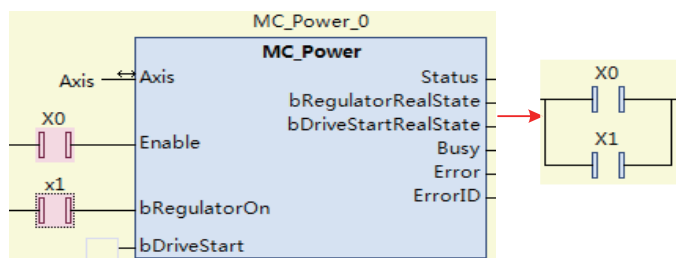
If you only select input elements but do not select the operation block, a parallel logic is formed.

2 Box select

The elements within the rectangle that starts from where the mouse is pressed and ends where the mouse is released, are selected. See the following figure.
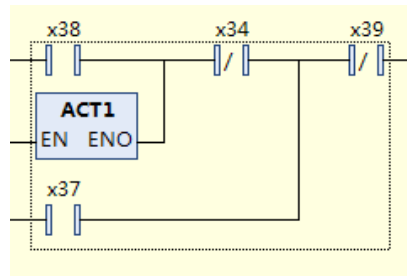
NOTE

Box select is only applicable to single network selection. Selection starts from the blank area where the mouse is pressed.

3 Multi-element selection through Ctrl and Shift

Multi-element selection through **Ctrl** and **Shift** complies with the standard multi-element selection method.

- When you press **Ctrl**, if the current element is not selected, it is added to the selection list. If the current element is already selected, it is removed from the selection list.

- Press **Shift** to select elements within the rectangle from the last selected element to the currently selected element.

4 Select all

Press **Ctrl+A** to select all networks.

5.3.5 Standard Edit Commands

The LD supports standard and common edit operations, such as copy, paste, delete, cut, undo, and restore. Standard edit shortcut keys are used.

1 Copy

Copies the selected element. The copy result is the selected element. For details, see the section about element selection.

The following elements of the LD can be copied: network, contact, coil, operation block, string, and branch line.

The copied elements may be consecutive or not, depending on the selection. The elements to be copied must be within a single network. If elements across networks are selected, only the data selected in the focused network is copied.

2 Paste

Pastes the copied elements. The paste rules are as follows:

- Paste on line

  When "paste on line" is selected, the copied element is inserted in the line position to form a serial relationship.

Select the element to be copied · Paste position · After paste
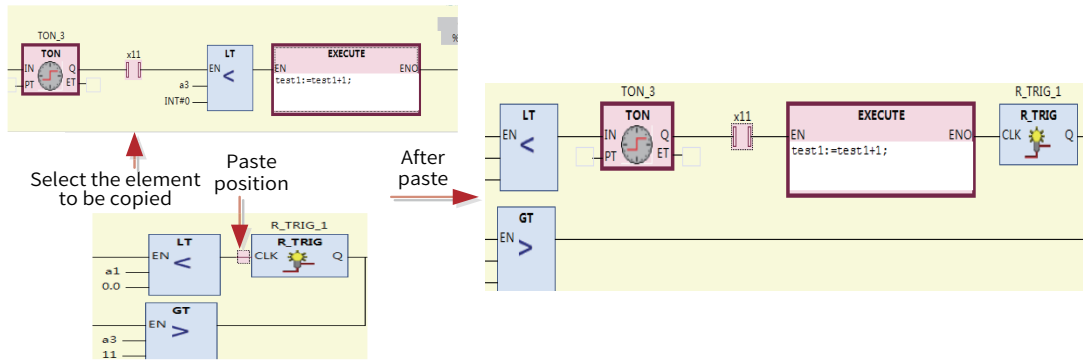
■ Paste on element

When "paste on element" is selected, the elements selected in batches form a parallel relationship. When a parallel relationship is formed, the selected elements must meet the parallel conditions for the paste operation. That is, the selected elements must be on the same line and consecutive and cannot span branches, and the start element and end element cannot connect to the branch internally and externally in parallel.

Copy the selected element · Select the element to be pasted · The selected element and copied element are connected in parallel after paste.
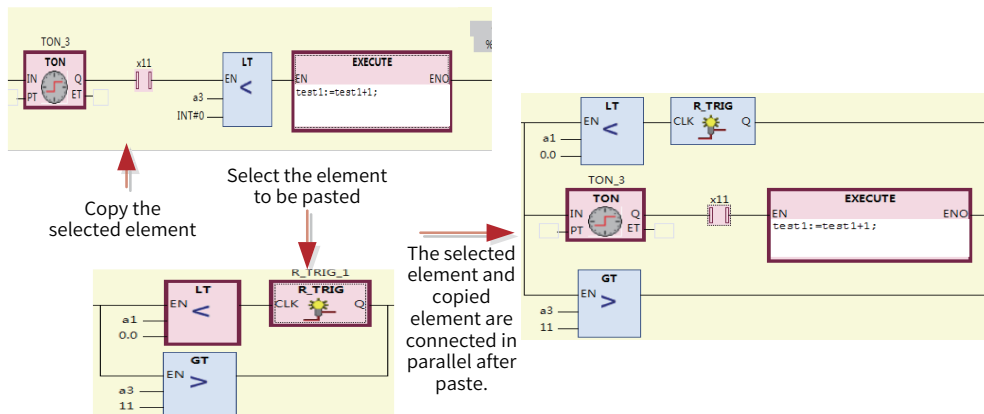
■ Paste in coil position

Because no elements exist on the right of a coil, special paste rules must be observed. The rules for jump elements and return elements are the same as those for coils. Coil is used as an example.

1) If a coil is selected (parallel element connection is selected), the pasted elements are located in a new branch below the coil, as shown in the following figure. Select Y5 to paste S0 and Y6. S0 and Y6 are located in a new branch below the Y5 coil.
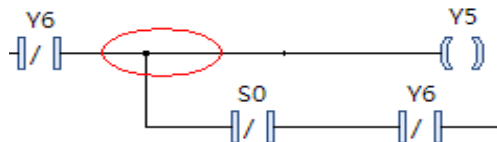
2) If a line before the coil is selected (serial line connection is selected), the copied content may include only one branch or multiple branches, depending on the copied elements.

Only one branch in the copied content

1) If the copied content does not include the coil, the copied elements are connected to the coil serially.

2) If the copied content includes the coil, the data before the copied coil is inserted in the line selection position, and the copied coil and the coil after the selection line form a non-closed parallel connection, as shown in the following figure. Select the connection line before Y5 to paste S0, Y6, and Y8. After pasting, S0 and Y6 before Y8 are serially connected to the line before Y5. Y8 and Y5 are connected in parallel through a branch.
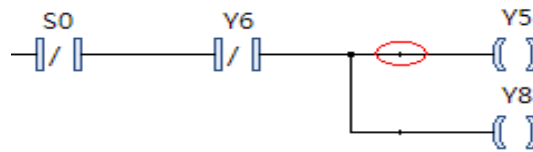
Multiple branches in the copied content

1) If the last branch horizontal to the copied content does not include the coil, the copied content is inserted in the line selection position.

2) If the last branch horizontal to the copied content includes a coil, connect this branch to the coil after the connection line in parallel mode, and keep other data connected in serial or parallel mode unchanged, as shown in the following figure. Paste the copied X1, Y0, and X2 to the connection line before the Y1 coil. Y0 is located below the Y1 coil, and X1 is located on the connection line before Y1.
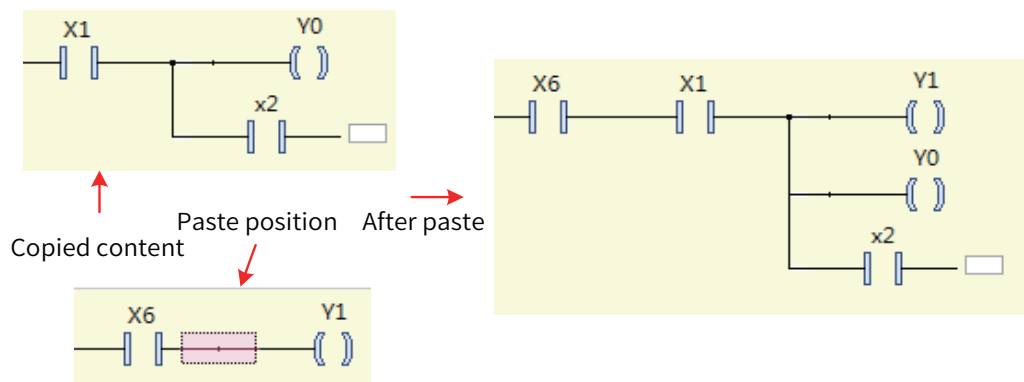
■ Paste multi-input line operation block

The multi-input line operation block is only located in the non-parallel position of the first branch, that is, the paste position. If the multi-input line operation block is not included, the connection elements of the copied operation block starting from the second input line are deleted, as shown in the following figure. The X0 and X1 contacts of the copied operation block are copied, and X1 is deleted when being pasted.
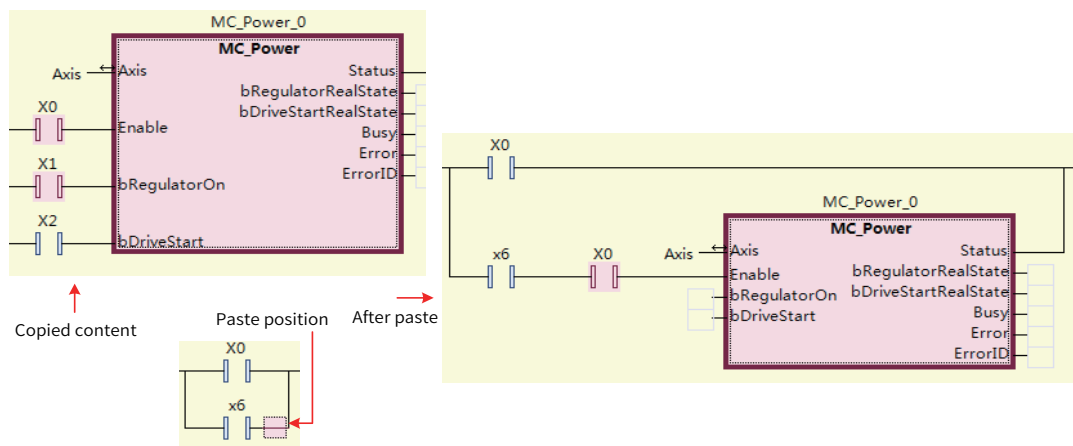
■ Paste network

You can select one or more networks for the copy and paste operations. Selection is required in advance.

■ Paste a single branch line

This function is used to add a single branch. It is the same as the function of inserting a branch upward or downward. You can copy a single branch line and paste it in the branch output position. The branch line is pasted below the selected branch, as shown in the following figure.

After paste

Copied content Paste position
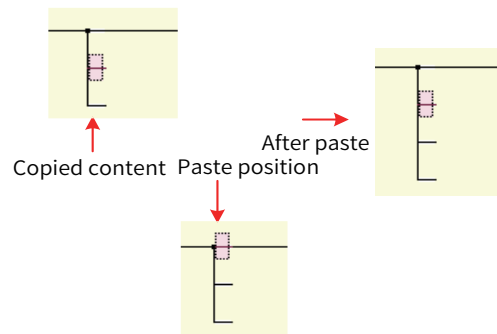
3 Delete

Deletes the selected element. After the current element is deleted, the next element is selected to ensure operation continuity.

4 Cut

Copies the selected element, pastes it, and deletes it.

5 Undo/Restore

Undo: returns to the previous edit status and restores the previously selected element.

Restore: restores the next edit status and the next selected element.

5.3.6 LD Menu Commands

Menu commands include the right-click menu commands and the LD commands in the LD toolbox.

1 Insert network

The "insert network" and "insert network (below)" menu commands are provided.

Command execution condition: A network is selected and another network is inserted above or below the selected network.

Insert network: icon - ⧉ ; shortcut key: **Ctrl**+**I,** which inserts an empty network above the selected network.

Insert network (below): icon - ⧉ ; shortcut key: **Ctrl**+**T**, which inserts an empty network below the selected network.

2 Switch network comment status

Icon - ⧉ ; shortcut key: **Ctrl**+**O**, which switches a network between the comment state and non-comment state.

In the comment state, the code of the entire network is invalid and not executed, and the execution block cannot be edited.

Command execution condition: A network is selected.

3 Insert network header information

A network header mainly includes the network title, network comment, and label.

The commands for inserting a network header include "insert label", "edit network title", and "edit network comment".

- Edit network title: icon - , which edits the title of the selected network.

- Command execution condition: A network is selected and **Show network title** is selected.

- Edit network comment: icon - , which edits the comment of the selected network.

- Command execution condition: A network is selected and **Show network comment** is selected.

- Insert label: icon - , which inserts a jump label to the selected network to indicate the jump position of a jump element.

- Command execution condition: A network is selected.

4 Insert operation block

The following five menu commands are provided: insert operation block, insert empty operation block, insert EN/ENO operation block, insert EN/ENO empty function block, and insert parallel operation block (below). The commands are used to insert operators, functions, function blocks, and programs. You can also drag and drop the operation block or EN/ENO operation block from the toolbox to insert an operation block.

The first four commands are used to insert serial operation blocks, and the last command is used to insert operation blocks parallel to selected elements.

Insertion position:

1) Select a horizontal line and insert an operation block on the horizontal line.

2) Select a vertical line (parallel brackets). For the left bracket, insert the operation block on the left line of the bracket. For the right bracket, insert the operation block on the right line of the bracket.

3) Select an element and insert an operation block to the left of the element.

- Insert operation block: icon - ; shortcut key: **Ctrl**+**B**, which displays the input assistant for you to select the operation block to be inserted.

- Insert empty operation block: icon - ; shortcut key: **Ctrl**+**Shift**+**B**, which inserts an empty operation block, without using the input assistant. You can specify the operation block type in the corresponding location.

- Insert EN/ENO operation block: icon - ; shortcut key: **Ctrl**+**Shift**+**E**, which displays the input assistant for you to select the operation block to be inserted. The operation block provides EN/ENO input and output. The EN/ENO operation block is executed only when EN is TRUE. It is not executed when EN is FALSE. ENO has the same result as EN.

- Insert EN/ENO function block: inserts an empty operation block, without using the input assistant. The operation block provides EN/ENO input and output.

- Insert parallel operation block (below): inserts an empty operation block below the selected element. The selected element can be a contact or operation block.

5 Insert execution block

Insert execution block: icon - , which inserts a serial execution block in the selected location. You can also drag and drop an execution block from the toolbox.

Insertion position:

1) Select a horizontal line and insert an execution block on the horizontal line.

2) Select a vertical line (parallel brackets). For the left bracket, insert the execution block on the left of the bracket. For the right bracket, insert the execution block on the right of the bracket.

3) Select an element and insert an execution block to the left of the element.

An execution block can be used to edit ST statements. Click the text area for editing. The execution block only provides EN/ENO input and output.

Drag to change execution block size: Drag the execution block frame in the editable state to control the block size, as shown in the following figure.

Drag to adjust the size of the execution block
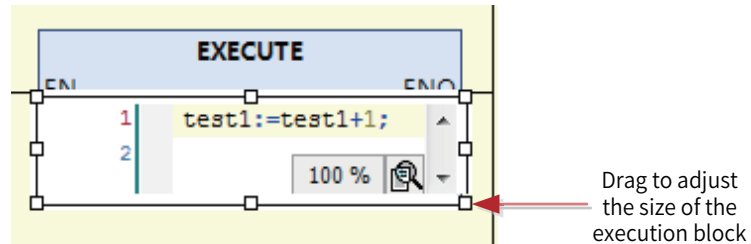
6 Insert input

Insert input: icon - ; shortcut key: **Ctrl**+**Q**, which adds input to an variable-input operation block.

Variable-input operation blocks: ADD, +, MUL, *, SEL, AND, &, OR, |, XOR, MAX, MIN, and MUX

Insertion position: When an input pin is selected, an input is added before the input pin. When an operation block is selected, the added input pin is located at the end.

7 Insert coil

Three menu commands are provided: insert coil, insert set coil, and insert reset coil. You can also insert coils by dragging and dropping **Coil**, **Set coil**, and **Reset coil** from the toolbox.

■ Command execution condition: The selected position cannot be located on the parallel branch or in the input position of the multi-input line operation block.

■ Insert coil: icon ; shortcut key: **Ctrl**+**Shift**+**A**, which outputs a coil in the current position.

■ Insertion position:

1) Select a horizontal line and insert a coil on the line. The coil and line are processed by a non-closed branch.

Insert a coil on the line

2) Select a vertical line (parallel brackets). For the left bracket, insert the execution block on the left of the bracket. For the right bracket, insert the execution block on the right of the bracket.

Insert a coil to the bracket on the right
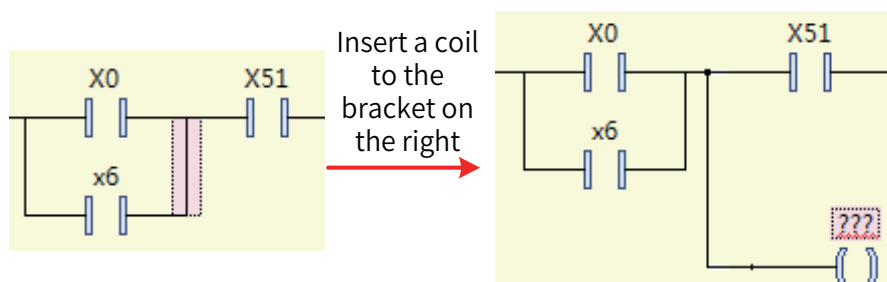
3) If **Network** is selected, the new coil is inserted at the end.

Select a network
and insert it to the
last branch
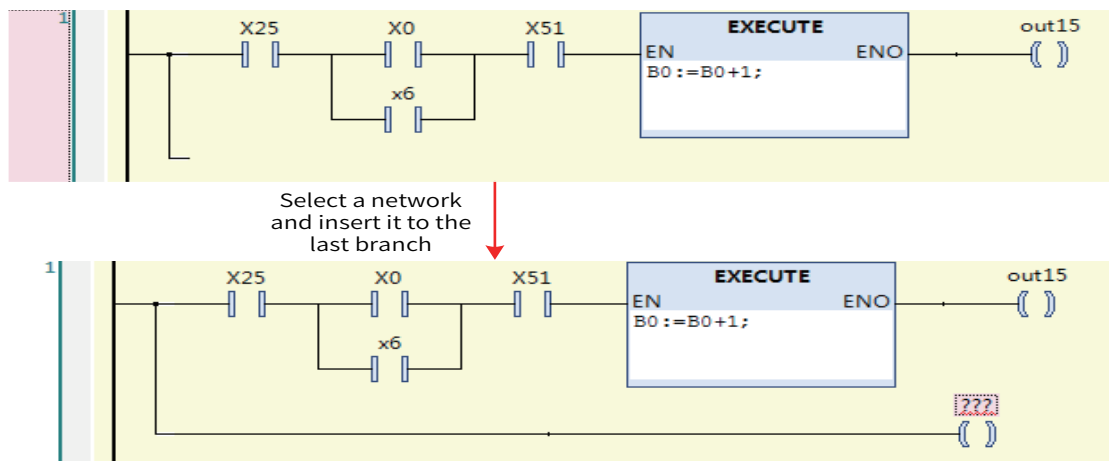
4) If **Coil**, **Return**, or **Jump** is selected, the new coil is inserted below the selected element.

Select a coil and
insert it below
another coil
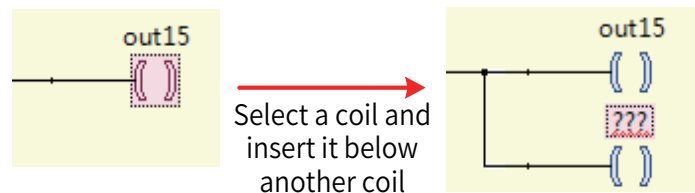
The default variable name of the inserted coil is "???". You need to enter the required variable or constant. You can use the input assistant (press **F2**) to select an input from the variable list.

■ Insert set coil: icon - , which inserts a set coil in the current position. The operation is the same as inserting a coil.

■ Insert reset coil: icon - - , which inserts a reset coil in the current position. The operation is the same as inserting a coil.

8 Insert contact

The following four menu commands are provided: insert contact, insert NC contact, insert parallel lower contact, and insert parallel upper contact. You can also insert a contact by dragging and dropping **Contact** or **Negated contact** from the toolbox.

■ Insert contact: icon - ; shortcut key: **Ctrl+K**, which inserts an NO contact in the current position in serial mode.

Insertion position:

1) Select a horizontal line and insert a contact on the horizontal line.

2) Select a vertical line (parallel brackets). For the left bracket, insert the execution block on the left of the bracket. For the right bracket, insert the execution block on the right of the bracket.

3) If a network is selected, the new contact is inserted at the end.

4) If an element is selected, the new contact is inserted on the left of the element.

The default variable name of the contact is "???". Click the variable or constant required by text input. You can use the input assistant (press **F2**) to select an input from the variable list.

■ Insert NC contact: icon - ▮▮ , which inserts an NC contact in the current position in serial mode. The operation is the same as inserting a contact.

■ Insert parallel lower contact: icon - ▮ ▮ ; shortcut key: **Ctrl+R**, which inserts an NO contact below the selected element in parallel mode. The selected element can be a contact or operation block.

■ Insert parallel upper contact: icon - ▮ ▮ ; shortcut key: **Ctrl+P**, which inserts an NO contact above the selected element in parallel mode. The operation is the same as inserting a parallel lower contact.

9 Insert branch

Three menu commands are provided: insert branch, insert branch above, and insert branch below. You can also insert branches by dragging and dropping **Branch** from the toolbox. A branch is a non-closed line and different from a parallel branch.

■ Insert branch: icon - ▮ ; shortcut key: **Ctrl**+**Shift**+**V**, which inserts a branch in the selected line position.

■ Command execution condition: The selected position cannot be located on the parallel branch or in the input position of the multi-input line operation block.

■ Insertion position:

1) If a line is selected, the branch is inserted below the line.

2) If a contact or coil is selected, the branch is inserted before the selected element.

3) If the left bracket is selected, the branch is inserted on the left of the bracket. If the right bracket is selected, the branch is inserted on the right of the bracket.

As shown in the following figure, each selected position indicates a branch.
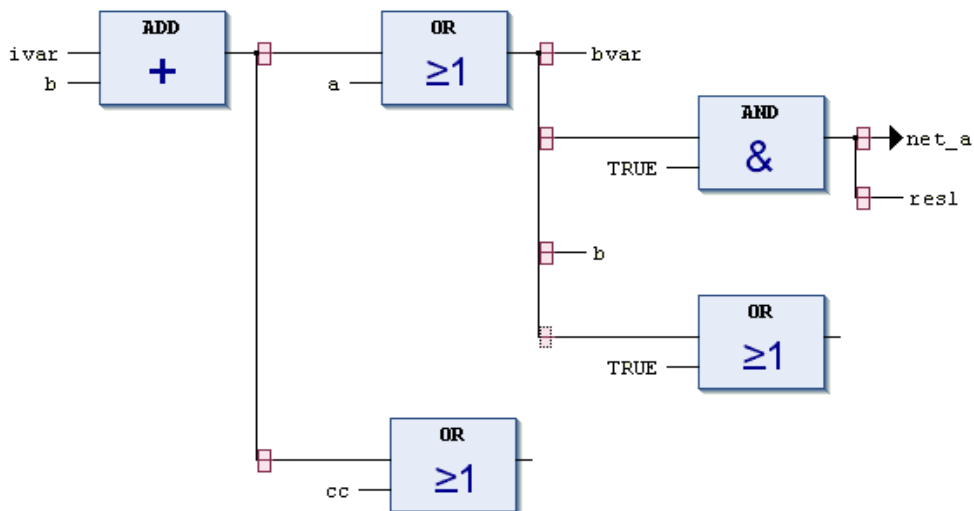
Figure 5-3 Branch label

■ Insert branch below: icon - ▮I , which adds a branch below the selected branch.

■ Command execution condition: A branch line is selected.

■ Insert branch above: icon - ▮I , which adds a branch above the selected branch. This command is executed only when a branch line is selected.

■ Command execution condition: A branch line is selected.

10 Jump and return

Two menu commands are provided: insert jump and insert return. Jump and Return are used to control the program execution sequence. In normal cases, programs are executed from top down and from left to right based on the network sequence. To add a jump or return element, drag and drop **Jump** or **Return** from the toolbox.

Like coils, the jump and return elements must be located on the rightmost side. Therefore, the rules for inserting jump and return elements are the same as those for inserting coils. For details, see the "insert coil" command.

■　Insert jump: icon - ; shortcut key: **Ctrl**+**L**, which inserts a jump element to jump to the specified label position.

　　The jump position is marked by a label in the network. That is, jump across networks is supported. Jump is executed only when the pre-jump input condition is met.

■　Insert return: icon - ⊲RET, which inserts a return element. When the input condition is met, the current POU executes return and returns results to the caller POU.

11 Negate

Icon - ; shortcut key: **Ctrl**+**N**, which negates the operation block input, operation block output, jump condition, return condition, contact value, or coil.

The negate command can be executed in the following two positions:

1)　Element negation: The main elements are contact and coil. A slash (/) is added to the contact and coil after negation.

2)　Line negation: The main elements are operation block input line, operation block output line, coil input line, jump input line, and return input line. A circle is added on the line.

Figure 5-4 shows the negation position.
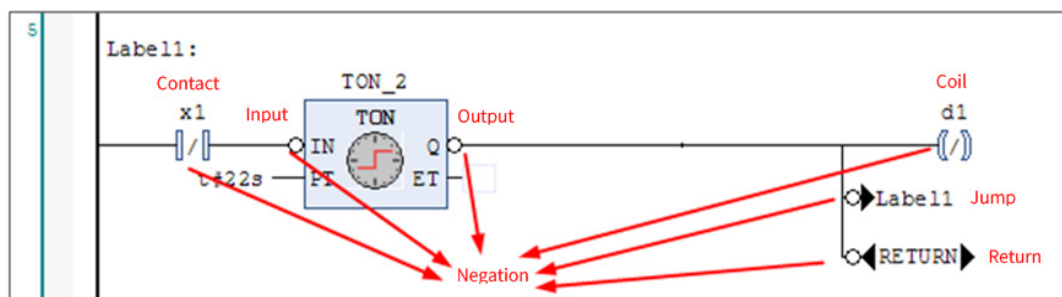
Figure 5-4　Negate operation

The negation status is switched back when the negate command is executed again.

12 Detect edge

Icon - ▨ ; shortcut key: **Ctrl**+**E**, which adds the edge trigger function to contacts, operation block input lines, coil input lines, jump element input lines, and return element input lines.

Rising edge detection is equivalent to the R_TRIG function block, and falling edge detection is equivalent to the F_TRIG function block.

The edge detection command can be executed in the following two positions:

1) Contact edge detection: Select a contact to run the edge detection command. The edge detection function is added to the contact. ⊣P⊢ indicates the rising edge, and ⊣N⊢ indicates the falling edge.

2) Add edge detection to line: The edge detection command of the execution block is applicable to the operation block input line, coil input line, jump element input line, and return element input line. The edge signal symbol is added to the line. The rising edge detection symbol is ▷ , and the falling edge detection symbol is ◁. The edge detection function is added only to input lines of the boolean type.

13 Set and reset

Icon - ⓢ ; shortcut key: **Ctrl**+**M**, which adds the set or reset output function. Set output is displayed as S, and reset output is displayed as R. The command can be executed multiple times and switches among set, reset, and normal output.

The set and reset commands can be executed in the following two positions:

1) Coil selection. This command sets or resets a coil. Set coil: ⊣(R) . Reset coil: ⊣(S)

2) Select the boolean-type output line (non-main output) of the operation block and configure the set or reset function, as shown in the following figure.
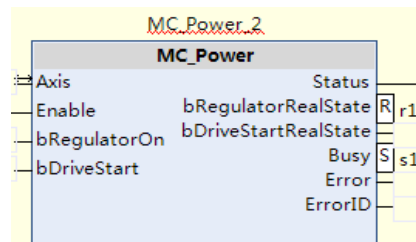
14 Set output connection

Icon - ⊨ ; shortcut key: **Ctrl**+**W**, which modifies the pin of the main output when the operation block contains multiple outputs. An operation block has only one main output, which is linked to subsequent elements. See Figure 5-5.

Select the output pin to be modified and run this command to modify the output connection.
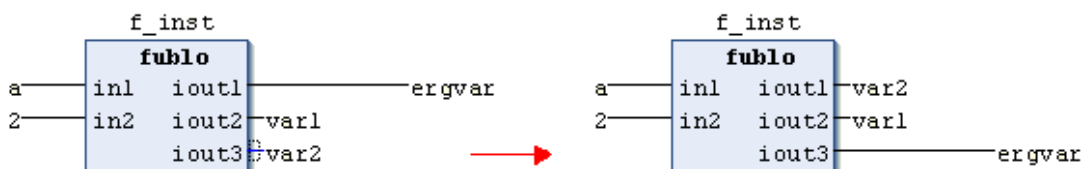
Figure 5-5 Output connection modification

15 Modify the input and output pin display

The following two menu commands are provided: update parameter and delete unused FB call parameter.

Update parameter: icon - ; shortcut key: **Ctrl**+**U**, which updates the input and output parameters of the selected operation block. When the input or output parameters of the operation block are changed, run the "update parameter" command to update the parameters.

Delete unused FB call parameter: icon - , which deletes the input and output pins of the unused operation block. That is, when the input or output of the operation block is "???" or empty, the input or output is not displayed.

16 Convert to LD language

Display as LD logic: shortcut key **Ctrl**+**2**. Convert FBD/IL to LD language: Because FBD and IL are no longer supported, use this command to convert FBD and IL in the LD language for old projects.

17 Jump to network

Jump to …: jumps to the specified network. Specify the target network number in the **Network number (1-2) or label:** text box.
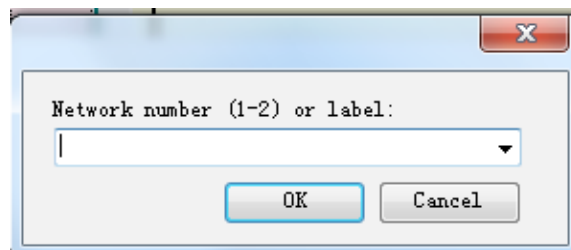
18 Edit operand comment

Edit operand comment: edits the comment of the selected operand.

Command execution conditions:

■   FBD/LD is selected, and **Show operand comment** is selected.

■   An operand string is selected.

Operand is a logical concept. Input variables, constants, and addresses are operands. Examples are operation block input variable, contact association variable, coil association variable, and operation block instance.

Select an operand string and run this command. The **Edit Comment** dialog box is displayed, as shown in the following figure. Edit the operand comment.
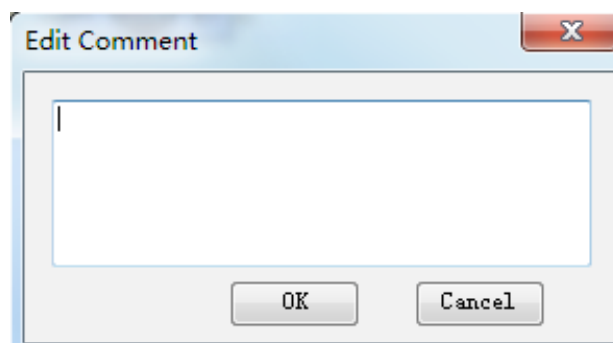
19 Switch parallel mode

Toggle Parallel Mode: switches the parallel mode of a parallel branch. The parallel mode is divided into the sequential parallel branch and the short-circuit-type parallel branch.

■ The sequential parallel bracket uses a single line. The output of a single branch is subjected to the OR operation to obtain the branch output result, as shown in the following figure. Branch results are obtained through OR.

Single bracket line
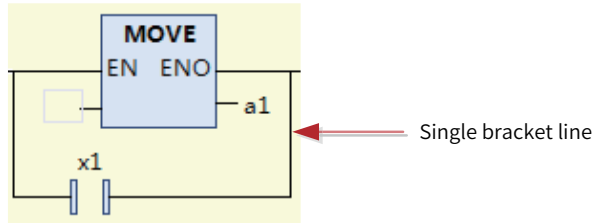
■ The short-circuit-type bracket uses double line. The branch output result must consider whether each branch includes a non-operation block.

The branch of the non-operation block is used as a condition. If a result of the branch is True, the branch with an operation block is not executed. It can be considered as an operation block of the contact short circuit type. As shown in the following figure, the first Move branch instruction is executed only when the results of the X1 and X2 branches are not TRUE.

The branch of the non-operation block must meet the following conditions:

1) The branch only includes a contact or operator block.

2) The contact does not include the edge signal.

3) The operator block is not of the EN/ENO type, and its input line does not contain the negation or edge signal.

Double bracket line

NOTE
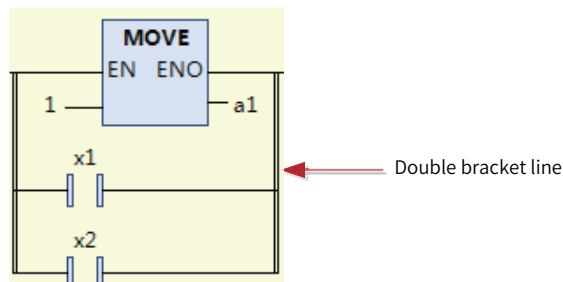
Branches of the short circuit type are not recommended for use considering the branch complexity.

5.3.7 Single-key Command

A single-key command enables fast editing by using a single-character shortcut key. The single-key command can be executed on a line or an element. Run a single-key command on a line to insert a serial element. A single-key command on an element is used to insert parallel elements or switch the element function.

To set a character for each command, choose **Options** > **FBD/LD** > **LD**.

Single-key commands on lines

■ Insert contact: The default single key is **C**.

■ Insert NC contact: The default single key is **/**.

■ Insert coil: The default single key is **Q**.

■ Insert reset coil: The default single key is **R**.

■ Insert set coil: The default single key is **S**.

■ Insert empty operation block: The default single key is **F**.

■ Insert empty EN/ENO operation block: The default single key is **E**.

Single-key commands on elements

■ Insert parallel contact: The default single key is **C**. The selected element can be a contact or operation block.

■ Insert empty parallel operation block: The default single key is **F**. The selected element can be a contact or operation block.

■ Insert empty parallel EN/ENO operation block: The default single key is **E**. The selected element can be a contact or operation block.

■ Insert coil: The default single key is **Q**. The selected element can be a coil, return element, or jump element.

■ Switch element negation: The default single key is **/**. Select a contact to switch between NO and NC. Select a coil to switch negation.

■ Switch element set/reset/edge signal: The default single key is space. Select a contact to switch among the rising edge signal, falling edge signal, and normal signal. Select a coil to switch among set, reset, and normal coil.

5.3.8 Parallel Line Connection

Select a line as the starting line and draw a line from the selected line to another line. Then, insert parallel contacts between the two lines. See the following figure.

Inserted parallel contact
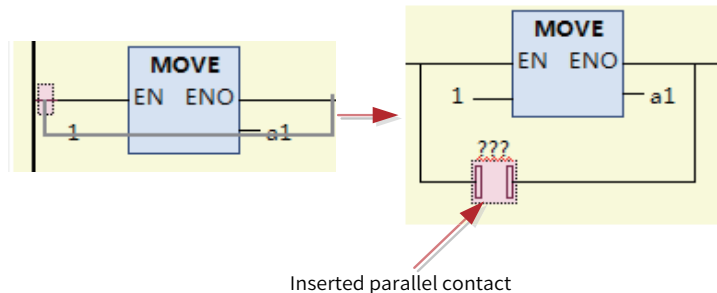
■ The starting point and end point must meet the conditions of parallel connection. That is, the starting point and end point cannot span the inside and outside of the parallel branch, cannot span branches, and cannot span from the input to the output of the multi-line operation block.

■ Horizontal lines and parallel branch brackets can be connected in parallel by drawing lines. The premise is that lines are available for selection.

■ You can drag the input and output pins of an operation block to exchange their positions. Therefore, the pin dragging area is close to the line drawing area (about 11 pixels) of the operation block pins. Lines cannot be drawn.

5.3.9 Drag and Drop

The LD supports the drag and drop operation on elements. You can drag the elements in the toolbox and drop them onto the network, and drag and drop elements on the LD interface or across interfaces.

When you drag and drop an element, the LD interface displays the available positions. The available positions are displayed in the following three modes:

■ Diamond ◇ : You can drag and drop an element onto the current position and insert it in serial mode (①).

■ Upper and lower triangles ◈ : You can insert a parallel element above or below the current element.

■ Upper and lower arrows ⬆⬇ : You can add a network in the upward or downward direction.

When you drag and drop an element onto the insertion position, each figure changes to green inside, for example, ◈ , indicating the insertion position (②).

The following figure shows the drag and drop process.
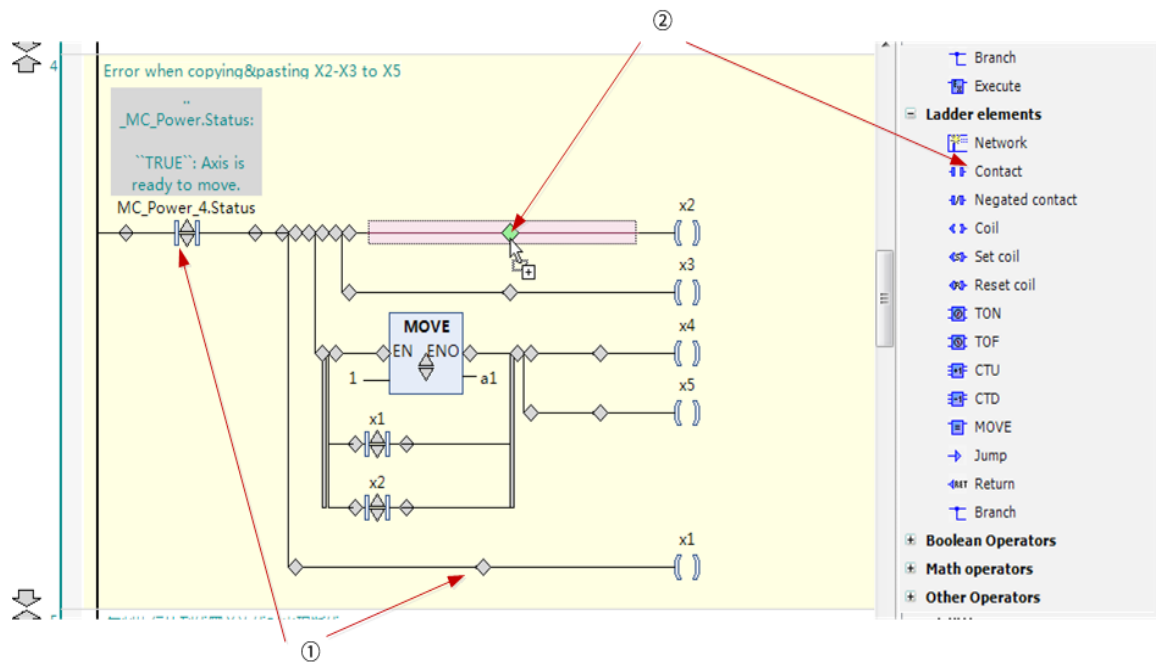
1 - Drag position. The triangle indicates the direction of parallel insretion, and the diamond indicates serial connection;
2 - After dragging, the color changes to green to indicate the insertion position

1 Drag and drop elements from the toolbox

You can drag and drop elements from the toolbox to the LD editor.

The toolbox provides general elements, LD elements, common boolean operators, math operators, other common operators, common function blocks, and POUs.

General elements and LD elements are frequently used elements, including network, operation block, execution block, contact, coil, and branch.

Boolean operators mainly include the AND and OR operators.

Math operators mainly include the frequently used ADD, SUB, MUL, DIV, GE, EQ, LE, and LT.

Other operators include either-or, choose-one, type conversion, and valuation.

Function blocks include the frequently used TON, TOF, R_TRIG, F_TRIG, and RS.

POUs include the programs, function blocks, functions, methods, and actions defined in the current project. A maximum of 200 POUs can be displayed. If this limit is exceeded, POUs are not displayed, to avoid content disorder.

When you drag and drop elements, the available positions are displayed. Observe the following rules:

■ You can drag and drop contacts onto contacts and operation blocks (including execution blocks) for parallel connection, and onto lines for serial connection.

■ You can drag and drop operation blocks onto contacts and operation blocks (including execution blocks) for parallel connection, and onto lines for serial connection.

■ You can drag and drop coils onto non-parallel branches and input lines of non-multi-line operation blocks for serial connection. You can also drag and drop coils above or below other coils, return elements, and jump elements.

2 Drag and drop elements on the edit page

On the LD interface, you can drag and drop the selected element from one position to another. You can drag and drop elements within the current edit page or onto another LD edit page.

You can select one or more elements for the drag and drop operation. For details, see the section about element selection.

The drag and drop operation is divided into normal drag and drop and copy-type drag and drop (press **Ctrl** for drag and drop). During normal drag and drop, after the selected element is dragged and dropped, it is deleted from the original position. During copy-type drag and drop, after the selected element is dragged and dropped, it is still retained in the original position.

The drag and drop function is implemented in the standard manner.

The drag and drop rules for one or more selected elements are the same as those of the paste operation by the standard edit command.

5.3.10 Graphic Display Tool

The LD graphic display tools are used to control the LD display mode, including the selection tool, drag and drop tool, magnifier tool, and zooming tool. By default, the LD adopts the selection tool. The graphic display tools are displayed in the lower-right corner of the LD interface, as shown in the following figure.
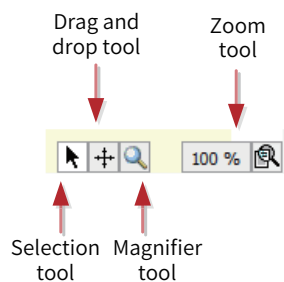
- Selection tool

  The selection tool is the default displayed tool. In selection tool mode, the cursor is displayed as ⬚ . You can select elements for editing.

- Drag and drop tool

  In drag and drop tool mode, the cursor is displayed as ⬚ . You can perform the drag and drop operation in areas.

- Magnifier tool

  In magnifier tool mode, the cursor is displayed as ＋ . Content is magnified with the cursor at the center. See the following figure.
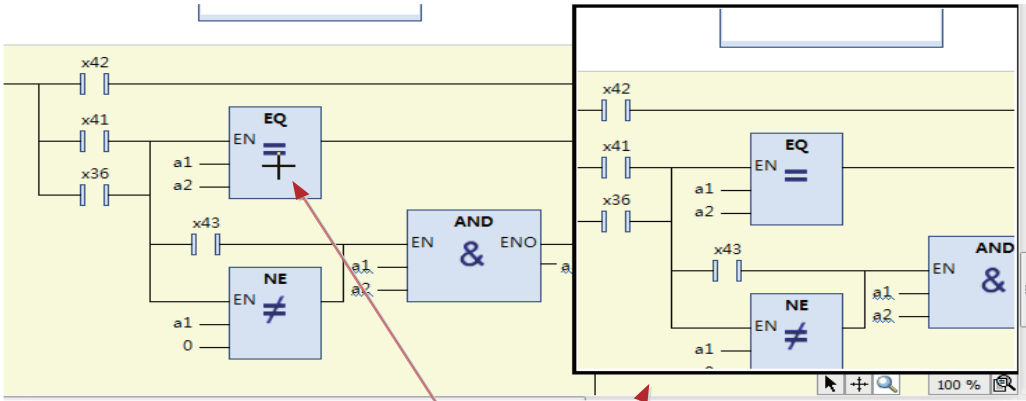
Zoom out with the mouse at the center

■ Zooming tool

The zooming tool displays the zoom ratio of the current interface and allows you to set the zoom ratio, as shown in the following figure.
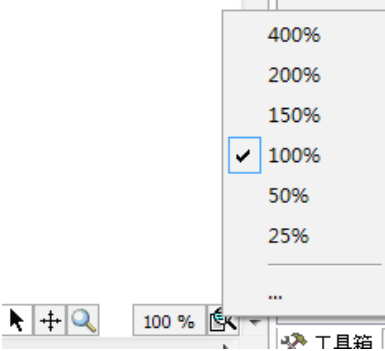
Click **...** The **Zoom Scale** dialog box is displayed. Enter a zoom ratio, as shown in the following figure.
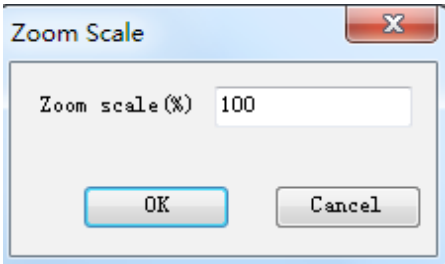
5.3.11 LD Debugging

The LD provides powerful debugging functions. In addition to the existing monitoring table, the LD also provides online monitoring, operand writing, mandatory value writing, breakpoint, and single step debugging.

1 Monitoring

In online mode, the LD interface displays the execution results of lines, elements, and operand variables in specific forms. See the following figure.

Display forms of lines, variables, and elements during monitoring
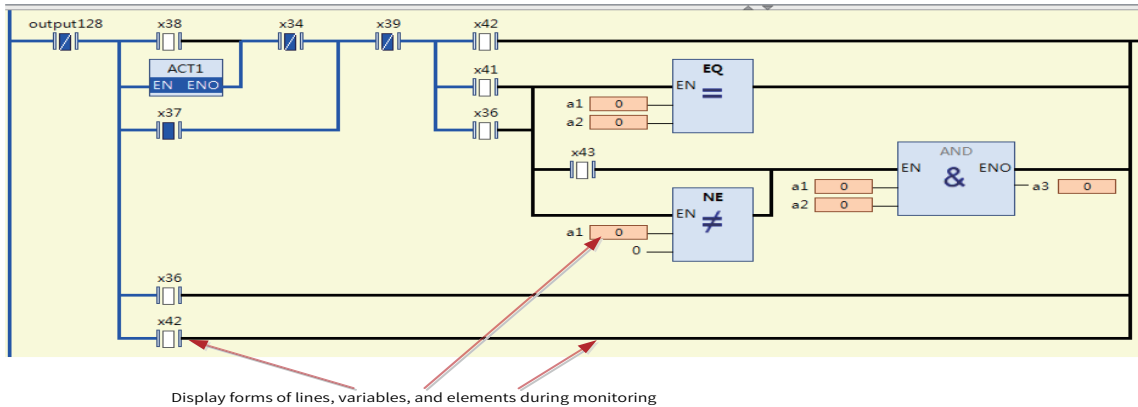
■ Line monitoring

1) Boolean lines are displayed in blue in the bold form in the conducting state (the value is TRUE); otherwise, they are displayed in black in the bold form.

2) Non-boolean lines (operation block input, output integer variable, time-type variable, and floating-point variable) are displayed as fine lines. When the value is 0, they are displayed as black fine lines. When the value is not 0, they are displayed as blue fine lines.

■ Element monitoring

1) When the contact is conducted, the NO contact is displayed as ▐█▐ , or the NC contact is displayed as ▐▨▐ . When the contact is not conducted, the NO contact is displayed as ▐▢▐ , or the NC contact is displayed as ▐▨▐ .

2) When the coil is conducted, the normal coil is displayed as (█) , or the negated coil is displayed as (▨) . When the coil is not conducted, the normal coil is displayed as (▢) , or the negated coil is displayed as (▨) .

3) The logic of the EN/ENO operation block is executed only when EN is TRUE. To allow you to understand the execution status of the EN/ENO operation block (whether it is enabled), we differentiate the text of the operation block type. If the operation block is executed (EN is TRUE), the operation block type is displayed in black text. If the operation block is not executed, the operation block type is displayed in gray text (the block is disabled), as shown in the following figure.
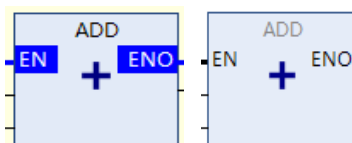
Figure 5-6 Operation block executed Operation block not executed

■ Variable monitoring

1) Monitored variables are displayed in different widths, depending on the specific type, to reduce space usage. For variable-length elements such as strings and enumerated elements (the enumeration name is displayed), the default length is 12 characters. If the displayed content is incomplete, ... is displayed, and the complete content is displayed in a prompt. For fixed-length elements such as integers and floating point numbers, content is displayed based on the maximum length.

2) You can drag and drop monitored variables to the monitored variable list.

3) To change the variable display mode, choose **Debug** > **Display Mode**.

> **NOTE**
>
> Functions and methods are executed instantly, resulting in only temporary data. Therefore, functions and methods cannot be monitored directly after login. To monitor functions and methods, you need to add breakpoints to the functions and methods to interrupt execution before monitoring, as shown in the following figure.

2 Mandatory value writing

You can write preparatory values to contacts, coils, and variables of the LD. Then, run the "write value" or "enforce value" command in the **Debug** menu to write or enforce values to variables. Before writing or enforcing values, you need to write preparatory values, as shown in the following figure.

Write preparatory values inside
<> for contacts and variables
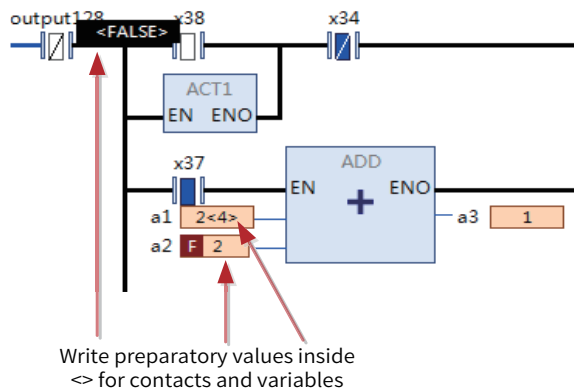
■ For contacts, coils, and boolean variables, double-click the element or variable value position to switch between the TRUE and FALSE preparatory values. For example, you can double-click in the middle of a contact or coil to switch the preparatory value.

■ Double-click the value position of a non-boolean variable. The **Prepare Value** dialog box is displayed. Enter a preparatory value, as shown in the following figure.
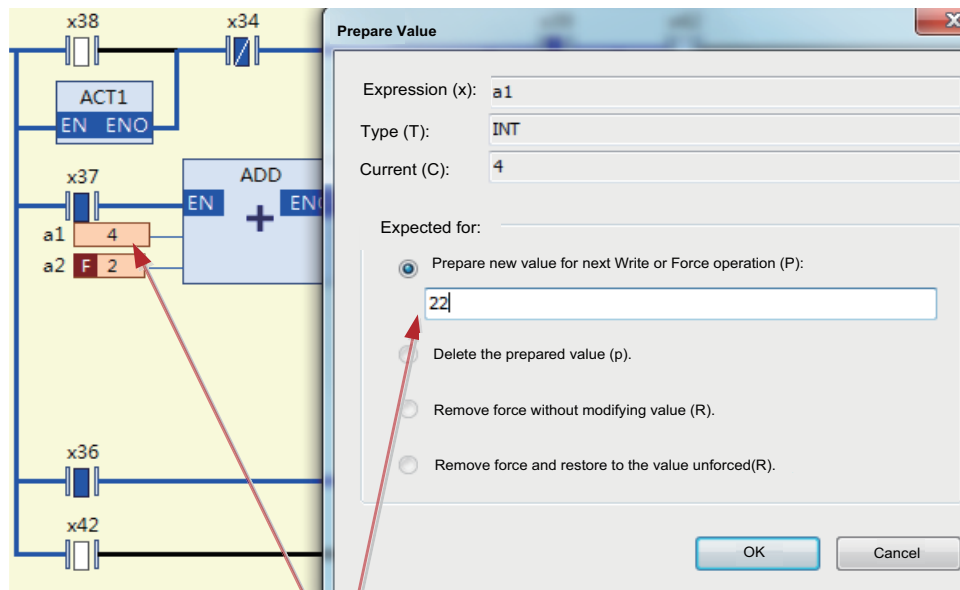
Write INT a1 into the Prepare value

■ After a mandatory value is written, the ▣ label is added before the value to indicate it is a mandatory value.

■ To release a mandatory value, choose **Online** > **Release value**.

3 Breakpoint

The LD supports the breakpoint function. After a breakpoint is added, program execution automatically stops at the breakpoint, and you can debug the program. Operations such as jump in, skip, jump out, and run to cursor are supported.

After a breakpoint is added, the breakpoint position (element) is marked by a rectangular box in light red. When program execution reaches the breakpoint, the breakpoint position is marked by a rectangular box in yellow. If a breakpoint exists in the network, a circle is displayed in the network decoration area, as shown in the following figure.
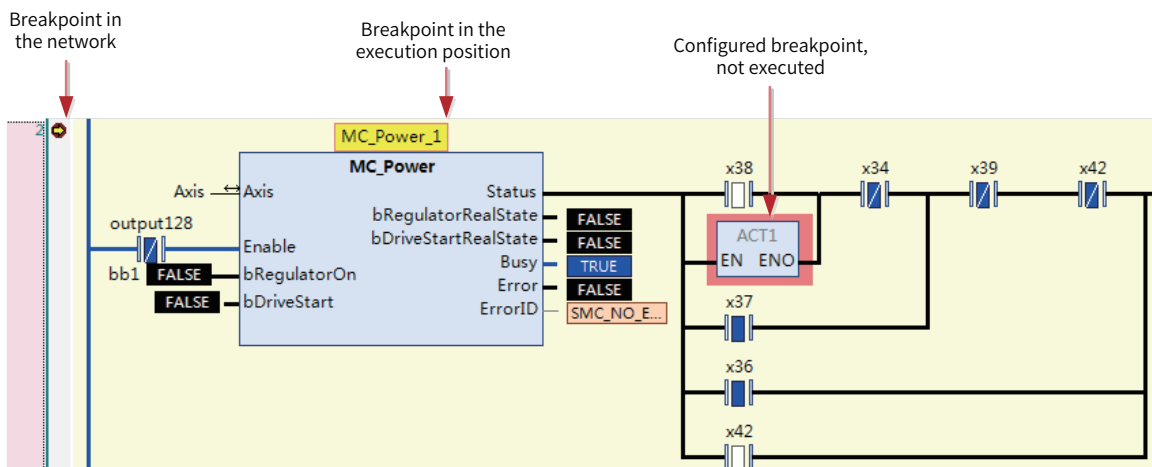
The LD is graphical and a breakpoint can be added only in a position with a logical statement. Logical statements exist only in limited areas of the LD for optimized performance. That is, breakpoints can be added only in limited areas. For example, breakpoints cannot be added in the contact position or non-EN/ENO operator block position.

Breakpoints exist in places with possible variable value change, program branches, POU call position, and places where output variables are assigned values. Choose **View** > **Breakpoints** to open the **Breakpoints** dialog box and view all possible breakpoint positions.

Breakpoints can be added in the following positions:

- Network start position, which is the position of the first possible breakpoint in the network. When a breakpoint is added to a network, it is added to the first breakpoint position.

- Operation blocks not including the EN/ENO operator, such as FB, action, program call, and execution block.

- Coil, return, and jump element positions.

5.3.12 LD Data Update

For InoProShop V1.3.0 and earlier, such as InoProShop V0.0.9.10, InoProShop V1.1.0, InoProShop V1.2.0, InoProShop V1.2.60.0, and InoProShop V1.2.70.1, the accessed LD data must be updated before you can use the functions of the optimized LD version.

LD data can be updated in the following two ways:

- In the **Project Version Information** dialog box that is displayed when a project is opened, click the **LD/FBD** tab. Select all the update marks shown in the following figure, and click **OK**.

- Choose **Project** > **Project Version Information**. In the **Project Version Information** dialog box, click the **LD/FBD** tab. Select all the update marks shown in the following figure, and click **OK**.

If the LD data is not updated, update description is displayed in the first network,, indicating that the data is not updated, the editing may not meet the expecations, and you need to open the project or choose **Project** > **Project Version Information** > **[LD/FBD]** and update the prorject LD data .
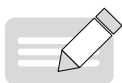
NOTE

LDs must be updated before use.

Memo No. _____

Date   /   /

Chapter 6 Inovance Instruction Library

6 Inovance Instruction Library

6.1 Cheat Sheet of Instructions

6.1.1 Instructions

In a PLC system, a command or a combination of commands for the CPU to finish an operation or implement a function is called an instruction. An instruction set is called the instruction system.

6.1.2 Instruction Classification

CoDeSys instructions can be implemented in function mode or in function block mode. You do not need to declare (instantiate) instructions implemented in function mode. You need to instantiate instructions implemented in function block mode. Based on 3S basic instructions (see "Appendix"), the AM600 provides diverse instructions and supports high-rate I/O, CANopen, and EtherCAT remote counting. Therefore, this section only describes motion control instructions. For the overview of 3S basic instructions, see the "Appendix".

6.1.3 Cheat Sheet of Motion Control Instructions

Motion control instructions include SM3_Basic, SM3_CNC, SM3_ETC, CmpHCBasic, IoDrvCANopenAxis, and CmpHSIO instruction libraries. As SM3_Basic, SM3_CNC, and SM3_ETC instruction libraries are provided by 3S and their functions are detailed in software manuals, the following only lists instructions. This section details CmpHCBasic, IoDrvCANopenAxis, and CmpHSIO instruction libraries provided by Inovance.

1 SM3_Basic

SM3_Basic instructions can be classified into basic drive instructions, file operation instructions, and PLCopen instructions.

Basic drive instructions

Type	Description	Name	Category
Brake control	Sets the brake status.	SMC3_BrakeControl	Function block
	Obtains the brake status.	SMC3_BrakeStatus	Function block
Configuration instruction	Saves the multi-turn encoder axis position (after restart).	SMC3_PersistPosition	Function block
	Saves the single-turn encoder axis position (after restart).	SMC3_PersistPositionSingleturn	Function block
	Retains the logical axis position (after restart).	SMC3_PersistPositionLogical	Function block
	Sets the control mode.	SMC_SetControllerMode	Function block

Type	Description	Name	Category
Diagnosis instruction	Checks the axis communication status.	SMC_CheckAxisCommunication	Function block
	Obtains the maximum axis acceleration or deceleration.	SMC_GetMaxSetAccDec	Function block
	Obtains the maximum axis speed.	SMC_GetMaxSetVelocity	Function block
	Obtains the tracking error.	SMC_GetTrackingError	Function block
	Monitors the axis position.	SMC_InPosition	Function block
	Measures the rotary axis rotation distance.	SMC_MeasureDistance	Function block
Direct axis control instruction	Monitors the axis position limit.	SMC_CheckLimits	Function block
	Writes the axis position without detection.	SMC_FollowPosition	Function block
	Writes the axis position and speed without detection.	SMC_FollowPositionVelocity	Function block
	Writes axis parameters without detection.	SMC_FollowSetValues	Function block
	Writes the axis speed without detection.	SMC_FollowVelocity	Function block
Error processing instruction	Deletes previous function block errors.	MC_ClearFBError	Function block
	Reads previous function block errors.	SMC_ReadFBError	Function block
	Formats error strings.	SMC_ErrorString	Function block
Homing instruction	Returns to home.	SMC_Homing	Function block
Zoom instruction	Changes the axis zoom ratio and axis type.	SMC_ChangeGearingRatio	Function block

2 File Operation Instructions

Type	Description	Name	Category
Cam instruction	Loads cam table data from files.	SMC_ReadCAM	Function block
	Writes cam table data in files.	SMC_WriteCAM	Function block
Diagnosis instruction	Logs axis diagnosis.	SMC_AxisDiagnosticLog	Function block

3 PLCopen Instructions

Type	Description	Name	Category
Additional instruction	Compensate the backlash.	SMC_BacklashCompensation	Function block
	Reads the current axis position.	SMC_ReadSetPosition	Function block
	Sets the torque.	SMC_SetTorque	Function block

Type	Description	Name	Category
Single-axis control instruction	Reads the current axis status.	MC_ReadStatus	Function block
	Reads motion control parameters.	MC_ReadParameter	Function block
	Resets the axis.	MC_Reset	Function block
	Stops axis motion (which cannot be interrupted).	MC_Stop	Function block
	Sets the axis speed at different time points.	MC_VelocityProfile	Function block
	Writes motion control BOOL parameters.	MC_WriteBoolParameter	Function block
	Writes motion control parameters.	MC_WriteParameter	Function block
	Ends the correlation between the function block and event.	MC_AbortTrigger	Function block
	Enables and disables digital cams.	MC_DigitalCamSwitch	Function block
	Reads the current torque.	MC_ReadActualTorque	Function block
	Reads the current speed.	MC_ReadActualVelocity	Function block
	Sets the axis position.	MC_SetPosition	Function block
	Records an axis position through event trigger.	MC_TouchProbe	Function block
	Moves the axis to a specified absolution position and keeps it moving at a specific speed.	SMC_MoveContinuousAbsolute	Function block
	Moves the axis in relative position mode to a specified position and keeps it moving at a specific speed.	SMC_MoveContinuousRelative	Function block
	Indicates jog control.	MC_Jog	Function block
	Controls motion in single step mode (the motion distance can be controlled).	SMC_Inch	Function block

4 SM3_CNC

SM3_CNC instructions can be classified into file operation instructions, CNC motion control instructions, and CNC conversion instructions.

File Operation Instructions

Type	Description	Name	Category
File Operation Instructions	Reads NC files.	SMC_ReadNCFile	Function block
	Reads OutQueue files.	SMC_ReadNCQueue	Function block

5 CNC Motion Control Instructions

Type	Description	Name	Category
Coordinates conversion instruction	Converts coordinates in cuboid mode.	SMC_DetermineCuboidBearing	Function block
	Converts three-dimensional coordinates.	SMC_CoordinateTransformation3D	Function block
	Converts three-dimensional coordinates inversely.	SMC_InvCoordinateTransformation3D	Function block
	Calculates coordinate vectors in the new coordinate system.	SMC_TeachCoordinateSystem	Function block
	Calculates the RPY angle of unit vector in the new coordinate system (corresponding to the old coordinate system).	SMC_UnitVectorToRPY	Function block
	Sets the motion path corresponding to rotation around the Z axis.	SMC_RotateQueue2D	Function block
	Scales the motion path.	SMC_ScaleQueue3D	Function block
	Converts the motion path data based on vectors.	SMC_TranslateQueue3D	Function block
Direct axis control instruction	Controls the axis by speed.	SMC_ControlAxisByVel	Function block
	Controls the axis by position.	SMC_ControlAxisByPos	Function block
	Controls the axis by position and speed.	SMC_ControlAxisByPosVel	Function block
G-code display instruction	Converts G-code data bit strings.	SMC_GCodeViewer	Function block
OutQueue instruction	Adds GEOINFO data to OutQueue.	SMC_AppendObj	Function block
	Deletes GEOINFO data from OutQueue.	SMC_DeleteObj	Function block
	Obtains the count of GEOINFO data in OutQueue.	SMC_GetCount	Function block
	Obtains GEOINFO data in OutQueue.	SMC_GetObj	Function block
	Finds GEOINFO data from the end of OutQueue.	SMC_GetObjFromEnd	Function block
	Initializes OutQueue.	SMC_OutQueueInit	Function block
	Sets the OutQueue data capacity.	SMC_SetQueueCapacity	Function block

Type	Description	Name	Category
Motion control instruction	Removes loops.	SMC_AvoidLoop	Function block
	Checks whether the paths are out of a rectangle and outputs paths within the rectangle..	SMC_CheckForLimits	Function block
	Checks path speeds and outputs paths within the range.	SMC_CheckVelocities	Function block
	Modifies the path speed and acceleration/deceleration to keep the speed within the preset range.	SMC_ExtendedVelocityChecks	Function block
	Interpolate.	SMC_Interpolator	Function block
	Interpolates (reverse interpolation supported).	SMC_Interpolator2Dir	Function block
	Supports low-priority reverse interpolation.	SMC_Interpolator2Dir_SlowTask	Function block
	Dynamically adjusts the speed and acceleration/deceleration to prevent them from exceeding the maximum.	SMC_LimitDynamics	Function block
	Limits the circular speed.	SMC_LimitCircularVelocity	Function block
	Parses the G-code.	SMC_NCDecoder	Function block
	Indicates the curve analyzer.	SMC_ObjectSplitter	Function block
	Recomputes the slopes of A axis, B axis, and C axis respectively.	SMC_RecomputeABCSlopes	Function block
	Smooths paths.	SMC_RoundPath	Function block
	Indicates the path segment analyzer.	SMC_SegmentAnalyzer	Function block
	Smooths additional axis paths.	SMC_SmoothAddAxes	Function block
	Smooth paths.	SMC_SmoothPath	Function block
	Processes the path offset.	SMC_ToolCorr	Function block
	Supports cam hybrid interpolation.	SMC_XInterpolator	Function block
	Obtains M function parameters.	SMC_GetMParameters	Function block
	Reads parameters earlier when the M function is enabled through interpolation.	SMC_PreAcknowledgeMFunction	Function block
BlockSearch instruction	Shortens the path.	SMC_BlockSearch	Function block
	Saves the current path position.	SMC_BlockSearchSavePos	Function block

6 CNC Conversion Instructions

Type	Description	Name	Category
Auxiliary instruction	Calculates the direction based on the three-dimensional vector coordinates.	SMC_CalcDirectionFromVector	Function block

Type	Description	Name	Category
Gantry system instruction	Calculates the tool center based on the axis position.	SMC_TRAFOF_5Axes	Function block
	Converts the two-dimensional coordinates into the G-code coordinates.	SMC_TRAFOF_Gantry2	Function block
	Converts the two-dimensional coordinates into the G-code coordinates in linear mode.	SMC_TRAFOF_Gantry2Tool1	Function block
	Converts the two-dimensional coordinates into the G-code coordinates in rectangular mode.	SMC_TRAFOF_Gantry2Tool2	Function block
	Converts the three-dimensional coordinates into the G-code coordinates.	SMC_TRAFOF_Gantry3	Function block
	Converts the two-dimensional coordinates with rotary axes into the G-code coordinates.	SMC_TRAFOF_GantryCutter2	Function block
	Converts the three-dimensional coordinates with rotary axes into the G-code coordinates.	SMC_TRAFOF_GantryCutter3	Function block
	Converts the T-shaped two-dimensional coordinates into the G-code coordinates.	SMC_TRAFOF_GantryT2	Function block
	Converts the H-shaped two-dimensional coordinates into the G-code coordinates.	SMC_TRAFOF_GantryH2	Function block
	Converts the G-code coordinates into the two-dimensional reverse coordinates.	SMC_TRAFOV_Gantry2	Function block
	Converts the G-code coordinates into the three-dimensional reverse coordinates.	SMC_TRAFOV_Gantry3	Function block
	Converts the G-code coordinates into the two-dimensional reverse coordinates with rotary axes.	SMC_TRAFOV_GantryCutter2	Function block
	Converts the G-code coordinates into the three-dimensional reverse coordinates with rotary axes.	SMC_TRAFOV_GantryCutter3	Function block
	Converts the G-code coordinates into the H-shaped two-dimensional reverse coordinates.	SMC_TRAFOV_GantryH2	Function block
	Calculates the position of the tool center relative to an axis.	SMC_TRAFO_5Axes	Function block
	Converts the G-code coordinates into the two-dimensional coordinates.	SMC_TRAFO_Gantry2	Function block
	Converts the G-code coordinates into the two-dimensional coordinates in linear mode.	SMC_TRAFO_Gantry2Tool1	Function block
	Converts the G-code coordinates into the two-dimensional coordinates in rectangular mode.	SMC_TRAFO_Gantry2Tool2	Function block
	Converts the G-code coordinates into the three-dimensional coordinates.	SMC_TRAFO_Gantry3	Function block
	Converts the G-code coordinates with rotary axes into the two-dimensional coordinates.	SMC_TRAFO_GantryCutter2	Function block
	Converts the G-code coordinates with rotary axes into the three-dimensional coordinates.	SMC_TRAFO_GantryCutter3	Function block
	Converts the G-code coordinates into the H-shaped two-dimensional coordinates.	SMC_TRAFO_GantryH2	Function block
	Converts the G-code coordinates into the T-shaped two-dimensional coordinates.	SMC_TRAFO_GantryT2	Function block

Type	Description	Name	Category
Parallel robot system instruction	Indicates positive conversion of a bipod arm.	SMC_TRAFOF_Bipod_Arm	Function block
	Indicates positive conversion of a tripod.	SMC_TRAFOF_Tripod	Function block
	Indicates positive conversion of a tripod arm.	SMC_TRAFOF_Tripod_Arm	Function block
	Indicates inverse conversion of a bipod arm.	SMC_TRAFO_Bipod_Arm	Function block
	Indicates inverse conversion of a tripod.	SMC_TRAFO_Tripod	Function block
	Indicates inverse conversion of a tripod arm.	SMC_TRAFO_Tripod_Arm	Function block
Robot kinematics conversion instruction	Indicates motion conversion of the articulated 6-DOF robot world coordinates into the 6-axis position coordinates.	SMC_TrafoF_ArticulatedRobot_6DOF	Function block
	Indicates inverse motion conversion of a 6-DOF robot.	SMC_Trafo_ArticulatedRobot_6DOF	Function block
Articulated robot system instruction	Indicates positive polar conversion.	SMC_TRAFOF_Polar	Function block
	Indicates positive conversion of a 2-axis Selective Compliance Assembly Robot Arm (SCARA).	SMC_TRAFOF_Scara2	Function block
	Indicates positive conversion of a 3-axis SCARA.	SMC_TRAFOF_Scara3	Function block
	Indicates inverse polar conversion.	SMC_TRAFO_Polar	Function block
	Indicates inverse conversion of a 2-axis SCARA.	SMC_TRAFO_Scara2	Function block
	Indicates inverse conversion of a 3-axis SCARA.	SMC_TRAFO_Scara3	Function block

7 ETC Instructions

Type	Description	Name	Category
EtherCAT parameter read/write instruction	Reads values corresponding to the slave index and subindex.	SMC3_ETC_ReadParameter_CoE	Function block
	Writes values corresponding to the slave index and subindex.	SMC3_ETC_WriteParameter_CoE	Function block

8 CmpHCBasic

Type	Description	Name	Category
Single-axis control	Indicates enhanced jog control.	MC_Jog_HC	Function block
EtherCAT drive reset	Resets single-drive communication and axes in enhanced mode.	MC_ResetDrive	Function block
EtherCAT slave reset	Resets single-slave communication.	MC_ResetRemoteModule	Function block
Position retention	Powers on/off and retains the position of the absolute value motor drive.	MC_PersistPosition	Function block

9 IoDrvCANopenAxis

Type	Description	Name	Category
Single-axis control	Enables the motor.	MC_Power_CO	Function block
	Positions the axis absolutely.	MC_MoveAbsolute_CO	Function block
	Positions the axis relatively.	MC_MoveRelative_CO	Function block
	Indicates the speed mode.	MC_MoveVelocity_CO	Function block
	Returns to home.	MC_Home_CO	Function block
	Stops the axis, which cannot be interrupted by other instructions.	MC_Stop_CO	Function block
	Stops the axis, which can be interrupted by other instructions.	MC_Halt_CO	Function block
	Resets the axis.	MC_Reset_CO	Function block
	Indicates jog control.	MC_Jog_CO	Function block
	Reads the current state machine.	MC_ReadStatus_CO	Function block
Parameter reading and writing	Reads slave object dictionary values.	MC_ReadParameter_CO	Function block
	Writes slave object dictionary values.	MC_WriteParameter_CO	Function block

10 CmpHSIO

Type	Description	Name	Category
High-speed counting	Enables the counter.	HC_Counter	Function block
	Sets coincident output.	HC_SetCompare	Function block
	Writes preset values.	HC_PresetValue	Function block
	Enables external interrupts and coincident interrupts.	HC_EnableInterrupt	Function block
	Reads latch values.	HC_TouchProbe	Function block
	Reads the measured pulse width.	HC_MeasurePulseWidth	Function block
	Samples counters, counting the number of counters within a period of time.	HC_Sample	Function block
	Reads parameters.	HC_ReadBoolParameter	Function block
	Writes parameters.	HC_WriteBoolParameter	Function block
	Sets up to 100 comparison values. "Done" indicates that signals are output, and "Value" indicates the index number currently output.	HC_SetCompareM	Function block
	Sets ring counting.	HC_SetRing	Function block
	Resets the port for coincident output (set in the programming software). The port for direct hardware output (ImRefresh and ImRefreshCycle of HC_SetCompare) is invalid.	HC_ResetCmpOutput	Function block
	Writes parameters.	HC_WriteInterruptParameter	Function block

Type	Description	Name	Category
High-speed axis	Indicates jog control.	MC_Jog_P	Function block
	Indicates homing motion.	MC_Home_P	Function block
	Indicates absolute motion.	MC_MoveAbsolute_P	Function block
	Indicates relative motion.	MC_MoveRelative_P	Function block
	Moves the axis in speed mode.	MC_MoveVelocity_P	Function block
	Stop	MC_Stop_P	Function block
	Resets the axis.	MC_Reset_P	Function block
	Enables the axis.	MC_Power_P	Function block
	Reads parameters.	MC_ReadParameter_P	Function block
	Writes parameters.	MC_WriteParameter_P	Function block
	Sets a logical address.	MC_SetPosition_P	Function block
	Reads the axis status.	MC_ReadStatus_P	Function block

6.2 High-speed I/O

Hardware feature: 16 input ports are available, supporting 8-channel A/B-phase inputs. The first eight ports can serve as interrupt ports. The maximum input frequency is 200 kHz. Eight output ports are available, supporting 4-channel outputs in pulse plus direction mode or CW/CCW mode. The first two outputs support homing, hardware limit, and speed/position switchover functions.

Function overview: The local high-speed I/O port has counting, sampling, frequency measurement, comparison output, touch probe, interrupt, pulse axis control, and other functions. The maximum frequency of sampling and output is 200 kHz. The maximum frequency of high-speed interrupts is 3 kHz.

Library name: CmpHSIO (applicable to V0.0.0.6 and later versions)

6.2.1 High-speed Counting

1 Function Blocks

Table 6-1 Input resource allocation table

Function	Input Resources X0-Xf
Single-phase	0 1 2 3 4 5 6 7
A/B-phase	0 1 2 3 4 5 6 7 8 9 a b c d e f
Frequency and rotational speed measurement	0 1 2 3
Pulse width	8 9 a b
Touch probe	8 9 a b
Sampling	8 9 a b

■ HC_Counter

Enable the counter.

Figure 6-1 Diagram of the HC_Counter function block

Table 6-2 Table of HC_Counter function block I/O parameters

Parameter Name	Parameter Type	I/O Type	Description
Counter	COUNTER_REF	In	Specifies the counter number [0..7].
Enable	Bool	In	Indicates the enabling bit (1: Enable the counter and start counting; 0: Stop counting).
Direction	Bool	In	0: Up counting; 1: Down counting It is applicable only to the single-phase input or clock mode. Flow changes are valid only in this case.
CounterValue	Dint	Out	Indicates the current value of the counter.
Frequency	UDINT	Out	Indicates the measured pulse frequency, in the unit of Hz.
RPM	REAL	Out	Indicates revolutions per minute, in the unit of r/min.
Valid	Bool	Out	1: Valid; 0: Invalid
Error	Bool	Out	Indicates the function block error flag.
ErrorID	Uint	Out	Indicates the error code.

Table 6-3 Allocation table of counter and counting port resources

Counting Mode	Port Counter	X0	X1	X2	X3	X4	X5	X6	X7	X8	X9	Xa	Xb	Xc	Xd	Xe	Xf
AB A/B-phase counting	Counter0	A	B							T							
	Counter1			A	B						T						
	Counter2 (△)					A	B						T				
	Counter3 (△)							A	B					T			
	Counter4 (△)									A	B						
	Counter5 (△)											A	B				
	Counter6 (△)													A	B		
	Counter7 (△)															A	B

Counting Mode	Port Counter	X0	X1	X2	X3	X4	X5	X6	X7	X8	X9	Xa	Xb	Xc	Xd	Xe	Xf
A-phase counting	Counter0	A															
	Counter1		A														
	Counter2			A													
	Counter3				A												
	Counter4 (△)					A											
	Counter5 (△)						A										
	Counter6 (△)							A									
	Counter7 (△)								A								

1) (△): Calculates the FrequencyValue and RotationRateValue by using the software.

2) When Counter0 or Counter1 counts in AB-phase mode, only the A-phase frequency is displayed.

3) T indicates externally triggered signals.

4) The first four counters are different from the last four counters:

Positive counting: When the count reaches 2147483647, the first four counters stop, while the last four counters continue counting from -2147483648.

Negative counting: When the count reaches -2147483648, the first four counters stop, while the last four counters continue counting from 2147483647.

5) The following describes differences between linear counting and ring counting:

Linear counting: (DownLimitValue, UpLimitValue)

Ring counting: [iRingDownValue, iRingUpValue]

Counters 0 to 3: The linear counting range is (-2147483648, 2147483647), -2147483648 and 2147483647 excluded.

The ring counting range is [-2147483648, 2147483647], -2147483648 and 2147483647 included.

Counters 4 to 7: The linear counting range is (-2147483648, 2147483647), -2147483648 and 2147483647 excluded.

6) Counters 4 to 7: When the count is close to the boundary value, the system may not generate a counter overflow alarm because of the scan cycle.

7) When a program is downloaded, the counter is not zeroed.

8) The HC_Counter function block and HC_MeasurePulseWidth function block cannot be used simultaneously. Apply either of them to recommended programs.

Note: The HC_Sample function block runs only after the HC_Counter function block is enabled.

■ HC_SetCompare

When setting coincident output, call HC_EnableInterrupt to enable interrupts.

Figure 6-2 Diagram of the HC_SetCompare function block

Table 6-4 Table of HC_SetCompare function block I/O parameters

Parameter Name	Parameter Type	I/O Type	Description
Counter	COUNTER_REF	In	Specifies the counter number [0..7].
Execute	Bool	In	Indicates the enabling bit, triggered by rising edges.
CompareValue	Dint	In	Indicates the preset comparison value of the counter. The value ranges from -2147483648 to 2147483647, -2147483648 and 2147483647 excluded.
ImRefresh	Bool		0: Configures the output port in the programming software. 1: Indicates direct hardware output. Counter 0: Y0 Counter 1: Y1 ... Counter 7: Y7
ImRefreshCycle	Uint		The output time ranges from 0 to 30000, in the unit of 100 µs. The maximum output time is 3,000 ms. For example, the value 10000 indicates that the output time is 1,000 ms.
Done	Bool	Out	Indicates the flag of successful function block initialization.
Error	Bool	Out	Indicates the function block error flag.
ErrorID	Uint	Out	Indicates the error code.

Note: Y0 to Y7 can be controlled in ImRefresh=0 mode or ImRefresh=1 mode.

1) ImRefresh=0: The counter is configured in the programming software for soft output control. Each counter outputs any of Y0 to Y7 with a latency.

The output time ranges from 100 µs to 3,000 ms.

If ImRefreshCycle is 0, call HC_ResetCmpOutput to pull low the output.

2) ImRefresh=1: Data is output immediately through hardware. Each counter outputs specified data of Y0 to Y7 without a latency. The output time is set through ImRefreshCycle.

The output time ranges from 0 to 3,000 ms.

3) Call HC_EnableInterrupt in advance to enable coincident interrupts.

4) After hot reset and cold reset, retain the previous comparison value.

■ HC_PresetValue

Write preset values.
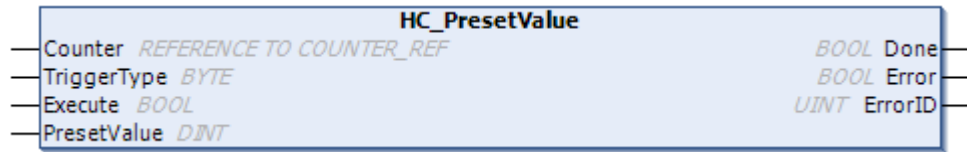
Figure 6-3 Diagram of the HC_PresetValue function block

Table 6-5 Table of HC_PresetValuefunction block I/O parameters

Parameter Name	Parameter Type	I/O Type	Description
Counter	COUNTER_REF	In	Specifies the counter number (0 to 7). Preset values can be written through external trigger only in counters 0 to 3.
TriggerType	BYTE	In	0: Triggered by rising edges 1: Triggered by external inputs External signals: couter0->x8 couter1->x9 couter2->xa couter3->xb 2: Preset during coincident output, triggered by rising edges
Execute	Bool	In	Indicates the enabling bit, triggered by rising edges.
PresetValue	Dint	In	Indicates the preset value of the counter. The value ranges from -2147483648 to 2147483647, -2147483648 and 2147483647 excluded.
Done	Bool	Out	Indicates that the function block execution is completed.
Error	Bool	Out	Indicates the function block error flag.
ErrorID	Uint	Out	Indicates the error code.

■ HC_EnableInterrupt

Enables external interrupts and coincident interrupts.
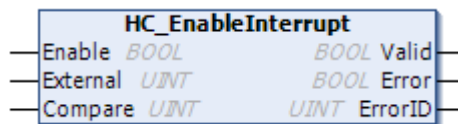
Figure 6-4 Diagram of the HC_EnableInterrupt function block

Table 6-6 Table of HC_EnableInterrupt function block I/O parameters

Parameter Name	Parameter Type	I/O Type	Description
Enable	Bool	In	Enables interrupts (1: Enable; 0: Invalid).
External	Uint	In	Enables external input interrupts. For example, if the value is 3 (2#11 in binary format), 0 and 1 bits of the port are enabled.
Compare	Uint	In	Enables coincident interrupts. For example, if the value is 3 (2#11 in binary format), 0 and 1 bits of the port are enabled.
Valid	Bool	Out	Indicates that interrupts are valid.
Error	Bool	Out	Indicates the function block error flag.
ErrorID	Uint	Out	Indicates the error code.

Note

1) If HC_WriteInterruptParameter is executed before HC_EnableInterrupt, HC_WriteInterruptParameter is valid.

2) If HC_EnableInterrupt is executed before HC_WriteInterruptParameter, interrupt parameters in the programming software are valid.

If HC_WriteInterruptParameter is executed again, HC_WriteInterruptParameter is valid.

■ HC_TouchProbe

When external interrupts are generated, read and latch the current values of counters 0 to 3.
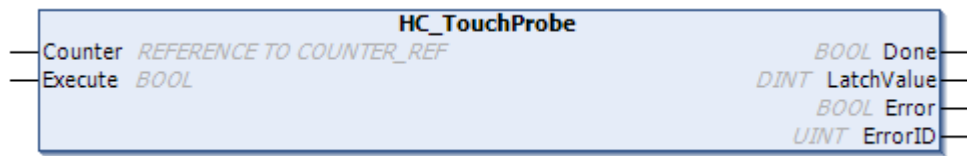
Figure 6-5 Diagram of the HC_TouchProbe function block

Table 6-7 Table of HC_TouchProbe function block I/O parameters

Parameter Name	Parameter Type	I/O Type	Description
Counter	COUNTER_REF	In	Specifies the counter number (0 to 3).
Execute	Bool	In	Interrupts are triggered by rising edges. Only one interrupt can be triggered at one time.
Done	Bool	Out	1: Execution finished
Busy	Bool	Out	Indicates that the function block is being executed.
Value	Dint	Out	Indicates the latch value.
Error	Bool	Out	Indicates the function block error flag.
ErrorID	Uint	Out	Indicates the error code.
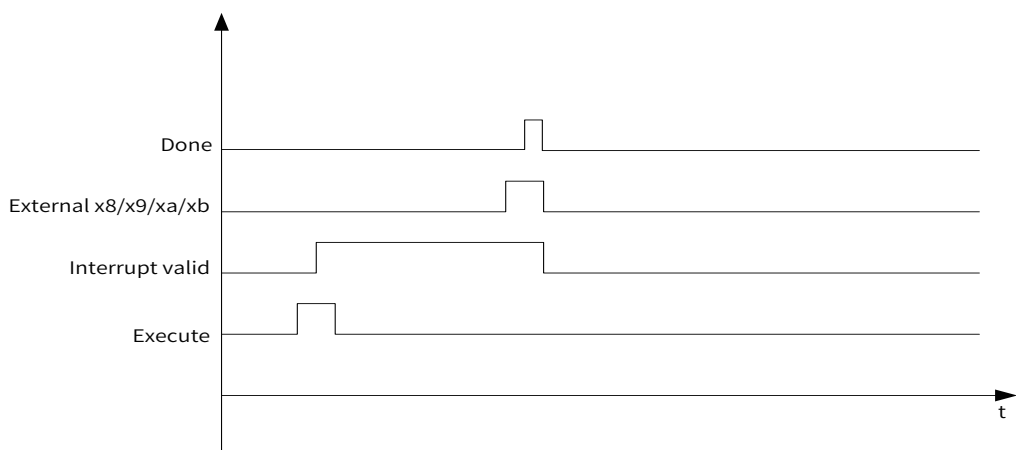
Figure 6-6 Sequence diagram of the HC_TouchProbe function block

■　HC_MeasurePulseWidth

When the external trigger is valid, read the measured pulse width.
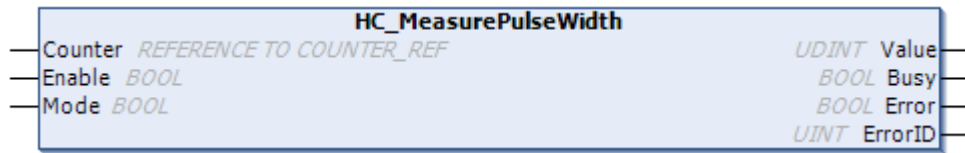
Figure 6-7 Diagram of the HC_MeasurePulseWidth function block

Table 6-8 Table of HC_MeasurePulseWidth function block I/O parameters

Parameter Name	Parameter Type	I/O Type	Description
Counter	COUNTER_REF	In	Specifies the counter number (0 to 3).
Enable	Bool	In	Indicates the enabling bit, triggered by level.
Value	Udint	Out	Indicates the measured pulse width, in the unit of μs, ranging from 0 to 4000000.
Mode	Bool	In	0: External signals at high level (high-level pulse width measured); 1: External signals at low-level (low-level pulse width measured)
Busy	Bool	Out	1: Sampling; 0: Not sampling
Error	Bool	Out	Indicates the function block error flag.
ErrorID	Uint	Out	Indicates the error code.

Note: The HC_Counter function block and HC_MeasurePulseWidth function block cannot be used simultaneously. Apply either of them to recommended programs.

■　HC_Sample

Sample counters, counting the number of counters within a period of time.
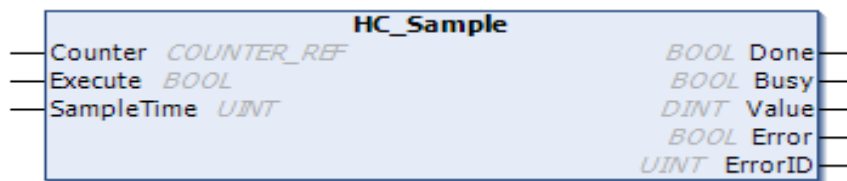
Figure 6-8 Diagram of the HC_Sample function block

Table 6-9 Table of HC_Sample function block I/O parameters

Parameter Name	Parameter Type	I/O Type	Description
Counter	COUNTER_REF	In	Specifies the counter number (0 to 3).
Enable	Bool	In	Indicates the enabling bit, triggered by level. Waits for triggers by external inputs (x8/x9/xa/xb).
SampleTime	Uint	In	10 to 65535 (10 ms to 65535 ms)
Value	Dint	Out	Indicates the sampled value.
Done	Bool	Out	1: Sampling finished; 0: Sampling unfinished
Busy	Bool	Out	1: Sampling; 0: Stop It is applicable only to counters 0 to 3.
Error	Bool	Out	Indicates the function block error flag.
ErrorID	Uint	Out	Indicates the error code.
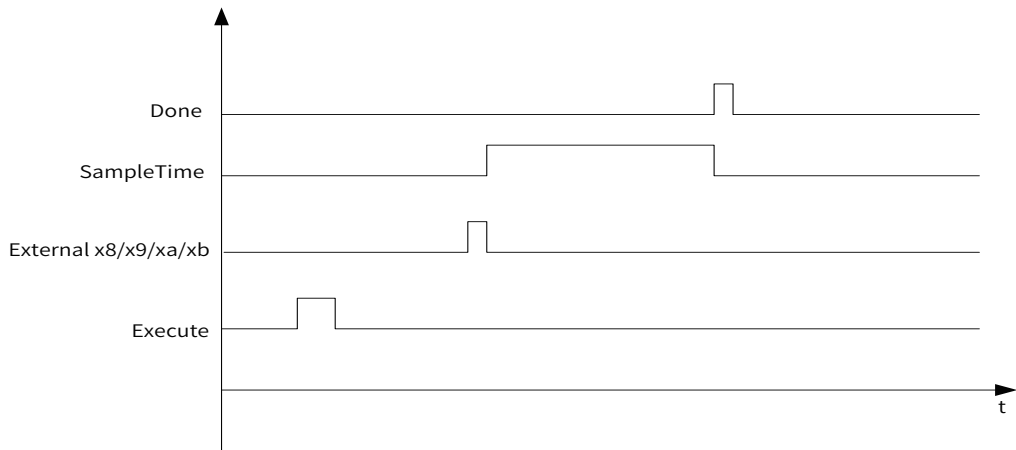
Figure 6-9 Sequence diagram of the HC_Sample function block

Note: The HC_Sample function block runs only after the HC_Counter function block is enabled.

■   HC_ReadBoolParameter

Read parameters.
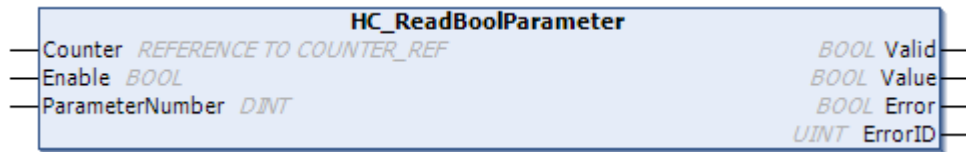
Figure 6-10 Diagram of the HC_ReadBoolParameter function block

Table 6-10 Table of HC_ReadBoolParameter function block I/O parameters

Parameter Name	Parameter Type	I/O Type	Description
Counter	COUNTER_REF	In	Specifies the counter number (0 to 7).
Enable	Bool	In	Indicates the enabling bit.
ParameterNumber	Dint	In	Indicates the parameter number.
Valid	Bool	Out	1: Valid; 0: Invalid
Value	Bool	Out	Indicates the parameter value.
Error	Bool	Out	Indicates the function block error flag.
ErrorID	Uint	Out	Indicates the error code.

Table 6-11 PN parameter mapping table

PN	Name	DataByte	R/W	Comments
10001	bDisableCmpEvent	Bool	R/W	Sets external comparison events for user tasks (0: Valid; 1: Invalid).
10002	bDisableOutput	Bool	R/W	Indicates whether data is output through the coincident output port (0: No; 1: Yes). It is invalid for ImRefresh.
10008	Direction	Bool	R/W	Indicates the counter direction.
10009	bDisableFrequency	Bool	R/W	Disables FrequencyValue in HC_Counter.
10010	bDisableRotationRate	Bool	R/W	Disables RotationRateValue in HC_Counter.

■ HC_WriteBoolParameter

Write parameters.
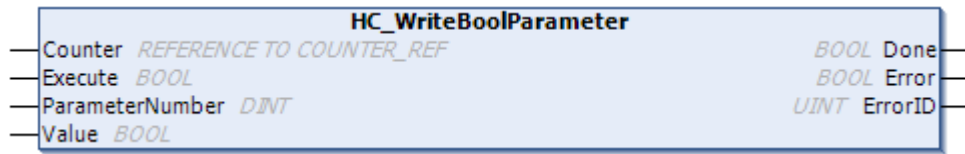
Figure 6-11 Diagram of the HC_WriteBoolParameter function block

Table 6-12 Table of HC_WriteBoolParameter function block I/O parameters

Parameter Name	Parameter Type	I/O Type	Description
Counter	COUNTER_REF	In	Specifies the counter number (0 to 7).
Execute	Bool	In	Indicates the enabling bit, triggered by rising edges.
ParameterNumber	DINT	In	Indicates the parameter number.
Value	BOOL	In	Indicates the parameter value.
Done	Bool	Out	1: Execution finished; 0: Execution unfinished
Error	Bool	Out	Indicates the function block error flag.
ErrorID	Uint	Out	Indicates the error code.

Table 6-13 PN parameter mapping table

PN	Name	DataByte	R/W	Comments
10001	bDisableCmpEvent	Bool	R/W	Sets external comparison events for user tasks (1: Invalid; 0: Valid).
10002	bDisableOutput	Bool	R/W	Indicates whether data is output through the coincident output port (0: Yes; 1: No). It is invalid when ImRefresh is **true**.
10008	Direction	Bool	R/W	Indicates the counter direction, which takes effect immediately.
10009	bDisableFrequency	Bool	R/W	Disables FrequencyValue in HC_Counter.
10010	bDisableRotationRate	Bool	R/W	Disables RotationRateValue in HC_Counter.

■ HC_SetCompareM

Coincident output: You can set up to 100 comparison values. "Done" indicates that signals are output, and "Value" indicates the index number currently output.

Figure 6-12 Diagram of the HC_SetCompareM function block

Table 6-14 Table of HC_SetCompareM function block I/O parameters

Parameter Name	Parameter Type	I/O Type	Description
Counter	COUNTER_REF	In	Specifies the counter number (0 to 7).
Enable	Bool	In	Indicates the enabling bit, triggered by rising edges.
CompareValues	ARRAY [0..99] OF DINT	In	Indicates one-dimensional arrays of counter comparison values. A maximum of 100 arrays are supported. Each value ranges from -2147483648 to 2147483647, -2147483648 and 2147483647 excluded.
Numbers	Dword	In	Indicates the number of one-dimensional arrays of CompareValues. The maximum number is 100.
ImRefresh	Bool	In	Data is output directly through hardware.
ImRefreshCycle	Uint	In	Indicates the time for output directly through hardware, in the unit of 100 µs. The maximum output time is 3,000 ms. For example, the value 10000 indicates that the output time is 1,000 ms.
Done	Bool	Out	Indicates the execution complete flag.
NumOfEqual	Dword	Out	Indicates the number of equal values.
Error	Bool	Out	Indicates the function block error flag.
ErrorID	Uint	Out	Indicates the error code.

Note

1) 1: When Enable=1, the counting direction cannot be changed.

2) Array elements must be increased gradually in positive or decreased gradually in negative. Equal values are not allowed.

3) Positive: The start value must be less than the first comparison value. Otherwise, the first comparison value cannot be implemented.

   Negative: The start value must be greater than the first comparison value. Otherwise, the first comparison value cannot be implemented.

4) Call HC_EnableInterrupt in advance to enable coincident interrupts.

5) When the device runs in the forward direction, parameters are defined in the following format:

   arr1 :ARRAY [0..99] OF DINT := [2000,4000,6000,8000,10000,12000,14000,16000,18000,20000];

   When the device runs in the reverse direction, parameters are defined in the following format:

   arr2 :ARRAY [0..99] OF DINT := [20000,18000,16000,14000,12000,10000,8000,6000,5000,3000];

6) This function depends on HC_Counter.

7) Y0 to Y7 can be controlled in ImRefresh=0 mode or ImRefresh=1 mode.

■ ImRefresh=0: The counter is configured in the programming software for soft output control. Each counter outputs any of Y0 to Y7 with a latency.

   The output time ranges from 100 µs to 3,000 ms.

   If ImRefreshCycle is 0, call HC_ResetCmpOutput to pull low the output.

■ ImRefresh=1: Data is output immediately through hardware. Each counter outputs specified data of Y0 to Y7 without a latency. The output time is set through ImRefreshCycle.

   The output time ranges from 0 to 3,000 ms.

■ Call HC_EnableInterrupt in advance to enable coincident interrupts.

8) After hot reset and cold reset, retain the previous comparison value.

■ HC_SetRing

Sets ring counting.
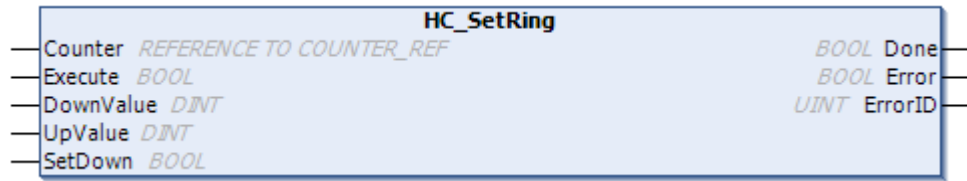
Figure 6-13 Diagram of the HC_SetRing function block

Table 6-15 Table of HC_SetRing function block I/O parameters

Parameter Name	Parameter Type	I/O Type	Description
Counter	COUNTER_REF	In	Specifies the counter number (0 to 3).
Execute	Bool	In	Indicates the enabling bit, triggered by rising edges.
DownValue	Dint	In	Indicates the lower limit, ranging from -2147483648 to 2147483647, -2147483648 and 2147483647 excluded.
UpValue	Dint	In	Indicates the upper limit, ranging from -2147483648 to 2147483647, -2147483648 and 2147483647 excluded.
SetDown	Bool	In	Forcibly counts from the lower limit.
Done	Bool	Out	Indicates that setting succeeded.
Error	Bool	Out	Indicates the function block error flag.
ErrorID	Uint	Out	Indicates the error code.

■ HC_ResetCmpOutput

Resets the port for coincident output (set in the programming software). The port for direct hardware output (ImRefresh and ImRefreshCycle of HC_SetCompare) is invalid.
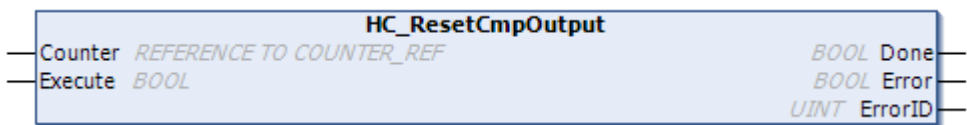
Figure 6-14 Diagram of the HC_ResetCmpOutput function block

Table 6-16 Table of HC_ResetCmpOutput function block I/O parameters

Parameter Name	Parameter Type	I/O Type	Description
Counter	COUNTER_REF	In	Specifies the counter number (0 to 7).
Execute	Bool	In	Indicates the enabling bit, triggered by rising edges.
Done	Bool	Out	Indicates the flag of successful setting.
Error	Bool	Out	Indicates the function block error flag.
ErrorID	Uint	Out	Indicates the error code.

■   HC_WriteInterruptParameter

Write interrupt parameters.

Figure 6-15 Diagram of the HC_WriteInterruptParameter function block

Table 6-17 Table of HC_WriteInterruptParameter function block I/O parameters

Parameter Name	Parameter Type	I/O Type	Description
Execute	Bool	In	Indicates the enabling bit, triggered by rising edges.
ParameterNumber	DINT	In	Indicates the parameter number.
Value	BOOL	In	Indicates the parameter value.
Done	Bool	Out	1: Execution finished; 0: Execution unfinished
Error	Bool	Out	Indicates the function block error flag.
ErrorID	Uint	Out	Indicates the error code.

Table 6-18 PN parameter mapping table

PN	Name	DataByte	R/W	Comments
10080	X0InterruptEvent	Byte	W	0: Rising edge; 1: Falling edge; 2: Rising edge + falling edge
10081	X1InterruptEvent	Byte	W	0: Rising edge; 1: Falling edge; 2: Rising edge + falling edge
10082	X2InterruptEvent	Byte	W	0: Rising edge; 1: Falling edge; 2: Rising edge + falling edge
10083	X3InterruptEvent	Byte	W	0: Rising edge; 1: Falling edge; 2: Rising edge + falling edge
10084	X4InterruptEvent	Byte	W	0: Rising edge; 1: Falling edge; 2: Rising edge + falling edge
10085	X5InterruptEvent	Byte	W	0: Rising edge; 1: Falling edge; 2: Rising edge + falling edge
10086	X6InterruptEvent	Byte	W	0: Rising edge; 1: Falling edge; 2: Rising edge + falling edge
10087	X7InterruptEvent	Byte	W	0: Rising edge; 1: Falling edge; 2: Rising edge + falling edge
10088	X8InterruptEvent	Byte	W	Counter0 HC_TouchProbe interrupt: 0: Rising edge; 1: Falling edge
10089	X9InterruptEvent	Byte	W	Counter1 HC_TouchProbe interrupt: 0: Rising edge; 1: Falling edge
10090	XaInterruptEvent	Byte	W	Counter2 HC_TouchProbe interrupt: 0: Rising edge; 1: Falling edge
10091	XbInterruptEvent	Byte	W	Counter3 HC_TouchProbe interrupt: 0: Rising edge; 1: Falling edge

■   HC_Reset

Clear errors.

Figure 6-16 Diagram of the HC_Reset function block

Table 6-19 Table of HC_Reset function block I/O parameters

Parameter Name	Parameter Type	I/O Type	Description
Counter	COUNTER_REF	In	Specifies the counter number (0 to 7).
Execute	Bool	In	Indicates the enabling bit, triggered by rising edges.
Done	Bool	Out	1: Execution finished; 0: Execution unfinished
Error	Bool	Out	Indicates the function block error flag.
ErrorID	Uint	Out	Indicates the error code.

2 ErrorID

Basic format:

Library + function block + error code

            3   3-bit

Library: The default high-speedI/O is 0.

Function block number: Function blocks are numbered from 01. Function blocks are detailed in 6.2.4. List of Function Blocks.

Error code: The error code starts from 01. Error codes are detailed in the list of counter error codes. If the error code is less than 500, it indicates a serious error. If the error code is greater than 500, it indicates a function block error.

In the example of 14506, **14** indicates HC_WriteParameter, and **506** indicates a parameter error.

Table 6-20 List of counter error codes

Indicates the error code.	Definition	Description
001	ERR_COUNTERID_INVALID	The entered channel number is invalid. A valid number ranges from 0 to 7.
003	ERR_CNT_OVERFLOW	The counter overflow/underflow is incorrect.
004	ERR_COUNTER_NOT_CHOSEN	No high-speed function is selected. Select a high-speed in the programming software.
007	ERR_COUNTER_NOT_ENABLED	HC_Counter is not enabled.
101	ERR_WRITEINTERRUPTPARAMETER_UNVALIAD	The write interrupt parameter is invalid.
102	ERR_INTERRUPT_NOT_CHOSE	**Interrupt Input** is not selected in the programming software.
501	ERR_SETCOMPARE_IMREFRESHCYCLE_ OVERFLOW	The comparison value ImRefreshCycle exceeds 30000. A valid comparison value ranges from 0 to 30000.
502	ERR_SETCOMPAREM_NUMBERS_OVERFLOW	The HC_SetCompareM number ranges from 1 to 100.
503	ERR_PREWR_VALUE_OVERFLOW	The preset value is out of range.
504	ERR_AVERAGE_PARA_UNVALIAD	The set average frequency and average rotational speed are invalid.
505	ERR_ROTATION_PULSES_UNIT_UNVALIAD	The set number of pulses per rotation is invalid.
506	ERR_WRITEBOOlPARAMETER_UNVALIAD	The set HC_WriteBoolParameter parameter is invalid.
507	ERR_READBOOLPARAMETER_UNVALIAD	The obtained HC_ReadBoolParameter parameter is invalid
508	ERR_MEASURE_WIDTH_OVERFLOW	The measured width is invalid.
509	ERR_SETCOMPAREM_IMREFRESHCYCLE_ OVERFLOW	The comparison value ImRefreshCycle exceeds 30000. A valid comparison value ranges from 0 to 30000.
510	ERR_PRESET_TRIGGERTYPE_OVERFLOW	The preset parameter is invalid.
511	ERR_WRITEPARAMETER_UNVALIAD	The set HC_WriteParameter parameter is invalid.
513	ERR_FUNC_COUNTERID_INVALID	The special channel number is invalid. A valid number ranges from 0 to 3.

Indicates the error code.	Definition	Description
514	ERR_COUNTER_NOT_CHOSE_EXETERNAL_X	**External Trigger** is not selected in the programming software.
515	ERR_CNT_FORMAT_NOT_RING	The ring counting type is incorrect. Select a correct type in the programming software.
516	ERR_RING_DOWNVAL_BEYOND_UPVAL	The lower limit for ring counting is equal to or greater than the upper limit.
517	ERR_SAMPLE_VALUE_LESS	The sampling time is too short. A valid value ranges from 10 to 65535, in the unit of ms.
518	ERR_RING_VALUE_OVERFLOW	The ring counting is out of range.

6.2.2 High-speed Axis

The maximum frequency of high-speed axis input is 200 kHz.

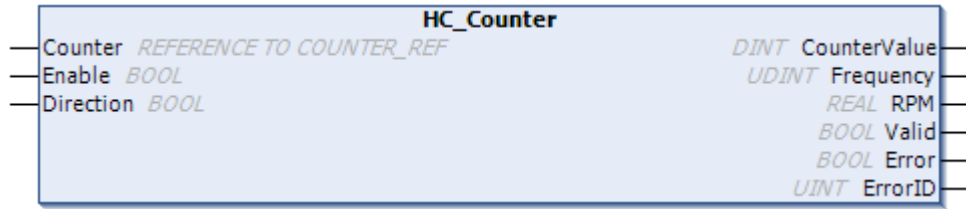### 1 Function Blocks

■　MC_Jog_P
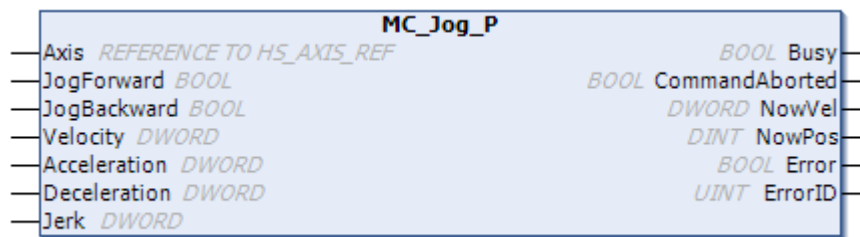
Figure 6-17　Diagram of the MC_Jog_P function block

Table 6-21　Table of MC_Jog_P function block I/O parameters

Parameter Name	Parameter Type	I/O Type	Description
Axis	HS_AXIS_REF	In	Indicates the instruction axis number, ranging from 0 to 3.
JogForward	Bool	In	Indicates a jog in the forward direction, triggered by level.
JogBackward	Bool	In	Indicates a jog in the reverse direction, triggered by level.
Velocity	Dword	In	Indicates the jog speed, in the unit of pulse/s. The maximum frequency is 200 kHz.
Acceleration	Dword	In	Indicates the acceleration, in the unit of pulse/s². The maximum value is 20000000 by default. It can be modified through dwMaxAcceleration. See remarks.
Deceleration	Dword	In	Indicates the deceleration, in the unit of pulse/s². The maximum value is 20000000 by default. It can be modified through dwMaxDeceleration. See remarks.
Jerk	Dword	In	Indicates the jerk, in the unit of pulse/s³.
Busy	Bool	Out	Indicates the jog flag, which is ON during motion and OFF when motion stops.
CommandAborted	Bool	Out	Indicates that the command is interrupted by other commands.

Parameter Name	Parameter Type	I/O Type	Description
Error	Bool	Out	Indicates the axis error flag.
ErrorID	Uint	Out	Indicates the error code.

1) The acceleration and deceleration values are multiples of 125.

$$Rate = \begin{cases} 1 & Vmax \leqq 8000 \\ \frac{Vmax}{8000} & Vmax > 8000 \end{cases}$$

$$Vel = \left[\frac{Vel}{Rate}\right] * Rate$$

$$Actual\ acceleration = \begin{cases} 125 * Rate & Acc \leqq 125 * Rate \\ \left[\frac{Acc}{125 * Rate}\right] * 125 * Rate & Acc > 125 * Rate \end{cases}$$

$[\ ]$: integer by roundoff

2) The speed cannot be changed during acceleration/deceleration.

3) S-shaped curve: The jerk is calculated internally. All jerk parameters are calculated internally.

4) S-shaped curve: The speed, acceleration, and deceleration cannot be calculated during motion.

5) S-shaped curve: When the deceleration is equal to or greater than the speed multiplied by 5, no smearing or truncation occurs.

■ MC_Home_P
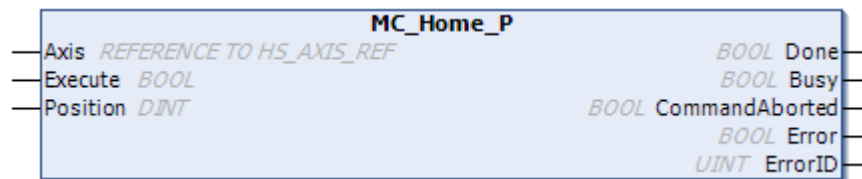
Figure 6-18 Diagram of the MC_Home_P function block

Table 6-22 Table of MC_Home_P function block I/O parameters

Parameter Name	Parameter Type	I/O Type	Description
Axis	HS_AXIS_REF	In	Indicates the instruction axis number, ranging from 0 to 3.
Execute	Bool	In	Indicates the enabling bit, triggered by rising edges.
Position	Dint	In	Indicates homing position.
Done	Bool	Out	Indicates the homing complete flag.
Busy	Bool	Out	Indicates the motion flag, which is ON during motion and OFF when motion stops.
CommandAborted	Bool	Out	Indicates that the command is interrupted by other commands.
Error	Bool	Out	Indicates the axis error flag.
ErrorID	Uint	Out	Indicates the error code.

mode0

Homing in the forward direction: Set the deceleration point and home as the home switch and set the right limit for homing.

Note: When homing is enabled, the position on the right of the final limit position is invalid.

mode1

Homing in the reverse direction: Set the deceleration point and home as the home switch and set the left limit for homing.

Note: When homing is enabled, the position on the left of the left final position is invalid.

mode2

Homing in the forward direction: Set the deceleration point and home as the home switch.
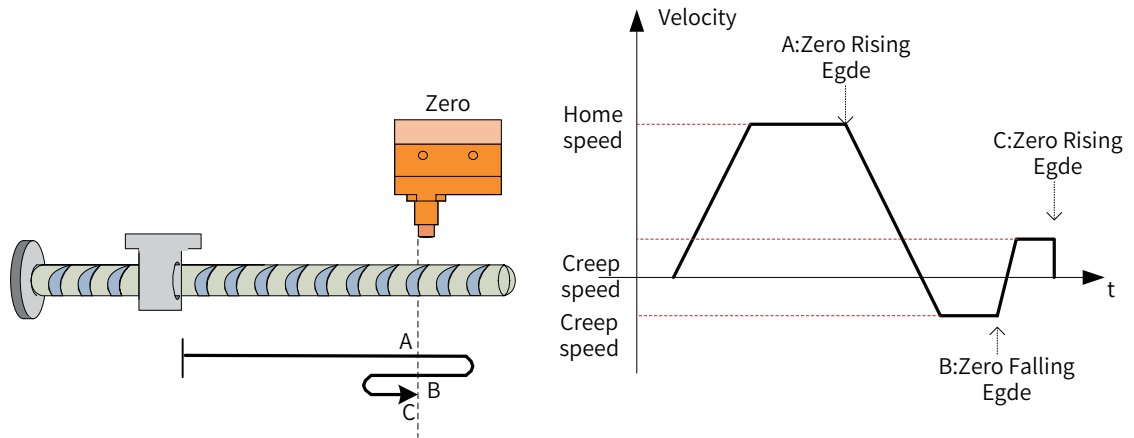
Note: When homing is enabled, the position on the center or right of the home position is invalid.

mode3

Homing in the reverse direction: Set the deceleration point and home as the home switch.
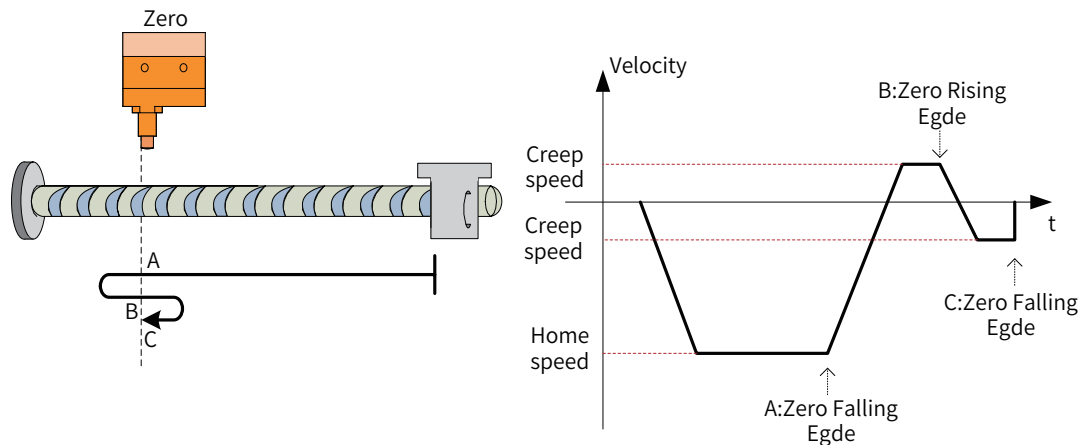
Note: When homing is enabled, the position on the center or left of the home position is invalid.

■ MC_MoveAbsolute_P
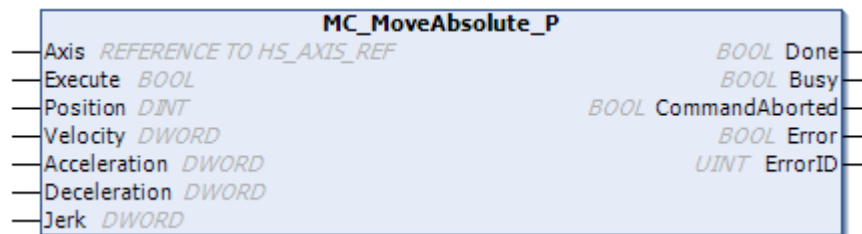
Figure 6-19 Diagram of the MC_MoveAbsolute_P function block

Table 6-23 Table of MC_MoveAbsolute_P function block I/O parameters

Parameter Name	Parameter Type	I/O Type	Description
Axis	HS_AXIS_REF	In	Indicates the instruction axis number, ranging from 0 to 3.
Execute	Bool	In	Indicates the enabling bit, triggered by rising edges.
Position	DINT	In	Position
Velocity	Dword	In	Indicates the speed, in the unit of pulse/s. The maximum frequency is 200 kHz.
Acceleration	Dword	In	Indicates the acceleration, in the unit of pulse/s². The maximum value is 20000000 by default. It can be modified through dwMaxAcceleration. See remarks.
Deceleration	Dword	In	Indicates the deceleration, in the unit of pulse/s². The maximum value is 20000000 by default. It can be modified through dwMaxDeceleration. See remarks.
Jerk	Dword	In	Indicates the jerk, in the unit of pulse/s³.
Done	Bool	Out	Complete flag
Busy	Bool	Out	Indicates the motion flag, which is ON during motion and OFF when motion stops.
CommandAborted	Bool	Out	Indicates that the command is interrupted by other commands.
Error	Bool	Out	Indicates the axis error flag.
ErrorID	Uint	Out	Indicates the error code.

Remarks:

1) The acceleration and deceleration values are multiples of 125.

$$\text{Rate} = \begin{cases} 1 & \text{Vmax} \leqq 8000 \\ \dfrac{\text{Vmax}}{8000} & \text{Vmax} > 8000 \end{cases}$$

$$\text{Vel} = \left[\dfrac{\text{Vel}}{\text{Rate}}\right] * \text{Rate}$$

$$\text{Actual acceleration} = \begin{cases} 125 * \text{Rate} & \text{Acc} \leqq 125 * \text{Rate} \\ \left[\dfrac{\text{Acc}}{125 * \text{Rate}}\right] * 125 * \text{Rate} & \text{Acc} > 125 * \text{Rate} \end{cases}$$

[ ]: integer by roundoff

2) The speed cannot be changed during acceleration/deceleration.

3) Ensure that the positioning distance value does not exceed 2147483647. Otherwise, the device runs in the reverse direction.

4) S-shaped curve: The jerk is calculated internally. All jerk parameters are calculated internally.

5) S-shaped curve: The speed, acceleration, and deceleration cannot be calculated during motion.

6) S-shaped curve: When the deceleration is equal to or greater than the speed multiplied by 5, no smearing or truncation occurs.

■   MC_MoveRelative_P
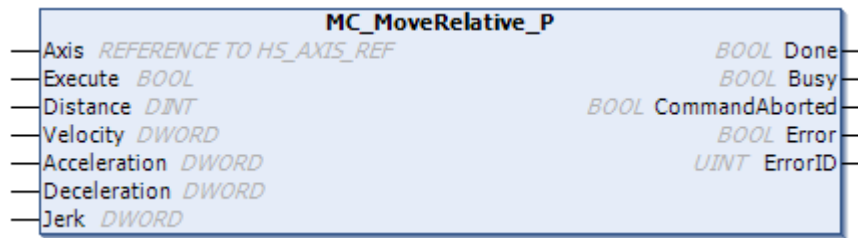
Figure 6-20 Diagram of the MC_MoveRelative_P function block

Table 6-24 Table of MC_MoveRelative_P function block I/O parameters

Parameter Name	Parameter Type	I/O Type	Description
Axis	HS_AXIS_REF	In	Indicates the instruction axis number, ranging from 0 to 3.
Execute	Bool	In	Indicates the enabling bit, triggered by rising edges.
Distance	DINT	In	Indicates the relative distance.
Velocity	Dword	In	Indicates the speed, in the unit of pulse/s. The maximum frequency is 200 kHz.
Acceleration	Dword	In	Indicates the acceleration, in the unit of pulse/s^2. The maximum value is 20000000 by default. It can be modified through dwMaxAcceleration. See remarks.
Deceleration	Dword	In	Indicates the deceleration, in the unit of pulse/s^2. The maximum value is 20000000 by default. It can be modified through dwMaxDeceleration. See remarks.
Jerk	Dword	In	Indicates the jerk, in the unit of pulse/s^3.
Done	Bool	Out	Complete flag
Busy	Bool	Out	Indicates the motion flag, which is ON during motion and OFF when motion stops.
CommandAborted	Bool	Out	Indicates that the command is interrupted by other commands.
Error	Bool	Out	Indicates the axis error flag.
ErrorID	Uint	Out	Indicates the error code.

Note

1)   The acceleration and deceleration values are multiples of 125.

$$\text{Rate} = \begin{cases} 1 & \text{Vmax} \leqq 8000 \\ \dfrac{\text{Vmax}}{8000} & \text{Vmax} > 8000 \end{cases}$$

$$\text{Vel} = \left[\dfrac{\text{Vel}}{\text{Rate}}\right] * \text{Rate}$$

$$\text{Actual acceleration} = \begin{cases} 125 * \text{Rate} & \text{Acc} \leqq 125 * \text{Rate} \\ \left[\dfrac{\text{Acc}}{125*\text{Rate}}\right] * 125 * \text{Rate} & \text{Acc} > 125 * \text{Rate} \end{cases}$$

[ ]: integer by roundoff

2)   When the distance value is negative, it indicates that the device runs again from the stop position.

In this case, stop pulses and then run the device the distance in the reverse direction.

3)   The speed cannot be changed during acceleration/deceleration.

4)   Ensure that the positioning distance value does not exceed 2147483647. Otherwise, the device runs in the reverse direction.

5) S-shaped curve: The jerk is calculated internally. All jerk parameters are calculated internally.

6) S-shaped curve: The speed, acceleration, and deceleration cannot be calculated during motion.

7) S-shaped curve: When the deceleration is equal to or greater than the speed multiplied by 5, no smearing or truncation occurs.

■ MC_MoveVelocity_P
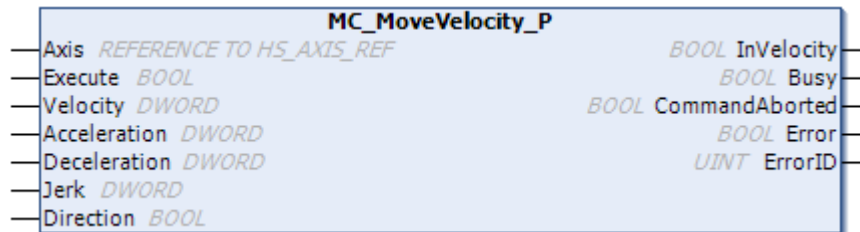
Figure 6-21 Diagram of the MC_MoveVelocity_P function block

Table 6-25 Table of MC_MoveVelocity_P function block I/O parameters

Parameter Name	Parameter Type	I/O Type	Description
Axis	HS_AXIS_REF	In	Indicates the instruction axis number, ranging from 0 to 3.
Execute	Bool	In	Indicates the enabling bit, triggered by rising edges.
Velocity	Dword	In	Indicates the speed, in the unit of pulse/s. The maximum frequency is 200 kHz.
Acceleration	Dword	In	Indicates the acceleration, in the unit of pulse/s^2. The maximum value is 20000000 by default. It can be modified through dwMaxAcceleration. See remarks.
Deceleration	Dword	In	Indicates the deceleration, in the unit of pulse/s^2. The maximum value is 20000000 by default. It can be modified through dwMaxDeceleration. See remarks.
Jerk	Dword	In	Indicates the jerk, in the unit of pulse/s^3.
Direction	Bool	In	Indicates the direction.
InVelocity	Bool	Out	Indicates that the speed reaches the set value.
Busy	Bool	Out	Indicates the motion flag, which is ON during motion and OFF when motion stops.
CommandAborted	Bool	Out	Indicates that the command is interrupted by other commands.
Error	Bool	Out	Indicates the axis error flag.
ErrorID	Uint	Out	Indicates the error code.

Note

1) The acceleration and deceleration values are multiples of 125.

$$\text{Rate} = \begin{cases} 1 & \text{Vmax} \leqq 8000 \\ \dfrac{\text{Vmax}}{8000} & \text{Vmax} > 8000 \end{cases}$$

$$\text{Vel} = \left[\dfrac{\text{Vel}}{\text{Rate}}\right] * \text{Rate}$$

$$\text{Actual acceleration} = \begin{cases} 125 * \text{Rate} & \text{Acc} \leqq 125 * \text{Rate} \\ \left[\dfrac{\text{Acc}}{125 * \text{Rate}}\right] * 125 * \text{Rate} & \text{Acc} > 125 * \text{Rate} \end{cases}$$

[ ]: integer by roundoff

2) The speed cannot be changed during acceleration/deceleration.

3) Ensure that the positioning distance value does not exceed 2147483647. Otherwise, the device runs in the reverse direction.

4) S-shaped curve: The jerk is calculated internally. All jerk parameters are calculated internally.

5) S-shaped curve: The speed, acceleration, and deceleration cannot be calculated during motion.

6) S-shaped curve: When the deceleration is equal to or greater than the speed multiplied by 5, no smearing or truncation occurs.

■ MC_Stop_P

Figure 6-22 Diagram of the MC_Stop_P function block

Table 6-26 Table of MC_Stop_P function block I/O parameters

Parameter Name	Parameter Type	I/O Type	Description
Axis	HS_AXIS_REF	In	Indicates the instruction axis number, ranging from 0 to 3.
Execute	Bool	In	Indicates the enabling bit, triggered by rising edges.
Deceleration	DWORD	In	Indicates the deceleration, in the unit of pulse/s². The maximum value is 20000000 by default. It can be modified through dwMaxDeceleration. Note 1. See remarks. 2. When the deceleration is 0, the device immediately stops.
Jerk	DWORD	In	Indicates the jerk, in the unit of pulse/s³.
Done	Bool	Out	Indicates the execution complete flag.
Busy	Bool	Out	Indicates the execution status flag.
CommandAborted	Bool	Out	Indicates that the command is interrupted by other commands.
Error	Bool	Out	Indicates the axis error flag.
ErrorID	Uint	Out	Indicates the error code.

Note

1) The acceleration and deceleration values are multiples of 125.

$$Rate = \begin{cases} 1 & Vmax \leqq 8000 \\ \dfrac{Vmax}{8000} & Vmax > 8000 \end{cases}$$

$$Vel = \left[\dfrac{Vel}{Rate}\right] * Rate$$

$$Actual\ acceleration = \begin{cases} 125 * Rate & Acc \leqq 125 * Rate \\ \left[\dfrac{Acc}{125*Rate}\right] * 125 * Rate & Acc > 125 * Rate \end{cases}$$

[ ]: integer by roundoff

2) S-shaped curve: When the deceleration is equal to or greater than the speed multiplied by 5, no smearing or truncation occurs.

3)

■   MC_Reset_P
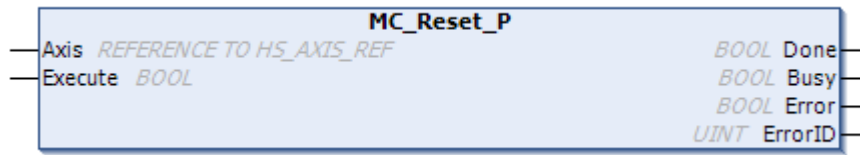
Figure 6-23 Diagram of the MC_Reset_P function block

Table 6-27 Table of MC_Reset_P I/O parameters

Parameter Name	Parameter Type	I/O Type	Description
Axis	HS_AXIS_REF	In	Indicates the instruction axis number, ranging from 0 to 3.
Execute	Bool	In	Indicates the enabling bit, triggered by rising edges.
Done	Bool	Out	Indicates the execution complete flag.
Busy	Bool	Out	Indicates the execution status flag.
Error	Bool	Out	Indicates the axis error flag.
ErrorID	Uint	Out	Indicates the error code.

■   MC_Power_P

Obtain parameters in the programming software and configure the basic FPGA register.
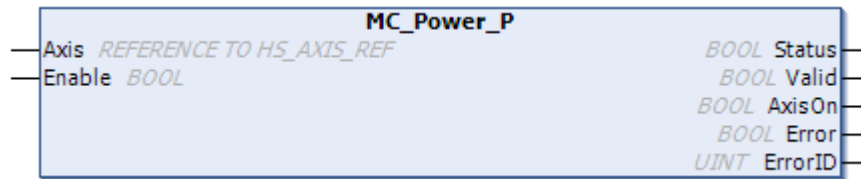
Figure 6-24 Diagram of the MC_Power_P function block

Table 6-28 Table of MC_Power_P function block I/O parameters

Parameter Name	Parameter Type	I/O Type	Description
Axis	HS_AXIS_REF	In	Indicates the instruction axis number, ranging from 0 to 3.
Enable	Bool	In	Indicates the enabling bit, triggered by level.
Status	Bool	Out	Indicates the execution status flag.
Valid	Bool	Out	Indicates the execution valid flag.
AxisOn	Bool	Out	Enables the axis.
Error	Bool	Out	Indicates the axis error flag.
ErrorID	Uint	Out	Indicates the error code.

■   MC_ReadParameter_P
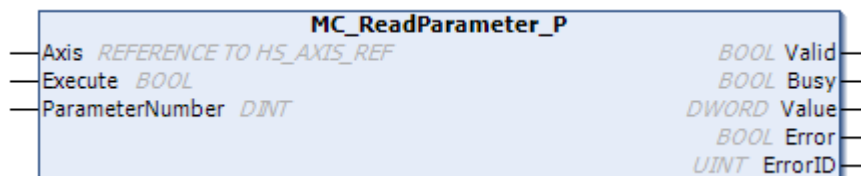
Figure 6-25 Diagram of the MC_ReadParameter_P function block

Table 6-29 Table of MC_ReadParameter_P function block I/O parameters

Parameter Name	Parameter Type	I/O Type	Description
Axis	HS_AXIS_REF	In	Indicates the instruction axis number, ranging from 0 to 3.
Enable	Bool	In	Enables the function block.
ParameterNumber	Dint	In	Indicates the parameter number.
Valid	Bool	Out	Indicates the execution valid flag.
Busy	Bool	Out	Indicates the execution status flag.
Value	Dint	Out	Indicates the returned value.
Error	Bool	Out	Indicates the axis error flag.
ErrorID	Uint	Out	Indicates the error code.

Table 6-30 PN parameter mapping table

PN	Name	DataByte	R/W	Comments
10030	diSWLimitPos	dint	R/W	Indicates the maximum soft limit.
10031	diSWLimitNeg	dint	R/W	Indicates the minimum soft limit.
10032	bEnableLimitPos	Bool	R/W	Indicates the minimum enable soft limit (0: False).
10033	bEnableLimitNeg	Bool	R/W	Indicates the maximum enable soft limit (0: False).
10034	Velocity	dint	R	Indicates the actual speed.
10035	Position	dint	R	Indicates the actual position.
10036	dwMaxAcceleration	dword	R/W	Indicates the maximum acceleration.
10037	dwMinAcceleration	dword	R/W	Indicates the minimum acceleration.
10038	dwMaxDeceleration	dword	R/W	Indicates the maximum deceleration.
10039	dwMinDeceleration	dword	R/W	Indicates the minimum deceleration.
10040	dwMaxVelocity	Dint	R/W	Indicates the maximum speed.
10041	dwHomeVelocity	dword	R/W	Indicates the homing speed.
10042	dwHomeCreepVelocity	dword	R/W	Indicates the homing creep speed.
10043	dwHomeAcceleration	dword	R/W	Indicates the homing acceleration.
10044	dwHomeDeceleration	dword	R/W	Indicates the homing deceleration.
10045	bHomeMode	Byte	R/W	Indicates the homing mode, [0, 3].
10046	bErrorStopMode	Bool	R/W	Indicates the stop mode when an axis error occurs (0: False).
10046	bSoftLimitStopMode	Bool	R/W	Indicates the stop mode for soft limit (0: False).

■ MC_WriteParameter_P

Figure 6-26 Diagram of the MC_WriteParameter_P function block

Table 6-31 Table of MC_WriteParameter_P function block I/O parameters

Parameter Name	Parameter Type	I/O Type	Description
Axis	HS_AXIS_REF	In	Indicates the instruction axis number, ranging from 0 to 3.
Execute	Bool	In	Indicates the enabling bit, triggered by rising edges.
ParameterNumber	Dint	In	Indicates the parameter number.
Value	Dint	In	Indicates the parameter value.
Done	Bool	Out	Indicates the execution complete flag.
Busy	Bool	Out	Indicates the execution status.
Error	Bool	Out	Indicates the axis error flag.
ErrorID	Uint	Out	Indicates the error code.

Table 6-32 PN parameter mapping table

PN	Name	DataByte	R/W	Comments
10030	diSWLimitPos	dint	R/W	Indicates the maximum soft limit. Note: It is the actual limit value. An error is reported at this position.
10031	diSWLimitNeg	dint	R/W	Indicates the minimum soft limit. Note: It is the actual limit value. An error is reported at this position.
10032	bEnableLimitPos	Bool	R/W	Indicates the maximum enable soft limit (0: False).
10033	bEnableLimitNeg	Bool	R/W	Indicates the minimum enable soft limit (0: False).
10036	dwMaxAcceleration	dword	R/W	Indicates the maximum acceleration.
10037	dwMinAcceleration	dword	R/W	Indicates the minimum acceleration.
10038	dwMaxDeceleration	dword	R/W	Indicates the maximum deceleration.
10039	dwMinDeceleration	dword	R/W	Indicates the minimum deceleration.
10041	dwHomeVelocity	dword	R/W	Indicates the homing speed.
10042	dwHomeCreepVelocity	dword	R/W	Indicates the homing creep speed.
10043	dwHomeAcceleration	dword	R/W	Indicates the homing acceleration.
10044	dwHomeDeceleration	dword	R/W	Indicates the homing deceleration.
10045	bHomeMode	Byte	R/W	Indicates the homing mode, [0, 3].
10046	bErrorStopMode	Bool	R/W	Indicates the stop mode when an axis error occurs (0: Stop by deceleration; 1: Stop immediately).
10047	bSoftLimitStopMode	Bool	R/W	Indicates the stop mode for soft limit (0: Stop by deceleration; 1: Stop immediately).

■ MC_SetPosition_P

Set a logical address.

Figure 6-27 Diagram of the MC_SetPosition_P function block

Table 6-33 Table of MC_SetPosition_P function block I/O parameters

Parameter Name	Parameter Type	I/O Type	Description
Axis	HS_AXIS_REF	In	Indicates the instruction axis number, ranging from 0 to 3.
Execute	Bool	In	Indicates the enabling bit, triggered by rising edges.
Position	Dint	In	Address
Relative	Bool	In	0: Absolute position of the logical address; 1: Relative position of the logical address
Done	Bool	Out	Indicates the execution complete flag.
Busy	Bool	Out	Indicates the execution status flag.
Error	Bool	Out	Indicates the axis error flag.
ErrorID	Uint	Out	Indicates the error code.

Note: 1. When a program is downloaded, the current position value is not zeroed.

■   MC_ReadStatus_P
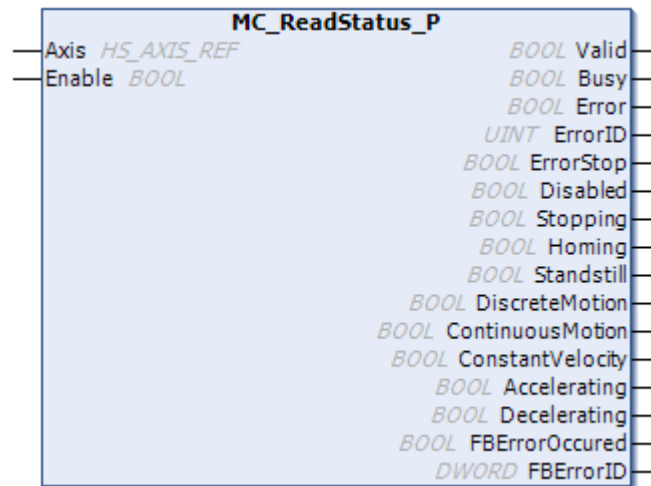
Figure 6-28 Diagram of the MC_ReadStatus_P function block

Table 6-34 Table of MC_ReadStatus_P function block I/O parameters

Parameter Name	Parameter Type	I/O Type	Description
Axis	HS_AXIS_REF	In	Indicates the instruction axis number, ranging from 0 to 3.
Enable	Bool	In	Indicates the enabling bit, triggered by level.
Valid	Bool	Out	Indicates the execution valid flag.
Busy	Bool	Out	Indicates the execution status flag.
Error	Bool	Out	Indicates the axis error flag.
ErrorID	Uint	Out	Indicates the error code.
ErrorStop	Bool	Out	Indicates the error stop mode.
Disabled	Bool	Out	Indicates that the mode is invalid.
Stopping	Bool	Out	Indicates the stopping mode.
Homing	Bool	Out	Indicates the homing mode.
Standstill	Bool	Out	Indicates the standstill mode.
DiscreteMotion	Bool	Out	Indicates the positioning mode.
ContinuousMotion	Bool	Out	Indicates the speed mode.
ConstantVelocity	Bool	Out	Indicates that the device is moving at constant speed.
Accelerating	Bool	Out	Indicates that the device is accelerating.

Parameter Name	Parameter Type	I/O Type	Description
Decelerating	Bool	Out	Indicates that the device is decelerating.
FBErrorOccured	Bool	Out	Indicates a function block error.
FBErrorID	Dword	Out	Indicates the function block error code.

2 State Machine
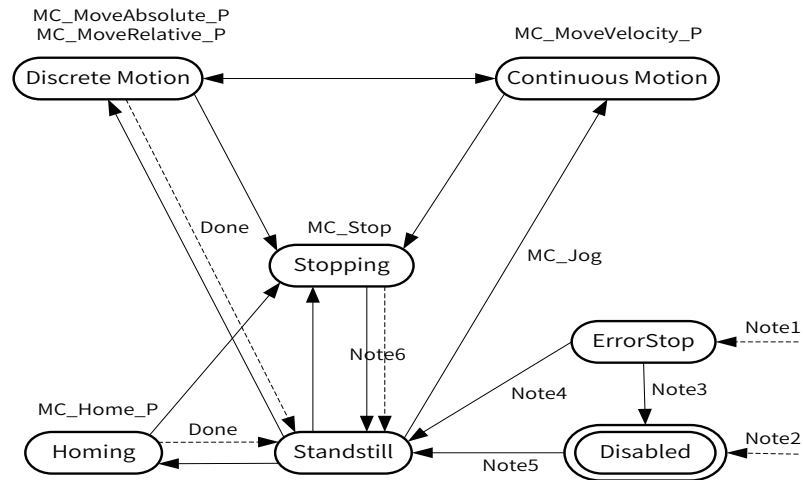
Note 1: From any state. An error in the axis occurred. Reference 2.3

Note 2: From any state. MC_Power.Enable = FALSE and there is no error in the axis.

Note 3: MC_Reset AND MC_Power.Status = FALSE

Note 4: MC_Reset AND MC_Power.Status = TRUE AND MC_Power.Enable = TRUE

Note 5: MC_Power.Enable = TRUE AND MC_Power.Status = TRUE

Note 6: MC_Stop.Done = TRUE AND MC_Stop.Execute = FALSE

Note7: MC_Jog only join Continuous Motion.

#define PMC_AS_Disabled   0

#define PMC_AS_ErrorStop   1

#define PMC_AS_Stopping   2

#define PMC_AS_Standstill   3

#define PMC_AS_DiscreteMotion   4

#define PMC_AS_ContinuousMotion 5

#define PMC_AS_Homing   7

3 ErrorID

Basic format:

Library + function block + error code

    3   3-bit

Library: The default high-speed I/O is 0.

Function block number: Function blocks are numbered from 01. Function blocks are detailed in 6.2.4. List of Function Blocks.

Error code: The error code starts from 01. Error codes are detailed in the list of counter error codes. If the error code is less than 500, it indicates a serious error. If the error code is greater than 500, it indicates a function block error.

In the example of 31520, **31** indicates MC_WriteParameter_P, and **520** indicates a parameter error.

Table 6-35 List of high-speed axis error codes

Indicates the error code.	Definition	Description
001	ERR_NOT_POWER	MC_Power is not enabled.
002	ERR_UP_SOFTWARE_LIMIT	The current position is beyond the software stroke limit (Up).
003	ERR_DOWN_SOFTWARE_LIMIT	The current position is beyond the software stroke limit (Down).
004	ERR_AXIS_FUNC_UNUSED	The high-speed axis is not enabled. Enable the axis in the programming software.
005	ERR_INPUT_CHANNAL_NUM_INVALID	The axis number is invalid. A valid number ranges from 0 to 3.
006	ERR_DEST_POS_OVER_SOFT_UP_LIMIT	The target position is beyond the upper software limit.
007	ERR_DEST_POS_OVER_SOFT_DOWN_LIMIT	The target position is beyond the lower software limit.
010	ERR_POS_DECPOINT_OVERLOW	The deceleration point is invalid: In position mode, when the device is repositioned, the deceleration length is greater than the actual distance.
011	ERR_VEL_DECPOINT_OVERLOW	The deceleration point is invalid: When you switch from the speed mode to the position mode, the deceleration length is greater than the actual distance.
012	ERR_POS_PLSNUM_OVERLOW	The maximum PLSNUM positioning length 2147483647 is exceeded.
013	ERR_POS_DECPOINT2_OVERLOW	An error occurred while recomputing the deceleration point.
501	ERR_ACC_SET_OVERFLOW	The acceleration exceeds the maximum value set by MC_WriteParameter_P.
502	ERR_ACC_SET_LOW	The acceleration is below the minimum value set by MC_WriteParameter_P.
503	ERR_DEC_SET_OVERFLOW	The deceleration exceeds the maximum value set by MC_WriteParameter_P.
504	ERR_DEC_SET_LOW	The deceleration is below the minimum value set by MC_WriteParameter_P.
505	ERR_VEL_SET_OVERFLOW	The set speed is out of range. Set the speed in the programming software or through MC_WriteParameter_P.
506	ERR_VEL_SET_LOW	The set speed is too low.
508	ERR_VEL_LESS_THAN_STARTVEL	The speed is less than the startup offset speed. Set the startup offset speed in the programming software.
509	ERR_STARTVEL_SET_LOW	The starting speed is too small.
510	ERR_FBD_MOVEMODE_INVALIAD	The motion mode of the function block is invalid.

Indicates the error code.	Definition	Description
511	ERR_WASNT_STANDSTILL	The axis is not in Standstill state.
512	ERR_WASNT_DISABLED	The axis is not in Disabled state.
513	ERR_IN_ERRORSTOP	The axis is in ErrorStop state.
514	ERR_NOT_READY_FOR_MOTION	The axis is not prepared to run.
515	ERR_INVLALID_VELOCITY_MODE	The speed mode is invalid.
516	ERR_INVLALID_POSTION_MODE	The position mode is invalid.
520	ERR_AXIS_WRITEPARAMETER_UNVALIAD	The MC_WriteParameter_P parameter is invalid.
521	ERR_AXIS_READPARAMETER_UNVALIAD	The MC_ReadParameter_P parameter is invalid.
522	ERR_HOME_MODE_UNVALIAD	The homing mode is invalid. Select a valid mode in the programming software.
523	ERR_AXIS_WRITEPARAMETER_HOME_MODE_UNVALIAD	The homing mode is invalid.

Errors can be classified into axis errors and function block errors.

Conditions for setting the axis in ErrorStop state:

1) Axis errors occur.

2) Function block errors occur when the axis is in DiscreteMotion, ContinuousMotion, or Homing state.

4 Timing for Speed Variation

Mode	Trapezoid Acceleration/Deceleration Mode			S-shaped Acceleration/Deceleration Mode		
	Accelerating	Constant speed	Decelerating	Accelerating	Constant speed	Decelerating
Indicates the speed mode.	---	The speed can be changed.	---	---	---	---
Position Mode	---	The speed can be changed.	---	---	---	---
Speed > position	---	The speed can be changed.	---	---	---	---
Position > speed	---	The speed can be changed.	---	---	---	---
Indicates the homing mode.	---	---	---	---	---	---
JOG	---	---	---	---	---	---

6.2.3 External Interrupt

1) The frequency 1 kHz is recommended. The limit input frequency is 3 kHz.

2) If the frequency of external input interrupts is too high, the system may not respond.

6.2.4 List of Function Blocks

Item	Name	No.
Counting	HC_Counter	1
	HC_SetCompare	2
	HC_PresetValue	3
	HC_ControlInterrupt	4
	HC_TouchProbe	5
	HC_MeasurePulseWidth	6
	HC_Sample	7
	HC_ReadBoolParameter	8
	HC_WriteBoolParameter	9
	HC_SetCompareM	10
	HC_SetRing	11
	HC_ResetCmpOutput	12
	HC_ReadParameter	13
	HC_WriteParameter	14
	HC_WriteInterruptParameter	15
	HC_Reset	16
Axis	Axis	20
	MC_Power_P	21
	MC_MoveAbsolute_P	22
	MC_MoveRelative_P	23
	MC_MoveVelocity_P	24
	MC_Stop_P	25
	MC_Home_P	27
	MC_Jog_P	28
	MC_Reset_P	29
	MC_ReadParameter_P	30
	MC_WriteParameter_P	31
	MC_SetPosition_P	32
	MC_ReadStatus_P	33

6.2.5 7-segment LED Display

High-speed I/O 7-segment LED Display	Definition
60	High-speed input error
62	High-speed output error
64	External interrupt and coincident interrupt errors

6.3 CANopen

6.3.1 CiA405

Function overview: Function blocks comply with CANopen communication and CiA405 standards, and support IEC61131-3. SDO read/write access, NMT state machine, emergency event packets, device node ID, and communication state machines of the slave are included.

Library name: CmpHCCiA405 (applicable to V0.1.2.1 and later versions)

1 CiA405 Overview

The CANopen interface and device configuration file for an IEC 61131-3 PLC (CiA405) describes two ways of accessing CANopen networks on the PLC: network variables and function blocks.

Network variables are usually mapped to PDOs to be received or transmitted. In an object dictionary, you can access IEC 61131-3 variables in the defined index range.

In addition, function blocks intended for CANopen are defined in the configuration file, for example, SDO, NMT, and emergency communication service.

The CAA CiA 405 library provides a group of function blocks for CiA405 to access CANopen networks through the application program (IEC61131-3 program) of the PLC (CANopen master). After the CANopen manager is added to the CAN bus interface of the PLC, the library manager of the PLC automatically declares the library.

In the library, organize function blocks as follows:

■ Network management function block

CIA405 .NMT: Controls the NMT state of a CANopen device.

CIA405 .RECV_EMCY: Scans EMCY storage of all devices.

CIA405 .RECV_EMCY_DEV: Obtains the last storage EMCY message from a specified device.

■ Self-owned node ID function block

CIA405 .GET_LOCAL_NODE_ID: Obtains the PLC CANopen manager node ID.

State query function block

CIA405 .GET_CANOPEN_KERNEL_STATE: Obtains the current state of the CANopen kernel.

CIA405 .GET_STATE: Obtains the current state of a specified device.

■ SDO access function block

CIA405 .SDO_READ: Reads objects of any size on specified devices.

CIA405 .SDO_READ4: Reads objects of no more than four bytes on specified devices.

CIA405 .SDO_WRITE: Writes objects of any size on specified devices.

CIA405 .SDO_WRITE4: Writes objects of no more than four bytes on specified devices.

2 Description of Function Blocks

General-purpose I/O and behaviors of function blocks

This section uses the CIA405 .RECV_EMCY function block as an example to describe regular management and execution of function blocks of the CAA CiA 405 library. The following describes I/O applicable to all function blocks. The function blocks are inherited from hidden function blocks of CiA405Base.

The following figure highlights the parameters shared by all function blocks of the CAA CiA 405 library.

The following table describes input parameters and output variables shared by all function blocks of the CAA CiA 405 library.

Parameter Name	Parameter Type	Initial Value	Description
VAR_IN			
NETWORK	USINT	1	Indicates the CAN channel on which requested services must be implemented. 1 (default): the first CAN bus interface 2: the second CAN bus interface (if any)
ENABLE	BOOL	FALSE	Enables the function block. Rising edge: Start the execution. Falling edge: If the execution is not finished, cancel the execution. Otherwise, output data is reset to 0.
TIMEOUT	UDINT	0	Indicates the maximum execution time, in the unit of ms. If the execution times out before response is received, the execution is aborted because of timeout. 0 (default): timeout disabled 1 to 65535: timeout value, in the unit of ms.
VAR_OUT			
CONFIRM	BOOL	FALSE	It is set to **TRUE** when the execution is completed.
ERROR	CANOPEN_ KERNEL_ERROR (USINT)	0	The execution of error codes returned from the CANopen kernel is included. 00 (hexadecimal): No execution errors are detected. 01 (hexadecimal): Errors are detected, but error codes are available on another function block rather than the CANopen kernel. 02 to FF (hexadecimal): Indicates error codes detected by the CANopen kernel.

Error codes detected by the CANopen kernel

The following table lists error codes and describes associated global variables.

Detected Error Code	Description
CANOPEN_KERNEL_NO_ERROR = 00 (hexadecimal)	No error codes are detected by the CANopen kernel.
CANOPEN_KERNEL_OTHER_ERROR = 01 (hexadecimal)	If **ERROR** is 01 (hexadecimal), errors are detected. If other error codes are output on the function block, specific data is included. Example: CIA405 .SDO_READ and CIA405.SDO_WRITE function blocks: The ERRORINFO output includes an SDO abortion message. CIA405 .RECV_EMCY and CIA405.RECV_EMCY_DEV function blocks: The ERRORINFO output includes received EMCY messages. If the ERRORINFO output includes EMCY messages, **ERROR** is 1 and **CONFIRM** is 0. CIA405 .GET_CANOPEN_KERNEL_STATE function block: No hexadecimal 01 value is provided because no other error codes are output.
CANOPEN_KERNEL_DATA_OVERFLOW = 02 (hexadecimal)	The CANopen object sending buffer or receiving buffer overflows.
CANOPEN_KERNEL_TIMEOUT = 03 (hexadecimal)	The function block execution timed out.
CANOPEN_KERNEL_CANBUS_OFF = 10 (hexadecimal)	CANopen nodes are disconnected from the CAN bus.
CANOPEN_KERNEL_CAN_ERROR_PASSIVE = 11 (hexadecimal)	CANopen nodes in passive state: Nodes can communicate but are not allowed to send motion error flags when errors are detected.
CANOPEN_INTERNAL_FB_ERROR = 21 (hexadecimal)	Indicates specific manufacturer error codes. Note: 21 (hexadecimal) to FF (hexadecimal) are dedicated for device manufacturers.

Diagram of the function block execution

Behaviors of control signals

The following describes three typical behaviors of ENABLE, CONFIRM, and ERROR control signals.

■   The execution ends when no errors are detected.

■   The execution is canceled by application programs.

■   The execution is aborted or ends when errors are detected.

The execution ends when no errors are detected.

When the system responds to the current request and no errors are detected, **CONFIRM** is set to **TRUE**, which is reserved (provided that the function block is called when **ENABLE** is set to **TRUE**). If the function block is called when **ENABLE** is reset to **FALSE**, **CONFIRM** is reset to **FALSE**, and the function block can be executed again.
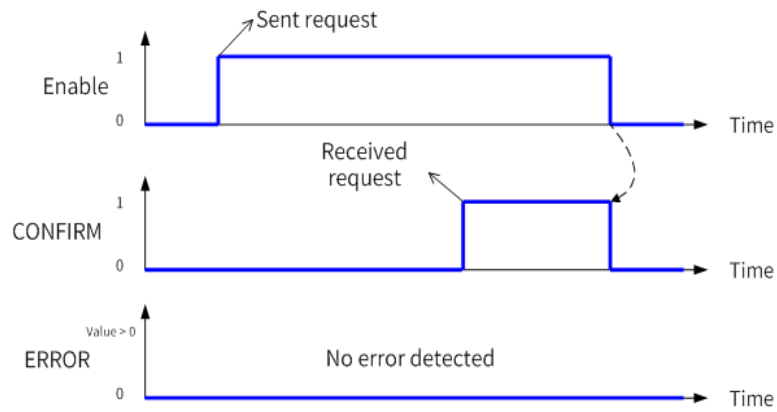
The execution is canceled by application programs.

Before the current execution ends, if the function block is called when **ENABLE** is reset to **FALSE**, the function block execution is canceled. The system may have responded to the canceled request or the response will be received later. However, the response is ignored.
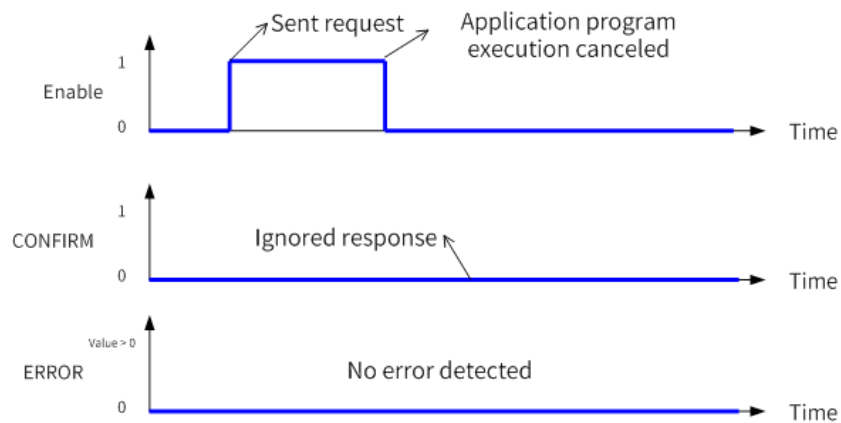
The execution is aborted or ends when errors are detected.

If the current execution is aborted because an SDO abortion message is received or errors are detected, **ERROR** is set to a non-zero value (for more details about detected error codes, see "Error codes detected by the CANopen kernel" on page 28). If the function block is called when **ENABLE** is reset to **FALSE**, **ERROR** is reset to **0**, and the function block can be executed again.
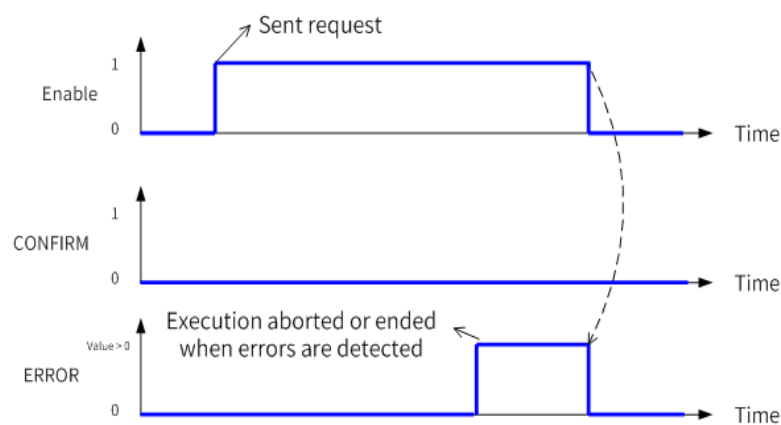
3 Network Management Function Block

1) Device NMT state management

Description of function blocks

NMT	Status Management
You can use the CIA405 .NMT function block to control the NMT state of a CANopen device through PLC application programs. The function block executes NMT service requests for target CANopen devices to implement request NMT state transition.	

Description of specific input parameters and output variables

Parameter Name	Parameter Type	Initial Value	Description
VAR_IN			
DEVICE	DEVICE(USINT)	0	Indicates the target CANopen device node ID. 0 (default): all NMT slaves 1 to 127: target CANopen device node ID
STATE	TRANSITION_STATE	FALSE	Indicates request NMT state transition.
VAR_OUT			
None	None	None	None

Graphic format

CIA405 .TRANSITION_STATE ENUM

The NMT state machine describes initialization and status of NMT slaves.

The following figure shows the NMT state, associated communication objects available (PDO, SDO, SYNC, EMCY, and NMT), and transition of five states (A to E).
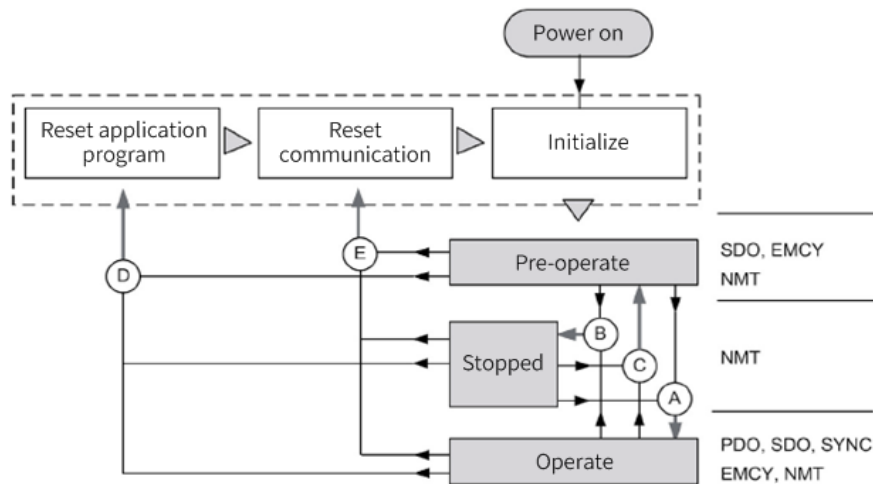
CIA405 .TRANSITION_STATE enumerated types include the NMT state transition command, as described in the following table.

Enumerator (hexadecimal)	Value (hexadecimal)	Description
STOP_REMOTE_NODE	0004	Switch to the Stopped state (transition B).
START_REMOTE_NODE	0005	Switch to the operational state (transition A).
RESET_NODE	0006	Switch to the application program reset state. Load saved data of the device configuration file and automatically switch from the communication reset state to pre-operation state (transition D).
RESET_COMMUNICATION	0007	Switch to the communication reset state. Load stored data of the communication configuration file and automatically switch from the communication reset state to pre-operation state (transition E).
ENTER_PRE_ OPERATIONAL	007F	Switch to the pre-operation state (transition C).
ALL_EXCEPT_NMT_AND_ SENDER	0800	Not implemented (invalid parameter)

Example

The following example describes how to send the "reset command" to the CANopen node 5 connected to the first CAN bus interface when the timeout is 1 second (1,000 ms). The command is sent when the Boolean variable EN_ResetNode is set to **TRUE** (set by users online or by application programs). When the execution succeeds (CONFIRM = TRUE) or errors are detected (ERROR <> 0), the **EN_ResetNode** command is reset to **FALSE**.
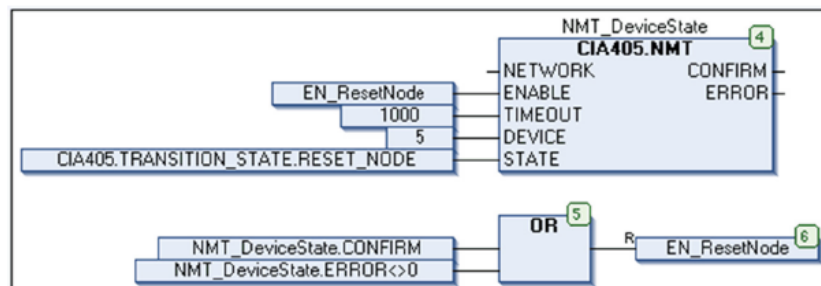
2)   EMCY message scan

Description of function blocks

RECV_EMCY	Message Scan
The CIA405.RECV_EMCY function block scans all EMCY messages stored on all existing CANopen devices in a cycle and returns EMCY messages found.	

Description of specific input parameters and output variables

Parameter Name	Parameter Type	Initial Value	Description
VAR_IN			
None	None	None	None
VAR_OUT			
DEVICE	DEVICE(USINT)	0	Indicates the node ID of a CANopen device associated with the returned EMCY message. 0: no EMCY messages found 1 to 127: CANopen device node ID
ERRORINFO	CS.EMCY_ ERROR	0	Indicates the last EMCY message received from the CANopen device whose node ID is DEVICE.

Graphic format

◆ After the function block is enabled, EMCY storage scan starts from the last scan stop point.

◆ If an EMCY message other than the No Error message is found, scan ends at this point, and the function block returns the EMCY message and the node ID of an associated device.

◆ If an EMCY message indicating no error is found but the message was not a No Error message, scan ends at this point, and the function block returns an EMCY message whose value is 0 and the node ID of an associated device.

◆ If no EMCY messages other than the No Error message are generated and no new No Error message is found in a complete scan process, the function block returns an EMCY message whose value is 0 and a device node ID whose value is 0.

3)　Obtaining device EMCY messages

Description of function blocks

RECV_EMCY_DEV	Obtaining device EMCY messages
The CIA405 .RECV_EMCY_DEV function block returns the last storage EMCY message received from a specified CANopen device.	

Description of specific input parameters and output variables

Parameter Name	Parameter Type	Initial Value	Description
VAR_IN			
DEVICE	DEVICE(USINT)	0	Indicates the node ID of a CANopen device to be checked. 0 (default): local device (PLC) 1 to 127: CANopen device node ID
VAR_OUT			
ERRORINFO	CS.EMCY_ERROR	0	Indicates the last EMCY message received from the CANopen device whose node ID is DEVICE.

Graphic format

CS.EMCY_ERROR structure

CS.EMCY_ERROR is associated with EMCY messages, including the following elements.

Element	Type	Description
EMCY_ERROR_CODE	Word	Indicates the error code of an EMCY message.
ERROR_REGISTER	Byte	Indicates the error register of an EMCY message (bit field).
ERROR_FIELD	ARRAY[1...5] OF BYTE	Indicates the specific device manufacturer error field of an EMCY message.
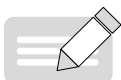
NOTE

The structure is declared in the CAA CANopen stack library (namespace = CS). Therefore, you need to use the full name (<namespace>.<data type>). The namespace of the CAA CiA 405 library is CIA405.

4 Self-owned Node ID Function Block

Obtain the PLC CANopen node ID.

Description of function blocks

GET_LOCAL_NODE_ID	Obtaining the PLC Node ID
The CIA405 .GET_LOCAL_NODE_ID function block returns the ID of the PLC CANopen node connected to a specified CAN bus interface. Graphic format	

Description of specific input parameters and output variables

Parameter Name	Parameter Type	Initial Value	Description
VAR_IN			
None	None	None	None
VAR_OUT			
DEVICE	DEVICE(USINT)	0	Indicates the PLC CANopen node ID. 0: invalid 1 to 127: PLC Node ID

Graphic format

5 State Query Function Block

1) Obtaining the CANopen kernel state

Description of function blocks

GET_CANOPEN_KERNEL_STATE	Obtaining the Kernel State
The CIA405 .GET_CANOPEN_KERNEL_STATE function returns the current state of the PLC CANopen kernel.	

Description of specific input parameters and output variables

Parameter Name	Parameter Type	Initial Value	Description
VAR_IN			
None	None	None	None
VAR_OUT			
STATE	CANOPEN_KERNEL_ ERROR		Indicates the current state of the PLC CANopen kernel.
DEVICE	DEVICE(USINT)	0	Indicates the PLC CANopen node ID. 0: invalid 1 to 127: PLC Node ID

Graphic format

2)   Obtaining the CANopen device state

Description of function blocks

GET _STATE	Obtaining the Device State
The CIA405 .GET_STATE function block returns the current NMT state of a specified CANopen device when heartbeat or node protection is activated.	

Description of specific input parameters and output variables

Parameter Name	Parameter Type	Initial Value	Description
			VAR_IN
DEVICE	DEVICE(USINT)	0	Indicates the node ID of a CANopen device to be checked. 0 (default): local device (PLC) 1 to 127: CANopen device node ID
			VAR_OUT
STATE	DEVICE_STATE	0	Indicates the NMT state of a CANopen device.

Graphic format

CIA405.DEVICE_STATE ENUM

CIA405 .DEVICE_STATE enumerated types include a list of CANopen device NMTstates.

Enumerator (hexadecimal)	Value	Description
INIT	0	Initializing state
RESET_COMM	1	Communication reset state
RESET_APP	2	Application program reset state
PRE_OPERATIONAL	3	Pre-operation state
STOPPED	4	Stopped state
OPERATIONAL	5	Operational state
UNKNOWN	6	Unknown NMT state The node protection or heartbeat function of the selected device is not activated, or the PLC is not a heartbeat consumer.
NOT_AVAIL	7	NMT state not available The node protection or heartbeat function of the selected device is activated, but the device fails to report its NMT state correctly before timeout.

Example

The following example describes how to obtain the state of the CANopen node 5 connected to the first CAN bus interface when the timeout is 1 second (1,000 ms). The CIA405.GET_STATE function is automatically executed to read the state continuously. The device NMT state is copied to DeviceState variables of the CIA405.DEVICE_STATE type.
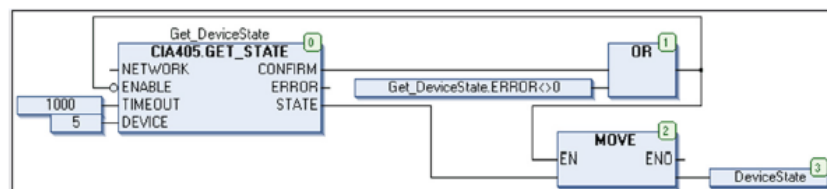
6 SDO Access Function Block

1) Reading CANopent objects of any size

Description of function blocks

SDO_READ	Reading CANopen objects of any size
The CIA405 .SDO_READ function block reads CANopen objects of any size on specified devices through SDO messages.	

Description of specific input parameters and output variables

Parameter Name	Parameter Type	Initial Value	Description
VAR_IN			
DEVICE	DEVICE (USINT)	0	Indicates the node ID of a CANopen device. 0 (default): local device (PLC) 1 to 127: CANopen device node ID
CHANNEL	USINT	1	Indicates the SDO channel number. The number is 1 by default.
INDEX	WORD	0	Indicates the object index. The value ranges from 0000 (hexadecimal) to FFFF (hexadecimal).
SUBINDEX	BYTE	0	Indicates the object subindex. The value ranges from 00 (hexadecimal) to FF (hexadecimal).
DATA	POINTER TO BYTE	NULL	Receives the data buffer address read from a device object. You need to use ADR standard functions to define the association pointer.
VAR_OUT			
ERRORINFO	SDO_ERROR (UDINT)	0	When **ERROR** is 1, an SDO abortion message (4 bytes) is returned.
VAR_IN/ VAR_OUT			
DATALENGTH	UINT	0	Input: Indicates the data buffer size (in the unit of bytes). Note: Use the SIZEOF standard function to initialize the DATALENGTH input correctly (when the function block execution is started at the rising edge through which **ENABLE** is input) to match the data buffer size. Output: Indicates the size of a read object (in the unit of bytes).

Graphic format

Example

The following example describes how to read the object index 203C (hexadecimal)/subindex 02 (hexadecimal) of the CANopen node 5 connected to the first CAN bus interface when the timeout is 1 second (1,000 ms). The CIA405.SDO_READ function block instance (ReadObject) is automatically executed for continuous reading.

Variable DataSize (UINT type):

It is initialized match the data buffer size (DataBuffer: N-byte array) (when **ENABLE** is **FALSE** and before the next execution is started).

The size of data read at the rising edge through which **CONFIRM** is output (in the unit of bytes) (the example does not demonstrate how to extract values from the data buffer or how to manage error detection) is included.
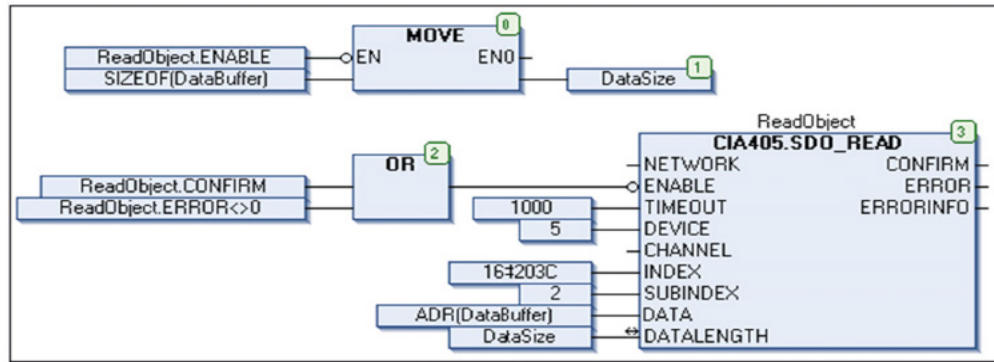
> **NOTE**
>
> The following parameters must be transmitted to the function block:
>
> 1) Device node ID
> 2) SDO client/server channel (by default, only one channel is defined)
> 3) CANopen object index/subindex
> 4) Pointer of the data buffer used to store object values
> 5) Data buffer size
>
> If Read operation succeeds, the function block returns the size of the read object. Data is available in the data buffer.
>
> If the object to be read contains four or less bytes, the CIA405.SDO_READ4 function block is recommended.

6) Reading CANopen Objects of No More Than Four Bytes

Description of function blocks

SDO_READ4	Reading CANopen Objects of No More Than Four Bytes
The CIA405 .SDO_READ4 function block reads CANopen objects of no more than four bytes on specified devices through SDO messages.	

Description of specific input parameters and output variables

Parameter Name	Parameter Type	Initial Value	Description
VAR_IN			
DEVICE	DEVICE (USINT)	0	Indicates the node ID of a CANopen device. 0 (default): local device (PLC) 1 to 127: CANopen device node ID
CHANNEL	USINT	1	Indicates the SDO channel number. The number is 1 by default.
INDEX	WORD	0	Indicates the object index. The value ranges from 0000 (hexadecimal) to FFFF (hexadecimal).
SUBINDEX	BYTE	0	Indicates the object subindex. The value ranges from 00 (hexadecimal) to FF (hexadecimal).

Parameter Name	Parameter Type	Initial Value	Description
VAR_OUT			
DATA	ARRAY[1...4] OF BYTE	0	Receives the data array read from a device object.
DATALENGTH	USINT	0	Indicates the size of a read object (in the unit of bytes).
ERRORINFO	SDO_ERROR (UDINT)	0	When **ERROR** is 1, an SDO abortion message (4 bytes) is returned.

Graphic format

The following table lists object sizes and corresponding DATA arrays.

Object Size Example	DATALENGTH	DATA(1)	DATA(2)	DATA(3)	DATA(4)
1-byte 01 (hexadecimal)	1	01 (hexadecimal)	Invalid	Invalid	Invalid
2-byte 01 23 (hexadecimal)	2	LSB 23 (hexadecimal)	MSB 01 (hexadecimal)	Invalid	Invalid
3-byte 01 23 45 (hexadecimal)	3	LSB 45 (hexadecimal)	23 (hexadecimal)	MSB 01 (hexadecimal)	Invalid
4-byte 01 23 45 67 (hexadecimal)	4	LSB 67 (hexadecimal)	45 (hexadecimal)	23 (hexadecimal)	MSB 01 (hexadecimal)

LSB = least significant bit

MSB = most significant bit

Example

The following example describes how to read the object index 203C (hexadecimal)/subindex 02 (hexadecimal) of the CANopen node 5 connected to the first CAN bus interface when the timeout is 1 second (1,000 ms). The CIA405.SDO_READ4 function block instance (Read4Object) is automatically executed for continuous reading. The variable DataBuffer (4-byte array) includes the value of data last read. The variable DataSize (USINT type) includes the size of data last read (maximum: 4 bytes). The example does not demonstrate how to manage error detection.
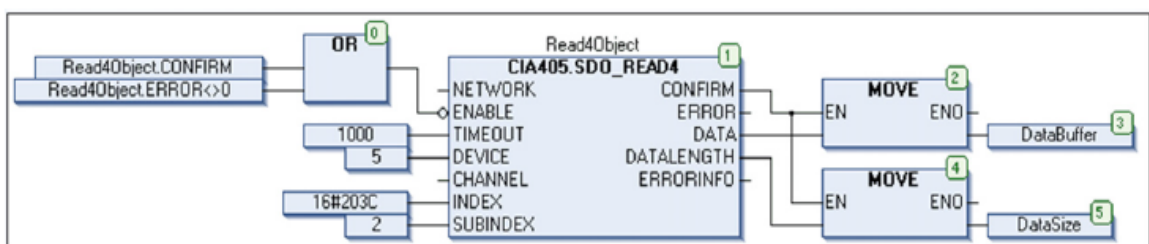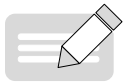
NOTE

The following parameters must be transmitted to the function block:

1) Device node ID

2) SDO client/server channel (by default, only one channel is defined)

3) CANopen object index/subindex

If Read operation succeeds, the function block returns the size of the read object. Data is available in a 4-byte array.

4) Writing CANopen objects of any size

Description of function blocks

SDO_WRITE	Writing CANopen objects of any size
The CIA405 .SDO_WRITE function block writes CANopen objects of any size on specified devices through SDO messages.	

Description of specific input parameters and output variables

Parameter Name	Parameter Type	Initial Value	Description
VAR_IN			
DEVICE	DEVICE (USINT)	0	Indicates the node ID of a CANopen device. 0 (default): local device (PLC) 1 to 127: CANopen device node ID
CHANNEL	USINT	1	Indicates the SDO channel number. The number is 1 by default.
INDEX	WORD	0	Indicates the object index. The value ranges from 0000 (hexadecimal) to FFFF (hexadecimal).
SUBINDEX	BYTE	0	Indicates the object subindex. The value ranges from 00 (hexadecimal) to FF (hexadecimal).
MODE	SDO_MODE	0	Indicates the data transmission mode. 0 (default) = AUTO (automatic mode selection)
DATA	POINTER TO BYTE	NULL	Stores the address of the data buffer for object values to be written. You need to use ADR standard functions to define the association pointer.
DATALENGTH	UINT	0	Indicates the size of an object to be written (in the unit of bytes).
VAR_OUT			
ERRORINFO	SDO_ERROR (UDINT)	0	When **ERROR** is 1, an SDO abortion message (4 bytes) is returned.

Graphic format

CIA405 .SDO_MODE ENUM

CIA405 .SDO_MODE enumerated types include a list of SDO transmission modes.

Enumerator (hexadecimal)	Value	Description
AUTO	0	Indicates automatic mode selection.
EXPEDITED	1	Indicates the SDO acceleration mode for data of no more than four bytes. Data is sent in response to an SDO request.
SEGMENTED	2	Indicates the SDO segmentation mode for data of more than four bytes. Data is segmented into 7-byte segments and sent through continuous SDO confirmation requests.
BLOCK	3	Indicates the SDO block mode for data of more than four bytes. Data sent through continuous frames is segmented into 7-byte data blocks. These blocks are not confirmed. Upon receipt of all blocks, the recipient sends confirmation. Note: This mode allows quick transmission, but your device may not support the mode. Therefore, the mode is a CANopen configuration item recently added.
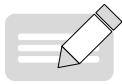
NOTE

The following parameters must be transmitted to the function block:

1) Device node ID
2) SDO client/server channel (by default, only one channel is defined)
3) CANopen object index/subindex
4) SDO mode
5) Pointer used to store the data buffer for object values to be written
6) Number of bytes to be written

If the object to be written contains four or less bytes, use the CIA405.SDO_WRITE4 function block.

7) Writing CANopen Objects of No More Than Four Bytes

Description of function blocks

SDO_WRITE4	Writing CANopen Objects of No More Than Four Bytes
The CIA405 .SDO_WRITE4 function block writes CANopen objects of no more than four bytes on specified devices through SDO messages.	

Description of specific input parameters and output variables

Parameter Name	Parameter Type	Initial Value	Description
VAR_IN			
DEVICE	DEVICE (USINT)	0	Indicates the node ID of a CANopen device. 0 (default): local device (PLC) 1 to 127: CANopen device node ID
CHANNEL	USINT	1	Indicates the SDO channel number. The number is 1 by default.
INDEX	WORD	0	Indicates the object index. The value ranges from 0000 (hexadecimal) to FFFF (hexadecimal).
SUBINDEX	BYTE	0	Indicates the object subindex. The value ranges from 00 (hexadecimal) to FF (hexadecimal).
DATA	ARRAY[1...4] OF BYTE	0	Stores the data array of the object value to be written.
DATALENGTH	USINT	0	Indicates the size of an object to be written (in the unit of bytes).
VAR_OUT			
ERRORINFO	SDO_ERROR (UDINT)	0	When **ERROR** is 1, an SDO abortion message (4 bytes) is returned.

Graphic format

The following table lists object sizes and corresponding DATA arrays.

Object Size Example	DATALENGTH	DATA(1)	DATA(2)	DATA(3)	DATA(4)
1-byte 01 (hexadecimal)	1	01 (hexadecimal)	Invalid	Invalid	Invalid
2-byte 01 23 (hexadecimal)	2	LSB 23 (hexadecimal)	MSB 01 (hexadecimal)	Invalid	Invalid
3-byte 01 23 45 (hexadecimal)	3	LSB 45 (hexadecimal)	23 (hexadecimal)	MSB 01 (hexadecimal)	Invalid
4-byte 01 23 45 67 (hexadecimal)	4	LSB 67 (hexadecimal)	45 (hexadecimal)	23 (hexadecimal)	MSB 01 (hexadecimal)

LSB = least significant byte

MSB = most significant byte

NOTE

The following parameters must be transmitted to the function block:

1) Device node ID
2) SDO client/server channel (by default, only one channel is defined)
3) CANopen object index/subindex
4) Values to be written
5) Number of bytes to be written (object size)

7 Terminology

Name	Definition
CAN	CAN is a serial bus system designed for embedded control.
CANopen	CANopen is a CAN-based high-level protocol used for embedded control systems and compliant with the international standard (EN 50325-4).
CiA	CAN in Automation (CiA) is an international organization composed of users and manufacturers. It is designed to develop and support CANopen and other CAN-based high-level protocols.
CiA405	IEC 61131-3 PLC CANopen interface and device configuration file
COB	CANopen protocol communication object
EMCY	Emergency
NMT	CANopen network management
OD	Protocol object dictionary
PDO	Protocol process data object
SDO	Protocol service data object

6.3.2 CANopen 402

Designed based on PLCopen and CiA402 standards, the function block supports CiA402 linear control modes, such as the profile speed mode, profile position mode, and homing mode. It does not support rotation mode, soft axis limit, or hardware limit. In linear control mode, the position ranges from -2147483648 to 2147483647. Axis state switchover is designed based on the diagram of PLCopen state machine, as shown in the following figure.
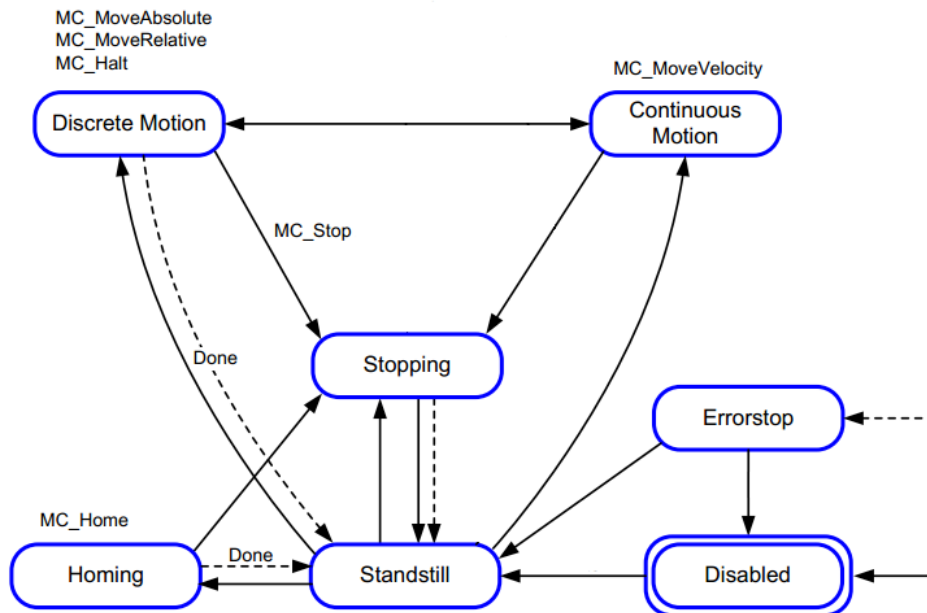
1 MC_Power_CO

1) Instruction Format

Graphic Expression

Instruction	Name	ST expression

MC_Power_CO	Motor enabling	MC_Power_CO(
		        Axis:= ,
		        Enable:= ,
		        bRegulatorOn:= ,
		        bDriveStart:= ,
		        Status=> ,
		        bRegulatorRealState=> ,
		        bDriveStartRealState=> ,
		        Busy=> ,
		        Error=> ,
		ErrorID=>);

2) Relevant Variables

■ I/O variable

I/O Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis	Axis	AXIS_REF_HC_CO	N/A	N/A	Indicates a CANopen axis.

■ Input variable

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Enable	Enable	BOOL	[FALSE,TRUE]	FALSE	(Triggered by level) TRUE: Enables the function block.
bRegulatorOn	Motor enabling	BOOL	[FALSE,TRUE]	FALSE	Enables the motor (used with DriveStart).
bDriveStart	Quick emergency stop	BOOL	[FALSE,TRUE]	FALSE	Quick emergency stop is not supported.

■ Output variable

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Status	Axis motion preparation status	BOOL	[FALSE,TRUE]	FALSE	Indicates the axis motion preparation status.
bRegulatorRealState	Enable valid state	BOOL	[FALSE,TRUE]	FALSE	Indicates whether the drive is enabled (TRUE: Enabled).
bDriveStartRealState	Valid state of the quick stop mechanism	BOOL	[FALSE,TRUE]	FALSE	Indicates emergency stop, not supported currently.
Busy	Execution flag	BOOL	[FALSE,TRUE]	FALSE	**TRUE** indicates that the function block is being executed.
Error	Error flag	BOOL	[FALSE,TRUE]	FALSE	**TRUE** indicates that an error occurs inside the function block.
ErrorID	Error ID	ERROR_CO	N/A	NO_ERROR	Indicates the error ID (see ERROR_CO for details).

3) Function

■ The function block is mainly used for axis enabling/disabling and quick emergency stop. Use of the quick emergency stop function depends on whether the drive supports the function.

■ If **RegulatorRealState** is **TRUE**, the drive is enabled but not necessarily controllable. Motion control can be implemented only when **Status** is **TRUE**.

■ The quick stop mode depends on the value of the object dictionary 16#605A. To change the stop mode, set the 16#605A parameter, depending on whether the drive supports the function.

4)   Sequence Diagram

5)   Error Description

When an error occurs, locate the error corresponding to **ErrorID** in **COUNTER_ERROR** through the help document, and find error causes.

Error ID	Enumerated Value	Description
0	NO_ERROR	No error
1	DI_GENERAL_COMMUNICATION_ERROR	Axis communication error
2	DI_AXIS_ERROR	Drive axis error
21	WRONG_OPMODE	Incorrect operation mode
33	AXIS_IN_ERRORSTOP	Axis in ErrorStop state
34	AXIS_NOT_READY_FOR_MOTION	Axis not ready for motion
35	MA_MR_MODULO_ACT_POS_NOT_MAPPED	Reserved
36	MV_INVALID_VELACCDEC_VALUES	Invalid speed or acceleration/deceleration
80	RAG_ERROR_DURING_STARTUP	Reserved
81	RAG_ERROR_WRITING_COMSTATE	Reserved
82	RAG_ERROR_READING_COMSTATE	Reserved
90	CGR_ZERO_VALUES	Reserved
91	CGR_AXIS_POWERED	Reserved
93	CGR_MODULOPERIOD_NOT_INTEGRAL	Reserved
94	CGR_MOVEMENTTYPE_INVALID	Reserved
95	CGR_MODULOPERIOD_NON_POSITIVE	Reserved
96	CGR_MODULOPERIOD_TOO_SMALL	Reserved
97	CGR_MODULOPERIOD_TOO_LARGE	Reserved
120	R_NO_ERROR_TO_RESET	No error to reset
121	R_DRIVE_DOESNT_ANSWER	No answer
122	R_ERROR_NOT_RESETTABLE	Error not resettable
123	R_DRIVE_DOESNT_ANSWER_IN_TIME	Reserved
130	RP_PARAM_UNKNOWN	Read parameter error
131	RP_REQUESTING_ERROR	Communication request error
132	RP_RCV_PARAM_CONVERSION_ERROR	Reserved
133	RP_LOCAL_PARAM_NOT_DONE_IMMEDIATELY	Reserved
134	RP_CANNOT_SEND_MSG	Unable to send messages

Error ID	Enumerated Value	Description
140	WP_PARAM_INVALID	Write parameter error
141	WP_SENDING_ERROR	Sending error
142	WP_TMT_PARAM_CONVERSION_ERROR	Reserved
143	WP_LOCAL_PARAM_NOT_DONE_IMMEDIATELY	Reserved
144	WP_CANNOT_SEND_MSG	Unable to send messages
170	H_AXIS_WASNT_STANDSTILL	Axis in Standstill state for homing
183	MS_AXIS_IN_ERRORSTOP	ErrorStop state not allowed for the mc_stop function block
184	MS_AXIS_IN_STOPPING	Stopping state not allowed for the mc_stop function block
10000	TIMEOUT_CHANGING_OPMODE	Mode switching timeout
10001	INTERNAL_UNKNOWN_CMD	Reserved
10002	CANNOT_START_MOVEMENT	Reserved
10003	CANNOT_START_HOMING	Reserved
10004	STOP_ALREADY_ACTIVE	Reserved
10005	POWER_ALREADY_ACTIVE	Reserved
10006	SMC_DI_VOLTAGE_DISABLED	Axis motion process interrupt enabled

2 MC_MoveAbsolute_CO

1) Instruction Format

Graphic Expression		

Instruction	Name	ST expression

MC_MoveAbsolute_CO	Absolute positioning	MC_MoveAbsolute_CO(
		    Axis:= ,
		    Execute:= ,
		    Position:= ,
		    Velocity:= ,
		    Acceleration:= ,
		    Deceleration:= ,
		    Done=> ,
		    Busy=> ,
		    CommandAborted=> ,
		    Error=> ,
		    ErrorID=>);

2) Relevant Variables

■ I/O variable

I/O Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis	Axis	AXIS_REF_HC_CO	N/A	N/A	Indicates a CANopen axis.

■ Input variable

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Execute	Execution input	BOOL	[FALSE,TRUE]	FALSE	(Triggered by rising edges) FALSE to TRUE: Execution starts.
Position	Target position	LREAL	(0, 1.7E 308)	0.0	Unit: unit, only linear mode supported
Velocity	Target speed	LREAL	(0, 1.7E 308)	0.0	Unit: unit/s, only linear mode supported
Acceleration	Target acceleration	LREAL	(0, 1.7E 308)	0.0	Unit: unit/s^2, only linear mode supported
Deceleration	Target acceleration	LREAL	(0, 1.7E 308)	0.0	Unit: unit/s^2, only linear mode supported

■ Output variable

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Done	Complete flag	BOOL	[FALSE,TRUE]	FALSE	TRUE: The motion execution is completed, and positioning succeeds.
Busy	Execution flag	BOOL	[FALSE,TRUE]	FALSE	TRUE: Motion is being executed.
CommandAborted	Abort flag	BOOL	[FALSE,TRUE]	FALSE	TRUE: Motion is interrupted.
Error	Error flag	BOOL	[FALSE,TRUE]	FALSE	TRUE: An error occurs during motion.
ErrorID	Error ID	ERROR_CO	N/A	NO_ERROR	Indicates the error ID (see ERROR_CO for details).

3) Function

■ The controlled axis is moved to a specified absolute position through the absolute positioning function. The pulse count for absolute positioning ranges from -2147483648 to 2147483647. If the pulse count is out of range, an error occurs during positioning. The motion function block cannot run normally.

■ **Velocity**, **Acceleration**, and **Deceleration** values must be greater than 0.

■ If the positioning length is out of range (-2147483648 to 2147483647), reduce the servo electronic gear ratio coefficient or lower the stepper drive.

■ When positioning is completed, **Done** is set to **TRUE** and the controlled axis is in Standstill state.

4) Sequence Diagram
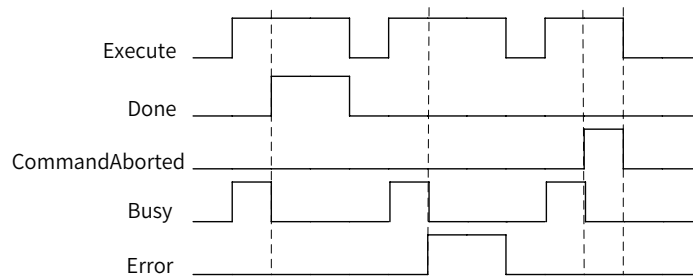
5) Error Description

When an error occurs, locate the error corresponding to **ErrorID** in **COUNTER_ERROR** through the help document, and find error causes.

3 MC_MoveRelative_CO

1) Instruction Format

Graphic Expression		
Instruction	Name	ST expression

MC_MoveRelative_CO	Relative positioning	MC_MoveRelative_CO(
		        Axis:= ,
		        Execute:= ,
		        Distance:= ,
		        Velocity:= ,
		        Acceleration:= ,
		        Deceleration:= ,
		        Done=> ,
		        Busy=> ,
		        CommandAborted=> ,
		        Error=> ,
		        ErrorID=>);

2) Relevant Variables

■ I/O variable

I/O Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis	Axis	AXIS_REF_HC_CO	N/A	N/A	Indicates a CANopen axis.

■ Input Variable

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Execute	Execution input	BOOL	[FALSE,TRUE]	FALSE	(Triggered by rising edges) FALSE to TRUE: Execution starts.
Distance	Indicates the relative distance.	LREAL	(0, 1.7E 308)	0.0	Unit: unit, only linear mode supported
Velocity	Target speed	LREAL	(0, 1.7E 308)	0.0	Unit: unit/s, only linear mode supported
Acceleration	Target acceleration	LREAL	(0, 1.7E 308)	0.0	Unit: unit/s^2, only linear mode supported
Deceleration	Target acceleration	LREAL	(0, 1.7E 308)	0.0	Unit: unit/s^2, only linear mode supported

■ Output variable

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Done	Complete flag	BOOL	[FALSE,TRUE]	FALSE	TRUE: The motion execution is completed, and positioning succeeds.
Busy	Execution flag	BOOL	[FALSE,TRUE]	FALSE	TRUE: Motion is being executed.
CommandAborted	Abort flag	BOOL	[FALSE,TRUE]	FALSE	TRUE: Motion is interrupted.
Error	Error flag	BOOL	[FALSE,TRUE]	FALSE	TRUE: An error occurs during motion.
ErrorID	Error ID	ERROR_CO	N/A	NO_ERROR	Indicates the error ID (see ERROR_CO for details).

3) Function

■ The relative positioning function block moves the controlled axis a distance (Motion target position = Standstill position when the function block is executed + Relative distance). If the relative positioning instruction is started while the axis is moving under the absolute positioning instruction, the motion target position is the absolutely defined target position plus relative distance.

■ If the positioning length is out of range (-2147483648 to 2147483647), reduce the servo electronic gear ratio coefficient or lower the stepper drive.

■ When the device is used with Inovance IS620_CO, if the relative positioning command or absolute positioning command being executed is interrupted by the relative motion command, the absolute target position calculated through a relative command is the position the axis moving relatively or absolutely should reach plus the relative position for the motion instruction.

■ When positioning is completed, **Done** is set to **TRUE** and the controlled axis is in Standstill state.

4) Sequence Diagram
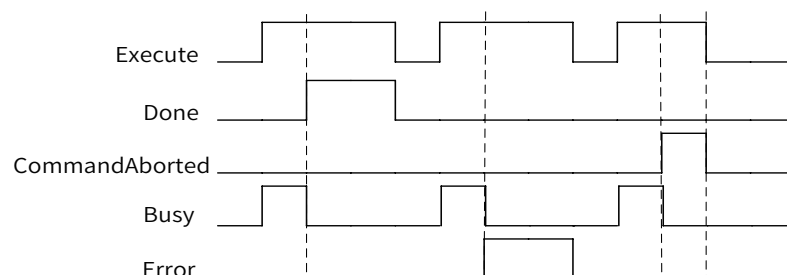
5) Error Description

When an error occurs, locate the error corresponding to **ErrorID** in **COUNTER_ERROR** through the help document, and find error causes.

4 MC_MoveVelocity_CO

1) Instruction Format

Graphic Expression		
Instruction	Name	ST expression

MC_MoveVelocity_CO	Motion in speed mode	MC_MoveVelocity_CO(
		    Axis:= ,
		    Execute:= ,
		    Velocity:= ,
		    Acceleration:= ,
		    Deceleration:= ,
		    InVelocity=> ,
		    Busy=> ,
		    CommandAborted=> ,
		    Error=> ,
		    ErrorID=>);

2) Relevant Variables

■ I/O variable

I/O Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis	Axis	AXIS_REF_HC_CO	N/A	N/A	Indicates a CANopen axis.

■ Input Variable

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Execute	Execution input	BOOL	[FALSE,TRUE]	FALSE	(Triggered by rising edges) FALSE to TRUE: Execution starts.
Velocity	Target speed	LREAL	(-1.7E 308, 1.7E 308)	0.0	Unit: unit/s, only linear mode supported
Acceleration	Target acceleration	LREAL	(0, 1.7E 308)	0.0	Unit: $unit/s^2$, only linear mode supported
Deceleration	Target acceleration	LREAL	(0, 1.7E 308)	0.0	Unit: $unit/s^2$, only linear mode supported

■ Output variable

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
InVelocity	Speed reached flag	BOOL	[FALSE,TRUE]	FALSE	TRUE: The axis speed reaches the preset value.

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Busy	Execution flag	BOOL	[FALSE,TRUE]	FALSE	TRUE: Motion is being executed.
CommandAborted	Abort flag	BOOL	[FALSE,TRUE]	FALSE	TRUE: Motion is interrupted.
Error	Error flag	BOOL	[FALSE,TRUE]	FALSE	TRUE: An error occurs during motion.
ErrorID	Error ID	ERROR_CO	N/A	NO_ERROR	Indicates the error ID (see ERROR_CO for details).

3) Function

■ On the function block for motion in speed mode, after the speed reaches the preset value, the controlled axis runs at the speed constantly. To stop the axis, you need to interrupt and abort the current function block by using another command. When the function block is interrupted by another command, the output parameter **InVelocity** must be reset.

■ The position varies in the range of -2147483648 to 2147483647.

■ When the device is used with IS620N_CO, if the position overflows (the pulse position changes from 2147483647 to -2147483648 or from -2147483648 to 2147483647) in motion mode, return the axis to home, switch to the position mode, and then use the position motion command because the overflow count is not memorized in position mode (different from the synchronization cycle mode).

■ During normal motion, the controlled axis is in ContinuesMoiton state.

4) Sequence Diagram
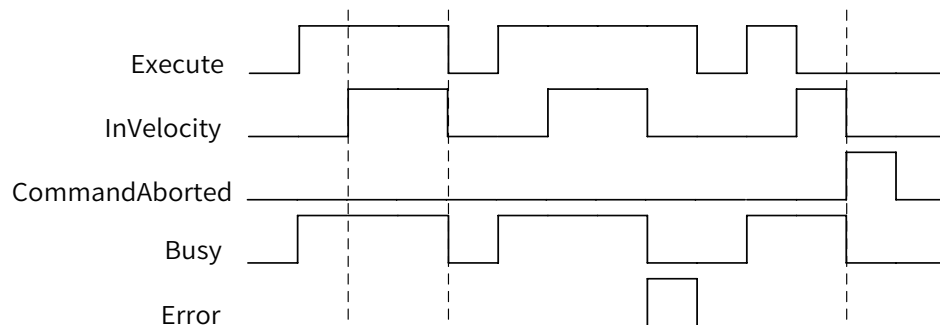
5) Error Description

When an error occurs, locate the error corresponding to **ErrorID** in **COUNTER_ERROR** through the help document, and find error causes.

5 MC_Home_CO

1) Instruction Format
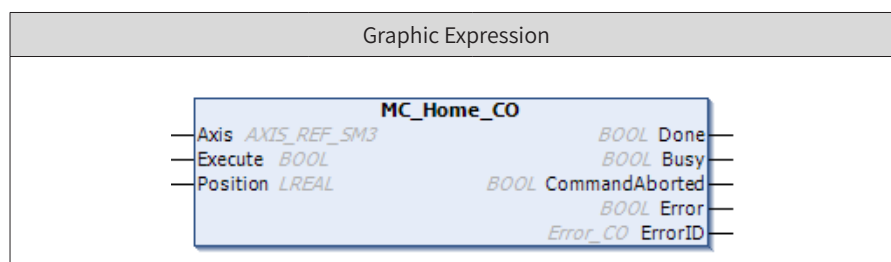
Graphic Expression		
Instruction	Name	ST expression

MC_Home_CO	Indicates homing motion.	```MC_Home_CO(
		    Axis:= ,
		    Execute:= ,
		    Position:= ,
		    Done=> ,
		    Busy=> ,
		    CommandAborted=> ,
		    Error=> ,
		    ErrorID=>);```

2) Relevant Variables

■ I/O variable

I/O Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis	Axis	AXIS_REF_HC_CO	N/A	N/A	Indicates a CANopen axis.

■ Input Variable

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Execute	Execution input	BOOL	[FALSE,TRUE]	FALSE	(Triggered by rising edges) FALSE to TRUE: Execution starts.
position	Home offset	LREAL	(0, 1.7E 308)	0.0	Unit: unit/s, only linear mode supported

■ Output variable

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Done	Complete flag	BOOL	[FALSE,TRUE]	FALSE	TRUE: Homing is completed.
Busy	Execution flag	BOOL	[FALSE,TRUE]	FALSE	TRUE: Motion is being executed.
CommandAborted	Abort flag	BOOL	[FALSE,TRUE]	FALSE	TRUE: Motion is interrupted.
Error	Error flag	BOOL	[FALSE,TRUE]	FALSE	TRUE: An error occurs during motion.
ErrorID	Error ID	ERROR_CO	N/A	NO_ERROR	Indicates the error ID (see ERROR_CO for details).

3) Function

■ On the homing function block, the controlled axis returns to home in preset mode. The axis stops when reference signals are detected. The current absolute position is updated to the value of the input parameter **position**.

■ Homing cannot be interrupted by the positioning command or speed command.

■ During normal motion, the controlled axis is in Homing state.

■ The position home offset is the distance the axis deviates from the absolute position of home (unit: unit). When the hardware model is detected and homing stops, the current absolute position is set to

position, and the controlled axis is in Standstill state.
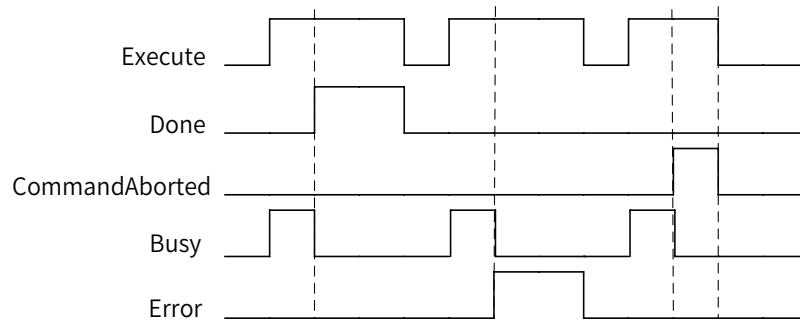
4) Sequence Diagram

5) Error Description

When an error occurs, locate the error corresponding to **ErrorID** in **COUNTER_ERROR** through the help document, and find error causes.

6 MC_Stop_CO

1) Instruction Format

Graphic Expression		
Instruction	Name	ST expression

MC_Stop_CO	Motion stop	MC_Stop_CO(
		   Axis:= ,
		   Execute:= ,
		   Done=> ,
		   Busy=> ,
		   Error=> ,
		   ErrorID=>);

2) Relevant Variables

■ I/O variable

I/O Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis	Axis	AXIS_REF_HC_CO	N/A	N/A	Indicates a CANopen axis.

■ Input variable

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Execute	Execution input	BOOL	[FALSE,TRUE]	FALSE	(Triggered by rising edges) FALSE to TRUE: Execution starts.

■ Output variable

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Done	Complete flag	BOOL	[FALSE,TRUE]	FALSE	TRUE: The axis stops.
Busy	Execution flag	BOOL	[FALSE,TRUE]	FALSE	TRUE: The stop command is being executed.
Error	Error flag	BOOL	[FALSE,TRUE]	FALSE	TRUE: A stop command error occurred.
ErrorID	Error ID	ERROR_CO	N/A	NO_ERROR	Indicates the error ID (see ERROR_CO for details).

3) Function

■ On the motion stop function block, after execution is triggered by rising edges, the controlled axis stops. If **Execute** is **TRUE** or the axis is not stopped, no other motion commands can be sent, and the axis is in Stopping state. After the axis is stopped and **Execute** is reset, the axis is in Standstill state.

■ As many manufacturers do not support execution stop through the control word bit 8. Through the stop function, the system switches to the speed mode, and you can set the target speed to 0 and stop motion by running speed commands.

■ After motion stops, you need to reset the Execute state. Otherwise, you cannot run other motion commands.

4) Sequence Diagram
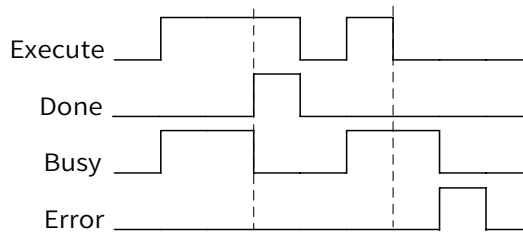
5) Error Description

When an error occurs, locate the error corresponding to **ErrorID** in **COUNTER_ERROR** through the help document, and find error causes.

7 MC_Halt_CO

1) Instruction Format

	Graphic Expression	

Instruction	Name	ST expression

MC_Halt_CO	Motion stop	MC_Halt_CO(
		    Axis:= ,
		    Execute:= ,
		    Deceleration:= ,
		    Done=> ,
		    CommandAborted=> ,
		    Busy=> ,
		    Error=> ,
		    ErrorID=>);

2) Relevant Variables

■ I/O variable

I/O Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis	Axis	AXIS_REF_HC_CO	N/A	N/A	Indicates a CANopen axis.

■ Input variable

Input variable	Name	Data Type	Valid Range	Initial Value	Description
Execute	Port for function block execution	BOOL	[FALSE,TRUE]	FALSE	(Triggered by rising edges) FALSE to TRUE: Execution starts.
Deceleration	Deceleration	LREAL	(0, 1.7E 308)	0.0	Unit: $unit/s^2$, only linear mode supported

■ Output variable

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Done	Complete flag	BOOL	[FALSE,TRUE]	FALSE	TRUE: The axis stops.
Busy	Execution flag	BOOL	[FALSE,TRUE]	FALSE	TRUE: The stop command is being executed.
CommandAborted	Abort flag	BOOL	[FALSE,TRUE]	FALSE	TRUE: Motion is interrupted.
Error	Error flag	BOOL	[FALSE,TRUE]	FALSE	TRUE: A stop command error occurred.
ErrorID	Error ID	ERROR_CO	N/A	NO_ERROR	Indicates the error ID (see ERROR_CO for details).

3) Function

■ When the axis is stopped through commands, the axis is in DiscreteMotion state until the speed is zero. When **Done** is **TRUE**, the axis is in Standstill state.

■ During the MC_Halt_CO execution stop, MC_Halt_CO can be interrupted by other motion commands.

■ After the MC_Halt_CO execution is completed, unlike MC_Stop_CO, you can use other motion commands without resetting the Execute port state.

4) Sequence Diagram
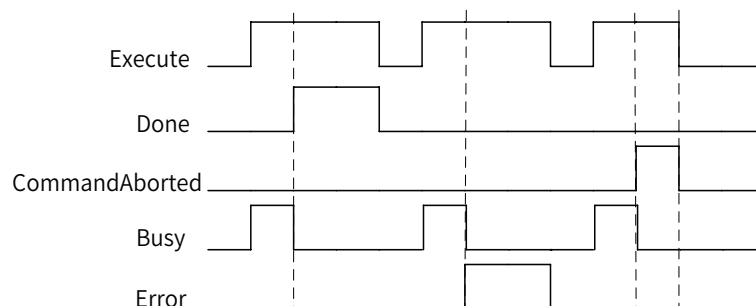
5) Error Description

When an error occurs, locate the error corresponding to **ErrorID** in **COUNTER_ERROR** through the help document, and find error causes.

8 MC_Reset_CO

1) Instruction Format

Graphic Expression

MC_Reset_CO

Axis AXIS_REF_SM3	BOOL Done
Execute BOOL	BOOL Busy
	BOOL Error
	Error_CO ErrorID

Instruction	Name	ST expression
MC_Reset_CO	Resets the axis.	MC_Reset_CO(Axis:= , Execute:= , Done=> , Busy=> , Error=> , ErrorID=>);

2) Relevant Variables

■ I/O variable

I/O Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis	Axis	AXIS_REF_HC_CO	N/A	N/A	Indicates a CANopen axis.

■ Input variable

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Execute	Execution input	BOOL	[FALSE,TRUE]	FALSE	(Triggered by rising edges) FALSE to TRUE: Execution starts.

■ Output variable

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Done	Complete flag	BOOL	[FALSE,TRUE]	FALSE	TRUE: Reset is completed.
Busy	Execution flag	BOOL	[FALSE,TRUE]	FALSE	TRUE: The reset command is being executed.
Error	Error flag	BOOL	[FALSE,TRUE]	FALSE	TRUE: A reset command error occurred.
ErrorID	Error ID	ERROR_CO	N/A	NO_ERROR	Indicates the error ID (see ERROR_CO for details).

3) Function

■ Reset all errors relating to internal axes, and forcibly switch axes from the ErrorStop state to the Standstill or PowerOff state. If you perform the reset action when the controlled axis is not in ErrorStop state, the function block reports an error.

■ Motion errors and errors for which the servo supports the reset action can be reset.

4) Sequence Diagram
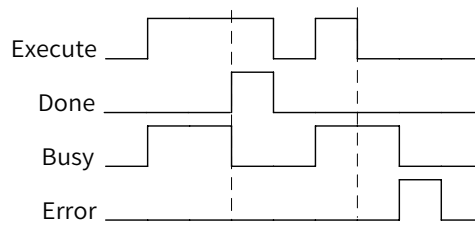
5) Error Description

When an error occurs, locate the error corresponding to **ErrorID** in **COUNTER_ERROR** through the help document, and find error causes.

9 MC_WriteParameter_CO

1) Instruction Format

Graphic Expression

Instruction	Name	ST expression

MC_WriteParameter_CO	Write remote device parameter	MC_WriteParameter_CO(
		    Axis:= ,
		    Execute:= ,
		    ParameterNumber:= ,
		    Value:= ,
		    Done=> ,
		    Busy=> ,
		    Error=> ,
		    ErrorID=>);

2) Relevant Variables

■ I/O variable

I/O Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis	Axis	AXIS_REF_HC_CO	N/A	N/A	Indicates a CANopen axis.

■ Input variable

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Execute	Execution input	BOOL	[FALSE,TRUE]	FALSE	(Triggered by rising edges) FALSE to TRUE: Execution starts.
ParameterNumber	Parameter	DINT	[0,2^32)	0	Indicates the parameter number, which should be operated in combined mode.
Value	Value	LREAL	(-1.7E 308, 1.7E 308)	0.0	Writes a value.

■ Output variable

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Done	Complete flag	BOOL	[FALSE,TRUE]	FALSE	TRUE: Reset is completed.
Busy	Execution flag	BOOL	[FALSE,TRUE]	FALSE	TRUE: The command is being executed.
Error	Error flag	BOOL	[FALSE,TRUE]	FALSE	TRUE: An error occurs on the function block.
ErrorID	Error ID	ERROR_CO	N/A	NO_ERROR	Indicates the error ID (see ERROR_CO for details).

3) Function

■ Write object dictionary data: Based on CANopen parameters, the specific axis parameter number consists of the object dictionary length 16 # UV (1 byte), object dictionary index 16 # ABCD (2 bytes) and subindex 16 # EF (1 byte), which form the -16 # UVABCDEF mode. Example: If the write object index value is 16 # 607C, the subindex value is 16#00, and the data length value is 16#04, the parameter number is -16 # 04607C00.

4) Sequence Diagram
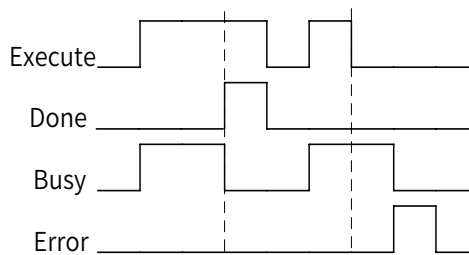
5) Error Description

When an error occurs, locate the error corresponding to **ErrorID** in **COUNTER_ERROR** through the help document, and find error causes.

10 MC_ReadParameter_CO

1) Instruction Format

Graphic Expression		
MC_ReadParameter_CO		

```
MC_ReadParameter_CO
Axis  AXIS_REF_SM3           BOOL  Valid
Enable  BOOL                 BOOL  Busy
ParameterNumber  DINT        BOOL  Error
                        Error_CO  ErrorID
                           LREAL  Value
```

Instruction	Name	ST expression
MC_ReadParameter_CO	Read remote device parameter	MC_ReadParameter_CO(　　Axis:= , 　　Enable:= , 　　ParameterNumber:= , 　　Valid=> , 　　Busy=> , 　　Error=> , 　　ErrorID=> , 　　Value=>);

2) Relevant Variables

■ I/O variable

I/O Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis	Axis	AXIS_REF_HC_CO	N/A	N/A	Indicates a CANopen axis.

■ Input variable

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Execute	Execution input	BOOL	[FALSE,TRUE]	FALSE	(Triggered by rising edges) FALSE to TRUE: Execution starts.
ParameterNumber	Parameter	DINT	[0,2^32)	0	Indicates the parameter number, which should be operated in combined mode.

■ Output variable

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Done	Complete flag	BOOL	[FALSE,TRUE]	FALSE	TRUE: Reset is completed.
Busy	Execution flag	BOOL	[FALSE,TRUE]	FALSE	TRUE: The command is being executed.
Error	Error flag	BOOL	[FALSE,TRUE]	FALSE	TRUE: An error occurs on the function block.
ErrorID	Error ID	ERROR_CO	N/A	NO_ERROR	Indicates the error ID (see ERROR_CO for details).
Value	Value	LREAL	(-1.7E 308, 1.7E 308)	0.0	Indicates a read value.

3) Function

■ Read object dictionary data: Based on CANopen parameters, the specific axis parameter number consists of the object dictionary length 16 # UV (1 byte), object dictionary index 16 # ABCD (2 bytes) and subindex 16 # EF (1 byte), which form the -16 # UVABCDEF mode. Example: If the read object index value is 16 # 607C, the subindex value is 16#00, and the data length value is 16#04, the parameter number is -16 # 04607C00.

4) Sequence Diagram
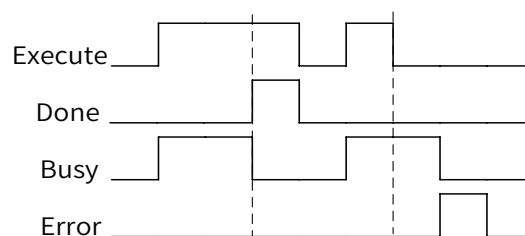
5) Error Description

When an error occurs, locate the error corresponding to **ErrorID** in **COUNTER_ERROR** through the help document, and find error causes.

11 MC_ReadStatus_CO

1) Instruction Format

Graphic Expression

```
                    MC_ReadStatus_CO
  —Axis AXIS_REF_SM3              BOOL Valid—
  —Enable BOOL                     BOOL Busy—
                                   BOOL Error—
                              Error_CO ErrorID—
                                BOOL ErrorStop—
                                 BOOL Disabled—
                                 BOOL Stopping—
                                  BOOL Homing—
                                BOOL Standstill—
                            BOOL DiscreteMotion—
                          BOOL ContinuousMotion—
```

Instruction	Name	ST expression

MC_ReadStatus_CO	Current axis motion state	MC_ReadStatus_CO(
		    Axis:= ,
		    Enable:= ,
		    Valid=> ,
		    Busy=> ,
		    Error=> ,
		    ErrorID=> ,
		    ErrorStop=> ,
		    Disabled=> ,
		    Stopping=> ,
		    Homing=> ,
		    Standstill=> ,
		    DiscreteMotion=> ,
		    ContinuousMotion=>);

2) Relevant Variables

■ I/O variable

I/O Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis	Axis	AXIS_REF_HC_CO	N/A	N/A	Indicates a CANopen axis.

■ Input Variable

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Enable	Execution input	BOOL	[FALSE,TRUE]	FALSE	(Triggered by level edges) TRUE: Indicates the Reading state.

■ Output variable

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Valid	Reading state validity flag	BOOL	[FALSE,TRUE]	FALSE	TRUE: Reset is completed.
Busy	Execution flag	BOOL	[FALSE,TRUE]	FALSE	TRUE: The command is being executed.
Error	Error flag	BOOL	[FALSE,TRUE]	FALSE	TRUE: An error occurs on the function block.
ErrorID	Error ID	ERROR_CO	N/A	NO_ERROR	Indicates the error ID (see ERROR_CO for details).

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
ErrorStop	Error stop	BOOL	[FALSE,TRUE]	FALSE	TRUE: The axis fails.
Disabled	Disabled	BOOL	[FALSE,TRUE]	FALSE	TRUE: The axis is disabled.
Stopping	Stopping	BOOL	[FALSE,TRUE]	FALSE	TRUE: The stop command is being executed.
Homing	Homing	BOOL	[FALSE,TRUE]	FALSE	TRUE: The axis is homing.
Standstill	Motor enabling	BOOL	[FALSE,TRUE]	FALSE	TRUE: The axis is enabled.
DiscreteMotion	Discrete motion	BOOL	[FALSE,TRUE]	FALSE	TRUE: The axis is moving in point-to-point mode.
ContinuousMotion	Continuous motion	BOOL	[FALSE,TRUE]	FALSE	TRUE: The speed command is being executed.

3) Function

■ Based on the PLCopen state machine standard, the motion state of controlled axis is displayed in the form of Boolean variable.

4) Sequence Diagram
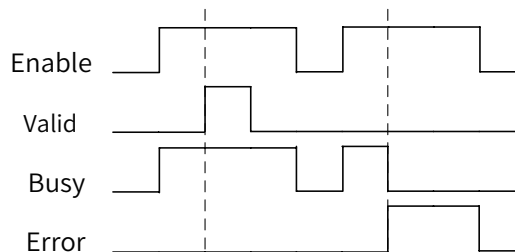
5) Error Description

When an error occurs, locate the error corresponding to **ErrorID** in **COUNTER_ERROR** through the help document, and find error causes.

12 MC_Jog_CO

1) Instruction Format

Instruction	Name	ST expression
MC_Jog_CO	Read remote device parameter	MC_Jog_CO(Axis:= , JogForward:= , JogBackward:= , Velocity:= , Acceleration:= , Deceleration:= , Busy=> , CommandAborted=> , Error=> , ErrorID=>);

2) Relevant Variables

■ I/O variable

I/O Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis	Axis	AXIS_REF_HC_CO	N/A	N/A	Indicates a CANopen axis.

■ Input Variable

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
JogForward	Jog in the forward direction	BOOL	[FALSE,TRUE]	FALSE	Indicates a jog in the forward direction.
JogBackword	Jog in the reverse direction	BOOL	[FALSE,TRUE]	FALSE	Indicates a jog in the reverse direction.
Velocity	Speed	LREAL	(0, 1.7E 308)	0.0	Indicates the jog speed.
Acceleration	Acceleration	LREAL	(0, 1.7E 308)	0.0	Indicates the acceleration.
Deceleration	Deceleration	LREAL	(0, 1.7E 308)	0.0	Indicates the deceleration.

■ Output variable

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Busy	Execution flag	BOOL	[FALSE,TRUE]	FALSE	TRUE: The command is being executed.
CommandAborted	Abort flag	BOOL	[FALSE,TRUE]	FALSE	TRUE: Motion is aborted.
Error	Error flag	BOOL	[FALSE,TRUE]	FALSE	TRUE: An error occurs on the function block.
ErrorID	Error ID	ERROR_CO	N/A	NO_ERROR	Indicates the error ID (see ERROR_CO for details).

3) Function

■ The function block is used for axis jog control in both forward and reverse directions.

■ If **JogBackword** and **JogForward** are set to **TRUE** simultaneously, motion stops. Velocity, Acceleration, and Deceleration values must be greater than 0.
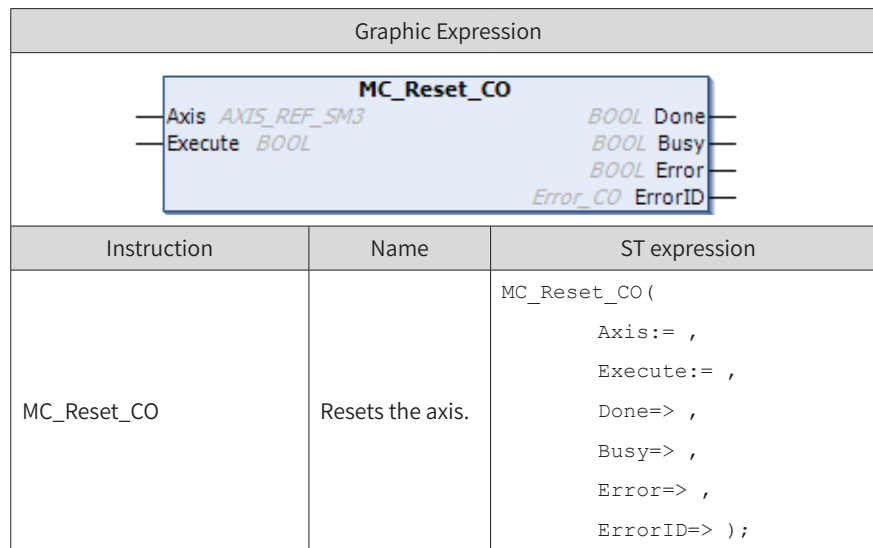
4) Sequence Diagram
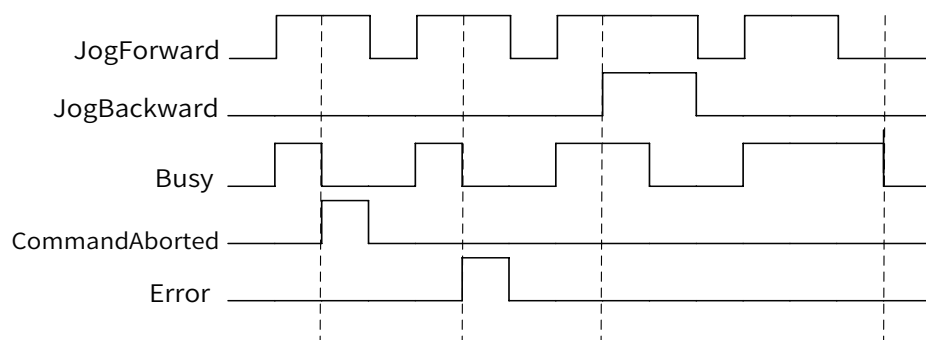
5) Error Description

When an error occurs, locate the error corresponding to **ErrorID** in **COUNTER_ERROR** through the help document, and find error causes.

6.3.3 CANopen 402 Parameter Setting

Configuration is an important part of Inovance Inoproshop programming software. Currently, Inoproshop supports IS620P-CO models only. To use drives of other manufacturers, import EDS files of other manufacturers first. In addition, confirm that drive programs of other manufacturers are designed in strict compliance with CANopen communication flags and CiA402 standard.

Before calling the CANopen402 function block, set parameters correctly.

1 Master Configuration

Major configuration items for the master include communication Baud rate, synchronization mode, synchronization cycle period, heartbeat, and producer time.

■ Communication Baud rate: The communication efficiency is improved as the Baud rate increases. However, the communication distance decreases as the Baud rate increases. Normally, the Baud rate is set to 500 Kbits/s.

■ Synchronization mode: **Enable Sync Producing** must be selected. Otherwise, the function block cannot work.

■ Synchronization cycle period: If only CANopen axis is available, it is recommended that you set the period to 4 ms, set the number of slaves to 3, and set the size of PDO configurations sent and received to less than eight bytes. It is recommended that the cycle period be equal to the CANopen task scan cycle period.

■ Heartbeat: The master sends heartbeat frames at intervals of producer time so that slaves can check whether the master is disconnected. The function must be used with slaves. Some slaves do not have the heartbeat check function, which is not required by default.

■ Producer time: The master sends heartbeat frames to slaves at intervals of preset producer time. The producer time is 300 ms by default and takes effect when **Enable Heartbeat** is selected.
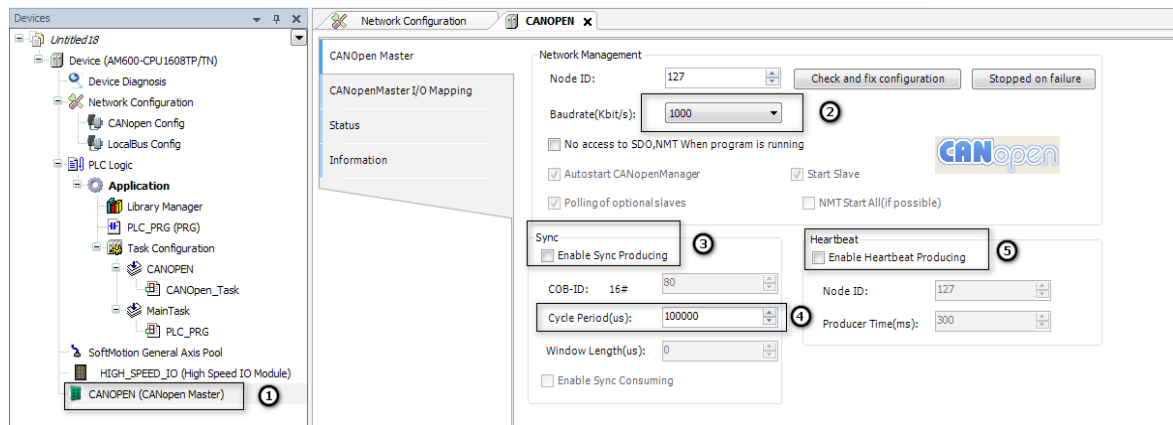
1 - CANopen master station; 2 - Baud rate; 3 - Synchronize the master station; 4 - Synchronization cycle period; 5 - Enable heartbeat as necessary.

Figure 6-29   CANopen master configuration

2 Slave Configuration

Major configuration items for a slave include node ID, heartbeat, producer time, PDO synchronization mode, and synchronization object dictionary. For details, see Figure 6-30, Figure 6-31, and Figure 6-32.

■ Node ID: Also called the station number, it is a basic parameter for slave communication. It should be consistent with the physical station number.

■ Heartbeat: The slave sends heartbeat frames to specified stations so that other stations can monitor

its communication status. **Enable Heartbeat** is selected by default.

■ Producer time: The slave sends heartbeat frames to specified stations at intervals of preset producer time. The producer time is 1000 ms by default and takes effect when **Enable Heartbeat** is selected.

■ PDO synchronization mode: The asynchronous mode is selected by default. You need to change it to the cyclic synchronization mode.

■ PDO dictionary configuration: The PDO dictionary ensures that slaves exchange data with the master every bus cycle. The more PDOs, the more efficient slaves exchange data with the master. However, the more PDOs, the more the bus load. Excessive PDOs may delay bus data transmission and even cause disconnection. Select **16#607A** and **16#6040** for PDO receiving. Select **16#6064** or **16#6063** and **16#6041** for PDO sending. In this way, the load demand can be met.
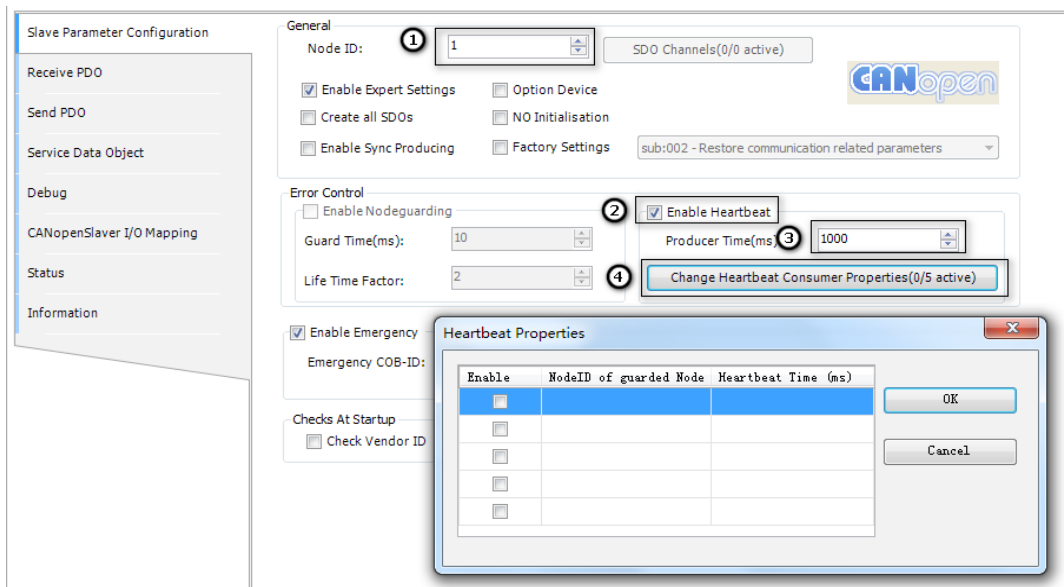
1 - Node ID; 2 - Enable heartbeat; 3 - Heartbeat cycle; 4 - Heartbeat consumer properties

Figure 6-30 CANopen slave parameter setting
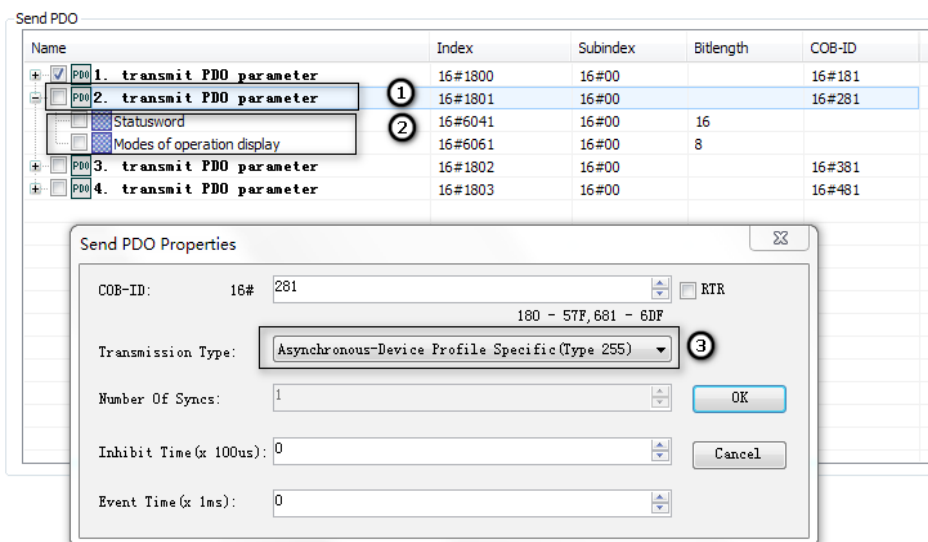
1 - Double click to set PDO sending from slave stations; 2 - Send the PDO dictionary; 3 - Set the synchronization cycle for sending PDOs from slave stations.

Figure 6-31 Configuration for CANopen slave sending PDOs
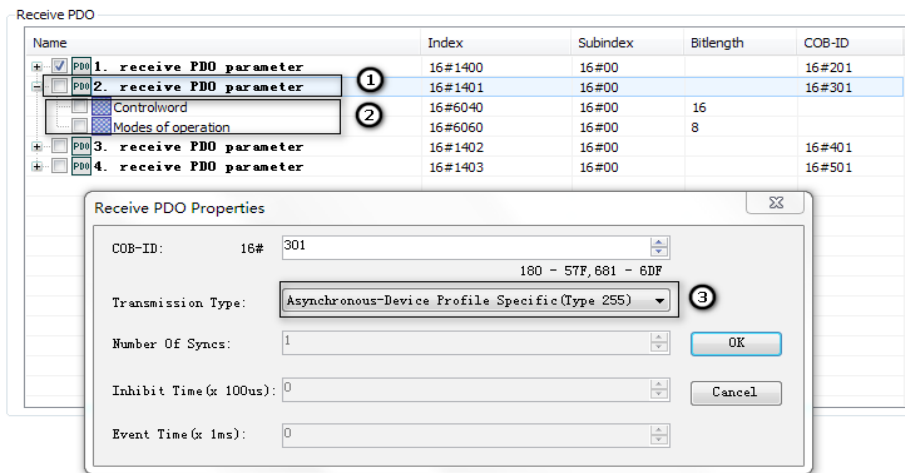
1 - Double click to set PDO receiving by slave stations; 2 - Receive the PDO dictionary; 3 - Set the synchronization cycle for PDO receiving by slave stations.

Figure 6-32 Configuration for CANopen slave receiving PDOs

3 Axis Configuration
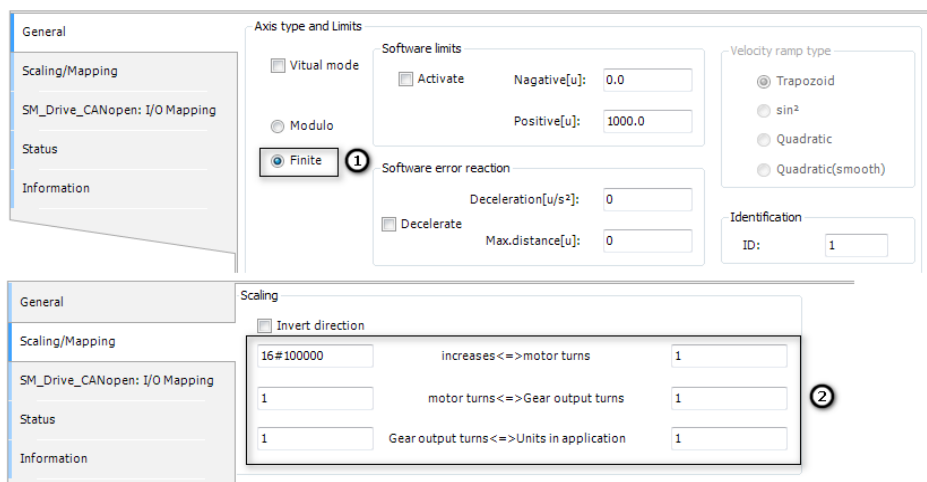
1 - Set it to the linear mode; 2 - Set the parameters based on the drive-encoder pulse zoom ratio. For details, see the EtherCAT axis.

Figure 6-33 CANopen axis configuration

6.3.4 CANopen 402 Error Diagnosis

Online Diagnosis

The system displays the last function block error or system error so that the error type can be diagnosed quickly.
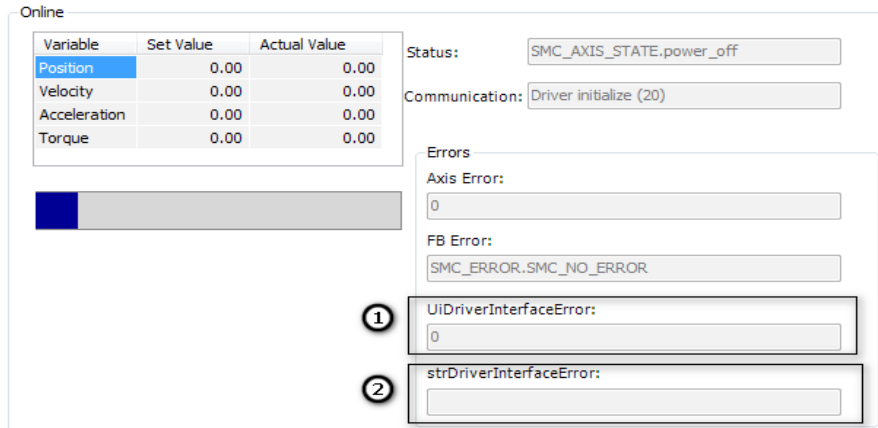
1 - Error ID; 2 - Error description

Figure 6-34 Online CANopen axis diagnosis

The following table lists error IDs and descriptions.

Error ID	Enumerated Value	Description
0	NO_ERROR	No error
1	DI_GENERAL_COMMUNICATION_ERROR	Axis communication error
2	DI_AXIS_ERROR	Drive axis error
21	WRONG_OPMODE	Incorrect operation mode
33	AXIS_IN_ERRORSTOP	Axis in ErrorStop state
34	AXIS_NOT_READY_FOR_MOTION	Axis not ready for motion
35	MA_MR_MODULO_ACT_POS_NOT_MAPPED	Reserved
36	MV_INVALID_VELACCDEC_VALUES	Invalid speed or acceleration/deceleration
80	RAG_ERROR_DURING_STARTUP	Reserved
81	RAG_ERROR_WRITING_COMSTATE	Reserved
82	RAG_ERROR_READING_COMSTATE	Reserved
90	CGR_ZERO_VALUES	Reserved
91	CGR_AXIS_POWERED	Reserved
93	CGR_MODULOPERIOD_NOT_INTEGRAL	Reserved
94	CGR_MOVEMENTTYPE_INVALID	Reserved
95	CGR_MODULOPERIOD_NON_POSITIVE	Reserved
96	CGR_MODULOPERIOD_TOO_SMALL	Reserved
97	CGR_MODULOPERIOD_TOO_LARGE	Reserved
120	R_NO_ERROR_TO_RESET	No error to reset
121	R_DRIVE_DOESNT_ANSWER	No answer
122	R_ERROR_NOT_RESETTABLE	Error not resettable
123	R_DRIVE_DOESNT_ANSWER_IN_TIME	Reserved
130	RP_PARAM_UNKNOWN	Read parameter error
131	RP_REQUESTING_ERROR	Communication request error
132	RP_RCV_PARAM_CONVERSION_ERROR	Reserved

Error ID	Enumerated Value	Description
133	RP_LOCAL_PARAM_NOT_DONE_IMMEDIATELY	Reserved
134	RP_CANNOT_SEND_MSG	Unable to send messages
140	WP_PARAM_INVALID	Write parameter error
141	WP_SENDING_ERROR	Sending error
142	WP_TMT_PARAM_CONVERSION_ERROR	Reserved
143	WP_LOCAL_PARAM_NOT_DONE_IMMEDIATELY	Reserved
144	WP_CANNOT_SEND_MSG	Unable to send messages
170	H_AXIS_WASNT_STANDSTILL	Axis in Standstill state for homing
183	MS_AXIS_IN_ERRORSTOP	ErrorStop state not allowed for the mc_stop function block
184	MS_AXIS_IN_STOPPING	Stopping state not allowed for the mc_stop function block
10,000	TIMEOUT_CHANGING_OPMODE	Mode switching timeout
10001	INTERNAL_UNKNOWN_CMD	Reserved
10002	CANNOT_START_MOVEMENT	Reserved
10003	CANNOT_START_HOMING	Reserved
10004	STOP_ALREADY_ACTIVE	Reserved
10005	POWER_ALREADY_ACTIVE	Reserved
10006	SMC_DI_VOLTAGE_DISABLED	Axis motion process interrupt enabled

Log Diagnosis

System program running errors can be logged for a period of time. You can locate the process of program error occurrence based on logs, in which error date, time, drive axis, function block, and error ID are recorded.

As shown in the following figure, the error occurred at 09:09:24 on 2017.9.20, the drive number is 4, the function block is Axis_CO, and the error message is "DI_GENERAL_COMMUNICATION_ERROR". According to the preceding error table, the axis error is a communication error.

Based on the log, you can determine that a communication error occurred and then the absolute positioning function block reported the error.
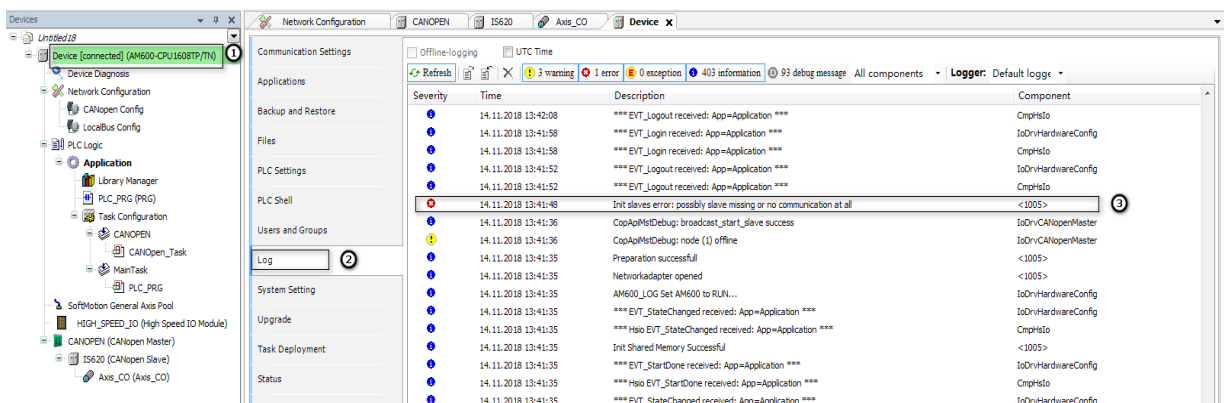
1 - Login; 2 - Log; 3 - Error date, time, and content

Figure 6-35 CANopen axis log diagnosis

6.3.5 Precautions

1 When used with Inovance IS620P (MCU version earlier than 11.0):

For an IS620P-CO object dictionary, none of the target speed (16#60FF) in profile speed mode, speed in profile position mode (16#6081), acceleration (16#6083) in profile position mode, and deceleration (16#6084) in profile position mode is in the unit of pulses. However, all input motion parameters of PLC function blocks are in the unit of pulses. Therefore, conversion is required.

The following figure shows the absolute positioning function block. The axis zoom ratio (facter) is 1:1048576. The target position assigned by the PLC to the servo is 500 units multiplied by facter (object dictionary 16#607A). The speed, acceleration, and deceleration are 1048.576 (0.001 × facter = 1048.576).
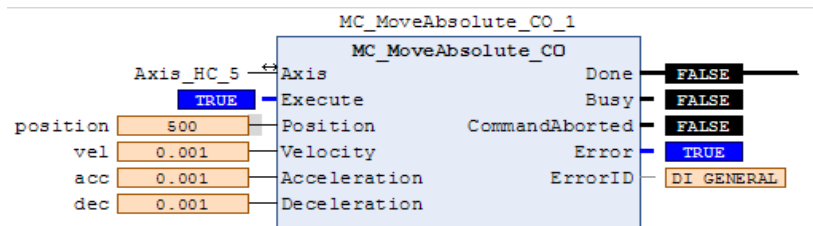
Figure 6-36 CANopen absolute positioning function block

Why are speed and acceleration values so small? Actually the preset values for the function block are large because the speed and acceleration for an IS620P-CO model are in the unit of rpm and rpm/ms, respectively. For the servo, the speed and acceleration/deceleration are set to 1,048 rpm (decimal places discarded) and 1,048 rpm/s (decimal places discarded), respectively. However, standard speed and acceleration/deceleration units are pulses/s and pulses/s^2, respectively. Therefore, convert units during use.

6081h	00h	Profile velocity	RW	YES	Uint32	RPM	0 to (2^{32} – 1)	100
6083h	00h	Profile acceleration	RW	YES	Uint32	RPM/ms	0 to (2^{32} – 1)	100
6084h	00h	Profile deceleration	RW	YES	Uint32	RPM/ms	0 to (2^{32} – 1)	100

Figure 6-37 IS620P-CO speed and acceleration/deceleration units

2 When used with Kinco FM8660:

If a Kinco drive is switched to another working mode during motion, the control mode is normal but the feedback mode is abnormal. As a result, the MC_Stop_CO function block cannot work.

Do not switch the mode if a Kinco drive is not enabled. Otherwise, the MC_Power_CO function block cannot work. It switches from the current control mode to the profile position control mode. When the device fails in profile speed mode, the device is disconnected. If the device is reset and re-enabled, the axis automatically runs again at the target speed previously set. To avoid this problem, we have processed the function block as described above. Therefore, the function block cannot be used with a Kinco drive.
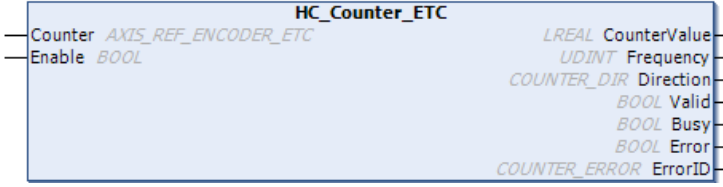
6.4 EtherCAT Remote Counting

Function overview: The EtherCAT slave module with a high-speed input port has counting, sampling, comparison output, touch probe, and other functions. The maximum frequency of sampling and output is 200 kHz. The maximum frequency of high-speed interrupts is 3 kHz.

Library name: IoDrvEtherCATEncoder (applicable to V1.0.5.0 and later versions)

6.4.1 HC_Counter_ETC

1) Instruction Format

Graphic Expression

Instruction	Name	ST expression

HC_Counter_ETC	Counter enabling	HC_Counter_ETC(
		     Counter:= ,
		     Enable:= ,
		     CounterValue=> ,
		     Frequency=> ,
		     Direction=> ,
		     Valid=> ,
		     Busy=> ,
		     Error=> ,
		 ErrorID=>);

2) Relevant Variables

■ I/O variable

I/O Variable	Name	Data Type	Valid Range	Initial Value	Description
Counter	Remote counter	AXIS_REF_ ENCODER_ETC	N/A	N/A	AXIS_REF_ENCODER_ETC Counter variable

■ Input variable

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Enable	Enabled	BOOL	[FALSE,TRUE]	FALSE	(Triggered by level) TRUE: Enables the counter.

■ Output variable

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
CounterValue	Count	LREAL	(-1.7E 308, 1.7E 308)	0.0	Indicates the current count, converted from count pulses.
Frequency	Frequency	UDINT	[0,2^32)	0	Indicates the frequency of collected pulses.
Direction	Indicates the direction.	COUNTER_DIR	[-1,1]	-1	Indicates the counting direction.
Valid	Enabled state	BOOL	[FALSE,TRUE]	FALSE	Indicates that the counter is enabled.
Busy	Execution flag	BOOL	[FALSE,TRUE]	FALSE	Executes the counter enabling command.
Error	Error flag	BOOL	[FALSE,TRUE]	FALSE	**TRUE** indicates that an error occurs inside the function block.
ErrorID	Error ID	COUNTER_ ERROR	N/A	COUNTER_ NO_ERROR	Indicates the error ID (see COUNTER_ERROR for details).

3) Function

■ The function block is used for remote counter enabling, counting, frequency measurement, and counting direction output.

4) Sequence Diagram
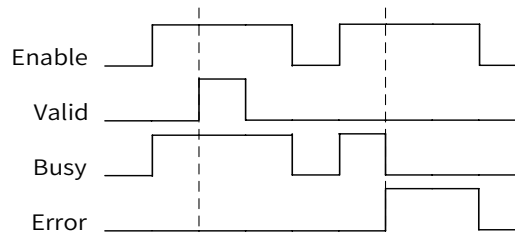
5) Error Description

When an error occurs, locate the error corresponding to **ErrorID** in **COUNTER_ERROR** through the help document, and find error causes.

Error ID	Enumerated Value	Description
0x00	COUNTER_NO_ERROR	No error occurs.
0x01	COUNTER_ERROR_ECT_COMMUNICATION	ETC communication fails.
0x02	COUNTER_ERROR_PV_PRESETVALUE_EXCEED	The preset value is out of range.
0x03	COUNTER_ERROR_CP_COMPARE_VAULE_INVALID	The comparison value is improperly set.
0x04	COUNTER_ERROR_FILT_PARAM_EXCEED	The filter coefficient is out of range.
0x10	COUNTER_ERROR_PV_DI_CONFIGURE_ERROR	The presetting function is not configured for the DI terminal.
0x11		The counter is zeroed, not in use.
0x12	COUNTER_ERROR_TP_DI_CONFIGURE_ERROR	The touch probe is not configured for the DI terminal.
0x30	COUNTER_ERROR_CP_DO_CONFIGURE_ERROR	The coincident output function is not configured for the DO output terminal.
0x50	COUNTER_ERROR_INPUT_PUSLE_FREQUENCY_TOO_HIGH	The input frequency is greater than 202 kHz.
0x51	COUNTER_ERROR_OVERFLOW	The count overflows.
0x52	COUNTER_ERROR_UNDERFLOW	The count underflows.
0x1001	COUNTER_ERROR_PV_TYPE_INVALID	The preset type is out of range.
0x1002	COUNTER_ERROR_DISABLE	The counter is disabled.
0x1003	COUNTER_ERROR_TP_PARAM_INVALID	Touch probe parameters are invalid.
0x1004	COUNTER_ERROR_CP_PARAM_INVALID	Comparison channel parameters are invalid.
0x1005	COUNTER_ERROR_R_NO_ERROR_CLEAR	No errors are cleared.
0x1006	COUNTER_ERROR_R_ERROR_CANNOT_RESET	Errors cannot be cleared.
0x1100	COUNTER_ERROR_PARAM_NO_PDO_MAPPING	No PDOs are configured for the instruction.

6.4.2 HC_SetCompare_ETC

1)   Instruction Format

Graphic Expression
HC_SetCompare_ETC — Counter *AXIS_REF_ENCODER_ETC*　　　*BOOL* Done — — Execute *BOOL*　　　　　　　　　　　*BOOL* Busy — — Abort *BOOL*　　　　　*BOOL* CommandAborted — — Channel *BYTE*　　　　　　　　　　*BOOL* Error — — CompareValue *LREAL*　　*COUNTER_ERROR* ErrorID — — ImRefreshCycle *UINT*

Instruction	Name	ST expression

HC_SetCompare_ETC	Coincident output function for a remote counter	`HC_SetCompare_ETC(`
		　　　　`Counter:= ,`
		　　　　`Execute:= ,`
		　　　　`Abort:= ,`
		　　　　`Channel:= ,`
		　　　　`CompareValue:= ,`
		　　　　`ImRefreshCycle:= ,`
		　　　　`Done=> ,`
		　　　　`Busy=> ,`
		　　　　`CommandAborted=> ,`
		　　　　`Error=> ,`
		`ErrorID=> );`

2)   Relevant Variables

■   I/O variable

I/O Variable	Name	Data Type	Valid Range	Initial Value	Description
Counter	Counter	AXIS_REF_ENCODER_ETC	N/A	N/A	AXIS_REF_ENCODER_ETC Counter variable

■   Input variable

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Execute	Enabled	BOOL	[FALSE,TRUE]	FALSE	(Triggered by edges) Indicates the switch used to execute coincident functions.
Abort	Abort	BOOL	[FALSE,TRUE]	FALSE	(Triggered by level) TRUE: Aborts the function block execution.
Channel	Comparison channel	BYTE	[1,2]	1	Selects a comparison channel for the counter (two comparison channels available).
CompareValue	Comparison value	LREAL	(-1.7E 308, 1.7E 308)	0.0	Indicates a comparison value, which is a floating point (unit: unit).
ImRefreshCycle	Output hold time	UINT	(0,2^32)	1000	Indicates the hold time for output port enabling, in the minimum unit of 100 μs. The default value is 1000 (1000 × 100 μs = 100 ms).

■ Output variable

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Done	Coincident function completed	BOOL	[FALSE,TRUE]	FALSE	The count pulse reaches the preset comparison value, and the output hold time runs out.
CommandAborted	Aborted state	BOOL	[FALSE,TRUE]	FALSE	Indicates the Aborted state (Abort = TURE).
Busy	Execution flag	BOOL	[FALSE,TRUE]	FALSE	Executes coincident functions.
Error	Error flag	BOOL	[FALSE,TRUE]	FALSE	**TRUE** indicates that an error occurs inside the function block.
ErrorID	Error ID	COUNTER_ERROR	N/A	COUNTER_NO_ERROR	Indicates the error ID (see COUNTER_ERROR for details).

3) Function

■ When the current count of the remote counter is equal to the preset comparison value, the function block triggers a hardware high-speed output port and holds it for a period of time.

■ The coincident output function is executed by a single trigger. When **Done** is **TRUE**, the coincident function is completed. To continue using the comparison function, trigger the execution again.

■ Confirm attributes of counter coincident output points configured in development software.

4) Sequence Diagram
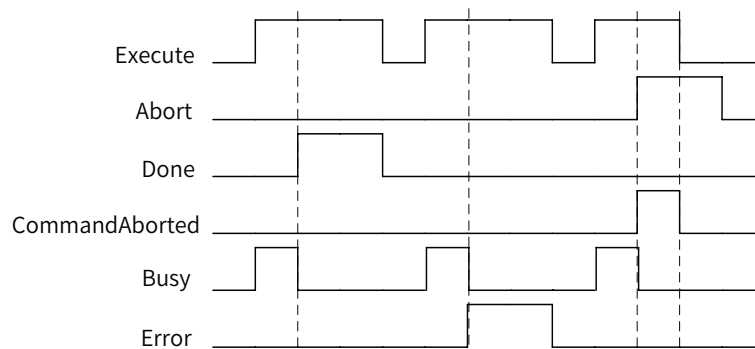
5) Error Description

When an error occurs, locate the error corresponding to **ErrorID** in **COUNTER_ERROR** through the help document, and find error causes.

6.4.3 HC_Presetvalue_ETC

1) Instruction Format

Graphic Expression		
HC_PresetValue_ETC Counter *AXIS_REF_ENCODER_ETC* — *BOOL* Done / Execute *BOOL* — *BOOL* Busy / Abort *BOOL* — *BOOL* CommandAborted / TriggerType *PRESET_TRIGGER* — *BOOL* Error / PresetValue *LREAL* — *COUNTER_ERROR* ErrorID		
Instruction	Name	ST expression

HC_PresetValue_ETC	Counter presetting	HC_PresetValue_ETC(
		     Counter:= ,
		     Execute:= ,
		     Abort:= ,
		     TriggerType:= ,
		     PresetValue:= ,
		     Done=> ,
		     Busy=> ,
		     CommandAborted=> ,
		     Error=> ,
		 ErrorID=>);

2) Relevant Variables

■ I/O variable

I/O Variable	Name	Data Type	Valid Range	Initial Value	Description
Counter	Remote counter	AXIS_REF_ ENCODER_ ETC	N/A	N/A	AXIS_REF_ENCODER_ETC Counter variable

■ Input variable

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Execute	Enabled	BOOL	[FALSE,TRUE]	FALSE	(Triggered by edges) Indicates the switch used to execute the presetting function.
Abort	Abort	BOOL	[FALSE,TRUE]	FALSE	(Triggered by level) TRUE: Aborts the function block execution.
TriggerType	Trigger type	BYTE	[1,7]	1	Internal trigger: 0x01 DI trigger: 0x02 Combination of internal and DI triggers: 0x03 Z-phase trigger: 0x04 Combination of internal and Z-phase triggers: 0x05 Combination of DI and Z-phase triggers: 0x06 Combination of internal, DI, and Z-phase triggers: 0x07
PresetValue	Preset value	LREAL	(-1.7E 308, 1.7E 308)	0.0	Indicates the preset value of the counter.

■ Output variable

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Done	Preset successful state	BOOL	[FALSE,TRUE]	FALSE	The count pulse reaches the preset comparison value, and the output hold time runs out.
CommandAborted	Aborted state	BOOL	[FALSE,TRUE]	FALSE	Indicates the Aborted state (Abort = TURE).
Busy	Execution flag	BOOL	[FALSE,TRUE]	FALSE	Executes the counter presetting command.
Error	Error flag	BOOL	[FALSE,TRUE]	FALSE	**TRUE** indicates that an error occurs inside the function block.
ErrorID	Error ID	COUNTER_ERROR	N/A	COUNTER_ NO_ERROR	Indicates the error ID (see COUNTER_ERROR for details).

3) Function

■ The function block is used to preset the remote counter. Seven presetting modes are available. When a combination of more than two triggers is selected, the counter is preset provided that any of conditions in the combination is met. The **Done** signal is output after the counter is preset. The presetting function block is executed by a single rising edge trigger.

4) Sequence Diagram
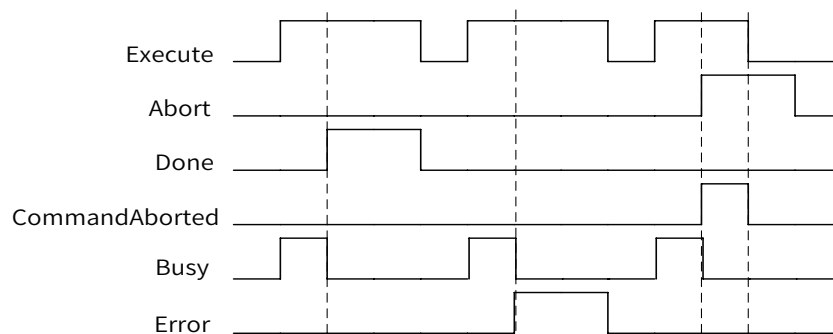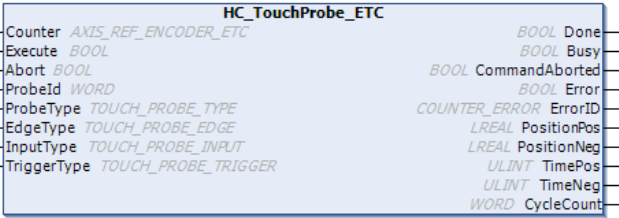
5) Error Description

When an error occurs, locate the error corresponding to **ErrorID** in **COUNTER_ERROR** through the help document, and find error causes.

6.4.4 HC_TouchProbe_ETC

1) Instruction Format

Graphic Expression

Instruction	Name	ST expression
HC_TouchProbe_ETC	Counter touch probe	HC_TouchProbe_ETC(　　　　Counter:= , 　　　　Execute:= , 　　　　Abort:= , 　　　　ProbeId:= , 　　　　ProbeType:= , 　　　　EdgeType:= , 　　　　InputType:= , 　　　　TriggerType:= , 　　　　Done=> , 　　　　Busy=> , 　　　　CommandAborted=> , 　　　　Error=> , 　　　　ErrorID=> , 　　　　PositionPos=> , 　　　　PositionNeg=> , 　　　　TimePos=> , 　　　　TimeNeg=> , CycleCount=>);

2) Relevant Variables

■ I/O variable

I/O Variable	Name	Data Type	Valid Range	Initial Value	Description
Counter	Remote counter	AXIS_REF_ENCODER_ETC	N/A	N/A	AXIS_REF_ENCODER_ETC Counter variable

■ Input variable

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Execute	Enabled	BOOL	[FALSE,TRUE]	FALSE	(Triggered by edges) Indicates the switch used to execute the presetting function.
Abort	Abort	BOOL	[FALSE,TRUE]	FALSE	(Triggered by level) TRUE: Aborts the function block execution.
ProbeId	Touch probe ID	WORD	[1,2]	1	Indicates the touch probe ID (only two touch probes available).

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
ProbeType	Touch probe type	TOUCH_PROBE_TYPE	N/A	PositionLatch	Indicates the touch probe type: time or position.
EdgeType	Edge type	TOUCH_PROBE_EDGE	N/A	RisingEdge	Indicates the edge type: rising edge or falling edge.
InputType	Hardware trigger type	TOUCH_PROBE_INPUT	N/A	DigtalInput	Indicates an external trigger: Z-phase trigger or DI trigger.
TriggerType	Trigger mode	TOUCH_PROBE_TRIGGER	N/A	TrigSingle	Indicates the trigger type: continuous trigger or single trigger.

■ Output variable

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Done	Touch probe execution status	BOOL	[FALSE,TRUE]	FALSE	Indicates that the touch probe function execution ends.
CommandAborted	Aborted state	BOOL	[FALSE,TRUE]	FALSE	Indicates the Aborted state (Abort = TURE).
Busy	Execution flag	BOOL	[FALSE,TRUE]	FALSE	Execute the counter touch probe function.
Error	Error flag	BOOL	[FALSE,TRUE]	FALSE	**TRUE** indicates that an error occurs inside the function block.
ErrorID	Error ID	COUNTER_ERROR	N/A	COUNTER_NO_ERROR	Indicates the error ID (see COUNTER_ERROR for details).
PositionPos	Rising edge latch position	LREAL	(-1.7E 308, 1.7E 308)	0.0	Indicates the rising edge latch position (unit: unit).
PositionNeg	Falling edge latch position	LREAL	(-1.7E 308, 1.7E 308)	0.0	Indicates the falling edge latch position (unit: unit).
TimePos	Rising edge latch time	LINT	[0,2^63)	0	Indicates the rising edge latch time (unit: ns).
TimeNeg	Falling edge latch time	LINT	[0,2^63)	0	Indicates the falling edge latch time (unit: ns).
CycleCount	Touch probe check count	WORD	[0,2]	0	Indicates a valid touch probe count in continuous trigger mode. The **CycleCount** value is increased when the touch probe is triggered. It varies between 0 to 2 cyclically.

3) Function

■ The function block is used for touch probes of the remote counter. Two touch probes are supported, both of which can be triggered in single or continuous mode.

■ The position touch probe and time touch probe can be used simultaneously. When using the touch probe function, check the configuration of counter touch probe attributes and PDO mapping. If they are improperly configured, when the function block is triggered (**Execute**), **Error** is set to **TRUE**. You can query the error table (COUNTER_ERROR) through the error ID (ErrorID).

4) Sequence Diagram
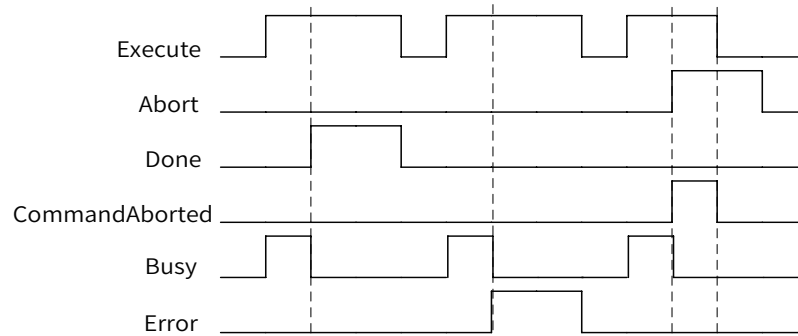
5) Error Description

When an error occurs, locate the error corresponding to **ErrorID** in **COUNTER_ERROR** through the help document, and find error causes.

6.4.5 HC_Reset_ETC

1) Instruction Format

Graphic Expression

Instruction	Name	ST expression

HC_Reset_ETC	Counter resetting	`HC_Reset_ETC(`
		`        Counter:= ,`
		`        Execute:= ,`
		`        Done=> ,`
		`        Busy=> ,`
		`        Error=> ,`
		`ErrorID=> );;`

2) Relevant Variables

■ I/O variable

I/O Variable	Name	Data Type	Valid Range	Initial Value	Description
Counter	Remote counter	AXIS_REF_ENCODER_ETC	N/A	N/A	AXIS_REF_ENCODER_ETC Counter variable

■ Input Variable

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Execute	Enabled	BOOL	[FALSE,TRUE]	FALSE	(Triggered by edges) Indicates the counter reset switch.

■ Output variable

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Done	Reset complete state	BOOL	[FALSE,TRUE]	FALSE	Indicates that the reset function execution ends.
Busy	Execution flag	BOOL	[FALSE,TRUE]	FALSE	**TRUE** indicates that the reset command is being executed.
Error	Error flag	BOOL	[FALSE,TRUE]	FALSE	**TRUE** indicates that an error occurs inside the function block.
ErrorID	Error ID	COUNTER_ERROR	N/A	COUNTER_NO_ERROR	Indicates the error ID (see COUNTER_ERROR for details).

3) Function

■ The function block is used to reset the remote counter if faults or errors occur.

■ If the counter fails, execute the HC_Reset_ETC instruction to clear the fault. If the faults cannot be cleared, power off and then restart the counting module.

■ If an error occurs on the function block, set **Execute** from **TRUE** to **FALSE** to clear the error flag.

4) Sequence Diagram
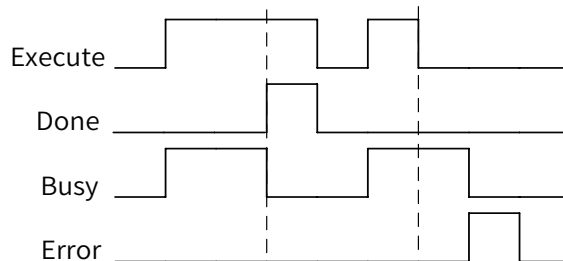
5) Error Description

When an error occurs, locate the error corresponding to **ErrorID** in **COUNTER_ERROR** through the help document, and find error causes.

6.5 Process Library

Function overview: Based on PLCopen function blocks, the function block integrates special processes, including Modbus floating point reconstitution, anisotropic bilateral polishing, anchor electronic cam, intelligent temperature control, auto-tuning PID, switching, and flying shear.

For more information, contact our technicians or visit our website (http://www.inovance.com/).
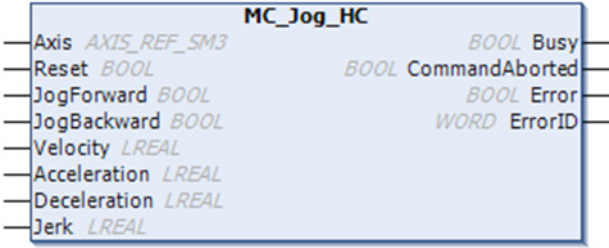
6.6 Others

To optimize our products, we provide practical function blocks, for example, bus reset and position saving function blocks.

Note: In the library manager, you can add CmpHCBasic.libaray to the project to use all function blocks described in the section.

6.6.1 MC_Jog_HC

1) Instruction Format

Graphic Expression		
MC_Jog_HC Axis *AXIS_REF_SM3* BOOL **Busy** Reset *BOOL* BOOL **CommandAborted** JogForward *BOOL* BOOL **Error** JogBackward *BOOL* WORD **ErrorID** Velocity *LREAL* Acceleration *LREAL* Deceleration *LREAL* Jerk *LREAL*		
Instruction	**Name**	**ST expression**

MC_Jog_HC	Jog function block	MC_Jog_HC(
		    Axis:= ,
		    Reset:= ,
		    JogForward:= ,
		    JogBackward:= ,
		    Velocity:= ,
		    Acceleration:= ,
		    Deceleration:= ,
		    Jerk:= ,
		    Busy=> ,
		    CommandAborted=> ,
		    Error=> ,
		ErrorID=>);

2) Relevant Variables

■ I/O variable

I/O Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis	Axis	AXIS_REF_SM3	N/A	N/A	AXIS_REF_SM3 variable

■ Input variable

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Reset	Reset	BOOL	[FALSE,TRUE]	FALSE	(Triggered by edges) It is set to TRUE when the motion function cannot be used. The function block can be used.
JogForward	JOG+	BOOL	[FALSE,TRUE]	FALSE	(Triggered by level) Indicates a jog in the forward direction.
JogBackward	JOG-	BOOL	[FALSE,TRUE]	FALSE	(Triggered by level) Indicates a jog in the reverse direction.
Velocity	Speed	LREAL	(0, 1.7E 308)	0.0	Indicates the speed (unit: unit/s).
Acceleration	Acceleration	LREAL	(0, 1.7E 308)	0.0	Indicates the acceleration (unit: unit/s^2).
Deceleration	Deceleration	LREAL	(0, 1.7E 308)	0.0	Indicates the deceleration (unit: unit/s^2).
Jerk	Jerk	LREAL	(0, 1.7E 308)	0.0	Indicates the jerk (unit: unit/s^3).

■ Output variable

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Busy	Execution flag	BOOL	[FALSE,TRUE]	FALSE	When **JogForward** or **JogBackward** is set to **TRUE**, **Busy** is set to **TRUE**. Otherwise, the axis slowly stops by deceleration. When the axis state machine is in Standstill state, **Busy** is set to **FALSE**.
CommandAborted	Abort flag	BOOL	[FALSE,TRUE]	FALSE	**TRUE** indicates that the function block is aborted by another function block.
Error	Error flag	BOOL	[FALSE,TRUE]	FALSE	**TRUE** indicates that an error occurs inside the function block.
ErrorID	Error ID	SMC_ERROR	N/A	NO_ERROR	Indicates the error ID (see SMC_ERROR for details).

3) Function

■ Similar to MC_Jog, the function block is used to control axis jog in the forward or reverse direction.

■ When hardware limit is enabled in the process of stop by deceleration and the axis cannot jog, trigger **Reset** so that the axis can jog.

4) Sequence Diagram
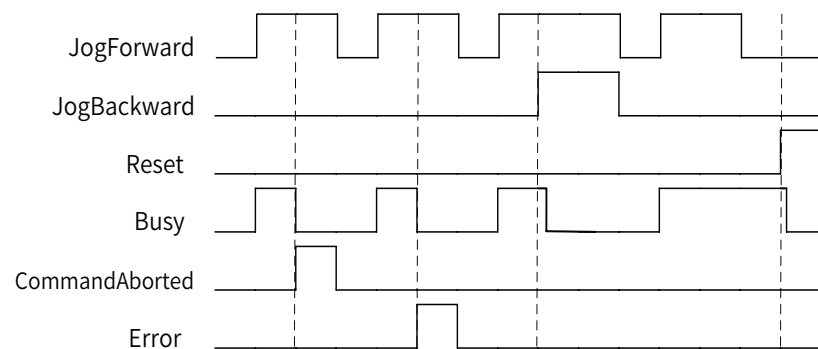
5) Error Description

When an error occurs, locate the error corresponding to **ErrorID** in **SMC_ERROR** through the help document, and find error causes.

6.6.2 MC_ResetDrive

1) Instruction Format

Graphic Expression		

Instruction	Name	ST expression

MC_ResetDrive	Reset of a single servo slave	MC_ResetDrive(
		    Axis:= ,
		    ETC_Master:= ,
		    ETC_Slave:= ,
		    Execute:= ,
		    TimeOut:= ,
		    Mode:= ,
		    Done=> ,
		    Busy=> ,
		    Error=> ,
		ErrorID=>);

2) Relevant Variables

■ I/O variable

I/O Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis	Axis	AXIS_REF_SM3	N/A	N/A	AXIS_REF_SM3 variable
ECT_Master	EtherCAT master	IoDrvEtherCAT	N/A	N/A	IoDrvEtherCAT variable, the EtherCAT master name
ETC_Slave	EtherCAT slave	ETCSlave	N/A	N/A	ETCSlave variable, the EtherCAT slave name

■ Input variable

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Execute	Reset	BOOL	[FALSE,TRUE]	FALSE	Triggered by edges, it is invalid at rising edges.
TimeOut	Timeout period	WORD	[0,2^32-1]	1000	Indicates the timeout period for communication state machine switching (1000 scan cycles by default). If state machine switching timed out, the state machine automatically returns to the Init state, and the process (Init > Pre-op > SafeOP > OP) starts again.
Mode	Reset mode	WORD	[0,1]	1	**0** indicates quick reset. **1** indicates normal reset (as the reset time is long, prevent the error Er.15 during reset).

■ Output variable

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Done	Complete flag	BOOL	[FALSE,TRUE]	FALSE	Indicates the reset complete flag (TRUE: Slave and axis reset).
Busy	Execution flag	BOOL	[FALSE,TRUE]	FALSE	When the execution is triggered, **Busy** is set to **TRUE**. When the command is set to **FALSE**, the variable remains in **TRUE** state, and the function block execution fails or succeeds.
Error	Error flag	BOOL	[FALSE,TRUE]	FALSE	**TRUE** indicates that an error occurs inside the function block.
ErrorID	Error ID	SMC_ERROR	N/A	NO_ERROR	Indicates the error ID (see SMC_ERROR for details).

3) Function

■ When a single servo is disconnected from slaves (the first slave excluded), it can be connected to networks and used.

■ **TimeOut** indicates the number of scan cycles. The value is the timeout period multiplied by the length of each scan cycle (ms).

■ **1** indicates normal reset. During reset, switch the axis to the security mode and wait 20 seconds to avoid clock jitter caused by servo adjustment. **0** indicates quick reset. During reset, switch the axis to security mode and directly reset the 402 state machine.

■ If the first slave is disconnected, the function block is inapplicable. You need to execute the global function block ETHERCAT and trigger ETHERCAT.xReStart at a rising edge to restart the entire network.

Example: Reset the slave IS620N_1 and master ETHERCAT where Axis_1 resides.
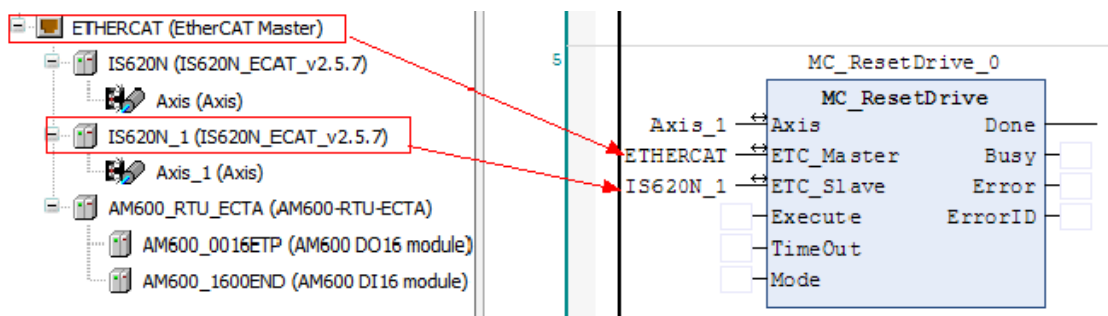
4) Sequence Diagram
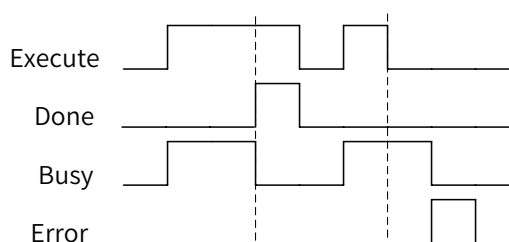
5)   Error Description

When an error occurs, locate the error corresponding to **ErrorID** in **SMC_ERROR** through the help document, and find error causes.

6.6.3 MC_ResetRemoteModule

1)   Instruction Format

Graphic Expression

```
            MC_ResetRemoteModule
— ETC_Slave  ETCSlave              BOOL  Done —
— Execute    BOOL                  BOOL  Busy —
— TimeOut    WORD                  BOOL  Error —
                           SMC_ERROR  ErrorID —
```

Instruction	Name	ST expression

MC_ResetRemoteModule	Reset of a single ECT slave	MC_ResetRemoteModule(
		        ETC_Slave:= ,
		        Execute:= ,
		        TimeOut:= ,
		        Done=> ,
		        Busy=> ,
		        Error=> ,
		ErrorID=>);

2)   Relevant Variables

■   I/O variable

I/O Variable	Name	Data Type	Valid Range	Initial Value	Description
ETC_Slave	EtherCAT slave	ETCSlave	N/A	N/A	ETCSlave variable, the EtherCAT slave name

■   Input variable

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Execute	Reset	BOOL	[FALSE,TRUE]	FALSE	Triggered by edges, it is invalid at rising edges.
TimeOut	Timeout period	WORD	[0,2^32-1]	1000	Indicates the timeout period for communication state machine switching (1000 scan cycles by default). If state machine switching timed out, the state machine automatically returns to the Init state, and the process (Init > Pre-op > SafeOP > OP) starts again.

■ Output variable

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Done	Complete flag	BOOL	[FALSE,TRUE]	FALSE	Indicates the reset complete flag (TRUE: Slave reset).
Busy	Execution flag	BOOL	[FALSE,TRUE]	FALSE	When the execution is triggered, **Busy** is set to **TRUE**. When the execution command is set to **FALSE**, **Busy** remains in **TRUE** state until the function block execution fails or succeeds.
Error	Error flag	BOOL	[FALSE,TRUE]	FALSE	**TRUE** indicates that an error occurs inside the function block.
ErrorID	Error ID	SMC_ERROR	N/A	NO_ERROR	Indicates the error ID (see SMC_ERROR for details).

3) Function

■ When a single ECT is disconnected from slaves (the first slave excluded), it can be connected to networks and used.

■ **TimeOut** indicates the number of scan cycles. The value is the timeout period multiplied by the length of each scan cycle (ms).

■ If the first slave is disconnected, the function block is inapplicable. You need to execute the global function block ETHERCAT and trigger ETHERCAT.xReStart at a rising edge to restart the entire network.

Example: Reset the ECT slave AM600_RTU_ECTA.
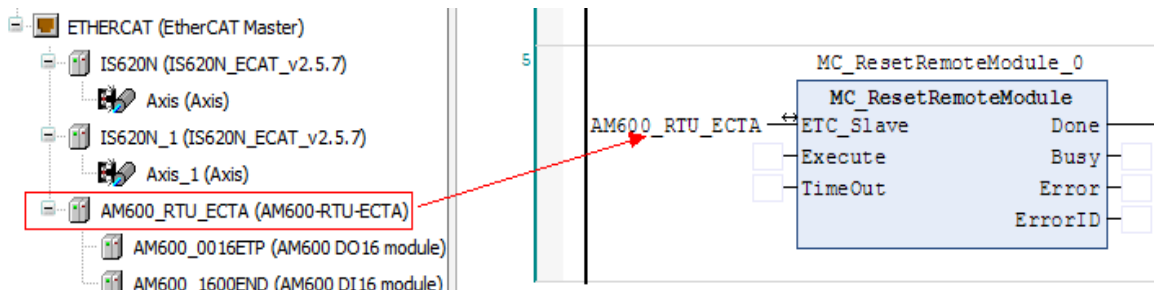
4) Sequence Diagram
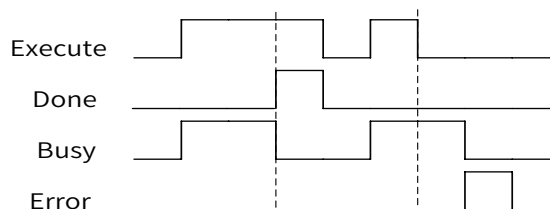
5) Error Description

When an error occurs, locate the error corresponding to **ErrorID** in **SMC_ERROR** through the help document, and find error causes.

6.6.4 MC_PersistPosition

1) Instruction Format

Graphic Expression

MC_PersistPosition

Axis *AXIS_REF_SM3* — — *BOOL* bPositionRestored
PersistPositionSingleturn_Data *SMC3_PersistPositionSingleturn_Data* — — *BOOL* bPositionStored
bEnable *BOOL* — — *BOOL* bBoundary
Threshold *UDINT* — — *BOOL* bBusy
— *BOOL* bError
— *MC_ERROR* eErrorID

Instruction	Name	ST expression

MC_PersistPosition	Retentive at power failure function block	MC_PersistPosition(
		    Axis:= ,
		PersistPositionSingleturn_Data:= ,
		    bEnable:= ,
		    Threshold:= ,
		    bPositionRestored=> ,
		    bPositionStored=> ,
		    bBoundary=> ,
		    bBusy=> ,
		    bError=> ,
		eErrorID=>);

2) Relevant Variables

■ I/O variable

I/O Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis	Axis	AXIS_REF_SM3	N/A	N/A	AXIS_REF_SM3 variable
PersistPositionSingleturn_Data	Structure variable	SMC3_PersistPositionSingleturn_Data	N/A	N/A	Structure variable used to store axis position data.

■ Input variable

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
bEnable	Enabled	BOOL	[FALSE,TRUE]	FALSE	Triggered by level, it is valid at high level.
Threshold	Threshold for encoder pulses during power-on/power-off	UDINT	[0,2^32-1]	16#800000	The axis moves after power-off. When the distance is beyond the threshold, the function block reports a recovery error.

■ Output variable

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
bPositionRestored	Recovery complete flag	BOOL	[FALSE,TRUE]	FALSE	**TRUE** indicates that recovery succeeds.
bPositionStored	Saving flag	BOOL	[FALSE,TRUE]	FALSE	**TRUE** indicates that data is being saved.
bBoundary	Boundary flag	BOOL	[FALSE,TRUE]	FALSE	**TRUE** indicates that the axis is at the boundary.

| Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Busy	Execution flag	BOOL	[FALSE,TRUE]	FALSE	When **bEnable** is triggered, **Busy** is set to **TRUE**.
Error	Error flag	BOOL	[FALSE,TRUE]	FALSE	**TRUE** indicates that an error occurs inside the function block.
eErrorID	Error ID	MC_ERROR	N/A	NO_ERROR	Indicates the error ID (see MC_ERROR for details).

3) Function

■ The function block is used to save absolute upper and lower positions of a single absolute value servo axis.

■ SMC3_PersistPositionSingleturn_Data indicates the data structure.

■ The threshold is important. When the PLC is powered off, the object dictionary 16#6064 (32-bit position sent by the servo to the PLC) is saved in the flash memory. When **bEnable** is set to **TRUE**, the system reads the actual position from the flash memory during startup. When the system bus can be used for PDO communication, the system reads the current position of the servo axis. If the absolute value of (read position - current position) is above the threshold, the system determines that recovery failed, and **bError** is set to **TRUE** (even if the recovered data is correct). To prevent the system from moving after power-off, you can increase the threshold, but keep it no greater than 2^31-1. Otherwise, you cannot determine whether the motor passes the boundary after power-off.

■ **bPositionRestored** must be set to **TRUE** with the flag of successful recovery of servo axis position data, which can be detected through rising edges. Before the absolute servo moves, you need to confirm that **bPositionRestored** is set to **TRUE**. Otherwise, a collision may occur.

■ The function block detects in each cycle whether an axis stops at the boundary. If **bBoundary** is set to **TRUE**, the axis is near the boundary (16#7F000000 < Axis .dwActPosition < 16#81000000). In this case, position data recovery may fail if you restart the PLC. Therefore, it is not recommended that you power off the PLC.

■ If the servo is used with an Inovance IS62N model, adjust two servo parameters: select the absolute linear mode and block the absolute position overflow alarm.

■ Set the **MC_PersistPosition** to the value at the beginning of program scan and set the default initial value of **bEnable** to **TRUE**. In this way, the system can scan the function block after being powered on.
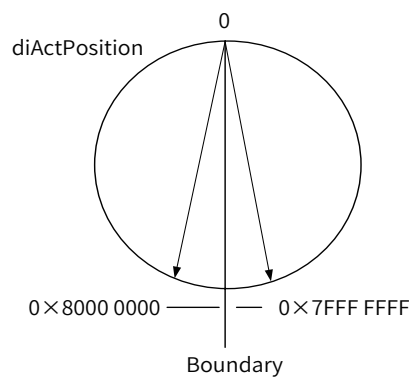
The following details SMC3_PersistPositionSingleturn_Data.

Member	Type	Initial Value	Description
fOffsetPosition	LREAL	0	Position offset
dwPosOffsetForResiduals	DWORD	0	Residual position offset error
dwActPosition	DWORD	0	Current encoder position
iIncrementsCompensated	INT	0	--
iTurn	INT	0	PLC-calculated overflow count
wCheckSum	WORD	0	Check value

4) Sequence Diagram
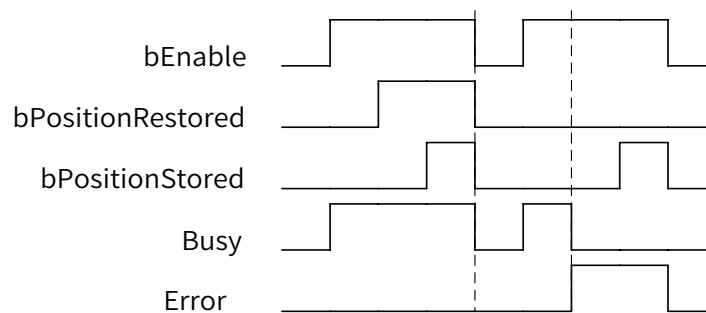
5) Error Description

The following table describes errors corresponding to **eErrorID** in MC_ERROR.

Member	Type	Enumerated Value	Description
MC_NO_ERROR	INT	0	No error
MC_PP_CRC_ERROR	INT	20000	Data recovery error, CRC error
MC_PP_THRESTHOLD_EXCEEDED	INT	20001	Difference between the current position and the saved position above the threshold

Memo No. _____

Date   /   /

Chapter 7 Diagnosis

7 Diagnosis

7.1 Overview

Diagnosis aims to quickly locate errors occurring while the PLC is running so that you can find solutions based on error information and status. The InoProShop diagnosis page can be accessed and displayed only when you log in to the PLC.

The InoProShop programming system can diagnose various communication devices and generate messages indicating faults, disconnection, and other errors based on the running status of devices.

Modules involved in fault diagnosis include CPU module, Modbus module, Modbus TCP module, EtherCAT module, CANopen module, CANlink module, and PROFIBUS DP module.

The InoProShop programming system allows you to obtain diagnosis information through the configuration diagnosis, list of diagnosis information, list of device self-diagnosis information, or diagnosis programming interface.

All diagnosis information is parsed and obtained through diagnosis codes. Diagnosis codes correspond to diagnosis programming interfaces.

7.2 Configuration Diagnosis

Configuration can be classified into network configuration and hardware configuration. The corresponding diagnosis can be classified into network configuration diagnosis and hardware configuration diagnosis. In configuration, the diagnosis state of each communication module is displayed through different icons: Running state, Stopped state, Disconnected state, and Faulty state.

: Running state: The device is running without faults.

: Stopped state: The device is stopped.

: Disconnected state: The device is disconnected or the device does not exist.

: Faulty state: The device is faulty and cannot run.

The device status is displayed on the configuration page.

7.2.1 Network Configuration Diagnosis

You can configure a PLC bus system, activate the bus, and add slaves. Log in to the system and access the network configuration page. The diagnosis state of each communication device is displayed, as shown in the following figure.
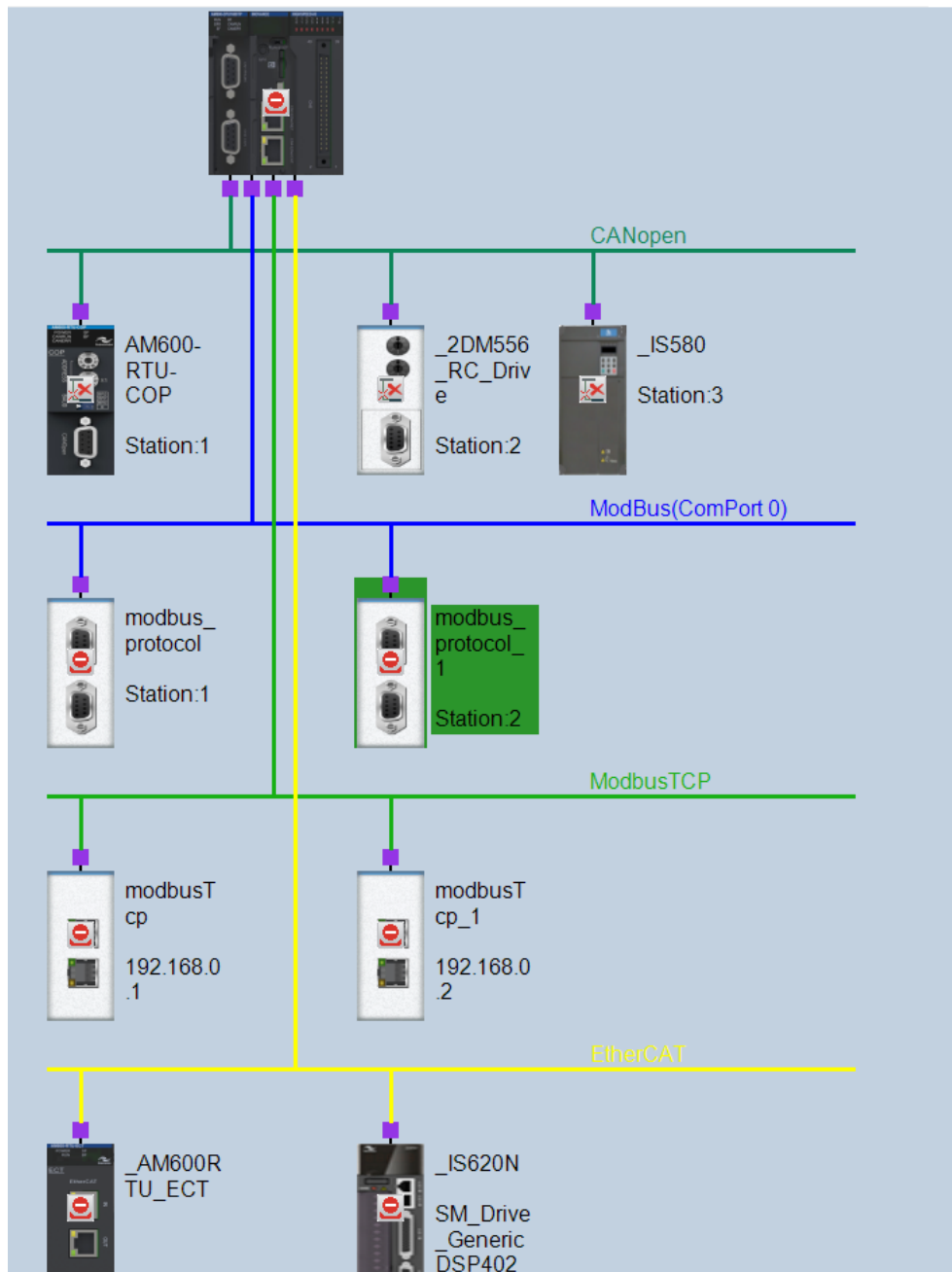
Figure 7-1 Network configuration diagnosis

After you log in, the status of each slave or CPU is displayed on the network configuration page: Running state, Faulty state, or Disconnected state. For details about network configuration, see hardware configuration.

7.2.2 Hardware Configuration Diagnosis

Hardware configuration is mainly used to add expansion modules corresponding to the bus, including local I/O hardware configuration, EtherCAT hardware configuration, and CANopen hardware configuration. CANlink, Modbus, and Modbus TCP modules are displayed only on the network configuration page. You can double click a network configuration subnode or slave module to access the hardware configuration page, or select another hardware configuration mode on the hardware configuration page. The hardware configuration diagnosis is similar to the network configuration diagnosis. The following figure shows CANopen hardware configuration diagnosis.
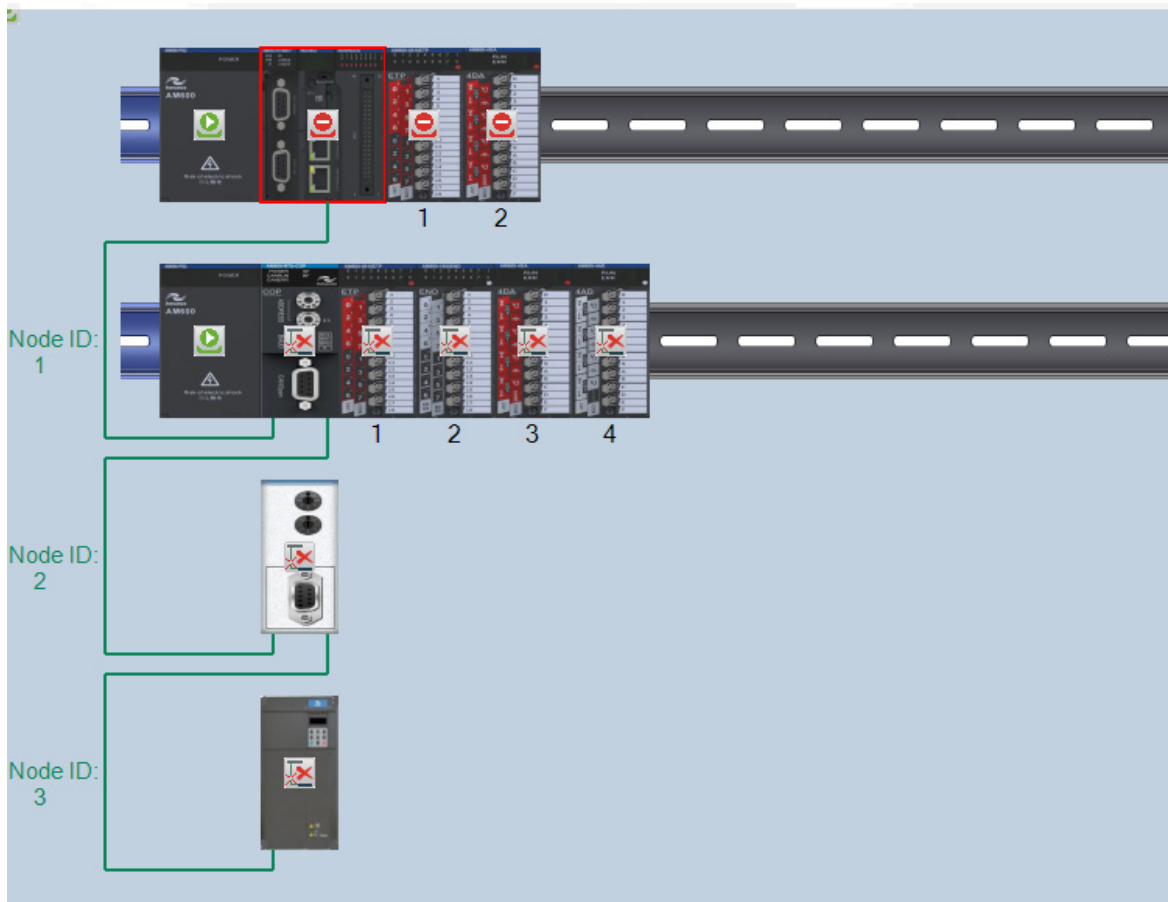
Figure 7-2 CANopen hardware configuration diagnosis

7.3 Fault Diagnosis

Fault diagnosis displays all device fault information, provides details about faults and troubleshooting, and provides diagnosis details under special circumstances.

After the device is connected, you can choose **Tool** > **Troubleshooting** to access the fault diagnosis page. The following figure shows the **Fault Diagnosis** page.
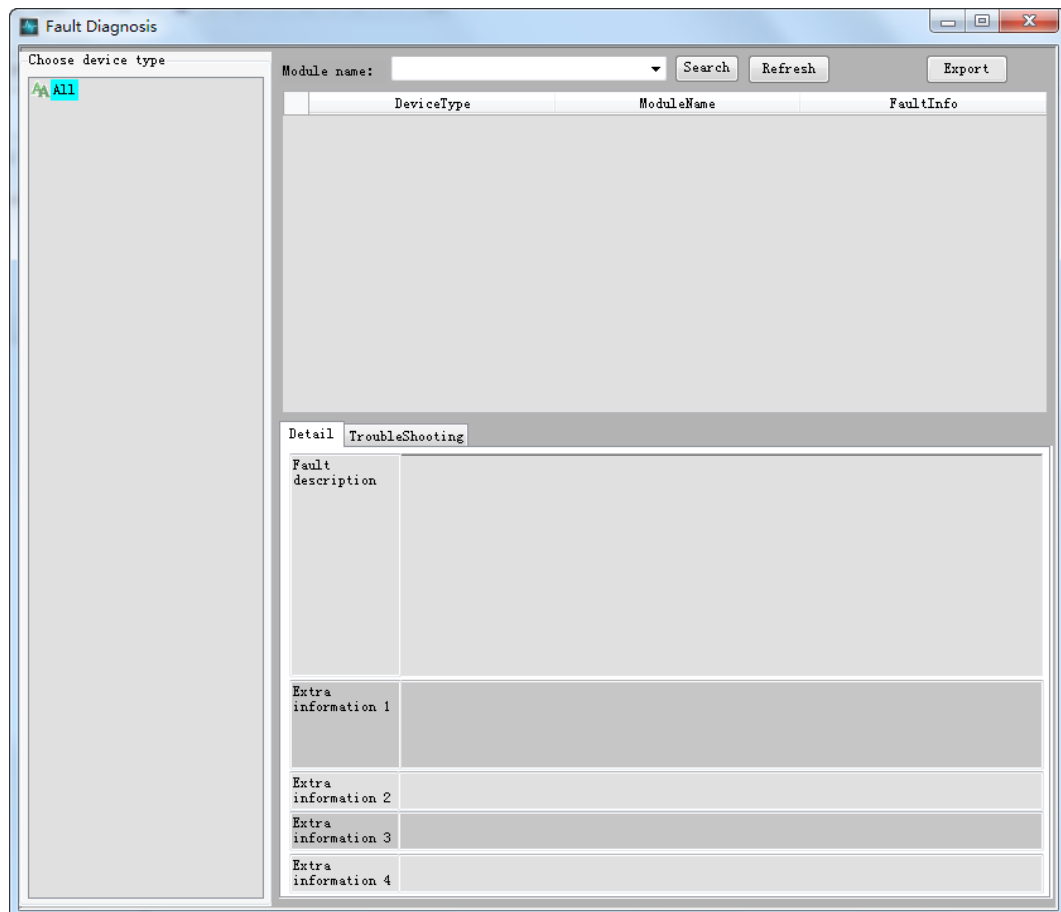
Figure 7-3 Fault diagnosis page
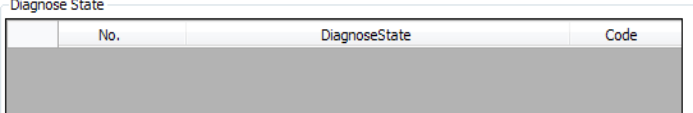
Function

■ Device Type window: On this window, device fault types are displayed, and fault information can be filtered based on the specific device type. Devices can be classified into CPU module, Modbus module, Modbus TCP module, local module, EtherCAT module, and CANopen module. Diagnosis information of corresponding types is displayed on the diagnosis list. By default, diagnosis information corresponding to all devices is displayed.

■ Buttons

a. **Module name**: Module names corresponding to different device types are included. By default, module names corresponding to all devices are included. You can use this button with the **Search** button to search for data.

b. **Search**: This button is used for Search and Filter operations based on a specific module or module name. Fuzzy search is supported.

c. **Refresh**: This button is used to refresh fault information.

d. **Export**: This button is used to export data from the list of fault information. The exported diagnosis information is in CSV format.

■ List of fault information: It is used to display module fault information, including the device type, module name, and faults.

■ Detail window: When you select a piece of fault information from the list of fault information, details about the fault are displayed in the Detail window. In this window, **Detail**, **Troubleshooting**, and **Deep Diagnosis** options are available.

a. **Detail**: Fault causes are described in the first column, which is followed by four columns of extra fault information.

b. **Troubleshooting**: Methods of troubleshooting are displayed.

c. **Deep Diagnosis**: For complex errors, more information is available to locate errors. Note: This option is not available for all error information. It is currently available for EtherCAT faults (master errors, slave errors, and ECTA module errors).

7.4 List of Device Self-diagnosis Information

In addition to the summary list of device diagnosis information, module self-diagnosis information is available on the device module page. The diagnosis state and code are displayed on the list of module self-diagnosis information. The following figure shows an example of Modbus RTU slave diagnosis dialog box.

	No.	DiagnoseState	Code

Figure 7-4 Modbus RTU slave self-diagnosis list dialog box

Diagnosis State

- **No.** indicates the diagnosis number.

- **DiagnoseState** indicates the current device diagnosis information.

- **Code** corresponds to diagnosis information. Some diagnosis information is parsed through the device state or device flag bit without a diagnosis code. For example, for the EtherCAT fault diagnosis and Modbus RTU and Modbus TCP master-slave flag bit diagnosis, only diagnosis information is displayed.

Devices with a self-diagnosis page include EtherCAT, CANopen, PROFIBUS DP, Modbus RTU, Modbus TCP, high-speed I/O, and I/O modules. No diagnosis page is available for CPU and CANlink modules. You can check diagnosis information on the list of diagnosis information.

7.4.1 CPU Diagnosis

No diagnosis page is available for a CPU. You can check diagnosis information on the list of diagnosis information.

For CPU diagnosis codes and diagnosis information, see CPU Diagnosis Code.

For descriptions of the self-diagnosis page, see the overview of the list of device self-diagnosis information.

7.4.2 EtherCAT Diagnosis

EtherCAT diagnosis is used to record and describe bus errors, including master diagnosis, slave diagnosis, slave module diagnosis, and slave drive diagnosis. EtherCAT diagnosis only parses errors of Inovance slaves. For details about diagnosis methods, see Section "7.3 Fault Diagnosis". Error IDs are listed in appendixes to this document.

In some application scenarios, error IDs are displayed on the touchscreen. You only need to assign the variable m_LastError for the EtherCAT master to a variable associated with the HMI address. As shown in the following figure, HMI_LastError is a word variable associated with the HMI address. IDs of errors diagnosed for EtherCAT are displayed on the touchscreen.

AM600 EtherCAT Slave Diagnosis

The following table lists the CANopen Emergency frame formats corresponding to the EtherCAT AM600 slave.

Emergency Error Code		Error Register	Manufacturer-specific Error Region				
BYTE0	BYTE1	BYTE2	BYTE3	BYTE4	BYTE5	BYTE6	BYTE7
BaseInfo	SlaveError	0x80	InterCommError	ConformenceError		IOModulePosError	

NOTE: The error register value 0x80 indicates the Emergency frame of the slave.

BaseInfo is not in use. For other diagnosis codes and diagnosis information, see CANopen Diagnosis Code. If the device is diagnosed in this format, the data is parsed into corresponding diagnosis information. Otherwise, the data is parsed into a message "An error occurred".

For descriptions of the self-diagnosis page, see the overview of the list of device self-diagnosis information.

7.4.3 I/O Diagnosis

The I/O module can be added to the CPU, CANopen AM600 slave, PROFIBUS DP AM600 slave, or EtherCAT AM600 slave. The CPU and slaves share the same diagnosis information. For diagnosis codes and diagnosis information, see I/O Module Diagnosis Code.

For descriptions of the self-diagnosis page, see the overview of the list of device self-diagnosis information.

7.4.4 CANopen Diagnosis

CANopen diagnosis information is obtained through the Emergency frame. On the Debug page of each slave, you can check the online status of the slave, diagnosis strings, and emergency information.

A device diagnosis page is available for the AM600 slave, used for special diagnosis of the station. For diagnosis codes and diagnosis information, see CANopen Diagnosis Code. For descriptions of the self-diagnosis page, see the overview of the list of device self-diagnosis information.

You can also obtain the CANopen slave status through CANopen special soft elements.

7.4.5 PROFIBUS DP Diagnosis

The PROFIBUS DP diagnosis refers to the PROFIBUS DP slave diagnosis. Data is included in the diagnosis array. Each slave has a Slave Diagnosis page, as shown in the following figure, on which slave diagnosis information is displayed. For non-AM600 modules of the slave, diagnosis information is displayed on the Slave Diagnosis page. For the I/O module of the AM600 PROFIBUS DP slave, diagnosis information is displayed on the diagnosis page of the I/O module. For details, see I/O Diagnosis.

The PROFIBUS DP channel diagnosis can be classified into defined channel diagnosis and GSD file-defined channel diagnosis. For details, see PROFIBUS DP Diagnosis Overview.

Figure 7-5 PROFIBUS DP Slave Diagnosis

■ **Master address**: Indicates the address of the master, which corresponds to the 4th byte in the diagnosis array.

■ **ID**: Indicates the slave-defined ID, which corresponds to the 5th and 6th bytes in the diagnosis array. The GSD file also defines the ID.

■ **Hex format**: Diagnosis array data is displayed in hex format.

■ **Standard diagnosis**: Indicates status diagnosis and identifier diagnosis included in the basic diagnosis and expanded diagnosis of slaves. For details, see PROFIBUS DP Diagnosis Overview.

■ **Channel diagnosis**: Indicates the channel diagnosis included in the expanded diagnosis of slaves, including defined channel diagnosis and GSD file-defined channel diagnosis. For details, see DP Diagnosis Overview.

PROFIBUS DP Diagnosis Overview

The following table shows the diagnosis array structure.

Table 7-1 Diagnosis array structure

First Six Bytes Indicating Basic Diagnosis Information (mandatory)	Alarm or Status Information Block (4-63 bytes) (optional)	Flag Module Diagnosis Information Block (optional)	Channel Diagnosis Information Block (3 bytes per channel) (optional)
1------------------------6 basic diagnosis information	7---244 expanded diagnosis information		
A data unit (DU) contains a minimum of 6 bytes and a maximum of 244 bytes.			

1)   Basic Diagnosis Information

1st byte:

The station does not exist (set by the master).

The station is not ready to exchange data.

The configuration is incorrect (the slave does not receive the latest configuration data).

The slave has expanded diagnosis information.

The slave does not support the requested function.

The slave response is invalid (set by the master, always 0).

The parameter is incorrect.

The slave is locked by different masters.

2nd byte:

Reset the parameter.

It is a static diagnosis.

Always 1

It is a monitoring activation ingress (1 when watchdog is activated).

The slave is in Freeze mode.

The slave is in Sync mode.

Reserved

The slave is not activated by the master (set by the master).

3rd byte:

Reserved

The diagnosis data overflows. The slave has excessive diagnosis data reports.

The 4th byte indicates the master address, ranging from 0 to 7Dh (0 to 125). When the value is FFh (255), the slave is not controlled by any slave or is not set.

The 5th byte is a PROFIBUS ID high byte for the slave, ranging from 0 to FFh (0 to 255).

The 6th byte is a PROFIBUS ID low byte, ranging from 0 to FFh (0 to 255).

2)   Expanded Diagnosis Information

Expanded diagnosis includes status diagnosis, identifier diagnosis, and channel diagnosis.

■   Status diagnosis

7th byte:

It indicates the diagnosis data block length (including the head byte), ranging from 000100 to 111111.

00 indicates alarm or state diagnosis information, that is, device diagnosis information saved on DP-V0.
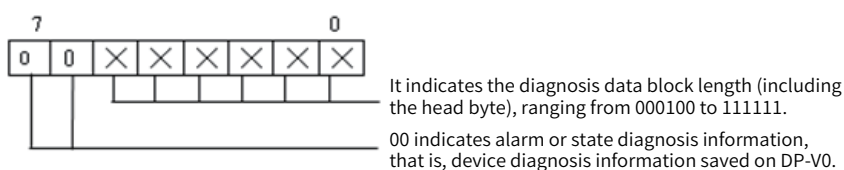
8th byte:
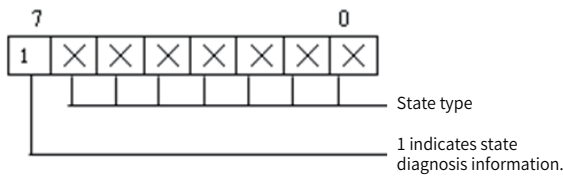
When bit 7 is 1, it indicates status diagnosis information, and bits 0 to 6 correspond to the following status information types, respectively.

0: Reserved

1: Indicates that the byte corresponding to status details is followed by status information.

2: Indicates that the byte corresponding to status details is followed by module status information (affecting the bytes following the ninth byte).

3-31: Reserved

32-126: Indicates that the byte corresponding to status details is followed by special manufacturer data.

127: Reserved

9th byte: Indicates the slot number of the slave reporting an error, ranging from 0 to 254.

10th byte: Indicates detailed features of a state.
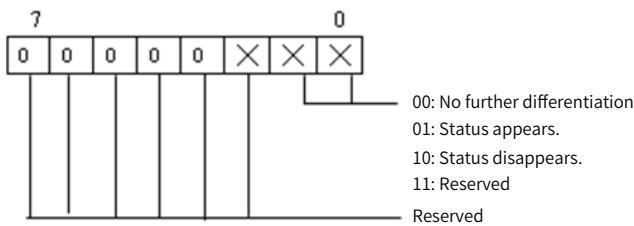
Byte following the 11th byte: Indicates a user data byte.

If the 8th byte corresponds to status type 2, that is, module status information, the 9th byte is 0; that is, the slave slot number is 0. Therefore, bytes following the 11th byte are no longer user data bytes. The following describes the structure and definition.
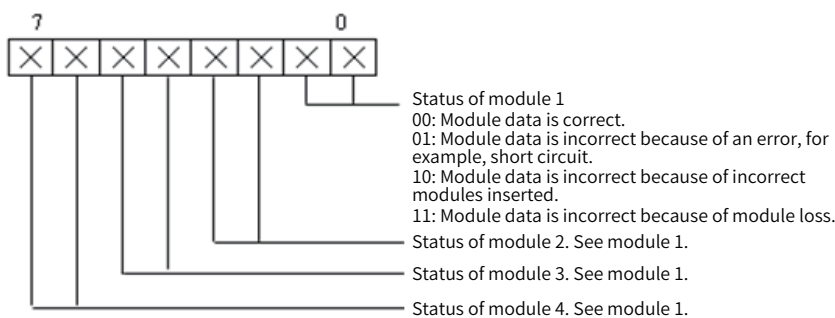
12th byte: Indicates the status of modules 5 to 8.
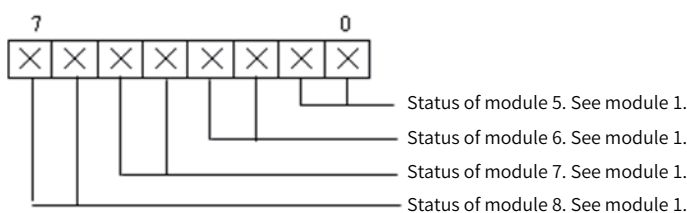
Bytes that follow can be arranged based on the preceding rule until information on all modules is entered.

■ Identifier diagnosis

Head Byte	Diagnosis Data Byte Relating to Flag (Module)
01××××××	Bytes 1-62

Similar to the preceding head byte, the head byte indicates the type and length of flag diagnosis information (the number of bytes), including the head byte. The following details the head byte structure and definition.

It indicates the diagnosis data block length (including the head byte), ranging from 000010 to 111111.
01 indicates flag (module) diagnosis information.

Flag byte 0 (module 1) indicates diagnosis information.
Flag byte 1 (module 2) indicates diagnosis information.
Flag byte 2 (module 3) indicates diagnosis information.
Flag byte 3 (module 4) indicates diagnosis information.
Flag byte 4 (module 5) indicates diagnosis information.
Flag byte 5 (module 6) indicates diagnosis information.
Flag byte 6 (module 7) indicates diagnosis information.
Flag byte 7 (module 8) indicates diagnosis information.

If the number of modules exceeds 8, you can continue using bytes that follow to specify flag byte numbers (or module numbers).

■ Channel diagnosis

Each piece of channel diagnosis information contains 3 bytes. Channel diagnosis includes diagnosis information for multiple channels. The following table shows the structure of diagnosis information for one channel.

Head Byte	Diagnosis Data Byte Relating to Channel
10×××××	2 bytes (3 bytes if the head byte is included)

The head byte specifies the type of channel diagnosis information and the number of a faulty module. The following describes the structure and definition of the head byte.

It indicates the number of faulty module, ranging from 000000 to 111111. For example, 000010 indicates that module 3 is faulty, and 111111 indicates that module 64 is faulty.
10 indicates channel diagnosis information.
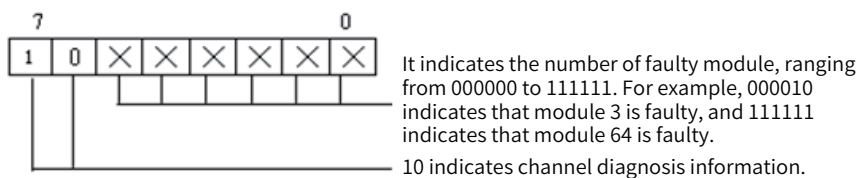
The 2nd byte indicates the channel byte type. The following details the structure and definition.

It indicates the number of faulty channel, ranging from 000000 to 111111. For example, 000010 indicates that module 3 is faulty, and 111111 indicates that module 64 is faulty.
01: The channel byte is input.
10: The channel byte is output.
11: The channel byte is input/output.
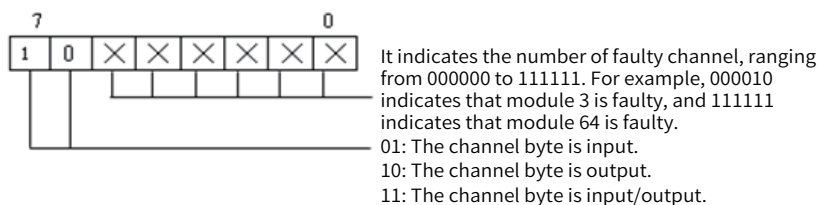
The 3rd byte indicates the byte length and fault type of the channel.
The following details the structure and definition.

001: It is a bit channel.
010: It is a 2-bit channel.
011: It is a 4-bit channel.
100: It is a byte channel.
101: It is a word channel.
110: It is a double-word channel.

1: Short circuit
2: Low voltage
3: High voltage
4: Overload
5: High temperature
6: Broken cable
7: Above the upper limit
8: Below the lower limit
9: General
10 to 15: Reserved
16 to 31: Related to the device
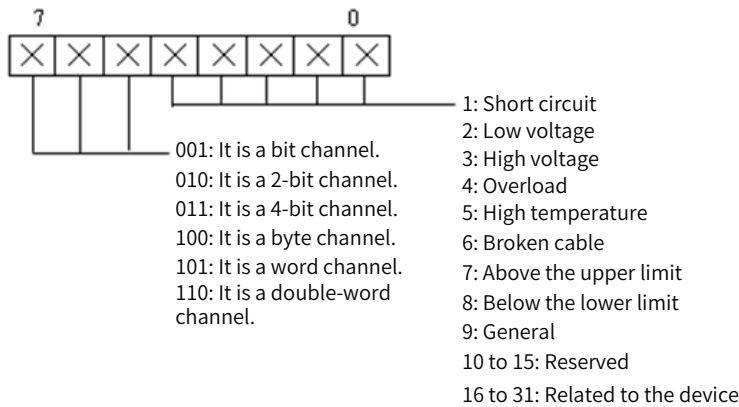
7.4.6 Modbus RTU Diagnosis

Modbus RTU supports buses of Modbus serial ports 0 and 1. Modbus serial port 0 or 1 can serve as a Modbus master or Modbus slave.

When a Modbus serial port serves as a master, you can add slaves (remote). On both master configuration page and slave configuration page, the **Device Diagnosis** option is displayed. Master diagnosis information is used to identify slave configuration faults without fault causes. Therefore, no fault codes are displayed on the Device Diagnosis page. On the Device Diagnosis page for a slave, fault information corresponding to specific configuration items is displayed.

When a Modbus serial port serves as a slave, a Device Diagnosis page is available, on which master-slave communication faults are displayed. For details, see the list of device self-diagnosis information.

The diagnosis codes and diagnosis information for a Modbus serial port remain unchanged whether it serves as a master or a slave. See Modbus diagnosis codes for details.

7.4.7 Modbus TCP Diagnosis

An AM600 PLC can serve as a Modbus TCP master or a Modbus TCP slave.

When a Modbus TCP device serves as a master, you can add slaves (remote). On both master configuration page and slave configuration page, the **Device Diagnosis** option is displayed. Master diagnosis information is used to identify slave configuration faults without fault causes. Therefore, no fault codes are displayed on the Device Diagnosis page. On the Device Diagnosis page for a slave, fault information corresponding to specific configuration items is displayed.

When a Modbus TCP device serves as a slave, a Device Diagnosis page is available, on which master-slave communication faults are displayed. For details, see the list of device self-diagnosis information.

The diagnosis codes and diagnosis information for a Modbus TCP device remain unchanged whether it serves as a master or a slave. See Modbus diagnosis codes for details.

7.4.8 CANlink Diagnosis

No Device Diagnosis page is available for CANlink devices. However, you can enable the monitoring function on the CANlink network management page to check the online status and running status of the slave. For details, see CANlink Network Management.

You can check the status of CANlink stations through soft elements. For details, see CANlink Soft Elements.

After logging in to the PLC, you can view CANlink diagnosis information from the list of diagnosis information. For CANlink diagnosis codes and diagnosis information, see CANlink Diagnosis Code.

For descriptions of the self-diagnosis page, see the overview of the list of device self-diagnosis information.

7.5 Diagnosis Programming Interface

Note: Only AM400 and AM600 models support this function.

7.5.1 Overview

A diagnosis programming interface allows you to obtain diagnosis information from user programs: you can check diagnosis information for modules in user programs and take action based on the information.

A diagnosis programming interface exists in library form. You can add it on the **Library Manager** interface, as shown in the following figure.
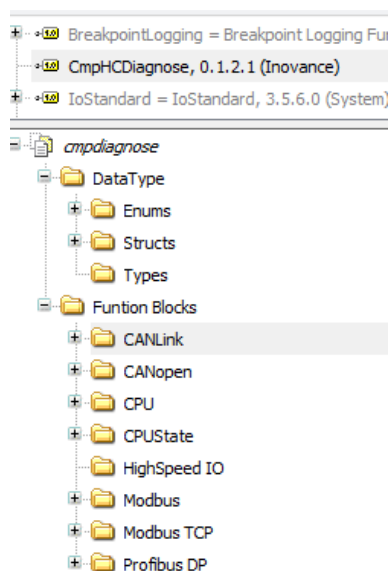
Programming interfaces include diagnosis programming interfaces corresponding to CANlink, CANopen, CPU, Modbus, Modbus TCP, and PROFIBUS DP modules, respectively. Each diagnosis corresponds to one function block so that you can obtain corresponding diagnosis codes.

User-defined diagnosis results and diagnosis states are defined in **DataType**. **HC_enumERROR** indicates whether diagnosis succeeds, as shown in the following table.

Enumerator (hexadecimal)	Value (decimal)	Description
NO_ERROR	0	No error
WRONG_PARAMETER	1	Parameter error
UNKNOWN_DEVICEID	2	Unknown device ID
INVAILD_DEVICEID	3	Invalid device ID
INVAILD_IO_POS	4	Invalid I/O position
UNSUPPORT_DIAGNOSE	5	Unsupported diagnosis
TIME_OUT	6	Timeout
INTERNAL_FB_ERROR	7	Internal function block error
UNKNOWN_ERROR	8	Unknown error
INVAILD_IP	9	Invalid IP address

7.5.2 CPU Diagnosis Programming Interface

1 CPU Diagnosis Programming Interface

Obtaining CPU Diagnosis Data: GET_CPU_DIAGNOSE			

Parameter Name	Parameter Type	Initial Value	Description
Input parameter			
xEnable	BOOl	FALSe	Indicates the enabling bit, triggered by level.
Output parameter			
xDone	Bool	FALSE	Indicates whether the diagnosis result is obtained.
eError	HC_enumERROR	NO_ERROR	Indicates whether the diagnosis result is obtained.
sCPUDiagnoseData	HC_tagDIAGNOSE_DATA_CPU		Indicates CPU diagnosis data.

HC_tagDIAGNOSE_DATA_CPU data is structure data, as shown in the following table. The correlation between the diagnosis code and diagnosis information is detailed in CPU Diagnosis Code.

Name	Type
SDCardError	BYTE
FlashError	BYTE
SystemError	BYTE
InterCommError	BYTE
ConformenceError	WORD
IOModulePosError	WORD
FunctionErrorCode	WORD

Example

```
PROGRAM POU

VAR

    get_cpu_diag: GET_CPU_DIAGNOSE;

    Enable: BOOL;

    eError: HC_enumERROR;

    xDone: BOOL;

    sCPUDiag: HC_tagDIAGNOSE_DATA_CPU;

END_VAR
```

2 CPU Local I/O Expansion Module

| Obtaining CPU I/O Diagnosis Data: GET_CPU_IOMODULE_DIAGNOSE |

Parameter Name	Parameter Type	Initial Value	Description
Input parameter			
xEnable	BOOl	FALSe	Indicates the enabling bit, triggered by level.
byModulePos	BYTE (1..16)	0	Indicates the obtained I/O position.
Output parameter			
xDone	Bool	FALSE	Indicates whether the diagnosis result is obtained.
eError	HC_enumERROR	NO_ERROR	Indicates whether the diagnosis result is obtained.
sIODiagnoseData	HC_tagDIAGNOSE_DATA_IOMODULE		Indicates I/O diagnosis data.

HC_tagDIAGNOSE_DATA_IOMODULE data is structure data, as shown in the following table. The correlation between the diagnosis code and diagnosis information is detailed in I/O Diagnosis Code.

Structure Member	Type	Description
ModuleError	BYTE	Indicates the module error.
ChannelError	ARRAY[0..3] OF BYTE	Indicates the channel error.

Example

```
PROGRAM POU

VAR

    get_cpu_iomodule_diag: GET_CPU_IOMODULE_DIAGNOSE;

    Enable: BOOL;

    eError: HC_enumERROR;

    xDone: BOOL;

    byModulePos: BYTE (1..16);

    sIODiag: HC_tagDIAGNOSE_DATA_IOMODULE;

END_VAR
```
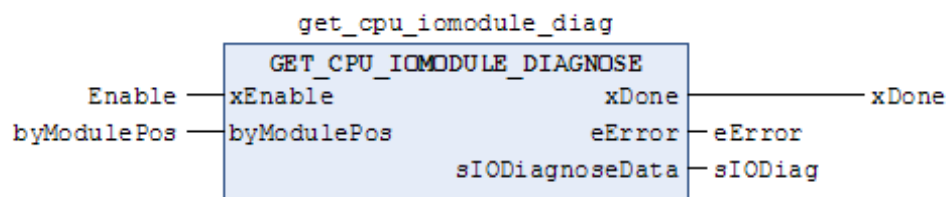
7.5.3 CANopen Diagnosis Programming Interface

1 CANopen Slave Diagnosis Programming Interface

Obtaining CANopen Slave Diagnosis Data: GET_CANOPEN_SLAVE_DIAGNOSE			

Parameter Name	Parameter Type	Initial Value	Description
Input parameter			
xEnable	BOOl	FALSe	Indicates the enabling bit, triggered by level.
bySlaveID	BYTE (1..127)	0	Indicates the obtained slave node ID, ranging from 1 to 127.
Output parameter			
xDone	Bool	FALSE	Indicates whether the diagnosis result is obtained.
eError	HC_enumERROR	NO_ERROR	Indicates whether the diagnosis result is obtained.
sSlaveDiagnoseData	HC_tagDIAGNOSE_DATA_SLAVE		Indicates CANopen slave diagnosis data.

HC_tagDIAGNOSE_DATA_SLAVE data is structure data, as shown in the following table. The correlation between the diagnosis code and diagnosis information is detailed in CANopen Diagnosis Code.

Structure Member	Type	Description
SlaveError	BYTE	Indicates the slave error.
InterCommError	BYTE	Indicates the internal communication error.
ConformenceError	WORD	Indicates the consistency error.
IOModulePosError	WORD	Indicates the I/O module position error.

Example

```
PROGRAM POU

VAR

    get_CANOpen_slave_diag: GET_CANOPEN_SLAVE_DIAGNOSE;

    Enable: BOOL;

    eError: HC_enumERROR;

    xDone: BOOL;

    bySlaveId: BYTE (1..127);

    sSlaveDiag: HC_tagDIAGNOSE_DATA_SLAVE;

END_VAR
```
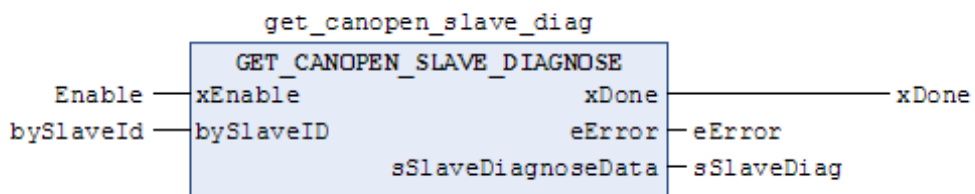
2 CANopen Slave I/O Diagnosis Programming Interface

Obtaining CANopen Slave I/O Diagnosis Data: GET_CANOPEN_IOMODULE_DIAGNOSE			

GET_CANOPEN_IOMODULE_DIAGNOSE

```
—xEnable   BOOL                                              BOOL xDone—
—bySlaveID BYTE (1..127)                            HC_enumERROR eError—
—byModulePos BYTE (1..16)        HC_tagDIAGNOSE_DATA_IOMODULE sIODiagnoseData—
```

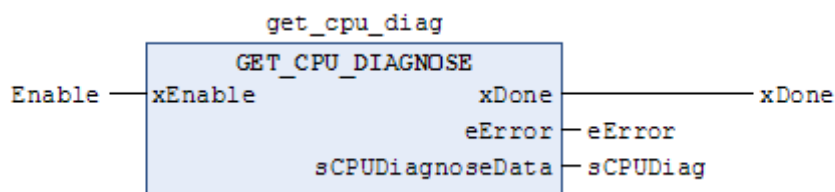| Parameter Name	Parameter Type	Initial Value	Description
Input parameter			
xEnable	BOOl	FALSE	Indicates the enabling bit, triggered by level.
bySlaveID	BYTE (1..127)	0	Indicates the slave node ID, ranging from 1 to 127.
byModulePos	BYTE (1..16)	0	Indicates the diagnosed I/O position.
Output parameter			
xDone	Bool	FALSE	Indicates whether the diagnosis result is obtained.
eError	HC_enumERROR	NO_ERROR	Indicates whether the diagnosis result is obtained.
sIODiagnoseData	HC_tagDIAGNOSE_DATA_IOMODULE		Indicates I/O diagnosis data.

HC_tagDIAGNOSE_DATA_IOMODULE data is structure data, as shown in the following table. The correlation between the diagnosis code and diagnosis information is detailed in I/O Diagnosis Code.

Structure Member	Type	Description
ModuleError	BYTE	Indicates the module error.
ChannelError	ARRAY[0..3] OF BYTE	Indicates the channel error.

Example

PROGRAM POU

VAR

    get_CANOpen_iomodule_diag: GET_CANOPEN_IOMODULE_DIAGNOSE;

 Enable: BOOL;

    eError: HC_enumERROR;

    xDone: BOOL;

    bySlaveId: BYTE (1..127);

    byModulePos: BYTE (1..16);

    sIODiag: HC_tagDIAGNOSE_DATA_IOMODULE;

END_VAR

```
                   get_canopen_iomodule_diag
              GET_CANOPEN_IOMODULE_DIAGNOSE
   Enable ——xEnable                    xDone ———————— xDone
 bySlaveId —— bySlaveID                eError — eError
byModulePos —— byModulePos    sIODiagnoseData — sIODiag
```

7.5.4 PROFIBUS DP Diagnosis Programming Interface

1 PROFIBUS DP Slave Diagnosis Programming Interface

Obtaining DP Slave Diagnosis Data: GET_DP_SLAVE_DIAGNOSE			

GET_DP_SLAVE_DIAGNOSE			
xEnable *BOOL*			*BOOL* xDone
bySlaveID *BYTE (1..125)*			*HC_enumERROR* eError
			HC_tagDIAGNOSE_DATA_SLAVE_DP sSlaveDiagnoseData

| Parameter Name	Parameter Type	Initial Value	Description
Input parameter			
xEnable	BOOl	FALSe	Indicates the enabling bit, triggered by level.
bySlaveID	BYTE (1..125)	0	Indicates the obtained slave address, ranging from 1 to 125.
Output parameter			
xDone	Bool	FALSE	Indicates whether the diagnosis result is obtained.
eError	HC_enumERROR	NO_ERROR	Indicates whether the diagnosis result is obtained.
sSlaveDiagnoseData	HC_tagDIAGNOSE_DATA_SLAVE_DP		Indicates DP slave diagnosis data.

HC_tagDIAGNOSE_DATA_SLAVE_DP data is structure data, as shown in the following table. The correlation between the diagnosis code and diagnosis information is detailed in DP Diagnosis Code.

Structure Member	Type	Description
Length	BYTE	Indicates the diagnosis data length.
ExtDiagData	ARRAY[0..243]OF BYTE	Indicates diagnosis data.

Example

```
PROGRAM POU

VAR

    get_dp_slave_diag: GET_DP_SLAVE_DIAGNOSE;

    Enable: BOOL;

    eError: HC_enumERROR;

    xDone: BOOL;

    bySlaveID: BYTE (1..125);

    sSlaveDiagData: HC_tagDIAGNOSE_DATA_SLAVE_DP;

END_VAR
```
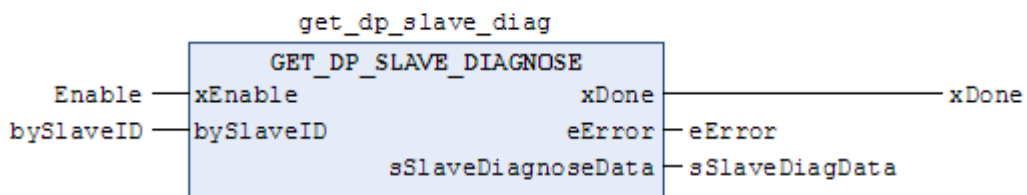
2 PROFIBUS DP Slave I/O Diagnosis Programming Interface

Obtaining DP Slave I/O Diagnosis Data: GET_DP_IOMODULE_DIAGNOSE			
GET_DP_IOMODULE_DIAGNOSE			
xEnable *BOOL* bySlaveID *BYTE (1..125)* byModulePos *BYTE (1..16)*		*BOOL* xDone *HC_enumERROR* eError *HC_tagDIAGNOSE_DATA_IOMODULE* sIODiagnoseData	

Parameter Name	Parameter Type	Initial Value	Description
Input parameter			
xEnable	BOOl	FALSE	Indicates the enabling bit, triggered by level.
bySlaveID	BYTE (1..125)	0	Indicates the slave address, ranging from 1 to 125.
byModulePos	BYTE (1..16)	0	Indicates the diagnosed I/O position.
Output parameter			
xDone	Bool	FALSE	Indicates whether the diagnosis result is obtained.
eError	HC_enumERROR	NO_ERROR	Indicates whether the diagnosis result is obtained.
sIODiagnoseData	HC_tagDIAGNOSE_DATA_IOMODULE		Indicates I/O diagnosis data.

HC_tagDIAGNOSE_DATA_IOMODULE data is structure data, as shown in the following table. The correlation between the diagnosis code and diagnosis information is detailed in I/O Diagnosis Code.

ModuleError	BYTE
ChannelError	ARRAY [0..3] OF BYTE

Example

```
PROGRAM POU

VAR

    get_dp_iomodule_diag: GET_DP_IOMODULE_DIAGNOSE;

    Enable: BOOL;

    eError: HC_enumERROR;

    xDone: BOOL;

    bySlaveID: BYTE (1..125);

    byModulePos: BYTE (1..16);

    sIODiagData: HC_tagDIAGNOSE_DATA_IOMODULE;

END_VAR
```
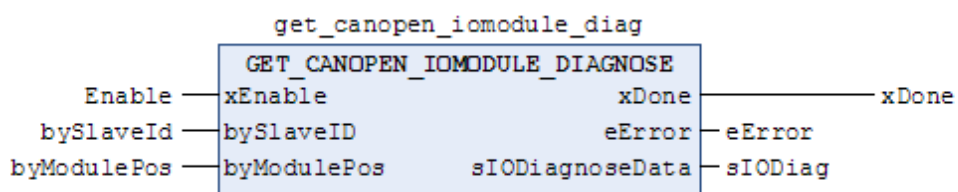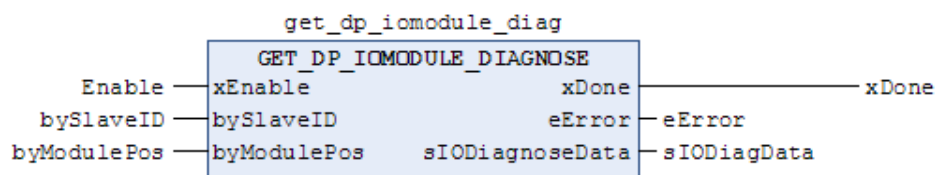
7.5.5 CANlink Diagnosis Programming Interface

Obtaining CANlink Diagnosis Data: GET_CANLINK_DIAGNOSE			
	GET_CANLINK_DIAGNOSE		
xEnable *BOOL*			*BOOL* xDone
byStationID *BYTE (1..63)*			*HC_enumERROR* eError
			HC_tagDIAGNOSE_DATA_CANLINK sCanlinkDiagnoseData

| Parameter Name	Parameter Type	Initial Value	Description
Input parameter			
xEnable	BOOl	FALSe	Indicates the enabling bit, triggered by level.
byStationID	BYTE (1..63)	0	Indicates the obtained station node ID, ranging from 1 to 63.
Output parameter			
xDone	Bool	FALSE	Indicates whether the diagnosis result is obtained.
eError	HC_enumERROR	NO_ERROR	Indicates whether the diagnosis result is obtained.
sCanlinkDiagnoseData	HC_tagDIAGNOSE_DATA_CANLINK		Indicates CANlink station diagnosis data.

HC_tagDIAGNOSE_DATA_CANLINK data is structure data, as shown in the following table. The correlation between the diagnosis code and diagnosis information is detailed in CANlink Diagnosis Code.

Structure Member	Type	Description
IsUsed	BOOL	Indicates whether it is used.
IsMaster	BOOL	Indicates whether it is the master.
StationStatus	WORD	Indicates the CANlink station status.
CfgFrameError	WORD	Indicates the configuration frame error.
CmdFrameError	WORD	Indicates the command frame error.

Example

```
PROGRAM POU

VAR

    get_canlink_diagnose:GET_CANLINK_DIAGNOSE;

 Enable: BOOL;

    StationId: BYTE (1..63);

    eError: HC_enumERROR;

    canLinkDiagData: HC_tagDIAGNOSE_DATA_CANLINK;

    xDone: BOOL;

END_VAR
```
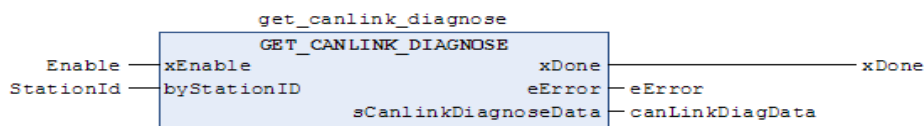
7.5.6 Modbus Diagnosis Programming Interface

1 Modbus Local Slave Diagnosis Programming Interface

Obtaining Modbus Local Slave Diagnosis Data: GET_MODBUS_SLAVE_DEVICE_DIAGNOSE			
Parameter Name	Parameter Type	Initial Value	Description
Input parameter			
xEnable	BOOl	FALSe	Indicates the enabling bit, triggered by level.
byComID	BYTE (0..1)	0	Indicates the serial port number for the local slave, ranging from 0 to 1.
Output parameter			
xDone	Bool	FALSE	Indicates whether the diagnosis result is obtained.
eError	HC_enumERROR	NO_ERROR	Indicates whether the diagnosis result is obtained.
byDiagData	Byte		Indicates the diagnosis code. The correlation between the diagnosis code and diagnosis information is detailed in Modbus Diagnosis Code.

Example

```
PROGRAM POU

VAR

    get_modbus_slave_dev_diag: GET_MODBUS_SLAVE_DEVICE_DIAGNOSE;

   Enable: BOOL;

    eError: HC_enumERROR;

    xDone: BOOL;

    byComId: BYTE (0..1);

    byDiagData: BYTE;

END_VAR
```
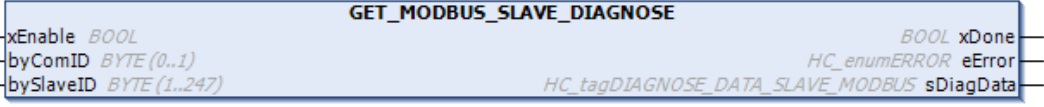
2 Modbus Remote Slave Diagnosis Programming Interface

Obtaining Modbus Remote Slave Diagnosis Data: GET_MODBUS_SLAVE_DIAGNOSE			
GET_MODBUS_SLAVE_DIAGNOSE			

```
GET_MODBUS_SLAVE_DIAGNOSE
—xEnable  BOOL                                         BOOL xDone—
—byComID  BYTE (0..1)                            HC_enumERROR eError—
—bySlaveID BYTE (1..247)        HC_tagDIAGNOSE_DATA_SLAVE_MODBUS sDiagData—
```

Parameter Name	Parameter Type	Initial Value	Description
Input parameter			
xEnable	BOOl	FALSE	Indicates the enabling bit, triggered by level.
byComID	BYTE (0..1)	0	Indicates the serial port number for the master, ranging from 0 to 1.
bySlaveID	BYTE (1..247)	0	Indicates the slave address, ranging from 1 to 247.
Output parameter			
xDone	Bool	FALSE	Indicates whether the diagnosis result is obtained.
eError	HC_enumERROR	NO_ERROR	Indicates whether the diagnosis result is obtained.
sDiagData	HC_tagDIAGNOSE_DATA_SLAVE_MODBUS		Indicates slave diagnosis data.

HC_tagDIAGNOSE_DATA_SLAVE_MODBUS data is structure data, as shown in the following table. The correlation between the diagnosis code and diagnosis information is detailed in Modbus Diagnosis Code.

Name	Type
ChannelNum	BYTE
DiagData	BYTE

Example

```
PROGRAM POU

VAR

    get_modbus_slave_diag: GET_MODBUS_SLAVE_DIAGNOSE;

    Enable: BOOL;

    eError: HC_enumERROR;

    xDone: BOOL;

    byComId: BYTE (0..1);

    bySlaveId: BYTE (1..247);

    sDiagData: HC_tagDIAGNOSE_DATA_SLAVE_MODBUS;

END_VAR
```

```
                        get_modbus_slave_diag
              GET_MODBUS_SLAVE_DIAGNOSE
    Enable —xEnable                    xDone ——————— xDone
   byComId —byComID                    eError — eError
  bySlaveId —bySlaveID              sDiagData — sDiagData
```

7.5.7 Modbus TCP Diagnosis Programming Interface

Modbus TCP Local Slave Diagnosis Programming Interface

Obtaining Modbus TCP Local Slave Diagnosis Data: GET_MODBUSTCP_SLAVE_DEVICE_DIAGNOSE			
GET_MODBUSTCP_SLAVE_DEVICE_DIAGNOSE —xEnable *BOOL* *BOOL* xDone— *HC_enumERROR* eError— *BYTE* byDiagData—			
Parameter Name	Parameter Type	Initial Value	Description
Input parameter			
xEnable	BOOl	FALSe	Indicates the enabling bit, triggered by level.
Output parameter			
xDone	Bool	FALSE	Indicates whether the diagnosis result is obtained.
eError	HC_enumERROR	NO_ERROR	Indicates whether the diagnosis result is obtained.
byDiagData	Byte		Indicates the diagnosis code. The correlation between the diagnosis code and diagnosis information is detailed in Modbus Diagnosis Code.

Example

```
PROGRAM POU

VAR

    get_modbustcp_slave_dev_diag: GET_MODBUSTCP_SLAVE_DEVICE_DIAGNOSE;

    Enable: BOOL;

    eError: HC_enumERROR;

    xDone: BOOL;

    byDiagData: BYTE;

END_VAR
```
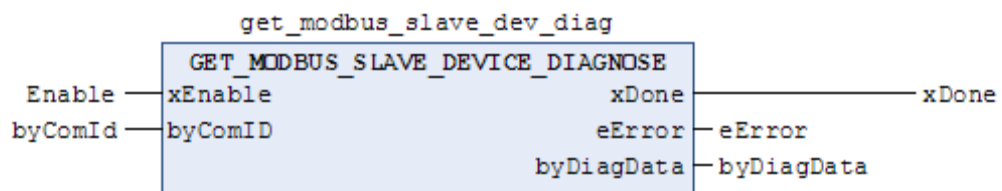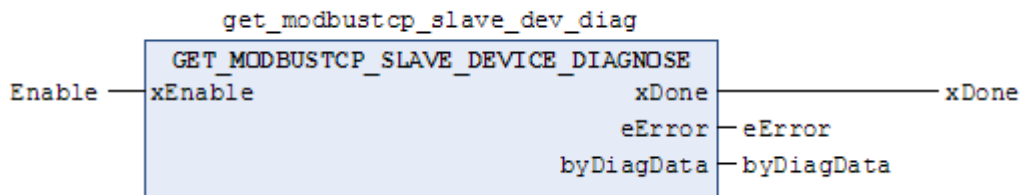
Modbus TCP Remote Slave Diagnosis Programming Interface

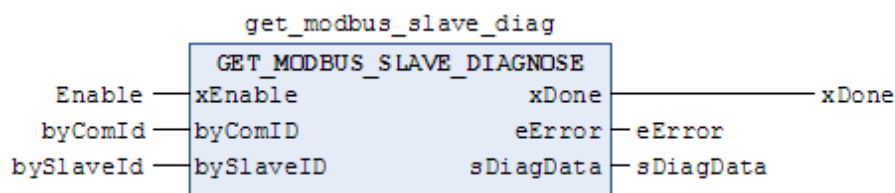| Obtaining Modbus TCP Remote Slave Diagnosis Data: GET_MODBUSTCP_SLAVE_DIAGNOSE			
GET_MODBUSTCP_SLAVE_DIAGNOSE —xEnable *BOOL* —strSlaveIP *STRING(15)* *BOOL* xDone— *HC_enumERROR* eError— *HC_tagDIAGNOSE_DATA_SLAVE_MODBUS* sDiagData—			
Parameter Name	Parameter Type	Initial Value	Description
Input parameter			
xEnable	BOOl	FALSE	Indicates the enabling bit, triggered by level.
strSlaveIP	STRING(15)	''	Indicates the remote slave IP address.
Output parameter			
xDone	Bool	FALSE	Indicates whether the diagnosis result is obtained.
eError	HC_enumERROR	NO_ERROR	Indicates whether the diagnosis result is obtained.
sDiagData	HC_tagDIAGNOSE_DATA_SLAVE_MODBUS		Indicates slave diagnosis data.

HC_tagDIAGNOSE_DATA_SLAVE_MODBUS data is structure data, as shown in the following table. The correlation between the diagnosis code and diagnosis information is detailed in Modbus Diagnosis Code.

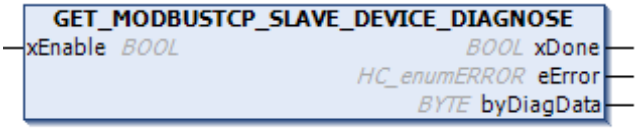| Name	Type
ChannelNum	BYTE
DiagData	BYTE

Example
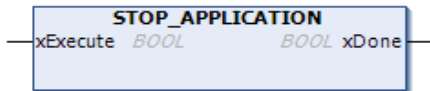
```
PROGRAM POU

VAR

    get_modbustcp_slave_diag: GET_MODBUSTCP_SLAVE_DIAGNOSE;

   Enable: BOOL;

    eError: HC_enumERROR;

    xDone: BOOL;

    sDiagData: HC_tagDIAGNOSE_DATA_SLAVE_MODBUS;

    strSlaveIP: STRING(15);

END_VAR
```
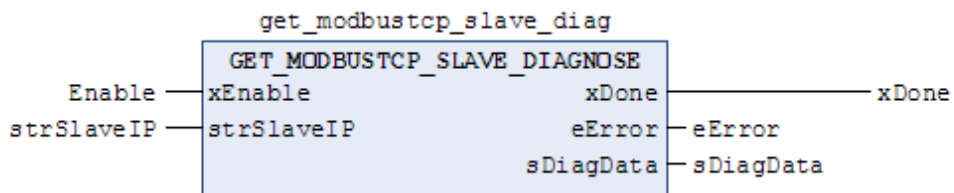
7.5.8 CPU Stop Control

Description of function blocks

Stopping Application Program: STOP_APPLICATION			
STOP_APPLICATION —xExecute *BOOL* *BOOL* xDone—			
Parameter Name	Parameter Type	Initial Value	Description
Input parameter			
xExecute	BOOL	FALSE	Indicates the enabling bit, triggered by rising edges.
Output parameter			
xDone	BOOL	FALSE	Indicates execution complete output.

Example

```
PROGRAM POU
VAR
    stop_app: STOP_APPLICATION;
    xExecute: BOOL;
END_VAR
```

```
                         stop_app
                    STOP_APPLICATION
xExecute ———————  xExecute      xDone  ——————— xDone
```
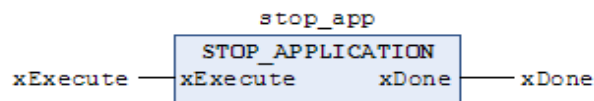
7.5.9 EtherCAT Diagnosis

EtherCAT diagnosis is used to record and describe bus errors, including master diagnosis, slave diagnosis, slave module diagnosis, and slave drive diagnosis. EtherCAT diagnosis only parses errors of Inovance slaves. For details about diagnosis methods, see Section 7.3 Fault Diagnosis. Error IDs are listed in appendices to this document.

In some application scenarios, error IDs are displayed on the touchscreen. You only need to assign the variable m_LastError for the EtherCAT master to a variable associated with the HMI address. As shown in the following figure, HMI_LastError is a word variable associated with the HMI address. IDs of errors diagnosed for EtherCAT are displayed on the touchscreen.

Memo No. _____

Date   /   /
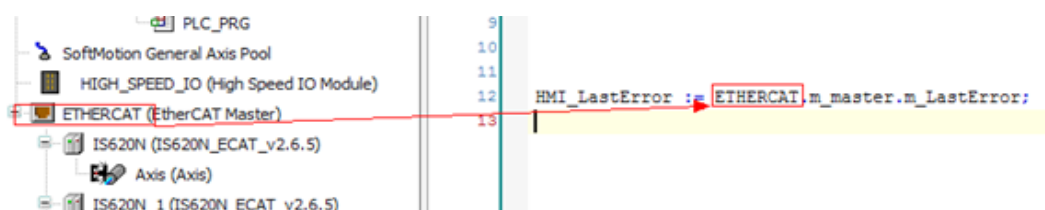
Appendix

Appendix

Appendix A Communication Protocols for Communication Ports

By default, medium-sized PLCs are provided with Mini-USB ports, serial communication ports, Ethernet ports, EtherCATports, Mini-SD card slots, CAN communication ports, PROFIBUS DP communication ports, high-speed I/O interfaces, and local bus expansion interfaces. The following describes how to set protocols for ports.

A.1 Mini-USB Port and Built-in Communication Protocol

The Mini-USB port is used to download PLC user programs, monitor and debug the system. Therefore, the port has a specific communication protocol, and you do not need to select one. As long as a USB drive is installed on the PC, you can use InoProShop to download or monitor user programs of the medium-sized PLC on the PC at any time.

As the built-in download protocol for the Mini-USB port is a special protocol of Inovance, you cannot download programs of medium-sized PLCs through third-party programming software.

After the InoProShop programming software is installed for the first time, the USB drive is automatically installed. To install different versions of InoProShop on one PC, you need to install them in different directories.

A.2 COM0/COM1 Communication Port and Built-in Protocol

COM0 and COM1 ports are basic ports of a PLC for external communication. They are integrated on one DB9 physical port, mainly used for RS-485 or Modbus communication.

The following table lists protocols supported by COM0 and COM1 ports as well as definitions of set units.

COM0/COM1 Protocol	Half-duplex/Full-duplex Mode	Communication Format	Baud Rate	Data Bit	Stop Bit	Parity Check
Modbus-RTU master	Half-duplex	Fixed	4,800 Bits/s 9,600 Bits/s	7bit 8bit		
Modbus-RTU slave	Half-duplex	Fixed	19,200 Bits/s	8bit	1bit 2bit	NONE ODD EVEN
RS-485 free protocol	Half-duplex	Unfixed	38,400 Bits/s 57,600 Bits/s 115,200 Bits/s	8bit		
Modbus-ASCII master	Not supported	N/A	N/A	N/A	N/A	N/A
Modbus-ASCII slave	Not supported	N/A	N/A	N/A	N/A	N/A

The following describes the preceding protocols.

■ Modbus master protocol

As a control host, the PLC usually uses this protocol to communicate with AC drives, servos, and other lower computers, or to read data from smart meters and sensors. PLCs communicate with each other through Modbus protocols, improving the flexibility of communication.

■ Modbus slave protocol

A host computer reads internal PLC data usually through Modbus protocols. The PLC serves as a communication slave. When you set the port as a Modbus slave on the PLC, the PLC automatically responds based on the communication command of the host computer.

■ Free communication protocol

Protocols other than PLC built-in communication protocols are called "free communication protocols". To enable communication through a free protocol, programmers must fully understand the frame structure definition of the protocol. Programmers prepare date strings (stored in the register) in user programs in advance based on the slave communication protocol and required communication operations. The system specifies data in the register and automatically sends the data to serial ports successively. Then the serial ports receive the data. The system saves the received data in specified regions. Upon receipt of data of specified length, the serial ports notify user programs through system flags so that user programs can parse received data based on the protocol.

The register can be operated through AM600 free communication protocols. That is, user programs can access the communication buffer to process the data sending and receiving buffer so that communication can be enabled through user-defined protocols. You need to configure and prepare for serial communication during programming so that communication can be conducted based on requirements. Related tasks include configuring the data sending/receiving mode of serial ports, baud rate, bits, parity bit, software protocols, and timeout conditions; preparing data for the sent/received data buffer; processing sending/receiving labels.

A.3 CANopen Communication Protocol

The function block reads and writes SDOs/PDOs, through which CANopen communication is enabled. Assign communication variables (object dictionary data in the protocol) to corresponding input parameters of the function block, and trigger execution conditions to access slave data. AM600 protocols support the CANopen master only.

A.4 CANlink Communication Protocol

CANlink communication can be enabled through table configuration. Preset communication variables, communication frequency, and trigger conditions by completing a table. When a device serves as a CANlink network master, it can be connected to various Inovance remote expansion modules, MD380/500 AC drives, IS620 servos, and other slaves. In addition, it can be connected to other devices as a CANlink network slave.

CANlink3.0 communication protocol provides the following communication frames.

■ Communication frames triggered in timed or conditional mode are used to exchange communication data between ordinary slaves.

■ Synchronous trigger is used to control multiple highly real-time devices that can be controlled synchronously, for example, synchronous position control of multiple servos.

■ Heartbeat frames are used to monitor the communication status of each CANlink network slave so that the system can promptly respond to control system errors to avoid further losses.

A.5 EtherNET Port and Communication Protocol

The EtherNET port can be used to download PLC user programs, monitor and debug the system (like the Mini-USB port), and can also be used for Ethernet communication, including TCP/IP Modbus communication and free communication. You can configure communication parameters and address registers in the programming software through Modbus protocols, and access register values in user programs to exchange data with remote Modbus devices. If free communication protocols are used, data can be exchanged only by operating the socket function block.

A.6 EtherCAT Port and Communication Protocol

The EtherCAT port is used for communication (linear topology, full-duplex mode, Baud rate: 1 Mbit/s) through standard EtherCAT protocol. The maximum communication distance between slave nodes is 100 m. The master supports synchronization events and DC mode. The maximum task jitter is 120 us (typical value).

A.7 High-speed I/O Interface

The high-speed I/O interface has high-speed pulse control and high-speed pulse counting functions.

■ The high-speed pulse control function is used to control pulse servo drives and stepper drives.

■ The high-speed pulse counting function is used to collect A/B-phase, single-phase, CW/CCW pulse signal frequency and counting.

A.8 Mini-SD Card Slot

The Mini-SD card slot is mainly used to upgrade the underlying PLC firmware. This card slot is unavailable to users.

A.9 Local Bus Expansion Interface

A PLC can be directly connected to the I/O module through a local bus expansion interface. The PLC updates addresses of I/O module data mapped to the PLC based on the internal bus cycle.

You can access the mapped addresses to operate the I/O module.

A.10 PROFIBUS DP Port

The PROFIBUS DP port and CAN port are integrated on one DB9 hardware port. Currently, the DP function is used only for AM610 series.

Appendix B Soft Element

Soft elements are global variables predefined on the AM600 programming system, which can be used directly. As direct variables, soft elements are mapped to the M area (%M) and are retentive at power failure (**RETAIN**). The AM600 programming system includes SD soft elements and SM soft elements. SD soft elements are INT direct global variables. SM soft elements are BOOL direct global variables.

The M area (%M) has a capacity of 512 KB, the first 480 KB of which is for users and the last 32 KB is for the system. Do not use the address directly. The first 30,000 bytes of the 32 KB space are used for SD and SM soft elements to implement special functions, such as CANlink, CANopen, high-speed I/O instructions, and Modbus, as detailed in the following table. You can access the soft elements.

SD Range	Function	SM Range	Function
0 to 7999	Register elements for users: 0 to 7000 for CANlink (CANlink configuration, compatible with small-sized PLCs)	0 to 7999	Bit elements for users: 0 to 3071 and 8000 to 8511 for CANlink (CANlink configuration, compatible with small-sized PLCs) 0 to 7999 indicate Modbus/Modbus TCP-triggered variables, which are enabled by slaves.
8000 to 8999	Register elements for the system: CANlink/CANopen	8000 to 8999	Bit elements for the system: CANlink/CANopen
9000 to 9999	Register elements for the system: currently used only for high-speed I/O	9000 to 9999	Bit elements for the system: currently used only for high-speed I/O

NOTE
- ◆ The system is automatically reset after Modbus-triggered variables are set.
- ◆ System soft elements can be read and cannot be written. Otherwise, a system error may occur.

For details about how to use soft elements, see descriptions of CANlink soft elements, Modbus soft elements, and Modbus TCP soft elements.

Appendix C Cheat Sheet of Basic Instructions

Type	Description	Name	Category
Arithmetic operation instruction	Addition	ADD	Function
	Multiplication	MUL	Function
	Subtraction	SUB	Function
	Division	DIV	Function
	Remainder	MOD	Function
Valuation instruction	Valuation	MOV	Function
Logic instruction	AND operation	AND	Function
	OR operation	OR	Function
	XOR operation	XOR	Function
	NOT operation	NOT	Function
Shift instruction	Left shift	SHL	Function
	Right shift	SHR	Function
	Rotation left	ROL	Function
	Rotation right	ROR	Function
Selection instruction	Either-or selection	SEL	Function
	Maximum	MAX	Function
	Minimum	MIN	Function
	Limit	LIMIT	Function
	One-from-multiple selection	MUX	Function
Comparison instruction	Greater than	GT	Function
	Less than	LT	Function
	Greater than or equal to	GE	Function
	Less than or equal to	LE	Function
	Equal to	EQ	Function
	Not equal to	NE	Function
Basic mathematical operation instruction	Absolute	ABS	Function
	Square root	SQRT	Function
	Natural logarithm	LN	Function
	Common logarithm	LOG	Function
	Exponent	EXP	Function
	Sine	SIN	Function
	Cosine	COS	Function
	Tangent	TAN	Function
	Arcsine	ASIN	Function
	Arc cosine	ACOS	Function
	Arc tangent	ATAN	Function
	Exponential	EXPT	Function

Type	Description	Name	Category
Auxiliary mathematical operation instruction (Util library)	Differential	DERIVATIVE	Function block
	Integral	INTEGRAL	Function block
	Integral statistics	STATISTICS_INT	Function block
	Real statistics	STATISTICS_REAL	Function block
	Variance	VARIANCE	Function block
Type conversion instruction	Boolean conversion	BOOL_TO_<TYPE>	Function
	Byte conversion	BYTE_TO_<TYPE>	Function
	Date conversion	DATE_TO_<TYPE>	Function
	Long integer conversion	DINT_TO_<TYPE>	Function
	Date-time conversion	DT_TO_<TYPE>	Function
	Double-word conversion	DWORD_TO_<TYPE>	Function
	Integer conversion	INT_TO_<TYPE>	Function
	Word conversion	WORD_TO_<TYPE>	Function
	Real number conversion	REAL_TO_<TYPE>	Function
	Short integer conversion	SINT_TO_<TYPE>	Function
	Character conversion	STRING_TO_<TYPE>	Function
	Clock conversion	TIME_TO_<TYPE>	Function
	Time conversion	TOD_TO_<TYPE>	Function
	Long unsigned integer conversion	UDINT_TO_<TYPE>	Function
Address operation instruction	Address	ADR	Function
	Address content	^	Function
	Bit address	BITADR	Function
	Index	INDEXOF	Function
	Data size	SIZEOF	Function
Calling instruction	Call	CAL	Function
Initialization instruction	Initialize	INI	Function
String processing instruction (standard library)	String length	LEN	Function
	Left string	LEFT	Function
	Right string	RIGHT	Function
	Middle string	MID	Function
	String concatenation	CONCAT	Function
	String insertion	INSERT	Function
	Character deletion	DELETE	Function
	String replacement	REPLACE	Function
	String finding	FIND	Function
Bistable instruction (standard library)	Setting priority bistable trigger	SR	Function block
	Reset priority bistable trigger	RS	Function block
Trigger instruction (standard library)	Rising edge trigger detection	R_TRIG	Function block
	Falling edge trigger detection	F_TRIG	Function block
Counter (standard library)	Incremental counter	CTU	Function block
	Decremental counter	CTD	Function block
	Incremental/decremental counter	CTUD	Function block

Type	Description	Name	Category
Timer (standard library)	Ordinary timer	TP	Function block
	Power-on delay timer	TON	Function block
	Power-off delay timer	TOF	Function block
	RTC	RTC	Function block
BCD conversion instruction (Util library)	BCD-to-integer conversion	BCD_TO_INT	Function
	Integer-to-BCD conversion	INT_TO_BCD	Function
Bit/byte operation instruction (Util library)	Bit extraction	EXTRACT	Function
	Bit packing	PACK	Function
	Bit unpacking	UNPACK	Function block
	Bit valuation	PUTBIT	Function
Counter instruction (Util library)	PD counter	PD	Function block
	PID counter	PID	Function block
	PID counter	PID_FIXCYCLE	Function block
Signal generator instruction (Util library)	Pulse signal generator	BLINK	Function block
	Cyclic signal generator	GEN	Function block
Robot operation instruction (Util library)	Characteristic curve	CHARCURVE	Function block
	Integral speed limit	RAMP_INT	Function block
	Real speed limit	RAMP_REAL	Function block
Analog parameter processing instruction (Util library)	Hysteresis	HYSTERESIS	Function block
	Upper/lower limit alarm	LIMITALARM	Function block

Appendix D Guide to PLC Programming Software Upgrade

Version

1) The programming software InoProShop V1.1.0 and earlier versions are incompatible with the latest version in terms of persistent variable, hard disk partition, high-speed I/O function, and EtherCAT bus I/O module. It is recommended that you upgrade the software to the latest version. In addition, versions (earlier than V1.3.2) not mentioned in this section are not recommended. Contact local vendors if necessary.

2) Slave files, for example, EtherCAT description file (.xml), CANopen description file (.eds), and PROFIBUS DP description file (.gds), must match the slave firmware version. If you have any questions, contact local vendors. Slaves not installed for V1.3.2 by default can be supported by installing corresponding device files.

3) For details about how to use AM400/AM600/AC800 series, see hardware manuals or contact vendors.

Upgrade Method

■ Application software installation

A Windows 7 or Windows 10 operating system is required, and the memory must not be less than 4 GB. It is recommended that you use a 64-bit instead of a 32-bit Chinese-English operating system.

Install the software based on the wizard or set the installation path during installation as required. The default installation path is C:\Inovance Control\InoProShop.

<table>
<tr><td>NOTE</td><td>Do not install the software in the same folder with other versions.</td></tr>
</table>

■ User project

When you open a project in an earlier version, the Project Version window is displayed. If you do not want to update the project, select **Not update** to edit or use it directly. However, the LD must be updated.

The Project Version window can be displayed automatically when you open a previous project. Alternatively, you can choose **Project** > **Project Version** to display the window.

You can update all or some projects.

1) To update all projects, on the Project Version window, select **Update all projects to the latest version,** and then click **OK**.

2) To update some projects, on the Project Version window, select tabs corresponding to the projects to be updated, and then click **OK**.
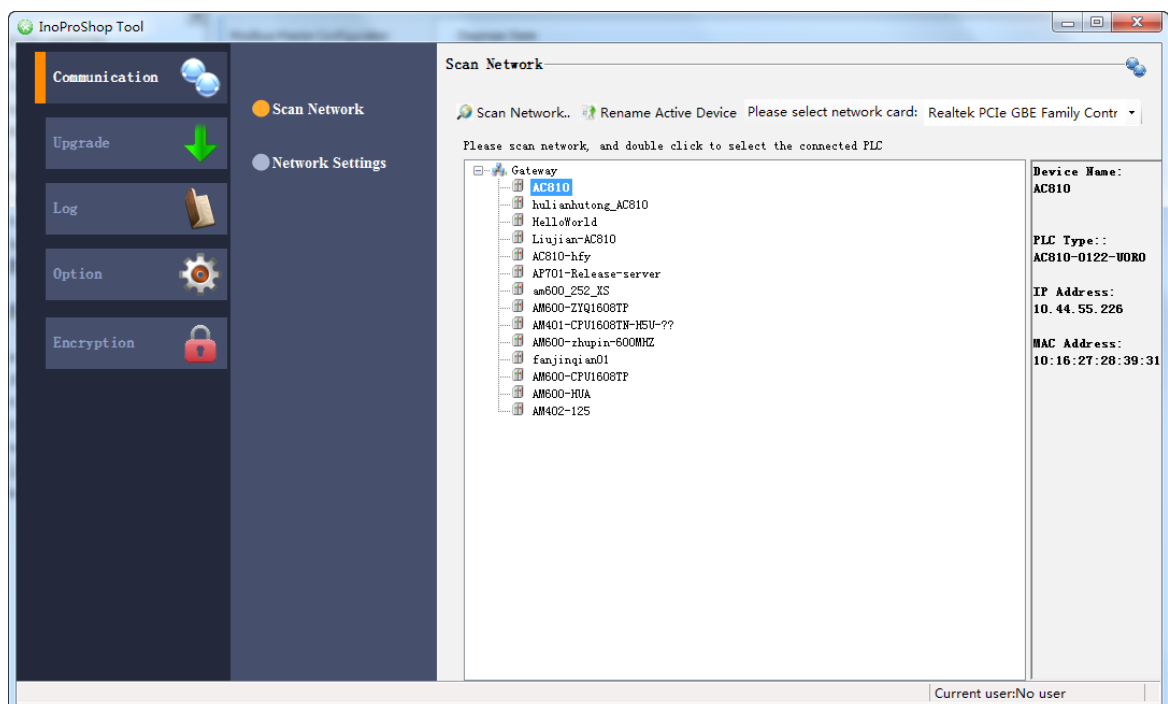
<table>
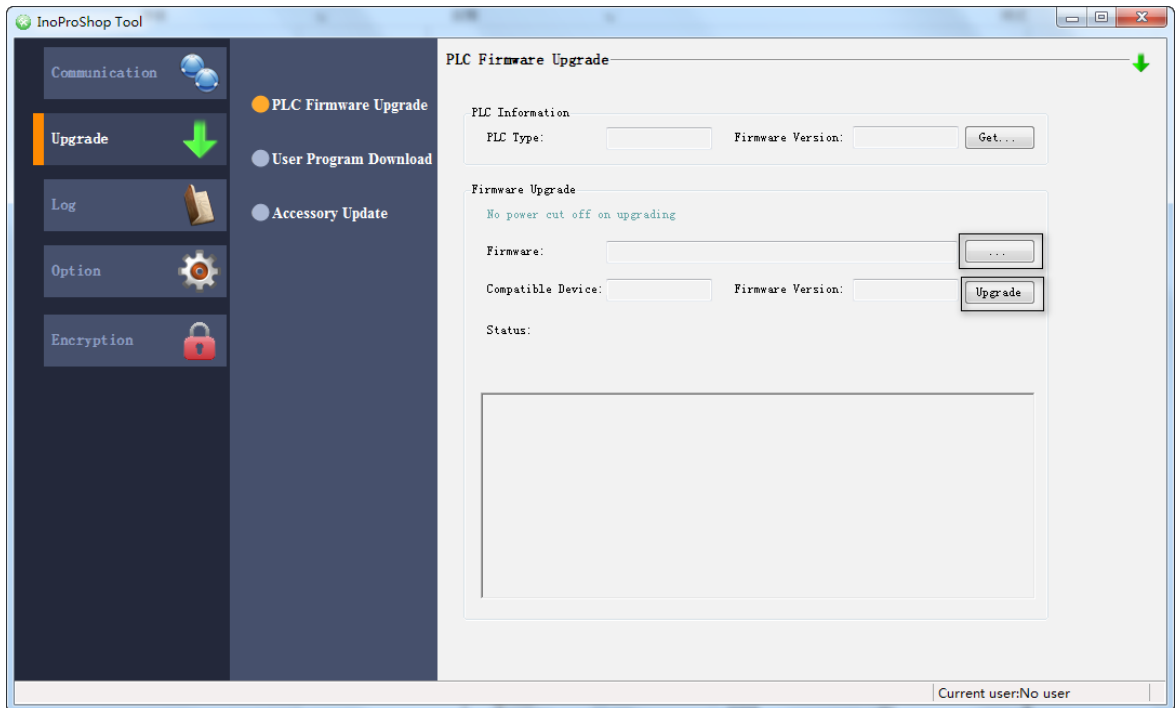<tr><td>NOTE</td><td>The LD must be updated.</td></tr>
</table>

■ Online firmware upgrade

PLC (CPU module) upgrade

Step 1: Choose **Tool** > **InoProshop Tool** > **Scan Network**, and select a device.
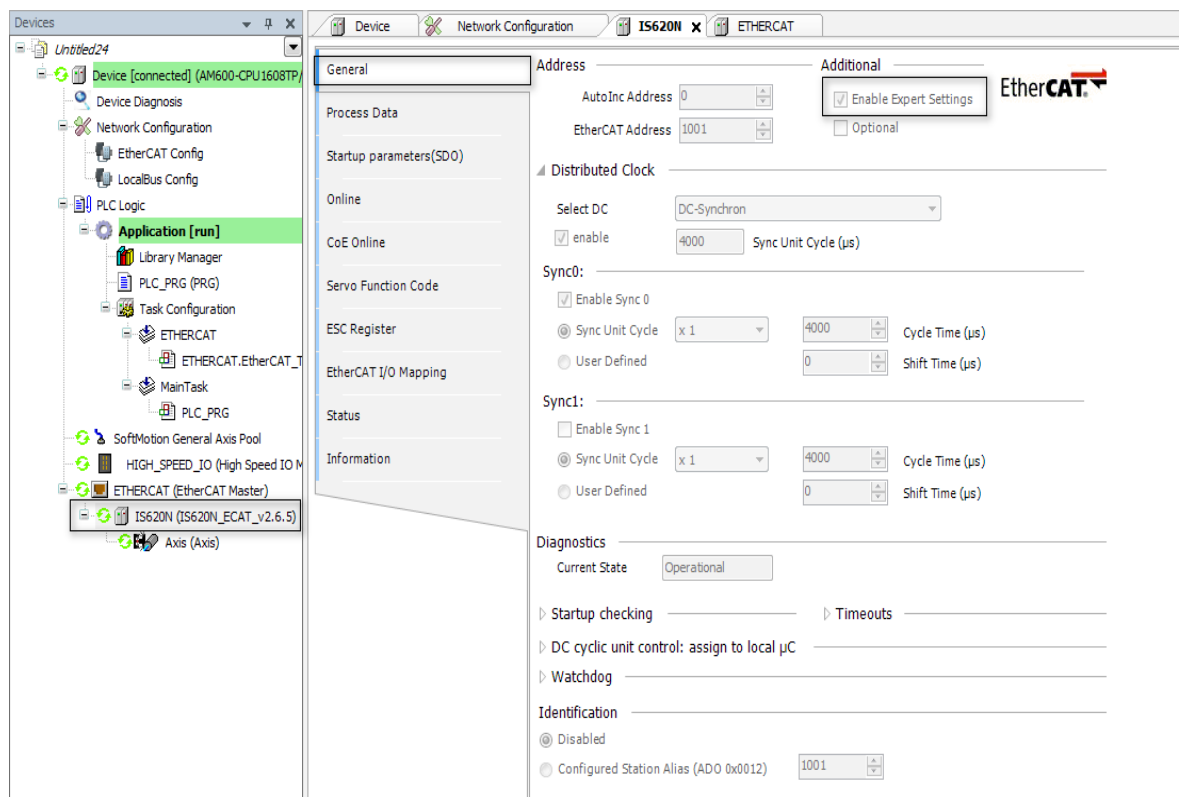
Step 2: Choose **Upgrade** > **Select Firmware** > **Upgrade**.

■ EtherCAT module upgrade

Step 1: Choose **Device** > **General**, select **Enable Expert Settings**, and then choose **Download** > **Run**.

Step 2: Choose **Device** > **Online** > **Bootstrap**. Then choose **FoE** > **Download**. On the dialog box displayed, find and select the corresponding firmware file with the extension **.bin** to start upgrade.

- **Library upgrade**

    See the section "How to add a compiled library to the project" of FAQs.
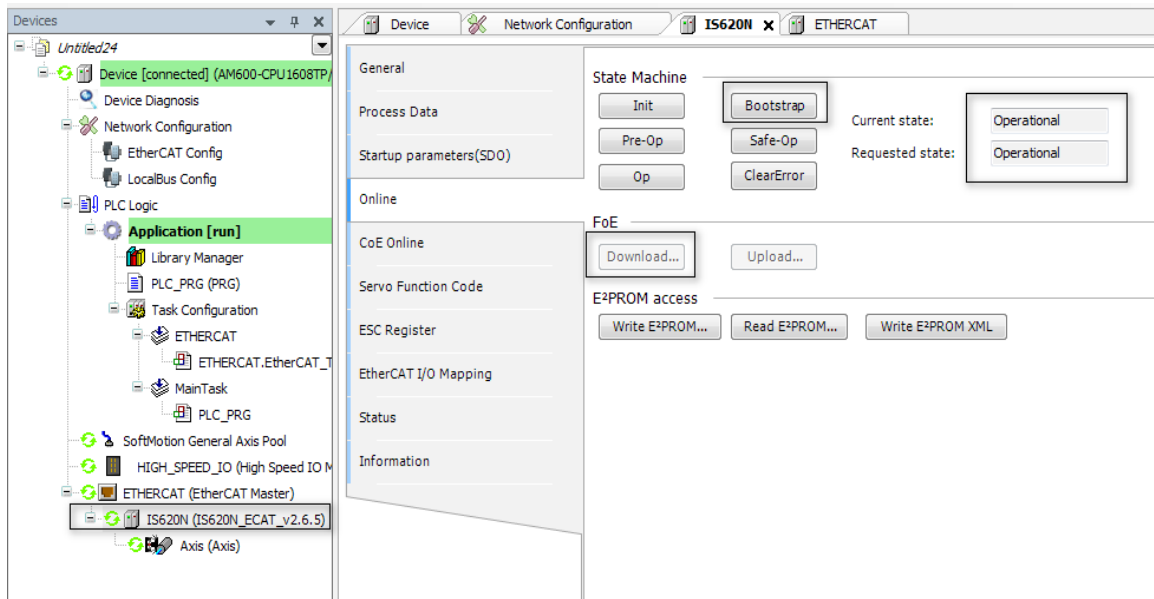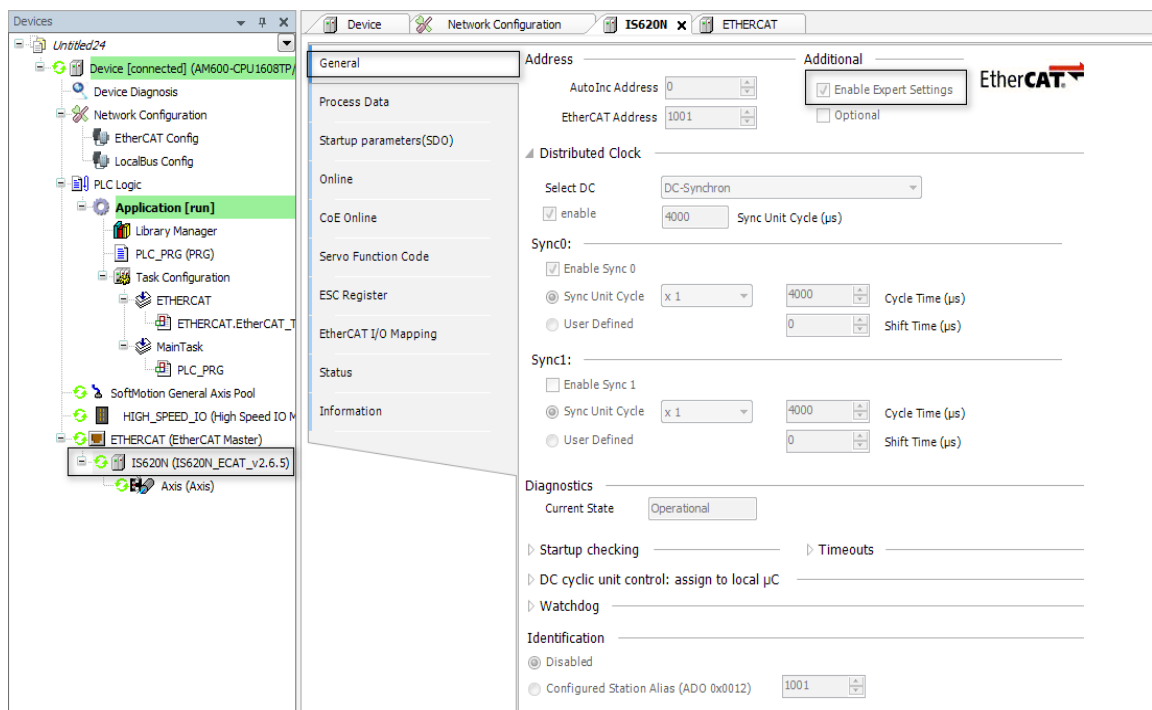
- **EtherCAT device file upgrade**

    Step 1: Choose **Device** > **General**, select **Enable Expert Settings**, and then choose **Download** > **Run**.

Step 2: Choose **Device** > **Online** > **Write EEPROMXML**. On the dialog box displayed, find and select the corresponding XML file to start upgrade.
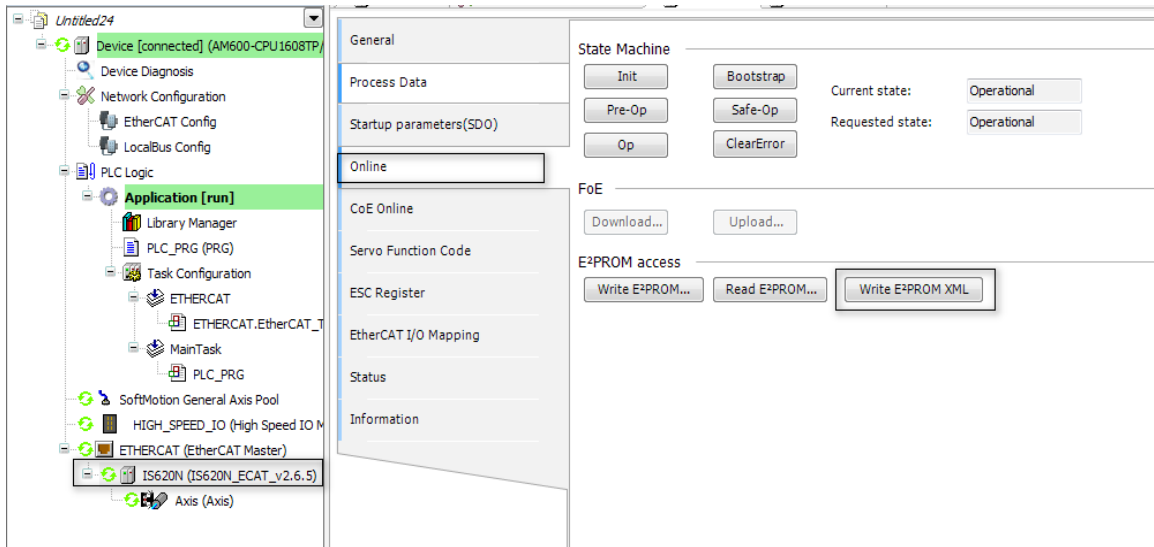
FAQs

1) How to check the version

   Choose **Device** > **Upgrade** > **Get PLC Information**.
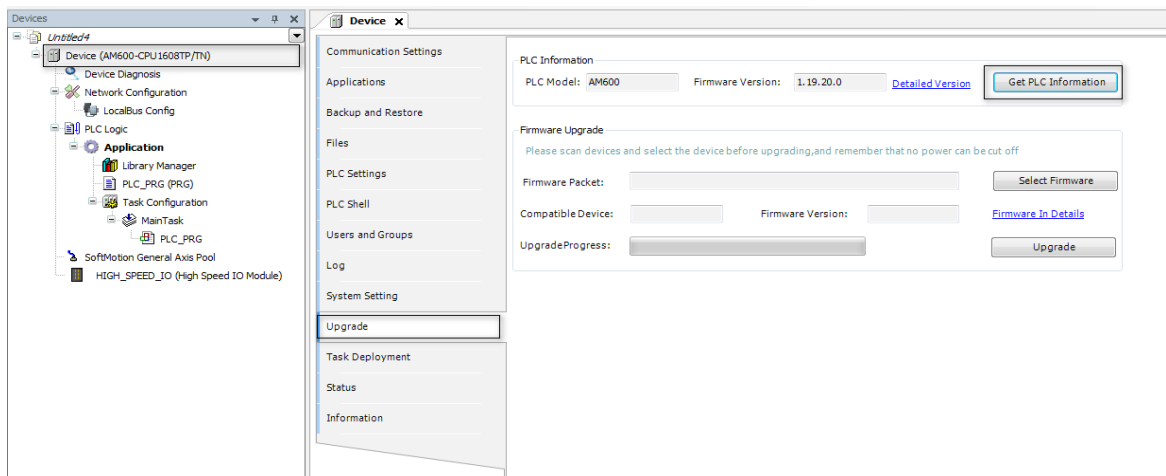
2) Target system not matching the connected device
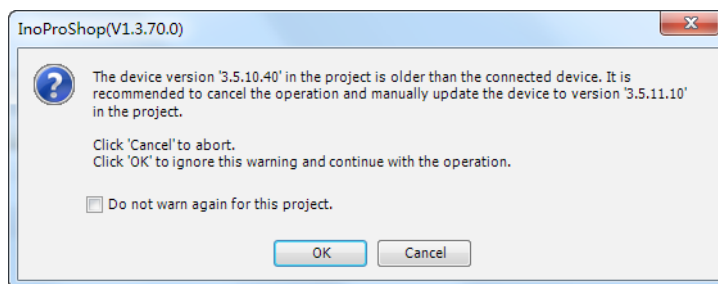
Figure D-1 Target system not matching the connected device

Cause: The PLC for InoProShop is in 3.5.11.10 version, while the actual PLC version is V3.5.10.20. The InoProShop device version cannot be later than the actual version.

Solution 1: Upgrade the PLC firmware to match the device version (3.5.11.10).

Step 1

Right-click **Device**, and click **Update Device**. On the window displayed, select **Display all versions** to find the corresponding version. If no matching version is found from the device list, you can select a version carrying the same first three numbers.

As shown in the following figure, the version "3.5.10.20" does not exist on the list. In this case, you can select "3.5.10.40" (carrying the same first three numbers) and update the device.
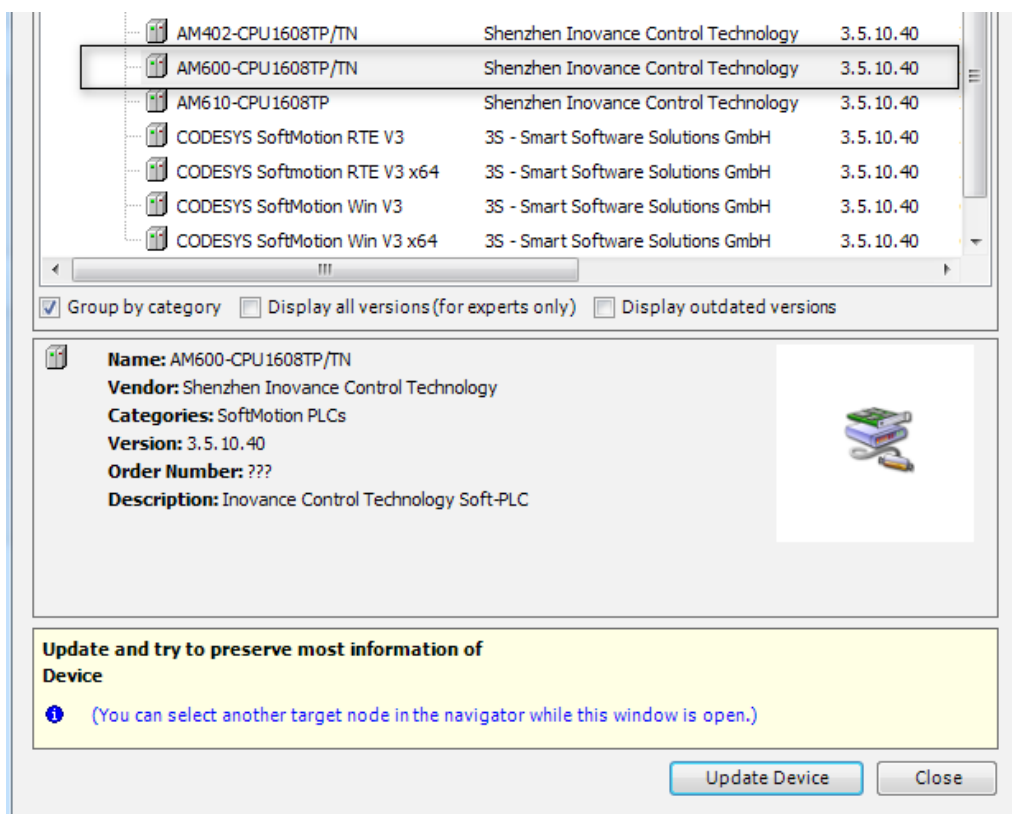
Figure D-2 Updating the PLC

Step 2

Rescan and select the corresponding device. No error is reported, shown as follows.
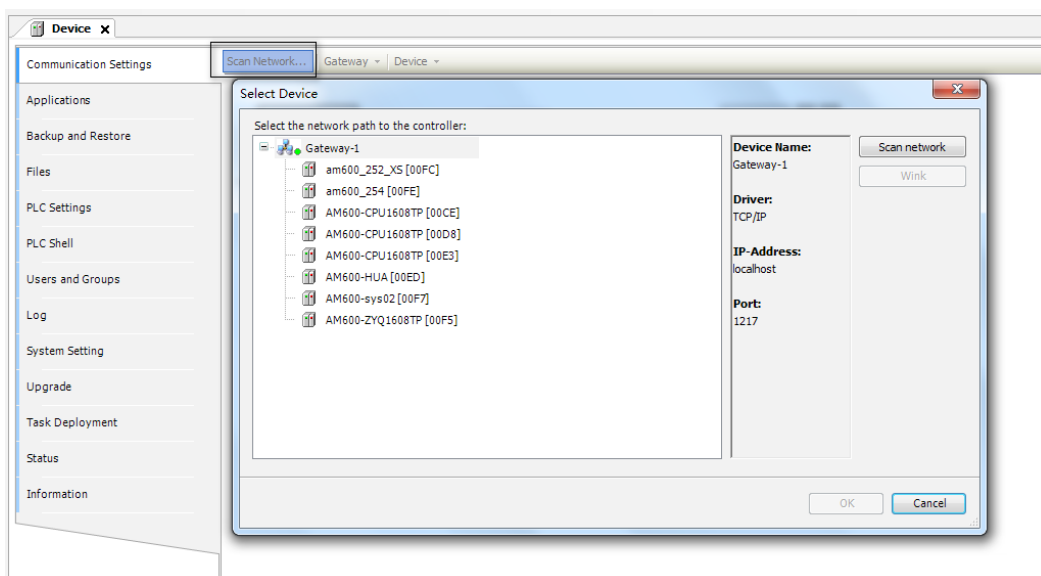
Figure D-3 Connecting the PLC

Step 3

Choose **Device** > **Upgrade** to upgrade the firmware on the Firmware Upgrade page.
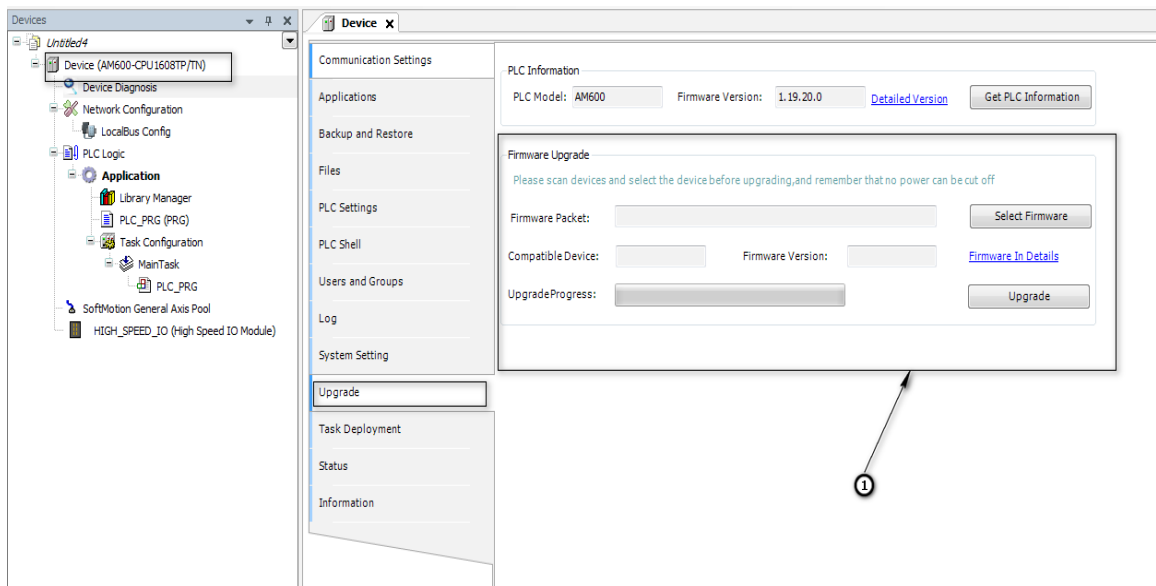
1 - Online Firmware Upgrade page

Figure D-4 Upgrading firmware online

Step 4

After the firmware is upgraded, upgrade the PLC to V3.5.11.10 as described in Step 1. Then you can use the latest version of PLC and its firmware.

Solution 2: Degrade the device to match the firmware version.

Take Step 1 of Solution 1. However, device files of PLCs in earlier versions can be used only with IEC libraries that match the PLC versions.

When an IEC library is added to the project, as the latest version of the IEC library is added by default, a compilation error may occur during program compilation because the library version does not match the PLC version. In this case, you can change the IEC library version manually.

3) Compilation error occurred while adding a library by using the latest version of programming software

Use V1.3.2 to open a project created by software in versions earlier than V1.3.0 (V1.2.0 as an example), choose **Add library** > **CmpHCBasic**, and use the MC_ResetDrive function block. A compilation error occurred.

1 - Added IEC library; 2 - Inconsistent version; 3 - Inconsistent version; 4 - Compilation error

Figure D-5 Compilation error occurred while adding a library

Cause: The version of specific libraries that the CmpHCBasic library depends on (SM3_Basic and IODrvEtherCAT libraries) does not match the version of specific libraries that project library management depends on. The IODrvEtherCAT version that CmpHCBasic (V1.8.0.0) depends on is V3.5.11.10, while the referenced version is V3.5.10.0. The SM3_Basic version CmpHCBasic (V1.8.0.0) depends on is V4.2.2.0, while the referenced version is V4.2.1.0.

Solution

Step 1: Double click **Library Manager** to display the Library Manager page. Select **CmpBasis** from the library list. The library version is V1.8.0.0.

Step 2: On the Library Manager interface, select **Properties**. In the window displayed, find the **Version** tab, select **1.6.0.0** (project created by V1.2.0) from the drop-down list, which is the IEC library version matching the PLC, and then click **OK**.
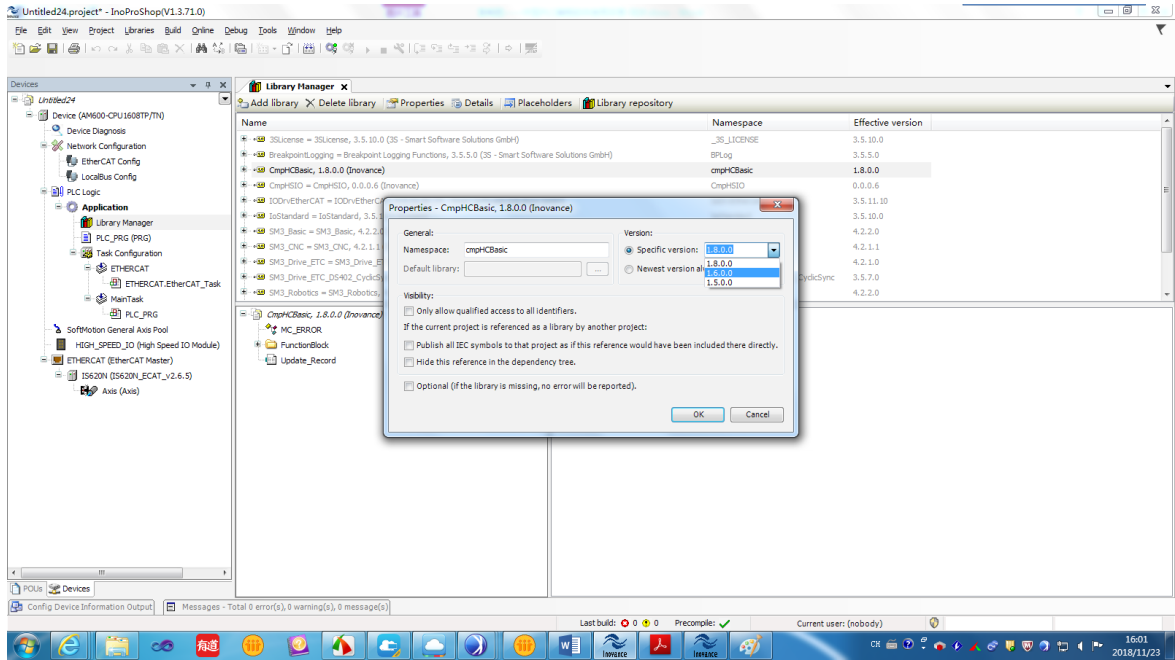
Figure D-6 Updating the IEC library manually

Step 3: Compile the project, as shown in the following figure.
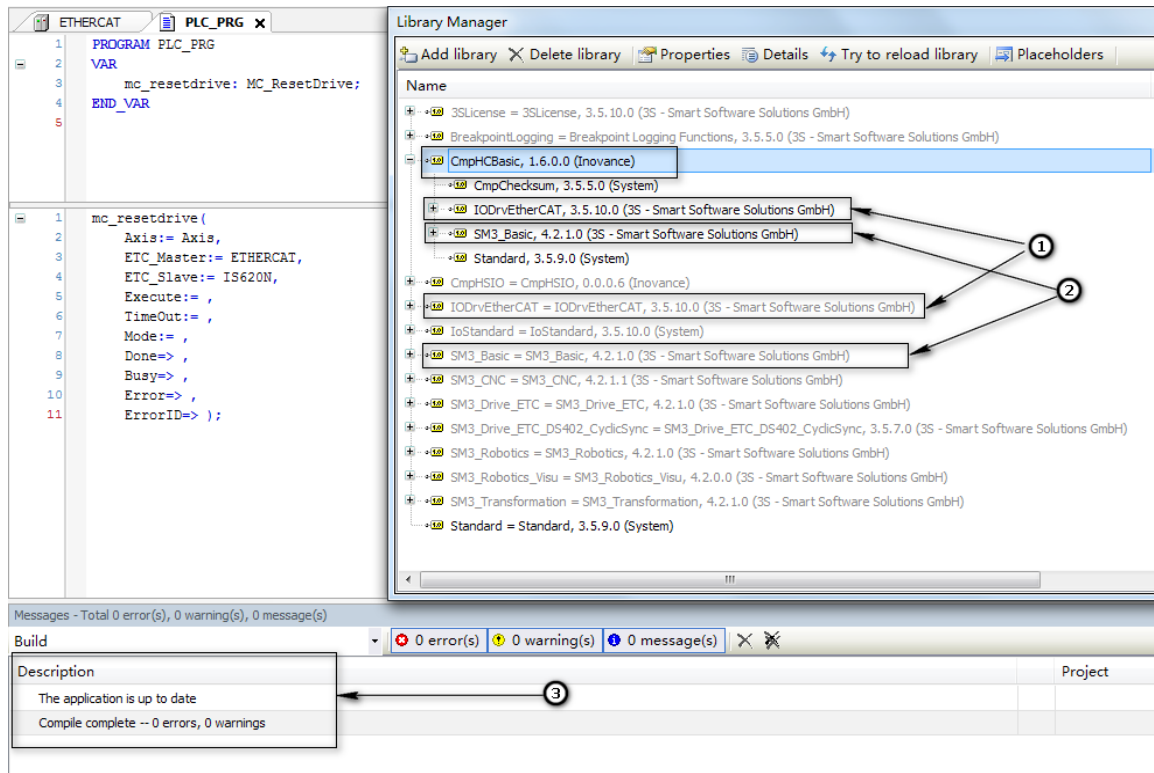
1&2 - Same version; 3 - Compilation succeeded

Figure D-7 Updated IEC library and compilation information

NOTE

The library is incompatible mainly because the version of the system library that the library depends on does not match the version of the system library included in the previous project. Common libraries include IODrvEtherCAT and SM3_Basic libraries.

4)   How to add a compiled library to the project

Example: CmpHCBasic. compiled-library (V1.11.0.0), software tool (V1.2.60)

This example is applicable to other versions of software tool.

Step 1: Install a compiled library.

Open a project and click **Library Manager**.
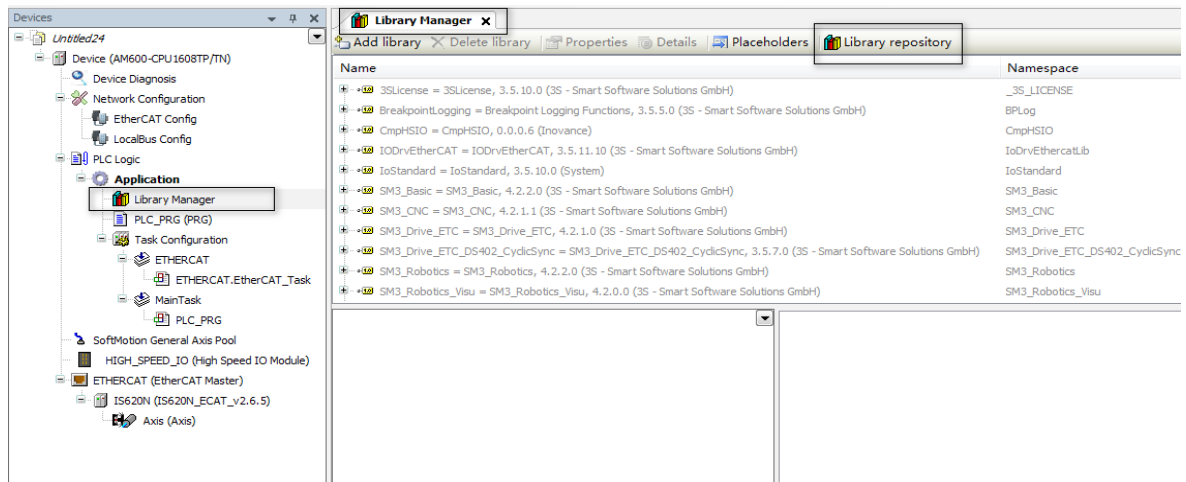
On the Library Manager interface, click **Library repository** > **Install**.
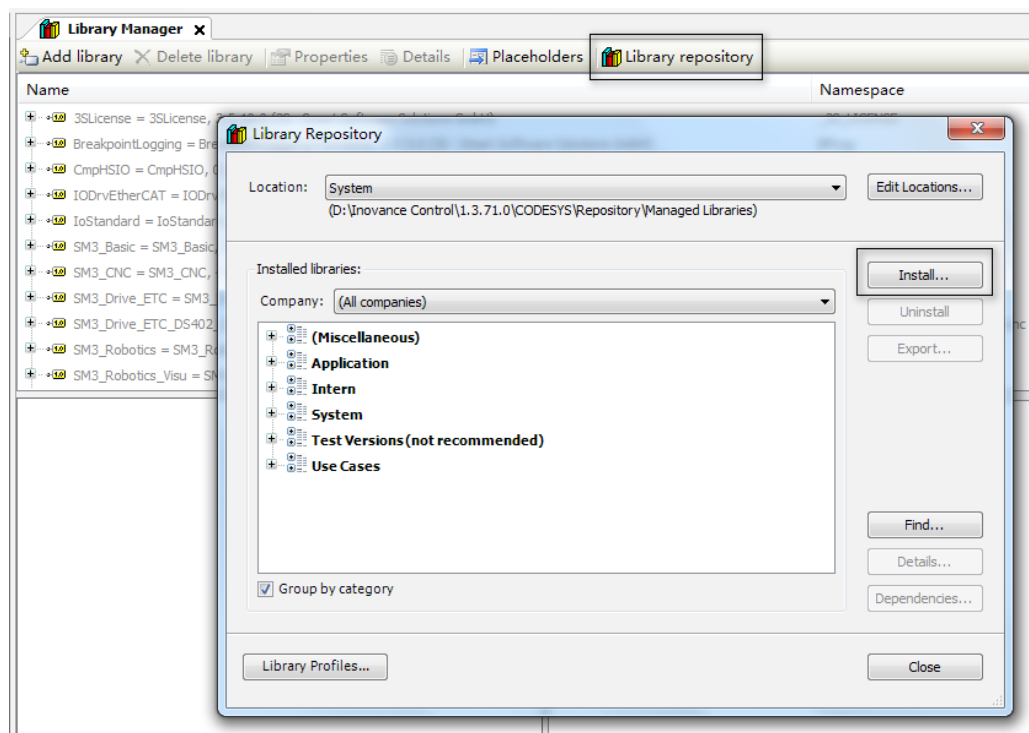
Find the compiled library (CmpHCBasic.compiled-library V1.11.0.0) and open it.

Step 2: Add the library to the project.

On the Library Manager interface, click **Add library** > **(Miscellaneous)** > **+**.
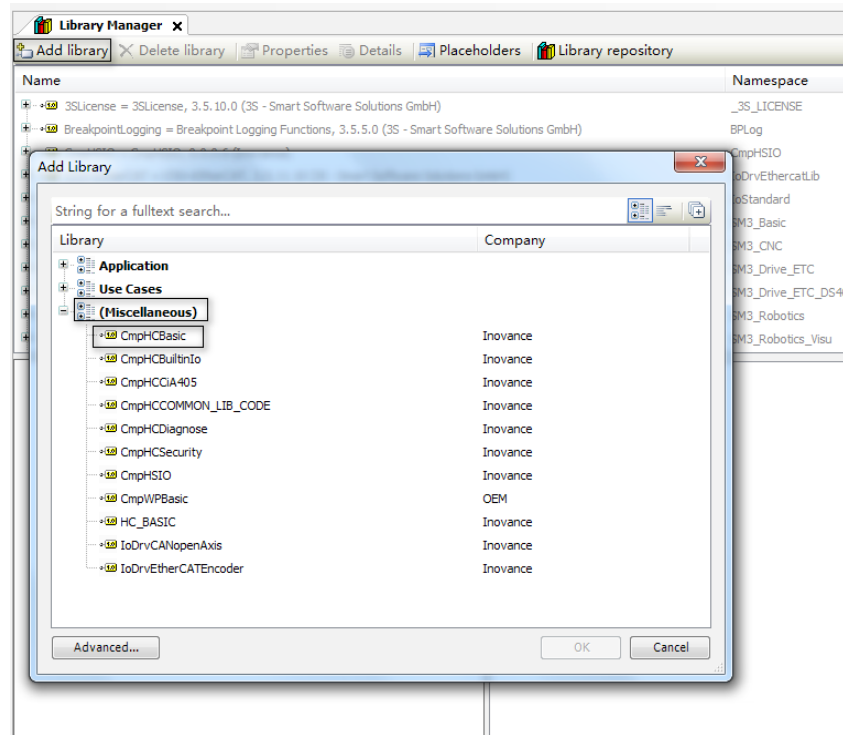
Select **CmpHCBasic** and click **OK**. By default, the latest version is added to the project.

Step 3: Select the library version manually.

On the Library Manager interface, select **CmpHCBasic**. Click **Properties**. The Properties window is displayed. You can select a version from the drop-down list of **Version** (confirm that the library version matches the software tool; otherwise, the system reports a compilation error when you use the library).
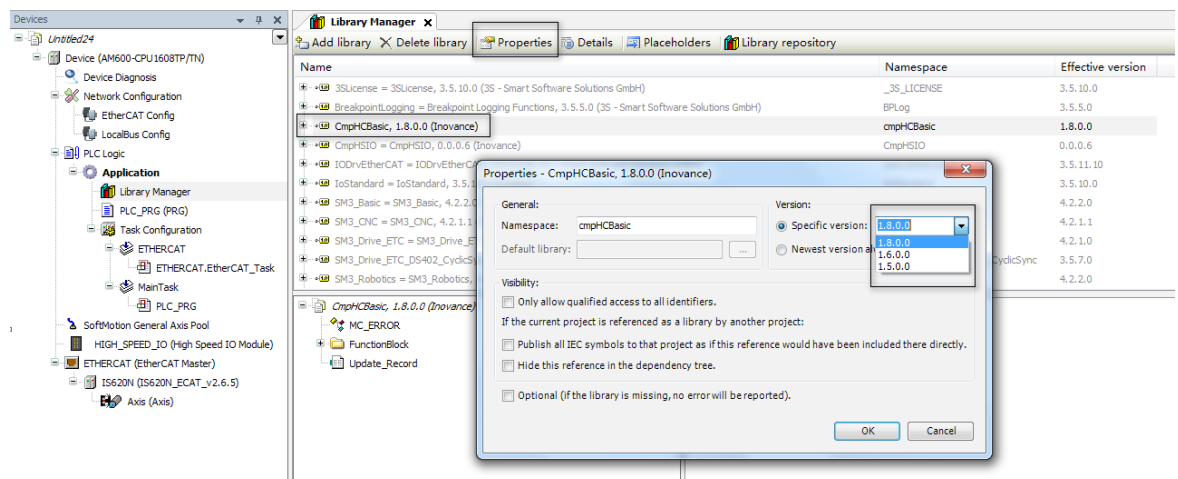
NOTE

◆ Note:

1) To add a library to the project or update a library, choose **Build** > **Clear All** first.

2) After the library is compiled, log in and download it again (a PLC error may be caused by online download).

Appendix E High-speed I/O Compatibility

User Guide

CmpHCBuiltinIo: Earlier high-speed I/O function block library

CmpHSIO: Latest high-speed I/O function block library

InProShop in V1.2.0 (temporary version: 1.1.60.0), AM600 firmware in V1.19.70.0 and FPGA in A624 version and later versions support the latest high-speed I/O function block library. The following figures show the software UIs for earlier and latest I/O devices respectively.

UI for high-speed I/O device in earlier version

UI for high-speed I/O device in the latest version

Note

1) Match the latest version of high-speed I/O device to the latest version of high-speed I/O library, and match the earlier version of high-speed I/O device to the earlier version of high-speed I/O library.

2) To use functions of the latest version of high-speed I/O device, you need to upgrade both PLC firmware and FPGA.

3) The latest and earlier high-speed I/O functions can be switched over to each other (incompatible with homing). If the latest high-speed I/O device is mixed with the earlier high-speed I/O device (the latest high-speed I/O project with an earlier PLC high-speed I/O device or an earlier I/O project with the latest PLC high-speed I/O device), a message is displayed reminding you to switch the PLC, as shown in the following figure. You need to switch the version of high-speed I/O device and restart the PLC.
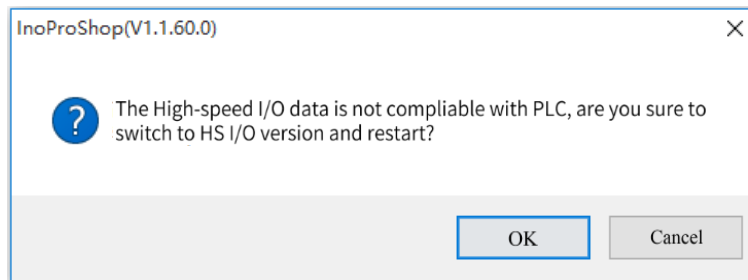
4) If you do not want to switch the PLC, you can switch the version of high-speed I/O device for the project. Display the Update Device interface, as shown in the following figure. The earlier high-speed I/O device is in 0.0.0.10 version, and the latest I/O device is in 0.0.0.20 version.
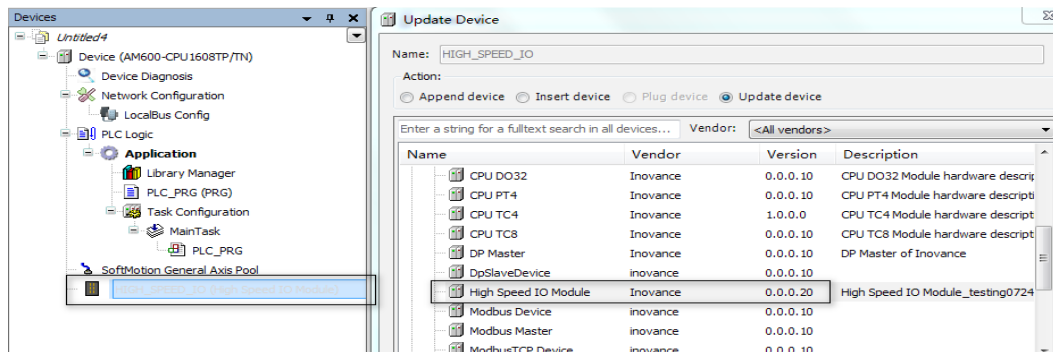
5) If you use earlier software tool (for example, V1.1.0 or V0.0.9.10) to download an earlier high-speed I/O project to the PLC with the latest version of firmware (later than V1.19.70.0), the error indicating version not matched is reported.

Solution: Install the latest version of software tool, and use it to open and download an earlier project (the high-speed I/O function not used) to the PLC with the latest version of firmware. In this case, no error is reported.

High-speed I/O Diagnosis

(1) High-speed I/O device in the latest version

Basic format

Library + function block + error code

<div align="center">3   3-bit</div>

■ Library: The default high-speed I/O is 0.

■ Function block number: Function blocks are numbered from 01. Function blocks are detailed in "6.2.4 List of Function Blocks".

■ Error code: The error code starts from 01. Error codes are detailed in the list of counter error codes. If the error code is less than 500, it indicates a serious error. If the error code is greater than 500, it indicates a function block error. In the example of 14506, **14** indicates HC_WriteParameter, and **506** indicates a parameter error. In the example of 31520, **31** indicates MC_WriteParameter_P, and **520** indicates a parameter error.

<div align="center">Table E-2 List of counter error codes</div>

Error Code	Definition	Description
001	ERR_COUNTERID_INVALID	The entered channel number is invalid. A valid number ranges from 0 to 7.
003	ERR_CNT_OVERFLOW	The counter overflow/underflow is incorrect.
004	ERR_COUNTER_NOT_CHOSEN	No high-speed function is selected. Select a high-speed in the programming software.
007	ERR_COUNTER_NOT_ENABLED	HC_Counter is not enabled.
101	ERR_WRITEINTERRUPTPARAMETER_UNVALIAD	The write interrupt parameter is invalid.
102	ERR_INTERRUPT_NOT_CHOSE	**Interrupt Input** is not selected in the programming software.
501	ERR_SETCOMPARE_IMREFRESHCYCLE_OVERFLOW	The comparison value ImRefreshCycle exceeds 30000. A valid comparison value ranges from 0 to 30000.
502	ERR_SETCOMPAREM_NUMBERS_OVERFLOW	The HC_SetCompareM number ranges from 1 to 100.
503	ERR_PREWR_VALUE_OVERFLOW	The preset value is out of range.
504	ERR_AVERAGE_PARA_UNVALIAD	The set average frequency and average rotational speed are invalid.
505	ERR_ROTATION_PULSES_UNIT_UNVALIAD	The set number of pulses per rotation is invalid.
506	ERR_WRITEBOOlPARAMETER_UNVALIAD	The set HC_WriteBoolParameter parameter is invalid.
507	ERR_READBOOLPARAMETER_UNVALIAD	The obtained HC_ReadBoolParameter parameter is invalid
508	ERR_MEASURE_WIDTH_OVERFLOW	The measured width is invalid.

Error Code	Definition	Description
509	ERR_SETCOMPAREM_IMREFRESHCYCLE_ OVERFLOW	The comparison value ImRefreshCycle exceeds 30000. A valid comparison value ranges from 0 to 30000.
510	ERR_PRESET_TRIGGERTYPE_OVERFLOW	The preset parameter is invalid.
511	ERR_WRITEPARAMETER_UNVALIAD	The set HC_WriteParameter parameter is invalid.
513	ERR_FUNC_COUNTERID_INVALID	The special function channel number is invalid. A valid number ranges from 0 to 3.
514	ERR_COUNTER_NOT_CHOSE_EXETERNAL_X	**External Trigger** is not selected in the programming software.
515	ERR_CNT_FORMAT_NOT_RING	The ring counting type is incorrect. Select a correct type in the programming software.
516	ERR_RING_DOWNVAL_BEYOND_UPVAL	The lower limit for ring counting is equal to or greater than the upper limit.
517	ERR_SAMPLE_VALUE_LESS	The sampling time is too short. A valid value ranges from 10 to 65535, in the unit of ms.
518	ERR_RING_VALUE_OVERFLOW	The ring counting is out of range.

Table E-3   List of high-speed axis error codes

Indicates the error code.	Definition	Description
001	ERR_NOT_POWER	MC_Power is not enabled.
002	ERR_UP_SOFTWARE_LIMIT	The current position is beyond the software stroke limit (Up).
003	ERR_DOWN_SOFTWARE_LIMIT	The current position is beyond the software stroke limit (Down).
004	ERR_AXIS_FUNC_UNUSED	The high-speed axis is not enabled. Enable the axis in the programming software.
005	ERR_INPUT_CHANNAL_NUM_INVALID	The axis number is invalid. A valid number ranges from 0 to 3.
006	ERR_DEST_POS_OVER_SOFT_UP_LIMIT	The target position is beyond the upper software limit.
007	ERR_DEST_POS_OVER_SOFT_DOWN_LIMIT	The target position is beyond the lower software limit.
010	ERR_POS_DECPOINT_OVERLOW	The deceleration point is invalid: In position mode, when the device is repositioned, the deceleration length is greater than the actual distance.
011	ERR_VEL_DECPOINT_OVERLOW	The deceleration point is invalid: When you switch from the speed mode to the position mode, the deceleration length is greater than the actual distance.
012	ERR_POS_PLSNUM_OVERLOW	The maximum PLSNUM positioning length 2147483647 is exceeded.
013	ERR_POS_DECPOINT2_OVERLOW	An error occurred while recomputing the deceleration point.
501	ERR_ACC_SET_OVERFLOW	The acceleration exceeds the maximum value set by MC_WriteParameter_P.
502	ERR_ACC_SET_LOW	The acceleration is below the minimum value set by MC_WriteParameter_P.
503	ERR_DEC_SET_OVERFLOW	The deceleration exceeds the maximum value set by MC_WriteParameter_P.
504	ERR_DEC_SET_LOW	The deceleration is below the minimum value set by MC_WriteParameter_P.
505	ERR_VEL_SET_OVERFLOW	The set speed is out of range. Set the speed in the programming software or through MC_WriteParameter_P.

Indicates the error code.	Definition	Description
506	ERR_VEL_SET_LOW	The set speed is too low.
508	ERR_VEL_LESS_THAN_STARTVEL	The speed is less than the startup offset speed. Set the startup offset speed in the programming software.
509	ERR_STARTVEL_SET_LOW	The starting speed is too small.
510	ERR_FBD_MOVEMODE_INVALIAD	The motion mode of the function block is invalid.
511	ERR_WASNT_STANDSTILL	The axis is not in Standstill state.
512	ERR_WASNT_DISABLED	The axis is not in Disabled state.
513	ERR_IN_ERRORSTOP	The axis is in ErrorStop state.
514	ERR_NOT_READY_FOR_MOTION	The axis is not ready to run.
515	ERR_INVLALID_VELOCITY_MODE	The speed mode is invalid.
516	ERR_INVLALID_POSTION_MODE	The position mode is invalid.
520	ERR_AXIS_WRITEPARAMETER_UNVALIAD	The MC_WriteParameter_P parameter is invalid.
521	ERR_AXIS_READPARAMETER_UNVALIAD	The MC_ReadParameter_P parameter is invalid.
522	ERR_HOME_MODE_UNVALIAD	The homing mode is invalid. Select a valid mode in the programming software.
523	ERR_AXIS_WRITEPARAMETER_HOME_MODE_UNVALIAD	The homing mode is invalid.

Errors can be classified into axis errors and function block errors.

Conditions for setting the axis to ErrorStop state:

1) Axis errors occur.

2) Function block errors occur when the axis is in DiscreteMotion, ContinuousMotion, or Homing state.

(2) High-speed I/O device in earlier version

High-speed I/O diagnosis information is displayed on the high-speed I/O self-diagnosis page. For descriptions of the self-diagnosis page, see the overview of the list of device self-diagnosis information.

High-speed I/O diagnosis information is obtained by obtaining high-speed I/O soft elements. High-speed I/O diagnosis results include channel errors, channel alarms, axis errors, axis alarms, and other faults. The diagnosis states and diagnosis codes of channel errors, channel alarms, axis errors, and axis alarms are indicated by soft elements. The diagnosis state indicates whether diagnosis information exists, and the diagnosis code indicates the error code. The following table shows soft elements, diagnosis codes, and diagnosis information corresponding to each type.

■ Channel error

   The following table lists the relationship among the channel number, error flag soft element, and error diagnosis code soft element.

Channel Number	Error Flag Soft Element	Error Diagnosis Code Soft Element
0	SM9030	SD9007
1	SM9080	SD9017
2	SM9130	SD9027
3	SM9180	SD9037
4	SM9230	SD9047
5	SM9380	SD9057
6	SM9330	SD9067
7	SM9380	SD9077

The following table lists the relationship between the diagnosis code and diagnosis information.

Diagnosis Code	Diagnosis Information
1001	Unmatching channel type
1002	Counter overflow
1003	Pulse width measurement overflow
1011	The lower limit for the ring counter above the upper limit
1012	Unmatching counter type
1013	Unused high-speed counting function
1014	Unmatching high-speed counter function
1015	Preset value out of range
1016	Invalid average parameter
1017	Invalid set number of pulses per rotation

■ Channel alarm

The following table lists the relationship among the channel number, alarm flag soft element, and alarm diagnosis code soft element.

Channel Number	Alarm Flag Soft Element	Alarm Diagnosis Code Soft Element
0	SM9031	SD9008
1	SM9081	SD9018
2	SM9131	SD9028
3	SM9181	SD9038
4	SM9231	SD9048
5	SM9381	SD9058
6	SM9331	SD9068
7	SM9381	SD9078

The following table lists the relationship between the diagnosis code and diagnosis information.

Diagnosis Code	Diagnosis Information
1501	Sampled value overflow

■ Axis error

The following table lists the relationship among the axis number, error flag soft element, and error diagnosis code soft element.

Axis Number	Error Flag Soft Element	Error Diagnosis Code Soft Element
0	SM9405	SD9105
1	SM9425	SD9125
2	SM9445	SD9145
3	SM9465	SD9165

The following table lists the relationship between the diagnosis code and diagnosis information.

Diagnosis Code	Diagnosis Information
2001	Hardware limit in the forward direction
2002	Hardware limit in the reverse direction
2003	Stop upon startup command ON
2004	Software limit in the forward direction
2005	Software limit in the reverse direction
2006	CPU module switched to Stop state during running
2007	Drive module ready OFF
2008	Zero signal ON
2009	Mechanical homing not executed
2010	Retry error
2011	ABS transmission timeout
2012	ABS transmission SUM
2013	Speed 0 error
2014	Acceleration/deceleration timeout
2015	Deceleration stop timeout
2016	Movement during speed/position switchover control out of range
2017	Speed/position switchover disabled
2018	Current value changed not in axis stop state
2019	Acceleration/deceleration time set to 0
2020	Axis not stopped upon startup
2021	Stop for axis command

■ Axis alarm

The following table lists the relationship among the axis number, alarm flag soft element, and alarm diagnosis code soft element.

Axis Number	Alarm Flag Soft Element	Alarm Diagnosis Code Soft Element
0	SM9406	SD9106
1	SM9426	SD9126
2	SM9446	SD9146
3	SM9466	SD9166

The following table lists the relationship between the diagnosis code and diagnosis information.

Diagnosis Code	Diagnosis Information
2501	Speed out of range
2502	Target position change disabled
2503	Speed change disabled

■ Other faults

Other faults indicate diagnosis of invalid input parameters of the high-speed I/O function block. The diagnosis data cannot be obtained through soft elements. You can check the data on the high-speed I/O self-diagnosis page and the diagnosis information list page. The following table lists diagnosis codes and diagnosis information.

Diagnosis Code	Diagnosis Information
1018	Invalid channel number for the high-speed input function block
1019	Invalid input parameter of the high-speed input function block
2022	Invalid channel number for the high-speed output function block
2023	Invalid input parameter of the high-speed output function block

Appendix F Diagnosis Code and Diagnosis Information

Each diagnosis code has a name, which matches the type name of the corresponding diagnosis programming interface. For details, see "7.5 Diagnosis Programming Interface".

CPU Diagnosis Code

Name	Diagnosis Code	Diagnosis Information
SDCardError	1	SD card error
FlashError	1	Flash error
SystemError	0 x 40	High-speed I/O interface board connection error
InterCommError	0x11	No I/O expansion module (inter-board communication error: read check failure)
	0x12	No I/O expansion module (inter-board communication error: write check failure)
	0x13	No I/O expansion module (inter-board communication error: ACK being high level)
	0x14	No I/O expansion module (inter-board communication error: ACK being low level)
	0x21	Actual number of I/O expansion modules below configured (inter-board communication error: read check failure)
	0x22	Actual number of I/O expansion modules below configured (inter-board communication error: write check failure)
	0x23	Actual number of I/O expansion modules below configured (inter-board communication error: ACK being high level)
	0x24	Actual number of I/O expansion modules below configured (inter-board communication error: ACK being low level)
	0x31	Actual number of I/O expansion modules above configured (inter-board communication error: read check failure)
	0x32	Actual number of I/O expansion modules above configured (inter-board communication error: write check failure)
	0x33	Actual number of I/O expansion modules above configured (inter-board communication error: ACK being high level)
	0x34	Actual number of I/O expansion modules above configured (inter-board communication error: ACK being low level)
	0x41	I/O expansion module type error (inter-board communication error: read check failure)
	0x42	I/O expansion module type error (inter-board communication error: write check failure)
	0x43	I/O expansion module type error (inter-board communication error: ACK being high level)
	0x44	I/O expansion module type error (inter-board communication error: ACK being low level)

Name	Diagnosis Code	Diagnosis Information
ConformenceError (Each bit indicates one module fault.)	1	I/O module corresponding to Slot 1 inconsistent with actual I/O module configuration
	2	I/O module corresponding to Slot 2 inconsistent with actual I/O module configuration
	4	I/O module corresponding to Slot 3 inconsistent with actual I/O module configuration
	8	I/O module corresponding to Slot 4 inconsistent with actual I/O module configuration
	16	I/O module corresponding to Slot 5 inconsistent with actual I/O module configuration
	32	I/O module corresponding to Slot 6 inconsistent with actual I/O module configuration
	64	I/O module corresponding to Slot 7 inconsistent with actual I/O module configuration
	128	I/O module corresponding to Slot 8 inconsistent with actual I/O module configuration
	256	I/O module corresponding to Slot 9 inconsistent with actual I/O module configuration
	512	I/O module corresponding to Slot 10 inconsistent with actual I/O module configuration
	1024	I/O module corresponding to Slot 11 inconsistent with actual I/O module configuration
	2048	I/O module corresponding to Slot 12 inconsistent with actual I/O module configuration
	4096	I/O module corresponding to Slot 13 inconsistent with actual I/O module configuration
	8192	I/O module corresponding to Slot 14 inconsistent with actual I/O module configuration
	16384	I/O module corresponding to Slot 15 inconsistent with actual I/O module configuration
	32768	I/O module corresponding to Slot 16 inconsistent with actual I/O module configuration
IOModulePosError (Each bit indicates one module fault. As fault information is displayed on the I/O module, the diagnosis information is not displayed but flagged.)	1	I/O module corresponding to Slot 1 faulty
	2	I/O module corresponding to Slot 2 faulty
	4	I/O module corresponding to Slot 3 faulty
	8	I/O module corresponding to Slot 4 faulty
	16	I/O module corresponding to Slot 5 faulty
	32	I/O module corresponding to Slot 6 faulty
	64	I/O module corresponding to Slot 7 faulty
	128	I/O module corresponding to Slot 8 faulty
	256	I/O module corresponding to Slot 9 faulty
	512	I/O module corresponding to Slot 10 faulty
	1024	I/O module corresponding to Slot 11 faulty
	2048	I/O module corresponding to Slot 12 faulty
	4096	I/O module corresponding to Slot 13 faulty
	8192	I/O module corresponding to Slot 14 faulty
	16384	I/O module corresponding to Slot 15 faulty
	32768	I/O module corresponding to Slot 16 faulty
FunctionErrorCode (Each bit indicates one bus fault, which is only flagged.)	0x01	DP bus faulty
	0x02	EtherCAT bus faulty
	0x04	CANopen bus faulty
	0x08	CANlink bus faulty
	0x10	Modbus TCP faulty
	0x20	Modbus serial port 0 faulty
	0x40	Modbus serial port 1 faulty
	0x80	High-speed I/O faulty
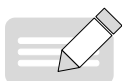
NOTE

As EtherCAT is implemented through CoDeSys, the PLC cannot obtain EtherCAT diagnosis information directly, and EtherCAT bus flags are invalid currently.

I/O Module Diagnosis Code

Name	Module Type	Diagnosis Code	Diagnosis Information
BaseInfo	All	64	System shut down upon fault event
ModuleError (Each bit indicates one module fault.)	AI	2	No external load voltage
		4	Analog chip connection error
	AO	2	No external load voltage
		4	Analog chip connection error
		8	Analog chip overheated
ChannelError[i] (Each array element indicates one channel diagnosis code, and each bit indicates one fault.)	AI	2	Overflow
		4	Underflow
		8	Beyond the upper limit
		16	Beyond the lower limit
		32	Disconnected
	AO	2	Overflow
		4	Underflow
		8	Current disconnected
		16	Voltage short-circuited
		32	Digital to Analog Converter (DAC) channel hardware fault

CANopen Diagnosis Code

Name	Diagnosis Code	Diagnosis Information
SlaveError	1	Slave disconnected
InterCommError	0x11	No I/O expansion module (inter-board communication error: read check failure)
	0x12	No I/O expansion module (inter-board communication error: write check failure)
	0x13	No I/O expansion module (inter-board communication error: ACK being high level)
	0x14	No I/O expansion module (inter-board communication error: ACK being low level)
	0x21	Actual number of I/O expansion modules below configured (inter-board communication error: read check failure)
	0x22	Actual number of I/O expansion modules below configured (inter-board communication error: write check failure)
	0x23	Actual number of I/O expansion modules below configured (inter-board communication error: ACK being high level)
	0x24	Actual number of I/O expansion modules below configured (inter-board communication error: ACK being low level)
	0x31	Actual number of I/O expansion modules above configured (inter-board communication error: read check failure)
	0x32	Actual number of I/O expansion modules above configured (inter-board communication error: write check failure)
	0x33	Actual number of I/O expansion modules above configured (inter-board communication error: ACK being high level)
	0x34	Actual number of I/O expansion modules above configured (inter-board communication error: ACK being low level)
	0x41	I/O expansion module type error (inter-board communication error: read check failure)
	0x42	I/O expansion module type error (inter-board communication error: write check failure)
	0x43	I/O expansion module type error (inter-board communication error: ACK being high level)
	0x44	I/O expansion module type error (inter-board communication error: ACK being low level)

Name	Diagnosis Code	Diagnosis Information
ConformenceError (Each bit indicates one module fault.)	1	I/O module corresponding to Slot 1 inconsistent with actual I/O module configuration
	2	I/O module corresponding to Slot 2 inconsistent with actual I/O module configuration
	4	I/O module corresponding to Slot 3 inconsistent with actual I/O module configuration
	8	I/O module corresponding to Slot 4 inconsistent with actual I/O module configuration
	16	I/O module corresponding to Slot 5 inconsistent with actual I/O module configuration
	32	I/O module corresponding to Slot 6 inconsistent with actual I/O module configuration
	64	I/O module corresponding to Slot 7 inconsistent with actual I/O module configuration
	128	I/O module corresponding to Slot 8 inconsistent with actual I/O module configuration
	256	I/O module corresponding to Slot 9 inconsistent with actual I/O module configuration
	512	I/O module corresponding to Slot 10 inconsistent with actual I/O module configuration
	1024	I/O module corresponding to Slot 11 inconsistent with actual I/O module configuration
	2048	I/O module corresponding to Slot 12 inconsistent with actual I/O module configuration
	4096	I/O module corresponding to Slot 13 inconsistent with actual I/O module configuration
	8192	I/O module corresponding to Slot 14 inconsistent with actual I/O module configuration
	16384	I/O module corresponding to Slot 15 inconsistent with actual I/O module configuration
	32768	I/O module corresponding to Slot 16 inconsistent with actual I/O module configuration
IOModulePosError (Each bit indicates one module fault. As fault information is displayed on the I/O module, the diagnosis information is not displayed but flagged.)	1	I/O module corresponding to Slot 1 faulty
	2	I/O module corresponding to Slot 2 faulty
	4	I/O module corresponding to Slot 3 faulty
	8	I/O module corresponding to Slot 4 faulty
	16	I/O module corresponding to Slot 5 faulty
	32	I/O module corresponding to Slot 6 faulty
	64	I/O module corresponding to Slot 7 faulty
	128	I/O module corresponding to Slot 8 faulty
	256	I/O module corresponding to Slot 9 faulty
	512	I/O module corresponding to Slot 10 faulty
	1024	I/O module corresponding to Slot 11 faulty
	2048	I/O module corresponding to Slot 12 faulty
	4096	I/O module corresponding to Slot 13 faulty
	8192	I/O module corresponding to Slot 14 faulty
	16384	I/O module corresponding to Slot 15 faulty
	32768	I/O module corresponding to Slot 16 faulty

DP Diagnosis Code

Name	Diagnosis Code	Diagnosis Information
ExtDiagData[0] (Each bit indicates one fault.)	0x02	Unready to exchange data
	0x04	Incorrect configuration
	0x08	Expanded diagnosis information existing on the slave
	0x10	The requested function not supported by the slave
	0x20	Invalid slave response
	0x40	Incorrect parameter
	0x80	Locked by different masters
ExtDiagData[1] (Each bit indicates one fault.)	0x01	Resetting the parameter
	0x02	A static diagnosis
	0x08	Activated Watchdog monitoring
	0x10	Processing slave data in latch mode
	0x20	Processing slave data in synchronous mode
	0x80	The slave not activated by the master
ExtDiagData[2] (Each bit indicates one fault.)	0x80	Diagnosis data overflow
ExtDiagData[3] (master address)		
ExtDiagData[4] and ExtDiagData[5] (slave ID)		

> **NOTE** The first 6 bytes correspond to the basic diagnosis, and the following diagnosis data bits correspond to the expanded diagnosis. For details, see DP Diagnosis.

CANlink Diagnosis Code

Name	Diagnosis Code	Diagnosis Information
CmdFrameError (diagnosis code: remainder when divided by 100)	1	Illegal command code
	2	Abnormal command code address
	3	Value out of allowable range
	4	Invalid command code operation
	5	Invalid command code length
	6	Command code timeout
CfgFrameError (diagnosis code: remainder when divided by 100)	1	Incorrect configuration code
	2	Incorrect configuration index
	3	Incorrect configuration information
	5	Incorrect configuration data length
	6	Configuration frames failing to respond

Modbus Diagnosis Code

Name	Diagnosis Code	Diagnosis Information
DiagData	0x70	Incorrect Modbus slave address
	0x71	Incorrect data frame length, serial port 0 (COM0)
	0x72	Illegal data address
	0x73	CRC error
	0x74	Unsupported instruction code
DiagData	0x75	Receiving timeout
	0x76	Illegal data value
	0x77	Buffer overflow
	0x78	Frame error
	0x79	Serial protocol error
	0x7C	Incorrect address
	0x7D	No data received
	0x7E	Incorrect data returned by the slave
	0x80	Incorrect Modbus slave address
	0x81	Incorrect data frame length, serial port 1 (COM1)
	0x82	Illegal data address
	0x83	CRC error
	0x84	Unsupported instruction code
	0x85	Receiving timeout
	0x86	Illegal data value
	0x87	Buffer overflow
	0x88	Frame error
	0x89	Serial protocol error
	0x8C	Incorrect address
	0x8D	No data received
	0x8E	Incorrect data returned by the slave
	0x90	Incorrect Modbus slave address
	0x91	Incorrect data frame length, Ethernet (Modbus TCP)
	0x92	Illegal data address
	0x93	CRC error
	0x94	Unsupported instruction code
	0x95	Receiving timeout
	0x96	Illegal data value
	0x97	Buffer overflow
	0x98	Frame error
	0x99	Serial protocol error
	0x9A	Slave not connected
	0x9B	Incorrect protocol identifier
	0x9C	Incorrect address
	0x9D	No data received
	0x9E	Incorrect data returned by the slave
	0x9F	Number of connected clients out of range
	0xA0	Illegal data value

EtherCAT Diagnosis Code

Name	Diagnosis Code	Diagnosis Information
m_LastError	0	No error
	1	PLC communication disconnected. Check the network cable.
	2	An error occurred to the working counter of the synchronization unit.
	3	An error occurred to the bus distribution clock.
	4	Failed to start the first network adapter.
	5	Failed to start the second network adapter.
	6	The first and the second network adapters share the same address.
	7	Failed to initialize all slaves.
	8	An error occurred to the working counter.
	9	Inconsistency between the read slave vendor ID and the configuration
	10	Inconsistency between the read product ID and the configuration
	11	An error occurred while writing an SDO.
	12	SDO transmission timing out
	13	Slave emergency message
	14	An error occurred while writing an SOE.
	15	SOE transmission timing out
	16	Watchdog timing out
	101	Unknown error
	102	Failed to request memory for the mailbox.
	106	Inconsistency between the firmware version and EEPROM
	107	Failed to update the firmware
m_LastError	117	Invalid request state change
	118	Uncertain state request
	119	Unsupported guidance mode
	120	Invalid firmware program
	121	Invalid mailbox configuration in Boot mode
	122	Invalid mailbox configuration in Pre-op mode
	123	Invalid SM channel configuration
	124	Invalid input data
	125	Invalid output data
	126	A synchronization error occurred.
	127	An error occurred to SM watchdog.
	128	Invalid SM type
	129	Invalid output configuration
	130	Invalid input configuration
	131	Invalid watchdog configuration
	132	Starting the slave in cold mode
	133	Initializing the slave
	134	Pre-operating the slave
	135	Operating the slave securely
	136	Invalid input mapping

Name	Diagnosis Code	Diagnosis Information
m_LastError	137	Invalid output mapping
	138	Inconsistent settings
	139	The slave failing to run in free mode
	140	The slave failing to run in synchronous mode
	141	The slave needs to run in three buffer modes in free mode.
	142	Watchdog
	143	Invalid I/O data
	144	A severe synchronization error occurred.
	145	No synchronization interrupt signals for the slave
	146	Too short synchronization cycle period
	147	
	148	Invalid DC synchronization configuration
	149	Invalid DC latch configuration
	150	A PLL error occurred.
	151	A DC I/O error occurred.
	152	Invalid DC timeout
	153	Invalid synchronization cycle period
	154	Inappropriate DC cycle for Sync0
	155	Inappropriate DC cycle for Sync1
	165	An MBX_AOE error occurred.
	166	An MBX_EOE error occurred.
	167	An MBX_COE error occurred.
	168	An MBX_FOE error occurred.
	169	An MBX_SOE error occurred.
	179	An MBX_VOE error occurred.
	180	Failed to access the EEPROM address.
	181	An EEPROM error occurred.
	182	External hardware unready
	212	Inconsistency between the configuration detected by the slave and that in the programming software

Shenzhen Inovance Technology Co., Ltd.

Add.: Building E, Hongwei Industry Park, Liuxian Road, Baocheng No. 70 Zone, Bao'an District, Shenzhen
Tel: +86-755-2979 9595
Fax: +86-755-2961 9897
Service Hotline: 400-777-1260
http: //www.inovance.com

Suzhou Inovance Technology Co., Ltd.

Add.: No. 16 Youxiang Road, Yuexi Town, Wuzhong District, Suzhou 215104, P.R. China
Tel: +86-512-6637 6666
Fax: +86-512-6285 6720
Service Hotline: 400-777-1260
http: //www.inovance.com