



雷赛中大型 PLC 产品

Leadsys Studio 编程及应用手册



MC500



LC1000



LC5000

- ◆ 非常感谢您本次购买雷赛产品
- ◆ 使用前请详细阅读此说明书，正确使用产品
- ◆ 请妥善保管此说明书

www.leisai.com

202505 V1.07

前言

感谢您选用深圳市雷赛智能控制股份有限公司中大型 PLC 产品。本手册提供了雷赛智能中大型 PLC 的编程及应用说明。对于初次使用的用户，请认真阅读本手册。若对产品的功能应用和性能方面有所疑惑，请咨询我司技术支持人员以获得帮助。

由于产品的改进，手册内容可能持续更新。

技术热线：400-885-5501

版本说明：

- (1) 禁止转载本书的部分或者全部内容。
- (2) 本书内容有可能变更，恕不另行通知。

目录

前言	1
目录	2
第一章 概述	11
1.1. 产品简介	11
1.1.1. 大型 PLC 产品介绍	11
1.1.2. 中型 PLC 产品介绍	13
1.2. LeadSys 软件介绍	15
1.2.1. 软件的获取与安装	16
1.2.2. LeadSys 卸载	17
1.3. 系统应用架构	18
1.3.1. 大型 PLC 系统应用架构	18
1.3.2. 中型 PLC 系统应用架构	20
1.3.3. 基本使用流程	23
第二章 快速入门	24
2.1. 编程环境及配置	24
2.2. 编写程序的典型步骤	25
2.3. 新建工程	26
2.4. 电脑与 PLC 的连接	28
2.4.1. 电脑网口 IP 设置并连接 PLC	28
2.4.2. 登录及运行 PLC	29
2.4.3. PLC 的 IP 地址读取和修改	31
2.5. 设备组态	31
2.5.1. 自动扫描设备组态	32
2.5.2. 离线设备组态	35

2.6. LC5000 控制器例程	38
2.6.1. LC5000 实验系统硬件	38
2.6.2. 用 LC5000 和 IO 模块控制 LED 灯的开关	39
2.6.3. 用 LC5000 控制总线型步进电机运动	48
2.7. PMC600 控制器例程	54
2.7.1. PMC600 实验系统硬件	54
2.7.2. 电机接口及参数的设置	54
2.7.3. 编写控制电机运动的程序	57
2.8. MC500 控制器例程	64
2.8.1. MC500 实验系统硬件	64
2.8.2. 用 LT2070E 和 MC500 控制总线型伺服电机运动	65
2.8.3. 用 MC500 和 EM32DX-C1 控制 IO 模块上的输出灯	79
第三章 编程基础 1：变量与函数	86
3.1. 直接地址	86
3.1.1. 直接地址定义	86
3.1.2. 直接地址范围	88
3.1.3. Modbus 地址范围	89
3.2. 变量	91
3.2.1. 变量定义	91
3.2.2. 变量类型	92
3.2.3. 数据类型	94
3.2.4. 数据单元类型	96
3.2.5. 掉电保持变量	101
3.2.6. 全局变量和局部变量	103
3.3. 配方	105
3.4. 新建 POU	110
3.5. 功能块与函数	112
3.5.1. 功能块	112

3.5.2. 函数	115
第四章 编程基础 2：编程语言	120
4.1. LeadSys 支持的编程语言类型	120
4.2. 指令表 (IL)	120
4.3. 结构化文本 (ST)	124
4.3.1. 表达式	125
4.3.2. 操作符	126
4.3.3. 操作数	127
4.3.4. 语句	127
4.3.5. 调用功能块	135
4.3.6. 注释	137
4.4. 梯形图 (LD)	137
4.4.1. 梯形图元素	138
4.4.2. 工具箱	142
第五章 任务配置	144
5.1. 添加任务	144
5.2. 任务设置	145
5.3. 任务执行	147
5.3.1. 任务优先级	147
5.3.2. 任务执行周期	149
5.4. POU 调用	150
5.4.1. 在任务配置中添加 POU 调用	150
5.4.2. 运动控制程序的任务配置	151
5.4.3. 判断 POU 是否被调用	153
5.4.4. POU 执行顺序	153
第六章 运动控制功能	155
6.1. 概述	155
6.1.1. 运动控制基本框架	155

6.1.2. 运动控制过程	155
6.1.3. PLCOPEN 状态机	156
6.1.4. 控制模式介绍	158
6.2. 轴组态	165
6.3. 本地脉冲轴设置	166
6.3.1. 基本参数设置	166
6.3.2. 运动程序编写	168
6.4. EtherCAT 轴设置	174
6.4.1. 通用参数的设置	175
6.4.2. 单位换算	179
6.4.3. 缩放/映射设置	183
6.4.4. 原点参数设置	184
6.5. 轴常用变量	185
6.6. 在线监控	190
6.7. 运动功能及指令	192
6.8. 单轴运动	192
6.9. 回零运动	200
第七章 连续插补轴组运动	204
7.1. 轴组的概念	204
7.2. 运动学模型的介绍	204
7.2.1. TRAFO.Kin_Gantry2（两轴运动学变化）	204
7.2.2. TRAFO.Kin_Gantry3（三轴运动学变化）	206
7.3. 轴组使用和建立方式	208
7.4. 轴组插补指令的介绍和使用	214
7.4.1. 轴组使能 SMC_GroupPower	218
7.4.2. 启用轴组 MC_GroupEnable	220
7.4.3. 坐标系转换 MC_SetCoordinateTransform	222
7.4.4. 绝对位置直线插补 MC_MoveLinearAbsolute	224

7.4.5. 绝对位置圆弧插补 MC_MoveCircularAbsolute	229
7.5. 连续插补插补中各个参数的介绍	234
7.5.1. 轴组状态机的介绍 SMC_AXIS_GROUP_STATE	234
7.5.2. 拐角过渡参数模式 MC_TRANSITION_MODE	237
7.5.3. 速度交接模式 MC_BUFFER_MODE	241
7.5.4. 圆弧插补模式 SMC_CIRC_MODE	243
7.5.5. 轴组方向模式 SMC_ORIENTATION_MOD	245
7.5.6. 工件坐标 SMC_POS_REF	246
7.5.7. 轴组位置关系 SMC_COORD_SYSTEM	246
7.5.8. 工件坐标转换成笛卡尔坐标	247
7.6. 插补子程序的编写和调用程序	248
7.6.1. 轴组初始化、启用轴组和坐标系的转换	248
7.6.2. 直线插补例程	252
7.6.3. 圆弧插补例程	256
7.6.4. 连续插补例程	261
7.7. 速度曲线和位移曲线的介绍和各个变量添加、使用方法	270
7.8. 离线仿真的方法	275
第八章 电子凸轮	276
8.1. 功能描述	276
8.2. 相关功能块	276
8.3. 数据结构	276
8.4. 电子凸轮使用流程	279
8.4.1. 电子凸轮流程图	279
8.4.2. 电子凸轮程序使用步骤	281
8.4.3. 凸轮表	287
8.4.4. 功能块参数对凸轮运行的影响	294
8.4.5. 挺杆功能	306
第九章 离线仿真	310

第十章 高速输入输出功能	312
10.1. 高速计数器简介	312
10.1.1. MC500 系列产品高速 IO 介绍	312
10.1.2. PMC600 系列产品高速 IO 介绍	313
10.1.3. LC1000 系列产品高速 IO 介绍	314
10.1.4. LC5000IO 产品高速 IO 介绍	315
10.2. 界面组态	316
10.2.1. PMC600 高速输入端口组态界面	316
10.2.2. PMC610 高速输入端口组态界面	317
10.2.3. MC500 高速输入输出端口界面	318
10.2.4. LC1000/LC5000IO 高速 IO 组态界面	319
10.3. 高速计数器	322
10.3.1. PMC600 高速计数器指令及功能	322
10.3.2. MC500 高速计数器指令及功能	323
10.3.3. LC1000/LC5000IO 高速计数器指令及功能	323
10.3.4. PMC600 实现高速计数器功能的方法	323
10.3.5. MC500 实现高速计数器功能的方法	328
10.3.6. LC1000/LC5000IO 实现高速计数器功能的方法	332
10.4. 编码器功能	336
10.4.1. PMC610 编码器指令及功能	336
10.4.2. LC5000IO 编码器功能编码器指令及功能	337
10.4.3. PMC610 编码器功能	337
10.4.4. LC5000IO 编码器功能	339
10.5. 高速锁存功能	340
10.5.1. PMC600/PMC610 高速锁存指令介绍	340
10.5.2. LC1000/LC5000IO 高速锁存指令介绍	341
10.5.3. PMC600/PMC610 高速锁存功能	341
10.5.4. LC1000/LC5000IO 高速锁存功能	344

第十一章 扩展模块	347
11.1. 扩展模块类型	347
11.2. 扩展模块组态	348
11.3. 扩展模块的配置、映射	354
11.3.1. IO 模块的配置、映射	354
11.3.2. 模拟量模块的配置、映射	357
第十二章 串口通讯	366
12.1. 基于 RS485 接口的 Modbus RTU 主从站通讯	366
12.1.1. Modbus RTU 主从站通讯基础知识	366
12.1.2. Modbus 通讯协议简介	366
12.1.3. Modbus 协议可访问的内部地址	368
12.1.4. Modbus 通信功能码	369
12.2. RS485 Modbus RTU 主站通讯	373
12.2.1. LC 系列 PLC 的 RTU 主站配置	374
12.2.2. MC 系列 PLC 的 RTU 主站配置	383
12.3. RS485 Modbus RTU 从站通讯	386
12.3.1. LC 系列 PLC RTU 从站的配置	386
12.3.2. MC 系列 PLC RTU 从站的配置	387
12.4. RS485 Modbus RTU 通讯例程	388
12.5. RS485 自由协议通讯	392
12.5.1. 自由协议通讯的使用	392
12.5.2. PMC600 专用自由协议通讯	397
12.6. RS232 Modbus RTU 主站通讯	398
12.6.1. LC 系列 PLC RS232 RTU 主站的设置	398
12.6.2. MC 系列 PLC RS232 RTU 主站的设置	399
12.7. RS232 Modbus RTU 从站通讯	400
12.7.1. LC 系列 PLC RS232 RTU 从站的设置	400
12.7.2. MC 系列 PLC RS232 RTU 从站的设置	401

12.8. RS232 自由协议通讯	401
第十三章 以太网通讯	407
13.1. Modbus TCP 通讯	407
13.1.1. Modbus TCP 主站通讯	407
13.1.2. Modbus TCP 从站通讯	416
13.2. TCP 自由协议	418
13.2.1. 库管理器添加 CAA 类型库	418
13.2.2. 使用 TCP 自由协议指令	420
13.2.3. TCP 自由协议例程	427
13.3. UDP 自由协议	449
13.3.1. UDP 指令的使用	450
13.3.2. UDP 通讯例程	453
13.4. 标签通讯	457
13.4.1. 标签通讯的流程	457
13.4.2. 标签通信例程	457
13.5. OPC UA 服务器	466
13.5.1. OPC UA 通讯的流程	466
13.5.2. PLC 配置 OPC 功能的方法	467
13.5.3. 使用 UaExpert 软件调试 OPC UA 通讯	471
13.6. Ethernet/IP 通信	475
13.6.1. 配置从站	476
13.6.2. 配置主站	479
13.6.3. 实现通讯功能	482
13.6.4. Ethernet/IP 主从站配置	483
第十四章 EtherCAT 通讯	485
14.1. EtherCAT 主站配置	485
14.1.1. 添加设备描述文件	485
14.1.2. 从站组态	488

14.1.3. 主站设置	492
14.1.4. EtherCAT 主站常用参数	493
14.2. EtherCAT 从站配置	494
14.2.1. 常规设置	494
14.2.2. 过程数据 (PDO)	499
14.3. 启动参数 (SDO)	501
14.3.1. 伺服从站添加 402 轴	504
14.3.2. 使能/失能从站	505
14.3.3. EtherCAT 从站常用参数	507
14.4. EtherCAT 冗余环网功能	508
第十五章 通过 U 盘下载 PLC 程序	511
第十六章 故障诊断	512
16.1. 程序编译错误的定位	512
16.2. PLC 程序运行异常处理	514
16.2.1. 常见的异常现象及处理措施	514
16.2.2. 因程序问题导致 PLC 失控的处理措施	521
16.3. 常见故障码	523
第十七章 LeadSys Studio 配置技巧	544
17.1. 在线帮助与联网帮助配置	544
17.2. 编码助手的配置	546
17.3. 梯形图 LD 网络注释的显示	546
17.4. 电机功能码	547

第一章 概述

1.1. 产品简介

深圳市雷赛智能控制股份有限公司（简称“雷赛智能”或“雷赛”，股票代码：002979）

中大型 PLC 产品主要包括：

- 大型 PLC 产品，LC 系列智能产线控制器
- 中型 PLC 产品，MC/PMC 系列运动控制器
- 本手册主要是针对以上中大型 PLC 产品介绍编程及应用方法。

1.1.1. 大型 PLC 产品介绍

雷赛智能产线控制器，属于大型 PLC 产品，命名 LC(Large Controller)，主要包括 LC1000\LC2000S\LC5000 系列，LC5000 系列还分为标准版和高速 IO 版两个版本。

LC 系列大型 PLC 产品是雷赛面向先进制造业推出的智能产线运动控制器。基于 CODESYS 运动控制平台研发，采用 EtherCAT 总线，最高可实现 256 轴运动控制。特别适合控制工艺复杂的生产线设备，广泛应用于光伏、锂电、半导体、包装、物流、特种机床等行业高端设备。

LC 系列控制器主要特点有：

- 1) 高速多轴：支持 250 μ s（8 轴）、500 μ s（64 轴）、1ms（256 轴）运算周期；
- 2) 精准控制：EtherCAT 总线抖动 30 μ s 以内；LC5000 等产品抖动小于 20 μ s；
- 3) 高性能：CPU：i5、RAM：128GSSD、ROM：8GDDR，为程序运算提供澎湃动力；
- 4) 功能丰富：单轴控制，电子齿轮/凸轮同步控制，轴组的直线、圆弧、螺旋线插补，CNC、机器人控制；
- 5) 冗余环网：支持 EtherCAT 冗余环网，任一从站故障不影响正常运行；
- 6) 双 EtherCAT 口：可以兼顾高响应实时轴控和 PTP 点位控制，使整个系

统做到最优；

- 7) 双 EtherNet 口：一个网口用于设备间的组网，一个网口连接信息层；IP 分开，安全隔离；
- 8) 内置 UPS 掉电保护功能、无需 UPS 接线，成本低、安全可靠；
- 9) 数据处理库可用于大数据处理、文件处理、产线数据收集与处理；
- 10) 丰富的通信互联功能：具有 Modbus TCP、OPC UA、Ethernet/IP 等协议；
- 11) LC-IO 系列本体自带高速 IO、编码器；响应速度更快，更可靠，且成本低。

LC 系列智能产线控制器的型号见表 1.1。

表 1.1 标准版智能产线控制器（大型 PLC 产品）一览表

系列	名称	型号	功能差异
LC1000	LC1016	LC1016-10004128-U0P	赛扬，16 轴，内置掉电保护，本体 8DI8DO
	LC1032	LC1032-10004128-U0P	赛扬，32 轴，内置掉电保护，本体 8DI8DO
	LC1048	LC1048-10004128-U0P	赛扬，48 轴，内置掉电保护，本体 8DI8DO
	LC1064	LC1064-10004128-U0P	赛扬，64 轴，内置掉电保护，本体 8DI8DO
	LC1096	LC1096-10004128-U0P	赛扬，96 轴，内置掉电保护，本体 8DI8DO
	LC1128	LC1128-10004128-U0P	赛扬，128 轴，内置掉电保护，本体 8DI8DO
LC2000S	LC2064S	LC2064S-10004064-U0P	Alder Lake，64 轴，内置掉电保护
	LC2096S	LC2096S-10004064-U0P	Alder Lake，96 轴，内置掉电保

			护
LC5000	LC5256P	LC5256-25008064-U0P	酷睿 i5 处理器, 256 轴, 内置 UPS
	LC5128P	LC5128-25008064-U0P	酷睿 i5 处理器, 128 轴, 内置 UPS

LC 系列高速 IO 版智能产线控制器见表 1.2。

表 1.2 高速 IO 版智能产线控制器大型 PLC 产品一览表

系列	名称	型号	功能差异
LC5000IO	LC5256IO	LC5256IO-25008064-U0P	酷睿 i5, 256 轴, 带高速 IO、编码器。内置 UPS, 掉电保护 5M
	LC5128IO	LC5128IO-25008064-U0P	酷睿 i5, 128 轴, 带高速 IO、编码器。内置 UPS, 掉电保护 5M

1.1.2. 中型 PLC 产品介绍

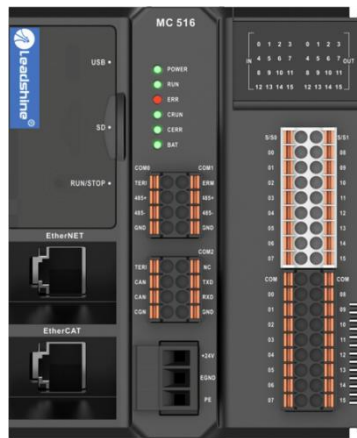
雷赛中型 PLC 产品, PMC600 系列主要包括 PMC600、PMC610 和 PMC616 三款产品, MC500 系列主要包括 MC516CS、MC532CS 两款产品。

雷赛中型 PLC 产品是雷赛面向自动化生产线推出的运动控制器。基于 CODESYS 运动控制平台研发, 支持 EtherCAT 总线, 最高可实现 32 轴运动控制, 特别适合控制工艺复杂的生产线设备, 广泛应用于锂电、包装、物流、3C、电子半导体等行业设备。

中型 PLC 主要特点有:

- 1) 模块化结构, 支持拓展本地模块;
- 2) 性能强大, 最多支持 32 轴的运动控制;
- 3) 通讯接口丰富, PMC600 系列自带 3 个网口, 可在不同的局域网中运行。(第 2、3 网口为交换机功能);

- 4) 运动控制功能丰富，自带高速输入输出，可实现脉冲式电机控制、飞拍、高速位置锁存等功能。



MC500



PMC600

中型 PLC 产品型号见表 1.3。

表 1.3 中型 PLC 产品一览表

型号	功能差异
PMC600	PMC600 系列产品，最大支持 32 轴 EtherCAT 总线控制，本体 200KHZ 高速输入输出，本体支持 4 路脉冲轴，6 路高速计数输入，支持 OPCUA、Ethernet/IP 等多种通讯，支持 PLC 之间进行级联。
PMC616	PMC600 系列产品，最大支持 16 轴 EtherCAT 总线控制，本体 200KHZ 高速输入输出，本体支持 4 路脉冲轴，6 路高速计数输入，支持 OPCUA、Ethernet/IP 等多种通讯，支持 PLC 之间进行级联。
PMC610	PMC600 系列产品，最大支持 32 轴 EtherCAT 总线控制，本体 200KHZ 高速输入输出，本体支持 4 路脉冲轴，6 路高速计数输入，支持 OPCUA、Ethernet/IP 等多种通讯。支持飞拍，高速位置锁存等功能，支持 PLC 之间进行级联。

MC516CS	双核处理器，最大支持 16 轴 EtherCAT 总线控制，本体 200KHZ 高速输入输出，支持 6 路脉冲轴，6 路高速计数输入。
MC532CS	双核处理器，最大支持 32 轴 EtherCAT 总线控制，本体 200KHZ 高速输入输出，支持 6 路脉冲轴，6 路高速计数输入。

1.2. LeadSys 软件介绍

雷赛的 LeadSys 软件基于 CODESYS V3.5 开发，包括 PLC 编程、PLC 配置、本地高速 IO 及编码器配置、扩展 IO 配置、EtherCAT 等总线配置、运动控制，是一个功能丰富的自动化软件。

LeadSys 软件还结合了雷赛特有的运动控制功能，将多种复杂运动控制功能封装成工艺库，供用户使用。

LeadSys 软件具有的特点：

- 采用国际标准 IEC61131-3 架构，支持梯形图 LD、指令表 IL、结构化文本 ST、功能块图 FBD、顺序功能流程图 SFC、控制流程框图 CFC 等多种编程语言；
- 强大的软件仿真、在线调试及程序检查能力，不需要连接 PLC 硬件即可实现程序的调试和仿真；
- 具备方便的产品配置功能，可以轻松快速的实现包括 CPU 配置、通信配置、本地 IO 功能配置、EtherCAT 总线配置等；
- 提供丰富的运动控制功能库和行业工艺库，涵盖多种过程控制和运动控制的应用场景。
- LeadSys 软件从架构上可以分为 3 层：应用开发层、通信层和设备层，如图 1.1 所示。

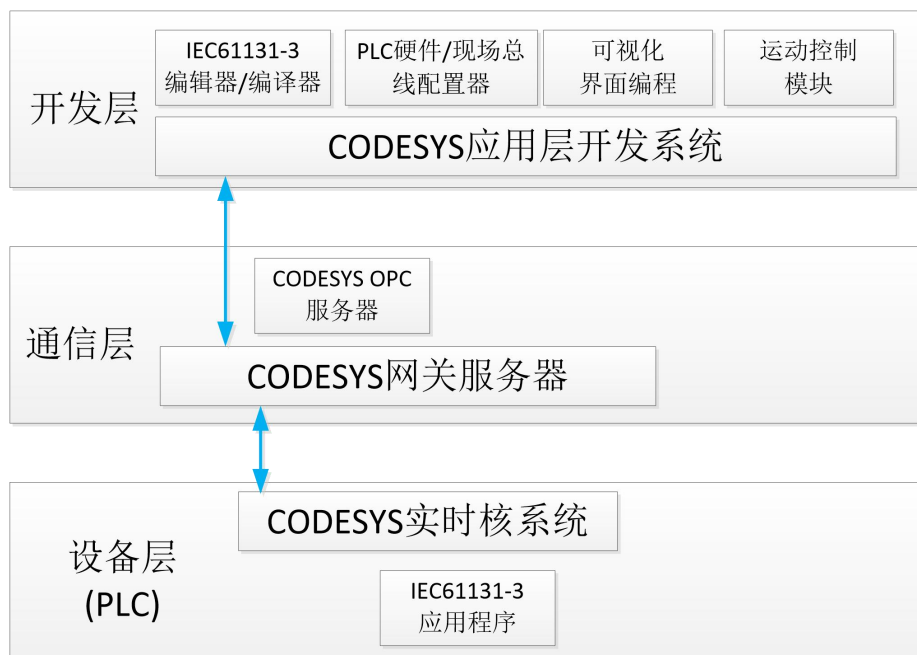


图 1.1 LeadSys 架构示意图

1.2.1. 软件的获取与安装

LeadSys 软件可从 leisai.com 官网下载，网址：<https://www.leisai.com/>。

LeadSys 软件目前只支持 64 位 Windows 操作系统，不再支持 32 位系统。

LeadSys 软件安装步骤如下：

- 1) 安装之前建议关闭 360 等杀毒软件。
- 2) 选择以 LeadSys Studio 开头的 exe 安装文件，点击鼠标右键以管理员的身份运行安装，安装路径下不能有中文；
- 3) 安装提示如图 1.2 所示。用户可选择空间较大的硬盘作为安装路径。在安装中，可选完全安装“Complete”；当遇见“CodeMester 控制中心”界面时，将其关闭即可。最后，点击“Finish”，表示安装成功，在桌面上自动生成图标为



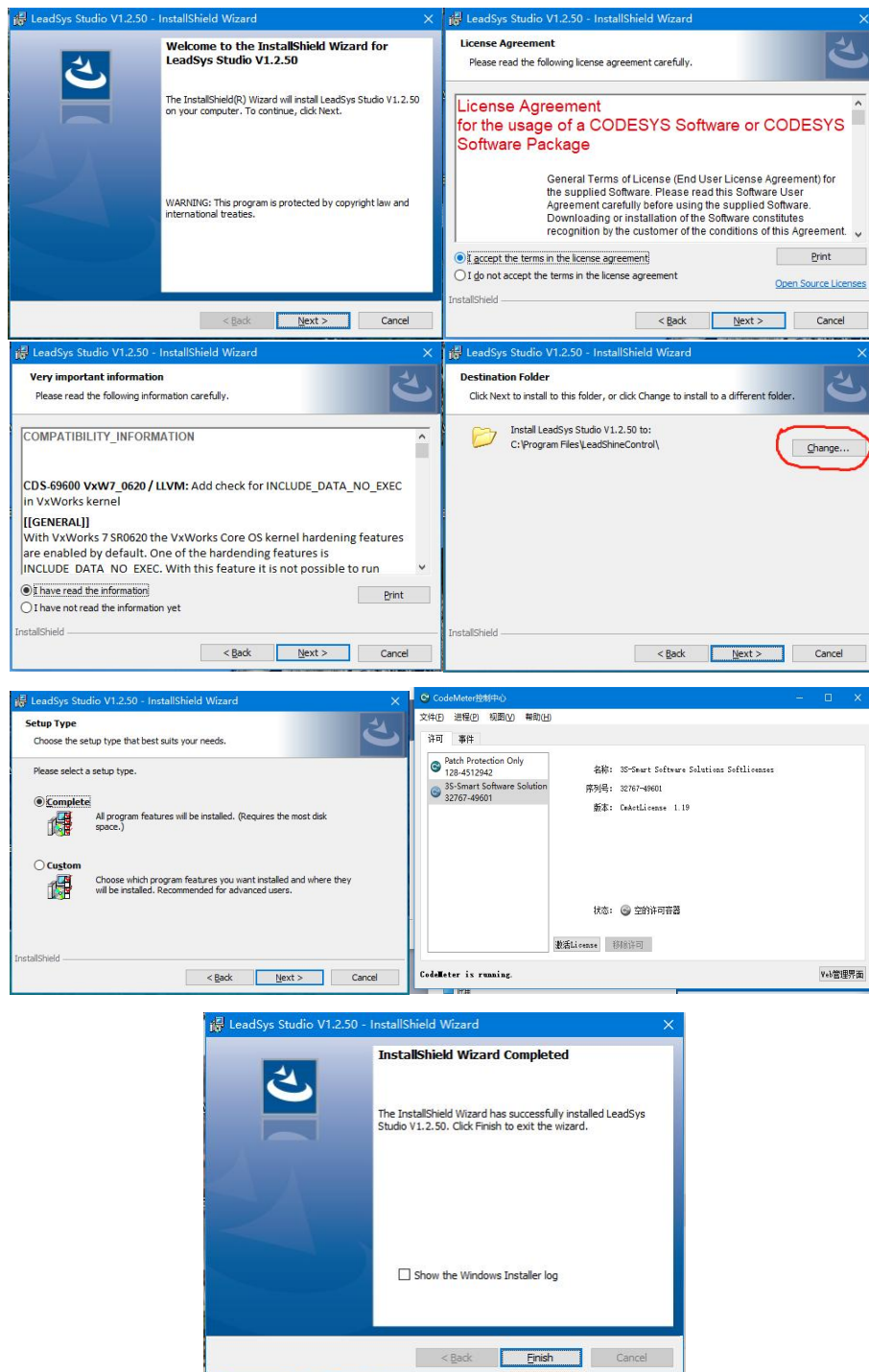


图 1.2 LeadSys 安装提示界面

1.2.2. LeadSys 卸载

有两种方法可卸载 LeadSys 软件。

方法 1: 鼠标点击“控制面板”→“程序”，如图 1.3 所示，用鼠标选中“LeadSysStudioV3.5.16”，在鼠标右键的菜单选“卸载”，可卸载 LeadSys 软

件。

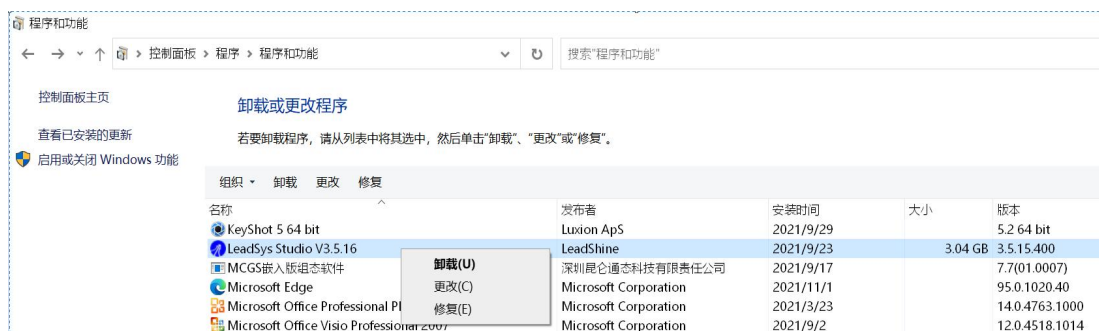


图 1.3 LeadSys 卸载方法 1

方法 2：以管理员身份运行以 LeadSys Studio 开头的 exe 安装文件，在第二步选择“remove”，可卸载 LeadSys 软件，如图 1.4 所示。

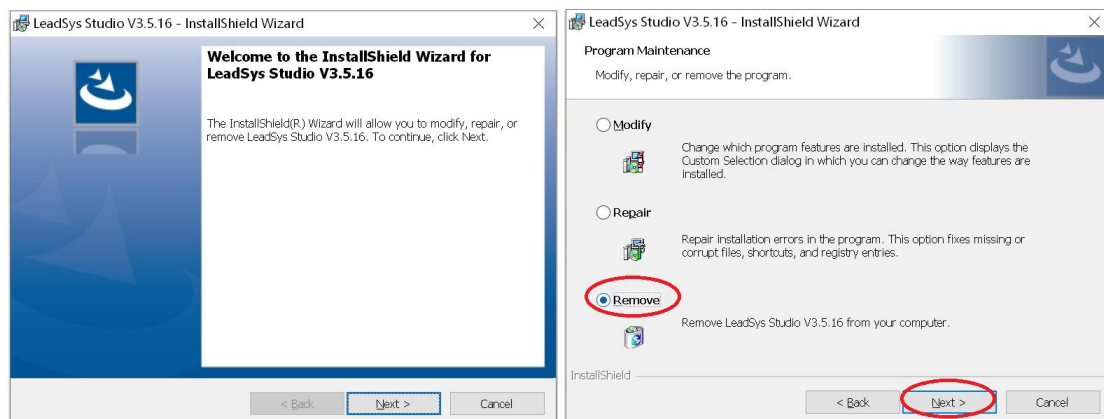


图 1.4 LeadSys 卸载方法 2

1.3. 系统应用架构

1.3.1. 大型 PLC 系统应用架构

1) 大型 PLC 电源连接

LC 控制器电源电压为 DC24V，需要外部开关电源提供 24V 输入。建议给控制器独立配备开关电源，避免与电机、传感器共用一个电源。

选择开关电源的负载能力时，可参照表 1.4。

表 1.4 LC 系列产品的最大功耗

产品系列	最大功耗
LC2000S 系列	20W
LC5000 系列	40W

2) 一站式解决方案

LC 控制器具备 EtherCAT 双主站、双 EtherNet 口、RS-485、RS-232、HDMI、USB2.0、USB3.0 接口。EtherNet 口支持 EIP、TCP\UDP (Socket)、ModbusTCP、OPC UA 等协议，支持触摸屏的标签通信连接。RS-485 和 RS-232 接口支持 Modbus RTU 协议。

EtherCAT 主站可接入 EtherCAT 总线型伺服电机和步进电机、ECat 分支器、雷赛 R1\R2 系列远程 IO 模块等，如图 1.5 所示。

LC2000S/LC5000 控制器的通信接口如图 1.6 所示。

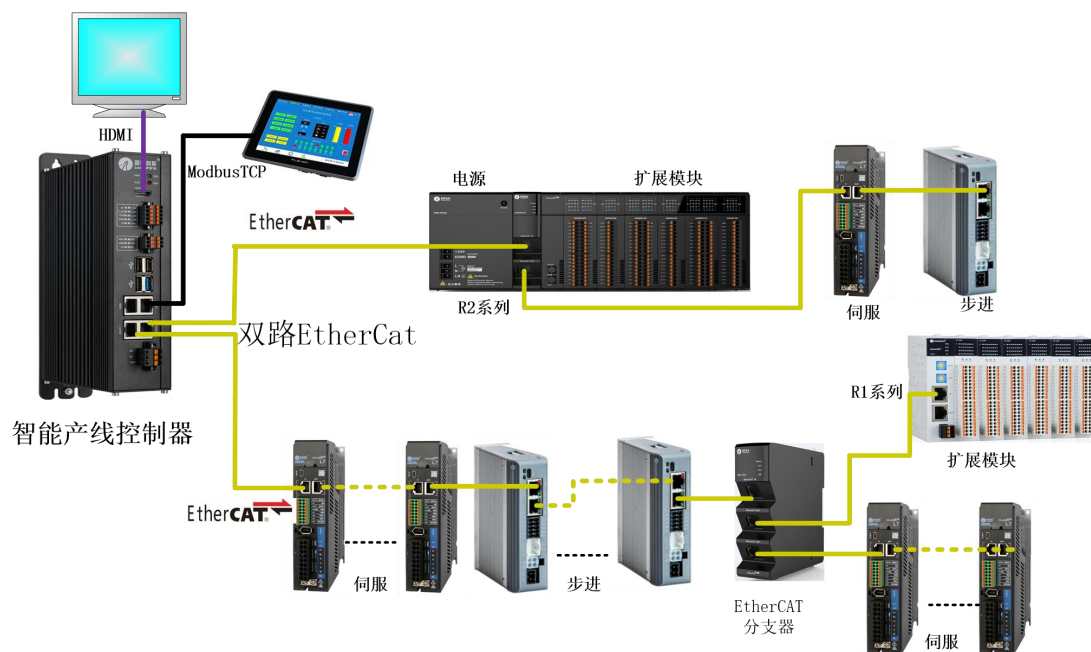


图 1.5 LC 系列控制器应用架构

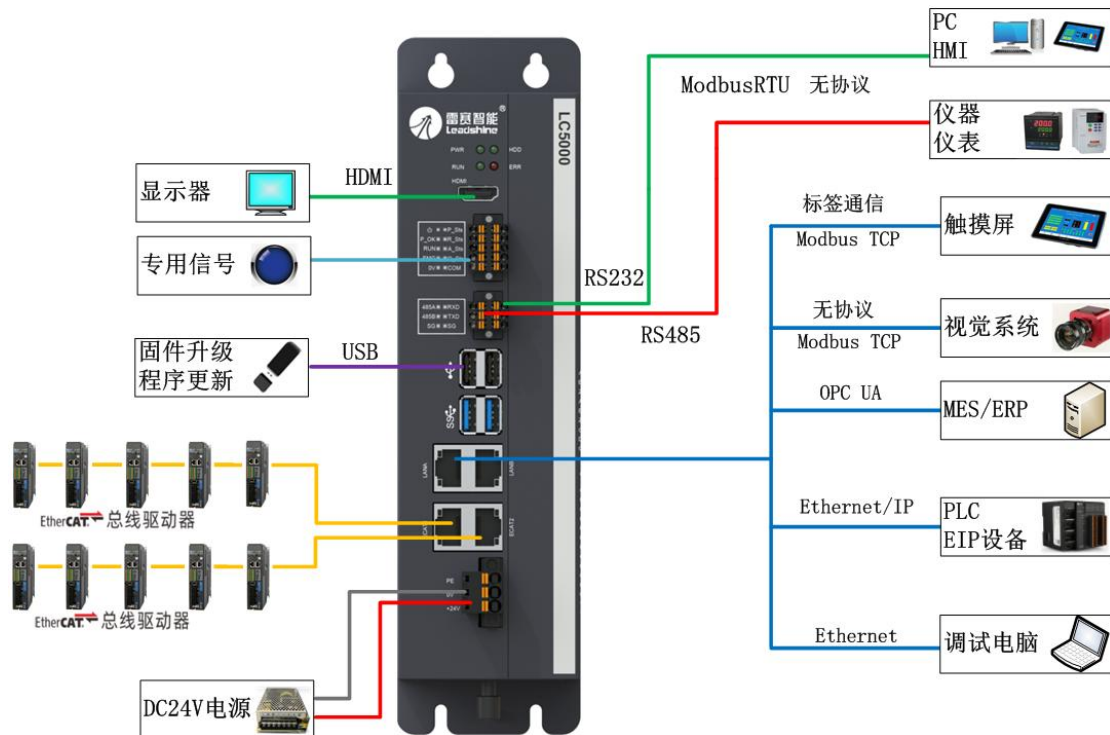


图 1.6 LC2000S/LC5000 控制器的通信接口

1.3.2. 中型 PLC 系统应用架构

PMC600 系列控制器的应用架构如图 1.7 所示。

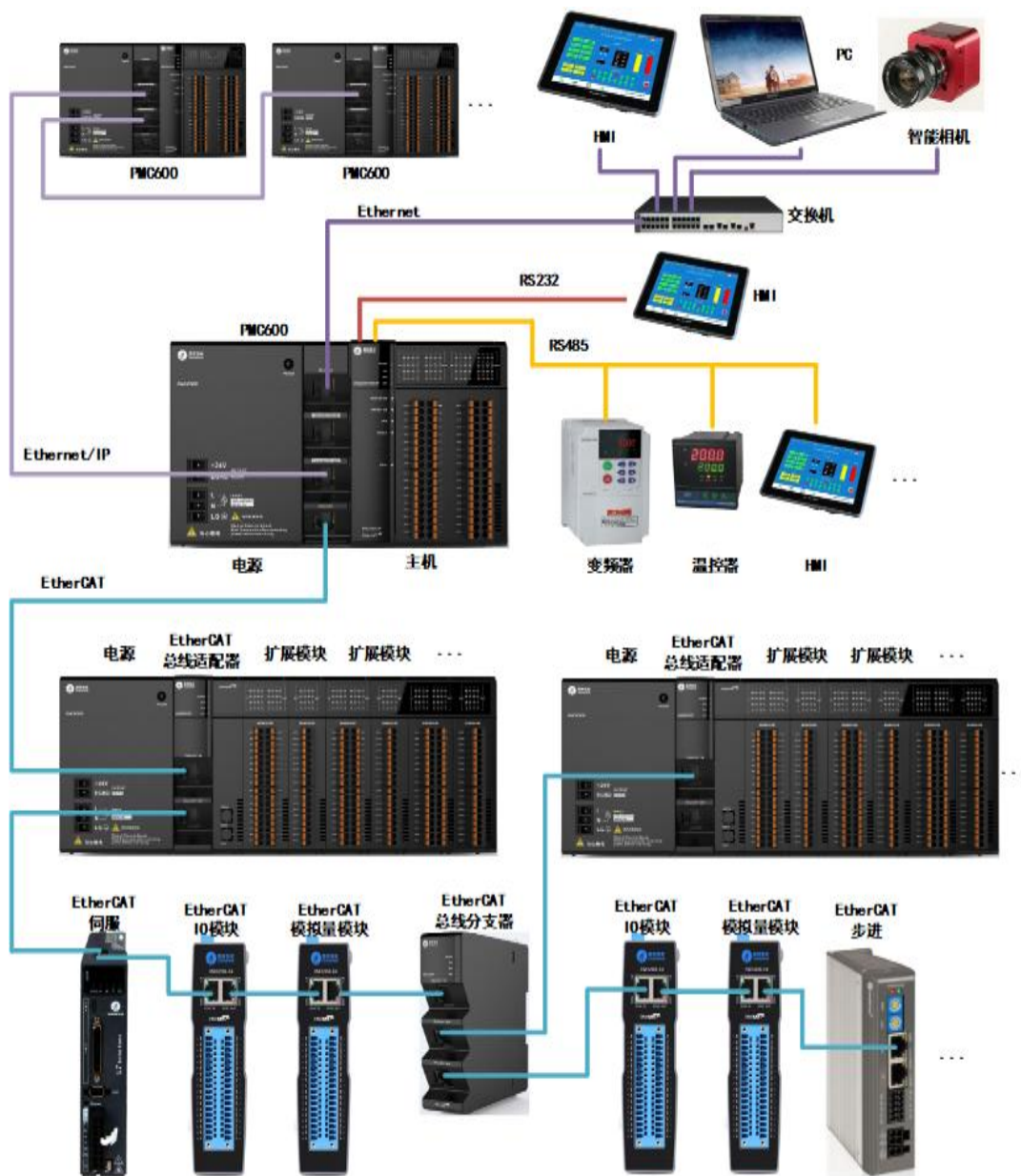


图 1.7 PMC600 系列控制器的应用架构

MC500CS 中型 PLC 产品提供 Ethernet 接口，EtherNet 口支持 EIP、TCP/UDP（Socket）、ModbusTCP、OPC UA 等协议，支持触摸屏的标签通信连接。提供 EtherCAT 总线接口，可以和雷赛伺服、步进，雷赛 R1/R2 系列远程扩展模块，其典型应用架构如图 1.8 所示。

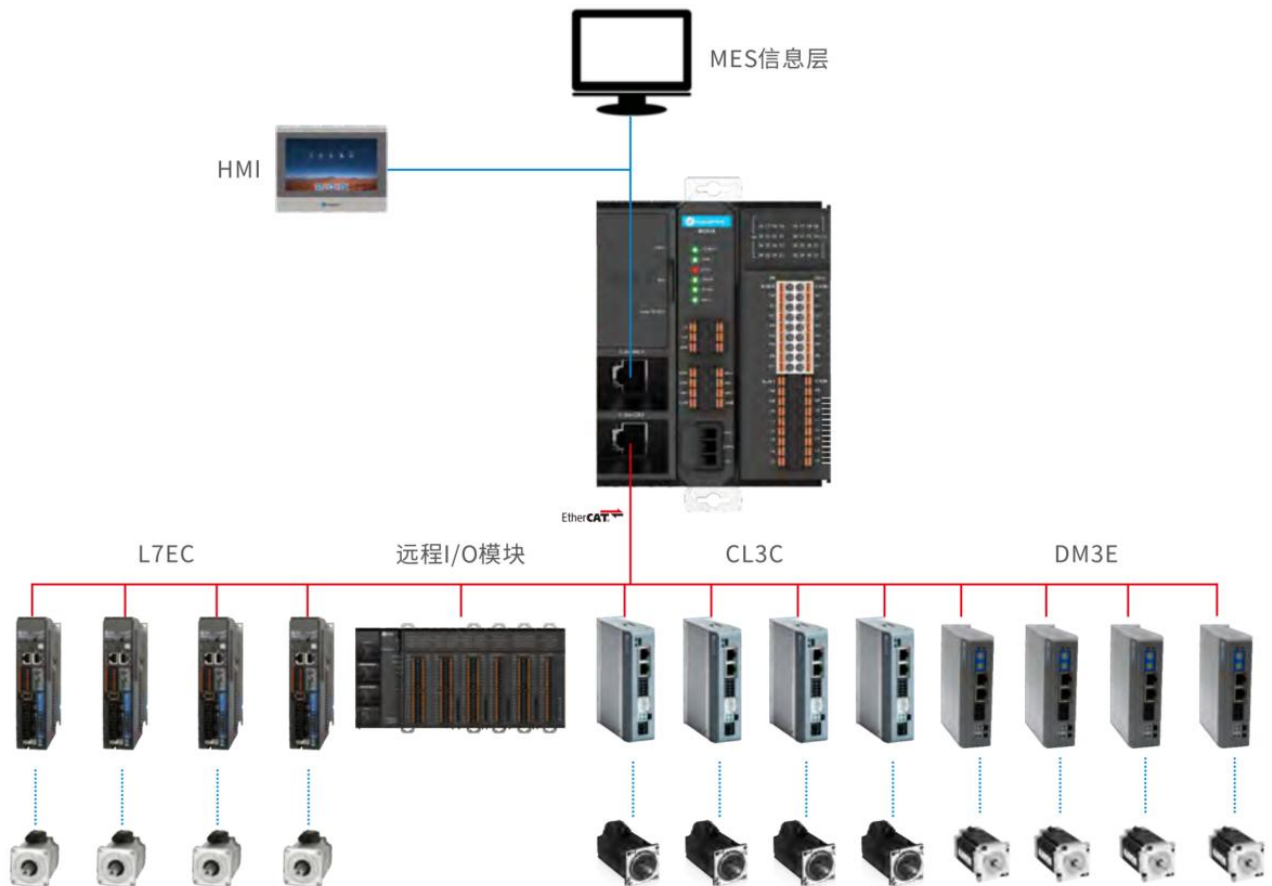


图 1.8 MC500CS 系列控制器的应用架构

除了 Ethernet 网口和 EtherCAT 总线接口外，雷赛中型 PLC 还提供 2 路 R485,232 串口，CAN 总线等常用的接口，方便进行多种外设连接。

1.3.3. 基本使用流程

大、中型 PLC 的基本应用流程如图 1.9 所示。PLC 的安装方法参见相应的硬件简易手册。



图 1.9 大、中型 PLC 的基本应用流程

第二章 快速入门

2.1.编程环境及配置

在 Windows 系统开始菜单中或者点击桌面的快速方式,打开 LeadSys Studio 软件。

LeadSys Studio 的 PLC 工程界面如图 2.1 所示。界面关键指引如图 2.2 所示。

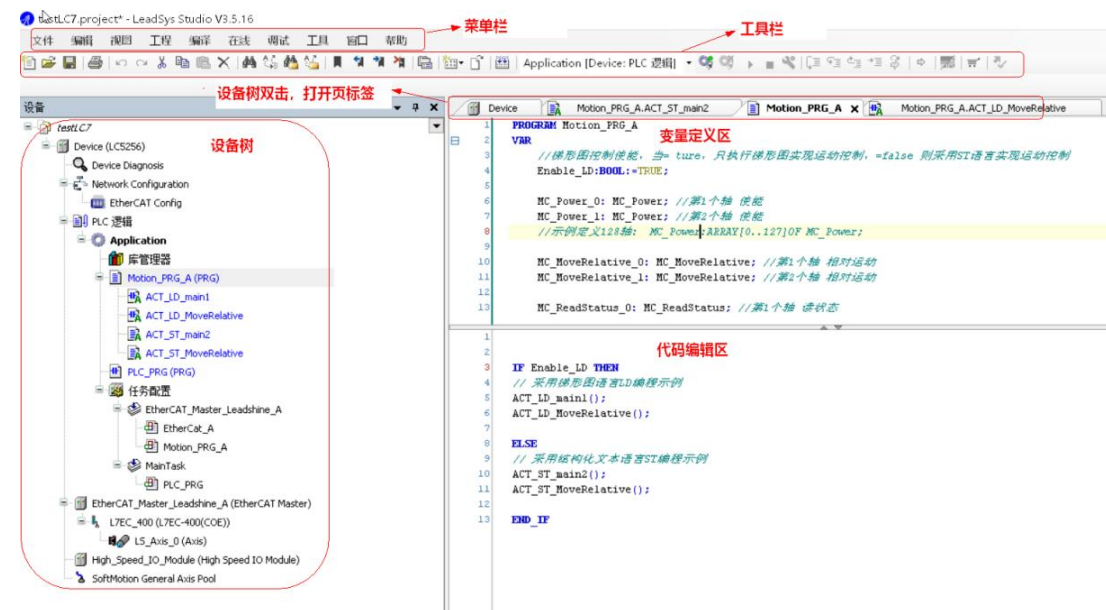


图 2.1 LeadSys Studio 的 PLC 工程界面

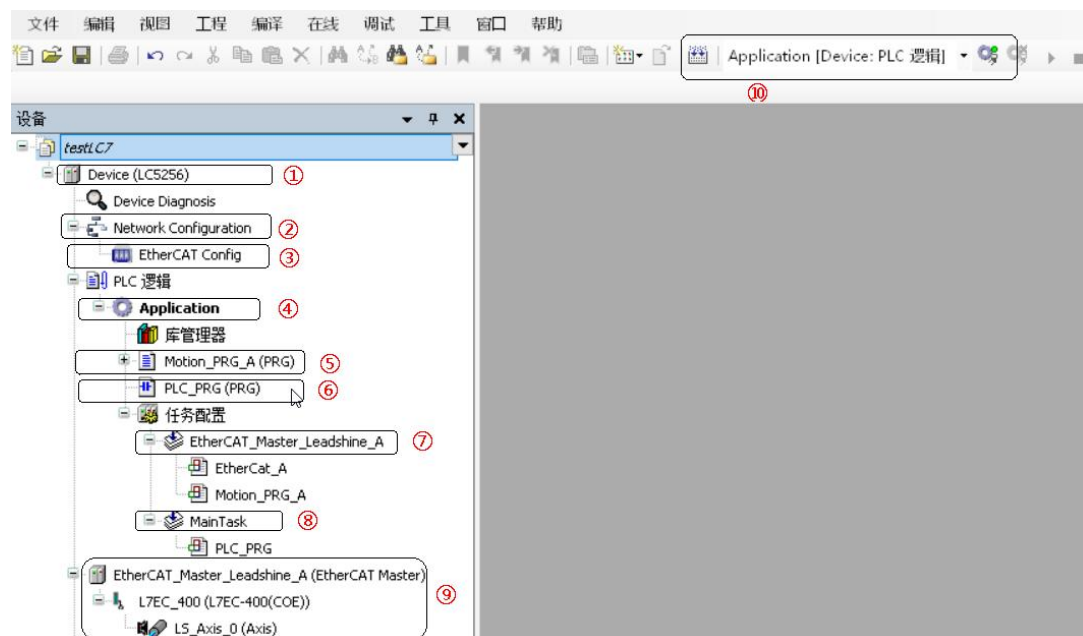


图 2.2 PLC 工程界面关键指引

图 2.2 中标注的各部分名称与注解如下：

- ① PLC 设备信息。点击鼠标右键可更换 PLC 型号
- ② EtherCAT 总线配置
- ③ 本地或远程 IO 模块配置
- ④ 用户程序管理单元
- ⑤ EtherCAT 总线 A 运动控制程序 Motion_PRG_A。双总线时，可建程序 Motion_PRG_B。
- ⑥ PLC_PRG 主程序，编程语言为新建工程时选择的语言。也可删除，重新建 LD 语言的主程序。主程序和运动控制程序的命名可由客户根据习惯或命名规范进行自定义。
- ⑦ EtherCAT 总线 A 的任务
- ⑧ 主程序的任务
- ⑨ EtherCAT 总线从站和 402 轴编译、登录及调试快捷键

2.2.编写程序的典型步骤

编写调试一个完整的用户程序一般需要以下 6 个步骤：

- 1) 按照 PLC 应用系统的硬件配置和网络架构进行硬件系统配置。
 - 将伺服电机驱动器、步进电机驱动器、远程 IO 扩展模块等标准 EtherCAT 总线设备，按照应用系统网络架构配置到 EtherCAT 总线上。
 - 若用到远程 IO 耦合器，需要在远程 IO 扩展界面上，按照应用系统配置 IO 模块。
- 2) 根据应用系统的控制工艺编写用户程序。
- 3) 将系统架构中的各硬件端口对应的输入端口变量（I）、输出变量（Q）或数值（M）与用户程序中的变量进行关联。
- 4) 配置网络通信的同步周期（如 EtherCAT 总线）。根据各任务的实时性要求，配置用户程序单元的执行周期。
- 5) 将编写好的程序单元 POU 按照需求放置到对应的任务中。

6) 登录 PLC，下载用户程序，仿真调试、排错，直到程序正确运行。

2.3.新建工程

所谓“工程”，就是一个完整的 PLC 应用程序包，它包含了控制器网络参数的设置、电机驱动器、IO 模块的参数配置和程序模块和代码等内容。

打开 LeadSys 软件后，鼠标点击菜单中的“文件”→“新建工程”，打开图 2.3 所示的界面，选择“标准工程”，选择本地电脑硬盘上的存放 PLC 工程的位置，填写本工程的名称，如：“testLC”，然后确定，进入下一个界面。

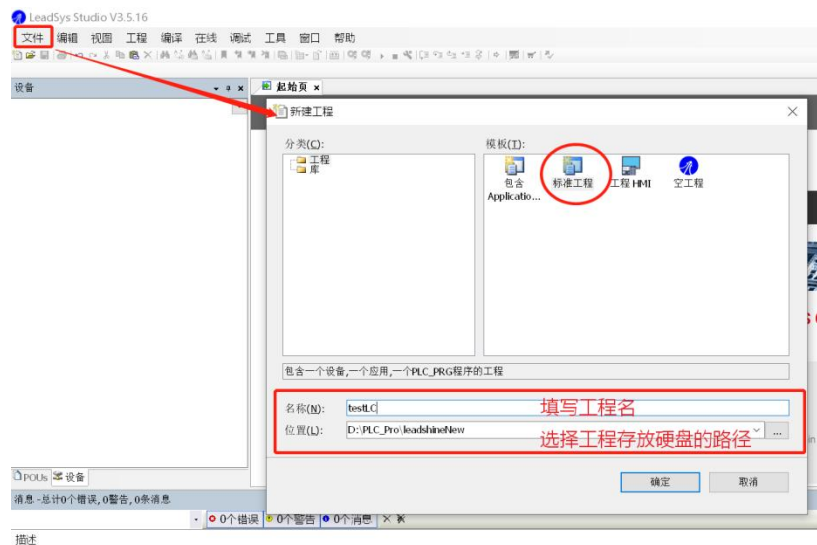


图 2.3 新建工程界面

然后选择 PLC 类型，如图 2.4 所示，请根据实际 PLC 类型选择设备



图 2.4 选择 PLC 类型

再选择主程序 PLC_PRG 的编程语言（默认为 ST），选择完成后点击“创建”，如图 2.5 所示。



图 2.5 选择初始语言与创建工程

创建完成后，进入编程界面，如图 2.6 所示。

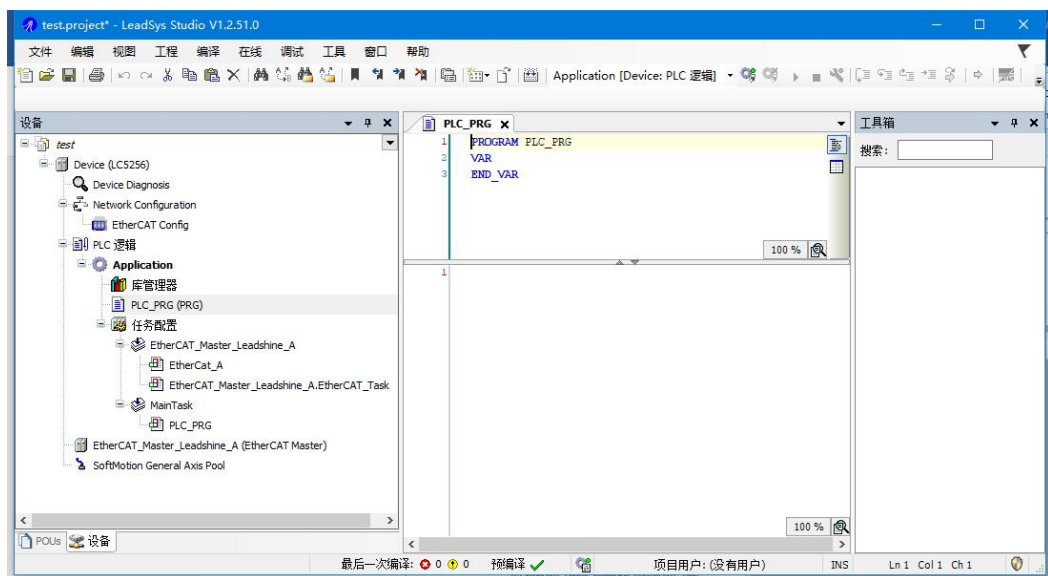


图 2.6 编程界面

默认的工程在任务配置后，已经建好了 A 口 EtherCAT 总线对应的任务名：

EtherCAT_Master_Leadshine_A，并在其下自动建好了总线相关的执行任务：

EtherCAT_Master_Leadshine_A.EtherCAT_Task

在编写与总线 A 口所连接的电机的控制程序时，需要建立一个 POU_A 程序，并

放在 EtherCAT_Master_Leadshine_A 下执行。

大型 PLC 支持双主站，我们简称为总线 A 和总线 B。采用双 EtherCAT 总线，则需要 A 和 B 口分别编写控制程序 POU，例如 POU_A 和 POU_B，如图 2.7 所示。

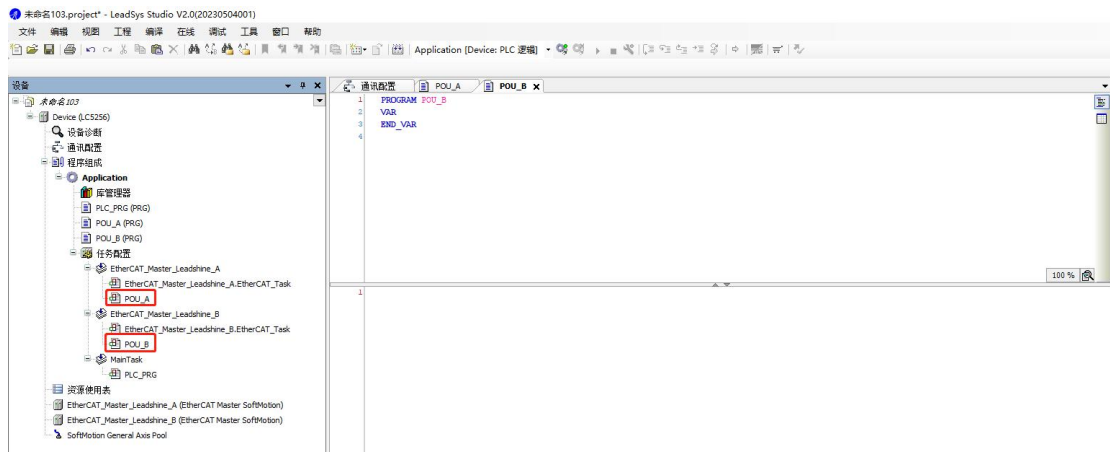


图 2.7 采用双 EtherCAT 总线时，2 个控制程序 POU_A、POU_B 的摆放位置

2.4. 电脑与 PLC 的连接

2.4.1. 电脑网口 IP 设置并连接 PLC

- 1) 用网线通过网口将电脑与 PLC 连接起来。然后，配置电脑网口 IP 地址与 PLC 设备 IP 地址在同一网段。

LC5000 系列 PLC 上的以太网 1 口 LANB 和 PMC600 上的 EtherNet 口的出厂默认地址为 192.168.1.3，PC 机要与其通信，必须将电脑上的 IP 地址设为同段，即前 3 段一样：192.168.1，最后一段要不一样。

- 2) 在 LeadSys 软件界面左侧树，双击“Device”，如图 2.8 所示，点击“扫描网络”，当右侧的 PLC 灯为绿色时，表示 PLC 设备连接成功，可下载 PLC 程序。

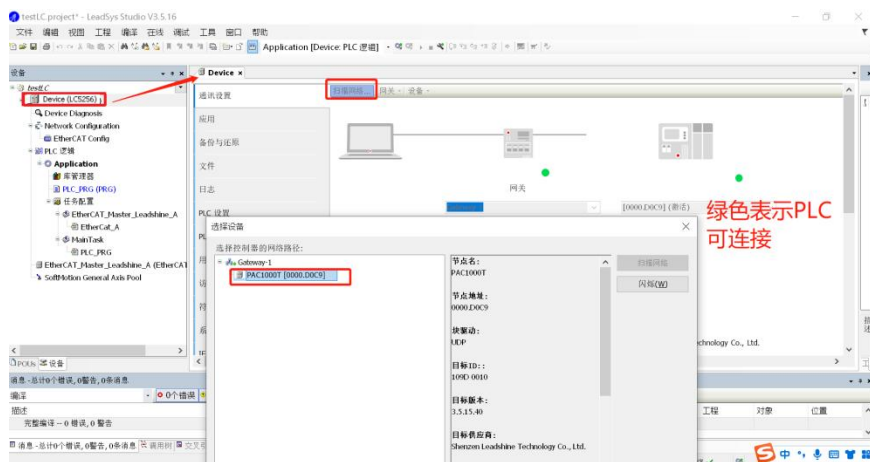


图 2.8 电脑与 PLC 连接

2.4.2. 登录及运行 PLC

新建工程后，可编写一个最简单的程序，测试一下电脑与 PLC 的连接状态。

在程序编辑区添加一个变量 test，主程序中 test 每个运行周期自增 1。程序如图 2.9 所示。

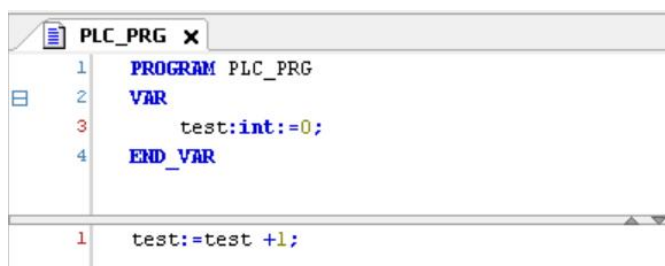


图 2.9 最简单的测试程序

鼠标点击工具栏中的快捷键“”编译 PLC 程序，再点击登录按钮“”，如图 2.10 所示，可登录 PLC；再点击按钮“”，可运行程序。

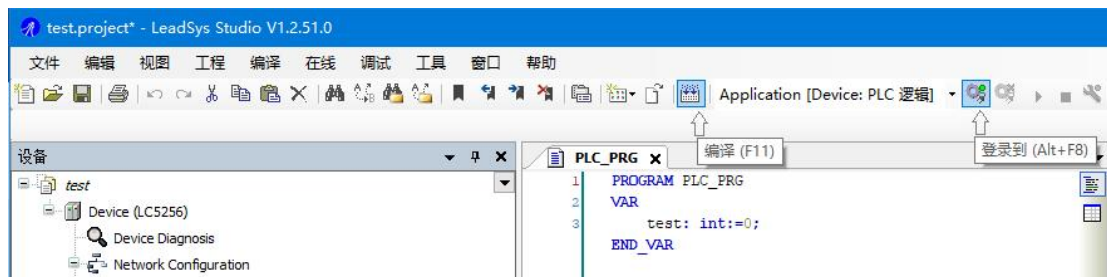


图 2.10 编译程序、登录 PLC

首次下载程序时，有如图 2.11 所示的提示界面。选择“是”即可。



图 2.11 首次下载程序的提示界面

如果 PLC 设备中已经有程序，再次下载修改过的程序时会出现不同的提示界面。一般选择第 2 项，如图 2.12 所示。其中“更新自动启动程序”的意思是：下次 PLC 重新上电后，PLC 将自动运行该程序。

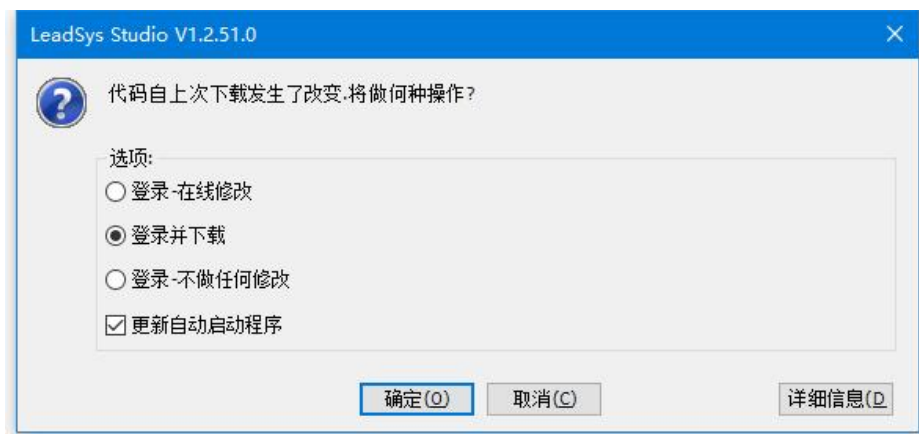


图 2.12 再次下载程序的提示界面

如果出现如图 2.13 所示的提示界面，表示网络不通，电脑与 PLC 连接的网络或 IP 地址设置有问题。



图 2.13 电脑与 PLC 连接失败

2.4.3. PLC 的 IP 地址读取和修改

登录 PLC 后，在工程树左侧鼠标双击“device”后，选择“系统设置”栏，在界面的下方可读取 PLC 的 IP 地址，也可在此界面修改 PLC 的 IP 地址，点击“设置”按钮，修改地址后需要重启 PLC 方可生效，如图 2.14 所示，地址生效后，需要重新连接 PLC。

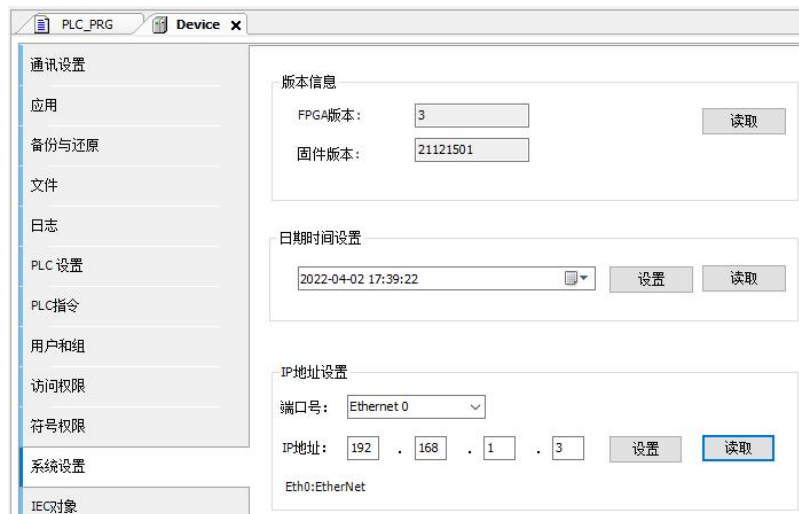


图 2.14 读取、修改 PLC 的 IP 地址

2.5. 设备组态

所谓设备组态即向新建的工程里添加 EtherCAT 总线上的电机、IO 模块或其他元器件。

设备组态通常有两种方式：

- 自动扫描设备组态：通常在 PLC 和 EtherCAT 设备上电，且 EtherCAT 总线已将伺服电机、IO 模块等 EtherCAT 设备连接起来后，通过 LeadSys 软件扫描总线设备，自动完成组态。
- 离线设备组态：在没有 PLC、伺服电机等设备网络连接的环境时，可以直接在 LeadSys 软件上进行设备可视化配置。

下面分别介绍两种设备组态方法。

2.5.1. 自动扫描设备组态

以“PLC+EtherCAT 总线连接 5 台伺服电机驱动器 L7EC-400”为例，演示自动扫描设备组态过程。

- 1) 双击“device”，如图 2.15 所示，扫描网络，选择扫描到的 PLC。

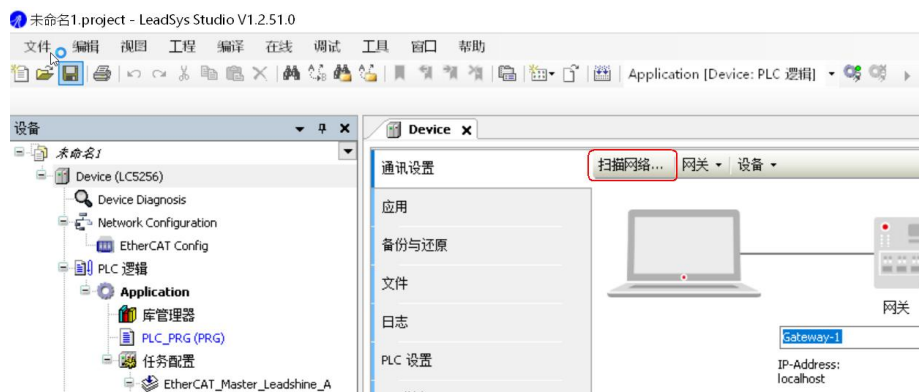


图 2.15 “扫描网络”，连接 PLC

- 2) 根据实际网口连接，在组态界面勾选 EtherCAT 主站，然后登录下载程序。
- 3) 退出登录，右键点击主站设备，选“扫描设备”，如图 2.16 所示，就能扫描到从站上的设备，并弹出扫描设备的结果，如图 2.17 所示。

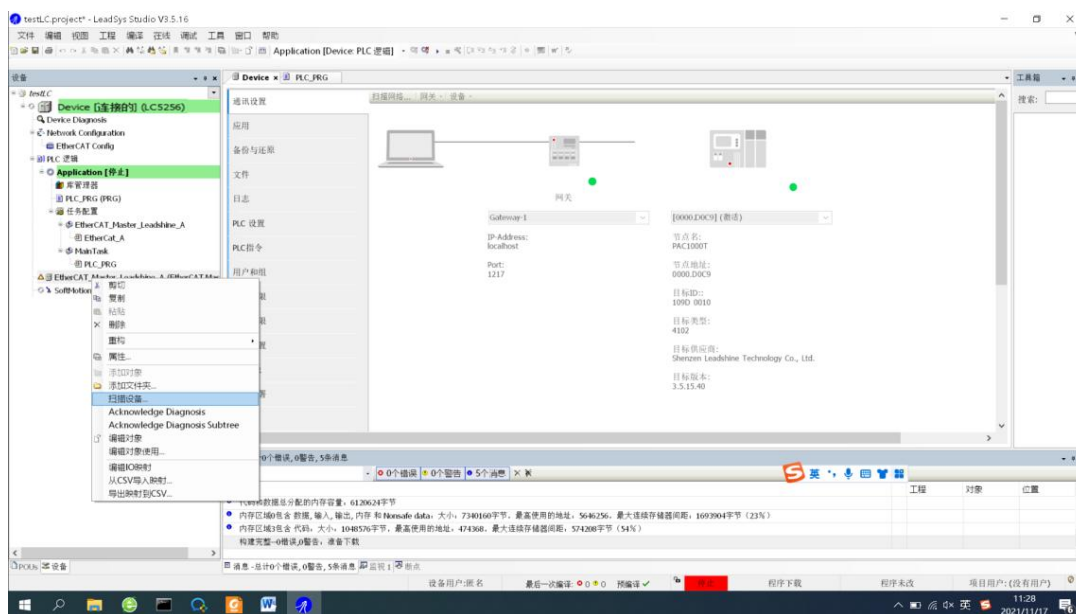


图 2.16 扫描网络设备

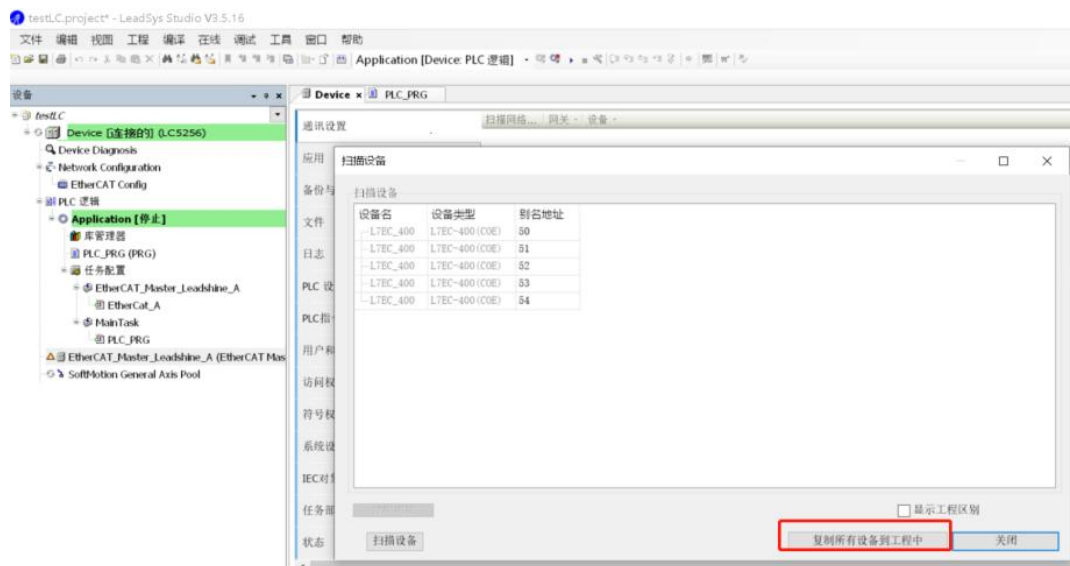


图 2.17 扫描到设备

点击“复制所有设备到工程中”按钮，将自动添加了 5 个伺服驱动器和对应的 402 轴。

注：雷赛伺服电机不需要后期手动加轴，第三方伺服电机需要手动添加。

- 1) 检查扫描组态的 5 台伺服电机，如果没有 402 轴，如图 2.18 所示，需要在离线状态下，手动添加 402 轴。（正常情况，不需要操作这一步）

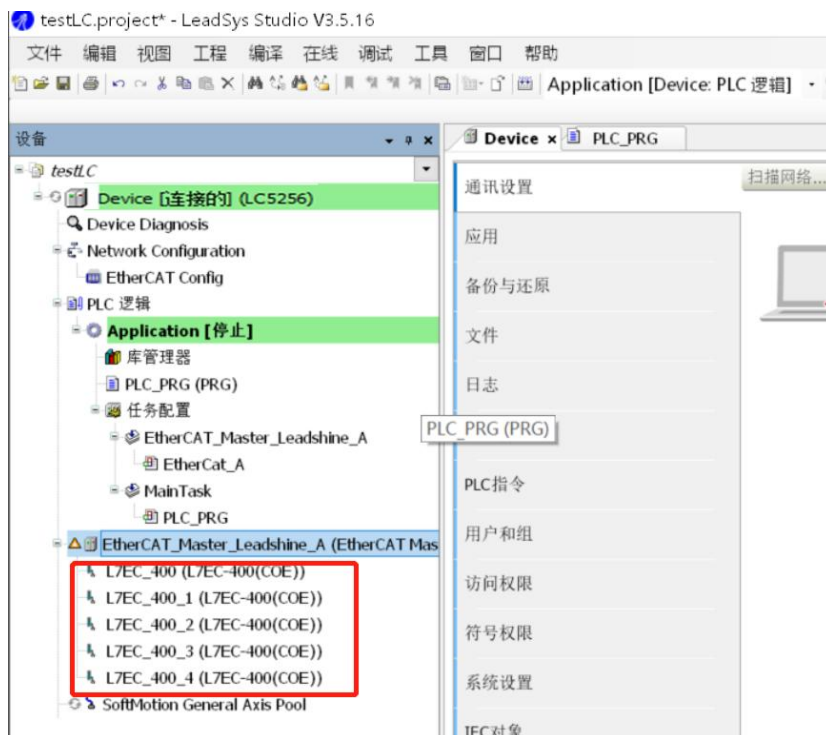


图 2.18 电机没有对应的 402 轴

- 2) 方法：选择没有 402 轴的伺服电机，如图 2.19 所示，选择“L7EC_400”，
点击右键，添加“softMotion CiA402 轴”。

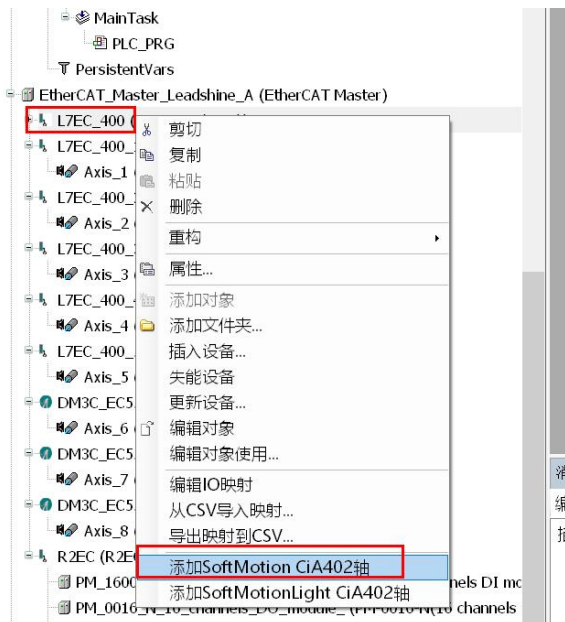


图 2.19 手动添加 402 轴

图 2.20 为手动添加 402 轴后的界面。

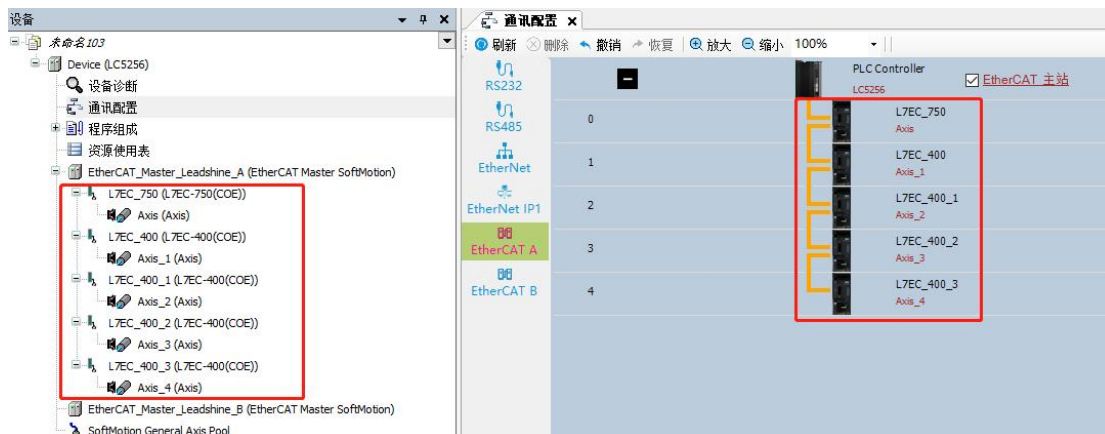


图 2.20 已添加 402 轴

以上 402 轴自动命名为 Axis、Axis_1、Axis_2 等轴变量名，用户可以将这些轴变量名改为自己习惯或规范的变量名。

2.5.2. 离线设备组态

下面以一个总线系统为例，讲解离线设备组态的方法。

该系统采用 LC5256 型 PLC，控制 6 个伺服电机驱动器、3 个步进电机驱动器和 1 个雷赛 R2 系列远程 E 总线耦合器且该耦合器配置 1 个 DI 模块、一个 DO 模块和一个 DX 模块。

设置步骤如下：

- 1) 使用鼠标双击、拖动、右键插入设备等方式，设置离线设备组态。如图 2.21 所示，打开网口 A 的 E 总线组态界面，在界面右侧的设备网口 A 下，鼠标选择 L7EC-400，拖入 6 个伺服电机驱动器。也可以在右侧设备网口 A 下，鼠标选择 L7EC-400，然后双击鼠标，在总线尾部增加 1 个伺服电机驱动器。

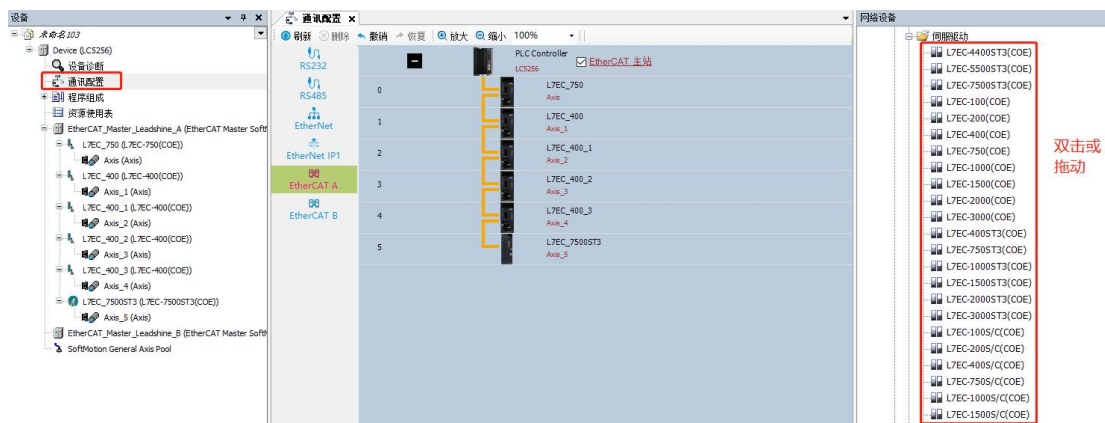


图 2.21 离线添加伺服电机驱动器

在总线上插入设备时，需要先选择一个从站设备，右键选择“插入设备”，如图 2.22 所示。

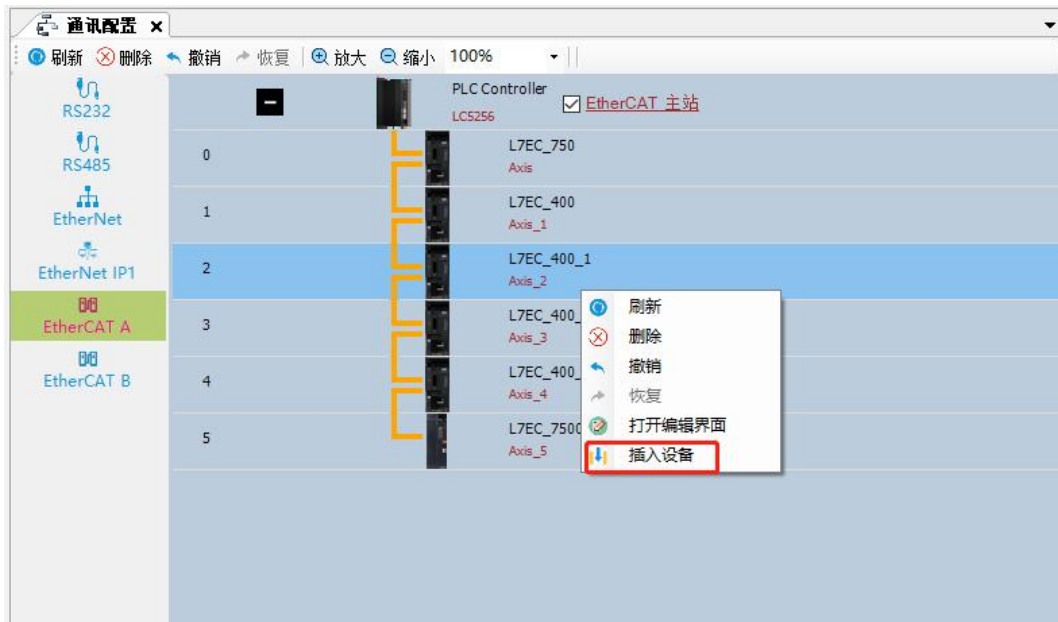


图 2.22 在总线上插入设备

弹出如图 2.23 所示界面，选择设备及插入数量，双击选择设备，完成设备插入。

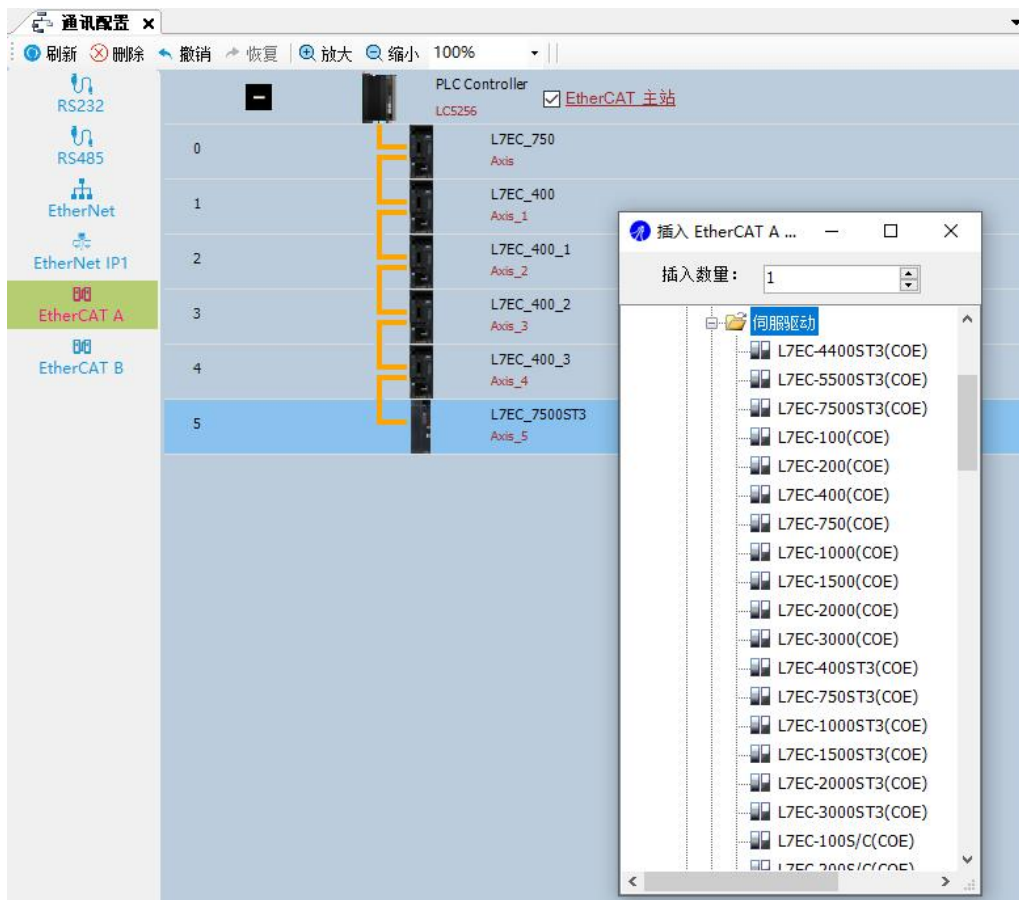


图 2.23 确定插入设备的数量和型号

- 2) 再拖入 3 个步进电机驱动器、1 个雷赛 R2 耦合器（设备顺序根据工程实际需求配置），如图 2.24 所示。

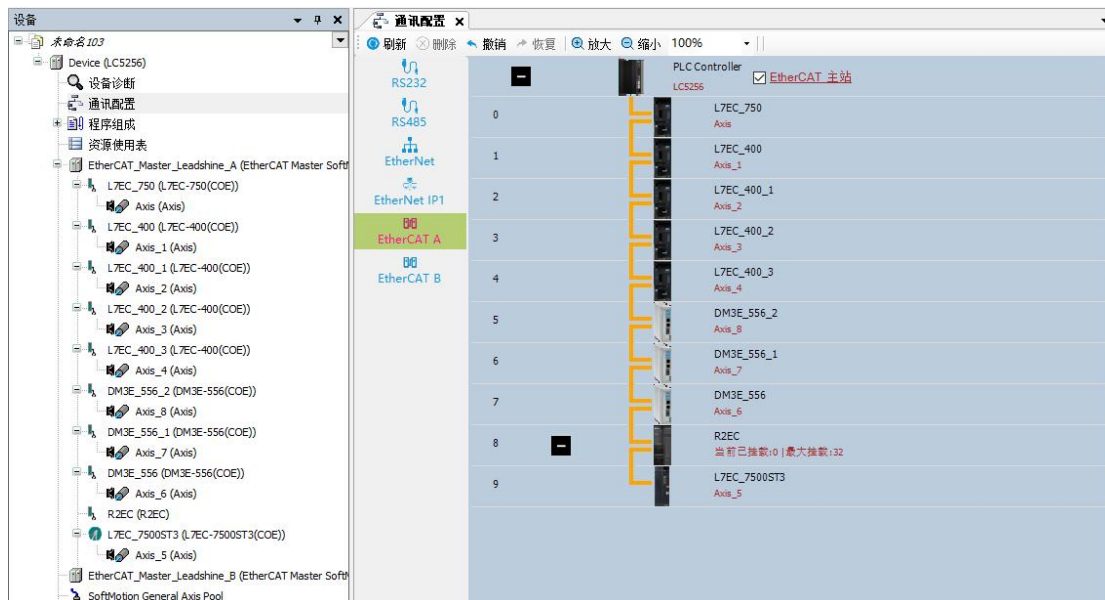


图 2.24 添加步进电机驱动器和 R2 耦合器

- 3) 点击 R2 耦合器，拖入或双击右侧工具箱中的 R2 系列 ECat 耦合器下的 PM-1600(16DI)、PM-0016-N（16DO）和 PM-1616-N（16DI16DO）模块。



图 2.25 添加 R2 耦合器上的模块

4) 保存工程。到此，完成离线设备组态工作。

2.6. LC5000 控制器例程

本节将通过 2 个简单例程详细介绍 LC5000 系列和 PMC600 系列 PLC 从新建工程、到配置 IO 接口、步进电机参数的设置，再到程序编写和运行的全过程。

2.6.1. LC5000 实验系统硬件

LC5000 系列 PLC 实验系统的硬件如图 2.26 所示。该系统由一台 PC 机、一台 LC5000 控制器、一个 EM32DX-E4IO 扩展模块、LED 指示灯、开关、一个 DM3E-522 总线型步进电机驱动器及步进电机组成。

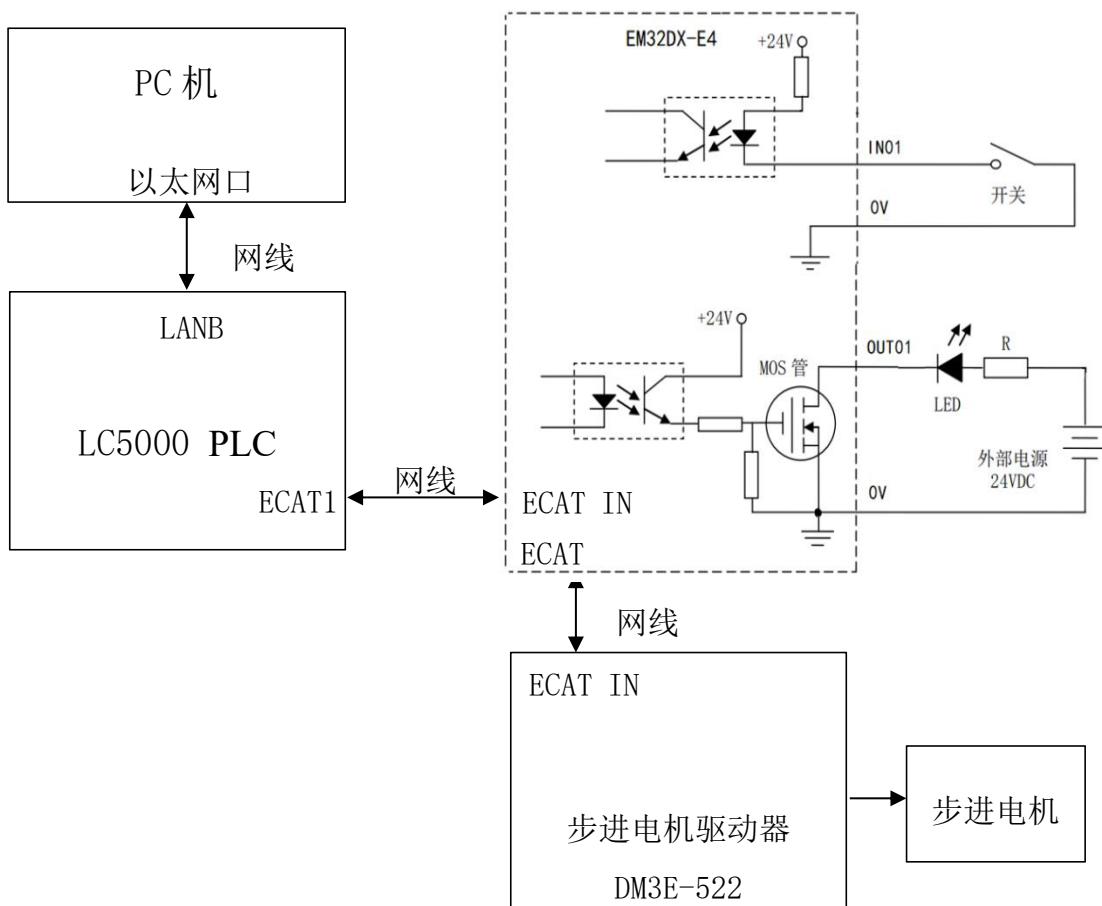


图 2.26 LC5000 实验硬件接线图

2.6.2. 用 LC5000 和 IO 模块控制 LED 灯的开关

用 LC5000 系列 PLC 编写一段程序，根据 IO 模块输入口的开关状态，控制 IO 模块输出口上的 LED 的亮与灭。要实现这个简单的功能需要完成以下几个步骤。

1) 设置 PC 机的 IP 地址

如图 2.26 所示，用网线将 LC5000 的 LANB 口与 PC 机的网口连接，然后上电。打开 Windows 的“网络和共享中心”界面，如图 2.27 所示；点击与 LC5000 连接的“以太网”，打开了“以太网状态”界面，如图 2.28 所示；点击上面的“属性”按钮，打开的“以太网属性”界面，如图 2.29 所示。

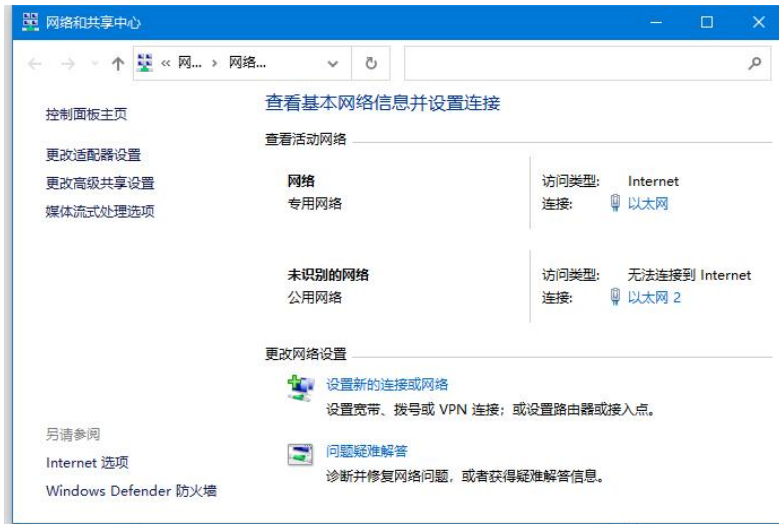


图 2.27 “网络和共享中心”界面



图 2.28 “以太网状态”界面

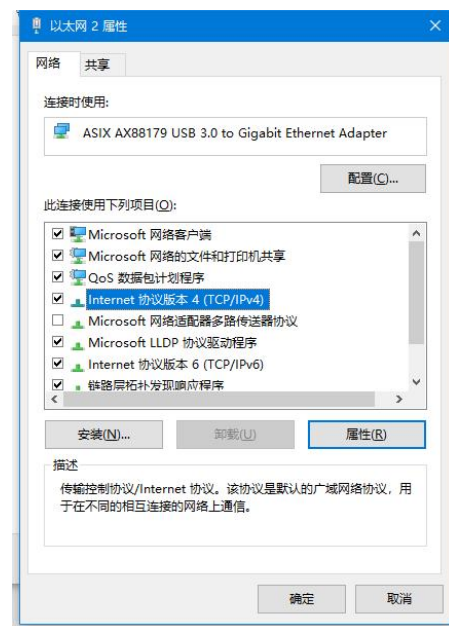


图 2.29 “以太网属性”界面

选择“Internet 协议版本 4（TCP、IPv4）”，再点击“属性”按钮，进入 Internet 协议版本 4（TCP、IPv4）属性界面，按图 2.30 所示，修改其参数，最后点击“确定”按钮。

注意：LC5000 系列 PLC 的网口 LANA 的出厂默认 IP 地址为：192.168.70.188；网口 LANB 的出厂默认 IP 地址为：192.168.1.3。

PC 机上的 IP 地址前 3 段要与 PLC 的相同，最后一个不能相同！

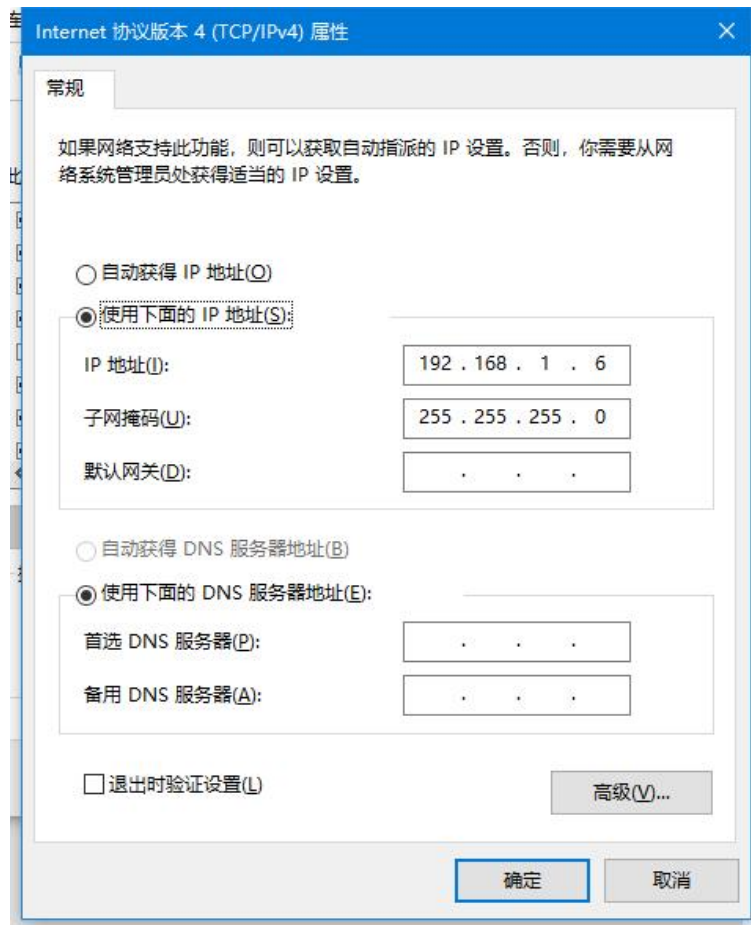


图 2.30 “Internet 协议版本 4（TCP、IPv4）”界面

2) 启动编程软件

双击 LeadSys Studio 软件图标如图 2.31 所示，启动软件，界面显示如图 2.32 所示。



图 2.31 LeadSys Studio 软件图标



图 2.32 启动 LeadSys Studio 软件

点击“新建工程”快捷栏或者点击“文件”→“新建工程”，新建一个工程项目；在如图 2.33 所示的界面中选择“Standard project”（标准工程），输入工程名称和保存位置，点击“确定”；然后在如图 2.34 所示界面中选择“设备”为 LC5256，编程语言选择“结构化文本（ST）”。得工程界面如图 2.35 所示。

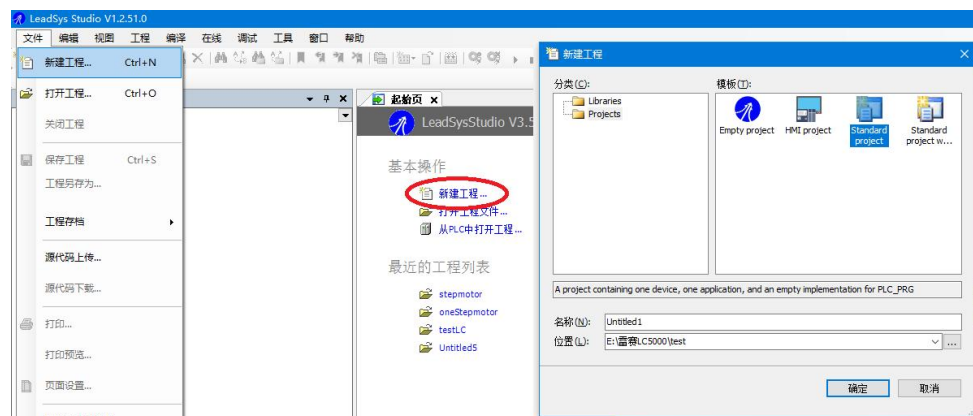


图 2.33 新建工程界面

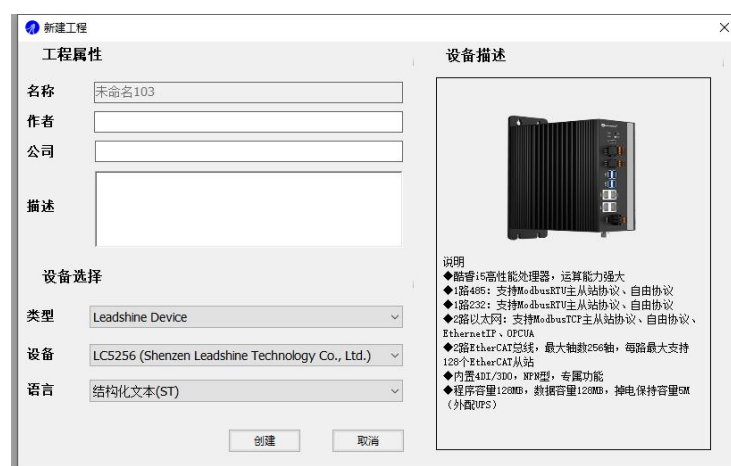


图 2.34 “标准工程选择”界面

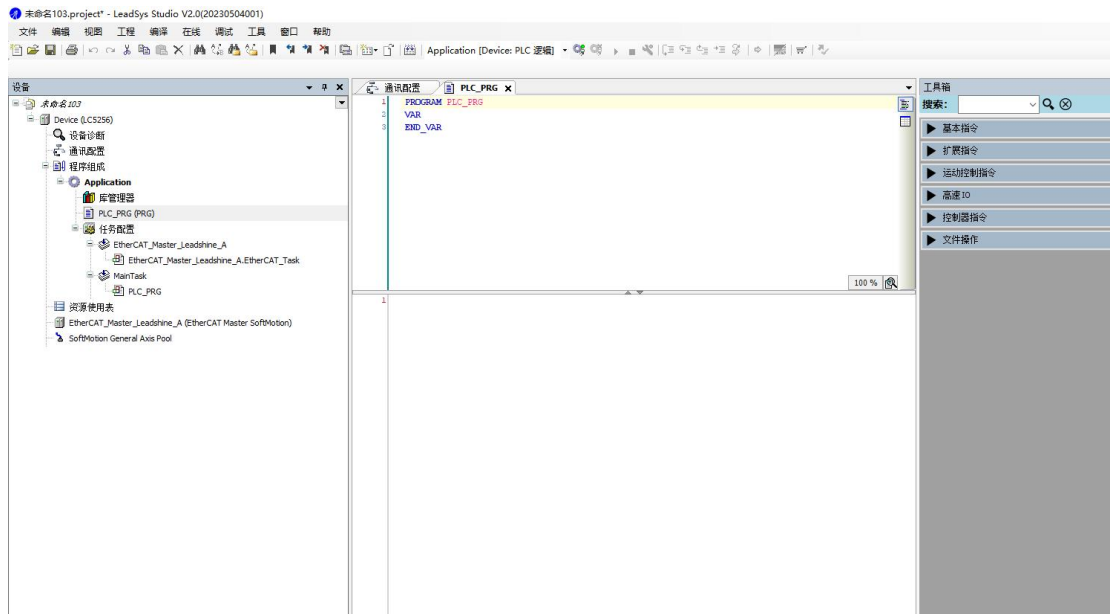


图 2.35 工程界面

3) 网络拓扑的配置

如图 2.35 所示，在工程目录树中已经加入了 LC5256 主机，但 IO 模块和步进电机驱动器都还没有添加。

双击工程目录树中的“Device”，出现“Device”界面如图 2.36 所示。点击“扫描网络”，出现“选择设备”界面，选中“LC5256”后，点“确定”。这时，“网关”和“节点”都显示绿点，表示软件和 LC5256 已连接。

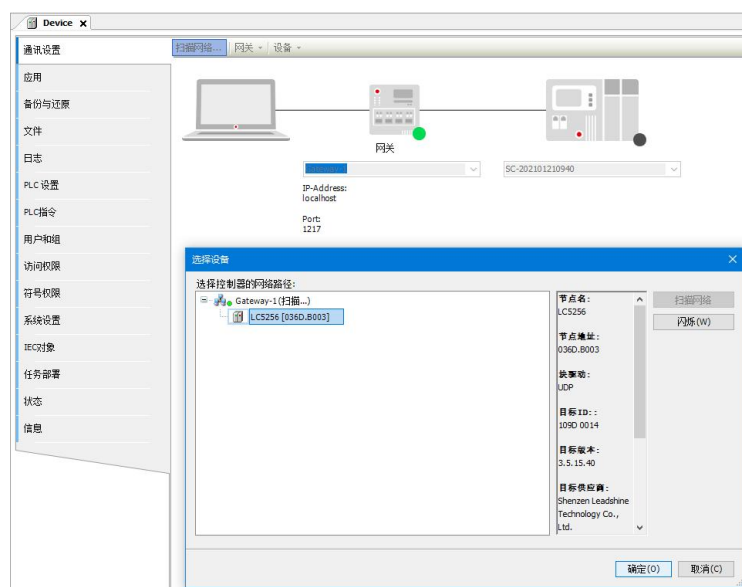


图 2.36 扫描网络，连接 PLC

用鼠标右键点击目录树中的“EtherCAT_Master_Leadshine(EtherCAT Master)”，选择“扫描设备”，如图 2.37 所示。

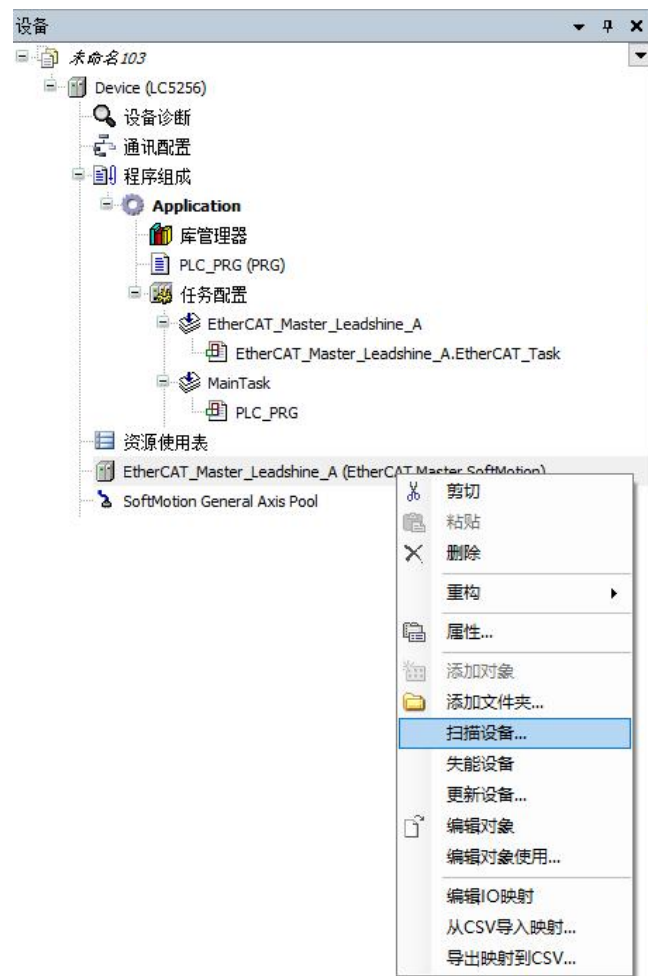


图 2.37 选择“扫描设备”

几秒钟后出现扫描结果如图 2.38 所示。点击按钮“复制所有设备到工程中”，扩展 IO 模块、总线型步进电机驱动器都自动地加入到系统软件中，如图 2.39 所示。

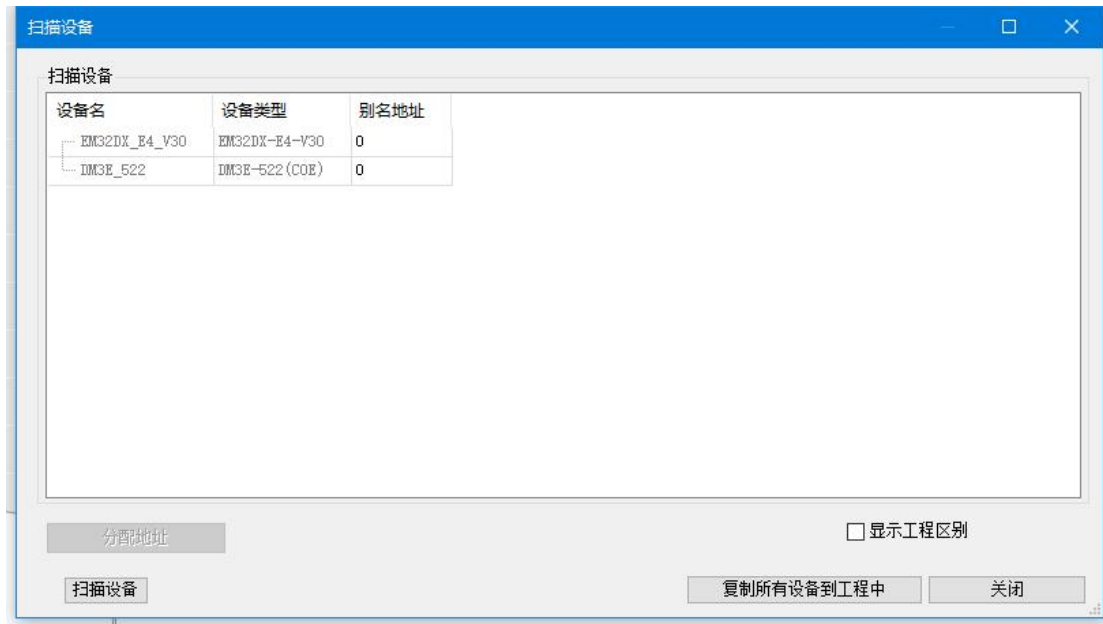


图 2.38 扫描到设备

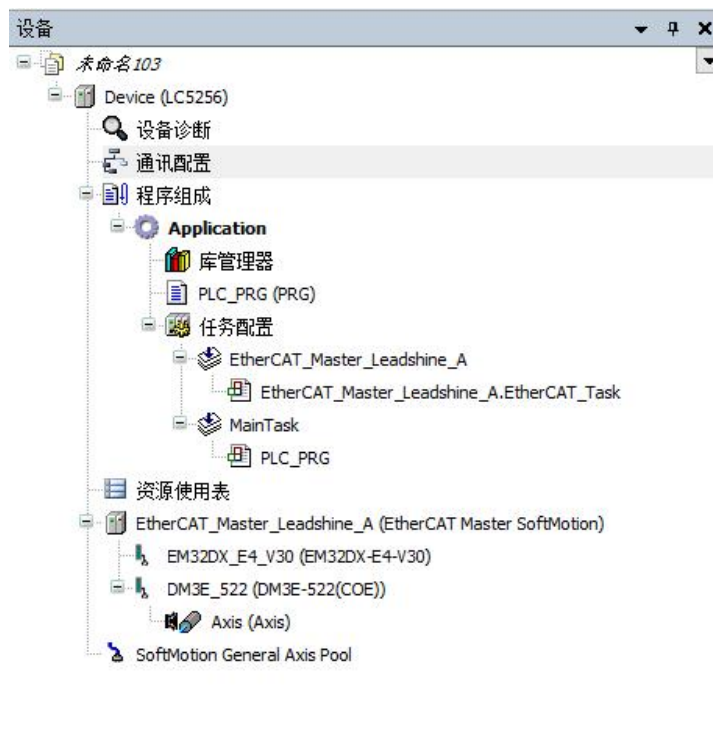


图 2.39 设备添加至工程目录树

4) 编程与 IO 变量的映射

工程树中的“任务配置”自动建立了 EtherCAT 总线 A 口的任务名 EtherCAT_Master_Leadshine_A，在其下自动建立了 1 个相关总线的执行任务：

➤ EtherCAT_Master_Leadshine_A.EtherCAT_Task

用户的程序在“PLC_PRG”中编写。要让用户的程序能正常运行，必须将其放置在 EtherCAT_Master_Leadshine_A 之下。

所以，先用鼠标左键选中图 2.39 中“MainTask”目录下的“PLC_PRG”，将其拖入“EtherCAT_Master_Leadshine_A”总线任务下，然后双击“PLC_PRG”，打开编程界面，在编程区键入代码。PLC_PRG 编辑区的上半部分是“变量定义区”，下半部分是“代码编辑区”，如图 2.40 所示。

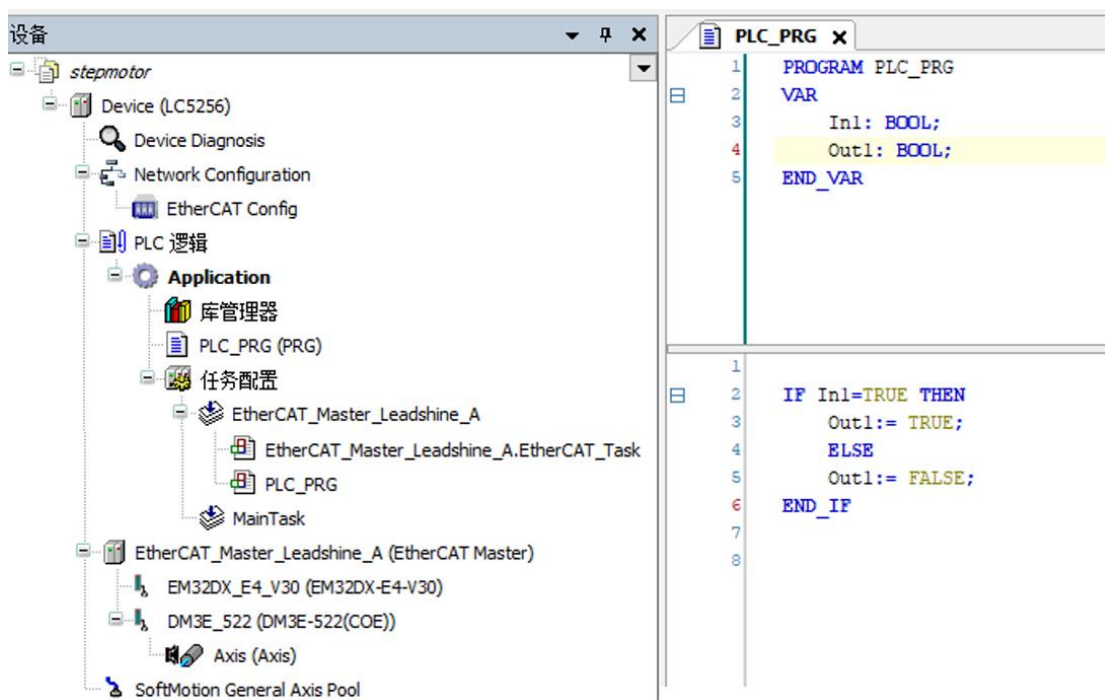


图 2.40 IO 模块的控制代码

双击目录树中的“EM32DX_E4_V30”，进入 IO 设置界面，如图 2.41 所示；点击“EtherCAT I/O 映射”栏，点击“变量”下的加号，显示 IO 模块所有的输出和输入口。

在“通道”栏找到 OUTBit1，将鼠标移至该行“变量”栏中双击，出现符号“...”，如图 2.41 所示；点击该符号，出现“输入助手”界面；依次点击“Application”和“PLC_PRG”前的加号，显示已定义的变量；选择“Out1”后，点击“确定”键，完成输出端口与输出变量的映射。

以同样的方法，可完成输入变量 In1 的映射，参见图 2.42。

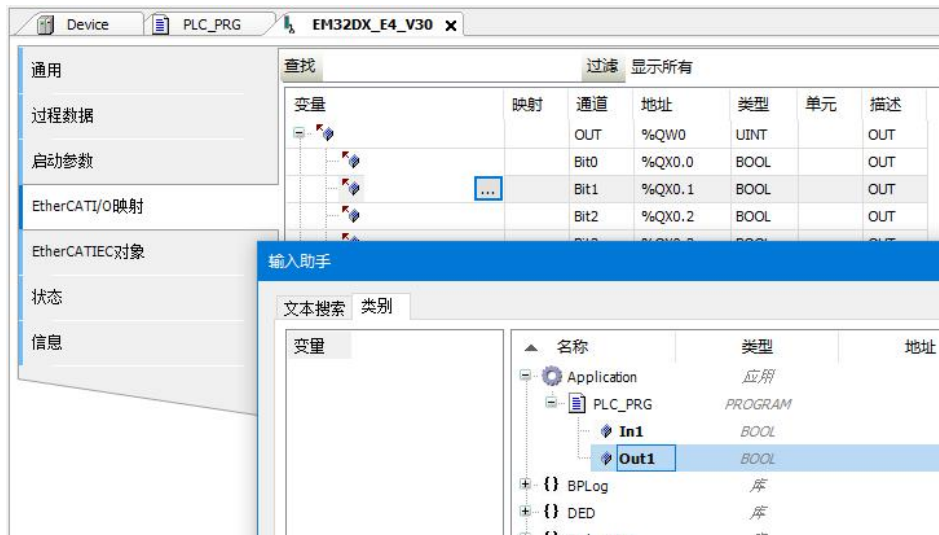


图 2.41 输出变量的映射

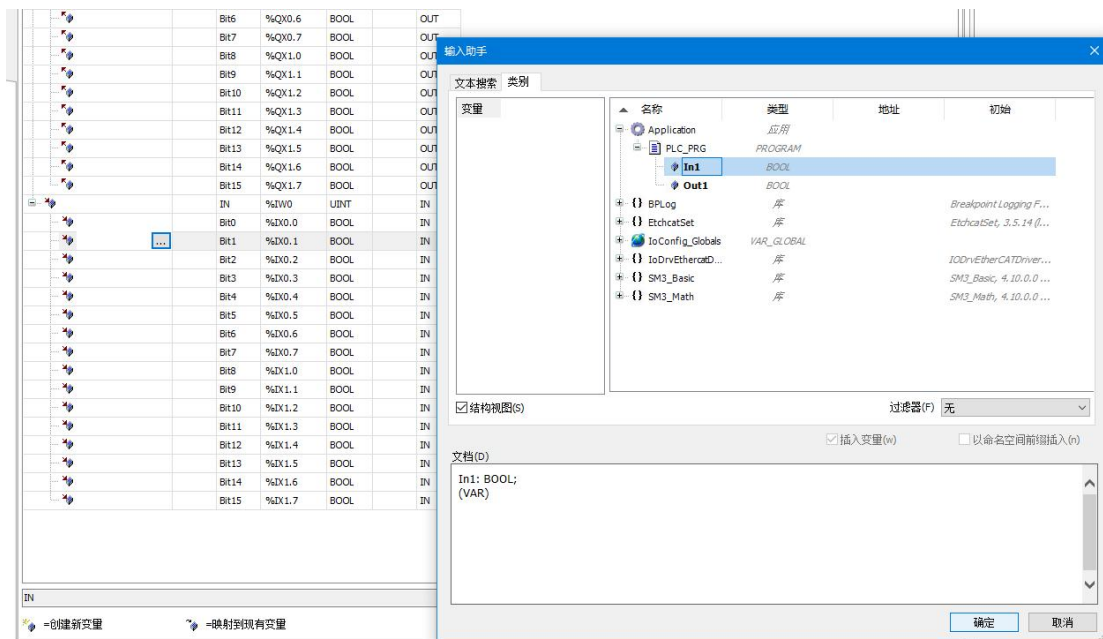


图 2.42 输入变量的映射

5) 编译、运行程序

在工具栏中点击图标“”编译代码；然后，点击图标“”登录 PLC，下载程序；如有对话框弹出，则选择“登录并下载”，如图 2.43 所示。

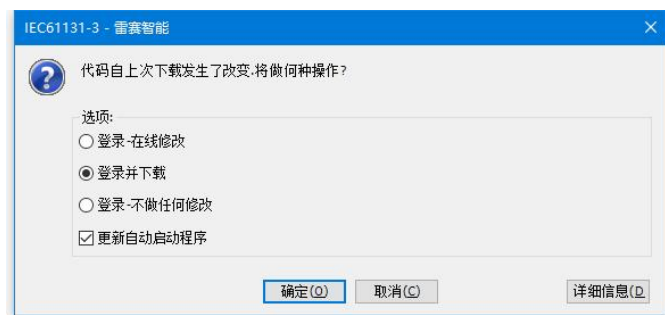


图 2.43 登录并下载程序

点击图标“▶”运行程序。

这时，闭合输入口 In1 上的开关，输出口 Out1 上的 LED 灯会亮；断开 In1 上的开关，Out1 上的 LED 灯熄灭。在此过程中，程序中的变量 In1、Out1 会显示当前值，TRUE 或 FALSE，如图 2.44 所示。

在工具栏中点击图标“■”可使程序结束运行；点击图标“🛑”可退出 PLC 登录，回到程序编辑状态。

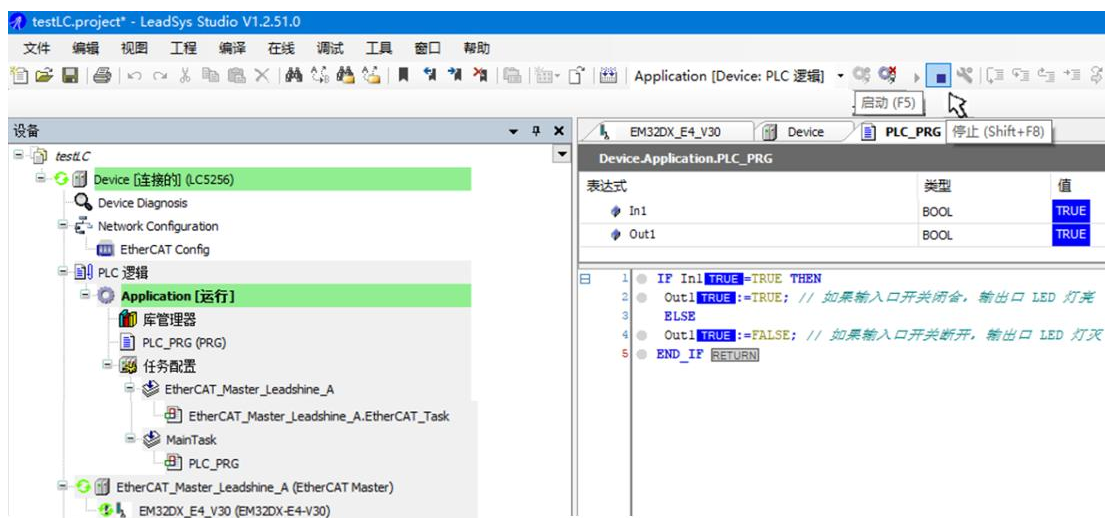


图 2.44 程序运行

2.6.3. 用 LC5000 控制总线型步进电机运动

在上述工程的基础上控制步进电机运动的步骤如下：

1) 设置步进电机的参数

在控制步进电机运动之前，要设置步进电机的参数。最重要的参数就是“指令脉冲数/转”。

用鼠标左键双击项目树中的“DM3E_522”，打开 DM3E_522 的“功能码”栏如图 2.45 所示。选中“功能码编号 Pr.00”，在“写入值”栏填 10000，点击菜单中的“写入选中参数”将参数下载至步进电机驱动器；再点击“读取选中参数”，查看参数是否写入正确。

注意：必须先登录 PLC、启动 PLC，才能读、写电机参数。



图 2.45 设置步进电机参数

再用鼠标左键双击项目树中的“DM3E_522”下面的“Axis”，打开“Axis”参数中的“脉冲当量换算”栏，设“电机旋转一圈的指令脉冲数”为 10000；勾选“不使用变速装置”，设“工作台旋转一圈的工作行程”为 10000，如图 2.46 所示。

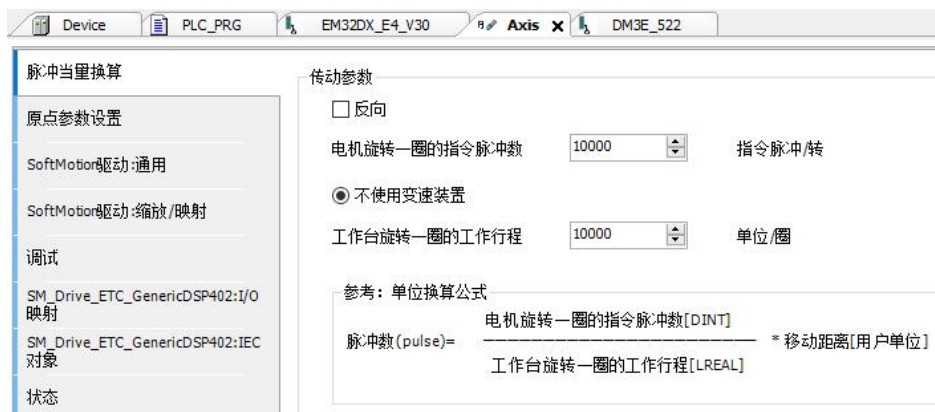


图 2.46 设置脉冲当量

然后，打开“SoftMotion 驱动：通用”栏，“轴类型与限位”选“有限”；“动态限制”下的速度、加速度、减速度、加加速度分别设为 10000、50000、50000、100000，如图 2.47 所示。



图 2.47 设置最大速度和加速度

然后，打开“SoftMotion 驱动：缩放/映射”栏，可以看到“电机类型”为“旋转”；增量“increments”为 10000，电机转数“motor turns”为 1；齿轮输出转数“gear output turns”为 1；应用单位“units in application”为 10000。如图 2.48 所示。这些参数是由“脉冲当量换算”栏设置的参数决定。

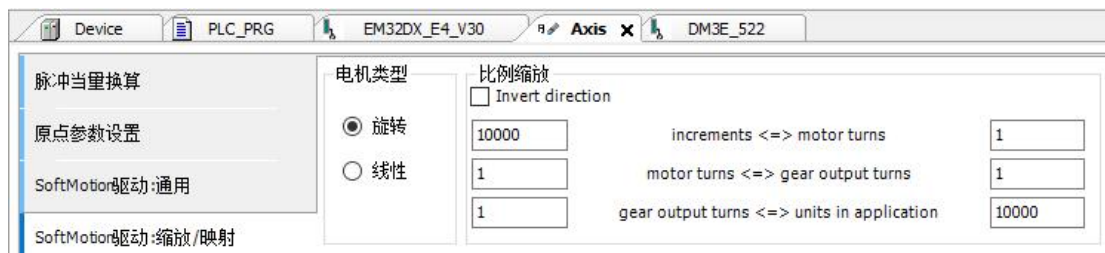


图 2.48 电机类型，脉冲数与输出轴转角关系

这些参数设置完后，电机旋转角度与脉冲数的关系就确定了，即 $360^\circ \leftrightarrow 10000$ 脉冲。

2) 编写程序

先在变量定义区添加 3 个变量，如下所示。

```
InitMotor: MC_Power;           // 电机初始化模块名称
MotorMove: MC_MoveRelative;    // 电机相对运动模块名称
Mstart: BOOL;                  // 控制电机启动的变量
```

在代码编辑区点击右键，选“输入助手”，如图 2.49 所示。在“文本搜索”栏里，输入“MC_Power”，在搜索结果里选中该模块，点“确定”，如图 2.50 所

示。将模块前的名称“MC_Power”改为“InitMotor”，在参数 Axis、Enable、bRegulatorOn、bDriveStart 后填 TRUE。

以同样方法输入 MC_MoveRelative 模块，各参数如下：

轴号 Axis 为 Axis；

启动信号为 Mstart；

运动距离 Distance 为 10000 个脉冲，即转动 1 圈，360 度；

最大速度 Velocity 为 5000 脉冲/秒；

加速度、减速度 Acceleration、Deceleration 均为 20000 脉冲/秒²；

其他参数不用设置，参见图 2.51。

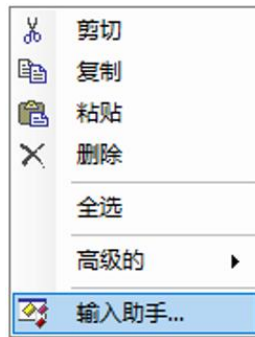


图 2.49 选中“输入助手”

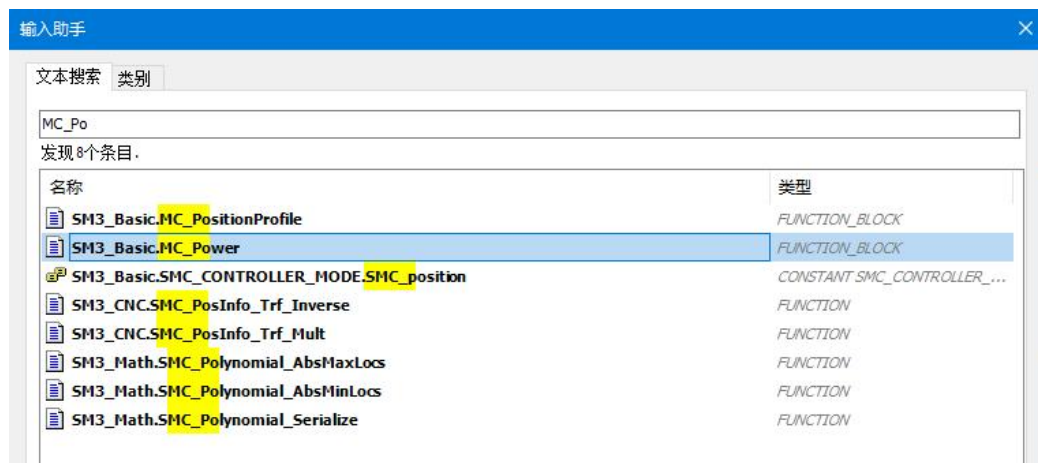


图 2.50 搜索模块



图 2.51 代码编辑界面

完整的代码如下：

```
PROGRAM PLC_PRG
```

```
VAR
```

```
InitMotor: MC_Power; // 电机初始化模块名称
```

```
MotorMove: MC_MoveRelative; // 电机相对运动模块名称
```

```
Mstart: BOOL; // 控制电机启动的变量
```

```
In1: BOOL;
```

```
Out1: BOOL;
```

```
END_VAR
```

```
InitMotor( Axis:= Axis, Enable:= TRUE, bRegulatorOn:=TRUE,
bDriveStart:=TRUE, Status=>, bRegulatorRealState=>, bDriveStartRealState=>,
Busy=>, Error=>, ErrorID=> ); // 电机初始化模块
```

```
IF In1=TRUE THEN
```

```
Mstart:= TRUE; // 开关按下，电机启动
```

```
Out1:= TRUE; // 开关按下，LED 灯亮
```

```
ELSE
```

```
Mstart:= FALSE;
```

```

Out1:= FALSE;

END_IF

MotorMove( Axis:=Axis, Execute:=Mstart, Distance:=10000,

          Velocity:=5000, Acceleration:=20000, Deceleration:=20000,

          Jerk:= , BufferMode:= ,

          Done=> , Busy=> , Active=> , CommandAborted=> ,

          Error=> , ErrorID=> );// 电机相对运动模块
    
```

编译程序后登录控制器、下载程序，运行程序，当项目树中的 Axis 轴前面的图标变为绿色时，如图 2.52 所示，可以开始控制电机运动。开关闭合一次，电机转动 1 圈，时间约 2 秒。

3) 程序独立运行

当一个应用程序在 PC 机上用 LeadSys 软件编写、下载、调试完成后，该程序已经保存在 PLC 的 Flash 中，调电后不会消失。

拔掉 PLC 与 PC 机连接的网线，PLC 开机后，就开始独立运行该应用程序。

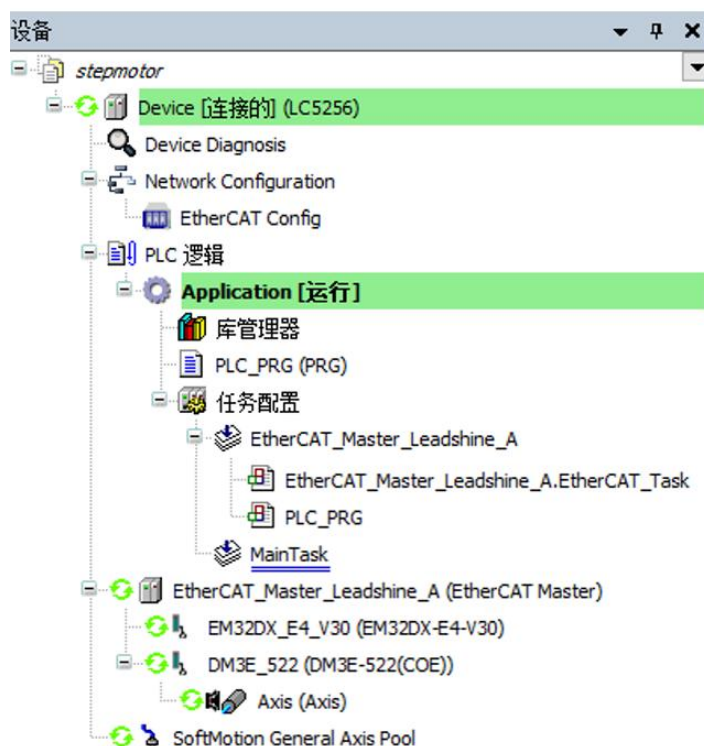


图 2.52 电机已使能

2.7. PMC600 控制器例程

2.7.1. PMC600 实验系统硬件

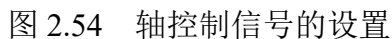
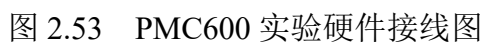
PMC600 系列 PLC 实验系统的硬件如图 2.53 所示。该系统由一台 PC 机、一台 PMC600 控制器、一个脉冲式步进电机驱动器及电机、2 个按钮组成。

2.7.2. 电机接口及参数的设置

设置脉冲式电机接口之前，参照 2.6 节的内容，以相同的方法和步骤链接电脑与 PMC600、新建工程、选择控制器型号和编程语言、扫描网络连接控制器。

然后，双击目录树中的“High_Speed_IO_Module”，进入 IO 设置界面；点击“硬件端口配置 1”栏，显示如图 2.54 所示界面。点击轴 0 前的方框，出现符号“√”，即选中轴 0，输出口 Out00、Out01 为电机的脉冲和方向信号；IN00、IN01、IN02 为原点信号、负限位和正限位信号。

注意：在脉冲、方向信号线中要串联一个 2k 限流电阻，以避免步进电机驱动器输入的光耦因电流过大而损坏。



点击“轴参数设置”栏，如图 2.55 所示界面，设置轴的参数。

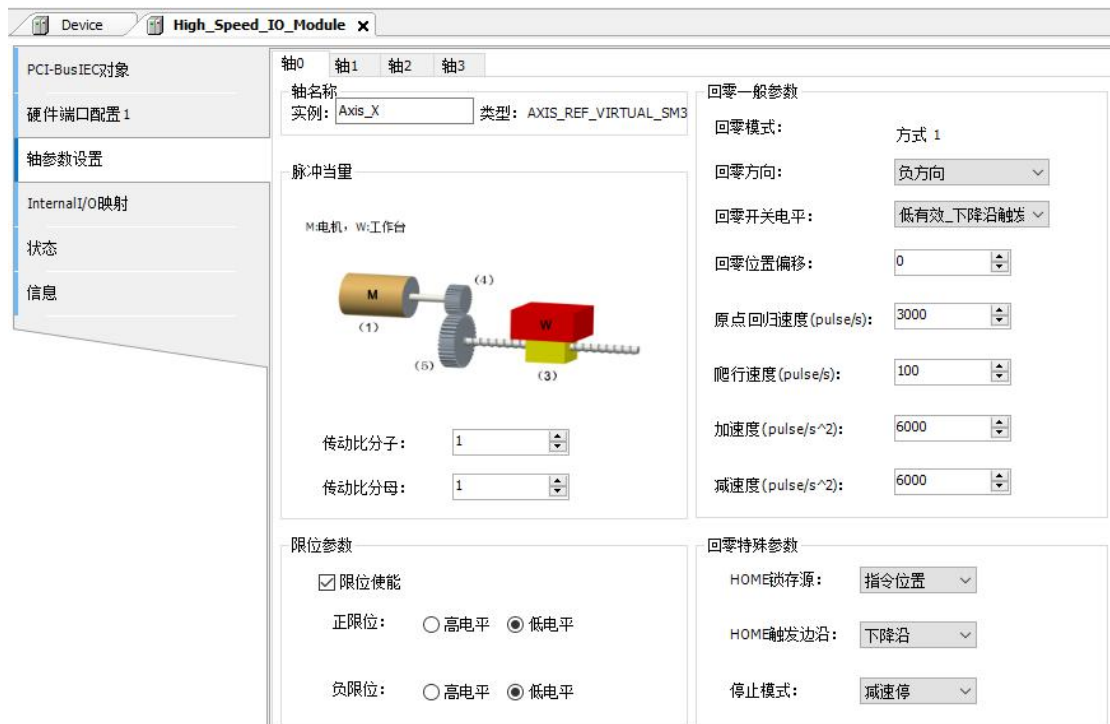


图 2.55 轴参数设置界面

双击目录树中的“SoftMotionGeneralAxisPool”下的轴号“LS_Axis_0”，设置该轴的参数，如图 2.56 所示。

单击目录树中的“SoftMotionGeneralAxisPool”下的轴号，可以修改轴的名称，如图 2.57 所示。



图 2.56 “SoftMotionGeneralAxisPool”中的轴参数设置界面



图 2.57 修改轴的名称

2.7.3. 编写控制电机运动的程序

1) 轴初始化

除了完成上述电机参数设置外，还要在程序中初始化脉冲轴，即调用子程序 LS_PulseAxis_MachineState。该子程序要求设置 4 个轴，所有还要再添加 3 个虚拟轴。

右键点击目录树中的“SoftMotionGeneralAxisPool”，选择“添加设备”，打开如图 2.58 所示的界面，填写设备名称，选取“虚拟驱动器”，点“添加设备”按钮即可。然后按此方法再添加 2 个虚拟轴。

在工程目录树中，用鼠标右键单击“PLC_PRG”，选“添加对象”→“动作”，如图 2.59 所示。所谓“动作”，可以理解为子程序。

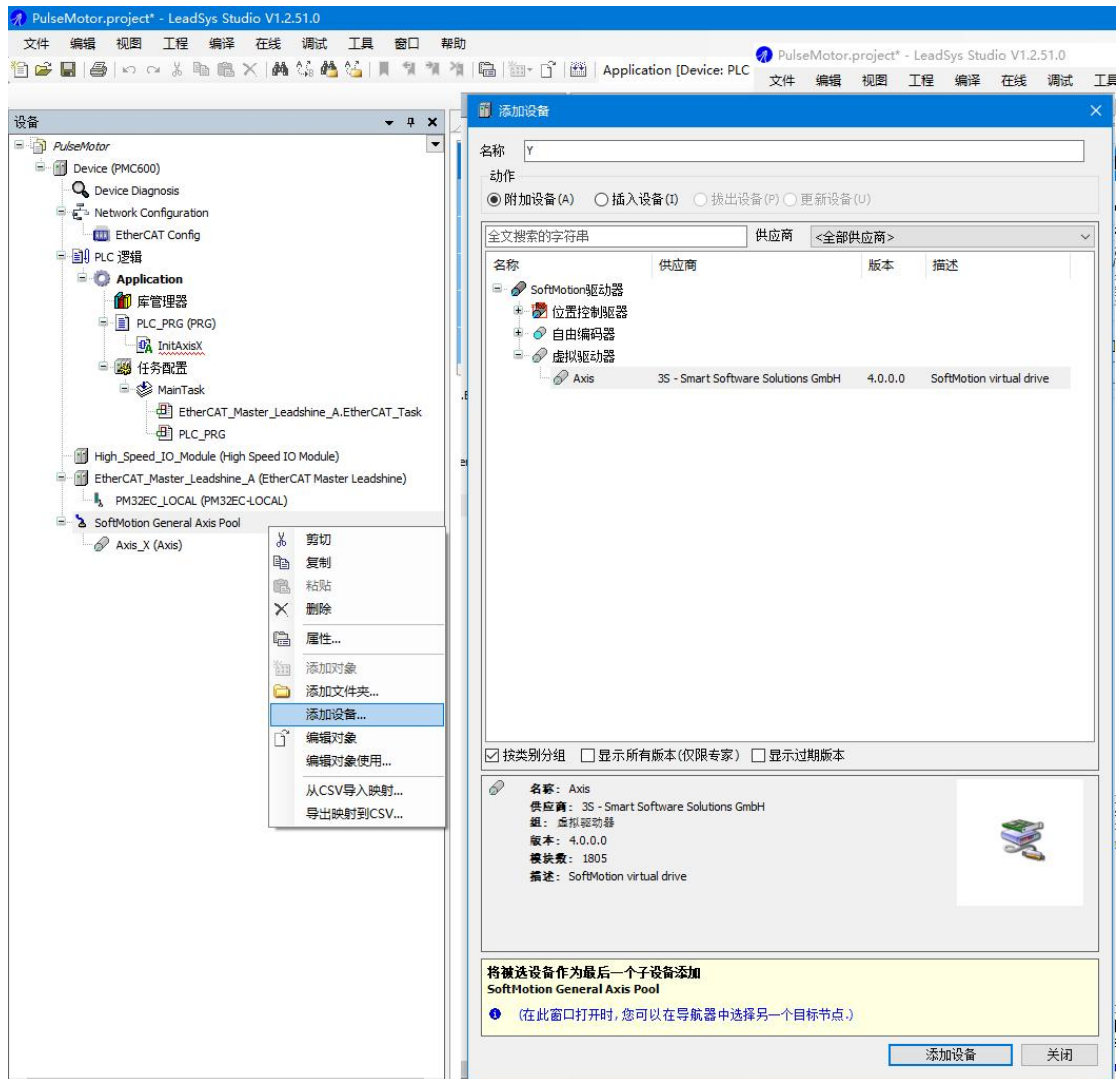


图 2.58 添加虚拟轴

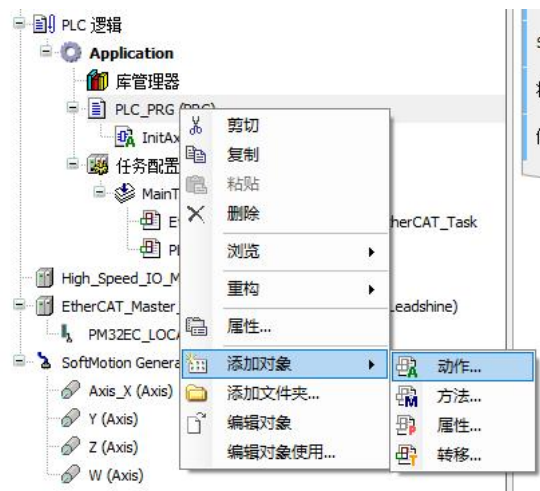


图 2.59 添加动作

设置新动作的名称、选择实现语言，如图 2.60 所示。

在功能块编辑区里点击鼠标右键，选“插入运算块”，如图 2.61 所示。在输入助手中的“文本搜索”栏里，输入“LS_PulseAxis_MachineState”，在搜索结果里选中该子程序，如图 2.62 所示，然后点击“确定”。

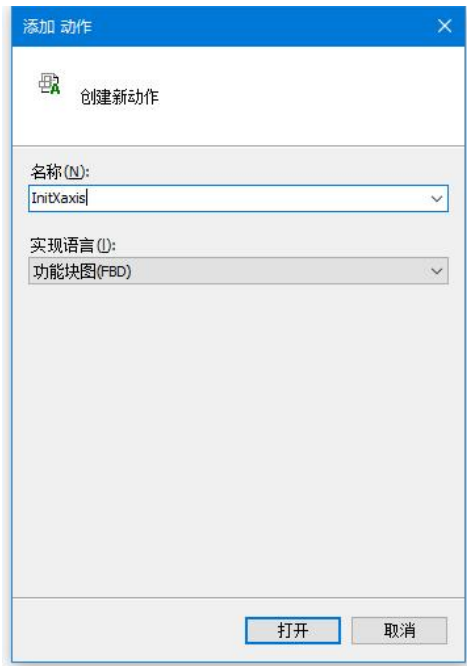


图 2.60 设置“动作”的名称

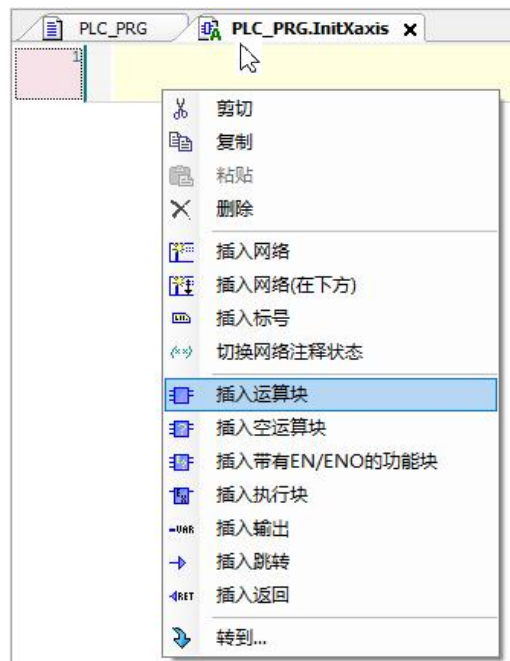


图 2.61 插入运算块

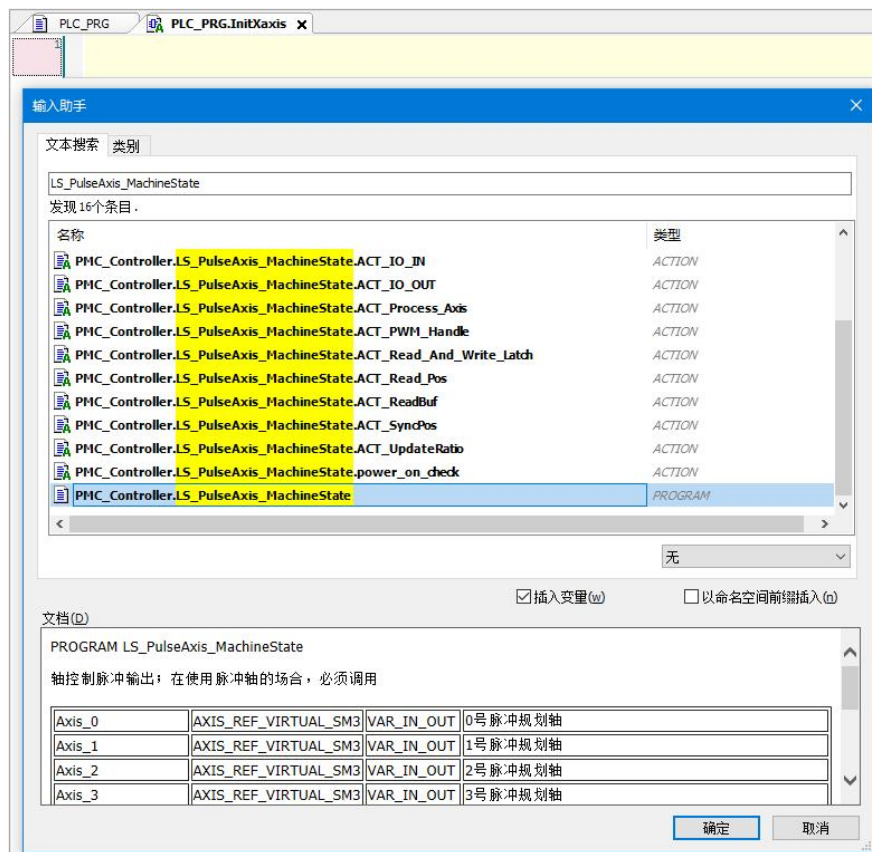


图 2.62 搜索要插入的模块

按照图 2.63 所示，设置 LS_PulseAxis_MachineState 的参数。有问号处，可用退格键清除。LS_PulseAxis_MachineState 是程序不是功能块，故不需要声明模块名称。



图 2.63 搜索要插入的模块

在已调用的模块下方点击鼠标右键，选择“插入网络”，如图 2.63 所示。然后添加电机使能模块 MC_Power；并自动声明模块的名称，如图 2.64 所示。

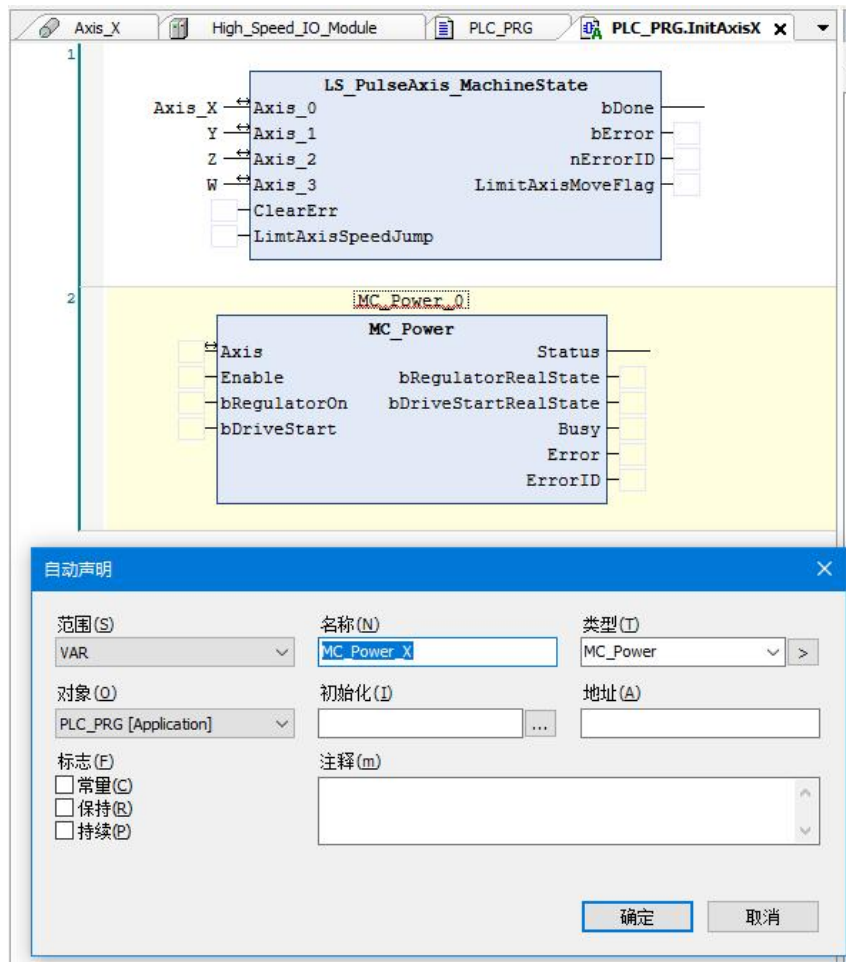


图 2.64 添加新模块并自动声明名称

完成的功能块程序 InitAxisX，如图 2.65 所示。

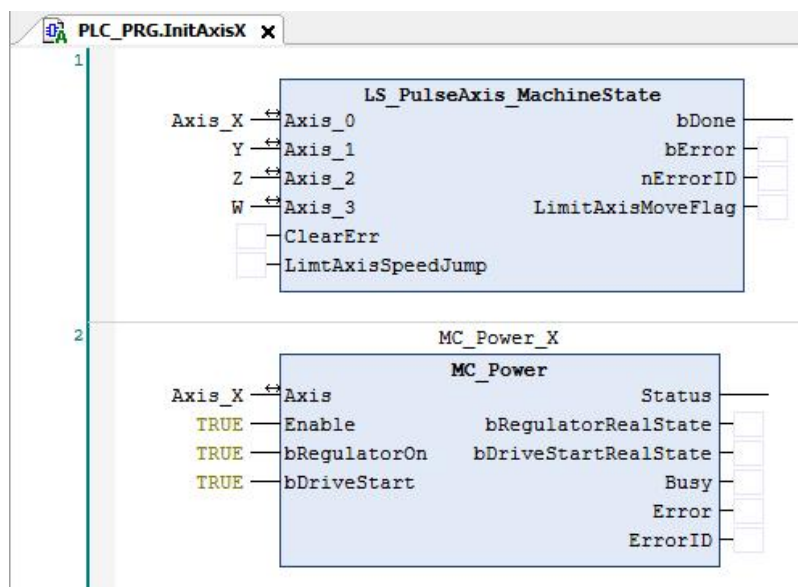


图 2.65 功能块程序 InitAxisX

2) 调用点动模块

用“输入助手”在 PLC_PRG 编辑区中输入点动模块“MC_Jog”，并设置参数；详见下面的程序代码。

3) 输入变量的映射

在“变量定义区”中定义 2 个输入变量 In6、In7。双击目录树中的“High_Speed_IO_Module”，进入“Internal I/O 映射”栏，然后映射变量。其方法和 2.6.2 节所述扩展 IO 模块设置方法一样。

4) 程序代码

通过输入口的 2 个按钮 In6 和 In7 可控制步进电机正转和反转。完整的代码如下：

```
PROGRAM PLC_PRG
VAR
    MC_Power_X: MC_Power;
    JogX: MC_Jog;
    JogFdX: BOOL; // X 轴正转启动信号
    JogBdX: BOOL; // X 轴反转启动信号
    In6: BOOL; // 按钮 6
    In7: BOOL; // 按钮 7
END_VAR

InitAxisX(); // 初始化 X 轴并使能

IF In6=TRUE THEN // 如果按钮 6 按下
    JogFdX:=TRUE; // X 轴正转
ELSE
    JogFdX:=FALSE; // 否则，X 轴不转
```


END_IF

IF In7=TRUE THEN // 如果按钮 7 按下

JogBdX:=TRUE; // X 轴反转

ELSE

JogBdX:=FALSE; // 否则，X 轴不转

END_IF

JogX(Axis:=Axis_X ,JogForward:=JogFdX ,JogBackward:=JogBdX ,
Velocity:=2000 ,Acceleration:=10000 ,Deceleration:=10000 ,Jerk:= ,
Busy=> ,CommandAborted=> ,Error=> ,ErrorId=>);

5) 运行程序

编译程序后登录控制器、下载程序，运行程序，当项目树中的各轴前的图标都变为绿色时，如图 2.66 所示，可以开始控制电机运动。

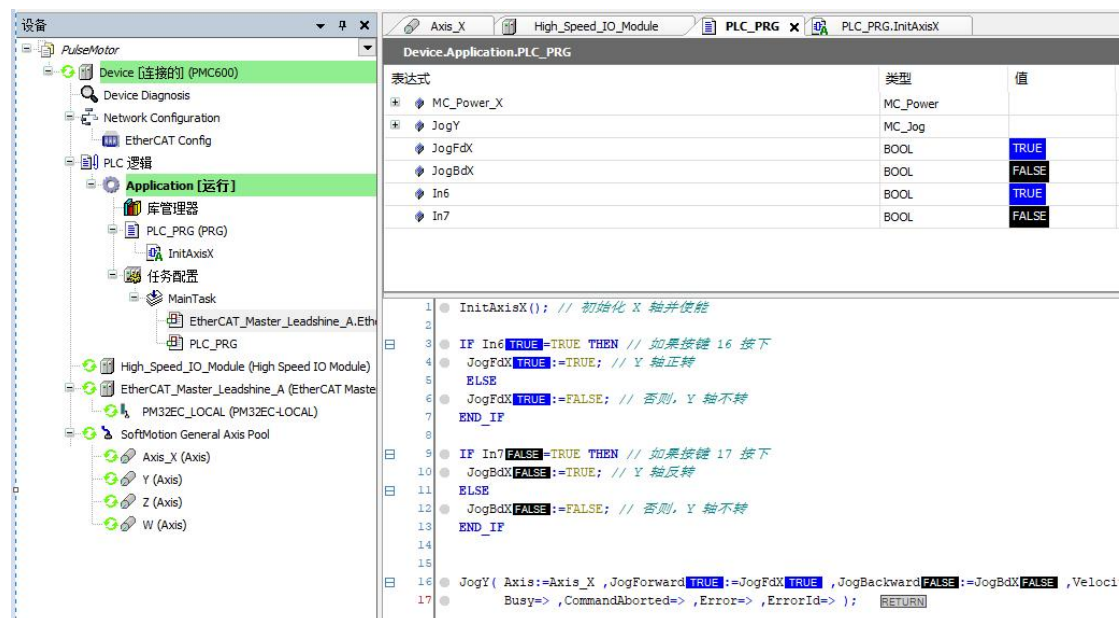


图 2.66 例程二运行结果

2.8. MC500 控制器例程

2.8.1. MC500 实验系统硬件

MC500 系列 PLC 实验系统的硬件如图 2.67、2.68 所示。该系统由一台 PC 机、一台 MC516 控制器、两个 L7EC-400S 型交流伺服电机驱动器及两个交流伺服电机、一个 R2EC 耦合器模块、一个 EM32DX-C1 扩展模块及一台 LT2070E 触摸屏组成。

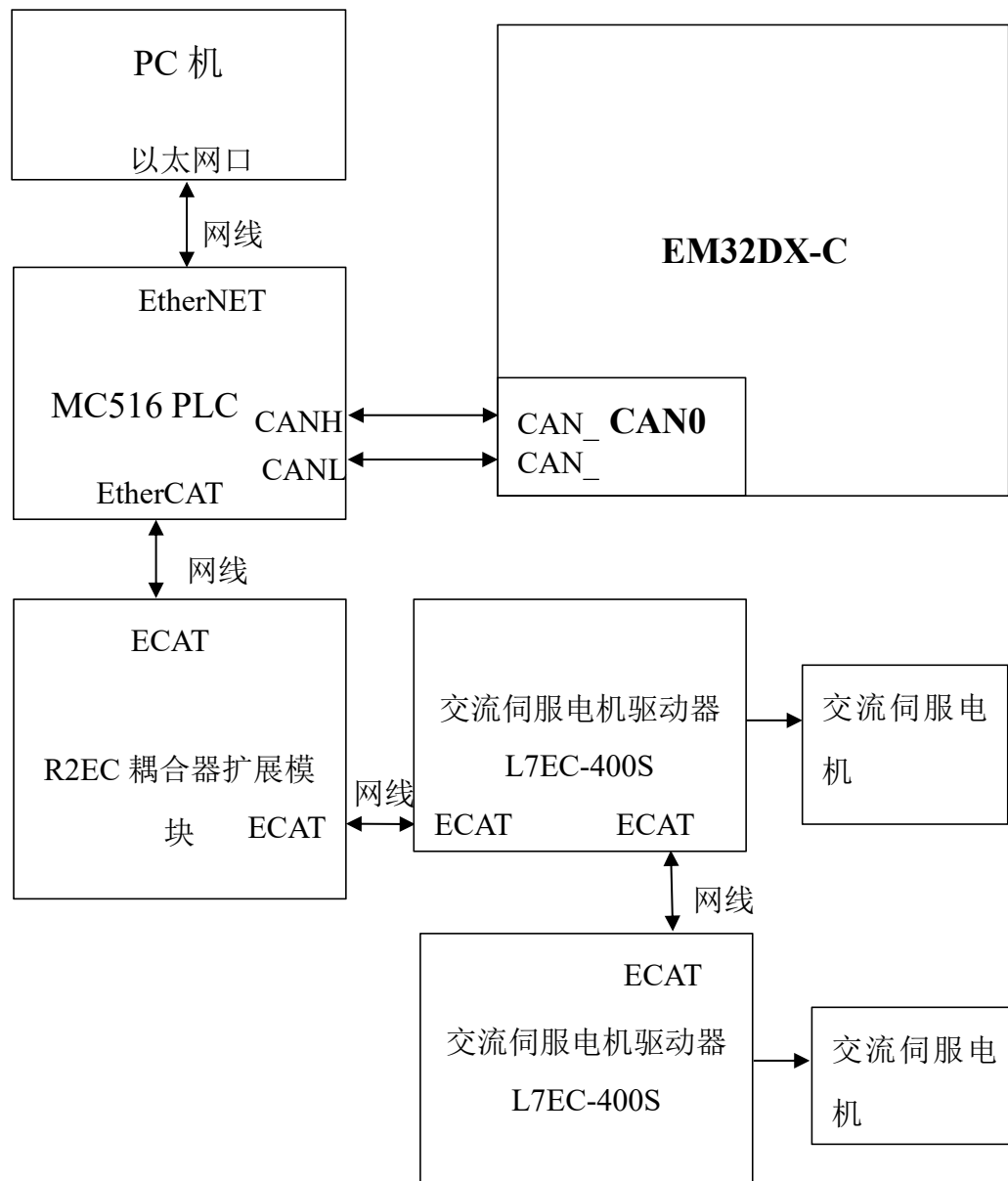


图 2.67 LC5000 实验硬件接线图 1

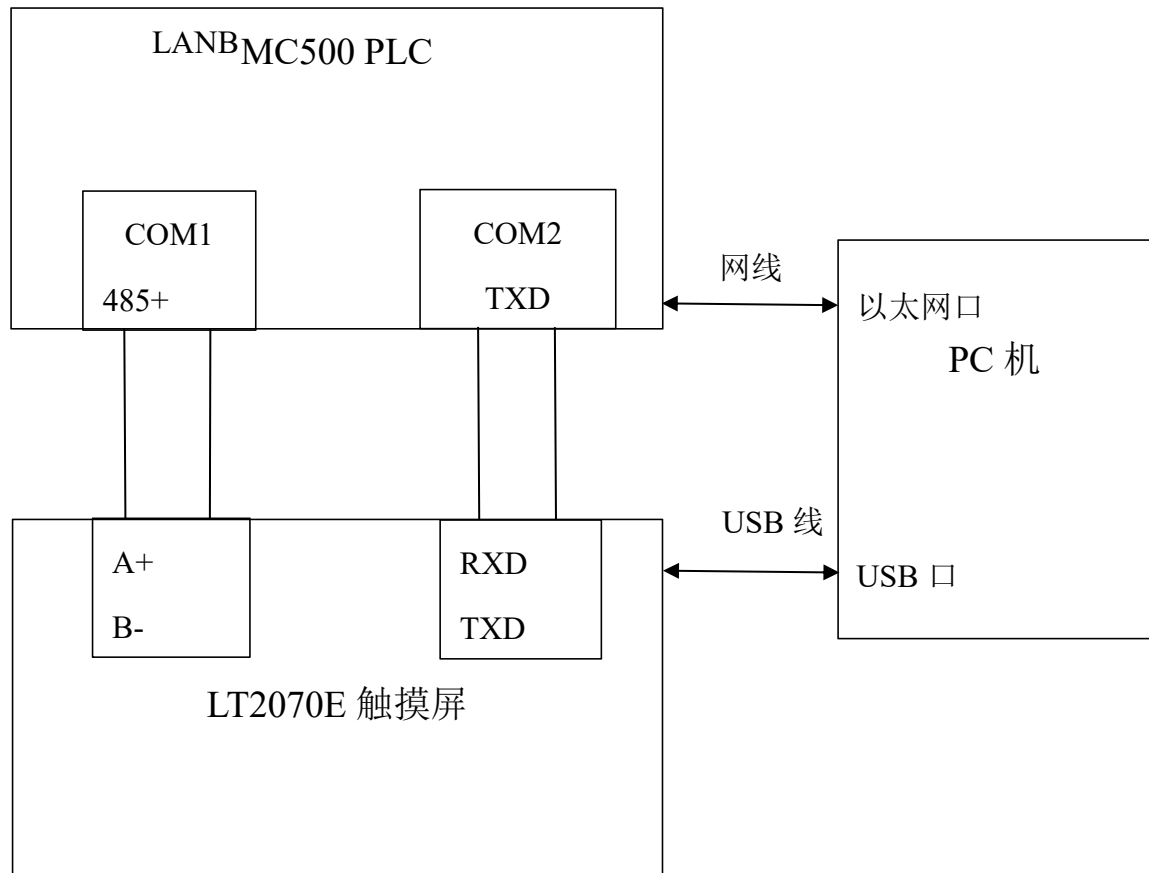


图 2.68 MC500 实验硬件接线图 2

2.8.2. 用 LT2070E 和 MC500 控制总线型伺服电机运动

扫描设备之前，参照 2.6 节的内容，用网线连接 PC 机和 MC500 的 EtherNET 口，再以相同的方法和步骤连接电脑与 MC516、新建工程、选择控制器型号和编程语言、扫描网络连接控制器。

注意：

MC500 系列 PLC 的 EtherNET 口的出厂默认 IP 地址为：192.168.1.3；

PC 机上的 IP 地址前 3 段要与 PLC 的相同，最后一个不能相同！

参照 2.6 节的内容，扫描并添加“EtherCAT_Master_Leadshine(EtherCAT Master)”下的设备。

当用户需要增加 IO 口数量，可扩展 IO 模块，以下介绍如何在软件中添加已插入到 PLC 的右扩展模块。

用鼠标右键点击目录树中的“LocalBus_Master_Module”，选择“背板扫描”，如图 2.69 所示。

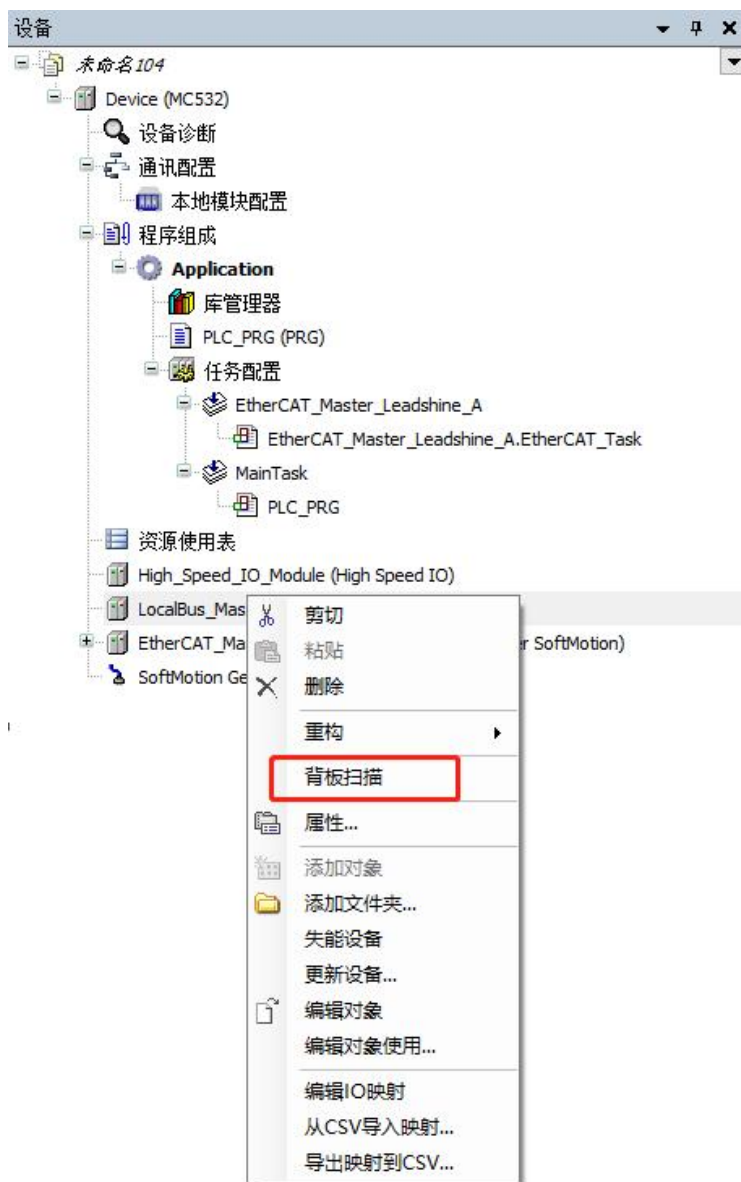


图 2.69 选择“背板扫描”

几秒钟后出现扫描结果如图 2.70 所示。点击按钮“复制所有设备到工程中”，扩展 IO 模块都自动地加入到系统软件中，如图 2.71 所示。

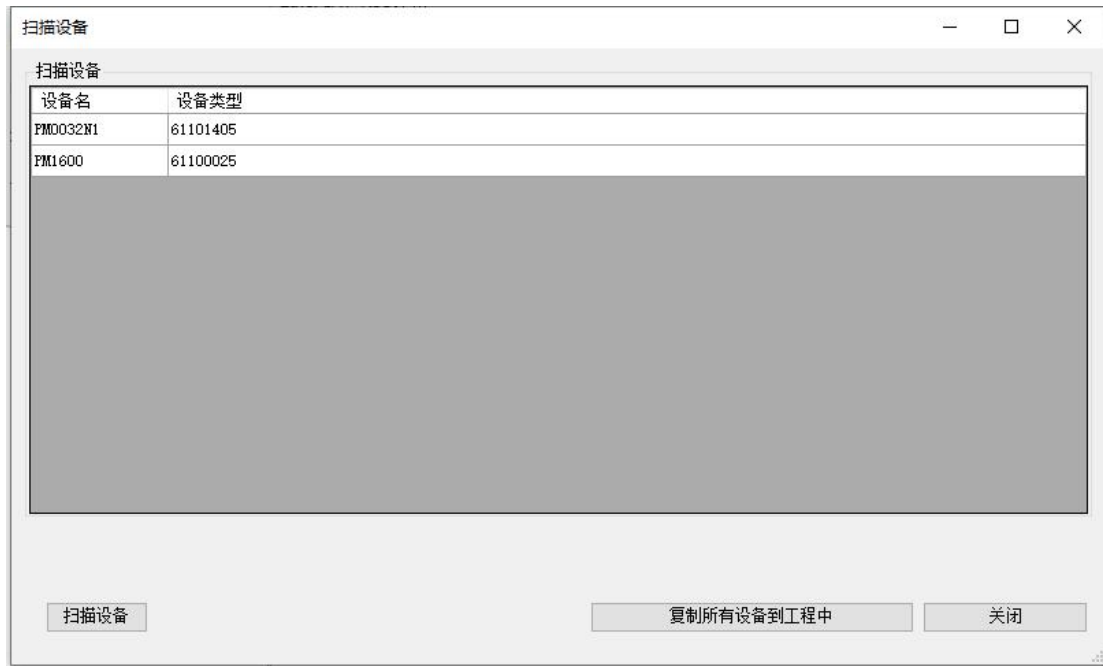


图 2.70 扫描到设备

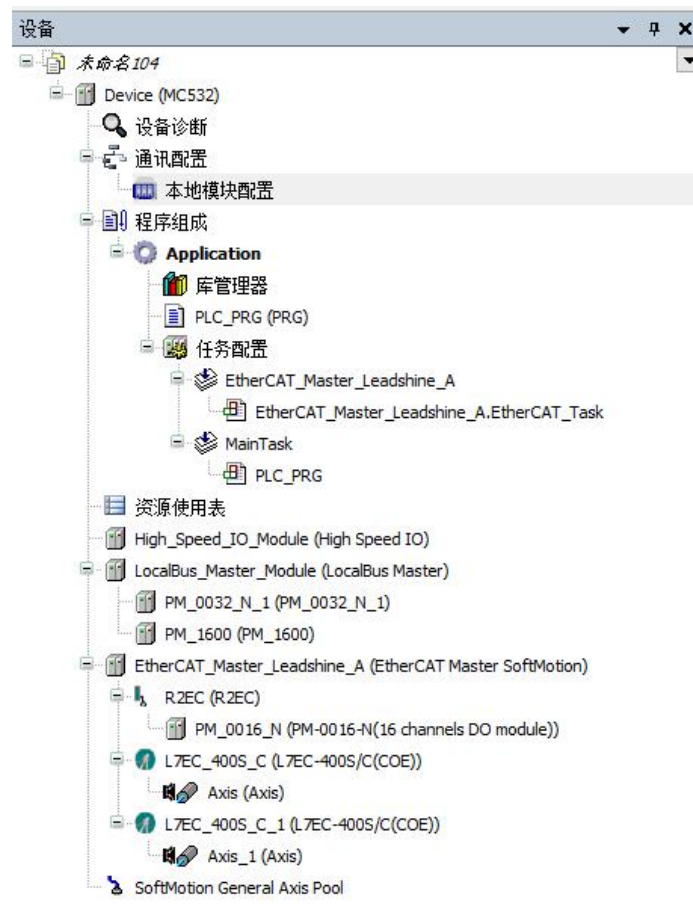


图 2.71 设备添加至工程目录树

用鼠标左键双击目录树中的“通讯配置”，出现界面如图 2.72 所示。

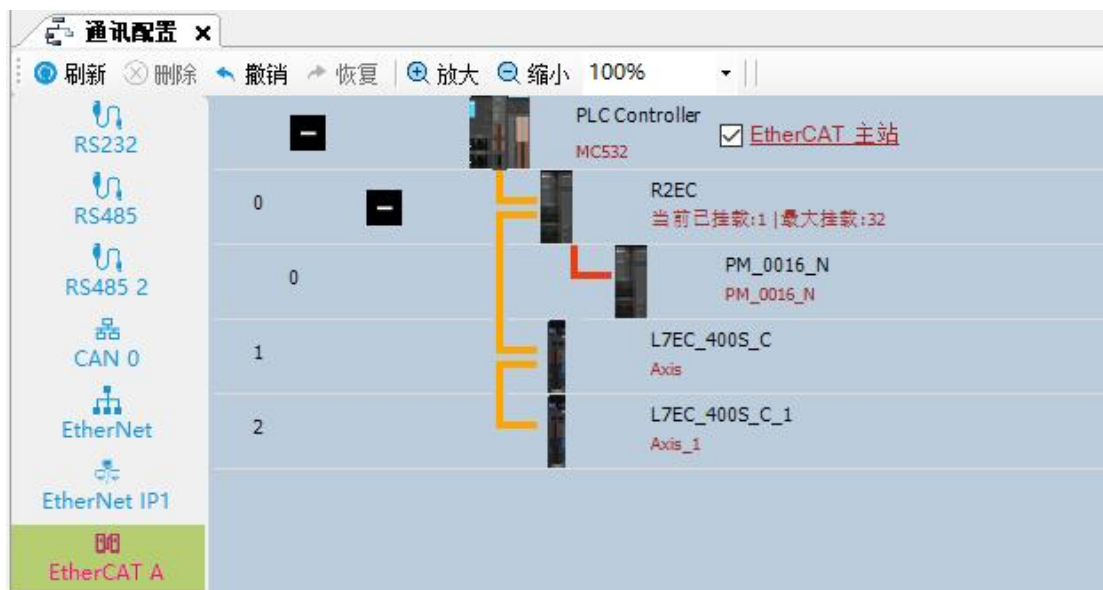


图 2.72 添加从站

MC500CS 控制器支持通过组态配置 CAN 总线、232 总线、485 总线，在主站上进行勾选即可将 CAN 主站（CANBus 设备以及下一级设备 CANOpen Manager、Modbus_Slave_RS485_COM2、Modbus_Slave_RS232_COM）添加到设备目录中。如下图所示：用鼠标左键单击红框内设备，勾选图中的 Modbus 从站与 CANopen 主站，如图 2.73 所示。

PLC 作为 RS232 串口及 RS485 串口的 Modbus 从站，CAN 口的 CANopen 主站。

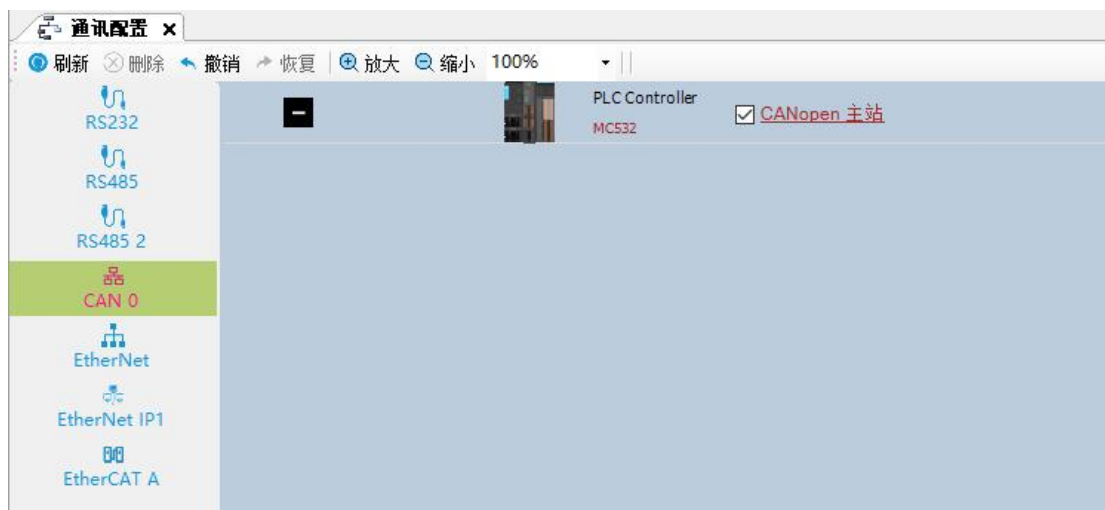


图 2.73 配置总线

用鼠标右键点击目录树中的“CANopen_Manager”，选择“扫描设备”，如图 2.74 所示。

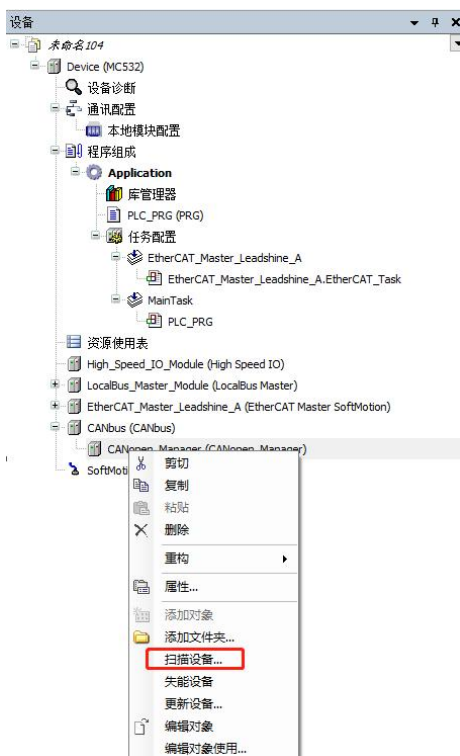


图 2.74 选择“扫描设备”

几秒钟后出现扫描结果如图 2.75 所示。点击按钮“复制所有设备到工程中”，之后 EM32DX-C1 耦合器模块便加入到系统软件中，如图 2.76 所示。

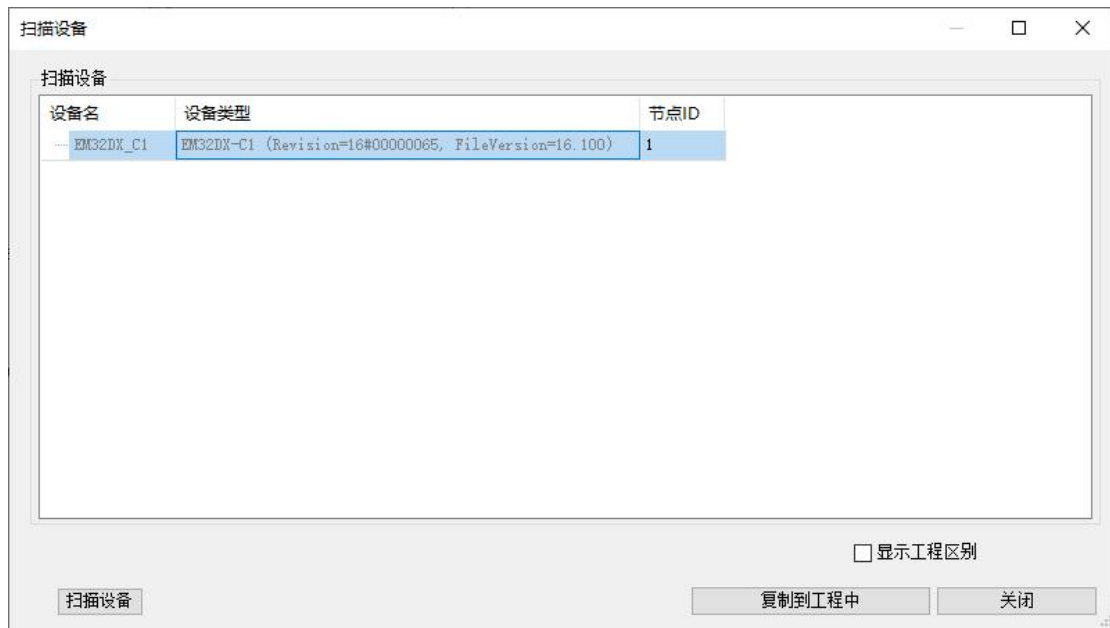


图 2.75 扫描到设备

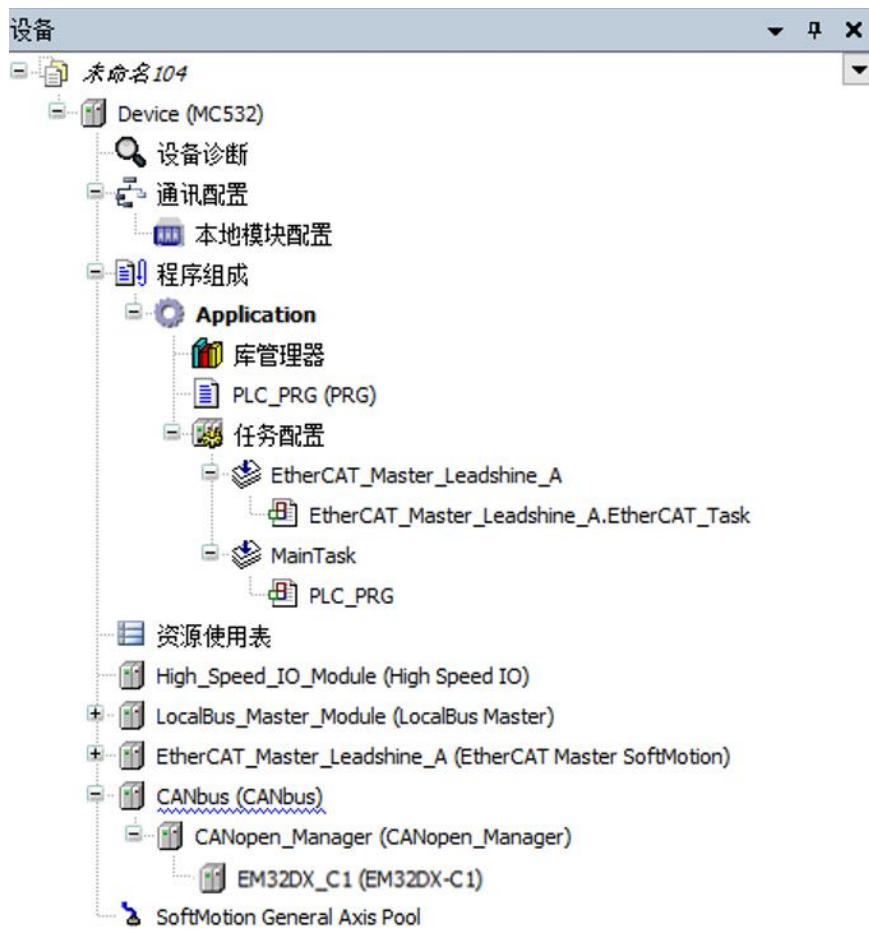


图 2.76 设备添加至工程目录树

1) 编程与 IO 变量的映射

双击目录树中的“High_Speed_IO_Module”，进入 IO 设置界面，点击“InternalI/O 映射”栏，点击通道“Genral_IOOut”该行“变量”栏下的加号，找到通道栏里的 Bit0 和 Bit1，将鼠标移至该两行“变量”栏中双击，输入变量名 Out0 和 Out1 回车，将这两个变量与后面的“地址”关联，如图 2.77 所示。

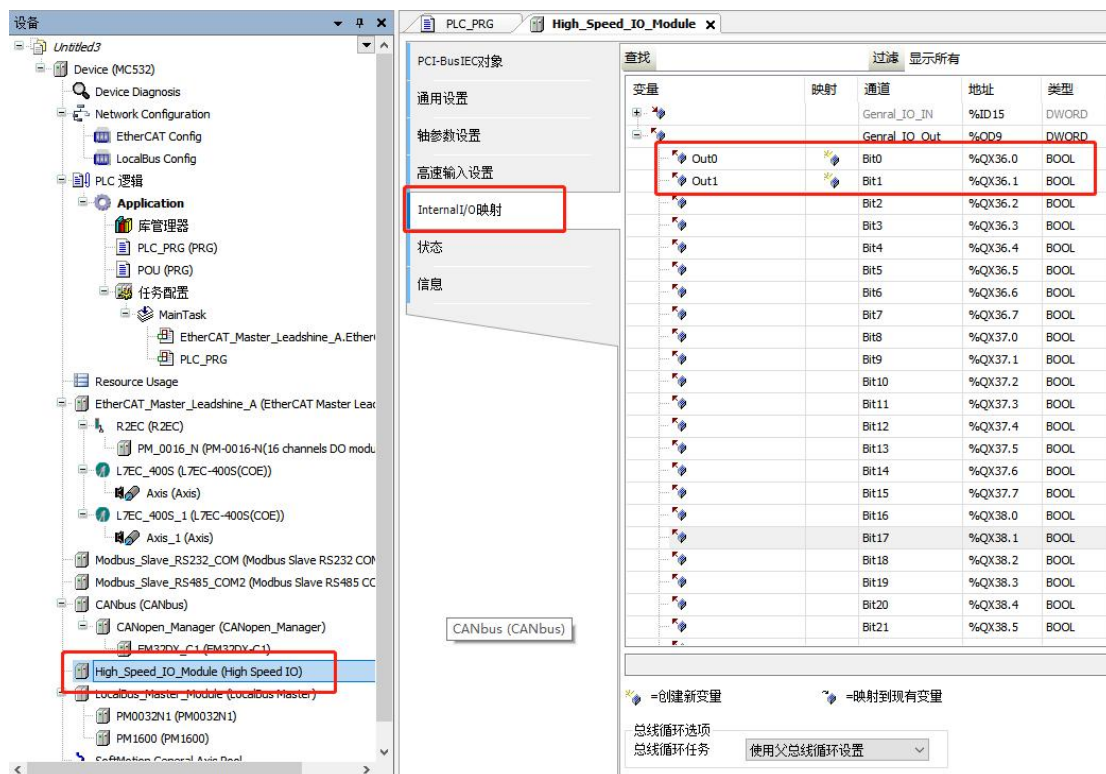


图 2.77 IO 变量映射

在目录树中，用鼠标右键单击“PLC_PRG”，选“添加对象”“动作”，如图 2.78 所示。

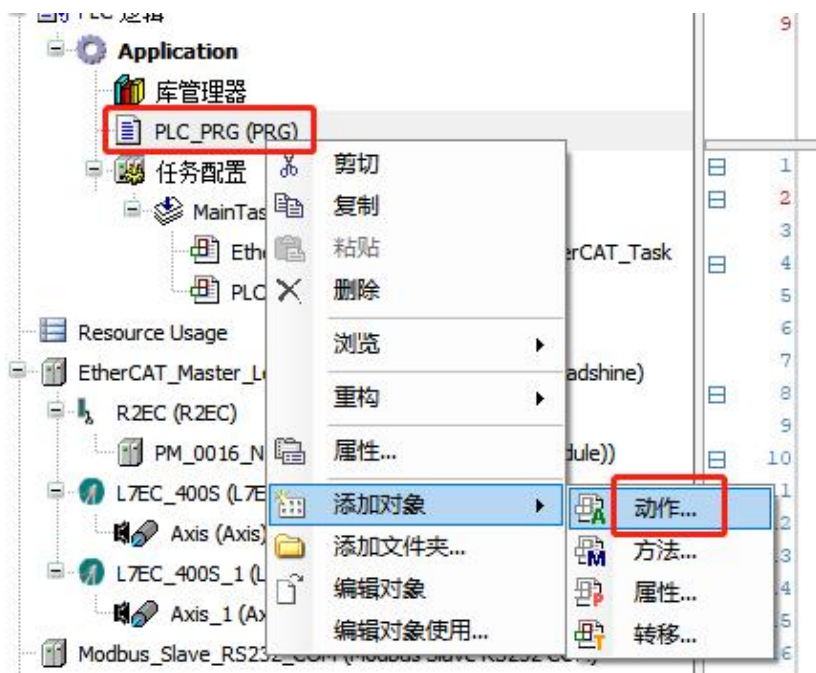


图 2.78 添加动作

参照 2.7 节的内容，设置新动作的名称、选择实现语言、添加电机使能模块 MC_Power，并命名模块名称为 InitMotor0 与 InitMotor1。修改模块 Axis 参数为之前添加的轴，将 Enable、bRegulatorOn、bDriveStart 参数置 True。

完成后的“动作”InitXaxis 功能块程序如图 2.79 所示。

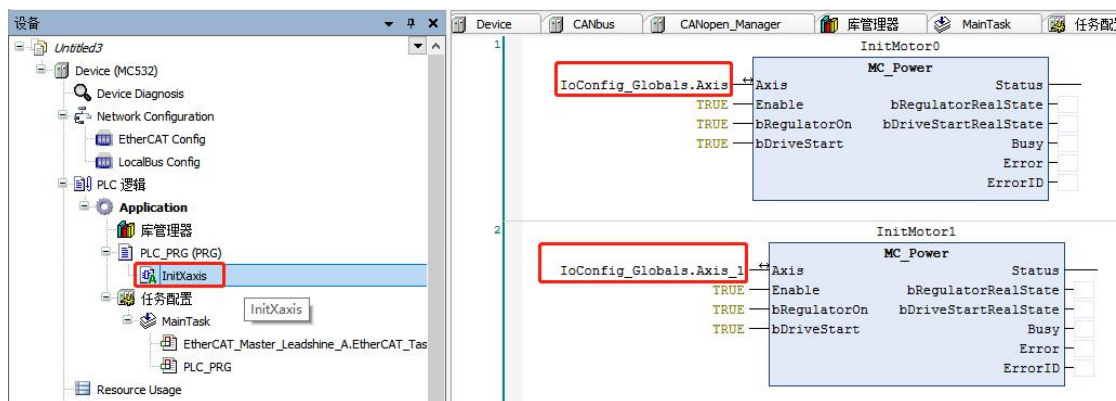


图 2.79 功能块程序 InitXaxis

再用鼠标左键双击项目树中的“L7EC_400S”下面的“Axis”，打开“Axis”参数中的“脉冲当量换算”栏，设“电机旋转一圈的指令脉冲数”为 36000；勾选“不使用变速装置”，设“工作台旋转一圈的工作行程”为 360，如图 2.80 所示。同理，设置“Axis_1”轴为相同的传动参数。

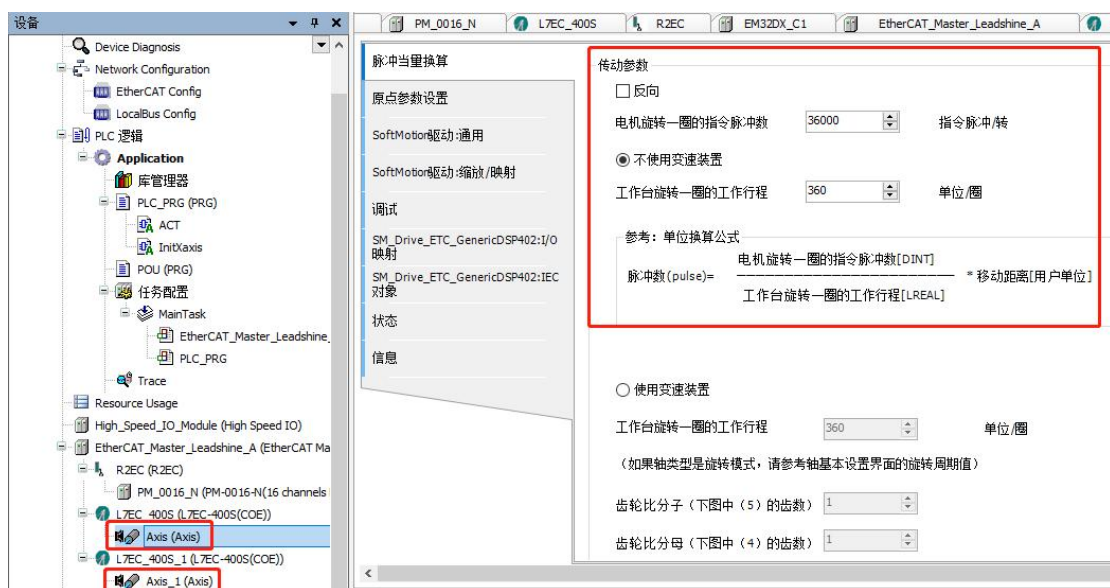


图 2.80 传动参数设置

然后，打开“SoftMotion 驱动：通用”栏，“轴类型与限位”选“有限”；“动态限制”下的速度、加速度、减速度、加加速度分别设为 10000、50000、50000、100000，如图 2.81 所示。

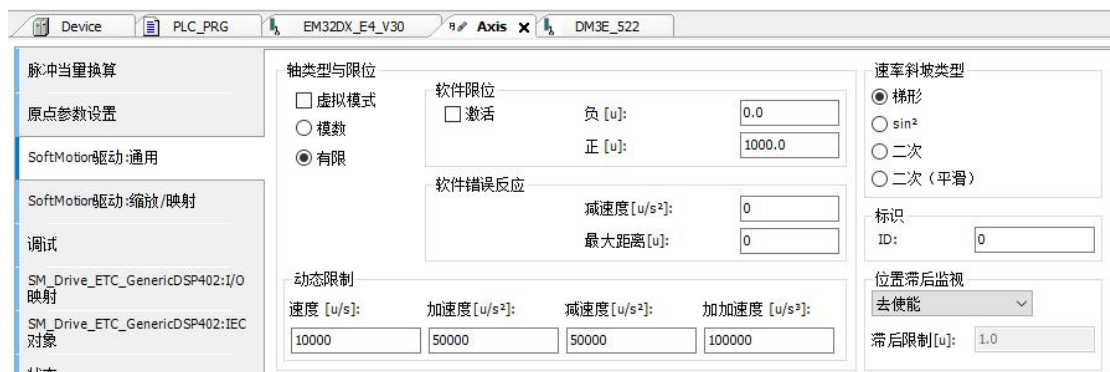


图 2.81 设置最大速度和加速度

这些参数设置完后，电机旋转角度与脉冲数的关系就确定了，即 $1^\circ \leftrightarrow 100$ 脉冲。

在“代码编辑区”以同样方法输入 MC_MoveRelative 模块，各参数如下：

轴号 Axis 为 Axis；

启动信号为 Mstart0；

运动距离 Distance 为 360 个脉冲，即转动 1 圈，360 度；

最大速度 Velocity 为 180 脉冲/秒;

加速度、减速度 Acceleration、Deceleration 均为 20000 脉冲/秒²;

2) LT2070E 触摸屏窗口按键设置

如图 2.83、2.84、2.85、2.86 所示, 需要设置 RS485 与 RS232 串口通讯主从站的串口配置及从站配置站号一致, 同时使用 232 串口及 485 串口通讯时需要分配不同的 Modbus 从站配置站号但串口配置可相同, 触摸屏软件配置详情见其使用手册。

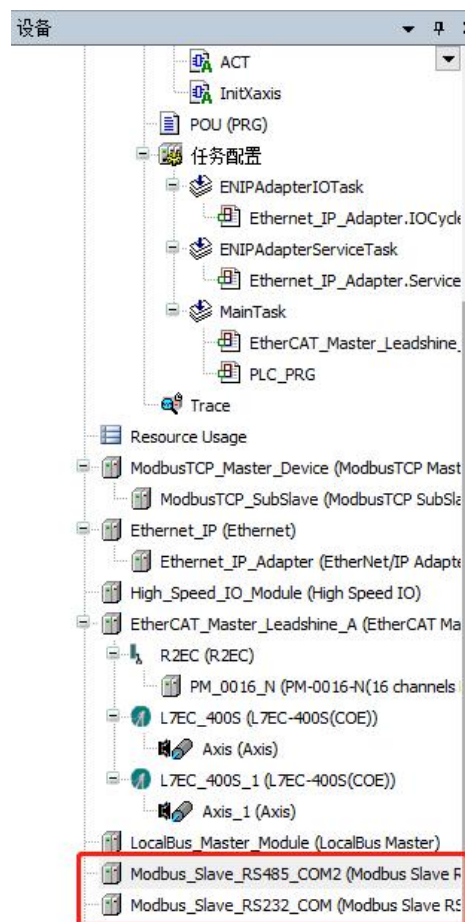


图 2.83 串口设备位置



图 2.84 串口配置



图 2.85 232 串口通讯设置



图 2.86 485 串口通讯设置

关于 LT2070E 触摸屏编程方法详见其使用手册，触摸屏编程之后的窗口如图 2.87 所示。

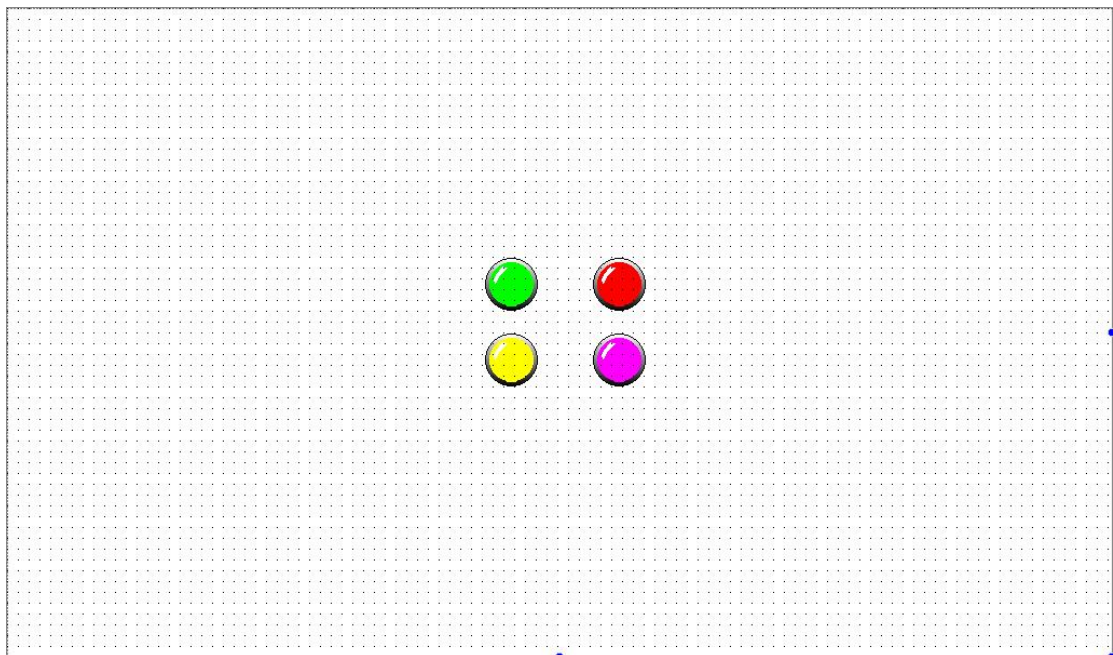


图 2.87 触摸屏界面

绿红指示灯控制 OUT0 开与闭，黄粉指示灯控制 OUT1 开与闭。而 OUT0 控制电机“Axis”运动，OUT1 控制电机“Axis_1”运动。

主程序 PLC_PRG 代码如下：

```
PROGRAM PLC_PRG
```

```
VAR
```

```
InitMotor0: MC_Power;// 电机初始化模块名称
```

```
InitMotor1: MC_Power;
```

```
MotorMove0: MC_MoveRelative;// 电机相对运动模块名称
```

```
MotorMove1: MC_MoveRelative;
```

```
Mstart0:BOOL;// 控制电机启动的变量
```

```
Mstart1:BOOL;
```

```
END_VAR
```

```
InitXaxis();
```

```
IF OUT0=TRUE THEN           // 触摸屏按下绿色指示灯，电机 Axis 启动，转一  
圈。
```

```
    Mstart0:=TRUE;
```

```
ELSE
```

```
    Mstart0:=FALSE;
```

```
END_IF
```

```
IF OUT1=TRUE THEN           // 触摸屏按下黄色指示灯，电机 Axis_1 启动，转  
一圈。
```

```
    Mstart1:=TRUE;
```

```
ELSE
```

```
    Mstart1:=FALSE;
```

END_IF

```
MotorMove0(Axis:= Axis, Execute:= Mstart0, Distance:= 360, Velocity:= 180,  
    Acceleration:=20000, Deceleration:=20000, Jerk:= , BufferMode:= ,  
    Done=> , Busy=> , Active=> , CommandAborted=> , Error=> , ErrorID=> );  
// 电机相对运动模块
```

```
MotorMove1(Axis:= Axis_1, Execute:= Mstart1, Distance:= 360, Velocity:= 180,  
    Acceleration:=20000, Deceleration:=20000, Jerk:= , BufferMode:= ,  
    Done=> , Busy=> , Active=> , CommandAborted=> , Error=> , ErrorID=> );  
// 电机相对运动模块
```

3) 编译、运行程序

编译程序后登录控制器、下载程序，运行程序，当项目树中的各轴前的图标都变为绿色时，如图 2.66 所示，可以开始控制电机运动。

这时，按下触摸屏上的绿色指示灯，打开输出口 Out0，“Axis”轴转一圈，如果要将“Axis”轴再运动，需要先断开输出口 OUT0，再打开输出口 OUT0。Axis_1 轴同理。如图 2.88 所示。

Device.Application.PLC_PRG					
表达式	类型	值	准备值	地址	注
InitMotor0	MC_Power				注
InitMotor1	MC_Power				电
MotorMove0	MC_MoveRelative				电
MotorMove1	MC_MoveRelative				控
Mstart0	BOOL	TRUE			
Mstart1	BOOL	TRUE			


```

1 InitXaxis();
2
3 IF OUT0 TRUE =TRUE THEN
4     Mstart0 TRUE :=TRUE;    // 触摸屏开关0按下, 电机Axis启动
5 ELSE
6     Mstart0 TRUE :=FALSE;
7 END_IF
8
9 IF OUT1 TRUE =TRUE THEN
10    Mstart1 TRUE :=TRUE;    // 触摸屏开关1按下, 电机Axis_1启动
11 ELSE
12    Mstart1 TRUE :=FALSE;
13 END_IF
14
15 MotorMove0(Axis:= Axis, Execute TRUE := Mstart0 TRUE, Distance 360 := 360, Velocity 180 := 180,
16    Acceleration 2E+04 :=20000, Deceleration 2E+04 :=20000, Jerk:= , BufferMode:= ,
17    Done=> , Busy=> , Active=> , CommandAborted=> , Error=> , ErrorID=> );
18 // 电机相对运动模块
19
20 MotorMove1(Axis:= Axis_1, Execute TRUE := Mstart1 TRUE, Distance 360 := 360, Velocity 180 := 180,
21    Acceleration 2E+04 :=20000, Deceleration 2E+04 :=20000, Jerk:= , BufferMode:= ,
22    Done=> , Busy=> , Active=> , CommandAborted=> , Error=> , ErrorID=> );
23 // 电机相对运动模块
24

```

图 2.88 程序运行

2.8.3. 用 MC500 和 EM32DX-C1 控制 IO 模块上的输出灯

主从站配置

1) CAN网络及波特率设置

双击“CANbus”，进入CAN 网络配置界面。

CAN 网络配置：MC500CS 有一个 CAN 口，CANbus 页面中的“网络”设置为 0。

波特率设置：CAN 总线有多档波特率，本例程中使用 250 的波特率。参数配置完成后，写入程序再重新上电控制器，波特率就会成功配置到 PLC。显示界面如图 2.89 所示：

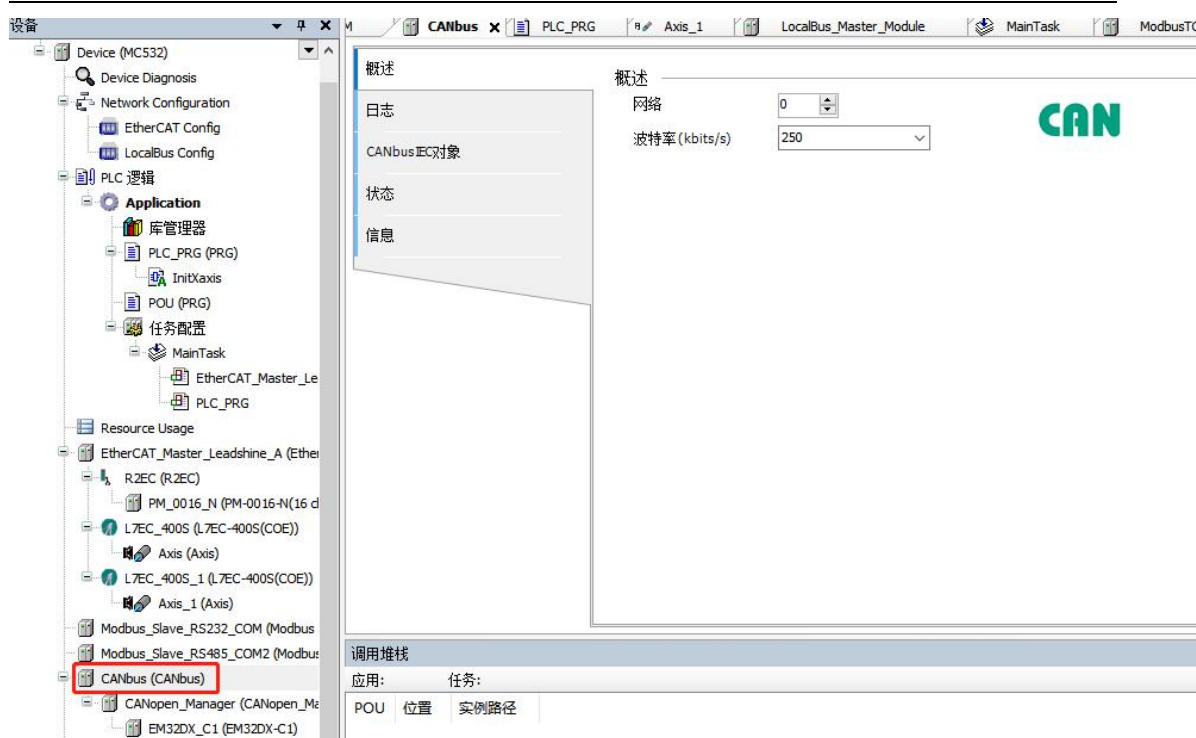


图 2.89 CAN 网络配置

2) CAN主站配置

双击“CANopen_Manager”，进入CAN 主站配置界面。

节点 ID: MC500CS 主机的 ID，默认为 127,可以采用默认配置。其他配置采用默认即可，配置完成后如图 2.90 所示：

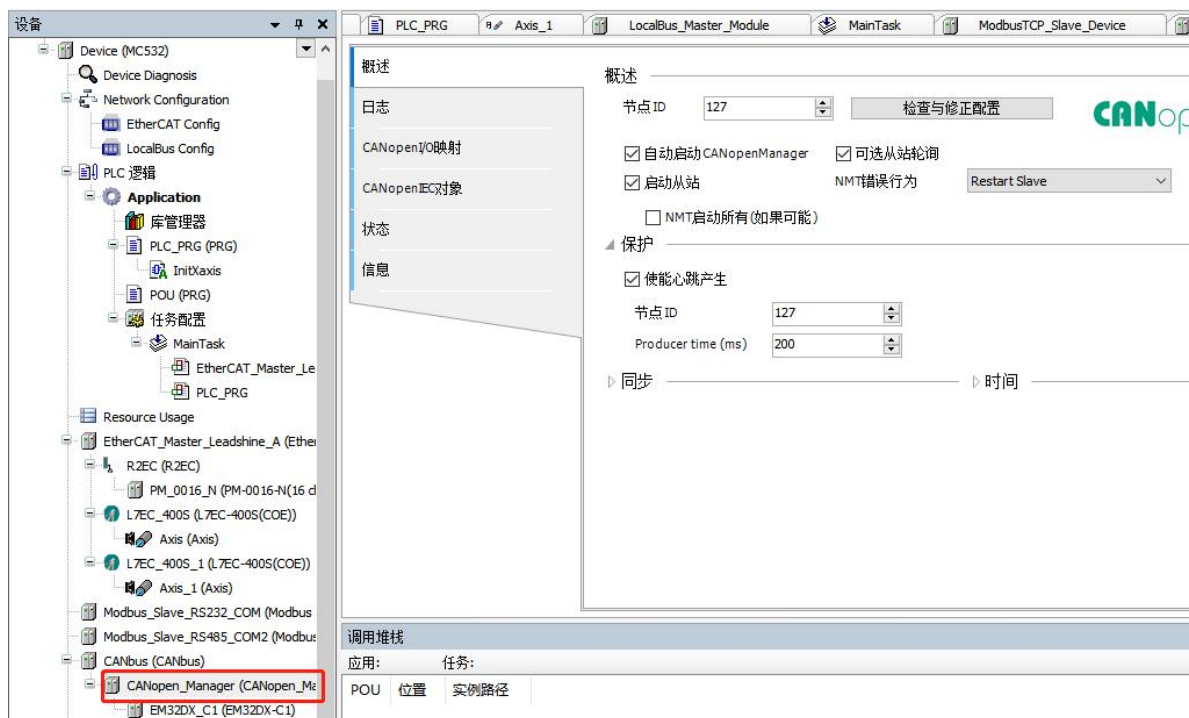


图 2.90 CAN 主站配置

3) 模块配置

双击“EM32DX-C1”，进入模块配置“概述”界面。

节点 ID：扩展模块的节点 ID，需要在从站的硬件上进行配置，本例中设置模块 ID 为1。

勾选“使能专家设置”后，可以看到多项隐藏的参数，一般情况下用户不需要设置这些参数，采用默认配置。配置完成后的界面如图 2.91 所示：

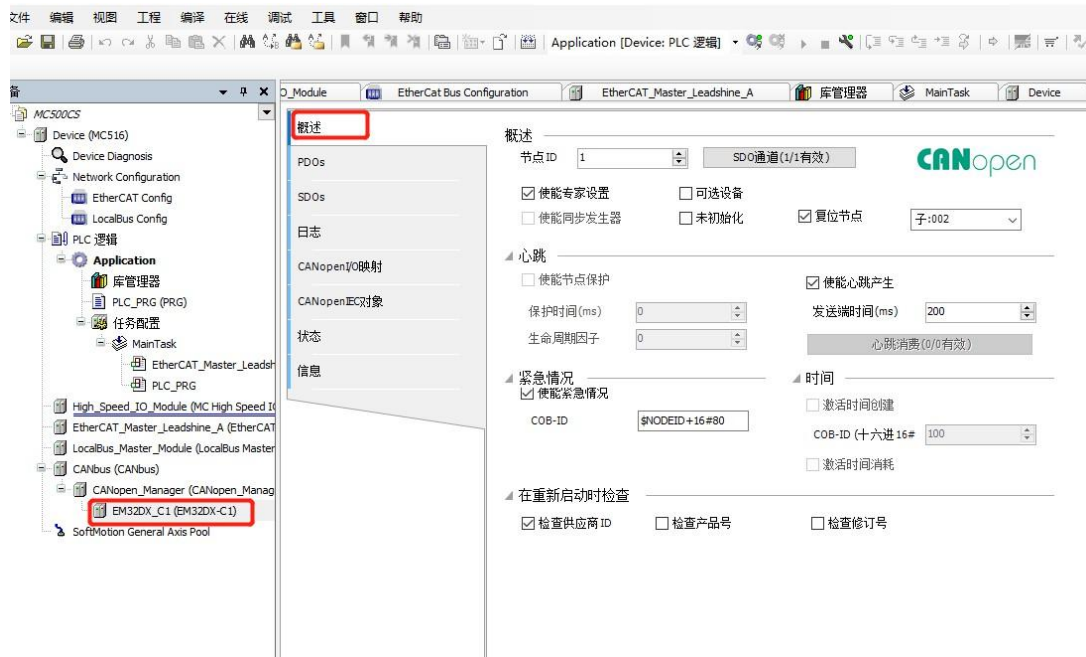


图 2.91 模块配置界面

点击“PDO 映射”，显示界面如图 2.92 所示。

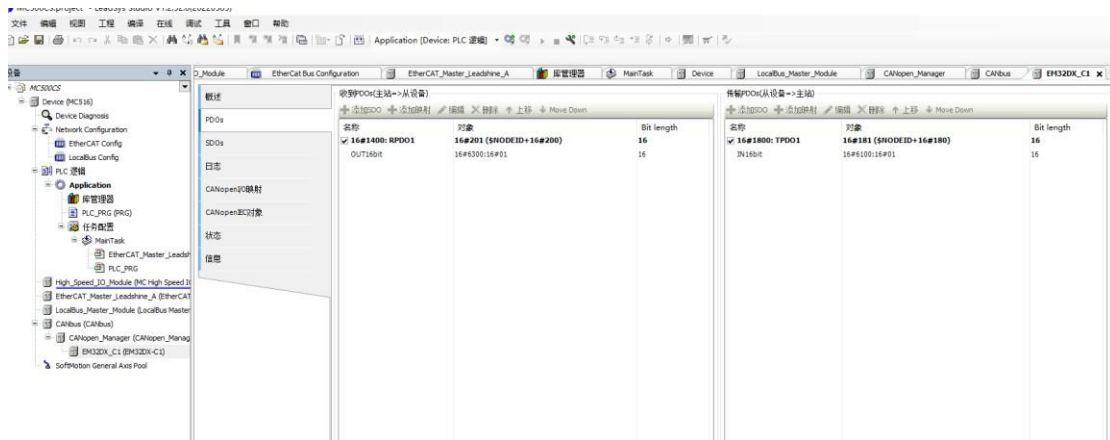


图 2.92 “PDO 映射”界面

双击“RPDO1”和“TPDO1”，在传输类型中选择“异步的-特定于设备配置文件 (类型 255)”，如图 2.93 所示：

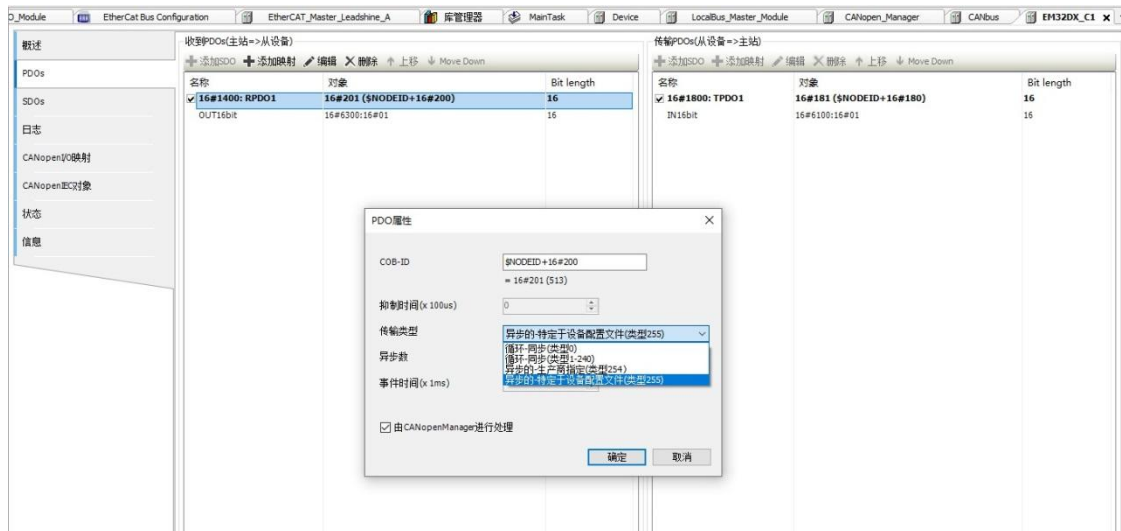


图 2.93 设置传输类型

配置完上述参数之后，还需要对刷新变量的方式进行设置，在 CANopen IO 映射中，需要选择“使能 1”，如图2.94所示：

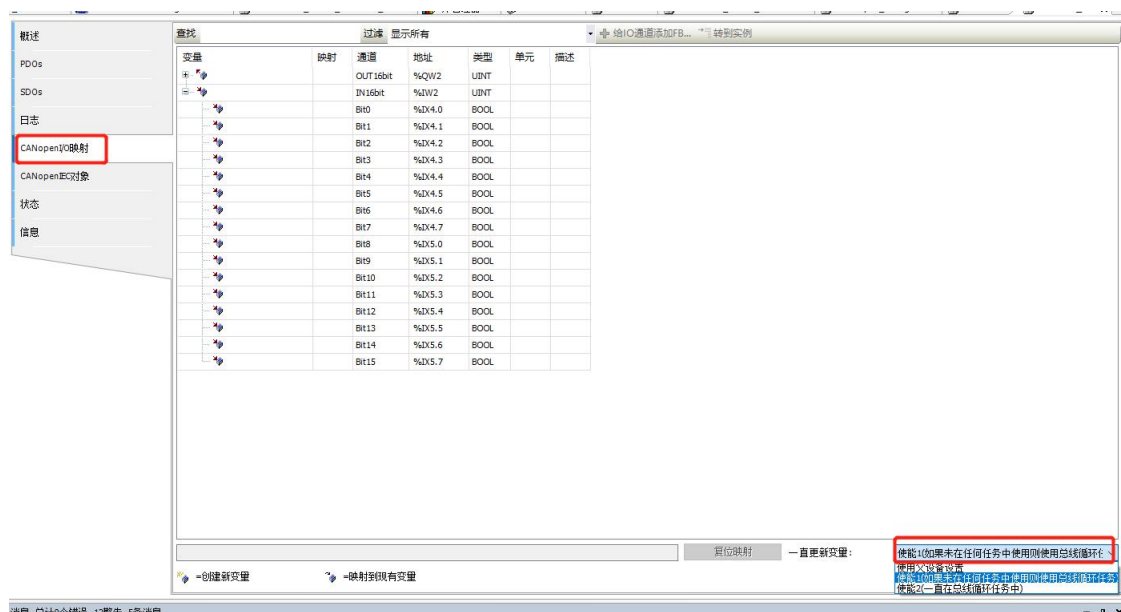


图 2.94 设置为“使能 1”

4) 模块拨码配置

从上面的配置过程可以得到：CAN 网络波特率为 250；模块节点号（模块 ID 号）

为 1；该模块为网络内的最后一个模块（整个网络内只有一个模块，也为最后一个模块），需要拨上终端电阻。EM32DX-C1 模块拨码表详见其使用手册。

在应用程序中定义变量” IO_OUT ”，该变量可以和 CANopen IO 变量直接进行映射，通过对该变量进行赋值，即可实现对 CANopen 模块的 IO 进行控制。如图 2.95 所示。

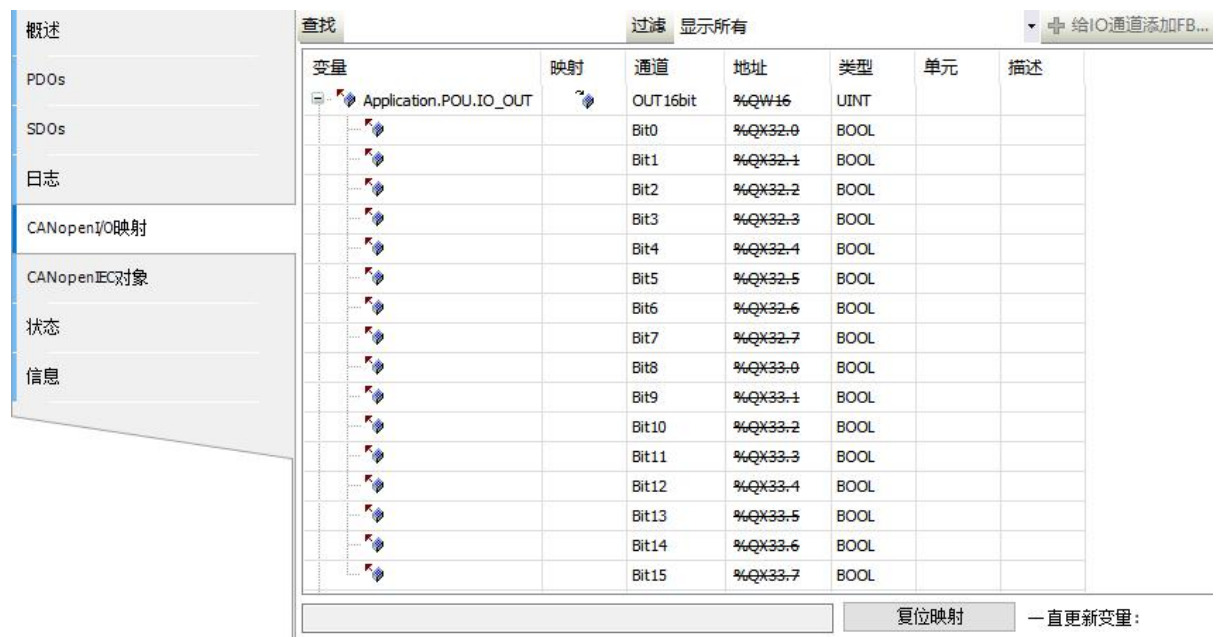


图 2.95 定义模块输出口

如下代码实现的是对 CAN总线模块上 IO 模块的输出灯进行间隔 50ms 的流水灯操作。

```

PROGRAM PLC_PRG
VAR
    TON_0:Standard.TON;//延时计时器
    IO_OUT:DWORD:=1;
    n:DWORD:=1;
    StartIO:BOOL;
END_VAR
TON_0(IN:=NOT TON_0.Q,PT:=T#50MS,Q=>,ET=>);
IF TON_0.Q AND StartIO THEN

```

```
IO_OUT:=ROL(IO_OUT,n);  
  
IF IO_OUT>65535 THEN  
  
    IO_OUT:=1;  
  
END_IF  
  
END_IF
```

实际运行效果如图 2.96 所示。

变量	映射	通道	地址	类型	当前值
Application.POU.IO_OUT		OUT 16bit	%QW16	UINT	1
		Bit0	%QX32.0	BOOL	TRUE
		Bit1	%QX32.1	BOOL	FALSE
		Bit2	%QX32.2	BOOL	FALSE
		Bit3	%QX32.3	BOOL	FALSE
		Bit4	%QX32.4	BOOL	FALSE
		Bit5	%QX32.5	BOOL	FALSE
		Bit6	%QX32.6	BOOL	FALSE
		Bit7	%QX32.7	BOOL	FALSE
		Bit8	%QX33.0	BOOL	FALSE
		Bit9	%QX33.1	BOOL	FALSE
		Bit10	%QX33.2	BOOL	FALSE
		Bit11	%QX33.3	BOOL	FALSE
		Bit12	%QX33.4	BOOL	FALSE
		Bit13	%QX33.5	BOOL	FALSE
		Bit14	%QX33.6	BOOL	FALSE
		Bit15	%QX33.7	BOOL	FALSE
		IN 16bit	%IW28	UINT	0

图2.96 实际运行效果

第三章 编程基础 1：变量与函数

传统的 PLC 编程方式要求编程者先分配寄存器地址，然后在此基础上命名标签名。这样，编程者在编程之前必须先确定详细的 I/O 点数和中间寄存器数，然后分配相应的寄存器地址，最后再给这些寄存器地址命名标签名。

LeadSys Studio 使用基于 IEC61131-3 国际标准的编程语言。其编程方式不同于传统的 PLC 编程方式，编程者可以无需考虑先分配具体的寄存器地址，可以直接先声明变量名，在程序的执行代码中使用变量名即可，编程者可以随时将某个特定变量名链接到某个寄存器地址，也可以随时更改链接到该变量名的地址，使整个编程过程更加灵活和方便，大大节省了编程者的开发时间。

3.1.直接地址

3.1.1. 直接地址定义

直接地址指映射到控制器设备具体寄存器的地址，包括输出单元 Q 区，输入单元 I 区，存储区单元 M 区,LeadSys 变量的存储位置可以由用户指定为直接地址，也可以不指定地址，由系统自动分配。直接地址都可以按位，按字节，按字和按双字进行访问，前缀符号如表 3.1 所示。

表 3.1 变量前缀符号

前缀符号	定义	数据类型
X	位（BIT）	BOOL
B	字节（BYTE）	BYTE
W	字（WORD）	WORD
D	双字（DWORD）	DWORD

直接地址的定义格式如下：

<地址区><前缀符号><数字>.<数字>

LC/PMC 控制器直接地址寻址规则如表 3.2 所示。I 区、Q 区、M 区地址寻址规则相同。Word 型的起始地址，遵循的是起始地址为偶数 Byte 地址；DWord 型寄存器的起始地址，遵循的是起始地址为偶数 Word 地址对齐，其索引号为 2 倍关系的原则。这样方便地址的计算。

其中 I、Q 区地址不能掉电保存，M 区有掉电保存。

表 3.2 直接地址寻址规则

M 区直接地址寻址规则			
按 BIT 寻址	按 Byte 寻址	按 Word 寻址	按 DWORD 寻址
MX0.7~MX0.0	MB0	MW0	MD0
MX1.7~MX1.0	MB1		
MX2.7~MX2.0	MB2	MW1	
MX3.7~MX3.0	MB3		
MX4.7~MX4.0	MB4	MW2	MD1
MX5.7~MX5.0	MB5		
MX6.7~MX6.0	MB6	MW3	
MX7.7~MX7.0	MB7		
...	...	...	...
Q 区直接地址寻址规则			
按 BIT 寻址	按 Byte 寻址	按 Word 寻址	按 DWORD 寻址
QX0.7~QX0.0	QB0	QW0	QD0
QX1.7~QX1.0	QB1		
QX2.7~QX2.0	QB2	QW1	
QX3.7~QX3.0	QB3		
QX4.7~QX4.0	QB4	QW2	QD1

QX5.7~QX5.0	QB5	QW3	
QX6.7~QX6.0	QB6		
QX7.7~QX7.0	QB7		
...	...	...	...
I 区直接地址寻址规则			
按 BIT 寻址	按 Byte 寻址	按 Word 寻址	按 DWORD 寻址
IX0.7~IX0.0	IB0	IW0	ID0
IX1.7~IX1.0	IB1		
IX2.7~IX2.0	IB2	IW1	
IX3.7~IX3.0	IB3		
IX4.7~IX4.0	IB4	IW2	ID1
IX5.7~IX5.0	IB5		
IX6.7~IX6.0	IB6	IW3	
IX7.7~IX7.0	IB7		
...	...	...	...

3.1.2. 直接地址范围

不同系列的控制器提供的直接地址范围不同。

PMC 系列控制器编程系统提供 128KB (Byte) 的输入区域 (I 区), 128KB (Byte) 输出区域 (Q 区) 和 480KB 存储区域 (M 区)。编程时, 用户可以直接访问直接地址, 也可以定义变量后把变量映射到直接地址间接访问。存储区域使用的地址范围如表 3.3 所示。

表 3.3 PMC 系列控制器直接地址范围

区域	大小	地址范围
I 区 (128KByte)	64KWords	IW0~IW65535
Q 区 (128KByte)	64KWords	QW0~QW65535
M 区 (480KByte)	240KWords	MW0~MW245759

LC 系列控制器提供 128KByte 的输入单元 I 区, 128KByte 的输出单元 Q 区和 5MByte 存储区单元 M 区。存储区域使用的地址范围如表 3.4 所示。

表 3.4 LC 系列控制器直接地址范围

区域	大小	地址范围
I 区 (128KByte)	64KWords	IW0~IW65535
Q 区 (128KByte)	64KWords	QW0~QW65535
M 区 (5MByte)	2.5MWords	MW0~MW2621439

3.1.3. Modbus 地址范围

LC/PMC 系列控制器作为 Modbus 从站, 支持与 HMI 或者上位机程序 Modbus TCP 或 Modbus。

RTU 主站通信, Modbus 主站侧访问的直接地址范围如表 3.5、表 3.6 所示。

(PMC600 请参考 PMC_Communication 库中的 GVL_HMI 变量)

表 3.5 LC/PMC 系列控制器可访问的 I、Q 区范围

地址范围	功能码	起始地址	线圈数量	说明
QW0~QW4095 (QX0.0 ~ QX8191.7)	0X01,0x05,0x0f	0	65536	通用标准 Modbus 协议都可以访问
IW0~IW4095	0X02	0	65536	通用标准 Modbus

(IX0.0 ~ IX8191.7)				协议都可以访问
--------------------	--	--	--	---------

表 3.6 LC/PMC 系列控制器可访问的 M 区范围

地址范围	功能码	起始地址	寄存器数量	说明
MW0~MW65535	0x03,0x06,0x10	0	65536	通用标准 Modbus 协议都可以访问

3.2. 变量

3.2.1. 变量定义

变量可以在程序组织单元 POU、全局变量 GVL 的声明编辑器中定义或者通过自动声明对话框定义。变量类型包括本地变量 VAR，临时变量 VAR_TEMP，输入变量 VAR_INPUT，输出变量 VAR_OUTPUT，输入输出变量 VAR_IN_OUT，全局变量 VAR_GLOBAL，掉电变量 VAR_GLOBAL PERSISTENT RETAIN。

变量声明的格式基于 IEC61131-3 标准。应注意以下几点：

- 1) 变量首字符不能是数字，可以是字母或下划线。
- 2) 变量名不区分大小写。
- 3) 变量名不允许出现空格和特殊字符。
- 4) 不能使用关键字、函数和功能块名称作为变量名。
- 5) 单行注释可以用符号“//”表示；

多行注释，可以选择符号“(* *)”。

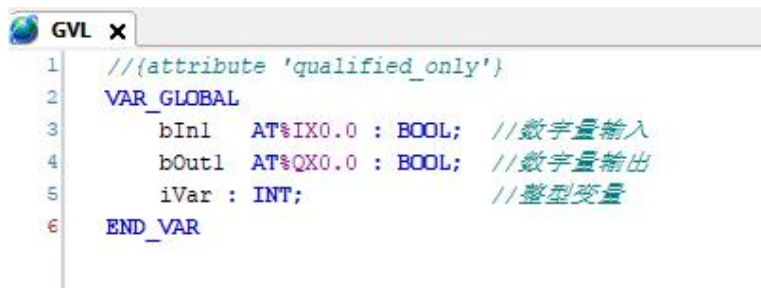
变量声明的格式如下：

〈变量名称〉{AT〈直接地址〉}：〈数据类型〉{:=〈初始值〉}； {///
注释内容〉}

位于大括号 {} 内的内容为可选部分。

直接地址的变量通过输入寄存器 I、输出寄存器 Q 和内存区 M 来区分变量存储区域，通过位 X、字节 B、字 W、双字 D 来定义变量类型。

如图 3.1 所示，在声明编辑器中定义一个数字量输入变量 bIn1，一个数字量输出变量 bOut1，一个整型变量 iVar。



```
1  //{attribute 'qualified_only'}
2  VAR_GLOBAL
3      bIn1   AT%IX0.0 : BOOL; //数字量输入
4      bOut1  AT%QX0.0 : BOOL; //数字量输出
5      iVar   : INT;          //整型变量
6  END_VAR
```

图 3.1 变量声明示例

如果用户不熟悉变量声明格式，可以用鼠标点击变量定义区右边的图标“”，将文本方式切换为表格方式声明变量，如图 3.2 所示。



图 3.2 变量声明表格视图

3.2.2. 变量类型

LeadSys 变量类型符合 IEC61131-3 标准，根据应用范围定义了变量属性，以及变量的附加属性。变量属性见表 3.7，变量的附加属性见表 3.8。

VAR、VAR_INPUT、VAR_OUTPUT 和 VAR_IN_OUT 是在程序组织单元 POU 中应用较多的几种变量类型。全局变量 VAR_GLOBAL 在实际工程项目中也会大量使用。

表 3.7 变量的属性

变量类型关键字	变量属性	外部读写	内部读写
VAR	局部变量		R/W
VAR_INPUT	输入变量，由外部提供	R/W	R
VAR_OUTPUT	输出变量，由内部变量提供给外部	W	R/W
VAR_IN_OUT	输入-输出变量	R/W	R/W
VAR_GLOBAL	全局变量，能在所有配置、资源内使用	R/W	R/W
VAR_TEMP	临时变量，程序和功能块内部存储使用		R

	的变量		
VAR_STAT	静态变量		
VAR_EXTERNAL	外部变量，能在程序内部修改，但需由全局变量提供	R/W	R/W

表 3.8 变量的附加属性

变量附加属性关键字	变量附加属性
RETAIN	保持性变量，用于掉电保持
PERSISTENT	保持型变量，用于掉电保持
VAR RETAIN PERSISTENT VAR PERSISTENT RETAIN	两者功能一样，皆为保持型变量，用于掉电保持
CONSTANT	常量

变量附件属性的定义方法：

1) RETAIN（保存）型变量

RETAIN 型变量在控制器正常关闭、打开（或收到在线命令“热复位”），甚至意外关闭之后仍能够保持原来的值。随着程序重新开始运行，存储的值能够继续发挥作用。

RETAIN 类型变量声明格式如下：

VAR RETAIN

<变量名称>:<数据类型>;

END_VAR

但 RETAIN 变量在“初始值复位”、“冷复位”和程序下载之后将会重新初始化。

RETAIN 型变量仅仅存储在一个单独的内存区中。

2) PERSISTENT RETAIN / RETAIN PERSISTENT（持续保存）型变量

PERSISTENT 类型变量声明格式如下:

VAR GLOBAL PERSISTENT RETAIN

<变量名称>:<数据类型>

END_VAR

与 RETAIN 变量一样, RETAIN PERSISTENT 和 PERSISTENT RETAIN 变量也存储在一个独立的内存区中。

3) CONSTANT 常量

在程序运行过程中只能读取数据而不能进行修改的量称为常量, 关键字为 CONSTANT。可以将常量声明为局部常量, 也可以声明为全局常量。

CONSTANT 常量声明格式如下:

VAR CONSTANT

<常量名称>:<数据类型> := <初始化值>;

END_VAR

在实际应用中通常可以将一些重要参数或系数设置成常量, 这样可以有效地避免由于其他变量对其修改最终影响系统整体稳定性及安全性。例如:

VAR CONSTANT

pi:REAL:=3.1415926;

END_VAR

程序一旦开始运行, 通过 CONSTANT 声明的变量在程序运行过程中是不允许被修改的。

3.2.3. 数据类型

无论声明的是变量还是常量, 都必须使用数据类型。数据类型决定了它将占用多大的存储空间及存储何种类型的值。LeadSys 的数据类型分为标准数据类型和扩展数据类型。

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、整型类型、实数类型、字符串类

型和时间数据类型，如表 3.9 所示。

表 3.9 数据类型

数据类型	关键字	占用内存	取值范围
位	BIT	1bit	0 和 1，仅定义结构体和功能库内部可使用
布尔	BOOL	1Byte	FALSE(0)或 TRUE(1)
整型	BYTE	8bit	0~255
	WORD	16bit	0~65535
	DWORD	32bit	0~4294967295
	LWORD	64bit	0~(2^{64} -1)
	SINT	8bit	-128~127
	USINT	8bit	0~255
	INT	16bit	-32768~32767
	UINT	16bit	0~65535
	DINT	32bit	-2147483648~2147483647
	UDINT	32bit	0~4294967295
	LINT	64bit	-263~(2^{63} -1)
实数	REAL	32bit	1.175494351e-38~3.402823466e+38
	LREAL	64bit	2.2250738585072014e-308~1.7976931348623158e+308
字符串	STRING	8×N bit	
	WSTRING	16×N bit	
时间	TIME	32bit	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) 扩展数据类型

扩展数据类型包括结构体、枚举、联合、别名、引用、指针等。详见表 3.10。

表 3.10 扩展数据类型

数据类型	关键字	占用内存	取值范围
结构体	STRUCT	根据定义	根据定义
枚举	ENUM	根据定义	根据定义
联合	UNION	根据定义	根据定义
别名	Reference TO	根据定义	根据定义
指针	POINTER TO	根据定义	根据定义
长时间	LTIME	64	纳秒~天
宽字节字符串	WSTRING	16 X (N+1)	

3.2.4. 数据单元类型

数据单元类型 DUT (Data Unit Type) 也称为用户自定义数据类型，包括结构体、枚举、别名和联合。

DUT 的创建方法如图 3.3 所示：在设备导航栏窗口中的“Application”中，鼠标右键单击“添加对象”，选择“DUT”，在弹出对话框中，选择需要添加的数据单元类型，给定名称后，点击“打开”进行创建。

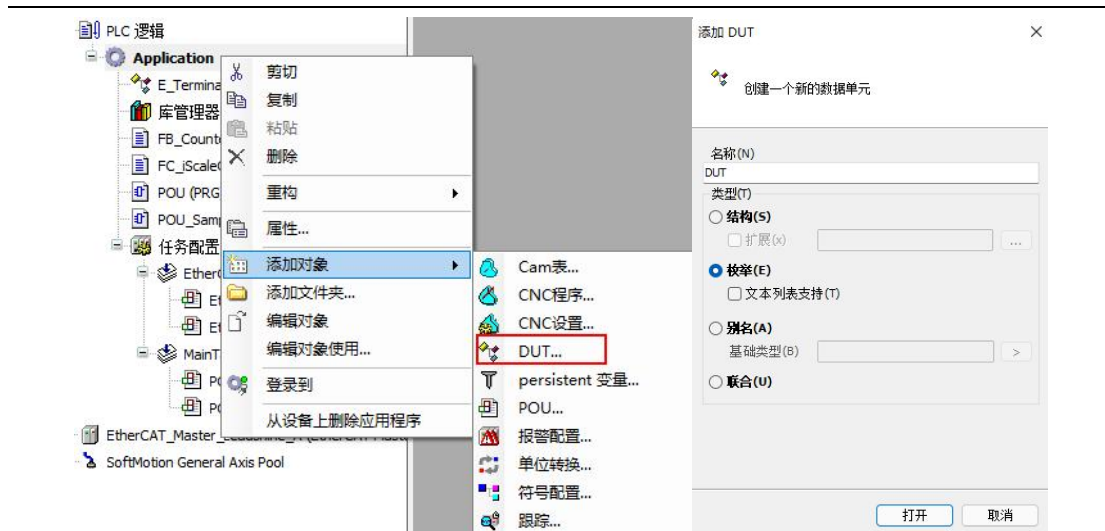


图 3.3 创建 DUT

可供选择的类型有结构体、枚举、别名和联合。其中结构体支持使用扩展（继承）的方式，用户可以使用面向对象的编程方式。通过一个已经在另一个工程中定义的结构体再扩展另一个结构体，这意味着扩展结构体将会自动针对当前结构体进行定义，这是对原有结构体的元素补充。

1) 结构体

结构体是由一系列相同或者不同类型的数据构成的数据集合，是一种用户自定义数据类型。用户有时候需要将不同数据类型的数据组合成一个整体对外引用。

例如：控制一台伺服电机，通常包括轴使能、运行速度、加速度、减速度、目标位置等信息，这些信息都和这台伺服电机关联，以独立变量进行声明很难反应变量和电机的内在关系，此时就可以使用结构体来管理。

可定义一个结构体 AxisStruct，如图 3.4 所示。

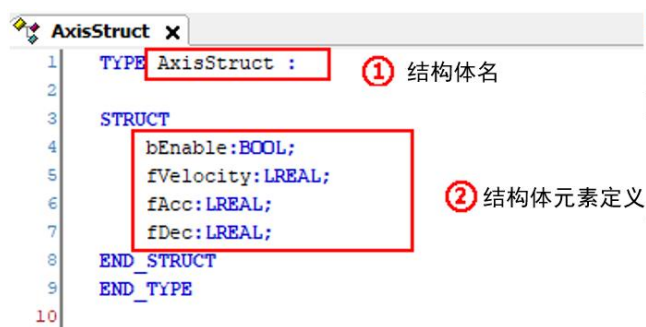


图 3.4 结构体的定义

结构体在程序中的调用需要首先在声明区实例化，之后可以直接在程序中使用已声明的结构体变量。结构体变量中的元素可以用索引符号“.”的方式调用，如图 3.5 所示，通过 axisX 实例可访问其关联的元素。

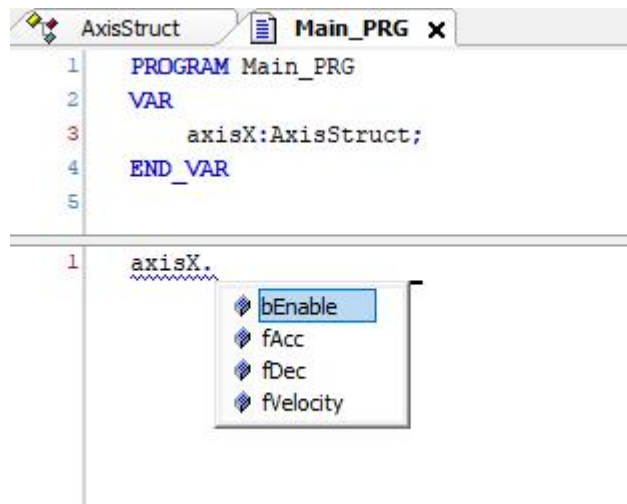


图 3.5 结构体的实例化和调用

2) 枚举

枚举是若干个被命名的整型常数的集合。枚举类型在程序中经常使用。如为了方便识别程序执行的步序，步序就可以定义为枚举。枚举默认为 INT 类型，也可以由用户明确指定。枚举类型在没有赋初值的情况下，从 0 开始按照 INT 整型数据递加。整数可以直接赋值给一个枚举类型。

枚举类型的定义如图 3.6 所示。

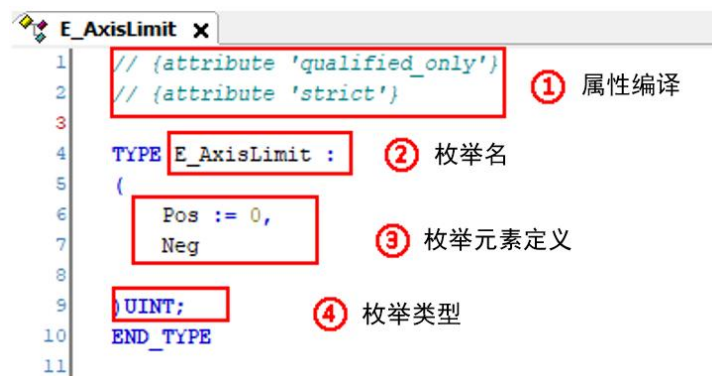


图 3.6 枚举的定义

注意：默认创建的枚举中会自动插入属性编译，如图 3.6 中的①，注释掉的

{attribute 'qualified_only'}表示定义的枚举需要通过指定的全局变量名寻址，{attribute 'strict'}表示定义的枚举不可以参与数学运算，也不可以将其其他数据类型包括常量的值直接赋值给枚举元素。如果用户不需要属性编译可以直接删除。

在程序中调用枚举的方法如图 3.7 所示。

```
IF iTest = Neg THEN
    str := 'iTest = Neg';
END_IF
```

图 3.7 枚举的调用

3) 别名

别名就是给一个基础数据类型、数组或自定义数据类型另一个名称，方便用户管理变量。例如定义一个长度固定的发送字符串，为了方便后期统一修改字符串长度，此时可以使用别名。

别名的定义方法如图 3.8 所示。



图 3.8 别名的定义方法

别名的使用方法如图 3.9 所示。如果在程序中使用别名 Sdata 并赋值，当 Sdata 在程序、功能块等多处使用的情况下，用户如果需要修改 Sdata 的长度时，只需要修改别名的定义即可全部完成修改。

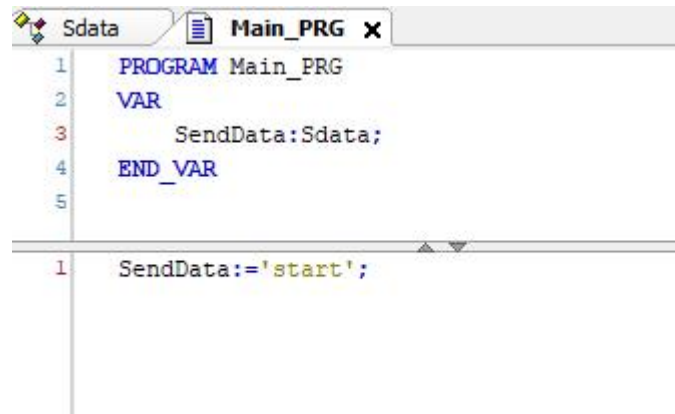


图 3.9 别名的使用方法

4) 联合

联合体也叫共用体，它是将所有类型的变量都存放到起始地址相同的内存单元里。联合体的定义方法如图 3.10 所示。

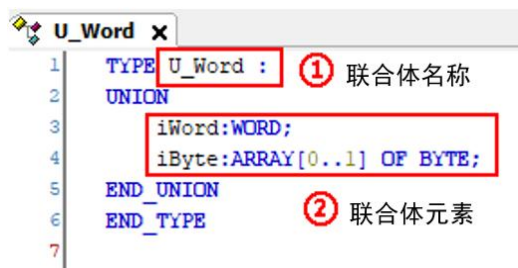


图 3.10 联合体的定义方法

例如在程序中调用联合体 U_Word，可以将两个字节变量合成一个字变量。程序代码如图 3.11 所示。

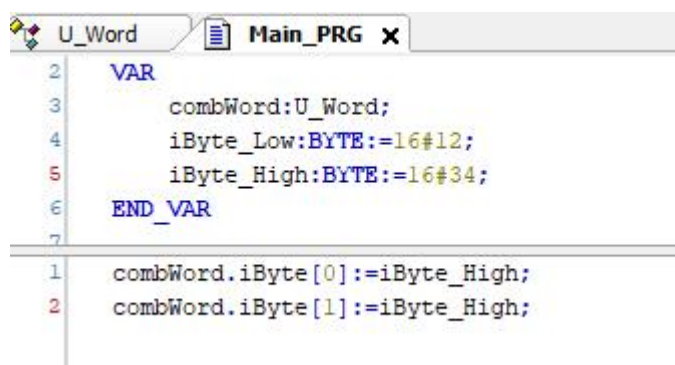
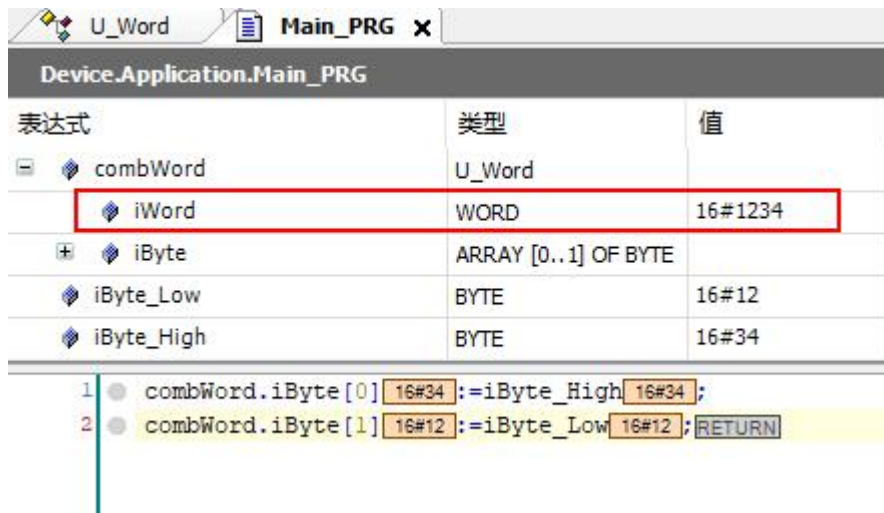


图 3.11 联合体的用法

如图 3.12 所示，登录查看运行结果，可以看到给联合体中的数组 iByte 中的元素赋值，iWord 由于共享内存的关系，自动有了数值 16#1234。



表达式	类型	值
combWord	U_Word	
iWord	WORD	16#1234
iByte	ARRAY [0..1] OF BYTE	
iByte_Low	BYTE	16#12
iByte_High	BYTE	16#34

```

1 combWord.iByte[0] 16#34 := iByte_High 16#34;
2 combWord.iByte[1] 16#12 := iByte_Low 16#12; RETURN
  
```

图 3.12 使用联合体的效果

3.2.5. 掉电保持变量

工业现场的控制器在实际运行过程中，往往会遇到设备异常断电或关机，控制器内部某些变量保存的数据希望在下次上电开机后，这些数据还保持不变，可以正常调用。

LeadSys 提供了两种保持型变量类型，分别是 RETAIN 和 PERSISTENT RETAIN，后者在实际的应用中使用更频繁。

当控制器使用不同复位指令时，不同关键字的掉电保持变量的状态是不同的，结果详见表 3.11。

表 3.11 掉电、复位对保持型变量的影响

在线命令	VAR	VAR_RETAIN	VAR PERSISTENT RETAIN
掉电	×	√	√
热复位	×	√	√
冷复位	×	×	√

初始值复位	×	×	×
程序下载	×	×	√
在线修改	√	√	√

注：表中符号×表示恢复初始值，符号√表示数据仍保留。

用户可以使用掉电保持变量列表添加掉电保持变量。在设备导航树中鼠标右击“Application”，选择“添加对象”->“persistent 变量...”，如图 3.13 所示。在弹出的对话框中输入掉电保持变量列表名称，然后单击“打开”按钮即可，如图 3.14 所示。

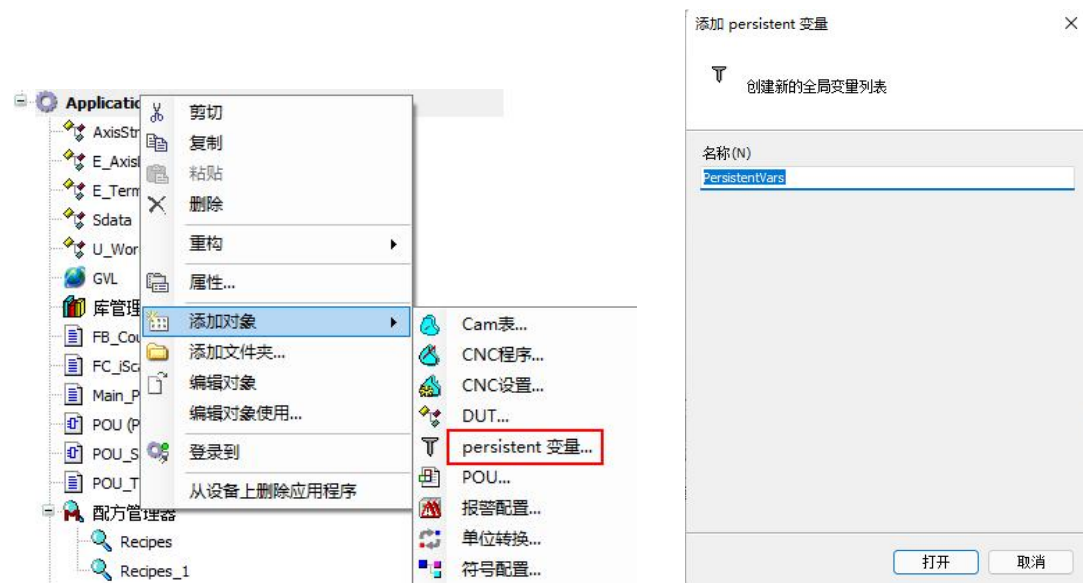


图 3.13 添加 persistent 变量表

图 3.14 设置 persistent 变量表名称

用户可以直接在掉电保持变量列表中声明掉电保持变量，如图 3.15 中的①所示。

该方式有一个缺点，不支持直接地址（如 MX0.0）定义的变量。

建议用户在全局变量列表中声明 PERSISTENT RETAIN 或 RETAIN 关键字，然后通过“添加所有实例路径”，在掉电保持列表中生成实例路径。如图 3.15 中的②所示。

注意：全局变量列表中声明的 PERSISTENT RETAIN 或 RETAIN 变量，至少需要一个变量在程序中使用，才能添加到掉电保持变量列表里。

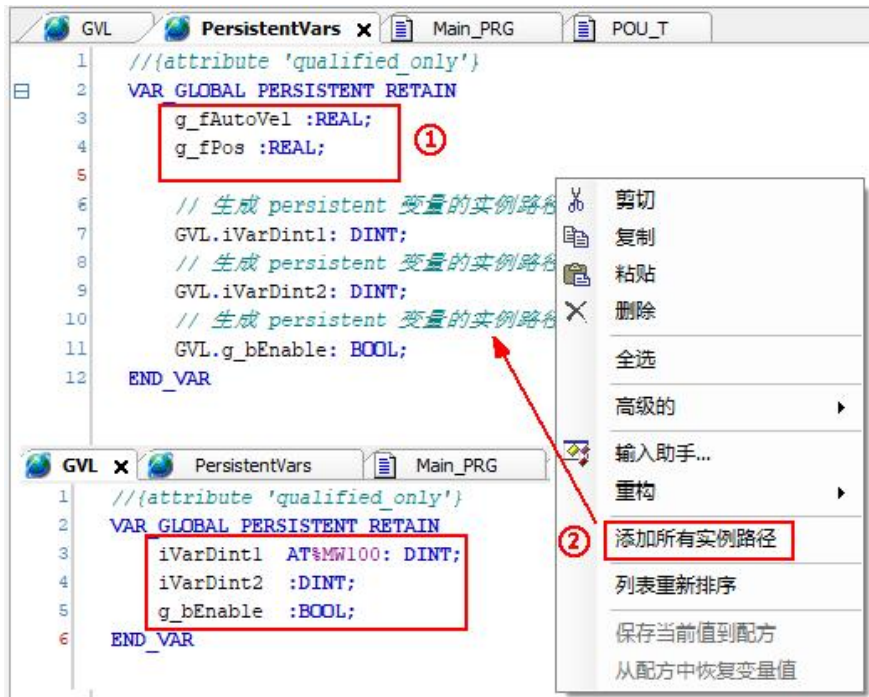


图 3.15 声明掉电保持变量的 2 种方法

3.2.6. 全局变量和局部变量

从范围上来划分，变量分为全局变量和局部变量。变量的定义范围确定了其在哪个程序组织单元 POU 中可以被调用。每个变量的范围由它被声明的位置和声明所使用的变量关键字所决定。

1) 全局变量

在程序组织单元 POU 之外定义的变量为全局变量。全局变量可以被工程中其它程序组织单元所共用，因此，全局变量可以非常便捷地实现两个不同程序或功能块之间的数据交换。

需要注意：一个系统中不能有两个同名的全局变量，所有的全局变量都需要在全局变量列表中声明。

用户可以通过全局变量列表添加全局变量。在设备导航树中鼠标右击“Application”，选择“添加对象”->“全局变量列表...”，如图 3.16 所示。在弹出的对话框中输入全局变量列表名称，然后单击“打开”按钮即可，如图 3.17 所示。

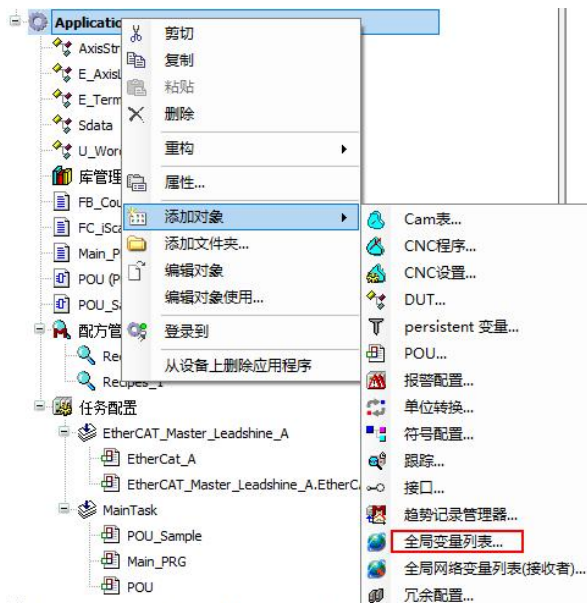


图 3.16 添加全局变量表图

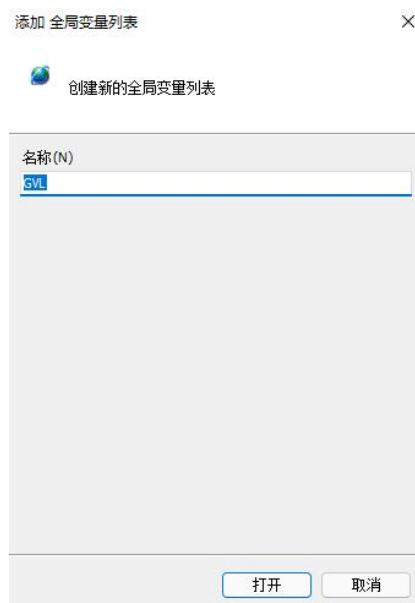


图 3.17 设置全局变量表名称

例如：在全局变量列表中定义一个名为 `g_bEnable` 的全局变量。如图 3.18 所示。

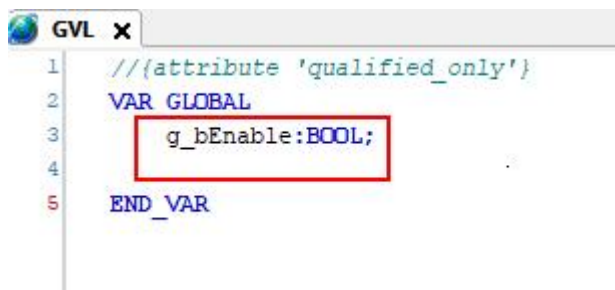


图 3.18 定义全局变量

2) 局部变量

在一个程序组织单元 POU 内定义的变量为内部变量。它只在该 POU 内有效，这些变量也称为局部变量。用户可以在多个独立的程序中使用同名的局部变量，因为它们之间没有映射关系，互不影响。

局部变量的定义如图 3.19 所示，`pos` 为 POU_T 内的一个局部变量。

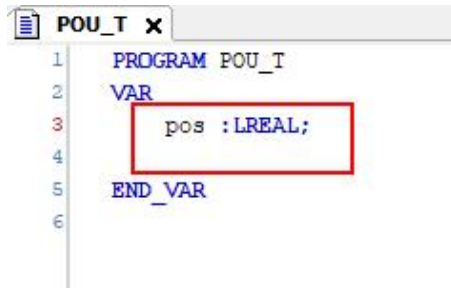


图 3.19 定义局部变量

3.3. 配方

配方是一组参数值，用来提供生产产品和控制生产过程中所需的信息。例如：饼干的原料，包括：白糖、鸡蛋、黄油、面粉等，烘焙时间等参数。配方也可以用于设置和监控 PLC 的控制参数。

为了达成这个目的，用户可以从 PLC 读取配方，也可对其写入数据。还可以从文件中载入配方或者将配方存成文件。这些相互作用通过已经设置好的视图元素即可实现。

在设备导航树中通过鼠标右击“Application”，选择“添加对象”->“配方管理器”来添加配方管理器。如图 3.20 所示。

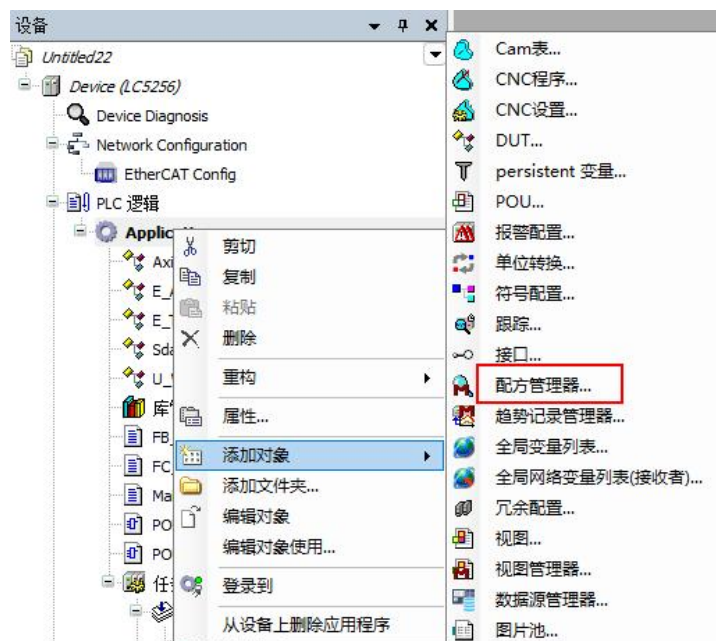


图 3.20 添加配方管理器

添加配方管理器后，界面如图 3.21 所示，点击“打开”后，弹出图 3.22 所示的配

方管理器的配置界面。

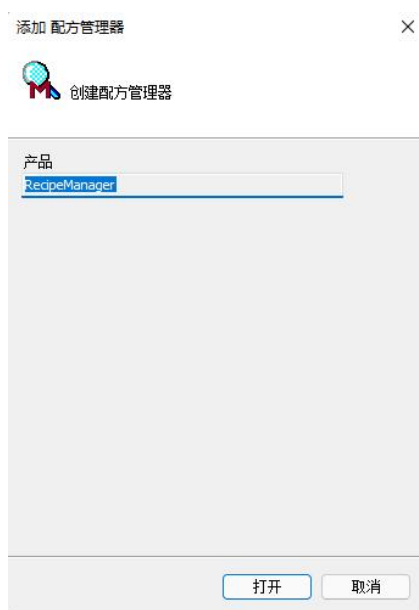


图 3.21 添加配方管理器



图 3.22 配方管理器配置界面

配方“存储类型”可以选择文本或二进制类型；

可以通过“文件路径”指定具体的存储位置，也可以定义存储配方的“文件扩展名”；文本的存储会根据所选的“分离器”将其分开，“分离器”只有在存储类型为文本时生效。

在配方管理器的配置页面中还显示了配方所有的“可用列”，右侧为当前配方定义的列，即“选择列”，“选择列”中的内容会被存储。可通过单击>或<图标按钮在“可用列”和“选择列”之间移动，也可以通过使用>>或<<图标按钮将所有条目一次性移动。图标按钮上和下可用来调整配方定义的顺序，存储文件会按照“选择列”中设定的顺序保存。

配置页面设置完成后，在设备导航树中通过鼠标右击“配方管理器”选择“添加对象”→“配方定义...”添加新的配方定义，如图 3.23 所示。

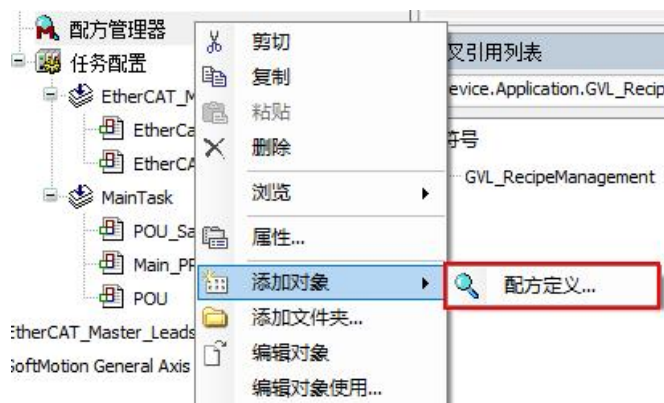


图 3.23 添加新的配方

然后设置新配方的名称。如图 3.24 所示。

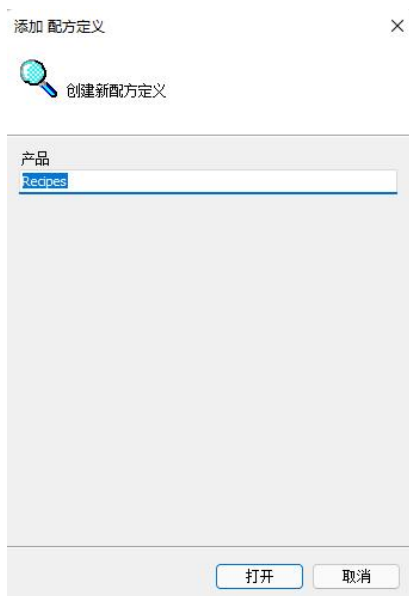


图 3.24 设置新配方的名称

点击打开后，添加的配方页面如图 3.25 所示。用户可以通过鼠标右击选择“添加新配方”来配置不同的配方值，也可以选中配方值（如配方 1）来删除配方、加载配方或保存配方。

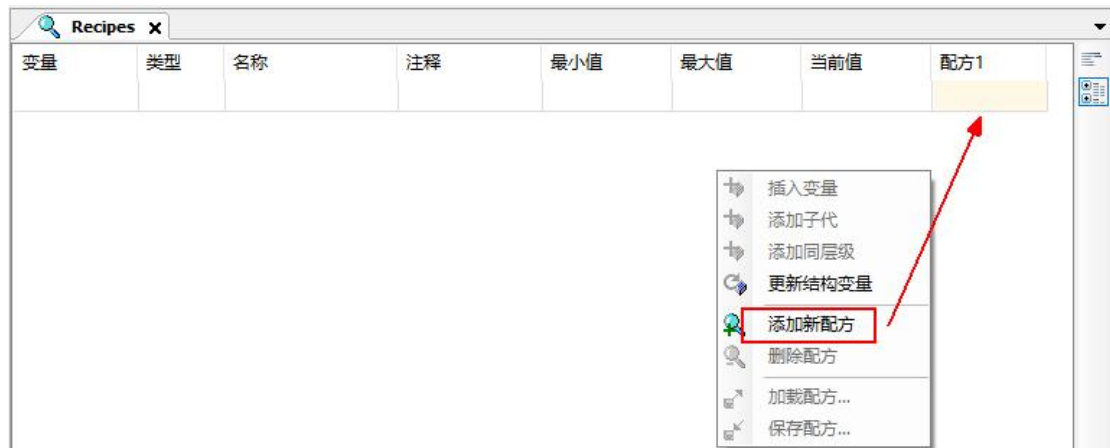


图 3.25 设置配方

除了在 LeadSys 界面中直接编辑配方文件，用户也可以通过“Recipe Management”库文件，对配方进行创建、删除、加载、保持等操作。如图 3.26 所示。

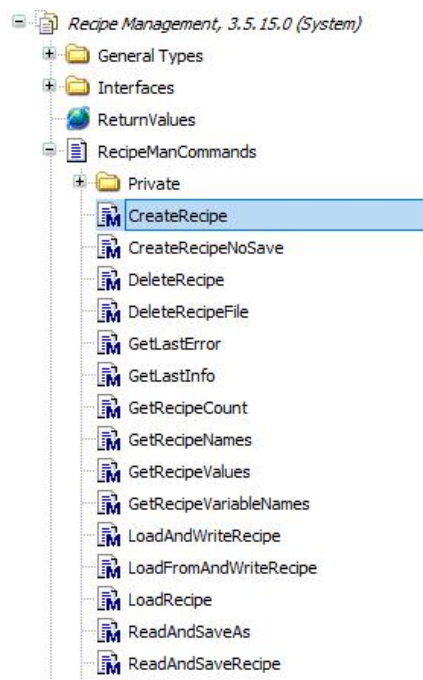


图 3.26 配方文件

添加完配方后，用户需在程序中定义配方中需要使用的变量，然后在配方中插入变量，插入变量方式如图 3.27 所示。

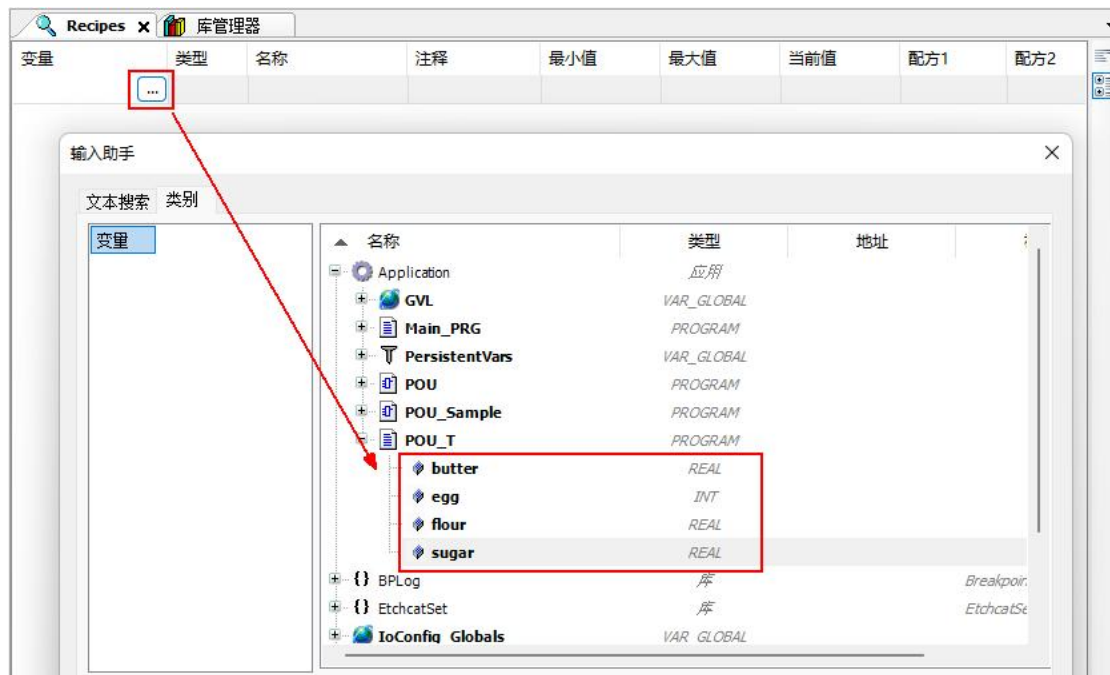


图 3.27 在配方中插入变量

变量插入完成后，用户可以在配方 1、配方 2 中填写参数，如图 3.28 所示。

改变的配方定义配置会在下次下载或在线变更时传输								
变量	类型	名称	注释	最小值	最大值	当前值	配方1	配方2
POU_T								
butter	REAL						50	70
egg	INT						1	3
flour	REAL						0.5	1
sugar	REAL						30	10

图 3.28 填写配方参数

在线登入后，用户可以通过“写配方”将当前选中的配方值写入控制器中，也可以通过“读配方”从控制器中读取当前值到配方中。如图 3.29 所示。

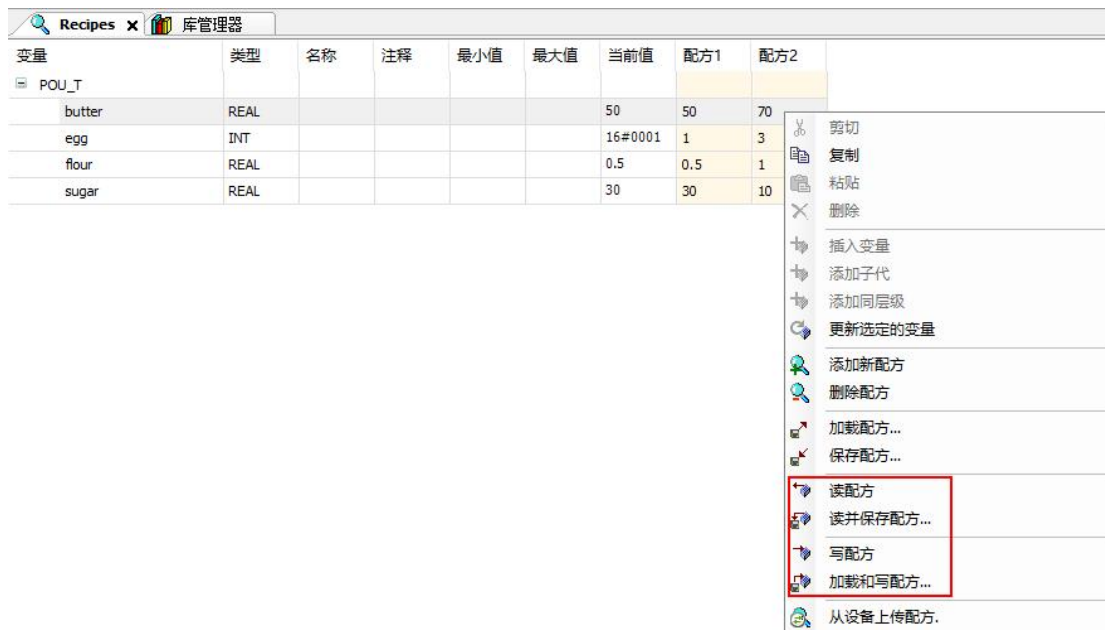


图 3.29 读、写配方

3.4.新建 POU

POU 是程序组织单元，由声明区和代码区两部分组成，是用户程序的最小软件单元。

LeadSys 软件编程时创建的 POU，按功能划分，可以分为函数（FUN）、功能块（FB）和程序（PRG）。用户编写程序时，FUN、FB 和 PRG 三者之间可以相互调用。PRG 可以调用 FUN、FB 和 PRG；FB 可以调用 FB 和 FUN；FUN 仅能调用 FUN。三者之间的调用关系如图 3.30 所示。

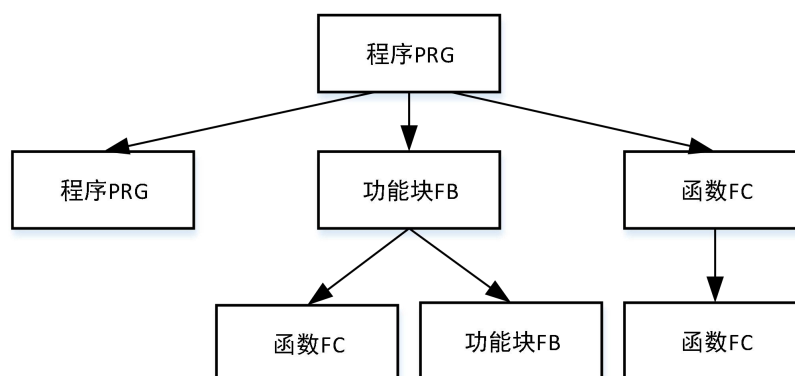


图 3.30 函数（FUN）、功能块（FB）和程序（PRG）的关系

添加 POU 方法为：用鼠标在设备导航栏窗口中的“Application”上右击，找到“添

加对象”后选择“POU”，如图 3.31 所示。

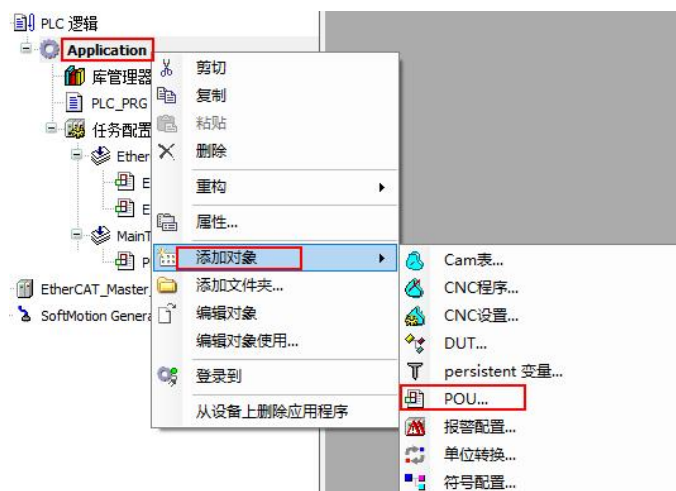


图 3.31 添加 POU 的方法

在弹出对话框中，用户可以选择添加程序、功能块或函数，在“实现语言”下拉菜单中可选择对应的编程语言。如图 3.32 所示。

创建 POU 之后，用户可以在代码区编写自己的程序代码，程序中使用的变量都需要在声明区定义，如图 3.33 所示。

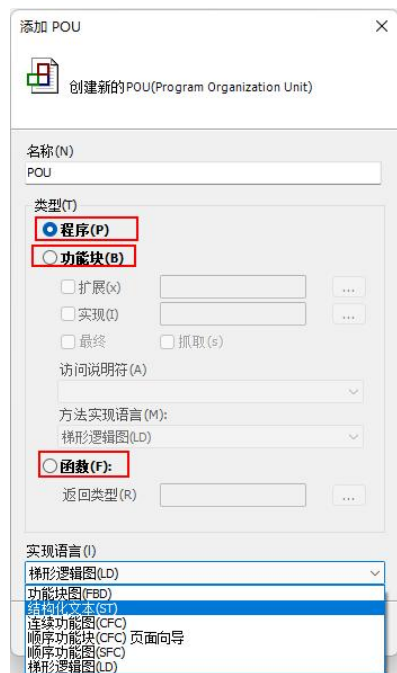


图 3.32 设置 POU 的类型

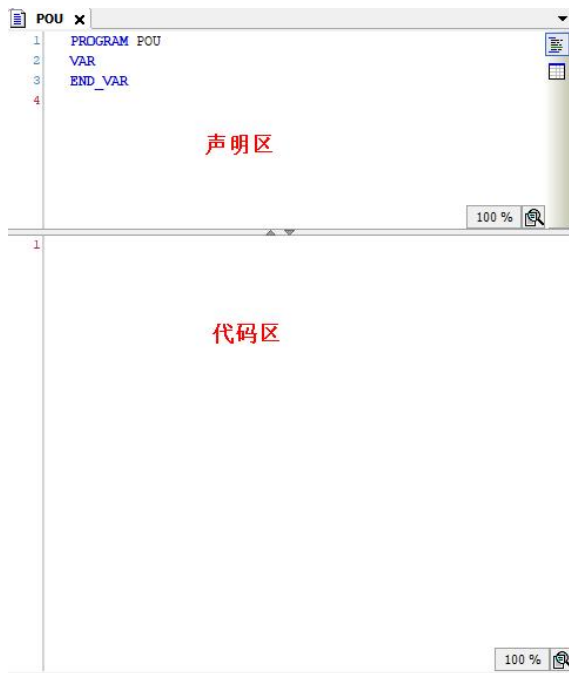


图 3.33 POU 的编辑界面

3.5.功能块与函数

3.5.1. 功能块

功能块 FB 是把程序中反复使用的部分程序抽象封装成一个通用程序块，它可以在程序中被任何一种编程语言重复调用。在编程中使用封装的功能块，不仅可提高程序的开发效率，也减少了编程中的错误，从而改善了程序质量。

功能块在执行时能够产生一个或多个值，功能块拥有自己的内部变量，这些内部变量在控制器内执行时需要分配内存，从而构成自身的状态特征。因此，对于相同参数的输入变量值，可能存在不同的内部状态变量，所以会得到不同的计算结果。

功能块是抽象的数据类型，因此功能块需要实例化后才能被程序调用和执行。由于是面向对象的编程方式，所以功能块也可以派生出“子”功能块。“子”功能块除了具有“父”功能块的属性外，还可以具有自己附加的特性，可以形象的称为“功能块的扩展”。

● 新建功能块

在设备导航栏窗口中的“Application”上，鼠标右键单击“添加对象”，选择“POU”，在弹出对话框中，选择“功能块”，点击“打开”即完成新建功能块如图 3.34 所示。自定义功能块的界面如图 3.35 所示。

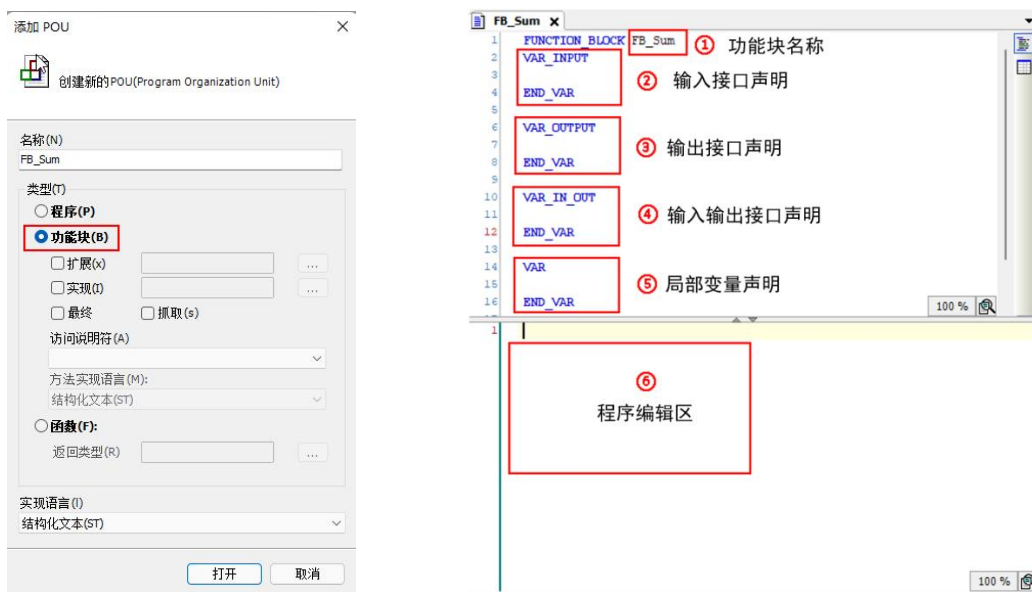


图 3.34 新建功能块图

图 3.35 功能块的编辑界面

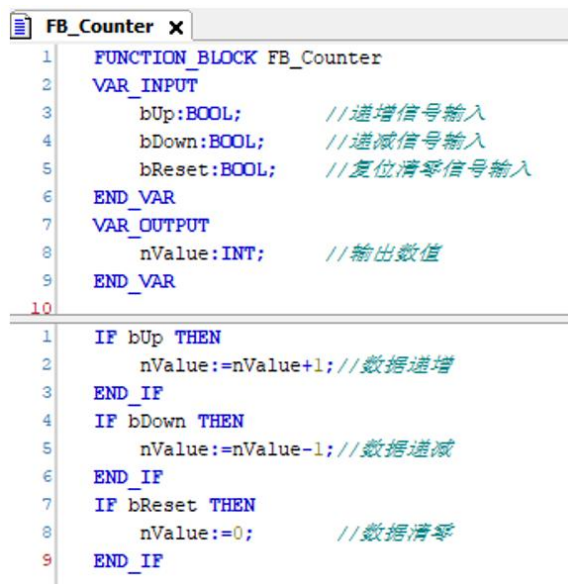
● 功能块编程

功能块程序编写语言可以使用 IEC61131-3 规范中的任意一种，在功能块程序里面，可以调用函数或功能块。

功能块编程过程中应注意以下事项：

- 1) 为确保功能块不依赖于硬件，功能块的变量声明中不允许使用固定地址变量，如：IX0.1、QW2，作为局部变量，但在调用时可以给其赋值。
- 2) 使用 VAR_INPUT 和 VAR_OUTPUT 会占用过多的内存。因此，在功能块编程时，尽可能使用 VAR_IN_OUT 替代，减少对存储区的占用。
- 3) 功能块支持添加方法 Method，扩展 Extend、实现 Implement 等面向对象的编程方式。
- 4) 功能块需要实例化，才能被调用，同一个程序多次调用同一个功能块时，建议定义不同的实例。

图 3.36 是一个自定义的功能块，实现递增/递减的功能。该功能块分为 3 个输入：增计数、减计数和复位；当前计数值为输出。功能块的变量声明及代码如图 3.36 所示。



```

1  FUNCTION_BLOCK FB_Counter
2  VAR_INPUT
3      bUp:BOOL;      //递增信号输入
4      bDown:BOOL;    //递减信号输入
5      bReset:BOOL;   //复位清零信号输入
6  END_VAR
7  VAR_OUTPUT
8      nValue:INT;     //输出数值
9  END_VAR
10
11 IF bUp THEN
12     nValue:=nValue+1; //数据递增
13 END_IF
14 IF bDown THEN
15     nValue:=nValue-1; //数据递减
16 END_IF
17 IF bReset THEN
18     nValue:=0;        //数据清零
19 END_IF
    
```

图 3.36 定义功能块 FB_Counter

● 功能块调用

功能块需要实例化后才能在程序中调用。

仍以功能块 FB_Counter 为例：为了方便查看功能块接口，我们新建 POU 程序的编程语言选择为 FBD，在该 POU 中调用 FB_Counter 功能块。在程序编辑区右侧的“工具箱”→“基本指令”中，选择“运算块”，按住鼠标左键拖拽到程序编辑区，填写完输入输出接口即可。参见图 3.37。



图 3.37 实例化功能块 FB_Counter

图 3.38 是功能块 FB_Counter 的运行结果。当输入信号 bUp 为 TRUE 时，输出值不断累加；当 Down 为 TRUE 时，输出值不断递减；当 bReset 为 TRUE 时，输出值清零。

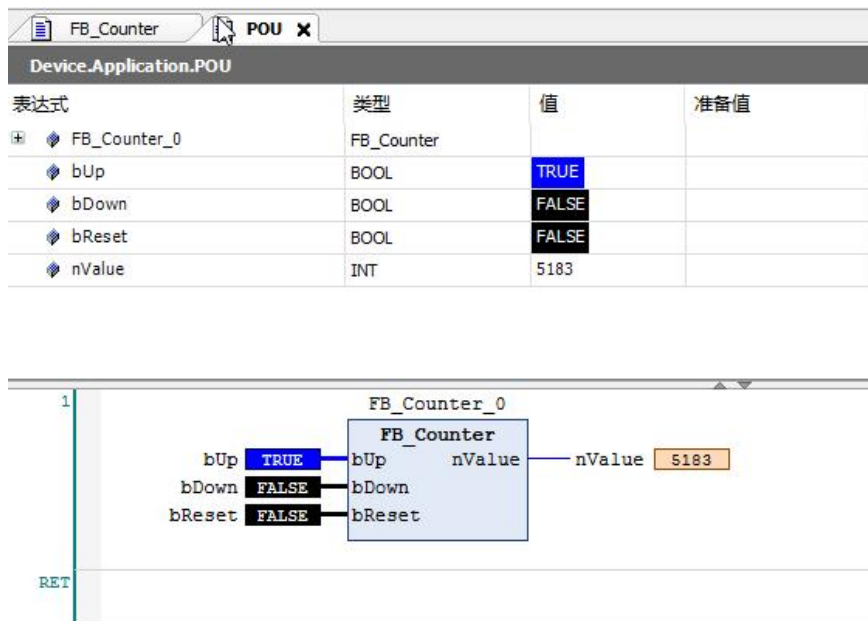


图 3.38 功能块 FB_Counter 运行情况

3.5.2. 函数

函数 FC 是至少有一个输入变量、无私有数据、仅有一个返回值的基本算法单元，可以被函数、功能块、程序调用。函数的一个重要特性是它不能使用内部变量存储数值。也就是说，只要给定相同的输入参数，必定得到相同的输出，这是函数 FC 与功能块 FB 之间的主要区别。

● 新建函数

用鼠标在设备导航栏窗口中的“Application”上，右键单击“添加对象”，选择“POU”，在弹出对话框中，选择“函数”，点击“打开”即完成新建函数，如图 3.39 所示。自定义函数的界面如图 3.40 所示。



图 3.39 新建函数

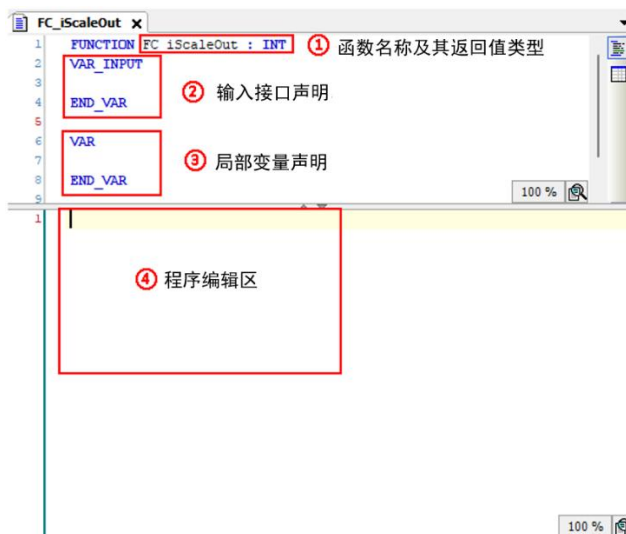


图 3.40 函数的编辑界面

● 函数编程

函数的编写语言可以使用 IEC61131-3 规范中的任意一种，函数名即为函数的返回值，也可以理解为函数的输出值。

函数编程应注意以下事项：

- 1) 函数可以拥有多个输入变量，但只能有一个返回值，由于没有限制返回值的数据类型，用户甚至可以将一个结构体作为返回值。
- 2) 函数的重要特性是它不能在内部变量存储数值，这点与功能块截然不同。
- 3) 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。
- 4) 函数只能调用函数，不能调用功能块。
- 5) 函数的输入端口 VAR_INPUT 可以是空的、常数、变量或者函数调用。

下面给出的自定义函数的例程可实现“模拟量输入信号转换为数字量信号”的功能。在模拟量信号转换为数字量信号时，常用的模拟量：电流信号有 0~20mA、4~20mA，电压信号有 0~10V、-10~10V。根据选定的输入参数的类型，将其转换为数字量值。FC_iScaleOut 函数声明及代码如图 3.41 所示。

```

1  FUNCTION FC_iScaleOut : INT
2  VAR_INPUT
3      rOut:REAL; //实际设备物理量输出值
4      rPhyMin:REAL; //物理量量程最小值
5      rPhyMax:REAL; //物理量量程最大值
6      eTerm:E_TerminalType; //输入信号类型
7  END_VAR
8  VAR
9      rTerMin:REAL;
10     rTerMax:REAL;
11     rPhyRange:REAL;
12     rTerRange:REAL;
13     rTerOut:REAL;
14 END_VAR
15
16 rTerMax:=32768.0;
17 CASE eTerm OF
18     eTerm_0mA_20mA:rTerMin:=0.0;
19     eTerm_4mA_20mA:rTerMin:=0.0;
20     eTerm_0V_10V:rTerMin:=0.0;
21     eTerm_neg0V_10V:rTerMin:=32768.0;
22 ELSE
23     rTerMin:=-32768.0;
24 END_CASE
25 rPhyRange:=rPhyMax-rPhyMin;
26 rTerRange:=rTerMax-rTerMin;
27 IF rPhyRange > 0.0 AND rTerRange > 0.0 THEN
28     rTerOut:=rTerMin+(rTerRange*(rOut-rPhyMin)/rPhyRange);
29 ELSE
30     rTerOut:=0.0;
31 END_IF
32 FC_iScaleOut:=REAL_TO_INT(rTerOut);

```

图 3.41 函数 FC_iScaleOut 的声明及代码

例程的输入变量定义中使用的枚举类型 E_TerminalType 定义如图 3.42 所示。

```

1  TYPE E_TerminalType :
2  (
3      eTerm_0mA_20mA,
4      eTerm_4mA_20mA,
5      eTerm_0V_10V,
6      eTerm_neg0V_10V
7  );
8  END_TYPE

```

图 3.42 枚举类型 E_TerminalType 的定义

● 函数调用

调用函数无需实例化。为了方便查看函数接口，我们新建 POU_Sample 程序的编

程语言选择为 FBD，在 POU_Sample 中调用函数 FC_iScaleOut。在程序编辑区右侧的“工具箱”→“基本指令”中，选择“运算块”，按住鼠标左键拖拽到程序编辑区，填写完输入输出接口即可。参见图 3.43。

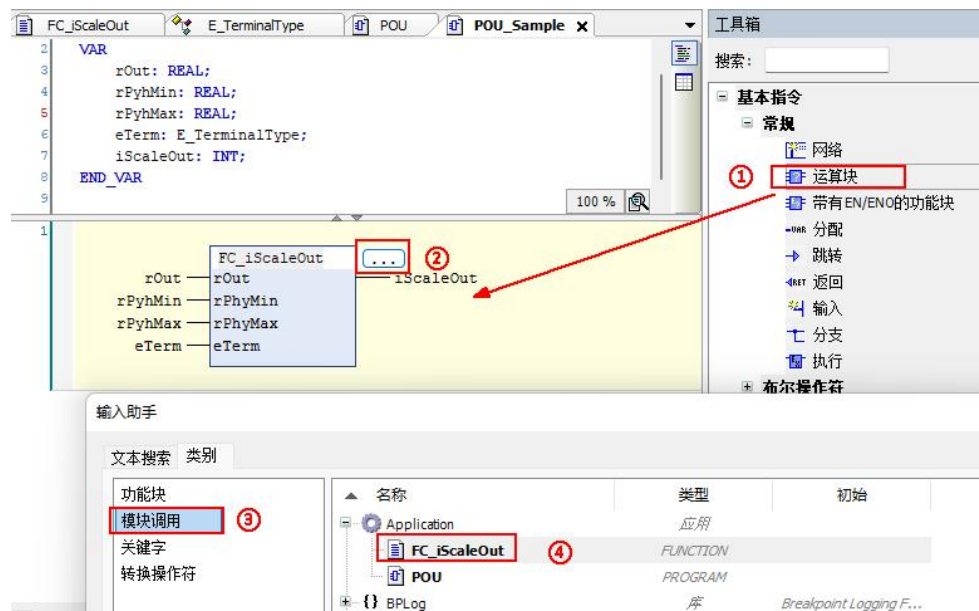


图 3.43 调用函数 FC_iScaleOut

程序调用 FC_iScaleOut 函数结果如图 3.44 所示。

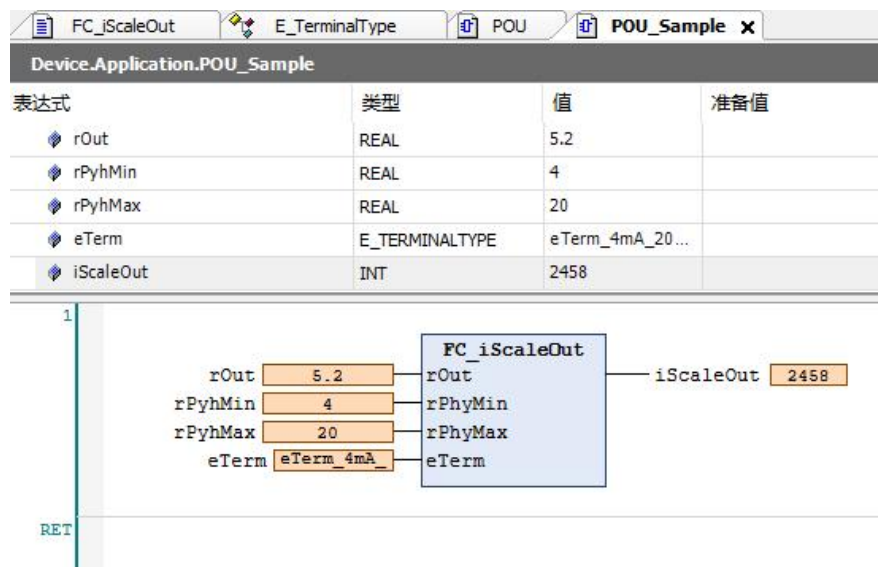


图 3.44 函数 FC_iScaleOut 的运行结果

通过函数和功能块的定义和使用，我们发现两者调用的表达式很相似，但两者还是具有明显的区别，主要区别见表 3.12。

表 3.12 函数和功能块的区别

	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	可以调用函数，但不能调用功能块	可以调用功能块或函数

第四章 编程基础 2：编程语言

4.1. LeadSys 支持的编程语言类型

LeadSys 支持以下 6 种编程语言，编程者可以根据实际的需要任意选择编程语言。

- 指令表 IL (Instruction List)
- 梯形图 LD (Ladder Diagram)
- 功能块图 FBD (Function Block Diagram)
- 结构化文本 ST (Structured Text)
- 连续功能图 CFC (Continuous Function Chart)
- 顺序功能图 SFC (Sequential Function Chart)

编程语言的多样性是 LeadSys 的一大优点。对于 LeadSys 编程而言，建议编程者在选择编程语言时应根据实际的编程方便来选择编程语言，而不是在整个程序中仅使用 LD。

那么，编程者如何根据具体情况选择编程语言呢？

从优化程序和编程便利性的等方面考虑，有如下建议：

涉及到算法部分的程序请选择 ST 语言，编写的程序往往简洁而高效；

涉及到流程控制部分的程序，请选择 SFC 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的程序，请选择 LD 语言，编写的联锁、互锁等功能逻辑简单易懂；

涉及到功能块部分的程序，请选择 CFC 或者 FBD，编写的程序会形成一个脉络清晰的网状电路图，容易读懂。

当然，在实际编程时，用户也可以根据自己的习惯来选择编程语言，虽然实现的方法不同，但是都能得到相同的结果。

4.2. 指令表（IL）

指令表 IL 编程语言是基于累加器的编程语言。与汇编语言类似，是一种助记符

编程语言。

指令表由一系列的指令组成。每条指令占据一行，包含一个操作符和一个或多个用逗号隔开的操作数。每行的指令前面可以有标号，标号后要有冒号。每行结束处可以加入注释。每行指令之间可以插入空行。

指令表每条指令首先使用 LD 操作符将值装载到累加器中，然后累加器将被执行，执行的结果存储在累加器中，使用 ST 操作符可以将该结果赋给其它变量。

IL 编程语言各元素具体如下：

- 标号：指令前可加标号，后面跟冒号，程序跳转时使用，标识跳转指令的目的地，标号名称不能与操作符同名。
- 操作符：指令执行过程中所用的符号（如 LD/ANDN/ST/JMPC/S 等）或函数名。
- 操作数：可为变量、常量、寄存器地址、函数等。如：btn1 为变量，T1_PT 为常量，QX0.0 为输出点地址。

IL 编辑器中的例程如图 4.1 所示，其中①为标号，②为操作符，③为操作数。

1		②	③	
	LD	btn1	按钮1	
	ANDN	btn2	按钮2	
	ST	out1		
	JMPC	lable_1		
	LD	btn3	按钮3	
	S	%QX0.0		
2				
	lable_1:	①		
	CAL	T1(		
		IN:= T1_EN		
		PT:= T1_PT,		
		Q=> T1_Q,		
		ET=> T1_ET)		
3				
	LD	T1_Q		
	R	%QX0.0		

图 4.1 IL 例程

IL 常用操作符如表 4.1 所示。

指令表与梯形图有一一对应关系，在 LeadSys 软件中，指令表（IL）与梯形图 LD、功能块图 FBD 之间可以相互转换。通过选择菜单栏“FBD/LD/IL”→“视图”进行切换，如图 4.2 所示。

表 4.1 IL 常用操作符

操作数	修饰符	描述	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时，将操作数（bool 型）设为真	S bVar1
R		当累加器中的数为真时，将操作数（bool 型）设为假	R bVar1
AND	N,()	将累加器中的数和操作数（取反）逐位进行与操作	AND bVar2
OR	N,()	将累加器中的数和操作数（取反）逐位进行或操作	OR xVar
XOR	N,()	将累加器中的数和操作数（取反）逐位进行异或操作	XORN bVar1
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，其结果装入到累加器中	ADD iVar1
SUB	()	将累加器中的数减去操作数，其结果装入到	SUB

		累加器中	iVar2
MUL	()	将累加器中的数和操作数相乘, 其结果装入到累加器中	MUL ivar2
DIV	()	将累加器中的数除以操作数, 其结果装入到累加器中	DIV 44
GT	()	检查累加器中的数是否大于操作数, 其结果 (bool 型) 装入到累加器中	GT 23
GE	()	检查累加器中的数是否大于或等于操作数, 其结果 (bool 型) 装入到累加器中	GE iVar2
EQ	()	检查累加器中的数是否等于操作数, 其结果 (bool 型) 装入到累加器中	EQ iVar2
NE	()	检查累加器中的数是否不等于操作数, 其结果 (bool 型) 装入到累加器中	NE iVar1
LE	()	检查累加器中的数是否小于或等于操作数, 其结果 (bool 型) 装入到累加器中	LE 5
LT	()	检查累加器中的数是否小于操作数, 其结果 (bool 型) 装入到累加器中	LT cVar1
JMP	CN	无条件 (有条件) 跳转到标号	JMPN next
CAL	CN	(有条件) 调用一个程序或功能块 (当累加器中的数为真时)	CAL prog1
RET		从当前 POU 中返回, 跳回到调用的 POU 中	RET

RET	C	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	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

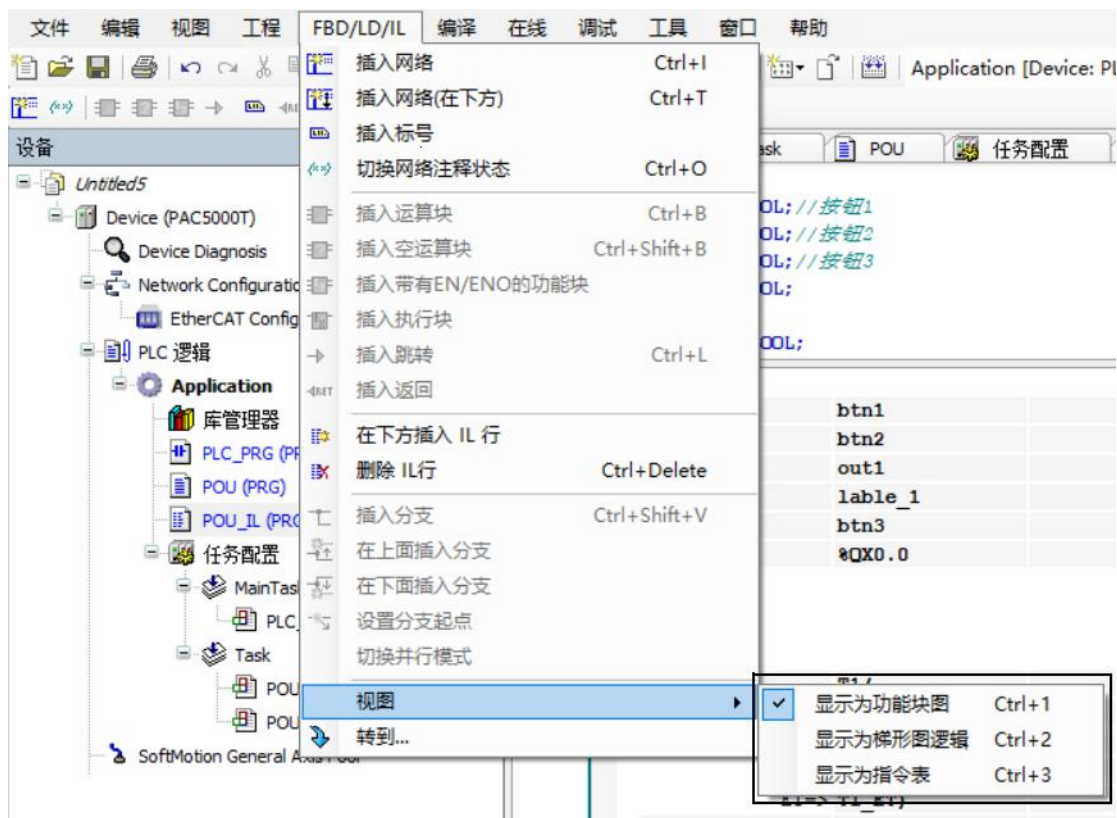


图 4.2 IL 与 LD、FBD 之间的转换

4.3.结构化文本（ST）

结构化文本 ST 是用结构化的描述文本来编写程序的一种编程语言，与 PASCAL 或 C 语言相似。ST 语言的特点是“高级文本编程”和“结构化”，具有高效、快捷、简洁的优点。适合于算法和结构较为复杂、用其它编程语言（如 LD、FBD 等）难以实现的情况。

不同于 IL 语言，ST 语句循环中可以包含众多的语句，因此允许开发复杂的结构。ST 编程语言各种元素具体如下：

- 表达式：由操作数和操作符组成的结构，在执行表达式时有返回值。
- 操作数：操作数表示变量、数值、地址、功能块等。
- 操作符：操作符是执行运算过程中所用的符号。
- 语句：语句用于将表达式返回的值赋给实际参数，并构造和控制表达式。每条语句要由分号“；”结尾。语句中的字母不分大小写！

ST 程序执行顺序由编排在 ST 编辑器中的语句顺序决定：从左到右，从上到下。这个顺序只能通过插入循环语句而改变。

ST 编辑器窗口如图 4.3 所示。



图 4.3 ST 语言编辑器

4.3.1. 表达式

ST 语言中的表达式是一个可计算出一个数值结果并将其返回给变量的结构。表达式由操作符和操作数组成。操作数可以是常量、变量、函数调用的返回值或其他表达式。例如：

123, t#10ms, 'abc' // 常量

ivar1, global1 // 变量

Func(in1,in2) // 函数调用

a + b, a AND b, x*y // 操作符调用

计算表达式时将根据操作符的优先级所定义的顺序将操作符应用于操作数表。首

先执行表达式中具有最高优先级的操作符，接着执行具有次优先级的操作符；以此类推，直到完成整个计算过程。优先级相同的操作符将根据它们在表达式中的书写顺序从左至右执行。可使用括号更改此顺序。

例如：如果 A、B、C 和 D 的值分别为 1、2、3 和 4，并按以下方式计算：
A+B-C*D，结果为-9。（A+B-C）*D，结果则为 0。

如果操作符包含两个操作数，则先执行左边的操作数。例如在表达式 SIN(x)*COS(y)中，先计算表达式 SIN(x)，后计算 COS(y)，然后计算二者的乘积。

4.3.2. 操作符

操作符是一种符号，它表示要执行的算术运算、要执行的逻辑运算、功能编辑调用。操作符是泛型的，即它们自动适应操作数的数据类型。

ST 常用的操作符及其执行的操作如表 4.2 所示。

表 4.2 ST 常用的操作符

操作符	执行的操作	优先级
(表达式)	括号	高  低
函数名（参数列表，由逗号分隔）	调用函数	
EXPT	指数运算	
-, NOT	取反	
*	乘	
/	除	
MOD	取余数	
+	加	
-	减	
<, >, <=, >=	比较运算	

=	等于	
≠	不等于	
AND	与	
XOR	异或	
OR	或	

4.3.3. 操作数

操作数可以是：地址、数值、变量、数组变量、结构体变量、数组/结构体变量的元素、功能调用、功能块输出等。处理操作数的语句中的数据类型必须相同。如果需要处理不同类型的操作数，则必须预先执行类型转换，否则可能存在数据丢失的情况。

在下面示例中，整数变量 Var1 在添加到实数变量 R1 中之前，先转换为实数变量。

```
R2:=R1+INT_TO_REAL(Var1);
```

4.3.4. 语句

ST 编程语言常用的语句主要分为以下 4 类：

- 赋值语句：:=
- 条件语句：IF、CASE
- 循环语句：WHILE、CONTINUE、FOR、REPEAT
- 跳转语句：JMP
- 跳出语句：EXIT、RETURN

对以上各语句的使用方法说明如下。

1) 赋值语句

赋值语句用表达式的求值结果替代单元素或多元素变量的当前值。

语法: $A := B$;

其中: A 是变量名称; B 是变量、求值的表达式或 FB/FC

$:=$ 是操作符, 将右边变量、表达式或 FB/FUN 的值赋给左边的变量

分别位于赋值操作符的左侧和右侧的两个变量数据类型必须相同。

数组是个特例。显式启用后, 也可对长度不同的两个数组执行赋值操作。

示例:

- 将数值赋给变量。

$iVar := 30$; // 将常数 30 赋给变量 iVar。

- 将运算值赋给变量

$X := (A+B-C)*D$; // 将 $(A+B-C)*D$ 的运算结果赋给变量 X。

- 将 FUN/FB 的值赋给变量

$A := MY_TOF.Q$; // 将 MY_TOF 功能块 (TOF 功能块的实例) 的

// Q 输出值赋给变量 A。(这不是功能块调用)。

$B := C \text{ MOD } A$; // 调用模数 (MOD) 功能并将计算结果赋给变量 B。

2) IF 语句

使用 IF 指令可以检查条件, 并根据此条件执行相应的语句。当相关的布尔表达式求值为 TRUE 时执行一组语句 (语句块), 条件为 FALSE 时不执行, 结束语句, 或者执行 ELSE 或者 ELSIF 后面的语句块。

常用的 IF 语句语法结构有以下 3 种:

- a) IF 条件 A THEN //当条件 A 满足时, 执行语句块 A。

语句块 A;

END_IF

- b) IF 条件 A THEN //当条件 A 满足时, 执行语句块 A;

语句块 A;

ELSE //否则, 执行语句块 B。

语句块 B;

END_IF

c) IF 条件 A THEN //当条件 A 满足时，执行语句块 A。
 语句块 A;
 ELSIF 条件 B THEN //当条件 B 满足时，执行语句块 B。
 语句块 B;
 ...
 ELSIF 条件 N-1 THEN //当条件 N-1 满足时，执行语句块 N-1。
 语句块 N-1;
 ELSE //如果以上条件都不满足，则执行语句块 N。
 语句块 N;
END_IF //指令结束。

示例如图 4.4 所示。当 iVarA 的值大于 5 时 iVarB 的值为-10，否则它的值为 10。

```
1  IF iVarA>5 THEN
2      iVarB:=-10;
3  ELSE
4      iVarB:=10;
5  END_IF
```

图 4.4 IF 语句示例

3) CASE 语句

CASE 语句包含一个整型数据或枚举数据控制变量和多组表达式列表。每组表达式都标记了一个或多个可应用的整数值或枚举值的范围。如果控制变量与其中一个选择值相等，则执行该组语句块。否则，执行 ELSE 后的语句块。

OF 语句指示范围的开头。END_CASE 关键字标记语句的结尾。

语法如下：

CASE 控制变量 OF
 选择值 1:
 语句块 A;
 选择值 2:

```
        语句块 B;  
        选择值 3,选择值 4: //控制变量等于选择值 3 或 4 时执行语句块 C  
        语句块 C;  
        选择值 5..选择值 9: //控制变量等于选择值 5~9 之内的值时,  
                               // 执行语句块 D  
        语句块 D;  
        ...  
        ELSE //可选项  
        语句块 E;  
    END_CASE
```

当使用 IF 指令有过多分层，或者需要使用多个 ELSIF，才能完成程序功能时，使用 CASE 指令替代 IF 指令，可以简化程序，并且能提高程序的可读性。示例如图 4.5 所示。当 iVarA=1 或 5 时，bVar1 为真；当 iVarA=2 时，bVar2 为真；当 iVarA=10~20 之间的数值时，bVar3 为真；否则，bVar1、bVar2、bVar3 均为假。

```
1  CASE iVarA OF  
2      1, 5: bVar1 := TRUE;  
3      2: bVar2 := TRUE;  
4      10..20: bVar3:= TRUE;  
5  ELSE  
6      bVar1 := bVar2 :=bVar3 :=FALSE;  
7  END_CASE
```

图 4.5 CASE 语句示例

4) FOR 循环

FOR 语句用于在发生次数可预先确定的情况下。否则可使用 WHILE 或 REPEAT 语句。FOR 语句会重复执行 FOR 语句到 END_FOR 语句之间的语句块，执行次数由起始值、结束值和控制变量决定。

语法：

```
FOR 控制变量:= 循环起始值 TO 循环结束值 {BY 递增步长} DO  
    语句块;
```

END_FOR

其中，{}内语句可根据需要省略，省略时步长默认为 1。

若要中断循环语句，请执行 EXIT 语句。

示例如图 4.6 所示。此程序的循环控制变量为 Counter，循环开始时控制变量初值为 1，每一次循环 Counter+1；当 Counter 等于 5 时，执行完 FOR 循环内容后，退出循环，执行下一条语句。语句 iVar1:=iVar1*2 一共执行 5 次；假设 iVar1 的初始值是 1，那么循环结束后，iVar1 的值为 32。

```
1  FOR Counter:=1 TO 5 BY 1 DO
2      iVar1:=iVar1*2;
3  END_FOR
```

图 4.6 FOR 语句示例

5) 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 循环的结束条件不是指定的循环次数，而是任意的逻辑表达式。当该表达式叙述的条件满足时，执行循环。

当循环条件为真时，WHILE 语句重复执行其后的语句块，直到其循环条件为假。

如果一开始该循环条件就为假，则根本不执行语句块。

DO 语句标识可重复执行的语句块的头，END_WHILE 关键字标记语句块的尾。

可以使用 EXIT 语句提前终止循环。

语法：

WHILE 循环条件 DO

语句块;

END_WHILE

示例如图 4.7 所示。此程序只要 Counter 不等于 0，则一直会执行 WHILE 循环中的指令，直到 Counter 等于 0 为止。每执行一次循环，通过语句 Counter := Counter-1 使 Counter 的值减 1，当 Counter 等于 0 时，循环结束。

```
1  WHILE Counter<>0 DO
2      iVar1 := iVar1*2;
3      Counter := Counter-1;
4  END_WHILE
```

图 4.7 WHILE 语句示例

6) REPEAT 循环

REPEAT 循环与 WHILE 循环一样，也是没有明确循环次数的循环。与 WHILE 循环的区别在于，REPEAT 循环在指令执行以后，才检查结束条件。因此无论结束条件怎样，循环至少执行一次。

UNTIL 语句标记结束条件。END_REPEAT 语句标记语句的结尾。可以使用 EXIT 语句提前终止循环。

语法：

```
REPEAT
    语句块;
UNTIL 循环结束条件
END_REPEAT
```

语句一直执行，直到循环结束条件为 TRUE 时，REPEAT 循环结束。如果一开始就为 TRUE，则循环只执行一次。

示例如图 4.8 所示，将 WHILE 语句的例程改写为 REPEAT 语句。

```
1  REPEAT
2      iVar1 := iVar1*2;
3      Counter := Counter-1;
4  UNTIL Counter < 0
5  END_REPEAT
```

图 4.8 REPEAT 语句示例

在一定意义上来说，WHILE 循环和 REPEAT 循环比 FOR 循环功能更强大，因为不需要在执行循环之前计算循环次数。因此，在有些情况下，用 WHILE 循环和 REPEAT 循环两种循环就可以了。然而，如果清楚知道循环次数，那么 FOR 循环更简单。

7) CONTINUE 语句

CONTINUE 语句用于在满足结束条件前终止循环语句(FOR、WHILE 或 REPEAT)的本次循环, 进入下次循环。如果 CONTINUE 语句位于嵌套的重复语句内, 则会继续最里层的循环。

语法:

CONTINUE;

示例如图 4.9 所示。此程序使用 CONTINUE 指令, 当 iVar1 等于 0 时, 直接结束本次循环, 开始下一次循环, 以避免指令“iVar2:=iVar2/iVar1”中对 iVar1 的除零操作。

```
1  FOR Counter:=1 TO 5 BY 1 DO
2      iVar1:= iVar1/2;
3      IF iVar1=0 THEN
4          CONTINUE;
5      END_IF
6      iVar2:=iVar2/iVar1;
7  END_FOR
```

图 4.9 CONTINUE 语句示例

8) JMP 语句

跳转语句可无条件的跳转到 label 所在的位置继续执行程序。JMP 指令容易造成程序结构混乱, 降低代码可读性, 不建议使用。

语法:

label:

.....

JMP label;

label 可以是任意确定的标识符, 它被放置在程序的开始处。JMP 指令必须有一个跳转目标, 也就是预定义的标签。当执行到 JMP 指令后, 程会跳转到指定的标签处继续执行。

示例如图 4.10 所示。当 i<10 时, 跳转回 label 所在行, 执行 i:=i+1。

```
1  i := 0;  
2  label: i := i+1;  
3  //.....  
4  IF i<10 THEN  
5    JMP label;  
6  END_IF
```

图 4.10 JMP 语句示例

这个例子中的功能同样可以通过使用 WHILE 或者 REPEAT 循环来实现。通常情况下，能够并且也应该避免使用跳转指令。

9) EXIT 语句

EXIT 语句用于在满足结束条件前终止循环语句（FOR、WHILE 或 REPEAT）。如果 EXIT 语句位于嵌套的循环语句内，则会离开最里面的循环（EXIT 所在的循环），并将执行循环结尾（END_FOR、END_WHILE 或 END_REPEAT）后的第一个语句。

语法：

EXIT;

示例如图 4.11 所示。当 iVar1 等于 0 时，FOR 循环结束。

```
1  FOR Counter:=1 TO 5 BY 1 DO  
2    iVar1:= iVar1/2;  
3    IF iVar1=0 THEN  
4      EXIT;  
5    END_IF  
6    iVar2:=iVar2/iVar1;  
7  END_FOR
```

图 4.11 EXIT 语句示例

10) RETURN 语句

结束 POU、函数或功能块的运行，程序返回至调用源处继续执行。

语法：

RETURN;

执行本指令前，请设置该 POU 的返回值、输出变量的值。若经常使用本指令，处理流程将变的复杂。敬请注意。

示例如图 4.12 所示。如果 b 为 TRUE，将不会执行 $a := a + 1$ ，而是直接退出 POU。

```
1  IF b:=TRUE THEN
2      RETURN;
3  END_IF
4  a := a + 1;
```

图 4.12 RETURN 语句示例

4.3.5. 调用功能块

库管理器中所有的函数和功能块都可以在 ST 语言中被调用。如果需要在 ST 中调用功能块，必须先对功能块进行实例声明。

对功能块进行实例声明后，可通过“实例名.参数名”，调用功能块中的参数；也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。

程序调用用户自行创建的功能块的方法与调用系统本身提供的标准功能块指令的方法相同。

实例针对功能块而言，每个功能块实例就是一个独立的、可完成特定逻辑功能的对象。不同的程序、不同的任务都可以定义和调用功能块的应用实例，每个调用实例都占用独立的内存，保留独立的逻辑状态。

语法：

```
name of FB instance(                                     //功能块实例名称
    FB input variable:=value or address,                 //输入引脚
    further FB input variable:=value or address,
    ...,
    FB output variables => value or address ,             //输出引脚
    further FB output variables => value or address ,
```

...);

如图 4.13 所示，当用户不想一个个输入功能块参数引脚名称时，可以在 ST 编辑器中鼠标右击，选择①“输入助手”→②“文本搜索”中输入功能块名称，可自动找到功能块，选中、点“确定”后，功能块即显示在编辑区。输入输出引脚已自动补全，用户只需为各参数引脚分配数值或变量，同时将功能块名称更换为实例名即可。

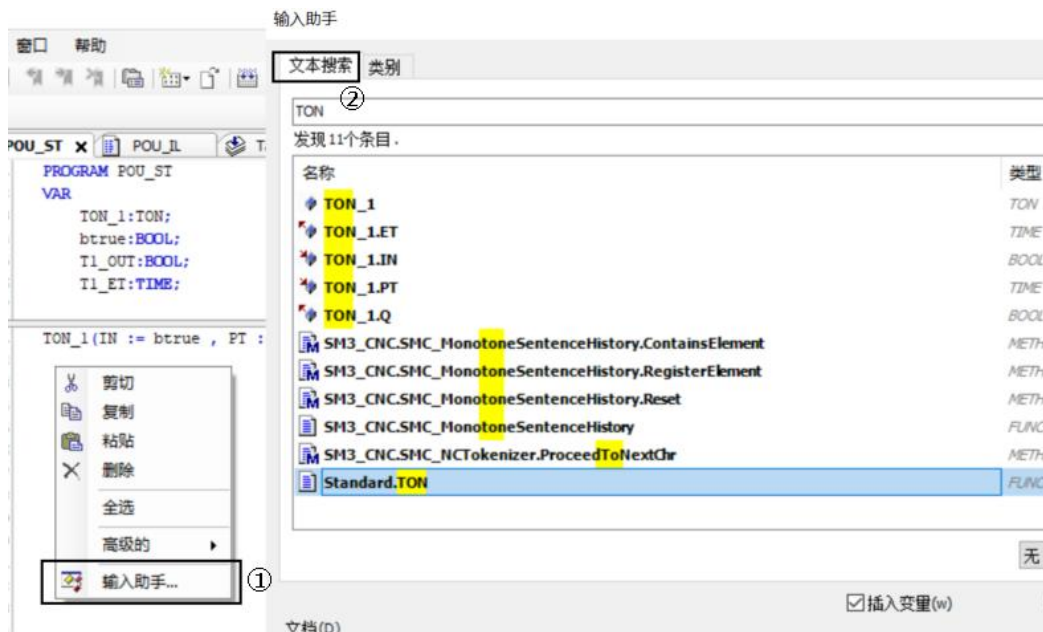


图 4.13 用输入助手查找功能块

示例：在 ST 中调用 TON 定时器，假设其实例名为 TON_1。结果如图 4.14 所示。

```

1  PROGRAM POU_ST
2  VAR
3      TON_1:TON;
4      btrue:BOOL;
5      Tl_OUT:BOOL;
6      Tl_ET:TIME;
7  END_VAR
1  TON_1(IN := btrue , PT := T#500MS , Q => Tl_OUT , ET => Tl_ET );
    
```

图 4.14 定义功能块的实例名

ST 程序调用函数的方法与调用功能块一样，只是函数没有实例名。故不再赘述。

4.3.6. 注释

注释是 ST 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 编辑器的变量声明区和程序编辑区都可以添加注释。

在 ST 语言中，有两种注释方法：

- 1) 注释以符号“(*”开始，以“*)”结束。这种注释方法允许多行注释
- 2) 注释以“//”开始，一直到本行结束。这是单行注释的方法。

示例如图 4.15 所示。

```
1  IF b:=TRUE THEN
2      (*a:=a-1;
3      b:=FALSE;*)
4  ELSE
5      a := a + 1;
6      //b:=TRUE;
7  END_IF
```

图 4.15 注释示例

4.4. 梯形图（LD）

梯形图语言是 PLC 程序设计中最常用的语言。由于与电气设计人员所熟悉的传统继电器电路图类似，因此梯形图语言得到了广泛的应用。

梯形图包含了一系列的网络（也称节），左右两边各有一个垂直的电流线（一般称为母线）限制其范围，中间是由触点、线圈、运算块（函数、功能块、执行块、动作、方法）、跳转、标签和连接线组成的电路图。

每一个网络的左边有一系列触点，这些触点根据布尔变量值的 TRUE 和 FALSE 来传递从左到右的“ON”和“OFF”的状态。每一个触点是一个布尔变量，如变量值为 TRUE，通过连接线从左到右传递“ON”状态；否则传递“OFF”的状态。

在网络最右边的线圈，根据左边的状态获得“ON”或“OFF”的状态，并相应地赋给一个布尔变量 TRUE 或 FALSE。

梯形图编辑界面如图 4.16 所示。

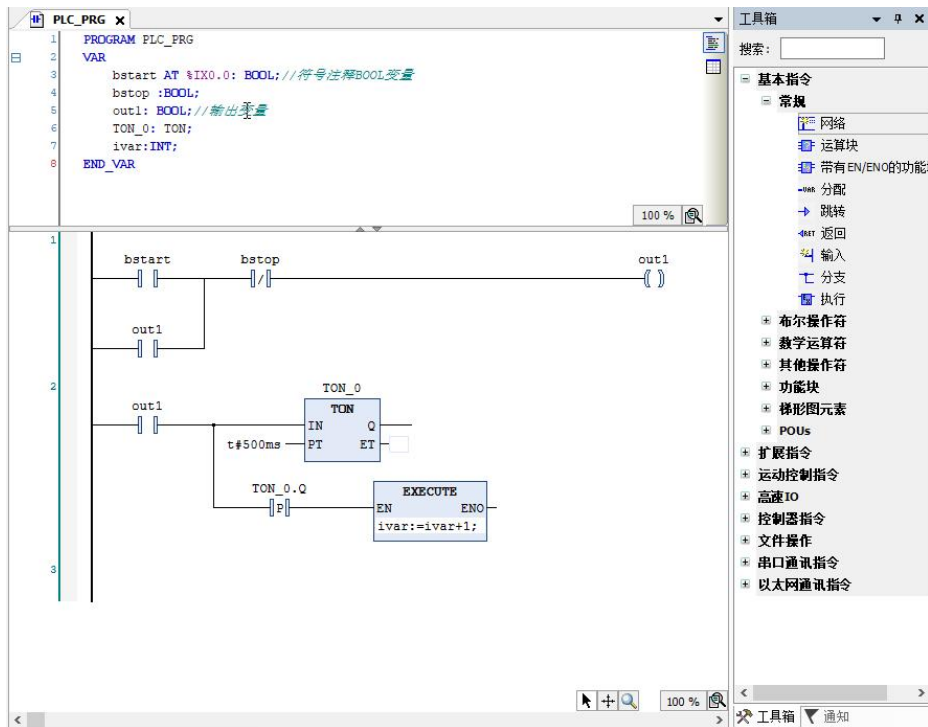


图 4.16 梯形图 LD 编辑界面

4.4.1. 梯形图元素

常用的梯形图元素包括网络、触点、线圈、运算块、执行块、分支、跳转、标号、返回。使用方法如下：

1) 网络

符号：

网络在左侧以垂直母线为界，其它所有的梯形图元素都位于网络右侧。网络中可插入网络标题和网络注释。网络标题和注释是否显示，可通过菜单“工具”→“选项”→“FBD,LD 和 IL 编辑器”→“General”来控制，如图 4.17 所示。

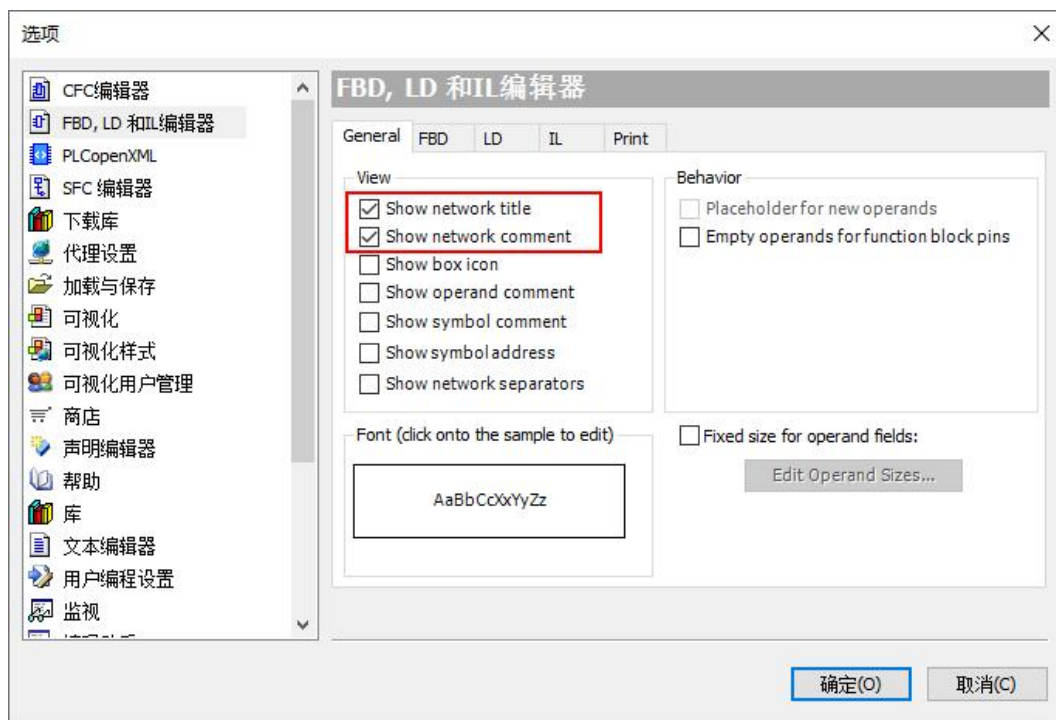


图 4.17 显示网络标题和注释的设置方法

Show network title（显示网络标题）：选中此项后，如果有网络标题，则会显示在横线上。

Show network comment（显示网络注释）：选中此项后，如果用户已经定义了网络注释，则这些注释将显示在网络左上角的横线下。

显示效果如图 4.18 所示。

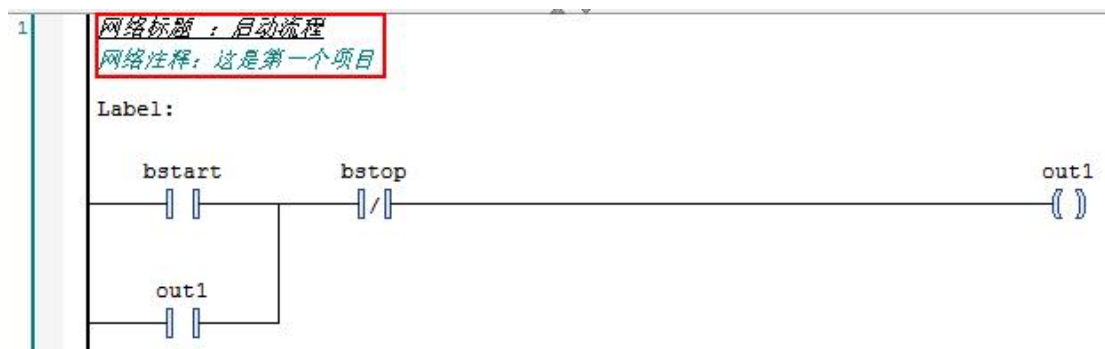


图 4.18 网络标题和注释的显示效果

2) 触点

触点分为常开和常闭触点。当选中一个已有触点，再调用触点命令时，插入的触点会出现在选中触点的左侧。

常开触点符号: 

常闭触点符号: 

触点从左到右传递"ON"(TRUE)或"OFF"(FALSE)状态。一个布尔型的变量被分配到每个触点。如果此变量为真,常开触点将向右传递真,否则右部为假。常闭触点传递值相反。

多个触点可串联也可并联,两个并联触点中只需一个触点为真,则平行线传输真,多个串联的触点要都为真,最后一个触点才能传输真,所以触点的串并联排列与串、并联电路相符。

3) 线圈

线圈分为线圈、取反线圈、置位线圈、复位线圈。通过右键菜单可以进行 4 种线圈类型之间的切换。

线圈符号: 

取反线圈符号: 

置位线圈符号: 

复位线圈符号: 

线圈位于网络的右端,它们只能平行的排列,即向上或者向下插入并联线圈。从左到右的逻辑运算结果,赋值给线圈变量。线圈变量只能是布尔类型,它的值为"ON"(TRUE)或"OFF"(FALSE),线圈也能在线圈符号取反,即线圈符号中的斜线。

4) 运算块

符号: 

运算块是一个复杂的元素,它可以是函数,如:定时器、计数器、算术运算、也可以是功能块。如果为功能块,则运算块框上面会增加编辑框来显示功能块实例。运算块可以有输入和输出,并可以由用户系统库提供或用户自行编写。不管怎样至少要有一个布尔输入和输出。

运算块也包括 EN/ENO 类型的运算块,EN/ENO 类型运算块,除了包含运算块本身所带的输入和输出外,还增加 EN 输入和 ENO 输出。

EN/ENO 运算块执行逻辑为：当 EN 为 TRUE 时执行运算块逻辑，执行完成后 ENO 为 TRUE，如果 EN 为 FALSE，不执行运算块，ENO 为 FALSE。EN/ENO 运算块输入连线只能连接在 EN 引脚，输出连线只能连接在 ENO 引脚。

5) 执行块

符号：

执行块是一个内嵌 ST 的块，在块中可以编写 ST 语句。特别适合在梯形图中编写计算类的程序。

6) 分支

符号：

该元素用于并联逻辑，分支形成一个非闭合的并联逻辑。

7) 标签

符号：

标签标示程序的执行位置信息，是跳转指令跳转的目的地。标签是字符串，标签不存在任何输入和输出引脚，需符合标识符命名规则，标签与跳转元素搭配使用。

8) 跳转

符号：

该元素可以实现程序的转移执行。当执行该指令时跳转到该指令引用的标签处。跳转指令存在唯一的 BOOL 值输入引脚。

9) 返回

符号：

执行该操作后，当前 POU 直接返回到调用该 POU 程序指令处。返回指令存在唯一的输入引脚，输入值为 BOOL 类型。

4.4.2. 工具箱

LD 编辑器提供了工具箱，工具箱中的指令支持拖拽的插入方式。工具箱可以通过“视图”菜单下“工具箱”子菜单打开，如图 4.19 所示。

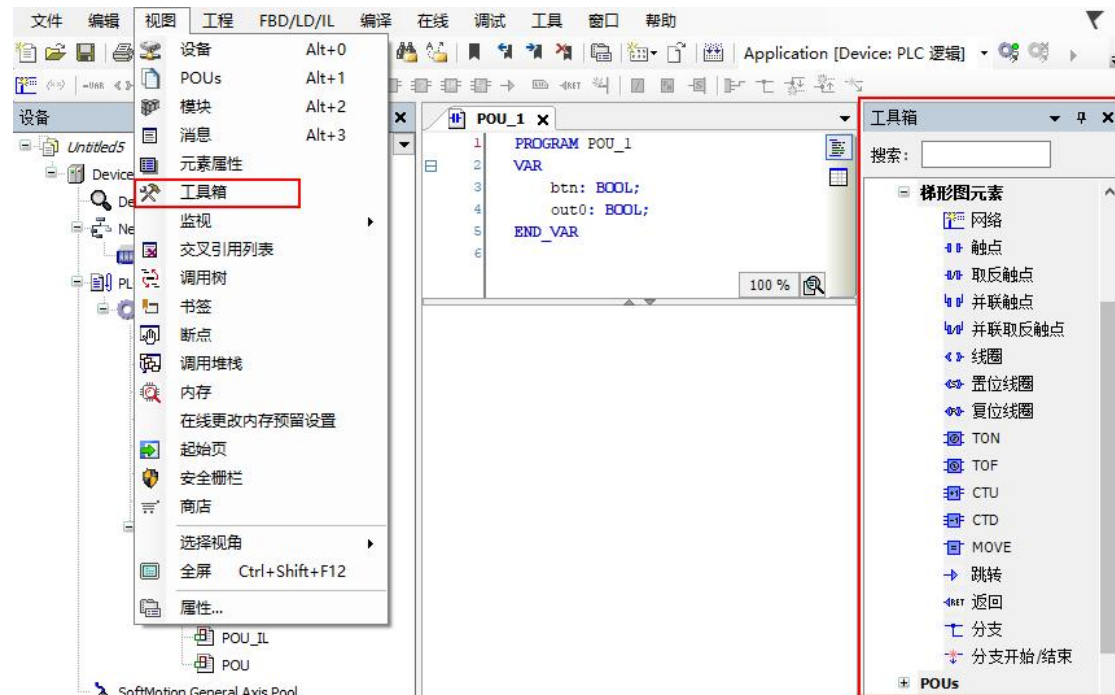


图 4.19 LD 编辑器的工具箱

在 LD 编辑器中插入一个元素，要在工具箱中鼠标左击选择它，然后拖动到编辑器窗口。当按住鼠标拖动元素到编辑器窗口中时，在可以插入的位置以菱形符号指示。最近的插入位置会变成绿色，当放开鼠标时，则这个元素将插入到绿色位置。如图 4.20 所示。

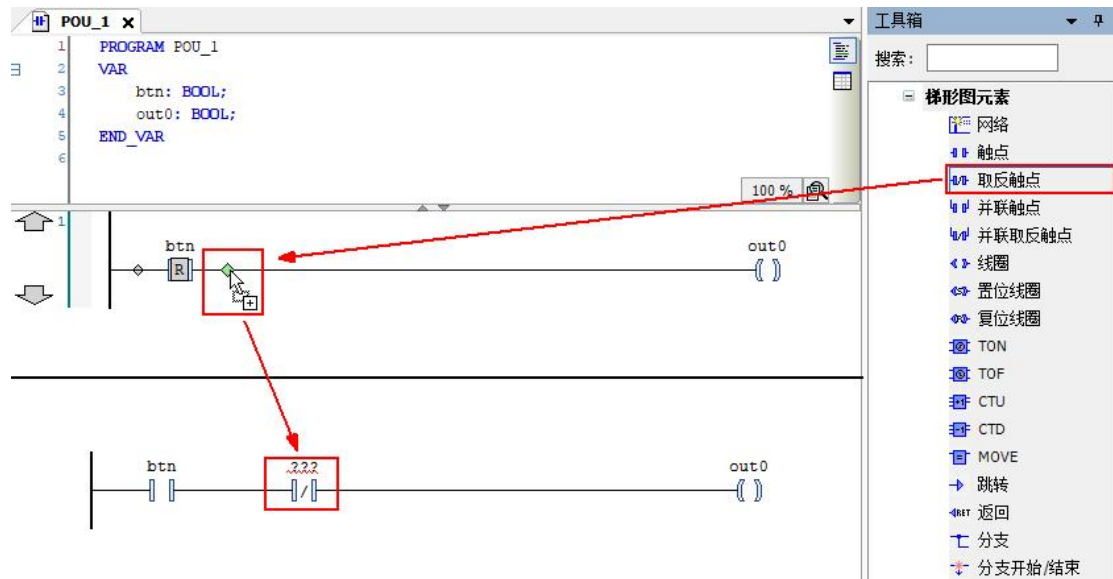


图 4.20 使用工具箱编写 LD

第五章 任务配置

每个 PLC 应用程序（工程）的工程目录树中都包含“任务配置”。在任务配置中，可定义一个或多个任务来控制和执行控制器中的应用程序。

“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。一个任务可调用一个或多个程序块(POU)。

通过设置任务的优先级和条件，可以定义处理任务的顺序。用户还可以为每个任务配置看门狗，当任务的执行周期过长时控制器提示异常。

5.1.添加任务

新建工程后，LeadSys 会自动添加“任务配置”；PMC600、PMC610 自动生成的任务如图 5.1 所示；LC 系列 PLC 会自动生成如图 5.2 所示的任务。

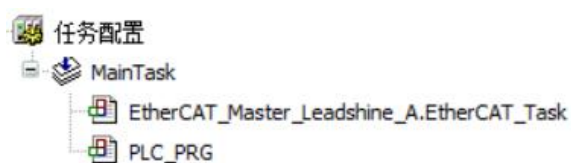


图 5.1 PMC600 的任务配置



图 5.2 LC 系列 PLC 的任务配置

用户如需添加新的任务，可以鼠标右击“任务配置”，选择“添加对象”→“任务”，如图 5.3 所示；在弹出界面中输入自定义的任务名称后，点击“打开”即可完成任务添加，如图 5.4 所示。

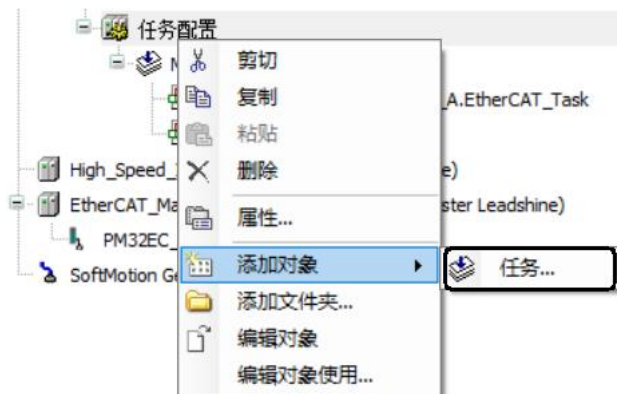


图 5.3 添加新的任务

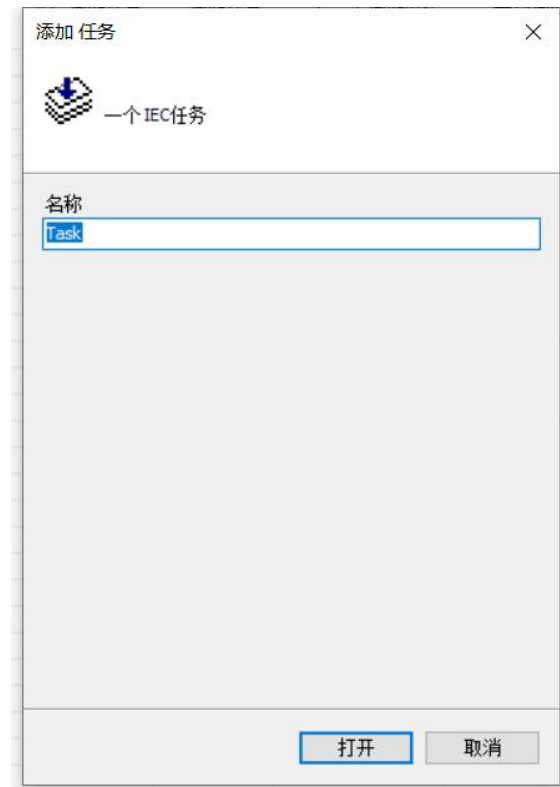


图 5.4 设置新任务的名称

5.2.任务设置

打开新建的任务后，会显示如图 5.5 所示的设置界面。

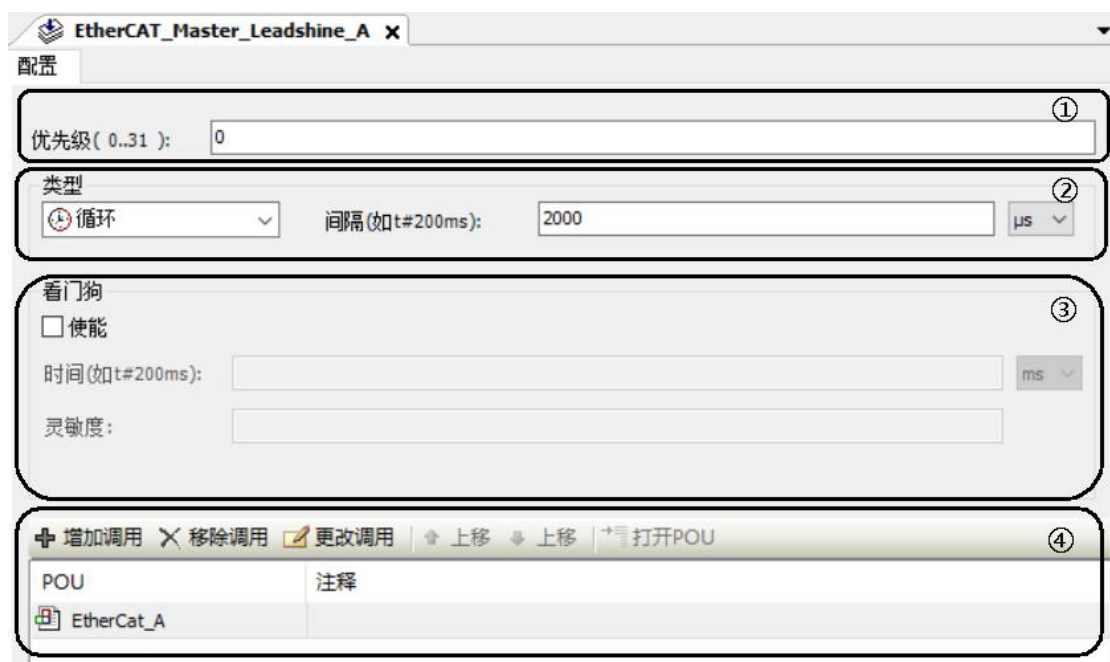


图 5.5 新任务的设置界面

图 5.5 中的标号解释如下：

- ① 优先级：可输入 0..31。数值越大优先级越低，0 级为最高优先级。
- ② 类型：任务类型及其执行逻辑。详见表 5.1。

表 5.1 任务的类型

类型	执行逻辑	设置项
循环 	按照设定的间隔循环执行	时间间隔
事件 	设置的全局变量触发上升沿后，任务开始执行	触发变量
惯性运行 	在程序开始时和完整过程结束时以连续循环的方式自动开始处理任务	/
状态 	设置的全局变量为 True 时，任务开始执行	触发变量

- ③ 看门狗：如果任务超过了看门狗的当前设置时间，则该任务将以错误状态（异常）暂停。

发生错误的任务所在的应用程序及其子应用程序也将暂停。这样，受影响的应用程序的所有任务也将停止。看门狗相关的设置见表 5.2。

表 5.2 看门狗的设置

选项	功能
使能	是否使能看门狗功能
时间	设置看门狗超时时间
灵敏度	设置数量，即超过设定时间几倍后，触发看门狗

- ④ POU 调用：可以对任务调用的 POU 进行设置，POU 调用的顺序会影响

POU 的执行顺序。设置选项如表 5.3 所示。

表 5.3 POU 调用的设置选项

选项	功能
增加调用	添加 POU 调用
移除调用	移除当前选中的 POU 调用
更改调用	更改当前选中的 POU 调用
上移	上移当前选中的 POU
下移	下移当前选中的 POU
打开 POU	打开当前选中的 POU

5.3.任务执行

5.3.1. 任务优先级

在 LeadSys 中，一个工程可以添加多个任务，但是实际执行时，在同一时间只能执行其中一个任务，任务执行的先后受任务的优先级影响，优先级高的任务会优先执行，且可以打断优先级低的任务执行，待优先级高 d 任务执行完成后，优先级低的任务才继续执行其剩余的程序。

LeadSys 中优先级共分 32（0~31）级，0 级为最高优先级，其中 0~15 级为同步任务，高优先级任务无法被低优先级任务打断。16 级以后为异步任务。

新建工程后，任务的优先级已经默认配置为：

EtherCAT_Master_LeadShine_A 任务的优先级默认为 0，0 为最高优先级。

主程序任务 MainTask 的优先级默认为 1。

可根据程序内容修改任务优先级。

EtherCAT 总线任务主要负责 EtherCAT 通信，要求实时性很高，因此，EtherCAT 总线任务优先级配置比普通任务的优先级要高。

若使用了两个 EtherCAT 口，需要注意两个任务的优先级为 0 和 1，下载前一定要检查一下，添加第二个 EtherCAT 口，其任务优先级默认为 0，需要手动修

改为 1。

若有读写文件的任务可将任务优先级调至 16 及以后。

例如：程序中存在三个优先级不同的任务，并且具有不同的扫描周期，任务执行的时序图如图 5.6 所示。

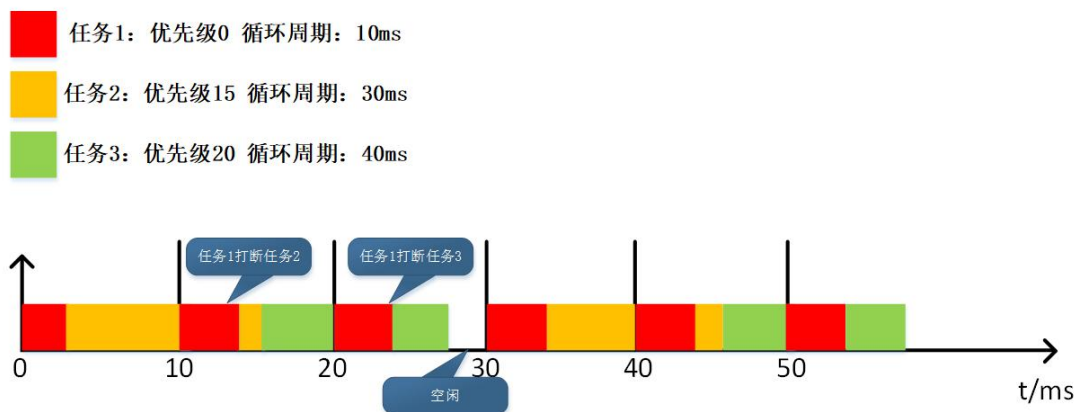


图 5.6 PLC 执行三个优先级不同的任务的时序图

0~10ms: 先执行任务 1，当任务 1 执行完成后执行任务 2；

10~20ms: 任务 1 的循环周期到达，由于任务 1 的优先级较高，故直接打断任务 2 的执行，待任务 1 执行完成后，继续执行任务 2 剩余的程序，待任务 2 也执行完成，接着执行任务 3；

20~30ms: 同上一周期，由于任务 1 的循环周期已到达，故任务 1 打断任务 3 执行，任务 1 执行完成后，接着执行任务 3 剩余程序，任务 3 执行完成后，控制器无其他任务需要执行，故处于空闲状态；

注意：

- 1) 在分配任务的优先等级时，请勿分配具相同优先级的任务；
- 2) 尽量不要在不同的任务中调用同一 POU，否则执行结果可能与预期不符；
- 3) EtherCAT 任务保持最高优先级且为最小循环周期；
- 4) 任务的循环周期不等于任务实际的执行时间；
- 5) 此处只列出任务执行的时序，实际控制器程序运行时，还包含输入扫描、输出刷新、内务处理等阶段。

5.3.2. 任务执行周期

可根据需要调整任务执行周期，如图 5.7 所示，MainTask 的默认任务周期为 4ms。

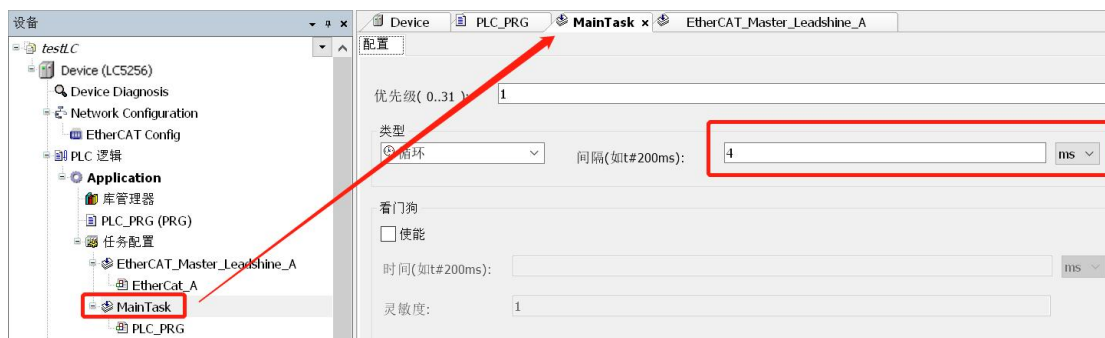


图 5.7 MainTask 的任务周期

在线模式下，可以在“任务配置”里的“监视”选项卡对任务的状态、周期、抖动等数据进行监视，如图 5.8 所示。

任务	状态	IEC-周期计算	周期计算	上一个周期(μs)	平均周期时间(μs)	最大周期时间(μs)	最小周期时间(μs)	抖动(μs)	最小抖动(μs)	最大抖动(μs)
MainTask	有效的	6429	7598	139	125	2427	11	1927	-1479	448
Task	有效的	3216	3800	21	33	37789	9	35603	-1804	33799
Task_1	有效的	3216	3800	23	37	37806	9	35640	-1822	33818

图 5.8 对任务的监视界面

任务监视窗口中参数的描述如表 5.4 所示。

表 5.4 任务监视窗口参数的描述

参数名	描述
任务	在任务配置中所定义的任务名
状态	有如下状态： 未创建：程序下载后一直未被建立，当使用事件触发任务可能会出现此状态。 创建：任务已经在实时系统中建立，但还未正式运

	行。 有效：任务正在被执行。 异常：任务出现异常。
IEC-周期计算	程序自开始运行至今的循环累积计数。
周期计算	已经运行的周期计数。取决于目标系统，它可以等于 IEC 循环计数或者更大值，此时即使应用程序没运行，周期也一样被计数。
上一个周期（us）	上一个任务周期的执行时间。
平均周期时间（us）	任务平均所需的执行时间。
最大/小周期时间（us）	任务最大/最小执行时间
抖动（us）	上个周期测量到的抖动值
最大/小抖动（us）	测量得到的最大/最小抖动时间

了解了上述各种时间的定义后，时间设定应遵循如下关系：

看门狗触发时间>固定周期循环时间>程序最大循环时间

按照此方法设定时间，可以更好的优化程序任务周期及看门狗时间，保障程序的稳定性和程序的实时性。

循环时间比固定周期循环时间长的情况下，CPU 会检测出程序有超出计数，此时，会影响程序的实时性。

如果设定的程序循环时间比看门狗时间长，CPU 会检测出看门狗故障，将停止程序执行。

5.4.POU 调用

5.4.1. 在任务配置中添加 POU 调用

在 LeadSys 中 POU 需要在任务中调用之后才能执行，添加 POU 调用有三种方式：

- 1) 通过任务设置进行添加，添加方法可参考 5.1 节内容；
- 2) 直接用鼠标拖拽 POU 到对应的任务添加。如图 5.9 所示，将“PLC_PRG”从“任务配置”上方拖至其下方。

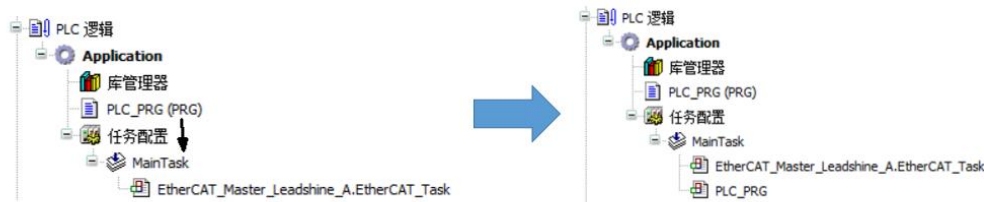


图 5.9 用鼠标拖 POU 至任务配置中

- 3) 在其他 POU 中调用（用于调用其他 POU 的 POU 需在任务中执行）
例如，在 PLC_PRG 中调用 POU_1，故 POU_1 仍会在任务 MainTask 里执行。
PLC_PRG 程序和任务配置如图 5.10 所示。

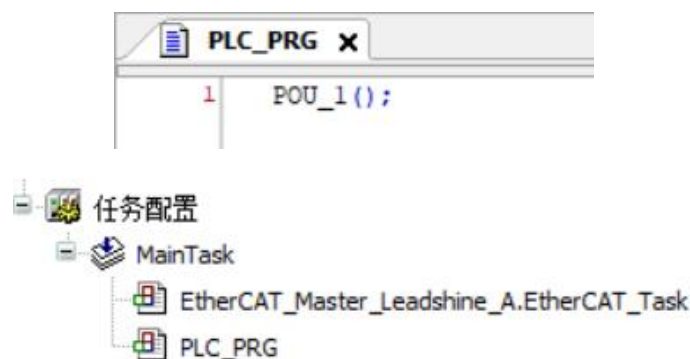


图 5.10 在 PLC_PRG 调用 POU

5.4.2. 运动控制程序的任务配置

注意：运动控制程序调用一定要与对应 EtherCAT 程序调用在同一任务下！

否则，轴使能成功后，轴运行时，会导致时序报错。如图 5.11 所示。

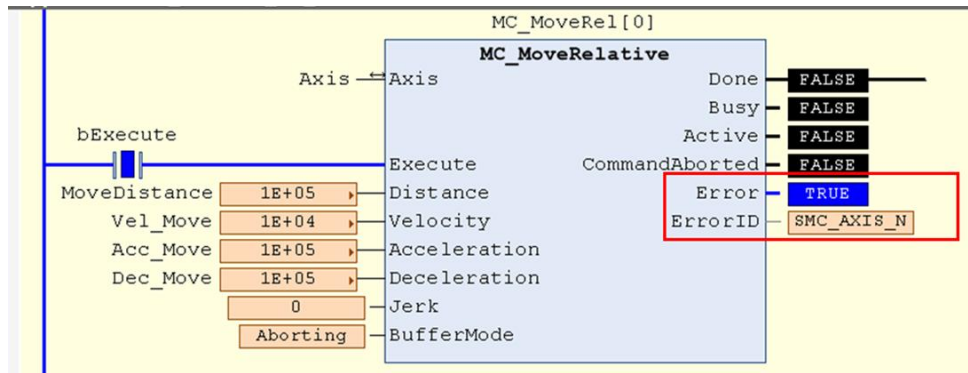


图 5.11 轴运行时报错

运动控制程序调用方法如下：

将编写 EtherCAT_A 总线所带轴的运动控制程序 PLC_PRG，在任务“EtherCAT_Master_Leadshine_A”下调用执行。如图 5.12 所示。

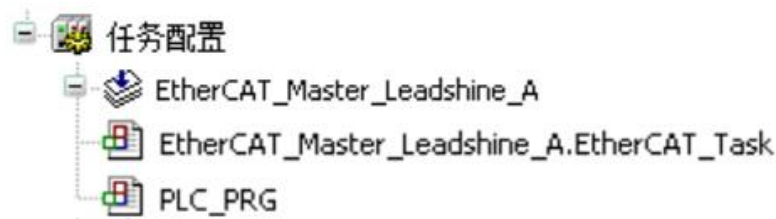


图 5.12 运动控制程序 PLC_PRG 的任务配置

通常，新建任务后，再新建 EtherCAT A 口和 EtherCAT B 口总线对应的运动控制程序 motion_A 和 motion_B，在对应的总线任务下调用。如图 5.13 所示。

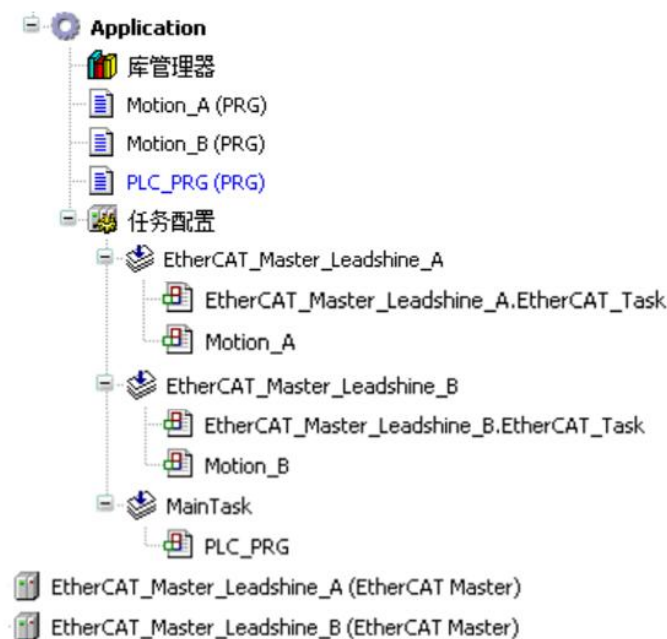


图 5.13 EtherCATA 口和 B 口的运动控制程序的任务配置

5.4.3. 判断 POU 是否被调用

如果一个 POU 没有在任何任务中调用，则编译后，该 POU 会显示为灰色，如图 5.14 所示；如果 POU 被任务调用，编译后则会显示为蓝色，如图 5.15 所示。用户可根据该方式直观的判断 POU 是否会在任务中执行。

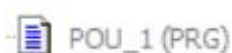


图 5.14 POU 未被任务调用



图 5.15 POU 被任务调用

5.4.4. POU 执行顺序

在任务中，POU 调用的顺序会影响 POU 的执行顺序，不同的执行顺序可能导致不同的结果。

故在调用 POU 时，顺序也是用户需要注意的一个要点，下面举例说明：

定义一个 Bool 型全局变量 ABC，并新建两个 POU，分别命名为 POU_1 和 POU_2。

POU_1 中的代码为：ABC := False;

POU_2 中的代码为：ABC := True;

情况 1：任务配置为图 5.16 所示的顺序。在此情况下，每周期结束时，ABC 的值为 True;

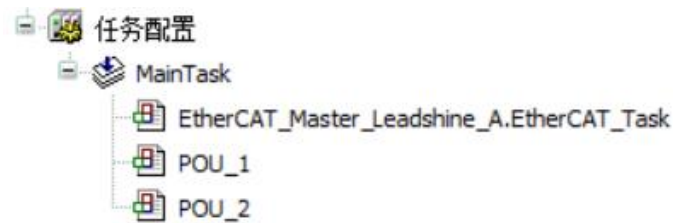


图 5.16 情况 1 的任务顺序

情况 2：任务配置为图 5.17 所示的顺序。在此情况下，每周期结束时，ABC 的值为 False。

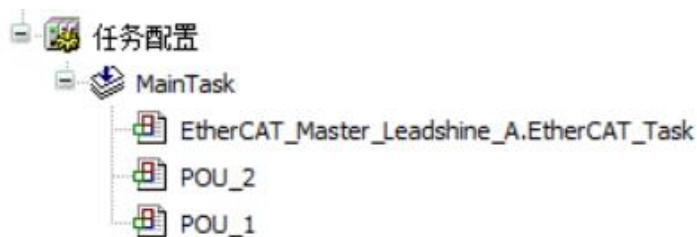


图 5.17 情况 2 的任务顺序

关于修改 POU 调用排序的方法，可参考本手册 5.2 节。

第六章 运动控制功能

6.1. 概述

6.1.1. 运动控制基本框架

LC/PMC 系列控制器的运动控制框架如图 6.1 所示。

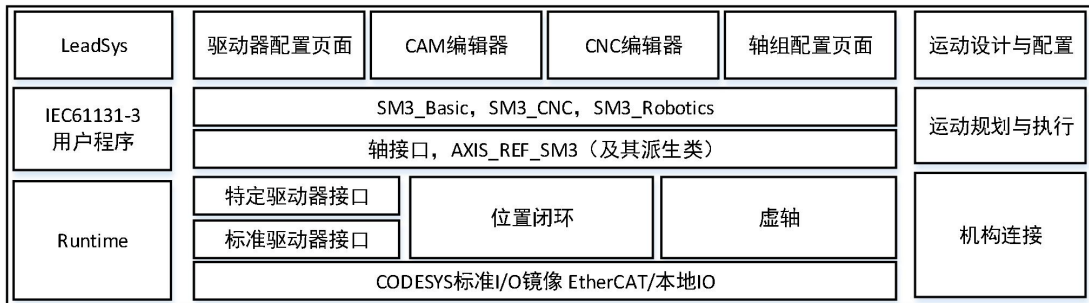


图 6.1 LC/PMC 系列控制器的运动控制框架

其中：

SM3_Basic: 是所有 SoftMotion 应用程序的基本库，符合 PLCopen 标准第 1、2 部分，可以控制单轴运动或两个轴（电子凸轮、电子齿轮）的主动/从动运动。

SM3_CNC: 是基于 SM3_Basic 的 CNC 库。除了用于运动学变换的功能之外，它还提供了轨迹生成，路径预处理和执行运动所需的所有 POU。

SM3_Robotics: 是包含符合 PLCopen 第 4 部分有关机器人技术的功能块。

6.1.2. 运动控制过程

在运动控制系统中，将运动控制的对象--电机称为轴。轴是连接电机驱动器和控制器指令间的桥梁。LC/PMC 的运动控制轴用于控制符合 CIA402 协议的 EtherCAT 总线驱动器，也可以控制本地高速脉冲输出轴。其控制过程如图 6.2 所示。

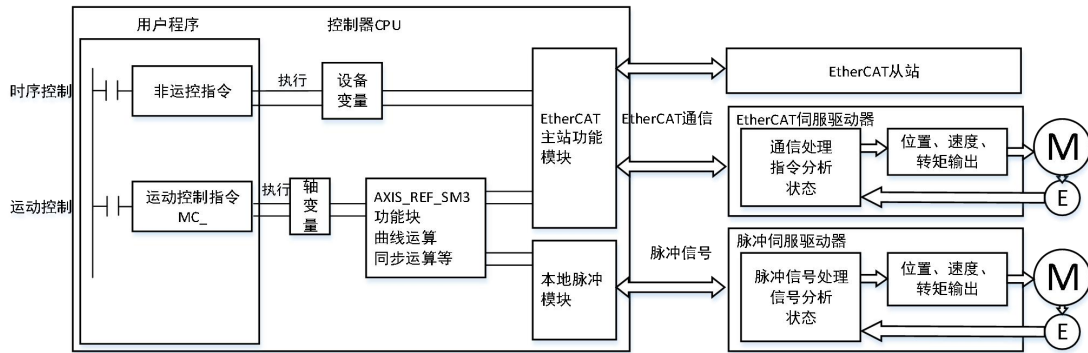


图 6.2 LC/PMC 系列控制器的运动控制过程

LC/PMC 系列控制器的运动控制过程如下：

- 1) 在用户程序中启动运动控制指令，根据相应的 PLCopen 运动库和雷赛运动库中的运动算法进行分析、计算。
- 2) 运动算法将按照循环周期的方式执行运动计算，生成的运动信息即指令值发送至总线伺服驱动器。生成的运动信息包括目标位置、目标速度、目标转矩。
- 3) 生成的指令值，将按照 EtherCAT 通信的过程数据通信周期通过 PDO 通信发送给驱动器。
- 4) 伺服驱动器根据在各周期接收到的指令值，执行对电机的位置、速度、转矩的控制。
- 5) EtherCAT 总线方式下，编码器的当前值或伺服驱动器的状态按照 EtherCAT 通信的过程数据通信周期发送回 CPU 单元。

6.1.3. PLCOPEN 状态机

LC 系列和 PMC 系列控制器基于 PLCOpen 状态机对轴的状态和运动进行管理，在每一个不同状态下完成不同的功能。轴从一个状态转移到另一个状态，需要运行对应的条件。如运行 MC 指令，或外部出现了故障，用户无法对其状态进行强制，编程时一定要按照逻辑要求，运行相关的指令。轴状态转移图如 6.3 所示。

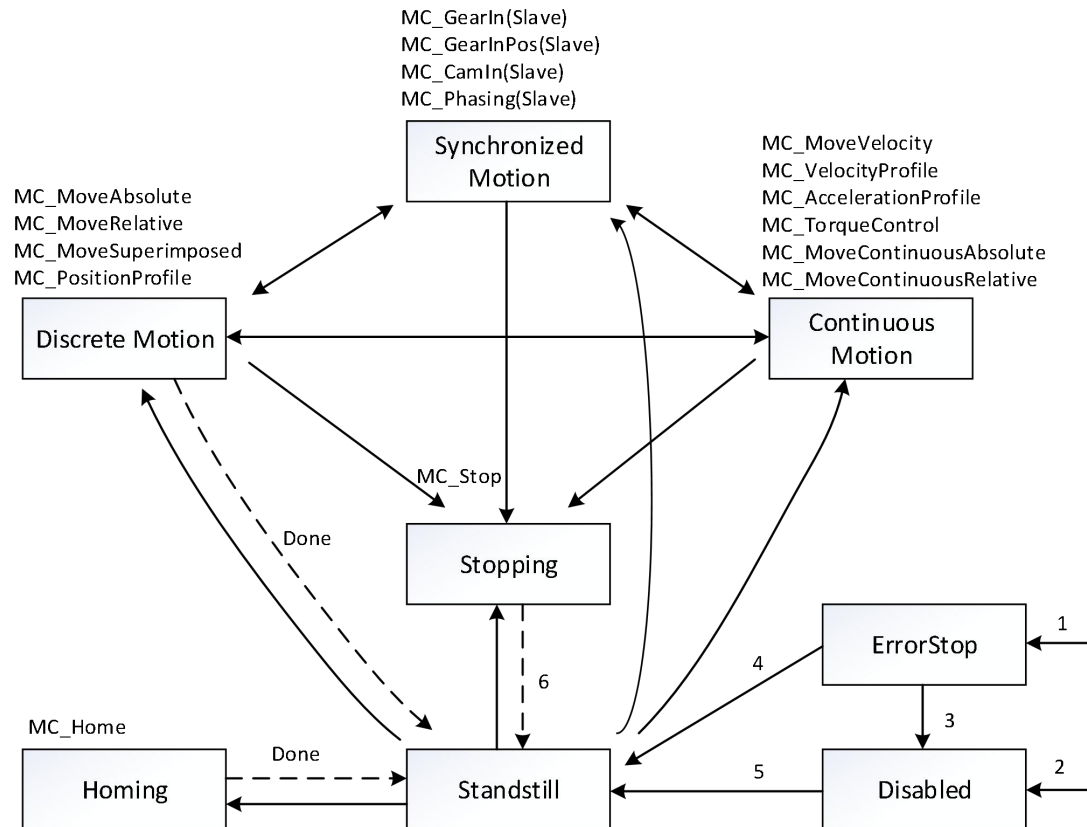


图 6.3 LC/PMC 系列控制器的轴状态转移图

如图 6.3 所示，轴的当前运行状态有 6 种。各状态间的转移条件见表 6.1。轴数据结构变量 (Axis.nAxisState) 为枚举变量，用来读取轴的当前运行状态，详见表 6.2。

表 6.1 轴状态的转移条件

序号	转移条件
1	检测到轴故障时立即进入该状态
2	轴无故障且 MC_Power.Enable=FALSE 时，进入该状态
3	调用 MC_Reset 复位轴故障且 MC_Power.Status=FALSE 时，进入该状态
4	调用 MC_Reset 复位轴故障且 MC_Power.Status=TRUE， MC_Power.Enable=TRUE 时，进入该状态
5	MC_Power.Enable=TRUE 且 MC_Power.Status=TRUE 时，进入该状态

6	MC_Stop.Done=TRUE 且 MC_Stop.Execute=FALSE 时，进入该状态
---	---

表 6.2 轴数据结构变量的取值与注释

名称	值	注释
Power_off (Disabled)	0	轴未使能，需执行 MC_Power 指令
Errorstop	1	故障停止状态，需先执行 MC_Reset/MC_Power 指令
Stopping	2	停止中，等待停机操作完成
Standstill	3	使能状态，轴已停止运行
Discrete_Motion	4	轴处于离散运行状态
Continuous_Motion	5	轴处于连续运行中
Synchronized_Motion	6	轴处于同步运行中
Homing	7	轴处于回零运行中，等待归零操作执行完成

如图 6.3 所示，MC 功能块可以使轴状态转移到指定的状态。用户应熟悉轴状态图的转移条件，并在编程时，正确调用相应的 MC 指令，才能使轴正确运行。在用户程序中，有时需要根据轴的状态，启动后续的控制逻辑。此时判断 Axis.nAxisState 的状态，比判断 MC 功能块的 done 信号更为准确、可靠。

6.1.4. 控制模式介绍

在自动控制系统中，应用程序之间通常有两种数据通讯形式：时间关键型和非时间关键型。

时间关键型通讯表示特定动作必须在指定时间窗口内完成，如果不能在指定时间窗口内完成通信，则有可能导致控制失效。周期性发送时间关键数据的过程称为

周期性过程数据通信（PDO），非时间关键数据可以非周期性发送，在 EtherCAT 中采用 MailBox 服务数据通信（SDO）。

LC/PMC 系列控制器作为 EtherCAT 主站控制符合 CIA402 协议的 EtherCAT 总线驱动器时，交互的 EtherCAT 数据正是通过过程数据对象（PDO）和服务数据对象（SDO）实现的。

控制模式由于不频繁切换，一般建议通过 SDO 配置，主站通过对象 6060h（Modes of operation）设定驱动器的控制模式，而从站通过对象 6061h（Modes of operation display）反馈当前设置的模式是否被驱动器所执行。对象 6060h 和 6061h 两者的定义是完全一致的，常用控制模式主要有以下 7 种，如表 6.3 所示。

表 6.3 常用控制模式及其 6060/6061 的值

英文名称	中文名称	6060/6061 的值
Profile position mode(PP)	轮廓位置模式	1
Profile velocity mode(PV)	轮廓速度模式	3
Profile Torque mode(PT)	轮廓转矩模式	4
Homing mode(HM)	原点回归模式	6
Cyclic synchronous position mode(CSP)	周期同步位置模式	8
Cyclic synchronous velocity mode(CSV)	周期同步速度模式	9
Cyclic synchronous torque mode(CST)	周期同步转矩模式	10

表 6.3 中的各运动模式说明如下：

- 轮廓位置模式(PP)

此模式主要用于点到点的定位运动。此模式下，LC/MC 系列控制器给定目标位置(绝对坐标值或相对坐标值)、位移过程的速度、加速度及减速度，伺服电机驱动器内部的轨迹发生器将根据设置参数生成目标位移曲线指令，驱动器内部完成位置控制、速度控制和转矩控制。基本 PDO 配置如表 6.4 所示。

● 轮廓速度模式(PV)

此模式下，LC/MC 系列控制器将目标速度、加速度、减速度发送给伺服电机驱动器，驱动器自己规划速度指令曲线，速度、转矩调节由驱动器内部执行。其基本 PDO 配置如表 6.5 所示。

表 6.4 PP 模式的 PDO 配置表

RPDO	TPDO	备注
Control word (控制字)：6040-00	Status word (状态字)：6041-00	必选
Target position (目标位置)：607A-00	Position actual value (位置反馈)：6064-00	必选
profile velocity (轮廓运行速度)：6081-00	-	必选
profile acceleration (轮廓加速度)：6083-00	-	可选
profile deceleration (轮廓减速度)：6084-00	-	可选
Modes of operation (模式选择)：6060-00	Modes of operation display (运行模式显示)：6061-00	可选

表 6.5 PV 模式的 PDO 配置表

RPDO	TPDO	备注
Control word (控制字)：6040-00	Status word (状态字)：6041-00	必选
Target velocity	-	必选

(目标速度) : 60FF-00		
-	Position actual value (位置反馈) : 6064-00	可选
-	Velocity actual value (实际速度) : 606C-00	可选
profile acceleration (轮廓加速度) : 6083-00	-	可选
profile deceleration (轮廓减速度) : 6084-00	-	可选
Modes of operation (模式选择) : 6060-00	Modes of operation display (运行模式显示) : 6061-00	可选

● 轮廓转矩模式(PT)

在此模式下 LC/MC 系列控制器将目标转矩 6071h、转矩斜坡常数 6087h 发送给伺服电机驱动器，驱动器自己规划速度指令曲线，转矩调节由驱动器内部执行。基本 PDO 配置如表 6.6 所示。

● 原点回归模式 (HM)

原点回零模式用于寻找机械原点，并定位机械原点与机械零点的位置关系。一般来说，原点运动只在增量式编码器电机时需要。原点运动完成后，将该点作为机械原点，在此基础上进行其他模式的运动。

通过设置 607Ch，可以设定机械原点与机械零点的关系：

$$\text{机械原点} = \text{机械零点} + 607\text{Ch}(\text{原点偏置})$$

原点回零模式的基本 PDO 配置如表 6.7 所示。

表 6.6 PT 模式的 PDO 配置表

RPDO	TPDO	备注
Control word (控制字) : 6040-00	Status word (状态字) : 6041-00	必选
Target Torque (目标转矩) : 6071-00	-	必选
Torque slope (转矩斜坡) : 6087-00	-	可选
-	Position actual value (位置反馈) : 6064-00	可选
-	Velocity actual value (实际速度) : 606C-00	可选
-	Torque actualvalue (实际转矩) : 6077-00	可选
Modes of operation (模式选择) : 6060-00	Modes of operation display (运行模式显示) : 6061-00	可选

表 6.7 HM 模式的 PDO 配置表

RPDO	TPDO	备注
Control word (控制字) : 6040-00	Status word (状态字) : 6041-00	必选
Homing method (回零方式) : 6098-00	-	必选
Home offset (原点偏置) : 607C-00	-	可选
Speed during search for switch	-	可选

(搜索减速点信号速度) : 6099-01		
Speed during search for zero (搜索原点信号速度) : 6099-02	-	可选
Homing acceleration (回零加速度) : 609A-00	-	可选
-	Position actual value (位置反馈) : 6064-00	可选
Modes of operation (模式选择) : 6060-00	Modes of operation display (运行模式显示) : 6061-00	可选

● 周期同步位置模式(CSP)

周期同步位置模式下，LC/MC 系列控制器完成位置指令规划，然后将规划好的目标位置 607Ah 周期性同步的发送给伺服电机驱动器。在此运行模式下，主站全程负责伺服电机每一个通信周期的运行位置的规划计算，而伺服电机只知道下一个 EtherCAT 时刻需要到达的目标点。其基本 PDO 配置如表 6.8 所示。

表 6.8 CSP 模式的 PDO 配置表

RPDO	TPDO	备注
Control word (控制字) : 6040-00	Status word (状态字) : 6041-00	必选
Target position (目标位置) : 607A-00	Position actual value (位置反馈) : 6064-00	必选
Modes of operation (模式选择) : 6060-00	Modes of operation display (运行模式显示) : 6061-00	可选

● 周期同步速度模式(CSV)

在周期同步速度模式下，LC/MC 系列控制器完成速度指令规划，然后将规划好的目标速度 60FFh 周期性同步的发送给伺服电机驱动器，速度、转矩调节由驱动器内部执行。其基本 PDO 配置如表 6.9 所示。

表 6.9 CSV 模式的 PDO 配置表

RPDO	TPDO	备注
Control word	Status word	必选
Target velocity	-	必选
-	Position actual value	可选
-	Velocity actual value	可选
Modes of operation	Modes of operation display	可选

● 周期同步转矩模式(CST)

在周期同步转矩模式下，LC/MC 系列控制器完成转矩指令规划，将规划好的目标转矩 6071h 周期性同步的发送给伺服电机驱动器，转矩调节由驱动器内部执行。当速度达到限幅值后将进入调速阶段。其基本 PDO 配置如表 6.10 所示。

表 6.10 CSV 模式的 PDO 配置表

RPDO	TPDO	备注
Control word	Status word	必选
Target Torque	-	必选
-	Position actual value	可选
-	Velocity actual value	可选
	Torque actualvalue	可选
Modes of operation	Modes of operation display	可选

6.2.轴组态

1) 本地脉冲轴组态

PMC600 系列控制器支持本地 4 路脉冲轴，如果需要使用本地轴相关功能，则需要的高速 IO 界面组态配置。

双击设备导航栏窗口上的“High_Speed_IO_Module”，进入 IO 设置界面；点击“硬件端口配置 1”栏，显示界面如图 6.6 所示。点击轴 0 前的复选框，出现符号“√”，即选中轴 0，输出口 Out00、Out01 自动分配为电机的脉冲和方向信号；IN00、IN01、IN02 自动分配为原点信号、负限位和正限位信号。轴 1~3 的原点限位信号依次往后排序。

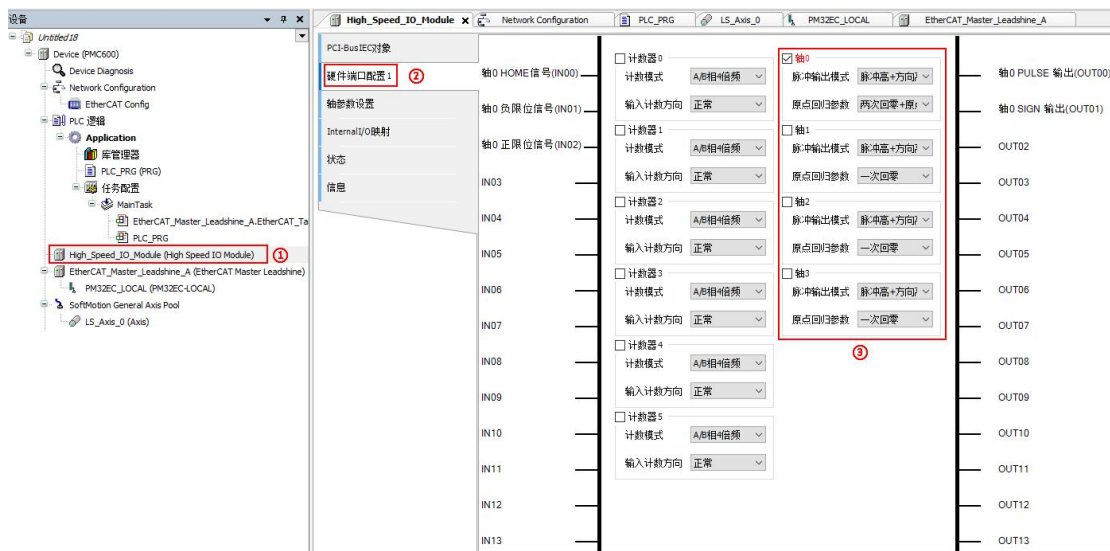


图 6.4 脉冲轴控制信号的设置

2) EtherCAT 轴组态

EtherCAT 轴使用之前必须组态总线网络配置。LC 系列控制器支持双 EtherCAT 主站，PMC 系列控制器仅支持单 EtherCAT 主站，下面以 LC5256 组态 EtherCAT 轴为例，其它控制器操作类似。

如图 6.5 所示，鼠标双击设备导航栏窗口上的①通讯配置对象，左键单击② EtherCAT 主站（用户因根据是否需要双主站进行勾选），在右侧网络设备导航栏窗口上双击③总线伺服电机驱动器，完成 EtherCAT 轴的组态。

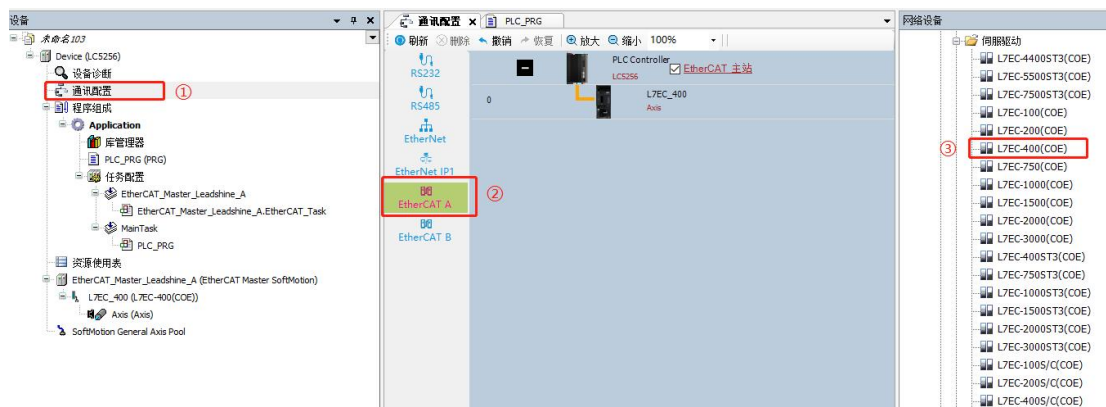


图 6.5 EtherCAT 轴的组态方法一

用户也可以通过鼠标左键拖拽实现总线轴组态,选中网络设备导航栏窗口上的总线伺服驱动器,鼠标左键按住不放,拖拽到左侧网络组态窗口中,鼠标显示+号时释放鼠标左键,总线轴即添加完成,如图 6.6 所示,左键拖拽组态 EtherCAT 轴。

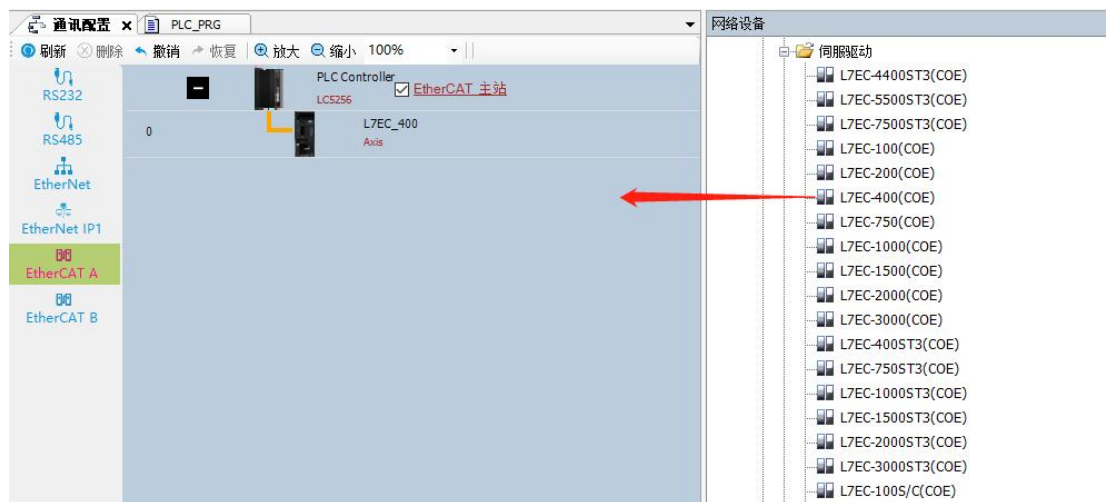


图 6.6 EtherCAT 轴的组态方法二

6.3.本地脉冲轴设置

6.3.1. 基本参数设置

PMC600 系列控制器组态界面设置了本地脉冲轴后,还需要设置轴的运动参数。

点击“轴参数设置”栏，界面如图 6.7 所示，设置轴的基本参数。

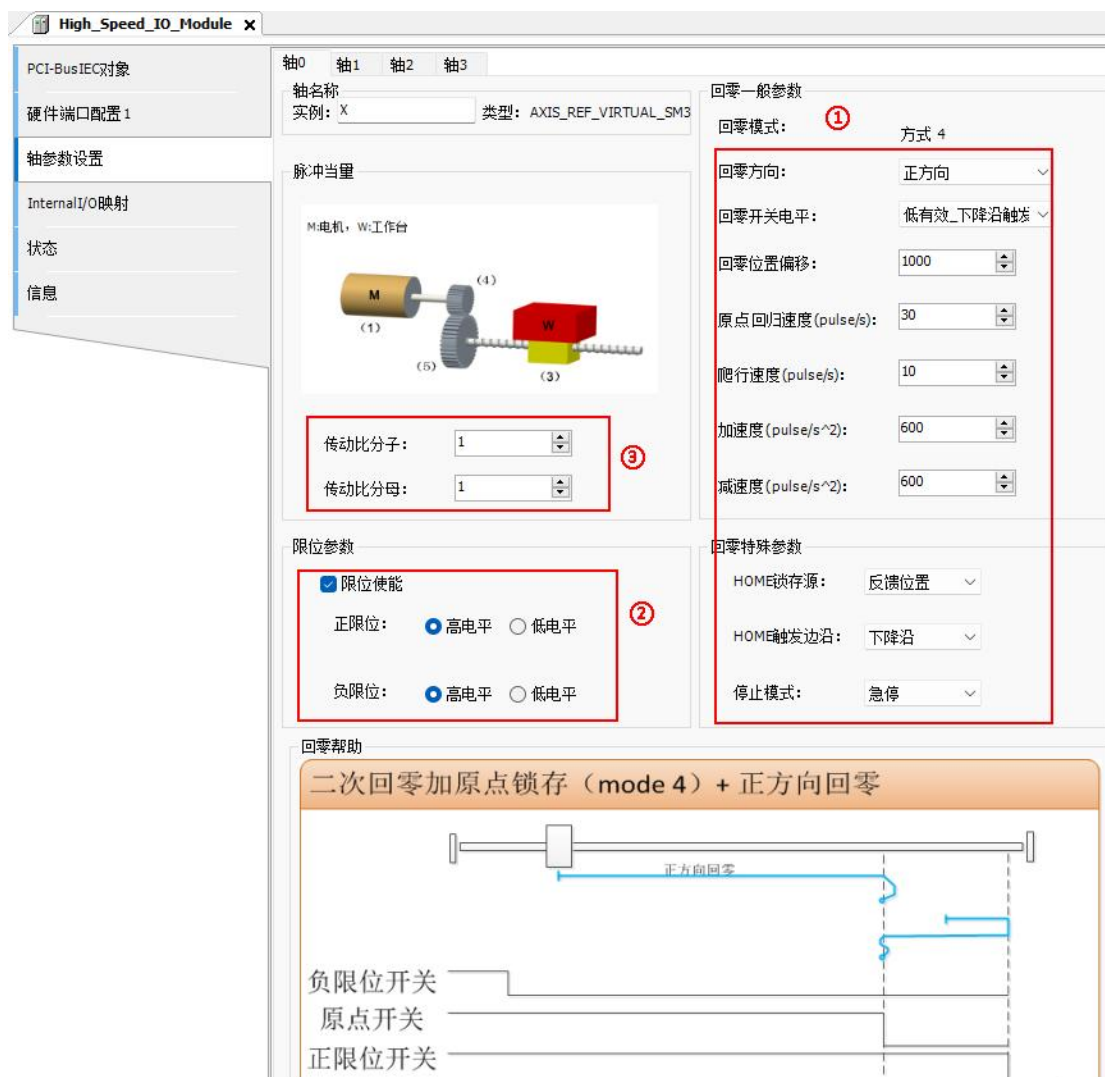


图 6.7 脉冲轴的基本参数

图中标注为：

- ①回零速度参数及锁存回零模数设置。
- ②轴正负限位使能及有效电平配置。
- ③用户单位与脉冲单位的转换设置。

例如：以 mm 为用户单位，轴的传动机构为滚珠丝杠，导程为 5mm，即电机一周平台移动 5mm；伺服电机每转脉冲数为 10000。则分母为 1，分子为 2000。

注意：设置好脉冲当量后，所有运动指令中的位移、速度和加速度等参数中的位移量必须使用用户单位。

6.3.2. 运动程序编写

完成上述参数设置后，可开始编写运动控制程序，主要步骤有：

1. 初始化脉冲轴

首先在程序中调用子程序 LS_PulseAxis_MachineState，初始化脉冲轴。该子程序要求设置 4 个轴，用户如果只使用一个脉冲轴，为了初始化该模块，还需要再添加 3 个虚拟轴。

虚拟轴添加方法如图 6.8 所示，鼠标右键点击目录树中的“SoftMotionGeneralAxisPool”，选择“添加设备”。单击“虚拟驱动器”下的“Axis”，再点击“添加设备”按钮，即添加了一个虚拟轴。然后按此方法再添加 2 个虚拟轴。

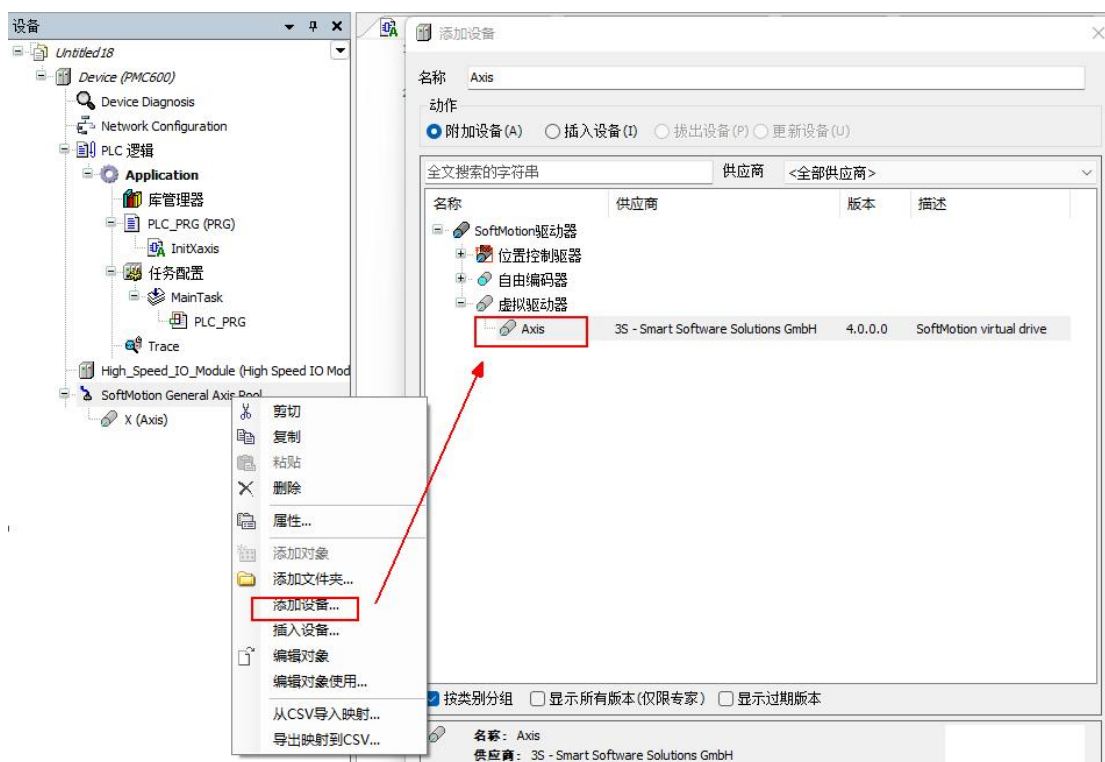


图 6.8 添加虚拟轴

在工程目录树中，用鼠标右键单击“PLC_PRG”，选“添加对象”→“动作”（所谓“动作”，可以理解为子程序），如图 6.9 所示。给新动作一个名称，不妨设为：InitXaxis，选择实现语言为 FBD。

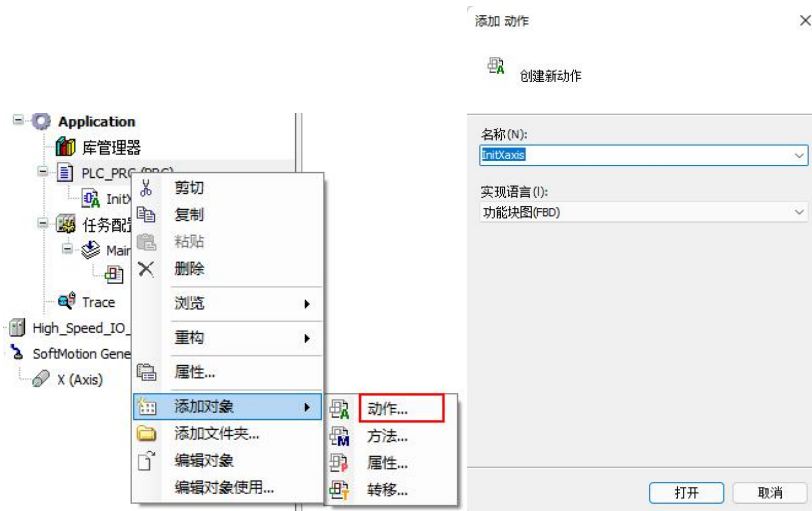


图 6.9 添加一个动作

在功能块编辑区里点击鼠标右键，选“插入运算块”。在输入助手中的“文本搜索”栏里，输入“LS_PulseAxis_MachineState”，在搜索结果里选中该子程序，然后点击“确定”，如图 6.10 所示。

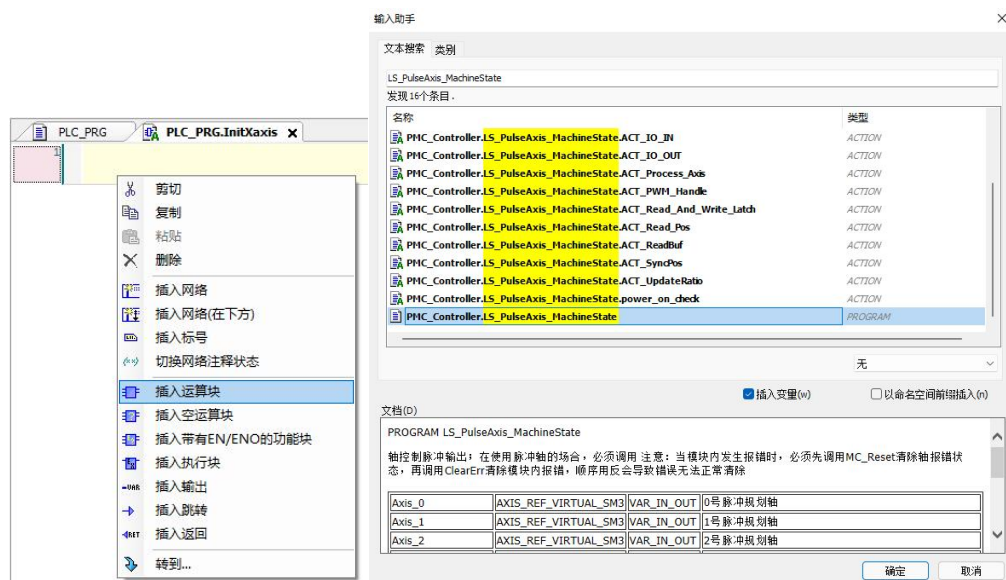


图 6.10 插入运算块

设置 LS_PulseAxis_MachineState 的参数，如图 6.11 所示。有问号处，用退格键清除。LS_PulseAxis_MachineState 是程序不是功能块，故不需要实例化。

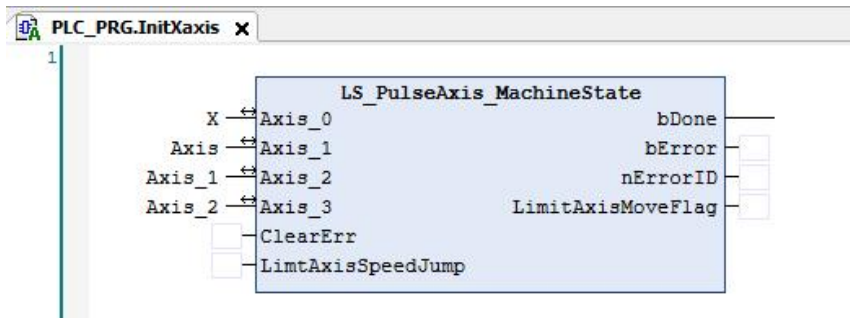


图 6.11 设置 LS_PulseAxis_MachineState 的参数

2. 调用轴使能模块

通过“插入网络”添加新的网络，在新网络中添加 MC_POWER 功能块并自动声明名称 MC_POWER_0。如图 6.12 所示。

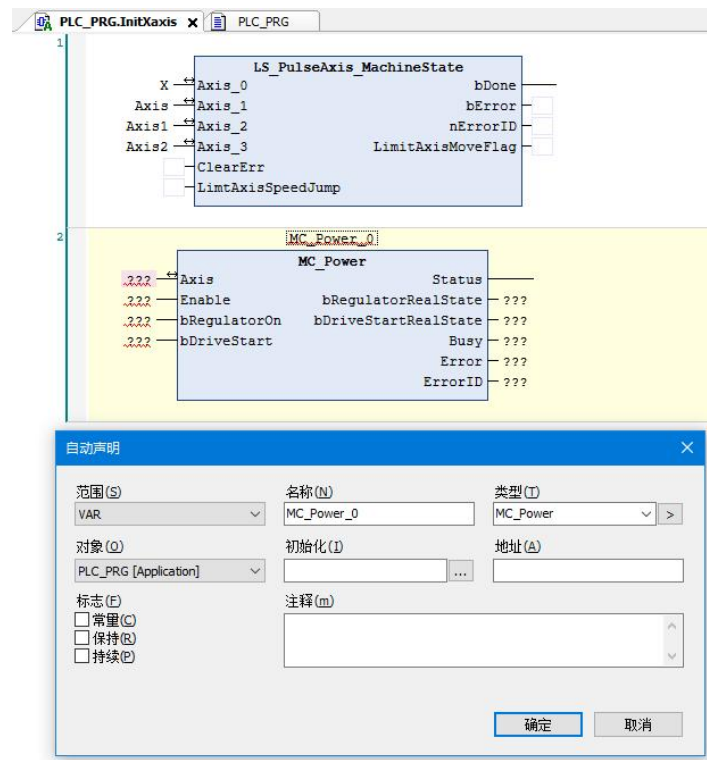


图 6.12 调用轴使能模块 MC_POWER

完成后的“动作” InitXaxis 程序如图 6.13 所示。

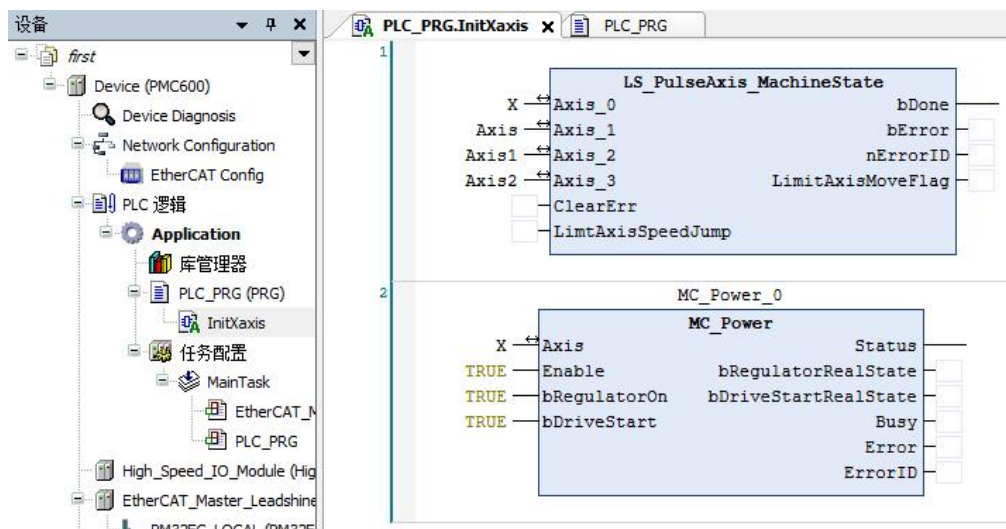


图 6.13 轴初始化与使能的子程序

3. 调用运动控制指令

以调用点位运动指令为例，在 PLC_PRG 编辑区点击右键，选“输入助手”。在“文本搜索”栏里，输入“MC_MoveRelative”，在搜索结果里选中该模块，点击确认，如图 6.14 所示。

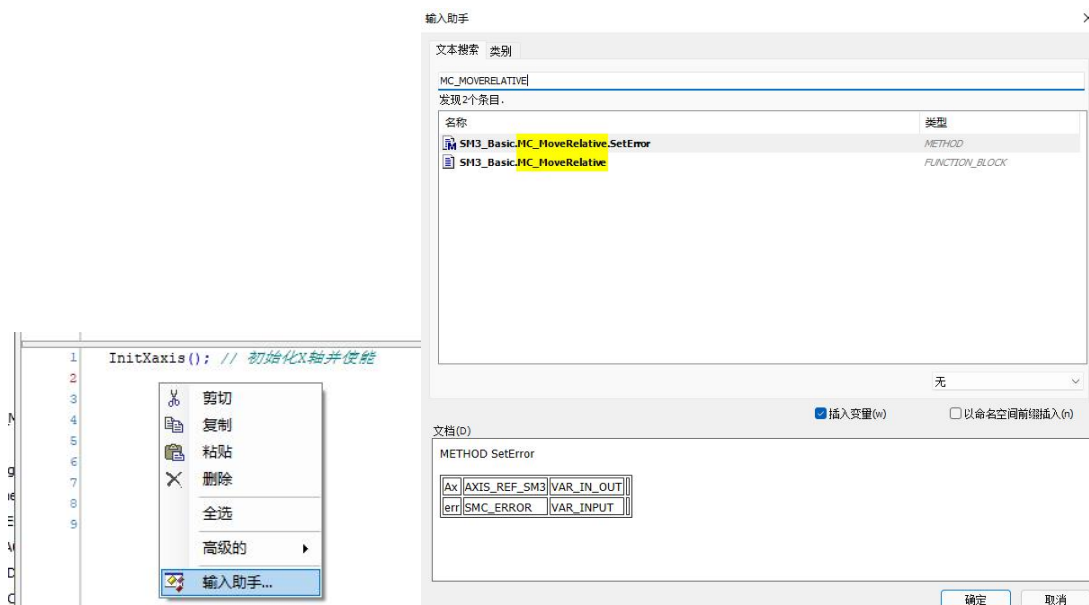


图 6.14 调用相对坐标的点位运动指令

然后声明 MC_MoveRelative 类型的功能块为 MoveX，如图 6.15 所示。设置相对点位运动模块的参数并编译程序，如图 6.16 所示。

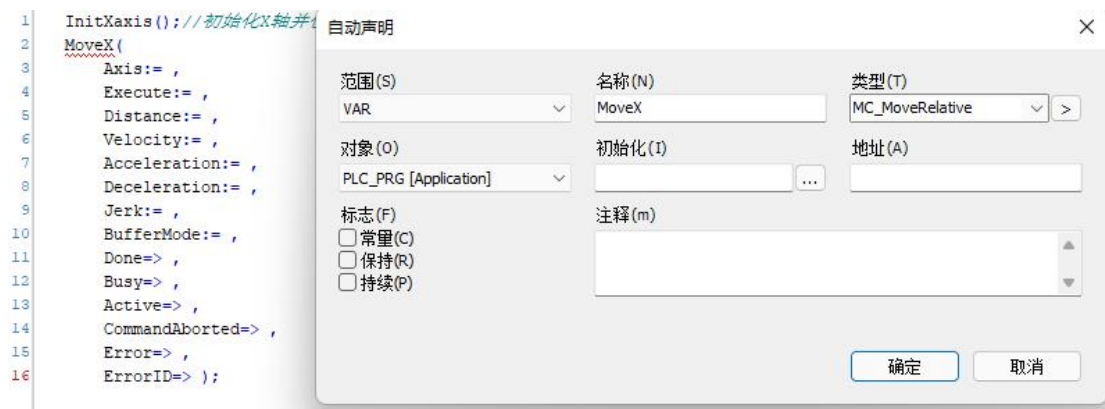


图 6.15 声明 MC MoveRelative



图 6.16 设置相对点位运动模块的参数并编译程序

4. 登录、下载程序

完成上述操作后，登录控制器，选择“登录并下载”，如图 6.17 所示，再运行程序。

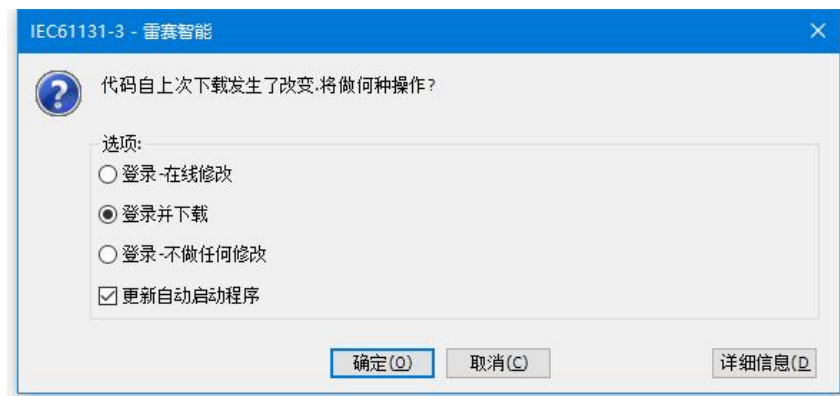


图 6.17 登录并下载程序

鼠标左键双击点位运动功能块 MoveX 中的 Execute 后面的变量 Xstart，使其准备值为 TRUE，如图 6.18 所示；然后按 Ctrl+F7 键，写入 TRUE。这时，电机开始转动。

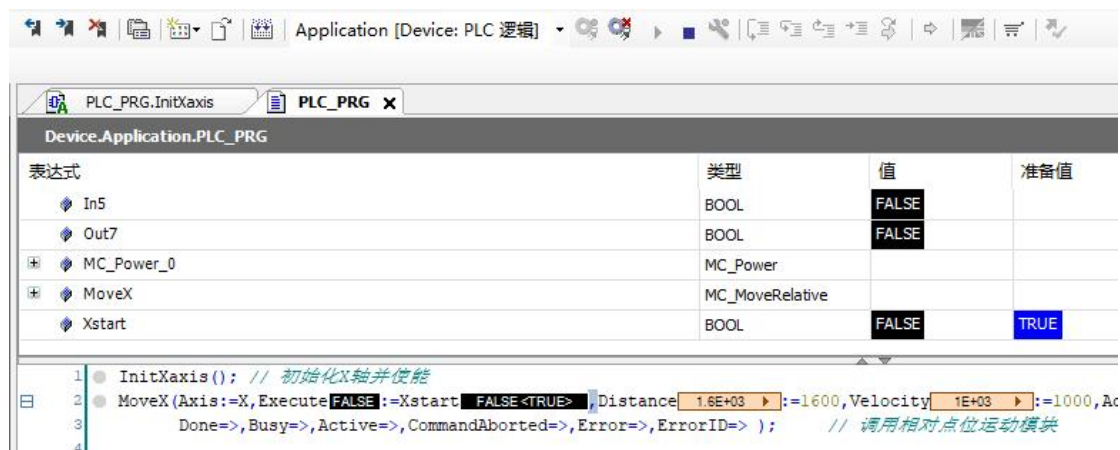


图 6.18 将变量 Xstart 设为 TRUE

点击项目目录树中的 X (Axis)，打开 X 轴的参数显示界面，监视脉冲轴运动状态，如图 6.19 所示，轴当前位移为 1600 个脉冲。

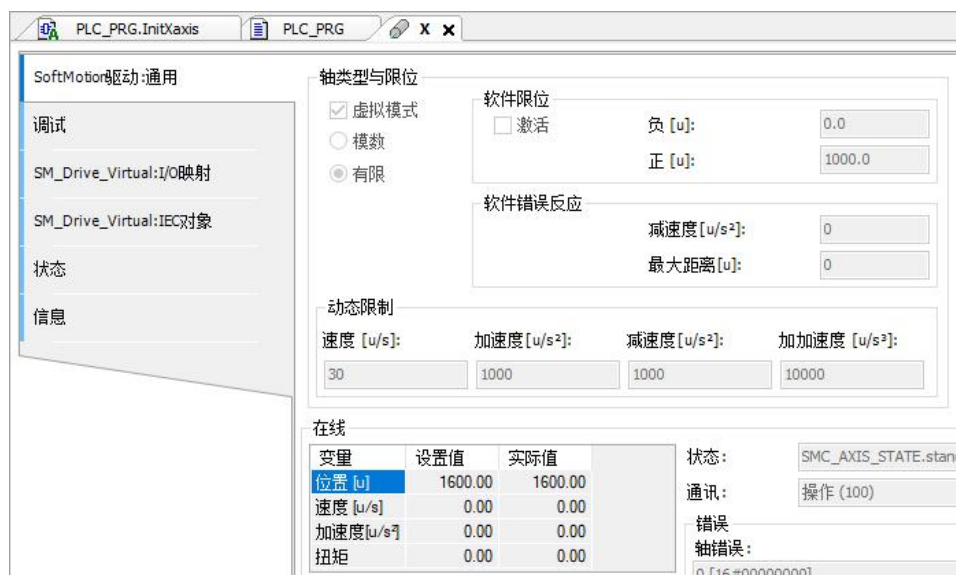


图 6.19 查看轴的实际位置

5. 回零运动

回零运动指令是运动控制程序中必不可少的指令。用户完成 6.3.1 节中的回零参数设置，再调用 MC_Home 指令即可实现回零运动。

按照添加单轴运动指令的方式，添加回零运动指令 MC_Home。鼠标左键双击回零指令 MoveHome 中的 Execute 后面的变量 XhomeSstart，使其准备值为 TRUE，然后按 Ctrl+F7 键，写入 TRUE。这时，电机开始回零运动。程序如图 6.20 所示。

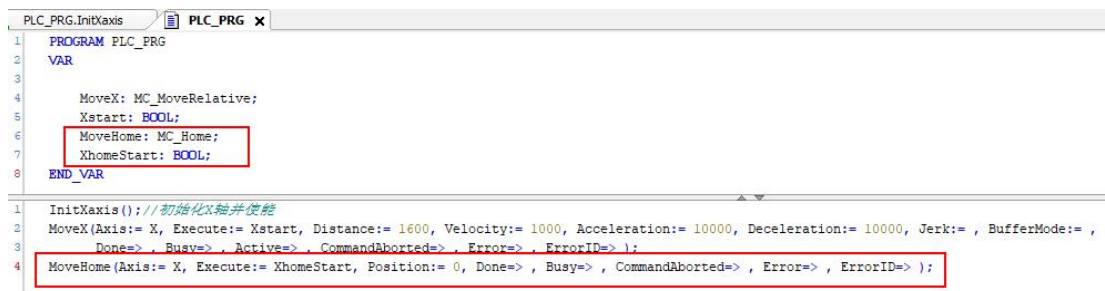


图 6.20 回零运动指令 MC_Home

6.4.EtherCAT 轴设置

如图 6.21 所示，添加 EtherCAT 伺服轴后，双击设备导航栏窗口上 CIA402 轴对象①Axis，在主窗口中会显示设置界面②。下面介绍通用参数、单位换算、回零参数的设置方法。

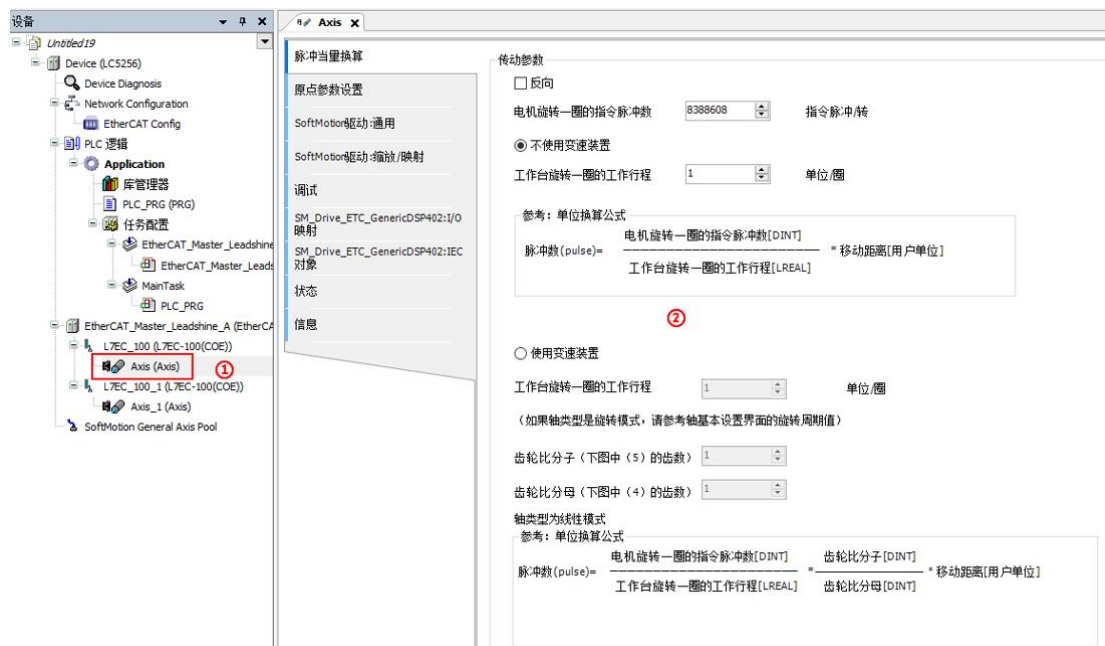


图 6.21 回零运动指令 MC_Home

6.4.1. 通用参数的设置

轴的通用参数设置界面用于设定轴的工作模式、软件限位、错误反应等参数。选择“SoftMotion 驱动：通用”选项卡，如图 6.22 所示，设置以下参数。

图 6.22 轴的通用参数

① 虚拟模式：设置虚轴、实轴两种工作模式

- 实轴的用途：用于控制实际的伺服电机，工程中的程序必须与伺服电机一起运行。有些参数必须在实轴模式才可以获取，如：伺服功能码、在线 COE。实轴模式会有外界的干扰，如在 TRACE 监控过程中，会影响到显示效果。
- 虚拟轴的用途：用于模拟运行调试，不需要实际的伺服电机。通过模拟运行，可以获得调试参数。

② 模数/有限：指定轴的类型。

- 模数：即旋转模式。电机无限旋转，不限制移动范围（皮带驱动等）。指令位置在 0~模数值 u 的区间内循环，如图 6.23 所示。模数值无法设置负值。通常在 X-Y 直角坐标系下的装置中使用。

图 6.23 模数设置

- 有限：即线性模式。指令位置的设置值为有限的值，轴位置以线性方式增加或者减少。有限模式下通常包含一个零点。运动时反馈位置增大则表示正向运动，反之为反向运动。允许设置正向软件限位和反向软件限位，当激活软件限位后轴只能在限位范围内运动。如图 6.24 所示。

☐ 虚拟模式 ☐ 模数 ☒ 有限	软件限位	
	☐ 激活	负 [u]: 0.0 正 [u]: 10000

图 6.24 选择“有限”类型

- ③ 软件限位：轴的类型为有限时，可以设置软限位。

勾选使能后，可设置正、负方向限制轴运动的范围，如图 6.25 所示。如果指令位置超出软限位的设置范围，将发生错误停止，并停止动作。当轴运动到软限位的位置时，控制器将按设置的参数迅速停止运动。停止时间可由“软件错误反应”参数中设置的减速度和最大距离计算而得；或由“动态限制”中设置的参数决定；哪种方式的停止时间短就用哪种方式减速停止。

软件限位对回零运动和力矩模式下的运动无效。

☐ 虚拟模式 ☐ 模数 ☒ 有限	软件限位	
	☒ 激活	负 [u]: 0.0 正 [u]: 10000

图 6.25 设置软限位

- ④ 软件错误反应：

设置在运动过程中发生错误时停止运动的参数：减速度和停止距离，如图 6.26 所示。该设置与软件限位关联，在软件限位使能的状态下才具有实际意义。

软件错误反应	
减速度 [u/s^2]:	0
最大距离 [u]:	0

图 6.26 软件错误反应参数

发生错误时运动停止和超过软限位时的运动停止过程中所用的减速度或减速距离,是由停止时间最短为原则,在以下 3 种参数中选用停止时间最短的那种参数。

- 软件错误反应的减速度
- 软件错误反应的最大距离
- 动态限制的参数

软件错误反应的减速度、最大距离设置为 0 时,则根据动态限制中设置的减速度停止动作。

⑤ 动态限制

主要用于有 CNC 功能轴的限制设置。速度、加速度、减速度不能设置为 0,如图 6.27 所示。设置为 0 时将产生警告。

动态限制			
速度 [u/s]:	加速度 [u/s^2]:	减速度 [u/s^2]:	加加速度 [u/s^3]:
30	1000	10000	10000

图 6.27 “动态限制”参数

可以通过"SMC_CheckLimits"功能块确认轴动作中是否超过动态限制中设置的值。此外,由于"SMC_CheckLimits"无法检测加加速度的超限,因此请勿使用加加速度栏。

⑥ 速度斜坡类型: 确定轴的速度变化曲线,如图 6.28 所示。



图 6.28 “速率斜坡类型”参数

⑦ 标识：轴的外部 ID，如图 6.29 所示。



图 6.29 轴的 ID

⑧ 位置滞后监视：设置位置滞后限制，监控轴的运行状态。

在执行定位指令和速度指令期间，伺服电机驱动器实际工作在 CSP（周期性同步位置）模式，位置曲线的规划在控制器侧完成。控制器通过 0x607A 向伺服驱动器发送目标位置，伺服驱动器驱动伺服电机运动，电机编码器的位置通过 0x6064 反馈到控制器，由于伺服驱动器和电机本身的原因，0x607A 和 0x6064 之间存在偏差值。

该差值即为位置滞后误差，如果轴的位置滞后误差的绝对值超过设置滞后限制值时，则轴报告跟随误差过大故障并进入停止状态。滞后限制值的设置如图 6.30 所示。



图 6.30 轴的滞后限制值

6.4.2. 单位换算

可选择“脉冲当量换算”选项卡，设置轴的脉冲当量。

伺服电机、步进电机转动的角度与电机驱动器内控制位移量的脉冲数成正比。而工作平台的位移量则使用毫米、英寸或角度、角分等常见的度量单位，我们称之为用户单位（Unit）。

为了能在运动指令中将用户单位作为位移单位，而不是以脉冲为位移单位，就需要设置脉冲当量：脉冲/用户单位，即平台运动 1 个用户单位，电机需要转动多少脉冲数。

在计算脉冲当量时，必需知道电机转动一圈的脉冲数，同时还需知道传动机构的参数，如：齿轮的减速比、丝杆导程、同步带节圆周长。

通过配置脉冲当量的相关参数，如图 6.31 所示，轴内部将自动完成脉冲数与用户单位之间的转换。图 6.31 中各参数设置方法如下。

图 6.31 脉冲当量换算界面

① 反向：勾选后，电机旋转方向与默认方向相反。

② 旋转一圈的指令脉冲数：即电机旋转一周需要的脉冲数

如：使用 23 位编码器的雷赛伺服电机，且电子齿轮比为 1：1，则电机旋转一周需要的脉冲数设为 8388608，如图 6.32 所示。

电机旋转一圈的指令脉冲数 指令脉冲/转

图 6.32 设置电机旋转一圈的指令脉冲数

③ 转换模式

工作台旋转一圈的工作行程：工作台传动机构的末端零件，如：同步带轮、齿轮、丝杆等，旋转一圈，工作平台移动的距离，以用户单位计。

如：使用导程为 10mm 的丝杆，设置如图 6.33 所示。

工作台旋转一圈的工作行程 单位/圈

图 6.33 设置工作台行程

转换模式包括两种，无变速机构和使用变速机构。

- 无变速机构

没有变速机构时，用户单位与脉冲单位的转换公式如图 6.34 所示。

参考：单位换算公式

$$\text{脉冲数 (pulse)} = \frac{\text{电机旋转一圈的指令脉冲数 [DINT]}}{\text{工作台旋转一圈的工作行程 [LREAL]}} \times \text{移动距离 [用户单位]}$$

图 6.34 无变速机构的单位换算公式

以雷赛 23 位编码器的伺服电机和 10mm 的丝杆导程为例，设定参数如下：

电机旋转一圈脉冲数 = 8388608；

电机旋转一圈，工作台的移动量 = 10 mm；

则当相对定位指令给定的目标位移为 100 时，指令为 MC_MoveRelative(...,distance:=100,...)，执行的结果是工作平台前进 100mm。驱动器控制轴实际产生的脉冲量为 83886080，电机旋转了 10 圈。

- 使用变速机构

有变速机构时，需要配置变速机构的减速比，若非整数，可以用齿轮比分子分母放大相同倍数的整数比值来表示。如图 6.35 所示的减速机齿轮比为 2:1，表示伺服电机轴每旋转 2 圈，减速机输出轴旋转 1 圈。



图 6.35 设置齿轮比

1) 有限（线性）模式下典型工作台如图 6.36 所示。

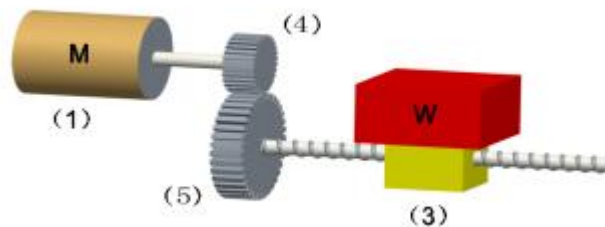


图 6.36 有限模式工作台的传动结构

图 6.36 中，(1)为伺服电机，(3)为工作平台及工件，(4)为主动齿轮（齿轮比分子），(5)为从动齿轮（齿轮比分母）及丝杆。

用户单位与脉冲单位的计算公式如图 6.37 所示。

参考：单位换算公式

$$\text{脉冲数 (pulse)} = \frac{\text{电机旋转一圈的指令脉冲数 [DINT]} \times \text{齿轮比分子 [DINT]}}{\text{工作台旋转一圈的工作行程 [LREAL]} \times \text{齿轮比分母 [DINT]}} \times \text{移动距离 [用户单位]}$$

图 6.37 有变速机构的单位换算公式

例如：采用雷赛 23 位编码器的伺服电机，丝杆导程为 10mm，减速机构的减速比为 2：1。设定的参数如下：

电机旋转一圈的脉冲数 = 8388608 脉冲；

工作台的丝杆旋转一圈的移动量 = 10 mm；

齿轮比分子= 2；

齿轮比分母= 1

若使工作平台位移 100mm，采用相对定位指令，则给定的运动距离为 100，指令为 MC_MoveRelative(...,distance:=100,...)。

执行的结果是工作平台前进 100mm。驱动器控制轴实际产生的脉冲量为 161772160，电机旋转 20 圈。

2) 模数（旋转）模式下典型工作台如图 6.38 所示。

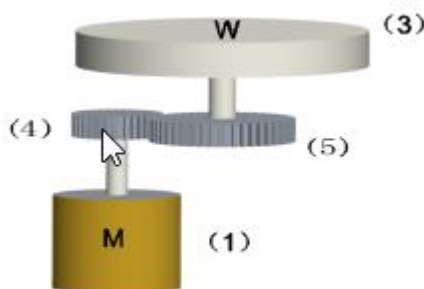


图 6.38 模数模式的工作台传动机构

图 6.38 中，(1)为伺服电机，(3)为平台及工件，(4)为主动齿轮（齿轮比分子），(5)为从动齿轮（齿轮比分母）。

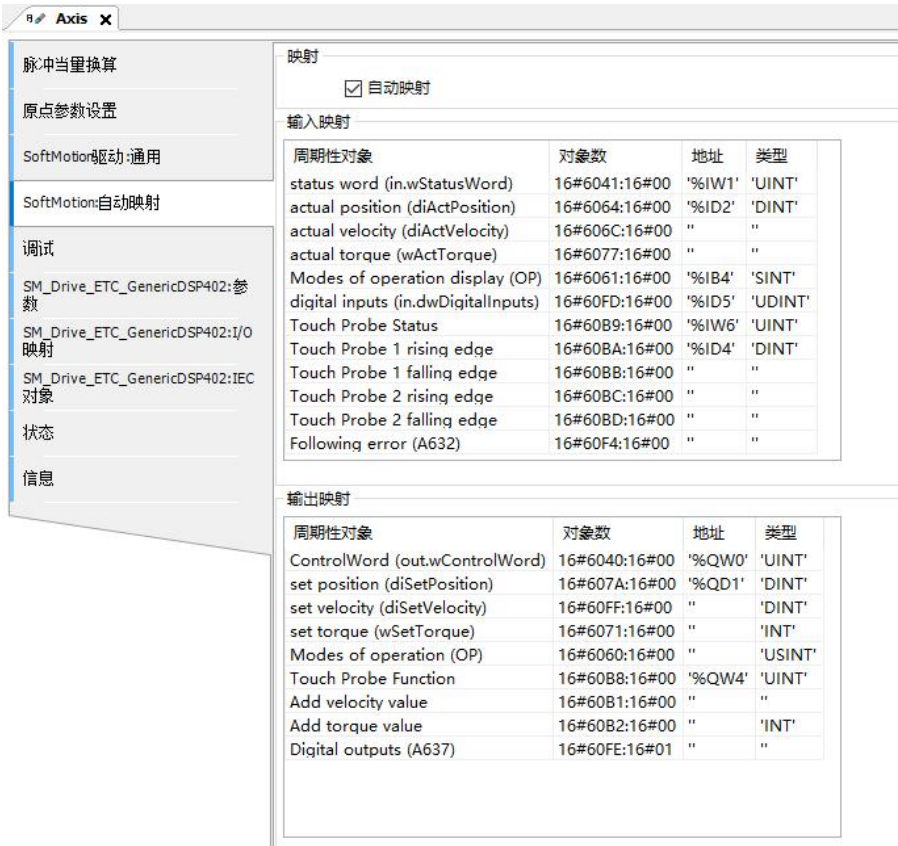
用户单位与脉冲单位的计算公式如图 6.39 所示。



图 6.39 模数模式的单位转换公式

6.4.3. 缩放/映射设置

选择"SoftMotion 自动映射"选项卡，如图 6.40 所示，可设置自动映射参数。



6.4.4. 原点参数设置

在运动控制系统中需要对平台的位置进行精确控制，因此在进行位置控制之前，需确定原点位置。在 MC 功能模块中，回零运动指令“MC_Home”，用于确定机械原点。

在调用 MC_Home 指令之前，需在图形化参数设置界面上设置回零方式、回零速度、回零加速度等参数。如图 6.41 所示，选择“原点参数设置”选项卡，设置下列参数。

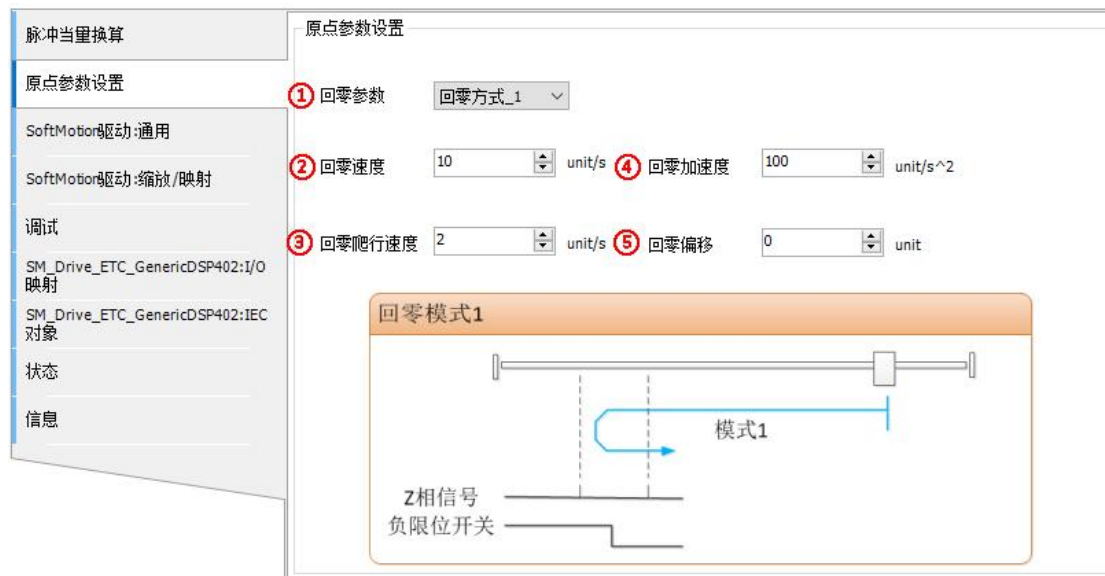


图 6.41 原点参数设置界面

① 回零参数：即回零方式

LeadSys 软件支持 CIA402 协议中的第 1~35 号，共 35 种回零方式。电机的实际回零方式由驱动器决定。每种回零方式，都有回零过程示意图。可根据实际需求选择合适的回零方式。

② 回零速度：即回零过程中的最大速度

位移量为用户单位，故速度单位为：用户单位/s。

③ 回零爬行速度：即平台进入原点开关（原点传感器）工作区域时的运动速度
速度单位为：用户单位/s。

④ 回零加速度：即回零过程中的最大加速度
加速度的单位为：用户单位/s²。

⑤ 回零偏置

回零偏置有两种方式，由驱动器决定。

方式 1：确定机械原点信号后，再移动偏置值，然后把当前位置设为零。

如雷赛 L7EC 伺服电机驱动器默认方式 1。

方式 2：确认机械原点信号后，将当前位置设为偏置值。

6.5.轴常用变量

轴变量是 `AXIS_REF_SM3` 的派生类实例，因此具备 `AXIS_REF_SM3` 的所有变量，如图 6.42 所示。主要包括以下内容：

- 1) 与轴的配置参数关联
- 2) 轴状态处理、通讯数据处理
- 3) 与 EtherCAT 主站功能块配合发送接收目标位置、控制字等到总线缓存
PDO
- 4) 接收运动控制指令信息



图 6.42 AXIS_REF_SM3 中的轴变量

如表 6.11 所示，当用户编写运动控制程序时，常需要使用其中的轴变量，用于判断轴的状态、设置轴的参数、切换轴的模式等。

表 6.11 轴变量及描述

变量名	数据类型	描述
nAxisState	SMC_AXIS_STATE	PLCopen 轴状态机的状态： 0:Power_off，去使能 1:ErrorStop，故障停机 2:Stopping，停止 3:StandStill，使能 4:Discrete_Motion，点位运动 5:Continuous_Motion，连续运动

		6: Synchronized_Motion, 同步运动 7: Homing, 原点回归
bRegulatorOn	BOOL	轴使能, MC_Power. bRegulatorOn
bDriveStart	BOOL	允许驱动, MC_Power. bDriveStart
bCommunication	BOOL	轴通讯正常标志
wCommunicationState	WORD	轴通讯状态 0~9: 初始化检测, 并将从站与同步轴关联 10~18: 通讯初始化 19: 通讯诊断初始化完成 20~28: 初始化 402 轴状态机和 SDO 收发 80~89: 等待主站同步所有从站 90: 初始化完成 100: 轴到达可操作状态 200: 接收到轴重新初始化命令 201~209: 准备重新初始化轴 210: 轴数据重新初始化完毕, 并将通讯状态切换到状态 10, 运行通讯初始化 ≥1000: 通讯异常状态
fTaskCycle	LREAL	任务周期
dwRatioTechUnitsNum	DWORD	轴缩放分子
iRatioTechUnitsNum	DINT	轴缩放分母

nDirection	MC_DIRECTION	轴运动方向设定，如 MC_MoveVelocity.Direction
fFactorACC	LREAL	加速度缩放比例
fFactorTor	LREAL	力矩缩放比例
fFactorJerk	LREAL	加加速度缩放比例
iMovementType	INT	轴类型，0:模数，1: 线性
fPositionPeriod	LREAL	模数值大小
eRampType	SMC_RAMPTYPE	速度斜坡方式
byControllerMode	BYTE	轴控制模式， SMC_SetControllerMode
byRealControllerMode	BYTE	轴控制模式反馈
fSetPosition	LREAL	设定位置
fActPosition	LREAL	反馈位置
fAimPosition	LREAL	目标位置
fSetVelocity	LREAL	设定速度
fActVelocity	LREAL	反馈速度
fMaxVelocity	LREAL	最大速度
fSWMaxVelocity	LREAL	限制速度
bConstantVelocity	BOOL	到达目标速度
fSetAcceleration	LREAL	设定加速度
fActAcceleration	LREAL	反馈加速度
fMaxAcceleration	LREAL	限制加速度
diSetPosition	DINT	设定脉冲 607A
diActPosition	DINT	反馈脉冲 6063
diSetVelocity	DINT	设定速度 60FF
diActVelocity	DINT	反馈脉冲 606C
fSetJerk	LREAL	设定加加速度

fActJerk	LREAL	反馈加加速度
fSetTorque	LREAL	设定力矩
fActTorque	LREAL	反馈力矩
fSWLimitPositive	LREAL	正限位置
fSWLimitNegative	LREAL	负限位置
usiSWEndSwitchState	USINT	轴限位状态
bSWLimitEnable	BOOL	轴限位开关
fOffsetPosition	LREAL	坐标系偏移量
dwPosOffsetForResiduals	DWORD	溢出标记
dwOneTurn	DWORD	模值脉冲
dwLastPosition	DWORD	累计脉冲数
iTurn	INT	圈数
dwActPosition	DWORD	反馈脉冲
bViRTUal	BOOL	虚轴开关

通过轴变量也能直接访问到 LeadSys 界面里的参数，如图 6.43 所示。



图 6.43 轴变量与轴参数的关系

图 6.43 中标注的参数与轴变量的关系如下：

- ① Axis_REF_SM3.bViRTUal;
- ② Axis_REF_SM3.iMovementType;

- ③ Axis_REF_SM3.bSWLimitEnable、Axis_REF_SM3.fSWLimitPositive、
Axis_REF_SM3.fSWLimitNegative;
- ④ Axis_REF_SM3.eRampType
- ⑤ Axis_REF_SM3.fSWMaxVelocity、Axis_REF_SM3.fSWMaxAcceleration、
Axis_REF_SM3.fSWMaxDeceleration、Axis_REF_SM3.fSWMaxJerk
- ⑥ Axis_REF_SM3.echeckPositionLag、Axis_REF_SM3.fMaxPositionLag

6.6.在线监控

LC/PMC 系列 PLC 可以通过 MC 指令、轴变量、轴在线界面三种方式获取轴的状态。

1) 通过指令方式获取轴状态

用户可通过 MC_ReadStatus、MC_ReadActualPosition 等状态指令获取轴的状态，如图 6.44 所示。

2) 通过轴变量获取轴状态

用户可通过轴变量实时查看各轴运动状态，如图 6.45 所示。

3) 通过轴在线界面获取轴状态

选择“SoftMotion 驱动：通用”选项卡，轴的在线界面如图 6.46 所示，可以实时查看轴的位置、速度、运动状态、通讯、错误码等信息。

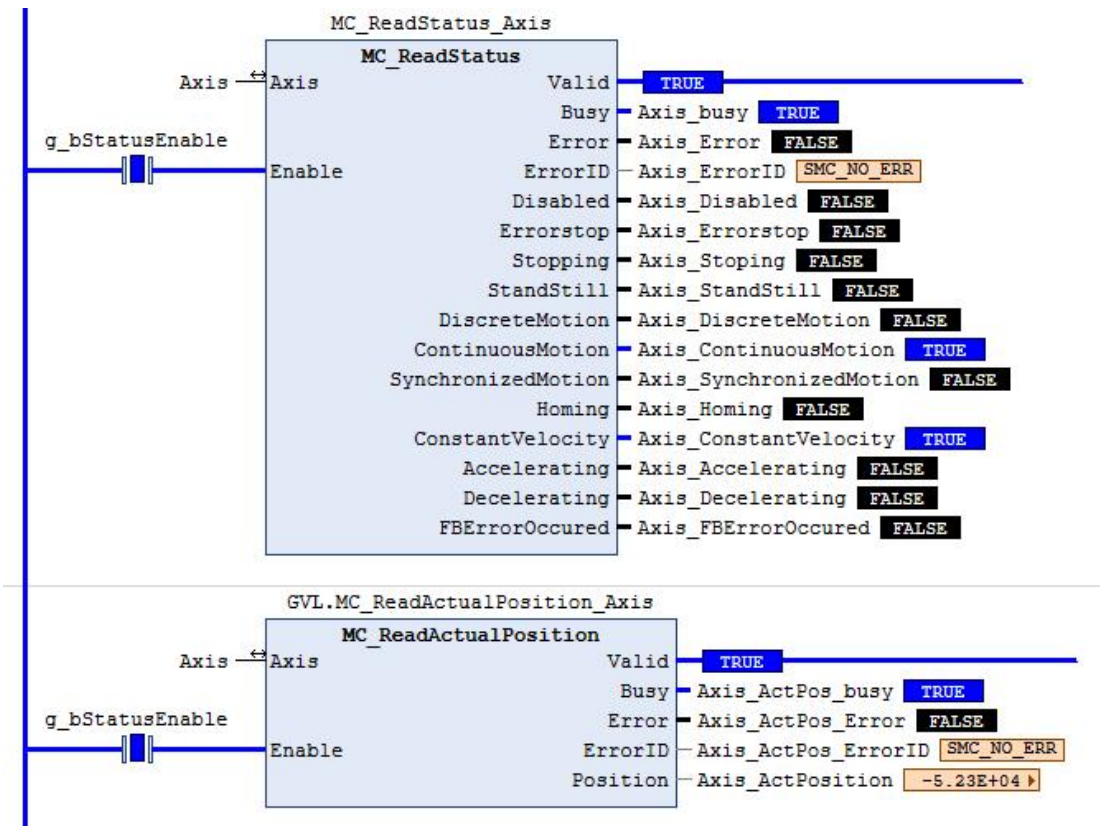


图 6.44 用指令获取轴状态

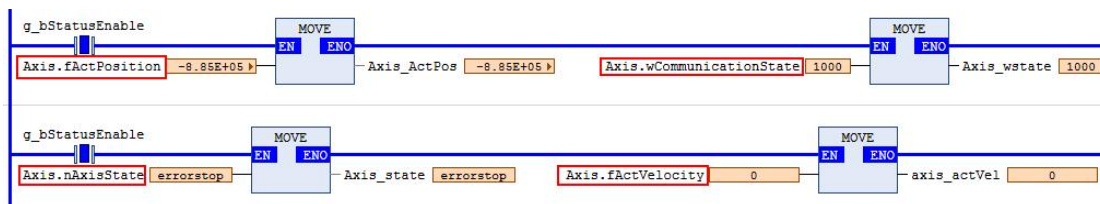


图 6.45 通过轴变量获取轴状态

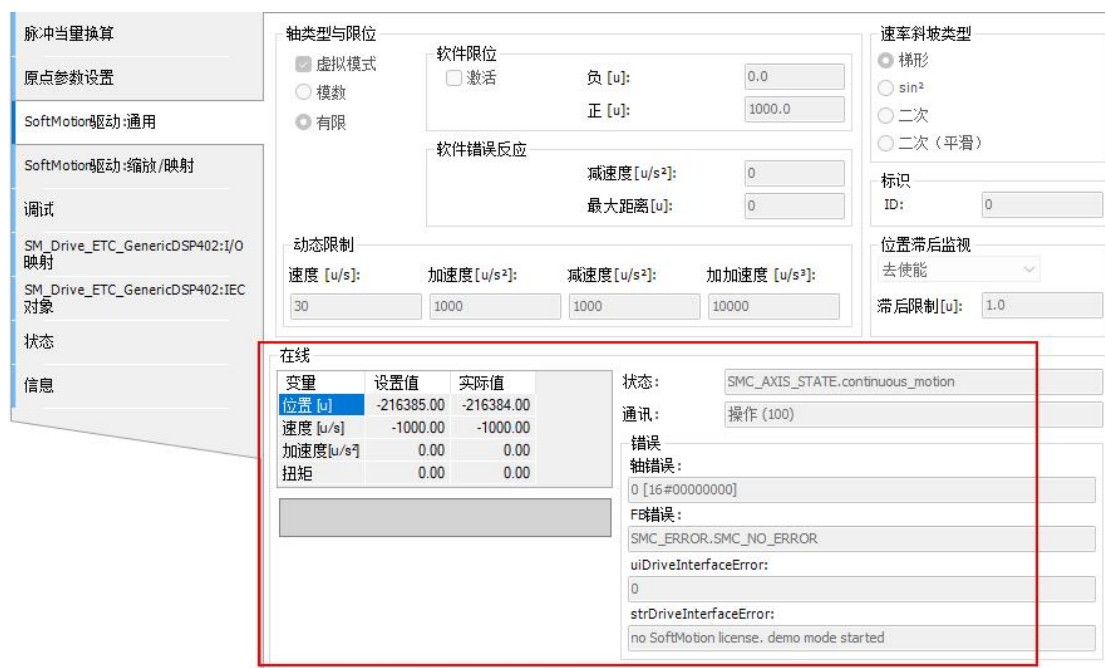


图 6.46 通过轴的参数界面获取轴状态

6.7.运动功能及指令

LC/PMC 系列 PLC 与 CIA402 轴配合的运动控制是基于 EtherCAT 总线网络来实现的，PLC 在每个 EtherCAT 总线周期中进行一次计算来实现对 CIA402 轴的控制。编程时需注意以下要点：

- 1) 用户 MC 控制程序是以 EtherCAT 任务周期执行的，因此 MC 相关的 POU 必须配置在 EtherCAT 任务下执行，若放在低优先级的任务中，无法正常运行。
- 2) MC 指令的执行，需要过程数据 PDO 来传递，因此 PDO 中需要配置伺服必需的对象。
- 3) 控制器可通过 SDO 对伺服的功能码进行初始化设定操作，如设定伺服电机的运行模式、电子齿轮比等参数。
- 4) 使用 MC 指令对伺服轴进行控制，需遵循轴状态转移的规则，根据轴当前的状态和期望的运动，调用合适的 MC 指令。

6.8.单轴运动

● 控制指令

单轴控制一般可通过位置控制（CSP）、速度控制（CSV）、转矩控制（CST）这 3 种模式控制轴运动。单轴控制中，常用的 MC 指令如表 6.12 所示，指令的详细说明请参考指令手册。

● 单轴指令与 PDO/SDO 配置

LC/PMC 控制器在执行用户程序的轴 MC 指令时，需要将执行 MC 指令时与伺服电机驱动器交互所需的过程数据，添加在通信 PDO/SDO 配置表中，以便完成所需的控制功能，表 6.13 中的 TPDO/RPDO 是单轴指令所需的基本配置参数。

表 6.12 常用的 MC 单轴控制指令

MC 指令	指令名称	概要
MC_Power	轴使能	将驱动器切换为可运行状态
MC_SetPosition	轴位置设置	将指令中的位置数据设置为当前轴的位置，该操作不会产生位移移动，仅产生坐标系偏移
MC_ReadStatus	读取轴的状态	指令读取轴的状态数据，保存在自己定义的变量单元中
MC_Jog	点动运行	根据指定目标速度执行点动移动
SMC_Inch	寸动运行	手动控制轴一段一段的朝指定方向运动
MC_MoveRelative	相对定位	指定自指令当前位置起的移动距离，进行定位
MC_MoveAdditive	相对叠加定位	在当前运行指令的基础上，再相对运行指定距离
MC_MoveAbsolute	绝对定位	指定绝对坐标的目标位置，进行定位

MC_MoveVelocity	速度控制	使用伺服驱动器的位置控制模式，命令伺服以指定的速度运行
MC_Halt	暂停	命令伺服暂停运行，可再次调用 MC_Movexxx 指令触发运行
MC_Stop	停止	可对单轴控制和同步控制功能指定减速度和跃度并减速停止
MC_Reset	轴报警复位	当伺服出现报警停止后，运行该指令进行复位。如轴状态 Axis.nAxisState 为 errorstop 需要切换到 StandStill 要用该指令复位
SMC3_ReinitDrive	重启驱动器	在总线同步帧丢失后、轴配置参数修改后需要使用该指令重新初始化轴
SMC_ClearFBError	清除功能块的 FB 错误	MC_ReadStatus 指令输出变量 FBerroroccured 为 True 时需要用该函数复位
SMC_SetControllerMode	切换轴运行模式	使用该指令可以让伺服选择位置、速度或力矩模式
SMC_SetTorque	力矩控制	命令伺服以指定的力矩运行
MC_Home	驱动器原点复位	命令伺服开始原点回归操作，应用系统的原点信号、两侧极限信号等都接在伺服的 DI 端口

表 6.13 单轴指令所需的基本配置参数

MC 指令	TPDO 对象	RPDO 对象
MC_Power MC_Halt MC_Stop MC_Reset MC_Home SMC_Homing	ControlWord（控制字）： 6040	StatusWord（状态字）： 6041 Errorcode（错误码）： 603F
MC_Jog MC_MoveRelative MC_MoveAdditive MC_MoveAbsolute	TargetPosition（目标位置）：607A	Position actual value （当前实际位置）：6064
MC_MoveVelocity	Target velocity（目标速度）：60FF Max profile velocity（最大轮廓速度）：607F	Velocity actual value （当前实际速度）：606C
MC_SetTorque	Target torque（目标力矩）：6071	Torque actual value （当前实际力矩）：6077
SMC_SetControllerMode	Modes of operation（运行模式）：6060 各模式值如下： 16#6060=8: CSP 模式 16#6060=9: CSV 模式 16#6060=10: CST 模式	Modes of operation display （运行模式显示）：6061

● 单轴运动程序编写

1) 单轴运动程序一般按以下流程编写：

- a) 判断 EtherCAT 总线通信是否正常，同步是否成功。如果 EtherCAT 主站.xConfigFinished 和 EtherCAT 主站.xDistributedClockInSync 都为 TRUE，则总线通信正常，参见图 6.44。
- b) B. 调用轴变量 axis.wCommunicationState，判断其值是否=100，确认 CIA402 轴为可操作状态，参见图 6.45。
- c) C. 调用 MC_Power 使能轴，确认轴使能正常且轴状态机 axis.nAxisState=standstill 状态。
- d) D. 调用 MC_MoveAbsolute, MC_MoveRelative, MC_MoveVelocity 等单轴运动指令，执行单轴运动。
- e) E. 如果运动中需要终止单轴运动，可通过调用 MC_Stop 指令停止轴运动。

注意：MC 单轴运动指令必须放到 EtherCAT 任务下执行，因此所有 MC 相关的运动指令可以放到一个 POU 中，将此 POU 放到 EtherCAT 任务下。

2) 单轴运动程序示例：

在 GVL 中定义程序中需要使用的全局变量，如图 6.47 所示。

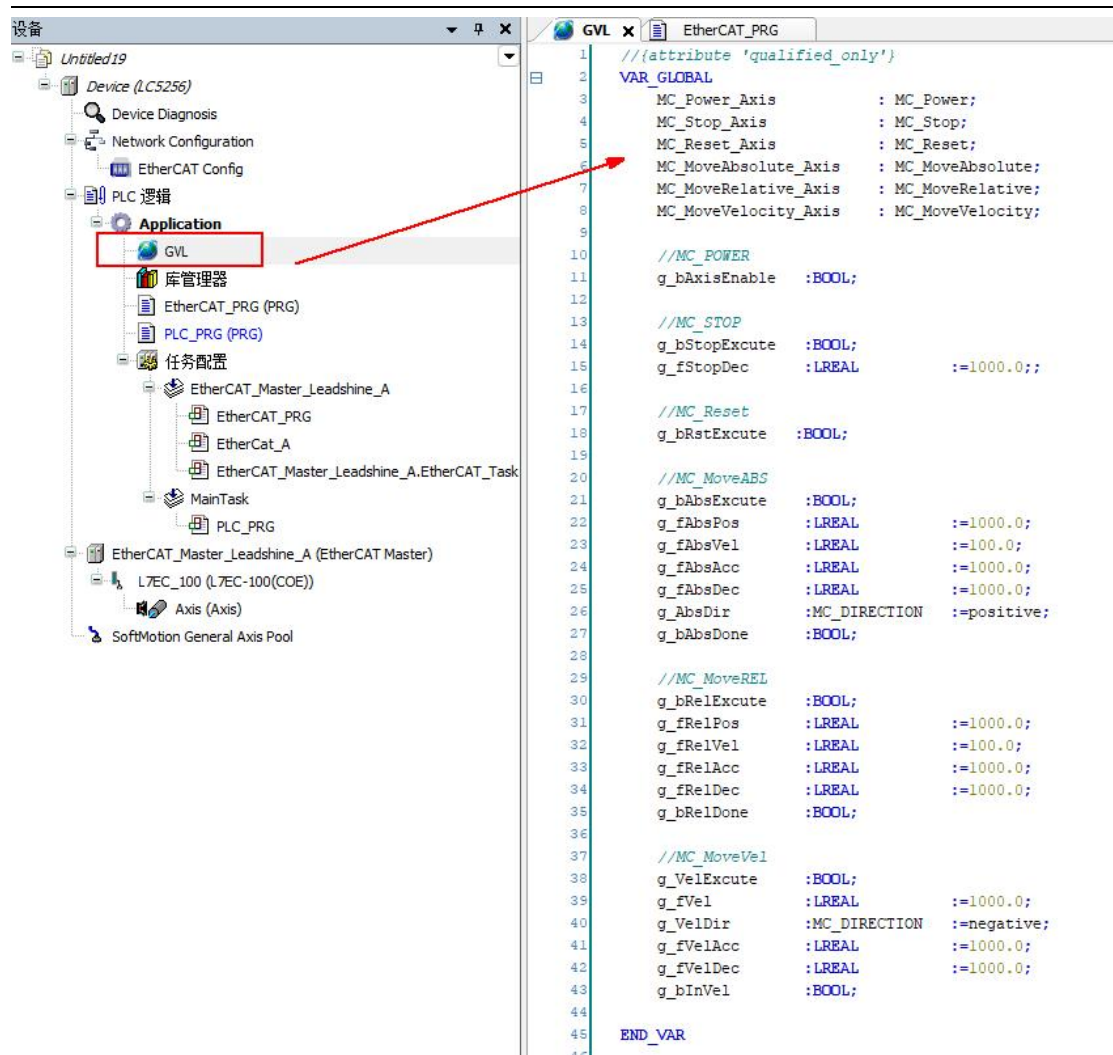


图 6.47 定义全局变量

新建 2 个 POU, 名称分别取为 EtherCAT_PRG 和 PLC_PRG, 编程语言均选择 ST。调用 MC_Power, MC_MoveAbsolute, MC_MoveRelative, MC_MoveVelocity, MC_Stop, MC_Reset 等单轴运动指令, 写在 EtherCAT_PRG 中, 如图 6.48 所示。逻辑控制指令写在 PLC_PRG 中, 如图 6.49 所示。

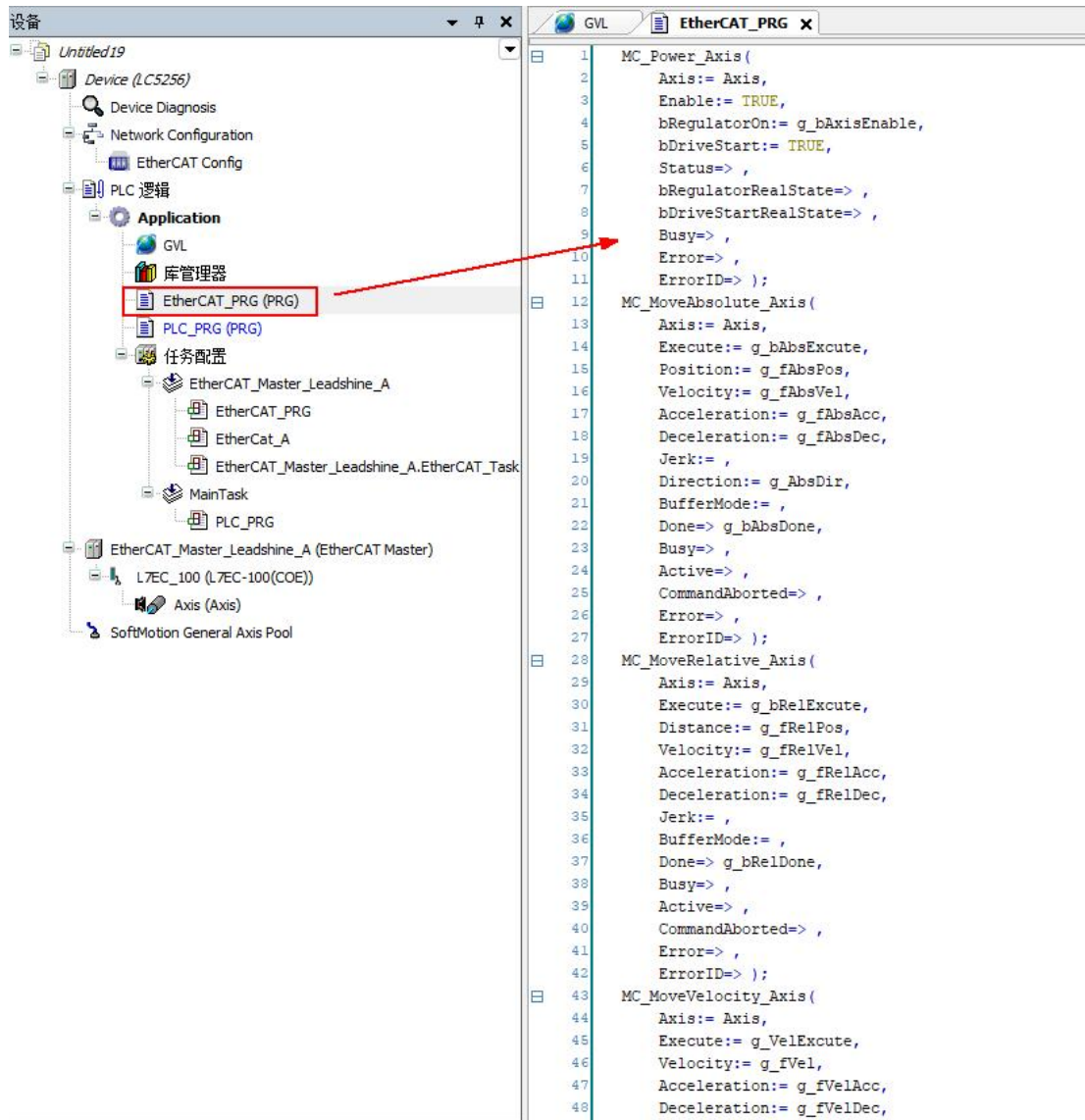


图 6.48 调用 MC 单轴运动指令

```

1 // 确定伺服通讯连接正常
2 IF EtherCAT_Master_Leadshine_A.xConfigFinished AND EtherCAT_Master_Leadshine_A.xDistributedClockInSync THEN
3     IF Axis.wCommunicationState=100 THEN // 轴可操作状态
4         EtherCAT_State:=TRUE;
5     END_IF
6 ELSE
7     EtherCAT_State:=FALSE;
8     RETURN; // 伺服总线通讯异常，不执行以下轴运动程序
9 END_IF
10 CASE Axis_state OF
11     0:
12         IF g_bAxisEnable THEN // 使能轴
13             Axis_state:=10;
14         END_IF
15
16     10:
17         IF MC_Power_Axis.Status AND Axis.nAxisState = standstill THEN // 轴使能正常且状态机为standstill
18             Axis_state:=20;
19         END_IF
20
21     20:
22         IF g_bAbsExcute THEN // 启动绝对运动
23             IF g_bAbsDone AND NOT MC_MoveAbsolute_Axis.Error THEN // 绝对运动正常完成
24                 g_bAbsExcute:=FALSE;
25             END_IF
26         END_IF
27         IF g_bRelExcute THEN // 启动相对运动
28             IF g_bRelDone AND NOT MC_MoveRelative_Axis.Error THEN // 相对运动正常完成
29                 g_bRelExcute:=FALSE;
30             END_IF
31         END_IF
32         IF g_VelExcute THEN // 启动速度运行
33             IF MC_MoveVelocity_Axis.Busy THEN
34                 g_VelExcute:=FALSE;
35             END_IF
36         END_IF
37         IF g_bStopExcute THEN // 停止运动
38             IF MC_Stop_Axis.Done THEN
39                 g_bStopExcute:=FALSE;
40             END_IF
41         END_IF
42         IF g_bRstExcute THEN // 复位轴errstop进入standstill状态
43             IF MC_Reset_Axis.Done THEN
44                 g_bRstExcute:=FALSE;
45             END_IF
46         END_IF
47     END_CASE

```

图 6.49 逻辑控制程序

单击鼠标左键拖拽 EtherCAT_PRG 到 EtherCAT_Master_Leadshine_A 任务下，PLC_PRG 到 MainTask 任务下，如图 6.50 所示；

配置 MainTask 的任务周期优先级低于 EtherCAT_Master_Leadshine_A 任务，如优先级 4，周期 4ms，如图 6.51 所示。

完成配置后，登入下载程序到 PLC 中，即可实现单轴运动控制。

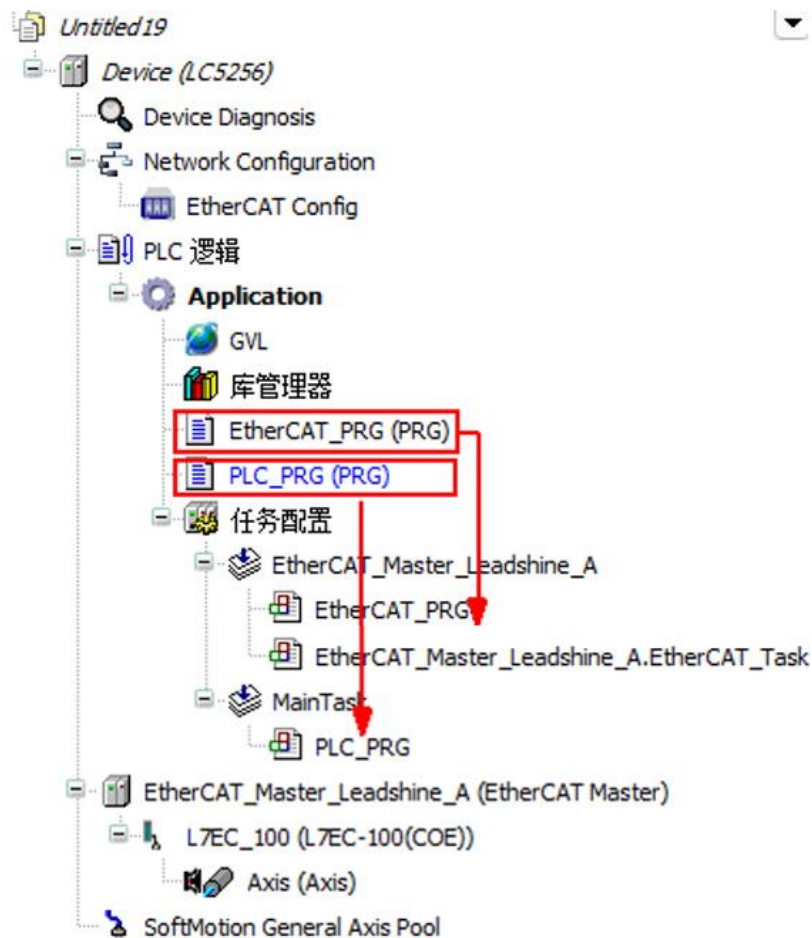


图 6.50 配置任务

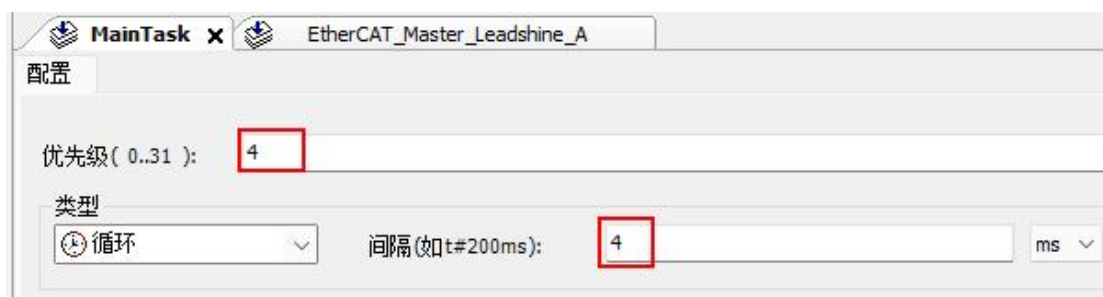


图 6.51 设置任务的参数

6.9.回零运动

完成 6.4.4 节中所描述的原点参数设置后，用户可通过 MC_Power 指令使能电机驱动器；然后通过 MC_Home 指令启动回零运动，该指令实现电机驱动器端原点复归功能（从站回零），所以原点及正、负限位信号需接到电机驱动器上，具体引脚需参考电机驱动器手册。

回零运动一般按以下流程编写：

- 判断 EtherCAT 总线通信是否正常，同步是否成功。如果 EtherCAT 主站.xConfigFinished 和 EtherCAT 主站.xDistributedClockInSync 都为 TRUE, 则总线通信正常。
- 调用轴变量 axis.wCommunicationState, 判断其值是否=100, 确认 CIA402 轴为可操作状态。
- 调用 MC_Power 指令使能轴, 确认轴使能正常且轴状态机 axis.nAxisState = standstill 状态。
- 调用 MC_Home 指令, 执行回零运动。
- 如果运动中需要终止回零运动, 可通过调用 MC_Stop 指令停止轴运动。

注意：MC 回零运动指令必须放到 EtherCAT 任务下执行，因此所有 MC 相关的回零运动指令可以放到一个 POU 中，将此 POU 放到 EtherCAT 任务下。

回零示例：

在 GVL 中定义程序中需要使用的全局变量，如图 6.52 所示。

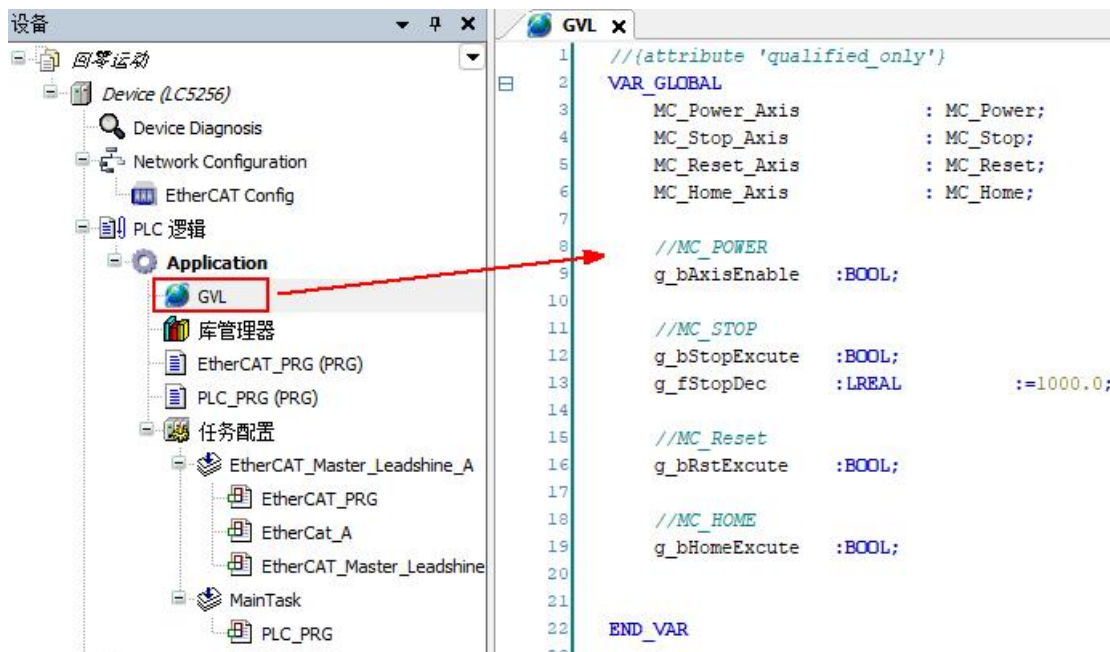


图 6.52 设置全局变量

新建 2 个 POU, 名称分别取为 EtherCAT_PRG 和 PLC_PRG, 编程语言均选择 ST。调用 MC_Power、MC_Home、MC_Stop、MC_Reset 等回零运动指令, 编写到 EtherCAT_PRG 中, 如图 6.53 所示。逻辑控制指令编写到 PLC_PRG 中, 如图 6.54 所示。

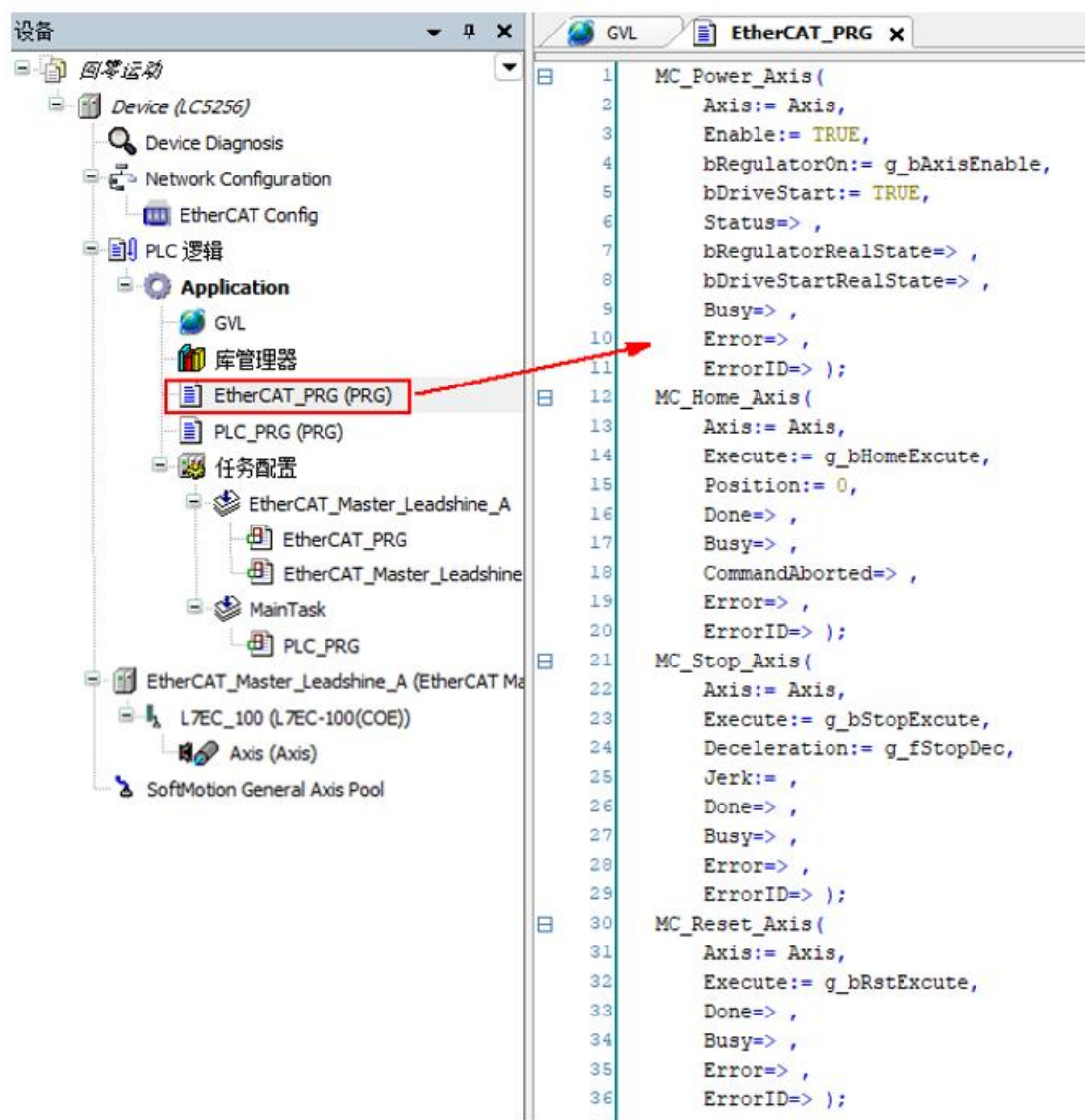


图 6.53 调用回零相关指令


```

1 // 确定伺服通讯连接正常
2 IF EtherCAT_Master_Leadshine_A.xConfigFinished AND EtherCAT_Master_Leadshine_A.xDistributedClockInSync THEN
3     IF Axis.wCommunicationState=100 THEN //轴可操作状态
4         EtherCAT_State:=TRUE;
5     END_IF
6     ELSE
7         EtherCAT_State:=FALSE;
8         RETURN; //伺服总线通讯异常，不执行以下轴运动程序
9     END_IF
10 CASE Axis_state OF
11     0:
12         IF g_bAxisEnable THEN //使能轴
13             Axis_state:=10;
14         END_IF
15
16     10:
17         IF MC_Power_Axis.Status AND Axis.nAxisState = standstill THEN //轴使能正常且状态机为standstill
18             Axis_state:=20;
19         END_IF
20     20:
21         IF g_bHomeExcute THEN //启动回零
22             IF MC_Home_Axis.Done AND NOT MC_Home_Axis.Error THEN
23                 g_bHomeExcute:=FALSE;
24             END_IF
25         END_IF
26         IF g_bStopExcute THEN //停止运动
27             IF MC_Stop_Axis.Done THEN
28                 g_bStopExcute:=FALSE;
29             END_IF
30         END_IF
31         IF g_bRstExcute THEN //复位轴errstop进入standstill状态
32             IF MC_Reset_Axis.Done THEN
33                 g_bRstExcute:=FALSE;
34             END_IF
35         END_IF
36 END_CASE

```

图 6.54 逻辑控制指令

单击鼠标左键拖拽 EtherCAT_PRG 到 EtherCAT_Master_Leadshine_A 任务下，PLC_PRG 到 MainTask 任务下，配置 MainTask 的任务周期优先级低于 EtherCAT_Master_Leadshine_A 任务，如优先级 4，周期 4ms。方法与上例相同。完成配置后，登入下载程序到控制器中，即可实现回零运动。

第七章 连续插补轴组运动

7.1. 轴组的概念

轴组是由多个运动轴所组成的轴的集合，轴组可将多个一维方向运动的轴组合起来并且映射到轴组里使轴按照设定的运动学模型进行二维、三维等多维方向的运动。轴组能够被相关的轴组指令所调用，以实现多轴的单段插补、多轴的连续插补功能。并且当轴组的运动学模型选择机器人的运动学模型时可对机械手进行控制。

7.2. 运动学模型的介绍

Leadsys 中含有各种的运动学模型，如图 7.1 所示。

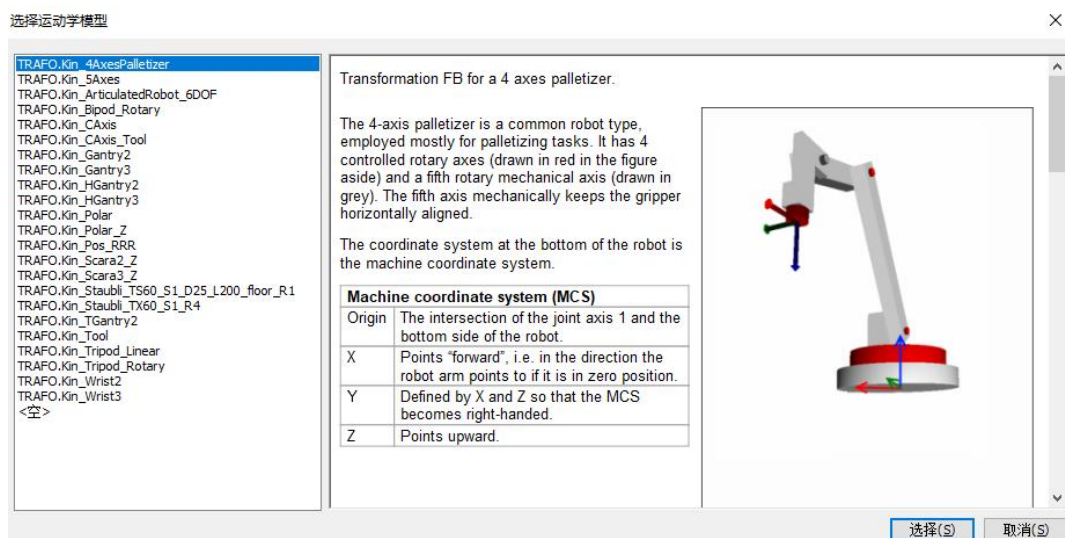


图 7.1 leadsys 中全部的运动学模型

以下重点介绍常用的两种运动学模型 TRAFO.Kin_Gantry2（两轴运动学变化）和 TRAFO.Kin_Gantry3（三轴运动学变化）

7.2.1. TRAFO.Kin_Gantry2（两轴运动学变化）

两轴运动学平面模型在 leadsys 中的选择界面如图 7.2 所示

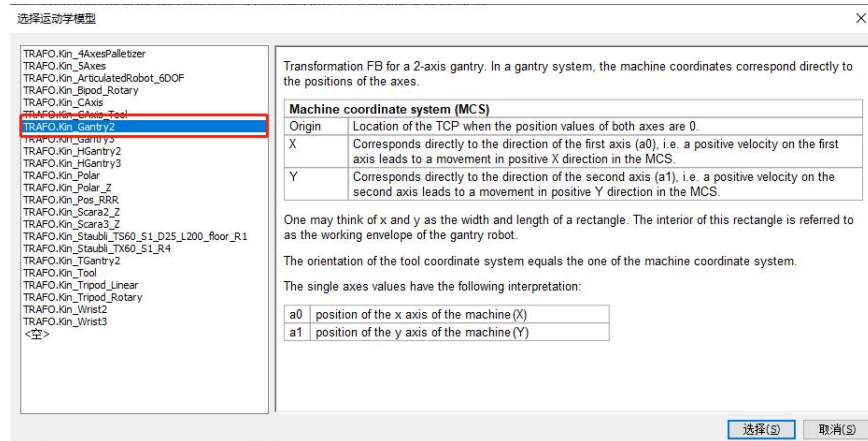


图 7.2 TRAFO.Kin_Gantry2 选择界面

TRAFO.Kin_Gantry2（两轴运动学变化）是平面类的运动学模型。如果在龙门系统中，机械坐标系与轴位置直接对应，用户可以如此认为 X 和 Y 的宽度和长度形成矩形。在这个矩形的内部称为龙门机器人的工作面。工具坐标系的方向与其中一个机械坐标系方向相同。

其中其机械坐标系（MCS）有如下含义

机械坐标系（MCS）	
原点	当全部轴为 0 的时候 TCP 位置。
X	与第一个轴(a0)的方向直接对应，即第一个轴上的正速度会导致 MCS 中 x 方向的正运动。
Y	与第二个轴(a1)的方向直接对应，即在第二轴上的正速度会导致 MCS 中 y 方向的正运动。

单轴值具有如下含义

a0	机械 X 轴位置 (X)
a1	机械 Y 轴位置 (Y)

为了方便用户理解，其运动学模型含义如图 7.3 所示，即为在一个平面内其 Y 方向对应 Y 轴即为 a1，其 X 方向对应为 X 轴即为 a0。

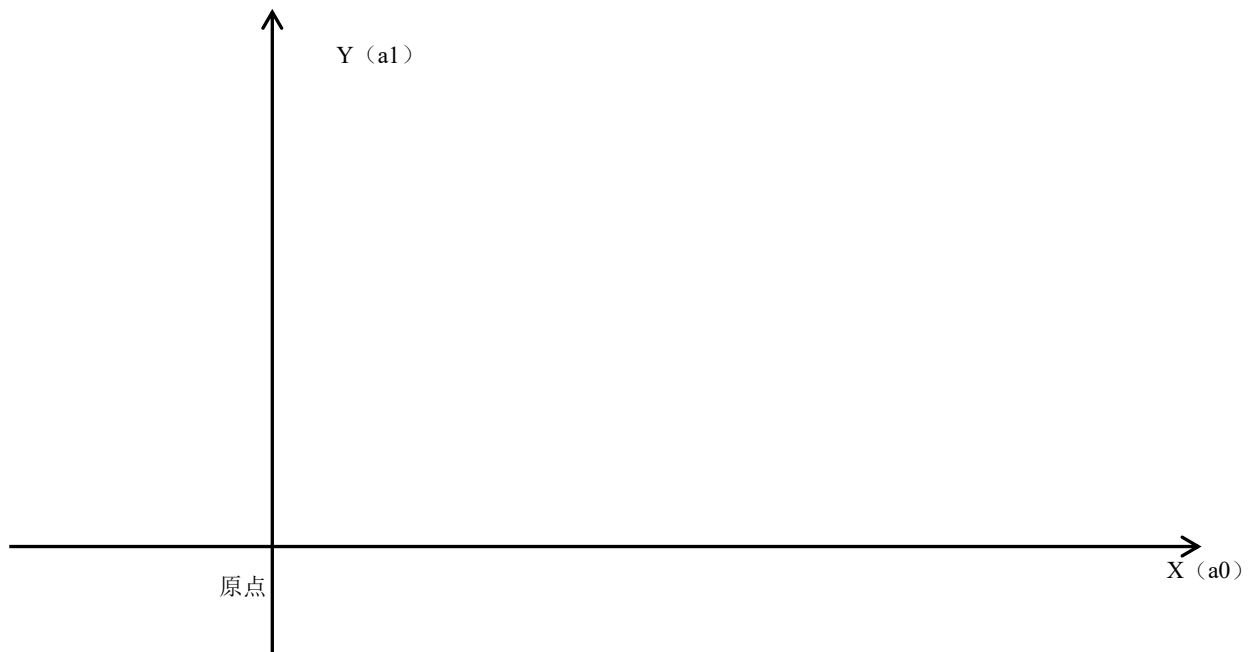


图 7.3 TRAFO.Kin_Gantry2 运动学模型含义

7.2.2. TRAFO.Kin_Gantry3（三轴运动学变化）

两轴运动学平面模型在 leadsys 中的选择界面如图 7.4 所示

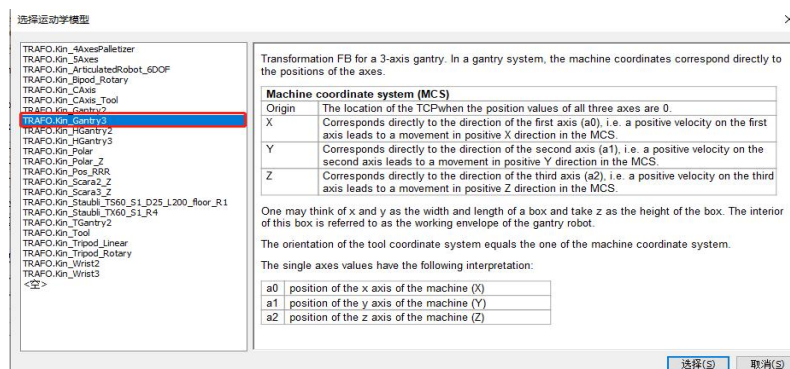


图 7.4 leadsys TRAFO.Kin_Gantry3 选择界面

TRAFO.Kin_Gantry3（三轴运动学变化）是空间类的运动学模型。在龙门系统中，机械坐标系与轴位置直接对应，用户可以这样理解假设在空间坐标系为一个盒子，其 x 和 y 看成是盒子的宽度和长度， z 看成盒子的高度。这个盒子的内部即可称为龙门机器人的工作面。

工具坐标系的方向与其中一个机械坐标系方向相同。

其中其机械坐标系（MCS）有如下含义

机械坐标系（MCS）	
原点	当全部轴为 0 的时候 TCP 位置。
X	与第一个轴(a0)的方向直接对应，即第一个轴上的正速度会导致 MCS 中 x 方向的正运动。
Y	与第二个轴(a1)的方向直接对应，即第二轴上的正速度会导致 MCS 中 y 方向的正运动。
Z	Z 与第三轴(a2)的方向直接对应，即第三轴上的正速度会导致 MCS 中 z 方向的正运动。

单轴值具有如下含义

a0	机械 x 轴位置(X)
a1	机械 Y 轴位置(Y)
a2	机械 Z 轴位置(Z)

为了方便用户理解，其运动学模型含义如 7.5 图所示，即为在一个空间内其 Y 方向对应 Y 轴即为 a1，其 X 方向对应为 X 轴即为 a0。其 Z 方向对应 Z 轴即为 a2。

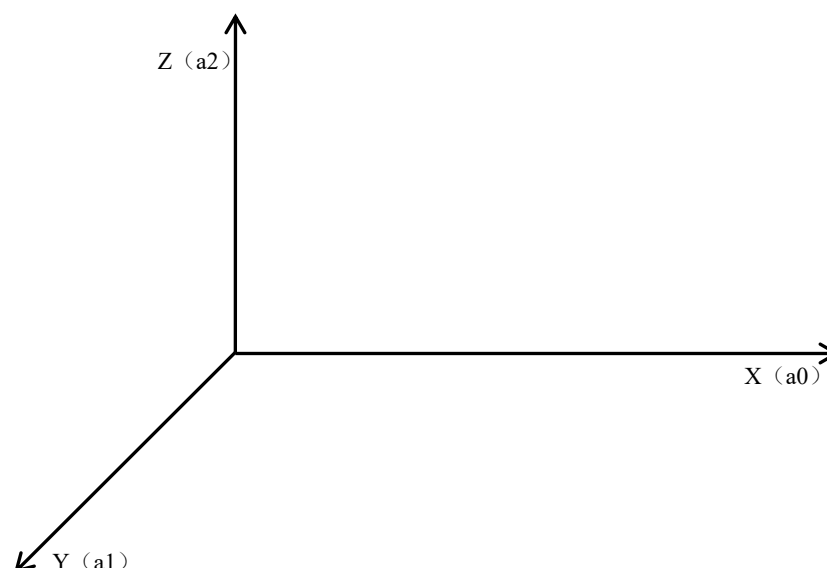


图 7.5 TRAFO.Kin_Gantry3 运动学模型

用户在选择坐标系时，可以根据连续插补需要进行插补类型来选择。

1) 如果选择平面类型的连续插补轴组，则选择 TRAFO.Kin_Gantry2 运动学模型；

2) 如果选择空间类型的连续插补轴组，则选择 TRAFO.Kin_Gantry3 运动学模型。

7.3.轴组使用和建立方式

首先在设备中找到 Application，在右击 Application 找到添加对象然后点击轴组即完成了轴组的建立，其建立顺序如图 7.6 所示，

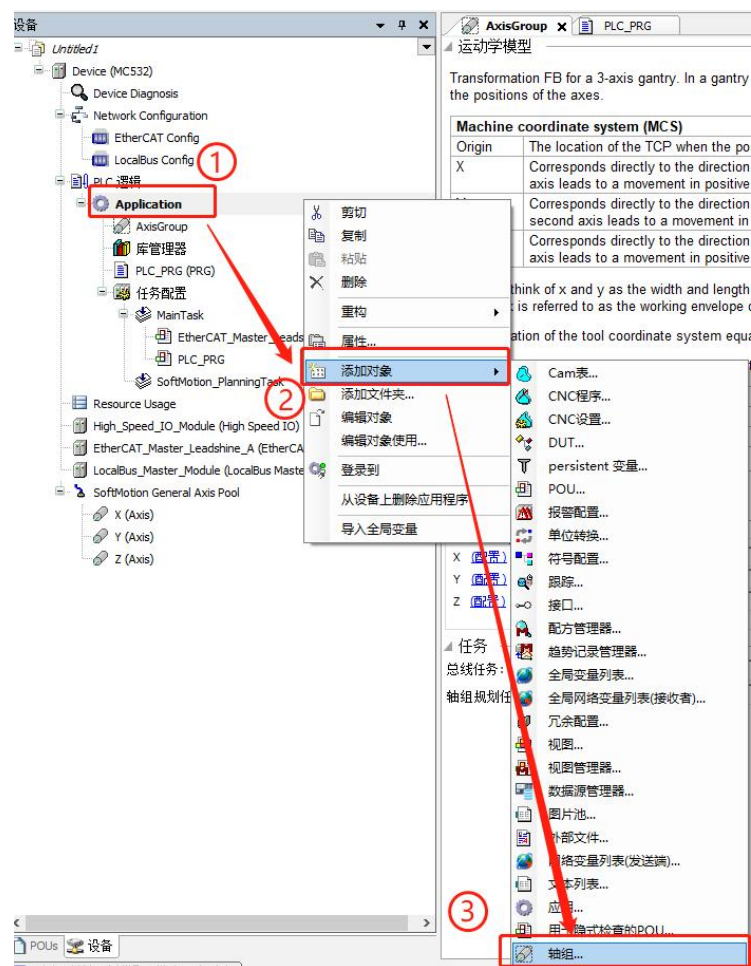


图 7.6 轴组建立顺序

当轴组建立好后，在设备的目录树下可以看到建立好的轴组（Axis Group），如

图 7.7 所示。

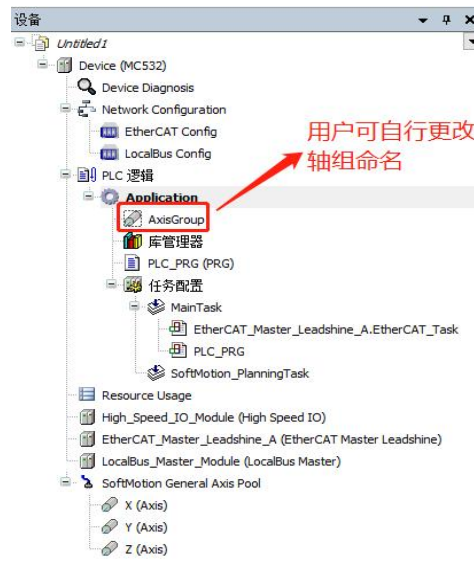


图 7.7 轴组建立好在目录树下的显示

更改轴组名称方式，先右击轴组（Axis Group）找到重构，点击重命名轴组，输入想要更改的轴组名称，点击确定，就完成对轴组的重命名的过程，其流程图分别如下图 7.8、7.9、7.10

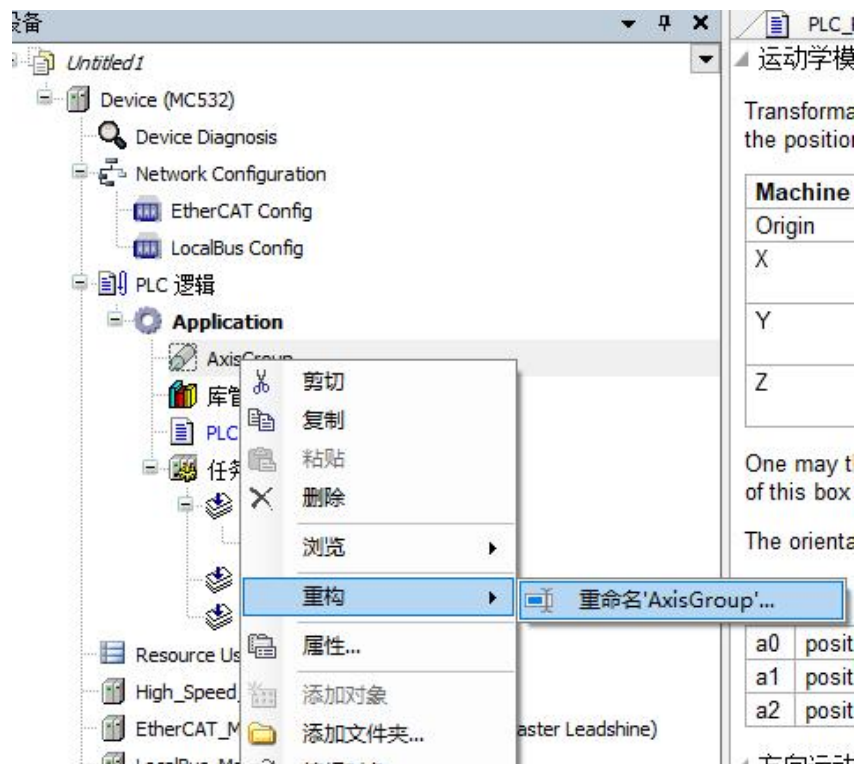


图 7.8 轴组重命名

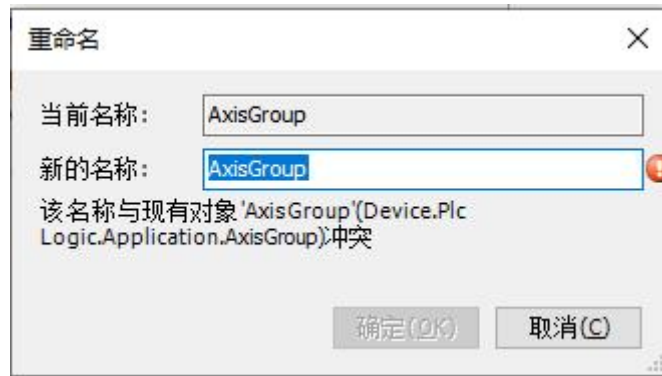


图 7.9 输入轴组重命名的名字

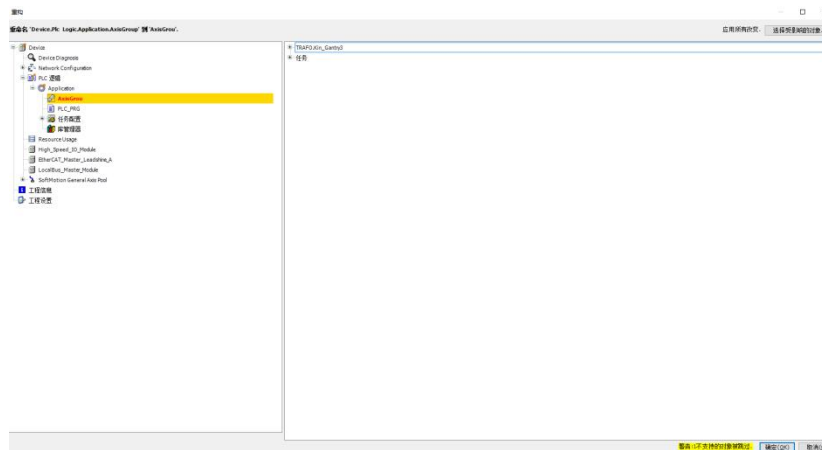


图 7.10 轴组重构界面

建立好轴组后点击 Axis Group 打开轴组界面，可以看到图 7.11 的界面。

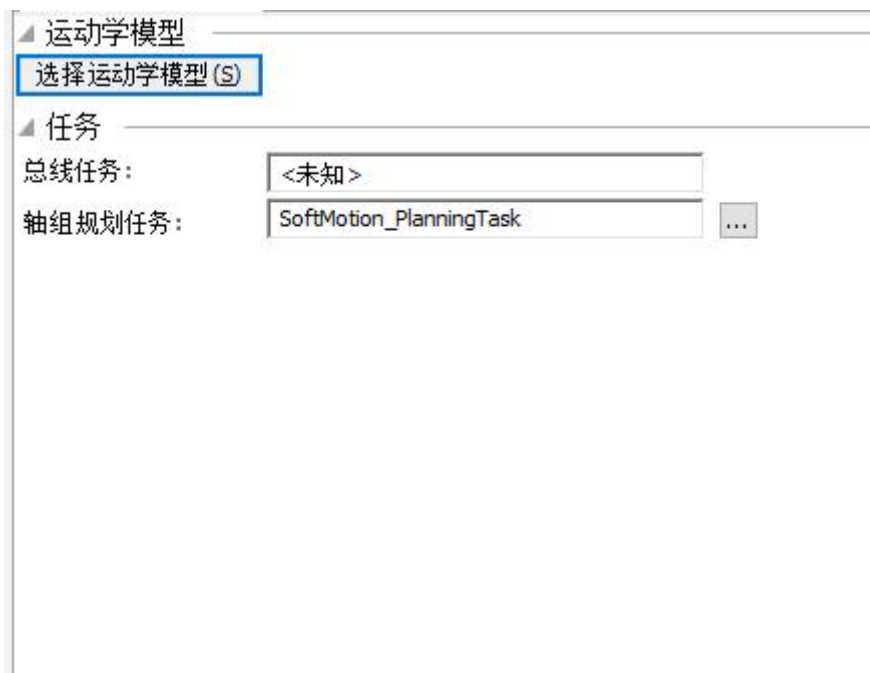


图 7.11 轴组打开界面

点击选择运动学模型，选择相对应的运动学模型（本次应用手册的程序例程主要使用 TRAFO.Kin_Gantry3 三轴运动模型，故以建立 TRAFO.Kin_Gantry3 为本次手册示例）点击确定其建立流程顺序如图 7.12 所示

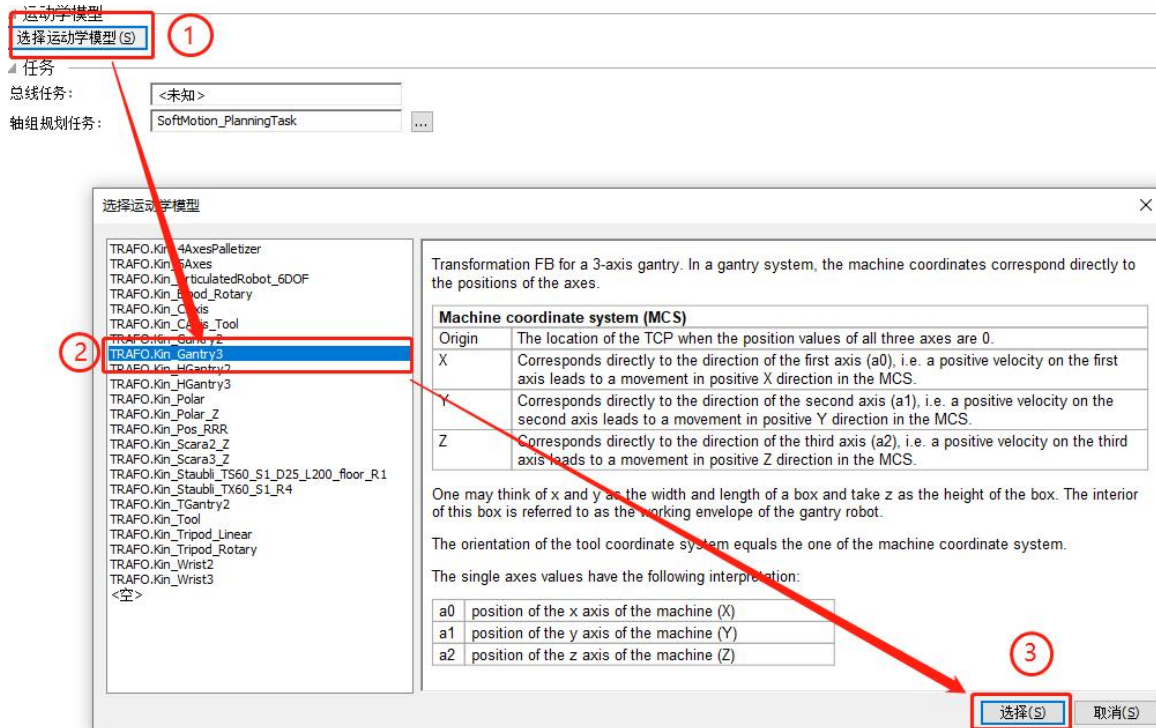


图 7.12 轴组建立顺序图

其建立好的界面如图 7.13 所示，在建立好的界面中找到映射到轴点击 X、Y、Z 旁边的...在打开的界面中 IoConfig_Globals 旁边的加号，点击启用的脉冲轴（映射到轴组的轴可以为脉冲轴或者总线轴），点击确定即可把轴映射到轴组里。其建立流程顺序图如 7.13 所示。

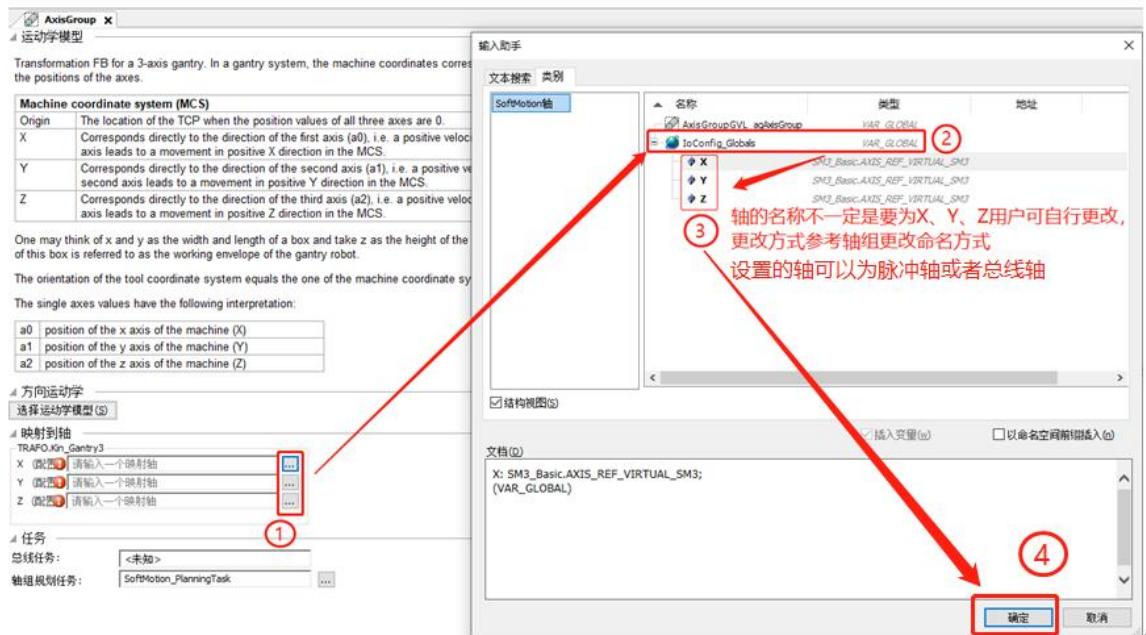


图 7.13 映射轴流程

在轴组的打开界面中找到任务，有两个任务分别为总线任务和轴组规划任务，如图 7.14 所示。

总线任务：轴的总线循环任务（如果轴已映射）。

轴组规划任务：用于规划 CP 任务的任务。插入第一个轴组时，将自动创建规划任务。此处使用的任务的优先级应低于轴组的总线周期任务（建议值：15）。这项任务应该是随心所欲的。不应配置看门狗，因为执行时间可能会大幅波动。如果删除此任务，则会向消息视图发出错误。要再次创建新任务，请单击项目→创建计划任务。

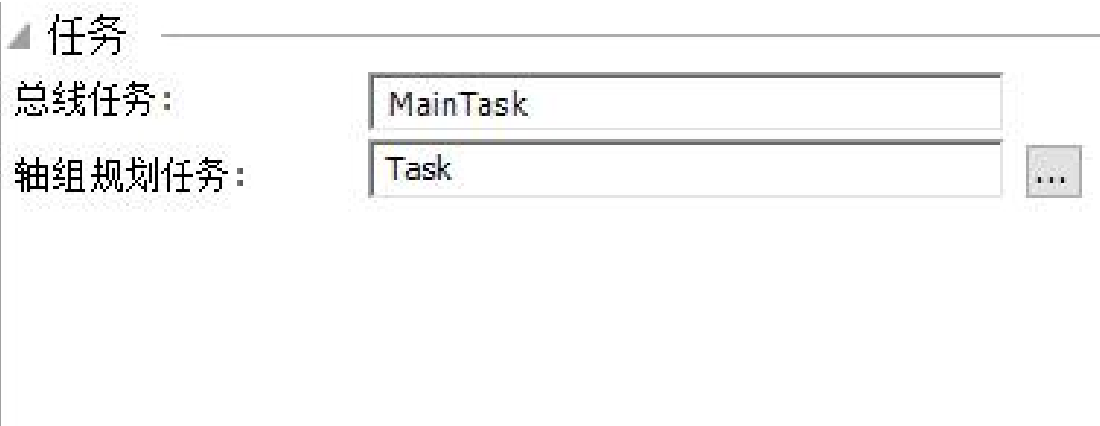


图 7.14 轴组任务配置

此处的总线任务和轴组规划任务用户可以自行更改。总线任务更改流程如下

首先在目录树下双击 **DEVICE**，打开后选择 **PLC 设置** 找到总线任务周期选项，点击总线周期任务用户即可自行选择总线周期任务。流程图如图 7.15 所示。

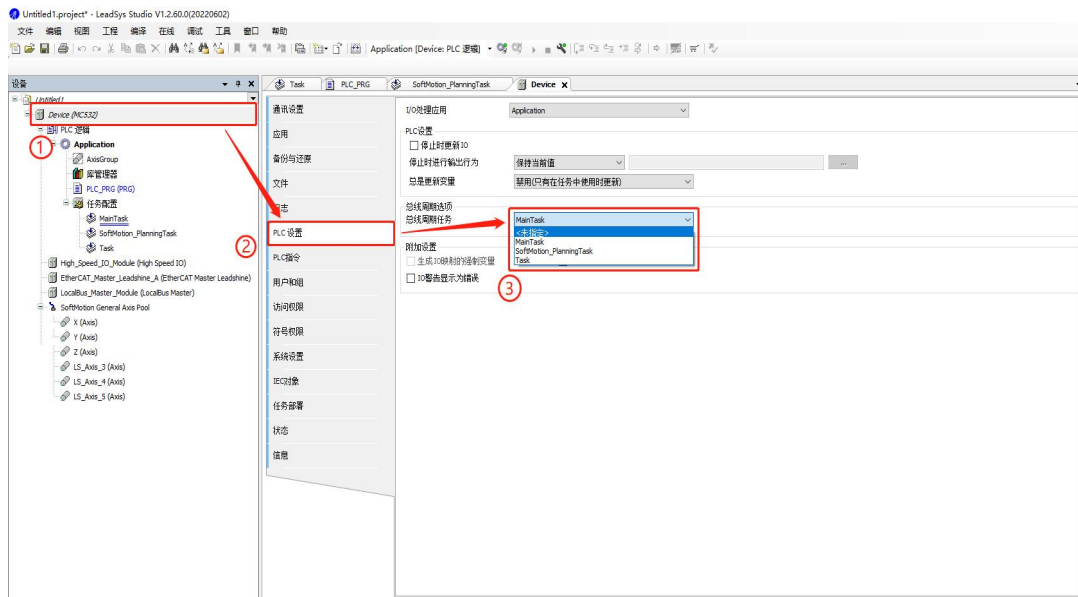


图 7.15 更改总线任务流程图

轴组规划任务更改流程如下：

在轴组打开界面中点击轴组任务规划旁边的...选择相对应你需要设置的轴组规划的任务即可（注意：轴组的规划任务首先需要将看门狗失效，目前只有 LC 系列的控制器才可以使得看门狗使能或者失效其余类型的控制器看门狗自动上使能，其失效流程图如图 7.16 所示）

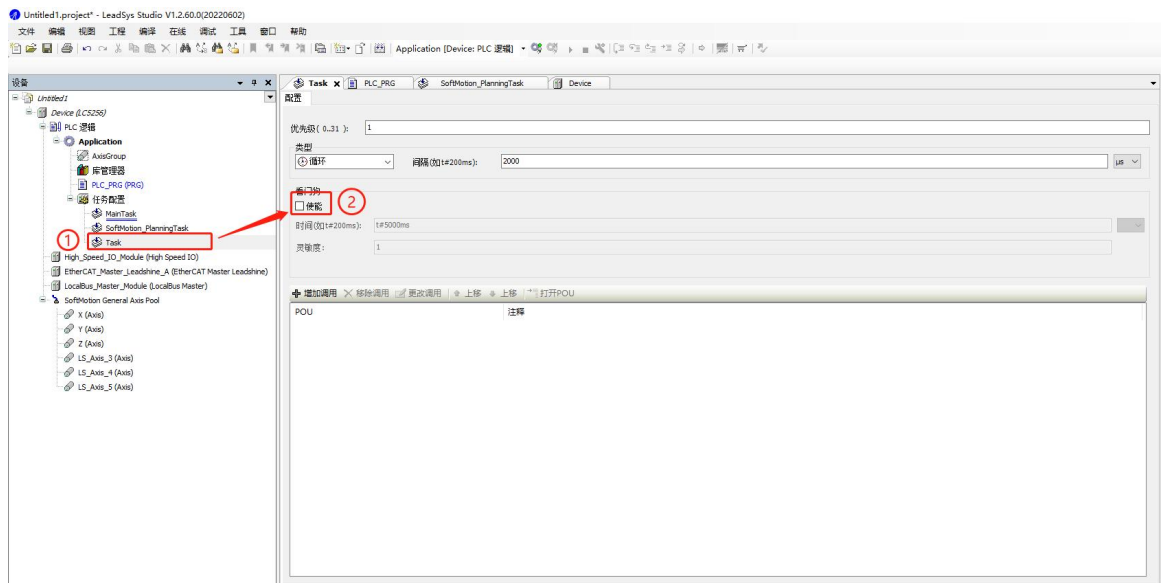


图 7.16 看门狗失效流程图

当轴组建立好后界面如图 7.17 所示。

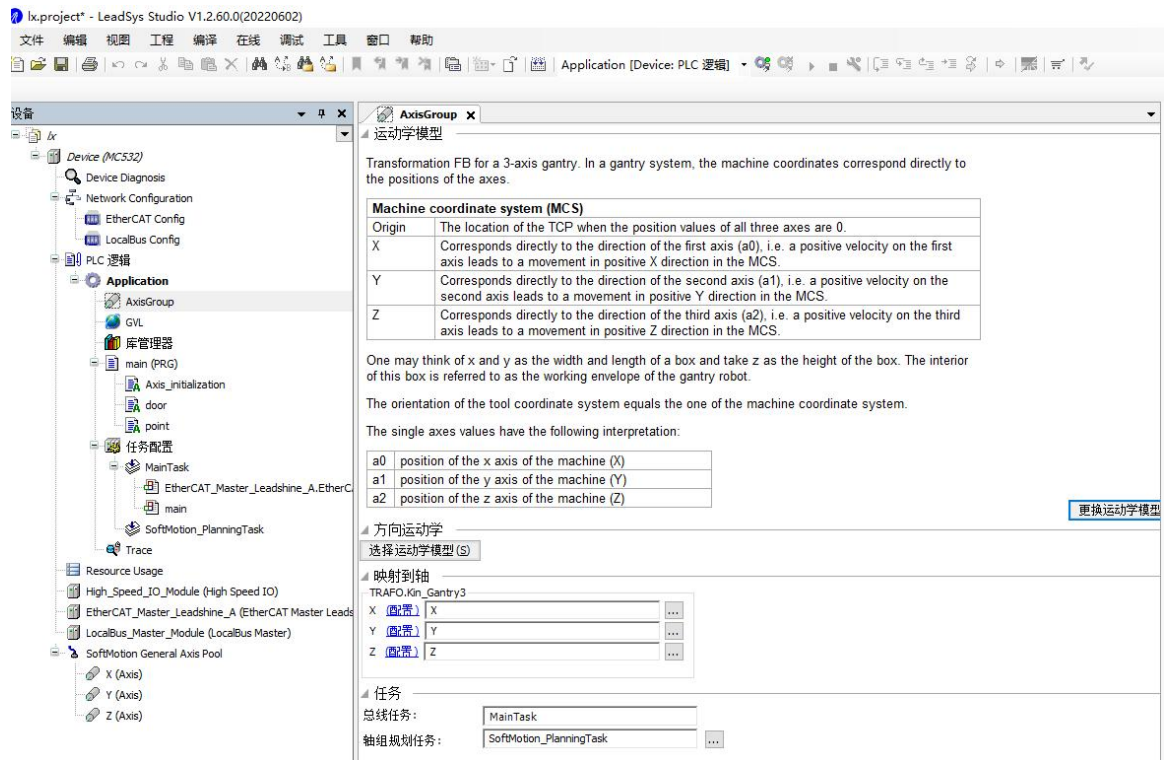


图 7.17 建立完成后轴组界面

7.4.轴组插补指令的介绍和使用

表 7.1 轴组运动指令表

指令	指令名称	功能说明
MC_AddAxisToGroup	添加轴到轴组	将单独的轴添加到轴组之中
MC_RemoveAxisFromGroup	从轴组中移除轴	将轴从轴组中移除
MC_GroupEnable	启用轴组	用于启用指定的轴组
MC_GroupDisable	使轴组无效	指定的轴组变成无效状态, 让其它轴组指令无法调用该轴组进行运动控制

MC_GroupReset	轴组复位	用于解除轴组及轴的异常状态
MC_GroupSetPosition	设置轴组指令位置	设置轴组中各轴的指令位置
MC_UngroupAllAxes	解除轴组	这个指令的作用是将某个轴组所包含的轴全部移除，解散该轴组
SMC_GroupPower	轴组使能	使能轴组下的所有轴，等价于轴组下的所有轴调用 MC_Power 指令
SMC_GroupSaveContinueData	保存轴组继续运动数据	保存轴组运动中暂停后的位置数据信息
SMC_GroupReadSetPosition	读取轴组指令位置	读取轴组在指定坐标系下的当前指令位置
SMC_GroupReadSetVelocity	读取轴组的设置速度	读取轴组在指定坐标系下的设置速度
SMC_GroupReadSetAcceleration	读取轴组的设置加速度	读取轴组在指定坐标系下的设置加速度
MC_GroupReadActualPosition	读取轴组反馈位置	读取轴组在指定坐标系下的反馈位置
MC_GroupReadActualVelocity	读取轴组反馈速度	读取轴组在指定坐标系下的反馈速度
MC_GroupReadActualAcceleration	读取轴组反馈加速度	读取轴组在指定坐标系下的反馈加速度
SMC_GroupTargetPosition	读取轴组目标位置	读取轴组在指定坐标系下的目标位置
SMC_GroupConvertPosition	轴组位置转换	轴组中在输入坐标系下的位置信息转换为输出坐标系下的位置

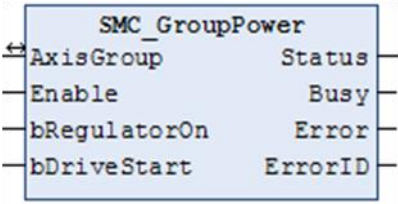
SMC_GroupGetContinuePosition	读取连续运动位置	读取轴组运动中中断时的位置
MC_GroupReadConfiguration	读取轴组配置参数	读取轴组包含的轴、数量等配置参数
MC_GroupReadStatus	读取轴组的当前运动状态	用于获取轴组的当前运动状态
MC_GroupReadError	读取轴组错误	获取轴组的错误信息
MC_SetCoordinateTransform	坐标系转换	用来转换不同参考坐标系的指令坐标
MC_SetDynCoordTransform	动态坐标系转换	指定的坐标系相对于 WCS 移动时，需要调用该指令实现坐标系转换
MC_GroupSetOverride	设置轴组超调值	轴组在 Moving 状态下，改变轴组的运动速度
SMC_GroupSetAncillaryAxisLimits	辅助轴限制	限制轴组的运动参数等
SMC_GroupSetAncillaryPathLimits	辅助路径限制	设置轴组的辅助动态路径限制
SMC_SetDynamicLimitFactors	动态限制因子设置	对轴组中所有轴的速度等运动参数做动态限制
MC_SetKinTransform	运动学坐标变换	设置轴组的运动学转换，从 ACS 坐标系到 MCS 坐标系
SMC_SetKinConfiguration	运动学坐标参数配置	配置机构的逆运动学参数
MC_GroupInterrupt	轴组中断	中断当前正在运动的轴组，可通过 MC_GroupContinue 指令继续执行未

		执行完的运动指令
MC_GroupContinue	轴组继续运行	解除轴组的中断状态, 继续执行未完成的指令
MC_GroupHalt	轴组暂停	用于暂停当前的轴组运动
MC_GroupStop	轴组停止	停止轴组的运动
MC_MoveCircularAbsolute	绝对圆弧插补	控制轴组执行绝对位置模式下的圆弧插补运动
MC_MoveCircularRelative	相对圆弧插补	控制轴组执行相对位置模式的圆弧插补运动
MC_MoveDirectAbsolute	绝对位置快速定位	控制轴组内所有轴各自以指定速度运行到绝对位置终点
MC_MoveDirectRelative	相对位置快速定位	控制轴组内所有轴各自以指定速度运行到相对位置的终点
MC_MoveLinearAbsolute	绝对位置直线插补	控制轴组在指定坐标系下的绝对位置模式的直线插补运动
MC_MoveLinearRelative	相对位置直线插补	控制轴组在指定坐标系下的相对位置模式的直线插补运动
SMC_GroupEnableResumeAfterError	错误复位后重启轴组运动	恢复因报错而被中断的轴组状态
SMC_GroupJog	轴组点动	控制轴组在指定坐标系下进行 Jog 运动
SMC_GroupWait	轴组运动延时	设定轴组的延时等待

表 7.1 包含了所有轴组的运动指令, 因本次连续插补只用到了轴组使能、启用轴组、轴组转换坐标系、绝对位置直线插补和绝对位置圆弧插补的指令, 因此本篇重点介绍以上 5 个指令, 剩余指令的详细内容可参考我司配套的指令手册。

7.4.1. 轴组使能 SMC_GroupPower

表 7.2 轴组使能指令外观

指令	FB/FUN	图形模块	结构文本
SMC_Group Power	FB		<pre> SMC_GroupPower(AxisGroup: =, Enable: = , bRegulatorOn: =, bDriveStart: =, Status=>, Busy=>, Error=>, ErrorID=>); </pre>

轴组使能指令外观如表 7.2 所示其中 AxisGroup 代表的是使能的轴组名称，当 Enable 置为 TRUE 时，功能块有效，当 bRegulatorOn 置为 TRUE 时轴组进入使能状态，bDriveStart 置为 TRUE 时功能块关闭紧急处理功能，成功使能轴组后 Status 置为 TRUE，如果还没有完成时置为 FALSE，如果功能块还在运行没有完成时 Busy 置为 TRUE，如果完成置为 FALSE，如果功能块报错时 Error 为 TRUE，ErrorID 显示相对应的错误代码。

功能块的说明如表 7.3 所示

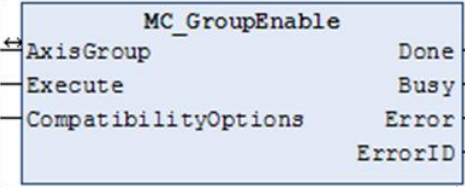
表 7.3 轴组使能功能块说明

VAR_IN _OUT	名称	类型	有效范围	初始值	描述
AxisGro up	轴组	AXIS_GROUP_RE F_SM3	-	-	指定的轴组

VAR_IN PUT					
Enable	有效	BOOL	TRUE-FALSE	FALSE	输入 TRUE 让模块开始运行
bRegulationOn	使能	BOOL	TRUE-FALSE	FALSE	设置 TRUE 让轴组进入使能状态
bDriveStart	驱动启用	BOOL	TRUE-FALSE	FALSE	设置为 TRUE 让功能块关闭紧急停止处理
VAR_OUTPUT					
Status	完成	BOOL	TRUE-FALSE	FALSE	输出 TRUE 则轴组能够开始运动
Busy	执行中	BOOL	TRUE-FALSE	FALSE	当模块被调用时为 TRUE
Error	报错	BOOL	TRUE-FALSE	FALSE	功能块执行错误
ErrorID	错误码	SMC_ERROR	-	0	错误代码指示，见 SMC_Error

7.4.2. 启用轴组 MC_GroupEnable

表 7.4 启用轴组指令外观

指令	FB/F UN	图形模块	结构文本
MC_GroupEnable	FB		<pre> MC_GroupEnable(AxisGroup: = , Execute: = , CompatibilityOptions: = , Done=>, Busy=> , Error=> , ErrorID=>); </pre>

轴组使能指令外观如表 7.4 所示其中 AxisGroup 代表的是使能的轴组名称，当 Execute 置为 TRUE 时，启用功能块，当成功启用轴组后 Done 置为 TRUE 轴组的状态置为 Standby，如果还没有完成时置为 FALSE，如果功能块还在运行没有完成时 Busy 置为 TRUE，如果完成置为 FALSE，如果功能块报错时 Error 为 TRUE，ErrorID 显示相对应的错误代码。

功能块的说明如表 7.5 所示

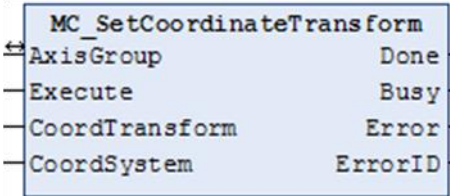
表 7.5 轴组使能功能块说明

VAR_IN_OUT	名称	类型	有效范围	初始值	描述
AxisGroup	轴组	AXIS_GROUP_REF _SM3	-	-	指定的轴组

VAR_IN_OUTPUT	名称	类型	有效范围	初始值	描述
VAR_INPUT					
Execute	启动	BOOL	TRUE-FALSE	FALSE	执行当前指令
CompatibilityOptions	兼容性选项	SMC_AXIS_GROUP_COMPATIBILITY_OPTIONS	-	-	为了兼容以前的版本而存在的参数
VAR_OUTPUT					
Done	完成	BOOL	TRUE-FALSE	FALSE	指令执行完成时为 TRUE
Busy	执行中	BOOL	TRUE-FALSE	FALSE	指令正在执行中为 TRUE
Error	报错	BOOL	TRUE-FALSE	FALSE	功能块执行错误
ErrorID	错误码	SMC_ERROR	-	0	错误代码指示, 见 SMC_Error

7.4.3. 坐标系转换 MC_SetCoordinateTransform

表 7.6 坐标系转换指令外观

指令	FB/FUN	图形模块	结构文本
MC_SetCoordinateTransform	FB		<pre> MC_SetCoordinate Transform(AxisGroup: = , Execute: = , CoordTransform: = , CoordSystem: = , Done=> , Busy=> , Error=> , ErrorID=>); </pre>

坐标系转换指令外观如表 7.6 所示其中 AxisGroup 代表需要转换坐标系的轴组，当 Execute 置为 TRUE 时，启用功能块，CoordTransform 代表待转换坐标系，CoordSystem 代表目标坐标系，当成功转换坐标系 Done 置为 TRUE，如果还没有完成时置为 FALSE，如果功能块还在运行没有完成时 Busy 置为 TRUE，如果完成置为 FALSE，如果功能块报错时 Error 为 TRUE，ErrorID 显示相对应的错误代码。

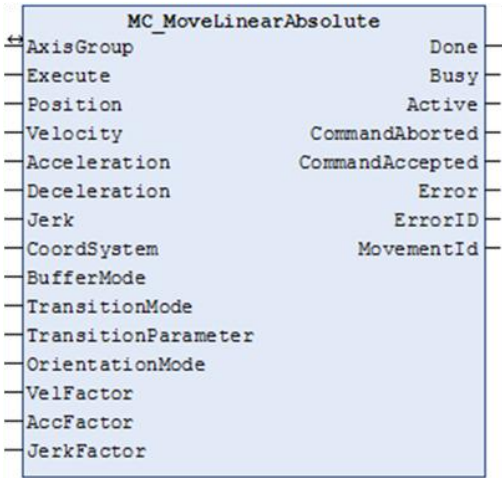
功能块的说明如表 7.7 所示

表 7.7 坐标系转换功能块说明

VAR_IN_OUT	名称	类型	有效范围	初始值	描述
AxisGroup	轴组	AXIS_GROUP_REF_SM3	-	-	指定的轴组
VAR_INPUT					
Execute	启动	BOOL	TRUE-FALSE	FALSE	执行当前指令
CoordTransform	待转换坐标系	BOOL	TRUE-FALSE	FALSE	
CoordSystem	目标坐标系	BOOL	TRUE-FALSE	FALSE	
VAR_OUTPUT					
Done	完成	BOOL	TRUE-FALSE	FALSE	指令执行完成时为 TRUE
Busy	执行中	BOOL	TRUE-FALSE	FALSE	指令正在执行中为 TRUE
Error	报错	BOOL	TRUE-FALSE	FALSE	功能块执行错误
ErrorID	错误码	SMC_ERROR	-	0	错误代码指示，见 SMC_Error

7.4.4. 绝对位置直线插补 MC_MoveLinearAbsolute

表 7.8 绝对位置直线插补

指令	FB/F UN	图形模块	结构文本
MC_MoveLinearAbsolute	FB		<pre> MC_MoveLinearAbsolute(AxisGroup: = , Execute: = , Position: = , Velocity: = , Acceleration: = , Deceleration: = , Jerk: = , CoordSystem: = , BufferMode: = , TransitionMode: = , TransitionParameter: = , OrientationMode: : = , </pre>

			<pre>VelFactor: = , AccFactor: = , JerkFactor: = , Done=> , Busy=> , Active=> , CommandAbort ed=> , CommandAccept ted=> , Error=> , ErrorID=> , MovementId=>);</pre>
--	--	--	---

表 7.8 是绝对位置直线插补指令为外观, AxisGroup 指定需要启用的轴组。Execute 设置为 TRUE 时轴组开始运行进行直线插补, Position 设置的是直线插补移动到的位置。设置相对应的参数 Velocity (速度)、Acceleration (加速度)、Deceleration (减速度)、Jerk (加加速度)、CoordSystem (参考坐标系)、BufferMode (缓存模式)、TransitionMode (拐角过渡模式)、TransitionParameter (拐角过渡参数)、OrientationMode (插补定位模式)、VelFactor (速度因子)、AccFactor (加速度因子)、JerkFactor (加加速度因子)。当功能块被成功调用和轴组成功被模块调用时 Active 和 CommandAccept 为 TRUE。功能块开始执行 Busy 置为 TRUE, 但是 Done 为 FALSE, 当直线插补完成后, Done 置为 TRUE。当功能块被其他命令打断时 CommandAborted 为 TRUE。当模块错误时 Error 为 TRUE, ErrorID 显示相对应的错误代码。MovementId 显示时是第几个运行。

功能块的说明如表 7.9 所示

表 7.9 绝对位置直线插补功能块

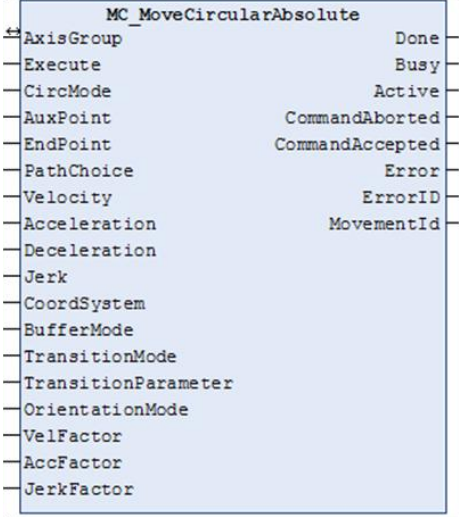
VAR_IN_OUT	名称	类型	有效范围	初始值	描述
AxisGroup	轴组号	AXIS_GROUP P_REF_SM3	-	-	指定的轴组
VAR_INPUT					
Execute	启动	BOOL	TRUE, FALSE	FALSE	上升沿触发指令运行
Position	位置	SMC-POS-REFERENCE	负数, 0, 正数	0	在指定参考坐标系中的绝对目标位置
Velocity	速度	LREAL	负数, 0, 正数	0	最大合速度[u/s]
Acceleration	加速度	LREAL	负数, 0, 正数	0	最大合加速度 [u/s ²]
Deceleration	减速度	LREAL	负数, 0, 正数	0	最大合减速度 [u/s ²]
Jerk	加加速度	LREAL	负数, 0, 正数	0	最大合加加速度 [u/s ²]
CoordSystem	参考坐标系	SMC_COORD_SYSTEM	SMC_COORD_SYSTEM	-	参考坐标系
BufferMode	缓存模式	MC-BUFFER-MODE	0-5	0	速度交接模式

TransitionMode	拐角过渡模式	MC_TRANSITION_MODE	TMNone , TMStart Velocity, TMCorn er Distance	0	拐角过渡模式
TransitionParameter	拐角过渡参数	ARRAY [0..(SMC_RC NST. MAX_TRAN S_PARAMS - 1)] OF LREAL	0, 正数	0	拐角过渡参数
OrientationMode	插补定位模式	SMC_ORIENTAT ION_MODE	GreatCircle, Axis	0	插补定位模式
VelFactor	速度因子	LREAL	0-1	1	速度因子
AccFactor	加速度因子	LREAL	0-1	1	加速度因子
JerkFactor	加加速度因子	LREAL	0-1	1	加加速度因子
VAR_OUTPUT					

Done	完成	BOOL	TRUE, FALSE	FALS E	如果回零已完成则 为 TRUE
Busy	执行中	BOOL	TRUE, FALSE	FALS E	当功能块执行还没 结束时为 TRUE。
Active	启动中	BOOL	TRUE, FALSE	FALS E	当模块被调用时为 TRUE
CommandAborted	运动中 断	BOOL	TRUE, FALSE	FALS E	如果该命令被终止 则为 TRUE。
CommandAccepted	运动接 收	BOOL	TRUE, FALSE	FALS E	当模块成功调用轴 组时为 True
Error	错误	BOOL	TRUE, FALSE	FALS E	在功能块内部发生 错误的信号。
ErrorID	错误代 码	SMC_ERRO R		0	错误指示，见 SMC_Error。
MovementId	运动标 志	SMC_Movem ent_Id	TRUE, FALSE	FALS E	当运动正在执行或 完成，则为 True

7.4.5. 绝对位置圆弧插补 MC_MoveCircularAbsolute

表 7.10 绝对位置圆弧插补指令外观

指令	FB/F UN	图形模块	结构文本
MC_MoveCircularAbsolute	FB		<pre> MC_MoveCircular Absolute(AxisGroup: = , Execute: = , CircMode: = , AuxPoint: = , EndPoint: = , PathChoice: = , Velocity: = , Acceleration: = , Deceleration: = , Jerk: = , CoordSystem: = , BufferMode: = , TransitionMode: = , TransitionParameter: = , OrientationMode: = , </pre>

			<pre> VelFactor: = , AccFactor: = , JerkFactor: = , Done=> , Busy=> , Active=> , CommandAborted => , CommandAccepte d=> , Error=> , ErrorID=> , MovementId=>); </pre>
--	--	--	--

表 7.10 为绝对位置圆弧插补指令的外观，AxisGroup 指定需要启用的轴组。Execute 设置为 TRUE 时轴组开始运行进行圆弧插补，CircMode（圆弧模式）、AuxPoint（辅助点）、EndPoint（终点）。设置下相对应的参数 Velocity（速度）、Acceleration（加速度）、Deceleration（减速度）、Jerk（加加速度）、CoordSystem（参考坐标系）、BufferMode（缓存模式）、TransitionMode（拐角过渡模式）、TransitionParameter（拐角过渡参数）、OrientationMode（插补定位模式）、VelFactor（速度因子）、AccFactor（加速度因子）、JerkFactor（加加速度因子）。当功能块被成功调用和轴组成功被模块调用时 Active 和 CommandAccept 为 TRUE。功能块开始执行 Busy 置为 TRUE，但是 Done 为 FALSE，当直线插补完成后，Done 置为 TRUE。当有其他命令打断时 CommandAborted 为 TRUE。当模块错误时 Error 为 TRUE，ErrorID 显示相对应的错误代码。MovementId 显示时是第几个运行的。

表 7.11 为绝对位置圆弧插补指令说明

表 7.11 绝对位置圆弧插补指令说明

VAR_IN_OUT	名称	类型	有效范围	初始值	描述
AxisGroup	轴组号	AXIS_GROUP_REF_SM3	-	-	指定的轴组
VAR_INPUT					
Execute	启动	BOOL	TRUE-FALSE	FALSE	上升沿触发指令运行
CircMode	圆弧模式	SMC-CIRC-MODE	0-2	0	指定圆弧插补的方法
AuxPoint	辅助点	SMC-POS-REF	负数, 0, 正数	0	辅助点位置
EndPoint	终点	SMC-POS-REF	负数, 0, 正数	0	终点位置
Velocity	速度	LREAL	负数, 0, 正数	0	最大合速度[u/s], 只能输入正数
Acceleration	加速度	LREAL	负数, 0, 正数	0	最大合加速度[u/s ²],
Deceleration	减速度	LREAL	负数, 0, 正数	0	最大合减速度[u/s ²],
Jerk	加加速度	LREAL	负数, 0, 正数	0	最大合加加速度[u/s ²],
CoordSystem	参考坐标系	SMC_COORD_SYSTEM	SMC_COORD_SYSTEM	-	参考坐标系

BufferMode	缓存模式	MC-BUFFER-MODE	0-5	0	速度交接模式
TransitionMode	拐角过渡模式	MC_TRANSITION_MODE	TMNone, TMStartVelocity, TMCornerDistance	0	拐角过渡模式
TransitionParameter	拐角过渡参数	ARRAY [0..(SMC_RCNST. MAX_TRANS_PARAMS - 1)] OF LREAL	0, 正数	0	拐角过渡参数
OrientationMode	插补定位模式	SMC_ORIENTATION_MODE	GreatCircle, Axis	0	插补定位模式
VelFactor	速度因子	LREAL	0-1	1	速度因子，每个轴的最大速度乘以这个速度因子，数值在[0, 1]之间
AccFactor	加速度因子	LREAL	0-1	1	加速度因子，每个轴的最大加速度乘以这个加速度因子，数值在[0, 1]之间
JerkFactor	加加	LREAL	0-1	1	加加速度因子，每

	速度因子				个轴的最大加加速度乘以这个加加速度因子，数值在[0, 1] 之间
VAR_OUTPUT					
Done	完成	BOOL	TRUE, FALSE	FALSE	如果回零已完成则为 TRUE
Busy	执行中	BOOL	TRUE-FALSE	FALSE	当功能块执行还没结束时为 TRUE。
Active	启动中		TRUE-FALSE	FALSE	当轴正在被这个模块调用 时为 TRUE
CommandAborted	运动中中断	BOOL	TRUE-FALSE	FALSE	如果该命令已被其他命令终止则为 TRUE。
CommandAccepted	运动接收	BOOL	TRUE-FALSE	FALSE	当模块成功调用轴组时为 True
Error	错误	BOOL	TRUE-FALSE	FALSE	在功能块内部发生错误的信号。
ErrorID	错误代码	SMC_ERROR		0	错误指示，见 SMC_Error。
MovementId	运动标志	SMC_Movement_Id	TRUE-FALSE	FALSE	当运动正在执行或完成，则为 True

7.5.连续插补插补中各个参数的介绍

7.5.1. 轴组状态机的介绍 SMC_AXIS_GROUP_STATE

轴组状态类型（SMC_AXIS_GROUP_STATE）是一种枚举类型共有六种状态其类型如表 7.12 所示，遵循了 PLCOpen Part4 的标准，如图 7.18 所示。

表 7.12 轴组状态枚举类型

名称	含义
Disabled	轴组无效
Standby	准备好
Moving	运动中
Homing	回零中
Stopping	停止中
ErrorStop	报错停止

轴组的状态图描述了轴组的命令状态。它位于每个轴的状态图顶部（如第 1 部分和第 2 部分所定义）。当轴处于组状态时，每个轴的单轴状态图也处于活动状态。

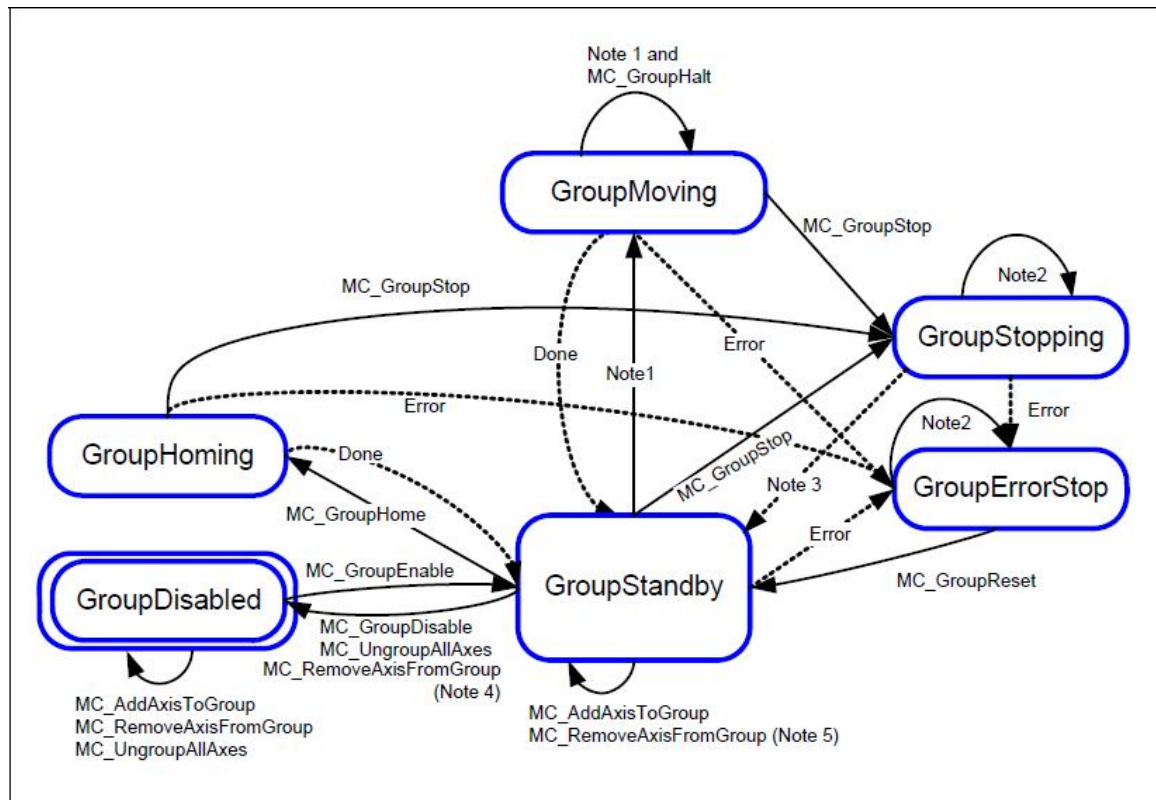


图 7.18 PLC Part4 标准

GroupDisabled 是通电时可以创建组的初始状态。发出 MC_GroupEnable 将离开此状态。

下一个状态是 GroupStandby。在此状态下，组启用，并且没有功能块可以控制组中的一个轴。在这种状态下，如果需要，还可以对组进行更改和驻留（状态 GroupHomeing）。

在状态 Group_Homing 状态中，可以定义为的一组轴的归位序列，由于各种电机的机械约束，需要以不同的方式进行定位

GroupStopping 是一种处理 MC_GroupStop 命令的特殊状态，该命令自动转换为状态

在 MC_GroupStop 中设置“Done”且“Execute”为 FALSE 时，立即执行 GroupStandby。

如果出现错误（在一个轴中），状态将更改为 GroupErrorStop，只能通过发出 MC_ResetGroup。

各个 Note 的含义：

- 1) Note1: 适用于所有非管理（移动）功能块。
- 2) Note2: 在 GroupErrorStop 或 GroupStopping 状态下，可以调用所有功能块，尽管它们不会被执行，但 GroupErrorStop 的 MC_GroupReset 和任何发生的错误除外,它们将生成到的转换分别为 GroupStandby 或 GroupErrorStop。
- 3) Note3: MC_GroupStop.DONE 而非 MC_GroupStop.EXECUTE。
- 4) Note4: 如果从组中删除最后一个轴，则过渡适用。
- 5) Note5: 当组不为空时，转换适用。
- 6) Note6: MC_GroupDisable 和 MC_UngroupAllAxes 可以在所有状态并将状态更改为 GroupDisabled。

轴组的各个状态可通过轴组界面看的到,首先打开 AxisGroup 打开轴组界面,再点击登录到,即可在线查看轴组状态。其流程图如图 7.19 所示。

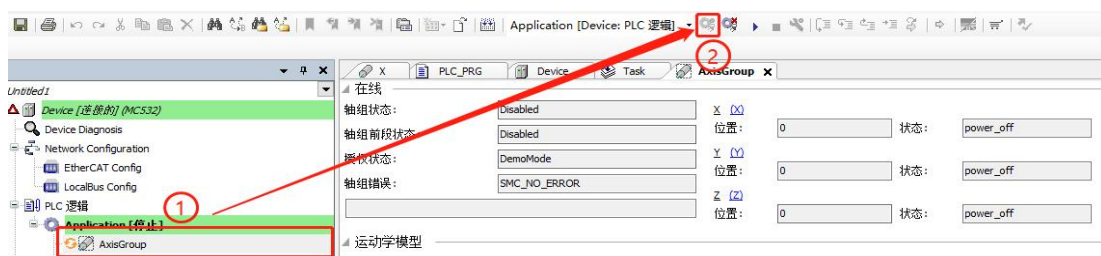


图 7.19 查看轴组状态流程图

7.5.2. 拐角过渡参数模式 MC_TRANSITION_MODE

拐角过渡模式（MC_TRANSITION_MODE）是一种枚举类型用于轴组的直线插补和圆弧插补中的 TransitionMode 参数里面，其类型如表格 7.13 所示。

表 7.13 拐角过渡模式枚举类型

名称	含义
TMNone	不设置拐角过渡
TMStartVelocity	基于速度的拐角过渡模式
TMCornerDistance	基于距离的拐角过渡模式

下面来分别介绍这三种不同的枚举类型。

1) TMNone（不设置拐角过渡）

不会修改运动块，也不会插入过渡曲线。这可能是缓冲模式中“mcBuffered”参数的唯一的拐角过渡模式。

因此当拐角过渡参数设置为 TMNone 时，任何具有 TMNone 参数的“mcBlending”缓冲模式都会产生与缓冲模式中“mcBuffered”相同的过渡方式，图 7.20 为当拐角过渡参数设置为 TMNone 路径示意图。

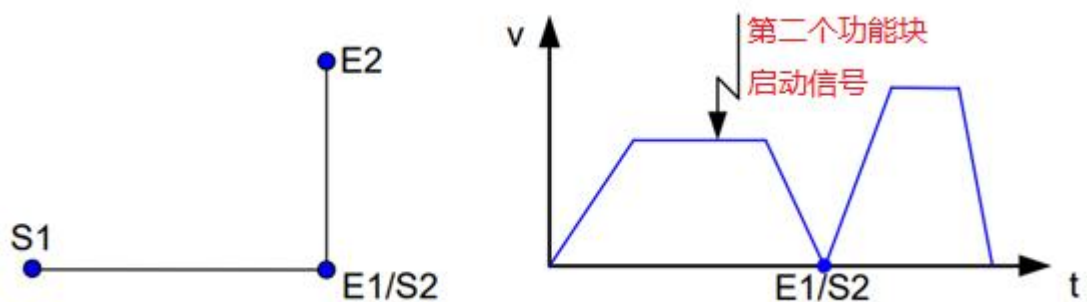


图 7.20 TMNone 拐角路径示意图

2) TMStartVelocity（基于速度的拐角过渡模式）

在这种类型的轮廓中，轮廓曲线的转角过渡距离（即图 7.21 中 E1'和 S2'红色曲线的距离）由预块的编程速度的指定最大百分比值“TPStartVelocity”定义，以便

启动过度曲线。缓冲区模式中的“mcBlending”模式不会针对此过渡模式进行计算评估。轮廓曲线上的速度不必是恒定的。例如：如果 $TPStartVelocity = 50\%$ ，则在减速期间速度达到预块中编程速度的 50% 之后开始生成轮廓曲线。图 7.21 为当拐角过度参数设置为 $TMStartVelocity$ 路径示意图。

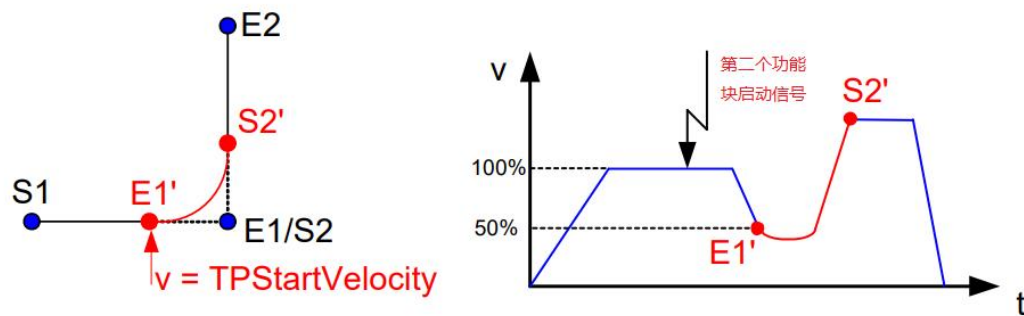


图 7.21 $TMStartVelocity$ 拐角路径示意图

其中的拐角过渡因子 $TransitionParameter[0]$ 作为一个因素，但不是时间上，而是与路径长度有关。值 1 表示减速斜坡的开始和加速斜坡的结束。值 0.5 表示正好介于两者之间。

在考虑所有约束作为半个元素长度、当前前瞻和截止距离之间的最大比率之后起作用。因此，小于 1 的值会导致 $A'B$ 和 BB' 的距离减小。大于 1 的值不一定会导致更长的距离，因为寻址的限制仍然有效。图 7.22 为当 $TransitionParameter[0]$ 等于 0、0.5、1 的时候速度拐角曲线的变化

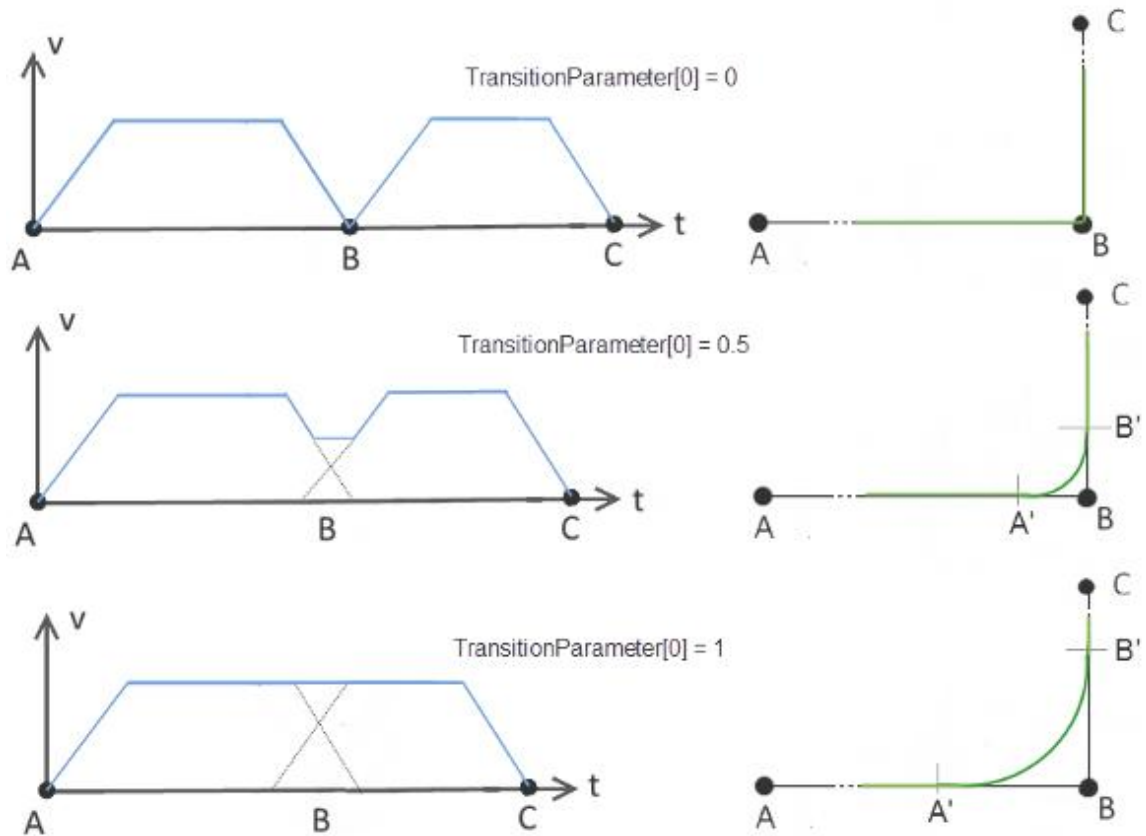


图 7.22 拐角过渡因子 $\text{TransitionParameter}[0]$ 等于不同的数值时的情况

3) TMCornerDistance (基于距离的拐角过渡模式)

如果原始轮廓的位置已知,用户可以明确指定前后功能块的距离(图 7.23 中的 d),通过该距离缩短边界运动块。

过渡速度由缓冲模式定义。然而,由于轮廓的曲率和轴给定的最大加速度,可能无法达到所需的速度。图 7.23 为当拐角过度参数设置为 TMCornerDistance 路径示意图。

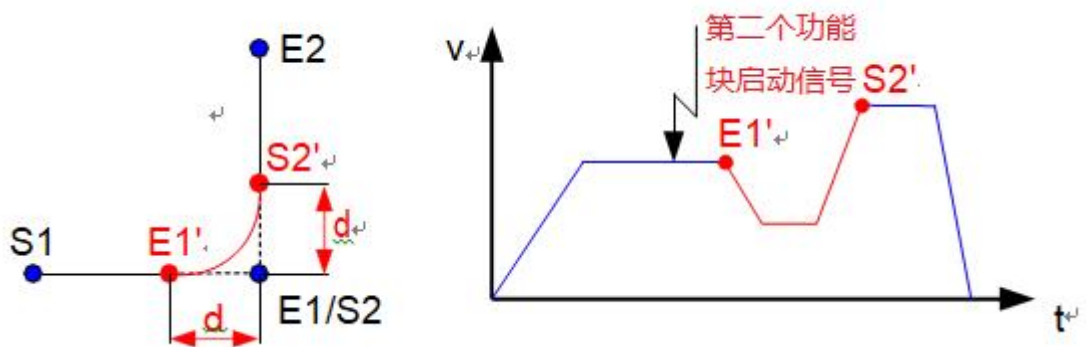


图 7.23 TMCornerDistance 拐角路径示意图

在基于距离的拐角过渡模式，TransitionParameter[0]是围绕 B 点的半径（第一次移动的终点 b/第二次移动的起点 a）如图 7.24 所示。

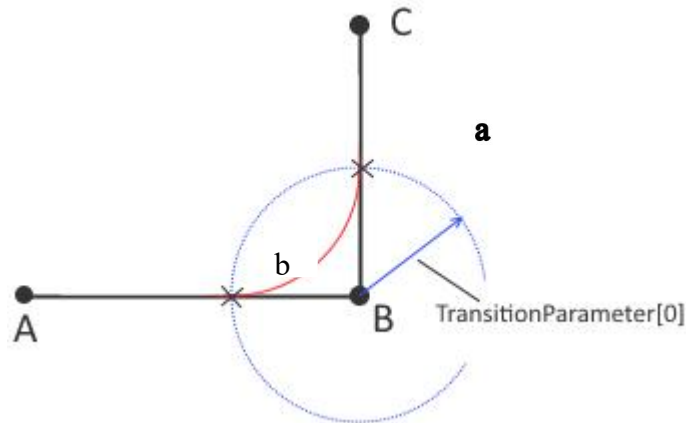


图 7.24 TransitionParameter[0]参数含义

混合时，路径的部分段在 A 点和 B 点之间以及 B 点和 C 点之间被切断。缓冲模式下的 CP 运动 Blending（不是 Aborting）以这样一种方式进行调整，即较长的截止段最多是较短段长度的五倍。在截断段的长度显着不同的情况下，会导致非常高的曲率，因此也会导致非常小的曲线半径。

第二个值，TransitionParameter[1]如图 7.25,代表着拐角过渡的半径。

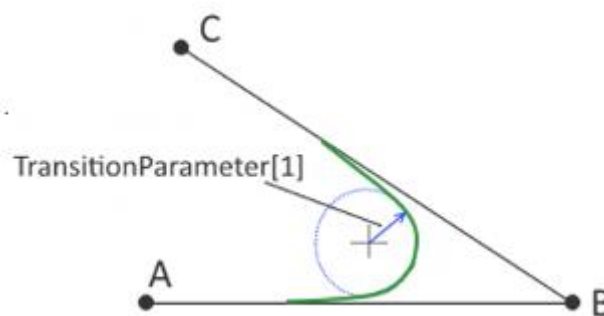


图 7.25 TransitionParameter[1]参数含义

仅在 CP 运动（连续插补运动）的情况下有效。该值确定可能的混合元素的最小曲率半径。此外，最小曲率半径是在内部计算的。这被选择为使得混合元素可以以最大允许路径速度的至少百分之一穿过。元件的实际曲率半径不得小于上述两个曲率半径中的最大值。否则，不会有任何混合，而是在运动之间停止移动。

7.5.3. 速度交接模式 MC_BUFFER_MODE

轴组运动速度交接模式。

对于同一个轴组，当轴组在运动过程之中，可以调用其它轴组运动指令，改变轴组的运动状态。当轴组在前后两个运动指令之间进行切换的时候，通过设置新调用指令的 BufferMode 参数，可以选择 6 种速度交接模式，用于直线插补和圆弧插补中的 BufferMode 参数里面，表 7.14 列出了每种模式的说明。

表 7.14 速度交接模式类型

名称	含义
mcAborting (打断)	立即中断当前的运动并执行新的运动指令
mcBuffered (等待)	等待当前运动正常执行结束后，立即执行下一个指令的运动
mcBlendingLow (低速滤波处理)	等待当前运动到目标位置后，以两个指令之中较低的速度进行动作的切换
mcBlendingPrevious (当前速度滤波处理)	等待当前运动到目标位置后，以当前运动指令的速度进行动作的切换
mcBlendingNext (下个速度滤波处理)	等待当前运动到目标位置后，以下一个运动指令的速度进行动作的切换
mcBlendingHigh (高速滤波处理)	等待当前运动到目标位置后，以两个指令之中较高的速度进行动作的切换

下面分别介绍三种交接模式打断、等待、滤波处理：

缓冲模式为“打断”的意思是中止正在进行的运动，并立即启动新运动，第一个运动停止然后开始下一个动作。图 7.26 为缓冲模式“中止”的示例。

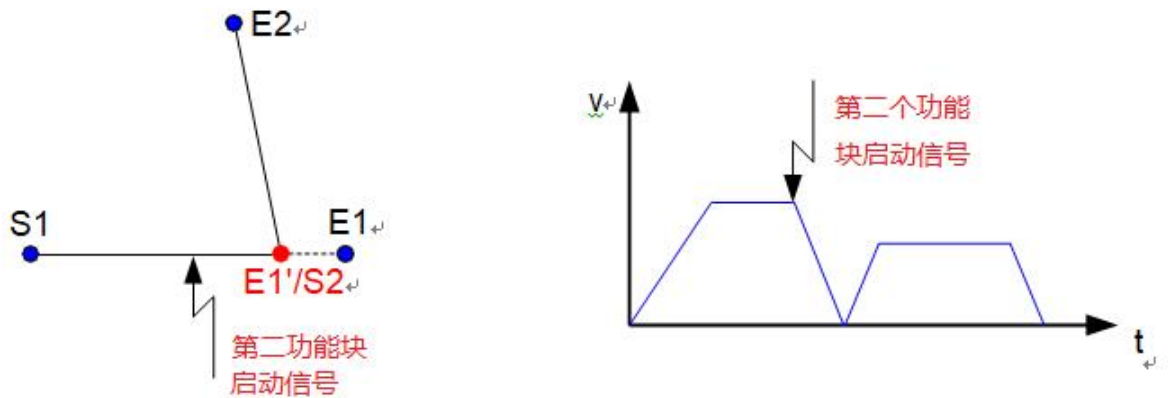


图 7.26 “mcAborting”示意图

缓冲模式为“等待”的意思是一旦上一个运动“完成”，下一个功能块就会影响轴组。没有混合，因此输入未计算 TransitionMode。图 7.27 为缓存模式为“等待”的示例。

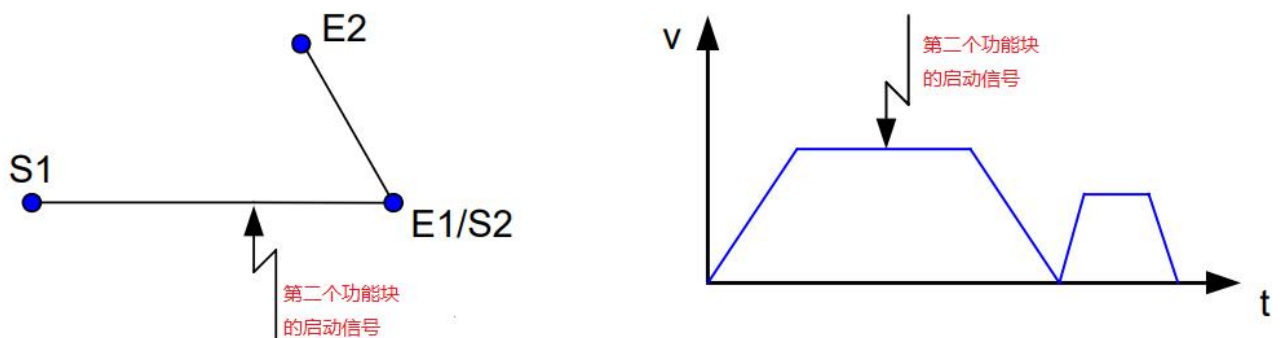


图 7.27 “mcBuffered”示意图

缓存模式为“滤波处理”的意思是当前和下一个运动功能块是混合的，因此轴组不会在运动之间停止。会根据指定的混合模式（BlendingLow、BlendingPrevious、BlendingNext、BlendingHigh）来进行混合速度。过渡轮廓由输入参数 TransitionMode 定义。图 7.28 展示为缓冲模式“BlendingNext”（下个速度滤波处理）的示例。

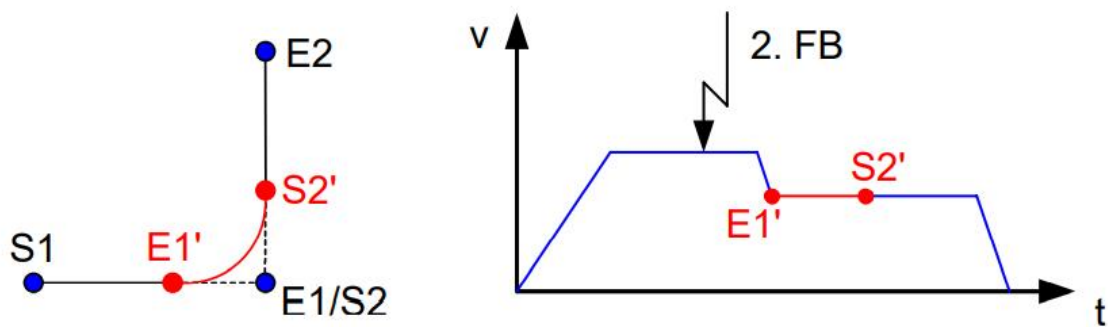


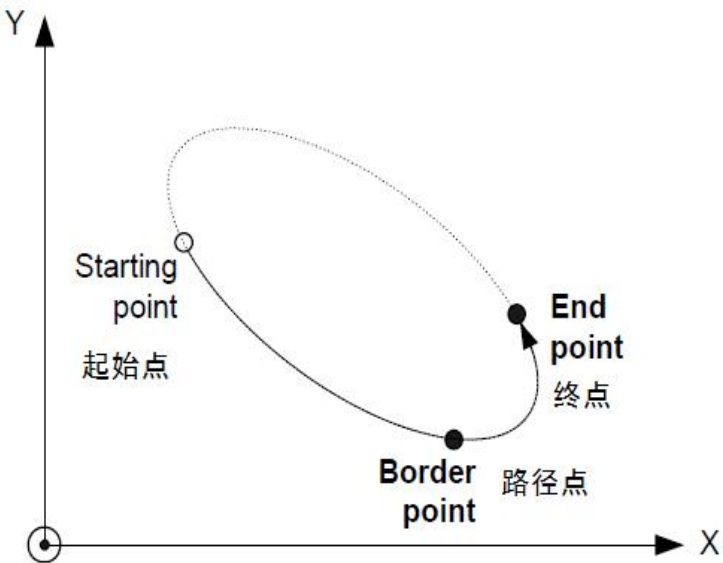
图 7.28 “BlendingNext”示意图

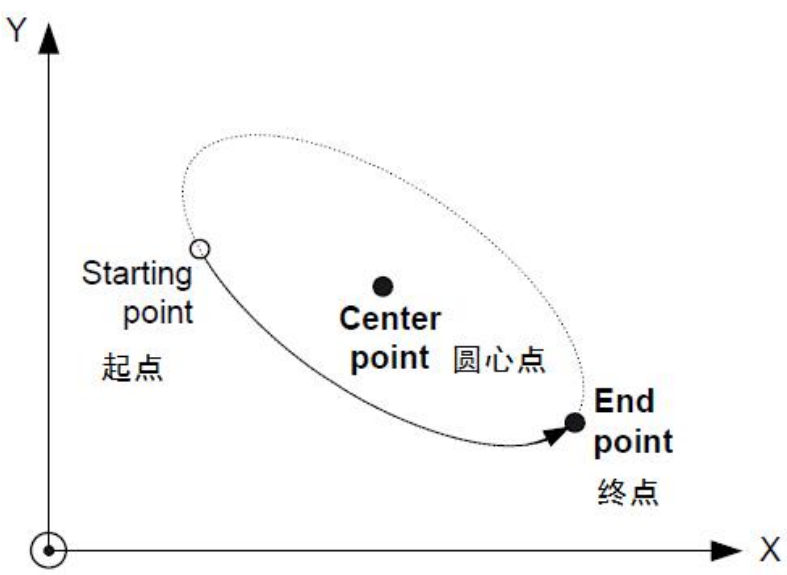
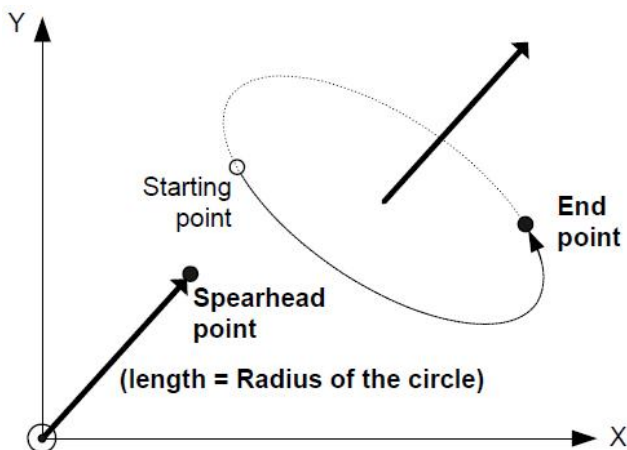
7.5.4. 圆弧插补模式 SMC_CIRC_MODE

圆弧插补模式，有三种方式分别为 BORDER、CENTER 和 RADIUS 如表 7.15 所示，定义圆弧插补使用何种方式进行，用于圆弧插补中的 CircleMode 参数里面。

表 7.15 圆弧插补模式

名称	说明
BORDER (三点圆弧模式)	三点圆弧：用户输入圆弧轨迹能经过的三点坐标去规划圆弧的起点、终点和路径点（辅助点）。 （Border 模式不考虑输入路径选择） 优点：输入点通常是圆弧轨迹能够达到的点，方便采集。 缺点：三点之间的角度一定是小于 2π。 注意：三个输入点一定是在同一个平面上，且三点不共一条直线。

	
CENTER (中心圆弧模式)	中心圆弧：需要输入圆弧的起点、终点和圆心点等参数，从而确定具体的圆弧轨迹。 注意：如果中心点到起点和终点的距离不完全相同（由于中心点编程的精度有限，这种情况经常发生），则将其投影到起点和终点的垂直双扇区上。一旦中心点太远（超过半径的 1%），就会返回错误。 缺点： 1) 在一个命令中限制$<2\pi$和$\neq \pi$的角度； 2) 由于与障碍物的碰撞，通常无法示教中心点。

	
RADIUS (半径圆弧模式)	终点半径模式：需要输入起点、终点和圆弧半径。 缺点： 1) 在单个命令中限制角度$\leq \pi$; 2) 必须计算垂直向量。 

7.5.5. 轴组方向模式 SMC_ORIENTATION_MOD

方向模式。用来设置在 CP 模式下，插补运动的方向，类型含义见表 7.16。

表 7.16 CP 模式下设置插补运动的方向

交接模式	说明
GreatCircle	沿着最短的路径从起始位置向目标位置运动，在这种模式下，即使起始位置和终点位置都在指定区域内，但实现的路径也可能会离开这个区域。
Axis	定位轴从起始位置到终点位置都在指定的区域内运动，不是所有的运动学变换都支持该种模式。

7.5.6. 工件坐标 SMC_POS_REF

表示 TCP（工件坐标系）的位置。

位置可以用笛卡尔坐标（X，Y，Z，A，B，C）或轴坐标（A0...A5）进行表示，如表 7.17 所示。

表 7.17 TCP（工件坐标系）位置的类型

名称	类型	注释
a	TRAFO.AXISPOS_REF	轴坐标系
c	MC_COORD_REF	笛卡尔位置
v	ARRAY [0..(SMC_RCNST.MAX_AXES - 1)] OF LREAL	数组数值，根据最终使用的坐标系来确定

7.5.7. 轴组位置关系 SMC_COORD_SYSTEM

用来表示轴组位置关系的坐标系，具体的坐标系名称以及说明见表 7.18。

表 7.18 轴组坐标系的名称和说明

名称	说明
----	----

名称	说明
ACS	轴坐标系：不属于笛卡尔坐标系，坐标系的维度与轴的数量相等，每个坐标对应轴组中的一个轴
MCS	机器坐标系，坐标系与轴组所对应的运动学模型相关，通过运动学变换定义了 MCS 的位置和方向。
WCS	世界坐标系，是所有轴组的通用坐标系。
PCS_1	产品坐标系 1，依据产品来确定，可能是静态的，也可能是动态的。
PCS_2	产品坐标系 2，依据产品来确定，可能是静态的，也可能是动态的。
TCS	工具坐标系，当使用了 MoveDirect-， MoveLinear- 或 MoveCircular-等 FB 模块时，位置数据与轴组在开始移动后的位置和方向都有关系，因此没有绝对位置和相对位置的区别。当调用的是其它的指令时（如 SMC_GroupConvertPosition），TCS 表示轴组在当前循环周期启动后的位置。

7.5.8. 工件坐标转换成笛卡尔坐标

SMC_POS_REF 包含了三种类型名称，故此可以定义成三种类型，见表 7.15 所示

在轴组直线插补和圆弧插补中 AUXpoint、ENDpoint 和 POSITIOIN 的参数变量类型为 SMC_POS_REF（工件坐标），因此需要将笛卡尔坐标系生成为工件坐标系。

示例如下

```
X1: lreal;
```

```
Y1: lreal;
```

```
Z1: lreal;
```

```
//定义 X1、Y1、Z1 为长实数
```

```
P1: SMC_POS_REF//定义 P1 为工件坐标（SMC_POS_REF）
```

```
P01: MC_COORD_REF//定义 P01 为笛卡尔坐标  
P01.X: =X1;  
P02.Y: =Y1;  
P03.Z : =Z1;  
//令 P01 点中的 XYZ 参数等于 X1、Y1、Z1 为长实数  
P1.C: =P01; //令工件坐标中的笛卡尔位置类型等于定义的笛卡尔坐标
```

7.6.插补子程序的编写和调用程序

本章节通过直线插补、圆弧插补和连续插补这三个例程的编写和调用来说明轴组运动的应用方法

7.6.1. 轴组初始化、启用轴组和坐标系的转换

第一步:

先建立轴组选择运动学模型 TRAFO.Kin_Gantry2（详细方法见第三章）映射好相对应的轴和规划好相对应的轴组任务，选择右击 Application 建立一个 POU 如图 7.29 所示，在图 7.30 中红色框选择的位置中，将名称设置为 main（主程序）和选择实现语言为结构化文本（ST），再右击 main 找到添加对象，添加一个动作如图 7.31 所示，并在图 7.32 红色框选择的位置，将名称设置为 Axis_initialization(轴初始化)和选择实现语言为结构化文本（ST）。

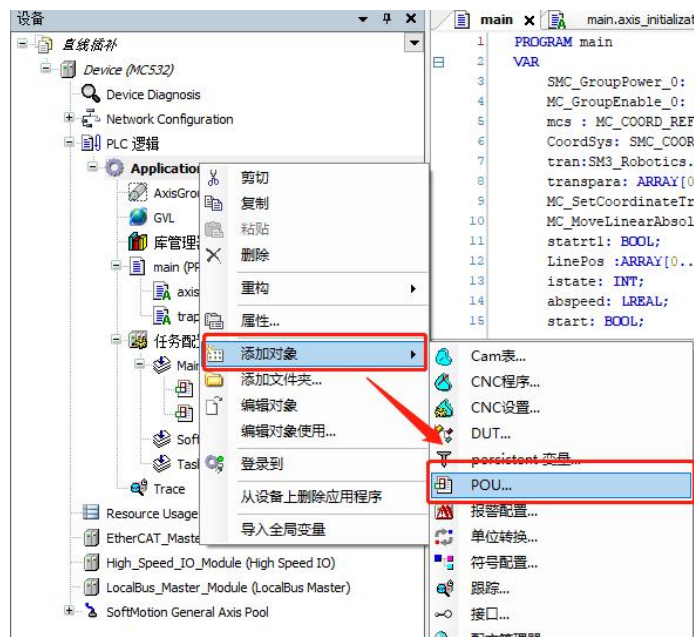


图 7.29 建立 POU 的方式



图 7.30 命名 main（主程序）和实现语言界面

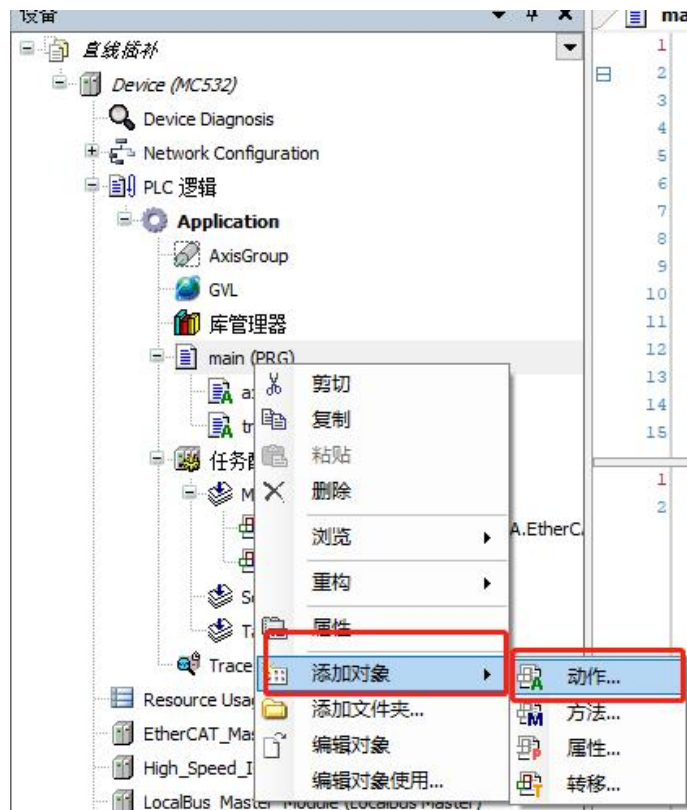


图 7.31 添加动作子程序界面

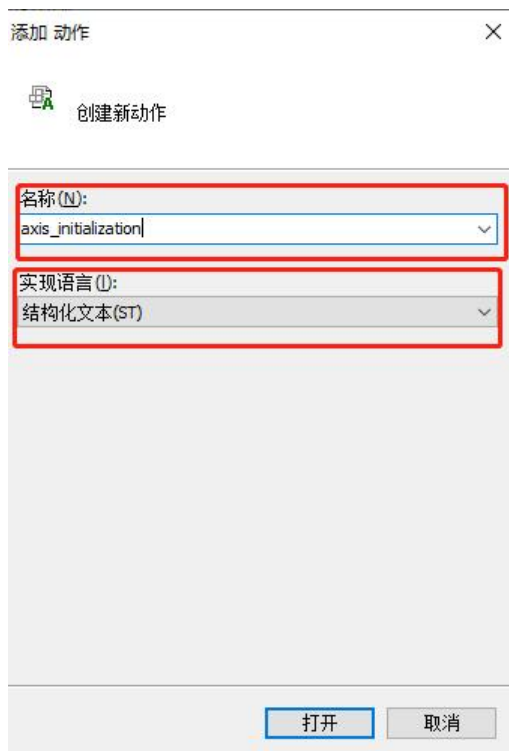


图 7.32 子程序命名和实现语言界面

第二步:

随后在 main（主程序）中变量定义去对功能块进行实例化和定义相对应的轴组坐标系，代码如下。

```
PROGRAM main
VAR
.....
    SMC_GroupPower_0: SMC_GroupPower;//实例化轴组使能模块
    MC_GroupEnable_0: MC_GroupEnable;//实例化启用轴组模块
    mcs : MC_COORD_REF;//定义坐 mcs 为笛卡尔坐标
    CoordSys: SMC_COORD_SYSTEM := SMC_COORD_SYSTEM.MCS;//定义
coordsys 为笛卡尔坐标系
    MC_SetCoordinateTransform_0: MC_SetCoordinateTransform;//实例化转换坐
标系模块
.....
END_VAR
```

在主程序定义的下方调用子程序

```
Axis_initialization();
```

Leadsys 界面如图 7.33 所示。



图 7.33 leadsys 中主程序界面

随后打开子程序 Axis_initialization 编写轴组初始化的功能块（轴组使能；坐标系

转换；轴组启用），代码如下：

```
SMC_GroupPower_0(  
    AxisGroup:=axisgroup,Enable:=TRUE,          bRegulatorOn:=TRUE,  
    bDriveStart:=TRUE,  
    Status=> , Busy=> , Error=> , ErrorID=> );  
MC_GroupEnable_0(  
    AxisGroup:=axisgroup, Execute:=TRUE, CompatibilityOptions:= ,  
    Done=> , Busy=> , Error=> , ErrorID=> );  
MC_SetCoordinateTransform_0(  
    AxisGroup:=axisgroup,      Execute:=TRUE,      CoordTransform:=mcs,  
    CoordSystem:=CoordSys,  
    Done=> , Busy=> , Error=> , ErrorID=> );
```

按照以上的顺序即可正常初始化轴组。

7.6.2. 直线插补例程

本次直线插补通过绘制一段梯形的轨迹曲线来说明，其轨迹曲线如下图 7.34 所示

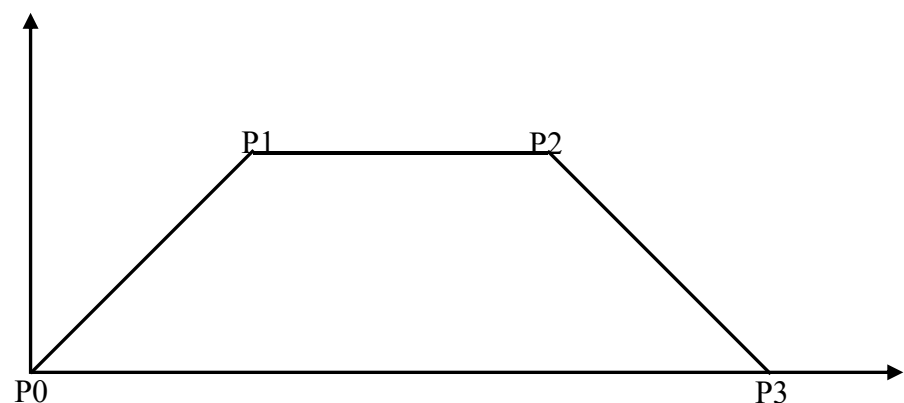


图 7.34 梯形轨迹曲线

从 P0（0，0）开始，直线插补到点 P1（200，600）；直线插补到点 P2（600，

600)，直线插补到点 P3（800，0）。

建立的轴组与 6.1 的轴组类型方式一致。

先在 main（主程序）中实例化功能块和定义变量。代码如下。

PROGRAM main

VAR

.....

SMC_GroupPower_0: SMC_GroupPower;//实例化轴组使能模块

MC_GroupEnable_0: MC_GroupEnable;//实例化启用轴组模块

mcs : MC_COORD_REF;//定义坐 mcs 为笛卡尔坐标

CoordSys: SMC_COORD_SYSTEM := SMC_COORD_SYSTEM.MCS;//

定义 coordsys 为笛卡尔坐标系

MC_SetCoordinateTransform_0: MC_SetCoordinateTransform;//实例化转

换坐标系模块

tran:SM3_Robotics.MC_TRANSITION_MODE:=MC_TRANSITION_MO

DE.TMStartVelocity;

transpara: ARRAY[0..(SMC_RCNST.MAX_TRANS_PARAMS-1)] OF

LREAL :=[20,1];//拐角过渡因子

MC_MoveLinearAbsolute:ARRAY[0..4] OF MC_MoveLinearAbsolute;//

实例化直线插补功能块

LinePos :ARRAY[0..3] OF SMC_POS_REF := [(0, (c := (X := 200, Y :=

600, Z := 0)), (c := (X := 600, Y := 600)), (c := (X := 800))];//直线插补点

start: BOOL;//定义启动变量

.....

END_VAR

在主程序定义的下方调用子程序

Axis_initialization();//轴组初始化子程序

Trapezoid();//直线插补子程序

然后在右击主程序建立相对应的子程序 trapezoid（详细方法见 6.1 轴组初始

化建立子程序的方式），随后点开子程序开始编写，代码如下

//P0 点插补

```
MC_MoveLinearAbsolute[0](  
    AxisGroup:=axisgroup,Execute:=start,Position:=LinePos[0],Velocity:=3000,  
    Acceleration:=1500,  
    Deceleration:=1500,Jerk:=500,CoordSystem:=CoordSys,  
    BufferMode:=MC_BUFFER_MODE.BlendingLow,TransitionMode:=tran,  
    TransitionParameter:=transpara, OrientationMode:= ,  
    VelFactor:= , AccFactor:= , JerkFactor:= , Done=> , Busy=> ,  
    Active=> , CommandAborted=> , CommandAccepted=> ,  
    Error=> , ErrorID=> , MovementId=> );
```

//P1 点插补

```
MC_MoveLinearAbsolute[1](  
    AxisGroup:=axisgroup,Execute:=start,Position:=LinePos[1],Velocity:=3000,  
    Acceleration:=1500,  
    Deceleration:=1500,Jerk:=500,CoordSystem:=CoordSys,  
    BufferMode:=MC_BUFFER_MODE.BlendingLow,TransitionMode:=tran,  
    TransitionParameter:=transpara, OrientationMode:= ,  
    VelFactor:= , AccFactor:= , JerkFactor:= , Done=> , Busy=> , Active=> ,  
    CommandAborted=> , CommandAccepted=> , Error=> , ErrorID=> ,  
    MovementId=> );
```

//P2 点插补

```
MC_MoveLinearAbsolute[2](  
    AxisGroup:=axisgroup,Execute:=start,Position:=LinePos[2],Velocity:=3000,  
    Acceleration:=1500,  
    Deceleration:=1500,Jerk:=500,CoordSystem:=CoordSys,  
    BufferMode:=MC_BUFFER_MODE.BlendingLow,TransitionMode:=tran,  
    TransitionParameter:=transpara, OrientationMode:= ,
```



```
VelFactor:= , AccFactor:= , JerkFactor:= , Done=> , Busy=> , Active=> ,  
CommandAborted=> , CommandAccepted=> , Error=> , ErrorID=> ,  
MovementId=> );
```

//P3 点插补

```
MC_MoveLinearAbsolute[3](  
    AxisGroup:=axisgroup,Execute:=start,Position:=LinePos[3],Velocity:=3000,  
    Acceleration:=1500,  
    Deceleration:=1500,Jerk:=500,CoordSystem:=CoordSys,  
    BufferMode:=MC_BUFFER_MODE.BlendingLow,TransitionMode:=tran,  
    TransitionParameter:=transpara, OrientationMode:= ,  
    VelFactor:= , AccFactor:= , JerkFactor:= , Done=> , Busy=> , Active=> ,  
    CommandAborted=> , CommandAccepted=>,Error=> , ErrorID=> ,  
    MovementId=> );
```

```
abspeed:=SQRT(X.fActVelocity*X.fActVelocity+Y.fActVelocity*Y.fActVelo
```

city);//合速度计算

轨迹曲线如图 7.35 所示其中蓝色代表 X 轴速度，绿色代表 Y 轴速度，浅蓝色代表 X 位移曲线，橙色代表 Y 轴位移曲线，灰色代表合速度曲线。

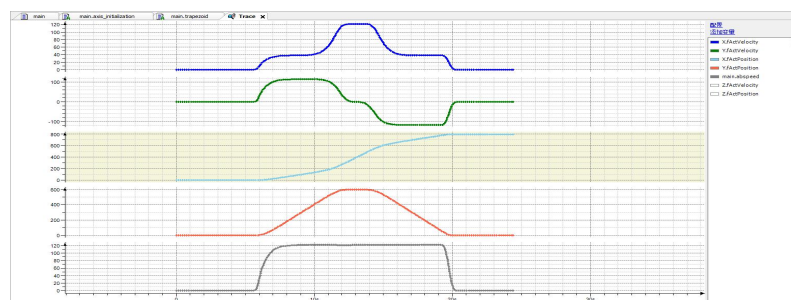


图 7.35 梯形轨迹速度位移曲线

直线插补例程目录数如图 7.36 所示。

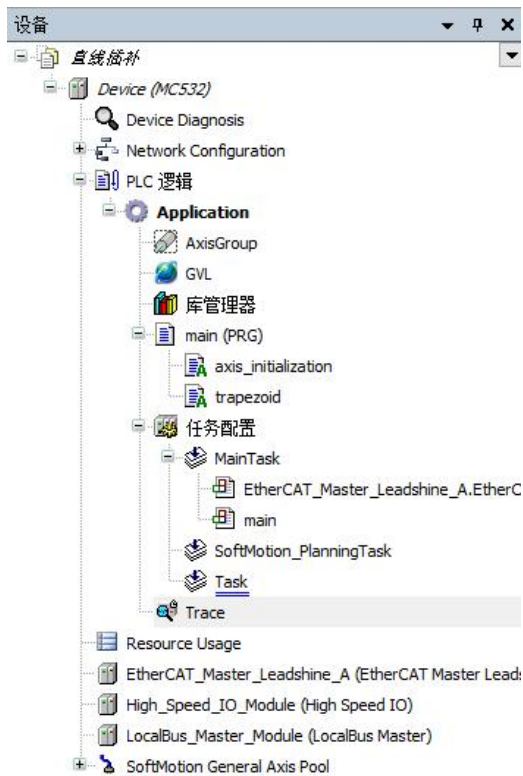


图 7.36 直线插补目录树

7.6.3. 圆弧插补例程

本次圆弧插补通过绘制一段圆形的轨迹曲线来说明，其轨迹曲线如下图 7.37 所示。

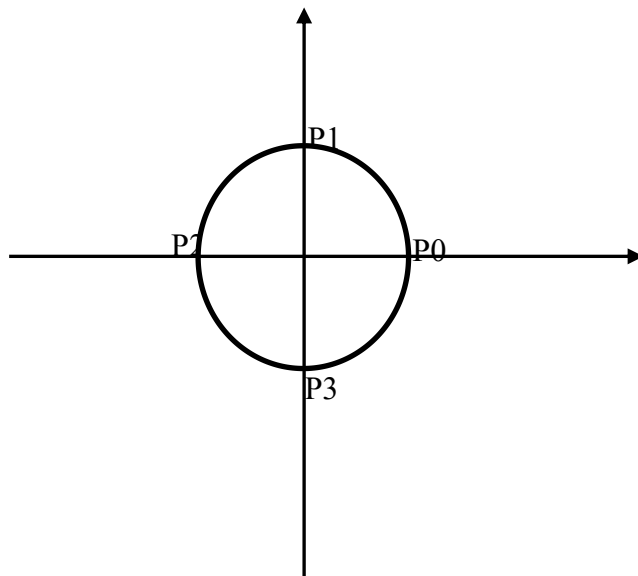


图 7.37 圆形轨迹曲线

从 P0 (180, 0) 开始, 圆弧插补到点 P1 (180, 0); 圆弧插补到点 P2 (-180, 0), 圆弧插补到点 P3 (0, -180), 最后圆弧插补回到 P0 (180, 0)。

建立的轴组与 6.1 的轴组类型方式一致。

先在 main (主程序) 中实例化功能块和定义变量。代码如下。

```
PROGRAM main
```

```
VAR
```

```
.....
```

```
SMC_GroupPower_0: SMC_GroupPower;//实例化轴组使能模块
```

```
MC_GroupEnable_0: MC_GroupEnable;//实例化启用轴组模块
```

```
mcs : MC_COORD_REF;//定义坐 mcs 为笛卡尔坐标
```

```
CoordSys: SMC_COORD_SYSTEM := SMC_COORD_SYSTEM.MCS;//
```

定义 coordsys 为笛卡尔坐标系

```
MC_SetCoordinateTransform_0: MC_SetCoordinateTransform;//实例化转
```

换坐标系模块

```
tran:SM3_Robotics.MC_TRANSITION_MODE:=MC_TRANSITION_MO
```

DE.TMStartVelocity;

```
transpara: ARRAY[0..(SMC_RCNST.MAX_TRANS_PARAMS-1)] OF
```

LREAL :=[20,1];//拐角过渡因子

```
MC_MoveCircularRelative:ARRAY[0..4] OF MC_MoveCircularRelative;//
```

实例化直线插补功能块

```
cPos :ARRAY[0..3] OF SMC_POS_REF := [(c := (X := 180)), (c := (X :=  
0, Y := 180, Z := 0)), (c := (X := -180)), (c := (Y := -180))];//圆弧插补点
```

```
start: BOOL;//定义启动变量
```

```
.....
```

```
END_VAR
```

在主程序定义的下方调用子程序

```
Axis_initialization();//轴组初始化子程序
```

```
circle();//直线插补子程序
```

然后在右击主程序建立相对应的子程序 **circle**（详细方法见 6.1 轴组初始化建立子程序的方式），随后点开子程序开始编写，代码如下

//P0 到 P1 的圆弧插补

```
MC_MoveCircularRelative[0](  
    AxisGroup:=axisgroup,Execute:=start1,CircMode:=0,AuxPoint:=cPos[0],  
    EndPoint:=cPos[1],PathChoice:=1,Velocity:=3000,Acceleration:=1500,  
    Deceleration:=1500,Jerk:=500,CoordSystem:=CoordSys,  
    BufferMode:=MC_BUFFER_MODE.BlendingLow,TransitionMode:=tran,  
    TransitionParameter:=transpara,OrientationMode:=,VelFactor:=,AccFactor:=  
    ,  
    JerkFactor:=,Done=>,Busy=>,Active=>,  
    CommandAborted=>,CommandAccepted=>,Error=>,ErrorID=>,MovementId  
d=> );
```

//P1 到 P2 的圆弧插补

```
MC_MoveCircularRelative[1](  
    AxisGroup:=axisgroup,Execute:=start1,CircMode:=0,AuxPoint:=cPos[1],  
    EndPoint:=cPos[2],PathChoice:=1,Velocity:=3000,Acceleration:=1500,  
    Deceleration:=1500,Jerk:=500,CoordSystem:=CoordSys,  
    BufferMode:=MC_BUFFER_MODE.BlendingLow,TransitionMode:=tran,  
    TransitionParameter:=transpara,OrientationMode:=,VelFactor:=,AccFactor:=  
    ,  
    JerkFactor:=,Done=>, Busy=>, Active=> ,CommandAborted=> ,  
CommandAccepted=> ,  
    Error=> , ErrorID=> , MovementId=> );
```

//P2 到 P3 的圆弧插补

```
MC_MoveCircularRelative[2](  
    AxisGroup:=axisgroup,Execute:=start1,CircMode:=0,AuxPoint:=cPos[2],  
    EndPoint:=cPos[3],PathChoice:=1,Velocity:=3000,Acceleration:=1500,
```

```
Deceleration:=1500,Jerk:=500,CoordSystem:=CoordSys,
BufferMode:=MC_BUFFER_MODE.BlendingLow,TransitionMode:=tran,
TransitionParameter:=transpara,OrientationMode:=,VelFactor:=,AccFactor:=
,
JerkFactor:=,Done=>, Busy=>, Active=>, CommandAborted=>,
CommandAccepted=>,
Error=>, ErrorID=>, MovementId=> );
//P3 到 P1 的圆弧插补
MC_MoveCircularRelative[3](
AxisGroup:=axisgroup,Execute:=start1,CircMode:=0,AuxPoint:=cPos[3],
EndPoint:=cPos[0],PathChoice:=1,Velocity:=3000,Acceleration:=1500,
Deceleration:=1500,Jerk:=500,CoordSystem:=CoordSys,
BufferMode:=MC_BUFFER_MODE.BlendingLow,TransitionMode:=tran,
TransitionParameter:=transpara,OrientationMode:=,VelFactor:=,AccFactor:=
,
JerkFactor:=,Done=>, Busy=>, Active=>, CommandAborted=>,
CommandAccepted=>,
Error=>, ErrorID=>, MovementId=> );
abspeed:=SQRT(X.fActVelocity*X.fActVelocity+Y.fActVelocity*Y.fActVelocity);//合速度计算
```

轨迹曲线如图 7.38 所示其中蓝色代表 X 轴速度，绿色代表 Y 轴速度，浅蓝色代表 X 位移曲线，橙色代表 Y 轴位移曲线，灰色代表合速度曲线。

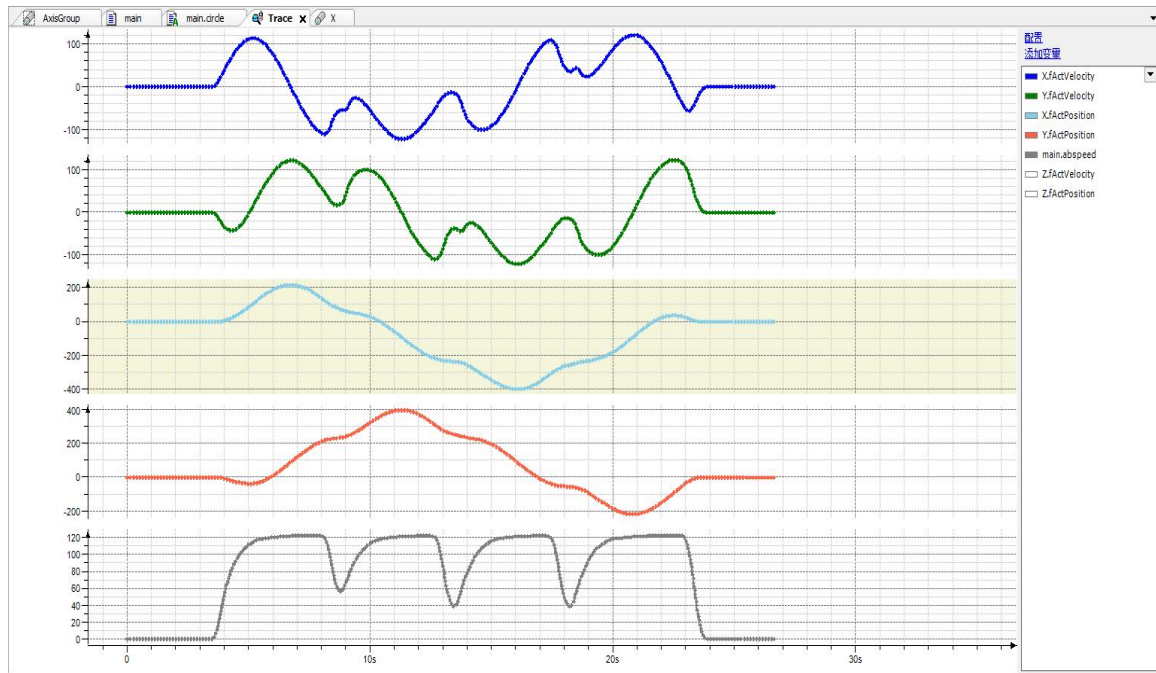


图 7.38 圆形轨迹速度和位移曲线

7.6.4. 连续插补例程

本次连续插补通过绘制一段门型轨迹来阐述连续插补，其轨迹曲线如下图 7.39 所示。

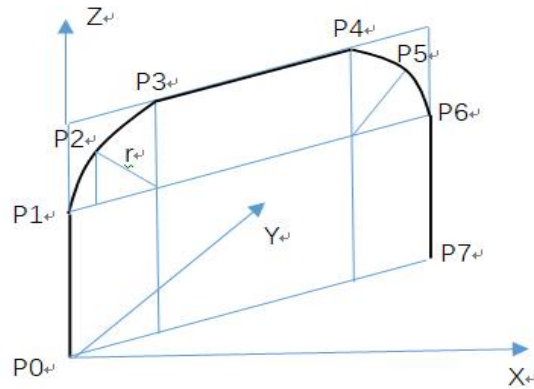


图 7.39 门型轨迹

已知 $P0$ 坐标为 $(X0, Y0, Z0)$ ， $P7$ 为 $(dx, dy, Z0)$ ；

设 Z 轴的高度为 $h1$ 圆弧半径为 $r0$ (用户可根据需要自行设置半径),由此可得

$P1$ 的坐标为 $(0+X0, 0+Y0, h1+Z0)$ ， $P6$ 的坐标为 $(dx, dy, h1)$ ；

$P2$ 的坐标为 $(m+ X0, n+Y0, h2+Z0)$ 和 $P5$ 的坐标为 $(dx-m+ X0, dy-n+Y0, h2+Z0)$ ，

$P3$ 的坐标为 $(p+ X0, q+Y0, h1+r+Z0)$ 和 $P4$ 的坐标为 $(dx-p+ X0, dy-q+Y0, h1+r+Z0)$ ，

其中：

$$k := (dy)/(dx);$$

$$L := r0 - r0 * \cos(\pi/4);$$

$$m := \sqrt{L * L / (1 + k * k)};$$

$$n := k * m;$$

$$h2 := h1 + r0 * \sin(\pi/4);$$

$$p := \sqrt{r0 * r0 / (1 + k * k)};$$

$$q := k * p;$$

先建立轴组选择运动学模型 TRAFO.Kin_Gantry3（详细方法见第三章）映射好相

对应的轴和规划好相对应的轴组任务。

先右击 Application 找到添加对象，选择全局变量列表，添加成功后在把最上面的灰色字{attribute 'qualified_only'}给删除掉，操作顺序图如图 7.40 所示。

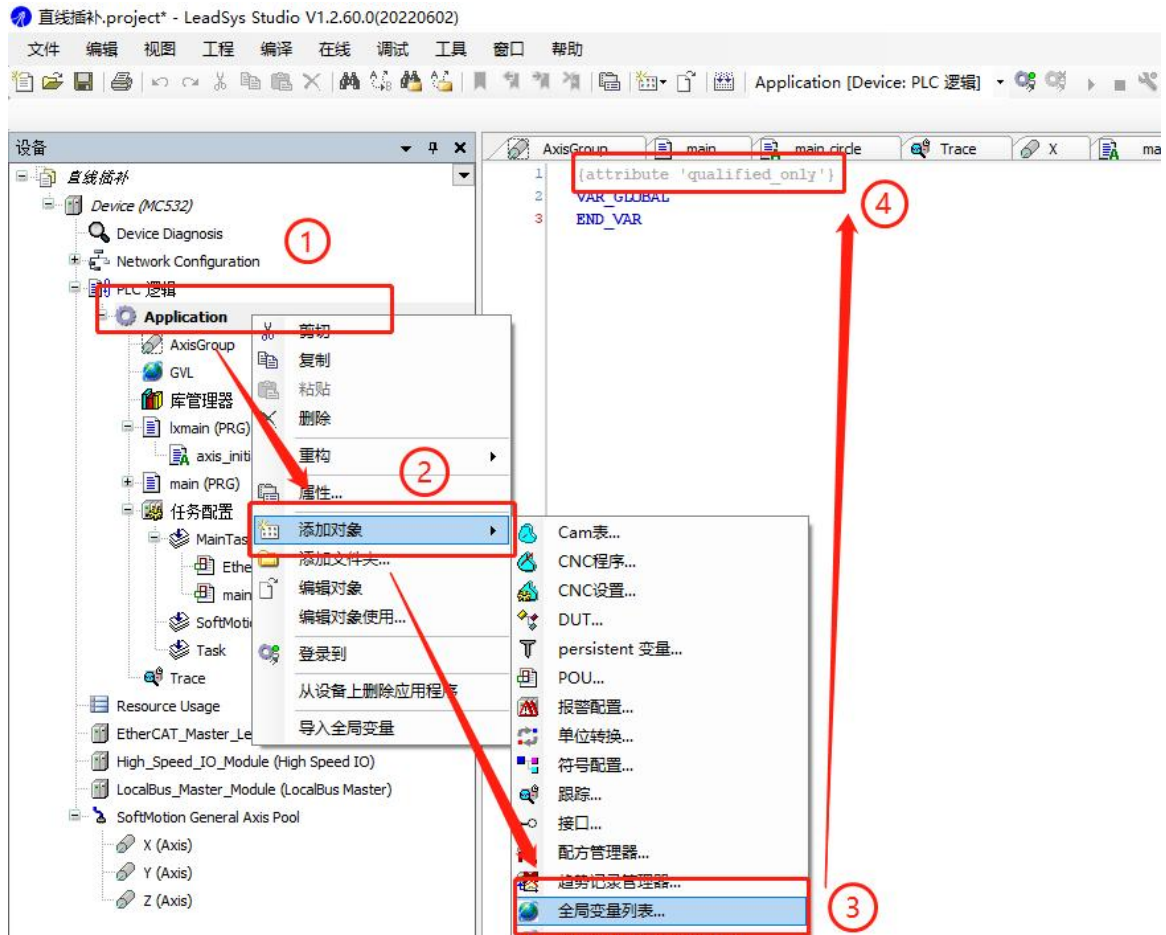


图 7.40 建立全局变量顺序图

下面为本次例程的详细代码

GVL（全局变量列表定义）代码如下：

VAR_GLOBAL

SMC_GroupPower_0: SMC_GroupPower;//实例化轴组使能模块

MC_GroupEnable_0: MC_GroupEnable;//实例化启用轴组模块

mcs : MC_COORD_REF;//定义坐 mcs 为笛卡尔坐标

CoordSys: SMC_COORD_SYSTEM := SMC_COORD_SYSTEM.MCS;//

定义 coordsys 为笛卡尔坐标系

MC_SetCoordinateTransform_0: MC_SetCoordinateTransform;// 实例化转

换坐标系模块

```
tran:SM3_Robotics.MC_TRANSITION_MODE:=MC_TRANSITION_MO  
DE.TMStartVelocity;  
  
transpara: ARRAY[0..(SMC_RCNST.MAX_TRANS_PARAMS-1)] OF  
LREAL :=[20,1];//拐角过渡因子  
  
MC_MoveLinearAbsolute:ARRAY[0..4] OF MC_MoveLinearAbsolute;  
MC_MoveCircularRelative:ARRAY[0..4] OF MC_MoveCircularRelative;  
  
start1:BOOL;  
  
px:ARRAY[0..7] OF LREAL;  
py:ARRAY[0..7] OF LREAL;  
pz:ARRAY[0..7] OF LREAL;  
Targetpiont:ARRAY[0..7] OF SMC_POS_REF;  
piont:ARRAY[0..7] OF MC_COORD_REF;  
  
m:LREAL;  
n:LREAL;  
dx:LREAL;  
dy:LREAL;  
q:LREAL;  
p:LREAL;  
k:LREAL;  
r0:LREAL;  
h1:LREAL;  
h2:LREAL;  
PI:LREAL;  
L:LREAL;  
L2:LREAL;  
k2:LREAL;  
r2:LREAL;
```

```
abspeed: LREAL;  
END_VAR
```

Main 主程序代码如下:

```
Axis_initialization();  
door();  
point();  
abspeed:=SQRT(X.fActVelocity*X.fActVelocity+y.fActVelocity*y.fActVelocity+  
z.fActVelocity*z.fActVelocity);//合速度计算
```

Axis_initialization (轴组初始化) 子程序代码如下:

```
SMC_GroupPower_0(  
    AxisGroup:=axisgroup,      Enable:=TRUE,      bRegulatorOn:=TRUE,  
bDriveStart:=TRUE,  
    Status=> , Busy=> , Error=> , ErrorID=> );  
MC_GroupEnable_0(  
    AxisGroup:=axisgroup, Execute:=TRUE, CompatibilityOptions:= ,  
    Done=> , Busy=> , Error=> , ErrorID=> );  
MC_SetCoordinateTransform_0(  
    AxisGroup:=axisgroup,      Execute:=TRUE,      CoordTransform:=mcs,  
CoordSystem:=CoordSys,  
    Done=> , Busy=> , Error=> , ErrorID=> );
```

Door (门型轨迹插补) 子程序代码如下

```
MC_MoveLinearAbsolute[0](  
    AxisGroup:=AxisGroup,Execute:=start1,Position:=Targetpion[1],Velocity:=  
    3000,  
    Acceleration:=1500,Deceleration:=1500,Jerk:=1000,CoordSystem:=CoordSy
```

```
s,      BufferMode:=SM3_Common.MC_BUFFER_MODE.BlendingLow,
TransitionMode:=tran,
TransitionParameter:=transpara,OrientationMode:=,VelFactor:=
,
AccFactor:= ,
JerkFactor:=,Done=>,Busy=> ,Active=>,CommandAborted=>,
CommandAccepted=>,Error=>,ErrorID=> , MovementId=> );
//P1 点的直线插补
```

```
MC_MoveCircularRelative[0](
    AxisGroup:=axisgroup,Execute:=start1,CircMode:=SMC_CIRC_MODE.BO
RDER,
    AuxPoint:=Targetpiont[2],EndPoint:=Targetpiont[3],PathChoice:=0,Velocity:
=3000,
    Acceleration:=1500,Deceleration:=1500,Jerk:=1000,CoordSystem:=CoordSy
s,
    BufferMode:=SM3_Common.MC_BUFFER_MODE.BlendingLow,Transitio
nMode:=tran,
    TransitionParameter:=transpara,OrientationMode:=,VelFactor:=,AccFactor:=,
    JerkFactor:=,Done=>,Busy=>,Active=>,
    CommandAborted=>,CommandAccepted=>,Error=>,ErrorID=>,MovementI
d=> );
```

//P2 为圆弧点（辅助点）和 P3（终点）的圆弧插补

```
MC_MoveLinearAbsolute[1](
    AxisGroup:=AxisGroup,Execute:=start1,Position:=Targetpiont[4],Velocity:=
3000,
    Acceleration:=1500,Deceleration:=1500,Jerk:=1000,CoordSystem:=CoordSy
s, BufferMode:=SM3_Common.MC_BUFFER_MODE.BlendingLow,
    TransitionMode:=tran,TransitionParameter:=transpara,
```

```
OrientationMode:=,VelFactor:=,AccFactor:=,  
JerkFactor:=,Done=>,Busy=>,Active=>,  
CommandAborted=>,CommandAccepted=>,Error=>,ErrorID=>,MovementI  
d=> );
```

//P4 点的直线插补

```
MC_MoveCircularRelative[1](  
    AxisGroup:=axisgroup,Execute:=start1,CircMode:=SMC_CIRC_MODE.BO  
    RDER,  
    AuxPoint:=Targetpion[5],EndPoint:=Targetpion[6],PathChoice:=0,Velocity:  
    =3000,  
    Acceleration:=1500,Deceleration:=1500,Jerk:=1000,CoordSystem:=CoordSy  
    s,  
    BufferMode:=SM3_Common.MC_BUFFER_MODE.BlendingLow,Transitio  
    nMode:=tran,  
    TransitionParameter:=transpara,OrientationMode:=,VelFactor:=,AccFactor:=  
    ,  
    JerkFactor:=,Done=>,Busy=>,Active=> ,  
    CommandAborted=>,CommandAccepted=>,Error=>,ErrorID=>,MovementI  
    d=> );
```

//P5 为圆弧点（辅助点）和 P6（终点）的圆弧插补

```
MC_MoveLinearAbsolute[2](  
    AxisGroup:=AxisGroup,Execute:=start1,Position:=Targetpion[7],Velocity:=  
    3000,  
    Acceleration:=1500,Deceleration:=1500,Jerk:=1000,CoordSystem:=CoordSy  
    s,  
    BufferMode:=SM3_Common.MC_BUFFER_MODE.BlendingLow,Transitio
```

```
nMode:=tran,  
TransitionParameter:=transpara,OrientationMode:=,VelFactor:=,AccFactor:=,  
JerkFactor:=,Done=>,Busy=>,Active=>,  
CommandAborted=>,CommandAccepted=>,Error=>,ErrorID=>,MovementI  
d=> );  
//P7 点的直线插补
```

Point（生成工件坐标的位置）子程序代码如下：

```
dx:= 50;  
dy:= 60;  
r0:=5;  
h1:=10;  
PI:=3.14;  
k := (dy)/(dx);  
L:= r0 - r0*COS(PI/4);  
m := SQRT(L*L/(1+k*k));  
n := k*m;  
h2 := h1 + r0*SIN(PI/4);  
p := SQRT(r0*r0/(1+k*k));  
q := k*p;  
pX[1]:=pX[0];  
pY[1]:=pY[0];  
pZ[1]:=pZ[0]+h1;  
pX[2]:=m+pX[0];  
pY[2]:=n+pY[0];  
pZ[2]:=h2+pZ[0];  
pX[3]:=p+pX[0];  
pY[3]:=q+pY[0];
```

```
pZ[3]:=h1+r0+pZ[0];
```

```
pX[4]:=dx-p+px[0];
```

```
pY[4]:=dy-q+pY[0];
```

```
pZ[4]:=h1+r0+pZ[0];
```

```
px[5]:=dx-m+px[0];
```

```
pY[5]:=dy-n+pY[0];
```

```
pZ[5]:=h2+pZ[0];
```

```
pX[6]:=dx+pX[0];
```

```
pY[6]:=dy+pY[0];
```

```
pZ[6]:=h1+pZ[0];
```

```
pX[7]:=dX+pX[0];
```

```
pY[7]:=dY+pY[0];
```

```
pz[7]:=pZ[0]-h1;
```

```
//门型轨迹公式
```

```
piont[0].X:=pX[0];
```

```
piont[0].Y:=pY[0];
```

```
piont[0].Z:=pZ[0];
```

Targetpiont[0].c:=piont[0]; //以 P0 点转换为例，剩下的 P1-P7 点的转换方式与 P0 点一致。

其运行后的轨迹曲线如图 7.41 所示，其中深蓝色代表 X 轴位置，绿色代表 Y 轴位置，棕色代表 Z 轴位置，灰色代表 X 速度，青色代表 Y 轴速度，橙色代表 Z 轴速度，红色代表合速度。

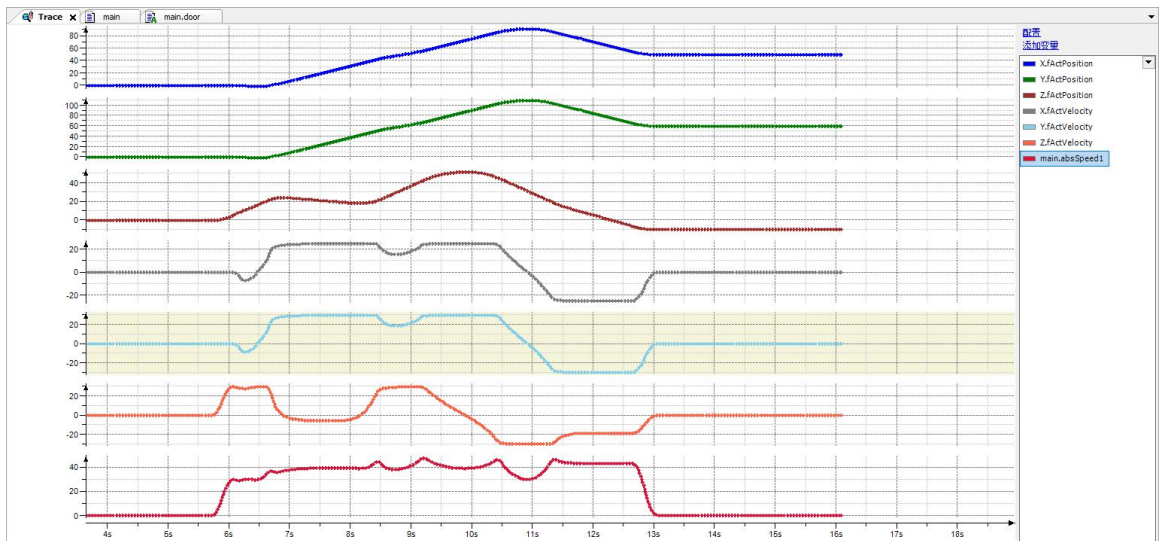


图 7.41 连续插补速度和位移曲线图

从图 7.41 可看出，轴 X、Y、Z 的速度曲线在各线段的衔接处没有先减速、后加速的过程；插补运动的合速度基本恒定，即插补速度是连续的。

连续插补的目录树如图 7.42 所示

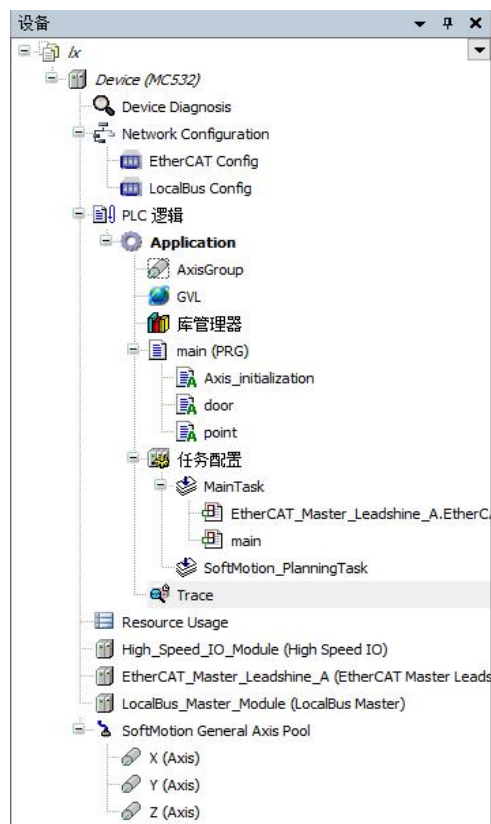


图 7.42 连续插补目录树

7.7.速度曲线和位移曲线的介绍和各个变量添加、使用方法

变量的跟踪曲线在程序调试过程中，雷赛 leadsys 软件可显示变量变化的跟踪曲线。使用方法如下。

鼠标右键点击项目树中的“Application”，选择“跟踪”，如图 7.43 所示。

在弹出的对话框中填写一个跟踪曲线的名称，如图 7.44 所示的 Trace。

在项目树中用鼠标双击跟踪曲线名，打开跟踪曲线的界面。

在跟踪曲线界面的右上角，点击“配置”，弹出对话框如图 7.45 所示。将其中的“任务”选“Main Task”。再点击界面右上角的“添加变量”，弹出如图 7.46 所示的对话框。在“跟踪变量”中选“IOConfig_Globals”下的“X”，找到 X 轴的位置变量 fActPosion，点“确定”。再添加 X 轴的速度变量 fActVelocity。

然后在“Y 和 Z”中选其位置和速度变量。再添加 XYZ 轴的合速度 absSpeed。该变量在“Application”下的“GVL”中。

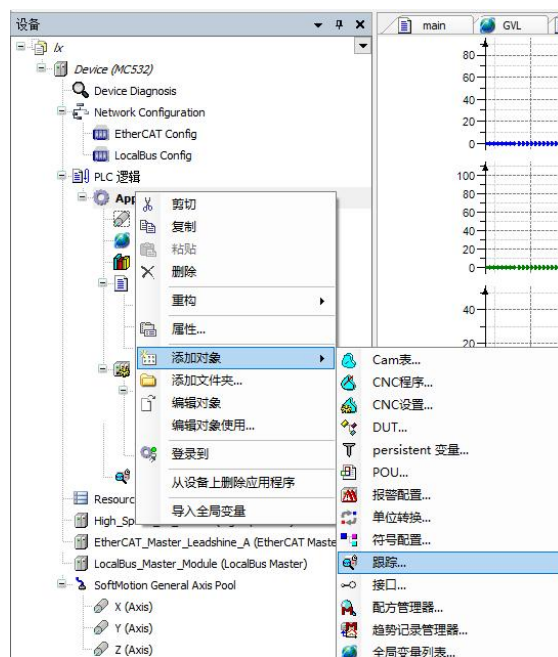


图 7.43 添加跟踪曲线



图 7.44 跟踪曲线的名称

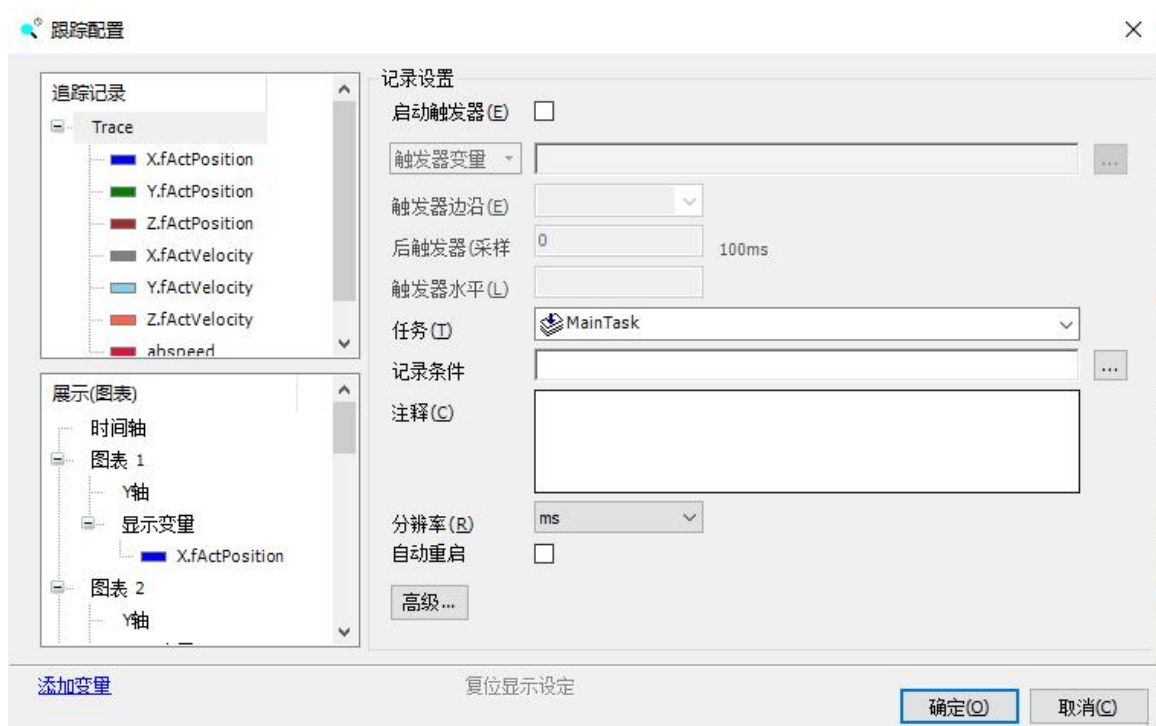


图 7.45 跟踪配置对话框

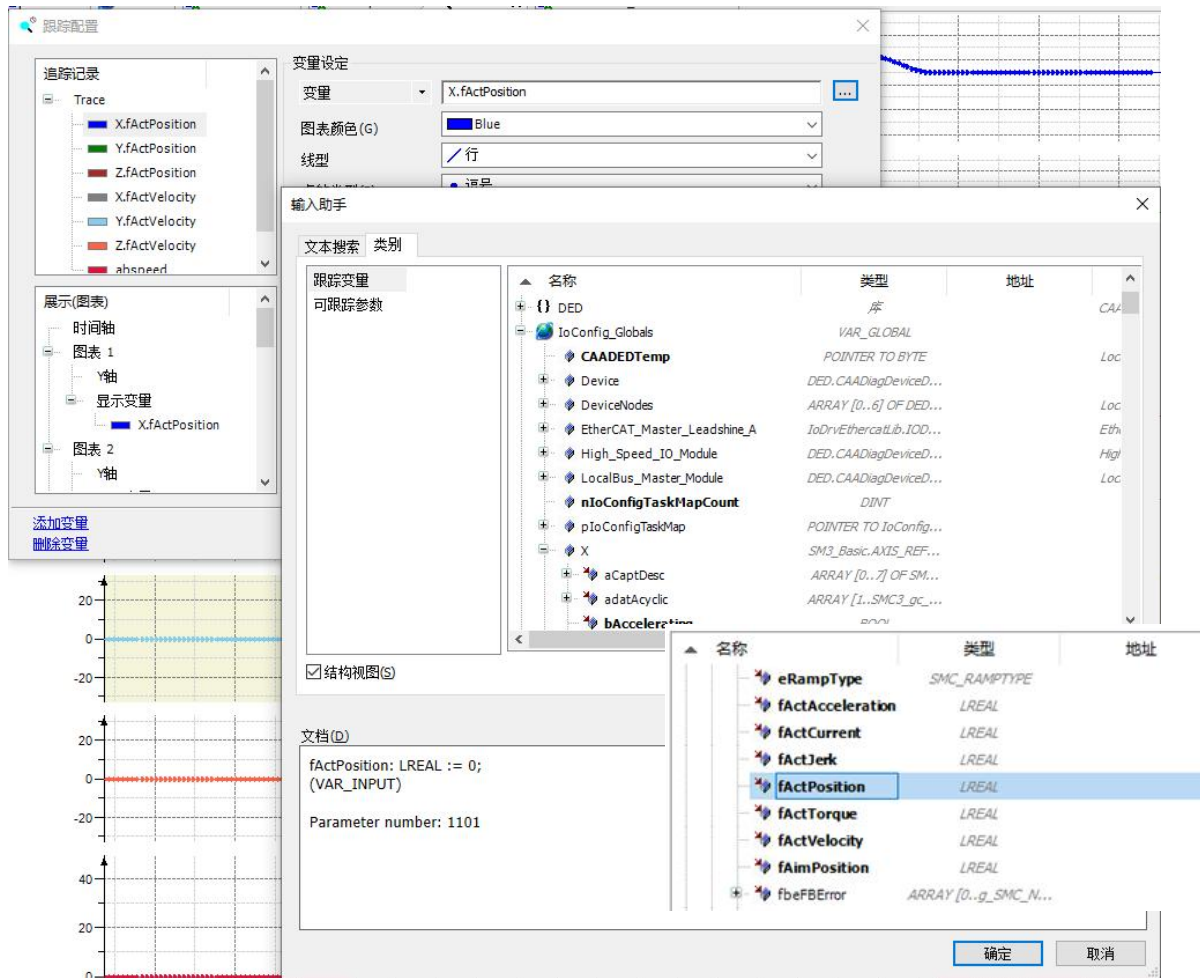


图 7.46 添加变量

点击界面中的“高级”按钮，打开“高级跟踪设置”，将“每个变量的追踪编辑器缓冲大小”改为 5000000，即曲线长度为 2 小时 46 分钟 39s。如图 7.47 所示。

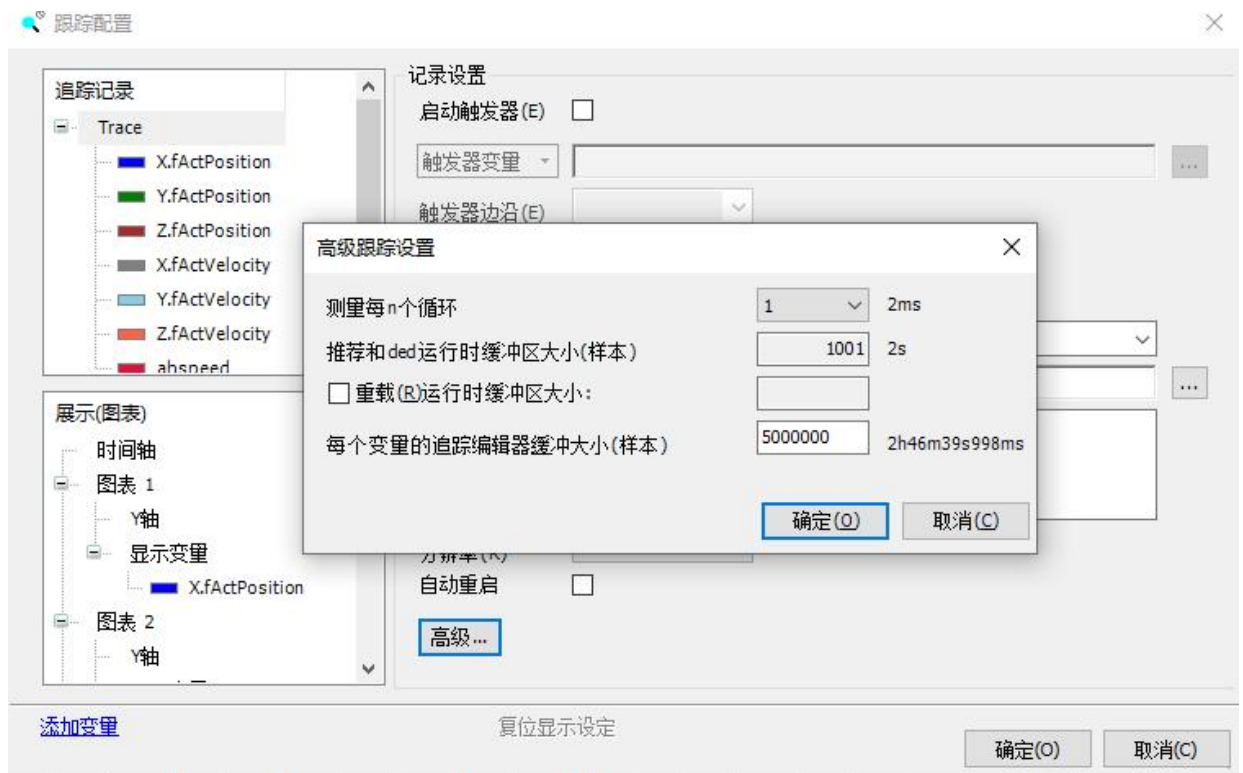


图 7.47 高级跟踪设置界面

运行程序后，在变量跟踪界面上点击鼠标右键。可以先选“转换为多通道”，使每个信号有自己的纵坐标，如图 7.48 所示。

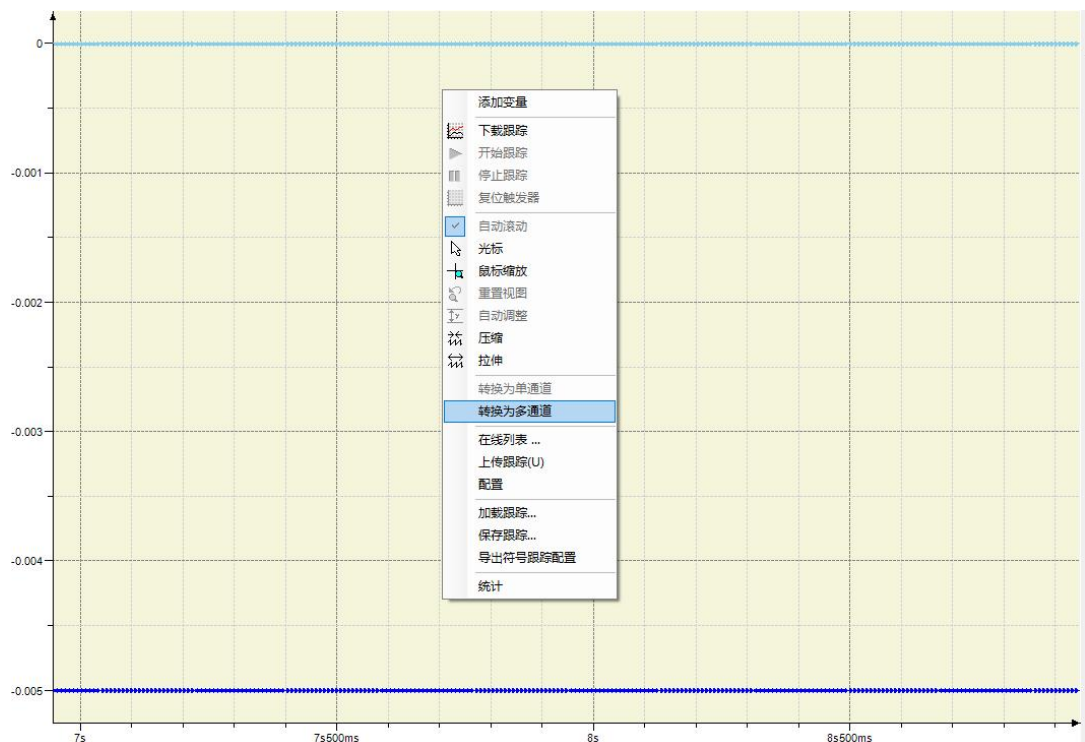


图 7.48 选择转换为多通道

再选“下载跟踪”，开始显示变量曲线，如图 7.49 所示。

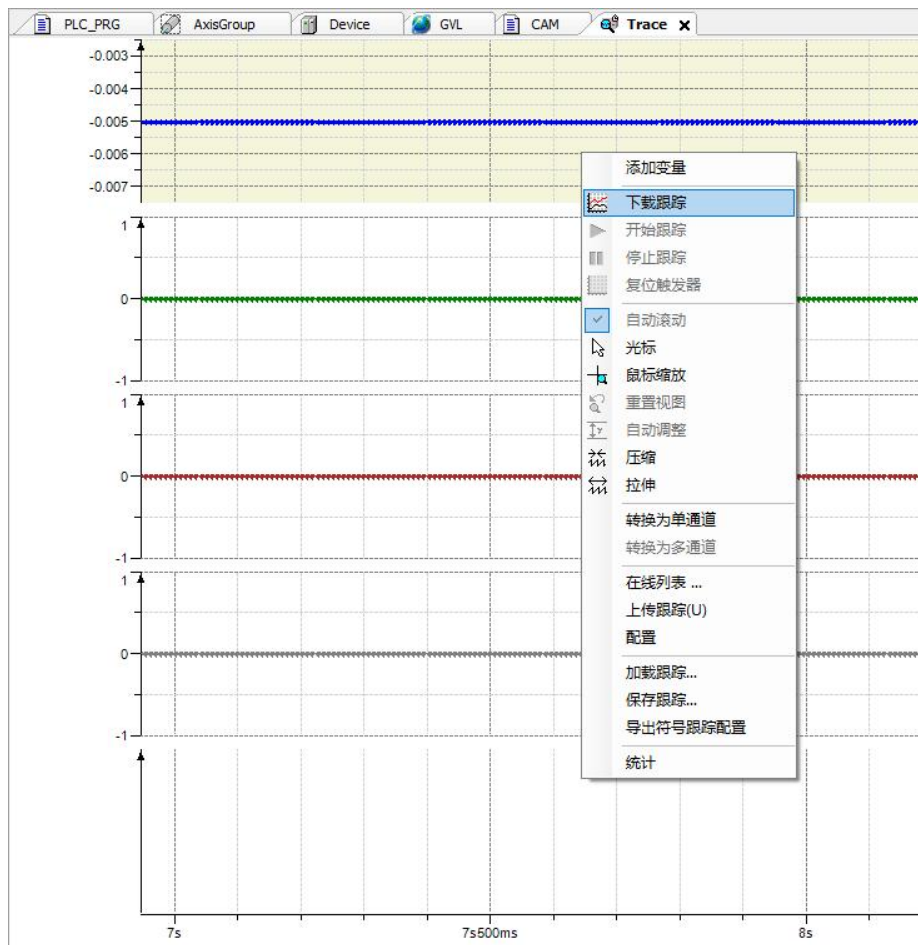


图 7.49 下载跟踪曲线

要暂停曲线跟踪，在界面上点击右键，选“停止跟踪”。

停止跟踪后，转动鼠标滚轮可放大、缩小曲线，按住鼠标左键可拖拽曲线。

7.8. 离线仿真的方法

在调试程序过程中，使用 LeadSys 软件的离线仿真功能十分方便。

离线仿真，即运动控制器、电机不需要上电，PC 机独立运行程序，就可以模拟系统的运行结果。

打开工程，在菜单中选择“在线”→“仿真”，如图 7.50 所示；程序编译后，在菜单中选“登录到”，弹出对话框如图 7.51 所示，选择“是”。点击菜单中的“运行”，即开始仿真。

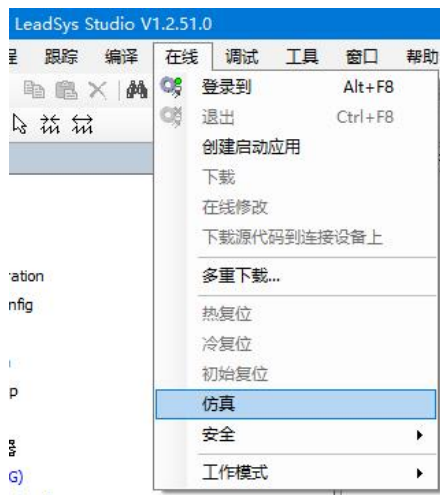


图 7.50 选择离线仿真

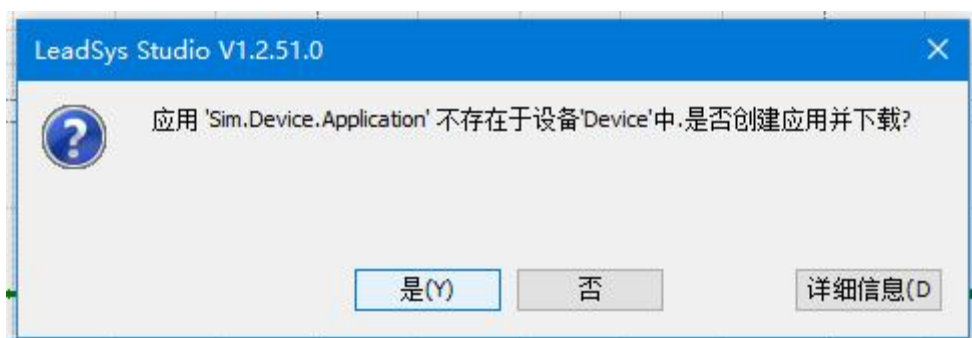


图 7.51 下载仿真程序

第八章 电子凸轮

8.1.功能描述

电子凸轮是利用构造的凸轮曲线来模拟机械凸轮，以达到与机械凸轮系统相同的从轴与主轴之间相对运动的软件功能。

电子凸轮对比机械凸轮，具有使用灵活、节约成本、减小机械噪音等优点。

8.2.相关功能块

表 8.1

名称	FB/FC	功能
MC_CamTableSelect	FB	凸轮表选择
MC_CamIn	FB	电子凸轮耦合
MC_CamOut	FB	电子凸轮脱离
SMC_GetTappetValue	FB	读取挺杆状态

8.3.数据结构

1) MC_CAM_REF（凸轮表数据结构）

表 8.2

变量	数据类型	初始值	描述
wCamStructID	WORD	16#DC3 4	凸轮表 ID
byType	BYTE	0	描述凸轮类型，即凸轮的表示方式。以下设置有效： 0：多项式描述（5 次） 1：等距、一维从站位置表

			2: 非等距, 相关主从位置二维表 3: 特定点的多项式描述, 包括主位置、从位置、从速度和从加速度 (XYVA)
byVarType	BYTE	0	仅在 byType=1 或 byType=2 时使用 定义曲线表中输入的变量类型: 0: <未使用> 1:INT 2:UINT 3:DINT 4:UDINT 5:REAL 6:LREAL
xStart	LREAL		主轴起始位置
xEnd	LREAL		主轴结束位置
nElements	INT		元素数量, 取决于凸轮类型、从位置、主/从位置或 XYVA 点的数量。
nTappets	INT		挺杆开关动作的次数
pce	POINTER TO BYTE	0	指针指向类型取决于 byType 1: pce 指向 SMC_CAMTable_..._1 例如:

			SMC_CAMTable_LREAL_1 28_1 2: pce 指向 SMC_CAMTable_..._2 例如: SMC_CAMTable_UDINT_2 56_2 3: pce 指向一个 SMC_CAMXYVA 数组
pt	POINTER TO SMC_CAM Tappet	0	
dwTappetActiveBits	DWORD		内部变量
strCAMName	STRING	“	凸轮表名称
byInterpolationQuality	BYTE	1	1: 线性插值 3: 三次插值
byCompatibilityMode	BYTE	0	兼容模式: Bit0: TRUE, 定期执行具有 主宽度的表
bChangedOnline	BOOL	FALSE	内部变量
xPartofLM	BOOL		内部变量

2) SMC_CAMXYVA (凸轮关键点数据结构)

表 8.3

变量	数据类型	初始 值	描述
dY	LREAL	0	主轴相位
dX	LREAL	0	从轴位移
dV	LREAL	0	一阶导数 dY/dX , 从轴速度
dA	LREAL	0	二阶导数 d^2Y/dX^2 , 从轴加 速度

8.4. 电子凸轮使用流程

8.4.1. 电子凸轮流程图

下图描述了 Leadsys 软件中电子凸轮的典型使用流程。

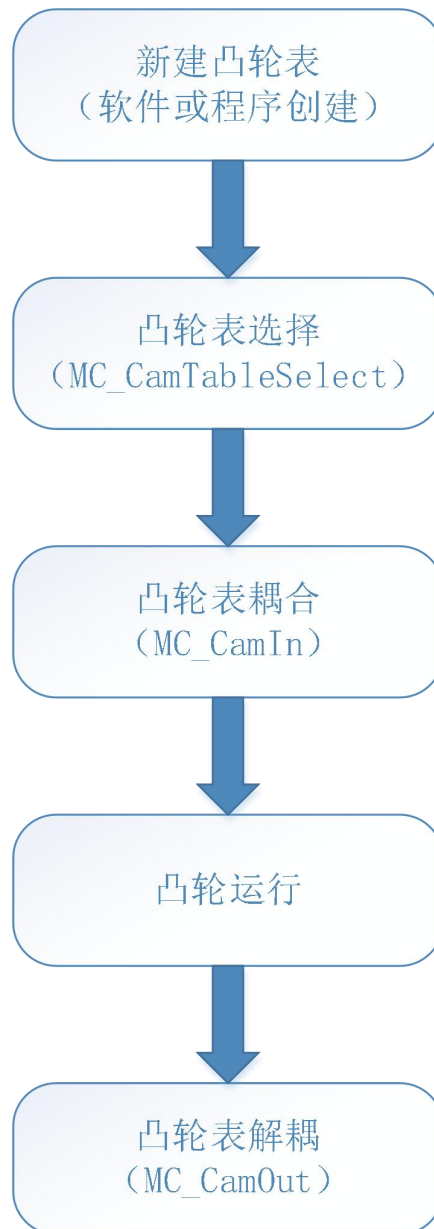


图 8.1 流程示例

注意：

- 在调用 MC_CamIn 前，需要先对轴进行使能操作；
- 调用 MC_CamOut 时，如果从轴状态机没有在 synchronized_motion 状态，执行后会报错；
- 使能 MC_CamOut 后，从轴状态机为 continuous_motion，如果分离时轴速度不为 0，从轴会以分离时的速度匀速运行，故 MC_CamOut 一般与 MC_Stop 等配合使用。

8.4.2. 电子凸轮程序使用步骤

本节通过举例，简要说明如何使用电子凸轮功能，具体实现功能为：触发主轴运行速度指令，从轴跟随主轴实现凸轮运动。操作步骤如下：

1) 右键点击“Application”选择“添加对象”-“Cam 表”，并根据需要修改凸轮表曲线。

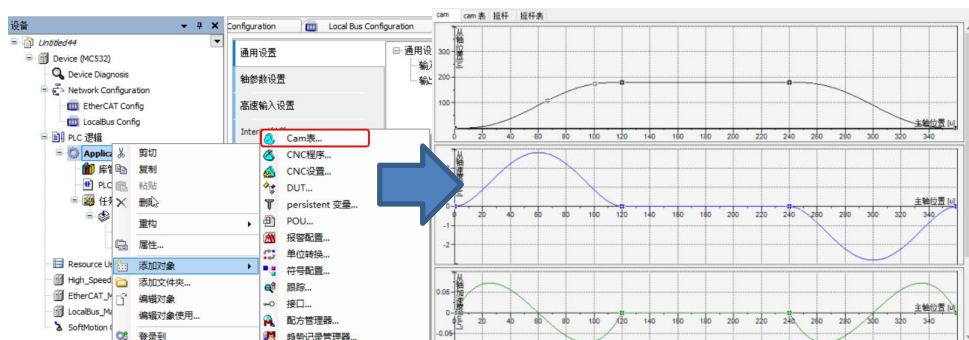


图 8.2

2) 添加运动控制轴（此处以虚轴为例），右键点击“SoftMotion General Axis Pool”选择“添加设备”添加虚拟驱动器。

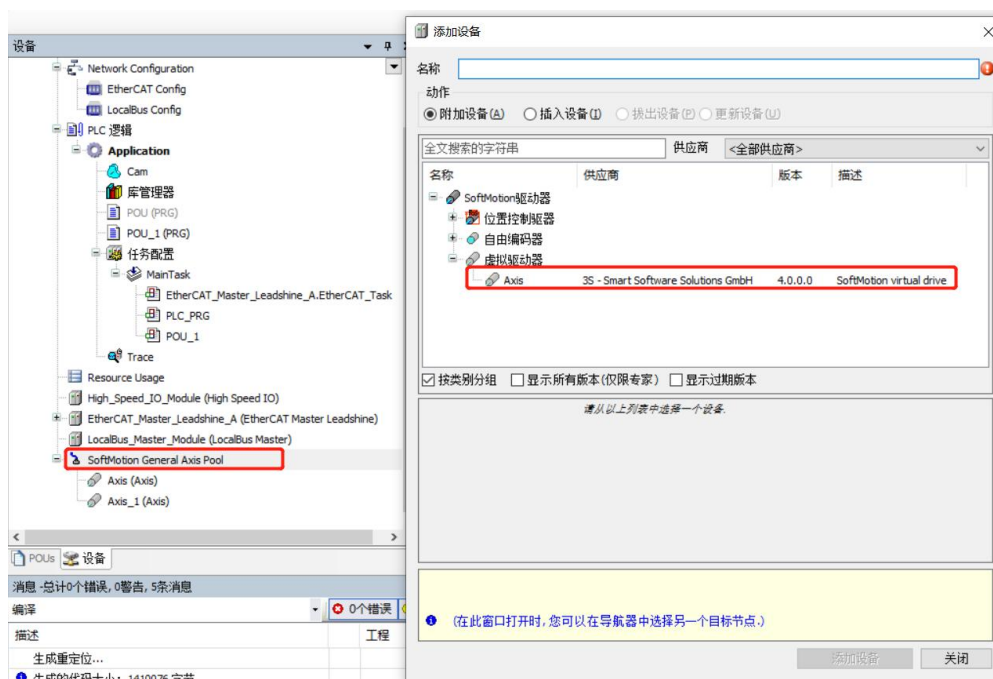


图 8.3

3) 实例化功能块，声明相关变量。

```
PROGRAM POU_2
VAR
    MC_Power_0: MC_Power;
    MC_Power_1: MC_Power;
    MC_CamTableSelect_0: MC_CamTableSelect;
    xCamIn: BOOL;
    MC_CamIn_0: MC_CamIn;
    MC_MoveVelocity_0: MC_MoveVelocity;
    xGoVel: BOOL;
    fVelo: LREAL :=100;
    xPeriodic: BOOL;
    xMasterAbsolute: BOOL;
    xSlaveAbsolute: BOOL;
    MasterScaling: LREAL :=1;
    SlaveScaling: LREAL :=1;
    MasterOffset: LREAL;
    SlaveOffset: LREAL;
    StartMode: SM3_Basic.MC_StartMode;
    VelocityDiff: LREAL;
    Acceleration: LREAL;
    Deceleration: LREAL;
END_VAR
```

图 8.4

VAR

```
MC_Power_0: MC_Power;

MC_Power_1: MC_Power;

    MC_CamTableSelect_0: MC_CamTableSelect;

xCamIn: BOOL;

MC_CamIn_0: MC_CamIn;

MC_MoveVelocity_0: MC_MoveVelocity;

xGoVel: BOOL;

fVelo: LREAL :=100;

xPeriodic: BOOL;

xMasterAbsolute: BOOL;

xSlaveAbsolute: BOOL;

MasterScaling: LREAL :=1;

SlaveScaling: LREAL :=1;

MasterOffset: LREAL;

SlaveOffset: LREAL;

StartMode: SM3_Basic.MC_StartMode;

VelocityDiff: LREAL;
```

Acceleration: LREAL;

Deceleration: LREAL;

END_VAR

4) 编写程序。

```
1  //轴使能
2  MC_Power_0(
3      Axis:= Axis,
4      Enable:=1,
5      bRegulatorOn:= 1,
6      bDriveStart:= 1,
7      Status=> ,
8      bRegulatorRealState=> ,
9      bDriveStartRealState=> ,
10     Busy=> ,
11     Error=> ,
12     ErrorID=> );
13
14  MC_Power_1(
15     Axis:= Axis_1,
16     Enable:= 1,
17     bRegulatorOn:= 1,
18     bDriveStart:= 1,
19     Status=> ,
20     bRegulatorRealState=> ,
21     bDriveStartRealState=> ,
22     Busy=> ,
23     Error=> ,
24     ErrorID=> );
25
26  //主轴速度指令
27  MC_MoveVelocity_0(
28     Axis:= Axis,
29     Execute:= xGoVel,
30     Velocity:= fVelo,
31     Acceleration:= 100,
32     Deceleration:= 100,
33     Jerk:= ,
34     Direction:= ,
35     BufferMode:= ,
36     InVelocity=> ,
37     Busy=> ,
38     Active=> ,
39     CommandAborted=> ,
40     Error=> ,
41     ErrorID=> );
42
43  //凸轮表选择
44  MC_CamTableSelect_0(
45     Master:= Axis,
46     Slave:= Axis_1,
47     CamTable:= Cam,
48     Execute:= xCamIn,
49     Periodic:= xPeriodic,
50     MasterAbsolute:= xMasterAbsolute,
51     SlaveAbsolute:= xSlaveAbsolute,
52     Done=> ,
53     Busy=> ,
54     Error=> ,
55     ErrorID=> ,
56     CamTableID=> );
```

图 8.5

```

58 //凸轮耦合
59 MC_CamIn_0(
60     Master:= Axis,
61     Slave:= Axis_1,
62     Execute:= MC_CamTableSelect_0.Done,
63     MasterOffset:= MasterOffset,
64     SlaveOffset:= SlaveOffset,
65     MasterScaling:= MasterScaling,
66     SlaveScaling:= SlaveScaling,
67     StartMode:= StartMode,
68     CamTableID:= MC_CamTableSelect_0.CamTableID,
69     VelocityDiff:= VelocityDiff,
70     Acceleration:= Acceleration,
71     Deceleration:= Deceleration,
72     Jerk:= ,
73     TappetHysteresis:= ,
74     InSync=> ,
75     Busy=> ,
76     CommandAborted=> ,
77     Error=> ,
78     ErrorID=> ,
79     EndOfProfile=> ,
80     Tappets=> );

```

100 %

图 8.6

//轴使能

```

MC_Power_0(
    Axis:= Axis,
    Enable:=1,
    bRegulatorOn:= 1,
    bDriveStart:= 1,
    Status=> ,
    bRegulatorRealState=> ,
    bDriveStartRealState=> ,
    Busy=> ,
    Error=> ,
    ErrorID=> );

```

```

MC_Power_1(
    Axis:= Axis_1,
    Enable:= 1,
    bRegulatorOn:= 1,
    bDriveStart:= 1,

```

```
Status=> ,  
bRegulatorRealState=> ,  
bDriveStartRealState=> ,  
Busy=> ,  
Error=> ,  
ErrorID=> );
```

//主轴速度指令

```
MC_MoveVelocity_0(  
    Axis:= Axis,  
    Execute:= xGoVel,  
    Velocity:= fVelo,  
    Acceleration:= 100,  
    Deceleration:= 100,  
    Jerk:= ,  
    Direction:= ,  
    BufferMode:= ,  
    InVelocity=> ,  
    Busy=> ,  
    Active=> ,  
    CommandAborted=> ,  
    Error=> ,  
    ErrorID=> );
```

//凸轮表选择

```
MC_CamTableSelect_0(  
    Master:= Axis,  
    Slave:= Axis_1,
```

```
CamTable:= Cam,  
Execute:= xCamIn,  
Periodic:= xPeriodic,  
MasterAbsolute:= xMasterAbsolute,  
SlaveAbsolute:= xSlaveAbsolute,  
Done=> ,  
Busy=> ,  
Error=> ,  
ErrorID=> ,  
CamTableID=> );
```

//凸轮耦合

```
MC_CamIn_0(  
    Master:= Axis,  
    Slave:= Axis_1,  
    Execute:= MC_CamTableSelect_0.Done,  
    MasterOffset:= MasterOffset,  
    SlaveOffset:= SlaveOffset,  
    MasterScaling:= MasterScaling,  
    SlaveScaling:= SlaveScaling,  
    StartMode:= StartMode,  
    CamTableID:= MC_CamTableSelect_0.CamTableID,  
    VelocityDiff:= VelocityDiff,  
    Acceleration:= Acceleration,  
    Deceleration:= Deceleration,  
    Jerk:= ,  
    TappetHysteresis:= ,  
    InSync=> ,
```



```
Busy=> ,  
CommandAborted=> ,  
Error=> ,  
ErrorID=> ,  
EndOfProfile=> ,  
Tappets=> );
```

5) 触发电子凸轮执行

触发 xCamIn 及 xGoVel 后，电子凸轮开始执行，运行曲线如下：

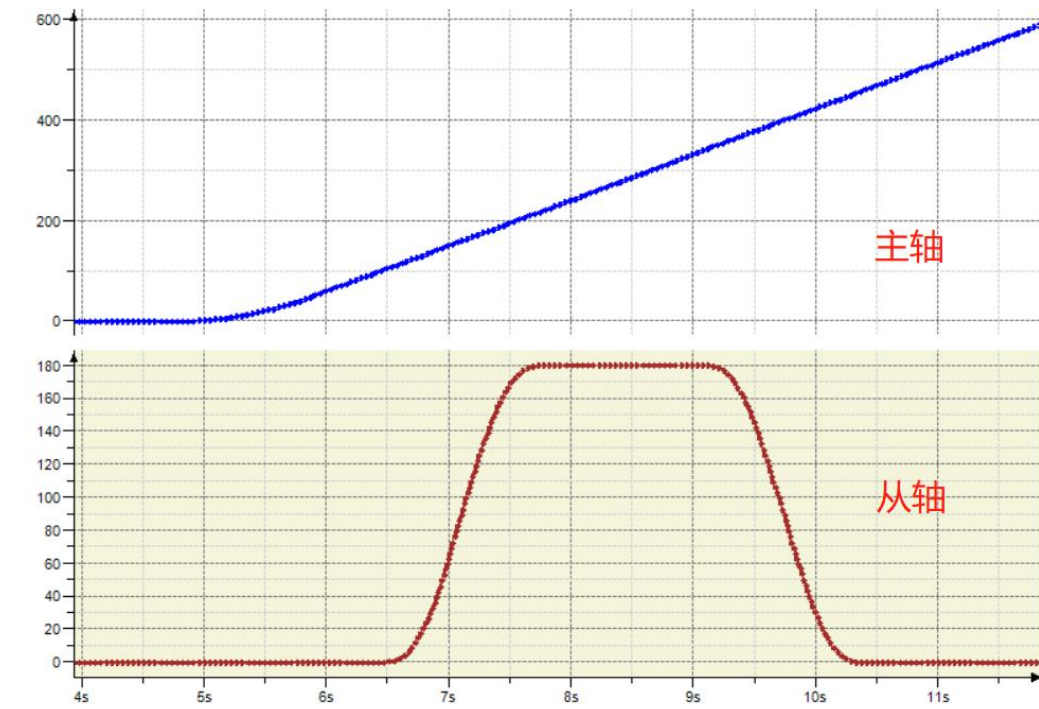


图 8.7

8.4.3. 凸轮表

在凸轮功能中，将由主轴相位和从轴位移组成的一对数据称之为凸轮关键点，凸轮表即为多个凸轮关键点数据组成的数据组合。

凸轮动作中，根据主轴相位和设定的曲线类型计算出从轴的位移，从而控制从轴的动作。

8.4.3.1通过软件编辑凸轮表

Leadsys 软件支持直接通过软件创建凸轮表，具体的操作步骤如下：

1) 右键点击“Application”选择“添加对象”-“Cam 表”。

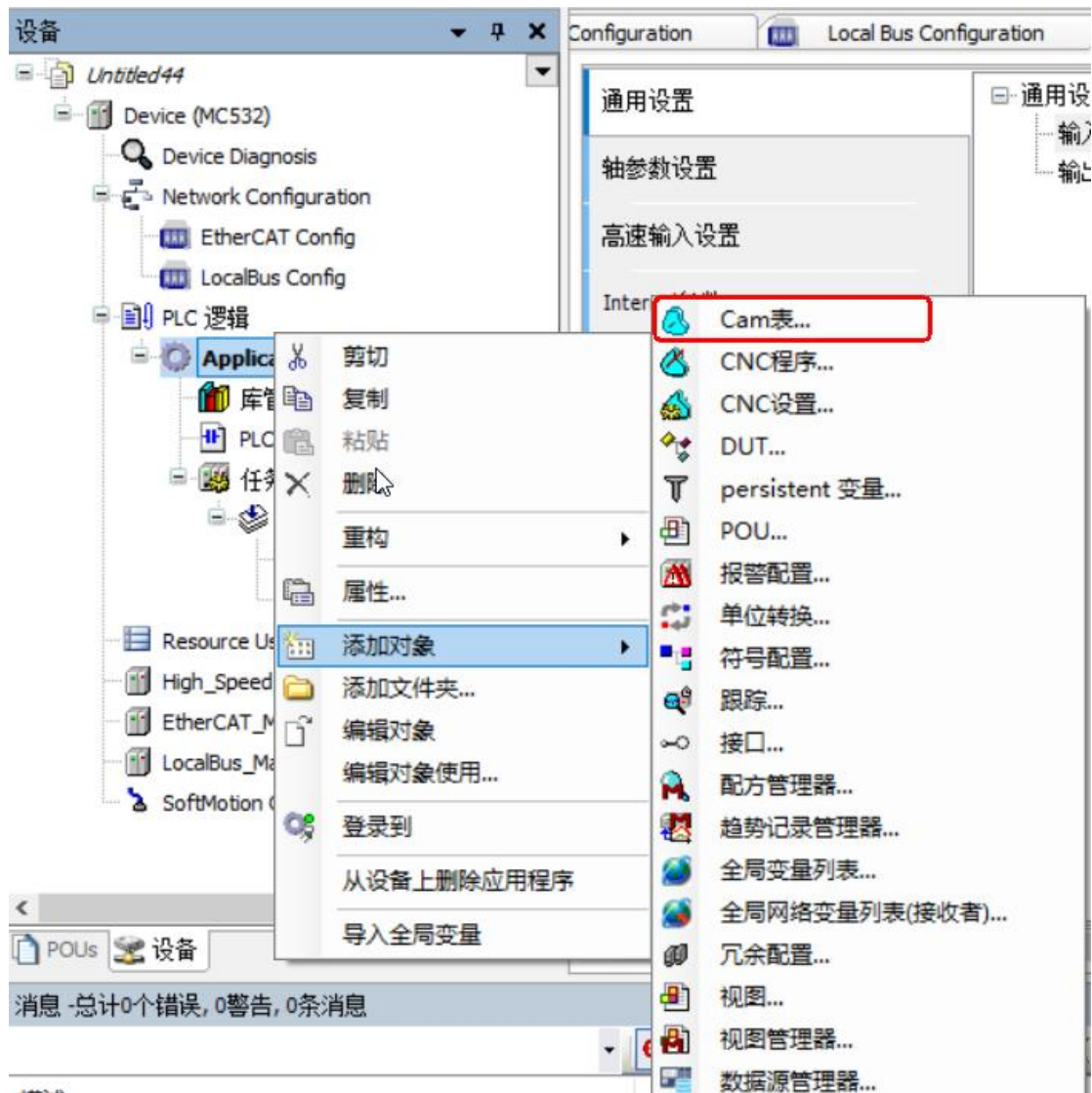


图 8.8

2) 双击“Cam”，打开凸轮表配置界面。

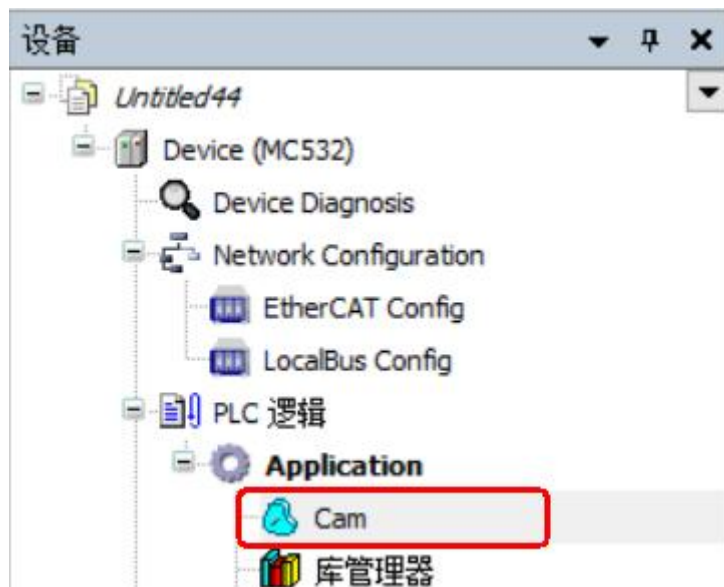


图 8.9

“Cam”页面用于显示凸轮表的位置曲线、速度曲线、加速度曲线、加加速度曲线，并且用户可以对关键点进行拖动，修改关键点的数据；

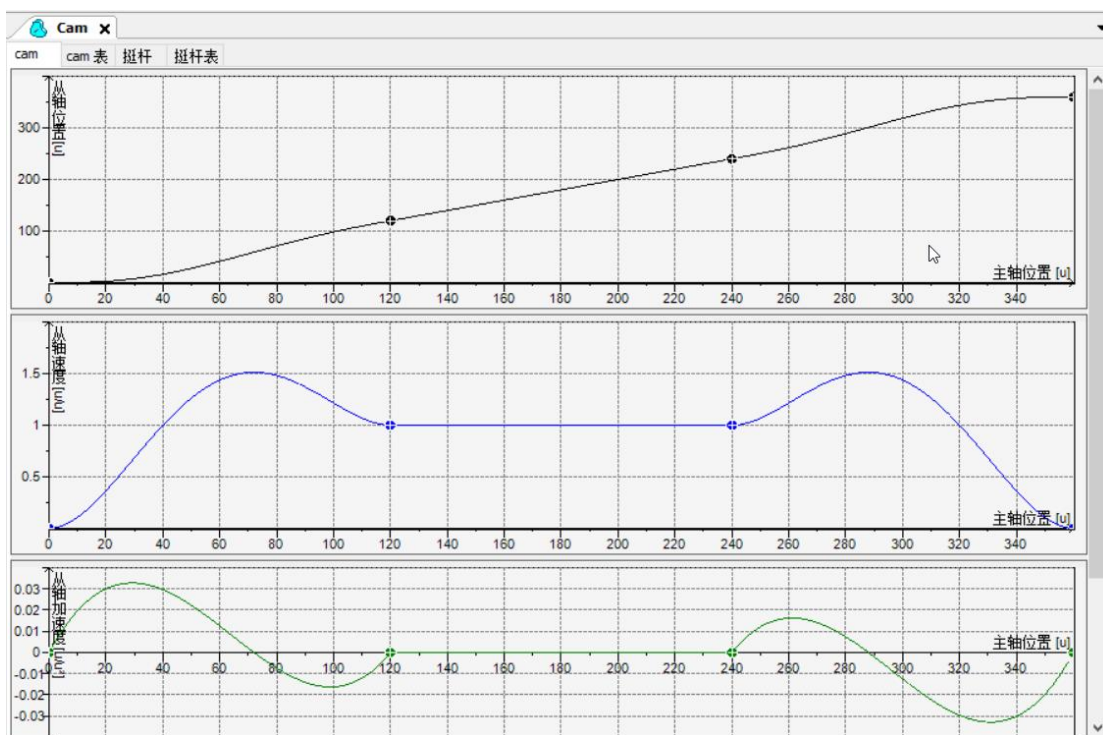


图 8.10

3) “Cam 表”界面可以对凸轮关键点数据进行详细的设置。

Cam x										
cam	cam 表	挺杆	挺杆表							
	X	Y	V	A	J	段类型	最小(...)	最大(...)	最大(...)	最大(...)
	0	0	0	0	0					
						Poly5	0	105.71...	1.3115...	0.0272...
	120	105.71...	1	0	0					
						Poly5	105.71...	240	1.2232...	0.0057...
	240	240	1	0	0					
						Poly5	240	360	1.512	0.0328...
	360	360	0	0	0					

图 8.11

表 8.4

选项	功能
	删除关键点
	插入关键点
X	主轴相位
Y	从轴位移
V	连接点速度比
A	连接点加速度比
J	连接点加加速度比
段类型	凸轮曲线类型，Poly5（五次曲线）、Line（直线）
最小（位置）	该段中从轴的最小位置，不可修改
最大（位置）	该段中从轴的最大位置，不可修改
最大（速度）	从轴速度比的最大值
最大（加速度）	从站加速度比的最大值

8.4.3.2 通过程序直接编辑凸轮表

本节讲述两种通过程序编辑凸轮表的情况：

- 1) 通过软件生成凸轮表，在程序中修改关键点数据；

通过软件生成 CAM 表时，Leadsys 会自动生成 MC_Cam_REF 类型的凸轮表数据，命名为“CAM 表定义名称”及凸轮表关键点 SMC_CAMXYVA 类型的数组，命名为“CAM 表定义名称+_A”，例：



图 8.12

上图新建凸轮表“CAM”，则对应生成的凸轮表数据“CAM”及凸轮表关键点数组 “CAM_A”，用户可以直接在工程中修改。

如将第 3 个关键点位置改为（200,200）：

```
CAM_A[2].dx:=200
```

```
CAM_A[2].dy:=200;
```

[-] GVL_for_Cam.Cam	Device.Application	MC_CAM_REF	
✖ wCamStructID		WORD	56372
✖ byType		BYTE	3
✖ byVarType		BYTE	0
✖ xStart		LREAL	0
✖ xEnd		LREAL	360
✖ nElements		INT	4
✖ nTappets		INT	4
⊕ ✖ pce		POINTER TO BYTE	16#000001D7C80C2...
⊕ ✖ pt		POINTER TO SM...	16#000001D7C80C2...
✖ dwTappetActiveBits		DWORD	0
✖ strCAMName		STRING	'Cam'
✖ byInterpolationQuality		BYTE	1
✖ byCompatibilityMode		BYTE	0
✖ bChangedOnline		BOOL	FALSE
✖ xPartofLM		BOOL	TRUE
[-] GVL_for_Cam.Cam_A	Device.Application	ARRAY [0..3] O...	
⊕ GVL_for_Cam.Cam_A[0]		SMC_CAMXYVA	
⊕ GVL_for_Cam.Cam_A[1]		SMC_CAMXYVA	
⊕ GVL_for_Cam.Cam_A[2]		SMC_CAMXYVA	
⊕ GVL_for_Cam.Cam_A[3]		SMC_CAMXYVA	

图 8.13

请注意，修改之后的凸轮关键点数据会立即生效，故操作时要注意是否有撞机风险。

2) 直接通过程序生成凸轮表；

直接通过程序生成凸轮表需要三个步骤，下面举例说明：

第一步定义变量。

VAR

Cam : MC_Cam_REF;

arCam : Array[0..3] of MC_CAMXYVA;

END_VAR;

第二步根据需求设置关键点并给 arCam 参数赋值。

```
arCam[0].dX:=0;
```

```
arCam[0].dY:=0;
```

```
arCam[0].dV:=1;
```

```
arCam[0].dA:=0;
```

```
arCam[1].dX:=100;
```

```
arCam[1].dY:=100;
```

```
arCam[1].dV:=1;
```

```
arCam[1].dA:=0;
```

```
arCam[2].dX:=200;
```

```
arCam[2].dY:=200;
```

```
arCam[2].dV:=1;
```

```
arCam[2].dA:=0;
```

```
arCam[3].dX:=360;
```

```
arCam[3].dY:=360;
```

```
arCam[3].dV:=1;
```

```
arCam[3].dA:=0;
```

第三步设置凸轮属性。

Cam

```
( wCamStructID:= ,
```

```
byType:= 3,           //设置凸轮表关键点类型
```

```
byVarType:= ,
```

```
xStart:= 0,           //设置凸轮表主轴起点
```



```
xEnd:= 360,           //设置凸轮表主轴终点
nElements:= 4,        //设置凸轮表关键点个数
nTappets:= ,
pce:= ADR(arCam),     //指定凸轮表关键点数组
pt:= ,
dwTappetActiveBits:= ,
strCAMName:= ,
byInterpolationQuality:= ,
byCompatibilityMode:= ,
bChangedOnline:= ,
xPartofLM:= );
```

8.4.4. 功能块参数对凸轮运行的影响

8.4.4.1 MC_CamTableSelect. Periodic

该参数决定电子凸轮的执行为周期或非周期类型，为 **True** 时电子凸轮周期执行，为 **False** 时电子凸轮只执行一个周期。

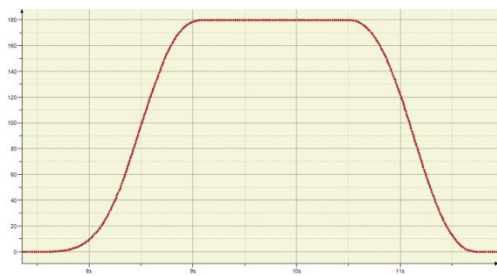


图 8.14 Periodic:=false，执行一个周期

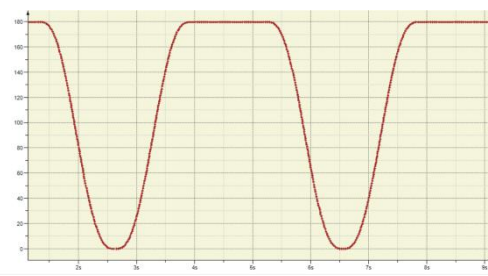


图 8.15 Periodic:=true，周期执行

8.4.4.2 MC_CamTableSelect. MasterAbsolute

该参数决定电子凸轮主轴跟踪距离坐标系是按绝对位置还是相对位置，**True** 时为绝对位置，**False** 时为相对位置。

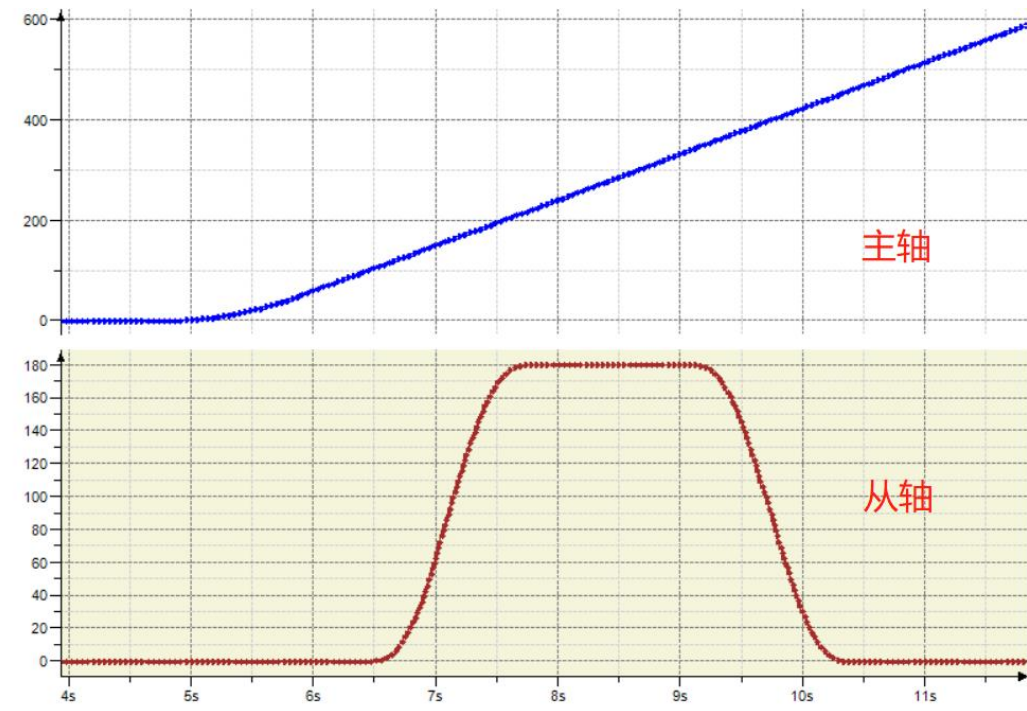


图 8.16

当 MasterAbsolute 为 False 时，主轴跟踪距离坐标系为相对位置，即启动时主轴相位为 0；

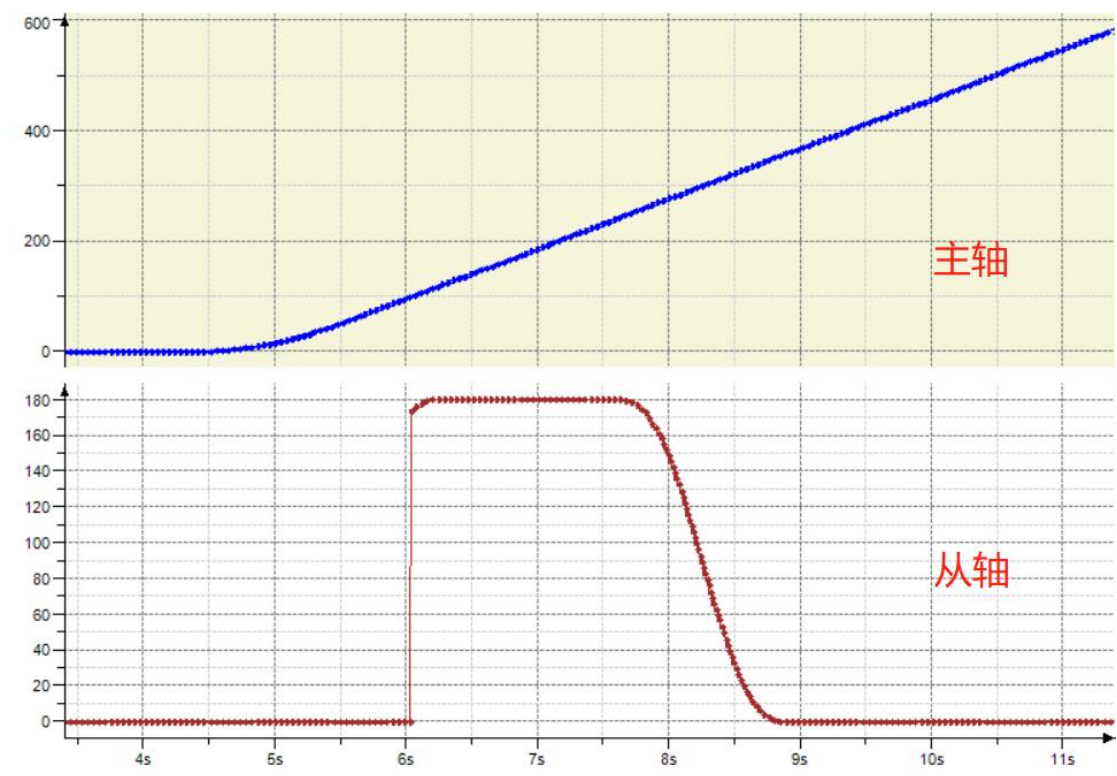


图 8.17

当 MasterAbsolute 为 True 时，主轴跟踪距离坐标系为绝对位置，即启动时主轴相位为主轴当前位置，当主轴位置不在凸轮表主轴的范围内时，从轴报错（SMC_CI_MASTER_OUT_OF_SCALE），使用这种方式启动凸轮时一定要注意，容易出现飞车风险；

8.4.4.3 MC_CamTableSelect. SalveAbsolute

该参数与 MC_CamIn 指令中的 StartMode 综合指定从轴当前指令位置为凸轮表的绝对（当前主轴相位对应的凸轮表从轴的位移）还是相对（当前主轴相位对应的凸轮表从轴的位移叠加指令开始时从轴位置），True 时为绝对位置，False 时为相对位置。（此处固定 MC_CamIn. StartMode:=0）

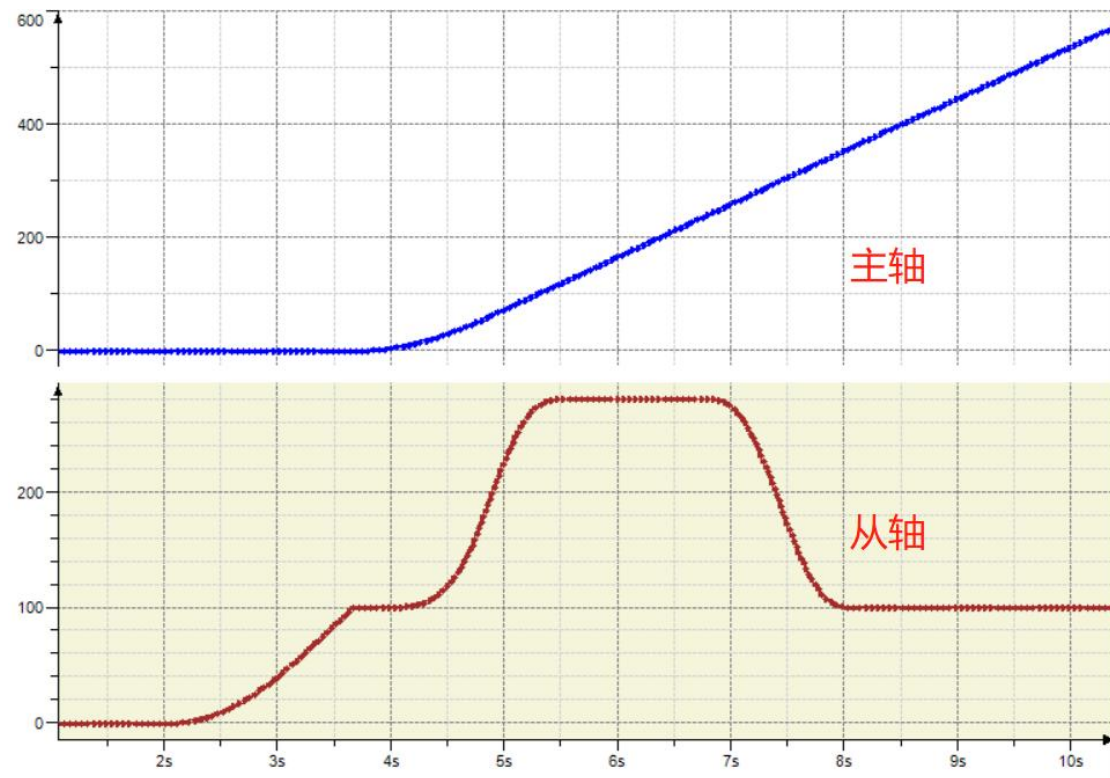


图 8.18

当 MasterAbsolute 为 False 时，从轴坐标系为相对位置，启动时从轴指令位置为当前主轴相位对应的凸轮表从轴的位移叠加指令开始时从轴位置；

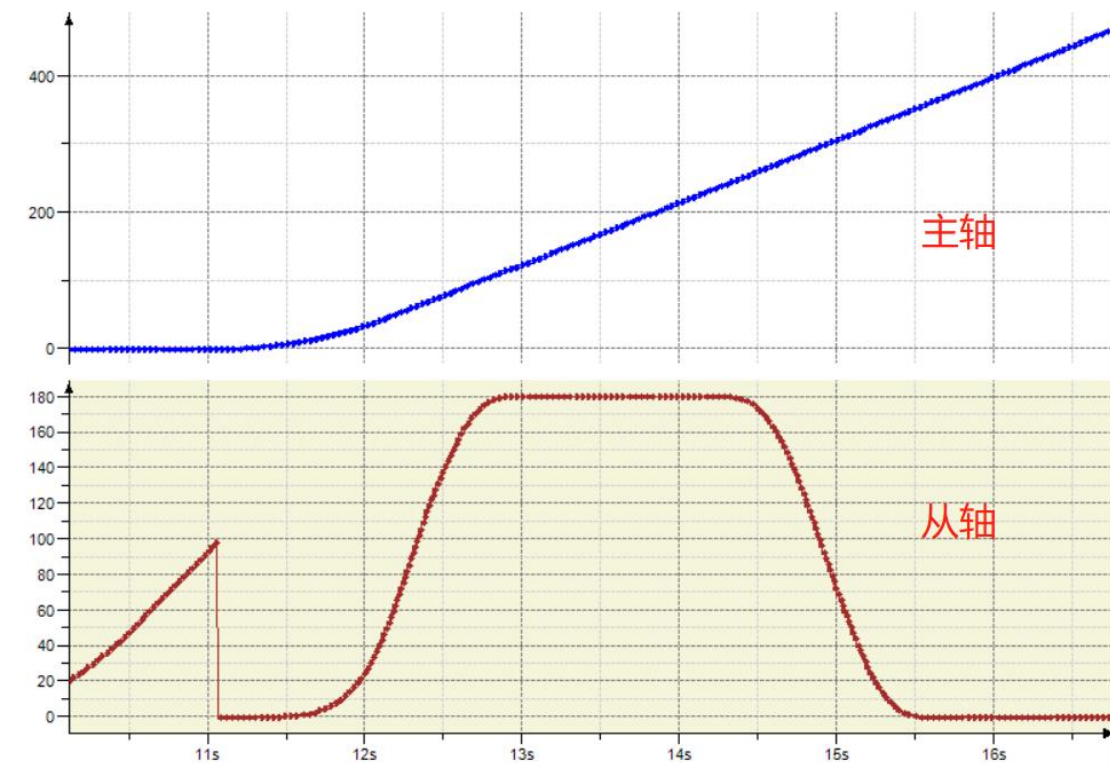


图 8.19

当 MasterAbsolute 为 True 时，从轴坐标系为绝对位置，启动时从轴指令位置为当前主轴相位对应的凸轮表从轴的位移，使用这种方式启动凸轮时一定要注意，容易出现飞车风险；

8.4.4.4 MC_CamIn. MasterOffset, MasterScaling

主轴输入转换的位置是根据以下公式进行的，并且使用转化后的 X 作为凸轮的输入：

$$X = \text{MasterScaling} * \text{MasterPosition} + \text{MasterOffset}$$

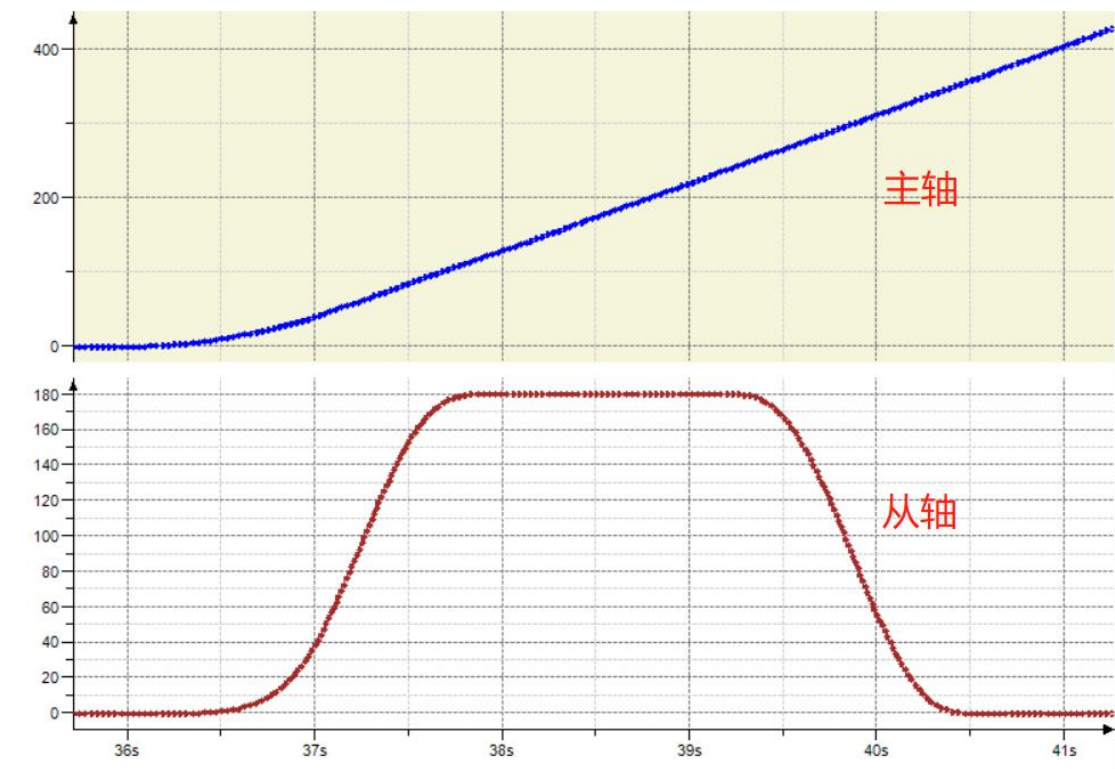


图 8.20

当 MasterScaling:=1, MasterOffset:=0, SalveScaling:=1, SalveOffset:=0 时，参考曲线如上。

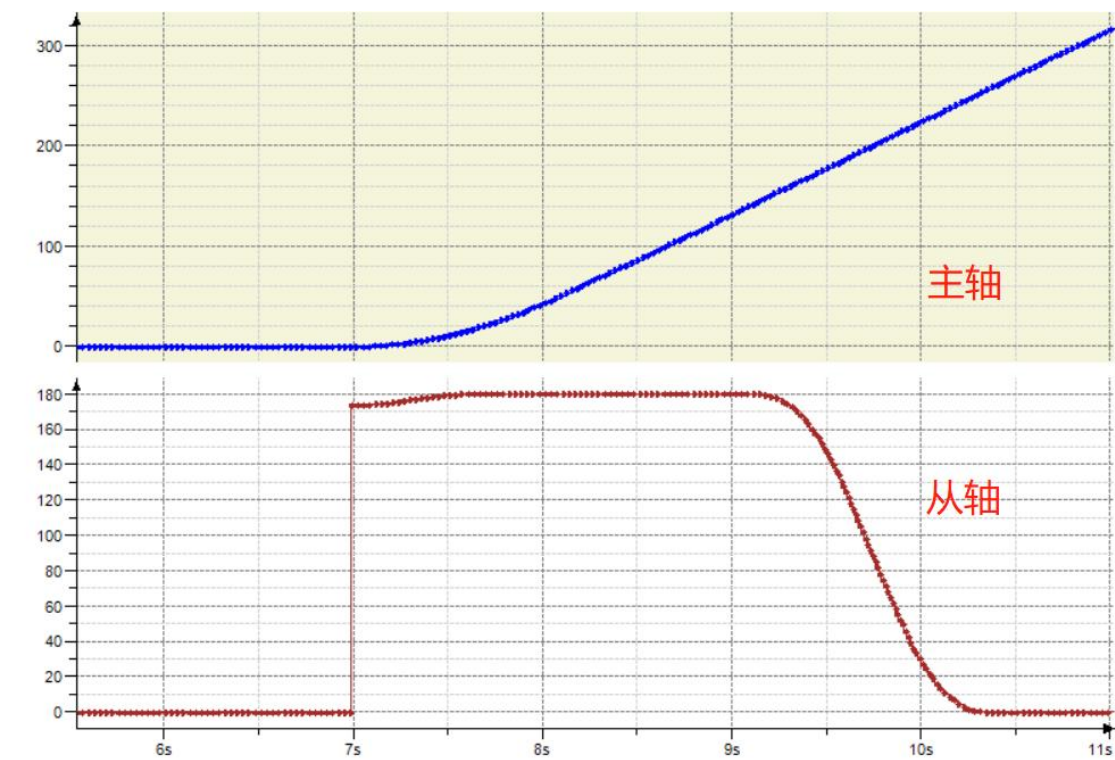


图 8.21

当 MasterScaling:=1, MasterOffset:=100 时, 凸轮启动时从轴指令位置为凸轮表主轴相位为 100 时对应的从轴位移, 即从轴侧观察主轴的相位叠加 MasterOffset 设置值。

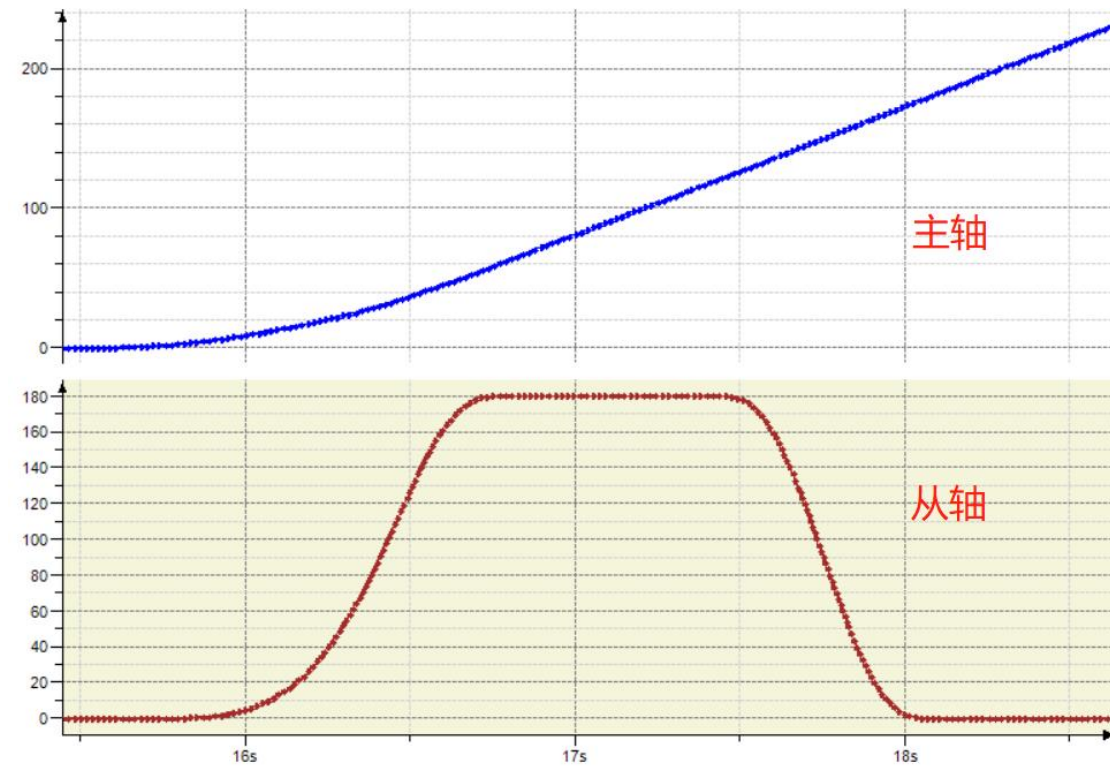


图 8.22

当 MasterScaling:=2, MasterOffset:= 0 时, 凸轮从轴的运行速度变为原来的 2 倍, 即从轴侧观察主轴的相位变为原来的两倍。

8.4.4.5 MC_CamIn. SalveOffset, SalveScaling

从轴输出转换的位置是根据以下公式进行的, 并且使用转化后的 Y 作为凸轮的输出:

$$Y = \text{SlaveScaling} * \text{CAM}(X) + \text{SlaveOffset}$$

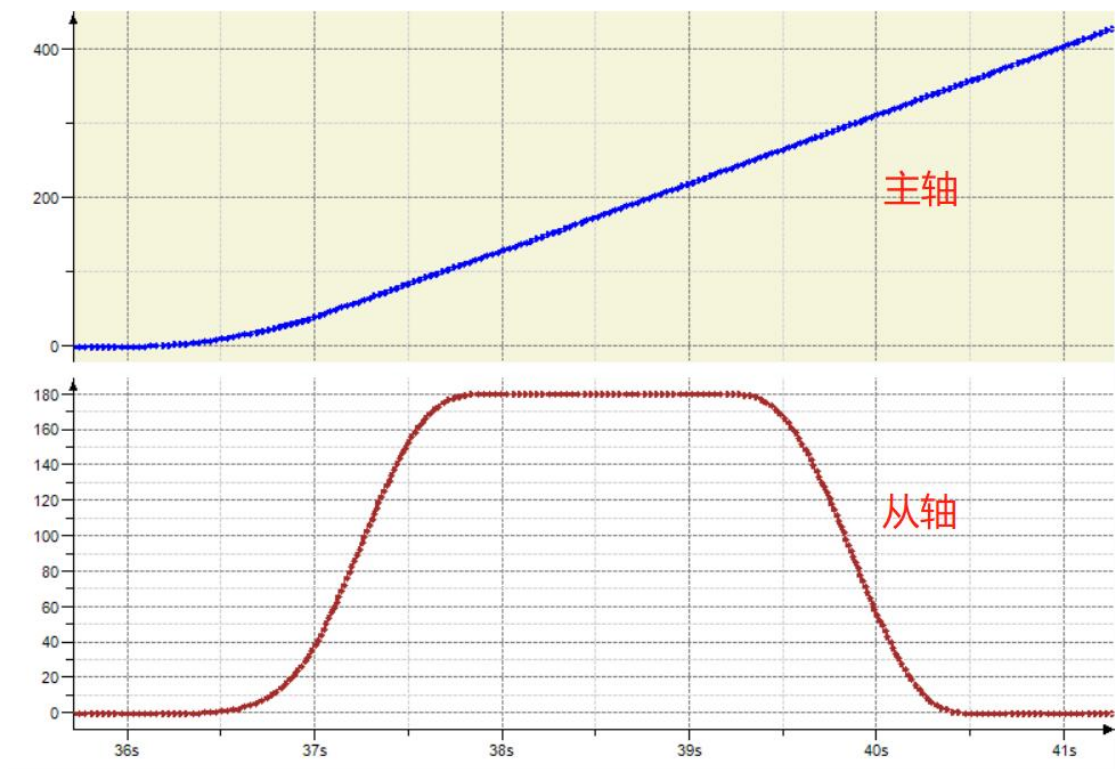


图 8.23

当 MasterScaling:=1, MasterOffset:=0, SalveScaling:=1, SalveOffset:=0 时，参考曲线如上。

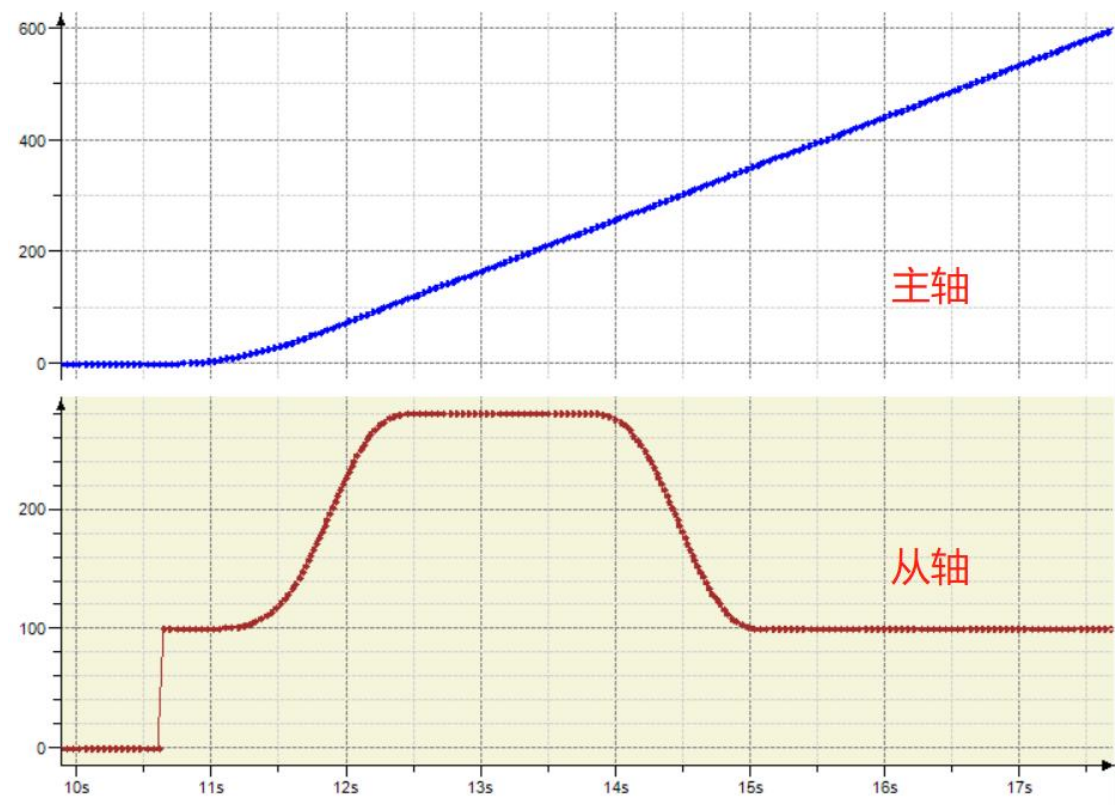


图 8.24

当 $\text{SalveScaling}=1$, $\text{SalveOffset}=100$ 时, 凸轮启动时从轴指令位置偏移 100, 此时会有一个位置跳变。

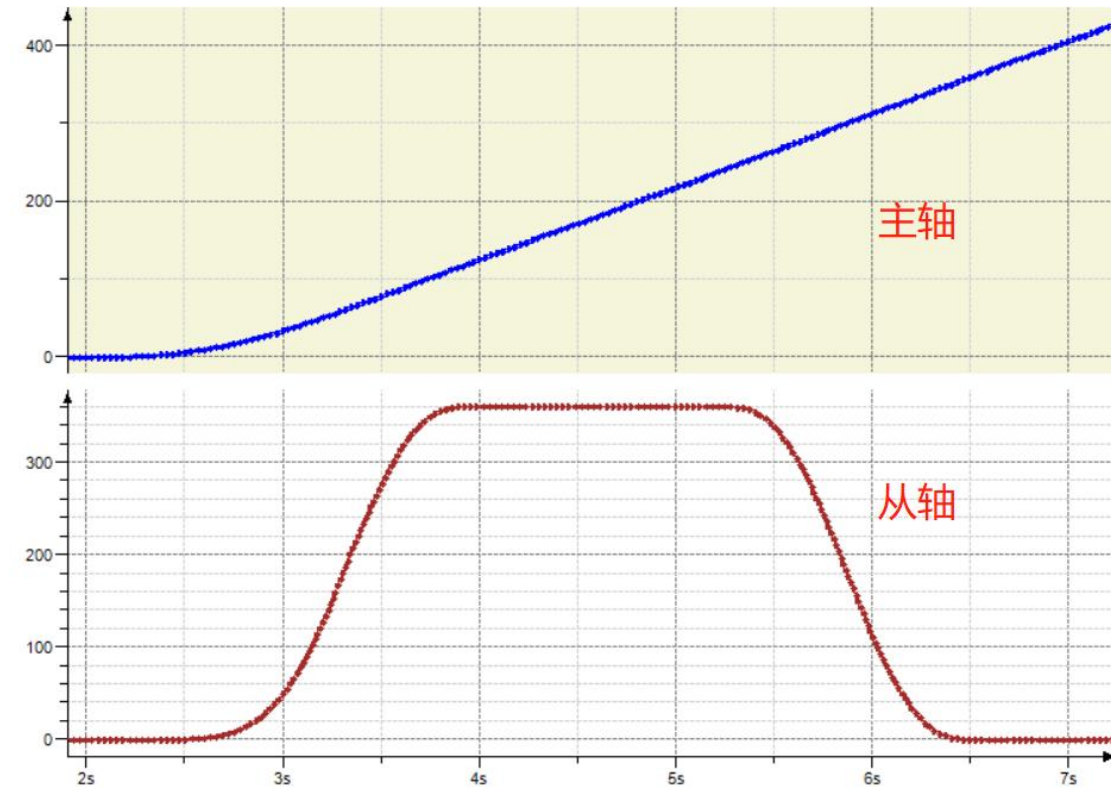


图 8.25

当 $\text{SalveScaling}=2$, $\text{SalveOffset}=0$ 时, 凸轮启动时从轴指令位置输出会放大两倍, 从轴的范围将会增加。

8.4.4.6 MC_CamIn.StartMode

参数与 MC_CamTableSelcet 指令中的 SalveAbsolute 综合指定从轴当前指令位置为凸轮表的绝对位置还是相对位置, 以下描述 StartMode 对凸轮运行的影响:

- 0: 绝对模式, 在电子凸轮启动时, 电子凸轮的计算与当前从轴的位置无关。如果从轴当前位置与凸轮计算出的从轴位置不相同, 从轴的位置将存在跳变。
- 1: 相对模式: 在电子凸轮启动时, 电子凸轮的计算会根据当前从轴的位置进行改变; 即启动时从轴位移为 0。如果在凸轮表规划中, 与主轴起始位置对应的从轴位置不是 0, 从轴的位置将存在跳变。

1、3、4：斜坡输入，通过增加一个补偿运动（根据 VelocityDiff、加速度、减速度得出的运动）来防止电子凸轮在启动时的潜在跳变。

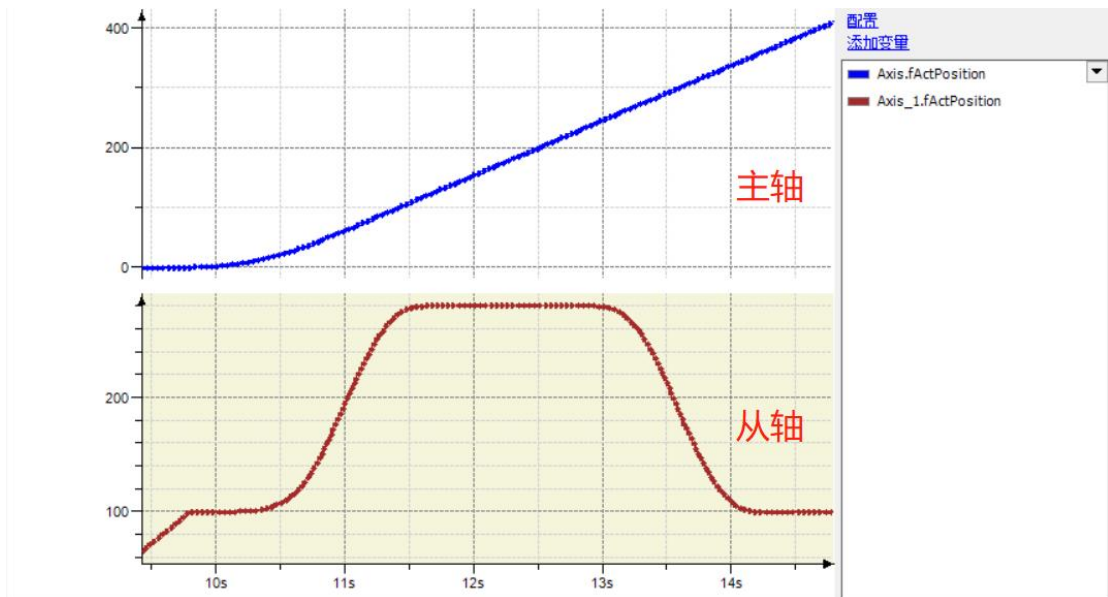


图 8.26

当 StartMode:=0, MC_CamTableSelect. SalveAbsolute:=false 时, 在从轴 factposition 位置为 100 时, 触发凸轮, 此时, 从轴运行在相对模式下。

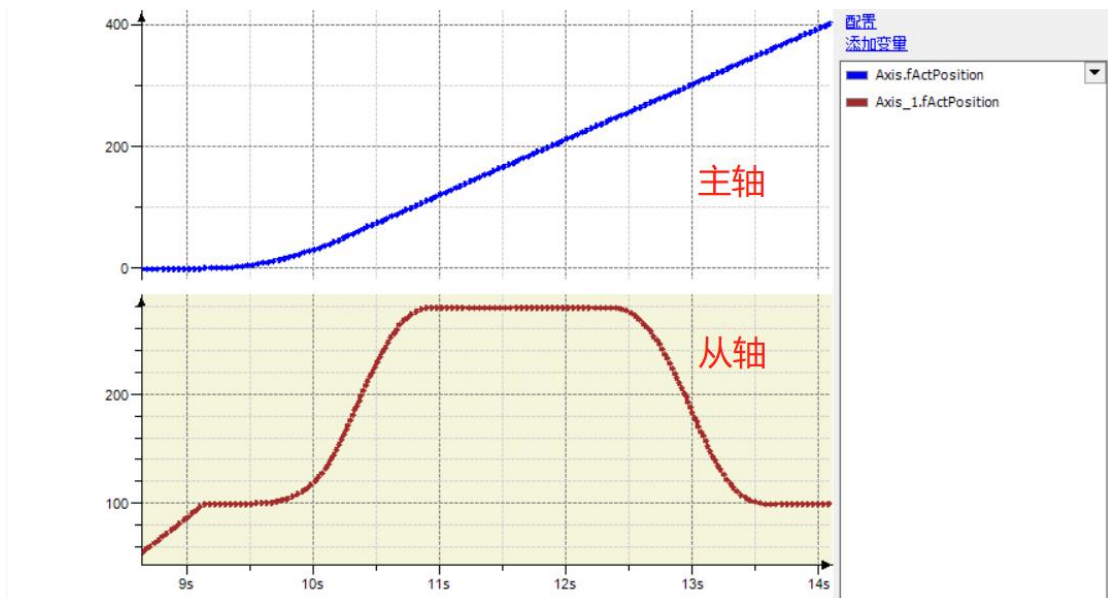


图 8.27

当 StartMode:=1, MC_CamTableSelect. SalveAbsolute:=false 时, 在从轴 factposition 位置为 100 时, 触发凸轮, 此时, 从轴运行在相对模式下。

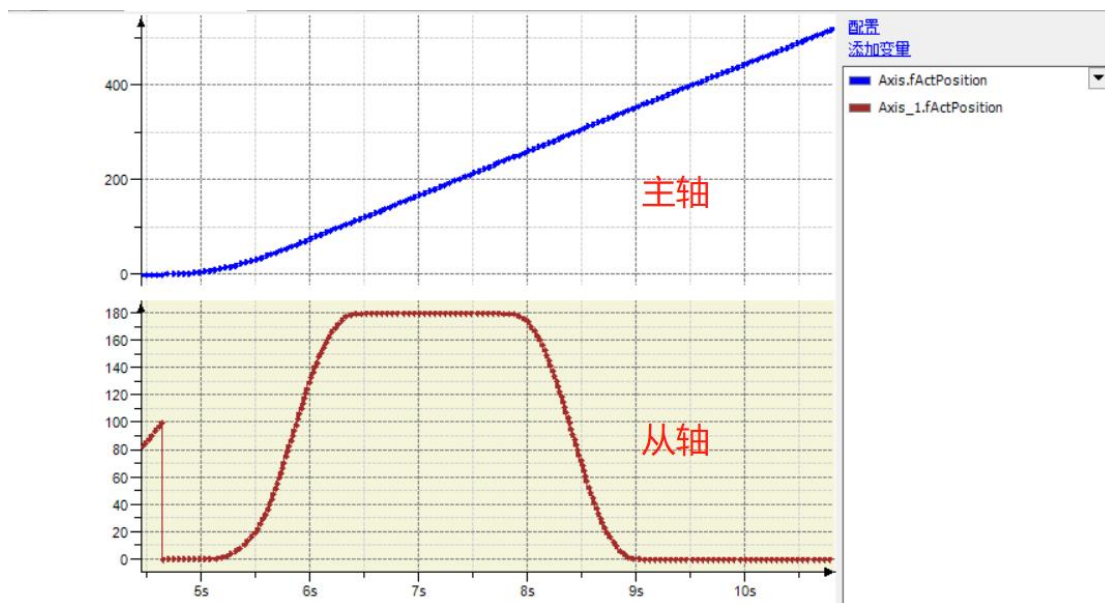


图 8.28

当 $\text{StartMode}=0$ ， $\text{MC_CamTableSelect.SalveAbsolute}:=\text{True}$ 时，在从轴 factposition 位置为 100 时，触发凸轮，此时，从轴运行在绝对模式下。

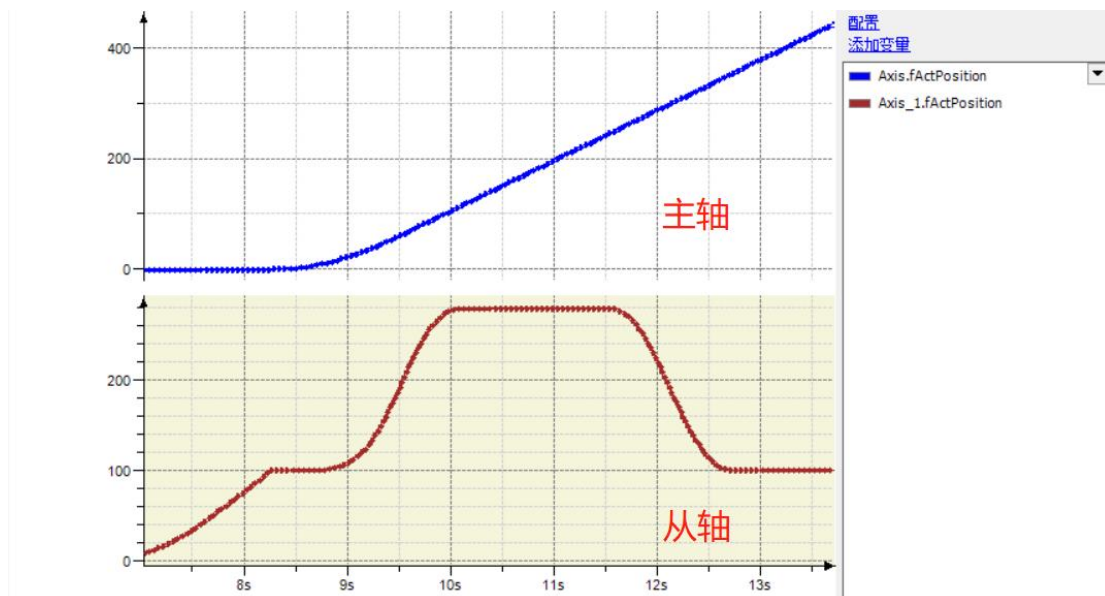


图 8.29

当 $\text{StartMode}=1$ ， $\text{MC_CamTableSelect.SalveAbsolute}:=\text{True}$ 时，在从轴 factposition 位置为 100 时，触发凸轮，此时，从轴运行在相对模式下。

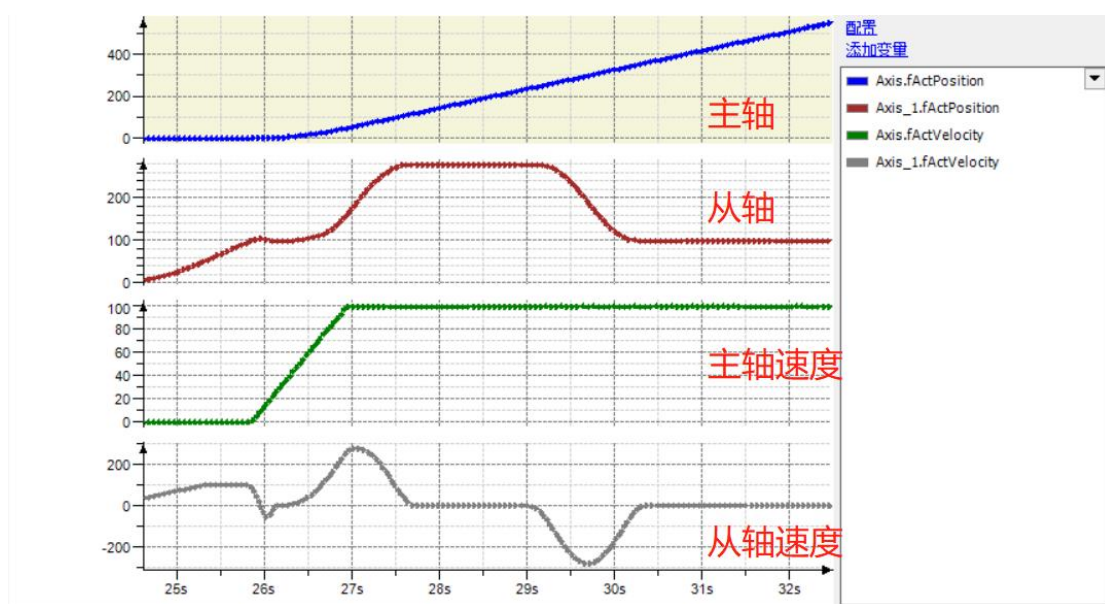


图 8.30

当 $\text{StartMode}:=2$ ， $\text{MC_CamTableSelect.SalveAbsolute}:=\text{False}$ 时，在从轴 factposition 位置为 100 时，触发凸轮，此时，从轴运行在相对模式下，并且叠加一个补偿运动，用于避免切入时的潜在跳变（补偿运动的计算与参数 VelocityDiff, Acceleration, Deceleration 相关）。

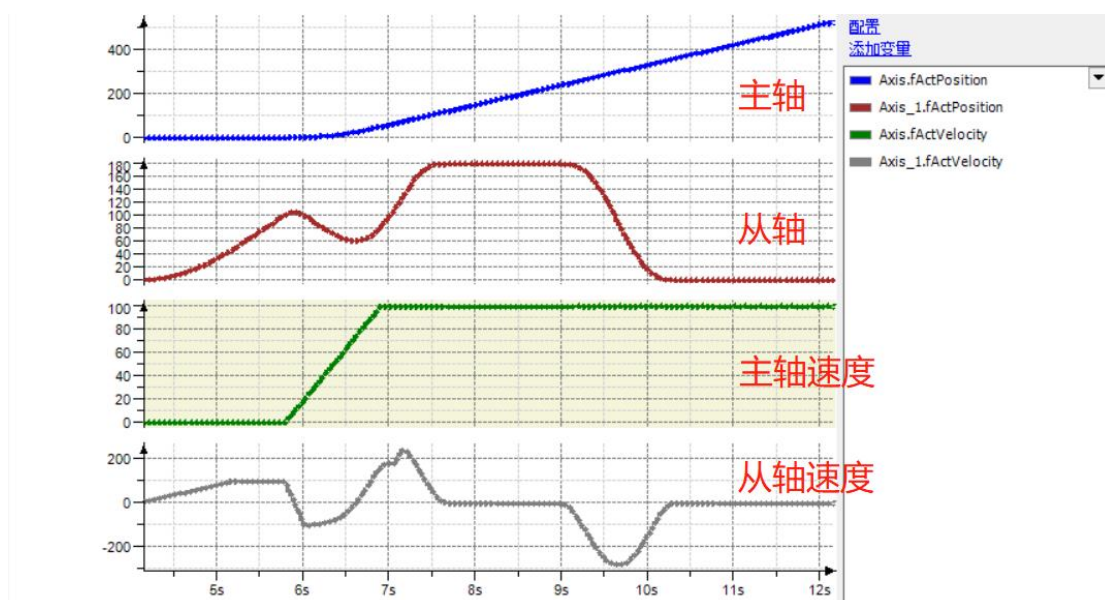


图 8.31

当 $\text{StartMode}:=2$ ， $\text{MC_CamTableSelect.SalveAbsolute}:=\text{True}$ 时，在从轴 factposition

位置为 100 时，触发凸轮，此时，从轴运行在绝对模式下，并且叠加一个补偿运动，用于避免切入时的潜在跳变（补偿运动的计算与参数 VelocityDiff, Acceleration, Deceleration 相关）。

StartMode:=3、4 时，从轴为“旋转模式”则 ramp_in_pos 只会沿着轴正向运动方向补偿，ramp_in_neg 只会沿着轴反向运动方向补偿，当轴工作为线性模式时 ramp_in 、ramp_in_pos、ramp_in_neg 三种启动模式工作情况是一样的。

总结 MC_CamIn.StartMode 与 MC_CamTableSelect. SalveAbsolute 对电子凸轮从轴运行的影响如下：

表 8.5

MC_CamIn.StartMode	MC_CamTableSelect. SalveAbsolute	从轴运行模式
absolute	True	绝对模式
absolute	False	相对模式
relative	True	相对模式
relative	False	相对模式
ramp_in	True	斜坡接入绝对模式
ramp_in	False	斜坡接入相对模式
ramp_in_pos	True	正向斜坡接入绝对模式
ramp_in_pos	False	正向斜坡接入绝对模式
ramp_in_neg	True	反向斜坡接入绝对模式
ramp_in_neg	False	反向斜坡接入绝对模式

8.4.5. 挺杆功能

挺杆可以实现在主轴的指定位置控制变量的输出状态的功能。

8.4.5.1 挺杆数据结构

SMC_CAMTappet（挺杆数据结构）

表 8.6

变量	数据类型	初始值	描述
ctt	SMC_CAMTAPPET TYPE	0	挺杆类型，决定了必须沿哪个方向通过挺杆才能运行。
cta	SMC_CAMTAPPET ACTION	0	挺杆动作方式，决定了当通过挺杆时执行什么切换动作
dwDelay	DWORD	0	当 cta=TAPPETACTION.TAPPET ACTION_time 时，该值确定以 μs 为单位的延迟时间。
dwDuration	DWORD	0	当 cta=TAPPETACTION.TAPPET ACTION_time 时，该值确定以 μs 为单位的挺杆打开时间。
iGroupID	INT	0	被切换的挺杆输出的组或轨道 ID
x	LREAL	0	挺杆切换时主轴的位置
dwActive	DWORD	0	内部变量

8.4.5.2挺杆设置

用户可在 CAM 配置的“挺杆”和“挺杆表”界面对挺杆的参数进行配置。

1) “挺杆”界面中可以修改当前添加的挺杆，支持通过拖动修改挺杆动作位置。

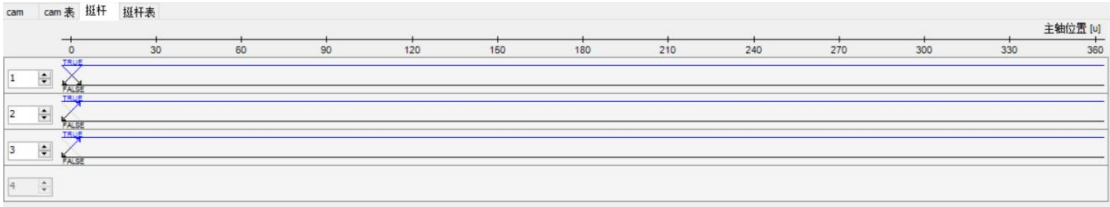


图 8.32

用户可以通过点击来修改挺杆的动作类型，一共有以下几种组合：

表 8.7

图例	正向通过	反向通过
	无动作	无动作
	切换到开启	无动作
	切换到关闭	无动作
	无动作	切换到开启
	无动作	切换到关闭
	切换到开启	切换到关闭
	切换到开启	切换到关闭
	切换到关闭	切换到开启
	切换到关闭	切换到关闭
	取反	无动作
	无动作	取反

	切换到开启	取反
	取反	切换到开启
	取反	切换到关闭
	切换到关闭	取反
	取反	取反

2) “挺杆表”界面中可以添加及修改挺杆的触发位置，其中 X 即为凸轮中的主轴相位。

cam	cam 表	挺杆	挺杆表	
		跟踪ID	X	正向通过 反向通过
		1		
			100	关闭 关闭
		2		
			200	反转 反转
		3		
			300	打开 关闭
				

图 8.33

8.4.5.3 挺杆使用步骤

程序中使用 SMC_GetTappetValue 或 SMC_CamRegister 来读取挺杆的输出，但是两者存在冲突，所以在同一个凸轮挺杆表中只使用一种。此处以 SMC_GetTappetValue 为例介绍挺杆的使用步骤。

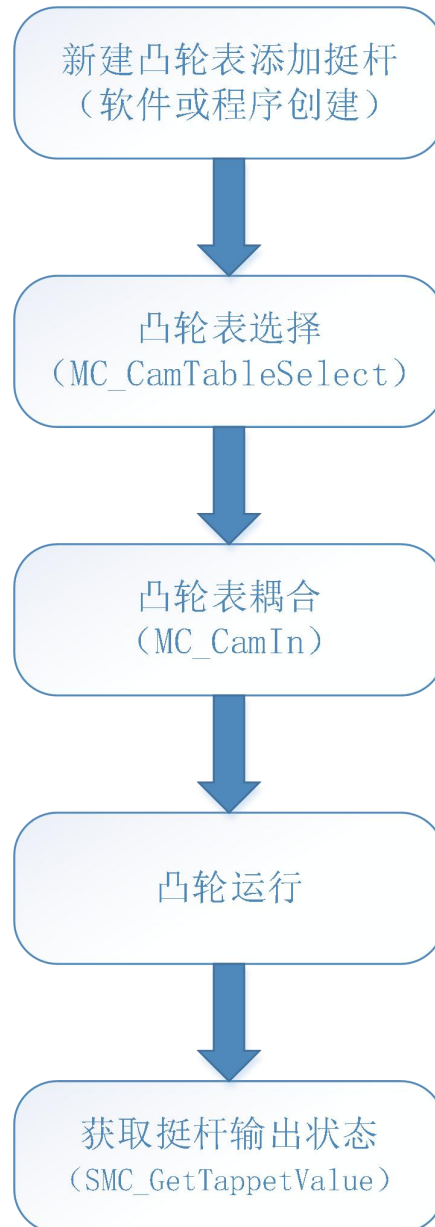


图 8.34

第九章 离线仿真

在调试程序过程中，使用 LeadSys 软件的离线仿真功能十分方便。

离线仿真，即运动控制器、电机不需要上电，PC 机独立运行程序，就可以模拟系统的运行结果。

打开工程，在菜单中选择“在线”→“仿真”，如图 9.1 所示；程序编译后，在菜单中选“登录到”，弹出对话框如图 9.2 所示，选择“是”。点击菜单中的“运行”，即开始仿真。

如图 9.3 所示，在主程序界面，用鼠标左键双击程序第 1 行中的变量 In10，然后按 Ctrl+F7 键，相当于开关 In10 闭合，程序做连续插补，得到仿真结果如图 9.4 所示。

仿真曲线数据和实际曲线一致，只是更光滑些。

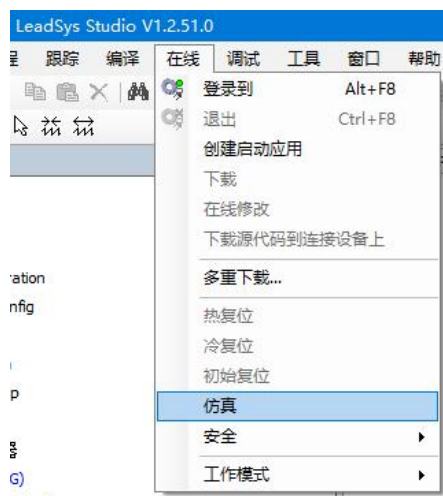


图 9.1 选择离线仿真

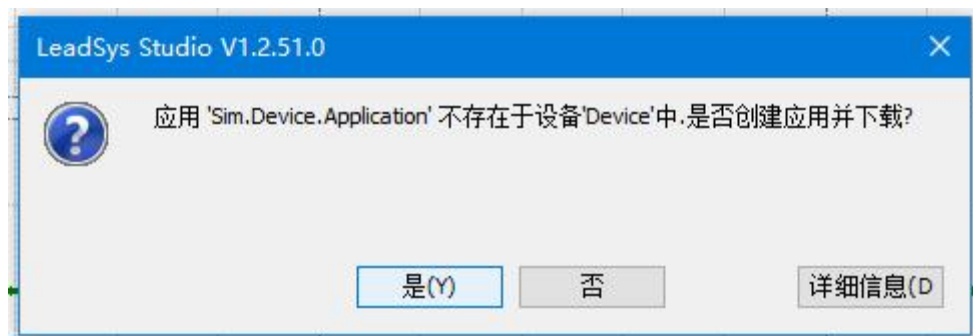


图 9.2 下载仿真程序

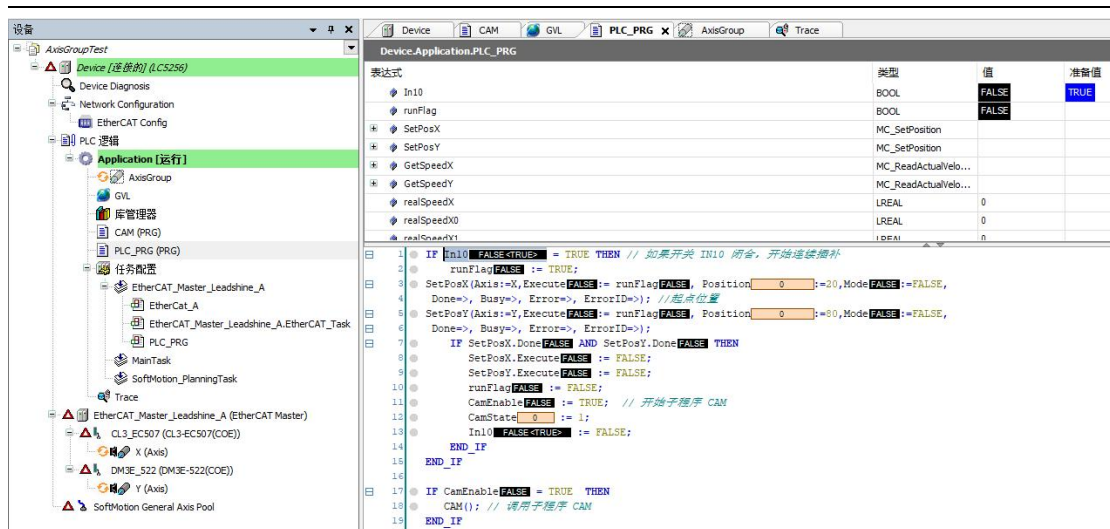


图 9.3 手动设置输入变量 In10 的值

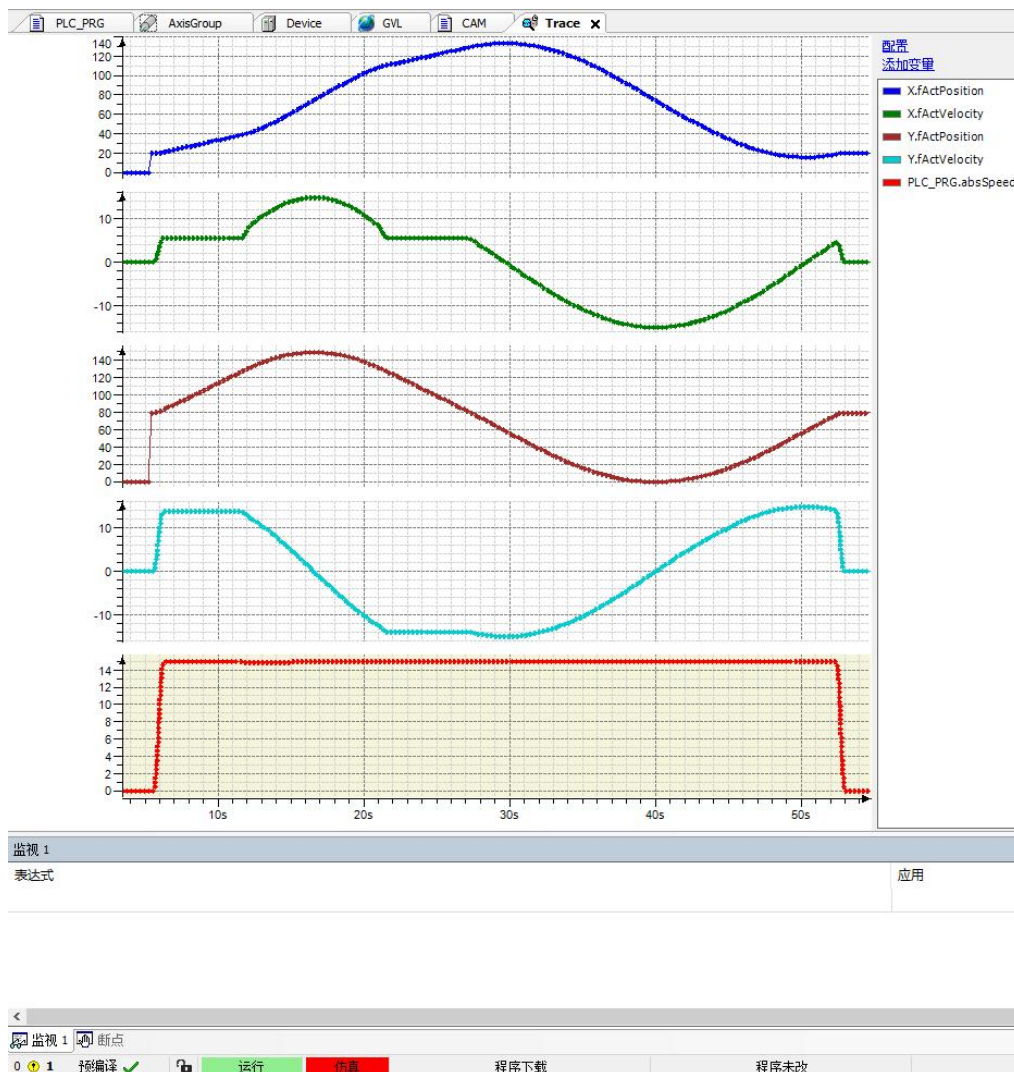


图 9.4 连续插补的仿真结果

第十章 高速输入输出功能

本节介绍雷赛智能 MC500、PMC600、LC1000、LC5000IO 系列产品的高速输入输出功能。

10.1. 高速计数器简介

10.1.1. MC500 系列产品高速 IO 介绍

MC516 及 MC532 均配有 12 路（IN0~IN11）响应频率为 200K 的单端高速输入端口，可支持 A/B 相 1/2/4 倍频、脉冲+方向、单相及 CW/CCW 计数，详细规格见表 10.1。

表 10.1 MC516 及 MC532 高速输入端口规格

项目		规格
高速输入点数		12 点
输入形式		直流（漏型或源型）
输入电压/电流		24V(-15% - +20%),5mA
动作 位准	OFF→ON	高于 DC 15V，电流 4.5mA 以上
	ON→OFF	低于 DC 5V，电流 1mA 以下
输入 频率	IN0~IN11	200 kHz 高速输入（单端）
输入阻抗		2.7K Ω
输入保护		光电耦合隔离、抗干扰滤波
输入动作显示		输入接通时 LED 灯亮
输入公共端		每 8 点使用一个公共端 S/S

10.1.2. PMC600 系列产品高速 IO 介绍

PMC600 配有 12 路（IN0~IN11）响应频率为 200K 的单端高速输入端口，可支持 A/B 相 4 倍频、脉冲+方向及单相计数，详细规格见表 10.2。

表 10.2 PMC600 高速输入端口规格

项目		规格
高速输入点数		12 点
输入形式		直流（漏型或源型）
输入电压/电流		DC24V(-15 - +20), 5mA
动作 位准	OFF→ON	高于 DC 15V，电流 3mA 以上
	ON→OFF	低于 DC 5V，电流 1mA 以下
输入 频率	IN0~IN11	200 kHz 高速输入（单端）
输入阻抗		4.7K Ω
输入保护		光电耦合隔离、抗干扰滤波
输入动作显示		输入接通时 LED 灯亮
输入公共端		每 8 点使用一个公共端 S/S

PMC610 配有 16 路（IN0~IN11 及 IN16~IN19）响应频率为 200K 的单端高速输入端口。其中 IN0~IN11 可支持 A/B 相 4 倍频、脉冲+方向及单相计数；IN16~IN19 可支持锁存功能使用。除此以外，还配有 3 组差分编码器输入口，分别为 EA0+/-、EB0+/-、EZ0+/-、EA1+/-、EB1+/-、EZ1+/-及 EA2+/-、EB2+/-、EZ2+/-，详细规格见表 10.3。

表 10.3 PMC610 高速输入端口规格

项目	规格
----	----

高速输入点数		16 点
输入形式		直流（漏型或源型）
输入电压/电流		DC24V(-15 ~ +20), 5mA
动作 位准	OFF→ON	高于 DC 15V，电流 3mA 以上
	ON→OFF	低于 DC 5V，电流 1mA 以下
输入 频率	IN0~IN11	200 kHz 高速输入（单端）
	IN16~IN19	
	IN12~IN15	10 kHz 普通输入（单端）
编码器		支持 4MHZ 输入（差分）
输入阻抗		4.7K Ω
输入保护		光电耦合隔离、抗干扰滤波
输入动作显示		输入接通时 LED 灯亮
输入公共端		每 8 点使用一个公共端 S/S

10.1.3. LC1000 系列产品高速 IO 介绍

LC1000 配有 8 路（IN0~IN7）响应频率为 200K 的单端高速输入端口，可支持 A/B 相 4 倍频、脉冲+方向、CW/CCW 及单相计数，详细规格见表 10.4。

表 10.4 LC1000 高速输入端口规格

项目		规格
高速输入点数		8 点
输入形式		直流（漏型或源型）
输入电压/电流		24V(-15% - +20%),5mA
动作 位准	OFF→ON	高于 DC 15V，电流 1.5mA 以上
	ON→OFF	低于 DC 5V，电流 1mA 以下

输入 频率	IN0~IN7	200 kHz 高速输入（单端）
输入阻抗	4.3K Ω （最大值）	
输入保护	光电耦合隔离、抗干扰滤波	
输入动作显示	输入接通时 LCD 屏中指示灯亮	
输入公共端	每 8 点使用一个公共端 S/S	

10.1.4. LC5000IO 产品高速 IO 介绍

LC5000IO 配有 8 路（IN0~IN7）响应频率为 200K 的单端高速输入端口，可支持 A/B 相 4 倍频、脉冲+方向、CW/CCW 及单相计数，详细规格见表 10.5。

表 10.5 LC5000IO 高速输入端口规格

项目		规格
高速输入点数		8 点
输入形式		直流（漏型或源型）
输入电压/电流		24V(-15% - +20%),5mA
动作 位准	OFF→ON	高于 DC 15V，电流 1.5mA 以上
	ON→OFF	低于 DC 5V，电流 1mA 以下
输入 频率	IN0~IN7	200 kHz 高速输入（单端）
输入阻抗		4.7K Ω （最大值）
编码器		支持 4MHZ 输入（差分）
输入保护		光电耦合隔离、抗干扰滤波
输入公共端		每 6 点使用一个公共端 S/S

10.2. 界面组态

10.2.1. PMC600 高速输入端口组态界面

图 10.1 为 PMC600 高速输入端口的组态界面，图中标注项解释如下：

- ① 鼠标左键双击“High_Speed_IO_Module(High Speed IO Module)”，弹出 High Speed IO Module 设置界面；
- ② 鼠标左键单击“硬件端口配置 1”进入配置界面；
- ③ 若要使用计数器功能，勾选“计数器 0”，则相应的输入端口做计数器的输入信号源使用；若不勾选，则可做普通输入端口使用。

IN00 及 IN01 为计数器 0 的输入信号源，后续计数器与高速输入端口对应关系以此类推；

- ④ “计数方式”：可选择三种计数模式，如：“A/B 相 4 倍频”、“脉冲+方向”或“单相计数”。

若选择使用“A/B 相 4 倍频”或“脉冲+方向”计数模式，IN00 及 IN01 为计数器 0 的输入信号源。选择“单相计数”，则 IN00 为计数器 0 的输入信号源。

使用“单相计数”时，计数器 1 的输入信号源为 IN02，后续计数器与高速输入端口对应关系以此类推；

- ⑤ “输入计数方向”，可选择“正常”或“反向”。

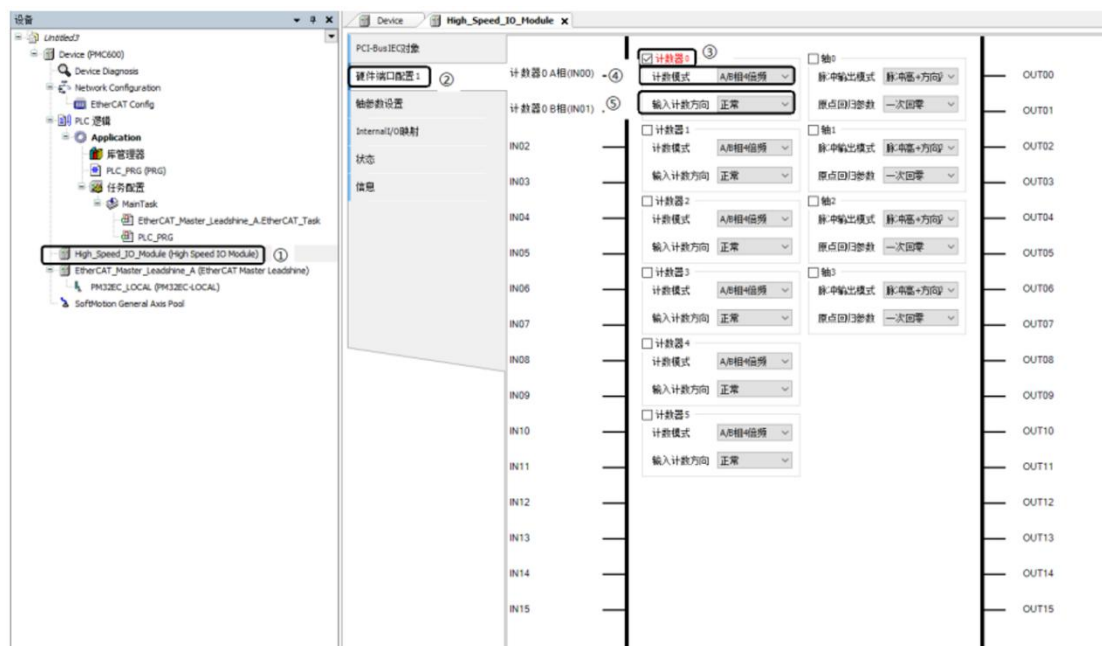


图 10.1 PMC600 高速输入端口的组态界面

10.2.2. PMC610 高速输入端口组态界面

图 10.2 为 PMC610 高速输入端口的组态界面，高速计数器端口与 PMC600 一样。以下为锁存端口的设置方法。

- ① 鼠标左键双击“High_Speed_IO_Module(High Speed IO Module);
- ② 鼠标左键单击“硬件端口配置 2”进入配置界面 2;
- ③ 若使用锁存功能，只需左键单击勾选“高速锁存 0”，则 IN16 为锁存功能 0 的信号输入源。后续的锁存与高速输入端口对应关系以此类推。若不勾选，则可做普通输入端口使用。

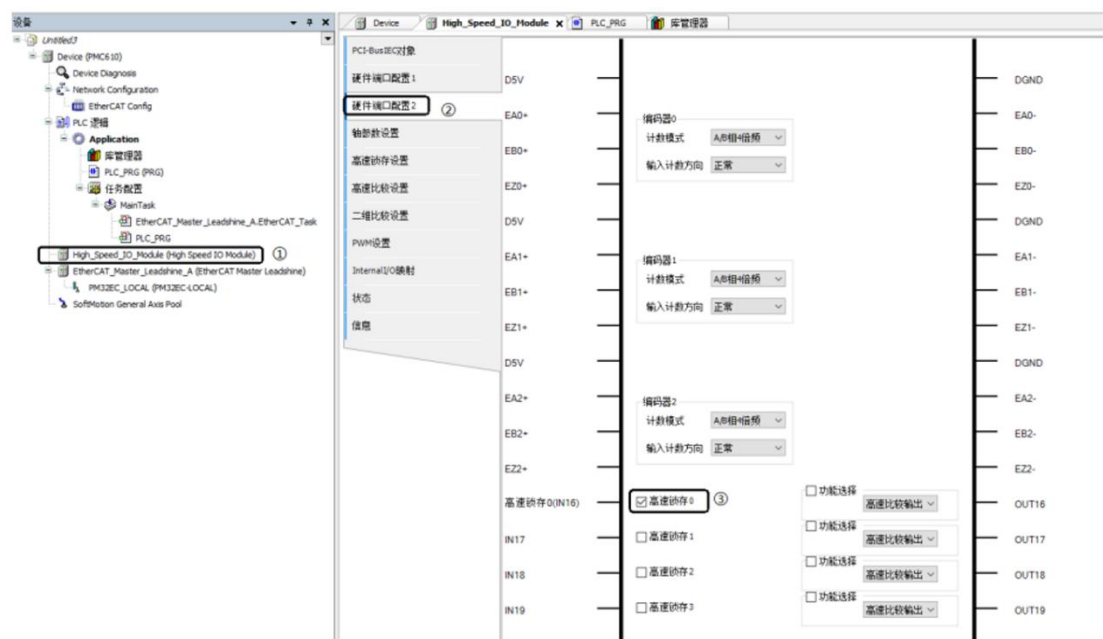


图 10.2 PMC610 高速输入端口的组态界面

10.2.3. MC500 高速输入输出端口界面

图 10.3 为 MC516 及 MC532 高速输入端口的组态界面，图中标注项解释如下：

- ① 鼠标左键双击“High_Speed_IO_Module(High Speed IO Module)”，弹出 High Speed IO Module 设置界面；
- ② 鼠标左键单击“高速输入设置”进入配置界面；
- ③ 选择所需的“计数器”；
- ④ 若要使用计数器功能，勾选“启用”，则相应的输入端口做计数器的输入信号源使用；若不勾选，则可做普通输入端口使用。

IN00 及 IN01 为计数器 0 的输入信号源，后续计数器与高速输入端口对应关系以此类推；

- ⑤ “工作模式”为计数器的计数模式，可选择 6 种计数模式，如：“A/B 相 1 倍频”、“A/B 相 2 倍频”、“A/B 相 4 倍频”、“脉冲+方向”、“单相计数”、或“CW/CCW”；

若选择使用“A/B 相 1/2/4 倍频”、“脉冲+方向”或“CW/CCW”计数模式，IN00 及 IN01 为计数器 0 的输入信号源。选择“单相计数”，则 IN00 为计数器 0 的输入信号源；

使用“单相计数”时，计数器 1 的输入信号源为 IN02，后续计数器与高速输

入端口对应关系以此类推；

- ⑥ “输入计数方向”，可选择“正常”或“反向”；
- ⑦ “计数模式”，可选择“环形”或“线性”；
- ⑧ “上限值”，默认为“2147483647”，范围“-2147483648”~“2147483647”（注意：上限值不得小于下限值，若设置值小于下限值，系统自动将下限值变成与上限值一致）；
- ⑨ “下限值”，默认为“-2147483648”，范围“-2147483648”~“2147483647”



图 10.3 MC516 及 MC532 高速输入端口的组态界面

10.2.4. LC1000/LC5000IO 高速 IO 组态界面

LC1000 高速 IO 端口的组态界面如下，如图 10.4，图中标注项解释如下：

- ① 启用计数器

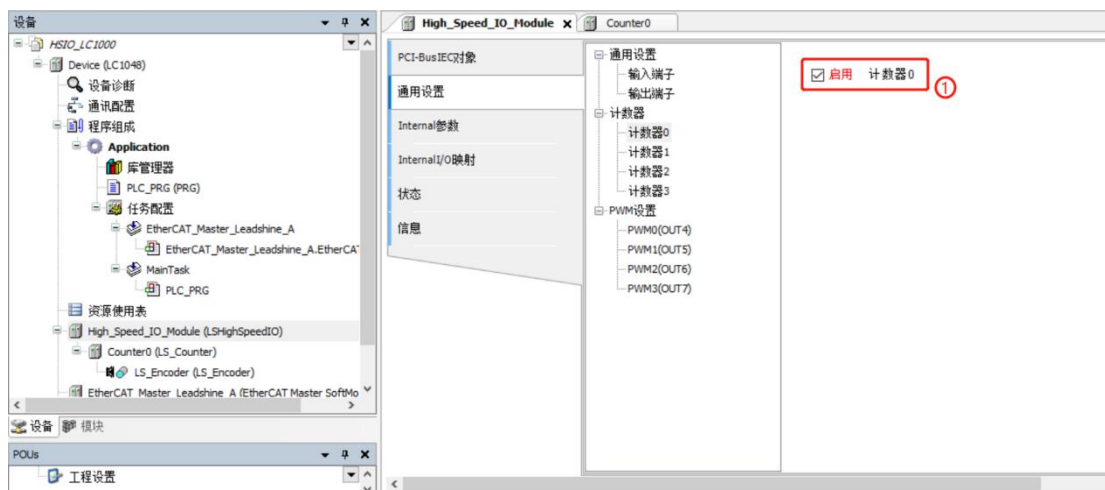


图 10.4 LC1000 高速计数器

②计数器参数配置

工作模式：可选择 6 种工作模式，包括“A/B 相 1 倍频”、“A/B 相 2 倍频”、“A/B 相 4 倍频”、“脉冲+方向”、“单相计数”和 “CW/CCW”；若选择使用“A/B 相 1/2/4 倍频”、“脉冲+方向”或“CW/CCW”计数模式，IN0 及 IN1 为计数器 0 的输入信号源。选择“单相计数”，则 IN0~IN7 中的任意一个输入可配置为计数器 0 的输入信号源；其他计数器类同。

“计数方向”：可配置“正向”计数或“负向”计数。

“计数模式”：可配置“环形”或“线性”。

“上限值”：默认为“2147483647”，范围“-2147483648”~“2147483647”（注意：上限值不得小于下限值，若设置值小于下限值，系统自动将下限值变成与上限值一致）。

“下限值”：默认为“-2147483648”，范围“-2147483648”~“2147483647”。

“预置输入口”：当预置功能块配置为外部 DI 预置模式时，需要配置该输入口，可在 IN0~IN7 中任意选择一个输入口，实现预置值功能。

“探针 0”：0 号探针输入口，每路计数器可配置 2 路探针输入口，实现锁存计数器值功能，可配置为 IN0~IN7 任一输入口。

“探针 1”： 1 号探针输入口，可配置为 IN0~IN7 任一输入口。

“比较输出信号”：使用高速比较（单点/线性/队列）输出功能时，配置该输出口，可配范围为 OUT0~OUT7。

“信号源”：配置计数器的输入口来源。



参数	值	备注
工作模式	A/B相4倍频	计数器工作模式设置
计数方向	正向	计数方向设置,只对A/B相模式有效
计数模式	线性	计数模式设置
上限值	2147483647	上限值设置,范围-2147483648~2147483647
下限值	-2147483648	下限值设置,范围-2147483648~2147483647
预置输入口	IN6	预置输入端子设置
探针0	IN2	探针0输入口设置
探针1	IN3	探针1输入口设置
比较输出信号	OUT0	比较输出信号设置
信号源	IN0-A相,IN1-B相	端口设置

图 10.5 LC1000 高速计数器参数配置

③设置计数器的缩放比例

用于配置用户单位和脉冲单位的对应关系。



图 10.6 LC1000 高速计数器缩放比例设置

④高速输入口的使用情况查看

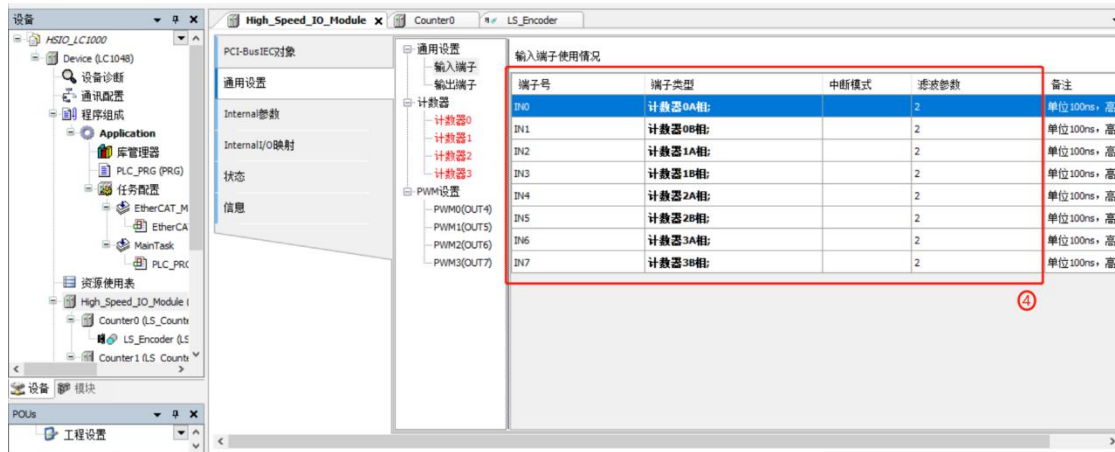


图 10.7 LC1000 高速输入使用查看

端子类型：显示该输入口已配置的功能

中断模式：当启用外部中断时使用，配置中断输入口的触发沿，包括上升沿、下降沿和任意沿。

滤波参数：配置该输入口的滤波时间，单位为 100ns。

⑤PWM 设置

程序中使用 PWM 功能块时，必须在此启动 PWM。

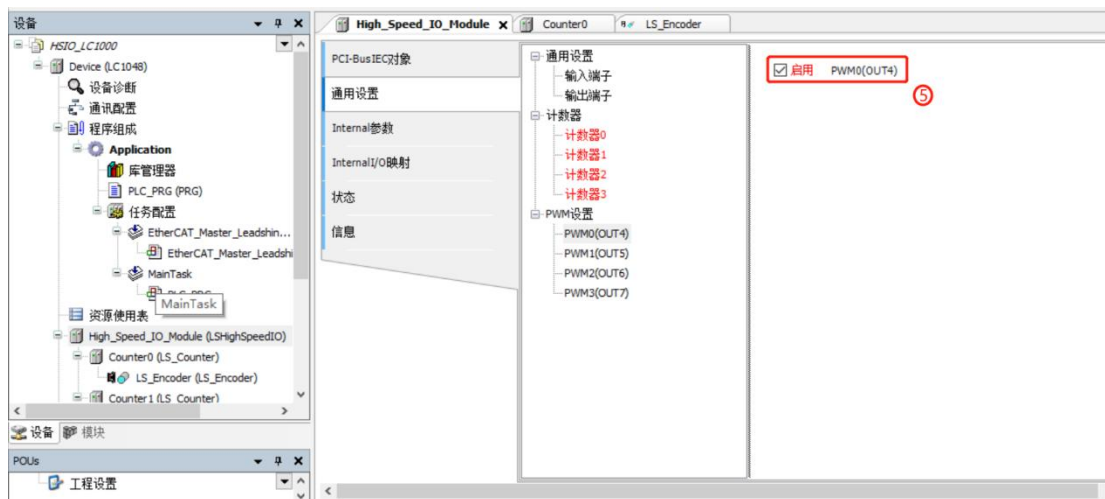


图 10.8 LC1000 PWM 输出设置

⑥高速出口的使用情况查看

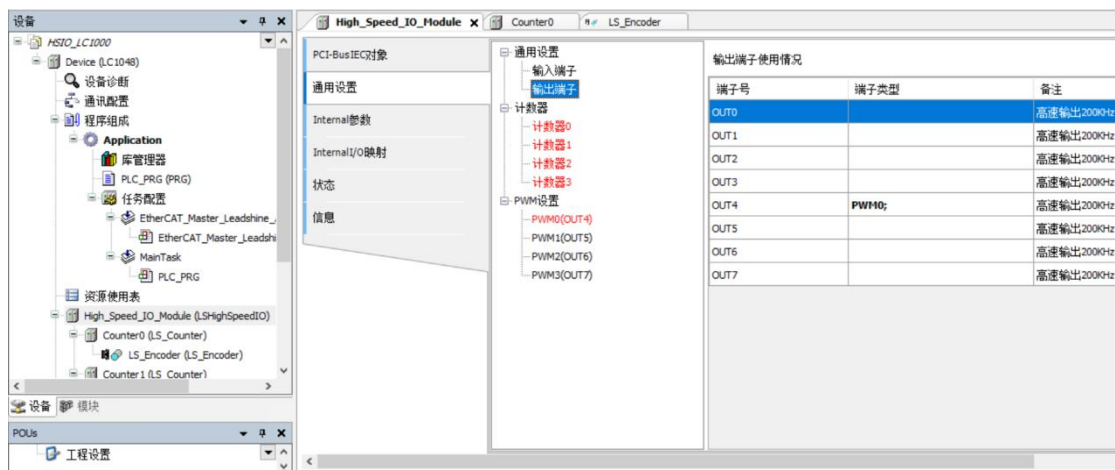


图 10.9 LC1000 高速输出使用查看

10.3. 高速计数器

10.3.1. PMC600 高速计数器指令及功能

表 10.6 PMC600 及 PMC610 高速计数器指令及功能

指令名	功能说明
LS_Counter_Clear	计数器清零功能
LS_Counter_GetNum	计数器计数值读取功能
LS_Counter_GetWorkMode	计数器计数模式读取功能

LS_Counter_SetWorkMode	计数器计数模式设置功能
LS_Counter_Clear_FB	计数器清零功能块
LS_Counter_GetNum_FB	计数器计数值读取功能块
LS_Counter_GetWorkMode_FB	计数器计数模式读取功能块
LS_Counter_SetAxisMove	计数器数值转化成指定轴运动功能块
LS_Counter_SetWorkMode_FB	计数器计数模式设置功能块

10.3.2. MC500 高速计数器指令及功能

表 10.7 MC516 及 MC532 高速计数器指令及功能

指令名	功能说明
LS_Counter	计数器计数值读取功能
LS_PresetValue	计数器计数值预置功能

10.3.3. LC1000/LC5000IO 高速计数器指令及功能

表 10.8 LC1000/LC5000IO 高速计数器指令及功能

指令名	功能说明
LS_Counter	计数器计数值读取
LS_PresetValue	预置计数值

10.3.4. PMC600 实现高速计数器功能的方法

下面通过具体例程说明高速计数器的使用方法。

a.使用 PMC600 本体的高速输出口发脉冲，并将其输入给本体的高速计数器，实现计数功能

1) 在 LeadSys Studio 软件界面选择所使用的计数器及高速输出口，设置计数模式、输入计数方向及高速输出模式，如图 10.10 所示。

① 鼠标左键双击“High_Speed_IO_Module”(High Speed IO Module);

② 左键单击“硬件端口配置 1”进入配置界面。勾选“计数器 2”;

计数模式为“AB 相 4 倍频”，输入方向选为“正常”;

勾选“轴 0”作为高速输出使用，脉冲输出模式为“AB 相”，原点回归参数选择默认的“一次回零”即可（本例程不使用回零功能）；



图 10.10 设置 PMC600 高速输入端口为计数器 2 和轴 0

2) 新建如图 10.11 所示的变量。

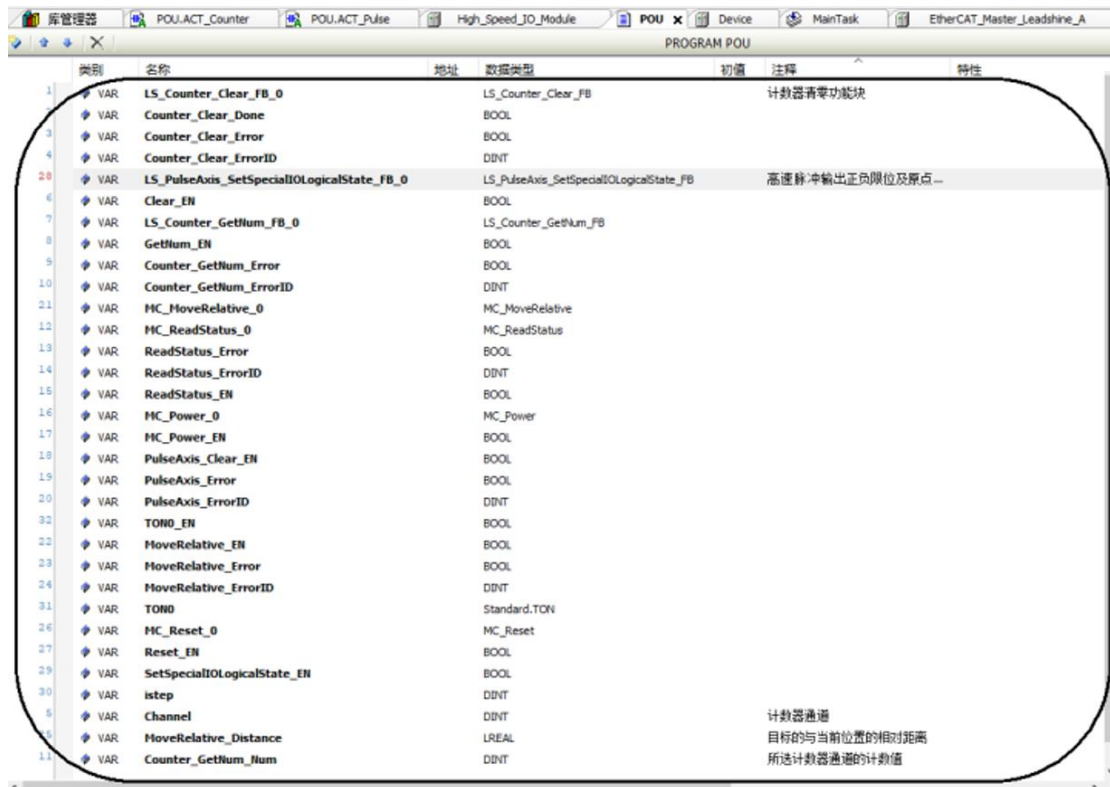


图 10.11 创建变量

3) 编写计数器清零及计数器计数功能块，如图 10.12 所示。

① 计数器清零功能块

② 计数器计数功能块

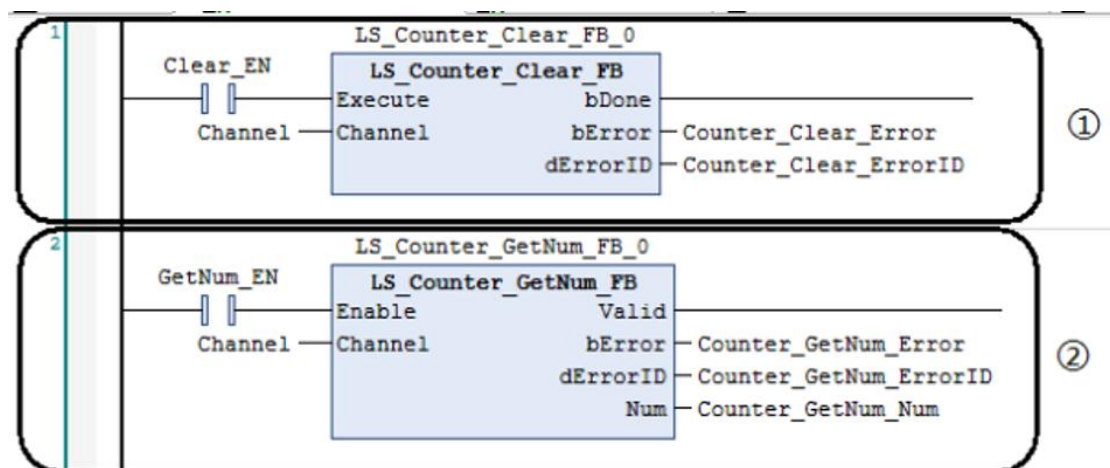


图 10.12 计数器清零模块和计数模块

4) 编写脉冲输出口的功能块

图 10.13 中标注的模块功能为：

- ① 读取轴的详细状态
- ② 设置脉冲轴的特殊 IO 信号的启用与逻辑电平
- ③ 复位轴

图 10.14 中的标注的模块功能为：

- ④ 轴使能
- ⑤ 控制 PMC600/610 主机脉冲轴的脉冲输出。在使用脉冲轴前必须调用，否则没有脉冲输出
- ⑥ 轴相对运动指令

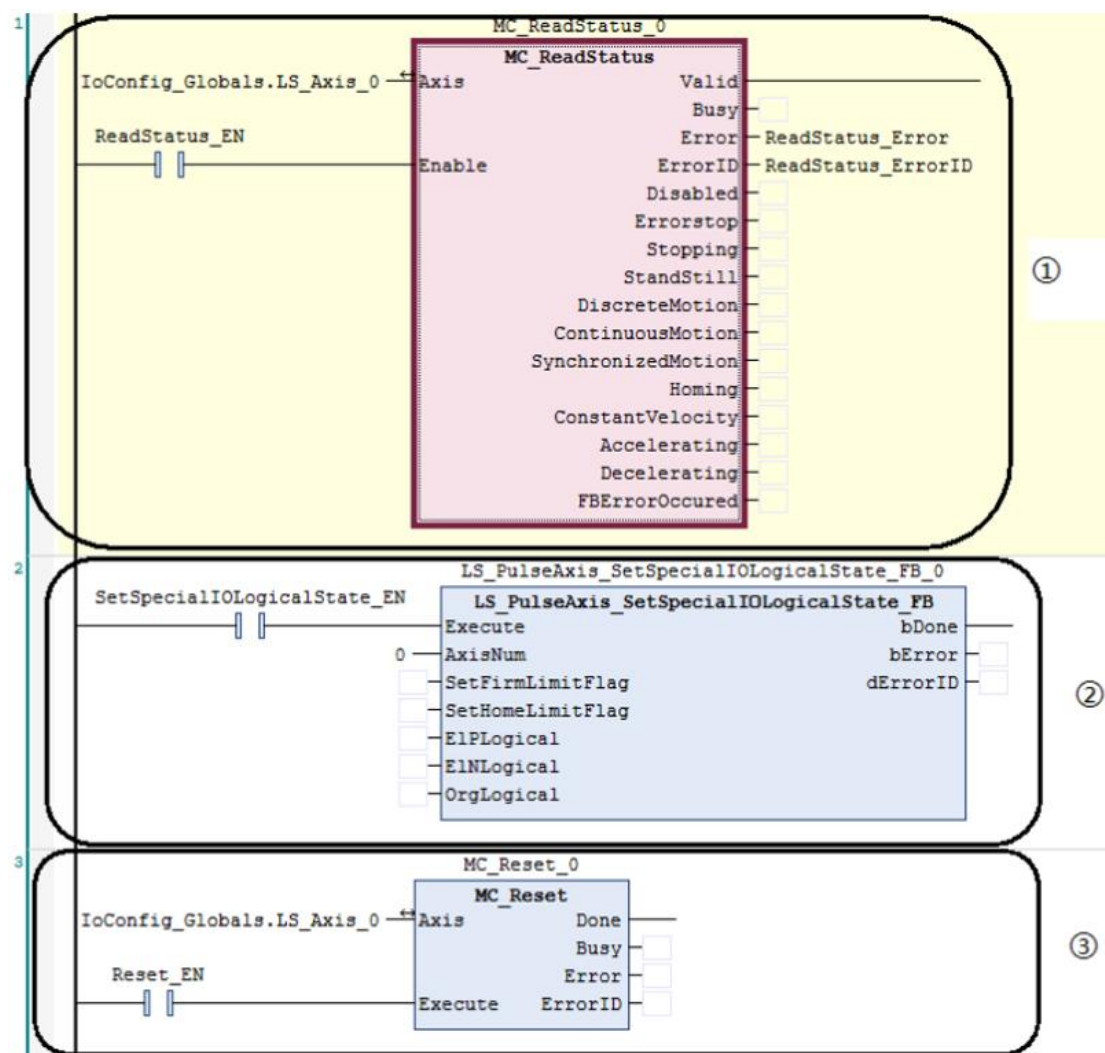


图 10.13 总线读取、脉冲端口逻辑设置及复位功能块

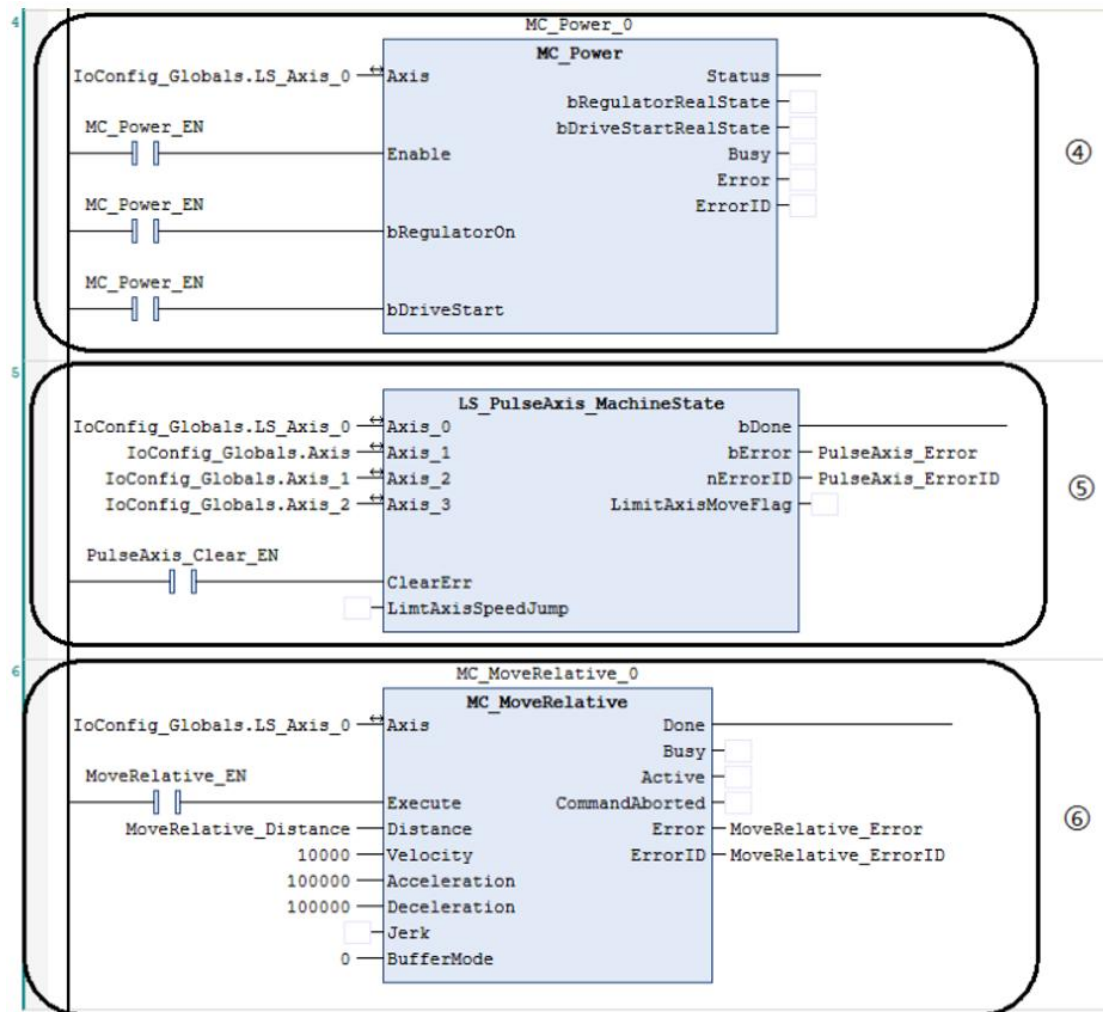


图 10.14 使能、脉冲轴绑定及相对运动功能块

5) 编写逻辑控制程序。

图 10.15 为逻辑控制程序，图中标注的程序解释如下：

- ① 设置相对运动距离为 100000，计数器通道选择为通道 2
- ② 对计数器进行清零操作
- ③ 触发计数器功能块进行计数、设置脉冲硬件限位无效及轴使能
- ④ 触发脉冲输出功能块进行脉冲输出，输出完成后复位，图 10.16 为计数器模块运行的结果。图中标注的模块解释如下
- ⑤ 计数器成功对输出脉冲数进行计数，由于计数器设置为 AB 相 4 倍频，所以计数值为 400000，计数器计数成功

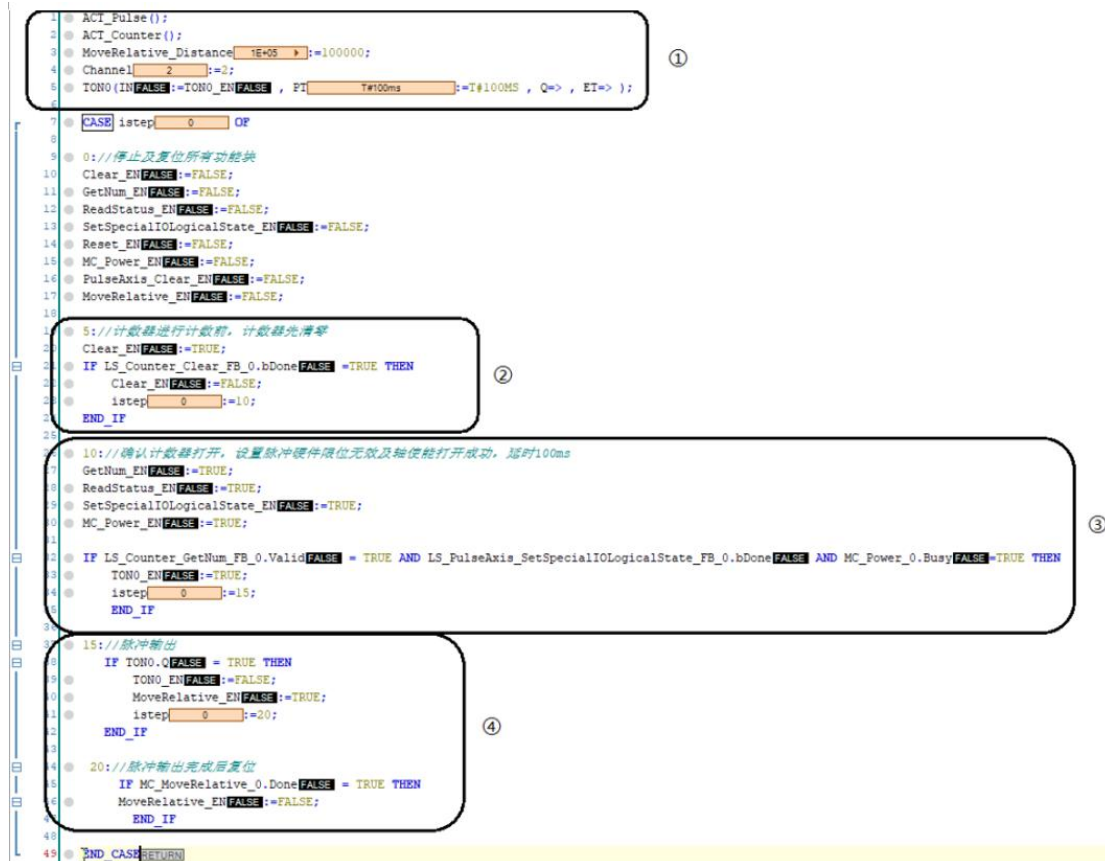


图 10.15 逻辑控制程序

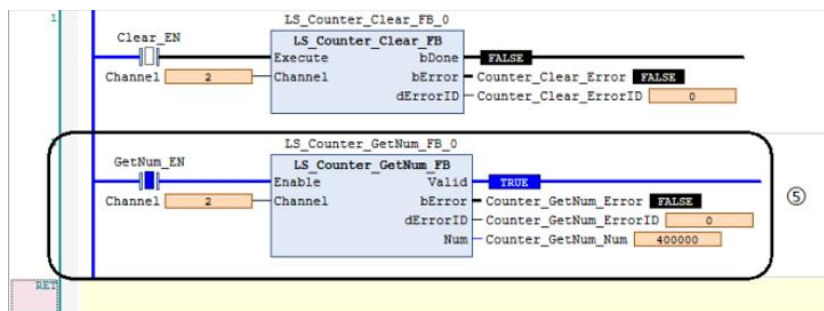


图 10.16 计数器的运行情况

10.3.5. MC500 实现高速计数器功能的方法

以 MC500 系列产品 MC532 为例说明,使用 MC532 本体的高速输出口发脉冲,并将其输入给本体的高速计数器,实现计数功能。

- 1) 在 LeadSys Studio 软件界面选择所使用的计数器及高速输出端口, 设置

计数模式、输入计数方向及高速输出模式，如图 10.17 及图 10.18 所示。

- ① 鼠标左键双击“High_Speed_IO_Module”(High Speed IO Module);
- ② 左键单击“轴参数设置”进入高速输出端口配置界面，选“轴 0”，勾选“启用”，脉冲输出模式为“AB 相”，其余设置采用默认参数；
- ③ 左键单击“高速输入设置”进入高速输入端口配置界面，选“计数器 0”，勾选“启用”，工作模式为“A/B 相 1 倍频”，计数方向为“正向”，计数模式为“线性模式”，上限值及下限值采用默认的最大计数范围。

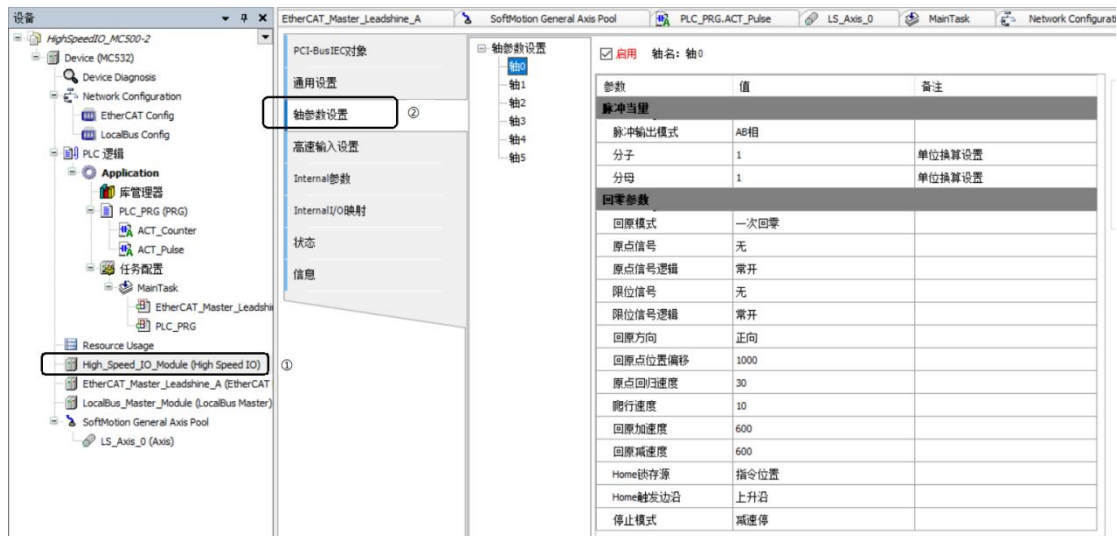


图 10.17 高速输出端口设置



图 10.18 高速输入端口设置

2) 创建如图 10.19 所示的变量。

```

1  PROGRAM PLC_PRG
2  VAR
3      LS_MotionControl_P_Clear:BOOL;
4      LS_ClearCounter_Execute:BOOL;
5      LS_Counter_0: MC_HSI0.LS_Counter;
6      LS_Counter_Execute:BOOL;
7      MC_MoveRelative_0: MC_MoveRelative;
8      MC_MoveRelative_Execute:BOOL;
9      MC_MoveRelative_Error:BOOL;
10     MC_MoveRelative_ErrorID:DINT;
11     PulAxis_0: DUT_Pulse_Axis;
12     LS_PresetCounterValue_Execute:BOOL;
13     MC_Power_0: MC_Power;
14     MC_Power_Enable:BOOL;
15     MC_Reset_0: MC_Reset;
16     MC_Reset_Execute:BOOL;
17     MC_Stop_0: MC_Stop;
18     MC_Stop_Execute:BOOL;
19     istep:DINT;
20     TON0: Standard.TON;
21     TON0_EN: BOOL;
22     LS_PresetValue_0: LS_PresetValue;
23 END_VAR

```

图 10.19 创建变量

3) 编写计数器预置及计数器计数功能块，如图 10.20。

- ① 计数器预置功能块（清零则可预置“0”，清零与预置值功能使用方式一致，只是预置值的不同）
- ② 计数器计数功能块

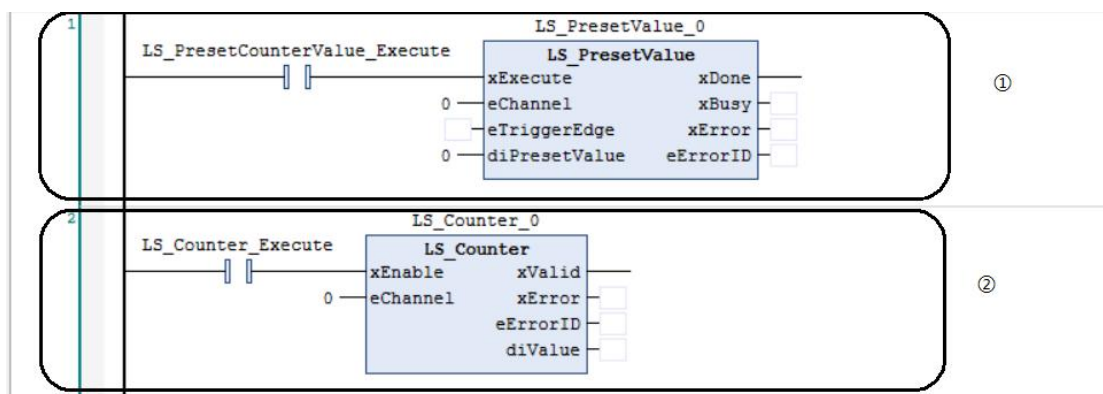


图 10.20 计数器预置功能块及计数器计数功能块

4) 编写脉冲输出口的功能块

图 10.21 及图 10.22 中标注的模块功能为：

- ① MC516 及 MC532 脉冲轴绑定功能（使用脉冲轴时必须调用），当模块内发生报错，必须先调用 MC_Reset 清除轴报错状态，再触发 xClearRrr 清除模块内错误
- ② 轴使能
- ③ 轴相对运动功能块

图中的标注的模块功能为：

- ④ 清除轴报错功能块
- ⑤ 轴停止运动功能块



图 10.21 脉冲轴绑定、轴使能及轴相对运动功能块



图 10.22 轴复位及轴停止功能块

- 5) 编写逻辑控制程序。

图 10.23 为逻辑控制程序，图中标注的程序解释如下：

- ① 调用 PRG 下的脉冲输出程序段及计数程序段进行调用；

LS_MotionControl_P 功能块的 `stAxis` 的端口指针变量地址赋值；

- ② 对所有功能块进行复位
- ③ 计数器计数前的清零(注意:计数器清零要在 LS_Counter_Enable 为 TURE 情况下,再触发 LS_PresetValue 才可清零;预置值与清零功能使用方式一致,只是预置值的不同)
- ④ 计数器打开
- ⑤ 触发脉冲输出功能块进行脉冲输出
- ⑥ 脉冲输出完成后复位



图 10.23 逻辑控制程序

10.3.6. LC1000/LC5000IO 实现高速计数器功能的方法

LC1000 系列产品本体的 PWM 输出发脉冲,并将其输入给本体的高速计数器,实现计数功能。

1) 在 LeadSys Studio 软件界面选择所使用的计数器,本例选择计数器 0,如图 10.24 所示。

- ① 鼠标左键双击“High_Speed_IO_Module”(LSHighSpeedIO)。
- ② 左键单击“通用设置”进入设置界面。
- ③ 左键单击“计数器 0”进入计数器启用界面。
- ④ 勾选“启用”计数器 0。

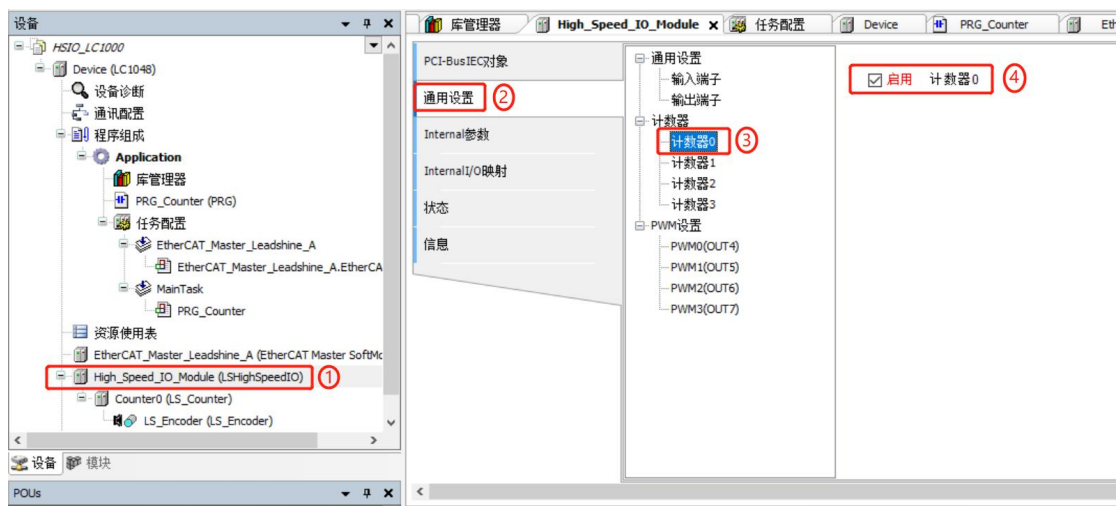


图 10.24 高速计数器启用

2) 配置计数器参数，包括计数器工作模式、计数方向、计数模式、计数上下限值及信号源。本例配置 Counter0 为单相计数工作模式、正向计数，线性模式、上下限使用默认值、IN0 为计数输入源。如图 10.25 所示

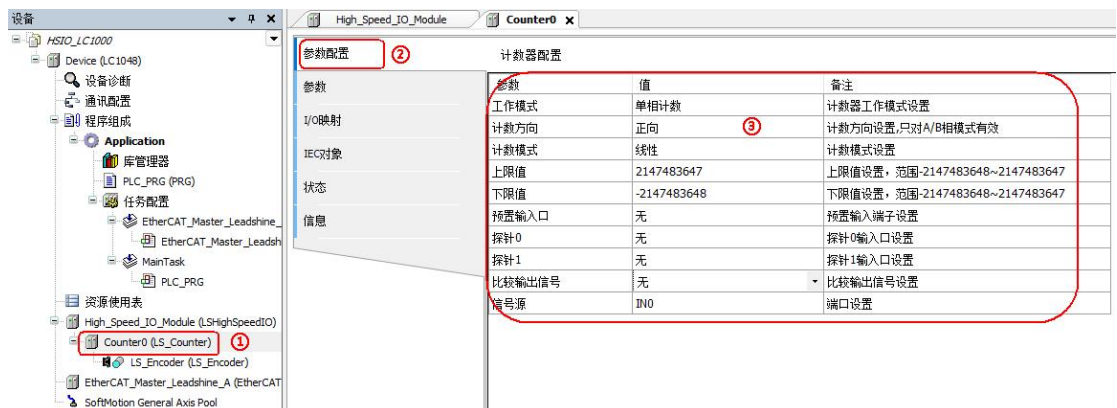


图 10.25 高速计数器参数配置

3) 配置计数器缩放比例，本例为 1: 1，一个脉冲对应一个用户单位。

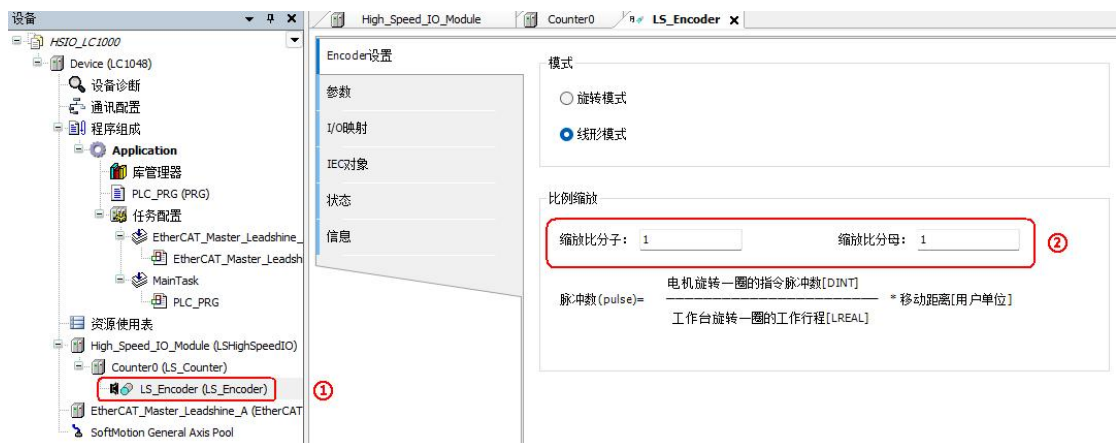


图 10.26 高速计数器缩放比例

4) 在 LeadSys Studio 软件界面选择所使用的 PWM 输出，本例选择 PWM0，如图 10.27 所示。

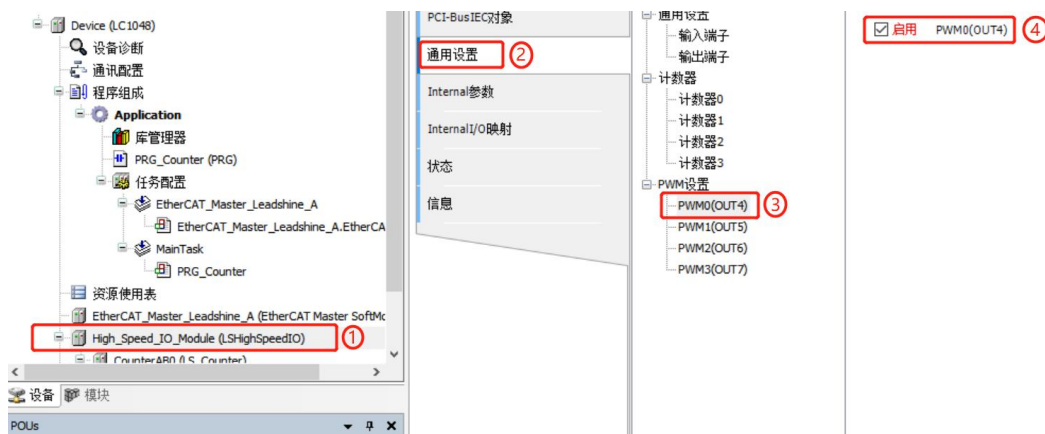


图 10.27 PWM 输出

4) 程序编写

①新建 POU (PRG_Counter)，程序语言选择梯形图，在 PRG_Counter 中创建如下所示变量。

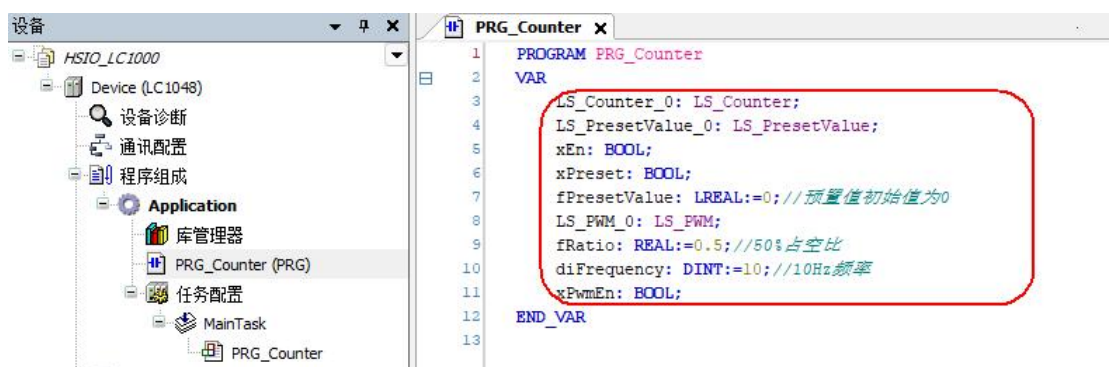


图 10.28 变量定义

③ 编写计数器预置及计数器计数功能块

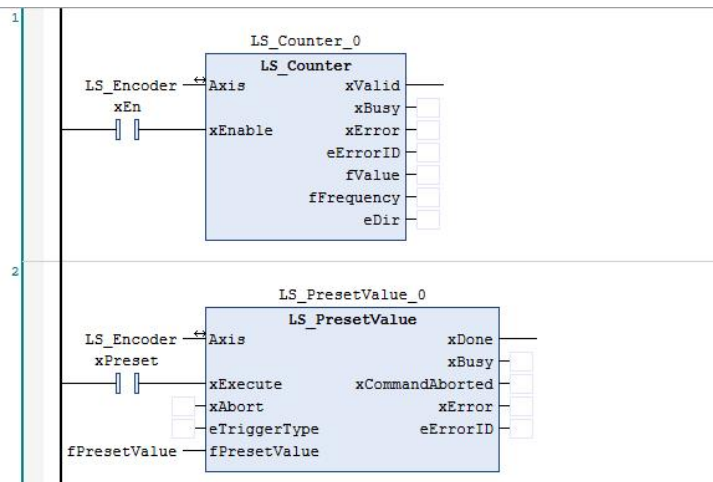


图 10.29 预置及计数功能块编写

③编写 PWM 输出功能块

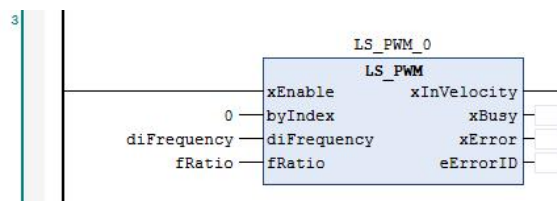


图 10.30 PWM 功能块编写

5) 将程序拖到 MainTask 任务下，下载运行程序，将 xEn 和 xPwmEn 置 TRUE，使能计数器和 PWM 输出，此时计数器开始计数，计数器每隔 0.1s 增加一个值，运行效果如下。

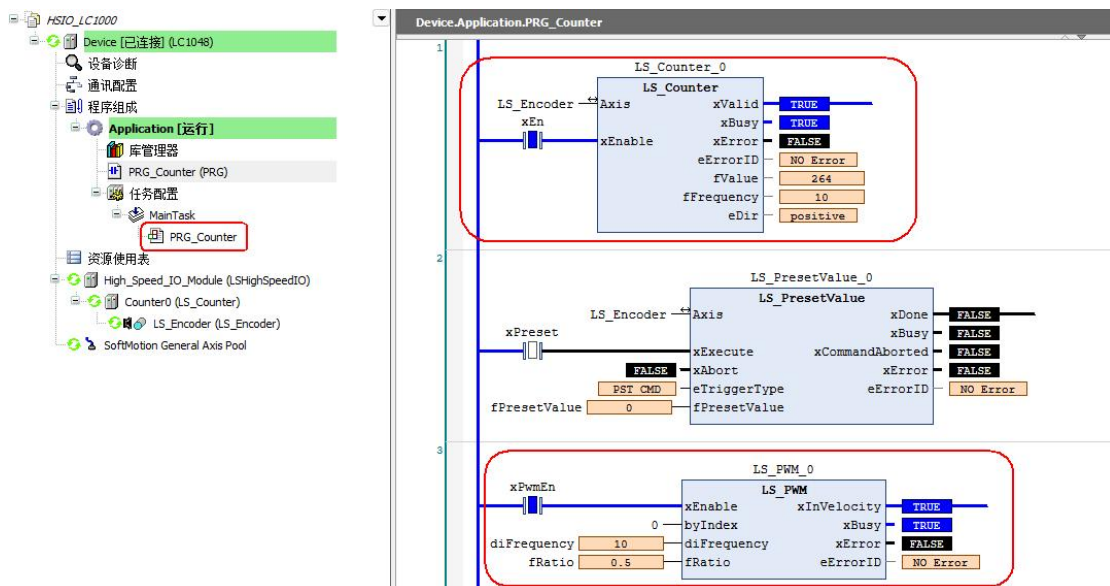


图 10.31 计数器运行效果

10.4. 编码器功能

10.4.1. PMC610 编码器指令及功能

表 10.9 编码器指令及功能

指令名	功能说明
LS_Encoder_EzClearSet	EZ 清零参数设置功能
LS_Encoder_EzClearRead	EZ 清零配置读取功能
LS_Encoder_EzSetFilterTime	EZ 滤波参数设置功能
LS_Encoder_EzGetFilterTime	EZ 滤波参数读取功能
LS_Encoder_SetVal	设置编码器数值功能
LS_Encoder_GetVal	读取编码器数值功能
LS_Encoder_SetWorkMode	设置编码器工作模式功能
LS_Encoder_SetAxisMove	编码器数值转换轴运动功能
LS_Encoder_EzClearSet_FB	EZ 清零参数设置功能块
LS_Encoder_EzClearRead_FB	EZ 清零配置读取功能块
LS_Encoder_EzSetFilterTime_FB	EZ 滤波参数设置功能块
LS_Encoder_EzGetFilterTime_FB	EZ 滤波参数读取功能块
LS_Encoder_SetVal_FB	设置编码器数值功能块
LS_Encoder_GetVal_FB	读取编码器数值功能块
LS_Encoder_SetWorkMode_FB	设置编码器工作模式功能块
LS_Encoder_SetAxisMove_FB	编码器数值转换轴运动功能块

10.4.2. LC5000IO 编码器功能编码器指令及功能

表 10.10 编码器指令及功能

指令名	功能说明
LS_Counter	计数器计数值读取

10.4.3. PMC610 编码器功能

PMC610 控制器具有 3 路差分编码器输入。用 LeadSys Studio 软件设置计数模式及输入计数方向，配合功能块可实现编码器计数。

下面通过一个例程详细介绍编码器计数功能使用方法。

使用一个手轮（手摇脉冲发生器）输出 AB 相脉冲，并接入 PMC610 本体的差分编码器端口，软件实现编码器计数功能。

1) 如图 10.32 所示，在 LeadSys Studio 软件界面选择所使用的编码器计数端口，设置计数模式、输入计数方向及高速输出模式。图中标注部分解释如下：

- ① 鼠标左键双击“High_Speed_IO_Module”(High Speed IO Module);
- ② 左键单击“硬件端口配置 2”进入配置界面;
- ③ 使用“编码器 0”设置作为编码器计数通道，设置计数模式为“AB 相 4 倍频”，输入计数方向为“正常”。



图 10.32 配置编码器参数

2) 新建变量，如图 10.33 所示，标注部分的解释如下。

- ① 新建的变量。
- ② 调用后续在 ACT()里编写的程序。



图 10.33 主程序界面

3) ACT()模块，如图 10.34 所示，图中标注部分解释如下。

- ① 编码器计数功能块。
- ② 转动手轮，编码器 0 正常计数。

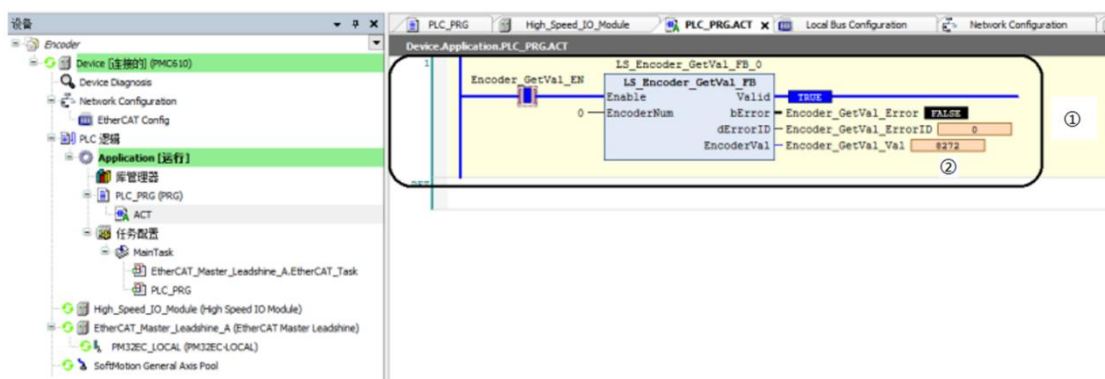


图 10.34 编码器模块的运行情况

10.4.4. LC5000IO 编码器功能

1) 在 LeadSys Studio 软件界面选择所使用的编码器，本例选择编码器 0，如图 10.35 所示。

- ① 鼠标左键双击“High_Speed_IO_Module”(LSHighSpeedIO)。
- ② 左键单击“通用设置”进入设置界面。
- ③ 左键单击“编码器 0”进入计数器启用界面。
- ④ 勾选“启用”编码器 0。

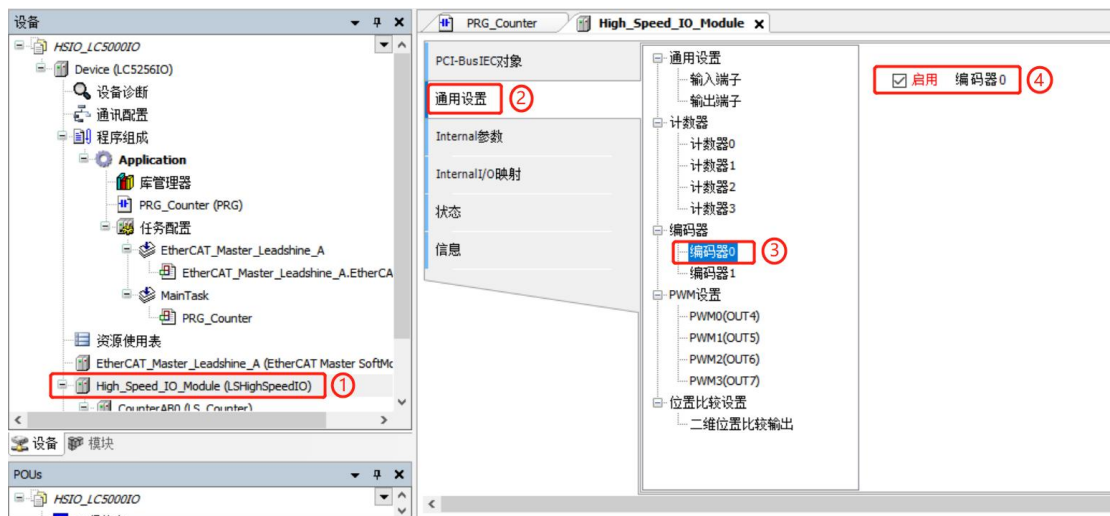


图 10.35 配置编码器参数

3) 配置编码器参数，包括编码器工作模式、计数方向、计数模式、计数上下限值。本例配置 CounterAB0 为 A/B 相 4 倍频工作模式、正向计数，线性模式、上下限使用默认值。如图 10.36 所示

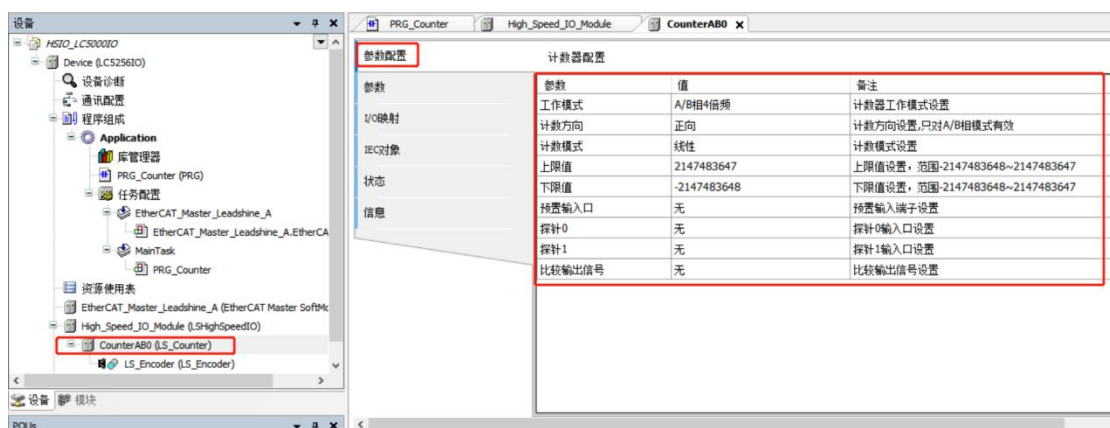


图 10.36 编码器参数

ACT()模块，如图 10.37 所示，图中标注部分解释如下。

- ③ 编码器计数功能块。
- ④ 转动差分编码器，编码器 0 正常计数。

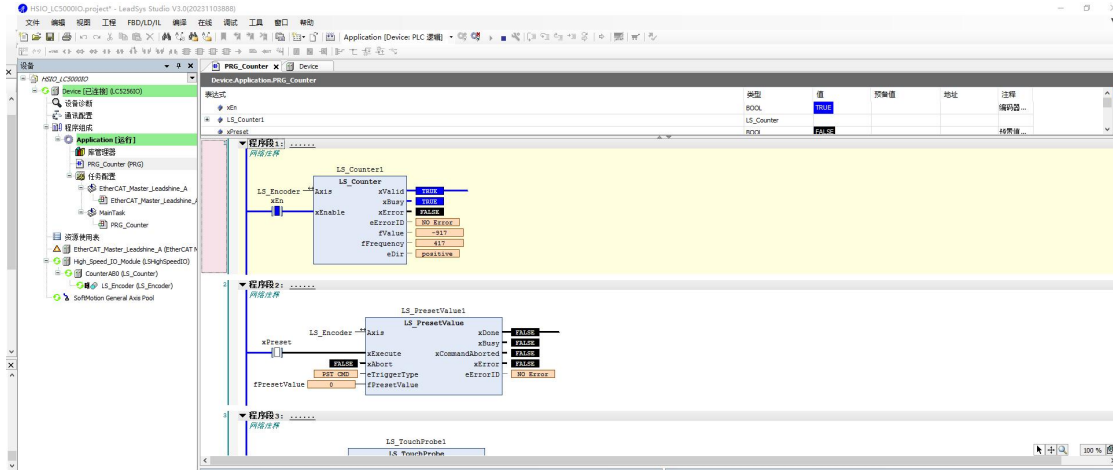


图 10.37 编码器模块的运行情况

10.5. 高速锁存功能

10.5.1. PMC600/PMC610 高速锁存指令介绍

表 10.11 PMC600/PMC610 高速锁存指令及功能

名称	功能
HighSpeedLatch_SetWorkMode	设置高速锁存参数功能
HighSpeedLatch_SetSource	设置高速锁存数据源功能
HighSpeedLatch_ReadState	读取高速锁存状态功能
HighSpeedLatch_ReadVal	读取当前锁存值功能
HighSpeedLatch_FIFOReadVal	读取当前锁存值，多周期指令功能
HighSpeedLatch_FIFOReadNumberbe	读取当前锁存的数据个数功能
HighSpeedLatch_SetWorkMode_FB	设置高速锁存参数功能块
HighSpeedLatch_SetSource_FB	设置高速锁存数据源功能块
HighSpeedLatch_ReadState_FB	读取高速锁存状态功能块
HighSpeedLatch_ReadVal_FB	读取当前锁存值功能块

HighSpeedLatch_FIFOReadVal_FB	读取当前锁存值，多周期指令功能块
HighSpeedLatch_FIFOReadNumberbe	读取当前锁存的数据个数功能块

10.5.2. LC1000/LC5000IO 高速锁存指令介绍

表 10.12 LC1000/LC5000IO 探针锁存指令及功能

名称	功能
LS_TouchProbe	本地计数器值探针锁存

10.5.3. PMC600/PMC610 高速锁存功能

PMC610 控制器具有 4 路高速输入锁存端口。用 LeadSys Studio 软件设置锁存端口的选择，配合功能块可实现锁存。

下面通过一个例程详细介绍高速锁存功能使用方法。

使用一个手轮输出 AB 相脉冲，再用锁存触发信号及软件模块锁存手轮的数值。

1) 如图 10.38 所示，选择对应锁存端口做锁存信号输入源。图中标注部分解释如下：

- ① 鼠标左键双击“High_Speed_IO_Module”(High Speed IO Module)
- ② 左键单击“硬件端口配置 2”进入配置界面。
- ③ 使用“高速锁存 0”设置作为锁存端口做锁存信号输入源。

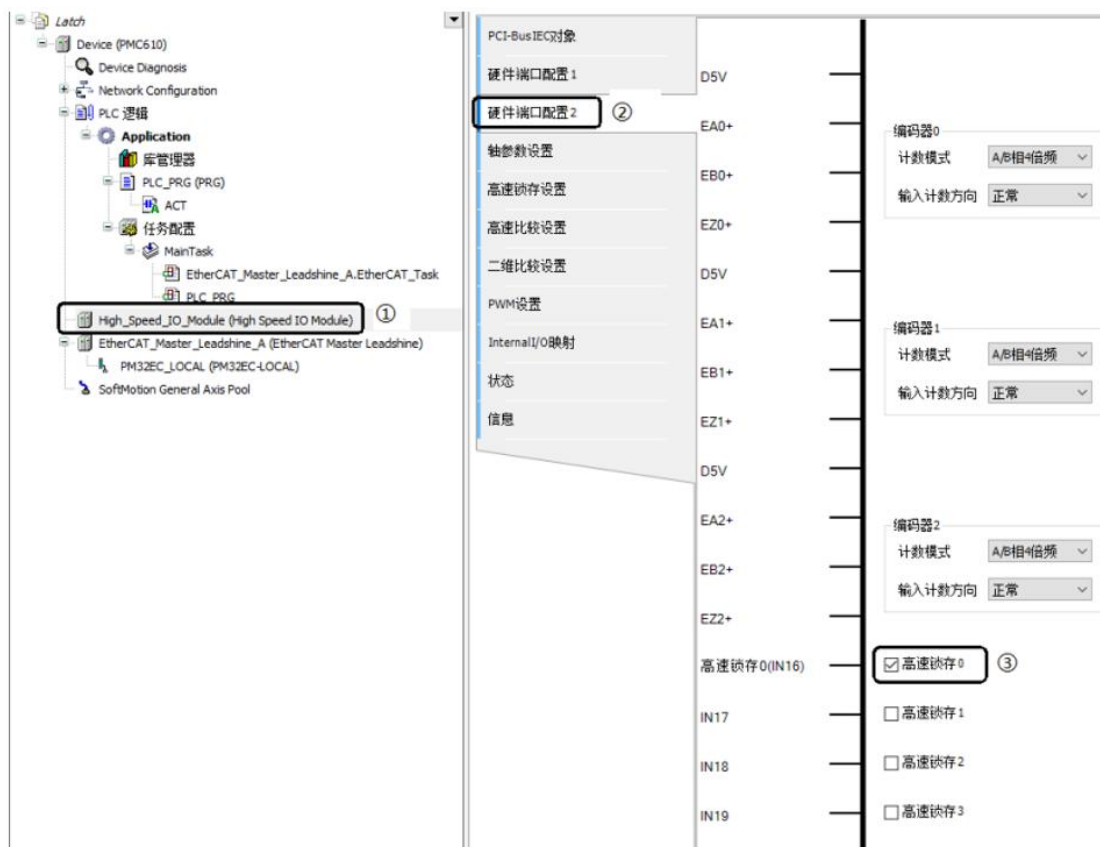


图 10.38 配置高速锁存信号

1) 新建变量，如图 10.39 所示，图中标注部分解释如下。

① 新建的变量。

② 参数赋值；设定此次为单次锁存模式、上升沿触发锁存编码器反馈位置；调用 ACT() 程序。

2) ACT() 程序如图 10.40 所示，图中标注部分解释如下。

① 编码器计数功能块。

② 设置高速锁存的通道、轴号及其数据源。这里配置成高速锁 0，轴号 0，使用编码器反馈值作为锁存值。

③ 设置锁存模式为单次锁存。

④ 使用手轮输出 AB 相差分脉冲，触发锁存信号锁存后，成功锁存编码器值。

表达式	类型	值	注释
LS_Encoder_GetVal_FB_0	LS_Encoder_GetVal_FB		
Encoder_GetVal_EN	BOOL	TRUE	
Encoder_GetVal_Error	BOOL	FALSE	
Encoder_GetVal_ErrorID	DINT	0	
Encoder_GetVal_Val	DINT	3192	
LS_HighSpeedLatch_SetWorkMode_FB_0	LS_HighSpeedLatch...		
Latch_SetWorkMode_EN	BOOL	TRUE	
Latch_Mode	UINT	0	
Latch_Follow	UINT	1	
Latch_Filter	UINT	0	
Latch_SetWorkMode_Error	BOOL	FALSE	
Latch_SetWorkMode_ErrorID	DINT	0	
LS_HighSpeedLatch_ReadVal_FB_0	LS_HighSpeedLatch...		
HighSpeedLatch_EN	BOOL	TRUE	
HighSpeedLatch_Error	BOOL	FALSE	
HighSpeedLatch_ErrorID	DINT	0	
HighSpeedLatch_Val	DINT	3192	
HighSpeedLatch_ValLR	LREAL	3192	
HighSpeedLatch_SetSource_EN	BOOL	TRUE	
SetSource	UINT	1	
HighSpeedLatch_SetSource_Error	BOOL	FALSE	
HighSpeedLatch_SetSource_ErrorID	DINT	0	
LS_HighSpeedLatch_SetSource_FB_0	LS_HighSpeedLatch...		

1	ACT();
2	Latch_Mode(0):=0; //设置锁存模式: 0-单次锁存; 1-连续锁存(系统原PIFO数量为256), 参数值需要保持
3	Latch_Follow(1):=1; //选择高速锁存触发沿: 0-下降沿锁存; 1-上升沿锁存; 2-任意沿锁存(上升沿或下降沿)
4	SetSource(1):=1; //设置锁存源: 0-指令位置; 1-编码器反馈位置 RETURN

图 10.39 高速锁存的主程序

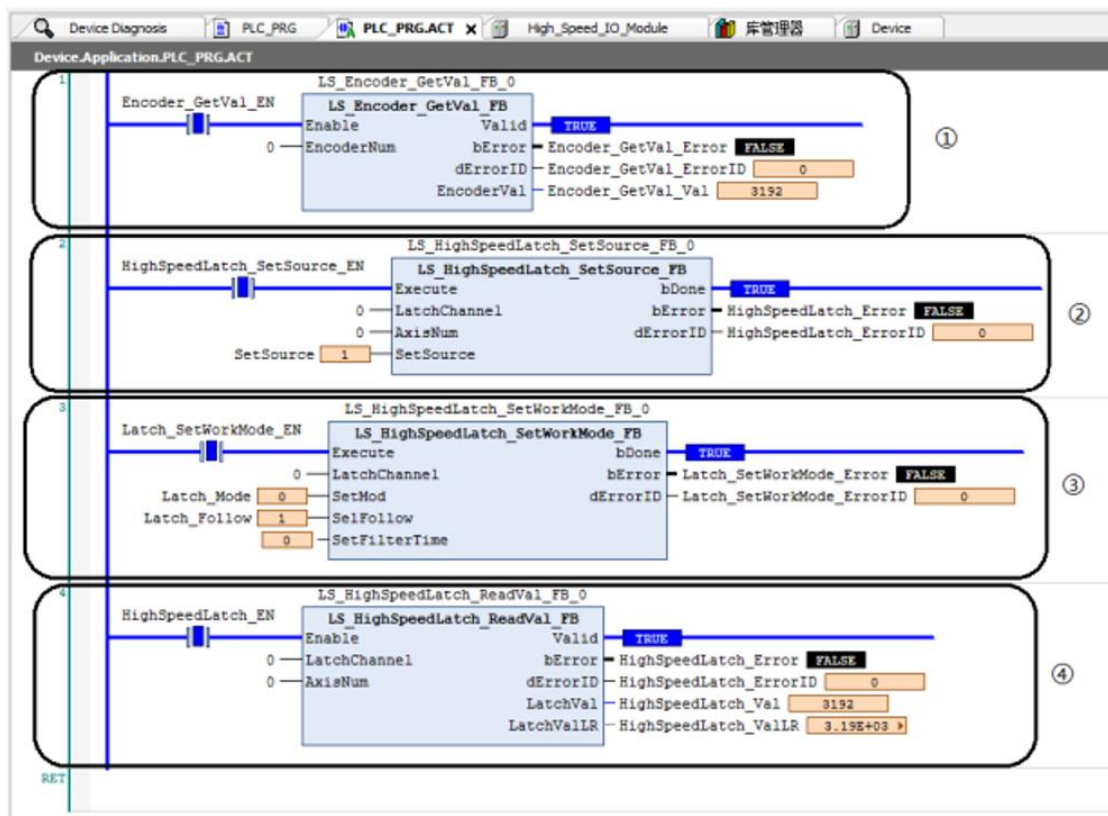


图 10.40 高速锁存模块运行情况

10.5.4. LC1000/LC5000IO 高速锁存功能

LC1000 控制器每路本地计数器都有 2 路探针锁存端口。用 LeadSys Studio 软件设置探针端口后，配合功能块可实现锁存功能，LC5000IO 与此类似。

下面通过一个例程详细介绍探针锁存功能使用方法。在 10.3.6 节例程 [C.使用 LC1000 本体的 PWM 输出发脉冲，并将其输入给本体的高速计数器，实现计数功能](#) 的基础上，增加计数器 0 的探针 0 设置（配置为 IN2 输入端口），实现当 IN2 检测到输入信号时，触发锁存本地计数器 0 的值。

1) 配置探针输入口



图 10.41 探针输入口配置

2) 程序编写

在 PRG_Counter 中增加如下所示变量。



图 10.42 变量定义

在 PRG_Counter 中增加探针功能块。

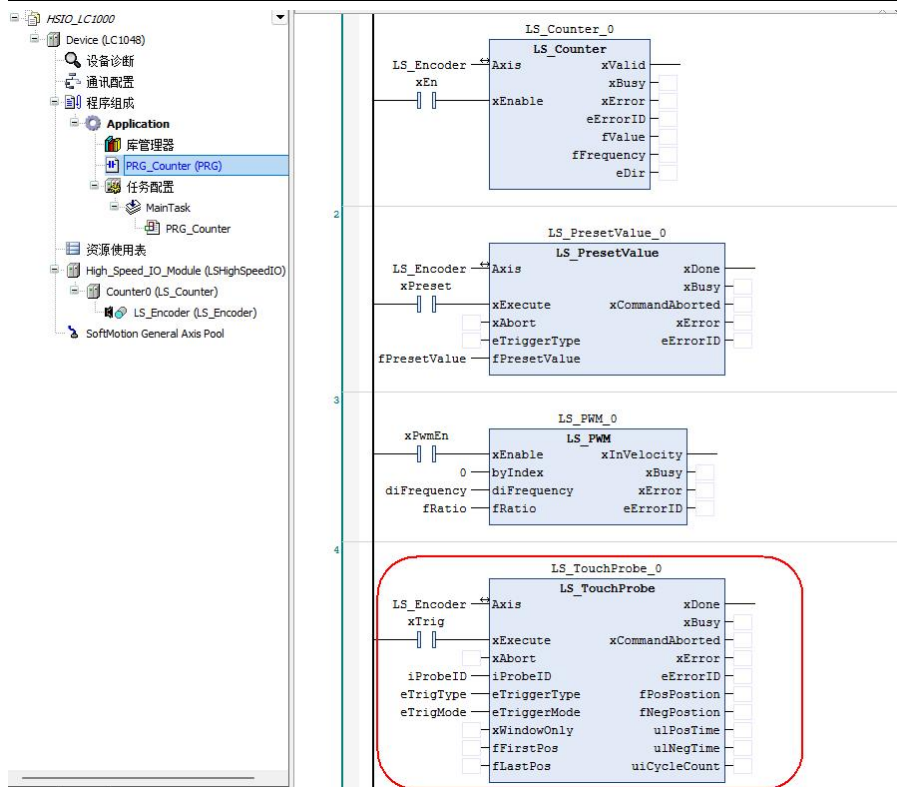


图 10.43 探针功能块编写

重新编译，下载运行程序，将 **xEn** 和 **xPwmEn** 置 TRUE，使能计数器和 PWM 输出，此时计数器开始计数。再将 **xTrig** 置 TRUE，使能探针功能块，此时 IN2 输入口检测到上升沿信号时，**fPosposition** 会记录当前的锁存值，同时 **ulPosTime** 会记录当前锁存的时间戳，运行效果如下。

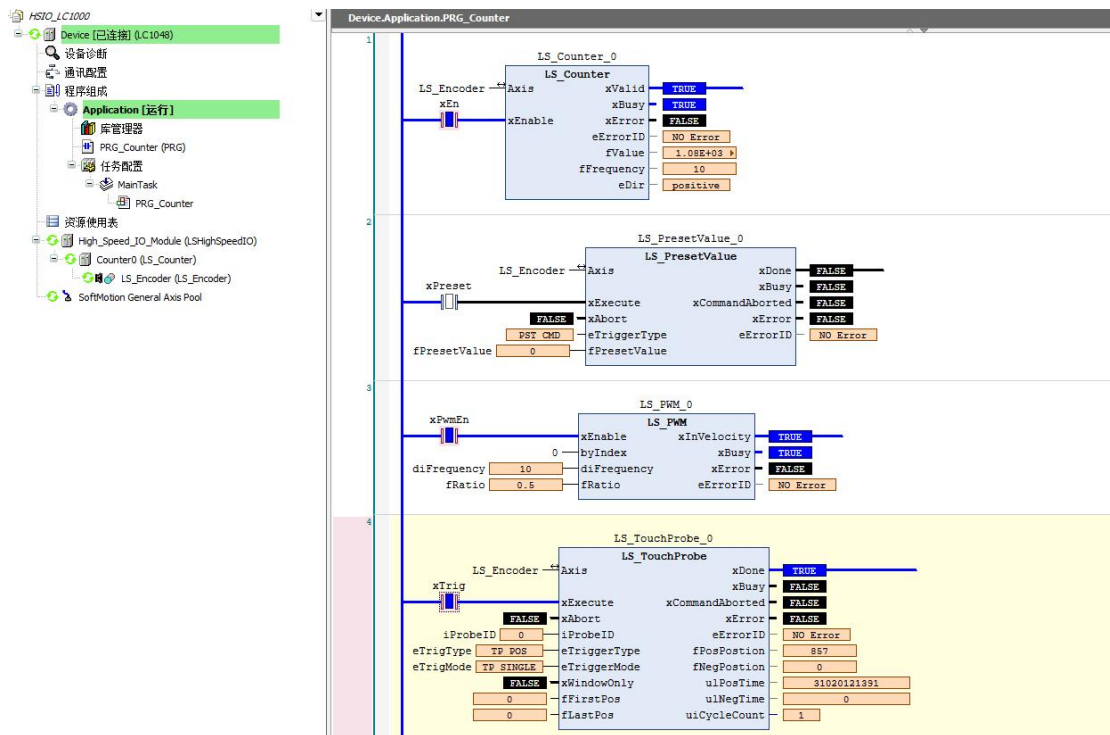


图 10.44 探针锁存程序运行效果

第十一章 扩展模块

11.1. 扩展模块类型

MC500 系列 PLC 本体可直接扩展 R2 系列模块,也可通过 EtherCAT 总线与 R2EC 耦合器连接来使用扩展模块,单个 MC500 系列 PLC 或单个 R2EC 耦合器最大可扩展 32 个 R2 系列模块。

PMC600 系列和 LC 系列 PLC 本体无法直接连接 R2 系列模块,但是 PLC 可通过 EtherCAT 总线与 R2EC 耦合器连接,将扩展模块挂在耦合器上。单个 R2EC 耦合器最多可扩展 32 个模块。R2 系列模块型号及功能描述详见表 11.1。

表 11.1 R2 系列模块型号及描述

模块类型	型号	描述
耦合器	R2EC	EtherCAT 总线适配器,可扩展 32 个总线节点
数字量输入模块	PM-1600	16 路数字量输入模块,源型/漏型输入
	PM-3200	32 路数字量输入模块,源型/漏型输入
	PM-3200-1	32 路数字量输入模块,源型/漏型输入,MIL 接口
	PM-3200-2	32 路数字量输入模块,源型/漏型输入,富士通接口
数字量输出模块	PM-0016-N	16 路数字量输出模块,漏型输出
	PM-0016-R	16 路数字量输出模块,继电器输出
	PM-0032-N	32 路数字量输出模块,漏型输出
	PM-0032-N-1	32 路数字量输出模块,漏型输出,MIL 接口
	PM-0032-N-2	32 路数字量输出模块,漏型输出,富士通接口
	PM-0016-P	16 路数字量输出模块,源型输出
数字量输入输出模块	PM-1616-N	32 路数字量输入输出模块,源型/漏型输入,漏型输出

模拟量输入模块	PM-A0400-IV	4 路模拟量输入模块，电流/电压输入
模拟量输出模块	PM-A0004-IV	4 路模拟量输出模块，电流/电压输出
脉冲模输出块	PM-P0004-D*	4 路脉冲输出模块，差分输出
编码器输入模块	PM-E0200-S	2 路编码器输入模块，单端输入
	PM-E0200-D	2 路编码器输入模块，差分输入
温度模块	PM-T0400-TC	4 路温度模块，热电偶类型
	PM-T0400-TR	4 路温度模块，热电阻类型

11.2. 扩展模块组态

MC500 系列、PMC600 系列及 LC 系列 PLC 按照以下两种方式进行模块组态：

方式一：手动添加

- 1) 打开“LeadSys Studio”编程软件，新建工程，选择编程语言。具体操作方法和第 2 章、第 6 章相同。
- 2) 添加所需扩展模块

a.通过 R2EC 耦合器使用模块（MC500 系列、PMC600 系列及 LC 系列 PLC 均可参考使用）

- ① 在设备树栏中，鼠标左键双击“通讯配置”进入配置界面；
- ② 在网络设备栏里，鼠标左键双击“R2EC”即可添加耦合器，如图 11.1 所示；
- ③ 选择所需的模块，可用鼠标左键双击所需模块或左键单击所需模块后拖动至配置页面表中，如图 11.2 所示。

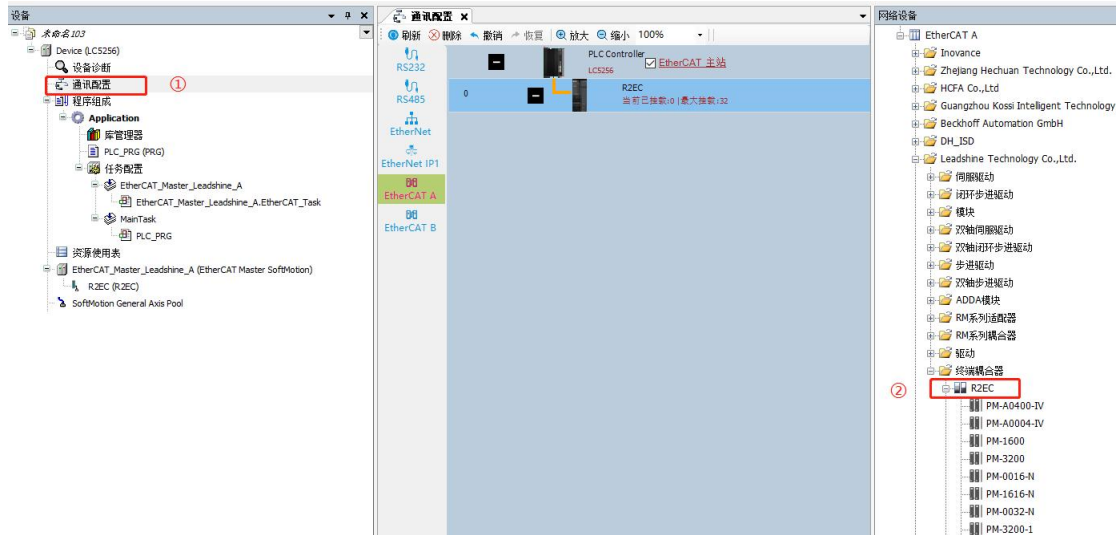


图 11.1 手动添加耦合器

图 11.2 手动添加扩展模块

b.在控制器本体添加模块（MC500 系列 PLC）

- ① 在设备树栏中，鼠标左键双击“LocalConfig”进入 PLC 本体模块配置界面；
- ② 在本地 IO 模块栏里，鼠标双击所需模块或单击后将模块拖动到配置页面即可完成模块增添，如图 11.3 所示；
- ③ 可在配置页面表可见已经选择增加的模块，如图 11.3 所示；

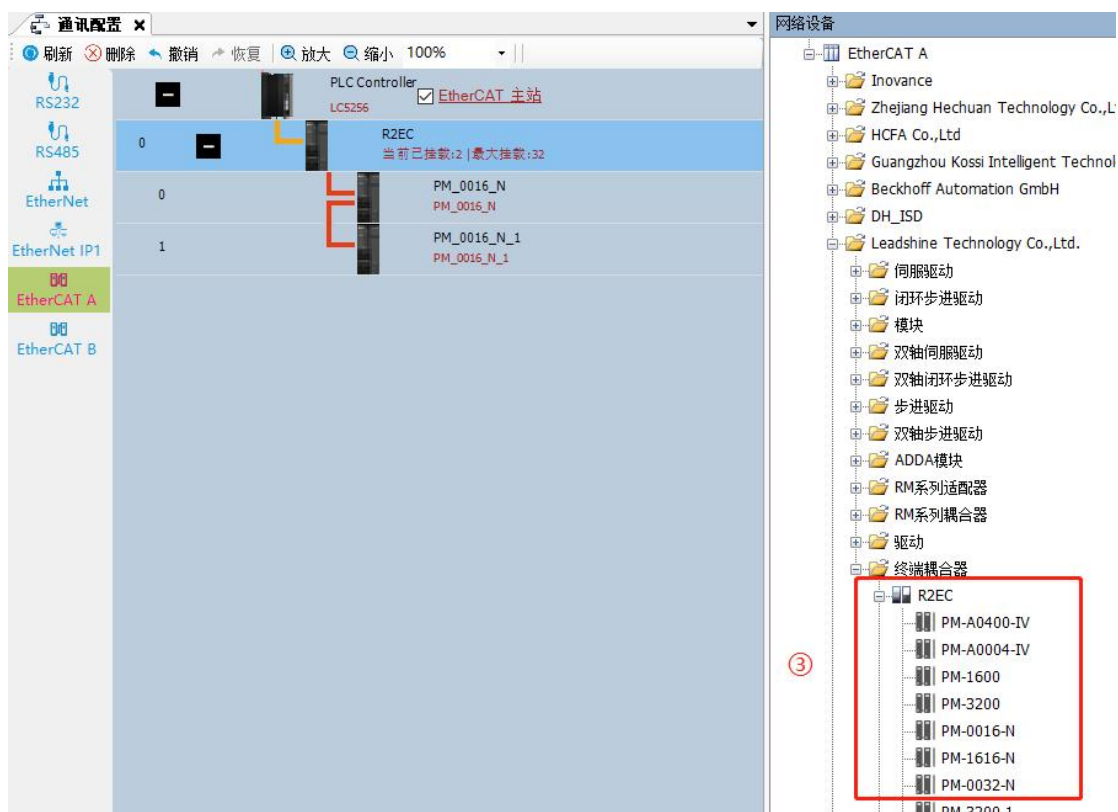


图 11.3 PLC 本体添加扩展模块

方式二：扫描添加

a.扫描 EtherCat 总线配置（MC500 系列、PMC600 系列及 LC 系列 PLC 均可参考使用）

- 1) 打开“LeadSys Studio”编程软件，新建工程，选择编程语言。
- 2) 连接 PLC。具体操作方法和第 2 章、第 6 章相同。
- 3) 设置主站及总线参数设置
 - ① 在设备树中，鼠标左键选择“EtherCAT_Master_Leadshine”→“通用”进入主站配置页面；
 - ② 勾选主动“配置主站/从站”；
 - ③ 网络设备名称使用默认的“eth0”；
 - ④ 选择默认的“按名称选择网络”；
 - ⑤ 总线周期根据负载而定，总线控制器支持 250us、500us、1ms、2ms、4ms、

8ms 总线周期，本次设置“2000”us；

- ⑥ 同步该值配置范围为 1~50，推荐设置成“20”，如图 11.4 所示；
- ⑦ 鼠标左键单击选择“EtherCAT I/O 映射”；
- ⑧ 在总线周期循环任务选择“MainTask”，若选择 MC500 系列 PLC 的本体扩展模块，则需要双击“Local_Master_Module”的“EtherCAT I/O 映射”界面内选择“MainTask”如图 11.5 所示。



图 11.4 设置 EtherCAT 总线参数

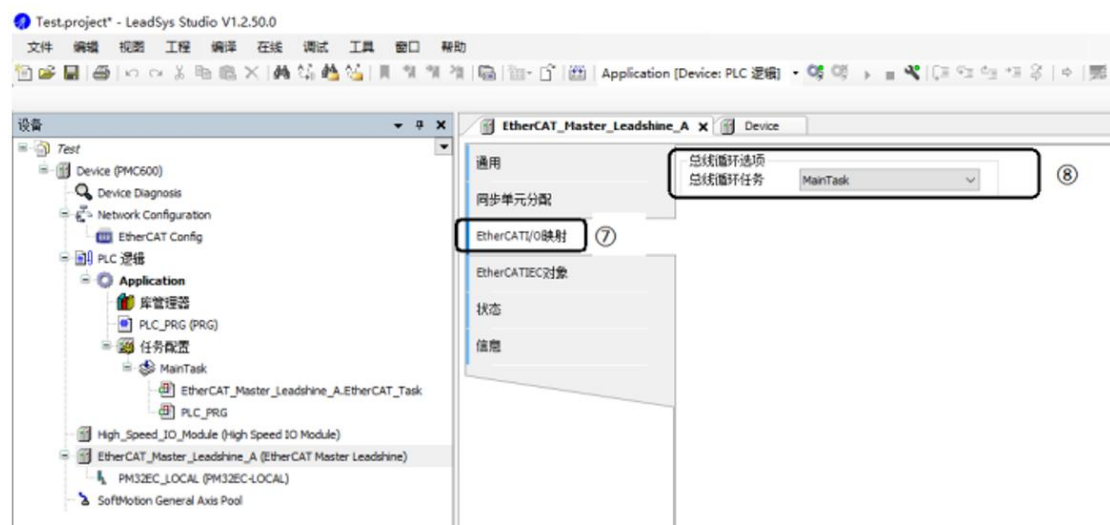


图 11.5 设置 EtherCAT 总线循环任务

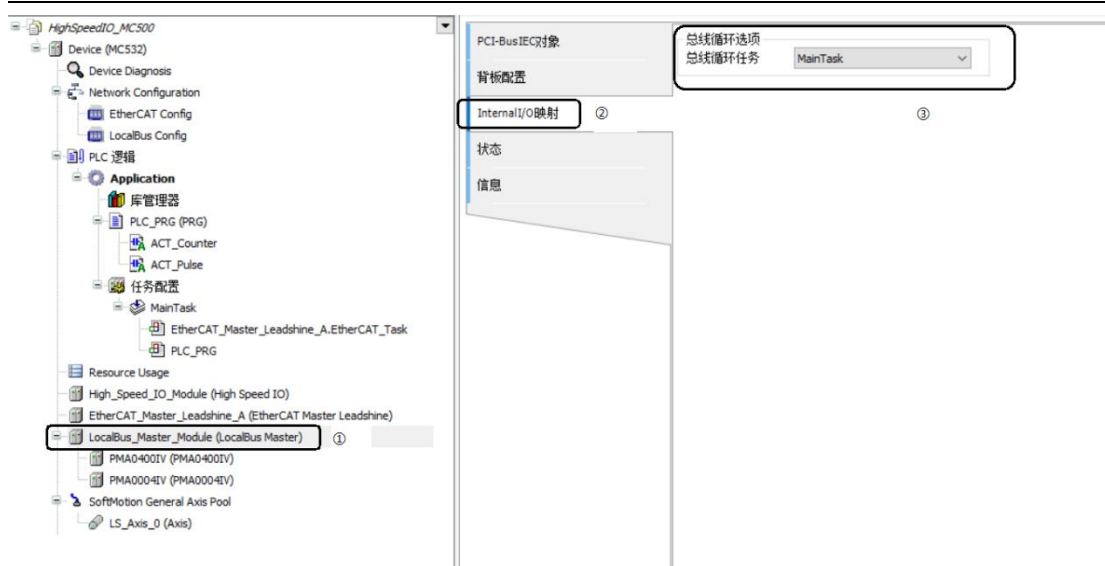


图 11.6 设置本地总线循环任务

4) 扫描扩展模块

- ① 在设备树中，鼠标右键选择“EtherCAT_Master_Leadshine”→“扫描设备”；
- ② 鼠标左键单击“复制所有设备到工程中”，即可完成从站的扫描。方法与扫描添加总线电机一样。扫描结果如图 11.7 所示。

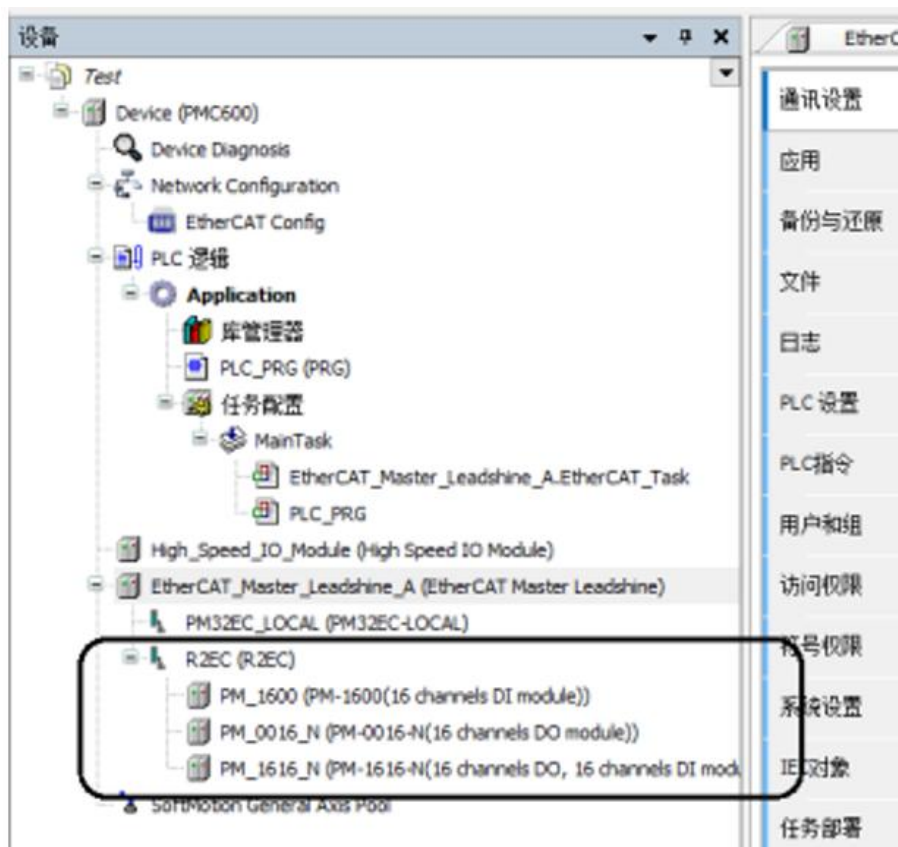


图 11.7 扫描后添加的扩展模块

b.扫描 PLC 本地总线配置（MC500 系列）

- 1) 打开“LeadSys Studio”编程软件，新建工程，选择编程语言。
- 2) 连接 PLC。具体操作方法和第 2 章、第 6 章相同。
- 3) 扫描本体扩展模块
 - ① 在设备树中，鼠标右键选择“Local_Master_Module”；
 - ② 左键单击“背板扫描”如图 11.8 所示。
 - ③ 鼠标左键单击“复制所有设备到工程中”，即可完成从站的扫描。方法与扫描添加总线电机一样。扫描结果如图 11.9 所示。

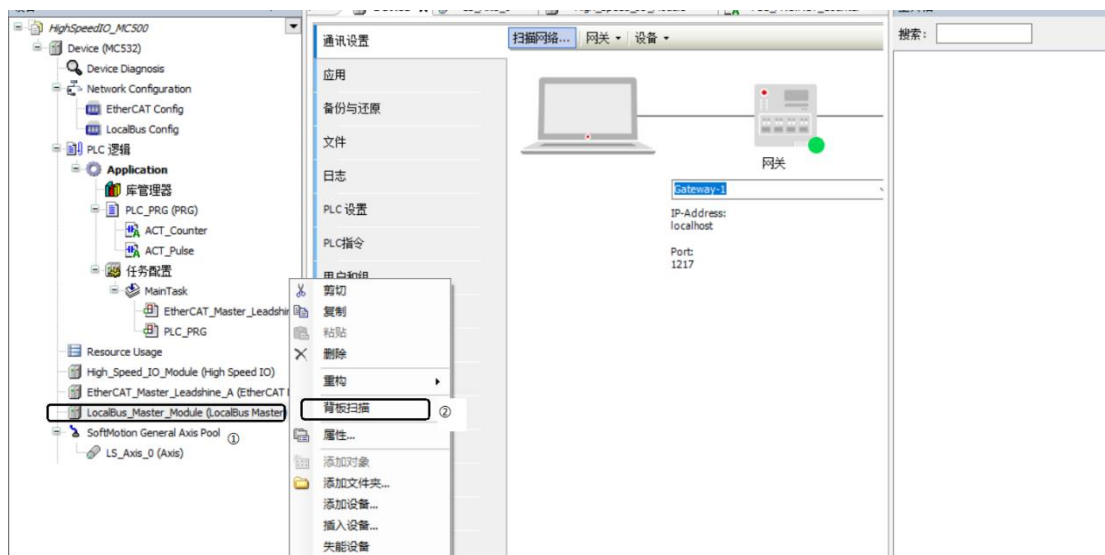


图 11.8 PLC 本体背板扫描

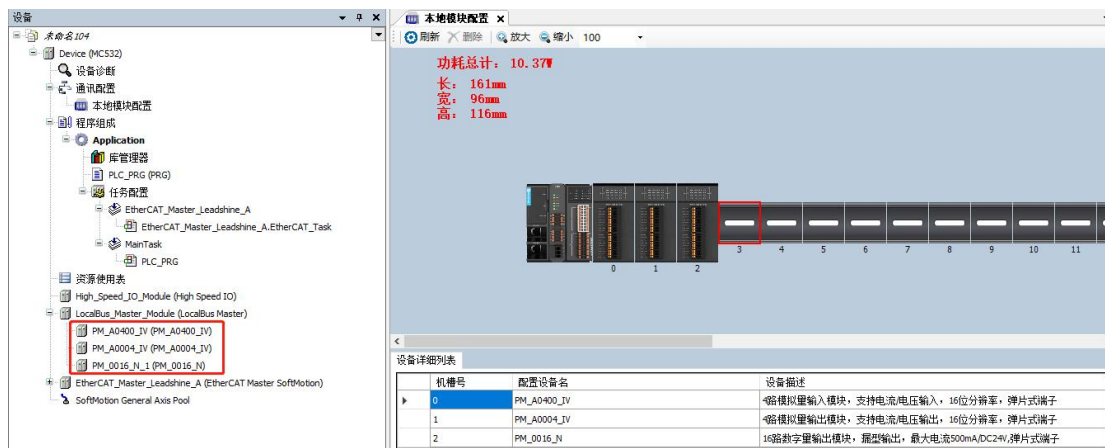


图 11.9 扫描后添加的扩展模块

11.3. 扩展模块的配置、映射

11.3.1. IO 模块的配置、映射

下面以 PM-0016-N 输出模块和 PM-1600 输入模块为例(PLC 本体挂载模块或 R2EC 耦合器挂载模块配置及映射方法一致), 介绍 IO 模块的配置、映射方法。
(详细使用方法可参考“R2 系列插片式扩展模块用户手册”)

- 1) 添加所需要的模块，可参考 11.2 节，PLC 本体添加或通过 R2EC 耦合器添加模块均可，添加完成后如图 11.10 或图 11.11 所示；

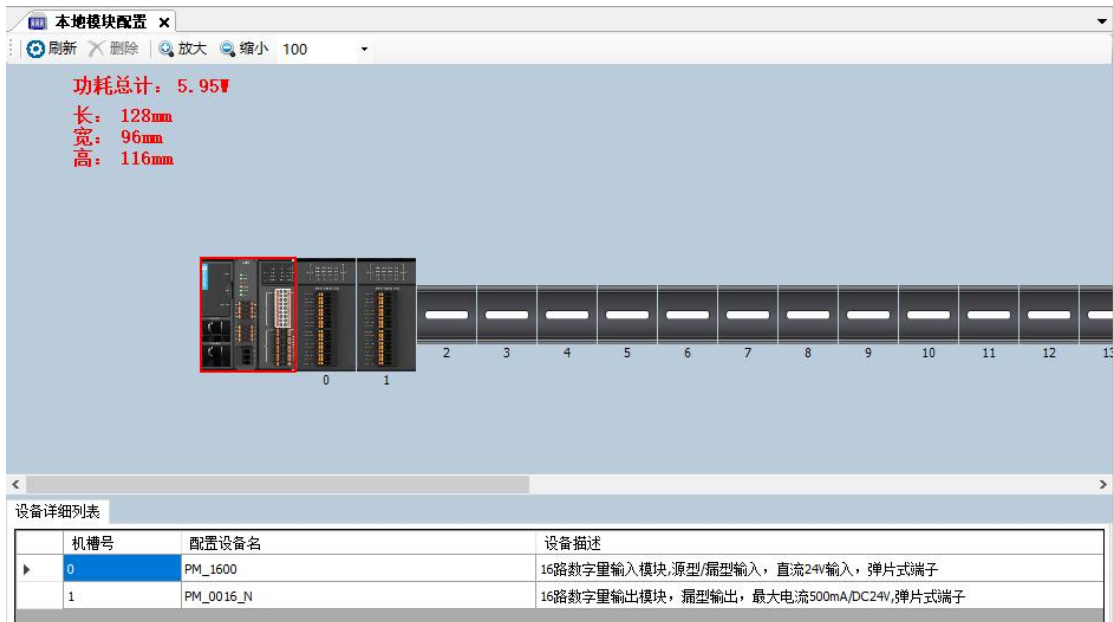


图 11.10 PLC 本体模块添加

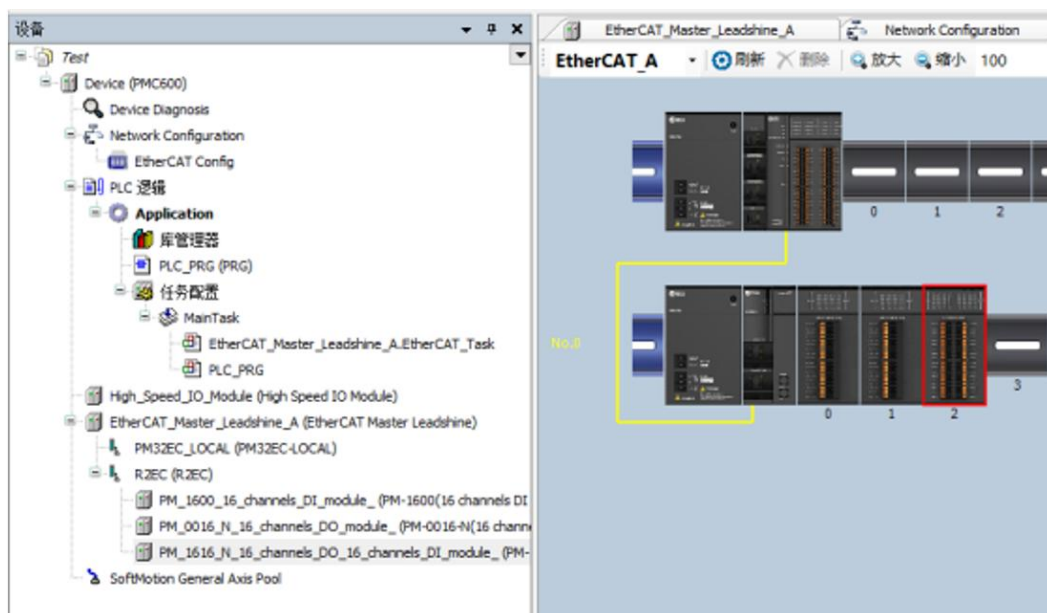


图 11.11 完成添加模块

- 2) 设置总线循环任务，可参考 11.2 节的图 11.6（若选择 MC500 系列 PLC 本体

总线添加模块，则需要设置本地总线循环任务）

3) 设置耦合器 R2EC 的变量更新形式

- ① 鼠标左键双击“R2EC(R2EC)”进入耦合器设置界面；
- ② 鼠标左键单击选择“EtherCAT I/O 映射”进入变量设置界面；
- ③ 在右下角“一直更新变量”选择“使能 2（一直在总线循环任务中）”，如图 11.12 所示。

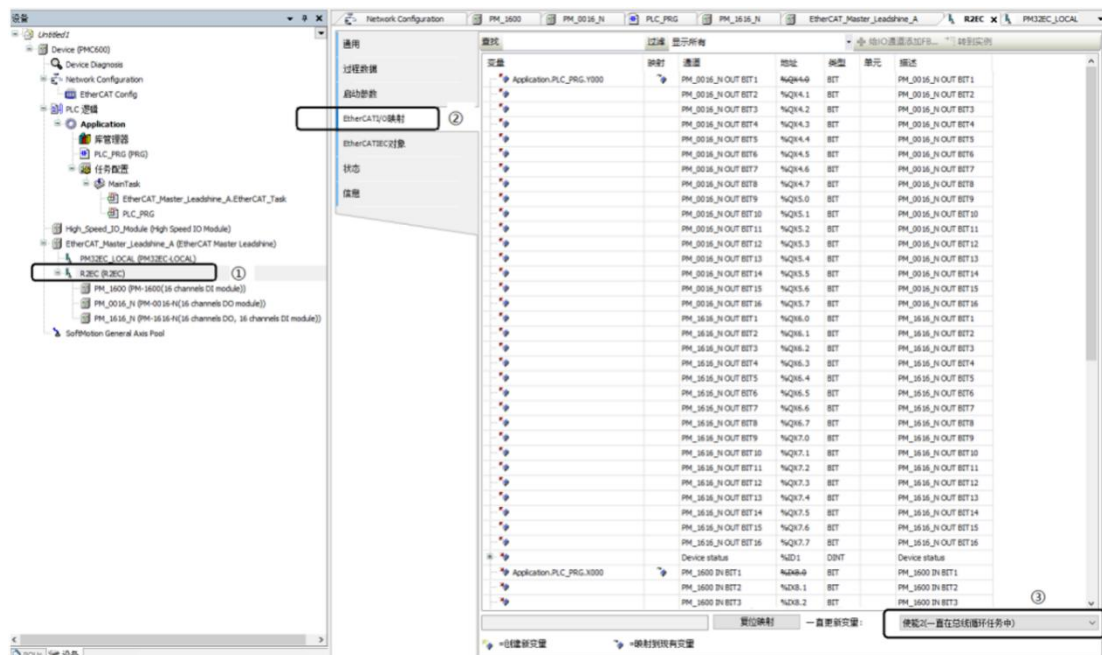


图 11.12 设置 R2EC 的变量更新形式

4) 声明及建立输入、输出变量，与地址关联映射后，即可正常使用。

- ① 在设备树中，鼠标左键选择“PLC_PRG(PRG)”进入编程页面新建变量，如图 8.11 所示；
- ② 鼠标左键双击“R2EC”；
- ③ 鼠标左键单击“EtherCAT I/O 映射”；
- ④ PM-0016-N 输出模块端口关联映射；
- ⑤ PM-1600 输入模块端口关联映射，如图 11.13 所示。

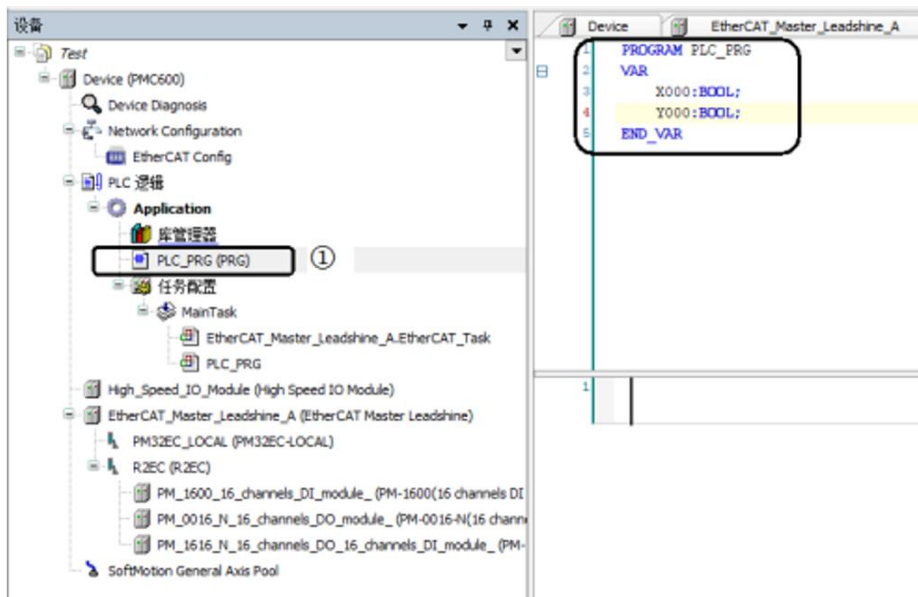


图 11.13 建立输入、输出变量 X000、Y000

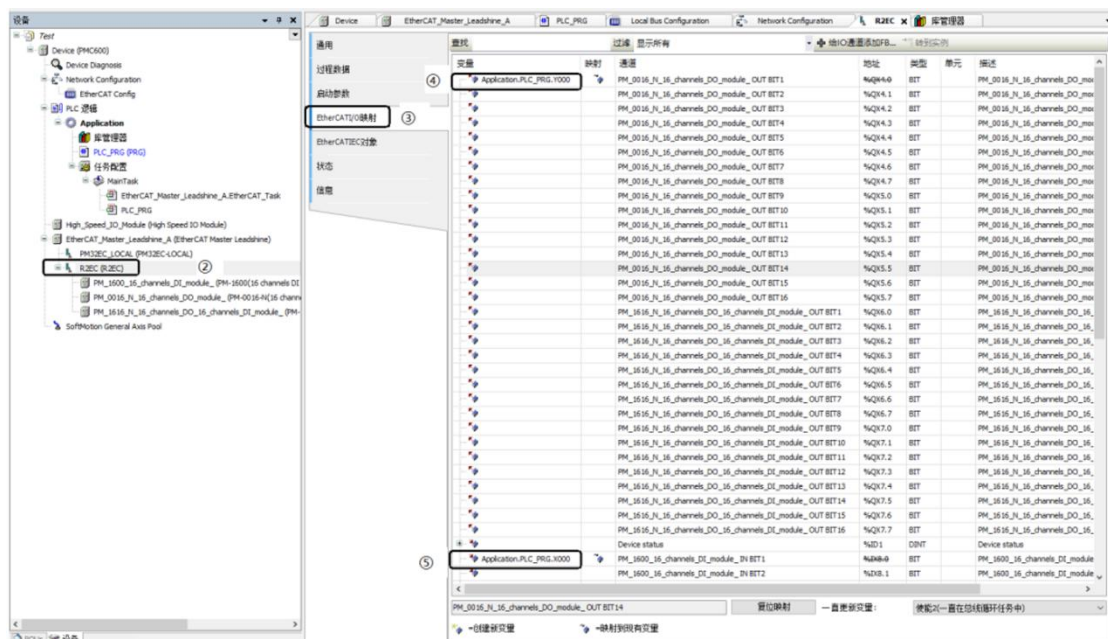


图 11.14 输入输出变量与输入输出接口关联映射

11.3.2. 模拟量模块的配置、映射

下面以 PM-A0400-IV 和 PM-A0004-IV 模块为例(PLC 本体挂载模块或 R2EC 耦合器挂载模块配置及映射方法一致), 介绍 AD、DA 模块的配置、映射方法。(详

细使用方法可参考“R2 系列插片式扩展模块用户手册”)

- 1) 添加所需要的模块，PLC 本体添加或通过 R2EC 耦合器添加模块均可，可参考 11.2 节，添加完成后如图 11.15 或图 11.16 所示；

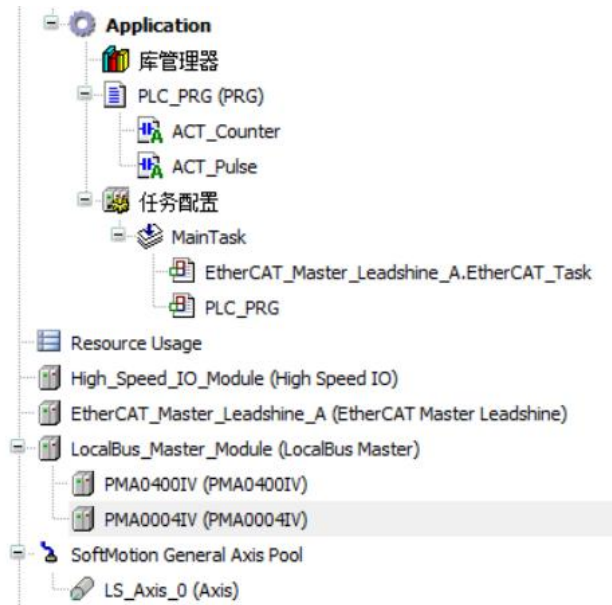


图 11.15 PLC 本体添加的 AD 和 DA 模块

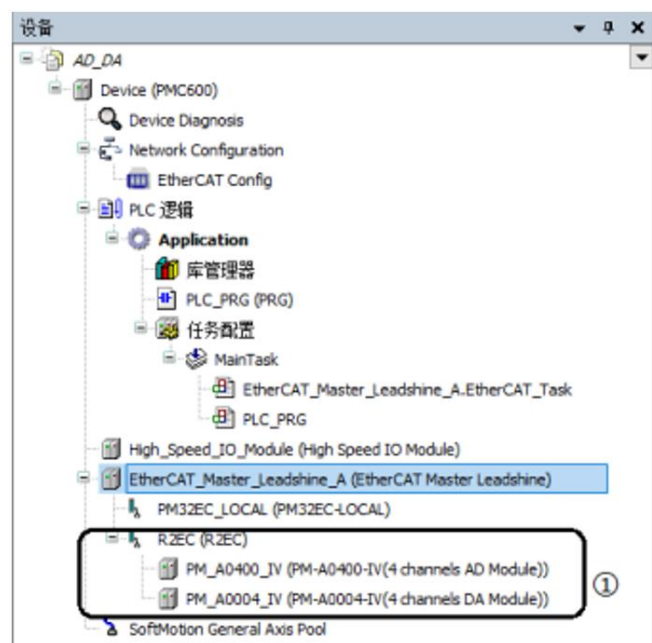


图 11.16 添加的 AD 和 DA 模块

- 2) 设置总线循环任务，可参考 11.2 节的图 11.5 或图 11.6（图 8.5 为 EtherCat

总线循环任务，图 11.6 为 MC500 系列 PLC 本地总线循环任务)

3) 确定从站 R2EC 耦合器的节点地址 (若选择 MC500 系列 PLC 本体总线添加模块, 则不需要设置节点地址)

- ① 鼠标左键双击“R2EC”;
- ② 鼠标左键单击“通用”进入从站设置界面;
- ③ “EtherCAT 地址”为默认 1002 (可启用专家设置的配置的站别名使用自己设置的从站号), 后续的 SDO_CO_SdoWrite 及 ETC_CO_SdoRead 指令需要使用。如图 11.17 所示。

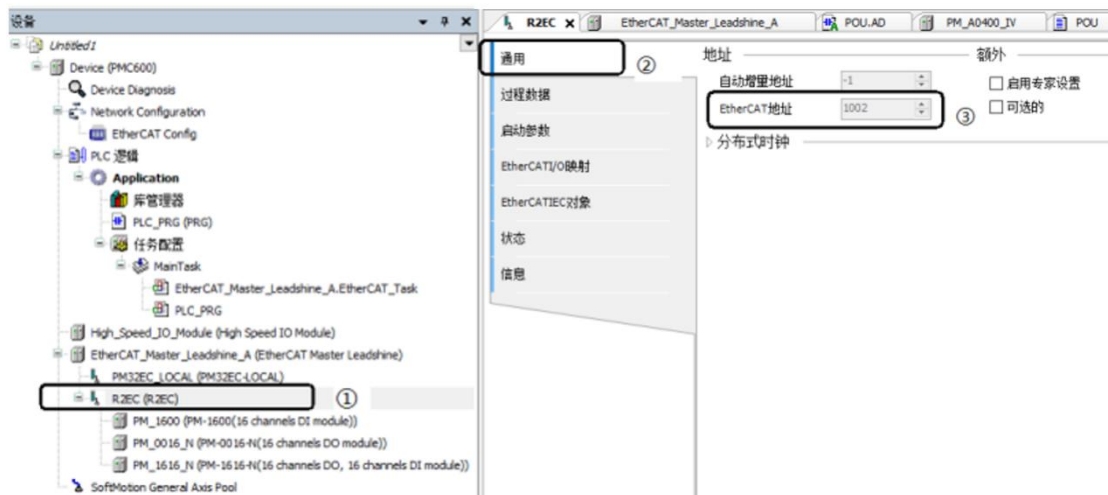


图 11.17 添加 R2EC 的节点地址

4) 设置模块的参数 (任选以下 a 或 b 两种方法, 若 2 种办法均使用, 则以通过 SDO 指令设置为准)

a. 通过界面设置参数

- ① 设置 PM-A0400-IV 模拟量输入模块的参数, 左键双击设备树下“PM-A0400-IV”进入设置页面 (EtherCAT 总线下 R2EC 的 PM-A0400-IV 或 LocalBus 下的 PM-A0400-IV 均可)。
- ② 左键单击“参数配置”, 进入模块参数配置界面, 如图 11.18 所示。
- ③ 选择所需要的量程, 本例程选择“-10V~10V”量程, 如图 11.18 所示。。
- ④ 通道滤波参数设置, 根据实际需要选择滤波参数, 如图 11.18 所示

- ⑤ 设置 PM-A0004-IV 模拟量输出模块的参数，左键双击设备树下“PM-A0004-IV”进入设置页面（EtherCAT 总线下 R2EC 的 PM-A0400-IV 或 LocalBus 下的 PM-A0004-IV 均可）。
- ⑥ 左键单击“参数配置”，进入模块参数配置界面，如图 11.19 所示。
- ⑦ 设置通道使能，勾选“通道 1”则为通道 1 进行使能（必须勾选，否则无法使用），如图 11.19 所示。
- ⑧ 选择所需要的量程，本例程选择“-10V~10V”量程，如图 11.19 所示。
- ⑨ 根据实际需要设置断线后输出状态，如图 11.19 所示。



图 11.18 PM-A0400-IV 模拟量输入模块参数配置

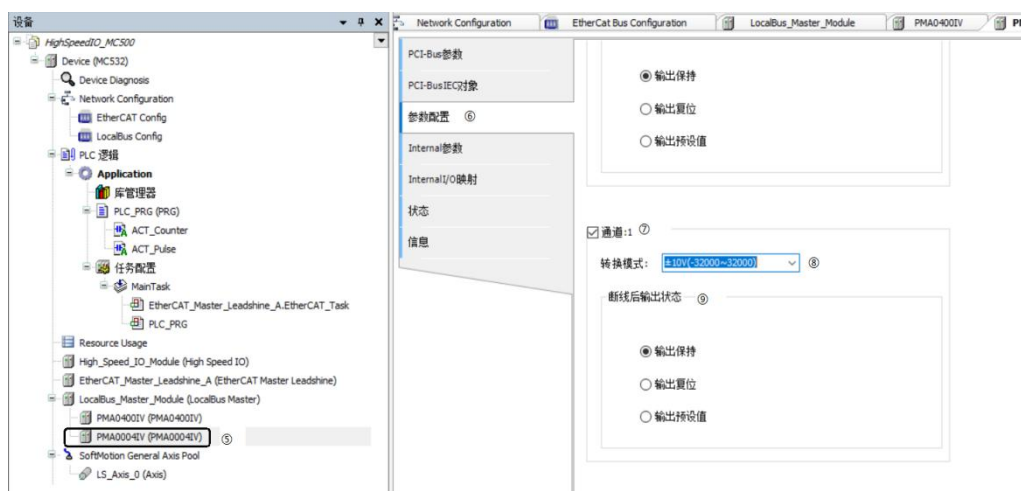


图 11.19 PM-A0004-IV 模拟量输出模块参数配置

b.通过 SDO 设置模块参数

- ① 声明及建立如图 11.20 所示的变量。(端口映射地址如下图所示)(可打开“样例程序”参考)

	类别	名称	地址	数据类型	初值	注释	特性
1	VAR	AD_Start		BOOL		模拟量输入模块使用使能	
2	VAR	DA_Start		BOOL		模拟量输出模块使用使能	
3	VAR	AD_Error		BOOL		模拟量输入模块错误	
4	VAR	DA_Error		BOOL		模拟量输出模块错误	
5	VAR	DA_Enable		BOOL		模拟量输出模块的输出使能标志	
6	VAR	AD_D0		USINT		模拟量输入模块的里程选择	
7	VAR	AD_D1		USINT		模拟量输入模块的输入通道滤波时间	
8	VAR	AD_D2		INT		模拟量输入模块的通道1读取值	
9	VAR	DA_D0		USINT		模拟量输出模块的通道使能选择	
10	VAR	DA_D1		INT		模拟量输出模块的输出使能回读	
11	VAR	DA_D2		USINT		模拟量输出模块的输出里程	
12	VAR	DA_D3		USINT		模拟量输出模块的断线后输出状态设置	
13	VAR	DA_D4		INT		模拟量输出模块的断线后预制值	
14	VAR	DA_D5		INT		模拟量输出模块输出通道1值设置	
15	VAR	ETC_CO_SdoWrite_0		ETC_CO_SdoWrite		模拟量输出模块输出使能	
16	VAR	ETC_CO_SdoRead_0		ETC_CO_SdoRead		回读模拟量输出模块是否使能成功 SDO	
17	VAR	ETC_CO_SdoWrite_1		ETC_CO_SdoWrite		模拟量输出模块输出通道里程选择 SDO	
18	VAR	ETC_CO_SdoWrite_2		ETC_CO_SdoWrite		模拟量输入模块输入通道里程选择 SDO	
19	VAR	ETC_CO_SdoWrite_3		ETC_CO_SdoWrite		模拟量输入模块输入通道滤波 SDO	
20	VAR	ETC_CO_SdoWrite_4		ETC_CO_SdoWrite		模拟量输出模块断线输出状态设置 SDO	
21	VAR	ETC_CO_SdoWrite_5		ETC_CO_SdoWrite		模拟量输出模块断线输出值 SDO	

图 11.20 添加 R2EC 的节点地址

- ② 确定模拟量输入模块相关参数索引及子索引号

模块的索引配置顺序为 $R2EC \times 1 + PM-A0400-IV \times 1 + PM-A0004-IV \times 1$

模块对象字典偏移公式为：对象字典首索引地址 + 模块节点号 $\times 10$

注意：从耦合器后第一个模块开始排序，从 0 开始，0、1、2、...0x1F（最多 32 个扩展模块），以此类推。

因此，模拟量输入模块的节点号为 0，模拟量输出模块的节点号为 1。

查询模块对象字典可知，模拟量输入模块主要有 2 个参数：输入量程及滤波。根据对象字典偏移公式，量程及滤波索引号（16 进制）的计算如下：

输入通道量程：

$$\text{索引号} = 0x8000 + 10 \times \text{节点号}(0 \sim 1F) = 0x8000 + 10 \times 0 = 0x8000$$

输入通道滤波：

索引号 = $0x8001 + 10 \times \text{节点号}(0 \sim 1F) = 0x8001 + 10 \times 0 = 0x8001$

子索引号根据模拟量输入模块的通道 1~通道 4 分别为 01H~04H。

③ 确定模拟量输出模块相关参数索引及子索引号

查询模块对象字典可知,模拟量输出模块主要有 2 个参数:输出量程及输出使能。

输出通道量程:

索引号 = $0x8000 + 10 \times \text{节点号}(0 \sim 1F) = 0x8000 + 10 \times 1 = 0x8010$

输出通道使能:

索引号 = $0x8001 + 10 \times \text{节点号}(0 \sim 1F) = 0x8001 + 10 \times 1 = 0x8011$

子索引号根据模拟量输出模块的通道 1~通道 4 分别为 01H~04H。

④ 通过 SDO_CO_SdoWrite 功能块设置模拟量输入模块的参数:

用 SDO_CO_SdoWrite_2 功能块,设置模拟量模块输入通道量程;

用 SDO_CO_SdoWrite_3 功能块,设置模拟量模块通道滤波,如图 11.21 所示。

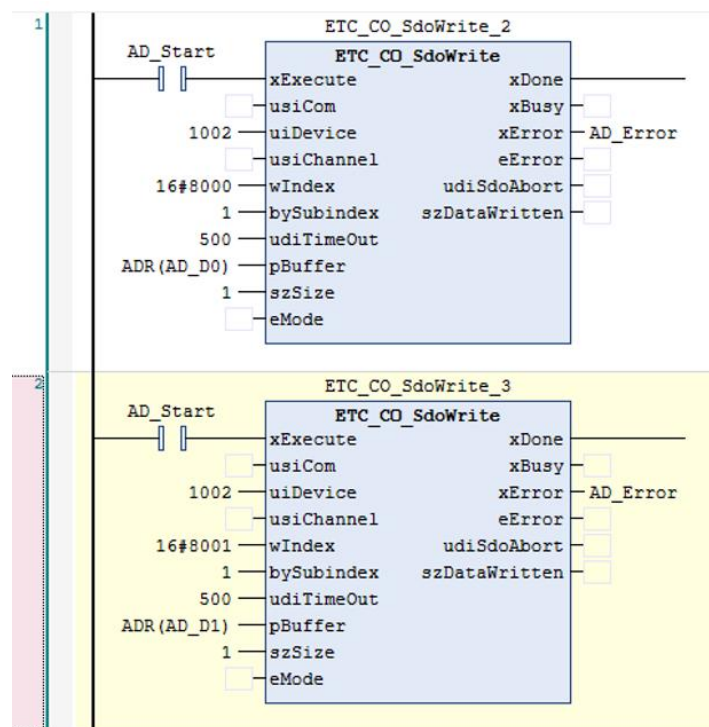


图 11.21 用 SDO_CO_SdoWrite 功能块设置 AD 模块的参数

⑤ 通过 SDO_CO_SdoWrite 功能块设置模拟量输出模块的参数：

SDO_CO_SdoWrite_0 功能块设置模拟量输出模块的输出使能，

SDO_CO_SdoWrite_1 功能块设置模拟量输出模块的量程，

SDO_CO_SdoWrite_4 功能块设置模块断线后模拟量输出模块端口状态，

SDO_CO_SdoWrite_5 功能块设置模拟量输出模块断线后输出预置值。

ETC_CO_SdoRead_0 功能块为模拟量输出使能设置读取判断。如图 11.22、11.23 所示。

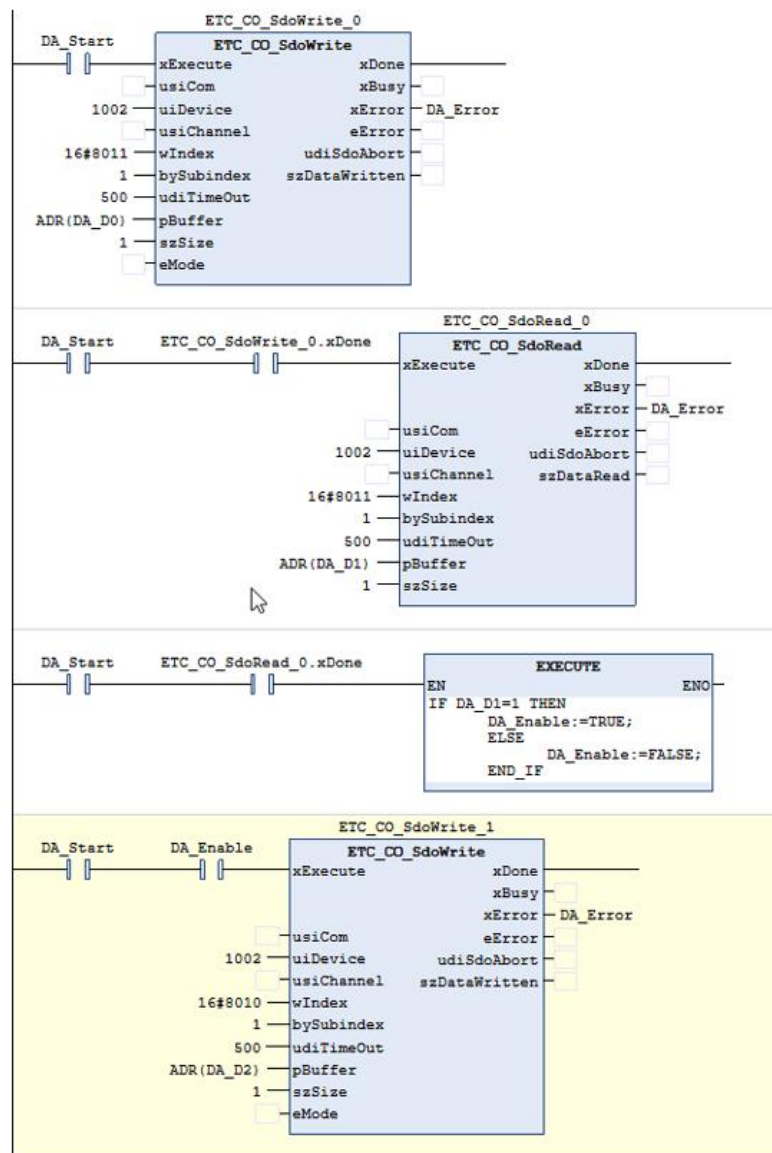


图 11.22 用 SDO_CO_SdoWrite、ETC_CO_SdoRead_0 功能块设置 DA 模块的参数

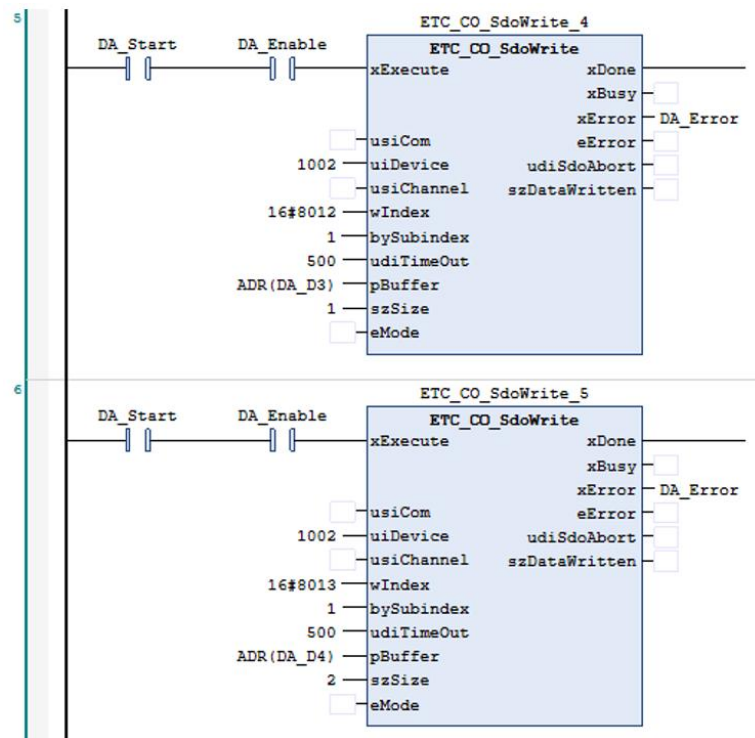


图 11.23 用 SDO_CO_SdoWrite 功能块设置 DA 模块的参数

5) 变量映射过程如图 11.24 及图 11.25 或图 11.26 所示。

- ① 鼠标左键双击“R2EC”（使用 R2EC 耦合器扩展模块）或左键双击“PM-A0400-IV”及“PM-A0004-IV”（MC500 系列 PLC 本体扩展模块）；
- ② 鼠标左键单击“EtherCAT I/O 映射”进入变量映射界面；
- ③ 绑定变量 DA_D5 为模拟量输出模块通道 1 的输出变量；
- ④ 绑定变量 AD_D2 为模拟量输入模块通道 1 的输入变量。

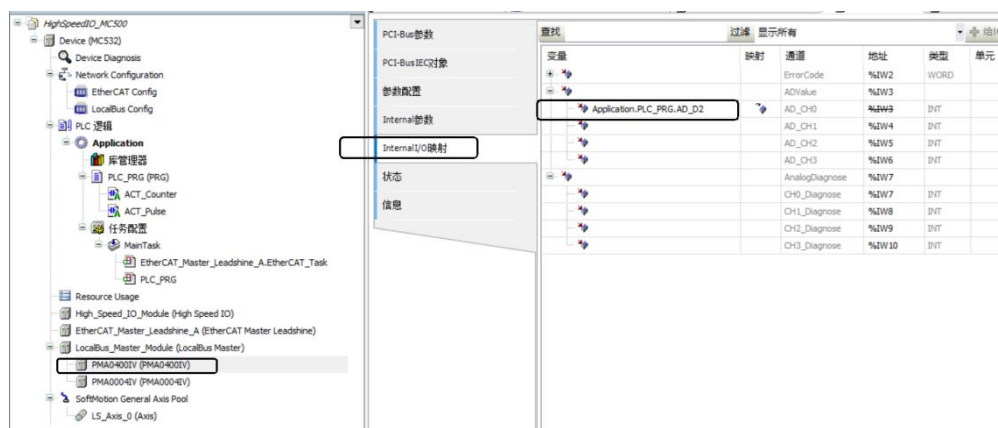


图 11.24 模拟量输出模块映射

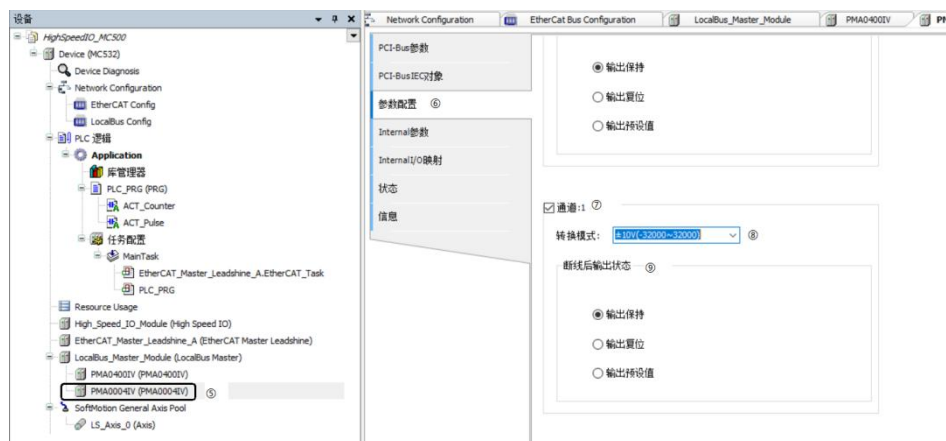


图 11.25 模拟量输出模块映射

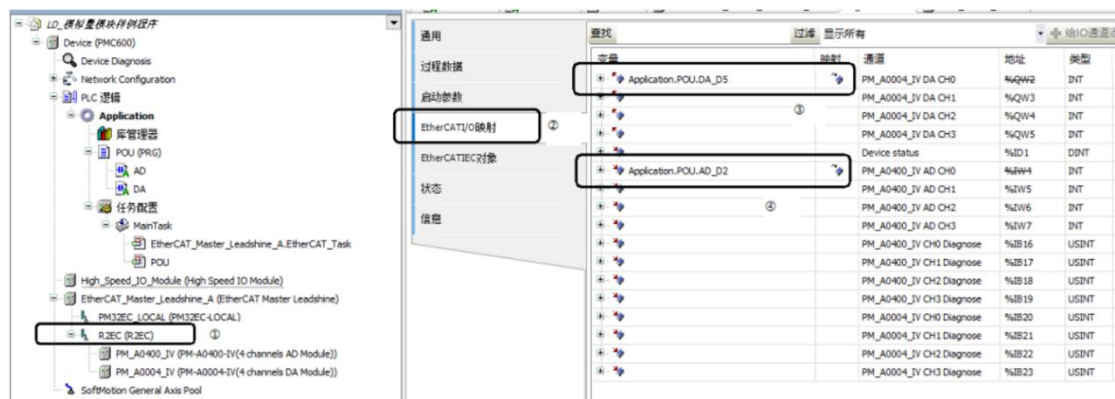


图 11.26 模拟量模块的变量映射

- 6) 与相关变量进行映射后，即可在程序中通过变量读取模拟量输入值、设置模拟量输出值。

第十二章 串口通讯

12.1. 基于 RS485 接口的 Modbus RTU 主从站通讯

12.1.1. Modbus RTU 主从站通讯基础知识

本节介绍 PLC 基于 RS485 接口的 Modbus RTU 主从站通信。通常以一个 PLC 为主站，变频器、伺服电机驱动器和其他 PLC 等设备为从站。

RTU（Remote Terminal Unit）为远程终端单元。

Modbus RTU 主从站通讯过程如图 12.1 所示。Modbus 主站的特点有：

- ① 可以主动发出指令；
- ② 具有唯一性，通讯总线上只能有一个主站；
- ③ 可以对接多个 Modbus 从站。

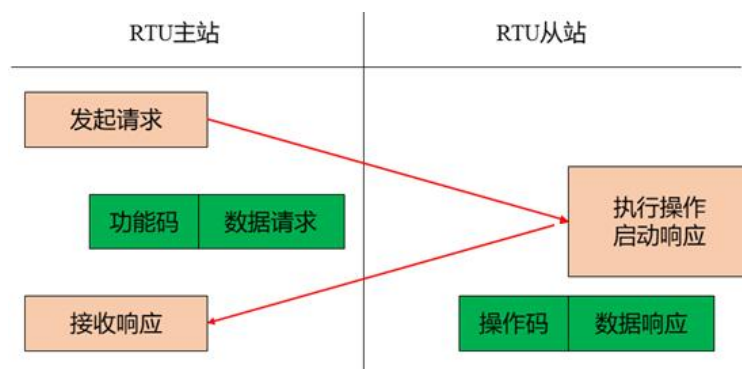


图 12.1 Modbus RTU 主从站通讯过程

12.1.2. Modbus 通讯协议简介

Modbus 协议是一个主/从（master/slave）架构的协议。有一个节点是 master 节点，其他使用 Modbus 协议参与通信的节点是 slave 节点。每一个 slave 设备都有一个唯一的地址。在通讯网络中，只有被指定为主节点的节点可以启动一个命令。一个 Modbus 命令包含一个将执行该命令的设备的 Modbus 地址。所有设备都会

收到该命令，但只有指定地址的设备会执行该命令并回应指令。

所有的 Modbus 命令包含了检查码，以确定到达的命令没有被破坏。基本的 Modbus 命令能指定一个 RTU 改变其寄存器的值，控制或者读取一个 I/O 端口，以及指挥设备回送一个或者多个其寄存器中的数据。

Modbus 协议中按照通信访问的数据宽度划分数据类型，主要有 Bit 型和 Word 型两种数据。

依照行业惯例，本文中有时将 Bit 型变量称为“线圈”或“触点”，将 Word 型变量则称为“寄存器”。

Modbus 协议通信数据帧格式，如图 12.2 所示。

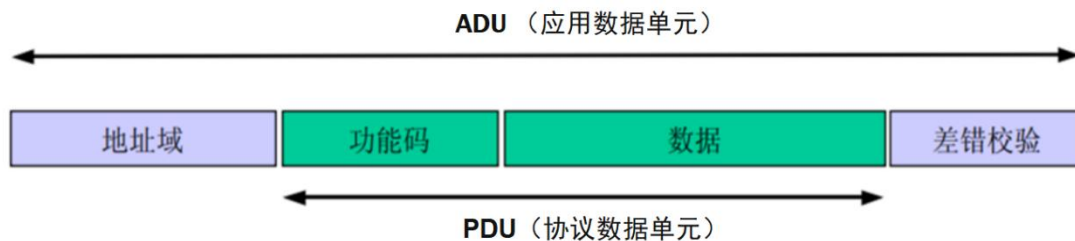


图 12.2 Modbus 通信帧格式

例如：主站要读地址为 1 的从机中的 2 个寄存器的值，两个寄存器的地址为 0004、0005。该指令的能码为 03。报文交互过程如下：

主站的请求命令的通讯帧内容为：01 03 00 04 00 02 85 ca，详见表 12.1。

从站响应该请求，返回相应数据。该动作的功能码为 03。从站发出的通讯帧内容为：01 03 04 00 00 00 00 21 33，详见表 12.2。

表 12.1 主站发出的通讯帧

从机地址	功能码	从站寄存器 起始地址	读取寄存器个数	CRC 校验
01	03	00 04	00 02	85 ca

表 12.2 从站返回的通讯帧

从机地址	功能码	返回字节个数	寄存器 0005 数据	寄存器 0006 数据	CRC 校验
01	03	04	00 00	00 00	21 33

12.1.3. Modbus 协议可访问的内部地址

LC/MC 系列 PLC 的变量区有 Q 区、I 区和 M 区这三种，分别都可以按位、按字节、按字和按双字进行访问。寄存器地址索引规则如图 12.3 所示。

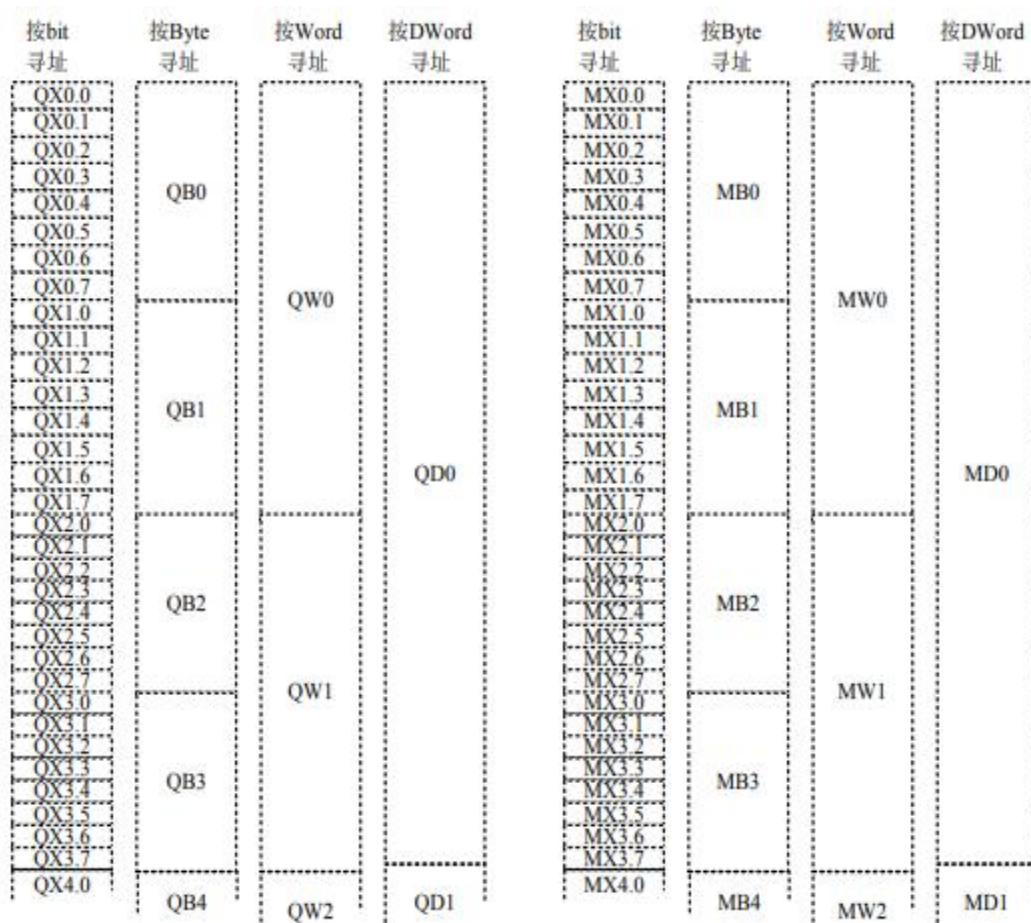


图 12.3 寄存器地址索引规则

为了方便地址的计算，Word 型寄存器的起始地址，遵循的是起始地址为偶数 Byte 地址；DWord 型寄存器的起始地址，遵循的是起始地址为偶数 Word 地址对齐，

其索引号为 2 倍关系的原则。

I 区、Q 区、M 区地址编址规则相同。如 QX、QB、QW、QD 编址如下：

$$QB0 = (QX0.0 \sim QX0.7) ;$$

$$QW0 = (QB0 \sim QB1) = QX0.0 \sim QX0.7) + (QX1.0 \sim QX1.7) ;$$

$$QD0 = (QW0 \sim QW1) = (QB0 \sim QB3) = (QX0.0 \sim QX0.7) + (QX1.0 \sim QX1.7) \\ + (QX2.0 \sim QX2.7) + (QX3.0 \sim QX3.7)$$

标准的 Modbus 协议都可以访问的 PLC 内部 I、Q 区范围如表 12.3 所示。

表 12.3 PLC 内部 I、Q 区范围

地址范围	功能码	起始地址	线圈数量	说明
QW0~QW4095 (QX0.0~QX8191.7)	0X01,0x05,0x0f	0	65536	通用标准 Modbus 协议都可以访问
IW0~IW4095 (IX0.0~IX8191.7)	0X02	0	65536	通用标准 Modbus 协议都可以访问

标准的 Modbus 协议可访问的 PLC 内部 M 区范围如表 12.4 所示。

表 12.4 PLC 内部 M 区范围

地址范围	功能码	起始地址	寄存器数量	说明
MW0~MW65535	0x03,0x06,0x10	0	65536	通用标准 Modbus 协议都可以访问

12.1.4. Modbus 通信功能码

1) 功能码 0x01，读线圈寄存器，可以读取 Q 区变量。

请求帧格式：从机地址+0x01+线圈起始地址+线圈数量+CRC 检验，如表 12.5 所

示。

响应帧格式：从机地址+0x01+字节数+线圈状态+CRC 检验，如表 12.6 所示。

表 12.5 功能码 0x01 请求帧详解

序号	数据 (字节) 意义	字节数量	说明
1	从机地址	1 个字节	取值 1~247
2	0x01	1 个字节	读线圈
3	线圈起始地址	2 个字节	高位在前，低位在后，见线圈编址
4	线圈数量 N	2 个字节	高位在前，低位在后
5	CRC 校验	2 个字节	高位在前，低位在后

表 12.6 功能码 0x01 响应帧详解

序号	数据 (字节) 意义	字节数量	说明
1	从机地址	1 个字节	取值 1~247
2	0x01	1 个字节	读线圈
3	字节数	1 个字节	值：[(N+7) / 8]
4	线圈状态	[(N+7) / 8] 个字节	每 8 个线圈合为一个字节，最后一个若不足 8 位，未定义部分填 0。前 8 个线圈在第一个字节，最地址最小的线圈在最低位。依次类推
5	CRC 校验	2 个字节	高位在前，低位在后

2) 功能码 0x03，读保持寄存器，可以读取 M 区变量。

请求帧格式：从机地址+0x03+寄存器起始地址+寄存器数量+CRC 检验，如表 12.7 所示。

响应帧格式：从机地址+0x03+字节数+寄存器值+CRC 检验，如表 12.8 所示。

表 12.7 功能码 0x03 请求帧详解

序号	数据 (字节) 意义	字节数量	说明
1	从机地址	1 个字节	取值 1~247
2	0x03	1 个字节	读寄存器
3	寄存器起始地址	2 个字节	高位在前，低位在后，见寄存器编址
4	寄存器数量	2 个字节	高位在前，低位在后，N
5	CRC 校验	2 个字节	高位在前，低位在后

表 12.8 功能码 0x03 响应帧详解

序号	数据 (字节) 意义	字节数量	说明
1	从机地址	1 个字节	取值 1~247
2	0x03	1 个字节	读寄存器
3	字节数	1 个字节	值: $N \times 2$
4	寄存器值	$N \times 2$ 个字节	每两字节表示一个寄存器值, 高位在前低位在后。寄存器地址小的排在前面
5	CRC 校验	2 个字节	高位在前, 低位在后

3) 功能码 0x05, 写单个线圈, 可以写 Q 区变量

请求帧格式: 从机地址+0x05+线圈地址+线圈状态+CRC 检验, 如表 12.9 所示。

响应帧格式: 从机地址+0x05+线圈地址+线圈状态+CRC 检验, 如表 12.10 所示。

表 12.9 功能码 0x05 请求帧详解

序号	数据 (字节) 意义	字节数量	说明
1	从机地址	1 个字节	取值 1~247
2	0x05	1 个字节	写单线圈
3	线圈地址	2 个字节	高位在前, 低位在后, 见线圈编址
4	线圈状态	2 个字节	低位在前, 高位在后。非 0 即为有效
5	CRC 校验	2 个字节	高位在前, 低位在后

表 12.10 功能码 0x05 响应帧详解

序号	数据 (字节) 意义	字节数量	说明
1	从机地址	1 个字节	取值 1~247
2	0x05	1 个字节	写单线圈
3	线圈地址	2 个字节	高位在前, 低位在后, 见线圈编址
4	线圈状态	2 个字节	低位在前, 高位在后。非 0 即为有效
5	CRC 校验	2 个字节	高位在前, 低位在后

4) 功能码 0x06, 写单个寄存器, 可以写 M 区变量

请求帧格式: 从机地址+0x06+寄存器地址+寄存器值+CRC 检验, 如表 12.11 所示。

响应帧格式: 从机地址+0x06+寄存器地址+寄存器值+CRC 检验, 如表 12.12 所示。

表 12.11 功能码 0x06 请求帧详解

序号	数据 (字节) 意义	字节数量	说明
1	从机地址	1 个字节	取值 1~247
2	0x06	1 个字节	写单寄存器
3	寄存器地址	2 个字节	高位在前, 低位在后, 见寄存器值编址
4	寄存器值	2 个字节	高位在前, 低位在后。非 0 即为有效
5	CRC 校验	2 个字节	高位在前, 低位在后

表 12.12 功能码 0x06 响应帧详解

序号	数据 (字节) 意义	字节数量	说明
1	从机地址	1 个字节	取值 1~247
2	0x06	1 个字节	写单寄存器
3	寄存器地址	2 个字节	高位在前, 低位在后, 见寄存器值编址
4	寄存器值	2 个字节	高位在前, 低位在后。非 0 即为有效
5	CRC 校验	2 个字节	高位在前, 低位在后

5) 功能码 0x0f, 写多个线圈, 可以写连续的多个 Q 区变量。

请求帧格式: 从机地址+0x0f+线圈起始地址+线圈数量+字节数+线圈状态+CRC 检验, 如表 12.13 所示。

响应帧格式: 从机地址+0x0f+线圈起始地址+线圈数量+CRC 检验, 如表 12.14 所示。

表 12.13 功能码 0x0f 请求帧详解

序号	数据 (字节) 意义	字节数量	说明
1	从机地址	1 个字节	取值 1~247
2	0x0f	1 个字节	写多个单线圈
3	线圈起始地址	2 个字节	高位在前, 低位在后, 见线圈编址
4	线圈数量 N	2 个字节	高位在前, 低位在后。最大为 1968
5	字节数	1 个字节	值: 值: $\lceil (N+7) / 8 \rceil$
6	线圈状态	$\lceil (N+7) / 8 \rceil$ 个字节	每 8 个线圈合为一个字节, 最后一个若不足 8 位, 未定义部分填 0。前 8 个线圈在第一个字节, 最地址最小的线圈在最低位。依次类推
7	CRC 校验	2 个字节	高位在前, 低位在后

表 12.14 功能码 0x0f 响应帧详解

序号	数据 (字节) 意义	字节数量	说明
1	从机地址	1 个字节	取值 1~247
2	0x0f	1 个字节	写多个单线圈
3	线圈起始地址	2 个字节	高位在前, 低位在后, 见线圈编址
4	线圈数量	2 个字节	高位在前, 低位在后。
5	CRC 校验	2 个字节	高位在前, 低位在后

6) 功能码 0x10, 写多个寄存器, 可以写连续的多个 M 区变量

请求帧: 从机地址+0x10+寄存器起始地址+寄存器数量+字节数+寄存器值+CRC 检验, 如表 12.15 所示。

响应帧格式: 从机地址+0x10+线圈起始地址+线圈数量+CRC 检验, 如表 12.16 所示。

表 12.15 功能码 0x10 请求帧详解

序号	数据 (字节) 意义	字节数量	说明
1	从机地址	1 个字节	取值 1~247
2	0x10	1 个字节	写多个寄存器
3	寄存器起始地址	2 个字节	高位在前, 低位在后, 见寄存器编址
4	寄存器数量	2 个字节	高位在前, 低位在后, 数量为 N
5	字节数	1 个字节	值: $N*2$
6	寄存器值	$N*2$ ($N*4$)	
7	CRC 校验	2 个字节	高位在前, 低位在后

表 12.16 功能码 0x10 响应帧详解

序号	数据 (字节) 意义	字节数量	说明
1	从机地址	1 个字节	取值 1~247
2	0x10	1 个字节	写多个寄存器
3	寄存器起始地址	2 个字节	高位在前, 低位在后, 见寄存器编址
4	寄存器数量	2 个字节	高位在前, 低位在后, N
5	CRC 校验	2 个字节	高位在前, 低位在后

若想详细了解 Modbus 协议可参见《Modbus 协议规范》。

12.2. RS485 Modbus RTU 主站通讯

12.2.1. LC 系列 PLC 的 RTU 主站配置

1) 使能 PLC 作为 Modbus RTU 主站

鼠标双击网络组态中的“通讯配置”，如图 12.4 所示，单击 PLC 设备，弹出所支持的通信协议的使能窗口。勾选“Modbus 主站”，使能后，变为红色。Modbus 主站使能后，左侧树中，出现了对应的通信配置，如图 12.4 中④处所示“ModbusCOM_Master_1”。

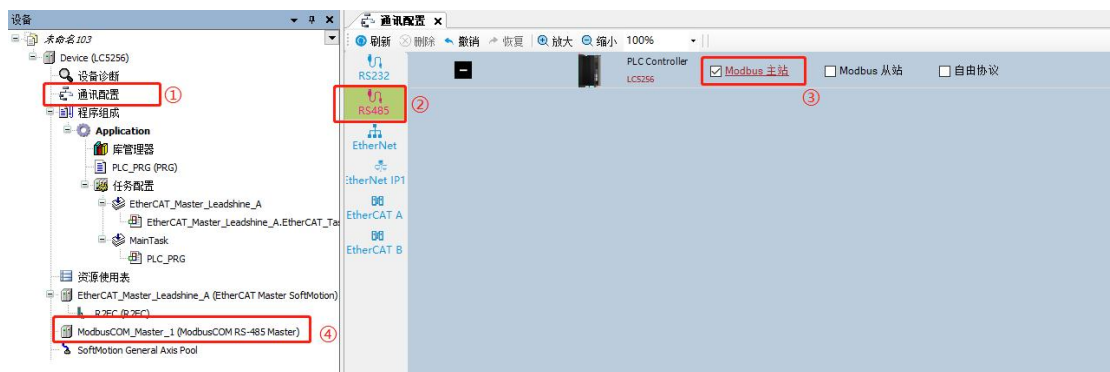


图 12.4 使能 PLC 作为 Modbus RTU 主站

2) 添加 Modbus RTU 从站设备

使能 PLC 作为 Modbus RTU 主站后，如图 12.5 所示，从右侧“网络设备”COM R485 中拖入或双击“ModbusCOM Slave”从站设备，拖到蓝色的总线“Modbus_Master_COM1”下面，从站设备添加后，左侧树中，出现了对应的通信配置，如图 12.5 中③处所示“Modbus_Com_Slave”。如果有多个从站，可重复以上步骤添加多个从站设备。

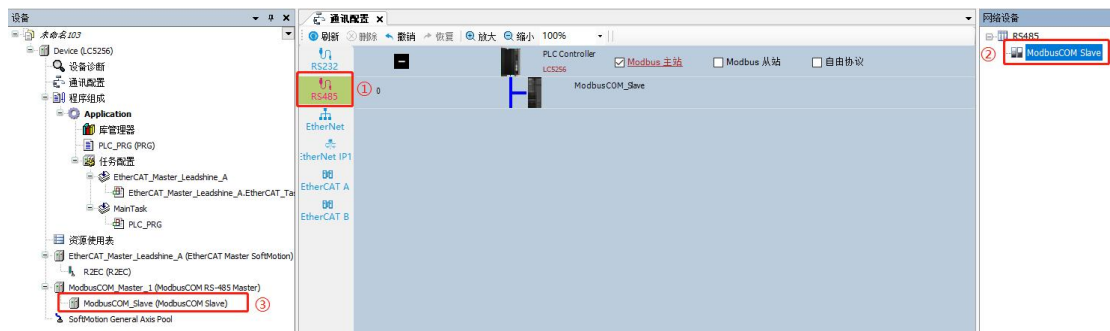


图 12.5 添加从站设备

3) Modbus RTU 主站配置

双击设备树中的主站设备“ModbusCOM_Master_1”，打开 Modbus 主站配置窗口，如图 12.6 所示。

Modbus 主站配置相关参数与配置示例，如表 12.17、表 12.18 所示。

注意：Modbus 主从站通信参数配置必须一致，才能正常通信。

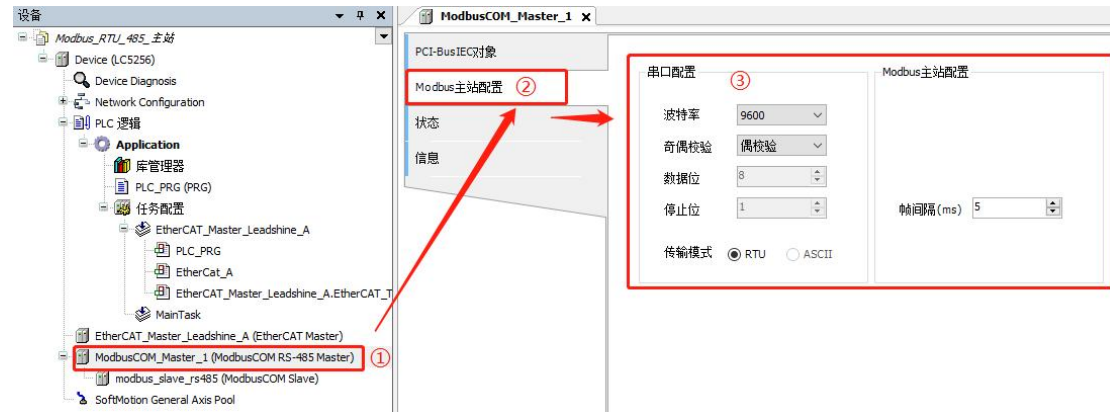


图 12.6 Modbus 主站配置

表 12.17 Modbus 主站配置相关参数

配置项	功能
波特率	通信时的速率，支持 4800-115200
奇偶校验	通信帧的校验方式，与停止位匹配使用
数据位	RTU 模式下为 8，ASCII 模式下为 7
停止位	奇/偶校验时为 1，无校验时为 2
传输模式	RTU
帧间隔	主站接收上一个响应数据帧到下一个请求数据帧之间等待的时间间隔, RTU 模式下，两帧由时长至少 3.5 个字符时间区分， 估算公式： $(3.5 * 11 * 1000 / \text{波特率})\text{ms}$

表 12.18 配置示例

配置项	配置值
波特率	115200
奇偶校验	偶校验
数据位	8
停止位	1
传输模式	RTU
帧间隔	2ms

4) Modbus RTU 从站设备配置

a) Modbus 从站设置

双击设备树中的主站设备“Modbus_slave_rs485”，打开 Modbus 从站配置窗口，如图 9.7 所示。

Modbus 从站配置相关参数，如表 12.19 所示。

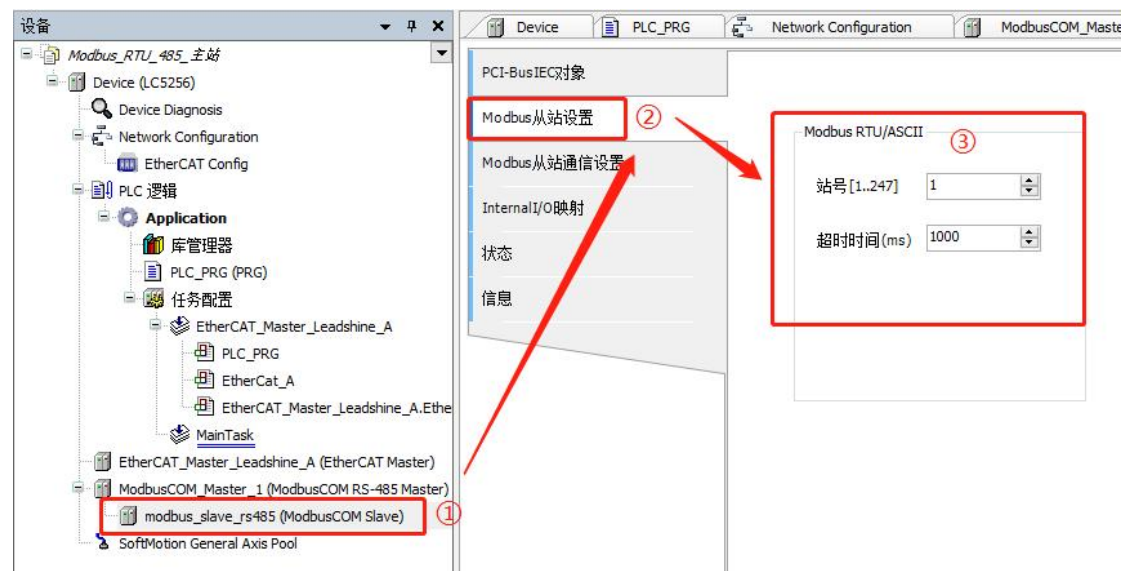


图 12.7 Modbus 从站配置

表 12.19 Modbus 从站配置相关参数

配置项	功能
从站站号	标识从站号，范围 1~247
超时时间	主站发帧后，超过该时间从站未响应，主站报接收超时

b) Modbus 从站通讯设置

Modbus 从站设置完相关参数后，需要配置从站设备通讯数据的存储类型、控制模式、读/写寄存器等参数，操作步骤如图 12.8 所示。

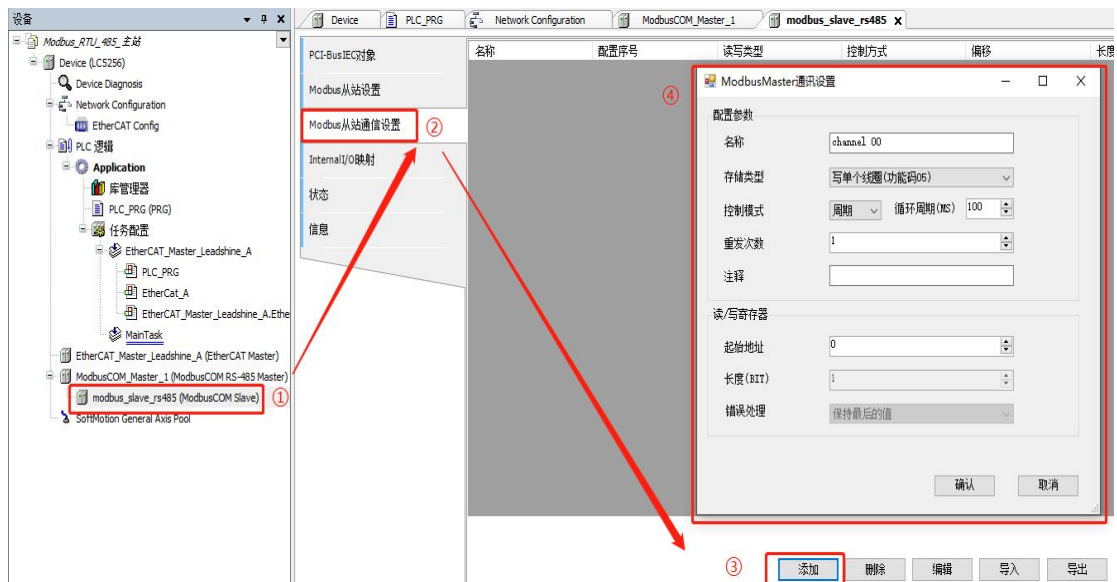


图 12.8 Modbus 从站通讯设置

ModbusMaster 通讯设置相关参数以及功能码详细说明，如表 12.20、表 12.21 所示。按照参数配置要求设置通讯参数，配置完成后如图 12.9 所示。

表 12.20 ModbusMaster 通讯设置相关参数

序号	配置项	功能
		读线圈状态（功能码 01）
		读保持寄存器（功能码 03）

1	功能码	写单个线圈（功能码 05） 写单个寄存器（功能码 06） 写多个线圈（功能码 15） 写多个寄存器（功能码 16）
2	触发器	循环执行：周期触发的请求。 电平触发：编程进行改变触发变量电平时触发请求。（触发变量在 IO 映射中设置）
3	重发次数	本次发生通信故障未获得从站返回帧，则按重发次数进行重新发送。
4	注释	可以对数据进行描述的简短文本区域。
5	起始地址	读\写的寄存器位置开始地址。
6	长度	读取的寄存器个数。
7	错误处理	保持最后的值：使数据保持最后一次的有效值。

表 12.21 功能码详细说明

功能码	访问类型	寄存器个数
01	读线圈状态	1~125
03	读保持寄存器	1~125
05	写单个线圈	1
06	写单个寄存器	1
15	写多个线圈	1~125
16	写多个寄存器	1~125

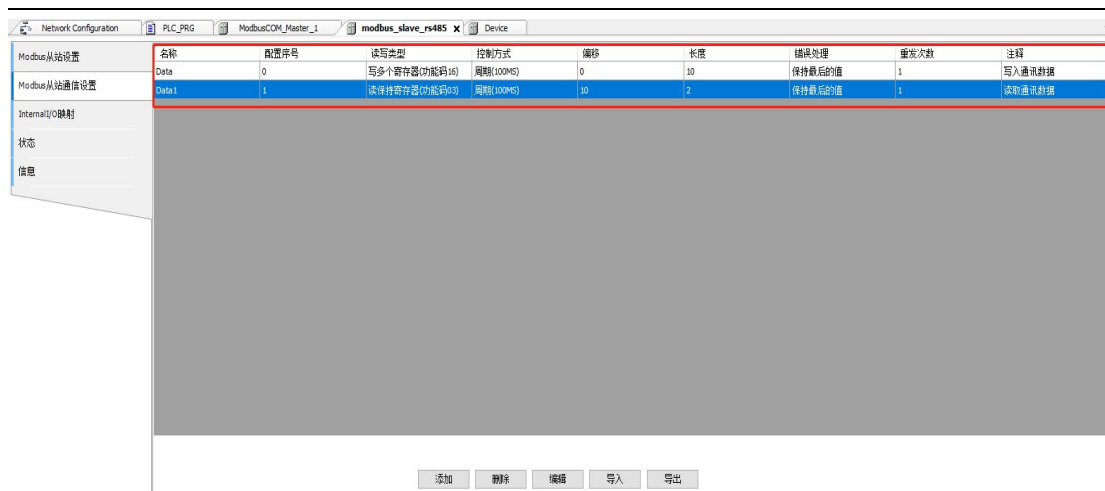


图 12.9 从站设备通讯数据

5) 从站通讯配置导入导出功能

LC 系列 PLC 支持 Modbus 从站通讯配置参数导入、导出功能。导入和导出的文件为 xml 格式。导入、导出参数的过程如图 12.10 和 12.11 所示。

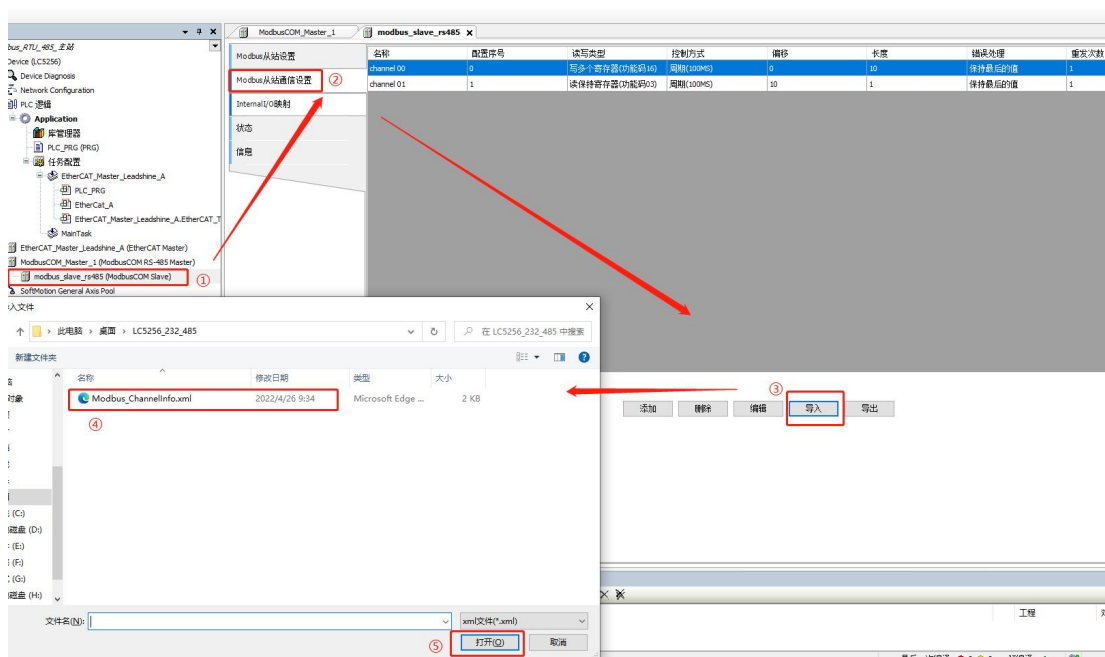


图 12.10 导入从站通讯配置参数的过程

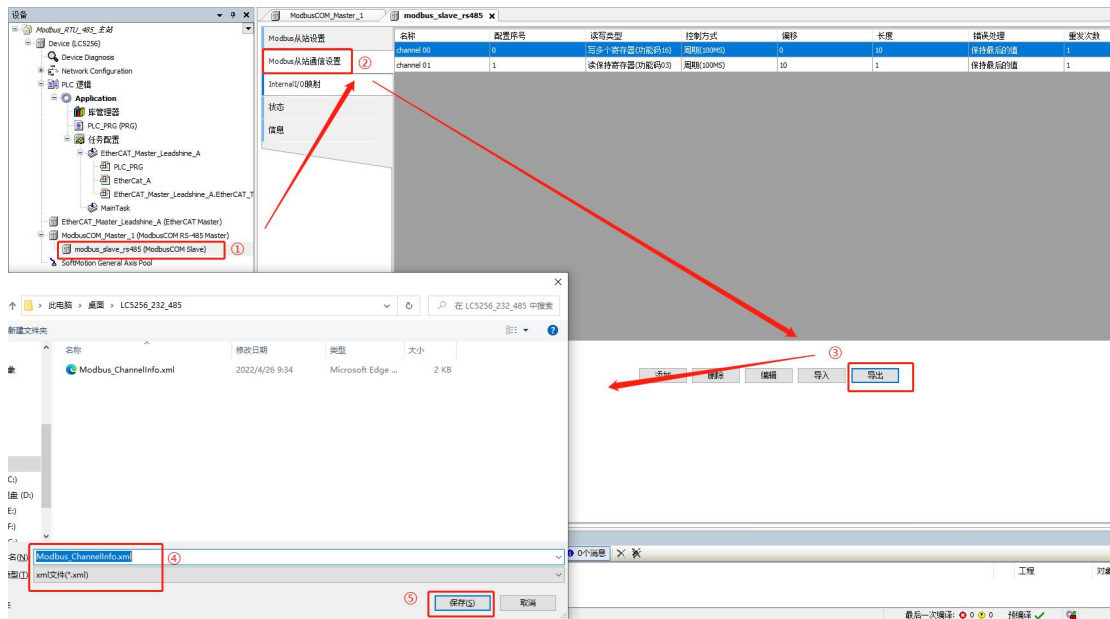


图 12.11 导出从站通讯配置参数的过程

6) Modbus 从站 Internal I/O 映射

在 Modbus 从站通信设置中添加主从站通信配置后，Internal I/O 映射中会自动分配每条配置的 PLC 内部映射地址，如图 12.12 所示，第一行的 %QW0 表示第一个寄存器值将写入到 PLC 内部 %QW0 这个地址。

通过直接输入示例变量路径（如图 12.13 所示）或者声明变量（如图 12.14 所示），将程序中的自定义变量映射到 I/O 地址。

变量	映射	通道	地址	类型	单元	描述
		错误码	%IW0	WORD		Modbus_RTU 通讯错误码
+		Data	%QW0	ARRAY [0..9] OF WORD		写多个寄存器
+		Data	%IX2.0	BOOL		处理完成状态
+		Data 1	%IW2	ARRAY [0..0] OF WORD		读多个寄存器
+		Data 1	%IX6.0	BOOL		处理完成状态

图 12.12 Internal I/O 映射

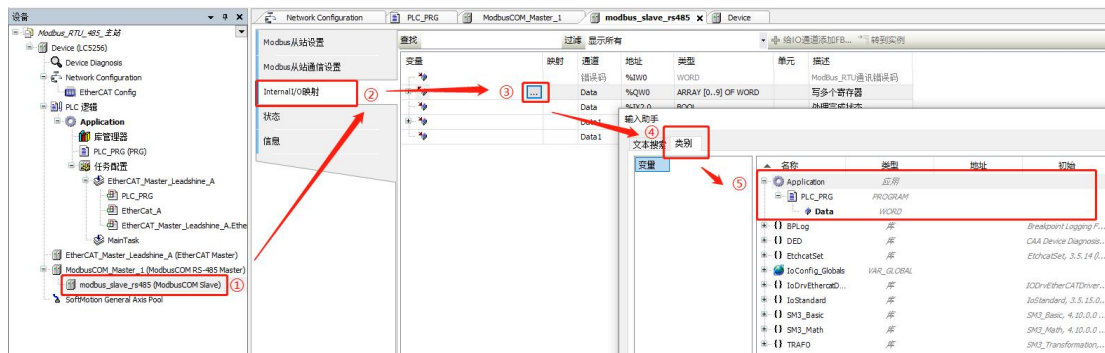


图 12.13 Internal I/O 映射输入变量路径

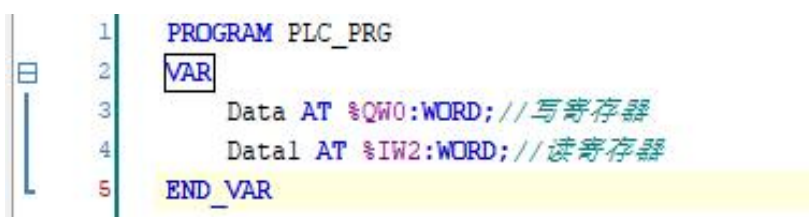


图 12.14 Internal I/O 映射

7) Modbus RTU 常见故障与错误码

Modbus 主站连接从站时发生的主要故障如下：

- Modbus 主站与 Modbus 从站配置不一致，导致主站与从站通信无法建立。
- Modbus 主站访问 Modbus 从站非法地址，返回错误应答。
- Modbus 主站操作 Modbus 从站写寄存器，但是 Modbus 从站该寄存器只支持读不支持写操作，Modbus 主站会收到 Modbus 从站返回的出错应答。
- 报文中的错误码与 LeadSys Studio 编程软件在线调试显示的错误码关系为：
在线调试显示的错误码 = error + 40000。错误码定义，如表 12.22 所示。

表 12.22 错误码定义

故障码	说明	故障码	说明
40001	非法的功能码	40011	目标设备未能回应
40002	非法的数据地址	40012	无效的 CRC
40003	非法的数据值	40013	无效的数据

40004	从站设备故障	40014	无效的异常码
40005	AK 异常	40015	保留
40006	从站设备忙	40016	数据过多
40007	否定应答	40017	响应与查询地址不匹配
40008	内存奇偶校验错误	40018	Modbus TCP 连接超时
40009	未定义	40019	Modbus TCP 通讯掉线
40010	网关路径不可用		

e) 错误码查询与错误响应帧格式

- ① 通过 PLC 在线调试，可以查看当前通信错误码，正常通信时，错误码为 0，如图 12.15 所示。

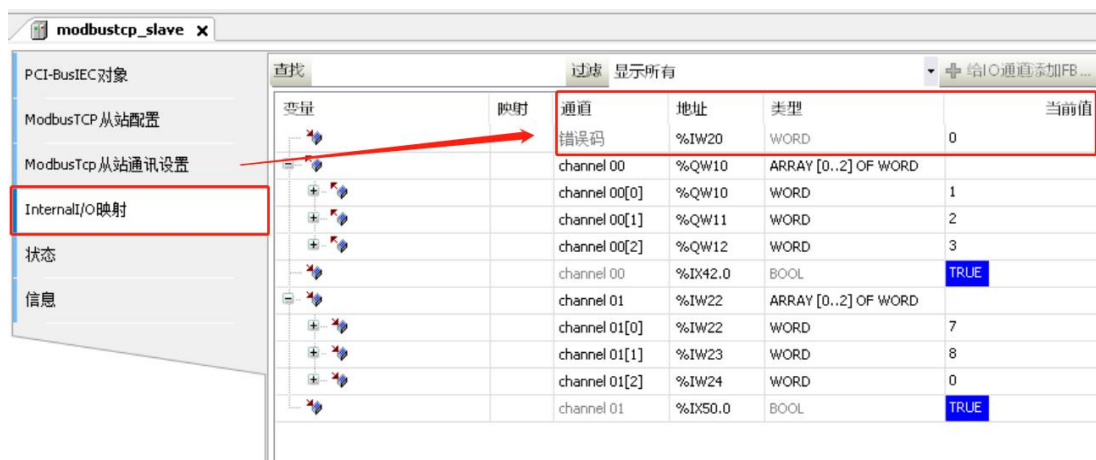


图 12.15 错误码查询

例：如图 12.16 所示，当通信错误时，错误码为 40017，查表 9.22 可知当前错误为“响应与查询地址不匹配”。



变量	映射	通道	地址	类型	当前值	预备值	单元
		错误码	%IW12	WORD	40017		
		channel 00	%QW5	ARRAY [0..4] OF WORD			
Application.GVL1.ww1		channel 00[0]	%QW5	WORD	1		
Application.GVL1.ww2		channel 00[1]	%QW6	WORD	12		
		channel 00[2]	%QW7	WORD	0		

图 12.16 错误码

② 错误响应帧格式：（本错误帧适合所有的操作命令帧）

错误响应帧：从机地址 +（命令码 +0x80）+ 错误码 +CRC 校验。

以从机地址为 1，功能码 03 为例，介绍错误响应帧：

主站请求：01 05 00 0e 00 00 ac 09

从站响应：01 85 02 c3 51，如表 12.23 所示

表 12.23 从站响应

从机地址	错误响应码	返回错误码	CRC 校验
01	85	02	c3 51

0x85 是功能码 0x05 的错误响应码，返回错误码 = 02。

根据 LeadSys Studio 编程软件对应的显示规则，可知错误码为 40002，查表 9.22 可知，该错误为“非法的数据地址”。

注：PMC600 不支持该错误码。

12.2.2. MC 系列 PLC 的 RTU 主站配置

1) 使能 PLC 作为 Modbus RTU 主站

双击网络组态中的“通讯配置”，如图 12.17 所示，单击 PLC 设备，弹出所支持的通信协议的使能窗口。勾选“Modbus 主站”，使能后，变为红色。Modbus 主站使能后，左侧树中，出现了对应的通信配置，如图 12.17 中④处所示

“ModbusCOM_Master_1”。

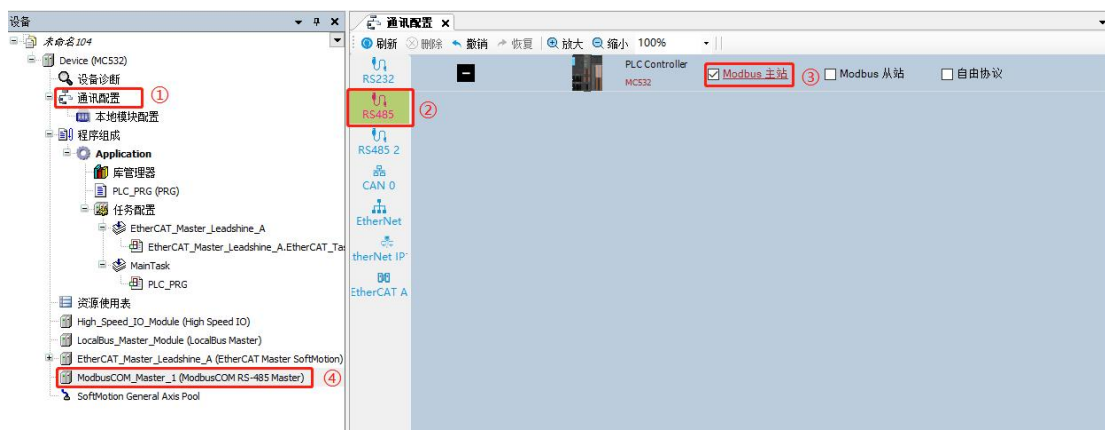


图 12.17 从站设备通讯数据

2) Modbus RTU 主站配置

双击设备树中的主站设备“Modbus_Master_RS485_COM”，打开 Modbus 主站配置窗口，如图 12.18 所示。

注意：Modbus 主从站通信参数配置必须一致，才能正常通信。

Modbus 主站配置相关参数与配置示例，见表 9.17、表 9.18。

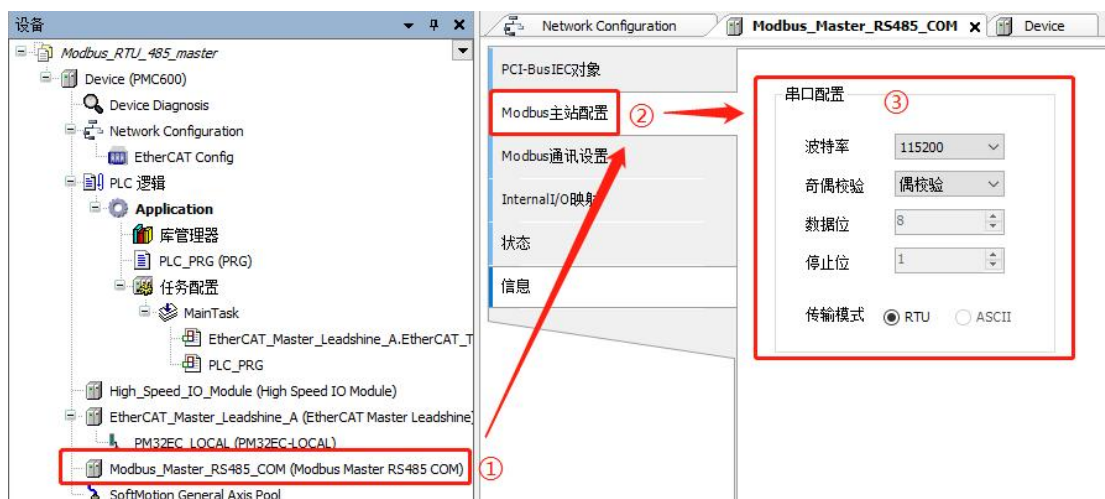


图 12.18 Modbus 主站配置

3) Modbus RTU 从站设备配置

双击设备树中的主站设备“Modbus_Master_RS485_COM”，配置从站设备的从

站号（范围 1~247）、通讯数据的存储类型、控制模式、读/写寄存器等参数，操作步骤如图 12.19 所示。

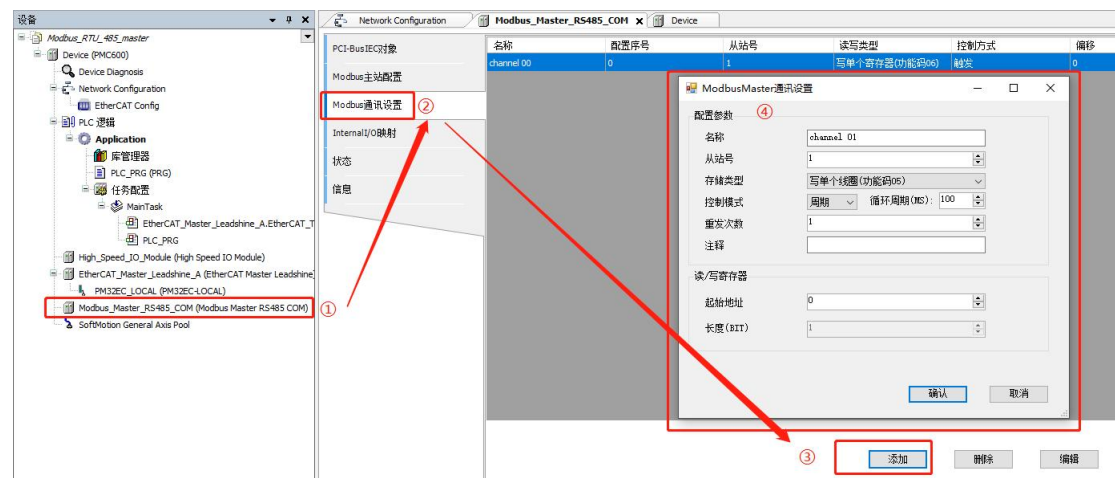


图 12.19 Modbus 从站通讯设置

ModbusMaster 通讯设置相关参数以及功能码详细说明，见上表 9.20、表 9.21 所示。按照参数配置要求设置通讯参数，配置完成后如图 12.20 所示。

4) 其它相关参数配置

MC 系列其它参数基本与 LC 系列相同，请参见 12.1、12.2.1 节。

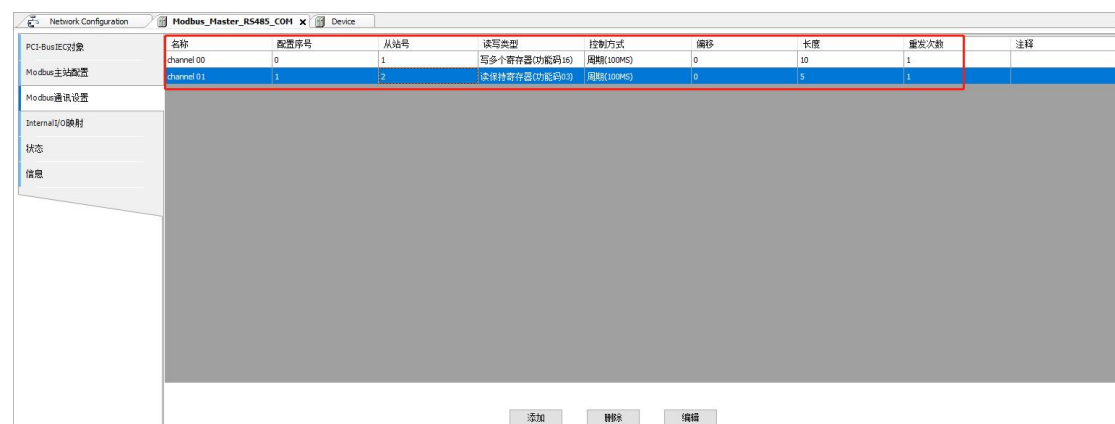


图 12.20 从站设备通讯数据

12.3. RS485 Modbus RTU 从站通讯

本节介绍 PLC 作为 ModbusRTU 从站，上位机、触摸屏等设备作为主站的通讯功能使用方法。

Modbus 从站特点：

- ① 只能接收主站发出的指令；
- ② 不具有唯一性，通讯总线上可以有多个从站；
- ③ 只能对接一个 Modbus 主站。

12.3.1. LC 系列 PLC RTU 从站的配置

1) 使能 PLC 作为 Modbus RTU 从站

鼠标双击网络组态中的“Network Configuration”，如图 12.21 所示，单击 PLC 设备，弹出所支持的通信协议的使能窗口。勾选“Modbus 从站”，使能后，变为红色。Modbus 从站使能后，左侧树中，出现了对应的通信配置，如图 12.21 中④处所示“Modbus_Slave_RS485_COM”。

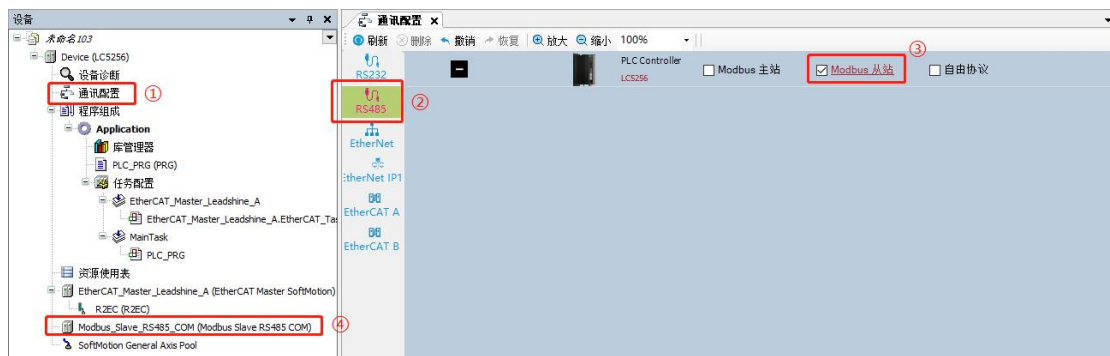


图 12.21 使能 PLC 作为 Modbus RTU 从站

2) Modbus RTU 从站配置

双击设备树中的从站设备“Modbus_Slave_RS485_COM”，打开 Modbus 从站配置窗口，如图 12.22 所示。

Modbus 从站配置相关参数与配置示例，请参见表 9.17、表 9.18。

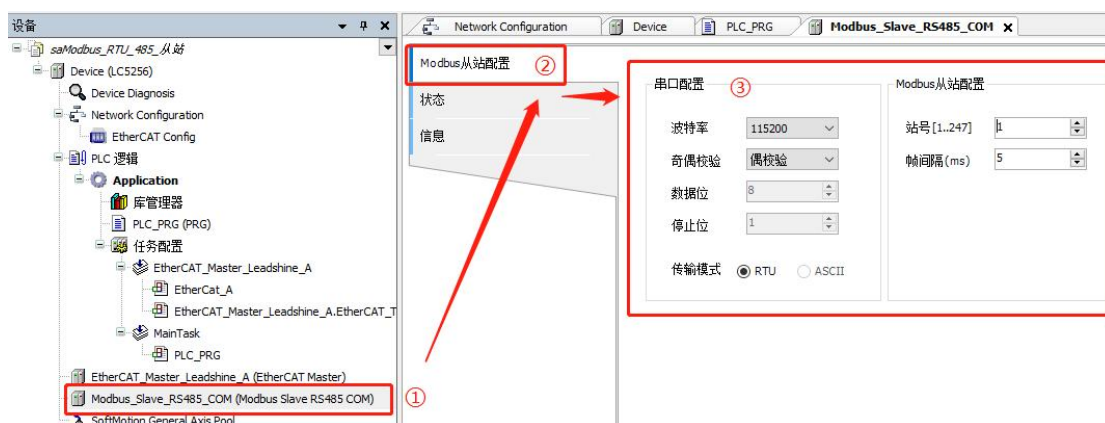


图 12.22 使能 PLC 作为 Modbus RTU 从站

3) 其它相关参数配置

请参见 12.1、12.2 节内容。

12.3.2. MC 系列 PLC RTU 从站的配置

MC 系列 PLC 从站使能、配置方法与 LC 系列的相同。

另外，PMC600 系列 PLC 可以使用 PMC_Communication 库，如图 12.23 所示，可在程序中直接调用，且有对应的 Modbus 地址。若无该库也可以通过“添加库”→“高级”→输入“PMC_Communication”进行添加。

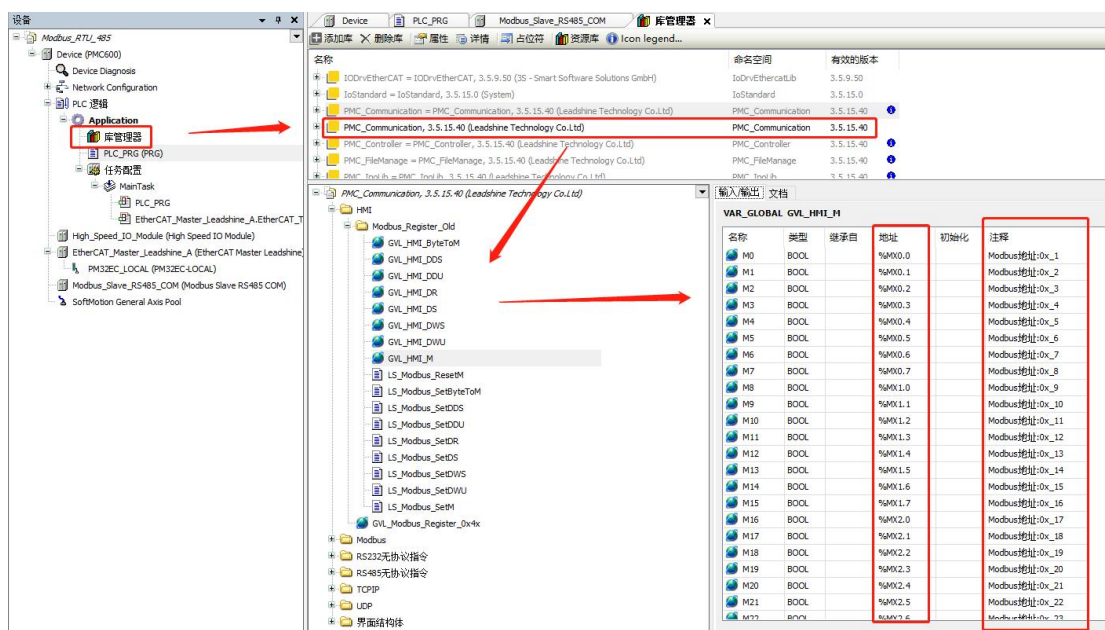


图 12.23 PMC_Communication 库

12.4. RS485 Modbus RTU 通讯例程

本例程采用 LC5256 PLC 作为 Modbus RTU 主站,PMC600 PLC 作为 Modbus RTU 从站,通过读、写操作线圈和寄存器的方式,传输 PLC 状态、读取当前坐标等数据,简单演示基于 RS485 的 Modbus RTU 通讯。

1) 硬件接线,如图 12.24 所示。

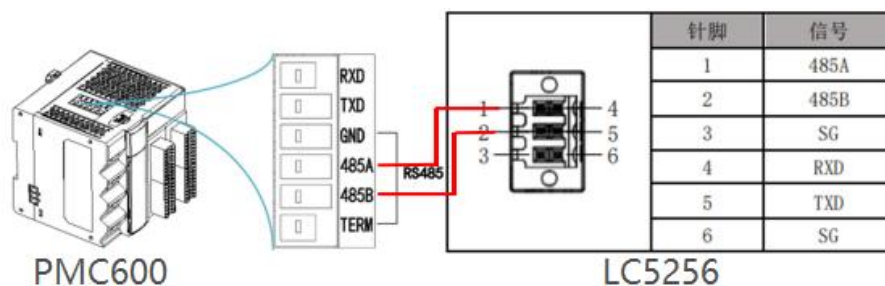


图 12.24 PMC600 与 LC5256 的 RS485 接线图

2) PLC 主从站通讯配置

① LC5256 PLC 主站与从站设备通讯配置,如图 12.25 至图 12.28 所示。

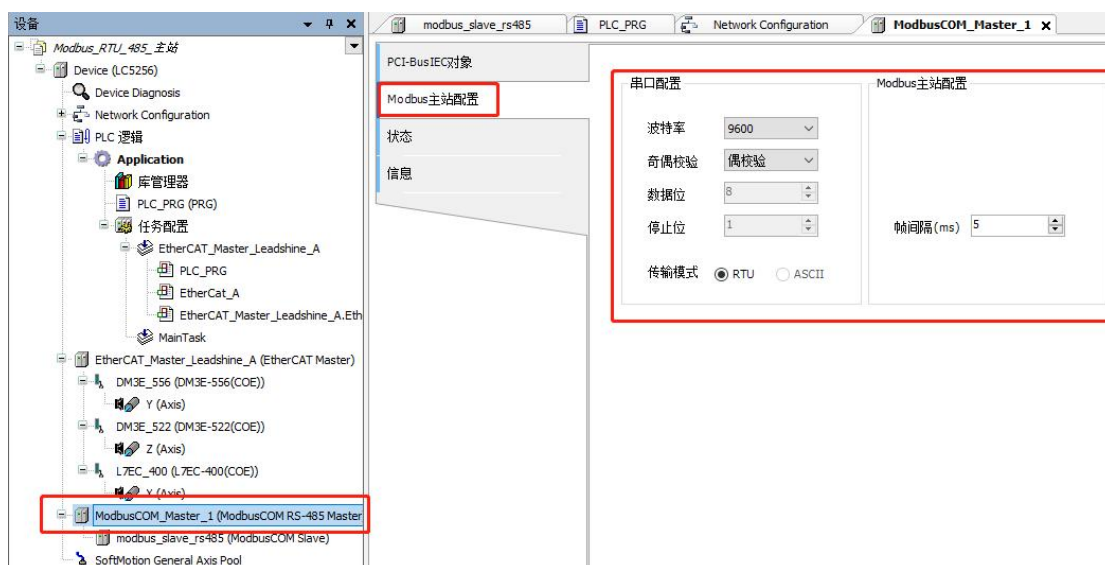


图 12.25 Modbus 主站配置

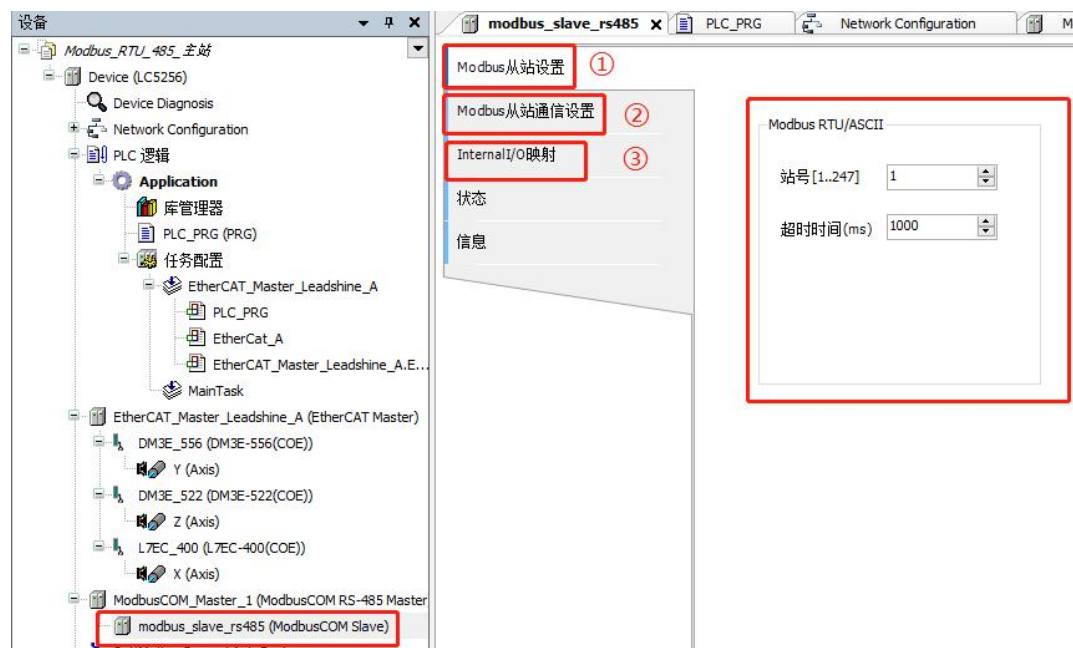


图 12.26 Modbus 从站设备配置-从站设置

名称	配置序号	读写类型	控制方式	偏移	长度	错误处理	重复次数	注释
LC5256状态	0	写单个线圈(功能码05)	周期(1000MS)	0	1	保持最后的值	1	
PMC600状态	1	读线圈寄存器(功能码01)	周期(1000MS)	10	1	保持最后的值	1	
Xpos	2	写多个寄存器(功能码16)	周期(1000MS)	100	1	保持最后的值	1	X轴位置数据
Ypos	3	写多个寄存器(功能码16)	周期(1000MS)	101	1	保持最后的值	1	Y轴位置数据
Zpos	4	写多个寄存器(功能码16)	周期(1000MS)	102	1	保持最后的值	1	Z轴位置数据

图 12.27 Modbus 从站设备配置-从站通讯设置

变量	映射	通道	地址	类型	单元	描述
Application.PLC_PRG.LC5256status		错误码	%IW36	WORD		Modbus_RTU通讯错误码
		LC5256状态	%QW17	ARRAY [0..0] OF WORD		写单个线圈
		LC5256状态[0]	%QW17	WORD		
		Bit0	%QX34.0	BOOL		Write #0000
		Bit1	%QX34.1	BOOL		
		Bit2	%QX34.2	BOOL		
		Bit3	%QX34.3	BOOL		
		Bit4	%QX34.4	BOOL		
		Bit5	%QX34.5	BOOL		
		Bit6	%QX34.6	BOOL		
		Bit7	%QX34.7	BOOL		
		Bit8	%QX35.0	BOOL		
		Bit9	%QX35.1	BOOL		
		Bit10	%QX35.2	BOOL		
		Bit11	%QX35.3	BOOL		
		Bit12	%QX35.4	BOOL		
		Bit13	%QX35.5	BOOL		
		Bit14	%QX35.6	BOOL		
		Bit15	%QX35.7	BOOL		
		LC5256状态	%IX74.0	BOOL		处理完成状态
		PMC600状态	%IW38	ARRAY [0..0] OF WORD		读多个线圈
		PMC600状态	%IX78.0	BOOL		处理完成状态
Application.PLC_PRG.Xpos		Xpos	%QW18	ARRAY [0..0] OF WORD		写多个寄存器
Application.PLC_PRG.Ypos		Ypos	%QW19	ARRAY [0..0] OF WORD		写多个寄存器
Application.PLC_PRG.Zpos		Zpos	%QW20	ARRAY [0..0] OF WORD		写多个寄存器
		Xpos	%IX78.1	BOOL		处理完成状态
		Ypos	%IX78.2	BOOL		处理完成状态
		Zpos	%IX78.3	BOOL		处理完成状态

图 12.28 Modbus 从站设备配置-I/O 映射

② PMC600 PLC 从站通讯配置，如图 12.29 所示。

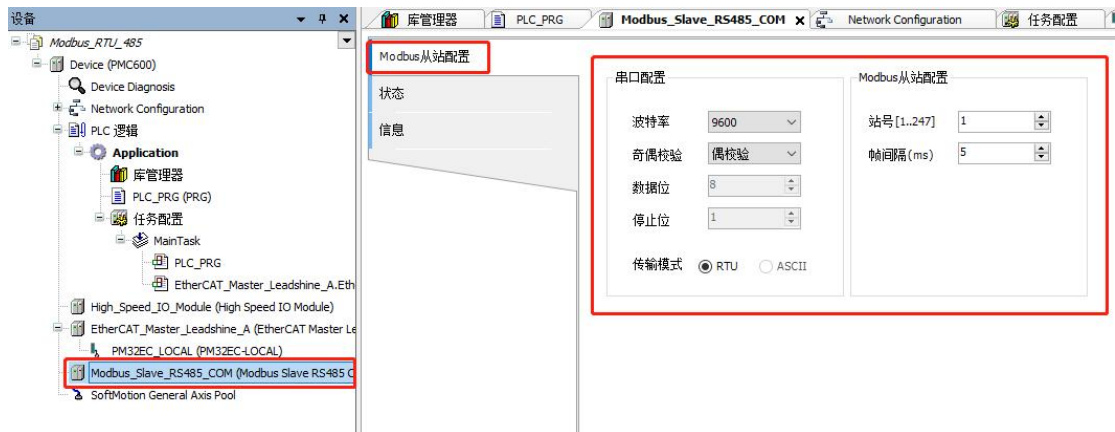


图 12.29 Modbus 从站配置

3) Modbus 主站（LC5256）的程序代码

PROGRAM PLC_PRG//声明变量

VAR

LC5256Stauts:BOOL;//主站 LC5256 状态

PMC600Stauts:BOOL;//从站 PMC600 状态

Xpos:WORD;//X 坐标

Ypos:WORD;//Y 坐标

Zpos:WORD;//Z 坐标

X_ReadActualPosition:MC_ReadActualPosition;//读取 X 轴坐标

Y_ReadActualPosition:MC_ReadActualPosition;//读取 X 轴坐标

Z_ReadActualPosition:MC_ReadActualPosition;//读取 X 轴坐标

END_VAR

//程序内容

X_ReadActualPosition(Axis:=X, Enable:= TRUE, Valid=>, Busy=>, Error=> ,

```
ErrorID=> , Position=> ); //X 轴坐标  
Y_ReadActualPosition(Axis:=Y , Enable:= TRUE, Valid=> , Busy=> , Error=> ,  
ErrorID=> , Position=> ); //Y 轴坐标  
Z_ReadActualPosition(Axis:=Z , Enable:= TRUE, Valid=> , Busy=> , Error=> ,  
ErrorID=> , Position=> ); //Z 轴坐标  
Xpos:=LREAL_TO_WORD(X_ReadActualPosition.Position);  
Ypos:=LREAL_TO_WORD(Y_ReadActualPosition.Position);  
Zpos:=LREAL_TO_WORD(Z_ReadActualPosition.Position);
```

4) Modbus 从站（PMC600）的程序代码

```
PROGRAM PLC_PRG//声明变量  
VAR  
    LC5256Stauts:BOOL;//主站 LC5256 状态  
    PMC600Stauts:BOOL;//从站 PMC600 状态  
    Xpos:LREAL;//X 坐标  
    Ypos:LREAL;//Y 坐标  
    Zpos:LREAL;//Z 坐标  
END_VAR  
  
//程序内容  
LC5256Stauts:=PMC_Communication.M0;  
PMC_Communication.M10:=PMC600Stauts;  
Xpos:=PMC_Communication.DWS[100];  
Ypos:=PMC_Communication.DWS[101];  
Zpos:=PMC_Communication.DWS[102];
```

5) 程序运行结果,如图 12.30 所示。

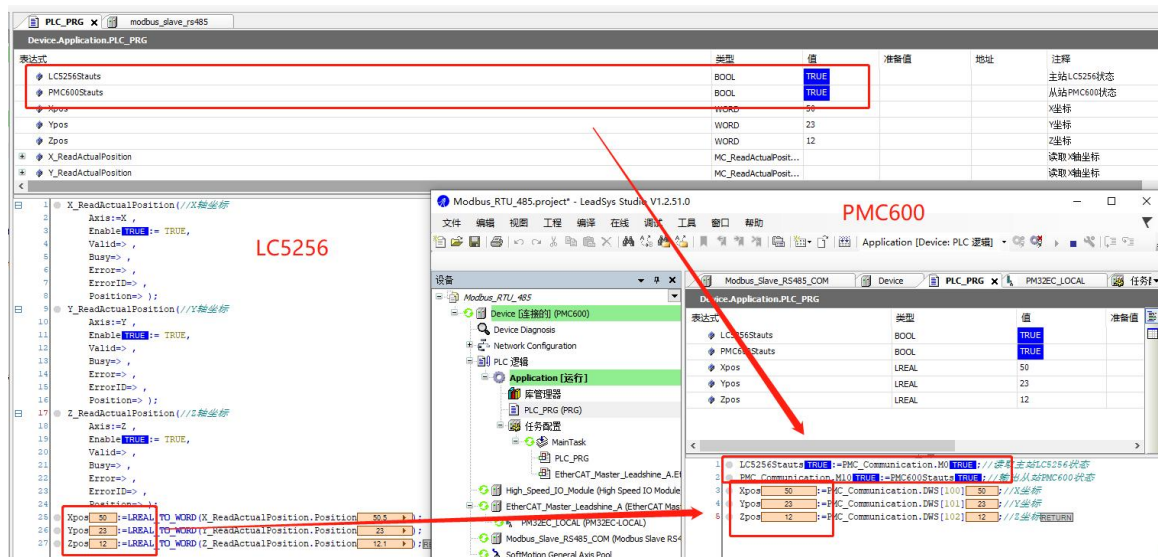


图 12.30 程序运行结果

12.5. RS485 自由协议通讯

自由协议通讯也称无协议通讯，需要根据对方设备的通讯数据格式编写一个临时协议，虽然随着标准协议（Modbus，USS 等）的普及，自由协议通讯应用越来越少，但是对于一些小的设备如扫码枪、LCD 显示屏等，由于没有集成标准通讯协议，所以只能选用自由协议通讯。

12.5.1. 自由协议通讯的使用

1) 485 总线使能自由协议通讯

鼠标双击网络组态中的“通讯配置”，如图 12.31 所示，单击 PLC 设备，弹出所支持的通信协议的使能窗口。勾选“自由协议”，使能后，变为红色。自由协议使能后，左侧树中，出现了对应的通信配置，如图 12.31 中④处所示“Free_Protocol_RS485_COM”。

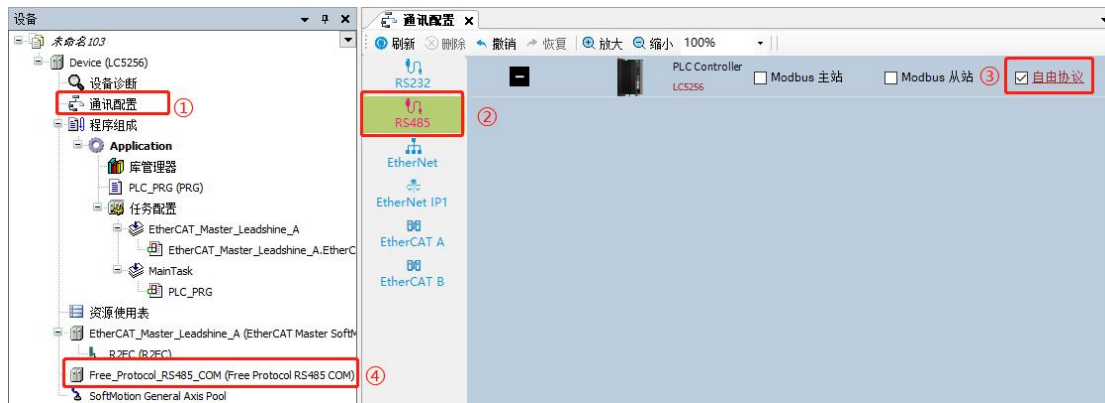


图 12.31 485 总线使能自由协议

大型 PLC 自由协议可采用 CAA SerialCom 库函数实现,如图 12.32 和 12.33 所示。

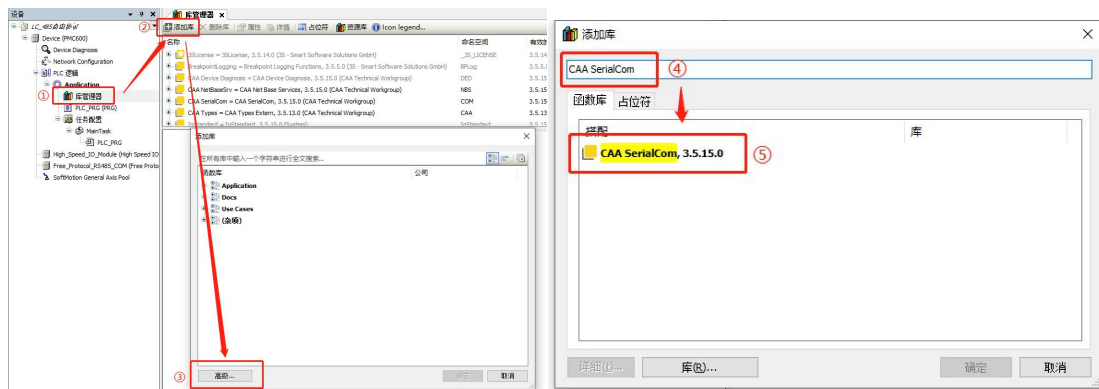


图 12.32 添加 CAA SerialCom 库函数

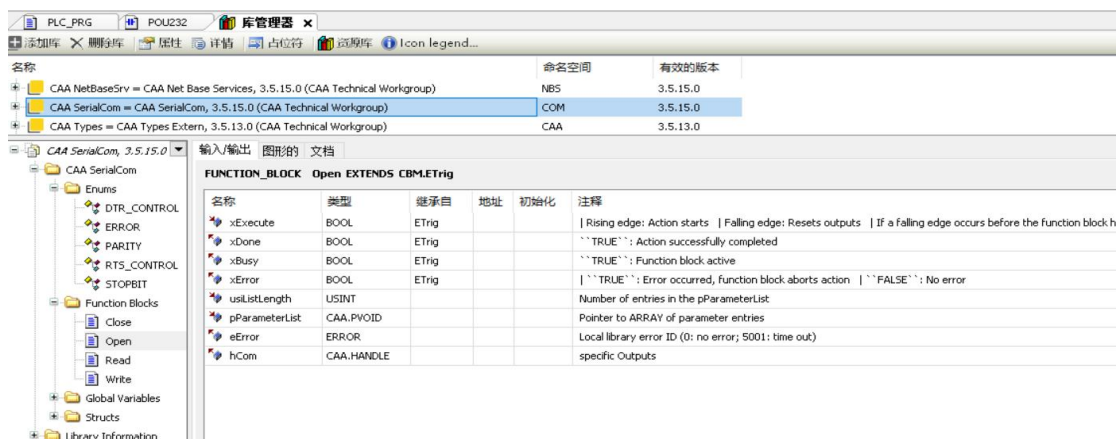


图 12.33 CAA SerialCom 库函数

自由协议是选择 232 还是 RS485 通讯链路,通过 COM 口区分,如表 12.24 所示。

表 12.24 通讯链路选择表

产品型号	232 通讯口	485 通讯口
LC2000S\LC5000 系列	COM2 (udiPort=2)	COM1 (udiPort=1)

2) 485 总线自由协议通讯案例与例程

本案例采用 LC 系列 PLC 与雷赛智能 SMC304 运动控制器进行 485 总线自由协议通讯，接收 SMC304 运动控制器发送的信号，通讯接线方法如图 12.34 所示。

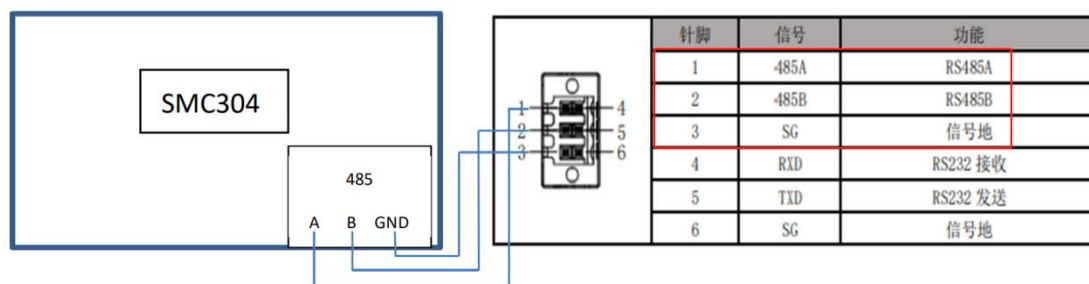


图 12.34 通讯接线图

PLC 485 总线自由协议通讯例程代码如下：

(一) 变量定义：

```

PROGRAM PLC_PRG
VAR
    aParamsB1 : ARRAY [1..7] OF COM.PARAMETER := [
        (udiParameterId := COM.CAA_Parameter_Constants.udiPort, udiValue := 1),
            //COM 口,对于 LC 产品, 1 为 485, 2 是 232
        (udiParameterId := COM.CAA_Parameter_Constants.udiBaudrate, udiValue := 9600),
            //波特率
        (udiParameterId := COM.CAA_Parameter_Constants.udiParity,
            udiValue := INT_TO_UDINT(COM.PARITY.NONE)),           //无校验位
        (udiParameterId := COM.CAA_Parameter_Constants.udiStopBits,
            udiValue := INT_TO_UDINT(COM.STOPBIT.ONESTOPBIT)),    //停止位为 1
        (udiParameterId := COM.CAA_Parameter_Constants.udiTimeout, udiValue := 0),
            //连接超时
    ]

```

```
(udiParameterId := COM.CAA_Parameter_Constants.udiByteSize, udiValue := 8),  
    //每个字节 8 位  
  
(udiParameterId := COM.CAA_Parameter_Constants.udiBinary, udiValue := 0)];  
  
fbComOpen    : COM.Open; //COM 口开启函数实例化  
fbComWrite   : COM.Write; //COM 口写数据函数实例化  
fbComRead    : COM.Read; //COM 口读数据函数实例化  
fbComClose   : COM.Close; //COM 口关闭函数实例化  
  
xCloseCom: BOOL;           //关闭串口  
xRead: BOOL;               //数据读取  
xSend: BOOL;               //数据发送  
xComConnect: BOOL;        //串口连接  
xOpenCom: BOOL:=TRUE; //打开串口  
xError: BOOL; //错误状态  
  
hCom: COM.CAA.HANDLE;           //串口连接句柄  
dataBuf: ARRAY [0..127] OF BYTE; //写数据区  
dataBuf2: ARRAY[0..127] OF BYTE; //读数据区  
  
Timer1:TON;           //定时器  
Timer1_Start:BOOL;    //定时器启用  
data:STRING;           //读取的字节转换为字符串  
  
END_VAR
```

(二)程序代码:

```
    //打开串口  
  
fbComOpen(xExecute := xOpenCom, usiListLength :=  
SIZEOF(aParamsB1)/SIZEOF(COM.PARAMETER), pParameterList :=  
ADR(aParamsB1),xError => xError, hCom => hCom);  
  
IF hCom <> 0 THEN //判断连接状态  
    xComConnect := TRUE; //连接状态
```

```
//写数据

fbComWrite(xExecute:= xSend, xAbort:= , udiTimeOut:= ,hCom:=hCom ,
pBuffer:= ADR(dataBuf) , szSize:=100 , eError=> );

//读数据

fbComRead(xExecute:=xRead , xAbort:= , udiTimeOut:= , hCom:=hCom ,
pBuffer := ADR(dataBuf2) ,szBuffer := 100 , eError=> );

ELSE

    xComConnect :=FALSE;//连接状态

END_IF

IF SIZEOF(dataBuf2)>0 THEN           //将读取的字节转换为字符串

    SysMem.SysMemCpy(ADR(data),ADR(dataBuf2),8);

END_IF

Timer1(IN:=Timer1_Start=FALSE , PT:=T#0.5S , Q=> , ET=> );

//间隔 1 秒触发一次发送

IF Timer1.Q THEN

    xRead:=FALSE;

    xSend:=TRUE;

    Timer1_Start:=TRUE;

ELSE

    xRead:=TRUE;

    xSend:=FALSE;

    Timer1_Start:=FALSE;

END_IF

//关闭串口

fbComClose(xExecute:= xCloseCom, hCom:= hCom, xError => );
```

程序运行结果如图 12.35 所示。

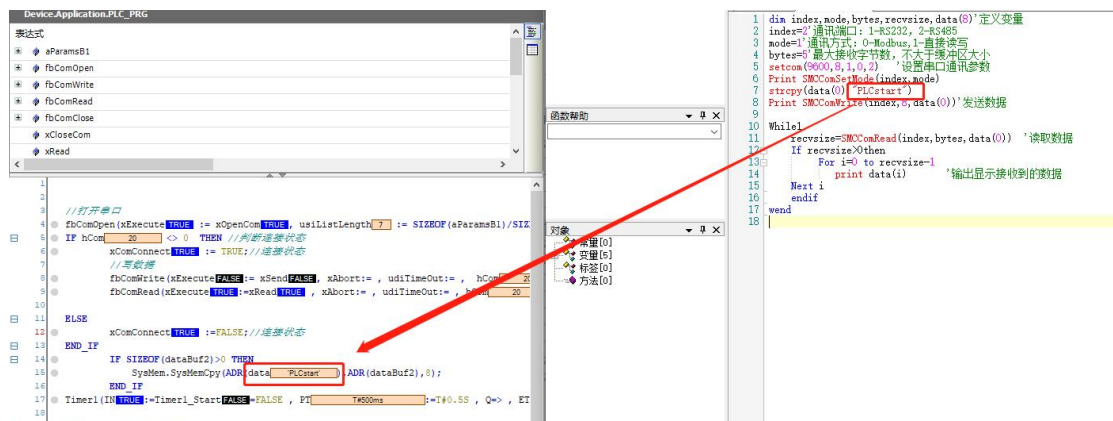


图 12.35 程序运行结果

12.5.2. PMC600 专用自由协议通讯

对于 PMC600 采用以下 PMC_Communication 库实现，如图 12.36 中的 RS232 无协议指令和 RS485 无协议指令：



图 12.36 PMC_Communication 库

详细使用方法见 PMC600 系列例程：PMC610 例程\4.通讯功能\RS485 通讯 自由协议

12.6. RS232 Modbus RTU 主站通讯

采用 RS232 接口进行 Modbus 协议的通讯,除硬件接线方法不同外,其他与 RS485 接口进行 Modbus 协议通讯的方法基本相同。

12.6.1. LC 系列 PLC RS232 RTU 主站的设置

1) 使能 PLC 作为 Modbus RTU 主站

鼠标双击网络组态中的“通讯配置”,如图 12.37 所示,单击 PLC 设备,弹出所支持的通信协议的使能窗口。勾选“Modbus 主站”,使能后,变为红色。Modbus 主站使能后,左侧树中,出现了对应的通信配置,如图 12.37 中④处所示“ModbusCOM_Master_0”。



图 12.37 使能 PLC 作为 RS232 的 Modbus RTU 主站

2) 添加 Modbus RTU 从站设备

使能 PLC 作为 Modbus RTU 主站后,如图 12.38 所示,从右侧“网络设备”COM R232 中拖入或双击“ModbusCOM Slave”从站设备,拖到蓝色的总线“Modbus_Master_COM0”下面,从站设备添加后,左侧树中,出现了对应的通信配置,如图 12.38 中③处所示“Modbus_slave_rs232”。如果有多个从站,可重复以上步骤添加多个从站设备。

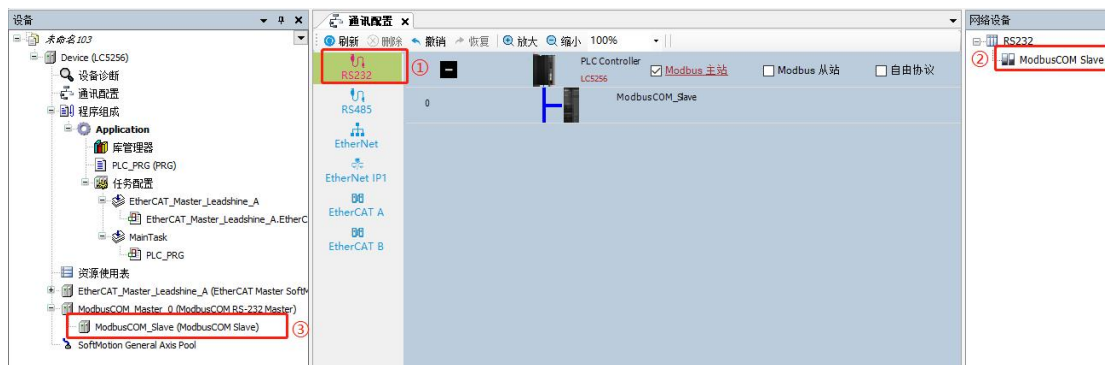


图 12.38 添加 RS232 从站设备

3) 其它相关参数配置

其它相关参数配置与 RS485 通讯 LC 系列 PLC 相同，请参见 12.1、12.2 节。

12.6.2. MC 系列 PLC RS232 RTU 主站的设置

1) 使能 PLC 作为 Modbus RTU 主站

双击网络组态中的“通讯配置”，如图 12.39 所示，单击 PLC 设备，弹出所支持的通信协议的使能窗口。勾选“Modbus 主站”，使能后，变为红色。Modbus 主站使能后，左侧树中，出现了对应的通信配置，如图 12.39 中④处所示“Modbus_Master_RS232_COM”。

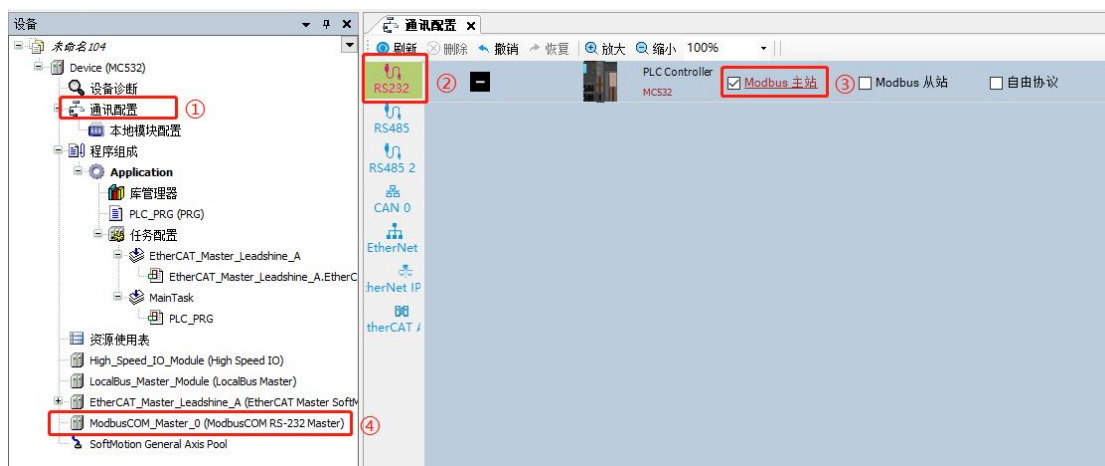


图 12.39 使能 PLC 作为 RS232 Modbus RTU 主站

2) 其它相关参数配置

其它相关参数配置与 RS485 通讯 MC 系列 PLC 相同，请参见 12.2.2 节。

12.7. RS232 Modbus RTU 从站通讯

12.7.1. LC 系列 PLC RS232 RTU 从站的设置

1) 使能 PLC 作为 Modbus RTU 从站

鼠标双击网络组态中的“通讯配置”，如图 12.40 所示，单击 PLC 设备，弹出所支持的通信协议的使能窗口。勾选“Modbus 从站”，使能后，变为红色。Modbus 从站使能后，左侧树中，出现了对应的通信配置，如图 12.40 中④处所示“Modbus_Slave_RS232_COM”。



图 12.40 使能 PLC 作为 RS232 Modbus RTU 从站

2) Modbus RTU 从站配置

双击设备树中的从站设备“Modbus_Slave_RS232_COM”，打开 Modbus 从站配置窗口，如图 12.41 所示。

Modbus 从站配置相关参数与配置示例，请参见表 12.17、表 12.18。

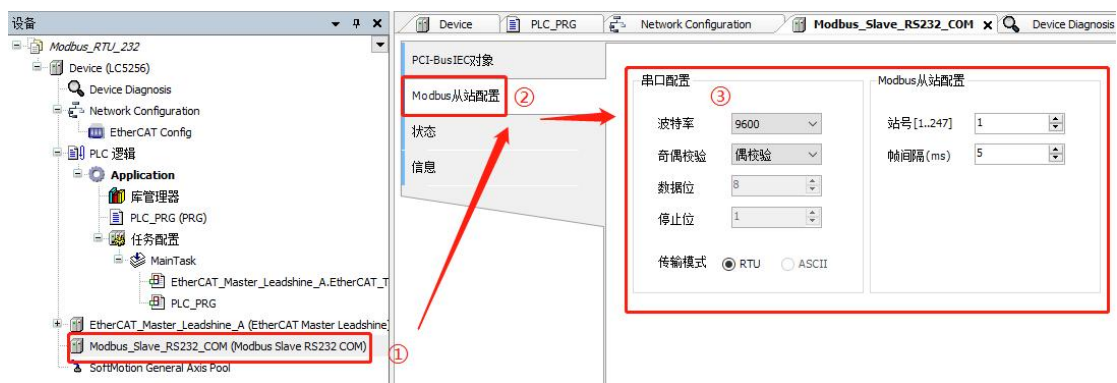


图 12.41 Modbus RTU 从站配置

3) 其它相关参数配置

其它相关参数配置与 LC 系列 PLC RS485 通讯相同，请参见 12.3 节。

12.7.2. MC 系列 PLC RS232 RTU 从站的设置

MC 系列 PLC 从站使能、配置与 LC 系列的相同。

另外，MC 系列 PLC 可以使用 PMC_Communication 库，使用方法见 12.3.2 节。

12.8. RS232 自由协议通讯

1) RS232 总线使能自由协议

鼠标双击网络组态中的“通讯配置”，如图 12.42 所示，单击 PLC 设备，弹出所支持的通信协议的使能窗口。勾选“自由协议”，使能后，变为红色。自由协议使能后，左侧树中，出现了对应的通信配置，如图 12.42，④处所示“Free_Protocol_RS232_COM”。

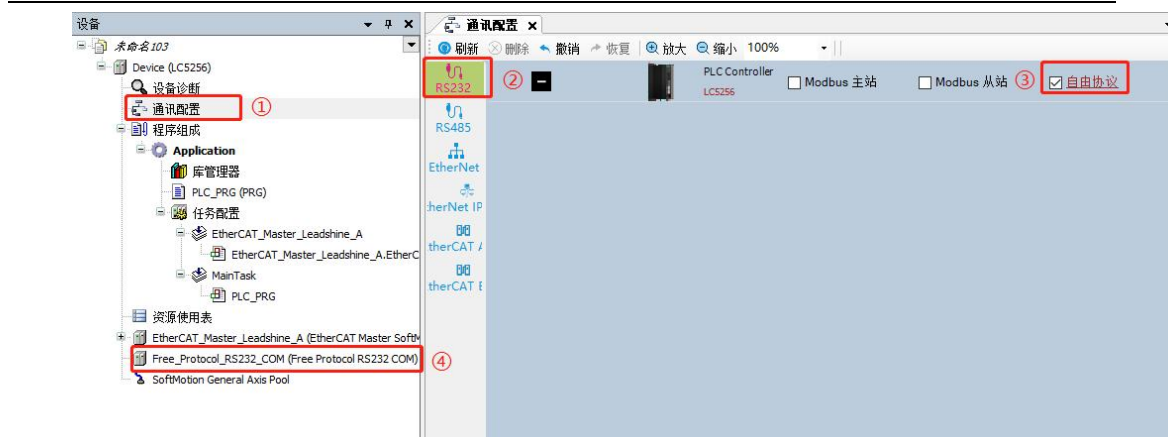


图 12.42 RS232 总线使能自由协议

2) RS232 总线自由协议通讯案例与例程

本案例采用了一款带有 RS232 接口的 LCD 显示屏与 LC5000 系列 PLC 进行 RS232 总线自由协议通讯，向显示屏发送特定的指令及相关数据（ASCII 码），显示屏显示设备当前坐标位置。

RS232 通讯接线如图 12.43 所示。

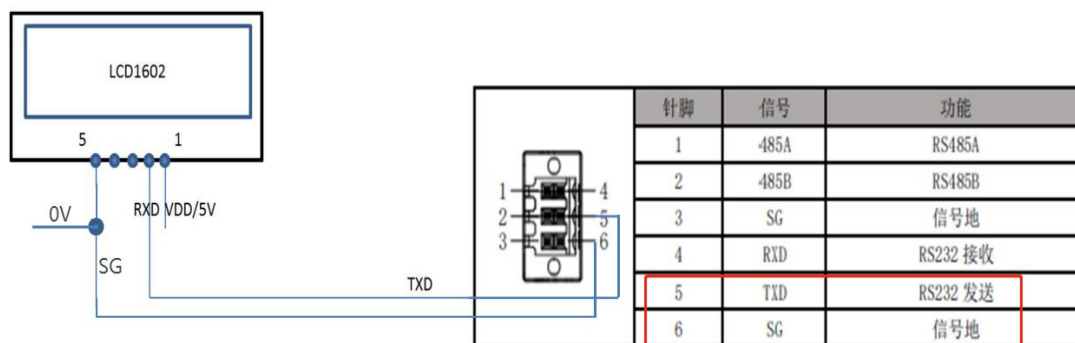


图 12.43 RS232 通讯接线图

PLC 的 RS232 总线自由协议通讯例程如下。

(一) 变量定义：

```
PROGRAM PLC_PRG
```

```
VAR
```

```
aParamsB1 : ARRAY [1..7] OF COM.PARAMETER := [
```

```
(udiParameterId := COM.CAA_Parameter_Constants.udiPort,udiValue:=2),  
//COM 口,对于 LC 产品, 1 为 485, 2 是 232  
  
(udiParameterId:= COM.CAA_Parameter_Constants.udiBaudrate,udiValue := 9600),  
//波特率  
  
  (udiParameterId := COM.CAA_Parameter_Constants.udiParity,  
udiValue := INT_TO_UDINT(COM.PARITY.NONE)),//无校验位  
  
  (udiParameterId := COM.CAA_Parameter_Constants.udiStopBits,  
udiValue := INT_TO_UDINT(COM.STOPBIT.ONESTOPBIT)),//停止位为 1  
  
(udiParameterId := COM.CAA_Parameter_Constants.udiTimeout, udiValue := 0),  
//连接超时  
  
(udiParameterId := COM.CAA_Parameter_Constants.udiByteSize, udiValue := 8),  
//每个字节 8 位  
  
(udiParameterId := COM.CAA_Parameter_Constants.udiBinary, udiValue := 0)];  
  
fbComOpen : COM.Open;  //COM 口开启函数实例化  
fbComWrite: COM.Write; //COM 口写数据函数实例化  
fbComClose: COM.Close; //COM 口关闭函数实例化  
  
xCloseCom: BOOL;      //关闭串口  
  
xSend: BOOL;          //数据发送  
  
xComConnect: BOOL;    //串口连接  
  
xOpenCom: BOOL:=TRUE; //打开串口  
  
xError: BOOL;         //错误状态  
  
hCom: COM.CAA.HANDLE; //串口连接句柄  
  
dataBuf:ARRAY [0..51] OF BYTE:=[      //LCD1602 显示数据数组  
16#AA,16#20,0,0,  16#AA,16#25,88,58, 32,32,32,32,32,32,16#0D, //X  
16#AA,16#20,0,9,  16#AA,16#25,90 ,58,32,32,32,32,32,16#0D,    //Z  
16#AA,16#20,1,0,  16#AA,16#25,89,58, 32,32, 32,32,32,32,32,    //Y  
82,117,110,105,110,103,32,16#0D];  
  
Timer1:TON;          //定时器
```

```
Timer1_Start:BOOL;//定时器启用  
PosX:STRING;      //X 坐标数据  
PosY:STRING;      //Y 坐标数据  
PosZ:STRING;      //Z 坐标数据  
X_ReadActualPosition:MC_ReadActualPosition;//读取 X 当前坐标  
Y_ReadActualPosition:MC_ReadActualPosition;//读取 Y 当前坐标  
Z_ReadActualPosition:MC_ReadActualPosition;//读取 Z 当前坐标  
END_VAR
```

(二)程序代码:

```
X_ReadActualPosition(//获取 X 位置  
    Axis:= X, Enable:= TRUE, Valid=> , Busy=> , Error=> ,  
    ErrorID=> , Position=> );  
Y_ReadActualPosition(//获取 Y 位置  
    Axis:=Y , Enable:=TRUE , Valid=> , Busy=> ,  
    Error=> , ErrorID=> , Position=> );  
Z_ReadActualPosition(//获取 Z 位置  
    Axis:= Z, Enable:=TRUE , Valid=> , Busy=> ,  
    Error=> , ErrorID=> , Position=> );  
  
//坐标转换  
PosX:=LREAL_TO_STRING(X_ReadActualPosition.Position);  
SysMem.SysMemCpy(ADR(dataBuf[8]),ADR(PosX),5);  
PosY:=LREAL_TO_STRING(Y_ReadActualPosition.Position);  
SysMem.SysMemCpy(ADR(dataBuf[37]),ADR(PosY),5);  
PosZ:=LREAL_TO_STRING(Z_ReadActualPosition.Position);  
SysMem.SysMemCpy(ADR(dataBuf[23]),ADR(PosZ),5);  
  
//打开串口  
fbComOpen(xExecute := xOpenCom, usiListLength :=
```

```
    SIZEOF(aParamsB1)/SIZEOF(COM.PARAMETER), pParameterList :=  
ADR(aParamsB1), xError => xError, hCom => hCom);  
IF hCom <> 0 THEN //判断连接状态  
    xComConnect := TRUE; //连接状态  
    //写数据  
    fbComWrite(xExecute:= xSend, xAbort:= , udiTimeOut:= , hCom:=hCom ,  
pBuffer:= ADR(dataBuf) , szSize:=52 , eError=> );  
ELSE  
    xComConnect := FALSE; //连接状态  
END_IF  
//间隔 1 秒触发一次发送  
Timer1(IN:=Timer1_Start=FALSE , PT:=T#1S , Q=> , ET=> );  
IF Timer1.Q THEN  
    xSend:=TRUE;  
    Timer1_Start:=TRUE;  
ELSE  
    xSend:=FALSE;  
    Timer1_Start:=FALSE;  
END_IF  
//关闭串口  
fbComClose(xExecute:= xCloseCom, hCom:= hCom, xError => );
```

PLC 运行程序的结果如图 12.44 所示。



图 12.44 LCD 显示屏显示结果

3) PMC600 产品专用自由协议通讯

对于 PMC600 可采用如图 12.45 所示的 PMC_Communication 库中的 RS232 无协议指令。



图 12.45 PMC_Communication 库

详细使用方法见 PMC600 系列例程: PMC610 例程\4.通讯功能\RS232 通讯 自由协议

第十三章 以太网通讯

13.1. Modbus TCP 通讯

Modbus TCP 通讯与 Modbus RTU 通信链路不同，采用以太网链路，但 Modbus 应用层协议格式和使用方法基本相同。

13.1.1. Modbus TCP 主站通讯

PLC 作 Modbus TCP 主站时，为 Modbus TCP 客户端（发起通讯的一方）。

本节介绍 PLC 作为 Modbus TCP 主站，触摸屏、变频器等设备作为 Modbus TCP 从站的通讯功能使用方法。

1) 使能 PLC 作为 Modbus TCP 主站

鼠标双击网络组态中的“通讯配置”，如图 13.1 所示，单击 PLC 设备，弹出所支持的通信协议的使能窗口。勾选“ModbusTCP 主站”，使能后，变为红色。ModbusTCP 主站使能后，左侧树中，出现了对应的通信配置，如图 13.1 中④处所示“ModbusTCP_Master_Device”。



图 13.1 使能 PLC 作为 Modbus TCP 主站

2) 添加 Modbus TCP 从站设备

使能 PLC 作为 Modbus TCP 主站后，如图 13.2 所示，从右侧“网络设备”Ethernet 口中拖入或双击“ModbusTCP SubSlave”从站设备，拖到蓝色的总线

“ModbusTCP_Master_Device”下面，从站设备添加后，左侧树中，出现了对应的通信配置，如图 13.2 中④处所示“ModbusTCP_SubSlave”。如果有多个从站，可重复以上步骤添加多个从站设备。

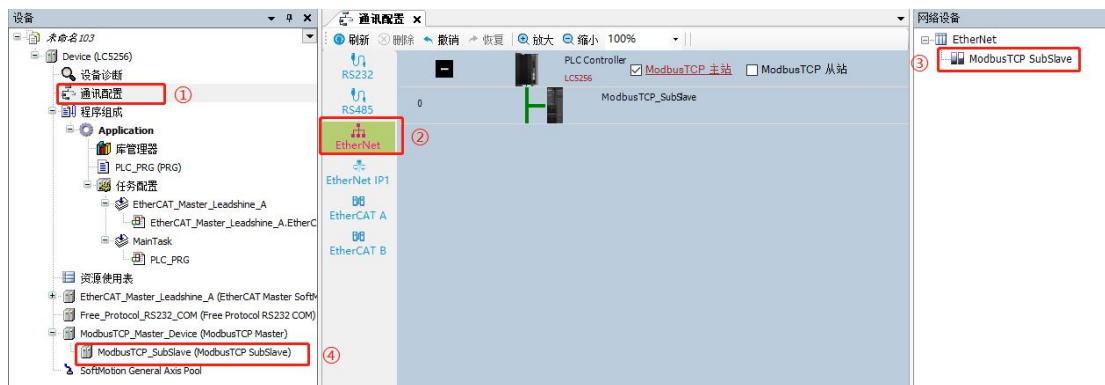


图 13.2 添加从站设备

3) Modbus TCP 主站配置

双击设备树中的主站设备“ModbusTCP_Master_Device”，打开 ModbusTCP 主站配置窗口，如图 13.3 所示。

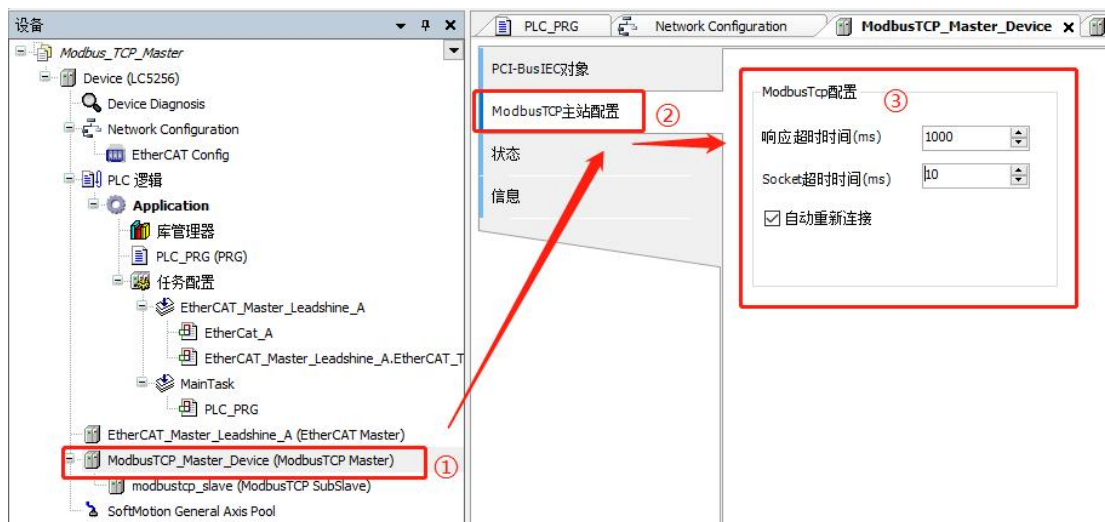


图 13.3 Modbus 主站配置

4) Modbus TCP 从站设备配置

a) Modbus 从站设置

双击设备树中的主站设备“Modbustcp_slave”，打开 ModbusTCP 从站配置窗口，

如图 13.4 所示。ModbusTCP 从站配置参数，如表 13.1 所示。

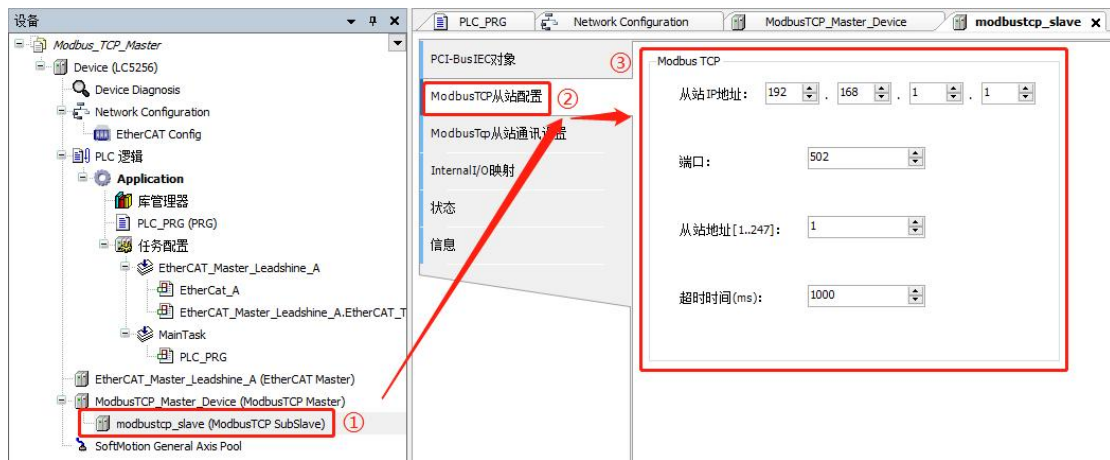


图 13.4 ModbusTCP 从站配置

表 13.1 Modbus TCP 从站配置参数

配置项	功能
从站 IP 地址	主站连接 Modbus TCP 从站的 IP 地址
端口	主站连接 Modbus TCP 从站的 TCP 端口号
从站地址	主站连接 Modbus TCP 从站的协议站地址
超时时间	启动超时功能并指定一个用毫秒表示的超时时长

b) 其它配置参数与 Modbus RTU 相同

Modbus TCP 主站通讯功能的其他配置与 Modbus RTU 相同，可参见本手册“Modbus RTU 主站通讯”章节，如添加或删除通道 Channel、编辑每个通道的配置、配置表的导入导出、Modbus 从站 Internal I/O 映射等。其配置界面也基本相同，如图 13.5 所示。

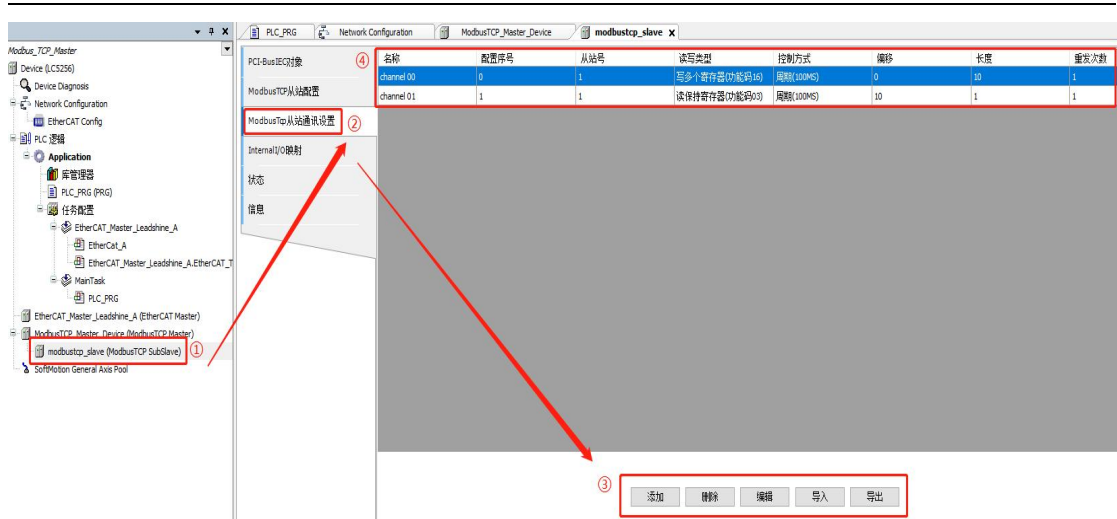


图 13.5 ModbusTCP 从站通讯设置

5) Modbus TCP 常见故障与错误码

Modbus TCP 主站连接 Modbus TCP 从站时发生的主要故障如下：

- Modbus TCP 主站连接 Modbus TCP 从站配置 IP 不正确，导致主站与从站通信无法建立。
- Modbus TCP 主站访问 Modbus TCP 从站非法地址，返回错误应答。
- Modbus TCP 主站操作 Modbus TCP 从站写寄存器，但是 Modbus TCP 从站该寄存器只支持读不支持写操作，Modbus TCP 主站会收到 Modbus TCP 从站返回的出错应答。

Modbus TCP 错误码（40001~40017）与本手册介绍 Modbus RTU 常见故障与错误码”相同，错误码定义见表 12.22。

d) 错误响应帧格式

事务元标识符+协议标识符+长度+从机地址+（命令码+0x80）+错误码+CRC 校验。

本错误帧适合所有的操作命令帧，如表 13.2 所示。

表 13.2 错误响应帧格式

序号	数据（字节）意义	字节数量	说明
----	----------	------	----

1	事务元标识符	2 个字节	MODBUS 请求 / 响应事务处理的识别码
2	协议标识符	2 个字节	0=MODBUS 协议
3	长度	2 个字节	以下字节的数量
4	从机地址	1 个字节	取值 1~247
5	命令码 +0x80	1 个字节	错误命令码
6	错误码	1 个字节	1~4

注：PMC600 不支持该错误码

6) Modbus TCP 通讯帧格式

a) 读线圈，功能码 0x01，可以读取 Q 区变量；

请求帧格式：

事务元标识符+协议标识符+长度+从机地址+0x01+线圈起始地址+线圈数，
如表 13.3 所示。

响应帧格式：

事务元标识符+协议标识符+长度从机地址+0x01+字节数+线圈状态，如表 13.4 所示。

表 13.3 功能码 0x01 请求帧详解

序号	数据 (字节) 意义	字节数量	说明
1	事务元标识符	2 个字节	MODBUS 请求 / 响应事务处理的识别码
2	协议标识符	2 个字节	0=MODBUS 协议
3	长度	2 个字节	以下字节的数量
4	从机地址	1 个字节	取值 1~247
5	0x01	1 个字节	读线圈
6	线圈起始地址	2 个字节	高位在前，低位在后，见线圈编址
7	线圈数量 N	2 个字节	高位在前，低位在后

表 13.4 功能码 0x01 响应帧详解

序号	数据 (字节) 意义	字节数量	说明
1	事务元标识符	2 个字节	MODBUS 请求 / 响应事务处理的识别码
2	协议标识符	2 个字节	0=MODBUS 协议
3	长度	2 个字节	以下字节的数量
4	从机地址	1 个字节	取值 1~247
5	0x01 _h	1 个字节	读线圈
6	字节数	1 个字节	值: $[(N+7) / 8]$
7	线圈状态	$[(N+7) / 8]$ 个字节	每 8 个线圈合为一个字节, 最后一个若不足 8 位, 未定义部分填 0。前 8 个线圈在第一个字节, 最地址最小的线圈在最低位。依次类推

b) 读寄存器, 功能码 0x03, 可以读取 M 区变量;

请求帧格式:

事务元标识符+协议标识符+长度+从机地址+0x03+寄存器起始地址+寄存器数量, 如表 13.5 所示。

响应帧格式:

事务元标识符+协议标识符+长度+从机地址+0x03+字节数+寄存器值, 如表 13.6 所示。

表 13.5 功能码 0x03 请求帧详解

序号	数据 (字节) 意义	字节数量	说明
1	事务元标识符	2 个字节	MODBUS 请求 / 响应事务处理的识别码
2	协议标识符	2 个字节	0=MODBUS 协议
3	长度	2 个字节	以下字节的数量
4	从机地址	1 个字节	取值 1~247
5	0x03 _h	1 个字节	读寄存器
6	寄存器起始地址	2 个字节	高位在前, 低位在后, 见寄存器编址
7	寄存器数量 N	2 个字节	高位在前, 低位在后

表 13.6 功能码 0x03 响应帧详解

序号	数据 (字节) 意义	字节数量	说明
1	事务元标识符	2 个字节	MODBUS 请求 / 响应事务处理的识别码
2	协议标识符	2 个字节	0=MODBUS 协议
3	长度	2 个字节	以下字节的数量
4	从机地址	1 个字节	取值 1~247
5	0x03	1 个字节	读寄存器
6	字节数	1 个字节	值: $N*2$
7	寄存器值	$N*2$ 个字节	每两字节表示一个寄存器值, 高位在前低位在后。寄存器地址小的排在前面

c) 写单个线圈，功能码 0x05，可以写 Q 区变量

请求帧格式：

事务元标识符+协议标识符+长度+从机地址+0x05+线圈地址+线圈状态，如表 13.7 所示。

响应帧格式：

事务元标识符+协议标识符+长度+从机地址+0x05+线圈地址+线圈状态，如表 13.8 所示。

表 13.7 功能码 0x05 请求帧详解

序号	数据(字节)意义	字节数量	说明
1	事务元标识符	2 个字节	MODBUS 请求 / 响应事务处理的识别码
2	协议标识符	2 个字节	0=MODBUS 协议
3	长度	2 个字节	以下字节的数量
4	从机地址	1 个字节	取值 1~247
5	0x05	1 个字节	写单线圈
6	线圈地址	2 个字节	高位在前，低位在后，见线圈编址
7	线圈状态	2 个字节	低位在前，高位在后，非 0 即为有效

表 13.8 功能码 0x05 响应帧详解

序号	数据(字节)意义	字节数量	说明
1	事务元标识符	2 个字节	MODBUS 请求 / 响应事务处理的识别码
2	协议标识符	2 个字节	0=MODBUS 协议
3	长度	2 个字节	以下字节的数量
4	从机地址	1 个字节	取值 1~247
5	0x05	1 个字节	写单线圈
6	线圈地址	2 个字节	高位在前，低位在后，见线圈编址
7	线圈状态	2 个字节	低位在前，高位在后，非 0 即为有效

d) 写单个寄存器，功能码 0x06，可以写 M 区变量。

请求帧格式：

事务元标识符+协议标识符+长度+从机地址+0x06+寄存器地址+寄存器值，如表 13.9 所示。

响应帧格式：

事务元标识符+协议标识符+长度+从机地址+0x06+寄存器地址+寄存器值，

如表 13.10 所示。

表 13.9 功能码 0x06 请求帧详解

序号	数据(字节)意义	字节数量	说明
1	事务元标识符	2 个字节	MODBUS 请求 / 响应事务处理的识别码
2	协议标识符	2 个字节	0=MODBUS 协议
3	长度	2 个字节	以下字节的数量
4	从机地址	1 个字节	取值 1~247
5	0x06	1 个字节	写单寄存器
6	寄存器地址	2 个字节	高位在前, 低位在后, 见寄存器值编址
7	寄存器值	2 个字节	高位在前, 低位在后。非 0 即为有效

表 13.10 功能码 0x06 响应帧详解

序号	数据(字节)意义	字节数量	说明
1	事务元标识符	2 个字节	MODBUS 请求 / 响应事务处理的识别码
2	协议标识符	2 个字节	0=MODBUS 协议
3	长度	2 个字节	以下字节的数量
4	从机地址	1 个字节	取值 1~247
5	0x06	1 个字节	写单寄存器
6	寄存器地址	2 个字节	高位在前, 低位在后, 见寄存器编址
7	寄存器值	2 个字节	高位在前, 低位在后。非 0 即为有效

e) 写多个线圈, 功能码 0x0f, 可以写连续的多个 Q 区变量。

请求帧格式:

事务元标识符+协议标识符+长度+从机地址+0x0f+线圈起始地址+线圈数量
+字节数+线圈状态, 如表 13.11 所示。

响应帧格式:

事务元标识符+协议标识符+长度+从机地址+0x0f+线圈起始地址+线圈数,
如表 13.12 所示。

表 13.11 功能码 0x0f 请求帧详解

序号	数据 (字节) 意义	字节数量	说明
1	事务元标识符	2 个字节	MODBUS 请求 / 响应事务处理的识别码
2	协议标识符	2 个字节	0=MODBUS 协议
3	长度	2 个字节	以下字节的数量
4	从机地址	1 个字节	取值 1~247
5	0x0f	1 个字节	写多个单线圈
6	线圈起始地址	2 个字节	高位在前，低位在后，见线圈编址
7	线圈数量 N	2 个字节	高位在前，低位在后。最大为 1968
8	字节数	1 个字节	值：值： $[(N+7)/8]$
9	线圈状态	$[(N+7)/8]$ 个字节	每 8 个线圈合为一个字节，最后一个若不足 8 位，未定义部分填 0。前 8 个线圈在第一个字节，最地址最小的线圈在最低位。依次类推

表 13.12 功能码 0x0f 响应帧详解

序号	数据 (字节) 意义	字节数量	说明
1	事务元标识符	2 个字节	MODBUS 请求 / 响应事务处理的识别码
2	协议标识符	2 个字节	0=MODBUS 协议
3	长度	2 个字节	以下字节的数量
4	从机地址	1 个字节	取值 1~247
5	0x0f	1 个字节	写多个单线圈
6	线圈起始地址	2 个字节	高位在前，低位在后，见线圈编址
7	线圈数量	2 个字节	高位在前，低位在后。

f) 写多个寄存器，功能码 0x10，可以写连续的多个 M 变量。

请求帧格式：

事务元标识符+协议标识符+长度+从机地址+0x10+寄存器起始地址+寄存器数量+字节数+寄存器值，如表 13.13 所示。

响应帧格式：

事务元标识符+协议标识符+长度+从机地址+0x10+线圈起始地址+线圈数量，如表 13.14 所示。

表 13.13 功能码 0x10 请求帧详解

序号	数据 (字节) 意义	字节数量	说明
1	事务元标识符	2 个字节	MODBUS 请求 / 响应事务处理的识别码
2	协议标识符	2 个字节	0=MODBUS 协议
3	长度	2 个字节	以下字节的数量
4	从机地址	1 个字节	取值 1~247
5	0x10	1 个字节	写多个寄存器
6	寄存器起始地址	2 个字节	高位在前，低位在后，见寄存器编址
7	寄存器数量	2 个字节	高位在前，低位在后，N
8	字节数	1 个字节	值：N*2
9	寄存器值	N*2 (N*4)	

表 13.14 功能码 0x10 响应帧详解

序号	数据 (字节) 意义	字节数量	说明
1	事务元标识符	2 个字节	MODBUS 请求 / 响应事务处理的识别码
2	协议标识符	2 个字节	0=MODBUS 协议
3	长度	2 个字节	以下字节的数量
4	从机地址	1 个字节	取值 1~247
5	0x10	1 个字节	写多个寄存器
6	寄存器起始地址	2 个字节	高位在前，低位在后，见寄存器编址
7	寄存器数量	2 个字节	高位在前，低位在后，N

7) 其它相关参数配置

其它相关参数配置与 Modbus RTU 相同，请参见 12.1、12.2 节内容。

13.1.2. Modbus TCP 从站通讯

PLC 作 Modbus TCP 从站时，为 Modbus TCP 服务端（响应通讯的一方）。

1) 使能 PLC 作为 Modbus TCP 从站

双击网络组态中的“通讯配置”，如图 13.6 所示，单击 PLC 设备，弹出所支持的通信协议的使能窗口。勾选“ModbusTCP 从站”，使能后，变为红色。ModbusTCP 从站使能后，左侧树中，出现了对应的通信配置，如图 13.6 中④处所示“ModbusTCP_Slave_Device”。

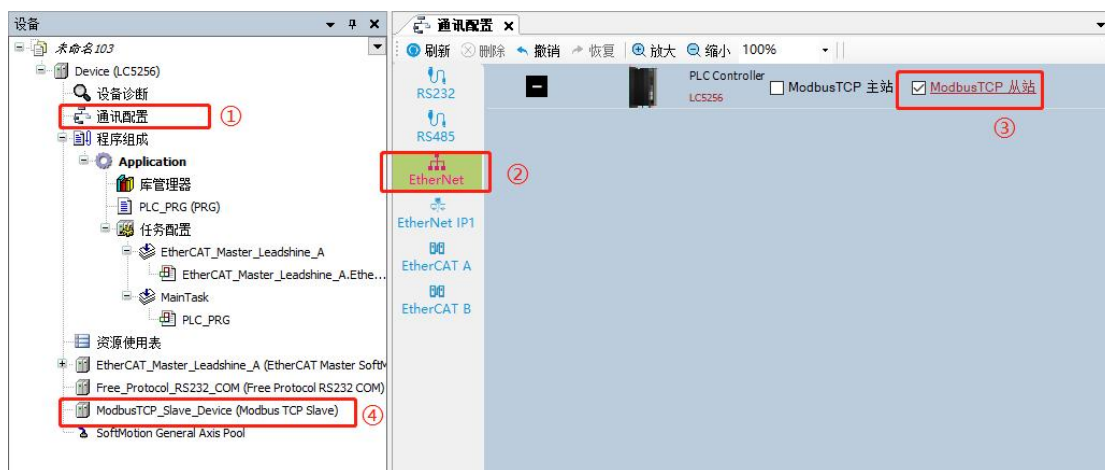


图 13.6 使能 PLC 作为 ModbusTCP 从站

2) Modbus TCP 从站配置

双击设备树中的从站设备“Modbus_Slave_Device”，打开从站参数设置窗口，如图 13.7 所示。

Modbus 从站设置相关参数说明，请参见表 13.15 所示。

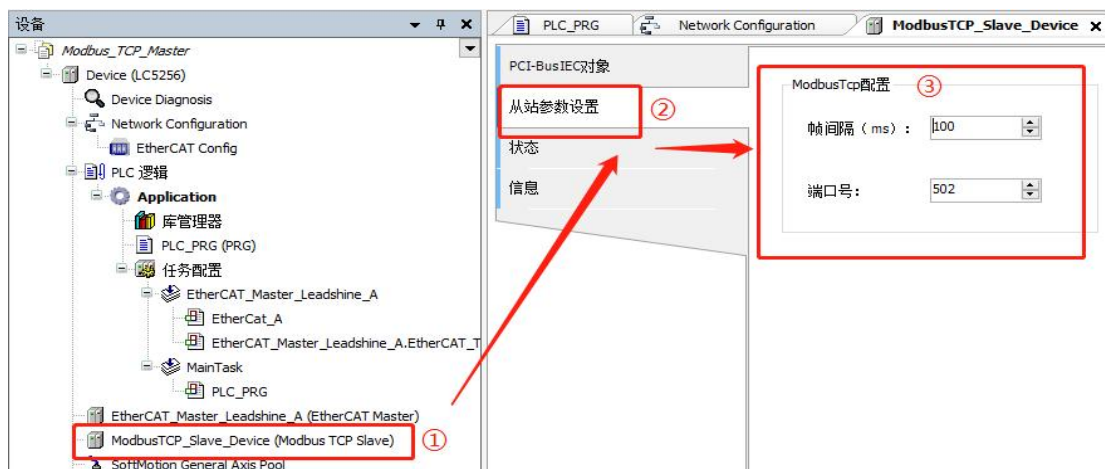


图 13.7 使能 PLC 作为 ModbusTCP 从站

表 13.15 从站设置参数说明

配置项	功能
帧间隔	Modbus TCP 从站接收通信帧后延迟时间发送回复帧。 LC 系列 PLC 此参数暂时没用到，即无延迟。

端口号	Modbus TCP 从站的 TCP 端口号
-----	------------------------

3) 其它相关参数配置

其它相关参数配置与 Modbus RTU 相同，请参见 11.3 节。

13.2. TCP 自由协议

本节介绍采用 LeadSys 软件自带库 CAA 与 TCP 库，实现 TCP 自由协议通讯的方法。

13.2.1. 库管理器添加 CAA 类型库

鼠标双击左侧树中的“库管理器”，打开库管理器，点击“添加库”，如图 13.8，选择“高级”，搜索“CAA Types Extern”，选择库，添加后如图 13.9 所示。

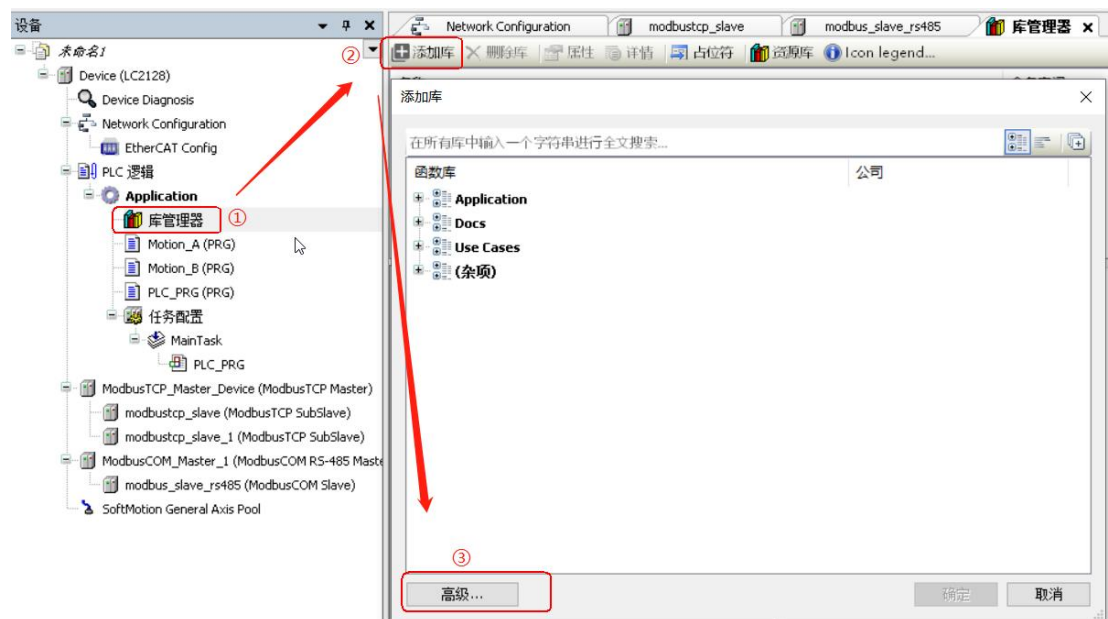


图 13.8 添加 CAA 与 TCP 库

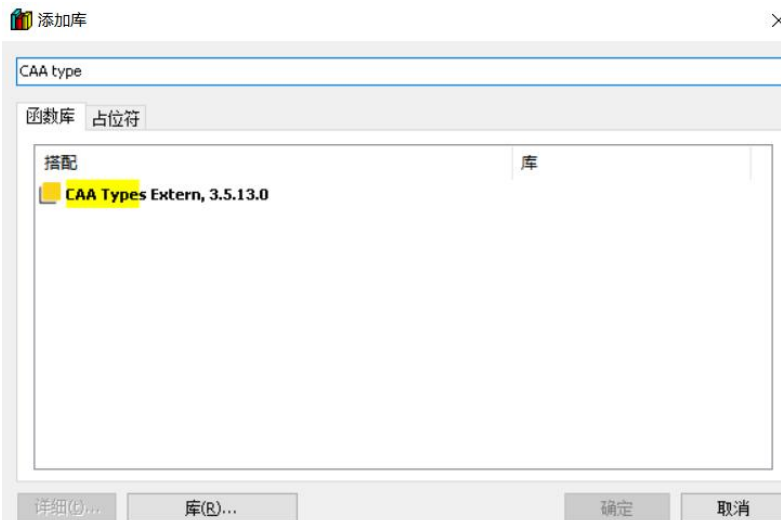


图 13.9 添加 CAA Types Extern 库

重复以上步骤，添加“CAA Net Base Services”库，安装库后，在库管理器中可看到库函数，如图 13.10 所示。

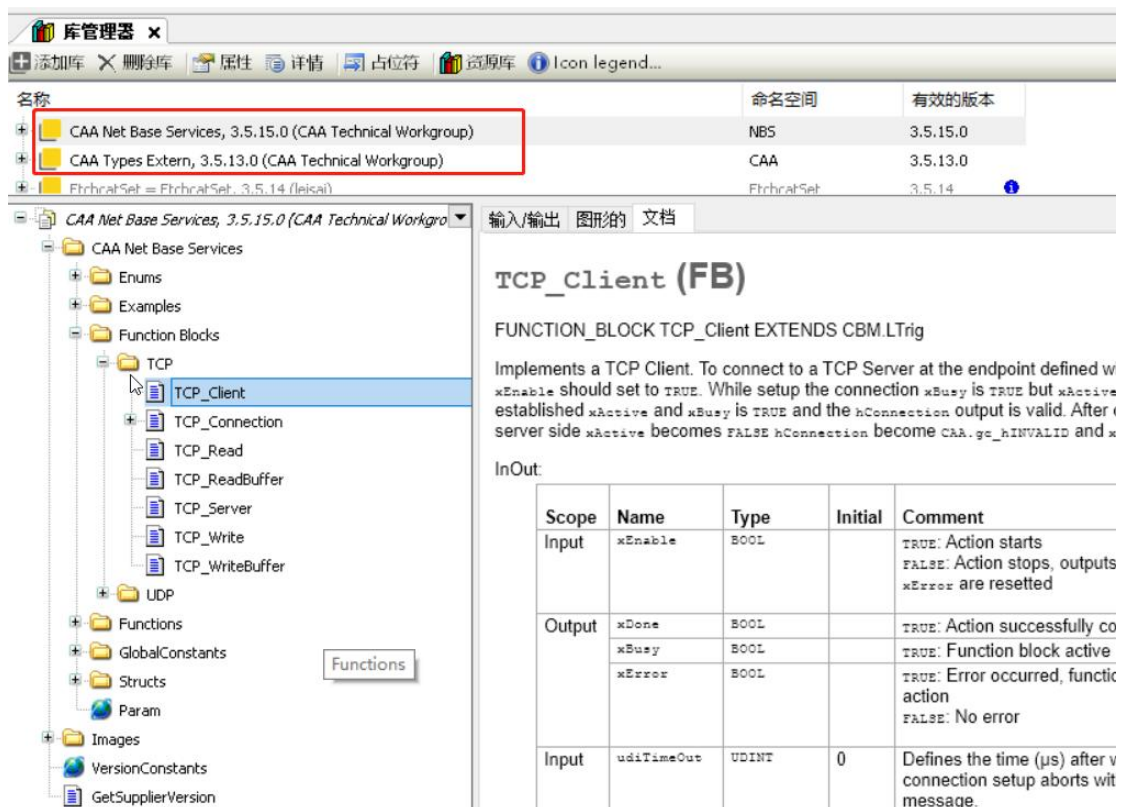


图 13.10 添加 CAA Types Extern 库与 CAA Net Base Services 库

13.2.2. 使用 TCP 自由协议指令

CAA 提供了两种读写操作方式，即 TCP_Read 和 TCP_Write。其功能与 TCP_ReadBuffer 和 TCP_WriteBuffer 一样，两种方式任选。

本文采用非缓存区的形式进行读写。

各指令说明如下。

1) 功能块 TCP_Client: 开启 tcp 客户端模式



图 13.11 功能块 TCP_Server

功能块 TCP_Client 如图 13.11 所示，通过 ipAdd 和 uiPort 中指定的 ip 和端口连接 TCP 服务器。输入 xEnable 设置为 TRUE，功能块开始执行 xBusy 置为 TRUE，但是 xActive 为 FALSE，当连接建立后，xActive 置为 TRUE，hConnect 输出有效。当服务端关闭连接后，xActive 变为 FALSE，hConnect 变为 CAA_gc_hINVALID，xDone 变为 TRUE。

功能块 TCP_Client 说明，如表 13.16 所示。

表 13.16 功能块 TCP_Client 说明

名称	数据类型	描述	默认值	详细
xEnable	BOOL	客户端连接使能	FALSE	TRUE: 客户端连接 FALSE: 断开连接
ipAddr	IP_ADDR	Ip 地址	STRUCT(sAddr := '0.0.0.0')	配置连接的服务器的 ip 地址

uiPort	UINT	端口号	0	服务器端口号
xDone	BOOL	执行成功		
xBusy	BOOL	功能块执行中		
eError	ERROR	错误		
hConnection	CAA.HANDLE	客户端句柄		客户端句柄， xActive 为 True 则有效，用于读写操作的句柄
UdiTimeOut	UDINT	超时时间	0	单位为 μ s

2) 功能块 TCP_Server: 开启 TCP 服务器模式



图 13.12 功能块 TCP_Server

功能块 TCP_Server 如图 13.12 所示，可通过 ipAddr 和 uiPort 中指定的 ip 和端口设置服务器。一个服务器可以处理多个连接。这些功能块的输出句柄 hServer 可用于通过 TCP_Connection 建立连接。

功能块 TCP_Server 说明，如表 13.17 所示。

表 13.17 功能块 TCP_Server 说明

名称	数据类型	描述	默认值	详细
xEnable	BOOL	服务器开启使能	FALSE	TRUE: 服务器开启

				FALSE: 停止, 输出信号复位
ipAddr	IP_ADDR	Ip 地址	STRUCT(sAddr : = '0.0.0.0')	配置服务器的 ip 地址
uiPort	UINT	端口号	0	服务器端口号
xDone	BOOL	执行成功		
xBusy	BOOL	功能块执行中		
eError	ERROR	错误		
hServer	CAA.HANDLE NDLE	服务器句柄		用于建立服务器句柄

3) 功能块 TCP_Connetction: 获取连接句柄

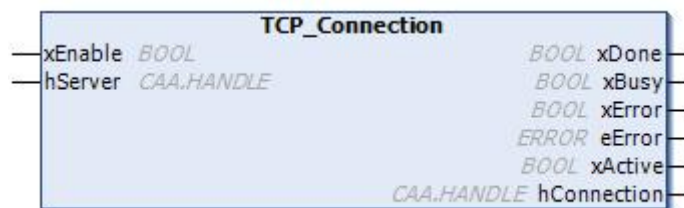


图 13.13 功能块 TCP_Connetction

功能块 TCP_Connetction 如图 13.13 所示, 根据服务器句柄建立连接来激活。当 xActive 为 TRUE, 连接的句柄就有效。如果 xActive 为 FALSE, 则连接句柄无效。hConnection 可用作功能块 TCP_Write、TCP_WriteBuffer、TCP_Read 和 TCP_ReadBuffer 的输入 hConnection。

若遇到多个可客户端连接该服务器, 可实例化多个 TCP_Connetction 来建立连接。功能块 TCP_Connetction 说明如表 13.18 所示。

表 13.18 功能块 TCP_Connetction 说明

名称	数据类型	描述	默认值	详细
xEnable	BOOL	连接使能	False	为一组 pAxisRef 结构的接口最多数量限制在 8
hServer	CAA.HANDLE	服务器句柄		
xDone	BOOL	指令完成	False	指令完成后标志位
xBusy	BOOL	任务进行中	False	指令任务中标志
xError	BOOL	错误标志	False	产生错误标志
xActive	BOOL	指令激活中	False	指令激活中
hConnection	CAA.HANDLE			跟 TCPserver 等配合生成连接句柄

注：当功能块 TCP_Connetction 与 TCP_Server 配合使用时，PLC 处于“监听状态”，如果有多个客户端与 PLC 连接，则需要配置多个连接功能块。

4) 功能块 TCP_Read: 读取数据

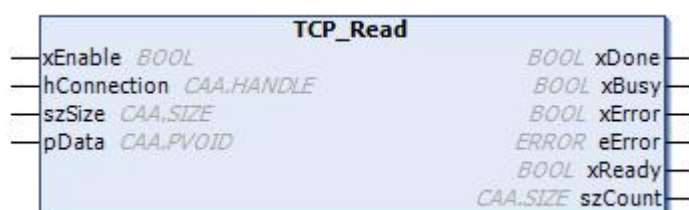


图 13.14 功能块 TCP_Read

功能块 TCP_Read 如图 13.14 所示，该功能块用于从 hConnection 中给定的先前建立的连接中读取数据。只要 xEnable 为 TRUE，连接的句柄就有效。指针 pData 指定要写入读取数据的内存区域。szSize 定义要读取的最大字节数。当数据读取成功时，xReady 设置为 TRUE，szCount 表示实际读取的字节数。

功能块 TCP_Read 说明如表 13.19 所示。

表 13.19 功能块 TCP_Read 说明

名称	数据类型	描述	默认值	详细
xEnable	BOOL	连接使能	False	TRUE: 读取开始 FALSE: 停止
hConnection	CAA.HANDLE	连接句柄		连接句柄
szSize	CAA.SIZE	数据范围		读取数据大小
pData	CAA.PVOID	指针		读取数据存储指向的区域
xDone	BOOL	完成	False	完成标志
xBusy	BOOL	任务进行中	False	指令任务中标志
xError	BOOL	错误标志	False	产生错误标志
eError	ERROR	错误		错误内容
xReady	BOOL	读到	False	完成一次读取
szCount	CAA.SIZE	字节数		实际读取的字节数

5) 功能块 TCP_ReadBuffer: 读取数据缓冲区

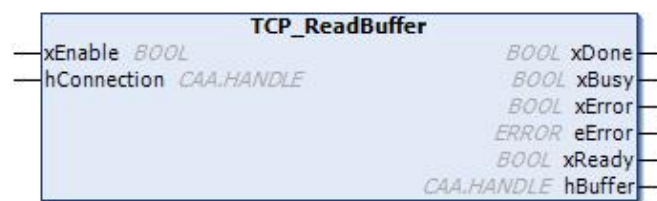


图 13.15 功能块 TCP_ReadBuffer

功能块 TCP_ReadBuffer 如图 13.15 所示，与 TCP_Read 一样，此功能块用于从 hConnection 中给出的先前建立的连接中读取数据。只要 xEnable 为 TRUE，连接的句柄就有效。该功能块返回包含数据的缓冲区的句柄。如果 xReady 为 TRUE，则数据已成功读取且缓冲区句柄有效。缓冲区被传递而不是像 TCP_Read 那样被

复制。

功能块 TCP_ReadBuffer 说明如表 13.20 所示。

表 13.20 功能块 TCP_ReadBuffer 说明

名称	数据类型	描述	默认值	详细
xEnable	BOOL	连接使能	False	
hConnection	CAA.HANDLE	连接句柄		连接句柄
xDone	BOOL	完成	False	完成标志
xBusy	BOOL	任务进行中	False	指令任务中标志
xError	BOOL	错误标志	False	产生错误标志
eError	ERROR	错误		错误内容
xReady	BOOL	读到	False	完成一次读取
hBuffer	CAA.HANDLE	缓冲区句柄		读取缓存区

6) 功能块 TCP_Write: 写入数据

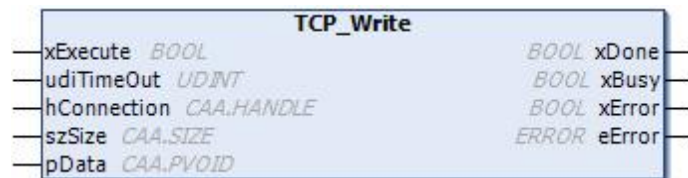


图 13.16 功能块 TCP_Write

功能块 TCP_Write 如图 13.16 所示，该功能块用于从 hConnection 中给出的先前建立的连接中读取数据。由指针 pData 指示的内存区域的内容包含要写入的数据。szSize 定义要写入的字节数。当 xDone 设置为 TRUE 时，数据已成功写入。超时时间在这个功能块上没有影响。

功能块 TCP_Write 说明如表 13.21 所示。

表 13.21 功能块 TCP_Write 说明

名称	数据类型	描述	默认值	详细
----	------	----	-----	----

xExecute	BOOL	连接使能	False	上升沿：执行 下降沿：复位输出信号
hConnection	CAA.HANDLE	连接句柄		连接句柄
xDone	BOOL	完成	False	完成标志
xBusy	BOOL	任务进行中	False	指令任务中标志
xError	BOOL	错误标志	False	产生错误标志
eError	ERROR	错误		错误内容
szSize	CAA.SIZE			
pData	CAA.PVOID	指针		写数据存储指向的区域

7) 功能块 TCP_WriteBuffer: 写入数据缓冲区



图 13.17 功能块 TCP_WriteBuffer

功能块 TCP_WriteBuffer 如图 13.17 所示，与 TCP_Write 一样，此功能块用于从 hConnection 中给出的先前建立的连接中写入数据。该功能块输入不是指向数据的指针，而是写入数据缓存区的句柄，然后缓存区被传递。要结合缓存区进行操作。缓冲区被传递而不是像 TCP_Write 那样被复制。

功能块 TCP_WriteBuffer 说明，如表 13.22 所示。

表 13.22 功能块 TCP_WriteBuffer 说明

名称	数据类型	描述	默认值	详细
xExecute	BOOL	连接使能	False	上升沿：执行

				下降沿：复位输出信号
hConnection	CAA.HANDLE	连接句柄		连接句柄
xDone	BOOL	完成	False	完成标志
xBusy	BOOL	任务进行中	False	指令任务中标志
xError	BOOL	错误标志	False	产生错误标志
eError	ERROR	错误		错误内容
hBuffer	CAA.HANDLE	缓冲区句柄		写数据到缓存区

13.2.3. TCP 自由协议例程

本节将采用 LC 系列 PLC 以“通讯连接”、“数据发送”、“数据接收”这三个功能为例，分别说明当 PLC 为客户端或服务器时，如何与 PC 机进行 TCP 自由协议通讯。

1) PLC（服务器）与上位机（客户端）的通讯例程

a) 硬件连接，如图 13.18 所示。

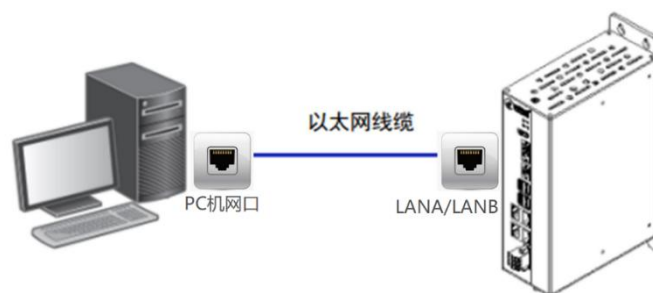


图 13.18 PC 机与 PLC 通过以太网连接

b) 设置 PC 机的 IP 地址，如图 13.19 所示。

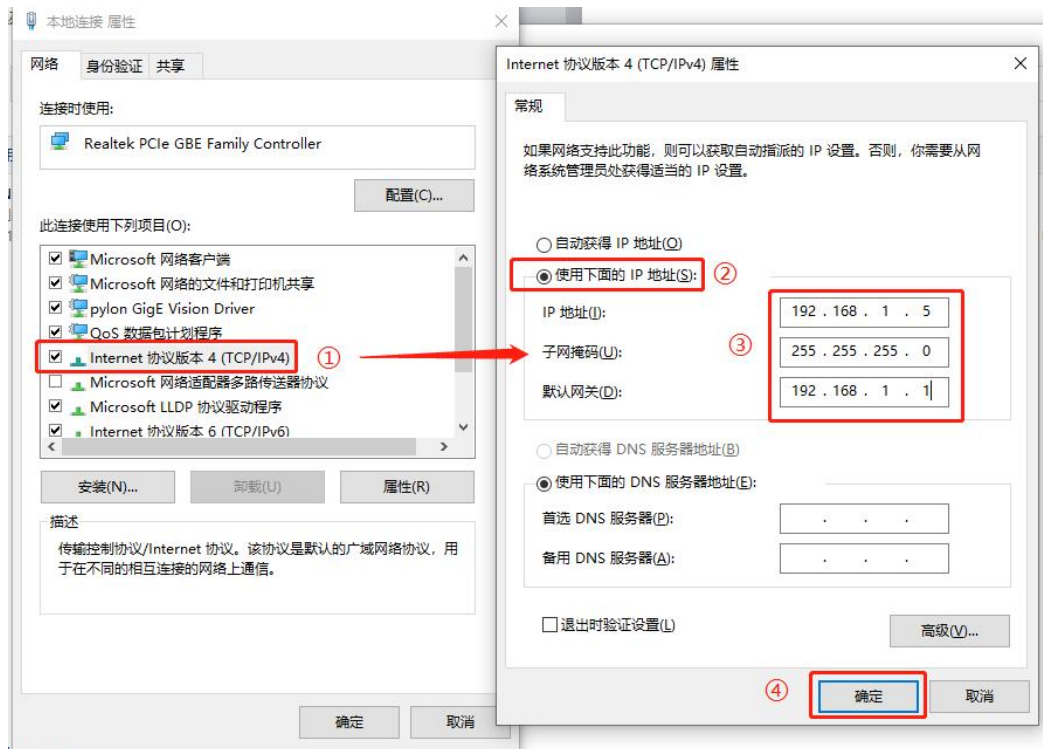


图 13.19 设置 PC 机的 IP 地址

c) PLC 程序中 IP 地址的设置。如图 13.20 所示，设置 PC 机的 IP 地址；如图 13.21 所示，设置 PLC 的 IP 地址。



图 13.20 在 PLC 程序中设置 PC 机的 IP 地址


```

PROGRAM TCP_Server_PRG
VAR
    (*****TCP_Server*****)
    tcpSvr: NBS.TCP_Server; //服务器实例化
    IP: NBS.IP_ADDR:=(sAddr:='192.168.1.3'); //PLC IP地址
    Port: UINT:=8000; //端口号
    (*****TCP_Connection*****)
    tcpConnect: NBS.TCP_Connection;
    (*****数据读写*****)
    read: NBS.TCP_Read; //读数据
    write: NBS.TCP_Write; //写数据
    recvData: ARRAY[0..255] OF BYTE; //读数据区
    sendData: ARRAY[0..255] OF BYTE; //写数据区
    (*****收发内容*****)

```

PLC (服务器)

PLC IP地址

图 13.21 在 PLC 程序中设置 PLC 的 IP 地址

d) PLC（服务器）的代码

① 变量声明

```

PROGRAM PLC_PRG                                //声明变量名称
VAR
    (*****TCP_Server*****)
    tcpSvr: NBS.TCP_Server;                    //服务器实例化
    IP: NBS.IP_ADDR:=(sAddr:='192.168.1.3');
//IP 地址(只能使用 192.168.1.3)
    Port: UINT:=8000;                          //端口号
    (*****TCP_Connection*****)
    tcpConnect: NBS.TCP_Connection;            //获取连接句柄
    (*****数据读写*****)
    read: NBS.TCP_Read;                        //读数据函数
    write: NBS.TCP_Write;                     //写数据函数
    recvData: ARRAY[0..255] OF BYTE;          //读数据区
    sendData: ARRAY[0..255] OF BYTE;          //写数据区
    (*****收发内容*****)
    strRecvData : STRING;                      //自定义接收数据（类型可改）
    strSendData : STRING;                      //自定义发送数据（类型可改）
    xSend: BOOL;                               //写数据信号

```

```
iState: INT:=0;                //程序运行状态位
ErrorID: NBS.ERROR;            //错误标志
xSendOld: BOOL;                //上一次写数据信号
xError: BOOL;                  //错误状态
ton:TON;                        //重连定时器
realReadSize: NBS.CAA.SIZE;    //读取数据大小
END_VAR
```

② 程序代码

//开启 TCP 服务器模式

```
tcpSvrer(xEnable:= , xDone=> , xBusy=> , xError=> , ipAddr:= IP,
  uiPort:= Port, eError=> , hServer=> );//获取连接句柄
tcpConnect( xEnable:= , xDone=> , xBusy=> , xError=> ,
  hServer:= tcpSvrer.hServer, eError=> , xActive=> , hConnection=> );
//读取数据
read(xEnable:= tcpConnect.xActive, xDone=> , xBusy=> , xError=> ,
  hConnection:= tcpConnect.hConnection, szSize:= SIZEOF(recvData),
  pData:= ADR(recvData), eError=> , xReady=> , szCount=> );
//写入数据
write(xExecute:= xSend, udiTimeOut:= , xDone=> , xBusy=> , xError=> ,
  hConnection:= tcpConnect.hConnection, szSize:= SIZEOF(sendData),
  pData:= ADR(sendData), eError=> );
```

CASE iState OF

0:

//建立服务器

tcpSvrer.xEnable:=TRUE;

IF tcpSvrer.hServer <> 0 THEN //服务器状态正常

iState:=10;

```
END_IF

IF tcpSrvr.xError THEN                                //建立服务器错误

    ErrorID:=tcpSrvr.eError;

    iState:=100;

END_IF

10:

//创建连接

tcpConnect.xEnable:=TRUE;                            //使能获取连接句柄

IF tcpConnect.xActive THEN                            //连接激活

    iState:=20;

    xError :=FALSE;

END_IF

IF tcpConnect.xError THEN                            //连接错误

    ErrorID:=tcpConnect.eError;

    iState:=100;

END_IF

20:

//写数据

IF xSend AND NOT xSendOld THEN                        //上升沿激活写入状态

    SysMem.SysMemSet(ADR(recvData),0,SIZEOF(recvData));

    //清除读数据区

    //将写入的数据（BYTE）转换为字符串（类型可改）

    SysMem.SysMemCpy (ADR (sendData), ADR (strSendData), SIZEOF (str

SendData));

END_IF

xSendOld:=xSend;                                     //保存上次状态（用于产生上升沿）

IF write.xError THEN                                 //写入错误

    ErrorID:=write.eError;
```

```
iState:=100;

END_IF

100:
    //连接报错后定时一秒后重连
    xError:=TRUE;
    tcpConnect.xEnable:=FALSE;
    ton.IN:=TRUE;
    IF ton.Q THEN
        iState:=10;
        ton.IN:=FALSE;
    END_IF
END_CASE

//读数据
IF read.xReady THEN                                //准备读取数据
    realReadSize:=read.szCount;                    //读取数据的大小
    SysMem.SysMemSet(ADR(sendData),0,SIZEOF(sendData));
    //清除写数据区
    SysMem.SysMemSet(ADR(strRecvData),0,SIZEOF(strRecvData));
    //清空读取数据
    //将读取的数据（BYTE）转换为字符串（类型可改）
    SysMem.SysMemCpy(ADR(strRecvData),ADR(recvData),realReadSize);
END_IF
IF read.xError THEN                                //读取出错
    ErrorID := read.eError;
    iState := 100;
END_IF
ton(IN:= , PT:=T#1S , Q=> , ET=> );                //连接报错后定时一秒后重连
```

e) 上位机（客户端）例程

① C#Socket 通讯流程和主要 API

- 组装 Socket 通信的地址和端口，建立通信的对象：

```
IPAddress ipAddress = IPAddress.Parse(textBox1.Text); //获取 IP 地址
```

```
int Port = Convert.ToInt32(textBox2.Text); //获取端口号
```

- 创建 Socket 对象，建立与服务器的 TCP 连接：

```
MySocket.BeginConnect(ipAddress, Port,  
                        new AsyncCallback(ConnectedCallback), MySocket);
```

- Socket TCP 通信数据交互：

```
socket.BeginReceive(TCPBuffer, 0, TCPBufferSize, 0,  
                    AsyncCallback(ReadCallback), socket); //接收数据
```

```
MySocket.BeginSend(byteArray, 0, byteArray.Length, 0, null, null);  
//发送数据
```

- 关闭 Socket 通信：

```
MySocket.BeginDisconnect(false, null, null); //断开连接
```

② 程序窗口（如图 13.22 所示）和程序内容：



图 13.22 客户端窗口

③ 程序代码：

```
using System;
```

```
using System.Net;
```

```
using System.Text;

using System.Windows.Forms;

using System.Net.Sockets;           //Socket 对应的命名空间
namespace WindowsFormsApp1         //文件名称
{
    public partial class Form1 : Form
    {
        private Socket MySocket = null;    // Socket

        public const int TCPBufferSize = 20000;

        //缓存的最大数据个数(PLC 最大缓存数组容量)

        public byte[] TCPBuffer = new byte[TCPBufferSize];

        //缓存数据的数组

        string IP = "192.168.1.3";    //PLC 地址

        int Port = 8000;              //PLC 端口号

        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            textBox1.Text = IP;

            textBox2.Text = Port.ToString(); ;
        }

        private void button1_Click(object sender, EventArgs e)
        {
            if (button1.Text == "连接")
```

```
{ //IP 地址 和 端口号输入不为空

if (string.IsNullOrEmpty(textBox1.Text) == false &&
    string.IsNullOrEmpty(textBox2.Text) == false)
{
    try
    {
        IPAddress ipAddress = IPAddress.Parse(textBox1.Text); //获取 IP 地址
        int Port = Convert.ToInt32(textBox2.Text); //获取端口号
        MySocket = new Socket(AddressFamily.InterNetwork,
                                SocketType.Stream, ProtocolType.TCP);

        //使用 BeginConnect 异步连接
        MySocket.BeginConnect(ipAddress, AsyncCallback(ConnectedCallback),
                               MySocket);
    }
}

catch (Exception)
{
    MessageBox.Show("IP 地址或端口号错误!", "提示");
}

else
{
    MessageBox.Show("IP 地址或端口号为空!", "提示");
}

else
{
    try
    {
```



```
        button1.Text = "连接";

        MySocket.BeginDisconnect(false, null, null); //断开连接

        label3.Text = "与服务器连接断开!"; //对话框追加显示数据
    }

    catch (Exception) {}

}

}

private void button2_Click(object sender, EventArgs e)
{
    String Str = textBox4.Text.ToString(); //获取发送文本框里面的数据

    try
    {
        if (Str.Length > 0)
        {
            byte[] byteArray = Encoding.Default.GetBytes(Str); //Str 转为 Byte 值
            MySocket.BeginSend(byteArray, 0, byteArray.Length, 0, null, null);
            //发送数据
        }
    }

    catch (Exception) {}
}

void ConnectedCallback(IAsyncResult ar) // 连接异步回调函数
{
    Socket socket = (Socket)ar.AsyncState;

    try
    {

```

```
socket.EndConnect(ar);
```

```
//设置异步读取数据,接收的数据缓存到 TCPBuffer,接收完成跳转
```

ReadCallback 函数

```
socket.BeginReceive(TCPBuffer, 0, TCPBufferSize, 0,
```

```
new AsyncCallback(ReadCallback), socket);
```

```
Invoke((new Action() =>
```

```
{
```

```
    label3.Text = "成功连接服务器! ";//对话框追加显示数据
```

```
    button1.Text = "断开";
```

```
    }));
```

```
}
```

```
catch (Exception e)
```

```
{
```

```
    Invoke((new Action() =>
```

```
    {
```

```
        label3.Text = "连接失败:" + e.ToString();//对话框追加显示数据
```

```
    }));
```

```
}
```

```
}
```

```
void ReadCallback(IAsyncResult ar)          // 接收到数据回调函数
```

```
{
```

```
    Socket socket = (Socket)ar.AsyncState;//获取链接的 Socket
```

```
    int CanReadLen = socket.EndReceive(ar);//结束异步读取回调,获取读取的数据  
    个数
```

```
    if (CanReadLen > 0)
```

```
    {
```

```
        string str = Encoding.Default.GetString(TCPBuffer, 0, CanReadLen);
```

//Byte 值根据 ASCII 码表转为 String

Invoke((new Action() => //C# 3.0 以后代替委托的新方法

{

textBox5.Text = str;//对话框追加显示数据

}}));

//设置异步读取数据,接收的数据缓存到 TCPBuffer,接收完成跳转

ReadCallback 函数

socket.BeginReceive(TCPBuffer, 0, TCPBufferSize, 0,

new AsyncCallback(ReadCallback), socket);

}

else//异常

{

Invoke((new Action() => //C# 3.0 以后代替委托的新方法

{

button1.Text = "连接";

label3.Text = "异常断开";//对话框追加显示数据

}}));

try

{

MySocket.BeginDisconnect(false, null, null);//断开连接

}

catch (Exception) { }

}

}

}

}

f) 程序运行结果如图 13.23、图 13.24 所示。



图 13.23 上位机发送字符串到 PLC



图 13.24 PLC 发送字符串到上位机

2) PLC（客户端）与上位机（服务器）通讯例程

a) PLC（客户端）例程

① 变量声明

PROGRAM PLC_PRG

VAR

(*****TCP_Client*****)

TCPClient :NBS.TCP_Client;

IP : NBS.IP_ADDR := (sAddr:='192.168.1.123'); //服务器 IP 地址

Port :UINT:=8000; //端口号

(*****数据读写*****)

read :NBS.TCP_Read; //读数据函数

```
write:NBS.TCP_Write;                //写数据函数

realReadSize :ULINT;                //读取数据大小

RecvData :ARRAY[0..255]OF BYTE;    //读数据区

SendData :ARRAY[0..255]OF BYTE;    //写数据区

(*****收发内容*****)

strRecvData : STRING;              //自定义接收数据（类型可改）

strSendData : STRING;             //自定义发送数据（类型可改）

xConnect   :BOOL;                  //连接状态

xWrite     : BOOL;                 //写入状态

xWriteOld  : BOOL;                 //上一次写入状态

xError     :BOOL;                  //错误状态

ErrorID: NBS.ERROR;               //错误标志

ton:ton;                            //重连定时器

iState : INT :=0;                  //程序运行状态位

END_VAR
```

② 程序内容

//开启 TCP 客户端模式

```
TCPClient(xEnable:= xConnect, xDone=> , xBusy=> , xError=> ,
          udiTimeOut:= 1000000, ipAddr:= IP, uiPort:= Port,
          eError=> , xActive=> , hConnection=> );
```

//读取数据

```
read(xEnable:= TCPClient.xActive, xDone=> , xBusy=> , xError=> ,
     hConnection:= TCPClient.hConnection, szSize:= SIZEOF(RecvData),
     pData:= ADR(RecvData), eError=> , xReady=> , szCount=> );
```

//写入数据

```
write(xExecute:= xWrite, udiTimeOut:= , xDone=> , xBusy=> , xError=> ,
      hConnection:= TCPClient.hConnection, szSize:= SIZEOF(SendData),
```

```
pData:= ADR(SendData), eError=> );  
  
CASE iState OF  
    0://连接服务器  
        IF xConnect THEN          //使能客户端连接  
            iState :=10;  
        END_IF  
    10://判断是否建立连接  
        IF TCPClient.xActive <> 0 THEN //连接状态  
            iState := 20;  
        END_IF  
        IF TCPClient.xError AND  
            (TCPClient.hConnection = CAA.gc_hINVALID) THEN //连接错  
误  
            ErrorID := TCPClient.eError;  
            iState :=100;  
        END_IF  
    20://写数据  
        IF xWrite AND NOT xWriteOld THEN    //上升沿激活写入状态  
            //清除读取数据  
            SysMem.SysMemSet(ADR(strRecvData), 0,  
SIZEOF(strRecvData));  
            //清除读数据区  
            SysMem.SysMemSet(ADR(RecvData), 0, SIZEOF(RecvData));  
  
            //将写入的数据（BYTE）转换为字符串（类型可改）  
            SysMem.SysMemCpy(ADR(SendData),ADR(strSendData),  
                SIZEOF(strSendData));  
        END_IF
```

```
xWriteOld := xWrite;    //保存上次状态（用于产生上升沿）

IF write.xError THEN    //写入错误

    ErrorID := write.eError;

    iState := 100;

END_IF

100://连接报错定时一秒后重连

xError :=TRUE;

xConnect := FALSE;

ton.IN := TRUE;

IF ton.Q THEN

    iState := 0;

    ton.IN := FALSE;

    xConnect := TRUE;

END_IF

END_CASE

//读数据

IF read.xReady THEN    //准备读取数据

    realReadSize := read.szCount;//读取数据的大小

    SysMem.SysMemSet(ADR(SendData), 0, SIZEOF(SendData));

    //清除写数据区

    SysMem.SysMemSet(ADR(strRecvData), 0, SIZEOF(strRecvData));

    //清空读取数据

    //将读取的数据（BYTE）转换为字符串（类型可改）

    SysMem.SysMemCpy(ADR(strRecvData),ADR(RecvData),realReadSize);

END_IF

IF read.xError THEN    //读取出错

    ErrorID := read.eError;

    iState := 100;
```


END_IF

ton(IN:= , PT:=T#1S , Q=> , ET=>);//连接报错后定时一秒后重连

b) 上位机（服务器）例程

① C#Socket 通讯流程和主要 API

- 组装 Socket 通信的地址和端口，建立通信的对象：

IPAddress ipAddress = IPAddress.Parse(textBox1.Text);//获取 IP 地址

int Port = Convert.ToInt32(textBox2.Text); //获取端口号

- 创建 Socket 对象，建立与服务器的 TCP 连接：

MySocket.Bind(new IPEndPoint(ipAddress, Port)); //启用监听

MySocket.Listen(10); //监听设备数量

- Socket TCP 通信数据交互：

MySocket.Receive(bytes, bytes.Length, 0); //接受数据

MySocket.BeginSend(byteArray, 0, byteArray.Length, 0, null, null);

//发送数据

- 关闭 Socket 通信：

MySocket.Close(); //关闭服务器

② 程序窗口，如图 13.25 所示。



图 13.25 客户端窗口

③ 程序代码：

using System;

using System.Net;

```
using System.Net.Sockets;           //Socket 对应的命名空间
using System.Text;
using System.Threading;
using System.Windows.Forms;
namespace WindowsFormsApp1
{
    public partial class Form1 : Form
    {
        private Socket MySocket = null; // Socket
        Thread Listen;                  //监听线程
        Thread Receive;                 //接收线程
        string IP = "192.168.1.5";      //PC 地址
        int Port = 8000;                 //端口号
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            textBox1.Text = IP;
            textBox2.Text = Port.ToString(); ;
        }
        private void button1_Click(object sender, EventArgs e)
        {
            if (button1.Text == "打开")
            {
                //IP 地址 和 端口号输入不为空
                if (string.IsNullOrEmpty(textBox1.Text) == false &&
```

```
string.IsNullOrEmpty(textBox2.Text) == false)

{
    try
    {
        IPAddress ipAddress = IPAddress.Parse(textBox1.Text); //获取 IP 地址
        int Port = Convert.ToInt32(textBox2.Text);           //获取端口号 MySocket =
        new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.TCP);
        MySocket.Bind(new IPEndPoint(ipAddress, Port));      //监听
        MySocket.Listen(10);                                //监听设备数量
        label3.Text = "处于监听状态，等待客户端连接";
        Listen = new Thread(new ThreadStart(ListenC));      //新建监听线程
        Listen.IsBackground = true; //设置为后台线程，不需要用户主动关闭
        Listen.Start();                                     //启动监听线程
    }
    catch (Exception)
    {
        MessageBox.Show("IP 地址或端口号错误!", "提示");
    }
}

else
{
    MessageBox.Show("IP 地址或端口号为空!", "提示");
}

else
{
    try
    {
```

```
        if (Listen != null)
            Listen.Abort();
        if (Receive != null)
            Receive.Abort();
        MySocket.Shutdown(SocketShutdown.Both);
        MySocket.Close();
        MySocket = null;
        label3.Text = "与客户端连接断开!"; //对话框追加显示数据
        button1.Text = "打开";
    }
    catch (Exception)
    {
    }
}

private void button2_Click(object sender, EventArgs e)
{
    String Str = textBox4.Text.ToString(); //获取发送文本框里面的数据
    try
    {
        if (Str.Length > 0)
        {
            byte[] byteArray = Encoding.Default.GetBytes(Str); //Str 转为 Byte 值
            MySocket.BeginSend(byteArray, 0, byteArray.Length, 0, null, null);
            //发送数据
        }
    }
}
```

```
        catch (Exception) { }
    }
    public void ListenC()
    {
        MySocket = MySocket.Accept();    //与 PLC 客户端连接
        Invoke(new MethodInvoker(delegate ()
        {
            button1.Text = "关闭";
            label3.Text = "连接成功！ ";
        }));
        Receive = new Thread(new ThreadStart(ReceiveC));    //新建接收线程
        Receive.IsBackground = true; //设置为后台线程，不需要用户主动关闭
        Receive.Start();    //开启接收线程
    }
    public void ReceiveC()
    {
        byte[] bytes = new byte[1024];    //数组容量
        int len = MySocket.Receive(bytes, bytes.Length, 0);
        //将消息转为字符串
        string str = Encoding.Default.GetString(bytes, 0, len);
        //Byte 值根据 ASCII 码表转为 String
        Invoke(new MethodInvoker(delegate ()
        {
            textBox5.Text=Encoding.Default.GetString(bytes, 0, len);
        }));
    }
}
}
```

c) 程序运行结果，如图 13.26、图 13.27 所示。



图 13.26 上位机发送字符串到 PLC

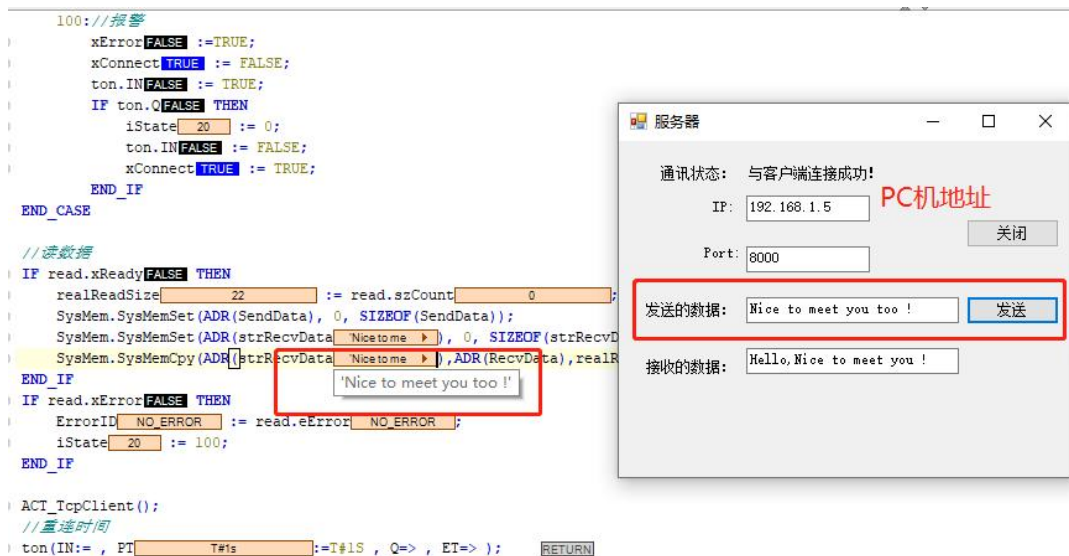


图 13.27 上位机发送字符串到 PLC

注意：

- ① strSendData 为发送数据，本例中为 string 类型，看具体情况，结构体或其他类型同样适用，只需将需要发送的数据拷贝到 byWrite 字节数据即可。若发送数据量较大，可修改字节数组大小（0~1999）；
- ② strReceiveData 为接收数据，本例中为 string 类型，看具体情况，若为结构体，或其他类型同样适用，只需要将 byRead 字节数组拷贝到相应的类型即可，若接收数据量较大，则需修改 byRead 字节数组的大小（0~1999）。

- ③ 若要连接多个客户端，则实例化多个 TCP_Connect，得到 TCP_Connect 句柄后输入给 TCP_Read 和 TCP_Write 即可进行收发数据。将 TCP_Server 的句柄 tcpSvr.hServer 输入给每个 TCP_Connect。程序与单个客户端连接时，只需要实例化一个 TCP_Server 即可。

另外，TCP 库在附件程序中给出。这里不做详细说明。用法与上述 CAA 库相似，不再赘述。

3) TCP 通讯故障码

故障码可在 CAA Net Services 库文件中查询相关说明，如图 13.28 所示。

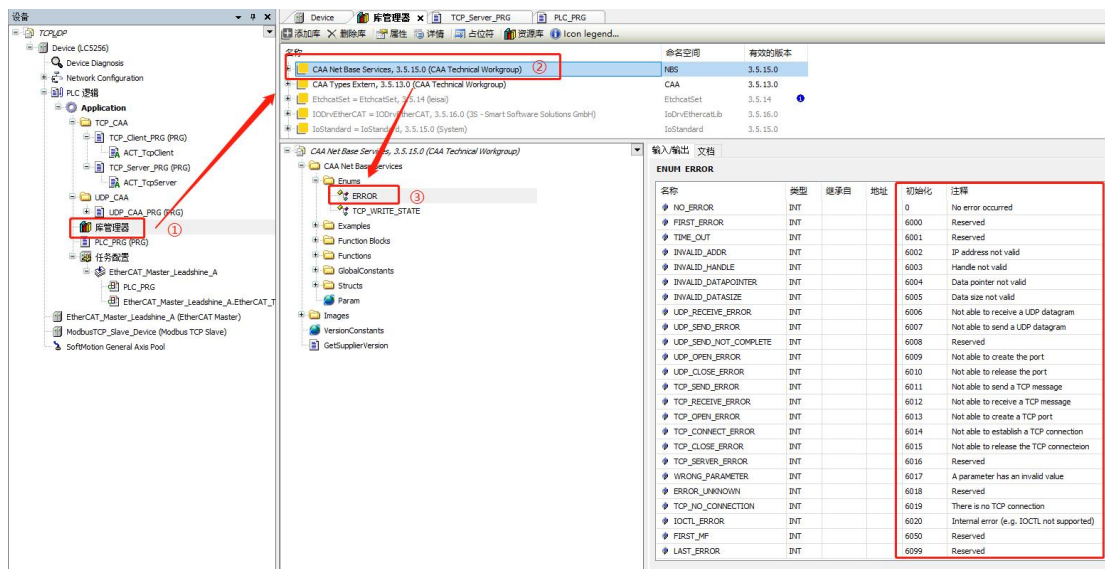


图 13.28 查询 TCP 通讯故障码

13.3. UDP 自由协议

本节介绍 UDP 自由协议通讯。UDP (User Datagram Protocol) 是一个简单的面向消息的传输层协议，其特点是：简单、轻量化；但没有流控制，没有应答确认机制，不能解决丢包、重发、错序问题。

使用 UDP 自由协议通讯，通讯库的配置方法与 TCP 相同。

13.3.1. UDP 指令的使用

使用 UDP 指令相比 TCP 指令更简单，是直接往端口上发数据。UDP 通讯指令如下。

1) 功能块 UDP_Peer

通过 xEnable 信号激活 UDP，设置本机端口即可进行数据收发。设置 IP 地址的功能块 UDP_Peer 如图 13.29 所示，参数定义如表 10.22 所示。

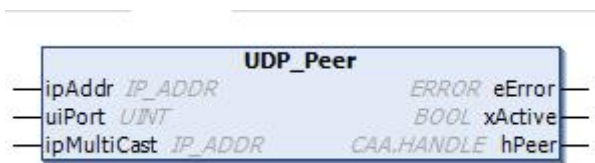


图 13.29 功能块 UDP_Peer

功能块 UDP_Peer 的参数定义

Scope	Name	Type	Initial	Comment
Input	ipAddr	IP_ADDR	STRUCT(sAddr := '0.0.0.0')	'0.0.0.0' => INADDR_ANY, '127.0.0.1' => INADDR_LOOPBACK, '255.255.255.255' => INADDR_NONE
	uiPort	UINT		
	ipMultiCast	IP_ADDR		Multicast address. '255.255.255.255' => INADDR_NONE
Output	eError	ERROR		
	xActive	BOOL		TRUE if the handle is valid
	hPeer	CAA_HANDLE		

2) 功能块 UDP_Receive

如图 13.30 所示，功能块 UDP_Receive 用于接收数据。pData 是将读到的数据写入内存区域的指针，szSize 定义要读取的最大字节数。当成功读取数据时，xReady 被设置为 TRUE，szCount 表示实际读取的字节数。ipFrom 和 uiPortFrom 表示接收数据的 ip 地址和端口。参数定义如表 13.23 所示。

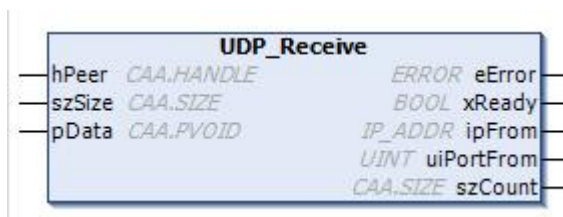


图 13.30 功能块 UDP_Receive

表 13.23 功能块 UDP_Receive 的参数定义

Scope	Name	Type
Input	hPeer	CAA.HANDLE
	szSize	CAA.SIZE
	pData	CAA.PVOID
Output	eError	<u>ERROR</u>
	xReady	BOOL
	ipFrom	<u>IP_ADDR</u>
	uiPortFrom	UINT
	szCount	CAA.SIZE

3) 功能块 UDP_ReceiveBuffer

与 UDP_Receive 一样，该功能块用于接收数据，如图 13.31 所示。功能块返回包含数据的缓存区句柄，xReady 为 TRUE，可以读取缓存区数据，同时可接收到远端的 IP 和端口号；缓冲区被传递，而不像 UDP_Receive 那样被复制。参数定义如表 13.24 所示。

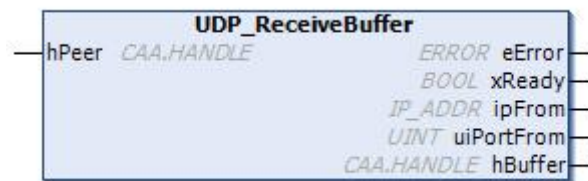


图 13.31 功能块 UDP_ReceiveBuffer

表 13.24 功能块 UDP_ReceiveBuffer 的参数定义

Scope	Name	Type
Input	hPeer	CAA.HANDLE
Output	eError	<u>ERROR</u>
	xReady	BOOL
	ipFrom	<u>IP_ADDR</u>
	uiPortFrom	UINT
	hBuffer	CAA.HANDLE

4) 功能块 UDP_Send

如图 13.32 所示，功能块 UDP_Send 用于发送数据。指针 pData 指示的内存区域的内容包含要发送的数据。szSize 定义要发送的字节数。当 xDone 设置为 TRUE

时，数据发送成功。参数定义如表 13.25 所示。

超时时间对这个函数块没有影响。

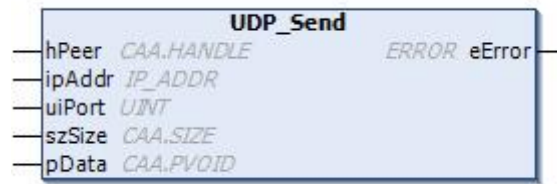


图 13.32 功能块 UDP_Send

表 13.25 功能块 UDP_Send 的参数定义

Scope	Name	Type
Input	hPeer	CAA.HANDLE
	ipAddr	IP_ADDR
	uiPort	UINT
	szSize	CAA.SIZE
	pData	CAA.PVOID
Output	eError	ERROR

5) 功能块 UDP_SendBuffer

与 UDP_Send 一样，UDP_SendBuffer 功能块用于发送数据，如图 13.33 所示。这个功能块不是一个指向数据的指针，而是一个包含要写入数据的缓冲区句柄。缓冲区被传递而不是像 UDP_Send 那样被复制。参数定义如表 13.26 所示。

超时值对这个函数块没有影响。



图 13.33 功能块 UDP_SendBuffer

表 13.26 功能块 UDP_SendBuffer 的参数定义

Scope	Name	Type
Input	hPeer	CAA.HANDLE
	ipAddr	<u>IP_ADDR</u>
	uiPort	UINT
	hBuffer	CAA.HANDLE
Output	eError	<u>ERROR</u>

13.3.2. UDP 通讯例程

PLC 与 PC 机连接,PC 机用调试助手软件收发数据。PLC 的程序如图 13.34、13.35 所示。

```
1  PROGRAM UDP
2  VAR
3      UDP : NBS.UDP_Peer;
4      ipAddr : NBS.IP_ADDR := (sAddr:='0.0.0.0'); //IP地址
5      uiPort :UINT:=10001;
6      Receive : NBS.UDP_Receive;
7      Send : NBS.UDP_Send;
8      byRead : ARRAY[0..1499]OF BYTE;
9      bySend : ARRAY[0..1499]OF BYTE;
10     strSendData : STRING;
11     strRecvData :STRING;
12     realReadSize :ULINT;
13     (*****远端IP和端口*****)
14     ipRemote :NBS.IP_ADDR:= (sAddr:='192.168.70.113');;
15     uiRemotePort :UINT:=10002;
16     uiRemotePortbak :UINT:=10011;
17
18     (*****读取的IP和端口*****)
19     ipReceive :NBS.IP_ADDR;
20     uiReceivePort :UINT;
21
22     xEnable :BOOL;
23     xSend : BOOL;
24     xSendOld : BOOL;
25
```

图 13.34 UDP 通讯例程：变量声明

```

1  (****UDP通讯****)
2  xEnable :=TRUE;
3  (*****发送数据到端口****端口号可以任选, 可以修改端口号发送到不同的设备上****)
4  Send.uiPort := uiRemotePort;
5  //Send.uiPort := uiRemotePortbak;
6  Send.ipAddr := ipRemote;
7  IF xSend AND NOT xSendOld THEN
8      SysMem.SysMemSet(ADR(bySend), 0, SIZEOF(bySend));
9      SysMem.SysMemCpy(ADR(bySend),ADR(strSendData),SIZEOF(strSendData));
10 END_IF
11 xSendOld := xSend;
12
13 (*****从远端端口接收数据***** )
14 IF Receive.xReady THEN
15     ipReceive := Receive.ipFrom;
16     uiReceivePort := Receive.uiPortFrom;
17     realReadSize := Receive.szCount;
18     SysMem.SysMemSet(ADR(strRecvData), 0, SIZEOF(strRecvData));
19     SysMem.SysMemCpy(ADR(strRecvData),ADR(byRead),realReadSize);
20 END_IF
21
22 (*****功能块实例***** )
23 UDP( xEnable:=xEnable , xDone=> , xBusy=> , xError=> , ipAddr:= ipAddr, uiPort:=uiPort ,
24     ipMultiCast:= , eError=> , xActive=> , hPeer=> );
25 Receive( xEnable:=UDP.xActive , xDone=> , xBusy=> , xError=> , hPeer:=UDP.hPeer , szSize:=SIZEOF(byRead) ,
26     pData:= ADR(byRead) , eError=> , xReady=> , ipFrom=> , uiPortFrom=> , szCount=> );
27 Send( xExecute:= xSend , udiTimeOut:= , xDone=> , xBusy=> , xError=> , hPeer:=UDP.hPeer ,
28     ipAddr:= , uiPort:= , szSize:=SIZEOF(bySend) , pData:= ADR(bySend) , eError=> );
29

```

图 13.35 UDP 通讯例程：程序代码

例程运行效果如图 13.36 所示，“调试助手”软件向 PLC 端口号为 10001 的端口



发送 “how are you ?”

图 13.36 “调试助手” 软件向 PLC 发送数据

PLC 接收到数据，并显示远端发送的 IP 和端口号，如图 13.37 所示。

Device.Application.UDP

```

1  (*****UDP通讯*****)
2  xEnable TRUE := TRUE;
3  (*****发送数据到端口***端口号可以任选, 可以修改端口号发送到不同的设备上****)
4  Send.uiPort 10002 := uiRemotePort 10002;
5  //Send.uiPort := uiRemotePortbak;
6  Send.ipAddr := ipRemote;
7  IF xSend FALSE AND NOT xSendOld FALSE THEN
8    SysMem.SysMemSet (ADR (bySend), 0, SIZEOF (bySend));
9    SysMem.SysMemCpy (ADR (bySend), ADR (strSendData 2222222), SIZEOF (strSendData 2222222));
10   END_IF
11   xSendOld FALSE := xSend FALSE;
12
13  (*****从远端端口接收数据*****)
14  IF Receive.xReady FALSE THEN
15    ipReceive := Receive.ipFrom;
16    uiReceivePort 10002 := Receive.uiPortFrom 0;
17    realReadSize 13 := Receive.szCount 0;
18    SysMem.SysMemSet (ADR (strRecvData how are yo), 0, SIZEOF (strRecvData how are yo));
19    SysMem.SysMemCpy (ADR (strRecvData how are yo), ADR (byRead), realReadSize 13);
20   END_IF
21

```

表达式	应用	类型	值	准备值	执行点
TCPClient.strRecvData	Device.Application				循环监!
TCPClient.strSendData	Device.Application				循环监!
UDP.ipReceive	Device.Application	NBS.IP_ADDR	192.168.70.113		循环监!
sAddr		STRING(80)			循环监!

图 13.37 PLC 接收到数据和对方的 IP 地址

PLC 向 “调试助手 1”（端口号为 10000）发送字符串 “I am fine! Thank you. And you ?”，需要指定 IP 和端口号(10002)；如图 13.38 所示。

PLC 同时向 “调试助手 2”（端口号为 10010）发送字符串 “How old are you ?”，如图 13.39 所示。

Device.Application.UDP

```

4  Send.uiPort 10002 := uiRemotePort 10002;
5  //Send.uiPort := uiRemotePortbak;
6  Send.ipAddr := ipRemote;
7  IF xSend FALSE AND NOT xSendOld FALSE THEN
8    SysMem.SysMemSet (ADR (bySend), 0, SIZEOF (bySend));
9    SysMem.SysMemCpy (ADR (bySend), ADR (strSendData I am fine!), SIZEOF (strSendData I am fine!));
10   END_IF
11   xSendOld FALSE := xSend FALSE;
12
13  (*****从远端端口接收数据*****)
14  IF Receive.xReady FALSE THEN
15    ipReceive := Receive.ipFrom;
16    uiReceivePort 10002 := Receive.uiPortFrom 0;

```

网络通信窗口:

- UDP (2) 本地主机地址: 192.168.70.113
- (3) 本地主机端口: 10002
- 接收设置: ASCII, 按日志模式显示
- 日志: [2021-11-17 16:17:15.536]# SEND ASCII TO 192.168.70.113:10001 how are you ?
- [2021-11-17 16:23:30.463]# RECV ASCII FROM 192.168.70.113:10001 I am fine! Thank you. And you ?

图 13.38 “调试助手 1” 发送数据

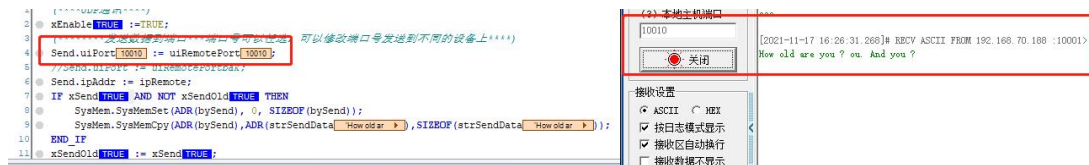


图 13.39 “调试助手 2”发送数据

注意：

- ① 数据接收需要根据远端的 IP 和端口号进行校验，明确发送数据的设备；
- ② 数据发送需要指定远端的 IP 和端口号，正确的发送到指定设备中；
- ③ UDP 发送数据是作为生产者，接收数据作为消费者。

本例程中使用的是 CAA 的库中的功能块。如果想使用自带的 TCP 库和 UDP 库做通讯，请参见 LC 系列产品的例程 TCP&UDP.project。

13.4. 标签通讯

标签通讯常用于 PLC 与触摸屏通讯。

13.4.1. 标签通讯的流程

标签通讯的流程如图 13.40 所示。

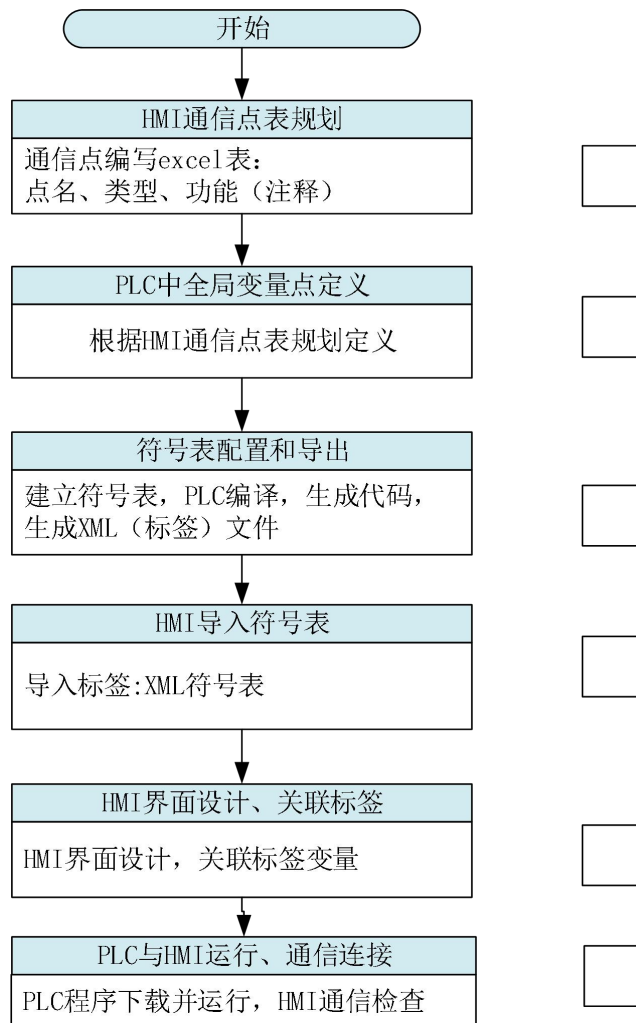


图 13.40 标签通讯的流程

13.4.2. 标签通信例程

下面以 PLC 与 PC 机上的虚拟触摸屏通讯为例，介绍标签通讯的方法。

- 1) 在当前工程左侧树中选中“application”，鼠标右键后，点“添加对象”，选“符号配置”。如图 13.41 所示。

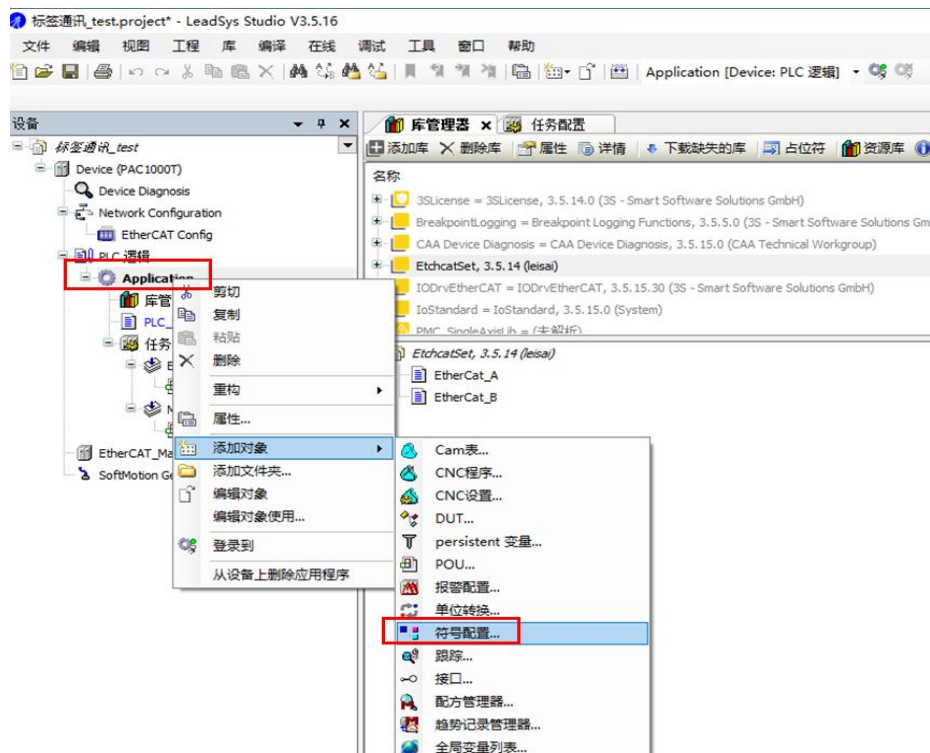


图 13.41 添加“符号配置”

- 2) 弹出下面窗口后，勾选“支持 OPC UA”特征，如图 13.42 所示。



图 13.42 选择“支持 OPC UA”

- 3) 在左侧树中双击“符号配置”，打开符号配置页面，PLC_PRG 代码区增加一行加法计算： $m0 = m1 + m2$ ；然后编译工程。如图 13.43 所示。

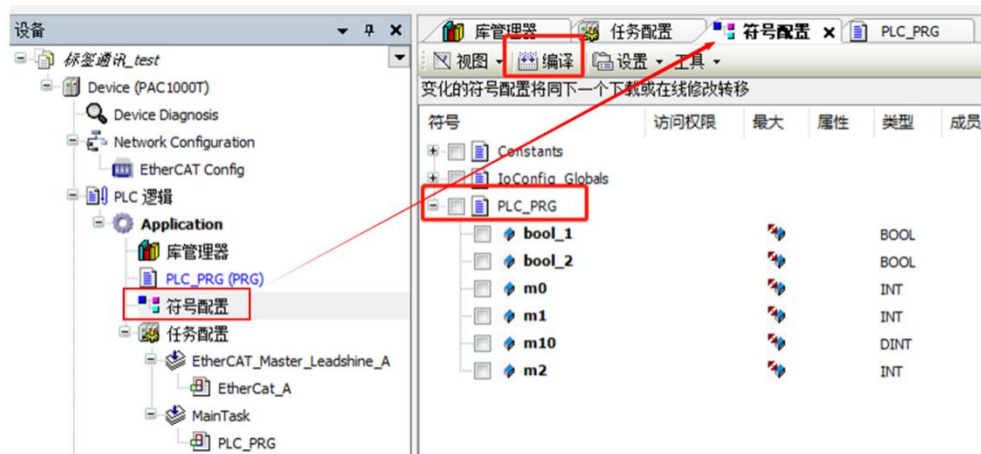


图 13.43 打开“符号配置”页面

- 4) 然后勾选需要进行标签通信的变量（图 13.44 示例中变量都选了，实际使用时只勾选需要的变量），然后点击“生成代码”，如图 13.44 所示，可编译生成标签通信的 XML 文件。

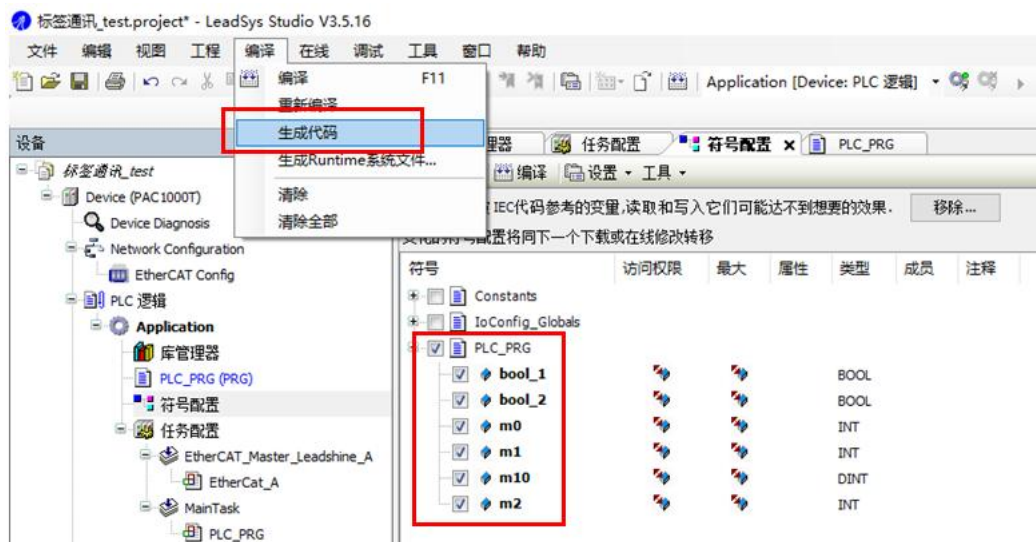


图 13.44 勾选进行标签通信的变量

- 5) 打开 PLC 工程所在目录，可看见生成的 XML 文件，如图 13.45 所示。这个 XML 文件就是用于标签通信。

名称	修改日期	类型	大小
标签通讯_test.Device.Application	2021/11/16 16:17	XML 文档	2 KB
标签通讯_test	2021/11/16 16:11	CODESYS project	136 KB
标签通讯_test.project.~u	2021/11/16 16:17	~U 文件	1 KB
标签通讯_test-Administrator-DESKTO...	2021/11/16 16:11	OPT 文件	15 KB
标签通讯_test-AllUsers.opt	2021/11/16 16:11	OPT 文件	1 KB

图 13.45 查看生成的 XML 文档

做完 PLC 上述 5 个步骤后，以威纶通触摸屏 MT8102iE 配置为例，介绍触摸屏的配置。

- 6) 打开 EasyBuilder Pro 软件，新建文件，选择 MT8102iE，进行系统参数设置，如图 13.46 所示，点击“新增设备及服务器”。

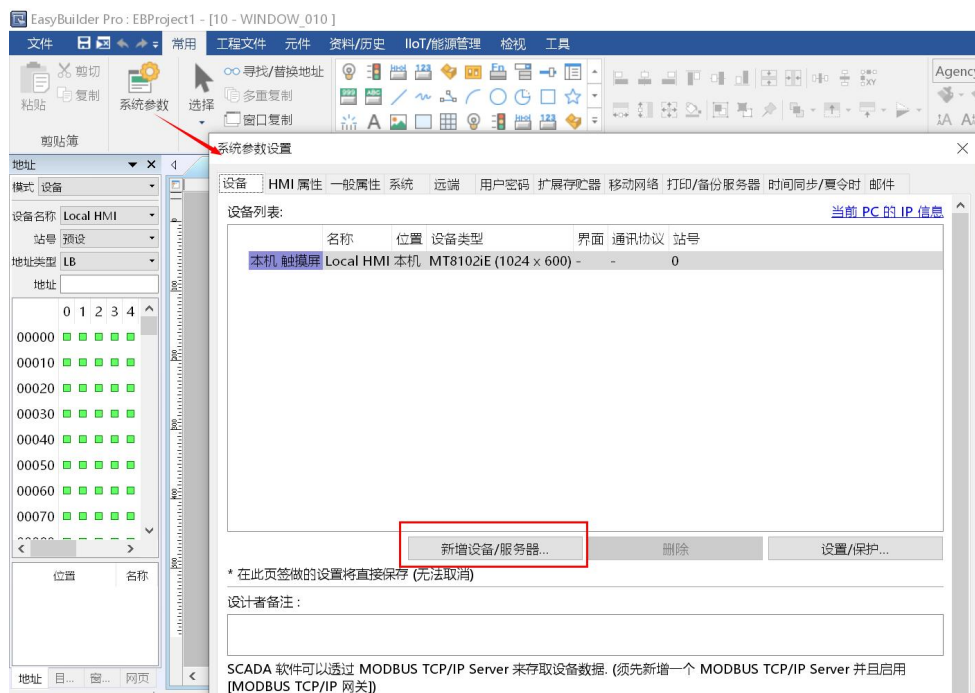


图 13.46 EasyBuilder Pro 软件界面

- 7) 点击“新增设备及服务器”后，配置触摸屏通信配置，如图 13.47、13.48 所示，输入 codesys，点击查找，选择 CODESYS V3(ethernet)。

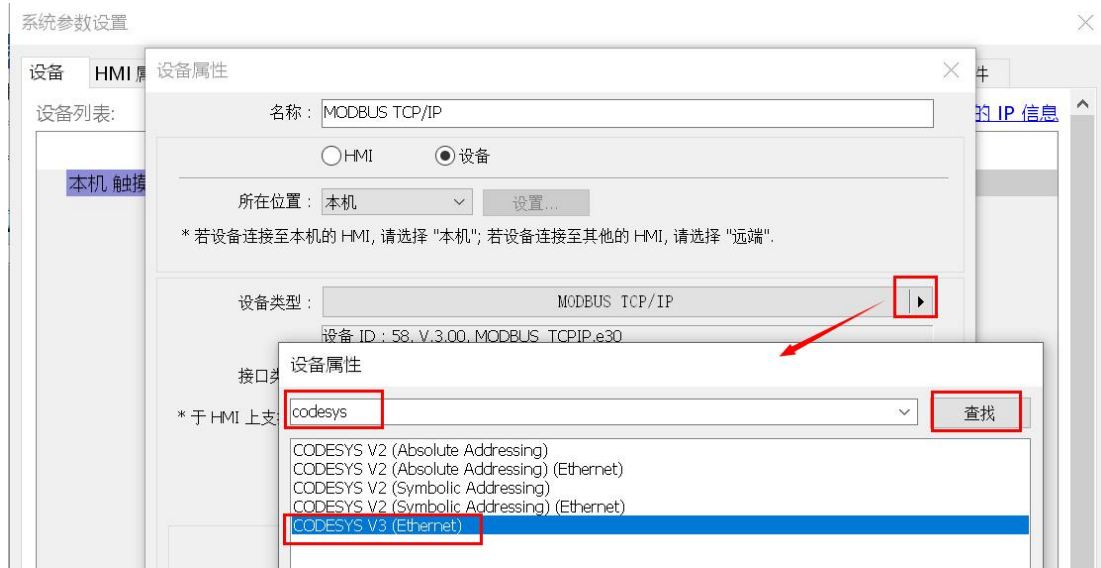


图 13.47 选择通讯设备

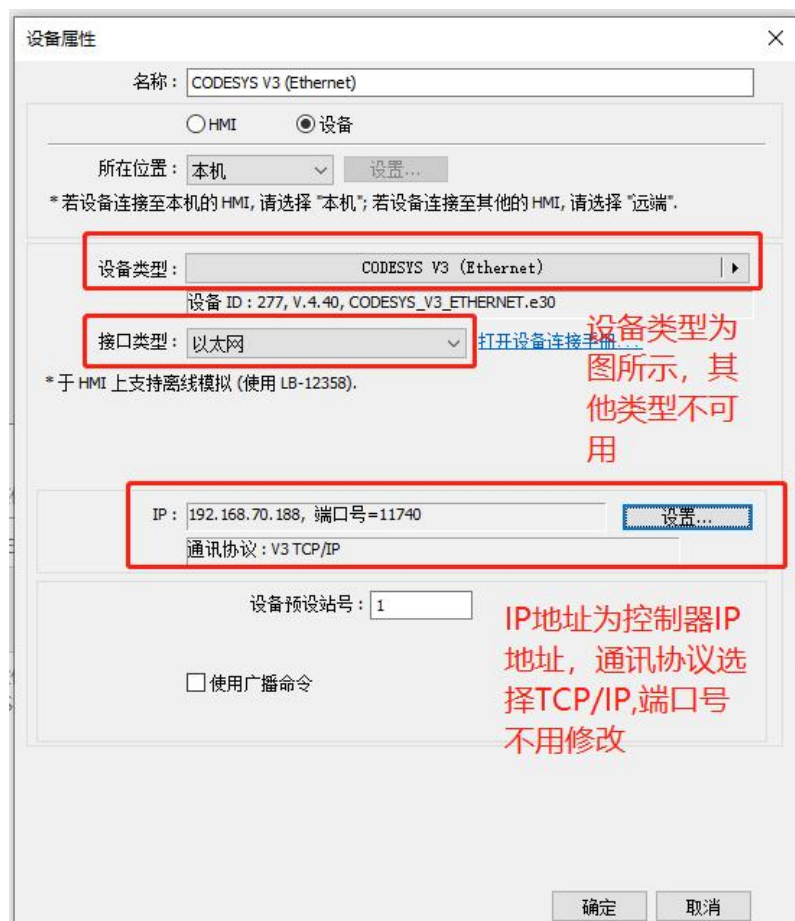


图 13.48 设置通讯设备的参数

8) 确定后，点击“导入标签”，如图 13.49 所示。



图 13.49 导入标签

9) 选择刚才 PLC 编译生成代码时，生成的 XML 文档，如图 13.50 所示。

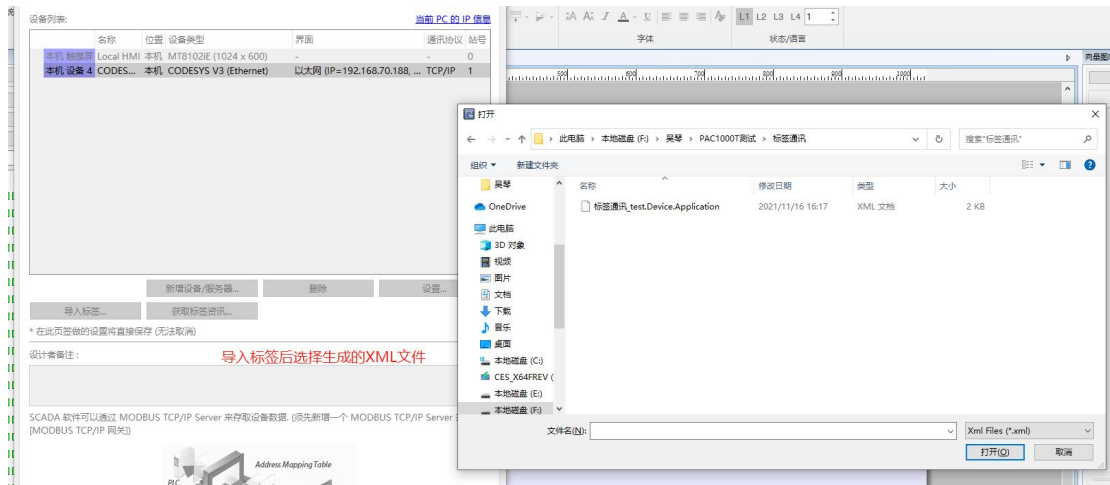


图 13.50 选择 XML 文件

10) 触摸屏界面增加一个数字元件，选择对应的标签，第一个选 m0，然后确定，如图 13.51 所示。

11) 依次选择 m1 和 m2，触摸界面如图 13.52 所示。然后运行 PLC 程序。

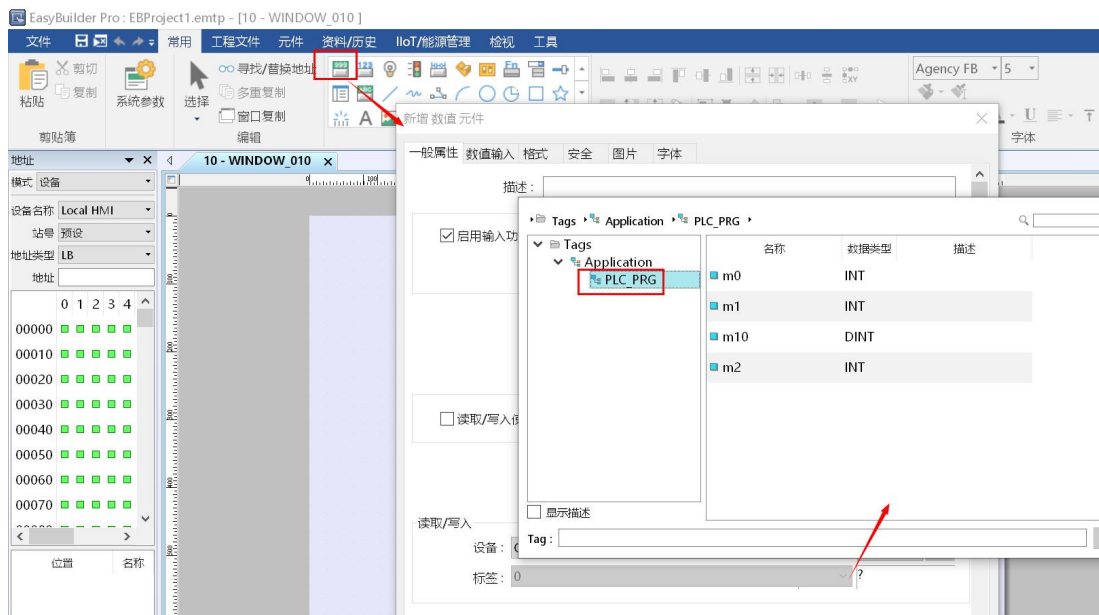


图 13.51 新增数字元件

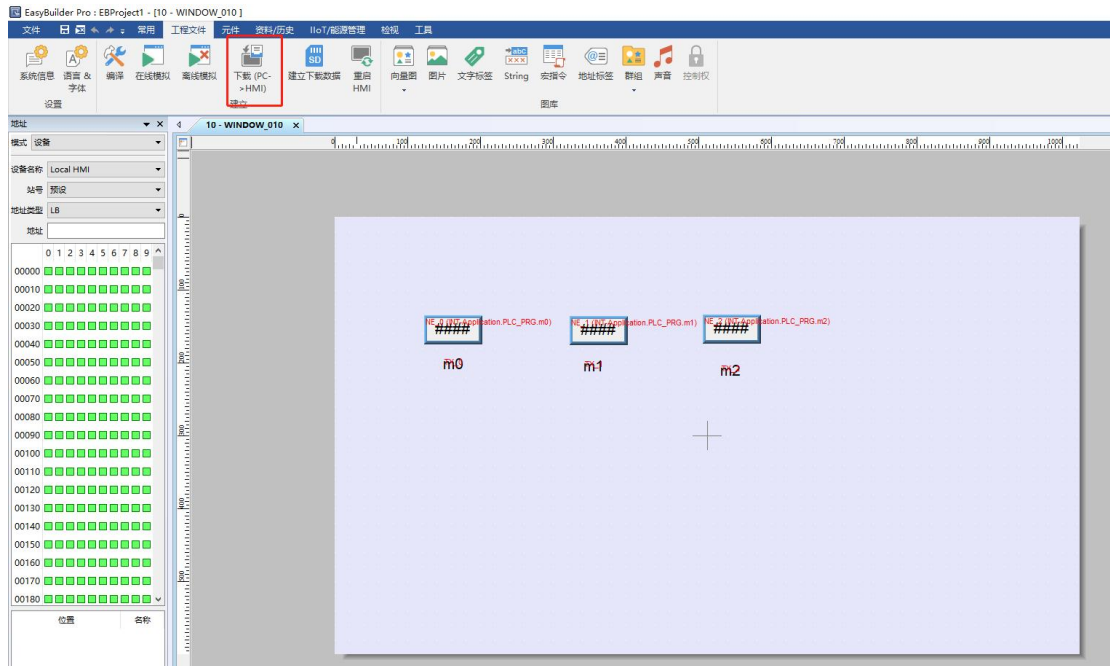


图 13.52 触摸屏界面

12) 如图 13.53 所示，打开在线模拟，将 m1 设置为 80，m2 设置为 50。可以看见模拟界面，和触摸屏上 $m0 = 130$ ，如图 13.54 所示。这表明触摸屏与 PLC 标签通信正常。



图 13.53 启动“在线模拟”

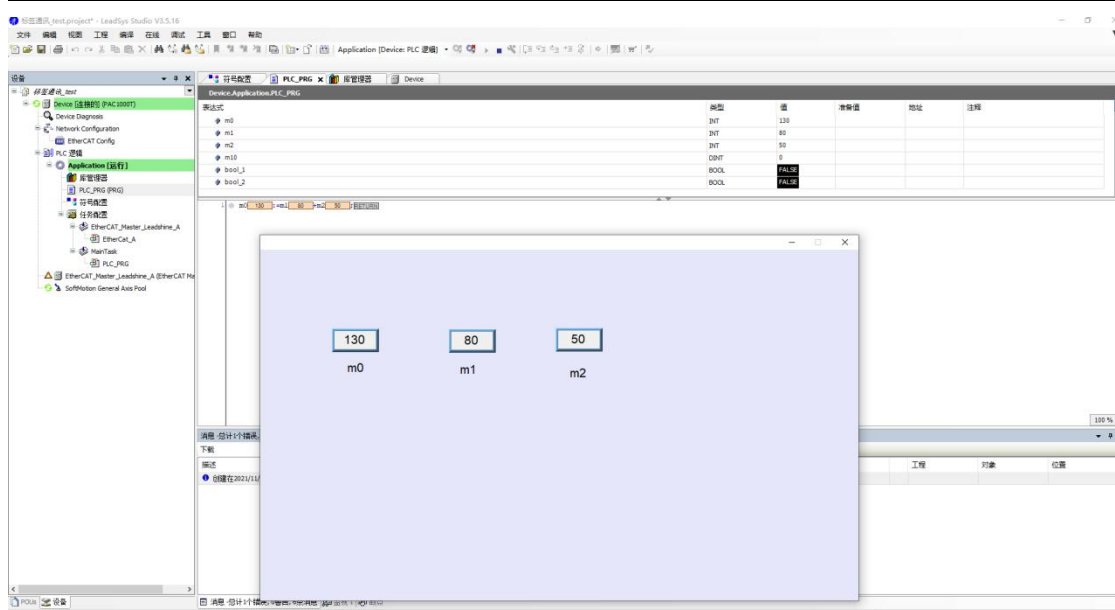


图 13.54 运行结果

13.5. OPC UA 服务器

OPC UA（OPC Unified Architecture）基于国际电工委员会（IEC）和电气和电子工程师协会（IEEE）国际标准搭建，为工业互联网网络体系构建提供了标准化模块，是建立从传感器到云端大带宽、高同步、广兼容通讯的技术。使用 OPC UA 通讯统一了各种各样不同品牌控制器的通讯机制和数据交互格式，实现了通讯标准化。

13.5.1. OPC UA 通讯的流程

OPC UA 通讯的流程如图 13.55 所示。

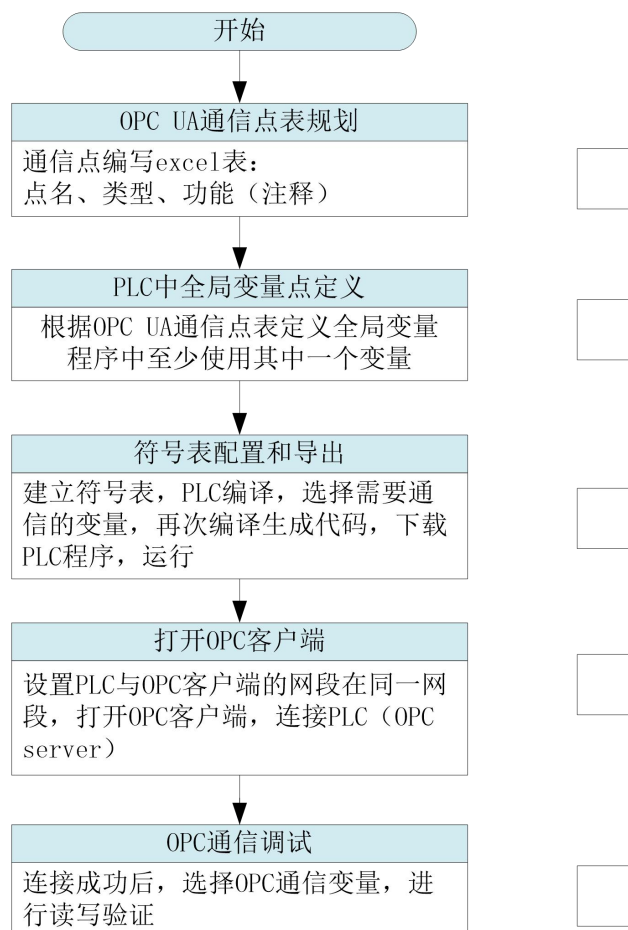


图 13.55 OPC UA 通讯的流程

13.5.2. PLC 配置 OPC 功能的方法

- 1) 将需要进行 OPC 通信的变量定义在全局变量中，如图 13.56 所示。

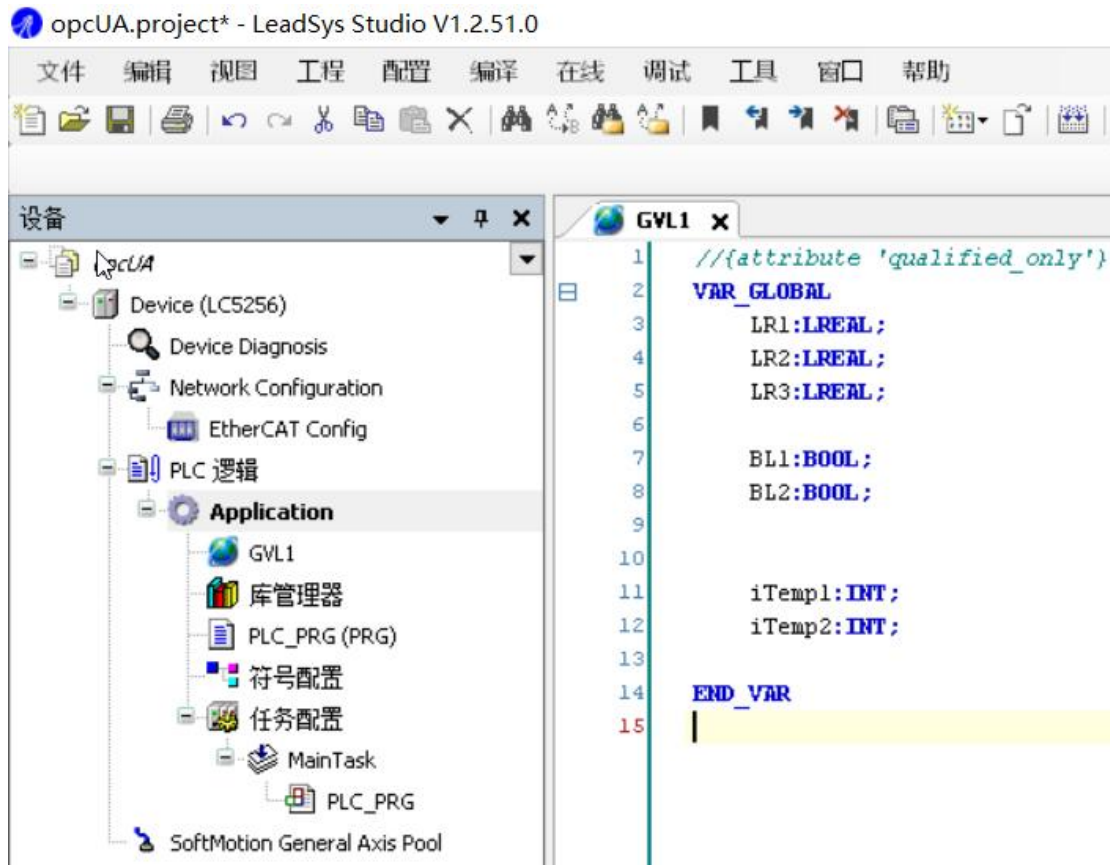


图 13.56 定义全局变量

- 2) 在程序中使用全局变量(因为未被使用的变量,不会出现在通信列表中), 如图 13.57 所示。

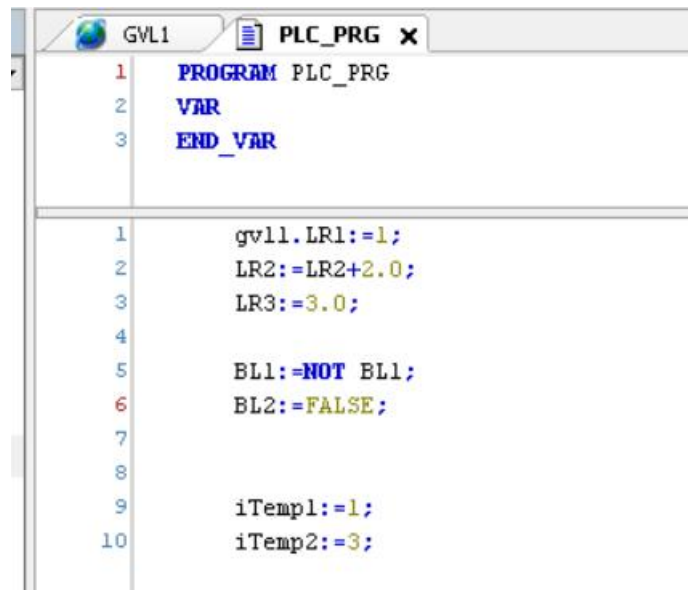


图 13.57 在程序中使用全局变量

- 3) 添加符号配置表。鼠标右键点击左侧“设备”窗口里的“Application”，选择“添加对象”→“符号配置”，如图 13.58 所示。

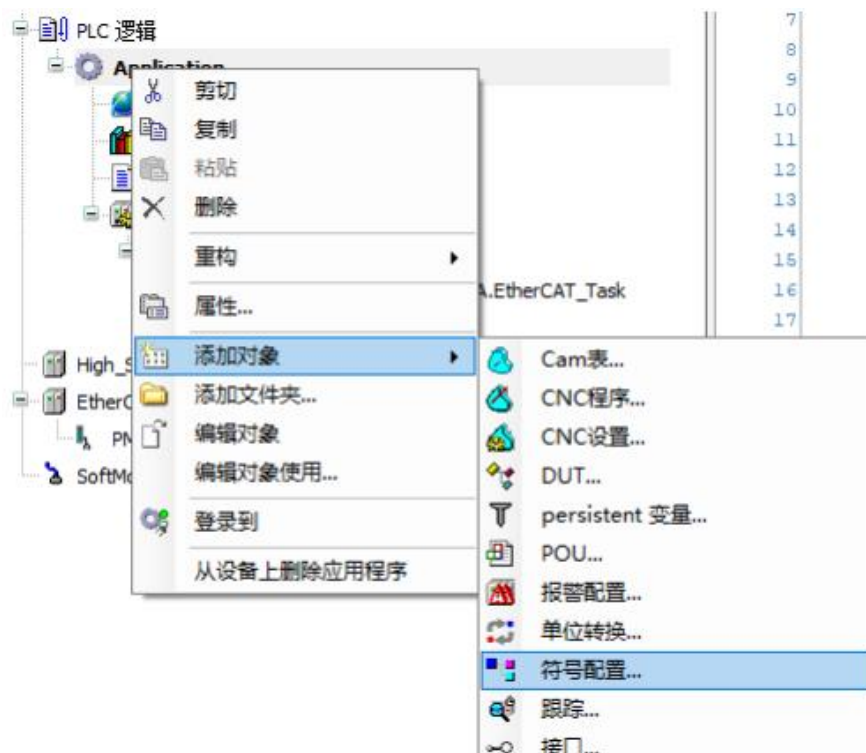


图 13.58 添加“符号配置”

在弹出界面中选择“支持 OPC UA 特征”，再点击“打开”，如图 13.59 所示。



图 13.59 勾选“支持 OPC UA 特征”

4) 选择用于通讯的变量。在符号配置表中点击“编译”，如图 13.60 所示。



图 13.60 在“符号配置”界面里点击“编译”

编译之后，会在下方列出程序中可用于通讯的变量，如图 13.61 所示。有两种情况变量不会出现在列表中：

- a) 中文变量；
- b) 变量表中的变量在程序中没有使用过。



图 13.61 可用于通讯的变量

编译之后，用户根据自身需要选择通讯的变量，这里我们选择 GVL_1 的部分变量；（如果程序中没有使用到该变量，会弹出下方提示，如图 13.62 所示，可自行选择是否移除）

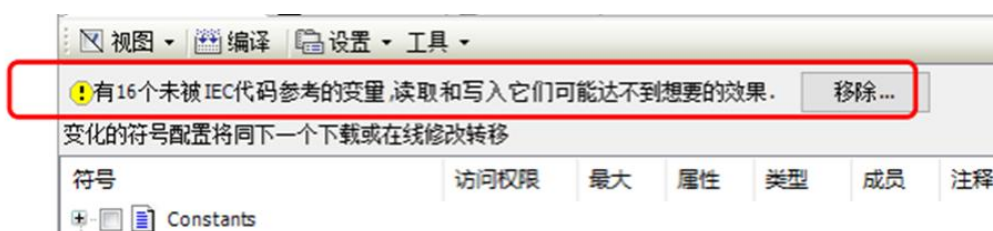


图 13.62 提示信息

- 5) 将程序下载到 PLC 中，即可使配置生效。
- 6) 选择一个 Ethernet 口作为 OPC UA 通信接口，记录该网口的 IP，也可以通过编程工具 LeadSys Studio 查询网口的 IP，如图 13.63 所示，在“IP 地址设置”栏，选择“端口号”，点击“读取”。

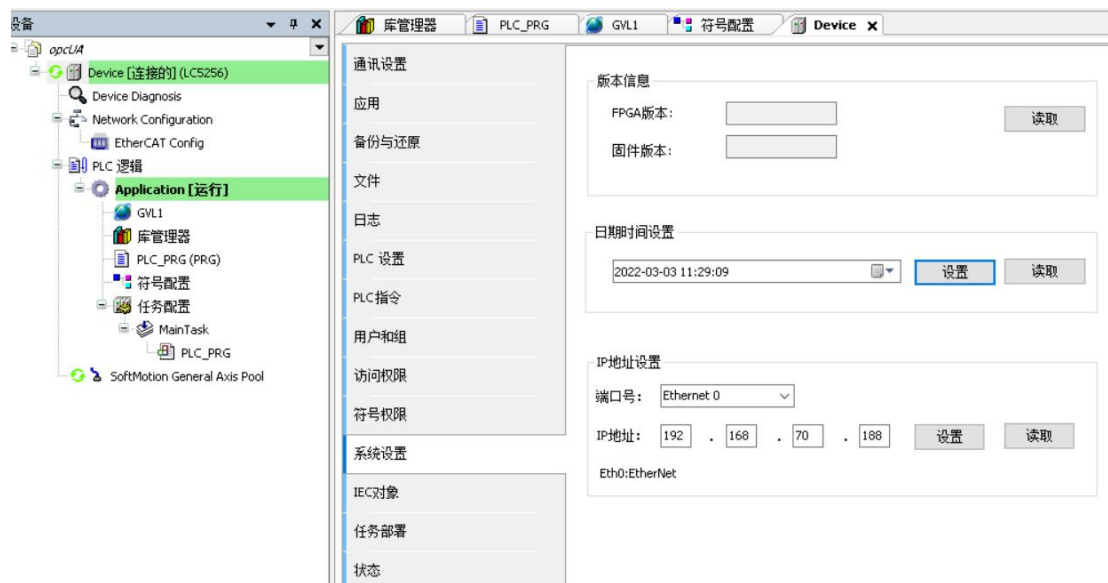


图 13.63 读取网口的 IP 地址

13.5.3. 使用 UaExpert 软件调试 OPC UA 通讯

在项目实际应用时，客户可以选择自己的 OPC UA 进行通信连接。本节介绍使用 UaExpert 软件进行 OPC UA 通讯的调试，验证 PLC 的 OPC UA 通信功能。方法如下。

1) 安装 UaExpert 软件后，打开 UaExpert 软件，添加 OPC UA 服务器。

打开 UaExpert 后，鼠标右键点击“Server”，选择“Add”，添加服务器，如图 13.64 所示。

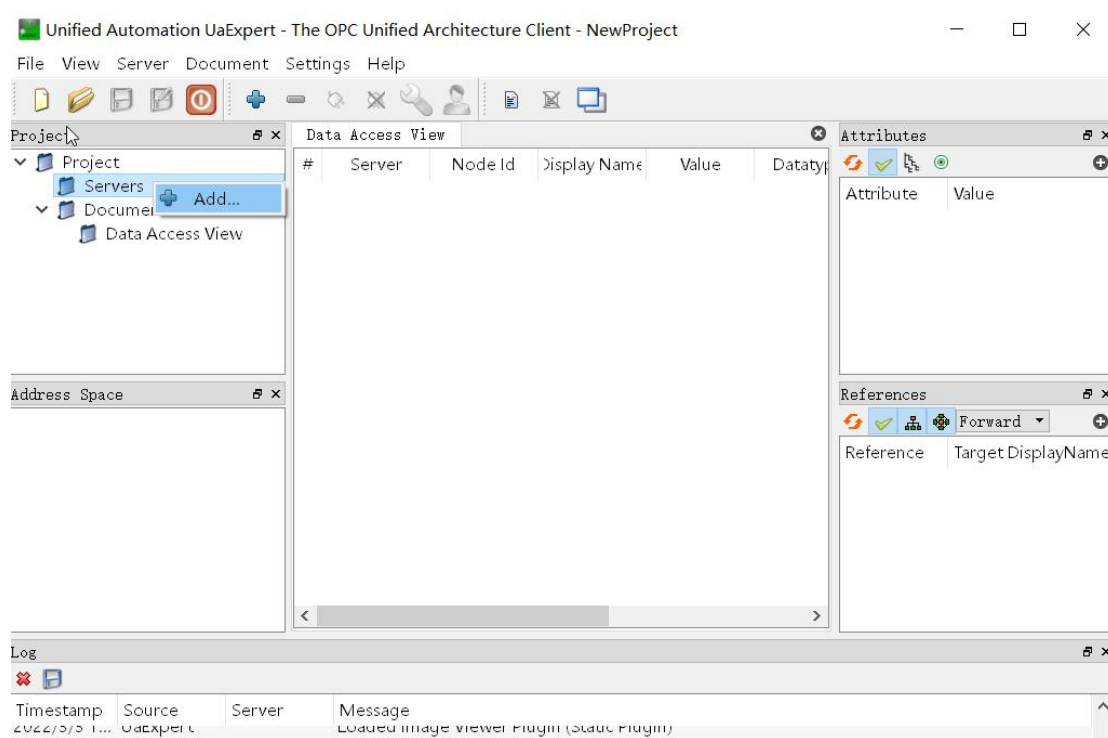


图 13.64 UaExpert 软件界面

- 2) 在弹出的窗口中，选择“Advance”选项卡，在“Configuration Name”中输入服务器名称，可自定义，如 LC5000。然后输入服务器的 URL，格式为 `opc.TCP://<PLC IP 地址>: 4840`；此例中 PLC 的 IP 地址为 192.168.70.188，故对应的 URL 为 `opc.TCP://192.168.70.188:4840`。输入完成后点击“OK”即可。如图 13.65 所示。

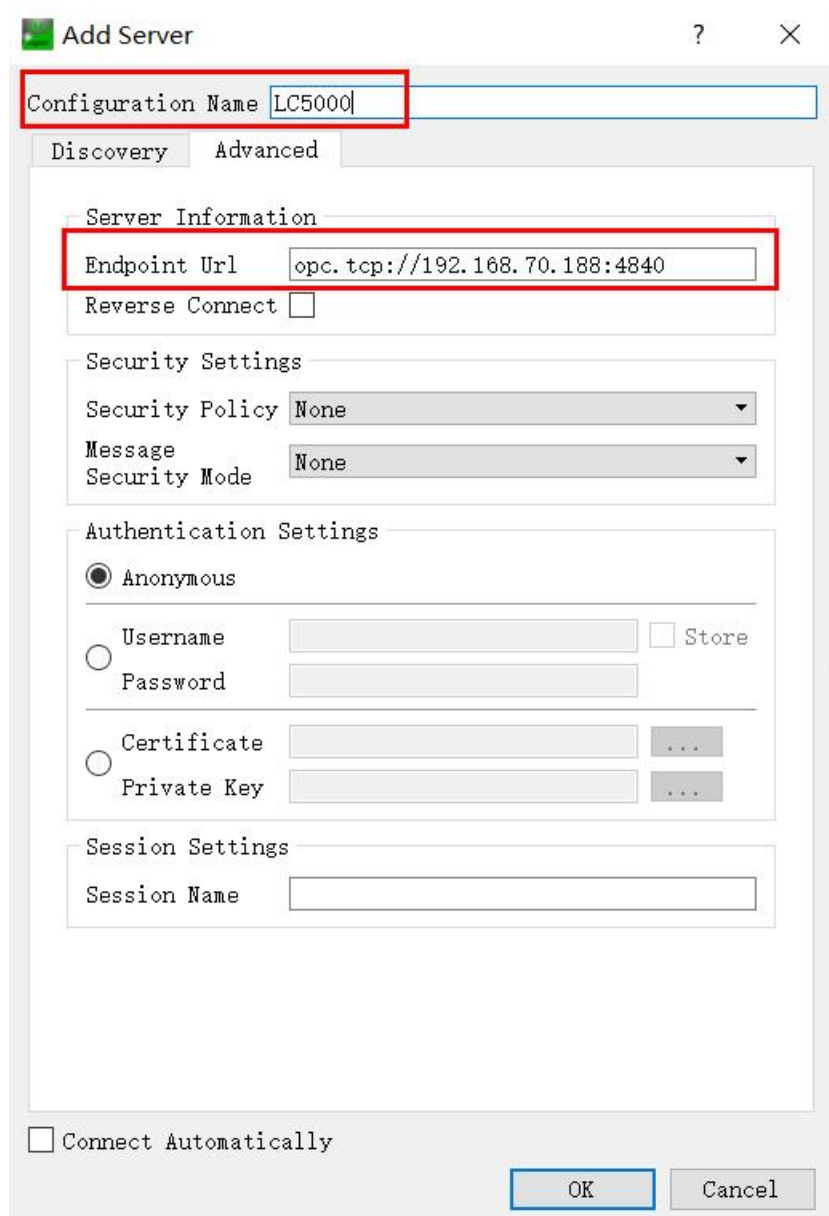


图 13.65 设置服务器参数

- 3) 设置调试电脑的网口 IP 地址与 PLC 的 IP 在同一网段，然后调试电脑的网口连接 PLC 的网口。
- 4) 在调试电脑的 UaExpert 软件界面，如图 13.66 所示，连接 PLC。

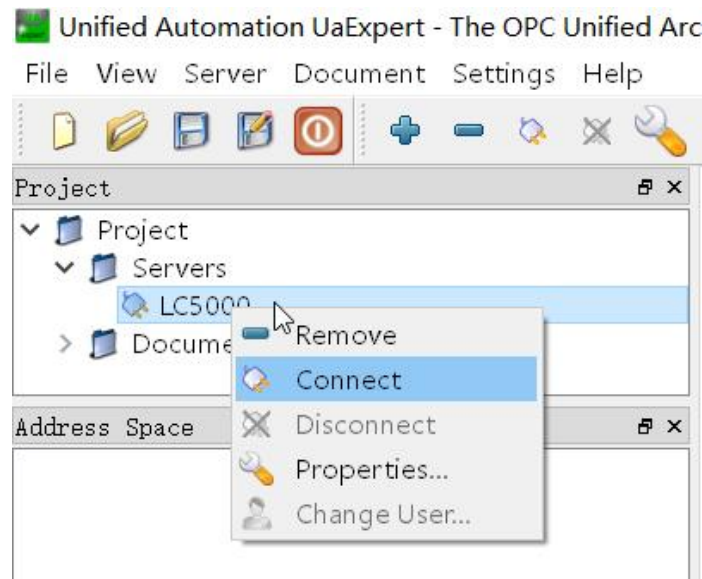


图 13.66 连接 PLC

连接成功后，找到 Globalvars 下面的变量，核对是否与我们之前 PLC 配置的 OPC 通信变量单一致。

5) 读 PLC 的 OPC 通信变量

将图 13.67 左侧树中的变量拖入“data Access View”，可访问 PLC 变量。在 PLC 中修改变量的值，可发现图 13.67 中的对应变量同步变化。

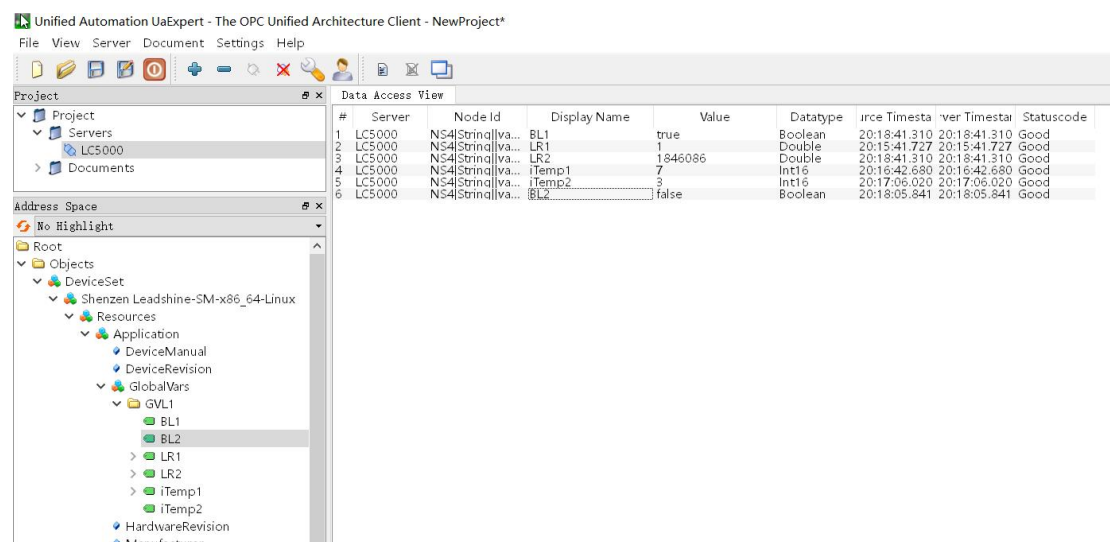


图 13.67 读 PLC 的 OPC 通信变量

6) UaExpert 软件通过 OPC UA 写 PLC 的变量

在图 13.68 中双击 value，填入写入值，回车后，值被写入 PLC。

Data Access View									
#	Server	Node Id	Display Name	Value	Datatype	urce Timesta	ver Timesta	Statuscode	
1	LC5000	NS4 String va...	BL1	false	Boolean	1:15:14.585	1:15:14.585	Good	
2	LC5000	NS4 String va...	LR1	1	Double	1:12:12.406	1:12:12.406	Good	
3	LC5000	NS4 String va...	LR2	888	Double	1:15:05.821	1:15:05.821	Good	
4	LC5000	NS4 String va...	iTemp1	1	Int16	1:12:12.406	1:12:12.406	Good	
5	LC5000	NS4 String va...	iTemp2	3	Int16	1:12:12.406	1:12:12.406	Good	
6	LC5000	NS4 String va...	BL2	false	Boolean	1:12:09.065	1:12:09.065	Good	

图 13.68 向 PLC 传送数据

可用编程软件在线核对写的值是否正确，如图 13.69 所示，写值成功。

Application [Device: PLC 逻辑]			
Device.Application.GVL1			
表达式	类型	值	准备值
LR1	LREAL	1	
LR2	LREAL	888	
LR3	LREAL	3	
BL1	BOOL	TRUE	
BL2	BOOL	FALSE	
iTemp1	INT	1	
iTemp2	INT	3	

图 13.69 在 PLC 上核对结果

13.6. Ethernet/IP 通信

Ethernet/IP 是一个面向工业自动化应用的工业应用层协议。它建立在标准是由 ODVA(Open Device Net Vendors Association)和 Control Net International 所推出的一种与 TCP/IP 以太网相结合的工业以太网标准。

Ethernet/IP 通讯协议是适合工业环境应用的协议体系（IP 是“Industrial Protocol”的简称），基于传统的以太网协议和标准的 TCP/IP 协议，可以实现工业设备之间应用信息的高效交换。EtherNet/IP 应用层协议使用标准的面向对象的 CIP 协议。

大型 PLC 产品 LC 系列和中型 PLC 产品的 LAN 网口均支持 Ethernet/IP 通信协议。下面通过一个简单的例程，介绍 EthernetIP 的功能和使用方法。

将一个 SMC632E 控制器作为主站（Scanner），一个 PMC600 控制器作为从站

(Adapter)。两个控制器的 IP 地址分别为 192.168.1.11 和 192.168.1.3，并用交换机将这两个控制器用网线连接在一个网络中。

1) 点击登录，下载程序，然后运行程序，如图 13.77 所示。

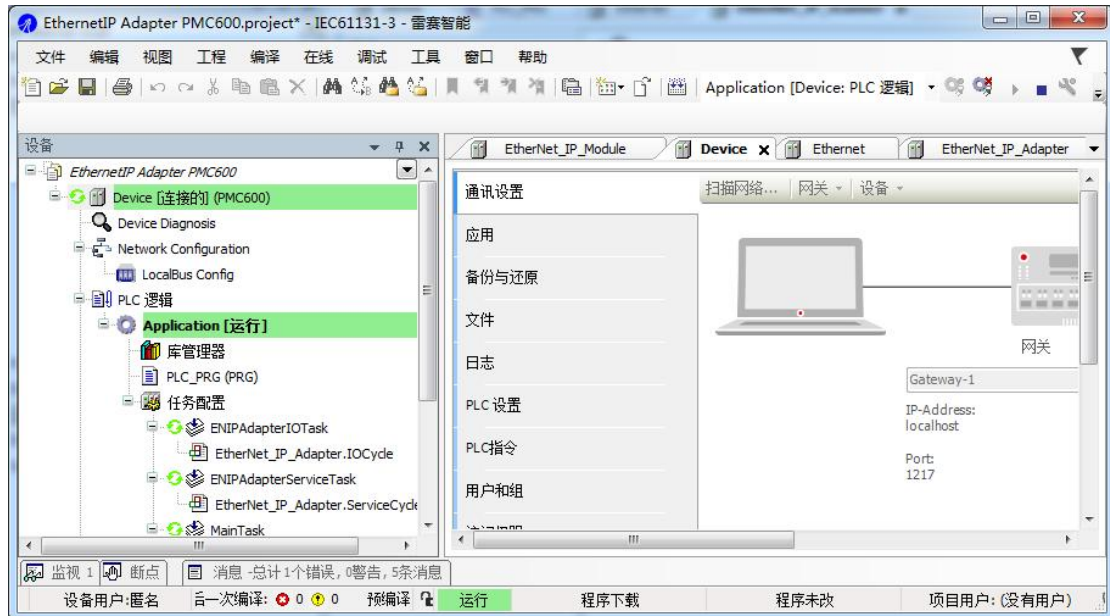


图 13.77 运行程序

13.6.1. 配置从站

2) 打开从站的程序，添加以太网适配器，如图 13.70 所示。

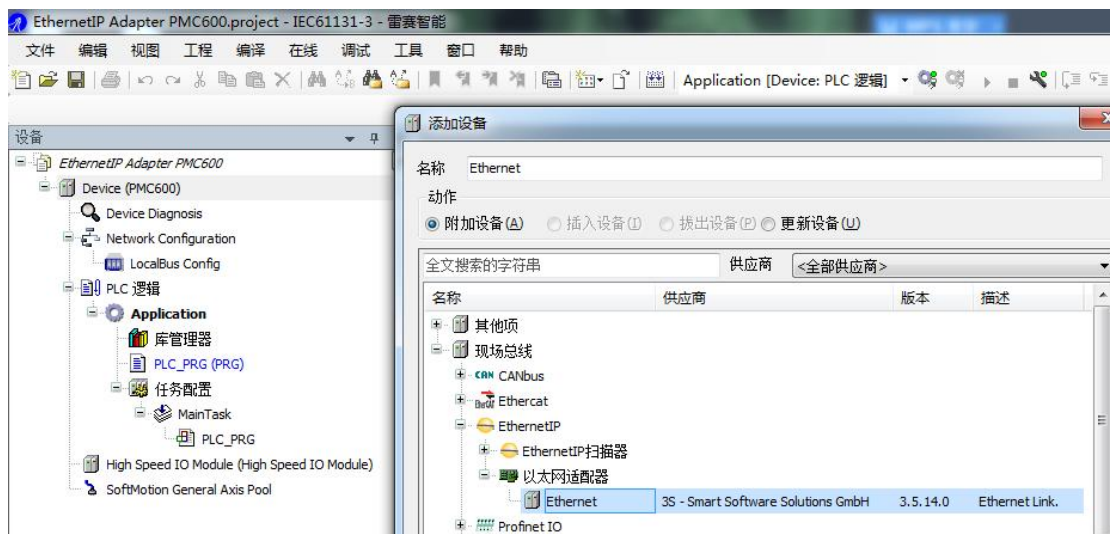


图 13.70 添加以太网适配器

3) 鼠标双击 Ethernet 设备，选择所使用的网口，这里用的是 192.168.1.3，然后点击确认，如图 13.71 所示。

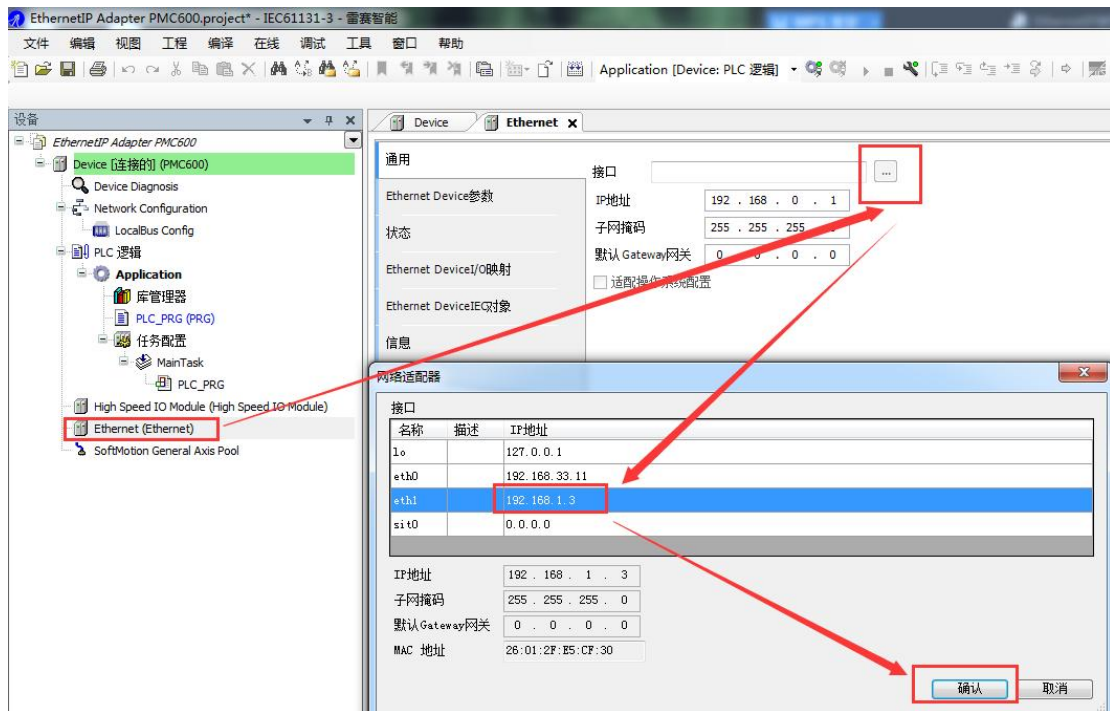


图 13.71 设置 IP 地址

4) 在 Ethernet 设备上右击，添加“Ethernet/IP Adapter”设备，如图 13.72 所示；然后在 Adapter 上添加“Ethernet_IP_Module”设备，如图 13.73 所示。

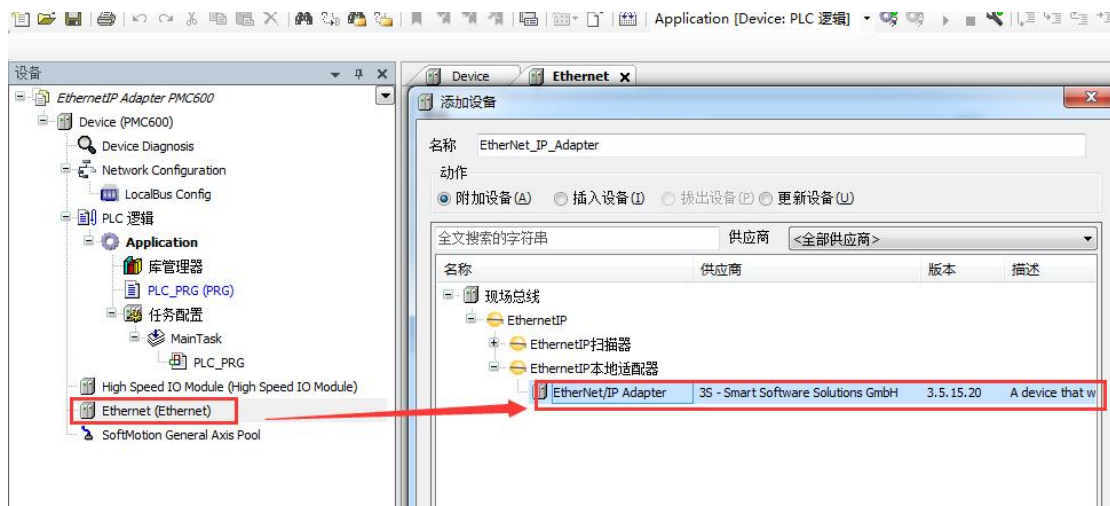


图 13.72 添加“Ethernet/IP Adapter”设备

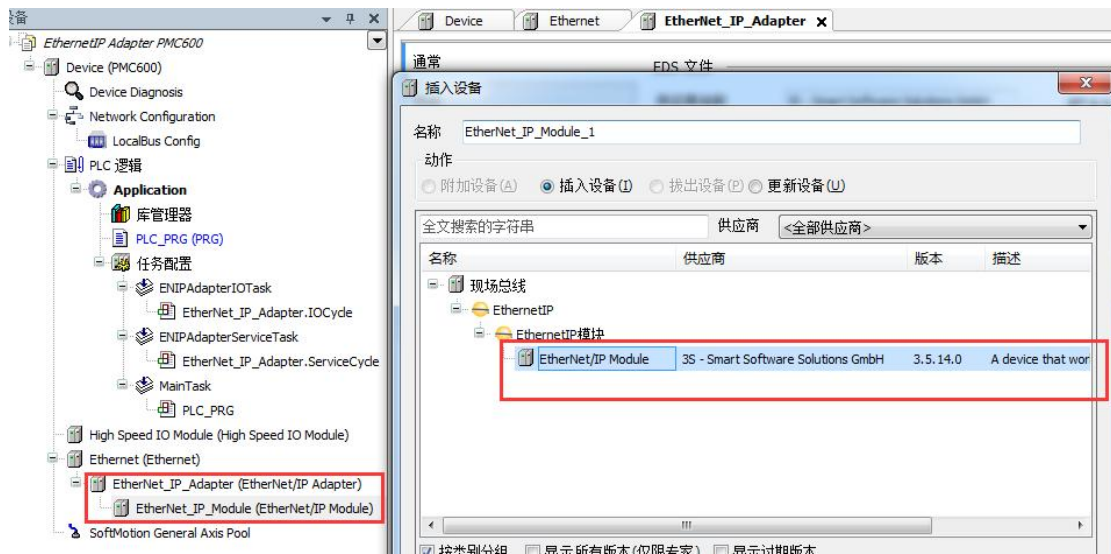


图 13.73 添加“Ethernet_IP_Module”设备

5) 设置“Ethernet_IP_Module”为输入模块“Byte Input Module”，如图 13.74 所示；设置循环模式为“使能 2”，如图 13.75 所示。

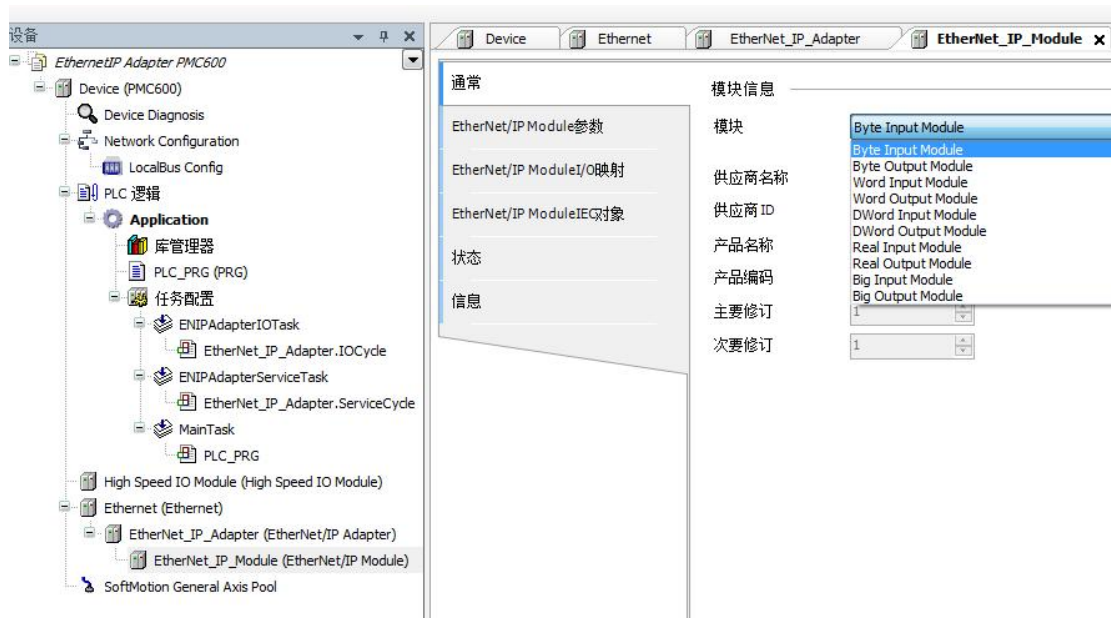


图 13.74 设置输入模块为“Byte Input Module”

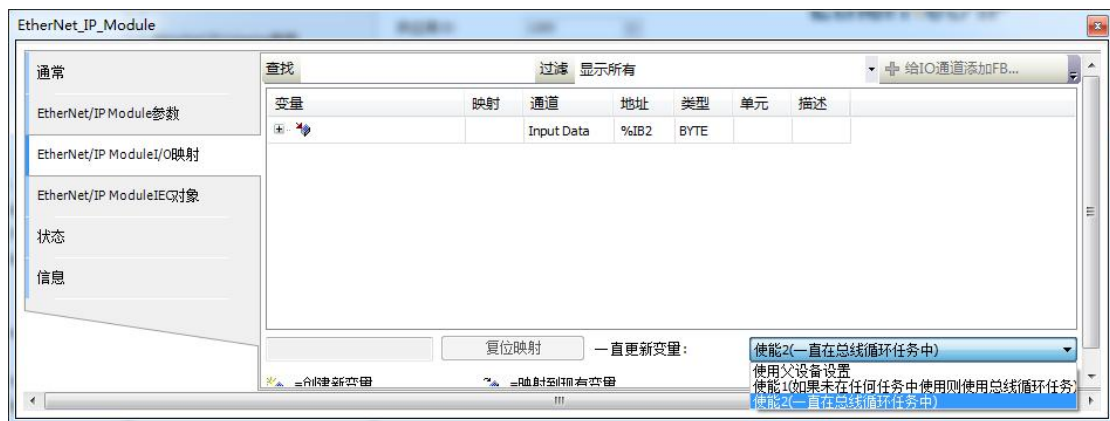


图 13.75 设置循环模式为“使能 2”

6) 回到“Ethernet/IP Adapter”，点击图 13.76 中的按钮，将设备安装到设备库中。

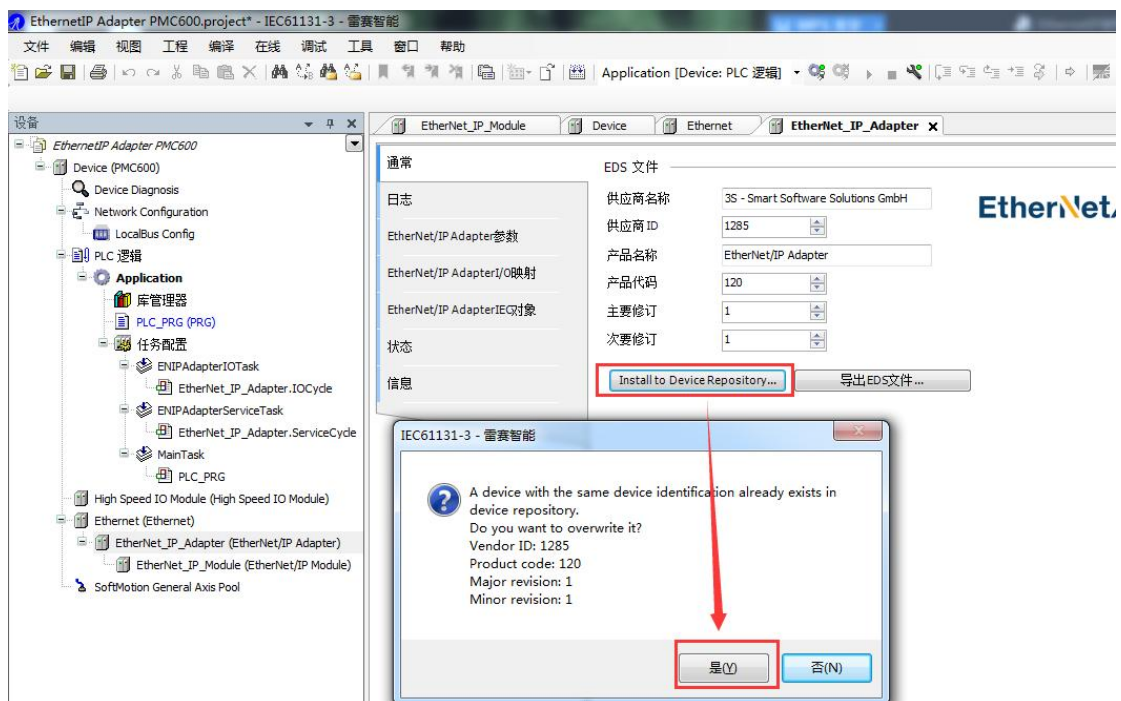


图 13.76 将设备安装到设备库中

13.6.2. 配置主站

1) 和配置从站一样，配置主站时，先添加“Ethernet”设备，如图 13.78 所示。

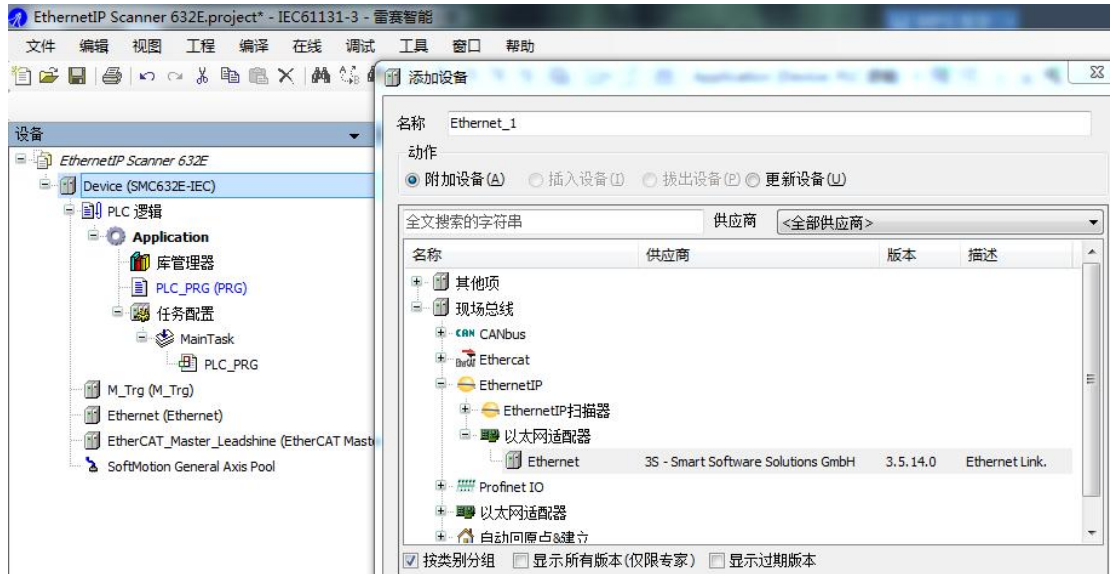


图 13.78 添加“Ethernet”设备

2) 鼠标双击 Ethernet 设备，选择所使用的网口，这里用的是 192.168.1.11，然后点击确认，如图 13.79 所示。

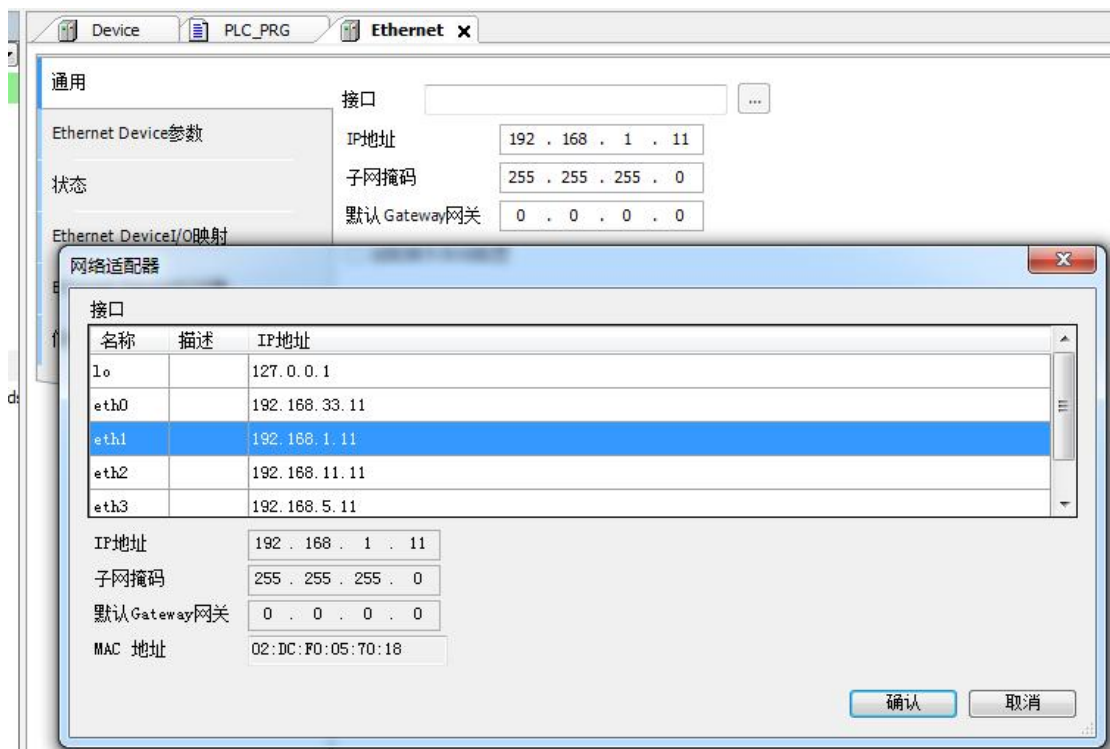


图 13.79 设置 IP 地址

3) 添加设备“Ethernet/IP Scanner”，如图所示 13.80 所示。

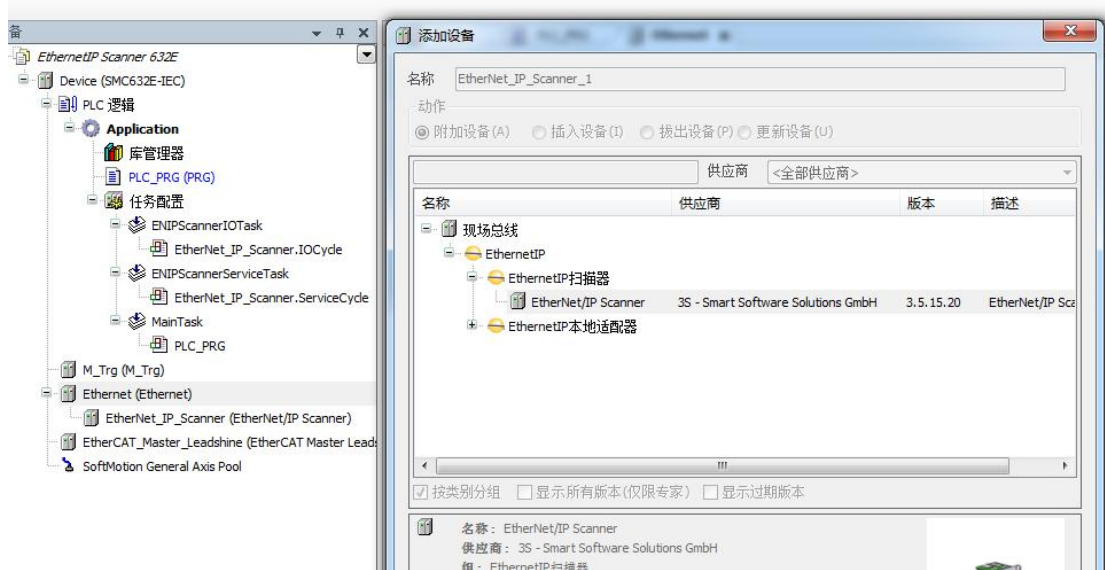


图 13.80 添加设备“Ethernet/IP Scanner”

4) 右击“Ethernet/IP Scanner”，选择扫描，等待一下，将扫到的从站适配器设备复制到工程之中。如图 13.81 所示。

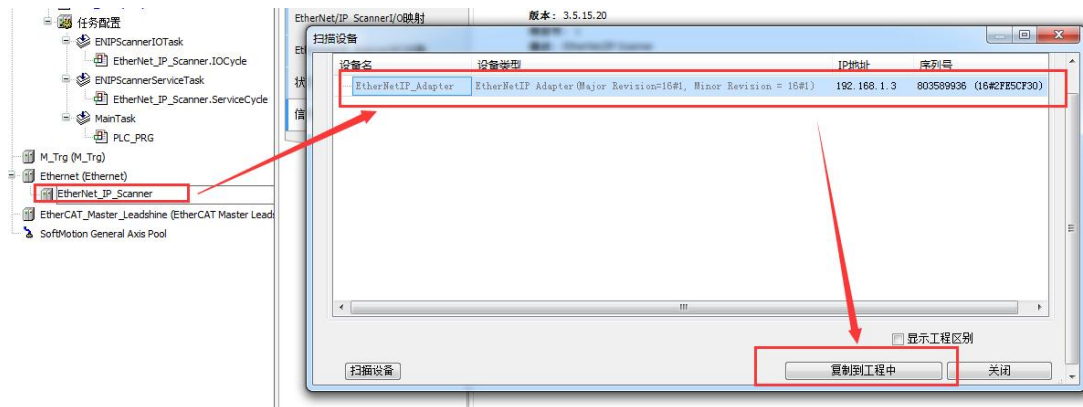


图 13.81 扫描设备

5) 设置从站适配器设备的扫描方式为“循环 2”，如图 13.82 所示。



图 13.82 设置扫描方式为“循环 2”

6) 点击登录，下载程序，然后运行程序，如图 13.83 所示。

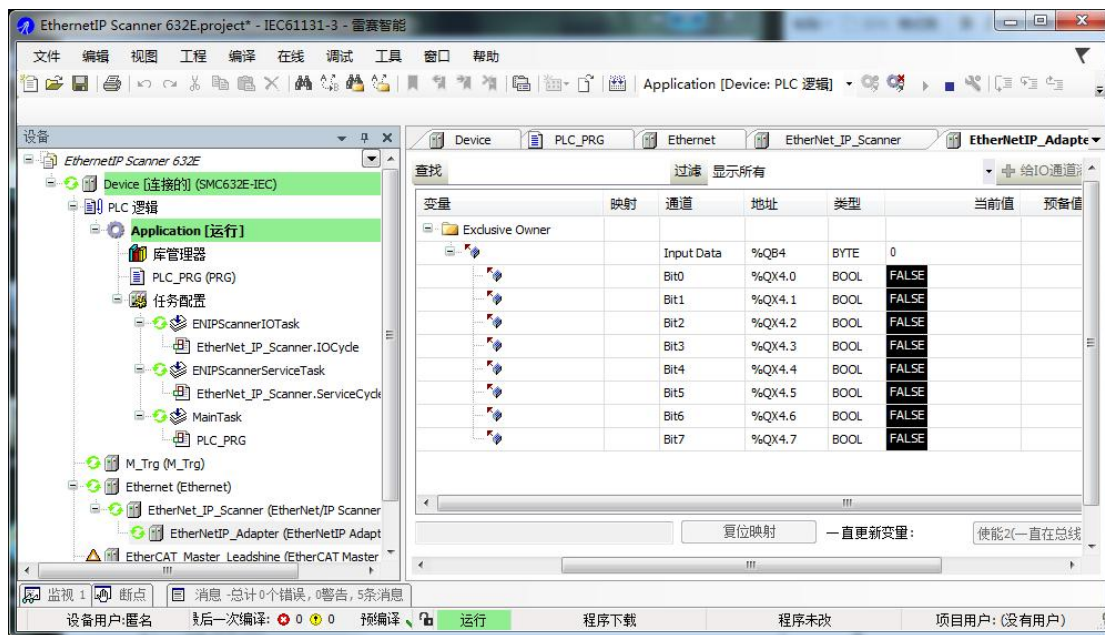


图 13.83 运行程序

13.6.3. 实现通讯功能

如图 13.79 所示，到“EtherNetIP_Adapter”界面下的“EtherNet/IPI/O 映射”栏，将几个变量设置为 True；再回到从站设备，可以看到从站的变量也变成了 True，如图 13.84 所示。

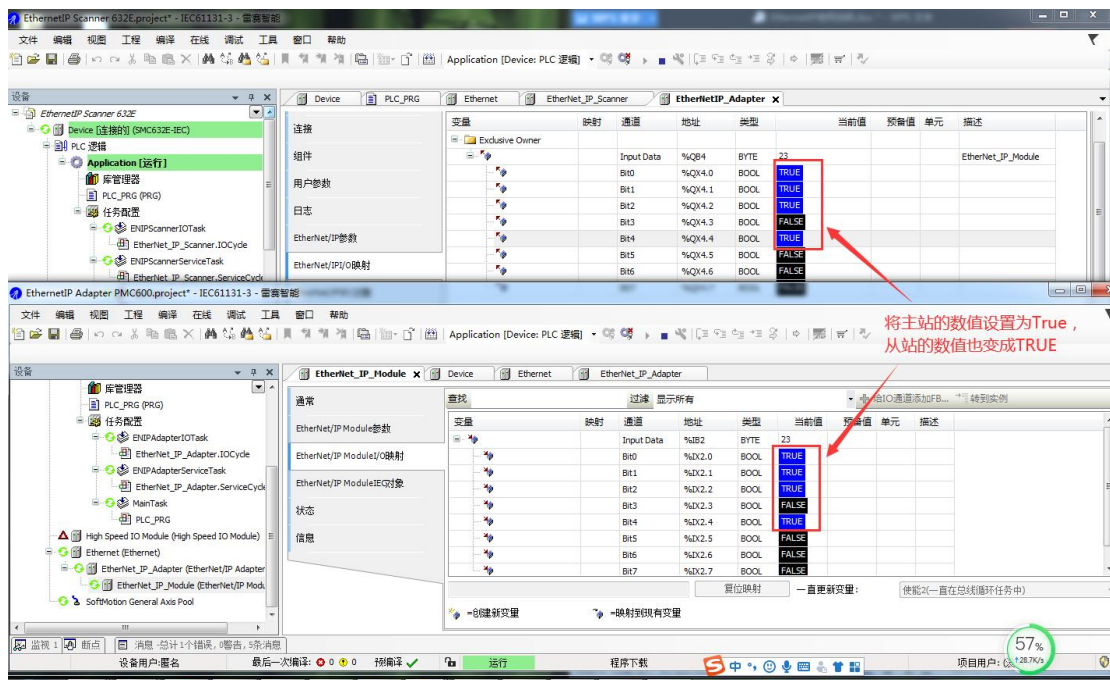


图 13.84 运行结果

13.6.4. Ethernet/IP 主从站配置

Ethernet/IP 主从站的配置采用可视化方式，双击网络组态中的“通讯配置”，如图 13.85 所示，单击 PLC 设备，弹出所支持的通信协议及主/从站的使能窗口。在“Ethernet IP”下勾选“扫描器”，使能后，变为红色，配置为主站。

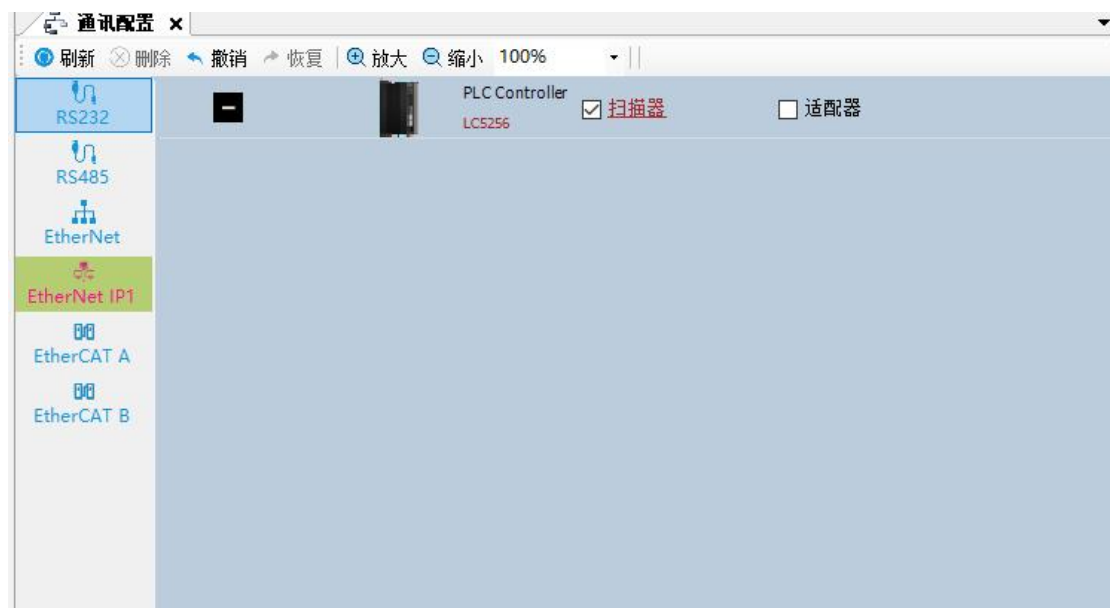


图 13.85

同样的方法，在“Ethernet IP”下勾选“适配器”，使能后，变为红色，配置为从站。

第十四章 EtherCAT 通讯

EtherCAT (Ethernet for Control Automation Technology) 是一个开放架构、是以太网为基础的现场总线系统。

由于 EtherCAT 通讯利用了逻辑环网结构和全双工快速以太网而又有实际环网结构，“主站时钟”可以简单而精确地确定对每个“从站时钟”的运行补偿。分布时钟基于该值进行调整，在网络范围内实现信号抖动很小、精度非常高的时钟。与其他现场总线相比，EtherCAT 具有性能好、设备同步精度高、拓扑结构灵活、应用容易、成本低等特点。

当前，越来越多的设备使用 EtherCAT 总线进行通讯。雷赛 LC/PMC600 系列 PLC 支持最多 128 个 EtherCAT 从站，用户拓展运动控制系统的空间很大。

14.1. EtherCAT 主站配置

14.1.1. 添加设备描述文件

LeadSys 中已经预添加了雷赛公司的 EtherCAT 从站设备，用户如使用雷赛公司的产品，可直接跳过该步骤。

用户在使用软件中尚未添加设备描述文件的 EtherCAT 从站时，需要自行将对应的描述文件添加到软件中，具体的操作步骤如下：

- 1) 打开 LeadSys 软件，选择菜单栏中的“工具”选项卡，在二级菜单中选择“设备储存库”，如图 14.1 所示。

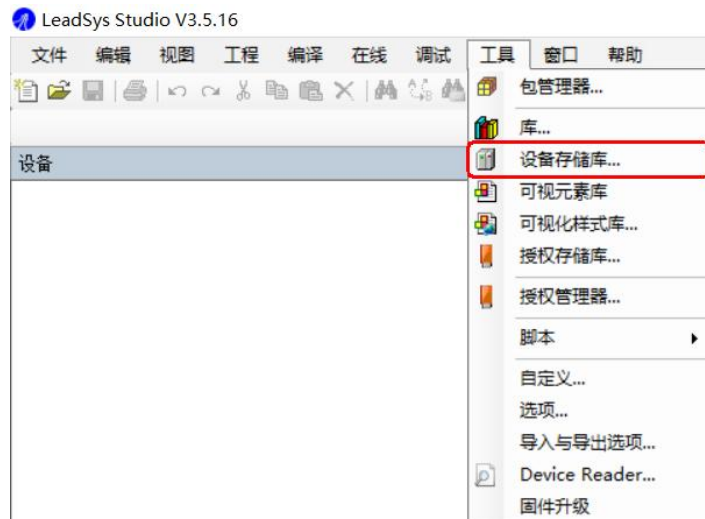


图 14.1 选择“设备存储库”

2) 在弹出的窗口中点击“安装”按钮，如图 14.2 所示。

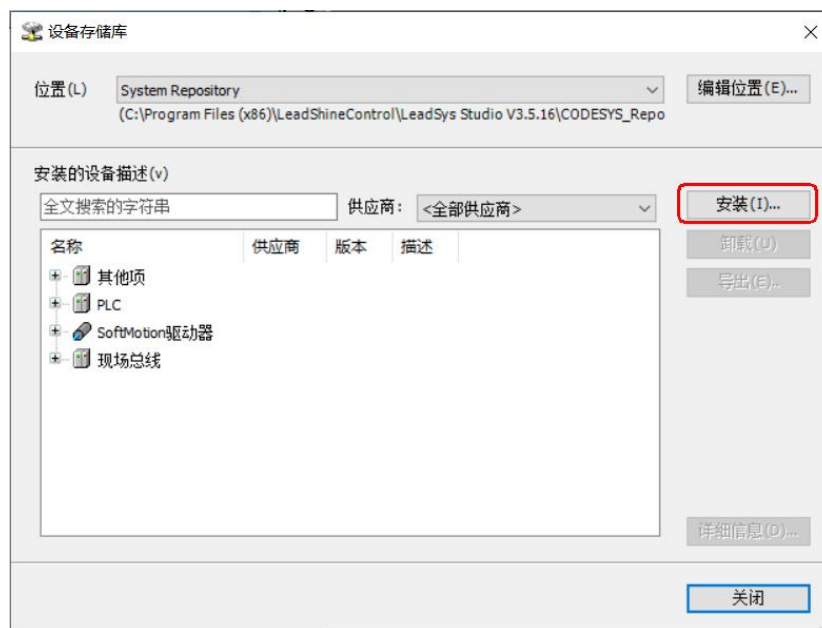


图 14.2 安装“设备存储库”

3) 在弹出窗口中找到存放设备描述文件的路径。

注意：此次要将文件类型选择为 EtherCAT XML 设备描述配置文件，如果选择为其他文件类型，会导致无法正常选取 XML 文件，找到对应的 XML 后，点击“打开”按钮，如图 14.3 所示。

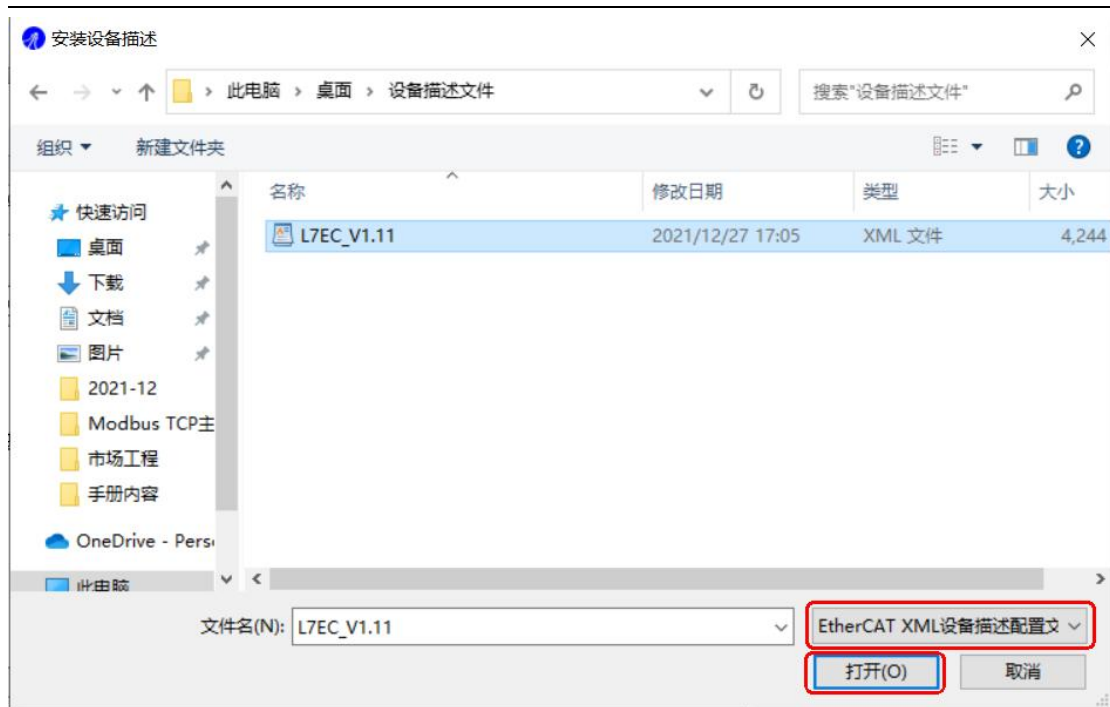


图 14.3 选择“设备描述文件”的路径

- 4) 打开文件之后，软件会自动导入设备，此时可观察信息输出框，确认安装完成后，点击“关闭”按钮即可。

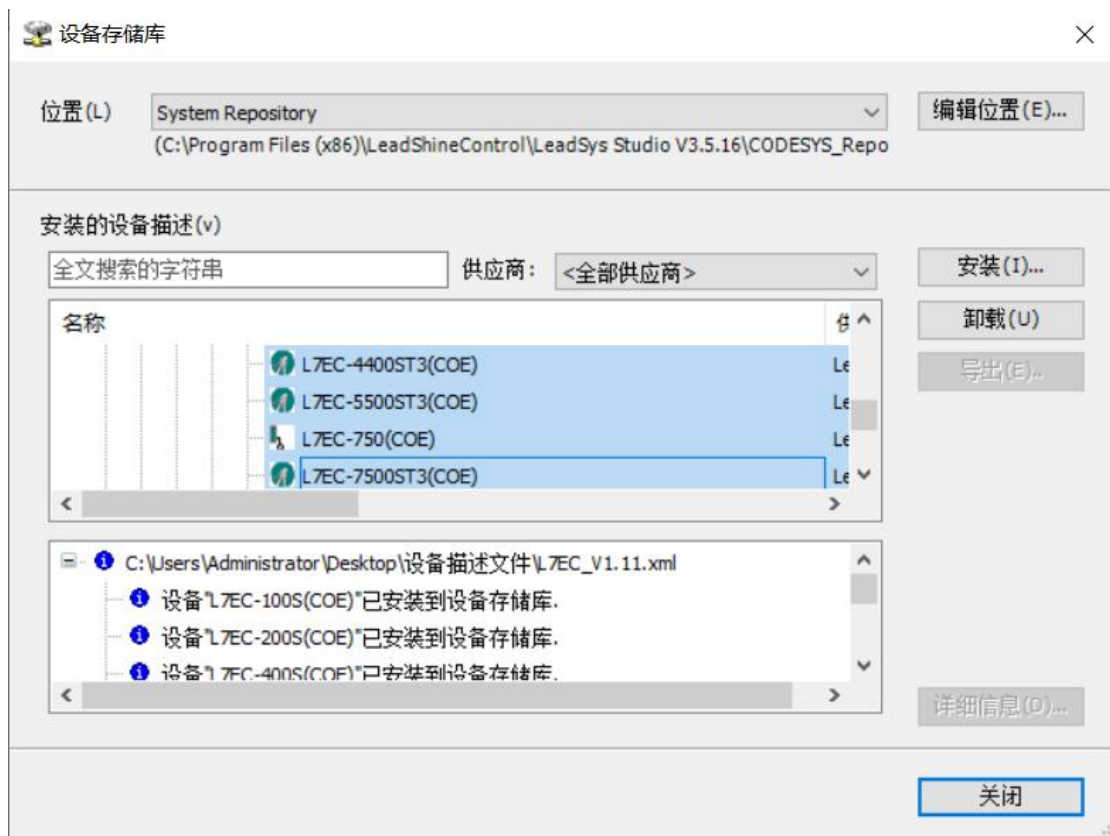


图 14.4 将“设备描述文件”导入设备

14.1.2. 从站组态

在 LeadSys 软件中可以通过“扫描添加”和“手动添加”两种方式来实现 EtherCAT 从站的组态。

1) 扫描添加

通过扫描添加 EtherCAT 从站时，首先要保证工程中使用到从站设备硬件正确的连接在 PLC 上，只有在能够正常通讯时，PLC 才能扫描到 EtherCAT 从站设备。具体操作步骤如下：

a) 新建或打开一个工程，型号与实际连接的 PLC 型号一致，如图 14.5 所示；

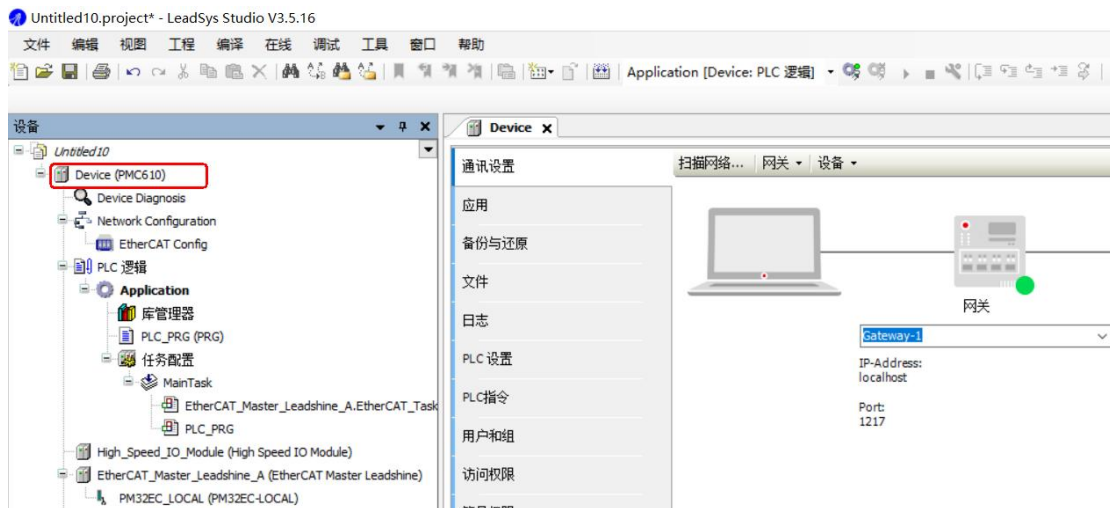


图 14.5 确定 PLC 型号

b) 添加 EtherCAT 主站；

鼠标双击设备树下的“通讯配置”，在打开的界面中点击 PLC，会弹出勾选设备组态的窗口，勾选“EtherCAT 主站”即可。

勾选后，设备树下会自动添加“EtherCAT_Master_Leadshine_A”设备。

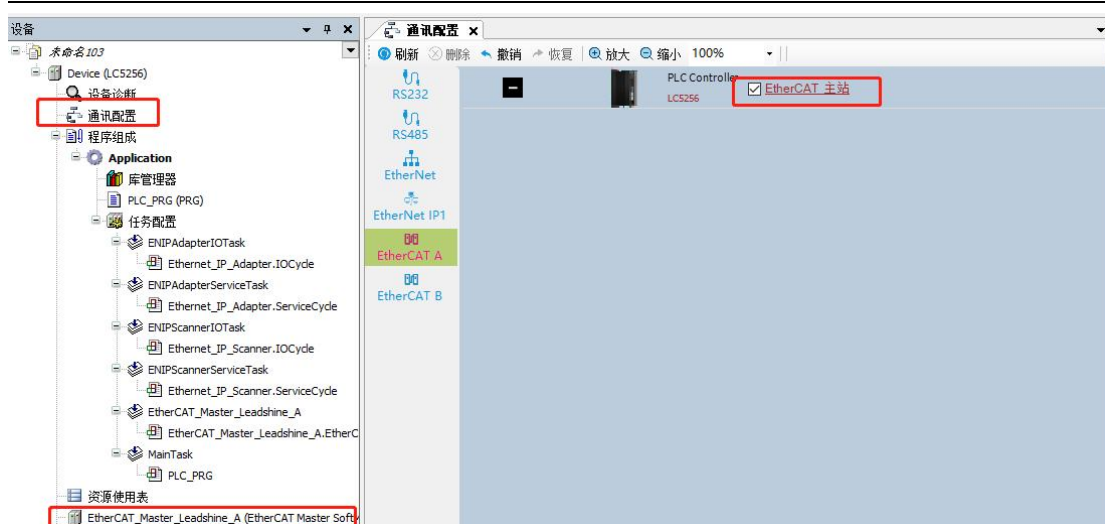


图 14.6 添加 EtherCAT 主站

c) 登录，下载程序至 PLC；

在“Device”→“通讯设置”→“扫描网络”里选择要连接的设备，选择后点击确定，如图 14.7 所示。

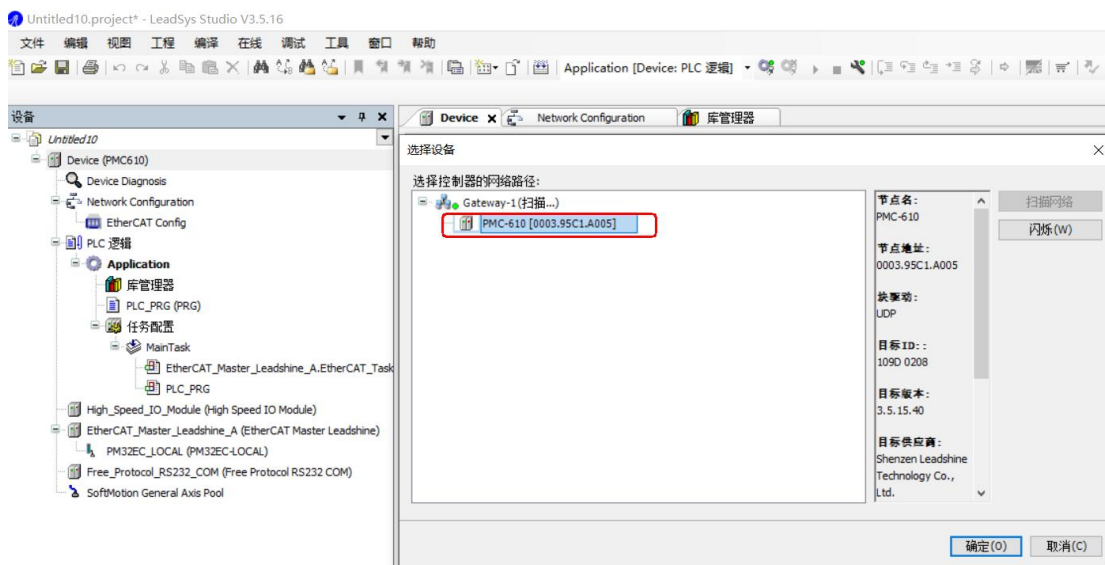


图 14.7 选择已连接的设备

选择完成后，点击快捷工具栏的“”按钮，登录 PLC，同时将程序下载到 PLC 中。

d) 扫描 EtherCAT 从站设备；

用鼠标右键点击设备树中的 EtherCAT 主站设备，在打开的提示框中选“扫描设

备”，可打开扫描设备窗口。如图 14.8 所示。

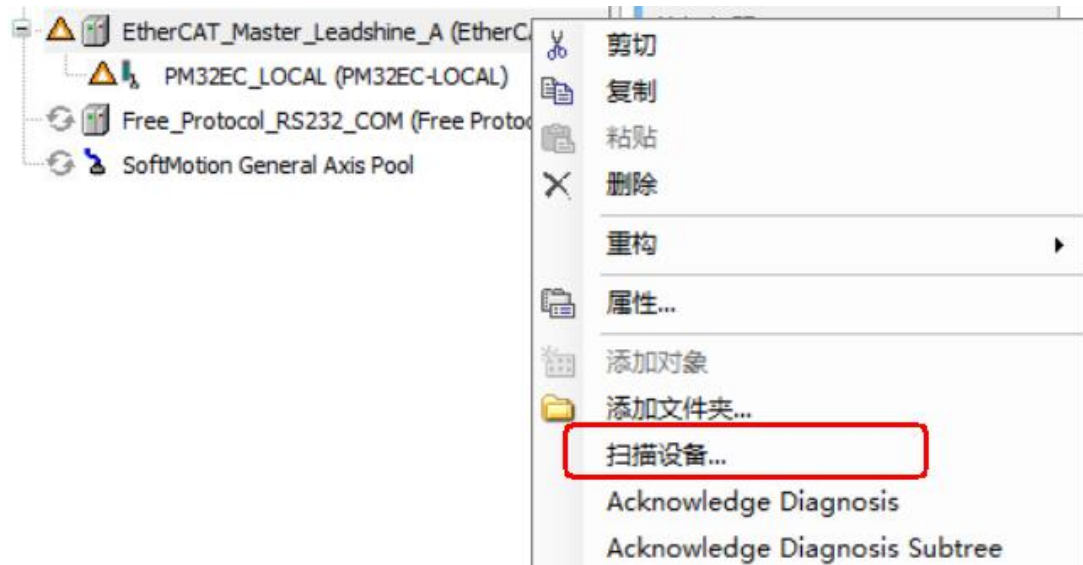


图 14.8 选择“扫描设备”

如图 14.9 所示，点击窗口中的“扫描设备”按钮，PLC 自动扫描连接的 EtherCAT 从站设备，扫描出来的设备会显示在窗口中。

确认无误后，点击“复制所有设备到工程中”按钮，完成 EtherCAT 从站的添加，并关闭该窗口。



图 14.9 将扫描到的设备复制到工程中

添加设备后，可在设备树下找到对应的从站设备，如图 14.10 所示。

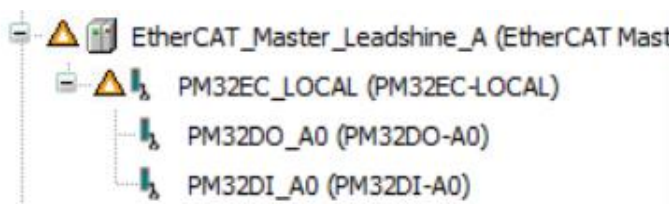


图 14.10 设备树中显示的设备

2) 手动添加

如果用户在编程时，不具备与实际硬件连接的条件，也可以通过手动添加的方式，来实现设备的组态。

具体操作步骤如下：

- a) 新建或打开一个工程，型号与实际连接的 PLC 型号一致，如图 14.11 所示；

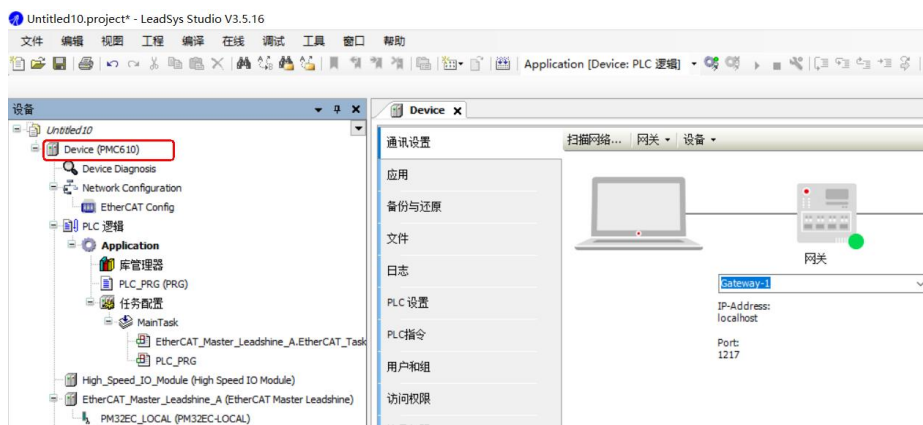


图 14.11 设置 PLC 的型号

- b) 添加 EtherCAT 主站；

双击设备树下的“通讯配置”，在打开的界面中点击 PLC，会弹出勾选设备组态的窗口，勾选“EtherCAT 主站”即可。

勾选后，设备树下会自动添加“EtherCAT_Master_Leadshine_A”设备。如图 14.12 所示。

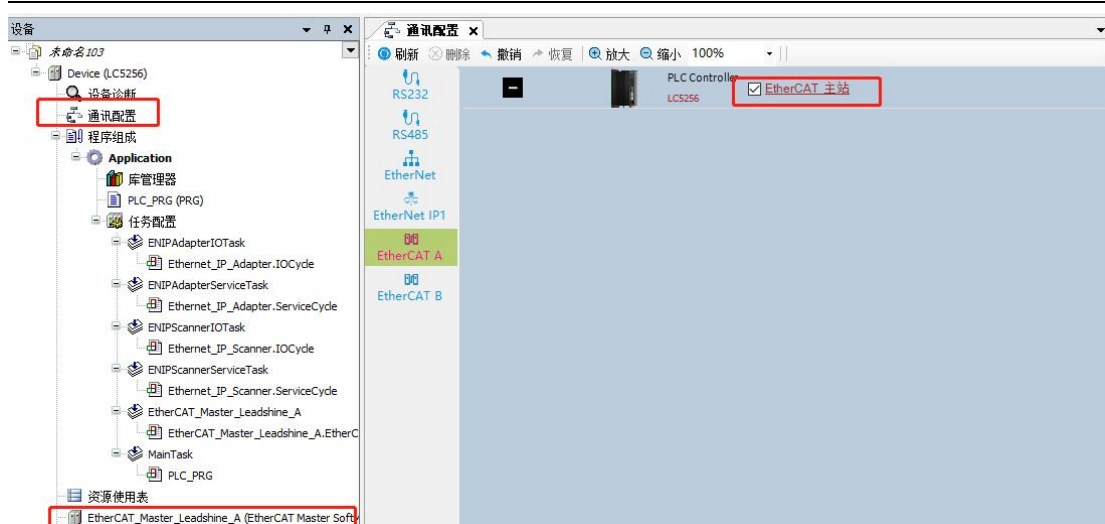


图 14.12 添加 EtherCAT 主站

- c) 用户根据自己的需要，在“网络设备”中选择设备，双击可将对应的设备添加到组态中，如图 14.13 所示，添加 L7EC_100 和 L7EC_200 两个驱动器；

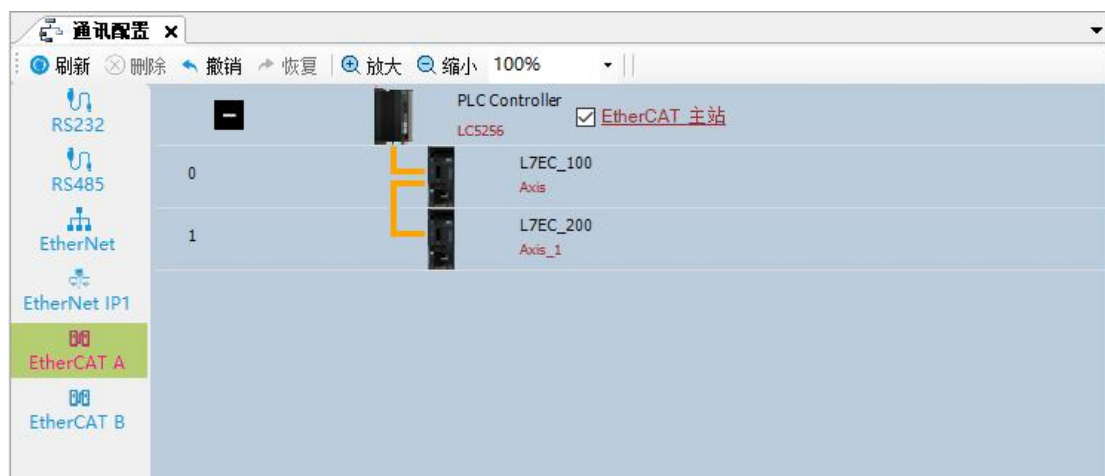


图 14.13 手动添加设备

14.1.3. 主站设置

EtherCAT 主站设置是设置 EtherCAT Master 的参数。如图 14.14 所示，要设置时钟周期和同步偏移。

- ① 周期: EtherCAT 数据帧发送间隔, 该时间需要与 EtherCAT 任务的循环时间相同。
- ② 同步偏移: EtherCAT 任务相对于从站的 Sync0 中断的相对偏移的百分比, 该值默认为 30%, 无特殊情况请勿修改。



图 14.14 设置主站的周期和同步偏差

在 EtherCAT 通讯的 DC 模式中, 从站在解析主站传输过来的数据后, 还需要在 DC 中断中处理数据。由于主站数据传输到每个从站的时间长短不一, 所以为了保证数据的同步, 在数据帧和同步中断之间必须有足够的时间给从站接收和处理数据, 这段时间即为偏移时间。

14.1.4. EtherCAT 主站常用参数

EtherCAT 主站常用参数如表 14.1 所示。在使用时, 需要加上主站名, 如主站名为 EtherCAT_Master_Leadshine_A, 则需要重启主站时, 需要触发的变量为 EtherCAT_Master_Leadshine_A.xRestart。

表 14.1 EtherCAT 主站常用参数

变量名	功能
xRestart	上升沿触发，触发后重启 EtherCAT 主站
xConfigFinished	EtherCAT 主站配置完成
xDistributedClockInSync	EtherCAT 分布时钟同步
xError	EtherCAT 主站存在错误

14.2. EtherCAT 从站配置

14.2.1. 常规设置

1) EtherCAT 地址

EtherCAT 从站的组态地址，是指从站设备在 LeadSys 设备树中的顺序地址，从 1001 开始，顺序递增，如图 14.15 所示。用户可以在从站的设置界面中，找到该从站的 EtherCAT 地址。

在调用读写 sdo 功能块时，会使用到该地址，如图 14.16 所示。

The screenshot shows the 'EtherCAT' configuration window. On the left is a sidebar with '通用' (General) selected. The main area is divided into '地址' (Address) and '额外' (Additional) sections. In the '地址' section, '自动增量地址' is set to 0, and 'EtherCAT地址' is set to 1001 and highlighted with a red rectangle. The '额外' section has checkboxes for '启用专家设置' and '可选的'. Below this is the '分布时钟' (Distributed Clock) section with a dropdown set to 'FreeRun/SM-Synchron' and a '2000' value for '同步单元周期(μs)'. The 'Sync0' and 'Sync1' sections each have radio buttons for '启用 Sync0'/'启用 Sync1', '同步单元周期', and '用户定义', along with input fields for '周期时间(μs)' and '偏移时间(μs)'.

图 14.15 EtherCAT 从站地址

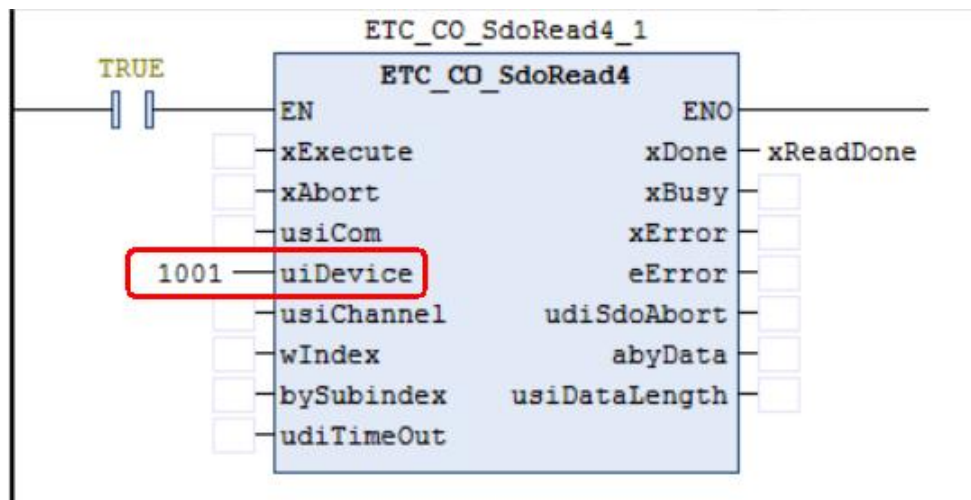


图 14.16 SdoRead 功能块

2) 分布式时钟

该选项用于设置从站的同步运行模式。同步模式有 3 种：自由运行模式（FreeRun）、同步于输入输出事件的运行模式（SM-Synchron）和同步于分布时钟的运行模式（DC-Synchron）。

选择的从站不同，同步模式支持的选项会有不同。用户一般无需修改该选项，只要保证 DC-Synchron 运行模式下的同步单元周期与 EtherCAT 任务周期保持一致即可。如图 14.17 所示。



图 14.17 分布式时钟的参数

3) 站点别名

在使用 EtherCAT 地址时，当实际设备的连接顺序与组态顺序不一致，总线无法正常运行，如果用户希望实际从站连接顺序不受组态顺序的影响，可以使用别名功能。

只有启用专家设置后才可以设置从站的站点别名。要正确使用站点别名功能首先需要设置从站的站点别名，设置站点别名的步骤如下。

- a) 正确建立组态，不勾选“启用专家设置”，下载程序后等待从站正常启动。如图 14.18 所示。

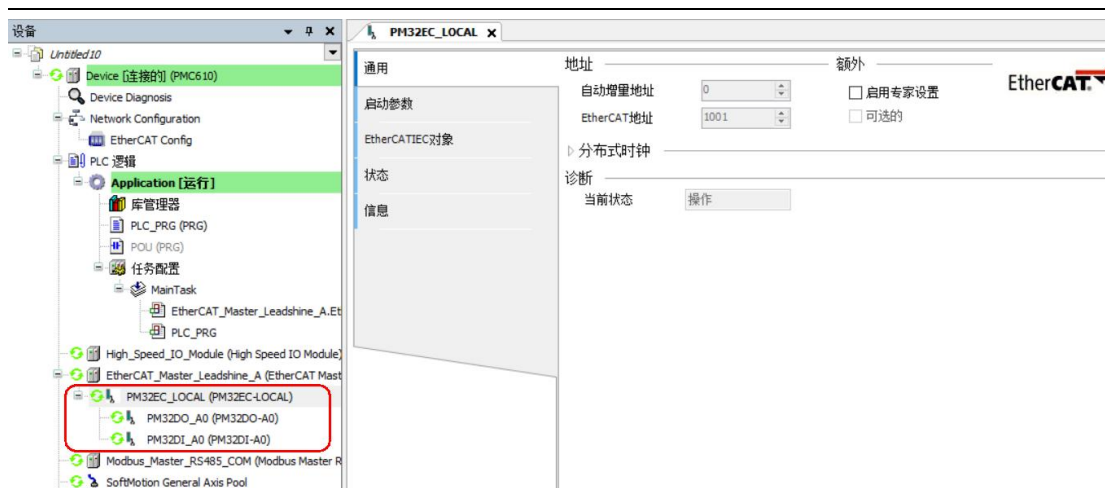


图 14.18 不勾选“启用专家设置”

b) 勾选“启用专家设置”，选择“配置的站别名”，如图 14.19 所示；



图 14.19 选择“配置的站别名”

c) 根据需要写入站点别名，例如：写入“1”，写入值后，点击“写入 EEPROM”，如图 14.20 所示；



图 14.20 写入站点别名

d) 根据提示重启设备，如图 14.21 所示；

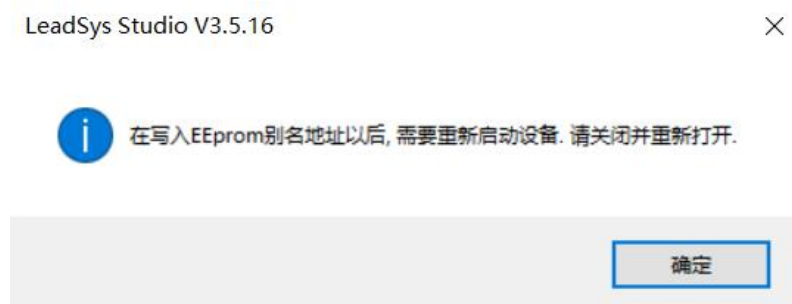


图 14.21 重新启动设备

e) 重启后需要重新登录，下载 PLC 程序；

f) 下载成功后可看到从站的实际地址已经成功写入，站点别名功能启用，如图 14.22 所示。



图 14.22 站别名生效

14.2.2. 过程数据（PDO）

1) 过程数据

如图 14.23 所示，该界面会显示从站描述文件自带的输入输出 PDO，用户可以根据自身的需要进行选择，勾选后即启用该组 PDO。

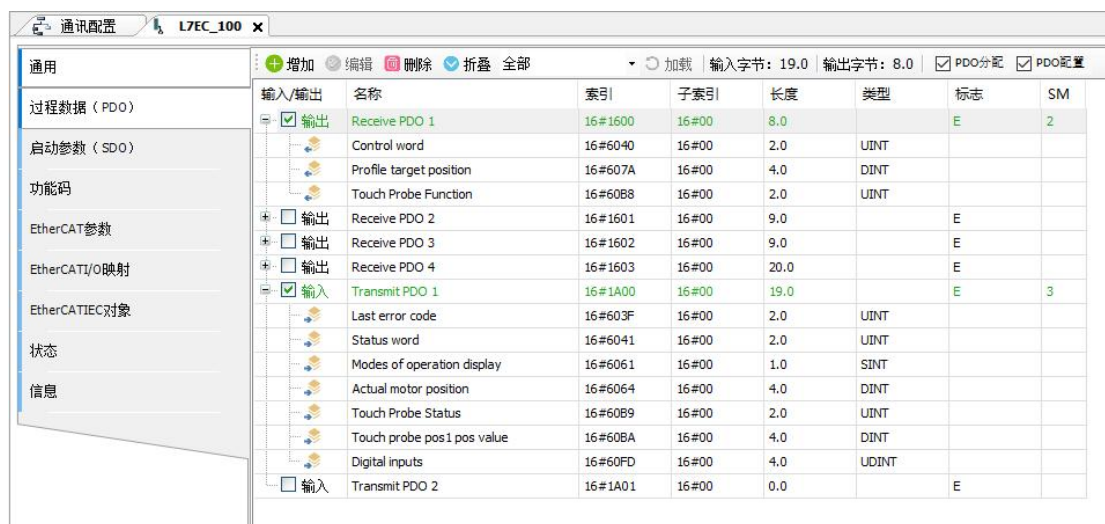


图 14.23 从站的输入输出 PDO

编辑 PDO 的具体操作步骤如下。

a) 选择需要修改的 PDO 组，如图 14.25 所示；

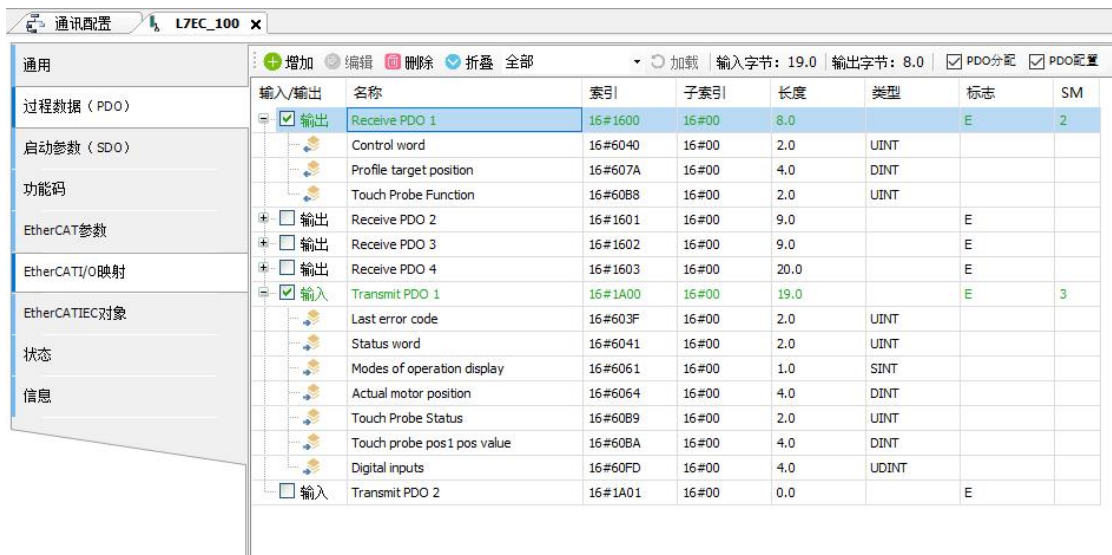


图 14.25 选择需要修改的 PDO 组

- b) 如图 14.26 所示，用户可点击“增加”、“编辑”、“删除”、按钮，对 PDO 的内容进行编辑，参见表 14.2；

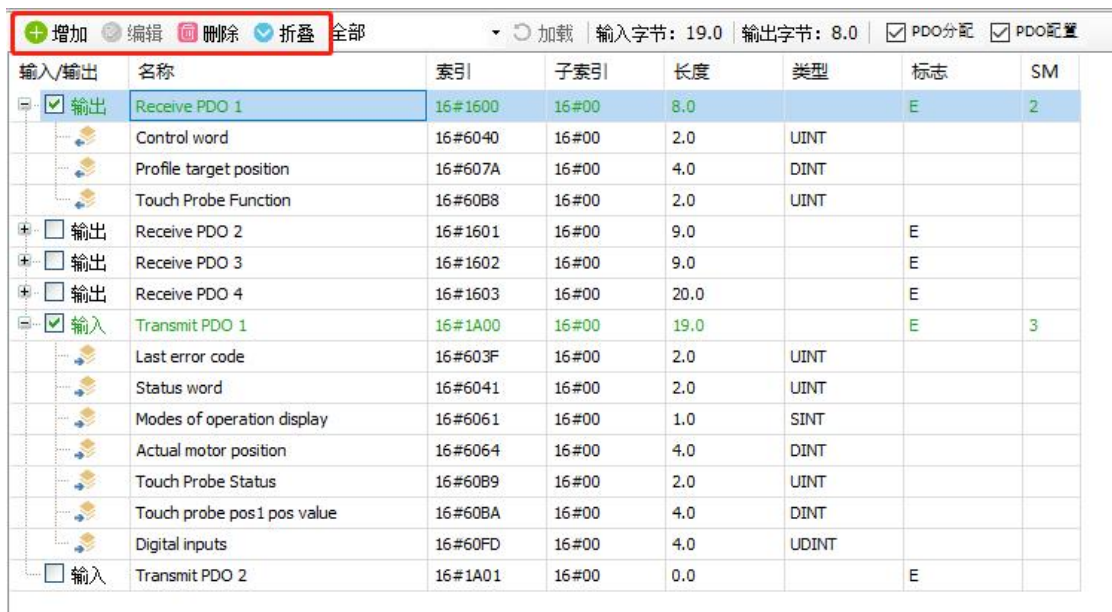


图 14.26 PDO 内容编辑框

表 14.2 编辑功能

选项	功能
添加	添加一条 PDO 配置

编辑	编辑当前选中的 PDO 配置
删除	删除当前选中的 PDO 配置

c) 在界面勾选需要启用的 PDO 组，如图 14.27 所示。



输入/输出	名称	索引	子索引	长度	类型	标志	SM
☒ 输出	Receive PDO 1	16#1600	16#00	8.0		E	2
	Control word	16#6040	16#00	2.0	UINT		
	Profile target position	16#607A	16#00	4.0	DINT		
	Touch Probe Function	16#60B8	16#00	2.0	UINT		
☐ 输出	Receive PDO 2	16#1601	16#00	9.0		E	
☐ 输出	Receive PDO 3	16#1602	16#00	9.0		E	
☐ 输出	Receive PDO 4	16#1603	16#00	20.0		E	
☒ 输入	Transmit PDO 1	16#1A00	16#00	19.0		E	3
	Last error code	16#603F	16#00	2.0	UINT		
	Status word	16#6041	16#00	2.0	UINT		
	Modes of operation display	16#6061	16#00	1.0	SINT		
	Actual motor position	16#6064	16#00	4.0	DINT		
	Touch Probe Status	16#60B9	16#00	2.0	UINT		
	Touch probe pos1 pos value	16#60BA	16#00	4.0	DINT		
	Digital inputs	16#60FD	16#00	4.0	UDINT		
☐ 输入	Transmit PDO 2	16#1A01	16#00	0.0		E	

图 14.27 PDO 分配

14.3. 启动参数（SDO）

启动参数可以将从站的一些参数在从站处于 PreOP 状态时通过写 SDO 的形式写入从站，如图 11.28 所示。



行	索引: 子索引	名称	值	位长度	是否下载	有错误退出	有错误跳行	下一行	注释
1	16#6060: 16#00	Modes of Operation	8	8	☒	☐	☐	0	Modes of Operation

图 14.28 启动参数

在实际使用过程中，用户可根据自己的需求添加启动参数。

以 L7EC_100 伺服驱动器为例，如需要在启动时修改回原点模式为 19，操作过

程如下。

1) 点击“启动参数”界面的“添加”按钮，如图 14.29 所示；



图 14.29 “启动参数”中的“添加”按钮

2) 在弹出的窗口中找到对象字典 16#6098，即回原点模式，如图 14.30 所示。

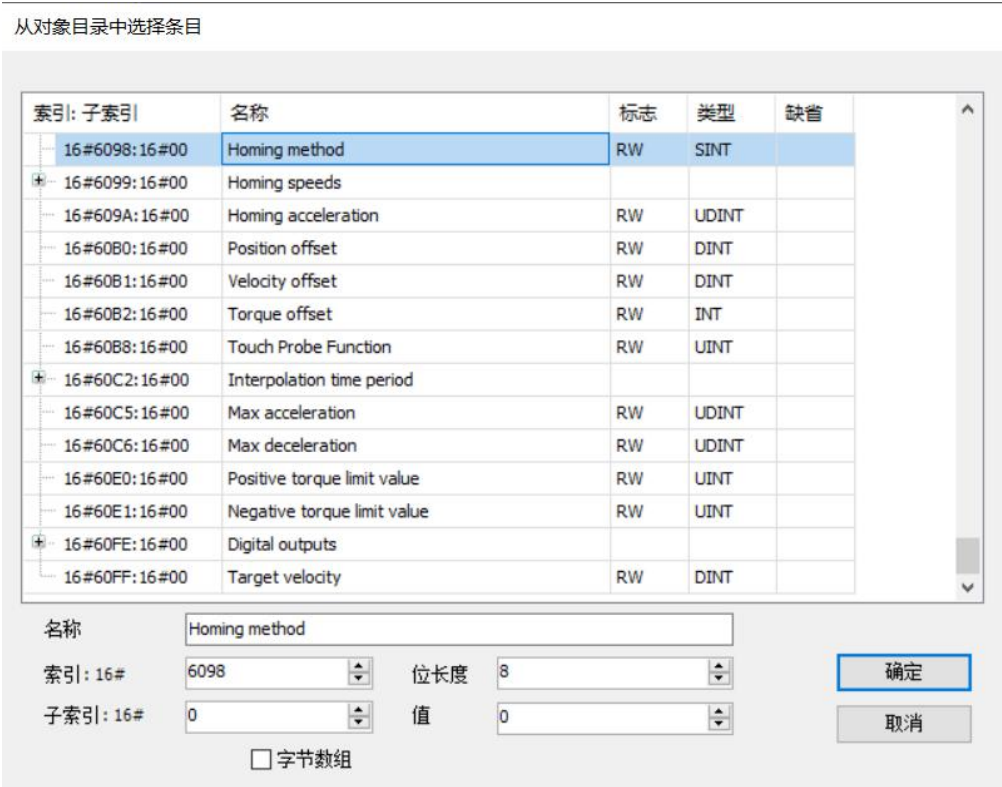


图 14.30 选择“Homing method”

3) 将值设置为 19，点击“确定”按钮。

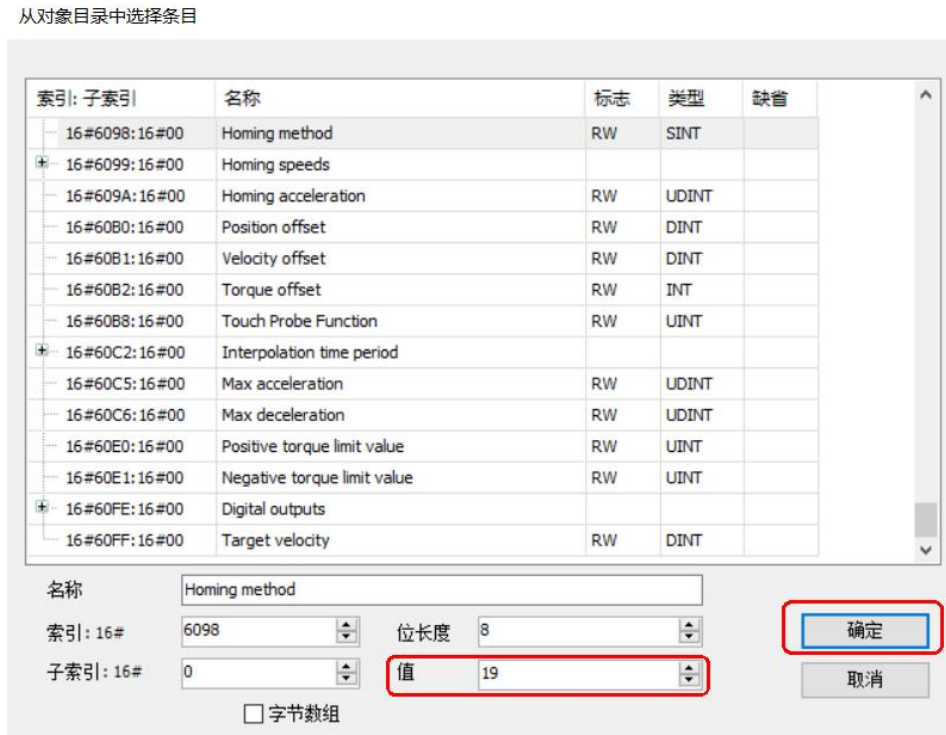


图 14.31 写入新值

4) 添加完成后,在列表中会显示对应的项目,代表启动参数添加成功。如图 11.32 所示。



图 14.32 启动参数添加成功

5) 登录并下载 PLC 程序,即可完成启动参数配置。

14.3.1. 伺服从站添加 402 轴

在 LeadSys 中使用运动控制功能时，需要先在 EtherCAT 从站下添加 402 轴，该轴在程序中作为控制对象。

针对雷赛公司自己的从站，LeadSys 已经进行了预处理，添加从站时会自动添加与其对应的轴。如图 14.33 所示。

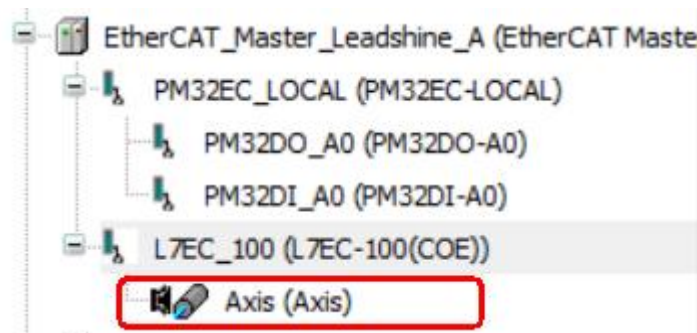


图 14.33 添加了雷赛 L7 伺服电机

对于用户自己添加的第三方从站，在使用时需要手动添加 402 轴，操作步骤如下：

- 1) 添加 EtherCAT 从站，具体操作可参考 14.2.从站组态章节；
- 2) 用鼠标右键点击 EtherCAT 从站，选择“添加 SoftMotion CiA402 轴”，如图 14.34 所示。

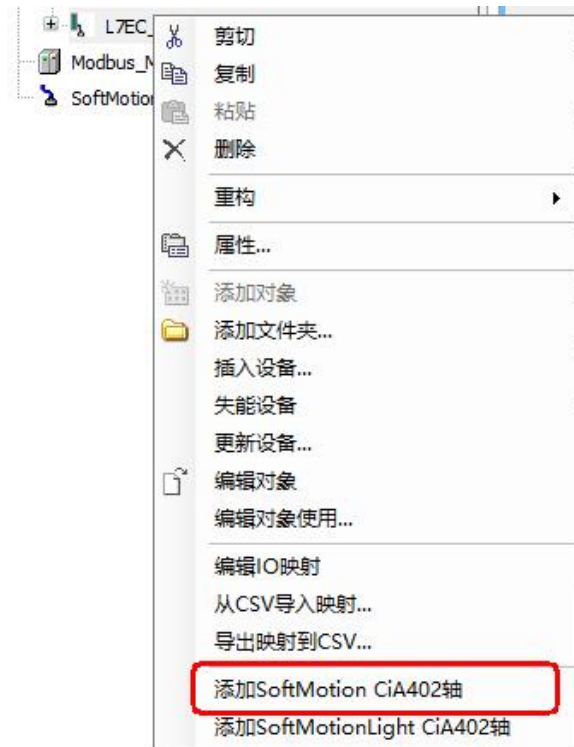


图 14.34 添加 SoftMotion CiA402 轴

14.3.2. 使能/失能从站

在实际应用中，用户可能会遇到组态与实际硬件连接不同的情况，导致总线无法正常运转。

针对这种情况，LeadSys 提供了一种解决方案，用户可以失能没有连接的从站，以保证总线能正常运转。

程序中使用到设备相关的变量时，采用失能设备的方式禁用设备，也不会导致编译报错。如图 14.35 所示。

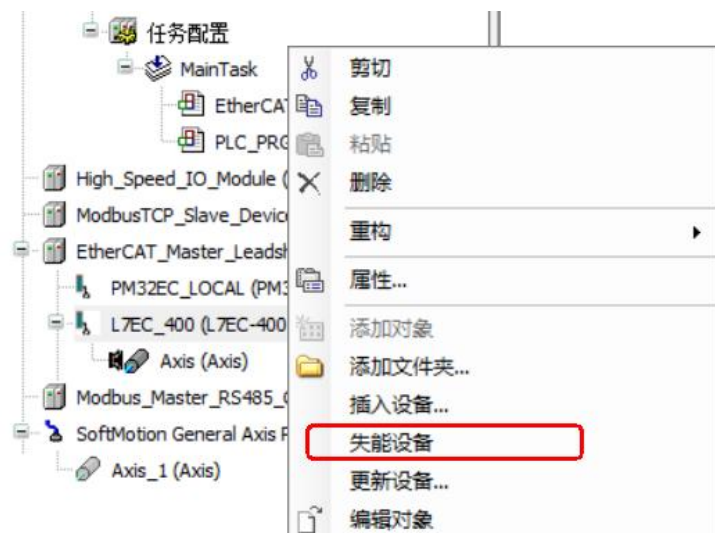


图 14.35 设置“失能设备”

被失能的设备在树中的输入以浅灰色字体显示，如图 14.36 所示。

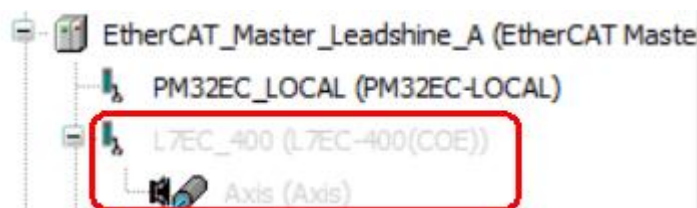


图 14.36 设备已失能

当登录时，被失能的设备将不会显示运行状态，如图 14.37 所示。

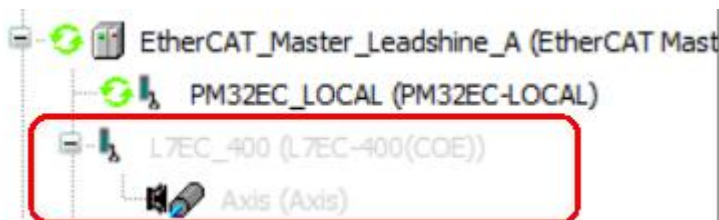


图 14.37 程序运行失能设备不运行

需要重新使能设备时，点击“使能设备”即可，如图 14.38 所示。

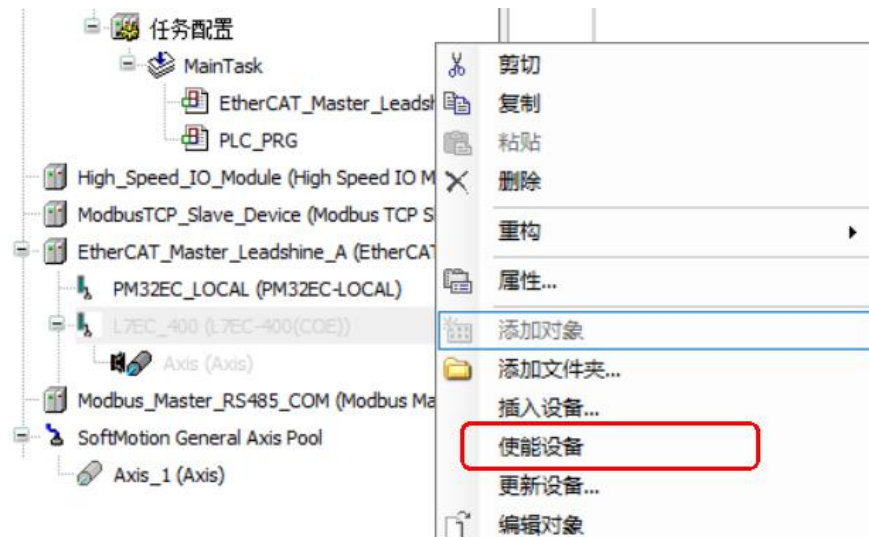


图 14.38 设置“使能设备”

14.3.3. EtherCAT 从站常用参数

EtherCAT 从站常用参数如表 14.3 所示。在使用时，需要加上主站名，如从站名为 L7EC_400，则读取从站状态时，对应的变量为 L7EC_400.wState。

表 14.3 EtherCAT 从站常用参数

变量名	功能
wState	1: ETC_SLAVE_INIT 2: ETC_SLAVE_PREOPERATIONAL 3: ETC_SLAVE_BOOT 4: ETC_SLAVE_SAVEOPERATIONAL 8: ETC_SLAVE_OPERATIONAL
SlaveAddr	从站地址

14.4. EtherCAT 冗余环网功能

LC 系列 PLC 产品 LC2000S、LC5000 都支持双 EtherCAT 网口。两个 EtherCAT 网口可以独立运行，也可以配置为冗余环网功能，如图 14.39 所示的冗余环网连接方案。

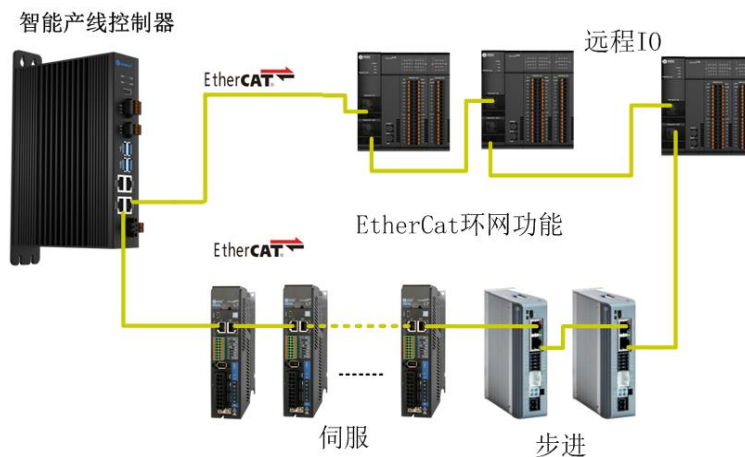


图 14.39 冗余环网连接方案

冗余环网具有以下特点：

- 1) 任一从站故障（第一个从站除外），其他从站的通信正常、功能正常。
- 2) 任一根网线故障（第一个从站入口网除外），所有从站均正常运行。
- 3) 故障的从站重新上电后，可通过运动控制指令（MC_RESET、SMC3_ReinitDrive1 等）恢复从站运行。

注意：按照 EtherCAT 协议标准，第一个从站是标准时钟，因此第一个从站故障，会影响整个总线的通信。

冗余环网的配置方法如下：

- 1) 检查 EtherCAT 主站是否采用 3.5.16.30 版本，可以选择左侧树的 EtherCAT 主站，更新设备，检查或选择 3.5.16.30，如图 14.40 所示。

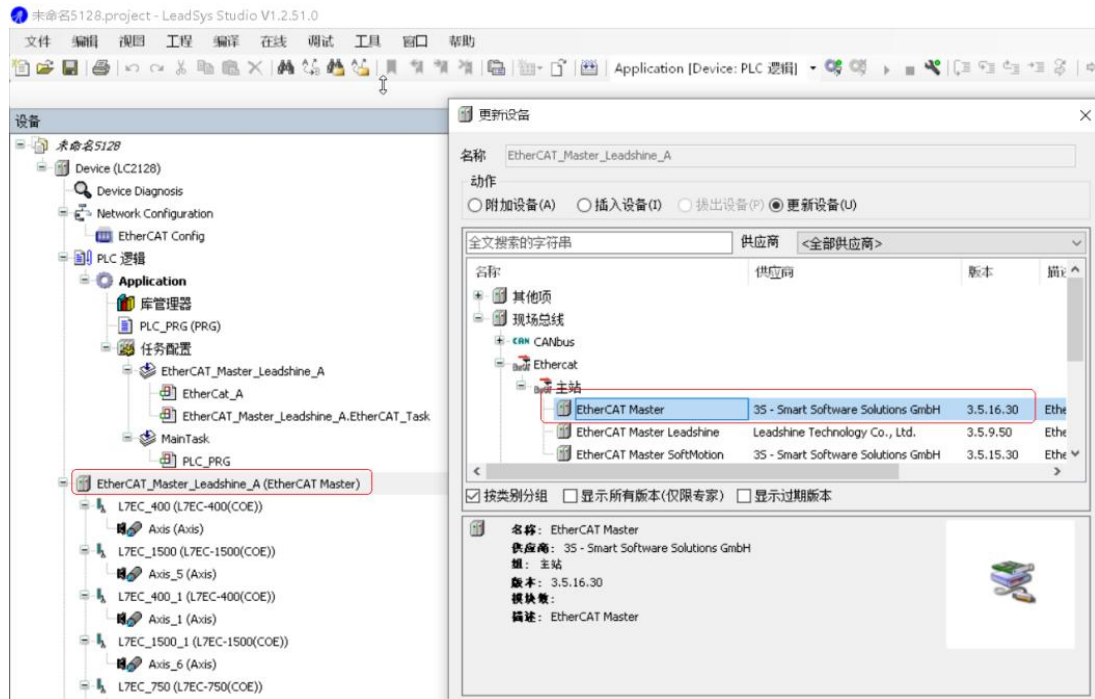


图 14.40 检查 EtherCAT 主站的版本号

- 2) 在“EtherCAT NIC 设置”和“冗余 EtherCAT NIC 设置”中，建议统一选用“按 MAC 选择网络”或“按名称选择网络”。图 14.41 是采用“按 MAC 选择网络”。
- 3) “Device”扫描并连接 PLC 后，点击“浏览”，设置 EtherCAT 的两个网口，如图 14.41 所示，从 EtherCAT 总线连接，从上面的网口开始，环形连接从站，最后连接到图 14.41 中的第二个网口。
- 4) 勾选“选项”下面的“自动重启从站”。

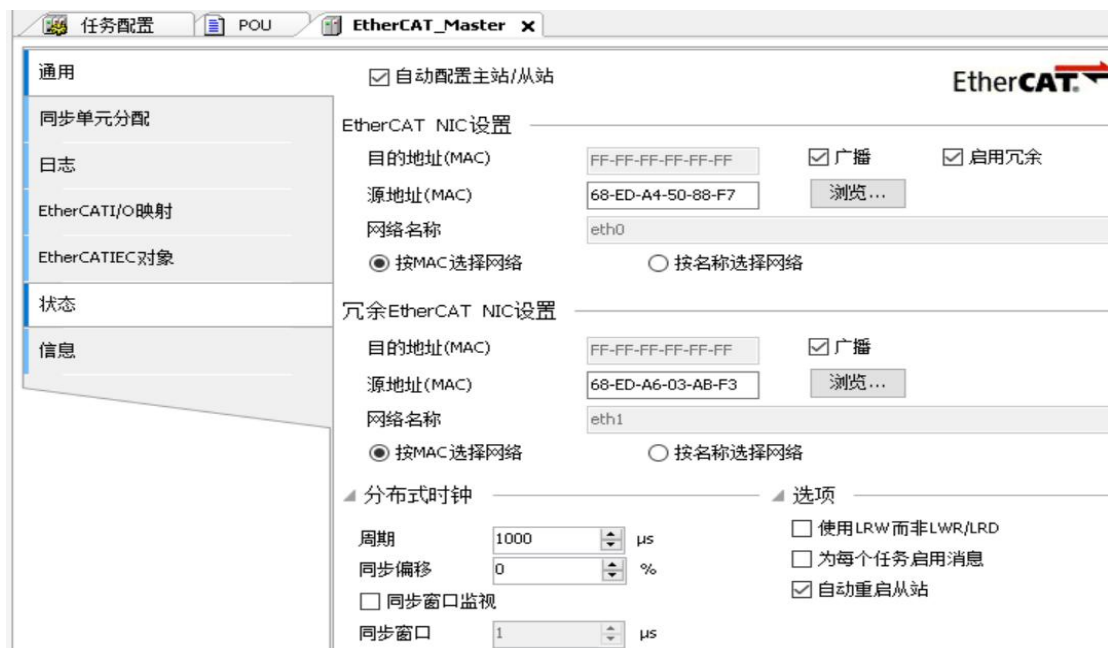


图 14.41 冗余环网参数的设置

第十五章 通过 U 盘下载 PLC 程序

中大型 PLC 可将 U 盘中保存的应用程序包下载并替换 PLC 中的应用程序。

通过 U 盘下载 PLC 程序操作如下：

1) PLC 应用程序包生成

U 盘中包含由 LeadSys 软件打包生成的命名为 Application.app 和 Application.crc 的应用程序包，应用程序包通过工具“在线”→“创建启动应用”生成，如图 15.1 所示。



图 15.1 生成程序包

- 2) 将应用程序包存放在 U 盘的根目录下
- 3) 在 PLC 任意 USB 接口中插入包含应用程序包的 U 盘，将控制器断电重启可进行程序下载。程序下载完成后，程序运行中 RUN 指示灯常亮。若下载失败或程序未运行，RUN 指示灯熄灭。

第十六章 故障诊断

LeadSys Studio 使用的编程语言具有执行效率高、编程灵活等特点，但是对用户的编程能力也有较高要求。

用户在编写程序时需要避免指针非法访问、除数为 0、数组越界、类型隐式转换、死循环等异常操作，否则将可能导致 PLC 系统出现执行异常甚至崩溃的情况。

本章节主要对 PLC 可能出现的异常情况提供排查措施，用户可根据实际情况参考使用。

程序编译错误的定位

1) 编译错误的定位

编译工程后，LeadSys 软件默认显示输出编译错误信息。在编译输出窗口，可以查看编译错误，双击错误显示行，可以定位到出错的代码，如图 16.1 所示。

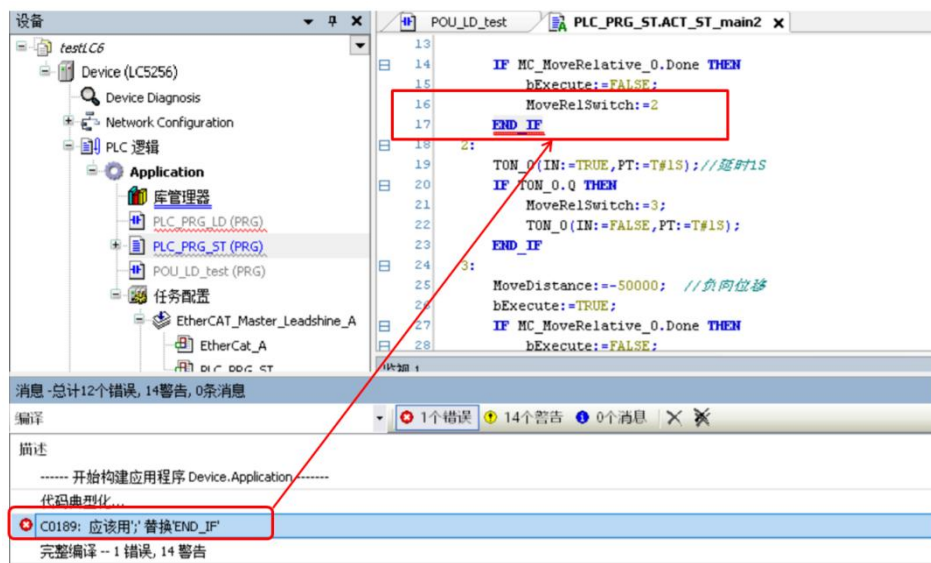


图 16.1 定位到出错代码位置

2) 预编译错误的定位

在显示编译错误信息窗口，可切换到“预编译”，查看预编译的错误，如图 16.2 所示。

预编译错误，是在编译之前进行代码语法分析后得到的错误信息。“预编译”输出显示，如图 16.3 所示，有 10 个错误。

3) 算法库错误信息

用同样的方法可切换到“库管理器”，查看算法库错误提示信息，如图 16.4 所示。

打开库管理器，如图 16.5 所示，如果不用到此库，可删除。用到了此库，则手动安装算法库。



图 16.2 切换至“预编译”

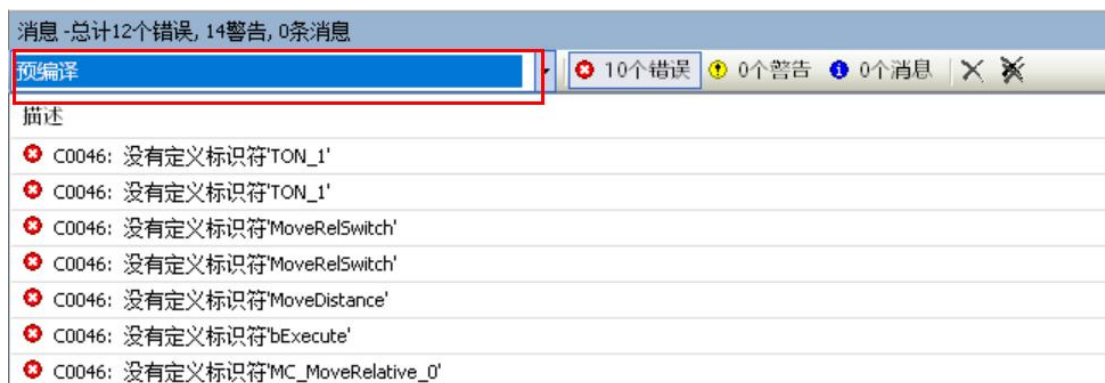


图 16.3 “预编译”显示的提示

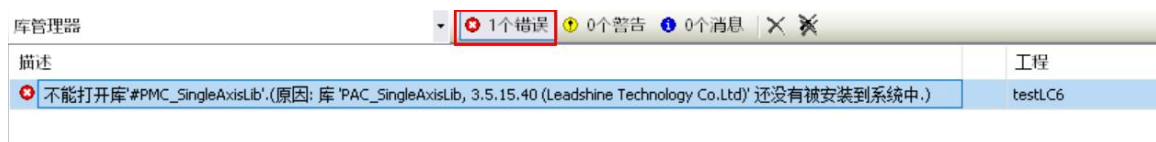


图 16.4 “库管理器”显示的提示

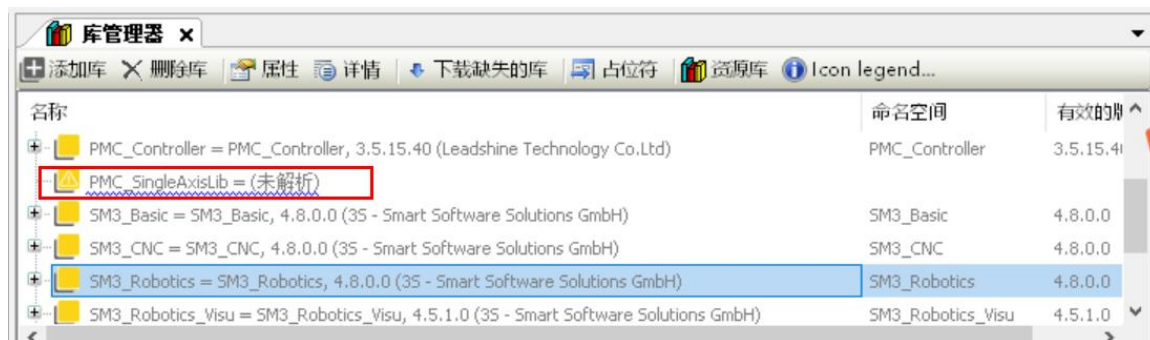


图 16.5 打开库管理器，添加或删除某个库

16.1. PLC 程序运行异常处理

16.1.1. 常见的异常现象及处理措施

在编写、调试软件的过程中可能遇见以下异常现象：

- 1) 下载程序后编程软件无法扫描到相应的 PLC 设备，PLC 运行拨码置为 stop 之后再重新上电，可正常扫描。
- 2) 下载程序时或者运行一段时间后，PLC 信息显示栏提示错误内容“程序下载-异常”，如图 16.6 所示。



图 16.6 程序下载异常

此时在日志界面可以看到有  类型错误，程序无法正常运行，如图 16.7 所示。



图 16.7 日志界面

- 3) 下载程序后或者运行一段时间，登录 PLC 后，弹出“没有源代码使用于这个项目...”提示框，如图 16.8 所示。



图 16.8 提示框，“没有源代码使用于这个项目...”

- 4) 下载程序时，弹出“下载失败：PLC 异常。”提示框，如图 16.9 所示。



图 16.9 提示框，“程序下载失败”

排查上述异常现象的措施有 3 种：

a) 隐含检查的 POU—指针检查

该方法适用于排查空指针（即指针为 0）的情况，具体操作步骤如下：

- 右键“Application”，选择“添加对象”→“用于隐式检查的 POU”，如图 16.10 所示；
- 在弹出界面勾选“指针检查”，点击“打开”，如图 16.11 所示；
- 如图 16.12 所示，在添加的函数“CheckPointer”里面手动输入以下代码：

```
IF ptToTest = 0 THEN
    CheckPointer := ptToTest;
END_IF
```

- 登录 PLC，将程序运行状态切换至“停止”状态；
- 在新增代码处添加断点，如图 16.13 所示，并激活（快捷键 F9）；
- 运行程序，当检查到空指针时，进入断点，如图 16.14 所示；
- 当进入断点后，按 F10，直到定位到程序中出现问题的位置，如图 16.15 所示；
- 定位到问题程序段后，用户需要根据实际情况修改程序，避免异常出现。

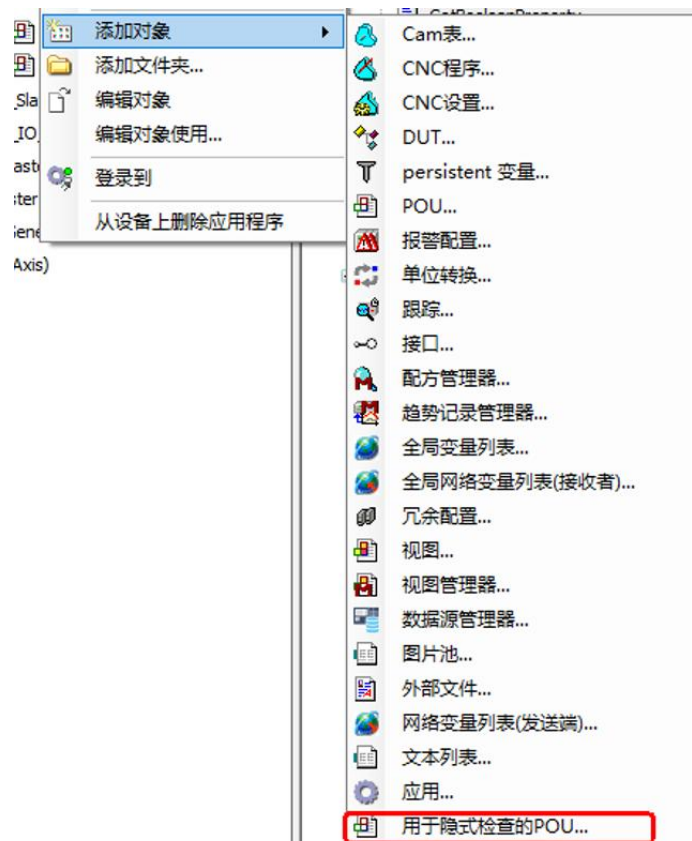


图 16.10 添加“用于隐式检测的 POU”

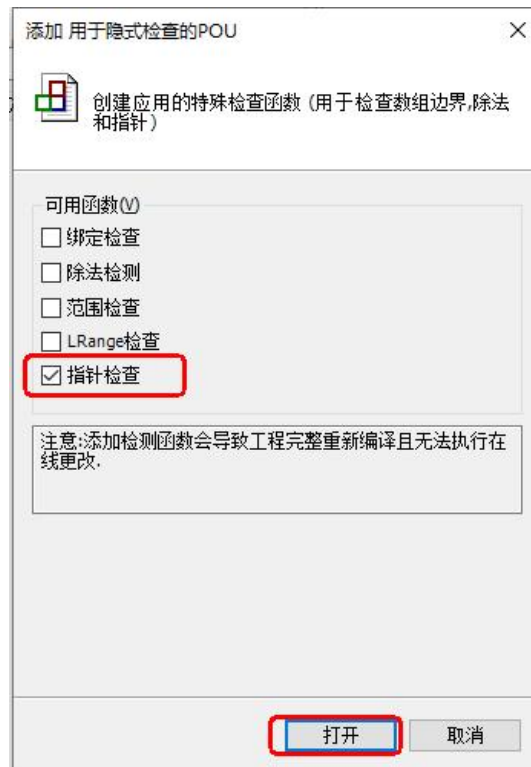


图 16.11 勾选“指针检查”

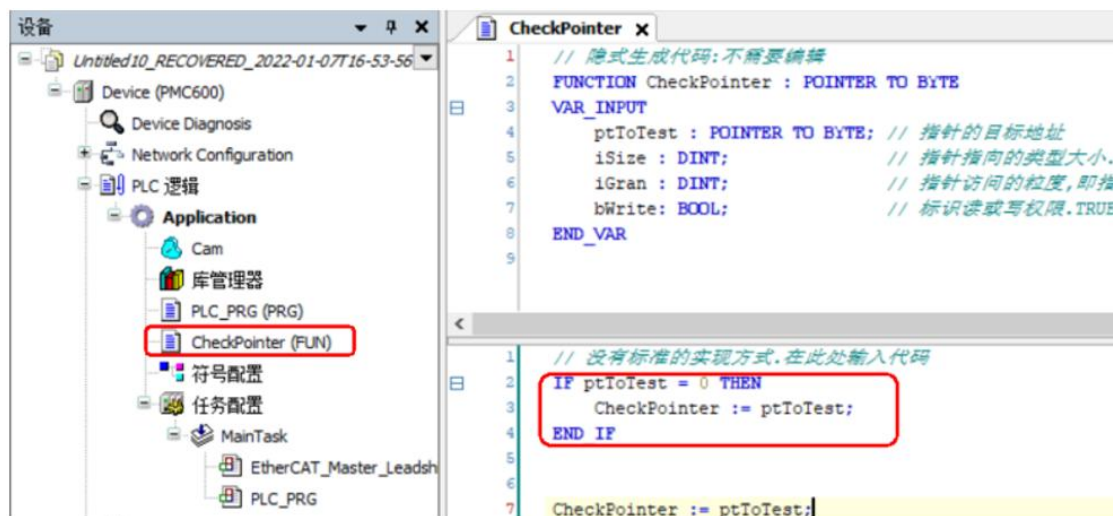


图 16.12 “指针检查”的代码

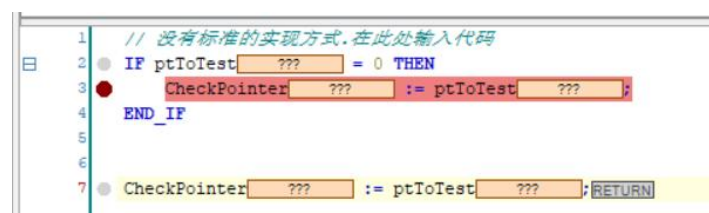


图 16.13 添加断点

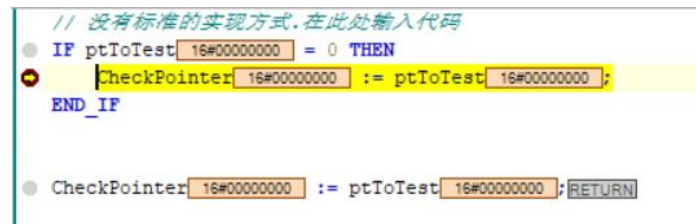


图 16.14 程序运行，进入断点

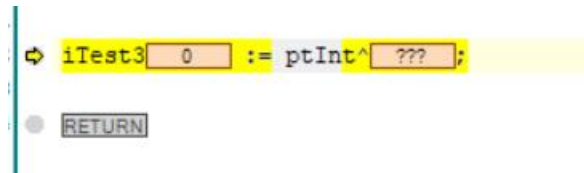


图 16.15 按 F10 键，定位到程序出错处

b) 隐含检查的 POU—绑定检查

该方法适用于排查数组越界的情况，具体操作步骤如下：

- 右键“Application”，选择“添加对象”→“用于隐式检查的 POU”；
- 在弹出界面勾选“绑定检查”，点击“打开”，如图 16.16 所示；
- 登录 PLC，停止程序运行，在新增的绑定函数“CheckBounds”里面的代码“CheckBounds := lower;”和“CheckBounds := upper;”处添加断点并激活（快捷键 F9），如图 16.17 所示；
- 运行程序，当检测到数组越界后，进入断点，如图 16.18 所示；

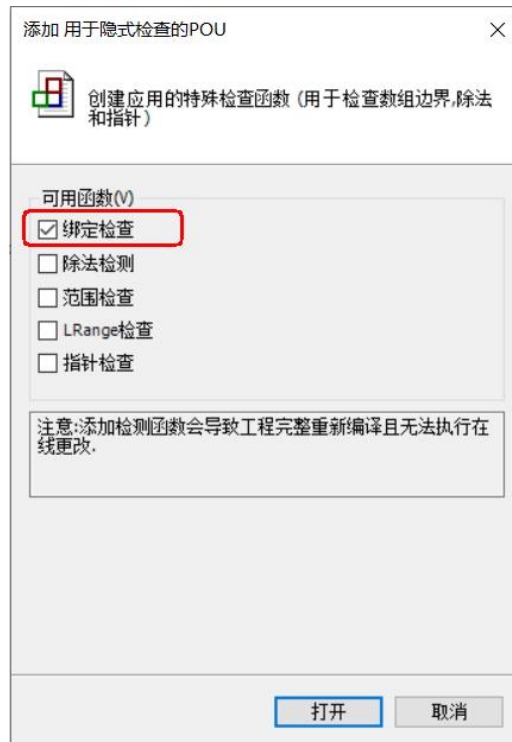


图 16.16 勾选“绑定检查”

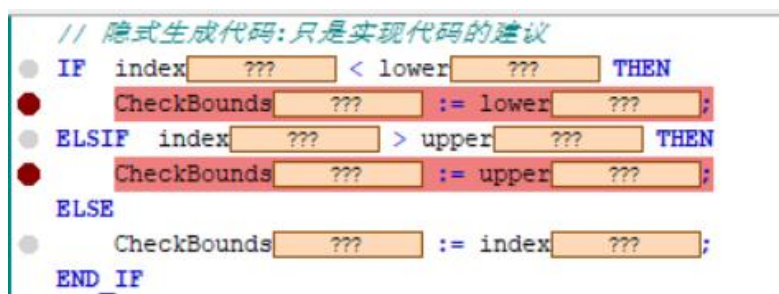


图 16.17 在程序中添加断点并激活

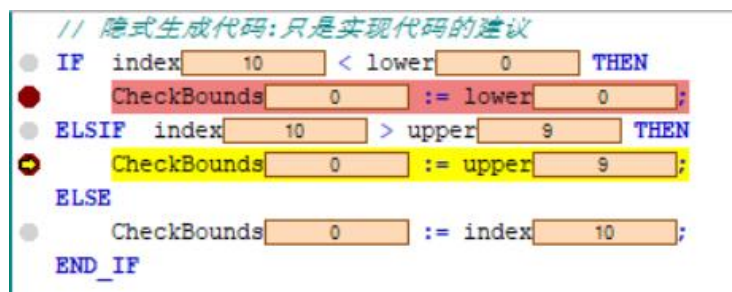


图 16.18 数组越界，程序跳至断点处

➤ 使用快捷键 F10 直到定位到程序中异常的位置，如图 16.19 所示；

- 定位到问题程序段后，用户需要根据实际情况修改程序，避免异常出现。

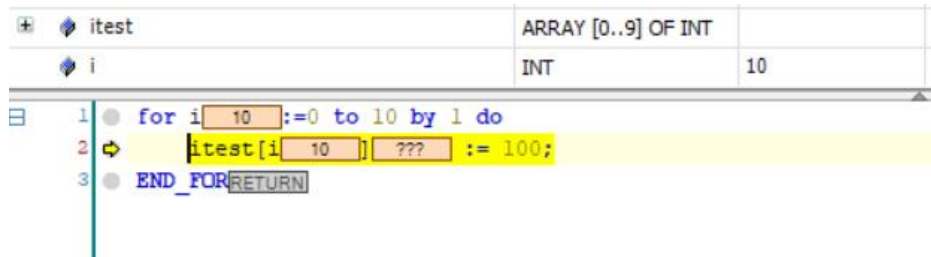


图 16.19 定位至程序异常处

c) 隐含检查的 POU—除法检测

该方法适用于排查除数为 0 的情况，具体操作步骤如下：

- 右键“Application”，选择“添加对象”→“用于隐式检查的 POU”；
- 在弹出界面勾选“除法检测”，点击“打开”，如图 16.20 所示；
- 登录 PLC，停止程序运行，在新增函数里的代码“CheckDivDInt:=1;”、“CheckDivLInt:=1;”、“CheckDivLReal:=1;”位置增加断点并激活（快捷键 F9），如图 16.21 所示。

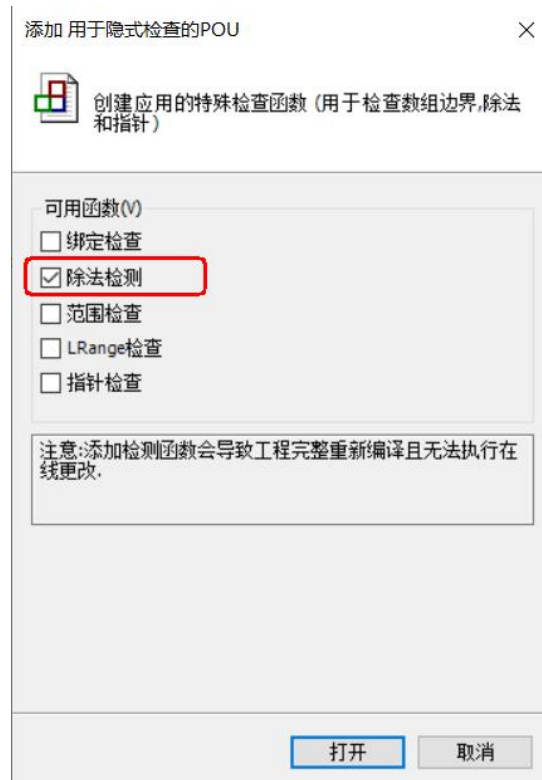


图 16.20 勾选“除法检测”

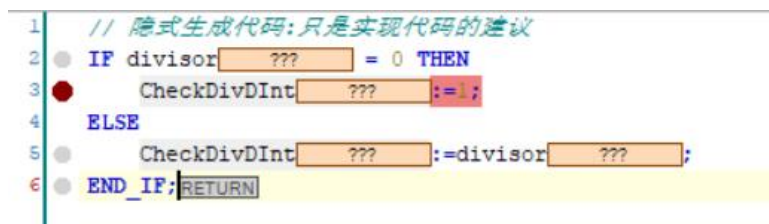


图 16.21 在程序中添加断点并激活

- 运行程序，直到检测到除数为 0 的情况，进入断点，如图 16.22 所示；
- 使用快捷键 F10 直到定位到程序中异常的位置，如图 16.23 所示；
- 定位到问题程序段后，用户需要根据实际情况修改程序，避免异常出现。

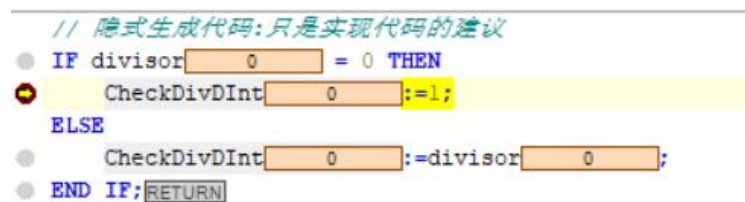


图 16.22 检测到除数为 0 的情况，进入断点

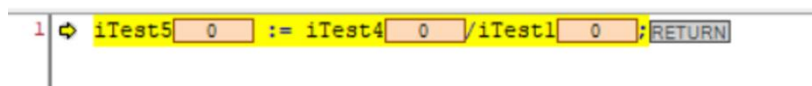


图 16.23 定位至除数为 0 处

16.1.2. 因程序问题导致 PLC 失控的处理措施

常见的 PLC 应用程序问题：

- 1) 应用程序中出现死循环或 循环数过大
- 2) 应用程序访问空指针或指针越界
- 3) 应用程序调用底层功能块导致 runtime 崩溃

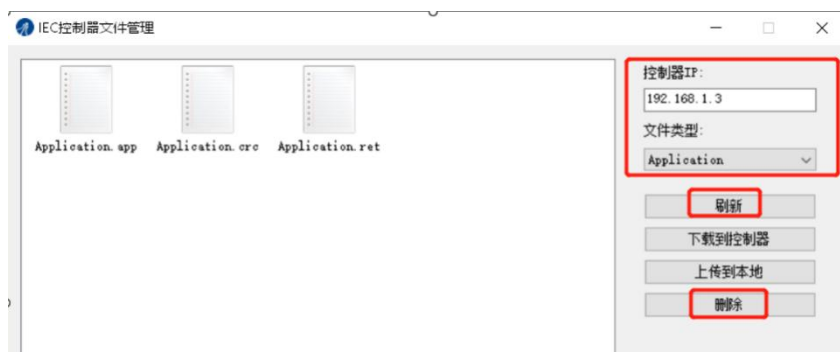
因程序编写不当，下载该程序至 PLC，程序启动后，可能会出现 PLC 资源耗尽，LeadSys 软件对 PLC 无法控制的现象，扫描不到设备或 ping 不通，或者无

法连接、无法在线或下载 PLC。恢复 PLC 至正常状态的方法如下：

方法一：中型 PLC 将 RUN/STOP 拨到 STOP 状态；大型 PLC 用短导线短接 PLC 上的 0V 和 RUN 端子，如图所示；重新上电启动后，扫描到 PLC，下载一个正确的应用程序，再断开短接线。



方法二：如果 PLC 网口还 ping 的通，可以在 Leadsys Studio 软件上，工具，打开固件升级工具，删除应用程序。



方法三：大型 PLC 采用恢复出厂设置，可以将 PLC 的 IP 地址恢复，并删除应用程序。

方法四：LC1000 通过液晶屏+按钮，删除应用程序，或强制停止应用程序。

方法五：大型 PLC 通过 U 盘下载一个正确 PLC 程序，可删除之前的应用程序。或者：U 盘（FAT32 格式，32G 及以下）根目录放入 RmApp 文件。插上 U 盘，上电重启，则应用 app 被删除。再拔掉 U 盘，可正常使用。

采用以上方法恢复 PLC 后，需要排查应用程序错误后，再下载程序。

16.2. 常见故障码

错误码 SMC_ERROR (Enumeration)

这个数据结构包括了 SoftMotion 功能块可能返回的错误编号，可以通过错误编号了解发生了什么错误，方便及时处理。调用 SMC_ErrorString 功能块可以根据错误号生成错误字符串输出量。

表 16.1 错误码及其描述

错误编号	模块	枚举量	描述
0	所有	SMC_NO_ERROR	无错误
1	DriveInterface	SMC_DI_GENERAL_COMMUNICATION_ERROR	通讯错误（例如 e.g. Sercos 环断开）
2	DriveInterface	SMC_DI_AXIS_ERROR	轴错误
3	DriveInterface	SMC_DI_FIELDBUS_LOST_SYNCHRONITY	总线周期同步失败
10	DriveInterface	SMC_DI_SWLIMITS_EXCEEDED	超出了软件限位
11	DriveInterface	SMC_DI_HWLIMITS_EXCEEDED	超出了硬件限位
12	DriveInterface	SMC_DI_LINEAR_AXIS_OUTOFRANGE	超过轴的位置限制
13	DriveInterface	SMC_DI_HALS_OR_QUICK_STOP_NOT_SUPPORTED	不支持暂停和快速停止
14	DriveInterface	SMC_DI_VOLTAGE_DISABLED	轴未使能
15	DriveInterface	SMC_DI_IRREGULAR_ACTPOSITION	从驱动获取的实际位置为非法值，请检查通信。

16	DriveInterface	SMC_DI_POSITIONLAGERR OR	跟随误差超出限制 错误。设置位置和实际位置之间的误差超过了给定的限制。
17	DriveInterface	SMC_DI_HOMING_ERROR	回零过程错误
18	DriveInterface	SMC_DI_LICENSING_ERRO R	软件授权失败
20	all motion generating modules	SMC_REGULATOR_OR_STA RT_NOT_SET	控制器不能执行或 者正在刹车
21	All motion generating modules	SMC_WRONG_CONTROLLE R_MODE	轴处于错误的控制 模式。
25		SMC_INVALID_ACTION_FO R_LOGICAL	无效的逻辑动作
30	DriveInterface	SMC_FB_WASNT_CALLED_ DURING_MOTION	在运动没有结束前， 运动模块没有被调用。
31	All modules	SMC_AXIS_IS_NO_AXIS_R EF	所给的 AXIS_REF 变量不是 AXIS_REF 类型的。
32	All motion generating modules	SMC_AXIS_REF_CHANGED _DURING_OPERATION	在运动过程中，输入的轴 AXIS_REF 变量被更换。
33	DriveInterface	SMC_FB_ACTIVE_AXIS_DI ABLED	在运动过程中，轴失效 (MC_Power.bRegul

			atorOn)
34	All motion generating modules	SMC_AXIS_NOT_READY_F OR_MOTION	轴没有处于准备好的状态，不能执行运动
35	All motion generating function blocks	SMC_AXIS_ERROR_DURIN G_MOTION	轴在运动过程中报错
40	ViRTUalDrive	SMC_VD_MAX_VELOCITY _EXCEEDED	超过最大速度 (fMaxVelocity)
41	ViRTUalDrive	SMC_VD_MAX_ACCELE RATION_EXCEEDED	超过最大加速度 (fMaxAcceleration)
42	ViRTUalDrive	SMC_VD_MAX_DECELE RATION_EXCEEDED	超过最大减速度 (fMaxDeceleration)
50	SMC_Homing	SMC_3SH_INVALID_VELAC C_VALUES	无效的速度或加速度值
51	SMC_Homing	SMC_3SH_MODE_NEEDS_H WLIMIT	模式要求（可能基于安全原因）关闭限位开关
60		SMC_FRC_NO_FREE_HAND LE	无空闲句柄用于打开文件
70	SMC_SetCont rollerMode	SMC_SCM_NOT_SUPPORTE D	模式不支持
71	SMC_SetCont rollerMode	SMC_SCM_AXIS_IN_WRON G_STATE	在当前模式下，不能改变控制器模式。
72	SMC_SetCont rollerMode	SMC_SCM_INTERRUPTED	设置控制器模式被MC_Stop 指令或者

			errorstop 打断
75	SMC_SetTorque	SMC_ST_WRONG_CONTROLLER_MODE	轴不在正确的模式下
90	SMC_ChangeGearingRatio	SMC_CGR_ZERO_VALUES	无效值
91	SMC_ChangeGearingRatio	SMC_CGR_DRIVE_POWERED	电子齿轮参数在轴受控时不可被修改
92	SMC_ChangeGearingRatio	SMC_CGR_INVALID_POSITION	旋转轴无效的模值 (≤ 0)
93	SMC_ChangeGearingRatio	SMC_CGR_POSPERIOD_NOT_INTEGRAL	参数值非整数
110	MC_Reset	SMC_P_FTASKCYCLE_EMPTY	设置的轴运动循环周期错误 (fTaskCycle = 0)
120	MC_Reset	SMC_R_NO_ERROR_TO_RESET	轴无错误
121	MC_Reset	SMC_R_DRIVE_DOESNT_ANSWER	轴不执行“错误复位”
122	MC_Reset	SMC_R_ERROR_NOT_RESETTABLE	错误不能被复位
123	MC_Reset	SMC_R_DRIVE_DOESNT_ANSWER_IN_TIME	轴的通讯状态不响应
130	MC_ReadParameter, MC_ReadBoolParameter	SMC_RP_PARAM_UNKNOWN	参数个数不明确
131	MC_ReadPara	SMC_RP_REQUESTING_ERROR	向驱动设备传递时

	meter, MC_ReadBool Parameter	ROR	出错；可通过见功能 块 ReadDriveParameter 获取错误号。
132	MC_ReadPara meter, MC_ReadBool Parameter	SMC_RP_DRIVE_PARAMET ER_NOT_MAPPED	驱动参数没有映射
133	MC_ReadPara meter, MC_ReadBool Parameter	SMC_RP_PARAM_CONVER SION_ERROR	参数值转换失败
140	MC_WritePar ameter, MC_WriteBool Parameter	SMC_WP_PARAM_INVALID	不可识别的参数编 号，或写错误
141	MC_WritePar ameter, MC_WriteBool Parameter	SMC_WP_SENDING_ERROR	可通过功能块 WriteDriveParameter 获取错误号。
142	MC_WritePar ameter, MC_WriteBool Parameter	SMC_WP_DRIVE_PARAMET ER_NOT_MAPPED	驱动参数没有映射
143	MC_WritePar ameter, MC_WriteBool Parameter	SMC_WP_PARAM_CONVER SION_ERROR	参数值转换失败

170	MC_Home	SMC_H_AXIS_WASNT_STANDSTILL	轴没有处于 standstill 状态
171	MC_Home	SMC_H_AXIS_DIDNT_START_HOMING	启动回零运动时出错。
172	MC_Home	SMC_H_AXIS_DIDNT_ANSWER	通讯错误。
173	MC_Home	SMC_H_ERROR_WHEN_STOPPING	回零运动错误（没有设置减速）
174	MC_Home	SMC_H_AXIS_IN_ERRORSTOP	回零运动无法执行（轴处于错误停止状态）
180	MC_Stop	SMC_MS_UNKNOWN_STOPPING_ERROR	停止时出现未知错误
181	MC_Stop	SMC_MS_INVALID_ACCDEC_VALUES	无效速度或加速度值
182	MC_Stop	SMC_MS_DIRECTION_NOT_APPLICABLE	方向信号不能使用 shortest
183	MC_Stop	SMC_MS_AXIS_IN_ERRORSTOP	停止无法执行（轴处于错误停止状态）
184	MC_Stop	SMC_BLOCKING_MC_STOP_WASNT_CALLED	MC_Stop 功能块未调用
185		SMC_MS_AXIS_ALREADY_STOPPING	正在执行停止运动，功能块无法中断
200		SMC_UNKNOWN_TASK_INTERVAL	不确定的总线任务时间
201	MC_MoveAbsolute	SMC_MA_INVALID_VELOCITY_VALUES	速度或加速度值无效

202	MC_MoveAbsolute	SMC_MA_INVALID_DIRECTION	方向错误
226	MC_MoveRelative	SMC_MR_INVALID_VELOCITY_VALUES	速度或加速度值无效
227	MC_MoveRelative	SMC_MR_INVALID_DIRECTION	方向错误
251	MC_MoveAdditive	SMC_MAD_INVALID_VELOCITY_VALUES	速度或加速度值无效
252	MC_MoveAdditive	SMC_MAD_INVALID_DIRECTION	方向错误
276	MC_MoveSuperImposed	SMC_MSI_INVALID_VELOCITY_VALUES	速度或加速度值无效
277	MC_MoveSuperImposed	SMC_MSI_INVALID_DIRECTION	方向错误
301	MC_MoveVelocity	SMC_MV_INVALID_ACCELERATION_VALUES	速度或加速度值无效
302	MC_MoveVelocity	SMC_MV_DIRECTION_NOT_APPLICABLE	方向错误，不能使用shortest/fastest
325	MC_PositionProfile	SMC_PP_ARRAYSIZE	数组大小错误
326	MC_PositionProfile	SMC_PP_STEP0MS	每段间隔时间= t#0s
350	MC_VelocityProfile	SMC_VP_ARRAYSIZE	数组大小错误
351	MC_VelocityProfile	SMC_VP_STEP0MS	每段间隔时间= t#0s
375	MC_AccelerationProfile	SMC_AP_ARRAYSIZE	数组大小错误

	ionProfile		
376	MC_AccelerationProfile	SMC_AP_STEP0MS	每段间隔时间= t#0s
400	MC_TouchProbe	SMC_TP_TRIGGEROCCUPIED	触发已经激活
401	MC_TouchProbe	SMC_TP_COULDNT_SET_WINDOW	DriveInterface不支持窗口函数
402	MC_TouchProbe	SMC_TP_COMM_ERROR	通讯错误
410	MC_AbortTrigger	SMC_AT_TRIGGERNOTOCCUPIED	触发已重新分配
426	SMC_MoveContinuousRelative	SMC_MCR_INVALID_VELOCITY_VALUES	速度或加速度值无效
427	SMC_MoveContinuousRelative	SMC_MCR_INVALID_DIRECTION	无效的方向
451	SMC_MoveContinuousAbsolute	SMC_MCA_INVALID_VELOCITY_VALUES	速度或加速度值无效
452	SMC_MoveContinuousAbsolute	SMC_MCA_INVALID_DIRECTION	无效的方向
453	SMC_MoveContinuousAbsolute	SMC_MCA_DIRECTION_NOT_APPLICABLE	Direction= fastest 不能使用
475	SMC_Change	SMC_SDL_INVALID_AXIS_	轴状态不正确，应该

	DynamicLimits	STATE	为 standstill 或 power_off 才能调用该模块
476	SMC_ChangeDynamicLimits	SMC_SDL_INVALID_VELOCITY_VALUES	无效的速度，加减速速度，加加速度
600	SMC_CamRegister	SMC_CR_NO_TAPPETS_IN_CAM	CAM 不包含任何 tappings
601	SMC_CamRegister	SMC_CR_TOO_MANY_TAPPETS	Tappet 组 ID 超过 MAX_NUM_TAPPETS
602	SMC_CamRegister	SMC_CR_MORE_THAN_32_ACCESSES	在一个 CAM_REF 上有超过 32 个访问连接
625	MC_CamIN	SMC_CI_NO_CAM_SELECTED	没有 CAM 被选择
626	MC_CamIN	SMC_CI_MASTER_OUT_OF_SCALE	主轴在超出有效值范围
627	MC_CamIN	SMC_CI_RAMPIN_NEEDS_VELOCITY_VALUES	对 ramp_in 函数，速度和加速度值必须被指定。
628	MC_CamIN	SMC_CI_SCALING_INCORRECT	缩放变量 fEditor / TableMasterMin / Max 不正确
629	MC_CamIn	SMC_CI_TOO_MANY_TAPPETS_PER_CYCLE	在同一周期内有太多的 Tappet 被激活使用

640	SMC_CAMBo unds, SMC_CamBo unds_Pos	SMC_CB_NOT_IMPLEMENTED	调用 CAM 数据表的功能块没有执行
675	MC_GearIn	SMC_GI_RATIO_DENOM	分母为 0
676	MC_GearIn	SMC_GI_INVALID_ACC	加速度无效
677	MC_GearIn	SMC_GI_INVALID_DEC	减加速度无效
678	MC_GearIn, MC_CamIn	SMC_GI_MASTER_REGULATOR_CHANGED	在不允许的情形下， 主轴的 Enable/Disable 状态 切换
679	MC_GearIn	SMC_GI_INVALID_JERK	无效的加加速度
725	MC_Phase	SMC_PH_INVALID_VELACCDEC	速度、减加速度或加速度无效
726	MC_Phase	SMC_PH_ROTARYAXIS_PERIOD0	旋转轴 fPositionPeriod = 0
750	All modules using MC_CAM_REF as input	SMC_NO_CAM_REF_TYPE	所给的 CAM 不是 MC_CAM_REF 类型的
751	MC_CamTableSelect	SMC_CAM_TABLE_DOES_NOT_COVER_MASTER_SCALE	CamTable 的数据没有覆盖 Master area 从 xStart 到 xEnd 的范围
752	MC_CamTableSelect	SMC_CAM_TABLE_EMPTY_MASTER_RANGE	凸轮表的主轴范围 为空
753	MC_CamTable	SMC_CAM_TABLE_INVALID	凸轮表的主轴范围

	eSelect	D_MASTER_MINMAX	最大最小值无效
754	MC_CamTableSelect	SMC_CAM_TABLE_INVALID_SLAVE_MINMAX	凸轮表的从轴范围 最大最小值无效
775	MC_GearInPos	SMC_GIP_MASTER_DIRECTION_CHANGE	在与从轴连接期间， 主轴改变了旋转方向
776	MC_GearInPos	SMC_GIP_SLAVE_REVERSAL_CANNOT_BE_AVOIDED	VAR_INPUT“Avoid Reversal”被设置，但是从轴反转无法避免
777	MC_GearInPos	SMC_GIP_AVOID_REVERSAL_FOR_FINITE_AXIS	如果是直线轴， VAR_INPUT“Avoid Reversal”不需要设置
800	SMC_BacklashCompensation	SMC_BC_BL_TOO_BIG	齿轮间隙 (fBacklash) 太大 (> 位置回合/2)
825	All motion generating function blocks	SMC_QPROF_DIVERGES	内部计算错误
826	All motion generating function blocks	SMC_QPROF_DISTANCE_TOO_SHORT	内部计算错误
827	All motion generating function	SMC_QPROF_NO_RESULT	内部计算错误

	blocks		
728	All motion generating function blocks	SMC_QPROF_INVALID_NEW_LBD	内部轨迹二次方计算错误
729	All motion generating function blocks	SMC_QPROF_BAD_NEGOTIATION	内部轨迹二次方计算错误
730	All motion generating function blocks	SMC_QPROF_INVALID_INTERVAL	内部轨迹二次方计算错误
731	All motion generating function blocks	SMC_QPROF_NOT_ENOUGH_PHASES	内部轨迹二次方计算错误
850	SMC_SetRam pType	SMC_SRT_NOT_STANDSTILL_OR_POWEROFF	只允许轴在 STANDSTILL 和 POWER_OFF 状态 下设置
851	SMC_SetRam pType	SMC_SRT_INVALID_RAMPTYPE	无效的类型
852	SMC_SetRam pType	SMC_SMT_NOT_STANDSTILL_OR_POWEROFF	只允许轴在 STANDSTILL 和 POWER_OFF 状态 下设置

853	SMC_SetRam pType	SMC_SMT_INVALID_MOVE MENTTYPE_OR_POSITIONP ERIOD	无效的运动类型或 位置周期
854	SMC_SetRam pType	SMC_SMT_AXIS_NOT_VIRT UAL	只允许在虚轴上使 用的功能块
1000	CNC function blocks which are supervising the licensing	SMC_NO_LICENSE	无 CNC 许可
1001	SMC_Interpol ator	SMC_INT_VEL_ZERO	轨迹未被处理，因为 设置速度为 0
1002	SMC_Interpol ator	SMC_INT_NO_STOP_AT_EN D	上一个轨迹对象最 终的速度 Vel_End > 0。
1003	SMC_Interpol ator	SMC_INT_DATA_UNDERRU N	警告：在 DataIn 中 处理了 GEOINFO 表，但是列表最后速 度未到达。 原因：忘记设置在 DataIn 中的队列的 EndOfList，或者 SMC_Interpolator 速 度比轨迹预处理模 块要快。
1004	SMC_Interpol ator	SMC_INT_VEL_NONZERO_ AT_STOP	停止时速度>0。

1005	SMC_Interpolator	SMC_INT_TOO_MANY_RECURSIONS	多个 SMC_Interpolator 使用同一个轴
1006	SMC_Interpolator	SMC_INT_NO_CHECKVELOCITIES	在数据预处理阶段，SMC_CheckVelocities 不是最后一个模块。
1007	SMC_Interpolator	SMC_INT_PATH_EXCEEDED	内部/数字错误。
1008	SMC_Interpolator	SMC_INT_VEL_ACC_DEC_ZERO	速度、加速度或减加速度值为空或太小。
1009	SMC_Interpolator	SMC_INT_DWIPOTIME_ZERO	dwIpoTime 为 0
1010	SMC_Interpolator	SMC_INT_JERK_NONPOSITIVE	加加速度必须是正值
1011	SMC_Interpolator	SMC_INT_QPROF_DIVERGES	内部计算错误
1012	SMC_Interpolator	SMC_INT_INVALID_VELOCITY_MODE	无效的速度模式
1013	SMC_Interpolator	SMC_INT_TOO_MANY_AXES_INTERPOLATED	太多轴插补
1050	SMC_Interpolator2Dir	SMC_INT2DIR_BUFFER_TOO_SMALL	数据缓冲区太小
1051	SMC_Interpolator2Dir	SMC_INT2DIR_PATH_FITS_NOT_IN_QUEUE	轨迹在队列中没有运行完。
1070		SMC_XINT_INVALID_DIRECTION	无效的方向输入值

1071		SMC_XINT_NOINTERSECTI ON	根据给定的 CNC 路 径中的 X 轴位置，无 法确定终止位置
1080	SMC_Interpol ator	SMC_WAR_INT_OUTQUEU E_TOO_SMALL	OutQueueDataIn 太 小
1081	SMC_Interpol ator	SMC_WAR_END_VELOCITI ES_INCORRECT	终止速度不一致
1100	SMC_CheckV elocities	SMC_CV_ACC_DEC_VEL_N ONPOSITIVE	速度、加速度或减速 度值不为正值
1120	SMC_Control axisbypos	SMC_CA_INVALID_ACCDE C_VALUES	fGapVelocity / fGapAcceleration / fGapDeceleration 不 为正值
1200	SMC_NCDec oder	SMC_DEC_ACC_TOO_LITT LE	不允许的加速度值
1201	SMC_NCDec oder	SMC_DEC_RET_TOO_LITTL E	不允许的减速度值
1202	SMC_NCDec oder	SMC_DEC_OUTQUEUE_RA N_EMPTY	数据队列为空
1203	SMC_NCDec oder	SMC_DEC_JUMP_TO_UNK NOWN_LINE	由于行号未知，跳转 指令不能执行
1204	SMC_NCDec oder	SMC_DEC_INVALID_SYNT AX	语法无效
1205	SMC_NCDec oder	SMC_DEC_3DMODE_OBJEC T_NOT_SUPPORTED	对象不支持 3D 模 式。
1206	SMC_NCDec oder	SMC_DEC_NEGATIVE_PERI OD	辅助轴周期为负值 不正确

1207	SMC_NCDecoder	SMC_DEC_DIMENSIONS_EXCLUSIVE_AU	插补中，轴 A 与轴 U 不能同时执行
1208	SMC_NCDecoder	SMC_DEC_DIMENSIONS_EXCLUSIVE_BV	插补中，轴 B 与轴 V 不能同时执行
1209	SMC_NCDecoder	SMC_DEC_DIMENSIONS_EXCLUSIVE_CW	插补中，轴 C 与轴 W 不能同时执行
1300	SMC_GCodeViewer	SMC_GCV_BUFFER_TOO_SMALL	缓冲区过小
1301	SMC_GCodeViewer	SMC_GCV_BUFFER_WRONG_TYPE	缓冲区元素类型错误
1302	SMC_GCodeViewer	SMC_GCV_UNKNOWN_IPO_LINE	当前行不能从插补器中找到
1500	All function blocks using SMC_CNC_REF	SMC_NO_CNC_REF_TYPE	给出的 CNC 程序不是 SMC_CNC_REF 类型的
1501	All function blocks using SMC_OUTQUEUE	SMC_NO_OUTQUEUE_TYPE	给出的 OutQueue 不是 SMC_OUTQUEUE 类型的
1502	All function blocks using pbyBuffer	SMC_GEOINFO_BUFFER_MISALIGNED	在 pbyBuffer 中，没有使用 4 字节对齐
1600	CNC function blocks	SMC_3D_MODE_NOT_SUPPORTED	功能块只支持 2D 模式
1700	SMC_SmoothAddAxes	SMC_SAA_SMOOTHAREA_TOO_LARGE	平滑范围过大

1701	SMC_Smooth AddAxes	SMC_SAA_SP_INVALID_IN PUT	dSmoothingPart 输入 参数无效，有效范围 [0..1]
1800	SMC_Segmen tAnalyzer	SMC_SA_QUEUE_NOT_IN_ BUFFER	功能块检测到 OutQueue 缓存已经 满，但是数据没有结 束
1801	SMC_Segmen tAnalyzer	SMC_SA_QUEUE_CHANGE D_DURING_OP	当功能块操作 OutQueue 时， OutQueue 缓存发生 改变
1820		SMC_OS_INVALID_PARAM ETER	无效参数
1830	SMC_BlockSe archSavePos	SMC_BSSP_IPO_NOT_ACTI VE	位置不能保存，插补 器是无效的
1831	SMC_BlockSe arch	SMC_BS_SAVEDPOS_NOT_ REACHED	保存的位置没有找 到，可能是一个不同 的路径
1832	SMC_BlockSe arch	SMC_BS_NO_POS_STORED	SMC_BlockSearchSa vePos 不能执行或者 为一个错误的状态
1900		SMC_INVALID_FEATURE_F LAG	特征标记有效值范 围为[1..31]
1901		SMC_SMB_HFUN_NOT_SUP PORTED	功能块不支持 H 代 码
1902		SMC_SMB_ONLY_3DMODE	功能块只能工作在 3D 模式

1903		SMC_SMB_ERROR_COMPUTING_SPLINE	计算样条错误
1910		SMC_SMM_INVALID_PARAMETER_NUMBER	辅助参数值太大
2000	SMC_ReadNC File	SMC_RNCF_FILE_DOESNT_EXIST	文件不存在
2001	SMC_ReadNC File	SMC_RNCF_NO_BUFFER	无缓冲区被分配
2002	SMC_ReadNC File	SMC_RNCF_BUFFER_TOO_SMALL	缓冲区太小
2003	SMC_ReadNC File	SMC_RNCF_DATA_UNDER_RUN	缓冲区为空。
2004	SMC_ReadNC File	SMC_RNCF_VAR_COULDN'T_BE_REPLACED	占位符变量不能被替换。
2005	SMC_ReadNC File	SMC_RNCF_NOT_VARLIST	输入 pvl 不指向 SMC_VARLIST 对象。
2006	SMC_ReadNC File	SMC_RNCF_NO_STRINGBUFFER	读 NC 文件没有定义缓存
2007	SMC_ReadNC File	SMC_RNCF_STRINGBUFFER_OVERFLOW	读 NC 文件的缓存溢出
2050	SMC_ReadNC Queue	SMC_RNCQ_FILE_DOESNT_EXIST	文件不存在。
2051	SMC_ReadNC Queue	SMC_RNCQ_NO_BUFFER	无缓冲区被定义
2052	SMC_ReadNC Queue	SMC_RNCQ_BUFFER_TOO_SMALL	缓冲区太小

2053	SMC_ReadNC Queue	SMC_RNCQ_UNEXPECTED _EOF	文件异常结束
2100	SMC_AxisDia gnosticLog	SMC_ADL_FILE_CANNOT_ BE_OPENED	文件不能打开
2101	SMC_AxisDia gnosticLog	SMC_ADL_BUFFER_OVERR UN	缓冲区过载； WriteToFile 一定是 被过于频繁地调用
2200	SMC_ReadCA M	SMC_RCAM_FILE_DOESNT _EXIST	文件不能打开
2201	SMC_ReadCA M	SMC_RCAM_TOO_MUCH_D ATA	要存储地 CAM 过大
2202	SMC_ReadCA M	SMC_RCAM_WRONG_COM PILE_TYPE	错误地编译模式
2203	SMC_ReadCA M	SMC_RCAM_WRONG_VER SION	文件版本错误
2204	SMC_ReadCA M	SMC_RCAM_UNEXPECTED _EOF	文件异常结束
3001	SMC_WriteDr iveParamsToFi le	SMC_WDPF_CHANNEL_OC CUPIED	SMC_WDPF_TIME OUT_PREPARING_ LIST
3002	SMC_WriteDr iveParamsToFi le	SMC_WDPF_CANNOT_CRE ATE_FILE	文件不能被创建
3003	SMC_WriteDr iveParamsToFi le	SMC_WDPF_ERROR_WHEN _READING_PARAMS	读取参数出错
3004	SMC_WriteDr	SMC_WDPF_TIMEOUT_PRE	准备参数表时超时

	iveParamsToFile	PARING_LIST	
5000	SMC_Encoder	SMC_ENC_DENOM_ZERO	编码参考的转变因数 (dwRatioTechUnits Denom) 分母为 0。
5001	SMC_Encoder	SMC_ENC_AXISUSED BYOT HERFB	其他模块尝试在编码器轴上处理运动。
5002	DriveInterface	SMC_ENC_FILTER_DEPTH_INVALID	无效的过滤器深度。

错误码 CAA NetERROR

这个数据结构包括了调用“CAA_NetBaseServices.lib”库返回的错误编号。可通过错误编号找到对应的报错信息。

表 16.2 报错名及其描述

报错名	报错代码	注释
NO_ERROR	0	没报错
FIRST_ERROR	6000	
TIME_OUT	6001	超时
INVALID_ADDR	6002	IP 地址无效
INVALID_HANDLE	6003	句柄无效
INVALID_DATAPOINTER	6004	数据指针无效
INVALID_DATASIZE	6005	数据尺寸无效
UDP_RECEIVE_ERROR	6006	接收不到 UDP 通讯数据
UDP_SEND_ERROR	6007	无法发送 UDP 通讯数据

UDP_SEND_NOT_COMPLETE	6008	UPD 通讯数据发送未完成
UDP_OPEN_ERROR	6009	创建 UPD 端口失败
UDP_CLOSE_ERROR	6010	关闭 UPD 端口失败
TCP_SEND_ERROR	6011	无法发送 TCP 通讯数据
TCP_RECEIVE_ERROR	6012	接收不到 TCP 通讯数据
TCP_OPEN_ERROR	6013	创建 TCP 端口失败
TCP_CONNECT_ERROR	6014	无法建立 TCP 连接
TCP_CLOSE_ERROR	6015	无法关闭 TCP 连接
TCP_SERVER_ERROR	6016	TCP 服务报错
WRONG_PARAMETER	6017	有一个无效的参数
ERROR_UNKNOWN	6018	未知错误
TCP_NO_CONNECTION	6019	没有 TCP 连接
IOCTL_ERROR	6020	内部错误 (例如不支持 IOCTL)
FIRST_MF	6050	
LAST_ERROR	6099	

第十七章 LeadSys Studio 配置技巧

17.1. 在线帮助与联网帮助配置

LeadSys Studio 软件自带的帮助信息很丰富，默认建议使用软件自带的在线帮助，打开 LeadSys Studio 软件，从菜单→工具→选项，去掉下图的勾选。

如果需要联网帮助，再勾选下面的选项。

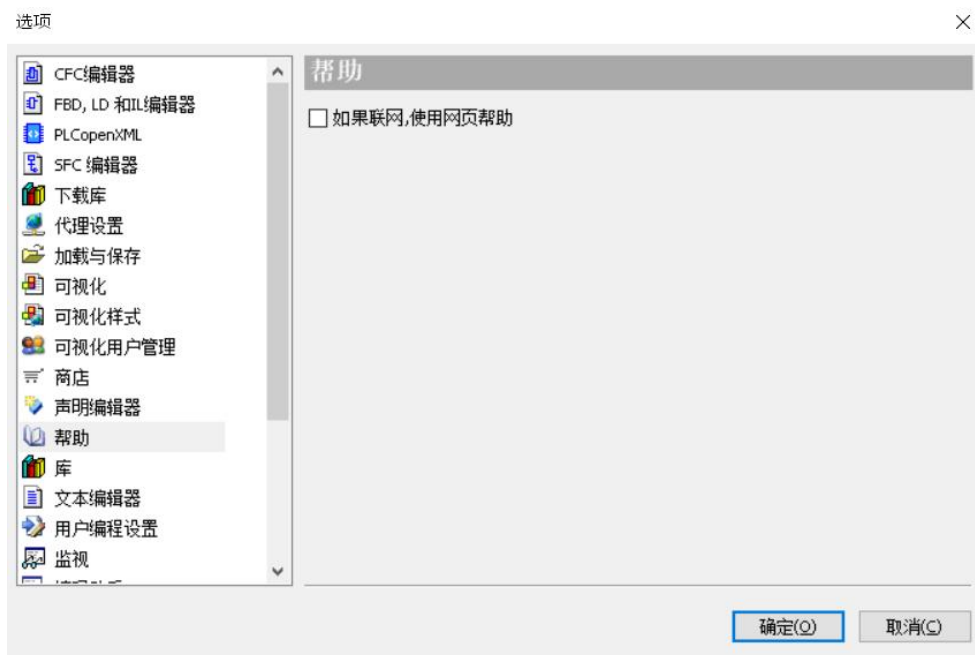


图 17.1 联网帮助选项

然后从菜单的“帮助”可打开在线帮助，如图 17.2 所示。

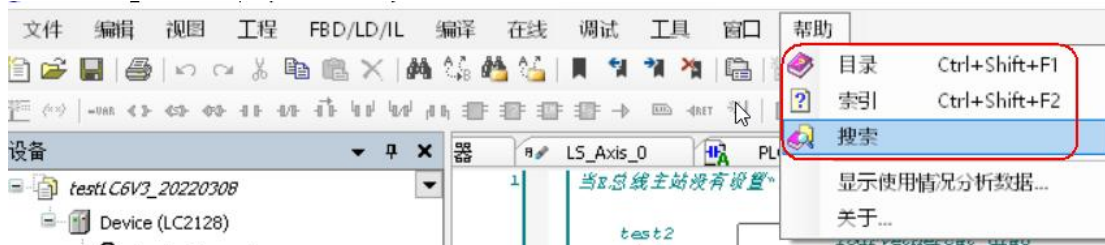


图 17.2 “帮助”菜单

示例：搜索功能块 SMC3_ReinitDrive 的帮助信息。如图 17.3 所示。

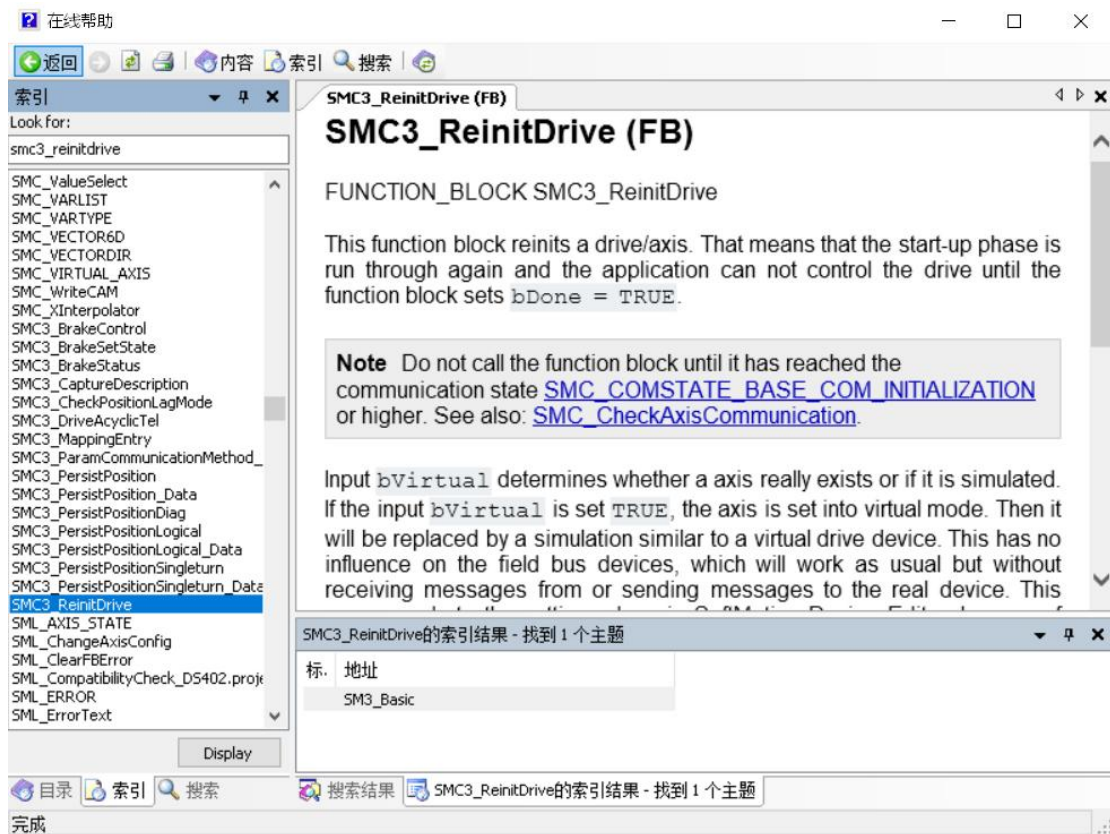


图 17.3 功能块 SMC3_ReinitDrive 的帮助信息

在编程过程中，当对某指令不清楚时，可将光标移至该指令上，再按 F1 键，即可弹出相关帮助信息。如图 17.4 所示，查询 CASE 指令的帮助信息。

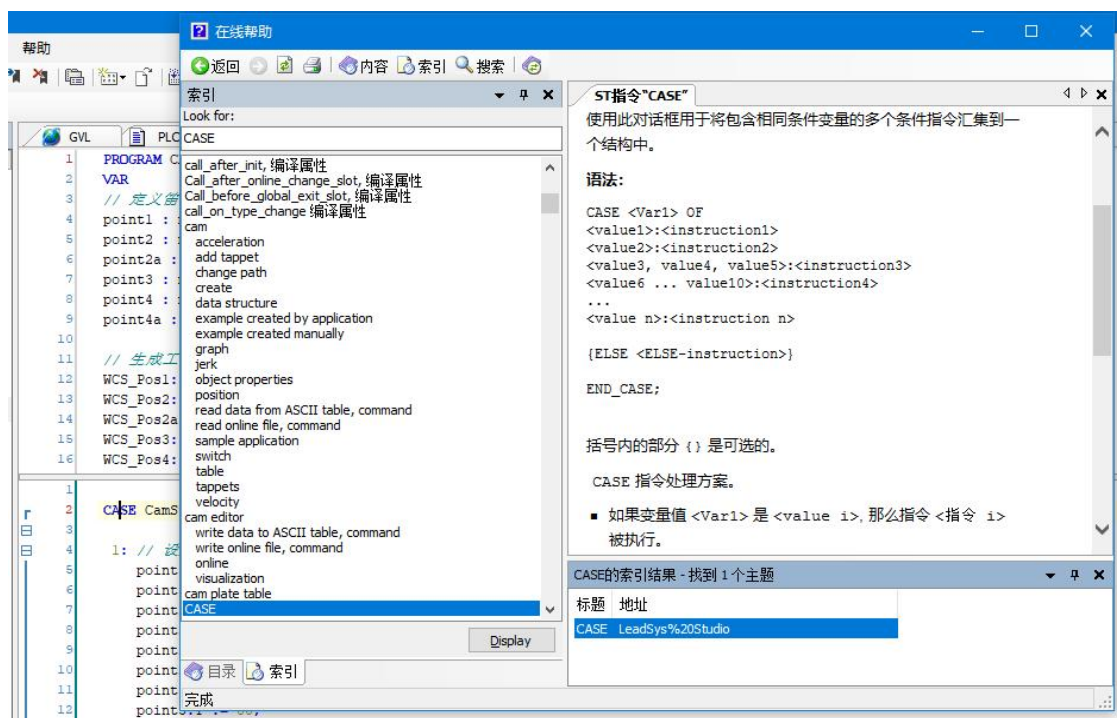


图 17.4 按 F1 键获取帮助信息

17.2. 编码助手的配置

LeadSys Studio 软件默认已经使能了编程助手，如当敲一个未定义的变量，需要提示自动定义变量。编程助手的配置方法为：从菜单→工具→选项中选“编码助手”，可根据自己的喜好选择项目，如图 17.5 所示。

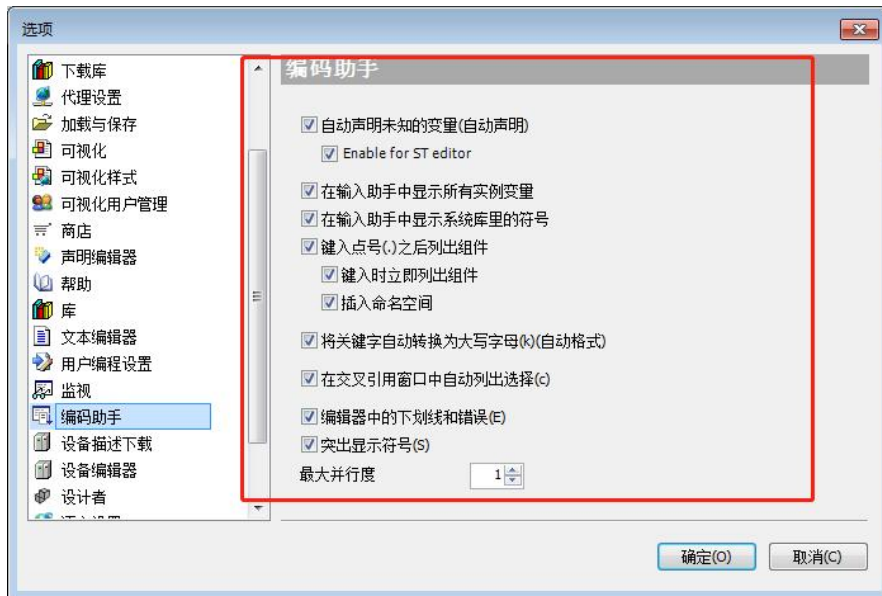


图 17.5 编码助手的配置

17.3. 梯形图 LD 网络注释的显示

显示或隐藏梯形图 LD 网络注释的方法：菜单→工具→选项，如图 17.6 所示，选“FBD, LD 和 IL 编辑器”，在 General 栏中，勾选或不选“Show network comment”即可。梯形图网络注释的显示效果如图 17.7 所示。

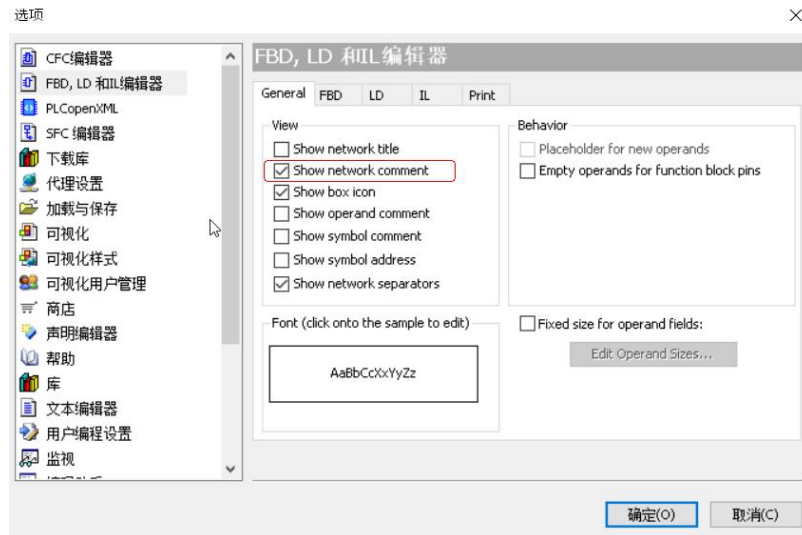


图 17.6 FBD, LD 和 IL 编辑器的选项



图 17.7 显示梯形图的网络注释

17.4. 电机功能码

LeadSys Studio 软件可读写总线型伺服电机驱动器和步进电机驱动器的参数，如图 17.8 所示。

但目前只能支持雷赛公司的电机，型号为：LCEC 系列、L7EC-S 系列、CL3C 系列、DM3C 和 DM3E 系列。

支持参数在线读写

显示参数生效方式

参数编号	名称	当前值	写入值	出厂值	范围	读写权限	有效类型
Pr0.00	模型跟随带宽			1	0~5000	R/W	立即有效
Pr0.01	控制模式			9	9~9	R/W	断电有效
Pr0.02	实时自动调整模式			0x1	0x0~0xFF	R/W	立即有效
Pr0.03	实时自动调整增益			11	0~31	R/W	立即有效
Pr0.04	惯量比			250	0~20000	R/W	立即有效
Pr0.06	指令极性设置			0	0~1	R/W	断电有效
Pr0.07	探针信号极性设置			3	0~3	R/W	断电有效
Pr0.08	每转指令脉冲数			0	0~8388608	R/W	断电有效
Pr0.13	第1转矩限制			300	0~500	R/W	立即有效
Pr0.14	位置偏差过大设置			30	0~500	R/W	立即有效
Pr0.15	绝对式编码器设置			0	0~32767	R/W	断电有效
Pr0.16	再生制电阻阻值			100	40~500	R/W	立即有效
Pr0.17	再生放电电阻功率			50	20~5000	R/W	立即有效
Pr0.19	摩擦力补偿功能设置			0	0~1000	R/W	立即有效
Pr0.23	从站别名			2	0~32767	R/W	断电有效
Pr0.24	从站别名来源			0	0~1	R/W	断电有效
Pr0.25	同步补偿时间1			10	1~100	R/W	立即有效
Pr0.26	同步补偿时间2			50	1~2000	R/W	立即有效
Pr0.27	同步模式指令延时周期数			0	0~50	R/W	立即有效
Pr0.28	CSP模式安全自运行位置...			10	0~10000	R/W	立即有效

图 17.8 读、写电机驱动器参数

客户咨询中心

目录索取 · 技术咨询 · 产品解惑

400-885-5521 销售热线

400-885-5501 技术热线

更多最新的雷赛资讯，请扫码关注！



公众号



视频号

成就客户 共创共赢

深圳市雷赛智能控制股份有限公司
China Leadshine Technology Co., Ltd.

深圳市南山区沙河西路 3157 号南山智谷产业园 B 栋 15-20 层
邮 编：518052
电 话：400-885-5521 传 真：0755-26402718
网 址：www.leisai.com E-Mail: marketing@leisai.com

上海分公司
上海市嘉定区金园五路 601 号

济南代表处
济南市天桥区滨河商务中心 D 座 2003 室

广佛代表处
广州市番禺区汉溪大道 218 号李锦记大厦 A 塔 8032

温州代表处
浙江省温州市瓯海区潘桥街道宁波路阳光城愉景嘉园 8 幢 2604

杭州代表处
浙江省杭州市钱塘区白杨街道 6 号大街 260 号正泰中自科技园 19 幢 1006 室

北京分公司
北京市大兴区绿地启航国际 3 号楼 1109 室

苏州代表处
江苏省苏州市苏州工业园区金尚路 1 号仙峰大厦南楼 7 层

中珠江代表处
中山市东区长江路 33 号汉宏盈基商务中心 9 层 906 室

东莞代表处
广东省东莞市南城区黄金路 1 号东莞天安数码城 F 区 3 栋 604

武汉代表处
湖北省武汉市东湖新技术开发区长城园路 2 号海贝孵化器 209

南京代表处
南京市江宁区科建路天韵南京科创产业园 1155 号 F 栋 403 室

长沙代表处
长沙市开福区湘江北路三段 1500 号北辰时代广场 A3 区 3426 房

川渝代表处
成都市武侯区人民南路四段 27 号商鼎国际 1 栋 1 单元 23 楼 A2309 房

大连代表处
辽宁省大连市沙河口区滨河街 60-1 号新星海中心 A 座 1106 室

※本产品资料中所刊载的产品性能和规格，如因产品改进等原因发生变更时，恕不另行通知，敬请谅解。

2025 年 5 月版