AX Series Programmable Controller Software Manual

SHENZHEN INVT ELECTRIC CO., LTD.

No.	Change description	Version	Release date
1	First release	V1.0	September 2020
2	1. Modified the variables PLC and ErrorID to setError and getError in Table 4-1. 2. Updated the remarks of inTime and inDate in Table 4-1 and Table 4-2. 3. Updated the variable definition table in A.1.1.1, A.1.2.1, and A.2.1.1.	V1.1	June 2021
3	1. Added a Table 3-2 "Example of bit, byte, word, and double word correspondence of AX series controllers" and updated the data in Table 3-1 and Table 3-3. 2. Updated the section 5.2, changing the "error code" to "fault code". 3. Added the description of PC communication configuration for Windows10 when the hardware is connected with Mini USB cable in section 2.3.	V1.2	November 2021
4	1. Added Figure 2-28, Figure 2-29, and updated related descriptions in section 2.6.2. 2. Added section 4.2.1.1 "P-type model port configuration description" and section 4.2.1.2 "N-type model port configuration description". 3. Added Figure 4-7, Figure 4-8, and Figure 4-9, and updated related descriptions in sections 4.4 and 4.5. 4. Deleted section 4.5.3 "Temperature module" and Appendix A "Function module command". 5. Added section 4.7 "Distributed I/O module", section A.3 "Controller and DA200 Series Servo CANopen Configuration Example" and Appendix B "SMC_ERROR description".	V1.3	August 2022

Preface

Thank you for using the AX series programmable controller (programmable controller for short).

This manual contains the information required to use the AX series programmable controllers. Please read this manual carefully before using the product. Then you can fully understand the functions, performance, and system build-up, which helps to give full play to the advanced performance.

Target audience

Personnel with electrical professional knowledge (such as qualified electrical engineers or personnel with equivalent knowledge)

Applicable product

AX70 programmable controller

AX71 programmable controller

AX72 programmable controller

AX series programmable controller backplane expansion modules

AX series programmable controller bus expansion modules

Online support

You can also obtain product documentation and technical support from INVT website:

http://www.invt.com

If the product is ultimately used for military affairs or weapon manufacture, abide by the export control regulations in the *Foreign Trade Law of the People's Republic of China* and complete related formalities.

The manual is subject to change without prior notice.

Contents

1 Product Introduction

1.1 AX series programmable controller

1.1.1 Overview

The AX series programmable controller is a high-performance programmable controller designed with a modular structure to provide users with intelligent automation solutions. It adopts IEC61131-3 programming language system and supports six standard programming languages: IL, LD, FBD, ST, SFC, and CFC. High-level motion control functions such as electronic cams, electronic gears, synchronous control, and positioning can be realized through EtherCAT bus. Supporting 200kHz high-speed I/O, the programmable controller can realize motion control functions such as linear interpolation and circular interpolation.

The programmable controller is rack-mounted. Each rack can embrace 16 functional extension modules, including digital input/output modules, analog input/output modules, temperature modules and communication modules. Remote I/O extension can be carried on via EtherCAT fieldbus.

In addition, programmable controller supports various communication interfaces such as EtherCAT, CANopen, RS485 and Ethernet to meet the diverse application requirements of users.

1.1.2 Product configuration and module description

The AX series programmable controller CPU supports the following modules: power supply module, digital input module, digital output module, analog input module, analog output module, temperature module and communication module. Take AX70-C-1608P as an example, the diagram of system combination is as follows.

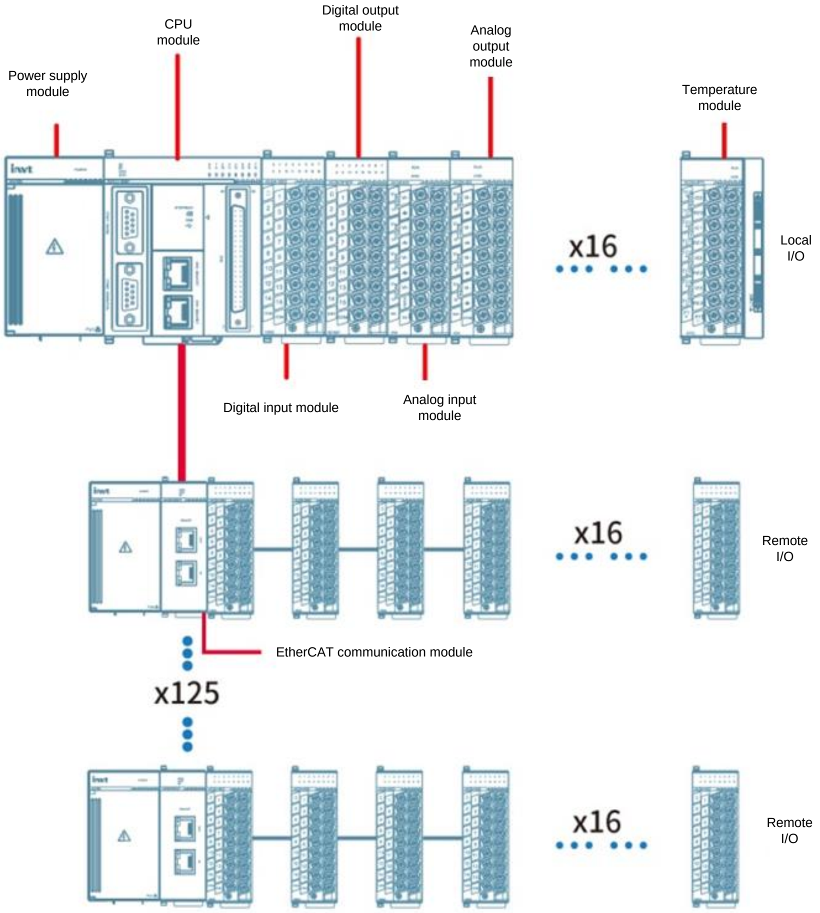


Figure 1-1 System integration

1.1.3 System application process

1. Install the power supply module, CPU module, and expansion modules.
2. Provide power and perform wiring for related modules.
3. Turn on the power only after confirming that the wiring of each module is correct and the power supply voltage meets the specifications.
4. Connect the computer that hosts Invtmatic Studio to the CPU module.
5. Download the program created on Invtmatic Studio and related parameters to the CPU module.
6. Ensure that the nixie tube of the CPU module does not show any fault code and the fault indicators of the CPU module and other modules do not turn on.

1.2 Programming platform

1.2.1 Invtmatic Studio

Invtmatic Studio is a programming platform developed by Shenzhen INVT Electric Co., Ltd. It fully supports the IEC61131-3 programming language system and six standard programming languages: IL, LD, FBD, SFC, ST, and CFC.

1.2.2 Software programming interface

The interface of **Invtmatic Studio** software after creating an application project is shown in Figure 1-2.

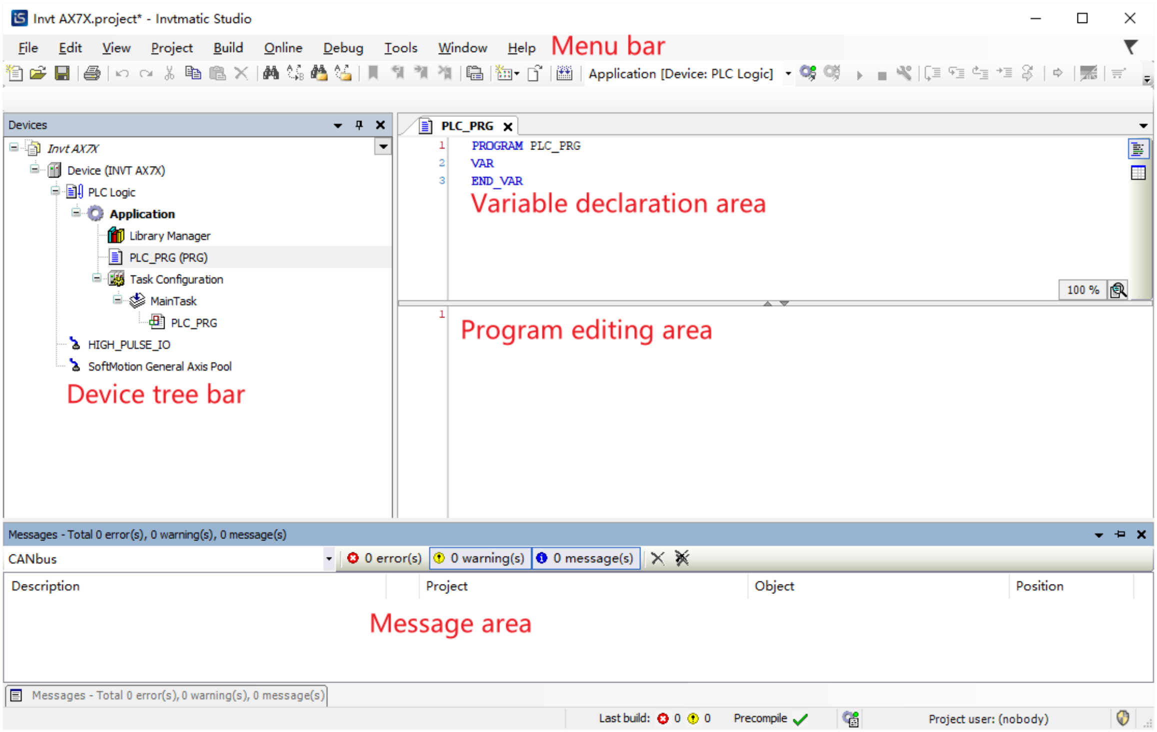


Figure 1-2 Invtmatic Studio software application engineering interface

1.3 PLCopen specification

Founded in 1992, PLCopen is a vendor- and product-independent worldwide association. One of the core activities of PLCopen is focused around IEC 61131-3, the only global standard for industrial control programming. A standard programming interface allows people with different backgrounds and skills to create different elements of a program during different stages of the software lifecycle: specification, design, implementation, testing, installation and maintenance. Yet all pieces adhere to a common structure and work together harmoniously. The standard includes the definition of six programming languages: Continuous Function Chart (CFC), Sequential Function Chart (SFC), Instruction List (IL), Ladder Diagram (LD), Function Block Diagram (FBD) and Structured Text (ST). Via decomposition into logical elements, modularization, and modern software techniques, each program is structured, increasing its re-usability. For programmers, the programming technology based on IEC61131-3 can be widely used in the entire industrial control field.

Invtmatic Studio programming platform used in AX series programmable controller fully supports the PLCopen specification and allows users to reference many standard function libraries. The high-level language programming approach makes it easy for controller manufacturers and users to develop their own proprietary function blocks and instruction libraries and to borrow existing similar control programs to form industry-specific "process packages", which can significantly improve user programming efficiency.

2 Getting Started

2.1 Software installation and uninstallation

2.1.1 Software obtaining

INVT AX series programmable controller user programming software contains Invtmatic Studio platform, installation files and related reference materials. You can get them by the following ways:

1. Visit INVT website (www.invt.com) and go to **Support** > **Download** > **Software** to download the software installation package for free.
2. Obtain software installation CDs from all levels of INVISTA distributors.

2.1.2 Software installation requirements

You can install the software on a computer or desk:

3. Installed with Windows 7/ Windows 8/ Windows 10 operation system
4. CPU clock speed: 2GHz or higher
5. Memory: 2GB or higher
6. Available hardware space: 5GB or higher

2.1.3 Preparing

If it is the first time to install Invtmatic Studio, check whether your computer meets the software installation requirements. If yes, you can install it directly.

If you want to install the latest version of Invtmatic Studio, check the version information about the installed software by choosing **Help** > **About**. If it is not the latest version, you can upgrade the software using the online upgrade method.

Figure 2-1 Version information

2.1.4 Installing the software

1. Locate the installation file storage path, and double-click **Invtmatic Studio Setup 64 Vx.x.x.exe**. (take V1.0.2 as an example)
2. The installation starts. See the following figure.

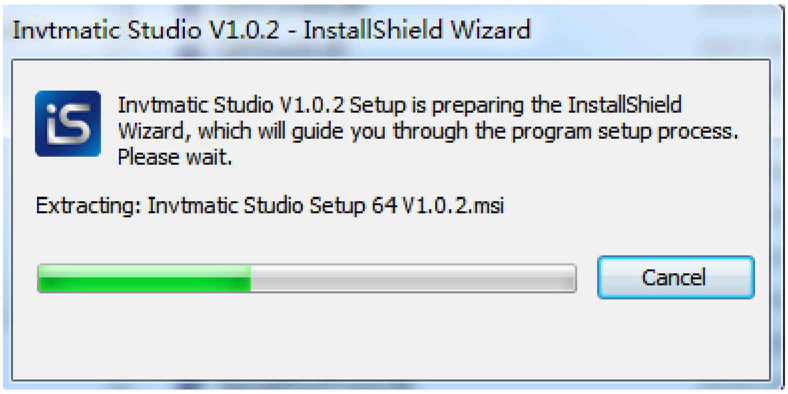


Figure 2-2 Installation preparation

3. When the dialog box shown in the following figure appears, click **Next**.

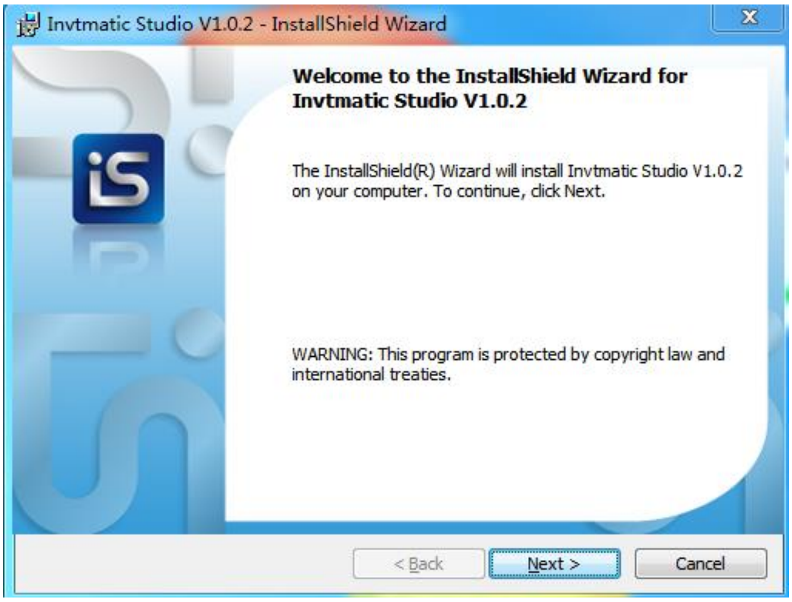


Figure 2-3 Installation wizard

4. Then the license agreement dialog box appears. Select **I accept the terms in the license agreement**, and then click **Next**.

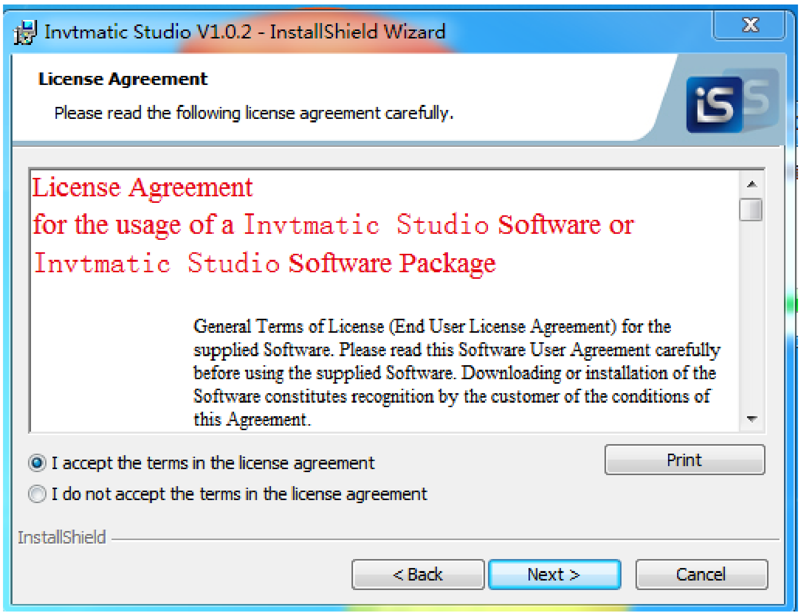


Figure 2-4 License agreement

5. Set the software installation path, and click **Next**.

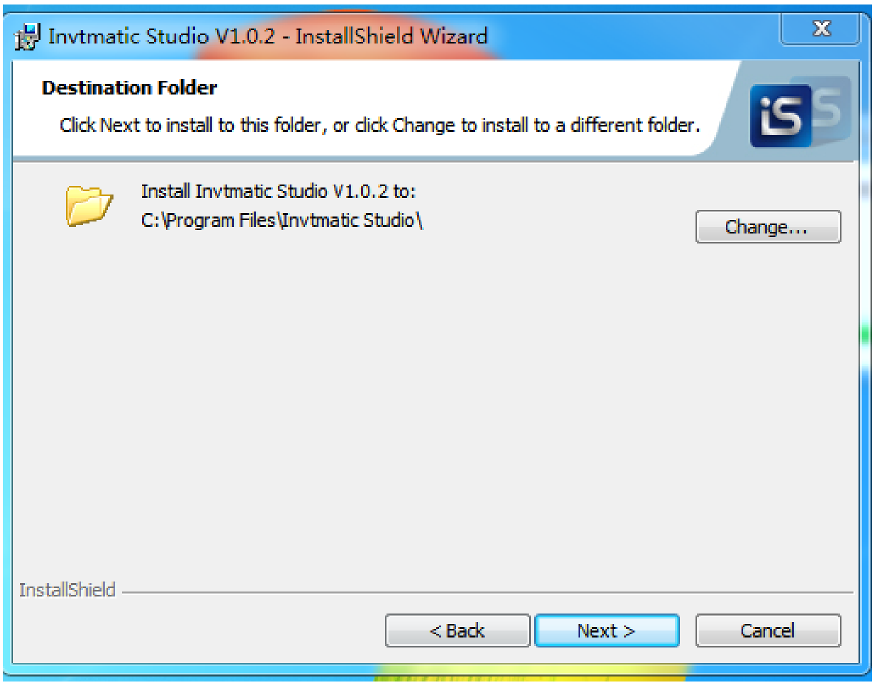


Figure 2-5 Installation path

6. The installation component selection interface appears. Select an installation option. If you have no special requirement, keep the default selection, and click **Next**.

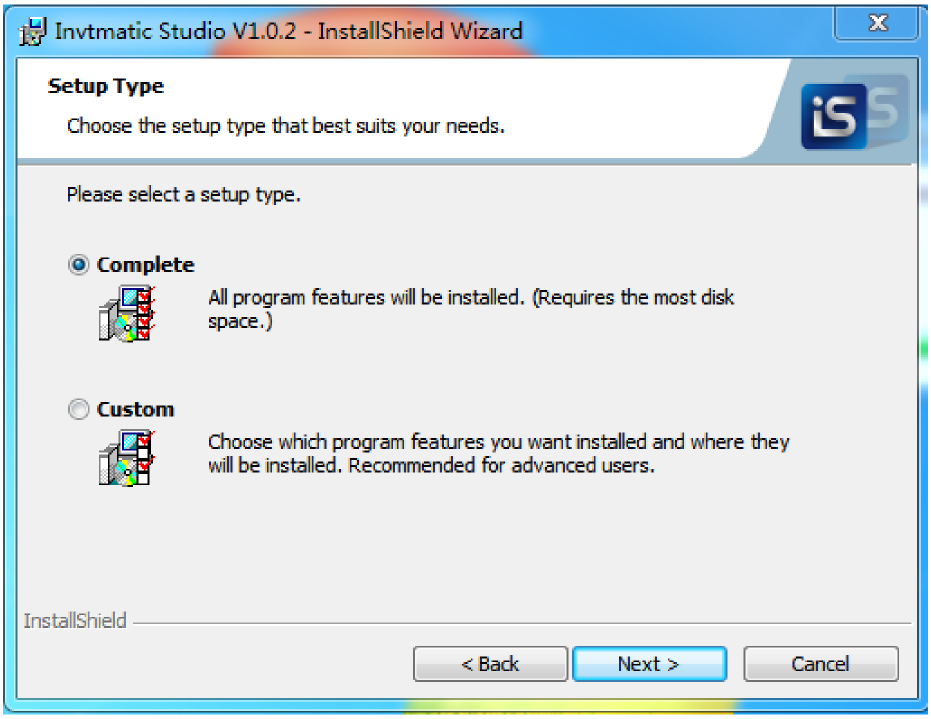


Figure 2-6 Installation type

7. When the following interface appears, click **Install**.

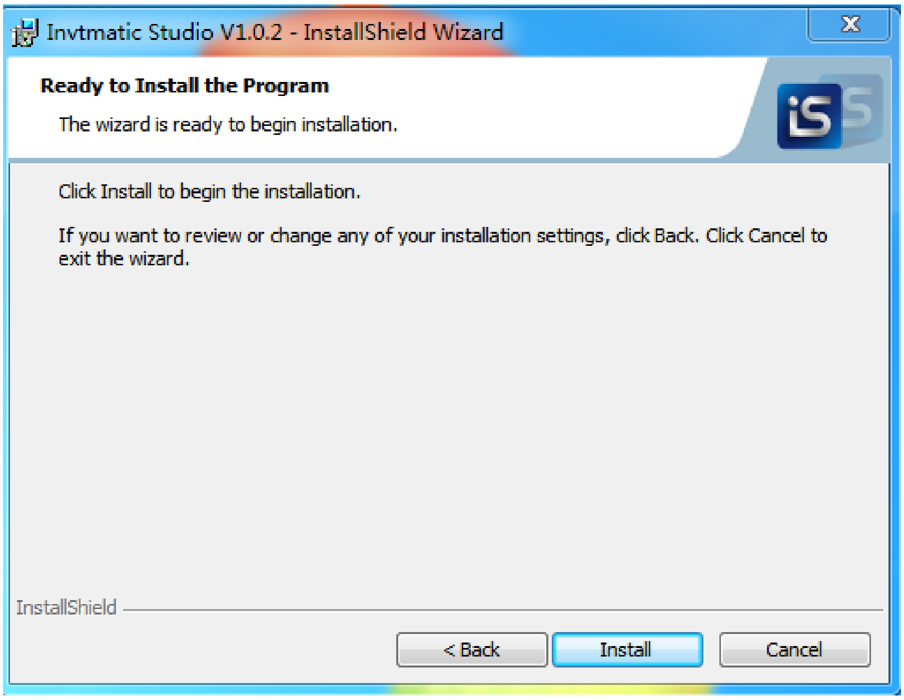


Figure 2-7 Start installation

8. An installation progress bar appears. Click **Finish** when the installation is completed.

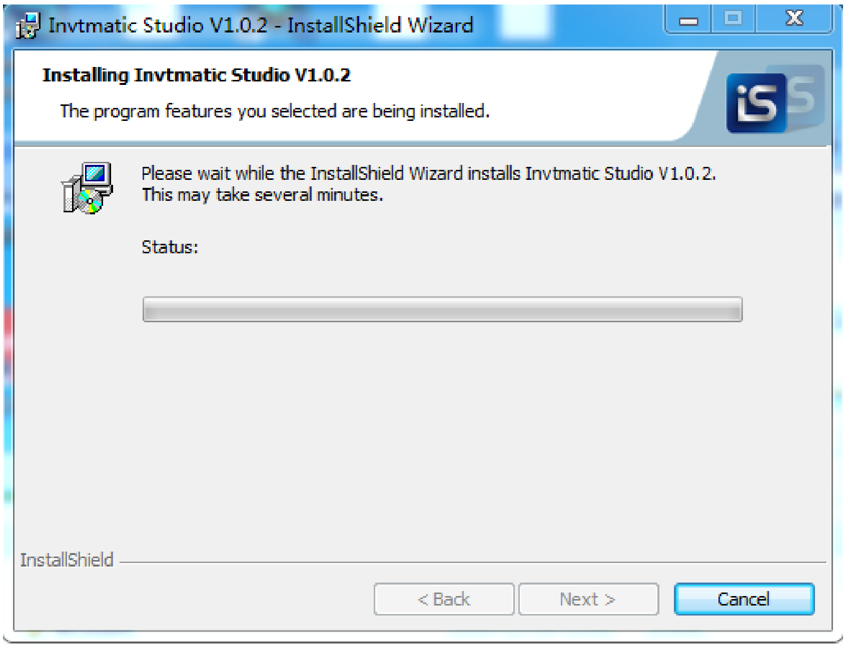


Figure 2-8 Installation progress

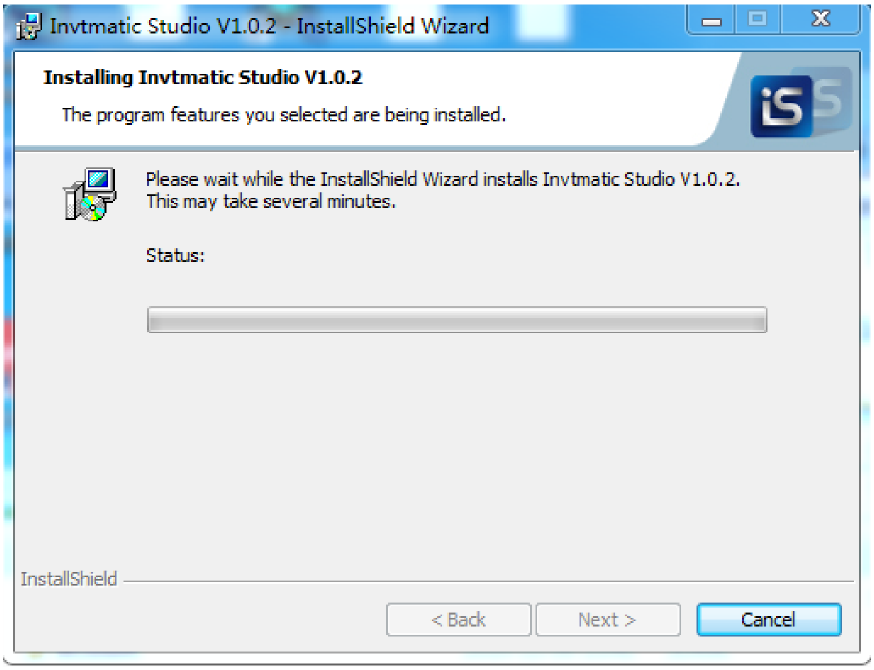


Figure 2-9 Installation complete

2.1.5 Uninstalling the software

Uninstall Invtmatic Studio by using the standard software uninstallation method of a Windows system. The procedure is as follows:

Step 1 Shut down Invtmatic Studio running programs, including the backend running program.

Step 2 Enter the control panel, find and right-click Invtmatic Studio, and click **Uninstall**.

Step 3 Wait until the software is uninstalled.

2.2 AX series programmable controller connection

There are two types of hardware connections between an upper computer and programmable controller:

1. Using Mini USB cable

2. Using LAN network cable

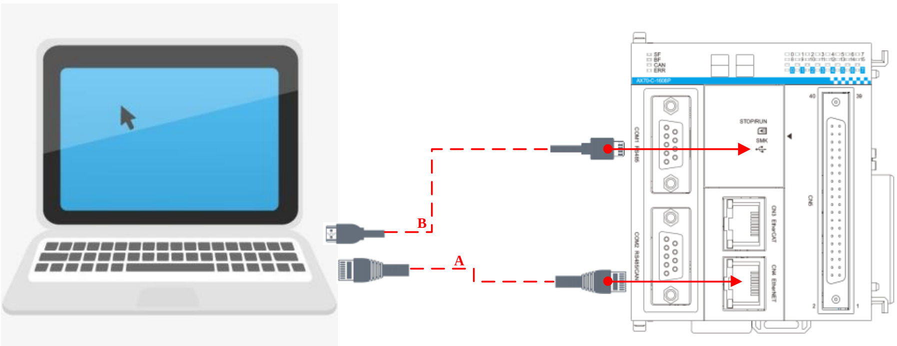


Figure 2-10 Hardware connection diagram

2.3 PC communication configuration

- If the hardware is connected with a LAN network cable, ensure that the IP address of the PC and the IP address of the controller are in the same network segment. The factory default IP address of the AX series programmable controller is 192.168.1.10, so the IP address of the PC should be set to 192.168.1.xxx. (xxx means any integer value in the range of 1－254 except the end address of the controller IP).

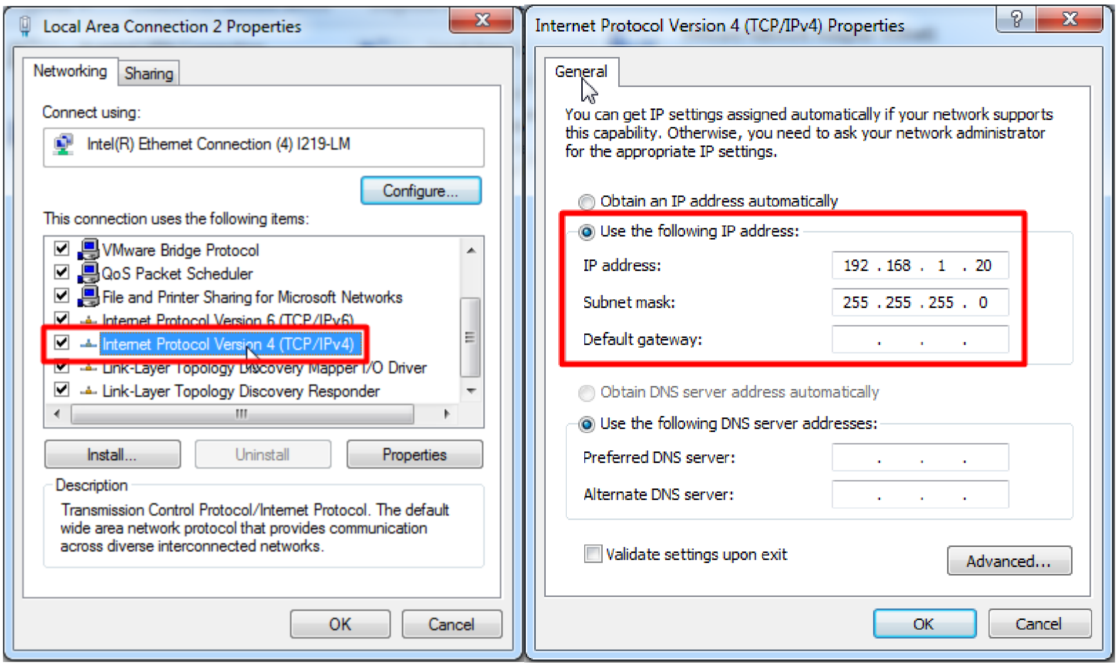


Figure 2-11 PC communication configuration for LAN network cable connection

- If the hardware is connected with Mini USB cables, configure the PC as follows.

When the PC runs Windows7:

✧ Install USB drive

1) In **Computer Management** window, select **Device Manager**, right click the RNDIS/Ethernet Gadget device and select **Update driver**.

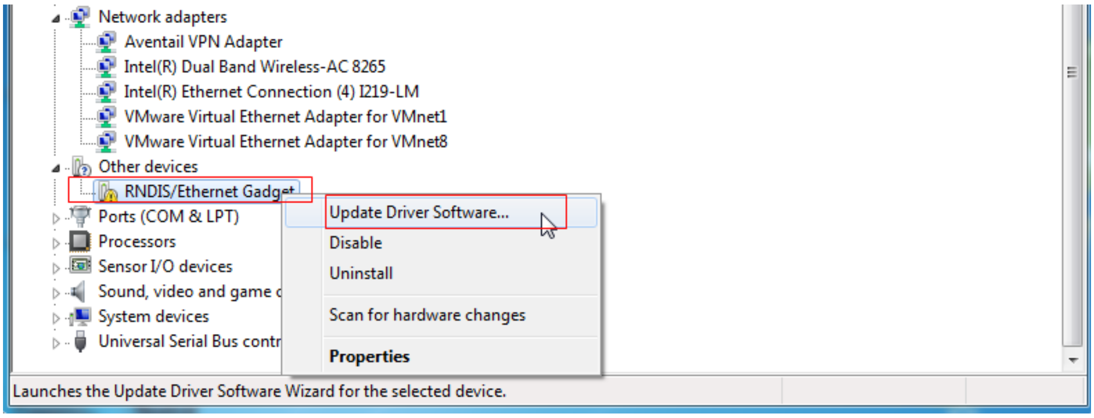


Figure 2-12 RNDIS/Ethernet Gadget

2) Select **Browse my computer for driver software** > **Let me pick from a list of device drivers on my computer** > **Network adapter** > **Microsoft Corporation** > **Remote NDIS Compatible Device**, and then click **Next**.

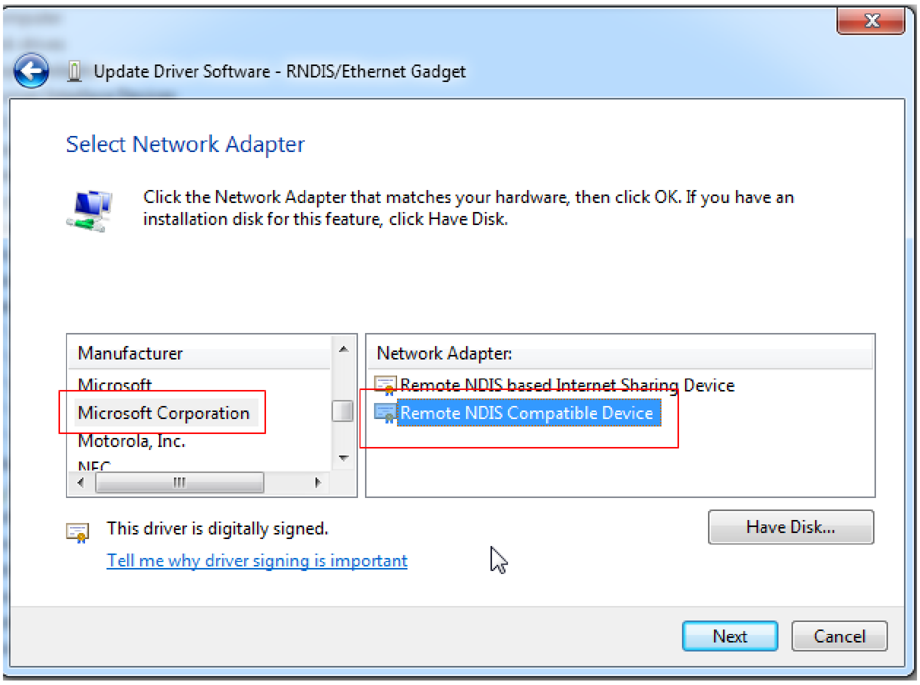


Figure 2-13 Select driver software

3) After the installation, start the controller and connect it to the PC with a Mini USB cable. The USB driver is displayed in the computer device manager.

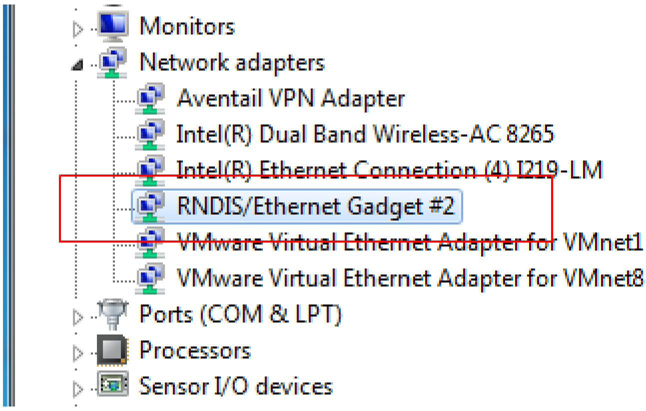


Figure 2-14 Install the driver

✧ Configure USB IP address

1) Go to **Control Panel** > **Network and Internet**, right click **Local Area Connection** of RNDIS and select **Properties**. In the **Properties** window, select **Internet Protocol Version 4 (TCP/IPv4)**.

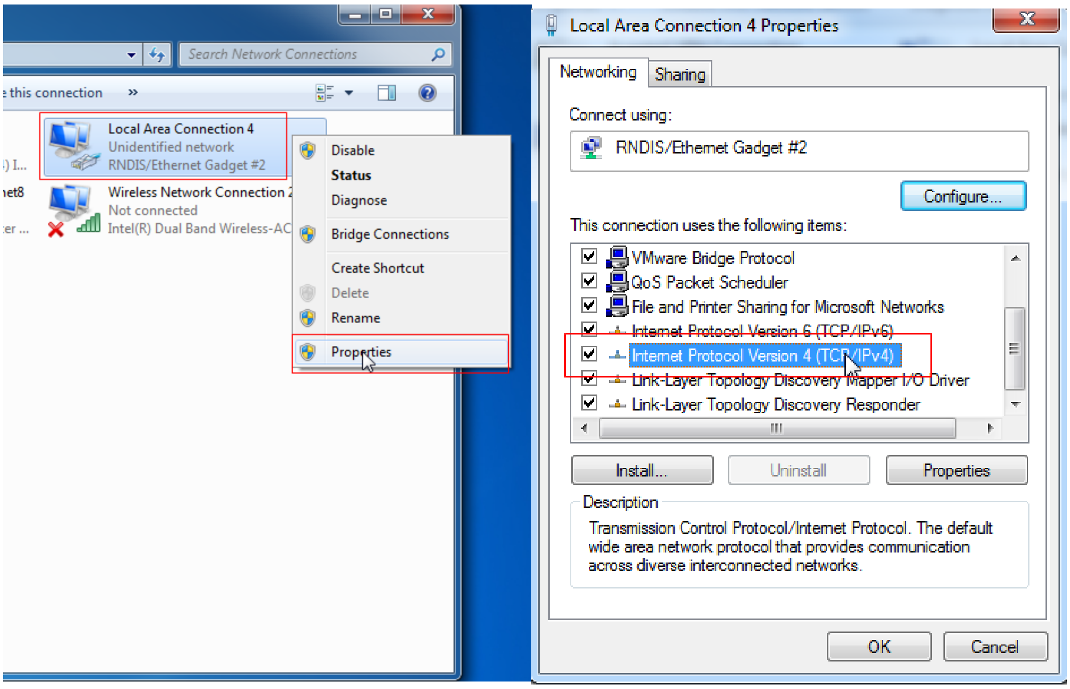


Figure 2-15 Select local area connection of RNDIS

2) Configure the IP address on network segment 192.168.2.xxx, in which xxx is within 1-254 (except 10). Click **OK** to complete the IP address configuration.

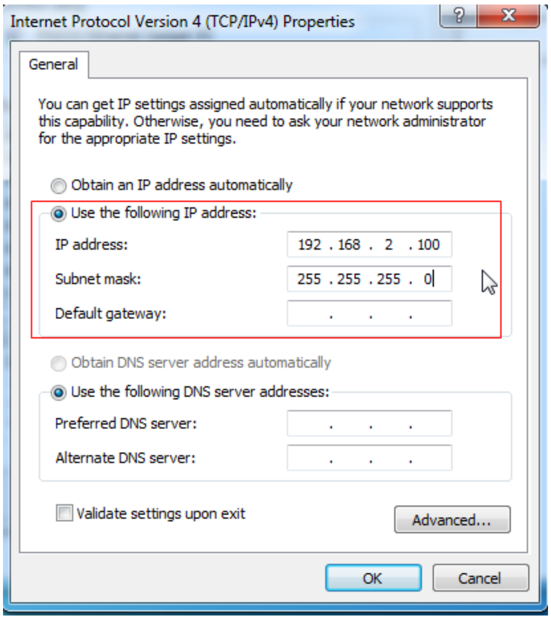


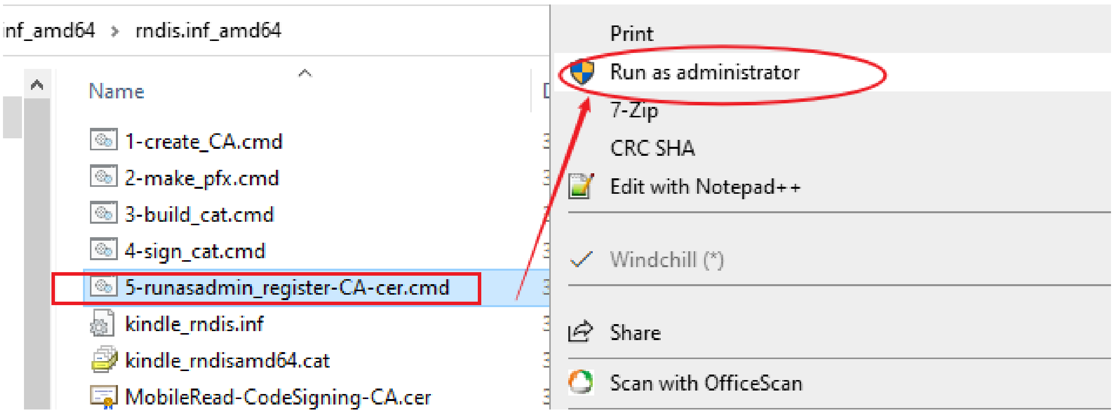
Figure 2-16 IP address configuration

When the PC runs Windows10:

✧ Install the driver

kindle_rndis.inf_amd64 is the USB driver file.

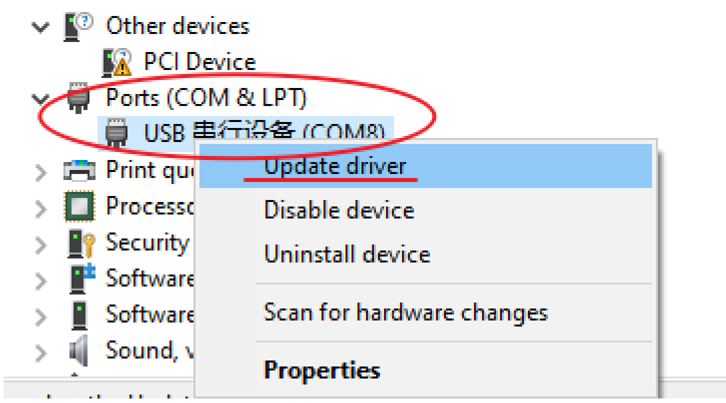
1) Right-click the file “**5-runasadmin_register-CA-cer.cmd**” and select **Run as administrator**.



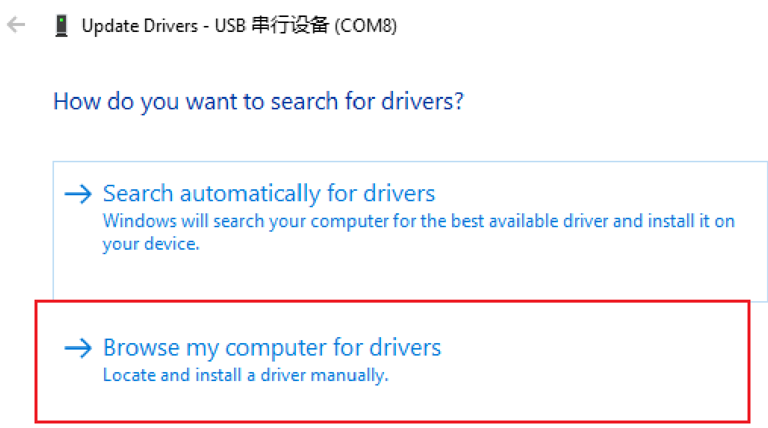
2) Press any key.

```
Exact match:
Element 2:
Serial Number: 3c914b8e900338844fafb20aadb325c8
Issuer: CN=MobileRead-CodeSigning-CA, O=MobileRead Forums, OU=Marco77
 NotBefore: 3/20/2016 7:36 PM
 NotAfter: 1/1/2040 7:59 AM
Subject: CN=MobileRead-CodeSigning-CA, O=MobileRead Forums, OU=Marco77
Signature matches Public Key
Root Certificate: Subject matches Issuer
Cert Hash(sha1): 58e285d47509e810dccfa865a8f6d99b8a297fa5

Certificate "MobileRead-CodeSigning-CA" already in store.
CertUtil: -addstore command completed successfully.

C:\Users\Administrator\Desktop\rndis.inf_amd64\rndis.inf_amd64>pause
Press any key to continue . . .
```

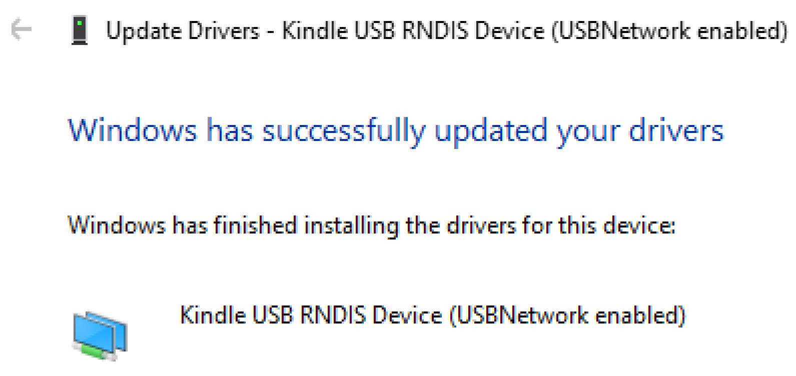
3) Connect the computer and the PLC with a USB cable, open **Device Manager**, and right-click the USB serial device under the **Ports** node.



4) Click Browse my computer for drivers and select the driver folder.



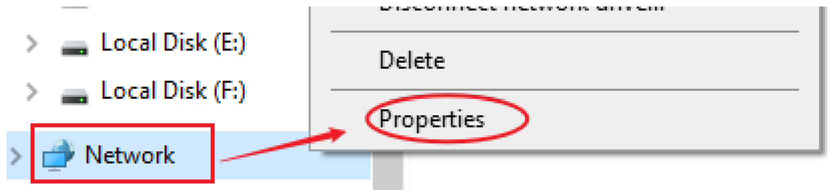
5) Wait for the installation process completed.



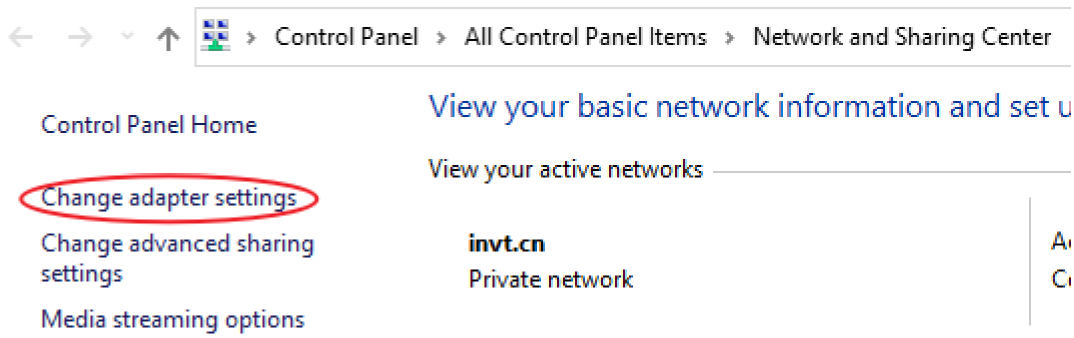
The USB RNDIS item has been added to the Network Adapters node in Device Manager.

✧ Configure USB network port

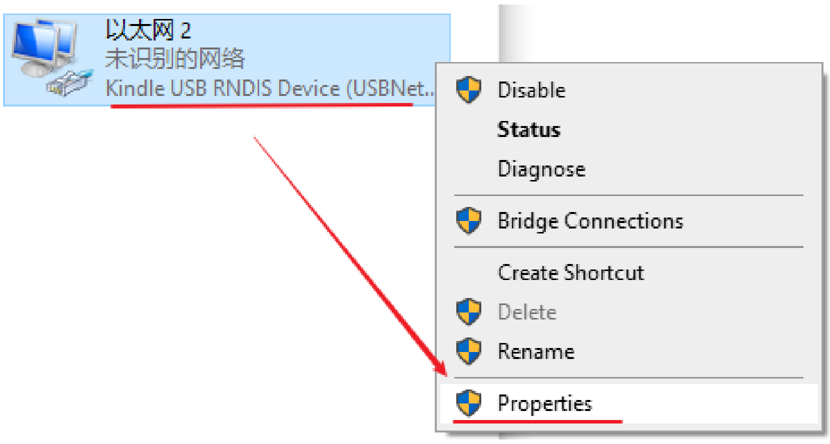
1) Right-click the **Network** menu and select **Properties**.



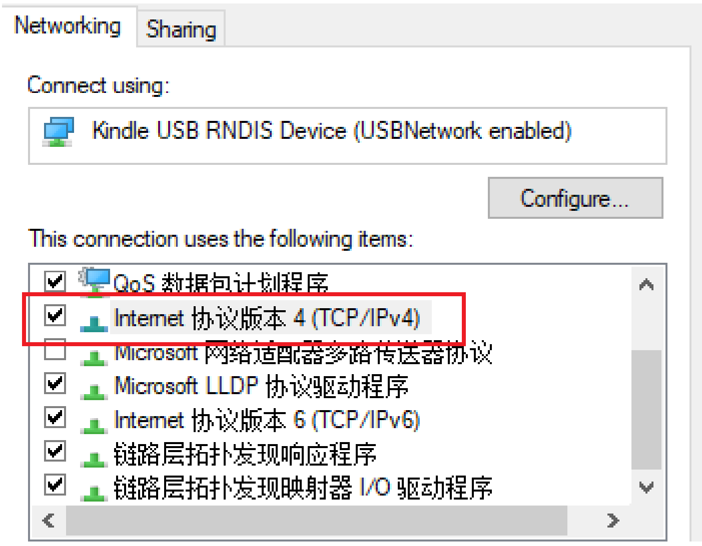
2) Click **Change adapter settings**.



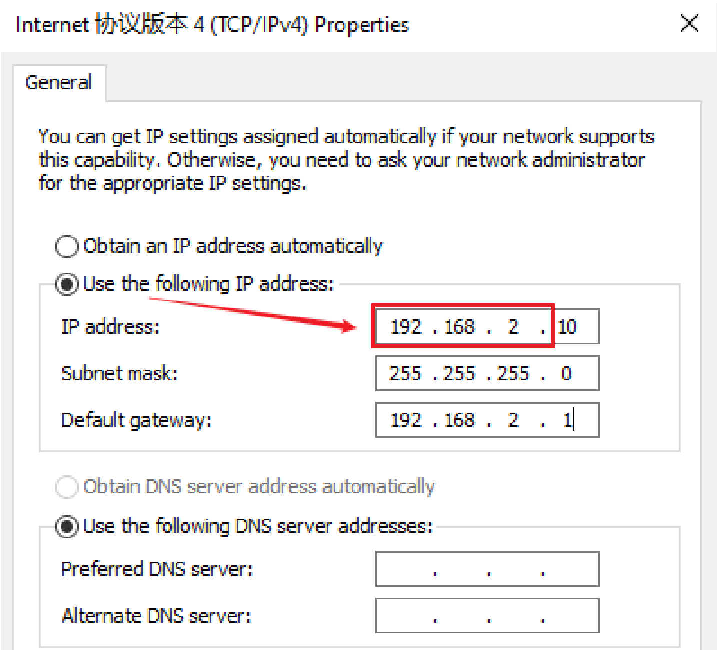
3) Right-click the Unidentified network with “**USB RNDIS**” in its name, and select **Properties**.



4) Select **Internet Protocol Version 4 (TCP/IPv4)** and click **Configure…**.



5) Set the IP address manually. The IP address must be in the network segment 192.168.2.xxx, in which xxx is within 1-254 (except 10).



2.4 Project creation

2.4.1 Starting the programming environment

1. Take Invtmatic Studio V1.0.2 as an example. Double-click the software icon of Invtmatic Studio. The programming environment is as follows:

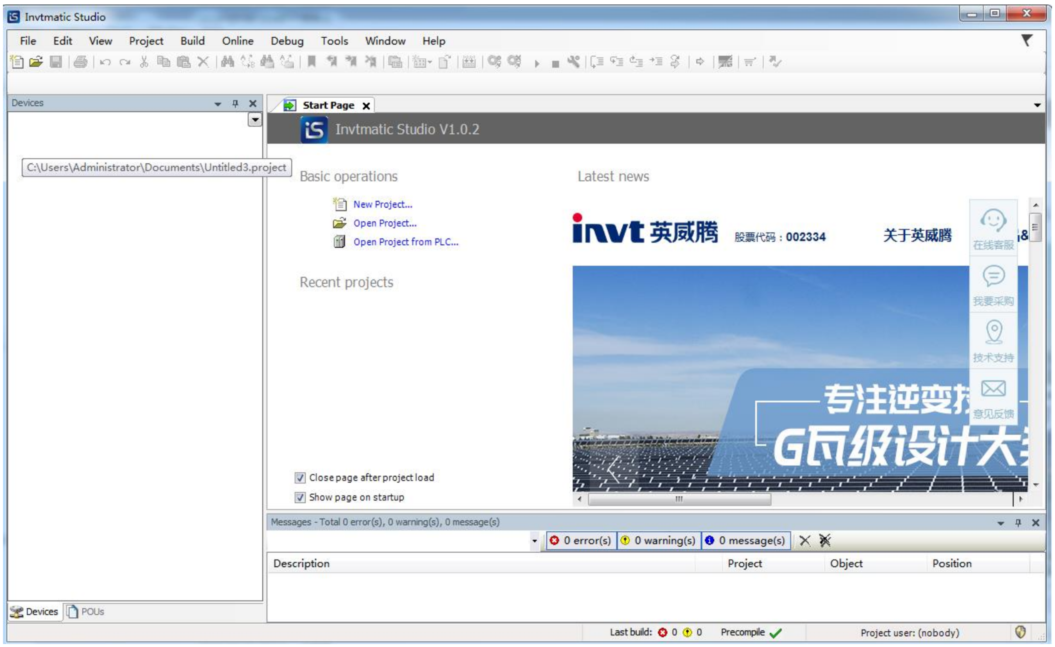


Figure 2-17 Invtmatic Studio homepage

2. In the tool bar, select **Tool** > **Device repository** to add a device profile.

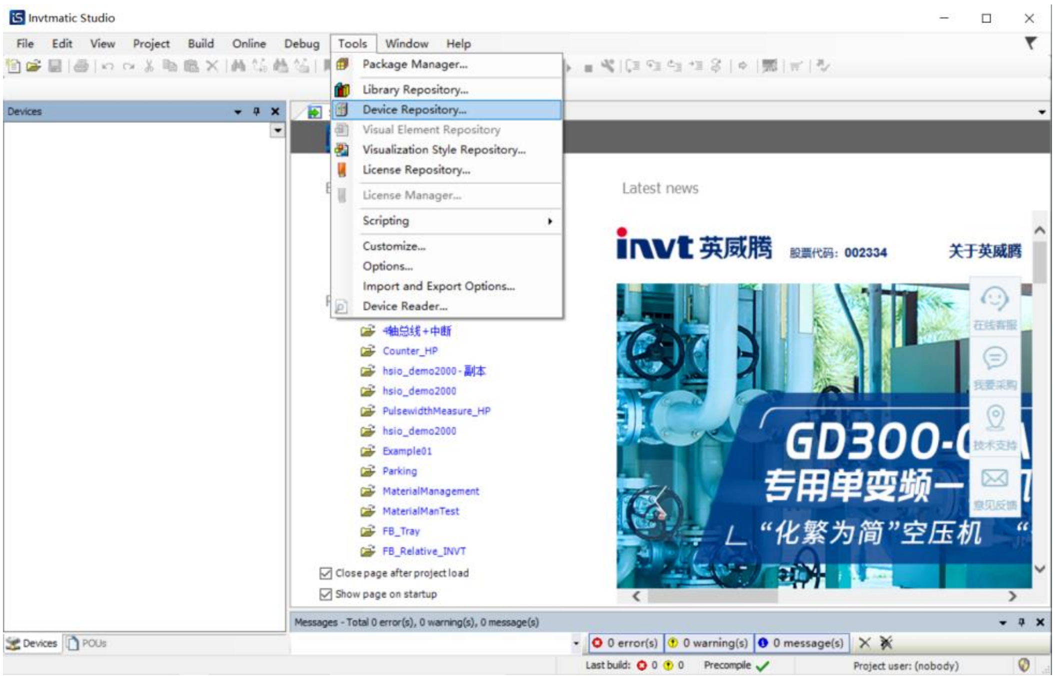


Figure 2-18 Add device profile

3. In the **Device repository** pop-up window, click **Install**.

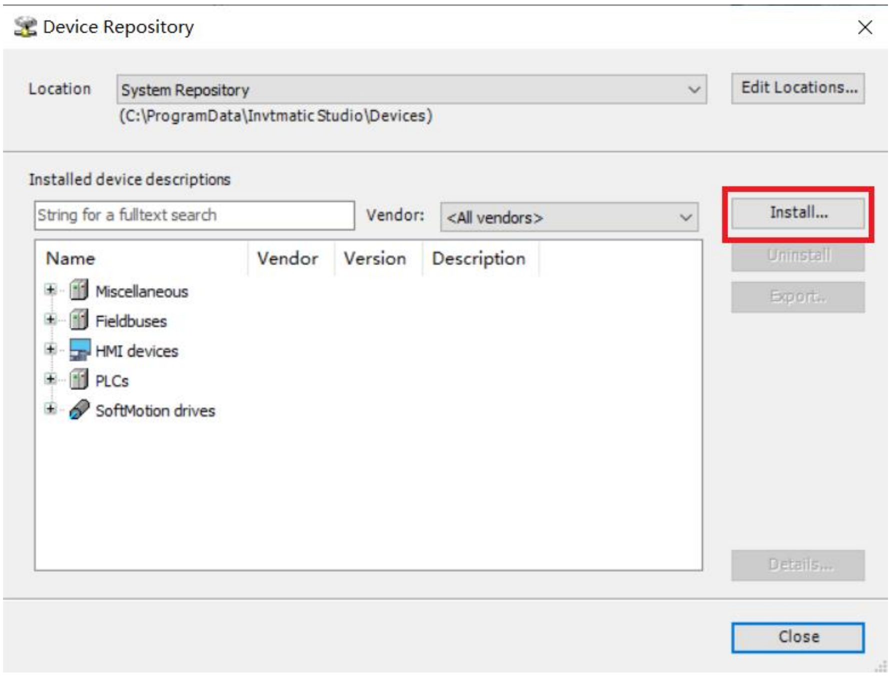


Figure 2-19 Install device

4. From the **Install device profile** window, select the device profile to be installed from a local folder and then click **Open**.

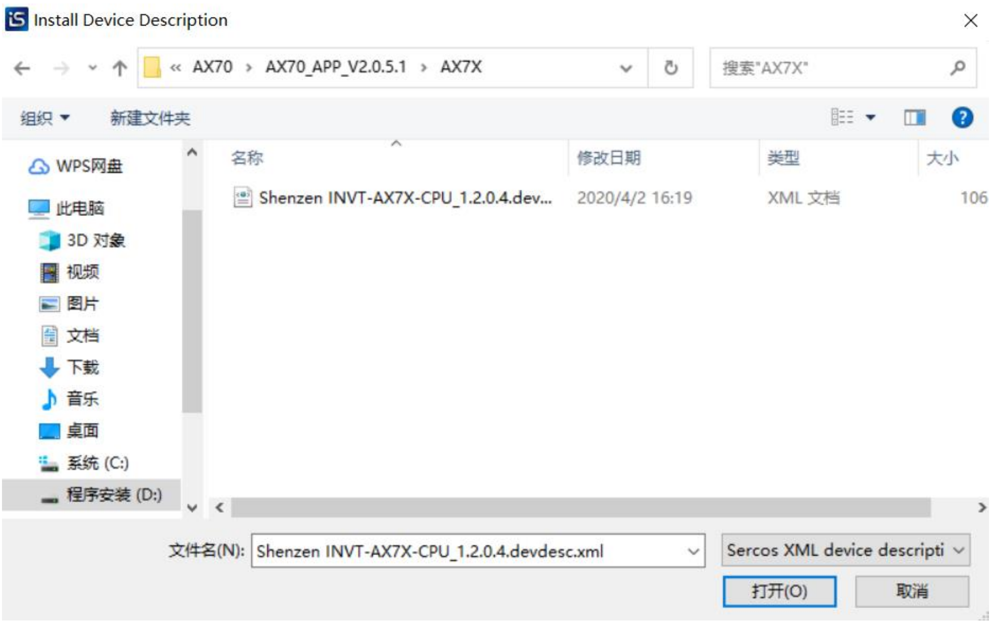


Figure 2-20 Install device profile

Note: All device profiles provided by INVT can be added by following the steps above.

2.4.2 Creating new project

1. Click the project creation icon at the upper left corner or choose **File** > **New Project**, or directly click **New Project** in the window to quickly create a project. Select the project category, template, save path and file name, as shown in the following figure.

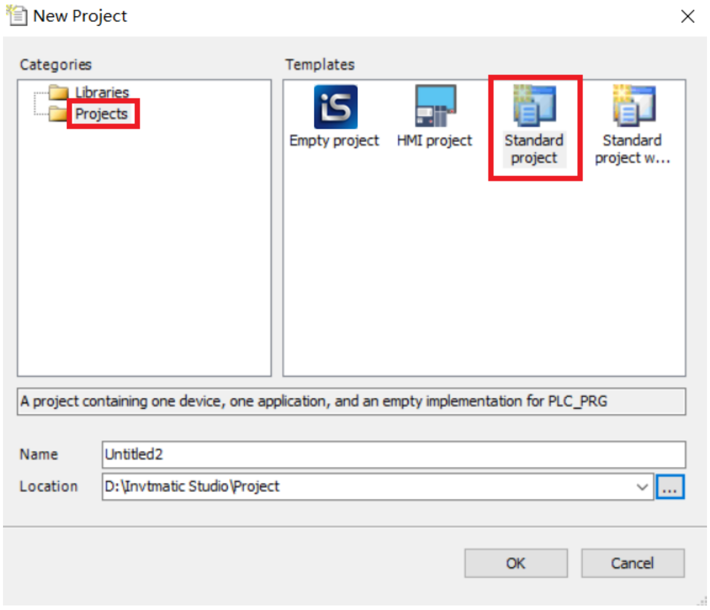


Figure 2-21 New project

2. Click **OK**. On the standard project setting interface that appears, select the device type and programming language. See the following figure.

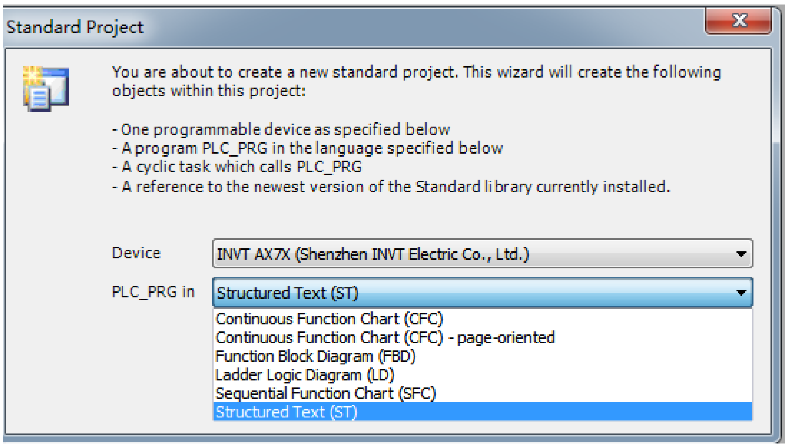


Figure 2-22 Standard project setting page

3. On the configuration and programming interface, double-click **PLC_PRG(PRG)** to write programs. See the following figure.

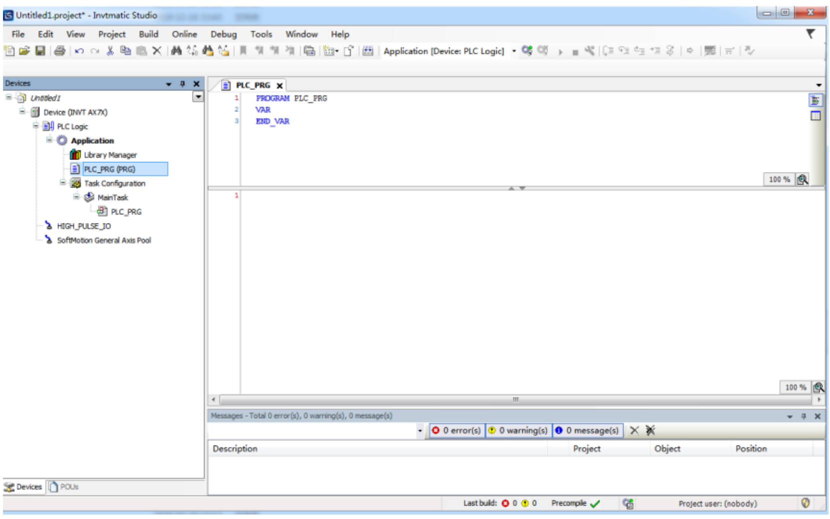


Figure 2-23 Invtmatic Studio configuration and programming page

2.5 Typical steps of project writing

From the above example, writing a user program with MC motion control functions generally requires the following steps.

Step 1 Application system hardware configuration

Configure network according to the main controller, expansion module, network type, servo slave node and other hardware used.

Step 2 User program writing

According to the control function to be implemented, write motion control with one POU (such as POU1), and write common logic control with a POU (such as POU2).

Step 3 Servo driver parameter configuration

Configure the objects of SDO and PDO according to the servo name in the hardware configuration and the operation mode of the servo. Ensure that the communication objects required between the MC function block of the user program and the servo are filled in the configuration table.

Step 4 Servo motor parameter configuration

Correctly fill in the resolution of the servo motor encoder, the transmission ratio of the mechanical structure, the characteristics of the axis movement range and other parameters, so that the displacement command of the control object corresponds accurately to the actual displacement.

Step 5 Task arrangement

Based on the real-time requirements of control, execute the motion control function POU1 in the EtherCAT task and set the cycle to 4ms, the priority to 0; execute the common logic control POU2 in common tasks and set the cycle to 20ms, the priority to 16.

Step 6 Online debugging

Connect the AX series programmable controller to PC via LAN network correctly. Power on the programmable controller, download and debug the user program, and eliminate user program bugs (if possible, you can connect the servo drive system to the programmable controller and then debug. If the servo system is not available, you can set the servo as a virtual axis; if the programmable controller is not available, you can simulate and debug the user program on the PC to eliminate possible errors in the user program).

2.6 Examples of program writing and debugging

Here is an example of a basic servo control program to give you a first glimpse of the programming process before you go through the principle of the programming system and the method of compiling the motion control program.

Write a simple program that allows the AX series CPU programmable controller to implement the following functions:

The servo motor repeats rotating forward 50 revolutions, and then reversing 50 revolutions.

The programming method and steps of the routine are as follows:

Step 1 Add the corresponding equipment: EtherCAT master node, servo drive, motor shaft.

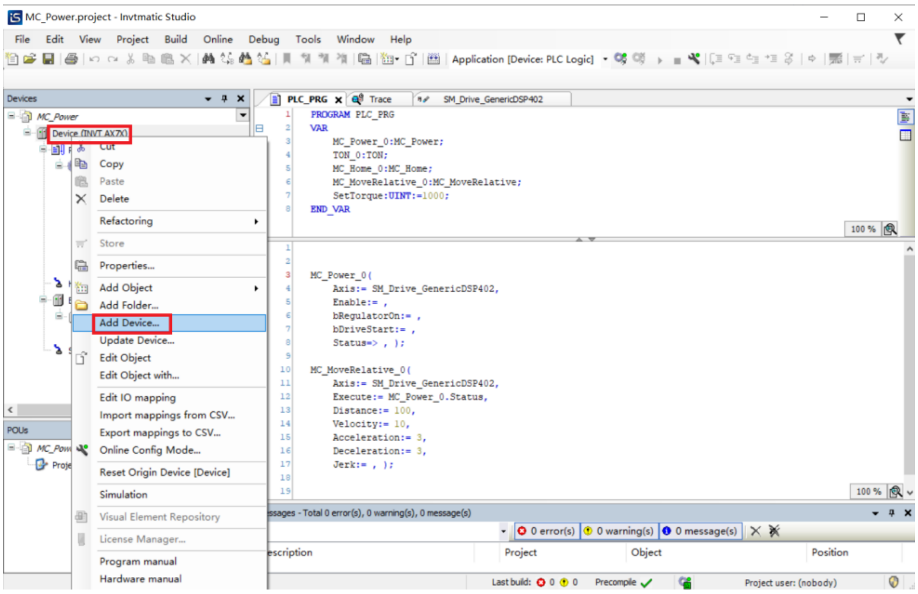
Step 2 Handle the motion control of the servo in the high real-time EtherCAT task cycle.

Step 3 Set relevant parameters.

Step 4 Write program.

2.6.1 Adding devices

1. Add an EtherCAT SoftMotion master node and an EtherCAT network bus.



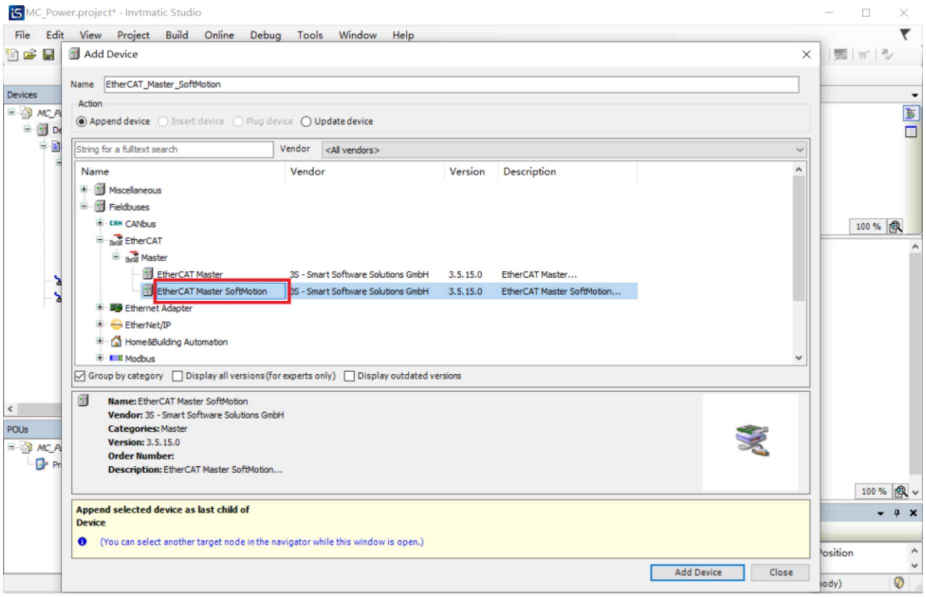
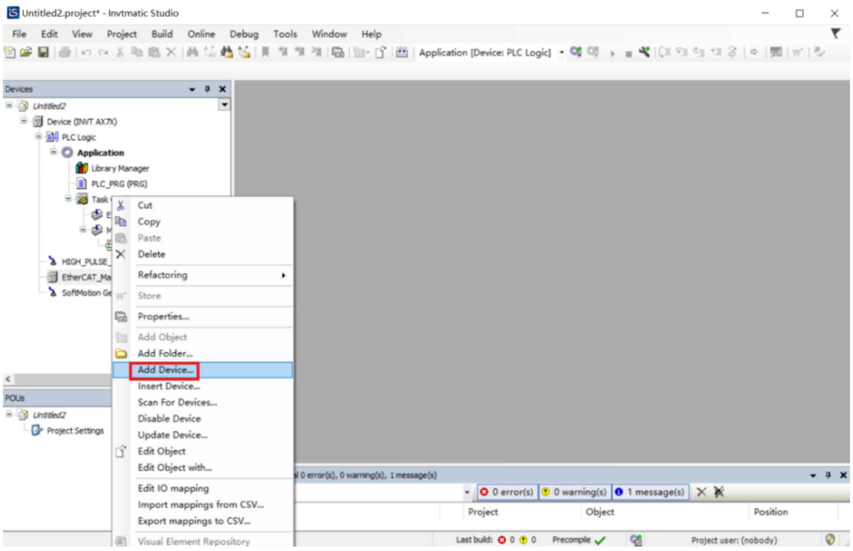


Figure 2-24 Add EtherCAT master node

2. Add a servo device.



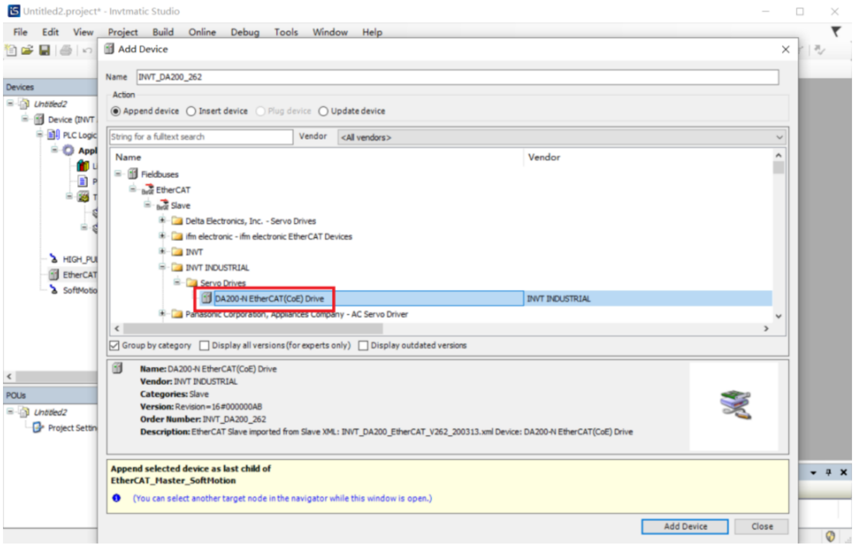


Figure 2-25 Add EtherCAT slave node

3. Add a servo axis.

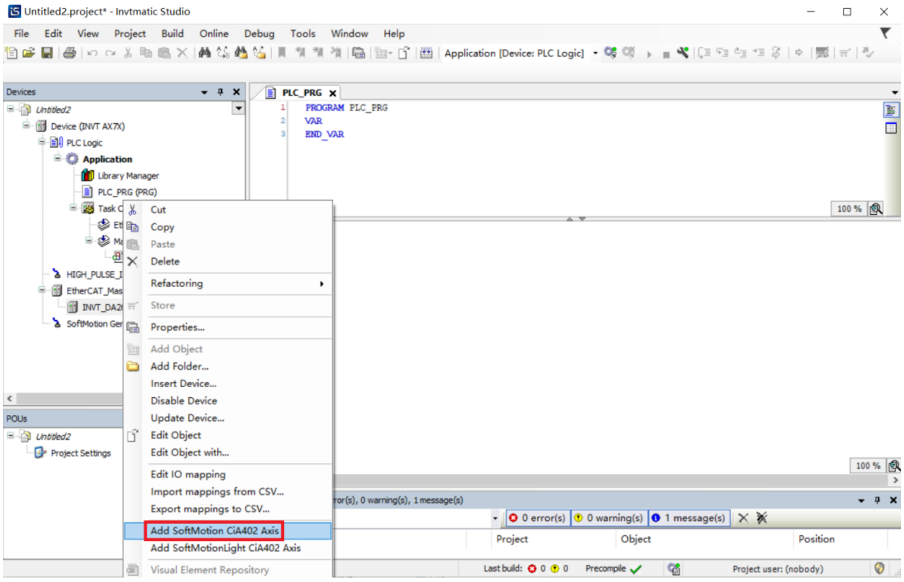


Figure 2-26 Add a servo axis

2.6.2 Writing a function to handle POU

In Invtmatic Studio programming environment, there is an EtherCAT_Task task and a MainTask task for the default task configuration. The MainTask task contains a POU named PLC_PRG which is created at the same time as the new project is created. Create a POU for servo control under the EtherCAT_Task task.

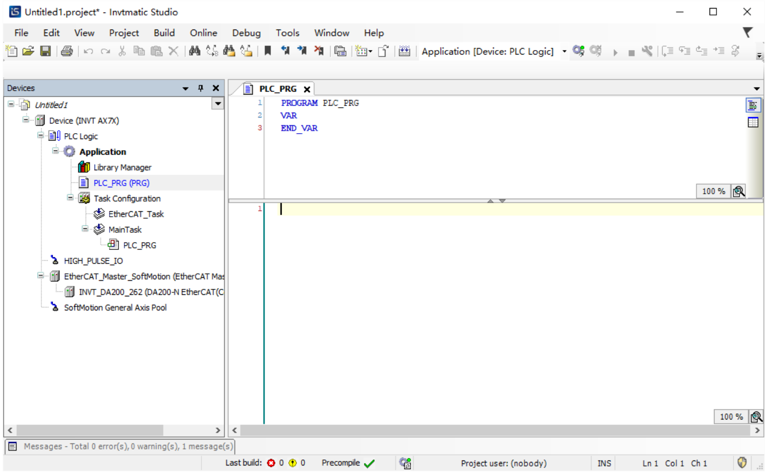


Figure 2-27 PLC_PRG programming page

1. Right-click the **Application** in the device tree, select **Add Object** > **POU** to add a POU for EtherCAT servo control.

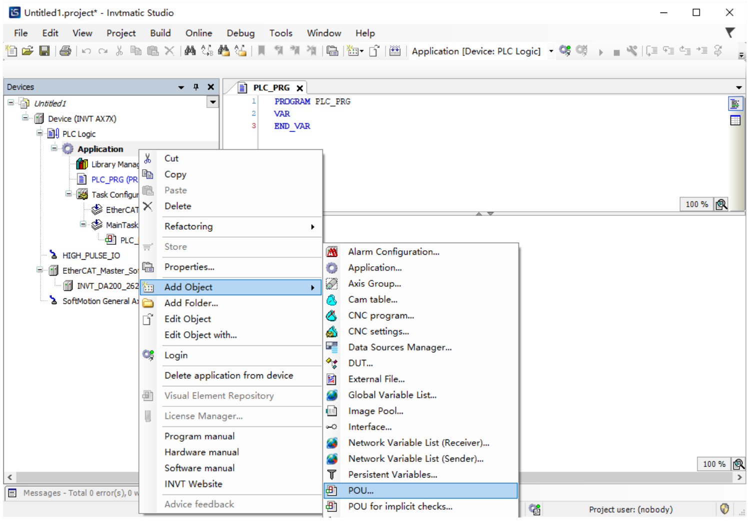


Figure 2-28 Add a POU

2. Double-click **EtherCAT_Task** in the device tree and click **Add Call** in the configuration interface to select **EtherCAT POU**.

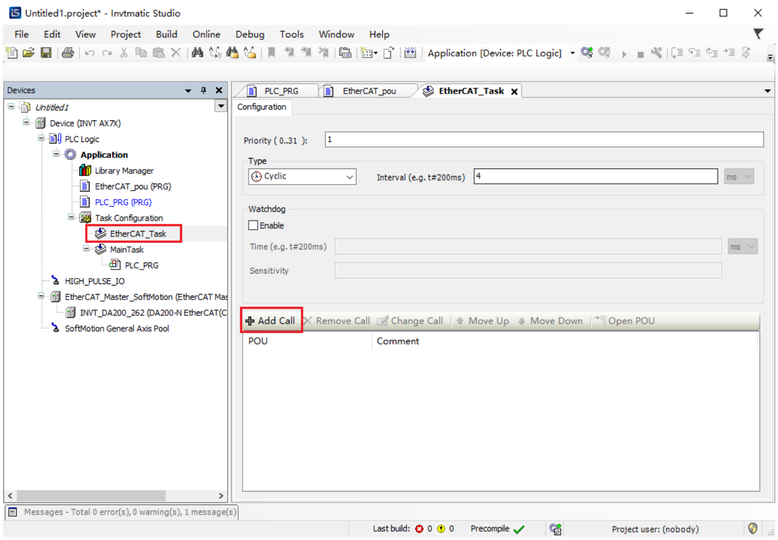


Figure 2-29 Call a POU for EtherCAT task

2.6.3 Setting motor parameters

For precise control of the movement position, the programmable controller must accurately calculate the position of the servo motor. Based on the operating characteristics and stroke characteristics of the application system, select the **Axis type and limit**. Therefore, the programmable controller can calculate the feedback information of the motor encoder to obtain the accurate position, and then avoid errors caused by the accumulated overflow of the encoder pulse number.

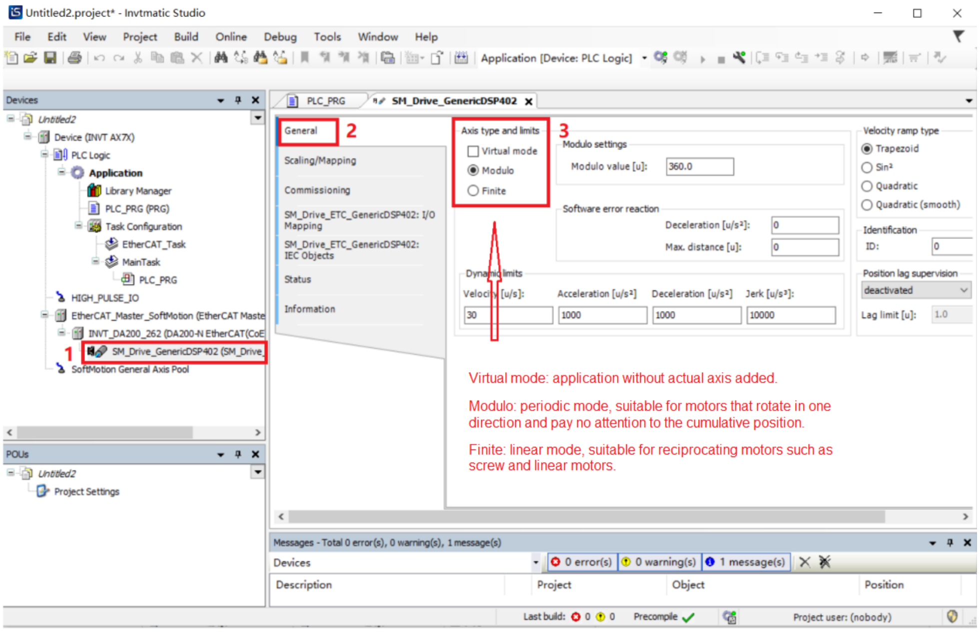


Figure 2-30 Motor parameter settings

For the reciprocating mechanism of the lead screw type, **Finite** is preferred as the lead screw stroke is limited and we should know its absolute position within the stroke range.

For a single-direction shaft, **Modulo** is preferred as the linear mode may cause position counting overflow, resulting in position calculation errors.

The encoder parameters of the motor (such as resolution) and the mechanical deceleration ratio of the application system may be different. They need to be set based on the actual situation during programming, as shown in the following figure.

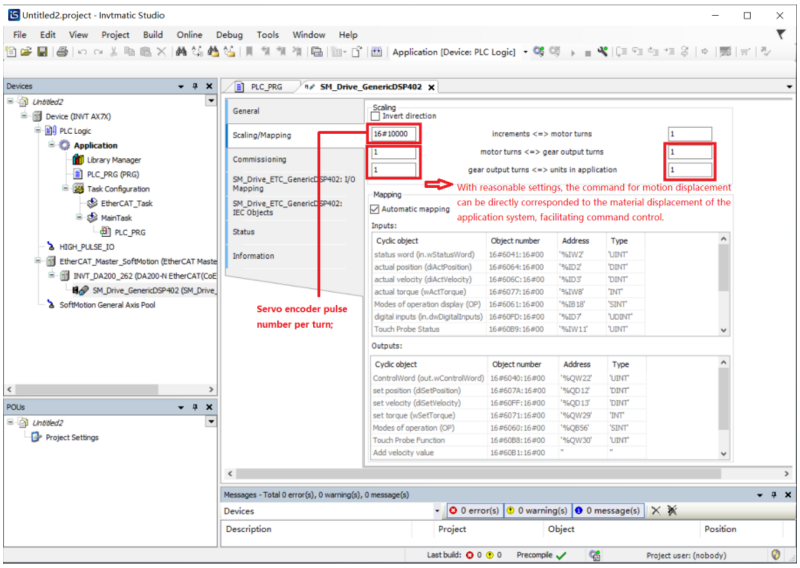


Figure 2-31 Motor encoder parameter settings

The DA200 servo matching motor has two typical resolutions. The resolution of normal incremental encoders is 20bit, that is, 1048576 pulses per revolution; and the resolution of absolute encoders is 23bit, i.e. 8388608 pulses per revolution. In actual operation, the programmable controller sends the required number of pulses to the servo drive by EtherCAT communication to control the servo operation. Therefore, the encoder resolution needs to be accurately set according to the actual situation, as shown in the figure above. Take a 20bit encoder without a reducer as an example. When the servo is commanded to run 1 unit, the servo will select 1 revolution (axis moves 360°). If the field unit in application (circled in red in the figure above) is set to 360, the servo will select 1/360 circle (axis moves 1°) when the servo is commanded to run 1 unit, and so on. After setting the corresponding parameters (commonly known as electronic gear ratio) according to the actual mechanical structure, you can input the distance command according to the physical unit of the application system movement distance, making the control parameters intuitive and easy to understand.

Please note that only integer numbers can be entered in the fields circled in red in the figure above. Because the ratio of the parameters in the corresponding rows on the left and right sides is effective, you can enter appropriate integer values in the corresponding rows on the left and right sides. For example, to enable the drive lead with screw rod 6.8mm (that is, the screw rod rotates 1 circle and the screw slide block moves 6.8mm) to move after the servo motor passes through a mechanical deceleration mechanism with a ratio of 4:1, please set as shown in the following figure.

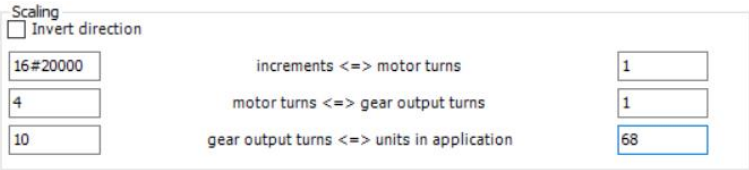


Figure 2-32 Setting example

The dimension of the parameters circled in red can be used as the dimension of the distance in the MC control command later. The settings of the servo driver and motor described above must be set and verified in the corresponding items of the servo axis, otherwise the motor will not operate as expected.

2.6.4 Writing motor positive and reverse

For the motion control of the servo axis, the default synchronization period is 4ms. Users can choose according to the actual need, as shown in the following figure.

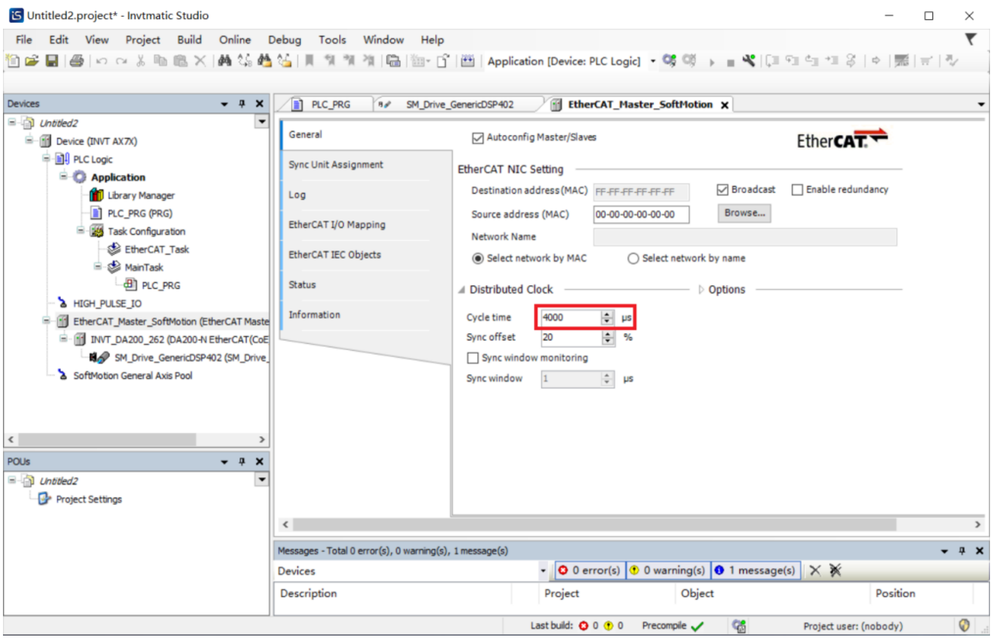


Figure 2-33 Servo axis motion control cycle setting

The program in the above figure is written in ST language. The relevant code is as follows:

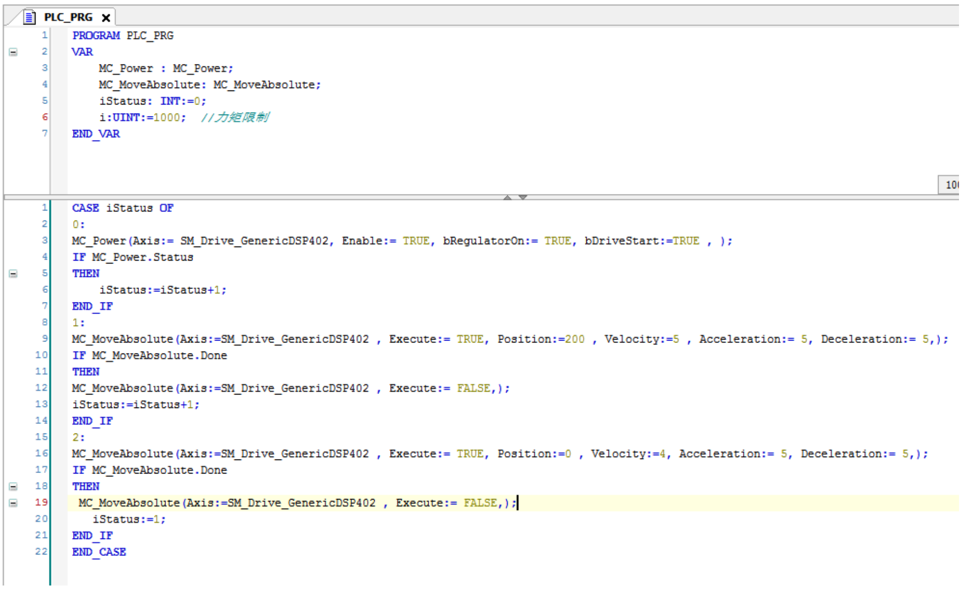


Figure 2-34 ST codes

2.6.5 Compiling user program

If there is a writing error, the error type and reason will be listed in Figure 2-30. Double-click the error description, and the cursor will jump to the corresponding program editing window to facilitate revision. After the revision, compile again until all compilation problems are eliminated.

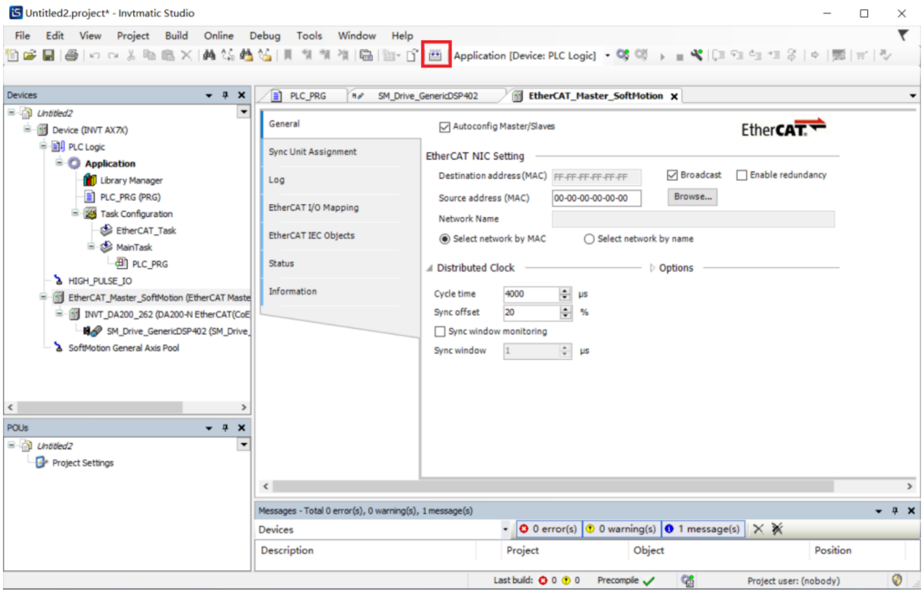


Figure 2-35 Program compilation

Click **Device** > **Communication Settings** > **Scan Network** to select the device and click **Wink**, then the SF, BF, CAN and ERR indicators of the connected device will wink three times. After the device is confirmed, download the user program into the AX series CPU module.

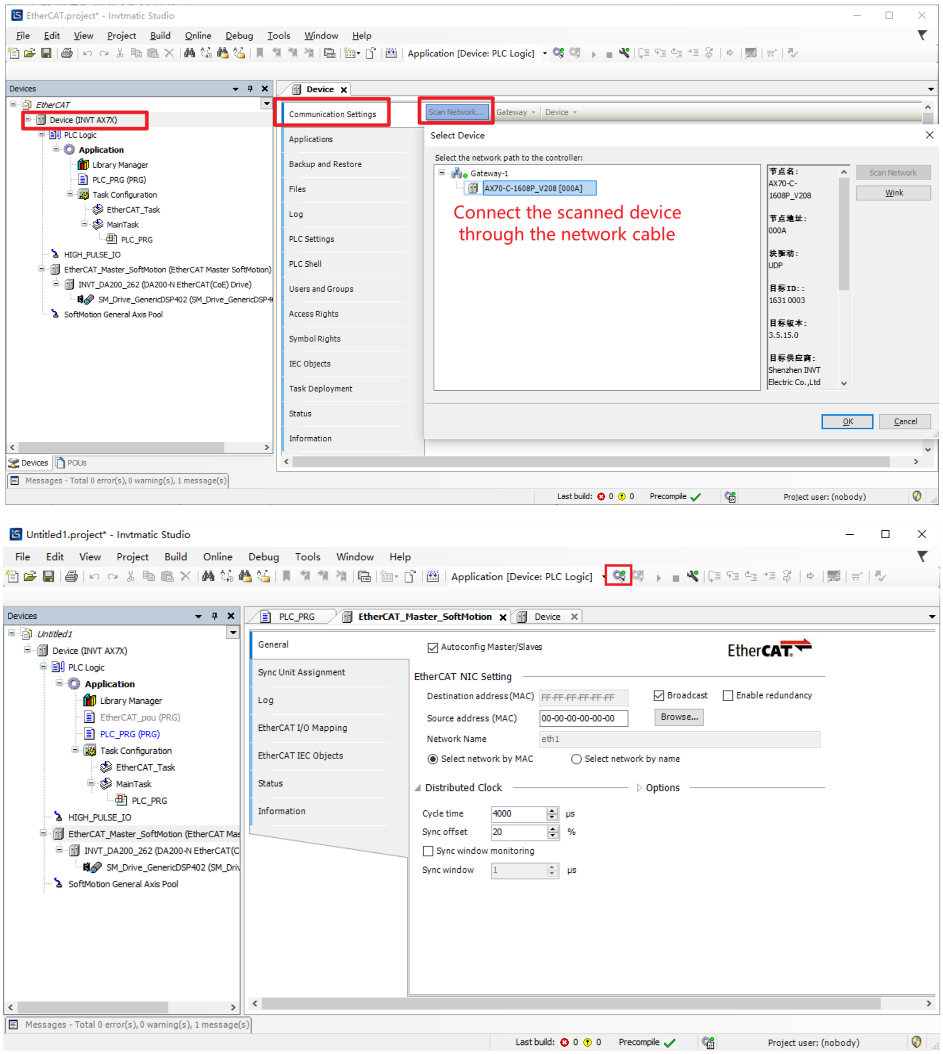


Figure 2-36 User program download

2.6.6 Running monitor program

After logging in to the device through the button marked in a red square in Figure 2-36, the program is running if you can observe the actual operation of the servo or check the position value of the servo axis of the host computer. At this point, the required servo jogging and the 2-cycle running triggering functions has been implemented, which shows the programming process is complete.

3 Network Configuration

The network configuration of AX series programmable controllers mainly includes: ModbusTCP, ModbusRTU, EtherCAT, and CANopen.

3.1 ModbusTCP

3.1.1 ModbusTCP_Master

The number of variables that ModbusTCP can access is defined as follows:

- Read coil (0x01), number of coils 1–2000 (0x7D0)
- Read discrete coils (0x02), number of coils 1–2000 (0x7D0)
- Read holding register (0x03), number of registers 1–125 (0x7D)
- Read input register (0x04), number of registers 1–125 (0x7D)
- Write a single coil (0x05)
- Write a single register (0x06)
- Write multiple coils (0x0F), number of coils 1–1968 (0x7B0)
- Write multiple register (0x10), number of register 1–120 (0x78)

ModbusTCP_**Master** is an important component of the ModbusTCP_**Master** function module. Before using the master node, the corresponding library files must be added as follows:

- Create an application project for the ModbusTCP_**Master**.
- Add the library file "CmpModbusTCP_Master_x.x.x.x.library" required by this module.

3.1.2 ModbusTCP_Slave

- Create an application project for the ModbusTCP_Slave.
- Add the library file "ModbusTCP_Slave_x.x.x.x.library" required by this module.

The ModbusTCP_Slave defines the storage area that can be accessed from outside. The detailed area is as follows:

Table 3-1 ModbusTCP_Slave function codes

Function code of TCP master node	Address name	Range	Offset
01	%QX	0.0-511.7	N/A
05	%QX	0.0-511.7	N/A
02	%IX	0.0-511.7	N/A
04	%IW	0-511	N/A
03/06	%MW	0-8192	5000
03/06	%QW	0-511	N/A
01	%MX	0.0-7565.7	5000
05	%MX	0.0-7565.7	5000

Table 3-2 Example of bit, byte, word, and double word correspondence of AX series controllers

<table>
<tr><td>%_X</td><td>195.7 – 195.0</td><td>194.7 – 194.0</td><td>193.7 – 193.0</td><td>192.7 – 192.0</td></tr>
<tr><td>%_B</td><td>195 (8 most significant bits)</td><td>194 (8 leaset significant bits)</td><td>193 (8 most significant bits)</td><td>192 (8 leaset significant bits)</td></tr>
<tr><td>%_W</td><td colspan="2">97 (16 most significant bits)</td><td colspan="2">96 (16 leaset significant bits)</td></tr>
<tr><td>%_D</td><td colspan="4">48</td></tr>
</table>

3.2 ModbusRTU

AX series programmable controllers support two Modbus serial communications, COM1_RS485 and COM2_RS485, both of which support the standard ModbusRTU protocol, and can be independently configured as a master or slave, supporting 2400, 4800, 9600, 19200, 38400, 57600, 115200, etc. 7 baud rates.

The number of variables that ModbusRTU can access is defined as follows:

- Read coil (0x01), number of coils 1–2000
- Read discrete coils (0x02), number of coils 1–2000 (0x7D0)
- Read holding register (0x03), number of registers 1–125 (0x7D)
- Read input register (0x04), number of registers 1–125 (0x7D)
- Write a single coil (0x05)
- Write a single register (0x06)
- Write multiple coils (0x0F), number of coils 1–1968 (0x7B0)
- Write multiple register (0x10), number of register 1–120 (0x78)

3.2.1 ModbusRTU_Master

Create an application project for the ModbusRTU_Master. There are two serial ports in AX series programmable controllers. To add ModbusRTU_Master module, the corresponding library files "ModbusRTU_Master1_x.x.x.x.library" and "ModbusRTU_Master 2_x.x.x.x.library" are needed (ModbusRTU_Master1_x.x.x.x.library for the hardware COM1_RS485 port and ModbusRTU_Master2_x.x.x.x.library for the hardware COM2_RS485 port).

3.2.2 ModbusRTU_Slave

Create an application project for the ModbusRTU_Slave. There are two serial ports in AX series programmable controllers. To add ModbusRTU_Slave module, the corresponding library files "ModbusRTU_Slave1_x.x.x.x.library" and "ModbusRTU_Slave2_x.x.x.x.library" are needed (ModbusRTU_Slave1_x.x.x.x.library for the hardware COM1_RS485 port and ModbusRTU_Slave2_x.x.x.x.library for the hardware COM2_RS485 port).

The ModbusRTU_Slave defines the storage area that can be accessed from outside. The detailed area is as follows:

Table 3-3 ModbusRTU_Slave function code

Function code of RTU master node	Address name	Range	Offset
01	%QX	0.0-511.7	N/A
05	%QX	0.0-511.7	N/A
02	%IX	0.0-511.7	N/A
04	%IW	0-511	N/A
03/06	%MW	0-8192	5000
03/06	%QW	0-511	N/A
01	%MX	0.0-7565.7	5000
05	%MX	0.0-7565.7	5000

3.3 EtherCAT master node

For the parameter configuration of the EtherCAT master node, please refer to the relevant instruction in Invtmatic Studio help documents. Here is an example of the connection between an EtherCAT master and a DA200 servo drive slave for reference.

1. Creating the DA200 servo application project

Add the library file "INVT_DA200_xxx.devdesc.xml" required for this module. Take INVT_DA200_262 as an example.

Note:

- The highest priority 0 is recommended for the creation of EtherCAT Master SoftMotion projects.
- It is recommended that the synchronization period and the task period be set consistently at 4ms or more.
- Create EtherCAT Master SoftMotion through a separate task. Separate the EtherCAT Master SoftMotion tasks from I/O, analog input/output, Modbus communication and other tasks.

2. Select the motion controller device profile in the device tree, right-click it and add the EtherCAT Master SoftMotion as shown in the following figure.

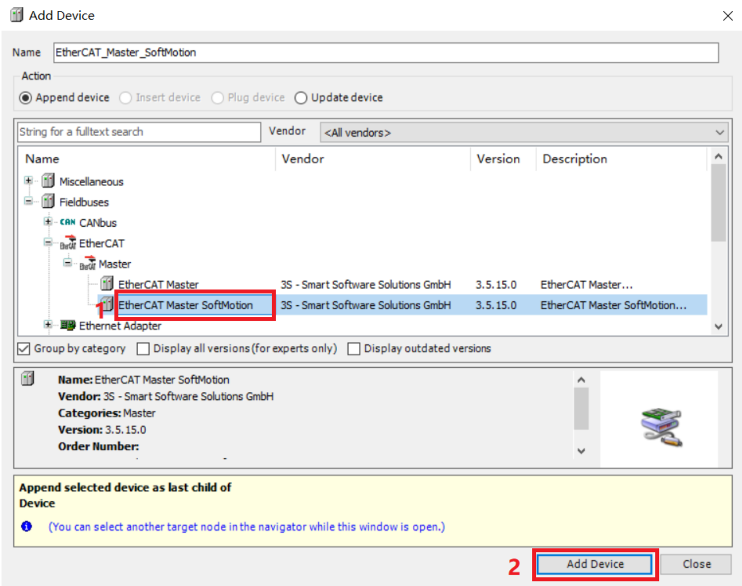


Figure 3-1 Add the EtherCAT motion control master

3. Select EtherCAT_Master_SoftMotion in the device tree, right-click it and add INVT DA200 servo drive as shown in the following figure.

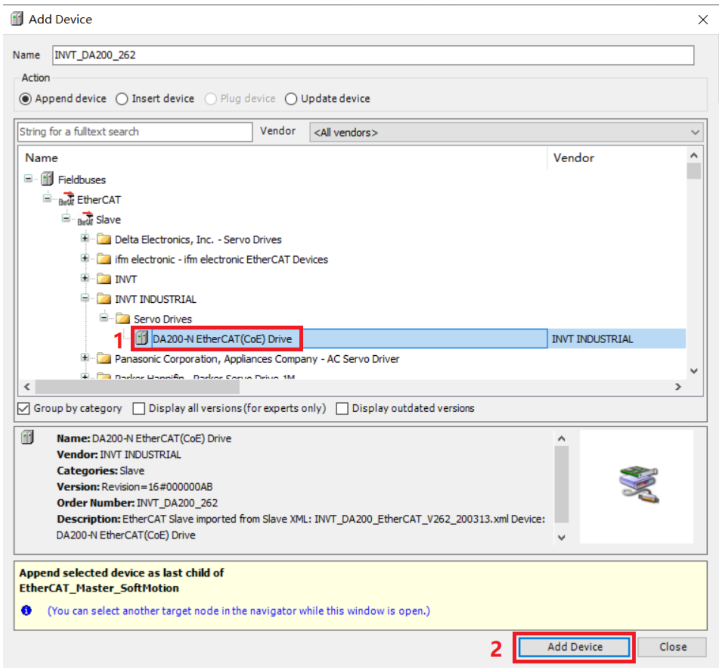


Figure 3-2 Add the DA200 servo drive

4. Select the INVT_DA200_262 in the device tree, right-click it and add the motor axis (select **SoftMotion's CiA 402 axis**). Add the call program as shown in the following figure.

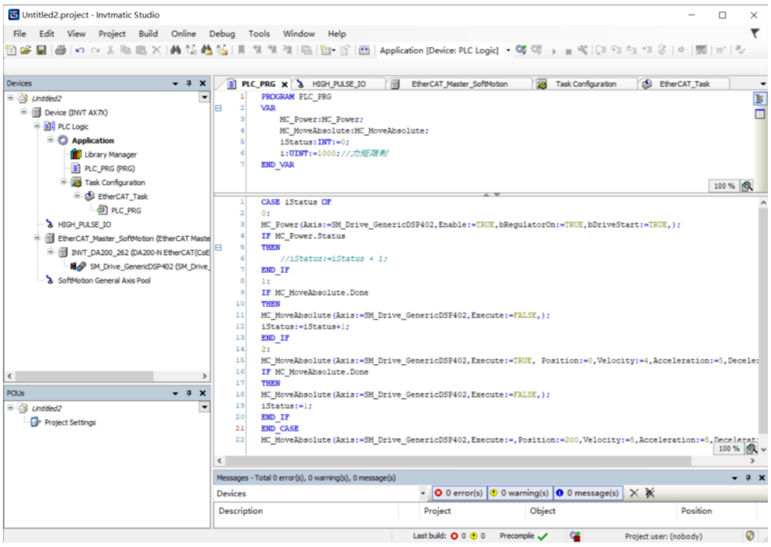


Figure 3-3 DA200 servo drive application example

3.4 CANopen

CANopen is a high-level communication protocol that is based on the CAN (Controller Area Network) protocol, including communication profile and device profile.

The communication model defines four types of messages (communication objects).

- **Management message**

Layer management, network management and ID assignment services: such as initialization, configuration and network management (including: node protection).

The services and protocols conform to the LMT, NMT and DBT services sections of the CAL. These services are based on the master-slave communication mode, which means there can only be one LMT, NMT or DBT master node and one or more slave nodes in a CAN network.

- **Service Data Object (SDO)**

By using indexes and sub-indexes (in the first few bytes of a CAN message), the SDO enables clients to access items (objects) in the device (server) object dictionary.

SDO is implemented through a multi-domain CMS object in CAL that allows the transfer of data of any length. The data will be split into several messages when it exceeds 4 bytes.

The protocol confirms the service type: generating an answer for each message (two IDs are required for an SDO). SDO request and answer messages always contain 8 bytes (meaningless data lengths are indicated in the first byte which carries the protocol information). SDO communication has many protocols.

- **Process Data Object (PDO)**

PDO is used to transfer real-time data from a creator to one or more recipient s. Data transfer is limited to 1 to 8 bytes (for example, one PDO can transfer up to 64 digital I/O values, or 4 16-bit AD values).

PDO communication has no protocol defined. PDO data content is defined only by its CAN ID, assuming that the creator and recipient s know the data content of the PDO.

Each PDO is described by two objects in the object dictionary:

1. PDO communication parameters: determine which COB-ID will be used by the PDO, transmission type, prohibition time, and timer period.

2. PDO mapping parameter: a list of objects in the object dictionary that are mapped to the PDO, including their data

lengths (in bits). The creator and recipients must know this mapping to interpret PDO content.

PDO message content is predefined (or configured at network startup).

Mapping application objects to the PDO is described in the device object dictionary. If the device (creator and recipients) supports variable PDO mappings, the PDO mapping parameters can be configured using SDO messages.

PDO can be delivered in the following modes:

1. Synchronization (by receiving SYNC objects)

Aperiodic: The transmission is pre-triggered by a remote frame or by an object-specific event defined in the device profile.

Periodic: The transmission is triggered after every 1 to 240 SYNC messages.

2. Asynchronization

The transmission is triggered by a remote frame or by an object-specific event defined in the device profile.

- **Predefined messages or special function objects:**

SYNC

Time Stamp

Emergency

Node guarding

3.4.1 CANopen master node configuration

3.4.1.1 Master node usage process

- Install the CANopen slave devices.

The associated CANopen slave device profile must first be installed into the system. The device profile can be a *.Devdesc.xml file or an EDS (Electronic Data Sheet) file for the manufacturer.

- Add CANbus to the device tree.

The base node of CANopen (the uppermost entry in the CANbus configuration tree) must be a CANbus object. A CANbus can be inserted underneath the device node of the AX series programmable controller. The device tree structure after adding a CANbus is shown in the following diagram.

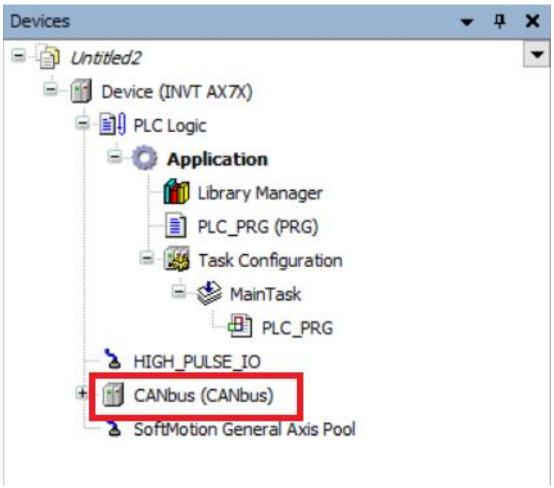


Figure 3-4 Device tree structure with a CANbus

3.4.1.2 Adding CANopen management device

Under the CANbus, add a **CANopen Management** device, which can be used as a CANopen master. The device tree structure after adding the device is shown in the following diagram.

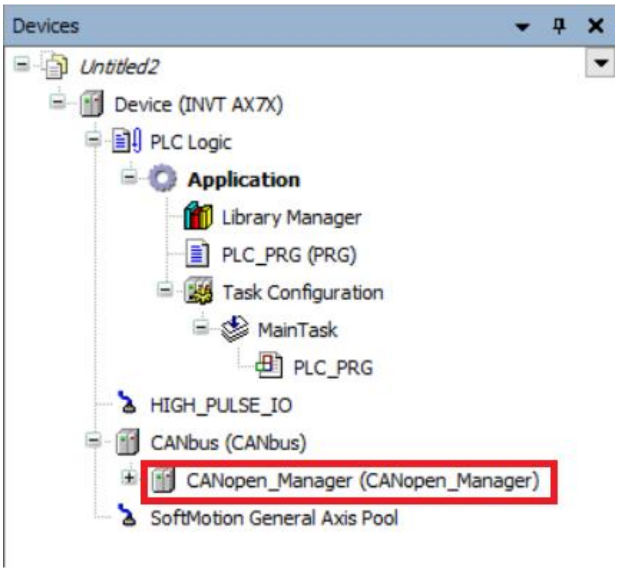


Figure 3-5 Device tree structure with a CANopen master

3.4.1.3 Adding CANopen slave node

Take our DA200 CANopen slave as an example. Add the DA200 slave device under CANopen Manager after adding the EDS file of this slave, as shown in the following diagram.

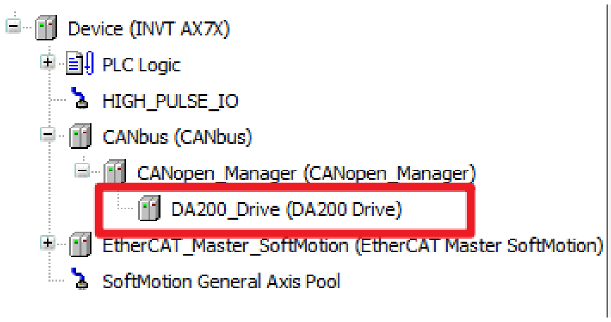


Figure 3-6 Device tree structure with a CANopen slave

The software configuration of the CANopen master is complete.

3.4.2 Parameter configuration of CANopen master

Configure **Network** and **Baud Rate** of the CANbus first.

Network: the number of CAN networks connected via the CANbus, range: 0–100.

Baud Rate: the baud rate used for transmission on the bus, the following baud rates can be set: 10kbits/s, 20kbits/s, 50kbits/s, 100kbits/s, 125kbits/s, 250kbits/s, 500kbits/s, 800kbits/s, and 1000kbits/s.

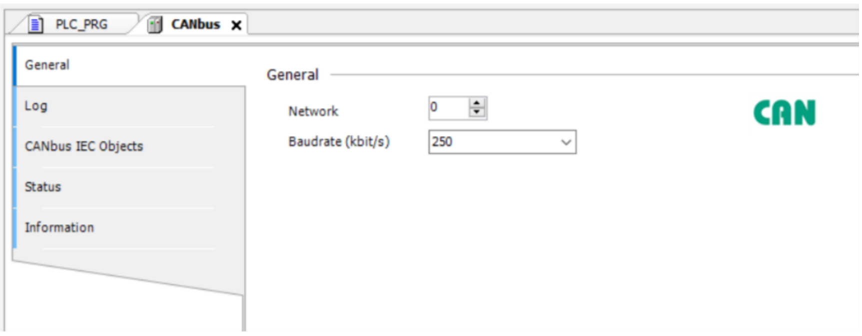


Figure 3-7 Parameter configuration of CANbus

CANopen Management is a node under the CANbus node that supports CANbus configuration through internal functions.

It is generally used as the CANbus master. The configuration page is shown in the following figure.

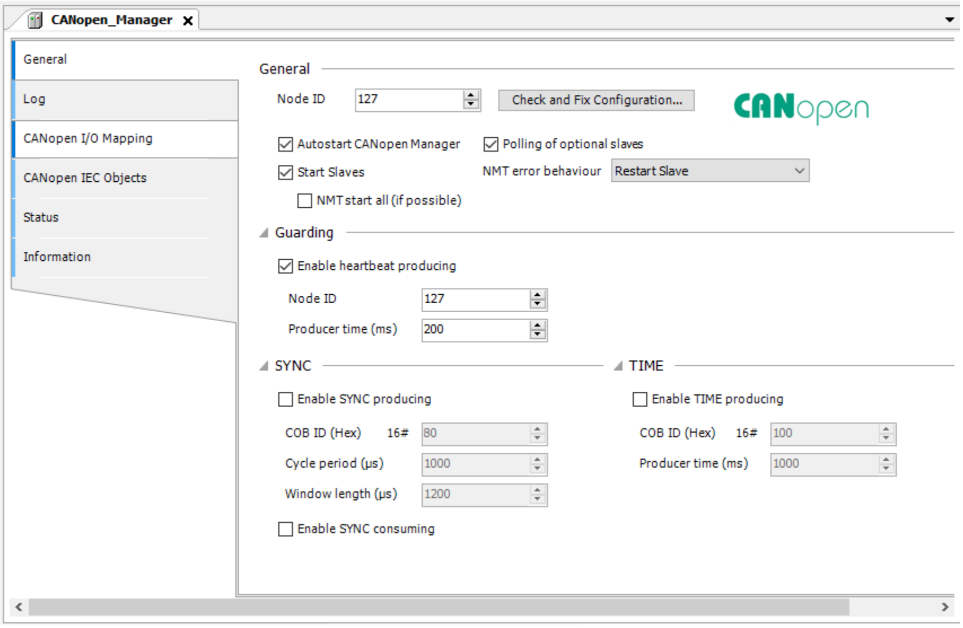


Figure 3-8 Parameter configuration of CANopen master

Node ID: Provides an array pair module that CANopen Manager can correspond to one-to-one, with ID values of 1-127 (must be a decimal integer).

Guarding: Heartbeat mode is a traditional protection mechanism that can be handled by the master station and the slave station modules, different form node protection. Normally the master is configured to send a heartbeat to the slave.

Enable heartbeat producing: If this option is enabled, the master will send heartbeats continuously according to an internally defined heartbeat time. If a new slave heartbeat function is added, their heartbeat actions will be automatically activated and configured, i.e. the node ID is automatically set in the management configuration and the heartbeat interval is automatically multiplied by a factor of 1 and 2. If this option is disabled, the node protection (with a life time factor of 10 and a protection time of 100ms) is activated in the slave.

Node ID: Unique identifier of heartbeat generation (1–127) on the bus.

Producer time (ms): Defines the internal heartbeat time in milliseconds.

4 Module Configuration

4.1 CPU module

Please follow the steps to configure the AX series programmable controller real time and IP address.

1. Create a controller Cfg project.

Add the library file **CmpPlcCfg_x.x.x.x.library** required for this module to create a standard project.

2. Define and use variables.

Table 4-1 Variable definition

Variable	Type		Function	Remarks
setEnable	INPUT	BOOL	Time setting function	0: Disabled 1: Enabled
getEnable		BOOL	Time reading function	0: Disabled 1: Enabled
inTime		ARRAY OF UINT	Time to be entered in format: hour minute second	E.g. 14 48 56
inDate		ARRAY OF UINT	Date to be entered in format: year month day	E.g. 2018 12 26
r_Enable		BOOL	IP reading function	0: Disabled 1: Enabled
w_Enable		BOOL	IP setting function	0: Disabled 1: Enabled
new_IP		STRING	Set a new IP	E.g. 192.168.1.16
new_netmask		STRING	Set a new subnet mask	E.g. 255. 255. 255.0
setDone	OUTPUT	BOOL	Completion mark of time setting	0: The execution of commands is in progress. 1: The execution of commands is completed.
getDone		BOOL	Completion mark of time obtaining	0: The execution of commands is in progress. 1: The execution of commands is completed.
setError		INT	Configuration error sign	See Controller Cfg error code table
getError		INT	Get error sign	See Controller Cfg error code table
outTime		ARRAY OF UINT	Read the native hour, minute and second information.	E.g. 14 48 56
outDate		ARRAY OF UINT	Read the native year, month and day information.	E.g. 2018 12 26
Done		BOOL	Completion mark	0: The execution of commands is in progress. 1: The execution of commands is completed.
read_IP		STRING	IP read	E.g. 192.168.1.16
read_netmask		STRING	Subnet mask read	E.g. 255. 255. 255.0

Table 4-2 Local time configuration

Variable	Function	Remarks
setEnable	Time setting function	0: Disabled 1: Enabled
getEnable	Time reading function	0: Disabled 1: Enabled
inDate	Date to be entered in format: year month day	E.g. 2018 12 16
inTime	Time to be entered in format: hour minute second	E.g. 14 48 56

According to the time array in format inTime and inDate, where inTime[0] is hour, inTime[1] is minute, inTime[2] is second, inDate[0] is year, inDate[1] is month, inDate[2] is day, enter the time (all inputs are required). After the settings, enable setEnable to set the above time to current time of the AX series programmable controller CPU.

Enable getEnable to get the real time of the AX series programmable controller CPU, which is displayed in outTime and outDate arrays.

Table 4-3 Local IP configuration

Variable	Function	Remarks
r_Enable	IP reading function	0: Disabled 1: Enabled
w_Enable	IP setting function	0: Disabled 1: Enabled
new_IP	Set a new IP	E.g. 192.168.1.16
new_netmask	Set a new subnet mask	E.g. 255. 255. 255.0

Enter the IP and subnet mask in the required format, and then enable w_Enable to set the above IP and subnet mask to the current IP and subnet mask of AX series programmable controller EtherNET network port after entering the setup.

Note: The USB virtual network port is independent of the EtherNET network port, and the IP and subnet mask modified by CmpPlcCfg_x.x.x.x.library is still the IP and subnet mask of the EtherNET network port when the device is connected with a USB. After the IP or subnet mask modification, it will take some time for the AX series CPU to connect to Invtmatic Studio on the PC.

Enable r_Enable to get the IP address and subnet mask of the controller EtherNET network port, which are displayed in the read_IP and read_netmask strings respectively.

4.2 High speed I/O module

4.2.1 Creating high speed I/O module project

Create the high speed I/O module application and add the corresponding library file. Then complete the corresponding variable configuration in HIGH_PULSE_IO device.

HSIO stands for High Speed Input and Output. HSIO can be used for high speed counting and high speed pulse output with three interrupt functions that can be configured as needed. HSIO contains the device profile Shenzen INVT-AX7X-CPU_x.x.x.x.devdesc, the high speed counting function block library CmpHSIO_C.library and the motion control function block library CmpHSIO_M.library or CmpIMC_P.library.

The HSIO device profile is used to configure various functions of the high-speed IO, including input/output port function, counter, high-speed pulse output, filter parameters, and interruption.

The high-speed counting function block library CmpHSIO_C.library contains several function blocks, such as counter setting, count value reading, latching, preset value, pulse width measurement, timing sampling, and count value comparison. These function blocks can be called to complete the application needed for counting.

The motion control function block library CmpHSIO_M.library is described in detail via dedicated instructions.

AX7☐-C-1608P, hereinafter referred to as the P-type model, and AX7☐-C-1608N, hereinafter referred to as the N model. The software of the P model and the N model are the same but the hardware ports are different.

4.2.1.1 P-type model port configuration description

At present, AX7☐-C-1608P programmable controller integrates 16-channel high-speed pulse input (The first 6 channels support 24V single-ended input or differential input, and the last 10 channels support 24V single-ended input) and 8-channel pulse output which supports pulse+direction mode, FWD/REV pulse mode and quadrature pulse mode, and each port can be configured with different functions. The configuration table is shown as follows.

Input port	Common input function (default)	Counting function	Trigger latching and Z-signal function	Positive and negative limit zero function	Pulse width measure-ment function	**Output port**	Common input function (default)	High speed pulse output function	Compare Output Function
	Function value is 0	Function value is 1	Function value is 2	Function value is 3	Function value is 4		Function value is 0	Function value is 1	Function value is 2
X0 (In0)	Common input	C0A		CH0N		Y0 (Out0)	Common output	CH0CW/PULS0	CMP0
X1 (In1)	Common input	C0B		CH1N		Y1 (Out1)	Common output	CH0CCW/SIGN0	CMP1
X2 (In2)	Common input	C1A		CH2N		Y2 (Out2)	Common output	CH1CW/PULS1	CMP2
X3 (In3)	Common input	C1B		CH3N		Y3 (Out3)	Common output	CH1CCW/SIGN1	CMP3
X4 (In4)	Common input	C4A	C0Z	CH0P		Y4 (Out4)	Common output	CH2CW/PULS2	CMP4
X5 (In5)	Common input	C4B	C1Z	CH1P		Y5 (Out5)	Common output	CH2CCW/SIGN2	CMP5
X6 (In6)	Common input	C5A	C2Z	CH2P		Y6 (Out6)	Common output	CH3CW/PULS3	CMP6
X7 (In7)	Common input	C5B	C3Z	CH3P		Y7 (Out7)	Common output	CH3CCW/SIGN3	CMP7
X8 (In8)	Common input	C2A	C0T		PWC0				
X9 (In9)	Common input	C2B	C1T		PWC1				
XA (InA)	Common input	C3A	C2T		PWC2				
XB (InB)	Common input	C3B	C3T		PWC3				
XC (InC)	Common input	C6A		CH0Z					

Input port	Common input function (default)	Counting function	Trigger latching and Z-signal function	Positive and negative limit zero function	Pulse width measure -ment function	Output port	Common input function (default)	High speed pulse output function	Compare Output Function
	Function value is 0	Function value is 1	Function value is 2	Function value is 3	Function value is 4		Function value is 0	Function value is 1	Function value is 2
XD (InD)	Common input	C6B		CH1Z					
XE (InE)	Common input	C7A		CH2Z					
XF (InF)	Common input	C7B		CH3Z					

Note:

- X0-XF is the input port and Y0-Y7 is the output port.
- Common input and common output mean a common I/O signal, usually a switching signal.
- CxA, CxB, and CxZ are signals of encoder A, B, and Z respectively.
- CxT refers to the trigger and latch function channel and supports 4 channels, C0T–C3T.
- CHxP and CHxN refer to positive and negative limit signals, with N being the negative direction and P being the positive direction. CHxZ refers to the zero signal.
- PWCx means pulse width check signal.
- CHxCW is a clockwise signal and CHxCCW is a counterclockwise signal.
- PULSx means pulse.
- SIGNx means the direction of the pulse.
- CMPx means the output comparison.

4.2.1.2 N-type model port configuration description

At present, AX7☐-C-1608N programmable controller integrates 16-channel high-speed pulse input (The first 4 channels support differential input, and the last 12 channels support 24V single-ended input) and 8-channel high-speed pulse output which supports pulse+direction mode, FWD/REV pulse mode and quadrature pulse mode, and each port can be configured with different functions. The configuration table is shown as follows.

Input port	Common input function (default)	Counting function	Trigger latching and Z-signal function	Positive and negative limit zero function	Pulse width measure -ment function	Output port	Common input function (default)	High speed pulse output function	Compare Output Function
	Function value is 0	Function value is 1	Function value is 2	Function value is 3	Function value is 4		Function value is 0	Function value is 1	Function value is 2
A0 (In0)	Common input	C0A		CH0N		Y0 (Out0)	Common output	CH0CW/PULS0	CMP0
B0 (In1)	Common input	C0B		CH1N		Y1 (Out1)	Common output	CH0CCW/SIGN0	CMP1
A1 (In2)	Common input	C1A		CH2N		Y2 (Out2)	Common output	CH1CW/PULS1	CMP2

Input port	Common input function (default)	Counting function	Trigger latching and Z-signal function	Positive and negative limit zero function	Pulse width measure-ment function	Output port	Common input function (default)	High speed pulse output function	Compare Output Function
	Function value is 0	Function value is 1	Function value is 2	Function value is 3	Function value is 4		Function value is 0	Function value is 1	Function value is 2
B1 (In3)	Common input	C1B		CH3N		Y3 (Out3)	Common output	CH1CCW/SIGN1	CMP3
X4 (In4)	Common input	C4A	C0Z	CH0P		Y4 (Out4)	Common output	CH2CW/PULS2	CMP4
X5 (In5)	Common input	C4B	C1Z	CH1P		Y5 (Out5)	Common output	CH2CCW/SIGN2	CMP5
X6 (In6)	Common input	C5A	C2Z	CH2P		Y6 (Out6)	Common output	CH3CW/PULS3	CMP6
X7 (In7)	Common input	C5B	C3Z	CH3P		Y7 (Out7)	Common output	CH3CCW/SIGN3	CMP7
X8 (In8)	Common input	C2A	C0T		PWC0				
X9 (In9)	Common input	C2B	C1T		PWC1				
X10 (InA)	Common input	C3A	C2T		PWC2				
X11 (InB)	Common input	C3B	C3T		PWC3				
X12 (InC)	Common input	C6A		CH0Z					
X13 (InD)	Common input	C6B		CH1Z					
X14 (InE)	Common input	C7A		CH2Z					
X15 (InF)	Common input	C7B		CH3Z					

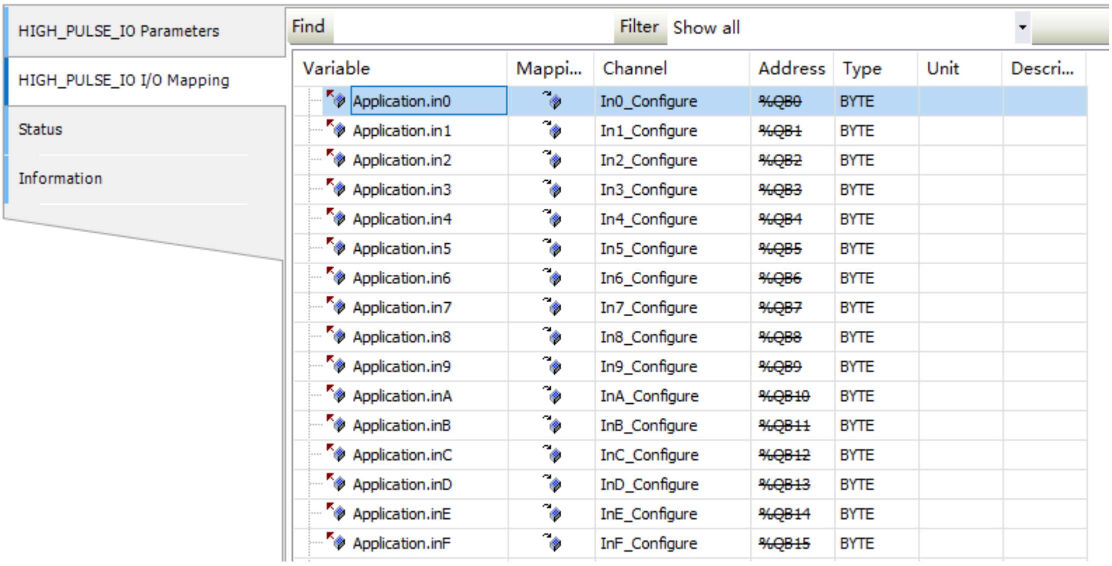
Note:

- A0/B0/A1/B1/X4-X15 is the input port and Y0-Y7 is the output port.
- Common input and common output mean a common I/O signal, usually a switching signal.
- A0/B0/A1/B1 is recommended not to be a normal input port. In special cases, if it is used as a normal input port, a 2K resistor needs to be connected in series in the circuit, otherwise the point will be burned out.
- CxA, CxB, and CxZ are signals of encoder A, B, and Z respectively.
- CxT refers to the trigger and latch function channel and supports 4 channels, C0T–C3T.
- CHxP and CHxN refer to positive and negative limit signals, with N being the negative direction and P being the positive direction. CHxZ refers to the zero signal.
- PWCx means pulse width check signal.
- CHxCW is a clockwise signal and CHxCCW is a counterclockwise signal.
- PULSx means pulse.
- SIGNx means the direction of the pulse.

- CMPx means the output comparison.

4.2.2 Input port function description

The input port can be set to five functions, which are: common input function, counting function, triggering latch and Z-signal function, positive and negative limit zero function, and pulse width measurement function. Here is the mapping table of configuration input function corresponding to Inx_Configure parameters, where x ranges from 0 to F.



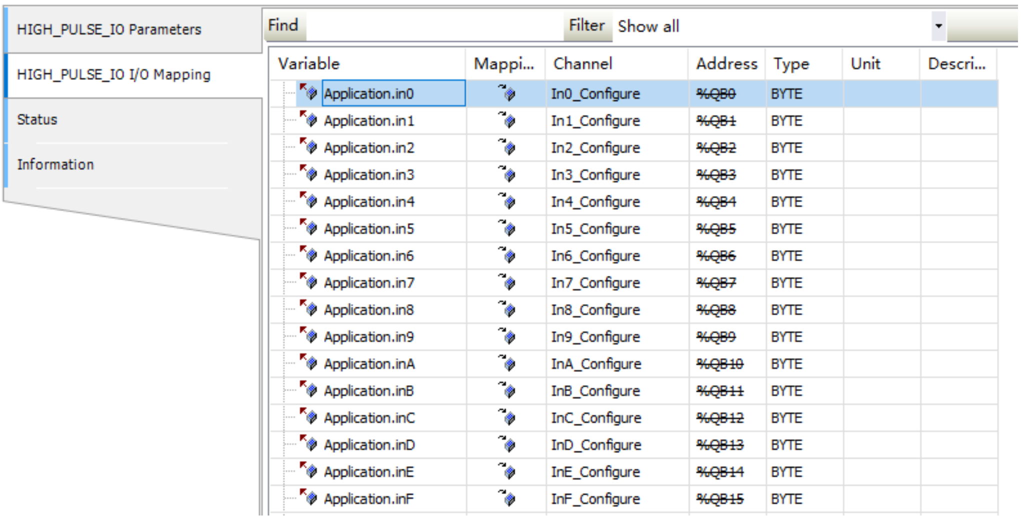
HIGH_PULSE_IO Parameters
HIGH_PULSE_IO I/O Mapping
Status
Information

Find | Filter Show all

Variable	Mappi...	Channel	Address	Type	Unit	Descri...
Application.in0		In0_Configure	%QB0	BYTE		
Application.in1		In1_Configure	%QB1	BYTE		
Application.in2		In2_Configure	%QB2	BYTE		
Application.in3		In3_Configure	%QB3	BYTE		
Application.in4		In4_Configure	%QB4	BYTE		
Application.in5		In5_Configure	%QB5	BYTE		
Application.in6		In6_Configure	%QB6	BYTE		
Application.in7		In7_Configure	%QB7	BYTE		
Application.in8		In8_Configure	%QB8	BYTE		
Application.in9		In9_Configure	%QB9	BYTE		
Application.inA		InA_Configure	%QB10	BYTE		
Application.inB		InB_Configure	%QB11	BYTE		
Application.inC		InC_Configure	%QB12	BYTE		
Application.inD		InD_Configure	%QB13	BYTE		
Application.inE		InE_Configure	%QB14	BYTE		
Application.inF		InF_Configure	%QB15	BYTE		

For P-type model, the In0_Configure–InF_Configure port function configuration parameters correspond to ports X0–XF in turn.

For N-type model, the In0_Configure–InF_Configure port function configuration parameters correspond to ports A0/B0/A1/B1/X4–X15 in turn.



HIGH_PULSE_IO Parameters
HIGH_PULSE_IO I/O Mapping
Status
Information

Find | Filter Show all

Variable	Mappi...	Channel	Address	Type	Unit	Descri...
Application.in0		In0_Configure	%QB0	BYTE		
Application.in1		In1_Configure	%QB1	BYTE		
Application.in2		In2_Configure	%QB2	BYTE		
Application.in3		In3_Configure	%QB3	BYTE		
Application.in4		In4_Configure	%QB4	BYTE		
Application.in5		In5_Configure	%QB5	BYTE		
Application.in6		In6_Configure	%QB6	BYTE		
Application.in7		In7_Configure	%QB7	BYTE		
Application.in8		In8_Configure	%QB8	BYTE		
Application.in9		In9_Configure	%QB9	BYTE		
Application.inA		InA_Configure	%QB10	BYTE		
Application.inB		InB_Configure	%QB11	BYTE		
Application.inC		InC_Configure	%QB12	BYTE		
Application.inD		InD_Configure	%QB13	BYTE		
Application.inE		InE_Configure	%QB14	BYTE		
Application.inF		InF_Configure	%QB15	BYTE		

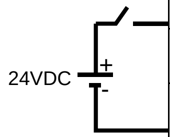
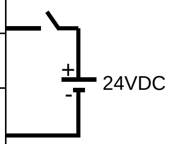
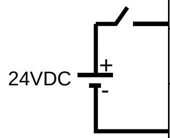
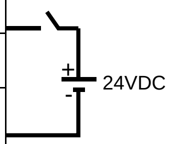
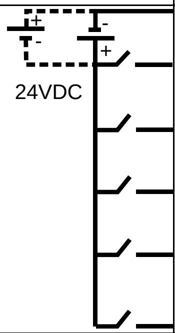
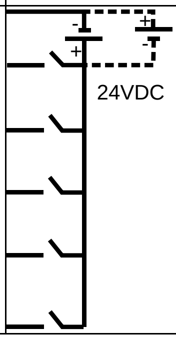
4.2.2.1 Common input function

If the function value is 0, the signal port is configured to be used as a common input port.

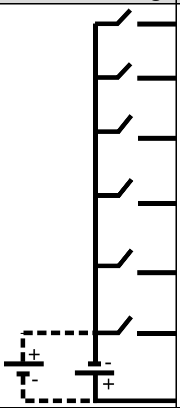
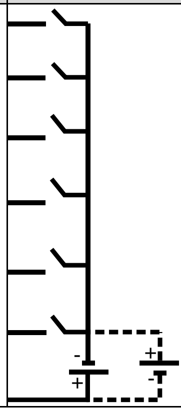
Wiring of common input ports

Common input: P-type model							
External wiring	**Port**	**Function**	**CN5 terminal No.**		**Function**	**Port**	**External wiring**
24VDC	X0	Common input	40	39	Common input	X1	24VDC
			38	37			
	COM	Input	36	35	Input	COM	

Common input: P-type model

External wiring	Port	Function	CN5 terminal No.		Function	Port	External wiring
		common port			common port		
	X2	Common input	34	33	Common input	X3	
			32	31			
	COM	Input common port	30	29	Input common port	COM	
	X4	Common input	28	27	Common input	X5	
			26	25			
	COM	Input common port	24	23	Input common port	COM	
	SS1	Input common port	22	21	Input common port	SS2	

Common input: N-type model

External wiring	Port	Function	CN5 terminal No.		Function	Port	External wiring
	X4	Common input	X4	X5	Common input	X5	

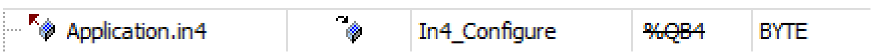
Configuration of common input ports

Define the variables to configure the ports and map them to the high speed pulse mapping table.

Configuration routine:

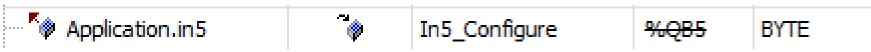
1: Configure X4 port of P-type model and X4 port of N-type model as common input ports.

in4:=0;



2: Configure X5 port of P-type model and X5 port of N-type model as common input ports.

in5:=0;

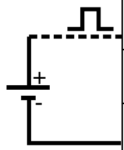
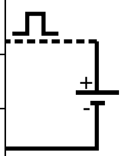
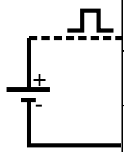
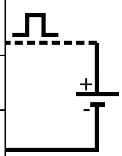
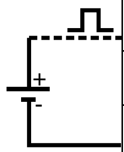
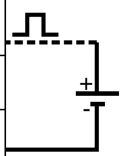
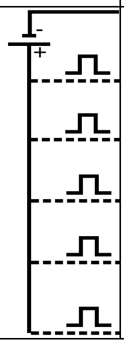
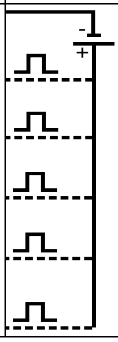


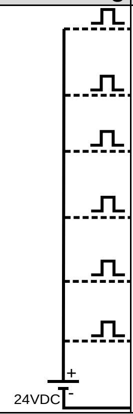
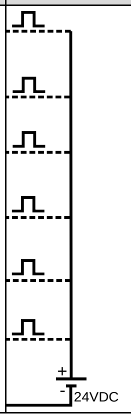
4.2.2.2 Counting function

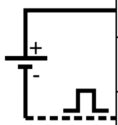
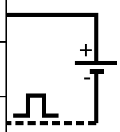
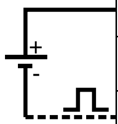
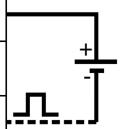
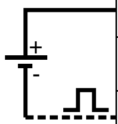
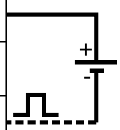
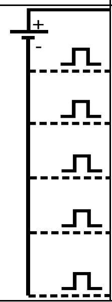
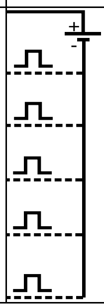
If the function value is 1, the signal port is configured as a counter function and all 16 input ports can be used as counter inputs.

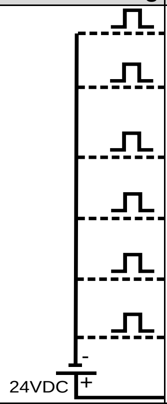
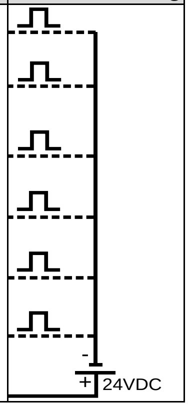
Counting function module can count and calculate the input pulse, and detect the position, speed and frequency. The maximum frequency of input pulse is 200kHz.

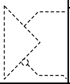
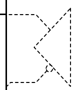
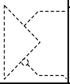
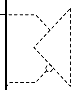
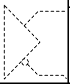
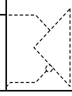
Wiring of counting function ports

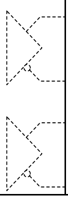
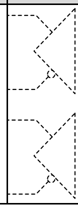
Counting function (Single-end source): P-type model							
External wiring	**Port**	**Function**	**CN5 terminal No.**		**Function**	**Port**	**External wiring**
	C0A	Phase A pulse input	40	39	Phase B pulse input	C0B	
			38	37			
	COM	Input common port	36	35	Input common port	COM	
	C1A	Phase A pulse input	34	33	Phase B pulse input	C1B	
			32	31			
	COM	Input common port	30	29	Input common port	COM	
	C4A	Phase A pulse input	28	27	Phase B pulse input	C4B	
			26	25			
	COM	Input common port	24	23	Input common port	COM	
	SS1	Input common port	22	21	Input common port	SS2	

Counting function (Single-end source): N-type model							
External wiring	Port	Function	CN5 terminal No.		Function	Port	External wiring
	C4A	Phase A pulse input	X4	X5	Phase B pulse input	X5	

Counting function (Single-end sink): P-type model							
External wiring	Port	Function	CN5 terminal No.		Function	Port	External wiring
	COM	Input common port	40	39	Input common port	COM	
			38	37			
	C0A	Phase A pulse input	36	35	Phase B pulse input	C0B	
	COM	Input common port	34	33	Input common port	COM	
			32	31			
	C1A	Phase A pulse input	30	29	Phase B pulse input	C1B	
	COM	Input common port	28	27	Input common port	COM	
			26	25			
	C4A	Phase A pulse input	24	23	Phase B pulse input	C4B	
	SS1	Input common port	22	21	Input common port	SS2	

Counting function (Single-end sink): N-type model							
External wiring	Port	Function	CN5 terminal No.		Function	Port	External wiring
	C4A	Phase A pulse input	X4	X5	Phase B pulse input	C4B	

Counting function (differential signal): P-type model							
External wiring	Port	Function	CN5 terminal No.		Function	Port	External wiring
			40	39			
	C0A+	Phase A differential +	38	37	Phase B differential +	C0B+	
	C0A-	Phase A differential -	36	35	Phase B differential -	C0B-	
			34	33			
	C1A+	Phase A differential +	32	31	Phase B differential +	C1B+	
	C1A-	Phase A differential -	30	29	Phase B differential -	C1B-	
			28	27			
	C4A+	Phase A differential +	26	25	Phase B differential +	C4B+	
	C4A-	Phase A differential -	24	23	Phase B differential -	C4B-	

Counting function (differential signal): N-type model							
External wiring	Port	Function	CN5 terminal No.		Function	Port	External wiring
	C4A	Phase A pulse input	X4	X5	Phase B pulse input	C4B	

Configuration of counting ports

Function value configuration:

Define the variables to configure the ports with data type BYTE, and map them to the high-speed pulse mapping table.

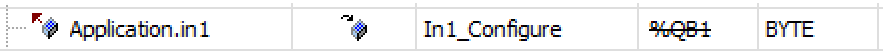
Configuration routine:

1) Configure X0 port of P-type model and A0 port of N-type model as counting ports.

in0:=1;

2) Configure X1 port of P-type model and B0 port of N-type model as counting ports.

in1:=1;



Configure other ports by analogy.

4.2.2.3 Trigger, latch and Z-signal function

If the function value is 2, the signal port is configured as trigger, latch and Z-signal functions.

The trigger function can preset count value for the counter and the rising edge of the trigger signal is valid. The preset value will be written to the counter once the signal is valid. Normally there are three ways to write the preset value of the counter: software writing, external trigger writing, and consistent comparison trigger writing. This product uses external trigger writing.

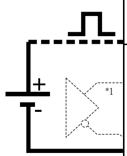
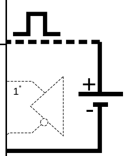
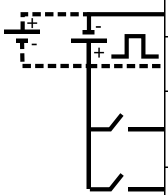
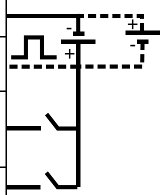
The latch function can lock the counter value instantly for the upper computer to read.

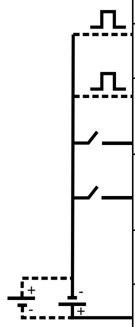
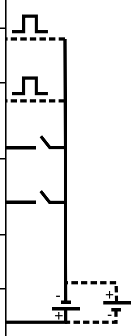
The trigger and latch functions support 4 channels, C0T–C3T (P-type model mapping ports X8, X9, XA, XB and N-type model mapping ports X8, X9, X10, X11).

Z-signal function is used for Z clearing and Z compensation functions and Z-signal encoders generate one pulse per revolution.

Z-signal function supports 4 channels, C0T–C3T (mapping ports X4, X5, X6, X7)

Wiring of trigger, latch and Z-signal ports

Input function 3: (CnT wiring refers to common input; CnZ wiring refers to counting pulse input) P-type model							
External wiring	**Port**	**Function**	**CN5 terminal No.**		**Function**	**Port**	**External wiring**
	C0Z	Phase Z single-ended input	28	27	Phase Z single-ended input	C1Z	
	C0Z+	Phase Z differential input	26	25	Phase Z differential input	C1Z+	
	COM	Input common port	24	23	Input common port	COM	
	SS1	Input common port	22	21	Input common port	SS2	
	C2Z	Z signal input	20	19	Z signal input	C3Z	
	C0T	Probe signal input	18	17	Probe signal input	C1T	
	C2T	Probe signal input	16	15	Probe signal input	C3T	

Input function 3: (CnT wiring refers to common input; CnZ wiring refers to counting pulse input) N-type model							
External wiring	Port	Function	CN5 terminal No.		Function	Port	External wiring
	C0Z	Z signal input	X4	X5	Z signal input	C1Z	
	C2Z	Z signal input	X6	X7	Z signal input	C3Z	
	C0T	Probe signal input	X8	X9	Probe signal input	C1T	
	C2T	Probe signal input	X10	X11	Probe signal input	C3T	
	SS	Input common port	SS	SS	Input common port	SS	

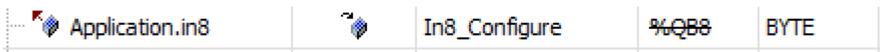
Configuration of the trigger, latch and Z-signal ports

Function value configuration: Define the variables to configure the ports with data type BYTE, and map them to the high-speed pulse mapping table.

Configuration routine:

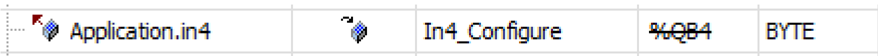
1) Configure X8 port of P-type model and X8 port of N-type model as trigger and latch ports.

in8:=2;



2) Configure X4 port of P-type model and X4 port of N-type model as Z-signal ports

in4:=2;

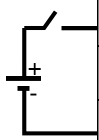
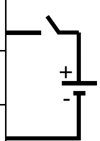
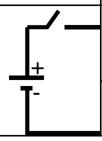
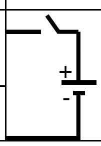


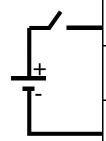
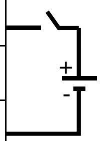
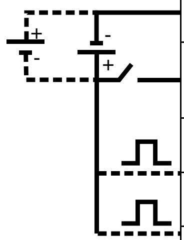
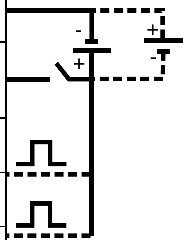
4.2.2.4 Positive and negative limit zero function

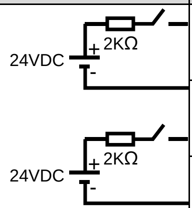
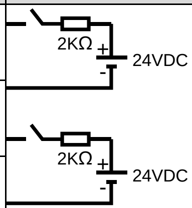
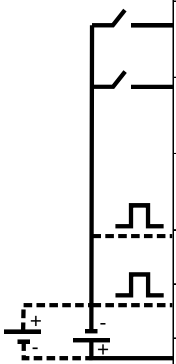
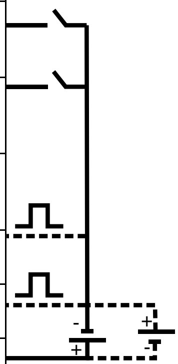
If the function value is 3, the signal port is configured as positive and negative limit zero function.

CHxP/CHxN/CHxZ are positive limit, negative limit and zero signal functions on the x channel, where x ranges from 0 to 3. The positive limit serves to limit the positive direction, where motor movement needs to stop or reverse. The negative limit serves to limit the negative direction, where motor movement needs to stop or reverse.

Wiring of positive and negative limit zero ports

Input function 4: (CHnZ and CHnP wiring refers to common input; CHnZ wiring refers to counting pulse input) P-type model							
External wiring	Port	Function	CN5 terminal No.		Function	Port	External wiring
	CH0N	Negative limit input	40	39	Negative limit input	CH1N	
			38	37			
	COM	Input common port	36	35	Input common port	COM	
	CH2N	Negative limit input	34	33	Negative limit input	CH3N	
			32	31			

	COM	Input common port	30	29	Input common port	COM	
	CH0P	Positive limit input	28	27	Positive limit input	CH1P	
			26	25			
	COM	Input common port	24	23	Input common port	COM	
	SS1	Input common port	22	21	Input common port	SS2	
	CH2P	Positive limit input	20	19	Positive limit input	CH3P	
	CH0Z	Home signal	14	13	Home signal	CH1Z	
	CH2Z	Home signal	12	11	Home signal	CH3Z	

Input function 4: (CHxZ and CHxP wiring refers to common input; CHxZ wiring refers to counting pulse input) N-type model							
External wiring	**Port**	**Function**	**CN5 terminal No.**		**Function**	**Port**	**External wiring**
	CH0N	Negative limit input	A0+	B0+	Negative limit input	CH1N	
	COM	Input common port	A0-	B0-	Input common port	COM	
	CH2N	Negative limit input	A1+	B1+	Negative limit input	CH3N	
	COM	Input common port	A1-	B1-	Input common port	COM	
	CH0P	Positive limit input	X4	X5	Positive limit input	CH1P	
	CH2P	Positive limit input	X6	X7	Positive limit input	CH3P	
			X8	X9			
			X10	X11			
	CH0Z	Home signal	X12	X13	Home signal	CH1Z	
	CH2Z	Home signal	X14	X15	Home signal	CH3Z	
	SS	Input common port	SS	SS	Input common port	SS	

Configuration of positive and negative limit zero ports

Function value configuration:

Define the variables to configure the ports, and map them to the high-speed pulse mapping table.

Configuration routine:

1. Configure X3 port of P-type model and B1 port of N-type model as negative limit ports.

in3:=3;

Application.in3		In3_Configure	%QB3	BYTE

2. Configure XC port of P-type model and X12 port of N-type model as zero ports.

inC:=3;

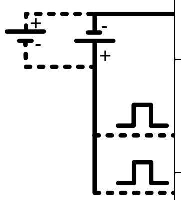
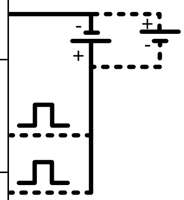
Application.inC		InC_Configure	%QB12	BYTE

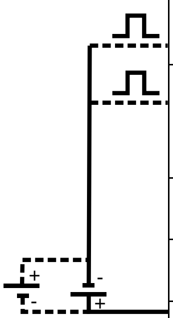
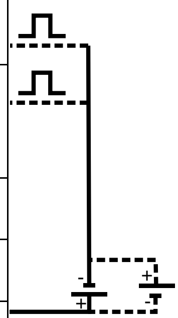
4.2.2.5 Pulse width measurement function

If the function value is 4, the signal port is configured as a pulse width measurement function.

PWCx is a pulse width measurement input channel x, where x ranges from 0 to 3. The ports corresponding to P-type model are X8, X9, XA and XB, and the ports corresponding to N-type model are X8, X9, X10 and X11.

Wiring of pulse measurement ports

Input function 5: (PWCn wiring refers to counting pulse input) P-type model							
External wiring	**Port**	**Function**	**CN5 terminal No.**		**Function**	**Port**	**External wiring**
	SS1	Input common port	22	21	Input common port	SS2	
	PWC0	Pulse measurement signal	18	17	Pulse measurement signal	PWC1	
	PWC2	Pulse measurement signal	16	15	Pulse measurement signal	PWC3	

Input function 5: (PWCn wiring refers to counting pulse input) N-type model							
External wiring	**Port**	**Function**	**CN5 terminal No.**		**Function**	**Port**	**External wiring**
	PWC0	Pulse measurement signal	X8	X9	Pulse measurement signal	PWC1	
	PWC2	Pulse measurement signal	X10	X11	Pulse measurement signal	PWC3	
			X12	X13			
			X14	X15			
	SS	Input common port	SS	SS	Input common port	SS	

Configuration of pulse width measurement ports

Function value configuration:

Define the variables to configure the ports, and map them to the high-speed pulse mapping table.

Configuration routine:

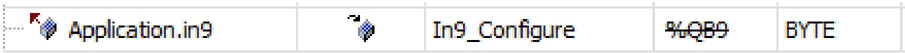
1. Configure X8 port of P-type model and X8 port of N-type model as a pulse width measurement port

in8:=4;

Application.in8		In8_Configure	%QB8	BYTE

2. Configure X9 port of P-type model and X9 port of N-type model as a pulse width measurement port.

in9:=4;



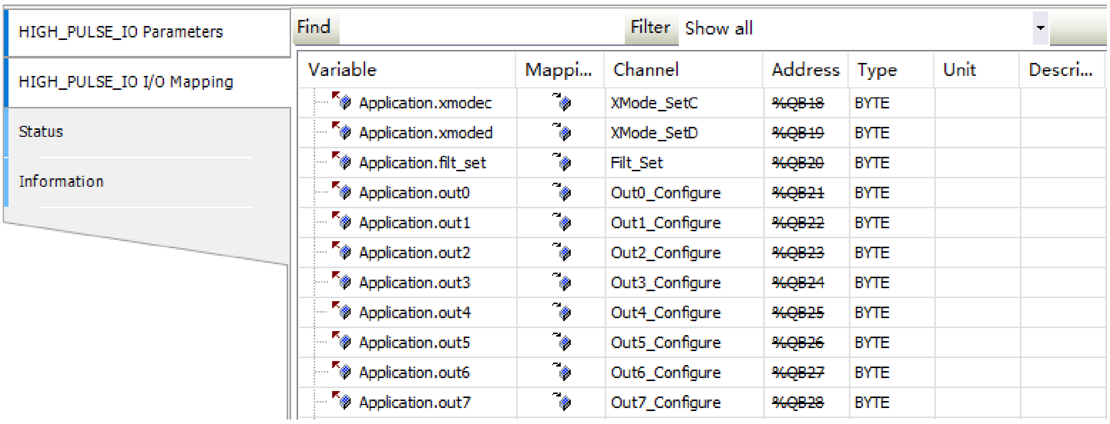
Variable	Mapping	Channel	Address	Type
Application.in9		In9_Configure	%QB9	BYTE

4.2.3 Output port function description

The output port can be set for 3 functions: common output function, high-speed pulse output function and output comparison function.

4.2.3.1 Common output function

If the function value is 0, the signal port is configured to be used as a common output port. The following are the parameters of Outx_Configure in the mapping table of the configuration output function, where the range of x is 0–7.

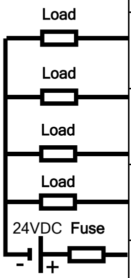
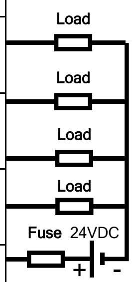


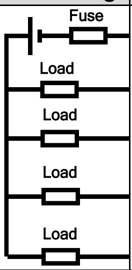
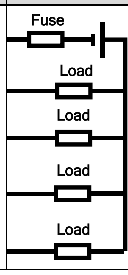
HIGH_PULSE_IO Parameters
HIGH_PULSE_IO I/O Mapping
Status
Information

Find Filter Show all

Variable	Mappi...	Channel	Address	Type	Unit	Descri...
Application.xmodec		XMode_SetC	%QB18	BYTE		
Application.xmoded		XMode_SetD	%QB19	BYTE		
Application.filt_set		Filt_Set	%QB20	BYTE		
Application.out0		Out0_Configure	%QB21	BYTE		
Application.out1		Out1_Configure	%QB22	BYTE		
Application.out2		Out2_Configure	%QB23	BYTE		
Application.out3		Out3_Configure	%QB24	BYTE		
Application.out4		Out4_Configure	%QB25	BYTE		
Application.out5		Out5_Configure	%QB26	BYTE		
Application.out6		Out6_Configure	%QB27	BYTE		
Application.out7		Out7_Configure	%QB28	BYTE		

Wiring of common output ports

Common port: P-type model							
External wiring	Port	Function	CN5 terminal No.		Function	Port	External wiring
	Y0	Common output	10	9	Common output	Y1	
	Y 2	Common output	8	7	Common output	Y3	
	Y4	Common output	6	5	Common output	Y5	
	Y6	Common output	4	3	Common output	Y7	
	COM	Output common port	2	1	Output common port	COM	

Common port: N-type model							
External wiring	Port	Function	CN5 terminal No.		Function	Port	External wiring
	COM	Output common port	COM	COM	Output common port	COM	

The output port of P-type model contains 8 output signals. Only single-ended outputs are supported, and the signal type is source type output. Y0, Y2, Y4 and Y6 share the common port COM1, and Y1, Y3, Y5 and Y7 share the common port

COM2.

The output port of N-type model contains 8 output signals. Only single-ended outputs are supported, and the signal type is sink type output. Y0–Y7 share the common port COM2.

Configuration of common output ports

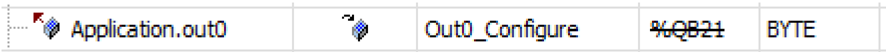
Function value configuration:

Define the variables to configure the ports and map them to the high speed pulse mapping table.

Configuration routine:

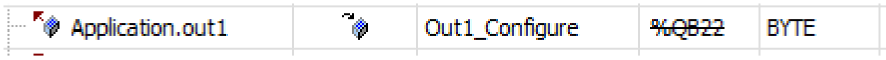
1. Configure Y0 as a common output port.

out0:=0;



2. Configure Y1 as a common output port.

out1:=0;



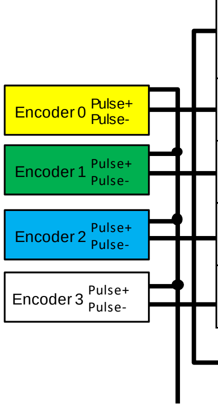
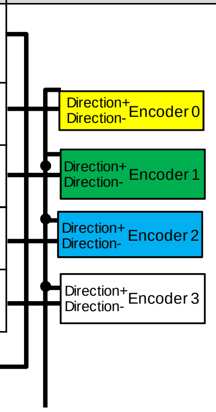
4.2.3.2 High speed pulse output function

If the function value is 1, the signal port is configured as a high-speed pulse output function, and all 8 output ports can be configured for high-speed pulse output.

The high-speed pulse output support pulse + direction, FWD/REV pulse, and quadrature pulse modes.

Wiring of high-speed pulse output ports

Output pulse: P-type model								
External wiring	Port	Function	CN5 terminal No.		Function	Port	External wiring
Encoder0 Pulse + Pulse -	Plus0	Pulse output	10	9	Direction output	Sign0	Direction+ Direction- Encoder0
Encoder1 Pulse+ Pulse-	Plus1	Pulse output	8	7	Direction output	Sign1	Direction+ Direction- Encoder1
Encoder2 Pulse+ Pulse-	Plus2	Pulse output	6	5	Direction output	Sign2	Direction+ Direction- Encoder2
Encoder3 Pulse+ Pulse-	Plus3	Pulse output	4	3	Direction output	Sign3	Direction+ Direction- Encoder3
	COM	Output common port	2	1	Output common port	COM	
24VDC							

Output pulse: N-type model							
External wiring	Port	Function	CN5 terminal No.		Function	Port	External wiring
	COM	Output common port	COM	COM	Output common port	COM	
	Plus0	Pulse output	Y0	Y1	Direction output	Sign0	
	Plus1	Pulse output	Y2	Y3	Direction output	Sign1	
	Plus2	Pulse output	Y4	Y5	Direction output	Sign2	
	Plus3	Pulse output	Y6	Y7	Direction output	Sign3	

Configuration of high-speed pulse output ports

Function value configuration:

Define the variables to configure the ports, and map them to the high-speed pulse mapping table.

Configuration routine:

1. Configure Y0 as a high-speed pulse output port.

out0:=1;

Application.out0		Out0_Configure	%QB21	BYTE

2. Configure Y1 as a high-speed pulse output port.

out1:=1;

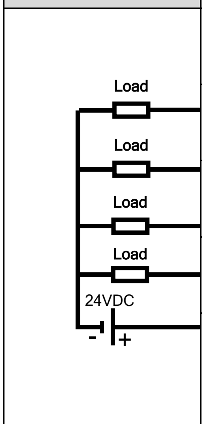
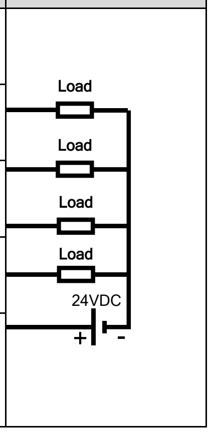
Application.out1		Out1_Configure	%QB22	BYTE

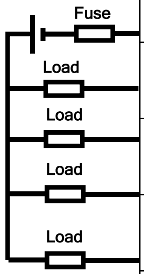
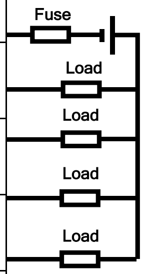
4.2.3.3 Output comparison function

If the function value is 2, the signal port is configured as an output comparison function with 8 channels.

The output comparison outputs the result of the counter single value comparison, and each counter channel has an output comparison function. If the counter value is equal to the set comparison value, it will output high, and if it is not equal, it will output low.

Wiring of output comparison ports

Comparison consistent output: P-type model							
External wiring	Port	Function	CN5 terminal No.		Function	Port	External wiring
	Y0	Common output	10	9	Common output	Y1	
	Y2	Common output	8	7	Common output	Y3	
	Y4	Common output	6	5	Common output	Y5	
	Y6	Common output	4	3	Common output	Y7	
	COM	Output common port	2	1	Output common port	COM	

Comparison consistent output: N-type model							
External wiring	Port	Function	CN5 terminal No.		Function	Port	External wiring
	COM	Output common port	COM	COM	Output common port	COM	
	Y0	Common output	Y0	Y1	Common output	Y1	
	Y2	Common output	Y2	Y3	Common output	Y3	
	Y4	Common output	Y4	Y5	Common output	Y5	
	Y6	Common output	Y6	Y7	Common output	Y7	

Configuration of output comparison ports

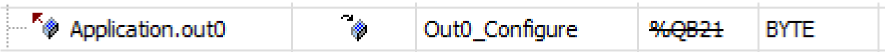
Function value configuration:

Define the variables to configure the ports, and map them to the high-speed pulse mapping table.

Configuration routine:

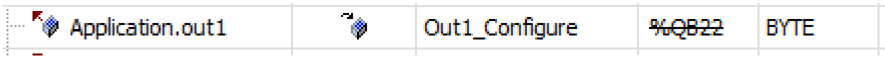
1. Configure Y0 as a comparison output port.

out0:=2;



2. Configure Y1 as a comparison output port.

out1:=2;



4.2.4 High-speed I/O mapping table

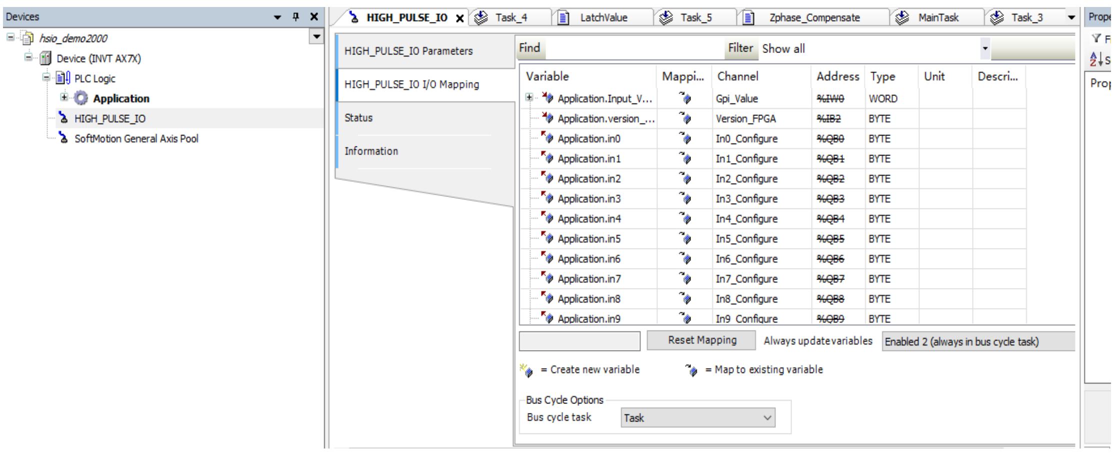
The device profile Shenzen INVT-AX7X-CPU_x.x.x.x.devdes is a CPU device profile that contains description of the high speed counting function, which is used for functional configuration of the input and output ports as well as the use and configuration of the interrupt function. See the following table.

Serial No.	Variable	Input/output type	Data type	Meaning
1	Gpi_Value	IN	Word	16-Channel general input feedback
2	Version_FPGA	IN	BYTE	FPGA version number. bit6–bit7: major version. bit3–bit5: minor version. bit0-bit2: revision number.
3	In0_Configure	IN	BYTE	Input terminal function configuration 0: Standard input function 1: Counting function 2: Trigger, latch and zero-signal function 3: Positive and negative limit zero function
4	In1_Configure	IN	BYTE	
5	In2_Configure	IN	BYTE	
6	In3_Configure	IN	BYTE	
7	In4_Configure	IN	BYTE	
8	In5_Configure	IN	BYTE	

Serial No.	Variable	Input/output type	Data type	Meaning
9	In6_Configure	IN	BYTE	4: Pulse width measurement function
10	In7_Configure	IN	BYTE	
11	In8_Configure	IN	BYTE	
12	In9_Configure	IN	BYTE	
13	InA_Configure	IN	BYTE	
14	InB_Configure	IN	BYTE	
15	InC_Configure	IN	BYTE	
16	InD_Configure	IN	BYTE	
17	InE_Configure	IN	BYTE	
18	InF_Configure	IN	BYTE	
19	XMode_SetA	OUT	BYTE	Counting function configuration for channel 0 (bit0-bit3), channel 1(bit4-bit7): 0: Single pulse 1: Quadrature encoder pulses (QEP) 2: Timing 3: SIGN+PULS
20	XMode_SetB	OUT	BYTE	Counting function configuration for channel 2 (bit0-bit3), channel 3(bit4-bit7) 0: Single pulse 1: Quadrature encoder pulses (QEP) 2: Timing 3: SIGN+PULS
21	XMode_SetC	OUT	BYTE	Counting function configuration for channel 4 (bit0-bit3), channel 5(bit4-bit7) 0: Single pulse 1: Quadrature encoder pulses (QEP) 2: Timing 3: SIGN+PULS
22	XMode_SetD	OUT	BYTE	Counting function configuration for channel 6 (bit0-bit3), channel 7(bit4-bit7) 0: Single pulse 1: Quadrature encoder pulses (QEP) 2: Timing 3: SIGN+PULS
23	Filt_Set	OUT	BYTE	Input signal filter parameter setting (unit: 0.25us)
24	Out0_Configure	OUT	BYTE	Output terminal function configuration 0: Common output function 1: High-speed pulse output function 2: Comparison output function 3–255: Reserved
25	Out1_Configure	OUT	BYTE	
26	Out2_Configure	OUT	BYTE	
27	Out3_Configure	OUT	BYTE	
28	Out4_Configure	OUT	BYTE	
29	Out5_Configure	OUT	BYTE	
30	Out6_Configure	OUT	BYTE	
31	Out7_Configure	OUT	BYTE	

Serial No.	Variable	Input/output type	Data type	Meaning
32	GPO_Set	OUT	BYTE	Common output signal setting bit0-bit7
33	Run_Enable	OUT	BYTE	bit0: Output channel 0 (1: enabled, 0: disabled) bit1: Output channel 1 (1: enabled, 0: disabled) bit2: Output channel 2 (1: enabled, 0: disabled) bit3: Output channel 3 (1: enabled, 0: disabled) bit6–bit7: Reserved.
34	YMode_Set	OUT	BYTE	Reserved
35	Interrupt	OUT	BOOL	Global interrupt enable
36	Interrupt_Enable	OUT	DWORD	Interrupt enable bit0: Interrupt 0 enable bit1: Interrupt 1 enable … bit19: Interrupt 19 enable
37	Interrupt_Mode	OUT	DWORD	Interrupt mode bit0-bit1: X0 interrupt mode bit2-bit3: X1 interrupt mode bit4-bit5: X2 interrupt mode bit6-bit7: X3 interrupt mode bit8-bit9: X4 interrupt mode bit10-bit11: X5 interrupt mode bit12-bit13: X6 interrupt mode bit14-bit15: X7 interrupt mode bit16-bit17: Probe 0 interrupt mode bit18-bit19: Probe 1 interrupt mode bit20- bit21: Probe 2 interrupt mode bit22-bit23: Probe 3 interrupt mode 0: rise edge 1: fall edge 2: Two edges

The operation interface of Invtmatic Studio is displayed as follows:



4.2.4.1 General input value

The variable corresponding to the device profile is Gpi_Value with the data type of WORD. This parameter is used when the input signal is set to the common input function. The input signals corresponding to the bits of the variable Gpi_Value are shown in the following table.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
XF	XE	XD	XC	XB	XA	X9	X8	X7	X6	X5	X4	X3	X2	X1	X0

If you need to read a common input signal, you can use either WORD mapping or bit mapping.

In WORD variable mapping mode, 16 input signal values can be read at the same time.

Application.Input_Value	Gpi_Value	~~%IW0~~	WORD

In Bit mapping mode, one variable can only read one signal value, and the variable type is BOOL.

	Gpi_Value	%IW0	WORD
Application.Xn0_Bit	Bit0	~~%IX0.0~~	BOOL
	Bit1	%IX0.1	BOOL
	Bit2	%IX0.2	BOOL
	Bit3	%IX0.3	BOOL

4.2.4.2 Version

The variable corresponding to the device profile is Version_FPGA with data type BYTE. It is used to read the FPGA version, where bit6–bit7: major version, bit3–bit5: minor version, bit0-bit2: revision number.

Application.version_fpga	Version_FPGA	~~%IB2~~	BYTE

4.2.4.3 Input terminal function configuration

Configure the function of the input port with data type BYTE. There are 16 input ports that can be configured for 5 functions. Including standard input function, counting function, triggering, latching, and Z-signal function, positive and negative limit zero function, and pulse width measurement function.

Application.in0	In0_Configure	~~%QB0~~	BYTE
Application.in1	In1_Configure	~~%QB1~~	BYTE
Application.in2	In2_Configure	~~%QB2~~	BYTE
Application.in3	In3_Configure	~~%QB3~~	BYTE
Application.in4	In4_Configure	~~%QB4~~	BYTE
Application.in5	In5_Configure	~~%QB5~~	BYTE
Application.in6	In6_Configure	~~%QB6~~	BYTE
Application.in7	In7_Configure	~~%QB7~~	BYTE
Application.in8	In8_Configure	~~%QB8~~	BYTE
Application.in9	In9_Configure	~~%QB9~~	BYTE
Application.inA	InA_Configure	~~%QB10~~	BYTE
Application.inB	InB_Configure	~~%QB11~~	BYTE
Application.inC	InC_Configure	~~%QB12~~	BYTE
Application.inD	InD_Configure	~~%QB13~~	BYTE
Application.inE	InE_Configure	~~%QB14~~	BYTE
Application.inF	InF_Configure	~~%QB15~~	BYTE

4.2.4.4 Counting mode configuration

There are 4 variables to configure the counting mode with the data type BYTE. Each variable can be configured for the counting mode of 2 channels. A total of 8 counter modes can be configured. See the following figure.

Application.xmodea	XMode_SetA	~~%QB16~~	BYTE
Application.xmodeb	XMode_SetB	~~%QB17~~	BYTE
Application.xmodec	XMode_SetC	~~%QB18~~	BYTE
Application.xmoded	XMode_SetD	~~%QB19~~	BYTE

Use 4 bits to set the counter mode with the following values:

Bit	Counting mode
0	Single pulse
1	Quadrature encoder pulses
2	Timing counting
3	Pulse + direction

Configure the bits of XMode_SetA to set the mode of different counters.

7	6	5	4	3	2	1	0
Counter 1				Counter 0			

Configure the bits of XMode_SetB to set the mode of different counters.

7	6	5	4	3	2	1	0
Counter 3				Counter 2			

Configure the bits of XMode_SetC to set the mode of different counters.

7	6	5	4	3	2	1	0
Counter 5				Counter 4			

Configure the bits of XMode_SetD to set the mode of different counters.

7	6	5	4	3	2	1	0
Counter 7				Counter 6			

4.2.4.5 Filter parameters

The variable of the corresponding device profile is Filt_Set in 0.25μs, which sets the filter parameters of input and output signals, with the data type BYTE and the maximum filter width 64μs. Adjust this parameter to improve the anti-interfere of the signal.

If the signal interference is strong, set the parameter value larger. If the interference is weak, set it smaller. The filter parameters are usually set to 1/4–1/3 (no more than 1/2) of the reference width which is the smaller one of the high pulse and low pulse width. The upper limit is 64μs. A parameter value that is too large will filter out the effective pulses, while a value that is too small may not filter out the clutter effectively.

Application.filt_set	Filt_Set	~~%QB20~~	BYTE

4.2.4.6 Output terminal function configuration

Configure the function of the output port with data type BYTE. There are 8 output ports that can be configured for 3 functions. For details, see the output port function description.

Application.out0	Out0_Configure	~~%QB21~~	BYTE
Application.out1	Out1_Configure	~~%QB22~~	BYTE
Application.out2	Out2_Configure	~~%QB23~~	BYTE
Application.out3	Out3_Configure	~~%QB24~~	BYTE
Application.out4	Out4_Configure	~~%QB25~~	BYTE
Application.out5	Out5_Configure	~~%QB26~~	BYTE
Application.out6	Out6_Configure	~~%QB27~~	BYTE
Application.out7	Out7_Configure	~~%QB28~~	BYTE

4.2.4.7 Common output value

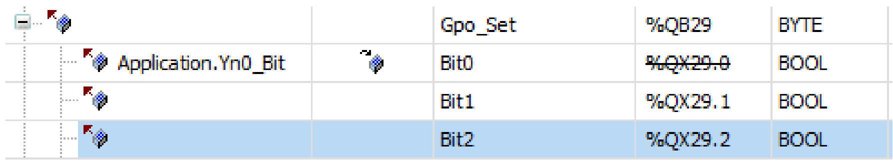
Common means the common function output. The variable corresponding to the device profile is GPO_Set with the data type of BYTE. This parameter is used when the output signal is set to the standard output function. The output signals corresponding to the bits of the variable GPO_Set are shown in the following table.

7	6	5	4	3	2	1	0
Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0

If you need to set a common output signal, you can use either BYTE mapping or bit mapping.

In BYTE variable mapping mode, 8 output signal values can be set at the same time.

In Bit mapping mode, one variable can only set one signal value, and the variable type is BOOL.



4.2.4.8 High-speed pulse output function

The variable corresponding to the device profile is Run_Enable with the data type of BYTE. This parameter is used for channel enable at high speed pulse output. The bits of the variable Run_Enable corresponds to the channel enable, 1 indicates enabled, 0 indicates disabled. The following table shows the correspondence between channels and bits.

7	6	5	4	3	2	1	0
Reserved				Channel 3	Channel 2	Channel 1	Channel 0

4.2.4.9 Global interrupt enable

The variable corresponding to the device profile is Interrupt, which is the master switch that enables all interrupts, with the data type of BOOL. 1 indicates total interrupt enabled and 0 indicates disabled.

Serial No.	Variable	Input/output type	Data type	Meaning
35	Interrupt	OUT	BOOL	Global interrupt enable

4.2.4.10 Interrupt enable

The variable corresponding to the device profile is Interrupt_Enable with the data type of DWORD. HSIO supports 20 types of interrupts, including 8 external input interrupts, 8 count-comparison interrupts, and 4 probe interrupts, each of which can be enabled with the bit of Interrupt_Enable. The mapping is shown in the following table.

19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Probe interrupt enable				Comparison interrupt enable								External interrupt enable							

Bit0–bit7 corresponds to external interrupt 0–7 respectively.

Bit8–bit15 corresponds to comparison interrupt 0–7 respectively.

Bit16–bit19 corresponds to probe interrupt 0–3 respectively.

4.2.4.11 Interrupt mode

The variable corresponding to the device profile is Interrupt_Mode with the data type of DWORD. Only external interrupts and probe interrupts require an interrupt mode. Each mode consists of 2 bits. The mapping of interrupt modes and bits is shown in the following table.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
External interrupt 7		External interrupt 6		External interrupt 5		External interrupt 4		External interrupt 3		External interrupt 2		External interrupt 1		External interrupt 0	

23	22	21	20	19	18	17	16
Probe interrupt 3		Probe interrupt 2		Probe interrupt 1		Probe interrupt 0	

Use 2 bits to set the interrupt mode with the following values:

Motion mode configuration	Motion mode
0	Rising edge
1	Falling edge
2	Two edges

4.2.5 Interrupt instruction

The HSIO supports 20 types of interrupts, including 8 external input interrupts, 8 count-comparison interrupts and 4 probe interrupts. To use the interrupt function, configure the corresponding IO port function. Then, enable the global interrupt and the required interrupt bits. If an external input interrupt or probe interrupt is used, the interrupt mode must also be set.

4.2.5.1 External interrupt instruction

The corresponding input port numbers for P-type model external interrupts are X0–X7, and the corresponding input port numbers for N-type model external interrupts are A0/B0/A1/B1/X4–X7. Configure these ports as common input ports, set an interrupt mode to enable interrupts, and configure the interrupt task so that the operations can be performed in the interrupt task.

External interrupt configuration

Follow the steps to implement the interrupt function:

Step 1 Set the input terminal as standard input function

For details, see Input terminal function description.

Step 2 Set global interrupt

Set Interrupt to true. See Global interrupt enable in the device profile parameter description.

Serial No.	Variable	Input/output type	Data type	Meaning
35	Interrupt	OUT	BOOL	Global interrupt enable

Step 3 Set input port interrupt

Set the 8 input port bits of the Interrupt_Enable the device profile, with Gpix of input port x set to true. Set a bit to enable the interrupt function mapping to that bit.

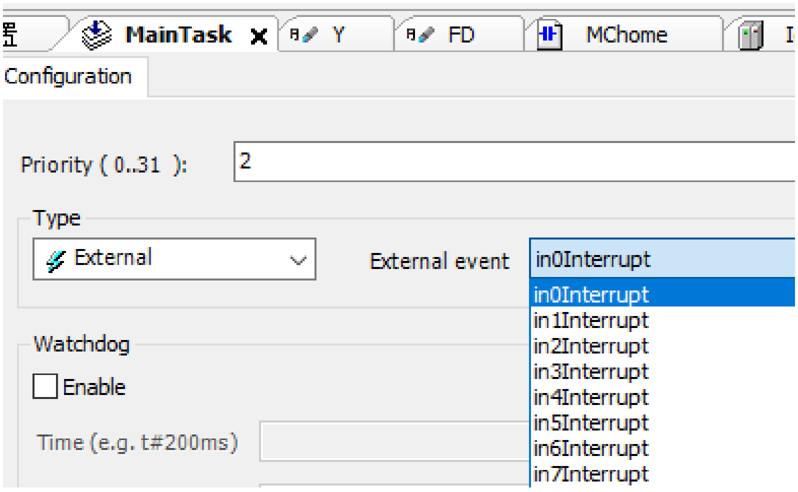
Interrupt_Enable	%QD9	DWORD
Gpi0	%QX36.0	BOOL
Gpi1	%QX36.1	BOOL
Gpi2	%QX36.2	BOOL
Gpi3	%QX36.3	BOOL
Gpi4	%QX36.4	BOOL
Gpi5	%QX36.5	BOOL
Gpi6	%QX36.6	BOOL
Gpi7	%QX36.7	BOOL

Step 4 Set interrupt mode

The interrupt mode setting consists of 2 bits, and different interrupts correspond to different bits. For details, see Interrupt mode in the device profile parameter description.

Step 5 Select interrupt task

In the Invtmatic Studio task, set the type to **External**, and select the event inxInterrupt of the input port X0–X7, where x ranges from 0 to 7.



An external signal generates an interrupt based on the interrupt mode and calls the corresponding task execution.

External interrupt timing

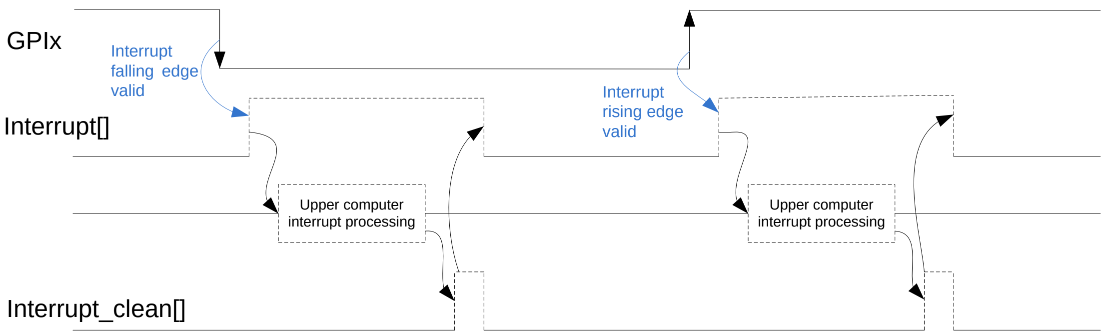


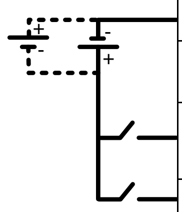
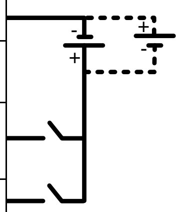
Figure 4-1 External input interrupt timing

GPIx represents the xth external general input channel where 0 =< x <= 7, and Interrupt[] is the interrupt state output of GPIx. The high-level pulse output by Interrupt[] uses a dotted line to indicate that interrupts can be output only if the interrupt mode is valid and the interrupt enable is valid. The upper computer interrupt process and the interrupt_clean[] signal only appear after the output of the Interrupt[], so they are also presented as dotted lines. Interrupt_clean[] is the clear signal given by the upper computer in response to the Interrupt[], which clears the Interrupt[] to zero.

4.2.5.2 Probe interrupt instruction

The corresponding input port numbers for P-type model probe interrupts are X8–XB (i.e. CxT, 0 =< x <= 3), and the corresponding input port numbers for N-type model probe interrupts are X8–X11. The input port signal function should be configured as a latching function.

Probe interrupt wiring

External wiring	Port	Function	CN5 terminal No.		Function	Port	External wiring
	SS1	Input common port	22	21	Input common port	SS2	
	C0T	Probe signal input	18	17	Probe signal input	C1T	
	C2T	Probe signal input	16	15	Probe signal input	C3T	

Probe interrupt configuration

Follow the steps to implement the interrupt function:

Step 1 Set the input terminal as latching function.

For details, see Input terminal function description.

Step 2 Set global interrupt.

Set Interrupt to true, see Global interrupt enable in the device profile parameter description.

Serial No.	Variable	Input/output type	Data type	Meaning
35	Interrupt	OUT	BOOL	Global interrupt enable

Step 3 Set input port interrupt.

Set the 4 input port bits of the Interrupt_Enable the device profile, with Trigx of input port x set to true. Set a bit to enable the interrupt function mapping to that bit.

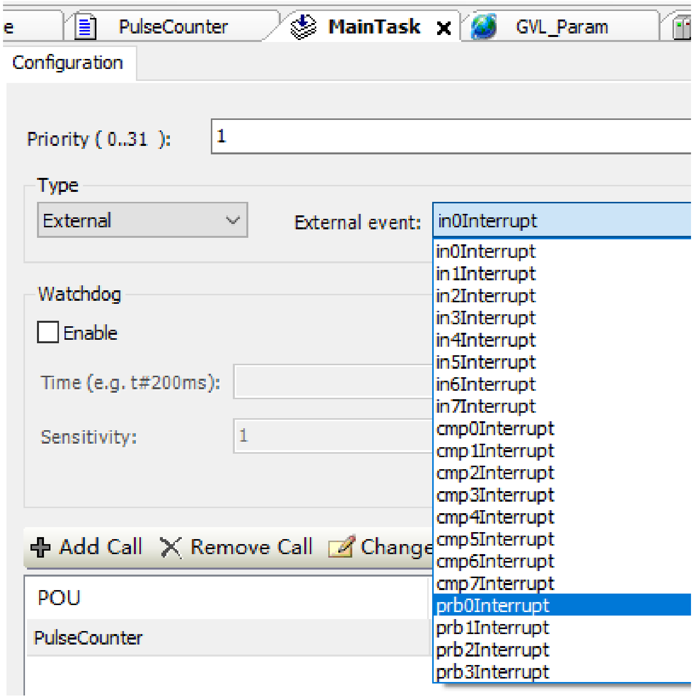
Application.P...		Trig0	~~%QX38.0~~	BOOL
Application.P...		Trig1	~~%QX38.1~~	BOOL
Application.P...		Trig2	~~%QX38.2~~	BOOL
Application.P...		Trig3	~~%QX38.3~~	BOOL

Step 4 Set interrupt mode.

The interrupt mode setting consists of 2 bits, and different interrupts correspond to different bits. For details, see Interrupt mode in the device profile parameter description.

Step 5 Select interrupt task.

In the Invtmatic Studio task, set the type to **External**, and select the event prbxInterrupt of the input port X8–XB (X8–X11), where x ranges from 0 to 3. Read the probe latching value in the *LatchValue_HP function block* via the interrupt task flag.



An external signal generates an interrupt based on the interrupt mode and calls the corresponding task execution.

Probe interrupt timing

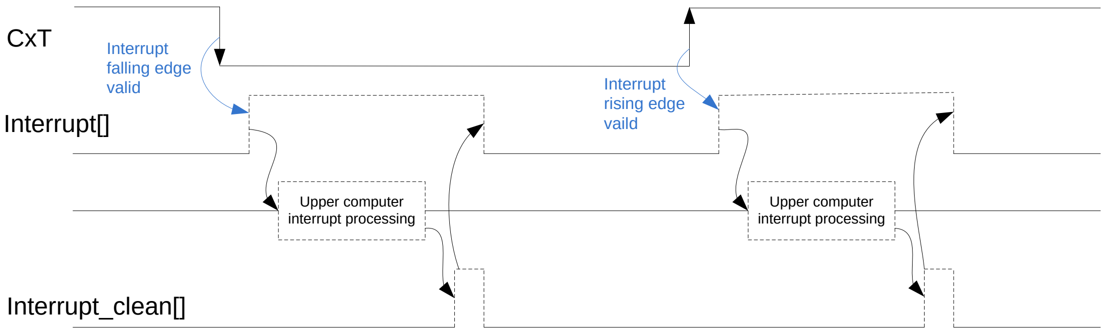


Figure 4-2 Probe input interrupt timing

CxT represents the xth probe input channel where 0=<x<=3, and Interrupt[] is the interrupt state output of CxT. The high-level pulse output by Interrupt[] uses a dotted line to indicate that interrupts can be output only if the interrupt mode is valid and the interrupt enable is valid. The upper computer interrupt process and the interrupt_clean[] signal only appear after the output of the Interrupt[], so they are also presented as dotted lines. Interrupt_clean[] is the clear signal given by the upper computer in response to the Interrupt[], which clears the Interrupt[] to zero.

4.2.5.3 Comparison interrupt instruction

Comparison interrupt includes single-value comparison interrupt and multi-value comparison interrupt. Single-value comparison interrupt is generated by calling the function block CompareSingleValue_HP, and multi-value comparison interrupt is generated by calling CompareMoreValue_HP. The following steps describe the generation of single-value interrupt and multi-value interrupt respectively.

Comparison interrupt configuration

- Single-value comparison interrupt:

  1: Set the input terminal as counting function.

  For details, see Input terminal function description.

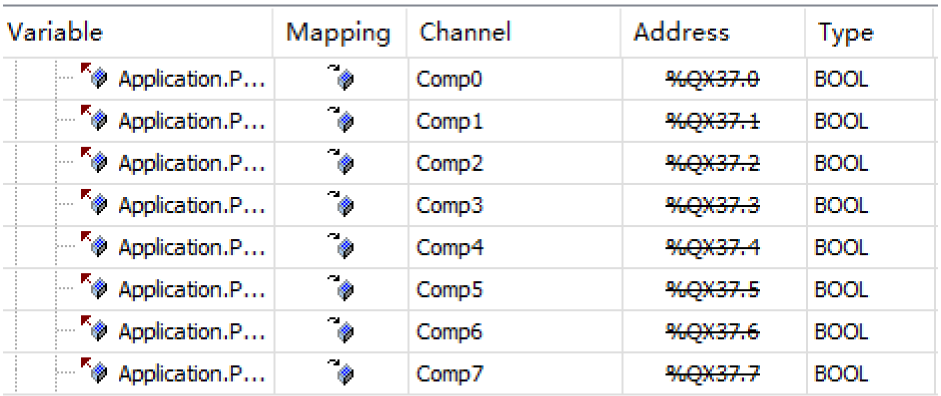
  2: Set global interrupt.

Set Interrupt to true, see Global interrupt enable in the device profile parameter description.

Serial No.	Variable	Input/output type	Data type	Meaning
35	Interrupt	OUT	BOOL	Global interrupt enable

3: Set input port interrupt.

Set the 8 input port bits of the Interrupt_Enable the device profile, with Compx of input port x set to true. Set a bit to enable the interrupt function mapping to that bit.



Variable	Mapping	Channel	Address	Type
Application.P...		Comp0	~~%QX37.0~~	BOOL
Application.P...		Comp1	~~%QX37.1~~	BOOL
Application.P...		Comp2	~~%QX37.2~~	BOOL
Application.P...		Comp3	~~%QX37.3~~	BOOL
Application.P...		Comp4	~~%QX37.4~~	BOOL
Application.P...		Comp5	~~%QX37.5~~	BOOL
Application.P...		Comp6	~~%QX37.6~~	BOOL
Application.P...		Comp7	~~%QX37.7~~	BOOL

4: Set the comparison interrupt output.

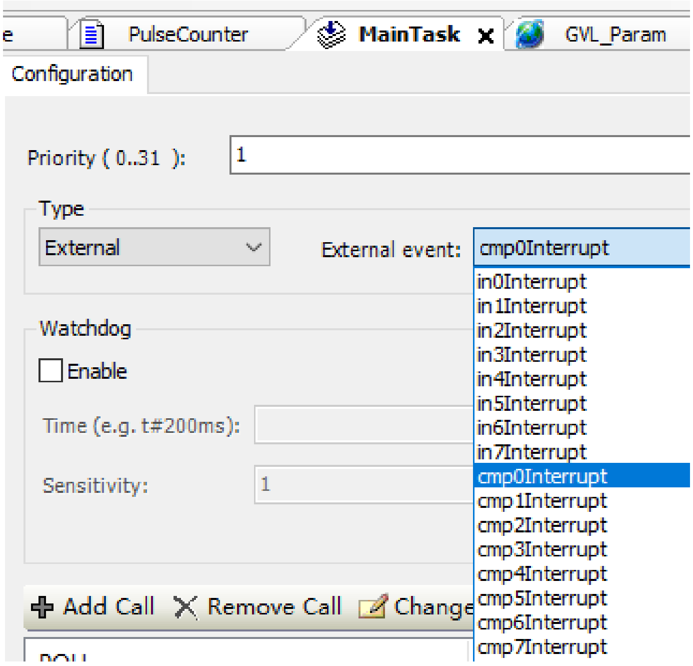
If comparison interrupt output is not needed, skip this step.

Select the port to be output, set the corresponding port in the device profile as the comparison output function, and select any one of the following 8 channels through the single-value comparison function block CompareSingleValue_HP parameter OutChannel. The OutChanne value ranges from 0 to 7. One output channel OutChannel value can only correspond to one CMP channel.

Output terminal	Standard output function	High-speed pulse output function	Comparison output function
Y0	General Common 0	CH0CW/PULS0	CMP0
Y1	Common 1	CH0CCW/SIGN0	CMP1
Y2	Common 2	CH1CW/PULS1	CMP2
Y3	Common 3	CH1CCW/SIGN1	CMP3
Y4	Common 4	CH2CW/PULS2	CMP4
Y5	Common 5	CH2CCW/SIGN2	CMP5
Y6	Common 6	CH3CW/PULS3	CMP6
Y7	Common 7	CH3CCW/SIGN3	CMP7

5: Select interrupt task.

In the Invtmatic Studio task, set the type to **External**, and select **cmpxInterrupt**, where x ranges from 0 to 7.



If the comparison value is equal, an interrupt is generated and the corresponding task execution is called. The channel x corresponds to the cmpxInterrupt comparison interrupt task and cannot be modified at will.

6: Call function block to generate interrupt

Single-value comparison calls the function block CompareSingleValue_HP to generate an interrupt. Setting the comparison value to be the same as the count value can also generate an interrupt output.

- Multi-value comparison interrupt:

1: Set the input terminal as counting function

For details, see Input terminal function description.

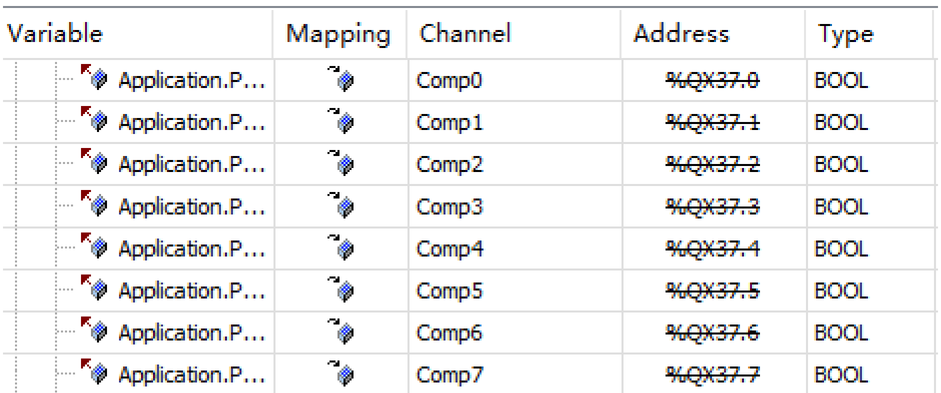
2: Set global interrupt

Set Interrupt to true, see Global interrupt enable in the device profile parameter description.

Serial No.	Variable	Input/output type	Data type	Meaning
35	Interrupt	OUT	BOOL	Global interrupt enable

3: Set input port interrupt

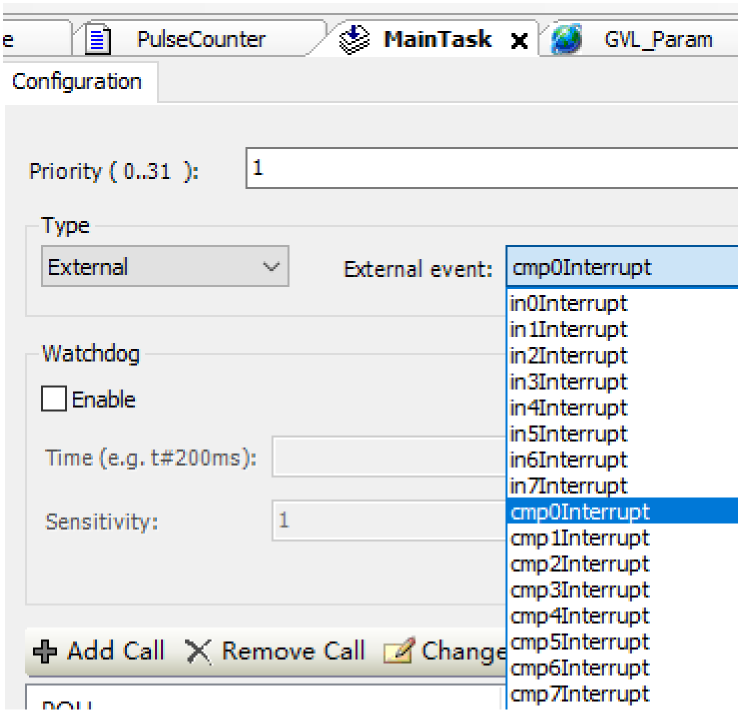
Set the 8 port bits of the Interrupt_Enable the device profile, with Compx of port x set to true. Since a multi-value comparison function block can be used to generate multiple interrupts, the first value is the enable bit of Cmp0 interrupt, the second value is the enable bit of Cmp1 interrupt, and so on, and the eighth value is the enable bit of Cmp7 interrupt. It cannot be modified arbitrarily.



Variable	Mapping	Channel	Address	Type
Application.P...		Comp0	%QX37.0	BOOL
Application.P...		Comp1	%QX37.1	BOOL
Application.P...		Comp2	%QX37.2	BOOL
Application.P...		Comp3	%QX37.3	BOOL
Application.P...		Comp4	%QX37.4	BOOL
Application.P...		Comp5	%QX37.5	BOOL
Application.P...		Comp6	%QX37.6	BOOL
Application.P...		Comp7	%QX37.7	BOOL

4: Select interrupt task

In the Invtmatic Studio task, set the type to **External**, and select **cmpxInterrupt**, where x ranges from 0 to 7.



The multi-value comparison function block has multiple comparison values, each of which corresponds to an interrupt enabled bit of Compx. It shares a one-to-one mapping with the interrupt task cmpxInterrupt where x ranges from 0 to 7 and cannot be modified at will.

5: Call function block to generate interrupt

Multi-value comparison calls the function block CompareMoreValue_HP to generate an interrupt. Setting the comparison value to be the same as the count value will generate an interrupt output. For now, only eight comparison values are supported for multi-value comparisons to generate interrupts, that is, the first eight values of a multi-value comparison can generate interrupts.

Comparison interrupt timing

- Single-value comparison interrupt

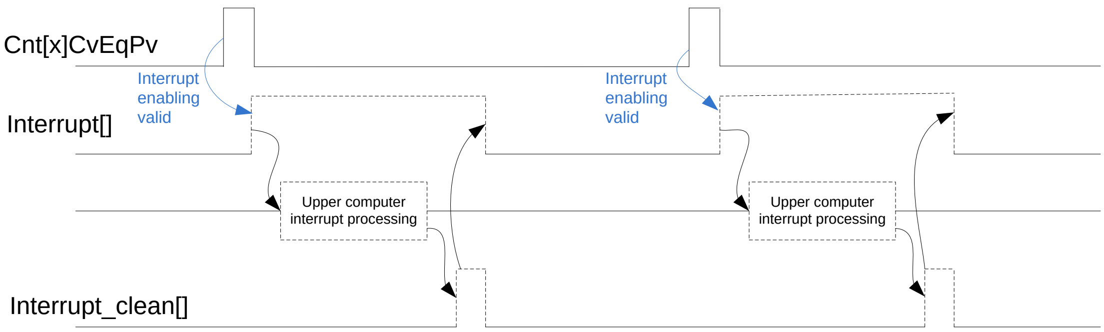


Figure 4-3 Single-value comparison interrupt timing

Cnt[x]CvEqPv represents the single-value comparison signal of the xth counting channel, in which 0 =< x <= 7. A high pulse indicates that cv and pv are equal. Interrupt[] is the interrupt state output corresponding to Cnt[x]CvEqPv. The high-level pulse output by Interrupt[] uses a dotted line to indicate that interrupts can be output if the interrupt enable is valid. The upper computer interrupt process and the interrupt_clean[] signal only appear after the output of the Interrupt[], so they are also presented as dotted lines. Interrupt_clean[] is the clear signal given by the upper computer in response to the Interrupt[], which clears the Interrupt[] to zero.

- Multi-value comparison interrupt

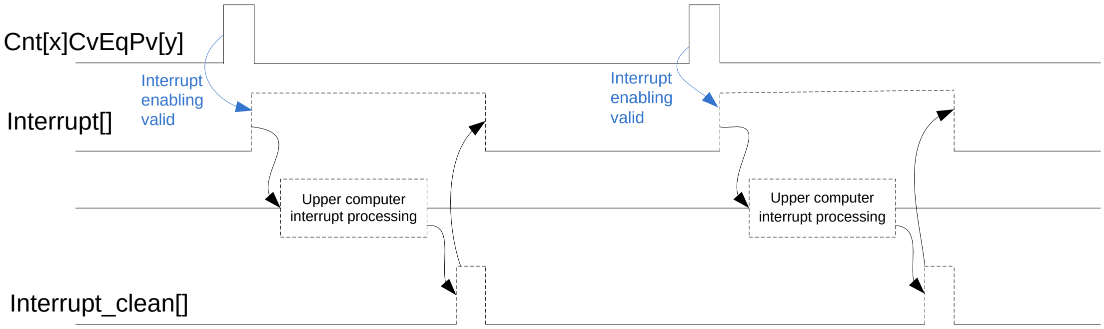


Figure 4-4 Multi-value comparison interrupt timing

Cnt[x]CvEqPv[y] represents the yth comparison value signal of the xth counting channel, in which 0=<x<=3 and 0=<y<=7. A high pulse indicates that cv and pv are equal. Interrupt[] is the interrupt state output corresponding to Cnt[x]CvEqPv[y]. The high-level pulse output by Interrupt[] uses a dotted line to indicate that interrupts can be output if the interrupt enable is valid. The upper computer interrupt process and the interrupt_clean[] signal only appear after the output of the Interrupt[], so they are also presented as dotted lines. Interrupt_clean[] is the clear signal given by the upper computer in response to the Interrupt[], which clears the Interrupt[] to zero.

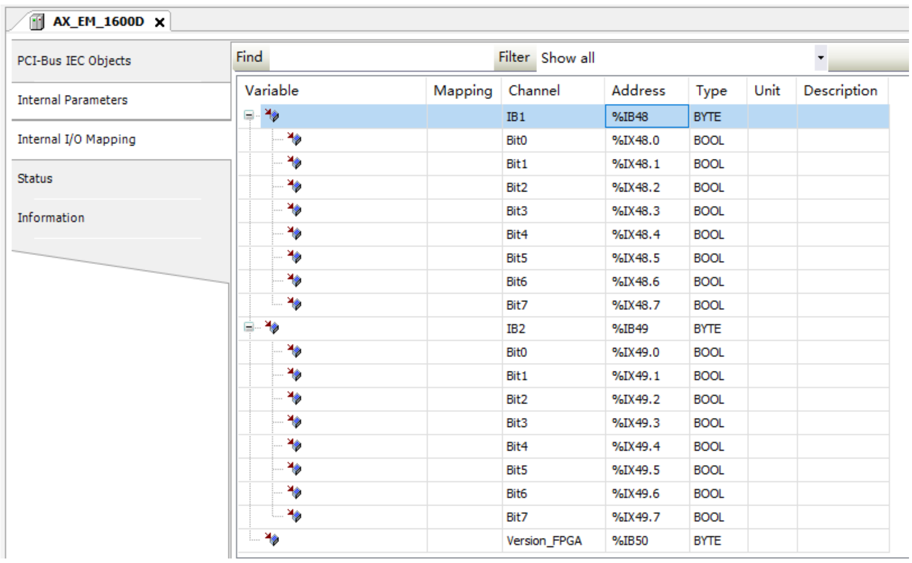
In the single-value comparison interrupt, each counting channel has only one interrupt signal output, and all counting channels (0-7) can output single-value comparison interrupt signals. In the multi-value comparison interrupts, only counting channels 0-3 can output multi-value interrupts, and each counter can output 8 (0-7) interrupt signals. When a multi-value counting channel is selected, its yth comparison value corresponds to the interrupt signal one by one. Only one counting channel is valid at a time for the multi-value comparison interrupt.

4.3 Digital input/output module

4.3.1 Creating a project for digital input/output module

Create a digital I/O application. Add the device profile **AX_EM_1600D_x.x.x.x.devdesc.xml**, **AX_EM_0016DP_x.x.x.x.devdesc.xml**, and **AX_EM_0016DN_x.x.x.x.devdesc.xml** required by the module.

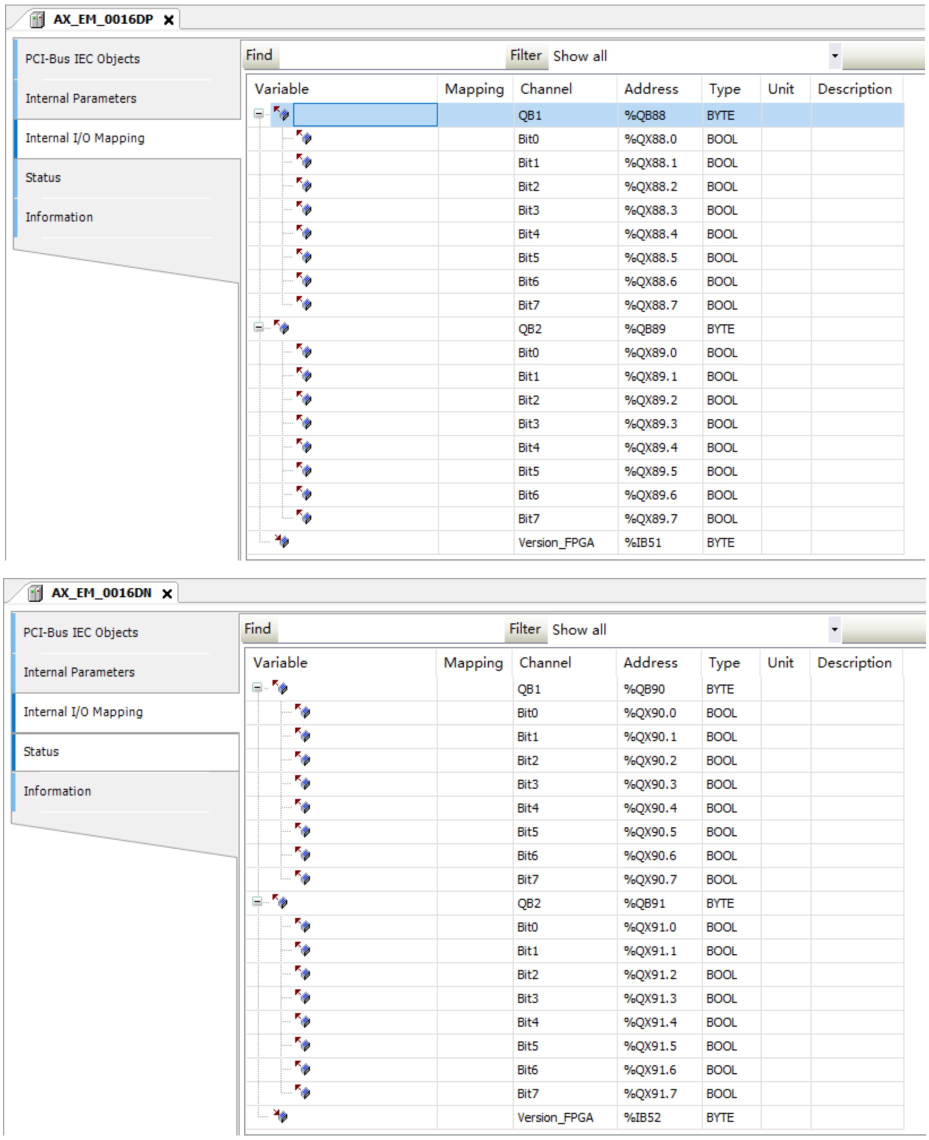
4.3.2 Variable definition and use



Variable	Mapping	Channel	Address	Type	Unit	Description
		IB1	%IB48	BYTE		
		Bit0	%IX48.0	BOOL		
		Bit1	%IX48.1	BOOL		
		Bit2	%IX48.2	BOOL		
		Bit3	%IX48.3	BOOL		
		Bit4	%IX48.4	BOOL		
		Bit5	%IX48.5	BOOL		
		Bit6	%IX48.6	BOOL		
		Bit7	%IX48.7	BOOL		
		IB2	%IB49	BYTE		
		Bit0	%IX49.0	BOOL		
		Bit1	%IX49.1	BOOL		
		Bit2	%IX49.2	BOOL		
		Bit3	%IX49.3	BOOL		
		Bit4	%IX49.4	BOOL		
		Bit5	%IX49.5	BOOL		
		Bit6	%IX49.6	BOOL		
		Bit7	%IX49.7	BOOL		
		Version_FPGA	%IB50	BYTE		

Figure 4-5 Variable mapping of input module

The IB1/IB2 input point status can be obtained by BYTE or BOOL type.



AX_EM_0016DP

PCI-Bus IEC Objects | Internal Parameters | Internal I/O Mapping | Status | Information

Find | Filter Show all

Variable	Mapping	Channel	Address	Type	Unit	Description
		QB1	%QB88	BYTE		
		Bit0	%QX88.0	BOOL		
		Bit1	%QX88.1	BOOL		
		Bit2	%QX88.2	BOOL		
		Bit3	%QX88.3	BOOL		
		Bit4	%QX88.4	BOOL		
		Bit5	%QX88.5	BOOL		
		Bit6	%QX88.6	BOOL		
		Bit7	%QX88.7	BOOL		
		QB2	%QB89	BYTE		
		Bit0	%QX89.0	BOOL		
		Bit1	%QX89.1	BOOL		
		Bit2	%QX89.2	BOOL		
		Bit3	%QX89.3	BOOL		
		Bit4	%QX89.4	BOOL		
		Bit5	%QX89.5	BOOL		
		Bit6	%QX89.6	BOOL		
		Bit7	%QX89.7	BOOL		
		Version_FPGA	%IB51	BYTE		

AX_EM_0016DN

PCI-Bus IEC Objects | Internal Parameters | Internal I/O Mapping | Status | Information

Find | Filter Show all

Variable	Mapping	Channel	Address	Type	Unit	Description
		QB1	%QB90	BYTE		
		Bit0	%QX90.0	BOOL		
		Bit1	%QX90.1	BOOL		
		Bit2	%QX90.2	BOOL		
		Bit3	%QX90.3	BOOL		
		Bit4	%QX90.4	BOOL		
		Bit5	%QX90.5	BOOL		
		Bit6	%QX90.6	BOOL		
		Bit7	%QX90.7	BOOL		
		QB2	%QB91	BYTE		
		Bit0	%QX91.0	BOOL		
		Bit1	%QX91.1	BOOL		
		Bit2	%QX91.2	BOOL		
		Bit3	%QX91.3	BOOL		
		Bit4	%QX91.4	BOOL		
		Bit5	%QX91.5	BOOL		
		Bit6	%QX91.6	BOOL		
		Bit7	%QX91.7	BOOL		
		Version_FPGA	%IB52	BYTE		

Figure 4-6 Variable mapping of output module

The QB1/QB2 output point status can be controlled by BYTE or BOOL type.

4.4 Analog input/output module

4.4.1 Creating a project for analog input/output module

Create an analog I/O application project. Add the device profile **AX_EM_4AD_x.x.x.x.devdesc.xml** and **AX_EM_4DA_x.x.x.x.devdesc.xml** required by the module.

4.4.2 Variable definition and use

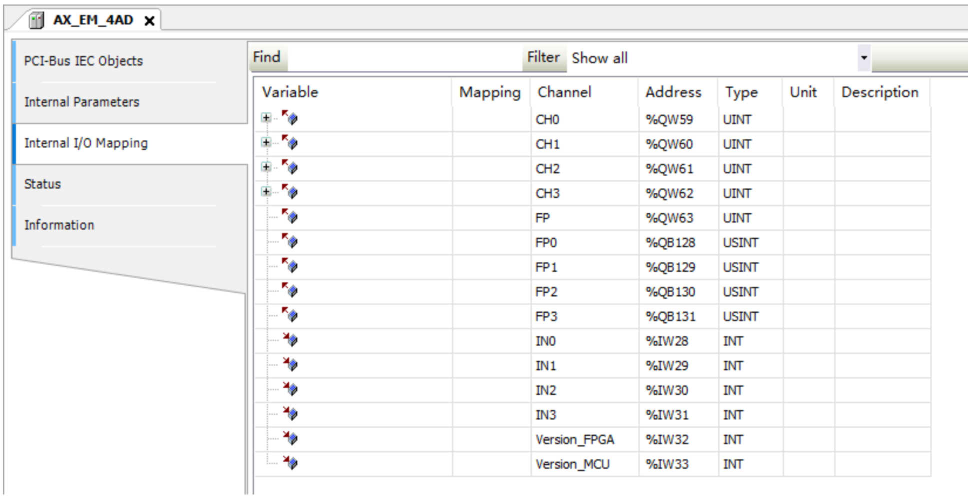


Figure 4-7 Variable mapping of analog input module

Variable	Input/output type	Data type	Meaning
CH0~CH3	IN	UINT	Control word of sampling channel 0–3. See the module user manual for specific meaning.
FP	IN	UINT	AD sampling chip filter selection. See the module user manual for specific meaning.
FP0~FP3	IN	USINT	Filter parameter configuration of sampling channel 0–3. See the module user manual for specific meaning.
IN0~IN3	OUT	INT	Analog code value of sampling channel 0–3
Version_FPGA	OUT	INT	FPGA version number, converted to octal number
Version_MCU	OUT	INT	MCU version number, converted to octal number

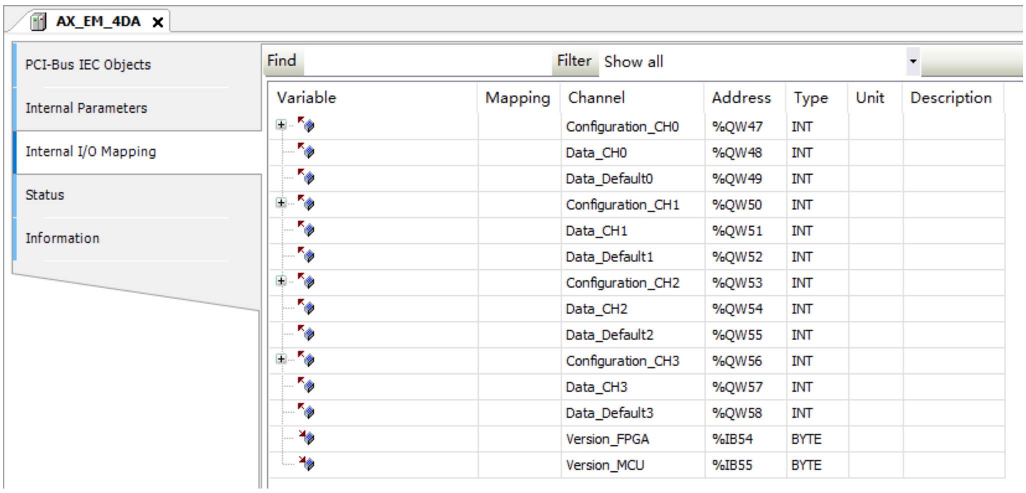


Figure 4-8 Variable mapping of analog output module

Variable	Input/output type	Data type	Meaning
Configuration_CH0~3	IN	INT	Control word of output channel 0–3. See the module user manual for specific meaning.
Data_CH0~3	IN	INT	Output analog code value of output channel 0–3
Data_Default0~3	IN	INT	Output analog preset code value of output

Variable	Input/output type	Data type	Meaning
			channel 0–3
Version_FPGA	OUT	BYTE	FPGA version number, converted to octal number
Version_MCU	OUT	BYTE	MCU version number, converted to octal number

4.5 Temperature module

4.5.1 Creating a project for temperature module

Create a temperature module application. Add the device profile **AX_EM_4PTC_.x.x.x.devdesc.xml** required by the module.

4.5.2 Variable definition and use

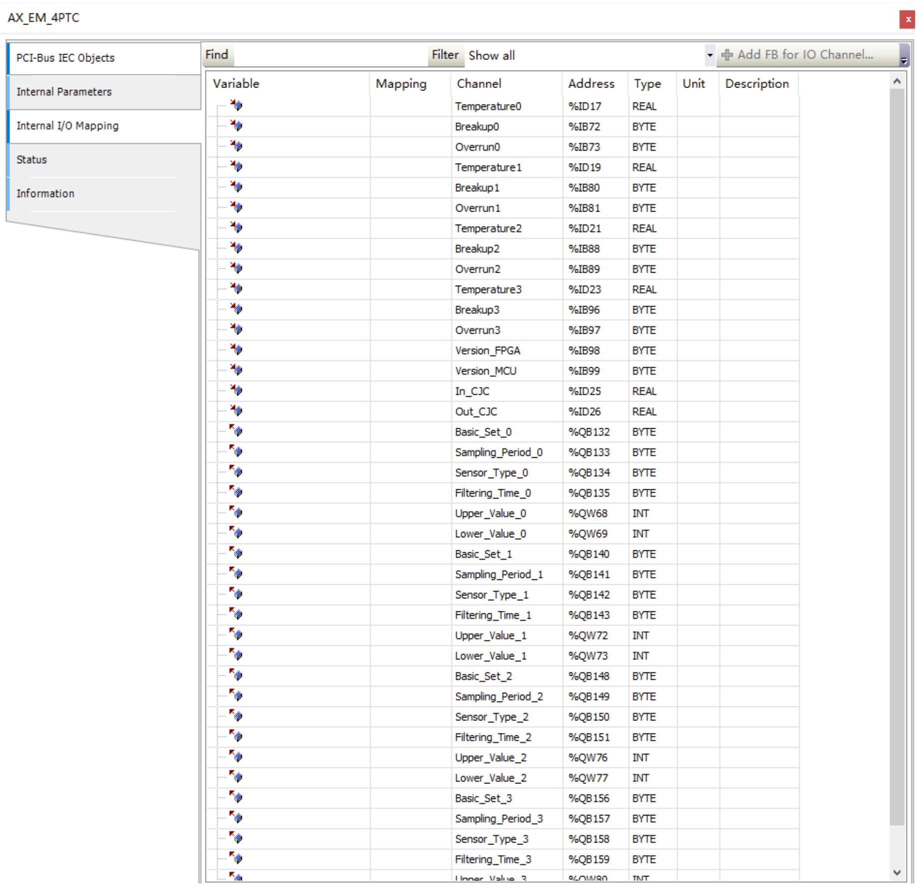


Figure 4-9 Variable mapping of temperature module

Variable	Input/output type	Data type	Meaning
Basic_Set_0~3	IN	BYTE	Control word of temperature sampling channel 0–3. See the module user manual for specific meaning.
Sampling_period_0~3	IN	BYTE	Sampling period of temperature sampling channel 0–3 (reserved)
Sensor_type_0~3	IN	BYTE	Sensor type of temperature sampling channel 0–3
Filtering_time_0~3	IN	BYTE	Sampling filter configuration of temperature sampling channel 0–3 (reserved)
Upper_value_0~3	IN	INT	Temperature upper limit of temperature sampling channel 0–3

Variable	Input/output type	Data type	Meaning
Lower_value_0~3	IN	INT	Temperature lower limit of temperature sampling channel 0–3
Temperature0~3	OUT	REAL	Temperature sampling value of temperature sampling channel 0–3
Breakup0~3	OUT	BYTE	Disconnection flag of temperature sampling channel 0–3 (reserved)
Overrun0~3	OUT	BYTE	Over-limit flag of temperature sampling channel 0–3
In_CJC	OUT	REAL	Internal cold junction temperature
Out_CJC	OUT	REAL	External cold junction temperature
Version_FPGA	OUT	BYTE	FPGA version number, converted to octal number
Version_MCU	OUT	BYTE	MCU version number, converted to octal number

4.6 Communication module

The EtherCAT communication module is used as an EtherCAT slave. Before using the module, add the device profile **INVT_ECAT_SLAVE_Vx.xx.xml**. For detailed instructions, refer to the case of adding DA200 servo drive to the EtherCAT master node.

1. Create a new project in the Invtmatic Studio upper computer, Right click **Device** to add a device, and add an EtherCAT Master SoftMotion module, as shown in the following figure:

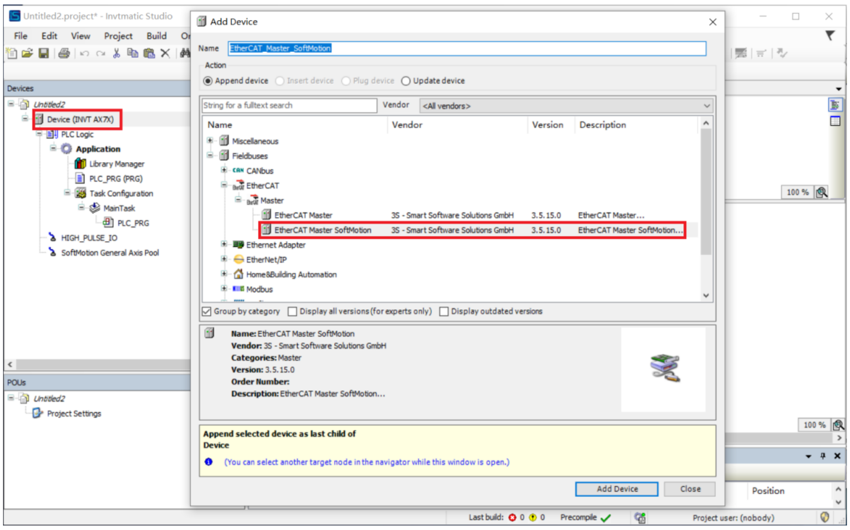


Figure 4-10 Add an EtherCAT Master SoftMotion module

2. Right click the EtherCAT Master SoftMotion module to add a device, and add the EtherCAT Slave Module (AX-EM-RCM-ET), as shown in the following figure:

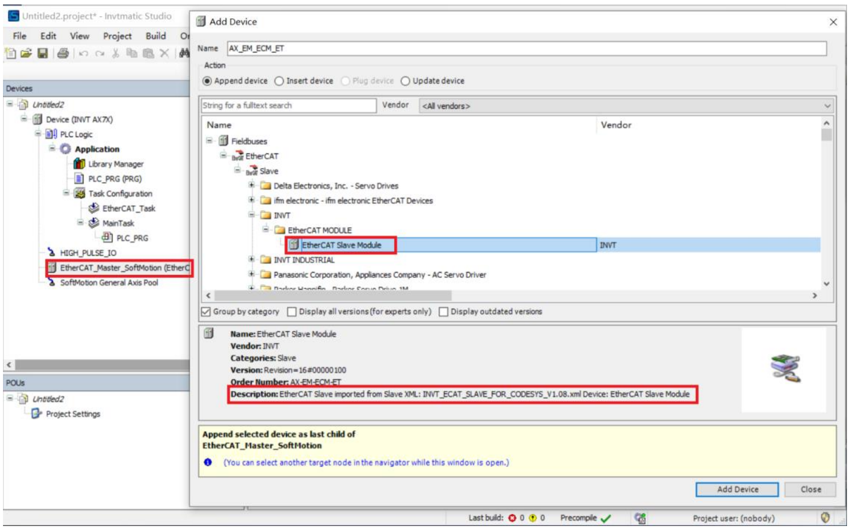


Figure 4-11 Add a EtherCAT remote expansion module

The following section explains how to use the EtherCAT remote expansion module to extend our existing IO.

4.6.1 Digital input module

EtherCAT remote extension module (AX-EM-Rcm-ET) is used to extend the digital input module (AX-EM-1600D) through the backplane. The instructions are as follows:

1. Right click **AX-EM-ECM-ET** in the device panel to add the digital input module (AX_EM_1600D). Control 16 channels through two sets of variables InByte0 and InByte1 in the Module/IO mapping tab, as shown in the following figure.

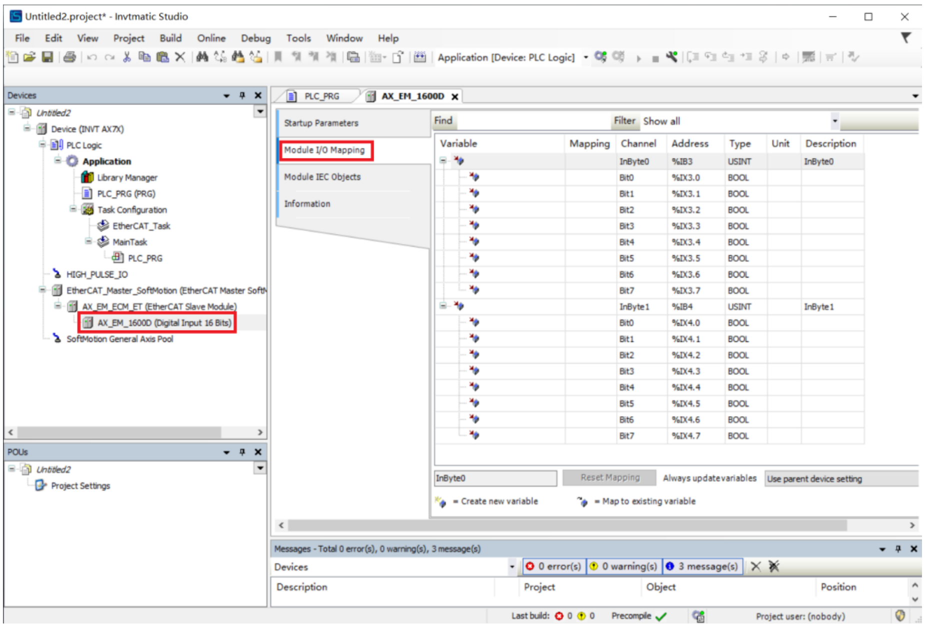


Figure 4-12 Variable mapping of digital input module

2. After compiling, log in to download the project and run it.

4.6.2 Digital output module

EtherCAT remote extension module (AX-EM-Rcm-ET) is used to extend the digital output module (AX-EM-0016DP/ AX-EM-0016DN) through the backplane. The instructions are as follows:

1. Right click **AX-EM-ECM-ET** in the device panel to add the digital output module (AX_EM_0016DP). Control 16 channels through two sets of variables OutByte0 and OutByte1 in the Module/IO mapping tab, as shown in the following figure.

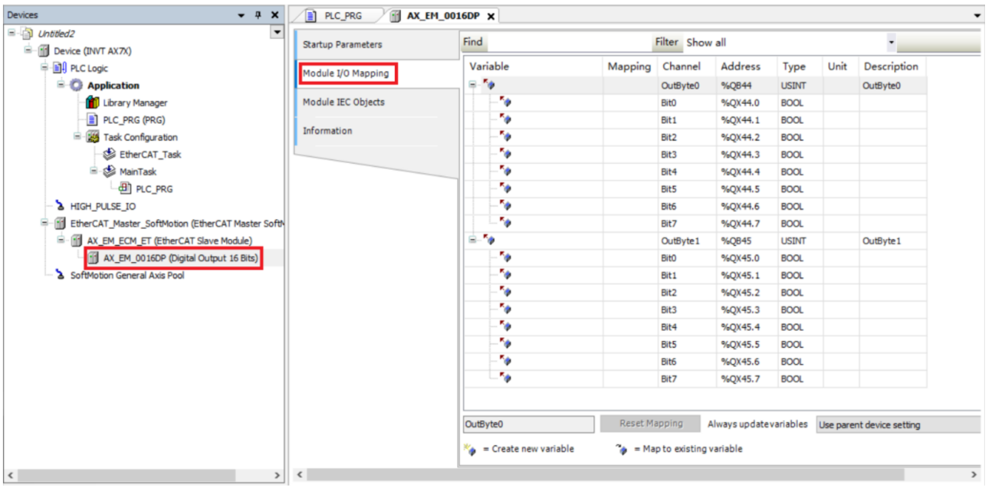


Figure 4-13 Variable mapping of digital output module

2. After compiling, log in to download the project and run it.

4.6.3 Analog input module

EtherCAT remote extension module (AX-EM-Rcm-ET) is used to extend the analog input module (AX-EM-4AD) through the backplane. The instructions are as follows:

1. Right click **AX-EM-ECM-ET** in the device panel to add the analog input module (AX_EM_4AD). Control the module through the multiple sets of variables in the Module/IO mapping tab, as shown in the following figure.

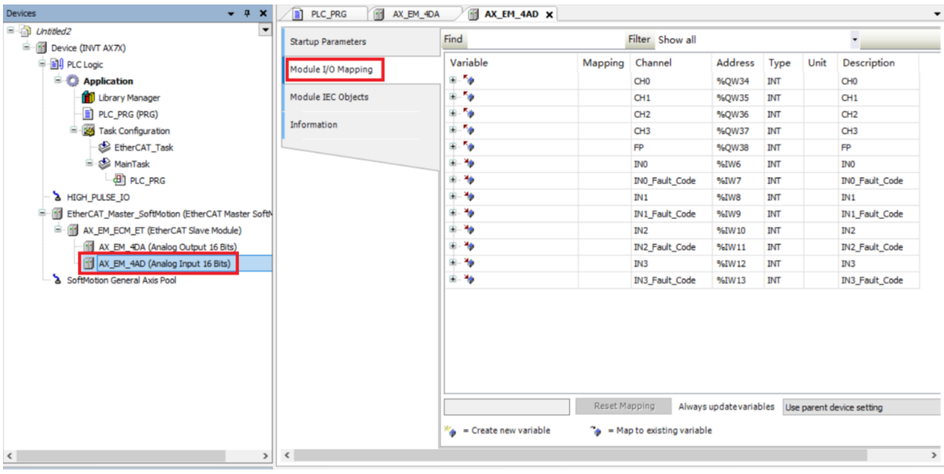


Figure 4-14 Variable mapping of analog input module

2. After compiling, log in to download the project and run it.
3. Variable description: the following table uses channel 0 as an example to illustrate the use of all variables for channel 0.

Table 4-4 Channel 0 variable description

<table>
<tr><th colspan="3">Parameters</th><th>Value</th><th>Valid bit</th><th>Variable name</th><th>Variable type</th></tr>
<tr><td rowspan="4">Filter</td><td colspan="2">sinc5+sinc1</td><td>00</td><td rowspan="4">[1:0]</td><td rowspan="4">FP</td><td rowspan="9">WORD</td></tr>
<tr><td colspan="2">sinc5+sinc1+enhance50/60</td><td>01</td></tr>
<tr><td colspan="2">sinc3</td><td>10</td></tr>
<tr><td colspan="2">Reserved</td><td></td></tr>
<tr><td rowspan="5">Channel 0 configuration</td><td rowspan="2">Enable channel 0</td><td>Enable</td><td>1</td><td rowspan="2">[0]</td><td rowspan="5">CH0</td></tr>
<tr><td>Disable</td><td>0</td></tr>
<tr><td rowspan="2">Disconnection detection</td><td>Enable</td><td>1</td><td rowspan="2">[1]</td></tr>
<tr><td>Disable</td><td>0</td></tr>
<tr><td>Conversion</td><td>0V–5V</td><td>000</td><td>[4:2]</td></tr>
</table>

Parameters			Value	Valid bit	Variable name	Variable type
	mode	0V–10V	001			
		-5–5V	010			
		-10V–10V	011			
		-20mA–20mA	100			
		0mA–20mA	101			
		4mA–20mA	110			
	Over-limit flag	Enable	1	[5]		
		Disable	0			
	Over range detection enable bit	Enable	1	[6]		
		Disable	0			
	Reserved			[15:7]		
Channel 0 data	Data			[15:0]	IN0	
Channel 0 fault code (See Table 4-6 for details)	Indicates the current fault information of the module.			[15:0]	IN0_Fault_Code	

Table 4-5 Mapping of rated range and actual input analog value

Type	Input rated range	Mapped digital value
Analog voltage input	-10V–10V	-10000–+10000
	0V–10V	0–10000
	-5V–+5V	- 5000–+5000
	0V–5V	0–5000
Analog current input	-20mA–20mA	-20000–20000
	0mA–20mA	0–20000
	4mA–20mA	4000–20000

Table 4-6 Channel fault code

Channel 0	Meaning
A0	Channel 0 is disconnected.
A1	Channel 0 exceeds the limits (exceeds the range of -25V–+25V)
A2	Channel 0 exceeds the upper limit of the range (exceeds the upper limit of the currently selected voltage range)
A3	Channel 0 exceeds the lower limit of the range (exceeds the lower limit of the currently selected voltage range)

Channel 1	Meaning
A4	Channel 1 is disconnected.
A5	Channel 1 exceeds the limits (exceeds the range of -25V–+25V)
A6	Channel 1 exceeds the upper limit of the range (exceeds the upper limit of the currently selected voltage range)
A7	Channel 1 exceeds the lower limit of the range (exceeds the lower limit of the currently selected voltage range)

Channel 2	Meaning
A8	Channel 2 is disconnected.
A9	Channel 2 exceeds the limits (exceeds the range of -25V–+25V)
AA	Channel 2 exceeds the upper limit of the range (exceeds the upper limit of the

Channel 2	Meaning
	currently selected voltage range)
Ab	Channel 2 exceeds the lower limit of the range (exceeds the lower limit of the currently selected voltage range)

Channel 3	Meaning
AC	Channel 3 is disconnected.
Ad	Channel 3 exceeds the limits (exceeds the range of -25V–+25V)
AE	Channel 3 exceeds the upper limit of the range (exceeds the upper limit of the currently selected voltage range)
AF	Channel 3 exceeds the lower limit of the range (exceeds the lower limit of the currently selected voltage range)

4.6.4 Analog output module

EtherCAT remote extension module (AX-EM-Rcm-ET) is used to extend the analog output module (AX-EM-4DA) through the backplane. The instructions are as follows:

1. Right click **AX-EM-ECM-ET** in the device panel to add the analog output module (AX_EM_4DA). Control the module through the multiple sets of variables in the Module/IO mapping tab, as shown in the following figure.

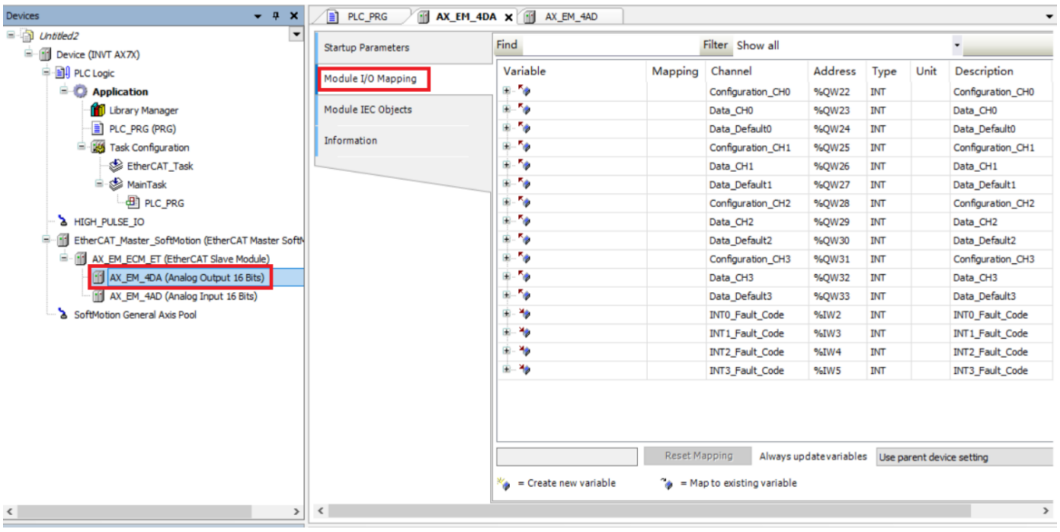


Figure 4-15 Variable mapping of analog output module

2. After compiling, log in to download the project and run it.

3. Variable description: the following table uses channel 0 as an example to illustrate the use of all variables for channel 0.

Table 4-7 Channel 0 variable description

Parameters			Value	Valid bit	Variable name
Channel 0 configuration	Enable channel 0	Enable	1	[0]	Configuration_CH0
		Disable	0		
	Disconnection detection	Reserved		[1]	
	Conversion mode	0V–5V	000	[4:2]	
		0V–10V	001		

Parameters			Value	Valid bit	Variable name
		-5V–5V	010		
		-10V–10V	011		
		4mA–20mA	100		
		0mA–20mA	101		
	Output status after stop	Clear output	00	[6:5]	
		Keep output	01		
		Output preset value	10		
	Reserved			[15:7]	
Channel 0 code value	Data			[15:0]	Data_CH0
Channel 0 output preset value	Output preset value			[15:0]	Data_Default0
Channel 0 fault code (See Table 4-9 for details)	Indicates the current fault information of the module.			[15:0]	INT0_Fault_Code

Table 4-8 Mapping of rated range and actual input analog value

Type	Input rated range	Mapped digital value
Analog voltage output	-10V–10V	-10000–+10000
	0V–10V	0–10000
	-5V–5V	-5000–+5000
	0V–5V	0–5000
Analog current output	4mA–20mA	4000–20000
	0mA–20mA	0–20000

Table 4-9 Channel fault code

Channel 0	Meaning
B0	The current output of channel 0 is disconnected.
B1	The voltage output of channel 0 is short-circuited.

Channel 1	Meaning
B2	The current output of channel 1 is disconnected.
B3	The voltage output of channel 1 is short-circuited.

Channel 2	Meaning
B4	The current output of channel 2 is disconnected.
B5	The voltage output of channel 2 is short-circuited.

Channel 3	Meaning
B6	The current output of channel 3 is disconnected.
B7	The voltage output of channel 3 is short-circuited.

Output module power failure	Meaning
B8	The 24V power board of the output module is disconnected.

4.6.5 Temperature module

EtherCAT remote extension module (AX-EM-Rcm-ET) is used to extend the temperature module (AX-EM-4PTC) through the backplane. The instructions are as follows:

1. Right click **AX-EM-ECM-ET** in the device panel to add the temperature module (AX_EM_4PTC). Control the module through the multiple sets of variables in the Module/IO mapping tab, as shown in the following figure.

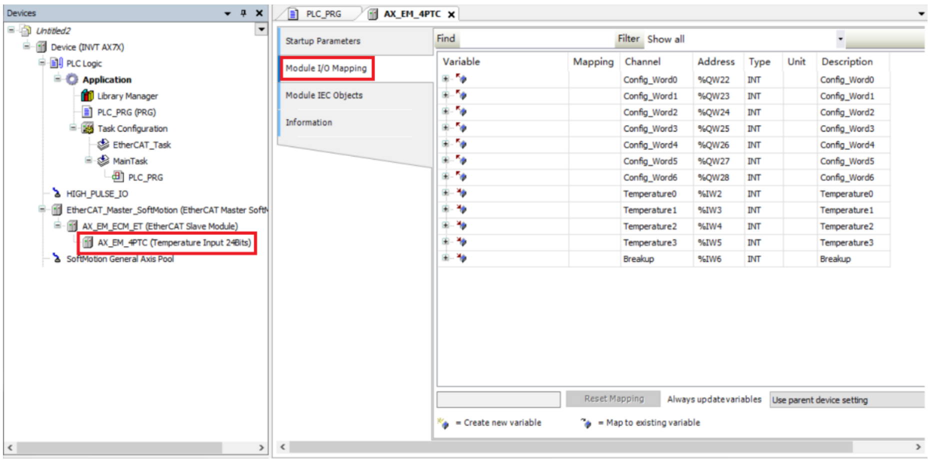


Figure 4-16 Variable mapping of temperature module

2. After compiling, log in to download the project and run it.
3. Variable description: the following tables describe the use of all variables for the four channels.

Table 4-10 Variable description

Note: The channel disconnection detection function and detection results are reserved.

<table>
<tr><th colspan="2">Parameters</th><th>Value</th><th>Valid bit</th><th>Variable name</th></tr>
<tr><td colspan="2">Temperature of channel 0</td><td></td><td>[15:0]</td><td>Temperature0</td></tr>
<tr><td colspan="2">Temperature of channel 1</td><td></td><td>[15:0]</td><td>Temperature1</td></tr>
<tr><td colspan="2">Temperature of channel 2</td><td></td><td>[15:0]</td><td>Temperature2</td></tr>
<tr><td colspan="2">Temperature of channel 3</td><td></td><td>[15:0]</td><td>Temperature3</td></tr>
<tr><td rowspan="2">Disconnection detection result of channel 0</td><td>Normal</td><td>00</td><td rowspan="2">[1:0]</td><td rowspan="8">Breakup</td></tr>
<tr><td>Disconnected</td><td>01</td></tr>
<tr><td rowspan="2">Disconnection detection result of channel 1</td><td>Normal</td><td>00</td><td rowspan="2">[3:2]</td></tr>
<tr><td>Disconnected</td><td>01</td></tr>
<tr><td rowspan="2">Disconnection detection result of channel 2</td><td>Normal</td><td>00</td><td rowspan="2">[9:8]</td></tr>
<tr><td>Disconnected</td><td>01</td></tr>
<tr><td rowspan="2">Disconnection detection result of channel 3</td><td>Normal</td><td>00</td><td rowspan="2">[11:10]</td></tr>
<tr><td>Disconnected</td><td>01</td></tr>
<tr><td rowspan="2">Enable channel 0</td><td>Enable</td><td>1</td><td rowspan="2">[0]</td><td rowspan="13">Config_Word0</td></tr>
<tr><td>Disable</td><td>0</td></tr>
<tr><td rowspan="2">Display mode</td><td>°C</td><td>0</td><td rowspan="2">[1]</td></tr>
<tr><td>°F</td><td>1</td></tr>
<tr><td rowspan="2">Cold junction compensation method</td><td>Internal cold junction compensation</td><td>0</td><td rowspan="2">[2]</td></tr>
<tr><td>External cold junction compensation</td><td>1</td></tr>
<tr><td rowspan="2">Sensor disconnection detection</td><td>Enable</td><td>1</td><td rowspan="2">[3]</td></tr>
<tr><td>Disable</td><td>0</td></tr>
<tr><td rowspan="2">Over-limit detection</td><td>Enable</td><td>1</td><td rowspan="2">[4]</td></tr>
<tr><td>Disable</td><td>0</td></tr>
<tr><td>Sensor type</td><td>B
E
J</td><td>000
001
010</td><td>[11:8]</td></tr>
</table>

Parameters		Value	Valid bit	Variable name
	K	011		
	N	100		
	R	101		
	S	110		
	T	111		
	PT100	1000		
	PT500	1001		
	PT1000	1010		
	CU500	1011		
	2-Wire 3-Wire 4-Wire (For RTD only)	00 01 10	[13:12]	
Filter time	0–100	0–100	[6:0]	Config_Word1
Enable channel 1	Enable	1	[8]	
	Disable	0		
Display mode	°C	0	[9]	
	°F	1		
Cold junction compensation method	Internal cold junction compensation	0	[10]	
	External cold junction compensation	1		
Sensor disconnection detection	Enable	1	[11]	
	Disable	0		
Over-limit detection	Enable	1	[12]	
	Disable	0		
Sensor type	B	000	[3:0]	Config_Word2
	E	001		
	J	010		
	K	011		
	N	100		
	R	101		
	S	110		
	T	111		
	PT100	1000		
	PT500	1001		
	PT1000	1010		
	CU500	1011		
	2-Wire 3-Wire 4-Wire (For RTD only)	00 01 10	[5:4]	
Filter time	0–100	0–100	[14:8]	
Enable channel 2	Enable	1	[0]	Config_Word3
	Disable	0		
Display mode	°C	0	[1]	
	°F	1		
Cold junction compensation method	Internal cold junction compensation	0	[2]	

Parameters		Value	Valid bit	Variable name
	External cold junction compensation	1		
Sensor disconnection detection	Enable	1	[3]	
	Disable	0		
Over-limit detection	Enable	1	[4]	
	Disable	0		
Sensor type	B E J K N R S T PT100 PT500 PT1000 CU500	000 001 010 011 100 101 110 111 1000 1001 1010 1011	[11:8]	
	2-Wire 3-Wire 4-Wire (For RTD only)	00 01 10	[13:12]	
Filter time	0–100	0–100	[6:0]	Config_Word4
Enable channel 3	Enable	1	[8]	
	Disable	0		
Display mode	°C	0	[9]	
	°F	1		
Cold junction compensation method	Internal cold junction compensation	0	[10]	
	External cold junction compensation	1		
Sensor disconnection detection	Enable	1	[11]	
	Disable	0		
Over-limit detection	Enable	1	[12]	
	Disable	0		
Sensor type	B E J K N R S T PT100 PT500 PT1000 CU500	000 001 010 011 100 101 110 111 1000 1001 1010 1011	[3:0]	Config_Word5
	2-Wire 3-Wire	00 01	[5:4]	

Parameters		Value	Valid bit	Variable name
	4-Wire (For RTD only)	10		
Filter time	0–100	0–100	[14:8]	
Sampling period of channel 0	250ms 500ms 1000ms	01 10 11	[1:0]	Config_Word6
Sampling period of channel 1	250ms 500ms 1000ms	01 10 11	[3:2]	
Sampling period of channel 2	250ms 500ms 1000ms	01 10 11	[5:4]	
Sampling period of channel 3	250ms 500ms 1000ms	01 10 11	[7:6]	

Table 4-11 Supported sensor types and measurement range

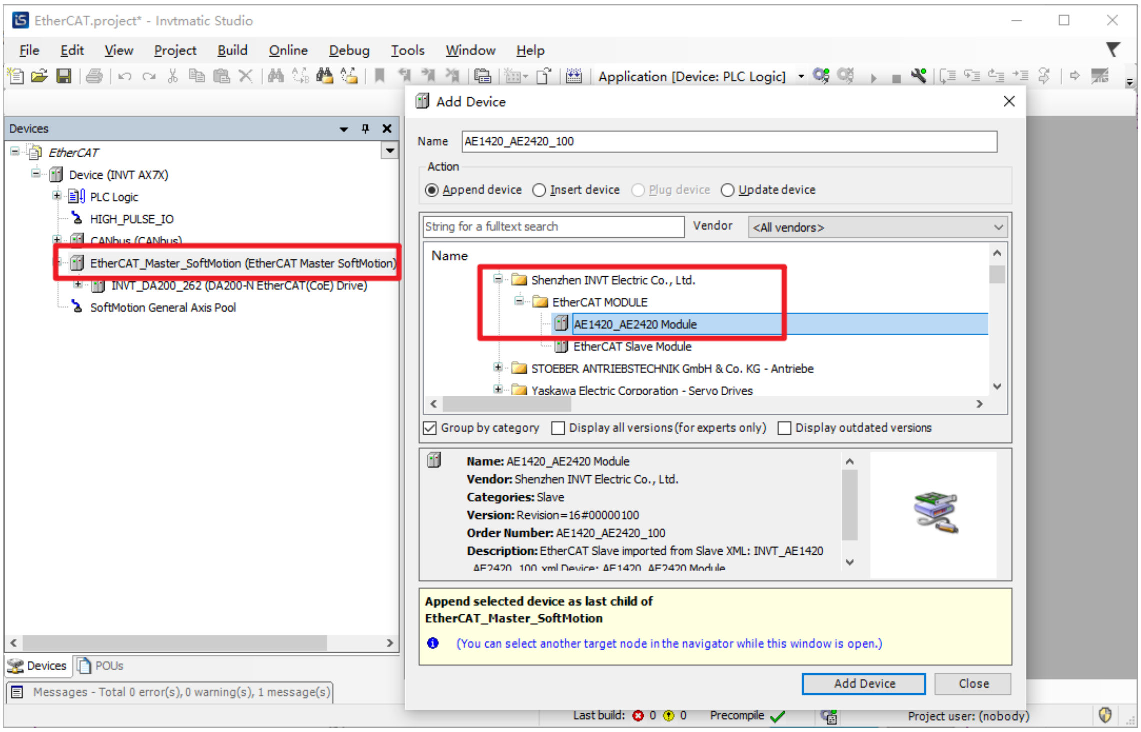
Item	Sensor name	Temperature range in Celsius	Temperature range in Fahrenheit
Thermal resistor type	PT100	-200.0°C–850°C	-328.0°F–1562.0°F
	PT500	-200.0°C–850°C	-328.0°F–1562.0°F
	PT1000	-200.0°C–850°C	-328.0°F–1562.0°F
	CU100	-50.0°C–150°C	-58.0°F–302.0°F
Thermocouples type	B	200.0°C–1800°C	392.0°F–3272.0°F
	E	-270.0°C–1000°C	-454.0°F–1832.0°F
	N	-200.0°C–1300°C	-328.0°F–2372.0°F
	J	-210.0°C–1200°C	-346.0°F–2192.0°F
	K	-270.0°C–1370°C	-454.0°F–2498.0°F
	R	-50.0°C–1765°C	-58.0°F–3209.0°F
	S	-50.0°C–1765°C	-58.0°F–3209.0°F
	T	-270.0°C–400°C	-454.0°F–752.0°F

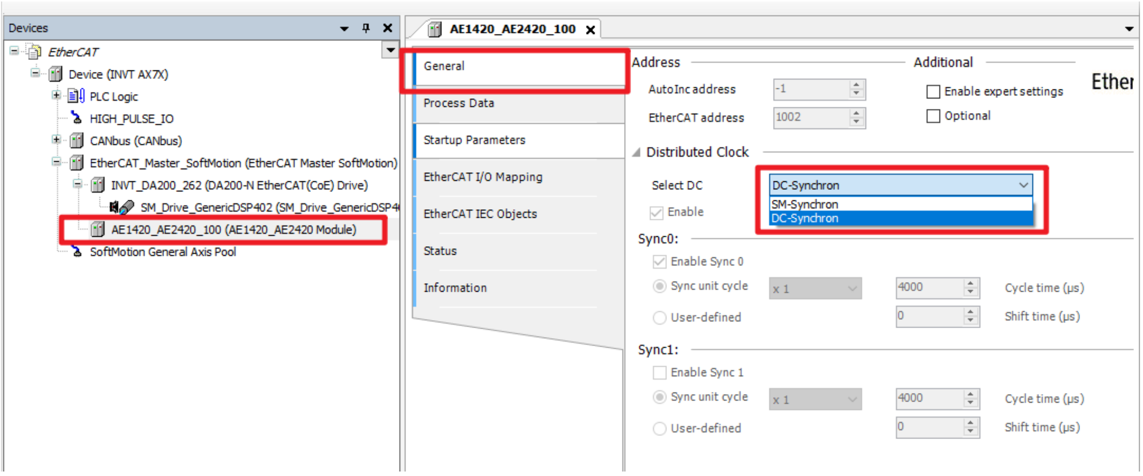
4.7 Distributed I/O module

The distributed I/O AE1420/AE2420 module is EtherCAT slave module with 32 digital inputs/32 digital outputs. Before using the module, you need to install the device profile **INVT_AE1420_AE2420_xxx.xml** for the module.

4.7.1 Creating a project for distributed I/O module

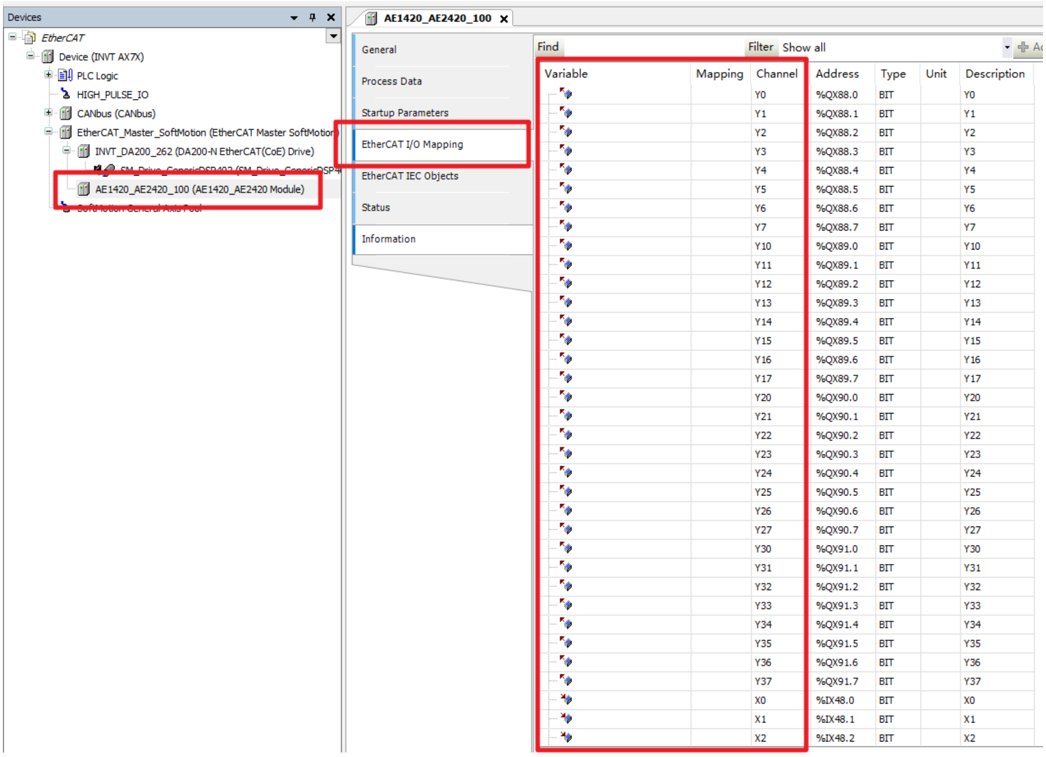
Create a distributed I/O module application, and install the device profile **INVT_AE1420_AE2420_xxx.xml** required by the module. Right click **EtherCAT_Master_SoftMotion** in the device panel to add the AE1420_AE2420 module.





Click the **AE1420_AE2420** module in the device panel, and configure the synchronization method in the **General** interface.

In the **EtherCAT I/O Mapping** interface, the output points are accessed via Y0–Y7/Y10–Y17/Y20–Y27/Y30–Y37, and the input points are accessed via X0–X7/X10–X17/X20–X27/X30–X37.



4.8 Priority setting of each module (recommended value)

4.8.1 Setting priority

If the created project contains multiple functional modules, create multiple tasks and set the task priority as follows. Table 4-12 shows the recommended values for task priority.

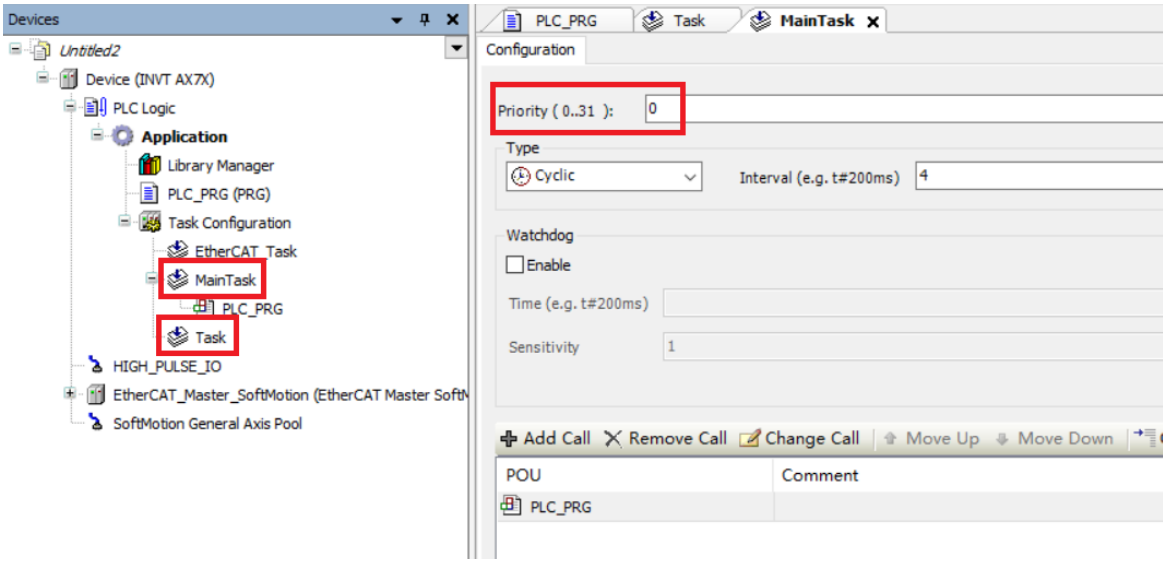


Figure 4-17 Example of task project priority settings

Table 4-12 Setting priority

Function module	Recommended priority
PlcCfg module	31
ModbusTCP	15–30
ModbusRTU	15–30
High-speed I/O	1–15
Analog input/output	1–15

Function module	Recommended priority
Temperature module	1–15
EtherCAT	0

4.8.2 Configuring sub-device bus cycle options

Under the **Controller settings** > **Bus cycle** > **Bus cycle task** of the AX7X device, the Bus cycle task list provides the tasks defined in the task configuration of the current valid project (such as "MainTask", "EtherCAT Master"). Select one of the tasks as the bus cycle of the current project, or select the option **<unspecified>**, which indicates that the shortest task cycle time or the fastest execution cycle will be applied. You can switch to another settings, but be sure to note the following.

Note: Before modifying the <**unspecified**> setting, be aware that it is a default action defined by the device description. By default, the task can be defined with a shortest cycle time or a longest cycle time. Please check this carefully before applying this setting.

To improve the stability of the system when using expansion modules and EtherCAT modules (especially the EtherCAT_Master_SoftMotion module), you should select the task corresponding to each module in **EtherCAT I/O Mapping** > **Bus Cycle Options**. The reference program is as follows.

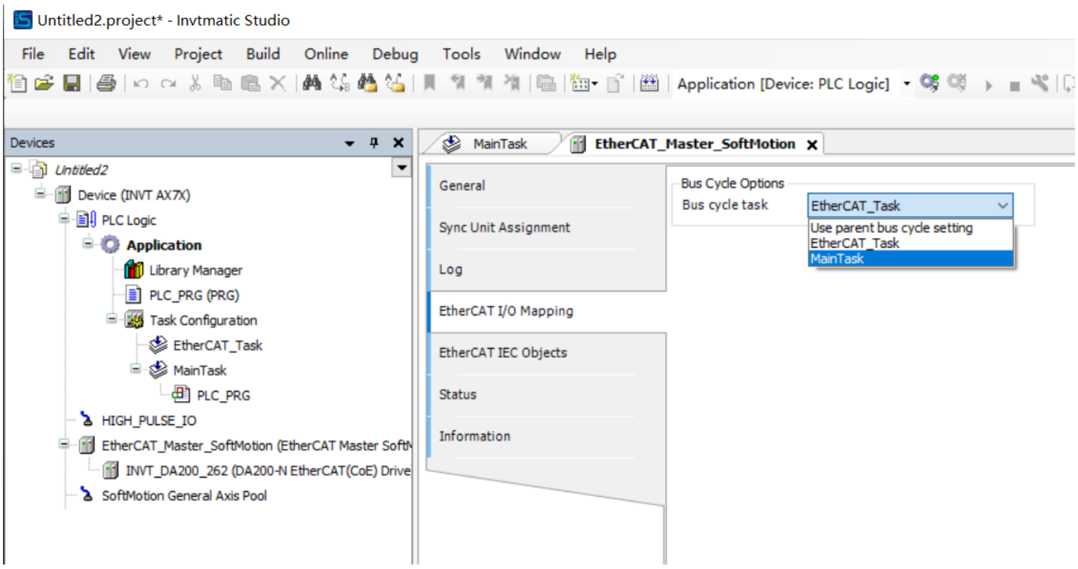


Figure 4-18 Expansion module bus cycle task setting

5 Device Diagnosis

AX series equipment diagnostic information is reflected in three ways, namely fault indicator, digital tube and diagnostic code. Fault indicators show the system and bus error. Digital tubes display the fault code of a specific function module. Diagnostic codes further indicate the specific types of faults, which can be generally searched by upper computer software.

5.1 Fault indicator

The AX series fault indicator is mainly composed of two parts. The first part is mainly the system and bus indicator lights. The second part is mainly digital tube indicators.

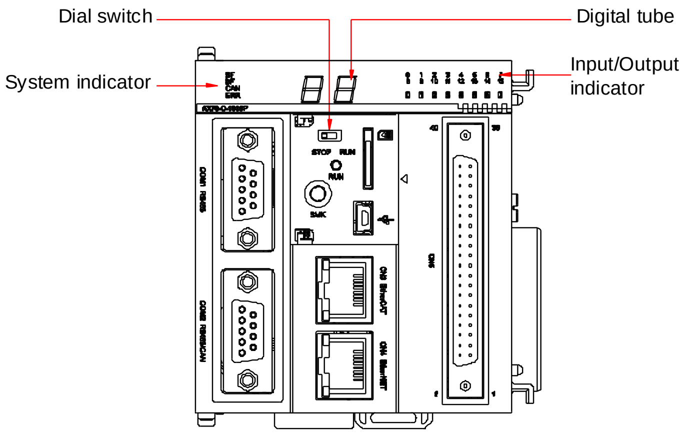


Figure 5-1 Fault indicator diagram

5.1.1 System and bus fault indicator

Table 5-1 System and bus fault indicator

Fault indicator name	Error type	Error content
SF	System fault	Codesys is not started
BF	Bus communication fault	Modbus RTU/Modbus TCP/backplane bus fault
CAN	CAN bus fault	Reserved
ERR	Module fault	Reserved

Note: When connecting multiple programmable controllers, you can click the **Wink** button on the software platform to observe the simultaneous flashing of the SF, BF, CAN, and ERR indicators to identify the device.

5.1.2 High-speed input/output indicator

If the output/input of the port is valid, the indicator corresponding to the port is on, and if the output/input is invalid, the corresponding indicator is off.

5.2 Digital tube fault code

Digital tube Fault code	Module	Fault type	Solution
16#10	CPU module PlcCfg	Error setting local new IP	Check the underlying network configuration file.
16#11		Error setting local new subnet mask	Check the underlying network configuration file.
16#12		Failed to read the local IP and subnet mask	Check the underlying network configuration file.
16#13		Abnormal time setting format	Check the time setting format.
16#14		Error setting motion controller time	Check the underlying code.
16#15		Error getting motion controller real time	The controller button battery voltage is insufficient, replace the battery.
16#20	COM1 485 ModbusRTU_Slave1	Failed to open serial port COM1	Check whether the underlying serial port number corresponds to the hardware.
16#21		Baud rate setting failed	Check the baud rate setting of the slave node
16#22		Data bit, stop bit or parity bit setting failed	Check the specific error code of Invtmatic Studio ErrorID. Data bit: ErrorID=3, check bit ErrorID=4, stop bit ErrorID=5.
16#23		Slave function enable failed	System error Err_Sym, or slave enable is turned on.
16#24		Slave read and write error	Check detailed parameter settings
16#25	COM1 485 ModbusRTU_Master 1	Failed to open serial port COM1	Check whether the underlying serial port number corresponds to the hardware.
16#26		SlaveID setting failed	Check the SlaveID number settings of the master node.
16#27		Data bit, stop bit or parity bit setting failed	Check whether the data bit setting value is 7 or 8, whether the check bit is 0, 1 or 2, and whether the stop bit is 1 or 2.
16#28		Master function enable failed	System error Err_Sym, or master enable is turned on.
16#29		One of the the following goes wrong: master read/write coil, read holding register, write a single register, write multiple registers	Check that the master-slave initialization parameter configuration is consistent and that the hardware connection is correct.
16#2A		Two function block enabled at the same time.	Ensure that only one of the function block is enabled in the program

Digital tube Fault code	Module	Fault type	Solution
16#30	COM2 485 ModbusRTU_Slave2	Failed to open serial port COM2	Check whether the underlying serial port number corresponds to the hardware.
16#31		Baud rate setting failed	Check the baud rate setting of the slave node
16#32		Data bit, stop bit or parity bit setting failed	Check the specific error code of Invtmatic Studio ErrorID. Data bit: ErrorID=3, check bit ErrorID=4, stop bit ErrorID=5.
16#33		Slave function enable failed	System error Err_Sym, or slave enable is turned on.
16#34		Slave read and write error	Check detailed parameter settings
16#35	COM2 485 ModbusRTU_Master 2	Failed to open serial port COM2	Check whether the underlying serial port number corresponds to the hardware.
16#36		SlaveID setting failed	Check the SlaveID number settings of the master node.
16#37		Data bit, stop bit or parity bit setting failed	Check whether the data bit setting value is 7 or 8, whether the check bit is 0, 1 or 2, and whether the stop bit is 1 or 2.
16#38		Master function enable failed	System error Err_Sym, or master enable is turned on.
16#39		One of the the following goes wrong: master read/write coil, read holding register, write a single register, write multiple registers	Check that the master-slave initialization parameter configuration is consistent and that the hardware connection is correct.
16#3A		Two function block enabled at the same time.	Ensure that only one of the function block is enabled in the program
16#60	modbusTCP_Slave	Error configuring slave IP	Check the underlying corresponding configuration.
16#61		Port setting error	Check the port settings
16#62		Failed to listen to sockets (failed to create socket, failed to bind socket, failed to listen to socket)	Check the corresponding configuration.
16#63		Failed to accept client	Check the corresponding configuration.
16#64		Failed to accept client data	Check the corresponding configuration.
16#65		Modbus reply error (modbus_reply)	Check the corresponding configuration.
16#66	modbusTCP_Master	Error setting slave IP or port	Check the IP setting or whether it is the default unit number.
16#67		Failed to set slave node	Check the parameter settings.
16#68		Failed to connect slave node	Check the parameter settings, such as slave IP or port.
16#69		Write slave register failure	Check the parameter settings.

Digital tube Fault code	Module	Fault type	Solution
16#6A		Read slave register failure	Check the parameter settings.
16#A0	Analog output module AX-EM-4AD	Channel 0 is disconnected.	Check whether the wires are connected properly.
16#A1		Channel 0 exceeds the limits (that is, the voltage exceeds the range of -25V–+25V, and the current exceeds the range of -104mA–104mA)	Check if the input voltage (current) is out of range.
16#A2		Channel 0 exceeds the upper limit of the range (exceeds the upper limit of the currently selected voltage range)	Reduce the input voltage (current) value, or use a wider range of conversion modes.
16#A3		Channel 0 exceeds the lower limit of the range (exceeds the lower limit of the currently selected voltage range)	Increase the input voltage (current) value, or use a wider range of conversion modes.
16#A4		Channel 1 is disconnected.	Check whether the wires are connected properly.
16#A5		Channel 1 exceeds the limits (that is, the voltage exceeds the range of -25V–+25V, and the current exceeds the range of -104mA–104mA)	Check if the input voltage (current) is out of range.
16#A6		Channel 1 exceeds the upper limit of the range (exceeds the upper limit of the currently selected voltage range)	Reduce the input voltage (current) value, or use a wider range of conversion modes.
16#A7		Channel 1 exceeds the lower limit of the range (exceeds the lower limit of the currently selected voltage range)	Increase the input voltage (current) value, or use a wider range of conversion modes.
16#A8		Channel 2 is disconnected.	Check whether the wires are connected properly.
16#A9		Channel 2 exceeds the limits (that is, the voltage exceeds the range of -25V–+25V, and the current exceeds the range of -104mA–104mA)	Check if the input voltage (current) is out of range.
16#AA		Channel 2 exceeds the upper limit of the range (exceeds the upper limit of the currently selected voltage range)	Reduce the input voltage (current) value, or use a wider range of conversion modes.
16#Ab		Channel 2 exceeds the lower limit of the range (exceeds the lower limit of the currently selected voltage range)	Increase the input voltage (current) value, or use a wider range of conversion modes.
16#AC		Channel 3 is disconnected.	Check whether the wires are connected properly.
16#Ad		Channel 3 exceeds the limits (that is, the voltage exceeds the range of -25V–+25V, and the current exceeds the range of -104mA–	Check if the input voltage (current) is out of range.

Digital tube Fault code	Module	Fault type	Solution
		104mA)	
16#AE		Channel 3 exceeds the upper limit of the range (exceeds the upper limit of the currently selected voltage range)	Reduce the input voltage (current) value, or use a wider range of conversion modes.
16#AF		Channel 3 exceeds the lower limit of the range (exceeds the lower limit of the currently selected voltage range)	Increase the input voltage (current) value, or use a wider range of conversion modes.
16#b0	Analog output module AX-EM-4DA	The current output of channel 0 is disconnected.	Check whether the current channel is disconnected and reconnect it if it is
16#b1		The voltage output of channel 0 is short-circuited.	Check whether the voltage channel is short-circuited. If so, restore it to normal.
16#b2		The current output of channel 1 is disconnected.	Check whether the current channel is disconnected and reconnect it if it is
16#b3		The voltage output of channel 1 is short-circuited.	Check whether the voltage channel is short-circuited. If so, restore it to normal.
16#b4		The current output of channel 2 is disconnected.	Check whether the current channel is disconnected and reconnect it if it is
16#b5		The voltage output of channel 2 is short-circuited.	Check whether the voltage channel is short-circuited. If so, restore it to normal.
16#b6		The current output of channel 3 is disconnected.	Check whether the current channel is disconnected and reconnect it if it is
16#b7		The voltage output of channel 3 is short-circuited.	Check whether the voltage channel is short-circuited. If so, restore it to normal.
16#b8		The 24V power board of the output module is disconnected.	Check whether the 24V power supply is normal and whether there is reverse connection.
16#C0	Temperature module AX-EM-4PTC	Channel 0 exceeds the upper limit of range (the actual temperature exceeds the set upper limit)	Check whether the set temperature upper limit is greater than the actual value.
16#C1		Channel 0 exceeds the lower limit of range (the actual temperature exceeds the set lower limit)	Check whether the set temperature lower limit is smaller than the actual value.
16#C2		Channel 1 exceeds the upper limit of range (the actual temperature exceeds the set upper limit)	Check whether the set temperature upper limit is greater than the actual value.
16#C3		Channel 1 exceeds the lower limit of range (the actual temperature exceeds the set lower limit)	Check whether the set temperature lower limit is smaller than the actual value.
16#C4		Channel 2 exceeds the upper limit	Check whether the set

Digital tube Fault code	Module	Fault type	Solution
		of range (the actual temperature exceeds the set upper limit)	temperature upper limit is greater than the actual value.
16#C5		Channel 2 exceeds the lower limit of range (the actual temperature exceeds the set lower limit)	Check whether the set temperature lower limit is smaller than the actual value.
16#C6		Channel 3 exceeds the upper limit of range (the actual temperature exceeds the set upper limit)	Check whether the set temperature upper limit is greater than the actual value.
16#C7		Channel 3 exceeds the lower limit of range (the actual temperature exceeds the set lower limit)	Check whether the set temperature lower limit is smaller than the actual value.
16#C8		Over-limit setting error (set upper limit is smaller than the lower limit)	Check whether the set temperature upper limit is greater than the lower limit.
16#C9		Channel 0 is disconnected. (Reserved)	
16#CA		Channel 1 is disconnected. (Reserved)	
16#CB		Channel 2 is disconnected. (Reserved)	
16#CC		Channel 3 is disconnected. (Reserved)	

6 Controller Program Structure and Execution

6.1 Program structure

The software model is represented by a hierarchical structure. Each layer implies many characteristics of the underlying layer. The software model describes the basic software elements and their interrelationships. These software elements contain: devices, applications, tasks, global variables, access paths, and application objects. Figure 6-1 shows their internal structure, which is consistent with the software model of the IEC 61131-3 standard.

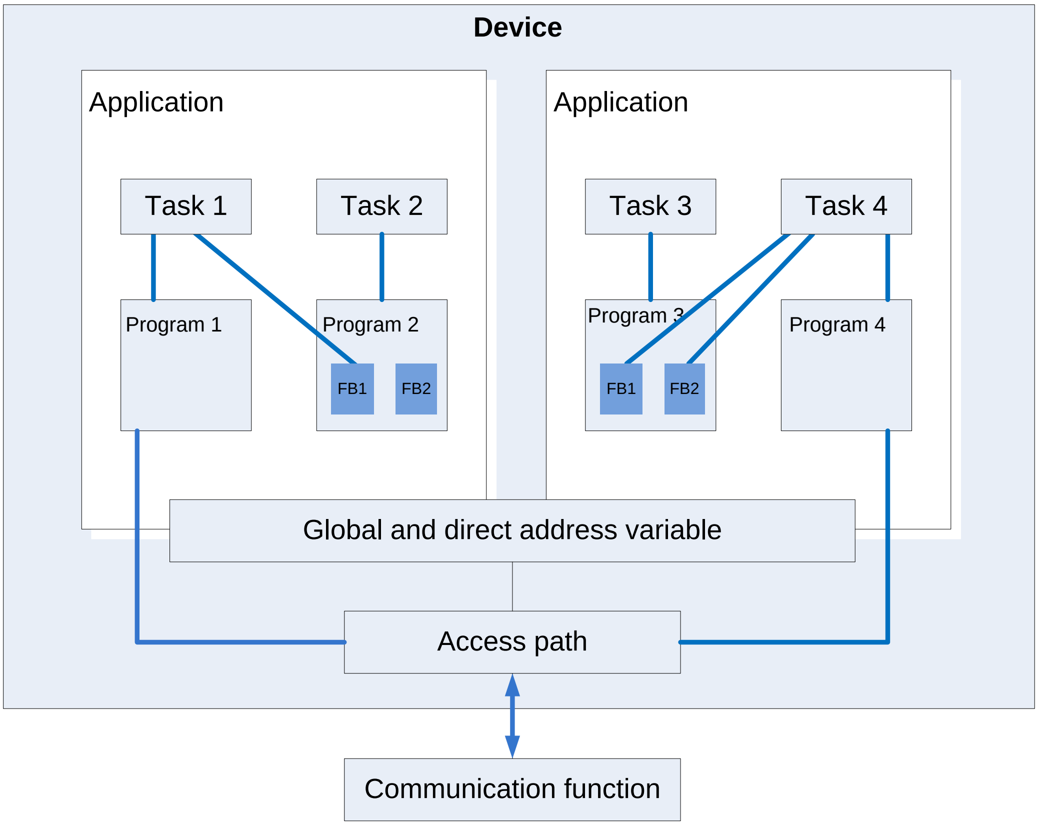


Figure 6-1 Program hierarchical structure

6.2 Task

A program can be written in different programming languages. A typical program consists of a number of interconnected function blocks that can exchange data with each other. The execution of different parts of a program is controlled by "tasks". Tasks can be configured to cause a series of programs or blocks to execute periodically or to be triggered by a specific event to start execution.

The **Task Manager** tab in the device tree can be used to control the execution of other subprograms within the project, in addition to the specific controller_PRG program. A task is used to specify the properties of a program organization unit at run time. It is an execution control element with the ability to be called. Multiple tasks can be established in a task configuration, and multiple program organization units can be called in a task. Once the task is set, it can control the program to execute periodically or to be triggered by a specific event to start execution.

In the task configuration, define it with name, priority, and startup type of the task. This startup type can be defined either by time (cyclic, random) or by the timing of an internal or external trigger task, such as a rising edge of a Boolean global variable or a particular event in the system. For each task, you can set a sequence of programs to be started by the task. If this task is executed in the current cycle, these programs will be processed within one cycle. The combination of priority and conditions will determine the timing of task execution. The task setting interface is shown in Figure 6-2.

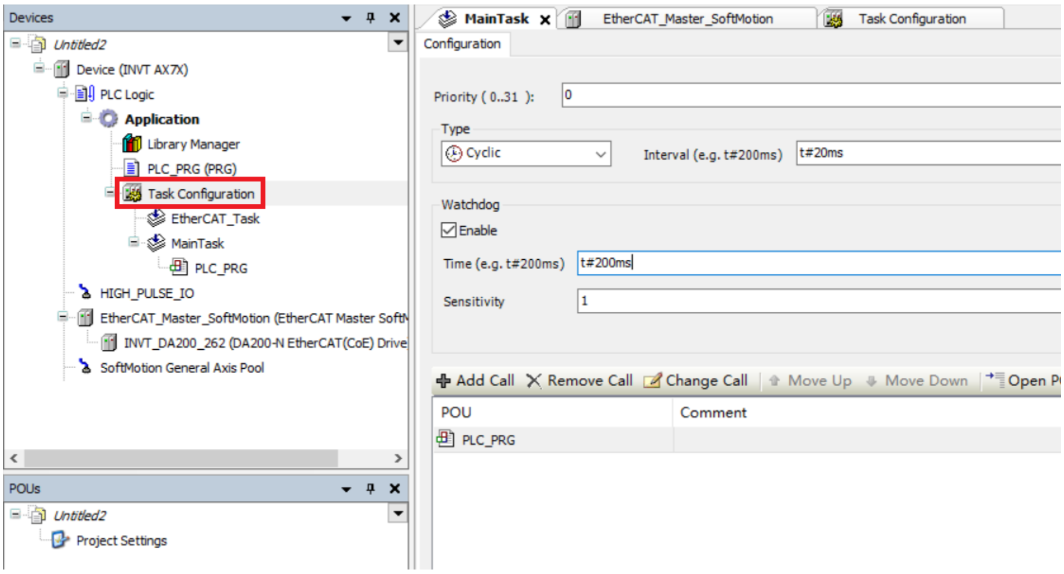


Figure 6-2 Task configuration interface

The programmer must follow the following rules:

1. The maximum number of cyclic tasks is 100.
2. The maximum number of free running tasks is 100.
3. The maximum number of event-triggered tasks is 100.
4. Depending on the target system, the PLC_PRG may be executed as a free program under any circumstances, instead of being manually inserted into the task configuration.
5. Programs are processed and called in a top-down order within the task editor.

6.3 Program execution

The following figure describes in detail the complete process of program execution inside the AX series programmable controller. The main process includes input sampling, program execution and output refresh.

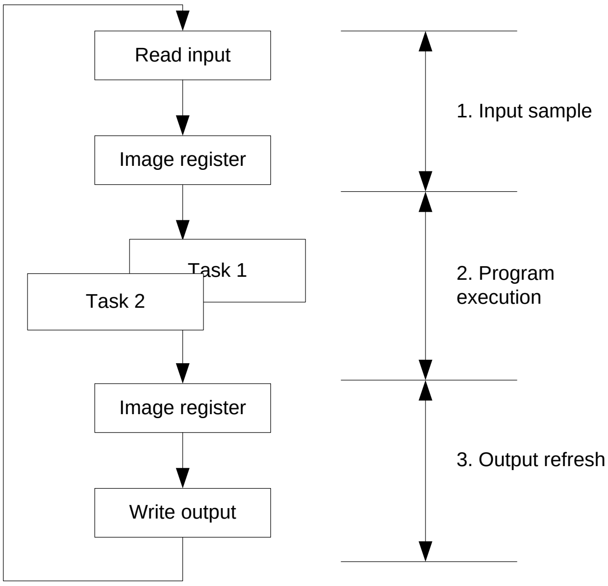


Figure 6-3 Controller execution

1) Input sampling

At the beginning of each scan cycle, the controller detects the state of the input device (such as switch, button) and writes the state to the input image register area. During program execution, the running system reads data from the input image area for program resolution. It is important to note that the input refresh only occurs at the beginning of a scan. During the scan, the input state will not change even if the output state changes.

2) Program execution

During the program execution phase of the scan cycle, the controller reads the status and data from the input image area or output image area and performs logical and arithmetic operations according to the commands. The operation results are stored in the corresponding unit in output image area. In this phase, only the contents in the input image registers remain unchanged, and the contents in other image registers will change with the execution of the program.

3) Output refresh

During the output refresh phase, also known as the write output phase, the controller transmits the state and data in the output image area to the output point, and isolates and amplifies the power in a certain way to drive the external load. The programmable controller completes not only the tasks of the above three phases, but also auxiliary tasks such as internal diagnosis, communication, public processing, and input/output services in a scan cycle.

The AX series programmable controller repeats the process of 1) to 3) above, and the time for each repetition is one work cycle (or scan cycle). It can be seen from the scanning method of the controller that the controller has a shorter scanning time to complete the control task to quickly respond to the change of input and output data, and the duty cycle is generally controlled within the order of ms. Therefore, it is necessary to develop a stable, reliable and fast-response real-time system for AX series programmable controller operation system.

Since the AX series programmable controller adopts a cyclic working mode, the input signal will only be refreshed at the beginning of each cycle, and the output will be concentrated at the end of each cycle. It will inevitably produce a lag between the output signal and the input signal. It takes a while for a signal input to change from the input of the AX series programmable controller to the output of the controller to respond to the change in the input signal. Lag time is an important parameter that should be understood when designing AX series programmable controller control system. Generally, the lag time is related to the following factors:

1. Filter time of the input circuit. It is determined by the time constant of the hardware RC filter circuit. The input lag time can be adjusted by changing the time constant. For example, Table 6-1 shows the technical parameters of the AX-EM-1600D digital input module, where "port filter time" indicates that the filter time of this input module is 10ms.

Table 6-1 AX-EM-1600D Digital input module parameters

Item	Specifications
Input channel	16
Input connection mode	18-point terminal
Input voltage level	24V (up to 30V)
Input current (typical)	4.7mA
ON voltage	>15VDC
OFF voltage	<5VDC
Port filter time	10ms
Input resistance	5.4kΩ
Input signal form	Voltage DC input
Isolation method	Optocoupler
Input dynamic display	When the input is valid, the indicator is on.

2. Lag time of the output circuit. It is related to the output circuit mode. Generally, the lag time of the relay output mode is about 10ms, and the lag time of the transistor output mode is less than 1ms.

3. Working mode of the controller cyclic scanning.

4. Arrangement of statements in the user program.

To allow readers to better understand the whole process, the following is a simple example of the ladder diagram program to show its input and output and how the lagging is produced The program logic is shown in Figure 6-4.

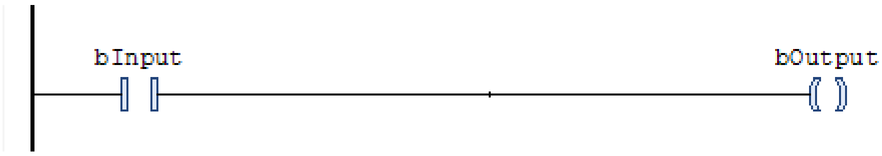


Figure 6-4 AX series programmable controller program

bInput has a hardware mapping relationship with the external input button. When the button is pressed, bInput is ON. bOutput has a hardware mapping relationship with the coil of the external relay. When bOutput is ON, the coil of the relay will also be energized. Within the AX series programmable controller, the handling relationship is shown in Figure 6-6. bInput is not immediately turned ON when the input button is pressed. Because the input sampling is only executed at the beginning of a cycle and the button signal has missed the sampling phase, it usually will be executed at the beginning of the next cycle. In the program in Figure 6-6, the state of bInput is assigned to bOutput. Since there is a certain program calculation during the program running, the bOutput needs a certain processing time of the program to be set to ON. Since the output refresh occurs at the end of the program process, it is at the end of the cycle that the bOutput passes its value to the actual hardware via the output refresh function before the coil is finally energized. The following figure is a relatively ideal state, with the final output having only one cycle of latency.

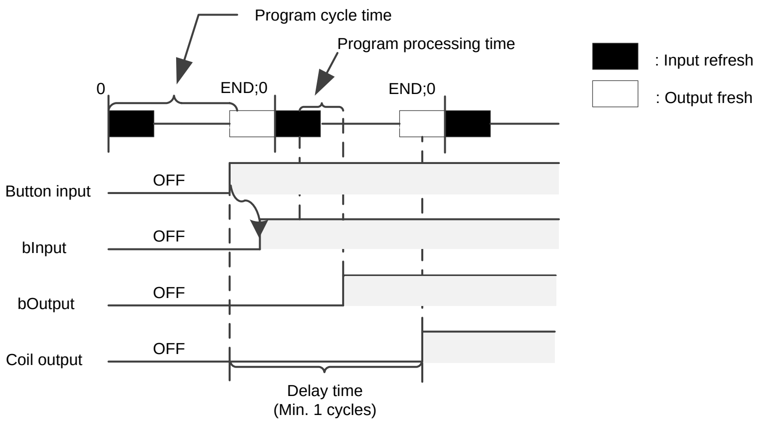


Figure 6-5 Fastest output case

In addition, we should also consider the worse situation. When a cycle of input sampling has just ended, the external input button is ON at this time. Since the input signal needs to be loaded into the input image area at the beginning of the next cycle and the actual output will not be loaded into the output image area until the end of the second cycle, the whole process is shown in Figure 6.7. In this case, the output delay is nearly 2 cycles, which is the output with longest delay time.

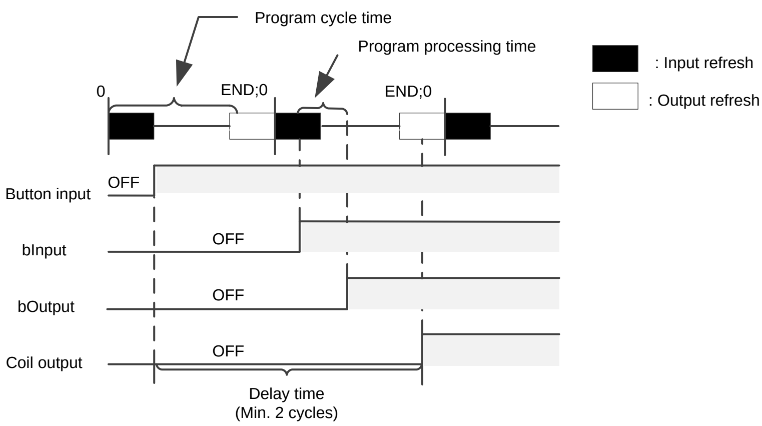


Figure 6-6 Slowest output case

6.4 Task execution type

At the top of the task configuration tree, there is a **Task Configuration** tab, which shows every defined task by their names. The call of POUs for specific tasks is not displayed in the task configuration tree. Each individual task can be edited and configured for the type of execution, which includes Cyclic, Event, Freewheeling, and Status. See Figure 6-7 for details.

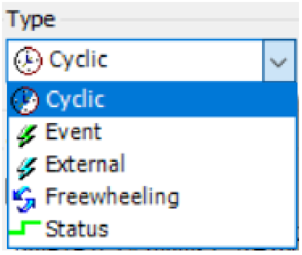


Figure 6-7 Task execution type

1. Cyclic

The processing time of the program will vary depending on whether the commands used in the program are executed or not. Therefore, the actual execution time varies with each scan cycle. By using the cyclic mode, the program can be executed repeatedly for a certain cycle time. Even if the execution time of the program changes, the refresh interval can be maintained. It is recommended that you give priority to the cyclic start mode. For example, if you set the corresponding task to the Cyclic mode and set the interval to 10ms, the actual program execution timing is shown in Figure 6-8.

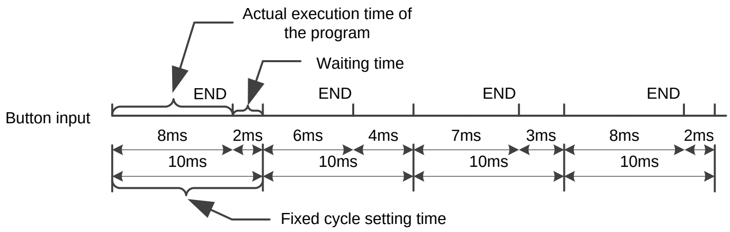


Figure 6-8 Cyclic execution sequence

If the actual execution time of the program is less than the set cyclic time, the remaining time is used for waiting. If there are low-priority tasks in the application that have not been executed, the remaining waiting time is used to execute these tasks. The priority of the task will be described in detail later.

2. Freewheeling

Tasks are processed as soon as the program starts running, and tasks will be automatically restarted in the next cycle after the end of a running cycle. This execution mode is not affected by the program scan cycle. That is to ensure that the last instruction of the program is executed each time before entering the next cycle. Otherwise, the program cycle will not end. Figure 6-9 shows the timing of freewheeling sequence.

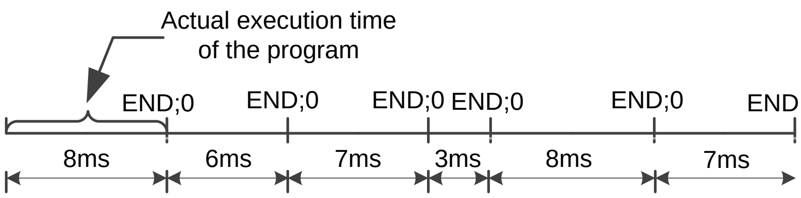


Figure 6-9 Timing of freewheeling sequence

Since the freewheeling execution mode does not have a fixed task time, the execution time may be different each time. Therefore, the real-time performance of the program cannot be guaranteed, and this mode is seldom used in practical applications.

3. Event

If the variable in the event area gets a rising edge, the task begins.

4. Status

If the variable in the event area is TRUE, the task begins. The Status mode is similar to the Event mode, except that the task will be executed when the trigger variable of status triggering is TRUE, and will not be executed when it is FALSE. The event trigger only collects the effective signal of the rising edge of the trigger variable. Figure 6-10 compares the event and status trigger modes, and the green solid line is the Boolean variable status selected by the two modes. Table 6-2 shows the comparison result.

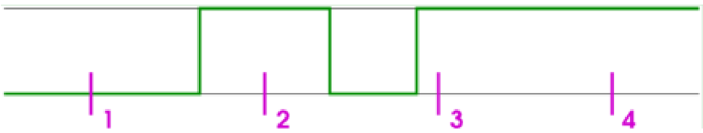


Figure 6-10 Task input trigger signal

Different types of tasks showed different responses at sampling points 1–4 (purple). The trigger condition of Status mode is fulfilled when a specific event is TRUE, but an event-driven task requires the event to change from FALSE to TRUE. If the sampling frequency of the task is too low, the rising edge of the event may not be detected.

Table 6-2 Comparison result between Event and Status trigger modes

Execution point	1	2	3	4
Event	No execute	Execute	Execute	Execute
Status	No execute	Execute	No execute	No execute

6.5 Task priority

1. Task priority setting

You can set the priority of the task, with a total of 32 levels (a number from 0 to 31, with 0 the highest priority and 31 the lowest priority). When a program is executing, tasks with high priority takes precedence over tasks with low priority. A task with high priority 0 can interrupt the execution of lower priority programs in the same resource, so that the execution of the program with low priority is slowed down.

Note: When assigning task priority levels, do not assign tasks with the same priority. If there are other task views that precede tasks with the same priority, the result may be uncertain and unpredictable.

If the task type is "Cyclic", it will be executed in a cycle according to the time set in "Interval". The specific settings are shown in Figure 6-11.

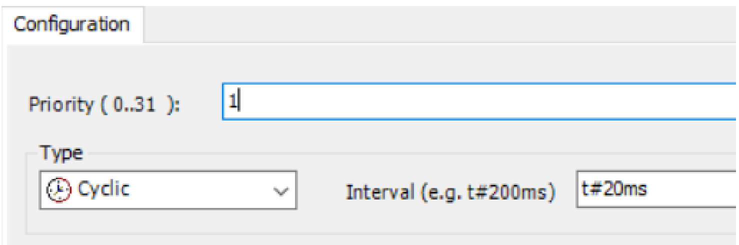


Figure 6-11 Cyclic mode configuration

Example: Suppose there are 3 different tasks with three different priority levels, the specific assignments are as follows.

: Task 1 with **Priority** set to 0 and **Interval** to 10ms

: Task 2 with **Priority** set to 1 and **Interval** to 30ms

: Task 3 with **Priority** set to 2 and **Interval** to 40ms

Inside the controller, the timing relationship of each task is shown in Figure 6-13, and the specific description is as follows:

0–10ms: Execute Task 1 first (highest priority), and if the program is finished within this cycle, the remaining time will be used to execute the Task 2 program. However, if Task 2 has not been fully executed after10ms, Task 2 will be interrupted because Task 1 is executed every 10 milliseconds and has a highest priority.

10–20ms: Execute the programs in Task 1 first. If there is any time left, execute the unfinished Task 2 in the previous cycle.

20–30ms: Since Task 2 is executed every 30ms and Task 2 has been finished within 10–20ms, there is no need to execute task 2 at this time, just execute Task 1 once.

30–40Ms: Similar to before.

40–50ms: Task 3 appears at this time. Since Task 3 has the lowest priority, Task 3 can only be executed after ensuring that Task 2 has been thoroughly executed.

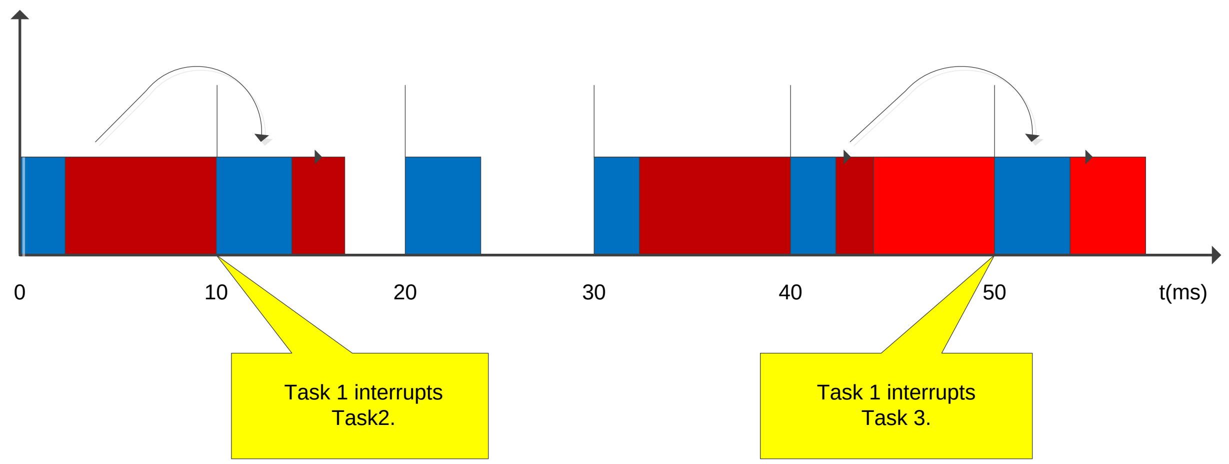


Figure 6-12 Task interrupt execution order

2. AX7x task priority configuration

When the upper computer software of AX series controller creates a new standard project, MainTask is created by default in the task configuration with a priority of 1. The priority of newly created tasks is also 1 by default, but to ensure that important tasks such as motion control are prioritized, the performance of the controller can be used appropriately in some applications that require high-performance motion control (MC). The following table shows the recommended task priority order setting (if there is only one task, the task priority can be set at will):

Table 6-3 Task priority configuration

Task Type	Recommended Priority
PlcCfg module	31
ModbusTCP	15–30
ModbusRTU	15–30
High-speed I/O	1–15
Analog input/output	1–15
Temperature module	1–15
EtherCAT	0

The smaller the priority value, the higher the priority. POU with a higher priority can interrupt the execution of POU with a lower priority, as shown in Figure 6-13, where ECT stands for EtherCAT.

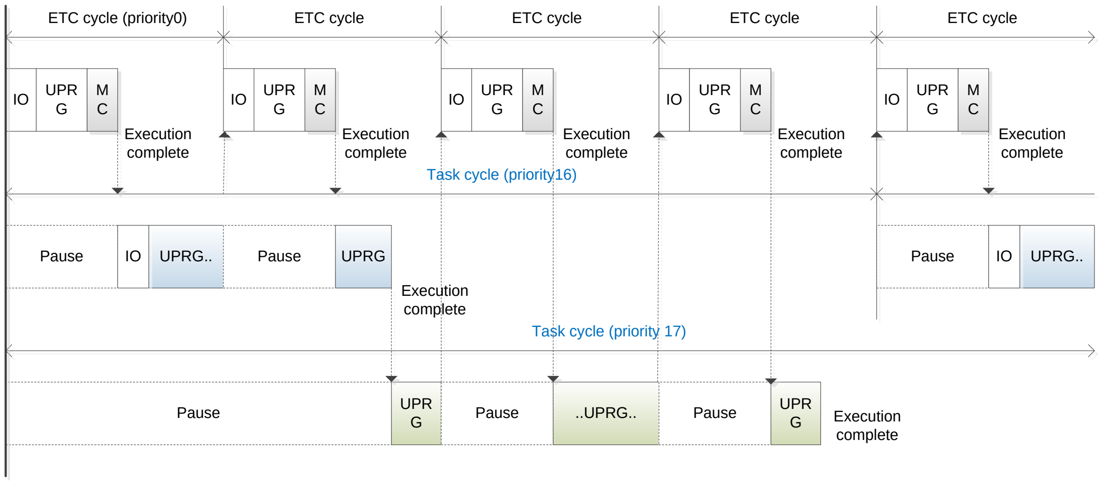


Figure 6-13 POU execution sequence

As shown in Figure 6-13,

When the controller executes a task, there is a time alignment point that is not observed by the user, as shown on the left side of the figure above. Starting at this point, the execution will start in the order of highest priority -> second highest priority -> lowest priority.

A low-priority task may be interrupted by a high-priority task while it is being executed, and when the execution of the high-priority task is complete, the interrupted task with low-priority will continues.

The EtherCAT task is the highest priority task, which is entered according to the EtherCAT cycle, and all POUs within the task are executed once before executing the lower priority task.

3. Requirements for execution cycle setting in task configuration

The AX series system upper computer software uses multitasking to execute the "tasks" of the user program, and each "task" is assigned a different execution cycle. Some global variables may be accessed and modified in different POUs, so the interactive synchronization of global variables should be carried out at the "time alignment point" of the task. For the cycle of a cyclic task setting, the cycle time of different cyclic task types is an integer multiple.

For example, the EtherCAT task cycle time is set to 4ms, 8ms, while the normal cycle is set to 400ms, and the cycle of lower priority is set to 100ms or 200ms. Do not set the EtherCAT task cycle to 5ms, 7ms, 9ms and so on, which may cause non-integer multiple of 2.

4. Configuring sub-device bus cycle options

Under the **Controller settings** > **Bus cycle** > **Bus cycle task** of the controller device, the Bus cycle task list provides the tasks defined in the task configuration of the current valid project (such as "MainTask", "EtherCAT Master"). Select one of the tasks as the bus cycle of the current project, or select the option **<unspecified>**, which means that the shortest task cycle time or the fastest execution cycle will be applied. You can switch to another settings, but be sure to note the following.

Before modifying the <unspecified> setting, be aware that it is a default action defined by the device description. By default, the task can be defined with a shortest cycle time or a longest cycle time. Please check this carefully before applying this setting.

Therefore, select the task corresponding to each module in EtherCAT I/O when using expansion modules and EtherCAT modules (especially the EtherCAT_Master_SoftMotion module) to improve the stability of the system. The reference program is shown in Figure 6-14.

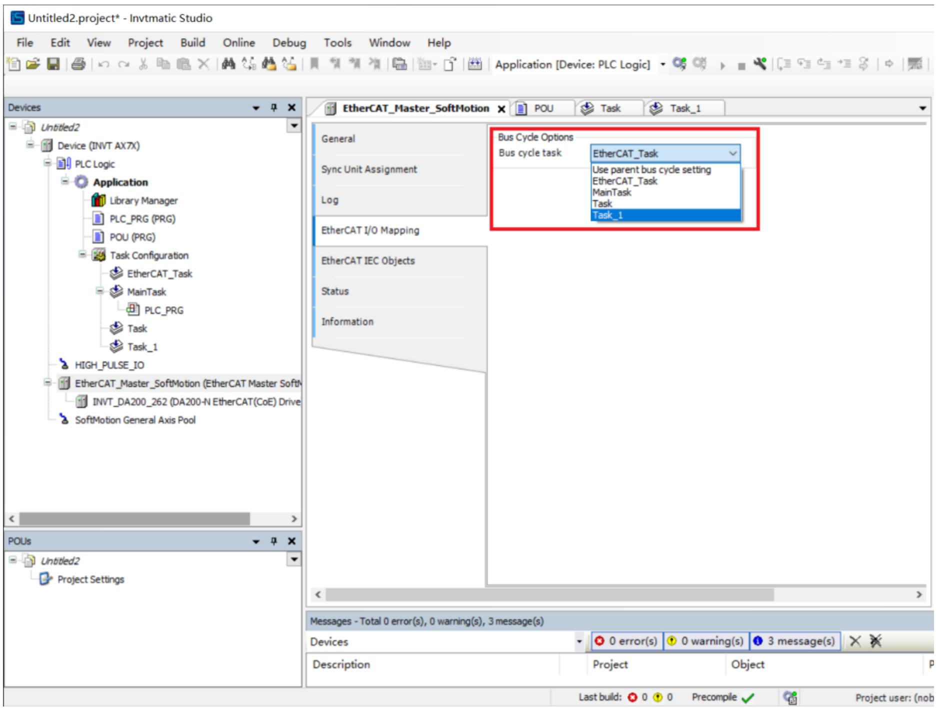


Figure 6-14 EtherCAT bus cycle task setting

6.6 Operation of multiple subprograms

In practical projects, the program can usually be divided into many subprograms according to the control flow or the object of the equipment. The designer can program each processing unit separately. As shown in Figure 6-15, the main program is divided into multiple subprograms with different processes through the control flow. The main purpose of the division is to make the main program clearer and facilitate future debugging.

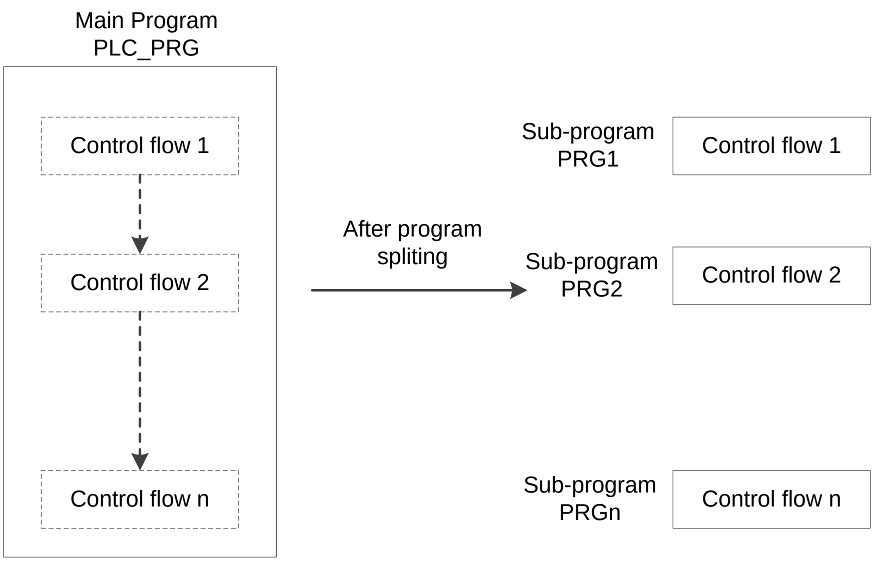


Figure 6-15 Split in multiple subprograms by process

The right part of Figure 6-15 displays the subprograms PRG1, PRG2…PRGn classified by the flow. The left part of the figure displays the main program PLC _PRG. The PRG1...PRGn subprograms can be called separately in the main program. There are two ways to run multiple subprograms. One is to add subprograms in the task configuration. The other is to call subprograms from the main program, which is more common and flexible.

1. Add subprograms in task configuration

Users can add subprograms in the task configuration page to realize the operation of multiple programs. Click **Add Call** to add subprograms in the order in which they are executed. As shown in Figure 6-16, after adding subprograms, the tasks will be executed in the top-to-bottom order specified by the user, or you can edit the order manually by using the **Move Up** and **Move Down** functions.

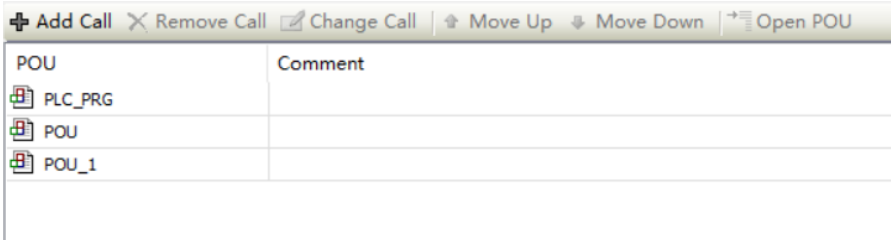


Figure 6-16 Add subprograms in a task

2. Call subprograms from main program PLC_PRG

PLC_PRG is the default main program of the system. In a sense, it can be understood as the battery of a car. In the production of a car, each part is assembled, which is equivalent to the writing of subprograms. When the car is assembled, it is necessary to check whether the car is usable. If you want to start the car, you must start the engine, lights and other parts through the battery which is equivalent to the entry point for starting the car. By calling the program in this way, the program becomes more operable and flexible. You can add judgment statements and use nesting in the program.

PLC_PRG is a special POU that runs by default with a coasting mode. This POU is called every control cycle by default without any additional task configuration. The configuration of the POU can be found in the task configuration. It can be used to call other subprograms and add necessary condition selection at the time of the call, or nest subprograms to make program calling more flexible. To implement the call relationship in Figure 6-17, write the following code in the main program PLC _PRG.

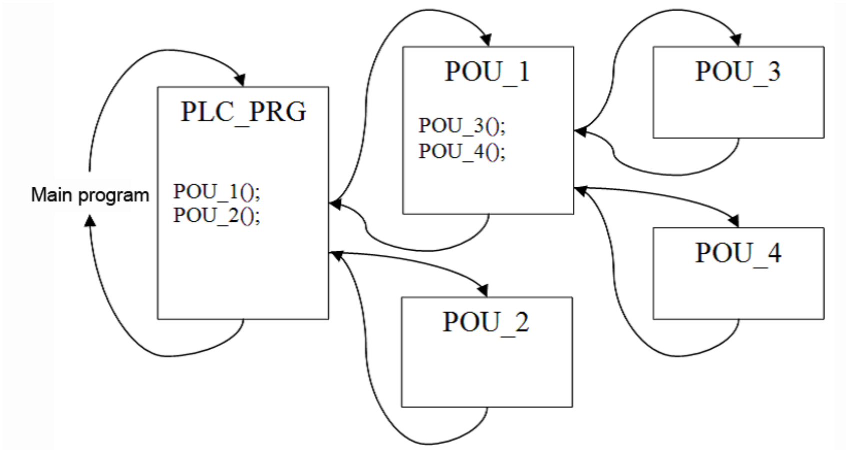


Figure 6-17 POU calling sequence

As shown in the Figure 6-17, the main program is PLC_PRG, which uses structured text programming language, and the program content is POU_1(); POU_2();.

The main function of the above programs is to call and execute POU_1 and POU_2 subprograms respectively. And POU_1 calls POU_3 and POU_4 respectively. The AX series programmable controller actually executes the programs in the following order:

A. AX series programmable controller program executes POU_1 first.

B. Since POU_3 and POU_4 are called sequentially in POU_1, POU_3 is executed first.

C. Execute POU_4 to complete POU_1.

D. Finally execute POU_2 to complete a full task cycle.

Repeating the above steps A to D is the internal execution sequence of the AX series programmable controller.

7 EtherCAT Bus Motion Control

7.1 EtherCAT operation principle

7.1.1 Protocol introduction

EtherCAT overcomes the inherent limitations of other Ethernet solutions. : On the one hand the Ethernet packet is no longer received then interpreted and process data then copied at every device, but the EtherCAT slave devices read the data addressed to them while the frame passes through the node. Similarly, input data is inserted while the telegram passes through. In the whole process, the frames are only delayed by a few nanoseconds.

The frame send by the master is passed through to the next device until it reaches the end of the segment (or branch). The last device detects an open port and therefore sends the frame back to the master. On the other hand, an EtherCAT frame comprises the data of many devices both in sending and receiving direction within one Ethernet frame. The usable data rate increases to over 90 %. The full-duplex features of 100 Mb/s TX are fully utilized, so that effective data rates of > 100 Mb/s (> 90 % of 2 x 100 Mb/s) can be achieved.

The EtherCAT master uses standard Ethernet Medium Access Controllers (MACs) without extra communication processors. Thus an EtherCAT master can be implemented on any equipment controller that provides an Ethernet interface, independently of the operating system or application environment. The EtherCAT slave uses an EtherCAT Slave Controller (ESC) for processing the data on-the-fly. Thus the performance of the network is not determined by the microcontroller performance of the slave but is handled complete in hardware. A process data interface (PDI) to the slave‘s application offers a Dual-Port-RAM (DPRAM) for data exchange.

Precise synchronization is particularly important in a wide range of distribution processes that require simultaneous actions, such as when several servo axes are performing simultaneous tasks. Precise calibration of distributed clocks is the most effective solution for synchronization. In the communication system, the stepwise calibration clock has the tolerance of error delay to a certain extent, compared with the fully synchronous communication.

7.1.2 Work counter WKC

The end of each EtherCAT message has a 16-bit working counter, WKC. WKC is a working counter used to record the number of reads and writes to the EtherCAT slave device. The EtherCAT slave controller calculates WKC in the hardware. The master receives the return data and checks the WKC in the sub-message. If WKC is not equal to the expected value, the sub-message has not been processed correctly. When a sub-message passes through a certain slave node, WKC will be increased by 1 if it is a single read or write operation. If it is a read and write operation, WKC will be increased by 1 upon read success, by 2 upon write success and by 3 upon complete. WKC is the accumulation of the processing results of each slave. The description of WKC increment is shown in Table 7-1.

Table 7-1 WKC increment

Command	Data type	Increment
Read	Read failed	–
	Read succeeded	+1
Write	Write failed	–
	Write succeeded	+1
Read/write	Failed	–
	Read succeeded	+1
	Write succeeded	+2
	Read and write succeeded	+3

7.1.3 Addressing mode

EtherCAT communication is realized by the master sending EtherCAT data frames to read and write the internal storage area of the slave device. EtherCAT messages use multiple addressing modes to operate the ESC internal storage area for multiple communication services. The addressing mode of EtherCAT is shown in Figure 7-1. An EtherCAT network segment is equivalent to an Ethernet device. The master first uses the MAC address of the Ethernet data frame header to address the network segment, and then uses the 32-bit address in the EtherCAT sub-message header to address the device in the segment. There are two ways to achieve in-segment addressing: device addressing and logical addressing. Device addressing performs read and write operations for a certain slave node. Logical addressing is oriented to process data and can be multicast. The same sub-message can read and write multiple slave devices.

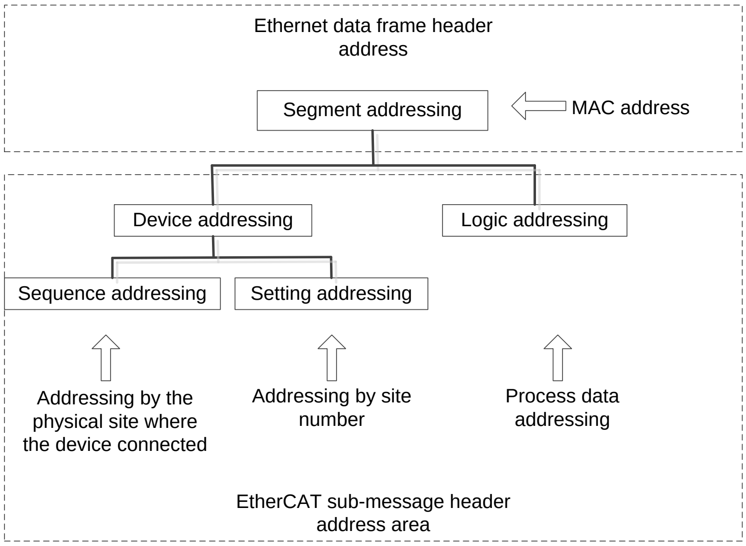


Figure 7-1 Addressing mode of EtherCAT

7.1.3.1 Segment addressing

Depending on how the EtherCAT master and its segment are connected, the segment can be addressed in two ways.

1. Direct connection mode

An EtherCAT segment is directly connected to the standard Ethernet port of the master device, as shown in Figure 7-2. In this case, the master uses the broadcast MAC address and the EtherCAT data frame is shown in Figure 7-3.

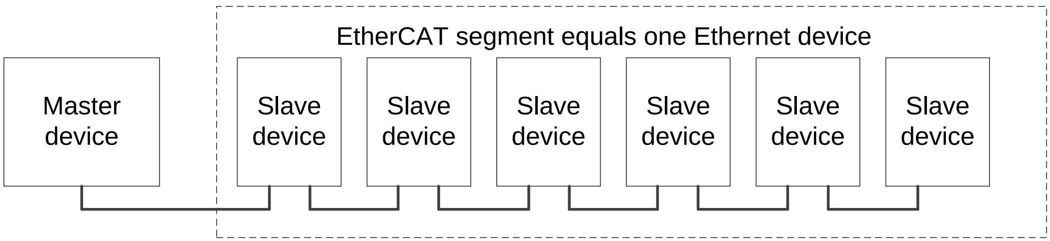


Figure 7-2 EtherCAT segment in direct connection mode

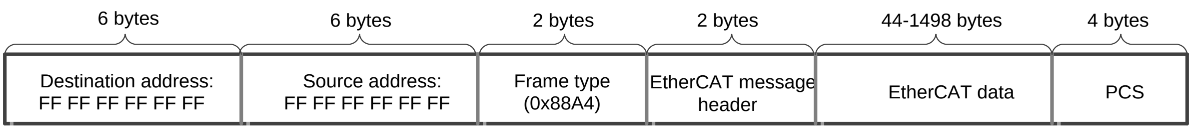


Figure 7-3 Addressing mode of EtherCAT in direct connection mode

2. Open mode

EtherCAT segment is connected to a standard Ethernet switch, as shown in Figure 7-4. In this case, a segment needs a MAC address and the address in the EtherCAT data frame sent by the master is the MAC address of the segment it controls, as shown in Figure 7-5. The first slave device in the EtherCAT segment has an ISO/IEC 8802.3 MAC address, which represents the entire segment. This slave is called a segment address slave, which can exchange the destination address area and source address area in the Ethernet. If EtherCAT data frame is sent over UDP, the device will also exchange the source and destination IP addresses and the source and destination UDP port numbers, making the response frame fully complied with the UDP/IP protocol.

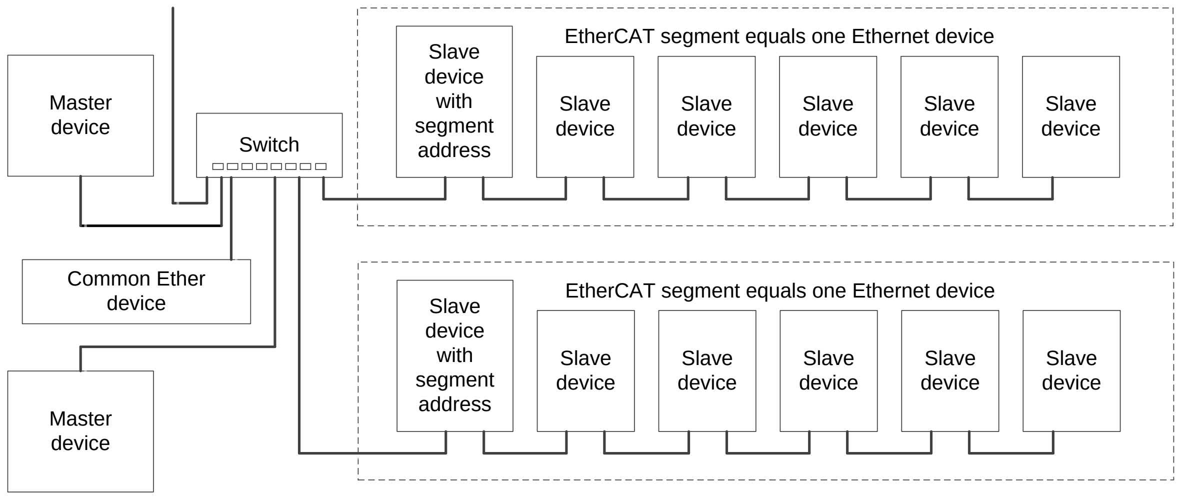


Figure 7-4 EtherCAT segment in open mode

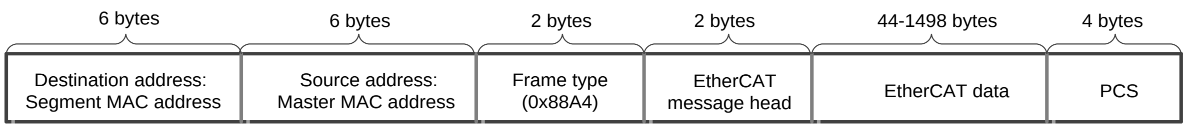


Figure 7-5 Addressing mode of EtherCAT in open mode

7.1.3.2 Device addressing

During device addressing, the 32-bit address in the EtherCAT sub-message header is divided into a 16-bit slave device address and a 16-bit slave device internal physical storage space address, as shown in Figure 7-6. The 16-bit slave device address can address 65535 slave devices, and each device can have up to 64 local address spaces.

Only one unique slave device is addressed per message in the device addressing mode, but there are two different mechanisms for addressing devices.

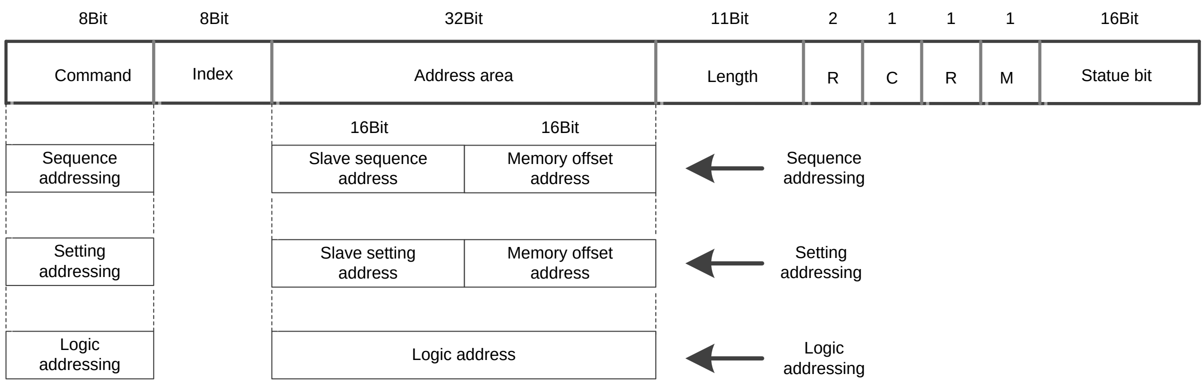


Figure 7-6 EtherCAT device addressing structure

3. Sequential addressing

For sequential addressing, the address of a slave is determined by its connection location within the segment, with a negative number indicating the location of each slave within the segment as determined by the wiring sequence. When the sequential addressing sub-message passes through each slave device, its sequential address is increased by 1. When the slave receives a message, the message with a sequential address of 0 is the message addressed to it. This mechanism is also known as "automatic incremental addressing" because it updates the device address as the message passes through.

In Figure 7-7, there are three slave devices in the segment that are sequentially addressed as 0, -1, -2, and so on. When the master uses sequential addressing to access the slave, the address change of the sub-message is shown in Figure 7.8. The master station sends 3 sub-messages to address 3 slave nodes, where the addresses are 0, -1 and -2 respectively, and the data frame is 1 as shown in the figure. When the data frame reaches the slave ①, the slave ① checks that the address in sub-message 1 is 0, thus knowing that sub-message 1 is the message addressed to itself. After the data frame passes through the slave ①, all sequential addresses are increased by 1, called 1, 0 and -1, as shown in the data frame 2 in Figure 7-8. When the data frame reaches the slave ②, the slave ② finds that the address in sub-message 2 is 0, which is its own message. Similarly, subsequent slave nodes are addressed in this way. As shown in Figure 7.7, in actual engineering applications, sequential addressing is mainly used in the startup phase, and the master node configures a site address for each slave node. After that, the slave node can be addressed using a site address that is independent of their physical location. The sequential addressing mechanism can be used to automatically address the slave node, as shown in Figure 7-8.

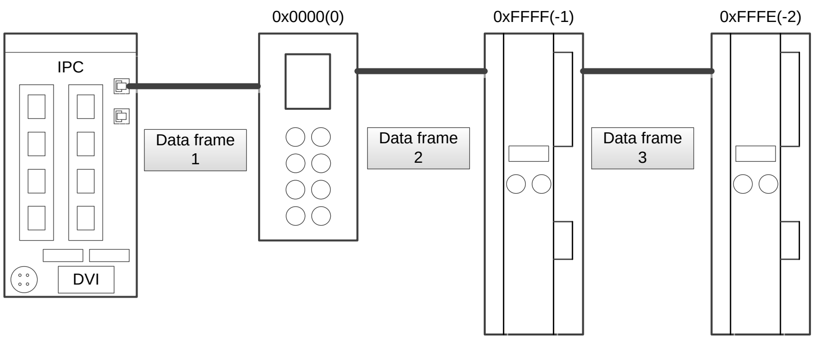


Figure 7-7 Sequentially addressed slave address

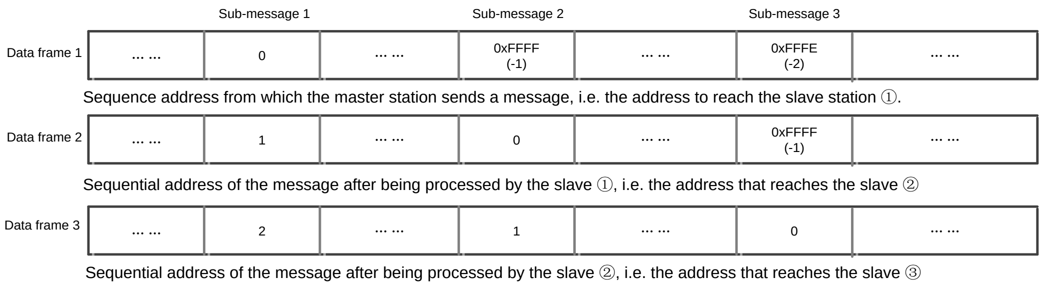


Figure 7-8 Change of sub-message address during sequential addressing

4. Setting addressing

When setting addressing, the slave node address is independent of its sequential order within the network segment. As shown in Figure 7-9, the address can be configured by the master to the slave in the data link start-up phase, or loaded by the configuration data of the slave in the power-on initialization phase, and then read by the master in the link start-up phase using the sequential addressing mode to set the address of each slave node. Its message structure is shown in Figure 7-10.

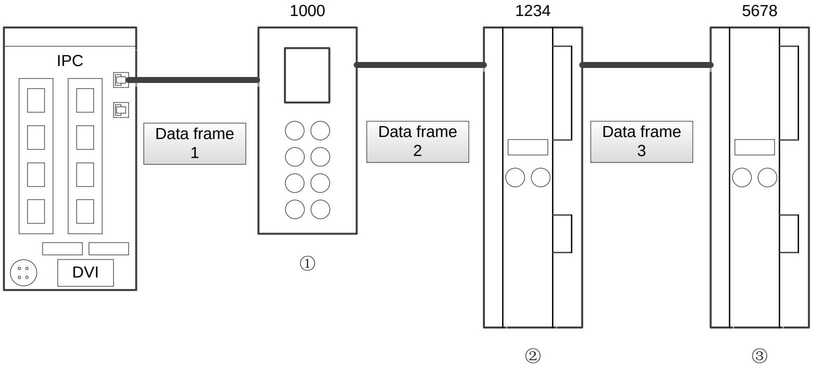


Figure 7-9 Slave address in setting addressing mode

		Sub-message 1		Sub-message 2		Sub-message 3	
Data frame 1	··· ···	1000	··· ···	1234	··· ···	5678	··· ···

Figure 7-10 Message structure in setting addressing mode

5. Logic addressing

For logical addressing, the slave address is not defined separately, but using a section of the 4GB logical address space in the addressing section. The 32-bit address area within the message is used as the overall data logical address to complete the logical addressing of the device. The logical addressing mode is implemented by the Fieldbus Memory Management Unit (FMMU). The FMMU function is located inside each ESC and maps the local physical storage address of the slave to the logical address of the segment. The schematic diagram is shown in Figure 7-11.

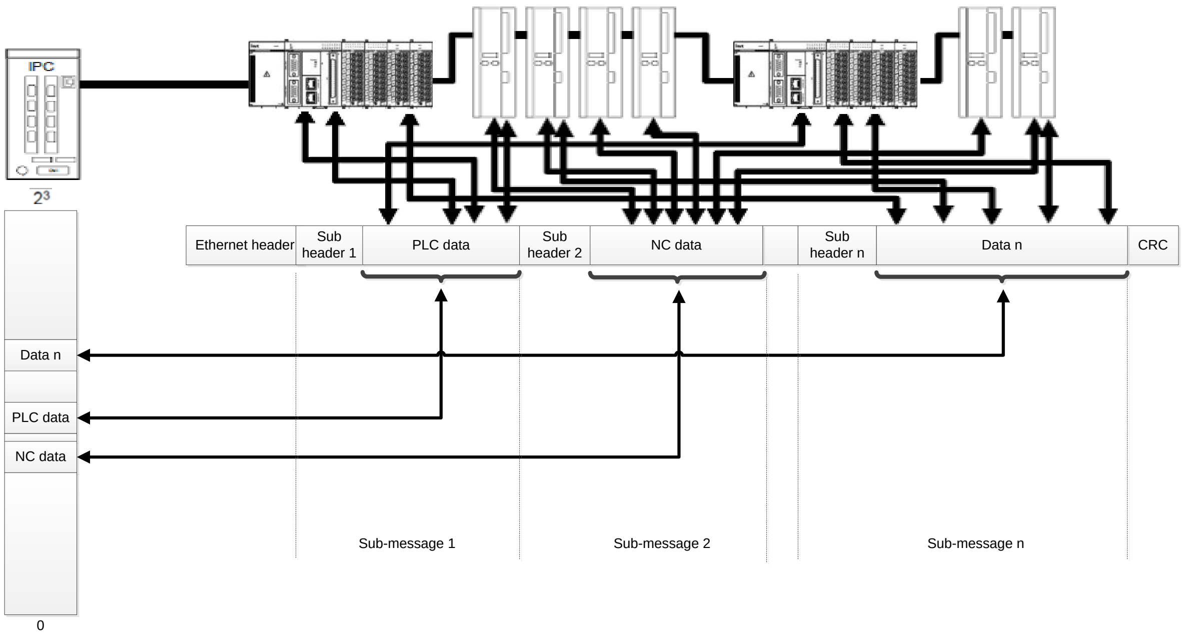


Figure 7-11 FMMU operating Principle

When receiving an EtherCAT sub-message of data logic addressing, the slave device will check for an FMMU unit address match. If the match exists, the slave device will insert the input type data into the corresponding position in the EtherCAT sub-message data area, and extracts the output type data from the corresponding position in the EtherCAT sub-message data area.

7.1.4 Distributed clocks

7.1.4.1 Concepts

In applications with spatially distributed processes requiring simultaneous actions, exact synchronization is particularly important. For example, this is the case for applications in which multiple servo axes execute coordinated movements. With this mechanism, the slave device clocks can be precisely adjusted to this reference clock. The first slave connected to the master with distributed clocking functions acts as a reference clock to synchronize the slave clocks of the other devices and the master. To achieve precise clock synchronization control, it is necessary to measure and calculate the data transmission delay and local clock offset, and to compensate for the drift of the local clock. The following 6 concepts are involved in the synchronization of the clock.

1. System time

The system time is the system timing used by the distributed clock. It starts at 0:00 on January 1, 2001, and is expressed in a 64-bit binary variable in nanoseconds (ns) and can be timed for up to 500 years. It can also be expressed as a 32-bit binary variable with a maximum of 4.2s, which is usually used for communication and time stamping.

2. Reference clock and slave clock

The EtherCAT protocol defines the first slave connected to the master with distributed clocking functions acts as a reference clock, and the clocks of other slave nodes are called slave clocks. The reference clock is used to synchronize the slave clocks and the master clock of other slave devices. The reference clock provides the EtherCAT system time.

3. Master clock

The EtherCAT master station also has a timing function, which is called the master clock. The master clock can be synchronized as a slave clock in a distributed clock system. In the initialization phase, the master can send the master clock to the reference clock slaves in system time format, which enables the distribution clocks to be timed using system time.

4. Local clock, initial offset and clock drift

Each DC slave has a local clock, which runs independently and is timed using the local clock signal. When the system starts, there is a certain difference between the local clock and the reference clock of each slave, which is called the initial clock offset. During operation, due to the fact that the reference clock and the DC slave clock use their own clock sources, their timing cycles drift to a certain extent, which will lead to the clock running out of sync and the local clock drifting. Therefore, the initial clock offset and clock drift must be compensated.

5. Local system time

The local clock of each DC slave generates a local system time after compensation and synchronization. The distributed clock synchronization mechanism is to keep the local system time of each slave consistent. The reference clock is also the local system clock of the corresponding slave.

6. Transmission delay

There will be a certain delay when data frames are transmitted between slaves, which includes device internal and physical connection delays. Therefore, when synchronizing slave clocks, the transmission delay between the reference clock and multiple slave clocks should be considered.

7.1.4.2 Clock synchronization process

Clock synchronization consists of the following three steps:

- Transmission delay measurement

When the distributed clock is initialized, the master will initialize the transmission delay for slave nodes in all directions, calculate the deviation value between the slave clocks and the reference clock, and write it into the slave clock.

- Reference clock offset compensation (system time)

The local clock of each slave will be compared with the system time, and then different comparison results will be written into different slaves, so that all slaves will get the absolute system time.

- Reference clock drift compensation

Clock drift compensation and local time are used to periodically compensate for local clock errors and fine-tuning. The following figure illustrates two application cases of compensation calculation. Figure 7-12 shows a case where the system time is less than the slave local clock. Figure 7-13 shows a case where the system time is greater than the slave local clock.

1. System time < local time

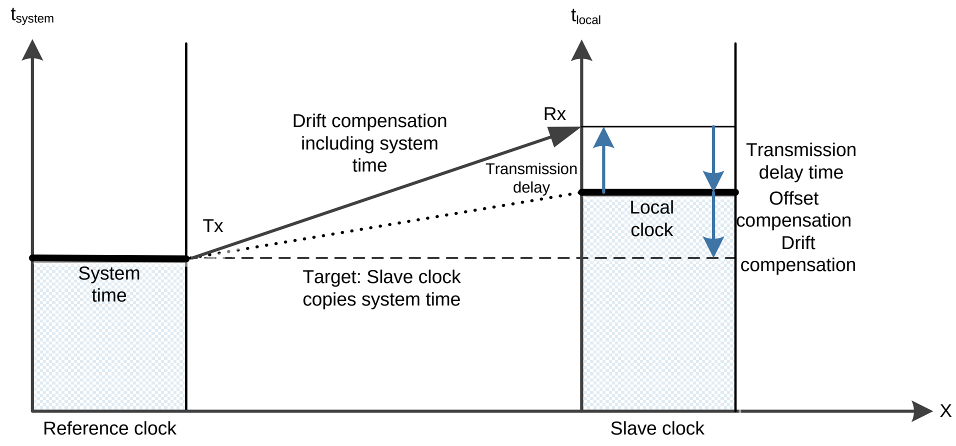


Figure 7-12 Clock synchronization: system time < local time

2. System time > local time

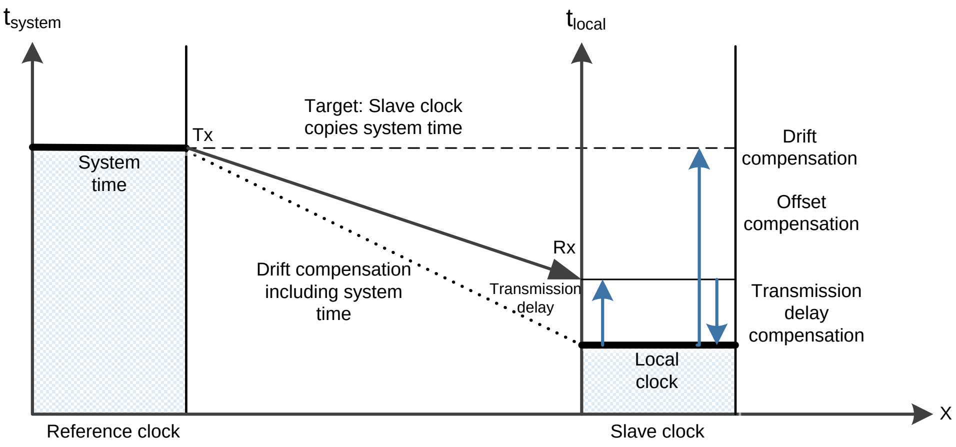


Figure 7-13 Clock synchronization: system time > local time

With EtherCAT, data exchange is completely hardware-based. Due to the logic ring structure of communication (with the help of the physical layer of full-duplex fast Ethernet), the master clock can simply and accurately determine the delay offset of slave clock propagation, and vice versa. The distributed clocks are adjusted based on this value, which indicates that a very precise deterministic synchronization error time base (less than 1 microsecond) can be used across the network. Its structure is shown in Figure 7-14.

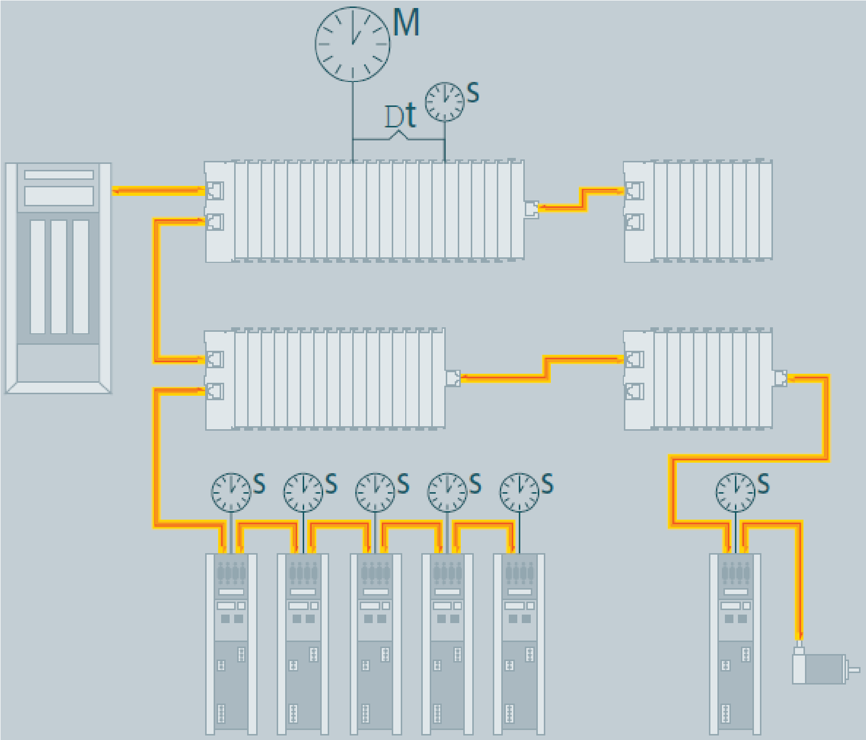


Figure 7-14 Clock synchronization principle

For example, there is a difference of 300 nodes between the two devices, and the cable length is 120 meters. Use an oscilloscope to capture the communication signal, and the result is shown in Figure 7-15.

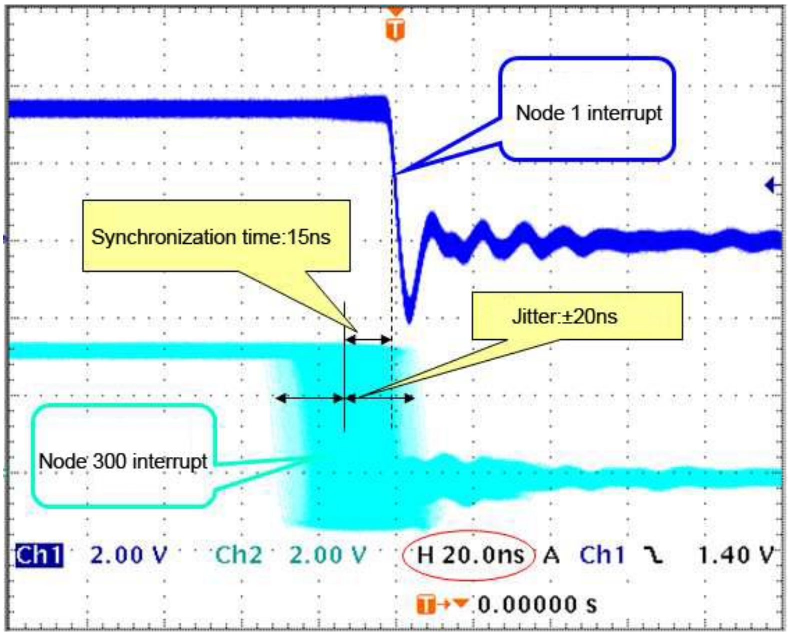


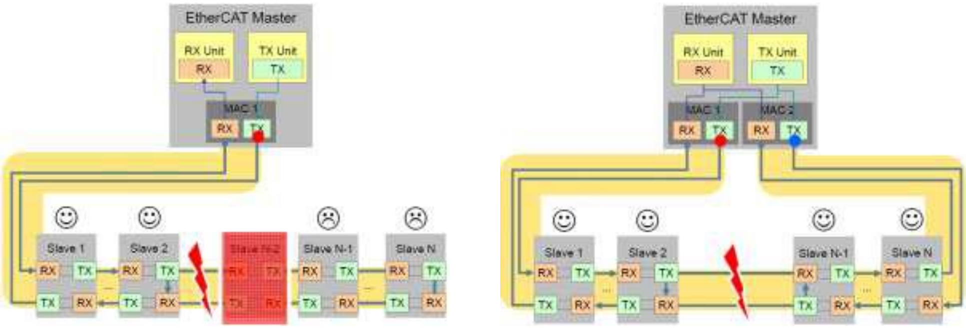
Figure 7-15 Performance test of clock synchronization

This function is very important for motion control. In such applications, velocity is typically derived from the measured position. Even very small jitter in the position measurement timing can translate to larger inaccuracies in the calculated velocity, especially relative to short cycle times. In EtherCAT, the introduction of time-stamped data types as a logical extension allows high-resolution system times to be added to the measured value, which is made possible by the huge bandwidth that Ethernet provides.

7.1.5 EtherCAT cable redundancy

Increasing demands in terms of system availability are catered for with optional cable redundancy that enables devices to be exchanged without having to shut down the network. Adding redundancy is very inexpensive: the only additional hardware is another standard Ethernet port (no special card or interface) in the master device and the single cable that turns the line topology into the ring. Switchover in case of device or cable failure only takes one cycle, so even demanding motion control applications survive a cable failure without problems.

EtherCAT also supports redundant masters with hot standby functionality. Since the EtherCAT Slave Controllers immediately return the frame automatically if an interruption is encountered, failure of a device does not necessarily lead to the complete network being shut down. For example, the standard EtherCAT topology is shown in Figure 7-16 a). If there is a network interruption between Slave2 and SlaveN-2 in this topology (the red part in the figure), all slave communication after Slave N-2 is interrupted accordingly. This is also the disadvantage of the standard topology.



a) Standard EtherCAT topology b) EtherCAT redundant topology

Figure 7-16 EtherCAT redundancy

Figure 7-16 b) shows the topology structure of the EtherCAT redundancy mode. Only two standard network ports are needed for the master to realize the topology. With these two ports, all slave nodes can form a loop. Even if the network is interrupted while in use, such as the disconnected red part in Figure 7-16, the master node will detect the error immediately and automatically divide the communication into two channels, and all the slave nodes can continue to communicate to ensure the stable operation of the system.

7.2 EtherCAT communication mode

In actual automation control systems, there are usually two forms of data exchange between applications: time-critical and time-non-critical. Time critical indicates that a specific action must be completed within a certain time window. If the communication cannot be completed within the required time window, it may cause control failure. Time-critical data is usually sent periodically, which is called periodic process data communication. Non-time-critical data can be sent out of cycle, and non-periodical mailbox data communication is used in EtherCAT.

7.2.1 Periodic process data communication

The master node can use logical read, write or read and write commands to control multiple slaves at the same time. In the periodic data communication mode, the master and the slave have multiple synchronous operation modes.

1) Slave device synchronization mode

✧ Free running

In free-run mode, the local control cycle is generated by a local timer interrupt. The cycle time can be set by the master, which is an optional feature of the slave. The local cycle in free-running mode is shown in Figure 7-17. In the figure, T1 is the time for the local microcontroller to copy data from the EtherCAT slave controller and calculate the output data; T2 is the output hardware delay, and T3 is the input latch offset time. These parameters reflect the time response performance of the slave.

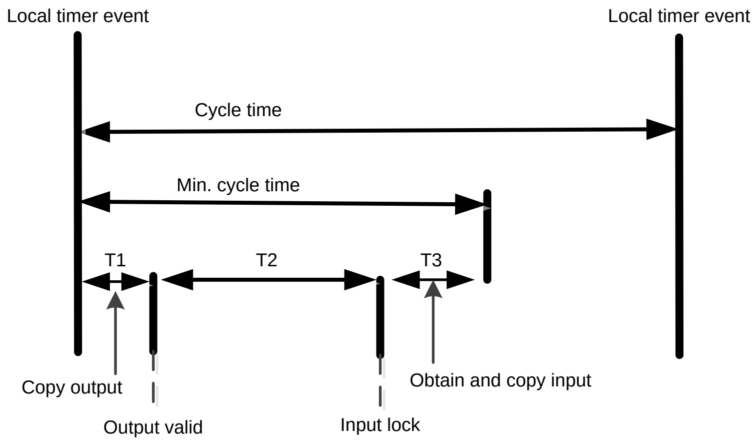


Figure 7-17 Local cycle in free-running mode

✧ Synchronization to data or output events

The local cycle is triggered on the occurrence of a data input or output event, as shown in Figure 7-18. The master can write the sending cycle of the process data frame into the slave. The slave will check if this cycle time is supported or if the cycle time is optimized locally. The slave can choose to support this feature. It is usually synchronized to the data output event. If the slave only has input data, the data is synchronized to the input event.

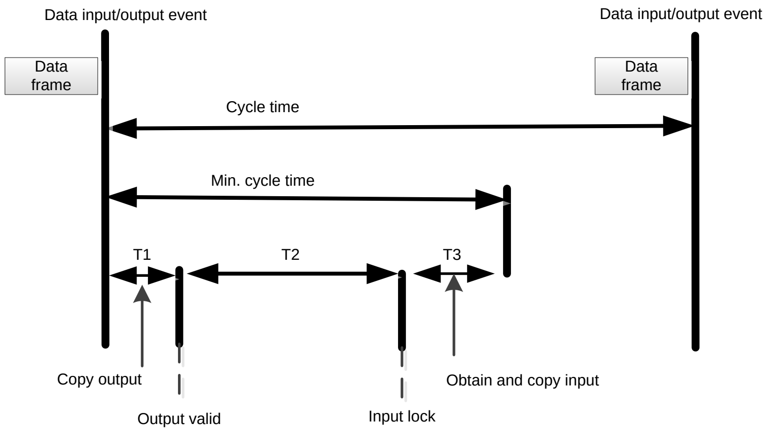


Figure 7-18 Local cycle of synchronization to data input or output events

✧ Synchronization to distributed clock synchronization event

The local cycle is triggered by the SYNC event, as shown in Figure 7-19. The master must complete the transmission of the data frame before the SYNC event. For this reason, the master clock must also be synchronized with the reference clock.

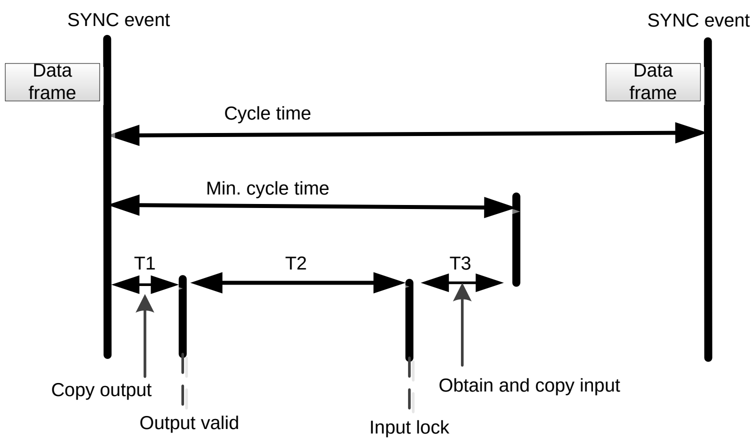


Figure 7-19 Local cycle of synchronization to SYNC event

To further optimize slave station synchronization performance, the master should copy the output information from the received process data frame when a data transmission and reception event occurs. After the SYNC signal arrives, continue the local operation. As shown in Figure 7-20, the data frame must arrive T1 time earlier than the SYNC signal. The slave has completed data exchange and control calculations before the SYNC event and can perform the output operation immediately after receiving the SYNC signal, further improving synchronization performance.

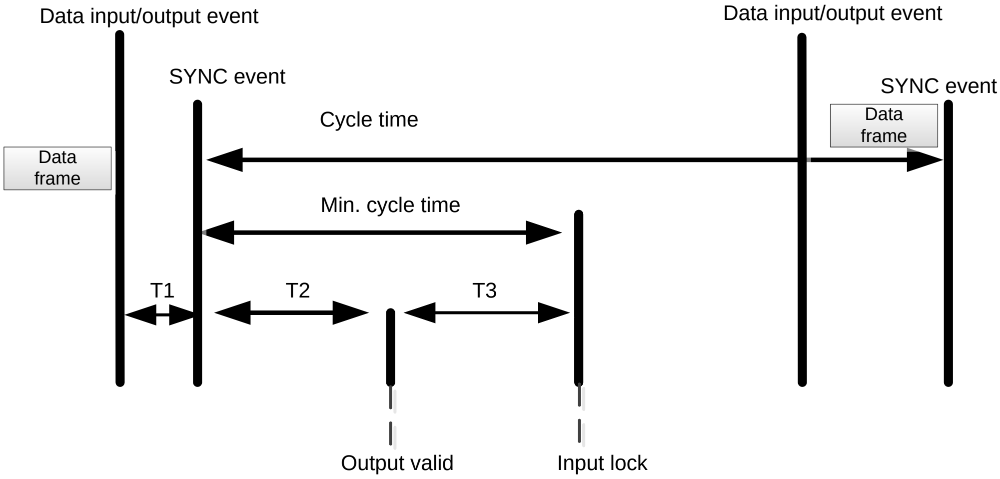


Figure 7-20 Local cycle of the optimized synchronization to SYNC event

2) Master device synchronization mode

There are two synchronization modes for the master.

✧ Cyclic mode

In cyclic mode, the master periodically sends process data frames. The master's cycle is usually controlled by a local timer. The slave node can run in free-running mode or in synchronization to received data event mode. For the slave in synchronization mode, the master should check that the cycle time of the corresponding process data frame is greater than the minimum cycle time supported by the slave.

The master can send a variety of periodic process data frames at different cycle times to get the most optimized bandwidth. For example, a shorter cycle is used to send motion control data and a longer cycle is used to send I/O data.

✧ DC mode

The master runs in DC mode similarly to cyclic mode, except that the local cycle of the master should be synchronized with the reference clock. The master's local timer should be adjusted based on the ARMW message that publishes the reference clock. After the ARMW message used to dynamically compensate clock drift is returned to the master, the master clock can be adjusted based on the read back reference clock time to be roughly synchronized with the reference clock time.

In DC mode, all DC-enabled slaves should be synchronized to the DC system time. The master should also synchronize the other communication cycles with the DC reference clock time. Figure 7-21 shows how the local cycle is synchronized with the DC reference clock.

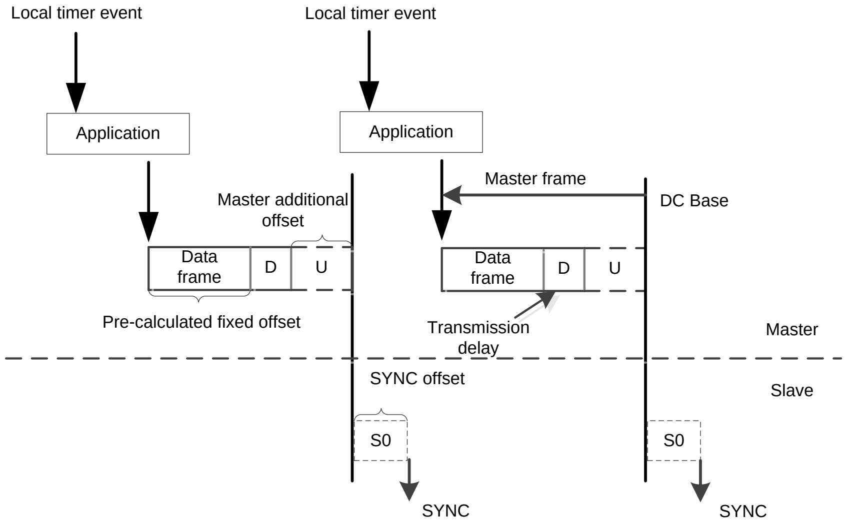


Figure 7-21 Master DC mode

The master local run is started by a local timer. The local timer should have an advance over the DC reference clock timing, which is the sum of the following times.

1. Control program execution time
2. Data frame transmission time
3. Data frame transmission delay (D)
4. Additional offset (U) (Related to the jitter value of the delay time of each slave and the jitter value of the control program execution time, used for the adjustment of the master cycle)

7.2.2 Non-periodic mailbox data communication

The non-periodical data communication in the EtherCAT protocol is called mailbox data communication, which can be carried out in both directions, i.e. from the master to the slave and from the slave to the master. It supports full duplex, two-way independent communication and multi-user protocols. The slave-to-slave communication is managed by the master as a router. The mailbox communication data header includes an address field that enables the master to resend mailbox data. Mailbox data communication is a standard way of realizing parameter exchange, and is used if periodic process data communication or other non-periodic services need to be configured.

The mailbox data message structure is shown in Figure 7-22. Usually the mailbox communication value corresponds to a slave station, so the device addressing mode is used in the message. The data elements in its data header are listed in Table 7-2.

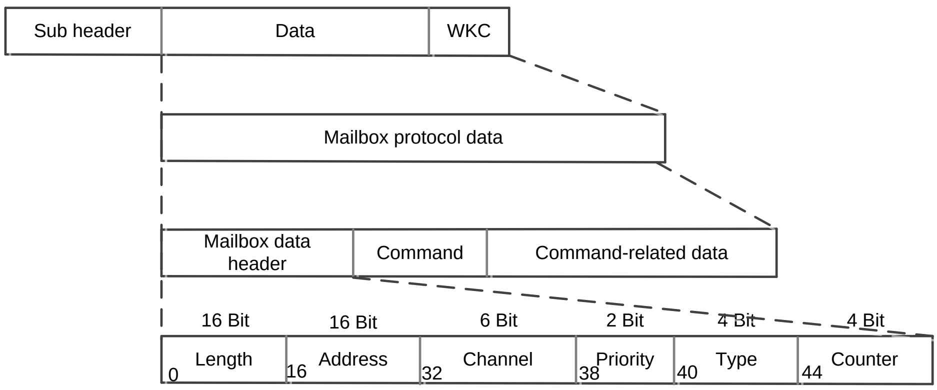


Figure 7-22 Mailbox data unit structure

Table 7-2 Mailbox data header

Data element	Bit	Description
Length	16 bits	Length of the followed mailbox service data
Address	16 bits	Slave address of data source for master-to-slave communication Slave address of data destination for master-to-slave communication
Channel	6 bits	Reserved
Priority	2 bits	Reserved
Type	4 bits	Mailbox type, i.e. type of subsequent protocol. 0: Mailbox communication error 2: EoE (Ethernet over EtherCAT) 3: CoE (CANopen over EtherCAT) 4: FoE (File Access over EtherCAT) 5: SoE (Sercos over EtherCAT) 15: VoE (Vendor Specific Profile over EtherCAT)
Counter (Ctr)	4 bits	Sequence number used for repeated detection, increasing by 1 for each new mailbox service (Only 1 to 7 is used for compatibility with older versions)

✧ Master-to-slave communication – write mailbox command

The master sends the write data area command to send mailbox data to the slave. The master will check the work counter WKC in the slave's answer message of mailbox command. If the work counter is 1, the write command is successful. Conversely, if the work counter is not increased, which is usually because the slave did not finish reading the previous command, or did not respond within a limited time, the master must resend the write mailbox data command.

✧ Master-to-slave communication – read mailbox command

To be sent from the slave to the master, the data must first be written to the input mailbox cache and then read by the master. If there is valid data waiting to be sent from the slave ESC input mailbox data area, the master will send the appropriate read command to read the slave data as soon as possible. There are two ways for the master to determine whether the slave has filled the mailbox data into the input data area. One is to use FMMU to periodically read a certain flag bit. Logical addressing can be used to read the flags of multiple slave s, but the disadvantage is that each slave requires an FMMU unit. The other way is to input a simple rotation training ESC into the input area of the mailbox. An increase of 1 in the work counter of the read command indicates that the slave has populated the input data area with new data.

7.3 EtherCAT state machine

EtherCAT State Machine (ESM) coordinates the state of the master and slave applications at initialization and runtime.

The EtherCAT device must support four states and an optional state.

✧ Init: initialization, abbreviated as I.

✧ Pre-Operational: abbreviated as P.

✧ Safe-Operational: abbreviated as S.

✧ Operational: abbreviated as O.

✧ Boot-Strap: (Optional) abbreviated as B.

The conversion relationship between the above states is shown in Figure 7-23. When the state is converted from the initialization state to the operational state, the conversion must be done in the order of "Init > Pre-Operational > Safe-Operational > Operational > Boot-Strap". The leapfrog conversion is only available when returning from the Operational state. The Boot-Strap state is optional and is only allowed to convert to and from the Init state. All state changes are initiated by the master node, which sends a state control command to the slave to request a new state, and the slave responds to this command by performing the requested state conversion and writing the result to the slave state indicator variable. If the requested state conversion fails, the slave will give an error flag. Table 7-3 shows the summary of state conversions.

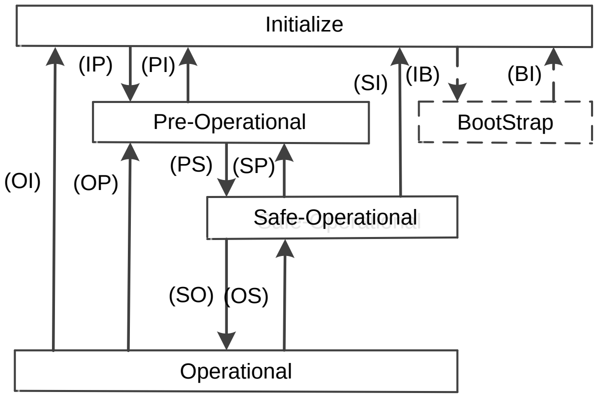


Figure 7-23 EtherCAT state conversion

✧ Init

The initialization state defines the initial communication relationship between the master and the slave at the application layer. At this time, the master and the slave cannot communicate directly at the application layer, and the master uses the initialization state to initialize some configuration registers of the ESC. If the master supports mailbox communication, configure the mailbox communication parameters.

✧ Pre-Operational

In Pre-Operational state, mailbox communication is activated. The master and slave can use mailbox communication to exchange application-related initialization operations and parameters. Process data communication is not allowed in this state.

✧ Safe-Operational

In Safe-Operational state, the slave application reads the input data, but does not generate an output signal. The device has no output and is in a "safe state". In this case, mailbox communication is still available.

✧ Operational

In Operational state, the slave application reads data, the master application sends out output data, and the slave device generates an output signal. In this case, mailbox communication is still available.

✧ Boot-Strap

The function of the boot strap state is to download the device firmware program. The master can download a new firmware program to the slave using FoE protocol mailbox communication.

Table 7-3 State conversion of EtherCAT state machine

State and state conversion	Description
Init	There is no communication at the application layer, and the master can only read and write ESC registers.
Init to Pre-OP (IP)	The master configures the slave site address register. Configure mailbox channel parameters if mailbox communication is supported. Configure DC related registers if distributed clocks are supported. The master writes state control register to request "Pre-Op" state.
Pre-Operational	Mailbox data communication at application layer

State and state conversion	Description
Pre-Op to Safe-Op (PS)	The master uses mailboxes to initialize process data mapping. The master configures the SM channel used for data communication. The master configures FMMU. The master writes state control register to request "Safe-Op" state.
Safe-Operational	The master sends valid output data. The master writes state control register to request "Op" state.
Operational	All inputs and outputs are valid. Mailbox communication is still available.

7.4 EtherCAT servo drive controller application protocol

IEC 61800 standard series is a general specification for variable speed electronic power drive systems. IEC 61800-7 defines the standard of communication interface between control system and power drive system, including network communication technology and application profile, as shown in Figure 7-24. EtherCAT, as a network communication technology, supports the profile CiA 402 in the CANopen protocol and the application layer of the SERCOS protocol, which are called CoE and SoE respectively.

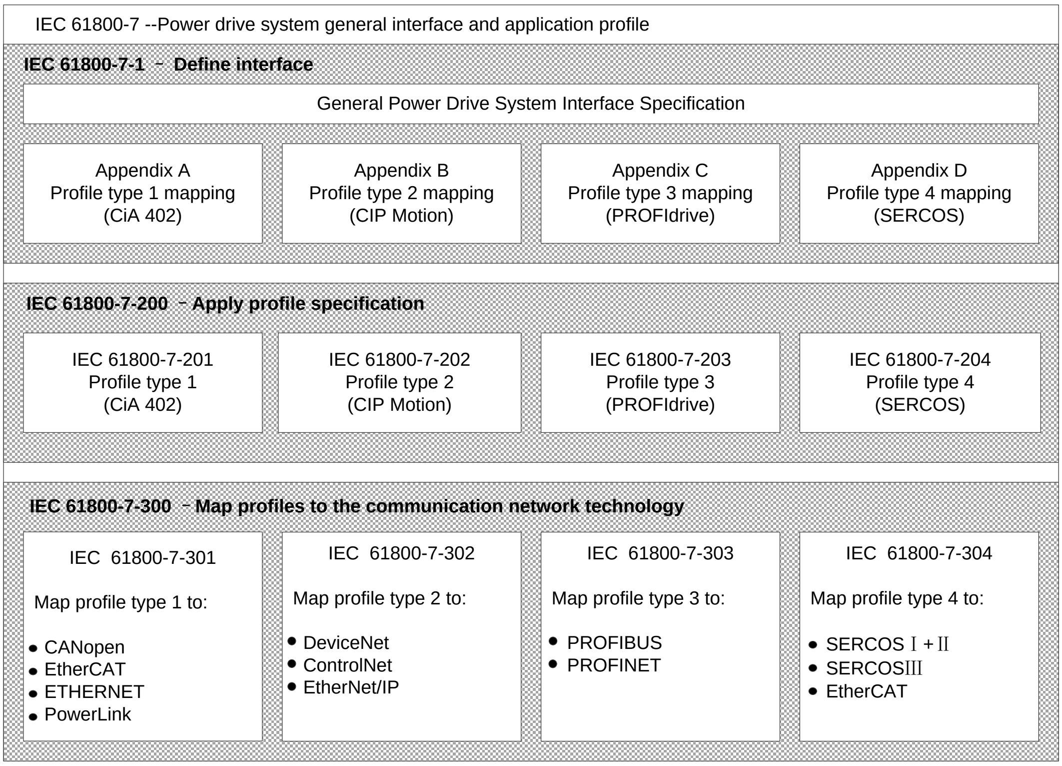


Figure 7-24 IEC 61800-7 architecture

7.4.1 EtherCAT-based CAN application protocol (CoE)

CANopen device and application profiles are available for a wide range of device classes and applications, ranging from I/O components, drives, encoders, proportional valves and hydraulic controllers to application profiles for plastic or textile machinery, for example. EtherCAT can provide the same communication mechanisms as the familiar CANopen mechanisms: object dictionary, PDO (process data objects) and SDO (service data objects) – even the network management is comparable. EtherCAT can thus be implemented with minimum effort on devices equipped with CANopen. Large parts of the CANopen firmware can be reused. Objects can optionally be expanded in order to account for the larger bandwidth offered by EtherCAT.

The EtherCAT protocol supports the CANopen protocol at the application level and is supplemented by the following main features:

- Network initialization by accessing the CANopen object dictionary and objects using mailbox communication
- Network management by using CANopen application objects and optional time-driven PDO messages.
- Mapping process data, cyclic transmission command data and state data by object dictionary.

Figure 7-25 shows the CoE device structure whose communication modes mainly include periodic process data communication and non-periodic data communication. The following section will introduce the differences between both modes in practical applications.

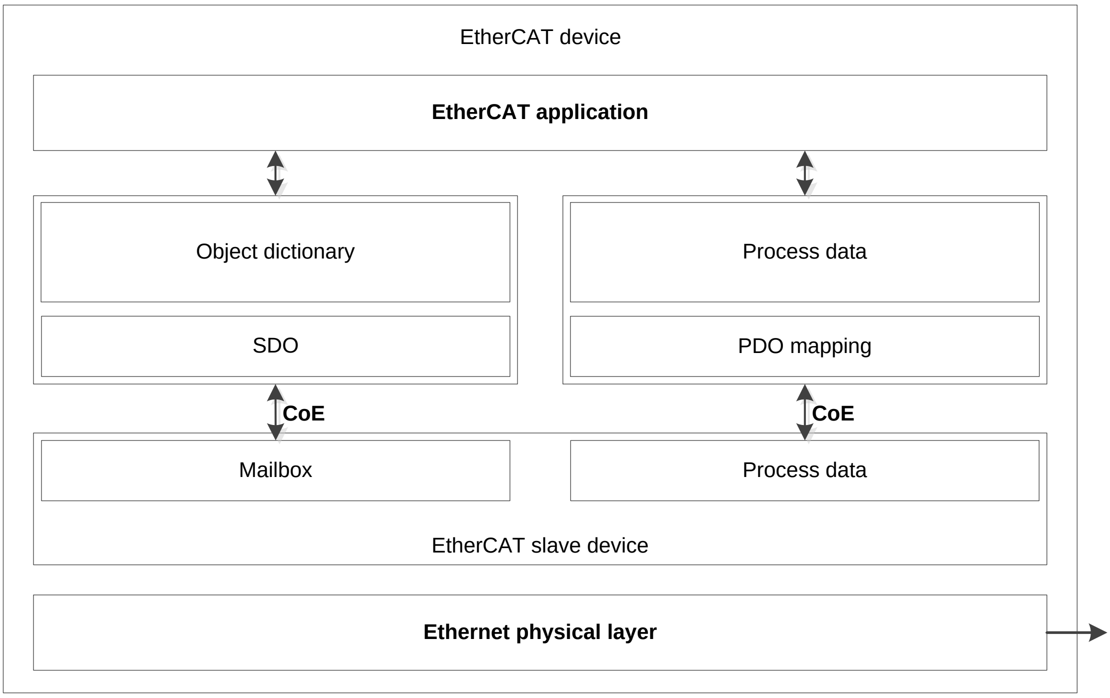


Figure 7-25 CoE device structure

7.4.1.1 CoE object dictionary

The CoE protocol fully complies with the CANopen protocol and has the same object dictionary definition as shown in Table 7-4.

Table 7-5 lists the CoE communication data objects, which extend the relevant communication objects 0x1C00–0x1C4F for EtherCAT communication to set the type of storage synchronization manager, communication parameters and PDO data allocation.

Table 7-4 CoE object dictionary definition

Index number range	Description
0x0000–0x0FFF	Data type description
0x1000–0x1FFF	Communication objects include: device type, identifier, PDO mapping, CANopen-compatible data object for CANopen. EtherCAT extension data object is reserved in EtherCAT.
0x2000–0x5FFF	Manufacturer definition object
0x6000–0x9FFF	Profile definition data object
0xA000–0xFFFF	Reserved

Table 7-5 CoE communication data object

Index	Description
0x1000	Device type
0x1001	Error register
0x1008	Vendor device name
0x1009	Manufacturer hardware version
0x100A	Manufacturer software version
0x1018	Device identifier
0x1600–0x17FF	RxPDO mapping
0x1A00–0x1BFF	TxPDO mapping
0x1C00	Sync manager communication type
0x0x1C10–0x1C2F	Process data communication sync manager PDO assignment
0x0x1C30–0x1C4F	Synchronization management parameters

7.4.1.2 CoE periodic process data communication (PDO)

In periodic data communication, the process data can contain multiple PDO mapping data objects. The data objects 0x1C10 to 0x1C2F used by the CoE protocol define the corresponding PDO mapping channels. Table 7-6 shows the specific structure of the communication data in the EtherCAT protocol.

Table 7-6 CoE communication data object

Index	Object type	Description	Type
0x1C10	Array	SM0 PDO assignment	Unsigned integer 16-bit
0x1C11	Array	SM1 PDO assignment	Unsigned integer 16-bit
0x1C12	Array	SM2 PDO assignment	Unsigned integer 16-bit
0x1C13	Array	SM3 PDO assignment	Unsigned integer 16-bit
…	…	…	…… …… …… ……
0x1C2F	Array	SM31 PDO assignment	Unsigned integer 16-bit

The following uses the allocation for SM2 PDO (0x1C12) as an example and Table 7-7 lists its value. If two data are mapped in PDO0, the first communication variable will be the control word with the corresponding mapped index and sub-index address 0x6040:00, and the second communication variable is the target position value with the corresponding mapped index and sub-index address 0x607A:00.

Table 7-7 Example of SM2 channel PDO assign object data 0x1C12

<table>
<tr><th rowspan="2">0X1C12
Sub-index</th><th rowspan="2">Numeric value</th><th colspan="4">PDO data object mapping</th></tr>
<tr><th>Sub-index</th><th>Numeric value</th><th>Bytes</th><th>Description</th></tr>
<tr><td>0</td><td>3</td><td></td><td></td><td>1</td><td>Number of PDO mapping objects</td></tr>
<tr><td rowspan="3">1</td><td rowspan="3">PDO0
0x1600</td><td>0</td><td>2</td><td>1</td><td>Number of data mapping data objects</td></tr>
<tr><td>1</td><td>0x6040: 00</td><td>2</td><td>Control word</td></tr>
<tr><td>2</td><td>0x607A: 00</td><td>4</td><td>Target position</td></tr>
</table>

0X1C12 Sub-index	Numeric value	PDO data object mapping			
		Sub-index	Numeric value	Bytes	Description
1	PDO1 0x1601	0	2	1	Number of data mapping data objects
		1	0x6071: 00	2	Target torque
		2	0x6087: 00	4	Target ramp
1	PDO2 0x1602	0	2	1	Number of data mapping data objects
		1	0x6073: 00	2	Max. current
		2	0x6075: 00	4	Motor rated current

There are several PDO mapping modes:

1. Simple devices do not require mapping protocols

- Use simple process data
- Read in the EEPROM of the slave

2. Readable PDO mapping

- Fix process data mapping
- Read with SDO communication

3. Selectable PDO mapping

- Multiple fixed PDO groups are selected by object 0x1C1X
- Read through SDO communication

4. Variable PDO mapping

- Configure through CoE communication

7.4.1.3 CoE non-periodic process data communication (SDO)

The EtherCAT master enables non-periodic data communication via reading and writing mailbox data SM channels. The CoE protocol mailbox data structure is shown in Figure 7-26.

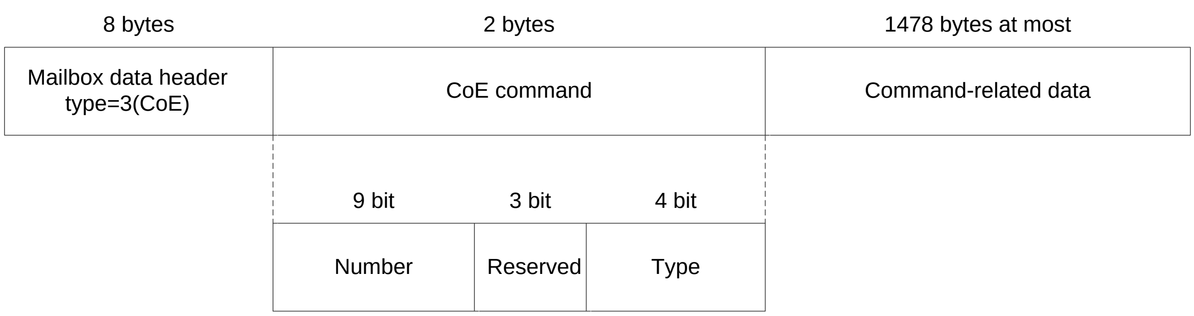


Figure 7-26 CoE data header

The numbered part in Figure 7-26 is explained in detail in Table 7-8.

Table 7-8 CoE command definition

CoE command field	Description
No.	Number when PDO is sent
Type	Message type: 0: Reserved 1: Emergency information 2: SDO request 3: SDO response 4: TxPDO 5: RxPDO 6: Remote TxPDO send request 7: Remote RxPDO send request 8: SDO information 9–15: Reserved

✧ SDO service

CoE communication service types 2 and 3 are SDO communication services, and the SDO data structure is shown in Figure 7-27.

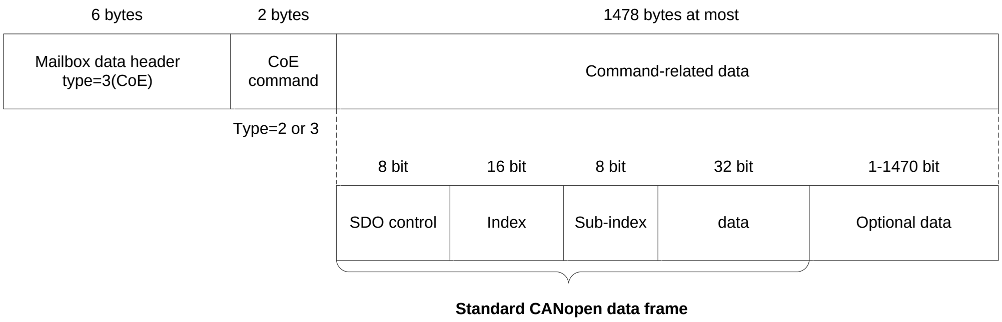


Figure 7-27 SDO data frame format

SDO usually has three transmission modes. Table 7-9 shows the specific content of the SDO data frame. Its structure is shown in Figure 7-28:

Fast transmission service: As with the standard CANopen protocol, only 8 bytes are used and up to 4 bytes of valid data can be transmitted.

Regular transmission service: More than 8 bytes can be used to transmit more than 4 bytes of valid data. The maximum valid data that can be transmitted depends on the storage area capacity managed by the mailbox SM.

Segmented transmission service: Use this service when the capacity of the mailbox is exceeded.

Table 7-9 CoE data frame content

SDO control	Standard CANopen SDO service
Index	Device object index
Sub-index	Sub-index
Data	Data in SDO
Data (Optional)	There are four bytes of optional data that can be added to the data frame.

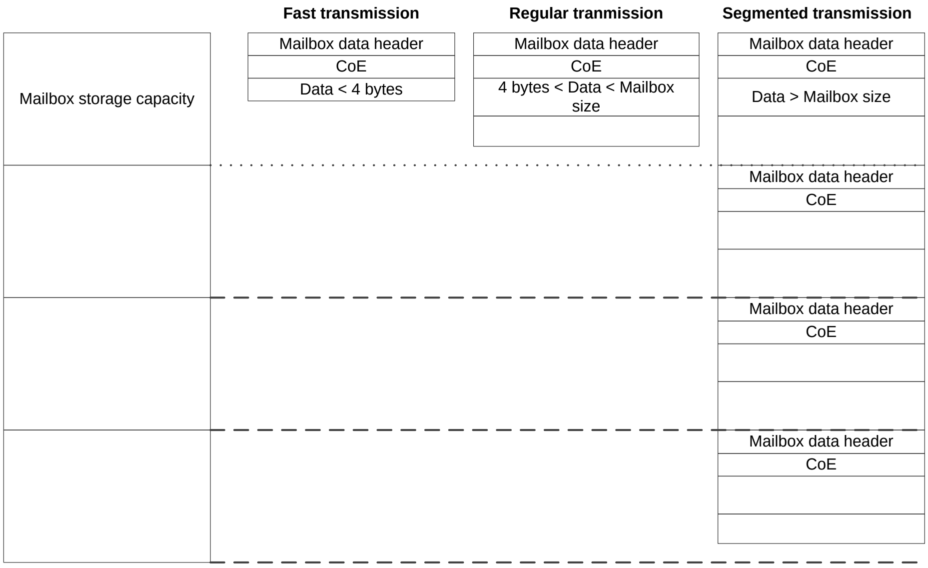


Figure 7-28 SDO transmission type

If the data to be transmitted is larger than 4 bytes, the regular transmission service is used. In regular transmission, the 4 data bytes in the fast transmission mode will be used to indicate the full size of the data to be transmitted. The valid data is transmitted in the extended data section. The maximum size of the valid data is the mailbox capacity minus 16.

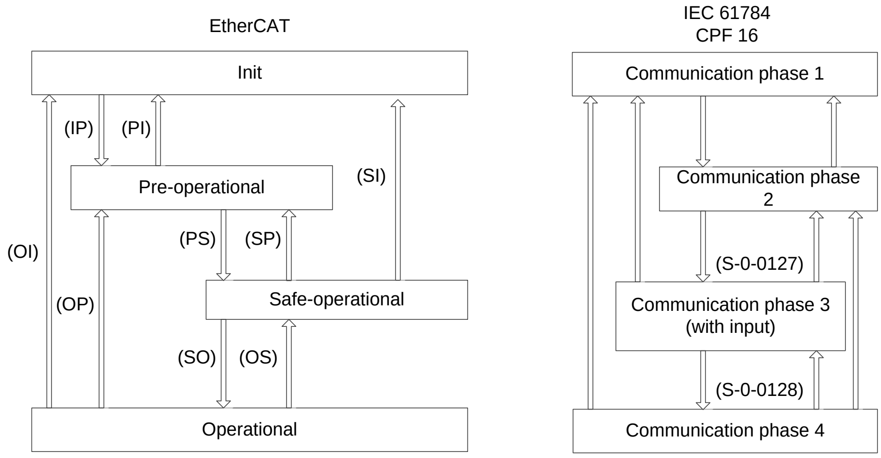
7.4.2 Servo drive profile according to IEC 61800-7-204 (SERCOS)

SERCOS is known as a real-time communication interface, especially for motion control applications. The SERCOS profile for servo drives is included in the international standard IEC61800-7-204. The mapping of this profile to EtherCAT is defined in section 304 of the standard. The service channel, including access to all drive-internal parameters and functions, is based on the EtherCAT mailbox. Here too, the focus is on compatibility with the existing protocol (access to value, attribute, name, units of the IDNs) and expandability with regard to data length limitation. The process data, with SERCOS in the form of AT and MDT data, are transferred using EtherCAT device protocol mechanisms. The mapping is similar to the SERCOS mapping. The EtherCAT slave state machine can also be mapped easily to the phases of the SERCOS protocol.

7.4.2.1 SoE state machine

A comparison between the communication phase of the SERCOS protocol and the EtherCAT state machine is shown in the Figure 7-29. The SoE state machine is featured as follows:

1. SERCOS protocol communication phase 0 and 1 are overwritten by EtherCAT initialization state.
2. Communication phase 2 corresponds to the operational state, allowing the use of mailbox communication to implement the service channel and operate IDN parameters.
3. Communication phase 3 corresponds to the safe operational state and starts transmitting periodic data, where only input data is valid and output data is ignored, implementing clock synchronization.
4. Communication phase 4 corresponds to the operational phase, where all inputs and outputs are valid.
5. Phase switching process commands S-0-0127 (communication phase 3 switchover check) and S-0-0128 (communication phase 4 switchover check) that do not use the SERCOS protocol are replaced by PS and SO state conversion respectively.
6. The SERCOS protocol only allows switching down from the advanced communication phase to communication phase 0, whereas EtherCAT allows any state switching down (as shown in a) in Figure 7-29. For example, switching from the operational state to the safe operational state or from the safe operational state to the pre-operational state. The SoE should also support this switchover as shown in b) in Figure 7-29. If the slave does not support this switchover, set the error bit in the EtherCAT AL state register.



a) EtherCAT state machine b) SERCOS state machine

Figure 7-29 SoE state machine

7.4.2.2 IDN inheritance

The SoE protocol inherits the DIN parameter definition of the SERCOS protocol. Each IDN parameter has a unique 16-bit IDN, which corresponds to a unique data block that holds all information about the parameter. The data block consists of 7 elements, as listed in Table 7-10. The IDN parameters are divided into standard data and product data, and each part consists of eight parameter groups with different IDN, as listed in Table 7-11.

Table 7-10 IDN data block structure

No.	Name
Element 1	IDN
Element 2	Name
Element 3	Attribute
Element 4	Unit
Element 5	Minimum allowable value
Element 6	Maximum allowable value
Element 7	Data value

Table 7-11 IDN number definition

Bit	15	14-12	11-0
Meaning	Classification	Parameter group	Parameter number
Value	0: Standard data (S) 1: Product data (P)	0–7: 8 parameter groups	0000-4095

When using EtherCAT as a communication network, some IDNs in the SERCOS protocol for communication interface control have been deleted, as listed in Table 7-12. And some IDN has been modified, as listed in

Table 7-13.

Table 7-12 Deleted IDN

IDN	IDN description
S-0-0003	Minimum start time of AT sending
S-0-0004	Time between sending and receiving state switching
S-0-0005	Minimum feedback sampling lead time
S-0-0009	Start address in the master data message

IDN	IDN description
S-0-0010	Master data message length
S-0-0088	Recovery time required for receiving MSTs after receiving MDTs
S-0-0090	Command processing time
S-0-0127	Communications phase 3 switchover check
S-0-0128	Communications phase 4 switchover check

Table 7-13 Modified IDN

IDN	Original description	Updated description
S-0-0006	Start time of AT sending	Time offset in which an application writes AT data to ESC memory after a synchronization signal within the slave.
S-0-0014	Communication interface state	Map slave DL state and AL state code.
S-0-0028	MST error technology	Map the slave RX error counter to the loss counter.
S-0-0089	Start time of MDT sending	Time offset of obtaining MDT data from ESC memory after a synchronization signal within the slave.

7.4.2.3 SoE periodic process data

Output process data (MDT data content) and input process data (AT data content) are configured by S-0-0015, S-0-0016 and S-0-0024. The process data only includes periodic process data, but not service channel data. The output process data includes servo control words and command data, while the input process includes status words and feedback data. S-0-0015 sets the type of periodic process data, as listed in Table 7-14, and the definition of parameters S-0-0016 and S-0-0024 are listed in Table 7-15. The master writes these three parameters via mailbox communication during the Pre-Operational phase to configure the contents of the periodic process data.

Table 7-14 Definition of parameter S-0-0015

S-0-0015	Command data	Feedback data
0: Standard type 0	None	No feedback data
1: Standard type 1	Torque command S-0-0080 (2 bytes)	No feedback data
2: Standard type 2	Speed command S-0-0036 (4 bytes)	Speed feedback S-0-0053 (4 bytes)
3: Standard type 3	Speed command S-0-0036 (4 bytes)	Position feedback S-0-0051 (4 bytes) Speed feedback S-0-0053 (4 bytes)
4: Standard type 4	Position command S-0-0047 (4 bytes)	
5: Standard type 5	Position command S-0-0047 (4 bytes) Speed command S-0-0036 (4 bytes)	Position feedback S-0-0051 (4 bytes) Or speed feedback S-0-0053 (4 bytes) + Position feedback S-0-0051 (4 bytes)
6: Standard type 6	Speed command S-0-0036 (4 bytes)	No feedback data
7: Custom	S-0-0024 configuration	S-0-0016 configuration

Table 7-15 Definition of parameters S-0-0016 and S-0-0016

Data word	S-0-0024 definition	S-0-0016 definition
0	Maximum length of output data (Word)	Maximum length of input data (Word)
1	Actual length of output data (Word)	Actual length of input data (Word)
2	First IDN of command data mapping	First IDN of feedback data mapping
3	Second IDN of command data mapping	Second IDN of feedback data mapping
…	…	… … …

7.4.2.4 SoE non-periodic service channels

The EtherCAT SoE Service Channel (SSC) is done by the EtherCAT mailbox communication function, which is used for non-periodic data exchange, such as reading and writing IDNs and their elements. The SoE data header format is shown in Figure 7-30.

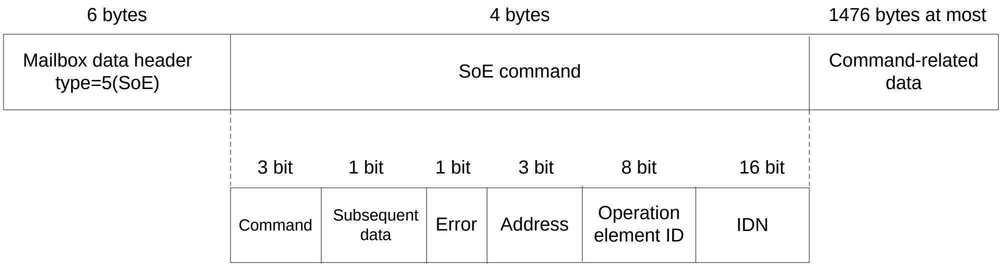


Figure 7-30 SoE data header format

Table 7-16 SoE data command description

Data area	Description
Command	Command type: 0x01: Read request 0x02: Read response 0x03: Write request 0x04: Write response 0x05: bulletin 0x06: Slave information 0x07: Reserved
Subsequent data	Subsequent data signal: 0x00: No subsequent data frame 0x01: Transmission incomplete, with subsequent data frame
Error	Error signal: 0x00: No error 0x01: Error occurred, 2-byte error code in data area
Address	Specific address of the slave device
Operation element identification	Element selection for single element operation, defined by bit, with each bit corresponding to one element. Number of elements for addressing constructs
IDN	IDN number of the parameter, or the remaining segments during the segment operation

Commonly used SSC operations include SSC read operations, SSC write operations, and process commands.

- SSC read operation: The master initiates the SSC read operation and writes the SSC request to the slave. After receiving the read operation request, the slave responds with the requested IDN number and data value. The master can read multiple elements at the same time, so the slave should answer multiple elements. If the slave only supports single element operation, it should respond with the first element requested.
- SSC write operation: This operation is used to download data from the master to the slave, which should answer with the result of the write operation. Segment operation consists of one or more segmented write operations and an SSC write response service.
- SSC process command: A process command is a special non-periodic data. Each process command has a unique IDN and specified data elements, which are used to start certain specific functions or processes of the servo device. It usually takes a while to execute these functions or processes. The process command only triggers the start of the process, so after that, the service channel it occupies will become immediately available for the transfer of other non-periodic data or process commands. There is no need to wait until the triggered functions or processes to complete their execution.

8 Application Programming

8.1 Single axis control

8.1.1 Single axis control programming description

The motion control of the AX series controller with the servo axis (such as DA200) is implemented based on the EtherCAT bus network. Each EtherCAT bus cycle will perform a calculation and issue a control command to control the servo. Different from the previous pulse control mode, EtherCAT bus is entirely based on the software. Pay attention to the following points when applying:

- MC-related POUs should be configured to execute under the EtherCAT task. Most MC function blocks cannot run normally when placed in the POU of the low-priority Main tasks.
- The PDO configuration table needs to be configured with relevant data objects. Otherwise the servo will not be able to run due to the missing communication data object configuration. No error alarm will be generated for this case, making it more difficult to troubleshoot.
- The controller can set the parameters of the servo by configuring SDO.
- MC function block instance can only be used for a unique servo axis control. Error occurs if it is used for multiple servo axis controls.
- MC function block must be used to monitor the running servo axis to avoid error caused by program logic jump without MC function block monitoring. Such error is usually difficult to detect.
- Pay attention to the safe handling of the debugging, and ensure that the signal configuration is consistent with the practical application. If the servo system uses incremental encoder, zeroing is required prior to normal operation. For movements within a limited range (e.g. a screw), limit and safety signals should be set.

8.1.2 MC function blocks commonly used for single-axis control

MC function block (FB) is also known as MC command. In fact, the object instance of MC function block is used in the user program, and the servo axis is controlled by MC object instance, for example:

```
MC_Power1: MC_Power;//Statement instance MC_Power1

MC_Power1 (Axis=Axis1,)
```

Single-axis control is generally used for positioning control, that is, the servo motor drives the external mechanism to move to the specified position. Sometimes the servo is required to run at a specified speed or torque. In single-axis control, the following MC function blocks are commonly used:

Table 8-1 MC function blocks commonly used for single-axis control

Control operation	Required MC command	Description
Enable servo	MC_Power	Run this command to enable the servo axis to perform subsequent running control.
Absolute positioning	MC_MoveAbsolute	Command the servo to run to a specified coordinate point.
Relative positioning	MC_MoveRelative	Runs the specified distance with the current location as a reference.
Servo jog operation	MC_Jog	The jog operation of the servo motor is often used for low-speed test runs to inspect equipment or adjust the position of the servo motor.

Control operation	Required MC command	Description
Relative superposition positioning	MC_MoveAdditive	Based on the current running command of the servo, run the specified distance relatively.
Speed control	MC_MoveVelocity	Command the servo runs at the specified speed.
Servo suspend	MC_Halt	Command the servo to suspend operation. If MC_Movexxx is triggered again, the servo can run again.
Emergency stop	MC_Stop	Command the servo to stop. The servo can run again only after the stop command is reset and MC_Movexxx is triggered.
Alarm reset	MC_Reset	When the servo stops with an alarm, this command is used to reset the servo.
Servo homing	MC_Home	Command the servo to start homing operation. Both the home signal of the application system and the limit signals on both sides are connected to the DI port of the servo.
Controller homing	MC_Homing	Command the control system to start homing operation. Both the home signal of the application system and the limit signals on both sides are connected to the DI port of the controller.

8.2 Cam synchronization control

Electronic cam (abbreviation ECAM) utilizes the constructed cam curves to simulate the mechanical cam to meet the relative motion software system between main shaft and camshaft system the same to mechanical cam system. Electronic cams can be applied to various fields, such as automobile manufacturing, metallurgy, machining, textiles, printing, and food packaging. The electronic cam curve is a function curve with the main shaft pulse (active shaft) input as X and the corresponding output of the servo motor (camshaft) as Y=F(X).

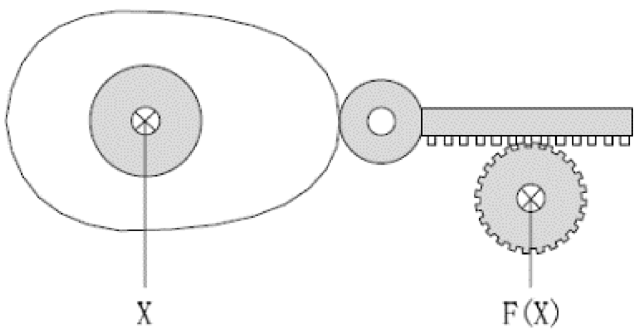


Figure 8-1 Electronic cam diagram

The AX series programmable controller electronic cam function has the following features.

- CAM curves are easy to draw: Cams can be described by cam chart, CAM curves or array. It supports multiple cam chart selection and dynamic switching during running.
- CAM curves are easy to correct: The running cam table can be modified dynamically.
- Support one master and multiple slaves: one main shaft can have multiple slave shafts corresponding to it.
- Cam lifter: multiple cam lifters and multiple setting intervals are allowed.
- Cam clutch: It can make the cam enter and exit the cam running through the user program.

- Special functions: Virtual main shaft, phase offset and output superposition are supported.

Note: "online modification of CAM curve" refers to the modification of the key point coordinates of the CAM curve according to the needs of control characteristics during the execution of the program written by the user. The content to be modified is generally the key point coordinates, but it can also be the number of key points, the distance range of the main axis.

The AX series programmable controller electronic cam function contains three control elements:

1. Main shaft: Reference for synchronous control.
2. Slave shaft: a servo axis that follows the movement of the main shaft according to the non-linear characteristics.
3. Cam table: Data table or cam curve describing the relative position, range, periodicity of the master-slave shafts.

The commonly used function blocks related to electronic cam are listed in the following table.

Table 8-2 Commonly used electronic cam function blocks

MC Command	Description
MC_CamTableSelect	Run this command to associate the main shaft, slave shaft and cam table.
MC_CamIn	Let the slave shaft enter the cam running
MC_CamOut	Let the slave shaft exit the cam running
MC_Phasing	Main shaft phase modification

8.2.1 Periodic mode of the cam table

1. Single cycle mode (Periodic:=0): After the cam table cycle is completed, the slave shaft leaves the cam running state, as shown in Figure 8-2.

Figure 8-2 Single cycle mode

2. Periodic mode (Periodic:=1): After the cam table cycle is completed, the slave shaft will start the next cam cycle until the user program commands it to exit the cam running state, as shown in Figure 8-3.

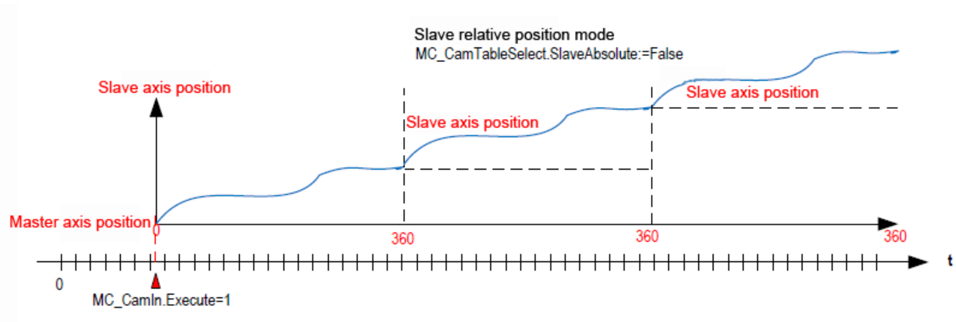


Figure 8-3 Periodic mode

8.2.2 Input method of cam table

1. When creating a new cam table, the system will automatically generate the simplest cam curve, on which the user can edit and customize the CAM curve table.
2. User can increase or decrease the number of key points in the cam curve or change the coordinates of the key points.
3. The line pattern between the two key points of the cam curve can be set to a straight line or a quantic polynomial, and the system will optimally optimize each curve to minimize sudden changes in speed and acceleration.

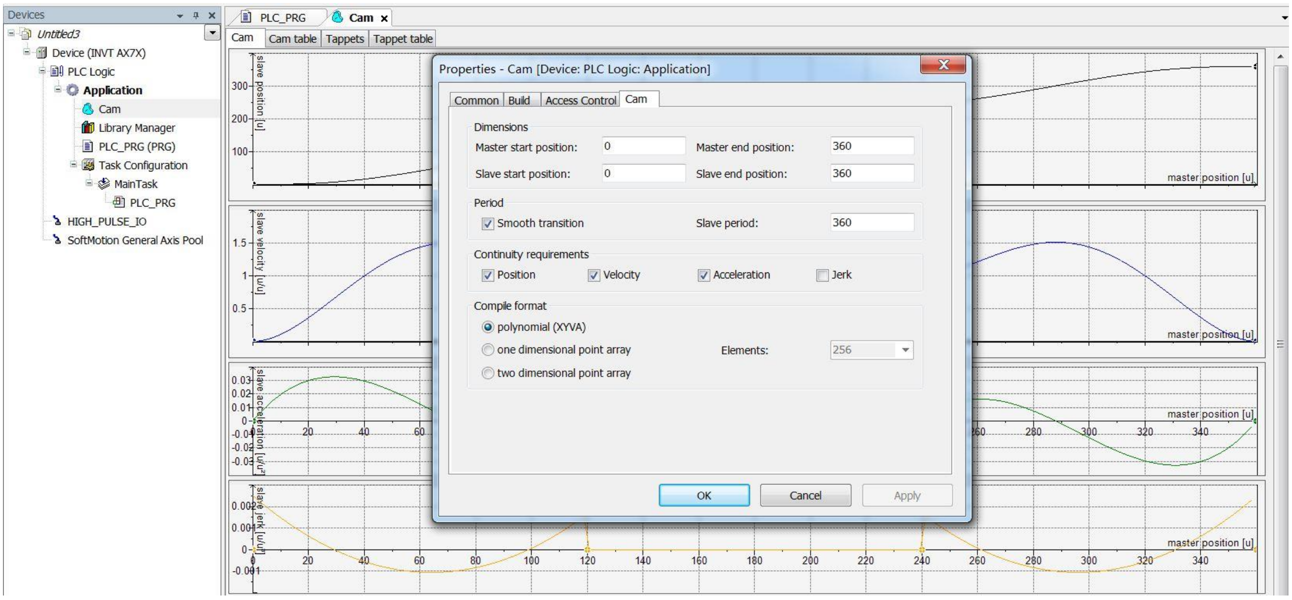


Figure 8-4 CAM curve

8.2.3 Data structure of cam table

Invtmatic Studio contains data structure for each CAM table that describes the feature data of the CAM table. The following figure describes the data structure of the "CAM0" cam table. Please note the names of the variables in the structure.

Cam | Cam table | Tappets | Tappet table

	X	Y	V	A	J	Segment Type	min(Position)	max(Position)	max(|Velocity|)	max(|Acceleration|)
	0	0	0	0	0					
						Poly5	0	120	1.51200000000...	0.032835282941414...
	120	120	1	0	0					
						Poly5	120	240	1	0
	240	240	1	0	0					
						Poly5	240	360	1.512	0.032835282941414...
	360	360	0	0	0					

Figure 8-5 Data structure of cam table

Invtmatic Studio has an internal data structure to characterize the CAM table. We can also write a CAM table manually, or modify the CAM feature data by accessing the data structure.

Note: When we state the CAM0 cam table, the system automatically states the CAM0 data structure of the global variable type by default, along with the CAM0_A[i] array. For example, modify the number of key points or coordinates of the CAM0 cam table in the user program.

`CAM0. nElements:=10;` // Change the number of key points to 10.

`CAM0. xEnd:=300;` // Change the end point of the main shaft to 300.

//For example, modify the coordinates of two key points in the user program.

```
CAM0_A[2].dx:=10;
CAM0_A[2].dy:=30;
CAM0_A[2].dv:=1;
CAM0_A[2].da:=0;
CAM0_A[3].dx:=30;
CAM0_A[3].dy:=50;
CAM0_A[3].dv:=1;
CAM0_A[3].da:=0;
```

8.2.4 CAM table reference and switch

CAM table is stored in the controller with an array, which can be pointed to by specific MC_CAM_REF variable type, such as statement:

CAM table q: `MC_CAM_REF;`

You can assign a value to this variable, namely pointing it to a specific CAM table:

```
CAM table q:= Cam0;  // Point to the required CAM table.
CAM table q: MC_CAM_REF;  // Cam table pointer;
TableID: uint; // Cam table selection command that can be set by HMI;
Case TableID of
0: CAM table q: = CAM table A;
1: CAM table q: = CAM table B;
2: CAM table q: = CAM table C;
End_case
MC_CamTableSelect_0(  //CAM relationship
Master:= Virtual main shaft,
Slave:= CAM slave shaft,
CamTable:= CAM table q,
Execute:= bSelect, // Rising edge triggers CAM table selection.
Periodic:= TRUE,
MasterAbsolute:=FALSE,
SlaveAbsolute:= FALSE);
```

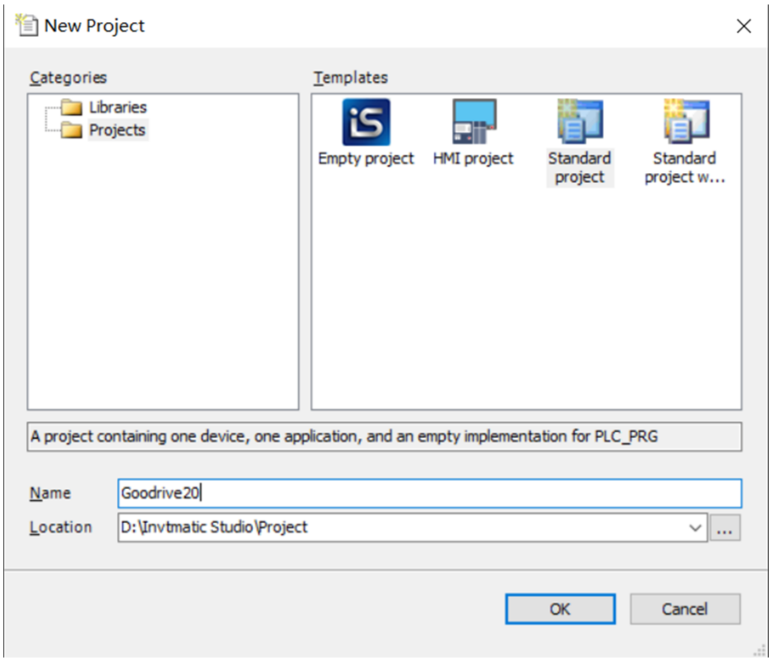
In the above example, the assignment operation of the MC_CAM_REF variable can be used to switch multiple CAM tables.

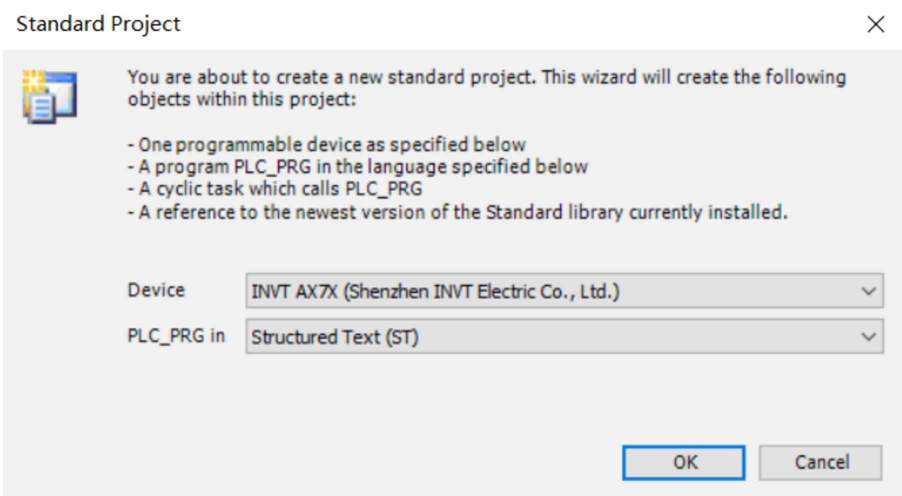
Appendix A Project Instance

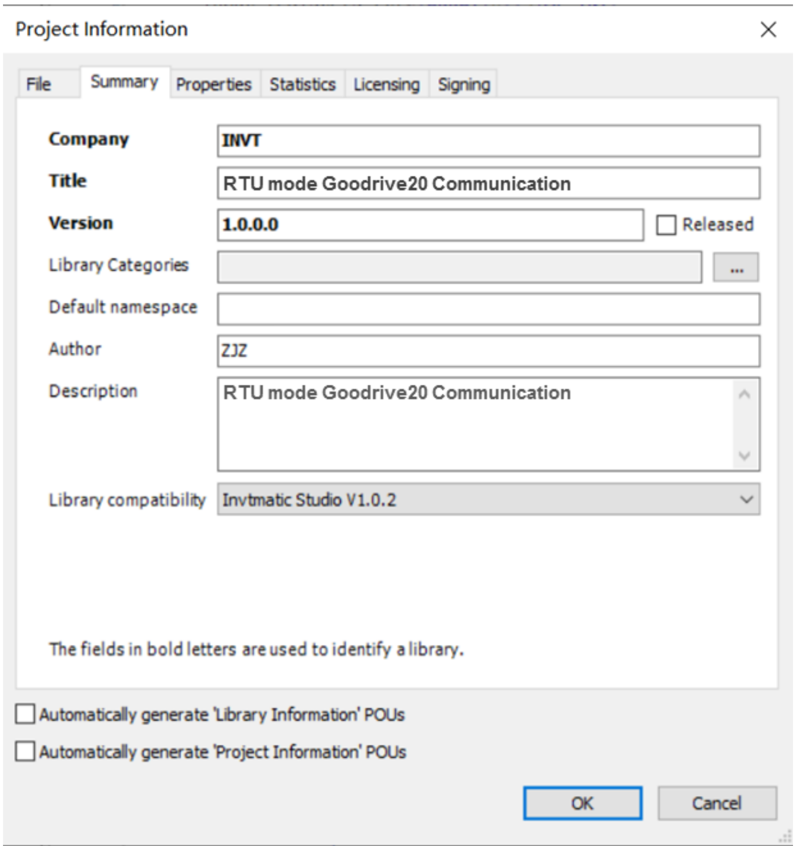
A.1 Controller and Goodrive20 Series VFD Configuration Example

The AX Series controller is now set up as the master and a Goodrive 20 Series VFD is set up as the slave. The controller uses the Modbus/RTU communication protocol with a two-wire RS485 physical layer and communicates with the VFD via the COM2 port. Let's write a small program that reads and writes the functional parameters of the Goodrive20 VFD with the upper computer.

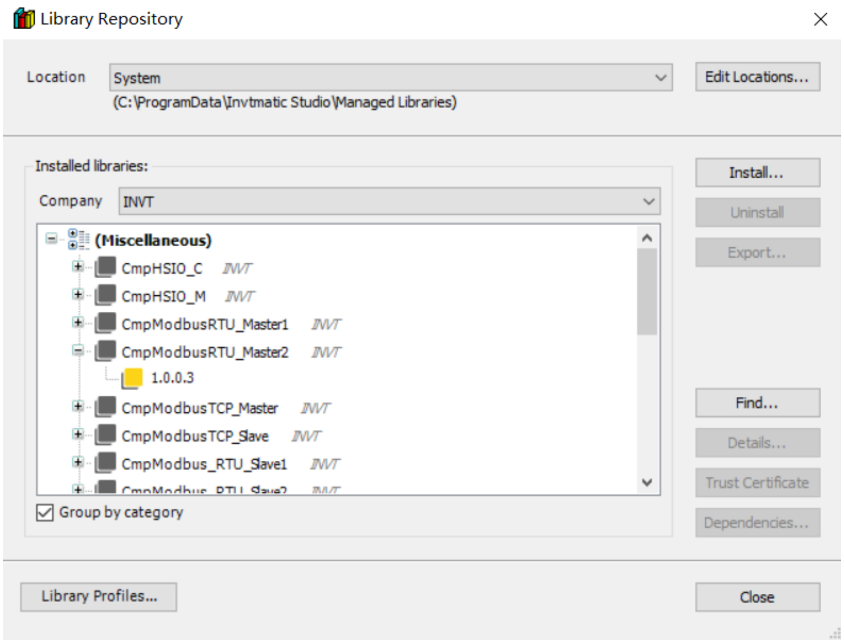
1. Select **File** > **New Project** from the menu to create a new standard project. Set the device to **INVT AX7X**, and select **Structured Text (ST)** as the programming language. Edit the project information as needed, as shown in the following figure.



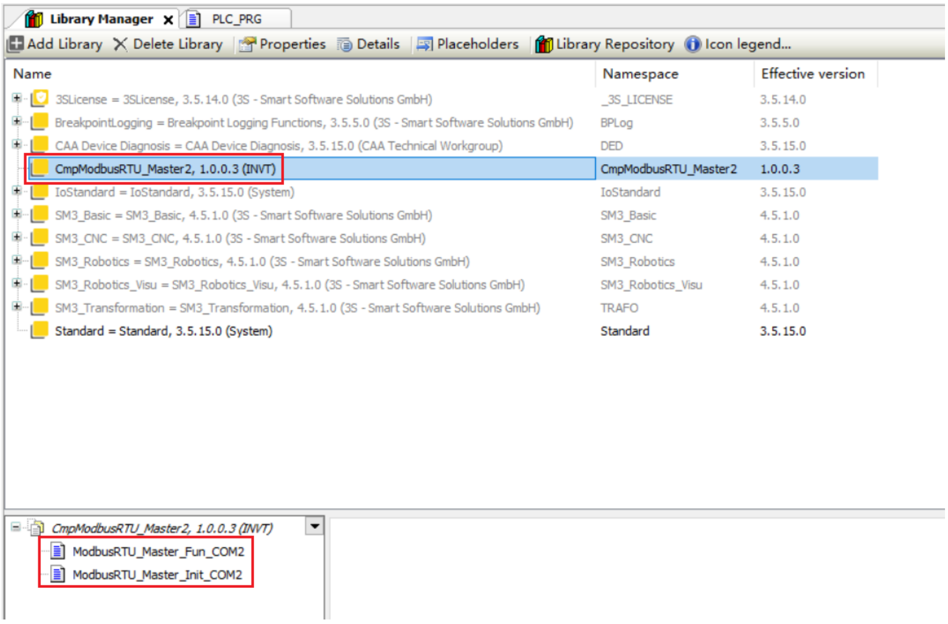




2. Select **Tool** > **Library Repository** from the menu, and install the library file **CmpModbusRTU_Master2_1.0.0.3.library**, as shown in the following figure.



3. Select **Library Manager** > **Add Library** to add the installed library to the application, as shown in the following figure.



4. Double-click the PLC_PRG and enter the following codes on the statement editor:

```
PROGRAM PLC_PRG
VAR
    ModbusRTU_Master_Fun_COM2: ModbusRTU_Master_Fun_COM2;
    ModbusRTU_Master_Init_COM2: ModbusRTU_Master_Init_COM2;
    DatePtr2:ARRAY[0..0]OF INT;
    input_registers_Ptr2:ARRAY[0..9]OF INT;
    CoilDataPtr2:ARRAY[0..9]OF BOOL;
    input_bits_Ptr2:ARRAY[0..9]OF BOOL;
    CoilSingleData2:INT;
    Fun_Code2:INT;
    Addr2:UINT;
    DataCount2 : UINT: =1;
END_VAR
```

Enter the following code in the main code editor:

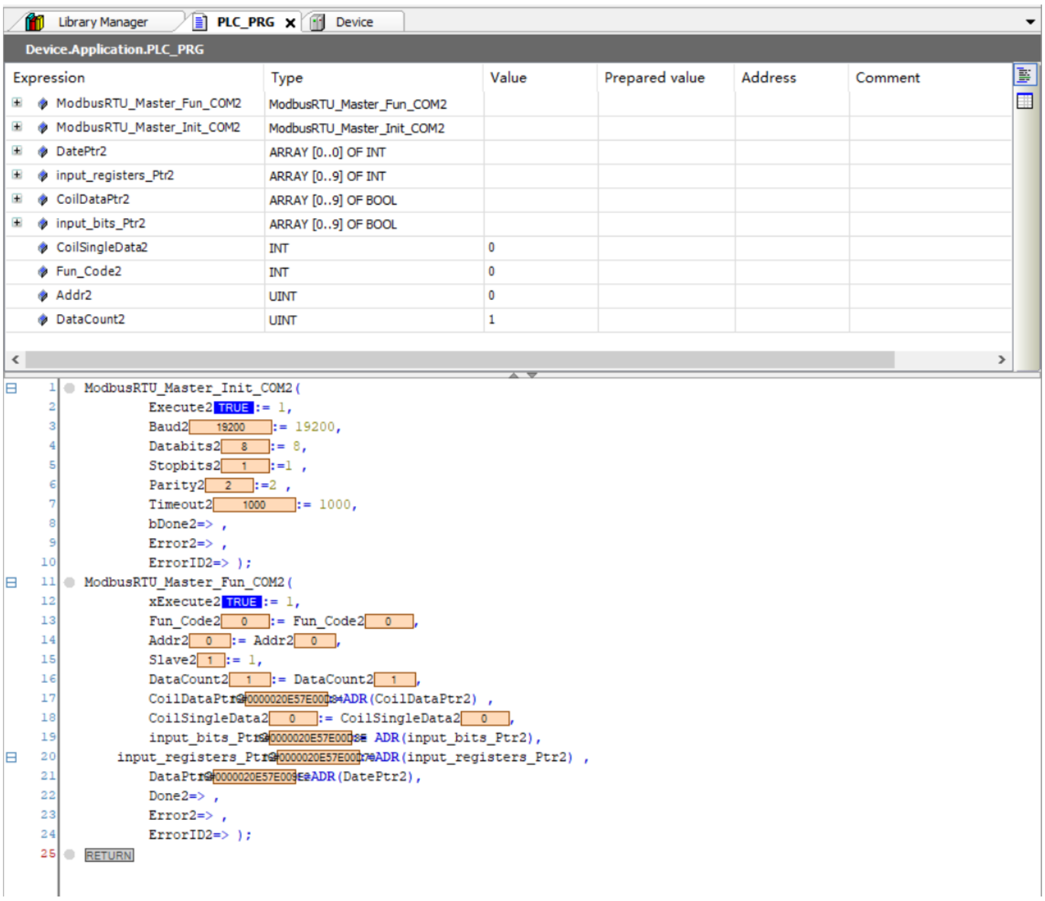
```
ModbusRTU_Master_Init_COM2(
        Execute2:= 1,
        Baud2:= 19200,
        Databits2:= 8,
        Stopbits2:=1 ,
        Parity2:=2 ,
        Timeout2:= 1000,
        bDone2=> ,
        Error2=> ,
```

```
        ErrorID2=> );

ModbusRTU_Master_Fun_COM2(

        xExecute2:= 1,

        Fun_Code2:= Fun_Code2,

        Addr2:= Addr2,

        Slave2:= 1,

        DataCount2:= DataCount2,

        CoilDataPtr2:=ADR(CoilDataPtr2) ,

        CoilSingleData2:= CoilSingleData2,

        input_bits_Ptr2:= ADR(input_bits_Ptr2),

    input_registers_Ptr2:=ADR(input_registers_Ptr2) ,

        DataPtr2:=ADR(DatePtr2),

        Done2=> ,

        Error2=> ,

        ErrorID2=> );
```

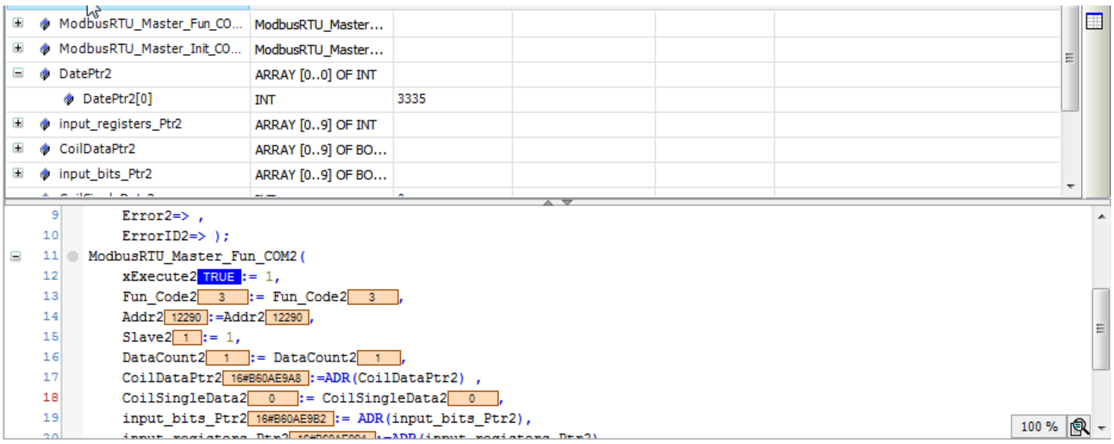
Here are some descriptions of the program. The program calls two function blocks of the CmpModbusRTU_Master2 library, ModbusRTU_Master_Init_COM2 and ModbusRTU_Master_Fun_COM2. ModbusRTU_Master_Init_COM2 is used to initialize the RTU Master2, where the baud rate is set to 19200, the data bit is 8, the stop bit is 1, the check bit is even check, and the timeout time is 1000ms. ModbusRTU_Master_Fun_COM2 is the enablement and specific application of the function module. The variable Fun_Code2 is the standard Modbus function code, Addr2 is the address of the VFD Goodrive20 function. For the address of other MODBUS functions, refer to the INVT Goodrive20 Series VFD product manual. Slave2 indicates the VFD slave address, which is set to 1 here.

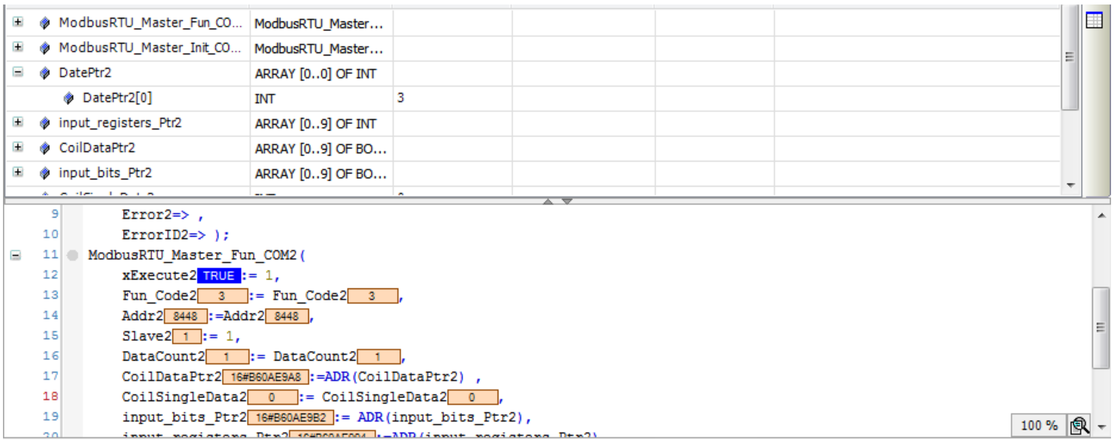
Connect the VFD and the controller with the two-wire RS485, and then start the VFD. Set the function code P00.01 to 2 through the VFD keypad, so that the running command can be controlled by the upper computer through communication modes. Set P00.06 to 8 to select the MODBUS communication mode. Set the serial communication parameters of group P14 to make it consistent with the initial parameter settings of the upper computer, including baud rate, data bit, parity bit, slave address, timeout time.

Click the  button on the toolbar to compile the code. After compiling, click the  button on the toolbar to log in to the controller. Check that the controller digital tube has no error, the VFD Goodrive20 is connected to the controller smoothly, and the communication is normal. The upper computer interface is shown in the figure.

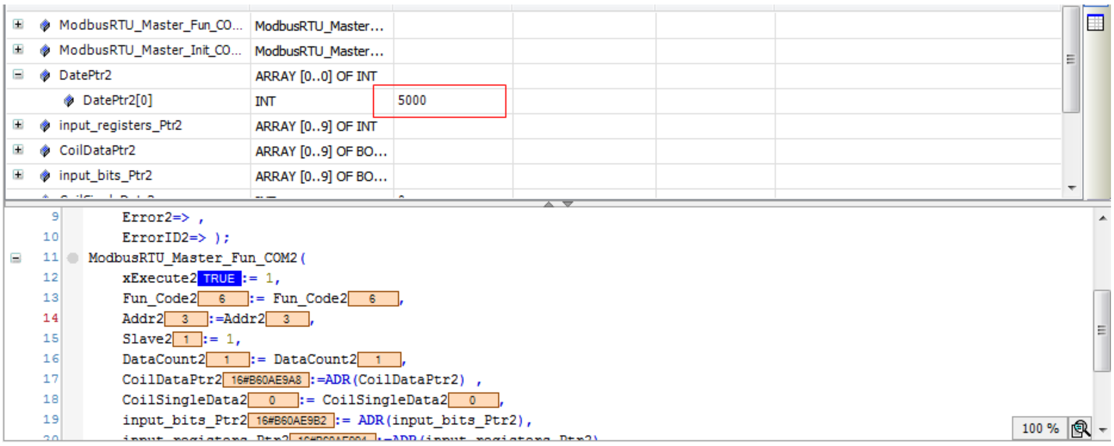


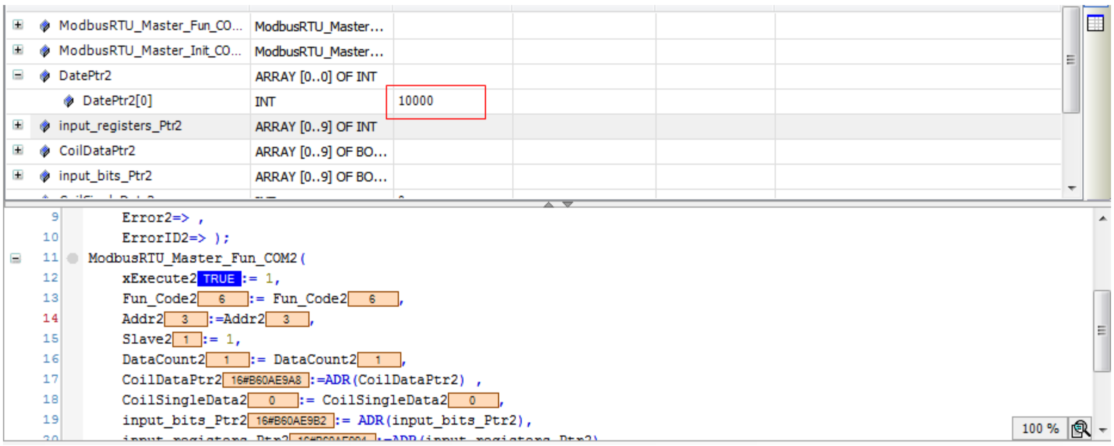
Now we take an example of the read operation. Write the value to the variable in the login state. Write 3 to the Fun_Code, which means 03H function code Read Holding Registers. Write 16#3002 to the Addr, which means that one address is read from 3002H. The value 3335 can be read from the array DataPtr2 (i.e. 3002H address), which means the bus voltage is 333.5V with reference to the VFD product manual. Similarly, write 3 to the Fun_Code, which means 03H function code Read Holding Registers. Writet 16#2100 to the Addr. The value 3 can be read from the array DataPtr2 (i.e. 2100H address), which means the VFD is down with reference to the VFD product manual.





Now we take an example of the write operation. Write the value to the variable in the login state. Write 6 to the Fun_Code, which means 06H function code Write Single Register. Write 16#0003 to the Addr, which means to write a value to the address 0003H. Referring to the VFD product manual, 0003H is the address of the maximum output frequency of the VFD with a default value of 50.00 HZ. Before writing the value of the address, the value of the address 0003H in the upper computer is 5000 which is obtained by 50.00Hz multiplied by the scale value of 100. If the maximum output frequency of the VFD is set to 100Hz, write the 0003H with value 100Hz*100, that is, 10000. After that, the value of P00.03 will change from 50.00 to 100.00, indicating that the controller wrote successfully to the VFD. See the figure.

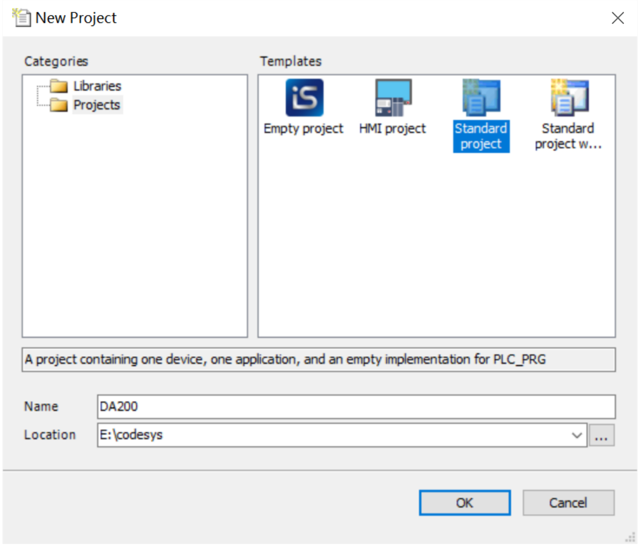


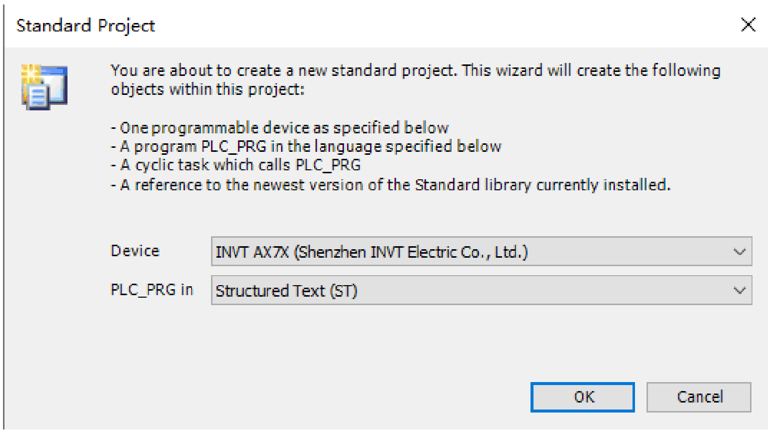


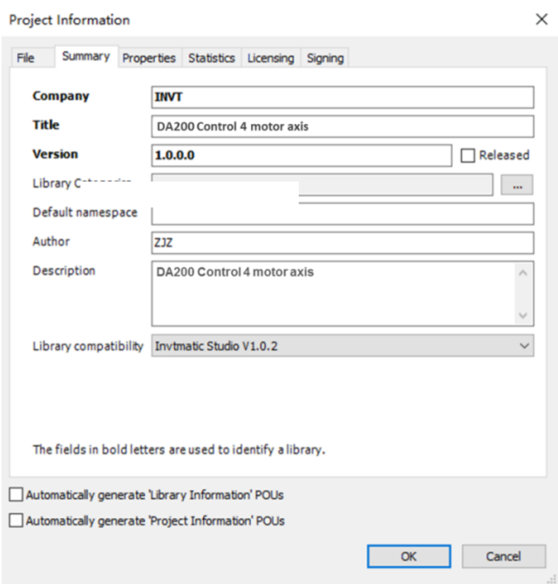
A.2 Controller and DA200 Series Servo Drive Configuration Example

In this section, we will write a program to control four DA200 series servo drives to drive four motor axes for constant forward and reverse motion.

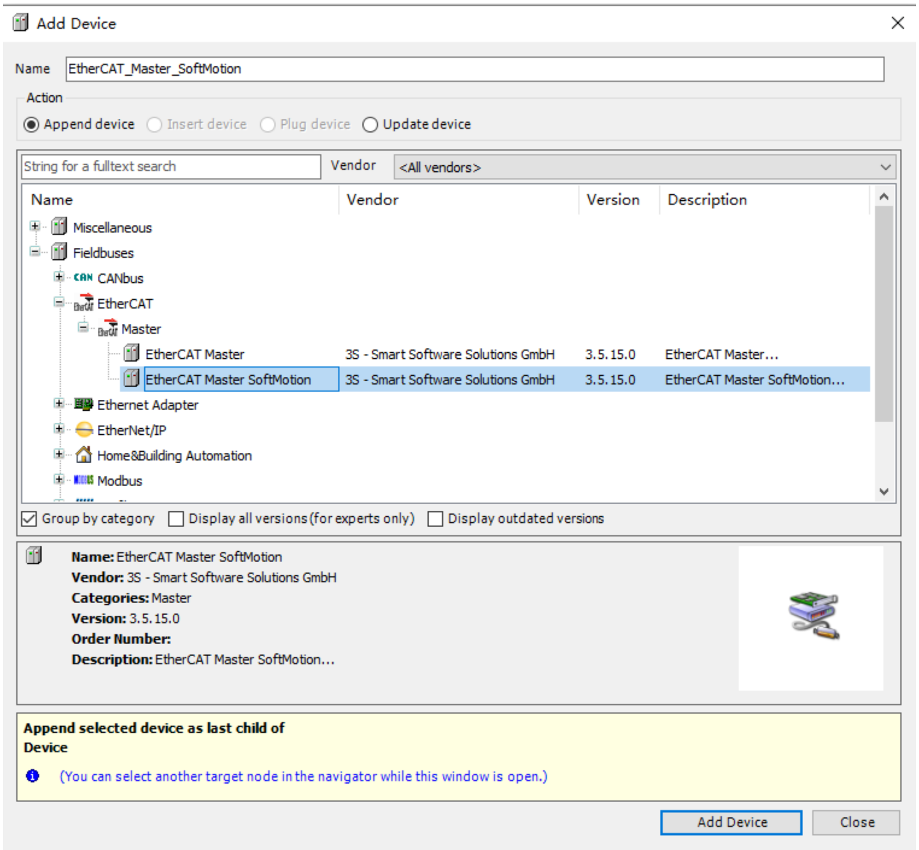
1. Select **File** > **New Project** from the menu to create a new standard project. Set the device to **INVT AX7X**, and select **Structured Text (ST)** as the programming language. Edit the project information as needed, as shown in the following figure.



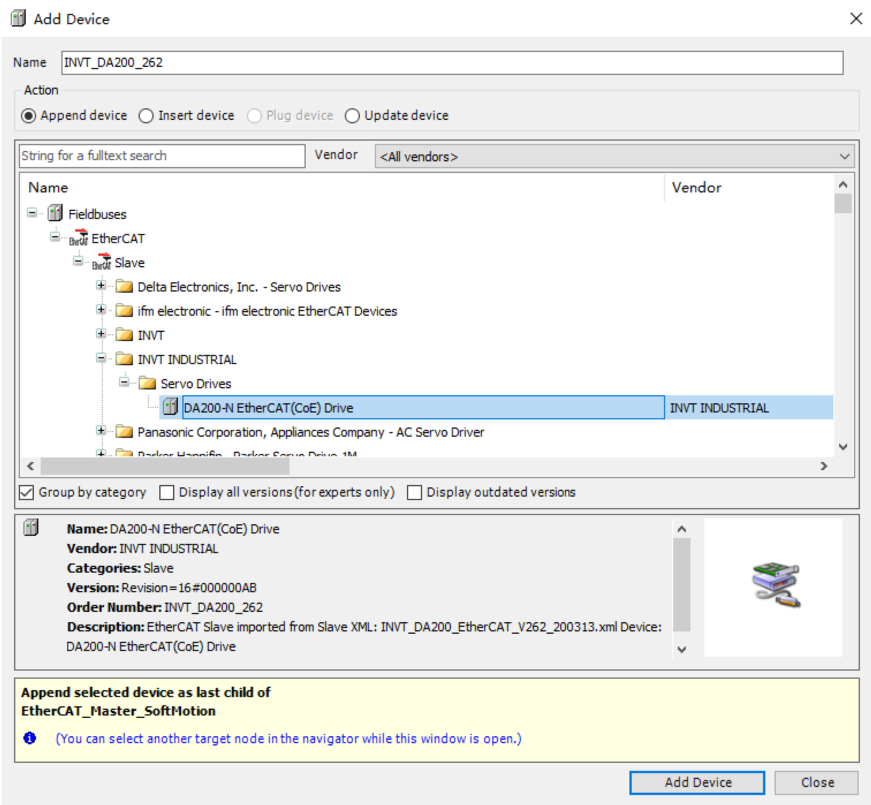




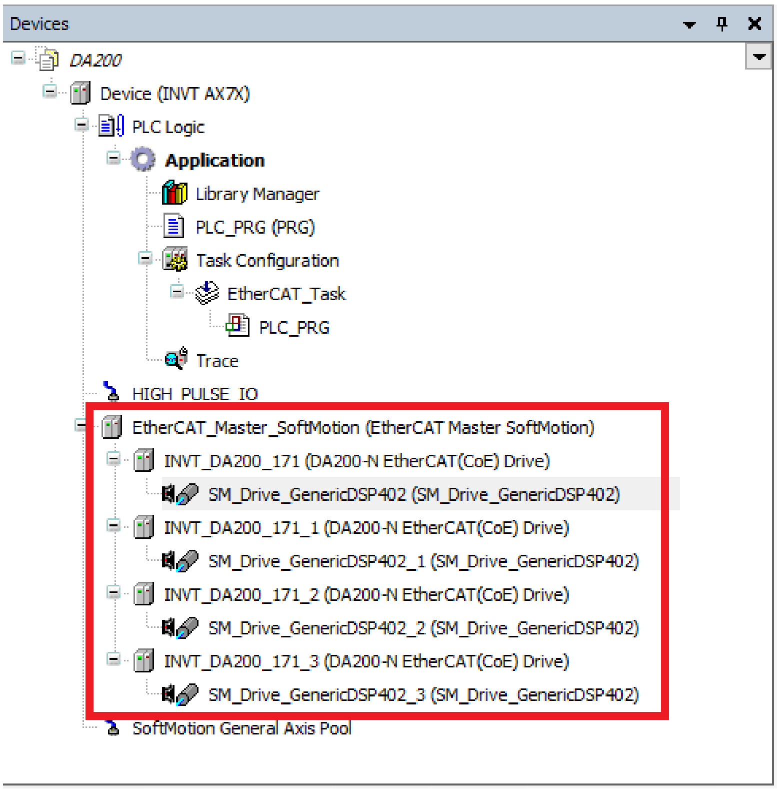
2. Right click the device from the device panel and select **Add Device** to add the EtherCAT master. Select **EtherCAT Master SoftMotion** with a version of 3.5.15.0, as shown in the following figure.



3. Right click the device **EtherCAT Master SoftMotion** from the device panel and select **Add Device** to add 4 servo drives. Select **INVT_DA200_171**, as shown in the following figure.



4. Right click an **INVT_DA200_171** device in the device panel and select **Add SoftMotion CiA402 Axis**. Preform the same procedure for the remaining 3 INVT_DA200_171 devices, as shown in the figure.



5. Double-click the PLC_PRG and enter the following codes on the statement editor:

```
PROGRAM PLC_PRG
VAR
    iStatus: INT;
    MC_Power_0: MC_Power;
    MC_Power_1: MC_Power;
    MC_Power_2: MC_Power;
    MC_Power_3: MC_Power;
    MC_MoveAbsolute_0: MC_MoveAbsolute;
    MC_MoveAbsolute_1: MC_MoveAbsolute;
    MC_MoveAbsolute_2: MC_MoveAbsolute;
    MC_MoveAbsolute_3: MC_MoveAbsolute;
END_VAR
```

6. Enter the following code in the main code editor:

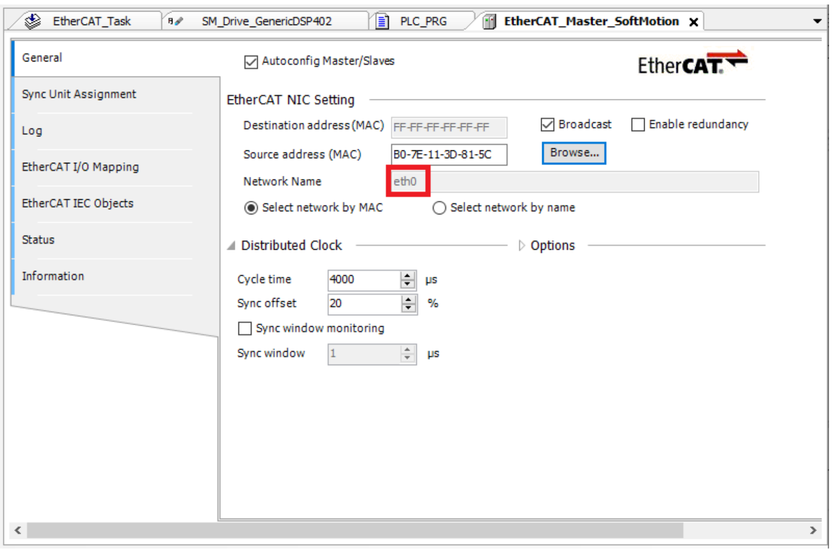
```
CASE iStatus OF
0:
MC_Power_0(Axis:= SM_Drive_GenericDSP402, Enable:= TRUE, bRegulatorOn:= TRUE, bDriveStart:=TRUE , );
MC_Power_1(Axis:= SM_Drive_GenericDSP402_1, Enable:= TRUE, bRegulatorOn:= TRUE, bDriveStart:=TRUE , );
```

```
MC_Power_2(Axis:= SM_Drive_GenericDSP402_2, Enable:= TRUE, bRegulatorOn:= TRUE, bDriveStart:=TRUE , );

MC_Power_3(Axis:= SM_Drive_GenericDSP402_3, Enable:= TRUE, bRegulatorOn:= TRUE, bDriveStart:=TRUE , );

IF MC_Power_0.Status AND MC_Power_1.Status AND MC_Power_2.Status AND MC_Power_3.Status
    THEN

            iStatus:=iStatus+1;

END_IF

1:

MC_MoveAbsolute_0(Axis:=SM_Drive_GenericDSP402 , Execute:= TRUE, Position:=50 , Velocity:=3 , Acceleration:= 2, Deceleration:= 100,);

MC_MoveAbsolute_1(Axis:=SM_Drive_GenericDSP402_1, Execute:= TRUE, Position:=50 , Velocity:=3 , Acceleration:= 2, Deceleration:=100,);

MC_MoveAbsolute_2(Axis:=SM_Drive_GenericDSP402_2, Execute:= TRUE, Position:=50 , Velocity:=3 , Acceleration:= 2, Deceleration:=100,);

MC_MoveAbsolute_3(Axis:=SM_Drive_GenericDSP402_3, Execute:= TRUE, Position:=50 , Velocity:=3 , Acceleration:= 2, Deceleration:=100,);

IF MC_MoveAbsolute_0.Done AND MC_MoveAbsolute_1.Done AND MC_MoveAbsolute_2.Done AND MC_MoveAbsolute_3.Done THEN

            MC_MoveAbsolute_0(Axis:=SM_Drive_GenericDSP402 , Execute:= FALSE,);

    MC_MoveAbsolute_1(Axis:=SM_Drive_GenericDSP402_1 , Execute:= FALSE,);

            MC_MoveAbsolute_2(Axis:=SM_Drive_GenericDSP402_2 , Execute:= FALSE,);

            MC_MoveAbsolute_3(Axis:=SM_Drive_GenericDSP402_3 , Execute:= FALSE,);

            iStatus:=iStatus+1;

END_IF

2:

MC_MoveAbsolute_0(Axis:=SM_Drive_GenericDSP402 , Execute:= TRUE, Position:=0 , Velocity:=3, Acceleration:= 2, Deceleration:= 100,);

MC_MoveAbsolute_1(Axis:=SM_Drive_GenericDSP402_1, Execute:= TRUE, Position:=0 , Velocity:=3 , Acceleration:= 2, Deceleration:=100,);

MC_MoveAbsolute_2(Axis:=SM_Drive_GenericDSP402_2, Execute:= TRUE, Position:=0 , Velocity:=3, Acceleration:= 2, Deceleration:=100,);

MC_MoveAbsolute_3(Axis:=SM_Drive_GenericDSP402_3, Execute:= TRUE, Position:=0 , Velocity:=3 , Acceleration:= 2, Deceleration:=100,);

IF MC_MoveAbsolute_0.Done AND MC_MoveAbsolute_1.Done AND MC_MoveAbsolute_2.Done AND MC_MoveAbsolute_3.Done THEN

            MC_MoveAbsolute_0(Axis:=SM_Drive_GenericDSP402 , Execute:= FALSE,);

            MC_MoveAbsolute_1(Axis:=SM_Drive_GenericDSP402_1 , Execute:= FALSE,);

            MC_MoveAbsolute_2(Axis:=SM_Drive_GenericDSP402_2 , Execute:= FALSE,);
```

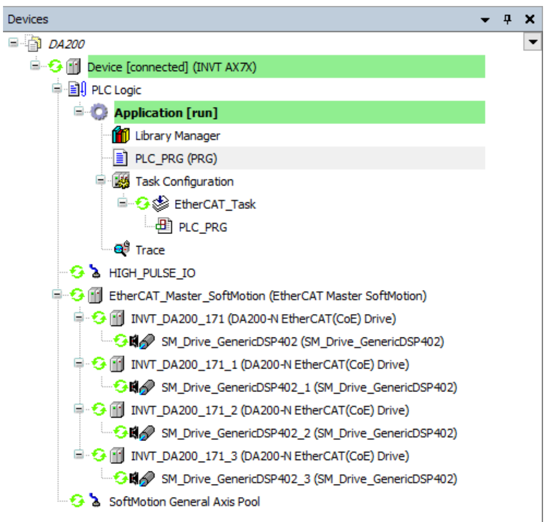
```
            MC_MoveAbsolute_3(Axis:=SM_Drive_GenericDSP402_3 , Execute:= FALSE,);
iStatus:=1;
END_IF
END_CASE
```

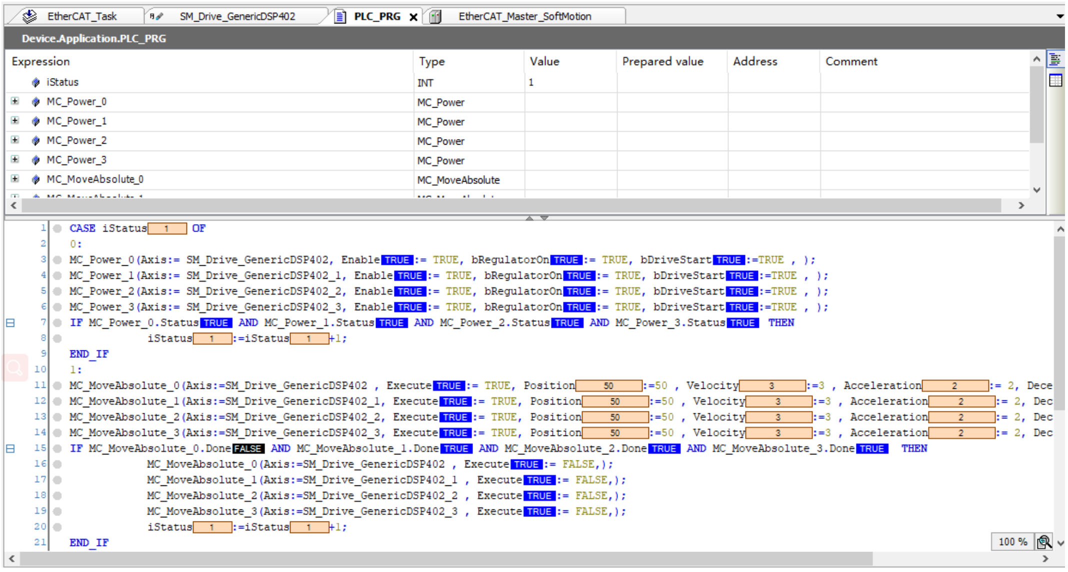
The main body of the program takes the form of a state machine that determines which part of the code to execute through the value of iStatus. When the program starts, the iStatus value is 0 and the program initializes the MC_Power function block and enables the corresponding motor shaft. If the enabling is successful, the iStatus value is 1 and the program enters the next state. When the iStatus value is 1, the MC_MoveAbsolute function block is executed, and the motor rotates to the specified position at the specified speed. If the motor moves normally to the specified position, the iStatus value is increased by 1, and the motor enters the next state. When the iStatus value is 2, execute the MC_MoveAbsolute function block in the other direction. The motor continues to rotate to the specified position at the speed specified by the function block. If the motor moves normally to the specified position, the iStatus value is reset to 1. The procedure is executed repeatedly to implement the forward and reverse movement of the motor.

Double-click **EtherCAT Master SoftMotion** from the device panel and click **Browse** to select the EtherCAT communication network **eth0**. Select the distributed clock as needed. In this example, select 4000us for the cycle time. See the figure.

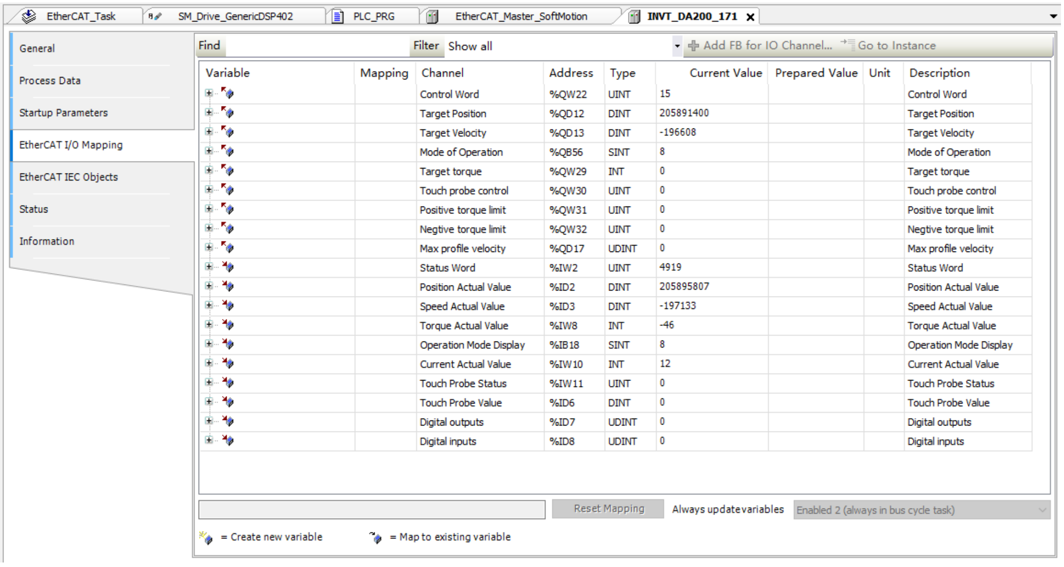


Click the button on the toolbar to compile the code. After compiling, click the button on the toolbar to log in to the controller. The servo starts normally, the motor runs smoothly, and the upper computer interface is shown in the following figure.

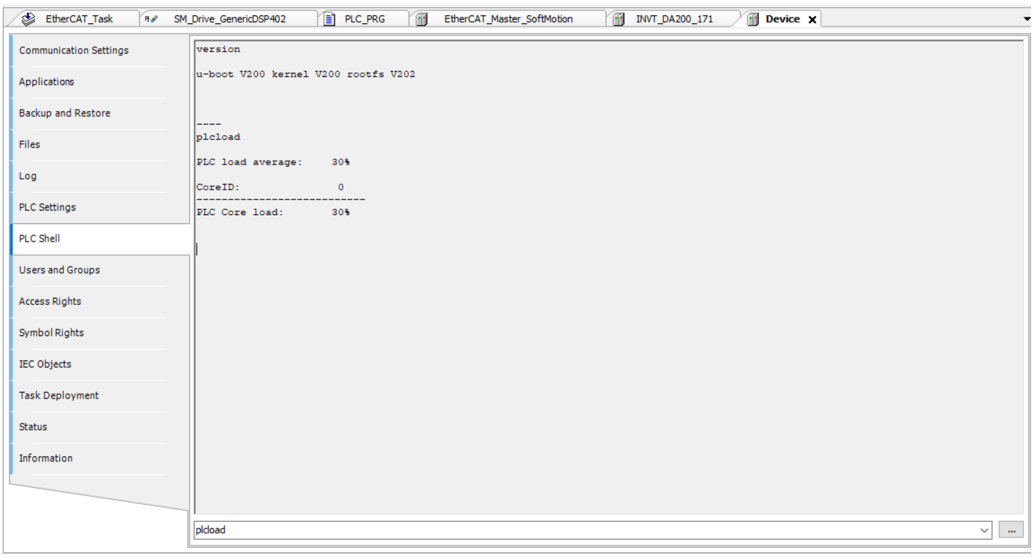




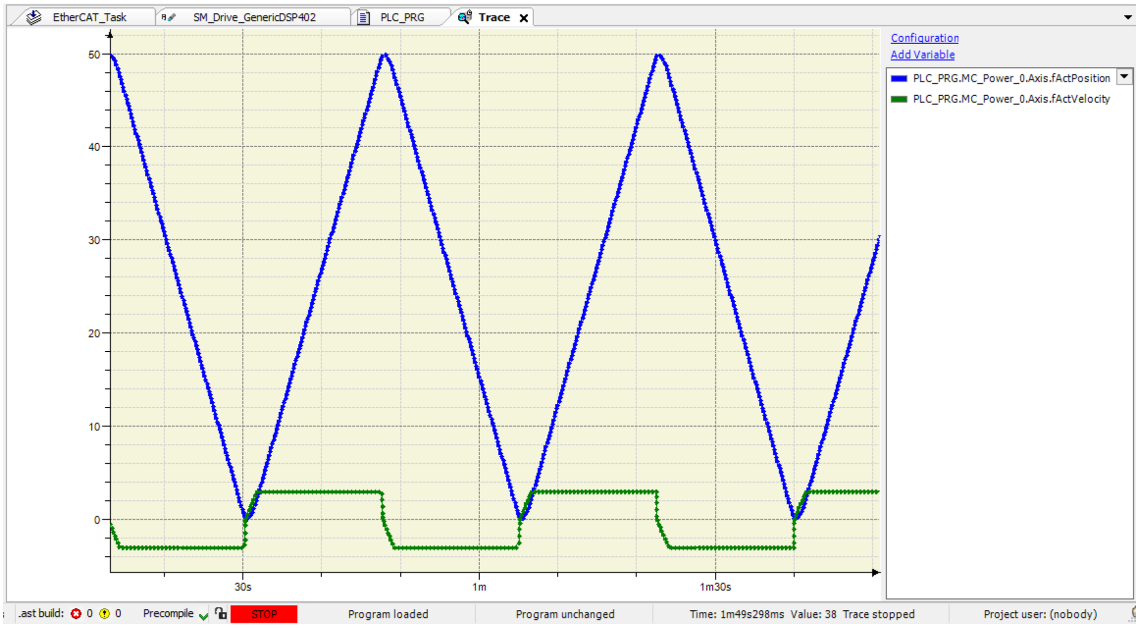
Double-click **INVT_DA200_171** from the device panel to view or set the current motor running parameters in the I/O mapping interface. See the figure.



Select **Device** > **PLCShell**. Click the  button at the bottom right corner and select **prcload**. Then the CPU load rate of the current controller will be shown as follows.



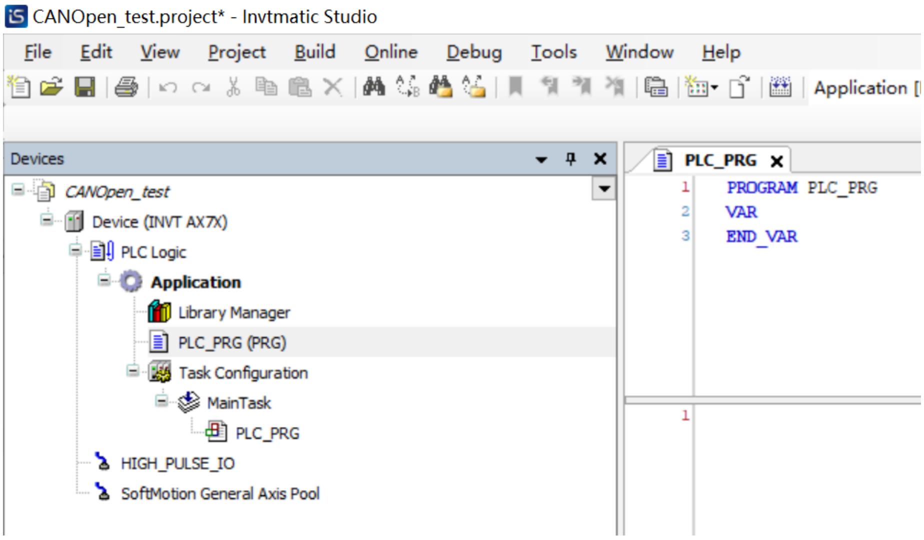
To observe the operation of the motor shaft in an intuitive way and track the actual position of the shaft, create a new trace. Right click **Application** and select **Add Object** > **Trace**. Set the task attribute to **EtherCAT_Task**, and add **PLC_PRG.MC_Power_0.Axis.fActPosition** and **PLC_PRG.MC_Power_0.Axis.fActVelocity** variables in **Trace**. Adjust the display properties of the coordinates appropriately. Right click the graph and select **Download Trace** to track the actual position and actual speed of the motor, as shown in the following figure.



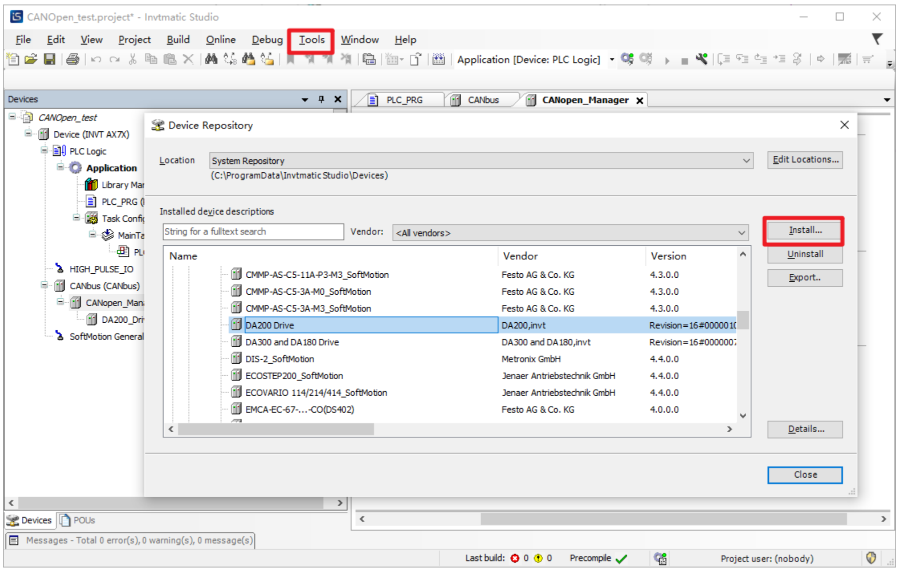
A.3 Controller and DA200 Series Servo CANopen Configuration Example

In this section, we will write a program to connect to the DA200 series servo drive through CANopen communication.

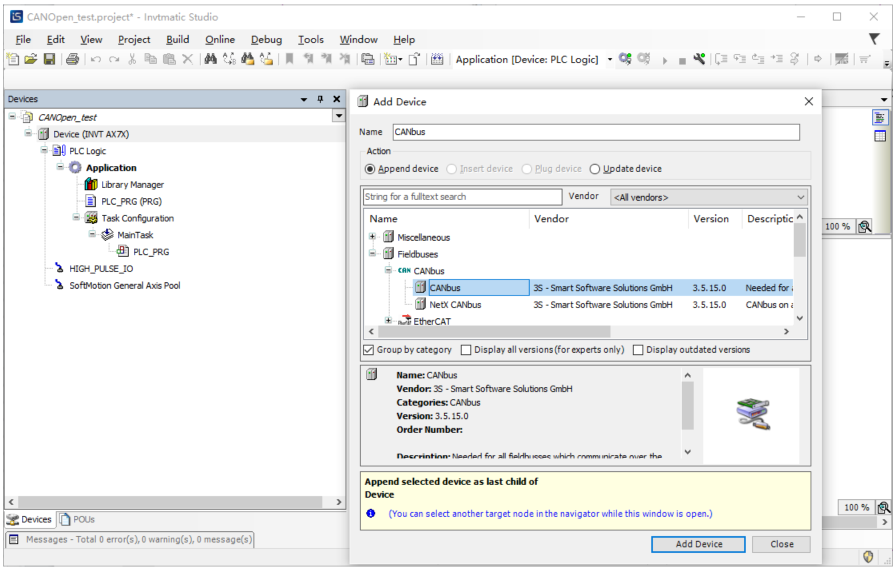
1. Refer to section 2.4 “Project creation” to creat a project, as shown in the following figure.



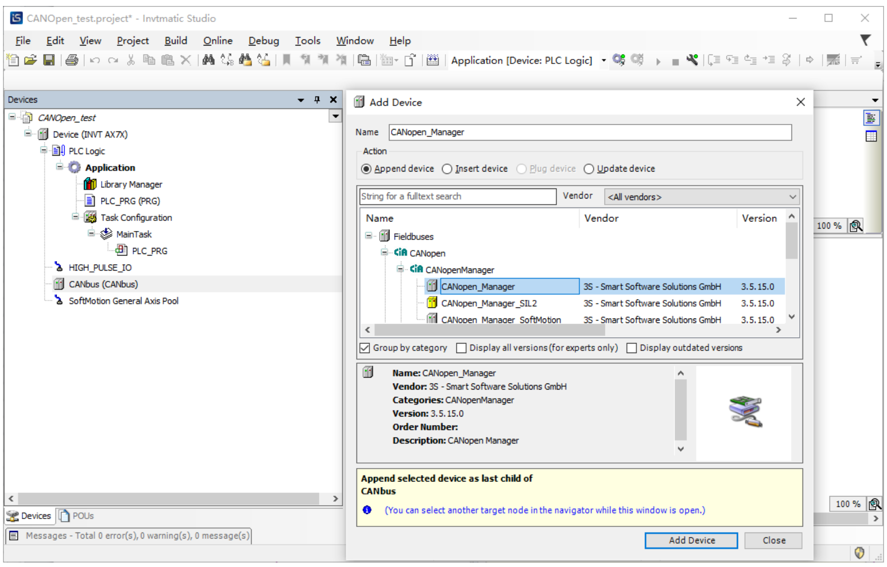
2. In the **Tool** bar, select **Tools** > **Device Repository**, click **Install** to select the device profile **INVT_DA200_CANopen.eds**, and click **Open**. Then the DA200 CANopen device profile can be added successfully.



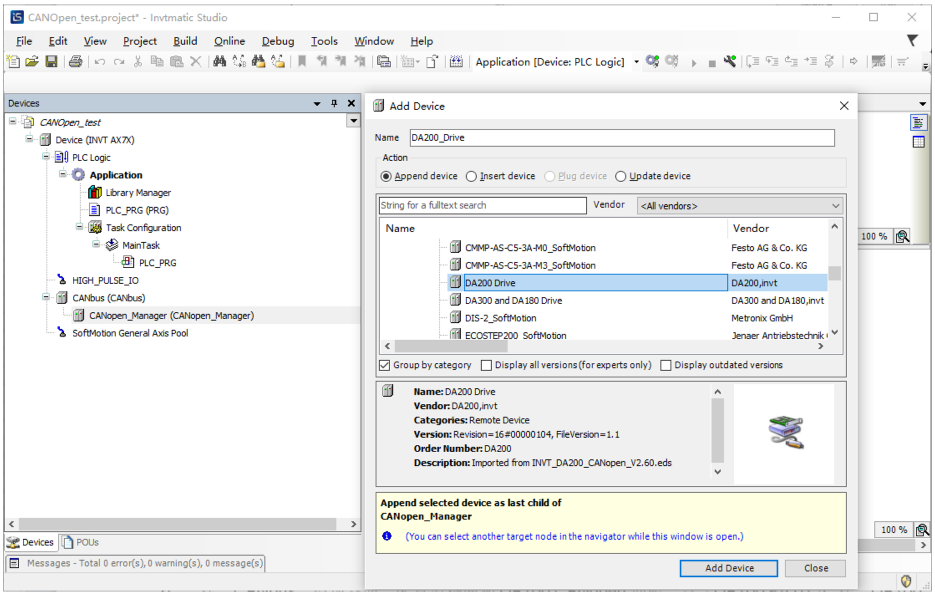
3. Right click **Device** in the device panel, and select **Add Device** > **CANbus**. Click **Add Device,** and **CANbus** is added successfully.



4. Right click **CANbus,** and select **Add Device** > **CANopen_Manager**. Click **Add Device**, and **CANopen_Manager** is added successfully.

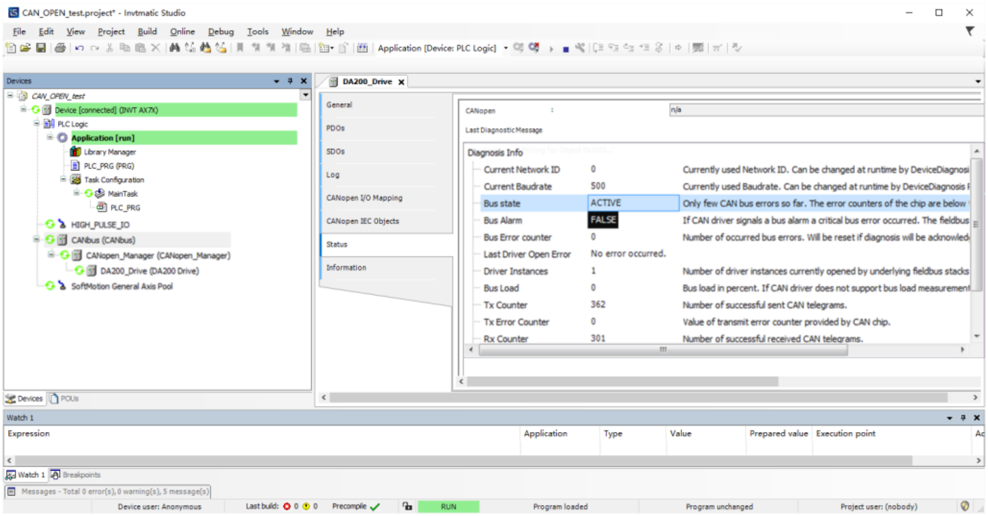


5. Right click **CANopen_Manager** and select **Add Device** > **DA200 Drive**. Click **Add Device**, and **DA200 Drive** is added successfully.



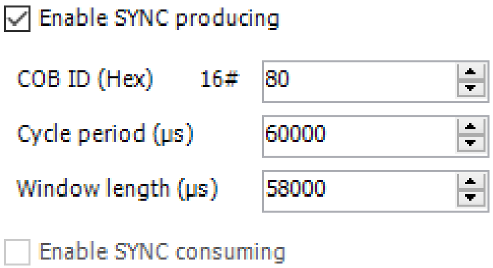
6. In the **CANbus** overview interface, set the baud rate to be the same as that of the DA200 CANopen servo (DA200 P4.02). In the **DA200_Drive** overview interface, set the node ID to be the same as that of the DA200 CANopen servo (DA200 P4.05).

7. After completing the physical connection of the device, download the program, log in the device, and you can see that the CANopen is connected to DA200 communication successfully.

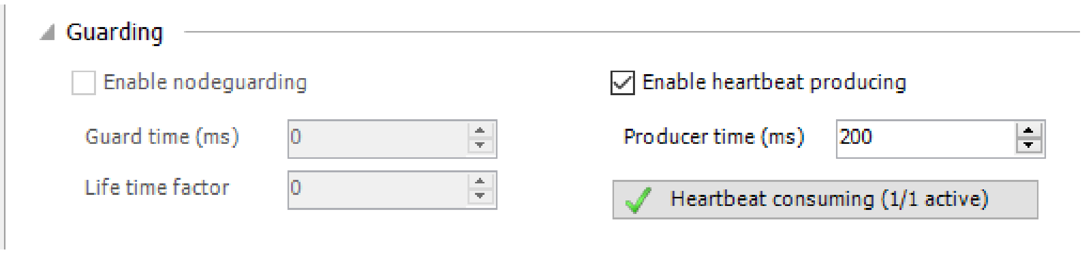


Note:

- If there is a high requirement for real-time data, the CAN bus load shall be less than 30% in order to avoid a small delay in data sending and receiving due to bus competition.
- For CAN buses with synchronization requirements, the set value of the window length in the bus synchronization message is slightly less than the cycle period.



- The task period of the task where CANopen is located shall be slightly longer than the actual execution time of the task.
- To ensure that the master monitors the slave properly, the **Enable heartbeat producing** option in the slave shall be checked.



Appendix B SMC_ERROR description

Error number	Function block	ENUM value	Description
0	All function blocks	SMC_NO_ERROR	No error
1	DriveInterface	SMC_DI_GENERAL_COMMUNICATION_ERROR	Communication error. For example, sercos ring has broken
2	DriveInterface	SMC_DI_AXIS_ERROR	Axis error
10	DriveInterface	SMC_DI_SWLIMITS_EXCEEDED	Position outside of permissible range of SWLimit
11	DriveInterface	SMC_DI_HWLIMITS_EXCEEDED	Hardware end switch is active
13	DriveInterface	SMC_DI_HALT_OR_QUICKSTOP_NOT_SUPPORTED	Drive status Halt or Quickstop is not supported
14	DriveInterface	SMC_DI_VOLTAGE_DISABLED	Drive has no power
15	DriveInterface	SMC_DI_IRREGULAR_ACTPOSITION	Current position given from the drive seems to be irregular. Check the communication.
16	DriveInterface	SMC_DI_POSITIONLAGERROR	Position lag error. Difference between set and current position exceeds the given limit
20	All motion generating function blocks	SMC_REGULATOR_OR_START_NOT_SET	Controller enable not done or brake applied
21	Axis in wrong controller mode	SMC_WRONG_CONTROLLER_MODE	Axis in wrong controller mode
30	DriveInterface	SMC_FB_WASNT_CALLED_DURING_MOTION	Motion creating module has not been called again before end of the motion.
31	All function blocks	SMC_AXIS_IS_NO_AXIS_REF	Type of given AXIS_REF variable is not AXIS_REF
32	Axis in wrong controller mode	SMC_AXIS_REF_CHANGED_DURING_OPERATION	AXIS_REF variable has been exchanged while the module was active.
33	DriveInterface	SMC_FB_ACTIVE_AXIS_DIABLED	Axis disabled while being moved. MC_Power.bRegulatorOn
34	All motion generating function blocks	SMC_AXIS_NOT_READY_FOR_MOTION	Axis in its current state cannot execute a motion command, because the axis doesn't signal currently that it follows the target values
40	VirtualDrive	SMC_VD_MAX_VELOCITY_EXCEEDED	Maximum velocity fMaxVelocity exceeded
41	VirtualDrive	SMC_VD_MAX_ACCELERATION_EXCEEDED	Maximum acceleration fMaxAcceleration exceeded
42	VirtualDrive	SMC_VD_MAX_DECELERATION_EXCEEDED	Maximum deceleration fMaxDeceleration exceeded
50	SMC_Homing	SMC_3SH_INVALID_VELACC_VALUES	Invalid velocity or acceleration values
51	SMC_Homing	SMC_3SH_MODE_NEEDS_HWLIMIT	Mode requests for safety reasons use of end switches

Error number	Function block	ENUM value	Description
70	SMC_SetControllerMode	SMC_SCM_NOT_SUPPORTED	Mode not supported
71	SMC_SetControllerMode	SMC_SCM_AXIS_IN_WRONG_STATE	In current mode, controller mode cannot be changed
75	SMC_SetTorque	SMC_ST_WRONG_CONTROLLER_MODE	Axis not in correct controller mode
80	SMC_ResetAxisGroup	SMC_RAG_ERROR_DURING_STARTUP	Error at startup of the axis group
90	SMC_ChangeGearingRatio	SMC_CGR_ZERO_VALUES	Invalid values
91	SMC_ChangeGearingRatio	SMC_CGR_DRIVE_POWERED	Gearing parameters must not be changed as long as the drive is under control
92	SMC_ChangeGearingRatio	SMC_CGR_INVALID_POSPERIOD	Invalid position period (<= 0)
110	MC_Power	SMC_P_FTASKCYCLE_EMPTY	Axis contain no information on cycle time (fTaskCycle = 0)
120	MC_Reset	SMC_R_NO_ERROR_TO_RESET	Axis without error
121	MC_Reset	SMC_R_DRIVE_DOESNT_ANSWER	Axis does not perform error-reset
122	MC_Reset	SMC_R_ERROR_NOT_RESETTABLE	Error could not be reset
123	MC_Reset	SMC_R_DRIVE_DOESNT_ANSWER_IN_TIME	Communication with the axis did not work
130	MC_ReadParameter, MC_ReadBoolParameter	SMC_RP_PARAM_UNKNOWN	Parameter number unknown
131	MC_ReadParameter, MC_ReadBoolParameter	SMC_RP_REQUESTING_ERROR	Error during transmission to the drives. See error number in function block instance ReadDriveParameter, Library SM_DriveBasic
140	MC_WriteParameter, MC_WriteBoolParameter	SMC_WP_PARAM_INVALID	Parameter number unknown or writing not allowed
141	MC_WriteParameter, MC_WriteBoolParameter	SMC_WP_SENDING_ERROR	See error number in function block instance WriteDriveParameter, Library Drive_Basic
170	MC_Home	SMC_H_AXIS_WASNT_STANDSTILL	Axis has not been in standstill state
171	MC_Home	SMC_H_AXIS_DIDNT_START_HOMING	Error at start of homing action
172	MC_Home	SMC_H_AXIS_DIDNT_ANSWER	Communication error
173	MC_Home	SMC_H_ERROR_WHEN_STOPPING	Error at stop after homing. Deceleration may not be set
180	MC_Stop	SMC_MS_UNKNOWN_STOPPING_ERROR	Unknown error at stop
181	MC_Stop	SMC_MS_INVALID_ACCDEC_VALUES	Invalid velocity or acceleration values

Error number	Function block	ENUM value	Description
182	MC_Stop	SMC_MS_DIRECTION_NOT_APPLICABLE	Direction = shortest not applicable
183	MC_Stop	SMC_MS_AXIS_IN_ERRORSTOP	Axis is in error stop status. Stop cannot be executed
184	MC_Stop	SMC_BLOCKING_MC_STOP_WASNT_CALLED	Instance of MC_Stop blocking the axis by Execute = TRUE has not been called yet. MC_Stop (Execute = FALSE) has to be called.
201	MC_MoveAbsolute	SMC_MA_INVALID_VELACC_VALUES	Invalid velocity or acceleration values
202	MC_MoveAbsolute	SMC_MA_INVALID_DIRECTION	Direction error
226	MC_MoveRelative	SMC_MR_INVALID_VELACC_VALUES	Invalid velocity or acceleration values
227	MC_MoveRelative	SMC_MR_INVALID_DIRECTION	Direction error
251	MC_MoveAdditive	SMC_MAD_INVALID_VELACC_VALUES	Invalid velocity or acceleration values
252	MC_MoveAdditive	SMC_MAD_INVALID_DIRECTION	Direction error
276	MC_MoveSuperImposed	SMC_MSI_INVALID_VELACC_VALUES	Invalid velocity or acceleration values
277	MC_MoveSuperImposed	SMC_MSI_INVALID_DIRECTION	Direction error
301	MC_MoveVelocity	SMC_MV_INVALID_ACCDEC_VALUES	Invalid velocity or acceleration values
302	MC_MoveVelocity	SMC_MV_DIRECTION_NOT_APPLICABLE	Direction = shortest/fastest not applicable
325	MC_PositionProfile	SMC_PP_ARRAYSIZE	Erroneous array size
326	MC_PositionProfile	SMC_PP_STEP0MS	Step time = t#0s
350	MC_VelocityProfile	SMC_VP_ARRAYSIZE	Erroneous array size
351	MC_VelocityProfile	SMC_VP_STEP0MS	Step time = t#0s
375	MC_AccelerationProfile	SMC_AP_ARRAYSIZE	Erroneous array size
376	MC_AccelerationProfile	SMC_AP_STEP0MS	Step time = t#0s
400	MC_TouchProbe	SMC_TP_TRIGGEROCCUPIED	Trigger already active
401	MC_TouchProbe	SMC_TP_COULDNT_SET_WINDOW	DriveInterface does not support the window function
402	MC_TouchProbe	SMC_TP_COMM_ERROR	Communication error
410	MC_AbortTrigger	SMC_AT_TRIGGERNOTOCCUPIED	Trigger already de-allocated
426	SMC_MoveContinuousRelative	SMC_MCR_INVALID_VELACC_VALUES	Invalid velocity or acceleration values
427	SMC_MoveContinuousRelative	SMC_MCR_INVALID_DIRECTION	Invalid direction
451	SMC_MoveContinuousAbsolute	SMC_MCA_INVALID_VELACC_VALUES	Invalid velocity or acceleration values

Error number	Function block	ENUM value	Description
452	SMC_MoveContinuousAbsolute	SMC_MCA_INVALID_DIRECTION	Invalid direction
453	SMC_MoveContinuousAbsolute	SMC_MCA_DIRECTION_NOT_APPLICABLE	Direction= fastest not applicable
600	SMC_CamRegister	SMC_CR_NO_TAPPETS_IN_CAM	CAM does not contain any tappets
601	SMC_CamRegister	SMC_CR_TOO_MANY_TAPPETS	Tappet group ID exceeds MAX_NUM_TAPPETS
602	SMC_CamRegister	SMC_CR_MORE_THAN_32_ACCESSES	More than 32 accesses on one CAM_REF
625	MC_CamIN	SMC_CI_NO_CAM_SELECTED	No cam selected
626	MC_CamIN	SMC_CI_MASTER_OUT_OF_SCALE	Master axis out of valid range
627	MC_CamIN	SMC_CI_RAMPIN_NEEDS_VELACC_VALUES	Velocity and acceleration values must be specified for ramp_in function
628	MC_CamIN	SMC_CI_SCALING_INCORRECT	Scaling variables fEditor/TableMasterMin/Max are not correct
640	SMC_CAMBounds, SMC_CamBounds_Pos	SMC_CB_NOT_IMPLEMENTED	Function block for the given cam format is not implemented
675	MC_GearIn	SMC_GI_RATIO_DENOM	RatioDenominator=0
676	MC_GearIn	SMC_GI_INVALID_ACC	Acceleration invalid
677	MC_GearIn	SMC_GI_INVALID_DEC	Deceleration invalid
725	MC_Phase	SMC_PH_INVALID_VELACCDEC	Velocity and acceleration/deceleration values invalid
726	MC_Phase	SMC_PH_ROTARYAXIS_PERIOD0	Rotation axis with fPositionPeriod = 0
750	All modules using MC_CAM_REF as input	SMC_NO_CAM_REF_TYPE	Type of given cam is not MC_CAM_REF
751	MC_CamTableSelect	SMC_CAM_TABLE_DOES_NOT_COVER_MASTER_SCALE	Master area, xStart and xEnd, from CamTable is not covered by curve data
775	MC_GearInPos	SMC_GIP_MASTER_DIRECTION_CHANGE	During coupling of slave axis, master axis has changed direction of rotation
800	SMC_BacklashCompensation	SMC_BC_BL_TOO_BIG	Gear backlash fBacklash too large (> position periode/2)
1000	CNC function blocks which are supervising the licensing	SMC_NO_LICENSE	Target is not licensed for CNC
1001	SMC_Interpolator	SMC_INT_VEL_ZERO	Path cannot be processed because set velocity = 0
1002	SMC_Interpolator	SMC_INT_NO_STOP_AT_END	Last path object has Vel_End > 0
1003	SMC_Interpolator	SMC_INT_DATA_UNDERRUN	Warning: GEOINFO-List processed in DataIn but end of list not reached.

Error number	Function block	ENUM value	Description
			Reason: EndOfList of the queue in DataIn not be set. SMC_Interpolator faster than path generating function blocks.
1004	SMC_Interpolator	SMC_INT_VEL_NONZERO_AT_STOP	Velocity at Stop > 0
1005	SMC_Interpolator	SMC_INT_TOO_MANY_RECURSIONS	Too much SMC_Interpolator recursions. SoftMotion-Error
1006	SMC_Interpolator	SMC_INT_NO_CHECKVELOCITIES	SMC_CHeckVelocities is not the last processed function block, that accesses to the OutQueue-data by poqDataIn
1007	SMC_Interpolator	SMC_INT_PATH_EXCEEDED	Internal or numeric error
1008	SMC_Interpolator	SMC_INT_VEL_ACC_DEC_ZERO	Velocity and acceleration/ deceleration is null or to low
1009	SMC_Interpolator	SMC_INT_DWIPOTIME_ZERO	FB called with dwIpoTime = 0
1050	SMC_Interpolator2Dir	SMC_INT2DIR_BUFFER_TOO_SMALL	Data buffer too small
1051	SMC_Interpolator2Dir	SMC_INT2DIR_PATH_FITS_NOT_IN_QUEUE	Path does not go completely in queue
1100	SMC_CheckVelocities	SMC_CV_ACC_DEC_VEL_NONPOSITIVE	Velocity and acceleration/ deceleration values non-positive
1120	SMC_Controlaxisbypos	SMC_CA_INVALID_ACCDEC_VALUES	Values of fGapVelocity /fGapAcceleration/fGapDeceleration non-positive
1200	SMC_NCDecoder	SMC_DEC_ACC_TOO_LITTLE	Acceleration value impermissible
1201	SMC_NCDecoder	SMC_DEC_RET_TOO_LITTLE	Deceleration value impermissible
1202	SMC_NCDecoder	SMC_DEC_OUTQUEUE_RAN_EMPTY	Data underrun. Queue has been read and is empty
1203	SMC_NCDecoder	SMC_DEC_JUMP_TO_UNKNOWN_LINE	Jump to line cannot be executed because line number is unknown
1204	SMC_NCDecoder	SMC_DEC_INVALID_SYNTAX	Syntax invalid
1205	SMC_NCDecoder	SMC_DEC_3DMODE_OBJECT_NOT_SUPPORTED	Objects are not supported in 3D mode
1300	SMC_GCodeViewer	SMC_GCV_BUFFER_TOO_SMALL	Buffer too small
1301	SMC_GCodeViewer	SMC_GCV_BUFFER_WRONG_TYPE	Buffer elements have wrong type
1302	SMC_GCodeViewer	SMC_GCV_UNKNOWN_IPO_LINE	Current line of the Interpolator could not be found
1500	All function blocks using SMC_CNC_REF	SMC_NO_CNC_REF_TYPE	Given CNC program is not of type SMC_CNC_REF
1501	All function blocks using SMC_OUTQUEUE	SMC_NO_OUTQUEUE_TYPE	Given OutQueue is not of type SMC_OUTQUEUE
1600	CNC function blocks	SMC_3D_MODE_NOT_SUPPORTED	Function block only works with 2D paths

Error number	Function block	ENUM value	Description
2000	SMC_ReadNCFile	SMC_RNCF_FILE_DOESNT_EXIST	File does not exist
2001	SMC_ReadNCFile	SMC_RNCF_NO_BUFFER	No buffer allocated
2002	SMC_ReadNCFile	SMC_RNCF_BUFFER_TOO_SMALL	Buffer too small
2003	SMC_ReadNCFile	SMC_RNCF_DATA_UNDERRUN	Data underrun. Buffer has been read, is empty
2004	SMC_ReadNCFile	SMC_RNCF_VAR_COULDNT_BE_REPLACED	Placeholder variable could not be replaced
2005	SMC_ReadNCFile	SMC_RNCF_NOT_VARLIST	Input pvl does not point to a SMC_VARLIST object
2050	SMC_ReadNCQueue	SMC_RNCQ_FILE_DOESNT_EXIST	File could not be opened
2051	SMC_ReadNCQueue	SMC_RNCQ_NO_BUFFER	No buffer defined
2052	SMC_ReadNCQueue	SMC_RNCQ_BUFFER_TOO_SMALL	Buffer too small
2053	SMC_ReadNCQueue	SMC_RNCQ_UNEXPECTED_EOF	Unexpected end of file
2100	SMC_AxisDiagnosticLog	SMC_ADL_FILE_CANNOT_BE_OPENED	File could not be opened
2101	SMC_AxisDiagnosticLog	SMC_ADL_BUFFER_OVERRUN	Buffer overrun. WriteToFile must be called more frequently
2200	SMC_ReadCAM	SMC_RCAM_FILE_DOESNT_EXIST	File could not be opened
2201	SMC_ReadCAM	SMC_RCAM_TOO_MUCH_DATA	Saved cam too big
2202	SMC_ReadCAM	SMC_RCAM_WRONG_COMPILE_TYPE	Wrong compilation mode
2203	SMC_ReadCAM	SMC_RCAM_WRONG_VERSION	File has wrong version
2204	SMC_ReadCAM	SMC_RCAM_UNEXPECTED_EOF	Unexpected end of file
3001	SMC_WriteDriveParamsToFile	SMC_WDPF_CHANNEL_OCCUPIED	SMC_WDPF_TIMEOUT_PREPARING_LIST
3002	SMC_WriteDriveParamsToFile	SMC_WDPF_CANNOT_CREATE_FILE	File could not be created
3003	SMC_WriteDriveParamsToFile	SMC_WDPF_ERROR_WHEN_READING_PARAMS	Error at reading the parameters
3004	SMC_WriteDriveParamsToFile	SMC_WDPF_TIMEOUT_PREPARING_LIST	Timeout during preparing the parameter list
5000	SMC_Encoder	SMC_ENC_DENOM_ZERO	Nominator of the conversion factor dwRatioTechUnitsDenom of the Encoder reference is 0
5001	SMC_Encoder	SMC_ENC_AXISUSEDBYOTHERFB	Other module trying to process motion on the Encoder axis
5002	DriveInterface	SMC_ENC_FILTER_DEPTH_INVALID	Filter depth is invalid

Service line:86-755-23535967 E-mail:overseas@invt.com.cn Website:www.invt.com

The products are owned by **Shenzhen INVT Electric Co.,Ltd.**

Two companies are commissioned to manufacture: (For product code, refer to the 2nd/3rd place of S/N on the name plate.)

Shenzhen INVT Electric Co.,Ltd. (origin code: 01)
Address: INVT Guangming Technology Building, Songbai Road, Matian, Guangming District, Shenzhen, China

INVT Power Electronics (Suzhou) Co.,Ltd. (origin code: 06)
Address: No. 1 Kunlun Mountain Road, Science & Technology Town, Gaoxin District, Suzhou, Jiangsu, China

Industrial Automation: ■HMI ■PLC ■VFD ■Servo System ■Elevator Intelligent Control System ■Rail Transit Traction System

Energy & Power: ■UPS ■DCIM ■Solar Inverter ■SVG ■New Energy Vehicle Powertrain System ■New Energy Vehicle Charging System ■New Energy Vehicle Motor

66001-00759

202208 (V1.3)