AK800 系列

可编程逻辑控制器编程手册

Ver.1.1

目录

前言

CoDeSys软件是AK800系列控制器所使用的基于Windows 操作系统的编程工具，是对控制器进行硬件配置和软件编程的标准软件包。其主要特点如下：

- 完全符合 IEC61131-3 标准，支持 LD、IL、ST、FBD、SFC、CFC 等多种编程语言；
- 支持自定义的数据类型，编程灵活，程序执行效率高；
- 具有丰富的扩展指令，支持用户自定义库，提高程序的复用性和功能扩展能力；
- 强大的数学运算功能，支持浮点数运算，支持多维数组；
- 强大的软件仿真、联机调试及程序检查能力。支持软件仿真，具有单步、单循环、设置断点、强制变量等功能；
- 完善的视图、报警和日志功能，可以通过视图功能实现控制过程的可视化；
- 强大的密码保护功能，可设置 8 个不同等级的密码和权限。
- 方便易用的总线配置工具，可以轻松配置总线设备。

导读

本软件手册的目的是为了协助您通过 CoDeSys 编程软件设计出一套完善的控制器控制程序，主要介绍如何使用 CoDeSys 软件和标准编程语言编写控制程序。

第一章：介绍了 CoDeSys 软件的安装，卸载及目标平台文件的安装。

第二章：对 CoDeSys 做了概述，同时向您详细描述了 CoDeSys 软件编程环境，包括主界面、菜单栏、快捷工具和对象组织器等。若您需要了解 CoDeSys 软件菜单或者快捷方式的选项，可以参考此章的内容。

第三章：快速入门，通过一个简单的例子，向您介绍了CoDeSys软件使用的基本步骤和基本使用方法。对于初学者，建议仔细学习此章的内容。

第四章：介绍了AK800系列控制器的存储区分配和CoDeSys对于变量的管理，包括地址和变量的定义、变量的分类等。假如您在地址或变量使用中有什么问题，请以参考此章的内容。

第五章：主要讲述CoDeSys对POU的管理，如：POU的建立、调用等。

第六章：在第五章的基础上，此章主要描述了控制器的工作方式以及任务的管理和配置。多任务可以高效的处理并行工作，无论您是有经验用户还是初次接触CoDeSys，您都应仔细阅读本章。

第七章：在了解了CoDeSys软件中如何管理地址、变量和POU后，此章主要讲述如何编写程序。以ST语言为例，介绍了工程的创建和配置，包括新建、控制器配置、添加库文件、制作库文件等。此章将有助于您熟悉软件的使用以及开始编写一个新工程。

第八章：编写完程序后的编译、下载和调试等步骤是此章的主要内容。您需要执行的相关操作或遇到的问题都在该章中有所涉及。

第九章：主要介绍了CoDeSys软件中的FBD、IL、ST、SFC等编程语言的使用。当您需要使用这些语言，请仔细阅读此章相应内容。

第十章：讲述了控制器常用应用指令，包括基本IEC指令，基本应用指令，扩展的串口通讯、CANopen通讯、以太网通讯和EtherCAT通讯等。当您用到这些功能时，请查阅此章相应内容。

第十一章：讲述了控制器如何修改 IP、如何与触摸屏连接、如何与伺服连接等，您用到这些功能时，请查阅此章相应内容。

第一章 安装指南

本章主要介绍 CoDeSys 软件的安装和卸载，并对安装目标平台文件进行详细的描述。

章节讲述软件的安装过程，1.2 章节讲述软件的卸载。假如您对 Windows 操作很了解，您可以跳过此二节。1.3 章节介绍了如何安装目标文件。

CoDeSys 是一款功能强大的控制器通用编程软件，安装目标平台的目的是将 CoDeSys 配置为 AK800 系列控制器所适用的编程环境。若您是第一次接触 CoDeSys，希望您能仔细阅读此章的内容。

1.1 CoDeSys 软件安装

Kinco 步科 AK800 系列控制器编程软件 CodeSys V3.5 为免费软件：

1、在 CodeSys 官网：http://www.codesys.cn 的资料下载页面进行下载；

2、在 Kinco 步科官网：https://www.kinco.cn 下载关于 AK800 系列控制器应用参考资料。

图 1.1-1：CODESYS 图标

注：在安装之前，强烈建议关闭电脑杀毒软件，安全卫士等！

在装有中文 Windows 操作系统的计算机上，双击打开 CoDeSys 安装程序，出现 CoDeSys 安装界面。首先提示的是需要安装 Visual_C++以及 Framework 4.0 软件，如果电脑已经安装这些软件则不提示，提示则需安装，请确保电脑已连接 Internet 网络。

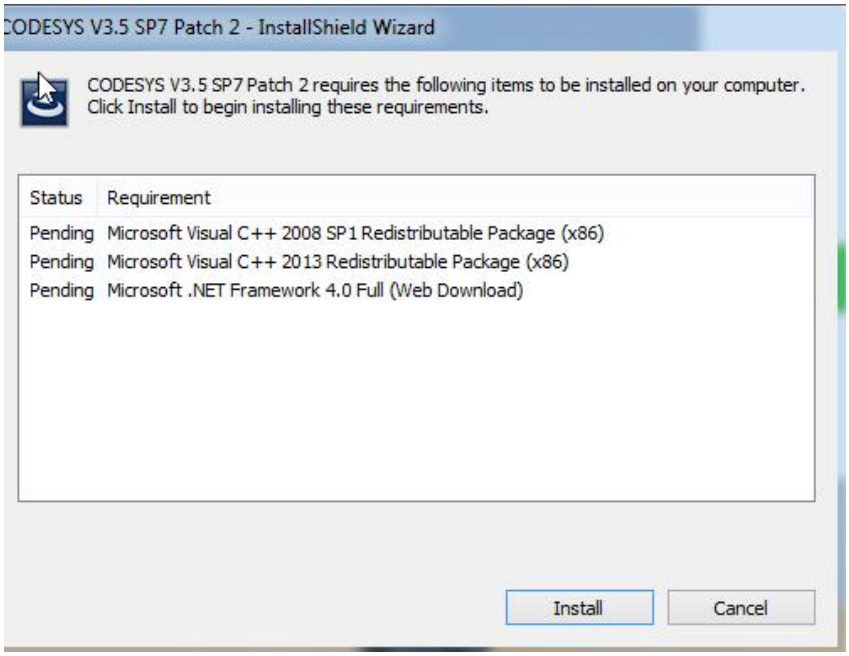


图 1.1-2：CODESYS 安装界面-1

安装程序安装必须的辅助软件后，出现安装界面，点击“next”，然后选择“yes”，到达设定安装的路径，如所示：

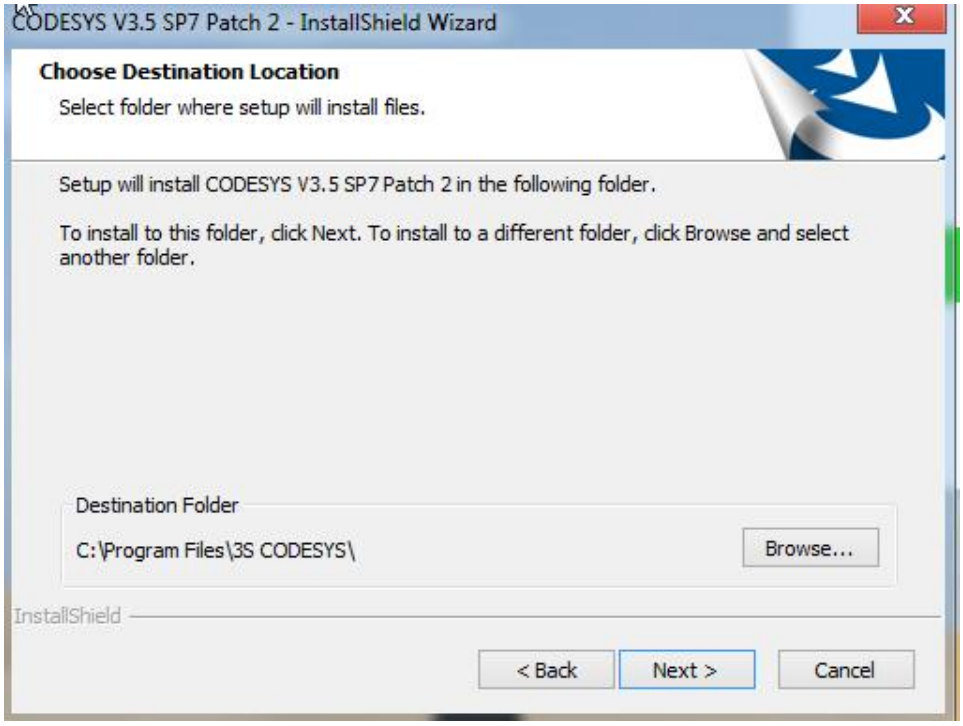


图 1.1-3：CODESYS 安装界面-2

点击“Browse…”按钮可以自定义安装路径，默认路径是 C：\Program Files\3S CODESYS（推荐使用默认路径安装）。然后点击“Next”按钮选择需要安装的组件，如下图所示：

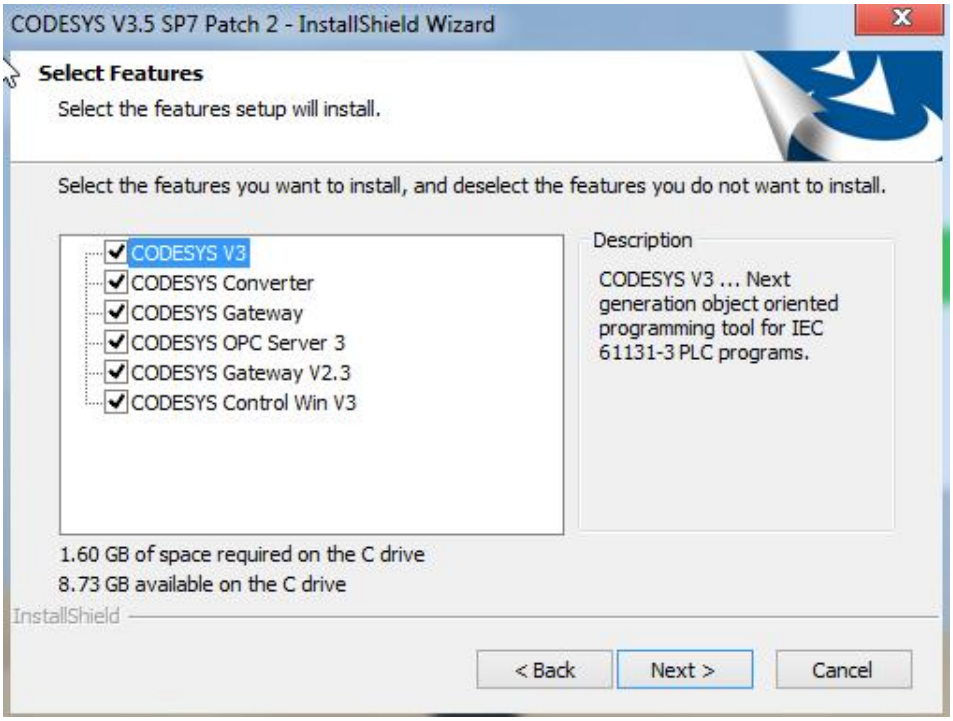


图 1.1-4：CODESYS 安装界面-3

组件的安装是可选择的，推荐安装全部组件以便使用所有功能。最后等待安装完成。

1.2 CoDeSys 软件卸载

如果计算机中已经安装了低版本的 CoDeSys 软件，当安装新版本的 CoDeSys 软件时，需要先将旧版本的 CoDeSys 软件卸载。在“控制面板”>>“程序”>>“卸载程序”中选择 CoDeSys，点击“卸载”便可对该程序进行卸载。

在卸载之前，必须要先退出桌面右下角系统托盘中 “Gateway.exe”和“ENI server”程序！

1.3 安装目标文件

要在软件工程“设备”选项中启用 AK800 控制器，必须在“包管理器”中先行装载 AK800 的软件包（文件格式： *.package），具体文件请从 Kinco 官方网站：https://www.kinco.cn 下载或咨询相关代理商或本公司客服部获取。

概述：启动 CoDeSys V3.5 后，在“工具”选项中选择“包管理器”（CODESYS Installer）安装 AK800 支持软件包，它包含使用 AK800 建立程序控制的一个标准环境所必需的所有文件和配置信息。

安装步骤：

①步骤 1 图示：在“工具”选项中选择“CODESYS Installer”，或双击“CODESYS Installer”图标。

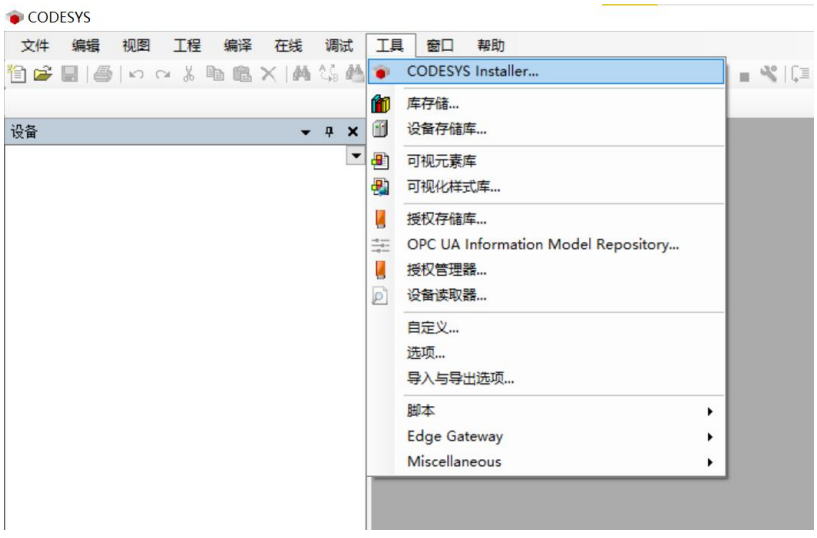


图 1.3-1：安装 AK800 软件包步骤 1

图 1.3-2：打开 CODESYS Installer 安装 AK800 软件包

②步骤 2 图示：点击“Install file”，打开文件列表。（若从图标进入，则需要点击“Change”进入如下界面）。

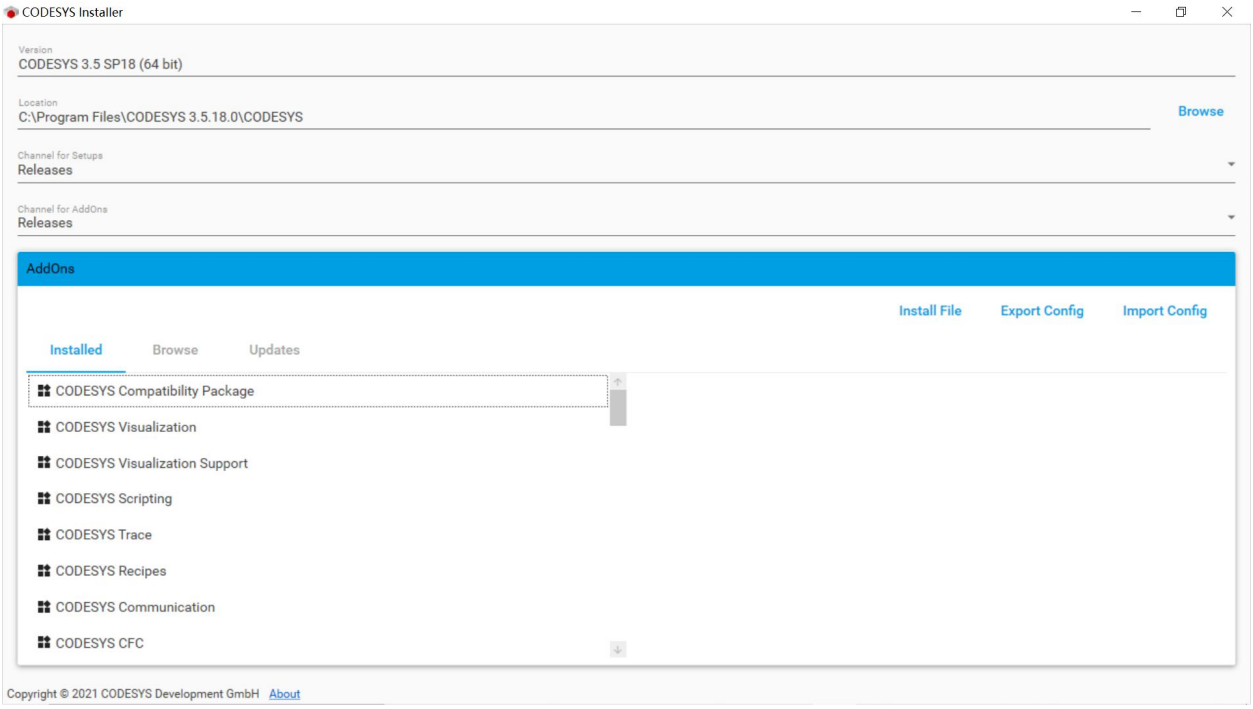


图 1.3-3：安装 AK800 软件包步骤 2

③步骤 3 图示：找到 AK800 设备软件包的储存文件夹，选中该文件并打开。

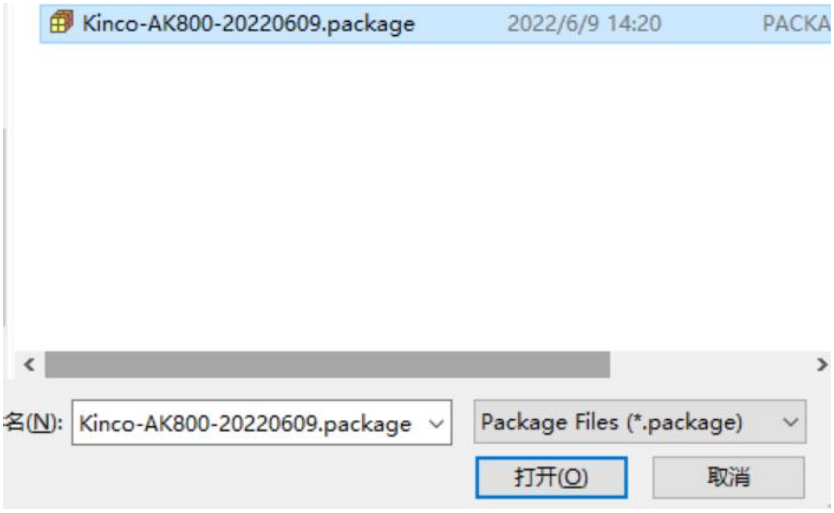


图 1.3-4：安装 AK800 软件包步骤 3

④步骤 4：打开以后根据提示进行完全安装，等待安装完成。

⑤步骤 5：等待安装完成，点击“Finished”退出向导。

⑥步骤 6 图示：安装完成以后可以在“Installed”查看到已安装的文件及版本等详细信息，接下来就可以新建工程，进行编程和调试。

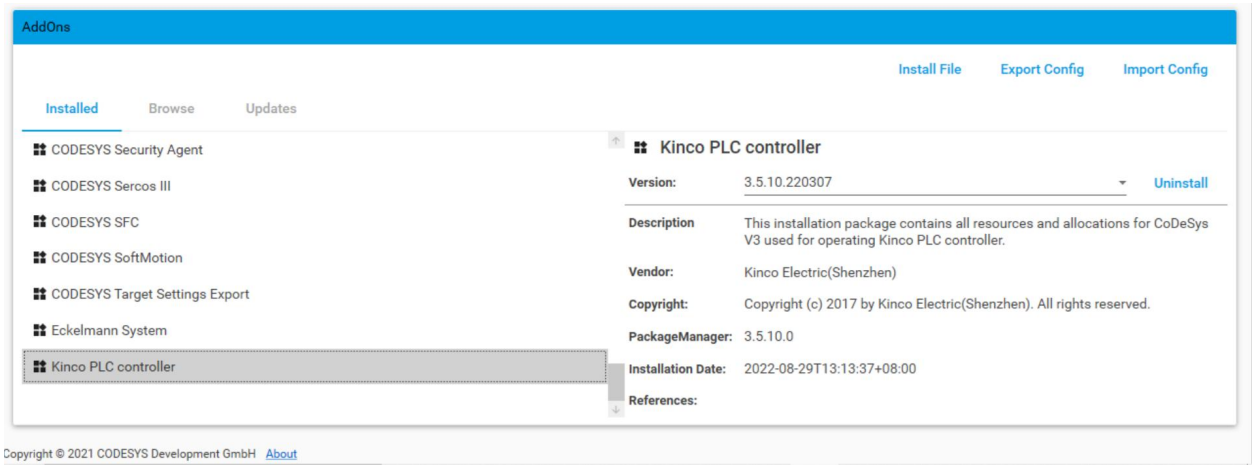


图 1.3-5：CODESYS Installer 界面中显示已安装的包

1.4 更新目标文件

当目标平台需要进行更新时，也可参照 1.3 中的操作进行更新，导入新的平台文件并安装即可。

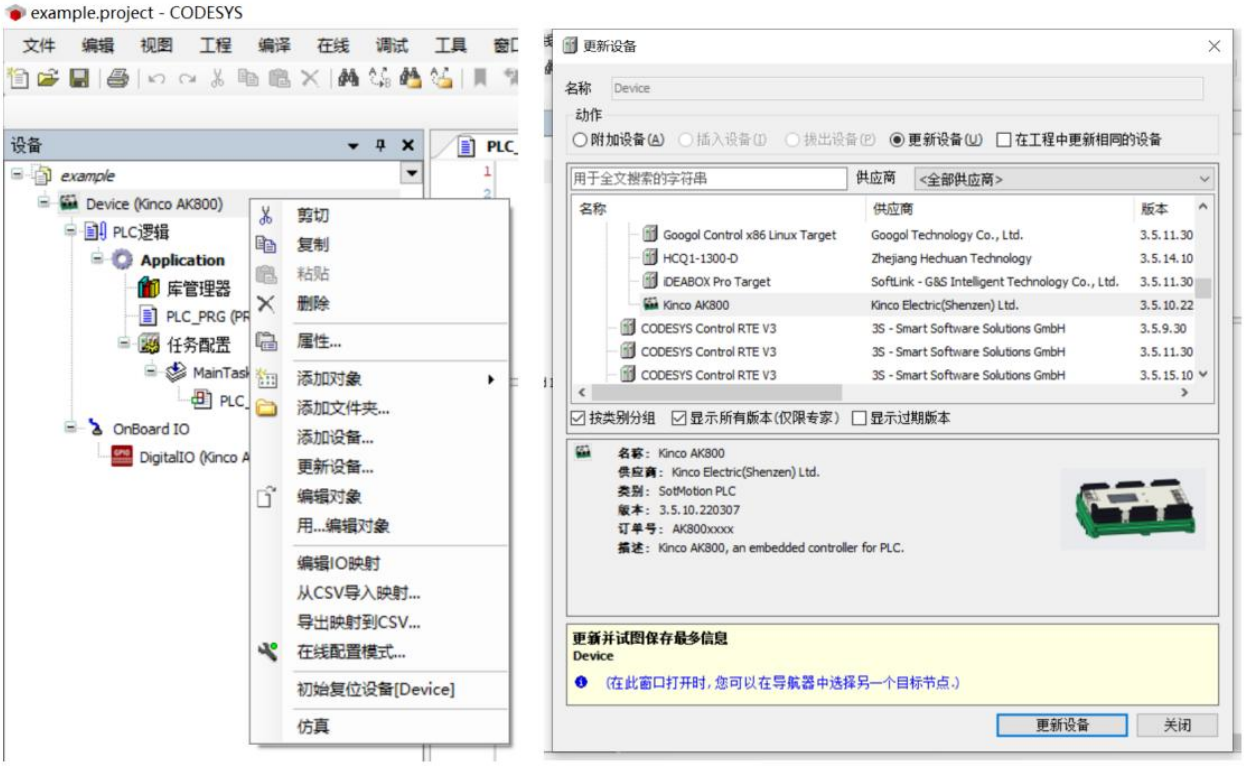


图 1.4-1-a：右击“Device”选择更新设备图　　1.4-1-b：选择设备版本进行更新

当软件版本或者目标平台有更新时，相应的库文件等可能都需要更新，此时打开用低版本编辑的工程文件时，需要对原来的库文件以及目标平台分别进行更新，如下图：右击设备栏“Device”，点击“更新设备”，在弹出的设备窗口中选择“显示所有版本”，选中需要的版本，点击“更新设备”即可。

第二章 CoDeSys 概述

2.1 CoDeSys 简介

CoDeSys 是可编程逻辑控制控制器的完整开发环境（CoDeSys 是 Controlled Developement System 的缩写），在控制器程序员编程时，CoDeSys 为强大的 IEC 语言提供了一个简单的实现方法。它支持 IEC61131-3 标准的 IL 、ST、 FBD 、LD、 CFC、 SFC 六种控制器编程语言，用户可以在同一项目中选择不同的语言编辑子程序、功能模块等。

变量是 CoDeSys 特有的一个概念，类似于高级语言的形式。变量在使用前需要声明，变量名还可按其功用命名，比起器件编号（如传统的 PLC 图形化编程中固定的定时器编号（T0、T1……）、计数器编号（C0、C1……）等）更容易辨认。变量还可分为全局与局部、输入与输出、掉电保持与不保持等多种类型。同时，利用 CoDeSys 强大的计算功能，还可以定义多种的数据类型。不仅包括布尔型、字节型、字型、双字型，而且还包括指针、枚举、多维数组、单精度浮点数等类型。关于变量的详细说明，请参见 4.4 章节。

CoDeSys 对程序的组织是完全模块化的。CoDeSys 提出 “POU”的概念，POU（Program Oginization Unit）即为程序组织单元。CoDeSys 的程序组织单元包括程序、函数和功能块，这三者共同组成了一个工程。CoDeSys 对程序的组织，都是主程序通过对其他 POU 的调用来实现的。这既便于多人参与编程，又便于程序的重复、阅读、调试，还可节省内存，确保程序安全。同时，CoDeSys 是一个开放的系统，用户可以根据需要，开发出适合自己的程序组织的指令，详细请参见第五章内容。

CoDeSys 是一个开放的系统。一方面，读者可以根据需要开发自己的指令；另一方面，CoDeSys 将控制器的许多参数和模块设定通过指令的形式开放给用户，用户可以根据自己的需求，在程序中完成设定，如串口通讯参数的设定等。

另外，CoDeSys 还提供了强大的仿真和调试功能，用户可以更方便地检查程序逻辑的正确性。关于仿真和调试功能，参见章节 8.4 。

2.2 CoDeSys 编程界面介绍

启动 CoDeSys 软件，创建一个新工程（此处为了较为完整地介绍 CoDeSys 编程界面，以已有工程界面进行统一介绍，新建工程操作介绍请参照第三章），进入如下图所示的编程界面。

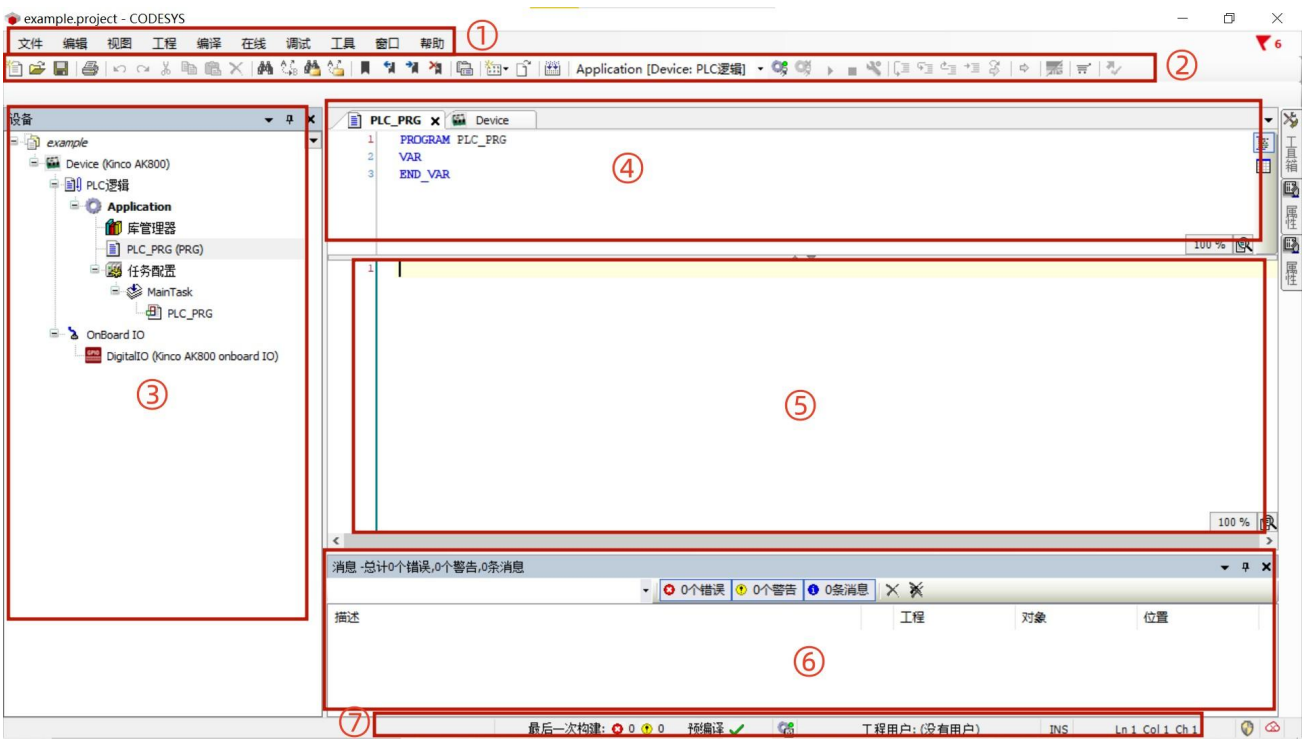


图 2.2-1：编程界面介绍

①：菜单栏　②：工具栏　③设备窗口　④变量定义区　⑤编程区　⑥：消息区　⑦：状态显示区

2.2.1 菜单栏

菜单栏涉及最常用的操作选项，如：项目新建、保存、程序编译、登录及下载等功能，模拟或在线调试等功能都需要菜单栏上的选项来实现，在 CoDeSys V3.5 中，常用的菜单栏功能列表概述如下表 2-1 所示。

表 2.2-1 菜单栏常用功能列表概述

名称	内容/功能
文件	涉及工程的新建、打开、关闭、保存、另存为、存档、打印及页面设置等
编辑	编辑器（如语言编辑器、声明编辑器）操作。
视图	激活某个特定窗口视图。
编译	编译工程对象和工程基本信息或清除编译信息。
在线	登录、退出控制器，加载控制器上的工程和复位等。
调试	控制运行在控制器上的程序（启动，停止）和调试操作（断点，单步，强制写入等）。
工具	该菜单包含的命令可以打开工具，这些工具用来配置工程的操作环境（例如：库和设备的安装、用户界面自定义、编辑器选项、加载和保存等）。
窗口	操作用户界面中的各个窗口（如排列、打开、关闭等命令）。与视图菜单功能类似。
帮助	打开在线帮助，获取系统帮助信息。

1.文件菜单如下图所示。

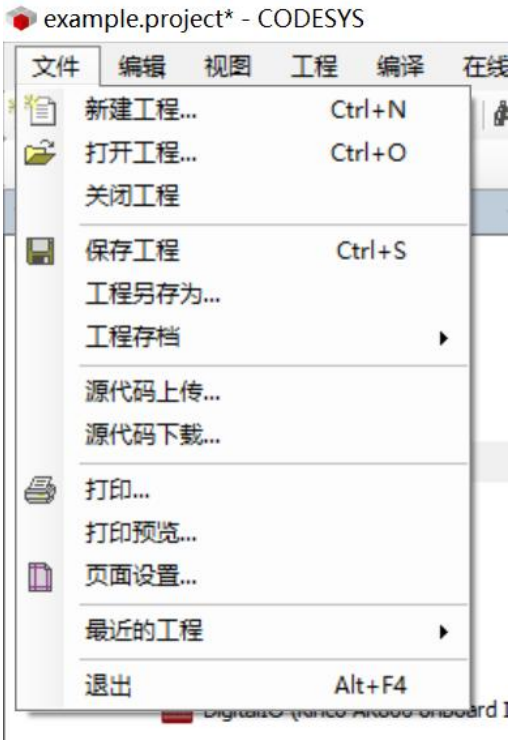


图 2.2-2：文件菜单目录

描述：

- 新建[Ctrl+N]：创建一个新工程。
- 打开[Ctrl+O]：打开一个工程。
- 关闭：关闭当前打开的工程。
- 保存[Ctrl+S]：对当前打开的工程进行保存。
- 另存为：将当前工程以新的文件名或路径或低版本格式保存。
- 工程存档：可以自动保存文档并将其压缩为“*.projectarchive”文件进行邮寄或者解压存档文件。
- 源代码上传：从设备中上传源代码上载到本地。
- 源代码下载：下载工程源代码到控制器中，下载后才可以从设备上传代码。
- 打印：打印当前窗口内容。
- 页面设置：对打印参数进行设置。打开此项会弹出对话框，可以选择打印机，对页面大小、份数和方向等参数进行设置，还可以对打印质量和打印布局进行设置。
- 退出[Alt+F4]：退出 CoDeSys 软件。

只发送工程文件给另外一个用户的时候，由于用户间的电脑系统环境配置不同，软件安装路径也不同，可能会导致另一个用户计算机上打开同样的程序编译后出现报错，一般报错的主要原因是缺少某些库文件或设备描述文件。为避免这一情况，建议用户在发送工程文件时采用格式为 “*.projectarchive” 的程序文件包，生成该格式文件的步骤如下：选择“文件”菜单下“工程存档”选项下的“保存存档”，在弹出的工程存档对话框中点击选中“所引用库”、“所引用设备”、“库配置文件”等保证程序正常编译的选项，点击确认生成格式为：“*.projectarchive” 的文件。同样，在接收到“*.projectarchive”文件时，其解压方式为：选择“文件”菜单下“工程存档”选项下的“解压存档”。

控制器支持上传控制器源代码的功能，默认软件联机下载工程至控制器时，并不会下载工程源代码至控制中，只有执行了“文件”菜单下的“源代码下载”操作时，工程的源代码才会被下载到控制器中。当需要上载控制器中的源代码至计算机时，则执行“文件”菜单下的“源

代码上传”操作即可。

2.编辑菜单如下图所示。

图 2.2-3：编辑菜单目录

描述：

- 撤销[Ctrl+Z]：撤消上一次操作。
- 恢复[Ctrl+Y]：重复上一次操作。
- 剪切[Ctrl+X]：把所选内容复制到剪贴板，并从当前位置删除。
- 复制[Ctrl+Insert]：把所选内容复制到剪贴板，但不删除所选内容。
- 粘贴[Ctrl+V]：把剪贴板上的内容粘贴到当前位置。
- 删除[Delete]：删除当前所选内容
- 全选[Ctrl+A]：选中当前页面所有内容
- 查找 替换：在当前编辑器中查找某一文本或者将查找到的目标替换成所需内容。
- 浏览：可以浏览当前变量的定义处或交叉索引处
- 将文件作为文本插入：将文本文件（TXT）的内容插入到当前打开的文本编辑器中光标所在的位置
- 高级：用于文本编辑器中，可以转换大小写，转到指定行，转到匹配括号，展开和收起当前完整语句等等
- 书签：用于在文本编辑器的当前行添加一个书签或者删除已有书签等
- 输入助手[F2]：可以快速输入相关内容。在编辑窗口中的当前光标位置，按 F2 功能键，会自动弹出当前位置可以插入的待选项，例如运算符、函数、功能块和变量类型等列表。在左边的列表中选择输入类型，在右边的列表中选择期望的输入，选择所需输入的内容，点击“确认”按钮，则所选内容便被输入。具体请参见 7.4.3 章节。
- 自动声明[Shift+F2]：弹出变量定义对话框。
- 下一个消息：自前向后查找并显示消息窗口的提示错误或警告。
- 前一个消息：自后向前查找并显示消息窗口的提示错误或警告。
- 转到源代码处：消息窗口中当前光标选定的消息源代码处（错误产生地）。

3.视图菜单如下图所示。

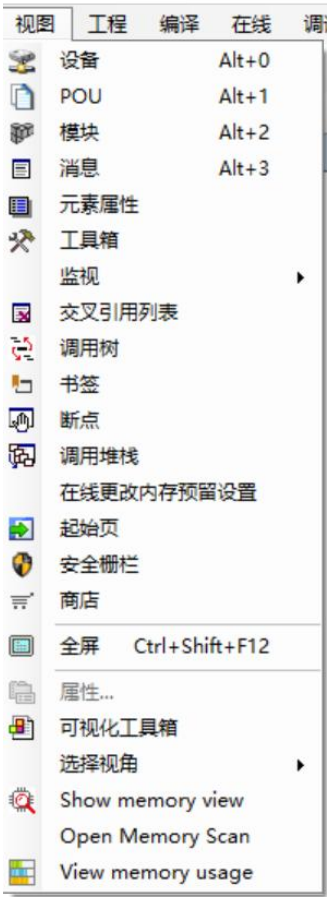


图 2.2-4：视图菜单目录

描述：

- 设备：在设备视图窗口下，该项目所需的工具以节点的形式显示在与项目同名称的根目录下。在每个控制器逻辑器件下都有相应的应用以及它们所关联的项目响应。更多的子设备是需要插入的。
- POU：在 POU 视图窗口中可以组织所有当前工程（控制器程序）的编程单元，并且可以在特定应用中演示这些单元的应用。
- 模块：显示设备所连模块等。
- 消息：打开信息提示窗口。
- 元素属性：SFC 元素的特性可以在“元素属性”窗口被视图和编辑。
- 工具箱：此命令用于打开当前编辑器（FBD>LD>IL）的工具箱。工具箱通常带有图形语言编辑器或可视化编辑器，并且通过拖放可以将图形化的编程元素插入到编辑器
- 监视：可对变量进而监视，用户可根据需要自定义四个监视或直接监视所有强制变量。
- 断点：用来打开断对话框。在对话框中，可以查看和修改当前工程中设置的所有断点参数、新建或删除断点、启用或禁用断点。
- 交叉引用列表：打开一个视图窗口，这个窗口列出了工程变量的交叉引用。交叉引用是指变量被使用的地点，这个列表可以包括整个工程的交叉引用或只包含定义变量的 POU 中的交叉引用。
- 调用树：用来打开调用树窗口。当在“在线”模式下对程序步进调试时，使用该命令会显示当前指令在代码中的位置，以及完整的调用路径
- 在线更改内存预留设置：将来对项目的功能块进行重大更改，这可能会导致功能块实例在在线更改期间被复制到其他存储位置。在将功能块下载到设备前，为功能块配置内存预留设置，可以使在线修改速度更快。
- 起始页：打开一个视图来提供选择命令的快速启动与新的或最近的项目，版本信息，以及浏览器 CODESYS 的主页
- 全屏：激活该选项，CODESYS 框架窗口将以全屏幕模式显示。 取消激活菜单项，或按 [Ctrl]+[Shift]+[F12]，可以切换回原来的显示模式。
- 属性：显示当前选中的项目在 PUO 中的属性或者是设备上的各类标签。显示类型取决于项目类型。
- Show memory view：查看内存。

4.工程菜单如下图所示。

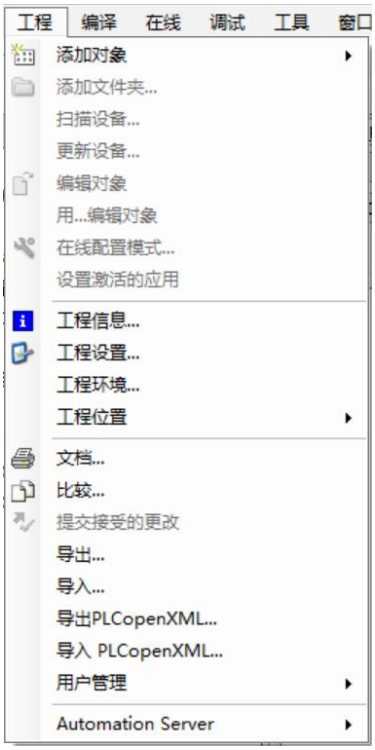


图 2.2-5：工程菜单目录

描述：

- 添加对象：打开一个子菜单，用于在当前选中的 POUs 或者设备树中添加一个对象。例如一个应用对象只有在设备窗口中选中一个设备对象时才能够被添加，一个动作对象只能在一个编程对象选中或者被标记的情况下才可以被添加。
- 添加文件夹：在当前选中的 POUs 或者设备树中添加一个文件夹。
- 扫描设备：启动前进行扫描连接。在一个对话框内检测和查看硬件组件使用户能够将它们映射到工程中的设备树中。此功能需根据目标平台确定，不是所有平台都支持。
- 更新设备：用新设备替换当前设备树中选中的设备，替换原有设备版本或者设备类型。此功能需根据目标平台确定，不是所有平台都支持。
- 编辑对象：将在适当的编辑器中打开对象。
- 用...编辑对象：将在适当的编辑器中打开对象。
- 在线配置模式：该命令用于打开和关闭在线配置模式。打开时，它会建立与 PLC 的连接并在那里加载隐式创建的应用程序.
- 设置激活的应用：使用这个命令去定义那个定义在设备视图中的应用作为当前活动应用。因为在登录的时候只登录当前活动的应用。
- 工程信息：查看和定义项目文件的属性和信息,如访问属性、版本号、作者和公司信息以及统计有关项目对象。
- 工程设置：用于进行版本设置，比如工程加密，用户以及访问权限，版本处理，以及用于打印的布局定义等。可获得的类别取决于当前安装的软件包。
- 工程环境：用于查看当前工程的工程版本信息，如库版本，编译器版本等。
- 比较：比较当前打开的工程与另外一个选择的工程的区别。
- 导出：导出特定的对象或者整个对象树到一个或者多个导出文件。导出文件默认创建格式为 XML 并且后缀扩展为 ".export"。
- 导入：用于导入一个或者多个通过 xml 格式导出的对象文件（后缀扩展为 ".export"）。
- 导出 PLCopen XML：用于导出所有或者当前工程中选中的对象，并根据 PLCopen 规定的格式导出为“*.xml”文件的形式。当然，前提是所导出的对象是符合 PLCopen 规范的
- 导入 PLCopen XML：导入以 PLCopen 形式的“*.xml”文件描述的对象。当然，前提是所导入的对象是符合 PLCopen 规范的.
- 用户管理：类似 Windows 操作系统一样，对用户进行登录，授权，退出等操作.

5.编译菜单如图所示。

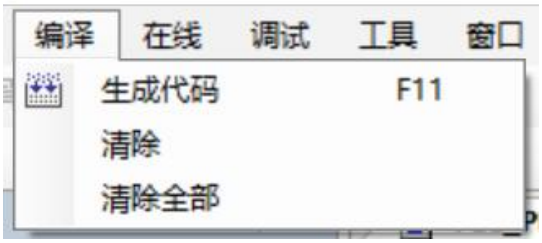


图 2.2-6：编辑菜单目录

描述：

- 生成代码[F11]：进行当前激活应用的编译。 这意味着所有属于这个应用的对象都将被进行检查。注意没有汇编代码将会产生于下载一个应用相同的结果！编译过程在登录一个改变的应用程序时自动进行。
- 清除：用于删除当前激活的应用的编译信息。编译信息是在应用进行最新一次下载之后保存在工程路径中的 *.compileinfo 文件。经过一个清除操作之后在线改变对当前的应用将成为可能。程序首先必须经过重新下载。
- 清除所有：删除所有应用程序的 编译信息。 此信息是在应用程序最后的下载中被创建的，它被保存在项目目录中的一个文件（*.compileinfo）里。此命令必须在重新登录之前下载。因此，在线变化不再可能实现。从 CODESYS 的 V3.5.3.40 版本起，对持久性变量的内部信息也被删除，因此，它们将在下载后重新初始化。

6.在线菜单如下图所示。

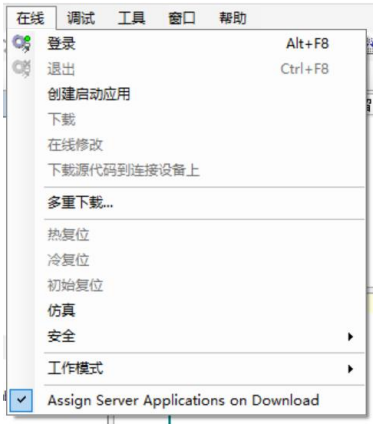


图 2.2-7：在线菜单目

描述：

- 登录(Alt+F8)：建立控制器与 CoDeSys 的连接。当 CoDeSys 程序与控制器内部一致时，自动进入调试状态；当两者不一致时，则提示是否下载程序。
- 退出(Ctrl+F8)：退出调试状态，切换到程序编辑状态。
- 创建启动应用：将下载到控制器中的工程作为控制器运行的默认工程。所谓启动工程，是指保存在控制器的 Flash 中且上电后可以运行的用户程序。如果不执行此操作，则控制器在断电后，原先在控制器中的运行程序将丢失。
- 下载：把工程装载到控制器中。这个只有在建立了控制器与 CoDeSys 的连接以后才有效。关于下载与登录的区别，请参见 8.3.3 章节
- 在线修改：是已经运行的工程有一部分发生了变化，且只需要将变化的部分重新装载到控制器中。注意在清除所有或者清除操作之后是不允许发生的！
- 下载源代码到连接设备：用于创建实际工程存档文件，并传到任意设备中。有关源码下载命令的目标设备、内容和时间，可通过工程-工程设置，类别“源代码下载”进行设置。
- 多重下载：实现多种应用下载。
- 热复位：用于复位除了保持 (retain, persist“EN”t) 变量的变量，所有应用程序中激活的变量都将会被初始化。
- 冷复位：除了“通常”以及保持变量外当前激活的应用程序中的返回变量，也将会被复位为初始值。初始状态为程序刚刚被下载到 控制器中并位于程序的开始时 （“冷启动”）。
- 初始复位：可以复位当前活动应用的所有变量，复位所有相关变量到初始值并擦除 控制器 上的应用程序，再次运行时将提示是否下载程序。
- 仿真：如果仿真模式被选择，则选中符号“√”将出现在菜单项的前面。在仿真模式下，用户程序运行在操作系统平台下的本地计算机内。此模式可用来检查工程。如果不运行于仿真模式，那么可以直接将程序运行于控制器中。
- 安全：主要可以设置用户组及用户密码等，以便对工程分级管。
- 工作模式：可以设置为锁定模式 Locked 或者调试模式 Debug，。

7.调试菜单如下图所示。

图 2.2-8：调试菜单目录

描述：

- 启动（F5）：启动设备中的应用程序。如果程序运行后到达断点，则程序终止，为了继续程序的运行，可以通过执行“运行”，或者进行“单步跳过”等操作。
- 停止（Shift+F8）：停止设备中的应用程序。
- 单循环：对当前激活的应用只执行一次扫描，而不是循环扫描。
- 新建断点：在当前位置设置一个断点并自动弹出断点属性设置窗口。新建断点：建立一个新的断点（默认断点禁用）。
- 设置或清除断点：设置并启用一个新断点或清除已经设置好的断点。
- 禁用断点：禁用某个断点。
- 使能断点：启用某个断点。
- 跳过：可以用来跳过当前断点或者单步执行 SFC 程序等。
- 跳入：通过此命令将会单步运行。程序会在下条指令前停止。如果必要会有一个打开的 POU 程序之间的切换。如果当前断点位置是一个函数或者功能块的调用，那么下一步程序将执行到调用的 POU 中程序的第一行。如果在当前位置是函数或功能块，则程序将执行到被调用程序的第一条指令。在其它情况下，与“联机>>跳过”命令一样。
- 跳出：如果应用程序中不包含任何的调用，“跳出”将会执行到应用程序的开始部分。如果当前的位置是在一个调用的 POUs 内部，“跳出”将会执行到实例调用的地方（注意：此处的跳出不同于 CODESYS V2.3）。所以在嵌套调用的情况下，“跳出”将会使程序一步一步的返回。这利于程序返回并跳入另一个 POU 调用内。
- 运行到光标处：相当于在程序运行的下一个位置设置断点并运行到断点处。将光标移动到描述的“下一个位置”并且处理命令。在当前断点和现在光标之间的代码将会被处理。
- 设置下一条语句：将光标放在所需的位置并执行命令，用于定义下一条指令。
- 显示当前语句：可以用于在线模式下返回到当前处理位置。这在程序跳转到其他的程序窗口中是非常有用的，或者在将光标移动到编程系统的其他位置的时候。之前处理的 POU 窗口将会再次出现，并且将光标移动到当前处理位置。
- 写入值（Ctrl+F7）：将控制器中的某个变量设置成一个自定义的值，此设定值在下次循环开始时有效。该命令影响当前当前激活的应用准备写入的所有变量。
- 强制值（F7）：同样用于调试时对变量赋值。在每个循环结束之后，被强制的变量都被写入强制值，直到执行“解除强制命令”命令为止。
- 释放值（A）：终止变量的强制命令。

- 流控制模式：用于打开或者关闭顺序控制（能量流）功能，此功能可以用在 ST, FUP, LD 以及 IL 编辑器中。打开流控制，当前变量的值，函数的调用和操作结果，都将以精确位置和实时执行情况在编辑视图里显示。相比之下，标准监控功能只是提供一个变量在两个执行循环中的值。打开流控制会增加扫描周期，调试结束后请关闭。
- 显示模式：选择变量显示的格式为“二进制”、“十进制”、“十六进制”。

8.工具菜单如下图所示。

图 2.2-9：工具菜单目录

描述：

- 包管理器：可以对硬件制造商提供的设备软件安装包进行管理。
- 库：该对话框显示当前定义的函数库的位置（资料库）以及安装的函数库。存储的函数库可以增加，修改或者从这个对话框中移除并且函数库也可以安装或者移除。
- 设备库：打开对话框设备管理。一个设备管理库是在本地系统上安装的设备的数据库，用于 CODESYS 开发系统使用设备。在设备管理库中可以添加或者移除设备的安装。
- 可视化类型库：可添加可视化类型的库和管理等，但前提是你的设备支持可视化。
- 授权存储库：添加许可授权。
- 授权管理器：使用证书管理器来对在 CODESYS 上的加密狗或软件容器的附加产品的证书进行管理。软件容器不是一个加密狗，而是一个软件解决方案。它可以在没有加密狗的情况下用来激活特定计算机的硬件证书。
- 脚本：用于添加一个 Python 动态语言编写的脚本文件或者执行脚本跟踪。
- 自定义：对工具栏进行重新排列或者增加、减少等。
- 选项：对软件和 CODESYS 编程系统接口进行配置，如语言设置，各编辑器的基本设置。
- 设备读取器：该命令打开标准对话框“选择设备”选择一个控件后，读出该控件的许可证和产品信息。此许可证和产品信息显示在对话框中 设备许可证阅读器显示。

9.窗口菜单如下图所示。

图 2.2-10：窗口菜单目录

描述：

- 下一个编辑器：如果当前打开了多个编辑器窗口，该命令命令可用于将焦点从当前窗口转移到下一个窗口，即当前活动窗口选项卡的右边选项卡所代表的窗口。
- 上一个编辑器：如果当前打开了多个编辑器窗口，该命令可用于将焦点从当前窗口中转移到前一个窗口，即当前活动窗口选项卡左边选项卡所代表的窗口。
- 关闭所有编辑器：关闭所有打开的编辑器窗口。
- 关闭所有未激活的编辑器：关闭所有未激活的编辑器窗口。
- 重置窗口布局：重置软件编程环境各窗口布局。
- 新建垂直序列组：新建一个所有窗口在工作区垂直排列，不重叠，且填充整个工作区。
- 新建水平序列组：新建一个所有窗口在工作区水平排列，不重叠，且填充整个工作区。
- 浮动：窗口将变为“浮动”状态，并可以放置在屏幕的任何位置。可使用“固定”命令固定一个浮动窗口。
- 固定：把窗口固定在当前所在位置。
- 自动隐藏：窗口没有被选中时会自动隐藏。窗口隐藏后，将显示为 CoDeSys 用户界面边框上的一个选项卡，只有在单击此选项卡后，该窗口才可见。
- 上/下一个窗口：可用于有两个或多个窗格的窗口，可以获取上/下一个窗格。

10.帮助菜单如下图所示。

图 2.2-11：帮助菜单目录

描述：

- 目录：弹出帮助主题，并将帮助的相关项目列出，以供方便、快捷地查找帮助项目。
- 索引：弹出对话框，按照字母顺序排列列出帮助内容。
- 搜索：弹出对话框，提示输入所需查找项目的关键字。
- 关于：软件名称及版本信息

2.2.2 工具栏

通过点击工具栏上的图标，用户可以更快速地操作。当将鼠标指针短时停留在工具栏上的一个符号时，会提示该符号的名称。当用户选择不同的对象时，工具栏里的内容会根据不同的对象发生改变，如使用梯形图（LD）的 POU 时，工具栏内会出现“线圈”、“触点”等功能，而当选用功能块图（FBD）时，则会出现添加功能块等选项，如下< 图：工具栏 >所示。用户可以根据自己的使用习惯自定义工具栏。

图 2.2-12：工具栏图标

修改/自定义工具栏的操作为：点击菜单栏上的“工具”，点击其目录下的“自定义”，在出现的“自定义窗口中”，选择“工具栏”选项，在其出现的选项中选择需要的命令，进行“添加命令”或“删除命令”。如想要自定义工具栏，则可以选择“添加工具栏”命令，同样在在新添加的空白工具栏中选择需要的命令，进行“添加命令”或“删除命令”即可，修改或自定义工具栏后点击确定即可。如下图所示：

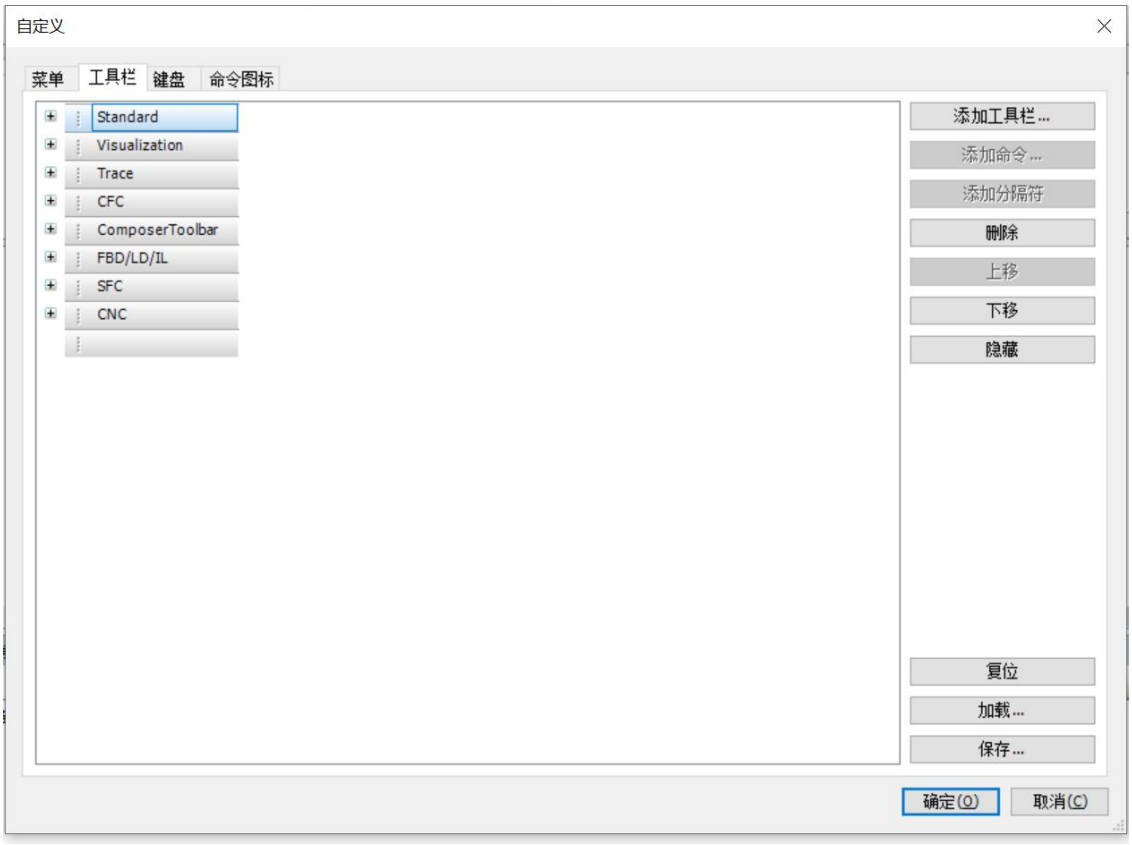


图 2.2-13：自定义工具栏

2.2.3 设备窗口

设备窗口包括所使用的到的设备选项卡和 POU。以树形结构管理工程中的资源对象和程序单元。在项目中，数据分层结构中以对象的形式进行保存。

设备选项卡这里包含工程中相关的设备，视图菜单中默认打开。主要用于完成硬件配置、库管理、工程选项等功能。

POU 对于工程的程序组织单元（POUs，DUTs 等）以树型结构进行管理（通过视图菜单打开）。主要用于对程序的管理。诸如新建子程序、变量，数据类型等都在此选项卡中完成。

2.2.4 变量定义区

CoDeSys 将数据分为地址和变量两大类。变量可以不指定具体地址，直接用符号来表示，同一符号的变量表示同一个变量。变量与地址不同，变量在使用时需要定义，而地址可以直接引用，<如图：变量定义> 所示。变量声明区就是用于显示所有定义的变量。

变量的定义有两种方式。一种是在编程时自动定义，并且显示在变量声明区中，另一种就是直接在变量声明区中定义。关于变量的定义，此处不进行详细描述，可参见 4.4 章节。

注意：CoDeSys 变量名是不区分大小写的，例如 Start 和 start,START 表示是同一个变量。

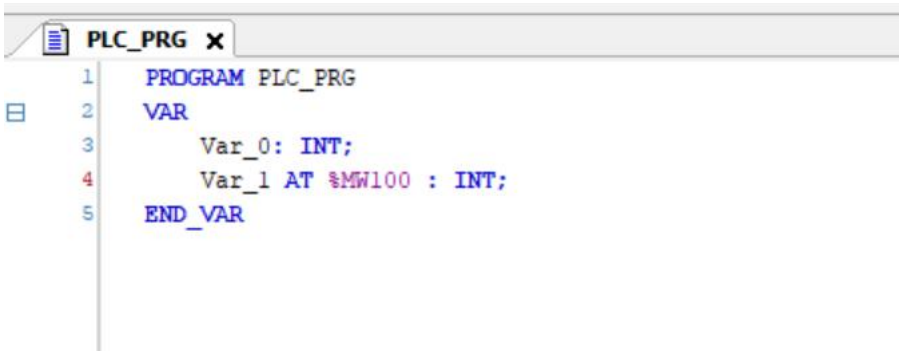


图 2.2-14：变量定义图例

2.2.5 编程区

编程区主要指程序、函数和功能块的编辑器窗口，用于编写控制算法。由于选择的编程语言不同，编辑环境也会有所不同。根据编程语言的特点，编程语言可以分为图形编辑语言和文本编辑语言两大类。LD、SFC、FBD 和 CFC 语言的编辑器属于图形编辑器。IL 和 ST 语言的编辑器属于文本编辑器，包含了 Windows 文本编辑器的所有通用功能。

关于编程区的操作，请参见 7.4 章节。关于其他编程语言，请参见第九章。

2.2.6 消息区

消息窗包含显示预编译、编译、生成器、下载和程序检查信息。用鼠标双击消息窗内的某条消息或右键点击“转到源代码处”，编辑器就会打开所选择行的对象。通过右键鼠标，点击

“下一个消息”或在选中某条信息后按下键的“F4（+Fn）”可以快速切换并对其到后一条错误信息的源代码，而右击选择“前一个消息”或在选中某条信息后按下键的“Shift+F4（+Fn）”组合功能键，则可以快速切换至前一个错误消息的源代码。消息窗口可设置为“自动隐藏”选项，也可以通过移动选择合适的摆放位置。

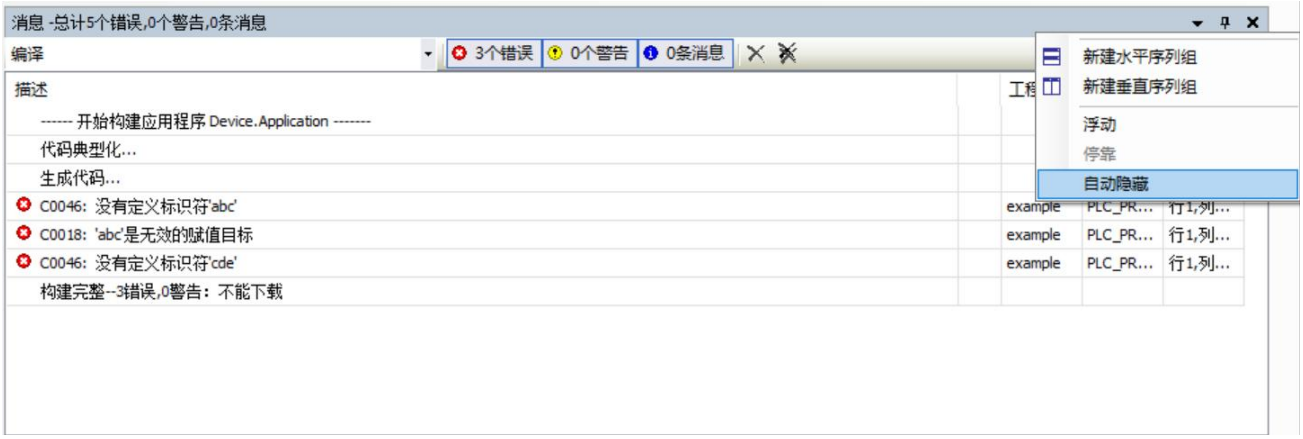


图 2.2.15：消息区信息

2.2.7 状态显示区

该区在线模式下显示控制器当前所处的模式以及状态，程序当前所在的位置等信息。

1. 在线模式下的状态描述如表 2-2 所示：

表 2.2-2：在线模式状态描述

状态/符号表示	说明
运行	程序正在运行。
停止	程序停止或未运行。
停止到 BP	程序停止于断点。
仿真	程序处于仿真模式。
程序下载	程序已下载到设备上。
程序未变	设备中的程序与编程系统中的程序一致。
修改程序(在线改变)	设备中的程序与编程系统中的程序不一致，需要在线修改。

2. 位置信息表示见图 ，位置信息说明见表 2-3：

图 2.2.16：软件界面位置信息

表 2.2-3：状态信息区的位置信息说明

名称/显示	描述
Ln/行	光标所指位置的行号。
Col/列	光标所指位置的列号。
Ch/字	光标所指位置的字符数，字符可以是单个字符或者数字，也可以是一个制表符。

2.3 快捷工具

首先介绍工具条中的快捷图标。如果想要知道各个快捷图标工具按钮的名称，可以把鼠标指针移至快捷图标工具按钮上，对应快捷图标工具按钮的名称就会出现在提示框中。菜单命令和工具按钮变灰表示该功能在当前窗口禁用。

2.3.1 文件工具

图 2.3-1：文件工具图标

如上图，从左到右依次快捷工具作用为：

- 新建工程：新建一个工程。
- 打开文件>工程：打开已经存在的文件>工程。
- 保存：保存文件。
- 打印：打印文件。

2.3.2 调试工具

图 2.3-2：调试工具图标

如上图，从左到右依次快捷工具作用为：

- 属性：当前对象的属性。
- 添加对象：添加一个对象。
- 编辑对象：编辑当前对象。
- 编译：对当前程序进行编译。
- 登录到：对当前工程登录到设备，进入调试状态。
- 退出：退出登录状态。
- 运行：启动登录程序的运行。
- 停止：停止程序的运行。
- 在线配置模式：在线配置参数。
- 跳过：单步执行程序，程序在执行之后停止。“跳过”命令遇到功能块或函数时，会跳过功能块执行下一条语句。
- 跳入：通过此命令将会单步运行。程序会在下条指令前停止。
- 跳出：跳出当前函数或程序。
- 运行到光标处：运行到光标所在处。
- 设置下一条语句：将光标放在所需的位置并执行命令，用于定义下一条指令。
- 显示当前语句：可以用于在线模式下返回到当前处理位置。

2.3.3 编辑工具

图 2.3-：编辑工具图标

如上图，从左到右依次快捷工具作用为：

- 撤销：撤销上一步操作。
- 恢复：恢复下一步操作。
- 剪切：将选中的部分剪切到剪贴板。
- 复制：将选中的部分复制到剪贴板。
- 粘贴：将剪贴板中的内容粘贴到当前窗口。
- 删除：删除当前选中内容。
- 在编辑器中查找：在当前已打开的活动编辑器中查找某一字符串。
- 在编辑器中替换：对已打开的活动编辑器中的某一字符串的内容进行替换。
- 在工程中查找：在整个工程中中查找某一字符串。
- 在工程中替换：对整个工程中的某一字符串的内容进行替换。
- 书签：对当前文本编辑器光标所在位置添加一个书签。
- 上一个书签：跳转到上一个书签处。
- 下一个书签：跳转到下一个书签处。
- 清除所有书签：清除已经设定的书签标记。

2.3.4 编程工具

工具菜单在不同编程语言下的表示形式是不相同的。按照编程语言种类的不同，编程菜单中的操作符的具体功能如下所述：

- LD 语言 ： 

功能依次为线圈、置位、复位、常开触点、常闭触点、串联右触点、向下并联触点、向下并联取反触点、向上并联触点、运算块、空运算块、带 EN/ENO 功能的运算块、带 EN/ENO 功能的功能块、插入跳转。

- FBD 语言 

功能依次为输出、运算块、空运算块、带 EN/ENO 的运算块、带 EN/ENO 的功能块、跳转、插入标号、插入返回、输入、取反、边缘检测、复位/置位、输出连接、插入分支、向下插入分支、向上插入分支、设置分支起始点/结束点。

- SFC 语言

➢ CFC 语言

对于 IL 和 ST 语言，无此类编程工具。具体快捷编程工具的应用，参见第九章。

2.4 对象组织器

主界面左侧的竖条窗口称为对象组织器，由“设备”、“POU”二个选项卡组成，包含了一个工程所必需的基本对象，如下图所示。

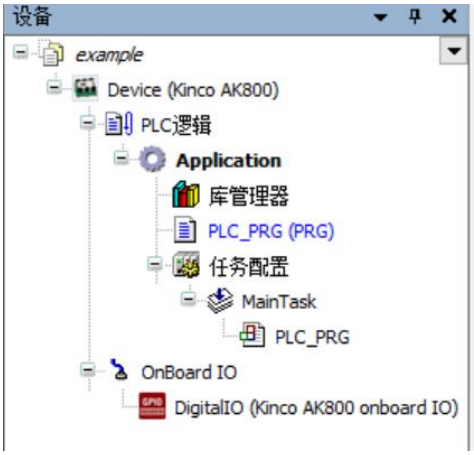


图 2.4-1：新建程序后的默认设备树组态

POU 对于工程的程序组织单元（POUs，DUTs 等）以树型结构进行管理（通过视图菜单打开）。主要用于对程序的管理。诸如新建子程序、变量，数据类型等都在此选项卡中完成。

2.4.1 设备树

设备选项卡这里包含工程中相关的设备，并以树型结构进行管理；视图菜单中默认打开。主要用于完成硬件配置、库管理、工程选项等功能。为较为全面地介绍设备窗口，下图以已有工程为例进行描述。

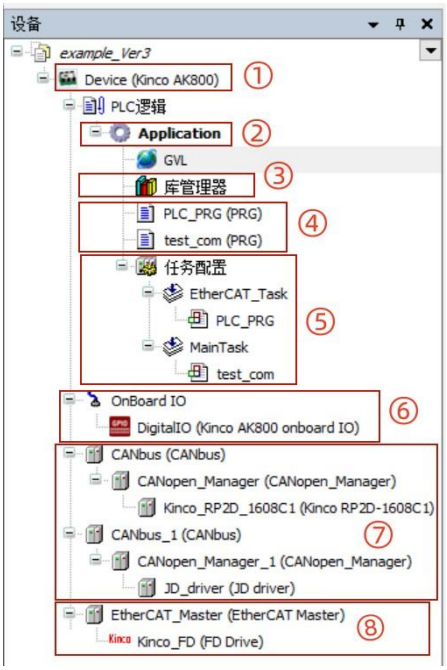


图 2.4-2：设备树列表

① 双击进入可进行设备通讯设置；
② 右击选择“添加”可添加 POU、DTU、CAM 表、全局变量表（GVL）等功能。
③ 显示当前使用的库文件，还可进行库文件添加、删除、更新等操作。
④ 程序部分，每个工程都需要起码调用一个主程序。
⑤ 创建周期不同的任务，可以让不同的 POU 按照特定的执行周期执行。
⑥ 本体 IO，为默认配置，包括 24 点输入和 24 点输出。
⑦ CAN 总线配置，AK800 可配置 2 路 CAN 总线。
⑧ EtherCAT 总线配置。

2.4.2 程序组织单元

“POU”选项用来定义全局变量、数据类型，程序管理和组织工程。下图为右击“Application”弹出的默认功能菜单。

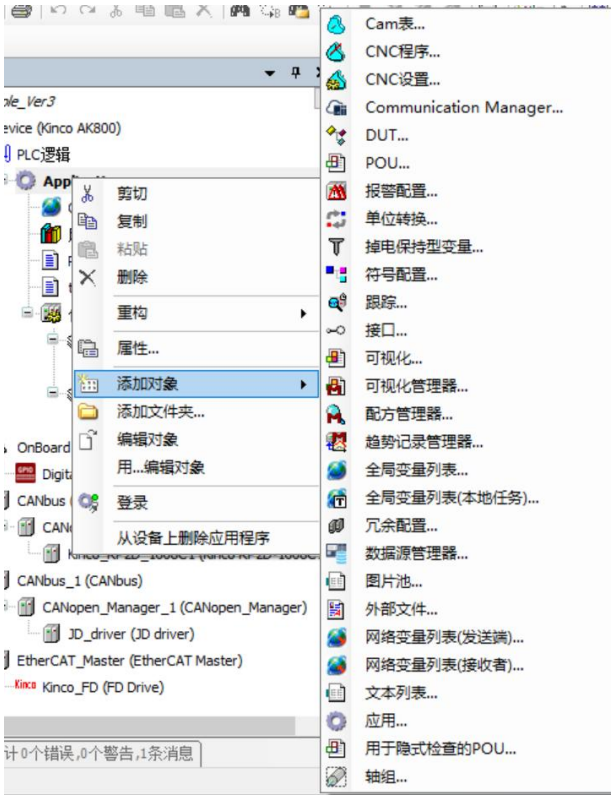


图 2.4-3：“POU”功能菜单

此外，还可以通过“工具”菜单下的“选项”子菜单对系统做一些设置，比如设置编辑器的字体、切换软件的语言、自动声明变量设置、编译设置等等。

第三章 快速入门

本章的主要内容是通过编写简单的程序，对CoDeSys的编程方式进行介绍，使初学者对CoDeSys的编程有一个初步的认识，了解CoDeSys的基本操作。假如您第一次使用CoDeSys，建议仔细阅读本章节。

本章示例程序的主要功能是在一定的时间间隔内，使控制器输出点不断地进行交替通断。在编程前，首先需要确定硬件配置。为简明起见，本示例在硬件组态上不配置任何扩展模块。

3.1 硬件连接

1.所需设备

- 一台已经安装 CoDeSys V3.5软件的PC（安装说明参考第一章），请确保 CoDeSys V3.5软件上已成功安装AK800系列软件包，具体请参照1.3：安装目标文件；
- 一台AK800可编程控制器；
- 一个24V直流电源；
- 标准超五类网线一根。

2.硬件接线

AK800系列控制器的硬件接线请参照《AK800系列可编程控制器硬件手册》。注意，在检查确认所有电缆连接无误后，再接通电源，以保证控制器可靠运行。

3.2 新建工程

概述：双击“CODESYS V3.5”图标，进入软件界面。然后选择菜单栏上的“文件”在弹出的目录下选择“新建工程”（或直接点击菜单栏上的“新建工程”按钮），在弹出的新建工程对话框中，默认选择类别“Project”和模板“Standard Project”，输入工程的名称，选择工程的保存路径后，点击“确认”，然后在弹出的“标准工程”对话框中，选择设备“Kinco AK800”，并为主程序编写选择一种编程语言，然后点击“确认”，等待工程自动创建完成。

新建工程的图例步骤演示为如下：

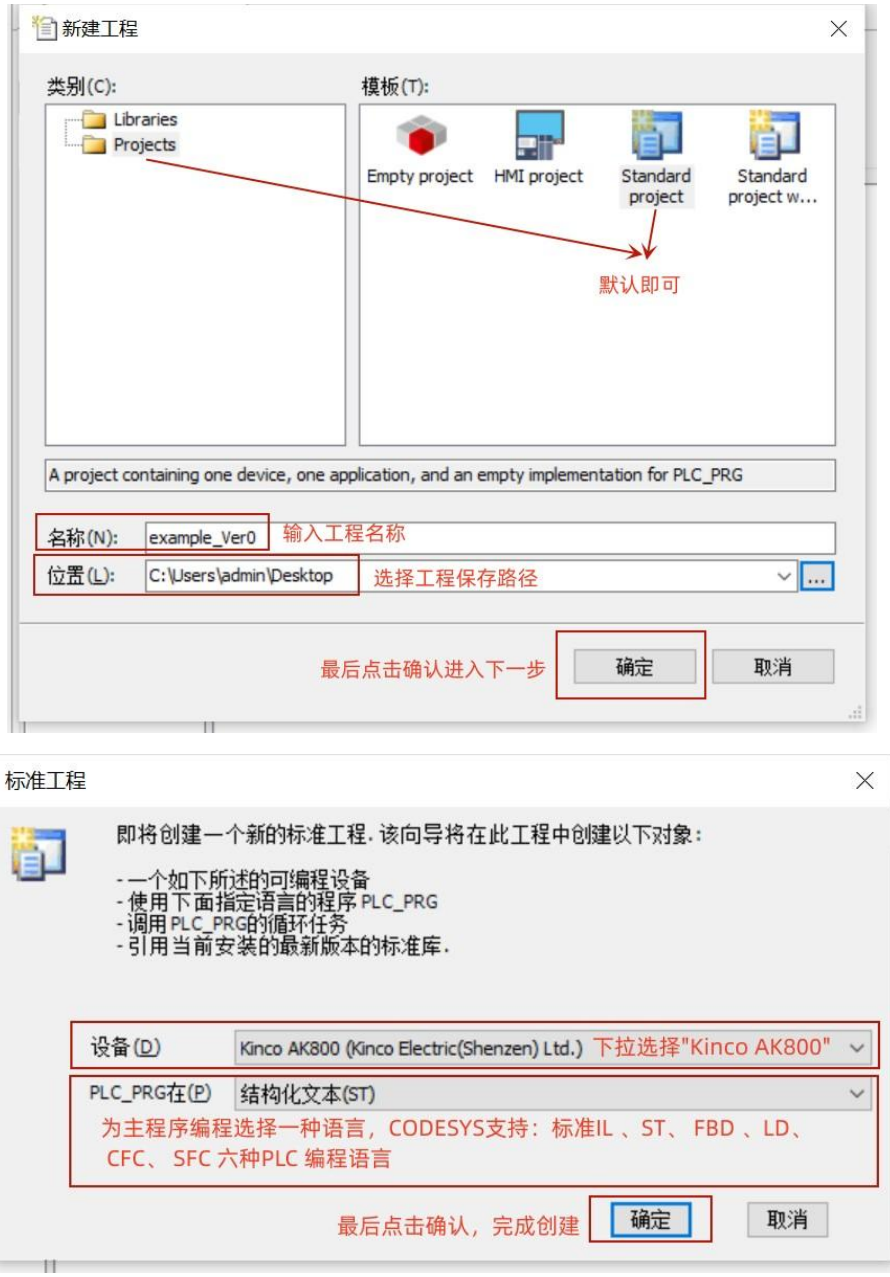


图 3.2-1：新建工程图例演示

如果目标平台设置弹出窗口下拉框中找不到“Kinco AK800”这个下拉选项，请确保CoDeSys V3.5软件上已成功安装AK800系列软件包，具体请参照1.3：安装目标文件；

3.3 编写程序

新建工程后，就可以在 PLC_PRG 中开始进行控制器程序的编写了。本例是一个简单的定时器应用程序，将以 ST 语言编写（各编程语言基础请参考第九章），本例程序目标是产生一个“1s 断、2s 通”的周期性脉冲信号，下面将分步骤实现该逻辑功能。

➢ Step1

根据目标要求可知，本例需要两个定时器，分别进行 1s 的“断”定时和 2s 的“通”定时，在 CODESYS 软件平台中，定时器在编程环境中是以包含在系统提供的库文件中的功能块形式体现的，光标选中界面的编程区，按 F2 调出“输入助手”，在“类别”中选择“功能块”选项，在右侧找到名为“Standard”的库文件，并在其目录下找到“Timer”选项中的接通延时定时器“TON”，点击“确定”，如下图所示：

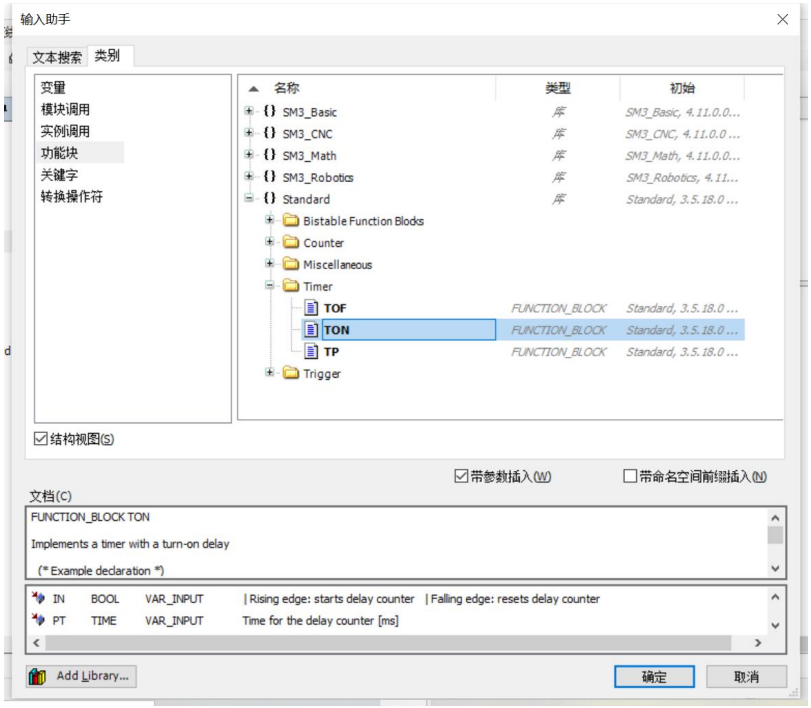


图 3.3.-1：利用输入助手调用“TON”功能块

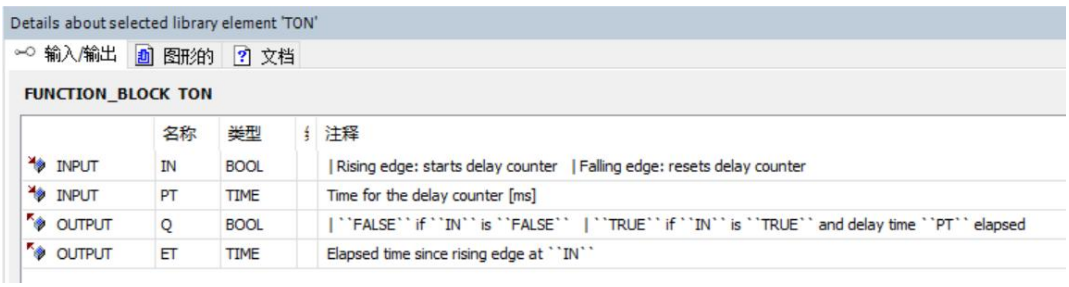


图 3.3.-2：“TON”功能块参数说明

➢ Step2

软件要求对功能块进行实例化，在弹出的“自动声明”窗口中，为该实例化定时器功能块命名，此处命名为“T1”，经过实例化后，“T1”则可以作为用于标识该特定 TON 指令的

一个标识符。对于程序中所有使用到的功能块，都需要用一个特定标识符来标识。关于这方面的详细信息，请参见 5.3.2 章节和 7.4.3 章节相关内容。此外，可以在注释栏根据需要为变量或实例化的功能块填写注释。

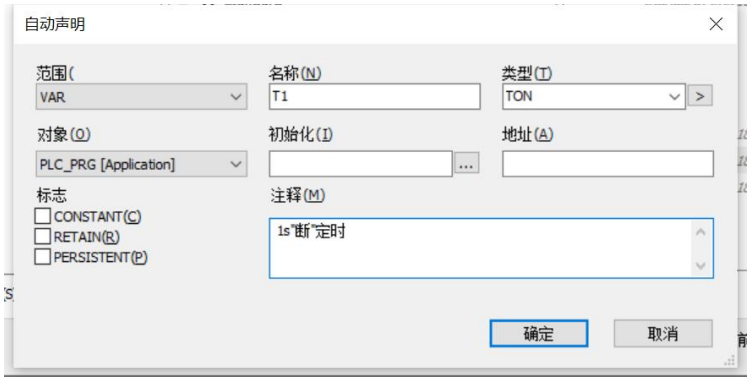


图 3.3.-3：实例化“T1”定时器功能块

➢ Step3

实例化成功之后，“T1”的定义将会自动出现在“变量定义区”。此后，还需要我们对“T1”各个参数进行设置（见图：“TON”功能块参数说明）。本例中，“T1”的触发变量为“Var1”，

定时时间“PT”设置为常量“1s”，实际计时时间“ET”由变量“ET1”输出，“Q”表示到达约定的定时时间后“T1”的接通状态输出标志，此处不定义接收变量，由“T1”本身的输出。

关于常量和变量，请参见 4.3 和 4.4 章节。“T1”的程序行如下图所示，“Var1”和“ET1”的定义可以通过自动声明变量进行（对于新增的系统未定义的变量，系统会自动要求对该变量进行声明，具体参考 4.4.3 变量定义章节），也可以通过在“变量定义区”内自行编辑定义。

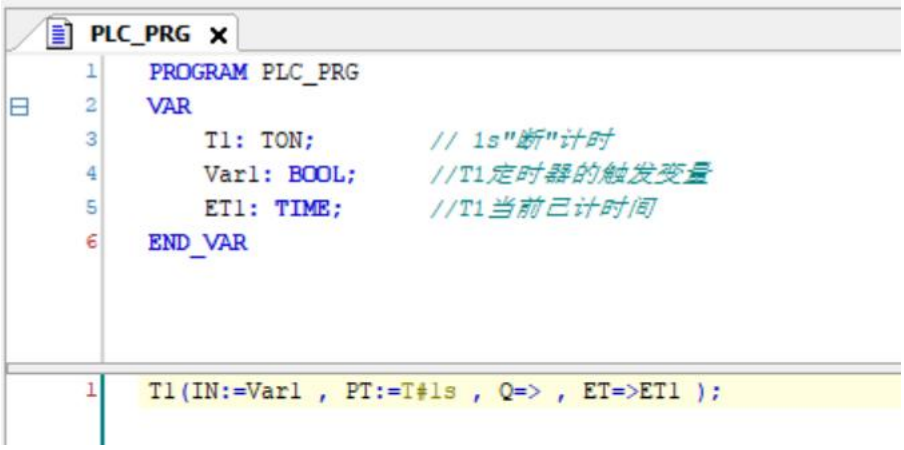


图 3.3-4：以“T1”为例编写的定时器程序

➢ Step4

参照“Step2”和“Step3”，再定义一个接通延时定时器“T2”，本例中“T2”定时器的触发变量为“T1”定时器定时输出接通标志，对于功能块的变量寻址（详情参考 5.3.2 章节），在其标识符（T1）后输入“.”会显示出来此实例化功能块下的所有参数，由<图

3.3-2：“TON”功能块参数说明>可知，定时器定时结束输出接通标志符号为“Q”。编写的程序实例如下<图 3.3-5：示例工程步骤 4 图例>所示：

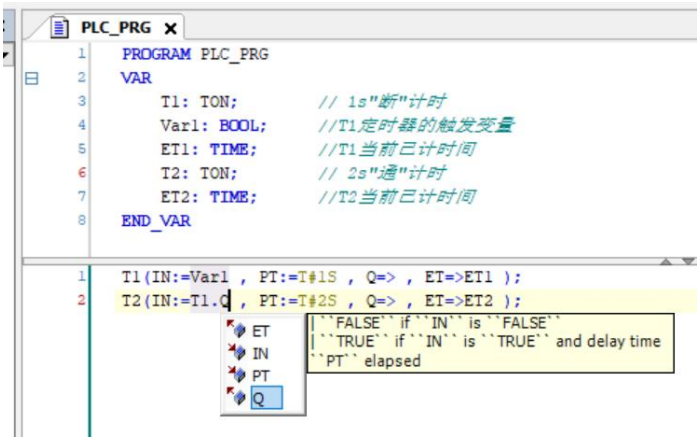


图 3.3-5：示例工程步骤 4 图例

- Step5

为使目标信号（此处采用一个 AK800 本体集成的第一个数字量输出点作为目标输出，在程序中该输出点的寻址地址为：%QX0.0，寻址方式详情参考第四章）周期性通断运行，还需把“T2”定时器的定时接通输出端互锁到“T1”定时器的触发端，这里还需要与触发“T1”的变量“Var1”建立一个“逻辑与（AND）”和“逻辑非（NOT）”的关系，逻辑关系运算符是一个标准 IEC 运算符，根据最初的逻辑要求，当“T1”接通时，“T2”无输出，目标信号%QX0.0 为“False”，当“T2”接通时，“T1”计时复位，目标信号%QX0.0 为“True”，即目标信号的输出“通断”与“T1”的输出有无呈正相关，故根据此逻辑，可以把目标信号的输出绑定到“T1”的输出上，故最终程序如下：

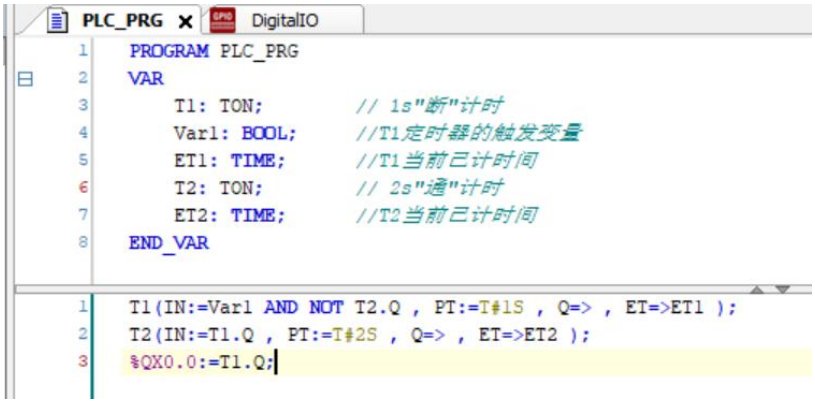


图 3.3-6：定时器示例最终程序

至此，我们就完成了一个完成的控制器程序的编写。需要注意的是，各种编程语言都有其既定的规则和形式，例如在本例的 ST 语言中，每一个语句必须以“；” 结束，赋值语句格式为“:=”且必须以英文字符（半角）下输入。

3.4 编译程序

程序编写完毕，应该对其进行编译。选择“编译”菜单下的“生成代码”或使用快捷键“F11（+Fn）”或快捷工具栏中单击编译图标。

编译完成后会在消息窗口会显示如图所示的编译信息。通过信息窗口可以帮助我们检查程序的语法错误。如果显示“0 个错误”则表示程序通过了编译。

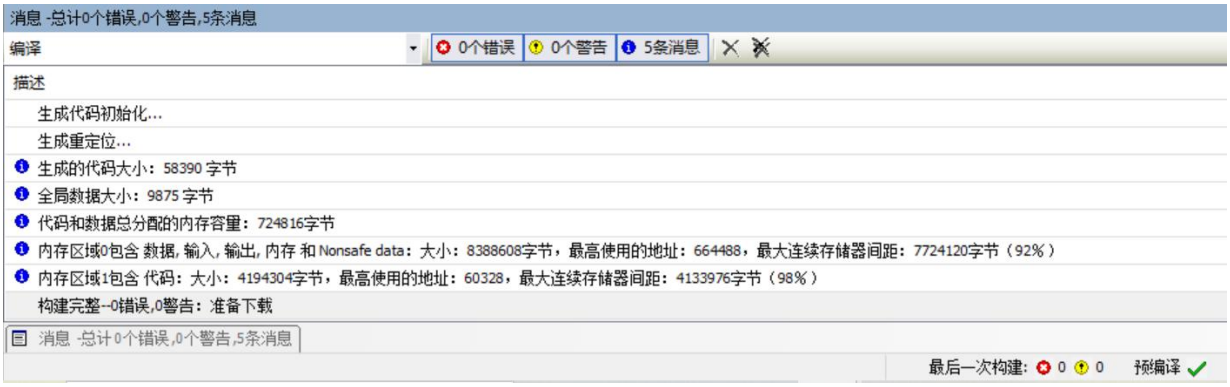


图 3.4-1：定时器示例程序编译信息

关于编译的更多详情，请参见 8.1 章节。编译结束后，双击“设备区”的“Device”，检查 PC 与控制器的通讯是否正常（执行“扫描”操作），执行菜单栏上的“在线”下的“登录”命令进入调试状态，并执行“调试”菜单下的“启动”命令启动程序运行。

在首次登录设备下载程序时，会弹出以下警告对话框，点击“确定”继续下载并登录即可。

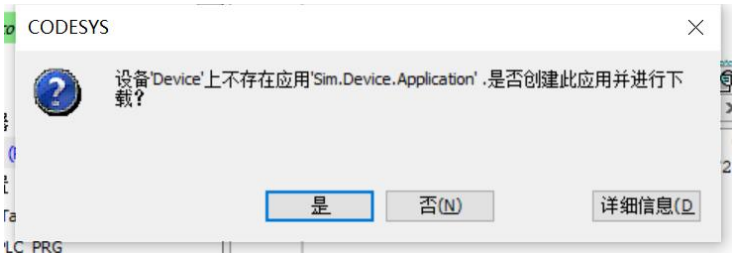


图 3.4-2：下载警告对话框

调试分为联机调试和仿真调试两种方式。在后面的介绍中，将按照联机调试和仿真模式两种方式分别介绍程序的调试概况。

3.5 仿真调试

Codesys 软件支持仿真调试，如果没有实际连接控制器，可在本地计算机模拟运行用户程序，称为仿真模式。在菜单栏中打开“在线”下拉菜单中选择“仿真模式”（该项被选中后会出现“√”，同时软件界面的状态显示区也会显示为“仿真”模式），在此模式下进行“登录”操作时，则可以对程序进行仿真调试。

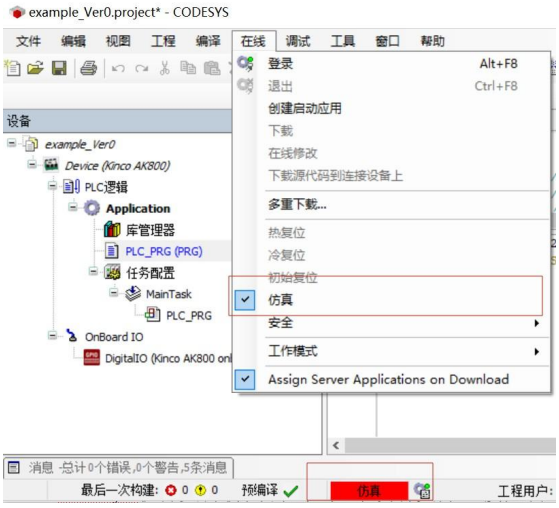


图 3.5-1：仿真调试操作及状态显示图例

如下<图 3.5-1：仿真模式下登录界面图例>所示，成功登录后的软件界面如下图所示：设备树窗口各硬件介质前方出现红三角形状，“Application”和状态显示区的状态均为“停止”状态，表示登陆成功，但程序未运行。此时再点击菜单栏“调试”目录下的“启动”（或使用或快捷键 F5（+Fn））即可以运行程序。

当程序登录并成功运行时，编程窗口和变量声明窗口会出现各变量的当前值。

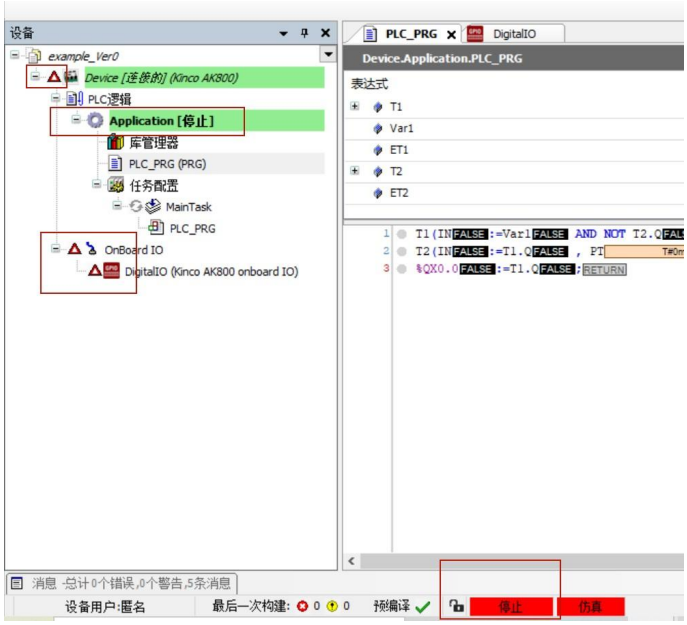


图 3.5-2：未启动运行状态下的在线模拟界面

在程序编辑窗口双击“T1”的触发变量“Var1”或者在变量声明窗口中找到变量“Var1”所在行的“准备值”，点击使之为“TRUE”，再使用组合快捷键 Ctrl+F(+Fn)或执行“调试”菜单下的“写入值”命令来将变量“Var1”置位。之后可看到程序中%QX0.0 按设计要求 1S 断，2S 通的状态在交替闪烁，同时也可看到其他各变量的运行情况，如定时器的当前值等。关于联机调试和仿真调试的详细操作，请参见 8.4 章节。

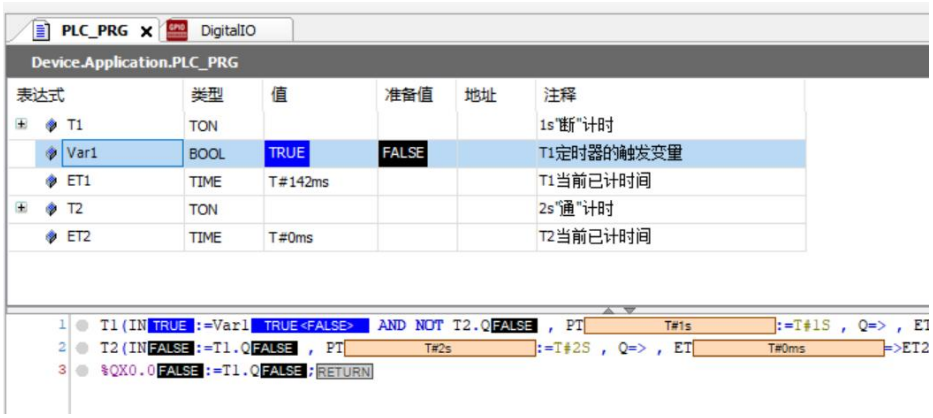


图 3.5-3：定时器示例模拟仿真调试运行中

3.6 联机调试

程序下载到控制器中称为联机调试。编译通过的目标文件在进行下载时，会将全部目标文件下载到控制器中，同时将控制器复位，所有变量返回到初始状态。执行“在线”菜单下的“登录”命令（注意当前是否处于仿真模式，如处于仿真模式则需要先退出仿真模式），建立本地计算机与控制器的连接（双击“Device”，执行“扫描操作”并连接控制器），下载时也会出现和仿真模拟一样的系统下载警告信息，参照仿真模拟步骤操作即可。

开始联机调试前必须设置好通讯参数。首先设置好电脑的 IP 与控制器的 IP 必须在同一个网段，Kinco AK800 系列控制器默认 IP 地址为 192.160.0.250，设置好电脑 IP 后，如果已经正确连接网线到控制器端口，控制器上电后，连接的网口绿色指示灯常亮，黄色指示灯闪烁则代表连接成功。联机之前，请确认电脑右下角的 Gateway  处于打开状态；

AK800 控制器默认以“EtherNet”太网通信端口进行程序的下载与调试。其以太网通信端口默认 IP 为：192.168.0.250。在进行联机调试前，应设置电脑的 IP 使之控制器的 IP 处于同一个网段（0 网段），操作如下：

“控制面板”>>“网络与 Internet”>>“网络和共享中心”>>“更改适配器配置”>>右击目标以太网选择“属性”>>双击“Internet 协议版本 4”>>将 IP 地址网段改为 0 网段>>“确认”保存即可。

另外，如果在 CoDeSys 中扫描不到 AK800 控制器，可能原因和对策如下：

- CoDeSys 网关没有启动；

解决方法：请检查启动网关后，再进行扫描。

- PC 的 IP 地址与 AK800 的 IP 地址不在同一个网段；

解决方法：查看 PC 的地址，是否在 192.168.1.xxx 网段，若不是，请先记下 PC 的 IP 设置，并修改成与控制器同样的网段，再进行扫描。

第四章 存储区与变量

在编程之前，需要了解控制器是如何管理数据的。本章为您讲述控制器的存储区分配、数据寻址以及变量使用等概念，为编程打下良好的基础。

控制器的数据存储区分为全局区、输入区（I 区）、输出区（Q 区）、M 区、掉电保持区共五类，每一类都有自己的特点和使用规则，4.1 章节讲述了各存储区的特点和使用方法。

4.2 章节讲述了地址的寻址方法和存储规则。在控制器的数据存储区中，输入区、输出区和 M 区属于寻址方式调用的数据区。

和高级语言类似，AK800 控制器也有常量和变量的概念。所谓的常量就是数值不变的数。4.3 章节详细讲述了常量的分类和使用方法。

变量是 IEC61131-3 标准提出的概念。在程序运行时，其数值会改变的量就是变量。变量用于初始化、存储和处理用户数据。每个变量都有其固定的数据类型。变量的存储位置可由用户指定为 I 区、Q 区或 M 区的地址，亦可不指定地址，由系统自行分配。

4.4 章节详细讲述变量的类型和使用方法。

控制器在数据管理上功能非常强大。4.5 章节和 4.6 章节为您讲述了如何处理数组以及如何自定义数据类型。

4.1 存储区分配及范围

AK800 控制器的存储区，CPU 的内存被划分为不同的存储区，每一部分存储区都有各自的特点和使用规则。存储区包括以下几大类：

全局存储区（G 区）

全局映像区。全局变量的存储区，在全局存储区定义的变量可以被工程中的所有程序或功能块调用，变量定义时，假如没有定义为全局变量，则该变量默认的即是局部变量，只能在其有定义的程序或功能块中才能被调用。关于全局变量的声明请参考 4.4.3 章节。

输入存储区（I 区）

输入映像区。CPU 及扩展模块的数字量输入占用输入存储区地址，模拟量输入同样也占用输入存储区地址。另外，一些特殊功能，诸如 CANOp“EN”通讯，也占用输入存储区的地址。输入存储区通过寻址方式访问，可以按位、字节、字、双字访问。具体访问方法请参见 4.2 章节。

输入存储区是只读的，并且不能掉电保持。在仿真模拟时，输入存储区的地址可以被输入，也可以被强制。但是在联机调试时，只能被强制。输入和强制是 CoDeSys 在调试时改变数据的方式，请参见 8.4.9 和 8.4.10 章节。

输出存储区（Q 区）

输出映像区。CPU 及扩展模块的数字量输出占用输出存储区地址，模拟量输出同样也占

用输出存储区地址。另外，一些特殊功能，诸如 CANOp“EN”通讯，也占用输出存储区的地址。

输出存储区通过寻址方式访问，可以按位、字节、字、双字访问。具体访问方法请参见 4.2 章节。

输出存储区的数据是可读写的，并且不能掉电保持。在仿真模拟或者联机调试时，该数据区地址均可以被输入或强制。

M 存储区

M 存储区是 控制器 的中间寄存器区，用于存储和管理中间过程产生的数据或状态。无论是位数据，还是字数据，均可以在 M 存储区实现。

M 存储区通过寻址方式访问，可以按位、字节、字、双字访问。具体访问方法请参见 4.2 章节。

M 存储区的数据地址与输出存储区的性质一样，可读写，也可以被输入或强制。M 存储区的地址中，均不具有掉电保持功能。

另外，要特别注意，M 存储区的地址是与 Modbus 通讯的 4X 地址对应的，即 4X1 对应 MW0。

R 存储区

R 存储区属于掉电保持区，其调用方式与 M 区一致，也是通过变量的方式访问，但无法指定其实际地址。R 存储区变量可以读写，可以被输入和强制。

变量定义时，假如没有选择保持功能，或者直接在局部变量中定义，则该变量存储在自由存储区，也即堆中，若选择了保持功能或直接在保持型变量中定义，则该变量存储于 R 区，具有掉电保持功能。关于如何定义掉电保持区变量，详细说明请参见 4.4.4 章节。

各存储区在控制器内部是有最大范围的，详情如表 4.1-1。

表 4.1-1：控制器各存储区的范围

	元件	名称	个数
元件软	I	输入继电器	32KWords
	Q	输出继电器	32KWords
	M	辅助继电器	256KWords

4.2 地址寻址方式

4.2.1 地址存储映射关系

表 4.2-1a：大小前缀定义

前缀符号	定义	约定数据类型
X	位（bit）	BOOL
B	字节（BYTE）	BYTE
W	字（WORD）	WORD
D	双字（DWORD）	DWORD
L	长字（LWORD）	LWORD
*	未特定位置的内部变量，系统自动分配。	

控制器的 I 区、Q 区和 M 区是按地址寻址方式访问的，这些存储区都有唯一的、明确的地址。用户可以通过地址寻址方式，即直接使用该存储区地址的方式，来读取和设置该存储区的值。在介绍访问规则之前，需要先了解这些存储区的存储格式。I 区、Q 区和 M 区的存储格式是一致的，所以这里以 I 区的 %ID48 为例说明，如下表 4.2-1 所示。

表 4.2-1b：内存映射表

%IX	96.0 - 96.7	96.8 - 192.15	97.0 - 97.7	97.8 - 97.15
%IB	192	193	194	195
%IW	96		97	
%ID	48			

如上图控制器所有直接寻址的存储区，是按字节存储的。每 8 个位（BIT）组成一个字节（BYTE）。每 2 个字节（BYTE）组成一个字（WORD），每两个字（WORD）组成一个双字（DWORD）。该地址内存映射表中，字地址%IW96 和%IW97 两个字组合后对应%ID48，因为 48* 2（字节）后的字节首地址为 96。同样的道理，字节地址%IB192、%IB 193、%IB 194 和%IB 195 这四个字节变量组合后对应%ID48， 因为 48 * 4（字节） 后对应的字节首地址正好为 192。双字、单字、字节地址之间是 2n，2n+1 的关系。

再如：%MX2.7 表示%MB2 的第 8 个位。%MW12 表示有%MB24 和%MB25 两个字节组成的字。这些存储区的地址，可以按位、字节、字或双字来访问，数据类型为 BOOL、BYTE、WORD 和 DWORD。要了解这些数据类型的范围，请查看 4.4.2 章节。假如需要使用 INT 或者 REAL 等其他数据类型，需要使用变量方式访问，具体内容也请参见 4.4 章节。

要注意，按不同数据类型访问，数据存储区可能是重叠的。诸如%MW0 的数值为 3，则%MB0 的值也为 3，%MX0.0 和 %MX0.1 的值也为 TRUE。又或者在程序中强制%MX0.0 为 TRUE，因为%MX0.0 是%MB0、%MW0 和%MD0 的第一个位，则同时，这些存储地址的

值也被强制为 1。对于 I 区和 Q 区，也是同样的存储方式。I 区和 Q 区的地址和实际 DI>DO 点如何对应，请参见 7.3 章节。

注意：高低字节的顺序，例如某个 WORD 型变量位于 MW601，假定该变量的值为十六进制的 1234，则%MB1202 存储的值为十六进制的 34，%MB1203 存储的值为十六进制的 12。控制器采用的是小端模式。所谓的小端模式，低位字节排放在内存的低地址端，高位字节排放在内存的高地址端。当与外设设备进行通讯连接或者在 PC 端进行模拟器，如果这些设备是大端模式存储，通过通讯从控制器中读取的 DWORD 类型的双字数据在大端模式设备中显示时可能需要高低字调换，单字则需要高低字节调换。注意位的存储方式，为 8 位存储方式。每个字节包含的 8 个位依次为 BIT0-BIT7，相邻的字节占用的 8 个位也是一样依次为 BIT0-BIT7，而不是 BIT8-BIT15 之类的。如%IB0 含%IX0.0-%IX0.7，%IB1 包含%IX1.0-%IX1.7，为了更好的帮助理解存储方式，下图给出一个实际运行的范例。

```
1     //16进制显示
2   %MD300 16#12345678 :=16#12345678;   // MD300=MW600--MW601
3   %MW600 16#5678 :=%MW600 16#5678 ;   // MW600=MB1200--MB1201
4   %MW601 16#1234 :=%MW601 16#1234 ;   // MW601=MB1202--MB1203
5   %MB1200 16#78 :=%MB1200 16#78 ;     // MB1200=MX1200.0--MX1200.7
6   %MB1201 16#56 :=%MB1201 16#56 ;     // MB1201=MX1201.0--MX1201.7
7   %MB1202 16#34 :=%MB1202 16#34 ;     // MB1202=MX1202.0--MX1202.7
8   %MB1203 16#12 :=%MB1203 16#12 ;     // MB1203=MX1203.0--MX1203.7
9   %MX1200.0 FALSE :=%MX1200.0 FALSE ;//---
10  %MX1200.1 FALSE :=%MX1200.1 FALSE ;//    |
11  %MX1200.2 FALSE :=%MX1200.2 FALSE ;//    |
12  %MX1200.3 TRUE :=%MX1200.3 TRUE ;//    |
13  %MX1200.4 TRUE :=%MX1200.4 TRUE ;//    | MB1200
14  %MX1200.5 TRUE :=%MX1200.5 TRUE ;//    |
15  %MX1200.6 TRUE :=%MX1200.6 TRUE ;//    |
16  %MX1200.7 FALSE :=%MX1200.7 FALSE ;//---
17
18  %MX1201.0 FALSE :=%MX1201.0 FALSE ;//---
19  %MX1201.1 TRUE :=%MX1201.1 TRUE ;//    |
20  %MX1201.2 TRUE :=%MX1201.2 TRUE ;//    |
21  %MX1201.3 FALSE :=%MX1201.3 FALSE ;//    |
22  %MX1201.4 TRUE :=%MX1201.4 TRUE ;//    | MB1201
23  %MX1201.5 FALSE :=%MX1201.5 FALSE ;//    |
24  %MX1201.6 TRUE :=%MX1201.6 TRUE ;//    |
25  %MX1201.7 FALSE :=%MX1201.7 FALSE ;//---
```

图 4.2-1：存储区的存储格式示例

4.2.2 地址访问格式

按 IEC61131-3 标准，所有的直接地址都从“%”开始。以 M 区为例，如下表所示：

表 4.2-2：程序中的地址访问格式

位寻址	格式	%MXm.n
	描述	X 表示是按位寻址； m 表示在 M 存储区中的字节编号； n 表示位于该字节的第几位，范围为 0~7
	数据类型	BOOL
	示例	%MX0.3、%MX100.0、%MX3212.7
字节寻址	格式	%MBm
	描述	B 表示按字节寻址 m 表示在 M 存储区中的字节编号
	数据类型	BYTE
	示例	%MB103、%MB2000
字寻址	格式	%MWm
	描述	W 表示按字寻址
	数据类型	WORD
	示例	%MW150、%MW3000
双字寻址	格式	%MDm
	描述	D 表示按双字寻址
	数据类型	DWORD
	示例	%MD300、%MD432

4.3 常量

在编程的时候，可能经常要用到一些数值不变的参数，诸如定时器的时间、换算的比例参数等，这些数值不变的参数称为常量。CoDeSys 支持多种数据类型的常量，常见的常量：布尔型、时间型、数字型等，如表所示。

表 4.3-1：各类型常量的表示方法

类型	表示方法	
布尔型	描述	布尔常量只有两个：逻辑值 TRUE 和 FALSE
	示例	TRUE
整型	描述	数字常量的数值可以是二进制、十进制、十六进制。如果整数值不是十进制值，可以用“进制”加符号“#”放在整数值前面来表示。十进制的 0 至 15 在十六进制中表示为 A 至 F。
	示例	14 (*十进制数 14*) 2#1001_0011 (*二进制数 1001_0011*) 16#AE (*十六进制数 AE*)
实型	描述	实数常量用十进制小数和指数来表示，遵循标准的科学计数法格式。实数常量的数据类型是 REAL 和 LREAL。

	示例	7.4 (*实数 7.4*) 1.64e+009 (*实数 1.64e+009*)
时间型	描述	时间常量一般用来操作时间，由“T#”（或“t#”）加上“时间值”构成，时间值的单位包括天（d）、小时（h）、分（m）、秒（s）和毫秒（ms）。注意，它们的正确顺序为 d、h、m、s、ms。
	示例	T#18ms (*18 毫秒的一个时间常量*) T#100s12ms (*100 秒 12 毫秒的一个时间常量，高单位允许超限*） t#12h34m15s (*12 小时 34 分 15 秒的一个常量*) 下面是错误的时间常量： t#5m68s (*低单位不允许超限*) 15ms (*没有 T# *) t#4ms13d (*顺序错误*)
时刻型	描述	时刻常量用于存储当前时刻，由“TOD#”（“tod#”、“TIME_OF_DAY#”或“time_of_day#”）加上“时刻值”构成。时刻值的格式为：小时：分钟：秒（可以用实数形式输入秒）。
	示例	TOD#00：00：00 (*0 点 0 时 0 分*) TIME_OF_DAY#15：36：30.123 (*15 点 36 分 30.123 秒*)
日期型	描述	日期常量由“D#”(“d#”、“DATE#”或“date#”)加上“日期值”构成。
	示例	DATE#2005-05-06 (*日期常量 2005 年 5 月 6 日*) dt#1980-09-22 (*日期常量 1980 年 9 月 22 日*)
日期时刻型	描述	日期常量和时刻常量合并起来称为日期时刻常量，由 “DT#”（ “dt#”、“DATE_AND_TIME#”或“date_and_time#”）加上“日期时刻值”构成。
	示例	DT#1980-09-22-15：45：18 (*时刻日期常量为 1980 年 9 月 22 日 15 点 45 分 18 秒*) date_and_time#2001-03-09-00：00：00 (*时刻日期常量为 2001 年 3 月 9 日 0 点 0 分 0 秒*)
字符串	描述	字符串常量在两个单引号之间，可以包含空格和特殊字符。
	示例	'Abby and Craig' (*字符串 Abby and Craig *)

常变量：

CoDeSys 中还可以定义一种“常变量”类型，常变量是常量和变量的部分结合体，和符号常量类似，值都不能改变，但又具备了变量的变量类型、变量名和变量值的属性。简单的说常变量就是其值不能改变的变量。CoDeSys 中常变量的定义以“CONSTANT”为标识符。

4.4 变量

所谓变量，就是用字母、数字和下划线组成的一个标识符。按照数据类型的不同，变量可以分为标准类型和用户自定义类型。

其中标准类型包括布尔型（BOOL）、整型（INT）、实型（REAL 或 LREAL）、字符串型（STRING）以及时间型（TIME）等。自定义类型包括结构体（STRUCT）和枚举(“ENUM)。按照使用范围的不同，变量可以分为全局变量和局部变量。局部变量只在整个工程的一部分程序中有效，其它程序不能引用。全局变量则可以被整个工程的任意程序引用，在整个工程中均有效。

按照属性的不同，变量分为中间变量、输入型变量、输出型变量、输入输出型变量等。

按照能否掉电保护，变量分为保持型变量和非保持型变量。

4.4.1 变量命名规则

变量命名必须遵循如下的规则：

- 必须以一个字母或者单一的下划线开始，随后是一定数量的字母、数字或下划线。
- 字母与大小写无关，ABC 和 abc 被认为是同一个变量。
- 关键字不能用于变量名。CoDeSys 定义了一些关键字，关键字是标准的标识符,其作用和命名已经在系统中定义。CodeSys 关键字如下所列。

CoDeSys 关键字列表：

ABS、ACOS、ADD、ADR、ADRINST、AND、ANDN、ARRAY、ASIN、AT、ATAN BITADR、BOOL、BY、BYTE、CAL 、CALC、CALCN、CASE、CONSTANT、COS、DATE、DINT、DIV、DO 、DT、DWORD、ELSE、ELSIF、END_CASE、END_FOR、END_FUNCTION、END_FUNCTION_BLOCK 、END_IF 、END_PROGRAM、END_REPEAT、END_STRUCT、END_TYPE、END_VAR、END_WHILE、EQ、EXIT、EXP、EXPT、FALSE、FOR、FUNCTION、FUNCTION_BLOCK、GE、GT、IF、INDEXOF、NI、INT、JMP、JMPC、JMPCN、LD、LDN、LE、LINT、LN、LOG、LREAL、LT、LWORD、MAX、MIN、MOD、MOVE、MUL、MUX、NE、NOT、OF、OR、ORN、PERSISTENT、POINTER、PROGRAM、READ_ONLY、READ_WRITE、REAL、REPEAT、RET、RETAIN、RETC、RETCN、RETURN、ROL、ROR、S、R、SEL、SHL、SHR、SIN、SINT、SIZEOF、SQRT、ST、STN、STRING、STRUCT、SUB、TAN、THEN、TIME、TO、TOD 、TRUE、

TRUNC、TYPE、UDINT、UINT、ULINT、UNTIL、USINT、VAR、VAR_ACCESS、VAR_CONFIG、VAR_CONSTANT、VAR_EXTERNAL、VAR_GLOBAL、VAR_IN_OUT、“VAR_INPUT”、“VAR_OUTPUT”、WHILE、WORD、WSTRING、XOR、XORN

4.4.2 变量数据类型

CoDeSys 软件支持的变量数据类型包括标准类型和用户自定义类型。关于自定义数据类型，请参见 4.6 章节。

CoDesys 支持的标准数据类型及范围，如下表 4.4-1 所示。

表 4.4-1：CoDesys 支持的标准数据类型及范围

类型	类型名称	数据下限	数据上限	存储空间	描述
BOOL	布尔型	0	1	1bit	
BYTE	字节型	0	255	8 Bit	
WORD	字型	0	65535	16 Bit	
DWORD	双字型	0	4294967295	32Bit	
SINT	短整型	-128	127	8 Bit	
USINT	无符号短整型	0	255	8 Bit	
INT	整型	-32768	32767	16 Bit	
UINT	无符号整型	0	65535	16 Bit	
DINT	长整型	-2147483648	2147483647	32 Bit	
UDINT	无符号长整型	0	4294967295	32 Bit	
REAL	实数型	1.401e-45	3.402823E+38	32Bit	单精度浮点数
LREAL	实数型	2.2250738585072014e-308	1.7976931348623158e+308	64Bit	双精度浮点数
TIME	时间型	最小时基毫秒	32Bit		示例：Time1:TIME:= t#3s;
LTIME	时间性	最小时基纳秒	64BIT		Time1:LTIME:= t#3us2ns;
TOD	时刻型	\			示例：Tod1:TOD:=TOD#00:00:00;
DATE	日期型	\			示例：Date1:DATE:=D#2008-8-8;
DT	日期时刻型	\			示例：DT1:DT:=dt#2008-08-08-20:08:08;
STRING	字符串型	只支持ASCII字符（不支持中文字符），默认最大长度80字符，字符串			示例：Str:STRING(35):=“HI” ;

		函数最大255字符			
WSTRING	字符串型	只支持UNICODE字符（支持中文字符），默认最大长度80字符，可以声明最大字符长度			
ARRAY	数组	\			示例：Arr1:ARRAY[1..5]OF BYTE:=[1,2,3,4,5];

4.4.3 变量定义

在使用变量之前，必须先对变量进行定义。CoDeSys 针对变量不同的功能，规定了不同的变量类型。在定义变量时，不单要定义数据类型，还要定义变量类型。在 CoDeSys 中，变量可以被定义为很多类型，如：全局变量、局部变量、输入变量、输出变量等。具体变量类型如下表所示。

表 4.4-2：变量类型

变量类型	数据类型
“VAR”	局部变量，仅在该程序中使用。在其余程序中可以定义相同名称的变量，被认为是两个变量。
“VAR_INPUT”	输入变量，当调用程序时，输入变量用于实现调用程序时的参数传递。在调用程序时，可以将参数通过输入变量传递至子程序或其余 POU中，具体参见 5.3 章节。
“VAR_OUTPUT”	输出变量，当调用程序时，输出变量用于实现调用程序时的参数传递。在调用程序时，可以将参数通过输出变量传递至调用该 POU的程序中，具体参见 5.3 章节。
“VAR_IN_OUTPUT”	输入/输出变量，“VAR_INPUT” 和 “VAR_OUTPUT” 变量的组合。同样用于参数传递。
“VAR_GLOBAL”	全局变量，若该变量定义为全局变量，则在任何程序中均可使用该变量。同时，不能再定义名称相同的变量。

“VAR”、“VAR_INPUT”、“VAR_OUTPUT”、“VAR_IN_OUTPUT”、“VAR_GLOBAL”是用于标识变量类型的关键词。定义时根据可需要对类别进行选择，变量的声明有两种方式：

自动声明和手动声明。

1.自动声明变量

系统支持变量自动定义功能。当程序中出现一个新变量时，系统会自动弹出对话框（也可按 shift+F2 自动声明），要求进行变量定义，其中类别、名字和类型是必须的，如下图所示。

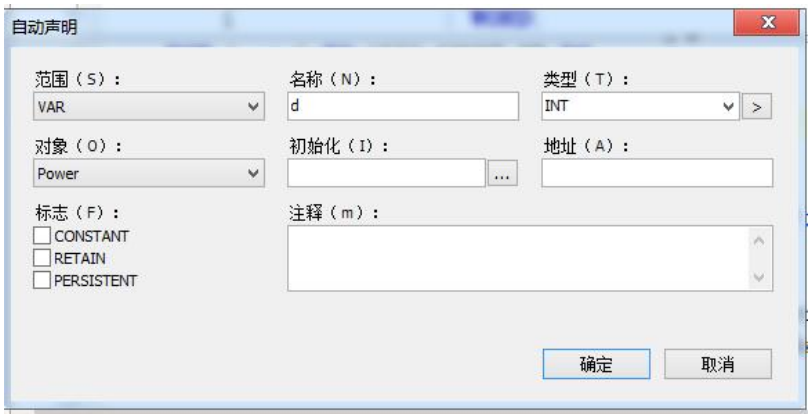


图 4.4-1：自动声明弹窗

自动定义变量对话框的各项含义，如下所述：

- 范围：类型选择。各类型区别请参见表 4-4-3，如：希望定义的变量在所有的 POU 中都能使用，则定义为全局变量，选择类型 “VAR_GLOBAL”。
- 名字：声明变量的名称，即标识符。关于变量命名的规则，请参见 4.4.1 章节。
- 类型：数据类型选择。可以直接在输入框中输入，也可以点击旁边按钮，在弹出的对话框中选择数据类型。各类数据类型请参见 4.4.2 章节。
- 对象：把变量归属于哪一组变量。
- 初始值：变量的初始值。这里可以填入一个与变量数据类型对应的常量，完成变量的初始化。
- 地址：指定变量的地址。这里可以指定也可以不指定，不是必须的，如果不指定地址，则变量存储在自由存储区，地址由系统随机分配。
- 注释：变量的含义。
- 标志：属性为常量（constant）或者保持型变量（retain）或者持久性变量（persistent）
- 在自动定义变量时，需要注意以下几点：
- 变量可以被指定一个地址，地址的格式与 4.2 章节所讲述的一致。当变量指定地址时,变量存储于该地址所指定的数据区，如图 4-4-1 所示的变量定义，则该变量 temp 存储于 M 存储区，与%MW100 占用同一存储区地址。在程序中，采用直接寻址方式改变%MW100 的值，则该变量 temp 相应改变。变量定义时，也可以不指定地址，则该变量存储于自由存储区。
- 在定义时，可以设置变量的初始值，初始值是一个常量，其类型应与变量的类型一致。诸如定义一时间类型变量，则初始值应是一个时间常量，例如 t#5s。定义初始值后，当控制

器 在上电瞬间，变量被赋值为初始值。

- 若定义的变量是全局变量，则会显示在资源选项中的全局变量中，而不是在变量声明区。
- 变量自动定义时，在自动定义对话框右下角有两个选项：常量和保持。当选择常量，则将该变量作为一个常量，程序中无法再改变其数值。当选择保持型或者持久型变量时，表示将该变量设置为具有掉电保持功能，该变量存储在R 存储区。
- 新建变量时，系统可以自动定义。但当变量被删除时，定义语句不会自动删除，继续保留在编辑器中，因此要注意变量不能定义重复。

2.手动定义变量

所谓的手动定义变量，就是不通过自动定义对话框进行定义，而是手动在变量声明区按变量声明的格式和规定添加变量。

变量声明的一般格式：

<标识符> {AT<地址>} : <数据类型> {:= <初始值>};

其中在{ }中的部分是可选的。

定义不同类型的变量，需要在不同的位置进行定义。诸如：定义局部变量，需要在“VAR”和“END_VAR” 之间定义，而定义输入变量，需要在“VAR INPUT” 和“END_VAR” 之间定义。变量声明区也可以定义为表格形式。在“工具”>“选项”>“声明编辑器”对话框里选中以表格形式声明项。

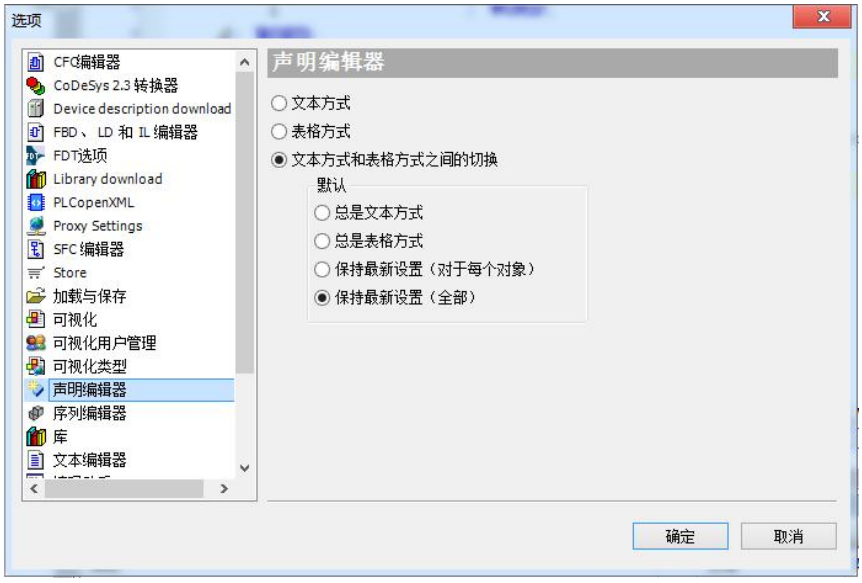


图 4.4-2：从工具菜单下的“选项”中打开“声明编辑器”

或在程序编辑区右侧菜单“表格视图”（下图右侧红框内），声明编辑器会显示成表格的形式，如图所示。

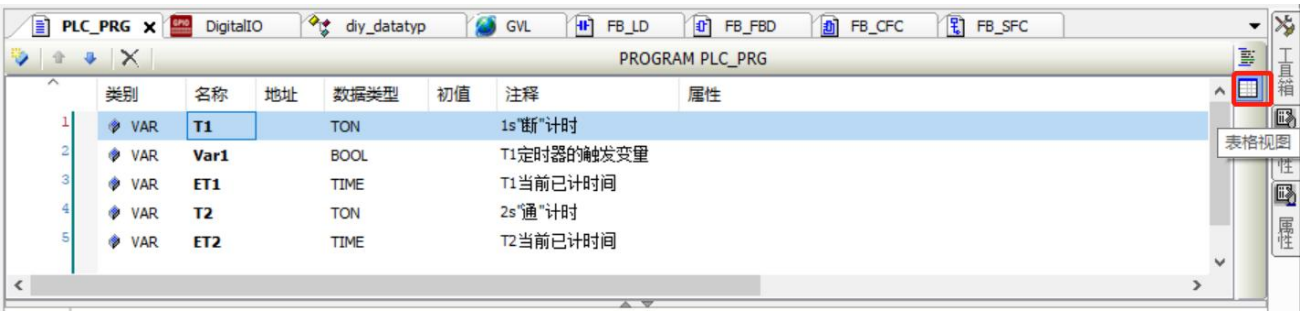


图 4.4-3：变量表的表格视图

选择右侧的文本形式，可切换至常规形式。表格声明变量以简洁的填表方式代了常规的语句声明方式，建议初学者使用自动变量定义或表格定义。

3.变量调用和地址调用方式的区别

当采用“变量+地址”方式定义变量，与直接地址调用类似，但是两者还是有区别的。直接地址调用的数据类型可为BOOL、BYTE、WORD、DWORD 等类型，而用“变量+地址”的方式调用，可定义的数据类型比直接地址调用方式多。

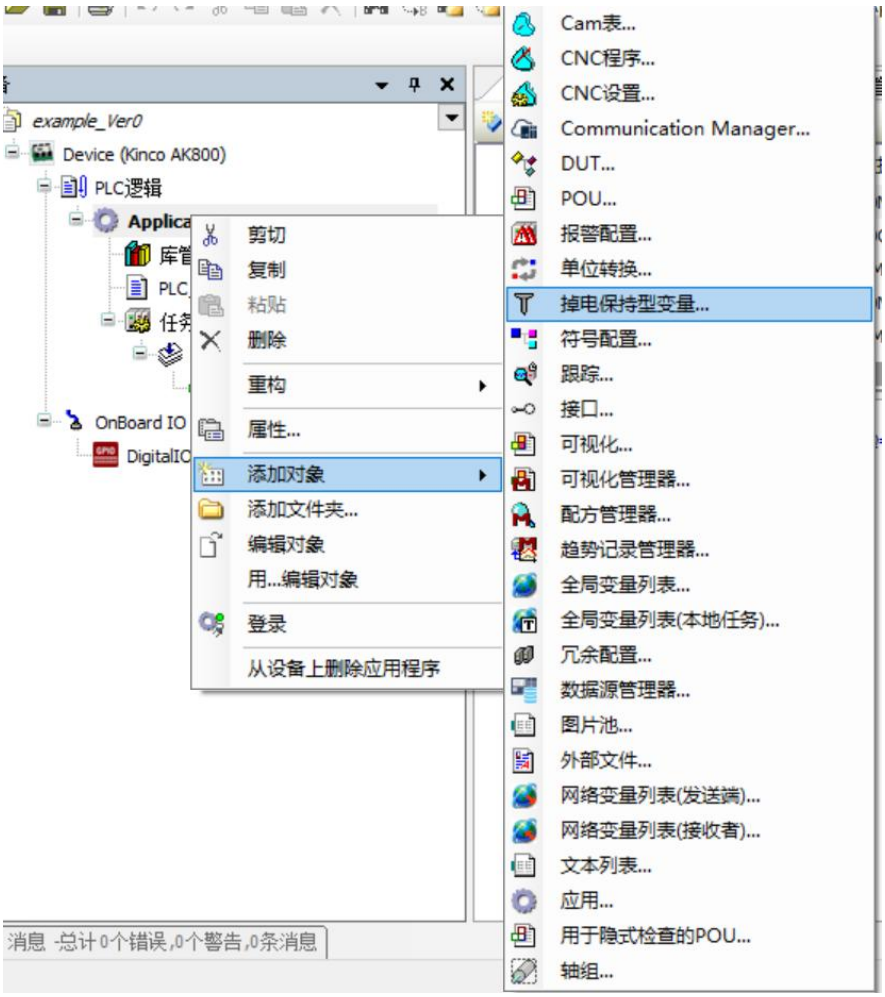
例如：需要定义REAL型变量，其地址为%MD100。若直接使用地址%MD其数据类型为DWORD型而不是REAL型。此时就需要用“变量+地址”的方式定义一个数据类型为REAL 型的变量，地址为%MW200（只需填入初始地址，长度根据数据类型自动判断），从而实现了在%MD100上定义一个REAL 型变量。

4.4.4 保持型和永久型变量

在很多工程中，通常需要数据具有掉电保持功能，使控制器断电后数据不丢失。在定义变量时，可以直接定义变量为保持型变量，此时变量自动保存于R存储区，具有掉电保持功能。在自动定义时，在自动定义对话框的左下角，选择“保持”（“RETAIN”）选项，变量就自动定义为保持型变量。在手动定义时，将变量定义在“VAR RETAIN”（POU局部变量时）或“VAR_GLOBAL RETAIN”（全局变量时） 和“END_VAR” 之间，也可以定义该变量为掉电保持变量。

“保留变量”在控制器正常启动时或者在指令热复位使用时，且即使控制器随机关机，都能保持它们的值。当程序重新开始时，保存的的值将继续使用。所有其他的变量将被重新初始化为定义时的值或者进行标准初始化。

除此之外，控制器还支持永久型变量定义，与保持型变量的区别是在“冷复位”和程序重新下载后也保留原值。所以它们只能按照使用“初始复位”进行初始化。永久型变量的定义方式只有一种，只支持全局变量，其建立途径如下图所示：



图：4.4-4：建立永久型变量表

注意：

1.保持性（“RETAIN”）变量可以定义为全局变量，也可定义为程序（PRG）和功能块（FB）的局部变量，但不能定义为函数（FUN）的局部变量。

2.永久型变量只可定义为全局变量，且只能在指定区域定义。

3.保持变量和永久保持变量在不同操作中对变量的保持特性见下表：（√表示能保持）

表 4.4-3：保持变量和永久保持变量的区别

	RETAIN	PERSISTENT RETAIN	备注
冷复位	×	√	
热复位	√	√	
初始复位	×	×	
掉电重启	√	√	
在线修改	√	√	
新程序登录并下载	×	√	

4.4.5 指针变量

指针变量是一类特殊的变量。在控制器中，所有的存储区都占用CPU 的地址，这个地址被称为绝对地址。不管是I 区、Q 区、M 区，都占用CPU 的一部分地址。而I 区、Q 区和M 区通过寻址访问的地址，诸如%MW100，可以认为是相对地址。指针是指数据区的绝对地址而不是相对地址。

相邻的两个相对地址，其绝对地址也是相邻的。假如%MW100 的绝对地址为pt，则%MW102 的绝对地址为pt+2。%MW200 的绝对地址就是pt+100。

在编程时，假如利用指针的这种关系，可以很方便地实现一些比较复杂的功能。在使用之前，同样需要定义指针变量。

指针变量的定义与其他数据类型定义类似，只是其数据类型为POINTER TO <数据类型>

指针定义的语法格式：

<指针名> ： POINTER TO <数据类型>功能块>;

示例：

1：变量声明及程序编写如下：

```
PROGRAM POU
VAR
    var_1: INT:=5;//定义一个整数，并赋初值
    Var_2: INT;//定义一个整数
    pt :POINTER TO INT;//定义一个整型数据的指针pt
END_VAR
```

```
pt := ADR(Var_1); //取出Var_1 变量的地址，将地址值赋给pt
Var_2:= pt^; //将指针pt 所指地址的值赋给Var_2，即Var2=5
```

图 4.4-5：变量声明及程序编写示例

2：仿真运行结果为：

Device.Application.POU

表达式	类型	值	准备值	地址	注释
var_1	INT	5			定义一个整数，并赋初值
Var_2	INT	5			定义一个整数
pt	POINTE...	16#0000026...			定义一个整型数据的指针 pt

```
pt:=16#0000026F6FD208AE= ADR(Var_1  5 ); //取出Var_1 变量的地址，将地址值赋给pt
Var_2  5 := pt^  5 ; //将指针pt 所指地址的值赋给Var_2，即Var2=5
RETURN
```

图 4.4-6：变量声明及程序编写示例运行结果

在 M 数据区的%MW100 开始的地址中，存放了100 个WORD 型变量，现在需要将这些值转移至%MW300 开始的100 个字内，使其具有掉电保持功能。因为数据区比较长，采用指针方式，可以很方便地实现数据区的转移。这里需要定义两个指针变量pt1 和pt2，一个指向%MW100，另一个指向%MW300。这里还需要用到一个取地址指令ADR 和读取指针数值指令 ^ 。

变量定义如下图所示，其中变量 Var_3 用于传输100 个字节。

在这里采用ST方式编写程序，关于ST语言编程，请参见9.3 章节。采用一个FOR循环语句，每次循环，pt1 和pt2 的值均加2（因为是WORD 类型指针），然后将pt1 的值赋值给pt2 即可。

```
PROGRAM POU
VAR

    pt :POINTER TO INT;//定义一个整型数据的指针pt
    Var_3:INT;
    pt1: POINTER TO WORD;
    pt2: POINTER TO WORD;
END_VAR
```

```
pt1:=ADR(%MW100);
pt2:=ADR(%MW300);
FOR Var_3:=1 TO 100 BY 1 DO
pt2^:=pt1^;
pt1:=pt1+2;
pt2:=pt2+2;
END_FOR
```

图 4.4-7：指针型变量示例

4.5 数组

CoDeSys 对数据的管理功能是非常强大的。他不但支持多种数据类型，也支持多维数据。在编程时，可以根据基本数据类型来定义一维、二维和三维数组。数组可以采用自动定义，也可以在变量声明区手动定义。

数组的标识符为 ARRAY 。数组定义的语法格式：

<数组名> ： ARRAY [<L1>..<U1>, <L2>..<U2>, <L3>..<U3>] OF <基本数据类型>;

其中L1、L2 和L3 表示字段范围的最小值，U1、U2 和U3 表示字段范围的最大值。字段范围必须是整数。假如是一维数组，则只需设置L1 和U1 即可；假如是二维数组，则需要设置L1、U1 和L2、U2；假如是三维数组，则L1、U1、L2、U2 和L3、U3 均需定义。

下图为定义一个数组元素数量为 10 的一个一维数组的自动定义对话框：

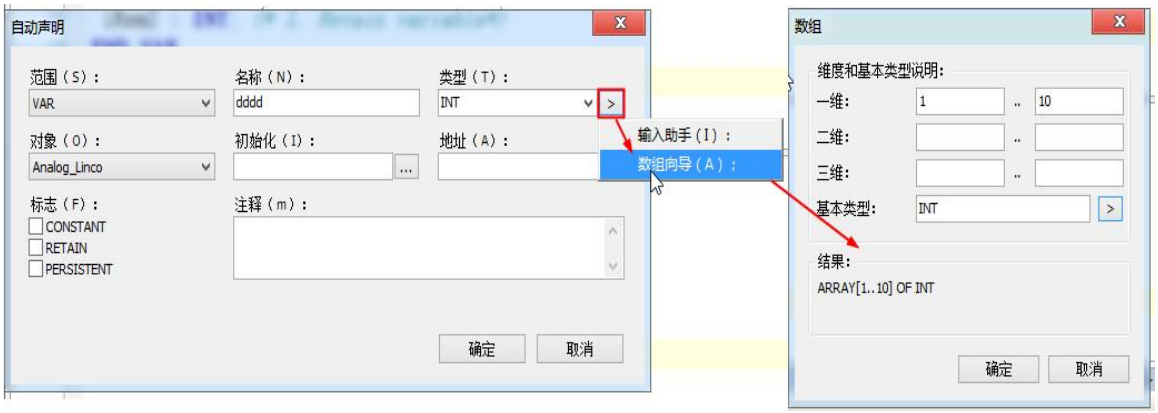


图 4.5-1：数组型变量定义

在数组定义的同时，给数组中的元素赋值称为初始化数组。在数组定义时，可以初始化数组中所有元素，也可以不进行初始化。

举例：定义数组

```
Card_gameL:ARRAY [1..13, 1..4] OF INT; (*定义一个整型的二维数组Card_game*)
```

图 4.5-2：以 ST 语言定义一个二维数组

举例：数组的完全初始化

```
Arr1:ARRAY [1..5] OF BYTE:= [1,2,3,4,5];
Arr2:ARRAY [1..2,1..2] OF INT := [1,3(7) ]; //即1,7,7,7 的缩写形式
Arr3:ARRAY [1..2,1..2,1..2] OF INT := [2(0),4(4),2,3];//即0,0,4,4,4,4,2,3 的缩写形式
```

图 4.5-3：以 ST 语言定义数组并赋初值

举例：数组的部分初始化

```
Arr4:ARRAY [1..10] OF BYTE:= [1,2];
```

图 4.5-4：数组的部分初始化

对于那些没有预先赋值的元素，按照基本数据类型的缺省初始值进行初始化。在此例中，元素[3]到[10]被初始化，其值为0。

4.6 自定义数据类型

在一些实际应用场合，需要用到一些配方数据。每组配方包含很多参数，而这些参数的数据类型是不同的，诸如可能包含REAL 型、WORD 型或TIME 型等，那么这里使用自定义数据类型就能非常方便地实现配方数据的管理。

在2.5.2 节曾经提到过自定义数据类型。在POU组织器单击右键，就可以添加自定义数据类型。

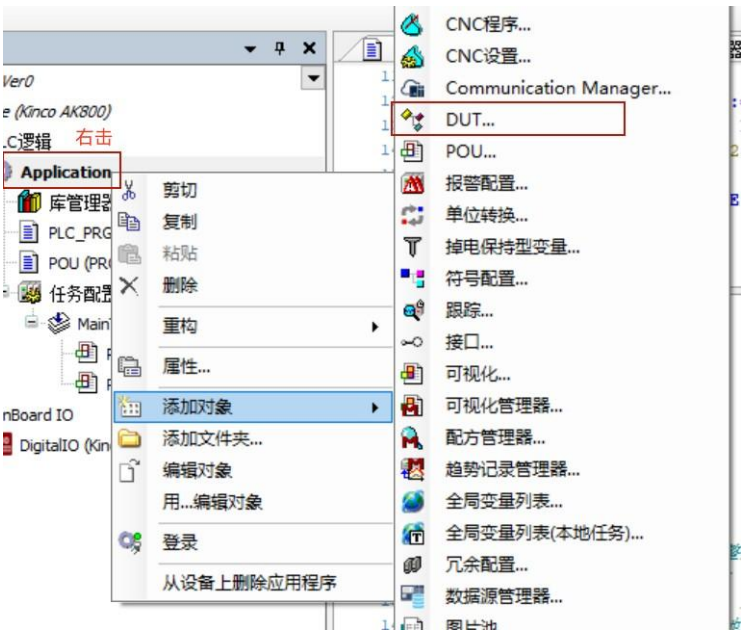


图 4.6-1：自定义数据类型路径

新建一个数据类型，命名规则同变量命名规则。命名完之后，该标志符就可以作为一个结构用来表示这个数据类型。

结构体变量以关键字“VAR INPUT”和“STRUCT 开始，关键字“END_STRUCT” 和“END_TYPE”结束。

定义结构体变量的语法格式：

```
TYPE <结构名>:
STRUCT
<变量声明1>
<变量声明 2>
...
<变量声明n>
END_STRUCT
END_TYPE
```

结构体是一种可以在整个工程中被识别的数据类型，而且可以象引用标准数据类型一样引用结构。唯一的限制是，结构变量不能指定地址，即不允许进行“AT”声明。

举例：定义名为“Polygonline”的结构体：

```
1  TYPE Polygonline :
2  STRUCT
3      Start:ARRAY [1..2] OF INT;
4      Point1:ARRAY [1..2] OF INT;
5      Point2:ARRAY [1..2] OF INT;
6      Point3:ARRAY [1..2] OF INT;
7      Point4:ARRAY [1..2] OF INT;
8      End:ARRAY [1..2] OF INT;
9  END_STRUCT
10 END_TYPE
11
```

图 4.6-2：结构体定义示范

举例：实例化结构体：

```
P1:Polygonline:=(Start:=[3,3], Point1:=[5,2], Point2:=[7,3], Point3:=[8,5], Point4:=[5,7], End:= [3,5]);
```

图 4.6-3：实例化结构体示范

举例：结构体数组的初始化：

```
1  TYPE STRUCT1 :
2  STRUCT
3      p1:INT;
4      p2:INT;
5      p3:DWORD;
6  END_STRUCT
7  END_TYPE
```

```
A1: ARRAY [1..3] OF STRUCT1:=[(p1:=1,p2:=10,p3:= 3),(p1:=2,p2:=0,p3:=2),(p1:=4,p2:=5,p3:=1)];
```

图 4.6-4：结构体数组的定义及初始化示范

访问结构成员的语法：

```
<结构名>.<结构成员名>
```

例如结构体名称为“Week”，其中的一个成员名为“Monday”，则可以用下面的形式访问该成员：“Week.Monday”，尽可能多的使用结构体来描述对象，而不是使用孤立的多个变量，这有利于管理程序中各个对象的状态和属性。比如，程序中需要控制一组控制轴对象（以CANopen控制为例），我们可以创建“Axis_CAN”这样的结构体，该结构体应该包含Node（站号），CurrentPosition（当前位置）,CurrentVelocity（当前速度），CtrolWord（控制字）等等这样的要素。这样我们可以方便的维护各个轴管理各个轴的数据。可以自定义数据类型，这是CoDeSys软件较其他编程软件的优势。

4.6.1 枚举

新建一个数据类型，命名规则同变量命名规则。命名完之后，该标志符就可以作为一个枚举用来表示这个数据类型。

枚举是由一长串的数字常量组成的自定义数字类型，这些常量称为枚举值。

枚举以关键字 “TYPE” 开始，以关键字“END_TYPE” 结束。声明枚举的语法：

```
TYPE<标识符>：(<enum_0>,<enum_1>,…,<enum_n>);
END_TYPE
```

注意：

枚举值中可以包含标识符变量，初始化时，该标识符变量被初始化为该枚举中的一个值。可以把任何数字量分配给枚举值。定义的枚举值与其他标准数据类型兼容，可以就像使用INT数据类型一样对其进行操作。

如果枚举值没有被初始化，则从第一个标识符开始，顺次赋给标识符0,1,2,...。但当枚举中的某个成员赋值后，其后的成员按依次加1的规则确定其值。

举例：如下图所示，由于开头的两个变量并未被赋初值，故第一个变量“red”初值为0，第二个变量“yellow”初值在上一个变量初值的基础上加一，即为1。

```
3  TYPE Traffic_signal :
4  (
5      red,
6      yellow,
7      green:=10
8  );
9  END_TYPE
```

图 4.6-5：结构体部分初始化规则-1

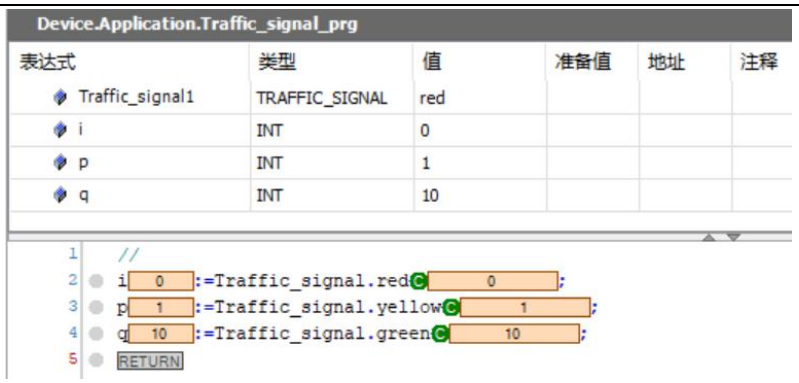


图 4.6-6：结构体部分初始化规则-2

程序不仅只用于数值计算，还更广泛地用于处理非数值的数据。例如：性别、月份、星期几、颜色、单位名、学历、职业等，都不是数值数据。在其它程序设计语言中，一般用一个数值来代表某一状态，这种处理方法不直观，易读性差。如果能在程序中用自然语言中有相应含义的单词来代表某一状态，则程序就很容易阅读和理解。也就是说，事先考虑到某一变量可能取的值，尽量用自然语言中含义清楚的单词来表示它的每一个值，这种方法称为枚举方法，用这种方法定义的类型称枚举类型。

4.6.2 联合

新建一数据类型，命名规则同变量命名规则。命名完之后，该标志符就可以作为一个联合用来表示这个数据类型。

联合表示几个变量公用一个内存位置，在不同的时间保存不同的数据类型和不同长度的变量。

联合变量以关键字“TYPE”和“UNION”开始，关键字“END_UNION”和“END_TYPE”结束。

定义结构体变量的语法格式：

```
TYPE <结构名>:
UNION<变量声明1>
<变量声明 2>
        ...
<变量声明n>
END_UNION
END_TYPE
```

举例说明联合的用法：

1.先定义一个联合，2个不同长度类型的变量：

```
TYPE DUT :
UNION
    a1:WORD;
    a2:DWORD;
END_UNION
END_TYPE
```

图 4.6-7：声明联合示范-1

2.在程序里声明变量“A”调用联合，并定义两个不同数据类型的变量，ST 语言编程如下：

```
PROGRAM POU_1
VAR
    A:  dut;
    Trig: BOOL;
    wA: WORD;
    dwA:DWORD;
END_VAR
```

```
IF Trig THEN
A.a1:=16#1234;//执行时，给联合的WORD变量赋值为16#1234
END_IF
wA:=A.a1;//用来监控联合的WORD变量值
dwA:=A.a2;  //用来监控联合的的DWORD变量值
```

图 4.6-8：声明联合示范-2

3.执行结果：

执行前：

```
IF TrigFALSE THEN
A.a1 0 :=16#1234;//执行时，给联合的WORD变量赋值为16#1234
END_IF
wA 0 :=A.a1 0 ;//用来监控联合的WORD变量值
dwA 0 :=A.a2 0 ;  //用来监控联合的的DWORD变量值RETURN
```

执行后：

```
IF TrigTRUE THEN
A.a1 16#1234 :=16#1234;//执行时，给联合的WORD变量赋值为16#1234
END_IF
wA 16#1234 :=A.a1 16#1234 ;//用来监控联合的WORD变量值
dwA 16#00001234 :=A.a2 16#00001234 ;  //用来监控联合的的DWORD变量值RETURN
```

图 4.6-9：声明联合示范-3

可以看到执行后给联合中定义的 WORD 变量赋值，则另外一个 DWORD 变量的低 8 位值也会变成一样的值。

结构和联合看起来很像，但实际应用有明显区别，具体如下：

1. 结构和联合都是由多个不同的数据类型成员组成，但在任何同一时刻,联合中只存放了一个被选中的成员，而结构的所有成员都存在。

2. 对于联合的不同成员赋值，将会对其它成员重写，原来成员的值就不存在了，而对于结构的不同成员赋值是互不影响的。

第五章 程序组织单元

本章主要讲述了CoDeSys 的一个重要概念—程序组织单元（POU）。程序组织单元是组成工程的基本结构。任何复杂的工程都是由众多POU组成的，POU包括程序、功能块和函数。

5.1 章节讲述了POU的一些基本概念，使得读者对POU的概念有一定的了解。

5.2 和 5.3 章节讲述POU的使用规则，包括POU的建立及调用等的一些规则。

5.4 章节讲述了CoDeSys 软件如何管理POU。在本章的学习过程中，若遇到一些概念和规则不易理解，建议初学者可以跳过这些内容，同时在学习过程中采用“边学习、边练习”的方式，这样有助于更快地理解本章的内容。同时建议在学习后面的内容时也采取这种方法。

5.1 POU 的概念

POU 是程序组织单元（Program Organization Unit）的简称。POU 可以是函数（Function）、功能块（Function Block）或程序（Program）。其中程序和功能块的编程语言可以在 LD、FBD、IL、ST、SFC 及 CFC 中进行选择。函数的编程语言可以是 LD、FBD、IL、ST 或 CFC，但不能是 SFC。

关于 POU 的编程语言的详细讲述，请参见第九章。

5.1.1 POU 的类型

POU 分为程序（Program）、功能块（Function Block）和函数（Function）等三种类型。

- 程序 (Program)

程序是为了完成某项任务而编写的语句序列，是一组指令的集合。程序是唯一可执行的POU，是逻辑执行的主体。程序可以通过任务组态来激活，也可以通过其它程序来调用。

- 功能块 (Function Block)

功能块是预先编好的、实现某种运算的程序。功能块本身不能单独执行，只能由程序调用功能块执行。在执行时，输入量可以是一个或多个值，输出量可以是一个或多个执行结果。与函数不同，功能块本身没有返回值。功能块就如一个模板，可以复制多个相似的功能块实例，去分别对不同的对象，进行操作处理。这样可减少相似编程代码的反复编写，提高编程效率。

- 功能 (Function)

功能也叫函数，函数是预先编好的、实现某种运算的程序。函数在执行时，会针对一系列特定的输入，产生一个输出结果，这个输出结果被赋给函数本身，称为返回值。函数只能被其它POU调用，函数本身不能单独执行。功能不需要实例化，没有保留内存，每次调用的入口参数相同，得到的反馈结果都相同。

三种类型的POU不同点介绍如下：

1. 如果一个程序被其它程序调用并且值已经发生改变，这些变化将持续到下次程序调用，

或者在此期间被其他POU调用。这与功能块的调用不同，在功能块中只有定义实例的变量值发生改变，所以这个改变只能在再次调用时影响对应功能块的实例。

2. 函数（与功能块或者程序相比）不包含内部信息，也就是说，对于相同的输入值（输入参数）会产生相同的结果（输出值）。因为这个原因，函数中不能包含全局变量或者变量地址。

3. 功能块与函数不同的是：输出变量的值和一些内部变量的值将会在功能块下次执行前保持不变。所以对于功能块来说，相同的输入可能导致不同的输出。

5.1.2 POU 的调用

POU 的调用有两种方法。

1.被其它已经调用的 POU 来调用。

2.通过任务配置来调用，这种方法仅限于程序调用。当程序中没有进行任务配置时，系统会自动调用主程序控制器_PRG。

POU的调用要遵循以下原则，如图所示。

- 程序可以调用函数、功能块和其它程序。
- 功能块可以调用函数和其它功能块。
- 功能可以调用功能。

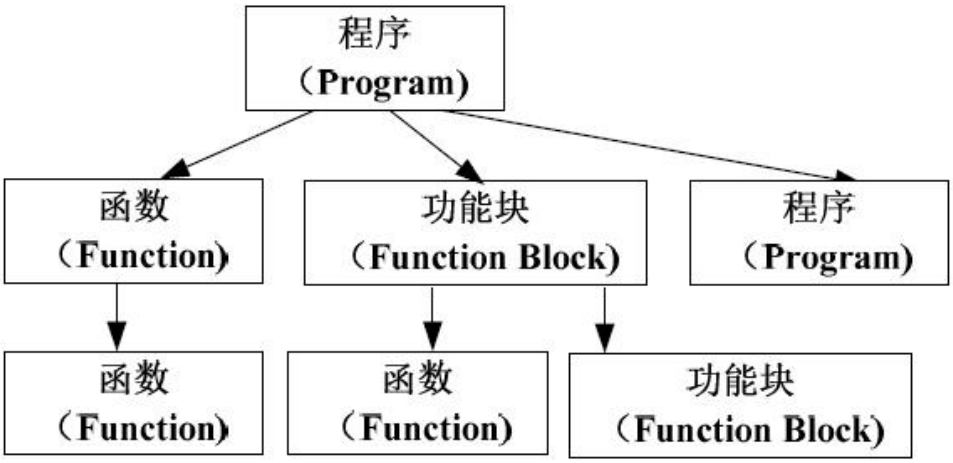


图 5.1-1：POU 的调用规则

5.1.3 POU 的组成

POU包含一个声明部分和一个代码部分（程序区）。

声明部分：在变量区创建、显示POU变量。用户可在引用变量之前进行定义，也可以在引用时利用变量定义对话框定义。无论是局部变量还是全局变量，在变量定义过程中必须遵

循一定的格式，具体的格式参见4.4章节。

代码部分：在程序区创建，是POU的主体，用户可以选用IEC标准编程语言来编写。

5.1.4 主程序 PLC_PRG

默认工程设置下，会生成一个默认的任务配置，程序“PLC_PRG”被默认为主程序，是一个特殊的POU。每个工程必须包含这个主程序才能正常运行。系统默认每个控制周期调用一次这个POU，不需要进行额外的任务组态。所以，工程必须以“PLC_PRG”为主程序，通过它来实现对其它POU的调用。

如果要修改这个默认配置，可以再任务配置编辑修改，详情参考第六章

5.2 创建 POU

5.2.1 创建程序

在POU组织器中右键选中“Application”选项卡，在程序列表中点击右键，弹出添加对象菜单，选择POU，在弹出的“创建程序组织单元”对话框中，选择“类型”为“程序”；“实现语言”可以选择IL、LD、FBD、SFC、ST、CFC 之一；“名称”为程序名，名称应尽量采用能反映其实际功能的字符，便于识别；点击“确认”，便可创建程序。

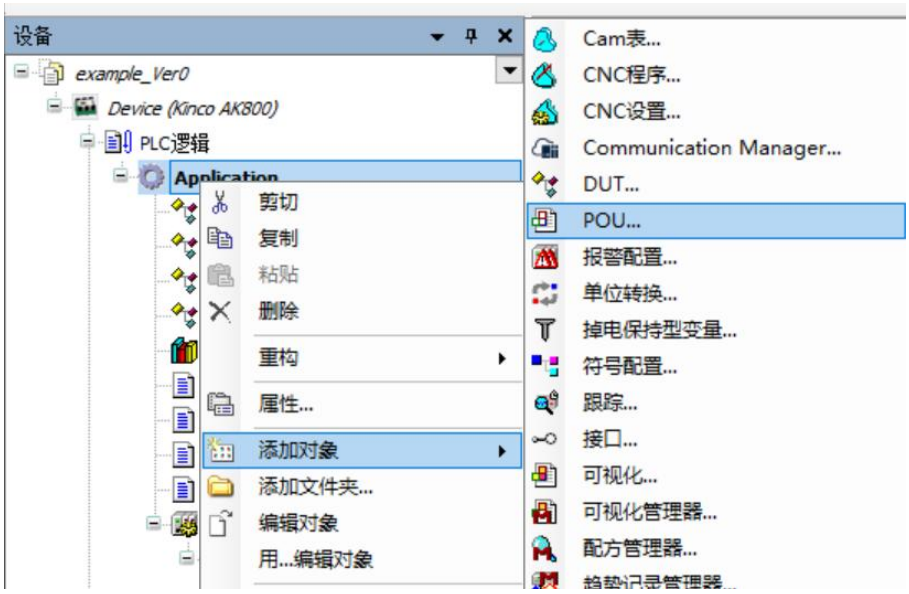


图 5.2-1：创建程序路径

5.2.2 创建功能块

功能块的创建和程序类似。右击“Application”,选择创建“POU”，在弹出的对话框中选择“类型”为“功能块”，然后进行编程语言的选择和命名即可。

功能块提供扩展功能块，多个接口调用功能块，还可设置访问权限等高级功能，以便给使

用者提供更多便利，但对使用者编程水平要求较高，如有需要，详情参考软件自带帮助（CoDeSys帮助平台：https://zh.helpme-codesys.com/，或直接从软件“帮助”菜单下的“目录”选项打开，如下图所示）。

图 5.2-2：CoDeSys 在线帮助路径

5.2.3 创建功能

功能的创建和外形结构都与功能块类似，区别是功能只有一个输出端。创建好后，自动生成函数结构体，可以在里面添加所需要的内容。下面以制作一个数学计算式为例来介绍函数的编写，如图所示。从“VAR_INPUT”到第一个“END_VAR” 之间用于定义输入变量，从“VAR”到第二个“END_VAR” 之间用于定义中间变量，然后在编辑窗口进行函数算法编写。编写完毕后保存。编译通过后，可以在其它程序中直接调用，如下图所示：

```
1  FUNCTION Fct : LREAL//求和
2  VAR_INPUT
3      Praml: REAL;
4      Pram2: REAL;
5
6  END_VAR
7  VAR
8
9  END_VAR
10
1  Fct:=Praml+Pram2;
```

图 5.2-3：“求和”功能示范-1

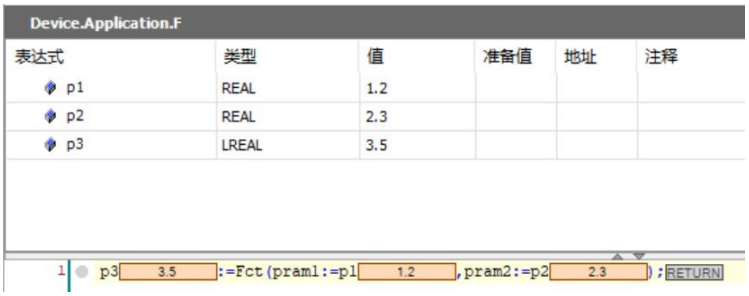


图 5.2-4：“求和”功能示范-2

注意：功能并没有所谓的“VAR_OUTPUT”，因为功能只有一个输出，它的输出即为它的返回值，功能的名称即是功能的返回值。

5.3 调用 POU

关于POU的调用方法和调用原则在5.1.2 中已经介绍，下面分别讲述了如何调用程序、功能块和函数。

5.3.1 调用程序

程序是唯一可执行的POU。程序可以调用功能块和函数。程序声明以关键“PROGRAM”开始，如图所示。

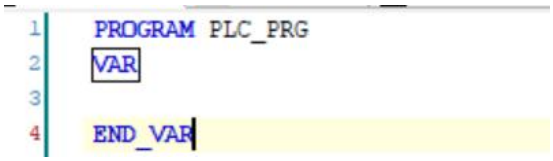


图 5.3-1：程序以“PROGRAM”为标识

程序可以被其它程序调用，但不允许在功能中调用程序，也不允许程序递归调用，也不允许调用自身。如果调用使程序的输出值发生变化，程序会保持这个结果，直到下一次被调用。

示例如下：

ST 语言调用：（示例程序名称为“AlarmHandle”）
AlarmHandle();
Warnning ： = AlarmHandle.Alarm;

图 5.3-2：ST 语言调用程序示范

5.3.2 调用功能块

- **功能块：**

一个功能块是一个 POU，本身没有返回值，可以输出一个或多个值。功能块声明以关键字“FUNCTION_BLOCK”开始。

- **功能块示例声明：**

要想调用功能块，必须对功能块进行实例声明。实例声明如下：

<实例名称>:<功能块名称>;

每个功能块实例就是一个独立的、可完成特定逻辑功能的活动对象。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。通过定义实例实现对功能块的调用。

从外部只能改变功能块输入和输出参数。不允许对功能块的内部变量直接赋值。功能块实例名可以作为函数或功能块的输入。功能块的调用，只改变功能块实例中的值，结果只有当相同实例被调用时才起作用。在文本语言 IL 和 ST 中，可以在功能块实例名后加圆括号，设置

输入参数的初始值。和声明变量时初始化一样，赋值使用符号：“:=”。

对于 SFC 语言，功能块调用只能发生在单步运行方式下。功能块执行以后，所有的值都保留到下次执行之前保持不变。由于功能块中存在中间变量，使表面上相同的输入参数实际上可能输出不同的输出值。

采用不同语言调用上述功能块如下图所示（此处均以调用“TON”功能块做演示）。

1.梯形图（LD）调用功能块：

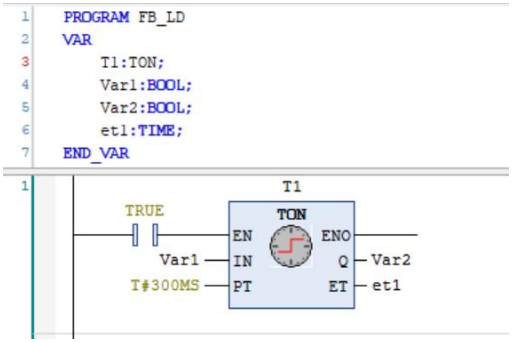


图 5.3-3：LD 调用功能块示范

2.结构化文本（ST）调用功能块：

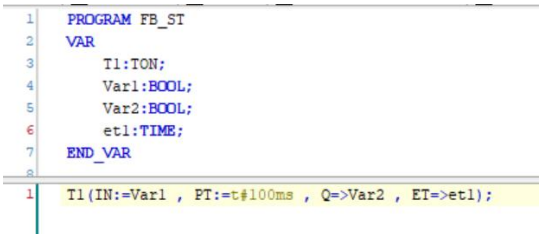


图 5.3-4：ST 调用功能块示范

3.功能块图（FBD）调用功能块：

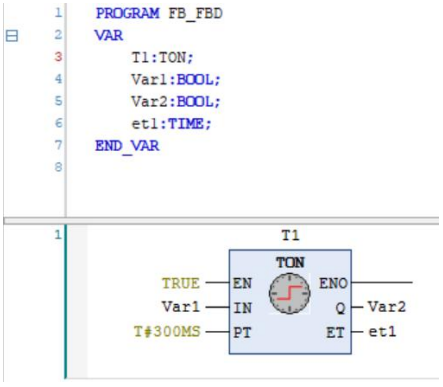


图 5.3-5：FBD 调用功能块示范

4.指令表（IL）调用功能块：

```
PROGRAM FB_IL
VAR
    T1:TON;
    Var1:BOOL;
    Var2:BOOL;
    et1:TIME;
END_VAR
```

```
CAL              T1(
             IN:= Var1,
             PT:= T#300MS,
              Q=> Var2,
             ET=> et1)
```

图 5.3-6：IL 调用功能块示范

5.连续功能图（CFC）调用功能块：

```
PROGRAM FB_CFC
VAR
    T1:TON;
    Var1:BOOL;
    Var2:BOOL;
    et1:TIME;
END_VAR
```

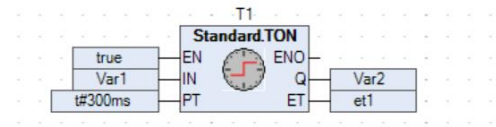


图 5.3-7：CFC 调用功能块示范

6.顺序流程功能图（SFC）调用功能块：

SFC 编程中，功能块的调用最终需要通过在“步”中以 ST/FBD/LD/IL/CFC 语言实现，SFC 语言的特性导致其无法直接调用功能块，SFC 可以在“步”中继续嵌套 SFC 步。

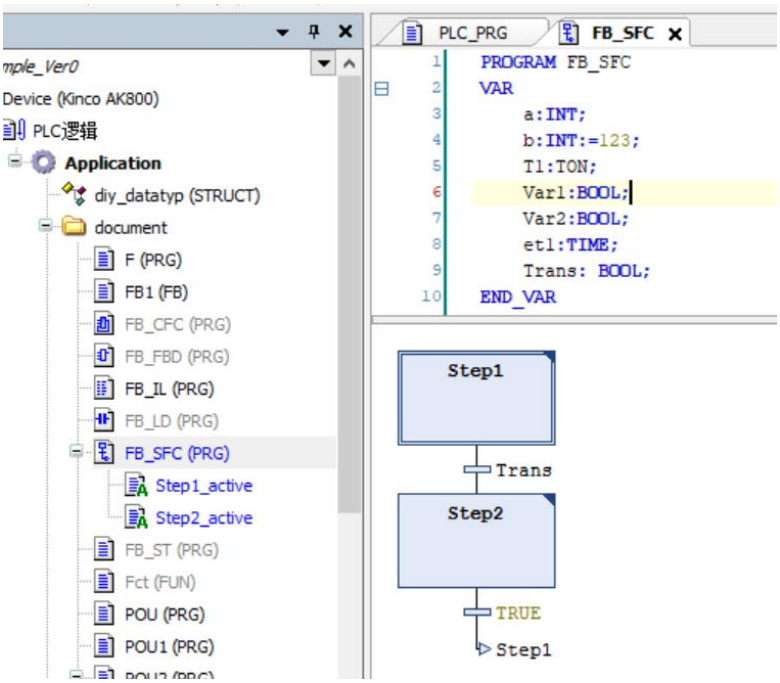


图 5.3-8：SFC 调用功能块示范-1

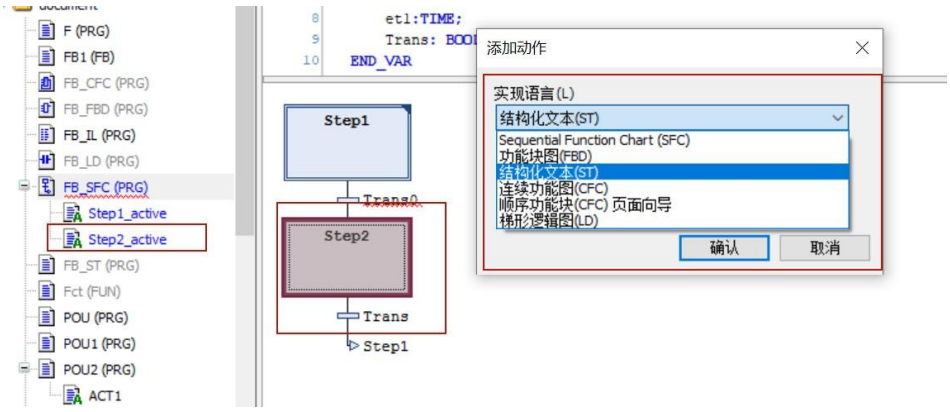


图 5.3-9：SFC 调用功能块示范-2

➢ 自定义功能块

此处以一个应用案例来说明自定义功能块的一般操作。以ST语言自定义了一个功能块“FB1”，有两个输入变量和两个输出变量。其中一个输出是两个输入的乘积，另一个输出是相等比较结果。功能块创建过程此处不再赘述，请参照5.2.2。

```
FUNCTION_BLOCK FB1
VAR_INPUT
    var_in1:DINT;
    var_in2:DINT;
END_VAR
VAR_OUTPUT
    var_mul:DINT;
    var_equal:BOOL;
END_VAR
VAR
END_VAR
```

```
var_mul:=var_in1+var_in2;
IF var_in1=var_in2 THEN
    var_equal:=TRUE;
ELSE
    var_equal:=FALSE;
END_IF
```

图 5.3-10：自定义功能块编程示例

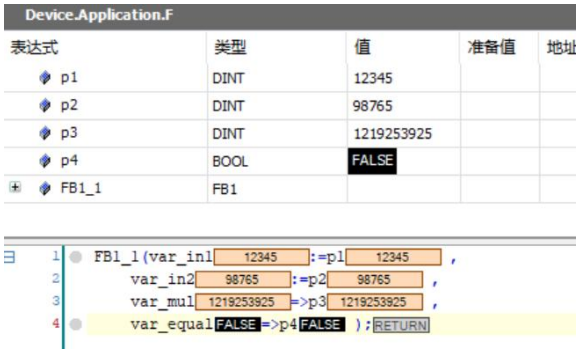


图 5.3-10：自定义功能块示例运行结果

带“EN”（使能端）与不带“EN”（使能端）的功能块的区别主要体现在图形编程中（如LD\FBD 等），带“EN”输入意为：根据“EN”的输入情形决定是否调用，若“EN”为 TRUE 则调用，否则不调用。而不带“EN”则为无条件调用。

在自定义的功能块中，不建议用户在自己的功能块上多构造一个类似的“EN”使能端，因

为功能块在系统编译生成的时候，就已经隐含存在一个使能端，且该使能端恒为“TRUE”，即为无条件执行状态。如果用户在此基础上再增加一个自定义的使能端，将导致：自定义的使能端条件成立下，功能块会按条件执行，当自定义的使能端条件不成立的代码则会无条件的执行，特别是当程序中不同的地方都有调用该功能块时，可能会出现难以理解的逻辑。

5.3.3 调用功能

功能也是 POU 的一种。功能执行时，会对一系列特定的输入产生唯一数据类型的输出结果。相对于功能块而言，功能只有一个输出结果，没有任何内部条件。也就是说，只要给定相同的输入参数，调用功能必定得到相同的运算结果。平时所使用的各种数学运算，例如：SIN(X)等就是典型的功能类型。当定义功能时，功能必须输出一个数据类型作为返回值（返回数据类型）。功能的计算结果赋给功能本身，即功能的输出变量就是功能名本身。功能定义以关键字“FUNCTION”开始。

功能的调用无需实例化，所以在程序变量声明区无需声明功能是何种类型。功能的内部变量不保存，其有效期只从调用开始到调用结束。功能块与之不同，无论你是否调用，功能块的内部变量在其实例化后就一直存在，直到控制器主程序退出为止，即功能块的内部变量有效期维持在整个控制器程序运行期间。

5.4 管理 POU

在 CoDeSys 软件主界 POU 组织器中，在程序列表中选择相应的程序，点击鼠标右键，弹出管理 POU 菜单，如下图所示，可以对程序进行删除，复制，重命名，添加动作等。

程序提供添加方法、属性、转移等高级功能，给使用者提供更多便利，但对使用者编程水平要求较高，如有需要，详情参考软件自带帮助（CoDeSys帮助平台：https://zh.helpme-codesys.com/，或直接从软件“帮助”菜单下的“目录”选项打开）。

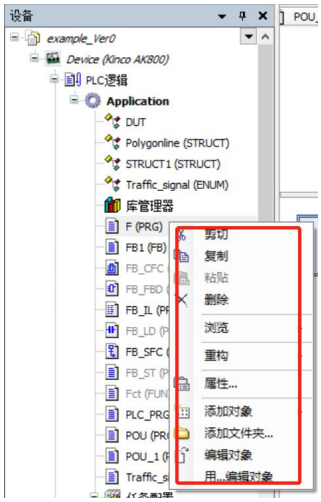


图 5.4-1：POU 管理操作

5.4.1 添加动作

动作代表某种功能，隶属于某个程序或功能块，可以被其他的程序或功能块调用。由于动作隶属于程序或功能块，所以动作中的变量在所隶属的程序或功能块变量区中声明，即动作自身无变量区。

调用动作的格式为：“程序名.动作名”，“实例名.动作名”。调用动作中变量的格式为“程序名.变量名”，“实例名.变量名”。

例如，编写程序 POU1 和 POU2，程序 POU2 添加一个动作 ACT1 ，再让程序 POU1 调用动作 ACT1 的步骤如下：

右击“Application”，选择新建“POU”，创建“POU1”和“POU2”2 个程序，然后右击“POU2”，在弹出的菜单中选择“添加对象”，选择对象为“动作”，在弹出的对话框中确定动作的编程语言以及动作名，命名要尽量采用能反映其实际用途的字符，便于识别。点击确认即可，如图所示。

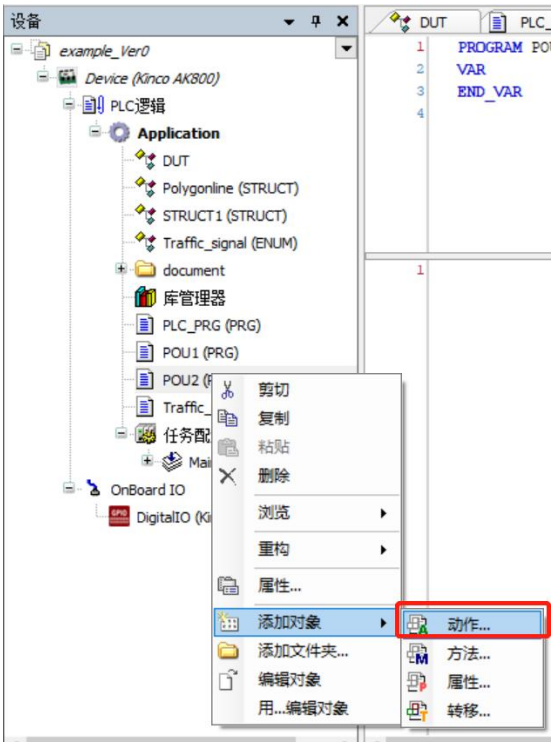


图 5.4-2：为程序添加动作

创建完成后，可在“POU2”目录下找到“ACT1”，双击打开即可进行编辑，如下图。

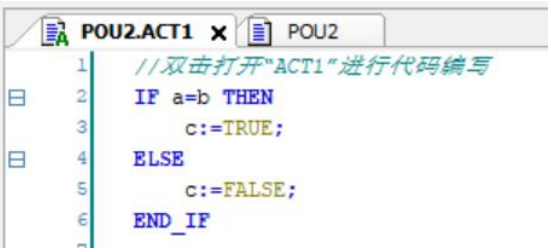


图 5.4-3：“步”编辑

动作创建完后，可在主程序“POU2”中调隶属其的动作“ACT1”，并在“POU1”中引用“ACT1”中的动作变量。此处以简单的编程例子作为演示：

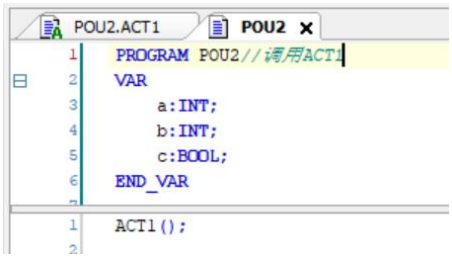


图 5.4-4：在隶属程序调用“步”

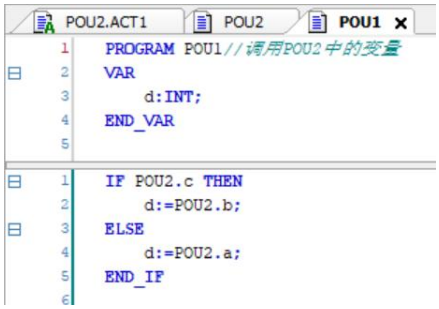


图 5.4-5：跨程序调用“步”

5.4.2 建立文件夹

对于较大的工程，为了便于管理和查阅，程序、数据类型和全局变量可以统一地组织在文件夹中，并且可以设置多级文件夹（文件夹嵌套）。通过左键点击选中后拖动可以在对象组织器的范围内移动程序。文件夹使工程的组织构架更加清楚，方便调试工程，对程序没有影响。右击“Application”，选择新建文件夹，这种方式创建的是一个一级文件夹，若右击某个文件夹，再新建文件夹讲创建一个隶属该文件夹的二级文件夹。可以通过单击文件夹名称或右击文件夹点击“属性”修改文件夹的名称。

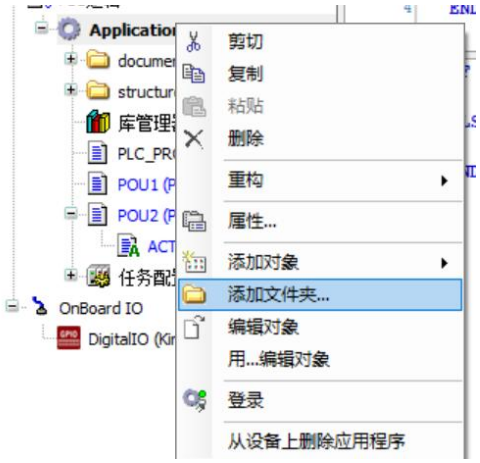


图 5.4-6：新建文件夹

第六章 控制器任务配置

在学习了控制器管理数据和 POU 之后，还需要了解控制器是如何工作的。6.1 章节就为您描述了控制器的工作方式，将有助于更好地理解程序的执行过程。

任务是控制器的一个概念。所谓的任务是指控制器所要执行的内容。一般来讲，控制器是按程序循环扫描，若需要产生中断或触发其他事件，则需在任务配置中进行设置。任务配置的过程将在第 6.2 章进行讲解。

6.1 控制器的工作过程

控制器以扫描周期循环扫描执行的方式工作。所谓扫描是指 CPU 连续执行用户程序和任务的循环过程。控制器的工作过程一般可以分为输入采样、程序执行和输出刷新等三个阶段，如图所示。

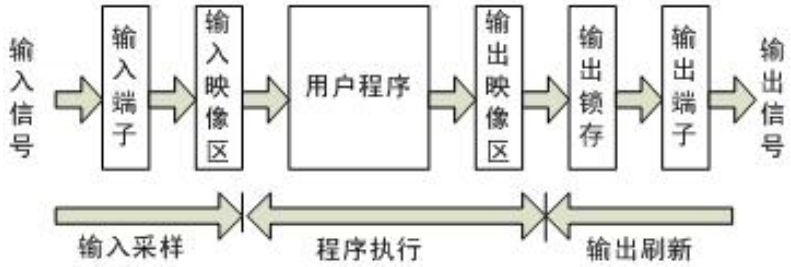


图 6.1-1：控制器工作过程图示

1.输入采样阶段：

控制器以扫描工作方式，按顺序将所有信号读入到寄存输入状态的输入映像寄存器中存储，这一过程称为采样。在本工作周期内，此采样结果的内容不会改变，而且采样结果将在控制器执行程序时被使用。

2.程序执行阶段：

控制器按顺序对程序进行扫描，即从上到下和从左到右地扫描每条指令，并分别从输入映像寄存器和输出映像寄存器中获取所需的数据，进行运算和处理，再将程序执行结果写入寄存执行结果的输出映像寄存器中保存。注意，在整个程序未执行完毕之前，程序执行结果不会送到输出端口上。

3.输出刷新阶段：

在执行完所有用户程序后，控制器 将映像寄存器中的内容送入到寄存输出状态的输出锁存器中，再去驱动用户设备，这就是输出刷新。

控制器重复执行上述三个阶段。每重复一次的时间称为一个扫描周期。在一个扫描周期中，控制器的输入扫描时间和输出刷新时间一般小于 1ms，而程序执行时间因程序长度的不同而不同。但是严格来说，控制器的执行还应当包括下述三个过程，这三个过程都是在输入扫描过程之后进行的：

1、系统自检测。检查程序执行是否正确，如果超时则停止 CPU 工作。

2、与 CoDeSys 软件交换信息。联机调试程序时才执行这一过程。

3、网络通信。当控制器配置有网络通信时，如 TCP 通讯或 CAN 通讯，则需要与通信对象进行数据交换。

当控制器投入运行后，将重复完成以上各个阶段的工作，即采用循环扫描工作过程，如下图所示。控制器工作的主要特点是输入信号集中批处理、执行过程集中批处理和输出控制集中批处理。控制器的这种“串行”工作方式，可以避免继电器-接触器控制系统中触点竞争和时序失配的问题，这是控制器可靠性高的原因之一。但是，循环扫描工作过程会导致输出相对输入在时间上的滞后，这也是控制器的缺点之一。

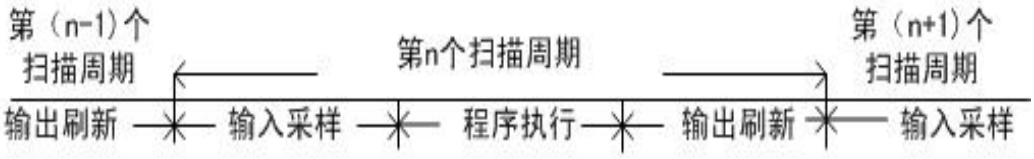


图 6.1-2：工作过程与扫描周期

控制器在执行程序时所使用的状态值不是直接从实际输入端口获得的，而是来源于输入映像寄存器和输出映像寄存器。输入映像寄存器的状态值取决于上一个扫描周期从输入端子采样取得的数据，并在程序执行阶段保持不变。输出映像寄存器中的状态值取决于执行程序输出指令的结果。输出锁存器中的状态值是上一个扫描周期的刷新阶段从输出映像寄存器转入的。

还需指出的是，在控制器中经常采用一种称之为“看门狗”（Watch dog）的定时监视器来监视控制器的实际工作周期是否超出预定的时间，以避免控制器在执行程序的过程中进入死循环或控制器执行非预定的程序而造成系统瘫痪。

6.2 任务配置

对于一个工程，可以根据需要配置多个任务，调用不同的程序。在通常情况下，建议只定义一个任务，并通过它调用主程序，其它程序则通过主程序来间接调用。这种调用方法使用与程序调用，对功能块和函数不适用。

严格地说，如果没有使用任务配置，在单任务环境中，系统默“PLC_PRG”为主程序，自动且唯一调用它，并通过它实现对其它程序的调用。如果使用任务配置，程序的调用依赖于任务分配。

6.2.1 任务配置

在“设备树”中默认有一个“MainTask”的任务，可以单击修改其名称，双击对其进行属性配置等操作。

在“设备树”中选项卡中右击“任务配置”，选择目录下的“新建任务”，输入任务名后即可插入新任务，如图所示：

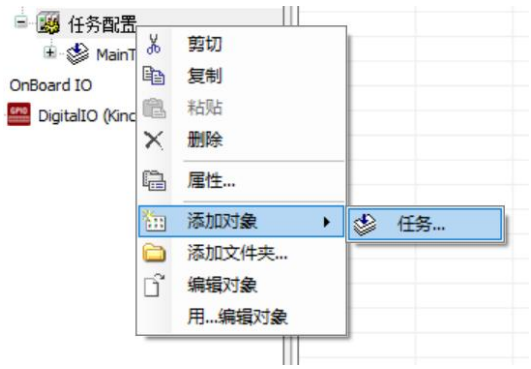


图 6.2-1：新建任务操作

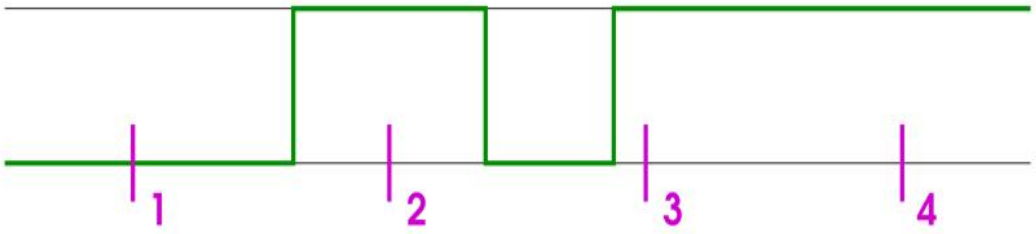
在弹出的“任务属性”窗口中可以分别填入任务类型、优先权、间隔等信息，如图所示。

图 6.2-2：任务属性编辑

- 任务名：务的名字以字母加数字的方式在新建任务时候会要求写入，之后想修改可单击任务名进行修改。
- 优先权：务的优先级（0-31）。数字越小优先级越高，系统优先处理。
- 类型：选择任务运行的方式，如循环运行（以固定的周期地处理任务）、事件触发（如果事件区域的变量得到一个上升沿，任务开始）、惯性滑行（只要启动程序，就处理任务，运行一次后自动重新启动程序，如此循环，循环的周期不固定）。
- 间隔：定义时间循环的间隔，单位可修改（ms/μs）。
- 看门狗：以设置使能和灵敏度等。

这个具体的事件为TRUE完成了状态驱动任务的条件，然而一个事件驱动任务需要事件从FALSE变为TRUE。如果任务计划的采样频率过低，事件的上升沿可能检测不到。

下面例子说明了任务对事件的反应（绿线）：



在采样点1-4（品红色）不同类型的任务展示了不同的反应：

点处行为:	1	2	3	4
状态	不开始	开始	开始	开始
事件	不开始	开始	不开始	不开始

图 6.2-3：事件任务处理规则

6.2.2 系统事件

系统事件用于在工程中调用POU，而不是用于在任务中调用POU。当相应事件触发时，调用相应的POU。

在“设备树”中选项卡中双击“任务配置”，右侧弹出任务配置窗口，点击打开右侧窗口中“系统事件”，点击添加一个新的系统事件，在弹出对话框中可选择系统事件类型，直接输入一个新函数名称以便系统自动创建所调用的函数，最后选择调用函数的编程语言。

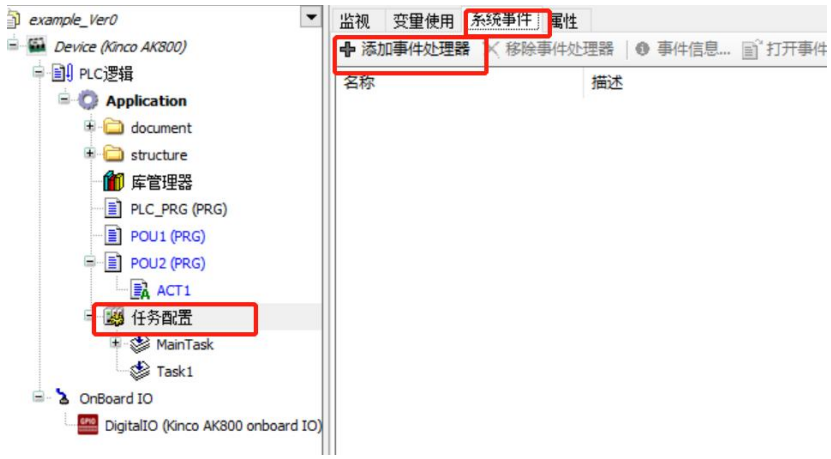


图 6.2-4：创建事件任务

可能的系统事件任务如下图：

事件	描述	任务	调试
准备开始	应用开始前调用	通讯任务	否
开始完成	应用开始后调用	通讯任务	否
准备停止	应用停止前调用	通讯任务	否
停止完成	应用停止后调用	通讯任务	否
准备重启	应用重启前调用	通讯任务	否
重启完成	应用重启后调用	通讯任务	否
准备在线改变	应用在线改变前调用	通讯任务	否
在线改变完成	应用线改变后调用	通讯任务	否
准备下载	应用下载前调用	通讯任务	否
下载完成	应用下载后调用	通讯任务	否
准备删除	应用删除前调用	通讯任务	否
删除完成	应用删除后调用	通讯任务	否
准备退出	应用退出前调用	通讯任务	否
退出完成	应用退出后调用	通讯任务	否
代码初步完成	事件在代码初始化之后发出。在任务安全部分调用，只在在线改变！（如复制代码在线改变在这里执行）	通讯任务	否
例外	时间发出，如果应用例外发生	运行系统处理任务例外——如果RTS没有任务——任务本身	取决于任务
登录	应用中登陆用户	通讯任务	否
注销	应用中注销用户	通讯任务	否
读取输入前	读取输入前调用	IEC任务	是
读取输入后	读取输入后调用	IEC任务	是
写输出前	写输出前调用	IEC任务	是
写输出后	写输出后调用	IEC任务	是
调试循环	事件周期性在调试循环发出，如果 IEC 任务 在断点处暂停	通讯任务	否
准备中止	事件恰在运行系统停止前发出	运行系统主循环	否
准备退出命令	事件在停止时退出通讯前发出	运行系统主循环	否
准备退出任务	事件在退出所有任务关闭前发出	运行系统主循环	否

图 6.2-5：允许的系统事件

6.2.3 任务调用程序

右击需要添加程序调用的任务，选择菜单下的“添加对象”，在下方菜单上选择“程序调用”，在弹出“程序调用”对话框中选择需要调用的程序，点击确认即可完成调用。

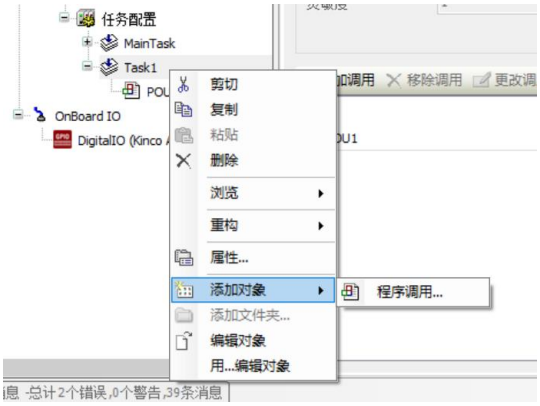


图 6.2-6：为任务配置程序调用操作-1

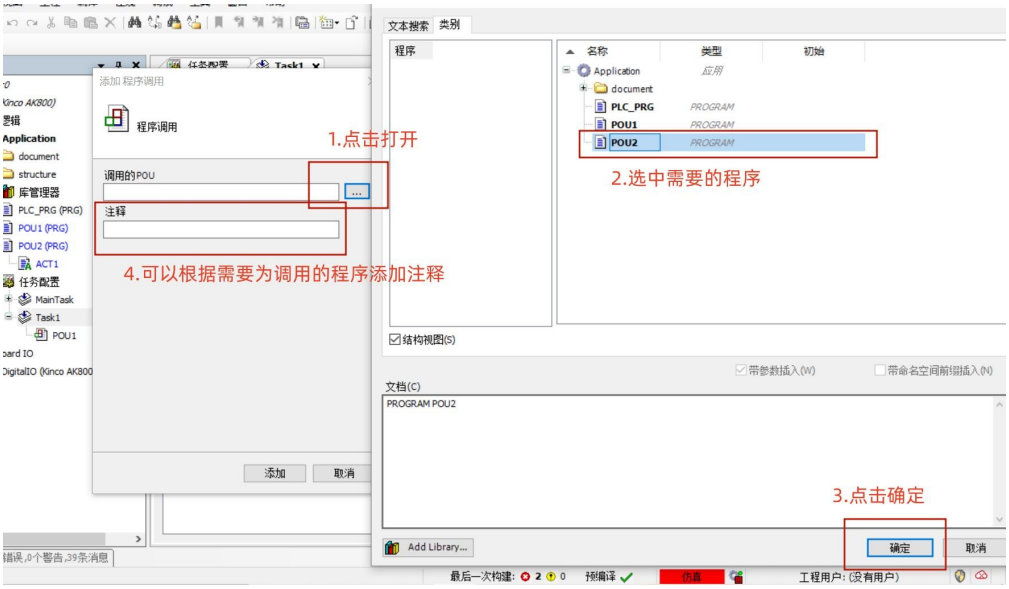


图 6.2-7：为任务配置程序调用操作-2

6.2.4 监控任务

双击任务配置菜单，在弹出窗口选择监视，即可监视各任务执行周期。

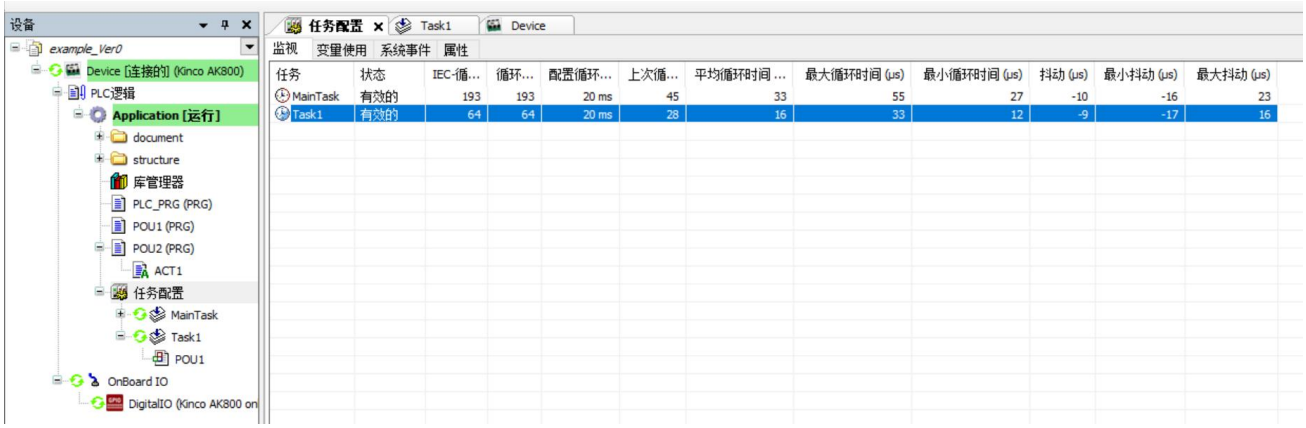


图 6.2-8：任务在线监视

CoDeSys支持多任务，因此控制器工程项目中，应该尽可能以任务来驱动，实时性要求比较高的选择自由循环类型任务，实时性要求不高的选择定时循环任务，由事件触发的任务可以选择事件触发任务。由系统事件触发的任务可以在系统事件中选择需要触发执行的任务。

任务配置中定义的任务执行顺序遵循以下原则：

- 条件已经到达任务将会执行；
- 如果同时有几个任务都达到了执行条件，那么将按照优先级按顺序执行。
- 如果同时有几个任务都达到了执行条件并有相等的优先级，则按照等待时间长短先后执行，等待时间最长优先执行。
- 程序调用处理会按照他们在任务编辑器中的顺序（从上到下）。

任务的个数是有限制的，任务配置的属性如下图所示：

图 6.2-9：任务配置容量

6.3 任务示例

对于一个工程，可以根据需要配置多个任务，调用不同的程序,可以满足不同的应用，如下举例说明两个具体应用

6.3.1 系统事件调用

一个工程中，往往需要在初次上电后对某些变量进行仅此一次的初始化或完成一些初始化动作，这个时候可以使用“StartDone”系统事件来完成，具体设置如下图所示：

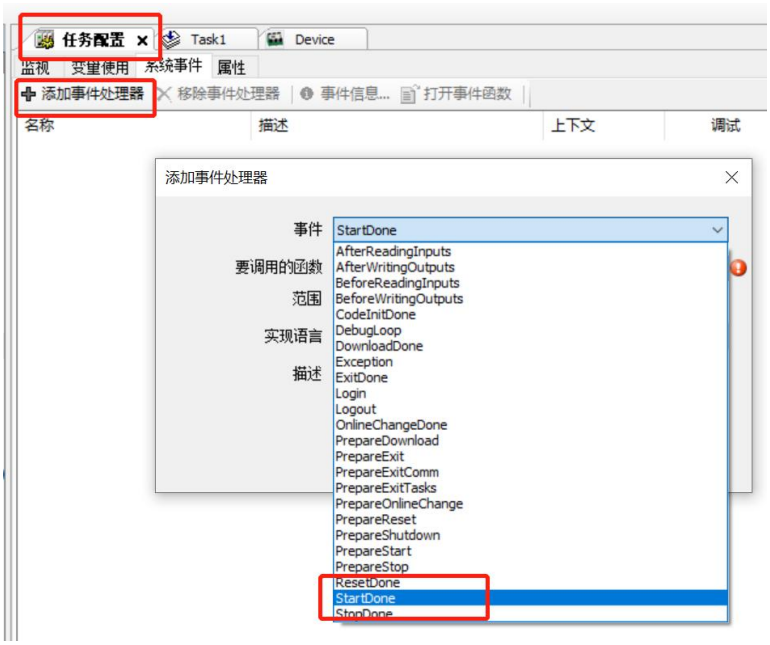


图 6.3-1：系统事件调用示例-1

6.3.2 IO 中断

实际应用中，有时会要求一些通过一些 IO 调用一个优先级最高的任务来完成动作，从而不受正在普通循环任务的扫描周期影响，也就是 IO 中断功能。具体 IO 可以是控制器本体的数字量输入输出，也可以使用程序当中的定义的 BOOL 型变量，还可以利用任务里的事件任务的来完成，此处以使用本体输入做 IO 中断为例做演示：

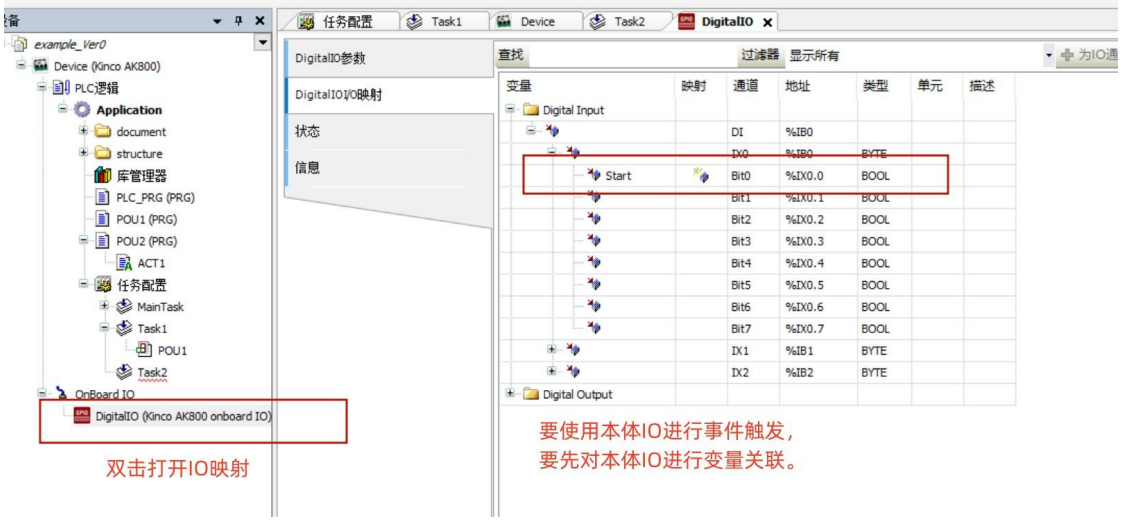


图 6.3-2：系统事件调用示例-2

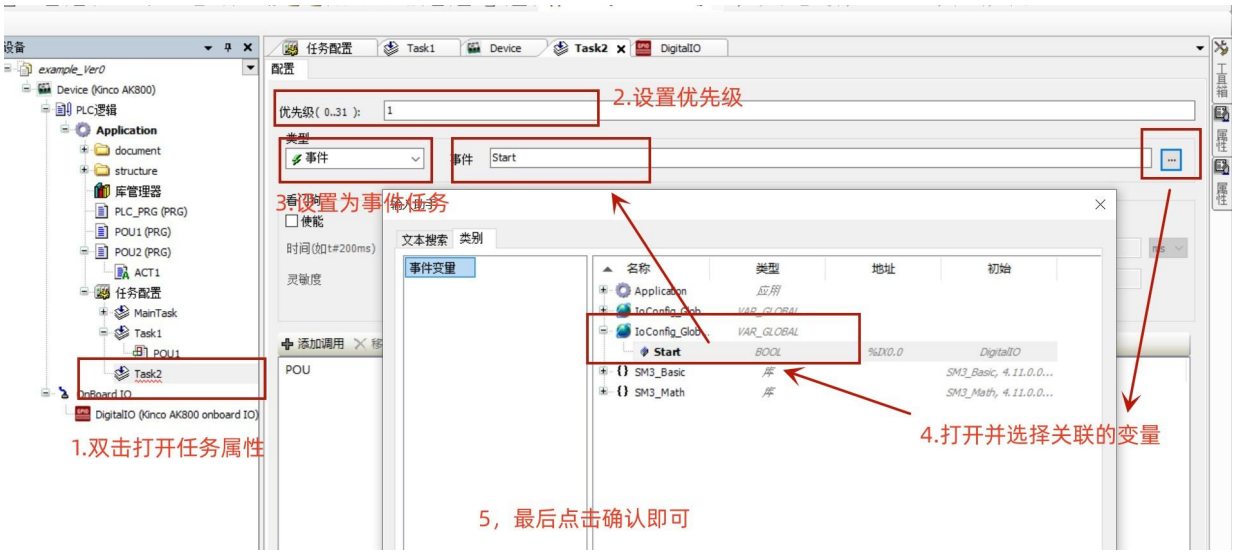


图 6.3-3：系统事件调用示例-3

第七章 创建和管理工程

在介绍了 CoDeSys 软件的编程环境之后，本章将介绍如何创建一个工程。工程包含控制器程序中的所有对象，包括 POU、数据类型、资源和算法等。创建一个新工程的顺序是可以灵活的，基本步骤主要包括目标设置、创建主程序、硬件配置和保存工程等。

7.1 目标设置

在打开 CodeSys 软件后，点击“文件”菜单下的“新建工程”（或直接点击工具栏上的 图标），在弹出的对话框中输入工程的名称以及工程保存的位置。

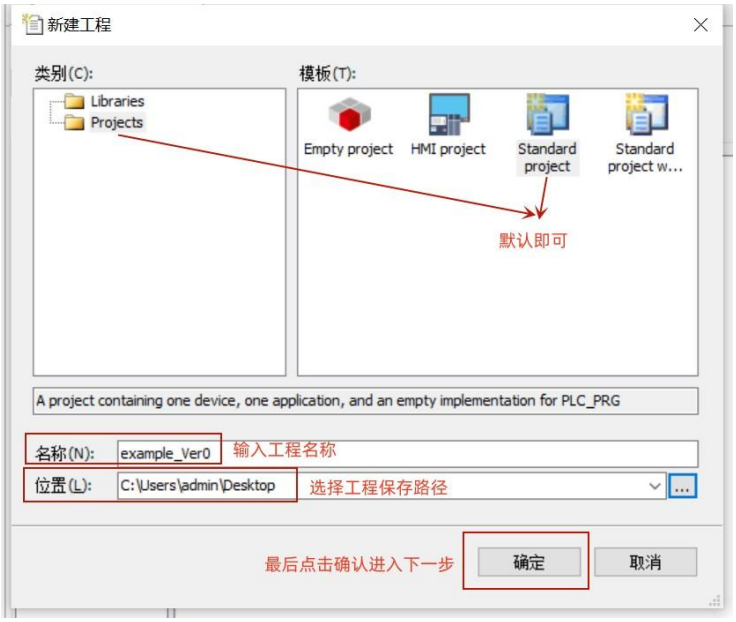


图 7.1-1：建立新工程操作-1

选择对应的 Kinco AK800 设备，选择一种“PLC_PRG”编程语言。

图 7.1-2：建立新工程操作-2

如图为新建工程默认的界面设备，导航栏包含：设备基本设置、库管理器、主程序、硬件接口等。

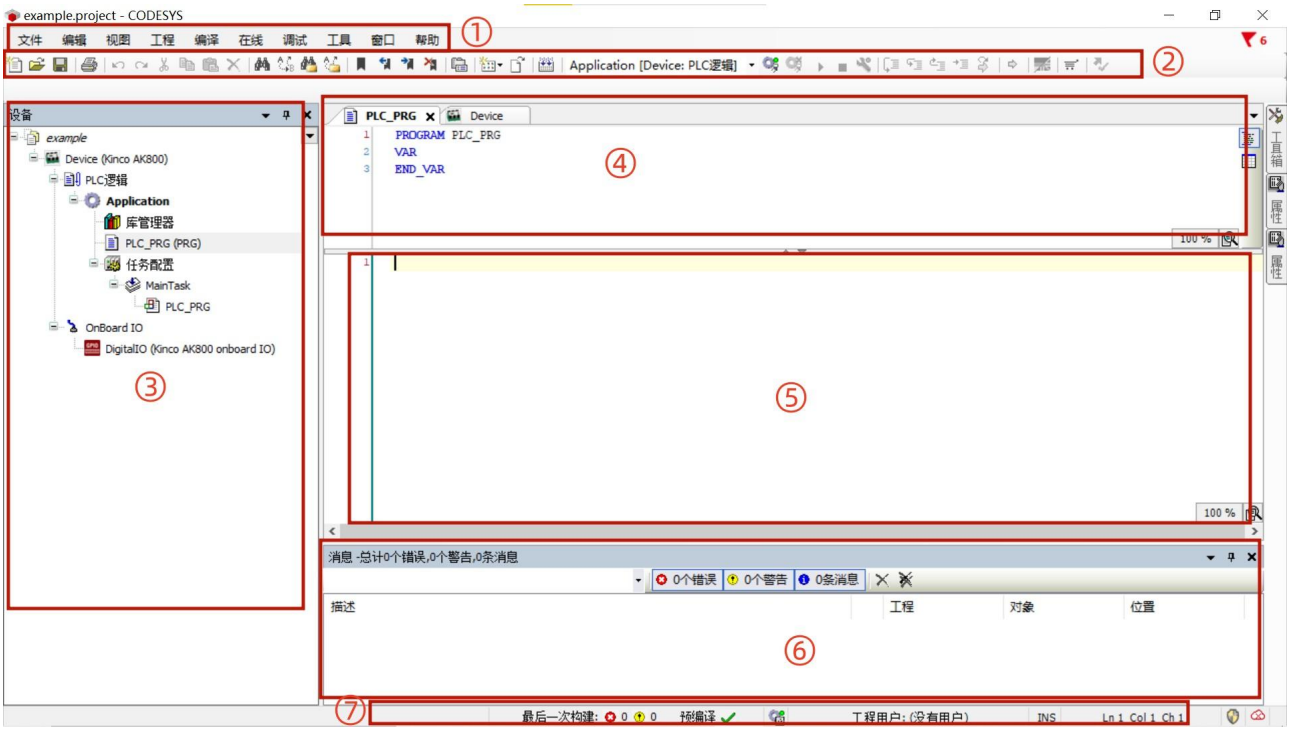


图 7.1-3：编程界面介绍

①：菜单栏　　②：工具栏　　③设备窗口　　④变量定义区　　⑤编程区

⑥：消息区　　⑦：状态显示区

7.2 创建任务

每一个工程都必须至少包含一个任务，任务中至少调用一个程序。系统默认建立一个“MainTask”任务，在这个任务中默认调用一个“PLC_PRG”主程序。如需修改任务和调用的程序设置，参考第六章。

7.3 硬件配置

AK800控制器本体硬件包含数字量IO、以太网接口、EtherCAT通讯口、CANopen通讯口、串口，用户可根据不同需求分别使用这些接口。

7.3.1 数字量 IO 口配置

AK800目标文件包内已包含硬件接口，IO接口等。双击设备树中“OnBoardIO”下的“DigitalIO”选项，可对IO变量进行映射，如下图，点击红框内小方框链接映射到已建内部程序变量，也直接在红框位置双击直接输入一个新变量的名字创建一个新变量名的映射。

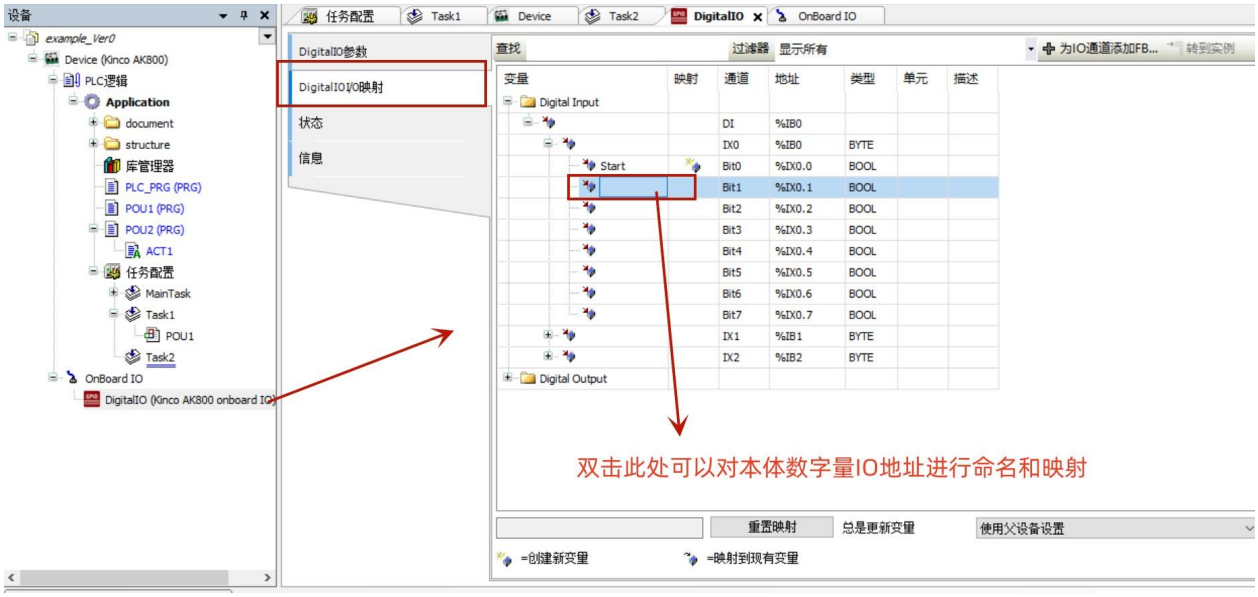


图 7.3-1：本体 IO 属性配置

7.3.2 CANopen 接口配置

控制器提供两路 CAN 接口，来控制 CANopen 从站设备，软件硬件接口需要进行相关配置，硬件接口设置可参考 AK800 的硬件手册。

7.3.3 EtherNet 接口配置

控制器提供 1 路 EtherNet 接口，可用来进行程序的上下载，此部分硬件接口设置内容参考第八章相关描述。该接口也可用来做为以太网通讯接口（支持 MODBUS TCP/UDP 协议等）来连接外部设备。硬件接口设置可参考 AK800 的硬件手册。

7.3.4 串行通讯接口配置

控制器提供 3 路串行通讯接口，可用来做为 RS232/RS485 串行接口（支持 MODBUS RTU 协议等）来连接外部设备。硬件接口设置可参考 AK800 的硬件手册。

7.3.5 EtherCAT 接口配置

控制器提供 1 路 EtherCAT 接口，来控制 EtherCAT 从站设备，软件硬件接口需要进行相关配置，硬件接口设置可参考 AK800 的硬件手册。

7.4 程序编写

在完成控制器配置后，可以开始进行程序编写。在新建程序时，可以选择程序的编程语言，包括 IL、LD、ST、SFC、FBD、CFC 等。POU 的所有编辑器包括声明部分和主体部分，它们

被屏幕分割器隔开。

ST语言编辑器是一个文本形式的编辑器，这里以常用的ST语言为例，介绍CoDesys编程的规范。其他语言的编程规范，请参见第九章。

7.4.1 变量及自定义数据类型的管理

变量是程序当中一个不可避免的重要概念，下方分别介绍了如何新建全局变量列表、掉电保持型变量列表、永久保持型变量列表、自定义结构体、局部变量列表等，变量的定义及类型等详情请参考第四章相关说明。

在进行变量声明讲解之前，需要针对CoDeSys平台的软件特性进行以下两点说明：

1.变量可以事先声明（便于管理增加程序可读性），也可在编程时通过自动弹出的“自动声明”窗口进行变量声明，但“自动声明”窗口只出现一次，此后只能手动进行定义。

2.自定义数据类型无法进行自动声明，只能手动进行定义。

- 全局变量列表

1.新建全局变量表：右击“Application”，选择“添加对象”目录下的“全局变量表”，在弹出的命名框中可以自定义该变量表的名称，点击确认即可完成创建，如下图所示。

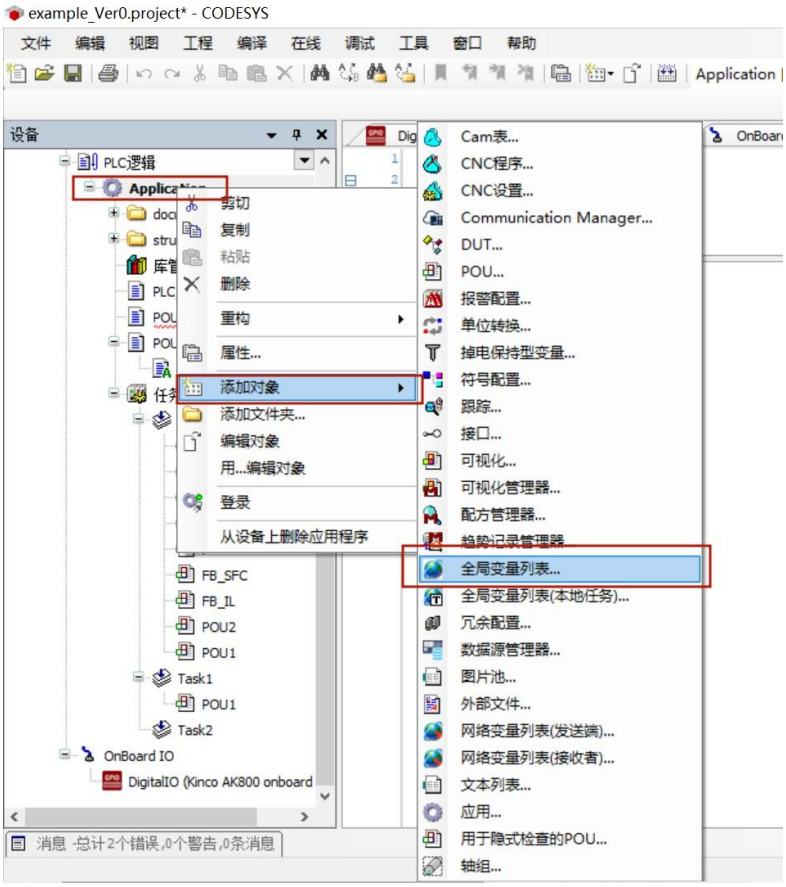


图 7.4-1：建立全局变量表操作

2.全局变量表的默认界面如下图所示：

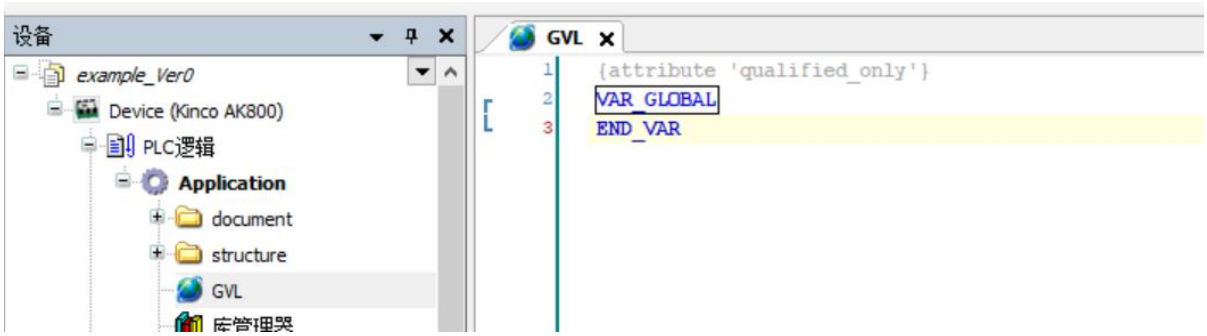


图 7.4-2：全局变量表界面

全局变量只需要在标识符“VAR_GLOBAL”和“END_VAR”之间添加即可，变量的声明方式与局部变量的声明方式一致。

关于“qualified_only”属性：

添加全局变量表后，系统会默认添加“qualified_only”属性，其效果是：使全局变量列表的变量只能通过指定全局变量表名称来寻址（以变量表名为前缀），即寻址方式为：<全局变量表>.<变量名称>，这样有助于避免被误认为是局部变量，而不完整的变量调用将产生错误。

关闭“qualified_only”属性后，全局变量的寻址也将通过变量名直接进行，但有可能与局部变量混淆，即全局变量的声明与局部变量的声明可以重复，且编译不会产生任何提示，在实际使用中，重复声明的变量被会误认为是局部变量。故此处建议用户使用默认开启的“qualified_only”属性。

- 局部变量表

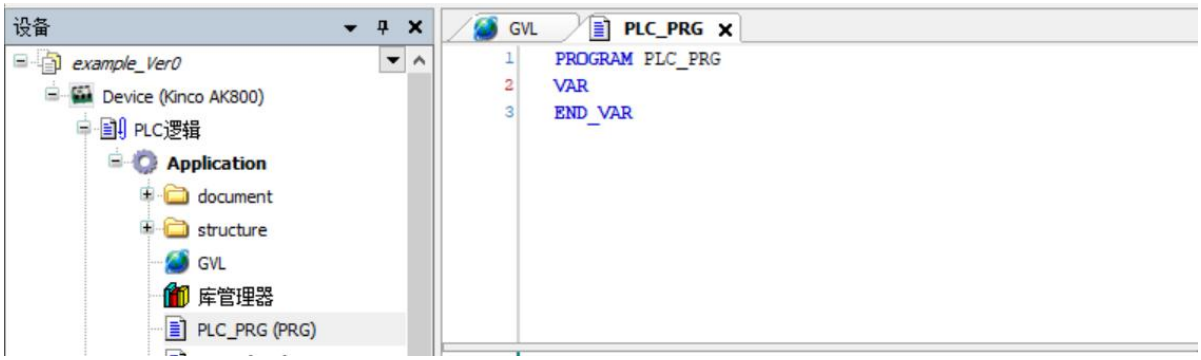


图 7.4-3：局部变量表界面

用户在新建程序后，出现的“变量定义区”即为局部变量的声明区域，局部变量的声明在标识符“VAR”和“END_VAR”之间进行。在此区间内声明的变量只能用于该“PROGRAM”（程序）。

- 保持型变量表

在全局变量表或局部变量表的第一个标识符后加“RETAIN”关键词，如下图所示：

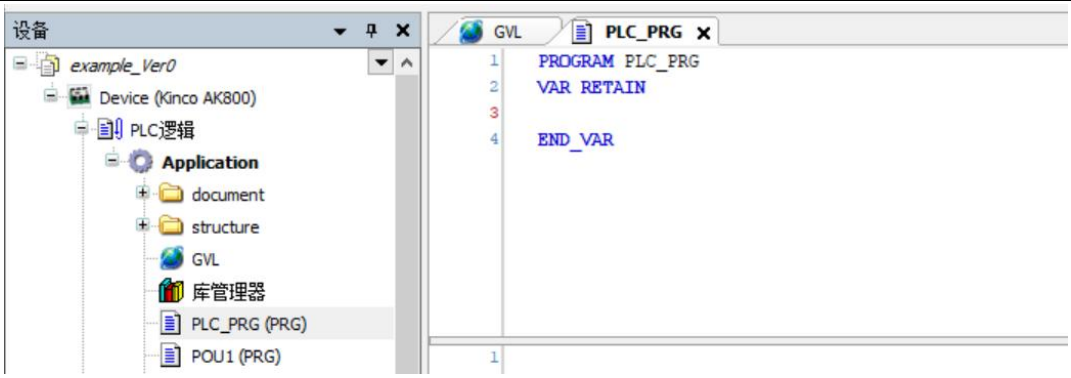


图 4.4-4：在局部变量中创建保持型变量

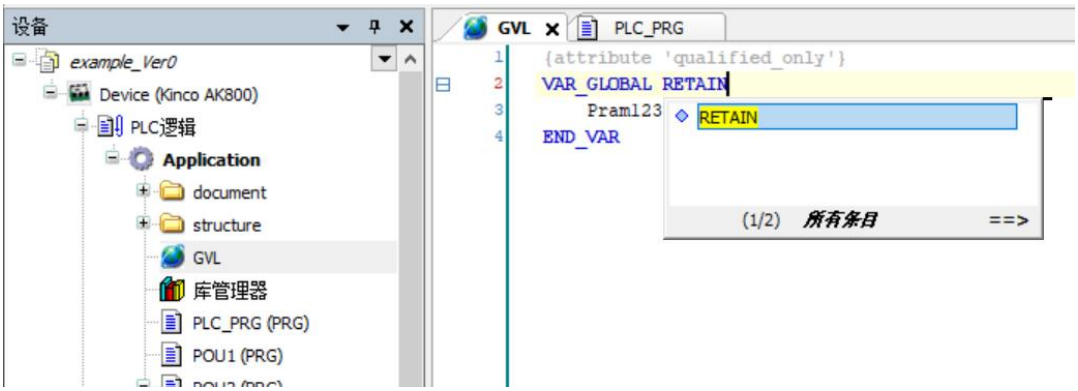


图 7.4-5：在全局变量中创建保持型变量

➢ 掉电保持型（永久）变量表

1.新建掉电保持型变量（永久型变量表）：右击“Application”，选择“新建对象”目录下的“掉电保持型变量”，在弹出的对话框中可以自定义变量表名称，点击确定即可完成创建。

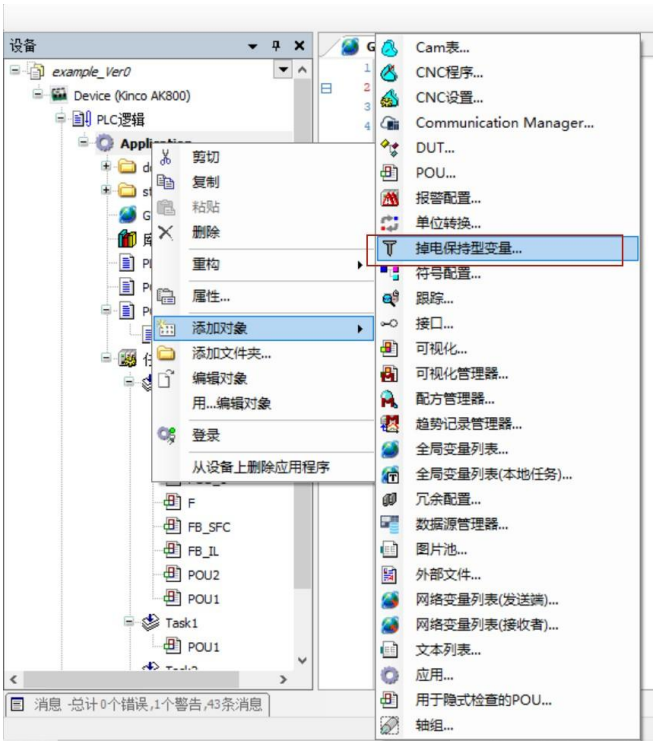


图 7.4-6：建立掉电保持型（永久）变量表操作

2.掉电保持（永久）型变量的界面如下图所示：

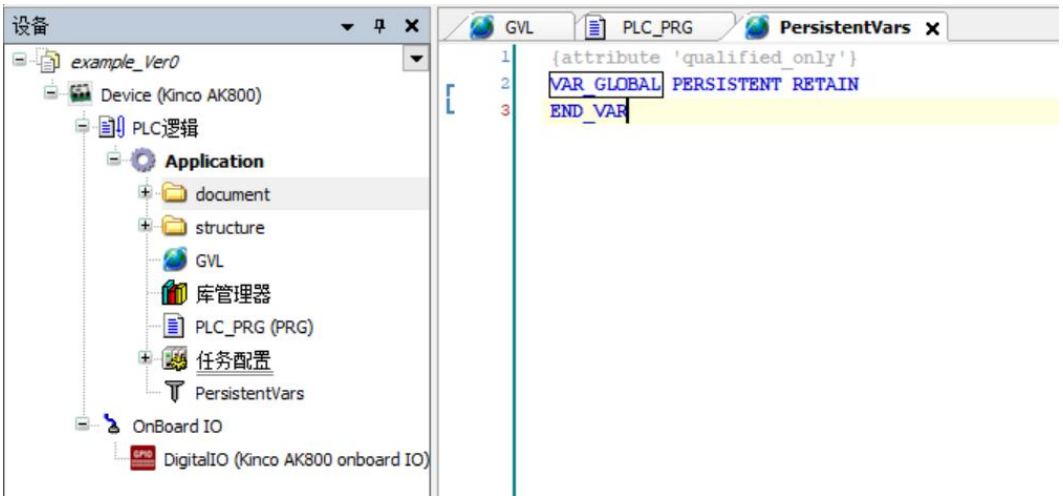


图 7.4-7：掉电保持型（永久）变量表界面

新建掉电保持型变量表也会默认开启“qualified only”属性，其声明方式与全局变量的声明方式一致，变量在“VAR_GLOBAL PERSISTENT RETAIN”和“END_VAR”之间声明。保持变量和永久型变量的区别请参照第四章说明。

- 自定义数据类型

1.新建“STRUCT”（结构体）

如图，先新建一个DUT，在弹出的对话框中选择类型为“结构体”（STRUCT），输入结构体的名称即可完成创建，此后，用户可以在生成的结构体中自定义需要的对象。

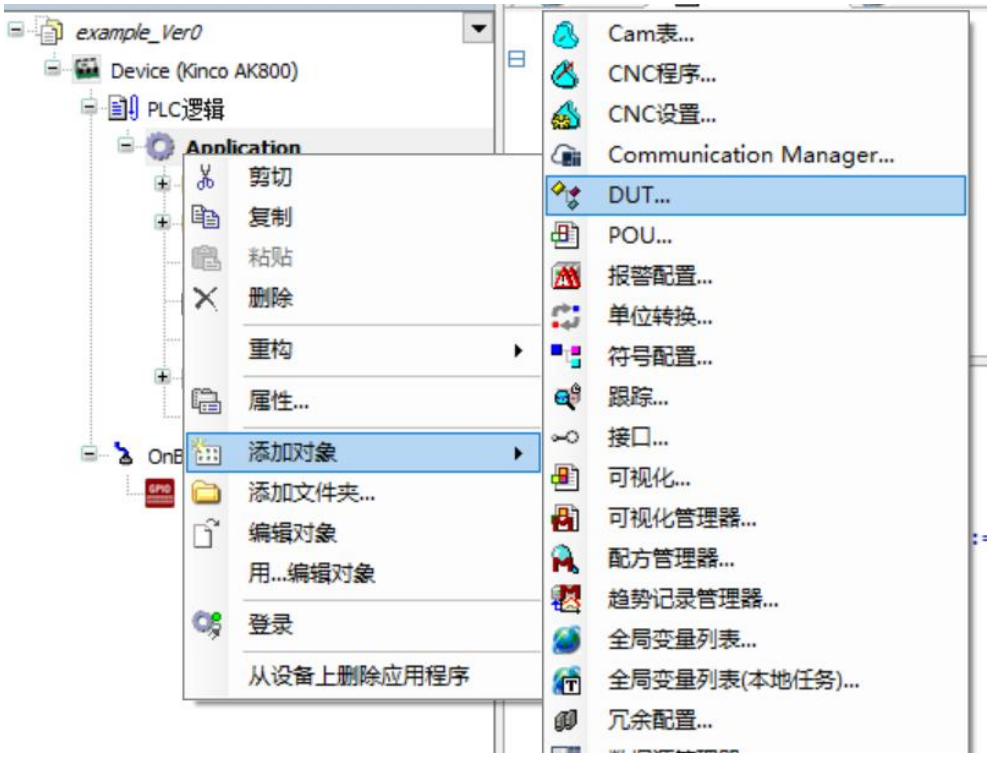


图 7.4-8：自定义数据类型操作

2.新建得到的数据类型界面如下图所示，自定义的数据的声明在“STRUCT”和“END_STRUCT”之间进行。创建完成之后，可以作为自定义的数据类型进行调用声明。

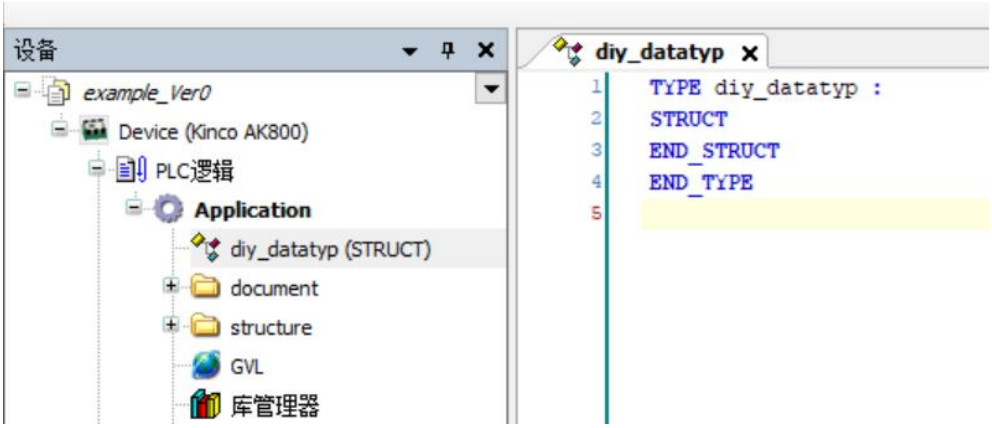


图 7.4-9：自定义数据类型编辑界面

➢ 常变量表

常变量表的定义以“CONSTANT”为标识符，常变量的声明通常直接赋值，否则编译后会出现常变量未赋值的警告。

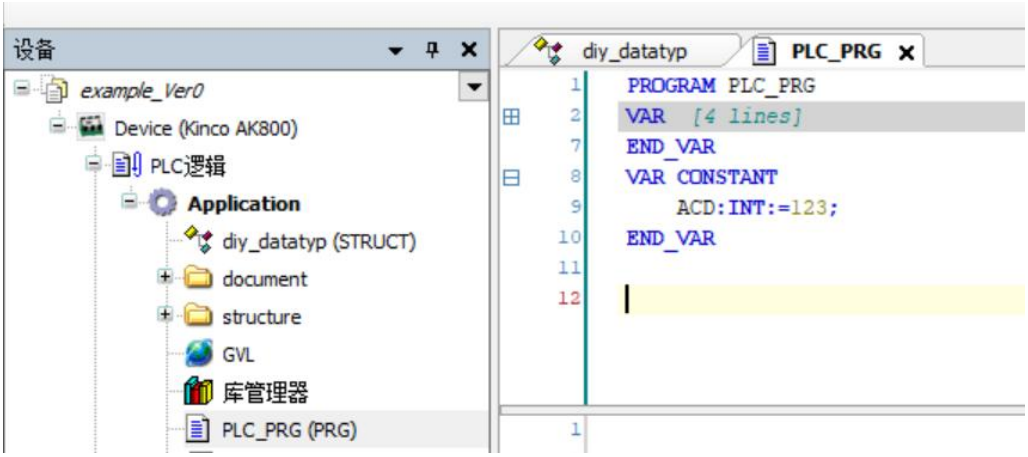


图 7.4-10：建立常变量的操作

7.4.2 POU 管理

创建不同 POU（程序，功能块，函数等）请参考 5.2 章节。

7.4.3 指令输入

在ST编程环境下，CoDeSys 的不同的对象的指令输入方式如下：

1.常量的输入：

在 CoDeSys 中，直接以数值形式的输入默认为十进制数，若要改变输入的进制，可以加前缀“进制名#”，如十六进制：16#0123，二进制：2#1000。此外，CoDeSys 中具有多种类型常量，其具体的表达方式可参照第 4.3 章。

2.实际物理地址的输入输出

下图是AK800本体IO的输入输出实际物理地址（自动分配）。在实际程序中，实际地址可以作为一个BYTE类型使用（%IB、%QB），也可以作为多个BOOL类型使用（%IX、%QX）。

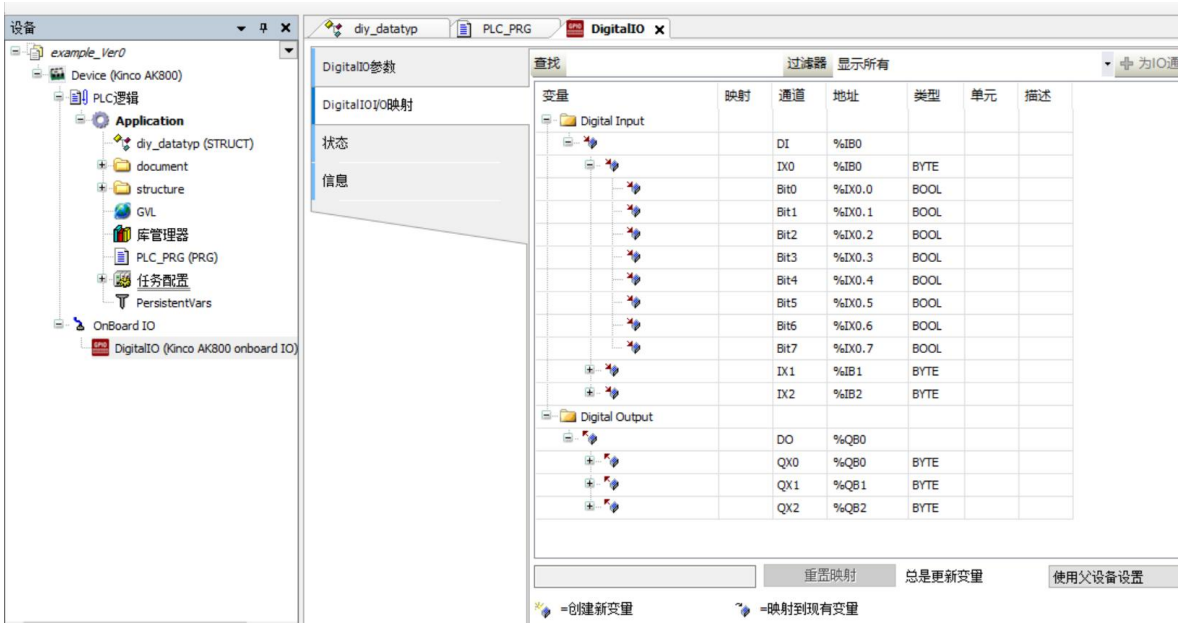


图 7.4-11：AK800 本体 IO 的实际物理地址表

3.变量的输入

程序中变量直接以其变量名称调用，开启了“qualified_only”属性的全局/保持变量需要配合全局表量表名称前缀进行调用。在“输入助手”（F2）中可以查看到工程中已建立的全部变量。

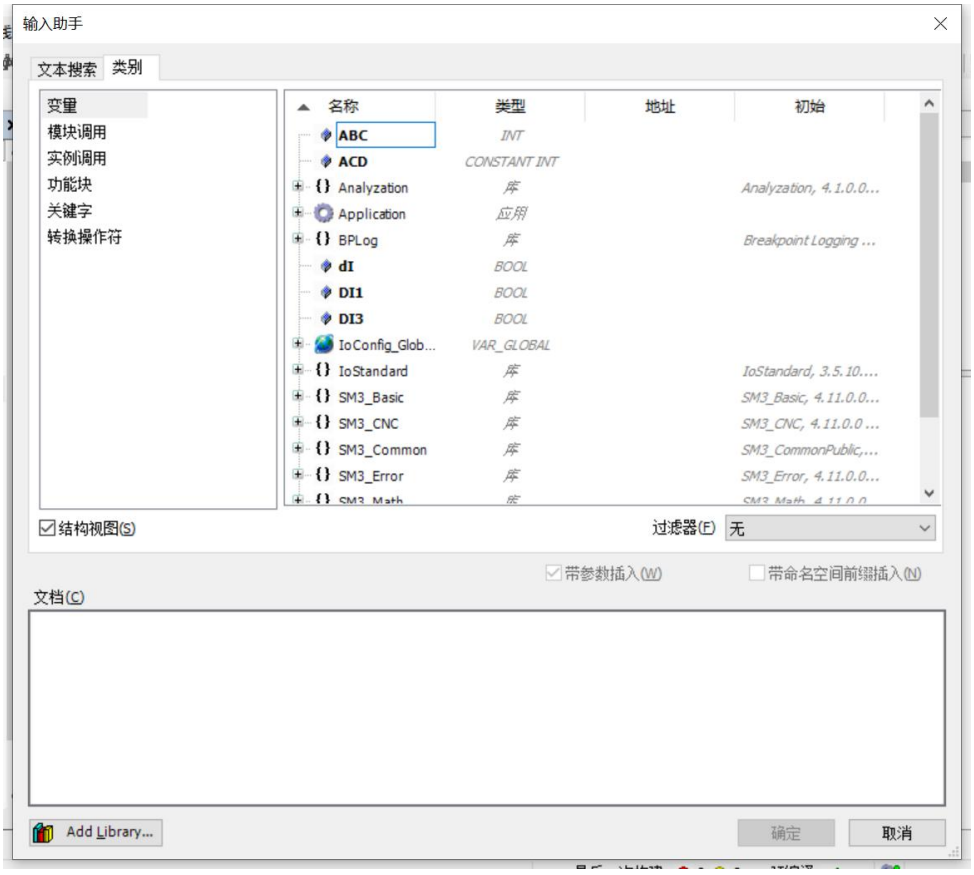


图 7.4-12：从输入助手查看变量

该操作需要确保“输入助手”的“显示变量”已开启。可进入工具菜单下的“选项”

中进行设置更改。参照下图所示路径。

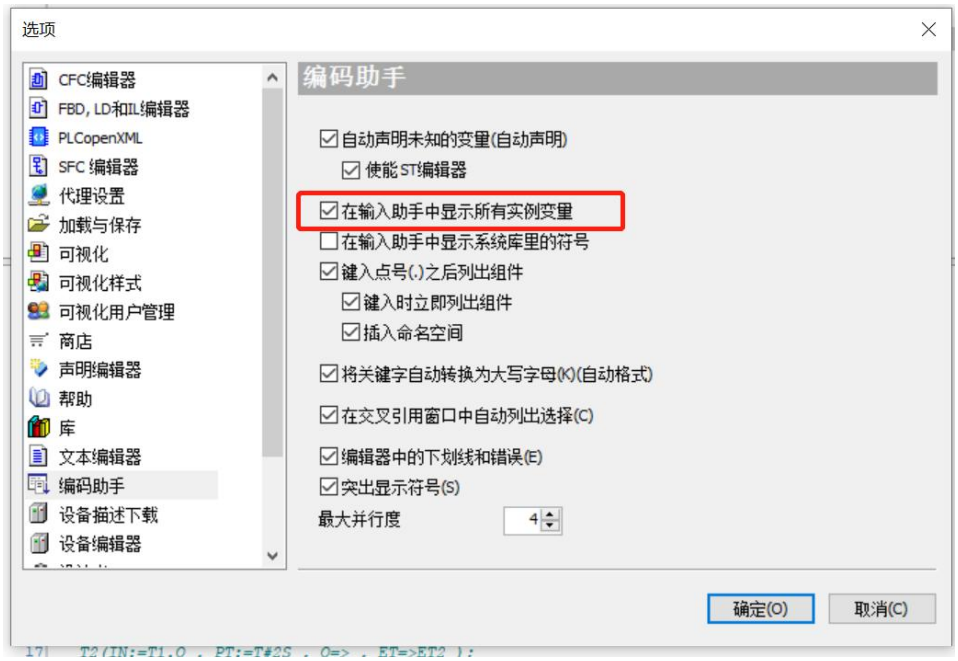


图 7.4-13：编码助手设置

4.IEC标准指令的输入

在Codesys的指令系统中，一些标准指令，如初等运算指令、比较指令、移位指令、赋值指令、类型转换指令、逻辑运算指令等，都应以运算符形式调用。可直接输入，也可以在输入助手的“关键字”中选择输入。与IEC编程语法有关的部分请参考第九章。

图 7.4-14：在输入助手查看 ICE 标准指令关键字

5.程序流程控制语句的输入

在实际编程中，需要对程序进行流程逻辑控制，以便实现复杂功能。常用的逻辑控制有：IF..ELSE\IF..ELSIF\WHILE\FOR等逻辑语句。可手动直接输入，也可以打开输入助手（F2）选择输入。

```
IF M0 THEN
    FOR i:=0 TO 10 BY 1 DO
        A:=A+1;
    END_FOR
END_IF
```

图 7.4-15：逻辑控制示范（IF/FOR）

图 7.4-16：在输入助手查看标准 ICE 逻辑控制关键字

6.功能调用

在程序和功能块中可以调用各种函数实现某些功能，可直接输入，也可以按F2打开输入助手选择输入，下图示范调用一个标准函数：

```
Cop_SetOperational(wCanNum:=0 , uiNodeID:=1 , pbyState:=BYTE1 );
```

图 7.4-17：调用标准函数示范

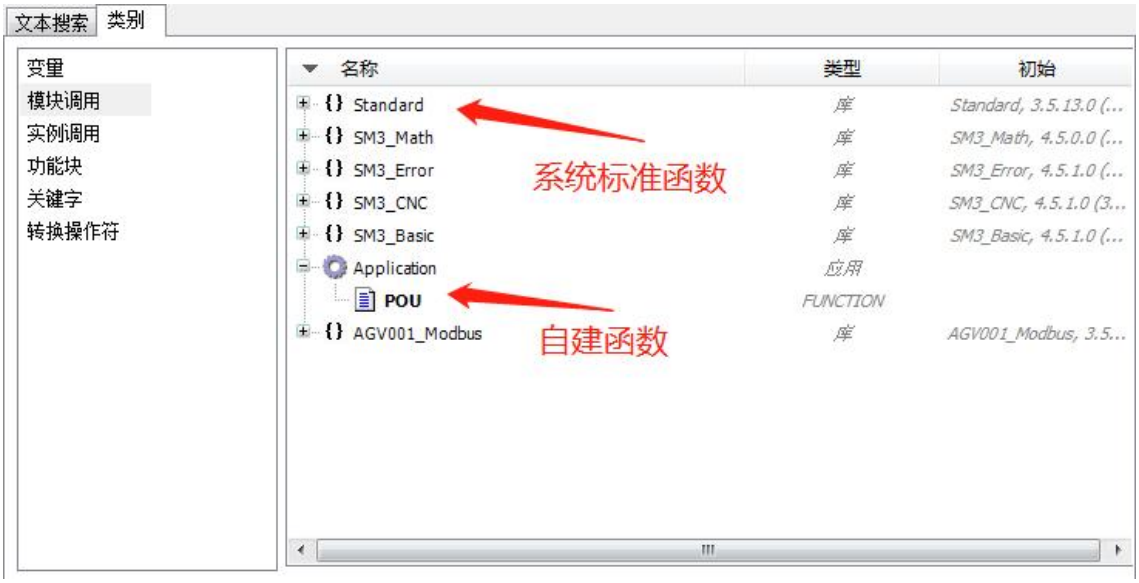


图 7.4-18：在输入助手中查找所有函数/功能

7.功能块调用

定时器、计数器、边沿触发器、通讯、高速输入输出、模拟量处理指令等标准指令，或一些由用户自定义的功能块，都应采用功能块的形式调用。功能块的调用必须实例化。详情参考5.3章节。功能块的指令输入可以直接输入也可以打开输入助手的“功能块”选项选择输入。

下图示范了一个标准接通延时定时器TON功能块的调用，T2是其实例化后的标识符。

```
T2(IN:=T1.Q , PT:=T#2S , Q=> , ET=>ET2 );//2S定时器
```

图 7.4-19：定时器功能块调用示范

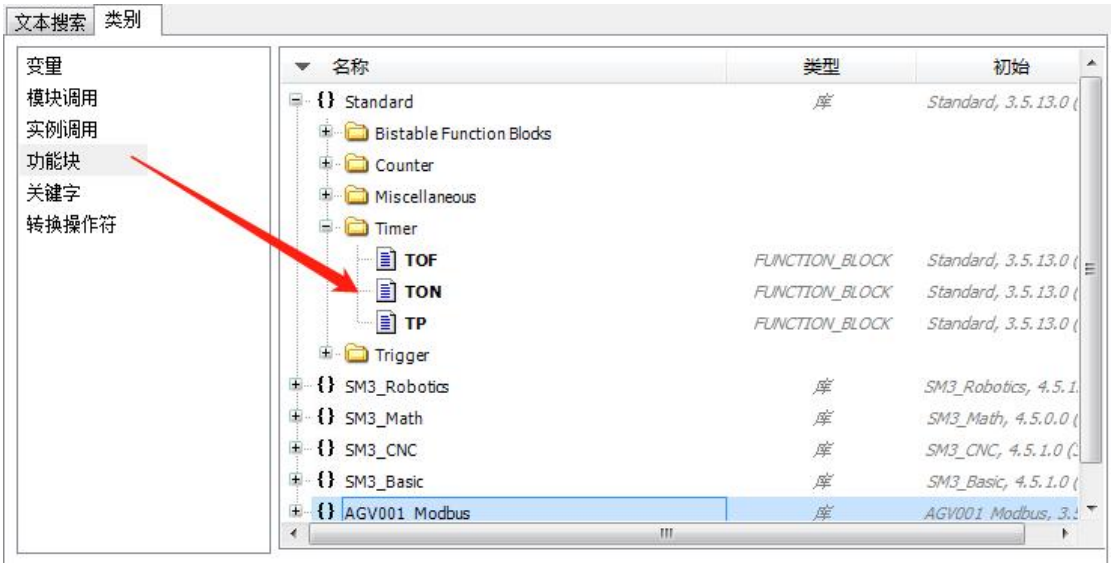


图 7.4-20：在输入助手中查找功能块

7.4.4 库管理

1.工程库管理器

对于一些常用指令或者功能块，如字符串处理指令、触发器功能块、计数器功能块、PID控制功能块等，软件平台或者部分用户会将其分类集合，然后建立专门的标准或专用库，方便在CoDeSys编程中导入到库管理器后直接调用。库是指令与功能块的集合，库文件的格式为“.library”（包含指令和功能块的输入输出代码），只需在工程的库管理器中导入相应的库文件，即可调用指令和功能块。工程的库管理器只能对当前工程的库进行管理（添加/删除/修改库等操作），并会随着程序下载到控制器当中（占用内存空间）。

标准库管理器窗口分成三个区域，如下图所示。绑定到工程的库列表位于左上区域，左下区域显示所选择库的程序、数据类型或全局变量。选择一个函数或功能块后，其变量声明、函数或功能块的图形显示以及文档说明显示在右下区域。

学会查看函数库是非常重要的。在函数库中，给出了函数的相关信息，例如，该函数中有几个输入变量及中间变量，它们各自的数据类型是什么，有哪些中间变量是必须赋予初始值的以及变量的注释等等。

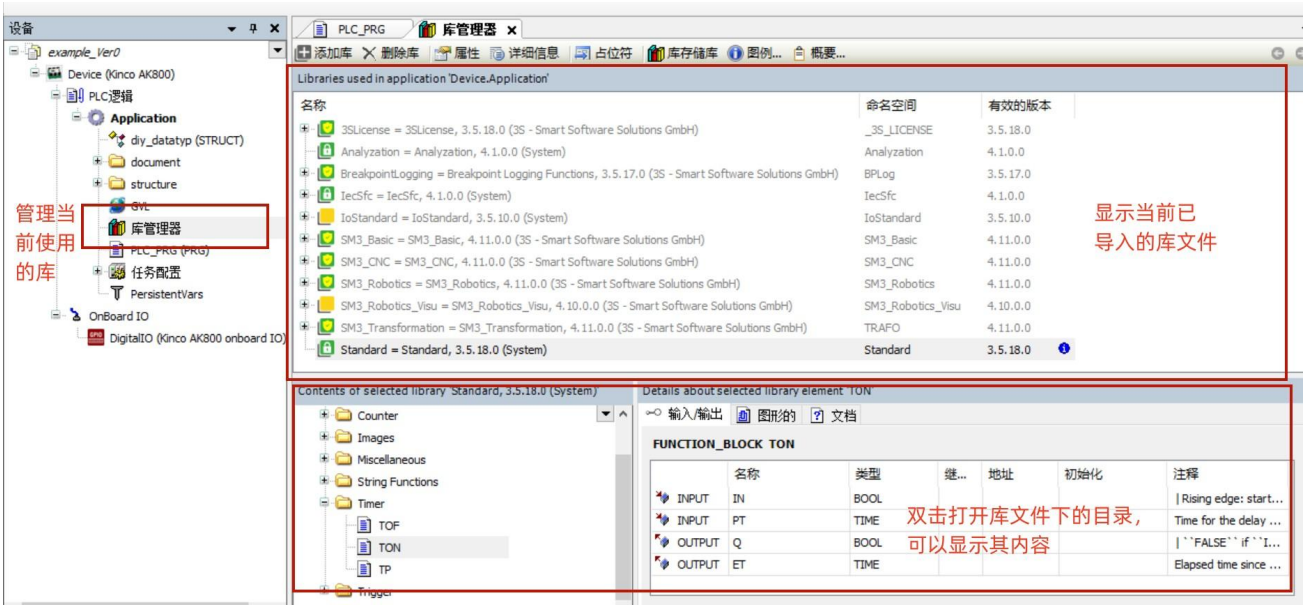


图 7.4-21：工程的库管理器界面

2.库存储器（库资源管理器）

库在导入到工程库管理器前，需要先被装载到库存储器中。库存储器用来管理库函数和功能块，包含了系统提供的所有标准函数和功能块以及曾导入过的其他库。选择“工具”菜单下的“库存储”选项可打开库管理器，如图所示：

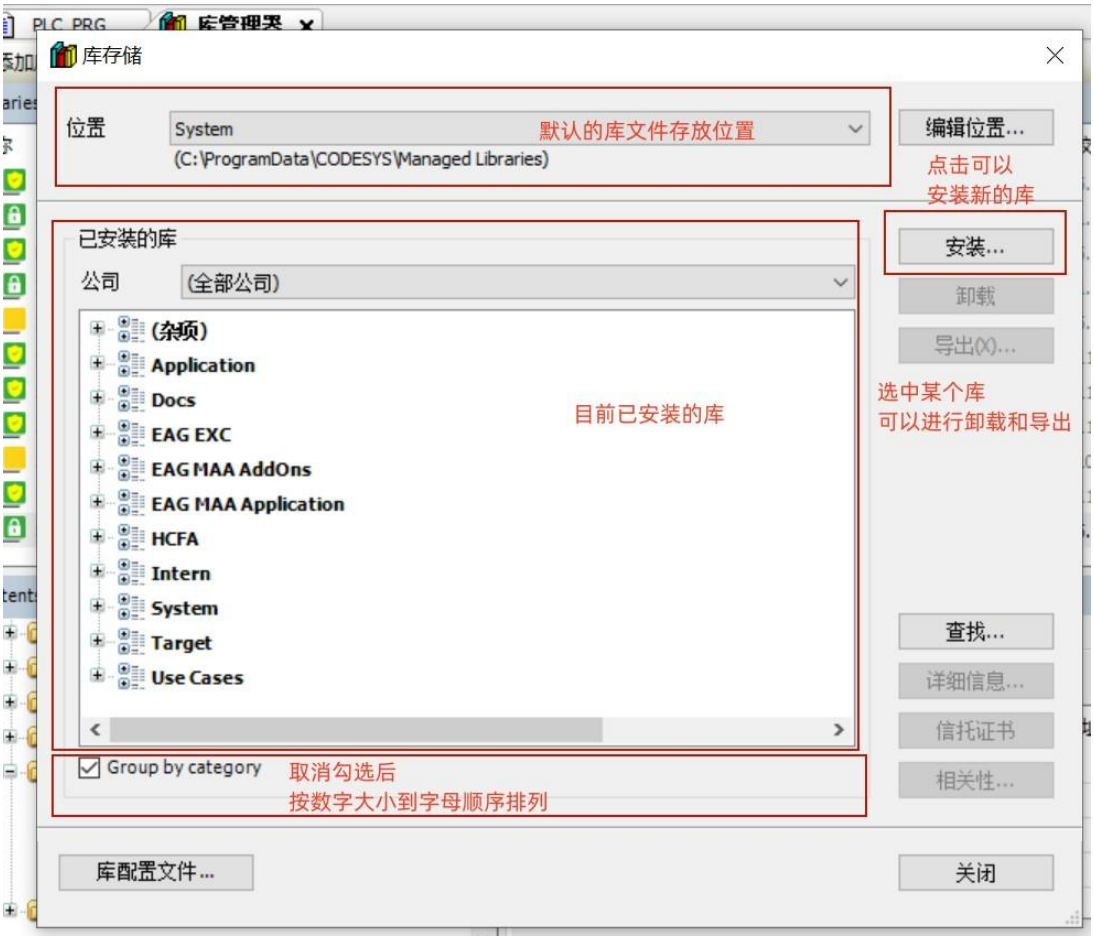


图 7.4-22：系统的库存储器界面

如上图所示，对话框中间显示在整个Codesys软件下已经安装的库，并按类别分组显示，要查找已经安装的库文件，可选择关键词查找，也可以按照公司分类或者字母顺序一个个查找等。

与工程的库管理器不同，库存储器中被安装过的库可以被任意工程使用，它保存在软件系统的PC上不随工程下载（不占内存）。

3.将库安装到库存储

当目前的工程库列表中的库文件已不能满足目前的编程功能需求时，需要将集成了所需功能的库文件添加到工程中。在将一个库导入到工程的库管理器之前，请确保该库已被安装到库存储器中。单击“工具”菜单下的“库存储”选项，打开库存储器，选择“安装”，在弹出的对话框中找到所需要安装的库的存放位置，选中该库文件，点击“打开”后等待安装完成即可。

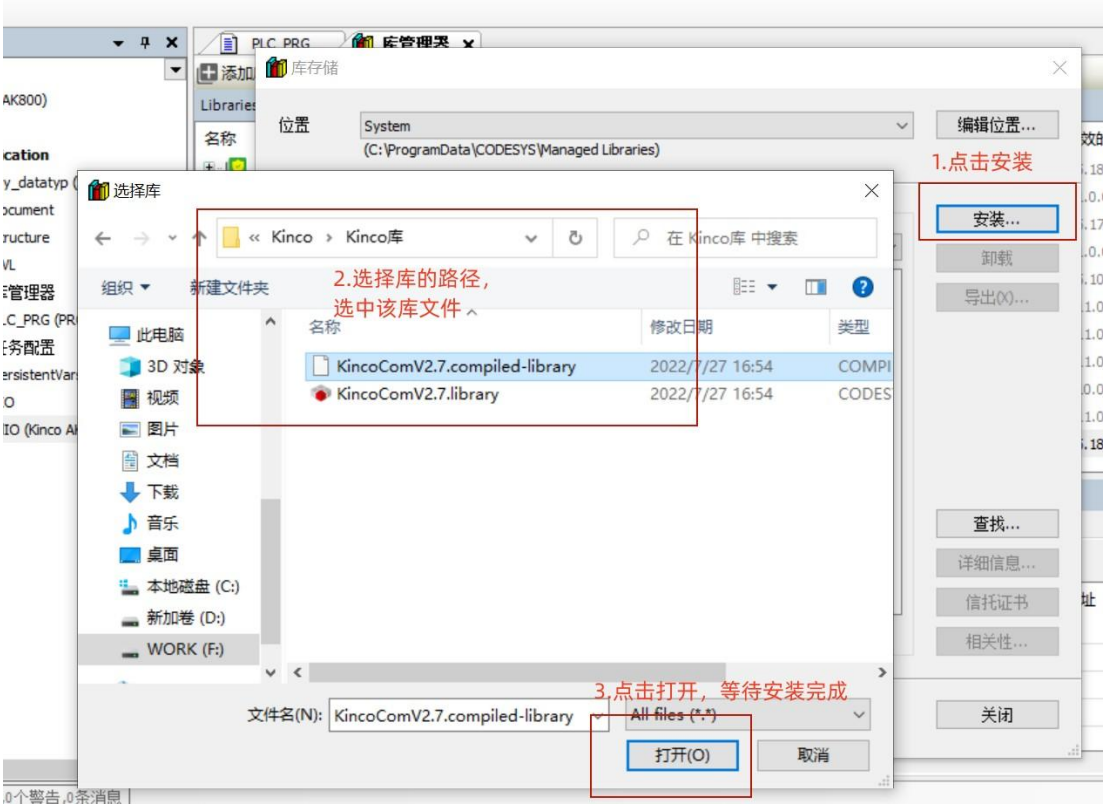


图 7.4-23：将库安装到系统库存储中

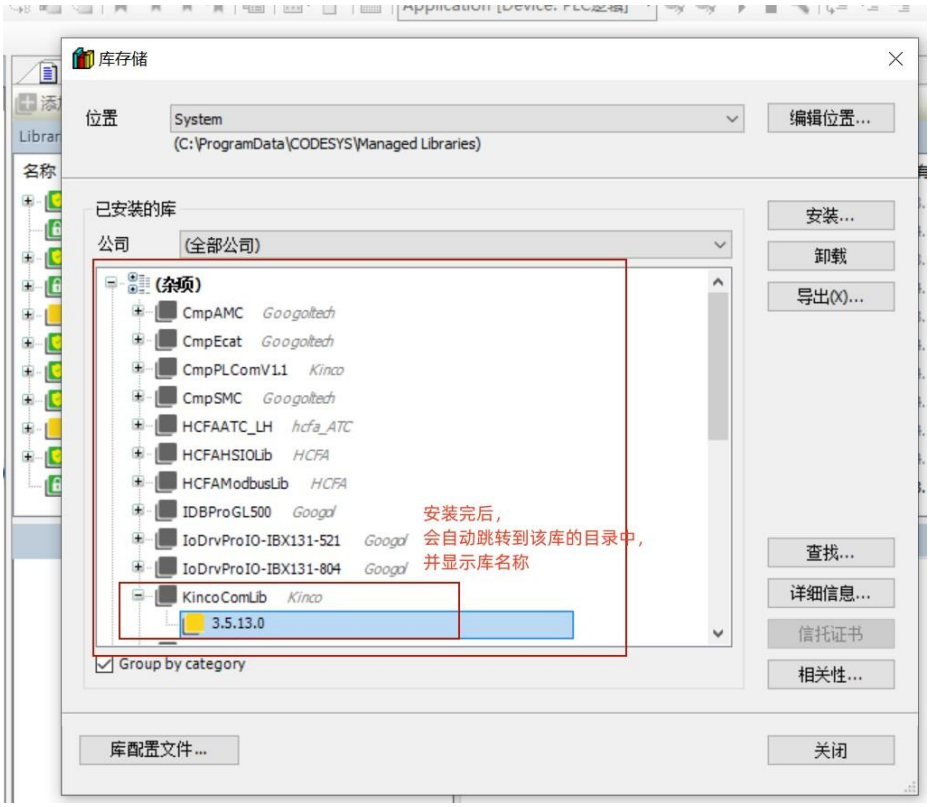


图 7.4-24：库存储界面显示已安装的库

4.将库导入到工程库管理

确保库已被装载到库存储中后，可以在库存储器中找到该库并将其添加到工程中。打开设备树中的“库管理器”，点击“添加库”按钮，在弹出的库存储目录中选中需要添加的库，点击确定即可。添加完成后，该库会在工程管理器库列表中显示。

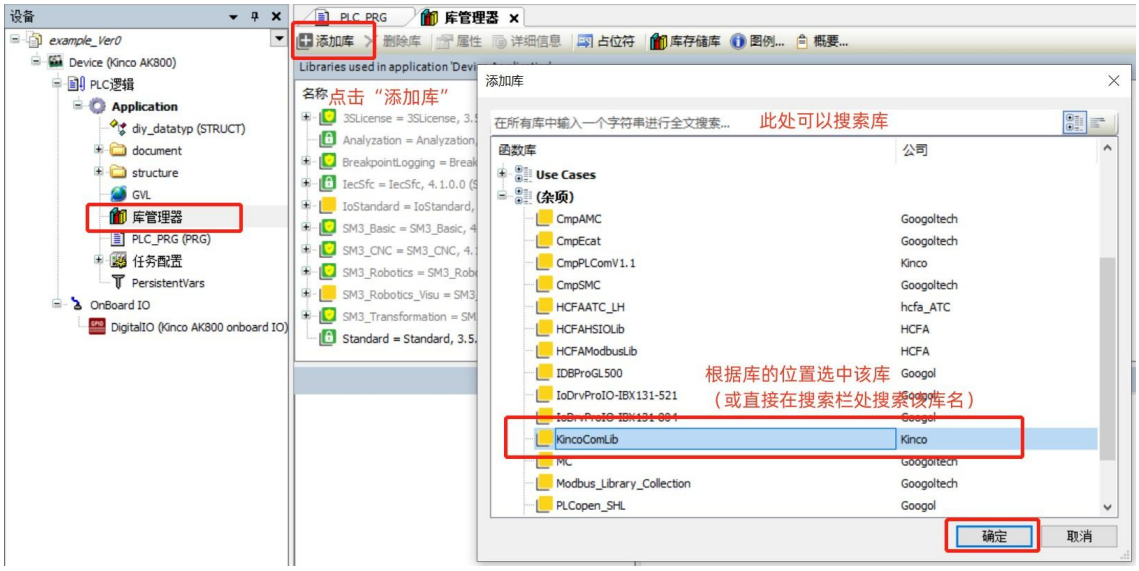


图 7.4-25：将库存储中的库导入到工程种

5.删除库

使用库管理器窗口右击需要删除的库名称，或选中需要删除的库后选择“删除库”，可以从工程库管理器中删除已添加的库。

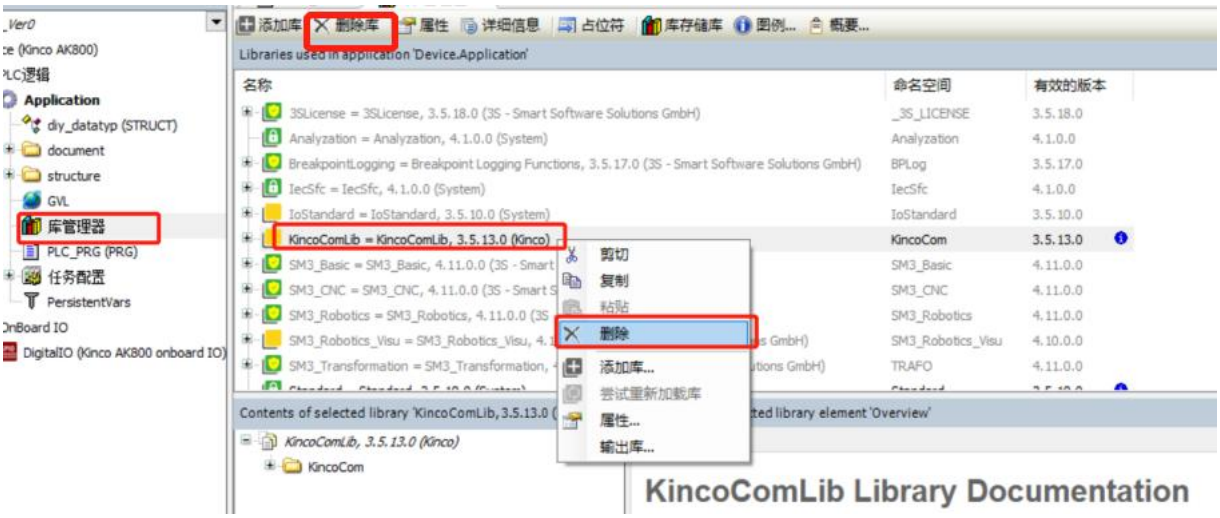


图 7.4-26：删除库操作

6.库版本升级

随着设备更新或者软件更新，有时候库版本也会要求进行对应的升级，此时会在添加库之后有蓝色下划线提示，同时上方出现可以在线联网升级库的提示。

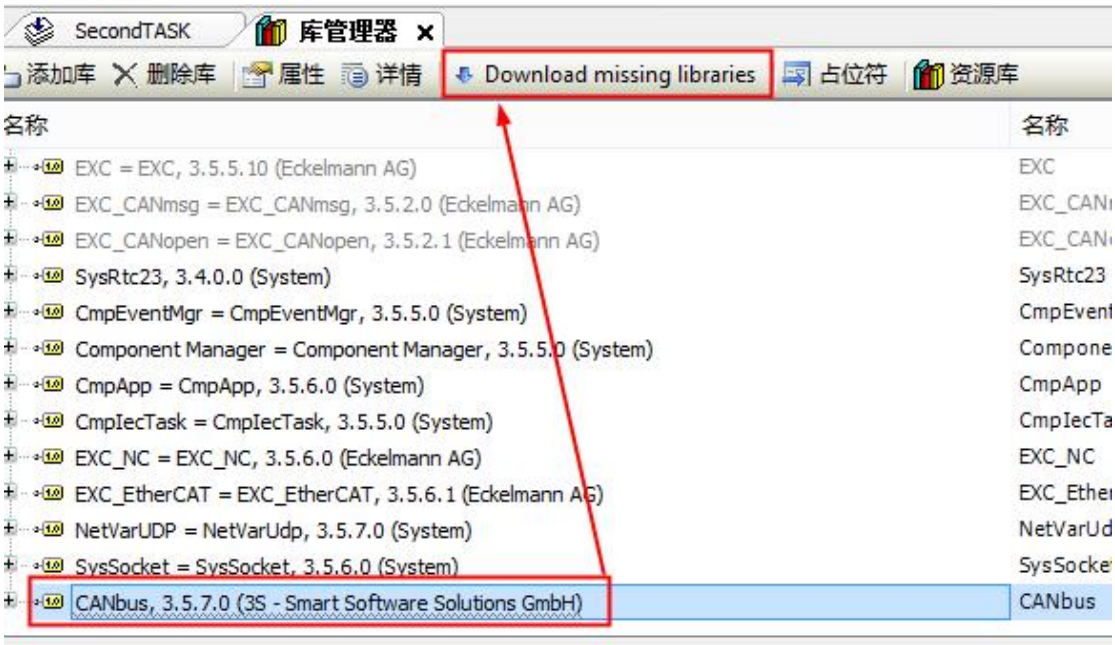


图 7.4-27：在线下载库文件操作

点击“Download missing libraries”在线升级库（该操作需联网）。

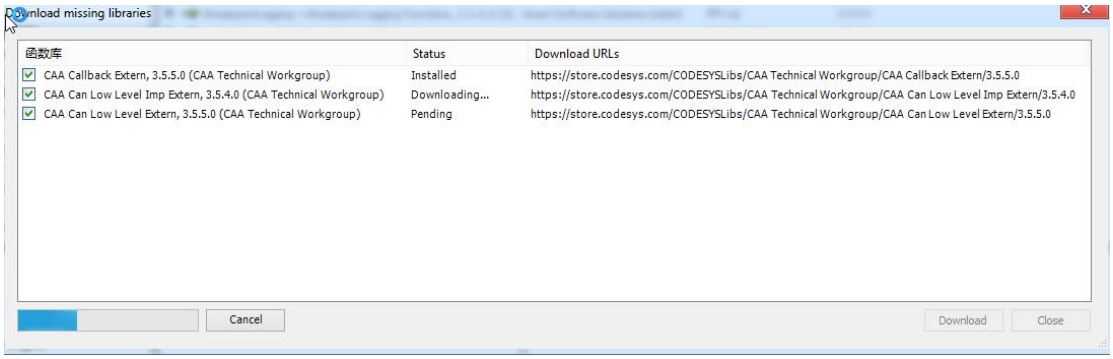


图 7.4-28：在线升级库文件操作

升级成功后，库文件的蓝色下划线提示和上方出现可以在线联网升级库的提示消失。在新建工程后，控制器会自动添加一些常用库，其余的库文件需要手动添加。注意，凡是添加到工程库管理器的库都会占用用户程序空间，所以建议用户只添加需要使用的库。常用的库会在第十章进行详细讲解。

7.4.5 库的制作

在工程实施中，如果有一些算法或逻辑是大量重复的，而且可以应用于多个工程中，则可以将这部分逻辑制作成库，以便于同一系列的程序编写、工程维护管理及技术资源共享。库文件可以加密，保护库的具体实现算法和逻辑不被查看和修改。以下将举例介绍制作库的过程：

本例是专为产生CRC16 校验而制作的一个ST 语言的自定义库，如图所示。编译通过后，就可以在其它的工程中使用此库。使用前注意在库管理器中先要加入此库。如果想要在其它的

计算机上使用此库，则只需要将此库文件复制到其它计算机的软件安装目录下，再添加到工程中即可。一个库中可以包括多个功能块或函数。如果想要添加功能块或函数，只需打开库文件，在程序中添加新的功能块即可。

注意，库中的功能块不允许递归调用，即不允许自己调用自己。

Step1：首先新建一个工程，目标设置中选择“libraries”，模板一般选择选择codesys library（内部库Internal Library），使算法逻辑在库文件中实现。还有一种是外部库（External Library），其算法逻辑在模块中实现，库文件中只定义使用的变量。下方给库文件命名并选择存放位置，如图所示。

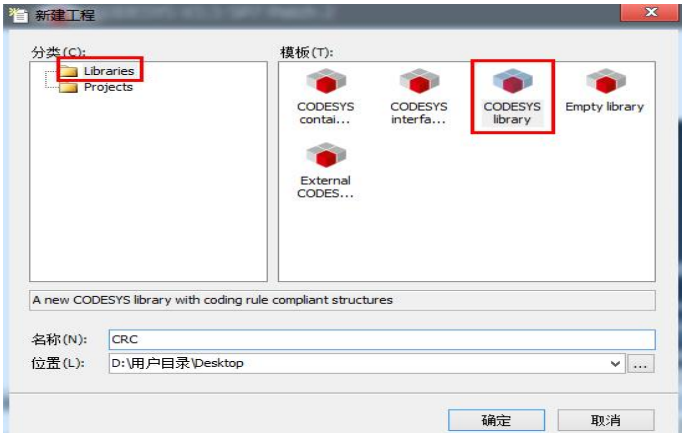


图 7.4-29：建立一个.library 的空白库文件

Step2：确认后，在弹出的工程窗口按照相关章节提到的方法，创建一个功能块，并命名。一般来说命名要尽量采用能反映其实际用途的名称，这里我们要做一个 CRC 校验的库，则命名为 CRC,“编程语言”选择了 ST。

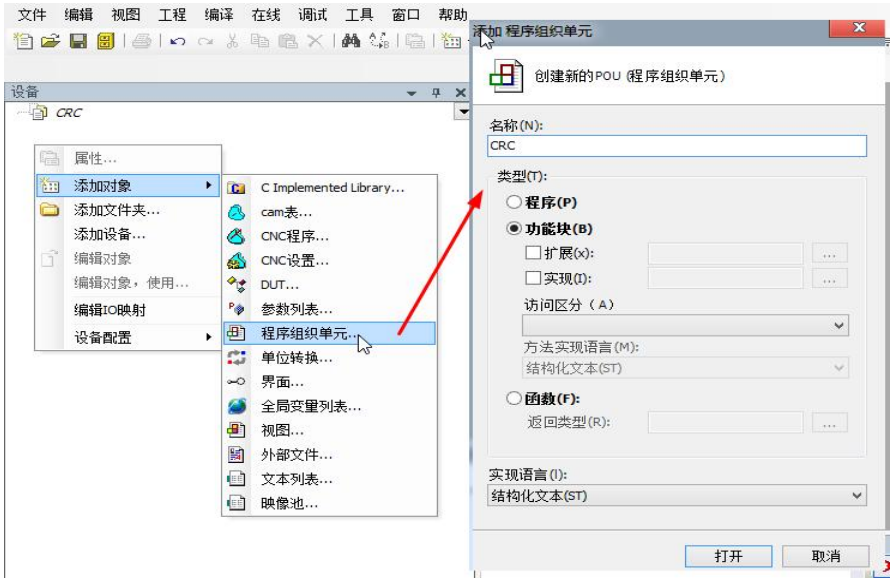


图7.4-30：为空白库文件添加程序组织单元

Step3：编写程序，如下图

```
VAR_OUTPUT
    CRCHigh: BYTE;
    CRCLow: BYTE;
    CRC:WORD;
END_VAR

IF TRUE THEN
CRCHigh:=BYTE#16#FF;
CRCLow:=BYTE#16#FF;
Data_index:=BYTE#16#00;

WHILE Data_index<Data_length DO
Chart_index:= CRCHigh XOR RawData[Data_index];
CRCHigh:=CRCLow XOR CRCHi[Chart_index];
CRCLow:=CRCLo[Chart_index];
Data_index:=Data_index+BYTE#16#01;
END_WHILE;
```

图 7.4-31：程序编写以实现功能

Step4：编译程序，如下图

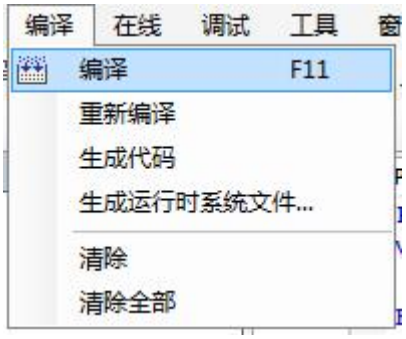


图 7.4-32：编译

Step5：对于自定义库，可以加口令保护。这与对程序加密的原理相同，只需在“工程”菜单中，选中“工程设置”，打开“安全”，设定相应的密码，即可实现加密，如图所示。

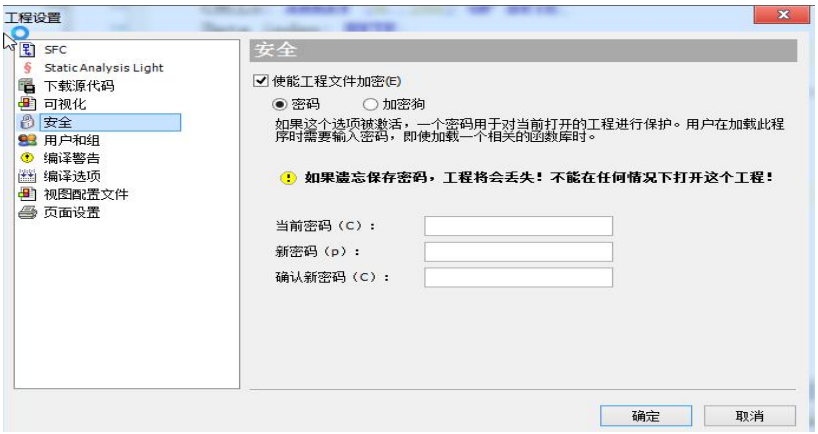


图 7.4-33：为自建库加密

Step6：菜单“工程”-“工程信息”中可设置库文件信息，如图所示。

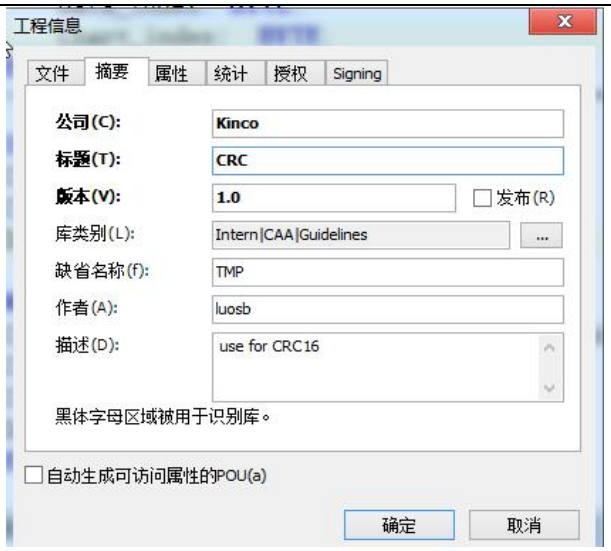


图 7.4-34：库文件信息设置

Step7：菜单“文件”保存设置好的库文件，如图所示。

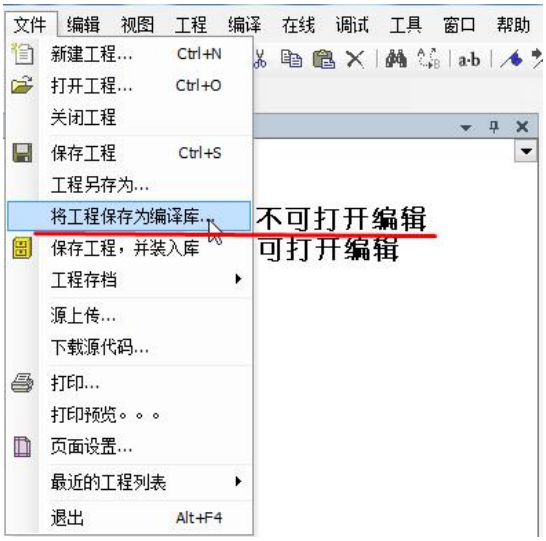


图 7.4-35：不同方式保存自定义库

7.4.6 添加注释

注释可以添加在程序中任何位置，在程序编写时，为程序段（行）、变量、某个地址、功能块或功能添加注释，有助于提高程序的可读性。此外在进行程序调试过程中，有时需要着重调试某个部分，则可以将无关部分代码以注释的方式解除其影响。注释内容并不会被编译（不生成实际代码），但会跟随代码被下载到控制器中（占内存，但实际很小）。对某个变量或地址进行注释后，将光标置于该变量或地址处，可自动显示注释内容。

CoDeSys提供多种程序注释的方式。在不同编程语言环境下，添加注释的方法不同，下文分别介绍在不同语言编程环境下的添加注释。

7.4.6.1 结构化文本（ST）中的注释

结构化文本（ST）的注释有两种：单行注释和多行注释

1.单行注释：

单行注释以“//”为标识符，注释内容书写于该标识符后，“//”不可跨行注释。

2.多行注释：

多行注释以“(*”开头，以“*)”结束，可跨行操作，注释多行。

3.注释的嵌套使用

单行注释只能被嵌套到最外层的多行注释内，在最外层的多行注释内还可嵌套多个单行注释和多行注释。

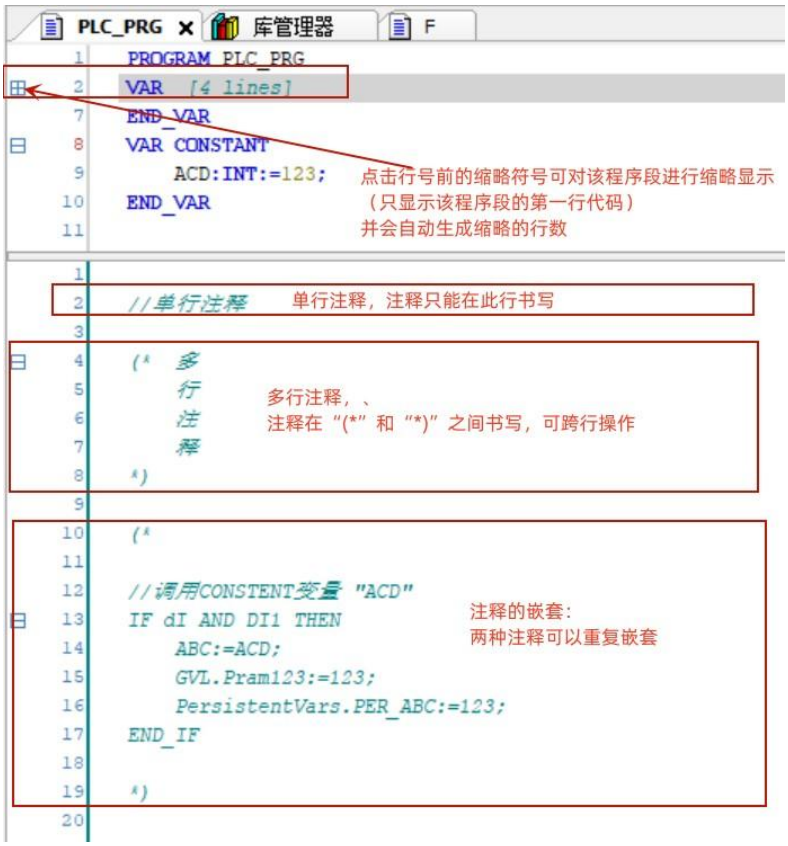


图 7.4-36：ST 语言注释示例

地址注释：

在ST编程环境下，不可以直接对实际地址（I区、Q区、M区）进行注释，需要注释的话，只能使用“AT”将其关联到一个变量上，再对该变量进行注释。

7.4.6.2 FBD、LD 和 IL 中的注释

对 FBD/LD/IL 语言编写的程序进行注释需要先在“工具”菜单下的“选项”中开启“显示节注释”，勾选该项后便可直接进行注释。

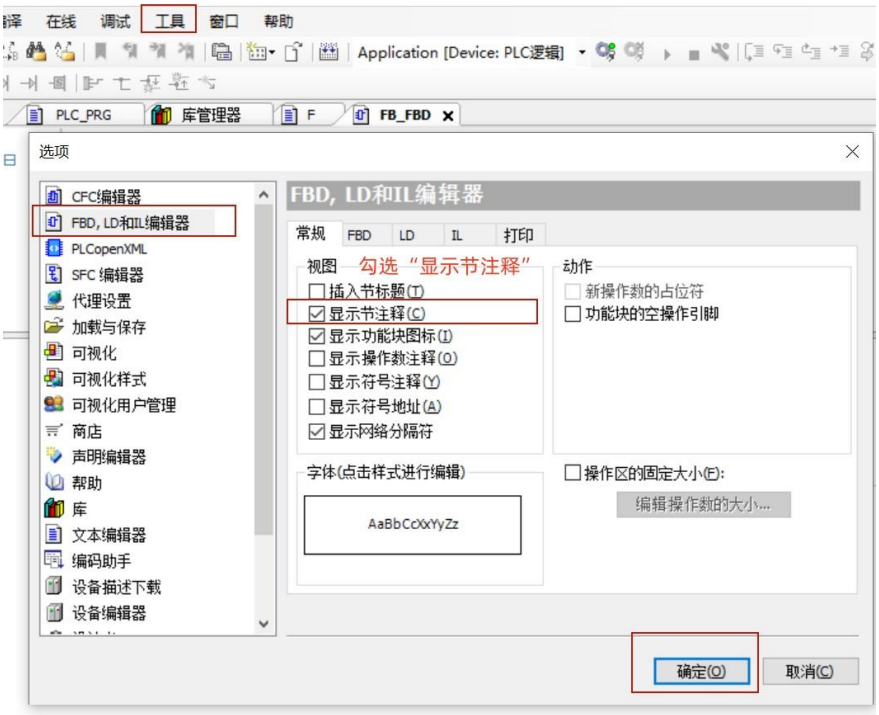


图 7.4-37：打开相应编程语言的注释器

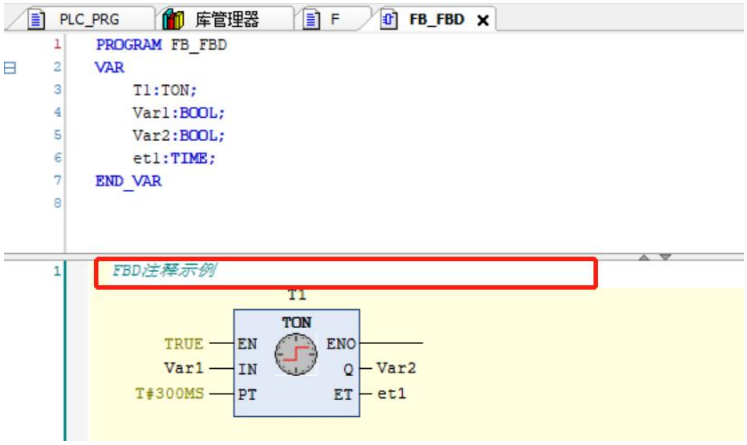


图 7.4-38：FBD 注释示例

7.4.6.3 CFC 中的注释

CFC 中的注释可以随意放置，只需要在“CFC 工具箱”中拖动“注释”图标到程序编辑区域中需要的位置，即可插入注释。注释内容在注释元件中书写。

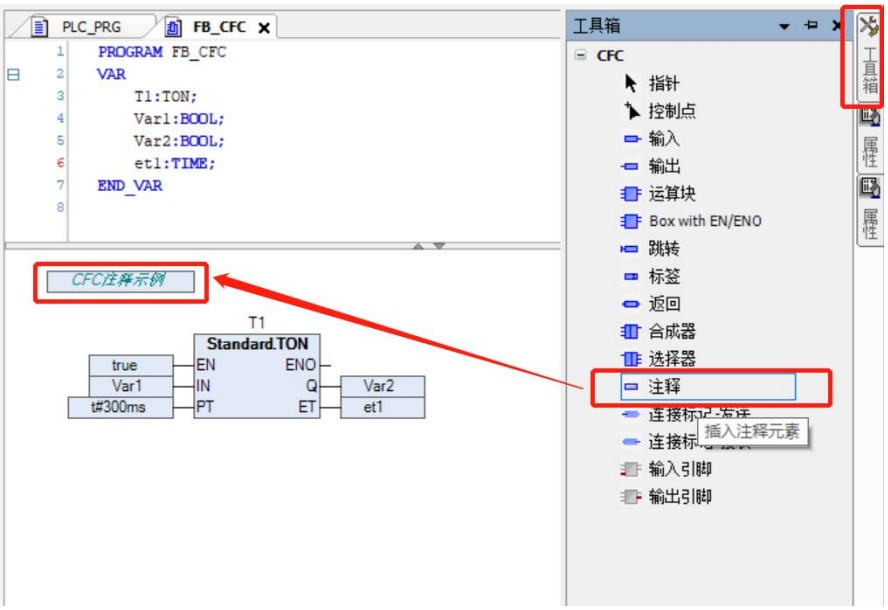


图 7.4-39：CFC 注释示例

7.4.6.4 SFC 中的注释

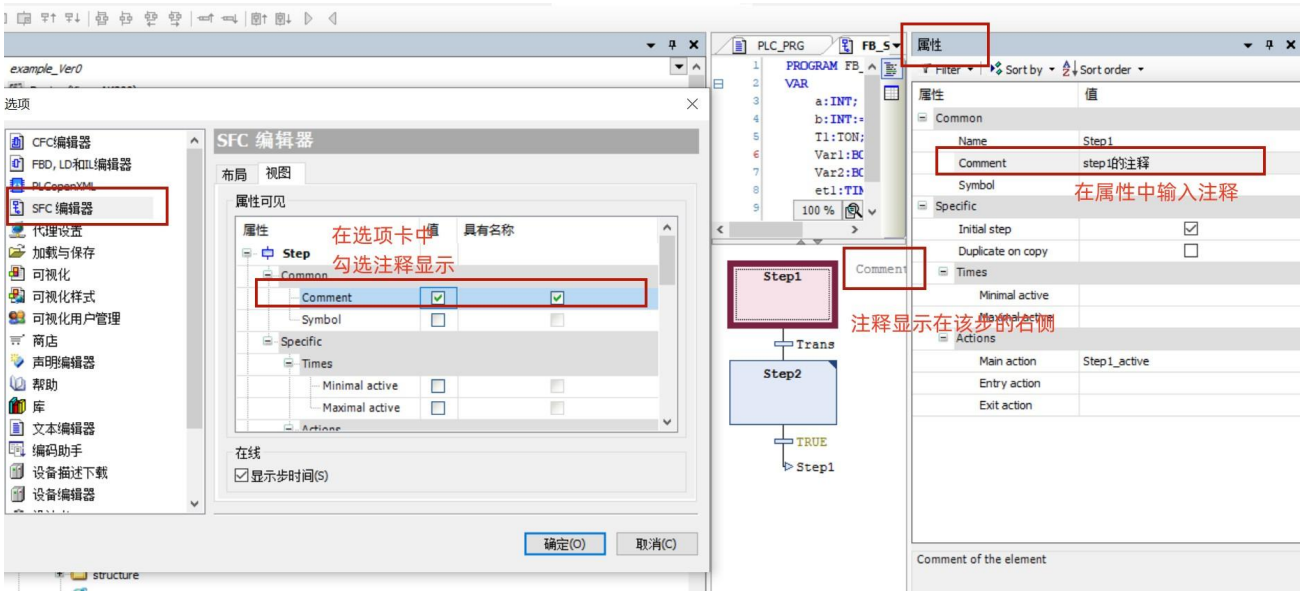


图 7.4-40：SFC 注释示例

如图，SFC 的注释需要先在“工具”菜单下的“选项”中勾选“Comment”（注释）选项。然后在 SFC 属性框的注释栏中直接输入即可，注释会显示相应步的右侧空白处。

7.5 工程管理

7.5.1 打印工程文件

使用“工程”>“打印工程”命令可以打印整个工程的文档，或者选择其中的一部分打印。工程文件由工程信息、文档内容、程序和资源等元素构成，如图7.5-1 所示，其中资源包括全局变量、设备配置、参数管理器等内容。

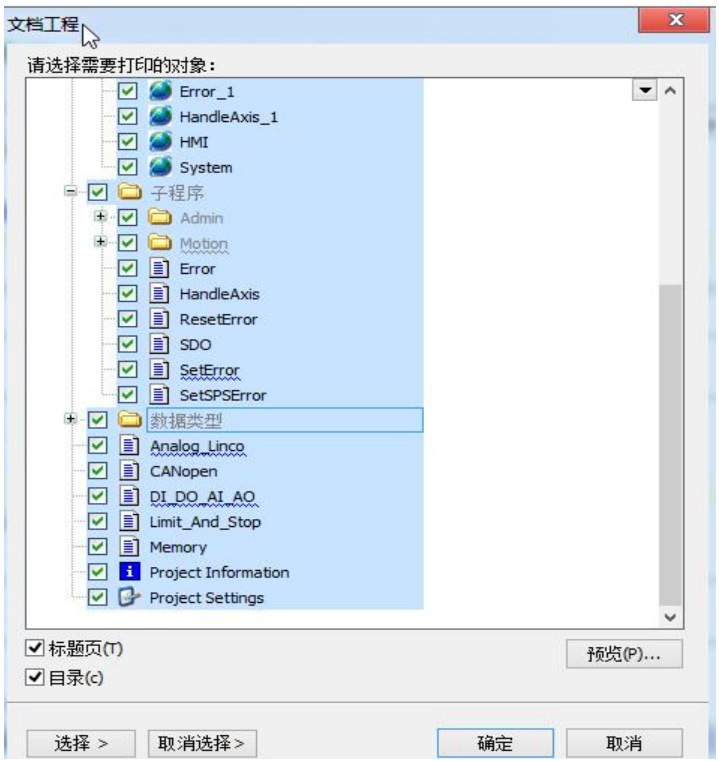


图 7.5-1：打印工程操作

用蓝色高亮可以选择需要打印的区域。如果要选定整个工程，直接在第一行选中工程文件夹即可。如果只想选定一个单独的对象，点击相应的对象。对象的前面带有加号的为多重对象，点击加号可扩展。选好后弹出打印对话框，可设置打印属性。可以按照一定的格式打印一个工程文件。

7.5.2 保存工程/工程存档

在主菜单中选择“文件”下的“保存”选项，或在工具栏中点击“  ”按钮，可以保存当前工程。在主菜单中选择“文件”>“另存为”菜单可以直接复制当前工程，在编程的过程中，要养成随时存盘的习惯，以免由于误操作而造成数据丢失。

按照上面的保存文件方法，保存的只是工程里边的程序等基本内容，但对于一个完整工程，还有设备配置，所引用的库函数等等，只有保存完整的工程，拷贝到其它设备时，才能被完整打开。

在主菜单中选择“文件”下“工程存档”选项，点击“保存存档”即可，在弹出的对话框中选择需要一同保存的元素，点击保存后会生成一个后缀名为“projectarchive”的文件。在工程存档中，为确保工程在传输后被正确打开（编译无报错），通常建议在存档的对话框中选择全部的设备信息和库文件。

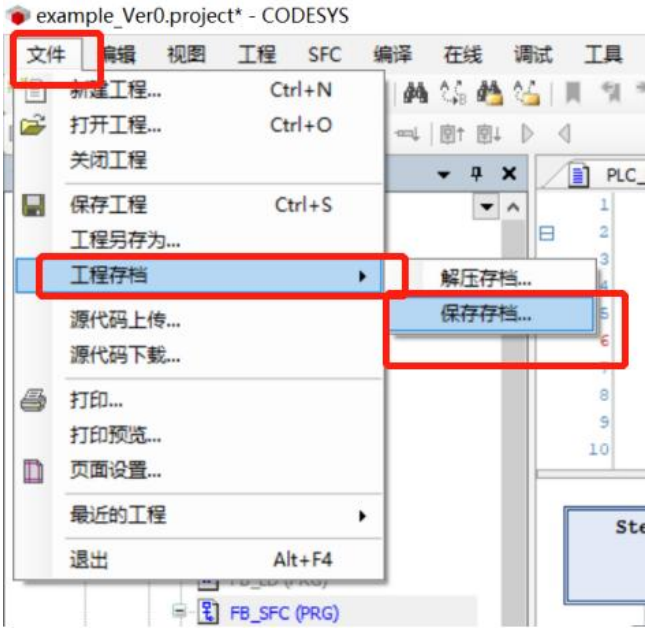


图 7.5-2：工程存档操作

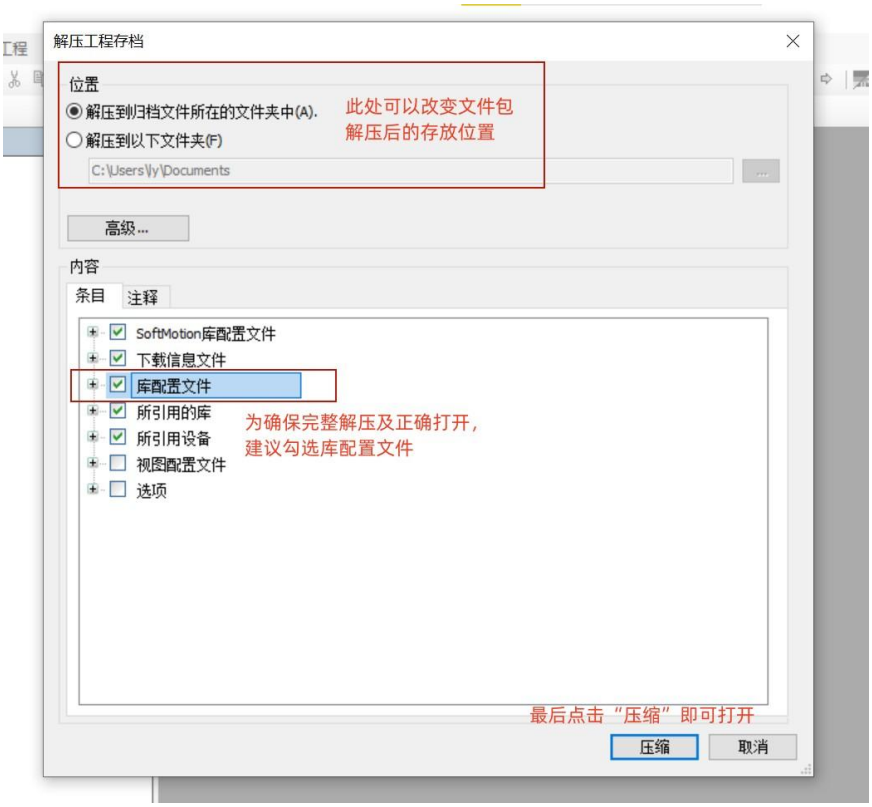


图 7.5-3：工程存档配置页

解压（打开）工程文件包（*.projectarchive 格式文件）的方法为：在主菜单中选择“文件”下“工程存档”选项，点击“解压存档”，在弹出的选项框中找到需要打开的工程文件包，点击打开即可。

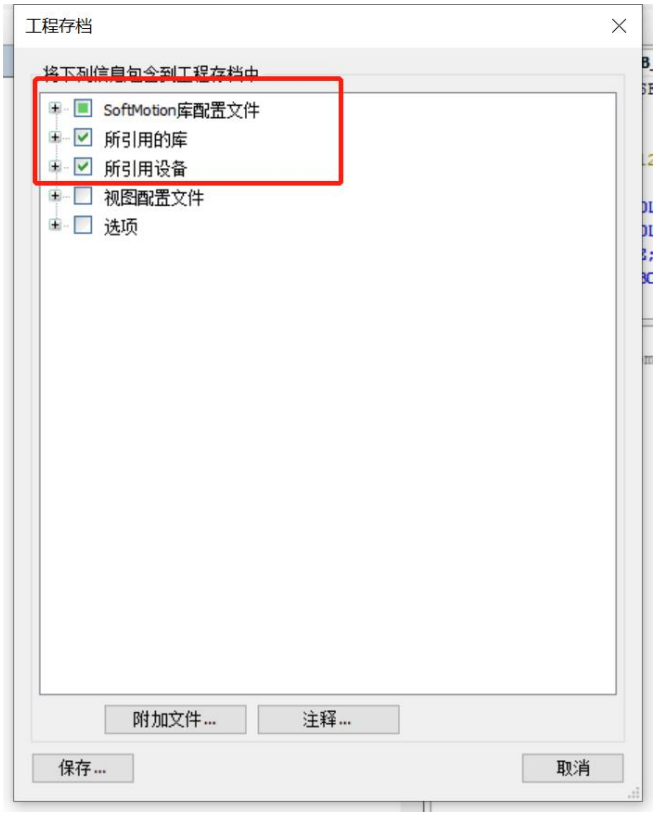


图 7.5-4：工程解压配置页

7.5.3 导入导出工程

选择“工程”菜单下的“导入”/“导出”命令可以导入或导出工程对象，以便在不同的工程文件中交换程序。在导出文件时，对话框底部的“每个子目录树有一个文件”选项，可以选择对象是导出到同一个文件中还是导出到不同的文件中，如下图所示。

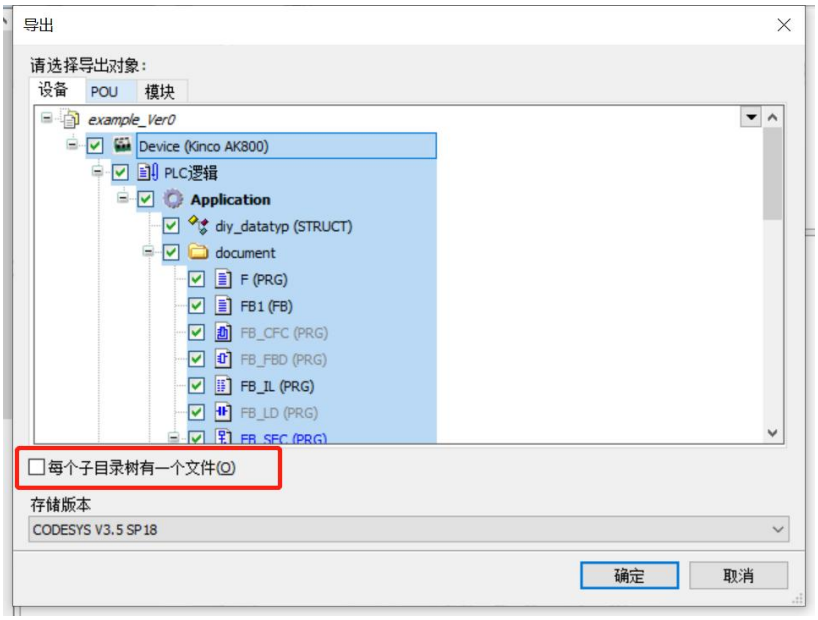


图 7.5-5：导出工程操作

设定完成后，弹出导出工程对话框。在对话框中指定导出对象的保存路径，即在指定目录下生成以 “*.export”为扩展名的若干导出文件，同时弹出消息窗口列表，显示相关信息。

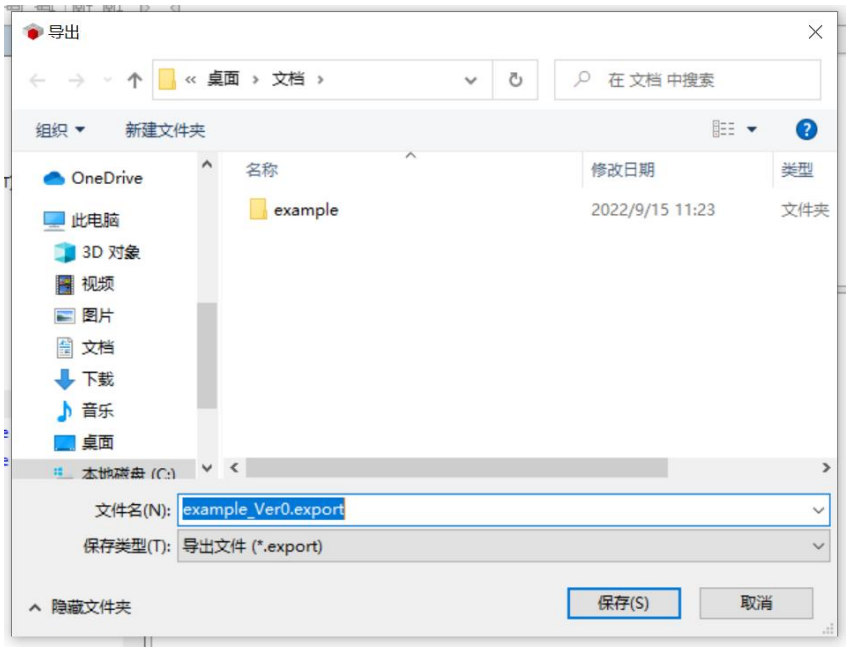


图 7.5-6：导出工程生成“*.export”文件

导入工程对象时，选定导入的文件“*.export ”，对象即被导入到当前工程的相应窗口中去。如果在该窗口中，已经有一个同名的对象存在，则会出现对话框，询问是否替换。

不仅仅是整个工程可以执行“导入”、“导出”操作，在CoDeSys中任何POU对象都可以被导入和导出（程序、全局变量表、用户自定义的数据类型、功能块等）。

7.5.4 比较工程

选择“工程”菜单下的“比较工程”命令可比较当前工程与另一个工程的所有对象内容。如果想知道是否在当前工程中做了修改，那么可以将当前打开的工程和它的前一个版本进行比较。当执行此命令后，弹出工程比较对话框，如图所示。

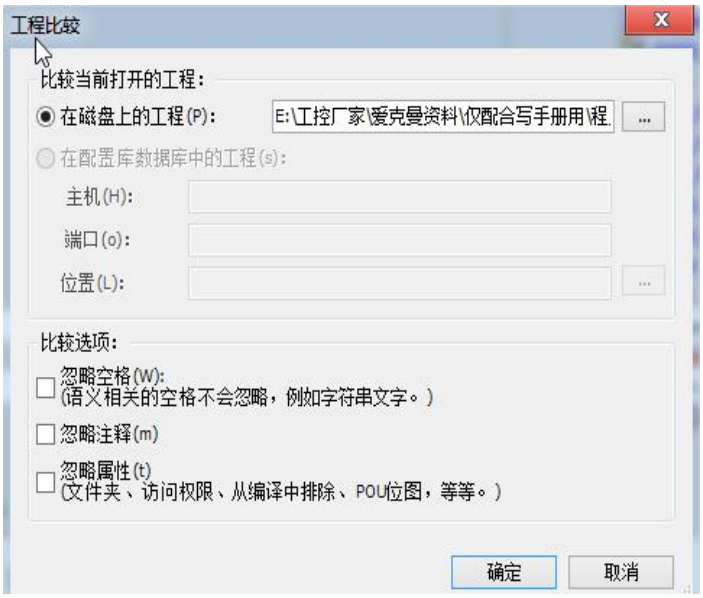


图 7.5-7：比较工程

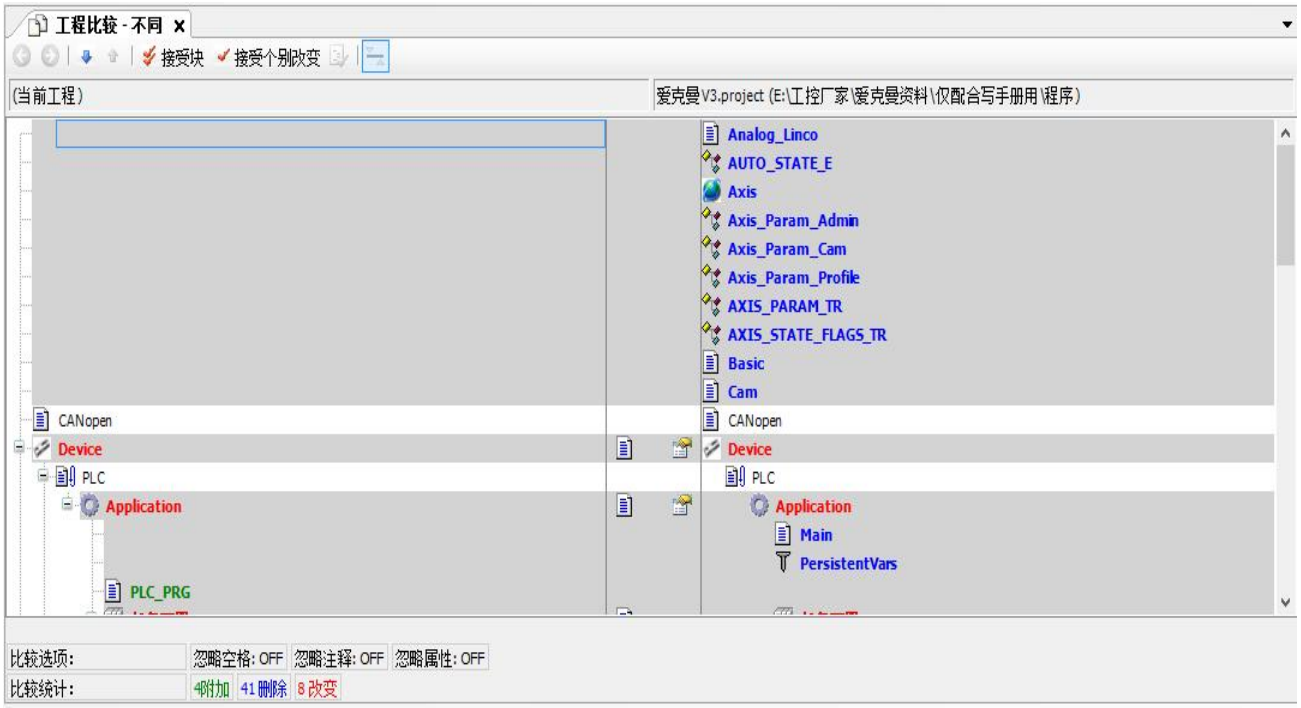


图 7.5-8：工程比较信息

选定与当前工程比较的目标工程后，弹出比较结果列表，如图所示。不同颜色的对象文字代表不同的比较结果，共有五种可能的颜色，如下表7.5-1所示：

表 7.5-1：比较工程颜色显示寓意

颜色	意义
黑色	对象内容相同。
红色	对象内容不一致。
蓝色	当前工程中新增加的对象。
绿色	当前工程中没有的对象。
灰色	当前工程中新增加的对象。

在具体的程序编写过程中，还可以利用“比较”命令，对修改前后的程序代码进行比较，如图所示。在比较模式下，不允许编辑工程对象，所有操作功能禁用。只有关闭了比较列表窗口，才能对工程进行编辑。另外，比较工程不支持硬件配置的比较。

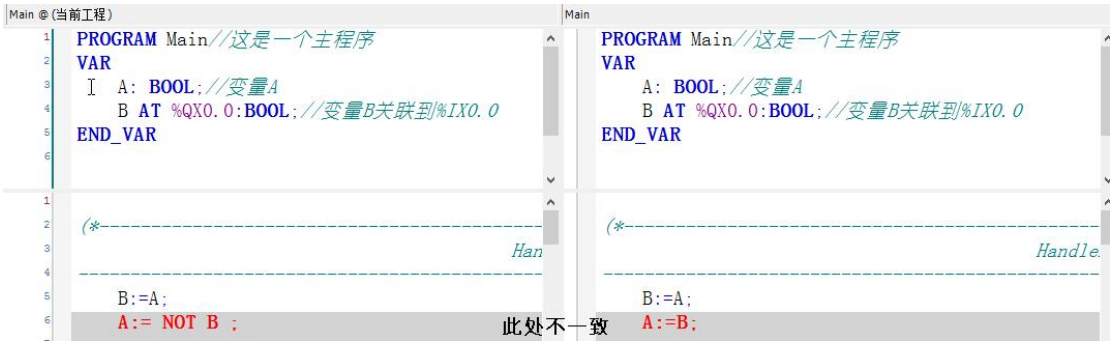


图 7.5-9：“比较”命令信息

7.5.5 工程加密

在“工程”菜单下的工程设置中找到其目录下的“安全”项，选中“加密”选项，可从下方选择“密码”、“加密狗”、“证书检查”三种加密方式，再根据不同的加密方式进行相应的加密设置即可。

图 7.5-10：工程加密操作

7.5.6 用户密码

工程加密可以保护版权和防止程序非法篡改。“工程”>“选项”>“密码”选项可以设定单一用户口令，设置CPU的授权访问，防止文件被随意地打开和更改，工程加密已经在之前的 7.5.5章节介绍。

对于一个写保护的工程，在没提供密码的情况下是不能被更改的。如果打开一个文件时系统提示输入写保护密码，而只是简单地电机“取消”的按钮，那么工程中所有的编辑设定功能都禁用，只可以使用编译或仿真等查看功能。如果存在多个用户，需要定义用户组，为每个用户组分配不同的访问权限。系统共提供了8 个用户组，它们对工程对象拥有不同的访问权限。每个用户组的成员都通过密码来确认身份。“工程”>“用户组命令”用来设定用户组的密码。用户组从0-7，0组拥有管理员权限，只有0组的成员可以决定其它用户组和对象的密码和访问权，如图示。建立一个新工程时，所有的密码初始状态都是空的。当一个密码被设置为0组，用户以0组成员的身份进入工程时，就可以对其它级别的组设定密码。如图所示，在左边的组合框“用户组”中选定组，在右边输入相应的密码。

图 7.5-11：用户组密码设置

使用“工程”>“对象”>“属性”命令可以为用户组设置不同的访问权限，如图所示。访问权限有三种可用的设置：

- 无访问权限：对象不允许被此用户组的成员打开。
- 只读权限：对象能够被此用户组的成员读入，但不能被更改。
- 完全权限：对象能够被此用户组的成员打开和更改。

访问权限的设置仅对选中的对象有效，每个POU、硬件配置或全局变量都是对象成员。

如果忘记了口令请及时与制造商联系，口令随工程一起保存。

第八章 编译与调试

当程序编写完毕后，要对程序进行编译。当编译通过后，才能将程序下载到控制器中。本章主要对 CoDeSys 软件的编译与下载过程进行介绍。

8.1 编译

CoDeSys软件的“编译”菜单提供了“生成代码”、“全部清除”和“清除”三种编译命令，用于检查程序有无语法错误以及代码生成。

➢ 生成代码[F11]：

进行当前激活应用的编译，意味着所有属于这个应用的对象都将被检查。编译过程在登录一个更改过的应用程序时自动进行。编译后生成当前应用的机器代码，执行登录命令时，生成代码默认执行。

➢ 清除：

用于删除当前激活的应用的编译信息。编译信息是在应用进行最新一次下载之后保存在工程路径中的“ *.compileinfo”文件。经过一个清除操作之后使在线改变对当前的应用将成为可能。

➢ 全部清除：

删除所有应用程序的编译信息。 此信息是在应用程序最后的下载中被创建的，它被保存在项目目录中的一个文件（*.compileinfo）里。此命令必须在重新登录之前下载。因此，在线变化不再可能实现。从 CODESYS 的 V3.5.3.40 版本起，对持久性变量的内部信息也被删除，因此，它们将在下载后重新初始化。

没有语法错误的程序才能生成可执行的目标文件。编译的结果会显示在消息窗口中，如图 8.1-2 所示。常见编译错误信息参见系统帮助文件。在消息窗口双击错误提示，可自动跳转到程序当中具体发生错误的位置。

执行“生成代码”后可以看到，添加到任务里面的“PLC_PRG”显示为蓝色，没有被添加到任务里面的程序或没有被调用的功能块和功能则显示为灰色，如图 8.1-1 所示。编译指令不会对灰色的 POU 进行语法检查，因为该程序单元没有处于活动状态，“生成代码”指令只针对于处于活动状态的 POU 进行语法检查与代码生成。

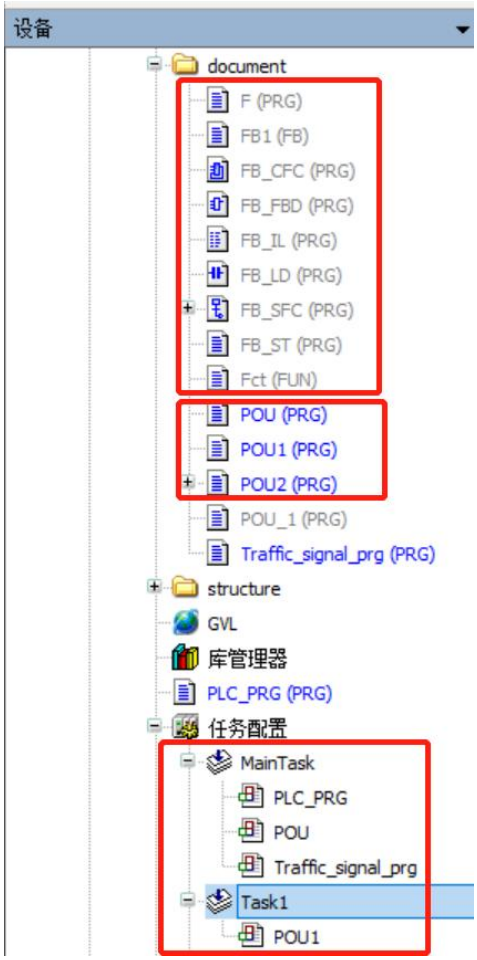


图 8.1-1：编译后的程序区

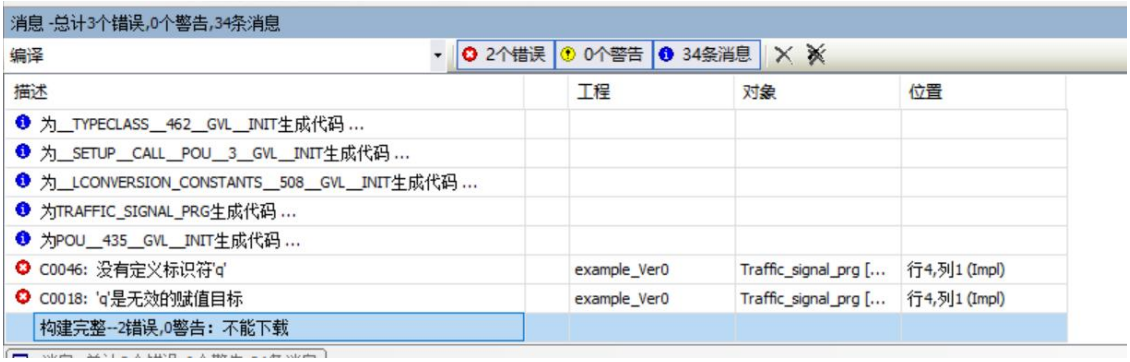


图 8.1-2：编译信息窗口

8.2 显示参考数据

CoDeSys 软件的“工程”菜单提供显示一些参考数据的命令，这些命令只有在编译通过后才有效。

8.2.1 查看交叉引用列表

“工程”>>“查看交叉引用”命令可以显示并查看所有应用程序点。所谓“应用程序点”

是指某一个变量、地址或程序在工程中的任意位置。除了通过工程菜单打开交叉引用列表，还可以右击符号名，选择“显示交叉引用”。在交叉引用列表中的“符号”出输入应用程序名或者变量名等，回车即可显示其引用列表，如下图8.2-1所示：

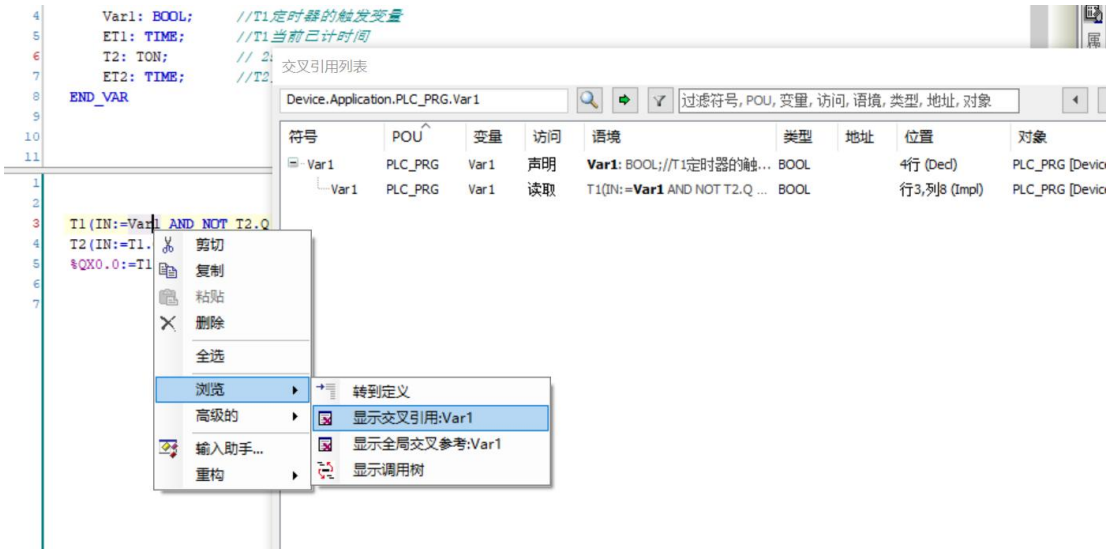


图 8.2-1：显示变量的交叉引用

选中交叉参考列表中的一行，按“转到”按钮，或者直接双击此行，程序会在编辑器中显示出相应的点。通过这种方式，可以任意跳到所需的应用程序点处，无须进搜索跳转。

8.2.2 查看定义

鼠标右击应用功能块名或者变量名处，选择转到定义处，可以快速跳转到变量定义的列表中（库的功能块则会跳转到库相关目录下的功能块或功能下），如下图所示。

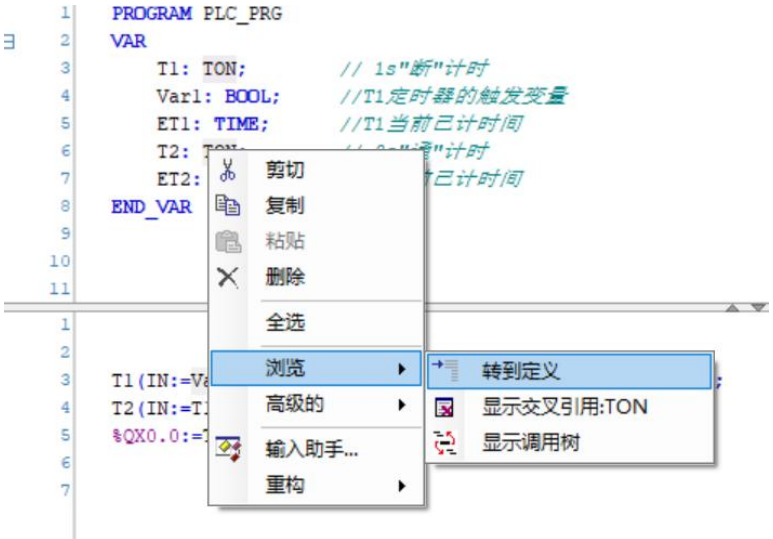


图 8.2-2：查看程序点定义

8.3 下载

8.3.1 设备安装与连接

1.设备安装：

根据实际工程的需要，规划并制定合理的接线方案，将现场设备正确连接到控制器的接线端子上。（请参考AK800硬件接线手册）在检查所有电缆连接无误后，再接通系统电源，并确认控制器LED显示正常，以保证控制器可靠运行。

2.建立PC通讯：

通过标准的RJ45网线，将控制器连接到个人计算机（PC）的以太网通信接口，建立数据传递通道。在这之前需要设置将PC与控制器处于同一网段，AK800默认的IP地址为：192.168.0.250 ，PC网段的修改方法参考 3.6章。

8.3.2 建立通信连接

设置好上位机 IP 后，如果已经正确连接网线到控制器端口，控制器上电后，AK800 的 EtherNet 通讯连接的网口绿色指示灯常亮，黄色指示灯闪烁则代表连接成功。软件上通过对设备的扫描，可得到以连接的设备信息，如下图 8.3-1 所示。

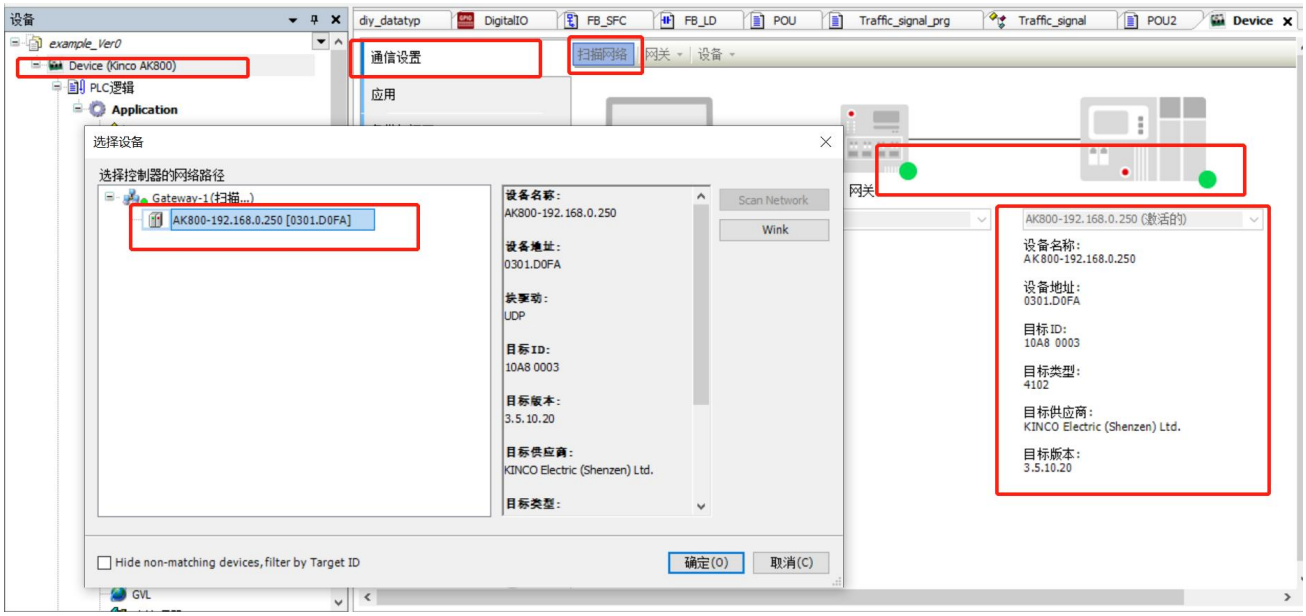


图 8.3-1：上位机与控制器通讯连接

8.3.3 程序下载与上载

确保设备与上位机成功连接后，点击“在线”菜单中“登录到”，可以实现程序的下载。编译通过后生成的目标代码在进行下载时，会将全部的目标文件下载到模块中。同时将模块复位，所有变量返回到初始状态。

当要下载的工程与控制器内的工程不一致或控制器处于运行状态时，则出现系统提示下载信息（一致的话无提示直接进入监控状态），如图所示。

图 8.3-2：登录方式选择

如果要在线监控程序，不想更新的程序下载到控制器中，只想建立连接，可以选择“没有变化后登陆”，若选择“登录-在线修改”，则系统会将新程序下载到CPU模块中去，但不会对变量进行初始化操作。选择“登录并下载”则会在重新下载生成的目标代码到CPU时，也会将对变量进行初始化。

下载程序到控制器后，为了保证在断电重新上电后，程序不丢失，还需创建引导工程。创建引导工程在“在线”菜单下“创建引导应用”。

如果需要将程序从控制器上传到上位机中，则需要在下载程序的同时，进行“源代码下载”的操作：在“在线”菜单下选择“下载源代码到连接设备上”，只有进行了这一步操作，才能使代码上传成为可能，否则源代码上传操作将无效。

在上面我们看到登录、下载、创建启动工程、还有源代码下载等操作，为了解释清楚这几个操作的区别，需要了解控制器程序的下载过程，详情描述如下

- 登录：CoDeSys 需要将编译好的程序下载到控制器，首先需要将CoDeSys 与控制器之间建立一个连接。“登录”命令的作用就是将控制器 和CoDeSys 建立一个连接。建立完连接后，CoDeSys 会自动判断程序是否改变。
- 下载：若程序没有改变，则自动转入联机监控状态，通过CoDeSys 就可以联机监控当前控制器的状态。若程序发生了改变，则会提示 “代码已经改变，想如何做”，选择“没有变化后登陆”，则不下载程序，转入联机监控状态；这时若想下载程序，可以通过在线菜单中的“下载”命令下载程序。也就是说，“下载”命令只有在登录以后才有效。若选择“在线修改后登陆”，则会把修改后的程序下载指控制器 的CPU内。
- 创建启动应用：但是因为CPU断电后，数据全部清除，所以为了保证程序在控制器重新上电后依然存在，在下载到CPU 后，还需要将程序保存到FLASH 内，这个过程叫创建启动应用。创建启动应用后断电重启程序才不会丢失。
- 源代码：下载启动应用只是供PLC自行调用，但不会提供代码上载功能（也就是通常所说

的上载程序），所以要执行源代码下载才可以执行程序上载功能。源代码下载可以在登录状态下，执行下载源代码，在弹出确认窗口中点击“Yes”确认即可下载，下载进程在软件界面左下角提示。

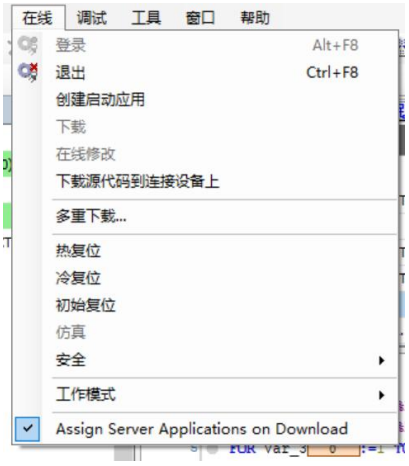


图 8.3-3：在线菜单目录

源代码上载则在建立通讯连接状态下，点击文件菜单下“源上传”，在弹出窗口中选择设备，点击确定，最后选择上传文件所在位置，即得到一个后缀名为“*.project”的上传文件，上传后软件还会提示是否打开上传文件。

用户还可以根据需要设定源代码下载的内容，以及什么时候执行源代码下载。具体为“工程”菜单下“工程设置”下载源代码选项

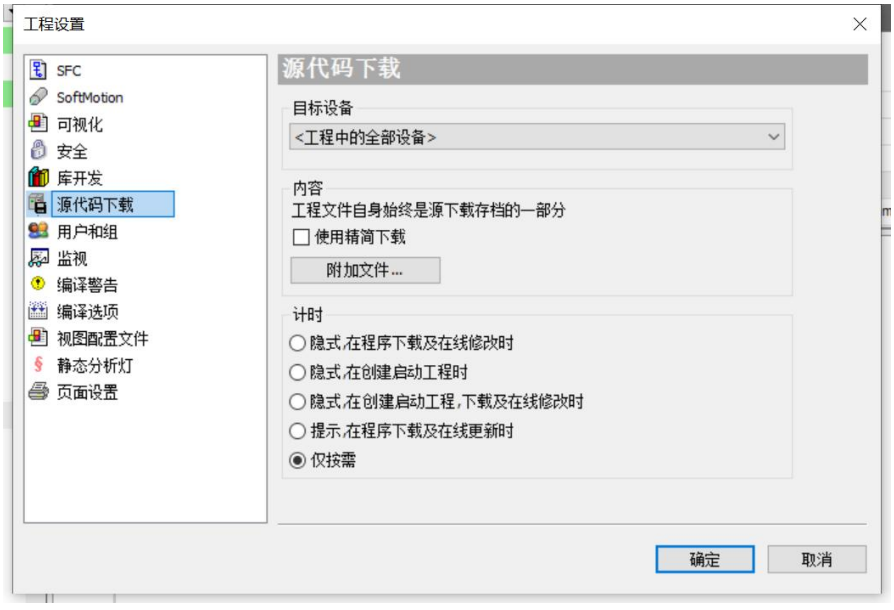


图 8.2-4：在工程设置中设置源代码下载选项

8.4 调试

系统所支持的调试命令都在“调试”菜单下，并且在调试状态下可以使用，在调试状态下，系统用不同的默认颜色来代表不同的状态和操作，例如逻辑真（蓝）、逻辑假（黑）、断点（浅

蓝）、流控制（绿色）等等。

8.4.1 进入调试状态

执行“在线”>“登录”命令进入调试状态。将程序下载到控制器的CPU模块中，称为在线调试状态。如果没有连接硬件，在本地计算机模拟运行用户程序，称为仿真模式。当“仿真模式”被勾选时，点击“登录”则会自动进入仿真调试状态。在程序中，如果调用了与硬件相关的功能块，则不可以使用仿真模式。例如，系统事件以及对实际内存的访问等都不可以使用仿真模式。

8.4.2 退出调试状态

执行“在线”>“退出”命令退出调试状态，回到编程状态。

8.4.3 运行程序

执行“调试”菜单下的“启动”命令，启动CPU模块中的程序或处于仿真模式下的程序。一般需要运行程序的情况有以下几种：

- 执行“在线”菜单下的“登录”命令或重新下载程序后。
- 执行了“在线”菜单下的“停止”命令后。
- 用户程序设置了断点之后。
- 执行一个“单循环”之后。

8.4.4 停止程序

执行执行“调试”菜单下的“停止”命令，可以暂停程序在模块或在仿真模式下的运行，保存当前变量值。此时程序仍处于调试状态下，执行“调试”菜单下的“启动”命令可重新启动程序，从上次停止的地方开始继续运行。

8.4.5 复位

控制器总共提供三种复位方式（均可在调试状态下通过执行“在线”目录下的复位操作进行）：

1.热复位：热复位后，除了保持型变量（PERSISTENT 和 RETAIN 变量）外，其它变量都被重新初始化。如果设置了初始值的变量，热复位后这些变量值还原为设定的初始值，否则变量都会被置为标准初始值 0。

2.冷复位：冷复位命令不但将普通变量的值设置为当前活动应用程序的初始值，而且将保持型变量（PERSISTENT 和 RETAIN 变量）的值也设置为初始值 0。

3.初始化值复位：当在设备树中选择一个可编程设备时，无论是在离线还是在在线状态下

都可以使用此命令。使用此命令将使设备复位到初始状态，即设备中的任何应用、引导工程和剩余变量都将被清除。由于所有工程信息被清除，重新登入后，需要重新“下载”程序，并“启动”运行。

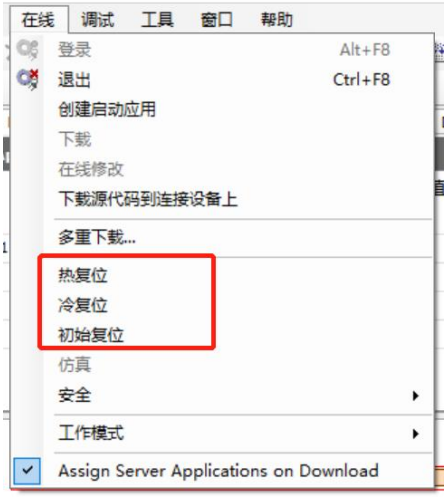


图 8.4-1：“在线”目录下的复位操作

8.4.6 断点和单步

8.4.6.1 断点

断点是指可以在程序中设置进程停止的地方。通过设置断点，可以在程序的具体地点观察其变量值，便于分段调试程序，有助于深入了解程序运行的机制，发现及排查程序故障。

断点调试只能在在线状态下进行。通过“调试”菜单下的断点命令组，可以进行不同的断点调试操作。

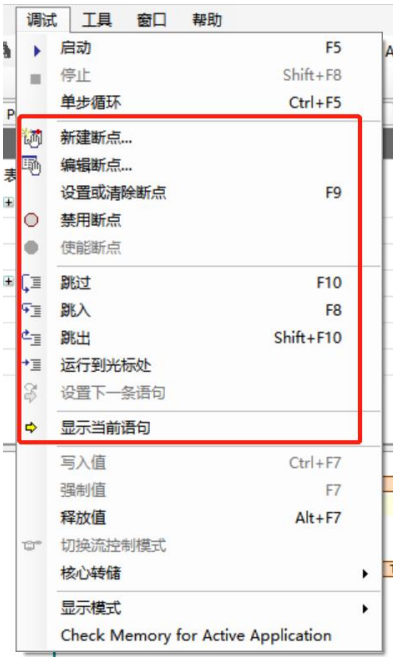


图 8.4-2：调试菜单目录

此处不再对调试目录下的选项功能进行详述，具体请参考2.2.1章节的第7点。

断点的设置位置取决于活动窗口中程序的语言类型。ST语言中断点的三种状态，如下图8.4-3所示。

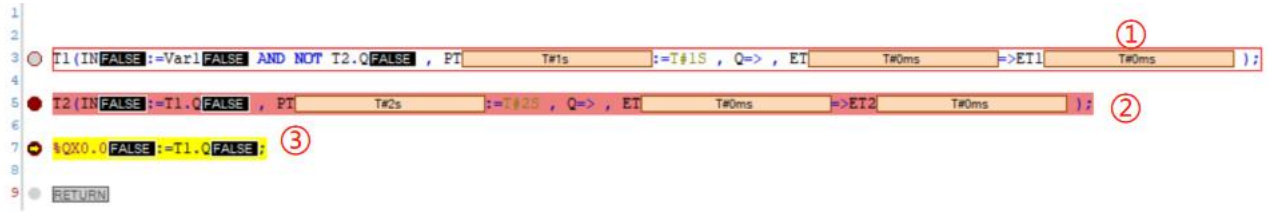


图 8.4-3：ST 编程中的断点标识

- 第一种是“断点未使能（已禁用）”状态，可以通过“新建断点”动作实现，右击该程序行选择“使能断点”可以让该断点进入使能状态。
- 第二种是“断点使（已启用）”状态，可以通过“设置或清除断点”操作或右击以建立断点的程序行选择“使能断点”实现，右击选择“禁用断点”解除启用状态。
- 第三种是“停止在 BP”状态，即程序停止在该断点状态，选择“调试”菜单下的“启动”（F5）或“跳入”可执行改行程序并自动定位到下一行要执行的程序处。

在IL和ST中，断点设置在当前光标所在的行。在FBD和LD中，断点设置在当前被选定的网络中。在SFC中，断点设置在当前选定的步上。

8.4.6.2 单步

通过不断使用单步的相关功能，使之一步一步地执行以查看程序是否按照设计进行工作，以此检查程序逻辑的正确性。在对应的活动窗口中，对于不同的编程语言，单步的含义有所不同。

单步功能包括：跳入、跳出、运行到光标处等。执行“跳出”或“跳入”命令实现单步执行。当遇到断点时，程序执行当前语句后停止。当遇到功能块或函数时，“跳出”命令会跳过功能块或函数执行下一条语句，而“跳入”命令则跳入功能块或函数的内部单步执行。各命令功能详情参考2.2.1章节的第7点关于调试目录的讲解。

8.4.7 单步循环

执行“调试”菜单下的“单步循环”命令，可以使程序执行一个周期即停止并等待下一次运行指令，也可使用快捷键[Ctrl+F5]执行一次单步循环命令。如果不点击“启动”，则程序一直处于停止模式。 如图 8.4-4 所示，通过单步执行查看计数程序是否准确运行。

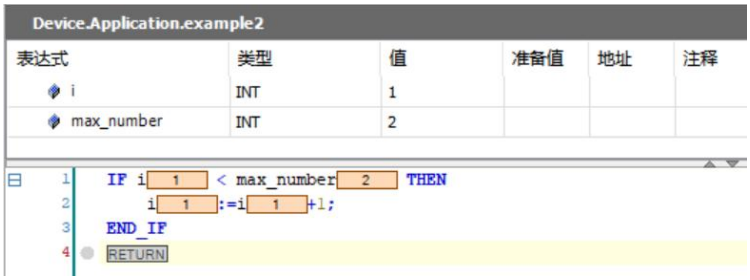


图 8.4-4-a：执行一次单步循环（i+1=0+1=1）

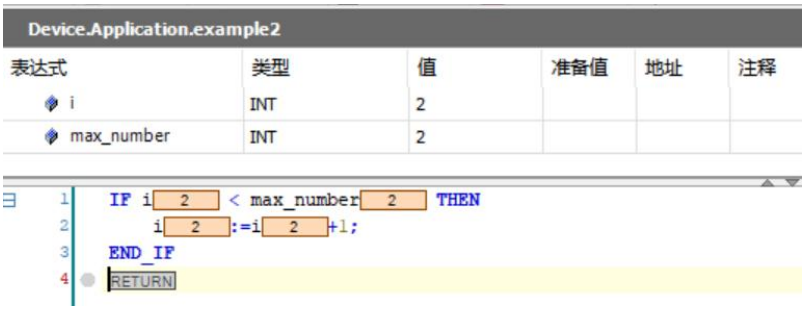


图 8.4-4-b：执行第二次单步循环（i+1=1+1=2）

8.4.8 变量赋值

8.4.8.1 设定新值

在在线调试状态下，为设备设定新值的方式有：

1.点击变量表上的“准备值”设定新值。如下图8.4-5所示：对于BOOL型变量，只需要单击“准备值”即可修改要写如的新值状态，对于非BOOL型变量，双击“准备值”可以进入变量新值修改。

图8.4-5：通过点击变量表上的“准备值”设定新值

2.直接在程序区域双击变量名称旁的当前值框以打开新值修改弹窗。如下图8.4-6所示：

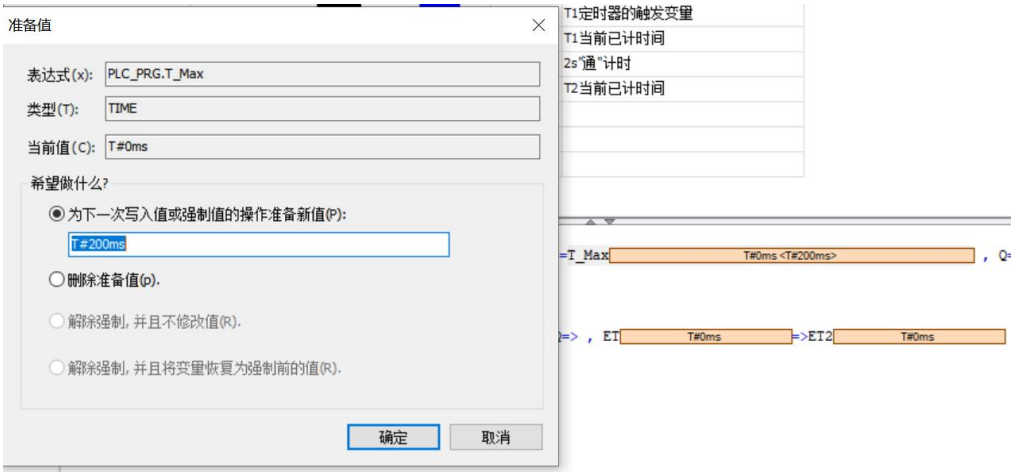


图8.4-6：通过双击变量名称旁的当前值框修改新值

无论用何种方式设定新值，设置新值成功后，除了在设定了新值的变量的准备值列内出现新值后，还会在程序区的变量当前值中显示当前值与新值的组合，如下图8.4-7所示：

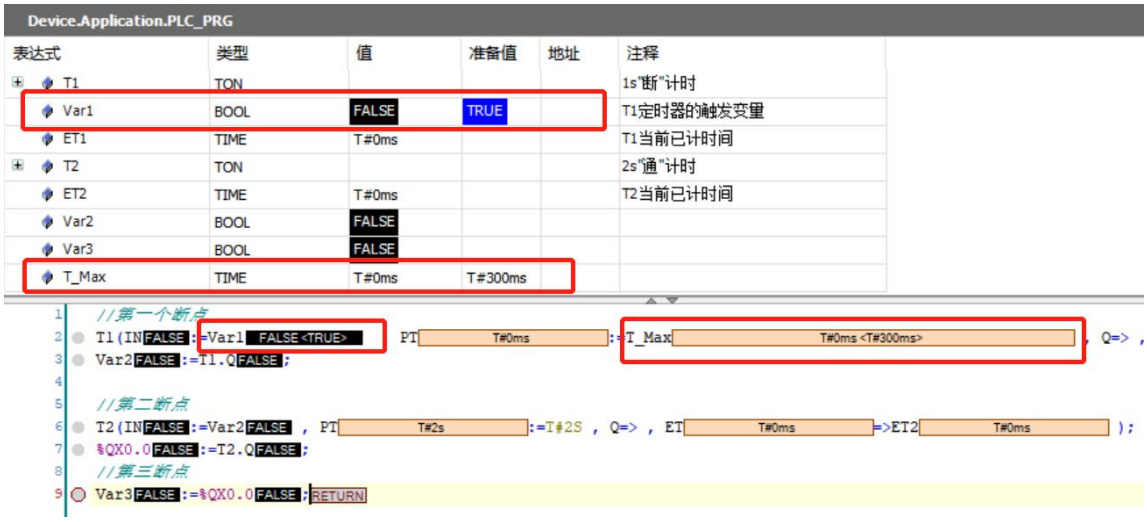


图 8.4-7：新值设定显示

8.4.8.2 激活/写入新值

在新值设定完成后，还要执行“写入”操作才能真正修改变量的当前值。新值的写入可以执行“调试”菜单下的“写入值”命令或直接使用快捷键[Ctrl+F7]执行写入值，如图8.4-8所示，也可以在程序区的任意位置右击，选择目录下的“写入Device.Application的所有值”执行新值的写入，如图8.4-9。注意，任何设定了新值的变量都会在执行新值写入时被修改当前值，即可以同时写入多个新值。

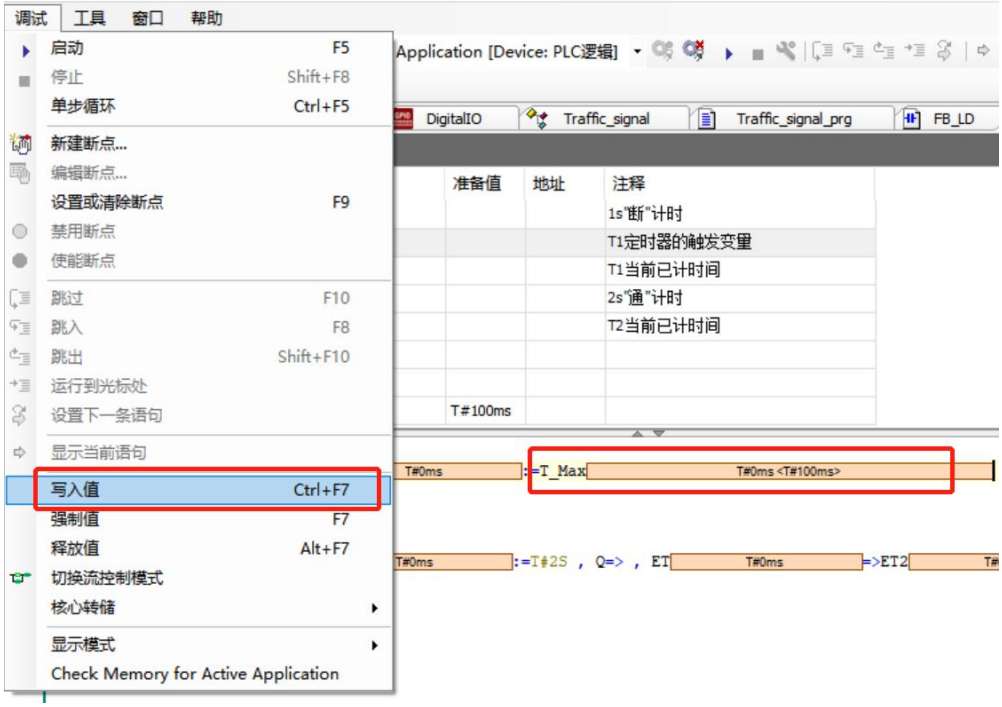


图 8.4-8：通过调试菜单执行新值写入

图 8.4-9：通过“写入 Device.Application 的所有值”执行新值的写入

8.4.9 变量强制值

强制写变量的方法与“写入”命令一样，先输入新值，然后执行“调试”菜单下的“强制”命令，或者使用快捷键[F7]写入强制值，还可以通过右击选择“强制Device.Application的所有值”强制变量值。当开关量的输入为物理点输入时，例如%IX0.0，在线模式下必须使用强制值，但在仿真模式下可以使用输入值。允许多个变量写入新值后执行“强制值”命令。变量被强制时，前面显示“F”强制符号，如图8.4-10所示。

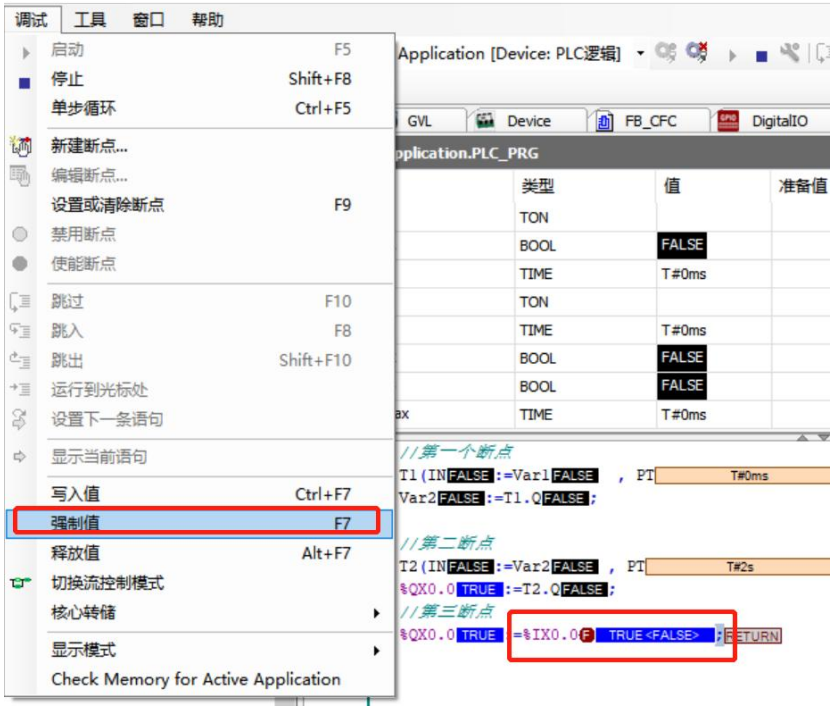


图 8.4-10：强制值

变量在被强制后，在程序的每个循环之后都被写入强制值，直到执行“解除强制”命令为

止。解除强制可以通过“调试菜单”下的“释放值”命令（快捷键[Alt+F7]）或右击程序区选择“释放Device.Application的所有值”解除对变量值的强制。

与“强制值”命令不同，执行“写入值”操作的变量只被写一次，在程序的运行过程中允许变量被其它程序赋值。

8.4.10 查看调用栈

在某个断点处，程序停止后执行时执行“视图”菜单下的“调用堆栈”命令，会显示调用堆栈里的当前程序的列表，如图所示。

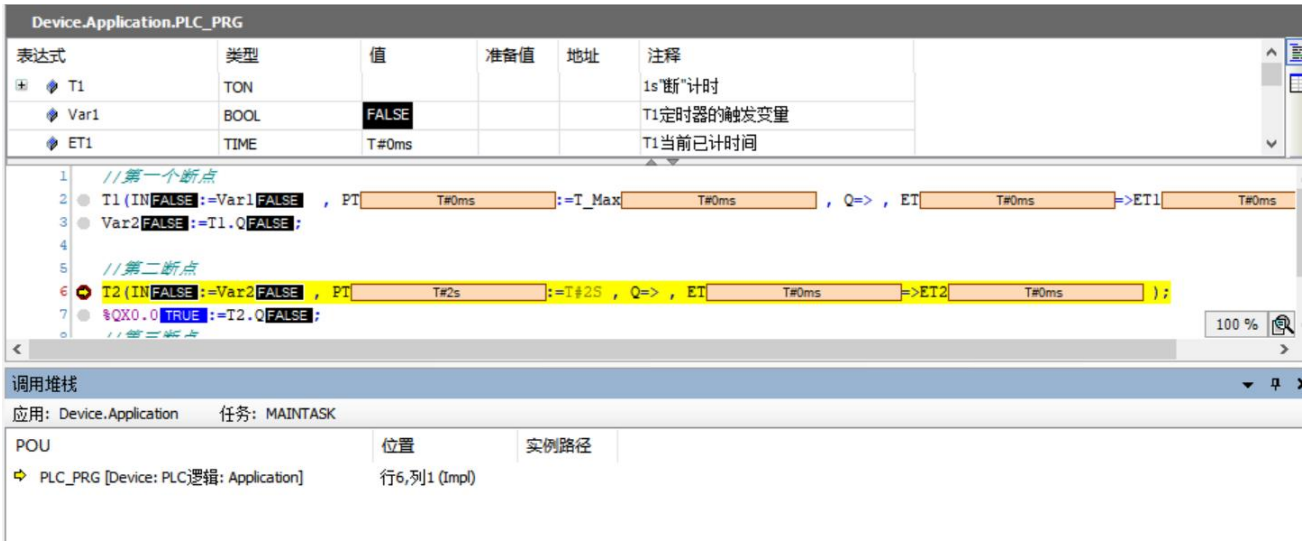


图 8.4-11：调用堆栈视图

当您要单步执行程序时，此视图非常有用。它以完整的调用路径显示当前到达的位置。“调用堆栈”窗口的元素如下表8.4-1所示。

表 8.4-1：“调用堆栈”视图组成介绍

应用	当前到达的执行处的 POU 所属的活动应用名称。
任务	当前到达的执行处的 POU 所属的任务名称。
POU	程序执行中所停留在的功能块的名称，列表中的第一行描述了当前的执行位置，用黄色箭头标记。如果这个位置在一个被另一个调用的 POU 中，第二行描述调用的位置。如果调用者又被另一个构建块调用，则此调用位置在第三行中描述，依此类推（嵌套）。
位置	程序执行所在的程序模块内的位置：包括文本编辑器中的行号和列号和图形编辑器中的网络或元素编号。
实例路径	程序执行所在的实例。

8.4.11 显示流控制

执行“调试”菜单下的“切换流控制模式”命令后，运行的程序行以绿色显示。在“单循

环”模式下，用该命令可非常直观地看到程序的当前运行流程。下图8.4-12-a中，变量i满足条件一，故执行的是自加一操作（以绿色显示），而在图8.4-12-b，由于条件一不满足，故自加一的操作未执行，以白色显示。

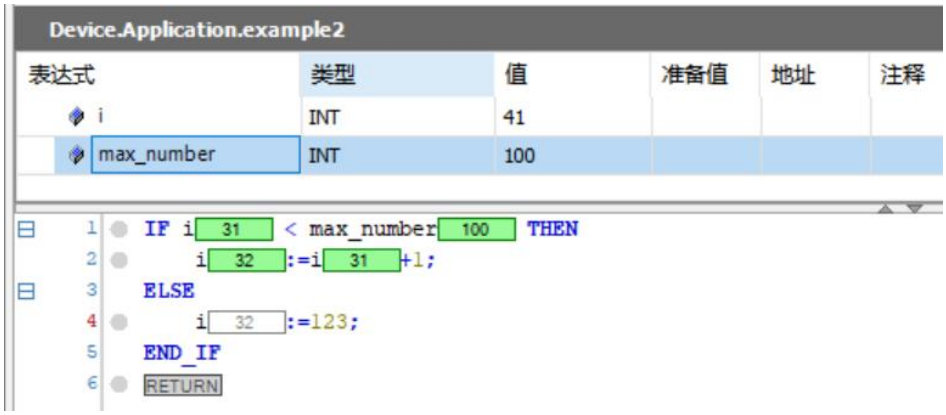


图 8.4-12-a：控制流显示例

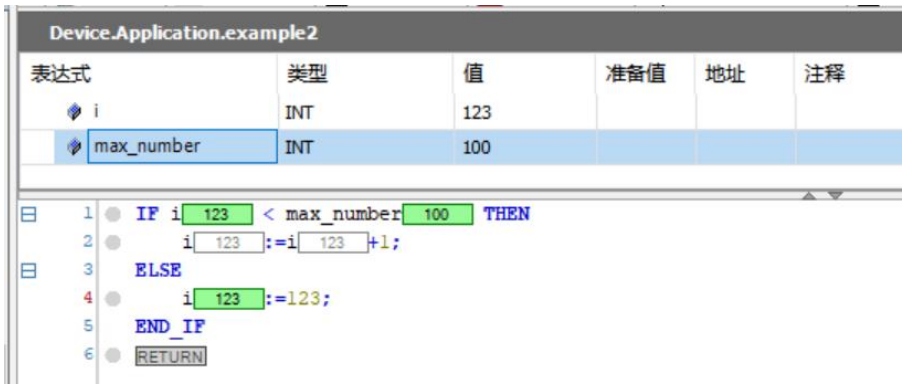


图 8.4-12-b：控制流显示示例

8.4.12 监视管理器

在“视图”菜单下的“监视”目录，可以打开监视窗口对变量进行状态监视，如下图8.4-14所示。在调试过程中，可以在监视窗口中集中监控工程中各程序的变量。CoDeSys默认提供4个普通监视窗口和一个专门用来监控强制变量的窗口。

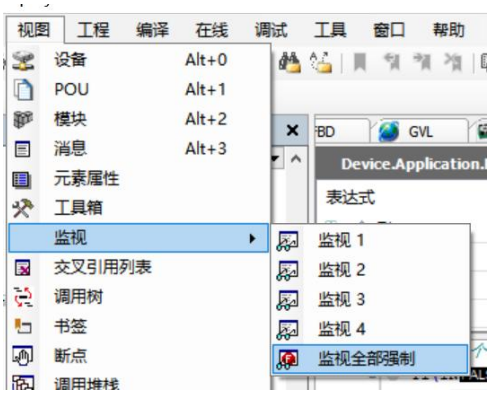


图 8.4-14：打开监视路径

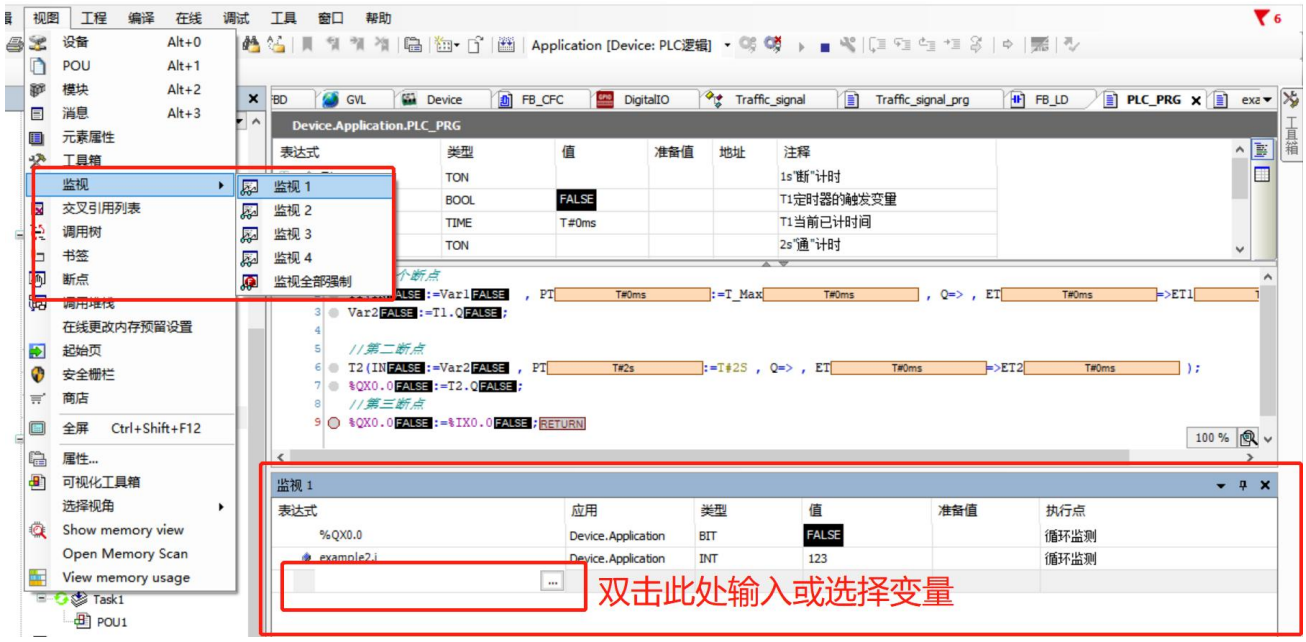


图 8.4-14：打开监视窗口与变量监视

8.4.13 跟踪管理器

为了更好的监控数据变化轨迹，需要对变量述职进行采样跟踪。右击设备树中的“Application”，右可以新建一个跟踪窗口，如下图8.4-15所示。

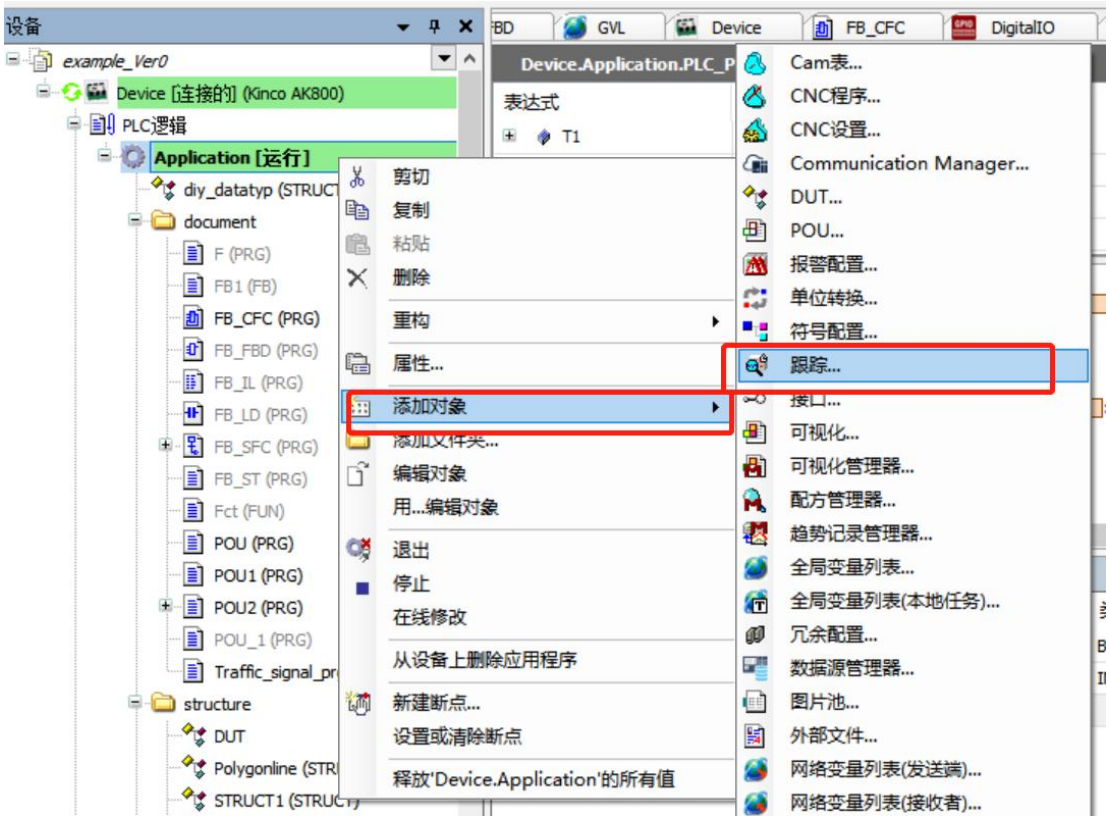


图 8.4-15：创建跟踪采样

双击进入跟踪界面，可以在右侧点击“添加变量”以配置需要跟踪的变量，可以为每个变量定义其跟踪参数，如颜色、线性，如下图 8.4-16：

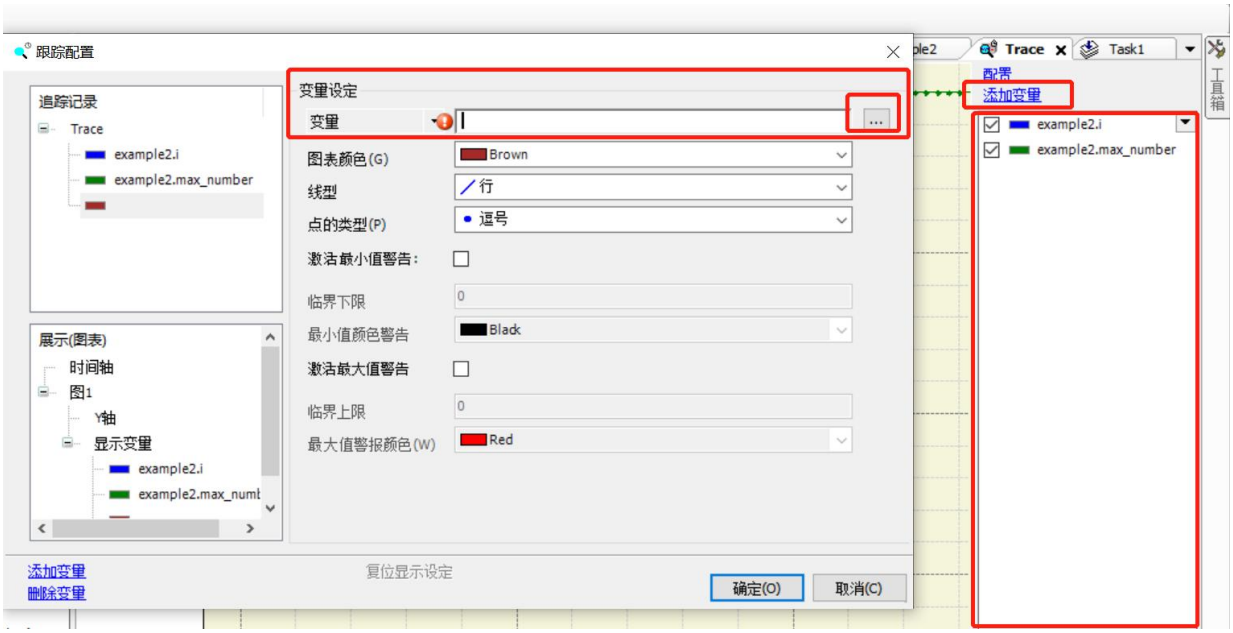


图 8.4-16：跟踪视图“添加变量”

在跟踪窗口右侧的“配置”，可对采样的周期、采样条件和任务等进行配置。如下图 8.4-17：

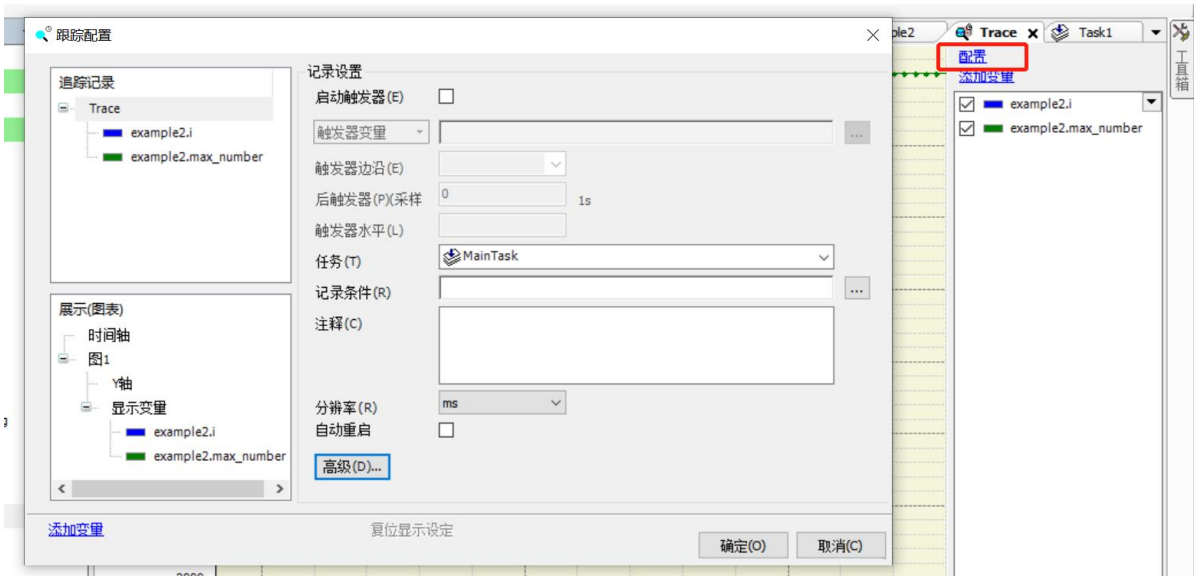


图 8.4-17：跟踪视图“配置”

设置好跟踪变量及采样条件等参数后，在线模式下，即可启动跟踪，工具菜单栏也会出现“跟踪”菜单，“跟踪”菜单下可以执行“启动跟踪”、“停止跟踪”、“下载跟踪”等命令。如下图 8.4-18。

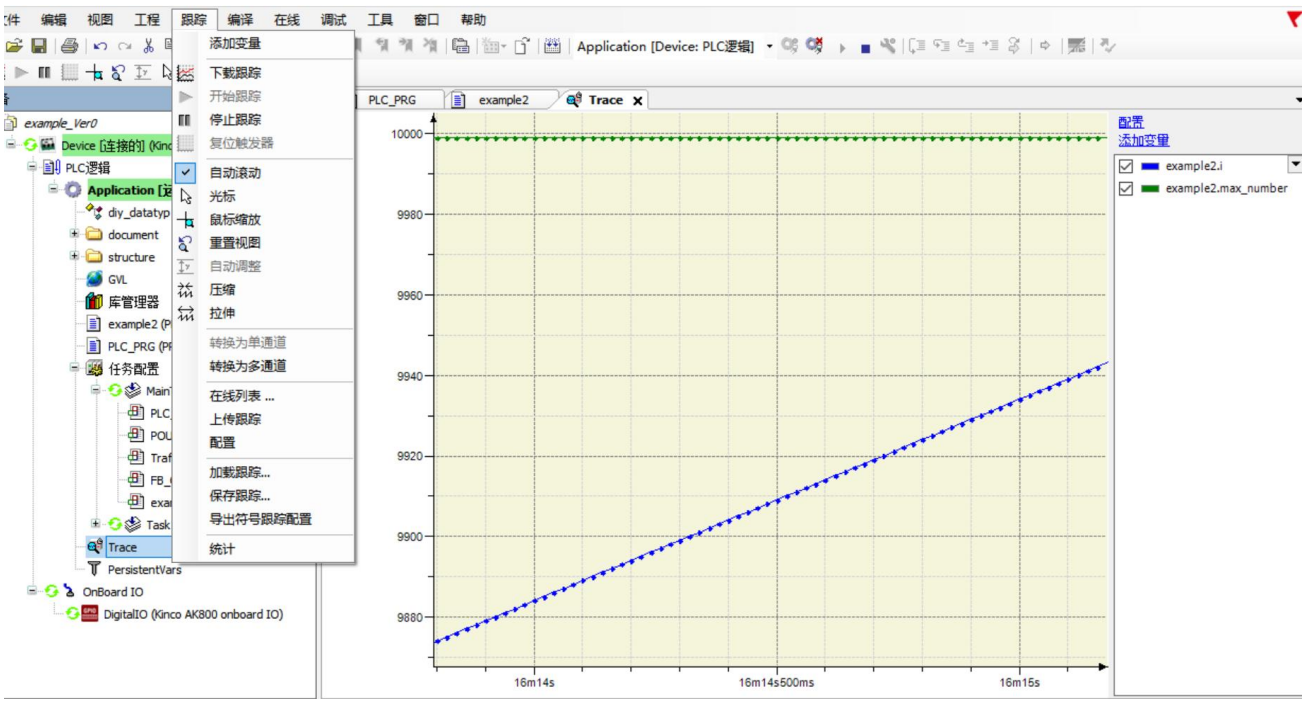


图 8.4-18：跟踪视图下会出现“跟踪”菜单

要想详细了解采样跟踪功能，详情可参看 CoDeSys 软件自带帮助里的相关说明。

第九章 IEC6113 编程基础

CoDeSys 软件遵循国际电工技术委员会（International Electro technical Commission，缩写为IEC）的 IEC 61131-3 标准。本章主要介绍FBD、IL、ST、SFC、CFC及LD六种IEC标准编程语言。具体操作与示例以CoDeSys V3.5软件特性为主。

9.1 梯形图 LD/功能块图 FBD

9.1.1 梯形图 LD

梯形图是 IEC 61131-3 的三种图形化编程语言中的一种，是传统 PLC 使用得最多的图形编程语言。它基于图形表示的继电器逻辑，程序的左、右两侧有两垂直的电力轨线，左侧的电力轨线名义上为功率流从左向右沿着水平梯级通过各个触点、功能、功能块、线圈等提供能量，功率流的终点是右侧的电力轨线。一个触点代表了一个布尔变量的状态，每一个线圈代表了一个实际设备的状态，功能或功能块与 IEC 61131-3 中的标准库或用户创建的功能或功能块相对应。

梯形图中的某些编程元件沿用了继电器这一名称，如线圈、触点等，但是它们不是真实的物理继电器，而是一些存储单元（软继电器），每一软继电器与 PLC 存储器中映像寄存器的一个存储单元相对应。该存储单元如果为 “TRUE”状态，则表示梯形图中对应软继电器的线圈“接通”。如果该存储单元为“FALSE”状态，对应软继电器的线圈和触点的状态与上述的相反，即处于“断开”状态。

CoDeSys 中的 LD 编程界面如下图 9.1-1 所示：

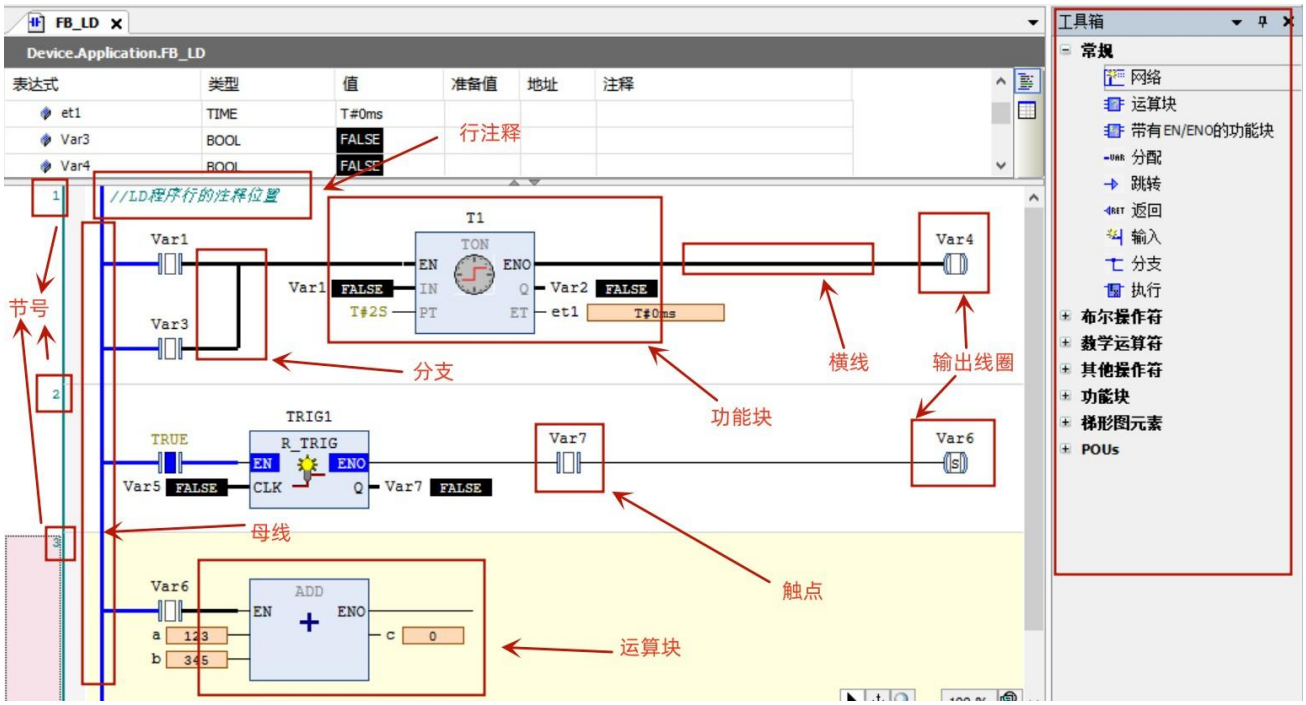


图 9.1-1：梯形图编程界面

9.1.2 功能块图 FBD

FBD 是功能块图（Function Block Diagram）的简称。FBD 是一种图形化的编程语言，与 LD 的结构类似。功能块图用来描述功能、功能块和程序的行为特征，还可以在顺序功能流程图中描述步、动作和转变的行为特征。

功能块图与电子线路图中的信号流图非常相似，在程序中，它可看作两个过程元素之间的信息流。功能块图普遍地应用在过程控制领域。功能块用矩形块来表示，每一功能块的左侧有不少于一个的输入端，在右侧有不少于一个的输出端，功能块的类型名称通常写在块内，但功能块实例的名称通常写在块的上部，功能块的输入输出名称写在块内的输入输出点的相应地方。

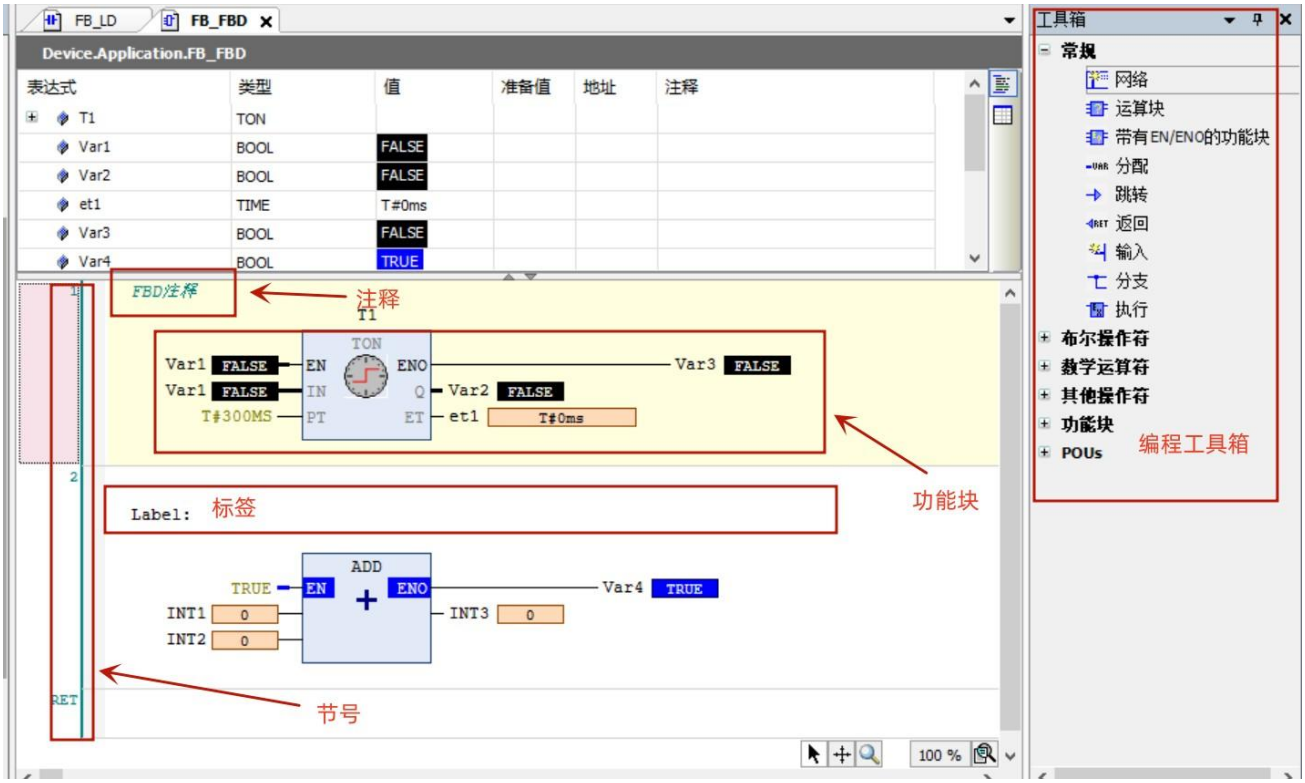


图 9.1-2：功能块图编程界面

9.1.3 连接元素

9.1.3.1 线元素

1. 母线：梯形图电源轨线（Power Rail）的图形元素亦称为母线。其图形表示是位于梯 形图左侧，也可称其为左电源母线。如图 9.1-1 所示。在梯形图编程中，母线侧一直“导通”。

2. 连接线：在无论在 LD 编程还是 FBD 编程中，元件与元件之间通过连接线进行信号传递。连接元素的状态从左向右传递，实现能流的流动。连接线的图形符号有水平线和垂直线（分支），如下图 9.1-3：

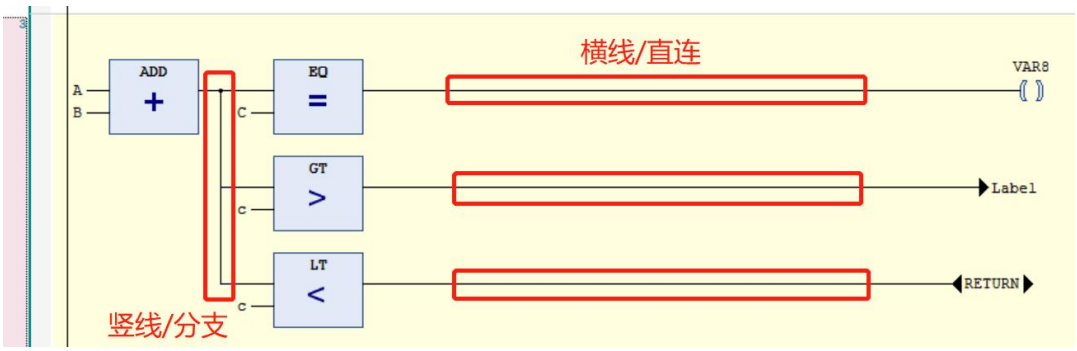


图 9.1-3：线元素

9.1.3.2 节

节是 LD 和 FBD 的基本实体， 在 LD/FBD 编辑器中，节是数值顺序安排的。每个节在左侧开始有一个节编号，并有一个由逻辑或算数表达式，程序，函数，功能块调用，跳转或返回指令组成的结构，见图 9.1-1 和图 9.1-2 图。

一个节同时还可以分配一个标题，注释和标号。节注释打开需要先在“工具”菜单下的“选项”中开启“显示节注释”，如下图。

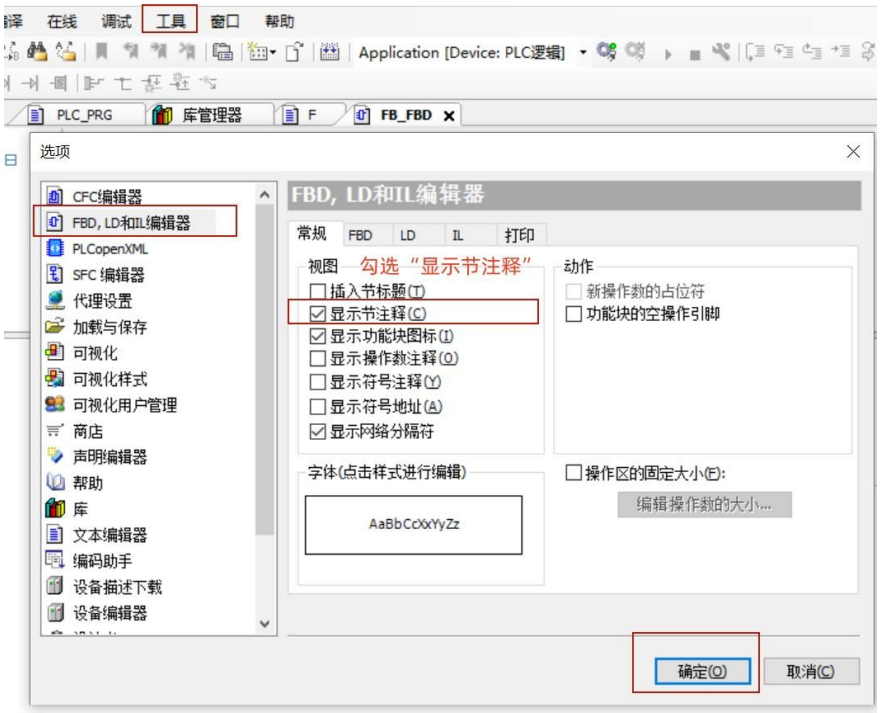


图 9.1-4：打开相应编程语言的注释器

9.1.3.3 标签

标签是一个可选的标识符且当定义跳转时可以确定其地址。它可以包含任何字符。在节区域下，每个 FBD，LD 或 IL 节都有一个文本输入区域来定义一个标号。标号是节的一个选择性辨识符，可以在定义跳转时寻址到标签处， 它可以包含任意顺序的字符。如图 9.1-5 所示，可以通过右击节，选择“插入标号”即可为节添加标签。

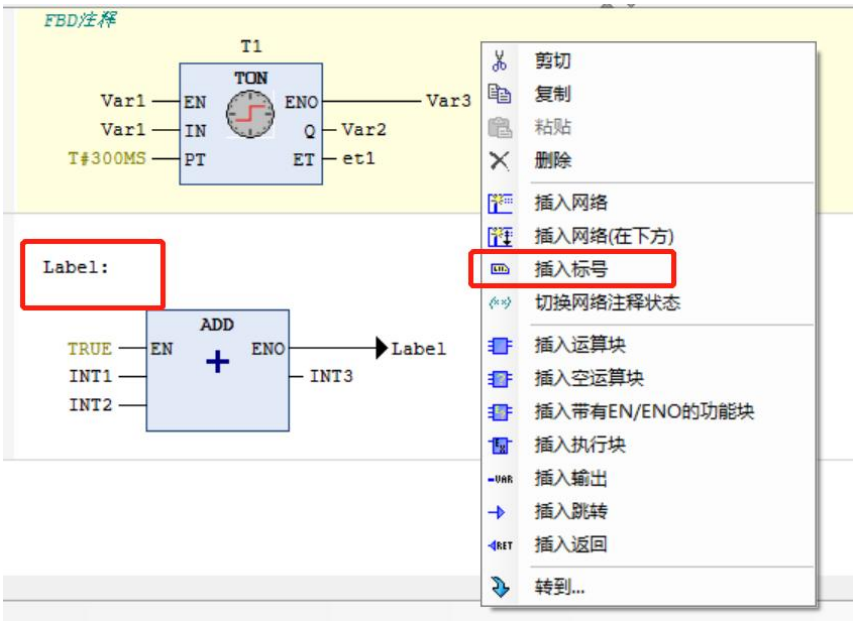


图 9.1-5：FBD 中插入节标号

9.1.3.4 触点/输入

1）LD 触点

触点是梯形图的图形元素。梯形图的触点（Contact）沿用了电气逻辑图的触点术语，用于表示布尔型变量的状态变化。触点是向其右侧水平连接元素传递一个状态的梯形图元素。触点可以分为常开触点（Normally Open Contact，NO）和常闭触点（Normally Closed Contact，NC）。常开触点指在正常工况下，触点断开，其状态为 FALSE。常闭触点指在正常工况下，触点闭合，其状态为 “TRUE”。表 9.1-1 列出了 CoDeSys V3.5 梯形图中常用的触点图形符号及说明。

表 9.1-1： CoDeSys V3.5 常用触点/输入图形符号及说明

类型/名称	图形符号	说明
常开触点		常开触点的无动作状态为“FALSE”，常开触点的动作状态对应当前布尔变量值为 “TRUE” ，这时该常开触点“吸合”，则状态“TRUE” 被传递至该触点右侧，使其右侧连接元素的输入状态为 “TRUE”。 反之，当布尔变量值为“FALSE”时该触点断开，能流不传递至右侧。
常闭触点		开触点的无动作状态为“TRUE”，常开触点的动作状态对应当前布尔变量值为 “FALSE” ，这时该常开触点“断开”，则能流不能被传递至该触点右侧。 反之，当常闭触点为无动作状态时，该触点吸合，能流将传递至右侧。
插入右串联常开触点		可以进行多个触点的串联，在右侧插入触点。多个串联的触点都为吸合状态时，才能使能量流传递到最终目标元件。
插入并联触点（向下）		可以进行多个触点的并联，在触点下侧并联插入常开触点。多个并联触点中只需任意一个触点为“TRUE”，则能流通过平行线传输。
插入并联取反触点（向下）		可以进行多个触点的并联，在触点下侧并联插入常闭触点。常闭触点默认为吸合状态，多个常闭触点并联时，只有所有触点都处于动作状态“TRUE”时，才能阻止能流传递。

插入并联触点（向上）		可以进行多个触点的并联，在触点上侧并联插入常开触点。多个并联触点中只需任意一个触点为“TRUE”，则能流通过平行线传输。

2）FBD 输入

在 FBD 中，没有“触点”的概念，只有变量输入元件（符号为：  ），FBD 中变量的并联可以通过“AND”运算块功能实现，变量串联功能可以通过“OR”运算块功能实现。

变量的取反可以通过“取反”（符号为：  ）功能实现，运算块的符号为：  。如下图 9.1-6 所示：

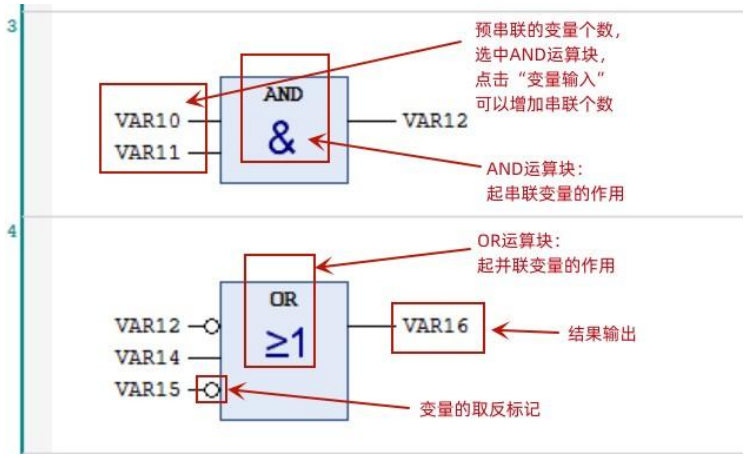


图 9.1-6：FBD 变量的串并联方式

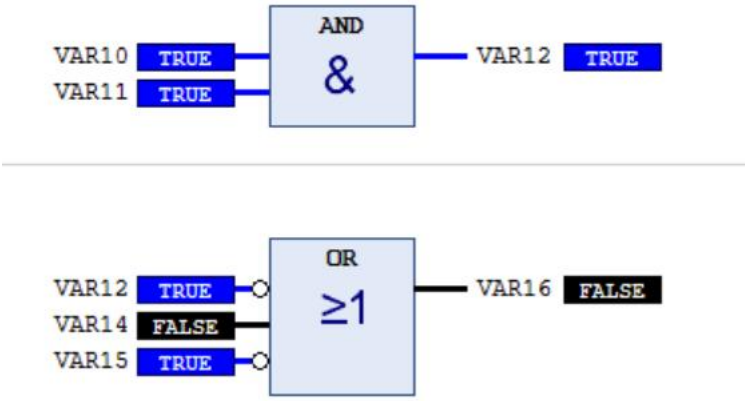


图 9.1-7：FBD 变量的串并联方式效果图

9.1.3.5 线圈

1）LD 中的输出线圈

线圈是梯形图的图形元素。梯形图中的线圈沿用了电气逻辑图的线圈术语，用于表示布尔型变量的状态变化。根据线圈的不同特性，可以分为瞬时线圈和锁存线圈，锁存线圈分为置位线圈和复位线圈。表 9.1-2 列出了 CoDeSys V3.5 中梯形图中常用的线圈图形符号及说明。

表 9.1-2：CoDeSys V3.5 中梯形图中常用的线圈图形符号及说明

类型/名称	图形符号	说明
输出线圈		根据其左侧连接的所有元件的状态的总和进行实时输出。如果线圈左侧连接元素的组合状态为“TRUE”，则线圈的状态为“TRUE”，反之线圈状态为“FALSE”。
变量输出		在 CoDeSys V3.5 梯形图中，此项功能与输出线圈的作用一致。
置位线圈		根据其左侧连接的所有元件的状态的总和进行保持输出。即当线圈左侧连接元素的组合状态为“TRUE”时，该置位线圈（SET）状态被置为“TRUE”并一致保持置位状态，直到该线圈被复位线圈（RESET）所复位。反之亦然。一般置位线圈要和复位线圈配合使用。
复位线圈		根据其左侧连接的所有元件的状态的总和进行保持输出。即当线圈左侧连接元素的组合状态为“FALSE”时，该置位线圈（RESET）状态被置为“FALSE”并一致保持复位状态，直到该线圈被置位线圈（SET）所复位。反之亦然。一般置位线圈要和复位线圈配合使用。

双线圈：

在用户程序中，同一个线圈使用了两次或两次以上的现象称为双线圈输出。如图 9.1-8a 中，有 2 个变量“Var0”的输出线圈，在同一个扫描周期，两个线圈的逻辑运算结果可能刚好相反，即变量“Var0”的线圈一个可能“通电”，另外一个可能“断电”。对于变量“Var0”控制来说，真正起作用的是最后一个“Var0”的线圈的状态。“Var0”的线圈的通断状态除了对外部负载起作用外，通过它的触点，还可能对程序中别的变量的状态产生影响。所以一般应避免出现双线圈输出现象，应尽量如图 9.1-8b 中采用的并联方式来解决双线圈问题。

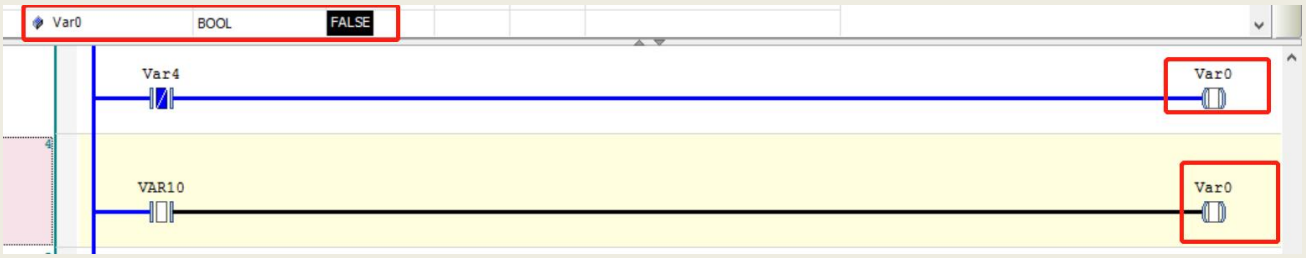


图 9.1-8a：双线圈输出示例

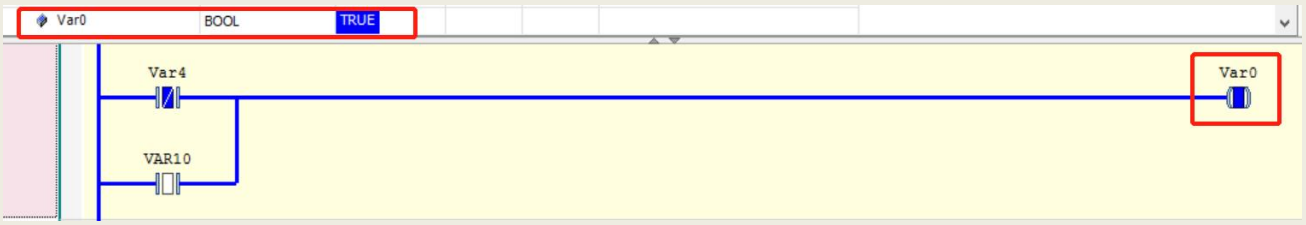


图 9.1-8b：双线圈输出修正

上升沿和下降沿：

上升沿和下降沿跳变线圈指的是在其左侧水平连接元素的状态总和从“FALSE”跳变至“TRUE”（上升沿）或从“TRUE”跳变到“FALSE”（下降沿），将有关线圈的变量保持一个求值周期（即一个扫描周期），其他时间将其左侧水平连接元素状态传递至其右侧连接水平元素。

2）FBD 中的变量输出

在 FBD 中，没有“线圈”输出的说法，但是 FBD 中的“插入输出”功能可以实现变量输出的功能，功能符号为：  。另外，在 FBD 中可以通过“插入置位/复位”（符号为：  ）功能来实现变量的置位和复位，具体的操作模式为：选中普通的输出变量，点击一次“插入置位/复位”符号，会将输出功能变为“置位输出”，同时显示“  ”符号；选中置位功能的变量后再点击一次“插入置位/复位”，则可以将“置位输出”更改为“复位输出”，同时显示“  ”符号，如下图 9.1-9 所示：

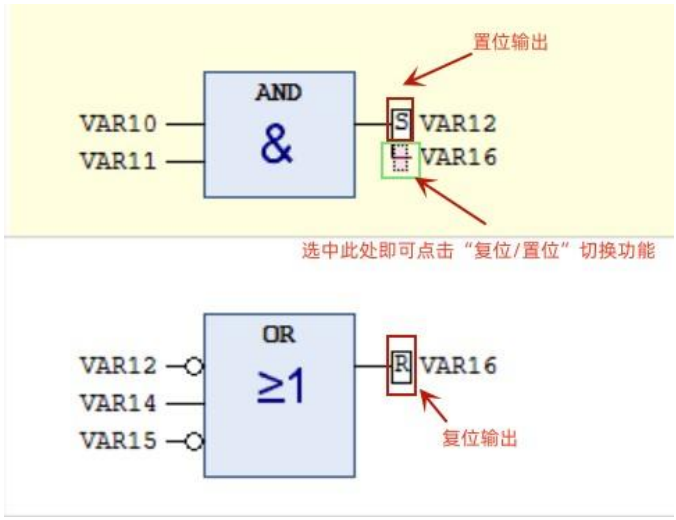


图 9.1-9：FBD 的复位置位功能

9.1.3.6 函数及功能块

如要实现函数或功能块的调用就要用到运算块，运算块可以代表所有的 POU，包括功能块、函数甚至包括程序。功能块如计时器，计数器等。可以在 FBD，LD 的节中插入。运算块可以有任意的输入，任意输出。函数及功能块的图形符号说明详见表 9.1-3。

表 9.1-3：CoDeSys V3.5 中的运算块图形符号及说明

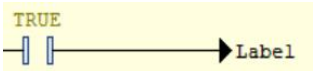
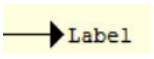
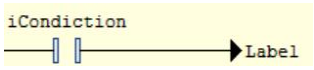
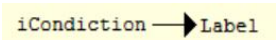
类型/名称	图形符号	说明
运算块		插入函数或功能块，点击后会自动弹出“输入助手”，在“关键字”中选择需要的运算块/功能块即可，该功能适合对函数和功能块不太熟悉者使用。
空运算块		插入函数或功能块，点击后会自动弹出在节处生成一个空的运算块，用户需要自行在运算块处输入功能关键字（功能块还需要定义功能块名称），该功能要求对函数/功能块关键字有一定的熟悉。

带 EN/ENO 的运算块		与直接插入“运算块”不同，该功能只有当“EN”（使能）端为“TRUE”时才执行函数或功能块并允许将能流传递至下游。点击后会自动弹出“输入助手”，在“关键字”中选择需要的运算块/功能块即可。
带 EN/ENO 的功能块		插入带“ EN/ENO”的功能块，只有当“EN”（使能）端为“TRUE”时才执行功能块并允许将能流传递至下游。用户需要自行在运算块处输入功能块关键字以及在功能块上方输入功能块的实例化名称。

9.1.3.7 跳转和返回

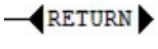
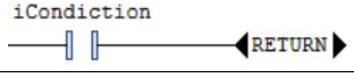
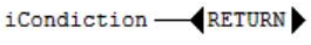
1）跳转设置（JMP）：在程序执行时将跳转到目标位置继续处理。在程序组织单元中，跳转目标是发生跳转的该程序组织单元内的一个标号。它表示跳转发生后，程序将从该目标开始执行。跳转指令的操作数是标号，不是操作数对应的数据存储单元地址。必须通过标签定义此目标位置。跳转执行的功能符号为：  ，为此，在  处输入标签字段，也可以打开“输入助手”进行选择。跳转执行的元素可以有以下几种，通过表 9.1-4 展开说明。

表 9.1-4：跳转元素说明

类型/名称		图形符号	说明
无条件跳转	LD		直接无条件跳转至 Label。
	FBD		
条件跳转	LD		当条件成立时跳转至 Label。
	FBD		

2）跳转返回（RETURN）：返回指令是没有操作数的指令，用于调用函数、功能块及程序的返回，当条件跳转返回的布尔输入量为“TRUE”时，程序执行将跳转返回到调用的实体。当布尔输入为 “FALSE” 时，程序执行将继续在正常方式进行。

表 9.1-5：跳转返回元素说明

类型/名称		图形符号	说明
无条件跳转返回	LD		直接无条件返回。
	FBD		
条件跳转返回	LD		当条件成立时返回。
	FBD		

9.2 指令列表 IL

指令列表（IL）是一种汇编语言风格的编程语言，程序不易阅读，但执行速度最快。IL语言包含一系列的指令，每条指令占据一行，包含一个运算符和一个或多个用逗号隔开的操作数。操作数之间用逗号分隔。每行开始可以有标签，标签后要有冒号。每行结束可以有注释，注释用“（*...*）”括起来。每行指令之间可以插入空行。IL 编辑器是一种文本编辑器，具有常见Windows 文本编辑器的功能，点击菜单栏或鼠标右键可以进行编辑。

若在新建POU中无法找到IL编程语言，需在“工具”菜单下的选项中使能IL编程器，如下图9.2-1所示：

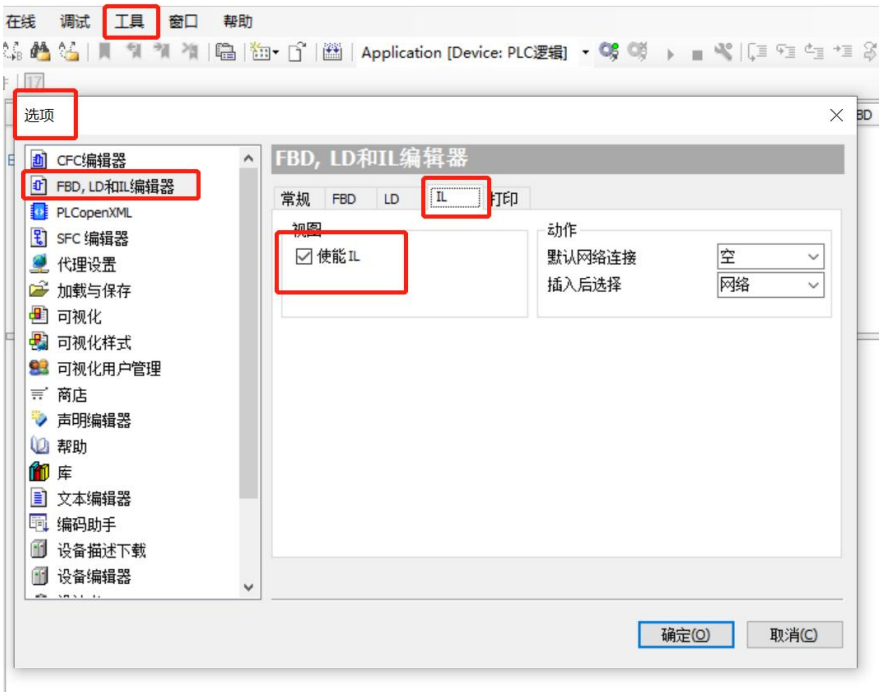


图 9.2-1：打开 IL 编辑器

9.2.1 IL 编程界面

IL 编程界面如下图 9.2-2 所示。指令表包含一系列指令，每个指令在新一行开始，包含操作符，根据操作的类型，一个或多个操作域用逗号分开，操作符可以用修饰符扩展。表 9.2-1 介绍了常用操作符和修饰符的含义。

图 9.2-2：IL 编程界面

表 9.2-1：IL 编程常用操作指令符

运算/操作符	修饰符	意义	符号	举例
LD	N	将操作数赋予当前值（取反）		
ST	N	将当前值赋予操作数（取反）		
S		当累加器中的数为真，将操作数（布尔型）设为真		
R		当累加器中的数为假，将操作数（布尔型）设为假		
AND	N、(	位逻辑运算符		
OR	N、(	位逻辑运算符		
XOR	N、(	位逻辑运算符		
ADD		加		
SUB	(	减		
MUL	(	乘		
DIV	(	除		
GT	(	大于判断	>	
GE	(	大于等于判断	>=	
EQ	(	等于判断	=	
NE	(	不等于判断	<>	
LE	(	小于等于判断	<=	
LT	(	小于判断	<	
JMP	CN	转移到标签		

CAL	CN	调用其它 POU		
RET		退出 POU 并且返回到调用处		
	C			
	CN			
)		计算延迟操作		

更详细的 IL 操作指令符号请参照 CoDeSys 帮助网站：https://content.helpme-codesys.com/

具体路径如下：CODESYS Essentials→Reference: Programming→Programming Languages and Editors→Function Block Diagram / Ladder Diagram / Instruction List (FBD/LD/IL)→Modifiers and Operators in IL

9.2.2 程序举例

举例

下面是一个用IL 语言实现的简单运算程序。

```
声明：
VAR
A：REAL；
B：REAL；
C：REAL；
END_VAR
程序：
LD      10      （*将数字 10 赋予当前值*）
ADD     A       （*当前值与变量 A 加运算后结果存入当前值*）
GE      B       （*当前值与变量 B 进行大于等于比较*）
JMPC    Next1   （*上一表达式结果为真，则跳转至标志 Next1 处*）
LD      A       （*将变量 A 的值赋予当前值*）
ADD     B       （*当前值与变量 B 加运算后结果存入当前值*）
ST      C       （*将当前值赋予变量 C*）
JMP     Next2   （*无条件跳转至标志 Next2 处*）
Next1：         （*标志*）
LD      A       （*将变量 A 的值赋予当前值*）
SUB     B       （*当前值与变量 B 减运算后结果存入当前值*）
ST      C       （*将当前值赋予变量 C*）
Next2：         （*标志*）
```

IL 语言支持两种修饰符：C和N。C 表示条件执行，只有当前一个表达式的值为（TRUE）时，指令才被执行。N 与JMP、CAL 和RET 连用时表示条件非执行，只有当前一个表达式的

值为假（FALSE）时，指令才被执行；其它情况下，表示操作数取负。表9-2-1 列出了IL的所有运算符以及它们的修饰符和相应意义。

以下是利用修饰符进行IL 编程的例子：

```
LD      TRUE        (*将TRUE 赋予当前值*)
ANDN    BOOL1       (*BOOL1 变量取反后与当前值进行与运算*)
JMPC    mark        (*如果结果是真，那么跳转到标签“mark”处*)
LDN     BOOL2       (*将BOOL2 取反后赋予当前值*)
ST      ERG         (*将当前值存入ERG*)
Mark：
LD      BOOL2 (*将BOOL2 赋予当前值 *)
ST      ERG (*将当前值存入ERG*)
```

如果在运算符之后插入括号，那么括号里的值可以看成是一个操作数。

例如：

```
LD      2
MUL     2
ADD     3
ST      ERG
```

运行后ERG的值是7。但是，如果加入一对圆括号：

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

运行后ERG的值是10，MUL 运算符只有到“）”才执行，等同于执行MUL 5。

举例：

IL 语言简单应用示例如图所示。本程序产生“1s 断2s 通”的脉冲信号

```
0001    (*当I0.0接通时，T1会产生1秒端，2s通的脉冲*)
0002    LD      %IX0.0
0003    ANDN    N
0004    ST      T1.IN
0005    CAL     T1(PT:=T#1000ms)
0006    LD      T1.ET
0007    ST      ET
0008    LD      T1.Q
0009    ST      M
0010    (*调整接通时间*)
0011    LD      M
0012    ST      T2.IN
0013    CAL     T2(PT:=T#2000ms)
0014    LD      T2.ET
0015    ST      ET1
0016    LD      T2.Q
0017    ST      N
0018    (*输出*)
0019    LD      M
0020    ST      %QX0.0
```

图 9.2-3：IL 编程举例

9.3 结构化文本 ST

结构化文本（ST）可以执行选择语句（IF…THEN…ELSE）和循环语句（WHILE…DO），类似于PASCAL 和BASIC 等高级语言。

9.3.1 ST 表达式

• ST 语言中的表达式由运算符和操作数组成。操作数可以是常量、变量、函数调用或另一个表达式。表达式的计算通过执行具有不同优先级的运算符完成。有最高优先级的运算符先被执行，然后依次执行下一个优先级的运算符，直到所有的运算符被处理完。有相同优先级的运算符按从左到右的顺序执行。ST 语言的运算符如表所示。更详细的 IL 操作指令符号请参照 CoDeSys 帮助网站：https://content.helpme-codesys.com/

9.3.2 ST 指令

ST 语言的指令如表9.3-1 所示。

表 9.3-1：ST 操作指令表

指令类型	例子
赋值	A:=B; CV := CV + 1; C:=SIN(X);
调用功能块并且赋初值	TP(IN:= %IX0.5, PT:=t#30);
使用功能块输出	A:=TP.Q;
返回	RETURN;

IF	D:=B*B;
	IF D<0.0 THEN
	C:=A;
	ELSE IF D=0.0 THEN
	C:=B;
	ELSE
	C:=D;
	END_IF

CASE	CASE INT1 OF 1: BOOL1 := TRUE; 2: BOOL2 := TRUE; ELSE BOOL1 := FALSE; BOOL2 := FALSE; END_CASE

FOR 循环	J:=101;
	FOR I:=1 TO 100 BY 2 DO
	IF ARR[I] = 70 THEN
	J:=I;
	EXIT;
	END_IF

WHILE 循环	J:=1; WHILE J<= 100 AND ARR[J] <> 70 DO J:=J+2; END_WHILE
REPEAT 循环	J:=-1; REPEAT J:=J+2; UNTIL J= 101 OR ARR[J] = 70 END_REPEAT
退出程序	EXIT;
空指令	;

1.赋值：

执行赋值操作时，等号左边是操作数（变量或地址），右边是被赋予的表达式的值。例如：Var1 := Var2 * 10;

2.调用功能块：

通过写入功能块实例的名字，并且在随后的圆括号中赋给参数值来调用功能块。

举例：

```
变量声明：
TPInst:TP;
VarBOOL1: BOOL;
VarBOOL2: BOOL;
程序：
TPInst(IN:= VarBOOL1,PT:= T#5s);（*参数IN 和PT 设定时钟脉冲的触发信号和高电平的长度*）
VarBOOL2:=TPInst.Q;（*输出脉冲值Q 赋给变量VarBOOL2*）
```

3.返回指令：

返回指令可以根据条件退出POU。

4.IF 指令：

使用IF指令可以检查条件，根据条件执行相应的指令。

语法：

```
IF <逻辑表达式> THEN
      <IF  指令>
{ELSIF <逻辑表达式1> THEN
      <ELSE IF  指令1>
ELSIF <逻辑表达式  n> THEN
      <ELSE IF  指令  n>
ELSE
<ELSE  指令>}
END_IF
```

其中{}的部分可选。

如果<逻辑表达式>返回TRUE，那么只有<IF 指令>被执行，其它的指令不被执行。同样，从<逻辑表达式1>开始，相继执行逻辑表达式，直到其中一个表达式返回TRUE 为止，返回TRUE 的逻辑表达式对应的指令被执行。

如果没有逻辑表达式生成TRUE，那么只有<ELSE 指令>被执行。举例如下:

```
IF temp<17 THEN
      heating_on ：= TRUE;
ELSE
      heating_on ：= FALSE;
END_IF
```

这里，如果温度低于17 度，打开加热器，反之则保持关闭状态。

5.CASE 指令：

使用CASE 指令，可以在结构中用一个相同的条件变量表示几个条件指令。语法如下：

```
CASE <Var1> OF
      <Value1>:                    <指令1>
      <Value2>:                    <指令2>
      <Value3, Value4, Value5>:  <指令3>
      <Value6 .. Value10>:         <指令4>
      ...
      <Value n>:                   <指令n>
      ELSE
          <ELSE 指令>
END_CASE
```

CASE 指令根据下面的模型来执行：

- 如果变量<Var1>有值<值 i>，那么<指令i>被执行。
- 如果变量<Var1>没有任何指定的值，那么<ELSE 指令>被执行。
- 如果变量的几个值都需要执行相同的指令，那么可以把几个值相继写在一起，用逗号隔开。
- 对于变量的一个范围需要执行相同的指令，可以写入初值和终值，中间用两个点分开。

举例：

```
CASE INT1 OF
    1,5:    BOOL1  := TRUE;
            BOOL3  := FALSE;
    2:      BOOL2  := FALSE;
            BOOL3  := TRUE;
    10,20:  BOOL1  :=TRUE;
            BOOL3  := TRUE;
    ELSE
            BOOL1 := NOT BOOL1;
            BOOL2 := BOOL1 OR BOOL2;
    END_CASE
```

6.FOR 循环：

使用FOR 循环，可以编写循环过程。语法：

```
FOR <INT_Var> := <INIT_VALUE> TO <END_VALUE> {BY <Step Size>} DO
<Instructions>
END_FOR
其中{}的部分可选。
```

只要计数<INT_Var>不大于<END_VAULE>，指令就会被执行。指令执行之前，首先检查这个条件，如果<INIT_VALUE>大于<END_VALUE>，指令就永远不会被执行。

当指令被执行时，<INT_Var>总是增加步长<Step Size>。步长可以是任意的整数值。如果不写步长，缺省值是1。当<INT_Var>大于<END_VALUE>时，循环结束。举例：

```
FOR Counter :=1 TO 5 BY 1 DO
    Var1 :=Var1*2;
END_FOR
Erg :=Var1;
假定Var1的缺省值是1，那么循环结束后，Var1的值为32。
```

注意：<END_VALUE>一定不能等于计数变量<INT_VAR>的极限值。如果计数变量的类型是SINT（范围从-128-127），<END_VALUE>是127，则会进入死循环。

7.WHILE 循环：

WHILE 循环用起来很象FOR循环，不同的是，结束条件可以是任意的逻辑表达式。指定一个条件，当条件满足的时候，循环被执行。语法：

```
WHILE <逻辑表达式>
    <指令>
END  WHILE
```

只要<逻辑表达式>的值返回TRUE，<指令>就会被重复执行。如果在第一次计算时，<逻辑表达式>的值已经是FALSE，那么指令永远不会被执行。如果<逻辑表达式>的值永远不会是FALSE，那么<指令>被无休止的执行，产生一个相对时间延迟。举例：

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

在一定意义上，“WHILE”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。然而，如果清楚知道循环次数，那么FOR 循环更好。

8.REPEAT 循环:

“REPEAT”循环不同于“WHILE”循环，因为只有在指令执行以后才检查中断条件。无论结束条件怎样，循环至少执行一次。语法：

```
REPEAT
<指令>
UNTIL <逻辑表达式>
END_REPEAT
```

直到<逻辑表达式>的值返回TRUE，<指令>才停止执行。如果在第一次计算时，<逻辑表达式>产生TRUE，那么<指令>只被执行一次，如果<逻辑表达式>不会产生TRUE，那么<指令>将无休止的循环，导致相对时延。举例：

```
REPEAT
Var1    := Var1*2;
Counter:= Counter-1;
UNTIL Counter = 0
END_REPEAT
```

9.EXIT 指令:

如果EXIT 指令出现在FOR、WHILE、REPEAT 循环中，那么不管中断条件如何，EXIT 出现时循环终止。举例：

下面是一个用ST 语言实现简单运算的小程序：

变量声明：

```
PROGRAM Program
VAR
    A    :BOOL;
    B    :BOOL;
    C    :INT;
END_VAR
```

编程：

```
IF A THEN
    C :=20;
ELSIF B THEN
    C :=30;
ELSE
    C :=50;
END_IF
```

9.4 顺序功能图 SFC

SFC 是顺序功能图（Sequential Function Chart）的简称，是一种图形化的编程语言，用来描述程序中不同动作的时间顺序。SFC 由一系列的步和转移组成，如图所示：步定义动作，转移控制顺序。

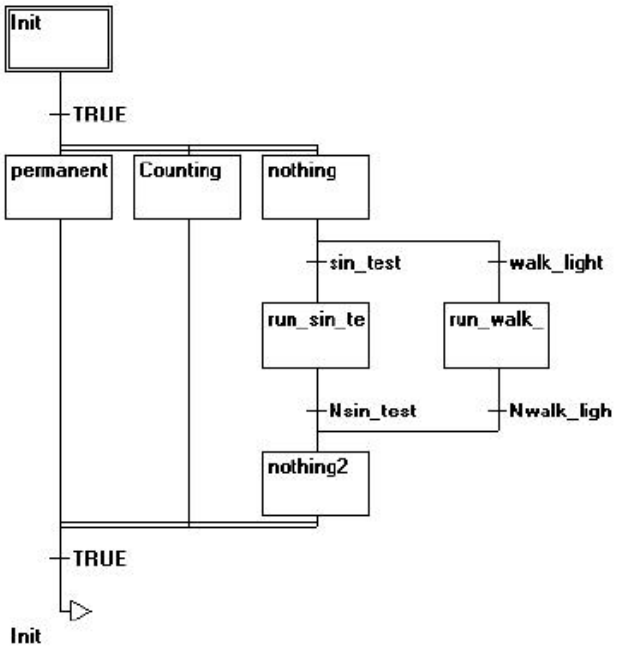


图 9.4-1：SFC 流程图

9.4.1 基本概念

1.单步:

SFC 语言包含一系列的步，这些步通过有向的连接彼此联系。步分为简化步和IEC步。简化步包含一个动作和一个显示步动作的标志。如果步包含动作，则在步的右上角出现一个小三角形。IEC步包含一个或多个动作和逻辑变量。新插入的步是否为IEC步，取决于是否选择了“高级”>“使用IEC步（I）”命令。必须添加特殊的SFC库“Iecsfc.lib”才能够使用IEC步。

2.动作:

动作是使用其它语言实现的一系列指令，可以是用IL或ST语言实现的指令语句，也可以是用LD、FBD 或SFC实现的网络。

对于简化步，动作总是和步直接相关。用鼠标双击动作所属的步，进行编辑。

对于IEC步，在对象组织器中选中SFC程序，点击右键，选择“添加动作”命令创建新动作。可以赋给IEC步多个动作，同时这些动作也可以被多个步重复使用。

IEC步的动作显示在步的右边框中。左边字段包含可能有时间常量的限定符，右边字段包含动作名即逻辑变量名。

IEC的动作分散在步中。在它们所属的POU中，这些动作可以被重复使用。使用“高级”>“关联动作”命令添加动作，使用“高级”>“清除动作或转移”命令删除已添加的动作。

包含两个动作的IEC步举例，如图所示。

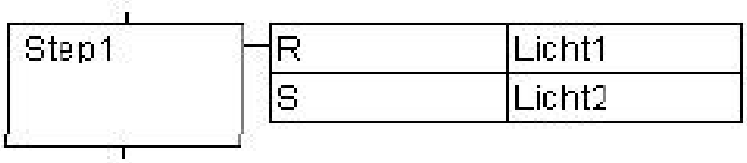


图 9.4-2：SFC 中包含 2 个动作的 ICE 步举例

3.入口动作和出口动作:

“添加入口动作”和“添加出口动作”命令可以在步中加入入口动作和出口动作。

- 入口动作只有当步在活动状态时立即执行一次。
- 出口动作只在步不活动之前执行一次。

入口动作的步在左下角有“E”标志进入，有出口动作的步右下角有“X”标志退出。可以用任意语言实现入口和出口动作，鼠标双击步的相应角，即可编辑入口或者出口动作。入口和出口动作步的举例，如下图所示。

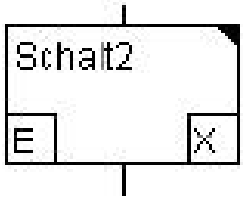


图 9.4-2：SFC 中动作标志

4.转换/转换条件:

步之间的切换就是转换。只有当步的转换条件为真时，步的转换才进行。即前步的动作执行完后，如果有出口动作则执行一次出口动作，后步如果有入口动作则执行一次后步的入口动作，然后按照控制周期执行该活动步的所有动作。

转换条件可以是逻辑变量、逻辑地址、逻辑常量或者是由其它语言编程实现的逻辑。

5.活动步:

调用POU之后，初始化步（双边的步）的动作首先执行。动作被执行的步称为活动步。每次循环执行活动步的动作。在线模式下，活动步以蓝色显示。为了更容易地跟随过程，联机模式下，所有的活动动作象活动步一样以蓝色显示。每次循环以后做一次检查，查看哪个动作是活动的。在一个控制周期中，执行活动步的所有动作。之后，如果下步的转换条件是TRUE，下步将变成活动步，在下个周期执行。

6.限定符:

把IEC步和动作关联，需要使用限定符。限定符对应的含义如表所示：

表 9.4-1：限定符

说明	名称	说明	备注
N	非存储	普通型： 动作随步活动，当步开始激活时，动作开始执行；当步退出时，动作停止执行。	
S	设置存储	置位型： 当步开始激活时，动作开始执行；当步退出时，动作还将继续执行。一直等到有 R 限定符给它复位之后，动作 才将停止执行。在动作没有被 R 停止之前，任何限定符也不能使得动作停止（包括 P 限定符）。	
R	重置	复位型：当步开始激活时，动作停止执行。	
L	时限	限时结束型：动作在一定时间被激活。最长时间是步活动时间。当步开始激活时，动作开始执行；同时开始计时，当时 间到达设定值后，动作将停止执行。如果时间还没有到达设 定值时，步的转移条件已经满足，此时，动作也将停止执行。	要加时间常数（设定值）
D	延时	延时开始型： 当步开始激活时，开始计时，当时间到达设定值后，动作将开始执行；当步退出时，动作停止执行。如果 时间还没有到达设定值时，步的转移条件已经满足，在这种 情况下，动作将不可能被执行。	要加时间常数（设定值）
P	脉冲	脉冲型： 当步开始激活时，动作开始执行，并且动作只被执行一次，之后动作将停止执行。	
SD	存储和延时	延时绝对开始置位型： 当步开始激活时，开始计时，当时间到达设定值后，动作将开始执行；当步退出时，动作还将执 行，一直等到有 R 限定符给它复位之后，动作才将停止执行。 在动作没有被 R 停止之前，任何限定符也不能使得动作停止（包括 P 限定符）。如果时间还没有到达设定值时，步的转 移条件已经满足，在这种情况下，计时还在进行，一直到时 间达到设定值后，动作将被执行，此时不管当前步还是不是原来的步了。	要加时间常数（设定值）

DS	延时和存储	延时开始置位型： 当步开始激活时，开始计时，当时间到达设定值后，动作将开始执行；当步退出时，动作还将执行， 一直等到有 R 限定符给它复位之后，动作才将停止执行。在 动作没有被 R 停止之前，任何限定符也不能使得动作停止（包括 P 限定符）。如果时间还没有到达设定值时，步的转 移条件已经满足，在这种情况下，动作将不可能被执行。	要加时间常数（设定值）
SL	存储和时限	绝对限时结束型： 在一定时间内动作是活动的。当步开始激活时，动作开始执行；同时开始计时，当时间到达设定值后， 动作将停止执行。如果时间还没有到达设定值时，步的转移 条件已经满足，此时，动作还将继续执行，直到当时间到达 设定值后动作才会停止执行。限制符 L、D、SD、DS 和 SL 需要 TIME 常量格式的时间值。	要加时间常数（设定值）

7.SFC中的隐含变量:

在SFC中可以隐含声明的变量。每步的标志都存储了步的状态。对于简化步，步标志为<步名>。对于IEC步，步标志为<步名>.x。当步是活动状态的时候，步标志的值为TRUE。当步是非活动状态的时候，步标志的值为FALSE，因此可以使用变量<动作名>.x 询问动作是否是活动，也可以使用隐含变量<步名>.t 用来询问步的活动时间。

8.SFC标志:

在SFC中，步的活动状态时间取决于它的属性状态时间，有时会设置一些特殊标志。同样，可以设置变量来控制顺序功能图中的程序流。要使用这些标志，必须在全局或局部变量中作为输入或者输出变量声明这些标志。

SFCenableLimit：BOOL型变量，变量值是TRUE时，步的超时将会注册在SFCWError中，忽略其它的超时。

SFCInit：BOOL型变量，变量值是TRUE时，顺序功能图返回到初始化步，同时其它的SFC标志重置。只要变量有TRUE 值，初始化步保持活跃，但不被执行。只有当SFCInit被重新设置为FALSE，功能块才恢复正常执行。

SFCQuitError：BOOL型的变量，变量值是TRUE 时，SFC图停止执行。在变量SFCError中，重置一个可能的超时。当变量重新获得FALSE时，在活动步中所有以前的时间重置。

SFCPause：BOOL型变量，变量值是TRUE时，SFC图停止执行。

SFCError：当超时发生在SFC图中时，设置这个逻辑变量。如果一个超时发生而变量SFCError不重置，则不会注册其它超时。

SFCTrans：BOOL型变量。当转换执行时，变量值为TRUE。

SFCErrorStep：STRING 型变量，存储一个导致超时发生的步名。

SFCErrorPOU：STRING 型变量，包含一个发生了超时的块名。

SFCCurrentStep：STRING 型变量，步名存储在活动变量中，不受时间监视。一旦在同时

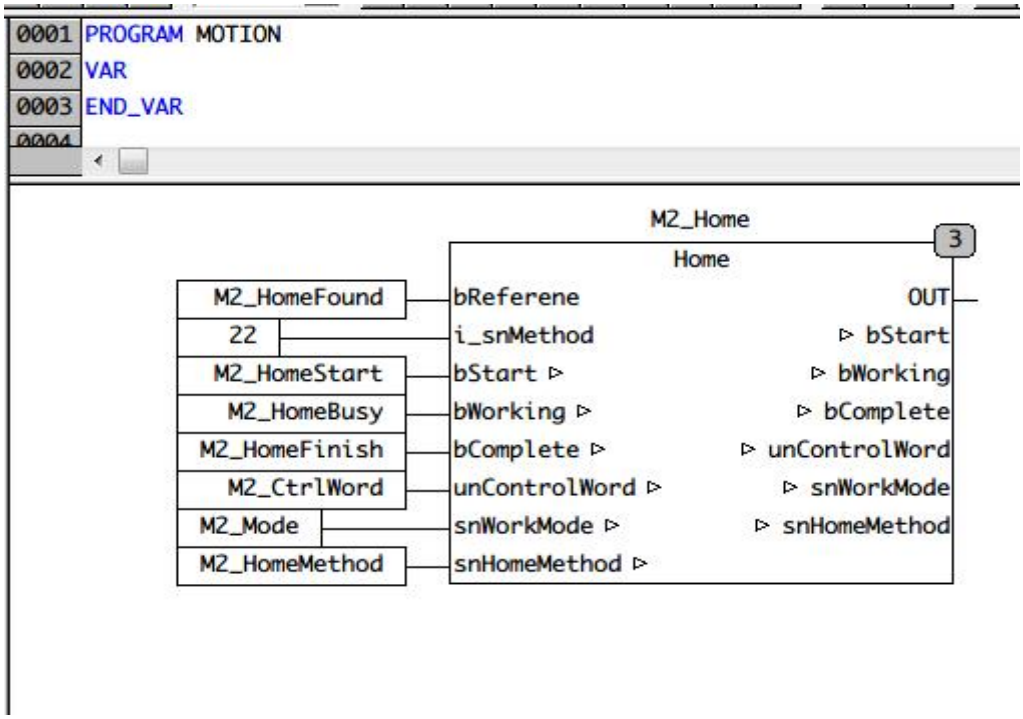
顺序下，步存储在右外侧的分支里。

9.5 连续功能图 CFC

9.5.1 CFC 编辑器

CFC 是连续功能块图（Continuous Function Chart）的简称。CFC 是一种图形化的编程语言。CFC 基于FBD 语言，但没有网络限制，摆放元素更加灵活。

CFC 编辑器是一种图形编辑器，如图所示，通过菜单栏或鼠标右键可以进行编辑。元素可以摆放在编程区任意位置。用鼠标拖拽在元素之间连线，当元素移动位置时，编辑器会自动调整连线长度。当元素之间空间不够时，连线会变为红色，一旦空间够用，红线会变为普通连线。

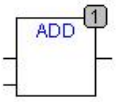
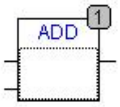
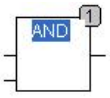
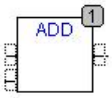
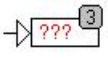
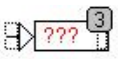
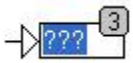
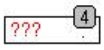
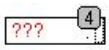
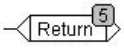
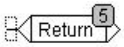


9.5.2 操作说明

CFC 的元素包括块、输入、输出、跳转、标记、返回和注释等。其中块分为操作符、函数、功能块和程序四种形式。

光标位置

名称	图形元素	选中元素	文本域	输入引脚和输出引脚
光标位置	编程工具元素	光标处于各元素的中继线	光标处于蓝色区域及注	光标处于元素的输入引脚和输出引脚上

块				
输入				
输出				
跳转				
标记				——
返回			——	
注释				——

选中元素在元素中继线处点击鼠标左键，可以选中元素。如果想同时选几个元素，按住<Shift>键并选中单个元素。也可以用鼠标左键在编辑器中画矩形区域选中其中几个元素。”高级”>“全选”选中所有元素。

移动元素：

当光标在位置a 时，或按住<Shift>键同时选中几个元素时，按住鼠标左键可以在编辑器中移动元素，到合适的位置后释放左键。如果释放位置处已有其它元素或超出编辑区，被移动元素会跳回原位置，移动失败。

连线：

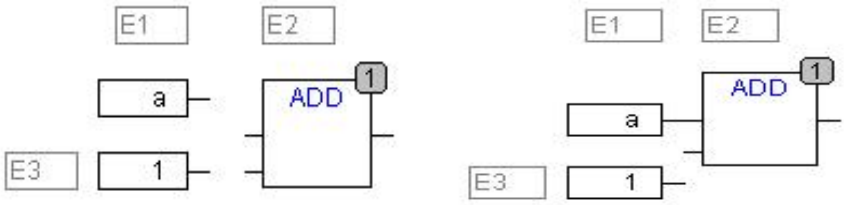
一个元素的输入引脚只能连一个输出引脚（本元素的输出引脚或其它元素的输出引脚），而一个元素的输出引脚可以连几个输入引脚（本元素的输入引脚或其它元素的输入引脚）。

如图所示，有三种方式在E1（a）和E2（ADD）之间连线。在连线时，编辑器会检查双方的数据类型是否匹配。如果不匹配，光标会变为“禁止”样式，连线失败。

把鼠标放在E1 的输出引脚上，按下左键，拖拽到E2 的输入引脚上，释放左键。

把鼠标放在E2 的输入引脚上，按下左键，拖拽到E1 的输出引脚上，释放左键。

按下左键不放，移动E1 或E2，使它们的输入引脚和输出引脚接触。



删除连线：

如图所示，有三种方式删除E1（a）和E2（ADD）之间的连线。选中E1 的输出引脚，按

下<Delete>键或“编辑”“删除”，如果E1 的输出引脚有几条线，则会同时删除。选中E2 的输入引脚，按下<Delete>键或“编辑”“删除”。选中E2 的输入引脚，用鼠标左键拖拽连线到编辑器空白位置，释放。

“块”：  ，快捷键<Ctrl>+<B>

执行此命令，可以插入操作符、函数、功能块和程序。新插入的块随鼠标移动，移动到合适的位置点击鼠标左键，插入成功。起初插入的是AND，可以把AND 改为其他名称。或者选中AND，按功能键F2，从帮助窗口中选择名称。

“输入”：  ，快捷键<Ctrl>+<I>

执行此命令，可以插入输入。新插入的输入随鼠标移动，移动到合适的位置点击鼠标左键，插入成功。文本域“???”要输入变量或常量。也可按功能键F2，从帮助窗口中选择。

“输出”： 

执行此命令，可以插入输出。新插入的输出随鼠标移动，移动到合适的位置点击鼠标左键，插入成功。文本域“???”要输入变量。也可以按功能键F2，从帮助窗口中选择。

“跳转”：  ，快捷键<Ctrl>+<J>

执行此命令，可以插入跳转。新插入的跳转随鼠标移动，移动到合适的位置点击鼠标左键，插入成功。文本域“???”要输入要跳转到的跳转标记。

“标记”：  ，快捷键<Ctrl>+<L>

执行此命令，可以插入标记。新插入的标记随鼠标移动，移动到合适的位置点击鼠标左键，插入成功。文本域“???”要输入跳转标记。在线模式时，编辑器自动在POU末尾自动插入标记“RETURN”。

“返回”：  ，快捷键<Ctrl>+<R>

执行此命令，可以插入返回。新插入的返回随鼠标移动，移动到合适的位置点击鼠标左键，插入成功。注意，此处的“返回”和在线模式时的POU末尾的标记“RETURN”不同，标记“RETURN”是为了POU结束本周期执行之前，自动执行下周期。

“注释”：  ，快捷键<Ctrl>+<K>

执行此命令，可以插入注释，用<Ctrl>+<“EN”ter>换行。新插入的注释随鼠标移动，移动到合适的位置点击鼠标左键，插入成功。

“排序>“显示排序号”

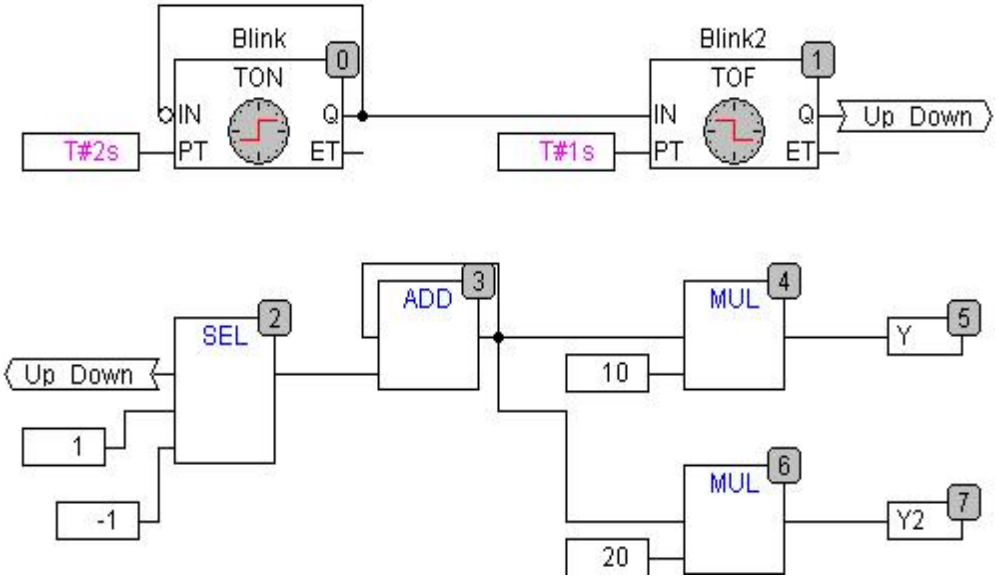
执行此命令，切换执行顺序的显示与否。缺省的是显示的。

“排序”>“拓扑排序”

执行此命令，执行顺序根据拓扑结构排列，也就是所有元素从上到下，从左到右排列。注意，执行顺序显示在元素的右上脚。

“排序”>“根据数据流排序”

执行此命令，执行顺序根据数据流排列，而不是根据元素所在的位置排列。CFC 应用举例如图所示。



第十章 常用指令

在控制器中，使CPU完成某种操作或实现某种功能的命令及多个命令的组合称为指令，指令的集合称为指令系统。指令系统是可编程控制器硬件和软件的桥梁，是控制器程序设计的基础。

控制器为用户提供了丰富的指令，这些指令均可通过CoDeSys编程软件进行调用，操作简单，使用方便。本章讲一一介绍这些指令，需要说明的是，对于一些简单指令，如IEC标准指令，只介绍指令名和列出可查详细说明地点，本文不做详细说明，而对于复杂应用指令等，讲详细说明以及给出范例

10.1 IEC 标准指令

编程软件 CoDeSys遵循国际电工技术委员会（IEC）的IEC61131-3 标准，在这个标准中明确规定了作为控制器必须具有的指令，即标准指令。标准指令包括类型转换指令、数字运算指令、位移位指令、比较指令、类型转换指令、定时器、计数器等，所有IEC61131-3标准规定指令，见下表。

IEC标准指令无需添加任何库，软件自行默认支持，如需查询指令详细用法，在软件自带帮助处查阅即可常用指令如下表：

表 10.1-1：ICE 标准指令表

指令类型	指令说明	指令名/符号	
算术运算指令	加法指令	ADD	+
	乘法指令	MUL	*
	减法指令	SUB	-
	除法指令	DIV	/
	取余指令	MOD	
赋值指令	赋值指令	MOVE	:=
逻辑运算指令	与指令	AND	
	或指令	OR	
	异或指令	XOR	
	取非指令	NOT	
移位指令	左移指令	SHL	
	右移指令	SHR	
	循环左移指令	ROL	
	循环右移指令	ROR	
选择指令	二选一指令	SEL	
	取最大值指令	MAX	
	取最小值指令	MIN	
	极限值指令	LIMIT	
	多选一指令	MUX	

比较指令	大于指令	GT	>
	小于指令	LT	<
	大于等于指令	GE	>=
	小于等于指令	LE	<=
	等于指令	EQ	=
	不等于指令	NE	<>
类型转换指令	布尔类型转换指令	BOOL_TO_<TYPE>	
	字节类型转换指令	BYTE_TO_<TYPE>	
	日期类型转换指令	DATE_TO_<TYPE>	
	长整型转换指令	DINT_TO_<TYPE>	
	日期时间类型转换指令	DT_TO_<TYPE>	
	双字类型转换指令	DWORD_TO_<TYPE>	
	整数类型转换指令	INT_TO_<TYPE>	
	字类型转换指令	WORD_TO_<TYPE>	
	单精度实数类型转换指令	REAL_TO_<TYPE>	
	短整型转换指令	SINT_TO_<TYPE>	
	字符类型转换指令	STRING_TO_<TYPE>	
	时钟类型转换指令	TIME_TO_<TYPE>	
	时间类型转换指令	TOD_TO_<TYPE>	
	无符号长整型转换指令	UDINT_TO_<TYPE>	
	无符号整型转换指令	UINT_TO_<TYPE>	
	无符号短整型转换指令	USINT_TO_<TYPE>	
	双精度实数类型转换指令	LREAL_TO_<TYPE>	
	截短转换成双整数指令	TRUNC	
	截短转换成整数指令	TRUNC_INT	
初等数学运算指令	绝对值指令	ABS	
	平方根指令	SQRT	
	自然对数指令	LN	
	常用对数指令	LOG	
	指数指令	EXP	
	正弦指令	SIN	
	余弦指令	COS	
	正切指令	TAN	
	反正弦指令	ASIN	
	反余弦指令	ACOS	
	反正切指令	ATAN	
	幂指令	EXPT	
地址运算指令	取地址指令	ADR	
	取地址内容指令	^	
	位地址指令	BITADR	
	数据类型大小指令	SIZEOF	

10.2 基本指令库

10.2.1 标准指令库“Standard.lib”

“Standard.lib”属于外部指令库，在工程建立时自动添加，包含的指令如下表所示。如需查询指令详细用法，点击帮助菜单下的在线帮助可查询。

表 10.2-1：“Standard.lib”库指令

指令类型	指令说明	指令名
定时器指令	RunTime 时钟定时器	RTC
	断电延时定时器	TOF
	通电延时定时器	TON
	普通定时器	TP
计数器指令	递减计数器	CTD
	递增计数器	CTU
	递增递减计数器	CTUD
触发器指令	下降沿检测触发器	F_TRIG
	上升沿检测触发器	R_TRIG
双稳态指令	复位双稳态器	RS
	置位双稳态器	SR
字符串指令	合并字符串指令	CONCAT
	删除字符串指令	DELETE
	查找字符串指令	FIND
	插入字符串指令	INSERT
	左边取字符串指令	LEFT
	取字符串长度指令	LEN
	中间取字符串指令	MID
	替换字符串指令	REPLACE
	右边取字符串指令	RIGHT

系统还提供一个名为“Standard64.lib”的库，与“Standard.lib”库区别就是提供了 64 位字符串和定时器指令，如下图：

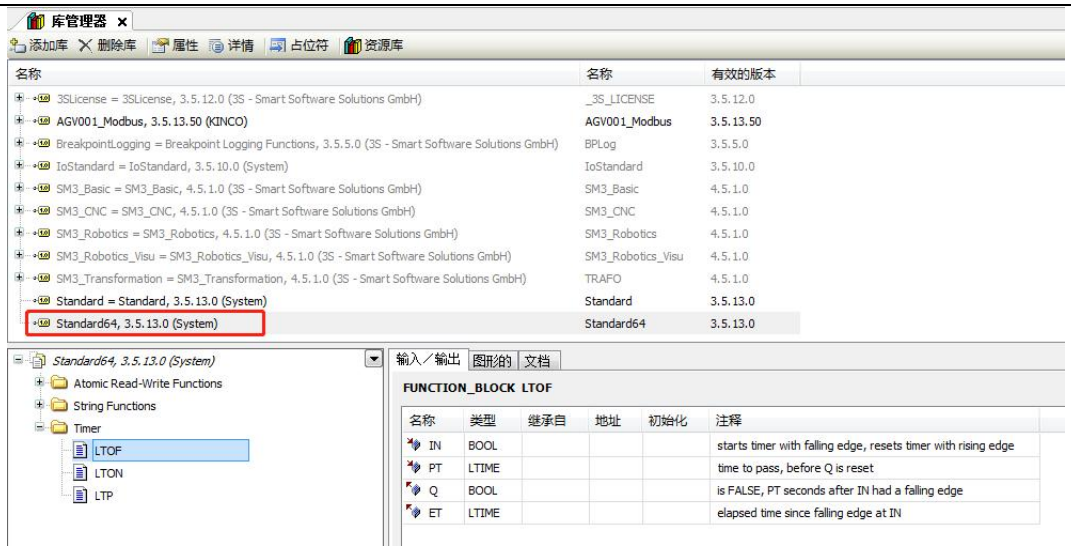


图 10.2-1：“Standard64.lib”库

10.2.2 应用指令库“Util.lib”

“Util.lib”属于外部指令库，在工程建立时不会自动添加，需用户手动添加，包含的指令如下表所示。如需查询指令详细用法，在codesys软件自带帮助如下图所示处查阅即可，也可在库下边找到指令参考右边的说明。

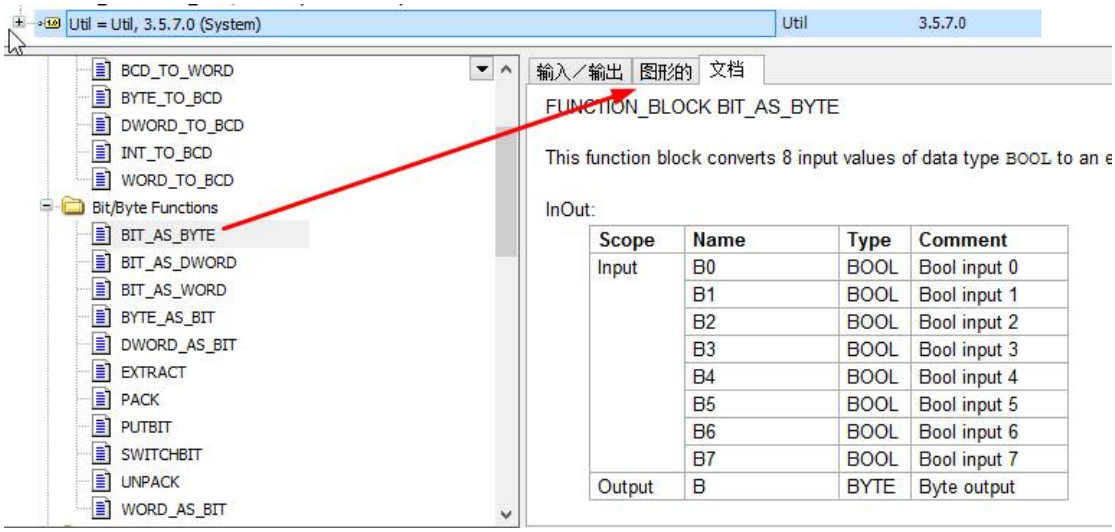


图 10.2-2：“Util.lib”库

“Util.lib”库的常用指令如下表：

表 10.2-2：“Util.lib”库的常用指令

指令类型	指令说明	指令名
BCD 码转换指令	BCD 码转整型指令	BCD_TO_INT
	整型转 BCD 码指令	INT_TO_BCD
	BCD 码转字指令	BCD_TO_WORD
	字转 BCD 码指令	WORD_TO_BCD
	BCD 码转双字指令	BCD_TO_DWORD

	双字转 BCD 码指令	DWORD_TO_BCD
	BCD 码转字节指令	BCD_TO_BYTE
	字节转 BCD 码指令	BYTE_TO_BCD
位处理指令	位整合字节指令	BIT_AS_BYTE
	位整合字指令	BIT_AS_WORD
	位整合双字指令	BIT_AS_DWORD
	字节拆分成位指令	BYTE_AS_BIT
	字拆分成位指令	WORD_AS_BIT
	双字拆分成位指令	DWORD_AS_BIT
	位整合指令	PACK
	位拆分指令	UNPACK
	位提取指令	EXTRACT
	位赋值指令	PUTBIT
	位选择指令	SWITCHBIT
模拟量处理指令	滞后指令	HYSTERESIS
	上下限报警指令	LIMITALARM
PID 控制指令	比例微分控制器	PD
	比例积分微分控制器	PID
	比例积分微分控制器（周期固定）	PID_FIXCYCLE
信号发生器指令	脉冲信号发生器）	BLINK
	典型周期信号发生器	GEN
	测量布尔输入信号的频率值	FREQ_MEASURE
高等数学辅助指令	微分	DERIVATIVE
	积分	INTEGRAL
	整型统计	STATISTICS_INT
	实型统计	STATISTICS_REAL
	平方偏差	VARIANCE
	线性变换	LIN_TRAFO
函数操纵器指令	特征曲线	CHARCURVE
	整型限速	RAMP_INT
	实型限速	RAMP_REAL
ASCII 码转换指令	字节转 ASCII 码	BYTE_TO_HEXinASCII
	ASCII 码转字节	HEXinASCII_TO_BYTE
	字转字符串	WORD_AS_STRING
格雷码转换指令	字节转格雷码	BYTE_TO_GRAY
	字转格雷码	Word_TO_GRAY
	双字转格雷码	Dword_TO_GRAY
	格雷码转字节	GRAY_TO_BYTE
	格雷码转字	GRAY_TO_word
	格雷码转双字	GRAY_TO_dword

10.2.3 实时时钟库“SysTImeRtc.lib”

“SysTImeRtc.lib”属于外部指令库，在工程建立时不会自动添加，需用户手动添加，如需查询指令详细用法，在 codesys 软件自带帮助处查阅即可，也可在库下边找到指令参考右边的说明

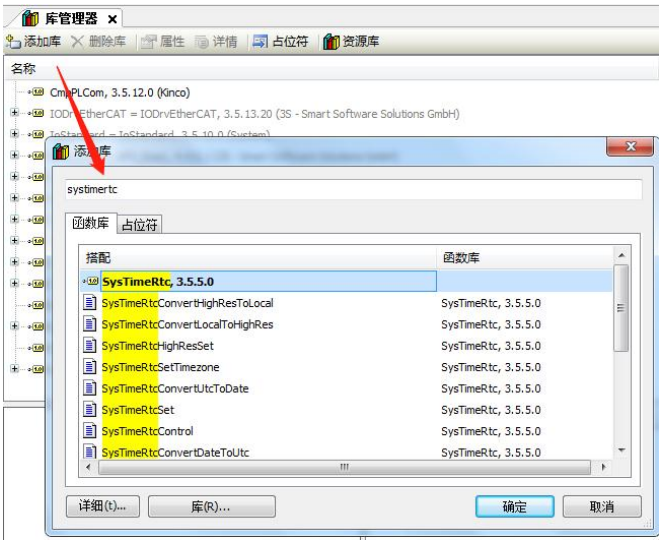


图 10.2-3：“SysTImeRtc.lib”库

包含的指令如下表所示。指令详细用法，如下图所实际演示：

```
RTCtime
PROGRAM RTCtime
VAR
    dwLocal_DateAndTime : DWORD; //获取到的本地时间
    Result : RTS_IEC_RESULT;
    dtLocal_DateAndTime : DATE_AND_TIME; //获取到的日期和时间
    todLocal_Time : TOD; //获取到的时间
    datLocal_Date : DATE; //获取到的日期
    bSetTime: BOOL; //设置时间触发位
    dtSet_DateAndTime: DATE_AND_TIME := DT#2022-01-23-00:00:00; //设置时间值
END_VAR

(*获取时间*)
dwLocal_DateAndTime:= SysTimeRtcGet(Result);//获取本地时间
dtLocal_DateAndTime := TO_DT(dwLocal_DateAndTime);//获取到的日期和时间
todLocal_Time := TO_TOD(dtLocal_DateAndTime);//获取到的时间
datLocal_Date := TO_DATE(dtLocal_DateAndTime);//获取到的日期

(*设定时间*)
IF bSetTime THEN
bSetTime := FALSE;
SysTimeRtcSet(DT_TO_DWORD(dtSet_DateAndTime));//设置格式: DT#2020-06-10-16:20:30
END_IF
```

图 10.2-4：“SysTImeRtc.lib”库指令演示

10.3 串口通讯指令库

10.3.1 串口自由通讯指令

串口自由通讯库包含在“CmpPLCom.lib”库中，属于外部指令库，在工程建立时不会自动添加，需用户手动添加，如下图：

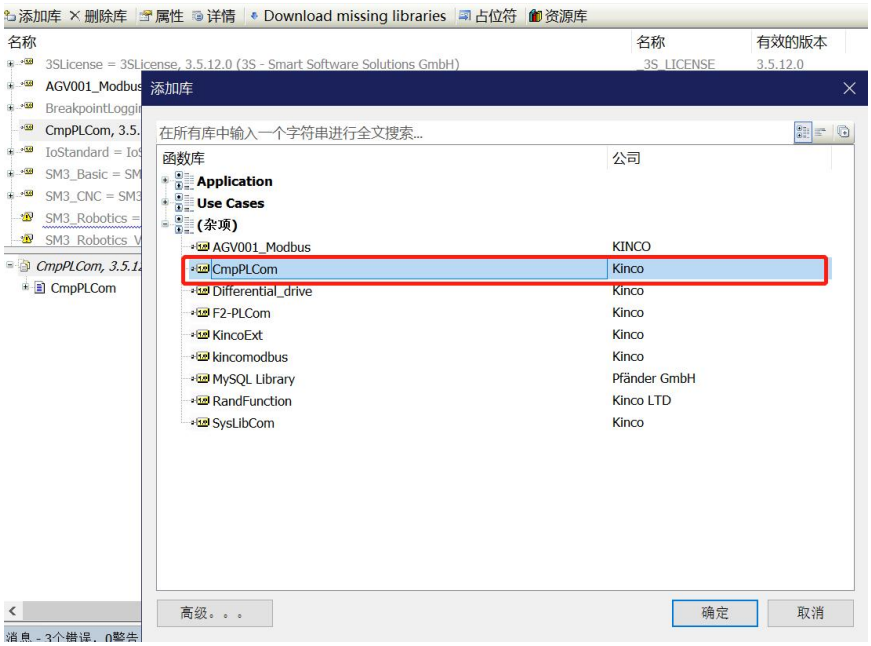


图 10.3-1：CmpPLCom.lib 库

串口自由通讯包含的指令如下表所示。下面一一介绍其用法：

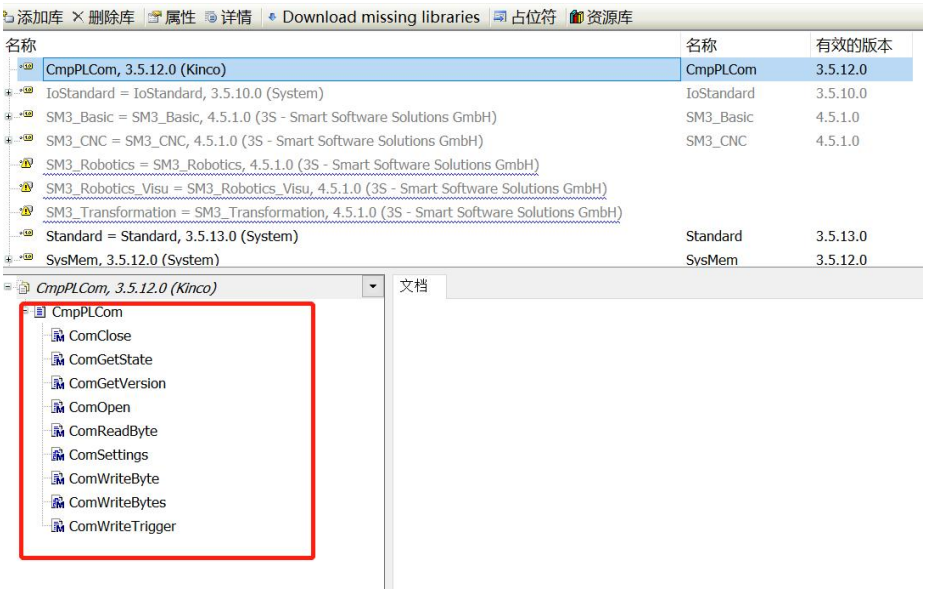


图 10.3-2：CmpPLCom.lib 库指令

图 10.3-1：CmpPLCom.lib 库指令介绍

指令类型	指令说明	指令名
串口自由通讯指令	打开串口	ComOpen
	关闭串口	ComClose
	获取 CmpPLCom 库版本	ComGetVersion
	设置串口	ComSettings
	获取串口端口状态	ComGetState
	从串口读一个字节	ComReadByte
	向串口写一个字节	ComWriteByte
	向串口写入多个字节	ComWriteBytes
	触发串口缓冲发送	ComWriteTrigger

ComOpen():

使用串口通讯前，必须使用此函数打开串口，之后可以使用 ComSettings()这个函数对这个打开的串口进行设置以满足自己的需要。仅当该接口是空闲的或请求的优先级高于当前优先级时才响应当前请求。

ComClose()这个函数可以关闭已经打开的串口。串口单元可能是 COM0(uiPort=0)、COM1(uiPort=1)、COM2(uiPort=2)、COM3(uiPort=3)和 COM4(uiPort=4)。

ComOpen 代表打开串口的标志位，为“TRUE”表示串口打开成功，为“FALSE”表示串口打开失败。

表 10.3-2：指令“ComOpen()”参数说明

	名称	类型	备注
输入	uiPort	UINT	物理接口编号：COM0=0，COM1=1、COM2=2、COM3=3、COM4=4
返回值	ComOpen	BOOL	打开串口标志位，成功返回 TRUE，失败返回 FALSE

```
声明：
    VAR
        PortUnit : UINT :=0; (*0=COM0, 1=COM1, 2=COM2, 3=COM3, 4=COM4*)
        bComOpenState : BOOL := FALSE;
    END_VAR
程序：
    bComOpenState := CmpPLCom.ComOpen(uiPort :=PortUnit); //  打开串口 COM0。
```

ComClose( ):

使用此功能，可以关闭已经打开的串口,该函数的返回值代表关闭串口的标志位，为 TRUE 表示串口关闭成功，为 FALSE 表示串口关闭失败。

表 10.3-3：指令“ComClose()”参数说明

	名称	类型	初始值	备注
输入	uiPort	UINT		物理接口编号：COM0=0、COM1=1、COM2=2、COM3=3、COM4=4
返回值	ComClose	BOOL		关闭串口标志位，成功返回 TRUE，失败返回 FALSE

```
声明：
    VAR
        PortUnit : UINT :=0; (*0=COM0, 1=COM1, 2=COM2, 3=COM3, 4=COM4*)
        bComCloseState : BOOL := FALSE;
    END_VAR
程序：
    bComCloseState :=CmpPLCom.ComClose(uiPort:= PortUnit); //  关闭串口 COM0。
```

ComGetVersion():

使用此功能，可以获取 CmpPLCom 库版本。该函数的返回值表示字符串版本字。

表 10.3-4：指令“ComGetVersion()”返回值说明

	名称	类型	初始值	备注
返回值	ComGetVersion	STRING		返回字符串版本字

```
    声明：
    VAR
        strVersion:STRING;
    END_VAR
    程序：
    strVersion := CmpPLCom.ComGetVersion(); //  获取 CmpPLCom 库版本。
```

ComSettings():

此函数用于设置已经打开的串口的参数，uiPort 为要设置的串口单元，串口单元可能是 COM0(uiPort=0)、COM1(uiPort=1)、COM2(uiPort=2)、COM3(uiPort=3)和 COM4(uiPort=4)。

dwBaudRate：要设置的串口波特率，波特率可能是 2400、4800、9600、19200、38400、56000、57600、100000、115200、125000、128000、200000、250000、500000、600000、750000、921600、1000000、1500000、2000000。

uiParity：奇偶校验，0=无校验、1=奇校验、2=偶校验。

ComSettings：设置串口标志位，返回值为 FALSE 时，表示设置失败，返回值为 TRUE 时，表示设置成功。

表 10.3-5：指令“ComSetting()”参数说明

	名称	类型	备注
输入	uiPort	UINT	物理接口编号：COM0=0、COM1=1、COM2=2、COM3=3、COM4=4
	dwBaudRate	DWORD	串口波特率
	uiParity	UINT	奇偶校验
返回值	ComSettings	BOOL	成功返回 TRUE，失败返回 FALSE

声明：

```
VAR
    PortUnit:UINT :=0;
    BaudRate:DWORD :=9600;
    Parity:UINT :=0;
    bComSettingsState : BOOL := FALSE;
END_VAR
```

程序：

```
bComSettingsState:=CmpPLCom.ComSettings(uiPort:=PortUnit, dwBaudRate:=BaudRate, uiParity:=Parity);
```

设置已经打开的串口的参数，通讯参数:无校验、8 个数据位、1 个停止位、波特率等。

ComGetState():

获取串口端口状态。

表 10.3-6：指令“ComGetState()”参数说明

Scope	名称	类型	备注
输入	uiPort	UINT	物理接口编号：COM0=0、COM1=1、COM2=2、COM3=3、COM4=4
输入输出	pdwValue	POINTER TO DWORD	返回端口的状态值 [Bit8]：1=TX 缓冲区不为空 0=TX 缓冲区为空 [Bit9]：1=RX 缓冲区不为空 0=RX 缓冲区为空
返回值	ComGetState	BOOL	成功返回 TRUE，失败返回 FALSE

```
声明：
    VAR
        PortUnit: UINT :=0;
        pdwValue: POINTER TO DWORD;
        bComGetState : BOOL := FALSE;
    END_VAR
程序：
    bComGetState :=CmpPLCom.ComGetState(uiPort:=pdwValue , pdwValue:= PortUnit );
    获取到串口 COM0 的端口状态
```

ComReadByte():

从接收缓冲区读取字符。 接收缓冲区为先入先出(FIFO)的执行方式，并且数据总是逐字符读取和写入。

表 10.3-7：指令“ComReadByte()”参数说明

Scope	名称	类型	备注
输入	uiPort	UINT	物理接口编号：COM0=0、COM1=1、COM2=2、COM3=3、COM4=4
输入输出	pbyValue	POINTER TO BYTE	返回从端口读出的一个字节
返回值	ComReadByte	BOOL	成功返回 TRUE，失败返回 FALSE

```
声明：
    VAR
        bValid : BOOL;
        PortUnit : UINT :=0;
        byRead : BYTE := 0; (*单次接收到的字符*)
        dwRxBytes : INT;(*接收字节数*)
        RecvBuffer:   ARRAY [0..255] OF BYTE;
    END_VAR
程序：
    REPEAT   (* read a character *)
            bValid := CmpPLCom.ComReadByte(uiPort:=PortUnit,pbyValue := ADR(byRead));
            IF bValid THEN
                IF (dwRxBytes <= 256) THEN (* array    limit reached    *)
                    dwRxBytes := dwRxBytes + 1;
                    RecvBuffer [dwRxBytes] := byRead;
                ELSE
                    dwRxBytes := 0;
                END_IF  (* received too many chars -> delete *)
            END_IF
        UNTIL (NOT bValid)
    END_REPEAT
    每接收到字符就依次存放到 RecvBuffer 这个数组中，缓存区变为空之后，接收完毕。
```

ComWriteByte():

写一个字符到发送缓冲区。该发送缓冲区采用先入先出执行方式，数据总是逐字符读取和写入。

表 10.3-8：指令“ComWriteByte()”参数说明

Scope	名称	类型	备注
输入	uiPort	UINT	物理接口编号：COM0=0、COM1=1、COM2=2、COM3=3、COM4=4
	byData	BYTE	要向串口写入的一个字节
返回值	ComWriteByte	BOOL	成功返回 TRUE，失败返回 FALSE

声明：

```
    VAR
        SendBuffer :ARRAY [0..10] OF BYTE;
    END_VAR
```

程序：

```
        FOR i :=0 TO 10 BY 1 DO
        CmpPLCom.ComWriteByte(uiPort:=PortUnit, byData := SendBuffer[i]);
        END_FOR
```

将 SendBuffer 数组中的数据逐字符发送出去。

ComWriteBytes():

向串口写入多个字节。

表 10.3-9：指令“ComWriteBytes()”参数说明

Scope	名称	类型	备注
输入	uiPort	UINT	物理接口编号：COM0=0、COM1=1、COM2=2、COM3=3、COM4=4
	pbyData	POINTER TO BYTE	要写入一段连续数据的首地址
	uiCount	UINT	这段连续数据的个数
返回值	ComWrite Bytes	BOOL	成功返回 TRUE，失败返回 FALSE

ComWriteTrigger():

清空发送缓存区。

表 10.3-10：指令“ComWriteTrigger()”参数说明

Scope	名称	类型	备注
输入	uiPort	UINT	物理接口编号：COM0=0、COM1=1、COM2=2、COM3=3、COM4=4
	benable	BOOL	当为 TRUE 时触发缓冲区发送
返回值	ComWriteTrigger	BOOL	成功返回 TRUE，失败返回 FALSE

10.3.2 Modbus RTU 串口指令库

AK800 控制器可以做为标准的 Modbus 主站和从站，在使用 modbus 功能时，首先要添加对应的库在软件系统中（库名：KincoComLib）。库安装的方法此处不再重复，请参考 7.4.4 章。AK800_Modbus 库获取方法：

- 从步科的各级经销商处获得此库进行安装；
- 在 Kinco 步科官网 https://www.kinco.cn 下载。

此库包含以下功能块：

1. Modbus RTU Master
2. Modbus RTU Slave
3. Modbus TCP IP Slave
4. Modbus UDP Slave

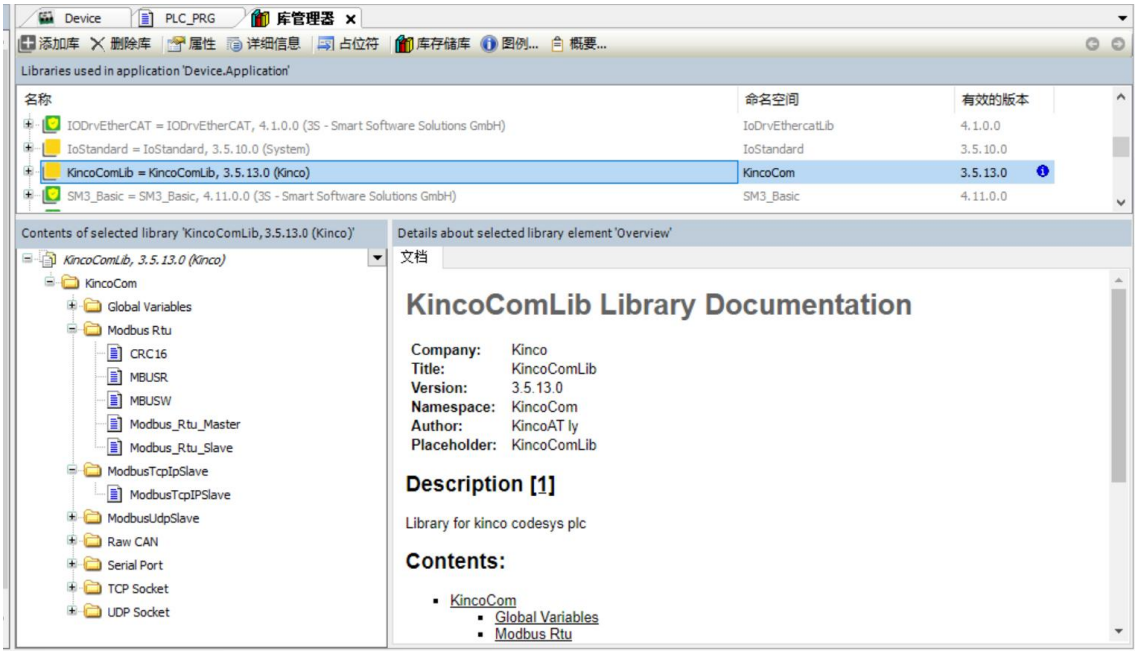


图 10.3-3：Kinco 的串口指令库

10.3.2.1 Modbus RTU 主站指令

（1）设置 AK800 控制器为 modbus RTU 主站，利用输入助手找到“Modbus RTU Master”功能块，选中然后点击确定，会弹出自动声明窗口，对此功能块进行实例化声明。

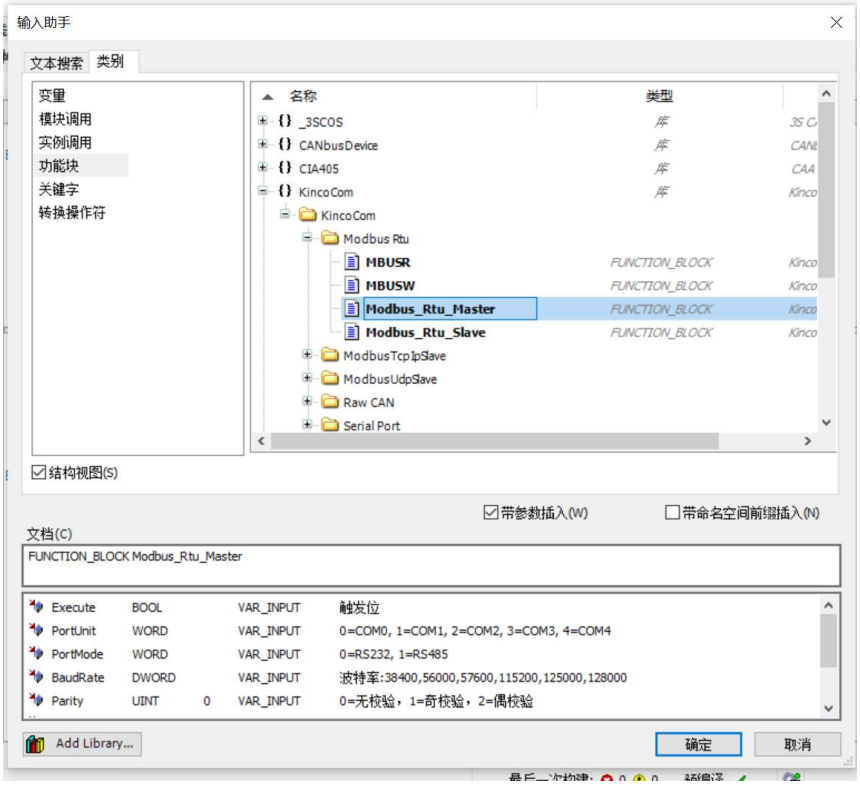
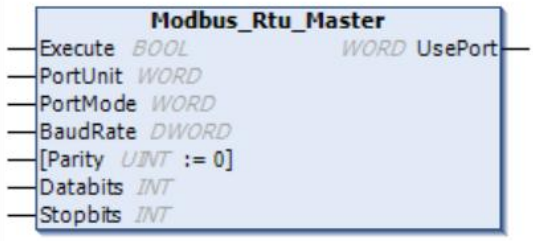


图 10.3-4：在输入助手查找指令

（2）指令块状图引脚显示如下图：



功能块输入输出各参数含义：

表 10.3-11：指令“ComWriteTrigger()”参数说明

	名称	类型	备注
输入	Execute	BOOL	上升沿执行
	PortUnit	WORD	选择串口：1=COM1, 2=COM2, 3=COM3
	PortMode	WORD	通讯方式：1=RS232, 2=RS485, 3=RS422
	BaudRate	DWORD	4800kbps、9600kbps、19200kbps、38400kbps、115200kbps
	Parity	INT	校验方式：0=无校验，1=奇校验，2=偶校验
	Databits	INT	数据位：0=8 个数据位，1=7 个数据位
	Stopbits	INT	停止位：0=1 个停止位，1=2 个停止位
输出	UsePort	WORD	当前使用的串口

（3）主站设置成功后，可使用 MBUSR 指令进行读数据操作，MBUSR 指令块的引脚如

下：

（4）功能块输入输出各参数含义如下表所示：

表 10.3-12：指令“MBUSR”参数说明

	名称	类型	备注
输入	Execute	BOOL	上升沿执行
	PortUint	DINT	将主站分配的 ReqPortUnit 赋给此输入
	SlaveID	BYTE	从站 ID
	FUN	WORD	功能码，支持 10#01,10#02,10#03,10#04
	StartAddress	UINT	起始地址
	Count	UINT	不能超过 64 个字，功能码为 16#01，16#02 时代表读取多少个位，功能码为 16#03,16#04 时代表读取多少个字
	timTimeOut	TIME	超时时间
	Big_OR_Small	BOOL	大端模式还是小端模式,针对功能码 03，04，接收的高低字节是否互换
输出	Read	ARRAY[0..127] OF BYTE	读取数据存放缓存区，一次性最多读取 64 个字
	TimeOutCount	DINT	通讯超时计数
	CrcError	BOOL	校验位错误
	IDError	BOOL	接收 ID 错误
	FunError	BOOL	返回功能码错误
输入输出	Done	BOOL	完成标志，可通过此标志触发下一条读或写指令，为方便使用，也可以在使用后将此标志清除。

（5）MBUSR 指令块功能介绍

MBUSR 功能块用于读取从站内的数据。该指令适用的功能码有 1（读 DO）、2（读 DI）、3（读 AO）和 4（读 AI）。定义了“SlaveID”目标从站的站号，允许的站号范围是 1~255。FUN 定义了功能码。“StartAddress”定义了要读取的寄存器的起始地址。“Count”定义了读取的寄存器的个数,不超过 64 个字。

“Execute”输入端的电平信号用于启动通讯。在功能块内部根据“Execute”输入变化产生上升沿信号，同时 MBUSR 进行一次数据交换：按照用户输入的站号、功能码等参数来组织报文并完成 CRC 校验，然后将报文发送出去并等待从站的回应；当接收到从站返回的报文后，

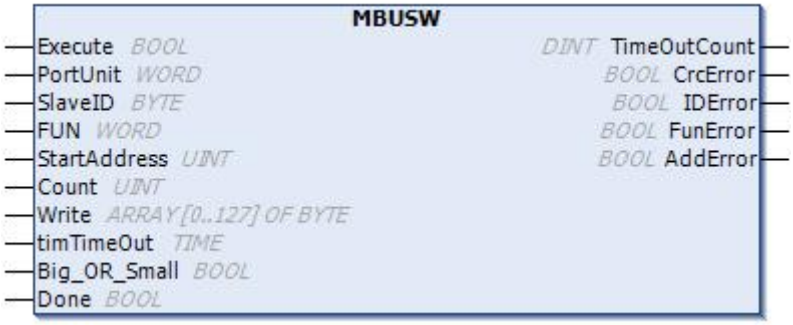
就对其进行 CRC 校验、站号校验和功能码校验。参数“Read”定义了数据缓冲区，读取的数据存放在该区域内。

当判断正确写入后，“Done”变为“TRUE”，表示指令完成，可触发下一条 modbus 读或写指令，也可手动清除完成信号，方便循环读写。

“Read”为读取到的数据区，“Big_OR_Small”为“FALSE”，功能码为 03、04 时，返回报文数据高字节在前，低字节在后，在此功能块中将读取到的高字节依次赋给 Read[0]、Read[2]……，读取到的低字节依次赋给 Read[1]、Read[3]……，也可以将“Big_OR_Small”（功能码为 03、04 时起效）变为 TRUE，交换高低字节，即在此功能块中将读取到的高字节赋给 Read[1]、Read[3]……，读取到的低字节赋给 Read[0]、Read[2]……。

功能码为 01、02 时，读取到的字节从 Read[0]、Read[1]……依次存放。

（6）MBUSW 指令块介绍



MBUSW 功能块输入输出各参数含义：

表 10.3-13：指令“MBUSW”参数说明

	名称	类型	备注
输入	Execute	BOOL	上升沿执行
	PortUint	DINT	将主站分配的 ReqPortUnit 赋给此输入
	SlaveID	BYTE	从站 ID
	FUN	WORD	功能码，支持 10#05,10#06,10#15,10#16
	StartAddress	UINT	起始地址
	Count	UINT	不能超过 64 个字，功能码为 10#05，10#06 时不起作用，功能码为 10#15 时代表写多少个位，功能码为 10#16 时代表写多少个字
	Write	ARRAY[0..127] OF BYTE	待写数据缓存区，一次性最多写入 64 个字
	timTimeOut	TIME	超时时间
	Big_OR_Small	BOOL	大端模式还是小端模式,针对功能码 06，16，接收的高低字节是否互换
输出	TimeOutCount	DINT	通讯超时计数
	CrcError	BOOL	校验位错误

	IDError	BOOL	接收 ID 错误
	FunError	BOOL	返回功能码错误
	AddError	BOOL	返回地址错误
输入 输出	Done	BOOL	完成标志，可通过此标志触发下一条读或写指令，为方便使用，也可以在使用后将此标志清除。

（7）MBUSW 指令块功能介绍

开启 Modbus RTU 主站后，MBUSW 功能块用于将数据写入从站内。该指令适用的功能码有 5（写一个 DO）、6（写一个 AO）、15（写多个 DO）和 16（写多个 AO）。“SlaveID”定义了目标从站的站号，允许的站号范围是 1~255。FUN 定义了功能码。“StartAddress”定义了要写入的寄存器的起始地址。“Count”定义了写寄存器的个数，不可超过 64 个字。参数“Write”定义了数据缓冲区，要写入从站的数据就存放在该区域内。

“Execute”输入端的电平信号用于启动通讯。MBUSW 指令执行时，在功能块内部根据“Execute”输入变化产生上升沿信号，同时 MBUSW 进行一次通讯：按照用户输入的目标站号、功能码、目标寄存器、数量、写入的数据等参数来组织报文并完成 CRC 校验，然后将报文发送出去并等待从站的回应；当接收到从站返回的报文后，就对其进行 CRC 校验、站号校验、地址和功能码校验，判读从站是否正确执行了刚才的写命令。

当判断正确写入后，“Done”变为“TRUE”，表示指令完成，可触发下一条 modbus 读或写指令，也可手动清除完成信号，方便循环读写。

“Write”为待写数据区，根据 modbus 协议，功能码 06、16 时，数据的高字节在前，低字节在后，当在“Big_OR_Small”输入为“FALSE”时，此功能块中也是这样传输的，Write[0]和 Write[1]构成一个字，高字节为 Write[1]，低字节为 Write[0]，Write[2]和 Write[3]构成一个字，高字节为 Write[3]，低字节为 Write[2]，依次类推。

也可以将“Big_OR_Small”输入（功能码为 06、16 时起效）变为“TRUE”，交换高低字节，即改变传输时的高低字节，即 Write[0]和 Write[1]构成一个字，高字节为 Write[0]，低字节为 Write[1]，Write[2]和 Write[3]构成一个字，高字节为 Write[2]，低字节为 Write[3]，依次类推。

功能码为 10#05，10#06 时，只传输 Write[0]和 Write[1]两个字节。

功能码为 05 时，强制值＝ 0xFF00，则置线圈为 ON；强制值＝0x0000，则置线圈为 OFF。Write[1]为高字节，Write[0]为低字节。

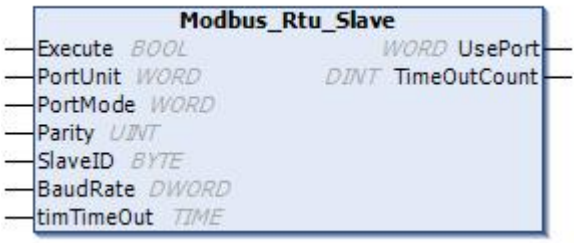
功能码为 15 时，为写多个线圈所以要写的数据在功能块中从 Write[0]，Write[1]……开始赋值，以字节形式依次写到从站。

10.3.2.2 Modbus RTU 从站指令

AK800 系列控制器可以通过 Modbus 协议与触摸屏或组态软件等第三方设备进行通讯。默认状态下，AK800 系列控制器作为 Modbus RTU 从站。AK800 系列控制器的 3 个串口均支持 Modbus RTU 从站协议。

（1）利用输入助手找到 Modbus_Rtu_Slave 功能块，选中并进行实例化调用（过程略）。

（2）功能块添加到 PLC 程序中显示如下图：



（3）“Modbus_Rtu_Slave”功能块输入输出参数含义：

表 10.3-14：“Modbus_Rtu_Slave”功能块输入输出参数含义

	名称	数据类型	描述
输入	Execute	BOOL	触发位
	PortUnit	WORD	0=COM0, 1=COM1, 2=COM2, 3=COM3, 4=COM4
	PortMode	WORD	0=RS232, 1=RS485
	Parity	UINT	0=无校验，1=奇校验，2=偶校验
	SlaveID	BYTE	从站站号
	BaudRate	DWORD	波特率：4800kbps、9600kbps、19200kbps、38400kbps、115200kbps
	timTimeOut	TIME	超时时间
输出	UsePort	WORD	当前使用的 COM 号
	TimeOutCount	DINT	超时计数

10.3.2.3 内存区说明

AK800 作为标准的 Modbus 从站，其能进行 Modbus 通讯访问的区域为 I 区、Q 区、M 区，各区如下表：

表 10.3-15：Modbus 通讯地址

类型	Modbus 功能码(十进制)	对应的 PLC 内存区域
DO（开关量输出，1XXXX）	01、05、15	Q 区
DI（开关量输入，0XXXX）	2	I 区
AO（模拟量输出，4XXXX）	03、06、16	M 区、Q 区
AI（模拟量输入，3XXXX）	4	I 区

*注：某些 Modbus RTU 主站的寄存器从 1 开始编号，此时将下表中的数据直接加 1 即可。

表 10.3-16：Modbus 通讯地址对应

内存区域	范围	类型	对应的 Modbus 寄存器（十进制）	长度
Q	Qx0.0——Qx7999.7	DO	0——63999	64000 位
I	Ix0.0——Ix7999.7	DI	0——63999	64000 位
MW	MW0——MW49999	AO	0——49999	50000 位
QW	QW0——QW15399	AO	50100——65499	15400 位
IW	IW0——IW49999	AI	0——49999	50000 位

地址对照表：双字、单字、字节之间的地址换算关系是 2n、2n+1；

位地址按字节地址编址。如：MD0=MW0+MW1=MB0+MB1+MB2+MB3。

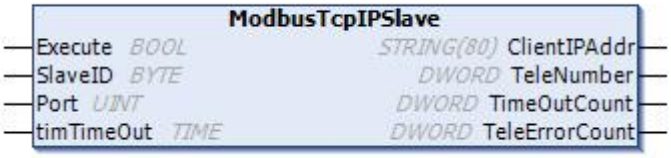
10.4 以太网口通讯指令库

AK800 本体上自带 1 个以太网口，都可以用来上传下载工程，也可以用来与以太网设备通讯，AK800 与其他以太网设备，如上位机则可以采用 Modbus UDP 或者 TCP 方式。此指令包含在 Modbus 库中不用再次安装，直接调用即可，其 Modbus 寄存器与控制器内存区的对应关系也是一样的。

10.4.1 ModbusTcpIPSlave 指令

（1）借助输入助手找到 ModbusTcpIPSlave 功能块，并进行实例化调用（过程略）。ModbusTcpIPSlave 指令示例请参照 11.2.1 章节。

（2）功能块引脚定义显示如下图：



（3）功能块输入输出参数含义：

表 10.4-1：ModbusTcpIPSlave 指令参数含义

	名称	类型	备注
输入	Execute	BOOL	上升沿执行
	SlaveID	BYTE	设置当前从站 ID
	Port	UINT	通讯端口号
	timTimeOut	TIME	超时时间
输出	ClientIPAddr	STRING(80)	返回主站 IP
	Telenumber	DWORD	报文发送数量
	TimeOutCount	DWORD	超时计数
	TeleErrorCount	DWORD	错误计数

10.4.2 ModbusUdpSlave 指令

（1）借助输入助手找到 ModbusUdpSlave 功能块，并进行实例化调用（过程略）。

（2）ModbusUdpSlave 功能块引脚示意如下图：

（3）功能块输入输出参数含义：

表 10.4-3：ModbusUdpSlave 指令参数含义

	名称	类型	备注
输入	Execute	BOOL	上升沿执行
	SlaveID	BYTE	设置当前从站 ID
	Port	UINT	通讯端口号
	timTimeOut	TIME	超时时间
输出	ClientIPAddr	STRING(80)	返回主站 IP
	Telenumber	DWORD	报文发送数量
	TimeOutCount	DWORD	超时计数
	TeleErrorCount	DWORD	错误计数

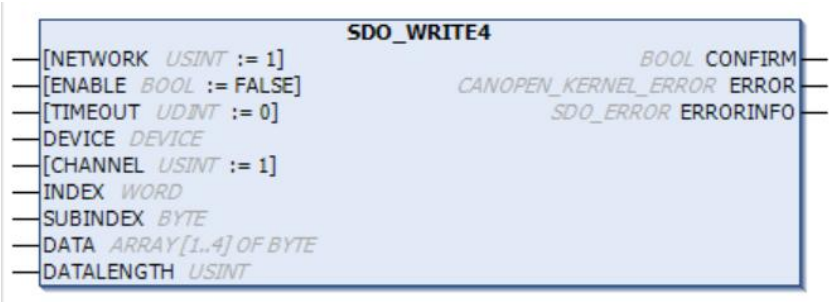
10.5 CAA_Cia405 指令库

CAA_Cia405 指令库描述了用于 IEC 61131-3 可编程设备（如 PLC）的标准化 CANopen 接口。此处仅介绍常用的单个 SDO 读写指令“SDO_READ4”和“SDO_WRITE4”。

10.5.1 SDO_WRITE4 指令块

“SDO_WRITE4”为单个 SDO 写入功能块。

（1）模块化指令图：



（2）功能块输入输出参数含义：

表 10.5-1：“SDO_WRITE4”指令参数含义

	名称	类型	备注
输入	NETWORK	USINT	CANopen 网络号(1=CAN0,2=CAN1)
	ENABLE	BOOL	触发执行位
	TIMEOUT	UDINT	超时时间/ms
	DEVICE	DEVICE	设备 NODE-ID
	CHANNEL	USINT	通道号

	INDEC	WORD	索引
	SUBINDEX	BYTE	子索引
	DATA	ARRAY [1..4] OF BYTE	目标数据组
	DATALENGTH	USINT	目标数据长度
输出	CONFIRM	BOOL	写入完成
	ERROR	CANOPEN_KERNEL_ERROR	错误标志位
	ERRORINFO	SDO_ERROR	错误代码

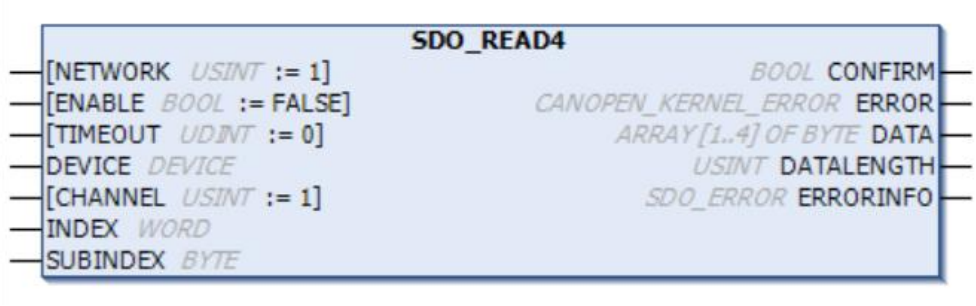
（3）事项：

- 索引与子索引请以 16 位数据形式写入，程序中的表示方法为：“16#****”；
- “DEVICE”填写要写入的设备的 NODE-ID；
- CAN Manager 中网络号 0 对应该功能块中“NETWORK”为 1，网络号 1 对应该功能块中“NETWORK”为 2；
- 目标数据“DATALENGTH”中，最大长度为“4”，代表 32BITS，最小为 1，代表 1BYTE（8BITS）。
- 错误代码“ERRORINFO”请参照 CoDeSys Online Help。
- 该功能块的使用示例请移步 11.3.5。

10.5.2 SDO_READ4 指令块

“SDO_READ4”为单个 SDO 读取功能块。

（1）模块化指令图：



（2）功能块输入输出参数含义：

表 10.5-2：“SDO_READ4”指令参数含义

	名称	类型	备注
输入	NETWORK	USINT	CANopen 网络号(1=CAN0,2=CAN1)
	ENABLE	BOOL	触发执行位
	TIMEOUT	UDINT	超时时间/ms
	DEVICE	DEVICE	设备 NODE-ID
	CHANNEL	USINT	通道号

	INDEC	WORD	索引
	SUBINDEX	BYTE	子索引
输出	DATA	ARRAY [1..4] OF BYTE	读取的数据组
	DATALENGTH	USINT	读取的数据长度
	CONFIRM	BOOL	读取完成
	ERROR	CANOPEN_KERNEL_ERROR	错误标志位
	ERRORINFO	SDO_ERROR	错误代码

（3）事项：

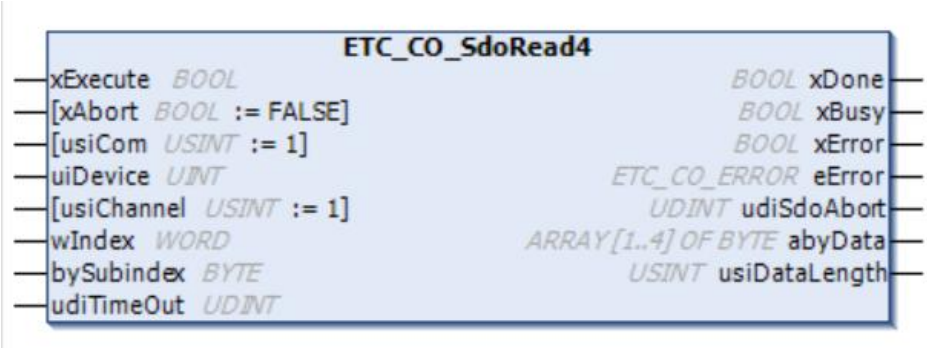
- 索引与子索引请以 16 位数据形式写入，程序中的表示方法为：“16#****”；
- “DEVICE”填写要写入的设备的 NODE-ID；
- CAN Manager 中网络号 0 对应该功能块中“NETWORK”为 1，网络号 1 对应该功能块中“NETWORK”为 2；
- 读取的目标数据“DATALENGTH”中，最大长度为“4”，代表 32BITS，最小为 1，代表 1BYTE（8BITS）。
- 错误代码“ERRORINFO”请参照 CoDeSys Online Help。
- 该功能块的使用示例请移步 11.3.5。

10.6 IODrvEtherCAT 指令库

如果添加了“EtherCAT”设备，则该库将自动集成到项目中。它包含用于读取和写入 EtherCAT 设备参数的功能块，即使在运行时也可以检查和更改各个参数。多个功能块可以同时处于活动状态。循环中的各个请求在内部进行管理并连续处理。本节主要介绍“EtherCATStackLivrary”下的“ETC_CO_SdoRead4”及“ETC_CO_SdoWrite4”两个常用功能块。其余功能块指令请参照“CoDeSys Online Help”。

10.6.1 ETC_CO_SdoRead4

（1）“ETC_CO_SdoRead4”可用于读取长度最大为 4 个字节的 EtherCAT 参数，其模块化指令图：



（2）功能块输入输出参数含义：

表 10.6-1：“ETC_CO_SdoRead4”指令参数含义

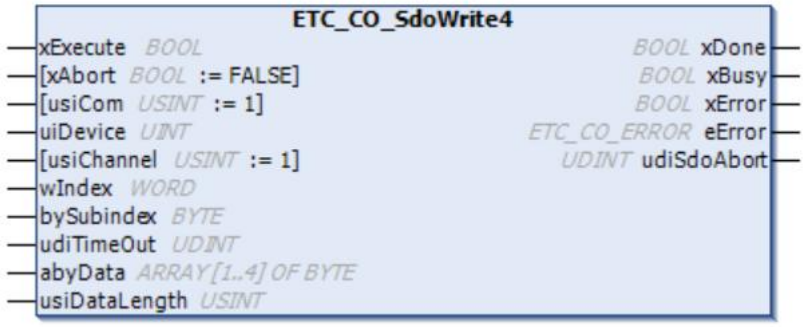
	名称	类型	备注
输入	xExecute	BOOL	触发执行位
	xAbort	BOOL	终止执行位
	usiCom	USINT	ETC 网络号
	uiDevice	UINT	用于访问 ETC 从站的物理从站地址
	usiChannel	USINT	通道号
	wIndex	WORD	索引
	bySubindex	BYTE	子索引
	udiTimeOut	UDINT	超时时间/ms
输出	xDone	BOOL	读取完成标志位
	xBusy	BOOL	读取执行中
	xError	BOOL	读取错误标志位
	eError	ETC_CO_ERROR	错误代码
	udiSdoAbort	UDINT	终止标志位
	abyDtata	ARRAY [1..4] OF BYTE	接收读取数据组
	usiDataLength	USINT	读取数据长度

（3）事项：

- 索引与子索引请以 16 位数据形式写入，程序中的表示方法为：“16#****”；
- “uiDEVICE”按 EtherCAT 连接顺序填写实际的从站设备地址；
- “usiChannel”为自动设置；
- 读取的目标数据“DATALENGTH”中，最大长度为“4”，代表 32BITS，最小为 1，代表 1BYTE（8BITS）。
- 错误代码“ETC_CO_ERROR”请参照 CoDeSys Online Help。
- 该功能块的使用示例请移步 11.5.5。

10.6.2 ETC_CO_SdoWrite4

（1）“ETC_CO_SdoWrite4”可用于写入长度最大为 4 个字节的 EtherCAT 参数，其模块化指令图：



（2）功能块输入输出参数含义：

表 10.6-2：“ETC_CO_SdoWrite4”指令参数含义

	名称	类型	备注
输入	xExecute	BOOL	触发执行位
	xAbort	BOOL	终止执行位
	usiCom	USINT	ETC 网络号
	uiDevice	UINT	用于访问 ETC 从站的物理从站地址
	usiChannel	USINT	通道号
	wIndex	WORD	索引
	bySubindex	BYTE	子索引
	udiTimeOut	UDINT	超时时间/ms
	abyDtata	ARRAY [1..4] OF BYTE	写入的目标数据组
	usiDataLength	USINT	目标数据长度
输出	xDone	BOOL	读取完成标志位
	xBusy	BOOL	读取执行中
	xError	BOOL	读取错误标志位
	eError	ETC_CO_ERROR	错误代码
	udiSdoAbort	UDINT	终止标志位

（3）事项：

- 索引与子索引请以 16 位数据形式写入，程序中的表示方法为：“16#****”；
- “uiDEVICE”按 EtherCAT 连接顺序填写实际的从站设备地址，与设备通用设置下的“EtherCAT 地址”对应，该地址默认为系统根据从站配置顺序自动分配；
- “usiChannel”为自动设置；
- 待写入的目标数据“DATALENGTH”中，最大长度为“4”，代表 32BITS，最小为 1，代表 1BYTE（8BITS）。
- 错误代码“ETC_CO_ERROR”请参照 CoDeSys Online Help。
- 该功能块的使用示例请移步 11.5.5。

第十一章 应用示例

11.1 AK800 更改 IP

AK800 控制器以太网通信端口默认 IP：192.168.0.250。

如果控制系统中利用多台控制器进行组网时，需要对控制器默认的 IP 进行修改，以免发生网络冲突，例如把 IP 修改为：192.168.0.100，修改步骤如下：

1.双击“Device”，进行设备扫描，确保设备正常连接（上位机与 AK800 控制器连接后，控制器图标显示为绿色原点，且会在右侧显示连接的设备信息）：

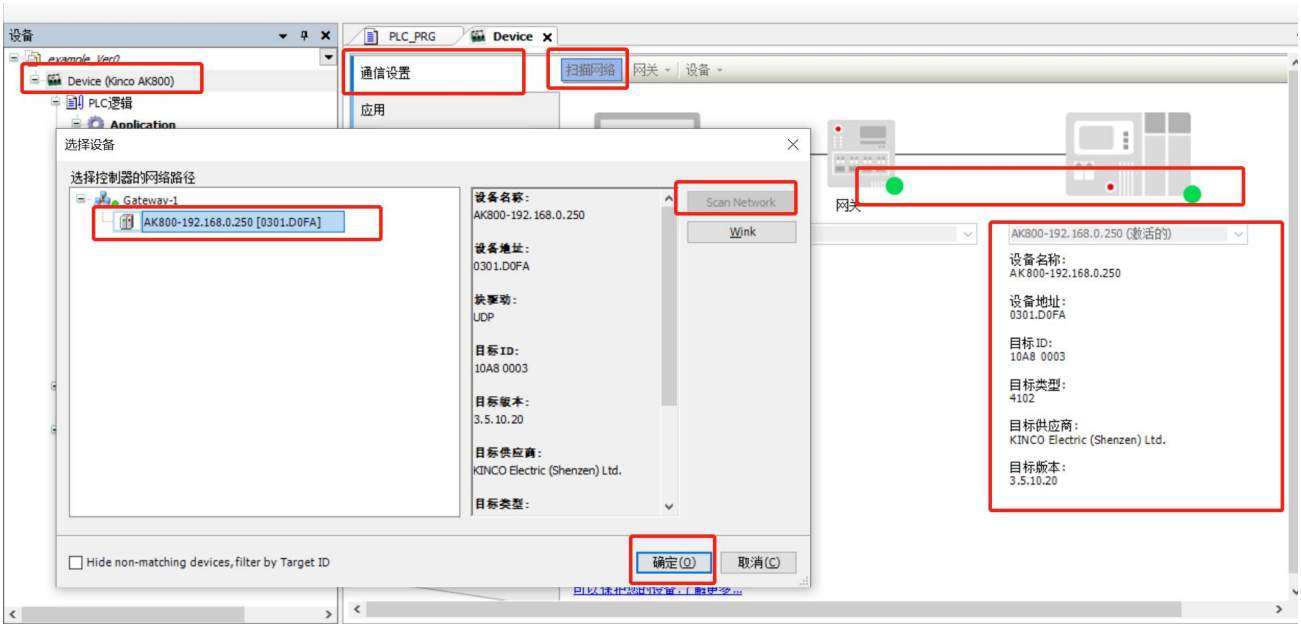


图 11.1-1：设备连接正常

2.打开“Device”设置下的“PLC 指令”，在下方的输入框中输入：

ipconfig eth0 -ip 192.168.0.100

点击回车（Enter），等待控制器修改 IP 完成后，会弹出设备断开的提示（此时 IP 已经修改成功，因为与之前的扫描连接的设备 IP 不匹配，故上位机与 AK800 控制器的连接将自动断开）。

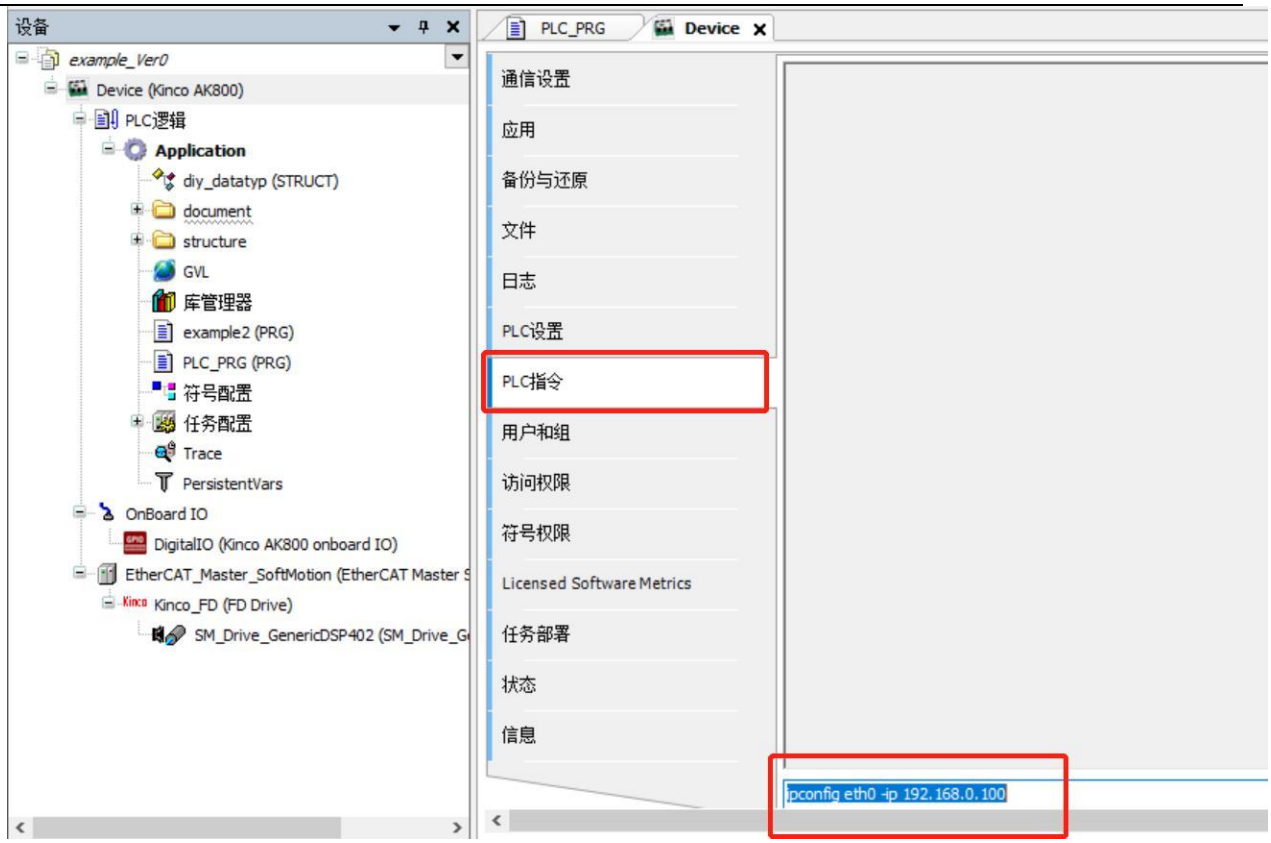


图 11.1-2：输入修改 IP 指令

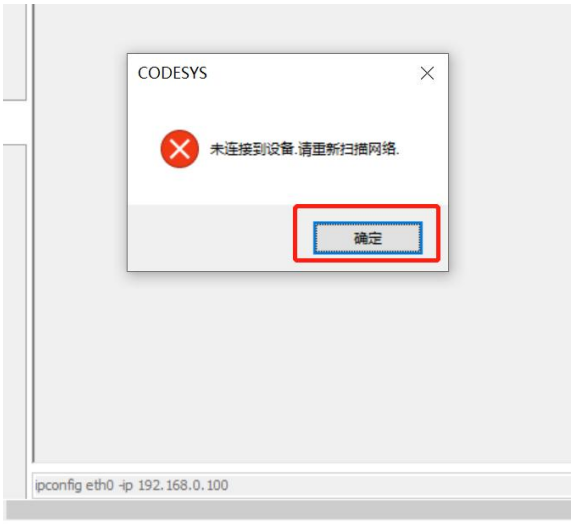


图 11.1-3：修改控制器 IP 成功后，PC 与控制器连接自动断开

3.在此对设备进行扫描并连接，可以看到扫描识别的设备 IP 地址已更改为“192.168.0.100”，同时 AK800 集成的监控面板上也显示了当前控制器的 IP 为“192.168.0.100”。

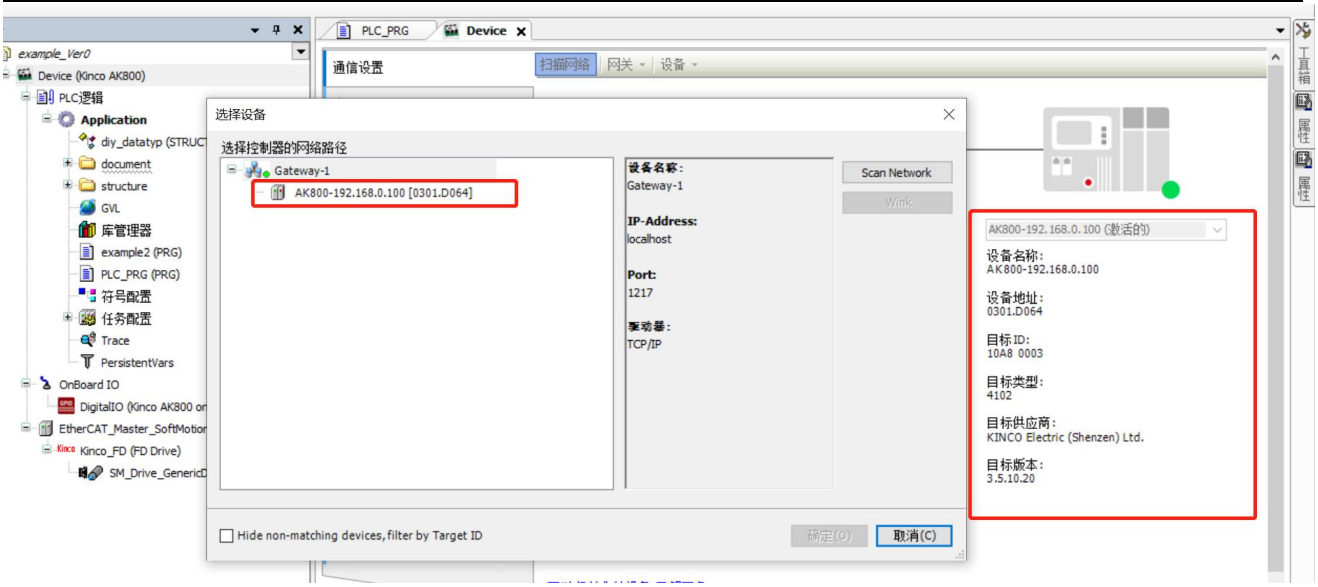


图 11.1-4：更改 IP 后重新扫描设备并里连接

图 11.1-5：AK800 控制器集成 LCD 面板监控

4.至此，控制器的 IP 修改完成，为避免不必要的误解，此处重申修改 IP 的指令格式为：

ipconfig<空格>eth0<空格>-ip<空格><IP 地址>

其中，<>中的内容直接进行内容替换，不需要输入“<>”符号。

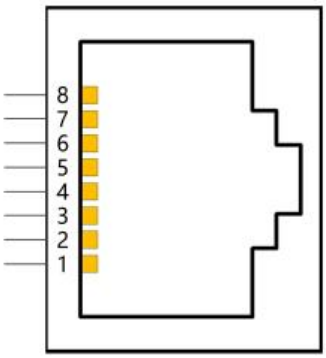
11.2 AK800 与 Kinco GL070E 触摸屏通讯

11.2.1AK800 与 Kinco GL070E 触摸屏通过 Modbus RTU 通讯

11.2.1.1 接线

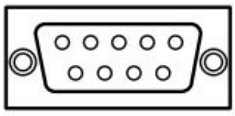
1.AK800 控制器的 COM 引脚分布如下表，本例中请按 COM0(RS485)接线。

表 11.2-1：AK800 控制器的 COM 引脚接线图

引脚	COM0(RS232)	COM0(RS485)	COM1(RS485)	COM0/COM1
1		A	A	
2		B	B	
3	TxD			
4				
5				
6	RxD			
7				
8	GND			

2.Kinco GL070E 触摸屏的 COM0 支持：RS232/RS485/RS422，引脚分布如下，此处请按 COM0-RS485 方式接线：

表 11.2-2：Kinco GL070E 触摸屏的 COM0 接线引脚



管脚	信号	功能		
		RS-232C	RS-485	RS-422A
1	RX-(B)	--	RS485B	接收数据
2	RXD	接收数据	--	--
3	TXD	发送数据	--	--
4	TX-	--	--	发送数据
5	SG	信号地		
6	RX+(A)	--	RS485A	接收数据
7	NC	--	--	--
8	NC	--	--	--
9	TX+	--	--	发送数据

11.2.1.2 组态与配置

在触摸屏软件上进行如下硬件组态：

（1）打开 Kinco 触摸屏软件 Dtools，点击“新建工程”，输入工程名称以及定义工程保存位置，选择 HMI 型号为：GL070E，点击下一步；

图 11.2-1：新建 HMI 工程

（2）在弹出的“系统参数设置”对话框中，新建 PLC 设备，设置通讯参数：

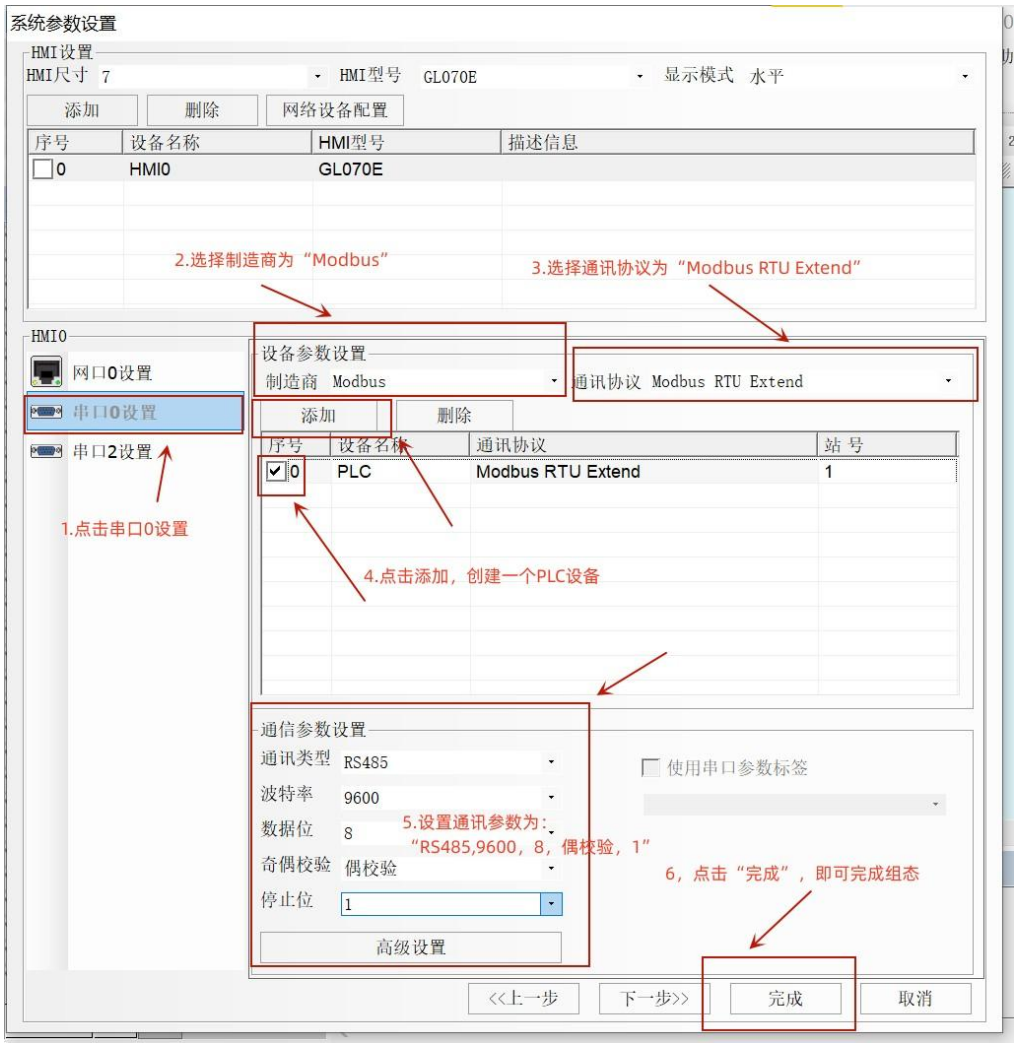


图 11.2-2：AK800 与 GL070E 组态设置

（3）在 Dtools 工程页面可以得到如下组态：

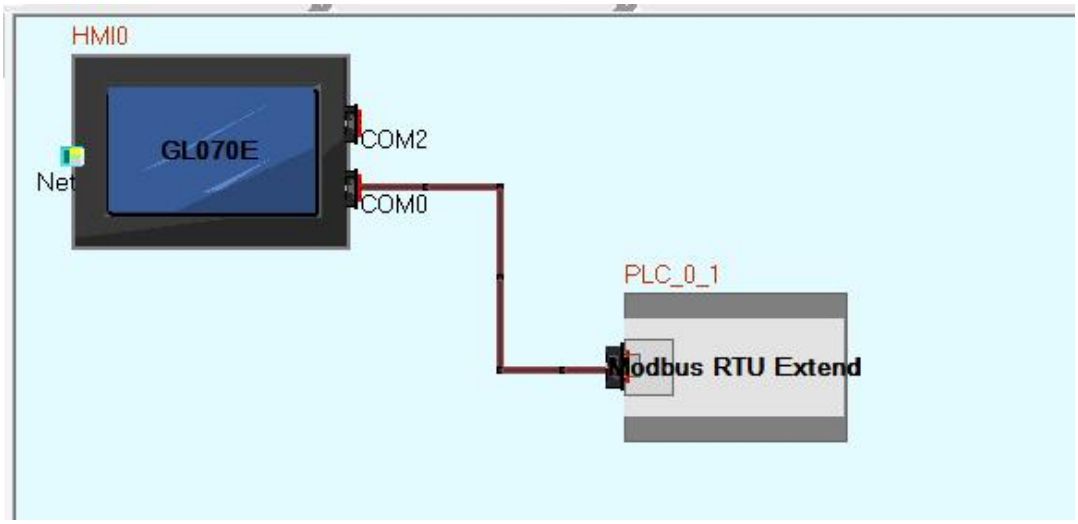


图 11.2-3：AK800 与 GL070E 以 RS485 模式组态

11.2.1.3 通讯示例工程

下面以一个简单的例子测试 AK800 与 HMI GL070E 的通讯：用按钮控制 AK800 的本体数字量输出及进行二次方运算。

1.在 Dtools 中进行如下 HMI 画面编程，并将工程完整编译后下载到 HMI 中。

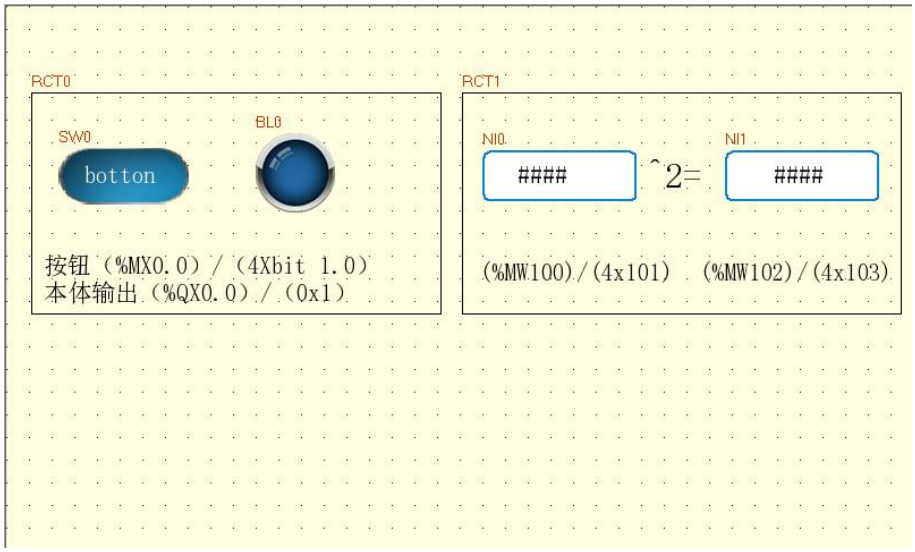


图 11.2-4：HMI 示例编程

2.控制器预设功能编程实现

新建 codesys 工程（参考 3.2 节），并编写程序点亮本体输出%QX0.0 以及进行二次方计算并输出结果。

3.Modbus RTU 通讯编程实现：调用“Modbus_RTU_Slave”指令打开通讯端口。如图下，通讯建立完成后登录控制器，把程序完整编译后下载到控制器中（使用“Modbus_RTU_Slave”指令需安装“KincoComLib”库）。如下图：

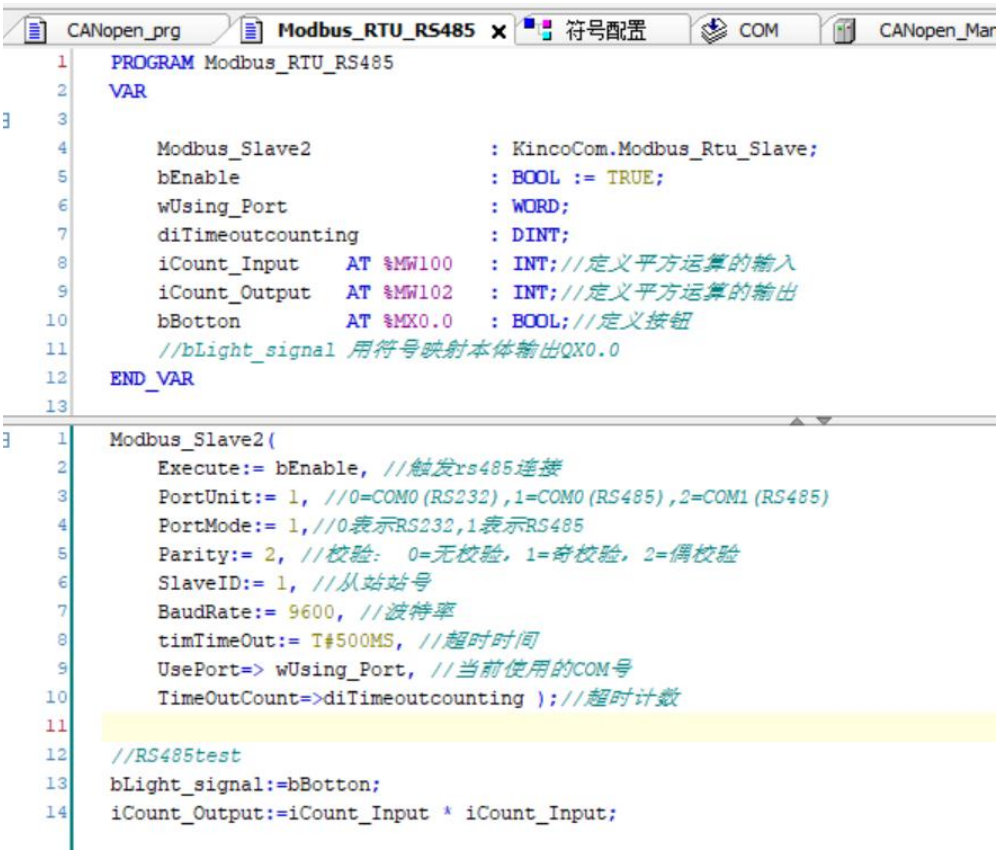


图 11.2-5：控制器预设功能与通讯功能编程实现

4.编程结束后，经过完全编译，将程序下载到控制器中并登陆运行，可以见到控制器已与 HMI 正常通讯。

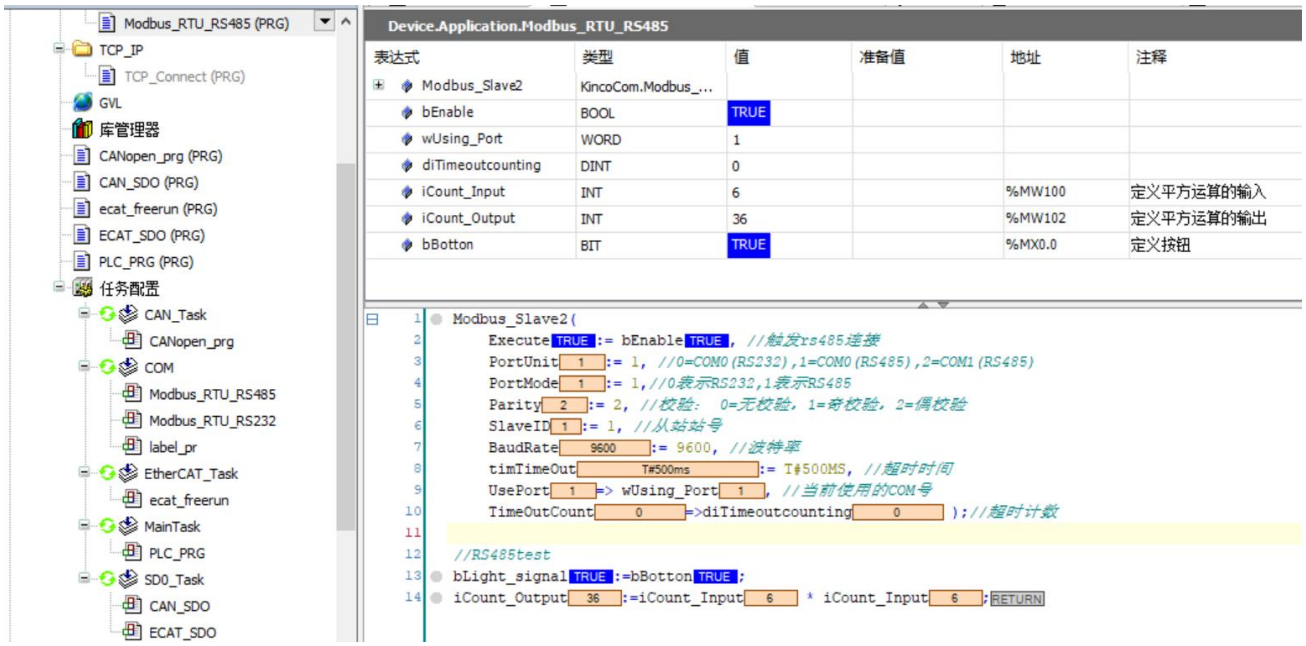


图 11.2-6：Modbus 通讯正常

5.HMI 工程测试：

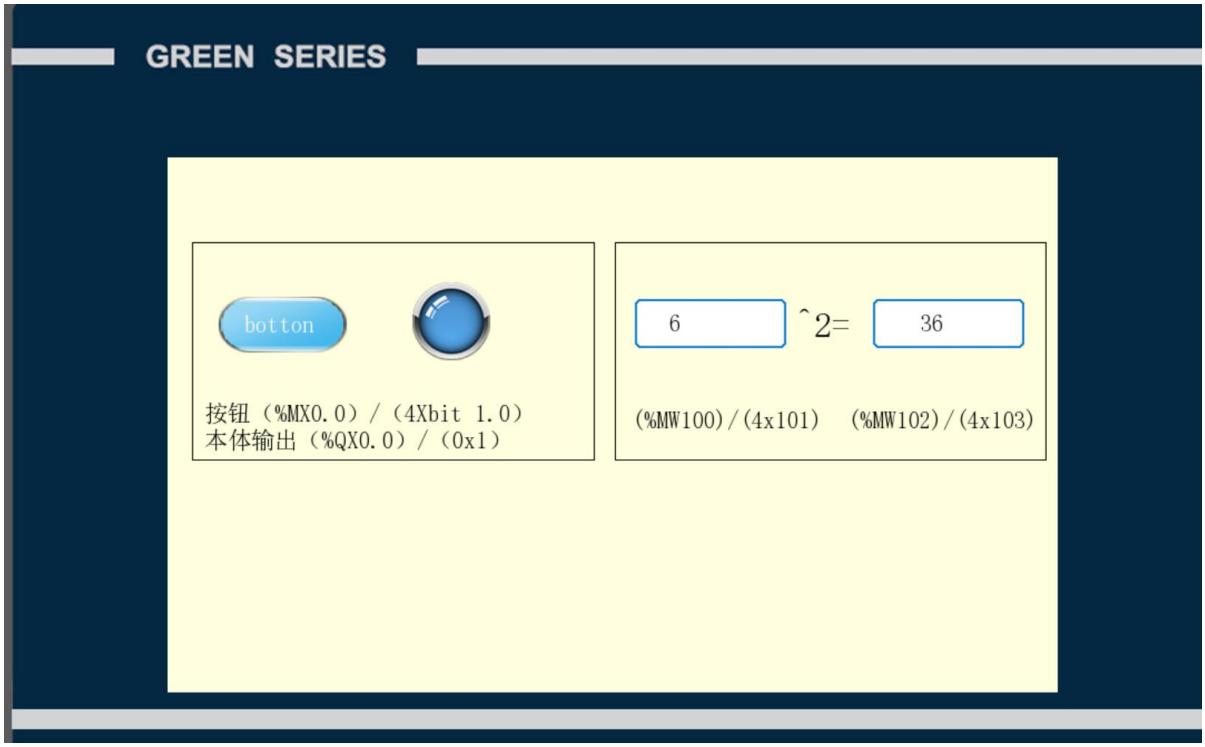


图 11.2-7：从触摸屏中可见预设功能正常运行

11.2.2AK800 与 Kinco GL070E 触摸屏通过 Modubus TCP 协议通讯

11.2.2.1 接线

只需用一根标准网线连接 GL070E 与 AK800 控制器，由于 AK800 和 HMI GL070E 都只有一个 Ethernet 通讯口，此处建议使用交换机进行连接，方便上位机监视，程序下载等。

11.2.2.2 组态与配置

在触摸屏软件 Dtools 上进行如下硬件组态：

（1）打开 Kinco 触摸屏软件 Dtools，点击“新建工程”，输入工程名称以及定义工程保存位置，选择 HMI 型号为：GL070E，点击下一步（创建 HMI 参照 10.2.1.1）；

（2）在弹出的“系统参数设置”对话框中，新建 PLC 设备，设置通讯参数：

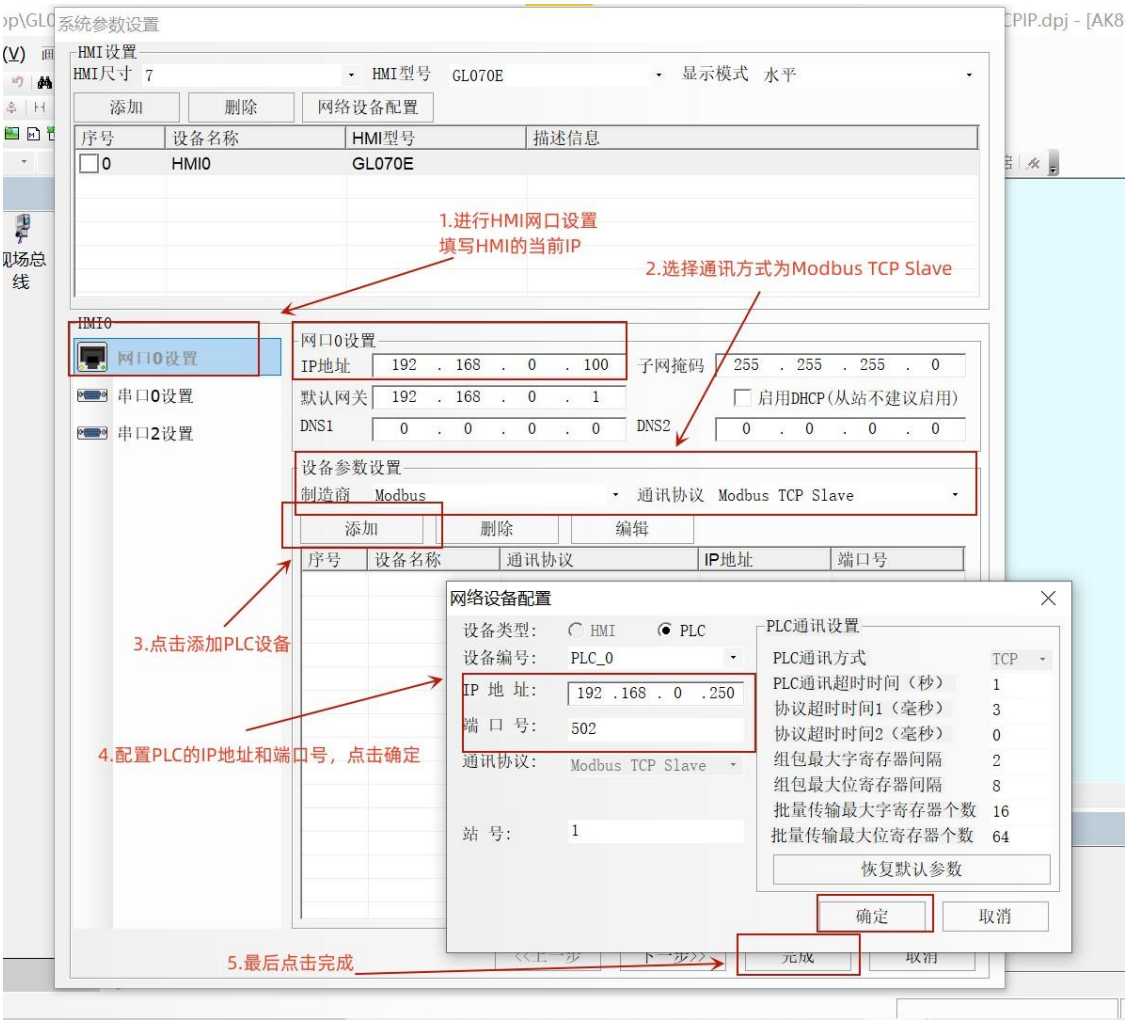


图 11.2-8：在 Dtools 上配置通讯参数

（3）在 Dtools 工程页面可以得到如下组态：

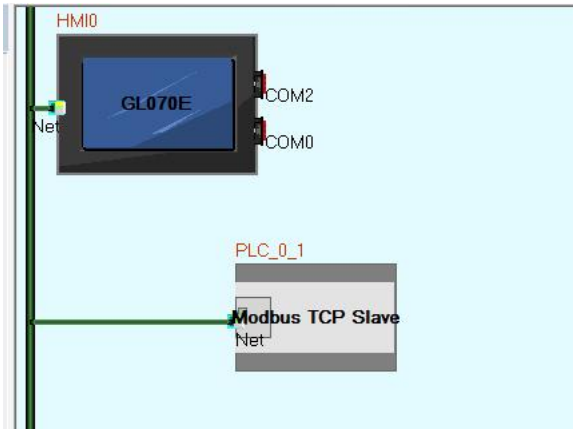


图 11.2-9：AK800 与 GL070E 以 TCP 模式组态

11.2.2.3 通讯示例工程

下面同样以一个简单的例子测试 AK800 与 HMI GL070E 的通讯：用按钮控制 AK800 的本体数字量输出及进行二次方运算。

1.在 Dtools 中进行如下 HMI 画面编程，并将工程完整编译后下载到 HMI 中。

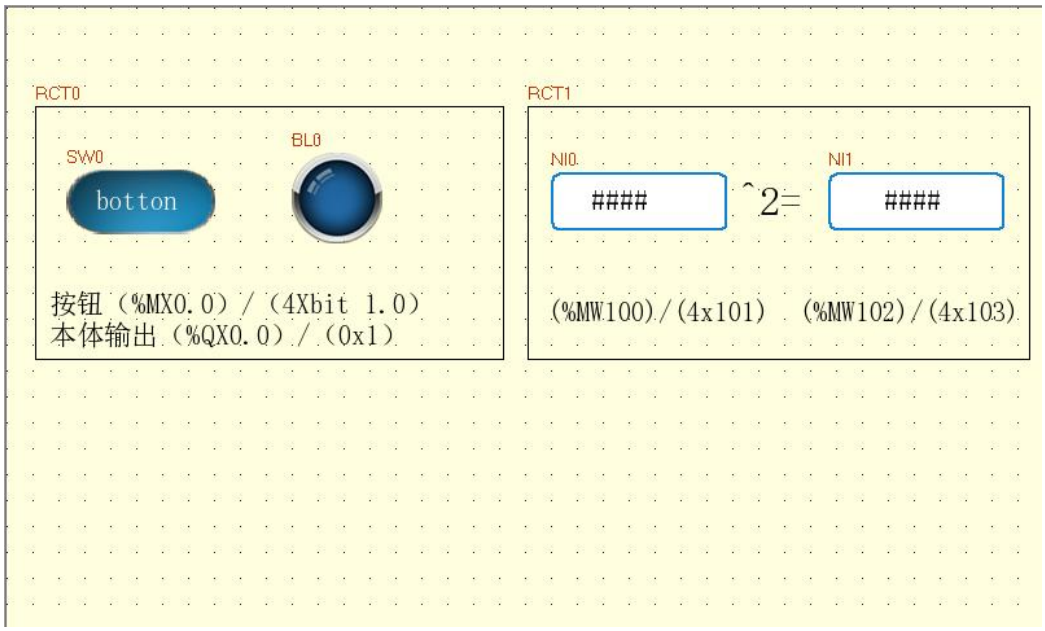


图 11.2-10：HMI 示例编程

2.控制器预设功能编程实现

新建 codesys 工程（参考 3.2 节），并编写程序点亮本体输出%QX0.0 以及进行二次方计算并输出结果。具体编程如下图。

3.Modbus TCP 通讯编程实现：调用“KincoCom.ModbusTcpIPSlave”指令打开通讯端口，如图下，通讯建立完成后登录控制器，把程序完整编译后下载到控制器中（使用“KincoCom.ModbusTcpIPSlave”指令需安装“KincoComLib”库）。

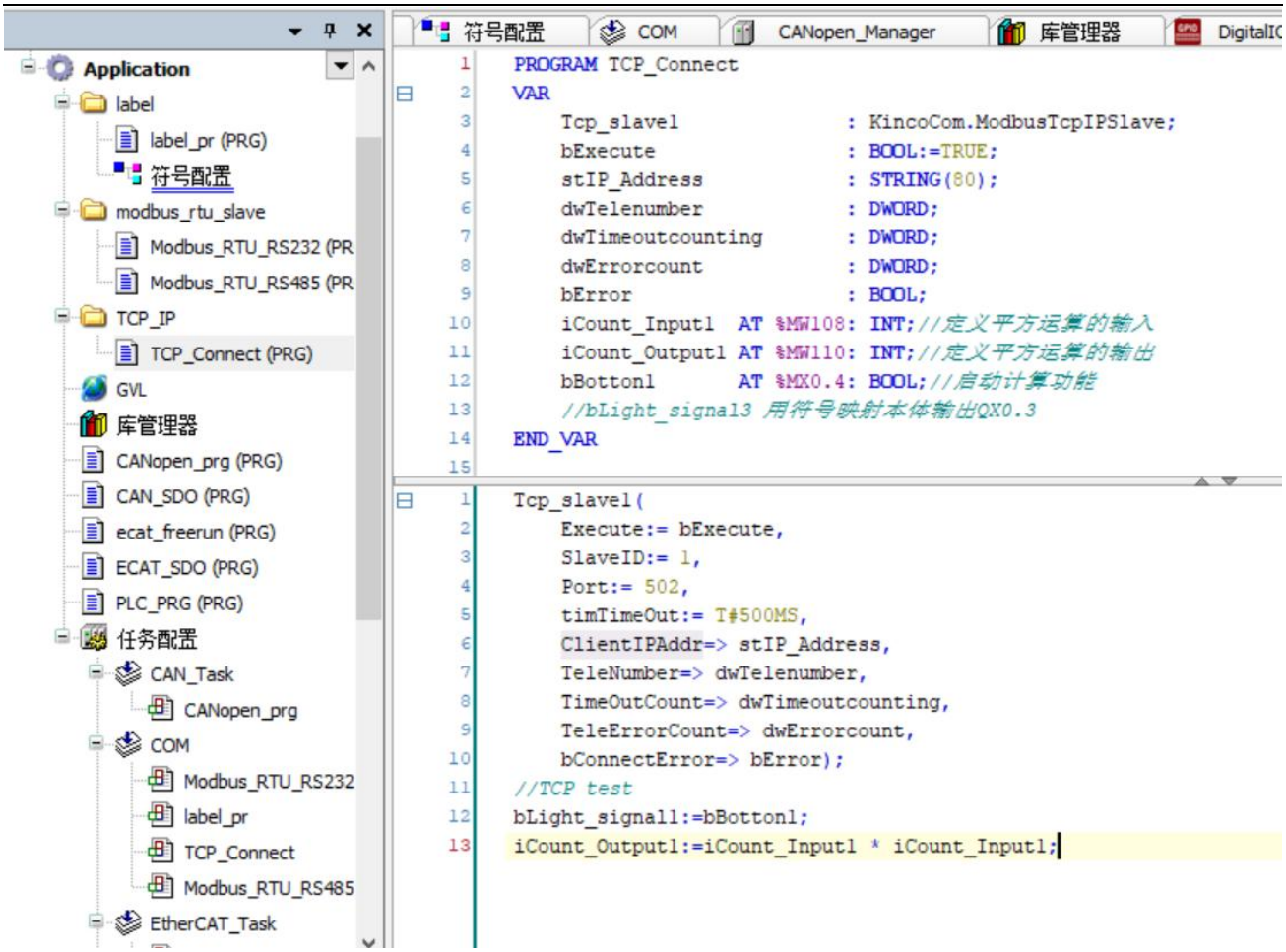


图 11.2-11：控制器预设功能与 Modbus RTU 通讯编程实现

4.编程结束后，经过完全编译，将程序下载到控制器中并登陆运行，同时创建启动应用，断电重启控制器，可以见到控制器已与 HMI 正常通讯。

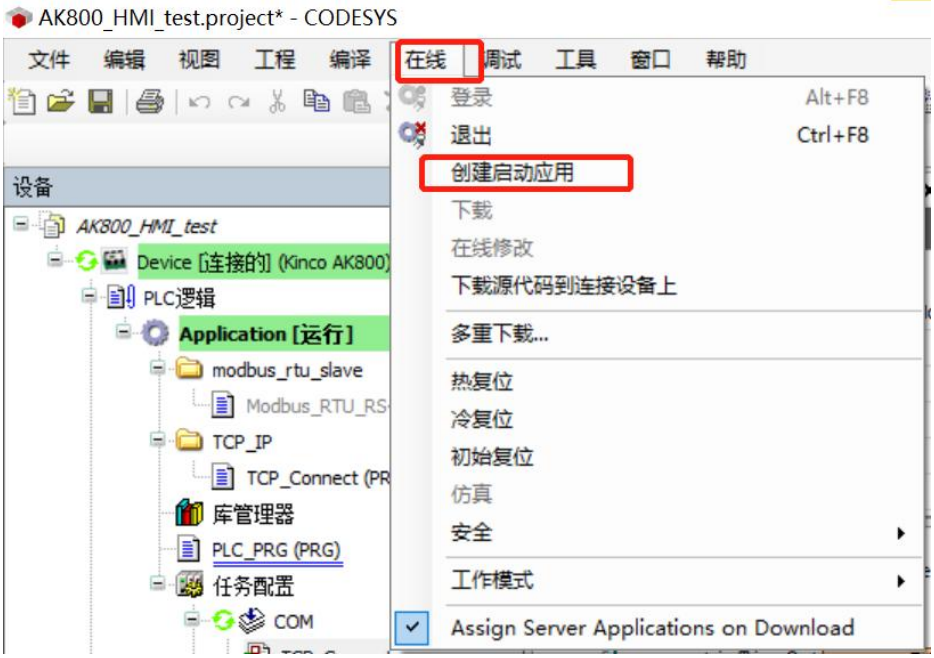


图 11.2-12：登录下载并创建启动应用

Device.Application.TCP_Connect

表达式	类型	值	准备值	地址	注释
dwTelenumber	DWORD	9499			
dwTimeoutcounting	DWORD	2			
dwErrorcount	DWORD	0			
bError	BOOL	FALSE			
iCount_Input1	INT	9		%MW108	定义平方运算的输入
iCount_Output1	INT	81		%MW110	定义平方运算的输出
bBotton1	BIT	TRUE		%MX0.4	启动计算功能

```
1  Tcp_slave1(
2      Execute:= bExecute,
3      SlaveID:= 1,
4      Port:= 502,
5      timTimeOut:= T#500MS,
6      ClientIPAddr:= stIP_Address,
7      TeleNumber:= dwTelenumber,
8      TimeOutCount:= dwTimeoutcounting,
9      TeleErrorCount:= dwErrorcount,
10     bConnectError:= bError);
11 //TCP test
12 bLight_signal1:=bBotton1;
13 iCount_Output1:=iCount_Input1 * iCount_Input1;RETURN
```

图 11.2-13：Modbus TCP 通讯正常

11.2.3AK800 与 Kinco GL070E 触摸屏进行标签通讯

11.2.3.1 接线

只需用一根标准网线连接 GL070E 的 EtherNET0 口与 AK800 控制器的 Ethernet 通讯口，由于 AK800 和 HMI GL070E 都只有一个 Ethernet 口，此处建议使用交换机进行连接，方便上位机监视，程序下载等。

11.2.3.2 组态与配置

在触摸屏软件 Dtools 上进行如下硬件组态：

（1）打开 Kinco 触摸屏软件 Dtools，点击“新建工程”，输入工程名称以及定义工程保存位置，选择 HMI 型号为：GL070E，点击下一步（创建 HMI 参照 10.2.1.1）；

（2）在弹出的“系统参数设置”对话框中，新建 PLC 设备，设置通讯参数：

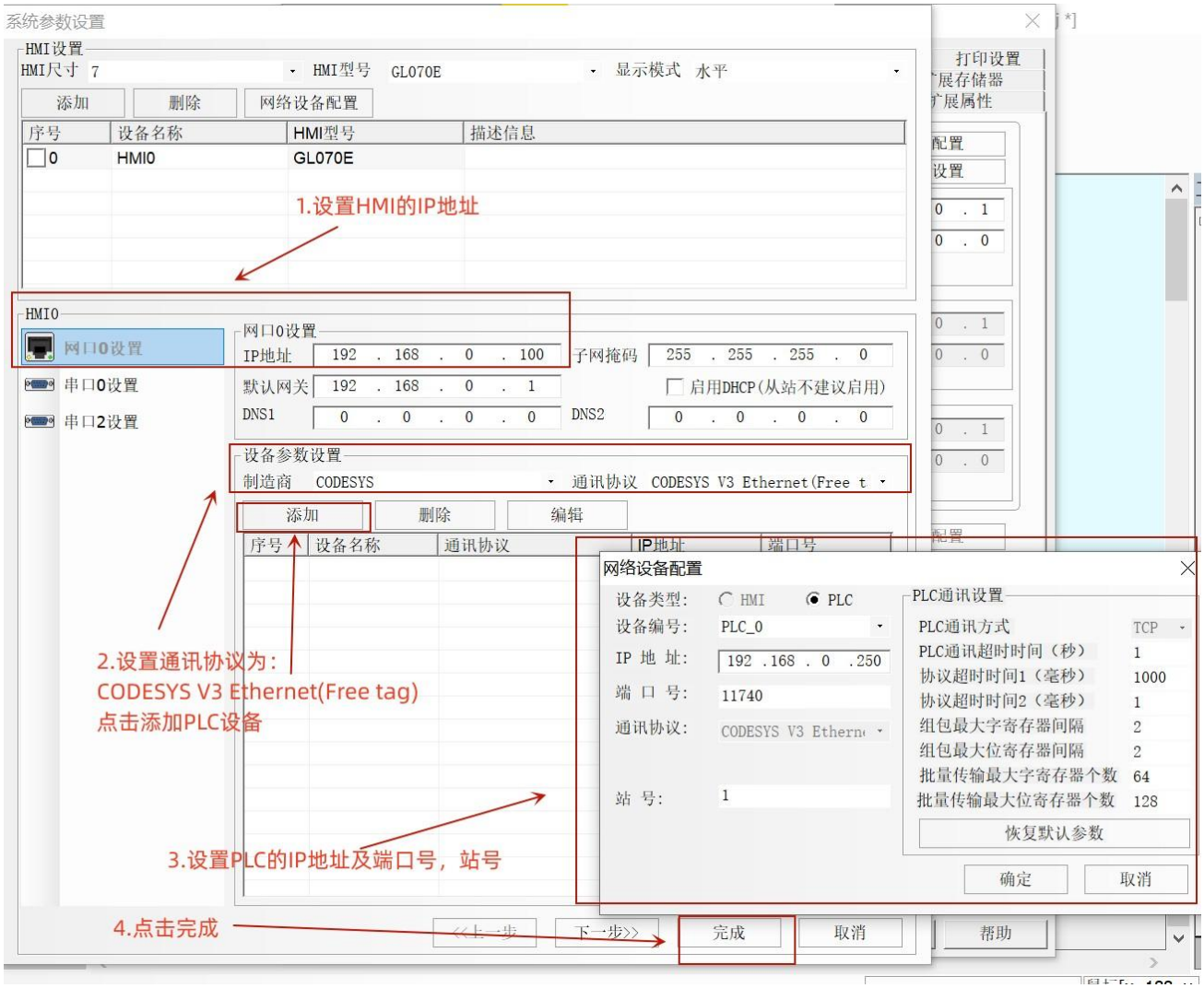


图 11.2-14：在 Dtools 上配置通讯参数

（3）在 Dtools 工程页面可以得到如下组态：

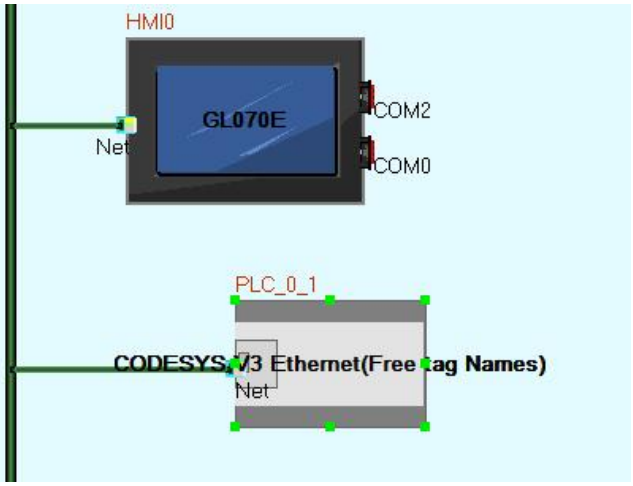


图 11.2-15：AK800 与 GL070E 以标签通讯模式组态

11.2.3.3 通讯示例工程

下面同样以一个简单的例子测试 AK800 与 HMI GL070E 的通讯：用按钮控制 AK800 的本体数字量输出及进行二次方运算。

1.控制器编程实现

新建 codesys 工程（参考 3.2 节），并编写程序点亮本体输出%QX0.4 以及进行二次方计算并输出结果。具体编程（此处不赘述，请参考示例工程文件）。

（1）创建全局变量表，本例新建如下变量；

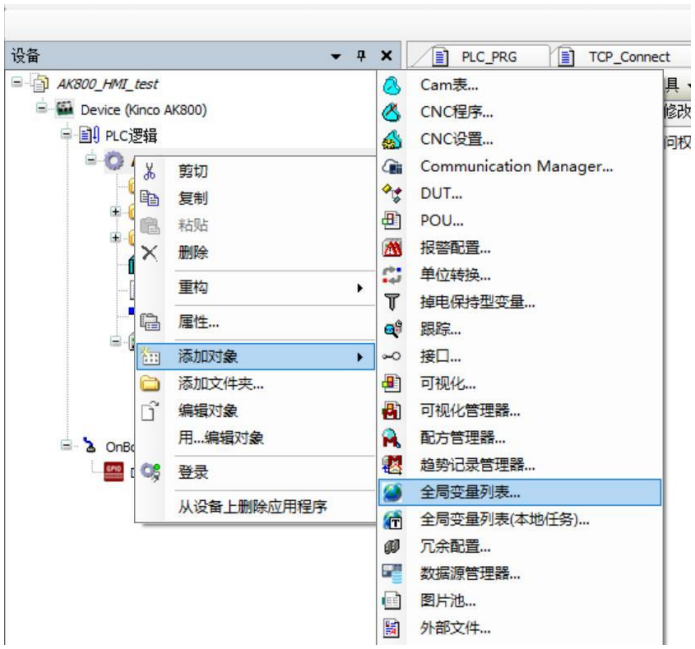


图 11.2-16a：创建全局变量表“GVL”1

图 11.2-16b：创建所需变量

图 11.2-16c：功能实现程序

（2）新建符号配置

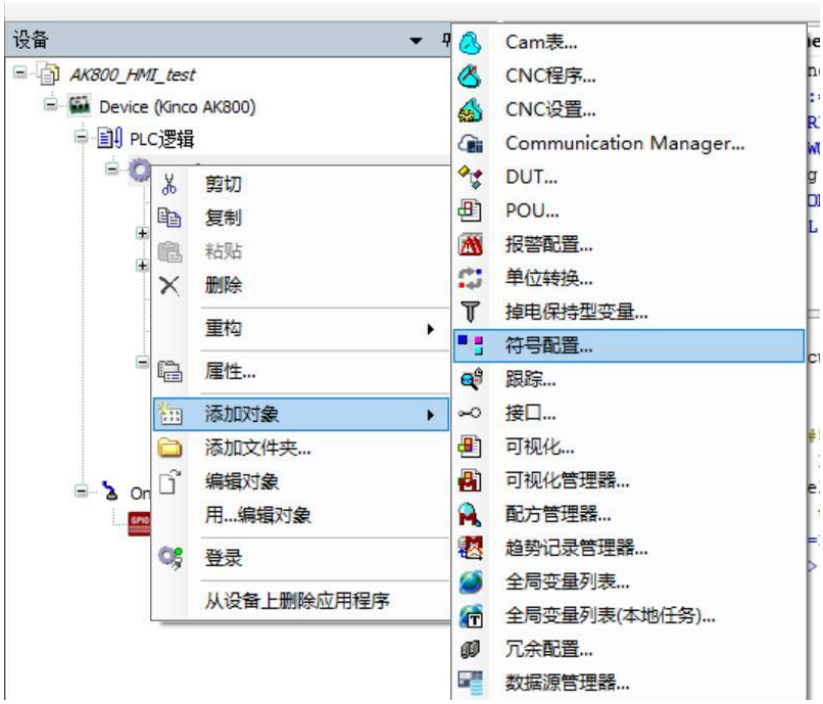


图 11.2-17：新建符号配置 a

图 11.2-18：新建符号配置 b

在符号配置窗口中，“编译”后选中“GVL”变量表，在此点击符号配置窗口上方的编译按钮，然后点击菜单栏的“生成代码”，在本工程的根目录处可找到“文件名.Application.xml”的文件。

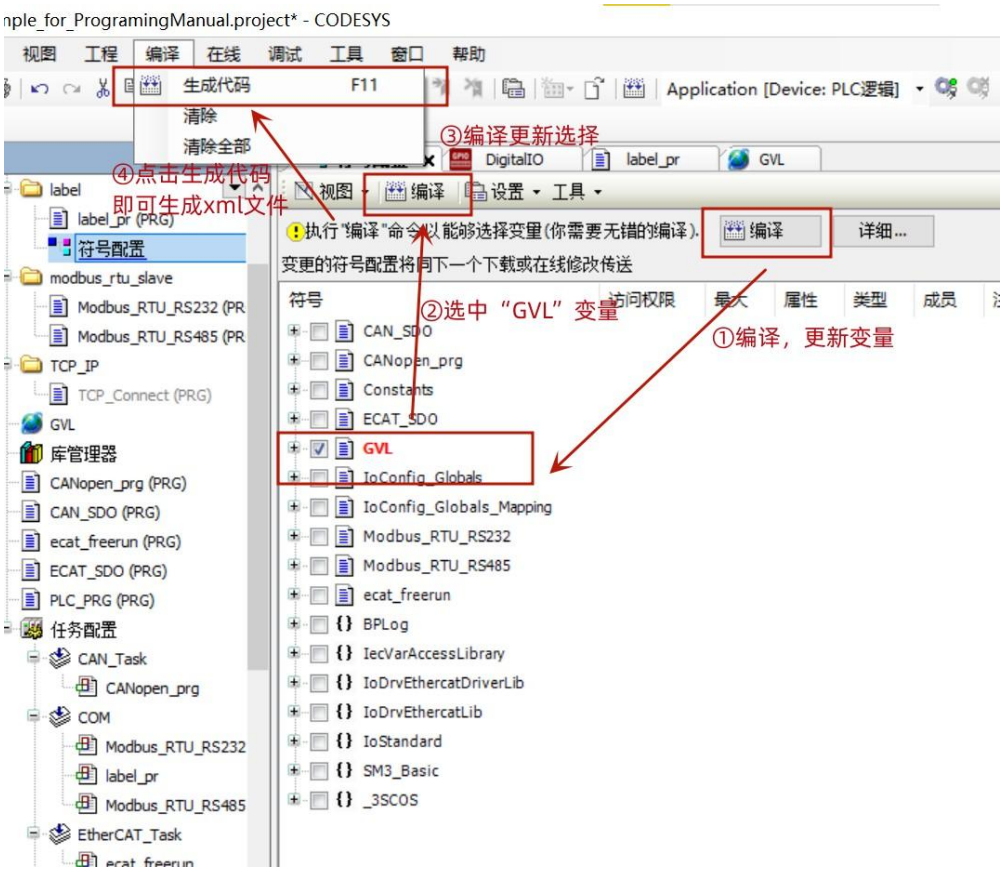


图 11.2-19：生成 .*XML 格式的标签文件

2.标签格式文件生成后，在 Dtools 中双击 PLC 设备打开设置，导入标签文件，然后进行 HMI 画面编程和地址对应，并将工程完整编译后下载到 HMI 中（过程略）。

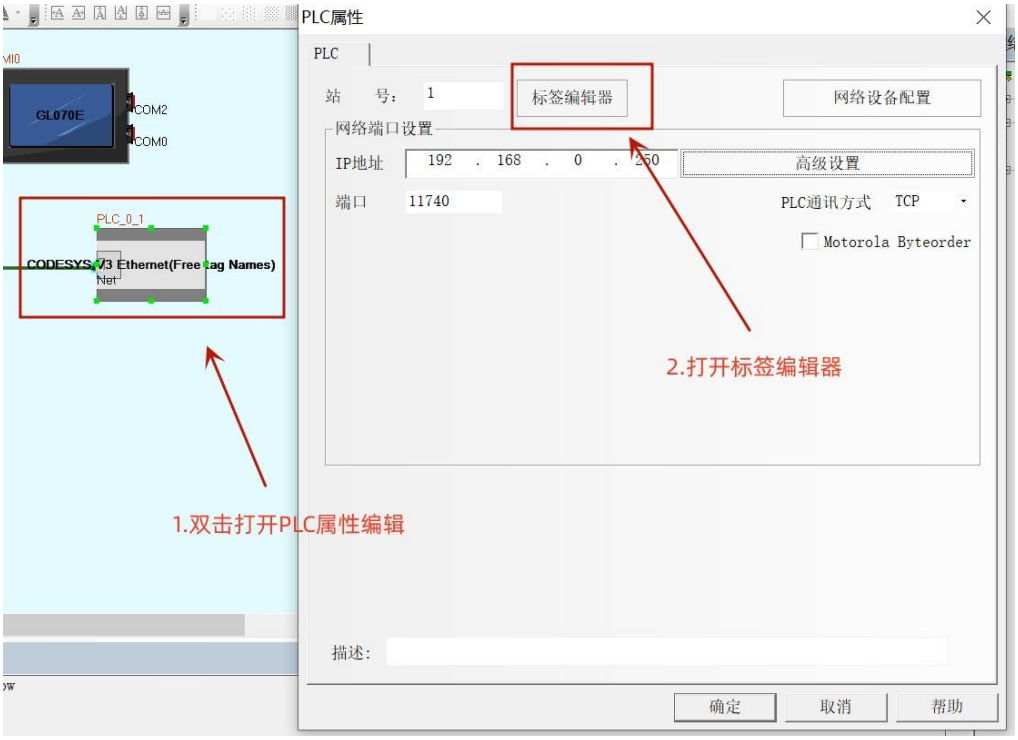


图 11.2-20a：Dtools 中导入标签文件

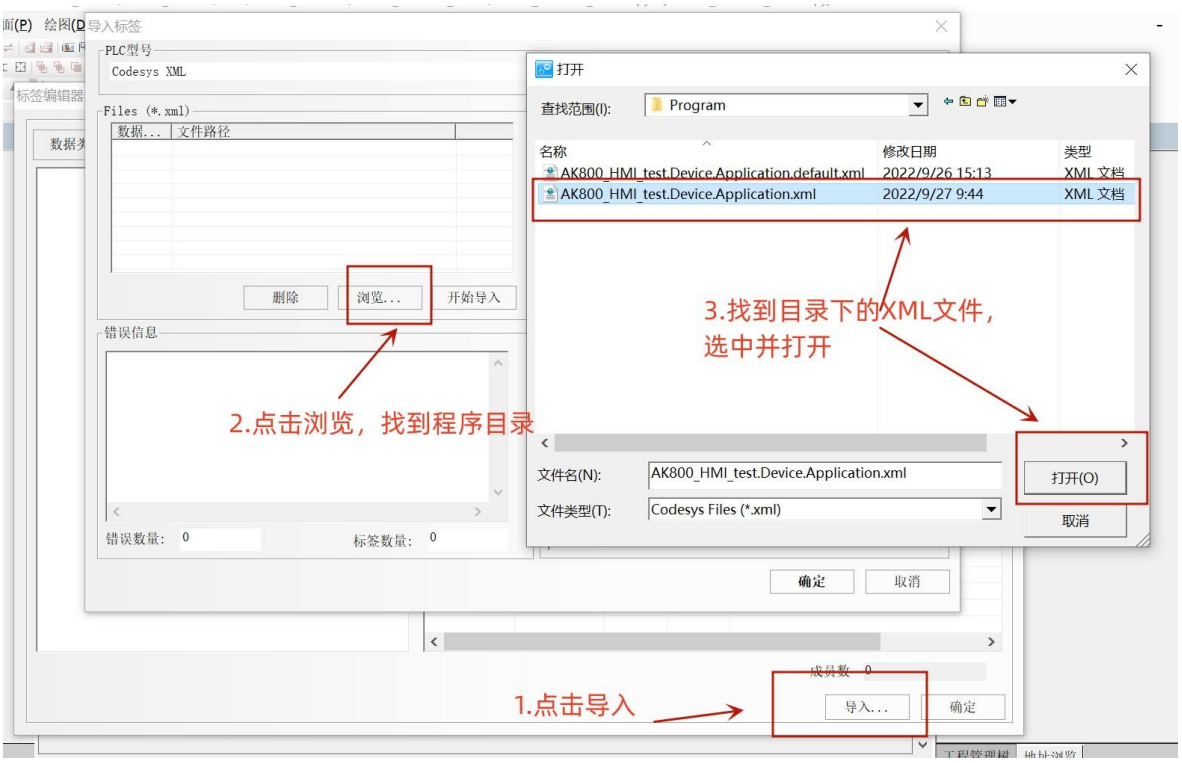


图 11.2-20b：Dtools 中导入标签文件

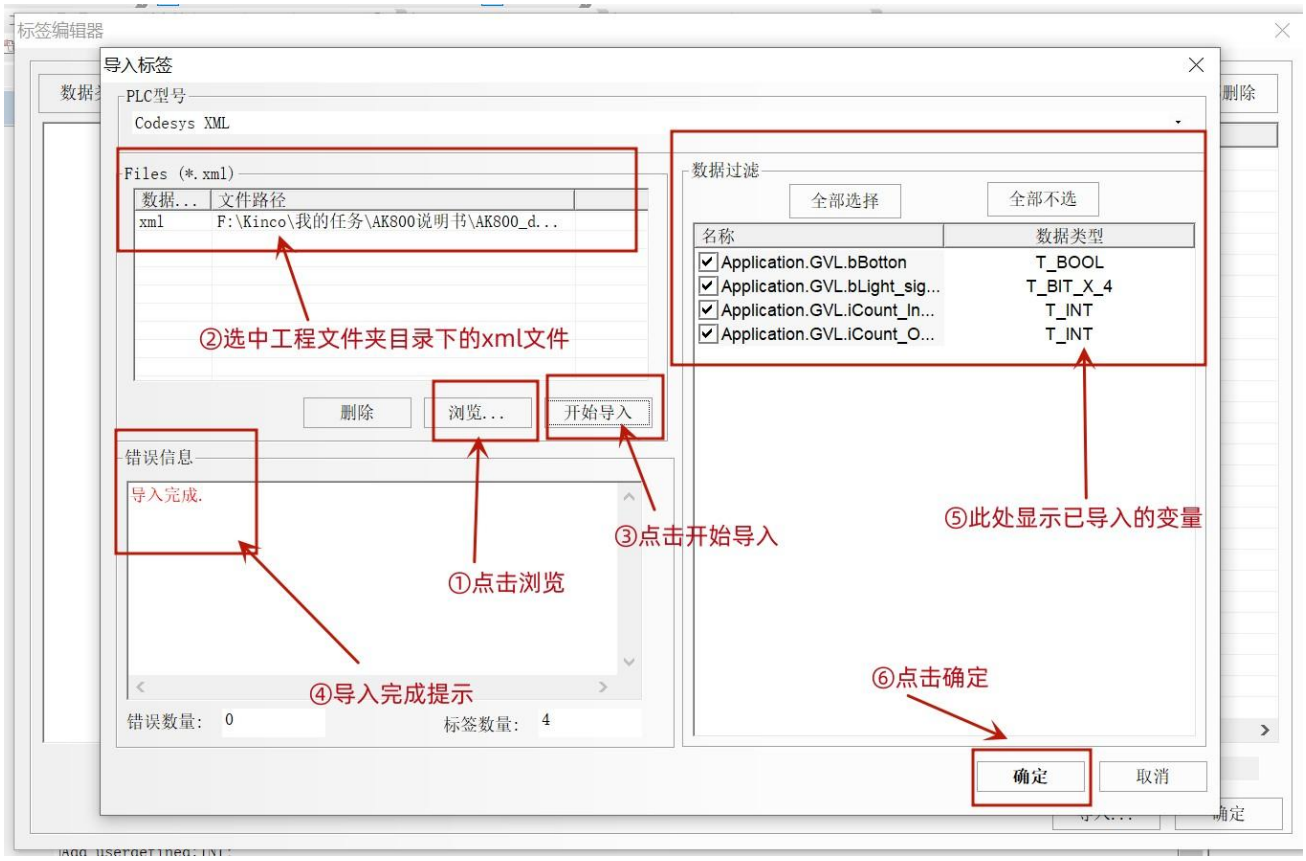


图 11.2-20c：Dtools 中导入标签文件

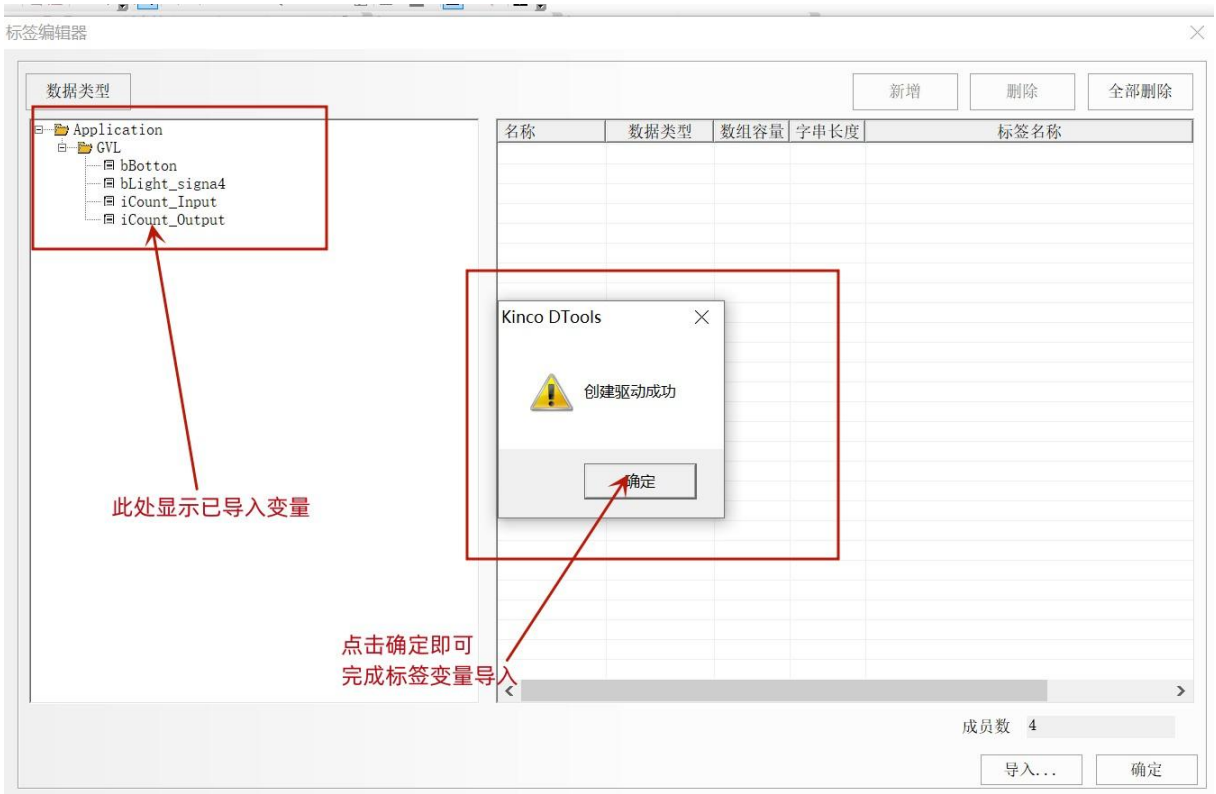


图 11.2-20d：Dtools 中导入标签文件

3.标签的使用

如下图所示，标签导入成功后，双击元件打开属性设置，在地址类型打开地址信息，可在“Application”下的“GVL”找到符合该元件类型的变量标签，选择需要的标签点击确定即可。

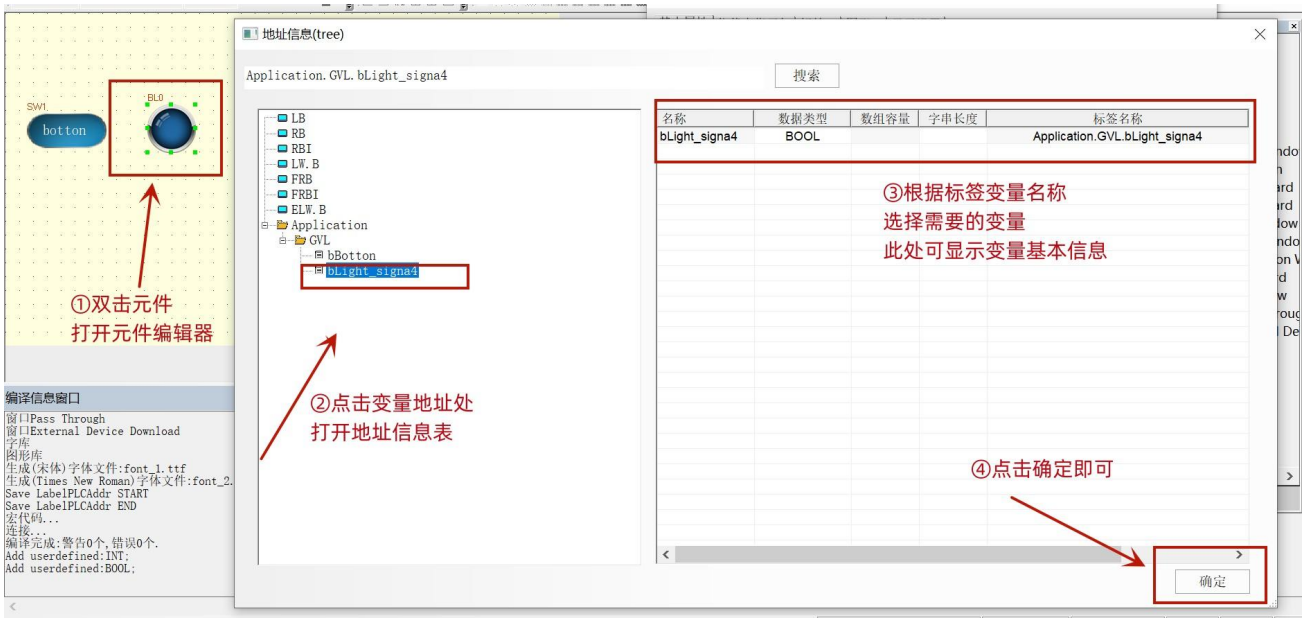


图 11.2-21：Dtools 中标签的使用

4.事项

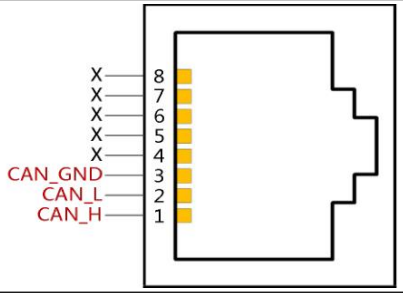
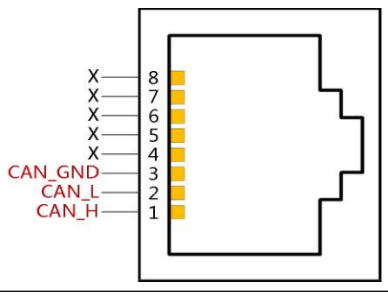
标签通讯要注意所用的 HMI 可兼容的数据类型，目前 Kinco 步科的 HMI 标签通讯支持常用的数据类型，包括功能块的输入输出，局部变量，复合数据类型等；暂不支持以下这几种类型：LDATE、LINT、LTIME、LWORD、ULINT、枚举。

另外，与 AK800 相关的硬件 IO（AK800 本体 IO 或外置的远程 IO）需要进行硬件地址实例化（变量名称 AT %QX(IX/MX) *.* : BOOL;）且添加在符号配置中一同生成，才可成功导出可应用标签变量。

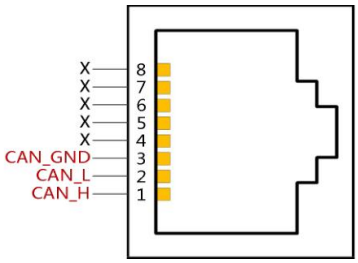
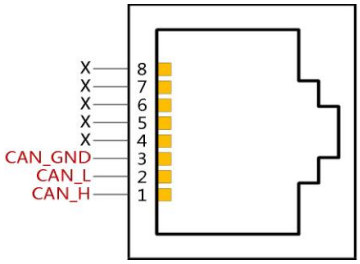
11.3 AK800 通过 CANopen 总线控制 Kinco FD425 系列伺服驱动器

11.3.1 CANopen 通讯线连接

1.AK800 总线 CAN 通讯接口引脚定义：

CAN0	CAN1
	

2.FD425-CA-000 总线 CAN 通讯接口引脚定义：

X4A(IN)	X4B(OUT)
	

AK800 控制器的 CAN0 和 CAN1 可以直接通过网线与 Kinco 伺服驱动器的 X4A(IN)口相连（推荐超五类屏蔽双绞线）。主站端和最后一个从站端不需要外接 120 欧姆的终端电阻，因为 AK800 控制器和 FD425-CA-000 伺服驱动器均内置终端电阻，FD425-CA-000 伺服驱动器的终端电阻通过 CAN 接口旁的拨码启动。同一路 CANopen 组网，只需要在首末设备两端接终端电阻，本例中只用到 AK800 和一个 FD425-CA-000 伺服驱动器，故将 FD425-CA-000 伺服驱动器的拨码拨置“ON”状态即可（即如果多个从站设备，只需对最末端设备的拨码开关进行设置）。如设备无内置终端电阻的，需要在最末端设备额外接一个 120Ω 的终端电阻，具体接线方式参考 11.4.1 章节中图 11.4-1。

11.3.2 CANopen Manager 功能及配置

1.添加从站 EDS 文件

Kinco 伺服的 EDS 文件可以在步科官网下载，地址：http://www.kinco.cn 。

（1）点击工具菜单下的“设备库”，点击“安装”，在弹出的浏览对话框中找到对应 EDS 文件，选中并打开，等待安装完成；

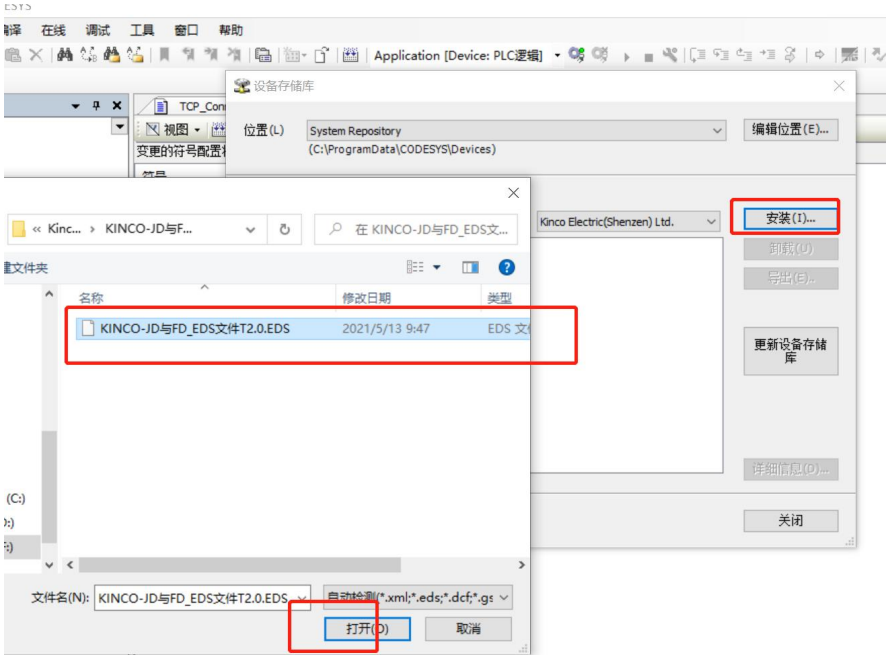


图 11.3-1：安装设备 EDS 文件

（2）EDS 文件安装成功后就可以在设备库列表中查看到 KINCO 伺服。

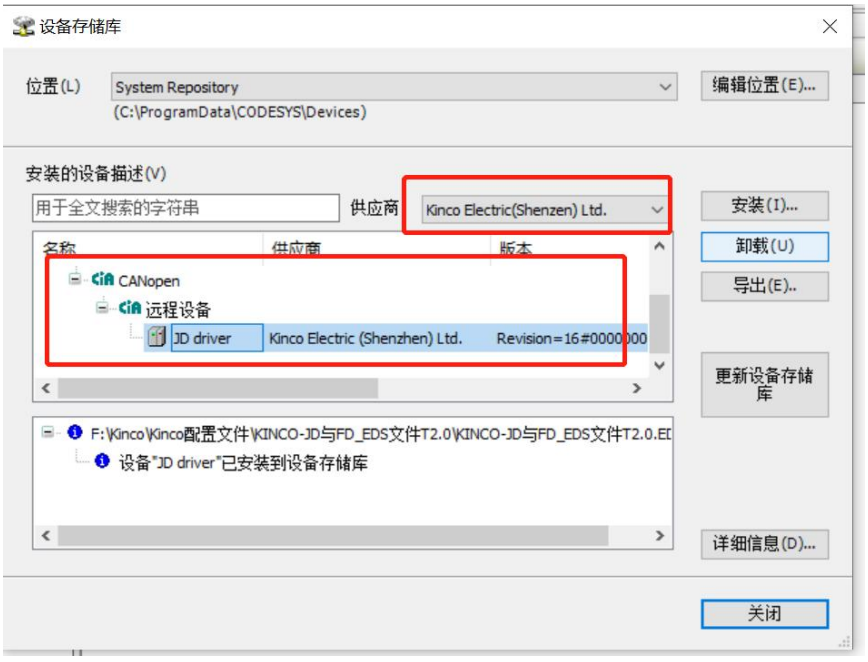


图 11.3-2：CANopen 伺服设备安装成功

2.添加“CANopen_Manager”设备

（1）右击“Device”，选择“添加设备”，在弹出的“添加设备”窗口中切换到“全部供应商”，在其目录下找到“现场总线”，选择目录下的“CANbus”，选中点击“添加设备”。

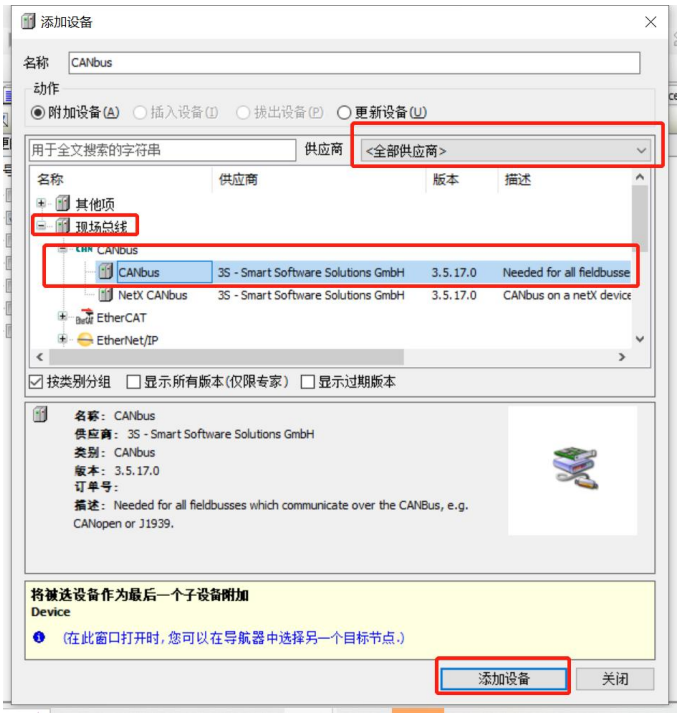


图 11.3-3：添加“CANbus”设备标识符

（2）右击设备树中新添加的设备“CANbus”，选择“添加设备”，在弹出的窗口中，按照相同路径找到“CANopen 管理器”，选择其目录下的“CANopen_Manager”，选中并添加。

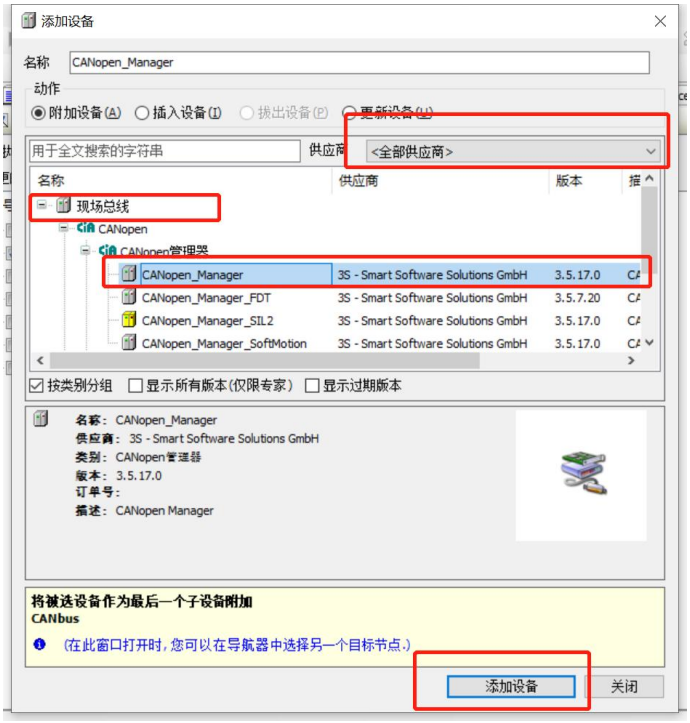


图 11.3-4：添加“CANopen_Manager”设备标识符

3.配置 CANopen 通讯

（1）双击“CANbus”，在“通用”设置中设置 CANopen 网络号（其中网络 0 表示 AK800-CAN0，网络 1 表示 AK800-CAN1），及波特率（该波特率需与所挂设备的波特率一致）。

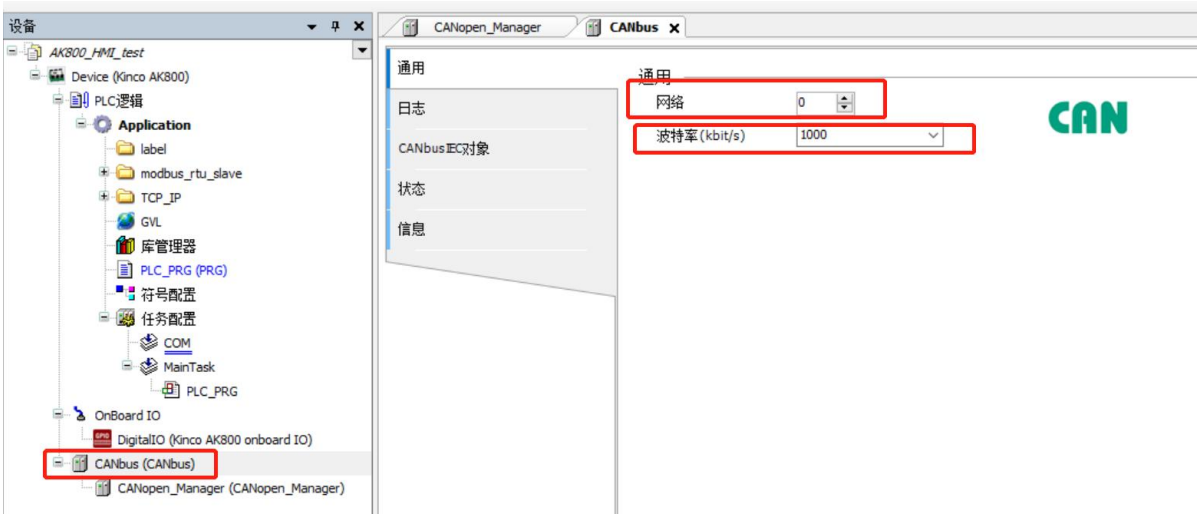


图 11.3-5：“CANbus”通用设置

（2）添加 Kinco 驱动器设备

右击“CANopen_Manager”，选择“添加设备”，在“供应商”处找到“Kinco Electric(Shenzhen) Ltd”，选择其目录下的“JD drive”设备，点击“添加”。

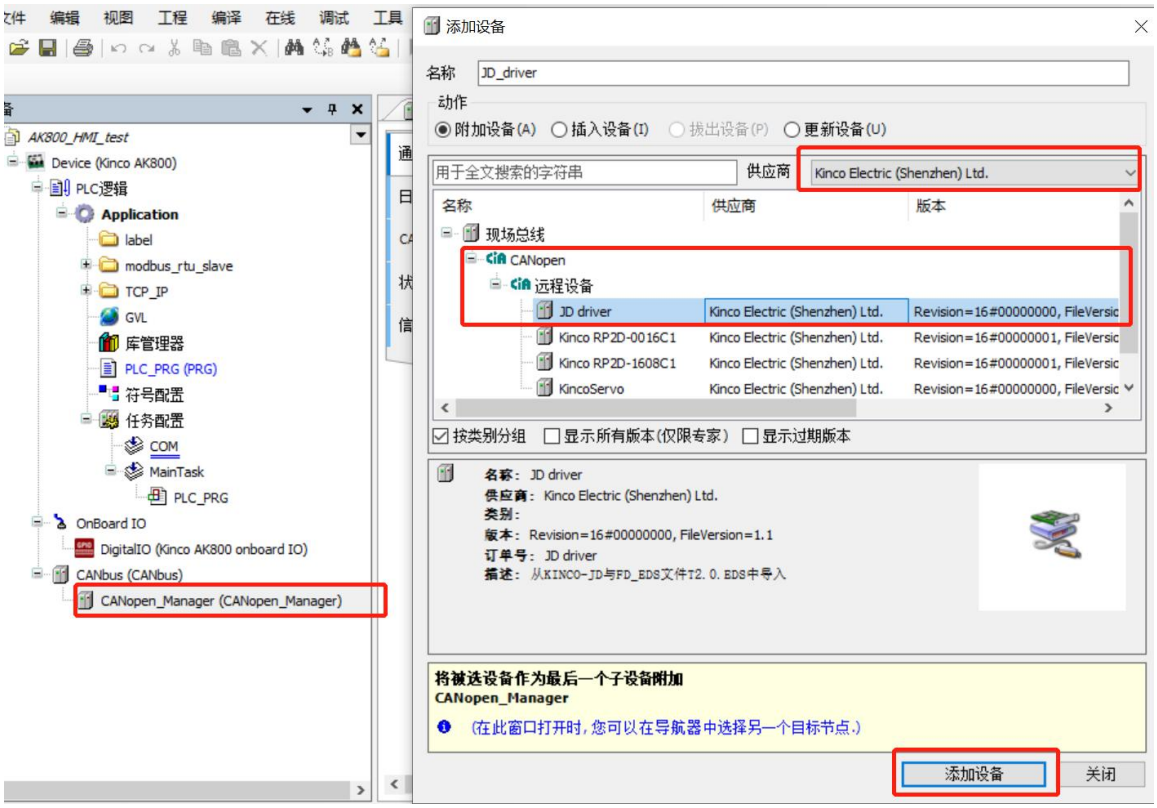


图 11.3-6：添加 Kinco 驱动器设备

（3）“JD Drive”从站设备通用配置

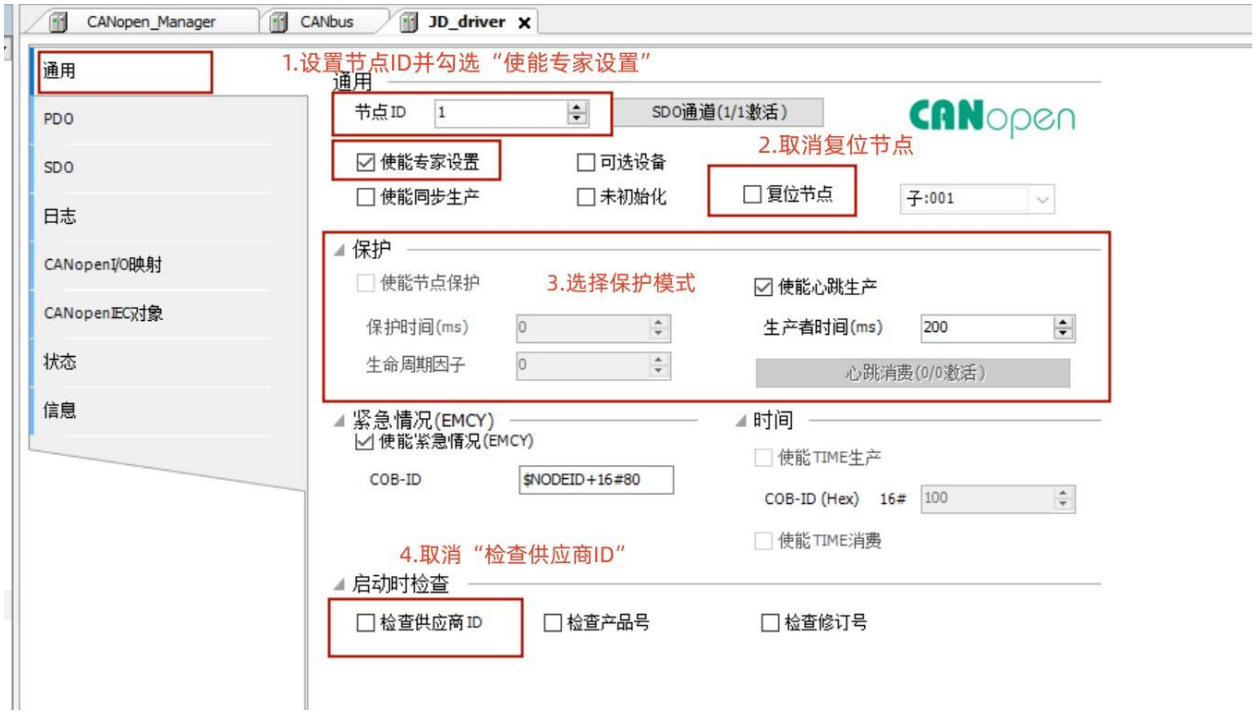


图 11.3-7：设备通用配置

（4）“JD drive”驱动 PDO 配置：如下图，可以在 PDO 配置界面对驱动的 PDO 进行自主配置。

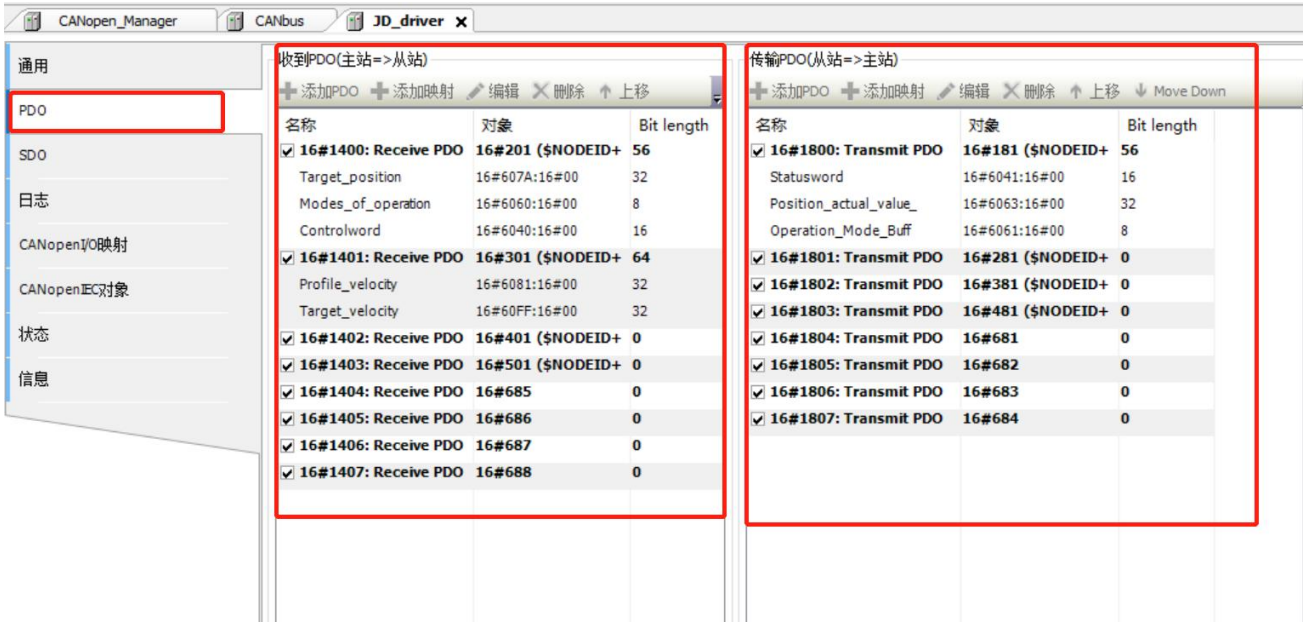


图 11.3-8：设备 PDO 配置

配置从站 PDO 参数，注意 PDO 中所配对象的长度总和不能超过 64 个字节。如果是标准 CANopen 伺服从站，推荐按如下表中的参数配置 RPOD 和 TPDO。

表 11.3-1：推荐的 RPDO 与 TPDO 配置

RPDO1	607A0020	Target Positon	目标位置
	60400010	Control Word	控制字
	60600008	Mode of Operation	工作模式
	60980008	Homeing Method	回原点方式
RPDO2	60810020	Profile Velocity	位置模式速度
	60FF0020	Target Velocity	速度模式速度
TPDO1	60410010	StatusWord	状态字
	60630020	Position Actual Value	实际位置

配置 PDO 的操作如下图所示：

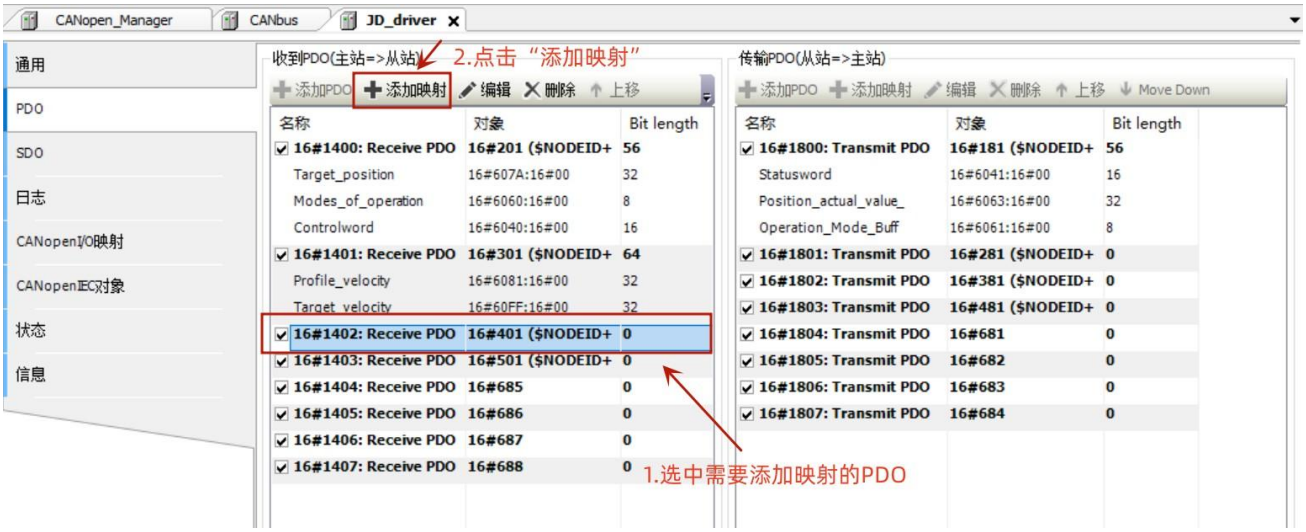


图 11.3-9：PDO 配置操作-a

图 11.3-10：PDO 配置操作-b

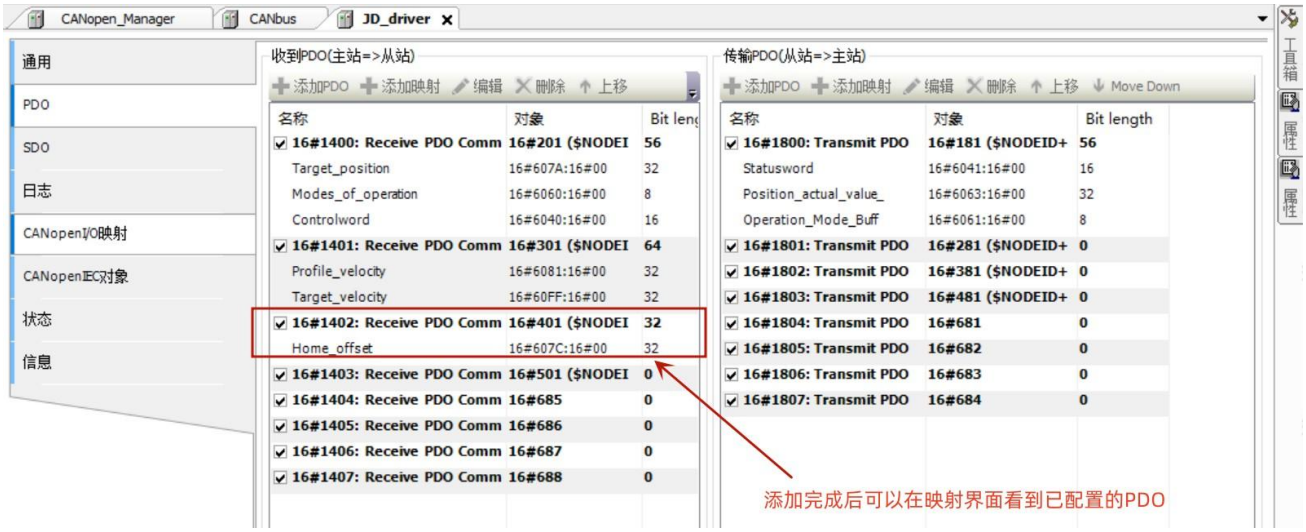


图 11.3-11：PDO 配置操作-c

（5）注意事项

对 PDO 还可以进行其它属性的编辑，**如 TPDO 需要注意的地方：对添加有实际位置或实际速度等实时反馈对象的 PDO 中必须设置 Inhibit time（禁止时间），这样就可以防止伺服发送通讯报文过快而导致通讯阻塞，出现 CAN 通讯异常**

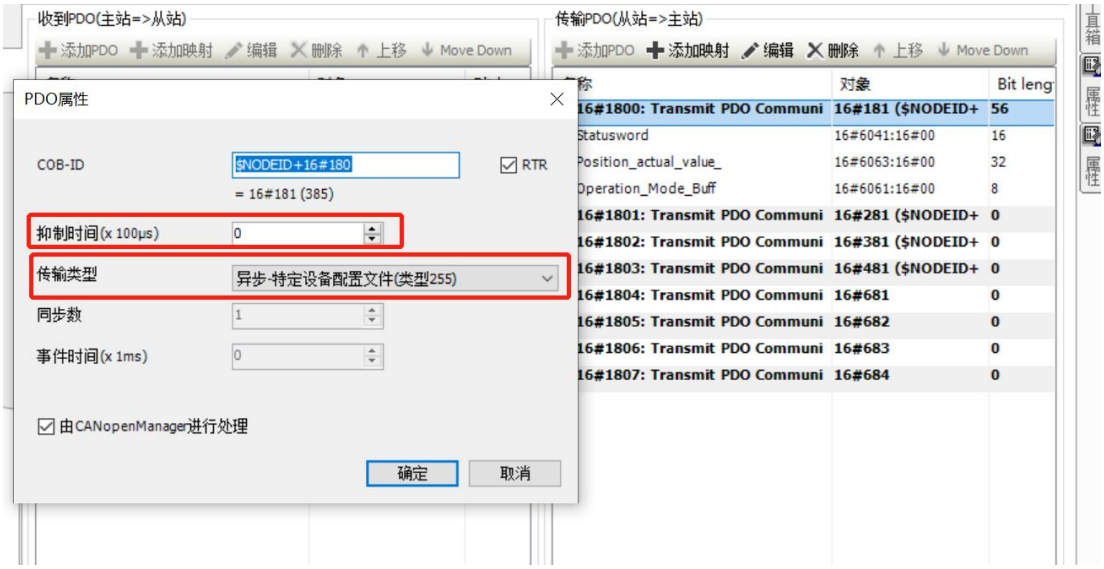


图 11.3-12：PDO 其他配置

***Inhibit time（禁止时间）的单位 PLC 默认为 100μs，但部分伺服可能无法支持，如 Kinco 伺服为 ms，图例中 20 表示 TPDO 最小报文间隔时间为 20ms。**

（6）查看配置：在“CANopen I/O Mapping(映射）”中可查看配置好的从站各 PDO 对象以及对应关联的主站。

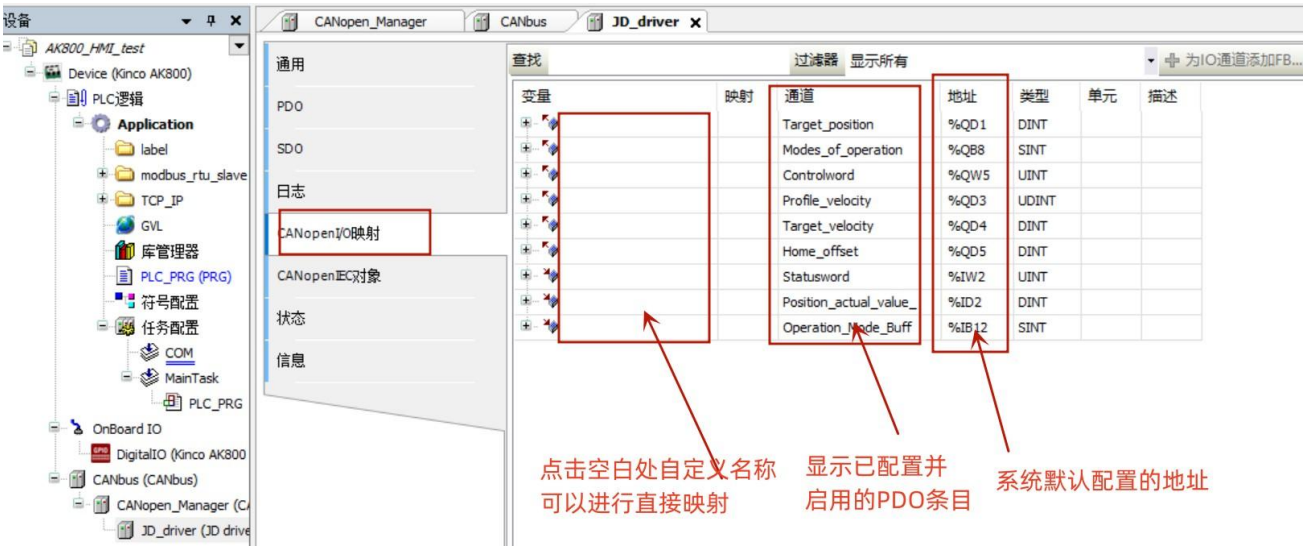


图 11.3-13：PDO 映射界面

11.3.3 Kinco 伺服驱动器设置（KincoServo+）

1.伺服从站的站号设置

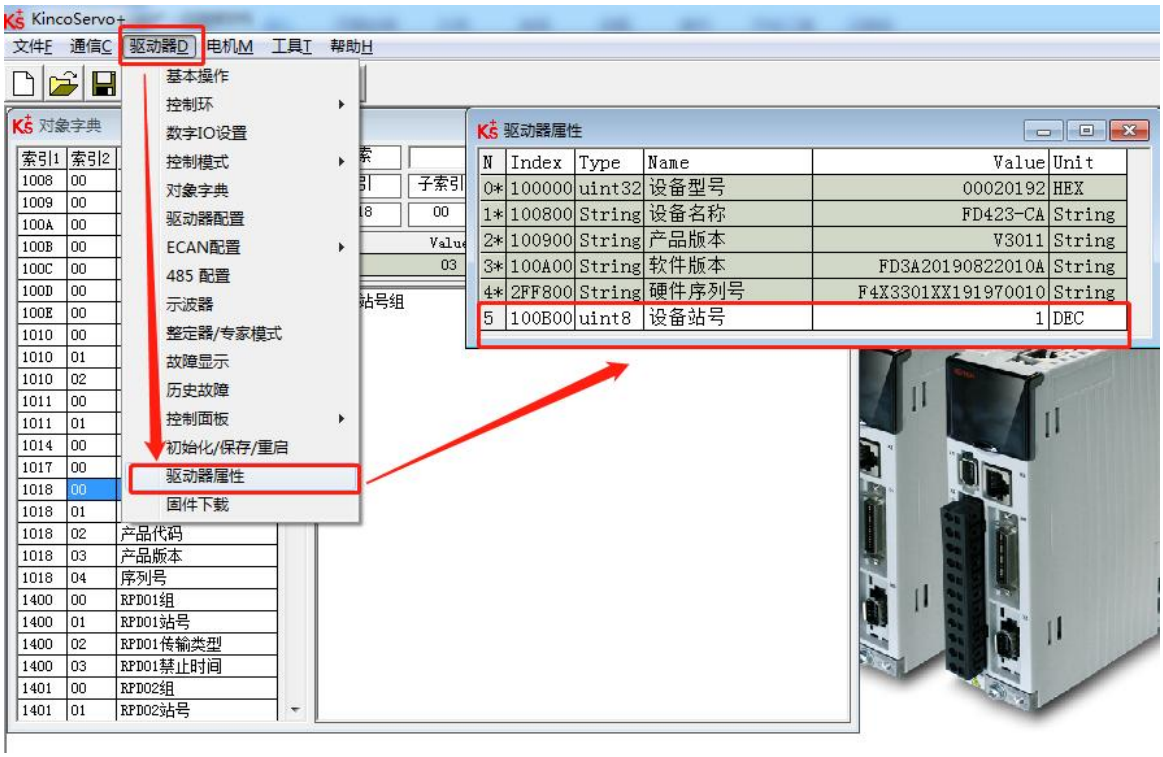


图 11.3-14：伺服站号配置界面

2.伺服从站的默认数字量输入配置要清空，若有其他需求可以再加。

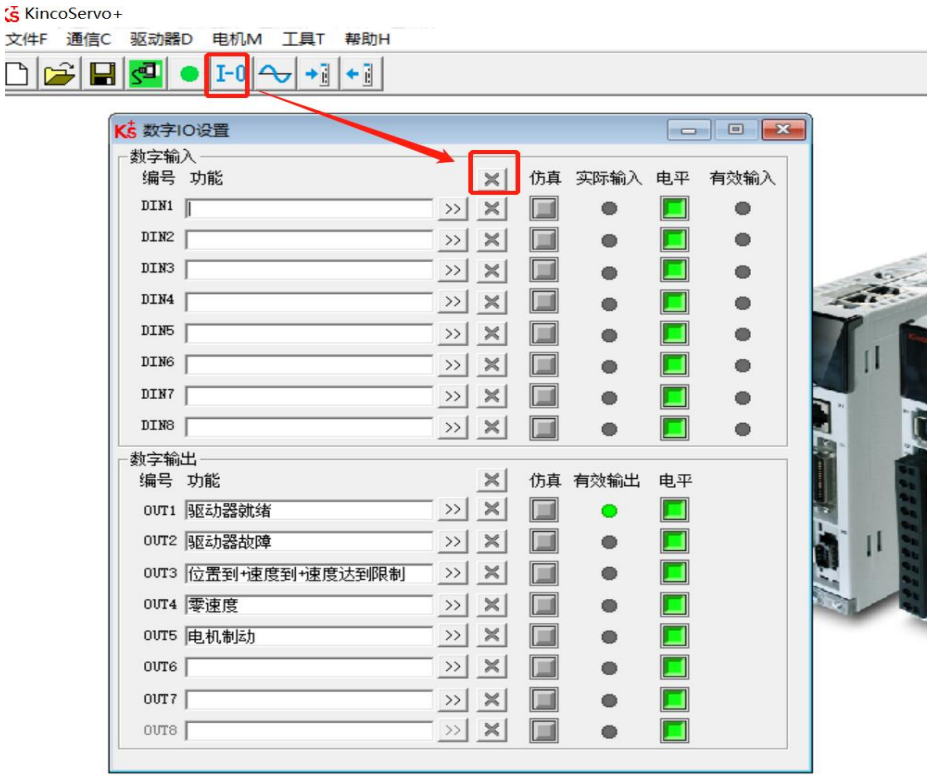


图 11.3-15：伺服 IO 配置界面

3.伺服从站的 CAN 波特率设置步骤（伺服参数中 50 即为 500Kbits/s），Kinco 伺服支持 50 Kbits/s、125 Kbits/s、250 Kbits/s、500 Kbits/s、1Mbits/s，注意，此处的波特率设置要与上文中“CANbus”的波特率一致。

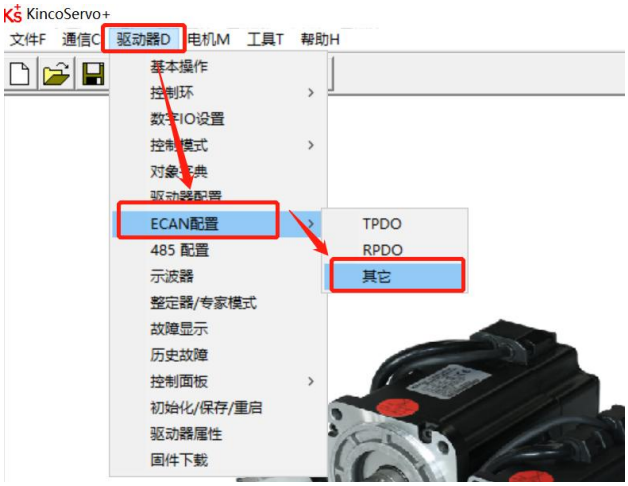
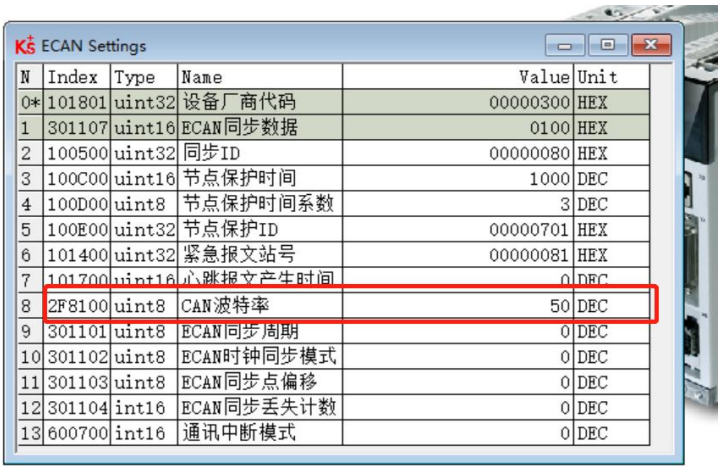


图 11.3-16：ECAN 配置路径



N	Index	Type	Name	Value	Unit
0*	101801	uint32	设备厂商代码	00000300	HEX
1	301107	uint16	ECAN同步数据	0100	HEX
2	100500	uint32	同步ID	00000080	HEX
3	100C00	uint16	节点保护时间	1000	DEC
4	100D00	uint8	节点保护时间系数	3	DEC
5	100E00	uint32	节点保护ID	00000701	HEX
6	101400	uint32	紧急报文站号	00000081	HEX
7	101700	uint16	心跳报文产生时间	0	DEC
8	2F8100	uint8	CAN波特率	50	DEC
9	301101	uint8	ECAN同步周期	0	DEC
10	301102	uint8	ECAN时钟同步模式	0	DEC
11	301103	uint8	ECAN同步点偏移	0	DEC
12	301104	int16	ECAN同步丢失计数	0	DEC
13	600700	int16	通讯中断模式	0	DEC

图 11.3-17：配置 CAN 波特率

11.3.4 编写简易的 CANopen 运动控制程序

1.运动控制程序配置如下：

- 波特率：1M/bits；
- 驱动器节点号：1；
- PDO 配置如下图所示：

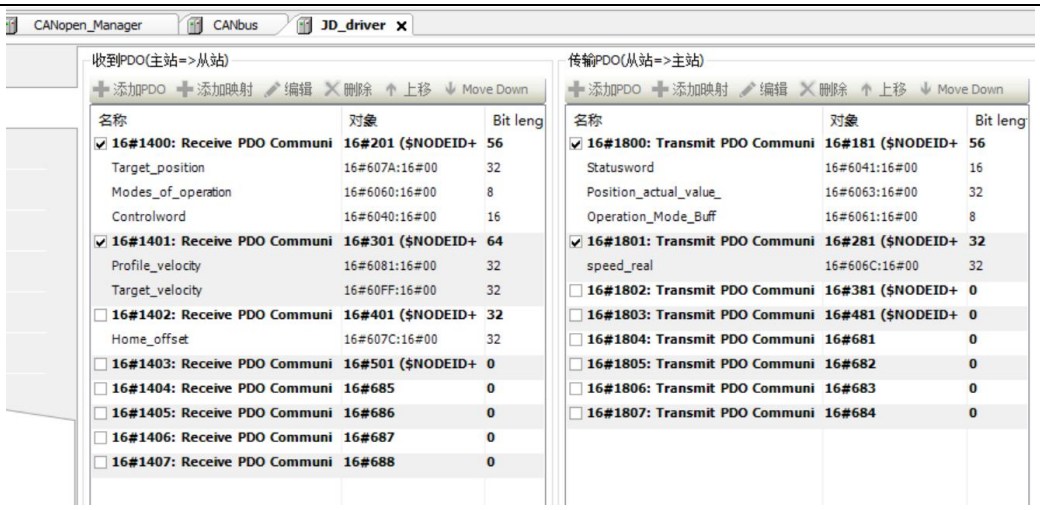


图 11.3-18：示例程序 PDO 配置

- 映射标签如下图所示：

图 11.3-19：标签地址映射

- 驱动器配置如下图所示：

图 11.3-20：示例程序驱动器配置

2.编写简单的 CANopen 运动控制程序，以 CANopen 通讯方式控制驱动器带动电机在速度模式下进行正反转及停止操作。注意 PLC 程序需要先向伺服控制字写入“6”之后才能正常控制伺服。具体程序编写如下图所示：

```
PROGRAM CANopen_prg
VAR
    tBootup_dealy     : Standard.TON;
    bBootup           : BOOL;
    bJog_R            : Standard.R_TRIG;
    bJog              : BOOL;
    diJog_Vel         : DINT:=178960;//FD425 驱动：1rpm=17896DEC
    bstp              : BOOL;END_VAR

    //上电初始化延迟
    tBootup_dealy(IN:=NOT bBootup, PT:=T#1S, Q=>, ET=>);
    IF tBootup_dealy.Q THEN
        bBootup:=TRUE;
        CAN_Axis1control_word:=6;//为驱动的 controlword 赋初值：6
    ELSIF NOT bBootup THEN
        RETURN;
    END_IF

    //速度模式正反转：反转时速度给负值
    bJog_R(CLK:=bJog, Q=>);
    IF bJog_R.Q THEN
        CAN_Axis1operation_mode:=3;//写入速度模式：3
        CAN_Axis1target_vel:= diJog_Vel;//速度值
        CAN_Axis1control_word:=16#F;//控制字：锁轴
        bJog:=FALSE;
    END_IF

    //停止
    IF bStp THEN
        CAN_Axis1operation_mode:=1;//
        CAN_Axis1target_vel:= 0;//速度值
        CAN_Axis1control_word:=16#6;//控制字：松轴
        bstp:=FALSE;
    END_IF
```

3.登录下载并运行程序，尝试置位变量“bJog”和“bStp”，电机将以 10rpm 运行及停止。

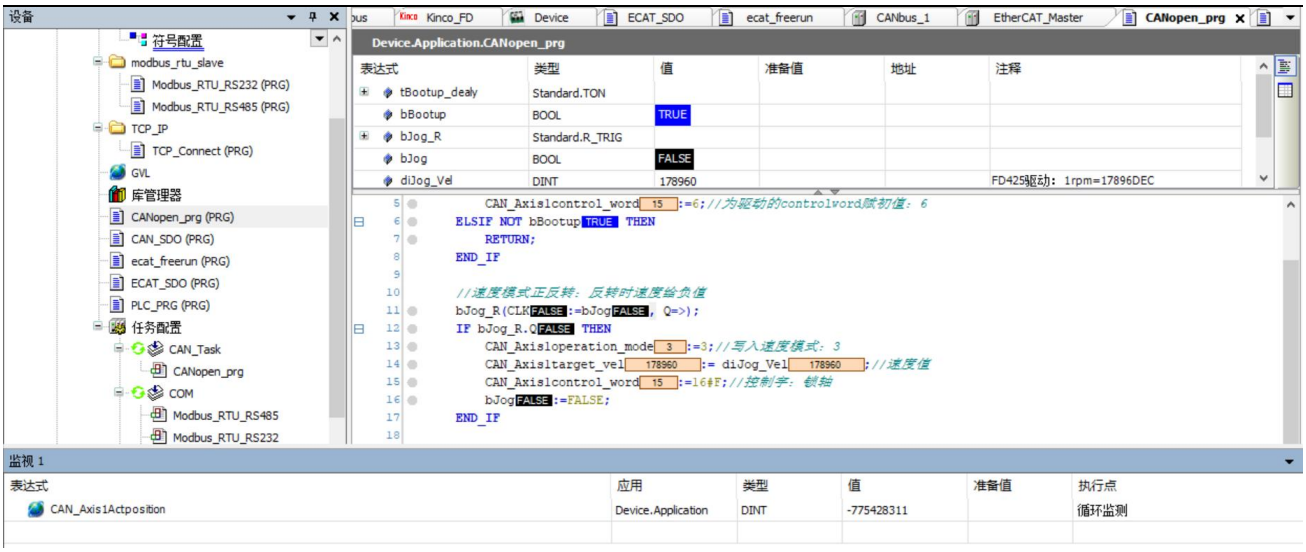


图 11.3-21：示例程序运行状态

11.3.5 CANopen SDO 读写程序示例

下面通过简单的 SDO 读写，以 AK800 示范“CAA_Cia405.lib”中的“SDO_WRITE4”和“SDO_READ4”的用法。程序要求：通过写 SDO 修改原点模式；在伺服驱动运行状态下通过 SDO 读取当前的有效工作模式。具体程序如下：

```
PROGRAM CAN_SDO
VAR
    COE_Read1          :    ETC_CO_SdoRead4;
    COE_Write1         :    ETC_CO_SdoWrite4;
    bExecute_r         : BOOL;
    bExecute_w         : BOOL;
    byAr_COE_rDtata  : ARRAY [1..4] OF BYTE;
    byAr_COE_WDtata: ARRAY [1..4] OF BYTE;
    usiDATA_rLength  : USINT;
    usiwDATA_wLength: USINT;
    bDone_r            : BOOL;
    bError_r           : BOOL;;
    vERROR_r           : ETC_CO_ERROR;
    bDone_w            : BOOL;
    bError_w           : BOOL;
    vError_w           : ETC_CO_ERROR;END_VAR
END_VAR
```

```
    //通过 SDO 修改原点模式：16#609800
    Write_Sdo1(
        NETWORK:= 1,          //CANOPEN 网络号,1 对应 CAN0,2 对应 CAN1
        ENABLE:= bWrite_sdo,
        TIMEOUT:= 500,        //超时时间
        CONFIRM=> bConfirm_w,
        ERROR=> vError_w,
        DEVICE:= 16#01,         //设备站号
        CHANNEL:= 0,          //通道
        INDEX:= 16#6098,      //索引
        SUBINDEX:= 16#00,   // 子索引
        DATA:= byAr_wSdo_data,   //目标数组位置，
        DATALENGTH:= usiSdo_wLength,//数据长度 ,单位 BYTE,1 代表 1BYTE=8BIT
        ERRORINFO=> vErrorinfo_w);
```

```
//通过 SDO 读取有效工作模式：16#606100
Read_Sdo1(
    NETWORK:= 1,
    ENABLE:= bRead_sdo,
    TIMEOUT:= 500,
    CONFIRM=> bConfirm_r,
    ERROR=> vERROR_r,
    DEVICE:= 16#01,
    CHANNEL:= 0,
    INDEX:= 16#6061,
    SUBINDEX:= 16#00,
    DATA=> byAr_rSdo_data,
    DATALENGTH=> usiSdo_rLength,
    ERRORINFO=> vErrorinfo_r);
```

程序运行效果如下图 11.3-22 和 11.3-23：

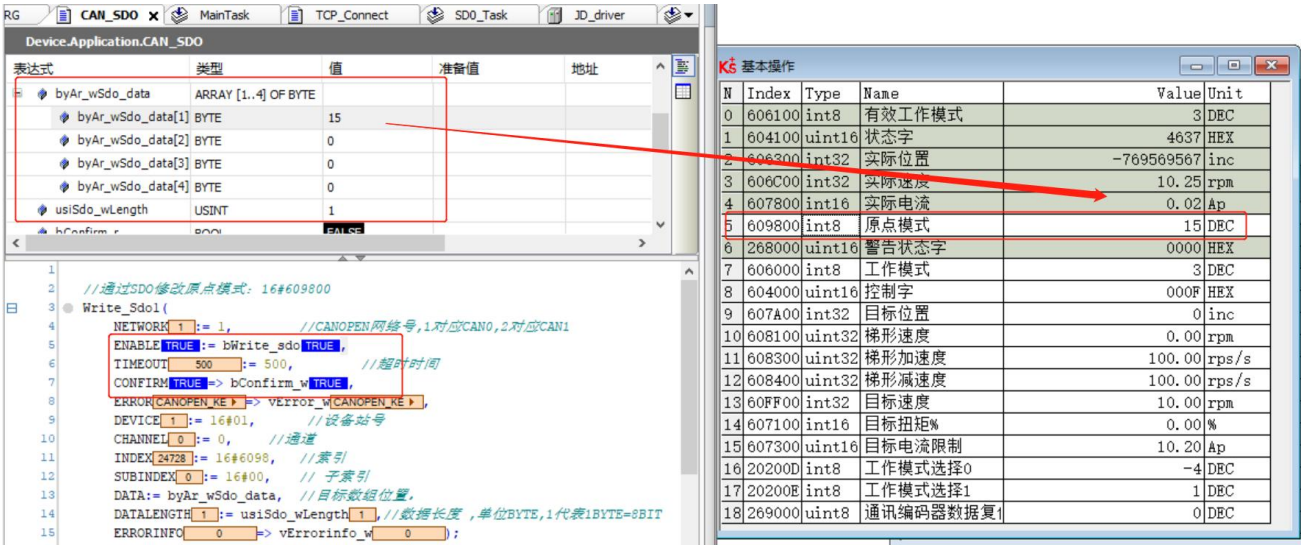


图 11.3-22：示例程序写入“15”号原点模式

如上图，将“byAr_wSdo_data[1]”赋值 15，代表要写入的 15 号原点模式，之后执行“bWrite_sdo”，执行 SDO 写入功能块，可在伺服软件（Kinco Servo+）处实时观测到原点模式（16#609800）被修改为 15。

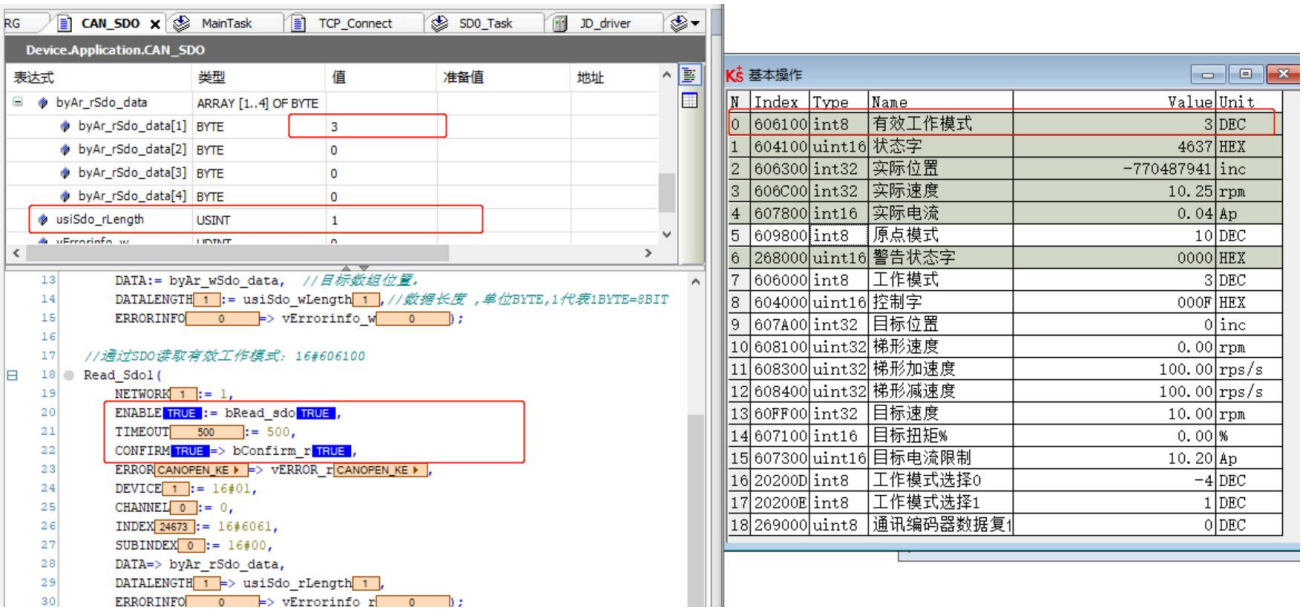


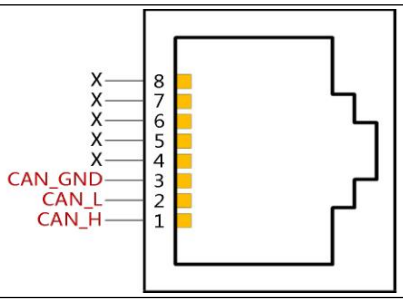
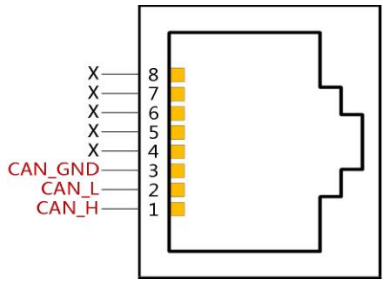
图 11.3-23：示例程序读取有效工作模式

如上图，在伺服运行状态中，执行“bRead_sdo”，执行 SDO 读取功能块，读取有效工作模式（16#606100），可见到程序中的 SDO 读取数组“byAr_rSdo_data[1]”成功读取“3”，与伺服当前有效工作模式对应，“usiSdo_rLength”为读取的数据长度，“1”表示 1Byte。

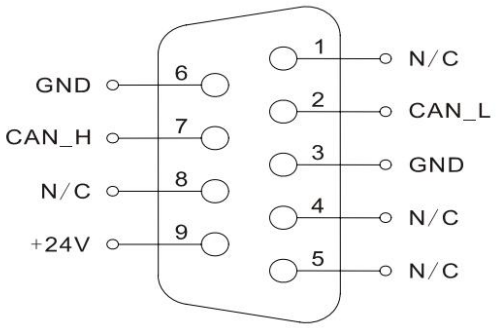
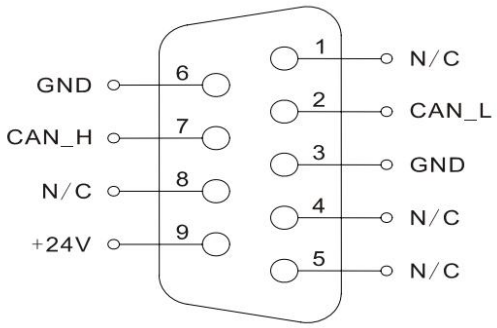
11.4 AK800 通过 CANopen 总线连接 Kinco RP2D 系列远程 IO 模块

11.4.1 CANopen 通讯线连接

1.AK800 CAN 总线通讯接口引脚定义：

CAN0（RJ45）	CAN1（RJ45）
	

2.PR2D-1608C1 远程模块的 CAN 总线通讯接口引脚定义：

IN（DB9）	OUT（DB8）
	

AK800 控制器与 Kinco RP2D 系列远程 IO 模块连接时，AK800 端为 RJ45 接口，RP2D 端为 DB9 接口，AK800 已内置了 120 欧姆的终端电阻，还需要在最末端的 Kinco RP2D 系列远程 IO 模块（IN）端（如接入多个，本例中只接一个设备进行通讯演示）接入一个 120Ω 的终端电阻。

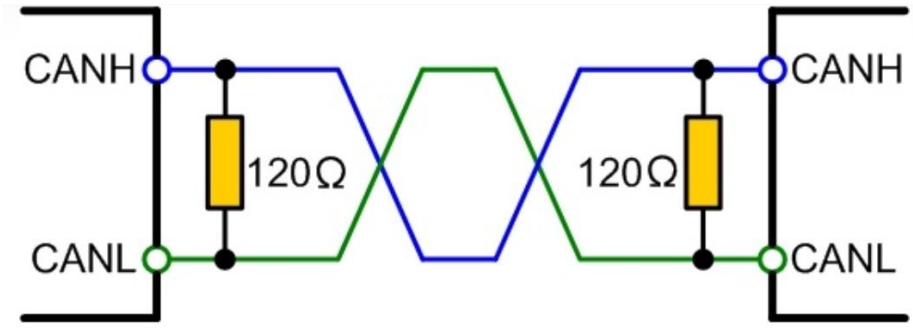


图 11.4-1：CANopen 终端电阻接线

11.4.2 CANopen Manager 功能及配置

1.添加从站 EDS 文件

Kinco RP2D 的 EDS 配置文件可以在步科官网下载，地址：http://www.kinco.cn 。

（1）点击工具菜单下的“设备库”，点击“安装”，在弹出的浏览对话框中找到对应 EDS 文件，选中并打开，等待安装完成；

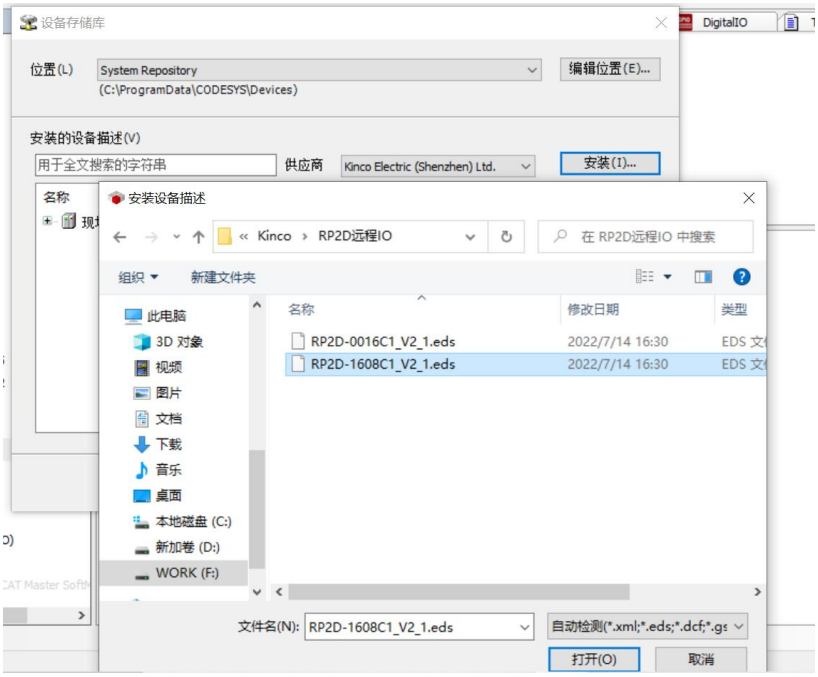


图 11.4-2：安装 RP2D 的 EDS 配置文件

（2）EDS 文件安装成功后就可以在设备库列表中查看到设备。

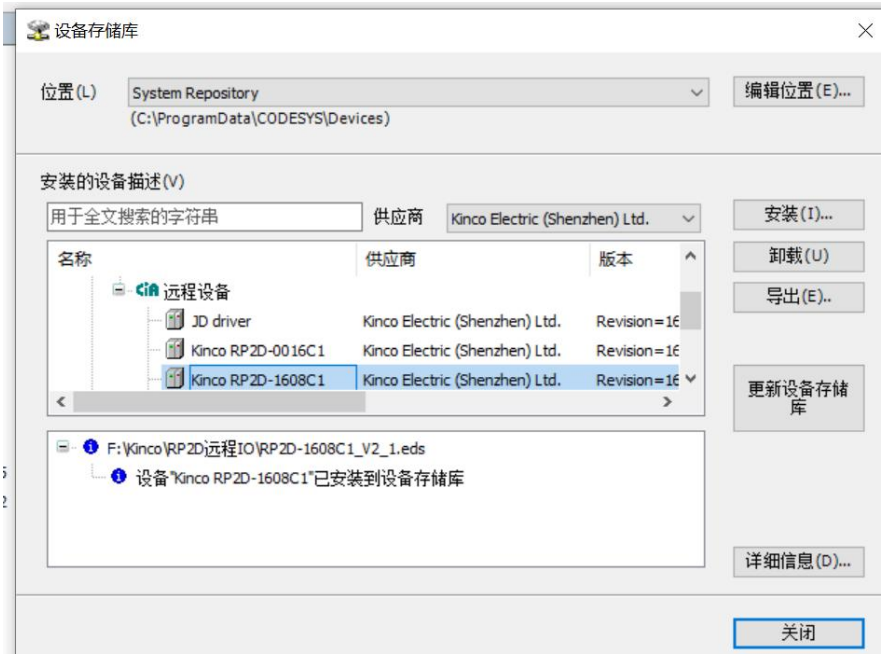


图 11.4-3：RP2D 设备安装成功

2.添加“CANopen_Manager”设备

（1）右击“Device”，选择“添加设备”，在弹出的“添加设备”窗口中切换到“全部供应商”，在其目录下找到“现场总线”，选择目录下的“CANbus”，选中点击“添加设备”。

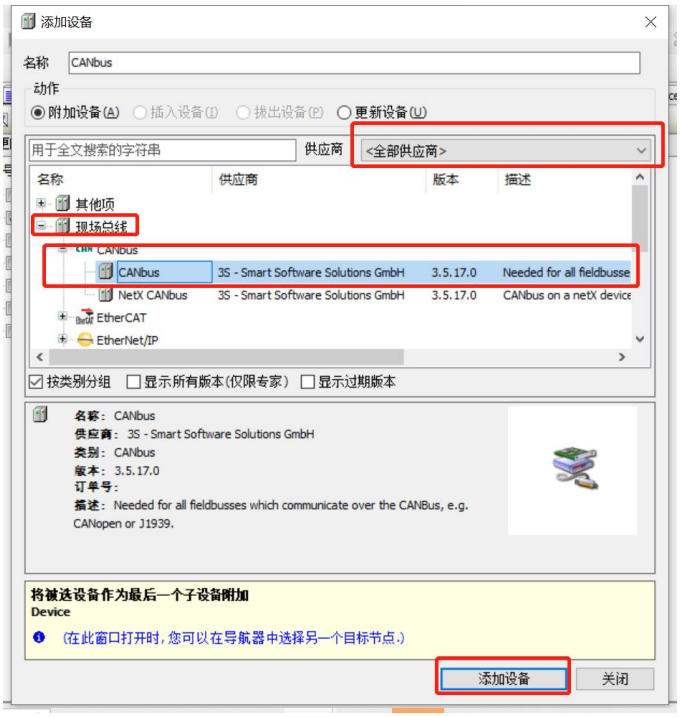


图 11.4-4：添加“CANbus”设备标识符

（2）右击设备树中新添加的设备“CANbus”，选择“添加设备”，在弹出的窗口中，按照相同路径找到“CANopen 管理器”，选择其目录下的“CANopen_Manager”，选中并添加。

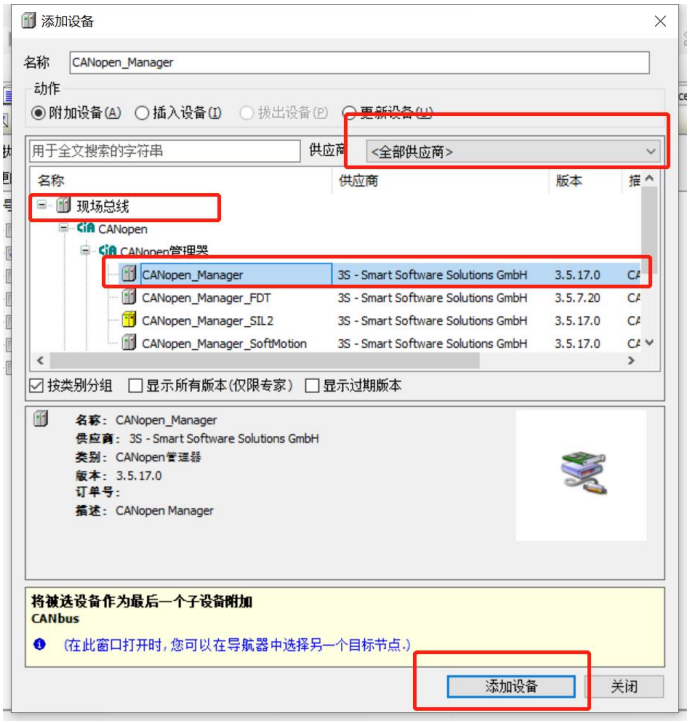


图 11.4-5：添加“CANopen_Manager”设备标识符

3.配置 CANopen 通讯

（1）双击“CANbus”，在“通用”设置中设置 CANopen 网络号（其中网络 0 表示 AK800-CAN0，网络 1 表示 AK800-CAN1），及波特率（该波特率需与所挂设备的波特率一致），本例中使用 CAN1 接口。

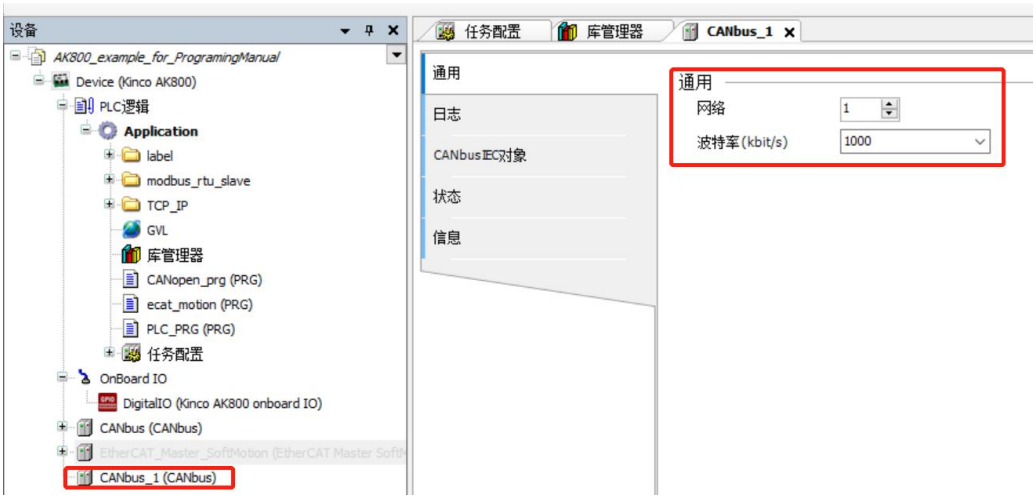


图 11.4-6：“CANbus”通用设置

（2）添加远程设备

右击“CANopen_Manager”，选择“添加设备”，在“供应商”处找到“Kinco Electric(Shenzhen) Ltd”，选择其目录下的“Kinco RP2D-1608C1”设备，点击“添加”，如若已经连接好所需要的硬件设备，可以在上位机与控制器连接成功后，登录并进行“扫描”，可以对已经连接的设备进行“一键添加”组态。

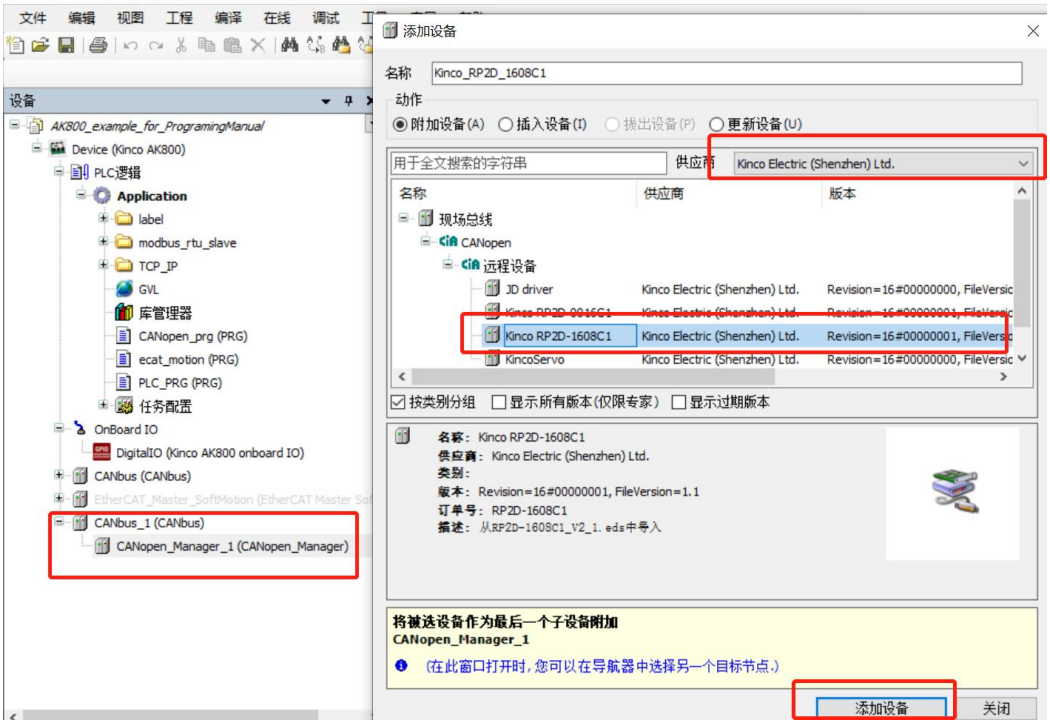


图 11.4-7：添加远程连接设备

（3）从站通用配置

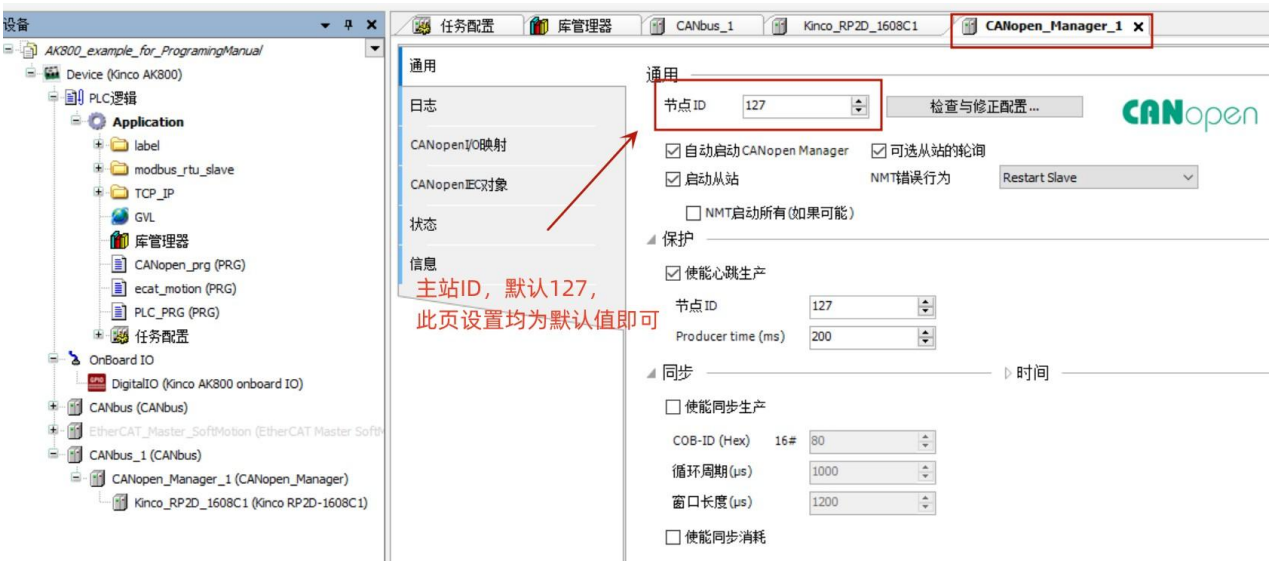


图 11.4-8：“CANopen Manager”通用配置

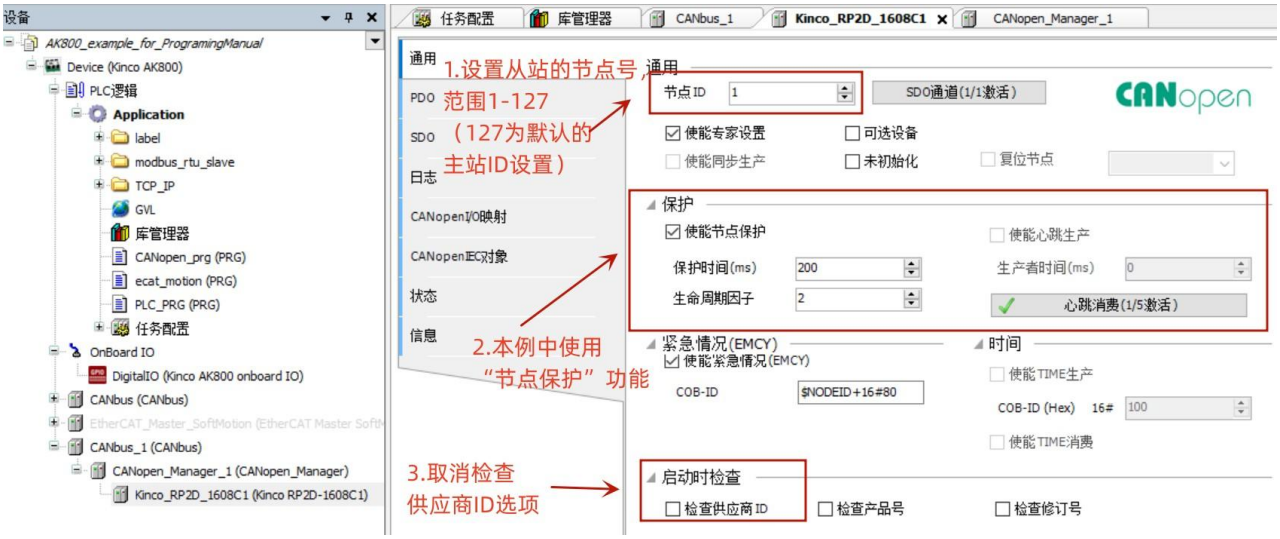


图 11.4-9：“Kinco_RP2D_1608C1”从站通用配置

（4）“Kinco_RP2D_1608C1”PDO 配置：如下图 RP2D 设备的 PDO 遵循默认配置即可。

图 11.4-10：RP2D 设备默认的 PDO 配置

（5）查看配置：PDO 配置好后，系统默认为配置的条目分配地址，在“CANopen I/O Mapping（映射）”中可查。

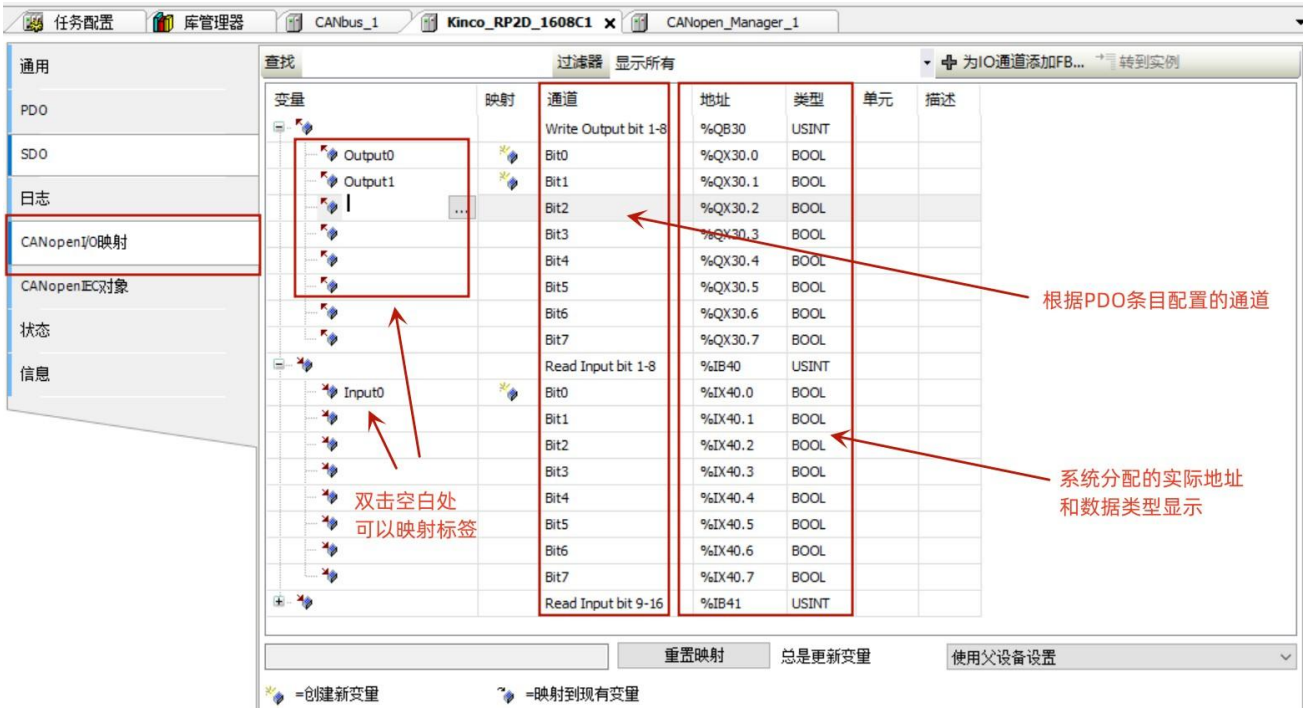


图 11.4-11：IO 配置映射界面

（6）启动参数 SDO：

在设备配置信息的“SDO”选项卡下，可以配置启动参数 SDO，此 SDO 配置仅在控制器启动后改写一次，此后不再动作。添加的方法如下图所示：如图，添加一个写入控制字（ControlWord：16#6040）为 6 的 SDO 启动参数。

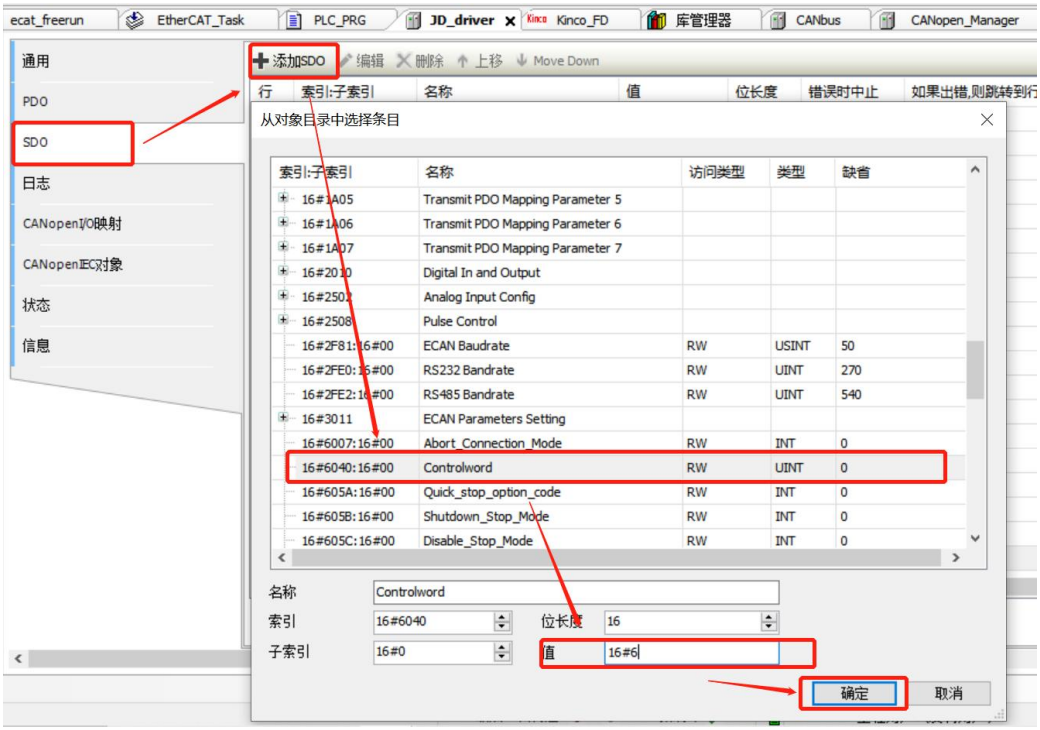


图 11.4-14：启动参数 SDO 配置步骤

11.4.3 PR2D 硬件配置及接线

1.节点 ID 设置

表 11.4-1：RP2D 节点 ID 拨码

DIP1 Bit0	DIP2 Bit1	DIP3 Bit2	DIP4 Bit3	DIP5 Bit4	DIP6 Bit5	DIP7 Bit6	Node ID
ON	OFF	OFF	OFF	OFF	OFF	OFF	1
OFF	ON	OFF	OFF	OFF	OFF	OFF	2
ON	ON	OFF	OFF	OFF	OFF	OFF	3
...	...	...	...	...	...	...	...
OFF	ON	ON	ON	ON	ON	ON	126
ON	ON	ON	ON	ON	ON	ON	127

注：每次设置完节点 ID 与波特率后需断电重启模块。

2.波特率设置

表 11.4-2：RP2D 波特率拨码

DIP8 Bit0	DIP9 Bit2	DIP10 Bit4	Baud rate
OFF	OFF	OFF	10K
ON	OFF	OFF	20K
OFF	ON	OFF	50K
ON	ON	OFF	125K
OFF	OFF	ON	250K
ON	OFF	ON	500K
OFF	ON	ON	800K
ON	ON	ON	1M

注：每次设置完节点 ID 与波特率后需断电重启模块。

3.硬件接线

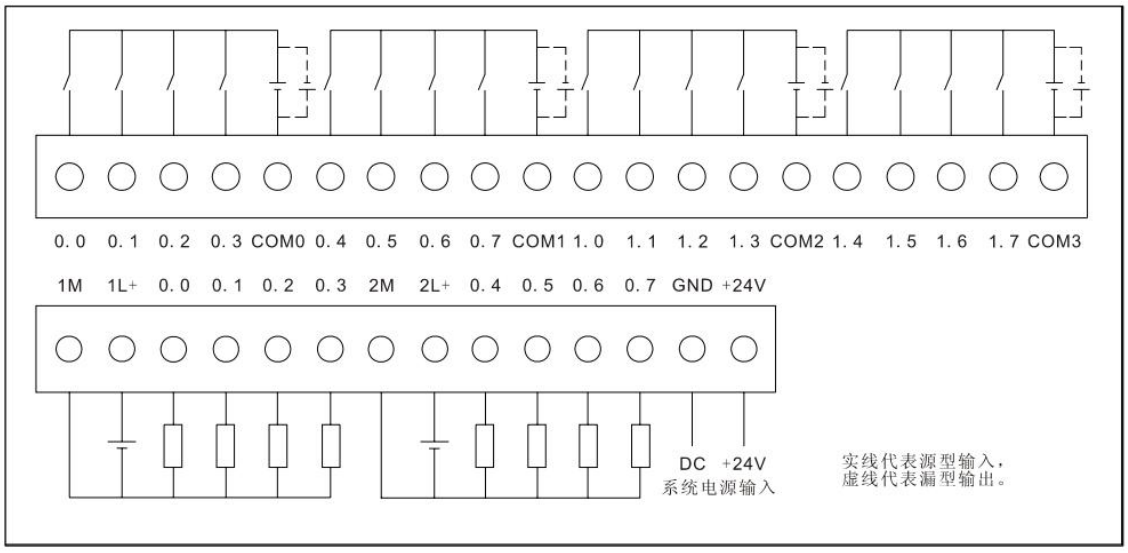


图 11.4-12：IO 硬件接线图

11.5 AK800 通过 EtherCAT 总线控制 Kinco MD 系列集成式低压伺服电机

11.5.1 EtherCAT 通讯连接

AK800 控制器的 EtherCAT 通讯口直接通过网线与 Kinco 伺服驱动器的 X1A(IN)口相连（建议超五类屏蔽双绞线）。

11.5.2 EtherCAT Master 配置

（1）添加“EtherCAT Master SoftMotion”设备标识符

右击“Device”，选择“添加设备”，在弹出的设备窗口中，根据下图路径选择 EtherCAT 目录下的“EtherCAT Master”，

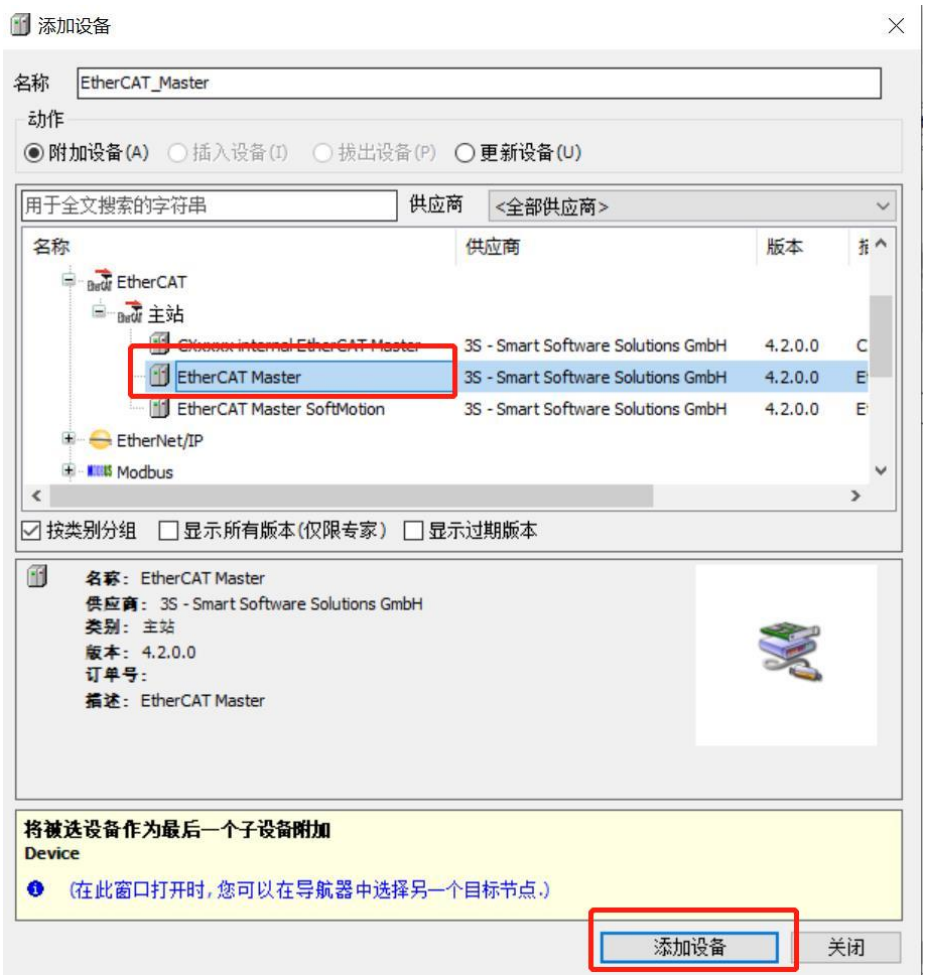


图 11.5-1：添加“EtherCAT Master ”设备标识符

（2）配置“EtherCAT Master ”选项

添加了“EtherCAT Master”设备标识符后，系统会自动添加“EtherCAT_Task”任务，需要将于 EherCAT 有关的 POU 都放到此任务下。本例中使用任务周期为 4ms。

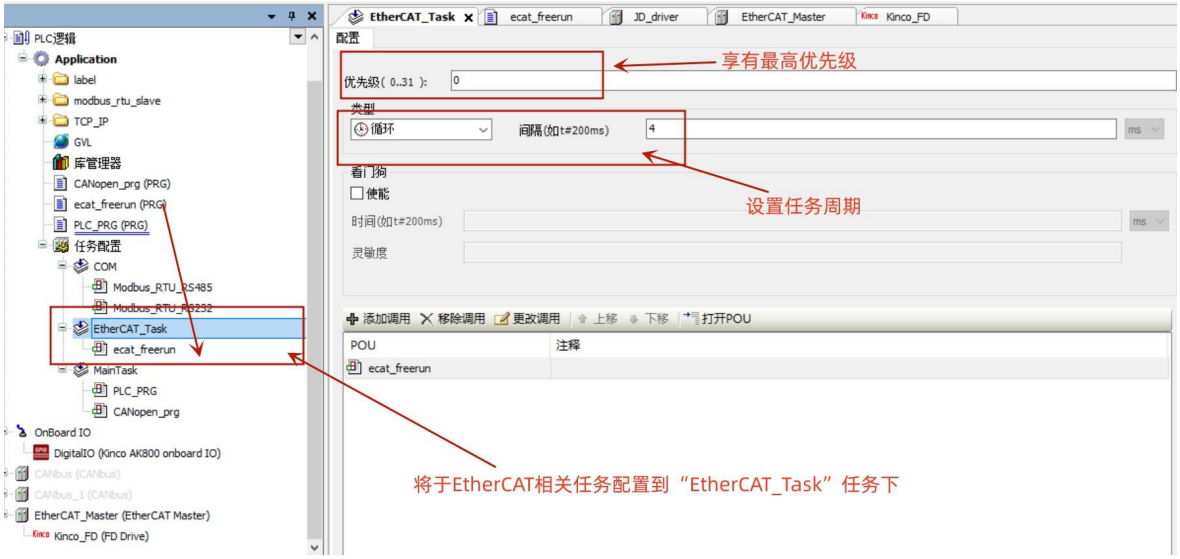


图 11.4-2：“EtherCAT Master SoftMotion”配置 a

打开“EtherCAT Master”通用设置页面，点击“浏览”，选择 EtherCAT 通道的网卡为“eth1”（AK800 中，“eth0”为 Ethernet）。

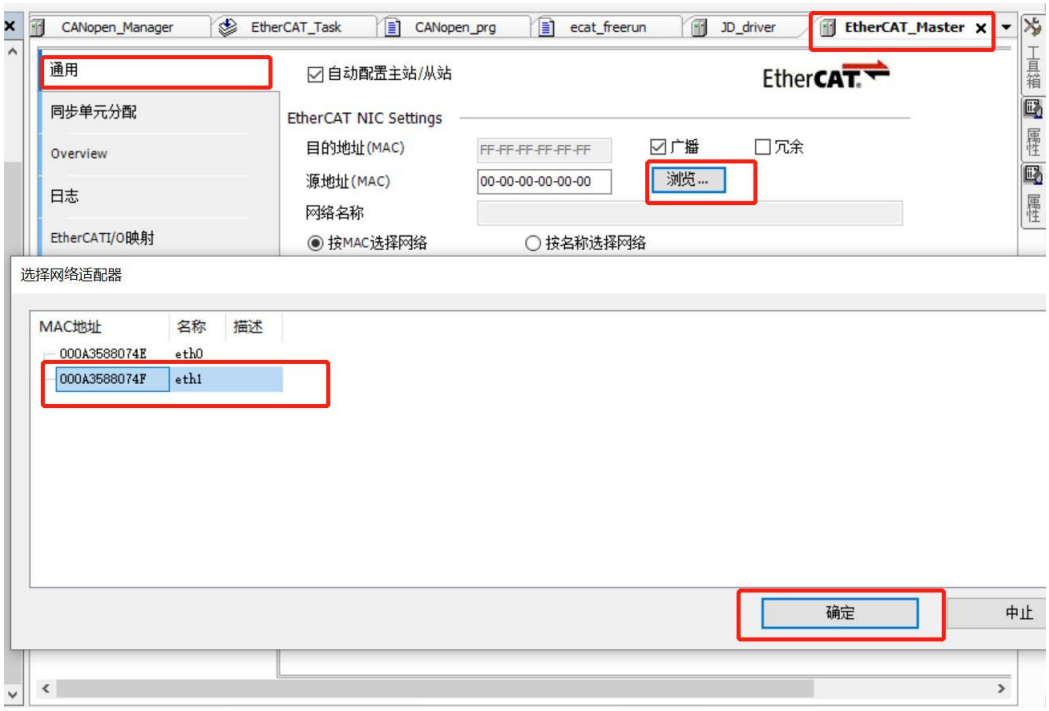


图 11.5-3：“EtherCAT Master”配置 b

（3）添加 EtherCAT 驱动设备

右击“EtherCAT Master”，在 Kinco 目录下找到“FD Drive”，选中并添加（也可以登录后进行扫描添加，请确保已安装设备描述文件（*.xml），相关文件，可从 Kinco 步科官方网站下载：kinco.cn）。

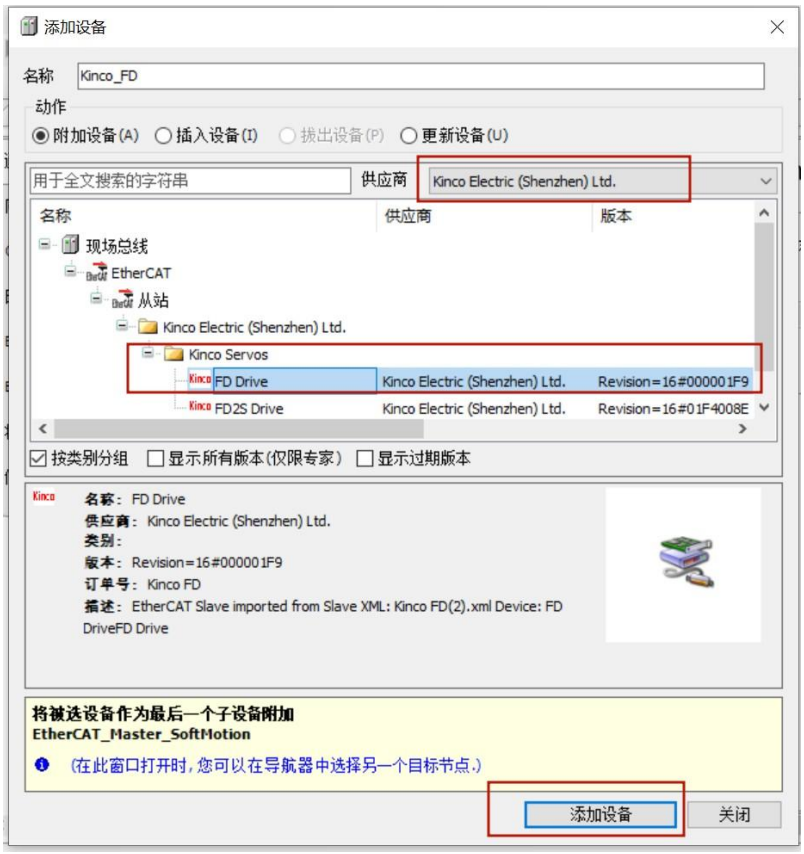


图 11.5-4：添加 EtherCAT 驱动设备

图 11.5-5：EtherCAT 设备通用配置

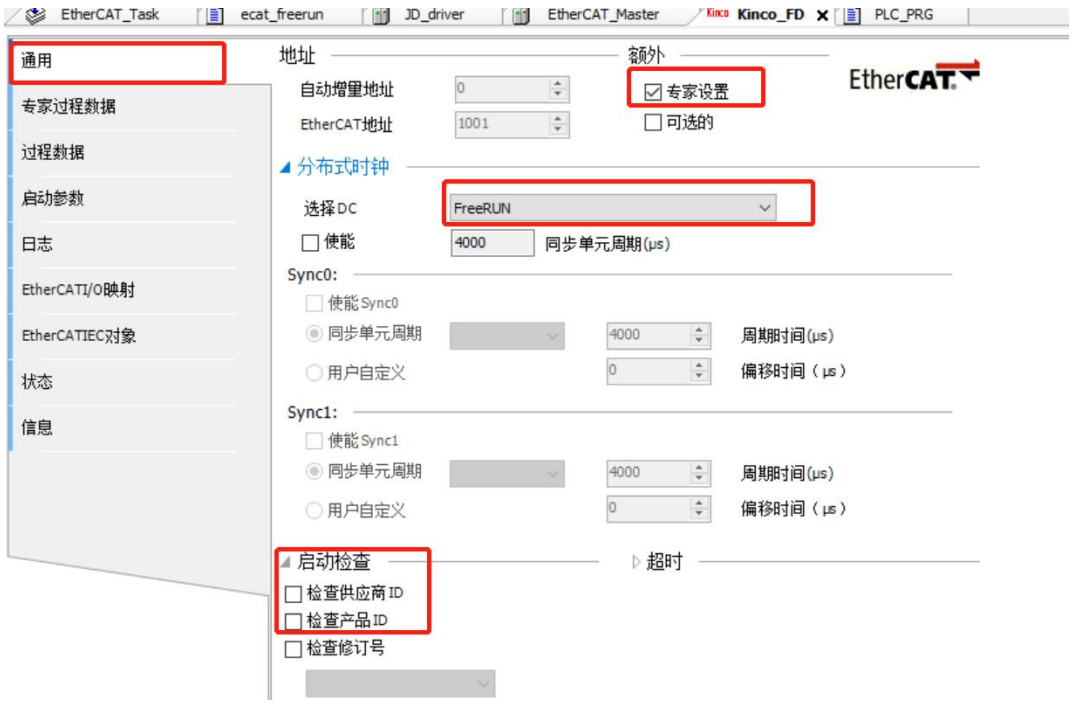


图 11.5-6：专家过程数据页

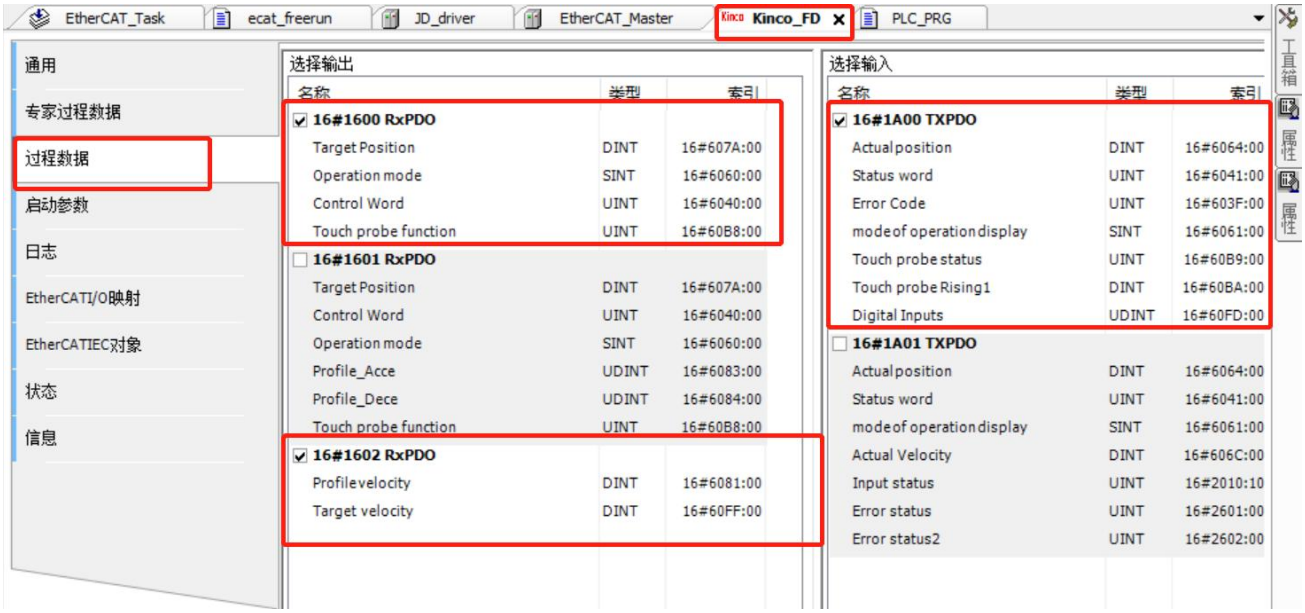


图 11.5-7：过程数据页

如图 11.5-6 和图 11.5-7，显示了设备描述文件所描述的从站输入和输出过程数据，这些变量用于将来与 PLC 的实时通讯。本例的最终过程数据输出变量为 16#1600 何 16#1602，输出为 16#1A00 中所配置的条目。

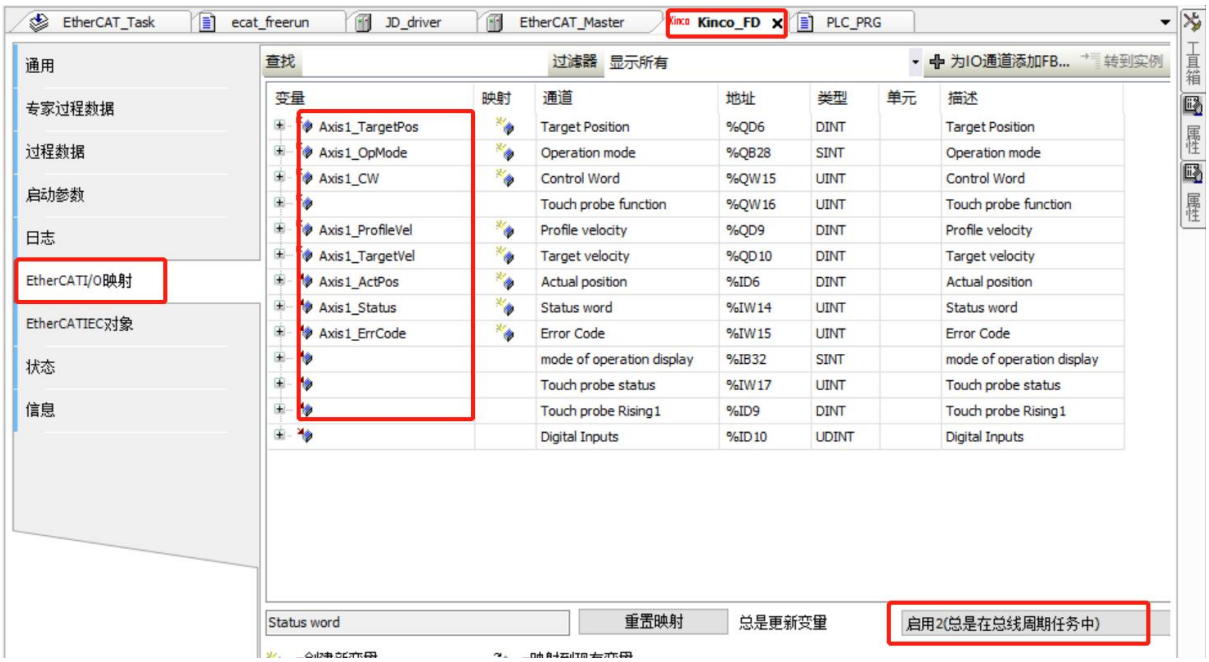


图 11.5-8：EtherCAT 变量映射

EtherCAT 通讯配置完成完成后，最终可以在“EtherCAT I/O 映射”中查看变量地址、类型及实施的状态，为方便后续程序编写，本例中将变量映射为标签，其显示界面如图 11.5-8 所示。

此外，还可以为设备配置 SDO 形式写入的启动参数，启动参数只会在控制器启动后执行一次，此后不再动作。配置方法如下图所示：下图配置了一个启动后写入同步周期（）为 2ms 的 SDO。

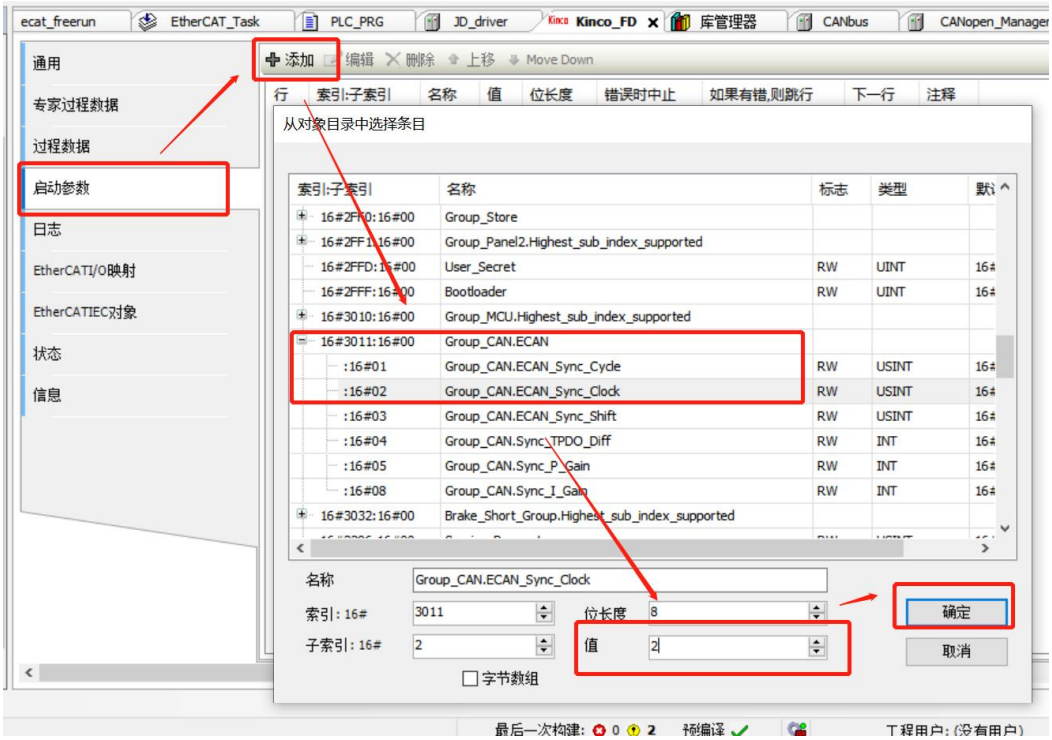


图 11.5-9：启动参数配置

11.5.3 Kinco 驱动器设置（KincoServo+）

（1）伺服从站的默认数字量输入配置要清空（I/O 优先原则）。

图 11.5-9：清除伺服驱动的数字量输入默认配置

（2）确保关闭“时钟同步模式”

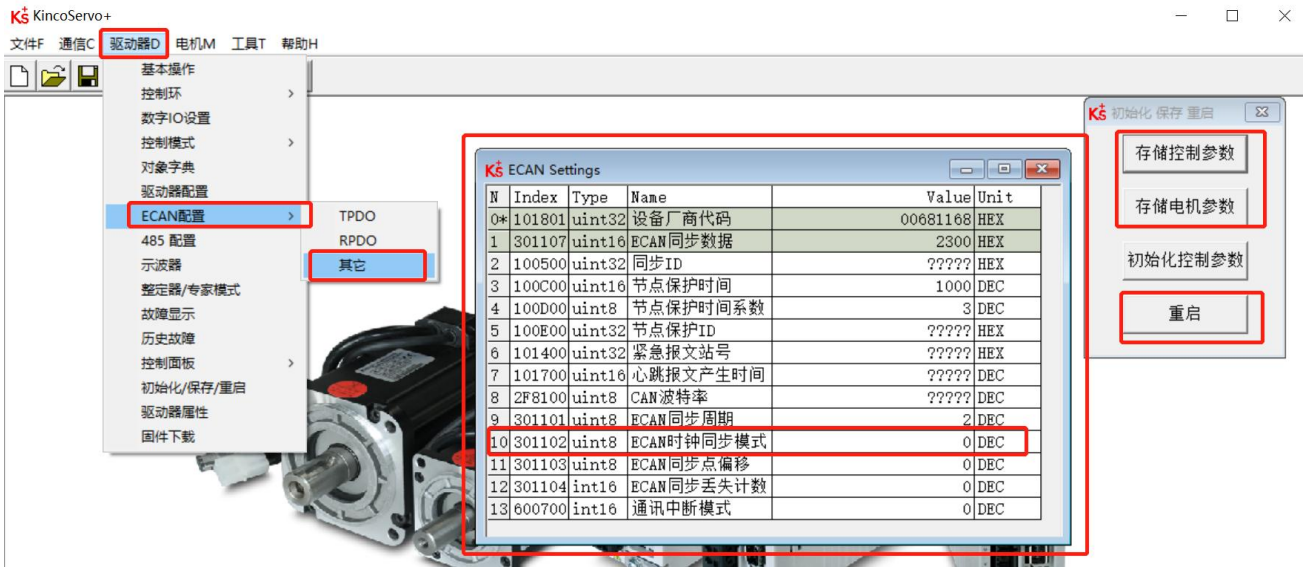


图 11.5-10：伺服驱动的 ECAN 参数设置

右击“基本操作”界面，选择添加，在弹出的对象字典中搜索（查找）上图参数（301102），将其设为 0，并“存储到控制器”及重启。

11.5.4 编写简易的 EtherCAT FreeRun 程序

1.本例配置如下：

- 单轴，写入时，速度为 10rpm=27307DEC；
- AK800 通过 FreeRun 模式与驱动连接；
- 过程数据配置按下图默认配置：

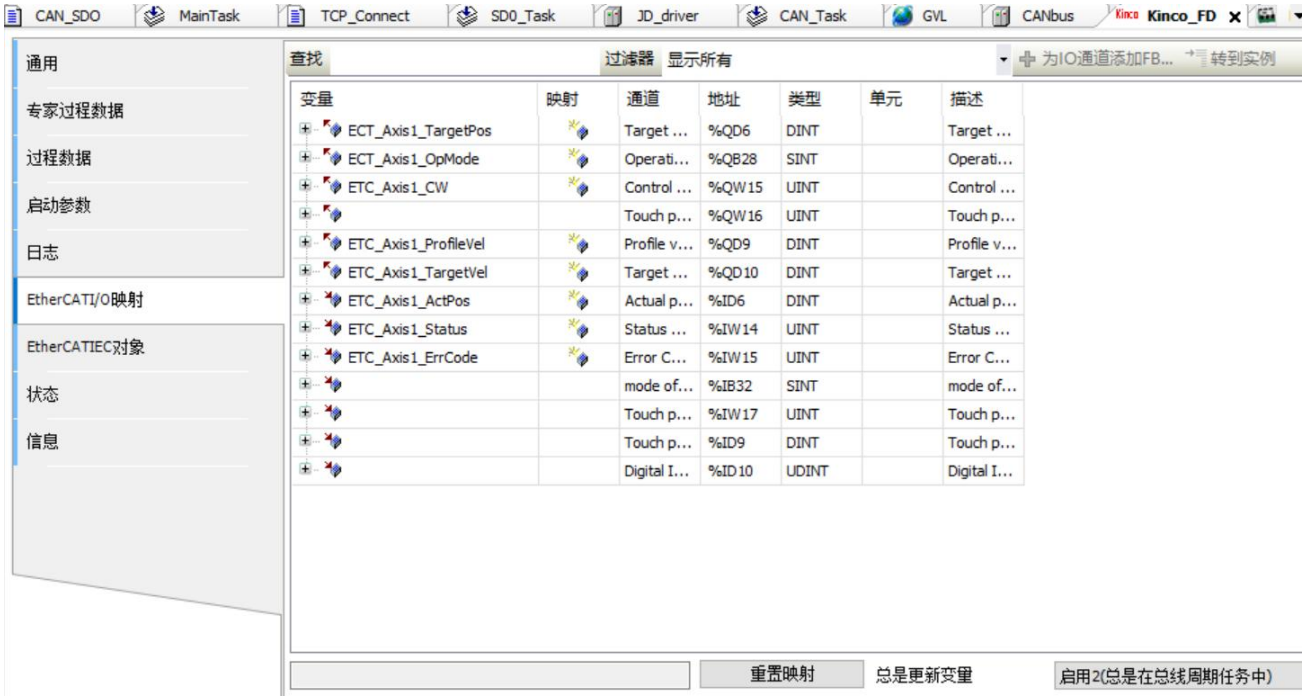


图 11.5-11：本例中的伺服过程数据配置

2.程序编写

```
PROGRAM ecat_freerun
VAR
    vBootup_R    : Standard.R_TRIG;
    bBootup      : BOOL;
    vJog_R       : Standard.R_TRIG;
    bJog         : BOOL;
    diJog_Vel    : DINT:=27307;//M 系一体机：Ethercat FreeRun 模式时，10rpm=27307;
    bstp         : BOOL;
    diActPos1    : DINT;
END_VAR
```

```
vBootup_R(CLK:=EtherCAT_Master.xConfigFinished,Q=>);
IF vBootup_R.Q THEN
    ETC_Axis1_CW:=6;//为驱动的 controlword 赋初值：6
END_IF

//速度模式正反转：反转时速度给负值
vJog_R(CLK:=bJog, Q=>);
IF vJog_R.Q THEN
    ECT_Axis1_OpMode:=3;//写入速度模式：3
    ETC_Axis1_TargetVel:= diJog_Vel;//速度值
    ETC_Axis1_CW:=16#F;//控制字：锁轴
    bJog:=FALSE;
END_IF

//停止
IF bStp THEN
    ECT_Axis1_OpMode:=1;//
    ETC_Axis1_TargetVel:= 0;//速度值
    ETC_Axis1_CW:=16#6;//控制字：松轴
    bstp:=FALSE;
END_IF
```

3.登录下载并运行程序，尝试置位变量“bJog”和“bStp”，电机将以 10rpm 正转运行或停止运行。

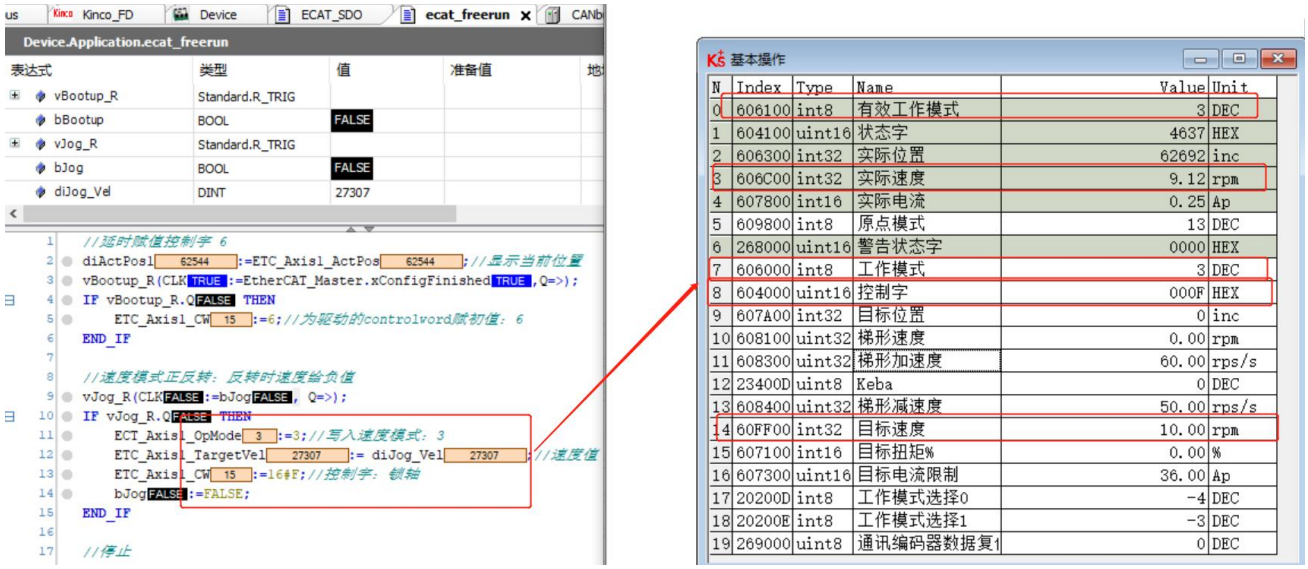


图 11.5-12：程序运行状态

11.5.5 EtherCAT COE 读写示例程序

下面通过简单的 COE_SDO 读写，以 AK800 示范“EtherCATStackLibrary”下的“ETC_CO_SdoRead4”及“ETC_CO_SdoWrite4”的用法。程序要求：通过写 COE_SDO 修改原点模式；在伺服驱动运行状态下通过 COE_SDO 读取当前的有效工作模式。具体程序如下：

```
PROGRAM ECAT_SDO
VAR
    COE_Read1        :ETC_CO_SdoRead4;
    COE_Write1       :ETC_CO_SdoWrite4;
    bExecute_r       : BOOL;
    bExecute_w       : BOOL;
    byAr_COE_rDtata  : ARRAY [1..4] OF BYTE;
    byAr_COE_WDtata: ARRAY [1..4] OF BYTE;
    usiDATA_rLength  : USINT;
    usiwDATA_wLength: USINT;
    bDone_r          : BOOL;
    bError_r         : BOOL;;
    vERROR_r         : ETC_CO_ERROR;
    bDone_w          : BOOL;
    bError_w         : BOOL;
    vError_w         : ETC_CO_ERROR;
END_VAR
```

```
//COE 读取当前有效工作模式 :606100
COE_Read1(
        xExecute:= bExecute_r,
        xAbort:= ,
        usiCom:= 1,
        uiDevice:= 1001,
        usiChannel:= ,
        wIndex:= 16#6061,
        bySubindex:=16#00 ,
        udiTimeOut:= 300,
        xDone=> bDone_r,
        xBusy=> ,
        xError=> bError_r,
        eError=> vERROR_r,
        udiSdoAbort=> ,
        abyData=> byAr_COE_rDtata,
        usiDataLength=> usiDATA_rLength);

//COE 写入原点模式：16#609800=13
COE_Write1(
        xExecute:= bExecute_w,
        xAbort:= ,
        usiCom:= 1,
        uiDevice:= 1001,
        usiChannel:= ,
        wIndex:= 16#6098,
        bySubindex:= 16#00,
        udiTimeOut:= 300,
        abyData:= byAr_COE_WDtata,
        usiDataLength:= usiwDATA_wLength,
        xDone=> bDone_w,
        xBusy=> ,
        xError=> bError_w,
        eError=> vError_w,
        udiSdoAbort=> );
```

程序运行效果如下图 11.5-13 和 11.5-14：

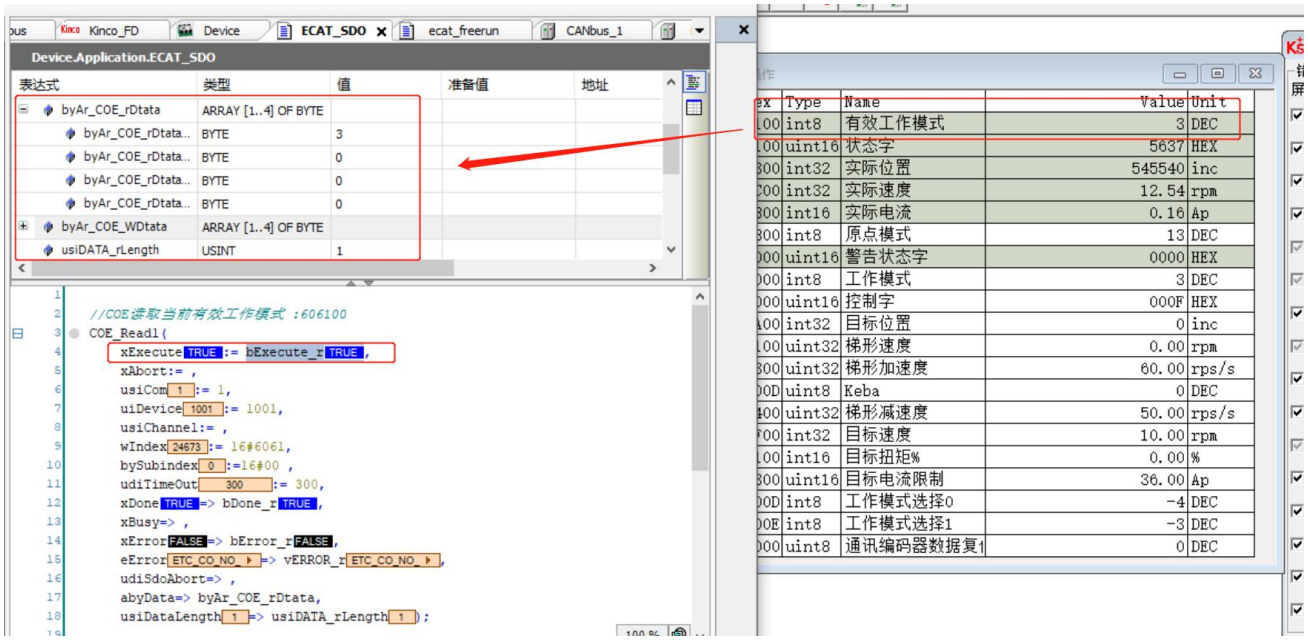


图 11.5-13：COE 示例读取

如上图，将“bExecute_r”赋值 TRUE，执行 SDO 写入功能块，可在伺服软件（Kinco Servo+）处实时观测到有效工作模式（16#606100）为 3，改参数长度为 1 BYTE。

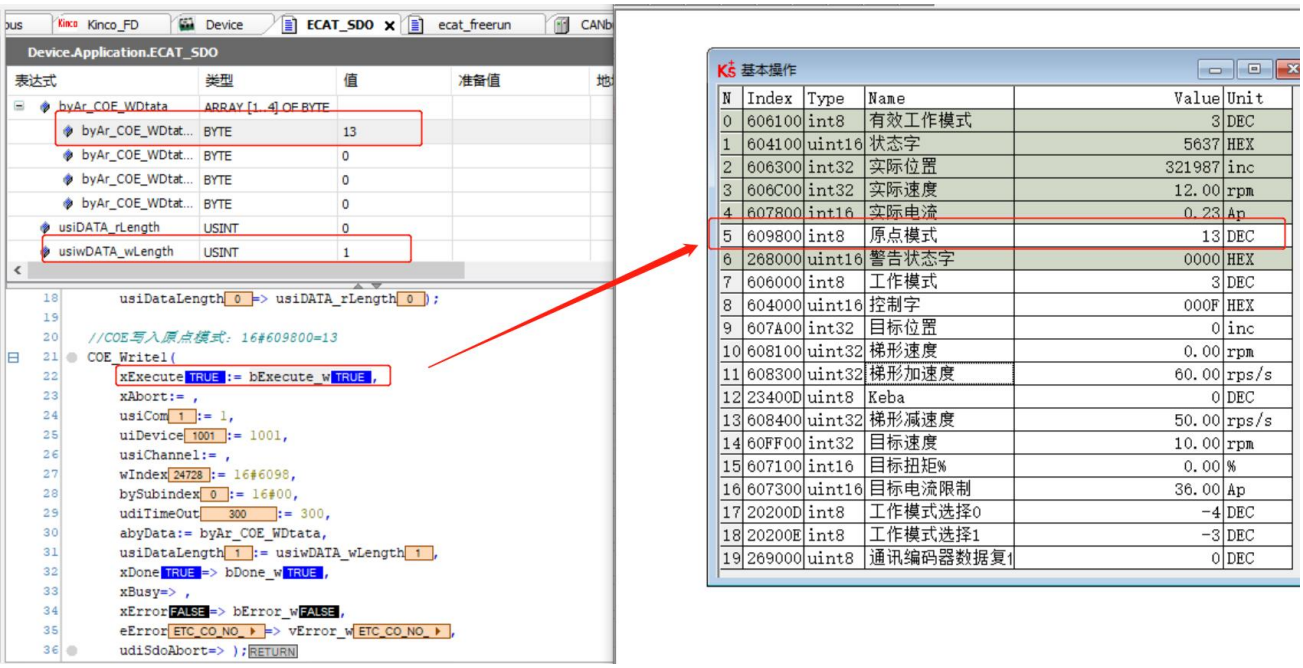


图 11.5-14：COE 示例写入

如上图，将“byAr_COE_WDtata[1]”赋值 13，代表要写入的 13 号原点模式，为“usiwDATA_wLength”赋数据长度为 1 (BYTE)，之后执行“bExecute_w”，执行 COE_SDO 写入功能块，可在伺服软件（Kinco Servo+）处实时观测到原点模式（16#609800）被修改为 13。

11.6 AK800 通过 CoDeSys 更新固件驱动

控制器 AK800 通过 CoDeSys 更新固件，更新前请确保上位机通过 EtherNet 正确连接 AK800 控制器，具体方法如下图所示：

图 11.6-1：AK800 通过 CoDeSys 更新固件驱动过程

如上图，在确保设备连接的情况下，将新的固件文件（*.fw）放置到控制器系统中后，执行上电重启操作，控制器会自动重新加载系统，等待系统升级完成后，可以在确保上位机与控制器正常连接的情况下，进入“Device”设置页面的“PLC 指令”操作页，输入指令：productinfo，然后点击回车，即可查询当前控制器的软件版本号，如下图所示。

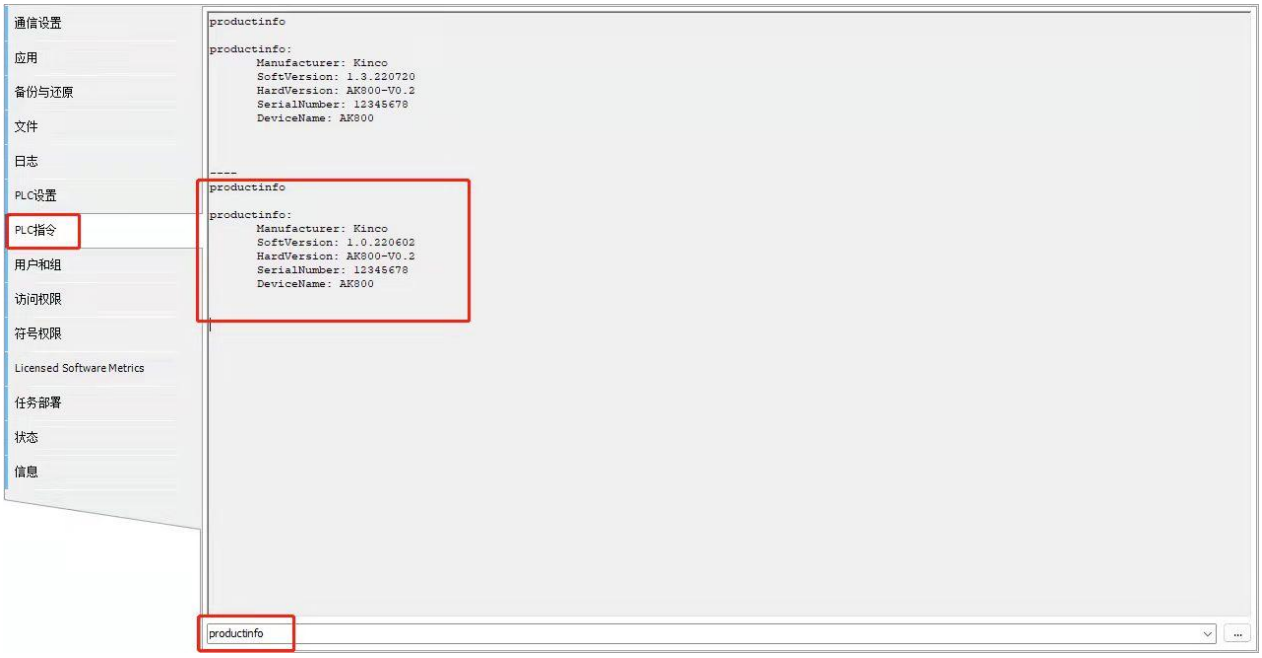


图 11.6-2：通过 PLC 指令查询控制器软件版本号