序　言

感谢您选用 EC400 系列 PLC 产品！

本公司以：

完美的质量，

竭诚的服务，

给您最真挚的回报。

EC400 系列 PLC 由欧瑞传动电气股份有限公司自主设计与研发，融合国际主流 PLC 的成功经验，改进其不足之处、瞄准当今 PLC 的最新发展方向，采用计算机、通信、电子和自动控制等领域的最新技术，在 CPU 性能、I/O 信号处理、现场总线通讯、软件开发及生产工艺等方面都具有优良性能。其精度及采样速度功能都极大的提升，其组网的灵活性、系统平台的开放性、编程软件的标准性以及智能性可使复杂的控制过程得以完美。

目录

第一章 EURA Studio 软件使用入门

1.1 EURA Studio 软件介绍

EURA Studio 软件是欧瑞传动新推出的，面向工业自动化领域的新一代工程软件平台。利用 EURA Studio 软件可以快速实现对 EC400 系列 PLC 设备的编程、程序下载、程序在线监控等功能。

1.1.1 硬件配置

- CPU：主频 133MHz 或更高
- 硬盘：30M 以上空间
- 内存：256M 以上
- 键盘、鼠标、串行通信口（COM 口）或者 USB 口（需另外使用 USB/RS232 转换器）

1.1.2 软件配置

操作系统：Windows 7 以上。

IE 内核：IE8.0 以上。

1.1.3 EURA Studio 的安装与卸载

EC400 系列 PLC，使用的编程软件为 EURA Studio，支持 Win7，Win8 和 Win10 操作系统。

1.1.3.1 安装

双击安装软件，显示安装界面，然后一直点击 Next 直到安装完成。安装软件时建议关闭 360 安全卫士等杀毒软件。

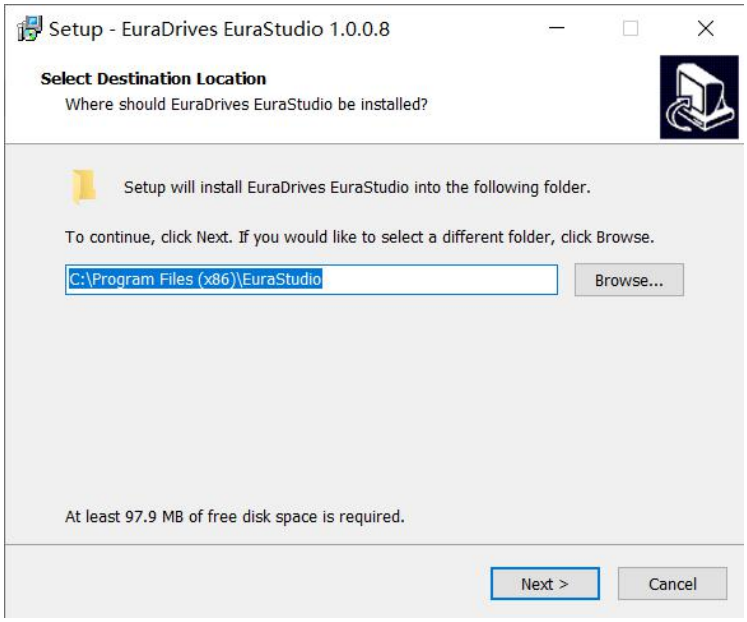


图 1.1 安装界面 1

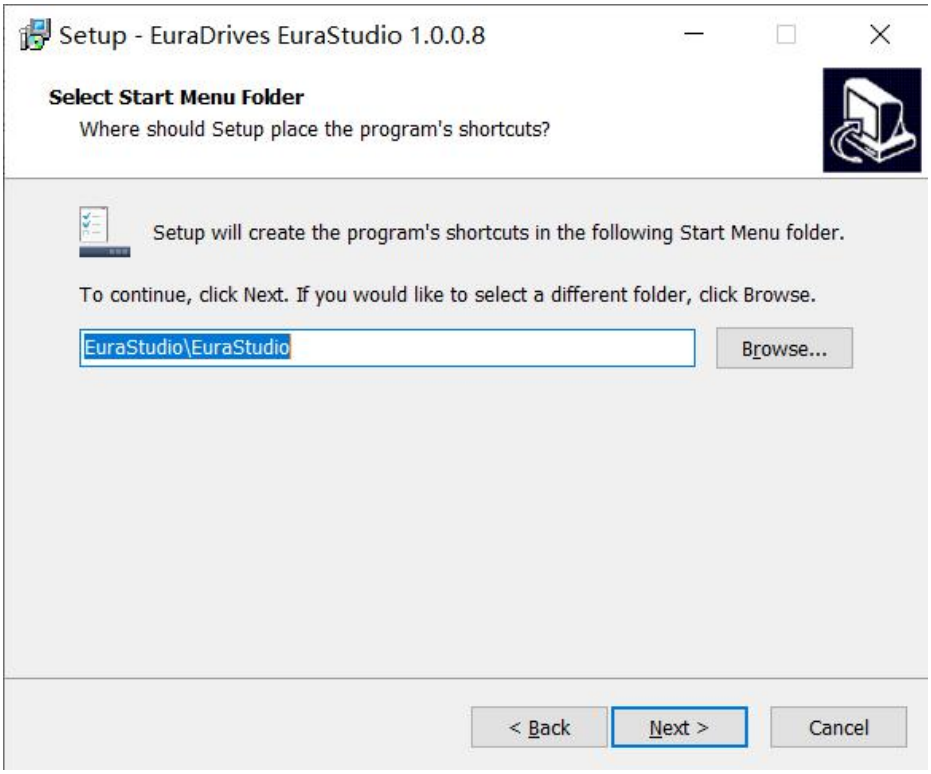


图 1.2 安装界面 2

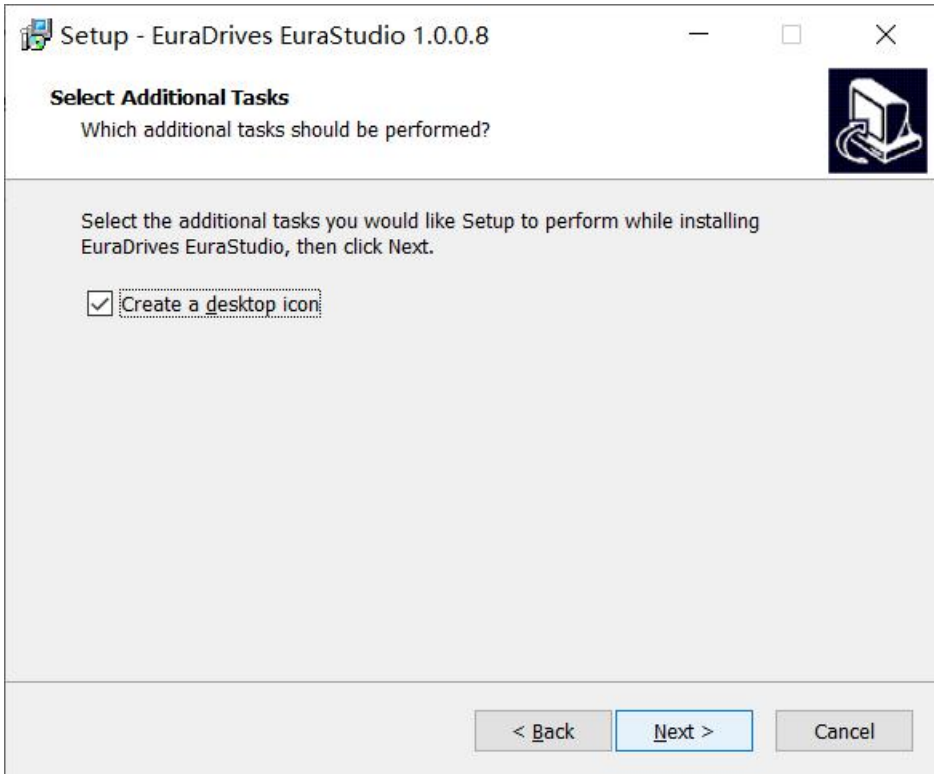


图 1.3 安装界面 3

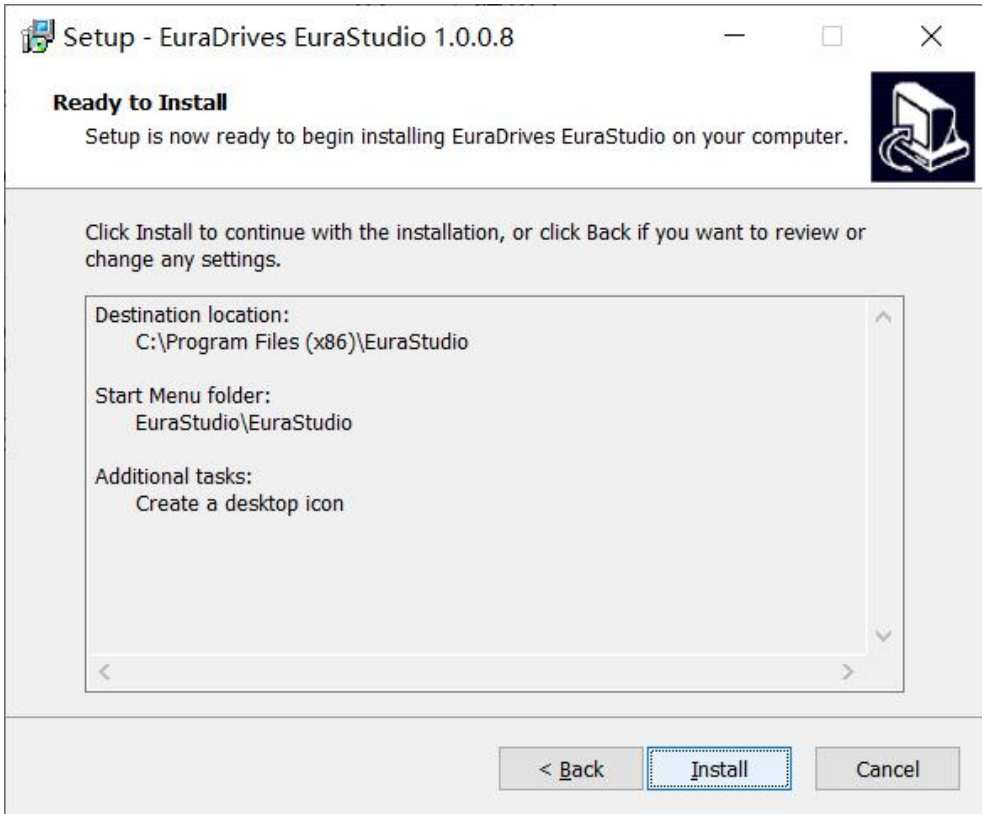


图 1.4 安装界面 4

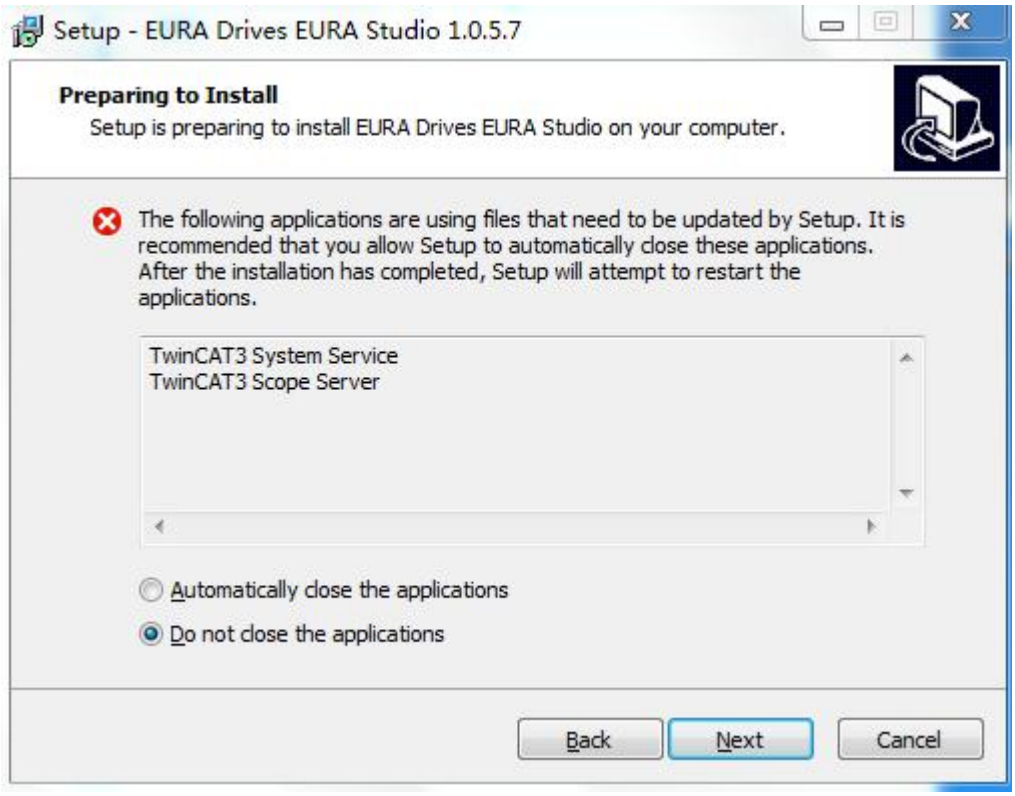


图 1.5 安装界面 5

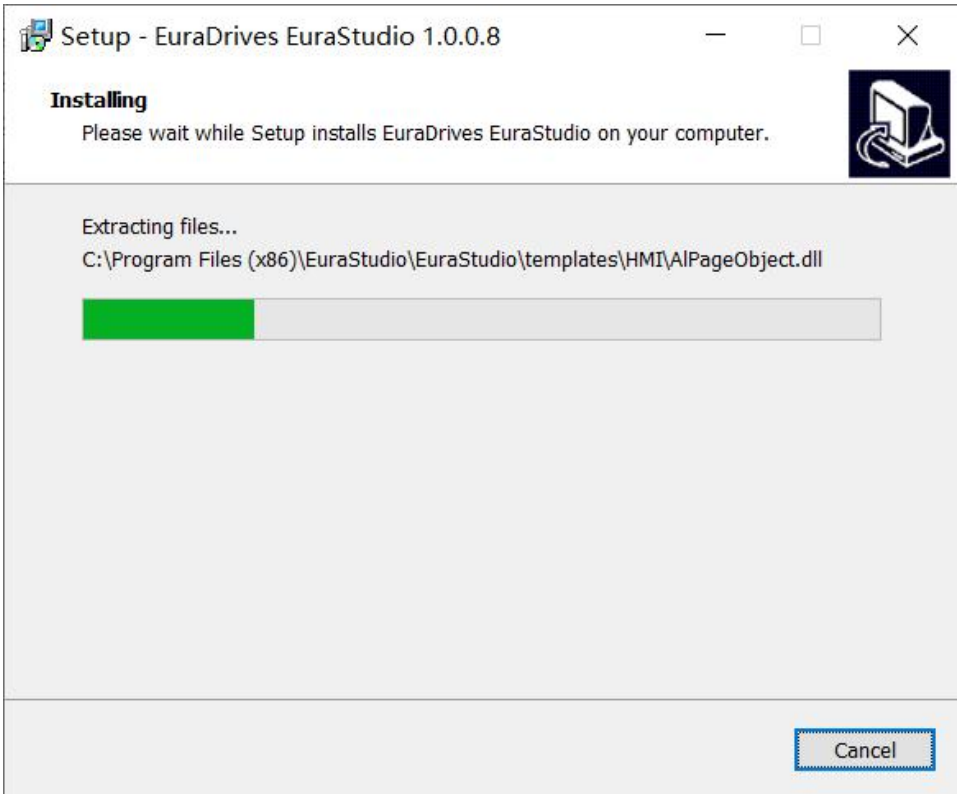


图 1.6 安装界面 6

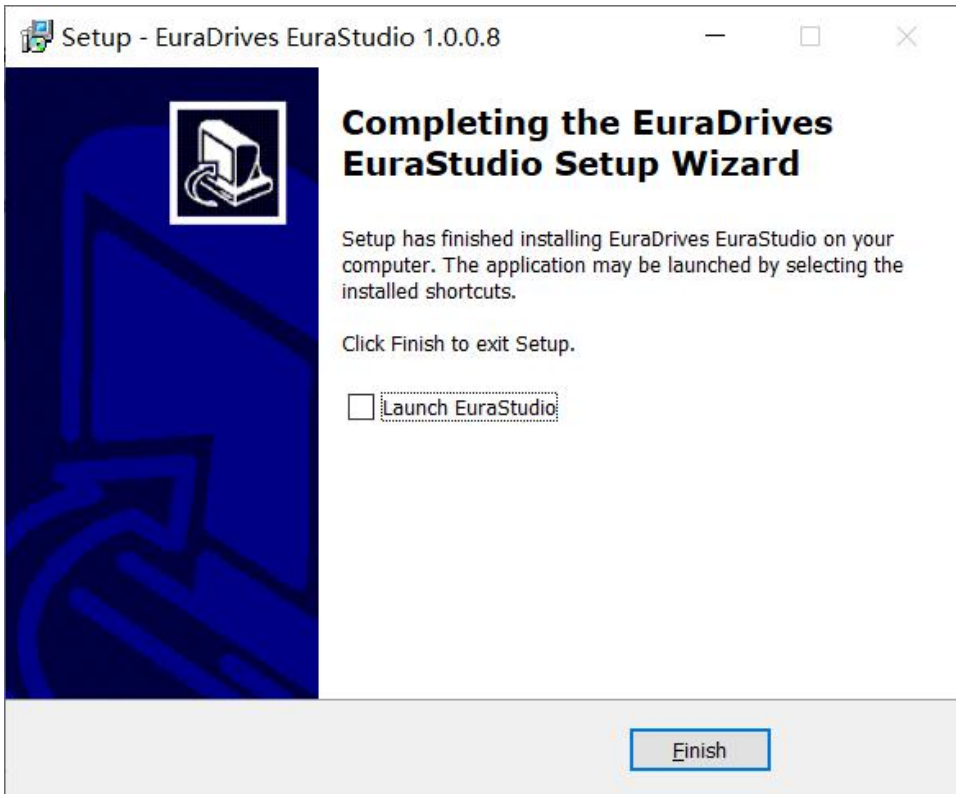


图 1.7 安装界面 7

至此编程软件安装完成，默认情况下将在程序菜单装编程序组，以及在桌面创建快捷方式，可以从这两处位置启动程序。

1.1.3.2 卸载

执行卸载前，请先退出 EURA Studio 程序。卸载 EURA Studio 软件有两种方法：

1) 执行【开始】→【程序】中 EURA Studio 快捷方式组中的【Uninstall EURA Studio】命令，将开始卸载过程。

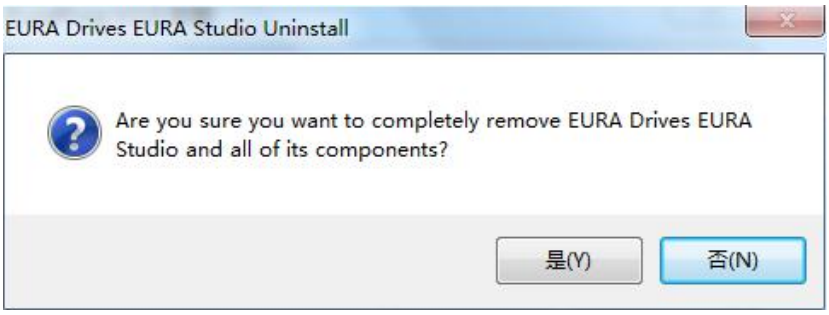


图 1.8 卸载确认

点击“是”进行卸载。

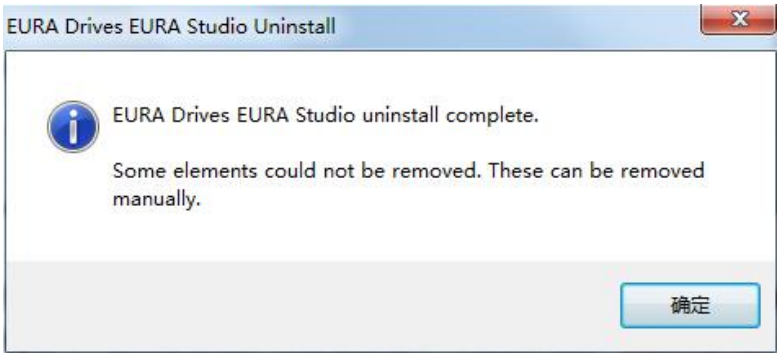


图 1.9 卸载完成

2) 单击【开始】→【控制面板】，进入控制面板，执行其中的【卸载程序】命令，继而在弹出的“卸载或更新程序”对话框中选择“EURA Drives EURA Studio x.x.x.x”（x.x.x.x 代表版本号），然后右键单击【卸载】按钮。

1.1.4 安装驱动程序

编程软件安装完成之后程需要单独安装 USB 调试接口的驱动程序。Win10 操作系统已经集成该驱动程序，无需再次安装。在 Win7/Win8 下需要独立安装驱动程序。

1) 解压缩安装目录 C:\Program Files (x86)\EURA Studio\USB-CDC Driver 下面的压缩文件。

图 1.10 USB 驱动程序动装界面 1

2) 将 PLC 上电，USB 下载线与 PC 端 USB 口连接，鼠标右键“计算机”打开设备管理器。

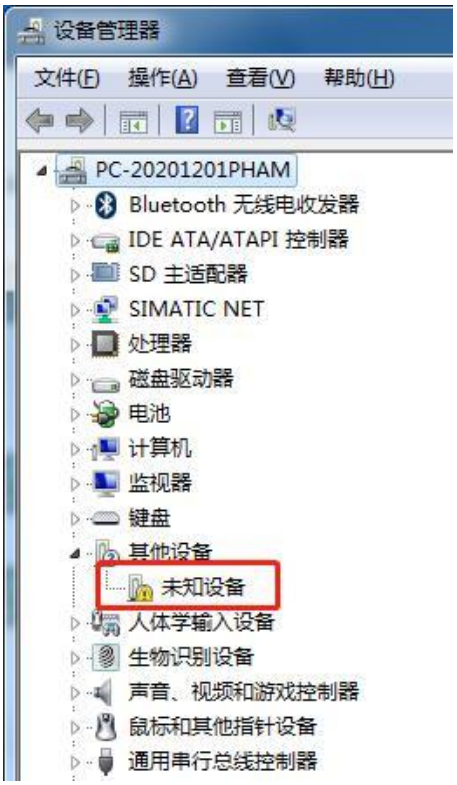


图 1.11 USB 驱动程序动装界面 2

3) 右键“未知设备”选择“更新驱动程序软件（P）”，弹出更新驱动程序对话框，选择“手动查找并安装驱动程序软件”。

图 1.12 USB 驱动程序动装界面 3

4) 选择“端口（COM 和 LPT）”，点击“下一步”。

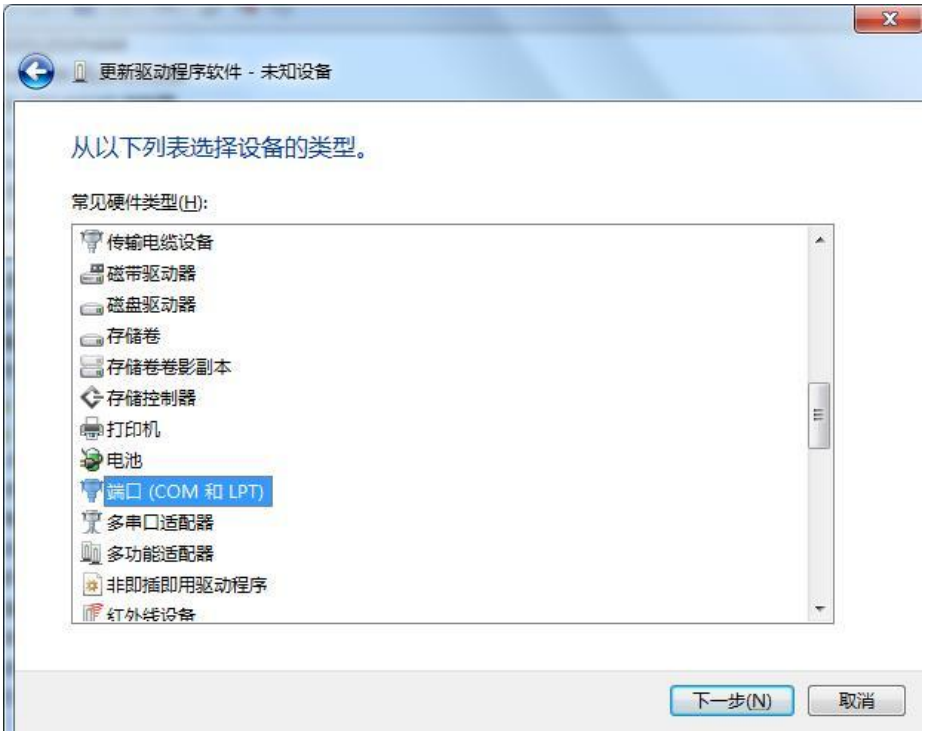


图 1.13 USB 驱动程序动装界面 4

5) 选择“从磁盘安装”。

图 1.14 USB 驱动程序动装界面 5

6) 点击“浏览”。

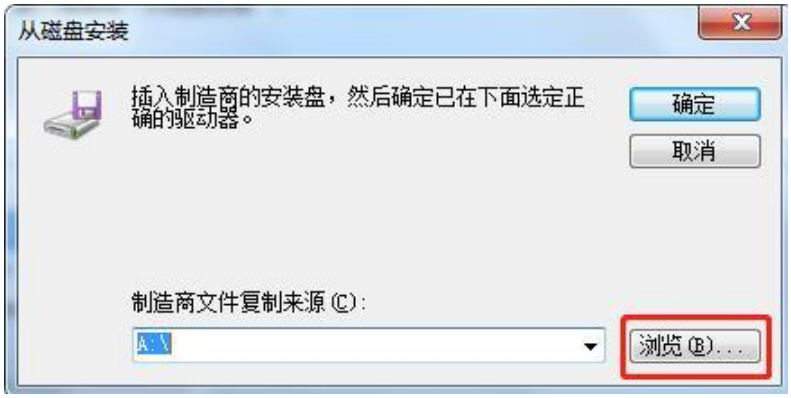


图 1.15　USB 驱动程序动装界面 6

7)　选择目录 C:\Program Files (x86)\EURA Studio\USB-CDC Driver\an-r11an0303eu0100-synergy-signed-usb-cdc-driver\SynergyUSBCDC 下面的 SynergyUSBCDC.inf 文件，点击“打开”。

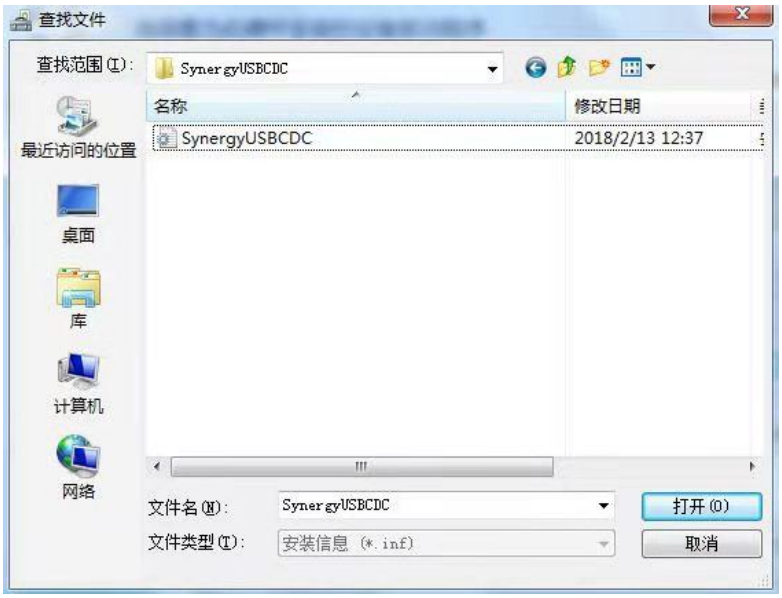


图 1.16　USB 驱动程序动装界面 7

8)　点击“确定”。

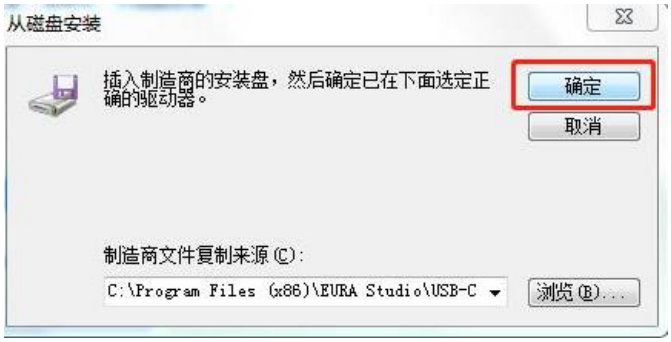


图 1.17　USB 驱动程序动装界面 8

9) 点击“下一步”。

图 1.18 USB 驱动程序动装界面 9

10) 在弹出的“更新驱动程序警告”对话框中，选择“是”。

图 1.19 USB 驱动程序动装界面 10

11) 在弹出的“Windows 安全”对话框中，勾选“始终信任”，点击“安装”。

图 1.20 USB 驱动程序动装界面 11

12) 等待驱动程序安装。

图 1.21 USB 驱动程序动装界面 12

13) 安装完成之后的界面如下图所示。

图 1.22 USB 驱动程序动装界面 13

14) 连接 PLC，将会在计算机设备管理器中新增一个串口，如下图所示，表示驱动安装成功。

图 1.23 USB 驱虚的虚拟串口

1.2 EURA Studio 软件界面

1.2.1 界面视图结构

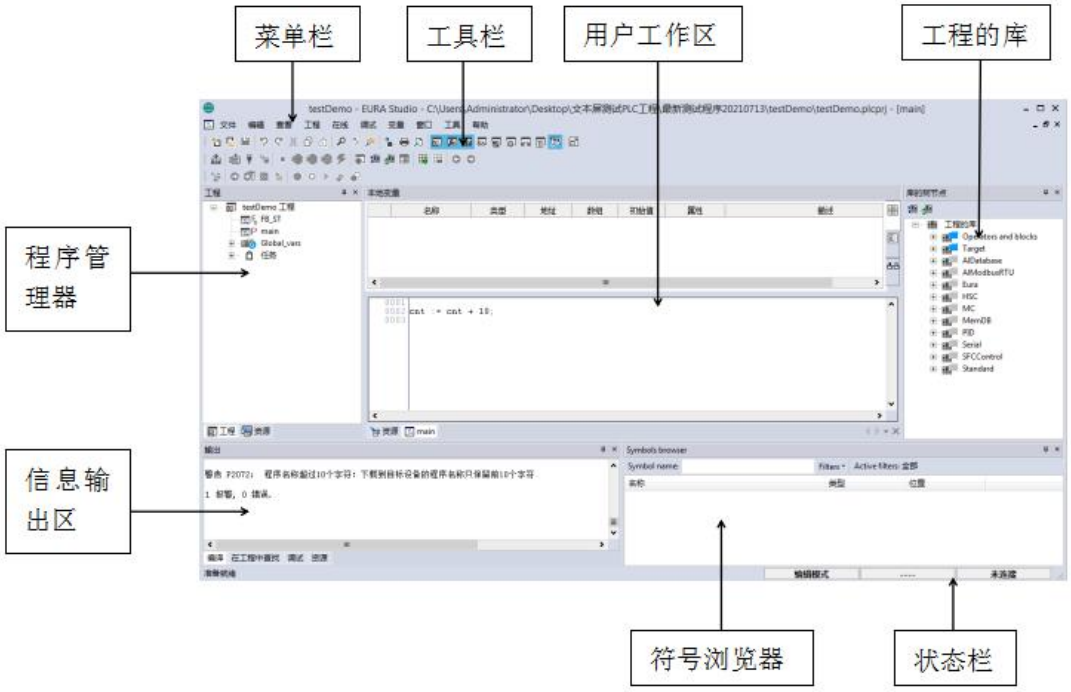


图 1.13 软件界面

- 菜　　单：菜单中包含了 EURA Studio 软件所有的操作命令。
- 工 具 栏：工具栏中包含了用户使用频度较高的一些操作命令。
- 状 态 栏：状态条提供了软件当前的状态信息和操作命令的提示信息。

- 工程管理器：工程管理器是界面中的主要窗口之一，以树状列表的形式直观地显示出了当前工程的所有组成部分，包括程序、配置、全局变量表等。用户可以在此窗口中对当前打开的工程进行管理、操作、维护。工程管理器的各个树节点大多支持右键，用右键单击某个节点将会弹出相应的右键菜单。
- 用户工作区：这是用户使用的主要区域，用户可以在此打开配置窗口、全局变量表、编辑器等窗口，完成配置、声明全局变量、编辑程序等任务。
- 符号浏览器：帮助查找工程中的全局变量、数据类型、操作符、功能块等符号。
- 信息输出区：用于显示 EURA Studio 软件的各种提示信息。其中“编译信息”窗口显示了用户最近一次的编译信息。

1.3 创建和编辑工程

1.3.1 创建工程

在菜单栏中依次点击“文件”—“新建工程”，弹出“新建工程”对话框。填写工程名，选择工程路径，选择目标设备，如 EC404-24DTD 1.2，然后点击“确认”。

图 1.14 新建工程界面

1.3.2 设备配置

在资源视图中可以配置工程中的 Modbus 数据配置、系统配置、在线诊断、扩展配置、工艺对象、RS232 和 RS485 串行通讯接口。

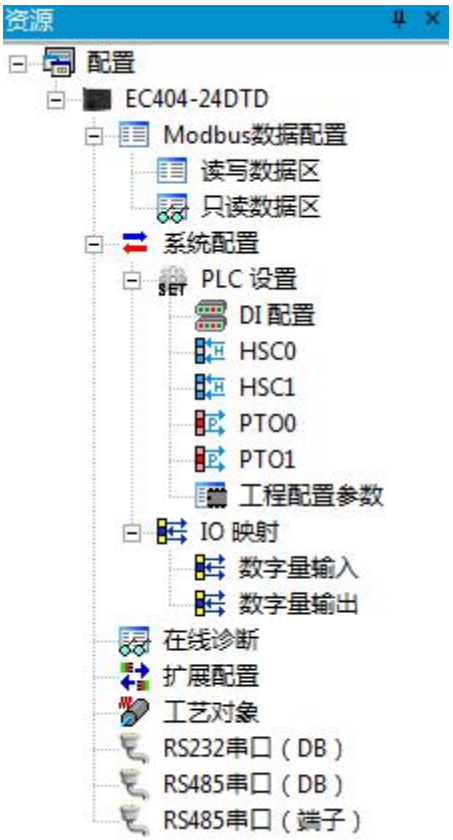


图 1.15 资源配置界面

在工程视图中，可以对任务进行配置。选中任意任务，右键选择任务配置（也可以双击“任务”节点），弹出任务配置对话框，可以设置任务的周期。

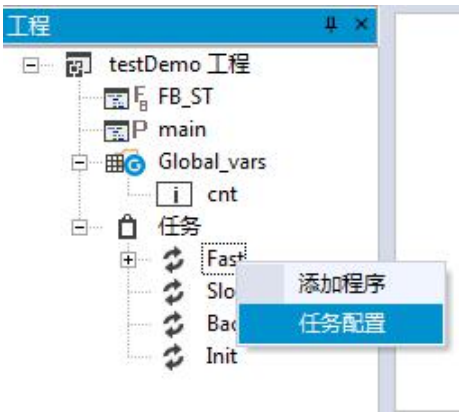


图 1.16 工程视图界面

图 1.17 任务配置界面

将程序分配给任务：选中任务，右键选择添加程序，弹出对象浏览器对话框，选择程序，点击确认。

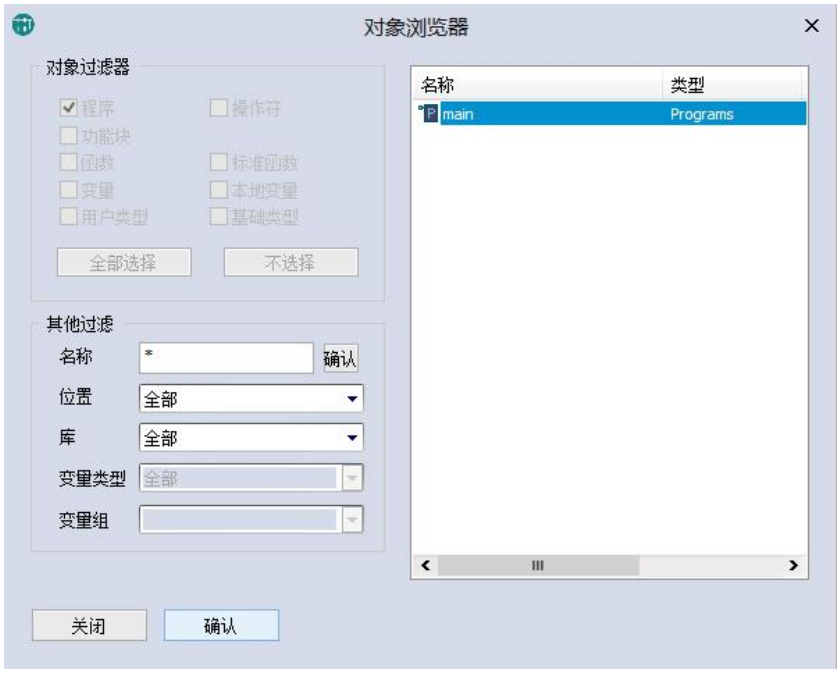


图 1.18 程序分配给任务界面

1.3.3 编辑项目

1. 添加全局变量

双击“Global_vars”打开全局变量列表，在右侧变量列表区鼠标右键，选择“插入”，或采用  工具，自动插入一个全局变量，更改变量名、变量类型等。

图 1.19 插入变量

2. 添加局部变量

双击设备树中的程序，打开程序编辑窗口，在本地变量区鼠标右键，选择“插入”，自动插入局部变量，更改变量名、变量类型等。点击本地变量窗口右侧的按钮可以切换变量的呈现形式，点击左侧按钮变量表现为 ST 语言格式，选择右侧按钮变量表现为表格格式。

图 1.20 局部变量表现形式切换

3. 编辑程序

用户编程可以使用五种语言：IL、LD、FBD、ST 和 SFC。

1）新建程序

选中工程名，右键，在弹出的菜单中选择添加——新建程序。

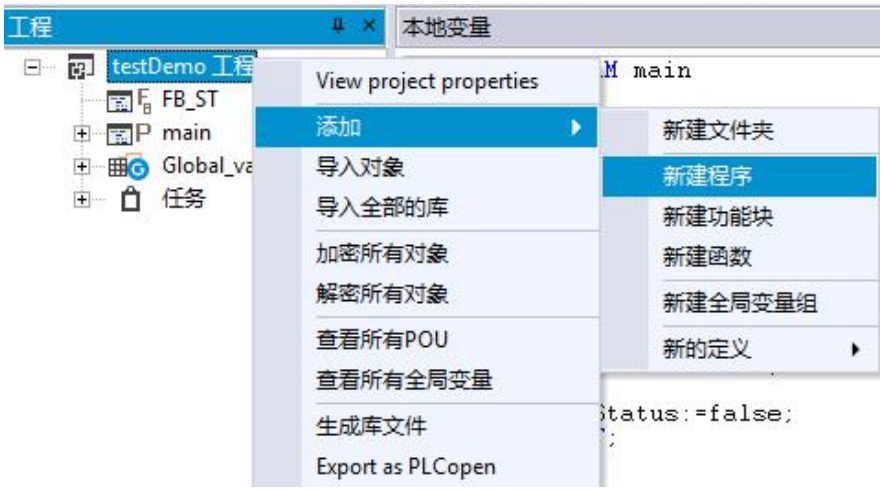


图 1.21 新建程序 1

在弹出的新建程序对话框中，选择编程语言，填写程序名，以及分配任务。

图 1.22 新建程序 2

2）程序编辑

不管什么 PLC 项目，编写程序总是必须的，以下介绍一个简单程序的输入和编译过程。

1. 双击工程树中的 main 打开主程序，如下图所示。在变量定义区编辑变量，在程序编辑区编辑程序。

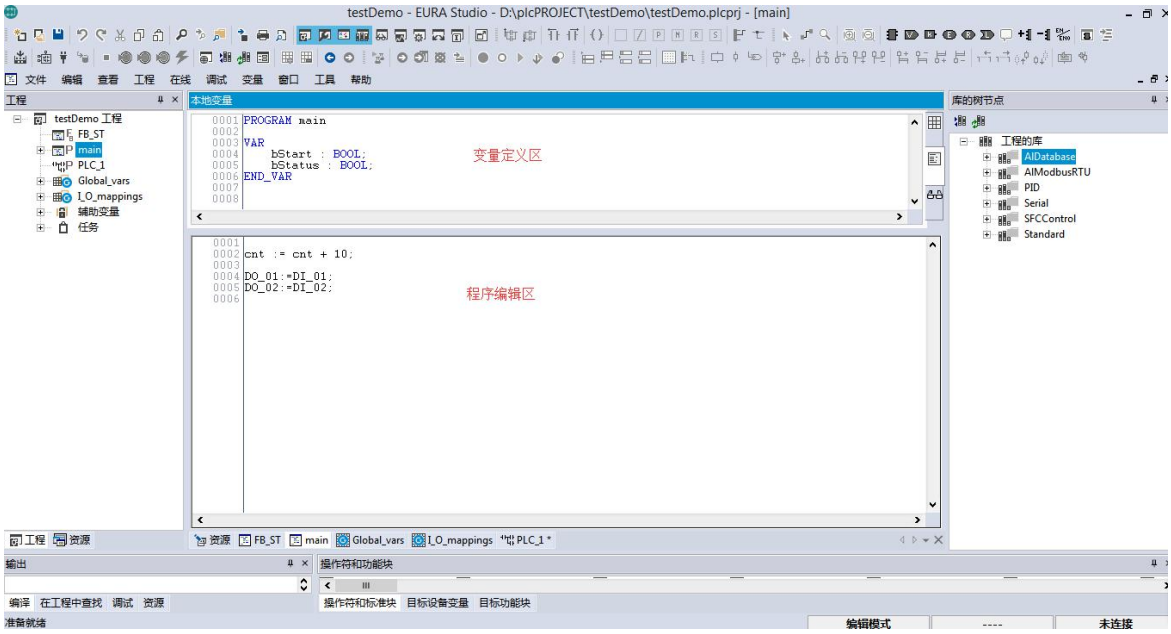


图 1.23 程序编辑视图

2. 保存项目。单击工具栏的“保存项目” 按钮，保存项目。

3）编译程序

在编写完工程后，必须对工程进行编译。在编译之前，软件会自动保存本工程，以确保编译的是用户最新的输入。但是，仍然建议用户注意随时保存自己的工作，以免发生意外。

用户可以使用如下任一方法来启动编译过程：

- 菜单栏选择 **工程→编译** 菜单命令；
- 单击工具栏上的  图标；
- 使用快捷键 **F7**。

编译的结果将会在下方信息输出窗口中的“编译信息”页面中显示。如果工程中有错误，用户需要根据错误信息提示对工程进行修改，直至编译成功。

1.4 工程资源配置

1.4.1 Modbus 数据配置

Modbus 数据配置分为读写数据区和只读数据区，均为全局变量。读写数据区可以被程序读写，而只读数据区只能被程序读取，不能被程序改写。读写数据区地址范围是 0x4000~0x4FFF，只读数据区地址范围是 0x6000~0x6FFF。

在配置树中，点击 Modbus 数据配置节点下的读写数据区或只读数据区，打开变量编辑窗口，点击“添加”按钮可插入一个变量，变量的地址是默认分配的，可以手动修改。添加变量后需要手动修改变量名称、变量类型和 PLC 类型等属性。

图 1.24 添加 modbus 通讯变量

选中变量所在的行，点击“删除”按钮，可以删除一个变量。按 Ctrl 或 Shift 选中多行，点击“删除”按钮，可以删除多个变量。

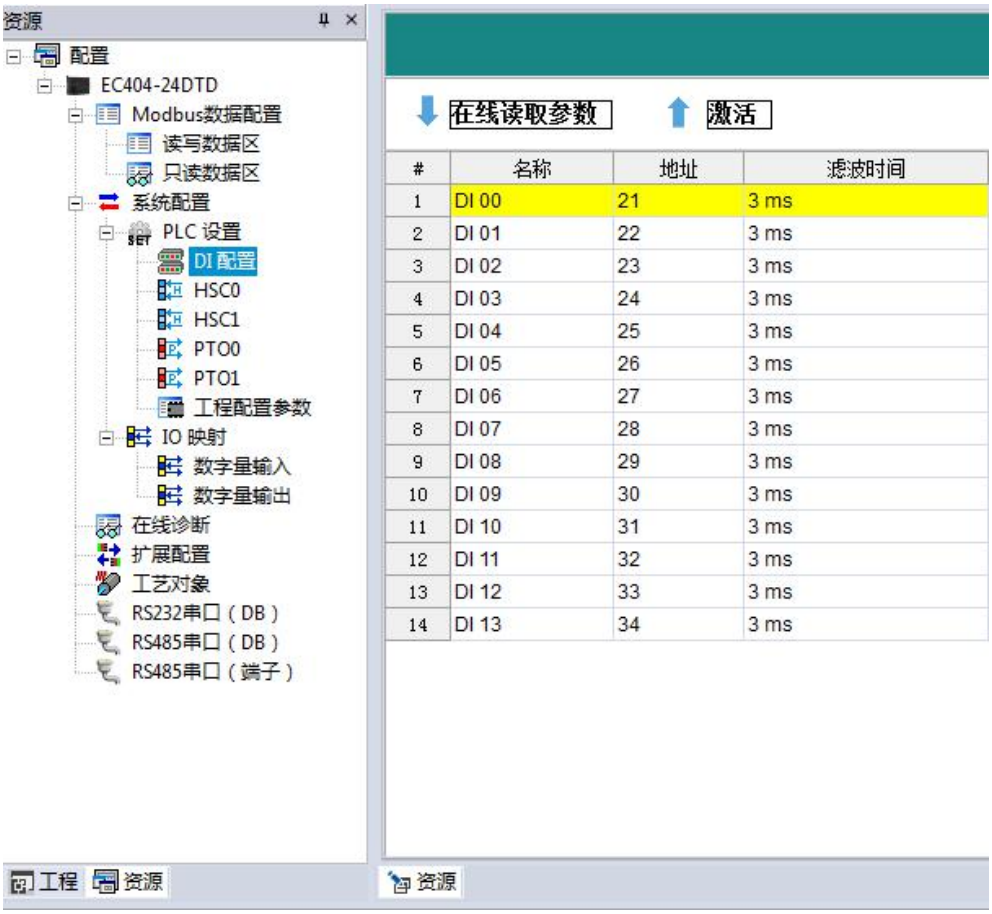
如果修改了某些变量的参数类型，变量的地址映射可能会出现错误，可以全部选中所有变量后，点击“重新计算”按钮，根据数据类型重新计算地址。

1.4.2 系统配置

1. PLC 设置

PLC 设置包含了数字量输入、高速输入和高速输出的功能，以及工程配置参数。

在 DI 界面可以设置数字量输入的滤波时间，配置范围 0~10ms。



#	名称	地址	滤波时间
1	DI 00	21	3 ms
2	DI 01	22	3 ms
3	DI 02	23	3 ms
4	DI 03	24	3 ms
5	DI 04	25	3 ms
6	DI 05	26	3 ms
7	DI 06	27	3 ms
8	DI 07	28	3 ms
9	DI 08	29	3 ms
10	DI 09	30	3 ms
11	DI 10	31	3 ms
12	DI 11	32	3 ms
13	DI 12	33	3 ms
14	DI 13	34	3 ms

图 1.25 DI 滤波配置界面

EC404-24DTD 型 PLC 提供了 2 路高速输入。HSC0 自动映射到 DI0 和 DI1，HSC1 自动映射到 DI2 和 DI3。在 HSC 配置界面配置高速输入的类型和信号类型等，在变量映射界面编辑变量名称，在程序中可直接调用该变量名，读取高速输入的数值。

图 1.26 DI 高速输入配置界面

使能高速输入 HSC0，HSC1 后，DI 映射界面如下图所示。

图 1.27 高速输入 DI 映射界面

EC404-24DTD 型 PLC 提供了 2 路高速输出。PTO0 自动映射到 DO0 和 DO1，PTO1 自动映

射到 DO2 和 DO3。在 PTO 配置界面配置高速输出的相关参数。

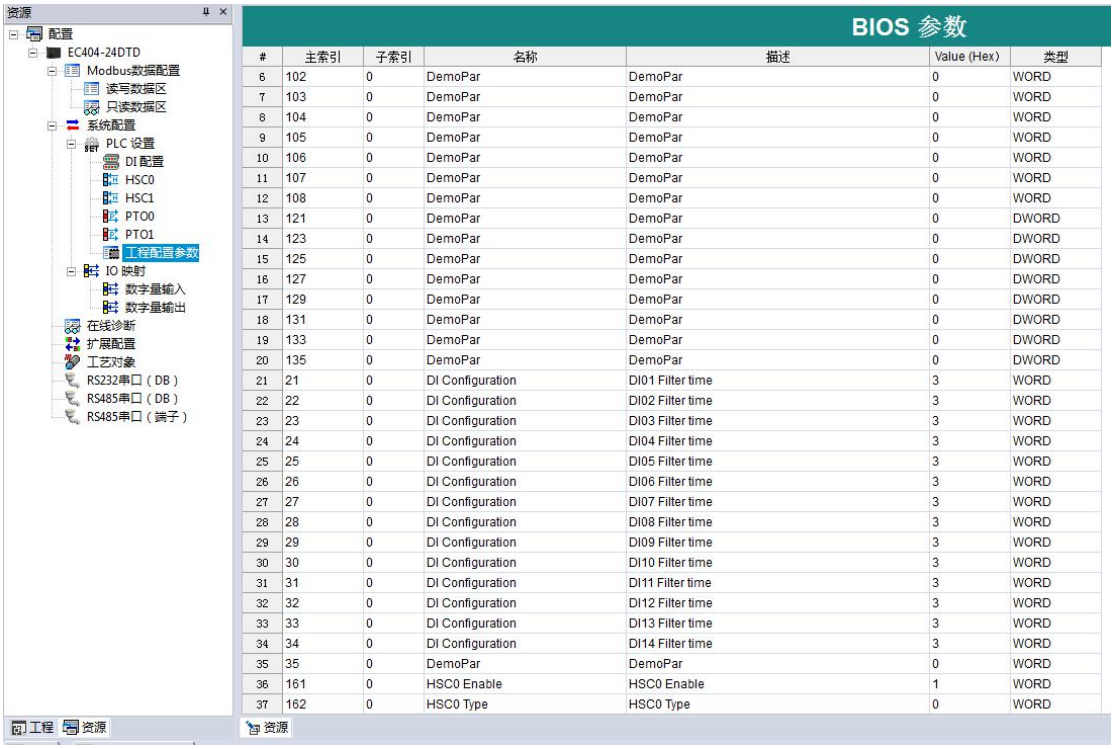
图 1.28 DO 高速输出配置界面

使能高速输出 PTO0，PTO1 后，DO 映射界面如下图所示，PTO 配置的模式不同，对应引脚的功能也不同。

#	名称	变量	类型	数据块	描述
1	DO 00		BOOL	%QX0.0	PTO0 Pulse/Direction A
2	DO 01		BOOL	%QX0.1	PTO0 Pulse/Direction B
3	DO 02		BOOL	%QX0.2	PTO1 Pulse/Direction A
4	DO 03		BOOL	%QX0.3	PTO1 Pulse/Direction B
5	DO 04		BOOL	%QX0.4	Digital output
6	DO 05		BOOL	%QX0.5	Digital output
7	DO 06		BOOL	%QX0.6	Digital output
8	DO 07		BOOL	%QX0.7	Digital output
9	DO 08		BOOL	%QX0.8	Digital output
10	DO 09		BOOL	%QX0.9	Digital output

图 1.29 高速输出 DO 映射界面

在工程配置参数界面，包含了 PLC 所有功能的相关配置，用户可以在此界面查看相关功能配置。



#	主索引	子索引	名称	描述	Value (Hex)	类型
6	102	0	DemoPar	DemoPar	0	WORD
7	103	0	DemoPar	DemoPar	0	WORD
8	104	0	DemoPar	DemoPar	0	WORD
9	105	0	DemoPar	DemoPar	0	WORD
10	106	0	DemoPar	DemoPar	0	WORD
11	107	0	DemoPar	DemoPar	0	WORD
12	108	0	DemoPar	DemoPar	0	WORD
13	121	0	DemoPar	DemoPar	0	DWORD
14	123	0	DemoPar	DemoPar	0	DWORD
15	125	0	DemoPar	DemoPar	0	DWORD
16	127	0	DemoPar	DemoPar	0	DWORD
17	129	0	DemoPar	DemoPar	0	DWORD
18	131	0	DemoPar	DemoPar	0	DWORD
19	133	0	DemoPar	DemoPar	0	DWORD
20	135	0	DemoPar	DemoPar	0	DWORD
21	21	0	DI Configuration	DI01 Filter time	3	WORD
22	22	0	DI Configuration	DI02 Filter time	3	WORD
23	23	0	DI Configuration	DI03 Filter time	3	WORD
24	24	0	DI Configuration	DI04 Filter time	3	WORD
25	25	0	DI Configuration	DI05 Filter time	3	WORD
26	26	0	DI Configuration	DI06 Filter time	3	WORD
27	27	0	DI Configuration	DI07 Filter time	3	WORD
28	28	0	DI Configuration	DI08 Filter time	3	WORD
29	29	0	DI Configuration	DI09 Filter time	3	WORD
30	30	0	DI Configuration	DI10 Filter time	3	WORD
31	31	0	DI Configuration	DI11 Filter time	3	WORD
32	32	0	DI Configuration	DI12 Filter time	3	WORD
33	33	0	DI Configuration	DI13 Filter time	3	WORD
34	34	0	DI Configuration	DI14 Filter time	3	WORD
35	35	0	DemoPar	DemoPar	0	WORD
36	161	0	HSC0 Enable	HSC0 Enable	1	WORD
37	162	0	HSC0 Type	HSC0 Type	0	WORD

图 1.30 BIOS 参数界面

2. IO 映射

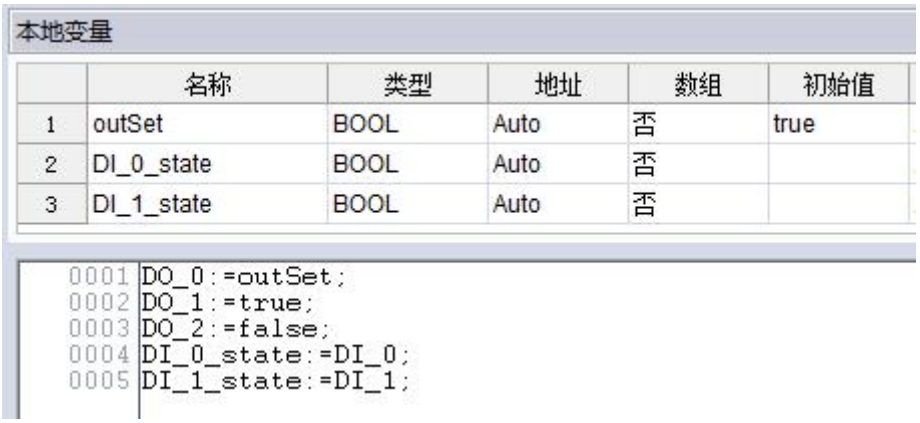
EC404-24DTD 型 PLC 提供了 14 个数字量输入和 10 个数字量输出通道。每个输入、输出通道自动映射了地址变量。在 DI DO 映射界面分别编辑变量名称，在程序中可直接调用该变量名，完成 IO 变量映射。当程序中没有配置 HSC 和 PTO 时，所有通道均作为普通输入输出口使用；当程序中配置了 HSC 或 PTO 时，相关通道作为高速输入输出通道使用，其余通道作为普通输入输出口使用。

#	名称	变量	类型	数据块	描述
1	DI 00	DI_0	BOOL	%IX0.0	Digital input
2	DI 01	DI_1	BOOL	%IX0.1	Digital input
3	DI 02	DI_2	BOOL	%IX0.2	Digital input
4	DI 03		BOOL	%IX0.3	Digital input
5	DI 04		BOOL	%IX0.4	Digital input
6	DI 05		BOOL	%IX0.5	Digital input
7	DI 06		BOOL	%IX0.6	Digital input
8	DI 07		BOOL	%IX0.7	Digital input
9	DI 08		BOOL	%IX0.8	Digital input
10	DI 09		BOOL	%IX0.9	Digital input
11	DI 10		BOOL	%IX0.10	Digital input
12	DI 11		BOOL	%IX0.11	Digital input
13	DI 12		BOOL	%IX0.12	Digital input
14	DI 13		BOOL	%IX0.13	Digital input

图 1.31 数字量输入映射

#	名称	变量	类型	数据块	描述
1	DO 00	DO_0	BOOL	%QX0.0	Digital output
2	DO 01	DO_1	BOOL	%QX0.1	Digital output
3	DO 02	DO_2	BOOL	%QX0.2	Digital output
4	DO 03		BOOL	%QX0.3	Digital output
5	DO 04		BOOL	%QX0.4	Digital output
6	DO 05		BOOL	%QX0.5	Digital output
7	DO 06		BOOL	%QX0.6	Digital output
8	DO 07		BOOL	%QX0.7	Digital output
9	DO 08		BOOL	%QX0.8	Digital output
10	DO 09		BOOL	%QX0.9	Digital output

图 1.32 数字量输出映射



	名称	类型	地址	数组	初始值
1	outSet	BOOL	Auto	否	true
2	DI_0_state	BOOL	Auto	否	
3	DI_1_state	BOOL	Auto	否	

```
0001 DO_0:=outSet;
0002 DO_1:=true;
0003 DO_2:=false;
0004 DI_0_state:=DI_0;
0005 DI_1_state:=DI_1;
```

图 1.33 程序调用变量

1.4.3 在线诊断

在线诊断功能用于用户在线调试时使用，用户连接上 PLC 后，可以读取和修改相关参数的配置。

主索引	子索引	名称	描述	数值（十进制）	数值（十六进制）	调试（十进制）	参数类型	操作类型	操作	参考
80	0	Extbus Enable	Extbus Enable	0	0		WORD	RW	Update	Ref.
81	0	Extbus Delay	Extbus Scan Delay	0	0		WORD	RW	Update	Ref.
82	0	Extbus Status	Extbus Status	0	0		WORD	RO	Read	Ref.
83	0	Extbus Act	Extbus Actual Cycle	0	0		WORD	RO	Read	Ref.
84	0	Extbus Retry	Extbus Retry	0	0		WORD	RW	Update	Ref.
100	0	Module Num	Extbus Module Number	0	0		WORD	RW	Update	Ref.
101	0	Module Type	Fist Extbus Module Type	0	0		WORD	RW	Update	Ref.
102	0	Module Type	Second Extbus Module Type	0	0		WORD	RW	Update	Ref.
103	0	Module Type	Third Extbus Module Type	0	0		WORD	RW	Update	Ref.
104	0	Module Type	Fourth Extbus Module Type	0	0		WORD	RW	Update	Ref.
105	0	Module Type	Fifth Extbus Module Type	0	0		WORD	RW	Update	Ref.
106	0	Module Type	Sixth Extbus Module Type	0	0		WORD	RW	Update	Ref.
107	0	Module Type	Seventh Extbus Module Type	0	0		WORD	RW	Update	Ref.
108	0	Module Type	Eighth Extbus Module Type	0	0		WORD	RW	Update	Ref.
121	0	Module Configuration	Fist Extbus Module Configration	0	0		DWORD	RW	Update	Ref.
123	0	Module Configuration	Second Extbus Module Configration	0	0		DWORD	RW	Update	Ref.
125	0	Module Configuration	Third Extbus Module Configration	0	0		DWORD	RW	Update	Ref.
127	0	Module Configuration	Fourth Extbus Module Configration	0	0		DWORD	RW	Update	Ref.
129	0	Module Configuration	Fifth Extbus Module Configration	0	0		DWORD	RW	Update	Ref.

图 1.34 在线诊断界面

1.4.4 扩展配置

ExtBus 基于高速 RS485 通信协议实现，传输速率为 1.5Mbps，用于 PLC 和远程 I/O 的 IO 拓展，具有简单易用、实施成本低、传输速率高等特点。在扩展模块配置界面可以设置是否开启 ExtBus 功能以及扫描延时时间和重试次数。

图 1.35 扩展模块配置界面

当“扩展配置”功能被使能后，选中“扩展配置”右击选中“添加”，弹出“设备目录”列表，在列表中可以选中工程中所配置的扩展模块类型，在确认所配置模块类型后，“扩展配置”节点下会自动添加所确认扩展模块类型。

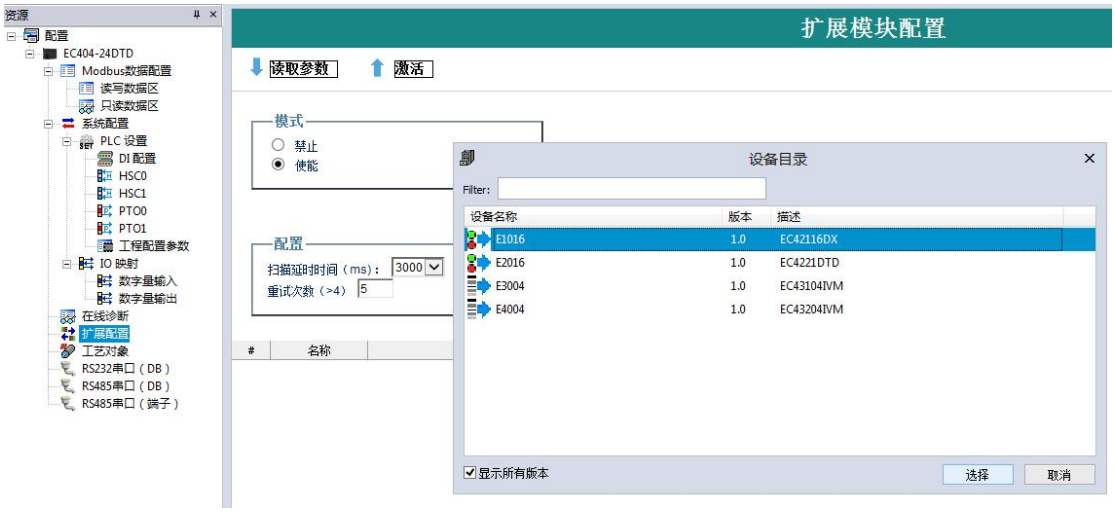


图 1.36 添加扩展模块

1.4.5 工艺对象

使能“工艺对象”功能后，选中“工艺对象”后右键鼠标，可以添加伺服轴。然后点击伺服轴，在右侧轴对象节点界面配置轴驱动和初始化参数，在配置轴的 PTO 配置时，必须

事先将 PTO 功能使能，方可在 PTO 配置选项中选择所需 PTO 通道。

图 1.37 轴对象节点配置界面

图 1.38 初始化参数

1.4.6 串行通讯接口

EC404-24DTD 型 PLC 提供了一个 RS232 和两个 RS485 通讯接口，这些串口默认采用 Modbus RTU 协议，既可以作为主站使用，也可以作为从站使用。在 Modbus 配置界面可以配置串口模式、波特率、以及从站地址等。但是工程设计中不允许设置 2 个 RS485 通讯主站。

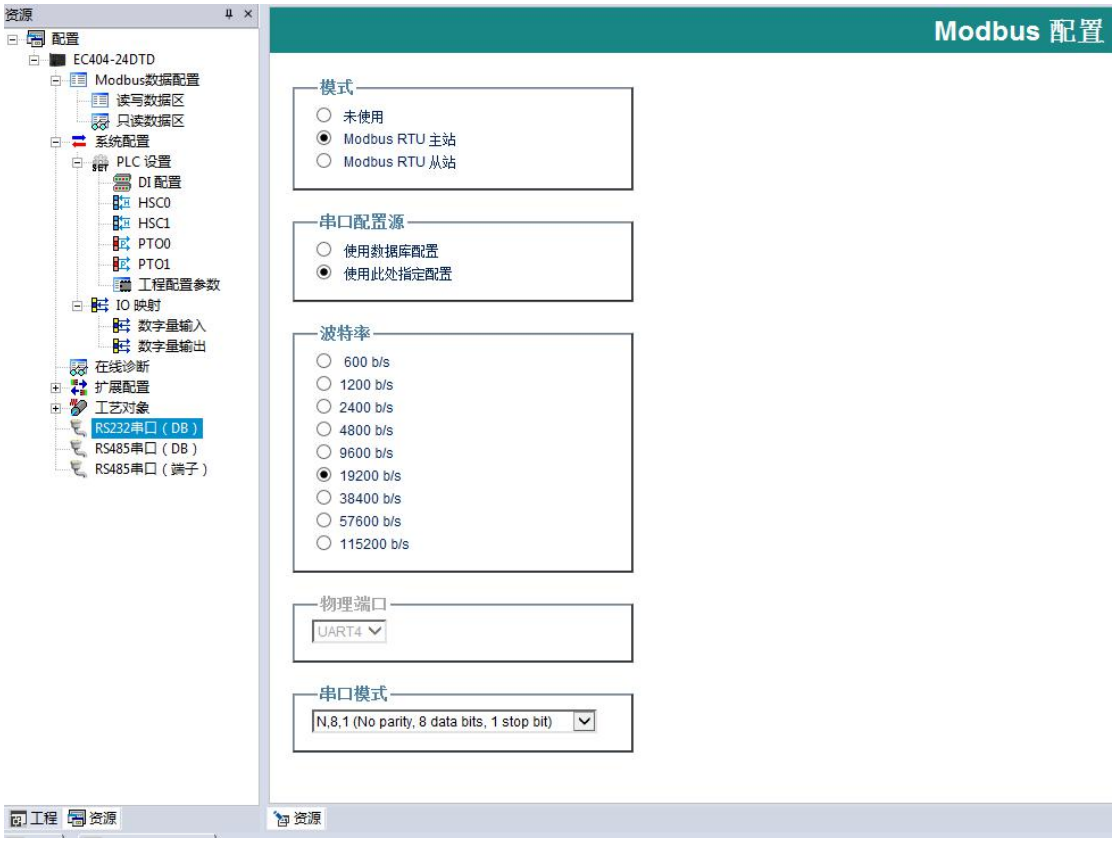


图 1.39 modbus 配置界面

1.5 程序下载

用户把硬件配置和程序编写完成后，即可将硬件配置和程序下载到 CPU 中。下载步骤如下。

1. 连接设备

用 USB 线缆将 PLC 与计算机连接起来，然后点击菜单栏“在线”——“建立通信”，如图所示。

图 1.40 建立通信

在弹出的设备连接管理器配置对话框中，选择 GDB，然后点击属性，如图所示。

图 1.41 设备连接管理器

GDB 设置窗口如下图所示，在 Port 下选择与 USB 对应的虚拟串口，然后点击“确认”，保存当前设置。

图 1.42 GDB 设置

点击连击按钮  ，或点击菜单栏“在线”——“连接”，建立与目标 PLC 之间的通信，如下图所示。

图 1.43 通信连接菜单

通信连接成功之后，在编示软件界面的右下角提示已经成功建立通信连接，如下图所示。

图 1.44 成功连立通信连接

2. 下载

在与 PLC 建立通信之后，点击 或单击菜单栏“在线”——“下载代码”，执行 PLC 程序下载。当程序中存在高速总线设计时，下载完毕后，将需要用户进行重启/Reboot 设备。

图 1.45 下载代码

点击  或者单击菜单栏“调试”——“ 在线调试模式”，进入在线调试模式，可以实时监控程序运行状态，在代码旁边动态显示数值的实时变化。

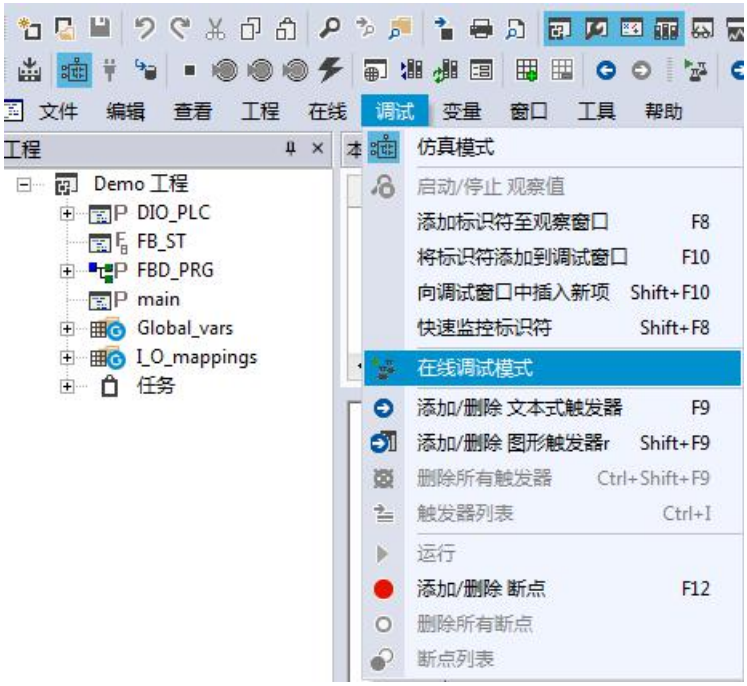


图 1.46 开启在线调试模式

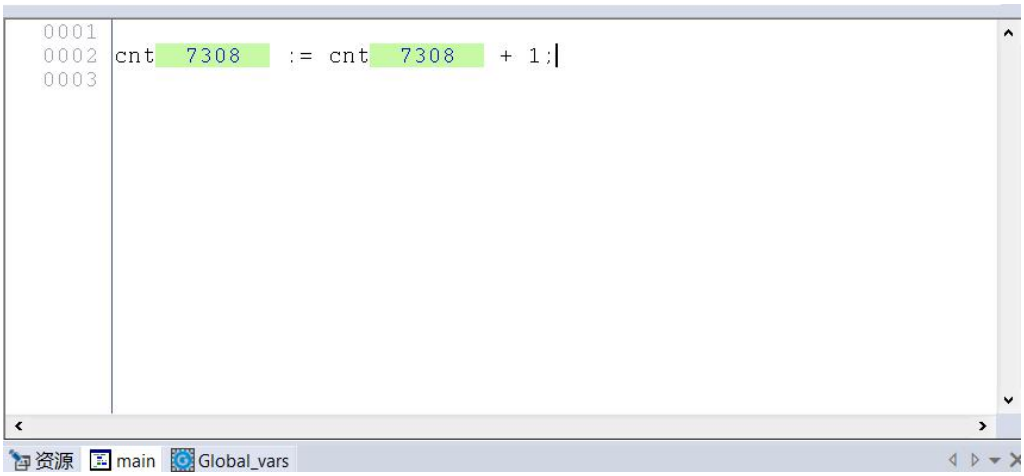


图 1.47 在线监控程序执行效果

1.6 程序调试

1.6.1 程序启动

EURA Studio 程序启动有如下三种方式，在在线菜单中进行选择，详见下图所示。

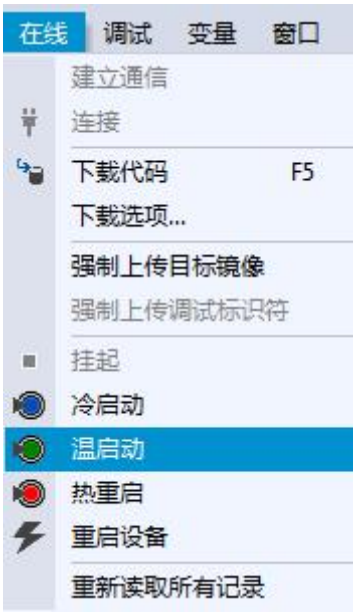


图 1.48 程序启动

1. 冷启动

冷启动命令不但将普通变量的值设置为当前活动应用程序的初始值，而且将保持型变量（PERSISTENT 和 RETAIN 变量）的值也设置为初始值 0。冷启动发生在程序下载到 PLC 之后，运行之前（冷启动），执行冷启动命令之前，为了安全考虑，系统会弹出提示框，如下图所示。

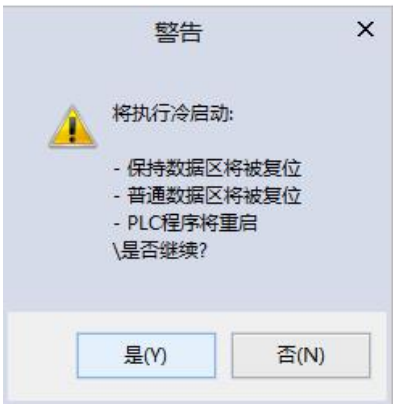


图 1.49 冷启动

2. 温启动

与“冷启动”不同的是，温启动后，除了保持型变量（PERSISTENT 和 RETAIN 变量）外，其它当前的应用的变量都被重新初始化。如果设置了初始值的变量，温启动后这些变量值还原为设定的初始值，否则变量都会被置为标准初始值 0。为了安全起见，当所有变量在初始化之前，系统也会给出提示，提示用户是否确认温启动操作，如下图所示。此情形只有在程序正在运行时断电，或者通过控制开关闭控制器，然后再打开时出现。

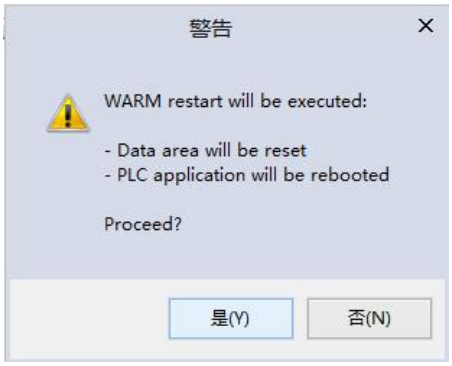


图 1.50 温启动

3. 热重启

执行热重启后，PLC 程序将重启。为了安全起见，在 PLC 重启之前，系统也会给出提示，提示用户是否确认热重启操作，如下图所示。

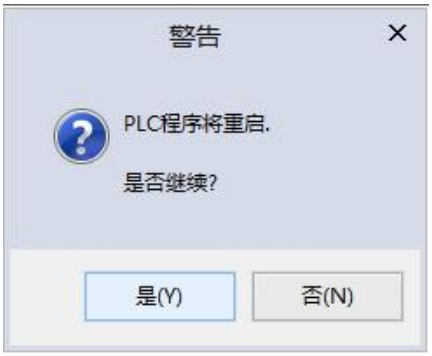


图 1.51 冷启动

1.6.2 程序调试

在 EURA Studio 中“调试”菜单的视图如下图所示。

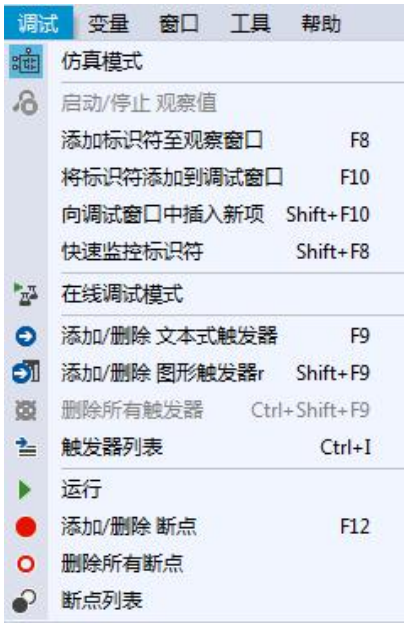


图 1.52 程序调试待菜单

1. 断点

断点是程序内处理停止的功能，当程序停止后，程序研发人员可以借此观察程序到断点位置时其变量及 I/O 等相关变量的内容，有助于深入了解程序运行的机制，发现及排除程序故障。

在 EURA Studio 中所有的编程语言都可以设置断点。在文本编辑器 ST 语言中中，断点设置在行上；在 FBD 和 LD 编辑器中设置在网络号上；而在 SFC 中，设置在步上。

1) IL 中断点位置的几行代码在内部被组合成一个单行的 C 代码，断点不能在每个行都进行设置。断点位置可以包括在一个程序内的所有变量值可以改变的位置，或者在程序流程分支闭合处。

2) 结构化文本 ST 允许以下位置设置断点：

- 在每个赋值处
- 在每个 RETURN 和 EXIT 指令处
- 正在评估的条件处（WHILE，IF，REPEAT）
- 在 POU 结束处。

(1) 设置断点

为了设置一个断点，点击需要设置断点的行，点击菜单“调试” -->“添加/删除 断点”完成，也可以使用功能键 [F12]，或使用工具条中的“ ”符号完成。则该行的颜色就会变成深绿色。

(2) 删除断点

相应地，要删除一个断点，可以点击需要删除断点的行，通过菜单“调试”-->“添加/删除 断点”完成，也可以使用功能键[F12]或使用工具条中的“ ”符号完成。

如果想要删除所有断点， 可以点击需要删除断点的行，通过菜单“调试”-->“删除所有断点”完成，也可以使用工具条中的“ ”符号完成。

(3) 在断点处发生了什么？

如果 PLC 程序到达一个断点，则在屏幕上显示相应行的断点，PLC 会处于停止模式。

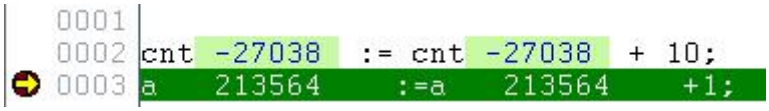


图 1.53 程序停在断点处

2. 单步执行

设置断点后，可以进行单步执行程序，该功能可以让程序一步一步的运行，方便编程人员进行调试，以便检查程序中的逻辑错误。当程序处于断点处，点击工具栏中的“  ”符号，程序会跳到下一行去执行。

3. 在线调试

在“在线调试”模式下，点击本地变量右侧的“  ”符号，可以实时观察变量的状态。

图 1.54 变量监控窗口

选择菜单栏中的“工具”--“工具窗口”--“示波器”，调出示波器窗口。

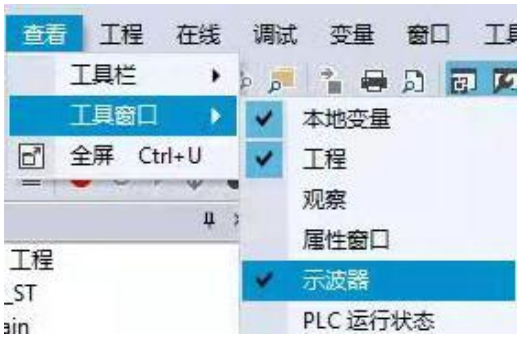


图 1.55 调用示波器

将工程树下的变量，拖拽至示波器窗口中，可以实时采集变量的数值。

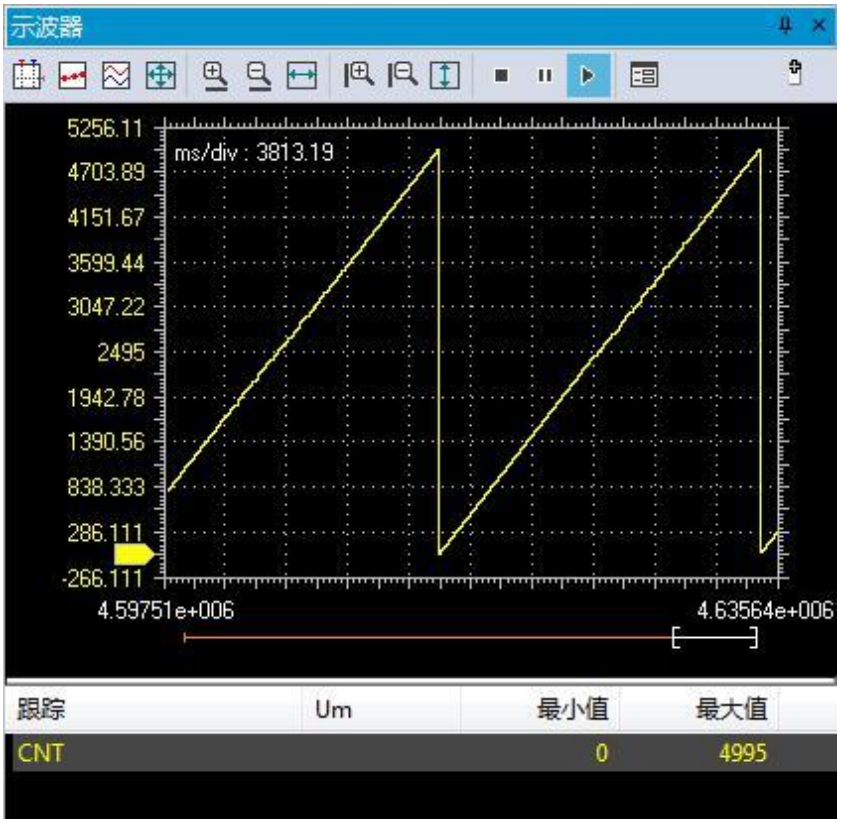


图 1.56 示波器窗口

1.7 仿真

用户在编程调试时，如果没有 EC400 PLC，可以使用 EURA Studio 的仿真功能来调试用户程序的逻辑。开启仿真功能的方法下图所示。

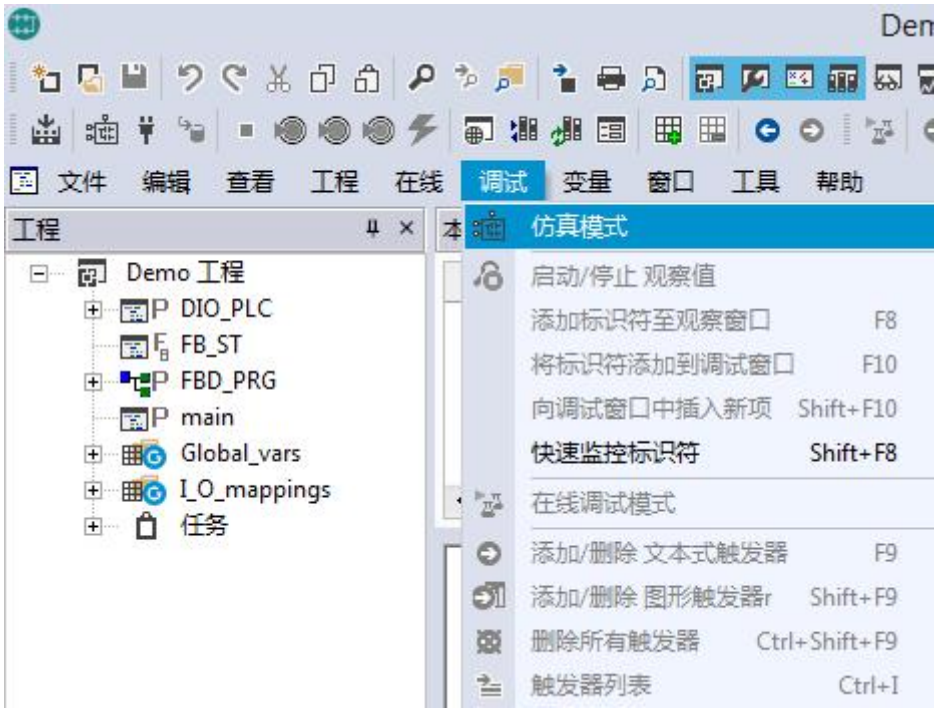


图 1.57 程序仿真开启

在弹出的“选择工作空间”对话框中填写名称和设置工作空间的路径，点击“确认”。

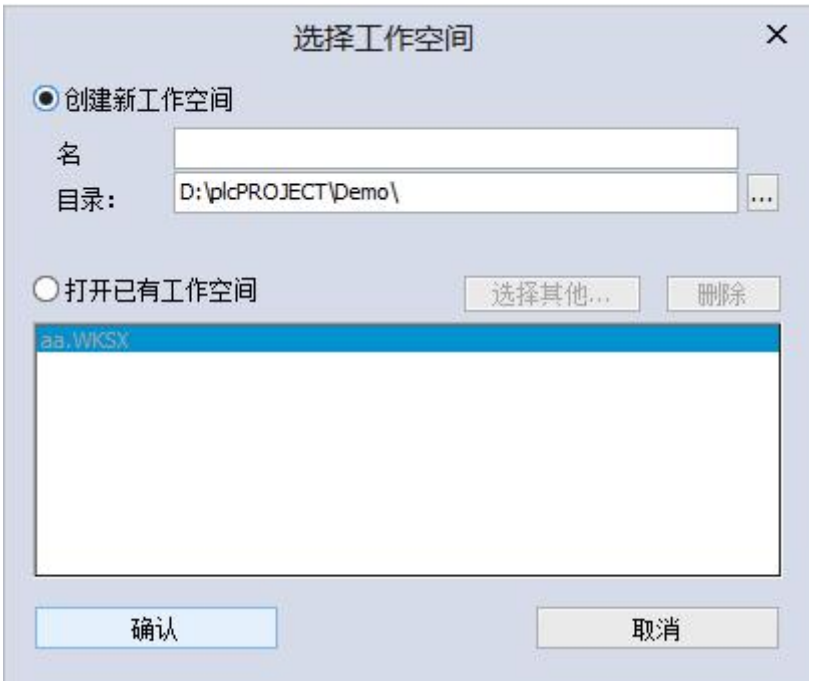


图 1.58 创建仿真工作空间

在仿真状态，编程软件下方状态栏有“已经连接”提示。

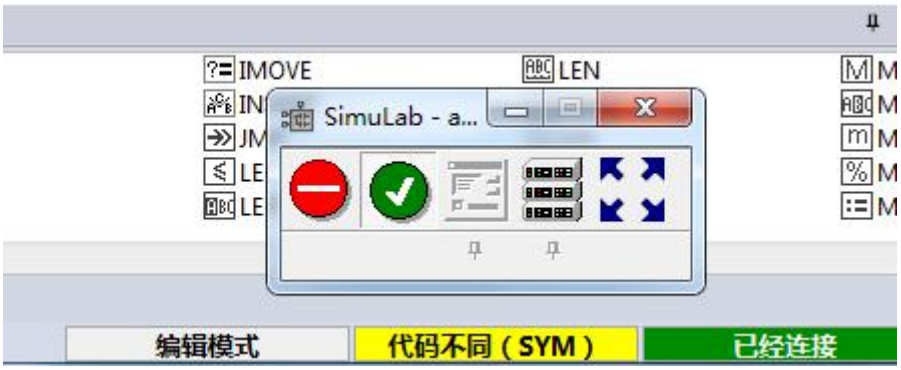


图 1.59 开启仿真后的状态栏

在仿真状态，可以编译用户程序，点击  下载代码，这里是将用户程序加载到 PC 的仿真器中。同样可以点击  进入“在线调试模式”运行模式，用户可以实时监控程序运行状态，进行强制修改参数的操作。

在仿真状态，单击菜单栏“调试”——“仿真模式”，退出仿真模式。

注：仿真模式下，仅能验证程序的逻辑功能，无法验证通信和运动控制功能。

1.8 项目保存

项目保存的目的是把整个项目的文档压缩到一个压缩文件中，以方便备份和转移。项目保存的步骤如下。

单击菜单栏的“文件”——“工程另存为”，弹出选择保存的路径和名称选择界面。单击“确认”按钮，生成一个后缀为“.plcprj”的工程文件。

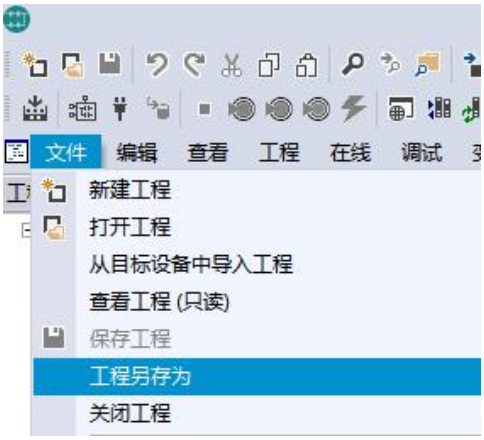


图 1.60 项目保存

图 1.61 选择保存的路径

1.9 创建一个完整的项目

1.9.1 新建工程

启动 EURA Studio，在菜单栏中选择“文件”——“新建工程”。在弹出的对话框中输入工程名称和存储路径，选择相应目标设备如 EC404-24DTD 1.2，点击确定。

图 1.62 新建工程

1.9.2 用户程序的编写

创建 IO 程序：在左侧工程树选项卡 Demo 工程节点右键，选择“添加”----“新建程序”。

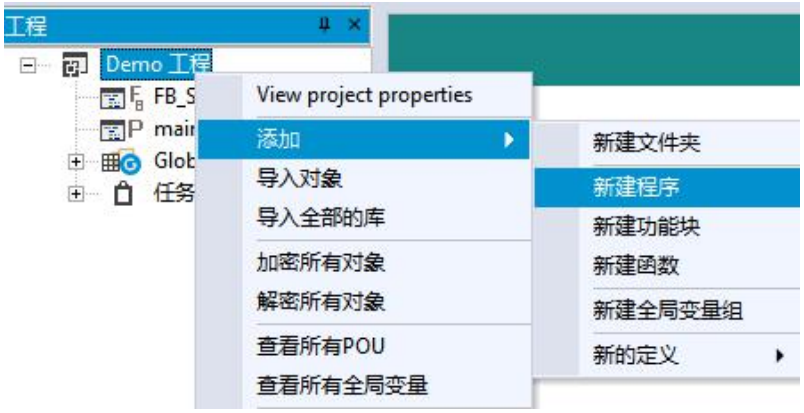


图 1.63 添加程序

填写程序名称，语言选择结构化文本(ST)，配置任务，点击“确认”，完成程序创建。

图 1.64 程序命名和任务配置

在打开的程序编辑器中完成 IO 部分流水灯程序的变量定义和程序实现。

本地变量

```
PROGRAM DIO_PLC

VAR
    bStart : BOOL := false;
    uiCount : UINT := 0;
    outValue: BOOL := false;
END_VAR
```

```
IF bStart THEN

    IF (uiCount MOD 4)=0 THEN

        IF outValue = TRUE THEN
            outValue := FALSE;
        ELSE
            outValue := TRUE;
        END_IF;

    END_IF;

    DO_01:=outValue;

    uiCount := uiCount+1;
    IF uiCount=3600 THEN
        uiCount:=0;
    END_IF;
END_IF;
```

图 1.65 程序编辑

1.9.3 配置用户程序的运行周期

配置任务：双击设备树选项卡“任务”节点，打开任务配置页面，或者选择任务，鼠标右键，选择“任务配置”来打开任务配置页面。设置 Slow 的设置周期为 Yes，周期设置为 20ms，点击确认。Fast 任务使用固定的 4ms 周期，不能被用户修改，用户仅可以修改 Slow 任务的周期值。

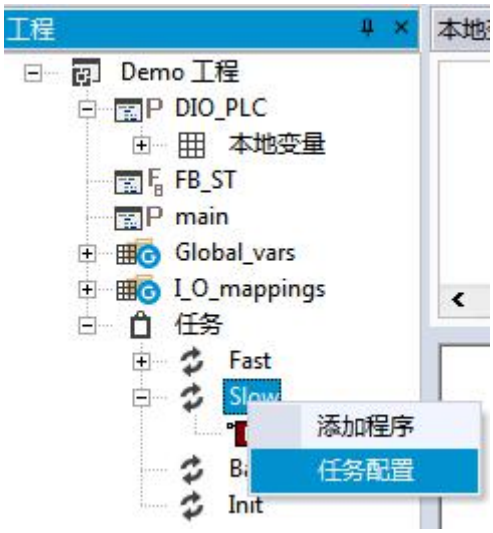


图 1.66 打开任务配置界面

图 1.67 配置任务

1.9.4 用户程序的变量与端口的关联配置

创建 IO 设备映射：在资源配置视图中，点击 IO 映射节点下的数字量输出节点，将程序中的变量 DO_01 与第一个 DO 输出点关联起来。

图 1.68 变量映射

1.9.5 用户程序的编译、下载和在线监控

程序编译：点击菜单栏“工程”菜单下的“编译”选项编译程序，编译信息会出现在消息区域中，如果有错误可以双击错误信息进行定位。

图 1.69 程序编译

程序建立通讯：点击菜单栏“在线”——“建立通信”，如下图所示。

图 1.70 建立通信

在弹出的设备连接管理器配置对话框中，选择 GDB，然后点击属性，如图所示。

图 1.71 设备连接管理器

在 Port 下选择与 USB 对应的虚拟串口，然后点击“确认”，保存当前设置。

图 1.72 GDB 设置

点击连击按钮，或点击菜单栏“在线”——“连接”，建立与目标 PLC 之间的通信。

图 1.73 通信连接菜单

通信连接成功之后，在编示软件界面的右下角提示已经成功建立通信连接，如下图所示。

图 1.74 成功连立通信连接

程序下载：在与 PLC 建立通信之后，点击或单击菜单栏“在线”——“下载代码”，执行 PLC 程序下载。

图 1.75 下载代码

在线监控：点击  ，进入“在线调试模式”运行模式，可以实时监控程序的运行状态，在代码旁边动态显示数值的实时变化。

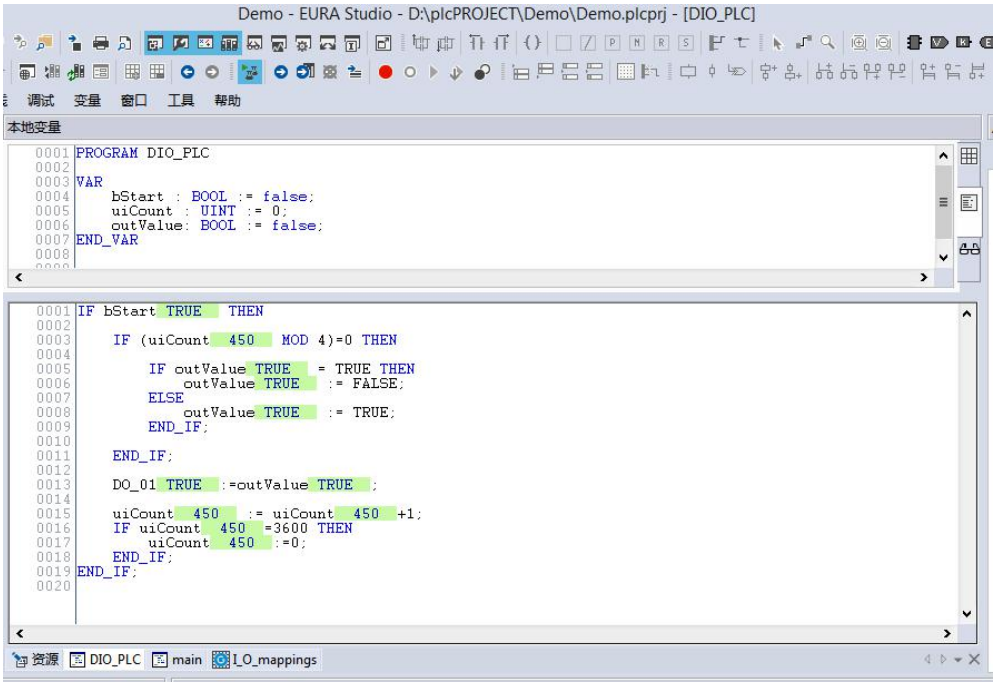


图 1.76 在线监控

在线调试：双击程序中的变量，弹出“写入值”对话框，可以在线修改变量的值。

图 1.77 程序调试

第二章 EC400PLC 编程基础知识

2.1 数制

PLC 是一种特殊的工业控制计算机，学习计算机必须掌握数制，对于 PLC 更是如此。

(1) 二进制

二进制数的 1 位（bit）只能取 0 和 1 两个不同的值，可以用来表示开关量的两种不同的状态，例如触点的断开和接通、线圈的通电和断电、灯的亮和灭等。在梯形图中，如果该位是 1 可以表示常开触点的闭合和线圈的得电，反之，该位是 0 可以表示常闭触点断开和线圈的断电。二进制表示方法是在数值前加前缀 2#，例如 2#1001 1101 1001 1101 就是 16 位二进制数值。十进制的运算规则是逢 10 进 1，二进制的运算规则是逢 2 进 1。

(2) 十六进制

十六进制的十六个数字是 0~9 和 A~F（对应于十进制中的 10~15，不区分大小写），每个十六进制数字可用 4 为二进制表示，例如 16#A 用二进制表示为 2#1010。16 进制的运算规则是逢 16 进 1。掌握二进制和十六进制之间的转换，对于学习欧瑞 PLC 来说十分重要。

(3) BCD 码

BCD 码用 4 位二进制数（或者 1 位十六进制数）表示一位十进制数，例如一位十进制数 9 的 BCD 码是 1001。4 位二进制有 16 种组合，但 BCD 码只用到前 10 个，而后六个（1010~1111）没有在 BCD 码中使用。

BCD 码的最高 4 位二进制数用来表示符号，16 位 BCD 码字的范围是-999~+999。32 位 BCD 码双字的范围是-9999999~+99999999.不同数制的数的表示方法见下表。

表 2-1 不同进制的数的表示方法

十进制	十六进制	二进制	BCD 码	十进制	十六进制	二进制	BCD 码
0	0	0000	00000000	8	8	1000	00001000
1	1	0001	00000001	9	9	1001	00001001
2	2	0010	00000010	10	A	1010	00010000
3	3	0011	00000011	11	B	1011	00010001
4	4	0100	00000100	12	C	1100	00010010
5	5	0101	00000101	13	D	1101	00010011
6	6	0110	00000110	14	E	1110	00010100
7	7	0111	00000111	15	F	1111	00010101

2.2 变量的表示和声明

变量可以用来表示一个数值，一个字符串值或一个数组等。

2.2.1 变量

变量是保存在存储器中待处理的抽象数据，是为了识别 PLC 的输入/输出、PLC 内部的存储区域而使用的名称，可以代替物理地址在程序中的使用。

用户可以根据需要随时改变变量中所存储的数据值。在程序执行过程中，变量的值可以发生变化。使用变量之前必须先声明变量，及指定变量的类型和名称。变量具有名称，类型和值。变量的数据类型确定它所代表的内存大小和类型。变量名即指在程序源代码中的标识符。

2.2.2 标识符

标识符就是变量的名称。在定义标识符时，根据 IEC 61131-3 标准，必须由字母、数字和下划 线字符组成。此外，还应遵循如下规则：

- 标识符的首字母必须是字母或下划线，最后一个字符必须是字母或数字，中间允许字母、数字、下划线。
- 标识符中不区分字母的大小写。
- 下划线是标识符的一部分，但标识符中不允许有两个或两个以上连续的下划线。
- 不得含有空格。
- 例如 ab_c、AB_de 和_AbC 是允许的标识符，而 1abc、__abc 和 a__bc 均不允许。

2.2.3 变量声明

变量声明就是指定变量的名称、类型和赋初始值，变量的声明非常重要，未经声明的变量是不能通过编译的，所以也无法在程序中使用。用户可以在程序组织单元（POU）、全局变量列表 （GVL）和自动声明对话框中进行变量的声明。在 EURA Studio 中变量声明分为两类，普通变量声明和直接变量。

1) 普通变量声明

最常用的变量声明，不需要和硬件外设或通讯进行关联的变量，仅供项目内部逻辑使用。普通变量声明须符合以下规则：

< 标识符>:<数据类型> {:=<初值>};

{}中为可选部分。

如 nTest:BOOL; nTest:BOOL:=TRUE;

2.3 数据类型

数据是程序处理和控制的对象，在程序运行过程中，数据是通过变量来存储和传递的。变量有两个要素：名称和数据类型。对程序块或者数据块的变量声明时，都要包含这两个要素。

2.3.1 标准数据类型

标准数据类型共分为 5 大类，分别为布尔类型、整数类型、实数类型、字符串类型和时间数据类型。

表 2-2 标准数据类型

数据大类	数据类型	关键字	位数	取值范围
布尔	布尔	BOOL	1	FALSE(0)或 TEUE(1)
整型	字节	BYTE	8	0～255
	字	WORD	16	0～65535
	双字	DWORD	32	0～4294967295
	长字	LWORD	64	0～(2^64-1)
	短整型	SINT	8	-128～127
	无符号短整型	USINT	8	0～255
	整型	INT	16	-32768～32767
	无符号整型	UINT	16	0～65535
	双整型	DINT	32	-2147483648～2147483647
	无符号双整型	UDINT	32	0～4294967295
	长整型	LINT	64	-2^63～(2^63-1)
实数	实数	REAL	32	1.175494351e-38～ 3.402823466e+38
	长实数	LREAL	64	2.2250738585072014e-308～ 1.7976931348623158e+308
字符串	字符串	STRING	8*N	
时间数据	时间	TIME	32	T#0ms～T#71582m47s295ms
		TIME_OF_DAY		TOD#0:0:0～TOD#1193:02:47.295
		DATE		D#1970-1-1～D#2106-02-06
		DATE_AND_TIME		DT#1970-1-1-0:0:0～ DT#2106-02-06-06:28:15

2.3.2 复合数据类型

复合数据类型是一种由其他数据类型组合而成的，或者长度超过 32 位的数据类型，在 EURA studio 软件中常用的复合数据类型有数组（array）和结构体（struct）。

1. 数组

数组类型在 EURA studio 中被大量使用，使用数组可以有效的处理大批量数据，提高工作效率。

数组是有序数据的结合，其中的每一个元素都拥有相同的数据类型。例如一台设备共有 20 个 需要测量温度的点，如不使用数组，需要声明 nTemp1，nTemp2，……，nTemp20 共 20 个变量作为测量点对应的具体温度值，而此时，如使用数组，只需定义一个数组 nTemp，将它的成员定义为 20 个，即可完成这 20 个温度变量的定义。具体指令如下所示。

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

中括号内的数据表示 20 个测量点，例如 nTemp[17]就表示第 17 个测量点的温度值。

在 EURA studio 中可以直接定义一维、二维和三维数组作为数据类型。用户可以在 POU 的声明部分或者全局变量表中定义数组，声明数组的语法如下。

```
<数组名>:ARRAY [<ll1>..<ul1>,<ll2>..<ul2>,<ll3>..<ul3>] OF <基本数据类型>
```

ll1, ll2, ll3 表示字段范围的最小值，ul1, ul2 和 ul3 表示字段范围的最大值。字段范围必须是整数。

2. 结构体

结构(struct)是由一系列具有相同类型或不同类型的数据构成的数据集合，也叫结构体。在实际项目中，结构体是大量存在的。编程人员常使用结构体来封装一些属性来组成新的类型。所以在项目中通过对结构体内部变量的操作将大量的数据存储在内存中，以完成对数据的存储和操作。结构体其最主要的作用就是封装，封装的好处就是可以再次利用。

结构体声明的语法如下：

```
TYPE <结构名>:
STRUCT
    <变量的声明 1>
    .
    .
    <变量声明 n>
END_STRUCT
END_TYPE
```

<结构名>是一种可以在整个工程中被识别的数据类型，可以像标准数据类型一样使用。

2.3.3 其他数据类型

1. 枚举

如果一种变量有几种可能的值，可以定义为枚举类型。所谓“枚举”是将变量的值一一列举出来，变量的值只能在列举出来的值的范围内。例如，必须定义一个变量，该变量的值表示一周中的一天。该变量只能存储七个有意义的值。若要定义这些值，可以使用枚举类型。如一周内星期可能取值的集合为：

{ Sun, Mon, Tue, Wed, Thu, Fri,Sat}

该集合可定义为描述星期的枚举类型，该枚举类型共有七个元素，因而用枚举类型定义的枚举变量只能取集合中的某一元素值。

枚举类型声明的语法如下：

```
    TYPE <标识符>:
    (<Enum_0> ,
    <Enum_1>,
    …,
    <Enum_n>) |<基本数据类型>;
END_TYPE
```

<基本数据类型>为可选项，如果不填写数据类型，系统默认为 INT 类型。

2. 指针

1) 指针的概念

所谓指针就是一个地址。如果在 EURA Studio 中定义了一个变量，然后对其进行编译，系统则会给这个变量分配相应的内存单元。系统根据变量的类型分配相应的内存空间，如一个 BYTE 变量，系统则为其分配 1 个字节的存储空间，如果是一个 REAL 类型变量，系统则为其分配 4 个字节的内存存储空间。内存以字节为单位，以“地址”进行编号，地址所标志的内存单元中存放着具体的数据。

一般而言，程序都是通过变量名来对内存单元进行存储操作的。这是因为程序经过 EURA Studio 编译后，已经将变量名转换为变量的内存地址，对变量的存储其实都是通过内存地址所进行的。如假设在程序中定义了 var1，var2 和 var3 三个变量，在声明时将其都定义为 WORD 类型，经过系统编译后，系统分配给 var1 的内存空间为两个字节，地址分别为 1000 和 1001，var2 为 1002 和 1003，var3 为 1004 和 1005。1000 和 1001 内存中的具体数据则是 var1 内的具体数据，var2 和 var3 也是相同的道理，示意图如图 2.1 所示。此种按变量地址存储数据的方式在高级语言中也称之为“直接访问”方式。

变量名	数据内容
var1	Byte_Low
	Byte_High
var2	Byte_Low
	Byte_High
var3	Byte_Low
	Byte_High

内存地址	数据内容
1000	Byte_Low
1001	Byte_High
1002	Byte_Low
1003	Byte_High
1004	Byte_Low
1005	Byte_High

图 2.1 变量名对应内存地址查询变量

2) 指针变量

如果有一个变量专门用来存放另一个变量的地址（指针），则称它为“指针变量”。 在 EURA Studio 中使用关键字“POINT TO+类型”对指针变量进行声明。类型可以为变量、程序、功能块、方法和函数的内存地址。它可以指向上述的任何一个对象以及任意数据类型，包括用户定义数据类型。

声明指针的语法如下：

```
    <标识符>: POINTER TO <数据类型 | 功能块 | 程序 | 方法 | 函数>;
```

取指针地址内容即意味着读取指针当前所指地址中存储的数据。通过在指针标识符后添加内容操作符“^”，可以取得指针所指地址的内容。

3. 引用

引用是一个对象的别名。这个别名可以通过变量名读写。与指针不同的是，引用所指向的数据将被直接改变，因此引用的赋值和所指向的数据是相同的。设置引用的地址用一个特定的赋值操作完成。一个引用是否指向一个有效的数据（不等于 0），可以使用一个专门的操作符来检查，如下所示。

用以下语法声明引用，语法格式如下。

<标识符> : REFERENCE TO <数据类型>

2.4 EC400PLC 系统配置

2.4.1 PLC 设置

1. DI：以 EC404-24DTD 型 PLC 提供了 14 个数字量输入通道，用户可以根据需要设置每个通道的滤波时间。

#	名称	地址	滤波时间
1	DI 00	21	3 ms
2	DI 01	22	3 ms
3	DI 02	23	3 ms
4	DI 03	24	3 ms
5	DI 04	25	3 ms
6	DI 05	26	3 ms
7	DI 06	27	3 ms
8	DI 07	28	3 ms
9	DI 08	29	3 ms
10	DI 09	30	3 ms
11	DI 10	31	3 ms
12	DI 11	32	3 ms
13	DI 12	33	3 ms
14	DI 13	34	3 ms

图 2.2 DI 配置

2. HSC0 和 HSC1：EC404-24DTD 型 PLC 提供了 2 个高速计数器，用于对高速的脉冲输入进行计数。

在配置页面进行配置高速计数器的工作模式等参数。

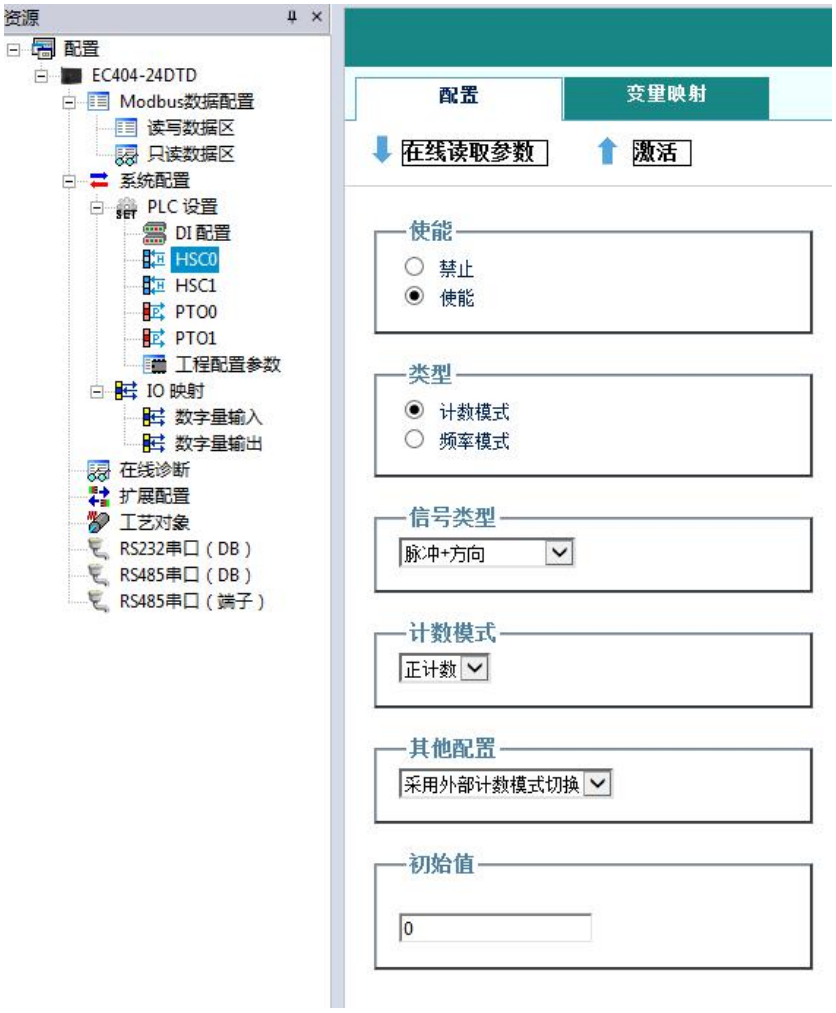


图 2.3 高速计数器配置

在变量映射页面获取高速计数器的数值。

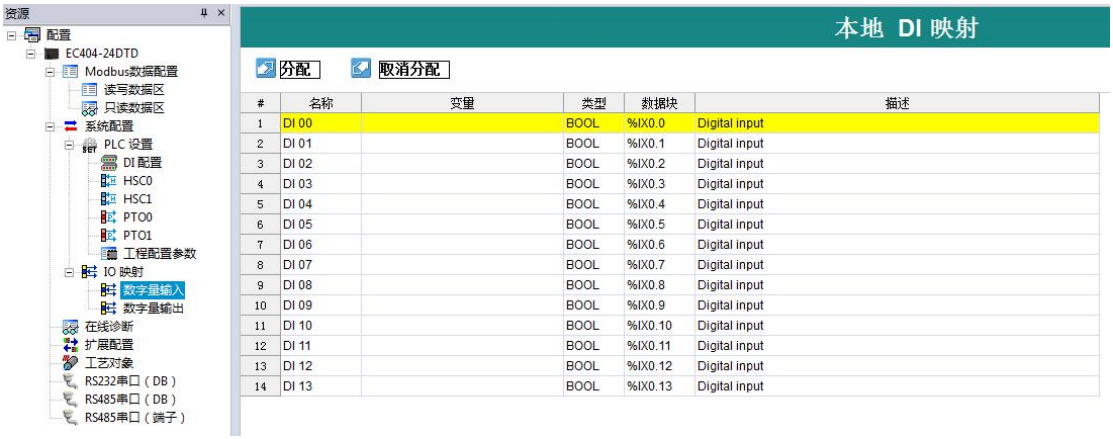
图 2.4 高速计数器变量映射

3. PTO0 和 PTO1：EC404-24DTD 型 PLC 提供了 2 路高速脉冲输出，高速脉冲输输出指的是 PTO 输出。分别在“PTO 0 配置”和“ “PTO 1 配置” ”页面配置 PTO0 和 PTO1 的参数。

图 2.5 高速脉冲输出配置

2.4.2 IO 映射

1. 数字量输入：EC404-24DTD 型 PLC 提供了 14 个数字量输入通道，可以映射到程序中的变量，用户可以在程序中在线监控输入通道的状态。高速输入计数器禁止的情况下，所有输入通道作为普通数字量输入通道使用。



#	名称	变量	类型	数据块	描述
1	DI 00		BOOL	%IX0.0	Digital input
2	DI 01		BOOL	%IX0.1	Digital input
3	DI 02		BOOL	%IX0.2	Digital input
4	DI 03		BOOL	%IX0.3	Digital input
5	DI 04		BOOL	%IX0.4	Digital input
6	DI 05		BOOL	%IX0.5	Digital input
7	DI 06		BOOL	%IX0.6	Digital input
8	DI 07		BOOL	%IX0.7	Digital input
9	DI 08		BOOL	%IX0.8	Digital input
10	DI 09		BOOL	%IX0.9	Digital input
11	DI 10		BOOL	%IX0.10	Digital input
12	DI 11		BOOL	%IX0.11	Digital input
13	DI 12		BOOL	%IX0.12	Digital input
14	DI 13		BOOL	%IX0.13	Digital input

图 2.6 本地 DI 映射

高速输入计数器使能后，根据高速输入计数器的信号类型不同，数字量输入映射随之改变。

#	名称	变量	类型	数据块	描述
1	DI 00		BOOL	%IX0.0	HSC0 Pulse/Direction A
2	DI 01		BOOL	%IX0.1	HSC0 Pulse/Direction B
3	DI 02		BOOL	%IX0.2	HSC1 CW/CCW A
4	DI 03		BOOL	%IX0.3	HSC1 CW/CCW B
5	DI 04		BOOL	%IX0.4	Digital input
6	DI 05		BOOL	%IX0.5	Digital input
7	DI 06		BOOL	%IX0.6	Digital input
8	DI 07		BOOL	%IX0.7	Digital input
9	DI 08		BOOL	%IX0.8	Digital input
10	DI 09		BOOL	%IX0.9	Digital input
11	DI 10		BOOL	%IX0.10	Digital input
12	DI 11		BOOL	%IX0.11	Digital input
13	DI 12		BOOL	%IX0.12	Digital input
14	DI 13		BOOL	%IX0.13	Digital input

图 2.7 含高速计数器的本地 DI 映射

2. 数字量输出：EC404-24DTD 型 PLC 提供了 10 个数字量输出通道，可以映射到程序中的变量，用户可以在程序中控制变量的值来控制输出通道。高速脉冲输出不使能的情况下，所有输出通道作为普通数字量输出通道使用。

#	名称	变量	类型	数据块	描述
10	DO 09		BOOL	%QX0.9	Digital output
9	DO 08		BOOL	%QX0.8	Digital output
8	DO 07		BOOL	%QX0.7	Digital output
7	DO 06		BOOL	%QX0.6	Digital output
6	DO 05		BOOL	%QX0.5	Digital output
5	DO 04		BOOL	%QX0.4	Digital output
4	DO 03		BOOL	%QX0.3	Digital output
3	DO 02		BOOL	%QX0.2	Digital output
2	DO 01		BOOL	%QX0.1	Digital output
1	DO 00		BOOL	%QX0.0	Digital output

图 2.8 DO 映射界面

高速脉冲输出使能后，根据高速脉冲输出的类型不同，数字量输出映射随之改变。

#	名称	变量	类型	数据块	描述
1	DO 00		BOOL	%QX0.0	PTO0 Pulse/Direction A
2	DO 01		BOOL	%QX0.1	PTO0 Pulse/Direction B
3	DO 02		BOOL	%QX0.2	PTO1 AB orth A
4	DO 03		BOOL	%QX0.3	PTO1 AB orth B
5	DO 04		BOOL	%QX0.4	Digital output
6	DO 05		BOOL	%QX0.5	Digital output
7	DO 06		BOOL	%QX0.6	Digital output
8	DO 07		BOOL	%QX0.7	Digital output
9	DO 08		BOOL	%QX0.8	Digital output
10	DO 09		BOOL	%QX0.9	Digital output

图 2.9 含高速脉冲输出的本地 DO 映射界面

2.5 全局变量与本地变量

2.5.1 全局变量表

全局变量也称为外部变量，其作用域是整个项目，所有程序、功能块、函数都能调用。

图 2.10 全局变量表

2.5.2 本地变量表

本地变量也称为局部变量，其作用域是本程序（或功能块、函数），其他程序、功能块和函数不能调用。

图 2.11 本地变量表

2.5.3 创建变量

在 EURA Studio 软件中，可以一次插入一个变量，也可以一次创建多个变量。

1. 创建一个变量：在变量列表任意位置，鼠标右键在弹出的菜单中选择“插入”，也可以在菜单栏中选择“变量”——“插入”，完成在变量列表中创建一个变量。

2. 创建多个变量：在菜单栏中选择“变量”——“创建多个...”，弹出“创建多个变量”对话框，填写名称、选择类型、属性等，点击“确认”完成多个变量的创建。

图 2.12 变量类型更改

2.5.4 修改变量

修改变量类型：在变量列表中，点击类型所在的列，变量类型的右侧会出现一个方形按钮，如下图所示。

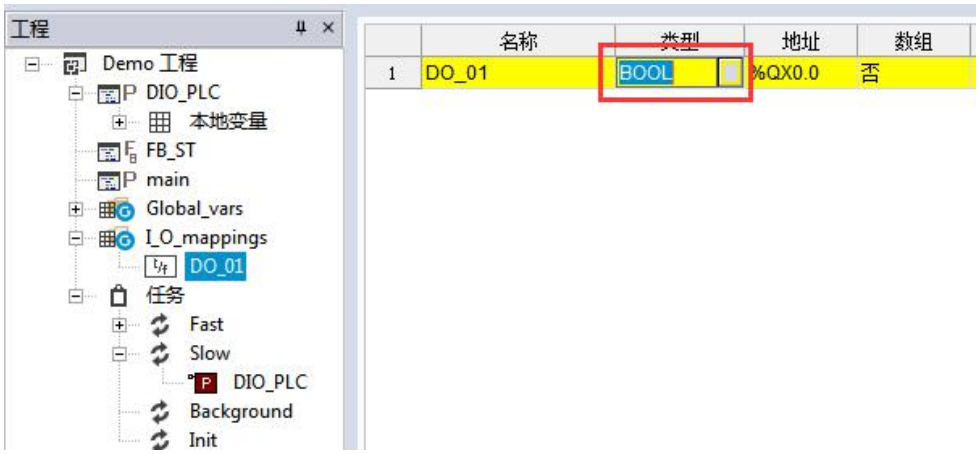


图 2.13 变量类型更改

点击变量类型右侧的方形按钮，弹出“对象浏览器”对话框，用户可以修改变量的类型。

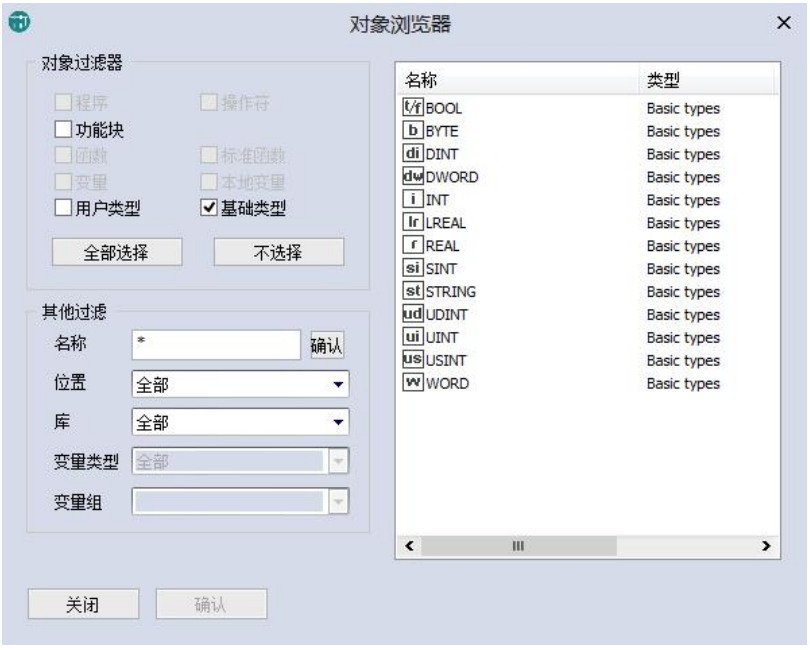


图 2.14 变量类型选择

修改变量地址：在变量列表中，点击地址所在的列，变量地址的右侧会出现一个方形按钮，如下图所示。

图 2.15 变量地址更改

点击变量地址右侧的方形按钮，弹出“变量地址”对话框，用户可以修改变量的地址。

图 2.16 变量地址更改

修改变量是否是数组：在变量列表中，点击数组所在的列，数组的右侧会出现一个方形按钮，如下图所示。

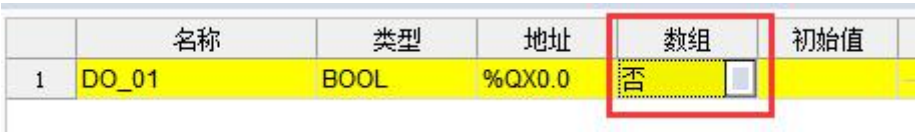


图 2.17 变量地址更改

点击右侧的方形按钮，弹出“变量组成”对话框，用户可以修改变量组成。

图 2.18 变量组成更改

修改变量初值：在变量列表中，点击初始值所在的列，数组的右侧会出现一个方形按钮，如下图所示。

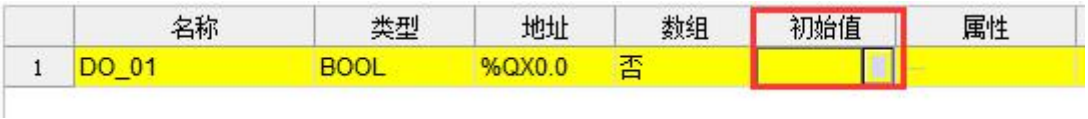


图 2.19 变量初始值更改

点击右侧的方形按钮，弹出“初始值”对话框，用户可以修改变量初始值。

图 2.20 变量初始值更改

2.6 EC400PLC 的程序结构

EC400 型 PLC 的程序结构如图所示，用户工程主要由若干个设备、 程序和任务组成。

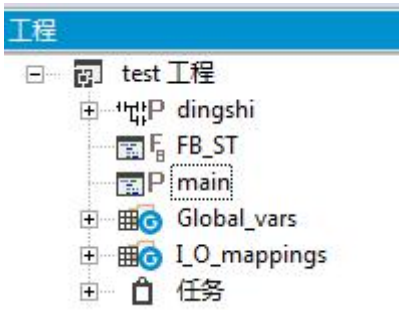


图 2.21 项目树

设备代表了一个具体的目标，即硬件对象。如本地 IO 设备、RS232、RS485。

程序组织单元（Program Organization Unit，POU）由变量区和代码区两部分组成，是用户程序的最小软件单元。按功能分程序组织单元（POU）可分为函数（FUN)、功能块（FB）和程序（PRG）。

2.6.1 函数

函数是一种可以赋予参数，但没有静态变量的程序组织单元。即用相同的输入参数调用某一函数时，该函数总能生成相同的结果作为函数值（返回值）。函数的一个重要特性是它们不能使用内部变量存储数值，这点与功能块完全不同。

函数（FUN）是没有内部状态（没有运行时的内存分配）的基本算法单元。也就是说只要给定相同的输入参数，调用函数必定得到相同的运算结果。常用的各种数学运算函数，如sin(x)、sqrt(x)等，就是典型的函数类型。

函数是有至少一个输入变量、无私有数据、仅有一个返回值的基本算法单元。函数可以被函数、功能块、程序所使用。

1. 函数的表示和声明

1) 自定义函数的表示

函数内部逻辑部分可以使用 6 种编程语言中的任意一种。函数名即是函数的返回值，也可以理解为是函数的输出值，如下为函数的语法表达式。

```
FUNCTION <函数名/返回值>: <返回值数据类型>
VAR_INPUT
    … (*函数的输入接口变量声明*)
END_VAR
VAR
    … (*函数的本地变量声明*)
END_VAR
```

```
…; (*函数内部逻辑*)
```

2) 函数中变量的声明

用户自定义函数时，应注意如下事项：

- 函数可以拥有很多个输入变量，但只能有一个返回值（输出变量）。但是并没有限制返回值的数据类型，所以可以为一个结构体作为返回值。
- 函数的重要特征是它们不能在内部变量存储数值，这点与功能块截然不同。函数没有指定的内存分配，不需要像功能块一样进行实例化。
- 函数只能调用函数，不能调用功能块。
- 配置到 VAR_INPUT 的自变量可以是空的、常数、变量或函数调用，在函数调用时，函数是作为实际的自变量被调用的。

2. 标准函数

IEC61131-3 定义的标准功能函数有以下几类：

- 算数运算符
- 位串运算符
- 移位运算符
- 选择运算符
- 比较运算符
- 地址运算符
- 类型转换运算符
- 数学函数

EURA Studio 支持所有 IEC 的 8 类标准函数。除此之外，还可以使用下列函数：ANDN、ORN、XORN、INDEXOF 和 SIZEOF、ADR、BITADR 等。

3. 函数的属性

1) 重载性

对某一个函数来说，如果其输入量以类属数据类型描述，则称为重载函数。这表示该功能的输入量不限于单一的某种数据类型，而是可用于不同的数据类型。EURA Studio 所有标准函数都具有重载属性，他能够适用于不同的数据类型。如果函数只适用于某数据类型，则需在函数名中给予声明，这称为函数的类型化。

例如一个 PLC 能识别 INT、DINT 和 SINT，则它支持类属数据类型 ANY_INT（包括 BYTE、WORD、DWORD、SINT、USINT、REAL 等）的重载功能 ADD。例如，ADD_INT 是一个限于数据类型的 INT 加法函数，它属于类型化函数，这样看重载功能是独立于类型的。重载函数说明如下图所示。

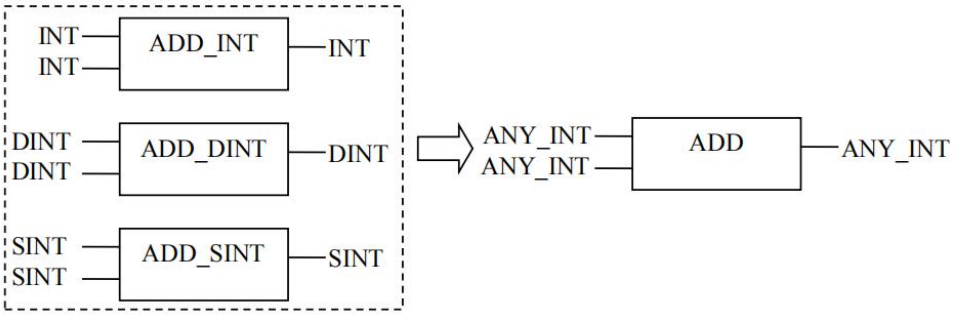


图 2.22 重载函数说明

使用重载函数时，系统会自动选择合适的数据类型。例如，如果调用的 ADD 实参数据类型是 DINT，则系统内部会调用 ADD_DINT 标准功能。

2) 可扩展性

函数的输入变量个数可以扩展的属性称为函数的可扩展属性。例如，ADD 函数的输入变量可以不仅限于两个，它可以实现多个输入变量的加法运算，因此，可以称 ADD 函数具有可扩展属性。并非所有标准函数都具有可扩展属性，该功能的扩展限度受 PLC 所强制的上限、图形编程语言中方框高度限制或函数本身功能定义上的限制，如 DIV 函数就具有该属性。具有可扩展属性的函数可简化程序，降低所需的存储空间。下图是具有可扩展属性的一些函数示例。

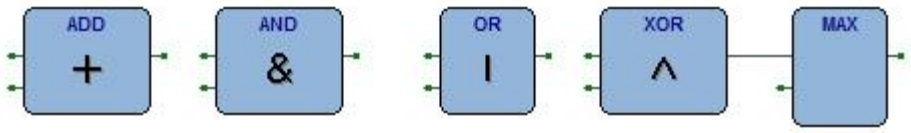


图 2.23 具有可扩展性的函数示例

3) EN 和 ENO

只有在梯形图和功能块图编程语言中该属性才有效。EN 和 ENO 分别是函数的输入使能和输出使能。所有的函数都可使用或禁用该属性。

使能输入使能输出的应用原则如下：

- 当该输入函数被调用时，EN 的值为 False，则该函数体定义的操作不会被程序执行，同时 ENO 的值为 False。
- EN 为 True 时，该函数被调用，函数体定义的操作被执行，同时 ENO 的值为 True。 EN 和 ENO 属性是附加属性，可根据实际需要使用或禁用该属性。

2.6.2 功能块

功能块（Function Block）是把反复使用的部分程序块转换成一种通用部件，它可以在程序中被任何一种编程语言所调用，反复被使用，不仅提高了程序的开发效率，也减少了编程中的错误。

功能块在执行时能够产生一个或多个值的程序组织单元。功能块保留有自己特殊的内部变量， 控制器目标执行系统必须给功能块的内部状态变量分配内存，这些内部变量构成自身的状态特征。 功能块的执行逻辑构成了自身的对象行为特征。所以，对于相同参数的输

入变量值，由于可能存在不同的内部状态变量，就可能得到不同的计算结果。在控制系统中，功能块可以是某种控制算法，例如 PID 功能模块被用于闭环控制，其他功能块可用于计数器，斜坡和滤波等。

1. 功能块的表示和声明

1) 自定义功能块的表示

与函数一样，功能块内部逻辑部分可以使用 6 种编程语言中的任意一种。函数名即是函数的返回值，也可以理解为是函数的输出值，如下为功能块的语法表达式。

```
FUNCTION_BLOCK <功能块名>
VAR_INPUT
    … (*功能块的输入接口变量声明*)
END_VAR
VAR_OUTPUT
    … (*功能块的输出接口变量声明*)
END_VAR
VAR
    … (*功能块的本地变量声明*)
END_VAR

…; (*功能块内部逻辑*)
```

2) 功能块中变量的声明

功能块中变量声明与函数中变量声明类似，编写时需注意如下事项：

- 功能块的内部和输出变量可用限定属性 RETAIN，用于表示该变量具有保持功能。而输入变量只能在调用时声明具有保持属性。
- 一般不允许对功能块输入变量赋值。只有当输入作为功能块的调用部分时，才允许对功能块输入变量赋值。
- 由于功能块可以调用函数和功能块，所以也可将调用功能块实例作为其他功能块的实例的变量。如 DB_FF (S1:=DB_ON.Q , R:=DB_OFF.Q)。功能块的输入不赋值表示保持他们的初始值。
- 为确保功能块不依赖于硬件，功能块的变量声明中不允许将具有固定地址的地址变量（如%IX1.1，%QD12）作为局部变量，但在调用时可以给其赋值。
- 使用 VAR_INPUT 和 VAR_OUTPUT 会造成占用过多的内存。为此，在功能块编程时，尽量使用 VAR_IN_OUT 替代，减少对存储区的占用。

2. 标准功能块

在标准库中已包含双稳态元素、边沿检测、计时器和定时器等功能块，后面的章节会详细对其进行说明。

3. 功能块的属性

1) 实例化

按照 IEC 61131-3 的标准，功能块的类型是抽象的结构类型的定义，而不是现实的数据实体， 如果不对其进行定义将其实例化，则不能被程序调用和执行。所以功能块是需要实例化后才能被使用。

实例化后的功能块是拥有私有数据、可按照既定逻辑完成特定功能、完全封装的、独立的结构型变量。从而将之前的抽象类型定义转换为数据实体。

【例 2.1】功能块实例化示例。

```
VAR
    EmStop : BOOL; (*布尔变量*)
    Time1 : TON; (*延时 ON 功能块*)
    Time2 : TON; (*延时 ON 功能块*)
    CountDown : CTD; (*减计数器*)
    GenCounter : CTUD; (*增/减计数器*)
END_VAR
```

上例中，尽管可以看到 Time1 和 Time2 的类型都为延时 ON 的 TON 功能块，但其实它们通过实例化后是两个独立且分开的功能块，代表了两个完全的定时功能块。

2) 扩展性

EURA Studio 支持面向对象的编程方式，所以功能块也可以派生出“子”功能块。这样“子”功能块具有“父”功能块的属性，并且可以具有自己附加的特性，可以形象的认为“子”功能块是对“父”功能块的扩展。所以在本文中，把这个叫做“功能块的扩展”。

在声明功能块时加上关键字“EXTENDS”就可以使用扩展功能。也可以通过在“添加对象”对话框添加功能块时，选择“extends”选项来实现扩展。

功能块实例名功能块类型（自定义）声明扩展功能块的格式如下：

```
FUNCTION_BLOCK <功能块名称> EXTENDS <功能块名称>
后面紧跟着是功能块中变量的声明。
```

【例 2.1】功能块的扩展性应用，定义功能块 fbA：

```
FUNCTION_BLOCK fbA
VAR_INPUT
  x:int;
  ....
```

定义功能块 fbB：

```
FUNCTION_BLOCK fbB EXTENDS fbA
VAR_INPUT
  ivar:int;
  ....
```

功能块 fbB 包含功能块 fbA 中所有的变量和方法，在使用功能块 fbA 的地方都可以用 fbB 代替。在功能块 fbB 中可以重写 fbA 中原有的方法。即可以在 fbB 中重新声明一个 fbA 中已有的方法，它的名称、输入、输出和 fbA 中的原有方法一样。fbB 中不允许使用与 fbA 中同样名称的功能块变量，否则程序在编译时会报错。使用功能块 fbB 时，可以直接使用 fbA 中的变量和方法，加上关键字**“SUPER”**即可（SUPER^.<method>）。

3) EN 和 ENO

功能块具有 EN 和 ENO 的附属属性，与函数中 EN 和 ENO 的使用方法类似。

4) 函数功能块的区别

综上所述，函数和功能块明显的区别如下表：

表 2-2 函数和功能块区别

	函数（FUN）	**功能块（FB）**
内存分配	没有指定的内存分配地址。	全部数据分配内存地址。
输入/输出变量	只允许一个输出变量。	多个输出变量或没有输出变量。
调用关系	可调用函数，但不能调用功能块。	可调用功能块或函数。

2.6.3 程序

程序（Program）是规划一个任务的主核心，程序拥有最大的调用权，可以调用功能块及函数。 一般而言分为主程序、子程序，广义上讲，也包含硬件配置、任务配置、通讯配置及目标设置信息。一般在程序中定义普通全局变量、映射硬件地址全局变量、局部变量。通过程序间调用实现应用逻辑。

1. 程序的表示和声明

程序采用如下的语法表达式表示，程序逻辑部分可以使用 6 种编程语言中的任意一种。

```
PROGRAM <程序名>
VAR_INPUT
    … (*程序的输入接口变量声明*)
END_VAR
VAR_OUTPUT
    … (*程序的输出接口变量声明*)
END_VAR
VAR
    … (*程序的本地变量声明*)
END_VAR

…; (*程序逻辑*)
```

2. 程序的性能

- 一个程序可包含地址的配置。允许声明存放 PLC 物理地址的直接表示变量，直接表示的地址配置仅用于程序中内部变量的声明。直接表示变量允许分级寻址方式描述，如可有如下的表示。可以在程序声明中按如下格式填写：

```
bTest AT %IX10.3: INT;
```

在程序编辑区可用如下的语句给直接表示变量赋值。

```
%QX0.0:=TRUE;
```

- 程序组织单元不能直接或间接调用其本身，即程序组织单元不能调用由相同类型和相同名称的程序组织单元实例。
- 程序仅在资源中实例化。在资源内被声明。程序的实例只需将程序与一个任务结合，否则程序不会被执行。而功能块仅能在程序或其他功能块中实例化。

3. 程序调用

1) 程序调用关系

在程序中允许调用功能块实例，函数甚至调用其他程序。程序组织单元的调用的关系如图 2.24 所示。

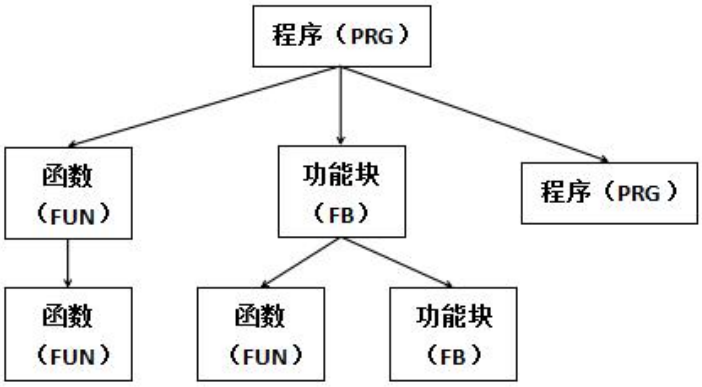


图 2.24 程序调用关系

根据图 2.17 中的显示，函数和功能块用于构成子程序，程序用于构成用户主程序，因此，程序被认为是全局的。程序是程序组织单元中的最大形式，它可以调用函数，功能块及程序。

功能块允许调用其它功能块及函数。由于函数不存在私有变量，故函数只能调用其他函数，不能调用功能块实例。

2) 如何调用程序

一般而言，在功能块实例的调用中，需要赋值的输入参数，可以将实参显式传递形参，不要赋值或保持原值的输入参数，可以不用任何显式赋值。功能块实例的调用允许不做任何形参的赋值。在函数调用中，如果出现形参，则必须将全部的形参都做显示赋值。如果不出现形参，则必须按照定义的顺序，将全部实参写入到函数的调用体内。

程序可以被其他的程序调用，但函数中不可以调用程序，程序也没有实例。

如果一个程序已调用一个子程序，从而引起这个程序的值的改变，这些改变将会保持不变，直到该 程序下一次调用这个子程序，即使其间该子程序被其他的程序调用。注意这与功能块的调用不同，只要功能块实例被调用了，而不管是被哪个程序调用的，它内部的值都会改变。

如果是在文本编辑器中，想在程序调用时设置输入/输出参数，那么可以在程序名后面的圆括 号中给各个参数赋值。对于输入参数，用":="赋值，就像在变量声明部分初始化变量一样。对于输 出参数，则使用"=>"。

■ 文本编程语言调用

使用文本编程语言时，在调用程序时必须添加括号，示例如下。

```
PRGexample();
erg := PRGexample.out_var;
```

上述例子通过先调用程序，将其输出的 out_var 变量赋值给 erg 变量。

■ 图形化编程语言调用

使用图形化编程语言时，显示的更为直观，只需要直接在接口处填写变量及数值即可，调用示例如图 2.25 所示。

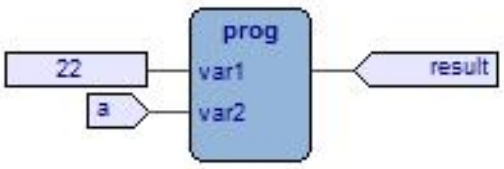


图 2.25 使用图形化编程语言调用程序

3) 函数功能块和程序的区别

综上所述，函数、功能块和程序明显的区别如下表：

表 2-3 函数、功能块和程序的区别

	函数（FUN）	功能块（FB）	程序（PRG）
内存分配	没有指定的内存分配地址。	全部数据分配内存地址。	全部数据分配内存地址。
输入/输出变量	只允许一个输出变量。	多个输出变量或没有输出变量。	多个输出变量或没有输出变量。
调用关系	可调用函数，但不能调用功能块。	可调用功能块或函数。	可调用功能块、函数和程序。

2.7 任务

EC400 是基于多任务机制执行代码，系统运行的多个功能模块以多任务执行的方式，对于用户程序，也可分为多个任务组成，根据任务优先级，分别执行。

一个程序可以用不同的编程语言来编写。典型的程序由许多互连的功能块组成，各功能块之间可互相交换数据。在一个程序中不同部分的执行通过“任务”来控制。“任务”被配置以后，可以使一系列程序或功能块周期性地执行或由一个特定的事件触发开始执行程序。

双击设备树下的任务节点，弹出任务配置对话框，用户可根据需求配置任务的周期。根据任务的类型可分为单次任务和周期任务。Init 任务只在程序初始化的时候运行一次，Background 任务用于内存的读写和 Modbus 通讯使用。Fast 和 Slow 可根据用户需求自主配置任务周期时间。

图 2.26 任务配置

2.7.1 单次任务

单次任务，只在程序初始化的时候执行一次。

2.7.2 周期任务

周期任务，按照给定的周期时间执行任务。如周期为 4ms，则每 4ms 执行一次任务中的 POU。

根据程序中所使用的指令执行与否，程序的处理时间会有所不同，所以实际执行时间在每个扫描周期都发生不同的变化，执行时间有长有短。通过使用固定周期循环方式，能保持一定的循环时间反复执行程序。即使程序的执行时间发生变化，也可以保持一定的刷新间隔时间。

第三章 编程语言

1) 梯形图（LD）：一种图形化编程语言，也是当前 PLC 使用得最多的编程语言。梯形图语言沿袭了继电器控制电路的形式，是在常用的继电器与接触器逻辑控制基础上简化了符号演变而来的，具有形象、直观、实用等特点，电气技术人员容易接受。

2) 结构化文本（ST）：文本式编程语言。语言形式与计算机高级编程语言相近，主要语法借鉴 Pascal 计算机编程语言。常用于实现复杂运算控制。

3) 顺序功能图（SFC，Sequential Function Chart）：图形化编程语言，又称为状态转移图或功能表图，常用于编制复杂的条件控制程序。

4) 功能块图（FBD）：。

5) 指令表（IL）：文本式编程语言，语言形式与计算机低阶编程语言（汇编）相近，但是不推荐在新项目中使用。

编程语言的多样性是 EURA Studio 一大优点。所以在实际的工程项目中，从优化程序和编程便利 性的角度建议用户，涉及到算法部分选择 ST 语言，编写的程序往往简洁而高效；涉及到流程控制部分，FBD 语言，编写的程序会条理清晰，逻辑关系不会混乱；涉及到逻辑控制部分，选择 LD 语言，编写的联锁，互锁等逻辑简单易懂；涉及到功能块部分，请选择 FBD，编写的程序会形成一个网络清晰的网状电路图，易于读懂。当然，在实际的编程时，用户也可以根据自己的使用习惯来选择编程语言，虽然实现的方法不同，但是都能得到同一个结果。

3.1 梯形图（LD）

3.1.1 简介

LD（梯形图）语言是 PLC 编程中被最广泛使用的一种图形化语言，源于机电一体化领域中图形表示的继电器逻辑，用它编写的程序能够与实际的电气操作原理图相对应，非常直观，易于学习和掌握。LD 语言的长处在于布尔逻辑的处理。

3.1.2 标准化图形对象

梯形图语言是对各 PLC 厂家的梯形图语言合理地吸收、借鉴，语言中的各图形符号与各 PLC 厂家的基本一致，下图为梯形图编辑器视图。

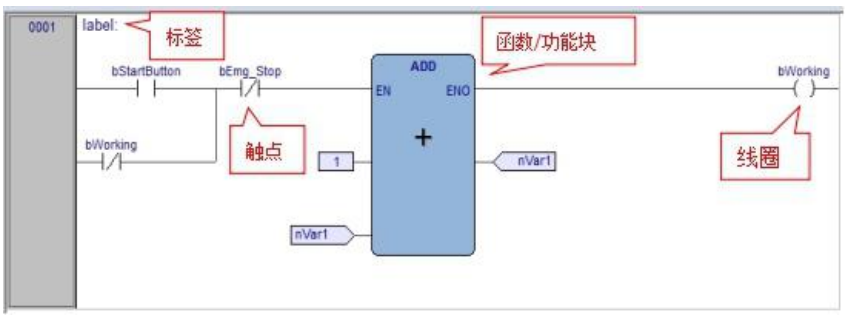


图 3.1 梯形图编辑器视图

1. 连接

LD 中使用水平连接线和垂直连接线，分别对应着串联和并联的关系。下表提到的连接线的值或者连接线某侧的值也可以理解为是否有能量流存在。

表 3-1 连接线的图形符号与说明

图形对象	名 称	描 述
——	水平连接	可以将其理解为一根理想的导线。 水平连接将线上任一点左侧的值传递到右侧。
┬ ┣ ┴	垂直连接 （含有水平连接）	垂直连接将它左侧所有水平连接的值首先进行逻辑“或”运算，然后再将运算结果传递到它右侧所有的水平连接上。

2. 触点

触点有两种类型：常开触点和常闭触点。

每个触点都必须对应一个有效的布尔型变量，变量名称被写在图形元素的上面，该变量的值决定了触点的状态（闭合或者断开），并进而决定了触点所在线路的通或者断。

表 3-2 触点元素的图形符号与说明

图形对象	名 称	描 述
变量名 ┤ ├	常开触点	若变量值为 TRUE，则触点闭合，它左侧连接的值被传递到右侧连接；否则触点断开，它右侧连接的值为 FALSE。
变量名 ┤/├	常闭触点	若变量的值为 TRUE，则触点断开，它右侧连接的值为 FALSE；否则触点闭合，它左侧连接的值被传递到右侧连接。

3. 线圈

线圈是梯形图的图形元素。梯形图中的线圈沿用了电气逻辑图的线圈术语，用于表示布尔型变量的状态变化。

表 3-3 线圈元素的图形符号与说明

图形对象	名 称	描 述
变量名 —( )—	线圈	将左侧连接的值传递给变量。

变量名 —(/)—	取反线圈	将左侧连接的值取反然后传递给变量。
变量名 —(S)—	置位线圈	若左侧连接的值为 TRUE，则变量值被置为 TRUE； 若左侧连接的值为 FALSE，则变量值保持不变。
变量名 —(R)—	复位线圈	若左侧连接的值为 TRUE，则变量值被置为 FALSE； 若左侧连接的值为 FALSE，则变量值保持不变。

4. **调用函数及功能块**

LD 支持对函数和功能块的调用。被调用的函数和功能块以矩形框来表示，其形参显示在矩形框内，实参显示在矩形框外部的连接线上。函数或者功能块的参数中至少有一个 BOOL 型的输入参数和一个 BOOL 型的输出参数，用于将它接至连接线上。

功能块必须有一个名为 EN 的 BOOL 型输入和一个名为 ENO 的 BOOL 型输出，用于控制该功能的执行。若 EN 的值为 1，则该功能被执行且 ENO 也将被置为 1；若 EN 的值为 0，则不执行该功能且 ENO 也将被置为 0。

LD 支持函数和功能块的调用。在函数和功能块调用时，需要注意如下事项：

1) 梯形图中，函数和功能块用一个矩形框表示。函数可以有多个输入参数但只能有一个返回参数。功能块可以有多个输入参数和多个输出参数。

2) 输入列于矩形框的左侧，输出列于矩形框的右侧。

3) 函数和功能块的名称显示在框内的上中部，功能块需要将其实例化，实例名列于框外的上中部。用功能块的实例名作为其在项目中的唯一标识。

4) 为了保证能流可以通过函数或功能块，每个被调用的函数或功能块至少应有一个输入和输出参数。为了使被连接的功能块执行，至少应有一个布尔输入经水平梯级连接到垂直的左电源轨线。

5) 功能块调用时，可以直接将实际参数值填写在该内部形参变量名的功能块外部连接线处。

3.1.3 EURA Studio 中的 LD 编辑器

当使用 LD 语言新建一个新程序时，就将进入 LD 编辑器；若打开一个用 LD 编写的程序，也将进入 LD 编辑器。LD 编辑器的外观如下图。

图 3.2 LD 编辑器

一个 POU 由两部分组成：参数、变量声明部分；代码部分。因此，编辑器相应地也分为两部分区域：

- 变量声明表格：用于声明该 POU 的输入/输入参数和局部变量，支持右键菜单。
- 程序编辑区：这个区域类似于一块画布，用户可以在这里输入各种 LD 图形元件。

3.2 结构化文本（ST）

3.2.1 结构化文本编程语言简介

结构化文本编程语言是一种高级语言，对于熟悉计算机高级语言开发的人员来说，结构化文本语言更易学易用，它可以实现选择、迭代、跳转语句等功能。此外，结构化文本语言还易读易理解，特别是当用有实际意义的标识符、批注来注释时，更是这样。在复杂控制系统中，结构化文本可以大大减少其代码量，使复杂系统问题变得简单，缺点是调试不直观，编译速度相对较慢。结构化文本的视图如下图所示。

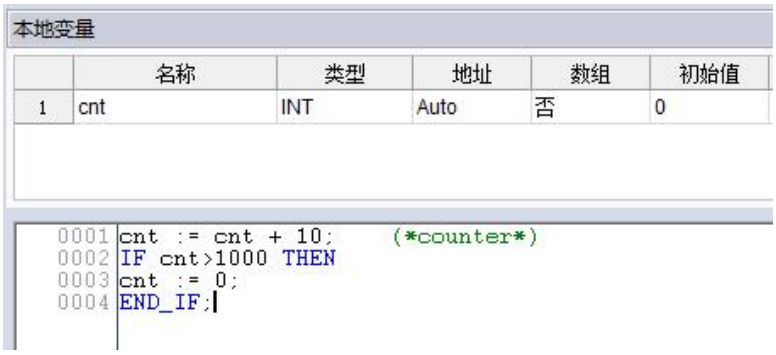


图 3.3 结构化文本视图

3.2.2 结构化文本程序执行顺序

1. **程序执行顺序**

使用结构化文本的程序执行顺序根据“行号”依次从上至下开始顺序执行，如下图所示。

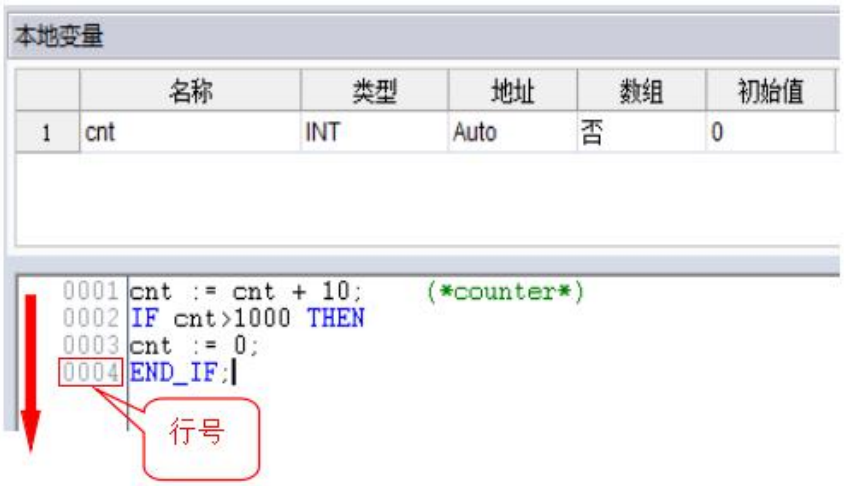


图 3.4 结构化文本程序执行顺序

2. 表达式执行顺序

表达式中包括操作符和操作数，操作数按照操作符指定的规则进行运算，得到结果并返回。操作数可以为变量、常量、寄存器地址、函数等。

如果在表达式中有若干个操作符，则操作符会按照约定的优先级顺序执行：先执行优先级高的操作符运算，再顺序执行优先级低的操作符运算。如果在表达式中具有优先级相同的操作符，则这些操作符按照书写顺序从左至右执行。操作符的优先级如下表所示：

表 3-4 操作符优先级

操作符	符号	优先级
小括号	()	最高
函数调用	Function name (Parameter list)	
求幂	EXPT	
取反	NOT	
乘法 除法 取模	* / MOD	
加法 减法	+ -	
比较	<,>,<=,>=	
等于 不等于	= <>	
逻辑与	AND	
逻辑异或	XOR	
逻辑或	OR	最低

3.2.3 指令语句

结构化文本语句表如下表所示。

表 3-5 结构化文本语句表

指令类型	指令语句	举例
赋值语句	:=	bFan:= TRUE;
功能块/函数调用	功能块/函数调名（）;	
选择语句	IF	IF <布尔表达式> THEN <语句内容>; END_IF
	CASE	
迭代语句	FOR	
	WHILE	
	REPEAT	
	REPEAT	
跳转语句	EXIT	
	CONTINUE	
	JMP	
返回语句	RETURN	
NULL 语句	;	

1. 赋值语句

赋值语句是结构化文本中最常用的语句之一，作用是将其右侧的表达式产生的值赋给左侧的操作数（变 量或地址），使用“:=”表示。 具体格式如下：

<变量>:=<表达式>;

2. 函数及功能块调用

功能块调用采用将功能块名进行实例化实现调用，如 Timer 为 TON 功能块的实例名，具体格式如下:

功能块实例名:(功能块参数);

如果需要在 ST 中调用功能块，可直接输入功能块的实例名称，并在随后的括号中给功能块的各参数分配数值或变量，参数之间以逗号隔开；功能块调用以分号结束。

例如，在结构化文本中调用功能块 TON 定时器，假设其实例名为 timer，具体实现如下图所示。

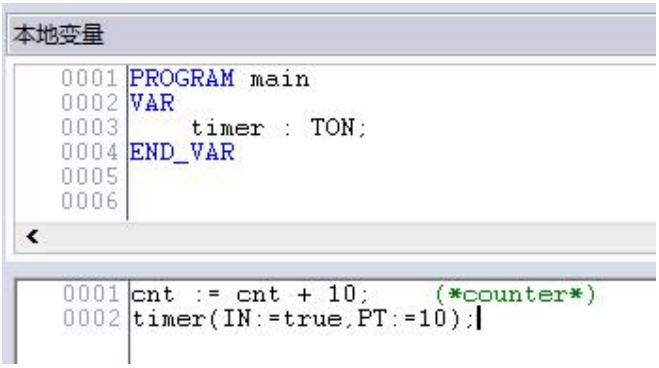


图 3.5 结构化文本调用功能块

3. 选择语句

1) IF 语句

用 IF 语句实现单分支选择结构，基本格式如下。

```
IF <布尔表达式> THEN
    <语句内容>;
END_IF
```

如果使用上述格式，只有当<布尔表达式>为 TRUE 时，才执行语句内容，否则不执行 IF 语句的<语句内容>。语句内容可以为一条语句或者可以为空语句，也可以并列多条语句，该语句表达式执行流程图如下图所示。

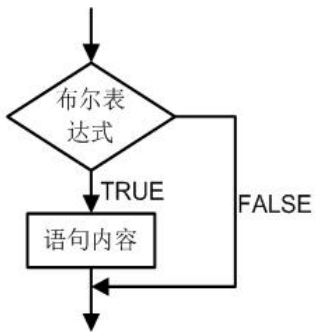


图 3.6 简单 IF 语句执行流程图

【例 3.1】使用 PLC 判断当前温度是否超过了 60 摄氏度，如果超过，始终打开风扇进行散热处理，具体实现代码如下。

```
VAR
    nTemp:BYTE;      (*当前温度状态信号*)
    bFan:BOOL;       (*风扇开关控制信号*)
END_VAR
```

```
nTemp:=80;
IF nTemp>60 THEN
    bFan:=TRUE;
END_IF
```

2) IF…ELSE 语句

用 IF 语句实现双分支选择机构，基本格式如下。

```
IF <布尔表达式> THEN
    <语句内容 1>;
ELSE
    <语句内容 2>;
END_IF
```

如上表达式先判断<布尔表达式>内的值，如果为 TRUE，则执行<语句内容 1>，如为 FALSE， 则执行<语句内容 2>，程序执行流程图如下图所示。

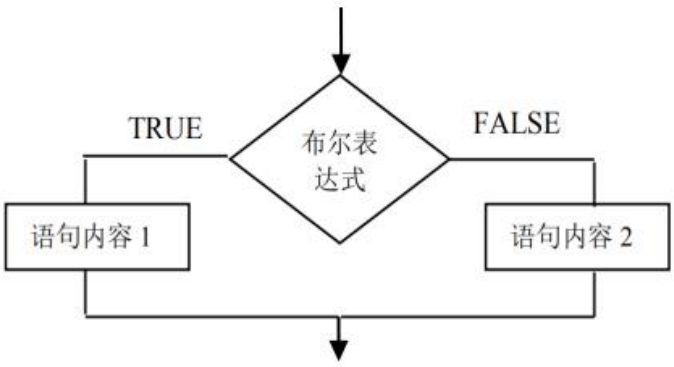


图 3.7 IF…ELSE 语句执行流程图

【例 3.2】使用 PLC 判断当温度小于到 20 摄氏度时，开启加热设备，否则（温度大于等于 20 摄氏度）加热设备断开状态。

```
VAR
    nTemp:BYTE; (*当前温度状态信号*)
    bHeating:BOOL; (*加热器开关控制信号*)
END_VAR
```

```
IF nTemp<20 THEN
    bHeating:=TRUE;
ELSE
    bHeating:=FALSE;
END_IF
```

当程序的条件判断式不止一个时，此时，需要再一个嵌套的 IF…ELSE 语句，即多分支选择结构，基本格式如下。

```
IF <布尔表达式 1> THEN
```

```
    IF <布尔表达式 2> THEN
        <语句内容 1>;
    ELSE
        <语句内容 2>;
    END_IF
ELSE
    <语句内容 3>;
END_IF
```

如上，在 IF…ELSE 中有放入了一个 IF…ELSE 语句，实现嵌套，下面通过一个例子说明嵌套的使用。

如上表达式先判断<布尔表达式 1>内的值，如果为 TRUE，则继续判断<布尔表达式 2>的值，如果<布尔表达式 1>的值为 FALSE，则执行<语句内容 3>，回到<布尔表达式 2>判断，如果<布尔表达式 2>为 TRUE，则执行<语句内容 1>，反之则执行<语句内容 2>。

【例 3.3】当设备进入自动模式后，如实际温度大于 50 摄氏度时，才开启风扇并关闭加热器，小于等于 50 摄氏度时关闭风扇打开加热器，如在手动模式时，加热器风扇均不动作。

```
VAR
    bAutoMode: BOOL;(*手/自动模式状态信号*)
    nTemp:BYTE;(*当前温度状态信号*)
    bFan:BOOL; (*风扇开关控制信号*)
    bHeating:BOOL;(*加热器开关控制信号*)
END_VAR
```

```
IF bAutoMode=TRUE THEN
    IF nTemp>50 THEN
        bFan:=TRUE;
        bHeating:=FALSE;
    ELSE
        bFan:= FALSE;
        bHeating:= TRUE;
    END_IF
ELSE
    bFan:= FALSE;
    bHeating:=FALSE;
END_IF
```

3) IF..ELSIF..ELSE 语句

此外，多分支选择结构还能通过如下方式来呈现。具体格式如下:

```
IF <布尔表达式 1> THEN
    <语句内容 1>;
ELSIF <布尔表达式 2> THEN
    <语句内容 2>;
```

```
ELSIF <布尔表达式 3> THEN
    <语句内容 3>;
    . . .
    . . .
ELSE
    <语句内容 n>;
END_IF
```

如果表达式<布尔表达式 1>为 TRUE，那么只执行指令<语句内容 1>，不执行其它指令。否则，从表达式<布尔表达式 2>开始进行判断，直到其中的一个布尔表达式为 TRUE，然后执行与此布尔表达式对应语句内容。如果布尔表达式的值都不为 TRUE，则只执行指令<语句内容 n>，程序执行流程图如下图所示。

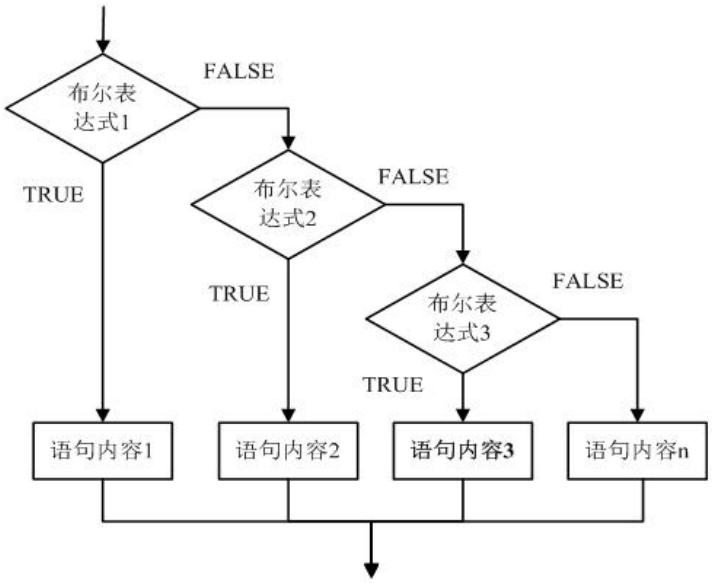


图 3.8 IF..ELSIF..ELSE 语句执行流程图

4) CASE 语句

CASE 语句是多分支选择语句，他根据表达式的值来使程序从多个分支中选择一个用于执行的分支，基本格式如下。

```
CASE <条件变量> OF
    <数值 1>: <语句内容 1>;
    <数值 2>: <语句内容 2>;
    <数值 3, 数值 4, 数值 5>: <语句内容 3>;
    <数值 6 .. 数值 10>: <语句内容 4>;
    <数值 n>: <语句内容 n>;
ELSE
    <ELSE 语句内容>;
END_CASE;
```

CASE 语句按照下面的模式进行执行：

- 如果<条件变量>的值为<数值 i>，则执行指令<语句内容 i>。
- 如果<条件变量>没有任何指定的值，则执行指令< ELSE 语句内容>。
- 如果条件变量的几个值都需要执行相同的指令，那么可以把几个值相继写在一起，并且用逗号分开。这样，共同的指令被执行，如上程序第四行。
- 如果需要条件变量在一定的范围内执行相同的指令，可以通过写入初、终值，以两个点分开。这样，共同的指令被执行，如上程序第五行。

【例 3.4】当前状态为 1 或 5 时，设备 1 运行和设备 3 停止；状态为 2 时，设备 2 停止和设备 3 运行；如当前状态在 10 至 20 之间，设备 1 和设备 3 均运行，其他情况时要求设备 1， 2， 3 均停止，具体实现的代码如下。

```
VAR
    nDevice1,nDevice2,nDevice3:BOOL; (*设备 1..3 开关控制信号*)
    nState:BYTE; (*当前状态信号*)
END_VAR
```

```
CASE nState OF
1, 5:
    nDevice1 := TRUE;
    nDevice3 := FALSE;
2:
    nDevice 2 := FALSE;
    nDevice 3 := TRUE;
10..20:
    nDevice 1 := TRUE;
    nDevice 3 := TRUE;
ELSE
    nDevice 1 := FALSE;
    nDevice 2 := FALSE;
    nDevice 3 := FALSE;
END_CASE;
```

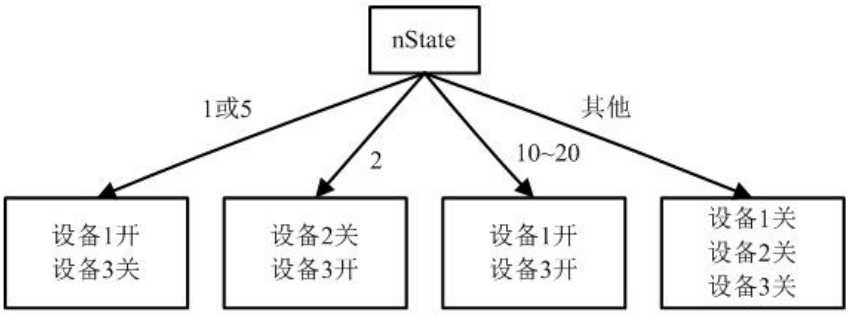


图 3.9 CASE 语句例流程图

CASE 语句流程图如上图所示，当 nState 为 1 或者 5 时，设备 1 开，设备 3 关；nState 为 2 时，设备 2 关，设备 3 开；nState 为 10~20 时，设备 1 关，设备 3 开；其他情况则设备 1 关，设备 2 关，设备 3 关。

4. 迭代语句

迭代语句主要用于重复执行的程序，在 EURA Studio 中，常见的迭代语句有 FOR，REPEAT 及 WHILE 语句。

1) FOR 循环

FOR 循环语句用于计算一个初始化序列，当某个条件为 TRUE 时，重复执行嵌套语句并计算一个迭代表达式序列，如果为 FALSE，则终止循环，具体格式如下。

```
FOR <变量> := <初始值> TO <目标值> {BY <步长>} DO
    <语句内容>
END_FOR;
```

FOR 循环的执行顺序如下：

- 计算<变量>是否在<初始值>与<目标值>的范围内。
- 当<变量>小于<目标值>，执行<语句内容>。
- 当<变量>大于<目标值>，则不会执行<语句内容>。
- 当每次执行<语句内容>时，<变量>总是按照指定的步长增加其值。步长可以是任意的整数值。如果不指定步长，则其缺省值是 1。当<变量>大于<目标值>时，退出循环。

FOR 循环是循环语句中最常用的一种，FOR 循环体现了一种规定次数、逐次反复的功能，但由于代码编写方式不同，也可以实现其他循环功能，下面通过一个实例，演示如何使用 FOR 循环。

【例 3.5】使用 FOR 循环实现 2 的五次方计算。

```
VAR
    Counter: BYTE;          (*循环计数器*)
    Var1:WORD;              (*输出结果*)
END_VAR
FOR Counter:=1 TO 5 BY 1 DO
    Var1:=Var1*2;
END_FOR;
```

假设 Var1 的初始值是 1，那么循环结束后，Var1 的值的为 32。

2) WHILE 循环

WHILE 循环与 FOR 循环使用方法类似。二者的不同之处是，WHILE 循环的结束条件可以是任意的逻辑表达式。即可以指定一个条件，当满足该条件时，执行循环，具体格式如下。

```
WHILE <布尔表达式>
    <语句内容>;
```

```
END_WHILE;
```

WHILE 循环的执行顺序如下：

- 计算<布尔表达式>的返回值。
- 当<布尔表达式>的值为 TRUE 时，重复执行<语句内容>。
- 当<布尔表达式>初始值为 FALSE，那么指令<语句内容>不会被执行，跳转至 WHILE 语句的结尾。其流程图如下图所示。

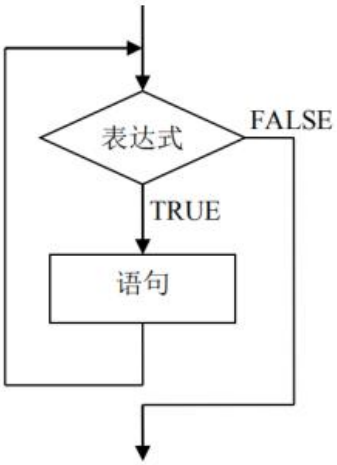


图 3.10 WHILE 语句流程图

WHILE 语句的作用像在工程中控制一台电机，当按下“启动”按钮时（布尔表达式为 TRUE 时）， 电机不停的旋转，当按下“停止”按钮后（布尔表达式为 FALSE 时），电机也立即停止。下面通过一个实例，演示如何使用 WHILE 循环。

【例 3.6】只要计数器不为零，则始终执行循环体内的程序。

```
VAR
    Counter: BYTE; (*计数器*)
    Var1:WORD;
END_VAR
```

```
WHILE Counter<>0 DO
    Var1 := Var1*2;
    Counter := Counter-1;
END_WHILE
```

在一定的意义上，WHILE 循环比 FOR 循环的功能更加强大，这是因为在执行循环之前，WHILE 循环不需要知道循环的次数。因此，多数情况下，程序中只使用这两种循环就可以了。然而，如果清楚地知道了循环的次数，那么 FOR 循环更好，因为 FOR 循环可以避免产生死循环。

3) REPEAT 循环

REPEAT 循环与 WHILE 循环不同，因为只有在指令执行以后，REPEAT 循环才检查结束条件。这就意味着无论结束条件怎样，循环至少执行一次。

具体格式如下。

```
REPEAT
    <语句内容>
UNTIL
    <布尔表达式>
END_REPEAT;
```

REPEAT 循环的执行顺序如下：

- 当<布尔表达式>的值为 FALSE 时，执行<语句内容>。
- 当<布尔表达式>的值为 TRUE 时，停止执行<语句内容>。
- 在第一次执行<语句内容>后，如果<布尔表达式>的值为 TRUE，那么<语句内容>只被执行一次。

下面通过一个实例，演示如何使用 REPEAT 循环。

【例 3.7】REPEAT 循环举例，当计数器为 0 时，则停止循环。

```
VAR
    Counter: BYTE;
END_VAR
```

```
REPEAT
    Counter := Counter+1;
UNTIL
    Counter=0
END_REPEAT;
```

此例的结果为，每个程序周期都进入该 REPEAT 循环，Counter 为 BYTE（0-255），即每个周期内都进行了 256 次自加一计算。

因为之前提到的“这就意味着无论结束条件怎样，循环至少执行一次”，所以每当进入该 REPEAT 语句时，Counter 先为 1，每个周期内都执行了 256 遍的 Counter := Counter+1 指令，直到将 Counter 变量累加至溢出为 0，跳出循环。再被加到溢出，如此往复。

5. 跳转语句

1) EXIT 语句

如果 FOR、WHILE 和 REPEAT 三种循环中使用了 EXIT 指令，那么无论结束条件如何，内循环立即停止，具体格式如下。

```
EXIT;
```

【例 3.8】使用 EXIT 指令避免当使用迭代语句时中出现除零。

```
FOR Counter:=1 TO 5 BY 1 DO
    INT1:= INT1/2;
    IF INT1=0 THEN
        EXIT;      (* 避免程序除零*)
    END_IF
    Var1:=Var1/INT1;
END_FOR
```

当 INT1 等于 0 时，则 FOR 循环结束。

2) CONTINUE 语句

该指令为 IEC 61131-3 标准的扩展指令，CONTINUE 指令可以在 FOR、WHILE 和 REPEAT 三种循环中使用。

CONTINUE 语句中断本次循环，忽略位于它后面的代码而直接开始一次新的循环。当多个循环嵌套时，CONTINUE 语句只能使直接包含它的循环语句开始一次新的循环，具体格式如下。

```
CONTINUE;
```

【例 3.9】使用 CONTINUE 指令避免当使用迭代语句时中出现除零。

```
VAR
    Counter: BYTE;      (*循环计数器*)
    INT1,Var1: INT;     (*中间变量*)
    Erg: INT;           (*输出结果*)
END_VAR
```

```
FOR Counter:=1 TO 5 BY 1 DO
    INT1:= INT1/2;
    IF INT1=0 THEN
        CONTINUE; (* 为了避免除零 *)
    END_IF
    Var1:=Var1/INT1; (*只有当 INT1 不等于 0 的情况下才执行*)
END_FOR;
Erg:=Var1;
```

3) JMP 语句

跳转语句，跳转指令可以用于无条件跳转到使用跳转好标记的代码行，具体格式如下。

```
<标识符>:
.
JMP <标识符>;
```

<标识符>可以是任意的标识符，它被放置在程序行的开始。JMP 指令后面为跳转目的地，即一个预先定义的标识符。 当执行到 JMP 指令时，将跳转到标识符所对应的程序行。

【例 3.10】使用 JMP 语句实现计数器在 0..10 范围内循环。

```
VAR
    nCounter: BYTE;
END_VAR
```

```
Label1: nCounter:=0;
Label2: nCounter:=nCounter+1;

IF nCounter<10 THEN
    JMP Label2;
ELSE
    JMP Label1;
END_IF
```

上例中的 Label1 和 Label2 属于标签，不属于变量，故在程序中不需要进行变量声明。

通过 IF 语句判断计数器是否在 0-10 的范围内，如果在范围内，则执行语句 JMP Label2，程序会在下一个周期跳转到至 Label2，执行程序 nCounter:=nCounter+1，将计数器进行自加 1，反之，则会跳转至 Label1，执行 nCounter:=0，将计数器清零。

此例子中的功能同样可以通过使用 FOR，WHILE 或者 REPEAT 循环来实现。通常情况下，应该避免使用 JMP 跳转指令，因为这样降低了代码的可读性和可靠性。

RETURN 指令

RETURN 指令是返回指令，用于退出程序组织单元（POU），具体格式如下。

```
RETURN;
```

【例 3.11】使用 IF 语句作为判断，当条件满足时，立即终止执行本程序。

```
VAR
    nCounter: BYTE;
    bSwitch: BOOL; (*开关信号*)
END_VAR
```

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

当 bSwitch 为 FALSE 时，nCounter 始终执行自加 1，如 bSwitch 为 TRUE 时，nCounter 保持上一周期的数值，立刻退出此程序组织单元（POU）。

6. 空语句

即什么内容都不执行。

具体格式如下。

```
;
```

7. 注释

注释是程序中非常重要的一部分，它使程序更加具有可读性，同时不会影响程序的执行。在 ST 编辑器的声明部分或执行部分的任何地方，都可以添加注释。

在 ST 语言中，有两种注释方法：

1) 多行注释以 (* 开始，以 *) 结束。这种注释方法允许多行注释。

2) 单行注释以“//”开始，一直到本行结束。

3.3 顺序流程图（SFC）

顺序功能流程图语言是为了满足顺序逻辑控制而设计的编程语言。编程时将顺序流程动作的过程分成步和转换条件，根据转移条件对控制系统的功能流程顺序进行分配，一步一步的按照顺序动作。每一步则代表一个控制功能任务，用方框表示。在方框内含有用于完成相应控制功能任务的梯形图逻辑。这种编程语言使程序结构清晰，易于阅读及维护，可以大大减轻编程的工作量，缩短编程和调试时间。用于系统的规模校大，程序关系较复杂的场合。它的特点是：以功能为主线，按照功能流程的顺序分配，条理清楚，便于对用户程序理解。

3.3.1 SFC 顺序流程图

SFC 程序按照梯形图表示的各步发生的具体控制把设备运行的顺序分解成不同的步。

1. **基本结构**

在 SFC 程序中，从初始步开始，当转移条件成立时按顺序执行转移条件的下一个步，通过 END 步结束一系列的动作。

1) 如果启动了 SFC 程序，将首先执行初始步。

在初始步的过程执行中，将检查初始步的下一个转移条件(图中的转移条件 0(t0))是否成立。

2) 在转移条件 0(t0)成立之前仅执行初始步。

当转移条件 0(t0)成立时，将停止初始步的执行，执行初始步的下一个步(图中的步 1(S1))。

在步 1(S1)的执行过程中，将检查步 1(S1)的下一个转移条件(图中的转移条件 1(t1))是否成立。

3) 当转移条件 1(t1)成立时，将停止步 1(S1)的执行，执行下一个步(图中的步 2(S2))。

4) 按照转移条件的成立顺序依次执行下一个步，当执行 END 步时相应块将结束。

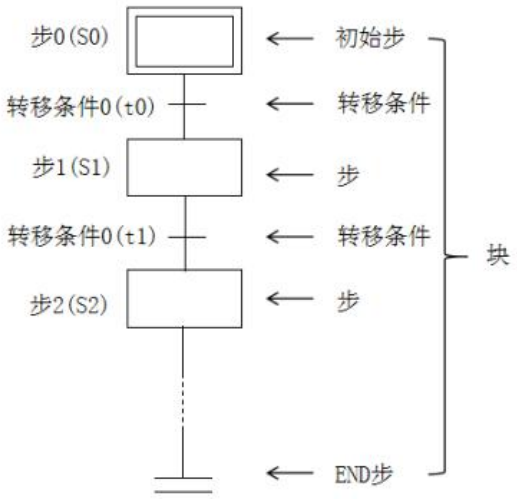


图 3.11　SFC 基本流程

2.　SFC 程序特点

1)　SFC 优点

- 执行顺控的最佳选择，将自动运行的顺序就照原样转换成图形描述即可，便于编程，且方便理解。
- 易于理解的结构化程序，可以采用图形进行层次化和模块化的编写工作，所以便于试运行以及维护工作。
- 工序（步）间不需要互锁，由于 CPU 只对动作中的步进行运算，所以前进动作和后退动作等的互锁不需要。
- 在多个工序（步）中可使用同一线圈。由于 CPU 只对动作中的步进行运算，所以即使在没有动作的步中存在相同的线圈，也不会被作为双线圈处理。（与主顺控程序中的线圈相同时，会变为双线圈处理。）
- 用图形监控动作状态。机械设备因为故障而停机时，通过监控，可以在画面上显示当前停止中的步，从而在第一时间查找出停机的原因，便于排除故障。此外，如果在各步上附有注释，那么动作时为什么会停止的原因就一目了然了。
- 设计的标准化 。按照控制流程以图形方式编程，从而达到设计图的标准化。
- 由多人分担程序编写。可以将控制内容分成多个，由不同的人编写程序，然后编辑成一个。
- 按工序进行运算处理。由于 CPU 只对动作中的步进行运算，所有通过好的编程方法可以缩短扫描时间。
- 此外，有效地使用 SFC 具备的功能，可以缩短机械动作的节拍时间。

2)　SFC 缺点

- 不适用的控制内容 。采用中断处理方式的紧急停止、一直监控、从计算机接收数据等程序都不是顺序控制，所以不适合编写 SFC 程序。(对这些内容的控制如果在主程序中编写梯形图程序，则便于汇总掌握。)
- 不习惯导致的不放心。因为不习惯，所以有不能在极其复杂的控制中使用的先入为主的成见。(通过 SFC 进行结构化和模块化，可以整理要控制的内容，所以可以只用梯形图编程。）

3.3.2 SFC 的结构

SFC 的基本元素是步加转换条件，整合各基本元素形成了几种基本结构，任何一个复杂的或简单的 SFC 结构都是以这些基本部件构成的。

1. **步**

步表示整个工业过程中的某个主要功能。它可以是特定的时间、特定的阶段或者是几个设备执行的动作。步属于 SFC 的执行体，所有的实现执行的逻辑代码都包含在其中，转换条件则是决定步的状态，当上一步的转换条件成立，这个步就被激活，被激活的步将进入执行的状态。

被激活期间，这个步被反复扫描，知道这个步的转换条件成立，步的激活被解除，退出执行到下一个步，下一个步则被激活，以此类推，直到整个 SFC 遇到 END 步，执行完毕。

SFC 顺序功能图中的步由一个方框表示，该方框内表示 “步名称”以及由连接线表示的上下转换关系。步的名称可以在当前位置直接编辑，并且步的名称必须在其所在的 POU 内是独一无二的，尤其是在 SFC 内使用动作编程时，需要特别注意。

在一个控制循环中激活步的所有动作都将执行。所以，当激活步之后的转换条件是 TRUE 时， 它之后的步被激活。当前激活的步将在下个循环中再执行。

2. 动作

每个步都可以执行多个（或单个）动作，动作包括了此步执行的详细描述，动作可以有梯形图、ST、SFC 等语言编写。

动作是步具体要执行的操作，例如，阀门的开启、电动机的起动或停止，工作或产品的移动等。

3. 转移

在步和步之间有所谓的转换。转换条件的值必须是 TRUE 或 FALSE。因而它可以是一个布尔变量、布尔地址或布尔常量。只有当步的转换条件为真时，步的转换才进行。即前步的动作执行完后，如果有出口动作则执行一次出口动作，后步如果有入口动作则执行一次后步入口的动作，然后按照控制周期执行该活动步的所有动作。

用顺序功能图编写的程序组织单元包含了一系列的步，这些步之间是通过定向连接（转换条件）实现的。

转移是以构成块的基本单位对转移条件进行指定。

转移条件是用于转移至下一个步的条件，通过条件成立转移至下一个步。

转换表示从一步到另一步的切换，这些切换是有条件的。当条件满足时转换才能发生，转换可以是布尔变量，也可以是一个逻辑判断或者是一个用编程实现的 POU。

几种常用的转移方式：

1) 串行转移

串行转移是指，通过转移条件成立转移至串行连接的下一步执行处理的方法。

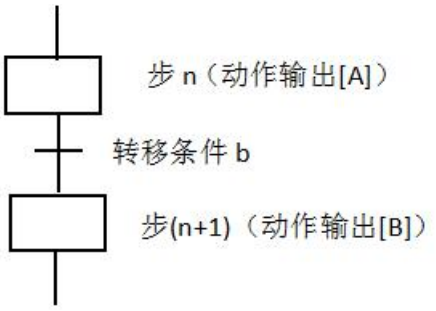


图 3.12 串行分支转移

如上图所示，执行步 n 的动作输出[A]时，如果转移条件 b 成立，则对动作输出[A]进行非执行处理，执行步(n＋1)的动作输出[B]。

2) 选择转移

选择转移是指，在并行连接的多个步中，仅执行最先成立的转移条件的步的方式。

a) 选择分支转移

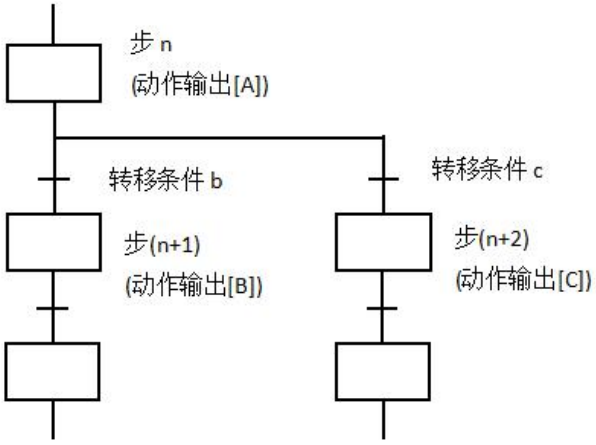


图 3.13 选择分支转移

- 执行步 n 的动作输出[A]时，选择转移条件 b 或者 c 中最先成立的条件的步(步(n+1)或者步(n+2))，执行该步的动作输出([B]或者[C])。
- 转移条件同时成立时，左侧的转移条件优先。对步 n 的动作输出[A]进行非执行处理。
- 选择后，依次执行所选列的各个步，直至进行合并为止。

b) 选择合并转移

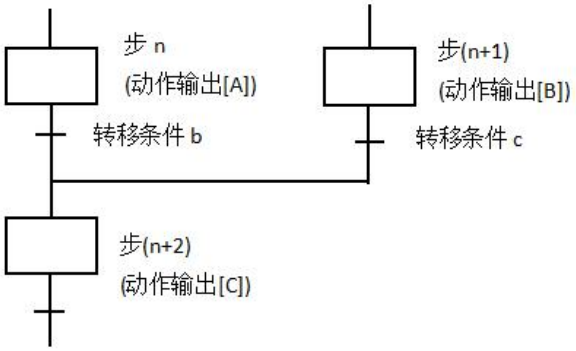


图 3.14 选择合并转移

◆ 如果分支中执行列的转移条件(b 或者 c)成立，则对执行步的动作输出([A]或者[B])进行非执行处理，执行步(n+2)的动作输出[C]。

3) 并行转移

并行转移是指，通过转移条件成立同时执行并行连接的多个步的方式。

a) 分支转移

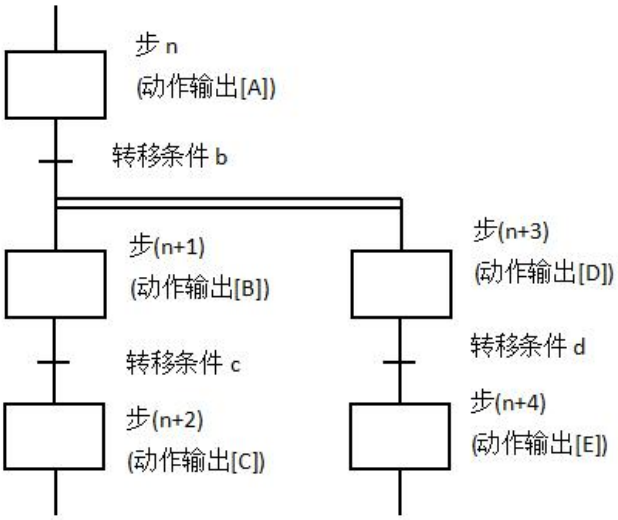


图 3.15 并行分支转移

◆ 执行步 n 的动作输出[A]时，如果转移条件 b 成立，则同时执行步(n+1)的动作输出[B]、步(n+3)的动作输出[D]。

◆ 转移条件 c 成立时转移至步(n+2)，转移条件 d 成立时转移至步(n+4)。

b) 合并转移

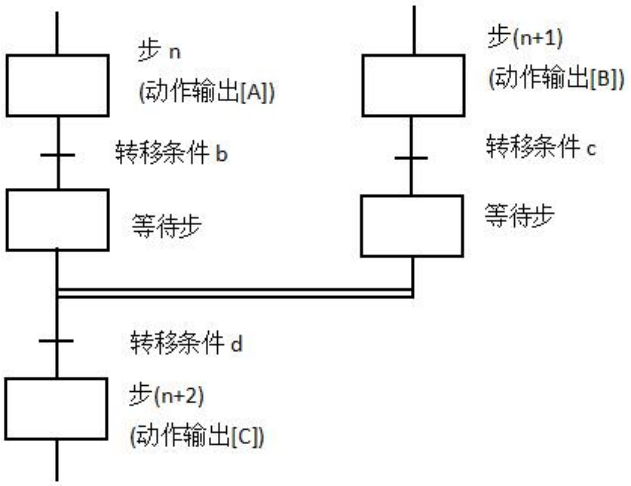


图 3.16 并行合并转移

- 执行步 n 的动作输出[A]、步(n+1)的动作输出[B]时，如果转移条件 b、转移条件 c 成立， 则分别对步 n 的动作输出[A]、步(n+1)的动作输出[B]进行非执行处理，转移至等待步。
- 等待步是用于使并行处理的步同步的步，通过将并行处理的所有的步转移至等待步，对转移条件 d 进行检查，如果转移条件 d 成立，则执行步(n+2)的动作输出[C]。
- 等待步被作为虚拟步，即使没有动作输出梯形图也没有关系。

4. 跳转

跳转是指，通过转移条件成立转移至同一个块内的指定步执行处理的方式。

跳转是通过一条竖线和水平箭头及跳转目标名展示的。

跳转定义了后续转换为 TRUE 时将要执行的步。因为处理线不能交叉或往上，跳转就需要了。

除了图最后的跳转，跳转只能用在分支的最后。当最后一个转换被选中时，可以通过 ↘ “插入前跳转”命令插入 。

跳转的目标可以通过关联的文本字符串给定，可内引编辑。它可以是一个步名或平行分支的标签。

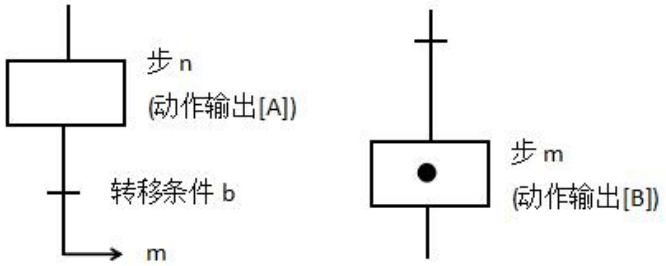


图 3.17 并行合并转移

◆ 执行步 n 的动作输出[A]时，如果转移条件 b 成立，则对动作输出[A]进行非执行处理，执行步 m 的动作输出[B]。

在并行转移内进行跳转时，只能在分支的各个纵方向进行跳转。

例如，从分支起至合并为止的纵方向内的跳转程序。

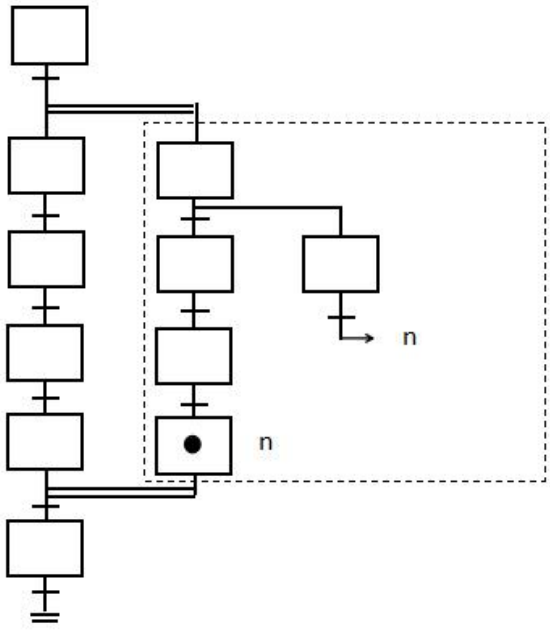


图 3.18 并行合并转移

不能创建如下所示的跳转程序：向分支内的其它纵向梯形图的跳转，向并行分支外部的跳及从并行分支外部向并行分支内的跳转。

例如，向并行分支外部的跳转程序(不能指定)。

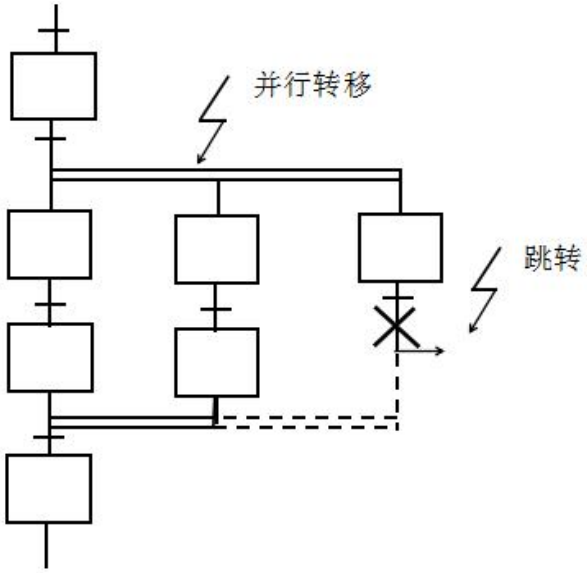


图 3.19 并行合并转移

例如，如下图所示的转移条件成立时，不要指定至当前步的跳转。

如果指定了至当前步的跳转，则无法正常动作。

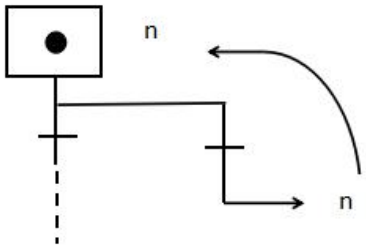


图 3.20 并行合并转移

3.4 功能块图（FBD）

3.4.1 简介

功能块图用来描述功能、功能块和程序的行为特征，还可以在顺序功能流程图中描述步、动作和转变的行为特征。功能块图与电子线路图中的信号流图非常相似，在程序中，它可看作两个过程元素之间的信息流。功能块图普遍地应用在过程控制领域。

功能块用矩形块来表示，每一功能块的左侧有不少于一个的输入端，在右侧有不少于一个的输出端，功能块的类型名称通常写在块内，但功能块实例的名称通常写在块的上部，功能块的输入输出名称写在块内的输入输出点的相应地方。

3.4.2 连接元素

FBD 在菜单栏中可以添加很多元素，本节主要介绍其中的常规连接元素，如下图所示。

图 3.21 FBD 工具栏

1. **变量/固定变量**

1) 变量

点击工具栏中的“  ”符号，在编辑区空白处点击鼠标左键，弹出变量类型选择对话框，选择变量类型为“输出”，在 FBD 编辑器中插入后的图示为“  ”。双击插入后的图标，弹出变量属性对话框，然后点击图标“  ”，弹出“对象浏览器”对话框，可以修改变量的类型。

图 3.22 修改变量属性

2) 固定变量

点击工具栏中的“ ”符号，在编辑区空白处点击鼠标左键，在 FBD 编辑器中插入后的图示为“ 0 ”。双击插入后的图标，弹出“常量数值”对话框，选择数值类型，输入数值，点击“确定”按钮。

图 3.23 常量

2. 运算块

可以代表所有的 POU，包括功能块、函数及程序，如计时器，计数器等。可以在 FBD，LD 或 IL 的节中插入。运算块可以有任意的输入，任意输出。

选中运算块拖入编辑框，弹出功能块实例对话框，输入实例名称，可以是函数名，功能块名或者程序名。

图 3.24 功能块实例化

选中功能块，点击菜单栏中的“  ”可以变成带有 EN/ENO 的功能块。

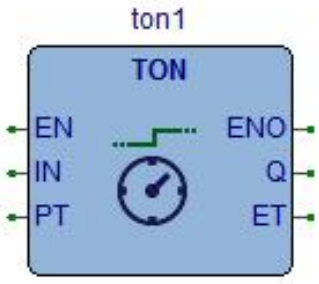


图 3.25 带有 EN/ENO 的功能块

【例 3.12】通过运算块，在 FBD 编程语言中调用定时器功能块。

新建 POU，使用 FBD 编程语言，将 TON 和 TOF 拖入编辑框，如下图所示。

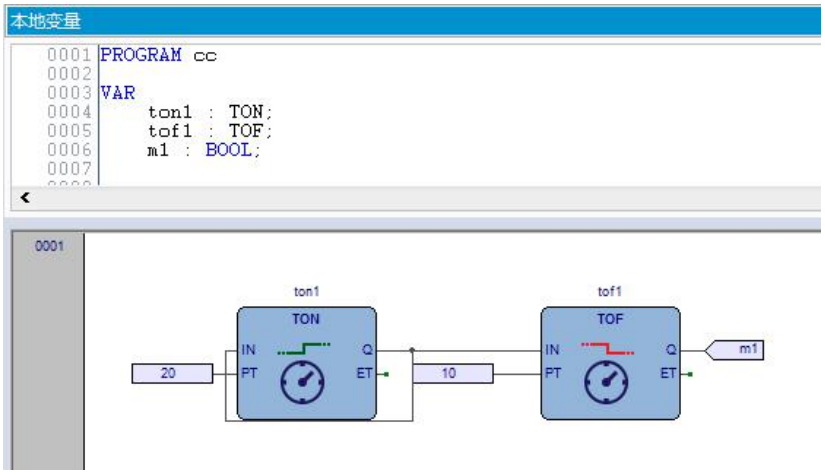


图 3.26 连续功能图调用功能块

如果需要在 ST 中调用功能块，可直接输入功能块的实例名称，并在随后的括号中给功能块的各参数分配数值或变量，参数之间以逗号隔开；功能块调用以分号结束。

若运算块被修改为另一个运算块（通过修改运算块名），而且新运算块的最大输入或输出引脚数，或者最小输入或输出引脚数与前者不同。运算块的引脚会自动做相应的调整。若要删除引脚，则首先删除最下面的引脚。

3. 表达式

点击工具栏中的“  ”符号，在编辑区空白处点击鼠标左键，弹出 Expression 对话框，在对话框中输入表达式，在 FBD 编辑器中插入后的图示为 “  ”。双击插入后的图标，可以修改表达式。

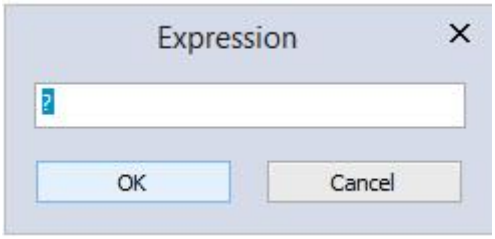


图 3.27 表达式输入

4. 返回

在 FBD 工具栏中插入“  ”符号可以实现返回功能。在 CFC 编辑器中插入后的图示为 “  ”。

命令返回一个入令插这个命注意在联机模式下带 RETURN 名字的跳转标签自动插入到第一列和编辑器最后元素的后面，在分支中，它自动跳转到执行将离开 POU 之前的地方。RETURN 指令是返回指令，用于退出程序组织单元（POU）。

5. 跳转

FBD 程序的跳转由跳转指令和标签两部分组成。

1) 跳转

点击工具栏中的“  ”符号，在编辑区空白处点击鼠标左键，弹出“标签列表”对话框，选中标签，点击确认，在 FBD 编辑器中插入后的图示为 “  ”。双击插入后的图标，可以修改标签。

图 3.28 标签列表

跳转用来指示程序下一步执行到哪，具体去哪最终“标签”所定义，如下会介绍“标签”的使用方法。

2) 标签

点击绘图，选择网络——标签，弹出新建标签对话框，在编辑框中输入标签名称，点击确认键后自动在网络的左上角添加网络标签。

图 3.29 新建标签

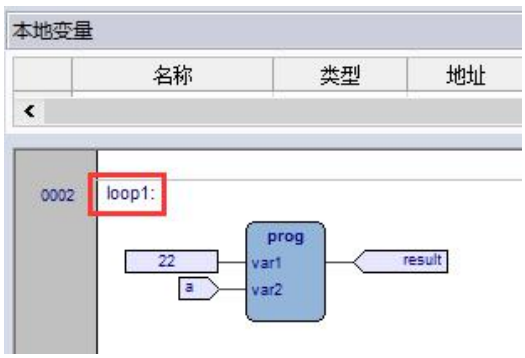


图 3.30 标签显示

标签名不属于变量，故在程序声明区中不需要对其定义。

6.注释

点击工具栏中的“  ”符号，在编辑区空白处点击鼠标左键，在 FBD 编辑器中插入后的图示为 “ 

”。双击插入后的图标，弹出注释编辑对话框。

通过添加该元素，可以在 FBD 程序中为图表添加注释。在注释编辑对话框中，直接输入注释即可。如需要换行，按<enter>键即可实现换行。

3.4.3 FBD 的组态

1. 在 FBD 程序中添加连接

点击工具栏中的“  ”符号，靠近连接程序块的管脚，管脚处会有一个方形区域出现，鼠标左键选中这个方形区域，按住鼠标至要连接的另一个连接点处松开鼠标，即完成了两部分的连接。

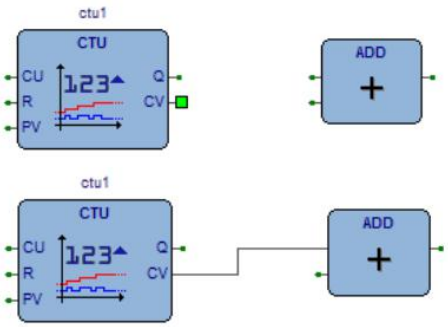


图 3.31 CFC 程序添加连接

2. 在 FBD 程序中删除连接

删除连接时，选中要删除的连接线，鼠标右键，在随后显示菜单栏中选择“删除”即可。

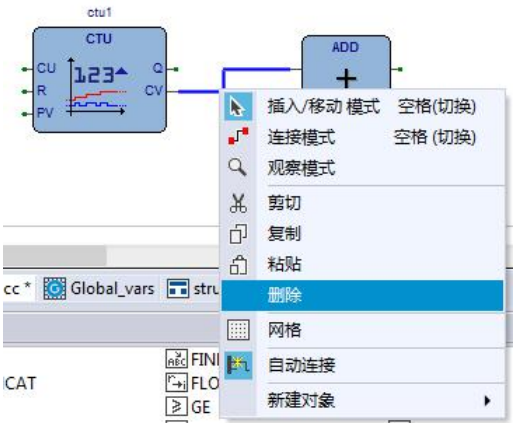


图 3.32 FBD 程序删除连接

3.5 指令表（IL）

3.5.1 指令表语言结构

指令表语言是由一系列指令组成的语言。每条指令在新一行开始，指令由操作符和紧随其后的操作数组成，操作数是指在 IEC 61131-3 的“公共元素”中定义的变量和常量。有些操作符可带若干个操作数，这时各个操作数用逗号隔开。指令前可加标号，后面跟冒号，在操作数之后可加注释。

指令表语言是一种面向行的语言，类似于程言的程序类编语汇编。一条指令，是 PLC 可执行的一项命令，它严格要求有一个行来描述。也允许空白行形式的空指令。

指令表编程语言的特点是：

- 采用助记符来表示操作功能，具有容易记忆，便于掌握。
- 与梯形图有对应关系。其特点与梯形图语言基本一致。

指令表 IL 编辑器的编程界面如下图所示。

```
0001 PROGRAM il
0002
0003 VAR
0004     bVar1 : BOOL;
0005     bVar2 : BOOL;
0006     bRes : BOOL;
0007 END_VAR
0008
0009
```

```
0001 LD      TRUE
0002 ANDN    bVar1
0003 JMPC    m1
0004 LDN     bVar2
0005 ST      bRes
0006
0007
0008 m1:
0009 LD      bVar2
0010 ST      bRes
```

图 3.33　指令表 IL 编辑器中的编程

3.5.2 连接元素

IL/FBD/LD 有共同的连接元素，只是在表达方式上略有不同。

指令表包含一系列指令。每个指令在新一行开始，包含操作符，根据操作的类型，一个或多个操作域用逗号分开。操作符可以用修饰符扩展。

在一行中的指令前，可以有身份标识符（标签），后跟一个冒号（:），如下例中所示的“ml:”。标号可以成为跳转的目标，如下例中所示的“JMPC next”。注释必须放在最后作为元素，指令之间可以插入空行。

1. 修饰符

修饰符有三种，C、N 和 N 如表 3-6 所示，修饰符不能独立构成，需要配合操作符才能构成一句完整的语句。

表 3-6 修饰符表格

修饰符	使用	功能
C	结合 JMP，CAL，RET 使用	本指令仅当前面的表达式结果为真时执行
N	结合 JMPC，CALC，RETC 使用	本指令仅当前面的表达式结果为假时执行
N	其他	操作数（而非累加器中的数）取反

2. 操作符

在引入操作符的概念之前，需要介绍一个概念，即累加器，他在指令表编程语言中尤为重要。

指令表编程语言提供了一个存储当前结果的累加器，与传统的可编程控制器使用的累加器不同，这种标准累加器存储位数是可变的，即标准指令表编程语言提供了一种存储位数可变的虚拟累加器，其存储位数取决于正在处理的操作数和数据类型。同样，虚拟累加器的数据类型也可以发生变化，以适应最新运算结果的操作数的数据类型。

指令执行过程中，数据存储采用的方法是：

运算结果：=当前运算结果 操作 操作数

因此，在操作符规定的操作下，当前运算结果与操作数进行由操作符规定的操作运算。运算结果作为新的运算将结果存放回当前运算结果的累加器。

如表 3-7 显示了可以跟修饰符配合使用的操作符指令。

表 3-7 操作符及修饰符含义

操作符	修饰符	含义	举例
LD	N	将操作数（取反）载入累加器	LD iVar
ST	N	将累加器中的数（取反）存入操作数变量中	ST iErg
S		当累加器中的数为真，将操作数（布尔型）设为真	S bVar1
R		当累加器中的数为假，将操作数（布尔型）设为真	R bVar1
AND	N,(	累加器中的数和（取反的）操作数逐位进行与运算	AND bVar2
OR	N,(	累加器中的数和（取反的）操作数逐位进行或运算	OR xVar
XOR	N,(	累加器中的数和（取反的）操作数逐位进行异或运算	XOR N,(bVar1,bVar2)
NOT		累加器中的数逐位取反	
ADD	(	将累加器中的数和操作数相加，其结果复制到累加器中	ADD (iVar1,iVar2)
SUB	(	累加器中的数减去操作数，其结果复制到累加器中	SUB iVar2

MUL	(	累加器中的数和操作数相乘，其结果复制到累加器中	MUL iVar2
DIV	(	累加器中的数除以操作数，其结果复制到累加器中	DIV 44
GT	(	检查累加器中的数是否大于操作数，其结果（布尔型）复制到累加器中 ； >	GT 23
GE	(	检查累加器中的数是否大于或等于操作数，其结果（布尔 GE iVar2 型）复制到累加器中 >=	GE iVar2
EQ	(	检查累加器中的数是否等于操作数，其结果（布尔型）复制 到累加器中; =	EQ iVar2
NE	(	检查累加器中的数是否不等于操作数，其结果（布尔型）复制到累加器中; <>	NE iVar1
LE	(	检查累加器中的数是否小于或等于操作数，其结果（布尔型）复制到累加器中; <=	LE 5
LT	(	检查累加器中的数是否小于操作数，其结果（布尔型）复制到累加器中 无条件（有条件）跳转到标签; <	LT cVar1
JMP	CN	无条件（有条件）跳转到标签	JMPN next
CAL	CN	（有条件）调用一个程序或功能块（当累加器中的数为正时）	CAL prog1
RET		从当前 POU 中返回，跳回到调用的 POU 中	RET
RET	C	有条件-仅当累加器中的数为真时，从当前 POU 中返回，跳回到调用的 POU 中	RETC
RET	CN	有条件-仅当累加器中的数为假时，从当前 POU 中返回，跳回到调用的 POU 中	RETCN
)		评估延迟的操作数	

注意：

- **累加器总是存储当前值，有后续操作时产生。**
- **CAL 的操作数应是一个被调用的功能块实例名。**
- **NOT 操作的结果是当前结果的位取反，修饰符 N 表示取反操作。**
- **RET 操作符不需要操作数**
- **修饰符 C 表示有关指令只有在当前运算的结果是布尔为 TRUE（或当操作符布尔值为 FALSE，并与“N”修饰符结合时）才执行。**
- **操作符可多于一个修正符，即可同时，也可以只用一个或不用。例如，JMP 操作符可以有 JMP、JMPC、 JMPN 三种格式。**

左圆括号“（”表示操作符的运算被延缓直到遇到右圆括号“）”。因此，对传统 PLC 中的程序块操作，主控操作等都可以采用该操作符实现。

【例 3.13】使用指令表实现电机的起保停控制。

```
0001 START:
0002 LD      bStart
0003 OR      bHold
0004 ANDN    bStop
0005 ST      bDone
```

图 3.34　起保停控制

3. 函数及功能块的调用

1) 函数调用：

在操作符区输入函数名。(第一个) 输入参数作为 LD 的操作数。如果有更多的参数，下一个必须是在同一行作为函数名输入的， 之后的参数可以添加到这一行并用逗号隔开，或添加到下面的行里。 函数返回值将被保存在累加器中，需要注意，根据 IEC 标准的规定，返回值只能有一个。

2) 功能块及程序调用:

使用 CAL 或 CALC 操作符 。在开括号前的操作域（第二栏）中输入功能块实例名或程序名。

在开括号后写入输入参数：

首栏：为参数名，操作符“:=”表示输入参数，“=>”表示输出参数。

第二栏：为实参，在最后参数的后面要以闭括号结尾。

例如：带有两个输入和两个输出参数的 POUToCAll 调用，相应的 ST 代码为:

```
POUToCall(Counter := MyCounter,
Decrement:=2,
bError=>Err,
wError=>ErrCode);
```

使用限定符编写的程序的例子：

```
LD TRUE
(* 把 TRUE 加载到累加器中*)
ANDN BOOL1
(* 执行 AND 和 BOOL1 变量的取反后“与” *)
JMPC mark
(* 当上面的结果为 TRUE 时, 跳转到标号“mark”处*)
LDN BOOL2
(* 保存 BOOL2 的反 *)
ST ERG
(* 把 BOOL2 保存在 ERG*)
Lable:
LD BOOL2
(* 保存 BOOL2 的值 *)
ST ERG
（*把 BOOL2 保存在 ERG*)
```

在 IL 中也可以在操作之后放一个圆括号。圆括号内的值被认为是一个操作数。例如：

```
LD 2
MUL 2
ADD 3
Erg
```

这里 Erg 的值为 7，但是如果加一个圆括号：

```
LD 2
MUL (2
ADD 3
)
ST Erg
```

Erg 的结果是 10，当到达")"时操作 MUL 才开始计算；此时操作数 5 计算 MUL。

```
0001 PROGRAM IL_EXAMPLE
0002
0003 VAR
0004     bErr : BOOL;
0005     wResult : WORD;
0006 END_VAR
0007
0008
```

```
0001 CAL          POUToCall(
0002     iCounter := 1,
0003   iDecrement := 1000,
0004       wError => wResult)
0005 LD           POUToCall.bError
0006 ST           bErr
```

图 3.35 IL 语句表调用功能块

第四章 基本指令

4.1 位指令

4.1.1 触点（Contact）

➢ 指令及其操作数说明

表 4-1 常开触点和常闭触点

LD	ST	功能说明

	IF IN THEN
	Statement;
	ELSE
	Statement;
	END_IF;
		常开触点

	IF NOT(IN) THEN
	Statement;
	ELSE
	Statement;
	END_IF;
		常闭触点

表 4-2 参数的数据类型

操作数	输入/输出	数据类型
IN	输入	BOOL

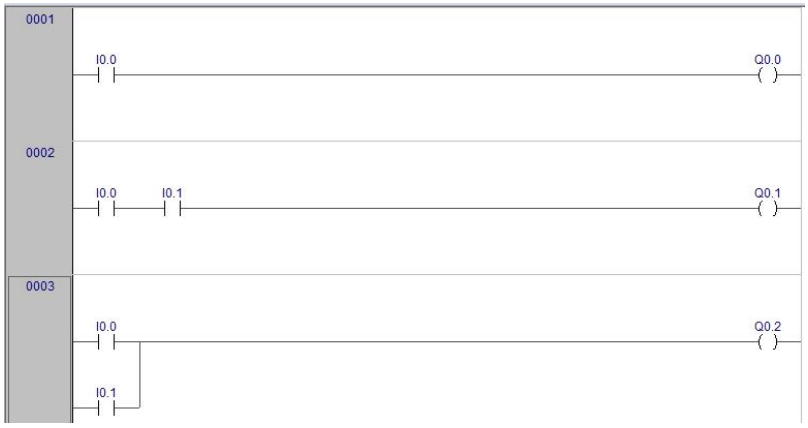
常开触点的作用是：若 IN 值为 1，则触点闭合，能量流继续向后传递；若 IN 值为 0，则触点断开，能量流也被切断。

常闭触点的作用是：若 IN 值为 0，则触点闭合，能量流继续向后传递；若 IN 值为 1，则触点断开，能量流也被切断。

➢ 指令使用举例：

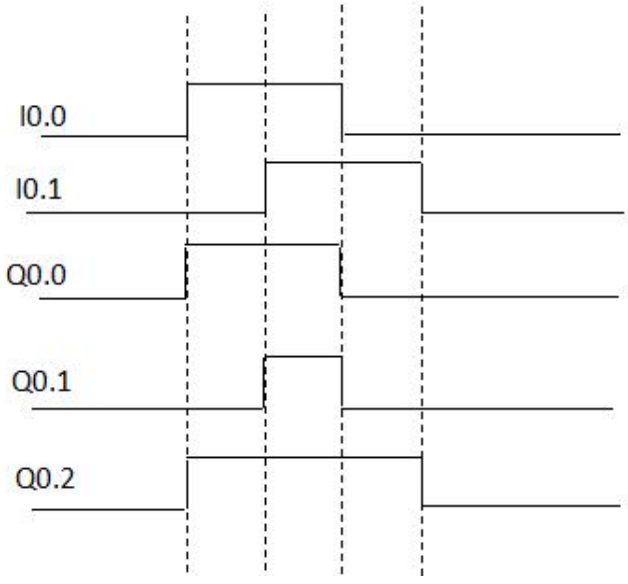
表 4-3 触点的使用举例

LD	ST




```
IF I0.0 THEN
   Q0.0:=TRUE;
ELSE
   Q0.0:=FALSE;
END_IF;
IF I0.0 AND I0.1 THEN
   Q0.1:=TRUE;
ELSE
   Q0.1:=FALSE;
END_IF;
IF I0.0 OR I0.1 THEN
   Q0.2:=TRUE;
ELSE
   Q0.2:=FALSE;
END_IF;
```

时序图



4.1.2 线圈（Coil）

- 指令及其操作数说明

表 4-4 线圈和取反线圈

LD	ST	功能说明
OUT —( )—	OUT := <布尔表达式>;	线圈

OUT -(/)-	OUT := NOT<布尔表达式>;	取反线圈

表 4-5 参数的数据类型

操作数	输入/输出	数据类型
OUT	输出	BOOL

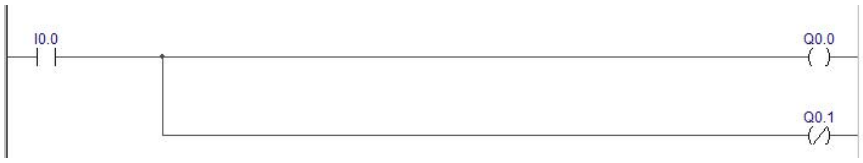
线圈的作用是：将线圈左侧能量流的值赋给 OUT 。

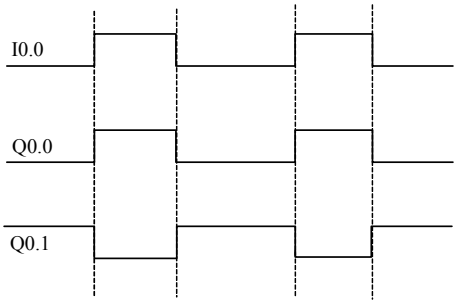
取反线圈的作用是：将线圈左侧能量流的值取反并赋给 OUT。

- 指令使用举例

表 4-6 线圈的使用举例

LD	ST

	IF I0.0 THEN
	Q0.0:=TRUE;
	Q0.1:=FALSE;
	ELSE
	Q0.0:=FALSE;
	Q0.1:=TRUE;
	END_IF;

时序图	
	

4.1.3 置位与复位

- 指令及其操作数说明

表 4-7 S 和 R 指令

LD	ST	功能说明
OUT -(R)-	不提供	复位
OUT -(S)-	不提供	置位

表 4-8 参数的数据类型

操作数	输入/输出	数据类型
OUT	输出	BOOL

复位线圈的作用是：若线圈左侧能量流的值为 1，则 OUT 值被置为 0，否则 OUT 值保持不变。

置位线圈的作用是：若线圈左侧能量流的值为 1，则 OUT 值被置为 1，否则 OUT 值保持不变。

4.1.4 边沿检测

- 指令及其参数说明

表 4-9 上升沿和下降沿跳变检测

LD	ST	功能说明
	r_trig(clk:=_in_, q=>_bool_out_);	上升沿检测
	f_trig(clk:=_in_, q=>_bool_out_);	下降沿检测

表 4-10 参数的数据类型

参数	输入/输出	数据类型
CLK	输入	BOOL
Q	输出	BOOL

R_TRIG 用于检测 CLK 输入的上升沿跳变：如果 CLK 值产生了由 0 到 1 的跳变，则 Q 输出为 1 并保持一个扫描周期，然后 Q 将输出为 0。

F_TRIG 用于检测 CLK 输入的下降沿跳变：如果 CLK 值产生了由 1 到 0 的跳变，则 Q 输出为 1 并保持一个扫描周期，然后 Q 将输出为 0。

- 指令使用举例：

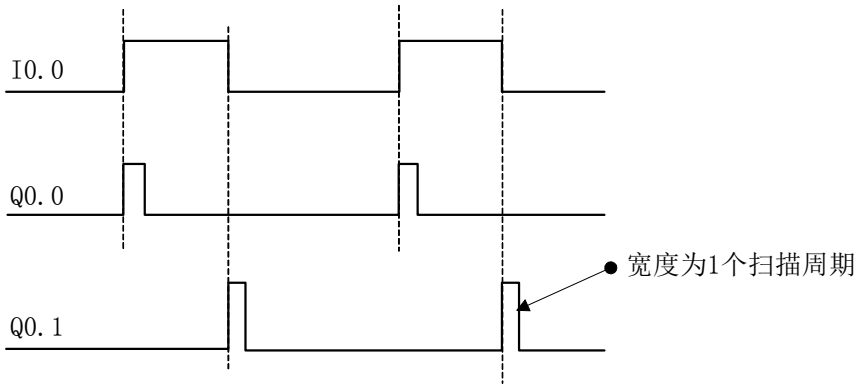
表 4-11 边沿检测使用举例

LD	ST


```
R_TRIG(
   clk:=I0.0,
   q=>Q0.0);

F_TRIG(
   clk:=I0.0,
   q=>Q0.1);
```

时序图



4.1.5 双稳态触发器

双稳态触发器是 IEC 61131-3 标准中定义的功能块之一，共有 SR 触发器（置位优先触发器）和 RS 触发器（复位优先触发器）两种。

4.1.5.1 置位优先触发器（SR）

- 指令及其操作数说明

表 4-12 置位指令

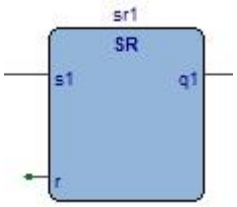
LD	ST	功能说明
	不提供	置位优先触发器

表 4-13 参数的数据类型

参数	输入/输出	数据类型
s1	输入	BOOL
r	输入	BOOL
q1	输出	BOOL

SR 触发器是置位端输入（s1）优先的触发器：若置位端输入（s1）和复位端输入（r）均为 1 时，则实例 SR1 的输出（q1）及其状态值均为 1。

SR 触发器的真值表如下：

表 4-14 SR 触发器真值表

S1	R	OUT 及 SR1 状态值
0	0	保持前一状态
0	1	0
1	0	1
1	1	1

4.1.5.2 复位优先触发器（RS）

- 指令及其操作数说明

表 4-15 复位指令

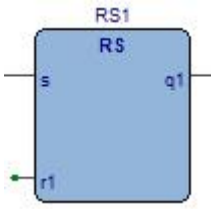
LD	ST	功能说明
	不提供	复位优先触发器

表 4-16 参数的数据类型

参数	输入/输出	数据类型
s	输入	BOOL
r1	输入	BOOL
q1	输出	BOOL

RS 触发器是复位端输入（r1）优先的触发器：若置位端输入（s）和复位端输入（r1）均为 1 时，则实例 RS1 的输出（q1）及其状态值均为 0。

RS 触发器的真值表如下：

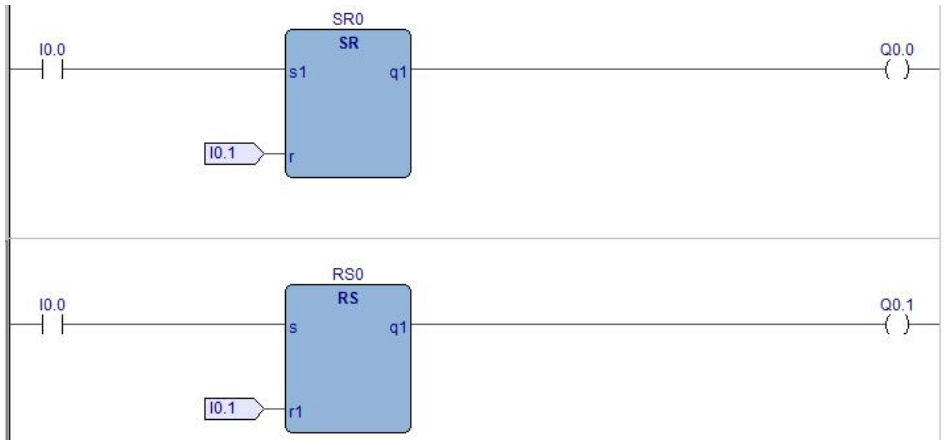
表 4-17　RS 触发器的真值表

R1	S	OUT 及 RS1 状态值
0	0	保持前一状态
0	1	1
1	0	0
1	1	0

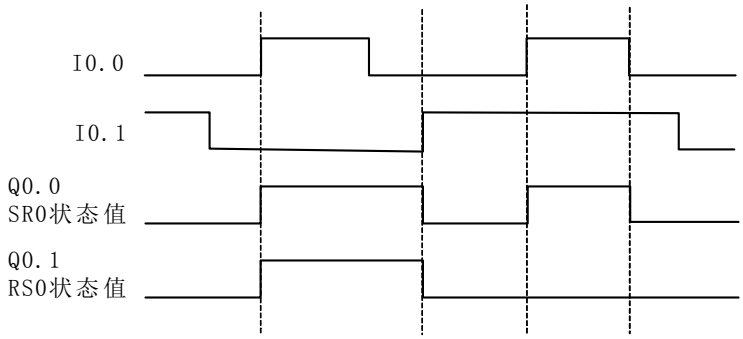
- RS、SR 使用举例

表 4-18　触发器使用举例

LD



时序图



4.2 定时器

定时器是 IEC 61131-3 标准中定义的功能块之一，共有 TON、TOF 和 TP 三种。

4.2.1 TON（接通延时定时器）

➢ 指令及其操作数说明

表 4-19 通电延时定时器指令

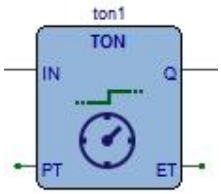
LD	ST	功能说明
	ton1(IN:=_bool_in_, PT:=_time_in_, Q=>_bool_out_, ET=>_time_out_);	TON

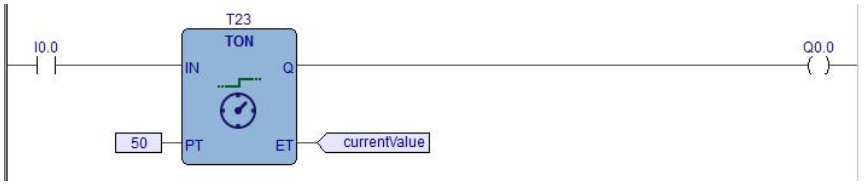
表 4-20 参数的数据类型

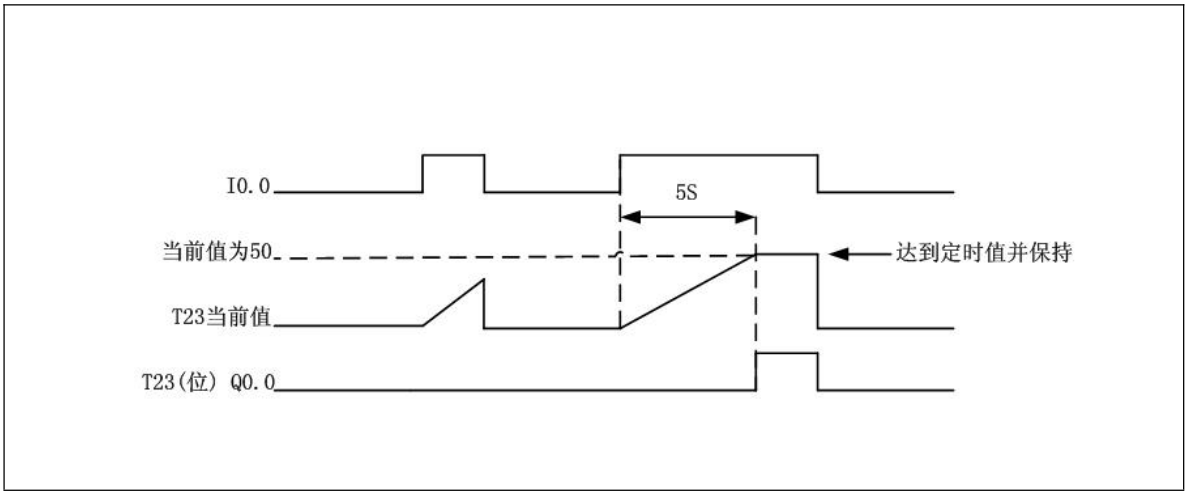
参数	输入/输出	数据类型
IN	输入	BOOL
PT	输入	INT
Q	输出	BOOL
ET	输出	INT

若检测到输入端 IN 的上升沿，则 ton1 开始启动定时，当计时值 ET 大于等于预设值 PT 时，ton1 停止，其输出 Q 及其状态值均被置为 1。若输入 IN 变为 0，则 ton1 被复位，其输出 Q 及状态值均被置为 0，同时计时值 ET 也被清零。

➢ 指令使用举例：

表 4-21 TON 使用举例

LD	ST
	T23(IN:=I0.0, PT:=50, Q=>Q0.0, ET=>currentValue);
举例结果如下：	



4.2.2 TOF（断开延时定时器）

➢ 指令及其操作数说明

表 4-22 断电延时定时器指令

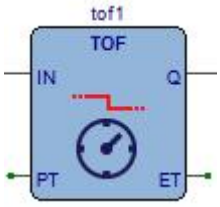
LD	ST	功能说明
	tof1(IN:=_bool_in_, PT:=_time_in_, Q=>_bool_out_, ET=>_time_out_);	TOF

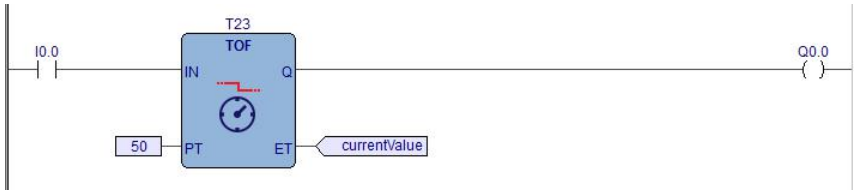
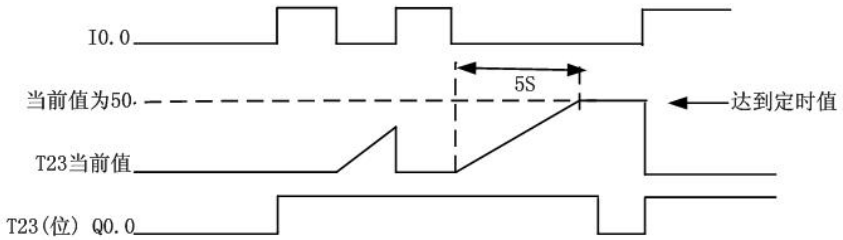
表 4-23 参数的数据类型

参数	输入/输出	数据类型
IN	输入	BOOL
PT	输入	INT
Q	输出	BOOL
ET	输出	INT

若检测到输入端 IN 的下降沿，则 tof1 开始启动定时，当计时值 ET 大于等于预设值 PT 时，tof1 停止，其输出 Q 及其状态值均被置为 0。若输入 IN 变为 1，则 tof1 被复位，其输出 Q 及状态值均被置为 1，同时计时值 ET 被清零。

➢ 指令使用举例：

表 4-24 TOF 使用举例

LD	ST
	T23(IN:=I0.0, PT:=50, Q=>Q0.0, ET=>currentValue);
举例结果如下：	
	

4.2.3 TP（脉冲定时器）

➢ 指令及其操作数说明

表 4-25 脉冲定时器指令

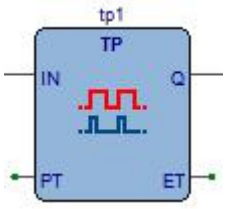
LD	ST	功能说明
	tp1(IN:=_bool_in_, PT:=_time_in_, Q=>_bool_out_, ET=>_time_out_);	TP

表 4-26 参数的数据类型

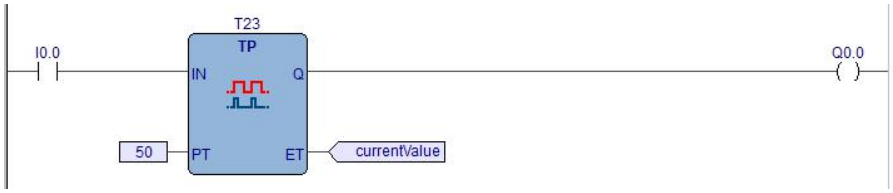
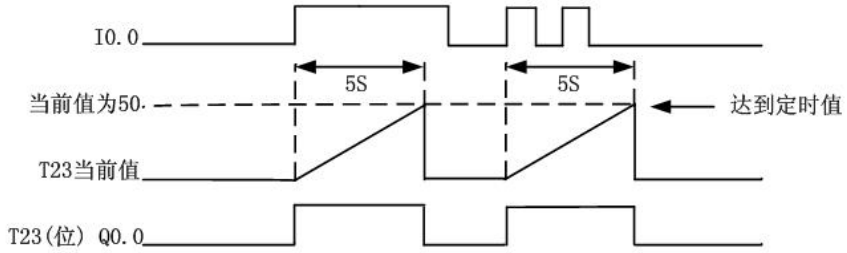
参数	输入/输出	数据类型
IN	输入	BOOL
PT	输入	INT
Q	输出	BOOL
ET	输出	INT

若检测到在输入端 IN 的上升沿，则 tp1 开始启动定时，在其输出端 Q 及其状态值输出

一个恒定宽度的脉冲，脉宽值为预设时间 PT。参数 ET 中存放着 tp1 的计时值。

➢ 指令使用举例

表 4-27 TP 使用举例

<table>
<tr><th>LD</th><th>ST</th></tr>
<tr><td> 
</td><td>T23(
IN:=I0.0,
PT:=50,
Q=>Q0.0,
ET=>currentValue);</td></tr>
<tr><td colspan="2">举例结果如下：</td></tr>
<tr><td colspan="2"> 
</td></tr>
</table>

4.3 比较指令

比较指令有等于、不等于、大于、小于、大于或等于和小于或等于。比较指令对输入操作数 1 和操作数 2 进行比较，如果比较结果为真，则逻辑运算结果为“1”，反之则为“0”。

注意：对于所有的比较指令，BYTE 类型的比较是无符号比较，其余的比较均为有符号比较。

表 4-28 比较说明

关系类型	满足以下条件时比较结果为真 ...
=	IN1 等于 IN2
<>	IN1 不等于 IN2
>	IN1 大于 IN2
<	IN1 小于 IN2
>=	IN1 大于或等于 IN2
<=	IN1 小于或等于 IN2

4.3.1 EQ（等于）

➢ 指令及其操作数说明

表 4-29　EQ 指令

LD	ST	功能说明

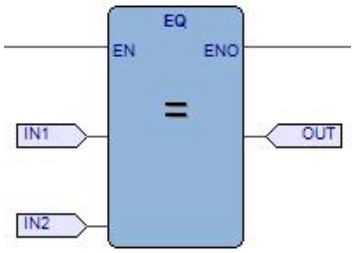
	IF IN1 = IN2 THEN
	　OUT := 1;
	ELSE
	　OUT := 0;
	END_IF;
		等于

表 4-30 参数的数据类型

参数	输入/输出	数据类型
IN1	输入	BYTE、INT、DINT、REAL
IN2	输入	BYTE、INT、DINT、REAL
OUT	输出	BOOL

参数 IN1、IN2 的数据类型必须一致。

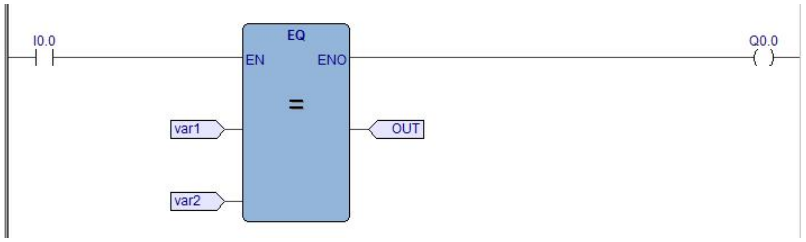
如果 EN 为 1，该指令被执行：若 IN1 等于 IN2，则在 OUT 输出为 1，否则输出为 0；

如果 EN 为 0，该指令不执行，在 OUT 端输出为 0。

- 指令使用举例：

表 4-31 等于使用举例

LD	ST

	IF var1 = var2 THEN
	　OUT := 1;
	ELSE
	　OUT := 0;
	END_IF;

4.3.2 NE（不等于）

- 指令及其操作数说明

表 4-32　NE 指令

LD	ST	功能说明

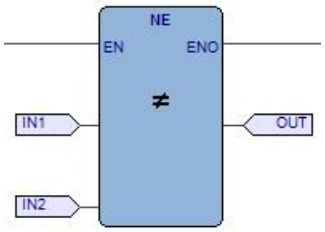
	IF IN1 <> IN2 THEN
	　OUT := 1;
	ELSE
	　OUT := 0;
	END_IF;
		不等于

表 4-33 参数的数据类型

参数	输入/输出	数据类型
IN1	输入	BYTE、INT、DINT、REAL
IN2	输入	BYTE、INT、DINT、REAL
OUT	输出	BOOL

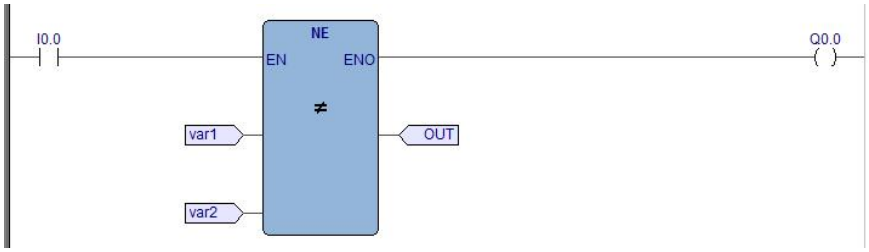
参数 IN1、IN2 的数据类型必须一致。

若 EN 为 1，该指令被执行：若 IN1 不等于 IN2，则在 OUT 输出为 1，否则输出为 0；若 EN 为 0，该指令不执行，在 OUT 端输出为 0。

- 指令使用举例：

表 4-34 不等于指令使用举例

LD	ST

	IF var1 <> var2 THEN
	　OUT := 1;
	ELSE
	　OUT := 0;
	END_IF;

4.3.3 GT（大于）

➢ 指令及其操作数说明

表 4-35 GT 指令

LD	ST	功能说明

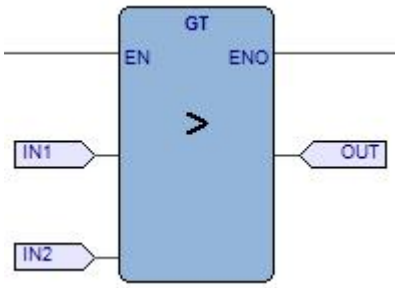
	IF IN1 > IN2 THEN
	OUT := 1;
	ELSE
	OUT := 0;
	END_IF;
		大于

表 4-36 参数的数据类型

参数	输入/输出	数据类型
IN1	输入	BYTE、INT、DINT、REAL
IN2	输入	BYTE、INT、DINT、REAL
OUT	输出	BOOL

参数 IN1、IN2 的数据类型必须一致。

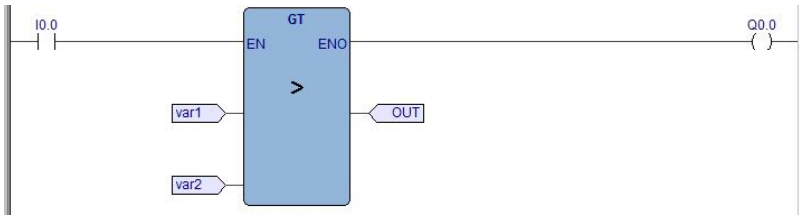
如果 EN 为 1，该指令被执行：若 IN1 大于 IN2，则在 OUT 输出为 1，否则输出为 0；

如果 EN 为 0，该指令不执行，在 OUT 端输出为 0。

➢ 指令使用举例：

表 4-37 参数的数据类型

LD	ST

	IF var1 > var2 THEN
	OUT := 1;
	ELSE
	OUT := 0;
	END_IF;

4.3.4 GE（大于等于）

➢ 指令及其操作数说明

表 4-38 GE 指令

LD	ST	功能说明

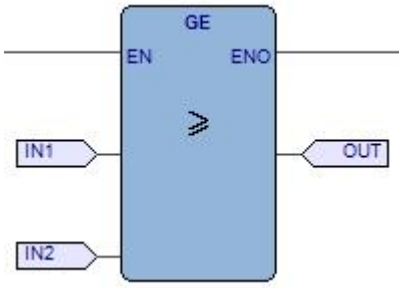
	IF IN1 >= IN2 THEN
	OUT := 1;
	ELSE
	OUT := 0;
	END_IF;
		大于等于

表 4-39 参数的数据类型

参数	输入/输出	数据类型
IN1	输入	BYTE、INT、DINT、REAL
IN2	输入	BYTE、INT、DINT、REAL
OUT	输出	BOOL

参数 IN1、IN2 的数据类型必须一致。

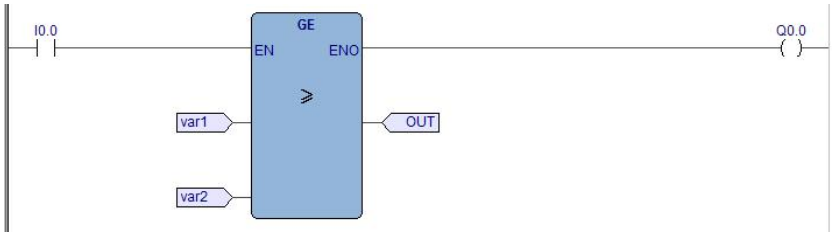
如果 EN 为 1，该指令被执行：若 IN1 大于等于 IN2，则在 OUT 输出为 1，否则输出为 0；

如果 EN 为 0，该指令不执行，在 OUT 端输出为 0。

➢ 指令使用举例：

表 4-40 大于等于指令使用举例

LD	ST

	IF var1 >= var2 THEN
	OUT := 1;
	ELSE
	OUT := 0;
	END_IF;

4.3.5 LT（小于）

- 指令及其操作数说明

表 4-41　LT 指令

LD	ST	功能说明

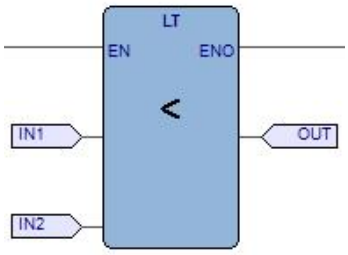
	IF IN1 < IN2 THEN
	　OUT := 1;
	ELSE
	　OUT := 0;
	END_IF;
		小于

表 4-42 参数的数据类型

参数	输入/输出	数据类型
IN1	输入	BYTE、INT、DINT、REAL
IN2	输入	BYTE、INT、DINT、REAL
OUT	输出	BOOL

参数 IN1、IN2 的数据类型必须一致。

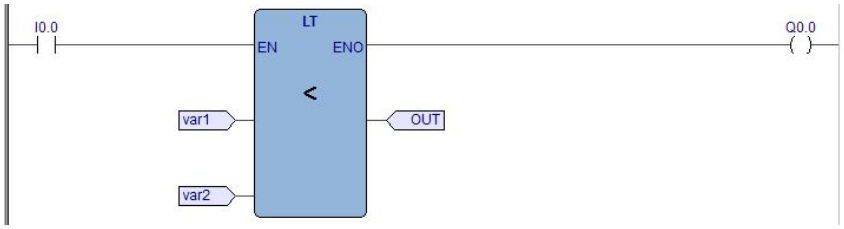
如果 EN 为 1，该指令被执行：若 IN1 小于 IN2，则在 OUT 输出为 1，否则输出为 0；

如果 EN 为 0，该指令不执行，在 OUT 端输出为 0。

- 指令使用举例：

表 4-43 小于指令使用举例

LD	ST

	IF var1 < var2 THEN
	　OUT := 1;
	ELSE
	　OUT := 0;
	END_IF;

4.3.6 LE（小于等于）

➢ 指令及其操作数说明

表 4-44 LE 指令

LD	ST	功能说明

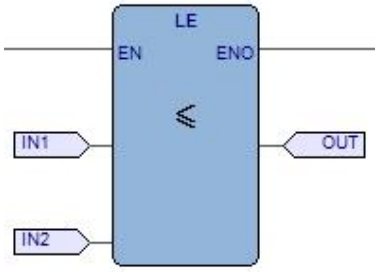
	IF IN1 <= IN2 THEN
	OUT := 1;
	ELSE
	OUT := 0;
	END_IF;
		小于等于

表 4-45 参数的数据类型

参数	输入/输出	数据类型
IN1	输入	BYTE、INT、DINT、REAL
IN2	输入	BYTE、INT、DINT、REAL
OUT	输出	BOOL

参数 IN1、IN2 的数据类型必须一致。

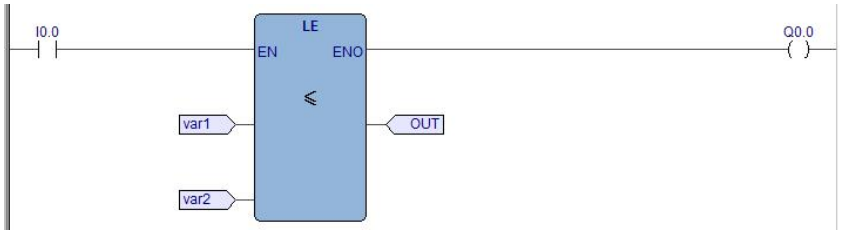
如果 EN 为 1，该指令被执行：若 IN1 小于等于 IN2，则在 OUT 输出为 1，否则输出为 0；

如果 EN 为 0，该指令不执行，在 OUT 端输出为 0。

➢ 指令使用举例：

表 4-46 小于等于指令举例

LD	ST

	IF var1 <= var2 THEN
	OUT := 1;
	ELSE
	OUT := 0;
	END_IF;

4.4 逻辑运算指令

4.4.1 NOT（按位取反）

➢ 指令及其操作数说明

表 4-47　NOT 指令

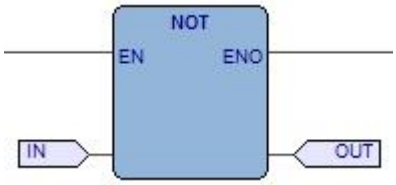
LD	ST	功能说明
	OUT:=NOT(IN);	取反

表 4-48 参数的数据类型

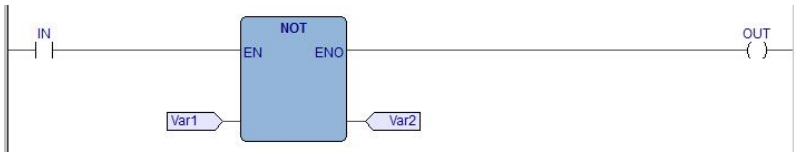
参数	输入/输出	数据类型
IN	输入	BYTE、WORD、DWORD
OUT	输出	BYTE、WORD、DWORD

参数 IN、OUT 的数据类型必须一致。

如果 EN 为 1，则该指令被执行：将 IN 的每一个二进制位都取反，然后将结果赋给 OUT。

- 指令使用举例：

表 4-49 取反指令使用举例

LD	ST
	Var2:=NOT Var1;
若 NOT 被执行，则结果如下：	
Var1　16#FFFF　　Var2　16#0000	

4.4.2 AND（按位与）

- 指令及其操作数说明

表 4-50　AND 指令

LD	ST	功能说明

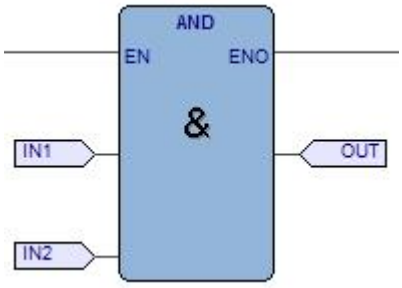
	OUT:=IN1 AND IN2;	按位与

表 4-51 参数的数据类型

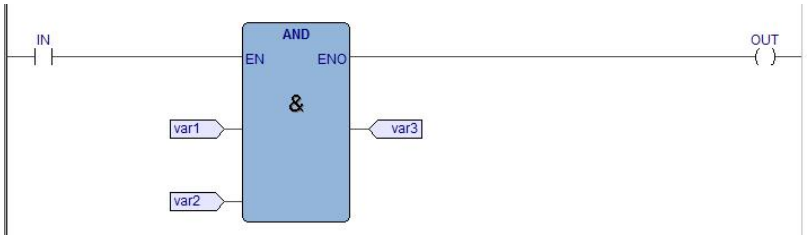
参数	输入/输出	数据类型
IN1	输入	BYTE、WORD、DWORD
IN2	输入	BYTE、WORD、DWORD
OUT	输出	BYTE、WORD、DWORD

参数 IN1、IN2、OUT 的数据类型必须一致。

若 EN 为 1，则该指令被执行：将 IN1 和 IN2 按二进制位进行“与”运算后，将结果赋给 OUT。

- 指令使用举例：

表 4-52 AND 指令使用举例

LD	ST
	var3:=var1 AND var2;
若 AND 被执行，则结果如下：	
var1 16#129B　　var2 16#960F　　var3 16#120B	

4.4.3 OR（按位或）

- 指令及其操作数说明

表 4-53　OR 指令

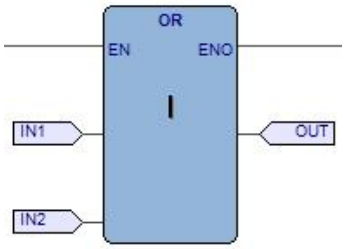
LD	ST	功能说明
	OUT := IN1 OR IN2;	按位或

表 4-54 参数的数据类型

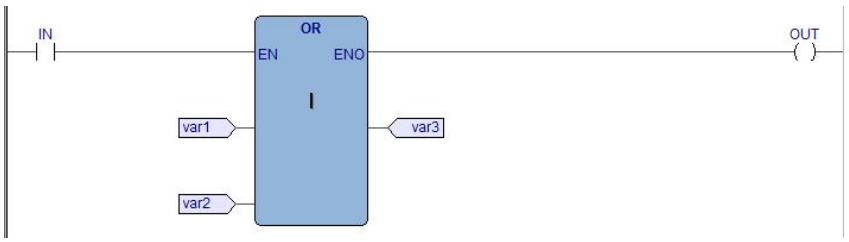
参数	输入/输出	数据类型
IN1	输入	BYTE、WORD、DWORD
IN2	输入	BYTE、WORD、DWORD
OUT	输出	BYTE、WORD、DWORD

参数 IN1、IN2、OUT 的数据类型必须一致。

若 EN 为 1，则该指令被执行：将 IN1 和 IN2 按二进制位进行“或”运算后，将结果赋给 OUT。

- 指令及其操作数说明

表 4-55 OR 指令使用举例

LD	ST
	var3:=var1 OR var2;
若 OR 被执行，则结果如下：	
var1 16#5555 var2 16#AAAA var3 16#FFFF	

4.4.4 XOR（按位异或）

- 指令及其操作数说明

表 4-56　XOR 指令

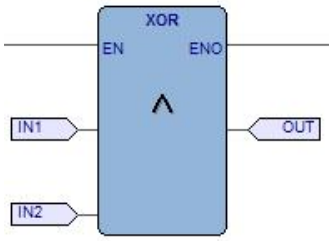
LD	ST	功能说明
	OUT := IN1 XOR IN2;	按位异或

表 4-57 参数的数据类型

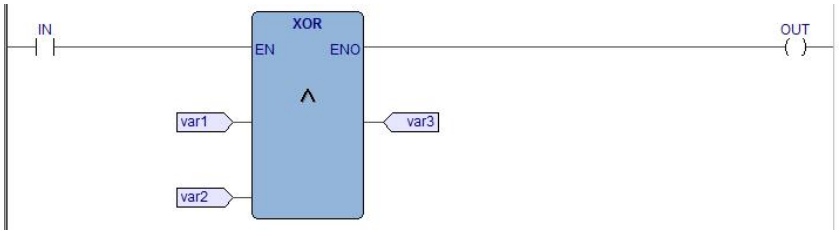
参数	输入/输出	数据类型
IN1	输入	BYTE、WORD、DWORD
IN2	输入	BYTE、WORD、DWORD
OUT	输出	BYTE、WORD、DWORD

参数 IN1、IN2、OUT 的数据类型必须一致。

若 EN 为 1，则该指令执行：将 IN1 和 IN2 按二进制位进行“异或”运算后，将结果赋给 OUT。

- 指令及其操作数说明

表 4-58　XOR 指令使用说明

LD	ST
	var3:=var1 XOR var2;
若 OR 被执行，则结果如下：	
var1 16#9514　var2 16#B9A1　var3 16#2CB5	

4.5 移位指令

4.5.1 SHL（左移）

➢ 指令及其操作数说明

表 4-59 SHL 指令

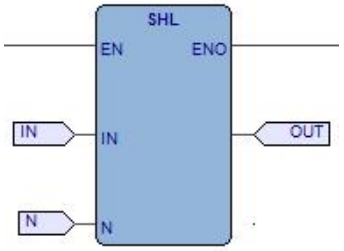
LD	ST	功能说明
	OUT:=SHL(IN:=_variant_in_, N:=_uint_in_);	左移

表 4-60 参数的数据类型

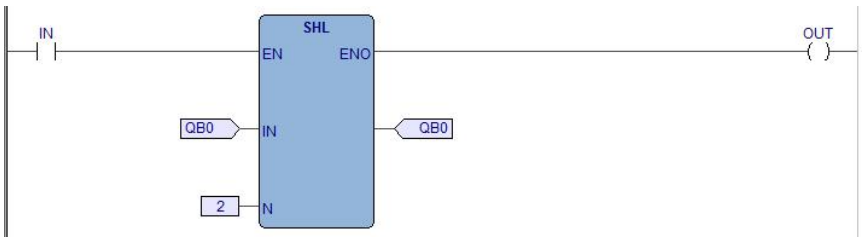
参数	输入/输出	数据类型
IN	输入	BYTE、WORD、DWORD
N	输入	BYTE
OUT	输出	BYTE、WORD、DWORD

参数 IN、OUT 的数据类型必须一致。

若 EN 为 1，则该指令被执行：将 IN 的全部二进制位向左移动 N 位，移出的高位被舍弃并且低位补 0，最终的结果赋给 OUT。

➢ 指令使用举例：

表 4-61 SHL 指令使用说明

LD	ST
	QB0:=SHL(IN:=QB0, N:=2);
若 SHL 被执行，则结果如下：	
QB0 初值：　B#2#10111110 执行一次 SHL 指令后 QB0 的值：　B#2#11111000	

4.5.2 ROL（循环左移）

➢ 指令及其操作数说明

表 4-62 ROL 指令

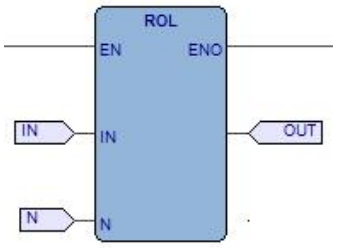
LD	ST	功能说明
	OUT:=ROL(IN:=_variant_in_, N:=_uint_in_);	循环左移

表 4-63 参数的数据类型

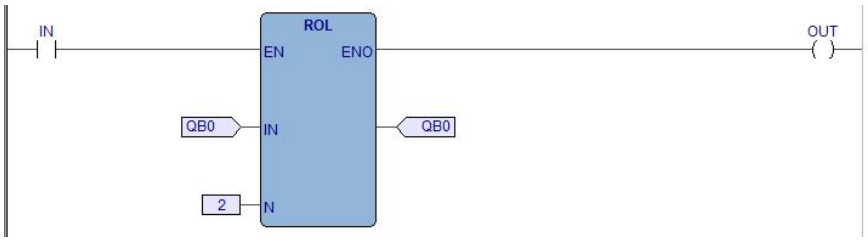
参数	输入/输出	数据类型
IN	输入	BYTE、WORD、DWORD
N	输入	BYTE
OUT	输出	BYTE、WORD、DWORD

参数 IN、OUT 的数据类型必须一致。

若 EN 为 1，则该指令被执行：将 IN 的全部二进制位向左移动 N 位，移出的高位被依次移进低位位置，最终的结果赋给 OUT。

➢ 指令使用举例：

表 4-64 ROL 指令使用说明

LD	ST
	QB0:=ROL(IN:=QB0, N:=2);
若 ROL 被执行，则结果如下：	

QB0 初值：	B#2#10111110
执行一次 ROL 指令后 QB0 的值：	B#2#11111010

4.5.3 SHR（右移）

- 指令及其操作数说明

表 4-65　SHR 指令

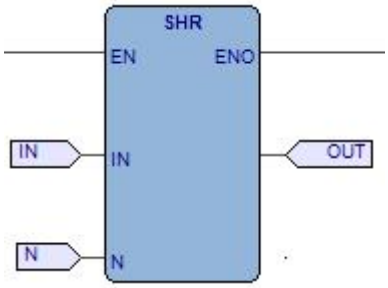
LD	ST	功能说明
	OUT:=SHR(IN:=_variant_in_, N:=_uint_in_);	右移

表 4-66 参数的数据类型

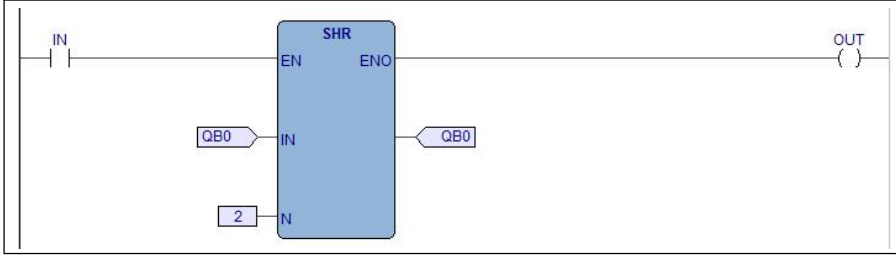
参数	输入/输出	数据类型
IN	输入	BYTE、WORD、DWORD
N	输入	BYTE
OUT	输出	BYTE、WORD、DWORD

参数 IN、OUT 的数据类型必须一致。

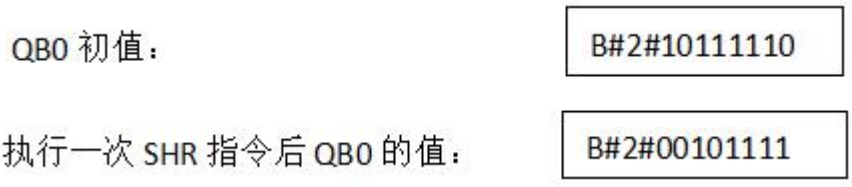
若 EN 为 1，则该指令被执行：将 IN 的全部二进制位向右移动 N 位，移出的低位被舍弃并且高位补 0，最终的结果赋给 OUT。

- 指令使用举例：

表 4-67　SHR 指令使用举例

LD	ST
	QB0:=SHR(IN:=QB0, N:=2);

若 SHR 被执行，则结果如下：



4.5.4 ROR（循环右移）

➢ 指令及其操作数说明

表 4-68 ROR 指令

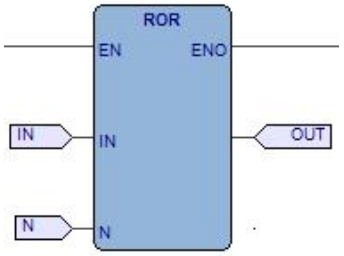
LD	ST	功能说明
	OUT:=ROR(IN:=_variant_in_, N:=_uint_in_);	循环右移

表 4-69 参数的数据类型

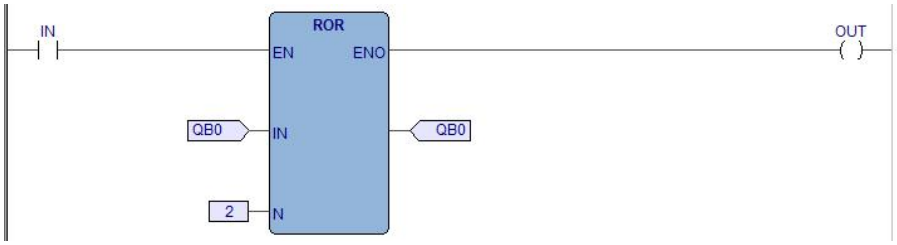
参数	输入/输出	数据类型
IN	输入	BYTE、WORD、DWORD
N	输入	BYTE
OUT	输出	BYTE、WORD、DWORD

参数 IN、OUT 的数据类型必须一致。

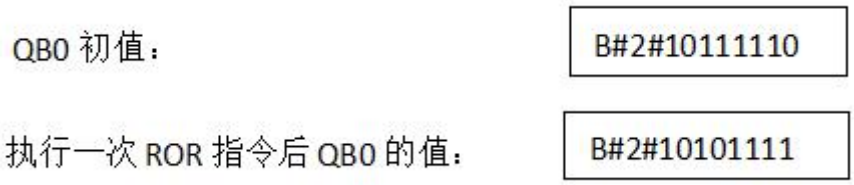
若 EN 为 1，则该指令被执行：将 IN 的全部二进制位向右移动 N 位，移出的低位被依次移进高位位置，最终的结果赋给 OUT。

➢ 指令使用举例：

表 4-70 ROR 指令使用举例

LD	ST
	QB0:=ROR(IN:=QB0, N:=2);

若 ROR 被执行，则结果如下：



4.6 赋值指令

4.6.1 MOVE（赋值）

- 指令及其操作数说明

表 4-71 MOVE 指令

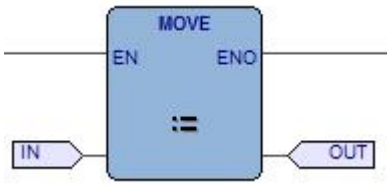
LD	ST	功能说明
	OUT:=IN;	赋值

表 4-72 参数的数据类型

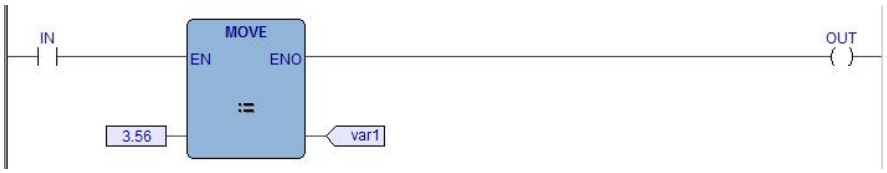
参数	输入/输出	数据类型
IN	输入	BYTE、WORD、DWORD、INT、DINT、REAL
OUT	输出	BYTE、WORD、DWORD、INT、DINT、REAL

该指令执行赋值操作，其参数 IN 与 OUT 的数据类型必须一致。

如果 EN 为 1，则该指令将输入变量 IN 的值赋给输出变量 OUT。

- 指令使用举例：

表 4-73 赋值指令使用举例

LD	ST
	var1:=3.56;

4.7 数学运算

4.7.1 ADD（加法）、SUB（减法）

➢ 指令及其操作数说明

表 4-74　ADD 和 SUB 指令

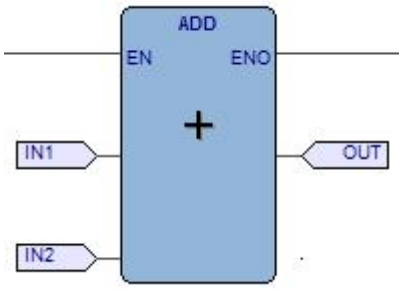
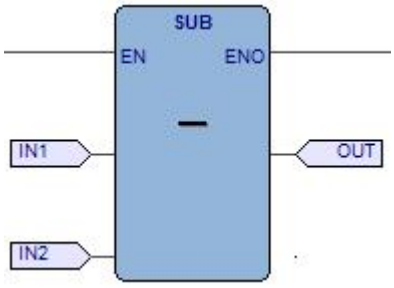
LD	ST	功能说明
	OUT := IN1 + IN2;	加法
	OUT := IN1 - IN2;	减法

表 4-75 参数的数据类型

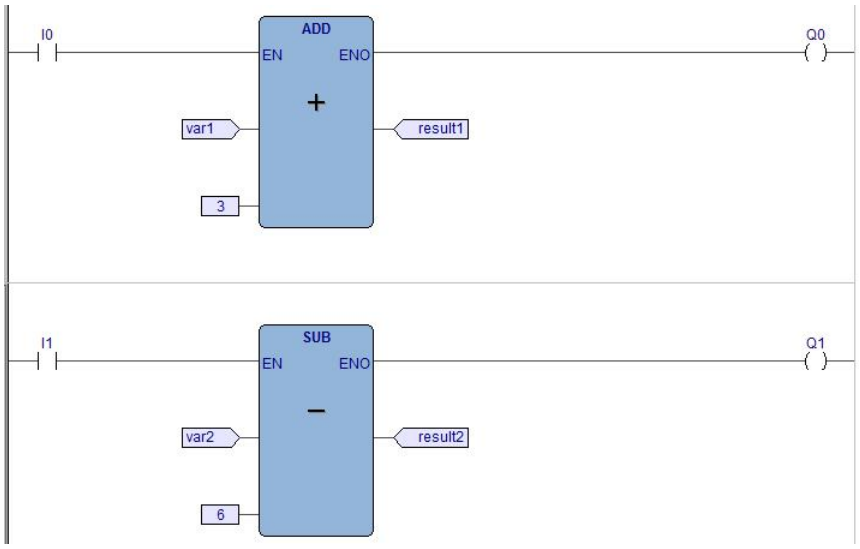
参数	输入/输出	数据类型
IN1	输入	INT、DINT、REAL
IN2	输入	INT、DINT、REAL
OUT	输出	INT、DINT、REAL

参数 IN1、IN2、OUT 的数据类型必须一致。

若 EN 值为 1，则指令被执行。其中，ADD 指令的功能是：OUT＝IN1＋IN2；SUB 指令的功能是：OUT＝IN1－IN2。

➢ 指令使用举例：

表 4-76　ADD 指令使用举例

LD	ST
	result1 := var1+3; result2 := var2-6;

4.7.2 MUL（乘法）、DIV（除法）

- 指令及其操作数说明

表 4-77　MUL 和 DIV 指令

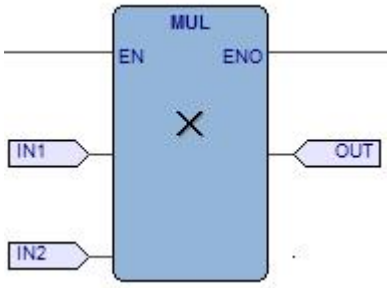
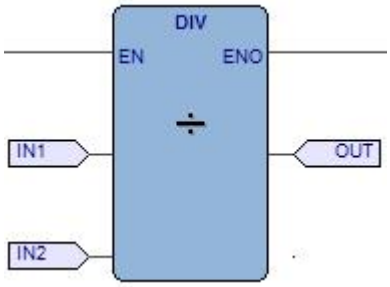
LD	ST	功能说明
	OUT := IN1 * IN2;	乘以
	OUT := IN1 / IN2;	除以

表 4-78 参数的数据类型

参数	输入/输出	数据类型
IN1	输入	INT、DINT、REAL
IN2	输入	INT、DINT、REAL
OUT	输出	INT、DINT、REAL

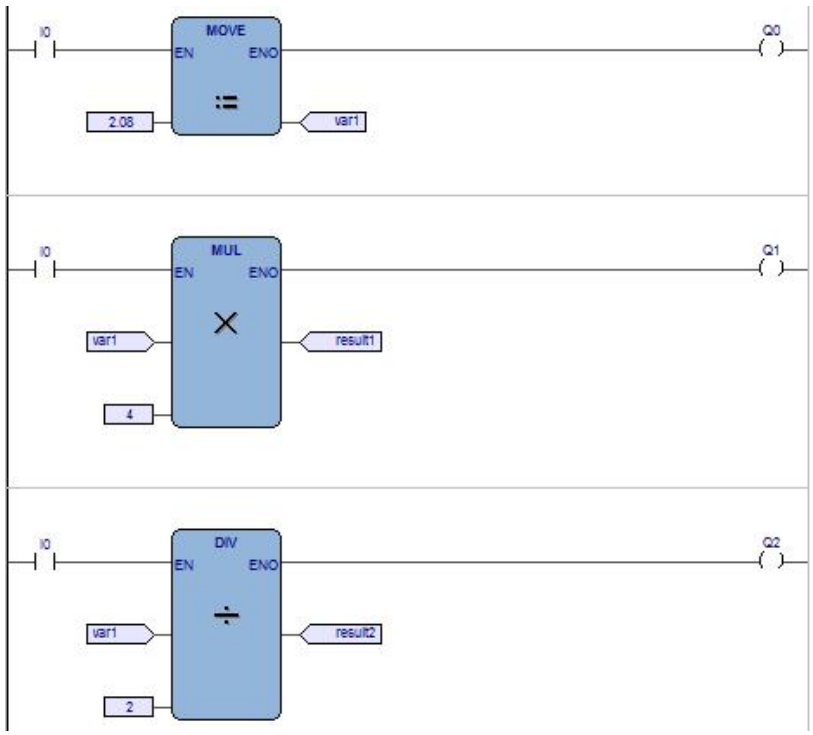
参数 IN1、IN2、OUT 的数据类型必须一致。

若 EN 值为 1，则指令被执行。其中，MUL 指令的功能是：OUT＝IN1×IN2；DIV 指令的功能是：OUT＝IN1÷IN2。

- 指令使用举例：

表 4-79 MUL、DIV 指令使用举例

LD	ST

	var1 := 2.08;
	result1 := var1 * 4;
	result2 := var1 / 2;

4.7.3 MOD（求余数）

- 指令及其操作数说明

表 4-80 MOD 指令

LD	ST	功能说明

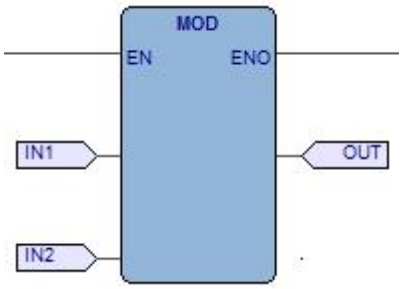
	OUT := IN1 MOD IN2;	取余

表 4-81 参数的数据类型

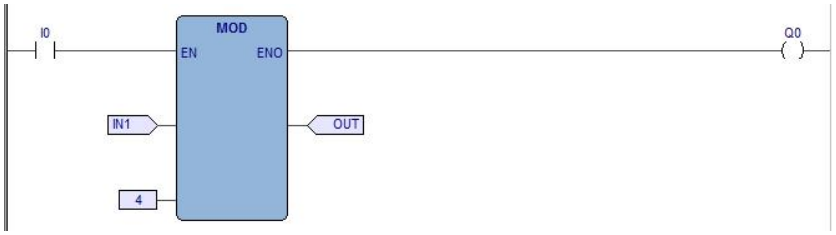
参数	输入/输出	数据类型
IN1	输入	INT、DINT
IN2	输入	INT、DINT
OUT	输出	INT、DINT

参数 IN1、IN2、OUT 的数据类型必须一致。

若 EN 值为 1，则该指令被执行：将 IN1 除以 IN2，并将余数赋给 OUT。

- 指令使用举例：

表 4-82 MOD 指令使用举例

LD	ST
	OUT:=IN1 MOD 4;
举例结果如下：	
IN1 值： 9 执行 MOD 指令后 OUT 的值： 1	

4.7.4 ABS（绝对值）

- 指令及其操作数说明

表 4-83　ABS 指令

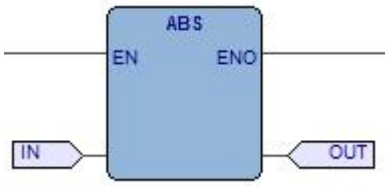
LD	ST	功能说明
	OUT := ABS(IN);	计算绝对值

表 4-84 参数的数据类型

参数	输入/输出	数据类型
IN	输入	INT、DINT、REAL
OUT	输出	INT、DINT、REAL

参数 IN、OUT 的数据类型必须一致。

该指令将输入 IN 求绝对值并将结果赋给 OUT，如下式所示：OUT＝|IN|。

若 EN 值为 1，则该指令被执行。

4.7.5 SQRT（平方根）

- 指令及其操作数说明

表 4-85 SQRT 指令

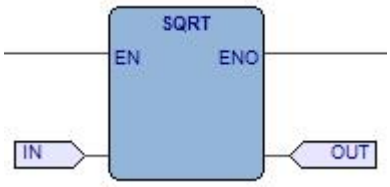
LD	ST	功能说明
	OUT := SQRT(IN);	计算平方根

表 4-86 参数的数据类型

参数	输入/输出	数据类型
IN	输入	REAL
OUT	输出	REAL

该指令将输入 IN 开平方并将结果赋给 OUT，如下式所示：OUT＝$\sqrt{IN}$ 。

若 EN 值为 1，则该指令被执行。

4.7.6 LN（自然对数）、LOG（常用对数）

➢ 指令及其操作数说明

表 4-87 LN 和 LOG 指令

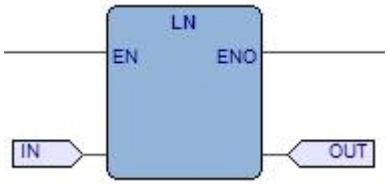
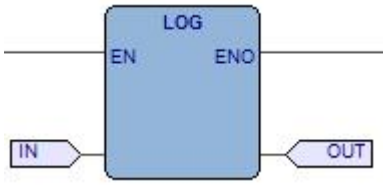
LD	ST	功能说明
	OUT := LN(IN);	自然对数运算
	OUT := LOG(IN);	常用对数运算

表 4-88 参数的数据类型

参数	输入/输出	数据类型
IN	输入	REAL
OUT	输出	REAL

LN 指令将输入 IN 求自然对数并将结果赋给 OUT，如下式所示：$OUT = \log_e(IN)$。

LOG 指令将输入 IN 求常用对数并将结果赋给 OUT，如下式所示：$OUT = \log_{10}(IN)$。

若 EN 值为 1，则指令被执行。

4.7.7 EXP（以 e 为底的指数）

➢ 指令及其操作数说明

表 4-87 EXP 指令

LD	ST	功能说明

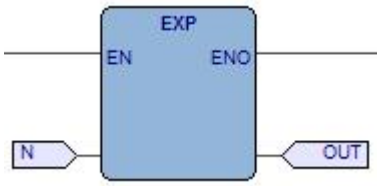
	OUT := EXP(IN);	自然指数运算

表 4-88 参数的数据类型

参数	输入/输出	数据类型
IN	输入	REAL
OUT	输出	REAL

该指令将输入 IN 求以 e 为底的指数并将结果赋给 OUT，如下式所示：OUT＝e^{IN}。

若 EN 值为 1，则该指令被执行。

4.7.8 SIN（正弦）、COS（余弦）、TAN（正切）

- 指令及其操作数说明

表 4-89 SIN、COS 和 TAN 指令

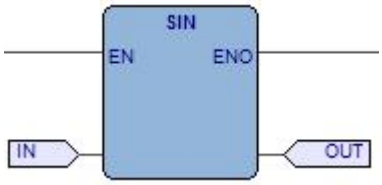
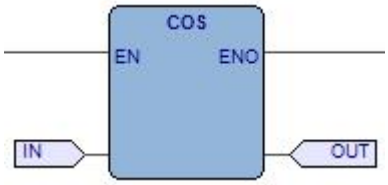
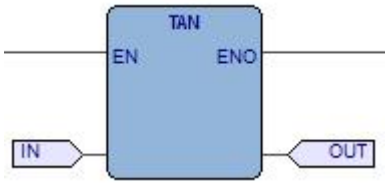
LD	ST	功能说明
	OUT := SIN(IN);	正弦运算
	OUT := COS(IN);	余弦运算
	OUT := TAN(IN);	正切运算

表 4-90 参数的数据类型

参数	输入/输出	数据类型
IN	输入	REAL

OUT	输出	REAL

输入 IN 表示的是弧度值（将角度从度数转换为弧度需要乘以π/180）。

SIN 指令将 IN 求正弦并将结果赋给 OUT，如下式所示：OUT＝SIN(IN)。

COS 指令将 IN 求余弦并将结果赋给 OUT，如下式所示：OUT＝COS(IN)。

TAN 指令将 IN 求正切并将结果赋给 OUT，如下式所示：OUT＝TAN(IN)。

若 EN 值为 1，则指令被执行。

4.7.9 ASIN（反正弦）、ACOS（反余弦）、ATAN（反正切）

- 指令及其操作数说明

表 4-91　ASIN、ACOS 和 ATAN 指令

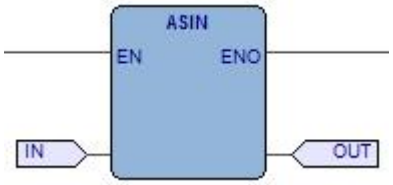
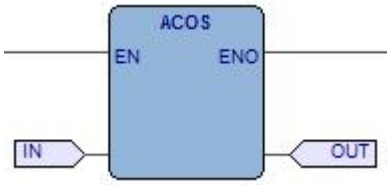
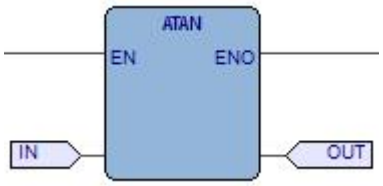
LD	ST	功能说明
	OUT := ASIN(IN);	反正弦运算
	OUT := ACOS(IN);	反余弦运算
	OUT := ATAN(IN);	反正切运算

表 4-92 参数的数据类型

参数	输入/输出	数据类型
IN	输入	REAL
OUT	输出	REAL

ASIN 指令将 IN 求反正弦并将结果赋给 OUT。

ACOS 指令将 IN 求余弦并将结果赋给 OUT。

TAN 指令将 IN 求正切并将结果赋给 OUT。

注意：结果是弧度值。

若 EN 值为 1，则指令被执行。

4.8 程序控制

4.8.1 标号及跳转指令

➢ 指令及其操作数说明

表 4-93　跳转指令

LD	ST	功能说明
	JMP lbl;	跳转

表 4-94 参数的数据类型

参数	描 述
lbl	合法的标识符

注意：跳转指令中指定的 *lbl* 标号必须存在而且必须与该指令在同一程序中。

LBL 指令用于在当前位置定义一个标号，该标号将作为跳转指令的目的地。标号不允许重复定义。LBL 指令是无条件执行的，不建议用户在 LBL 指令的左端加入任何元件。实际上，在编译时编译器会将 LBL 指令的左端的所有元件忽略掉。

JMP 指令用于无条件地将程序跳转到 lbl 标号处继续执行。

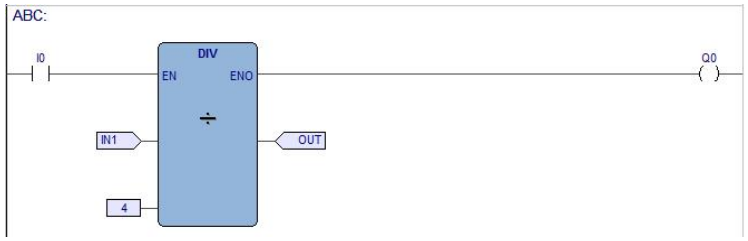
JMPC 指令的作用是：若指令左侧能量流的值为 1，则将程序跳转到 lbl 标号处继续执行，否则该指令不起作用，程序继续向下依次执行。

JMPCN 指令的作用是：若指令左侧能量流的值为 0，则将程序跳转到 lbl 标号处继续执行，否则该指令不起作用，程序继续向下依次执行。

➢ 指令使用举例：

表 4-95 跳转指令使用举例

LD	ST



…

…

…

ABC：

…

…

…

JMP ABC;

4.8.2 返回指令

➢ 指令及其操作数说明

表 4-96 返回指令

LD	ST	功能说明
	RETURN;	返回

返回指令只能在子程序和中断服务程序中使用，用于终止所在程序并返回到该程序的调用点继续执行。

在每个子程序和中断服务程序的结尾，EURA studio 软件都自动地隐含调用了 RETC 指令。

RETC 指令：若指令左侧能量流的值为 1，则该指令被执行，否则该指令不起作用，程序继续向下依次执行。

RETCN 指令：若指令左侧能量流的值为 0，则该指令被执行，否则该指令不起作用，程序继续向下依次执行。

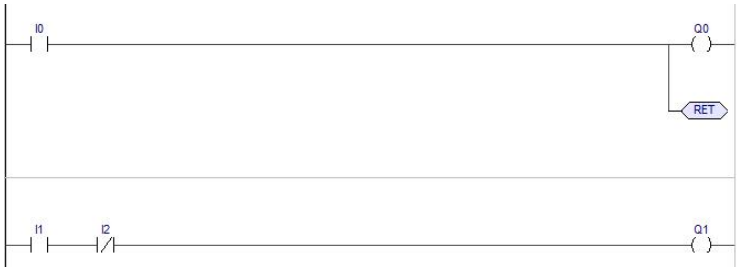
➢ 指令使用举例：

表 4-97 返回指令使用举例

LD	ST

MAIN：

SBR_0：



MAIN：

```
IF IN1 THEN
   SBR_0;
END_IF;
IF IN2 THEN
   SBR_1;
END_IF;
```

SBR_0：

```
IF I0 THEN
  RETURN;
END_IF;
IF I1 AND I2=0 THEN
  Q1:=1
ELSE
  Q1:=0;
END_IF;
```

第五章 通讯与应用

5.1 通信的基础知识

5.1.1 通讯的基本概念

(1) 串行通信和并行通讯

串行通讯和并行通讯是两种不同的数据传输方式。

串行通讯就是通过一对导线将发送方与接收方进行连接，传输数据的每个二进制位按照规定顺序在同一导线上依次发送与接收，如图 5.1 所示。例如，常用的优盘 USB 接口就是串行通讯接口。串行通讯的特点是通信控制复杂，通信电缆少，因此与并行通信相比，成本低。

并行通讯就是将一个 8 位数据（或 16 位、32 位）的每一个二进制位采用单独的导线进行传输，并将传送方和接收方进行并行连接，一个数据的各二进制位可以在同一时间内一次传输，如图 5.2 所示。例如老式打印机的打印口和计算机的通信就是并行通信。并行通信的特点是一个周期里可以一次传输多位数据，其连线的电缆多，因此长距离传送时成本高。

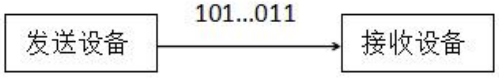
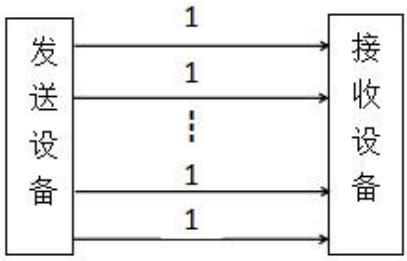


图 5.1 串行通信



5.2 并行通信

(2) 异步通讯与同步通讯

异步通讯与同步通信也称为异步传送与同步传送，这是串行通信的两种基本信息传送方式。从用户的角度上说，两者最主要的区别在于通讯方式的“帧”不同。

异步通信方式又称起止方式。它在发送字符时，要先发送起始位，然后是字符本身，最后是停止位，字符之后还可以加入奇偶校验位。异步通讯方式具有硬件简单、成本低的特点，主要用于传输速率低于 19.2Kbit/s 以下的数据通信。

同步通信方式在传递速度的同时，也传输时钟同步信号，并始终按照给定的时刻采集数据。其传输数据的效率高，硬件复杂，成本高，一般用于传输速率高于 20Kbit/s 以上的数据通信。

(3) 单工、全双工和半双工

单工、全双工与半双工是通信中描述数据传送方向的专用术语。

① 单工：指数据只能实现单向传送的通信方式，一般用于数据到输出，不可以进行数据交换。

② 全双工：也称双工，指数据可以进行双向数据传送，同一时刻既能发送数据，也能

接收数据。通常需要两对绞线连接，通信线路成本高。例如，RS-422 就是“全双工”通信方式。

③ 半双工：指数据可以进行双向数据传输，同一时刻，只能发送数据或者接收数据。通常需要一对双绞线连接，与全双工相比，通信线路成本低。例如，RS-485 只用一对双绞线时就是“半双工”通信方式。

5.1.2 Mosbus RTU 通讯协议

Modbus RTU 是一种主从通讯协议。在同一时刻，只有一个主节点连接于总线，一个或多个子节点 (最大编号为 247) 连接于同一个串行总线。Modbus 通信总是由主节点发起。子节点在没有收到来自主节点的请求时，从不会发送数据。子节点之间从不会互相通信。主节点在同一时刻只会发起一个 Modbus 事务处理。

Modbus RTU 通讯协议广泛用于 PLC 、控制器、触摸屏等设备间的数据交互。Modbus RTU 常用的物理接口为串口。EC400 系列 PLC 具有 1 个 RS232 接口和 2 个 RS485 接口，均可以独立的作为 Modbus 主站/从站使用。其技术参数如下表所示：

表 5-1 Modbus RTU 计数参数

Modbus 主站/从站	
物理接口	RS232/RS485
通讯格式	RTU
波特率	600/1200/2400/4800/9600/19200/38400/57600/115200
校验位	无校验/奇校验/偶校验
数据位	8 位
停止位	1 位/2 位
支持的 Modbus 功能码	03/06/16

RS-232 标准接口（又称 EIA RS-232）是常用的串行通信接口标准之一，要求通讯双方都采用一个标准接口，使不同的设备可以方便地连接起来进行通讯。RS-232 通信方式允许简单连接三线：Tx、Rx 和地线。但是对于数据传输，双方必须对数据定时采用使用相同的波特率。RS-232 接口主要有以下特点：

- 接口的信号电平值较高，易损坏接口电路的芯片，又因为与 TTL 电平不兼容故需使用电平转换电路方能与 TTL 电路连接。
- 传输速率较低，在异步传输时，波特率为 20Kbps。现在由于采用新的 UART 芯片 16C550 等，波特率达到 115.2Kbps。
- 接口使用一根信号线和一根信号返回线而构成共地的传输形式， 这种共地传输容易产生共模干扰，所以抗噪声干扰性弱。
- 传输距离有限，最大传输距离标准值为 50 米，实际上也只能用在 15 米左右。

RS-485 采用平衡发送和差分接收方式实现通信：发送端将串行口的 TTL 电平信号转换成

差分信号 A、B 两路输出，经过线缆传输之后在接收端将差分信号还原成 TTL 电平信号。由于传输线通常使用双绞线，又是差分传输，所以又极强的抗共模干扰的能力，总线收发器灵敏度很高，可以检测到低至 200mV 电压。故传输信号在千米之外都是可以恢复。

EC400 系列 PLCmodbus 通信方式主要分为两种：配置模式与功能块模式，两者模式都需要对串口通信进行配置，区别主要在于：

1) 触发机制，配置模式主要配置在内部 Background 任务中，通过配置设置通信地址，系统自行进行数据交互。功能块模式需要用户编写相应的读写功能块，并将所编写的程序配置到 Background 任务中，其何时触发需要用户根据逻辑对功能块进行触发。

2) 当设置为主站时，对 modbus 通信中掉电保持区写操作差异，配置模式下只有掉电保持区数据发生变化时才进行写操作；功能块模式是根据用户触发情况进行写操作的。

5.2 串口通信配置

5.2.1 Modbus RTU 主站配置

配置 Modbus 主站，并建立与从站通讯的通道的实现步骤如下所示：

1. 设置串口通讯格式：点击左侧资源配置树上的端口设备 RS232 串口或 RS485 串口，在右侧的 Modbus 配置页面中配置串口的基本参数，使其与通讯的从站相对应。

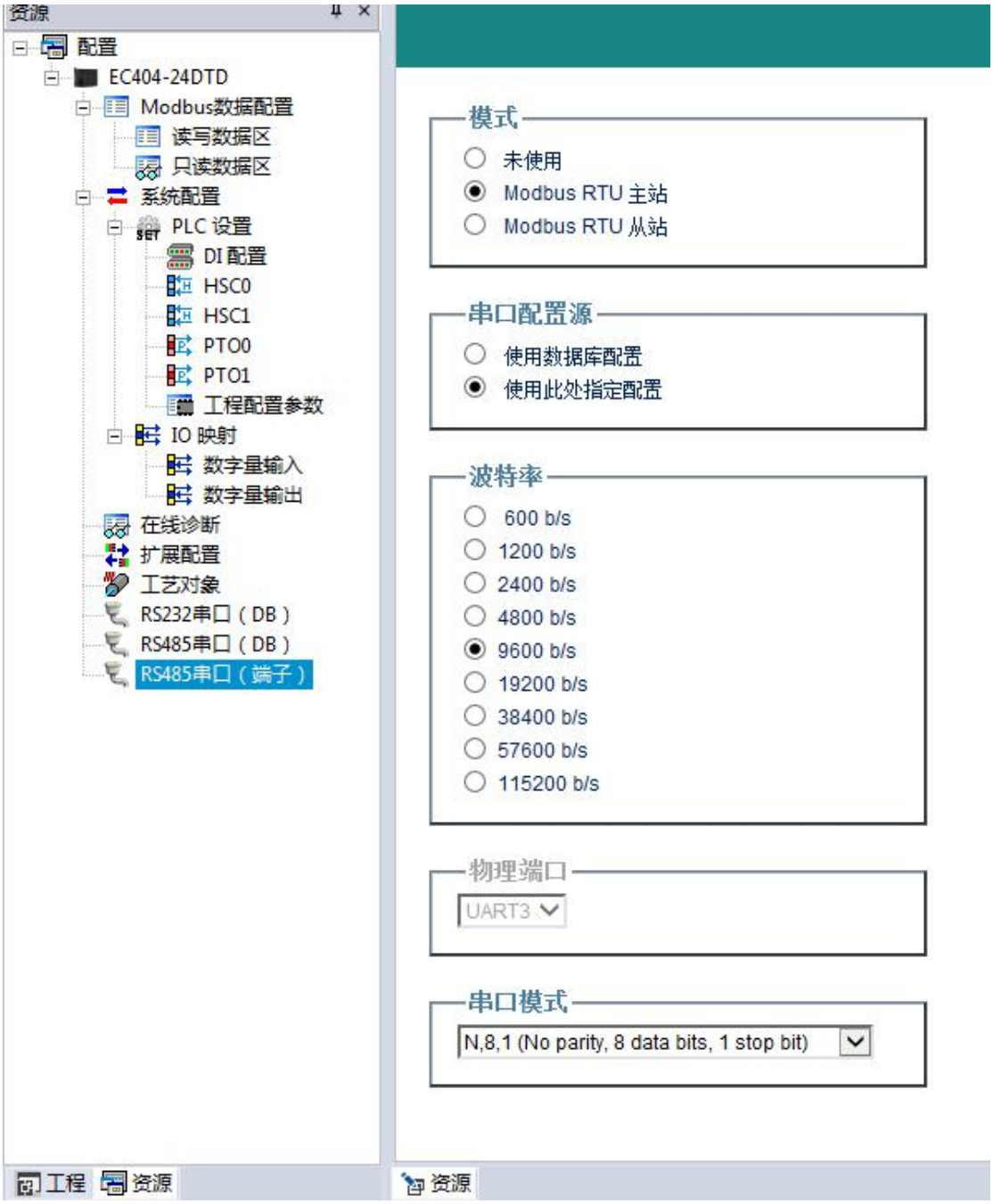


图 5.3 配置串口参数

2. 创建从站设备：点击菜单栏中的“工具”——“运行 Modbus 自定义编辑器”。

图 5.4 开启 Modbus 自定义编辑器

在弹出的“Modbus 自定义编辑器”对话框中，编辑从站名称以及添加从站变量，配置

设备信息，根据用户需求选择是否开启写入单个线圈和单个寄存器功能码等功能，编辑完成后点击“保存”按钮，保存为“.pct”文件，保存路径不能修改。

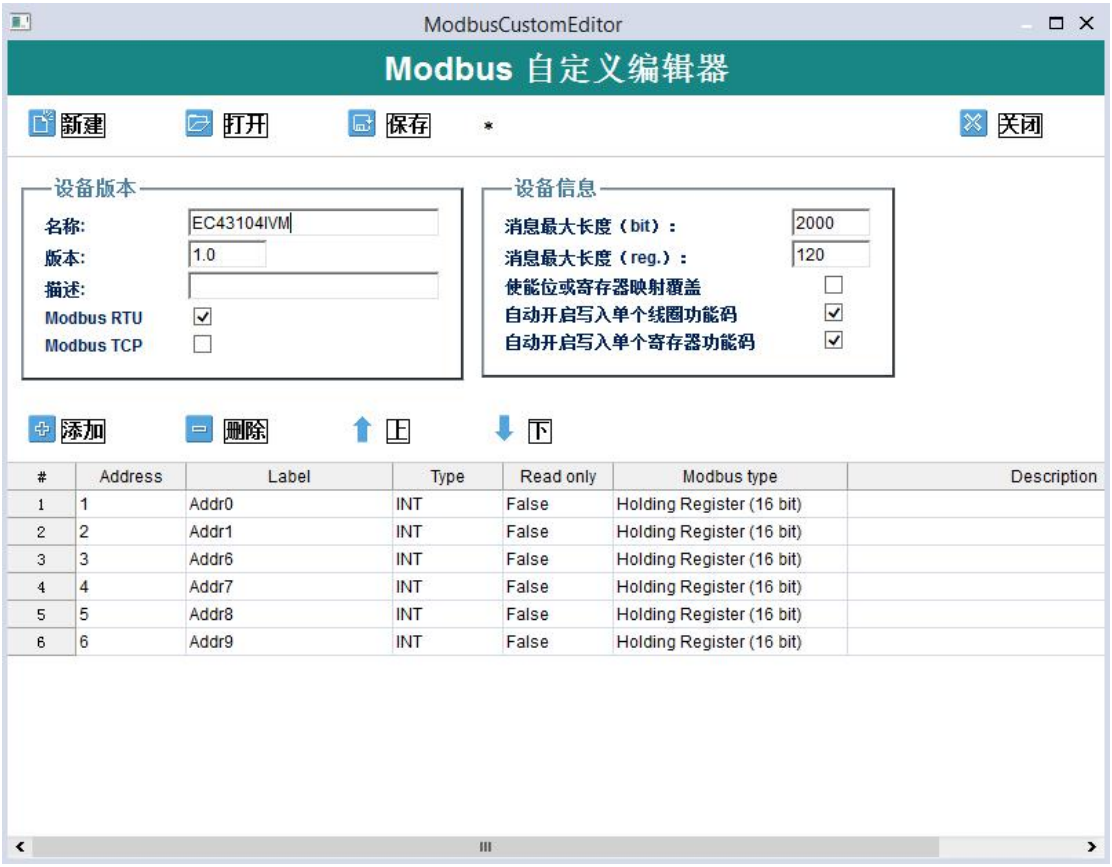


图 5.5 编辑从站设备

3. 添加从站设备：在资源配置树的相应串口节点右键，点击“添加”菜单，在弹出的的“设备目录”对话框中选择从站设备文件，点击“选择”按钮。

图 5.6 添加从站设备

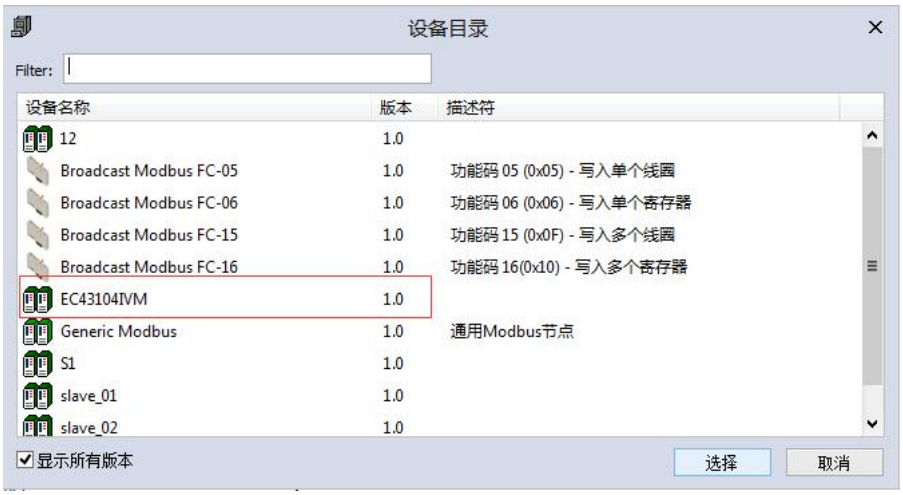


图 5.7 选择从站设备文件

图 5.8 从站设备节点

4. 配置从站设备：双击从站设备，弹出从站配置对话框，在“通用”选项卡中设置从站地址、超时时间和最小轮询时间。

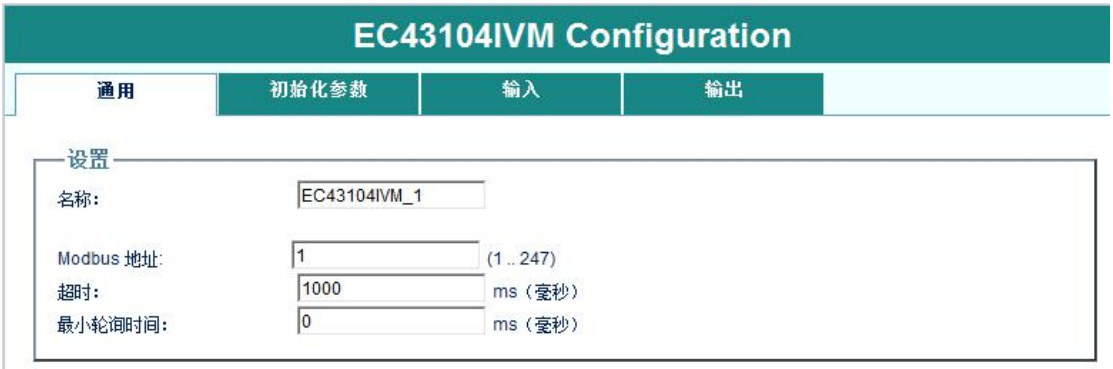


图 5.9 配置从站地址

5. 建立从站通道：点击“输入”选项卡界面中的“添加”按钮，弹出“Modbus 参数”对话框，选择参数，点击“确认”。

图 5.10 添加 Modbus 参数

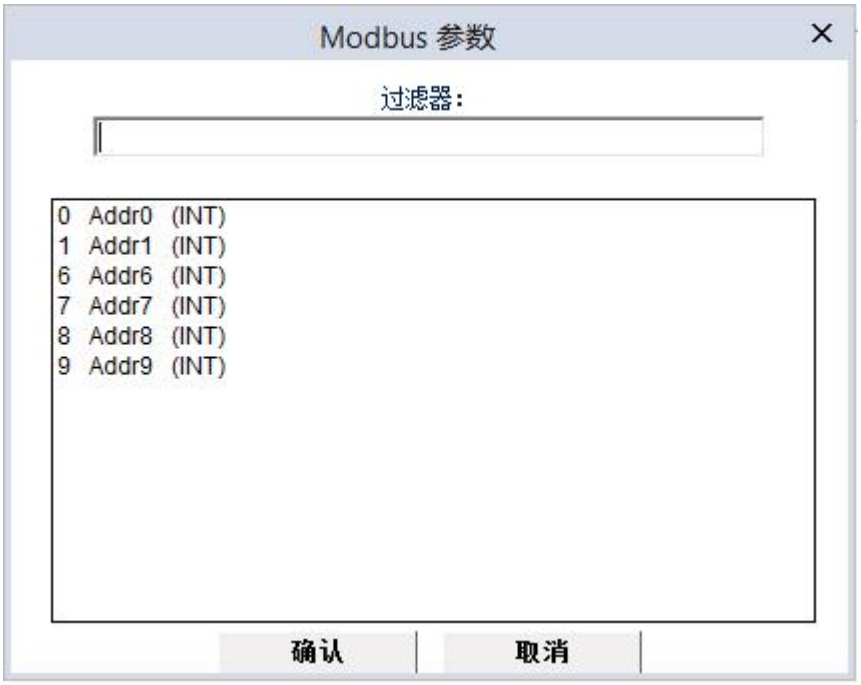


图 5.11 选择 Modbus 参数

在 Variable 列为每个参数变量映射 PLC 变量。

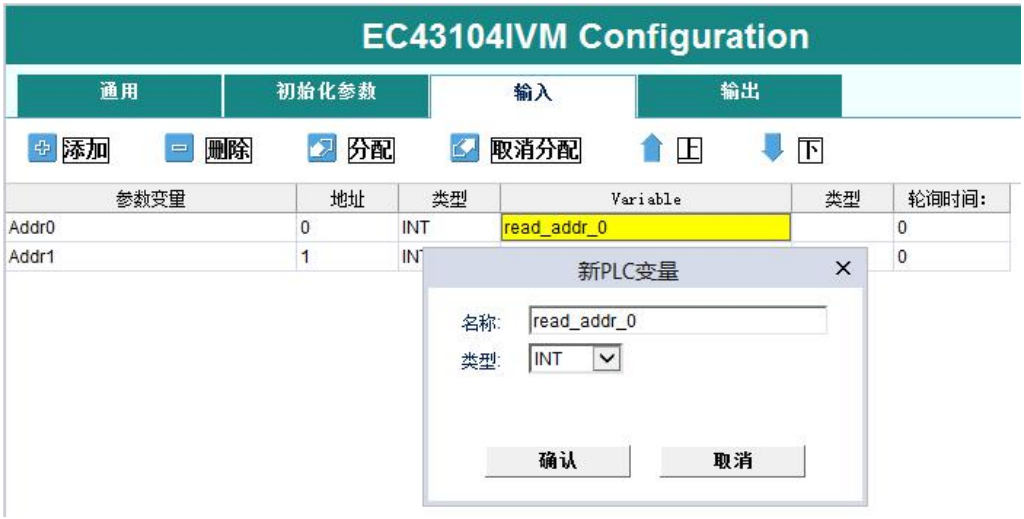


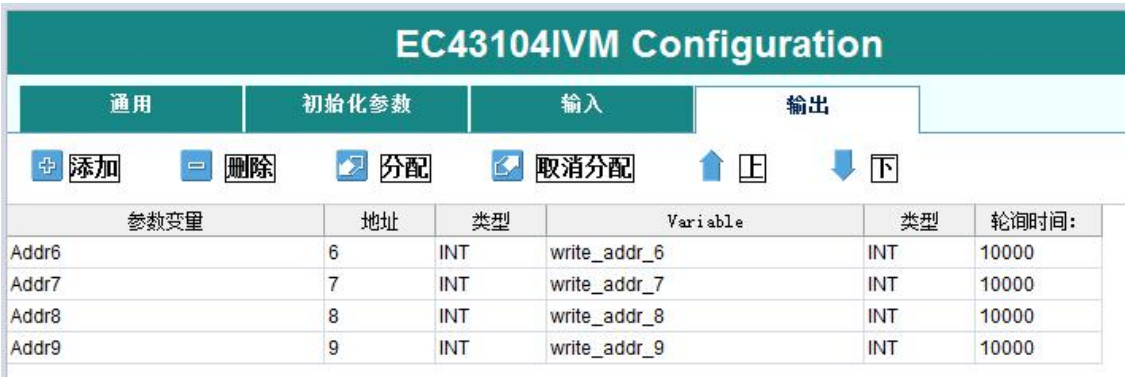
图 5.12 映射 PLC 变量

在“轮询时间”列，设置通讯轮询时间。

参数变量	地址	类型	Variable	类型	轮询时间:
Addr0	0	INT	read_addr_0	INT	1000
Addr1	1	INT	read_addr_1	INT	1000

图 5.13 设置轮询时间

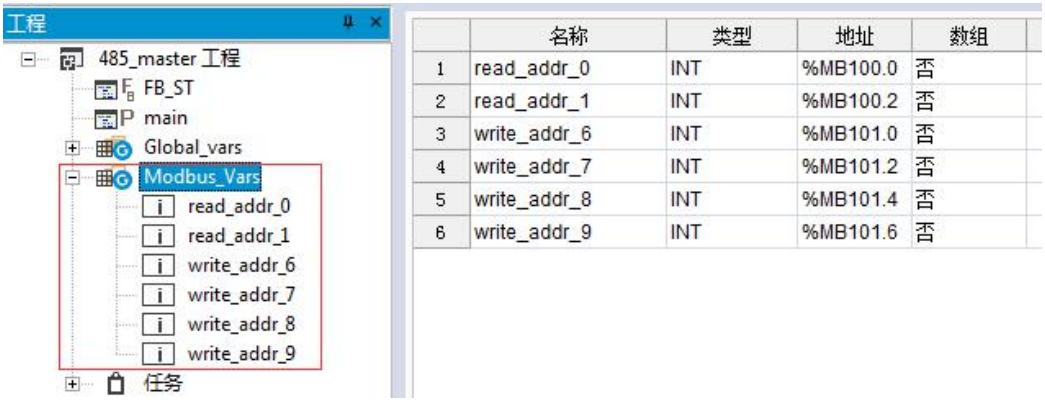
用同样的方法配置“输出”选项卡界面。



参数变量	地址	类型	Variable	类型	轮询时间:
Addr6	6	INT	write_addr_6	INT	10000
Addr7	7	INT	write_addr_7	INT	10000
Addr8	8	INT	write_addr_8	INT	10000
Addr9	9	INT	write_addr_9	INT	10000

图 5.14 配置输出界面

映射完成后，点击保存，在工程树会自动添加 Modbus_Vars 节点。



	名称	类型	地址	数组
1	read_addr_0	INT	%MB100.0	否
2	read_addr_1	INT	%MB100.2	否
3	write_addr_6	INT	%MB101.0	否
4	write_addr_7	INT	%MB101.2	否
5	write_addr_8	INT	%MB101.4	否
6	write_addr_9	INT	%MB101.6	否

图 5.15 映射完成后工程树界面

5.2.2 Modbus RTU 从站配置

Modbus RTU 从站中的数据交互区分为读写数据区和只读数据区两类。读写数据区中的数值既可以被主站读取，也可以被主站改写，其地址从 0x4000 开始。只读数据区只能被主站读取，不能被主站改写，其地址从 0x6000 开始。

从站的实现步骤如下所示：

1. 设置串口通讯格式：点击左侧资源配置树上的端口设备 RS232 串口或 RS485 串口，在右侧的 Modbus 配置页面中配置串口的基本参数，使其与通讯的主站相对应。

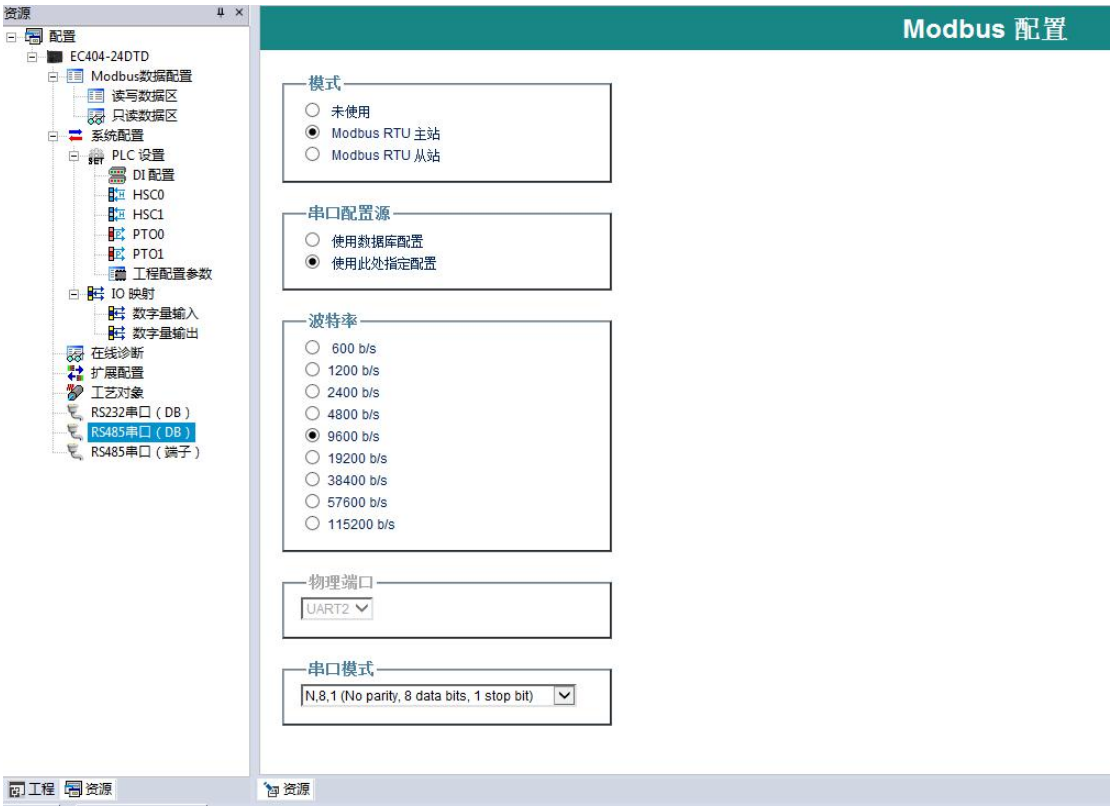


图 5.16 配置串口参数

2. 定义 Modbus 通讯数据变量：在读写数据区或只读数据区添加变量。

图 5.17 配置串口参数

5.3 串口通信实例

5.3.1 PLC 作为主站与变频器通信

1. 创建工程

图 5.18 创建工程

2. 配置串口通讯格式：点击左侧资源配置树上的端口设备 RS485 串口（DB），在右侧的 Modbus 配置页面中配置串口的基本参数，使其与通讯的从站相对应。

图 5.19 配置通讯参数

3. 创建从站设备：点击菜单栏中的“工具”——“运行 Modbus 自定义编辑器”。

图 5.20 开启 Modbus 自定义编辑器

在弹出的“Modbus 自定义编辑器”对话框中，勾选自动开启写入单个线圈功能，编辑从站名称以及添加从站 modbus 地址变量（Address、Label、Type、Read only、Modbus type），编辑完成后点击“保存”按钮，保存为“.pct”文件，保存路径不能修改。

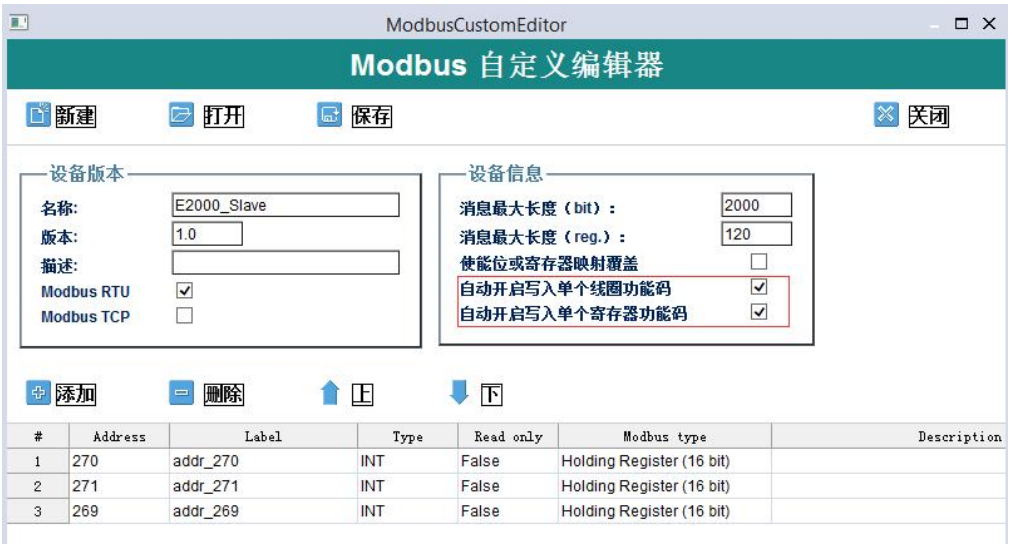


图 5.21 编辑从站设备

4. 添加从站设备：在资源配置树的相应串口节点右键，点击“添加”菜单，在弹出的的“设备目录”对话框中选择从站设备文件，点击“选择”按钮。

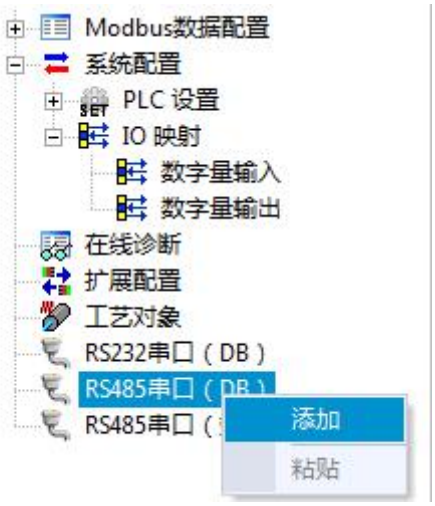


图 5.22 添加从站设备

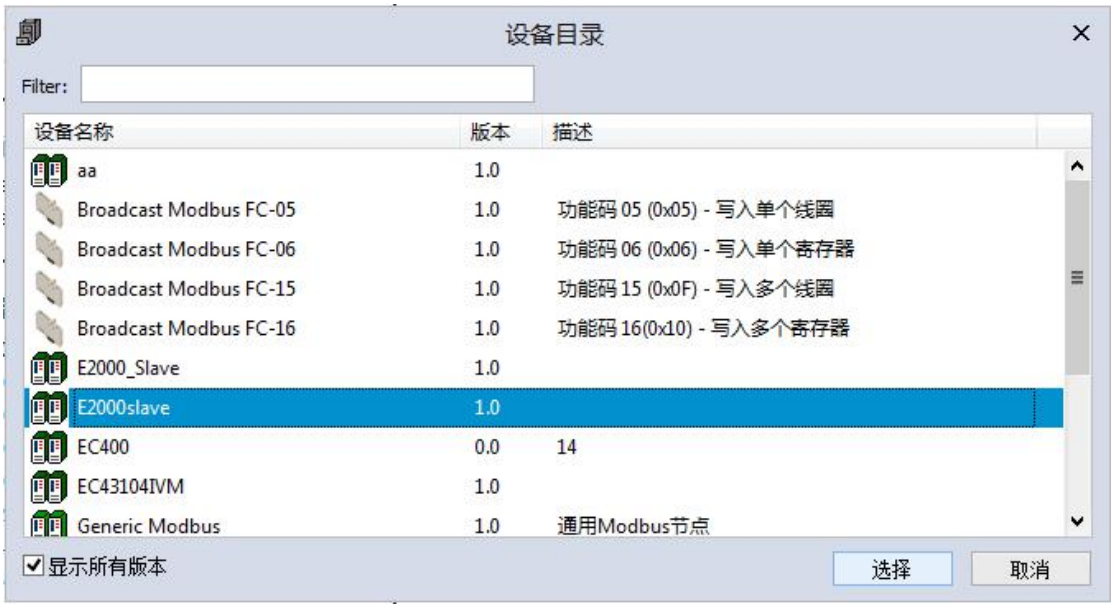


图 5.23 选择从站设备文件

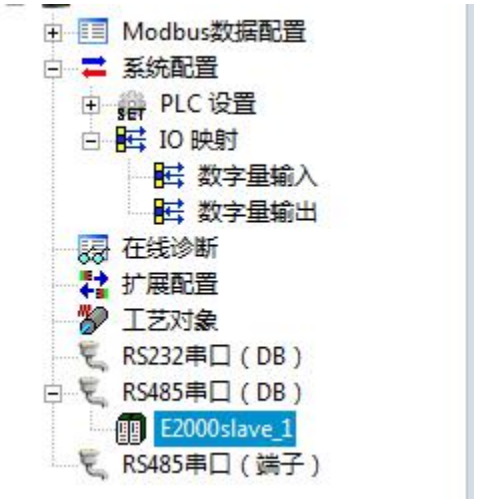


图 5.24 从站设备节点

5. 配置从站设备：双击从站设备，弹出从站配置对话框，在“通用”选项卡中设置从站地址、超时时间和最小轮询时间。

图 5.25 配置从站地址

6. 初始化参数：点击“初始化参数”选项卡界面中的“添加”按钮，弹出“Modbus 参数”对话框，用户根据需要选择需要初始化的参数，点击“确认”。

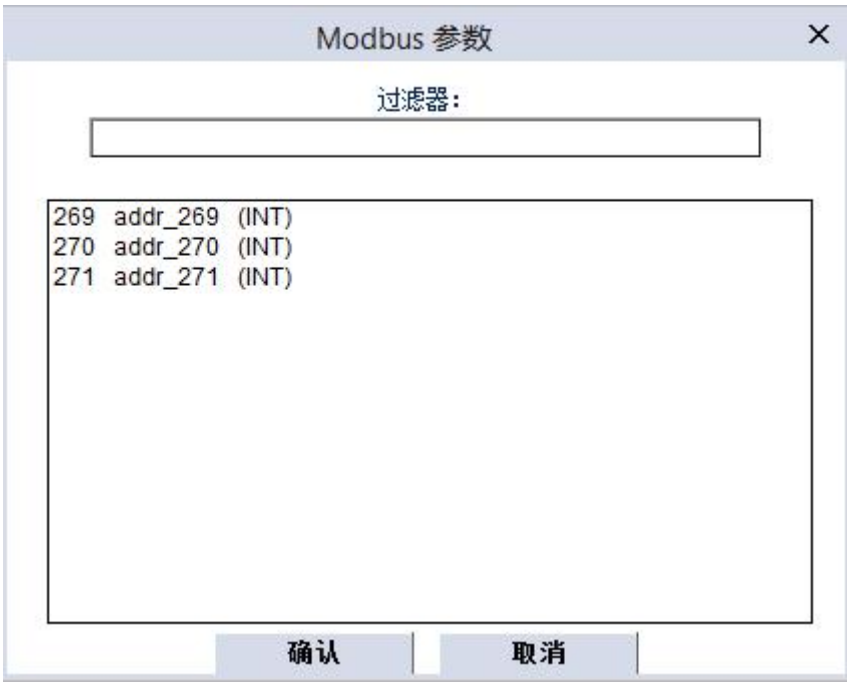


图 5.26 选择初始化参数地址

初始化参数添加完成后，修改需要写入的参数值。

图 5.27 初始化参数设定数值

7. 建立从站通道：点击“输入”选项卡界面中的“添加”按钮，弹出“Modbus 参数”对话框，选择参数，点击“确认”。

图 5.28 添加 Modbus 参数

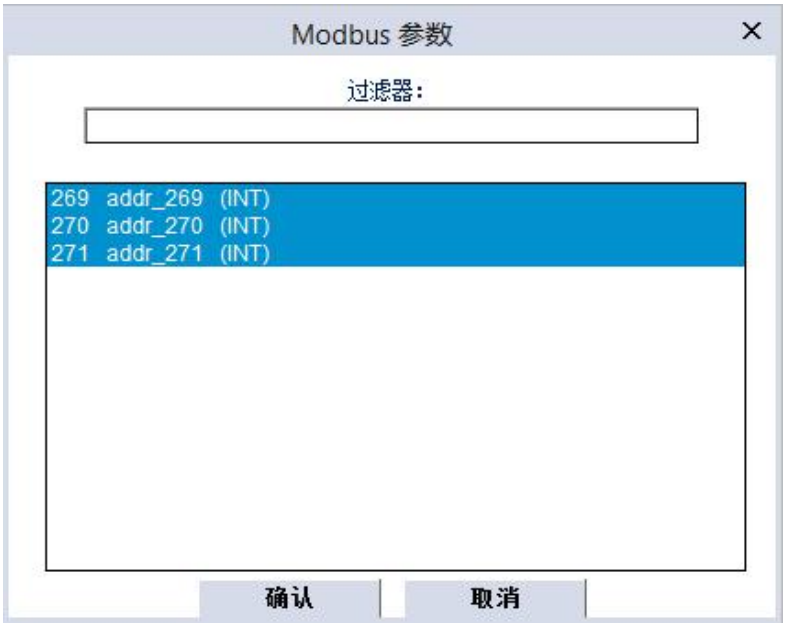


图 5.29 选择 Modbus 参数

在 Variable 列为每个参数变量映射 PLC 变量。

图 5.30 映射 PLC 变量

在“轮询时间”列，设置通讯轮询时间。

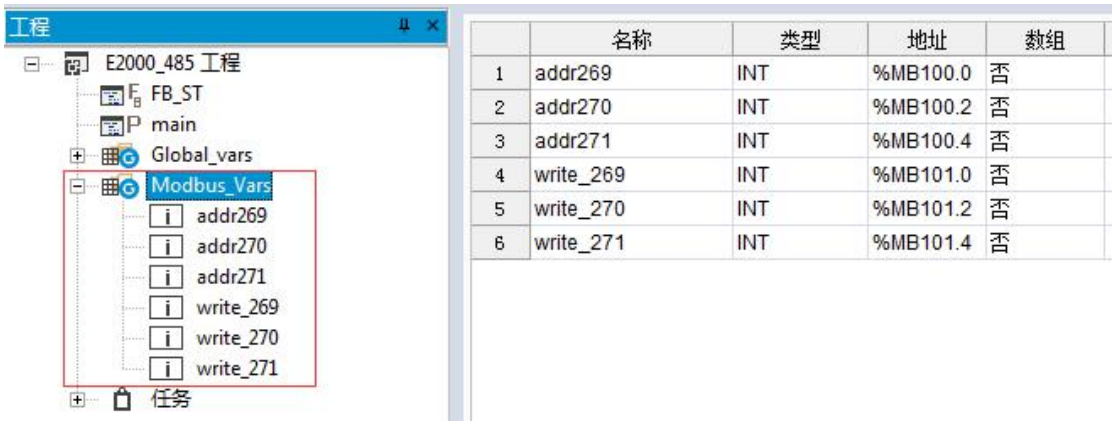
图 5.31 设置轮询时间

用同样的方法配置“输出”选项卡界面。

参数变量	地址	类型	变量	类型	轮询时间:
addr_269	269	INT	write_269	INT	10
addr_270	270	INT	write_270	INT	10
addr_271	271	INT	write_271	INT	10

图 5.32 配置输出界面

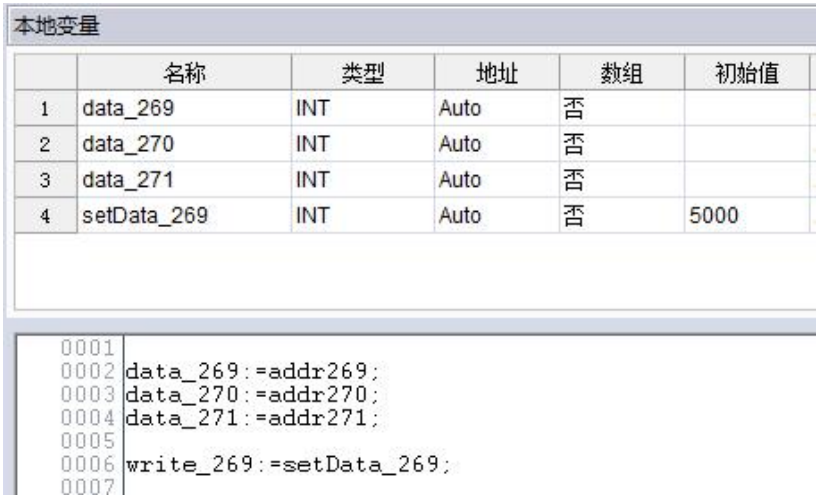
映射完成后，点击保存，在工程树会自动添加 Modbus_Vars 节点。



	名称	类型	地址	数组
1	addr269	INT	%MB100.0	否
2	addr270	INT	%MB100.2	否
3	addr271	INT	%MB100.4	否
4	write_269	INT	%MB101.0	否
5	write_270	INT	%MB101.2	否
6	write_271	INT	%MB101.4	否

图 5.33 映射完成后工程树界面

8. 程序中调用 Modbus_Vars 变量：



	名称	类型	地址	数组	初始值
1	data_269	INT	Auto	否	
2	data_270	INT	Auto	否	
3	data_271	INT	Auto	否	
4	setData_269	INT	Auto	否	5000

```
0001
0002 data_269:=addr269;
0003 data_270:=addr270;
0004 data_271:=addr271;
0005
0006 write_269:=setData_269;
0007
```

图 5.34 程序调用参数

9. 编译、下载程序、重启设备，在线调试程序。

本地变量

	名称	类型	地址	数组	初始值
1	data_269	INT	Auto	否	
2	data_270	INT	Auto	否	
3	data_271	INT	Auto	否	
4	setData_269	INT	Auto	否	5000

```
0001
0002 data_269   4800   :=addr269   4800   ;
0003 data_270    20    :=addr270    20    ;
0004 data_271    90    :=addr271    90    ;
0005
0006 write_269   4800   :=setData_269   4800   ;
0007
```

图 5.35 在线监控程序变量

注：PLC 以上述方式进行 Modbus 主站通信时，只有当写数据发生变化时，才能触发写寄存器操作。如果要实现不间断写操作时，需要调用写功能块进行发送数据。

5.3.2 PLC 作为从站与触摸屏（HMI）通信

1. 设置串口通讯格式：点击左侧资源配置树上的端口设备 RS485 串口(端子)，在右侧的 Modbus 配置页面中配置串口的基本参数，使其与通讯的主站相对应。

图 5.36 配置串口参数

2. 定义 modbus 通讯变量：添加 modbus 通讯变量，主站可以读写“读写数据区”变量，

也可以读 “只读数据区”的变量。

图 5.37 定义读写数据区变量名称

图 5.38 定义只读数据区变量名称

3. 程序中调用变量

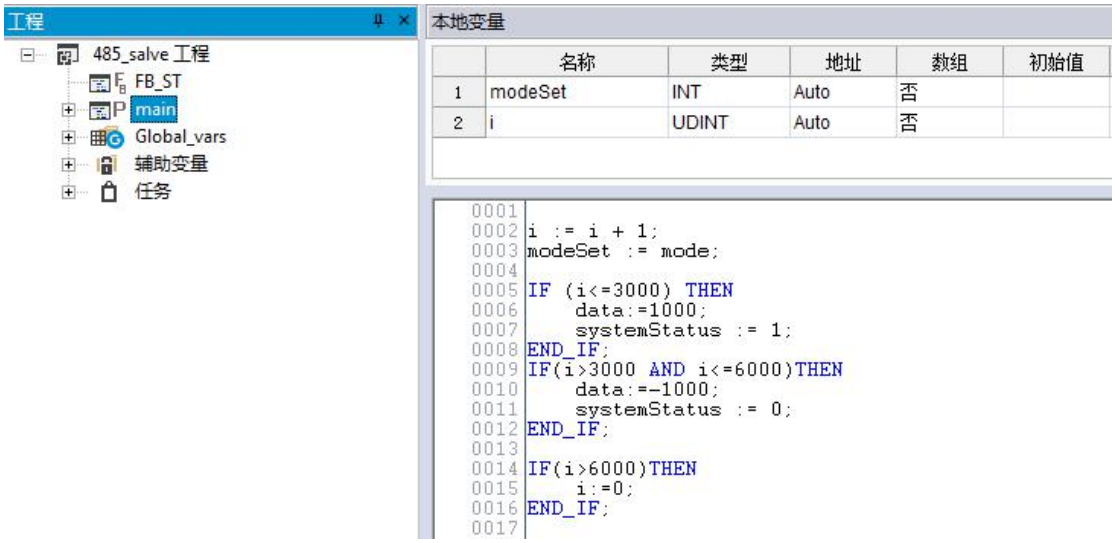


图 5.39 程序部分

4. 通讯结果

读取数据和状态:

图 5.40 读数值

主站将模式改为 1，在线调试程序，查看模式状态：

图 5.41 写数值

本地变量

	名称	类型	地址	数组	初始值	属性
1	modeSet	INT	Auto	否		..
2	i	UDINT	Auto	否		..

```
0001
0002 i    1103    := i    1103    + 1;
0003 modeSet  1     := mode  1    ;
0004
0005 IF (i    1103    <=3000) THEN
0006     data  1000   :=1000;
0007     systemStatus   1     := 1;
0008 END_IF;
0009 IF(i    1103    >3000 AND i    1103    <=6000)THEN
0010     data  1000   :=-1000;
0011     systemStatus   1     := 0;
0012 END_IF;
0013
0014 IF(i    1103    >6000)THEN
0015     i    1103    :=0;
0016 END_IF;
0017
```

图 5.42 成功写入数值

5.3.3 PLC 作为从站实现掉电保持功能

1. 新建程序：新建程序，并分配到 Init 任务中（Init 任务只在系统初始化的时候执行一次）。

图 5.43 新建程序

2. 在 modbus 读写数据区中定义寄存器变量。

#	地址（十进制）	地址（十六进制）	名称	参数类型	PLC 类型
1	16384	0x4000	a	Unsigned 16-bit	WORD
2	16385	0x4001	b	Unsigned 16-bit	WORD
3	16386	0x4002	c	Unsigned 16-bit	WORD

图 5.44 定义寄存器变量

3. 定义全局变量，并将参数属性设置为 RETAIN 型。

	名称	类型	地址	数组	初始值	属性
1	cnt	INT	Auto	否		---
2	GLV_a	WORD	Auto	否		RETAIN
3	GLV_b	WORD	Auto	否		RETAIN
4	GLV_c	WORD	Auto	否		RETAIN

图 5.45 定义全局变量

4. 编写程序

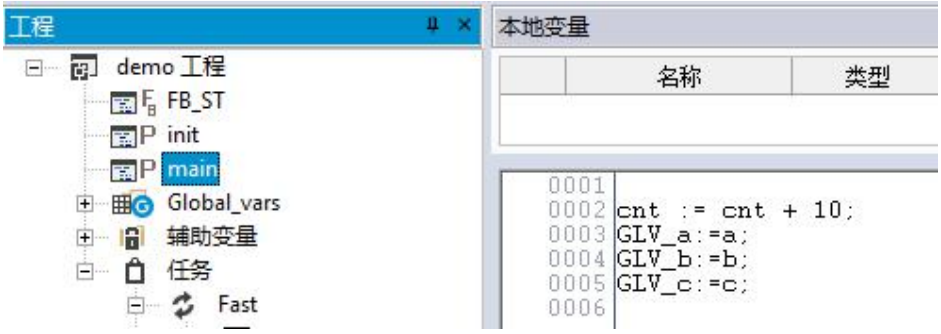


图 5.46 主程序

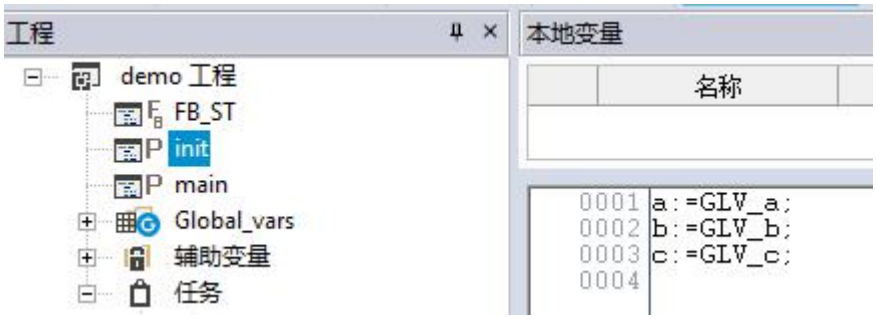


图 5.47 初始化程序

5. 触摸屏设置寄存器变量值，PLC 断电后重新上电，PLC 中寄存器变量的数值掉电保持不变。

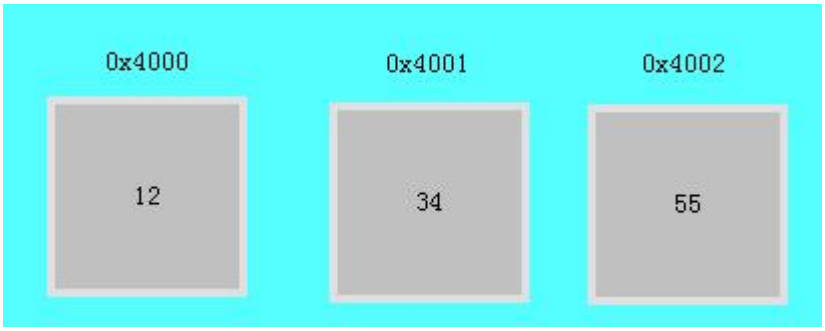


图 5.48 触摸屏设置寄存器数值

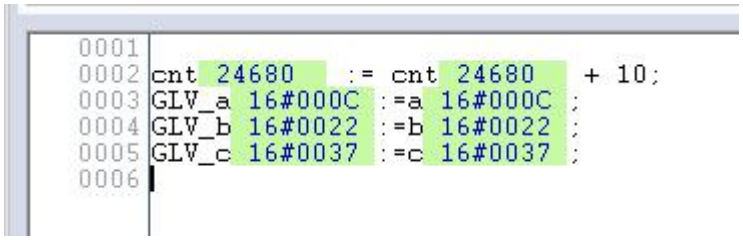


图 5.49 PLC 重新上电后寄存器数值

5.4 ModbusRTU 库函数

EC400PLC 支持标准 ModbusRTU 功能码（01 02 03 04 05 06 15 16），在库的树节点页面可以查看所有 ModbusRTU 功能块。

注：调用 ModbusRTU 库函数的程序必须配置在 Background 任务中。

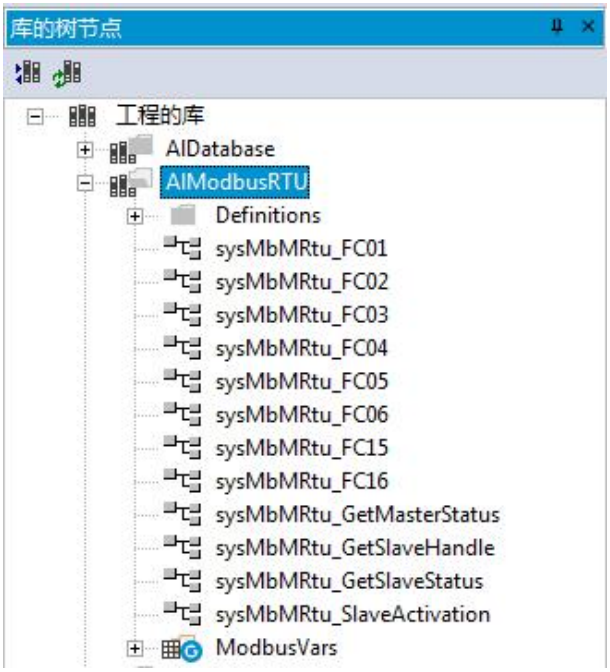


图 5.50 ModbusRTU 功能块

选中功能块，在右键菜单中选择“对象属性”（快捷键：Alt + Enter），可以在属性窗口中查看功能块的参数，以及参数含义。

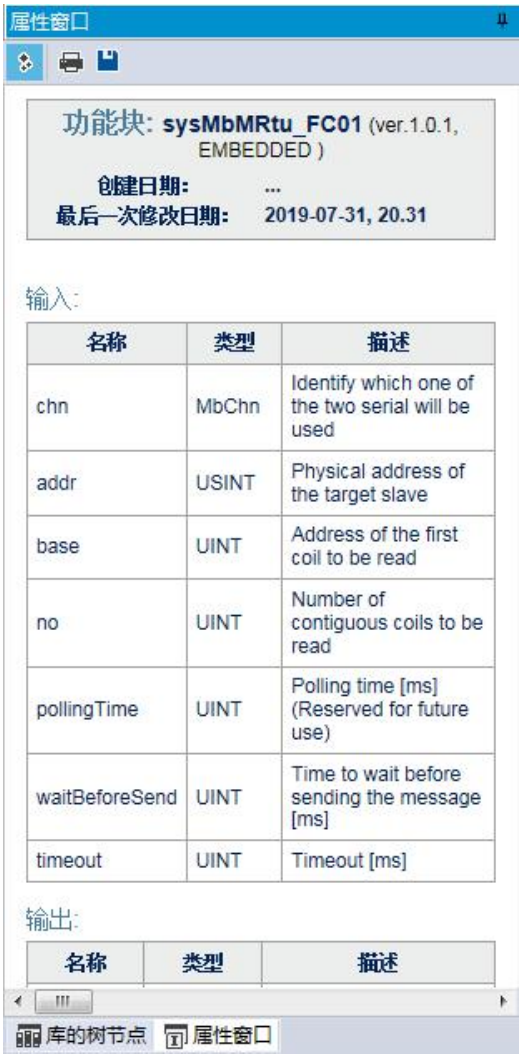


图 5.51　ModbusRTU 功能块属性

5.4.1 ModbusRTU 读取命令简介

下面对于各请求命令的“应答格式”的描述是从站正确的应答格式。若是异常应答，那么返回的应答帧中“功能码”部分变为如下数据：功能码的最高位置“1”后得到的数据。比如功能码为 0x01，若从站是异常的应答，则返回的功能码为 0x81。

- 功能码 01：读线圈（开关量输出）

请求格式：

目标站号	功能码	起始地址		读取个数		CRC
1 字节	01	高字节	低字节	高字节	低字节	2 字节

正确应答格式：

站号	功能码	返回数据字节数	返回数据字节 1	返回数据字节 2	...	CRC
1 字节	01	1 字节	1 字节	1 字节	...	2 字节

➢ 功能码 02：读输入状态（开关量输入）

请求格式：

目标站号	功能码	起始地址		读取个数		CRC
1 字节	02	高字节	低字节	高字节	低字节	2 字节

正确应答格式：

站号	功能码	返回数据字节数	返回数据字节 1	返回数据字节 2	…	CRC
1 字节	02	1 字节	1 字节	1 字节	…	2 字节

➢ 功能码 03：读保持寄存器（模拟量输出）

请求格式：

目标站号	功能码	起始地址		读取个数		CRC
1 字节	03	高字节	低字节	高字节	低字节	2 字节

正确应答格式：

站号	功能码	返回数据字节数	寄存器 1 高字节	寄存器 1 低字节	…	CRC
1 字节	03	1 字节	1 字节	1 字节	…	2 字节

➢ 功能码 04：读输入寄存器（模拟量输入）

请求格式：

目标站号	功能码	起始地址		读取个数		CRC
1 字节	04	高字节	低字节	高字节	低字节	2 字节

正确应答格式：

站号	功能码	返回数据字节数	寄存器 1 高字节	寄存器 1 低字节	…	CRC
1 字节	04	1 字节	1 字节	1 字节	…	2 字节

➢ 功能码 05：写单线圈（开关量输出）

请求格式：

目标站号	功能码	线圈地址		强制值		CRC
1 字节	05	高字节	低字节	高字节	低字节	2 字节

注：强制值 ＝ 0xFFFF，则置线圈为接通；强制值＝0x0000，则置线圈为断开。

应答格式：若设置成功，则原报文返回

- 功能码 06：写单保持寄存器（模拟量输出）

请求格式：

目标站号	功能码	寄存器地址		强制值		CRC
1 字节	06	高字节	低字节	高字节	低字节	2 字节

应答格式：若设置成功，则原报文返回

- 功能码 15：写多线圈（开关量输出）

请求格式：

目标站号	功能码	起始地址		读取个数		强制值字节数	...	CRC
1 字节	15	高字节	低字节	高字节	低字节	1 字节	...	2 字节

正确应答格式：

目标站号	功能码	起始地址		读取个数		CRC
1 字节	15	高字节	低字节	高字节	低字节	2 字节

- 功能码 16：写多线圈（开关量输出）

请求格式：

目标站号	功能码	起始地址		读取个数		强制值字节数	强制值 1 高字节	强制值 1 低字节	...	CRC
1 字节	16	高字节	低字节	高字节	低字节	1 字节	1 字节	1 字节	...	2 字节

正确应答格式：

目标站号	功能码	起始地址		读取个数		CRC
1 字节	16	高字节	低字节	高字节	低字节	2 字节

5.4.2 ModbusRTU 其他命令简介

1. 当 PLC 作为从站时，可以利用 sysMbMRtu_GetMasterStatus 指令，获取 Modbus RTU 主站状态。详细介绍如下：

sysMbMRtu_GetMasterStatus		
输入		
名称	类型	描述
chn	USINT	Modbus 主站通道号： RS232 为 0， RS485（DB）为 1， RS485（端子）为 2
输出		
名称	类型	描述
cfgOk	Bool	Modbus 主站配置完成
status	MBStatus	Modbus 主站状态机
lastError	MbError	Modbus 主站上次通信错误
nSalves	USINT	网站中从站个数
nQueueMsg	UINT	配置参数以及单从站数据个数
nQueuedBroadcastMsg	UINT	配置参数数据总数
nBusMsgCount	UINT	总线数据计数
nBusCommErrCount	UINT	总线错误计数

2、当 PLC 作为主站时，可以利用 sysMbMRtu_GetSlaveStatus 指令，获取 Modbus RTU 主站状态。详细介绍如下：

sysMbMRtu_GetSlaveStatus		
输入		
名称	类型	描述
chn	USINT	Modbus 通道号： RS232 为 0， RS485（DB）为 1， RS485（端子）为 2
hdl	USINT	Modbus 从站句柄，利用 sysMbMRtu_GetSlaveHandle
输出		
名称	类型	描述

adr	USINT	Modbus 从站地址
cfg	BOOL	是否已经配置完成
pres	BOOL	Modbus 存在总线
active	BOOL	通信状态
n_prm	UINT	配置参数通信数量
n_qmsg	UINT	从站配置参数数据
min_poll_time	UINT	最小轮询时间
status	MBStatus	Modbus 从站状态机
error	MbError	Modbus 从站上次通信错误
exception	MbException	
n_SlaveExceptionCount	UINT	从站返回错误数
n_SlaveMessageCount	UINT	从站通信数据计数
n_SlaveNoResponsCount	UINT	从站无响应计数
n_SlaveNAKCount	UINT	从站 NAK 计数
n_SlaveBusyCount	UINT	从站忙碌计数
n_SlaveChrovrCount	UINT	从站字节异常计数

3、获取从站句柄

sysMbMRtu_GetSlaveHandle		
输入		
名称	类型	描述
chn	USINT	Modbus 通道号： RS232 为 0， RS485（DB）为 1， RS485（端子）为 2
adr	USINT	Modbus 从站地址

4、从站在线

当从站掉站后，可以通过 sysMbMRtu_SlaveActivation 函数，实现重新连接后，自动重联。

sysMbMRtu_SlaveActivation		
输入		
名称	类型	描述
chn	USINT	Modbus 主站通道号： RS232 为 0， RS485（DB）为 1， RS485（端子）为 2
handle	USINT	Modbus 从站句柄，利用 sysMbMRtu_GetSlaveHandle
active	BOOL	True 时，从站掉站后实时触发

5、数据类型介绍

MbStatus 返回值

名称	描述
MbStatus_NotConfigurad	未配置
MbStatus_Configured	配置完成，等待运行
MbStatus_Configuring	配置中
MbStatus_Stopped	停止
MbStatus_Starting	开始，等待运行
MbStatus_Runint	运行

MbError 返回值

名称	描述
MbError_Ok	无错误
MbError_System	系统内部错误
MbError_CommOpen	打开串口失败
MbError_CommHdl	串口句柄错误
MbError_CommTx	发送请求错误
MbError_SlaveMaxNum	最大从站数量
MbError_SlaveAddressInvalid	无效从站地址
MbError_SlaveNotCfg	从站未配置到网络中
MbError_SlaveInUse	从站地址已存在
MbError_SlaveLoss	从站掉站

MbError_MsgMaxNum	Modbus 通信最多数量
MbError_MsgFnzInvalid	无效通信功能
MbError_MsgFnzSize	无效通信函数长度
MbError_MsgDatablockInvalid	无效数据地址
MbError_ParamInvalid	系统内部错误
MbError_ParamException	无响应
MbError_ParamTimeout	超时

第六章 PLC 拓展总线

6.1 通讯协议

EC400 型 PLC 支持模块拓展功能，使用 ExtBus 拓展总线协议，用于拓展 PLC 的 IO 的数量和种类，具有简单易用、传输速率高等特点。ExtBus 的功能拓扑结构，如图 6.1 所示。

图 6.1 extBus 功能拓扑结构

通信端包括 Master 和 Node 两种类型，Master 负责发送控制数据和采集 Node 的状态数据；Node 负责采集现场数据并反馈至 Master，同时将接收的 Master 控制数据输出至现场设备。

6.2 ExtBus 通信配置

6.2.1 添加扩展模块

1. 使能 ExtBus 功能：点击左侧资源配置树上的端口设备“扩展配置”，在右侧的“扩展模块配置”页面中使能 ExtBus 功能模式，并配置组态延时、重试次数等参数。

图 6.2 使能 ExtBus 功能

2. 添加扩展模块：选中左侧资源配置树中的“扩展配置”，右键选择“添加”菜单，弹出“设备目录”对话框，选择要添加的扩展模块。E1016 是数字量输入模块，E2016 是数字量输出模块，E3004 是模拟量输入模块，E4004 是模拟量输出模块。

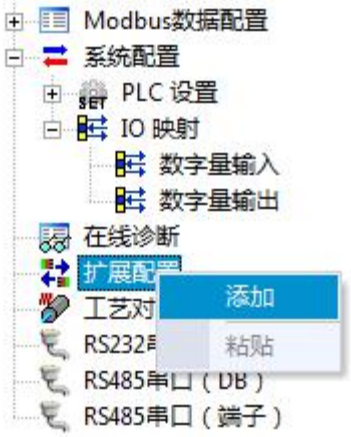


图 6.3 添加扩展模块

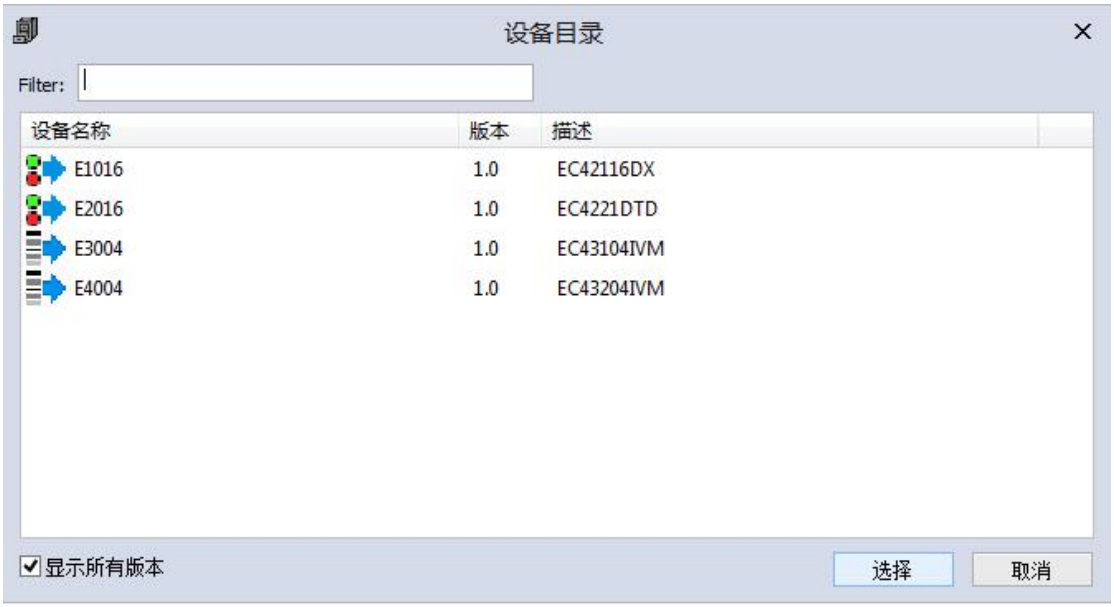


图 6.4 选择扩展模块

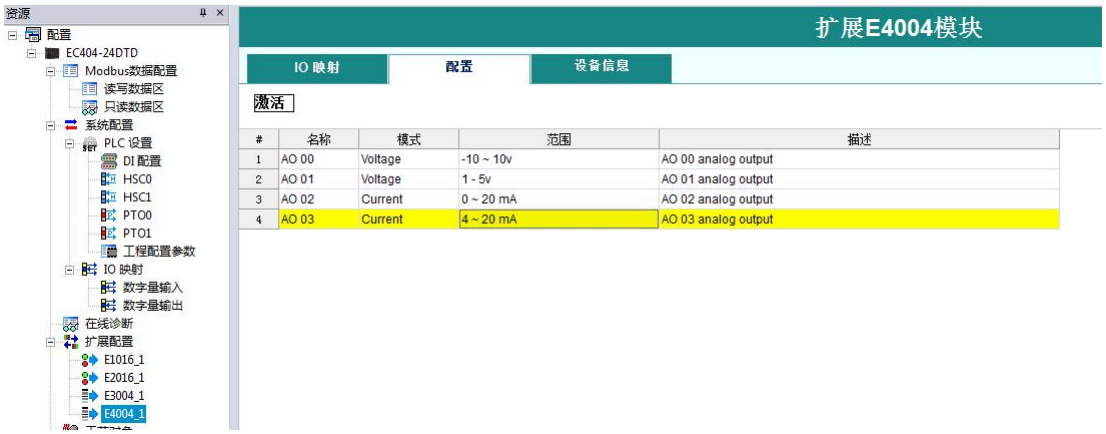
图 6.5 添加扩展模块后的设备树

3. 点击扩展模块在相应页面，进行变量映射

#	名称	变量	类型	数据块	描述
1	DI 00	DI_0	BOOL	%IX100.0	DI 00 digital output
2	DI 01		BOOL	%IX100.1	DI 01 digital output
3	DI 02		BOOL	%IX100.2	DI 02 digital output
4	DI 03		BOOL	%IX100.3	DI 03 digital output
5	DI 04		BOOL	%IX100.4	DI 04 digital output
6	DI 05		BOOL	%IX100.5	DI 05 digital output
7	DI 06		BOOL	%IX100.6	DI 06 digital output
8	DI 07		BOOL	%IX100.7	DI 07 digital output
9	DI 08		BOOL	%IX100.8	DI 08 digital output
10	DI 09		BOOL	%IX100.9	DI 09 digital output
11	DI 10		BOOL	%IX100.10	DI 10 digital output
12	DI 11		BOOL	%IX100.11	DI 11 digital output
13	DI 12		BOOL	%IX100.12	DI 12 digital output
14	DI 13		BOOL	%IX100.13	DI 13 digital output
15	DI 14		BOOL	%IX100.14	DI 14 digital output
16	DI 15		BOOL	%IX100.15	DI 15 digital output

图 6.6 映射变量

4. 参数配置：AI 和 AO 模块，用户可根据实际需求配置每个通道的参数。



#	名称	模式	范围	描述
1	AO 00	Voltage	-10 ~ 10v	AO 00 analog output
2	AO 01	Voltage	1 - 5v	AO 01 analog output
3	AO 02	Current	0 ~ 20 mA	AO 02 analog output
4	AO 03	Current	4 ~ 20 mA	AO 03 analog output

图 6.7 通道参数配置

6.3 ExtBus 通信实例

6.3.1 连接 DO 模块的 ExtBus 通信

1. 使能 ExtBus 功能：点击配置树节点下的“扩展配置”节点，弹出扩展模块配置页面，配置模式为使能。

图 6.8 使能 ExtBus 功能

2. 程序中添加 DO 模块：选中“扩展配置”节点右键，在右键菜单中选择“添加”，在弹出的“设备目录”中选择 E2016。

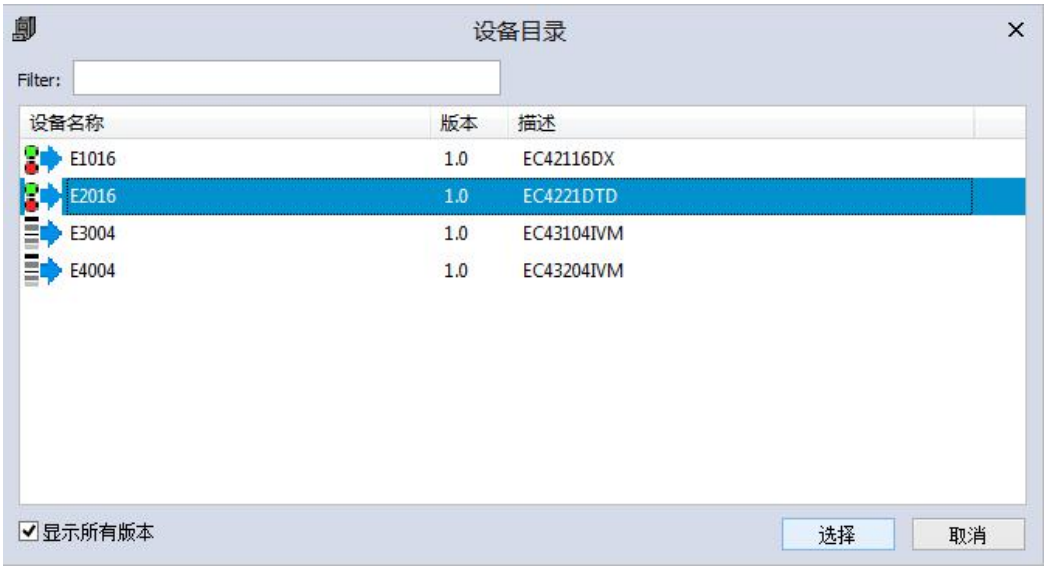


图 6.9 添加 DO 模块

3. 变量映射：在 Variable 列映射变量。

#	名称	变量	类型	数据块	
1	DO 00	DO_1	BOOL	%QX100.0	DO 00 digital output
2	DO 01	DO_2	BOOL	%QX100.1	DO 01 digital output
3	DO 02	DO_3	BOOL	%QX100.2	DO 02 digital output
4	DO 03	DO_4	BOOL	%QX100.3	DO 03 digital output

图 6.10 映射变量

4. 程序中调用变量

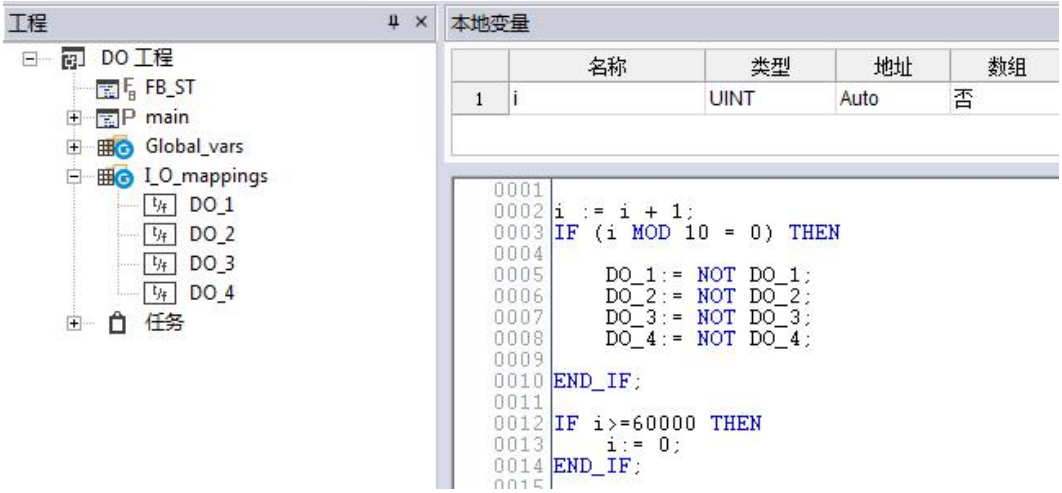


图 6.11 调用变量

5. 编译、下载程序，并重启设备。

6.3.2 连接 DI 模块的 ExtBus 通信

1. 使能 ExtBus 功能：点击配置树节点下的“扩展配置”节点，弹出扩展模块配置页面，配置模式为使能。

图 6.12 使能 extbus

2. 程序中添加 DI 模块：选中“ExtBus”节点右键，在右键菜单中选择“添加”，在弹出的“设备目录”中选择 E1016。

图 6.13 添加 DI 模块

3. 变量映射：在 Variable 列映射变量。

图 6.14 映射变量

4. 程序中调用变量

图 6.15 调用变量

5. 编译、下载程序，并重启设备。

6.3.3 连接 AO 模块的 ExtBus 通信

1. 使能 ExtBus 功能：点击配置树节点下的“扩展配置”节点，弹出扩展模块配置页面，配置模式为使能。

图 6.16 使能 ExtBus

2. 程序中添加 AO 模块：选中“扩展配置”节点右键，在右键菜单中选择“添加”，在弹出的“设备目录”中选择 E4004。

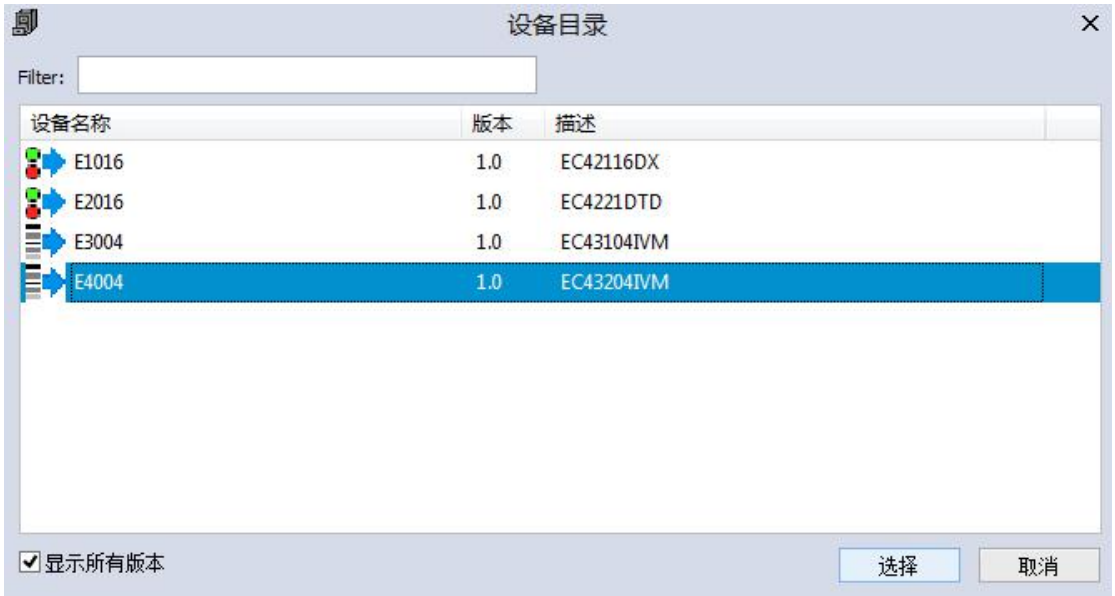


图 6.17 添加 AO 模块

3. 变量映射：在 Variable 列映射变量。

图 6.18 映射变量

4. 参数配置：在配置页面设置各个通道的模式、范围。

图 6.19 配置通道参数

5. 程序中调用变量

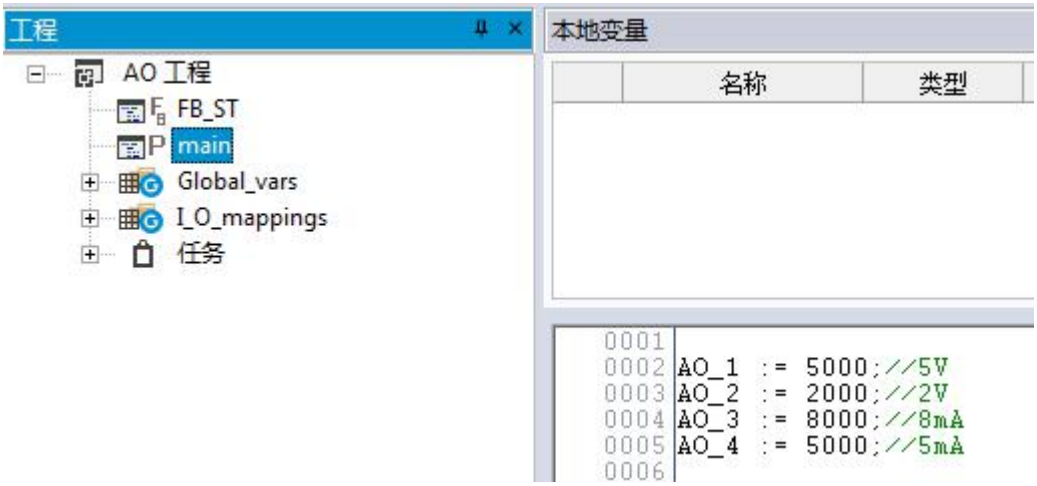


图 6.20 调用变量

6. 编译、下载程序，并重启设备。

6.3.4 连接 AI 模块的 ExtBus 通信

1. 使能 ExtBus 功能：点击配置树节点下的“扩展配置”节点，弹出扩展模块配置页面，配置模式为使能。

图 6.21 使能 extbus

2. 程序中添加 AI 模块：选中“扩展配置”节点右键，在右键菜单中选择“添加”，在弹出的“设备目录”中选择 E3004。

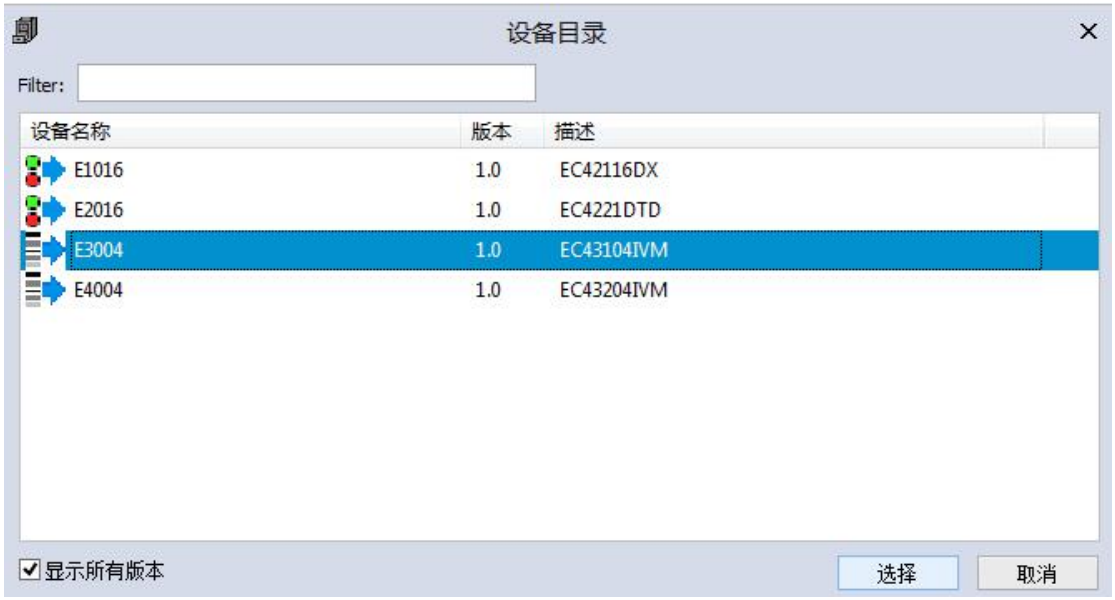


图 6.22 添加 AI 模块

3. 变量映射：在 Variable 列映射变量。

图 6.23 映射变量

4. 参数配置：在配置页面设置各个通道的模式、范围和滤波方式。

图 6.24 参数配置

5. 程序中调用变量

图 6.25 调用变量

6. 编译、下载程序，并重启设备。

第七章 PLC 高速计数器的应用和运动控制

7.1 高速计数器简介

高速计数器能对超出 CPU 普通计数器能力的脉冲信号进行测量。EC400PLC 提供了两个高速计数器（HSC0、HSC1）以响应快速脉冲输入信号。高速计数器的计数速度比 PLC 的扫描速度要快得多，因此高速计数器可独立与用户程序工作，不受扫描时间的限制。高速计数器的典型应用是利用光电编码器测量转速和位移。

7.1.1 高速计数器的工作模式

高速计数器有 5 种工作模式，每个计数器都有时钟、方向控制等特定输入。对于两个相位计数器，两个时钟都可以运行在最高频率，高速计数器的最高计数频率取决于 CPU 的类型和信号板的类型。高速计数器 5 种工作模式介绍如下。

(1) 单向计数，内部控制方向

单相计数的原理如下图所示，计数器采集并记录时钟信号的个数，当内部方向为高电平时，计数的当前数值增加；当内部方向信号为低电平时，计数的当前数值减小。

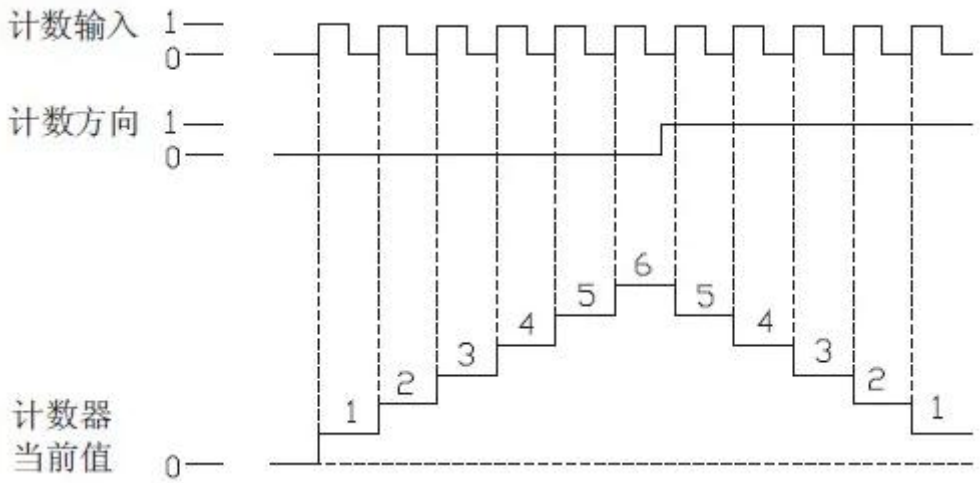


图 7.1 单相计数原理

(2) 单向计数，外部控制方向

单相计数的原理如图 7.1 所示，计数器采集并记录时钟信号的个数，当外部方向信号（例如外部按钮信号）为高电平时，计数的当前数值增加；当外部方向信号为低电平时，计数的当前数值减小。

(3) 两相计数，两路时钟脉冲输入

加减两相计数原理如图 7.2 所示，计数器采集并记录时钟信号的个数，加计数信号端子和减信号计数端子分开。当加计数有效时，计数的当前数值增加；当减计数有效时，计数的当前数值减少。

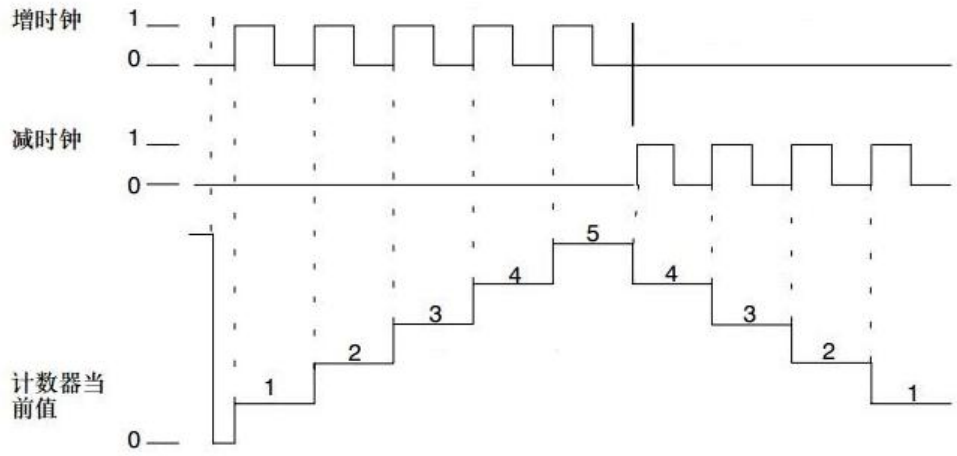


图 7.2 加减两相计数原理

(4) **A/B 相正交计数**

A/B 相正交计数原理如图 7.3 所示，计数器采集并记录时钟信号的个数，A 相计数信号端子和 B 相信号计数端子分开。当 A 相计数信号超前时，计数的当前数值增加；当 B 相计数信号超前时，计数的当前数值减少。利用光电编码器测量位移和速度，通常采用这种模式。

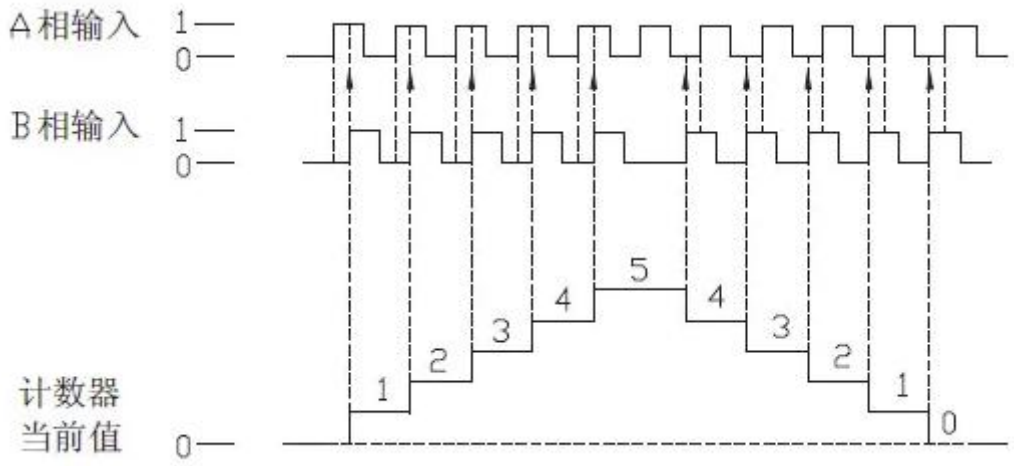


图 7.3　A/B 相正交计数原理

(5) 4 倍速 A/B 相正交计数

A/B 相正交计数原理如图 7.4 所示，计数器采集并记录时钟信号的个数，A 相计数信号端子和 B 相信号计数端子分开。当 A 相计数信号超前时，计数的当前数值以 4 倍速度增加；当 B 相计数信号超前时，计数的当前数值以 4 倍速度减少。

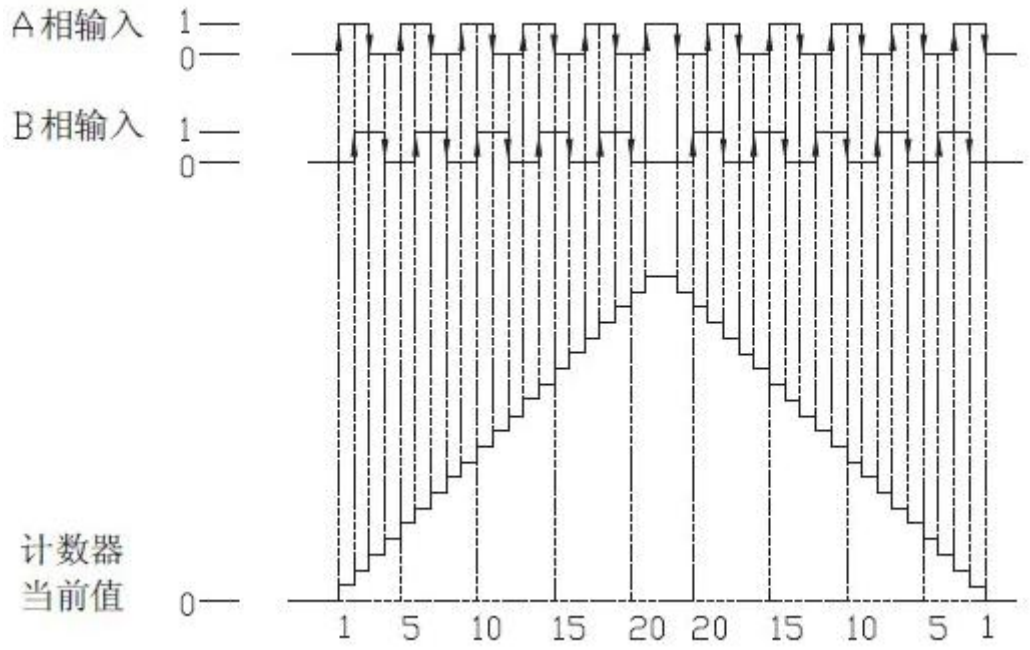


图 7.4 4 倍速 A/B 相正交计数原理

7.1.2 高速计数器的硬件输入

高速计数器的硬件输入接口与普通数字量接口使用相同的地址，已经定义用于高速计数器的输入点不能再用于其他功能。没有用到的输入点还可以用作开关量输入点。

表 7-1 PLC 模式和输入分配

项目	类型	模式	输入点	
HSC0	计数模式/频率模式	单相计数，内部方向控制	I0.0（时钟）	
		单相计数，外部方向控制	I0.0（时钟）	I0.1（方向）
		两相计数，两路时钟脉冲输入	I0.0（加时钟）	I0.1（减时钟）
		A/B 相正交正交计数	I0.0（A 相）	I0.1（B 相）
		4 倍速 A/B 相正交计数	I0.0（A 相）	I0.1（B 相）
HSC1	计数模式/频率模式	单相计数，内部方向控制	I0.2（时钟）	
		单相计数，外部方向控制	I0.2（时钟）	I0.3（方向）
		两相计数，两路时钟脉冲输入	I0.2（加时钟）	I0.3（减时钟）
		A/B 相正交正交计数	I0.2（A 相）	I0.3（B 相）
		4 倍速 A/B 相正交计数	I0.2（A 相）	I0.3（B 相）

7.1.3 高速计数器的应用

EC400PLC 的高速计数器主要支持计数模式与频率模式，其支持脉冲+方向、CW/CCW、AB 正交和 AB4 倍频正交等不同类型的输入信号。

计数模式：

1. 在配置树页面点击 HSC0，弹出 HSC 配置界面，在配置页面使能高速计数器功能，并配置高速计数器的工作模式。

图 7.5 高速计数器模式配置

2. 在“变量映射”页面映射变量，并获取高速计数器的数值，在程序中可直接调用此变量。

图 7.6 高速计数器变量映射

3. 在程序里可以利用相应的功能函数进行设置。

1) HSC_Work_Ctrl 高速计数运行函数，当用户调用此函数，则高速计数启停由用户编程控制。

HSC_Work_Ctrl		
输入		
名称	类型	描述
udiChn	UDINT	HSC通道号
xEnable	BOOL	高电平有效

2) HSC_CV_Ctrl 计数设置值设置

HSC_CV_Ctrl		
输入		
名称	类型	描述
udiChn	UDINT	HSC通道号
xEnCV	BOOL	高电平有效
DINewCV	DINT	初始值

3) HSC_Dir_Ctrl 计数方向设置

HSC_Dir_Ctrl		
输入		
名称	类型	描述
udiChn	UDINT	HSC通道号
xEnDir	BOOL	高电平有效
DiNewDir	DINT	方向设定 1为正向 -1为反向

频率模式：

1、 在 HSC 配置界面中，使能高速计数，并选择频率模式

图 7.7 高速计数器模式配置

2、在“变量映射”页面映射变量，并获取高速计数器的数值，在程序中可直接调用此变量。

图 7.8 高速计数器变量映射

3、在程序里可以利用相应的功能函数进行设置。

1) HSC_Work_Ctrl 高速计数运行函数，当用户调用此函数，则高速计数启停由用户编程控制。

HSC_Work_Ctrl		
输入		
名称	类型	描述
udiChn	UDINT	HSC通道号
xEnable	BOOL	高电平有效

2) HSC_Period_Ctrl 脉冲采样周期设置

HSC_Period_Ctrl		
输入		
名称	类型	描述
udiChn	UDINT	HSC通道号
xEnPeriod	BOOL	高电平有效
u16NewPeriod	DINT	采样周期

7.2 PLC 在运动控制中的应用

7.2.1 运动控制简介

运动控制起源于早期的伺服控制。简单的说，运动控制就是对机械运动部件的位置、速度等进行实时的控制管理，使其按照预期的运动轨迹和规定的运动参数进行运动。

EC400PLC 在运动控制中使用了轴的概念，通过轴的组态，包括硬件接口、位置定义、动态性能和机械特性等，与相关的指令块组合使用，可实现绝对位置、相对位置、点动、转速控制以及寻找参考点等功能。

EC400PLC 的运动控制指令块符合 PLCopen Motion 规范。

7.2.2 PLC 的运动控制功能

EC404-24DTD PLC 提供了两路高速脉冲输出（PTO0 和 PTO1），相应地就提供了两个 PTO 脉冲发生器用于产生 PTO 输出，输出频率最高可达 200kHz。在这里，高速脉冲输出指的是 PTO 或者 PWM 输出。

- 脉冲串输出（PTO）：内置于 CPU 中，输出指定数量的脉冲队列，可用于脉冲控制伺服驱动器或步进驱动器。

脉冲串操作（PTO）按照给定的脉冲个数和周期输出一串方波（占空比 50%）。PTO 可以产生单段脉冲串或多段脉冲串。可以 us 或者 ms 为单位指定脉冲宽度和周期。

7.2.3 高速脉冲输出的硬件输出分配

高速脉冲输出的硬件输出接口与普通数字量接口使用相同的地址，已经定义用于高速脉冲输出的输出点不能再用于其他功能。没有用到的输出点还可以用作开关量输出点。

注意： 若Q0.0、Q0.1、Q0.2、Q0.3是继电器类型的，则避免使用高速脉冲输出功能！

表 7-2 PLC 模式和输出分配

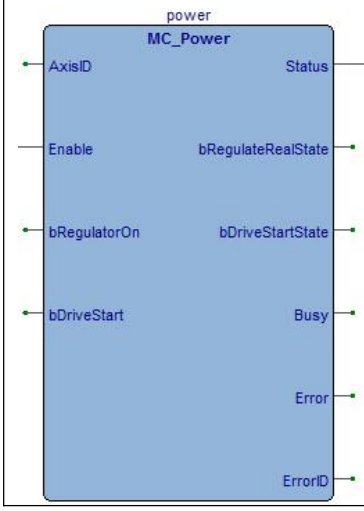
项目	模式	输出点	
PTO0	脉冲加方向	Q0.0（脉冲）	Q0.1（方向）
	双向脉冲	Q0.0（A 相）	Q0.1（B 相）
	AB 相正交脉冲	Q0.0（A 相）	Q0.1（B 相）
PTO1	脉冲加方向	Q0.2（脉冲）	Q0.3（方向）
	双脉冲	Q0.2（A 相）	Q0.3（B 相）
	AB 相正交脉冲	Q0.2（A 相）	Q0.3（B 相）

7.2.4 EC400PLC 的运动控制功能库

(1) MC_Power 轴使能指令

轴在运动之前，必须使用该功能块，以使 PLC 内部的运动控制块正常工作，其具体参数说明见表 7-3。

表 7-3 MC_Power 系统使能指令块的参数

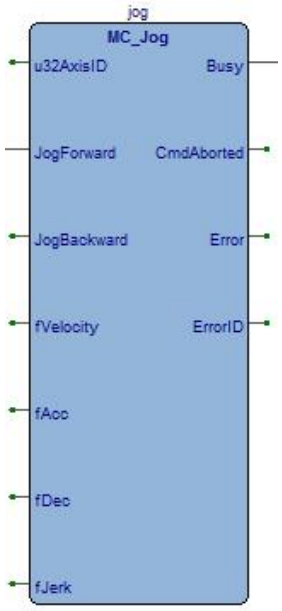
LD	ST	输入/输出	参数的含义
	power(AxisID, bDriveStart, bRegulatorOn, Enable, bDriveStartState, bRegulateRealState, Busy, Error, ErrorID, Status)	Enable	1：使能轴
		AxisID	0：PTO0 1：PTO1
		bDriveStart	1：有效，轴可运动控制
		bRegulatorOn	1：有效，轴使能控制
		Status	轴状态
		bDriveStartState	驱动具备快速停机
		bRegulateRealState	电源输出

		Busy	1：任务正在执行
		ErrorID	错误 ID 码
		Error	错误信息

(2) MC_Jog 点动运行功能块

MC_Jog 点动运行功能块用于轴的点动运动，当使能 JogForward 后，轴正向运动，当使能 JogBackward 后，轴反向运动。点动运行指令块具体参数说明见表 7-4。其具体参数说明见表 7-4。

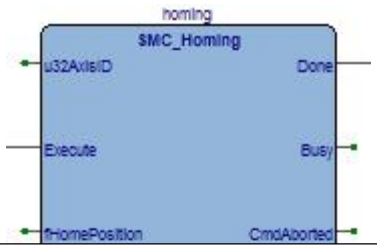
表 7-4 MC_Jog 点动指令块的参数

LD	ST	输入/输出	参数的含义
	jog(u32AxisID, JogForward, JogBackward, fVelocity, fJerk, fAcc, fDec, Busy, CmdAborted, Error, ErrorID);	AxisID	0：PTO0 1：PTO1
		JogForward	正向点动
		JogBackward	反向点动
		fVelocity	速度
		fJerk	加加速度
		fAcc	加速度
		fDec	减速度
		Busy	1：任务正在执行
		CmdAborted	1：任务正在执行期间被另一任务中止
		ErrorID	错误 ID 码
		Error	错误信息

(3) MC_Homing 回参考点指令块

参考点在系统中有时作为坐标原点，对于运动控制系统是非常重要的。回参考点指令块具体参数说明见表 7-5。

表 7-5 MC_Homing 回参考点指令块的参数

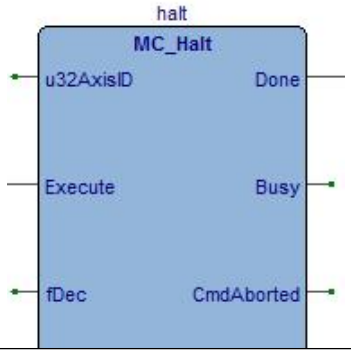
LD	ST	输入/输出	参数的含义
	homing(u32AxisID, Execute, fIndexPosition, fVelocityFast, fAcc, fDec,	Execute	上升沿使能
		u32AxisID	0：PTO0 1：PTO1
		fIndexPosition	未使用
		fVelocityFast	快速移动速度

	fJerk, fVelocitySlow, i32McDir, xIgnoreHWLimit, xIndexSignal, xRefSwitch, xReturnZero， u32Mode, Busy, CmdAborted, Done, Error, ErrorID);	fVelocitySlow	慢速
		fAcc	加速度
		fDec	减速度
		fJerk	刹车速度
		i32McDir方向，-1 反方向，0/1 正向	
		xIgnoreHWLimit	未使用
		xIndexSignal	未使用
		xRefSwitch	原点信号
		xReturnZero	未使用
		u32Mode	回原点模式，1 为碰到原点立即停止
		CmdAborted	1：任务正在执行期间被另一任务中止
		Error	错误信息
		ErrorID	错误码
		Done	1：任务完成
		Busy	1：任务正在执行

(4) MC_Halt 停止轴指令块

MC_Halt 停止轴指令块用于停止轴的运动，当上升沿使能 Execute 后，轴会按照组态好的减速曲线停车。停止轴指令块具体参数说明见表 7-6。

表 7-6 MC_Halt 停止轴指令块指令块的参数

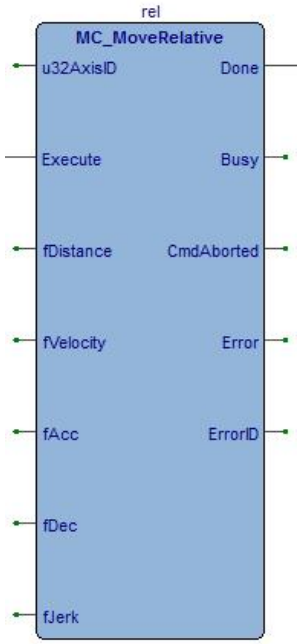
LD	ST	输入/输出	参数的含义
	halt(u32AxisID, Execute, fDec, fJerk, Busy, CmdAborted, Done, Error,	Execute	上升沿使能
		u32AxisID	0：PTO0 1：PTO1
		fDec	减速度
		fJerk	刹车速度
		Done	1：速度达到零
		Busy	1：正在执行任

	ErrorID);		务
		Error	错误信息
		ErrorID	错误码
		CmdAborted	1：任务正在执行期间被另一任务中止

(5) MC_MoveRelative 相对定位轴指令块

MC_MoveRelative 相对定位轴指令块的执行不需要建立参考点，只需要定义距离、速度和方向即可。当上升沿使能 Execute 后，轴按照设定的速度和距离运行，其方向由距离中的正负号（+/-）决定。相对定位轴指令块具体参数说明见表 7-7。

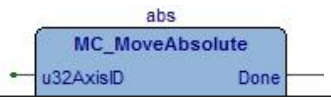
表 7-7 MC_MoveRelative 相对定位轴指令块指令块的参数

LD	ST	输入/输出	参数的含义
	rel(Execute, fAcc, fDec, fDistance, fJerk, fVelocity, u32AxisID, Busy, Done, Error, ErrorID, CmdAborted);	Execute	上升沿使能
		u32AxisID	0：PTO0 1：PTO1
		fAcc	加速度
		fDec	减速度
		fJerk	刹车速度
		fDistance	运行距离（正或者负）
		fVelocity	定义的速度
		Done	1：已到达目标位置
		Busy	1：任务正在执行
		Error	错误信息
		ErrorID	错误码
		CmdAborted	1：任务正在执行期间被另一任务中止

(6) MC_MoveAbsolute 绝对定位轴指令块

MC_MoveAbsolute 绝对定位轴指令块的执行需要建立参考点，通过定义距离、速度和方向即可。当上升沿使能 Execute 后，轴按照设定的速度和绝对位置运行。绝对定位轴指令块具体参数说明见表 7-8。

表 7-8 MC_MoveAbsolute 绝对定位轴指令块指令块的参数

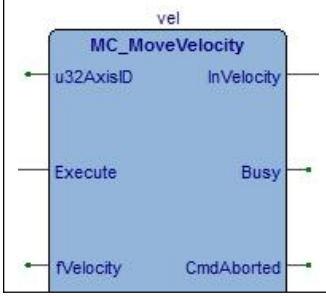
LD	ST	输入/输出	参数的含义
	abs(Execute, fAcc,	Execute	上升沿使能
		u32AxisID	0：PTO0

	fDec, fPosition, fJerk, fVelocity, u32AxisID, Busy, Done, Error, ErrorID, CmdAborted);		1：PTO1
		fAcc	加速度
		fDec	减速度
		fJerk	加加速度
		fPosition	运行距离（正或者负）
		fVelocity	定义的速度
		Done	1：已到达目标位置
		Busy	1：正在执行任务
		Error	错误信息
		ErrorID	错误码
		CmdAborted	1：任务正在执行期间被另一任务中止

(7) MC_MoveVelocity 指定速度指令块

MC_MoveVelocity 速度指令块的执行通过定义速度和方向即可。当上升沿使能 Execute 后，轴按照设定的速度和方向运行。速度指令块具体参数说明见表 7-8。

表 7-8 MC_MoveVelocity 绝对定位轴指令块指令块的参数

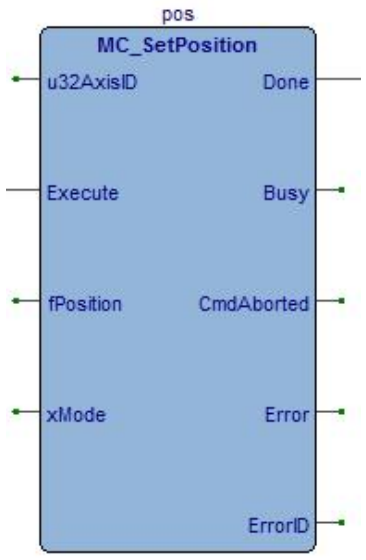
LD	ST	输入/输出	参数的含义
	vel(u32AxisID, Execute, fAcc, fDec, i32McDir, fJerk, fVelocity,	Execute	上升沿使能
		u32AxisID	0：PTO0 1：PTO1
		fAcc	加速度
		fDec	减速度
		i32McDir	方向信号

	InVelocity, Busy, Error, ErrorID, CmdAborted);	fVelocity	定义的速度 限制：启动/停止速度≤Velocity≤最大速度
		InVelocity	初次到达设置值
		Busy	1：正在执行任务
		Error	错误信息
		ErrorID	错误码
		CmdAborted	1：任务正在执行期间被另一任务中止

(8) MC_SetPosition 指定位置指令块

MC_SetPosition 位置指令块的执行通过定义速度和方向即可。当上升沿使能 Execute 后，轴按照设定的速度和方向运行。速度指令块具体参数说明见表 7-8。

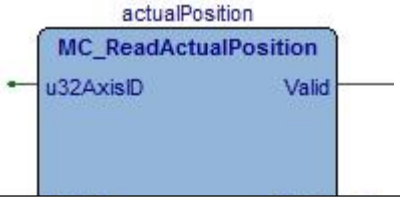
表 7-8 MC_SetPosition 位置指令块指令块的参数

LD	ST	输入/输出	参数的含义
	pos(u32AxisID, Execute, fPosition, xMode, fVelocity, Done, Busy, Error, ErrorID, CmdAborted);	Execute	上升沿使能
		u32AxisID	0：PTO0 1：PTO1
		fPosition	位置
		xMode	模式
		Done	1：运行到指定位置
		Busy	1：正在执行任务
		Error	错误信息
		ErrorID	错误码
		CmdAborted	1：任务正在执行期间被另一任务中止

(9) MC_ReadActualPosition 读取实时位置指令块

MC_ReadActualPosition 指令块具体参数说明见表 7-9。

表 7-9 MC_ReadActualPosition 读取实时位置指令块的参数

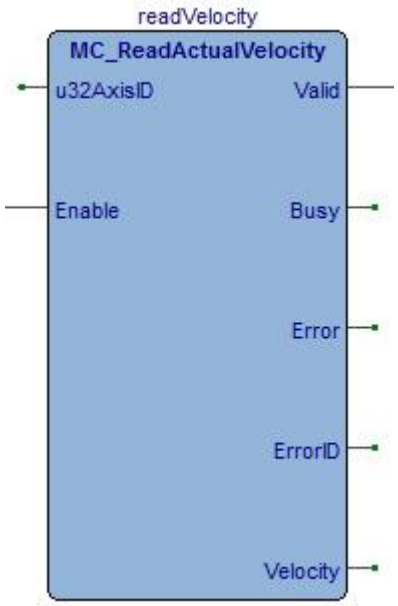
LD	ST	输入/输出	参数的含义
	actualPosition(u32AxisID, Enable, Valid, Busy,	Enable	1：使能
		u32AxisID	0：PTO0 1：PTO1
		Valid	有效输出

	Error, ErrorID, Position);	Busy	1：正在执行任务
		Error	错误信息
		ErrorID	错误码
		Position	伺服轴的实时位置

(10) MC_ReadActualVelocity 读取实际速度指令块

MC_ReadActualVelocity 指令块具体参数说明见表 7-10。

表 7-10 MC_ReadActualVelocity 读取实际速度指令块的参数

LD	ST	输入/输出	参数的含义
	readVelocity (u32AxisID, Enable, Valid, Busy, Error, ErrorID, Velocity);	Enable	1：使能
		u32AxisID	0：PTO0 1：PTO1
		Valid	有效输出
		Busy	1：正在执行任务
		Error	错误信息
		ErrorID	错误码
		Velocity	伺服轴的实际速度

7.2.5 EC400PLC 的运动控制实例

使用 EC400PLC 配合伺服实现相对运动正反转功能，实现步骤如下：

1、PTO 配置：在 PTO1 页面进行配置，使能输出，并指定输出模式。

图 7.9　PTO 配置

2、添加伺服轴：点击“工艺轴对象”，在右侧的配置页面中选择使能，右键“工艺对象”，选择“添加”，弹出“设备目录”对话框，选择轴，点击“选择”。

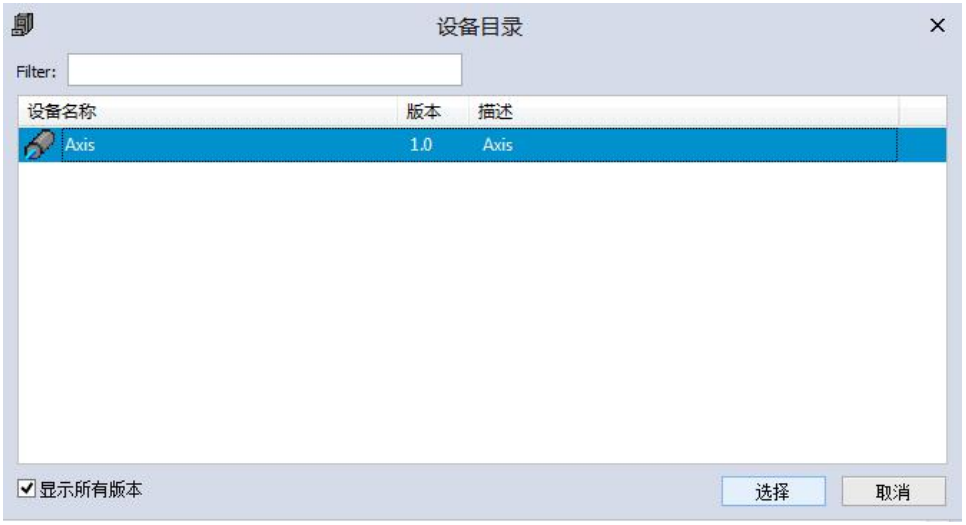


图 7.10 添加伺服轴

3、配置轴驱动：点击工艺对象下面的 Axis_1，在右侧的轴对象节点中配置轴的驱动为 PTO。

图 7.11 轴对象节点驱动配置

4、轴参数配置：在初始化参数中选择驱动为 PTO1。

图 7.12 轴对象节点初始化参数配置

5、编写控制程序，使伺服轴相对运动到 100000 的位置后再返回到起始位置，如此往复运动。同时，使用 MC_ReadActualPosition 获取伺服位置指令实时获取位置信息。

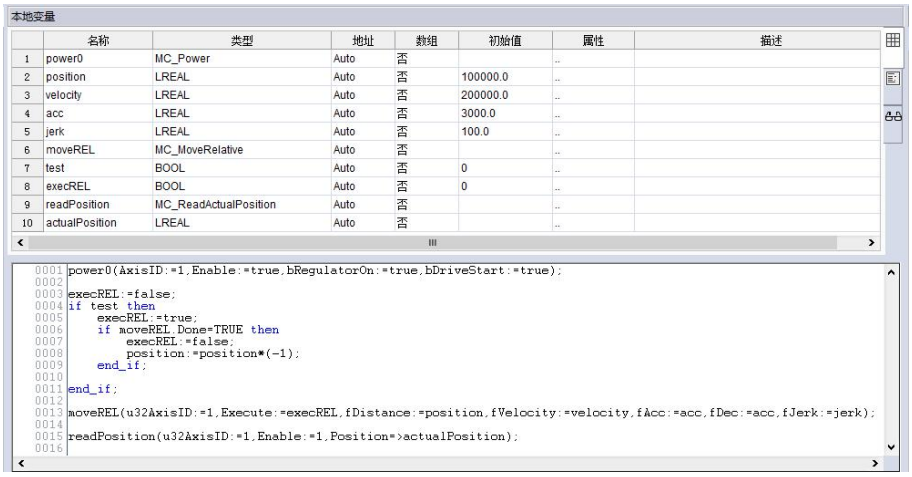


图 7.13 应用程序界面

其中,参数的单位均为脉冲个数，fVelocity 设置为 200k，即为 PLC 输出最高频率。

6、编译、下载工程，并重启 PLC，同时利用示波器工具进行监控伺服轴的实时位置。

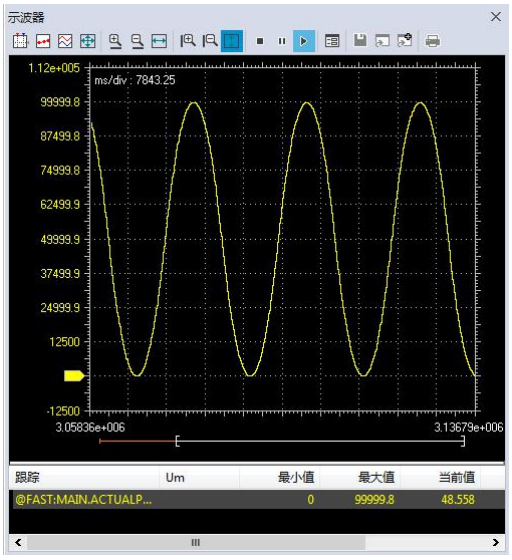


图 7.14 轴的实际运动轨迹

第八章 EURA Studio 其他功能

8.1 标准库函数

8.1.1 边缘检测

边沿检测指令用来检测 BOOL 信号的上升沿（信号由 0---->1）和下降沿（信号由 1--->0）的变化。在每个扫描周期中把信号状态和它在前一个扫描周期的状态进行比较，若不同则表明有一个跳变沿。因此，前一个周期里的信号状态必须被存储，以便能和新的信号状态相比较。边沿检测指令表如下所示。

表 8-1 边沿检测指令

功能名	图形化语言	文本化语言	说明
R_TRIG		R_TRIG	上升沿检测
F_TRIG		F_TRIG	下降沿检测

边沿检测指令参数如下表所示。

表 8-2 边沿检测指令参数

名称	定义	数据类型	说明
CLK	输入变量	BOOL	被检测信号输入
Q	输出变量	BOOL	触发器状态输出

1. 上升沿检测 R_TRIG

功能：用于检测上升沿。

语法：当 CLK 从“0”变为“1”时，该上升沿检测器开始启动，Q 输出先由“1”然后输出变为“0”，持续一个 PLC 运算周期；如果 CLK 持续保持为“1”或者“0”，Q 输出一直保持为“0”。

采集 bInput 信号的上升沿，程序如下图所示。

图 8.1 上升沿触发程序和时序

2. 下降沿检测 F_TRIG

功能：用于检测下降沿。

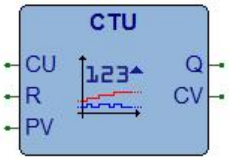
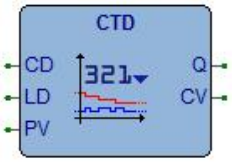
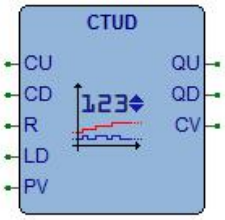
语法：当 CLK 从“1”变为“0”时，该下降沿检测器开始启动，Q 输出先由“1”然后输出变为“0”，持续一个 PLC 运算周期；如果 CLK 持续保持为“1”或者“0”，Q 输出一直保持为“0”。采集 bInput 信号的下降沿，当 bInput 由 True 变为 False 时，功能块 F_TRIG.Q 会根据下降沿的触发事件给出相应输出，输出时间维持在一个周期。程序如下图所示。

图 8.2 下降沿触发程序和时序

8.1.2 计数器

EURA Studio 标准功能库中提供了加、减计数功能块，系统提供了 CTU 加计数器、CTD 减计数器和 CTUD 加减计数器三个功能块。

表 8-3 标准计数器的图形化与文本化指令表

计数器指令	图形化语言	文本化语言	说明
计数器		CTU	增计数器
		CTD	减计数器
		CTUD	增/减计数器

1. 增计数器 CTU

表 8-4 增计数器指令参数

名称	定义	数据类型	说明
CU	输入变量	BOOL	直到 PV 值到达上限，每有一个上升沿，输入计算值加 1。
R	输入变量	BOOL	如果为真，CV 将会重置为 0。
PV	输入变量	INT	上限为 CV 的增量，这意味着 CV >>=PV 时 Q 为 TRUE 。
Q	输出变量	BOOL	如果 CV 大于等于 PV 给出的界限，就得到 TRUE。

CV	输出变量	INT	包含要增加的值，如果 RESET 为 TRUE;等到初始值为 0；直到到达 65535（16#FFFF）最大值，当 CU 是上升沿，就会上升到 1；如果 CV >= PV, Q 将会设置为 TRUE。

当计数器输入端 CU 的信号从状态“0”变为状态“1”时，当前计算值加 1，并通过输出端 CV 进行显示，第一次调用时（复位输入 R 信号状态为“0”），输入 PV 端的计数为默认值，当计数达到上限 32767 后，计数器将不会再增加，CU 也不会再起作用。

当复位输入端 R 的信号状态为“1”时，计数器的 CV 和 Q 都为“0”，只要输入端 R 状态为“1”，上升沿对 CU 就不再起作用。当 CV 值大于或等于 PV 时，输出端 Q 为“1”。此时 CV 仍可继续累加，输出端 Q 继续为输出“1” 。

增计数器 CTU 指令示例和时序图如下图所示。

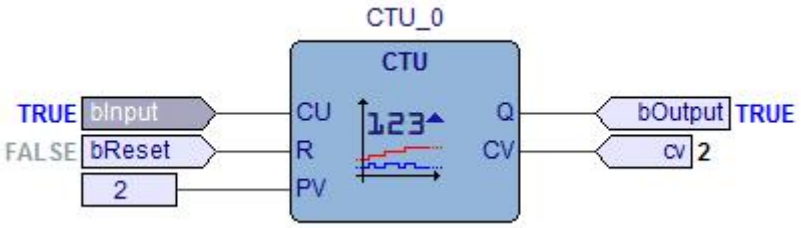


图 8.3 增计数器 CTU 使用举例

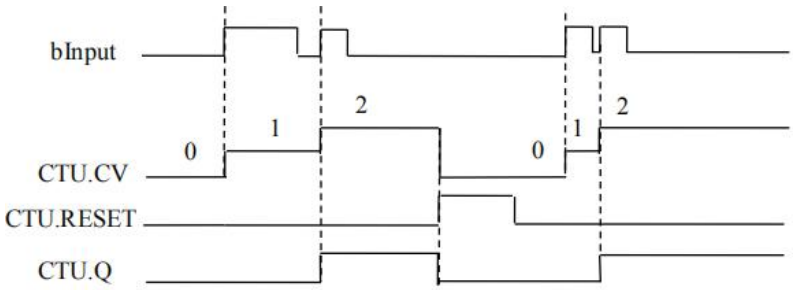


图 8.4 增计数器 CTU 特性时序图

增量功能块。输入变量 CU 和复位 RESET 以及输出变量 Q 是布尔类型的，输入变量 PV 和输出变量 CV 是 WORD 类型。

CV 将被初始化为 0 ，如果复位 RESET 是 TRUE 真的。如果 CU 有一个上升沿从 FALSE 为 TRUE ，CV 提升 1，Q 将返回 TRUE，如此 CV 将大于或等于上限 PV。

2. 减计数器 CTD

表 8-5 减计数器指令参数

名称	定义	数据类型	说明
CD	输入变量	BOOL	每有一个上升沿，计数器值（CV）将递减 1，直到 0。
LD	输入变量	BOOL	如果是 TRUE，CV 将会设置成 PV 给的初始值。
PV	输入变量	INT	初始值为 CV 的减少量。
Q	输出变量	BOOL	如果 CV 是 0 则得到 TRUE。
CV	输出变量	INT	直到到 0，从 PV 开始递减 1。

当减计数器输入端的 CD 信号从“0”变为状态“1”时，当前计数值减 1，并在输出端上 CV 显示当前值，第一次调用时（需要将加载输入端信号 LD 初始化，需要将其从“0”变为状态“1”，再变为状态“0”后功能块才能生效），输入 PV 端的计数为默认值，当计数达到 0 后，计数值将不在会减少，CD 也不再起作用。

当加载输入端信号 LD 为“1”时，计数值将设定成 PV 默认值，只要加载输入端信号 LD 状态为“1”，输入端的 CD 上升沿就不起作用。当 CV 值小于或等于 0 时，输出端 Q 为“1”。减计数器 CTD 指令示例和时序图如下图所示。

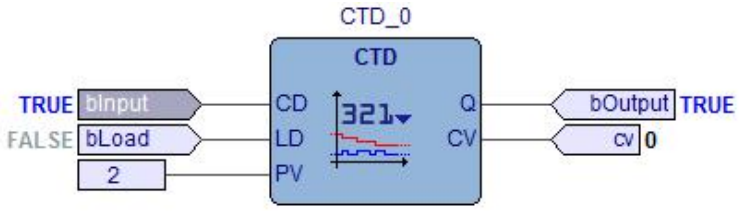


图 8.5 减计数器 CTD 使用举例

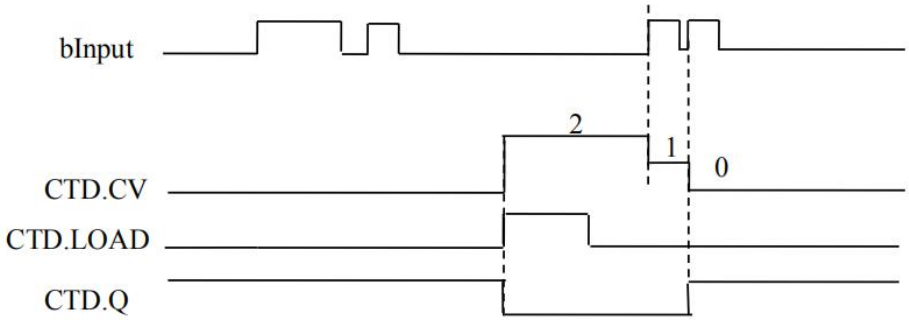


图 8.6 减计数器 CTD 特性时序图

3. 增/减双向计数器 CTUD

表 8-6 减计数器指令参数

名称	定义	数据类型	说明
CU	输入变量	BOOL	直到到达 PV 值，输入每有一个上升沿，计算值 CV 加 1。
CD	输入变量	BOOL	直到到达 PV 值，输入每有一个上升沿，计算值 CV 减 1。
R	输入变量	BOOL	如果是真，CV 将会设置到 0。
LD	输入变量	BOOL	如果是真，CV 将会设置到 PV。
PV	输出变量	INT	上限和相应的的起始值递增或递减 CV。
QU	输出变量	BOOL	当 CV 已经增加到 >= PV 时，返回 TRUE。
QD	输出变量	BOOL	当 CV 已经减少到 0 时，返回 TRUE。
CV	输出变量	INT	包含增加或者减少的值；如果 RESET 为 TRUE，将会初始化为 0；当 CU 得到一个上升沿从 FALSE 到 TRUE 将会上升 1；当 CD 得到下降沿从 TRUE 到 FALSE,，将会下降 1.这样将不

			会导致值低于 0。

当加计数输入端的 CU 信号从“0”变为状态“1”时，当前计数值加 1，并在输出 CV 上线时。当减计数输入端的 CD 的信号状态从“0”变为状态“1”时，当前计数值减 1，并在输出端 CV 上显示。如果两个输入端都是上升沿，当前计数值将保持不变。

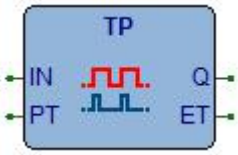
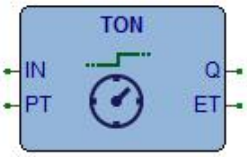
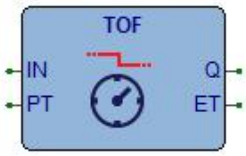
当计数值达到上限值 32767 后，加计数输入端 CU 的上升沿不再起作用。因此即使加计数输入端 CU 出现上升沿，及数值也不会增加。同理，当计数值达到下极限值 0 后，减计数输入端 CD 也不会在其作用，因此，即使减计数输入端 CD 出现上升沿，计数值也不会减少。

当 CV 值大于或等于 PV 值时，输出 QU 为“1”。当 CV 值小于或等于 0 时，输出 QD 为“1”。

8.1.3 定时器

定时器采用 IEC61131-3 标准的定时器，分为脉冲定时器 TP、通电延时定时器 TON、断电延时定时器 TOF 和实时时钟。

表 8-7 定时器指令的图形化与文本化指令表

定时器指令	图形化语言	文本化语言	说明
定时器		TP	脉冲定时器
		TON	通电延时定时器
		TOF	断电延时定时器

定时器时序图如下图所示：

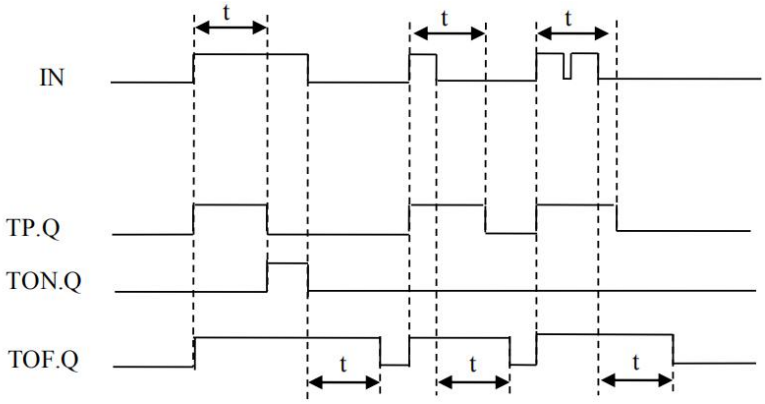


图 8.7 定时器特性时序图

1. 脉冲定时器 TP

功能：脉冲定时。

语法：在定时器的输入端 IN 从“0”变为“1”时，定时器则启动，无论定时器输入端 IN 如何变化，定时器的实际运行时间都是用户所定义的 PT 时间，在定时器运行时，其输出端 Q 的输出信号为“1”。

输出端 ET 为输出端 Q 提供定时时间。定时从 T#0s 开始，到设置的 PT 时间结束。当 PT 时间到时，ET 会保持定时时间直到 IN 变为“0”时。如果在达到 PT 定时时间之前输入 IN 已经变成“0”，输入 ET 编程 T#0s，PT 定时的时刻。为了复位该定时器，只需要设置 PT=T#0s 即可。TP 定时器参数表如下所示。

表 8-8 TP 定时器指令参数

名称	定义	数据类型	说明
IN	输入变量	BOOL	一个上升沿将启动 ET 端计时器
PT	输入变量	TIME	定时时间
Q	输出变量	BOOL	时间计时到 ET 输出为 TRUE
ET	输出变量	TIME	当前时间状态

2. 通电延时定时器 TON

功能：通电延时定时。

在定时器的输入端 IN 从“0”变为“1”时，定时器则启动，当到达定时时间 PT 且输入端的信号 IN 始终维持在“1”时，其输出端 Q 的输出信号为“1”，如果在定时器的定时时间到达之前，输入端 IN 信号由“1”变为“0”时，则定时器复位，下一个 IN 信号的上升沿定时器重启。

输出端 ET 提供定时时间，延时从 T#0s 开始，到设置的 PT 时间结束。PT 到达时，ET 将会保持定时时间直到 IN 变为“0”为止。如果在达到 PT 定时时间之前，输入 IN 变为“0”，输出 ET 立即变为 T#0s。为了重启定时器，可以设置 PT=T#0s，也可以将 IN=FALSE。TON 定时器参数表如下所示。

表 8-9 TON 定时器指令参数

名称	定义	数据类型	说明
IN	输入变量	BOOL	上升沿启动 ET 端定时
PT	输入变量	TIME	定时 ET (延迟时间) 的上限
Q	输出变量	BOOL	一旦 ET 到达显示时间 PV 输出将会变为 TRUE (定时时间结束)
ET	输出变量	TIME	当前延时定时器状态

3. 断电延时定时器 TOF

功能：断电延时定时。

在定时器的输入端 IN 从“0”变为“1”时，定时器的 Q 输出信号为“1”，定时器的启动输入端变为“0”时，定时器则启动，只要当定时器在运行，其输出 Q 一直为“1”，当到达定时时间时，输出端 Q 复位，在到达定时时间之前，如果定时器的输入端返回为“1”，则定时器复位，输出端的 Q 输出信号保持为“1”。

输出端 ET 提供定时时间，延时从 T#0s 开始到设置的定时时间 PT 结束。当 PT 时间到时，ET 将保持定时时间直到输入 IN 返回“1”为止。如果在达到 PT 定时时间之前，输入 IN 变为“1”，输出 ET 立即变为 T#0s。为了复位定时器，可以设置将 PT=T#0s。TOF 定时器参数表如下所示。

表 8-10 TOF 定时器指令参数

名称	定义	数据类型	说明
IN	输入变量	BOOL	下降沿启动 ET 端计时。
PT	输入变量	TIME	ET 端计时上限值 (延迟时间)。
Q	输出变量	BOOL	一旦 ET 到达上限值 PV 那么将会得到一个下降沿(延时时间结束)
ET	输出变量	TIME	当前延时定时器状态

4. 实时时钟

功能：修改系统时钟，返回当前日期和时间。

(1) 读取当前系统时间：首先定义一个读取系统时间的变量 time，数据类型设置为 SYSTEM_DATE_TIME，然后在程序中给 time 变量赋值（time:=SYSTEM_DATE_TIME;）。

(2) 修改系统时间：调用功能块 sysDateTimeModify

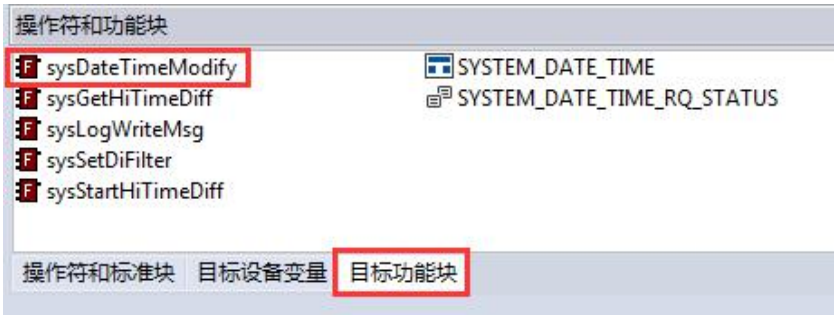


图 8.8 调用目标功能块

表 7-11 sysDateTimeModify 指令参数

名称	定义	数据类型	说明
year	输入变量	UINT	年
month	输入变量	USINT	月
day	输入变量	USINT	日
hours	输入变量	USINT	时

minutes	输入变量	USINT	分
seconds	输入变量	USINT	秒

8.2 库

8.2.1 导入库

点击窗口右侧库的树节点下的第一个按钮，如下图所示。

图 8.9 导入库

在弹出的“工程库列表”对话框中点击右侧的“添加”按钮。

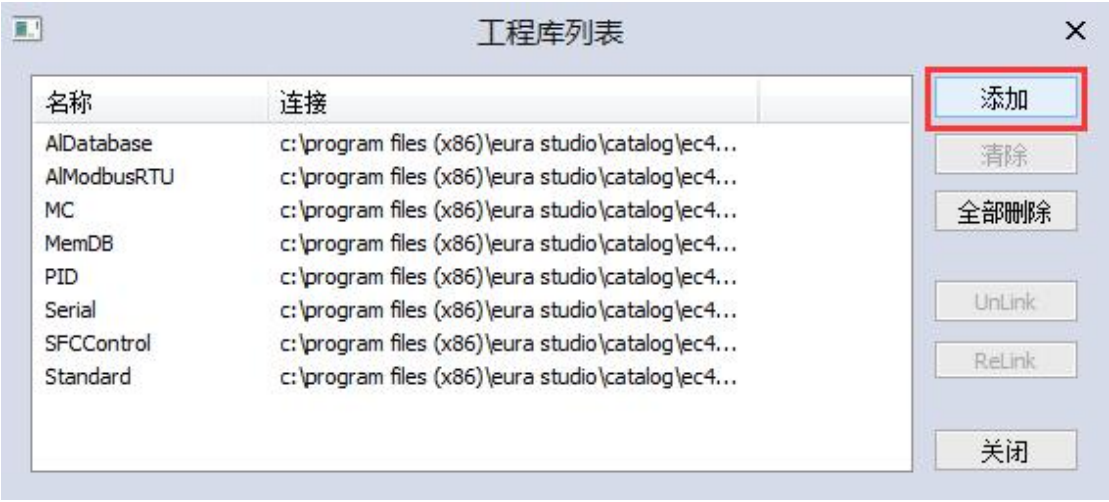


图 8.10 添加库

在弹出的对话框筛选文件类型一栏中选择“ALL PLC library files”，选择需要安装或者更新的文件，点击打开，系统会自动安装或更新。

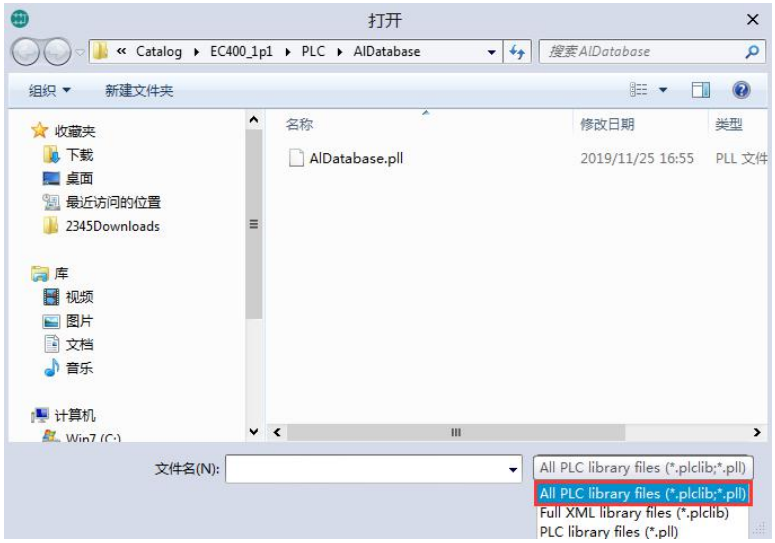


图 8.11 选择库文件

8.2.2 刷新库

导入库后，点击窗口右侧库的树节点下的第二个按钮，刷新系统中的库。

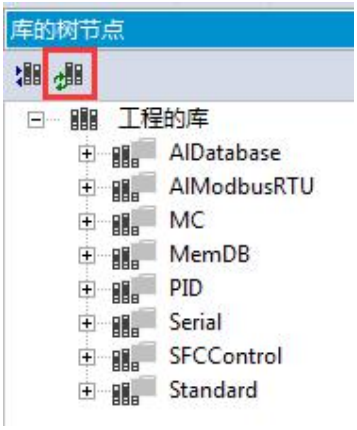


图 8.12 刷新库

8.2.3 工程的库

PLC 支持的库函数如下所示：

图 8.13 工程的库

8.3 加密

8.3.1 POU 加密与解密

EURA Studio 软件提供了密码保护功能，允许用户对 POU（程序、功能块、函数）进行加密。若 POU 被加密，则用户就不能查看该 POU 的内容。用户若想查看加密的 POU，首先要对该 POU 进行解密。密码长度至少 4 位，密码可以由数字、字母、下划线组成。

1. 加密

选中要加密的 POU，鼠标右键，在弹出的右键菜单中选择“加密”，弹出“加密 POU”对话框，在密码框和确认密码框中输入密码，两次输入的密码必须一致，点击“确定”按钮完成加密，加密后的 POU 会带有“  ”图标。

图 8.14 加密 POU

选中项目，鼠标右键，在弹出的右键菜单中选择“加密所有对象”，弹出“加密 POUS”对话框，在密码框和确认密码框中输入密码，两次输入的密码必须一致，点击“确定”按钮，实现对项目中所有的 POU 进行加密。

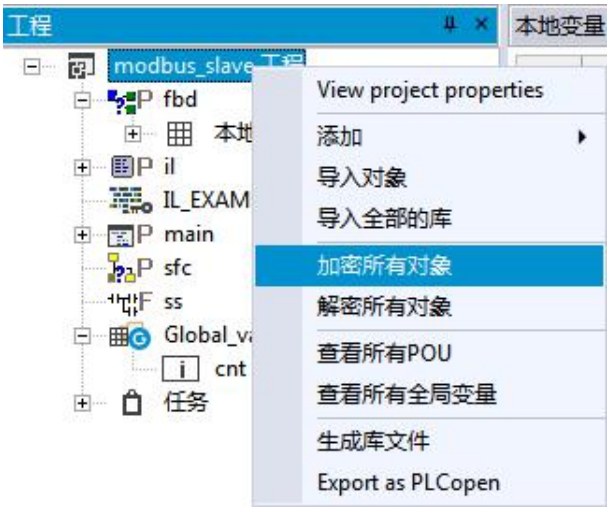


图 8.15 加密所有对象

2. 解密

选中要解密的 POU，鼠标右键，在弹出的右键菜单中选择“解密”，弹出“解密 POU”对话框，在密码框中输入密码，点击“确定”按钮完成解密。

图 8.16 解密 POU

选中项目，鼠标右键，在弹出的右键菜单中选择“解密所有对象”，弹出“解密 POUS”对话框，输入密码，点击“确定”按钮，实现对项目中所有的 POU 进行解密。

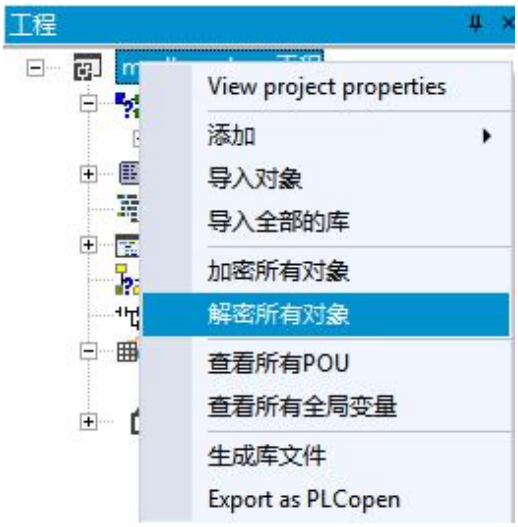


图 8.17 解密所有对象

8.3.2 工程上载与密码

Eura Studio 软件支持工程上载时需进行密码验证，在程序下载过程中用户可以根据自身需要进行密码设置。**注：下载耗时与下载量有关，建议调试过程中只下载 bin 文件，待调试完成后下载 cod 文件。**

1、 工程下载加密

在“工程”菜单中，选择“工程选项”，在弹出的对话框中选择“下载”页。用户选择是否使用密码保护，可以增加密码。

源码中“下载时间”选项，用户可以根据实际情况进行选择：

1）从不：表示程序下载过程中只下载 PLC 的 bin 执行文件，此选项不支持上载源码。

2）在 PLC 程序下载时：表示程序下载过程中下载 bin 执行文件与 cod 源码文件，此选项支持源码上载。

3）在断开连接时：表示程序下载过程中只下载 PLC 的 bin 执行文件，当用户断开连接时，PLC 自动检测 bin 文件与 cod 文件是否一致，提醒用户进行源码下载。此选项支持源码上载。

图 8.18 工程下载配置

2、 工程上载

EC400 系列 PLC 支持用户现场上载工程，在“文件”菜单中选择“从目标设备导入工程”，选择目标设备。点击“配置连接”配置连接。在对话中点击“Upload Sources”，进行上载操作，当工程下载时用户配置密码时，上载结束后会弹出获取密码对话进行密码验证。密码验证通过后，弹出保持文件设置对话框进行上载工程保存。

图 8.19 工程上载配置

图 8.20 工程上载配置

图 8.21 密码验证

8.4 掉电保持

EC400 提供了 2 种数据保持功能，Retain 变量保持机制和使用 MemDB 功能块。在 PLC 掉电或者停止运行时，将数据保持区数据内存保存到数据保持存储区，其支持 Retain 型实时掉电保持机制，其设定数量为 508 个字节，而对于系统设置、参数设置等数据，支持存储数据功能块，其最大存储量为 24K。在 PLC 上电时，将保存到掉电存储区的数据复制到相应内存区中。在程序中通过调用库函数 MemDB_Write 将数据写入内存中，PLC 重新上电时，通过调用库函数 MemDB_Read 将内存中的数据读出，调用 MemDB_Write 和 MemDB_Read 的程序，必须配置在 Background 任务中。操作内存的函数库如下图所示：

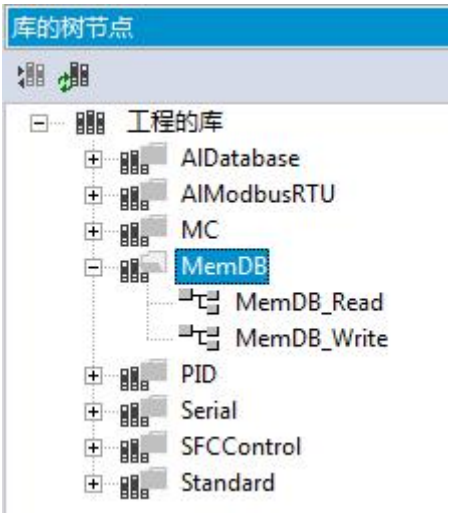


图 8.22 读写内存库函数

实现掉电保持的功能如下：

1. 创建结构体变量 paras。

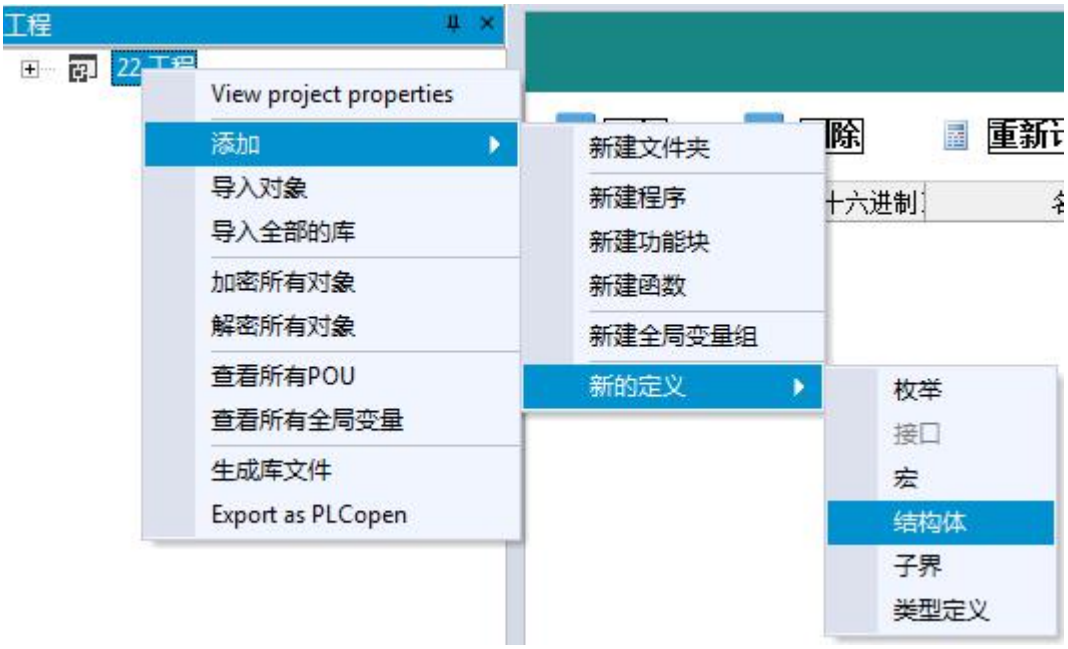
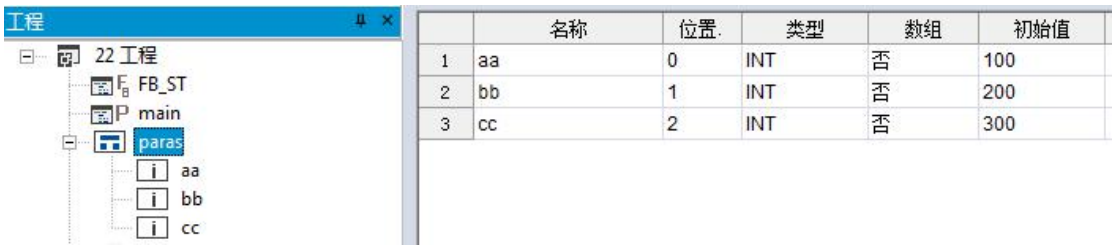


图 8.23 创建结构体

2. 添加结构体变量成员



	名称	位置.	类型	数组	初始值
1	aa	0	INT	否	100
2	bb	1	INT	否	200
3	cc	2	INT	否	300

图 8.24 添加结构体变量

3. 编写程序，并将程序配置在 Background 任务中，在线调试，在没有触发 test 参数之前，触发 MemDB_Read 的 Execute 为 1，读出结构体中的变量为默认的初始值。

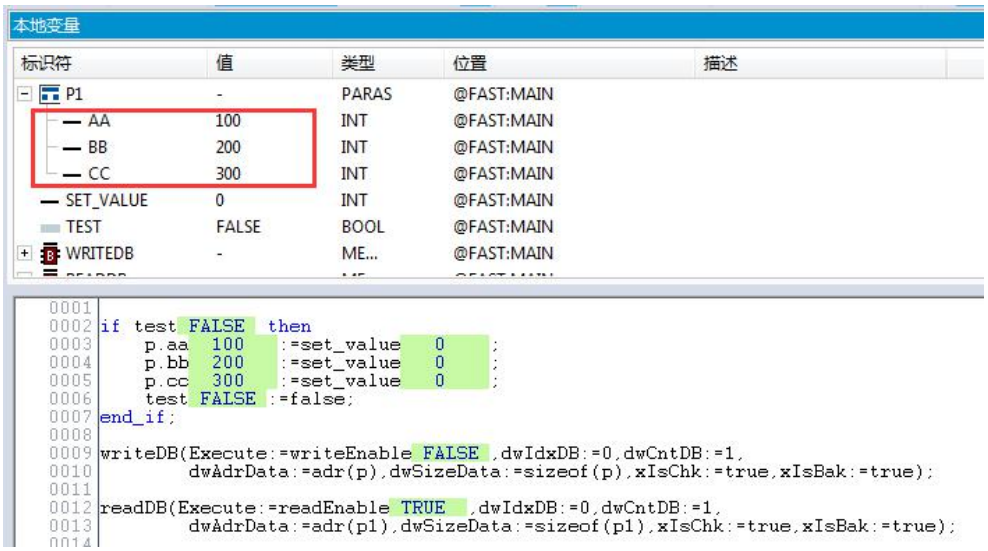


图 8.25 程序调试 1

强制写入 set_value 并触发 test 变量来修改结构体中的变量。

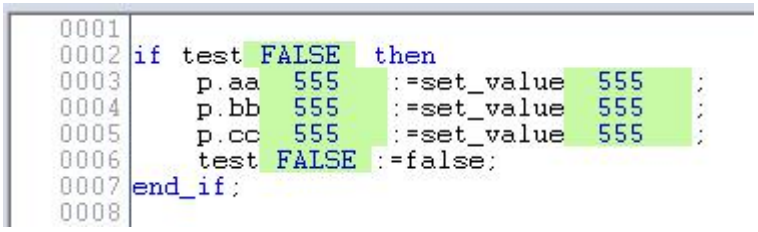


图 8.26 程序调试 2

触发 MemDB_Write 和 MemDB_Read 操作，Execute 设置为 1 ：

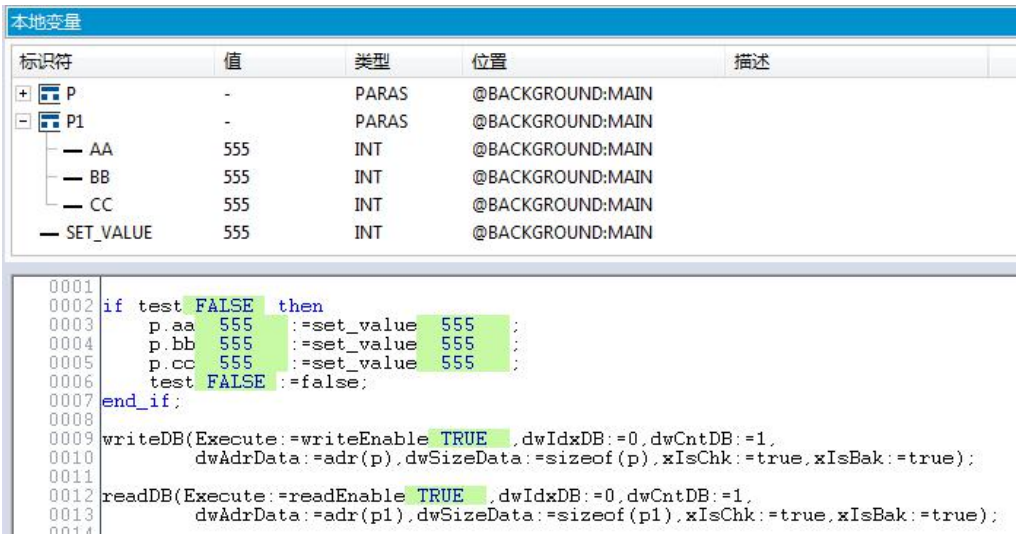


图 8.27 程序调试 3

将 MemDB_Write 和 MemDB_Read 的 Execute 设置为 0，重新上电，再次设置

MemDB_Read 的 Execute 为 1，读出的值为写入的数值。

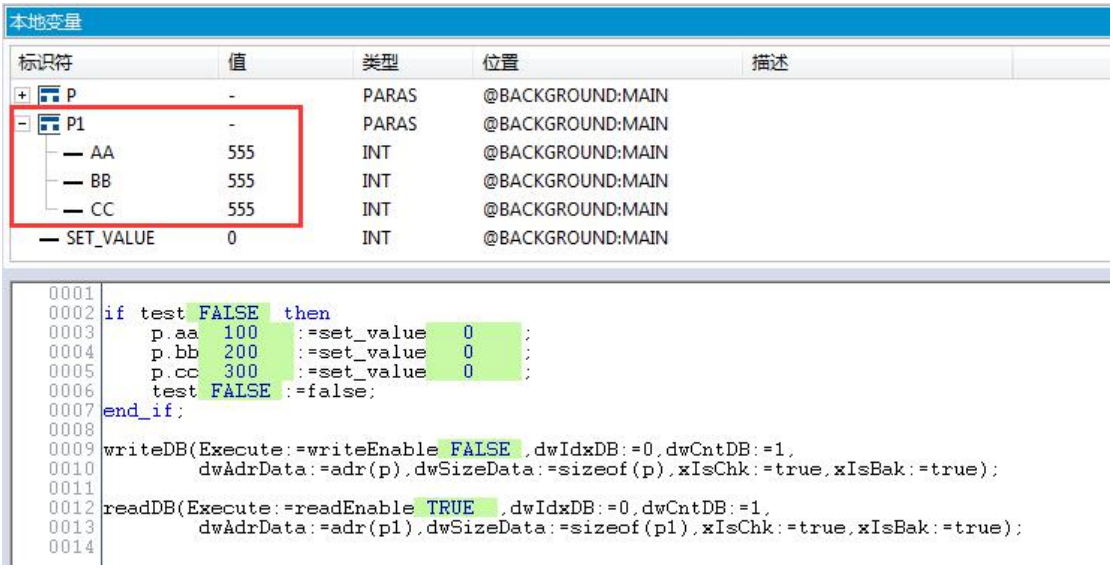


图 8.28 程序调试 4

8.5 在线诊断

在线诊断页面涵盖了 ExtBus、DI、高速输入、高速输出和伺服轴的相关参数配置，用户在调试程序时，连接 PLC 后可以在“在线诊断”页面点击“读取 Ram 数据”按钮读取参数的值，帮助用户排查问题。同时，也可以在这里修改某些参数的值，然后点击“更新参数”将多个修改的参数写入到 PLC。也可以在每行的操作列点击“Update”或“Read”更新或读取一个参数。参数配置可以点击“Ref”参考设置。

图 8.29 在线诊断操作

在线诊断参数说明如下表所示：

表 8-1 在线诊断参数说明

名称	说明
Extbus Enable	Extbus 使能，使能后才能添加扩展模块
Extbus Delay	PLC 上电初始化时，Extbus 延时扫描时间
Extbus Status	Extbus 状态
Extbus Act	Extbus 时间轮询时间
Extbus Retry	Extbus 发生错误后重试次数

Module Num	Extbus 扩展模块数据
Module Type	第 1~8 个模块，每个模块的类型 DI:1016 DO:2016 AI:3004 AO:4004
Module Configuration	第 1~8 个模块，每个模块的配置
DI Configuration	DI 滤波时间
HSC0 Enable	HSC0 使能
HSC0 Type	HSC0 类型 0：计数 1：频率 2：周期
HSC0 Mode	HSC0 模式 0：单脉冲 1：双脉冲 2：AB 正交 3：AB4 倍频正交
HSC0 Diretion	HSC0 计数方向 0：正向计数 1：反向计数
HSC0 Aux	HSC0 方向控制 0：使用内部方向 1：使用外部方向
HSC1 Enable	HSC1 使能
HSC1 Type	HSC1 类型 0：计数 1：频率 2：周期
HSC1 Mode	HSC1 模式 0：单脉冲 1：双脉冲 2：AB 正交 3：AB4 倍频正交
HSC1 Diretion	HSC1 计数方向 0：正向计数 1：反向计数
HSC1 Aux	HSC1 方向控制 0：使用内部方向 1：使用外部方向
PTO0 Enable	PTO0 使能
PTO0 Type	PTO0 类型 0：PWM 1：脉冲加方向 2：双脉冲 3：AB 正交 4：AB 正交 4 倍频
PWM0 Time Base	PWM0 时间基准
PWM0 Duty Unit	PWM0 占空比单位
PWM0 Period	PWM0 周期
PWM0 Duty	PWM0 占空比
PTO1 Enable	PTO1 使能
PTO1 Type	PTO1 类型 0：PWM 1：脉冲加方向 2：双脉冲 3：AB 正交 4：AB 正交 4 倍频
PWM1 Time Base	PWM1 时间基准
PWM1 Duty Unit	PWM1 占空比单位
PWM1 Period	PWM1 周期
PWM1 Duty	PWM1 占空比
AXIS0 Type	AXIS0 类型
AXIS0 Channel Finite	轴 0 输出通道选择

AXIS1 Type	AXIS1 类型
AXIS1 Channel Finite	轴 1 输出通道选择

在程序中添加两个扩展模块，PLC 不接扩展模块，下载程序到 PLC 后，run 和 error 指示灯点亮。点击配置树中的在线诊断节点，在线诊断界面中的 ExtBus 参数变红色了。

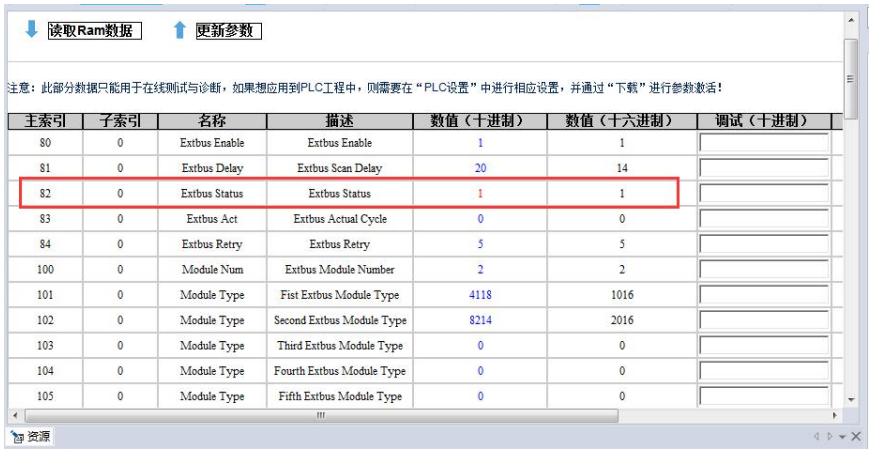


图 8.30 在线诊断界面

通过 extbus status 设置可知，是扩展模块数据与程序配置不一致。

表 8-2 extbus status 参数含义

序号	代码	说明
1	0	未启动
2	1	扩展模块数量与配置不一致
3	0x1000	配置正确
4	0x001x	配置数量正确，但是 x 节点的类型不对
5	0x106x	配置正确，在运行过程中 x 节点异常
6	0x2x	x 节点参数错误

8.5 工程资源发布与资源重利用

1、当 PLC 工程设计完成后，需要进行工程转发与共享时，可以利用菜单栏中“工程”->生成可发布的源码块。

图 8.31 工程发布界面

2、其支持密码保护。

图 8.32 工程发布加密界面

3、确认后，导出代码存放在工程目录下。

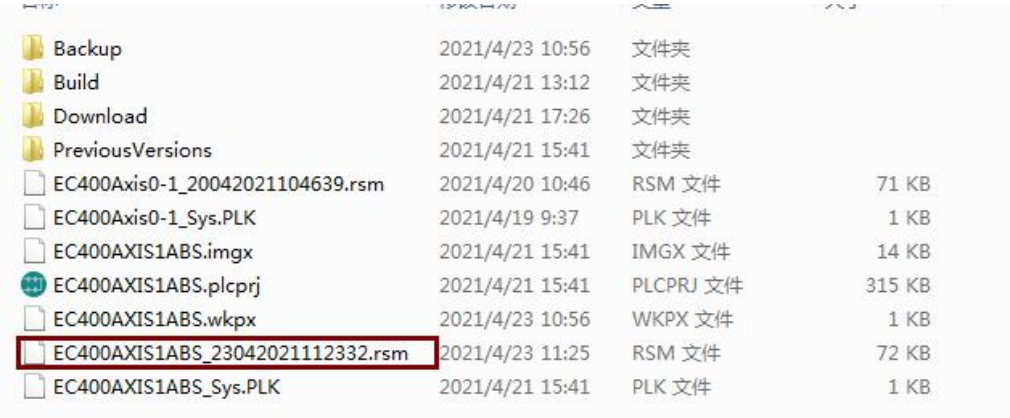


图 8.33 工程发布文件

4、此文件可以单独共享给其他人。

5、当用户接收到.rsm 文件后，利用菜单栏中“文件”->打开工程，在文件筛选框中选择所有文件，选择.rsm 文件。

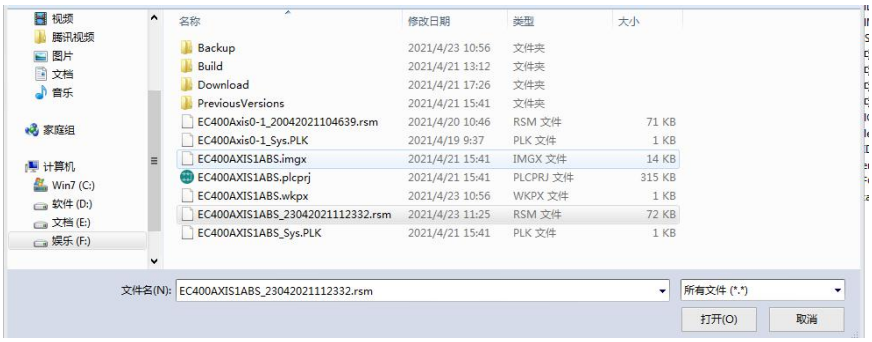


图 8.34 导入 rsm 文件

6、此时，如果工程中设置密码则提醒输入密码。

图 8.35 输入密码

7、设定工程解压缩位置。

图 8.36 选择位置

8、此时，解压缩完成，但是此时工程编译会提示存在错误。

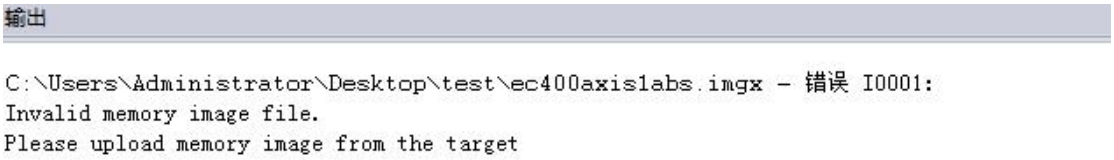


图 8.37 错误提醒

9、这个错误主要由于工程缺少资源 img 文件，具体步骤：工程在线，编程软件自动完成资源文件上载。

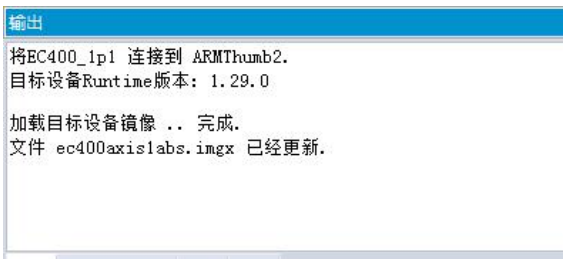


图 8.38 完成资源上载

10、此时，工程编译成功。

11、当用户想利用其它工程中的资源时，除了通过库文件进行资源共享外，还可以利用工程->导入对象，选择想要导入资源所在的工程。

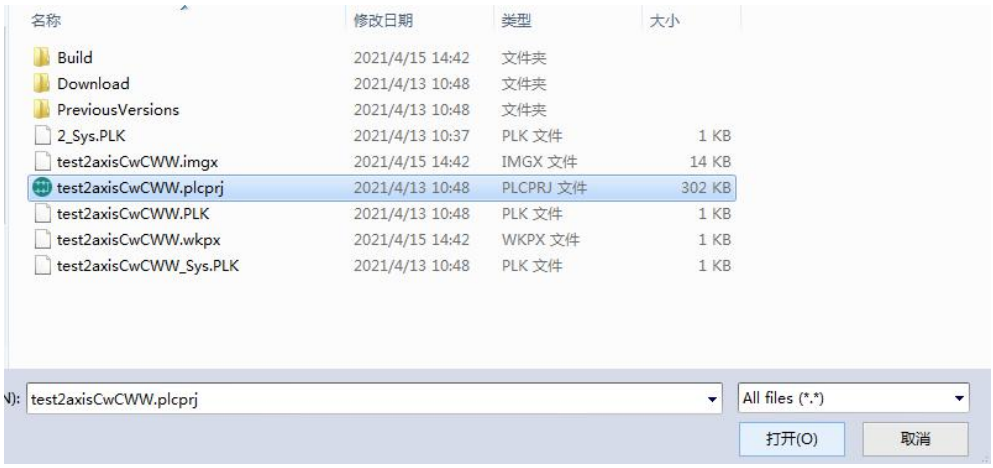


图 8.39 选择资源所在工程

12、确认后，选择想要导入资源，其包括了功能块、程序、变量。

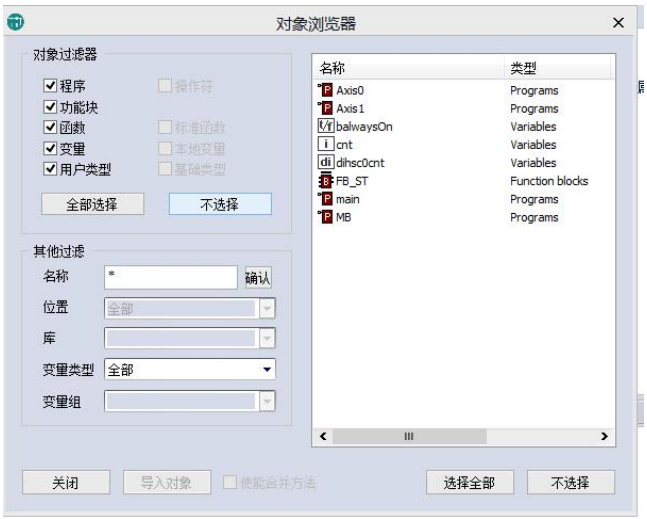


图 8.40 选择资源

13、选择资源后，点击“导入对象”，资源即可导入当前工程中。

附录 A 常用快捷键

常用指令快捷键			
指令	图标	快捷键	描述
打开工程资源窗口		Ctrl+W	显示/隐藏工程资源窗口
库窗口		Ctrl+L	显示/隐藏库资源窗口
输出窗口		Ctrl+R	显示/隐藏输出窗口
示波器窗口		Ctrl+K	显示/隐藏示波器窗口
观察窗口		Ctrl+T	显示/隐藏观察窗口
全屏		Ctrl+U	切换编辑区全屏显示
编译		F7	编译工程文件
编译全部		Ctrl+Alt+F7	重新编译工程
下载		Ctrl+F5	编译修改工程，将工程代码下载到设备中
增加/移除断点		F12	增加/移除断点
添加并联支路		Shift+P	在选择触点前添加并联支路
添加触点		Shift+C	在选择触点后添加触点
添加线圈		Shift+O	在选择的网络添加线圈
功能块		Shift+B	在选择触点后添加功能块
常量		Shift+K	在选择功能触点处添加数据常量
返回		Shift+R	在网络中添加返回操作
跳转		Shift+J	在网络中添加跳转操作
变量		Shift+V	在选择功能块触点处添加变量
表达式		Shift+E	在选择功能块触点处添加表达式
注释		Shift+M	在选择网络中添加批注
开启		O	触点/线圈状态为开启状态
取反		C	触点/线圈状态为常闭状态
正向		P	触点/线圈状态为开启状态

反向		N	触点/线圈状态为常闭状态
线圈置位		S	线圈置位
线圈复位		R	线圈复位
增加 PIN		Ctrl+″+″	功能块添加 PIN
减少 PIN		Ctrl+″-″	功能块减少 PIN
使能 EN/ENO PIN		E	功能块添加 EN/ENO PIN

敬告用户：

感谢您选用我司产品，为保证您正确使用本产品及得到我司最佳售后服务，请认真阅读下述条款，并做好相关事宜。

只有具备一定的电气知识的操作人员才能够对本产品进行接线、上电操作；手册中示例程序仅供参考，不保证其实用性。

本公司致力于产品的不断改善和升级，手册提供资料如有变更，恕不另行通知，请自行访问本公司网站获取。

产品保修范围：按使用要求正常使用情况下，所产生的故障。

产品保修期限：本公司产品的保修期为自出厂之日起，十二个月以内。保修期实行长期技术服务。

非保修范围：任何违反使用要求的认为意外、自然灾害等原因导致的损坏，以及未经许可而擅自对产品拆卸、改装及修理的行为，视为自动放弃保修服务。

从中间商处购入产品：凡从经销代理商处购买产品的用户，在产品发生故障时，请与经销商、代理商联系。

免责条款：因下列原因造成的产品故障不在厂家 12 个月免费保修服务范围之内：

（1）、厂家不依照《产品手册》中所列程序进行正确的操作；

（2）、用户未经与厂家沟通自行修理产品或擅自改造产品；

（3）、因用户环境不良导致产品器件异常老化或引发故障；

（4）、因用户超过产品的标准范围使用产品；

（5）、由于地震、火灾、风水灾害、雷击、异常电压或其他自然灾害等不可抗力的原因造成的产品损坏；

（6）、因购买后由于人为摔落及运输导致硬件损坏。

责任：无论从合同、保修期、疏忽、民事侵权行为、严格的责任、或其他任何角度讲，EURA 和他的供货商及分销商都不承担以下由于设备所造成的特殊的、间接的、继发的损失责任。其中包括但不仅仅局限于利润和收入的损失，使用供货设备和相关设备的损失，资金的花费，代用设备的花费，工具费和服务费，停机时间的花费，延误，及购买者的客户或任何第三方的损失。另外，除非用户能够提供有力的证据，否则公司及它的供货商将不对某些指控如：因使用不合格原材料、错误设计、或不规范生产所引发的问题责任。

解释权归欧瑞传动电气股份有限公司。

如果您对 EURA 的产品还有疑问，请与 EURA 公司或其办事处联系。技术数据、信息、规范均为出版时的最新资料，EURA 公司保留部事先通知而更改的权利，并对由此造成的损失不承担任何责任。解释权归 EURA 公司。

20210910001