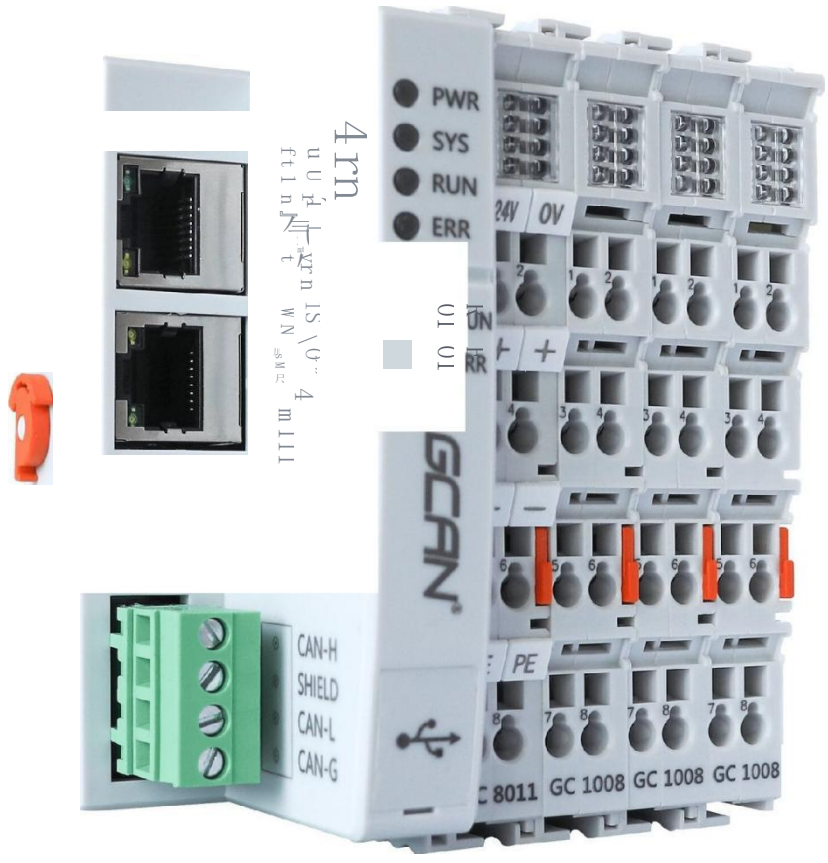
GCAN PLC入门应用指南



Document Name: GCANPLC Getting Started Application Guide					
document name	version number	Version update instructions	change date	changed by	release date
Getting Started with GCANPLC Application Guide	V0.1	Create documentation	2021.09.16	Ponschi	2021.11.05

directory

one, Product Basic Information 1
1. GCAN-PLC Series Product Introduction 1
1.1 Feature Overview 1
1.2 Interface description 1
1.2.1 Ethernet interface Ethernet RJ45 1
1.2.2 serial port RS232/485 RJ45 interface 3
1.2.3 CAN interface 3
1.2.4 Power supply interface 4
1.3 Technical parameters 5
2. GC-IO Module Product Brief 7
2.1 IO Module classification 7
2.2 IO Module selection table 8
3. Programming software (OpenPCS) installation, configuration 9
3.1 OpenPCS install 9
3.2 Install TARGET Component 13
3.3 OpenPCS Online configuration 13
3.4 Hardware connection, download program 16
3.4.1 Device Wiring 16
3.4.2 Revise PLC connected to computer network IP address 17
3.4.3 Download Program 18
two, Introductory Programming Fundamentals 22
1. OpenPCS Function Introduction 22
1.1 interface structure 22
1.2 Basic software functions 23
1.3 New construction 28
1.4 New file 29
1.4.1 New program 29
1.4.2 Create a new declaration file 30
2. Getting Started with Simple Programming Statements 31
2.1 variable declaration 32
2.2 Programming Syntax 33
2.2.1 Math Operators 34
2.2.2 comparison directive 34
2.2.3 Logical Operators 34
2.2.4 IF Statement 34
2.2.5 CASE Statement 35
2.2.6 function block call 36
3. IO Introduction to Module Configuration 36
three, Getting Started with Function Blocks 38
1. Basic Instruction Set 38
3.1.1 CTU 39
3.1.2 CTD 40
3.1.3 CTUD 40
3.1.4 TON 41
3.1.5 DT_CLOCK 42

3.1.6 F_TRIG 42
3.1.7 R_TRIG 43
3.1.8 SR 43
3.1.9 RS 43
3.1.10 PID 44
3.1.11 TOF 45
3.1.12 TP 46
3.1.13 CONCAT_STRING 47
3.1.14 MOD 47
3.2.1 ABS 47
3.2.2 ACOS 47
3.2.3 ADD 48
3.2.4 ASIN 48
3.2.5 ATAN 48
3.2.6 COS 48
3.2.7 DELETE 48
3.2.7 EQ(ST cannot be used in) 49
3.2.9 EXP 49
3.2.10 FIND 49
3.2.11 GE 50
3.2.12 GT 50
3.2.13 INSERT 50
3.2.14 LE 51
3.2.15 LT 51
3.2.16 LEFT 51
3.2.17 LEN 52
3.2.18 LIMIT 52
3.2.19 LN 52
3.2.20 LOG 53
3.2.21 MAX 53
3.2.22 MIN 53
3.2.23 MOD 53
3.2.24 NE 53
3.2.25 POINTER 53
3.2.26 RIGHT 54
3.2.27 ROL 54
3.2.28 ROR 55
3.2.29 SHL 55
3.2.30 SHR 55
3.2.31 SIN 56
3.2.32 SQRT 56
3.2.33 TAN 56
3.2.24 XOR 56
2. Extended function blocks 57
3.3.1 USART_INIT 59
3.3.2 USART_STATE 62
3.3.3 USART_READ_BIN 66
3.3.4 USART_WRITE_BIN 68

3.3.5 USART_READ_CHR....72
3.3.6 USART_WRITE_CHR....73
3.3.7 USART_READ_STR....75
3.3.8 USART_WRITE_STR....77
3.3.9 EXT_USART_INIT....80
3.3.10 EXT_USART_READ_BIN....83
3.3.11 EXT_USART_WRITE_BIN....84
3.3.12 CAN_INIT....87
3.3.13 CAN_MESSAGE_READ8....90
3.3.14 CAN_MESSAGE_WRITE8....92
3.3.15 CAN_NMT....96
3.3.16 CAN_GET_STATE....96
3.3.17 CAN_REGISTER_COBID....97
3.3.18 CAN_PDO_READ8....98
3.3.19 CAN_PDO_WRITE8....100
3.3.20 CAN_SDO_READ8....101
3.3.21 CAN_SDO_WRITE8....102
3.3.22 CAN_SDO_READ_STR....103
3.3.23 CAN_SDO_WRITE_STR....104
3.2.24 CAN_SDO_READ_BIN....106
3.3.25 CAN_SDO_WRITE_BIN....107
3.3.26 CAN_RECV_EMCY....109
3.3.27 CAN_RECV_EMCY_DEV....110
3.3.28 CAN_WRITE_EMCY....110
3.3.29 CAN_ENABLE_CYCLIC_SYNC....111
3.3.30 CAN_SEND_SYNC....112
3.3.31 CAN_ENABLE_CYCLIC_NODE_GUARD....113
3.3.32 CAN_SEND_NODE_GUARD....113
3.3.33 CAN_RECV_BOOTUP_DEV....114
3.3.34 CAN_RECV_BOOTUP....114
3.3.35 CAN_FILTER....115
3.3.36 LAN_INIT....115
3.3.37 LAN_GET_TCPCONNECT_SOCKET....117
3.3.38 LAN_TCP_SERVER_CREATE....118
3.3.39 LAN_TCP_CLIENT_CONNECT....118
3.3.40 LAN_TCP_CLIENT_CLOSE....119
3.3.41 LAN_TCP_RECV_BIN....120
3.3.42 LAN_TCP_SEND_BIN....120
3.3.43 LAN_UDP_CREATE_SOCKET....123
3.3.44 LAN_UDP_CLOSE_SOCKET....124
3.3.45 LAN_UDP_RECVFROM_BIN....124
3.3.46 LAN_UDP_SENDTO_BIN....126
3.3.47 LAN_UDP_RECVFROM_STR....127
3.3.48 LAN_UDP_SENDTO_STR....128
3.3.49 EXT_GPRS_INIT....129
3.3.50 EXT_GPRS_STATUC....130
3.3.51 EXT_GPRS_READ_BIN....130
3.3.52 EXT_GPRS_WRITE_BIN....131

3.3.53 MQTT_INIT....132
3.3.54 MQTT_STATUS....133
3.3.55 MQTT_SEND....133
3.3.56 MODBUS_SLAVE_INIT....134
3.3.57 MODBUS_SLAVE_CTRL....136
3.3.58 MODBUS_MASTER_INIT....138
3.3.59 MODBUS_MASTER_CTRL....139
3.3.60 MODBUS_TCP_SLAVE_INIT....140
3.3.61 MODBUS_TCP_SLAVE_CTRL....141
3.3.62 MODBUS_TCP_MASTER_INIT....144
3.3.63 MODBUS_TCP_MASTER_CTRL....144
3.3.64 DT_CLOCK....145
3.3.65 RTC....148
3.3.66 GETSYSTEMDATEANDTIME....148
3.3.67 SETSYSTEMDATEANDTIME....148
3.3.68 GETDATESTRUCT....149
3.3.69 NVDATA_BIT....150
3.3.70 NVDATA_BIN....151
3.3.71 NVDATA_STR....152
3.3.72 EXT_MOTOR_PWM_INIT....156
3.3.73 EXT_MOTOR_EN....157
3.3.74 CTU....158
3.3.75 CTD....158
3.3.76 CTUD....159
3.3.77 INIT_TBL....160
3.3.78 FIFO_TBL....161
3.3.79 LIFO_TBL....161
3.3.80 ADD_TBL....162
3.3.81 COUNT_TBL....163
3.3.82 PID1....163
3.3.83 SYS_PLC_RESET....165
3.3.84 PTO_PWM....166
3.3.85 PTO....173
3.3.86 GETVARDATA....174
3.3.87 GETVARFLATADDRESS....175
3.3.88 GETTASKINFO....175
Four, Basic Function Application Routine Explanation....177
1. Marquee Experiment Routine Explained....177
2. I/O Experiment....178
3. CAN Send and receive data experiment ST....178
4. serial port RS232 , 485 experiment ST....180
5. TCP SERVER Communication Lab....180

受　控

1. Basic product information

1. GCAN-PLC Series product introduction

1.1 Functional Overview

GCAN-PLC series products （ GCAN-PLC-400 ， GCAN-PLC-510 ， GCAN-PLC-511 ， etc.） are programmable logic controllers （ PLCs ） integrated with bus control functions . It not only has the characteristics of simple appearance and high cost performance, but also can be easily connected to the CAN bus system and Modbus system, etc., and can be expanded through the module insert type.

GCAN-PLC series products consist of a programmable main control module （ GCAN-PLC-510 ， etc.） ， several GC series IO modules and a terminal terminal module （ GC-0001 ） . GCAN-PLC series modules can be connected to all GC series IO modules. Users can choose to expand IO modules according to the actual needs of the site. In the case of sufficient power supply, the number of IO modules can be expanded to 32 at most. GCAN-PLC series can be inserted according to the The front and rear positions of the IO module are automatically assigned hardware addresses to realize automatic configuration. Users do not need to create a configuration interface and set parameters on the PC . The detailed description of the hardware configuration will be shown in Chapter 2, Section 3 of this book, and will not be explained in detail in this chapter .

GCAN-PLC series products can be programmed with OpenPCS software, which supports five standard programming languages specified in the IEC-61131-3 standard, which makes the program highly portable and reusable, and the software also With a variety of debugging functions (breakpoints, single-step, etc.), it is more convenient to debug programs.

GCAN-PLC - 510 can not only complete various digital / analog input / output, but also integrates a variety of common industrial field bus interfaces , such as CANbus, RS232/485 bus, Ethernet bus, and supports common Communication protocols such as: CANopen , Modbus RTU , Modbus TCP , etc.

GC series IO modules include: switch input and output modules, analog input and output modules, pulse input and output modules, communication port expansion modules (RS232/485 , 4G , Bluetooth, WiFi , Zigbee , GPRS , etc.) detailed selection list See Section 2 of this chapter .

1.2 Interface description

1.2.1 Ethernet interfaceE thernet RJ45

The Ethernet interface can be programmed through OpenPCS , and can be used as TCP SERVER , TCP CLIENT , UDP , modbus TCP . In the delivery state, Ethernet can be used to connect with OpenPCS and download programs.

受　控

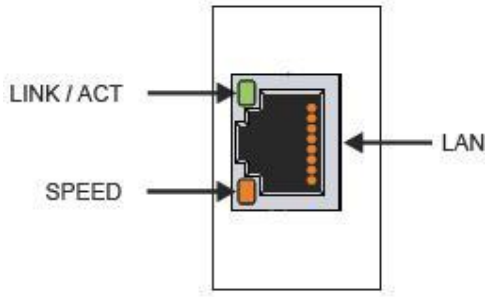


Figure 1.1： Ethernet Interface

The speed of the Ethernet interface is all 100 Mbit. An LED on the left side of the interface indicates the connection status. The upper LED（ LINK / ACT） indicates whether the interface is connected to the network.

If connected to the network, the LED is green.

The lower LED（ SPEED） is yellow. Speed blinks when data transfer is in progress .

Figure 1.2： Ethernet Interface, Pin Numbering

pin	Signal	describe
1	T2+	Pair 2
2	T2 -	
3	T3+	Pair 3
4	T1+	Pair 1
5	T1 -	
6	T3 -	Pair 3
7	T4+	Pair 4
8	T4 -	

Table **1.1**： Ethernet Interface Pin Assignments

1.2.2 Serial RS232/485 RJ45 interface

RS232 communication is RJ45, which can be programmed through OpenPCS. As RS232 communication, modbus RTU communication is used. We come with a free RJ45 to DB9 cable, the specific pins are shown in the table below.

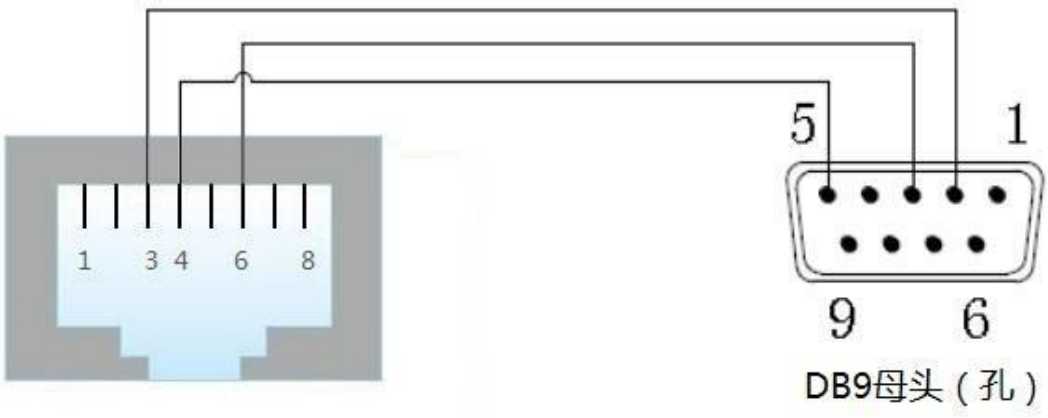


Figure 1.3 pin definition of the patch cord

terminal	RJ45 terminal serial number	DB9 terminal serial number	meaning
RS232_TX	3	2	RS232 data transmission
RS232_RX	6	3	RS232 data reception
GND	4	5	signal ground
RS485_A+	8	7	RS485 signal A+
RS485_B-	1	8	RS485 signal B-

Table 1.2： Serial RS232/485 RJ45 interface pin assignment

1.2.3 CAN interface

CAN communication is the OPEN terminal, which can be programmed through OpenPCS. As CAN communication, CANOPEN communication is used. The specific pins are shown in the figure below.

Figure 1.4 CAN pin assignment

1.2.4 Power supply interface

The power terminals require an external voltage source that provides 24V DC (-15 % /+20 %). The power terminals must provide 1A at 24V to ensure that the PLC will function properly in all conditions.

The cable connection of the PLC in the control cabinet must comply with the standard EN 60204-1： 2006 PELV = extra low voltage protection:

- Basic CPU " PE ″ and ″ 0 ″ of the module's voltage source The V ″ conductors must be at the same potential (connected in the control cabinet).

- Standard EN 60204-1： 2006 6.4.1： b _Section states that one side of a circuit or the energy source of that circuit must be connected to a protective earth system.

- PLC Separate power supply from the rear expansion module when necessary.

- The voltage of the power supply for the expansion module is 24V DC (-15 % / + 20 %), and the minimum power provided by this power supply is calculated according to the current consumption of the expansion module.

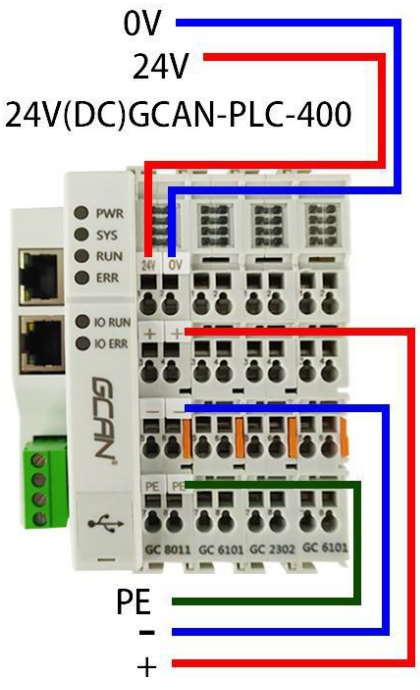


Figure 1.5 Schematic diagram of power supply wiring

serial number	describe
1	The spring loaded terminals labeled ″ 24 V ″ and ″ 0 V " power the PLC .
2	The spring terminals identify ″ + ″, ″ - " and supply power to the IO bus terminals via the power contacts .

Table 1.3 Power supply terminal pin assignment

1.3 technical parameter

	GCAN-PLC-510
Size (L * W * H)	47mm × 68mm × 97mm
weight	180g

surface 1.4 Technical data, dimensions and weights

technical parameter	GCAN-PLC-510
processor	ARM Cortex-M series
Flash (Program Memory)	32M bytes
SRAM (variable memory)	16M bytes
user data store	2k bytes
Power supply	24 V DC (-15 %/+20 %)
Maximum energy consumption	12W

CAN/485 isolation class	1500V

PC software	OpenPCS
Diagnostic Lights	One power light, one system light, one running light, one fault light, one IO bus light, one IO bus fault light
clock	Internal battery backed clock for time and date
Certification	CE

Table 1.5 technical data, general data

technical parameter	describe
Native I/O	No, need to expand GC series IO module
Configuration method	via IO bus
Number of expansion terminal modules	32 _
Digital I/O Signals	Supports up to 32 × 8 input / output
Analog I/O signals	Supports up to 32 × 4 inputs, 32 × 2 outputs

Table 1.6 technical data, I / O terminal

technical parameter	describe
Operating temperature	-40 ° C to +85 ° C
Working humidity	95%RH, no condensation

Vibration resistance	10 frequency sweeps on 3 axes 6 Hz < f < 58.1 Hz displacement 0.15 mm, constant amplitude

	58.1 Hz < f < 500 Hz acceleration 5 g, constant amplitude Complies with EN 60068-2-6
impact resistance	1000 shocks in each direction on 3 axes 15g, 11ms _ Complies with EN 60068-2-27

Table 1.7 Technical data, environmental conditions

Supported Communications	Interface form
CAN,CANOPEN	OPEN4 terminal
RS232, RS485, MODBUS RTU	RJ45
TCP,UDP,MODBUS TCP	RJ45

Table 1.8 Technical data, interface

2. GC-IO Module Product Introduction

2.1 IO Module classification

Digital Input Modules: GC-1008, GC-1018, GC-1502

Digital output modules: GC-2008, GC-2018, GC-2204, GC-2302

Analog Input Modules: GC-3604, GC-3624, GC-3644, GC-3654, GC-3664,

GC-3674 Temperature Acquisition Input Modules: GC-3804, GC-3822,

GC-3844, GC-3854, GC-3864

Analog output module: GC-4602, GC-4622, GC-4642, GC-4652, GC-4662, GC-4672,

GC-4674 Communication expansion module: GC-6101, GC-6221, GC-6501

2.2 IO Module selection table

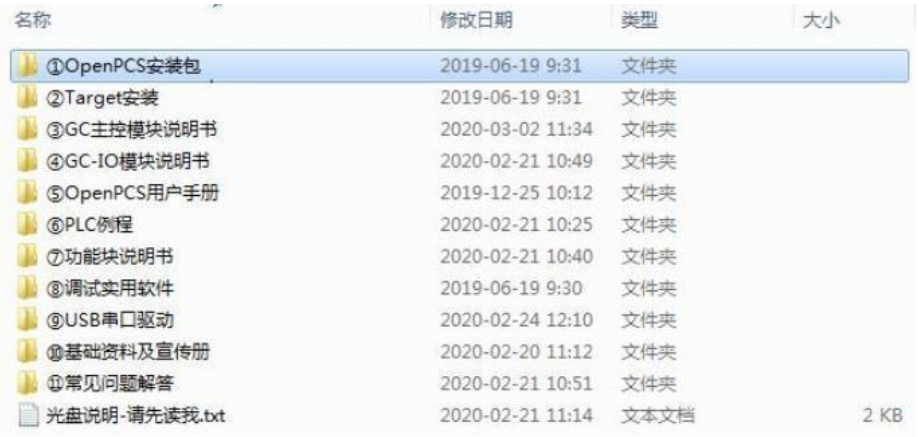
type	model	characteristic	Signal	number of channels
digital input	GC-1008	basic digital	24V DC	8 channels
	GC-1502	Counter (200kHz max)	-	2 channels
digital output	GC-2008	basic digital	24V DC	8 channels
	GC-2204	relay on	-	4 channels
	GC-2302	PWM (20Hz~200kHz)	-	2 channels
Analog input	GC-3604	Voltage input, 16 bit	-5V~+5V	4 channels
	GC-3624	Voltage input, 16 bit	-10V~+10V	4 channels
	GC-3644	Current input, 16 bit	0-20mA	4 channels
	GC-3654	Current input, 16 bit	4-20mA	4 channels
	GC-3664	Voltage input, 16 bit	0~+5V	4 channels
	GC-3674	Voltage input, 16 bit	0~+10V	4 channels
	GC-3804	2-wire PT100, 16 bit	Thermal resistance	4 channels
	GC-3822	3-wire PT100, 16 bit	Thermal resistance	2 channels
	GC-3844/3854/3864	Type K/Type S/Type T Thermocouples	Thermocouple	4 channels
	GC-4602	Voltage output, 16 bit	-5V~+5V	2 channels

Analog output	GC-4622	Voltage output, 16 bit	-10V~+10V	2 channels
	GC-4642	Current output, 16 bit	0-20mA	2 channels
	GC-4652	Current output, 16 bit	4-20mA	2 channels
	GC-4662	Voltage output, 16 bit	0~5V	2 channels
	GC-4672	Voltage output, 16 bit	0~10V	2 channels
Special Expansion Module	GC-6101	RS232/RS485 expansion	-	-
	GC-6201	GPRS extension	-	-
	GC-6221	4G extension	-	-
	GC-6501	WiFi extension	-	-

3. Programming software （ OpenPCS ） installation and configuration

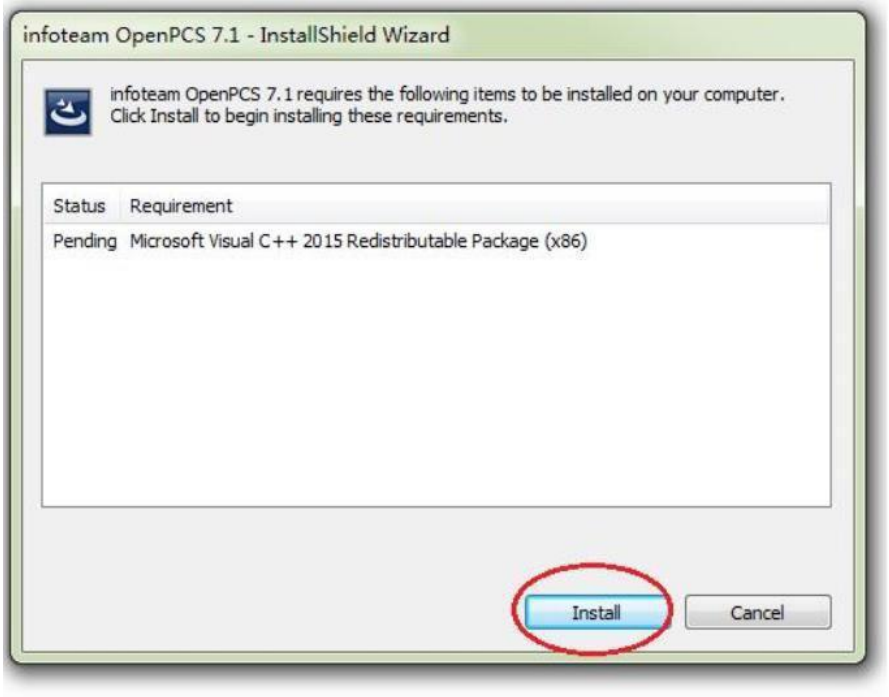
3.1 OpenPCS Install

（1） Open the folder ① in the network disk data, OpenPCS Installation package

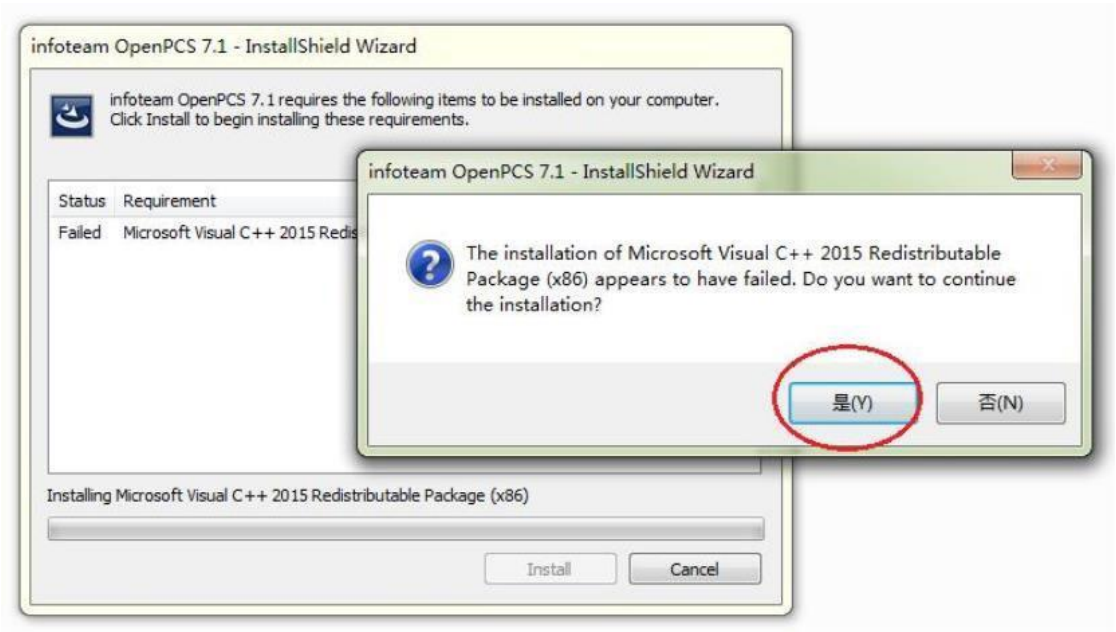


（2） Select the English version or the Chinese version to decompress and double-click the .exe File installation, this article only introduces the English version, the Chinese version installation process is the same as the English version.

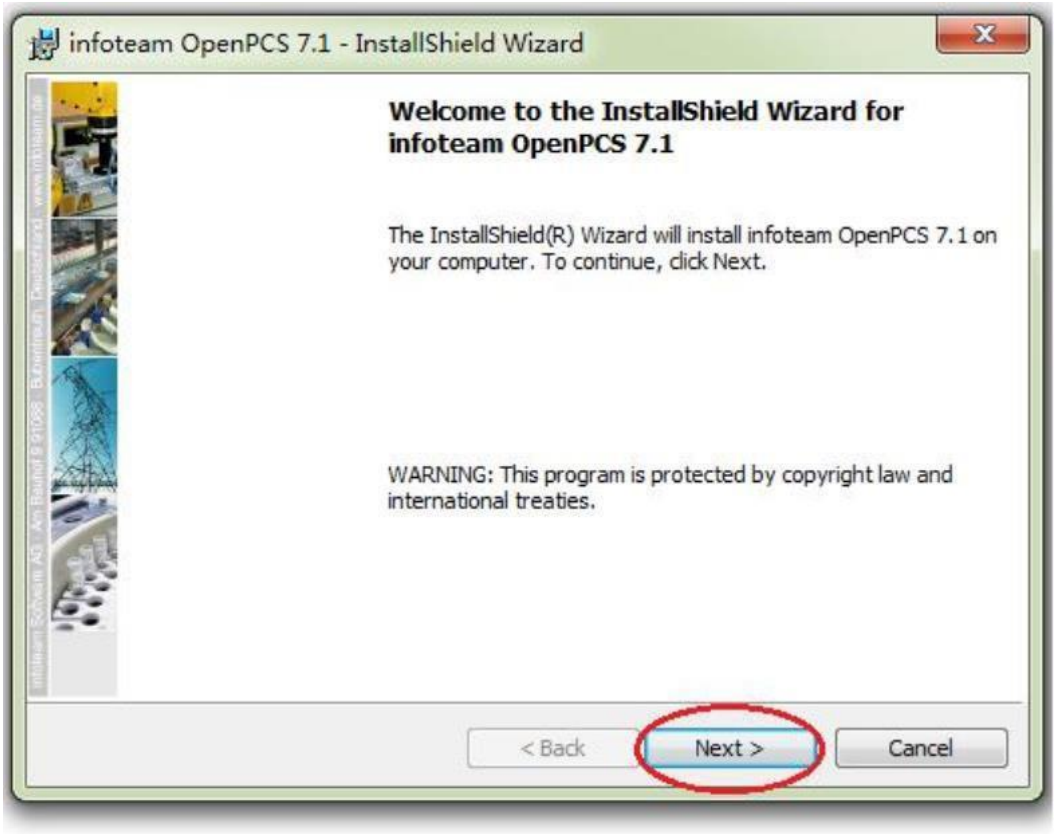
（3） The installation interface appears, click Install



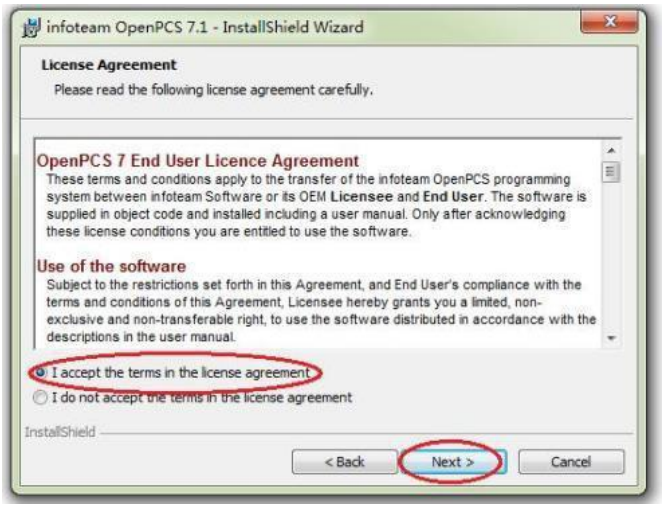
（4）The following prompts may appear during the installation process, click Yes（Y）



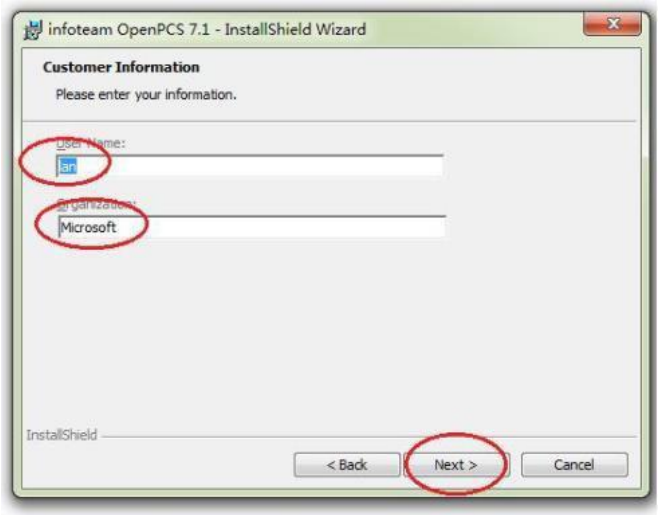
（5）Click Next to continue the installation



（6）Choose to accept the license and click Next

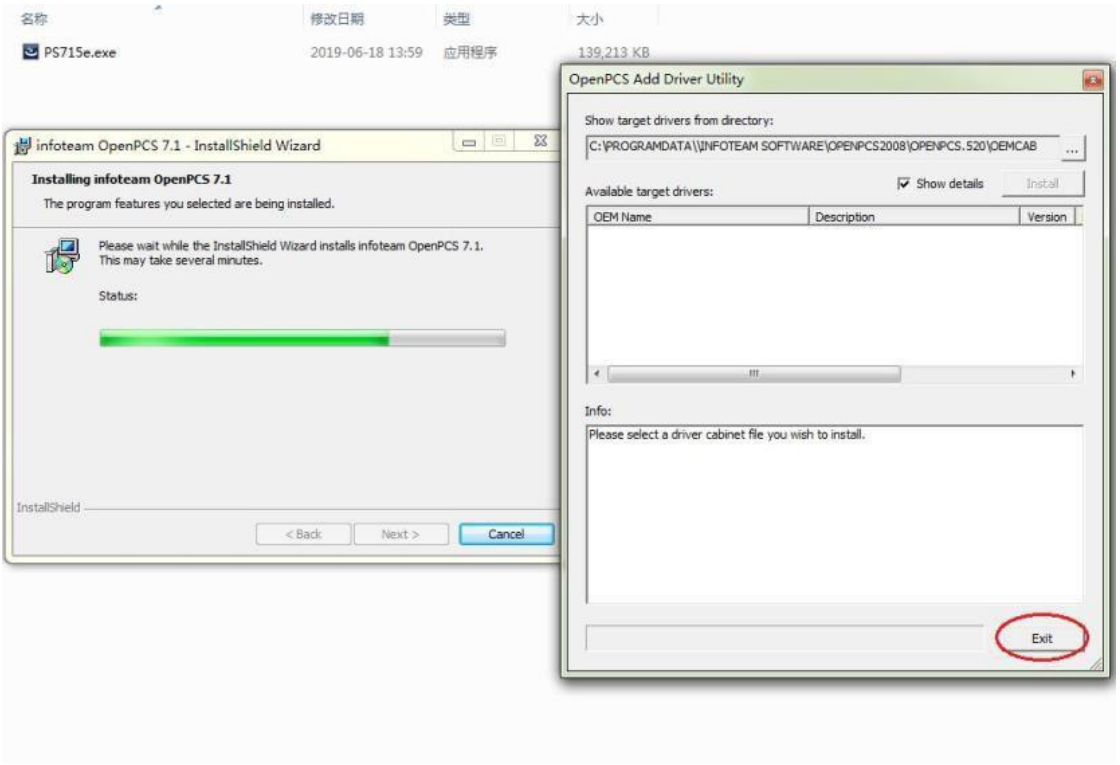


（7）Fill in the user name, here it is recommended to use the default and click Next

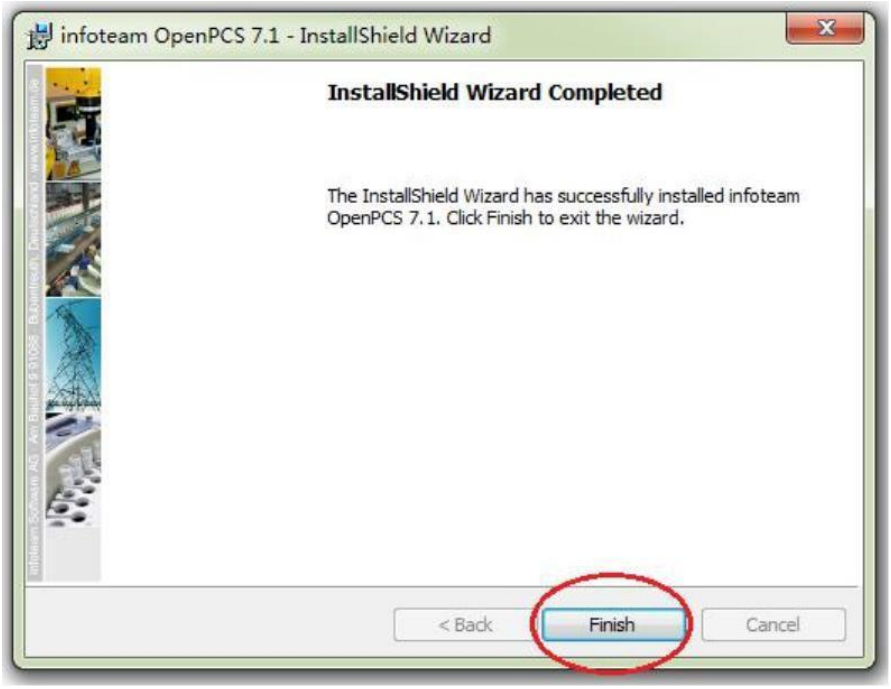


（8）Select the installation path, select it and click Next

(9) For the next steps, just click Next, start the installation, and wait for the installation progress bar until the Add Driver appears, click here to exit.



(10) The installation is complete

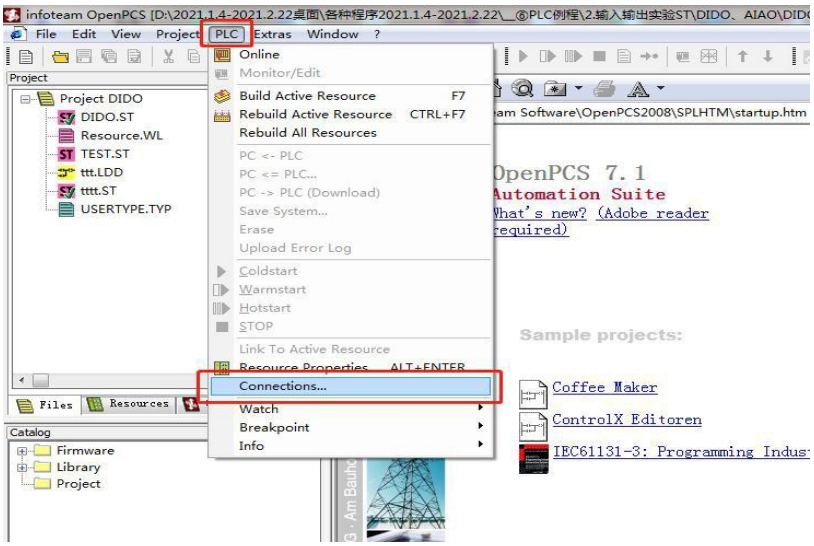


3.2 Install TARGET components

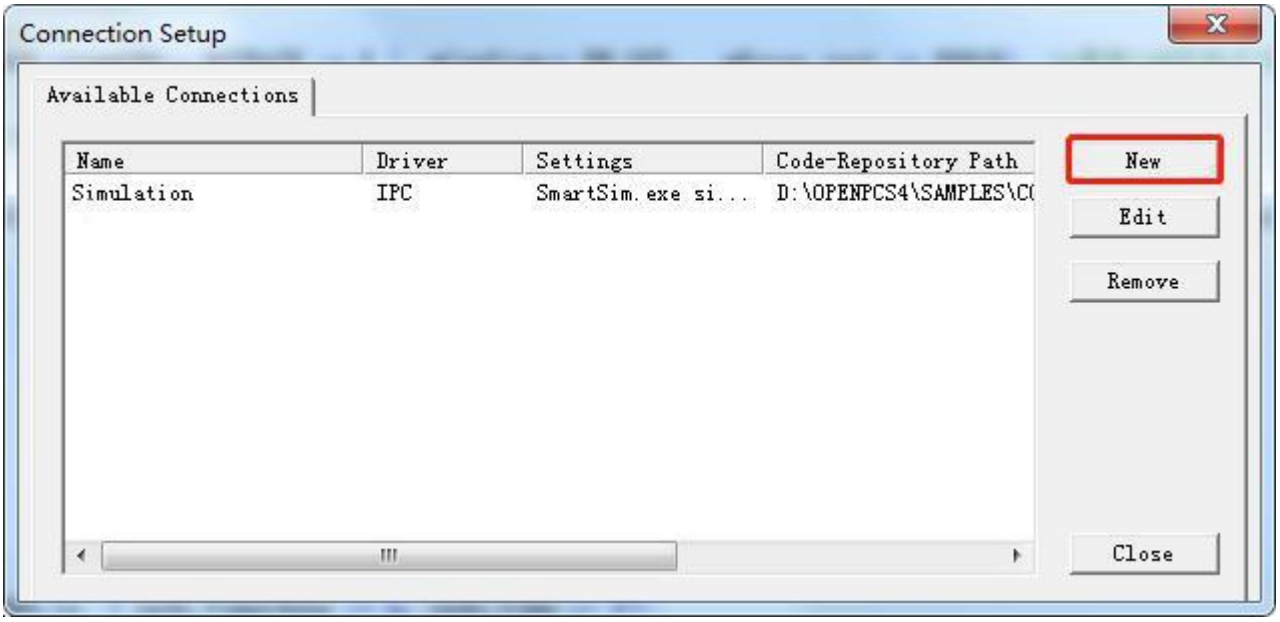
Open the folder ② in the network disk data, TARGET installation, choose the Chinese version and the English version, there is no difference, just choose the language of the installation wizard, select the .exe file inside, double-click, and click Install, the installation process only takes a few seconds , almost in a flash.

3.3 OpenPCS Online configuration

① Select PLC connection of drop-down list Go to connection settings.

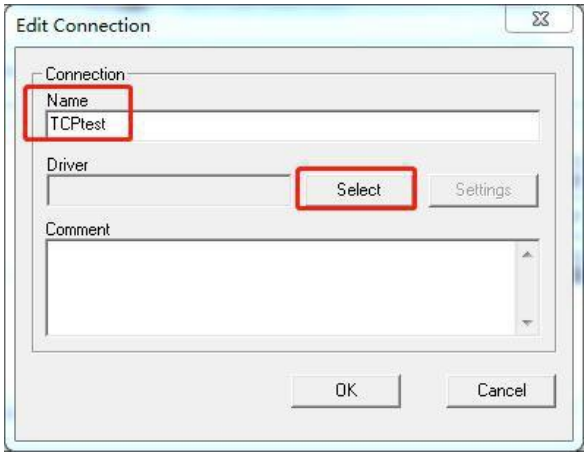


② Click **New** Create a new connection configuration.

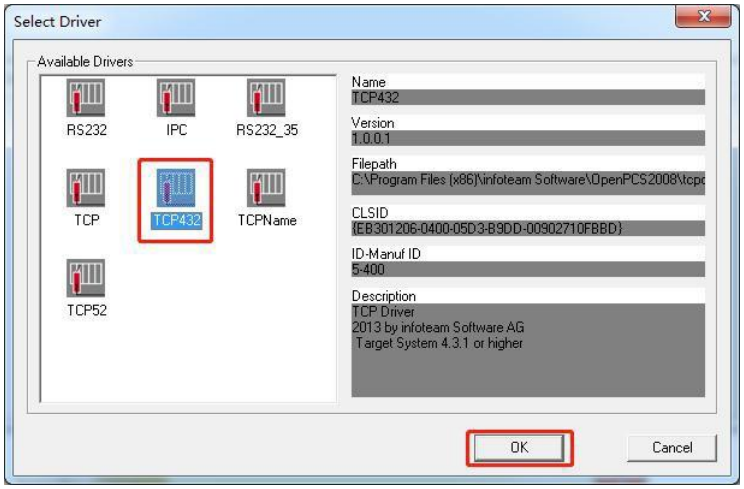


③ Name the new connection configuration, and name it according to user preferences. In this article, it is named **TCPtest**. After filling in the name, click **Select**.

Select the driver for the connection configuration.



④ **TCP432** in the pop-up driver selection box drive, and click **OK**.



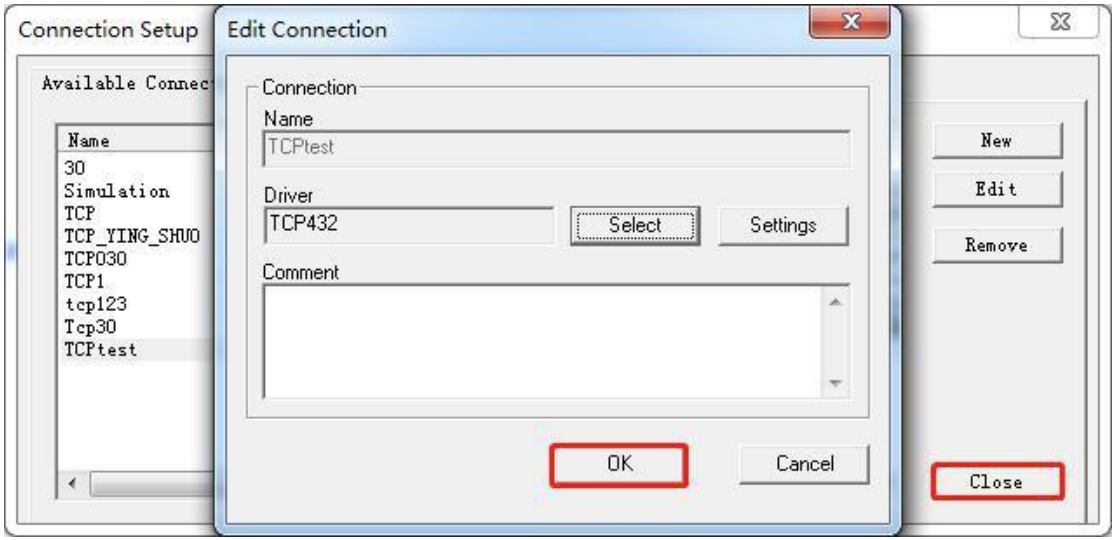
⑤ After selecting the driver, **Settings** The button will become clickable, click **Settings** Button is **TCP432** The driver sets properties.

⑥ Set the port number **Port** is 23042 **（ * port number is a fixed value and cannot be changed * ）** , **IP** Address is **PLC** device **IP** address

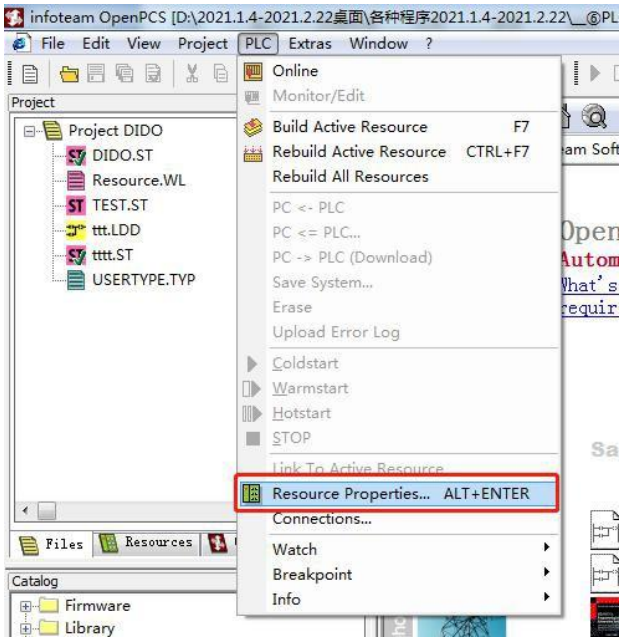
（ * The factory default IP of GCANPLC is 192.168.1.30 , when the user just got the device, the device IP is the default IP* ） Note: Please do not check the following **PLC using big endian**

mode . After setting, click the **OK** button on the right.

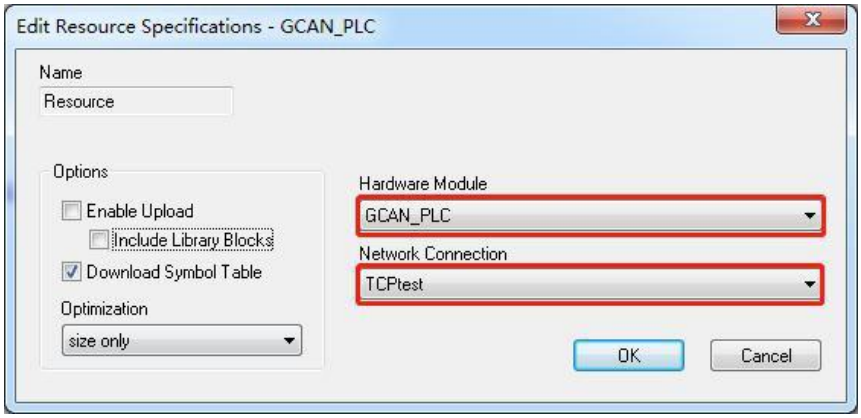
⑦ After editing the connection settings, click **OK** , click **OK** After that, the edit box disappears, click Close in the connection settings.



⑧ Click on the menu bar PLC Resource property Resource in drop down list Properties.



⑨ hardware module selects GCANPLC, and the network connection selects the one just set. In this article, it is TCPtest. The user can select the connection configuration set



by himself according to the actual situation.

3.4 Hardware connection, download program

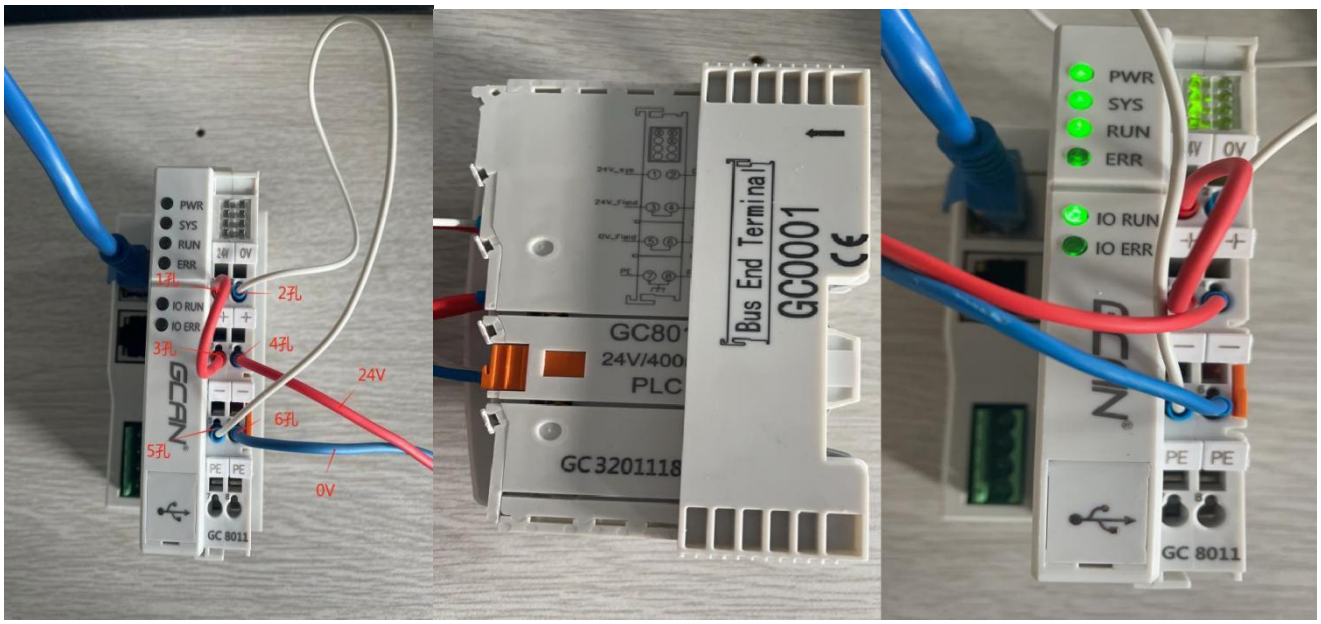
3.4.1 Equipment wiring

(1) The 24V switching power supply supplies power to the PLC. The PLC needs two groups of power supplies. The first group is connected to 24V through hole 1, and connected to 0V through hole 2 to supply power to the controller; 6 holes are connected to 0V to supply power to the IO module connected later, among which holes 3 and 4 are short-circuited inside, and

holes 5 and 6 are short-circuited inside.

So when wiring, connect 1 hole with 3 holes, 2 holes with 5 holes, 4 holes with 24V power supply , 6 holes with power supply 0V ,

You can use one power supply to supply power to the controller and the IO module at the same time. Note: GC0001 should be installed after the last IO module. If there is no IO module, it should be directly installed on the controller to form a loop, otherwise the IO ERR red light will be always on.

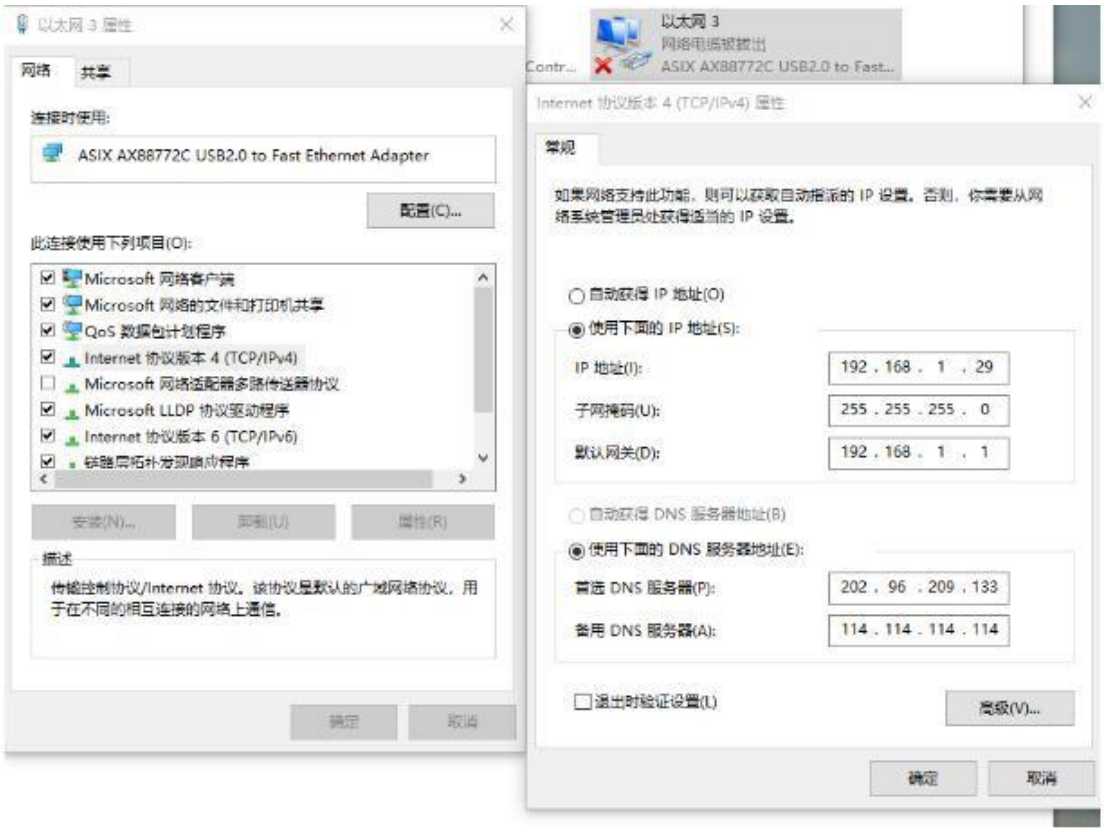


（2）Use a network cable to connect the computer and the PLC. The IP address of the PLC is 192.168.1.30. The IP address of the computer needs to be modified to be the same as the first three IP addresses, namely 192.168.1.XX, where XX is other than 30, which is After the PLC is powered on, the indicator PWR is always on, SYS is flashing slowly, RUN is flashing quickly, IO RUN is flashing quickly, and ERR and IO ERR are off.

3.4.2 Modify PLC IP connected to the computer network address

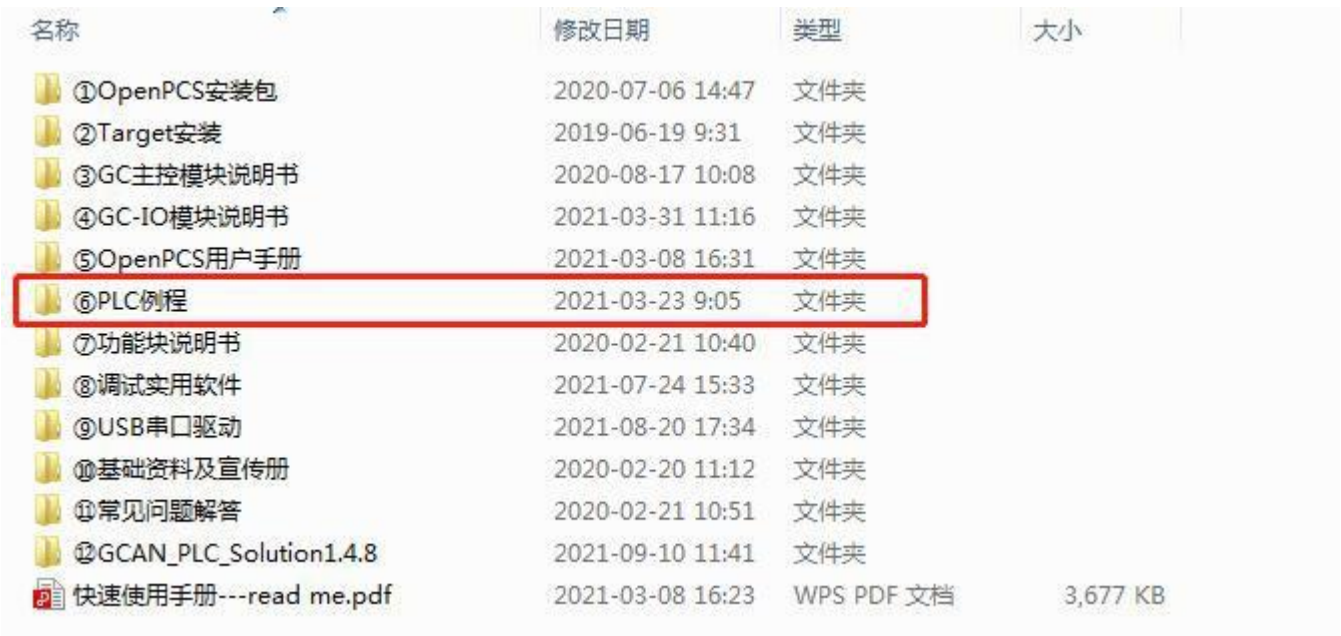
① Open Network and Internet Settings, select Ethernet option

② Click on the right to change the adapter option, right-click the network connection between the PLC and the computer, click Properties, select Internet Protocol Version 4 (TCP4/IPv4), select it and click Properties, and set the IP to the same network segment as the PLC , as shown in the figure.



3.4.3 Downloader

① Open the ⑥ folder PLC in the network disk file routine.

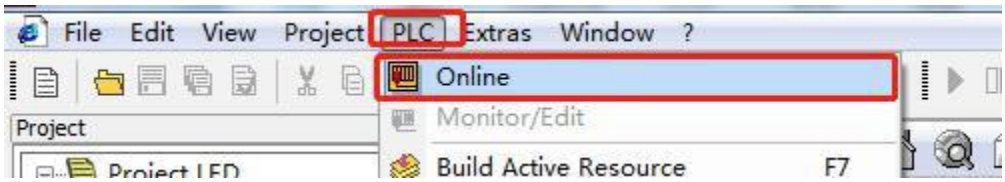


② LED in the folder folder.

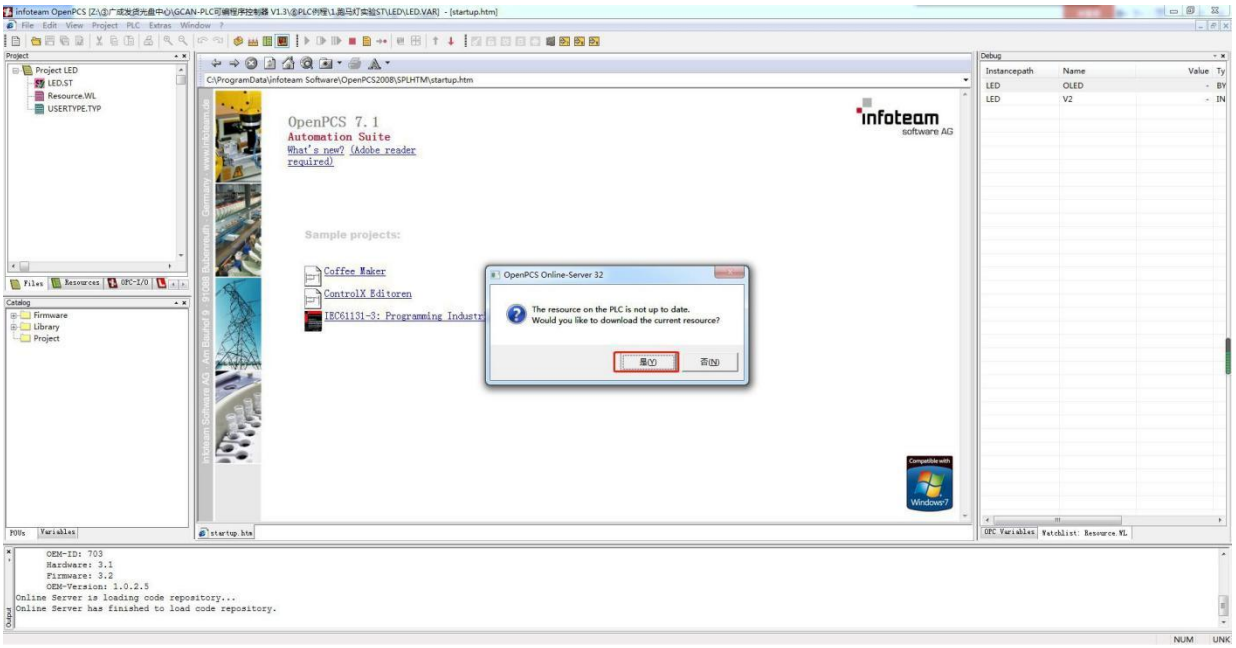
③ Double click on .VAR program file, open program

(**** Note: After opening the program, be sure to check the OpenPCS connection configuration! This is very important! For the check and setting steps, refer to the OpenPCS connection configuration in Chapter 1, Section 3.3 in this article !****)

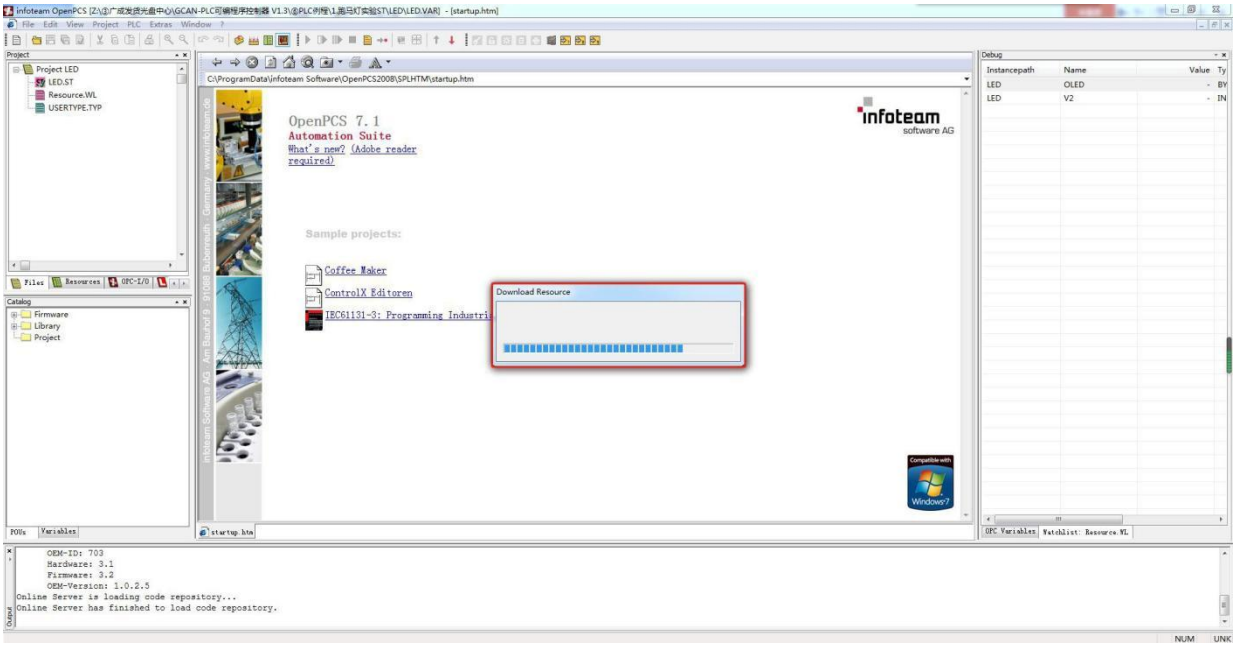
④ After opening the program, click on the menu bar PLC Online in drop down list



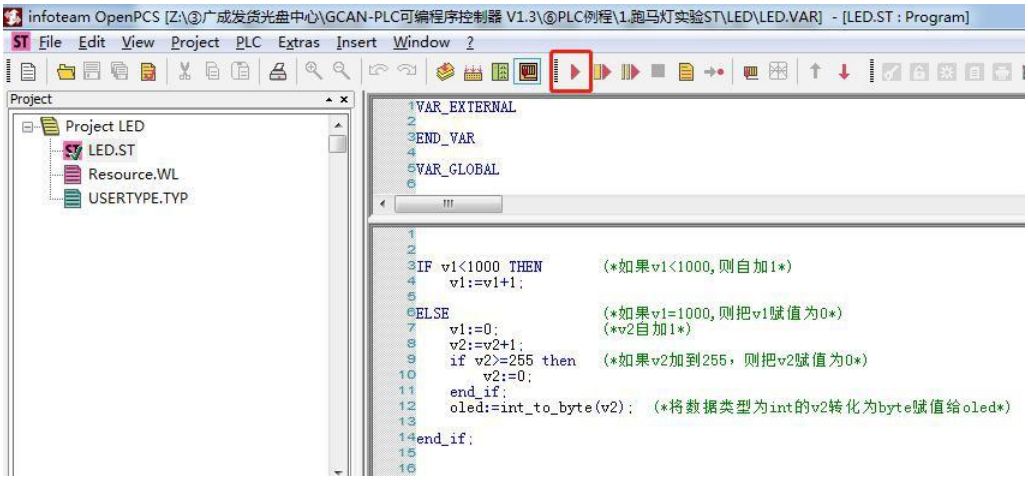
⑤ Before downloading the program, it will pop up: PLC If the program is not the latest, if you want to download the current program, click Yes.



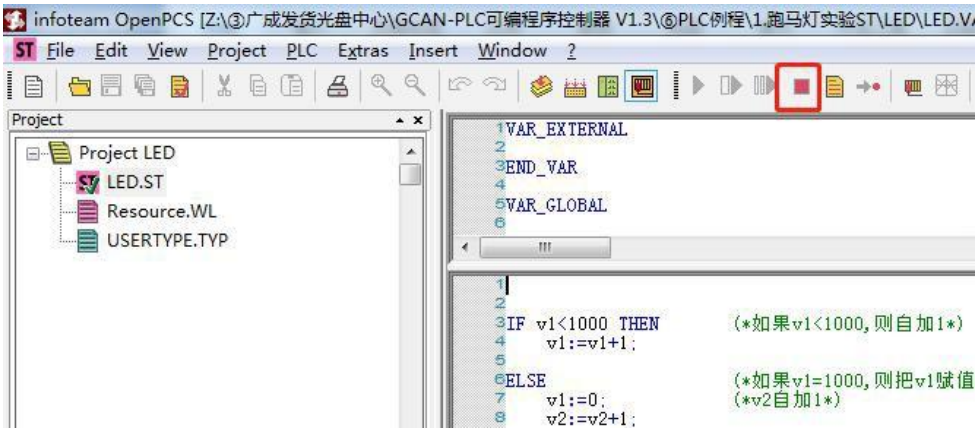
⑥ Wait for the download progress bar.



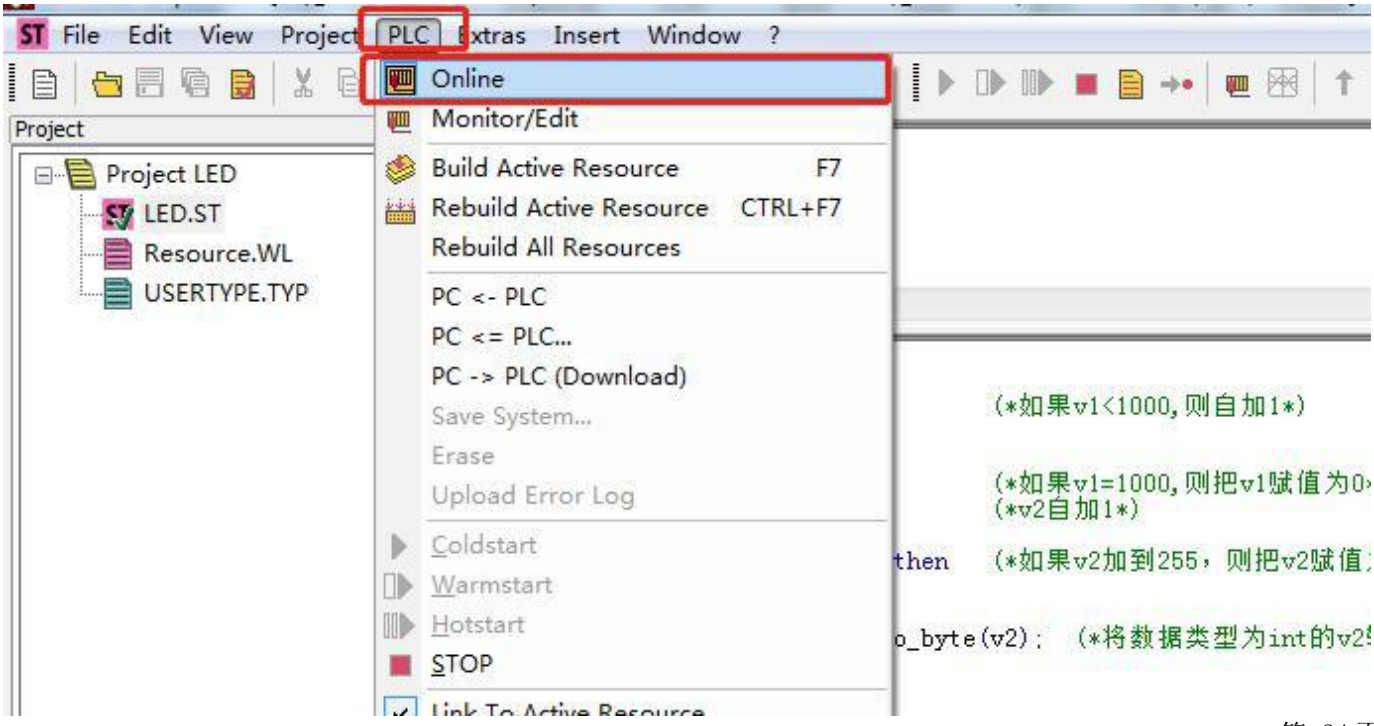
⑦ After the download is complete, click the Run button to run the program.



⑧ Click the stop button to stop the program from running.



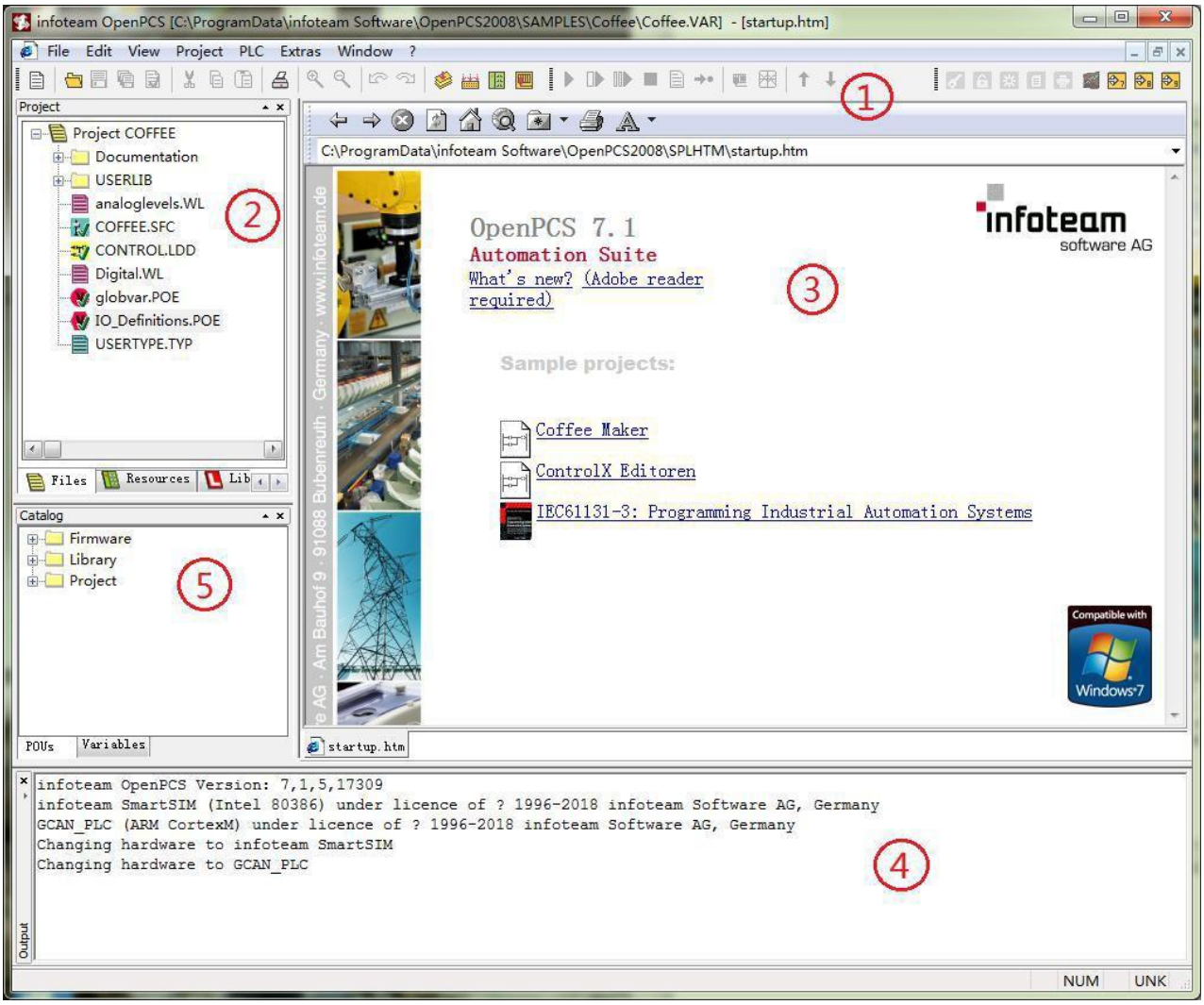
⑨ Click PLC again Online in drop down list The online state can be closed, i.e. PLC Offline with the computer.



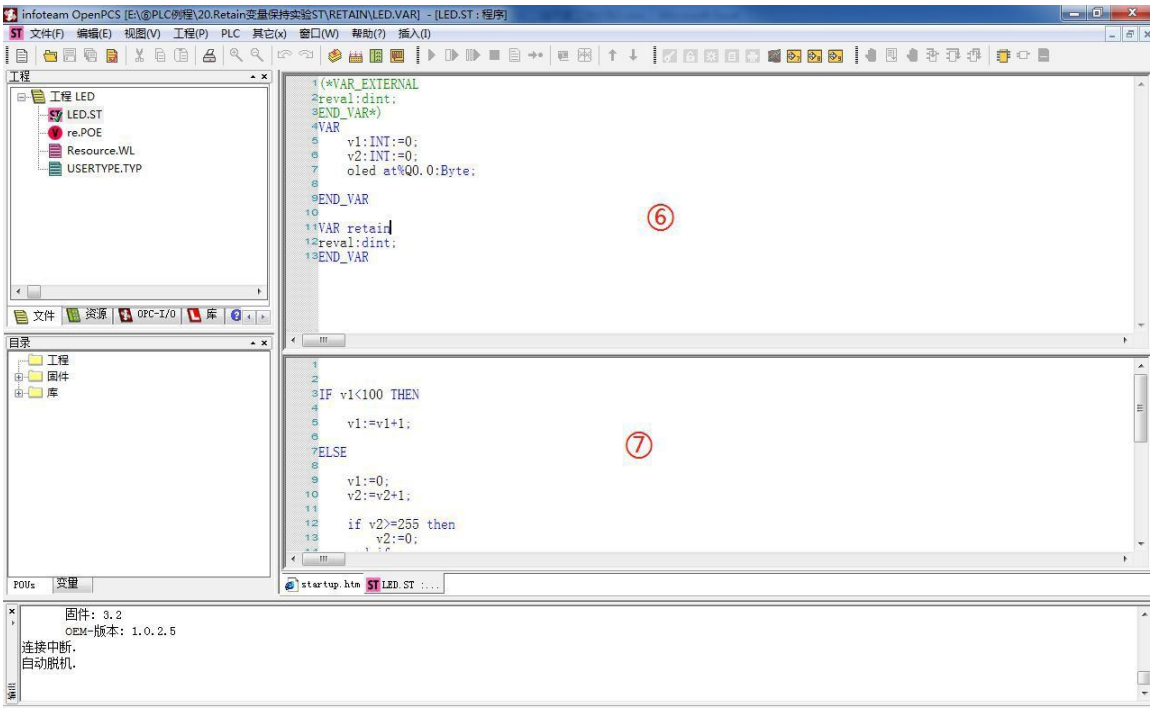
Second, the basics of programming

1. OpenPCS Function introduction

1.1 interface structure



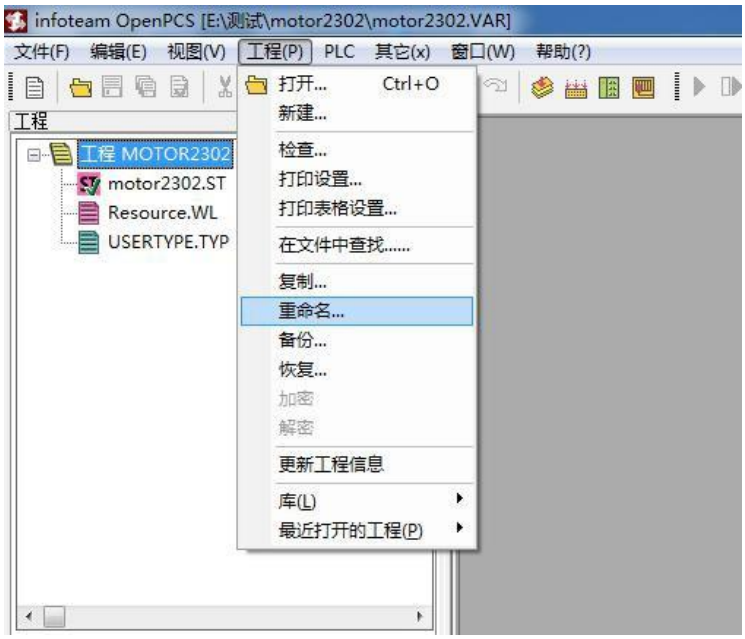
①Menu bar ②Project file box ③Information display window ④Debug window ⑤Function block window Double-click the program file in the project box, the following window will be displayed:



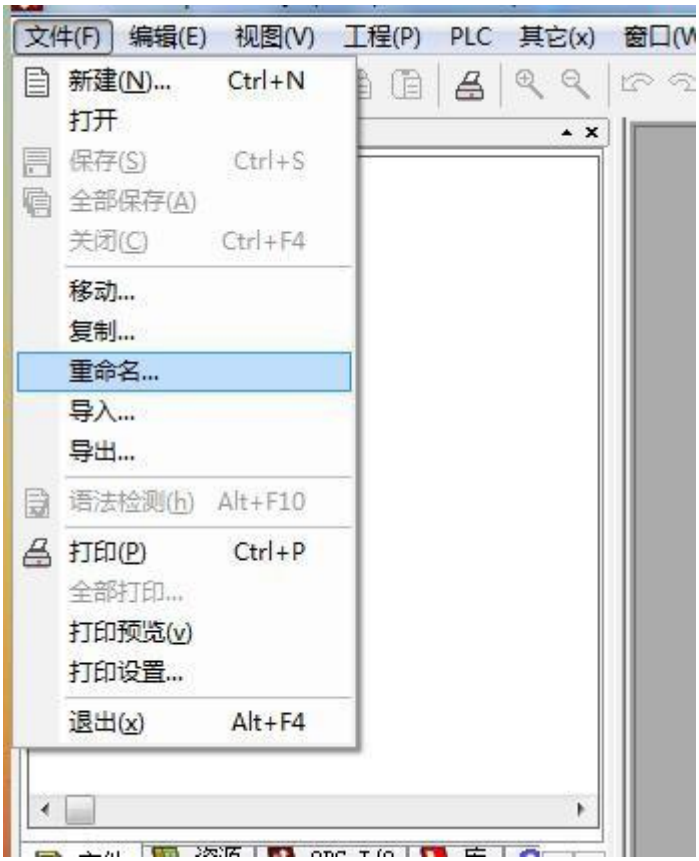
⑥Variable declaration area ⑦Program editing area

1.2 Basic software functions

(1) Project file rename: menu bar -project -rename.

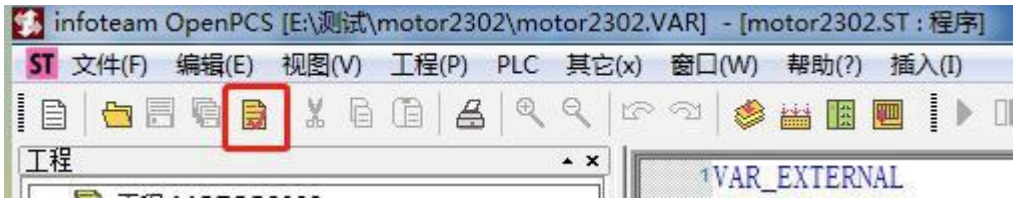


(2) Program file renaming: Select the file to be renamed, click the menu bar -File -Rename.



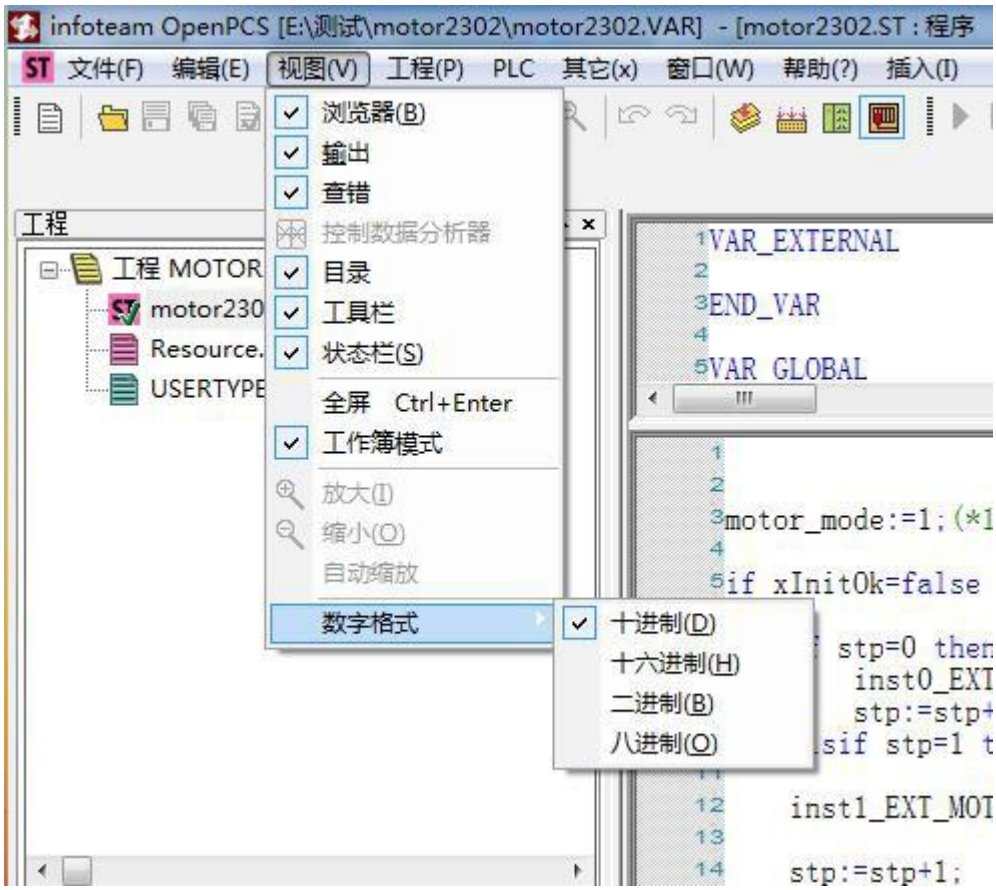
(3) Delete project file: Select the file to be deleted and click delete on the keyboard key.

(4) Grammar check: Click this icon. The purpose of syntax checking is to check whether the program is written for syntax errors.

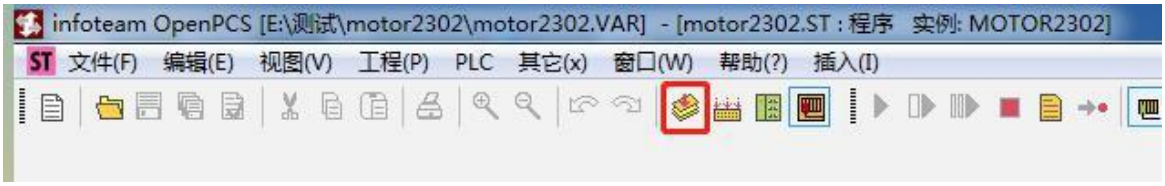


(5) Add variables to the variable monitoring area: click the resource in the project box, select the variable to be monitored and double-click.

（6）The data display format of the manipulated variable monitoring area: menu bar -view -number format.



（7）Compile: Click the Compile button in the menu bar.

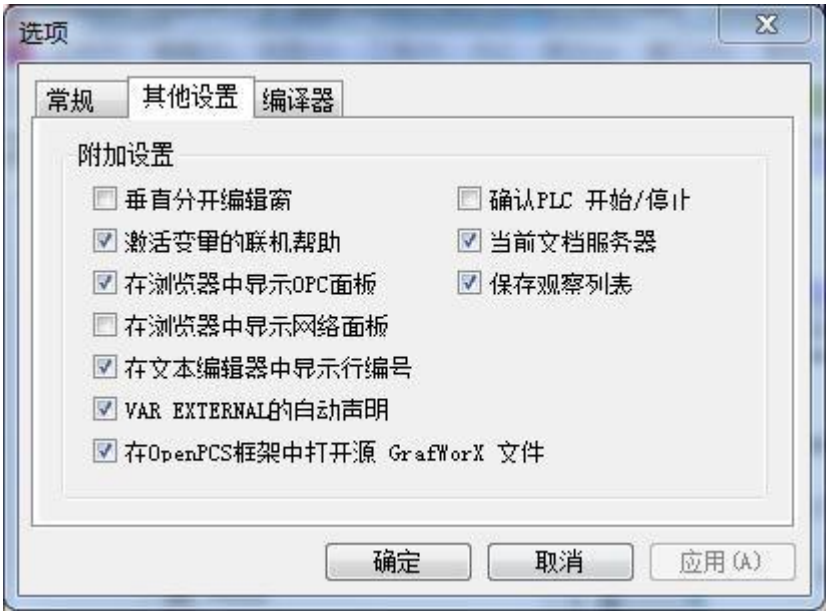


（8）Regenerate the current resource: Click the Regenerate button in the menu bar.

（9）Breakpoint debugging: ①Set breakpoint ②Toggle breakpoint ③Delete breakpoint ④Step forward ⑤Step over ⑥Step out.

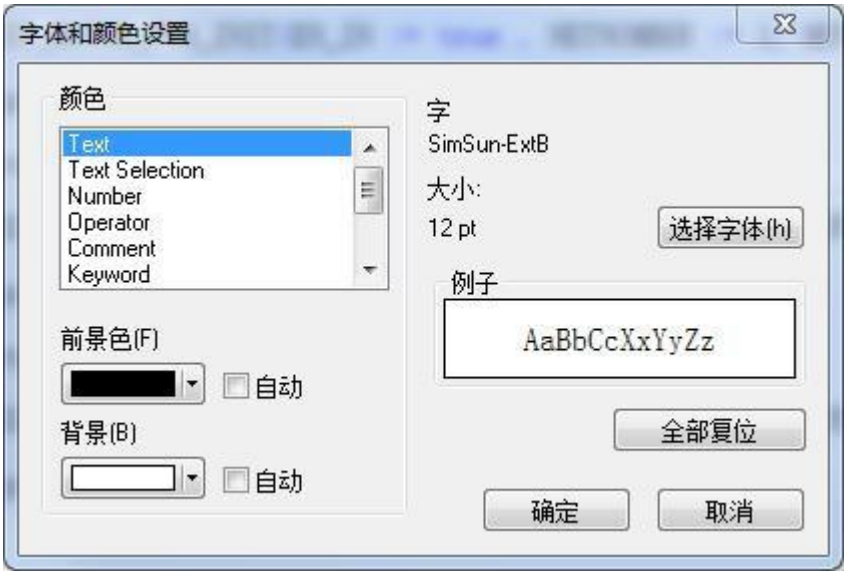
Note: The function of setting breakpoints is to let the program execute line by line, which is convenient to find out the problem.

（10）Display the number of lines of code: Menu bar -Other -Options -Browser, select other settings in the pop-up window, and check "Show line numbers in text editor". Then click Apply, OK.



（11）Adjust code font, color, background color: menu bar -other -font /color.

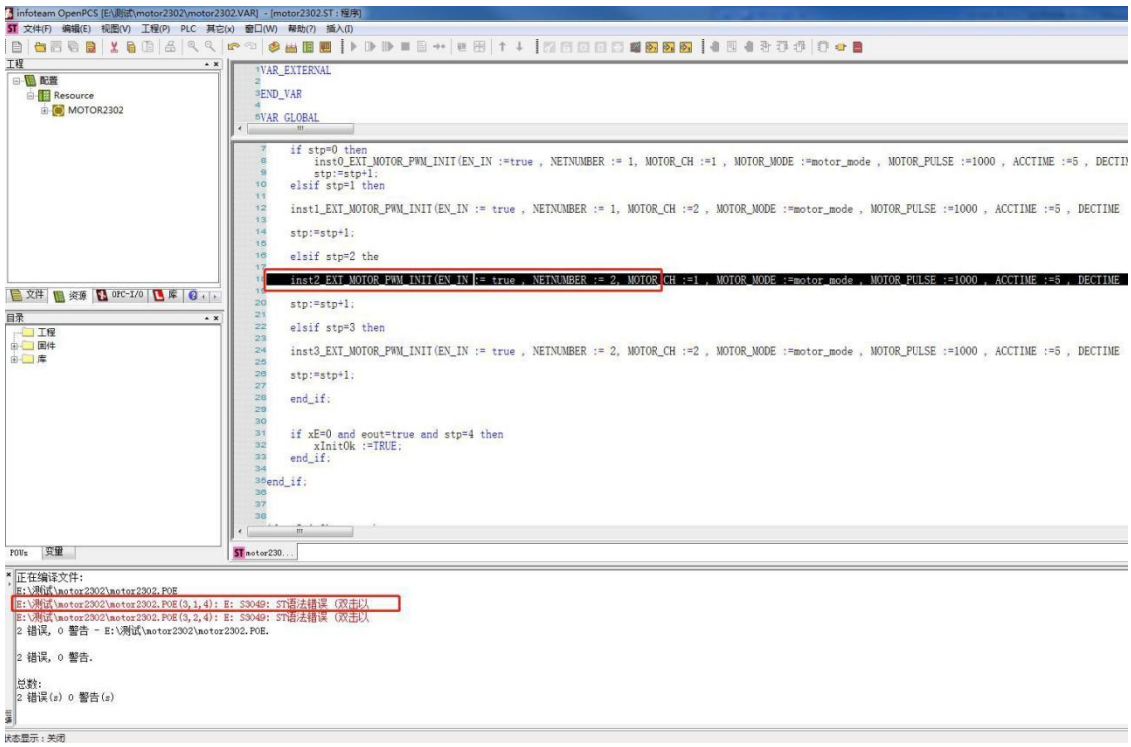
Here you can choose font, font size, foreground color, background color.



（12）Project backup and recovery; 1.Menu bar -Project -Backup 2.Menu bar -Project -Recovery

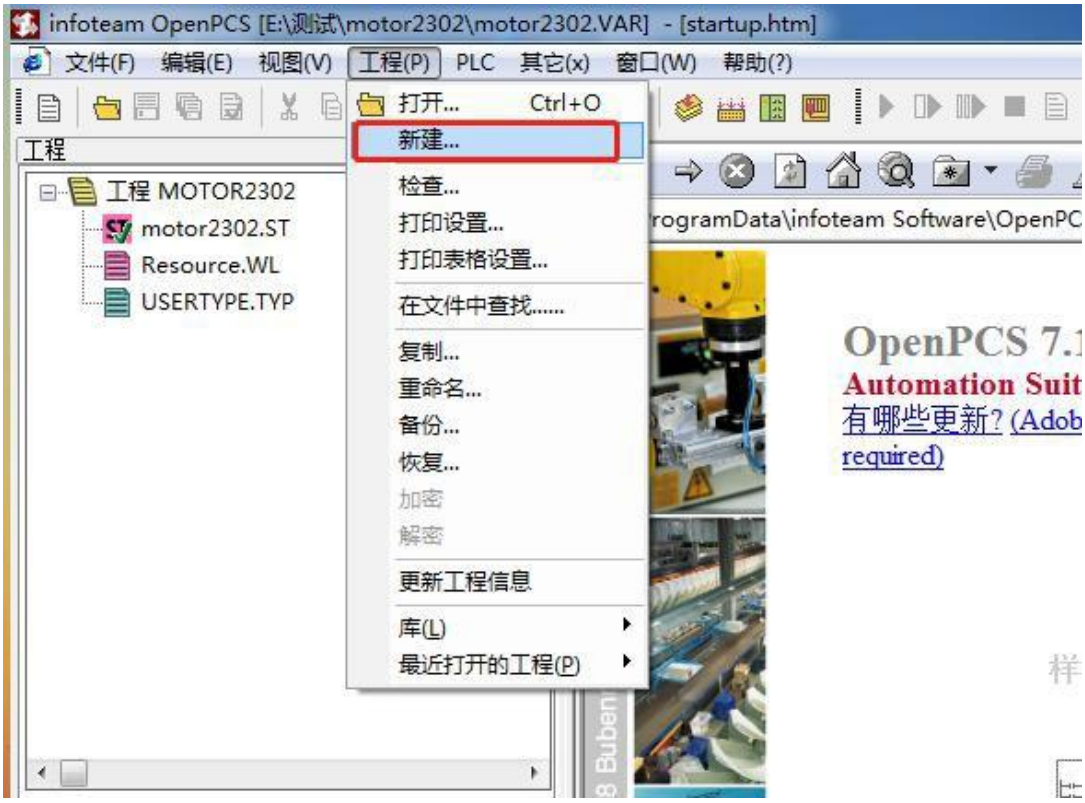
Backup will save the current project as a .BAK file, and restore can open the previously saved .BAK file. This can be backed up at any time when the program is written, which is convenient for comparison with the final version of the program. At the same time, when sending a project to others, it is not necessary to send the entire project folder, only the .BAK file needs to be sent.

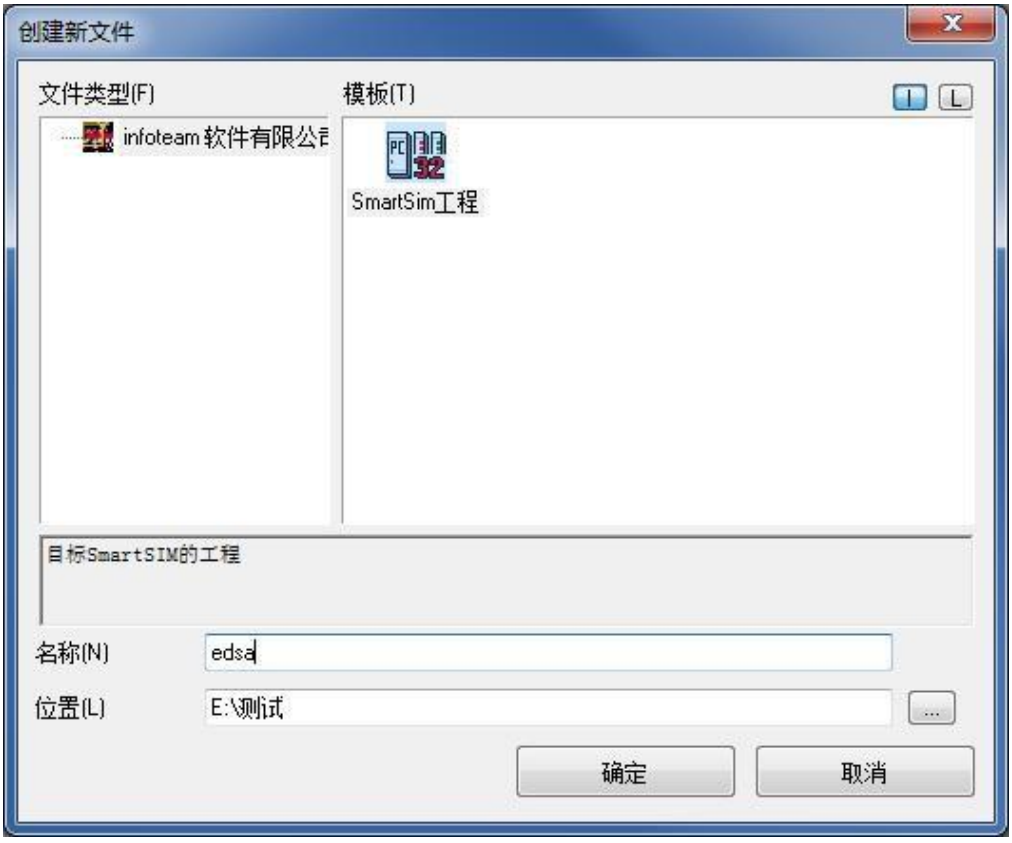
（13）Syntax error check: When an error is reported in compilation or syntax check, double-click the error message, and the cursor will jump to the wrong place. It may be this line, or it may be the previous line or the next line.



1.3 New Construction

New project: Click the menu bar -project -new -name





1.4 create a new file

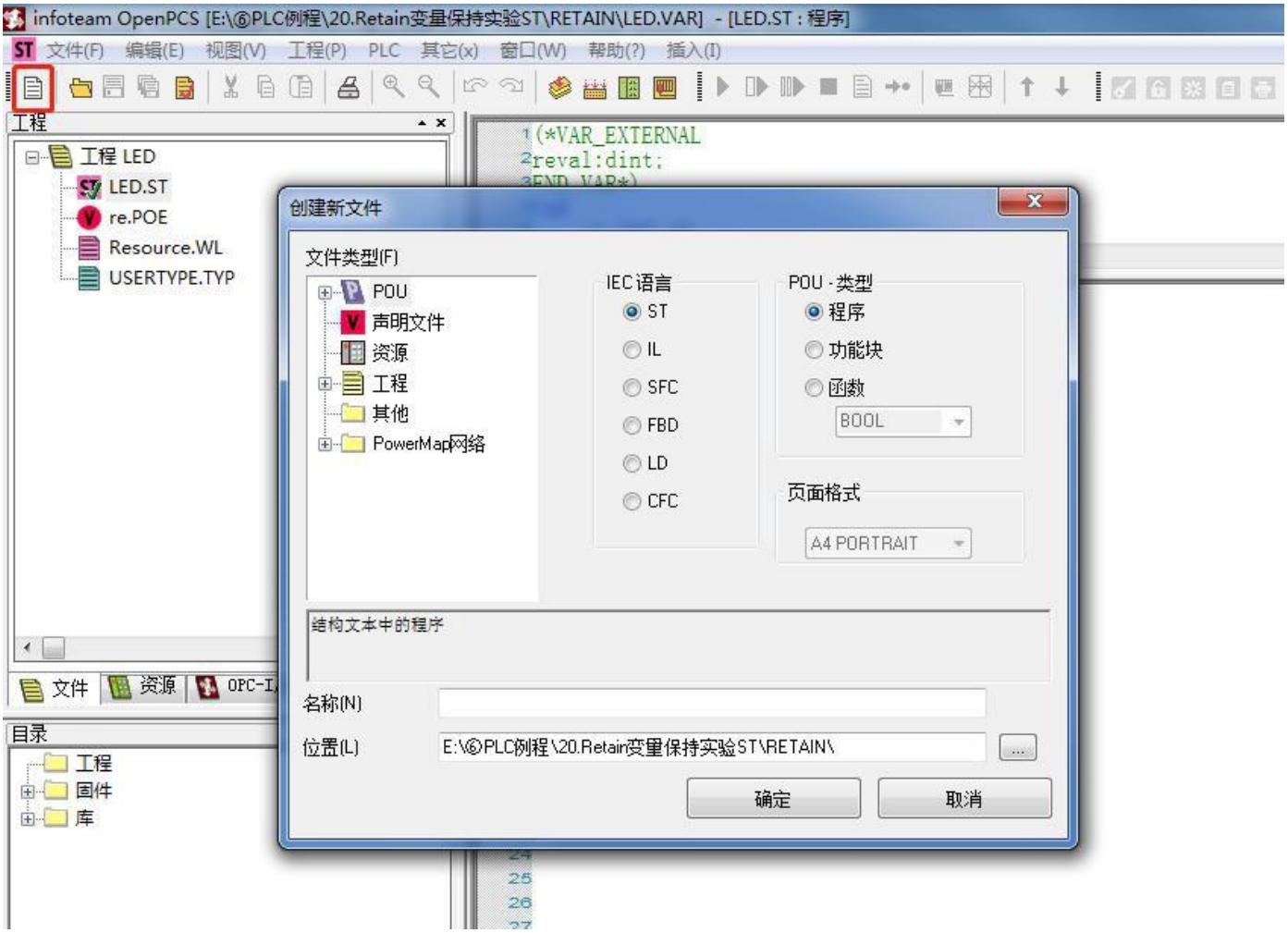
There are two types of new files, one is to create a new program POU , and the other is to create a new declaration file. There are 5 languages that can be selected for the new program file , namely , the 5 standard programming languages of IEC61131-3 . There are two types that can be selected: program and function block.

The newly created declaration file is divided into global variables and direct global variables. When the same variable is shared among multiple program files, the variable needs to be declared in the global variable; when the variable with direct physical address (AT%…) is shared among multiple program files, then the variable needs to be declared directly in the direct physical address. declared in a global variable.

When naming, the name of the file cannot start with a number, and the name cannot contain Chinese characters.

1.4.1 new program

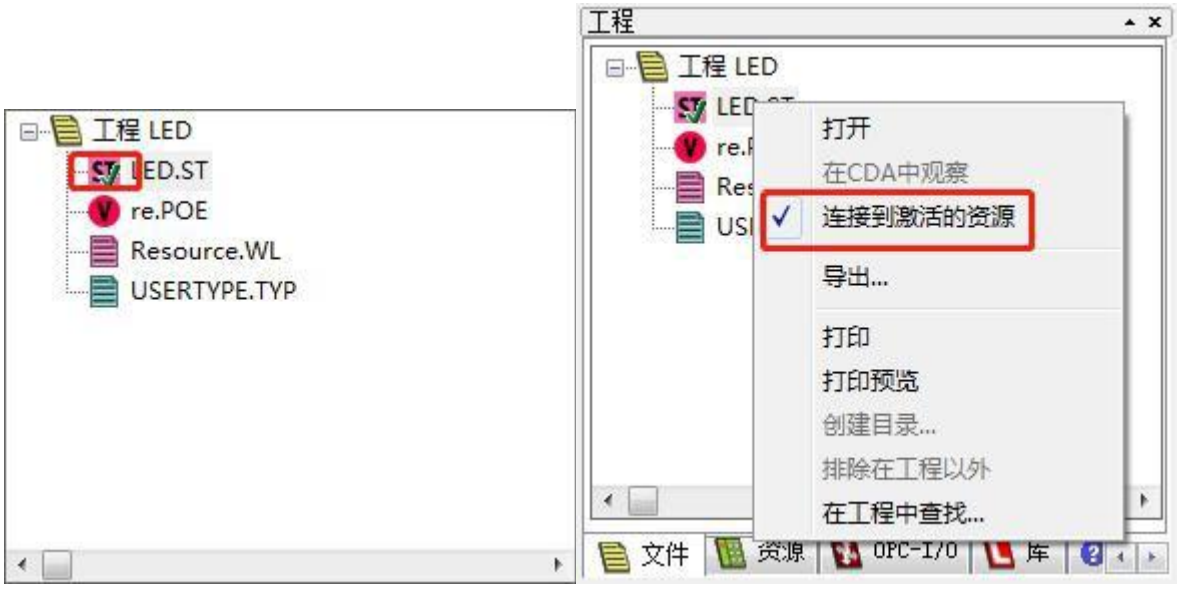
Click the white page icon in the upper left corner to create a new program page, select POU for the file type, select the language of the program page in the IEC language area, select the program type on the POU type page, and name the new program page.


infoteam OpenPCS [E:\⑥PLC例程\20.Retain变量保持实验ST\RETAIN\LED.VAR] - [LED.ST : 程序]
文件(F) 编辑(E) 视图(V) 工程(P) PLC 其它(x) 窗口(W) 帮助(?) 插入(I)
工程
工程 LED
LED.ST
re.POE
Resource.WL
USERTYPE.TYP
(*VAR_EXTERNAL
reval:dint;
END_VAR*)
创建新文件
文件类型(F)
POU
声明文件
资源
工程
其他
PowerMap网络
IEC 语言
ST
IL
SFC
FBD
LD
CFC
POU - 类型
程序
功能块
函数
BOOL
页面格式
A4 PORTRAIT
结构文本中的程序
名称(N)
位置(L)
E:\⑥PLC例程\20.Retain变量保持实验ST\RETAIN\
确定
取消
文件
资源
目录
工程
固件
库

After the creation is complete, click OK, and the following dialog box will appear:

Click Yes, and a green check mark will appear on the icon of the program file, indicating that the program file has been linked to the resource.

At the same time, if you want to cancel the link to the resource, right-click the program file and cancel the link to the active resource, so that the program



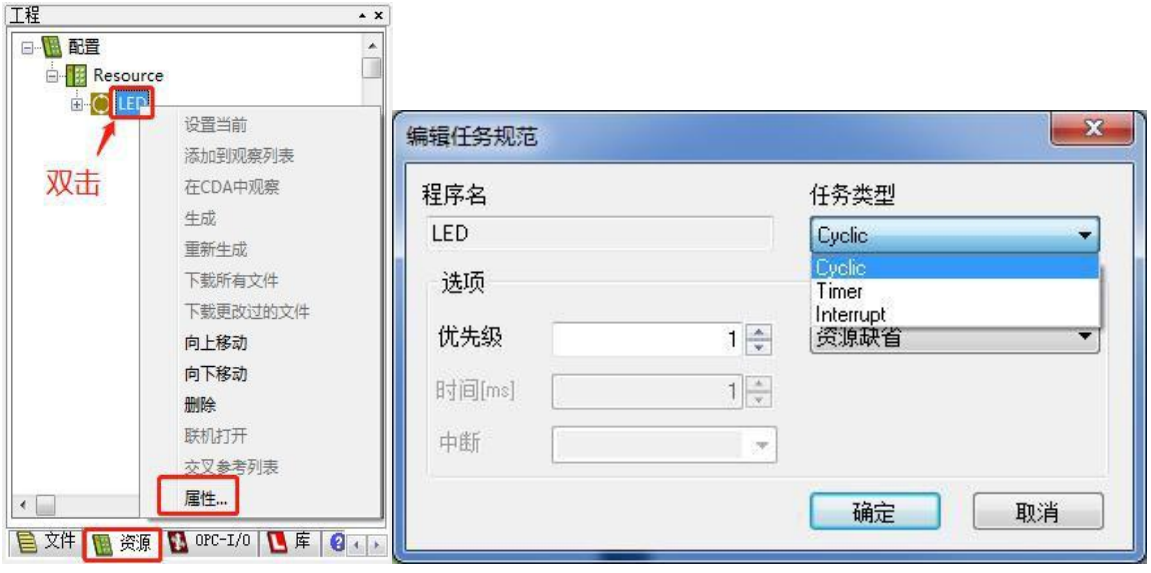
file will not be compiled and downloaded.

1.4.2 New declaration file

Click the white page icon in the upper left corner to create a new program page, and select the declaration file for the file type to create a global variable declaration page.

2. Getting Started with Simple Programming Statements

OpenPCS project can have multiple program files, and the program execution mechanism is simultaneous execution. There is no concept of main program subroutines. A program file is called a task in OpenPCS, and the task types are divided into Cyclic and Timer. When the task type is Cyclic, the loop period below 1000 lines of code is 1ms; when the task type is Timer, the loop time can be set freely, but it cannot be lower than the fastest loop period that the program can execute, that is, 1ms. The specific modification method is as follows:



2.1 variable declaration

Note: All symbols used in **OpenPCS** must be

English symbols. local variable in

VAR

END_VAR

statement in. All variables, basic instructions, function blocks are case-insensitive.

（1）Intermediate variable declaration: variable name : data type; Example: **GCANPLC** :BYTE;

（2）Initial value of variable setting: variable name : data type: = value; example: **GCANPLC** :BYTE:=0;

（3）Variable declaration with direct address:

Variable name AT%I0.0: data type; (represents the actual address

of the variable as I0.0, I represents input) Example: **GCANPLC**

AT%I0.0:BYTE;

Variable name AT%Q0.0: data type; (represents the actual address

of the variable as Q0.0, Q Representative output) Example:

GCANPLC AT%Q0.0:BYTE;

（4）Array declaration:

array name : array[0..9] of byte ; (represents that there are in the array 10

elements 0~9, the type of the elements in the array is byte) Example:

GCANPLC :array[0..9] of BYTE; the first element is GCANPLC[0] .

（5）Function block declaration:

- **defined name** :function block name ;(If a function block is to be used multiple times in a program ,it must be declared multiple times in the declaration area)

Example: **inst0_ton** :TON; **Inst1_ton** :TON;

The table of data types is shown in the following table:

type of data	meaning	Data length
BOOL	boolean	1 bit
BYTE	byte	8 bits
WORD	Character	16 bit
DWORD	double word	32 bit
SINT	short	8 bits
USINT	unsigned short	8 bits
INT	Integer	16 bit
UINT	unsigned int	16 bit
DINT	double integer	32 bit
UDINT	unsigned double integer	32 bit
REAL	real numbers	32 bit
CHAR	character	
STRING	string	
TIME	time	
DATE_AND_TIME	time and date	

2.2 programming syntax

All the statements in the example programs in this section have no actual meaning or function, but are only examples of the usage of the statements. If you need to use the statements, you can write your own programs according to the usage of the syntax examples in this section.

2.2.1 math operator

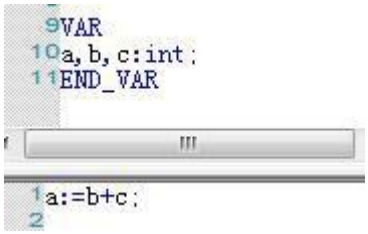
:= assignment operator + Addition- _

Subtraction * multiplication / Division example:

a := b + c; means that the b add c The result is

assigned to a.

Note: A semicolon needs to be added after the end of the statement, and the data type of the variable to be operated and the variable to be assigned must be consistent (**BYTE** cannot be directly operated, and needs to be converted to data types such as **INT**, **UINT**, **SINT**, **USINT**, etc.).



2.2.2 comparison directive

< less than > greater than = equal <= less

than or equal to >= greater than or equal to <> not

equal to 2.2.4 middle IF Statement display.

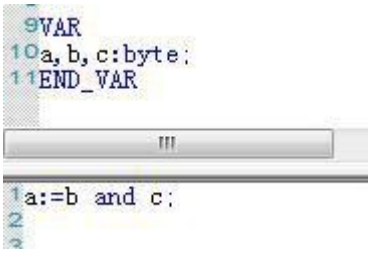
2.2.3 Logical Operators

or or and , & and not non- orn or not andn

NAND example: a := b and c; means to b and c The result of the

AND operation of each bit is assigned to a.

Note: A semicolon needs to be added after the end of the statement, and the data type of the variable to be operated and the variable to be assigned must be consistent.

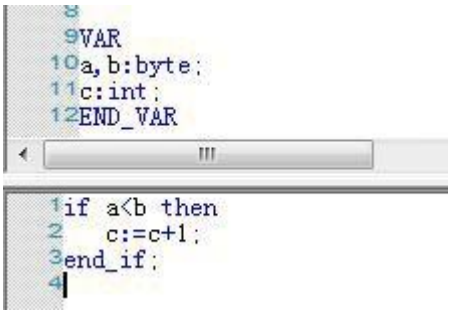


2.2.4 IF statement

（1） IF XXX Condition THEN execute XXX command END_IF;

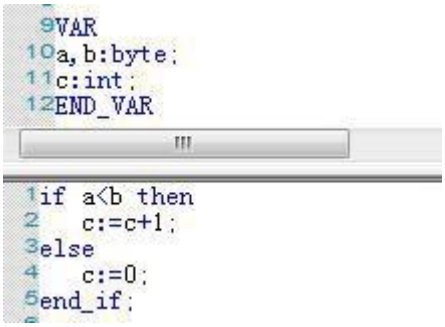
Example: IF a < b THEN c := c + 1; END_IF; means if a less than b, then c Execute self-add 1 operate.

TIPS: Although **BYTE** variables cannot perform mathematical operations, they can do mathematical comparisons.



（2）IF XXX Condition THEN execute XXX Command ELSE execute XXX command END_IF;

Example: IF a < b THEN c := c + 1; ELSE c := 0; END_IF; means if a less than b, then c Execute self-add 1 operation, otherwise the execution will c



assign as 0 operate.

2.2.5 CASE statement

CASE statement can choose to execute different commands depending on the value of the variable being checked, although ELSEIF is nested with IF

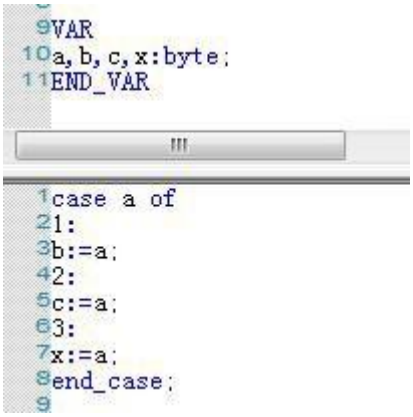
It can also be implemented, but when there are too many values for the variable being checked, it is more concise and convenient to use the CASE statement.

CASE xx variable OF 1: execute XXX command ; 2: execute XXX Order ..... N: execute XXX Command END_CASE;

例：case a of 1: b:=a; 2: c:=a; 3: x:=a; end_case;

means to check the value of variable a, when a=1, assign a to b; when a=2, assign a to

c ; when a is equal to 3 , assign a to x .



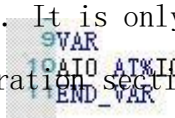
2.2.6 function block call

2.1 of this chapter . When programming, you need to call this function block according to the name of the function block declaration, for example: inst0_GPRS345G_STATUS(EN_IN := , NETNUMBER := | := STATUS, := EN_OUT);

inst0_GPRS345G_STATUS outside the brackets is the name of the function block when it is declared; inside the brackets, the vertical bar | in front is an input variable, and the right side of := can be filled with data or variables; the vertical bar | behind is an output variable, and the left side of := must be Fill in the intermediate variable to accept the output value of the function block. For the types and meanings of specific function blocks, please refer to Chapter 3.

3. IO Introduction to module configuration

IO module of GCAN-PLC series products is an automatic configuration method, which does not require manual configuration. The extended IO is inserted into the back of the PLC and automatically has an address. It is only necessary to create a variable with the actual address in the program declaration section, then this variable can represent the actual input and output points.



For example: AI0 AT% I0.0: INT ;

That is to say, the variable AI0 represents the input point whose starting address is I0.0 , and the data length is 16 bits. Analog IO of GCAN series

The data length occupied by each channel is 16 bits, so AI0 can represent exactly one channel of analog quantity.

GCAN series extended IO is related to the sequence of matching to the back of the PLC. Start with the IO closest to the PLC, address I0.0

Or Q0.0, the ones at the back go to the back in turn. **The input does not affect the address of the output, and the output does not affect the address of the input .**

For example, the IOs behind the GCAN-PLC are 1008, 2008, 3654, and 4652 in sequence. Then the first addresses of these 4 modules are I0.0, Q0.0, I1.0, Q1.0 respectively. Among them, the 8 channel addresses of 1008 are I0.0~I0.7, which can be defined as DI0 AT%I0.0:BOOL; to I0.7:BOOL;, or DI1008 AT%I0.0:BYTE;, the whole piece of 1008 The length is 1 byte, so the first channel address of 3654 is I1.0, the range is I1.0~I3.0, and so on; the output is the same, just replace I with Q.

The number of channels of each module and the length occupied by each channel are shown in the following table, which is convenient for calculating the IO address.

product signal	Function	number of channels	The data length occupied by each channel
GC1008	Digital input (PNP)	8	1 bit
GC1018	Digital input (NPN)	8	1 bit
GC2008	Digital output (PNP)	8	1 bit
GC2018	Digital output (NPN)	8	1 bit
GC1502	Pulse counter input	2	32 bits for count , 32 bits for frequency
GC2302	Pulse output (5V)	2	16 bits for speed , 32 bits for position
GC2204	relay output	4	1 bit (1 2204 module occupies 8 bits, the last 4 bits occupy null)
GC3604	Analog input (-5V~+5V)	4	16 bit
GC3624	Analog input (-10V~+10V)	4	16 bit
GC3644	Analog input (0~20mA)	4	16 bit
GC3654	Analog input (4~20mA)	4	16 bit
GC3664	Analog input (0V~+5V)	4	16 bit
GC3674	Analog input (0V~+10V)	4	16 bit
GC3804	PT100 temperature acquisition (two-wire system)	4	16 bit
GC3844	K-type thermocouple acquisition	4	16 bit
GC3822	PT100 temperature acquisition (three-wire system)	2	16 bit
GC3854	J-type thermocouple acquisition	4	16 bit
GC3864	T-type thermocouple acquisition	4	16 bit
GC4602	Analog output (-5V~+5V)	2	16 bit
GC4622	Analog output (-10V~+10V)	2	16 bit

GC4642	Analog output（0~20mA）	2	16 bit
GC4652	Analog output（4~20mA）	2	16 bit
GC4662	Analog output（0V~+5V）	2	16 bit
GC4672	Analog output（0V~+10V）	2	16 bit
GC4674	Analog output（0V~+10V）	4	12 bits (16 bits per channel, upper 4 bits are occupied)

3. Getting Started with Function Blocks

1.basic instruction set

Basic function block index table

name	Function	Chapter Index
CTU	the number of pulses of the input CD by addition	3.1.1
CTD	the number of pulses input to CU by subtraction	3.1.2
CTUD	Function block CTUD Count rising and falling edges pulse.	3.1.3
TON	timing function block	3.1.4
DT_CLOCK	clock function block	3.1.5
F_TRIG	Function block F_TRIG detects input operation CLK the state of the falling edge.	3.1.6
R_TRIG	function block R_TRIG detects the input operation CLK State change on rising edge.	3.1.7
SR	Function block SR statically converts a data element (output Q1) is a boolean 1 or 0 .	3.1.8
RS	Function block RS dynamically switches a data element (output Q1) is a boolean 1 or 0 .	3.1.9
PID	PID control	3.1.10
TOF	timing function block	3.1.11
TP	timing function block	3.1.12
CONCAT_STRING	Concatenation of two strings	3.1.13
MOD	remainder	3.1.14

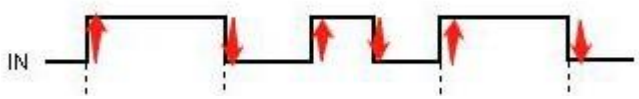
Basic function index table

Function name	Function	Chapter Index
ABS (temporarily unavailable)	absolute value	3.2.1
ACOS	arc cosine	3.2.2
ADD	Sum of two numbers (+ in ST)	3.2.3
ASIN	arcsine	3.2.4
ATAN	arctangent	3.2.5
COS	cosine	3.2.6

DELETE	subcharacters of length L starting at position P string	3.2.7
EQ (not available in ST)	true if the inputs are equal	3.2.8
EXP	Exponential power function	3.2.9
FIND	string lookup	3.2.10
GE (not available in ST)	comparison function (>=)	3.2.11
GT (not available in ST)	comparison function (>)	3.2.12

INSERT	insert string	3.2.13
LE	comparison function (<=)	3.2.14
LT	comparison function (<)	3.2.15
LEFT	L characters from the left of the string	3.2.16
LEN	string length	3.2.17
LIMIT	Limit	3.2.18
LN	logarithm to base e	3.2.19
LOG	base 10 logarithm	3.2.20
MAX	Maximum value of all input values	3.2.21
MID (temporarily unavailable)	N characters in the middle of the string	3.2.22
MIN	Minimum of all input values	3.2.23
NE (not available in ST)	not equal (<>)	3.2.24
POINTER (data type)	pointer	3.2.25
RIGHT	Take the L characters from the right of the string	3.2.26
ROL	Rotate the string to the left	3.2.27
ROR	Rotate the string to the right	3.2.28
SHL	shift string left	3.2.29
SHR	right shift string	3.2.30
SIN	Sine	3.2.31
SQRT	square root	3.2.32
TAN	Tangent	3.2.33
XOR	XOR	3.2.34

Note: a lot of content in this chapter will involve the rising edge and falling edge, so it is explained here uniformly, the rising edge is from 0 to 1 process, the next falling edge is from 1 to 0 process, a function block is enabled by the rising edge trigger, indicating that the effective enable of the function block is neither 0 nor is 1, but the transition of the enable bit from 0 to 1 is detected. The upward arrow in the figure is the rising edge, and the downward arrow is the falling edge.



3.1.1 CTU

```
inst1_CTU(CU := , RESET := , PV :=   | := Q,  := CV);
```

function block CTU calculates the number of pulses input to CU by addition. During

initialization, the counter is set to 0; if RESET receives a value of 1, the count value will be reset; each rising edge of the input CU causes the counter to increase by 1; the output CV is the current count value. If the count value is less than the PV upper limit, the output Q will be a Boolean 0, and if it is greater than or equal to the PV value, Q will be set to 1.

enter:

CU： Boolean counter

pulse **RESET：** Boolean

restart counter **PV：**

Integer count upper

limit

output:

Q： Whether the boolean

counter has reached the upper

limit value **CV：** Integer

current counter value

3.1.2 CTD

```
inst0_CTD(CD := , LOAD := , PV :=   := Q,   := CV);
```

function block CTD calculates the number of pulses input to CD by subtraction . During initialization, the counter is set to 0; if LOAD is set to 1, the PV value will be the initial value of the counter. Each rising edge of input CD will decrement the counter by 1; output CV is the current value of the counter. If the counter value is positive, output Q will be set to 0; if the counter value is 0 or negative, output Q will be set to 1.

enter:

CD： Boolean count pulse,

rising edge **LOAD：** Boolean

setting condition

PV： Integer

initial value

output:

Q： Boolean signal (whether

the count value reaches 0) **CV：**

Integer count value

3.1.3 CTUD

function block CTUD counts rising and falling pulses. During initialization, the counter is set to 0. Input each rising edge of CD; the counter will be incremented by 1; and each falling edge of CD, the counter will be decremented by 1; if LOAD is 1, the PV value will be preset to the counter; if the RESET value is 1, the count

The counter will reset; if the count value is less than the PV preset value, the output Q will display a boolean value of 0; if the count value reaches or exceeds the preset value, the output Q will be set to 1; if the count value is positive, the output QD will be a boolean value Value 0; if the count value is 0 or negative, output QD will be set to 1.

enter:

CU: Boolean calculation of

rising and falling pulses **CD:**

Boolean calculation of

rising and falling pulses

RESET: Boolean reset

condition **LOAD:** Boolean

download condition

PV: Integer

download value

output:

QU: Boolean signal (whether the

count value reaches the PV value) **QD:**

Boolean signal (whether the count

status is 0) **CV:** Integer count

status

3.1.4 TON

```
inst3_TON(IN := , PT :=   | := Q,  := ET);
```

input IN has a rising edge, the timer TON will be started , and the delay time is specified by PT ; when the timer is running, the output Q will be set to 0, if the time is up, the Q state will become 1, and its value will be maintained until IN becomes 0 ; If the PT value changes after the timer runs , it is only valid when the next rising edge IN is generated ; the output ET is the current timer value. If the time is up, ET will keep its value as long as the input IN value is 1 ; if IN becomes 0, the ET value will be 0 ; if the input IN is on, the output Q will turn on after the specified delay time.

enter:

IN : Start

condition **PT :**

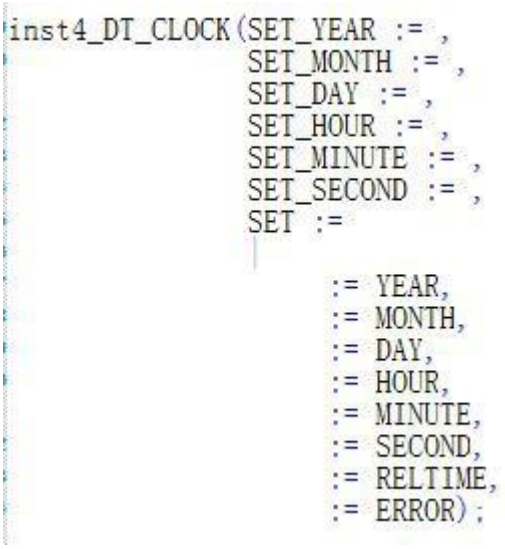
time Initial time

value Output:

Q : bool timer state

ET： time current time value

3.1.5 DT_CLOCK



This function block is to set the system clock function block, SET_YEAR, SET_MONTH, SET_DAY, SET_HOUR, SET_MINUTE, SET_SECOND, respectively set the initial year, month, day, hour, minute, second; SET is set, this parameter is set to 1 and then reset 0, the function block is valid; YEAR, MONTH, DAY, HOUR, MINUTE, SECOND are the current time year, month, day, hour, minute and second respectively. RELTIME represents the number of seconds from the computer epoch (January 1, 1970 0:00) to the current time.

ERROR indicates the display of error messages.

3.1.6 F_TRIG

function block F_TRIG detects the state of the input operation CLK; detects the change of the state from 1 to 0 in the processing cycle by outputting Q; in one processing cycle, that is, when CLK is detected and indicated by the falling edge, the output is 1.

enter:

CLK： Boolean operation input, only

for falling edge detection output:

Q： Boolean operation output, only set when CLK has a falling edge.

3.1.7 R_TRIG

```
inst6_R_TRIG(CLK :=   | := Q);
```

function block R_TRIG detects the state change of the input operation CLK ; if the state changes from 0 in the processing cycle, it is detected and indicated by the output Q being 1 ; only when the change of the CLK state within one cycle is detected and a rising edge is generated , the output state is 1.

enter:

CLK : Boolean input active

on rising edge Output:

Q : Boolean output ;indicates rising edge of CLK

3.1.8 SR

```
inst1_SR(SET1 := , RESET :=   | := Q1);
```

function block SR statically converts a data element (output Q1) to a Boolean value of 1 or 0 ; the conversion between 0 and 1 is based on the Boolean input operations SET1 and RESET ; at the beginning of the process, the output Q1 is initialized to 0 , if the value of SET1 is 1 , Function block action, output Q1 will be set to 1 ; after that, SET1 changes will not change output Q1 ; input RESET is always set to 0 when Q1 is 1 , if it can be reset; if both inputs have the value 1, the set will Dominance, that is, Q1 will be set.

enter:

Set1 : bool set

condition **Reset :**

bool reset

condition output:

Q1：output state of bool bistable element

3.1.9 RS

```
inst0_RS(SET := , RESET1 :=  |  := Q1);
```

The function block RS dynamically switches a data element (output Q1) to a Boolean value of 1 or 0. Switching between 0 and 1 is based on the Boolean input operation

SET and RESET1; before processing, the output Q1 is initialized to 0, if the SET value is 1, the function block is active, and the output Q1 will be set to 1; after that, the SET change does not change the output Q1; when the input RESET1 is 1, always set Q1 to 0; For example, it can reset the output; if both inputs have a value of 1, the reset will dominate. That is, Q1 will reset.

enter:

Set: bool set

condition **Reset1**:

bool reset condition

output:

Q1: output state of bool bistable element

3.1.10 PID

```
inst2_PID1(ENABLE := , SPV := , APV := , KP := , KI := , KD :=  |  := READY,  := CO,  := ERROR);
```

This function block is used to implement PID control, if a rising edge is detected at the input ENABLE, the function block receives the control parameters KR, TO, TD, TI and BIAS and starts the controller. In the first operation, the setpoint and the last actual value are set to the current actual value. The first time the correction variable is calculated, the initial deviation value is usually present, because the proportional gain, integral gain and derivative gain all default to 0 for the first time. The performance of the controller will be affected by the control parameters. If TI=t#0ms, the integral gain of the controller will not be calculated and will be assigned a value of 0. If TD=t#0ms, the value of the differential gain will be assigned to 0. If the gain of the parameter KR is 0, the proportional gain is not needed. Since the KR gain is related to both the proportional gain and the derivative gain, the integral and derivative gains are set to 1 here. P controller: TI=TD=0, KR ≠ 0, integral and differential gains are 1. PI controller: TD=0, KR, TI ≠ 0. PID controller: KR, TI, TD

≠ 0. There are many ways to determine control parameters (tangent to inflection point, oscillation test) , and parameters can also be determined by simulation tools. However, it is necessary that the complete control system be described as a model in terms of frequency and phase response . Models of the relevant transfer parameters of the control system can be built from known values and states and used to determine the parameters during the simulation. The set value, actual value, deviation value and correction variable are used as standard variables. The value of these standard variables varies from 0.0 to 1.0. When the function block starts to execute, the valid range of the deviation will be checked. When the deviation value exceeds the valid range Outputting ERROR will report an error. Setpoints and actual values are not checked during program run. The function block limits the value of the correction variable and the integral sum, and if the result of the calculation is negative, the variable value is set to 0. If the value exceeds 1, the variable value is set to 1. In addition, the integral sum is affected by the correction variable, also following the following rules:

If the value of the correction variable is

greater than 1, the integral sum is calculated

as follows: Integral sum = 1- (proportional gain

+ derivative gain)

If the value of the correction variable is

greater than 0, the integral sum is calculated

as follows: Integral gain = 1- (proportional

gain + derivative gain)

ENABLE: If the input rising edge is detected, the control factors KR, T0, TI, TD will be received by the system, and the accounting

Calculate the integral sum of the deviation from the initial value. The outputs READY, ERROR and CO are reset by setting ENABLE=0. The function block checks the valid area and, if necessary, displays an overflow error at the output ERROR during the transmission of parameters T0 and BIAS;

SPV: controller standard setting value (control variable), valid range is

0.0~1.0, the function block will automatically detect the parameters (but

will not detect overflow errors)

APV: controller standard actual value (process variable), valid range is

0.0~1.0, the controller will automatically detect the parameter (but will

not detect overflow error)

KP :

proportional

coefficient;

KI : integral

coefficient;

KD :

differential

coefficient;

READY：Output state of the PID controller TRUE = the function blocks of the controller are fully parameterized and have entered the pre-operational state. FALSE= The function block of the controller is not fully parameterized or incorrectly parameterized (the control parameter is outside the valid value range) ,the controller does not enter pre- operational mode.

CO：the output of the controller, calculate the correction variable of the controller (value range 0.0~1.0)

ERROR：The error code indicates the information of the execution

result of the function block, the possible error codes are

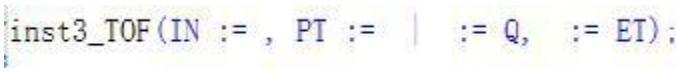
defined as follows: **0**：No error occurred during the execution

of the function block

8：The value of the specified parameter BIAS is invalid (less than 0 or greater than 1)

16：The specified value of parameter T0 is invalid (time equals 0)

3.1.11 TOF



If the input state IN is 1, it will transfer to the output Q without delay; if there is a falling edge, a timing function will start, and the delay time

There is PT decision in between; when the timer overflows, the operand Q changes to state 0 ; if the PT value changes after startup, it will take effect after the next rising edge IN ; ET contains the current time value, if the time is up, ET will Holds its value until the IN value is 0 . If the IN state changes to 1 . The ET value also becomes 0; if the input IN is off, the output Q will also be off after a specified delay.

enter:

IN : Start

condition **PT :**

time Initial time

value Output:

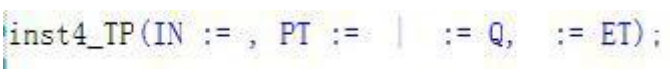
Q : bool timer

state **ET :** time

current time

value

3.1.12 TP



A rising edge of the input IN will start the time function TP , and the running time is determined by PT ; when the timing is running, the output Q is set to 1 , and any change of the input IN will be invalid; if it is started, the PT value will change to the next rising edge. IN takes effect; output ET is the current time value. If the time elapses when the IN value is 1 , ET will hold its value; any rising (falling) edge occurs without the timer running, and a command time pulse will be generated at output Q.

enter:

IN : Start

condition **PT：** time

Initial time value

output:

Q： bool timer state

ET： time current time value

3.1.13 CONCAT_STRING

```
ADC:=inst1_CONCAT_STRING(EN := , IN1 := , IN2 :=  | := ENO,  := OUT);
```

Strings IN1 and IN2 are concatenated into a new string and loaded into the working register. Strings IN1 and IN2 are concatenated in ascending order from left to right.

enter:

In1 : STRING first

string **In2** : STRING

second string Returns:

OUT: STRING two strings concatenated

3.1.14 MOD

```
inst0_MOD_USINT(EN := 1, IN1 :=122 , IN2 :=3  | ENEN := ENO, YUE := OUT);
```

MOD is the remainder, as shown in the figure, the remainder of 122 divided by 3 is 2, and the result of yue is 2.

3.2.1 ABS

```
RI:=abs(YUE);
```

figure, RI is the absolute value of YUE. The data types of RI and YUE must be the same.

3.2.2 ACOS

```
mama:=acos(mc);
```

figure, mama is the arc cosine of mc. Among them, the data types of mama and mc must be real.

3.2.3 ADD

The sum of two numbers is equivalent to +

3.2.4 ASIN

```
MC:=asin(6.0);
```

Arcsine: As shown in the figure, MC and 6.0 must be real types.

3.2.5 ATAN

```
MC:=atan(6.0);
```

Arctangent: As shown in the figure, MC and 6.0 must be real types.

3.2.6 COS

```
MC:=COS(6.0);
```

Cosine: As shown in the figure, MC and 6.0 must be real types.

3.2.7 DELETE

```
ADC:=DELETE(MB,1,2);
```

enter:

IN1: The basic string of the STRING part that needs to be removed

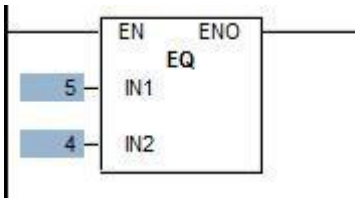
L: ANY_INT (according to support) to delete the length of the partial substring. L<0 is invalid. **P**: ANY_INT (as supported) substring start position. P<0 is invalid.

return:

STRING A shortened string. IN1 parameter is invalid.

DELETE function deletes a substring of length L in the given string IN1 starting at position P. As shown in the figure, delete the character string MB starting from the second character and having a length of 1.

3.2.7 EQ (cannot be used in ST)



in the figure, if IN1=IN2, output 1, otherwise output 0.

3.2.9 EXP

```
MC:=exp(6.0);
```

The exponent of e : As shown in the figure , the value of MC is the 6th power of e .

3.2.10 FIND

```
LS:=FIND(MB,wfe);
```

Find a string within another

string. enter:

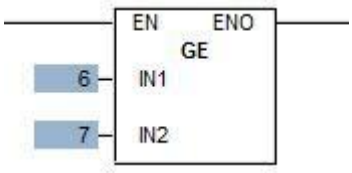
IN1 : String sequence of base strings to look for ; make this string valid via working register **IN2 :** String String to look for from the IN1 base string .

return:

The position of the first occurrence of INT

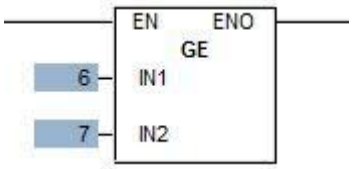
from the IN1 base string. If found, the first character position of the sequence goes into the working register, otherwise, 0 goes into the working register. If more than one is found, the character position found first goes into the working register.

3.2.11 GE



in the figure, if IN1>=IN2, output 1, otherwise output 0.

3.2.12 GT



in the figure, if IN1>=IN2, output 1, otherwise output 0.

3.2.13 INSERT

```
ADC:=INSERT(MB,wfe,2);
```

enter:

IN1: STRING string **IN2**:

STRING string to be

inserted

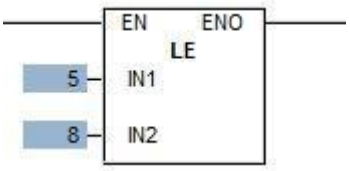
P: ANY_INT (as supported) starting position.

P<0 and P>LEN(IN1) are invalid and return:

STRING Combination of strings. IN1 invalid parameter

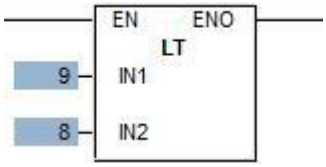
INSERT function inserts the string IN2 into the string IN1. The concatenated string consists of the first P characters of IN1, the complete IN2 and the remaining IN1.

3.2.14 LE



in the figure, if IN1<=IN2, output 1, otherwise output 0.

3.2.15 LT



in the figure, if IN1<IN2, output 1, otherwise output 0.

3.2.16 LEFT

```
ADC:=LEFT(MB,2);
```

LEFT function gets the left L characters of the current string and puts them into the working register. The input operand L defines the number of input characters.

enter:

IN: string type string

L: ANY_INT (according to support)

number of characters to get

returns:

L characters to the left of STRINGIN, IN if the parameter is invalid

3.2.17 LEN

```
LS:=LEN(MB);
```

function LEN calculates the length of the string (input operand type is STRING) and puts the length value in the working register as an INT number. enter:

In:STRING

string returns:

INT IN length

3.2.18 LIMIT

```
SHI:=LIMIT(56,YUE,89);
```

MN and MX values define upper and lower limits. The function compares the IN value with the MN and MX sizes. If IN is between the two limits ,it will be loaded into the working register. If IN is less than MN, output MN value .If IN is greater than MX, output MX value.

Input: **MN :**

Any_Num lower

limit **IN :**

Any_Num test

value **MX :**

Any_Num upper

limit Return:

Any_Num one of the input values

3.2.19 LN

```
mama:=LN(6.0);
```

logarithm to base e

in the figure, where mama and 6.0 must be real types

3.2.20 LOG

```
mama:=LOG(6.0);
```

Logarithm with base 10: As shown in the figure, where mama and 6.0 must be real types.

3.2.21 MAX

```
NIAN:=MAX(1,2,3,4,5,6,9,8);
```

Take the maximum value of all numbers: as shown, the final output value is 9.

3.2.22 MIN

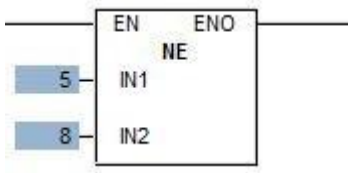
```
NIAN:=MIN(1,2,3,4,5,6,9,8);
```

Take the minimum value of all numbers: as shown, the final output value is 1.

3.2.23 MOD

See 3.1.14 for details

3.2.24 NE



equal: As shown in the figure, if IN1<>IN2, output 1, otherwise output 0.

3.2.25 POINTER

Pointer: POINTER is a data type that is a pointer. In ST, the role of a pointer is to point to an array. E.g:

PLT:=&ABUFER;

Among them, PLT is a variable of pointer type, and ABUFER is an array variable. In some function blocks, pointer-type variables are required to call arrays, read or write data.

3.2.26 RIGHT

```
adc:=right(wfe,2);
```

enter:

IN:STRING string

L:ANY_INT (according to support)

the number of characters to get

returns:

STRING the rightmost character of the string L , IN if the argument is invalid

RIGHT function loads the right part of the current string into the working register . The input operand L defines the number of characters to be entered.

3.2.27 ROL

```
b:=rol(a,2);
```

Rotate

left:

Enter:

IN : ANY_BIT bit

form N : UINT left

shifted by the

number of bits

Returns:

ANY_BITIN, rotate left by N bits

Bits shifted out to the left are rotated to the right.

3.2.28 ROR

```
b:=ror(a,2);
```

Rotate

right:

Enter:

IN : ANY_BIT bit

form N : UINT

right shift

number Returns:

ANY_BITIN, rotate right by N bits

Bits shifted out to the right are rotated to the left.

3.2.29 SHL

```
b:=shl(a,2);
```

Shift

left:

Enter:

IN : ANY_BIT bit

form N : UINT

left shift

number Returns:

ANY_BITIN, shift left by N bits

The rightmost bit is filled with zeros

3.2.30 SHR

```
b:=shr(a,2);
```

Shift right:

Input: IN :

ANY_BIT bit form

N : UINT right

shift number

Return:

ANY_BITIN, shift right by N bits

Leftmost bits are zero-padded

3.2.31 SIN

```
mc:=sin(5.0);
```

Sine: As shown in the figure, the data types of mc and 5.0 are real .

3.2.32 SQRT

```
mc:=sqrt(5.0);
```

Square root: As shown in the figure, the data types of mc and 5.0 are real . The value of mc is $\sqrt{5}$.

3.2.33 TAN

```
mc:=tan(5.0);
```

Tangent: As shown in the figure, the data types of mc and 5.0 are real .

3.2.24 XOR

```
b:=235 xor 119;
```

XOR: As shown, the result of b is 99 .

2. Extended function block

name	Function	Chapter Index
Serial function block		
USART_INIT	Serial port initialization	3.3.1
USART_STATE	Serial port status	3.3.2
USART_READ_BIN	Serial read (binary character stream)	3.3.3
USART_WRITE_BIN	Serial write (binary character stream)	3.3.4
USART_READ_CHR	Serial read (character)	3.3.5
USART_WRITE_CHR	Serial write (character)	3.3.6
USART_READ_STR	Serial read (string)	3.3.7
USART_WRITE_STR	Serial write (string)	3.3.8
EXT_USART_INIT	Initialize the extended serial interface	3.3.9
EXT_USART_READ_BIN	Extended serial interface to read binary character stream	3.3.10
EXT_USART_WRITE_BIN	Extended serial interface to write binary character stream	3.3.11
CAN function block		
CAN_INIT	CAN initialization	3.3.12
CAN_MESSAGE_READ8	CAN read	3.3.13
CAN_MESSAGE_WRITE8	CAN write	3.3.14
CAN_NMT	Send NMT management commands	3.3.15
CAN_GET_STATE	Used to request device node status	3.3.16
CAN_REGISTER_COBID	Used to register or delete PDOs and CAN Layer 2 Messages	3.3.17
CAN_PDO_READ8	Function block for reading PDO data	3.3.18
CAN_PDO_WRITE8	Function block for sending PDO data	3.3.19
CAN_SDO_READ8	Read Object Dictionary via SDO	3.3.20
CAN_SDO_WRITE8	Write object dictionary through SDO	3.3.21
CAN_SDO_READ_STR	Read string information in object dictionary through SDO	3.3.22
CAN_SDO_WRITE_STR	Write string information in object dictionary through SDO	3.3.23
CAN_SDO_READ_BIN	Read the binary data of the object dictionary through SDO	3.3.24
CAN_SDO_WRITE_BIN	Write the binary data of the object dictionary through SDO	3.3.25
CAN_RECV_EMCY	Read urgent messages from any node	3.3.26
CAN_RECV_EMCY_DEV	Read urgent messages of a specific node	3.3.27
CAN_WRITE_EMCY	Send application-specific emergency messages	3.3.28
CAN_ENABLE_CYCLIC_SYNC	Activate SYNC cycle synchronization	3.3.29
CAN_SEND_SYNC	Send sync frame (single frame)	3.3.30
CAN_ENABLE_CYCLIC_NODE_GUARD	Cyclic Send Node Protection	3.3.31

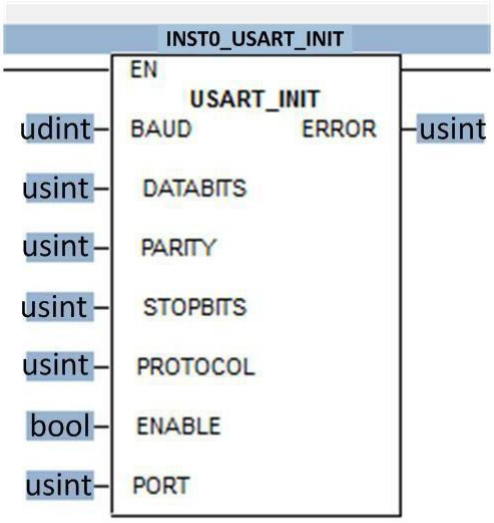
CAN_SEND_NODE_GUARD	Sending Node Protection (Single Frame)	3.3.32
CAN_RECV_BOOTUP_DEV	Read specific node bootup messages	3.3.33
CAN_RECV_BOOTUP	Read all node bootup messages	3.3.34
CAN_FILTER	CAN bus filtering	3.3.35
Ethernet, GPRS and MQTT function blocks		
LAN_INIT	Initialize the Ethernet interface	3.3.36
LAN_GET_TCPCONNECT_SOCKET	Used to establish a socket connection on the TCP side	3.3.37

LAN_TCP_SERVER_CREATE	Create a TCP SERVER connection	3.3.38
LAN_TCP_CLIENT_CONNECT	Create a TCP CLIENT connection	3.3.39
LAN_TCP_CLIENT_CLOSE	Close the TCP Client connection	3.3.40
LAN_TCP_RECV_BIN	Read binary character stream from TCP port	3.3.41
LAN_TCP_SEND_BIN	Speak binary character stream to TCP port	3.3.42
LAN_UDP_CREATE_SOCKET	Create UDPsocket connection	3.3.43
LAN_UDP_CLOSE_SOCKET	Close UDP SOCKET connection	3.3.44
LAN_UDP_RECVFROM_BIN	read UDP data	3.3.45
LAN_UDP_SENDTO_BIN	send UDP data	3.3.46
LAN_UDP_RECVFROM_STR	read UDP data (string)	3.3.47
LAN_UDP_SENDTO_STR	send UDP data (string)	3.3.48
EXT_GPRS_INIT	Extended 4G functional block	3.3.49
EXT_GPRS_STATUC	Extended 4G Status	3.3.50
EXT_GPRS_READ_BIN	Extended 4G read	3.3.51
EXT_GPRS_WRITE_BIN	Extended 4G write	3.3.52
MQTT_INIT		3.3.53
MQTT_STATUS		3.3.54
MQTT_SEND		3.3.55
Modbus function block		
MODBUS_SLAVE_INIT	Initialize modbus-RTU slave interface	3.3.56
MODBUS_SLAVE_CTRL	Execute Modbus-RTU slave control commands	3.3.57
MODBUS_MASTER_INIT	Initialize the modbus-RTU master interface	3.3.58
MODBUS_MASTER_CTRL	Execute Modbus-RTU master control commands	3.3.59
MODBUS_TCP_SLAVE_INIT	Initialize MODBUS_TCP slave interface	3.3.60
MODBUS_TCP_SLAVE_CTRL	Execute the MODBUS_TCP slave control instruction	3.3.61
MODBUS_TCP_MASTER_INIT	Initialize MODBUS_TCP U master interface	3.3.62
MODBUS_TCP_MASTER_CTRL	Execute the MODBUS_TCP master control command	3.3.63
Clock, parameter storage function block		
DT_CLOCK	Used to set the real-time clock and read time information	3.3.64
RTC	system time function block	3.3.65
GETSYSTEMDATEANDTIME	Get system time	3.3.66
SETSYSTEMDATEANDTIME	set system time	3.3.67
GETDATESTRUCT	convert date to struct	3.3.68
NVDATA_BIT	Read or write data to EEPROM	3.3.69
NVDATA_BIN	Read or write data to EEPROM (binary stream)	3.3.70
NVDATA_STR	Read or write data (string) to EEPROM	3.3.71

Pulse output, counting function block		
EXT_MOTOR_PWM_INIT	GC2302 initialization function block	3.3.72
EXT_MOTOR_EN	GC2302 enable motor function block	3.3.73
CTU	addition count	3.3.74
CTD	Subtraction count	3.3.75
CTUD	double count	3.3.76
INIT_TBL	Initialization list storage function block	3.3.77
FIFO_TBL	List members first in first out	3.3.78

LIFO_TBL	List members first in last out	3.3.79
ADD_TBL	Add list members	3.3.80
COUNT_TBL	list member count	3.3.81
other function blocks		
PID1	PID control	3.3.82
SYS_PLC_RESET	System reset restart function block	3.3.83
PTO_PWM	Pulse output PTO function block	3.3.84
PTO	Execute pulse timer	3.3.85
GETVARDATA	Get variable information	3.3.86
GETVARFLATADDRESS	Get memory space address	3.3.87
GETTASKINFO	Get the execution time of each task cycle	3.3.88

3.3.1 USART_INIT



用于初始化串行接口

Definition of Operands

BAUD： Set the baud rate used in serial communication, which can be set to the following values: 1200, 2400, 9600, 19200, 38400, 57600 ,115200 ;

DATABITS： Set the number of data bits to be used, for example: 7：data bits ； 8:8 data bits;

PARITY： Set parity for secure data transfer, for example: 0:no parity

1:odd check

2:even check;

STOPBITS： Set the number of stop bits to be used, for example: 1:1 stop bit; 2:2 stop bits;

PROTOCOL： Set the handshake protocol rules to use, for example:

0: no protocol

1: XON/XOFF

2: hardware handshake (RTS/CTS

flow control); **ENABLE：** enable

or disable serial interface:

true: enable serial interface

false: close the serial interface;

PORT： Serial interface definition number to be used, 1 is RS232, 2 is RS485;

ERROR： The error code describes information about the result of the execution of the function block. Possible error codes are defined as follows:

0：No error occurred while executing the function block

1：A hardware error occurred while executing the function block

2：The selected interface (PORT) does not support

4：The selected baud rate (BAUD) is not supported

8：The selected number of data bits (DATABITS) are not supported

16：The selected parity (PARITY) is not supported

32：The selected stop number is not supported

64：The selected

handshake protocol does

not support Description:

The function block initializes the serial interface with the specified parameters. Errors that may occur during execution are shown in ERROR . Different error codes will be superimposed and need to be split (for example, 6=2+4 means the selected

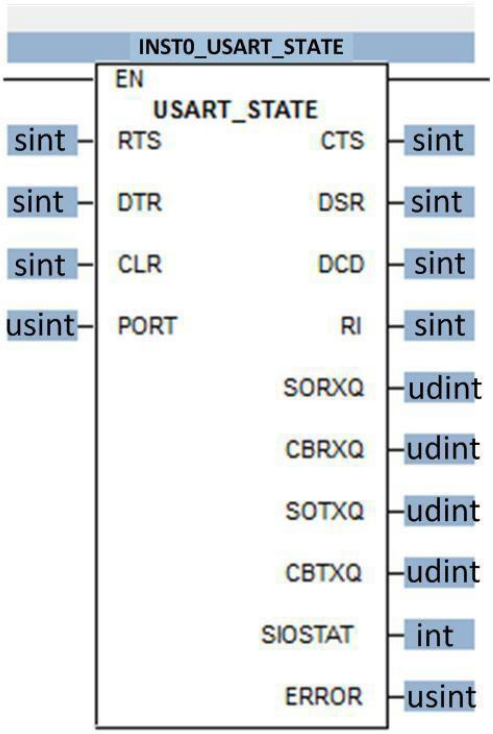
interface is not supported and the selected baud rate is not supported). The following routine uses the function block USART_INIT to initialize the serial interface, and sets the following parameters: serial port number is RS485, 9600 baud rate, 8 data bits, no parity, 1 stop bit, no protocol.

Sample program:

```
VAR
```

```
    mPORT:USINT;
    xInitOk : BOOL := FALSE; inst0_USART_INIT:USART_INIT;
    xE:usint;
    END_VAR


    mPORT:=2;
    if xInitOk=false then
    inst0_USART_INIT(BAUD :=9600 , DATABITS :=8 , PARITY :=0 , STOPBITS :=1 , PROTOCOL :=0 ,
ENABLE :=1 , PORT :=mPORT| xE := ERROR);
    if xE=0 then xInitOk :=TRUE
    end_if;
    end_if;
```

3.3.2 USART_STATE



用于设置和检索串行接口的状态信息

Definition of Operands

RTS： RTS signal state to set

-1：Does not

affect the current

state 0：

Deasserts the

signal

1：Set the signal to be valid

DTR： DTR signal state to be set

-1：Does not

affect the current

state 0：

Deasserts the

signal

1：Set the signal to be valid

CLR： clear send and receive buffers

-1：does not affect the current state

1: Clear the receive buffer

2: Clear the send buffer

3: Clear transmit and

receive buffers **PORT:**

Serial interface definition

number to be used **CTS:**

Determine CTS signal state

-1: Signal is

not supported 0:

Signal is set to

invalid

1: Signal set to valid

DSR: Determined DSR signal state

-1: Signal is

not supported 0:

Signal is set to

invalid

1: Signal set to valid

DCD: Determined DCD signal state

-1: Signal is

not supported 0:

Signal is set to

invalid

1：Signal set to valid

RI： Determined RI signal state

-1：Signal is

not supported 0：

Signal is set to

invalid

1：Signal set to valid

SORXQ： Determines the overall size of the receive

buffer (size of the Rx queue) **CBRXQ：** Current number of

characters in the receive buffer (number of bytes in

the Rx queue)

SOTXQ： Determines the overall size of the transmit buffer (Tx queue size) CBTXQ：Current number of characters in the transmit buffer (Tx

bytes in the queue)

SIOSTAT： USART specific status register (eg: overrun, framing error, etc .;see respective controls) **ERROR：** Error code information describing the result of function block execution.

Possible error codes are defined as follows: 0:No error occurred while executing the function block

1:A hardware error occurred while executing the function block

2:The selected interface (PORT) is not supported

8:If hardware handshake is active, the RTS signal is not affected (USART_INIT), PROTOCOL： =2

16:If hardware handshake is active, DTR signal is not affected (USART_INIT), PROTOCOL： =2

32:Direct setting of RTS signal is not supported

64:Direct setting of DTR signal is not supported

128:Clearing send and receive buffers is not supported

255:The selected interface (PORT) is not initialized

Description:

The function block is used to set and retrieve the status information of the serial interface. Actual availability or support of parameters depends on the corresponding hardware properties of the interface. Please see each control manual for more detailed information. Possible block errors during execution are displayed at output ERROR,several errors can be signaled simultaneously (eg 24=16+8=> DTR and RTS signals are not affected during active hardware handshake).

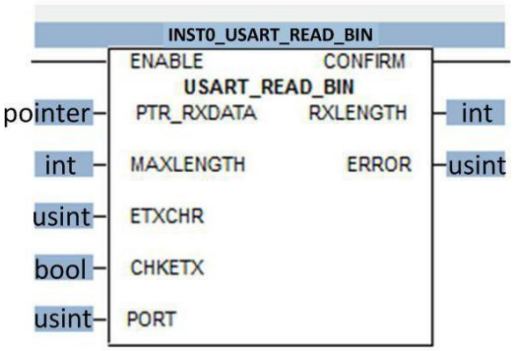
The following sample program shows how the application function

block USART_STAT determines the current state of the serial interface. Sample program:

```
VAR
FB_USARTState : USART_STATE;
xStatOk : BOOL := FALSE;
siCts : SINT;
siDsr : SINT;
siDcd : SINT;
```

```
siRi : SINT;
udiSoRrQ : UDINT;
udiCbRxQ : UDINT;
udiSoTxQ : UDINT;
udiCbTxQ : UDINT;
iUSARTStat : INT;
END_VAR

(* ----- Check USART State------*)
CheckState:
(* read current state from serial interface *)
CAL FB_USARTState (RTS := USART_STAT_DO_NOT_CHANGE,
DTR := USART_STAT_DO_NOT_CHANGE,CLR := USART_STAT_DO_NOT_CHANGE,PORT := PORTNUM
|siCts := CTS,siDsr := DSR,siDcd := DCD,siRi := RI,udiSoRrQ := SORXQ,udiCbRxQ := CBRXQ,udiSoTxQ := SOTXQ,udiCbTxQ := CBTXQ,iUSARTStat := USARTSTAT)
LD    FB_USARTState.ERROR
EQ USART_STAT_ERR_SUCCESS
ST xStatOk
... RET
```

3.3.3 USART_READ_BIN



用于从串行接口读取二进制字符流

Definition of Operands

ENABLE： enable or disable the input of the function block；**PTR_RXDATA：** address used to receive the read data byte object；

MAXLENGTH： the maximum number of bytes to read, if it is 0, the number of bytes to be read is determined by PTR_RXDATA；**ETXCHR：** the end delimiter of the binary character stream (only check if CHECKBOX=TRUE)；

CHKETX： enable / disable end delimiter character check, FALSE= end delimiter character is not checked ,TRUE= check end delimiter character is active；

PORT： The serial interface definition number to be used, 1 is RS232, 2 is RS485;

CONFIRM： Displays the completion of the function block, FALSE= reception is not successfully completed or terminated after an error ,TRUE= reception is completed successfully, characters of length RXLENGTH are available in the object addressed by PTR_RXDATA；

RXLENGTH： Number of characters read (if ERROR：=0)；ERROR：Error code describing information about the result of function block execution. Possible error codes are defined as follows:

0: No error occurred while executing the function block

1: A hardware error occurred while executing the function block

2: The selected interface (PORT) is not supported

8: The character of the terminal separator is not received, and the character of the length of MAXLENGHTH is received and terminated

128: pointer to object of unsupported data type

255: The selected interface

(PORT) is not initialized

Description:

The function block reads the binary data stream from the serial interface. The read character is stored in the object addressed by the input PTR_RXDATA . If input CHKETX is set to TRUE , check if the end delimiter defined at input EOTCHR is present in the read stream object. After the defined end delimiter is recognized, the read operation is completed and the output CONFIRM is set to TRUE . If the input CHKETX is set to FALSE or not present in the read binary stream

The defined end delimiter, the function block stops receiving after receiving the maximum number of bytes, (the internal size of the data object is determined by the address pointed to by PTR_RXDATA or the maximum number of characters defined by MAXLENGTH), in this case if the function block returns The function is normal, the CONFIRM of the output is also set to TRUE .

output RXLENGTH shows the number of characters stored in the data object addressed by PTR_RXDATA . if output

ERROR:=0 , the end delimiter defined by the input ETXCHR has been received. If ERROR:=8 is output , reception is terminated after receiving the maximum number of characters.

detection of a rising edge at input ENABLE , the block starts character reception. The function block is repeatedly called by the PLC program until the end delimiter or MAXLENGTH characters are received and terminated. To do this, the input must be set

ENABLE is TRUE to enable character reception. The function block will set the output CONFIRM to

TRUE . After finishing the process of receiving the string, the PLC program must call the function block with ENABLE:=FALSE to reset the inside of the function block to the initial state. Reception can then be continued after a successful rising edge by resetting the input ENABLE to TRUE . Active reception can be terminated at any time with ENABLE:=FALSE .

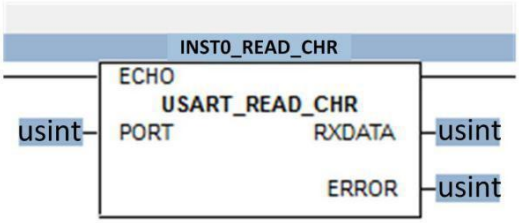
Possible errors during the execution of the function block are displayed at the output ERROR. Due to the simultaneous setting of various bits, multiple errors can be displayed at the same time. The working process of the function block USART_READ_BIN reading the serial interface character stream can be understood through the sample program .

Sample program:

In the sample program, the binary character stream is read by calling the USART_READ_BIN function block first. After the character stream is completely read, pass

The USART_WRITE_BIN function block rewrites it to the interface (refer to the sample program in Section 3.3.4).

3.3.4 USART_WRITE_BIN



用于从串行接口读取单个字符

Definition of Operands

ENABLE： enable or disable the input of the

function block; **PTR_TXDATA：** address of the

binary data object to be sent;

TXLENGTH： The number of data bytes to send, if the number is 0, the length of the object is determined internally by PTR_TXDATA as a pointer;

PORT： The serial interface definition number to be used, 1 is RS232, 2 is RS485;

CONFIRM： The result after the processing of the function block, FALSE indicates that the transfer was not successfully completed or terminated after an error, TRUE indicates that the transfer was successfully completed;

ERROR： The error code describes information about the result of the execution of the function block. Possible error codes are defined as follows:

0: No error occurred while executing the function block

1: A hardware error occurred while executing the function block

2: The selected interface (PORT) is not supported

8: Pointer refers to an object of an unsupported data type

128: The selected interface

(PORT) is not initialized

Description:

The function block writes a stream of binary characters to the serial interface. The address of the binary data to be written must be passed to the input

PTR_TXDATA. Input TXLENGTH specifies the number of valid bytes, if the value is 0,

it is determined internally by PTR_TXDATA. After a rising edge is detected at

input ENABLE , the block starts writing the character stream. The function block is called

repeatedly through the PLC program until

Character transfer is complete. To do this, the input ENABLE must be set to TRUE to enable character transfer. The function block passes the output

CONFIRM:=TRUE automatically shows that its transfer was successful. In more complex processes, the PLC program must reset its internal state via ENABLE:=FALSE . More data can then be transferred by setting the input ENABLE to TRUE and detecting a rising edge. Active transfers can be terminated at any time with ENABLE:=FALSE .

Possible errors during the execution of the function block are displayed at the output ERROR . Due to the simultaneous setting of various bits, multiple errors can be displayed at the same time.

The following sample program shows the combined application of the function blocks USART_READ_BIN and USART_WRITE_BIN . First, call the block USART_READ_BIN to read a binary character stream. After the character stream is completely read, call the block

USART_WRITE_BIN rewrites it on the interface. The sample program also adds initialization of the serial interface and flow control of program execution.

Sample program :

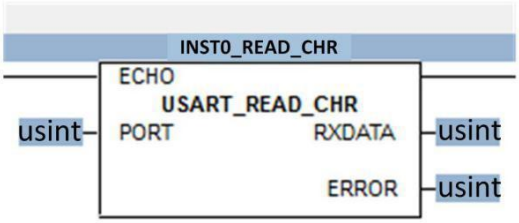
```
VAR
mPORT:USINT;
xInitOk : BOOL := FALSE;
inst0_USART_INIT:USART_INIT;
inst1_USART_INIT:USART_INIT;
xRdBinConfirm : BOOL := FALSE;
xWaitForReceipt : BOOL := true;
abDataBuffer : ARRAY[0..256] OF BYTE;
pDataObject : POINTER;
inst0_USART_WRITE_BIN:USART_WRITE_BIN;
xWrBinConfirm : BOOL := FALSE;
xTransmitting : BOOL := FALSE;
```

```
iRxDataSize : INT;

xReadStart:bool:=true;
```

```
    xWriteStart:bool:=FALSE;
    xERROR:usint;
    xE:usint;
    END_VAR


    mPORT:=2;
    if xInitOk=false then
    inst0_USART_INIT(BAUD :=115200 , DATABITS :=8 , PARITY :=0 , STOPBITS :=1 , PROTOCOL :=0 ,
ENABLE :=1 , PORT :=mPORT| xE := ERROR);
    if xE=0 then
    xInitOk :=TRUE;
    end_if;
    end_if;
    (* test USART_READ_BIN     USART_WRITE_BIN*)
    pDataObject:=&abDataBuffer;
    if xInitOk then
    if xReadStart then
    inst0_USART_READ_BIN(ENABLE :=0 ,     PORT :=mPORT   |     xERROR:= ERROR);
    xWaitForReceipt:=true;
    xWrBinConfirm:=false;
    xReadStart:=false;
    end_if;
```

```
if xWaitForReceipt then
inst0_USART_READ_BIN(ENABLE :=1 , PTR_RXDATA :=pDataObject ,
MAXLENGTH :=0 , ETXCHR :=6, CHKETX :=0, PORT :=mPORT | xRdBinConfirm:= CONFIRM,
iRxDataSize:= RXLENGTH, xERROR:= ERROR);
if xRdBinConfirm then xWaitForReceipt:=false; xWriteStart:=true;
end_if;
end_if;
if xWriteStart then
inst0_USART_WRITE_BIN(ENABLE :=0 , PORT :=mPORT | xERROR:= ERROR);
xRdBinConfirm:=false;
xWrBinConfirm:=false;
xTransmitting:=true;
xWriteStart:=false;
end_if;
if xTransmitting then
inst0_USART_WRITE_BIN(ENABLE :=1 ,PTR_TXDATA :=pDataObject ,TXLENGTH :=iRxDataSize,
PORT :=mPORT | xWrBinConfirm := CONFIRM, xERROR:= ERROR);
if xWrBinConfirm then
xTransmitting :=false;
xReadStart:=true;
end_if;
end_if;
end_if;
```

3.3.5 USART_READ_CHR



用于从串行接口读取单个字符

操作数的定义

ECHO： character echo on /off ,FALSE： no echo ,

TRUE： echo ;**PORT：** serial interface definition

number to be used ;

RXDATA： received character (if ERROR： =0) ;

ERROR： The error code describes information about the result of the execution of the function block. Possible error codes are defined as follows:

0:No error occurred while executing the function block

1:A hardware error occurred while executing the function block

2:The selected interface (PORT) is not supported

8:No characters are available in the receive buffer

16:Character echo is not supported

255:The selected interface

(PORT) is not initialized

Description:

The function block reads a single character from the serial interface. If ERROR： =8 is output, there are no characters available in the receive buffer. If ERROR： =0, the characters read are visible at output RXDATA. In this case, if you enter ECHO： =TRUE, the received character will be echoed back by the function block. Function

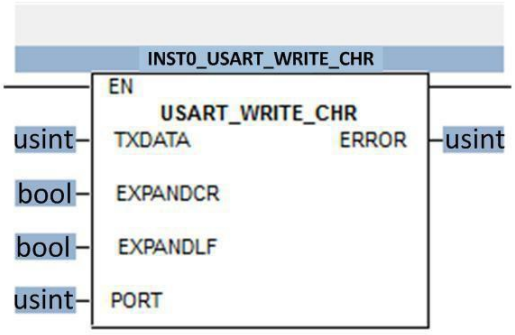
block errors that may occur during execution are displayed at the output ERROR. Due to the simultaneous setting of various bits, multiple errors can be displayed simultaneously.

Sample program :

function block USART_READ_CHR is used in conjunction with the function block USART_WRITE_CHR. First, call the function block in the program USART_READ_CHR to read characters from the interface, after successful reading, call the function block USART_WRITE_CHR to write characters on the interface

(see Section 3.3.6).

3.3.6 USART_WRITE_CHR



用于将单个字符写入串行接口

Definition of Operands

TXDATA： Enter the

character to be sent;

EXPANDCR： Auto-expand

carriage return on /off

FALSE： No auto-expand

carriage-return **TRUE：** Auto-

expand carriage-return

CR（ '$R'=13） automatically expands to CR+LF（ 'RL'=13+10）

EXPANDLF： auto-expand

newline on /off FALSE：no

auto-expand newline TRUE：

auto-expand newline

LF（ '$L'=10） expands to CR+LF（ 'RL'=13+10）

PORT： the serial interface definition number to be used;

ERROR： The error code describes information about the result of the execution of the function block. Possible error codes are defined as follows:

0：No error when executing the function block

1：A hardware error occurred while executing the function block

2：The selected interface（PORT）is not supported

8 : No space available in send buffer, characters have been lost

16 : Auto-expansion of carriage return is not supported

32 : Automatic expansion of newlines is not supported

255 : The selected interface

(PORT) is not initialized

Description:

This function block writes a single character to the serial interface. If ERROR : =1 is output , no space is available in the send buffer. If ERROR : =0 , output TXDATA successfully writes characters to the interface. If EXPANDCR : =TRUE is entered , the function block will automatically expand the character to a string with a carriage return. Similarly, if you enter EXPANDLF : =TRUE , the function block automatically expands the character to a string with a newline.

Errors that may occur during the execution of the function block are displayed as a bitmask in the output ERROR. Due to the simultaneous setting of various bits, multiple errors can be displayed at the same time. The following routine shows the function blocks USART_READ_CHR and

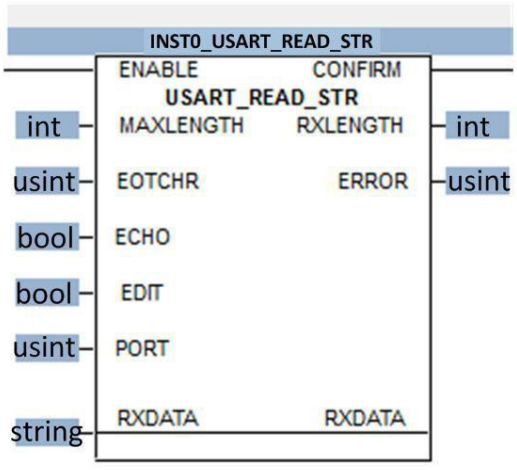
USART_WRITE_CHR is used in combination to implement serial interface read and write characters. First, the sample program calls the function block USART_READ_CHR to read a character from the interface, and then calls the function block USART_WRITE_CHR to write a character on the interface after success. Sample program:

```
VAR
xEcho : BOOL := FALSE; FB_USARTReadChr : USART_READ_CHR;
usiRxData : USINT; xRdCharSuccess : BOOL := FALSE; xExpandCR : BOOL := FALSE; xExpandLF : BOOL := FALSE;
```

```
FB_USARTWriteChr : USART_WRITE_CHR; xWrCharOk : BOOL := FALSE;

END_VAR



(* ----- Read Char------*)

CAL FB_USARTReadChr (ECHO := xEcho,PORT := PORTNUM|usiRxData := RXDATA)

(* check receive result *)
```

```
LD FB_USARTReadChr.ERROR (* character received ? *)
EQ USART_RCHR_ERR_SUCCESS
ST xRdCharSuccess RETCN
(* ----- Write Char------*)
CAL FB_USARTWriteChr (TXDATA := usiRxData,EXPANDCR :=
xExpandCR,EXPANDLF :=xExpandLF,PORT := PORTNUM)
LD FB_USARTWriteChr.ERROR EQ USART_WCHR_ERR_SUCCESS
ST xWrCharOk
...
RET
```

3.3.7 USART_READ_STR



用于从串行接口读取一个字符串

操作数的定义：

RXDATA： String variable used to receive read characters; **ENABLE：** Enable or disable the input of the function block;

MAXLENGTH： limit the number of characters to read, if the number is 0, the buffer

pass is determined internally

the length of the string and use as the number of characters to read for that delimiter (note: standard buffer size for strings in OpenPCS

is 32 characters).

EOTCHR: character for end-of-string delimiter (default: 10='$L'), eg: 0 (NUL), 10 ('$L'=newline), 13 ('$R'=carriage return);

ECHO: character echo on/off, FALSE: no echo, TRUE: return echo;

EDIT: edit mode on/off, FALSE:BS (8) are stored as normal characters in the receive string, TRUE:BS (8) are interpreted as correction characters;

PORT: the serial interface definition number to be used;

CONFIRM: output completed message through the function block, FALSE: receive unsuccessfully completed or terminated after an error, TRUE: receive successfully completed, RXLENGTH characters available, receive buffer RXDATA;

RXLENGTH: read length of string (if ERROR: =0);

ERROR: The error code describes information about the result of the execution of the function block. Possible error codes are defined as follows:

0: No error occurred while executing the function block

1: A hardware error occurred while executing the function block

2: The selected interface (PORT) is not supported

8: No terminal delimiter characters are received, terminate after receiving MAXLENGHTH characters

16: Character echo is not supported

32: Edit mode is not supported

255: The selected interface

(PORT) is not initialized

Description:

The function block reads a string from the serial interface. The read string is passed into the input RXDATA. The string terminates reception after reading the

trailing delimiter defined as the input EOTCHR , or if the string is full with the number of characters set to MAXLENGTH (taking into account EDIT, if MAXLENGTH : =0, the buffer length of the passed string determined internally and used as a delimiter) . In both cases, the returned block shows that reception is complete and the string is passed to output RXDATA, while output CONFIRM is set to TRUE to indicate that the string has been read. The output RXLENGTH shows the number of characters in the receive buffer (equal to LEN (RXDATA)) . If the output ERROR : =0 , it means that the end delimiter defined by the input EOTCHR has been received. If false: =8, read the set

After the number of characters, the reception terminates the maximum length. After detection of a positive edge at input ENABLE , the block starts character reception (first pass ENABLE : =TRUE). The function block is called repeatedly through the PLC program until the character reception (end delimiter or MAXLENGTH characters) is complete. To do this, the input ENABLE must be set to TRUE to enable character reception. The function block sets CONFIRM to TRUE after successful termination of reception . After processing the received string, the PLC program must call the function block with ENABLE : =FALSE to reset the block internally to the initial state. Operation can then continue by setting the input ENABLE to TRUE and detecting the rising edge of the received signal . Active reception can be terminated at any time by setting ENABLE : =FALSE .

If you enter ECHO : =TRUE , the function block automatically returns each received character as an echo. The character backspace (BS=8) is not stored as a normal character, but as a correction character, if EDIT : =TRUE is entered , then in the receive buffer, the last received character is deleted and the RXLENGTH reported received Character count decreased.

Possible errors during the execution of the function block are displayed at the output ERROR . Due to the simultaneous setting of various bits, several errors can be signaled.

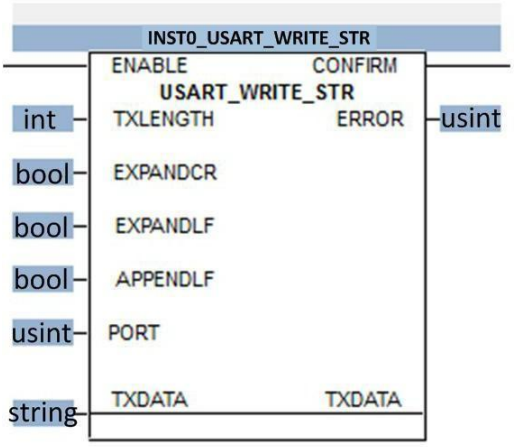
The example program shows the working process of the function block

USART_READ_STR reading the string of the serial interface. Sample

program:

The sample program shows the combined working process of the function block USART_READ_STR and the function block USART_WRITE_STR .First, the routine reads the string by calling the function block USART_READ_STR .After the string is completely read, it is rewritten to the interface via the function block USART_WRITE_STR .(Refer to the sample program in Section 3.3.8)

3.3.8 USART_WRITE_STR



用于将字符串写入串行接口

Definition of operands:

TXDATA： String to write; **ENABLE：**

Enable or disable the input of the

function block;

TXLENGTH： the number of characters to write, if the number is 0, the length of the characters and the characters TXDATA contained in the string are determined internally (equal to LEN（ TXDATA））;

EXPANDCR： Auto-expand

carriage return on /off

FALSE：No auto-expand

carriage-return; TRUE：

Auto-expand carriage-return;

CR（ '$R'=13） automatically expands to CR+LF（ 'RL'=13+10）;

EXPANDLF： auto-expand

newline on /off FALSE：no

auto-expand newline; TRUE：

auto-expand newline;

LF（ '$L'=10） automatically expands to CR+LF（ 'RL'=13+10）;

APPENDLF： auto append newline on /off

FALSE：no newline is appended

automatically, LF（ '$L'=10）; TRUE：newline

is appended automatically, CR+LF

（ 'RL'=13+10）; **PORT：** the serial

interface definition number to use;

CONFIRM： message completed via function block output, FALSE：transfer was not completed successfully or terminated after an error,

TRUE：the transfer completed successfully;

ERROR： The error code describes information about the result of the execution of the function block. Possible error codes are defined as follows:

0：No error occurred while executing the function block

1：A hardware error occurred while executing the function block

2：The selected interface（PORT）is not supported

16：Auto-expansion of carriage return is not supported

32: Auto-expansion of newlines is not supported

64: Automatic appending of newlines is not supported

255: The selected interface

(PORT) is not initialized

Description:

The function block writes a string to the serial interface. The string to transmit is passed at input TXDATA. enter

TXLENGTH specifies the number of valid characters. If the value is 0, the length of the string contained in the string is determined internally by TXDATA (equal to LEN(TXDATA)), and as the number of characters to process. In this case, the entire occupied string content is written. After a rising edge is detected at input ENABLE (by calling first), the block starts writing the string

(ENABLE: =TRUE). The function block is repeatedly called by the PLC program until the character transfer has been completed. To do this, the input ENABLE must be set to TRUE to enable character transfer. This block automatically signals successful completion by making the output CONFIRM TRUE. The PLC program must reset its initial state internally by calling the block with ENABLE: =FALSE. Further character transfers can then be initiated by detecting a rising edge at input ENABLE. Active transfers can be terminated at any time with ENABLE: =FALSE.

If you enter EXPANDCR: =TRUE, the block will automatically expand the string with ASCII + carriage return. Similarly, blocks also automatically add newlines to strings, which expand to strings if EXPANDLF: =TRUE is entered

+ line break. If the input APPENDLF: =TRUE, the block is passed to the output TXDATA with an internal newline appended after the complete transfer of the string. Depending on the input APPENDLF, this newline will be transferred as a string consisting of carriage return + newline, if needed.

Possible errors during the execution of the function block are displayed as a

bitmask at output ERROR. Due to the simultaneous setting of various bits, several errors can be signaled.

The following sample program shows the application of the function block USART_WRITE_STR. First, the block USART_READ_STR is called to read the string from the interface. After the string is completely read, the block USART_WRITE_STR rewrites it on the interface. The sample program integrates flow control for initializing the serial interface and program execution.

Sample program :

```
(* ----- Init USART ----- *) USARTInit:

CAL FB_USARTInit (BAUD := 9600,DATABITS := 8,PARITY := USART_INIT_PARITY_NO, STOPBITS := 1,PROTOCOL := USART_INIT_PROTOCOL_NO,ENABLE := TRUE,PORT := PORTNUM)
```

(* ----- Read String------*) ReadStringStart:

CAL FB_USARTReadStr (ENABLE := FALSE, (* Step 1: Reset FB *)RXDATA := strRxText, PORT := PORTNUM)

CAL FB_USARTReadStr (ENABLE := TRUE, (* Step 2: Start FB *)RXDATA := strRxText, MAXLENGTH := 0, (* no limit, use whole string length *)EOTCHR := usiEotChr, ECHO := xEcho,EDIT := xEdit,PORT := PORTNUM|xRdStrConfirm := CONFIRM, iRxDataSize := RXLENGTH)

(* ----- Write String------*) WriteStringStart:

CAL FB_USARTWriteStr (ENABLE := FALSE, (* Step 1: Reset FB *)TXDATA := strTxText, PORT := PORTNUM)

WriteStringCont:

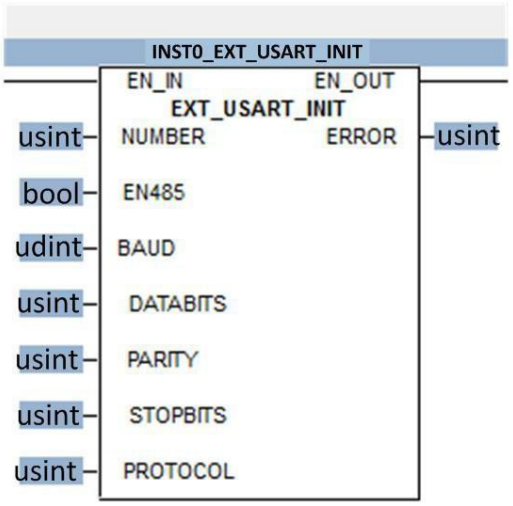
CAL FB_USARTWriteStr (ENABLE := TRUE, (* Step 2: Start FB *)TXDATA := strTxText, TXLENGTH := 0, (* no limit, transmit whole string *)EXPANDCR := xExpandCR, EXPANDLF := xExpandLF,APPENDLF := xAppendLF,PORT := PORTNUM|xWrStrConfirm := CONFIRM)

(* ----- Reset Flow Control Logic------*) LD FALSE

ST xTransmitting ST xWrStrConfirm ST xWaitForReceipt ST xRdStrConfirm RET

END_PROGRAM

3.3.9 EXT_USART_INIT



用于初始化扩展串行接口

Definition of operands:

EN_IN： Enable or disable the serial interface:

true: enable serial

interface; false:

disable serial

interface;

NUMBER： the serial number of the GC-6101 module used to expand the serial port, the serial number of the first GC-6101 module after the main control module is 1, the second is 2, and so on;

EN485： Set the serial communication mode, 1 is RS485 communication, 2 is RS232 communication:

BAUD： Set the baud rate used in serial communication, which can be set to the following

values: 1200, 2400, 9600, 19200, 38400, 57600 , 115200 ;

DATABITS： Set the number of data bits to use, for example: 7:7 data bits

8: 8 data bits;

PARITY： Set parity for secure data transfer, for example: 0: no parity

1: odd check

2: even check;

STOPBITS： Set the number of stop bits to use, for example:

1:1 stop bit

2: 2 stop bits;

PROTOCOL： Set the handshake protocol rules to use, for example:

0: no protocol

1: XON/XOFF

2:Hardware handshake (RTS/CTS flow control);

EN_OUT : function block enable output;

ERROR : The error code describes information about the result of the execution of the function block. Possible error codes are defined as follows :

0: No error occurred while executing the function block

1: A hardware error occurred while executing the function block

2: The selected interface (PORT) does not support

4: The selected baud rate (BAUD) is not supported

8: The selected number of data bits (DATABITS) is not supported

16: The selected parity (PARITY) is not supported

32: The selected stop number is not supported

64: The selected

handshake protocol

does not support the

description:

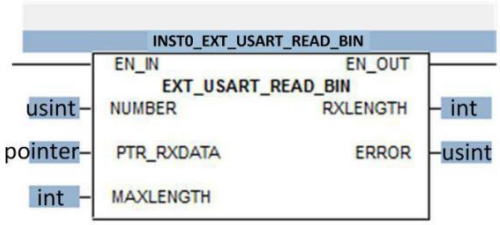
The function block initializes the serial interface with the specified parameters. Possible errors during execution are shown in the output ERROR. Different error codes will be superimposed and need to be split (for example, 6=2+4 means the selected interface is not supported and the selected baud rate is not supported). The following routine uses the function block USART_INIT to initialize the serial interface, and sets the following parameters: serial port number is RS485, 9600 baud rate, 8 data bits, no parity, 1 stop bit, no protocol.

Sample program:

```
VAR mPORT:USINT;

xInitOk : BOOL := FALSE; inst0_EXT_USART_INIT:USART_INIT;

xE:usint; END_VAR

mPORT:=2;

if xInitOk=false then
```

```
    inst0_EXT_USART_INIT(BAUD :=9600 , DATABITS :=8 , PARITY :=0 , STOPBITS :=1 , PROTOCOL :=0 ,
ENABLE :=1 , PORT :=mPORT| xE := ERROR);

    if xE=0 then xInitOk :=TRUE; end_if;

    end_if;
```

3.3.10 EXT_USART_READ_BIN



用于从扩展串行接口读取2进制字符流

Definition of operands:

EN_IN： Enable or disable the input of the function block ;

NUMBER： the serial number of the GC-6101 module used to expand the serial port, the serial number of the first GC-6101 module after the main control module is 1, the second is 2, and so on ;

PTR_RXDATA： the address used to receive the read data byte object ;

MAXLENGTH： the maximum number of bytes to read, if it is 0, the number of bytes to be read is determined by PTR_RXDATA ; **EN_OUT：** function block enable output ;

RXLENGTH： number of characters read (if ERROR：=0) ;

ERROR： The error code describes information about the result of the execution of the function block. Possible error codes are defined as follows:

0: No error occurred while executing the function block

1: A hardware error occurred while executing the function block

2: The selected interface (PORT) is not supported

8: The character of the terminal separator is not received, and the character of the length of MAXLENGHTH is received and terminated

128: pointer to object of unsupported data type

255: The selected interface

(PORT) is not initialized

Description:

The function block reads the binary data stream from the serial interface. The read character is stored in the object addressed by the input PTR_RXDATA.The function block stops receiving after receiving the maximum number of bytes, (the internal size of the data object is given by

PTR_RXDATA or the maximum number of characters defined by MAXLENGTH), in this case if the function block feedback function is normal, the output EN_OUT will also be set to TRUE.

output RXLENGTH shows the number of characters stored in the data object addressed by PTR_RXDATA. input detected

ENABLE, the block starts character reception. The function block is repeatedly called by the PLC program until the end delimiter or MAXLENGTH characters are received and terminated. To do this, input ENABLE must be set to TRUE to enable character reception. The function block will set the output CONFIRM to TRUE after successful termination of reception. After finishing the process of receiving the string, the PLC program must call the function block with ENABLE:=FALSE to reset the inside of the function block to the initial state. A rising edge can then successfully enable the successor by resetting the input ENABLE to TRUE

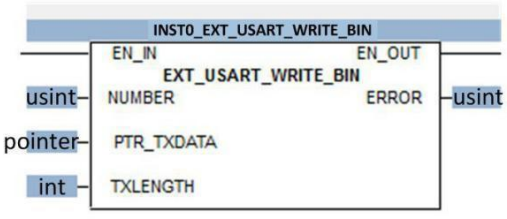
continue to receive. Active reception can be terminated at any time with ENABLE:=FALSE.

Possible errors during the execution of the function block are displayed at the output ERROR. Due to the simultaneous setting of various bits, multiple errors can be displayed at the same time. The working process of the function block EXT_USART_READ_BIN reading the serial interface character stream can be understood through the sample program.

Sample program:

In the sample program, the binary character stream is read by calling the EXT_USART_READ_BIN function block first. After the character stream is completely read, it is rewritten to the interface via the EXT_USART_WRITE_BIN function block (see Section 3.3.10).

3.3.11 EXT_USART_WRITE_BIN



用于将二进制字符流写入扩展串行接口

Definition of operands:

EN_IN： Enable or disable the input of the function block;

NUMBER： the serial number of the GC-6101 module used to expand the serial port, the serial number of the first GC-6101 module after the main control module is 1, the second is 2, and so on;

PTR_TXDATA： the address of the binary data object to be sent;

TXLENGTH： The number of bytes of data to send, if the number is 0, the length of the object is passed as a pointer in PTR_TXDATA

Internal addressing is determined;

EN_OUT： The result after the function block processing, FALSE indicates that the transfer was not completed successfully or terminated after an error,

TRUE indicates that the transfer has completed successfully;

ERROR： The error code describes information about the result of the execution of the function block. Possible error codes are defined as follows:

0: No error occurred while executing the function block

1: A hardware error occurred while executing the function block

2: The selected interface (PORT) is not supported

8: Pointer refers to an object of an unsupported data type

128: The selected interface

(PORT) is not initialized

Description:

The function block writes a stream of binary characters to the serial interface. The address of the binary data to be written must be passed to the input

PTR_TXDATA. Input TXLENGTH specifies the number of valid bytes, if the value is 0,

it is determined internally by PTR_TXDATA. After a rising edge is detected at

input EN_IN, the block starts writing the character stream. The function block is called

repeatedly through the PLC program until the word

Symbol transmission is complete. To do this, the input EN_IN must be set to TRUE to enable character transfer. The function block automatically indicates that its transfer was successful with the output EN_OUT:=TRUE. In more complex processes, the PLC program must reset its internal state via EN_IN:=FALSE. More data can then be transferred by setting the input EN_IN to TRUE and detecting a rising edge. Active transfers can be terminated at any time with EN_IN:=FALSE.

Possible errors during the execution of the function block are displayed at the

output ERROR. Due to the simultaneous setting of various bits, multiple errors can be displayed at the same time.

The following sample program shows the combined application of the function blocks EXT_USART_READ_BIN and EXT_USART_WRITE_BIN. First, the block EXT_USART_READ_BIN is called to read the binary character stream. After the character stream is completely read, the block EXT_USART_WRITE_BIN is called to rewrite it onto the interface. The sample program also adds initialization of the serial interface and flow control of program execution.

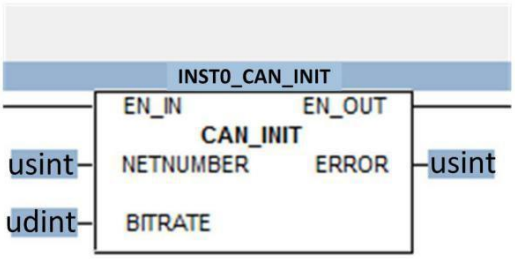
Sample program:

```
VAR
xInitOk : BOOL := FALSE; inst0_EXT_USART_INIT:EXT_USART_INIT;
```

```
xRdBinConfirm : BOOL := FALSE; xWaitForReceipt : BOOL := true;
inst0_EXT_USART_READ_BIN:EXT_USART_READ_BIN;
abDataBuffer : ARRAY[0..256] OF BYTE; pDataObject : POINTER;
inst0_EXT_USART_WRITE_BIN:EXT_USART_WRITE_BIN;
xWrBinConfirm : BOOL := FALSE; xTransmitting : BOOL := FALSE; iRxDataSize : INT;
xReadStart:bool:=true; xERROR:usint;
xE:usint; var4:INT; eout:bool;
END_VAR
if xInitOk=false then
inst0_EXT_USART_INIT(EN_IN :=TRUE , NUMBER :=1 , EN485 :=0, BAUD :=115200 , DATABITS :=8 ,
PARITY :=0 , STOPBITS :=1 , PROTOCOL :=0 | eout := EN_OUT, xE:=
ERROR);
if xE=0 and eout=true then xInitOk :=TRUE;
end_if; end_if;
(* test USART_READ_BIN USART_WRITE_BIN*)
pDataObject:=&abDataBuffer; if xInitOk then
if xReadStart then
inst0_EXT_USART_READ_BIN(EN_IN :=0 , NUMBER :=1 | xERROR := ERROR);
xWaitForReceipt:=true; xRdBinConfirm:=false; xReadStart:=false;
end_if;
if xWaitForReceipt then inst0_EXT_USART_READ_BIN(EN_IN := 1, NUMBER := 1 ,
PTR_RXDATA :=pDataObject , MAXLENGTH := 0 | xRdBinConfirm := EN_OUT, iRxDataSize :=
RXLENGTH,xERROR:= ERROR);
if xRdBinConfirm then var4:=iRxDataSize; xWaitForReceipt:=false;
```

```
end_if; END_if;
if xRdBinConfirm then
inst0_EXT_USART_WRITE_BIN(EN_IN :=0 , NUMBER :=1    | xERROR:= ERROR);
xRdBinConfirm:=false; xTransmitting:=true;
end_if;
if xTransmitting then abDataBuffer[0]:=49;
(* abDataBuffer[1]:=1; abDataBuffer[2]:=2; iRxDataSize:=3;*)
inst0_EXT_USART_WRITE_BIN(EN_IN := 1, NUMBER :=1 ,
PTR_TXDATA :=pDataObject, TXLENGTH :=iRxDataSize |    xWrBinConfirm := EN_OUT, xERROR :=
ERROR);
if xWrBinConfirm then
xTransmitting :=false; xReadStart:=true;
end_if; end_if;
end_if;
```

3.3.12 CAN_INIT



用于初始化CAN接口

操作数的定义：

BITRATE： The baud rate is in bits / second, classified as follows:

BITRATE： =0 means 1MBit/s BITRATE： =1 means 840kBit/s

BITRATE： =2 means 700kBit/s

BITRATE： =3 means 500kBit/s

BITRATE： =4 means 400kBit/s

BITRATE： =5 means 250kBit/s

BITRATE： =6 means 200kBit/s

BITRATE： =7 means 125kBit/s

BITRATE： =8 means 100kBit/s；

BITRATE： =9 means 80kBit/s

BITRATE： =10 means 50kBit/s

BITRATE： =11 means 40kBit/s

BITRATE： =12 means 20kBit/s

BITRATE： =13 means 10kBit/s AMR：Configuration value of CAN controller receive mask register （ AMR）;

Perform hardware filtering （ standard value for all CAN message processing: AMR： =16# FFFFFFFF）; ACR： The configuration value of the CAN controller receive code register （ ACR）, allowing the CAN identifier in the CAN controller

Perform hardware filtering (standard value for all CAN message processing: ACR： =16# 00000000）; NETNUMBER： network number;

ENABLE： enable or disable the input

of the function block; **CONFIRM：**

the output of the message done by

the function block;

ERROR： The error code refers to the data type ″ CAN_ERROR″, the possible error codes are defined as follows:

0: no error

1:Other errors

2:Invalid network number

3:invalid parameter

4:No message

5: Unsupported baud rate

6: Initialization failed

7: The device is busy

8: TX buffer overflow

9: No free passage

10:COBID has been registered

11: Pointer type

does not support

description:

function block CAN_INIT is used to initialize the CAN interface. This requires deactivating the CANopen functionality of this interface. If the PLC supports configuration via WEBFrontend, it is usually possible to set the "Enable Status" of the relevant interface to "Disabled". Therefore, the interface is not automatically initialized for CANopen, and the function can be implemented by calling the function block in the program through the PLC.

When entering BITRATE, the value of Bitrate must be specified in Bit/s, for example, 125000 is 125

kBit/s. The combination of the two values after the input AMR (mask register) and ACR (code register) are transferred to the AMR or ACR register of the CAN controller is an acceptance filter that can only pass CAN identifiers that meet the filter conditions

CAN message. With proper settings of AMR and ACR, all irrelevant interfering messages can be excluded from reception in the CAN controller, reducing the CPU load of the controller and preventing overflow of the receive buffer. Each CAN controller can obtain the AMR and ACR correlations and implemented filter values from their respective associated control tables. Normally, the filter receives all CAN messages as a CAN-Controller. For this, AMR and ACR must be set as follows: AMR:=16#FFFFFFFF;ACR:=16#

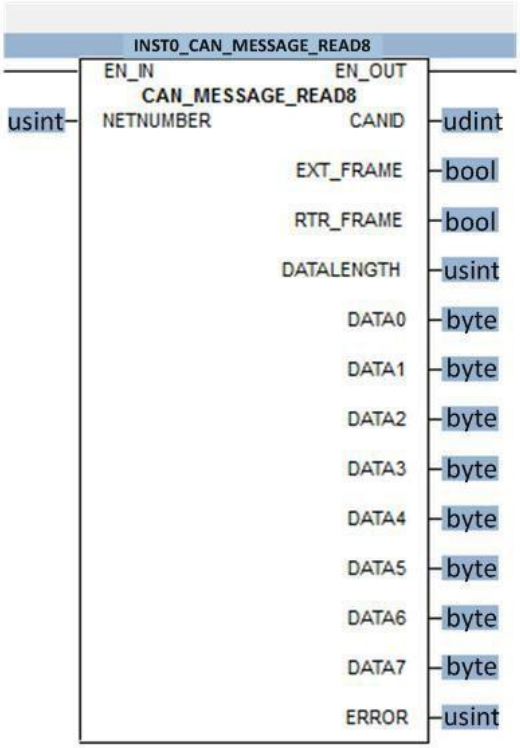
00000000;

Sample program:

```
VAR canCH:USINT;
inst0_CAN_INIT:CAN_INIT;
mCanConfirm:BOOL; mCanInit:BOOL:=false; xERROR:USINT;
END_VAR
canCH:=2;
if not mCanInit then
```

```
inst0_CAN_INIT(BITRATE :=0 , AMR :=0 , ACR :=0 , NETNUMBER :=canCH ,
ENABLE :=true |     mCanConfirm:= CONFIRM, xERROR := ERROR);
if xERROR=0 then
    mCanInit:=true;
end_if;
end_if ;
```

3.3.13 CAN_MESSAGE_READ8



用于读取接收的CAN消息（在功能块输出端传输数据）

Definition of operands:

NETNUMBER： network number;

ENABLE： Enable or disable the input of the function block; CANID： CAN identifier

EXT_FRAME： TRUE： send CAN extended frame （ 29-bit frame ID）; FALSE： send CAN standard frame （ 11-bit frame ID）; **RTR_FRAME：** TRUE： send remote frame; FALSE：

send data frame;

DATA0-DATA7：Data bytes received by CAN-Message; DATALENGTH：Length received by CAN-Message；

CONFIRM： message output done by function block;

ERROR： Error code of data type ″ CAN_ERROR″, possible error codes are defined as follows:

0: no error

1: Other errors

2: Invalid network number

3: invalid parameter

4: No message

5: Unsupported baud rate

6: Initialization failed

7: The device is busy

8: TX buffer overflow

9: No free passage

10:COBID has been registered

11: Pointer type

does not support

description:

function block CAN_MESSAGE_READ8 is used to read received CAN messages from the receive buffer of the CAN interface. Data is transferred byte by byte at the output of the function block. The displayed value of the output DATALENGTH is the number of valid data bytes (starting from DATA0).

CONFIRM is set to TRUE during a function block return , data 0 to data 7 contain a single byte of the message, and the output data length shows the number of valid bytes

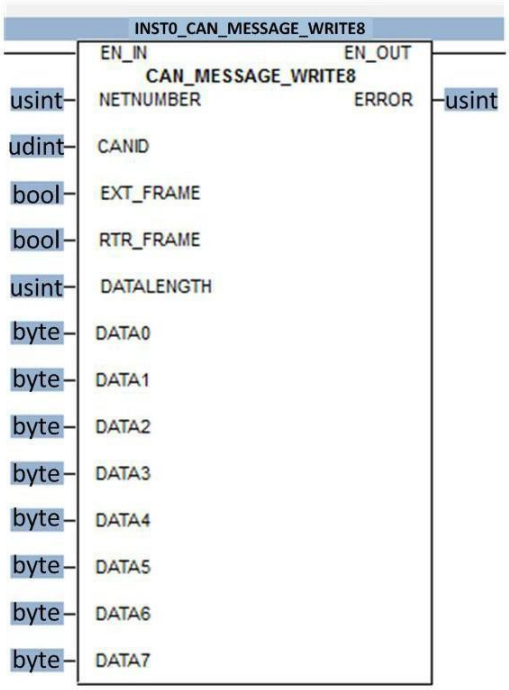
(starting at data 0), although CONFIRM has now been reset to FALSE, but the CAN interface does not contain messages, with the help of CONFIRM, it is possible to distinguish whether a valid message of length 0 bytes is received, and if no message is available, it is displayed by the error code at the output ERROR.

Sample program:

The sample program shows the combined working process of the function block CAN_MESSAGE_READ8 and the function block CAN_MESSAGE_WRITE8.

First, the routine reads CAN interface messages by calling the function block CAN_MESSAGE_READ8. After the message is completely read, it is rewritten to the interface via the function block CAN_MESSAGE_WRITE8 (refer to Section 3.3.13).

3.3.14 CAN_MESSAGE_WRITE8



用于将需要发送的CAN信息写到CAN总线上

Definition of operands:

CANID: the identifier of the CAN message to be sent ;

EXT_FRAME: TRUE: CAN identifier marks extended frame (29 bits); FALSE: CAN identifier marks standard frame (11 bits); **RTR_FRAME**: TRUE: CAN object is passed as RTR-Frame ; FALSE: CAN object is not passed as RTR-Frame ; **CHANNEL**: If the CAN object has been defined by the function block CAN_DEFINE_CANID, the channel number returned by this function block is CAN object passing. If the CAN object has function defined by CAN_DEFINE_CANID_RANGE, then 0 must be passed (note: objects defined by CAN_DEFINE_CANID_RANGE cannot be used for RTR- messages);

DATA0-DATA7： The data bytes of the CAN message to be sent ; **DATALENGTH：** The length of the CAN message to be sent ; **NETNUMBER：** The network number;

ENABLE： Enable or disable the input of the function block;

CONFIRM: message output done by function block;

ERROR: Error code of data type " CAN_ERROR", possible error codes are defined as follows:

0: no error

1: Other errors

2: Invalid network number

3: invalid parameter

4: No message

5: Unsupported baud rate

6: Initialization failed

7: The device is busy

8: TX buffer overflow

9: No free passage

10:COBID has been registered

11: Pointer type

does not support

description:

function block CAN_MESSAGE_WRITE8 is used to send CAN data. The data to be sent appears byte by byte at the input of the function block and is written in DATA0 to DATA7 as a single byte. The input DATALENGTH specifies the number of valid data bytes (starting at DATA0). When the function block CAN_MESSAGE_WRITE8 is called, the message to be sent is stored in the send buffer of the CAN interface. If no error has occurred (the message is correctly stored in the send buffer), the function block output CONFIRM is TRUE.

The following sample program shows the application of the function block CAN_MESSAGE_WRITE8. First, call the block

CAN_MESSAGE_READ8 reads data from the CAN interface. After the data is completely read, CAN_MESSAGE_WRITE8 rewrites it on the interface. The sample program integrates the process control of initializing the CAN interface and the program.

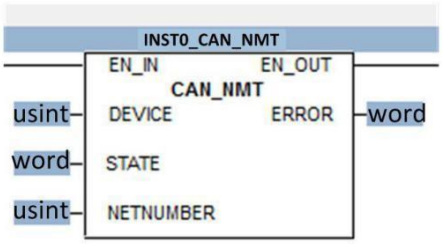
Sample program:

```
VAR canCH:USINT;
```

```
inst0_CAN_INIT:CAN_INIT;
mCanConfirm:BOOL; mCanInit:BOOL:=false; xERROR:USINT;
inst1_CAN_MESSAGE_READ8:CAN_MESSAGE_READ8;
inst2_CAN_MESSAGE_WRITE8:CAN_MESSAGE_WRITE8; mCANID:UDINT;
mEXT_FRAME:BOOL; mRTR_FRAME:BOOL; mDATA0:BYTE;
mDATA1:BYTE; mDATA2:BYTE; mDATA3:BYTE; mDATA4:BYTE; mDATA5:BYTE; mDATA6:BYTE;
mDATA7:BYTE; mDATALENGTH:USINT; mCONFIRM:BOOL; canID:UDINT;
canDAT0:Byte; canDAT1:Byte; canDAT2:Byte; canDAT3:Byte; canDAT4:Byte; canDAT5:Byte;
canDAT6:Byte; canDAT7:Byte; RecCount:INT; first:bool:=true; END_VAR


canCH:=2;
if not mCanInit then
inst0_CAN_INIT(BITRATE :=0 , AMR :=0 , ACR :=0 , NETNUMBER :=canCH ,
ENABLE :=true |     mCanConfirm:= CONFIRM, xERROR := ERROR); if xERROR=0 then
mCanInit:=true; end_if;
else
inst1_CAN_MESSAGE_READ8(
NETNUMBER :=canCH , ENABLE :=true    |    mCANID:= CANID,    mEXT_FRAME:= EXT_FRAME,
mRTR_FRAME:= RTR_FRAME, mDATA0:= DATA0,  mDATA1:= DATA1,    mDATA2:=
DATA2,
mDATA3:= DATA3,  mDATA4:= DATA4,    mDATA5:= DATA5,    mDATA6:= DATA6, mDATA7:=
DATA7,mDATALENGTH:= DATALENGTH,
mCanConfirm:= CONFIRM,
xERROR:= ERROR);
```

```
if mCanConfirm = 1 then canID: = mCANID; canDAT0: = mDATA0; canDAT1: = mDATA1; canDAT2: = mDATA2;
canDAT3: = mDATA3; canDAT4: = mDATA4; canDAT5: = mDATA5; canDAT6: = mDATA6; canDAT7: = mDATA7;

RecCount: = RecCount+1; end_if;

if mCanConfirm = 1 then mCANID: = mCANID+1;

inst2_CAN_MESSAGE_WRITE8(CANID := mCANID, EXT_FRAME := mEXT_FRAME, RTR_FRAME :=
mRTR_FRAME, CHANNEL := 1, DATA0 :=mDATA0 , DATA1 :=mDATA1 , DATA2 :=mDATA2 , DATA3 :=mDATA3 ,
DATA4 :=mDATA4 , DATA5 :=mDATA5 , DATA6 :=mDATA6 , DATA7 :=mDATA7 ,
DATALENGTH :=mDATALENGTH , NETNUMBER :=canCH , ENABLE := true |     mCONFIRM:= CONFIRM,

xERROR:= ERROR);

end_if;

(*

if first=true then

inst2_CAN_MESSAGE_WRITE8(CANID := 2, EXT_FRAME :=0, RTR_FRAME :=0,

CHANNEL := 1, DATA0 :=12 , DATA1 :=13 , DATA2 :=14 , DATA3 :=15 ,

DATA4 : =16 , DATA5 :=17 , DATA6 :=18 , DATA7 :=19 , DATALENGTH :=8 , NETNUMBER :=canCH ,
ENABLE := true |     mCONFIRM:= CONFIRM,xERROR:=

ERROR);

first:=false;

end_if;

*)

end_if;
```

3.3.15 CAN_NMT



用于发送NMT命令（网络管理命令）的功能块

Definition of operands:

ENABLE： Enable or disable the input of the function block;

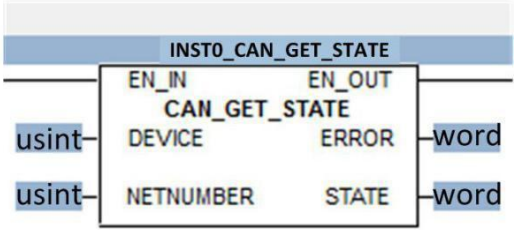
DEVICE： Node address to be controlled (1-127 or 0, 0 means to control all nodes)；**NETNUMBER：** Network number;

STATE： node state;

CONFIRM： The completed message is output by the function block, FALSE indicates that the transfer was not successfully completed or terminated after an error, TRUE indicates that the transfer was successfully completed;

ERROR： The error code describes information about the result of the execution of the function block.

3.3.16 CAN_GET_STATE



用于请求设备节点状态的功能块

Definition of operands:

ENABLE： Enable or disable the input of the function block;

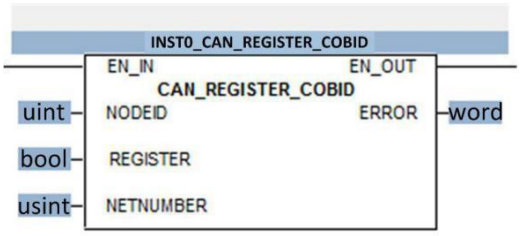
DEVICE： The node address that needs to be requested (1-127 or 0, 0 represents requesting all nodes);

NETNUMBER： network number;

CONFIRM： The completed message is output by the function block, FALSE indicates that the transfer was not successfully completed or terminated after an error, TRUE indicates that the transfer was successfully completed;

STATE： The current node state returned.

3.3.17 CAN_REGISTER_COBID



用于注册或删除通过网络层接收的PDO和CAN第2层消息

Definition of operands:

EN_IN： Enable or disable the input of the function block;

NODEID： COBID (CAN Identifier) of new information to be entered

into or deleted from the registry **REGISTER：** TRUE= enter COBID into

registry; FALSE= delete COBID from registry ;

NETNUMBER： network number (Note: if the PLC only supports one CANopen interface, this input can be skipped, because the variable value has been set to the initial value 0 according to the IEC61131 standard ;

EN_OUT： function block output

job completion message; **ERROR：**

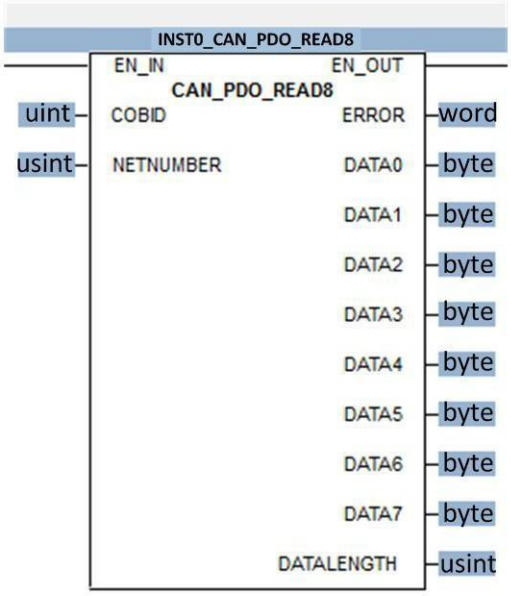
function block execution error

alarm information Description:

This function block can be used to register a PDO or CAN layer 2 message for reception by the network layer, or to delete such a registration . When the input REGISTER is 1, the function block is called, and the specified COBID of the message received

by the network layer will be registered. When the input REGISTER is 0, the function block is called, and the registry information of the corresponding COBID will be deleted. When REGISTER is 0 and COBID is 0, calling the function block will delete all registration information and all information stored in the network layer buffer. In fact, the network layer only supports accessing PDO and CAN layer 2 messages and registering the information by calling the function block CAN_PDO_READ8.

3.3.18 CAN_PDO_READ8



用于读取PDO数据的功能块

Definition of operands:

ENABLE： Enable or disable the input of the function block;

COBID： COBID of the PDO data to be read (ie

CAN frame ID)； **NETNUMBER：** network number;

DATA0-DATA7： received CAN data;

DATALENGTH： received CAN data

length; **ERRORINFO：** reserved;

CONFIRM： The completed message is output by the function block, FALSE indicates that the transfer was not successfully completed or terminated after an error, TRUE indicates that the transfer was successfully completed;

ERROR： The error code describes the information about the execution result of the function block ,the possible error codes are defined as follows:

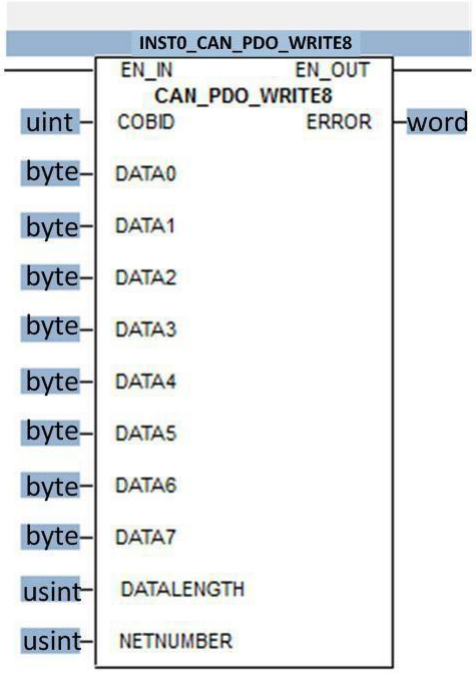
16#0000(= 00 dec)： no errors

16#0001(= 01 dec)： other errors

16#0002(=	02	dec): data overflow
16#0003(=	03	dec): timeout
16#0010(=	16	dec): CAN bus off
16#0011(=	17	dec): CAN bus passive error
16#0021(=	33	dec): reserved (error)
16#0022(=	34	dec): invalid function block
16#0023(=	35	dec): no master mode
16#0024(=	36	dec): invalid device
16#0025(=	37	dec): forwarding congestion (transfer busy)
16#0030(=	48	dec): no SDO channel available
16#0031(=	49	dec): SDO congestion
16#0032(=	50	dec): SDO initialization error
16#0033(=	51	dec): wrong SDO length
16#0034(=	52	dec): SDO errors (ERRORINFO SDO in abort code)
16#0040(=	64	dec): no valid data
16#0041(=	65	dec): this COBID is already registered
16#0042(=	66	dec): no COBID table entry available

16#0043(=	67	dec)：No such COBID is registered
16#0044(=	68	dec)：no receive channel available
16#0045(=	69	dec)：CAN data length shall not be 0

3.3.19 CAN_PDO_WRITE8



用于发送PDO数据的功能块

Definition of operands:

ENABLE： Enable or disable the input of the function block; **COBID：** The COBID (ie CAN frame ID) of the PDO message to be sent ; **DATA0-DATA7：** The data to be sent; **DATALENGTH：** The length of the CAN data to be sent ; **NETNUMBER：** The network

number;

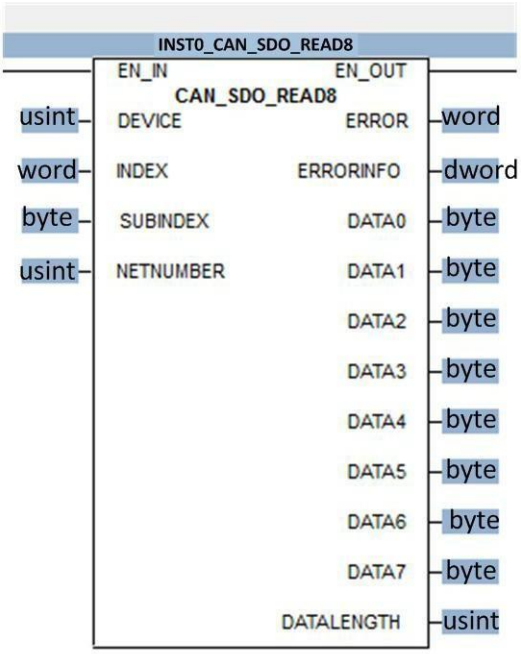
ERRORINFO： reserved;

CONFIRM： output completed message through the function block, FALSE indicates that the transfer was not completed successfully or terminated after an error, TRUE indicates that

The transfer completed successfully;

ERROR： The error code describes information about the result of the execution of the function block.

3.3.20 CAN_SDO_READ8



通过SDO（服务数据对象）传输，读对象字典功能块

Definition of operands:

ENABLE： Enable or disable the input of the function block;

DEVICE： Node address to be read (1-127 or 0， 0 means to write all nodes)；**SUBINDEX：** Sub-index item to be read;

INDEX： Index item to be read; **NETNUMBER：**

Network number;

DATA0-DATA7： data to be read;

DATALENGTH： CAN data length to be read;

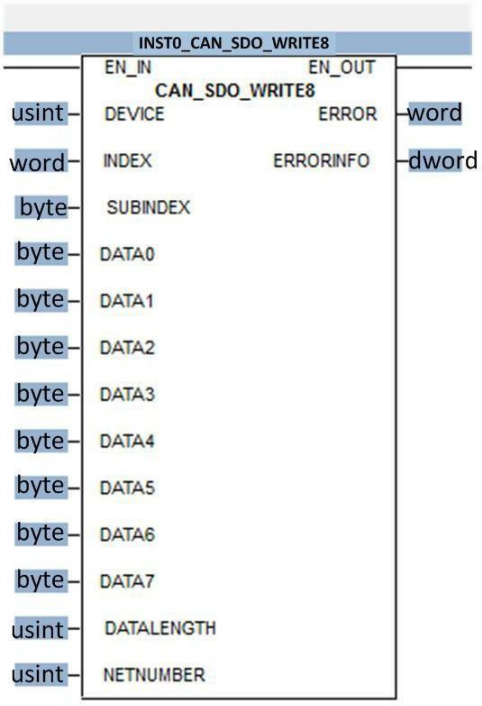
ERRORINFO **：**SDO abort code;

CONFIRM：message completed by function block output, FALSE indicates that the transfer was not completed successfully or terminated after an error, TRUE

Indicates that the transfer was completed successfully;

ERROR： The error code describes information about the result of the execution of the function block.

3.3.21 CAN_SDO_WRITE8



通过SDO（服务数据对象）传输，写对象字典功能块

Definition of operands:

ENABLE： Enable or disable the input of the function block;

DEVICE： The node address to be written (1-127 or 0， 0 means to write all nodes)；

SUBINDEX： Sub-index item to

be written; **INDEX：** Index item

to be written; **NETNUMBER：**

Network number;

DATA0-DATA7： data to be written;

DATALENGTH： data length to be

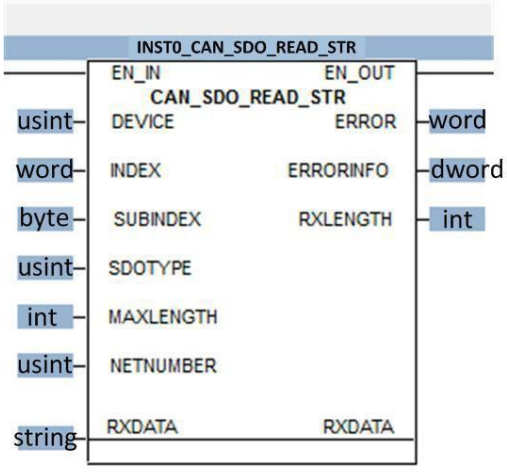
written; **ERRORINFO：** SDO abort code;

CONFIRM： output completed message via the function block, FALSE to indicate that the transfer did not complete successfully or to terminate after an error, TRUE to say

Indicates that the transmission was successfully completed;

ERROR： The error code describes information about the result of the execution of the function block.

3.3.22 CAN_SDO_READ_STR



用于通过SDO传输读取节点对象字典中的字符串信息

Definition of operands:

EN_IN： Enable or disable the input of the function block;

DEVICE： The node address to be read (local OD is 1-127 or 0); **INDEX：** The main index number to be read;

SUBINDEX： the subindex number to read;

SDOTYPE： The type of SDO transmission mode determined according to the data type; Note: If the input has no clear value (no input or unused), the network layer will automatically select the most appropriate SDO transmission mode according to the amount of transmitted data.

MAXLENGTH： The limit of the number of characters to be read; if the value is 0, the buffer length of the transmission string is automatically determined by the system ,(Note: The standard buffer length of strings in OpenPCS is 32 characters).

NETNUMBER： network number;

RXDATA： String variable used to receive read

characters; **EN_OUT：** Function block function

completion output message;

ERROR： The error code describes information about the result of the execution of the function block.

ERRORINFO： SDO termination code of

the communication partner .

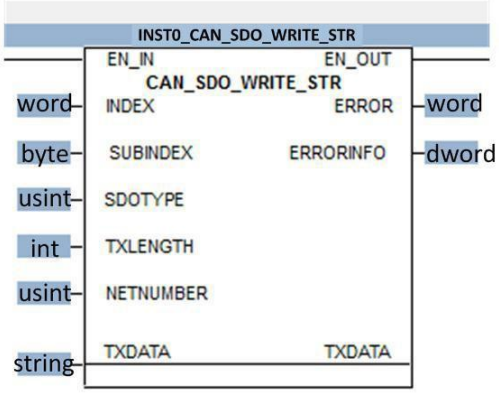
RXLENGTH： the length of the

string information;

describe:

The function block uses the SDO transfer of strings read from the node's object dictionary, which always happens in the background. Therefore, the program described in section 3.1.3 must synchronize the function block and the PLC program through the parameters ENABLE and CONFIRM . If the output CONFIRM is 1 when the function block is executed , the string transmitted as element RXDATA contains the string characteristics of the input object. The output RXLENGTH indicates the number of characters read (equal to LEN (RXDATA)), each network layer can only support a limited number of SDO transfers through the PLC program (up to 5 transfers in standard cases , if there is no input pin in the PLC manual make any changes), after the SDO transfer is started by setting ENABLE to 1 , the related SDO channel will be closed and will not be used by other function blocks. This closed state continues until the SDO function block is called again by setting ENABLE to 0 . . If you skip calling the function block with ENABLE set to 0 , the resource will remain closed and will not be used by the PLC . You can access the PLC local object dictionary by calling the function block with DEVICE set to 0. The same method can also be used to read Your local OD value.

3.3.23 CAN_SDO_WRITE_STR



用于通过SDO传输将字符串发送到相应节点的对象字典中

Definition of operands:

EN_IN： function block enable

input; **INDEX：** the main index

number of the data to be sent;

SUBINDEX： the sub-index number of the data to be sent;

SDOTYPE： SDO transmission mode used according to data type ″ CAN_SDO_Type ″ , note: if not explicitly specified to

The value of this input (input open or unused), the function block uses the default mode, and the network layer selects the most appropriate SDO transmission method according to the amount of data to be transmitted.

TXLENGTH： The number of characters to write, if the number is 0, the length of the object characters contained in the string TXDATA is determined internally (equal to LEN (TXDATA)) and is used as the number of characters to write.

NETNUMBER： Network number (Note: If the PLC only supports one CANopen interface, the setting of this input can be skipped, because according to the IEC61131 standard, the value of the variable has been set to the initial value of 0.

TXDATA： The string to be passed.

EN_OUT： Output message after function block execution is complete.

ERROR： Error code generated according to data type

CIA405_CANOPEN_KERNEL_ERROR . **ERRORINFO：** The communication object

SDO termination code generated according to the data type

CIA405_SDO_ERROR . describe:

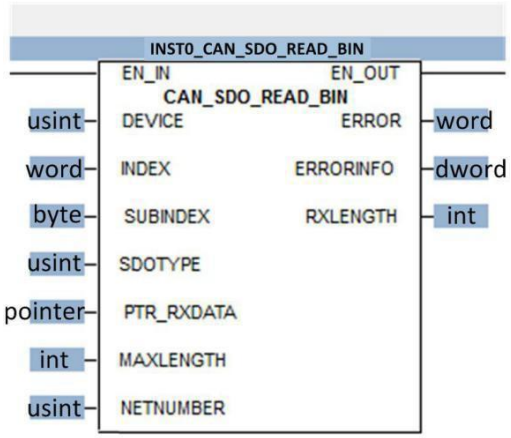
function block CAN_SDO_WRITE_STR is used to write a string to the object dictionary of the node via SDO transfer.

SDO transfers usually take place in the background, so the program described in section 4.1.3 must synchronize the function block and the PLC program by using ENABLE and CONFIRM, the strings written in the object dictionary must be transferred into the element TXDATA.

Enter TXLENGTH for a valid number of characters. If the value of this entry is 0, the number of characters contained in the string TXDATA is determined internally and used as the number of characters to be written. In this case, what the string contains will all be written. The network layer can only support a limited number of SDO transmissions at one time through the PLC program (standard: if the PLC manual has not changed the input items, up to 5 transmissions), after setting ENABLE to 1 to start SDO

transmission, each SDO channel will be It is closed and will not be used by other function blocks. This closed state will continue until the function block is called again through the input ENABLE as 0 before opening. If you skip calling the function block with ENABLE set to 0, the resource will be closed forever and will no longer be used by the PLC. The PLC's local object dictionary can be accessed by calling the function block with DEVICE equal to 0. This way can also be used to write values into your own object dictionary.

3.2.24 CAN_SDO_READ_BIN



功能块通过SDO传输读取节点对象字典中的二进制数据

Definition of operands:

EN_IN： function block enable input

DEVICE： Node address read (1 to 127 or local object dictionary 0)**INDEX：** Primary index number read

SUBINDEX： the subindex number read

SDOTYPE： The SDO transmission mode used (refer to the data type CAN_SDO_TYPE), note: if this input has no clear value (the input is open or not used), the function block will use the SDO type automatic allocation mode. The network layer will automatically select the most appropriate SDO transmission mode based on the amount of transmitted data.

PTR_RXDATA： The object address of the read data byte

MAXLENGTH： The limit on the number of bytes read, if the value is 0, the length of the object at the address defined by PTR_RXDATA is defined internally and used as the number of characters read (the maximum number of bytes read is equal to the number of bytes occupied by the object bytes)

NETNUMBER： Network number (Note：If the PLC only supports 1 CANopen interface, the setting of this item can be skipped, because according to the IEC61131 standard,

the value of the variable has been set to the initial value 0).

EN_OUT： output function block execution result

ERROR： Refer to the error code generated by the data type CIA405_CANOPEN_KERNEL_ERROR . **ERRORINFO：** Communication object SDO interrupt code generated according to data type CIA405_SDO_ERROR .

RXLENGTH： The number of

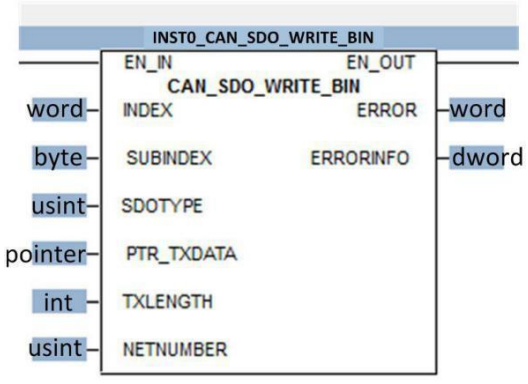
data bytes read Description:

function block CAN_SDO_READ_STR is used to read the binary data in the node object dictionary through SDO transmission.

SDO transfers usually take place in the background, so the program described in Section 4.1.3 must synchronize the function block and PLC program by using ENABLE and CONFIRM. If the output CONFIRM is 1 after the execution of the function block, the object whose address is defined by PTR_RXDATA contains the binary data of the object to be read, and the output RXLENGTH indicates the number of bytes read. The network layer can only support a limited number of SDO transmissions at one time through the PLC program (standard: if the PLC manual has not changed the input items, up to 5 transmissions), after setting ENABLE to 1 to start SDO transmission, each SDO channel will be Closed, not used by other function blocks, this closed state will continue until the function block passes the input again

When ENABLE is 0, it will be turned on. If you skip calling the function block with ENABLE set to 0, the resource will be closed forever and will no longer be used by the PLC. The PLC's local object dictionary can be accessed by calling the function block with DEVICE equal to 0. This way can also be used to write values into your own object dictionary.

3.3.25 CAN_SDO_WRITE_BIN



功能块用于通过SDO传输将二进制数据写入节点的对象字典中

Definition of

operands: **EN_IN：**

function block enable

input **INDEX：** main

index number to be

written

SUBINDEX： the subindex number to write

SDOTYPE： The SDO transmission mode used (refer to the data type CAN_SDO_TYPE), note: if this input has no clear value (input open or not used), the function block will use the SDO type automatic allocation mode. The network layer will

Quantity automatically selects the most suitable SDO transfer mode.

PTR_TXDATA： Object address of binary data to be written

TXLENGTH： The number of bytes to write, if the value is 0, the object length at the address defined by PTR_TXDATA is determined internally and used as the number of bytes to write.

NETNUMBER： Network number (Note：If the PLC only supports 1 CANopen interface, the setting of this item can be skipped, because according to the IEC61131 standard, the value of the variable has been set to the initial value 0).

EN_OUT： output function block execution result

ERROR： Refer to the error code generated by the data type

CIA405_CANOPEN_KERNEL_ERROR. **ERRORINFO：** Communication object SDO

interrupt code generated according to data type CIA405_SDO_ERROR.

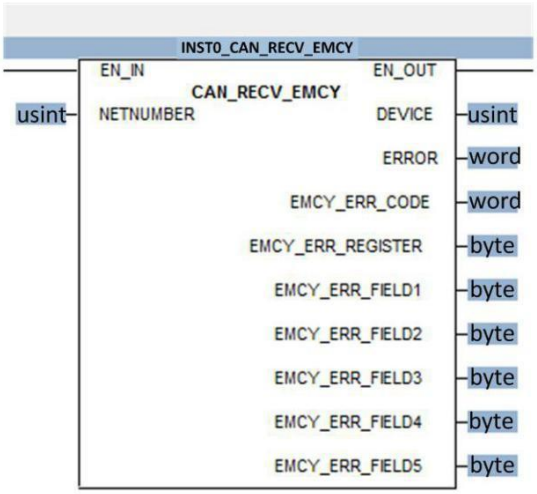
describe:

function block CAN_SDO_WRITE_STR writes binary data into the node object dictionary via SDO transmission. SDO transfers usually take place in the background, so the program described in Section 3.1.3 must synchronize the function block and PLC program by using ENABLE and CONFIRM.

The address of the object where the object dictionary of the binary data to be written must be passed to the parameter PTER_TXDATA. The input TXLENGTH specifies a valid number of bytes. If the value is 0, the length of the object will be defaulted and used as the bytes to be written. number. The network layer can only support a limited number of SDO transmissions at one time through the PLC program (standard: if the PLC manual has not changed the input items, up to 5 transmissions), after setting ENABLE to 1 to start SDO transmission, each SDO channel will be It is closed and will not be used by other function blocks. This closed state will continue until the function block is called again through the input ENABLE as 0 before opening. If you skip calling the function block with ENABLE set to 0, the resource will be closed forever and will no

longer be used by the PLC. The PLC's local object dictionary can be accessed by calling the function block with DEVICE equal to 0.This way can also be used to write values into your own object dictionary.

3.3.26 CAN_RECV_EMCY



用于读取网络层接收缓冲区中的任意节点紧急报文（Emergency）的功能块

Definition of operands:

ENABLE： Enable or disable the

input of the function block;

NETNUMBER： Network number;

DEVICE： The node address (1-127) that receives the emergency message（ Emergency）;

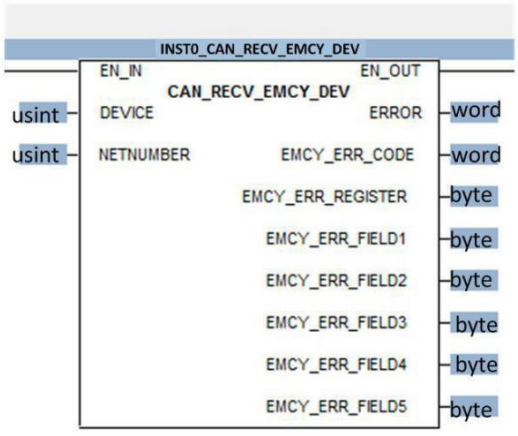
EMCY_ERR_CODE EMCY_ERR_REGISTER

EMCY_ERR_FIELD1-EMCY_ERR_FIELD5：urgent error message;

CONFIRM： The completed message is output by the function block, FALSE indicates that the transfer was not successfully completed or terminated after an error, TRUE indicates that the transfer was successfully completed;

ERROR： The error code describes information about the result of the execution of the function block.

3.3.27 CAN_RECV_EMCY_DEV



用于从网络层接收缓冲区读取特定节点紧急报文（Emergency）的功能块

Definition of operands:

ENABLE： Enable or disable the input of the function block;

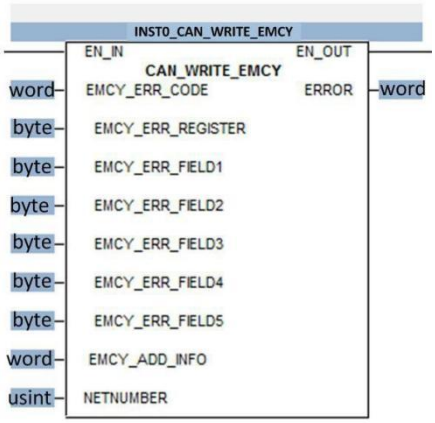
DEVICE： Node address to be controlled (1-127 or 0, 0 means to control all nodes)；**NETNUMBER：** Network number; EMCY_ERR_CODEEMCY_ERR_REGISTER

EMCY_ERR_FIELD1EMCY_ERR_FIELD5： urgent error message;

CONFIRM： The completed message is output by the function block, FALSE indicates that the transfer was not successfully completed or terminated after an error, TRUE indicates that the transfer was successfully completed;

ERROR： The error code describes information about the result of the execution of the function block.

3.3.28 CAN_WRITE_EMCY



通过网络层，发送特定应用程序紧急报文（Emergency）的功能块

Definition of operands:

ENABLE： Enable or disable the input of the function block;

NETNUMBER： network number; EMCY_ERR_CODE EMCY_ERR_REGISTER

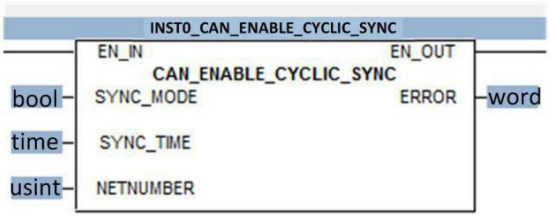
EMCY_ERR_FIELD1 - EMCY_ERR_FIELD5： urgent error message;

EMCY_ADD_INFO： Additional user-specific urgent error information, note that it is not part of the urgent error message that needs to be sent, but is used for diagnostic purposes, so it can be 0.

CONFIRM： The completed message is output by the function block, FALSE indicates that the transfer was not successfully completed or terminated after an error, TRUE indicates that the transfer was successfully completed;

ERROR： The error code describes information about the result of the execution of the function block.

3.3.29 CAN_ENABLE_CYCLIC_SYNC



功能块用于激活循环SYNC同步信息

Definition of operands:

EN_IN： function block enable input

SYNC_MODE： TRUE= activates the generation of cyclic SYNC synchronization messages ,

FALSE= deactivates the generation of cyclic SYNC synchronization messages

SYNC_TIME： time interval between two consecutive SYNC synchronization messages or 0 means

that SYNC synchronization messages are sent after each PLC scan cycle

step message.

NETNUMBER： Network number (Note：If the PLC only supports 1 CANopen interface, the setting of this item can be skipped, because according to the IEC61131 standard, the value of the variable has been set to the initial value 0).

EN_OUT： output function block execution result

ERROR： Refer to the error code generated by CIA405_CANOPEN_KERNEL_ERROR.

describe:

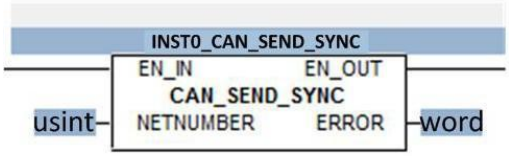
function block CAN_ENABLE_CYCLIC_SYNC is used to activate or deactivate the cyclic SYNC synchronization message. When activated, if the time interval between the last SYNC synchronization message and the current time is greater than the input SYNC_TIME , the control system will generate a SYNC synchronization message between two consecutive PLC cycles. , if the time interval between the last SYNC synchronization message and the present is less than the input SYNC_TIME, no SYNC synchronization message will be generated. An input value of 0 for SYNC_TIME will cause the control system to end each PLC scan cycle with a SYNC synchronization message . In this case, the information exchange in the process image area of the network layer will be carried out in the PLC's local input and output process image areas at the same time .

At the end of each PLC scan cycle, the control system checks the time details to send a SYNC synchronization message, so the time specified by the input SYNC_TIME is the minimum interval between two consecutive SYNC synchronization messages. In the system with the largest error, the two The actual time interval of consecutive SYNC synchronization messages varies with the time of the PLC scan cycle :

SYNC actual time := SYNC_TIME input time + PLC scan cycle

This function block can only be used in the CANopen master control system and can be used as a replacement function block for the function block CAN_SEND_SYNC.

3.3.30 CAN_SEND_SYNC



用于发送独立的同步报文（SYNC）的功能块

Definition of operands:

ENABLE： Enable or disable the

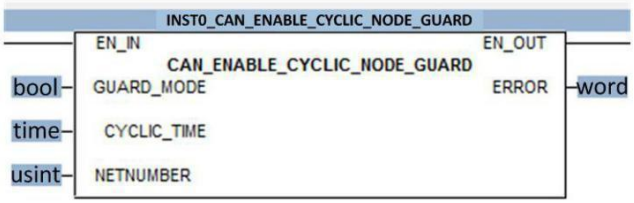
input of the function block;

NETNUMBER： Network number;

CONFIRM： The completed message is output by the function block, FALSE indicates that the transfer was not successfully completed or terminated after an error, TRUE indicates that the transfer was successfully completed;

ERROR： The error code describes information about the result of the execution of the function block.

3.3.31 CAN_ENABLE_CYCLIC_NODE_GUARD



用于激活或停用循环同步报文（SYNC）的功能块

Definition of operands:

ENABLE： Enable or disable the

input of the function block;

NETNUMBER： Network number;

SYNC_MODE： TRUE= activate cyclic synchronization message, FALSE=

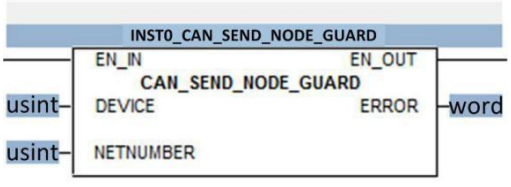
disable cyclic synchronization message; **SYNC_TIME：** time interval

between two consecutive synchronization messages;

CONFIRM： The completed message is output by the function block, FALSE indicates that the transfer was not successfully completed or terminated after an error, TRUE indicates that the transfer was successfully completed;

ERROR： The error code describes information about the result of the execution of the function block.

3.3.32 CAN_SEND_NODE_GUARD



使能CANopen从站发送节点保护

Definition of operands:

EN_IN： function block enable

input; **DEVICE：** CANopen slave node

number; **NETNUMBER：** PLC CANopen

port number; EN_OUT **：function**

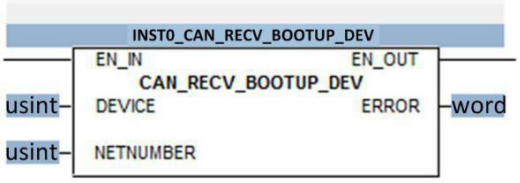
block enable output;

ERROR： The error code describes information about the result of the execution of the function block;

describe:

This function block is used to enable CANopen slave to send node protection message.

3.3.33 CAN_RECV_BOOTUP_DEV



用于从网络层的接收缓冲区中，读取特定节点Bootup消息的功能块

Definition of operands:

ENABLE： Enable or disable the

input of the function block;

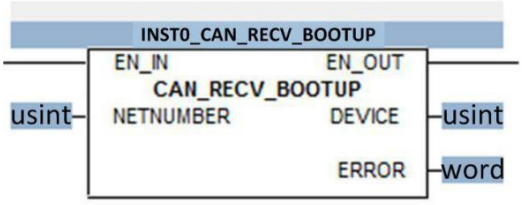
NETNUMBER： Network number;

DEVICE： Node number (1-127) used to check Bootup message reception ;

CONFIRM： The completed message is output by the function block, FALSE indicates that the transfer was not successfully completed or terminated after an error, TRUE indicates that the transfer was successfully completed;

ERROR： The error code describes information about the result of the execution of the function block.

3.3.34 CAN_RECV_BOOTUP



用于从网络层的接收缓冲区中，读取总线节点Bootup消息的功能块

Definition of operands:

ENABLE： Enable or disable the input of the function block;

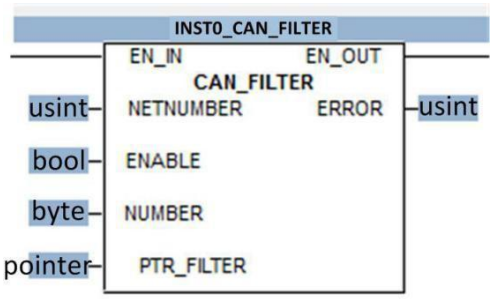
NETNUMBER： network number;

DEVICE： The address of the node that received the Bootup message (1-127)；

CONFIRM： The completed message is output by the function block, FALSE indicates that the transfer was not successfully completed or terminated after an error, TRUE indicates that the transfer was successfully completed;

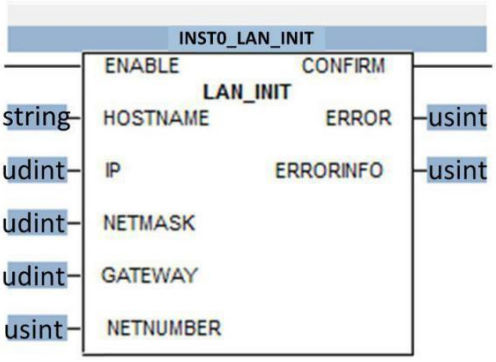
ERROR： The error code describes information about the result of the execution of the function block.

3.3.35 CAN_FILTER



用于CAN总线滤波

3.3.36 LAN_INIT



用于初始化以太网接口

Definition of operands:

ENABLE： enable or disable the function block,

true is enabled, false is disabled; **HOSTNAME：** set

the host name, for example: ’ PLCCore ’;

IP： Specify the IP address of the Ethernet side , for example: ’ 192.168.1.30’; **NETMASK：** Specify the subnet mask of the Ethernet side, for example: ’ 255.255.255.0’;

GATEWAY： Specifies the gateway on the Ethernet

side, for example: ’ 192.168.1.1’; **NETNUMBER：**

Network number;

CONFIRM： false means the function block execution fails, true means the function block execution succeeds;

ERROR： The error code describes information about the result of the execution of the function block. Possible error codes are defined as follows:

0:No error occurred while executing the function block

1:A hardware error occurred while executing the function block

2:The entered host name is not supported

4:The entered IP address is not supported

8:The filled subnet mask is not supported

16:The entered gateway does not support

32:The selected Ethernet

interface does not support

ERRORINFO： Secondary

error message. describe:

The function block initializes the Ethernet interface with the specified parameters. If an error occurs during execution, the function block will display it through the output ERROR, and the error code can be found in Table 15. The following sample program shows the application of the function block LAN_INIT to initialize the Ethernet interface, setting the following parameters: IP address 192.168.1.30, subnet mask 255.255.255.0, gateway 192.168.1.1.

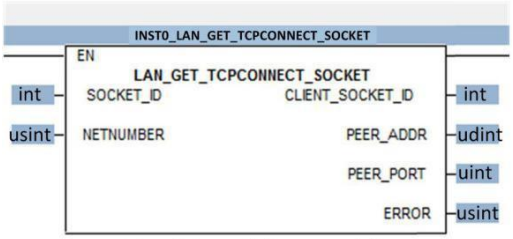
程序

VAR

```
lanInit: bool: = false; inst0_LAN_INIT: LAN_INIT;
mConfirm: bool; mError: usint; mErrorinfo: usint; mIP: udint; mNetMask: udint; mGateWay: udint;
mSocket: int;
END_VAR
if lanInit = false then
mIP: = LAN_ASCII_TO_INET ('192.168.1.30'); mNetMask: = LAN_ASCII_TO_INET ('255.255.255.0');
```

```
mGateWay: = LAN_ASCII_TO_INET ('192.168.1.0');
    inst0_LAN_INIT (ENABLE: = true, HOSTNAME: = 'PLCCore', IP: = mIP, NETMASK: = mNetMask,
GATEWAY: = mGateWay, NETNUMBER: = 1
    | mConfirm:= CONFIRM,    mError:= ERROR, mErrorinfo:= ERRORINFO); lanInit:=true;
    end_if ;
```

3.3.37 LAN_GET_TCPCONNECT_SOCKET



用于建立TCP端的socket连接

Definition of Operands

SOCKET_ID： generated socket index

value; **NETNUMBER：** network number;

CLIENT_SOCKET_ID: peer socket index

value;

PEER_ADDR: The address of the peer, which can be

obtained when the peer is connected; **PEER_PORT:** The

port number of the peer, which can be obtained when the

peer is connected;

ERROR： The error code describes information about the result of the execution of the function block. Possible error codes are defined as follows:

0：No error occurred while executing the function block

1：A hardware error occurred

while executing the function

block 2：The inserted socket

index value is not supported

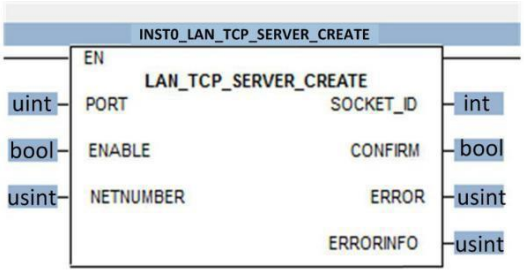
4：The selected Ethernet

interface is not supported

describe:

The function block initializes the Ethernet interface with the specified parameters. If an error occurs during execution, the function block will display it through the output ERROR, and the error code can be found in Table 16. The following sample program shows the application of the function block LAN_INIT to initialize the Ethernet interface, setting the following parameters: IP address 192.168.1.30, subnet mask 255.255.255.0, gateway 192.168.1.1.

3.3.38 LAN_TCP_SERVER_CREATE



用于初始化以太网TCP连接的SERVER端

Definition of operands:

PORT： the Ethernet port number to use;

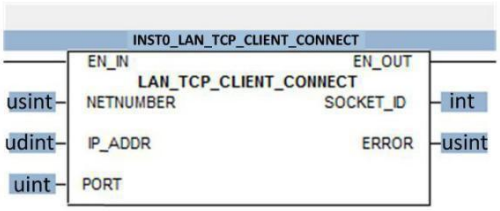
ENABLE： enable or disable the function block, true is enabled, false is disabled; **NETNUMBER：** network number;

SOCKET_ID： the generated socket index value;

CONFIRM： false means the function block execution fails, true means the function block execution succeeds; **ERROR：** the error code describes the information about the function block execution result.

ERRORINFO： Secondary error information.

3.3.39 LAN_TCP_CLIENT_CONNECT



用于建立TCP_CLIENT连接

Definition of operands:

EN_IN： function block

enable input;

NETNUMBER： network

number; **IP_ADDR：** IP

address; **PORT：** port

number;

EN_OUT： function block

enable output; **SOCKET_ID：**

SOCKET connection ID；

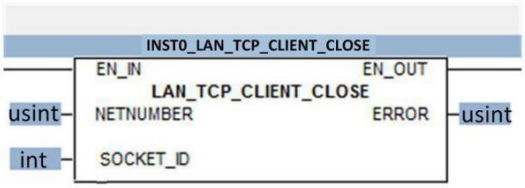
ERROR： The error code describes the

information about the result of the execution

of the function block Description:

This function block is used to create a socket connection of TCP_CLIENT.

3.3.40 LAN_TCP_CLIENT_CLOSE



用于关闭TCP_CLIENT连接

Definition of operands:

EN_IN： function block enable

input; **NETNUMBER：** network

number; **SOCKET_ID：** SOCKET

connection ID；

EN_OUT： function block enable output;

ERROR： The error code describes information

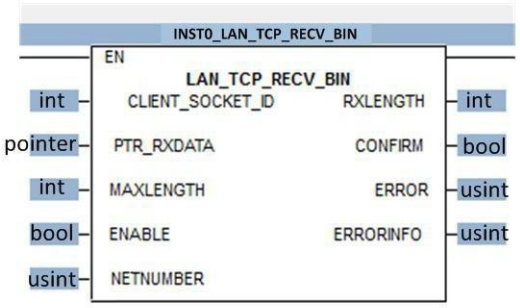
about the result of the execution of the

function block; description:

This function block is used to close the socket connection of TCP_CLIENT.

When this function block is executed, the socket connection needs to be opened in advance through the function block LAN_TCP_CLIENT_CONNECT.

3.3.41 LAN_TCP_RECV_BIN



用于从TCP端口读取二进制字符流

Definition of operands:

CLIENT_SOCKET_ID： The index value of the peer socket; PTR_RXDATA：**The address** used to receive the read byte object;

MAXLENGTH： limit of bytes to read, if 0, the length of the object is determined by PTR_RXDATA **ENABLE：** enable or disable the function block, true：enable, false：disable;

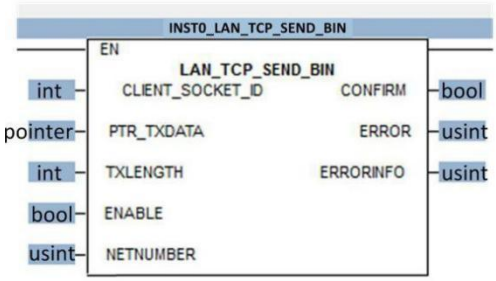
NETNUMBER： network number;

RXLENGTH： number of characters read (if ERROR：=0);

CONFIRM： false：function block execution failed;

true：function block execution succeeded; **ERROR：** error code explaining information about the function block execution result; **ERRORINFO：** secondary error information .

3.3.42 LAN_TCP_SEND_BIN



用于将二进制字符流写入TCP端口

Definition of operands:

CLIENT_SOCKET_ID: The index value of the peer

socket; PTR_RXDATA :**The address** used to

receive the read data byte object;

TXLENGTH： The number of characters to write, if the value is 0, the length of the characters and the characters TXDATA contained in the string are determined internally (equal to LEN (TXDATA))

ENABLE： enable or disable the function block,

true : enable, false : disable; **NETNUMBER：** network

number;

CONFIRM： false : the function block execution failed,

true : the function block execution succeeded; **ERROR：**

the error code explains the information about the

execution result of the function block; **ERRORINFO：**

secondary error information;

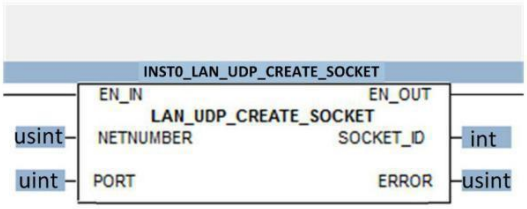
Sample program:

```
VAR
lanInit:bool:=false; inst0_LAN_INIT:LAN_INIT;
mConfirm:bool; mError:usint;mErrorinfo:usint; mIP:udint;mNetMask:udint;
mGateWay:udint;mSocket:INT; inst4_LAN_TCP_SERVER_CREATE:LAN_TCP_SERVER_CREATE;
mTcpCreateOK:bool:=false; mClientSocket:INT; mPeer:udint; mPeerPort:uint ;
inst5_LAN_GET_TCPCONNECT_SOCKET:LAN_GET_TCPCONNECT_SOCKET;
inst6_LAN_TCP_RECV_BIN:LAN_TCP_RECV_BIN; inst7_LAN_TCP_SEND_BIN:LAN_TCP_SEND_BIN;
mRxLen:int;
```

```
abDataBuffer : ARRAY[0..1000] OF BYTE;

pDataObject : POINTER;

END_VAR


if lanInit=false then
```

```
mIP := LAN_ASCII_TO_INET('192.168.1.30'); mNetMask := LAN_ASCII_TO_INET('255.255.255.0');
mGateWay := LAN_ASCII_TO_INET('192.168.1.0');
inst0_LAN_INIT(ENABLE :=true , HOSTNAME := 'PLCCore',IP := mIP,
NETMASK :=mNetMask,GATEWAY :=mGateWay ,NETNUMBER:=1
|mConfirm:= CONFIRM,mError:= ERROR, mErrorinfo:= ERRORINFO); lanInit:=true;
else
if mTcpCreateOK=false then
inst4_LAN_TCP_SERVER_CREATE(PORT :=8089 ,ENABLE := true, NETNUMBER :=
1|
mSocket:= SOCKET_ID, mConfirm:= CONFIRM, mError:= ERROR,      mErrorinfo:= ERRORINFO);
mTcpCreateOK:=true; end_if;
end_if; pDataObject:=&abDataBuffer; if mTcpCreateOK=true then
inst5_LAN_GET_TCPCONNECT_SOCKET(SOCKET_ID := mSocket,
NETNUMBER := 1| mClientSocket:= CLIENT_SOCKET_ID, mPeer:= PEER_ADDR, mPeerPort:=
PEER_PORT, mError:= ERROR);
if mClientSocket>=0 then
inst6_LAN_TCP_RECV_BIN(CLIENT_SOCKET_ID :=mClientSocket , PTR_RXDATA :=pDataObject ,
MAXLENGTH := 1500, ENABLE :=1 , NETNUMBER := 1|mRxLen:= RXLENGTH, mConfirm:= CONFIRM,
mError:= ERROR,
mErrorinfo:= ERRORINFO); if mRxLen>0 then
inst7_LAN_TCP_SEND_BIN(CLIENT_SOCKET_ID :=mClientSocket , PTR_TXDATA :=pDataObject ,
TXLENGTH := mRxLen, ENABLE :=1 ,
NETNUMBER :=1| mConfirm:= CONFIRM, mError := ERROR,      mErrorinfo:= ERRORINFO);
end_if; end_if;
end_if;
```

3.3.43 LAN_UDP_CREATE_SOCKET



可建立一个UDP的socket连接，用于发送和接收数据

Definition of operands:

PORT： the Ethernet port number to be used;

ENABLE： enable or disable the function block,

true is enabled, false is disabled; **NETNUMBER：**

network number;

SOCKET_ID： the generated socket index value (referenced

internally by the UDP layer of the PLC); **CONFIRM：** false： the

function block execution failed, true： the function block

execution succeeded; **ERROR：** the error code describes the

information about the function block execution result.

ERRORINFO： Secondary

error information.

describe:

function block LAN_UDP_CREATE_SOCKET can establish a UDP socket connection for sending and receiving data. If the connection is used to receive data, the input PORT must be a valid port number. Only under the premise that the port is valid, the PLC will call the internal function at the UDP layer, and can receive the data packet at the IP address matching its port number . In most systems, the port number less than 1024 is only used by the internal privileged program.

Use, port numbers in the range 1024 to 49151 are still reserved for default applications and managed by IANA (Internet Assigned Numbers Authority), so it is best to use port numbers from 49152 to 65535 for PLC UDP communication.

If the created connection is only used to send data, the port number is arbitrary. If the input PORT is set to 0, the PLC will call a free port number in the internal dedicated space of the UDP layer to send data. At this time, the internal The function is not necessary, however, the specification of the port number that is defined to send the data is necessary, eg an active firewall in the network only forwards data to a specific port.

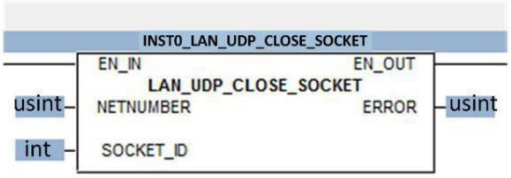
After the function block is executed correctly, the PLC will internally reference the UDP layer and generate a SOCKETID at the output, the value of this SOCKETID will be

Referenced in the relevant function block called subsequently, e.g. the function block LAN_UDP_SENDTO_STR or

LAN_UDP_RECVFROM_STR .

Socket connections that are no longer needed can be closed with the function block LAN_UDP_CLOSE_SOCKET .

3.3.44 LAN_UDP_CLOSE_SOCKET



用于关闭一个不再需要的UDP socket连接

Definition of operands:

SOCKET_ID： The index value of the socket that needs to be closed ;

ENABLE： enable or disable the function block,

true is enabled, false is disabled; **NETNUMBER：**

network number;

CONFIRM： false means the function block execution

fails, true means the function block execution

succeeds; **ERROR：** the error code describes the

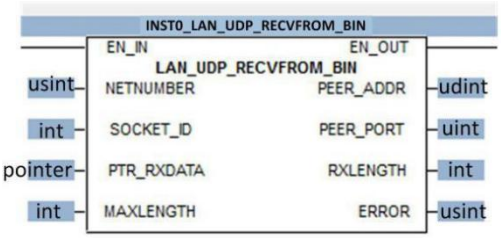
information about the function block execution result;

ERRORINFO： secondary error information.

describe:

SOCKETID is internally referenced by the UDP layer of the PLC when the function block LAN_UDP_CREATE_SOCKET is called. All unclosed socket connections will be automatically closed when the PLC program exits.

3.3.45 LAN_UDP_RECVFROM_BIN



用于从UDP层接收缓存区读取UDP数据包

Definition of operands:

SOCKET_ID： The index value of the socket to be polled ; **PTR_RXDATA：** The address where the data byte object (received) is stored;

MAXLENGTH： The limit of the number of bytes to be read;

ENABLE： enable or disable the function block, true is enabled, false is disabled;

NETNUMBER： network number;

PEER_ADDR: The address of the opposite end (remote end), which can be obtained when the opposite end is connected; **PEER_PORT:** The port number of the opposite end (remote end), which can be obtained when the opposite end is connected; **RXLENGTH：** The number of characters read ;

CONFIRM： false：the function block execution failed, true：the function block execution succeeded; **ERROR：** the error code explains the information about the function block execution result. **ERRORINFO：** Secondary error information.

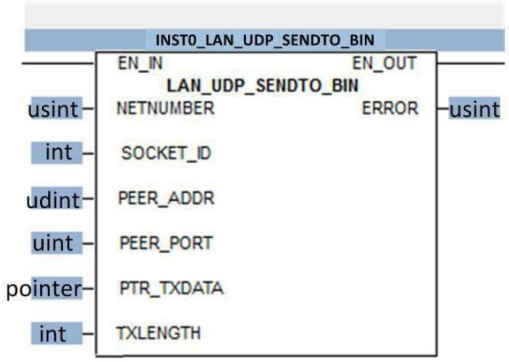
describe:

This function block is used to read UDP data packets from the receive buffer of the UDP layer. After the function block is successfully executed, confirm will be set to TRUE. PTR_RXDATA contains the received data bytes, and the output RXLENGTH specifies the number of data bytes read. If the function block fails, no data bytes will be output.

When the data packet is successfully received (CONFIRM is TRUE), the output

PEER_ADDR and PEER_PORT are used to store the peer (remote end) IP address and port number. If the PLC responds to the received data packet, the opposite end address and opposite end port must be used as the target specification for subsequent function block calls. When this function block is executed, it needs to pass the function The block LAN_UDP_CREATE_SOCKET opens a socket connection.

3.3.46 LAN_UDP_SENDTO_BIN



用于发送UDP数据包

Definition of operands:

SOCKET_ID： the socket index value for sending ;

PEER_ADDR： The IP address of the peer (remote), which can be obtained when the peer is connected; **PEER_PORT：** The port number of the peer (remote), which can be obtained when the peer is connected; **PTR_TXDATA：** The binary to be sent the address of the object of the data;

TXLENGTH： the number of data bytes to send;

ENABLE： enable or disable the function block,

true is enabled, false is disabled; **NETNUMBER：**

network number;

CONFIRM： false： the function block execution failed,

true： the function block execution succeeded; **ERROR：**

the error code explains the information about the

function block execution result. **ERRORINFO：**

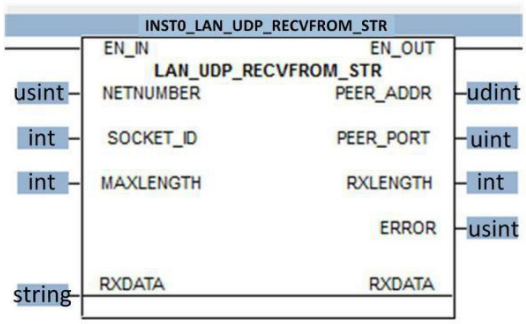
Secondary error information.

describe:

This function block is used to send UDP packets. PEER_ADDR and PEER_PORT are used to set the IP address and port number of the peer (remote end) , and are used in conjunction with the function block LAN_UDP_RECVFROM_BIN .

When this function block is executed, the socket connection needs to be opened in advance through the function block LAN_UDP_CREATE_SOCKET .

3.3.47 LAN_UDP_RECVFROM_STR



用于从UDP层的接收缓存区读取UDP数据包

Definition of operands:

RXDATA： a variable used to receive the read string; **SOCKET_ID：** the index value of the socket to be polled ; **MAXLENGTH：** the limit of the number of bytes to be read;

ENABLE： enable or disable the function block, true is enabled, false is disabled; **NETNUMBER：** network number;

RXDATA： a variable used to receive the read string;

PEER_ADDR: The address of the opposite end (remote end), which can be obtained when the opposite end is in a connected state;

PEER_PORT: The port number of the opposite end (remote end), which can be obtained when the opposite end is in a connected state;

RXLENGTH： Read the string length;

CONFIRM： false：function block execution failed, true：function block execution succeeded;

ERROR： The error code describes information about the result of the execution of the function block. Possible error codes are detailed in Table 27;

ERRORINFO： Secondary

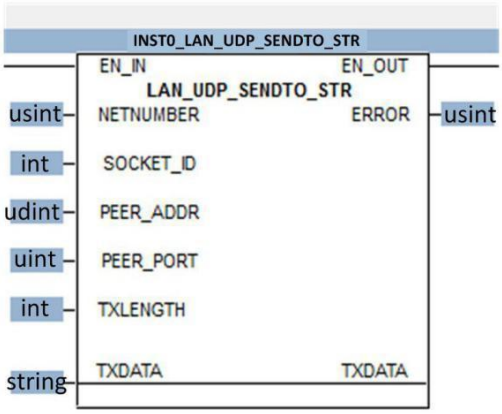
error information.

describe:

This function block is used to read UDP packets from the UDP receive buffer . PEER_ADDR and PEER_PORT are used to store the IP address and port number of the peer (remote end) .

When this function block is executed, the socket connection needs to be opened in advance through the function block LAN_UDP_CREATE_SOCKET.

3.3.48 LAN_UDP_SENDTO_STR



用于发送UDP数据包

Definition of operands:

TXDATA： String variable to be sent;

SOCKET_ID： Socket index value for

sending ;

PEER_ADDR： The IP address of the peer (remote), which can be

obtained when the peer is connected; **PEER_PORT：** The port

number of the peer (remote), which can be obtained when the

peer is connected;

TXLENGTH： the number of data bytes to send, if the number is 0, the length of the

object is determined by the PTR_TXDATA internal addressing; **ENABLE：** enable or

disable the function block, true: enable, false: disable;

NETNUMBER： network number;

TXDATA： string variable to

be sent;

CONFIRM： false：function block execution failed, true：function block execution succeeded;

ERROR： The error code describes information about the result of the execution of the function block. Possible error codes are defined as follows:

0: No error occurred while executing the function block

1: When calling the function block, an invalid network number was set

2: When calling the function block, an invalid parameter was set

3: Error in UDP initialization of PLC

4: When adding, sending, or receiving a socket

connection, the UDP layer of the PLC reports an error 5:

There is no available socket connection

6: An invalid socketID is set. The socket corresponding to

socketID cannot be used normally. 8: The sending buffer

area is too large, and the packet size is limited by the

maximum number of bytes.

9: The sending buffer is too small, no data can be sent

10: There is an error in the set host

11: The pointer points to an unsupported data type

ERRORINFO： Secondary

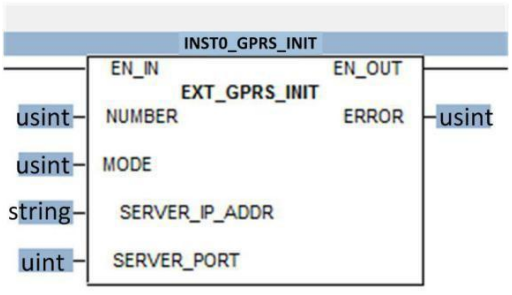
error information.

describe:

This function block is used to send UDP packets. PEER_ADDR and PEER_PORT are used to set the IP address and port number of the peer (remote end) .

When this function block is executed, the socket connection needs to be opened in advance through the function block LAN_UDP_CREATE_SOCKET .

3.3.49 EXT_GPRS_INIT



扩展GPRS通讯初始化功能块

Definition of operands:

EN_IN： function block

enable input; **NUMBER：**

network number;

MODE： Communication mode, 1 is TCP, 2 is UDP;

SERVER_IP_ADDR: server IP address;

SERVER_PORT: server port

number; **EN_OUT：** function

block enable output;

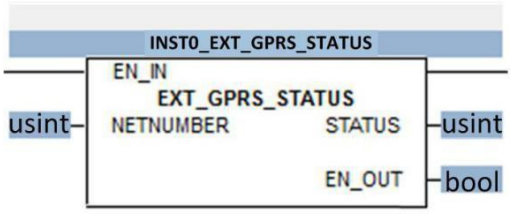
ERROR： The error code describes the

information about the result of the execution

of the function block Description:

This function block is used to initialize extended GPRS communication.

3.3.50 EXT_GPRS_STATUC



获取GPRS通讯状态功能块

Definition of operands:

EN_IN： function block

enable input;

NETNUMBER： network

number; **STATUS：** GPRS

communication status;

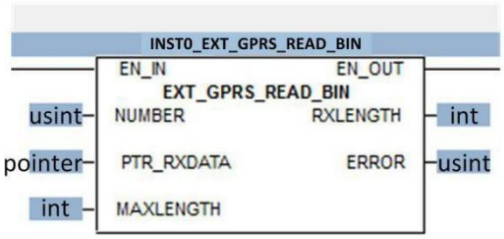
EN_OUT： function block

enable output;

description:

This function block is used to obtain the GPRS communication status.

3.3.51 EXT_GPRS_READ_BIN



GPRS通讯读取数据

Definition of operands:

EN_IN： function block

enable input;

NETNUMBER： network

number;

PTR_RXDATA： Array pointer stored

after reading data; **MAXLENGTH：**

Maximum length of data to be read;

EN_OUT： Function block enable output;

RXLENGTH： The length of the data actually read;

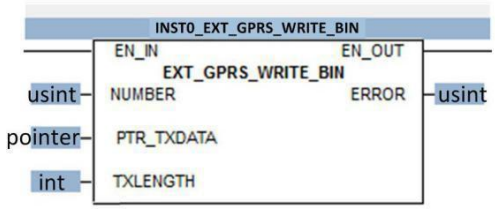
ERROR： The error code describes the error

information of the function block execution

result; description:

This function block is used for GPRS communication to read data.

3.3.52 EXT_GPRS_WRITE_BIN



GPRS通讯发送数据

Definition of operands:

EN_IN： function block

enable input;

NETNUMBER： network

number;

PTR_RXDATA： The array pointer where the data to be sent is stored;

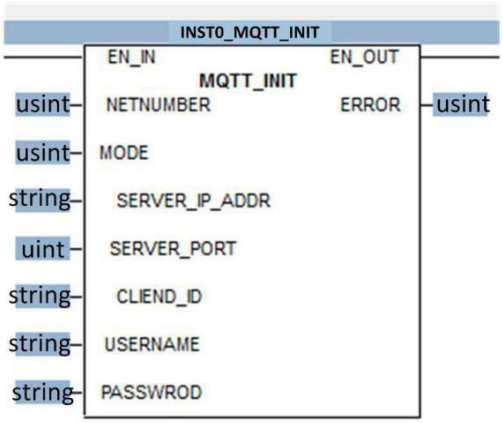
TXLENGTH： The length of the sent data;

EN_OUT： function block enable output;

ERROR： The error code describes the error information of the function block execution result; description:

This function block is used for GPRS communication to send data.

3.3.53 MQTT_INIT



MQTT初始化功能块

Definition of operands:

EN_IN： function block

enable input; **NETNUMBER：**

network number; **MODE：**

function mode;

SERVER_IP_ADDR： server IP

address; **SERVER_PORT：** server

port number; **CLIEND_ID：** client

ID name; **USERNAME：** username;

PASSWROD： password;

EN_OUT： function block enable output;

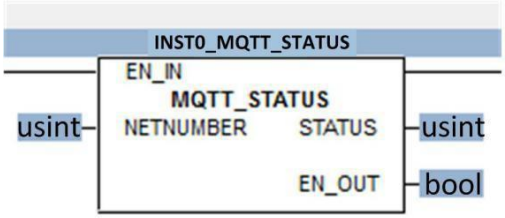
ERROR： The error code describes the error

information of the function block execution

result; description:

This function block is used for MQTT communication initialization.

3.3.54 MQTT_STATUS



MQTT状态功能块

Definition of operands:

EN_IN： function block

enable input;

NETNUMBER： network

number; **STATUS：** MQTT

communication status;

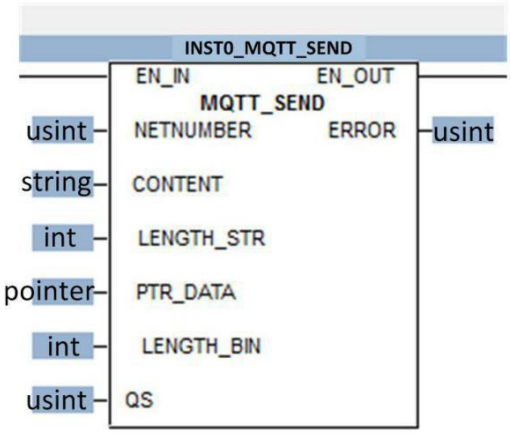
EN_OUT： function block

enable output;

description:

This function block is used to get the MQTT status.

3.3.55 MQTT_SEND



MQTT发送数据功能块

Definition of operands:

EN_IN： function block

enable input;

NETNUMBER： network

number;

CONTENT： data structure;

LENGTH_STR： length of the

string to be sent;

PTR_DATA： pointer to the storage

array of data to be sent;

LENGTH_BIN： length of the character

stream to be sent; **QS：** character

stream parameter;

EN_OUT： function block enable output;

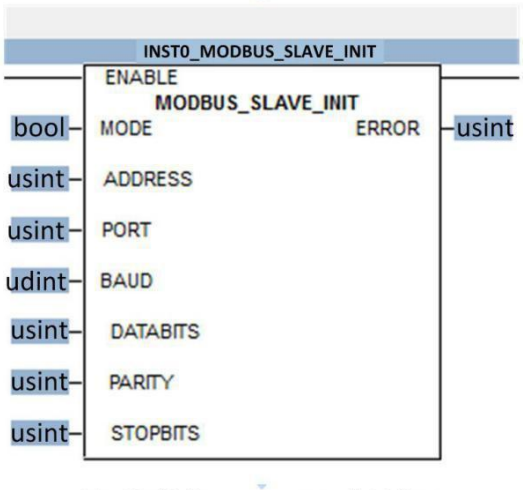
ERROR： The error code describes the

information about the execution result of

the function block; description:

This function block is used to send data through MQTT.

3.3.56 MODBUS_SLAVE_INIT



用于初始化Modbus-RTU从站接口

Definition of operands:

ENABLE： Enable or disable the input of the function block;

MODE： Select the communication protocol: enter a value of 1 to define the Port as Modbus protocol and enable the protocol, and enter a value of 0 to define the Port as PPI and disable the Modbus protocol.

ADDRESS： Set the site address

PORT： The serial interface definition number to be used, 1is RS232, 2is RS485BAUD：Set the baud rate. 1200，2400，4800，

9600, 19200, 38400, 57600, 115200

DATABITS: Number

of data bits

PARITY: Set parity

0: No parity ,1: Odd parity ,2: Even parity

STOPBITS: stop bits

ERROR: The error code describes the information of the execution result of the function block. Possible error codes are defined as follows:

0: no error

1: memory area range error

2: Illegal baud rate or parity

3: Illegal slave

address 4: Illegal

value of Modbus

parameter

5: The holding register is duplicated with the Modbus slave symbolic address

6: Receive check error

7: Receive CRC error

8: Illegal feature request /unsupported feature

9: Illegal memory area address in the request

10: The slave

function is not

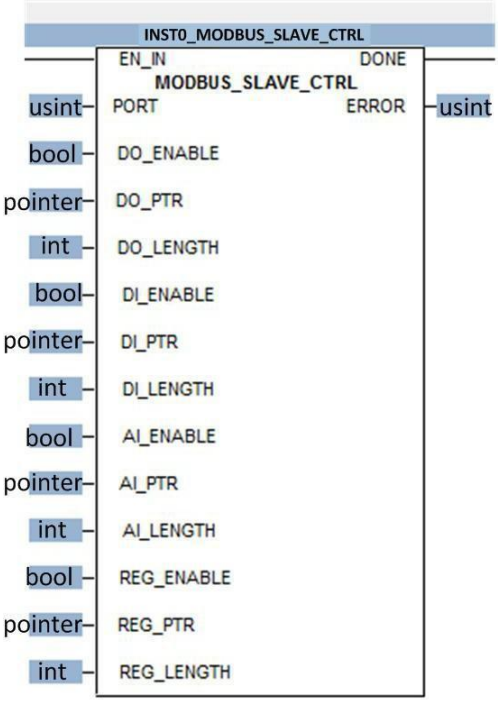
enabled

Description:

The MODBUS_SLAVE_INIT instruction is used to enable and initialize or disable Modbus communication.

The MODBUS_SLAVE_INIT instruction must be executed without errors before the MBUS_SLAVE instruction can be used. The MODBUS_SLAVE_INIT instruction must be executed before continuing with the next instruction . The MODBUS_SLAVE_INIT instruction should only be executed once per communication state change. Therefore, the ENABLE input should be pulse-triggered using an edge-detect element, or executed only once during the first loop cycle.

3.3.57 MODBUS_SLAVE_CTRL



执行Modbus-RTU从站接口的控制指令

Definition of operands:

PORT： Serial interface

definition number to be used;

DO_ENABLE： Digital output

enable; **DO_PTR：** Digital output

storage address; **DO_LENGTH：**

Digital output length;

DI_ENABLE： Digital input

enable; **DI_PTR：** Digital input

storage address; **DI_LENGTH：**

digital input data length;

AI_ENABLE： analog input enable;

AI_PTR： analog input storage address; **AI_LENGTH：** analog input data length; **REG_ENABLE：** register enable;

REG_PTR： Register storage address;

REG_LENGTH： Register storage data

length; **DONE:** Completion flag bit;

ERROR： According to the error code of the data type ″ CAN_ERROR ″, the possible error codes are defined as follows:

0: no error

1: Response verification error

2: unused

3: Receive timeout (no response from slave)

4: Request parameter

error 5: Modbus/

Freeport not enabled

6: Modbus is busy with other requests

7: Response error (response is not the requested operation)

8: Response CRC

checksum error

description:

Calling the function block MODBUS_SLAVE_CTRL can realize the

data transmission and reception of the MODBUS-RTU slave

station. Sample program:

Var; inst1_MODBUS_SLAVE_CTRL:MODBUS_SLAVE_CTRL;

modbusDOBuf : ARRAY[0..5] OF byte; DO_Ptr : POINTER;

modbusDIBuf : ARRAY[0..5] OF byte; DI_Ptr : POINTER;

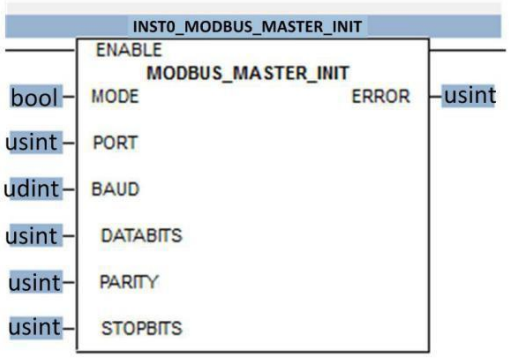
```
modbusAIBuf : ARRAY[0..10] OF int; AI_Ptr : POINTER;

modbusRegBuf : ARRAY[0..127] OF int; mPtr : POINTER;

xDONE:BOOL;

END_VAR
```

if xInitOk then mPtr:=&modbusRegBuf; DO_Ptr:=&modbusDOBuf; DI_Ptr:=&modbusDIBuf; AI_Ptr:=&modbusAIBuf;

inst1_MODBUS_SLAVE_CTRL(DO_ENABLE :=1 ,DO_PTR :=DO_Ptr , DO_LENGTH :=5 , DI_ENABLE :=1 , DI_PTR :=DI_Ptr , DI_LENGTH :=5 , AI_ENABLE :=1 , AI_PTR :=AI_Ptr ,AI_LENGTH :=10 , REG_ENABLE :=1 , REG_PTR :=mPtr , REG_LENGTH :=127 | xDONE := DONE, xERROR := ERROR);

end_if;

3.3.58 MODBUS_MASTER_INIT



用于初始化Modbus-RTU主站接口

Definition of operands:

ENABLE： enable or disable the function block,

true is enabled, false is disabled; **MODE：** mode

selection;

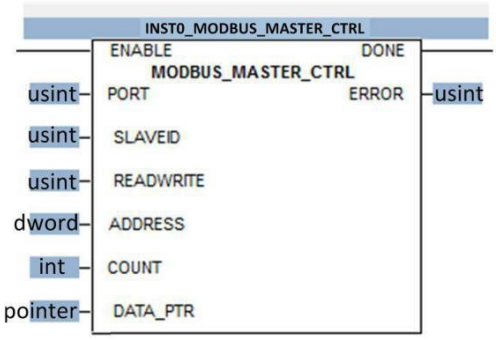
PORT： port number, 1 is 232 , 2 is 485 ;

BAUD： Baud rate, set the baud rate. 1200 , 2400 , 4800 , 9600 , 19200 , 38400 , 57600 , 115200 ; **DATABITS：** Number of data digits PARITY : Set check

0 : No parity, 1 : Odd parity, 2 : Even parity STOPBITS : Stop bits

ERROR： The error code describes the information of the execution result of the function block.

3.3.59 MODBUS_MASTER_CTRL



执行Modbus-RTU主
站接口的控制指令（功能码）

Definition of operands:

ENABLE： function block enable

PORT： Serial interface definition number to be used, 1 is 232, 2 is 485; **SLAVEID：** ID of Modbus_RTU slave station;

READWRITE： read and write enable; **ADDRESS：** slave address; **COUNT：** the number of read or write;

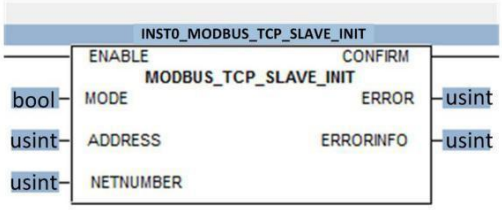
DATA_PTR： Array pointer to read or write; **DONE：** Completion flag bit;

ERROR： error message;

description:

Calling the function block MODBUS_MASTER_CTRL can realize the data sending and receiving of the MODBUS-RTU master station.

3.3.60 MODBUS_TCP_SLAVE_INIT



用于初始化Modbus-TCP从站接口

Definition of operands:

ENABLE： enable or disable the function block, true is enabled, false is disabled;

MODE： mode selection;

ADDRESS： specifies the slave address of the Ethernet TCP side;

NETNUMBER： network number;

CONFIRM： false：function block execution failed, true：function block execution succeeded;

ERROR： The error code describes information about the result of the

execution of the function block. Possible error codes are defined as

follows :0:No error occurred while executing the function block

1:A hardware error occurred while executing the function block

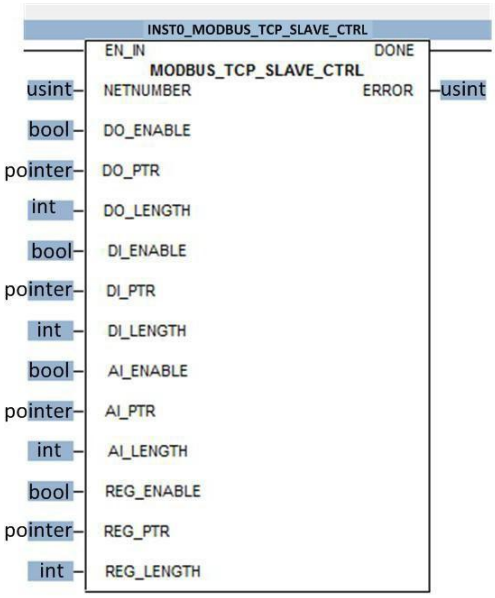
2:When calling the function block, an invalid mode was set

4:The set address is too large and exceeds the maximum available storage area

8:When calling the function block, an invalid ethernet interface was set

ERRORINFO： secondary error information

3.3.61 MODBUS_TCP_SLAVE_CTRL



用于执行Modbus-TCP从站接口的控制指令

Definition of operands:

NETNUMBER： network number;

DO_ENABLE： digital output

enable; **DO_PTR：** digital output

storage address; **DO_LENGTH：**

digital output length;

DI_ENABLE： digital input

enable; **DI_PTR：** digital input

storage address; **DI_LENGTH：**

digital input data length;

AI_ENABLE： analog input enable;

AI_PTR： analog input storage

address; **AI_LENGTH：** analog

input data length; **REG_ENABLE：**

register enable; **REG_PTR：**

register storage address;

REG_LENGTH： Register storage data

length; **DONE：** Completion flag bit;

ERROR： The error code describes information about the result of the execution of the function block. Possible error codes are defined as follows:

0: No error occurred while executing the function block

1: A hardware error occurred while executing the function block

2: When calling the function block, an invalid mode was set

4: The set address is too large and exceeds the maximum available storage area

8: When the function block is called, an

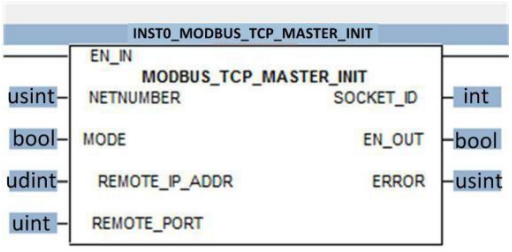
invalid Ethernet interface is set.

Example program:

```
VAR
lanInit:bool:=false; inst0_LAN_INIT:LAN_INIT;
mConfirm:bool; mError:usint; mErrorinfo:usint; mIP:udint; mNetMask:udint; mGateWay:udint;
mSocket:INT;
inst0_MODBUS_TCP_SLAVE_INIT:MODBUS_TCP_SLAVE_INIT;
mModbusInitOK:bool:=false; modbusDOBuf : ARRAY[0..5] OF byte; DO_Ptr : POINTER;
modbusDIBuf : ARRAY[0..5] OF byte; DI_Ptr : POINTER;
modbusAIBuf : ARRAY[0..10] OF int; AI_Ptr : POINTER;
modbusRegBuf : ARRAY[0..127] OF int; mPtr : POINTER;
xDONE:BOOL; inst4_MODBUS_TCP_SLAVE_CTRL:MODBUS_TCP_SLAVE_CTRL;
xError:usint;
mDI at %I0.0:byte; mAI at %I1.0:int; mDO at %Q0.0:byte;
END_VAR
```

```
if lanInit=false then
mIP := LAN_ASCII_TO_INET('192.168.1.31'); mNetMask := LAN_ASCII_TO_INET('255.255.255.0');
mGateWay := LAN_ASCII_TO_INET('192.168.1.1');
inst0_LAN_INIT(ENABLE :=true , HOSTNAME := 'PLCCore'， IP := mIP, NETMASK :=mNetMask,
GATEWAY :=mGateWay ,NETNUMBER:=1
| mConfirm:= CONFIRM,mError:= ERROR， mErrorinfo:= ERRORINFO); lanInit:=true;
else
if mModbusInitOK=false then
inst0_MODBUS_TCP_SLAVE_INIT(ENABLE := true, MODE := 1, ADDRESS :=1 ,
NETNUMBER :=1 | mConfirm:= CONFIRM, mError:= ERROR, mErrorinfo:= ERRORINFO);
mModbusInitOK:=true; end_if;
end_if; mDO:=modbusDOBuf[0]; modbusDIBuf[0]:=mDI; modbusAIBuf[0]:=mAI;
modbusRegBuf[0]:=byte_to_int(mDI); modbusRegBuf[1]:=mAI;
modbusRegBuf[2]:=byte_to_int(mDO);
mPtr:=&modbusRegBuf; DO_Ptr:=&modbusDOBuf; DI_Ptr:=&modbusDIBuf; AI_Ptr:=&modbusAIBuf;
if mModbusInitOK=true then
inst4_MODBUS_TCP_SLAVE_CTRL(NETNUMBER :=1,DO_ENABLE :=1 , DO_PTR :=DO_Ptr ,
DO_LENGTH :=5 , DI_ENABLE :=1 ,DI_PTR :=DI_Ptr , DI_LENGTH :=5 , AI_ENABLE :=1 ,
AI_PTR :=AI_Ptr , AI_LENGTH :=10 , REG_ENABLE :=1 , REG_PTR :=mPtr , REG_LENGTH :=127 | xDONE :=
DONE, xError := ERROR);
end_if;
```

3.3.62 MODBUS_TCP_MASTER_INIT



用于初始化Modbus-TCP主站接口

操作数的定义：

EN_IN： enable or disable the function block,

true is enabled, false is disabled; **MODE：** mode

selection;

REMOTE_IP_ADDR: Remote IP address

REMOTE_PORT: remote

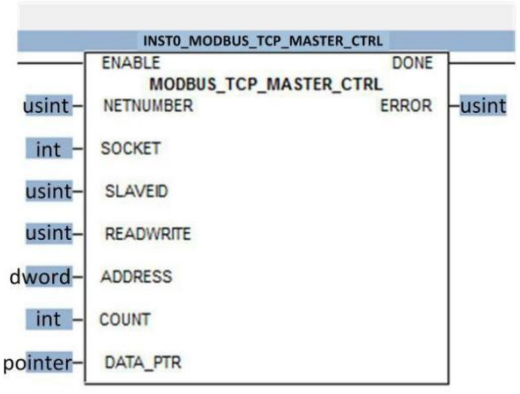
port number **EN_OUT:**

completion flag;

SOCKET_ID: network ID

ERROR: The error code describes information about the result of the execution of the function block.

3.3.63 MODBUS_TCP_MASTER_CTRL



用于执行Modbus-TCP主
站接口的控制指令（功能码）

Definition of

operands **ENABLE：**

function block

enable; **NETNUMBER：**

network number;

SOCKET： network ID；

SLAVEID： slave

station ID；

READWRITE： read and

write enable; **ADDRESS：**

slave station address;

COUNT： the number of

read and write;

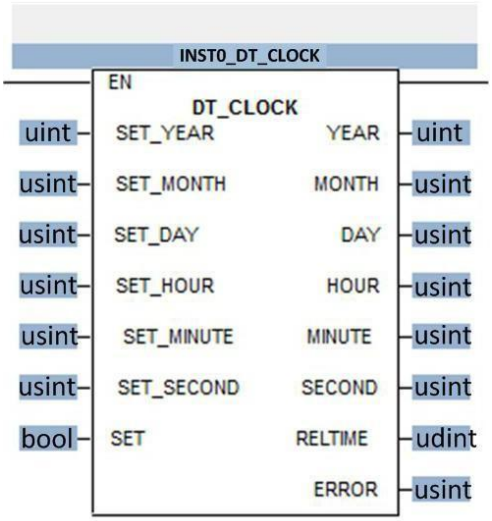
DATA_PTR： Pointer to store read and

write variable array; **DONE：**

Completion flag bit;

ERROR： The error code describes information about the result of the execution of the function block.

3.3.64 DT_CLOCK



用于设置实时时钟RTC并从PLC的RTC中读取日期和时间。这个功能块仅在配有RTC功能的控制器上可用

Definition of operands:

SET_YEAR, SET_MONTH, SET_DAY: year / month / day setting

SET_HOUR, SET_MINUTE, SET_SECOND: Hour / minute / second setting

SET: TRUE: Allows the inputs SET_YEAR, SET_MONTH and SET_DAY to change the date, allows the inputs SET_HOUR, SET_MINUTE and SET_SECOND to change the specific time, and at the same time, can read the setting at the respective outputs date and time.

FALSE: Only read the current date and time of the RTC, the date and time cannot be changed. YEAR, MONTH, DAY: output date of year / month / day

HOUR, MINUTE, SECOND: output time in hours / minutes / seconds

RELTIME: Read the relative form of date and time (seconds since 01.01.1980) ERROR: Error code information describing the result of function block execution. Possible error codes are defined as follows: No error occurred while executing the function block

A hardware error occurred while executing the function block

4 The mode (MODE) during a function block call is invalid

8 Power failure, invalid read time

16 Reading absolute time and date invalid

description:

If the function block is called by setting the input SET to TRUE, the input date (SET_YEAR, SET_MONTH and SET_DAY) and input time (SET_HOUR, SET_MINUTE and SET_SECOND) can be passed to the RTC of the PLC according to their respective input values. At the same time, the set date and time can be read on the respective outputs. However, if the function block is called by setting the input SET to FALSE, only the current

date and time can be read, the RTC is not affected, the setting input is discarded, and possible errors during the execution of the function block are displayed at the output ERROR.

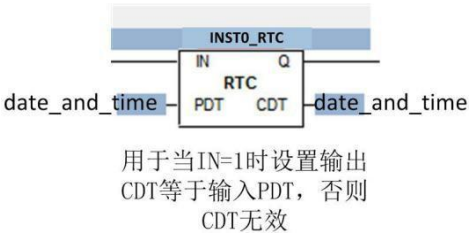
output is ERROR=3 after the execution of the function block DT_CLOCK, the power supply is RTC interrupted (power failure or empty battery) and the read time is invalid. This error state remains until the PLC's RTC is reset (the function block is called via the input SET: =TRUE) or the PLC is reset via the reset switch.

The following sample program shows the process of the function block DT_CLOCK setting and viewing the system RTC.

Sample program:

```
VAR
Year : UINT; Month : USINT; Day : USINT; Hour : USINT;
Minute : USINT; Second : USINT; RelTime : UDINT;
ErrorCode : ARRAY [0..1] OF USINT;
FB_DtClock : DT_CLOCK; END_VAR
LD 0
ST ErrorCode[0] ST ErrorCode[1]
CAL FB_DtClock (SET_YEAR := 2003,SET_MONTH := 8,SET_DAY := 6,SET_HOUR := 12, SET_MINUTE :=
3,SET_SECOND := 0,SET := TRUE|Error[0] := ERROR)
CAL FB_DtClock (SET :=FALSE)
LD FB_DtClock.YEAR ST Year
LD FB_DtClock.MONTH ST Month
LD FB_DtClock.DAY ST Day
LD FB_DtClock.HOUR ST Hour
LD FB_DtClock.MINUTE ST Minute
LD FB_DtClock.SECOND ST Second
LD FB_DtClock.RELTIME ST RelTime
LD FB_DtClock.ERROR ST ErrorCode[1] RET
END_PROGRAM
```

3.3.65 RTC



操作数的定义：

IN： Function block

enable input PDT：

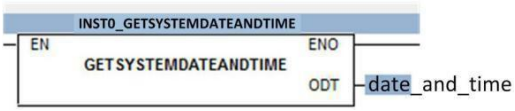
System date and

time Q： Same state

as IN

CDT： When IN=1, output the system date and time

3.3.66 GETSYSTEMDATEANDTIME



Definition of

Operands: **EN:**

Enable Input

ENO: Enable

Output **ODT:**

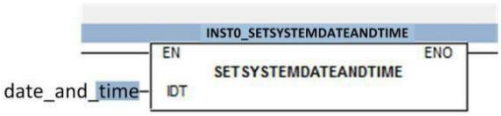
System Time

Description:

This function block is used to display the actual system time at the output ODT,

which is not defined by the IEC61131-3 standard.

3.3.67 SETSYSTEMDATEANDTIME



Definition of operands:

EN: enable input

IDT: Set value of system

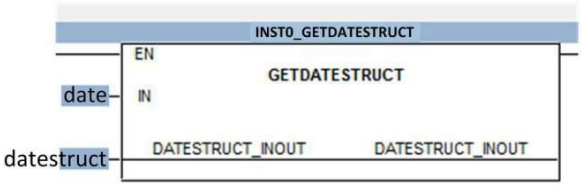
date and time **ENO:** Enable

output

describe:

sets the actual system time at the input IDT , which is not defined by the IEC61131-3 standard.

3.3.68 GETDATESTRUCT



由日期类型转为结构体类型

Definition of operands :

EN: enable input

IN : the date to convert

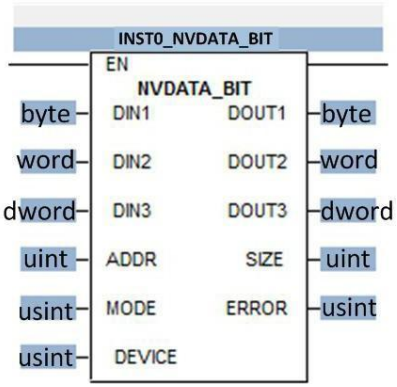
DATESTRUCT_INOUT: Converted date structure , such as:

datestruct : structYear : uint ; Month : usintDay :

usintend_struct Description:

GETDATESTRUCT converts a given date into a date structure, representing day, month, and year as separate integer variables.

3.3.69 NVDATA_BIT



用于在读取PLC存储单元（EEPROM）中数据的同时，写入过程数据(这些数据可以是1字节，或1个字，或4个字节)

Definition of operands:

DIN1 : write data of one byte; DIN2 : write data of two bytes; DIN3 : write data of four bytes;

ADDR : address in memory for reading and writing data (parameter mode needs to be set); MODE : setting for performing read or write operations, all supported modes are included in Table 31-1 ; DEVICE : device number, required set to 0 ;

DOUT1 : read one byte of data; DOUT2 : read two bytes of data; DOUT3 : read four bytes of data;

SIZE : When MODE is not 0 , it is used to output the

number of bytes read or written, and when MODE is

0, it is used to output the available memory size;

ERROR: The error code describes information about the result of the execution of the function block. Possible error codes are defined as follows: 0:No error occurred while executing the function block

1:A hardware error occurred while executing the function block

2: When calling the function block, an invalid device number was set

4: When calling the function block, an invalid mode was set

8: The set address is too large and exceeds the maximum available storage area

16: The pointer points to an

unsupported data type function

block call mode :

16#00 Determine available memory size

16#01 in DOUT1, read one byte of data

16#02 In DOUT2, read two bytes of data

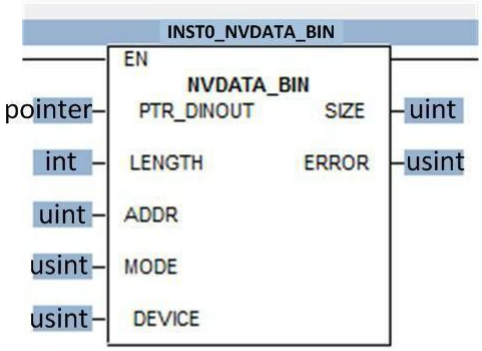
16#03 In DOUT3, read four bytes of data

16#81 in DIN1, write one byte of data

16#82 in DIN2, write two bytes of data

16#83 in DIN3, write four bytes of data

3.3.70 NVDATA_BIN



用于在读取PLC存储单元（EEPROM）中数据的同时，写入过程数据(这些数据为二进制数据)

Definition of operands:

PTR_DINOUT : Digital input and output storage

address (for reading and writing data); **LENGTH：**

Length of read **/** write bytes;

ADDR： address in memory for reading and writing data

(parameter mode needs to be set)；**MODE：** setting for

performing read or write operations, all supported modes

are included in Table 30-1；**DEVICE：** device number, required

set to 0；

SIZE： When MODE is not 0, it is used to output the number of bytes read or written,

MODE is 0, it is used to output the available memory size;

ERROR： The error code describes information about the result of the execution of the function block. Possible error codes are defined as follows:

0:No error occurred while executing the function block

1:A hardware error occurred while executing the function block

2:When calling the function block, an invalid device number was set

4:When calling the function block, an invalid mode was set

8:The set address is too large and exceeds the maximum available storage area

16:The pointer points to an

unsupported data type mode

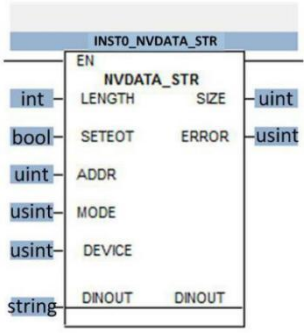
function:

16#00:Determine available memory size

16#09:Read binary data from memory ,address at PTR_DINOUT

16#89:Write binary data to memory at address PTR_DINOUT

3.3.71 NVDATA_STR



该功能块用于将字符串过程数据写入PLC非易失性存储区中或从PLC非易失性存储区中读取字符串过程数据

Definition of operands:

EN： Function block enable input

LENGTH： Limit on the number of strings to read or write. If the value is 0, the length of the string is internally defined (equal to LEN (DINOUT)); note: the standard string buffer length in OpenPCS is 32 characters. **SETEOT：** TRUE: string storage with terminating character FALSE: string storage without terminating character (default :TRUE) **ADDR：** address in memory for reading and writing data

MODE： The operating mode setting for read or write, list 12 includes all supported modes.

DEVICE： Device number, this parameter is determined by the respective control system. (Note: Most control systems only support device number 0, and device number 0 will be defaulted to the initial value, so it is not necessary to set it separately)

DINOUT： Input and output for reading and writing string values

SIZE： used to indicate the number of bytes read or written (MODE ≠ 0) or the available size of the non-volatile memory area (MODE=0) **ERROR：** The error code indicates the information of the execution result of the function block, and the possible error codes are defined as follows :

How to call the function block:

16#00 Determine the size of the available non-volatile memory

16#08 Read a string in a non-volatile buffer at the data output port DINOUT

16#88 Write a string to the non-volatile buffer at the

data input port DINOUT as described in:

The character string data in the non-volatile storage area can be read or written through the function block. According to the read or written data, the corresponding mode according to the list 12 must be set at the input MODE. In this case, the parameter DINOUT is used as input or output depending on the type of mode, and the value of the input ADDR is the base address where the read or write is performed. If the addressing range exceeds the maximum value, the function block will report an appropriate error. The PLC program is used to divide the available memory area and ensure that the values entered into ADDR do not cause overlapping of stored data. The number of bytes read or written is displayed to output SIZE. This value will be used to calculate the next free address (ADDRnew=ADDRold+SIZE), enter LENGTH to specify the number of characters valid during the write process, if With a value of 0, the length of the string is determined internally (equal to LEN (DINOUT)) and is used as the number of characters written, in which case the entire used string content is written. The input LENGTH is also used when reading to limit the number of characters to be processed to the specified value.

Use the input SETEOT to set whether or not the end character of the string is also stored (default is TRUE). Required (LENGTH=0), the function block will receive all character data until the end delimiter is detected in DINOUT. When the function block is called with SETEOT =false, the storage area of the terminator is empty. Therefore, each string will occupy less than one byte of non-volatile memory. But in this case the length of the string must be known and assigned to the input LENGTH during reading. If a terminator has already been written, the length of the terminator is still taken into account when the processed character is specified at the output SIZE. Therefore, when the function block is called with SETEOT=TRUE, the value of output SIZE is equal to LEN (DINOUT) +1. When the function block is called with MODE=0, the size of the available area of the non-volatile memory will be determined. In this case, the remaining size of the remaining area from the input ADDR will be fed back to the output SIZE (SIZE: =NVDATAFullSize-ADDR). Calling the function block with ADDR=0 will determine the full size of the non-volatile memory. Possible error messages during the execution of the function block will be displayed on the output ERROR and described in Table 10 (the error code is the same as the function block NVDATA_BIT).

The following sample program shows the application of the function block NVDATA_STR. Initially, a string is written from address 30 and then read from the same address, the standard settings remain the same, and the default device number is 0.

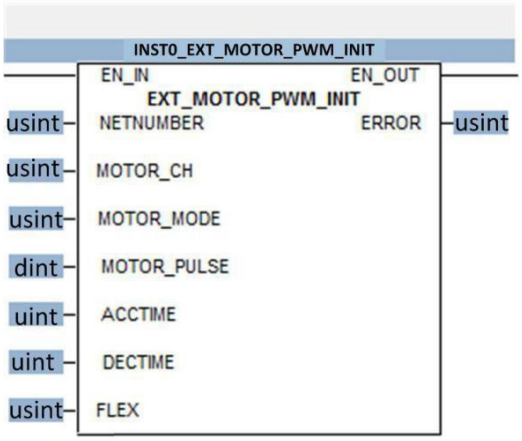
Sample program:

```
VAR CONSTANT

NVDSTR_MODE_GET_SIZE : USINT := 16#00; NVDSTR_MODE_RD_STRING : USINT := 16#08;
NVDSTR_MODE_WR_STRING : USINT := 16#88; NVDATA_ERROR_SUCCESS : USINT := 0;
NVDATA_ERROR_HW_ERROR : USINT := 1; NVDATA_ERROR_UNKNOWN_DEVICE :
USINT := 2; NVDATA_ERROR_INVALID_MODE : USINT := 4; NVDATA_ERROR_OUT_OF_MEM :
USINT := 8;

END_VAR
```

```
VAR
WriteDataString : STRING; WriteDataSize : UINT; ReadDataString : STRING; ReadDataSize : UINT;
Error : ARRAY[0..1] OF USINT;
FB_NvDataStr : NVDATA_STR;
```

```
END_VAR

(* write a STRING value into EEPROM *) LD    'HelloWorld'

ST      WriteDataString

CAL    FB_NvDataStr ( DINOUT := WriteDataString,

LENGTH := 0,          (* save whole string   *)          SETEOT := TRUE, (* include termination character *)

   ADDR := 30,

MODE := NVDSTR_MODE_WR_STRING

|

WriteDataSize := SIZE, Error[0] := ERROR)

(* read a STRING value from EEPROM *) CAL FB_NvDataStr

( DINOUT := ReadDataString,

LENGTH := 0,    (* read whole string *) ADDR := 30,

MODE := NVDSTR_MODE_RD_STRING

|

ReadDataSize := SIZE, Error[1] :=    ERROR) RET


END_PROGRAM
```

3.3.72 EXT_MOTOR_PWM_INIT



GC-2302模块初始化脉冲输出功能块

Definition of operands:

EN_IN： function block enable input;

NETNUMBER： The sequence number of the GC- 2302 module, the first GC-2302 module connected to the main control module or the coupler is 1, the second GC-2302 module is 2, and so on;

MOTOR_CH： The sequence number of the pulse output of each module, each GC-2302 module has two pulse outputs, the first pulse output serial number is 1, and the second pulse output serial number is 2;

MOTOR_MODE： The pulse output mode of the GC-2302 module： 1 is the speed mode, 2 is the position mode, 3 is the pulse mode Acceleration time, meaningless in pulse mode

DECTIME： Deceleration time,

meaningless in pulse mode **FLEX：**

Slope factor:

EN_OUT： function block enable output;

ERROR： The error code describes the

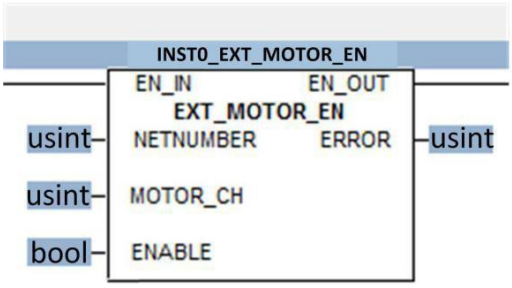
information about the result of the execution

of the function block Description:

This function block is used to initialize the pulse output of the GC-2302 module. There are three pulse output modes . When MOTOR_MODE is 1 and 2, the three parameters ACCTIME, DECTIME and FLEX are meaningful for configuring the motor. in the output address

Define and call the function block EXT_MOTOR_EN to configure the motor parameters. When MOTOR_MODE is 3, the output of the terminal module is pure pulse. Similarly, the output address can be defined in the program to configure the pulse duty cycle and frequency.

3.3.73 EXT_MOTOR_EN



GC2302模块使能电机功能块

Definition of operands:

EN_IN： function block enable input;

NETNUMBER： GC-2302 The serial number of the module, the first GC-2302 after the main control module or the coupler Module is 1, the second GC-2302 The module is 2, and so on;

MOTOR_CH： The sequence number of the pulse output of each module, each GC-2302 module has two pulse outputs, the first pulse output serial number is 1, and the second pulse output serial number is 2;

MOTOR_MODE： GC-2302 module pulse output

mode: 1 is speed mode, 2 is position mode,

3 is pulse mode **ENABLE：** motor enable

signal

EN_OUT： function block enable output;

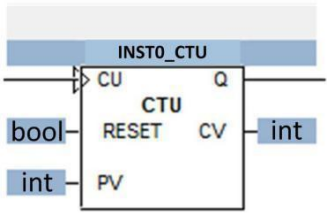
ERROR： The error code describes the

information about the result of the execution

of the function block Description:

This function block is used to enable the motor with the GC-2302 pulse output module. In the program, the function block EXT_MOTOR_PWM_INIT needs to be called in advance to initialize the output terminal module and configure the parameters. After the initialization is successful, the function block is called to enable the motor, and the motor will run according to the set parameters.

3.3.74 CTU



加计数

Definition of

operands: **CU：**

count pulse

RESET： reset

counter **PV：**

count limit value

Q： Displays whether the

counter reaches the limit

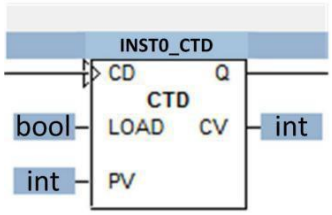
value **CV：** Current counter

value

describe:

The function block is used to count the number of pulses received from the input CU , when initialized, the value of the counter will be set to 0 , if the input RESET is set to 1 , the value of the counter will be reset, each detected rising edge at input CV It will make the value of the counter +1 , and the output CV is the current value of the counter. If the counter value is less than PV , the output Q is 0 , and if the counter value is greater than or equal to PV , the output Q is 1 .

3.3.75 CTD



减计数

Definition of

operand: **CD:**

count pulse

LOAD: set

initial value

PV: initial value

Q: Output 1 when the

counter value is 0

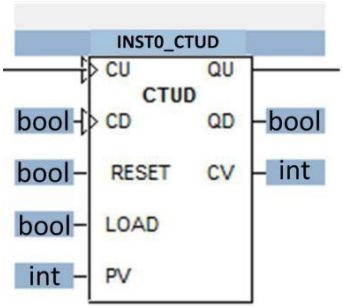
CV: The value of the

counter

describe:

function block CTD collects the pulse signal at the input CD and performs a countdown operation. If the value of the parameter LOAD is 1, the value of the operand PV will be loaded into the counter. Every time a rising edge is detected at the input CD, the counter will be counted. The value is minus 1, the output CV is the current value of the counter, if the value of the counter is positive, the value of the output Q is 0. If the value of the counter is less than or equal to 0, the value of the output Q is 1, and the function block has been approved by IEC61131-3 standard definition.

3.3.76 CTUD



加、减计数

Definition of

operands: **CU:**

count up on

rising edge **CD:**

count down on

rising edge

RESET: reset

counter **LOAD:**

load initial

value **PV:**

initial

value

QU: Displays whether the

value of the counter is

greater than PV **QD:**

Displays whether the

value of the counter is

greater than 0 **CV:**

Current value of the

counter

describe:

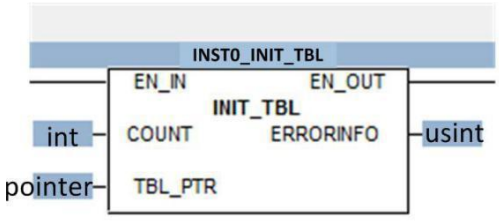
function block CTUD is used to count up and down. The value of the counter at initialization is 0. Each rising edge at the input CU will increment the counter by 1, and each rising edge at the output CD will decrement the value of the counter by 1.

If the value of operand LOAD is 1, the value of PV will be loaded into the counter

If the value of operand RESET is 1, the value of the counter will be 0. If RESET remains at 0, the normal operation of counting and loading will not be affected.

output CV is the current value of the counter. If the counter value is less than PV, the output QD will be 0. If the counter value is greater than or equal to PV, the output QD will be 1. If the counter value is positive, the output QD will be 0, if the counter value is less than or equal to 0, then the output QD will be 1. This function block has been defined in the IEC61131-3 standard.

3.3.77 INIT_TBL



初始化列表存储功能块

Definition of operands:

EN_IN: enable initialization list

storage function block; **COUNT:**

define the number of members in the

list; **TBL_PTR:** array pointer of list

members; **EN_OUT:** function block

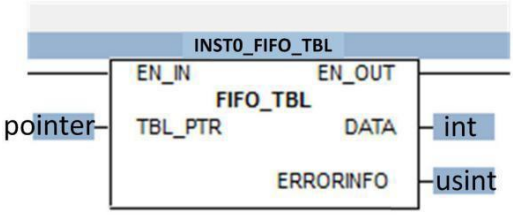
enable output; **ERRORINFO：** error

message description of the function

block execution result :

This function block is used to initialize the list storage function. The number of list members is defined by the input COUNT , and the array pointer where the list members are defined by TBL_PTR.

3.3.78 FIFO_TBL



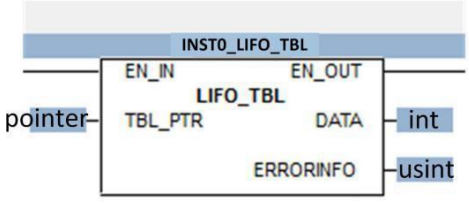
列表成员先进先出

Definition of operands:

EN_IN: enable list member FIFO function block; **TBL_PTR:** array pointer of list member; **EN_OUT:** function block enable output; **DATA:** First-in, first-out list members; **ERRORINFO：** Error information of the execution result of the function block Description:

This function block is used for first-in, first-out of list members. The function block is enabled by the input EN_IN , and TBL_PTR defines the array pointer where the list members are located.

3.3.79 LIFO_TBL



列表成员先进后出

Definition of operands:

EN_IN: Enable list member FIFO function block; **TBL_PTR:** Array pointer of list member; **EN_OUT:** Function block enable output;

DATA: First -in, last-out list member;

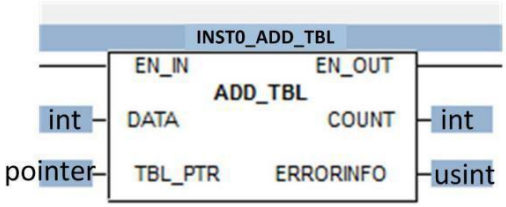
ERRORINFO： Error message

description of the function block

execution result:

This function block is used for the first-in, last-out of the list members. The function block is enabled by the input EN_IN , and TBL_PTR defines the array pointer where the list members are located.

3.3.80 ADD_TBL



添加列表成员

Definition of operands:

EN_IN: Enable adding list member function block; TBL_PTR: Array pointer of list member;

DATA: added list members;

EN_OUT: function block

enable output; **COUNT:** the

number of list members;

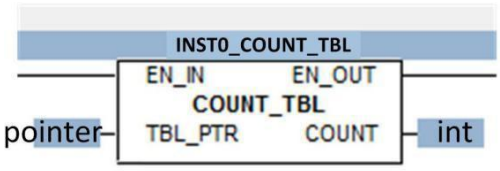
ERRORINFO： Error message

description of the function block

execution result:

enabled by the input EN_IN , and TBL_PTR defines the array pointer where the list members are located.

3.3.81 COUNT_TBL



列表成员计数

Definition of operands:

EN_IN: Enable list member

count function block; **TBL_PTR:**

Array pointer of list members;

EN_OUT: Function block enable

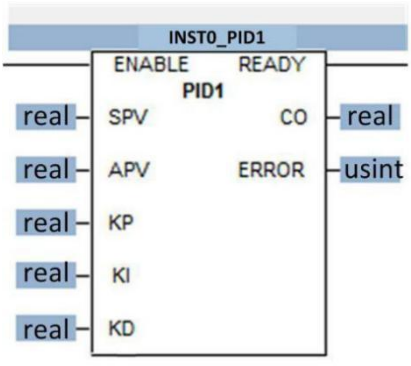
output; **COUNT：** Number of

list members;

describe

This function block is used to count the number of list members, the function block is enabled by the input EN_IN , and TBL_PTR defines the array pointer where the list members are located.

3.3.82 PID1



该功能块用于实现PID控制

Definition of Operands

ENABLE： If the input rising edge is detected, the control factors KR, T0, TI, TD will be received by the system, and the integral sum of the initial value deviation will be calculated at the same time. The outputs READY, ERROR and CO are reset by setting ENABLE=0. The function block checks the valid area and, if necessary, displays an overflow error at the output ERROR during the transfer of parameters T0 and BIAS;

SPV： controller standard setting value (control variable), valid range is 0.0~1.0, the function block will automatically detect the parameters (but will not detect overflow errors)

APV： controller standard actual value (process variable), valid range is 0.0~1.0, the controller will automatically detect the parameter (but will not detect overflow error)

KP **：**

proportional

coefficient;

KI **：** integral

coefficient;

KD **：**

differential

coefficient;

READY： Output state of the PID controller TRUE = the function blocks of the controller are fully parameterized and have entered the pre-operational state. FALSE= The function block of the controller is not fully parameterized or incorrectly parameterized (the control parameter is outside the valid value range), the controller does not enter pre- operational mode.

CO: The output of the controller, the correction variable of the controller is

calculated (the value range is 0.0~1.0)

ERROR: The error code indicates the information of the execution result of the function block. The possible error codes are defined as follows:

0: No error occurred during function block execution

8: The value of the specified parameter BIAS is invalid (less than 0 or greater than 1)

16: The value of the specified parameter T0

is invalid (time is equal to 0)

Description:

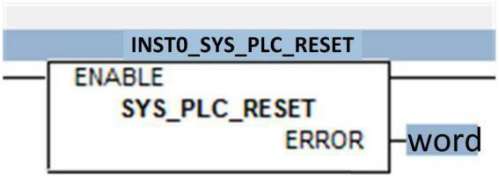
This function block is used to implement PID control, if a rising edge is detected at the input ENABLE, the function block receives the control parameters KR, TO, TD, TI and BIAS and starts the controller. In the first operation, the setpoint and the last actual value are set to the current actual value. The first time the correction variable is calculated, the initial deviation value is usually present, because the proportional gain, integral gain and derivative gain all default to 0 for the first time. The performance of the controller will be affected by the control parameters. If TI=t#0ms, the integral gain of the controller will not be calculated and will be assigned a value of 0. If TD=t#0ms, the value of the differential gain will be assigned to 0. If the gain of the parameter KR is 0, the proportional gain is not needed. Since the KR gain is related to both the proportional gain and the derivative gain, the integral and derivative gains are set to 1 here. P controller: TI=TD=0, KR ≠ 0,

The integral and derivative gains are 1. PI controller: TD=0, KR, TI ≠ 0. PID controller: KR, TI, TD ≠ 0. There are many ways to determine control parameters (tangent to inflection point, oscillation test), and parameters can also be determined by simulation tools. However, it is necessary that the complete control system be described as a model in terms of frequency and phase response. Models of the relevant transfer parameters of the control system can be built from known values and states and used to determine the parameters during the simulation. The set value, actual value, deviation value and correction variable are used as standard variables. The value of these standard variables varies from 0.0 to 1.0. When the function block starts to execute, the valid range of the deviation will be checked. When the deviation value exceeds the valid range Outputting ERROR will report an error. Setpoints and actual values are not checked during program run. The function block limits the value of the correction variable and the integral sum, and if the result of the calculation is negative, the variable value is set to 0. If the value exceeds 1, the variable value is set to 1.In addition, the integral sum is affected by the correction variable, also following the following rules:

If the value of the correction variable is greater than 1, the integral sum is calculated as follows: Integral sum = 1- (proportional gain + differential gain)

If the value of the correction variable is greater than 0, the integral sum is calculated as follows: Integral gain = 1- (Proportional gain + differential gain)

3.3.83 SYS_PLC_RESET



PLC系统复位重启功能块

Definition of operands:

ENABLE: Enable PLC system reset restart function block;

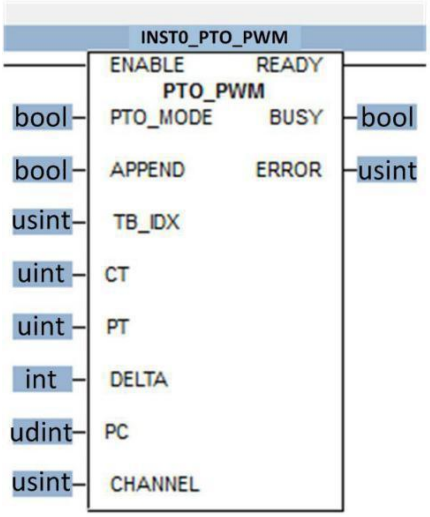
ERROR： The error code describes information

about the result of the execution of the

function block; description:

This function block is used to reset and restart the PLC system. When the input ENABLE is 1, the PLC will stop the current process and restart.

3.3.84 PTO_PWM



脉冲输出PTO功能块

Definition of operands:

ENABLE: enable or disable the function block,

true is enabled, false is disabled; PTO_MODE: mode

selection

1: PTO generator (pulse counter output, one-shot pulse train)

0: PWM generator (pulse width modulated output, continuous pulse train)

while input ENABLE is 1 terminates the functions set in the previous mode

APPEND: used to control extension parameters

1: Accept currently configured parameters as further parameter sets

0: Only update the output status of the function block, discarding the previously configured parameters

TB_IDX: used to set the basic period of the pulse generator, this parameter depends on the corresponding control mode. The valid values are as follows: 0=800ns

Basic period 1=1ms basic period

set after a rising edge is received at the input ENABLE

CT: PTO Mode:

Duration PWM Mode:

Cycle Time

The duration and cycle time depend on the basic cycle specified by the input TB_IDX
TB_IDX:=0:

125...65535(100µs-52428µs)TB_IDX:=1: 2...65535(2ms-65535ms)

PT: PTO Mode: Not used

PWM mode: Pulse duration, value range: 0...65535

DELTA: PTO mode: continuous period change between two pulses, value

range :-32768...+32767 PWM mode: not used

PC: PTO mode: number of pulses, value

range :1...4294967295 CHANNEL: channel

number used

READY: The output status of the pulse generator

1: The pulse generator is fully parameterized and the generator is ready to run

0: The pulse generator is not parameterized, or the

function block terminated with an error, the generator is

not ready BUSY: Status output of the pulse generator:

1:Pulse generator active (pulse train is being generated); generator controls digital output

0:The pulse generator is inactive (the pulse train has been generated); the process image controls the digital output (PLC program direct image output)

ERROR: The error code indicates the information of the execution

result of the function block. The possible error codes are

defined as follows: 0: There is no error during the execution of

the function block

1: A hardware error occurred during the execution of the function block

2: The selected channel number is not supported

8: The index of the selected base period is not supported

16：Overflow bug in recalculation of duration when DELTA is considered

32：No space available in the data record

buffer, the data record has been discarded

Description:

The function blocks directly parameterize the pulse generators PTO（Pulse Train Output) and PWM (Pulse Width Modulation Output). Pulse output mode

Is an alternative to digital output. If ENABLE=0 is entered, the process image directly affects the corresponding digital output. If you enter

ENABLE=1, the pulse generator controls the output.

PTO generator (PDO_MODE=1, pulse counter output, one-shot pulse train): The PTO generator creates a one-shot pulse train to control the digital output. A pulse train is described by a parameter set that includes the cycle duration (initial value) , the increment of the cycle duration (the change in value between two consecutive pulses), and the number of pulses to generate. The pulse width is set to 50% of the period duration (sampling ratio 1:1). Considering DELTA, the cycle duration Tn is calculated as follows: Tn=(CT+n*DELTA)*tB (0<=n<=PC)

If tB=1ms (tB_IDX:=1) and CT:=1000 , the initial cycle duration (n=0) is: Tn=1ms*1000=1 second (variation of period duration) and pulse trains of pulse number values are connected . To do this, the function block needs to be called with APPEND=1, making each parameter set expandable.

This extends up to 255 parameter sets. until ERROR=32 (no space available in the data record buffer, the data record has been discarded) . All parameter sets are based on the same time criteria. The time standard (input TB_IDX) can only be changed when the pulse generator is deactivated. Time standard indexing is only started when a rising edge is detected at input ENABLE . If the pulse train has been completely transmitted and no further parameterization is available, the pulse generator PTO is automatically switched off and the process image controls the digital outputs again. Therefore, the PLC program must store in the process image the desired state of the digital outputs after the pulse generator is deactivated. The PTO generator also automatically shuts down if an overflow or underflow error occurs during subsequent pulses . This is the case if Tn is greater than 65535 or less than 0 . The function block displays ERROR=32 (overflow error). Since it is an accumulation operation including DELTA, the result of ERROR is obtained after a series of consecutive successful operations.

PWM generator (PDO_MODE=0, pulse duration output, continuous pulse train): In the PWM generator function, a continuous pulse train is generated at the digital output. In this case, the cycle duration and pulse duration can be set to the number of basic cycles.

If the input ENABLE is 1, the PWM generator is activated directly after the parameters are acquired. If the pulse duration PT has a value of 0, the individual outputs remain inactive for the entire cycle duration. However, if the value of the pulse duration PT is greater than or equal to the cycle duration, the output is active for the entire cycle duration. The continuous duty time is always changed asynchronously, and the current duty state is interrupted by receiving a new value. The duration of the pulses is varied synchronously and accepted at the beginning of the next cycle duration. The parameters are changed by calling the function block with APPEND=1, and the generation of continuous pulse trains is terminated with ENABLE=0. If an error occurs during execution , the function block automatically shuts down. The function block can only be reused after it has been reset with ENABLE=0.

output READY indicates that the function block is fully parameterized and enters the pre-operational state, the parameter TB_IDX (used to set the index of the basic period of the pulse generator) is not changed (TB_IDX is only read when a rising edge is detected at input ENABLE), if the function block is called with ENABLE=0, READY=0 is output.

If the function block output BUSY=1, it means that the pulse generator is activated and controls the corresponding digital output (PTO mode: parameterized pulse train is transmitted, PWM mode: continuous pulse train is generated). BUSY=0 means the generator is not activated and is controlled by the PLC program

The corresponding digital outputs are directly influenced via the process image.

Possible error messages during function block execution are displayed in ERROR.

The example program below shows the application of the function block to generate a one-shot pulse train in PTO mode and a continuous pulse train in PWM mode . Due to the particularity of parameter selection, no additional techniques or test equipment are required to observe the LED light state of the pulse train at the output PWM . The PTO cycle mode is started by a rising edge at the output xStartButton . The burst starts with a 1s pulse (CT*TB=1000*1ms=1s), each subsequent pulse is shortened by 50ms (Delta=-50). A total of 15 pulses are generated (PC=15) . PWM mode is started by a rising edge at input xStartButtonPwm . The resulting burst duration is 500ms (CT*TB=500*1ms=0.5s->2Hz), the on-time of each pulse is 150ms

(PT*TB=150*1ms=150ms).

Sample program:

```
VAR   CONSTANT

(* Definition of TimeBase-Index*) PTO_TB_IDX_800_US:USINT:=0;      (* TimeBase-Index 800us *)
PTO_TB_IDX_1_MS:USINT:=1;    (* TimeBase-Index 1ms    *) (* Error Codes of FB PTO_TAB *)
PTOTAB_ERROR_SUCCESS:SINT=0;

PTOTAB_ERROR_HW_ERROR: USINT:=1;

PTOTAB_ERROR_UNKNOWN_CHANNEL:USINT:=2; PTOTAB_ERROR_UNKNOWN_TB_IDX:USINT:= 8;
PTOTAB_ERROR_DELTA_OVERFLOW:USINT:=16; PTOTAB_ERROR_INVALID_TAB:USINT:=64;
PTO_PWM_CHANNEL:USINT:=0;

END_VAR VAR

xStartButtonPto AT %IX0.0 : BOOL; xStartButtonPwm AT %IX0.1 : BOOL; xPtoPwmOut AT %QX2.4 :
BOOL;

FB_RTrigPto    : R_TRIG; FB_RTrigPwm    : R_TRIG; usiPtoTbIdx: USINT := 1; uiPtoCt  : UINT    := 1000;

iPtoDelta  : INT      := -50;
```

udiPtoPc : UDINT := 15; usiPwmTbIdx : USINT := 1; uiPwmCt : UINT := 500;

uiPwmPt : UINT := 150;

xPtoAppend : BOOL := TRUE; xPtoReady : BOOL := FALSE;

```
xPtoBusy : BOOL := FALSE;
xPwmAppend : BOOL := TRUE; xPwmReady : BOOL := FALSE;
xPwmBusy : BOOL := FALSE;
FB_PtoPwm     : PTO_PWM; usiPtoPwmError : USINT;
END_VAR

(* ----- Wait for Start------*) WaitForStart:
CAL    FB_RTrigPto
LD    FB_RTrigPto.Q JMPC   StartPtoMode CAL     FB_RTrigPwm
LD     FB_RTrigPwm.Q JMPC     StartPwmMode LD     xPtoBusy
JMPC RunPtoMode
LD      xPwmBusy
JMPC RunPwmMode
JMP    ProgExit

(* ----- Run PTO Mode------*)
StartPtoMode:
LD          FALSE (* preset output state, this state is     *) ST          xPtoPwmOut          (* used when PTO Generator isn't running *) LD          FALSE (* reset state flags *)
ST      xPtoReady
ST      xPtoBusy
ST      xPwmReady
ST      xPwmBusy
```

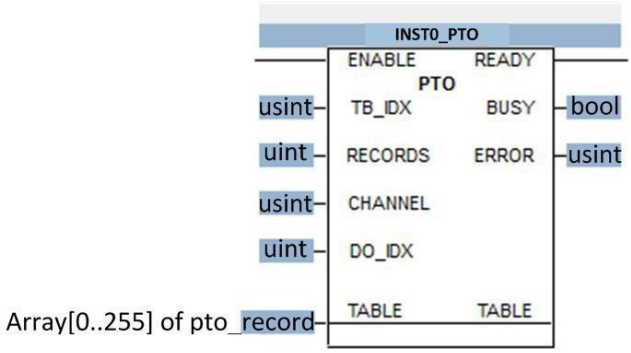
```
CAL    FB_PtoPwm ( ENABLE := FALSE,
CHANNEL := PTO_PWM_CHANNEL)
CAL    FB_PtoPwm ( ENABLE := TRUE, PTO_MODE := TRUE,
APPEND := xPtoAppend, TB_IDX := usiPtoTbIdx, CT := uiPtoCt,
DELTA := iPtoDelta, PC := udiPtoPc,
CHANNEL := PTO_PWM_CHANNEL
|
xPtoReady := READY, xPtoBusy := BUSY, usiPtoPwmError := ERROR)

RunPtoMode:
CAL    FB_PtoPwm ( ENABLE := TRUE, PTO_MODE := TRUE, APPEND := FALSE,
CHANNEL := PTO_PWM_CHANNEL
|
xPtoReady := READY, xPtoBusy := BUSY, usiPtoPwmError := ERROR)
JMP    ProgExit

(* ----- Run PWM Mode------*)
StartPwmMode:

                FALSE                   preset output state, this state is
        D                       *                                       )
                xPtoPwmO                used when PTO Generator isn't
            ut                      running
        T                       *                                       )
```

```
                FALSE                   reset state flags *)
        D                       *

                xPtoReady
        T

                xPtoBusy
        T



ST      xPwmReady

ST      xPwmBusy

CAL     FB_PtoPwm ( ENABLE := FALSE,

CHANNEL := PTO_PWM_CHANNEL)

CAL     FB_PtoPwm    ( ENABLE := TRUE, PTO_MODE := FALSE,

APPEND := xPwmAppend, TB_IDX := usiPwmTbIdx, CT := uiPwmCt,

PT := uiPwmPt,

CHANNEL := PTO_PWM_CHANNEL

|

xPwmReady := READY, xPwmBusy := BUSY, usiPtoPwmError := ERROR)

RunPwmMode:

CAL     FB_PtoPwm    ( ENABLE := TRUE, PTO_MODE := FALSE, APPEND := FALSE,

CHANNEL := PTO_PWM_CHANNEL

|

xPwmReady := READY, xPwmBusy := BUSY, usiPtoPwmError := ERROR)

JMP     ProgExit
```

(* ----- Cycle End ----- *) ProgExit:

RET END_PROGRAM

3.3.85 PTO



用于执行脉冲计时器。典型的应用例子是控制步进电机的单脉冲，实现步进电机控制的斜坡函数

Definition of operands:

TABLE： parameter setting table, containing the pulse sequence to be generated;

ENABLE： enable or disable the function block, true to enable,

false To disable;; **TB_IDX：** set the reference period for the

pulse generator, 0=800ns, 1=1ms; **RECORDS：** the address in

the memory, used to read and write data (need to set the

parameter mode); **CHANNEL：** the channel to be used No;

DO_IDX： device number, needs to be set to 0;

TABLE： read one byte of data;

READY： Indicates the output state of the pulse generator, True represents initialized, False represents uninitialized;

BUSY： Indicates the output state of the pulse generator, True means the pulse generator is busy, False means the pulse generator is idle;

ERROR： The error code describes information about the result of the execution of the function block. Possible error codes are defined as follows :0:No error occurred while executing the function block

1: A hardware error occurred while executing the function block

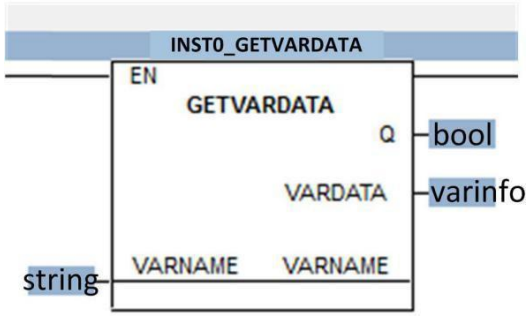
2: When calling the function block, an invalid device number was set

4: When calling the function block, an invalid mode was set

8: The set address is too large and exceeds the maximum available storage area

16: The pointer points to an unsupported data type

3.3.86 GETVARDATA



Definition of operands:

EN： Enable input

VarName： String name of the

variable to be accessed **Q：**

Output 1 if the variable is

valid **VarData：** Valid variable

information

describe:

The variable specified at the input is located in the memory address space and information about the variable is displayed at the output, if the variable cannot be addressed, the output Q is 0. OpenPCS must have the function of addressing by name. The premise of this function is to generate a mapping file in the resource option.

3.3.87 GETVARFLATADDRESS

查询变量地址

Definition of operands:

EN： Enable input.

VARNAME： The string name of the variable to be queried. **Q：** Output 1 if the variable is valid .

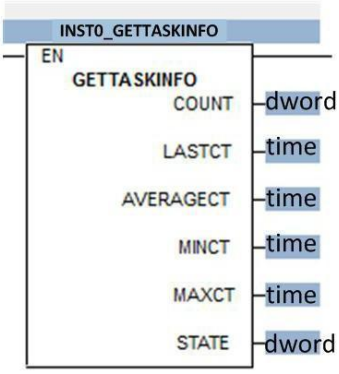
VARADDRESS： The address of the variable to be queried.

describe:

If the variable specified at the input is in the memory address space, the address of the memory space is output. If the variable is not in the memory address space, the output Q will be 0

In order for OpenPCS to be addressable by name, a memory-mapped file must be generated (in the resource options). The memory address space can only be used in the current program cycle and cannot be stored or called by other programs.

3.3.88 GETTASKINFO



获取任务周期各执行时间

Definition of operands:

COUNT： the number of cycles

to perform this task; **LASTCT：**

the time required for the

previous cycle;

AVERAGECT： Average time required for execution; MINCT：Minimum

time required for execution; **MAXCT：** Maximum time required for

execution;

STATE： not yet

used; description:

This function block returns information about the execution time of the last cycle of the current task. This function block has no input parameters.

4. Explanation of basic function application routines

Since the previous article has introduced how to open program files, compile programs, download programs, etc., this chapter will not describe in detail. The content of this chapter only explains the programming ideas, programming logic and functions implemented by the routines.

1. Explanation of Marquee Experiment Routine

```
VAR
    v1:INT:=0;          (*定义整形变量，初始值为0*)
    v2:INT:=0;
    oled at%Q0.0:Byte;  (*输出Q0.0~Q0.7共8位*)
END_VAR
```

```

IF v1<1000 THEN         (*如果v1<1000,则自加1*)
    v1:=v1+1;

ELSE                    (*如果v1=1000,则把v1赋值为0*)
    v1:=0;              (*v2自加1*)
    v2:=v2+1;
    if v2>=255 then     (*如果v2加到255，则把v2赋值为0*)
        v2:=0;
    end_if;
    oled:=int_to_byte(v2);  (*将数据类型为int的v2转化为byte赋值给oled*)

end_if;
```

In this program, the initial value of the V1 variable is 0. When V1 is less than 1000, V1 is incremented by 1. After 1000 scan cycles, the value of V1 is 1000. Enter ELSE, V1 returns to 0, and V2 starts to increment by 1. Return to zero, so the next cycle continues to enter V1 and increments by 1. The actual effect of this program is that every time V1 is added to 1000, V2 is incremented by 1.

When V2 is added to 255 each time, V2 is reset to zero, and the data type of V2 is converted to BYTE and assigned to oled every cycle

to output the marquee.

data conversion is performed because BYTE cannot directly perform mathematical operations, so it is necessary to use INT to add to realize the phenomenon of marquee.

2. Input and output experiment

```
    DI0 AT%I0.0:BOOL; (* 数字量输入1 *)
    DI1 AT%I0.1:BOOL; (* 数字量输入2 *)
    DI2 AT%I0.2:BOOL; (* 数字量输入3 *)
    DI3 AT%I0.3:BOOL; (* 数字量输入4 *)
    DI4 AT%I0.4:BOOL; (* 数字量输入5 *)
    DI5 AT%I0.5:BOOL; (* 数字量输入6 *)
    DI6 AT%I0.6:BOOL; (* 数字量输入7 *)
    DI7 AT%I0.7:BOOL; (* 数字量输入8 *)

    DO0 AT%Q0.0:BOOL; (* 数字量输出1 *)
    DO1 AT%Q0.1:BOOL; (* 数字量输出2 *)
    DO2 AT%Q0.2:BOOL; (* 数字量输出3 *)
    DO3 AT%Q0.3:BOOL; (* 数字量输出4 *)
    DO4 AT%Q0.4:BOOL; (* 数字量输出5 *)
    DO5 AT%Q0.5:BOOL; (* 数字量输出6 *)
    DO6 AT%Q0.6:BOOL; (* 数字量输出7 *)
    DO7 AT%Q0.7:BOOL; (* 数字量输出8 *)

    AI0 AT%I0.0:int; (* 模拟量输入1 *) (* int形式的变量用于定义-32767~+32767的输入 *)
    AI1 AT%I2.0:int; (* 模拟量输入2 *)
    AI2 AT%I4.0:int; (* 模拟量输入3 *)
    AI3 AT%I6.0:int; (* 模拟量输入4 *)

    AO0 AT%Q1.0:uint; (* 模拟量输出1 *) (* uint形式的变量用于定义0~+65535的输出 *)
    AO1 AT%Q3.0:uint; (* 模拟量输出2 *)

    AO2 AT%Q5.0:int;  (* 模拟量输出3 *) (* int形式的变量用于定义-32767~+32767的输出 *)
    AO3 AT%Q7.0:int;  (* 模拟量输出4 *)

    v1:bool;(* 中间变量1 *)
    v2:bool;(* 中间变量2 *)
```

```
DO0:=DI0; (* 数字量输出=数字量输入 *)
DO1:=DI1;
DO2:=DI2;
DO3:=DI3;
DO4:=DI4;
DO5:=DI5;
DO6:=DI6;
DO7:=DI7;

if AI0>5000 then (* 电压/电流输入换算值大于5000 *)

 v1:=1;
 v2:=0;

else              (* 电压/电流输入换算值小于等于5000 *)

 v1:=0;
 v2:=1;

end_if;

AO0:=65535;       (* 输出换算值为65535的电压/电流 *)
AO1:=50000;       (* 输出换算值为50000的电压/电流 *)

AO2:=32767;       (* 输出换算值为32767的电压/电流 *)
AO3:=-32767;      (* 输出换算值为-32767的电压/电流 *)
```

```
(*IO模块定义表*)
(*1008  8路数字量输入*)
(*2008  8路数字量输出*)
(*3604  4路模拟量输入-5~+5V   16位即-32768~+32767*)
(*3624  4路模拟量输入-10~+10V 16位即-32768~+32767*)
(*3644  4路模拟量输入0~+20mA  16位即0~+65535*)
(*3654  4路模拟量输入4~+20mA  16位即0~+65535*)
(*3664  4路模拟量输入0~+5V    16位即0~+65535*)
(*3674  4路模拟量输入0~+10V   16位即0~+65535*)
(*4602  2路模拟量输出-5~+5V   16位即-32768~+32767*)
(*4622  2路模拟量输出-10~+10V 16位即-32768~+32767*)
(*4642  2路模拟量输出0~+20mA  16位即0~+65535*)
(*4652  2路模拟量输出4~+20mA  16位即0~+65535*)
(*4662  2路模拟量输出0~+5V    16位即0~+65535*)
(*4672  2路模拟量输入0~+10V   16位即0~+65535*)
(*1502  2路计数器输入         32位即0~4294967295  注：计数和频率均为32位*)
(*2204  4路继电器输出         8位                 注：2204位宽为8位，但只用到了前4位，后4位为空*)
(*2302  2路PWM输出  具体应用参考2302使用例程*)
```

In this program, the variable declaration area reflects how the variable with the hardware address of the IO module is declared, and the program editing area reflects the use of simple statements, as well as the corresponding data type and value range of the IO module.

3. CAN Send and receive data experiment ST

```
(*该程序实现了将每个读到的CAN帧信息在CANID加1后重新写出去*)


canCH:=2;                                   (*使用的CAN端口号，PLC-400/510的CAN端口号为2，PLC-core的CAN端口号为1或2*)

if not mCanInit then                        (*如果未初始化CAN口*)

      inst0_CAN_INIT(EN_IN :=1 ,            (*PLC的CAN口初始化功能块，EN_IN为使能；NETNUMBER为端口号，PLC-400/510为2；初始化功能块只在上电第一个周期启动，若*)
       NETNUMBER :=canCH ,
       BITRATE :=0
       xERROR:= ERROR);                     (*BITRATE: = 0表示1 MBit/s   BITRATE: = 1表示840 kBit/s   BITRATE: = 2表示700 kBit/s   BITRATE: = 3表示500 kBit/s   BITRATE: = 4表示400 kBit/s*)
       if xERROR=0 then                     (* BITRATE: = 5表示250 kBit/s   BITRATE: = 6 表示200 kBit/s   BITRATE: = 7 表示125 kBit/s   BITRATE: = 8 表示100 kBit/s   BITRATE: = 9 表示80 kBit/s*)
       mCanInit:=true;                      (* BITRATE: = 10表示50 kBit/s   BITRATE: = 11表示40 kBit/s   BITRATE: = 12表示20 kBit/s   BITRATE: = 13表示10 kBit/s*)
      end_if;                               (*如果CAN初始化功能块执行过程无错误，则CAN初始化完成标志置为1*)

else

      inst1_CAN_MESSAGE_READ8(EN_IN :=can_read  (*如果CAN初始化成功，则启动CAN读功能块*)
     , NETNUMBER := canCH
     mCanConfirm := EN_OUT                  (*CAN读功能块读到CAN帧数据的标志*)
```

This section of the program is a CAN initialization function block. The use of the function block has been explained in Chapter 3, and will not be described in detail in this

section. The parameters for adjusting the baud rate can be seen in the program comments. The logic of this program is that if the CAN initialization completion flag is FALSE executes initialization, the initialization ERROR is 0, the flag bit becomes TRUE, and the next time it judges IF, it enters the ELSE program segment.

```
16
17else
18
19      inst1_CAN_MESSAGE_READ8(EN_IN :=can_read  (*如果CAN初始化成功，则启动CAN读功能块*)
20      , NETNUMBER := canCH |
21      mCanConfirm := EN_OUT,                    (*CAN读功能块读到CAN帧数据的标志*)
22      mCANID := CANID,                          (*将读到的CANID存放到变量中*)
23      mEXT_FRAME := EXT_FRAME,                  (*判断读到的CAN帧是否为扩展帧，若为扩展帧则值为1，若为标准帧则值为0*)
24      mRTR_FRAME := RTR_FRAME,                  (*判断读到的CAN帧是否为远程帧，若为远程帧则值为1，若为数据帧则值为0*)
25      mDATALENGTH := DATALENGTH,                (*读到的CAN帧数据长度，即包含多少字节*)
26      mDATA0:= DATA0,                           (*将读到的CAN数据第1个字节存放到变量中*)
27      mDATA1:= DATA1,                           (*将读到的CAN数据第2个字节存放到变量中*)
28      mDATA2 := DATA2,                          (*将读到的CAN数据第3个字节存放到变量中*)
29      mDATA3:= DATA3,                           (*将读到的CAN数据第4个字节存放到变量中*)
30      mDATA4:= DATA4,                           (*将读到的CAN数据第5个字节存放到变量中*)
31      mDATA5:= DATA5,                           (*将读到的CAN数据第6个字节存放到变量中*)
32      mDATA6 := DATA6,                          (*将读到的CAN数据第7个字节存放到变量中*)
33      mDATA7:= DATA7,                           (*将读到的CAN数据第8个字节存放到变量中*)
34      xERROR:= ERROR);                          (*CAN读功能块执行过程中的错误信息*)
35
36      if mCanConfirm=1 then                     (*只有在CAN读完成标志为1时读到的ID和数据才是实际值*)
37       canID:=mCANID;                           (*获取实际的CANID*)
38       canDAT0:=mDATA0;                         (*获取实际CAN数据的第1个字节*)
39       canDAT1:=mDATA1;                         (*获取实际CAN数据的第2个字节*)
40       canDAT2:=mDATA2;                         (*获取实际CAN数据的第3个字节*)
41       canDAT3:=mDATA3;                         (*获取实际CAN数据的第4个字节*)
42       canDAT4:=mDATA4;                         (*获取实际CAN数据的第5个字节*)
43       canDAT5:=mDATA5;                         (*获取实际CAN数据的第6个字节*)
44       canDAT6:=mDATA6;                         (*获取实际CAN数据的第7个字节*)
45       canDAT7:=mDATA7;                         (*获取实际CAN数据的第8个字节*)
46       can_read:=0;
47       can_write:=1;                            (*CAN成功读取后，将读指令赋值0，写指令赋值1*)
48      end_if;
49
50
51    if can_write=1 then                         (*如果写指令为1，则将之前读到的CANID加1*)
52        mCANID:=mCANID+1;
53    end_if;
54
```

Enter ELSE After the program segment, execute CAN_READ , CAN Read function block, where mDATA0-7 For the read data, only the actual data is read at the moment of reading, and the rest of the time is a random number, mCanConfirm In order to read the complete flag bit, it is only set to TRUE at the moment of reading the data, and it returns to FALSE immediately in the next cycle , so a judgment is made when the flag bit is 1 In that cycle, immediately assign the read data to another variable to save the data, turn off the read enable, and turn on the write enable. write enable as 1 Add CANID+1 to avoid writing out the ID The same as receiving data frame collision. (Note: It is the logic of the routine to turn off the read enable here, CAN Read function block enable is always set to 1 Can be read all the time.)

```
50
51    if can_write=1 then                       (*如果写指令为1，则将之前读到的CANID加1*)
52        mCANID:=mCANID+1;
53    end_if;
54
55        inst2_CAN_MESSAGE_WRITE8(             (*调用CAN写功能块将读到的CAN帧数据的CANID加1后写出去*)
56        EN_IN :=can_write ,                   (*CAN写功能块的使能条件为can_write=1*)
57        NETNUMBER :=canCH ,                   (*PLC-400/510的CAN写功能块端口号为2，PLC-core的CAN写端口号为1或2*)
58        CANID :=mCANID ,                      (*将CAN读到的帧ID加1写出去，将读和写的帧ID作区分*)
59        EXT_FRAME := mEXT_FRAME,              (*CAN写的帧信息，1位扩展帧，0为标准帧*)
60        RTR_FRAME := mRTR_FRAME ,             (*CAN写的帧信息，1为远程帧，0为数据帧*)
61        DATALENGTH :=mDATALENGTH ,            (*CAN写的帧长度，CAN数据的字节数*)
62        DATA0 :=mDATA0 ,                      (*CAN写数据的第1个字节*)
63        DATA1 :=mDATA1 ,                      (*CAN写数据的第2个字节*)
64        DATA2 :=mDATA2 ,                      (*CAN写数据的第3个字节*)
65        DATA3 :=mDATA3 ,                      (*CAN写数据的第4个字节*)
66        DATA4 :=mDATA4 ,                      (*CAN写数据的第5个字节*)
67        DATA5 :=mDATA5 ,                      (*CAN写数据的第6个字节*)
68        DATA6 :=mDATA6 ,                      (*CAN写数据的第7个字节*)
69        DATA7 :=mDATA7 |                      (*CAN写数据的第8个字节*)
70        mCONFIRM := EN_OUT,                   (*CAN写成功的标志位*)
71        xERROR:= ERROR);                      (*CAN写过程中的错误信息*)
72
73     if mCONFIRM=1 then                       (*如果CAN写成功后，将CAN读指令置为1，重新读取CAN帧信息；将CAN写指令置为0，关闭写通道*)
74
75        can_read:=1;                          (*程序在下一个扫描周期重新执行先读后写的过程*)
76        can_write:=0;
77     end_if;
78
79end_if;
80
81
```

CAN read program segment is processed and the CAN ID+1 is completed, the CAN write function block is entered. The enable trigger condition of this function block is the rising edge trigger. For the meaning of the rising edge trigger, please refer to page 39 of this book, and this section will not be repeated. To reiterate, the CAN write function block sends out the data after identifying the rising edge of the enable variable . When the frame data is sent, the CAN write success flag will be set to 1 (same as the CAN read success flag).

Only when this cycle is 1, the next cycle will change back to 0), when the flag bit is 1, the CAN write enable is turned off so that the rising edge is triggered next time, and the CAN read enable is turned on and read.

4. Serial RS232, 485 Experimental ST

```
(*本示例程序实现了串口数据的先读后写，且写的内容为先前读到的数据*)

mPORT:=2;  (*串行端口号，1为RS232,2为RS485*)

if xInitOk=false then  (*如果未初始化串口*)
inst0_USART_INIT(BAUD :=115200 , DATABITS :=8 , PARITY :=0 , STOPBITS :=1 , PROTOCOL :=0 , ENABLE :=1 , PORT :=mPORT  | xE := ERROR);  (*启用串口初始化功能块，BAUD为波特率，
                                                                                                                                    DATABITS为数据位数，PARITY为奇偶校验，0表示无校验*)
    if xE=0 then       (*如果串口初始化功能块执行无错误*)
        xInitOk :=TRUE; (*将串口初始化状态置1，不再执行串口初始化*)

    end_if;

end_if;
```

This routine realizes that the data received by the serial port is sent back intact. This program is serial port initialization. The serial port initialization function block parameters used have been explained in Chapter 3. The meaning of each parameter can also be seen in the comments. The meaning of the segment program is that if the initialization is not completed, the initialization function block will be called, and if the initialization has no errors, the initialization will be completed.

```
pDataObject:=&abDataBuffer;  (*指针指向数组*)

if xInitOk then    (*在初始化成功前提下*)


    if xReadStart then  (*如果启动串口读取模式*)

        inst0_USART_READ_BIN(ENABLE :=0 ,  PORT :=mPORT  |   xERROR:= ERROR); (*以使能ENABLE为0调用串口读取功能块，与接下来以ENABLE为1调用功能块相配合，生成使能ENABLE的上升沿指令*)
        xWaitForReceipt:=true;  (*置位串口预读取标志*)
        xRdBinConfirm:=false;   (*复位串口读取功能块执行完成标志*)
        xReadStart:=false;      (*复位串口模式的启动指令*)
    end_if;

    if xWaitForReceipt then     (*如果串口预读取标志为1*)
        inst0_USART_READ_BIN(ENABLE :=1 , PTR_RXDATA :=pDataObject , MAXLENGTH :=0 , ETXCHR :=10, CHKETX :=1, PORT :=mPORT  |  xRdBinConfirm:= CONFIRM,  iRxDataSize:= RXLENGTH,  xERROR:= ERROR);
                            (*以使能ENABLE为1调用功能块，将读取到的数据存放到数组abDataBuffer中，读取的最大长度为20个字节，实际读取长度赋值到iRxDataSize中*)
        if xRdBinConfirm then   (*如果串口读取功能块执行完成*)
            var4:=iRxDataSize;  (*将串口读到的实际数据长度赋值到该变量*)
            xWaitForReceipt:=false;  (*复位串口预读取标志*)
        end_if;

    end_if;

    if xRdBinConfirm then       (*如果串口读取功能块执行完成*)

        inst0_USART_WRITE_BIN(ENABLE :=0 ,  PORT :=mPORT |  xERROR:= ERROR);  (*以使能ENABLE为0调用串口发送功能块，与接下来以ENABLE为1调用功能块相配合，生成使能ENABLE的上升沿指令*)

        xRdBinConfirm:=false;   (*复位串口读取功能块完成标志*)
        xTransmitting:=true;    (*启动串口发送模式*)

    end_if;

    if xTransmitting then       (*如果串口发送模式已启动*)

        inst0_USART_WRITE_BIN(ENABLE :=1 , PTR_TXDATA :=pDataObject , TXLENGTH :=iRxDataSize, PORT :=mPORT | xWrBinConfirm := CONFIRM,  xERROR:= ERROR);
                            (*以使能ENABLE为1调用串口发送功能块，将先前读到数组abDataBuffer中的数据发送出去，发送长度为串口先前读取数据的实际长度*)
        if xWrBinConfirm then   (*如果串口发送完成*)
            IW:=IW+1;
            xTransmitting :=false;  (*关闭串口发送模式*)
            xReadStart:=true;       (*启动串口读取模式，继续先读后写的循环*)
        end_if;

    end_if;

end_if;
```

After the initialization is completed, it enters the serial port working mode. The first line is a pointer variable that points to an array. The pointer variable in ST language can only point to an array. Because the array cannot be used directly in the function block, the pointer pin of the function block needs to point to the array. Call data using an array.

Enter the serial port read mode, when the enable is 1, wait for the data to be read. When the read data is completed, the read completion flag will be set to 1. At this time, the serial port write function block is called (enable is 0), and the call will be enabled after the call is completed . Set to 1 and call the trigger rising edge again to write out the data in the array.

5. TCP SERVER Communication experiment

This routine implements PLC as a TCP SERVER, communicates with the TCP CLIENT device, and sends back the received data intact.

```
(*本示例程序实现了PLC作为TCP_SERVER与TCP_CLIENT通讯，将读取到的数据原样发送出去*)


if lanInit=false then      (*如果网口未初始化*)

mIP := LAN_ASCII_TO_INET('192.168.1.30');          (*将PLC的IP地址转化成ASCII码*)
mNetMask := LAN_ASCII_TO_INET('255.255.255.0');    (*将子网掩码转化成ASCII码*)
mGateWay := LAN_ASCII_TO_INET('192.168.1.1');      (*将网关转化成ASCII码*)
    inst0_LAN_INIT(ENABLE :=true ,                 (*调用网口初始化功能块*)
            HOSTNAME := 'PLCCore',                 (*主机名*)
            IP := mIP,                             (*PLC的IP地址*)
            NETMASK :=mNetMask,                    (*PLC的子网掩码*)
            GATEWAY :=mGateWay ,                   (*PLC的网关*)
            NETNUMBER:=1                           (*网络端口号*)
            |  mConfirm:= CONFIRM,                 (*初始化功能块执行完成标志*)
             mError:= ERROR,                       (*功能块执行完成报的错误信息*)
             mErrorinfo:= ERRORINFO);              (*功能块执行过程中发生终止报的错误信息*)
            lanInit:=true;                         (*将网口初始化状态置1，不再进行网口初始化*)
else                        (*如果网口已初始化*)
```

This section of the program is the network port initialization function block, the meaning of each parameter can be seen in the program comments, among which mIP stands for PLC IP, if not on the PLC IP _If it has been modified, fill in PLC here Factory default IP : 192.168.1.30 That is, if it has been modified, according to the IP modified by the user Fill in

```
            lanInit:=true;                         (*将网口初始化状态置1，不再进行网口初始化*)
else                        (*如果网口已初始化*)

    if mTcpCreateOK=false then   (*如果未建立TCP_SERVER*)
        inst4_LAN_TCP_SERVER_CREATE(PORT :=8089 ,  (*生成一个TCP_SERVER,端口号8089*)
         ENABLE := true,
         NETNUMBER := 1                            (*网络端口号为1*)
         |
         mSocket:= SOCKET_ID,                      (*mSocket存放SOCKET_ID*)
         mConfirm:= CONFIRM,
         mError:= ERROR,                           (*执行完成报的错误信息*)
         mErrorinfo:= ERRORINFO);                  (*执行过程中终止报的错误信息*)

         if mSocket>=0 then                        (*如果SOCKET_ID成功生成，表示TCP_SERVER成功建立*)
         mTcpCreateOK:=true;                       (*置位TCP_SERVER建立成功标志，后续不再建立TCP_SERVER*)
         end_if;

    end_if;

end_if;
```

(Note: This is not an IP How much to fill, PLC device IP will be modified to how much, if you need to modify the IP, please contact the technical engineer of Guangcheng Technology). The subnet mask and gateway can also be filled in according to the user's networking requirements.

After the initialization is successful, create a TCP SERVER, you can set the port number,

```
pDataObject:=&abDataBuffer;     (*指针指向网口读写数据的地址*)

if mTcpCreateOK=true then       (*如果TCP_SERVER建立成功*)


    inst5_LAN_GET_TCPCONNECT_SOCKET(SOCKET_ID := mSocket, NETNUMBER := 1 |  mClientSocket:= CLIENT_SOCKET_ID,  mPeer:= PEER_ADDR,  mPeerPort:= PEER_PORT,  mError:= ERROR);
    (*建立TCP_SERVER与TCP_CLIENT的SOCKET连接,SOCKET_ID为先前生成的ID，mClientSocket用于存放TCP_CLIENT的SOCKET_ID，mPeer存放网络对端地址，mPeerPort存放网络对端的端口号*)

    if mClientSocket>=0 then   (*如果TCP_CLIENT的SOCKET_ID获取成功*)

        inst6_LAN_TCP_RECV_BIN(CLIENT_SOCKET_ID :=mClientSocket ,   (*调用网口读取数据功能块，指针指向数据存储地址，最大读取长度为1500，实际读取长度存到mRxLen中*)
            PTR_RXDATA :=pDataObject ,
            MAXLENGTH := 1500,
            ENABLE :=1 ,
            NETNUMBER := 1 |  mRxLen:= RXLENGTH,  mConfirm:= CONFIRM,  mError:= ERROR,  mErrorinfo:= ERRORINFO);

        if mRxLen>0 then       (*如果网口读取数据成功*)
            tmplen:=mRxLen;
            inst7_LAN_TCP_SEND_BIN(CLIENT_SOCKET_ID :=mClientSocket ,  (*调用网口发送数据功能块，数据内容与读取到的内容相同，数据长度为实际读到数据的长度*)
                PTR_TXDATA :=pDataObject ,
                TXLENGTH := mRxLen,
                ENABLE :=1 ,
                NETNUMBER :=1  |  mConfirm:= CONFIRM, mError := ERROR,  mErrorinfo:= ERRORINFO);

        end_if;
    end_if;
end_if;
```

after the creation, the SOCKET ID will be greater than 0, then jump out of the creation and enter the main program.

This section of the program is the main program of the routine. First, establish a connection with the TCP CLIENT. When there is a CLIENT connection, mClientSocket will be greater than 0. At this time, TCP read is called to read the data sent by the CLIENT and save it in the pointer pointed to by the In the array, when the length of the received data is greater than 0, the TCP write function block is started, and the data in the array pointed to by the pointer is sent out. (For details of the video explanation, see https://www.bilibili.com/video/BV1Sq4y1E7Cn?spm_id_from=333.999.0.0 This link explains the case of TCP SERVER/TCP CLIENT and UDP communication, so this article will not go into details).