

STEP 机器人控制器软件使用说明书

出版状态：标准

产品版本：V1.0

中国上海新时达机器人有限公司全权负责本控制系统用户操作及编程指南的编制、印刷。

版权所有，保留一切权利。

没有得到上海新时达机器人有限公司的许可，任何单位和个人不得擅自摘抄、复制本书（软件等）的一部分或全部，不得以任何形式（包括资料和出版物）进行传播。

版权所有，侵权必究。内容如有改动，恕不另行通知。

**All Copyright© reserved by Shanghai STEP Robotics Co., Ltd.,
China**

Shanghai STEP Robotics Co., Ltd. authorizes in the documenting, printing to this robot control manual.

All rights reserved.

The information in this document is subject to change without prior notice. No part of this document may in any form or by any means (electronic, mechanical, micro-coping, photocopying, recording or otherwise) be reproduced, stored in a retrieval system or transmitted without prior written permission from Shanghai STEP Robotics Co., Ltd.

长沙工控帮教育科技有限公司

序言

内容提要

本操作指南对 STEP 机器人控制器软件功能进行了全面系统地阐述。本手册可作为使用 STEP 机器人控制器软件的参考资料。

为了确保能够正确地使用 STEP 机器人控制器功能，请在使用前仔细阅读本使用说明书。

读者对象

工程维护人员

用户技术支持人员

内容说明

本说明书内容会有补充和修改，请经常留意我公司网站，更新说明书。我公司网址：www.steprobots.com。

主要特点

STEP 机器人控制器 SRM1000 是一款支持多种工业现场总线、能够实现多轴同步运动控制的运动控制产品，并提供了设备组态、机器人算法库、CNC 插补库、PLCopen 库等多种运动控制功能。其完善、友好的集成开发环境（基于 CoDeSys）能够帮助用户快速、方便的构建面向不同行业的、适用于各种应用场合的运动控制产品。

与安全有关的标记说明

本使用说明书中，与安全相关的内容，使用下列标记。附有安全标记的叙述、内容重要，请务必遵守。



危险

错误使用时，会引起危险情况，可能导致人身伤亡。



注意

错误使用时，会引起危险，可能导致人身轻度或重度伤害和设备损坏。



重要

用户需要遵守和重点注意的部分。

长沙工控帮教育科技有限公司

目录

第一章 说明

第二章 开发环境搭建

第三章 设备组态及配置

第四章 机器人程序实例

第五章 其他功能汇总

长沙工控帮教育科技有限公司

长沙工控帮教育科技有限公司

目录

第一章 说明.....	7
1.1 文档作用.....	7
1.2 目标群体.....	7
1.3 其他相关文档.....	7
1.4 注意事项.....	7
第二章 开发环境搭建.....	9
2.1 开发环境搭建.....	9
2.1.1 安装过程.....	9
2.1.3 上位机网络配置.....	9
2.1.4 网络连接.....	10
2.2 测试例程.....	11
第三章 设备组态及配置.....	15
3.1 ETHERCAT 设备组态及配置.....	15
3.1.1 组态 STEP EtherCAT Servo Drive 伺服.....	15
3.1.2 单轴运动控制实例.....	24
3.1.3 组态 STEP SRE3010 IO 设备.....	28
3.2 总线同步模块.....	31
3.2.1 EL6692 总线同步模块.....	31
3.2.2 EL6692 配置及使用.....	32
3.3 PROFIBUS 设备组态及配置.....	41
3.3.1 组态 Profibus 模块 EL6731.....	41
3.3.2 运行.....	42
3.4 CANOPEN 设备组态及配置.....	42
3.4.1 组态 CANopen 设备.....	43
3.4.2 修改 PLC_PRG 程序.....	45
3.4.3 编译并运行.....	47
第四章 机器人程序实例.....	49
4.1 本例适用机器人.....	49
4.2 程序详解.....	49
4.2.1 伺服组态过程.....	49
4.2.2 IO 设备组态.....	50
4.2.3 附加轴.....	51
4.2.4 全局变量.....	52
4.2.5 IO 程序.....	53
4.2.6 通讯程序.....	53
4.2.7 机器人主程序.....	54
第五章 其他功能汇总.....	61
5.1 串口功能.....	61

5.1.1 添加串口库	62
5.1.2 添加变量	62
5.1.3 主程序代码	62
5.1.4 PC 端串口工具	64
5.2 套接字接口编程	65
5.2.1 STEP 控制器作为服务器的例程:	65
5.2.2 STEP 控制器作为客户端的例程	67
5.3 其他辅助功能	68
5.3.1 访问 FTP 服务器	69
技术支持.....	70

长沙工控帮教育科技有限公司

第一章 说明

1.1 文档作用

文档的作用在于描述机器人控制器软件功能、开发环境搭建、设备组态以及注意事项。此文档能够帮助用户学会使用机器人控制器。

1.2 目标群体

文档的目标群体是：机器人维护人员、机器人集成人员。下面表示描述的是目标群体需要的其它技能。

表 1.1 目标群体以及需要技能

目标群体	需要技能
维护人员	熟悉机器人功能和机器人电气图。
集成人员	熟悉机器人功能和机器人应用。

1.3 其他相关文档

1.4 注意事项

此手册主要向用户介绍机器人控制器软件功能，对于机器人控制器硬件知识、机器人操作使用手册请参考相应文档。

长沙工控帮教育科技有限公司

第二章 开发环境搭建

机器人控制器 SRM1000 的集成开发环境基于 CoDeSys。由于 CoDeSys 是通用的运动控制平台的集成开发环境，尚未包含 STEP 机器人运动控制扩展包以及一些相应的配置文件。

本章解决的主要问题是，用户按照何种方法，依照何种步骤来搭建机器人运动控制平台。

本文附件提供了 Setup_CoDeSysV35SP4.exe 安装包。请用户安装该安装包，不可安装其他的安装包。

2.1 开发环境搭建

2.1.1 安装过程

安装 Setup_CoDeSysV35SP4.exe，需要 Microsoft .NET Framework 4.0 以及 Microsoft Visual C++ x86 Redistributable 的支持。

2.1.3 上位机网络配置

STEP 控制器 SRM1000 在硬件上有三个网络接口，请将上位机网口与 LAN1 口用网线连接，并将上位机该网口的 IP 修改为如 **Error! Reference source not found.**所示的网段：

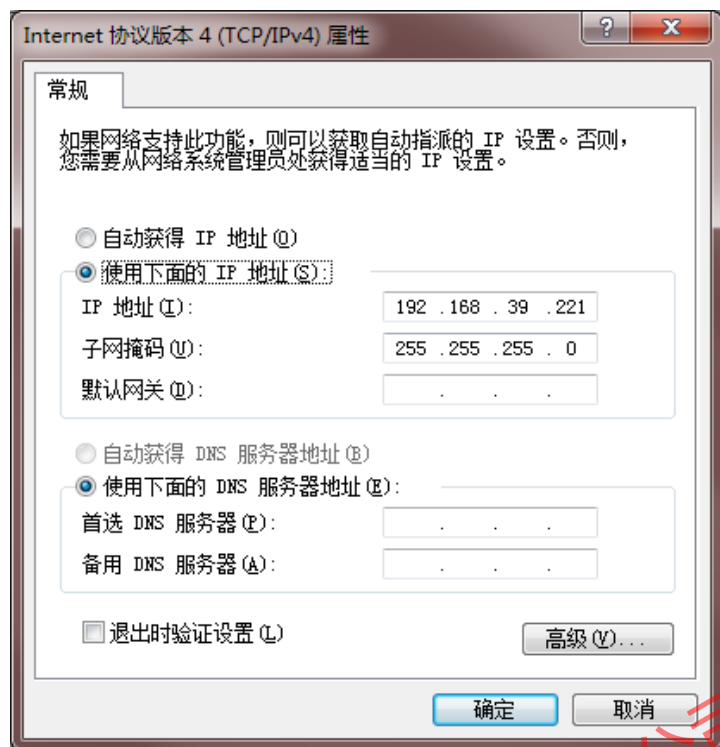


图 2.1 修改 IP 地址

2.1.4 网络连接

用户请用直连网线将上位机与控制器的 LAN1 网口^注直接，或者通过交换机等直连设备进行连接。

请将控制器连接至 24V 直流开关电源，等待 2 分钟至控制器完全启动完毕。

请在上位机运行 C:\Windows\system32\cmd.exe，并输入下列命令。



图 2.2 ping 命令及反馈

若反馈如上图所示，则表明，上位机与控制器的连接成功。

注：本说明书涉及到的控制器硬件接口，将在硬件说明文档中加以详细说明，请用户阅读该文档。

2.2 测试例程

到上一节为止，STEP 控制器在上位机中的开发环境已经搭建完毕。用户此时可以使用此环境来搭建一个简单的应用，来测试本开发环境与 STEP 控制器之间的匹配是否正确。

注：本测试例程与 CoDeSys Online Help→QuickStart→Create and run a project 中例程是完全一致的。用户可以打开 CoDeSys Online Help，并按照该例程的详细步骤一步步进行学习。

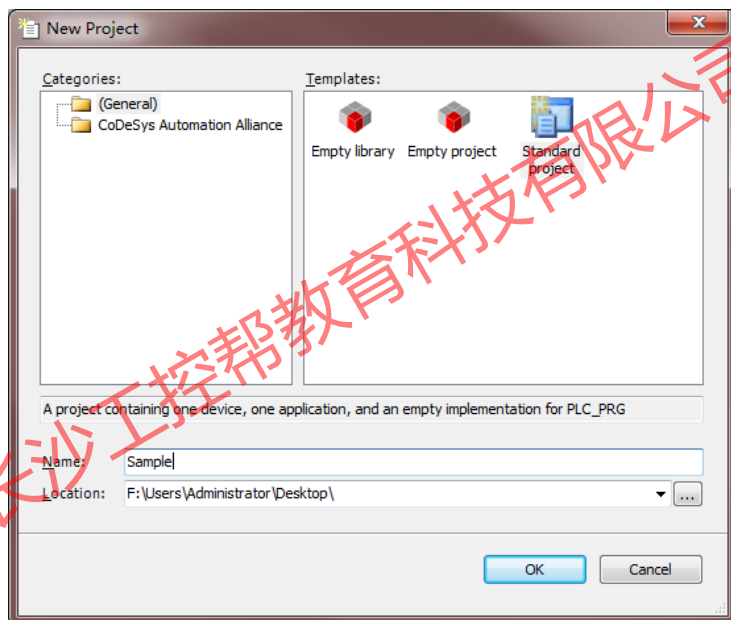


图 2.3 Sample 工程

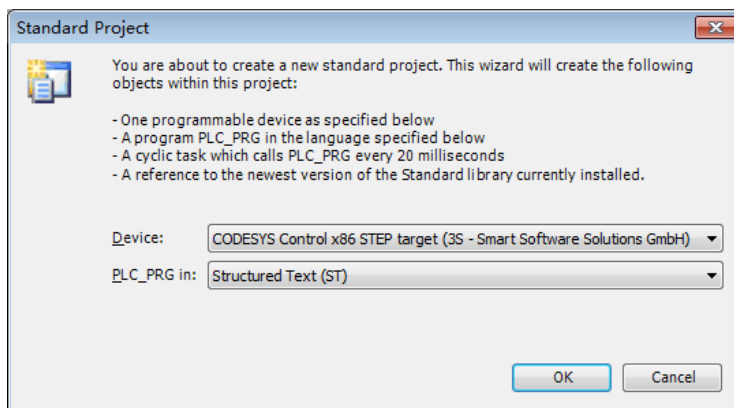


图 2.4 选择 STEP 控制器

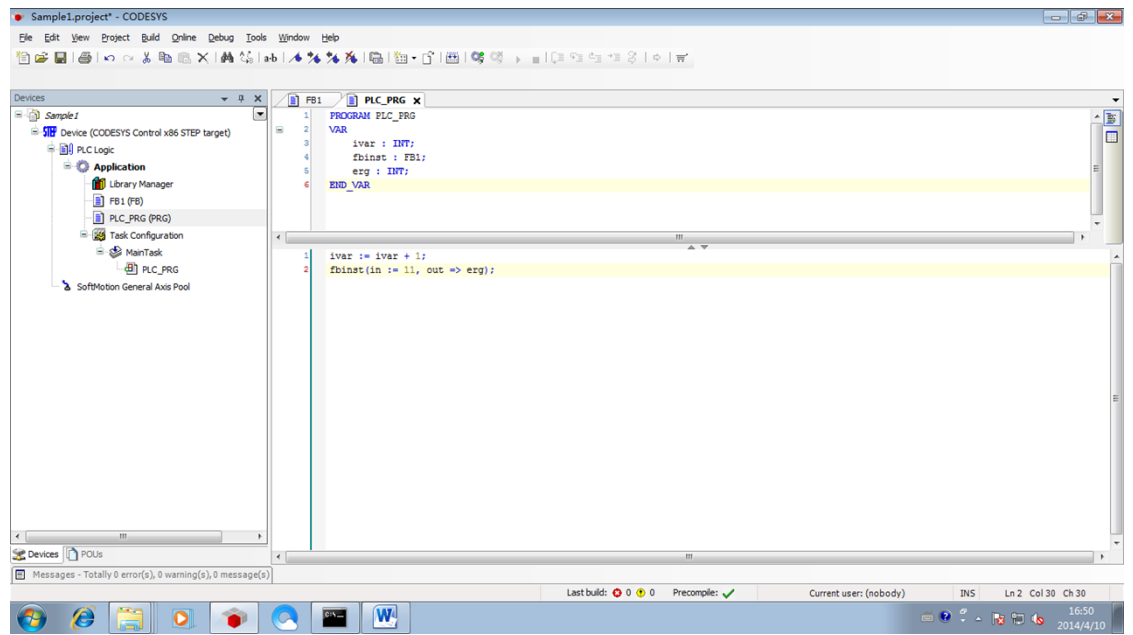


图 2.5 测试例程

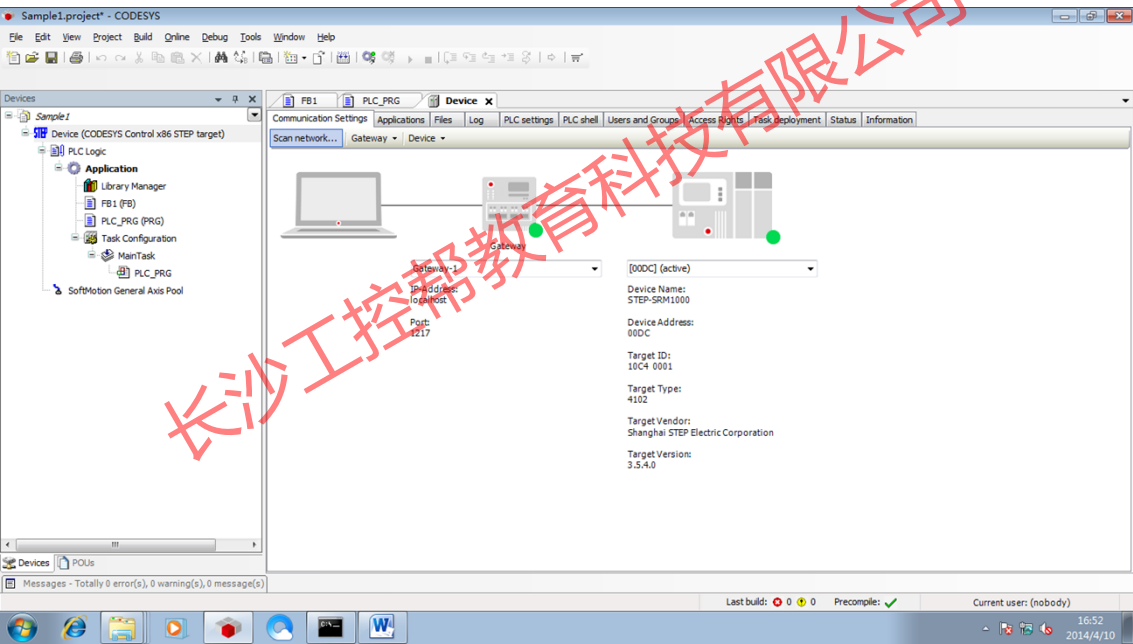


图 2.6 设置 Gateway

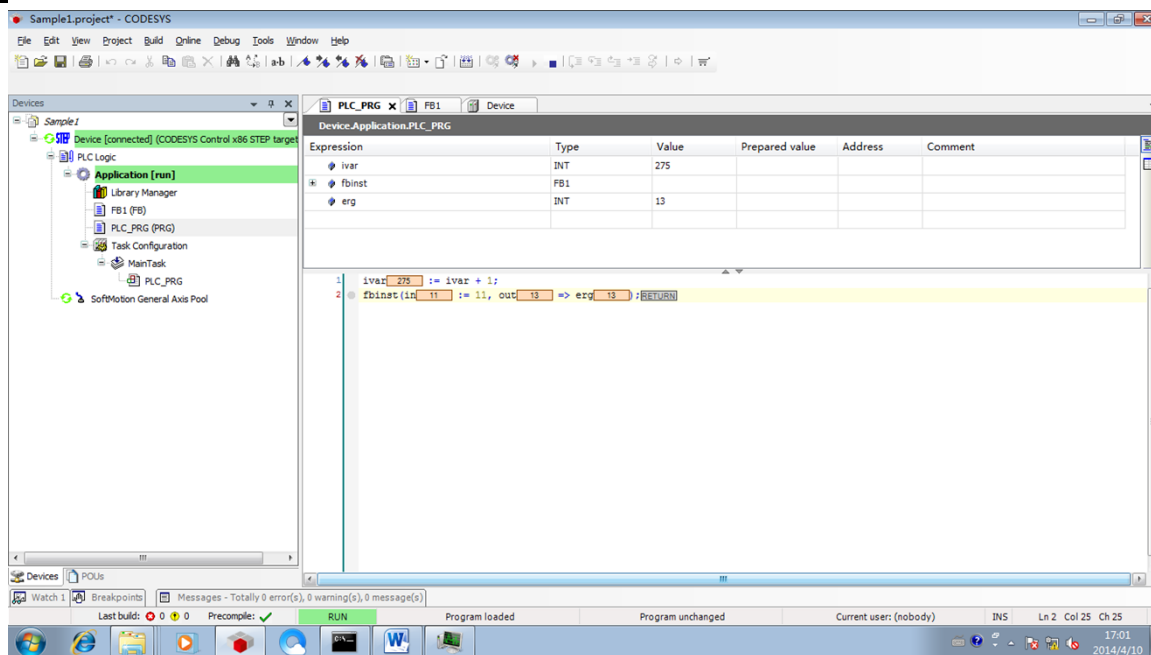


图 2.7 运行例程

若图中的例程正常运行起来，证明开发环境的搭建是十分成功的，用户可以进行下一步的配置与编程。

注：用户若不熟悉 CoDeSys 开发环境，那么 CoDeSys 自带的 Online Help 将会对用户有非常大的帮助。

长沙工控帮教育科技有限公司

长沙工控帮教育科技有限公司

第三章 设备组态及配置

机器人控制器 SRM1000 提供了多种总线设备的组态功能，用户可以根据自身的需求，选择不同的外部设备与控制器进行数据通讯，从而达成用户的目标。

3.1 EtherCAT 设备组态及配置

EtherCAT（以太网控制自动化技术）是一个开放架构，以以太网为基础的现场总线系统，其名称的 CAT 为控制自动化技术（Control Automation Technology）字首的缩写。

EtherCAT 的优势在于，高速度的以太网配合低成本通用网络器件，可以提供—个传输容量大、速度快、同步精度高的运动控制平台。

集成开发环境 CoDeSys 中，集成了 EtherCAT 设备的组态、配置等功能。

本节分别以我司提供的 STEP EtherCAT Servo Drives 伺服设备以及 SRE3010 数字量输入输出设备为例，介绍如何在 CoDeSys 中组态驱动设备和 IO 设备。

3.1.1 组态 STEP EtherCAT Servo Drive 伺服

为了能够组态、配置 STEP EtherCAT Servo Drive 伺服，需要安装从站设备。用户从供应商获得从站设备描述文件，并将其添加进入 CoDeSys 软件中，即可在 CoDeSys 中进行设备组态。

添加 Soft Motion PLC 设备：

打开 CoDeSys，新建一个工程，选择 CoDeSys Control from STEP 作为 SoftPLC 设备。

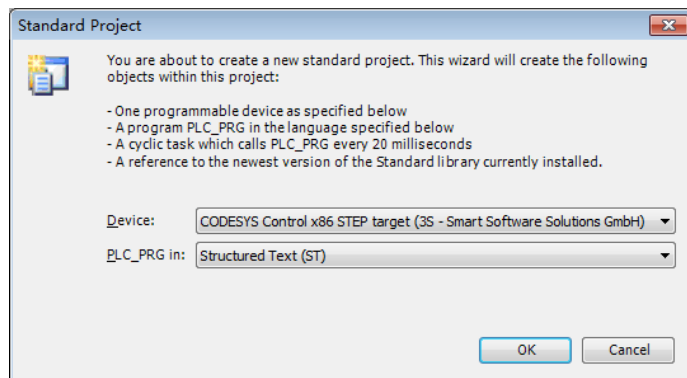


图 3.1 新建工程 Sample1

添加 EtherCAT Master 设备

在设备选项卡中，选中 Device(CoDeSys Control from STEP)，点击右键，选择 Add Device。

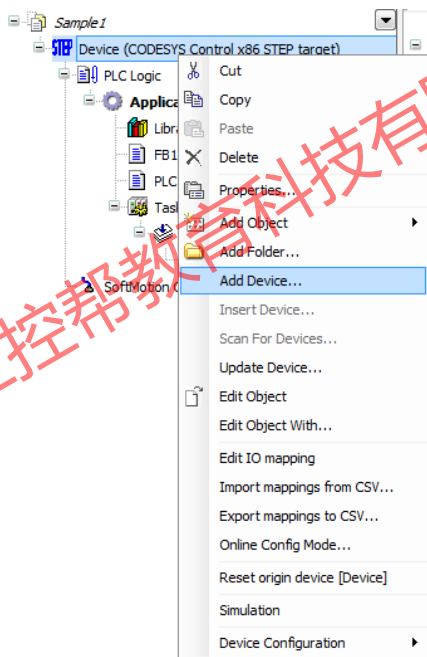


图 3.2 点击 Add Device

在弹出的 Add Device 窗口中，选择添加 Fieldbusses → EtherCAT → Master → EtherCAT Master。

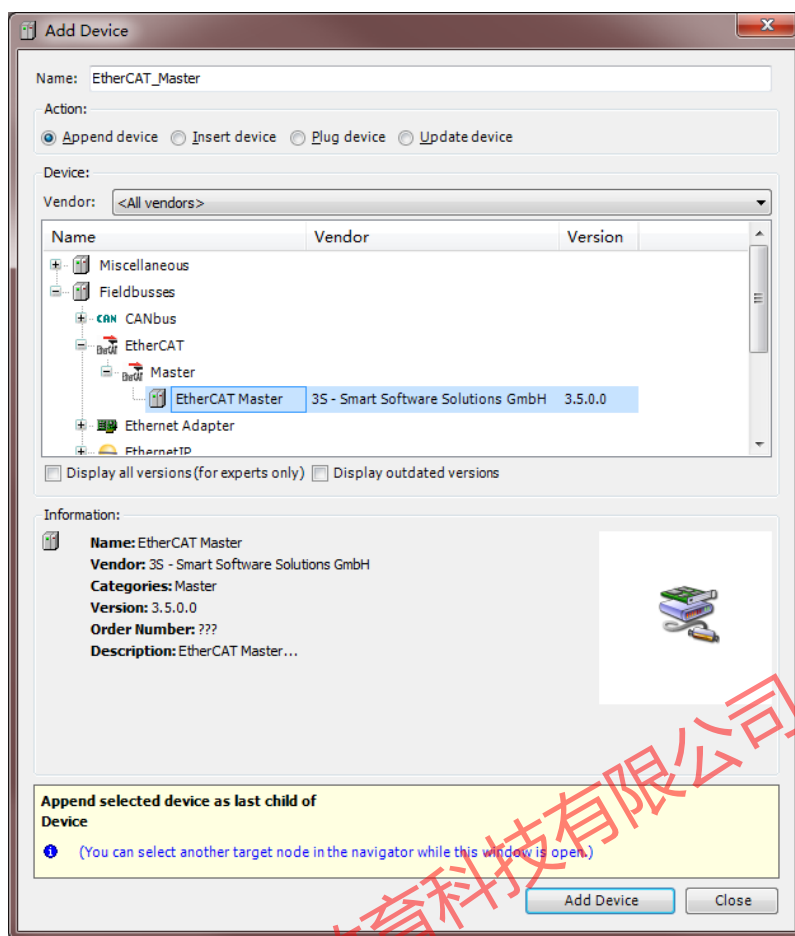


图 3.3 选择 EtherCAT Master

添加 EtherCAT Master 之后，整个设备选项卡出现了一些变化，Task Configuration 中自动添加了 EtherCAT_Master 任务；Library Manager 中自动添加了 IODrvEtherCAT.library 库。

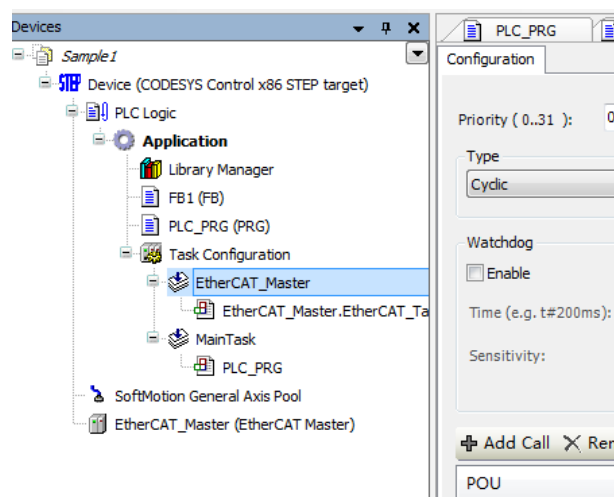


图 3.4 EtherCAT_Master 任务

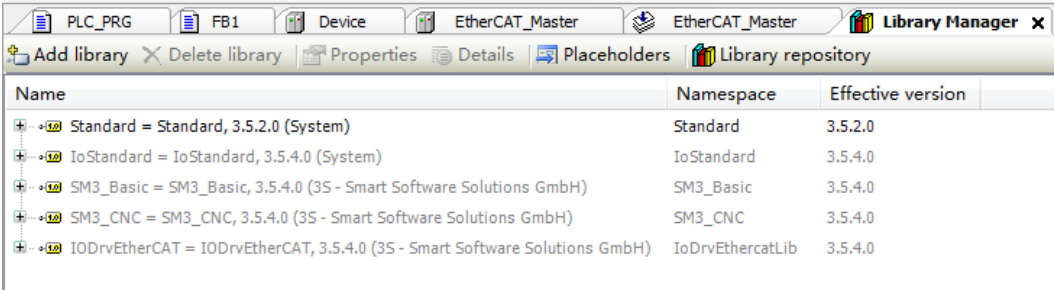


图 3.5 IODrvEtherCAT.library

此时，整个程序 Sample1 中包含了两个 Task，我们需要确认其中一个作为 Sample1 程序的主任务。

配置 EtherCAT Master:

在配置 EtherCAT 主站之前，需要在上位机中配置控制器所在网络的网关（Gateway），用户可以查询本说明书获得该配置方法。



图 3.6 配置主站

请用户配置好 Gateway，打开 EtherCAT_Master 设备，之后点击 按钮，弹出 Select Network Adapter 对话框。

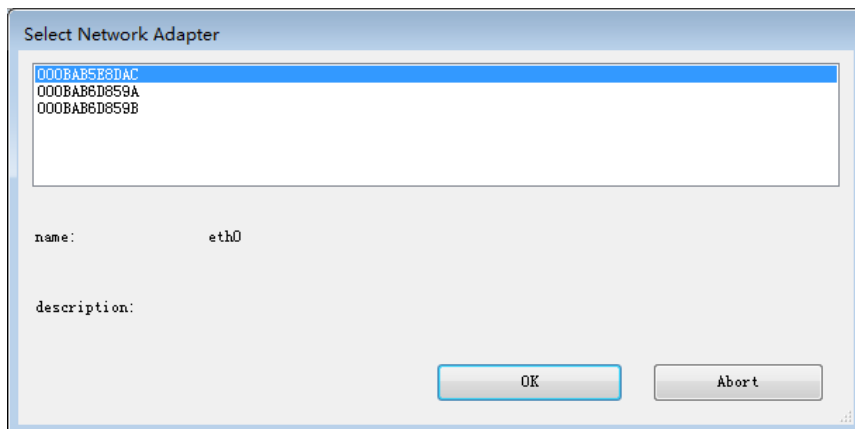


图 3.7 选择 EtherCAT 主站网卡

请在该对话框中选择 name 名为 eth0 的网卡设备，点击  按钮。

根据插补算法的周期选择分布式时钟 DC 的周期，在本说明文档中，DC 周期设为 1000us。再将 Auto Restart Slaves 勾选。如下图所示。

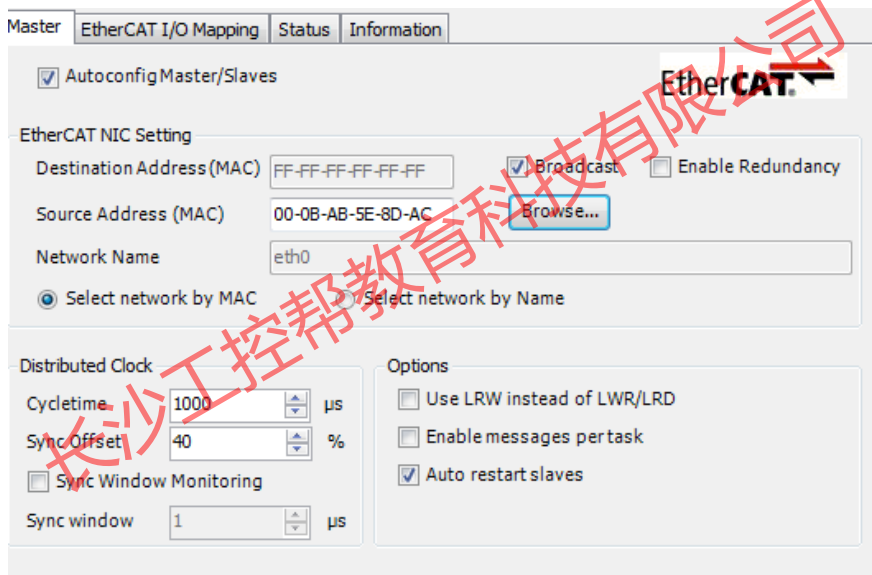


图 3.8 主站配置

添加 STEP EtherCAT 伺服：

选择 EtherCAT Master 设备，右键菜单中选择 Add Device。

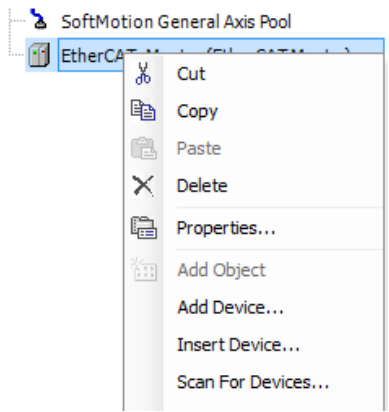


图 3.9 添加 Slave 设备

弹出 Add Device 对话框，在 Fieldbusses → EtherCAT → Slave 中选择 STEP EtherCAT Servo Drives，点击 **Add Device** 按钮。

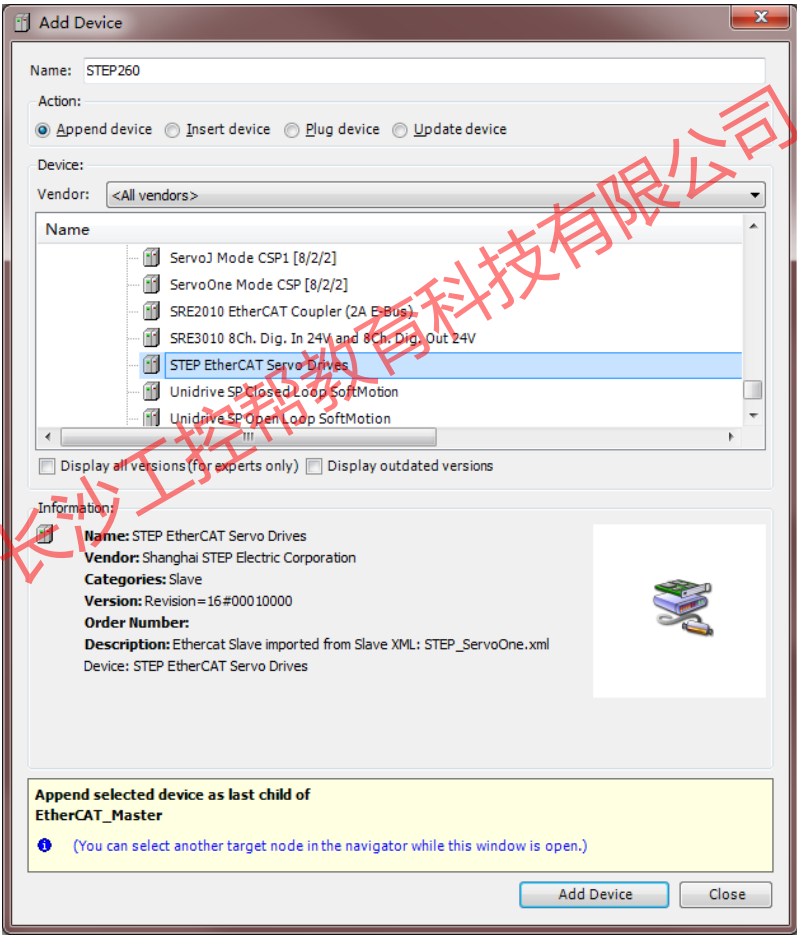


图 3.10 添加 STEP Drive

添加完毕后，设备选项卡将会增加 STEP EtherCAT 伺服、相应的 STEP 虚拟轴的图标。

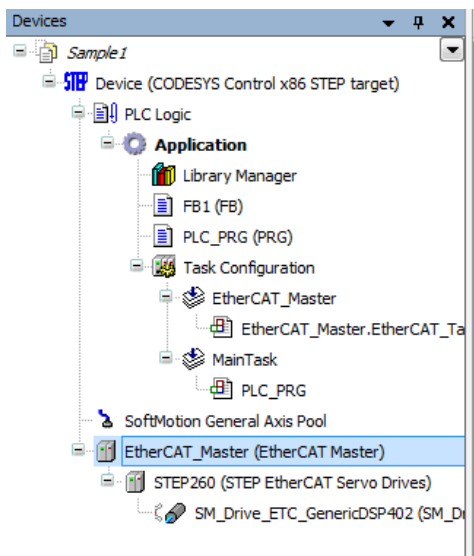


图 3.11 STEP 伺服及虚拟轴

配置 STEP EtherCAT 伺服：

双击 STEP260 设备，即可打开伺服设备的配置界面。

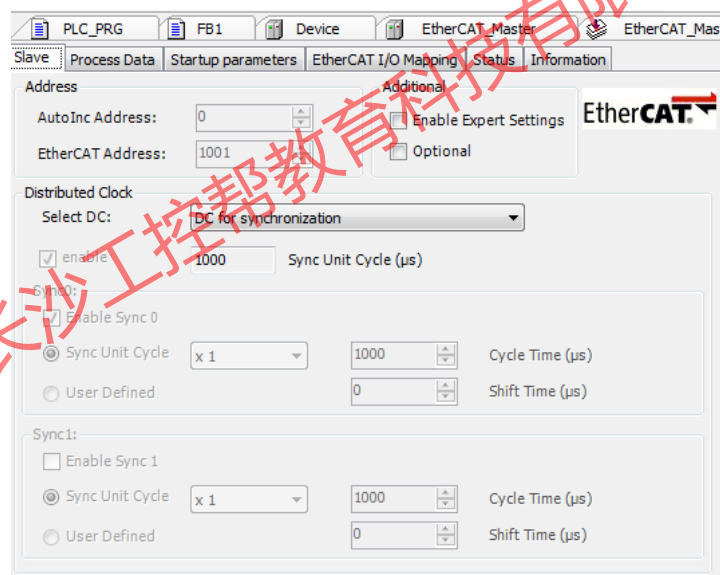


图 3.12 STEP260 设备配置

请打开分布时钟 DC。

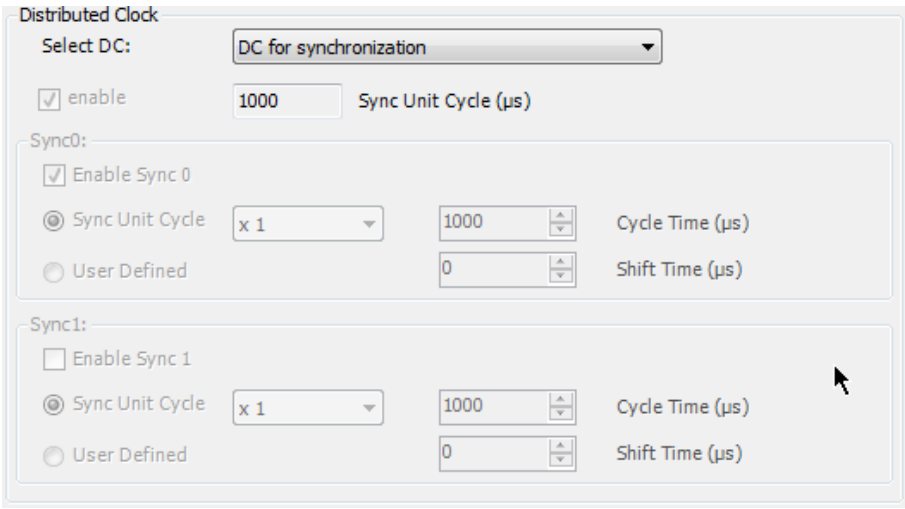


图 3.13 打开 DC

选项卡 EtherCAT I/O Mapping 中勾选 Always update variables。

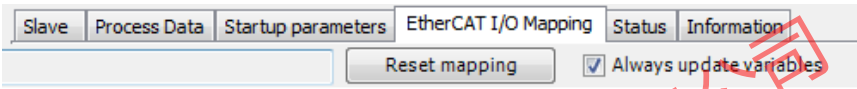


图 3.14 勾选更新选项

配置 STEP 虚拟轴

自动化设备（譬如机器人，机床等）的执行机构，大多是由电机驱动传动系统来实现运动的。那么电机到执行机构的映射就可以用下图表示：



虚拟轴的配置，就是用来配置电机到传动系统、传动系统到执行机构的映射关系的，跟许多伺服系统中的所提到的电子齿轮是类似的。故用户在使用本配置功能时，注意在伺服系统中要根据本功能的配置情况来配置电子齿轮功能。

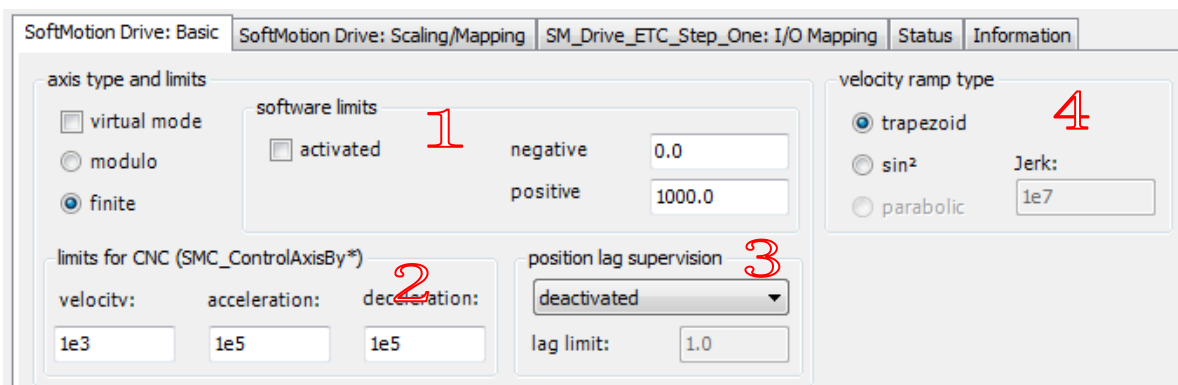


图 3.15 虚拟轴基础配置

Axis type and limits: 用来配置轴的虚拟状态、旋转模式、软限位。

Limits for CNC: 用来配置最大速度、最大加速度（只在某些特定 Function 下起作用）。

Position lag supervision: 用来激活 Position Lag 功能。

Velocity ramp type: 用来配置速度曲线（也只限定在某些 Function Block 下起作用）。

用户可根据自己设备的特殊的机械属性，来一一配置这些内容。

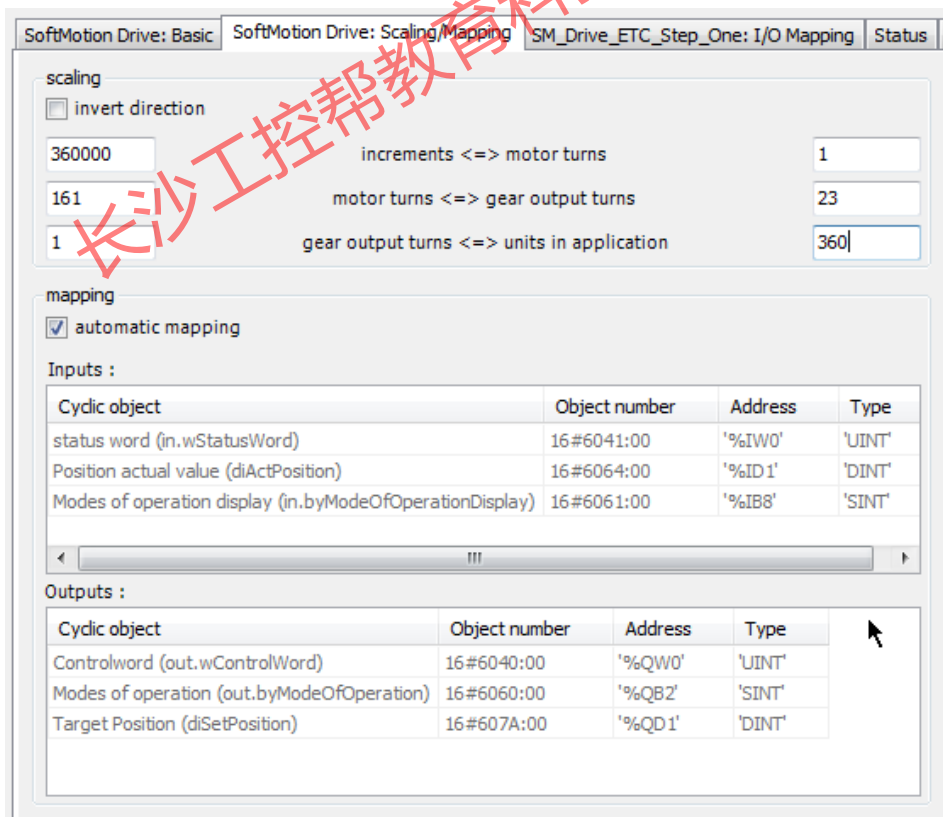


图 3.16 减速比与 PDO 映射

Scaling: 用来配置转向，以及传动系统的几何关系。

Mapping: 用来配置 PDO 对象字典与虚拟轴变量之间的映射。

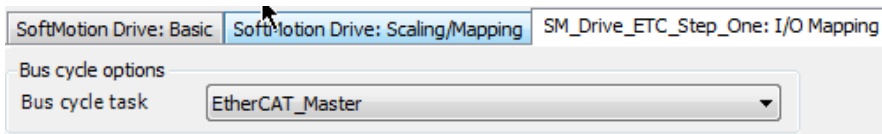


图 3.17 选择跟随主任务循环

Bus cycle Options: 确定了 Sample1 的主任务，要将虚拟轴匹配到该主任务上。

3.1.2 单轴运动控制实例

现在已经完整的将一个 EtherCAT 的伺服设备组态进入到了 Sample1 工程中。接下来，在此工程中添加一些运动控制的算法功能块和逻辑控制块，从而来实现一个单轴运动控制的实例。

添加 POU Sample

在设备选项卡中，右键选中 Application，选择添加 POU，名称为 Sample，类型为 Program，语言为 CFC。

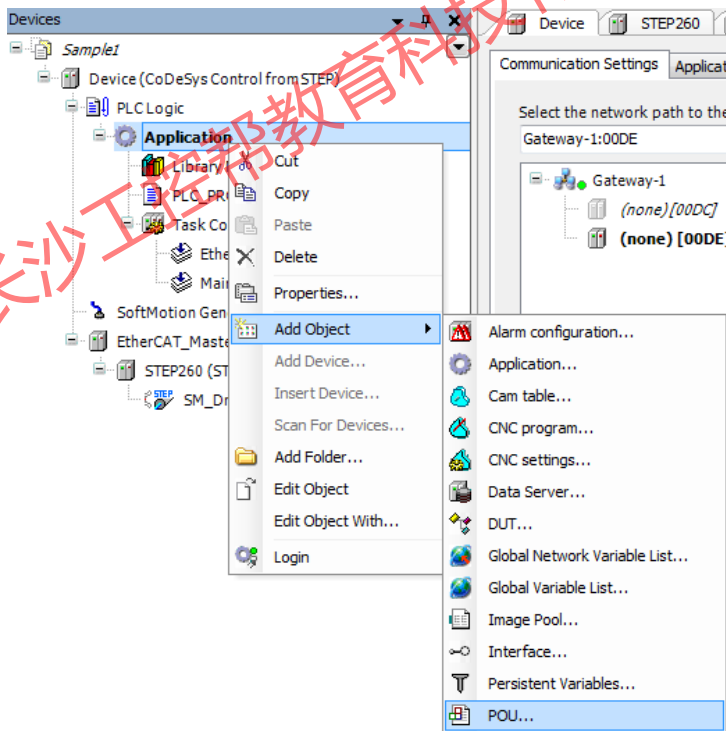


图 3.18 插入新的 POU

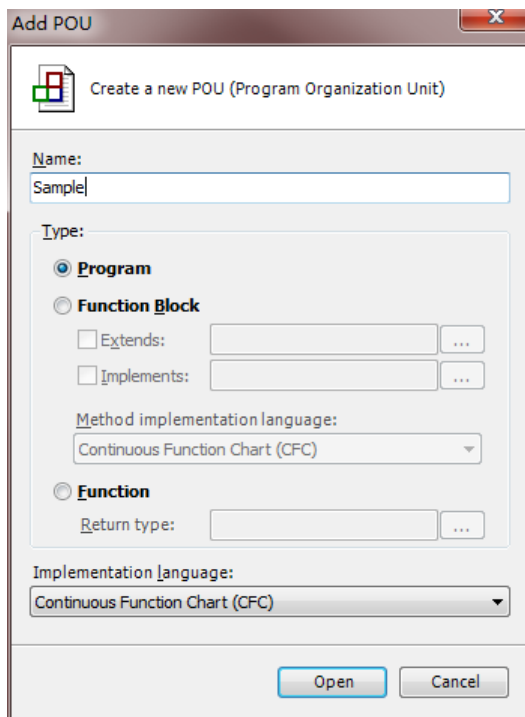


图 3.19 名为 Sample 的 Program

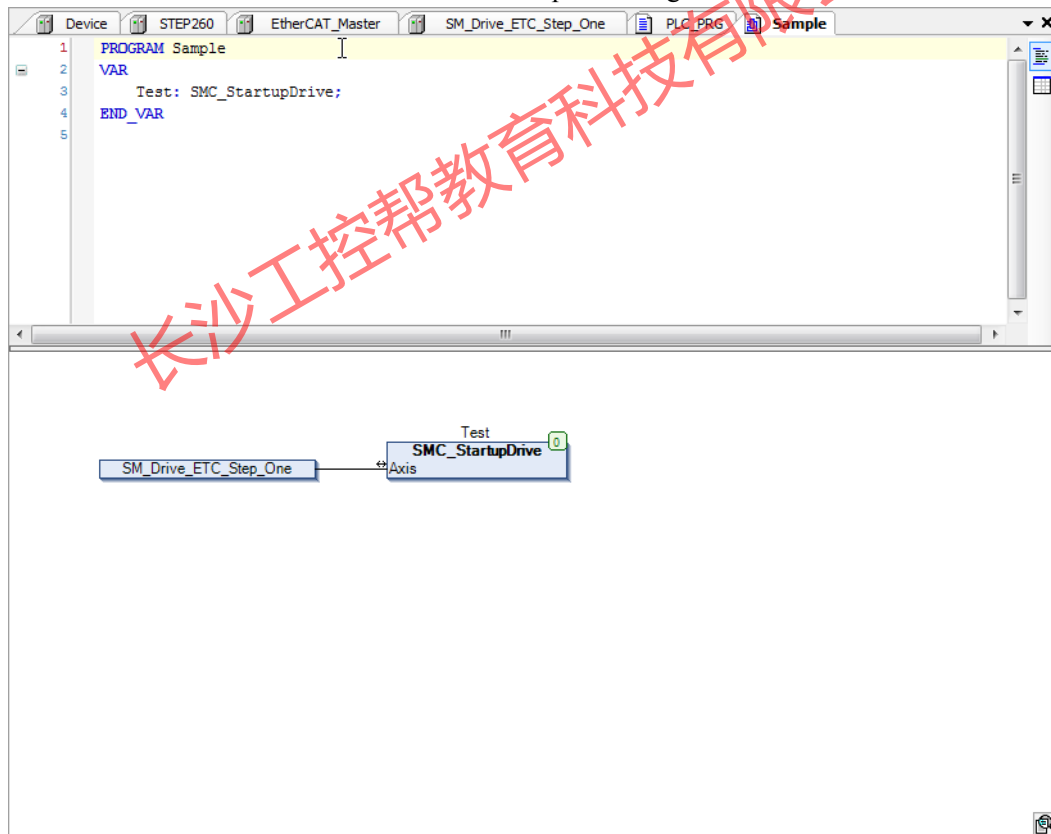


图 3.20 Sample 源码

Sample 源码如上图所示。其中 SMC_StartupDrive 为 CoDeSys 自带的 SM3_Basic.library 中集成的一个简单实例。

添加 Sample 到主任务

在编译程序之前，需要做的是将 Sample 添加到主任务 EtherCAT_Master 中。

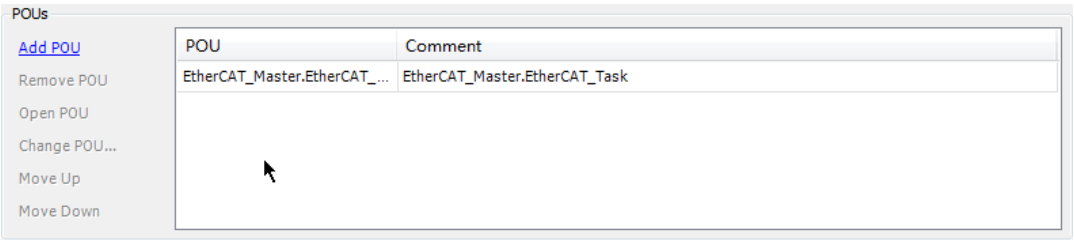


图 3.21 添加 Sample 到主任务

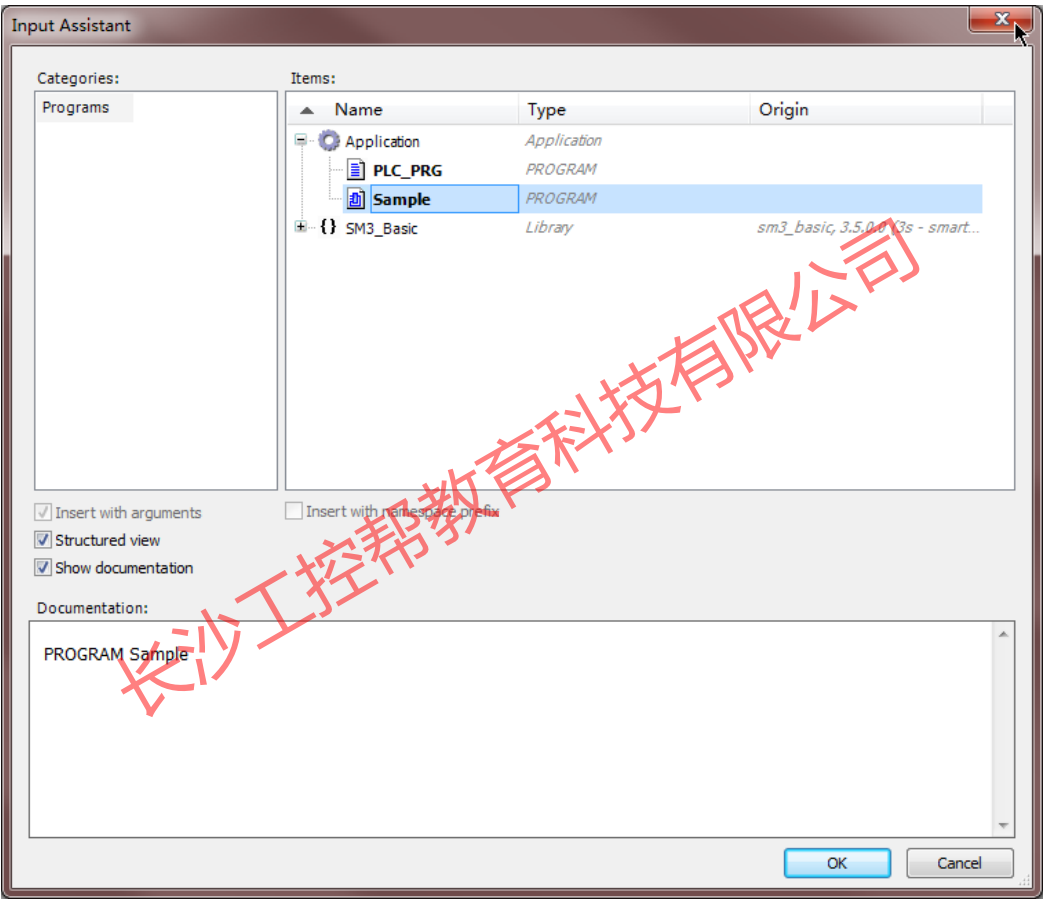


图 3.22 添加过程

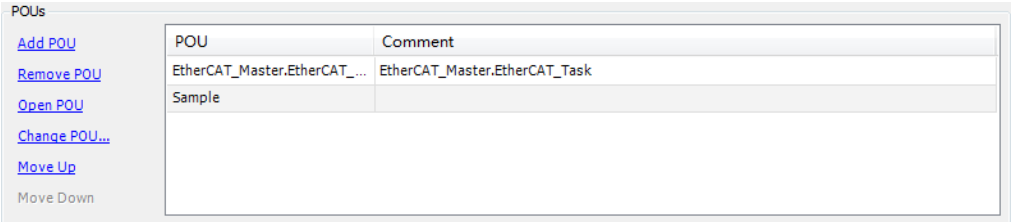
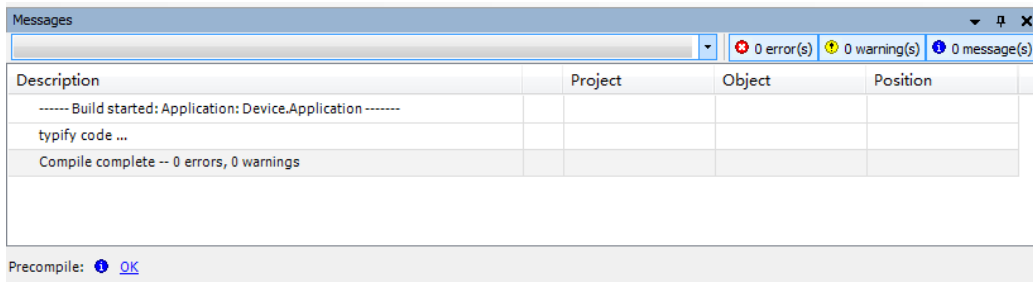


图 3.23 添加效果

编译并运行实例

单轴运动控制实例 Sample1 已经构建完毕，接下来需要做的就是对其进行编译，若编译正常通过，下载到控制器运行即可。

点击任务栏  图标，开始编译 Sample1，编译信息可在 Messages 窗口查看。



Watch 1				
Expression	Type	Value	Prepared value	Comment
Device.Application.Sample.Test	SMC_StartupDrive			
Axis	AXIS_REF_SM3			
mcma	MC_MoveAbsolute			
mcmv	MC_MoveVelocity			
mcp	MC_Power			
mcs	MC_Stop			
mcmr	MC_MoveRelative			
mch	MC_Home			
mcrcp	MC_ReadParameter			
mcrcbp	MC_ReadBoolParam...			
mcwcp	MC_WriteParameter			
mcwcbp	MC_WriteBoolParam...			
mcrs	MC_ReadStatus			
mcr	MC_Reset			
wDriveId	WORD	1		
fSetPosition	LREAL	0		
fActPosition	LREAL	0		
fSetVelocity	LREAL	0		
fActVelocity	LREAL	0		
smch	SMC_Homing			
mctp	MC_TouchProbe			
mcat	MC_AbortTrigger			
scm	SMC_SetControllerM...			
Trigger	TRIGGER_REF			
mcspl	MC_SetPosition			
mcj	MC_Jog			
mcj	SMC_Tech			

图 3.27 Watch1

从 Watch1 中可以看到，SMC_StartupDrive 实际上集成了许多的 PLCopen 标准函数。用户可以在这些函数中，任意的对虚拟轴（映射为伺服电机）进行使能，驱动电机正转、反转、相对位移、相对速度等不同方式的控制。

3.1.3 组态 STEP SRE3010 IO 设备

STEP SRE3010 数字量输入输出模块是基于 EtherCAT 总线协议开发的高速通用数字量 I/O，拥有 8 个输入节点以及 8 个输出节点，且不需要额外配置其他公司的总线耦合器产品，简化了接线。

新建 StepI0 工程

将 3.1.1 例程中的 STEP EtherCAT Servo Drives 设备更换为 STEP SRE3010 设备即可。

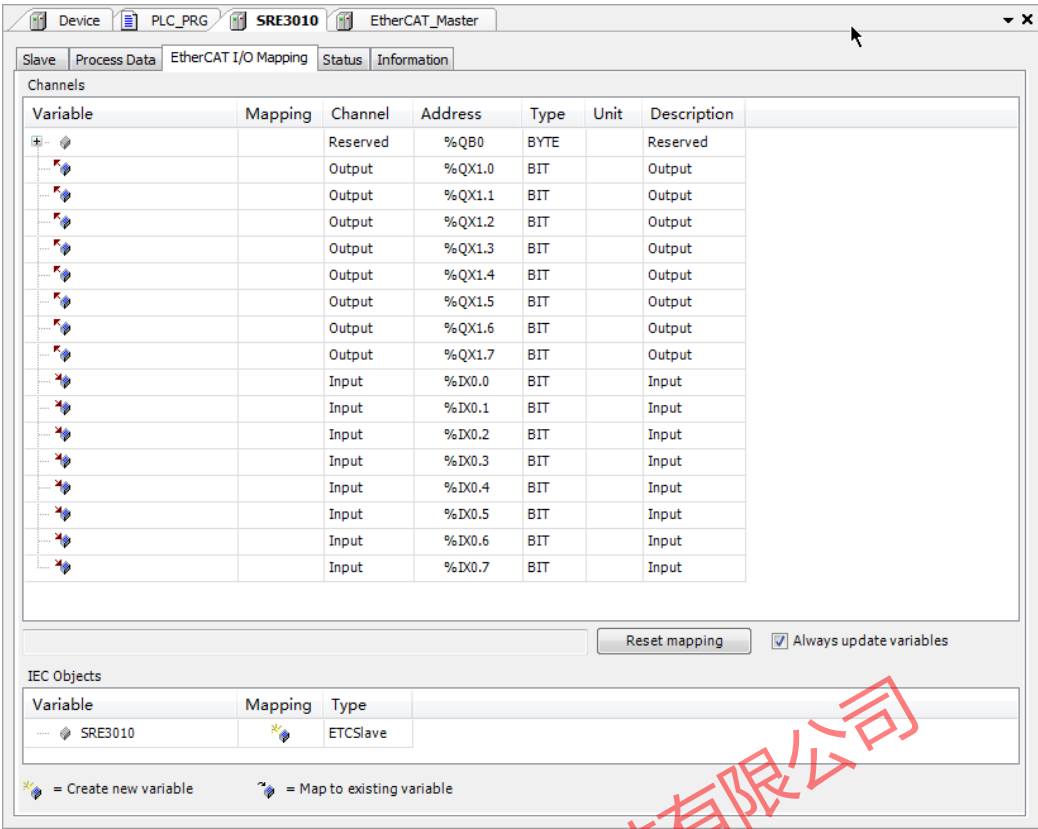


图 3.28 SRE3010

修改 PLC_PRG

先将 Task Configuration 中的 MainTask 删除，并将 PLC_PRG 添加到 EtherCAT_Master 任务中。

请参照下述代码修改 PLC_PRG 程序。

```
PROGRAM PLC_PRG
VAR
    iCnt: INT;
    timer : TON;
END_VAR
VAR_OUTPUT
    aOut : ARRAY [0..7] OF INT;
END_VAR
```

```
IF iCnt < 8 THEN
    {每个循环先清零}
    aOut[0] := 0;
    aOut[1] := 0;
    aOut[2] := 0;
    aOut[3] := 0;
    aOut[4] := 0;
    aOut[5] := 0;
    aOut[6] := 0;
    aOut[7] := 0;
```

```

{控制小于等于iCnt, 且差值为2的等差数列作为亮灯的Index}
aOut[iCnt] := 1;
IF iCnt > 1 THEN
    aOut[iCnt - 2] := 1;
END_IF
IF iCnt > 3 THEN
    aOut[iCnt - 4] := 1;
END_IF
IF iCnt > 5 THEN
    aOut[iCnt - 6] := 1;
END_IF
{每次iCnt要等0.2秒才累加1}
timer(IN := TRUE, PT := T#0.2S);
IF timer.Q THEN
    iCnt := iCnt + 1;
    timer(IN := FALSE);
END_IF
ELSIF iCnt < 15 AND iCnt >= 8 THEN
    {每次循环先清零}
    aOut[0] := 0;
    aOut[1] := 0;
    aOut[2] := 0;
    aOut[3] := 0;
    aOut[4] := 0;
    aOut[5] := 0;
    aOut[6] := 0;
    aOut[7] := 0;

    {控制大于8的小于16的Index跑完尾数灯}
    IF iCnt < 10 THEN
        aOut[iCnt - 6] := 1;
        aOut[iCnt - 4] := 1;
        aOut[iCnt - 2] := 1;
    END_IF
    IF iCnt >= 10 AND iCnt < 12 THEN
        aOut[iCnt - 6] := 1;
        aOut[iCnt - 4] := 1;
    END_IF
    IF iCnt >= 12 AND iCnt < 14 THEN
        aOut[iCnt - 6] := 1;
    END_IF
    timer(IN := TRUE, PT := T#0.2S);
    IF timer.Q THEN
        iCnt := iCnt + 1;
        timer(IN := FALSE);
    END_IF
ELSE
    aOut[0] := 0;
    aOut[1] := 0;
    aOut[2] := 0;
    aOut[3] := 0;
    aOut[4] := 0;
    aOut[5] := 0;

```

```
aOut[6] := 0;
aOut[7] := 0;
iCnt := 0;
END_IF
{此处将输出映射给组态的设备IO的地址}
%QX1.0 := aOut[0].0;
%QX1.1 := aOut[1].0;
%QX1.2 := aOut[2].0;
%QX1.3 := aOut[3].0;
%QX1.4 := aOut[4].0;
%QX1.5 := aOut[5].0;
%QX1.6 := aOut[6].0;
%QX1.7 := aOut[7].0;
```

编译并运行实例

参照 3.1.2 小节的编译及运行，将程序 StepIO 运行起来。SRE3010 的输出灯会按照一定的顺序依次亮起。

3.2 总线同步模块

在许多应用场合，可能会涉及到多于一个的控制器来对不同的执行机构来进行控制，从而达到用户所期望的运动控制的目标。譬如在工业机器人领域，就会设计到多个机器人协同作业来完成一些比较复杂的焊接、装配等应用场景。

从运动控制器的角度来讲，实现上述目的，可以有多种方法：

- 单个控制器控制多个机器人
- 多个控制器控制多个机器人

从控制的逻辑难易、拓扑结构的简易程度等多个方面来讲，第二个方案无疑更具实施性。用户可以添置总线同步模块，来进行同步、实时的多个控制器之间的数据的交互。

总线同步模块，就是 EL6692 模块。

3.2.1 EL6692 总线同步模块

EL6692，实际上是 EtherCAT 桥接端子模块，主要实现两个功能：

- 不同 EtherCAT 主站之间的数据交换
- 同步不同 EtherCAT 主站系统内的分布式时钟

连接两个 EtherCAT 系统的 EL6692 模块需要独立供电，以防止数据通讯由于其中某个系统断电而中断。

EL6692 通过映射 PDO 对象字典对象，包括 16 位的 SYNC 输入，以及用户自定义的数字量/模拟量的输入/输出（单个传输方向上最多传送 480Bytes），来进行多个 EtherCAT 主站之间的同步通讯。

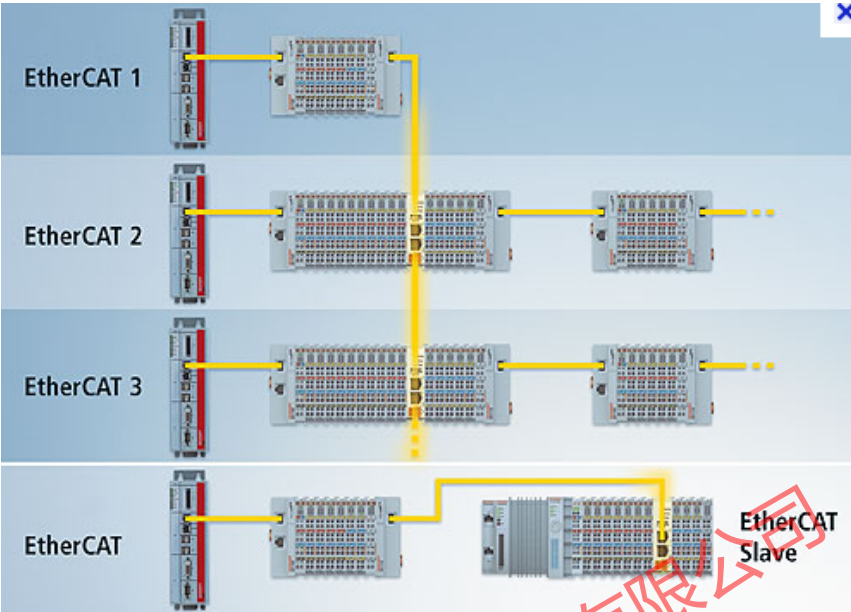


图 3.29 EL6692 使用拓扑图

3.2.2EL6692 配置及使用

按照步骤创建一个例程。在该例程中，将使用一个 EL6692 模块，在两个控制器之间传递数字量和模拟量，拓扑结构图如下：

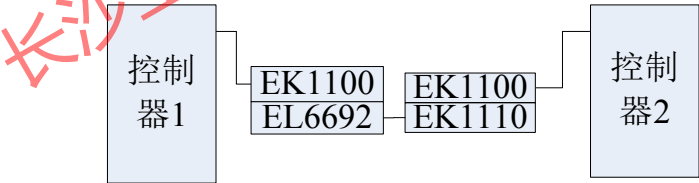


图 3.30 例程拓扑图

工程中添加两个控制器

新建一个 CoDeSys 工程，选择 Control From STEP，如下图：

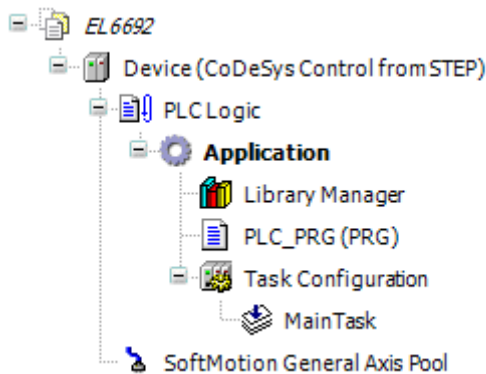


图 3.31 新建工程 EL6692

单击 EL6692，右键菜单选择 Add Device，添加另一个 Control From STEP。同时将新添加的 PLC 重命名为 Primary，原有的 PLC 命名为 Secondary，如下图。

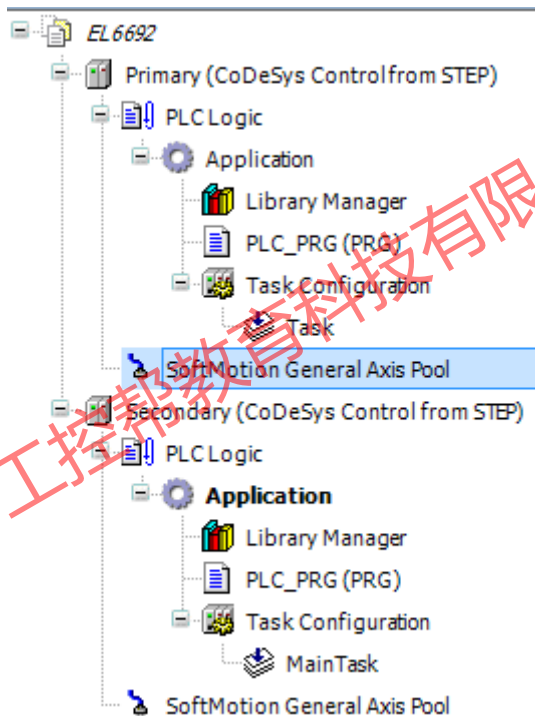


图 3.32 添加新的 PLC

添加 EL6692

用户在了解了 EtherCAT 设备的组态过程之后，能够迅速的按照下图所示的组态结果来进行组态。

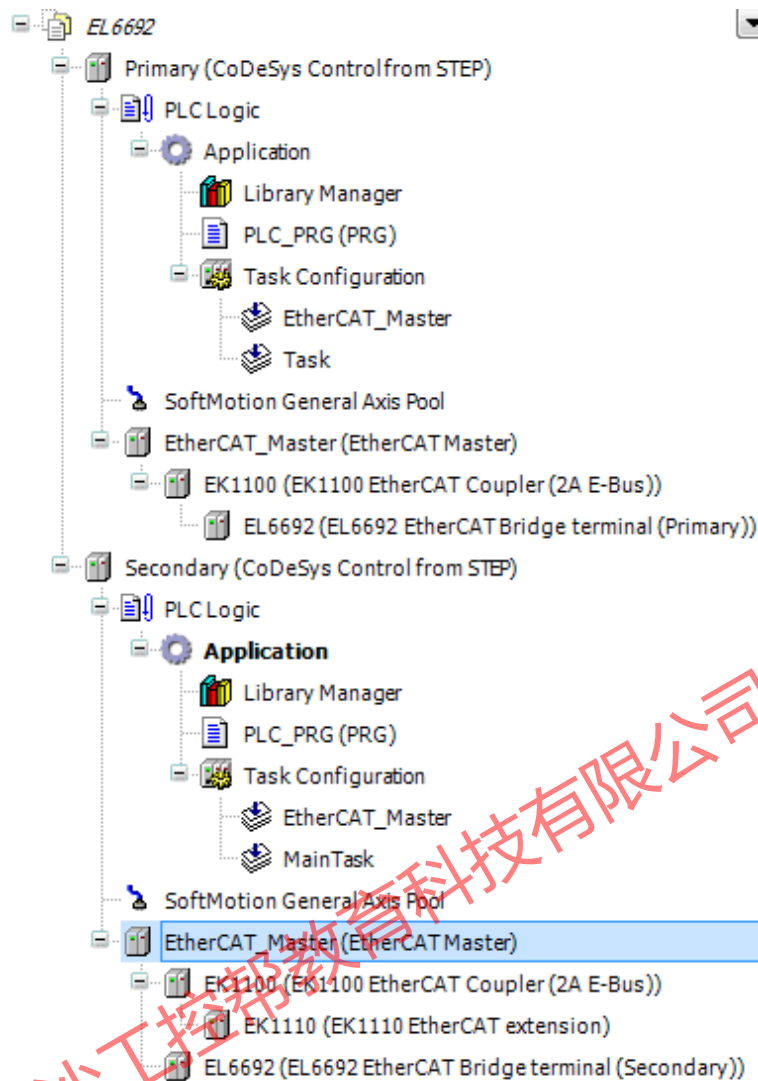


图 3.33 组态结果

EtherCAT_Master、Primary、Secondary 等设备的配置方法及内容，之前有详细介绍，用户需要按照这些指导来进行设备的组态配置。

其中 Primary 和 Secondary 对 EtherCAT_Master 中 MAC 地址的配置，应该按照相对应的 PLC 的 MAC 地址来配置。

配置 Primary EL6692

The screenshot shows the 'Expert Process Data' configuration window for a Primary EL6692. The interface includes several tabs: Slave, Expert Process Data (selected), Process Data, Startup parameters, EoE settings, EtherCAT I/O Mapping, Status, and Information. The 'Address' section shows 'AutoInc Address' set to -1 and 'EtherCAT Address' set to 1002. The 'Additional' section has 'Enable Expert Settings' checked. The 'Distributed Clock' section shows 'Select DC' set to 'DC-Synchron', 'enable' checked, and 'Sync Unit Cycle (µs)' set to 1000. The 'Sync0' section shows 'Enable Sync 0' checked, 'Sync Unit Cycle' set to 'x 1', 'Cycle Time (µs)' set to 1000, and 'Shift Time (µs)' set to 0. The 'Sync1' section shows 'Enable Sync 1' unchecked, 'Sync Unit Cycle' set to 'x 1', 'Cycle Time (µs)' set to 1000, and 'Shift Time (µs)' set to 0. The 'Startup checking' section has 'Check Vendor ID' and 'Check Product ID' both checked. The 'Timeouts' section shows 'SDO Access' set to 2000 ms, 'I -> P' set to 3000 ms, and 'P -> S / S -> O' set to 10000 ms. The 'DC cyclic unit control: assign to local µC' section has 'Cyclic Unit' unchecked, 'Latch Unit 0' unchecked, and 'Latch Unit 1' checked. The 'EtherCAT' logo is visible in the top right corner.

图 3.34 Primary 下 EL6692 配置 (1)

配置要点:

- 打开 DC-Sync
- 勾选 Enable Expert Settings

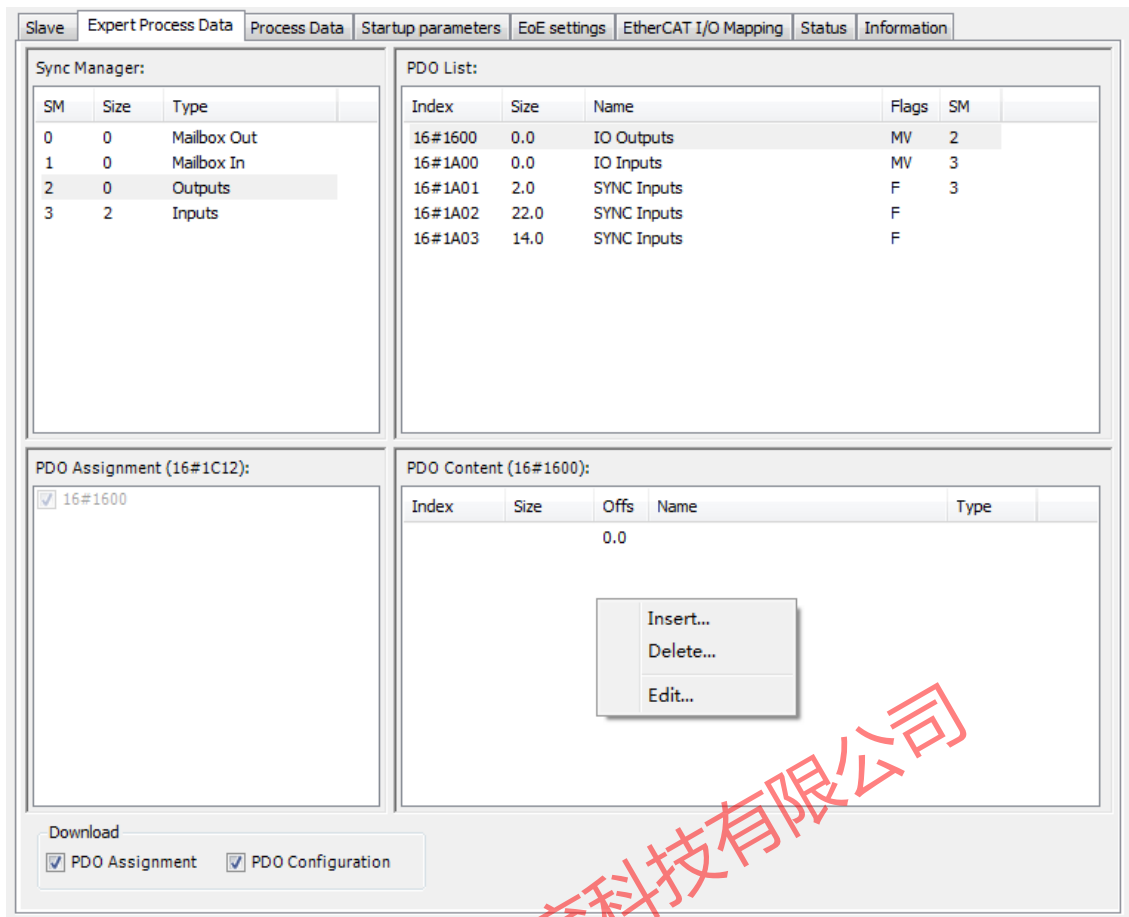


图 3.35 Primary 下 EL6692 配置（2）

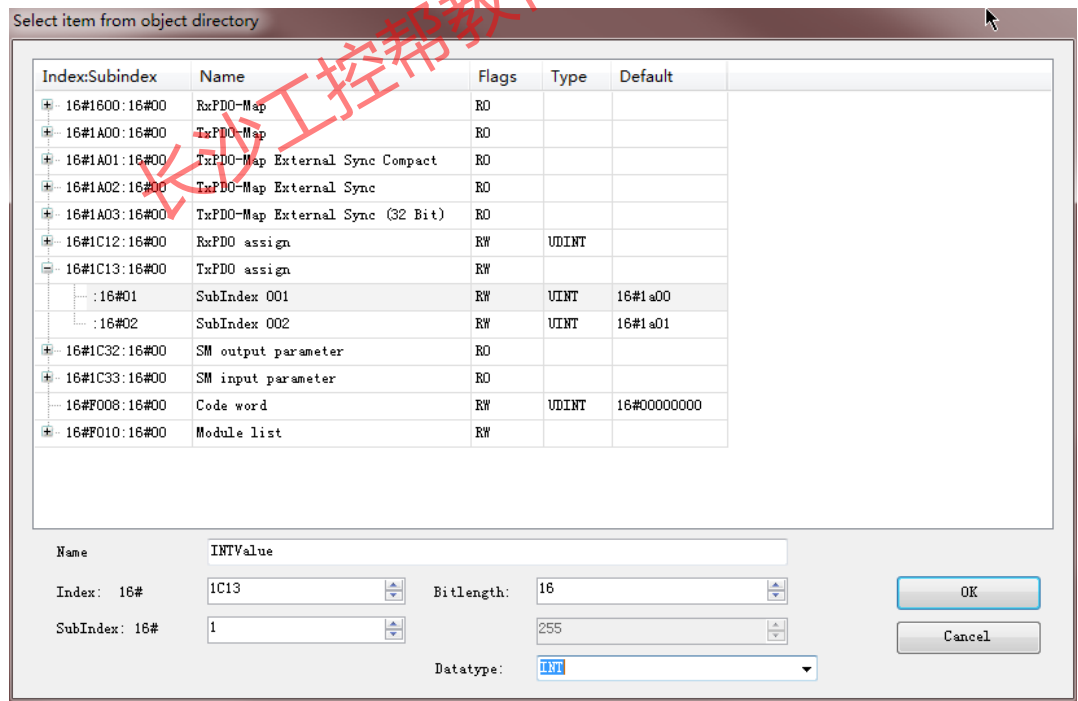


图 3.36 插入 PDO 映射

Index	Size	Offs	Name	Type
16#1C12:01	2.0	0.0	INTValue1	INT
16#1C12:01	4.0	2.0	REALValue	REAL
		6.0		

图 3.37 PDO 映射

配置要点：

在本应用场景中，Primary 作为主控制器，向 Secondary 从控制器发送数据。

所以

- PDO List 中选择 16#1600 IO Outputs
- PDO Content 中插入 16#1C12 RxPDO，在窗口下部的 Name 和 DataType 选择图 3.36 中的内容
- 再次插入 RxPDO，Name 和 DataType 填 REALValue 与 REAL

Secondary EL6692 配置

同样，选择 DC-Sync，勾选 Enable Expert Settings。

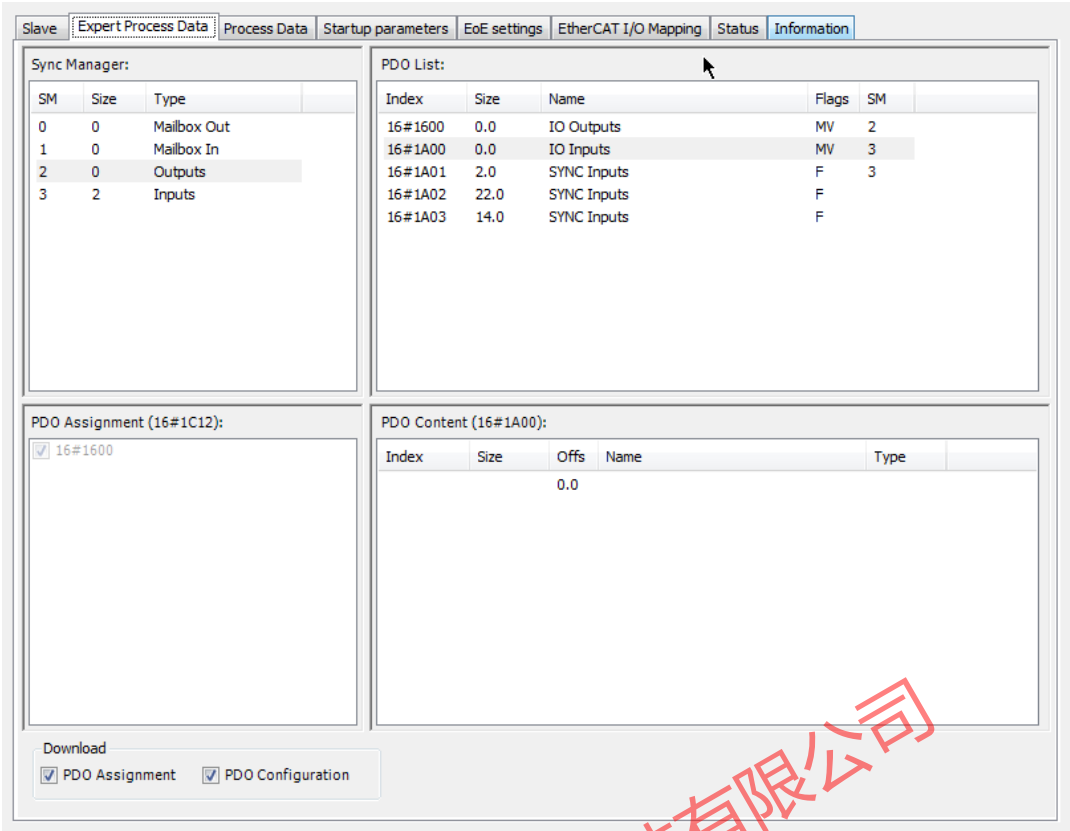


图 3.38 Secondary EL6692 配置（1）

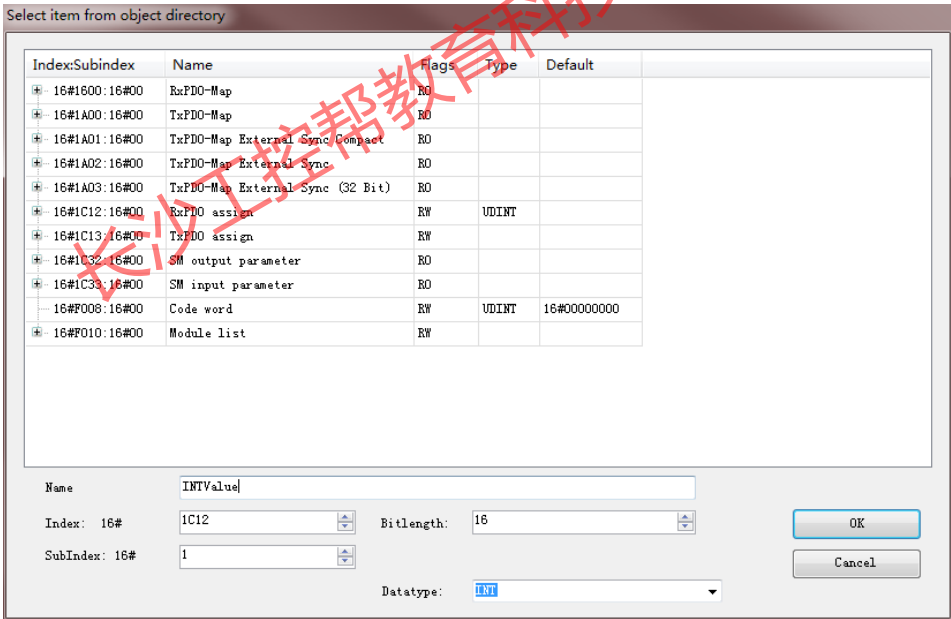


图 3.39 Secondary EL6692 配置（2）

Index	Size	Offs	Name	Type
16#1C13:01	2.0	0.0	INTValue	INT
16#1C13:01	4.0	2.0	REALValue	REAL
		6.0		

图 3.40 PDO 映射

配置要点：

根据 Primary EL6692 的配置，做如下配置。所以

- PDO List 中选择 16#1600 IO Outputs
- PDO Content 中插入 16#1C13 TxPDO，在窗口下部的 Name 和 DataType 选择图 3.39 中的内容
- 再次插入 TxPDO，Name 和 DataType 填 REALValue 与 REAL

变量映射及程序编写

Primary PLC 中，打开 PLC_PRG，代码输入如下：

```
PROGRAM PLC_PRG
VAR_OUTPUT
  iOUT : INT;
  rOUT : REAL;
END_VAR
```

```
iOut := iOut + 1;
rOut := rOut + 1.1;
```

并在 EL6692 设备的 EtherCAT I/O Mapping 选项卡中，填入如下变量：

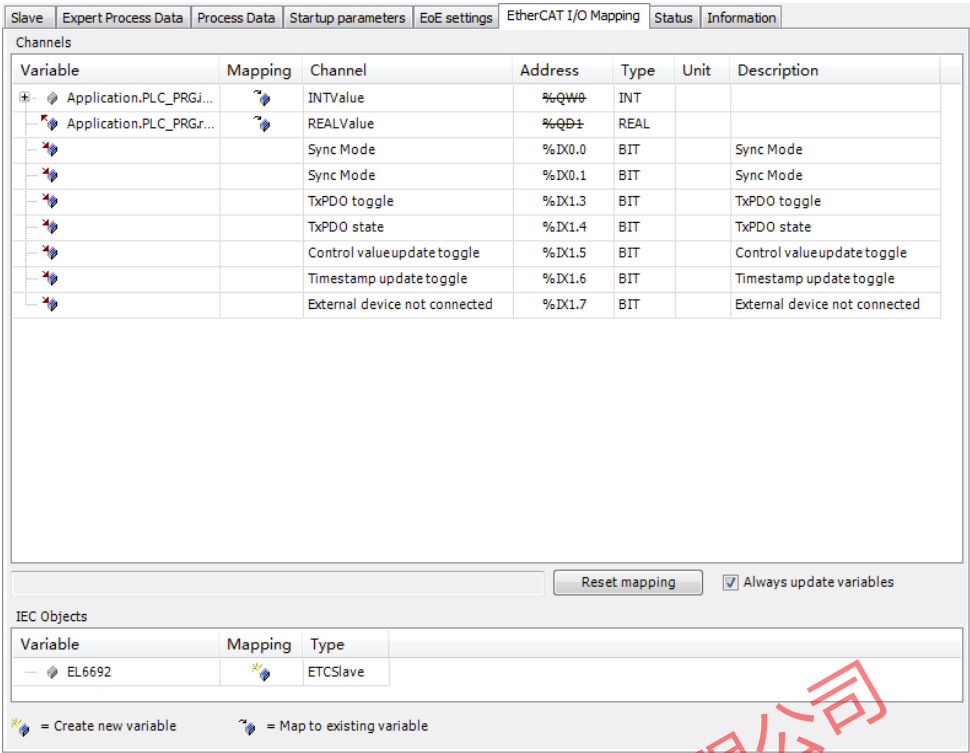


图 3.41 变量映射

Secondary PLC 中，也是如此。

```
PROGRAM PLC_PRG
VAR_INPUT
  iIN : INT;
  rIN : REAL;
END_VAR
```

长沙工控帮教育科技有限公司

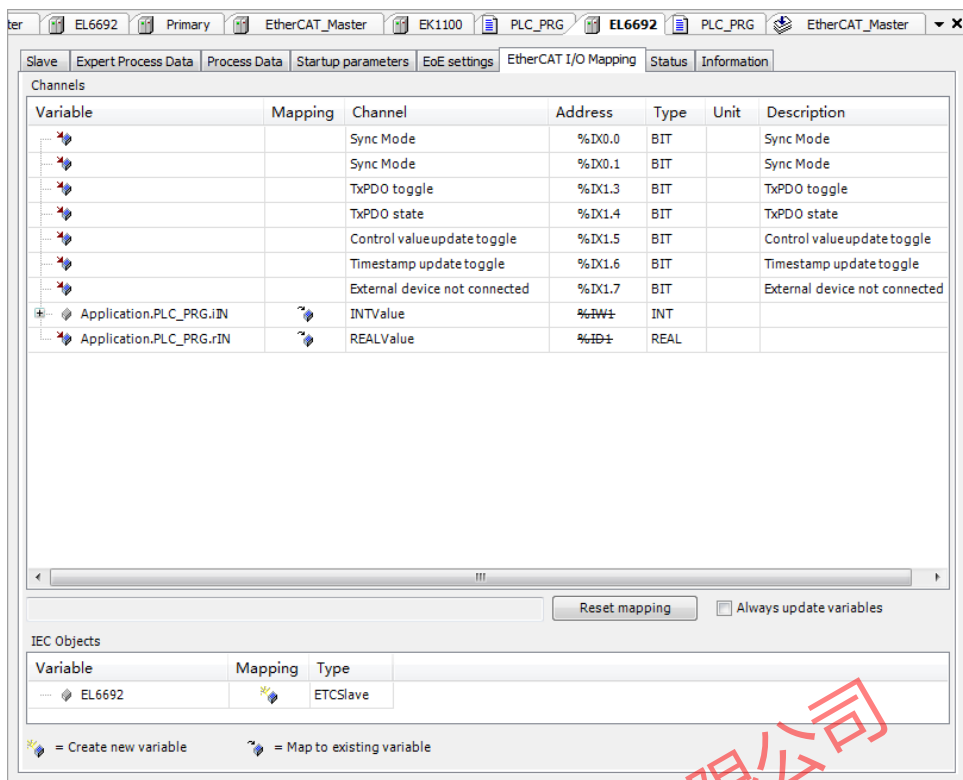


图 3.42 变量映射

运行及查看

将两个程序分别下载到两个控制器中，并运行之。

用户可以在 Secondary 的 PLC_PRG 中查看 iIN 以及 rIN 的值的变化的。

3.3 Profibus 设备组态及配置

STEP 控制器 SRM1000 对 Profibus 总线的支持，是通过在 EtherCAT 总线上挂接 Beckhoff 的 Profibus 总线耦合器模块 EL6731 来实现的。

3.3.1 组态 Profibus 模块 EL6731

在 EK1100 下添加 EL6731 设备时，需要添加两层，

- 一层是 EL6731(EL6731 PROFIBUS DP Master)（请注意，选择设备制造商 (Vendor) 为 3S 公司的 EL6731 模块，如红圈内所示）
- 第二层是 EL6731_PB_Master(EL6731-PB Master)

本例中，Profibus 的从站为 Beckhoff 的 BK3120 耦合器，BK3120 下面挂载几个 Beckhoff 的 KL 系列的 IO。

组态结果如下图所示：

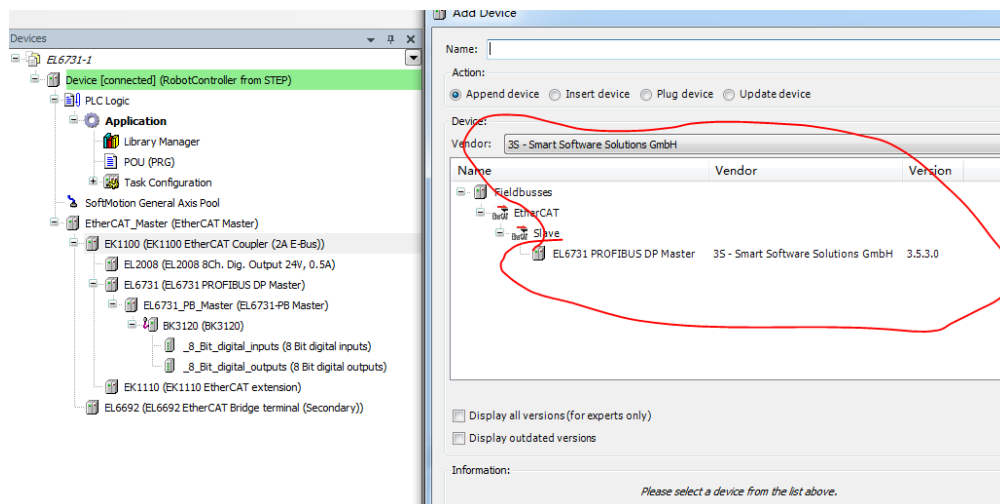


图 3.43 Profibus 组态结果

3.3.2 运行

组态完毕之后，将程序下载进入到控制器 SRM1000，运行。如下图：

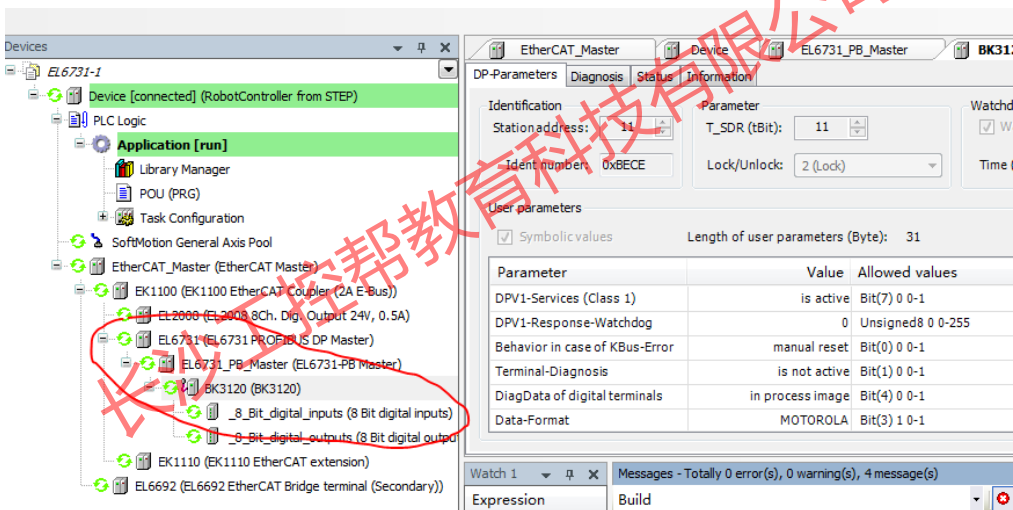


图 3.44 运行图

3.4 CANopen 设备组态及配置

CANopen 是一种架构在控制局域网络（Control Area Network, CAN）上的高层通讯协定，包括通讯子协定及设备子协定常在嵌入式系统中使用，也是工业控制常用到的一种现场总线。通俗讲，CANopen 实现了应用层的协议，而底层 CAN 协议实现了物理层和链路层的协议。

STEP 控制器 SRM1000 提供了一个标准 CAN 口，可作为 CANopen 网络的主站，与包括远程 IO、CANopen 伺服驱动等在内的 CANopen 设备通讯。

使用某公司的 CANopen 远程 IO 设备 RP2D-1608C1，作为 CANopen 从站，来介绍如何对 CANopen 设备组态与配置。

注：本例只是针对设置的应用场景所设计的，用户在配置组态设备时，应按照自身所面对的应用场景，合理设置参数，以保证用户设备的使用安全。

3.4.1 组态 CANopen 设备

此例中，远程 IO 设备的拨码开关拨动的位置为，设备栈号为 127，波特率为 1M。

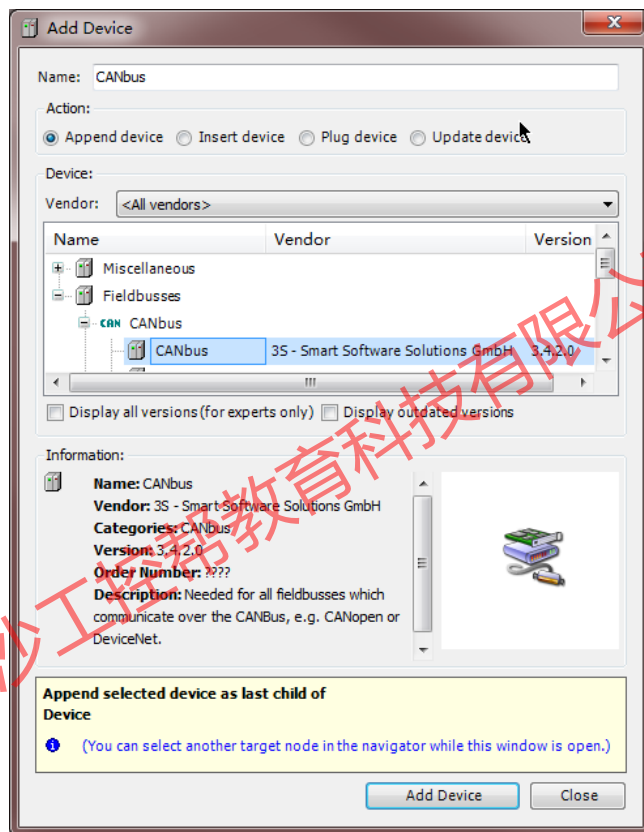


图 3.45 添加 CANbus 设备

打开 CANbus，将网络号（Network）设为 0，波特率（Baudrate）设为最高，1M。

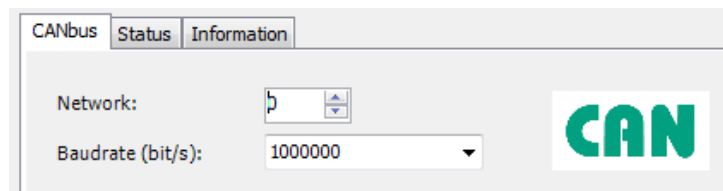


图 3.46 CANbus 配置

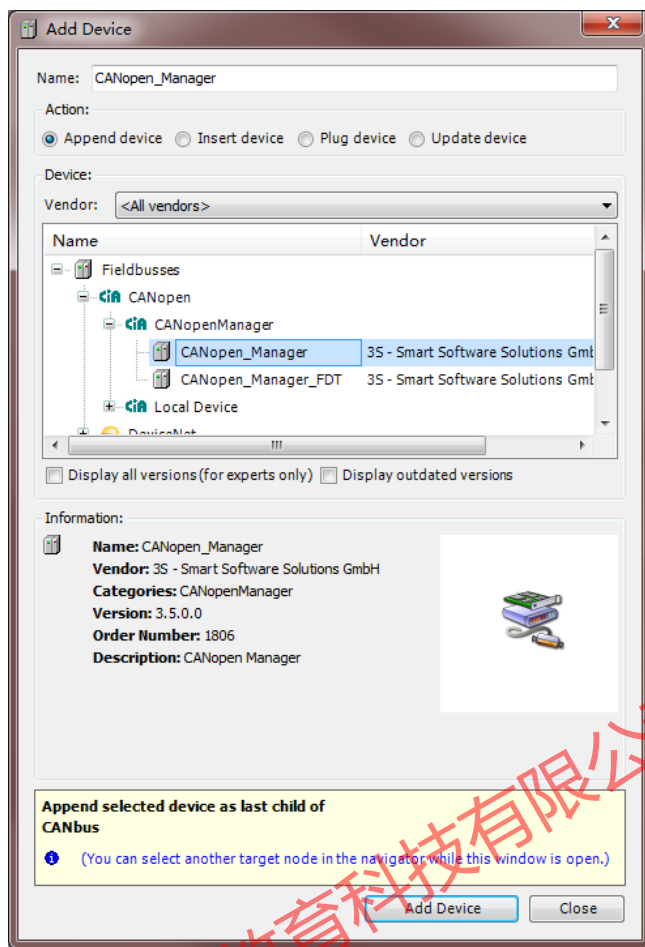


图 3.47 添加 CANopen_Manager

打开 CANopen Manager，由于远程 IO 设备栈号为 127，CANopen Manager 的栈号需要小于 127。请将 Node ID 设为 1，并将 Enable Heartbeat Producing 选项勾选去掉。同时在 CANopen I/O Mapping 选项卡中，将 Bus cycle options 的任务设置为 CANopen 相关的任务（在本文中 MainTask 作为 CANopen 相关的任务）。

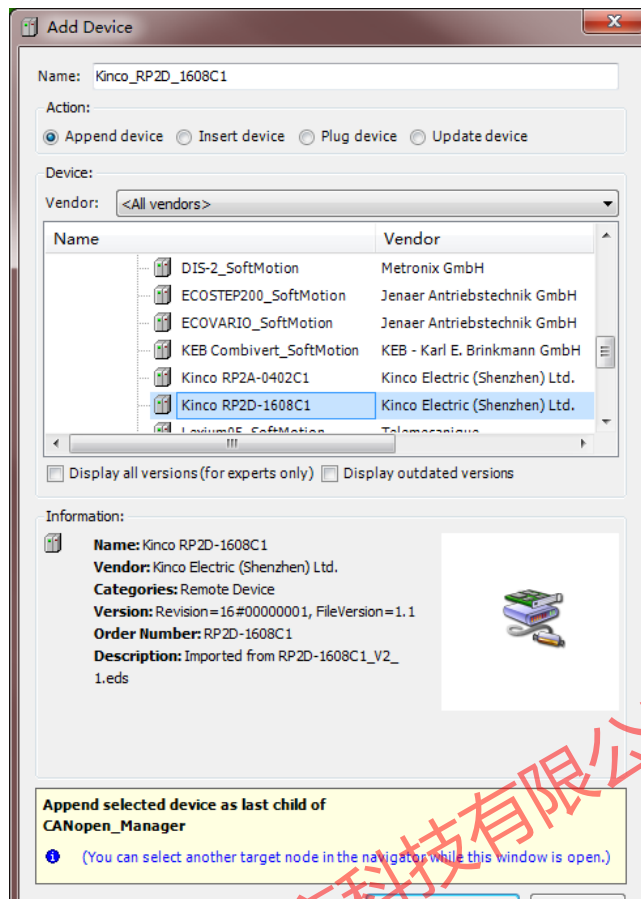


图 3.48 添加远程 IO

打开远程 IO 设备，请将 NodeID 设为 127，同时在 CANopen I/O Mapping 选项卡中，将 Always update variables 勾选。

到此时 CANopen 远程 IO 已经组态完毕，程序的 Device 窗口如下图所示。

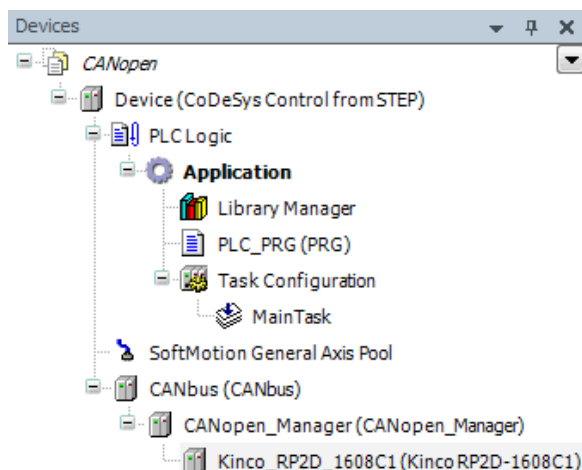


图 3.49 CANopen 程序

3.4.2 修改 PLC_PRG 程序

本例是在 IO 设备上进行的操作，所以将使用与 EtherCAT IO 设备相同的例程

来演示。

请参照下述代码修改 PLC_PRG 程序。

```
PROGRAM PLC_PRG
```

```
VAR
```

```
    iCnt: INT;
```

```
    timer : TON;
```

```
END_VAR
```

```
VAR_OUTPUT
```

```
    aOut : ARRAY [0..7] OF INT;
```

```
END_VAR
```

```
IF iCnt < 8 THEN
```

```
    {每个循环先清零}
```

```
    aOut[0] := 0;
```

```
    aOut[1] := 0;
```

```
    aOut[2] := 0;
```

```
    aOut[3] := 0;
```

```
    aOut[4] := 0;
```

```
    aOut[5] := 0;
```

```
    aOut[6] := 0;
```

```
    aOut[7] := 0;
```

```
    {控制小于等于iCnt, 且差值为2的等差数列作为亮灯的Index}
```

```
    aOut[iCnt] := 1;
```

```
    IF iCnt > 1 THEN
```

```
        aOut[iCnt - 2] := 1;
```

```
    END_IF
```

```
    IF iCnt > 3 THEN
```

```
        aOut[iCnt - 4] := 1;
```

```
    END_IF
```

```
    IF iCnt > 5 THEN
```

```
        aOut[iCnt - 6] := 1;
```

```
    END_IF
```

```
    {每次iCnt要等0.2秒才累加1}
```

```
    timer(IN := TRUE, PT := T#0.2S);
```

```
    IF timer.Q THEN
```

```
        iCnt := iCnt + 1;
```

```
        timer(IN := FALSE);
```

```
    END_IF
```

```
ELSIF iCnt < 15 AND iCnt >= 8 THEN
```

```
    {每次循环先清零}
```

```
    aOut[0] := 0;
```

```
    aOut[1] := 0;
```

```
    aOut[2] := 0;
```

```
    aOut[3] := 0;
```

```
    aOut[4] := 0;
```

```
    aOut[5] := 0;
```

```
    aOut[6] := 0;
```

```
    aOut[7] := 0;
```

```
    {控制大于8的小于16的Index跑完尾数灯}
```

```
    IF iCnt < 10 THEN
```

```
        aOut[iCnt - 6] := 1;
        aOut[iCnt - 4] := 1;
        aOut[iCnt - 2] := 1;
    END_IF
    IF iCnt >= 10 AND iCnt < 12 THEN
        aOut[iCnt - 6] := 1;
        aOut[iCnt - 4] := 1;
    END_IF
    IF iCnt >= 12 AND iCnt < 14 THEN
        aOut[iCnt - 6] := 1;
    END_IF
    timer(IN := TRUE, PT := T#0.2S);
    IF timer.Q THEN
        iCnt := iCnt + 1;
        timer(IN := FALSE);
    END_IF
ELSE
    aOut[0] := 0;
    aOut[1] := 0;
    aOut[2] := 0;
    aOut[3] := 0;
    aOut[4] := 0;
    aOut[5] := 0;
    aOut[6] := 0;
    aOut[7] := 0;
    iCnt := 0;
END_IF
{此处将输出映射给组态的设备IO的地址}
%X1.0 := aOut[0].0;
%X1.1 := aOut[1].0;
%X1.2 := aOut[2].0;
%X1.3 := aOut[3].0;
%X1.4 := aOut[4].0;
%X1.5 := aOut[5].0;
%X1.6 := aOut[6].0;
%X1.7 := aOut[7].0;
```

3.4.3 编译并运行

与前述的编译运行的步骤方法一致。

长沙工控帮教育科技有限公司

第四章 机器人程序实例

按照详细步骤分解，一步一步的向用户说明，完整机器人程序中各个模块的作用。

注：本实例只介绍了 CoDeSys 环境下，即软 PLC 环境下的机器人 PLC 程序。用户若欲了解机器人软件的使用，请参照《机器人操作软件使用说明书》（STEP 控制器版）。

4.1 本例适用机器人

本实例的机器人信息获得自 STEP SR18L8 型机器人，且配合 3 个附加轴进行运动，机械参数及电气参数都附带在出厂说明书中，请自行查阅。

此外，关于机器人配置软件 RobotConfig 的使用，请参照《配置界面用户说明书》。

4.2 程序详解

4.2.1 伺服组态过程

拥有 6 个 EtherCAT 伺服，并且有数字量 IO 等硬件。

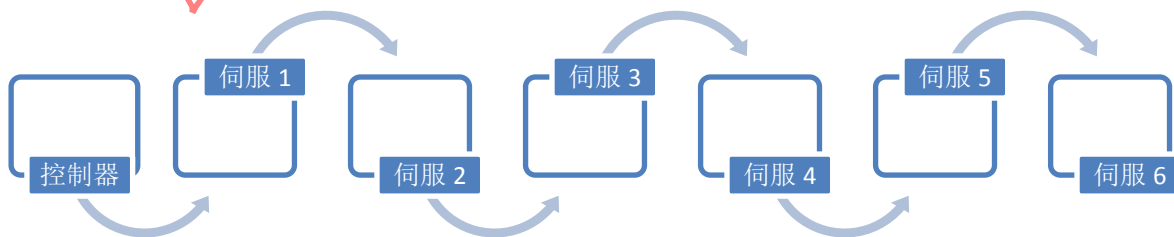


图 4.1 设备拓扑结构

根据第三章中介绍的组态方法，将 6 个 EtherCAT 伺服按照设备拓扑结构组态进入 StepRobot_SR18L8.project 中。并且根据机械参数与电气参数对各个轴进行配置。

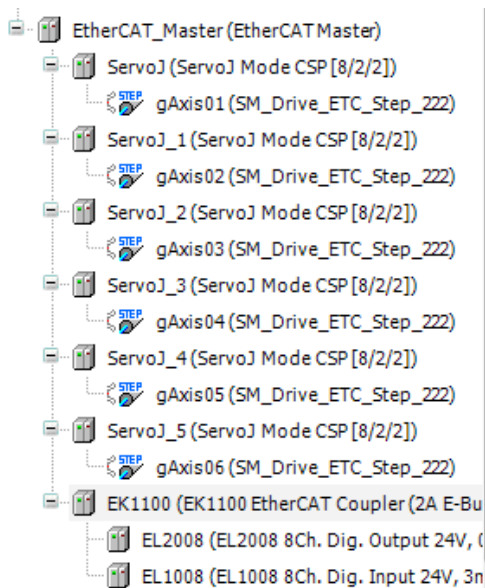


图 4.2 EtherCAT 设备组态完成

其中，Scaling 选项卡，请参照以下名字解释进行填充：

- Increments: 编码器线数
- Motor turns: 机械减速比分子
- Gear output turns: 机械减速比分母
- Units in application: 用户定义的单位（此处为°）

注：Bus cycle options 选用 use parent bus cycle setting。（Device→PLC Setting 中 Bus cycle options 选为 EtherCAT_Master）。

4.2.2 IO 设备组态

EtherCAT IO 设备的端口映射，是根据控制柜说明书中要求的输入、输出信号来匹配的。

截止本说明书发布之时，使用到的 IO 信号有 5 出（■）3 入（●），分别是：

- 主电源指示灯
- 使能指示灯
- 警告指示灯×2
- 安全扭矩（STO）信号
- 自动硬件使能
- 手动复位
- 钥匙开关

如下图所示，对 IO 端口进行映射

Slave	Process Data	EtherCAT I/O Mapping	Status	Information		
Channels						
Variable	Mapping	Channel	Address	Type	Unit	Description
 Application.MainPowerBit		Output	%QX70.0	BIT		Output
 Application.RobotEnableFlagBit		Output	%QX70.1	BIT		Output
 Application.AlarmFlagBit		Output	%QX70.2	BIT		Output
 Application.AlarmFlagBit		Output	%QX70.3	BIT		Output
 Application.STOControlBit		Output	%QX70.4	BIT		Output
		Output	%QX70.5	BIT		Output
		Output	%QX70.6	BIT		Output
		Output	%QX70.7	BIT		Output

图 4.3 输出 IO 映射

Slave	Process Data	EtherCAT I/O Mapping	Status	Information		
Channels						
Variable	Mapping	Channel	Address	Type	Unit	Description
 Application.AutoHWEEnableBit		Input	%IX48.0	BIT		Input
 Application.ManualResetBit		Input	%IX48.1	BIT		Input
 Application.KeySwitchLeftContro...		Input	%IX48.2	BIT		Input
		Input	%IX48.3	BIT		Input
		Input	%IX48.4	BIT		Input
		Input	%IX48.5	BIT		Input
		Input	%IX48.6	BIT		Input
		Input	%IX48.7	BIT		Input

图 4.4 输入 IO 映射

4.2.3 附加轴

本实例并无实际的附加轴，故而三个附加轴用虚拟轴来作替代。

在 SoftMotion General Axis Pool 中，添加三个虚拟轴。

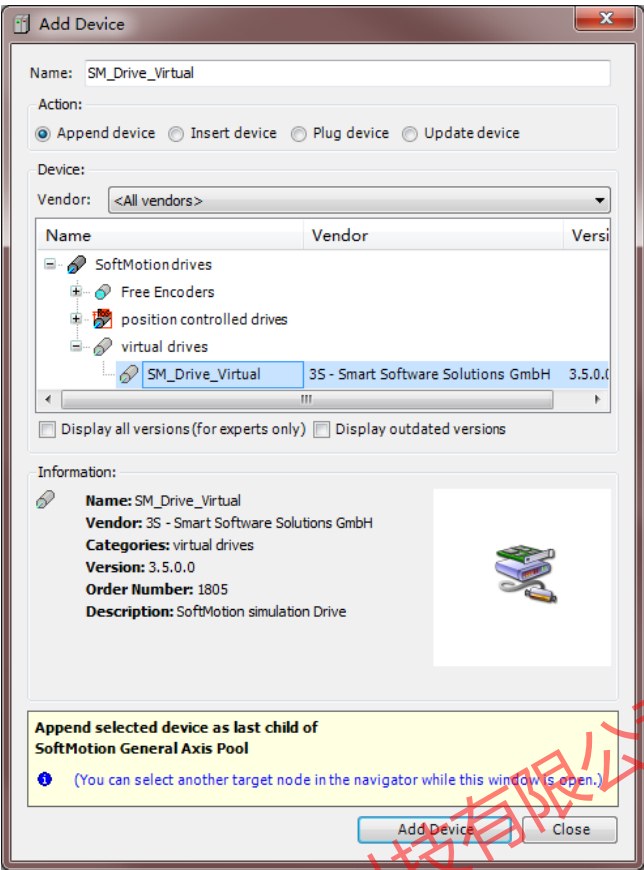


图 4.5 添加虚拟轴

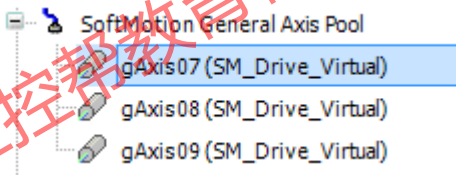


图 4.6 虚拟轴

4.2.4 全局变量

主要用来声明 IO 映射所需的变量。

```
VAR_GLOBAL
(*OutPut*)
MainPowerBit : BOOL := FALSE;
RobotEnableFlagBit : BOOL;
AlarmFlagBit : BOOL;
STOControlBit : BOOL := TRUE;

(*Input*)
AutoHWEEnableBit : BOOL;
ManualResetBit : BOOL;
KeySwitchLeftControlBit : BOOL;
END_VAR
```

4.2.5 IO 程序

主要是用来处理外部输入信号，并将处理结果进行输出。

```

STO_IO.RobotEnableFlagBit := FALSE;
IF gAxis06.bRegulatorRealState AND gAxis06.bDriveStartRealState THEN
  IF gAxis05.bRegulatorRealState AND gAxis05.bDriveStartRealState THEN
    IF gAxis04.bRegulatorRealState AND gAxis04.bDriveStartRealState THEN
      IF gAxis03.bRegulatorRealState AND gAxis03.bDriveStartRealState THEN
        IF gAxis02.bRegulatorRealState AND gAxis02.bDriveStartRealState THEN
          IF gAxis01.bRegulatorRealState AND gAxis01.bDriveStartRealState THEN
            STO_IO.RobotEnableFlagBit := TRUE;
          END_IF
        END_IF
      END_IF
    END_IF
  END_IF
END_IF

STO_IO.MainPowerBit := TRUE;
STO_IO.AlarmFlagBit := FALSE;
IF gAxis01.bError OR gAxis01.nAxisState = errorstop THEN
  STO_IO.MainPowerBit := FALSE;
  STO_IO.AlarmFlagBit := TRUE;
END_IF
IF gAxis02.bError OR gAxis02.nAxisState = errorstop THEN
  STO_IO.MainPowerBit := FALSE;
  STO_IO.AlarmFlagBit := TRUE;
END_IF
IF gAxis03.bError OR gAxis03.nAxisState = errorstop THEN
  STO_IO.MainPowerBit := FALSE;
  STO_IO.AlarmFlagBit := TRUE;
END_IF
IF gAxis04.bError OR gAxis04.nAxisState = errorstop THEN
  STO_IO.MainPowerBit := FALSE;
  STO_IO.AlarmFlagBit := TRUE;
END_IF
IF gAxis05.bError OR gAxis05.nAxisState = errorstop THEN
  STO_IO.MainPowerBit := FALSE;
  STO_IO.AlarmFlagBit := TRUE;
END_IF
IF gAxis05.bError OR gAxis05.nAxisState = errorstop THEN
  STO_IO.MainPowerBit := FALSE;
  STO_IO.AlarmFlagBit := TRUE;
END_IF

```

4.2.6 通讯程序

通讯程序与机器人程序相互配合，完成了机器人逻辑、运动控制与示教器程序之间的互动。

变量声明：

```

PROGRAM CommTask
VAR
    step : DINT := 0;
    ConnectRet : DINT := 0;
END_VAR

```

程序代码:

```

CASE step OF
    0:
        step := 1;
    1:
        ConnectRet := ConnectToHMI();
        IF ConnectRet <> -1 THEN
            IF ParamInit <> 3 THEN
                ParamInit := 2;
                step := 2;
            ELSE
                step := 3;
            END_IF
        END_IF
    2:
        step := 3;
    3:
        IF ParamInit = 4 THEN
            ConnectRet := CommunicateWithHMI();
            IF ConnectRet = -1 THEN
                step := 1;
            END_IF
        END_IF
END_CASE

```

4.2.7 机器人主程序

机器人主程序完成了机器人的创建、轴的添加、参数的初始化以及算法和运动控制的循环，是机器人程序的最主要程序。

变量声明:

```

PROGRAM PLC_PRG
VAR
    InitDone : USINT := 0;
    MC_Step_GroupCreate_0: MC_Step_GroupCreate;
    robot: POINTER TO AXES_GROUP_REF;
    MC_Step_AddAxisToGroup_0: MC_Step_AddAxisToGroup;
    MC_Step_AddAxisToGroup_1: MC_Step_AddAxisToGroup;
    MC_Step_AddAxisToGroup_2: MC_Step_AddAxisToGroup;
    MC_Step_AddAxisToGroup_3: MC_Step_AddAxisToGroup;
    MC_Step_AddAxisToGroup_4: MC_Step_AddAxisToGroup;
    MC_Step_AddAxisToGroup_5: MC_Step_AddAxisToGroup;
    MC_Step_AddAxisToGroup_6: MC_Step_AddAxisToGroup;
    MC_Step_AddAxisToGroup_7: MC_Step_AddAxisToGroup;
    MC_Step_AddAxisToGroup_8: MC_Step_AddAxisToGroup;
    MC_Step_GroupInitParam_0: MC_Step_GroupInitParam;

```

```

MC_Step_UpdateAxes_0: MC_Step_UpdateAxes;
MC_Step_RobotPower_0: MC_Step_RobotPower;
MC_Step_RobotReset_0: MC_Step_RobotReset;
MC_Step_GroupRun_0: MC_Step_GroupRun;
MC_Step_MoveAxesPosition_0: MC_Step_MoveAxesPosition;
MC_Step_GroupReset_0 : MC_Step_GroupReset;
iPowerCnt : INT := 0;
tTimer0 : TON;
END_VAR

```

程序代码:

```

CASE InitDone OF
  0:
    MC_Step_GroupCreate_0(GroupName := 'SR18L8',Execute := TRUE);
    IF (MC_Step_GroupCreate_0.Done) THEN
      robot := MC_Step_GroupCreate_0.AxesGroupAddr;
    END_IF
    robot^.NBaseJoints := 6;
    robot^.NAuxJoints := 0;
    robot^.NJoints := robot^.NBaseJoints + robot^.NAuxJoints;

    MC_Step_AddAxis();

    IF robot^.Joints[5].m_Axis <> 0 THEN
      tTimer0(IN := TRUE, PT := T#13S);
      IF tTimer0.Q THEN
        InitDone := 1;
      END_IF
    END_IF

  1:
    InitDone := 2;

  2:
    InitDone := 3;

  3:
    IF (ParamInit = 2) THEN
      MC_Step_GroupInitParam_0(Name := robot^.name, Enable := TRUE);
      IF(MC_Step_GroupInitParam_0.Done) THEN
        ParamInit := 3;
        InitDone := 4;
      END_IF
    END_IF

  4:
    InitDone := 5;

  5:
    IF iPowerCnt < 20 THEN
      iPowerCnt := iPowerCnt + 1;
    ELSE
      InitDone := 6;
      iPowerCnt := 0;
    END_IF

```

```
6:

    MC_Step_UpdateAxes_0(AxesGroup := robot, Execute := TRUE);

    StepInputs();

    UserPreAction();

    MC_Step_GroupRun_0(AxesGroup := robot, Enable := TRUE);

    UserPostAction();

    StepOutputs();

    MC_Step_Action();

    MC_Step_MoveAxesPosition_0(AxesGroup := robot, Execute := TRUE);

    ParamInit := 4;

100:

END_CASE
```

其中，调用了一些其他程序或者 Action。

长沙工控帮教育科技有限公司

StepInputs 程序：

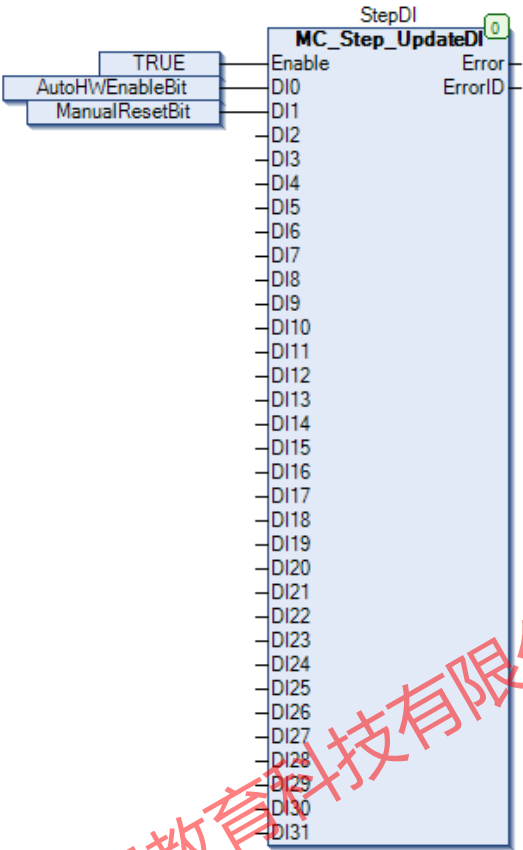


图 4.7 StepInputs

StepOutputs 程序:



图- 1 StepOutputs

MC_Step_Action:

```
MC_Step_RobotReset_0(AxesGroup := robot, Enable := 1);
MC_Step_RobotPower_0(AxesGroup := robot, Enable := TRUE);
```

MC_Step_AddAxis:

```
MC_Step_AddAxisToGroup_0(AxesGroup := robot, Axis := ADR(gAxis01), IdentInGroup := 0, Execute
:= TRUE);
IF MC_Step_AddAxisToGroup_0.Error THEN
    MC_Step_AddAxisToGroup_0.Execute := FALSE;
    MC_Step_AddAxisToGroup_0();
    InitDone := 100;
END_IF
IF MC_Step_AddAxisToGroup_0.Done THEN
    MC_Step_AddAxisToGroup_0.Execute := FALSE;
    MC_Step_AddAxisToGroup_0();
END_IF

MC_Step_AddAxisToGroup_1(AxesGroup := robot, Axis := ADR(gAxis02), IdentInGroup := 1, Execute
:= TRUE);
```

```

IF MC_Step_AddAxisToGroup_1.Error THEN
    MC_Step_AddAxisToGroup_1.Execute := FALSE;
    MC_Step_AddAxisToGroup_1();
    InitDone := 100;
END_IF
IF MC_Step_AddAxisToGroup_1.Done THEN
    MC_Step_AddAxisToGroup_1.Execute := FALSE;
    MC_Step_AddAxisToGroup_1();
END_IF

MC_Step_AddAxisToGroup_2(AxesGroup := robot, Axis := ADR(gAxis03), IdentInGroup := 2, Execute
:= TRUE);
IF MC_Step_AddAxisToGroup_2.Error THEN
    MC_Step_AddAxisToGroup_2.Execute := FALSE;
    MC_Step_AddAxisToGroup_2();
    InitDone := 100;
END_IF
IF MC_Step_AddAxisToGroup_2.Done THEN
    MC_Step_AddAxisToGroup_2.Execute := FALSE;
    MC_Step_AddAxisToGroup_2();
END_IF

MC_Step_AddAxisToGroup_3(AxesGroup := robot, Axis := ADR(gAxis04), IdentInGroup := 3, Execute
:= TRUE);
IF MC_Step_AddAxisToGroup_3.Error THEN
    MC_Step_AddAxisToGroup_3.Execute := FALSE;
    MC_Step_AddAxisToGroup_3();
    InitDone := 100;
END_IF
IF MC_Step_AddAxisToGroup_3.Done THEN
    MC_Step_AddAxisToGroup_3.Execute := FALSE;
    MC_Step_AddAxisToGroup_3();
END_IF

MC_Step_AddAxisToGroup_4(AxesGroup := robot, Axis := ADR(gAxis05), IdentInGroup := 4, Execute
:= TRUE);
IF MC_Step_AddAxisToGroup_4.Error THEN
    MC_Step_AddAxisToGroup_4.Execute := FALSE;
    MC_Step_AddAxisToGroup_4();
    InitDone := 100;
END_IF
IF MC_Step_AddAxisToGroup_4.Done THEN
    MC_Step_AddAxisToGroup_4.Execute := FALSE;
    MC_Step_AddAxisToGroup_4();
END_IF

MC_Step_AddAxisToGroup_5(AxesGroup := robot, Axis := ADR(gAxis06), IdentInGroup := 5, Execute
:= TRUE);
IF MC_Step_AddAxisToGroup_5.Error THEN
    MC_Step_AddAxisToGroup_5.Execute := FALSE;
    MC_Step_AddAxisToGroup_5();
    InitDone := 100;
END_IF
IF MC_Step_AddAxisToGroup_5.Done THEN

```

```

    MC_Step_AddAxisToGroup_5.Execute := FALSE;
    MC_Step_AddAxisToGroup_5();
END_IF

MC_Step_AddAxisToGroup_6(AxesGroup := robot, Axis := ADR(gAxis07), IdentInGroup := 6, Execute
:= TRUE);
IF MC_Step_AddAxisToGroup_6.Error THEN
    MC_Step_AddAxisToGroup_6.Execute := FALSE;
    MC_Step_AddAxisToGroup_6();
    InitDone := 100;
END_IF
IF MC_Step_AddAxisToGroup_6.Done THEN
    MC_Step_AddAxisToGroup_6.Execute := FALSE;
    MC_Step_AddAxisToGroup_6();
END_IF

MC_Step_AddAxisToGroup_7(AxesGroup := robot, Axis := ADR(gAxis08), IdentInGroup := 7, Execute
:= TRUE);
IF MC_Step_AddAxisToGroup_7.Error THEN
    MC_Step_AddAxisToGroup_7.Execute := FALSE;
    MC_Step_AddAxisToGroup_7();
    InitDone := 100;
END_IF
IF MC_Step_AddAxisToGroup_7.Done THEN
    MC_Step_AddAxisToGroup_7.Execute := FALSE;
    MC_Step_AddAxisToGroup_7();
END_IF

MC_Step_AddAxisToGroup_8(AxesGroup := robot, Axis := ADR(gAxis09), IdentInGroup := 8, Execute
:= TRUE);
IF MC_Step_AddAxisToGroup_8.Error THEN
    MC_Step_AddAxisToGroup_8.Execute := FALSE;
    MC_Step_AddAxisToGroup_8();
    InitDone := 100;
END_IF
IF MC_Step_AddAxisToGroup_8.Done THEN
    MC_Step_AddAxisToGroup_8.Execute := FALSE;
    MC_Step_AddAxisToGroup_8();
END_IF

MC_Step_GroupReset_0(AxesGroup := robot, Execute := TRUE);

```

UserPostAction 与 UserPreAction 是开放给用户的接口，用户可以在机器人状态机接口之前或之后进行一些用户程序的编写。

第五章 其他功能汇总

除了上述的一些功能外，机器人控制器还提供了串口、ftp 服务器等功能。

5.1 串口功能

本节将通过一个例程来演示 STEP 控制器的串口功能。

串口通信(Serial Communication)，是指外设和计算机间，通过数据信号线、地线、控制线等，按位进行传输数据的一种通讯方式。这种通信方式使用的数据线少，在远距离通信中可以节约通信成本，但其传输速度比并行传输低。

STEP 控制器提供了 RS232/RS485 两种形式的串行接口，两者只能选用其一，通过硬件接口(P103)来联通外部设备（详见《STEP 机器人控制器硬件使用说明书》）。符合 RS232/RS485 串行物理接口标准的设备从理论上都可以控制器进行通讯。

由于 RS232/485 都是物理接口标准，在操作系统中，只提供到链路层的接口，如果要实现数据的双向访问，就必须自己编写应用层协议或者按照应用层协议标准（譬如 MODBUS 协议）来编写通讯程序。从 STEP 控制器来讲，并没有提供类似 MODBUS 这样的应用层协议栈，那就只能自己编写应用层协议，再编写通讯程序。

回归到具体的通讯，两台带有串口功能的设备要相互通讯，要事先做好某些约定，之后双方按照约定的方式来发送/接收数据。

RS232 最常见的设置有：

- 端口号，STEP 控制器 RS232 串行接口的端口号为 1。
- 波特率 (Baud Rate)：是指从一设备发到另一设备的波特率，即每秒钟多少比特 bits per second (bit/s)。典型的波特率是 300, 1200, 2400, 9600, 19200, 115200 等 bit/s。一般通信两端设备都要设为相同的波特率。
- 奇偶校验(Parity Check)：是用来验证数据的正确性。奇偶校验一般不使用，如果使用，那么既可以做奇校验 (Odd Parity) 也可以做偶校验 (Even Parity)。
- 停止位(Stop Bit)：是在每个字节传输之后发送的，它用来帮助接受信号方硬件重新同步

RS485 配置有些不同：

- 端口号：STEP 控制器的 RS485 串行接口的端口号为 2。

- RTS 控制：请置位为 TRUE。

下面就以 STEP 控制器与一台 RS232 设备进行通讯作为示例来描述如何使用 CoDeSys 的串口库来编写应用程序。

5.1.1 添加串口库

在工程的 Library Manager 中添加本例将用到的 SysCom.library。

Name	Namespace	Effective version
Standard, 3.5.0.40 (System)	Standard	3.5.0.40
IoStandard, 3.5.0.30 (System)	IoStandard	3.5.0.30
SM3_Basic = SM3_Basic, 3.5.0.0 (3S - Smart Software Solutions GmbH)	SM3_Basic	3.5.0.0
SM3_CNC = SM3_CNC, 3.5.0.30 (3S - Smart Software Solutions GmbH)	SM3_CNC	3.5.0.30
SysCom, 3.3.2.50 (System)	SysCom	3.3.2.50
SysTypes, 3.1.2.0 (System)	SysTypes	3.1.2.0
CmpErrors, 3.3.1.40 (System)	CmpErrors	3.3.1.40

图 5.1 添加 Com 库

5.1.2 添加变量

```
VAR_GLOBAL
{串口参数设置}
pComSettings : COM_Settings;
pComSettingsEx : COM_SettingsEx;
ulTimeout : COM_Timeout := 0;
{串口句柄以及函数返回值}
hCom : RTS_IEC_HANDLE;
pResultOpen : RTS_IEC_RESULT;
pResultRead : RTS_IEC_RESULT;
pResultWrite : RTS_IEC_RESULT;
{读写缓冲区}
pbyReadBuffer : ARRAY[1..6] OF BYTE;
pbyWriteBuffer : ARRAY[1..100] OF BYTE := [20(16#FF),20(0),20(1),20(2),20(3)];
ulSizeRead : UDINT := 6;
ulSizeWrite : UDINT := 100;
udiReadNum : UDINT;
udiWriteNum : UDINT;
END_VAR
```

5.1.3 主程序代码

```
PROGRAM PLC_PRG
VAR
    iState : INT := 0;
    timer : TON;
    timerQ : BOOL;
    iCnt: INT;
    bTest: BOOL;
    aaa: UDINT;
END_VAR

CASE iState OF
    0: {串口参数设置}
```

```

pComSettings.byParity := SYS_NOPARITY;
pComSettings.byStopBits := SYS_ONESTOPBIT;
pComSettings.sPort := SYS_COMPORT1;
pComSettings.ulBaudrate := SYS_BR_9600;
pComSettings.ulBufferSize := 100;
pComSettings.ulTimeout := 0;
pComSettingsEx.bBinary := 1;
pComSettingsEx.byByteSize := 8;

{打开串口}
hCom := SysComOpen2(pSettings := ADR(pComSettings), pSettingsEx := ADR(pComSettingsEx),
pResult := ADR(pResultOpen));
IF pResultOpen = Errors.ERR_OK AND hCom > 0 THEN
    iState := 5;
ELSE
    iState := 10;
END_IF
5: {定时器开启, 循环读写}
    timer(IN := NOT timerQ, PT := T#1S, Q=> timerQ, ET=>);

    IF iCnt = 20 THEN
        udiReadNum := SysComRead(hCom := hCom, pbyBuffer := ADR(pbyReadBuffer), ulSize :=
ulSizeRead, ulTimeout := ulTimeout, pResult := ADR(pResultRead));
        ELSE
            udiReadNum := 0;
        END_IF

        IF udiReadNum > 0 THEN
            bTest := TRUE;
        ELSIF udiReadNum = 0 THEN
            bTest := FALSE;
        END_IF

        IF timer.Q THEN
            udiWriteNum := SysComWrite(hCom := hCom, pbyBuffer := ADR(pbyWriteBuffer), ulSize :=
ulSizeWrite, ulTimeout := ulTimeout, pResult := ADR(pResultWrite));
            //udiWriteNum := 0;
            IF udiWriteNum > 0 THEN
                iCnt := 1;
            END_IF
        ELSE
            udiWriteNum := 0;
        END_IF

        iCnt := iCnt + 1;

        {定义一个特殊的字符串就可以来控制串口程序的流程}
        IF udiReadNum > 0 AND pResultRead = Errors.ERR_OK THEN
            iState := 5;
        END_IF
10: {关闭串口}
    pResultClose := SysComClose(hCom);

```

```
IF pResultClose = Errors.ERR_OK THEN
    iState := 100;
END_IF
ELSE
    iCnt := iCnt + 1;
END_CASE
```

5.1.4 PC 端串口工具

使用 Linux 平台下的 xgcom 串口调试工具来测试上述例程。

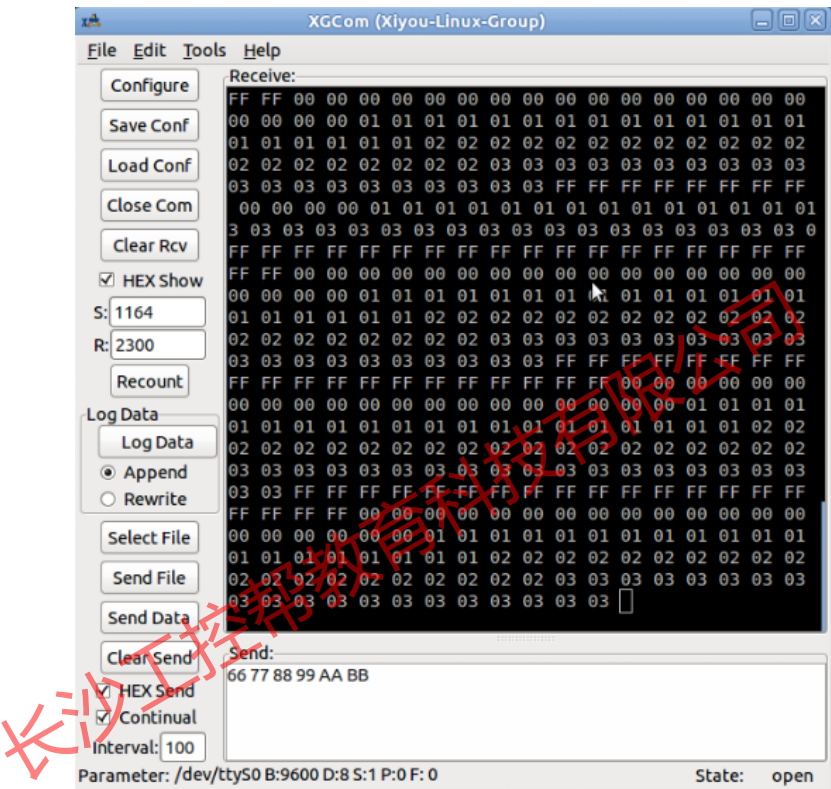


图 5.2 XGCom 测试

Expression	Type	Value	Prepared value	Comment
✱ pComSettings	COM_Settings			
✱ pComSettingsEx	COM_SettingsEx			
ulTimeout	COM_TIMEOUT	SYS_NOWAIT		
✱ hCom	POINTER TO BYTE	16#00000015		
pResultOpen	UDINT	0		
pResultRead	UDINT	0		
pResultWrite	UDINT	0		
pResultClose	UDINT	0		
▣ pbyReadBuffer	ARRAY [1..6] OF BYTE			
pbyReadBuffer[1]	BYTE	102		
pbyReadBuffer[2]	BYTE	119		
pbyReadBuffer[3]	BYTE	136		
pbyReadBuffer[4]	BYTE	153		
pbyReadBuffer[5]	BYTE	170		
pbyReadBuffer[6]	BYTE	187		
✱ pbyWriteBuffer	ARRAY [1..100] OF..			
ulSizeRead	UDINT	6		
ulSizeWrite	UDINT	100		
udiReadNum	UDINT	0		
udiWriteNum	UDINT	0		

图 5.3 CoDeSys 端变量查看

可以看到 XGCom 端发送的字符串 66 77 88 99 AA BB 在控制器的 ReadBuffer 中以十进制的形式出现。

5.2 套接字接口编程

以太网是一种计算机局域网组网技术。与上节的串口接口类似，以太网也只是一种硬件标准。在这种硬件标准上，操作系统又构建了基于 ISO/OSI 网络通讯模型的以太网协议栈。众所周知，OSI 模型中包含了 7 层，分别为物理层、数据链路层、网络层、传输层、会话层、表示层、应用层。

常用的网络通讯协议 TCP/UDP 位于七层中的传输层，所以与串口通讯一样，应用层的协议，还是由用户自行编写。用户编写的通讯程序，所使用的接口就是本节所讲的套接字接口。

所以套接字接口，属于应用层和传输层之间的一个抽象层，属于操作系统提供给用户的 API。

接下来我们以套接字接口来实现以 TCP 面向连接协议的通讯例程。

5.2.1 STEP 控制器作为服务器的例程：

```
PROGRAM PLC_PRG
VAR
    sa : NBS.SOCKADDRESS;    //套接字地址结构体
    hResult : NBS.RTS_IEC_RESULT;    //套接字返回句柄
    hSocket : NBS.RTS_IEC_HANDLE;    //套接字句柄
```

```

hAcceptSock : NBS.RTS_IEC_HANDLE; //握手返回套接字句柄
hClose : NBS.RTS_IEC_RESULT; //套接字关闭句柄
hAcceptRst : NBS.RTS_IEC_RESULT; //握手返回句柄
lpArray : ARRAY[1..6] OF BYTE; //数据缓冲区, 大小可自行调节
sdArray : ARRAY[1..6] OF BYTE;
lpBuffer : POINTER TO BYTE; //接收缓冲区指针
sdBuffer : POINTER TO BYTE; //发送缓冲区指针

dSize : DINT; //接收数据的长度
bTrue : BOOL := TRUE; //用于流程控制
dRecvSize : DINT; //接收数据的长度
dSendSize : DINT; //发送数据的长度

step : INT; //用于流程控制
END_VAR

```

{根据step的值进入不同的分支}

CASE step OF

{第一步, 建立一个Socket句柄}

0:

//因使用TCP协议, 需要使用AF_INET协议族, 以及STREAM, 并选择TCP协议

hSocket := NBS.SysSockCreate(NBS.SOCKET_AF_INET, NBS.SOCKET_STREAM,
NBS.SOCKET_IPPROTO_TCP, NULL);

IF hSocket <> NBS.RTS_INVALID_HANDLE THEN

step := 1;

ELSE

step := 100;

END_IF

{第二步, 将期望的客户端IP, 以及连接用的端口填入sa结构体中, 并将该结构体绑定到第0步所建立的句柄}

1:

SysMem.SysMemSet(ADR(sa),0,SIZEOF(sa));

sa.sin_family := NBS.SOCKET_AF_INET;

sa.sin_addr.ulAddr := NBS.SOCKET_INADDR_ANY; //所有的局域网内的IP都可以连接到本服务器

sa.sin_port := NBS.SysSockHtons(7777); //端口号设置为7777, 一般最好大于10000

IF NBS.SysSockBind(hSocket, ADR(sa),SIZEOF(sa)) <> NBS.RTS_INVALID_HANDLE THEN

step := 2;

ELSE

step := 100;

END_IF

{第三步, 通过Socket句柄来监听是否由客户端发来握手信号}

2:

IF NBS.SysSockListen(hSocket, 1024) <> NBS.RTS_INVALID_HANDLE THEN

dSize := SIZEOF(sa);

step := 3;

ELSE

step := 100;

END_IF

{第四步，监听到握手信号后，向客户端返回握手信号，并返回一个Socket句柄，用来与客户端通讯}

3:

hAcceptSock := NBS.SysSockAccept(hSocket, (ADR(sa)), ADR(dSize), ADR(hAcceptRst));

IF hAcceptSock <> NBS.RTS_INVALID_HANDLE THEN

step := 4;

END_IF

{通讯句柄正常，进入正常的发送，接收循环}

4:

lpBuffer := ADR(lpArray[1]);

dRecvSize := NBS.SysSockRecv(hAcceptSock, lpBuffer, 6, 0, ADR(hResult));

IF (dRecvSize < 0) OR (hResult = NBS.RTS_INVALID_HANDLE) THEN

dRecvSize := 0;

END_IF

IF dRecvSize > 0 THEN

sdBuffer := ADR(lpArray[1]);

dSendSize := NBS.SysSockSend(hAcceptSock, sdBuffer, 6, 0, ADR(hResult));

END_IF

5:

hClose := NBS.SysSockClose(hSocket);

hClose := NBS.SysSockClose(hAcceptSock);

100:

END_CASE

5.2.2 STEP 控制器作为客户端的例程

PROGRAM PLC_PRG

VAR

sa : NBS.SOCKADDRESS; //套接字地址结构体

hResult : NBS.RTS_IEC_RESULT; //套接字返回句柄

hSocket : NBS.RTS_IEC_HANDLE; //套接字句柄

hAcceptSock : NBS.RTS_IEC_HANDLE; //握手返回套接字句柄

hClose : NBS.RTS_IEC_RESULT; //套接字关闭句柄

hAcceptRst : NBS.RTS_IEC_RESULT; //握手返回句柄

lpArray : ARRAY[1..6] OF BYTE; //数据缓冲区，大小可自行调节

sdArray : ARRAY[1..6] OF BYTE;

lpBuffer : POINTER TO BYTE; //接收缓冲区指针

sdBuffer : POINTER TO BYTE; //发送缓冲区指针

dSize : DINT; //接收数据的长度

bTrue : BOOL := TRUE; //用于流程控制

dRecvSize : DINT; //接收数据的长度

dSendSize : DINT; //发送数据的长度

ipAddr : NBS.INADDR; //转换IP字符串到INADDR类型

step : INT; //用于流程控制

END_VAR

{根据step的值进入不同的分支}

```

CASE step OF
{第一步, 建立一个Socket句柄}
  0:
    //因使用TCP协议, 需要使用AF_INET协议簇, 以及STREAM, 并选择TCP协议
    hSocket := NBS.SysSockCreate(NBS.SOCKET_AF_INET, NBS.SOCKET_STREAM,
NBS.SOCKET_IPPROTO_TCP, NULL);
    IF hSocket <> NBS.RTS_INVALID_HANDLE THEN
      step := 1;
    ELSE
      step := 100;
    END_IF
{第二步, 向服务器发送握手信号}
  1:
    SysMem.SysMemSet(ADR(sa),0,SIZEOF(sa));
    sa.sin_family := NBS.SOCKET_AF_INET;
    NBS.SysSockInetAddr('192.168.0.111', ADR(ipAddr)); //单引号内为服务器IP
    sa.sin_addr := ipAddr;
    sa.sin_port := NBS.SysSockHtons(7777);
    IF NBS.SysSockConnect(hSocket, ADR(sa), SIZEOF(sa)) <> NBS.RTS_INVALID_HANDLE THEN
      step := 2;
    ELSE
      step := 100;
    END_IF

  2:
    lpBuffer := ADR(lpArray[1]);
    dRecvSize := NBS.SysSockRecv(hAcceptSock, lpBuffer, 6, 0, ADR(hResult));
    IF (dRecvSize < 0) OR (hResult = NBS.RTS_INVALID_HANDLE) THEN
      dRecvSize := 0;
    END_IF
    IF dRecvSize > 0 THEN
      sdBuffer := ADR(lpArray[1]);
      dSendSize := NBS.SysSockSend(hAcceptSock, sdBuffer, 6, 0, ADR(hResult));
    END_IF

  3:
    hClose := NBS.SysSockClose(hSocket);
    hClose := NBS.SysSockClose(hAcceptSock);

  100:

END_CASE

```

5.3 其他辅助功能

机器人控制器 SRM1000 的操作系统为 Linux, 且已经裁剪掉了绝大部分的常见功能, 譬如远程访问等。这样, 用户就无法像在 Windows 下面简便的通过远程访问或者本地访问控制器的文件系统。

但控制器提供了一些简单的功能，让用户能够通过 ftp 或者其他方式来对控制器进行访问。

5.3.1 访问 FTP 服务器

用户打开 Windows 的文件浏览器 Explorer.exe，在地址栏输入：ftp://192.168.39.220，即可访问控制器 SRM1000 的文件系统。或者用户也可以使用 ftp 客户端登陆也可。

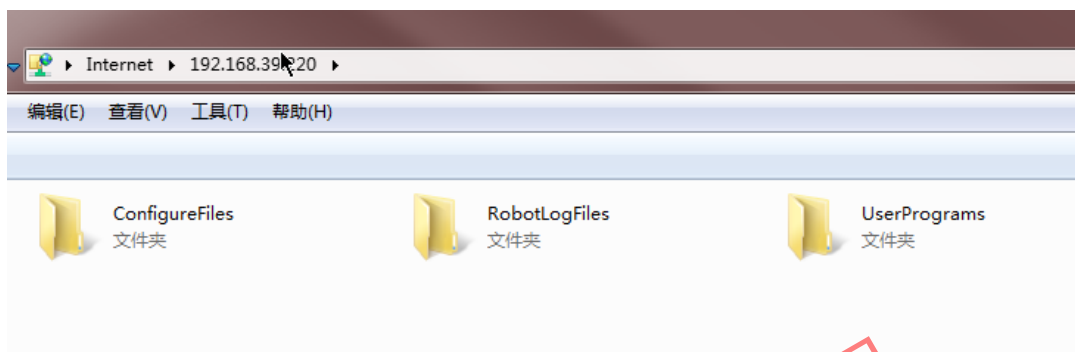


图 5.4 访问 SRM1000 文件系统

长沙工控帮教育科技有限公司



技术支持

◆ 技术服务

上海新时达机器人有限公司乐于提供有关机器运行及操作的信息，并可帮助您排除故障和提供详细咨询，如果您的机器人生产过程中出现故障，可立即联系我们的服务机构，并尽可能提供以下信息：

- ✧ 机器人型号及序列号
- ✧ 控制系统型号及序列号
- ✧ 控制系统系统版本号
- ✧ 额外的软件功能包（可选）
- ✧ 现有的应用程序
- ✧ 其他附加装置（变位机、导轨等，可选）
- ✧ 问题描述、故障持续时间及频率等

◆ 联系方式

上海新时达机器人有限公司

长沙工控帮教育科技有限公司