Kinco

Kinco-K 系列

可编程控制器

应用手册

前言

目录

上海步科自动化股份有限公司

基本说明

- 感谢您购买了 Kinco-K 系列可编程序控制器。
- 本手册主要介绍 Kinco-K 系列可编程序控制器的编程软件、指令应用等内容。
- 在使用产品之前，请仔细阅读本手册，并在充分理解手册内容的前提下，进行编程。
- 硬件接线方面的介绍，请查阅相关硬件手册。
- 请将本手册交付给最终用户。

用户须知

- 手册等其他技术资料中所列举的示例仅供用户理解、参考用。
- 将该产品与其他产品组合使用的时候，请确认是否符合有关规格、原则等。
- 使用该产品时，请自行确认是否符合要求以及安全。
- 请自行设置后备及安全功能，以避免因本产品故障而可能引发的机器故障或损失。

责任申明

- 手册中的内容虽然已经过仔细的核对，但差错难免，我们不能保证完全一致。
- 我们会经常检查手册中的内容，并在后续版本中进行更正，欢迎提出宝贵意见。
- 手册中所介绍的内容，如有变动，请谅解不另行通知。

前　言

上海步科自动化股份有限公司是致力于国产高品质 PLC 开发、营销、生产、服务的高科技企业。公司的创业者凭借在自动化领域十几年的实践经验和对 PLC 产品的深刻理解，利用我们所掌握的 PLC 软/硬件核心技术，开发与国际同步的高品质小型 PLC，以 100%自有知识产权、优良的品质、体贴到位的服务，创建适合国内用户的小型一体化 PLC 民族品牌，打破进口品牌垄断中国市场的局面，为民族自动化产业贡献一份力量。

自 2011 年以来，上海步科自动化股份有限公司推出了 Kinco-K 系列小型 PLC，包括 K5、K6、K2、KS、KW 等产品。其中 K5、K6、KS 属于标准型产品，具有功能丰富、高性能、高可靠性、扩展性良好等特点。K2 属于经济单机型产品，在保证功能、性能、可靠性的前提下，优化硬件设计以降低成本，更提供了 USB 编程口、晶体管型 DIO 点（DI、DO 复用）等更贴近用户需求的功能，具有很高的性价比。KW 是步科公司推出的为智慧工厂打造的无线产品，在延续 K 系列功能丰富、高性能、高可靠性的前提下，采用了性能更高的 CPU，更提供了本体自带无线网络接口、MicroUSB 编程，更高性能的高速输入/输出、紧凑型安装等更贴近用户需求的功能，能满足用户多种应用需求。自推向市场以来，K 系列 PLC 已经应用到环保机械、木工机械、纺织机械、建材机械、食品机械、中央空调以及小规模过程控制等领域，以其优良的性能价格比得到用户的一致认可。

为了方便用户的使用，我们编写了此手册，对 K 系列 PLC 的编程软件 KincoBuilder 进行了系统介绍。本手册内容详尽，详细描述了编程的基本概念、KincoBuilder 软件的界面功能、操作方法和指令集，其中穿插了大量的示例。另外，本手册对一些 IEC61131-3 的基本概念也进行了介绍，以照顾广大初学 PLC 的用户。

书中有疏漏不足之处，敬请广大用户指正，不胜感激。

公司地址：深圳市南山区高新科技园北区朗山一路 6 号意中利工业园 1 号厂房 1-3 层

邮　　编：518057

电　　话：0755-26585555

传　　真：0755-26616372

电子邮箱：sales@kinco.cn

网　　址：www.kinco.cn

目录

第一章 欢迎使用 KincoBuilder

1.1 KincoBuilder 简介

KincoBuilder 是步科公司 K 系列 PLC 的上位编程软件，编程环境符合 IEC61131-3 标准，是一套功能强大、使用方便、高效的开发系统。

IEC61131-3 是 IEC 为工业自动化编程制定的标准，是在吸收不同厂家编程语言风格、方言及适应未来软件技术发展要求制定的，独立于任何一家公司，适合不同领域、不同类编程人员习惯。自发布以来得到所有顶尖 PLC 厂家的认可，各厂家的编程软件也在尽量向 IEC 标准靠拢。KincoBuilder 完全是自主研发，采用了符合 IEC61131-3 标准的方案，由于许多用户通过各种渠道已经掌握了大部分编程技巧，这就使得 KincoBuilder 简便、易学，使用起来将会得心应手。

KincoBuilder 软件具有如下特点：

- 符合 IEC61131-3 标准，支持 IL（指令表）和 LD（梯形图）两种标准语言
- 丰富的指令集，内置 IEC61131-3 定义的标准功能、功能块以及一些特殊的应用指令
- 支持结构化的编程方式，用户的程序被组织成“工程”
- 支持中断服务程序，支持用户自定义功能块（子程序）
- 允许在程序中使用变量名称，便于工程的实施、维护
- 灵活的硬件配置方式，最大限度地允许用户自定义各种硬件参数
- 完善的联机功能，包括下载、上载、在线监视、强制、读写实时时钟等
- 定义了完善的快捷键、右键菜单，方便用户的使用
- 对用户的错误操作尽可能地予以屏蔽、提示，体贴用户的操作

1.2 本书常用名词术语

- 小型一体化可编程控制器

按国际通用分类原则，小型 PLC 一般指实际控制点数小于 128 点（并非设计控制点数）的 PLC，此类 PLC 通常采用一体化结构，即在 CPU 模块上集成一定数量的 I/O 以及输出电源、高速输入/输出等其他附件。

- CPU 本体

即 CPU 模块，是控制系统的核心。用户编写的应用程序经编程软件下载后存储于 CPU 模块中的永久性存储器中，在 CPU 运行软件的调度下执行用户程序的控制功能，运行软件同时还负责诊断各模块的工作状态和用户程序的执行错误。

- 扩展模块与扩展总线

扩展模块是用于扩展 CPU 本体功能的模块，分为扩展 I/O 模块（增加系统的输入/输出通道）和扩展功能模块（扩展 CPU 功能）。

扩展总线连接 CPU 模块和扩展模块的数据和信号通道，物理介质 K5 系列采用了 16 芯扁平电缆，其他系列采用普通直连网线。在扩展总线中集成有数据总线、地址总线和扩展模块工作电源。

- KincoBuilder

Kinco-K 系列 PLC 的编程软件，符合 IEC61131-3 标准，目前支持 LD 和 IL 两种标准语言。利用 KincoBuilder 可以对 Kinco-K 系列 PLC 实现编程、调试等各种功能。

- CPU 运行软件

又称 CPU 板级固件（firmware），存储于 CPU 本体的 Flash 存储器中，在 CPU 模块上电同时开始运行，管理、调度 CPU 的所有任务。用户编写的应用程序是在 CPU 运行软件的调度之下执行的。运行软件同时还负责诊断各模块的工作状态和用户程序的执行错误。

- 应用程序

又称用户程序、用户工程，是用户编写的完成特定控制功能的程序。应用程序下载后存储于 CPU 模块的永久性存储器中，CPU 上电运行时被读入 RAM 中执行。

- 主程序与程序扫描

在 CPU 中各种任务是被循环地连续执行的，这种循环执行任务的过程称为扫描。

主程序是应用程序的执行入口。在每个扫描周期内主程序都会被执行一次。应用程序中只能有唯一的主程序，但可以从主程序中调用多个子程序。

- I/O 映像区

物理 I/O 点的状态数据在 CPU 中的存储区域，分为输入映像区和输出映像区。

为保证在 CPU 一次扫描中数据的一致性及提高程序执行速度，在每次扫描中 CPU 首先将物理输入点的状态读入到输入映像区，应用程序执行过程中只对 I/O 映像区进行访问，在扫描结束时再将输出映像区中的数据发送到物理输出通道。本文中提到的 I/O 地址，如 I0.0、Q1.0、AIW0、AQW0 等，均是 I/O 映像区中的地址。

- 数据保持与数据备份

数据保持是指在 CPU 断电后 RAM 中的数据保持为断电瞬间的状态并供 CPU 在下一次上电的时候使用。数据保持靠一次性电池，在常温下，保持时间约为 3 年，用户可以自行更换电池。

数据备份是指用户通过指令将数据写往永久存储器中进行保存。注意：永久性存储器均有使用寿命，E^2PROM 允许写 100 万次，FRAM 允许写 100 亿次。

第二章 使用 KincoBuilder 快速入门

本章对 KincoBuilder 的安装以及运行环境进行了详细的描述。另外，也描述了如何连接 PLC 与计算机。最后，举例说明了开发一个工程的常见步骤。这一章的目的只是帮助用户快速入门，关于 KincoBuilder 各种功能的详细描述请参见后续章节。

2.1 系统需求

2.1.1 硬件需求

- CPU 主频 1GHz 或更高，硬盘 20M 以上空间，内存 512M 以上
- 键盘、鼠标、串行通信口(com 口)或者 USB 口(某些系列需另外使用 USB/RS232 或者 USB/RS485 转换器)
- 256 色或更高，分辨率 1024*768 或更高

2.1.2 软件需求

Windows XP(32bit), Windows Vista(32bit), Windows7(32/64bit), Windows8(32/64bit), Windows 8.1(32/64bit), Windows10(32/64bit)。

有些用户在 Windows7 或更高版本的操作系统下运行 Kincobuilder 可能会遇到问题，下面列出了常见的问题和解决办法。

- 【PC 机通信设置】对话框中的【COM 口】列表为空

  KincoBuilder 通过读取 Windows 注册表中的硬件信息来获取本机可用的 COM 口。

  在 Windows 系统某些版本中，用户必须以管理员身份来运行 KincoBuilder，否则 KincoBuilder 会因没有足够权限而无法访问注册表，也无法获取本机具有的 COM 口，此时将在【COM 口】列表中直接显示 COM1-COM255，用户可以自己从 windows 设备管理器中查看本机具有 COM 口号，并在列表中直接选择使用。

- 在某些计算机上无法运行 KincoBuilder

  请尝试设置以兼容 Windows XP 的模式来运行 KincoBuilder。设置方法如下：

  1) 在桌面或【开始】菜单中的“KincoBuilder”快捷方式中单击右键，然后在弹出菜单中执行【属性】命令。

  2) 在【属性】对话框中，进入【兼容性】页面进行设置，如图 2-1。

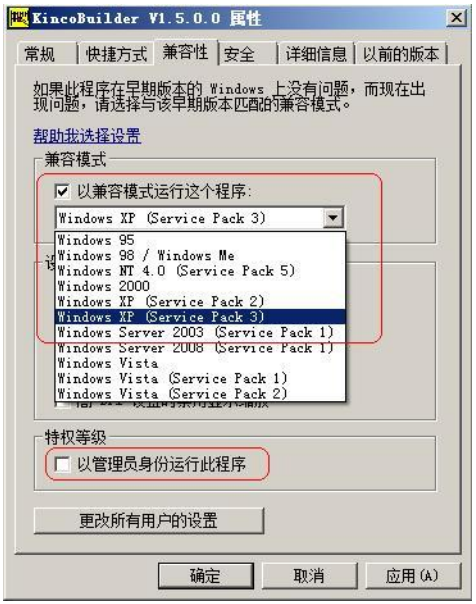


图 2-1Windows“兼容性”设置

图 2-2“以同步方式打开串口”

- 使用某些 USB 转 RS232 转换器时与 PLC 通信经常失败

这是由该转换器驱动程序的兼容性引起的。目前发现的几个失败事例以 64 位版本的 Win7 版本居多。

在 Kincobuilder 中，执行【工具】→【**软件设置…**】菜单命令，打开“软件设置”对话框，选中“以同步方式打开串口”，然后单击“确定”按钮退出即可。如图 2-2。

设置完成后，Kincobuilder 将以同步方式操作串口，在大多数情况下能够解决这个问题。

2.2 KincoBuilder 界面总体介绍

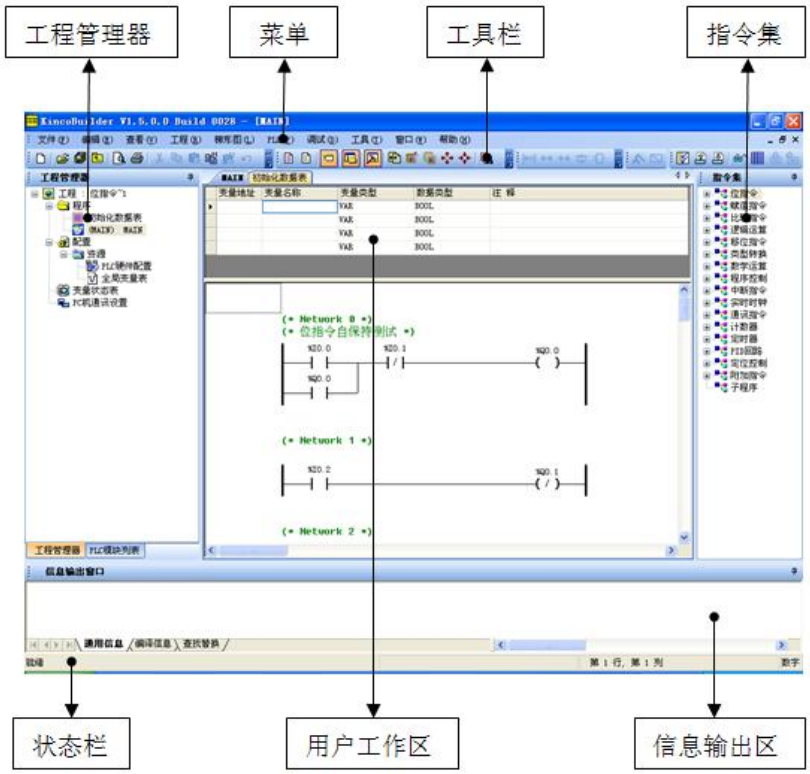


图 2-3　KincoBuilder 界面

- 菜单：菜单中包含了 KincoBuilder 软件所有的操作命令。
- 工具栏：工具栏中包含了用户使用频度较高的一些操作命令。
- 状态栏：状态条提供了软件当前的状态信息和操作命令的提示信息。
- 工程管理器：工程管理器是界面中的主要窗口之一，以树状列表的形式直观地显示出了当前工程的所有组成部分，包括程序、硬件配置、变量状态表、全局变量表等。用户可以在此窗口中对当前打开的工程进行管理、操作、维护。工程管理器的各个树节点均支持右键，用右键单击某个节点将会弹出相应的右键菜单来。

- 用户工作区：这是用户使用的主要区域，用户可以在此打开硬件配置窗口、全局变量表、编辑器等窗口，完成配置硬件、声明全局变量、编辑程序等任务。图 2-3 中显示的是编辑器，包括区域上部的变量定义表格和区域下部的程序编辑器（又分为 LD 编辑器、IL 编辑器）。
- 指令集：以树状列出了 Kinco-K 系列 PLC 支持的所有指令、功能、功能块和用户自己编写的子程序。指令集又分为 LD 指令集、IL 指令集。
- 信息输出区：用于显示 KincoBuilder 软件的各种提示信息。其中“编译信息”窗口显示了用户最近一次的编译信息，而“通用信息”窗口显示了最近一些操作的提示信息。

2.3 KincoBuilder 中应用程序的组织

2.3.1 工程的组织结构

在 KincoBuilder 中应用程序被组织成“工程（Project）”，工程中包含了用户程序、硬件配置等应用程序的所有信息。在本手册中，“用户程序”和“用户工程”这两个词的含义是相同的。

下表详细描述了工程的组织结构。表中注明“可选”的项表示此项并非工程的必要元素，用户在工程中可以忽略它们。

程序	初始化数据表（可选）	用户可在此为 V 区中 BYTE、WORD、DWORD、INT、DINT、REAL 类型的数据指定初始值。初始化数据表是工程中的可选部分，用户也可以不在表中输入任何内容。 在冷启动时 CPU 进入主循环之前，初始化数据表被处理一次。
	主程序	主程序是在 CPU 内运行的主循环任务，在 CPU 的每个扫描周期内都会被执行一次。 在工程中有且仅有 1 个主程序。
	中断服务程序（可选）	响应某个中断事件而被执行的程序称为中断服务程序。 中断服务程序不需要在主程序中调用。用户只需要通过 ATCH 指令将其与某个预定义的中断事件连接起来，那么当该中断事件发生时，CPU 就会自动调用该中断服务程序进行处理。

	子程序（可选）		子程序必须被主程序或者中断服务程序调用后才能被执行。 子程序是可重用的代码段，用户可以将常用的控制功能编写成一个个子程序。使用子程序可以更好地组织应用程序的结构，方便调试和维护，有效地缩短程序代码的长度。
配置	资源	硬件配置	用户在此对工程中用到的 PLC 模块及其参数进行配置。 CPU 将在冷启动时处理一次硬件配置，然后再执行其它任务。
		全局变量表（可选）	用户在此声明工程中需要的全局变量。

表 2-1 工程的组织结构

2.3.2 工程的存储目录

在建立工程时 KincoBuilder 软件将会要求用户输入工程的存放路径，然后就会生成一个空工程文件（扩展名为.kpr）并存放于此路径下。另外，在此路径下将会自动建立一个与工程名称相同的子目录，用于存放该工程所有的程序文件、变量文件以及其它的一些临时文件等。

例如，用户若选择在 c:\temp 目录下建立一个名为 project 的工程，则工程文件的路径是 c:\temp\project.kpr，其它文件存放在 c:\temp\project 目录中。

注意：工程应用时，工程文件（扩展名为.kpr）和相同名称的文件夹要同时存在，否则工程无效，若要保存工程要将两个同时备份才是一个完整的程序，缺一不可。工程备份也有快捷方法可以直接将工程备份为一个压缩文件，方便存储，避免遗忘文件造成严重的后果。具体导出方法，参照 2.3.3 导入工程和导出工程**。**

2.3.3 导入工程和导出工程

KincoBuilder 在【文件】菜单中提供了【导入工程…】和【导出工程…】命令以便于用户对工程进行备份以及管理。

- 【导出工程…】

  将当前打开的工程所涉及到的所有文件压缩至一个文件（扩展名.zip）中。在压缩文件中，

所有的文件名、相对路径以及工程的相关设置均保持不变。例如，工程名称为 myproj1，工程文件为 myproj1.kpr，则压缩文件中的工程文件仍旧为 myproj1.kpr，工程名称仍旧为 myproj1。

执行【导出工程…】后弹出“导出工程…”对话框，如下图所示。选择好路径并输入备份文件名后，单击【保存】按钮即可。

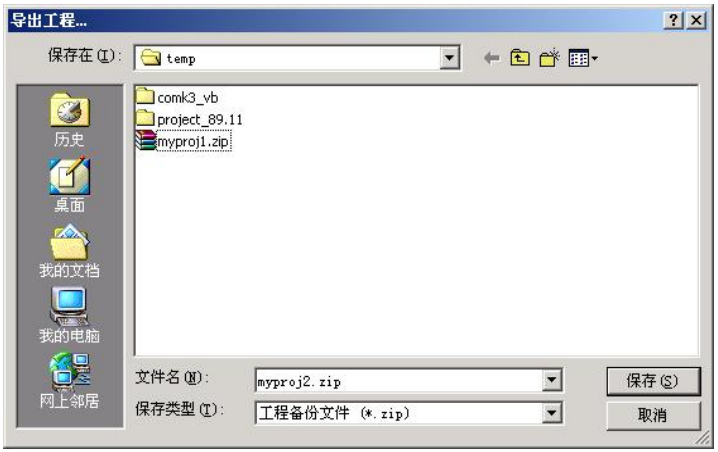


图 2-4 导出工程

- 【导入工程…】

  导入一个已存在的工程备份文件（扩展名.zip）并将其打开。

  执行该命令后，首先将弹出“导入工程…”对话框，如下图。

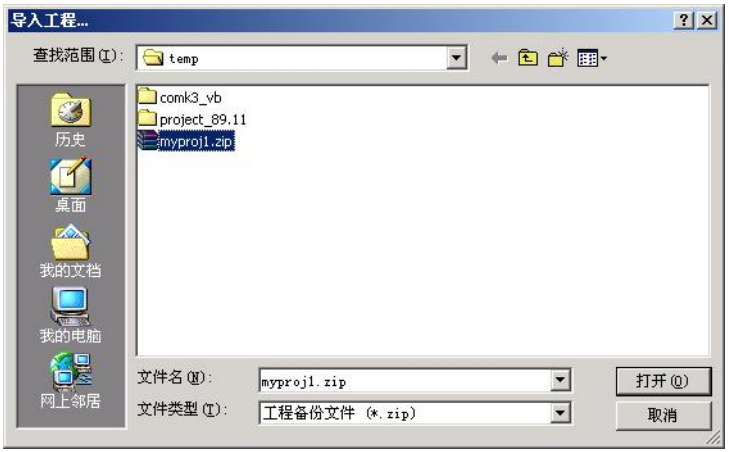


图 2-5 导入工程：选择文件

选择好备份文件后并单击【打开】按钮，将弹出如下对话框，选择解压之后工程文件的存放目录。选择好路径后，单击【确定】按钮即可将工程文件解压到选定的目录下并将其打开。

图 2-6 导入工程：选择目录

2.4 CPU 执行用户程序

CPU 循环地连续执行一系列任务的过程称为扫描。扫描是 CPU 的主要工作，通常也被称为主循环。在一个扫描周期内 CPU 将执行如下任务：

- CPU 自诊断
- 读输入：读取物理输入通道的信号数据并将其写至输入映像区
- 执行用户程序：直接执行用户工程中的主程序
- 处理通信请求
- 写输出：将输出映像区中的数据写至物理输出通道

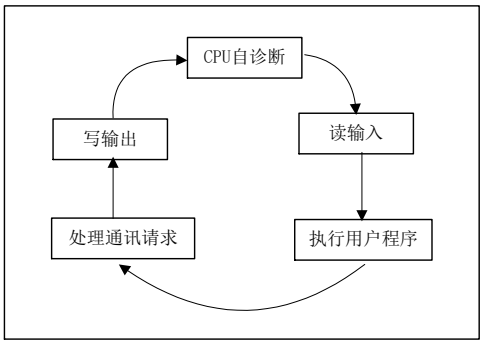


图 2-7 扫描周期

另外，需要注意的是，CPU 在扫描周期中执行的任务取决于它的工作状态。CPU 具有两种工作状态：运行（RUN）状态、停止（STOP）状态。这两种状态的主要差别就是：在 RUN 状态下 CPU 将执行用户程序，而在 STOP 状态下 CPU 不执行用户程序。

Kinco-K 系列 PLC 也支持中断，用户编写的中断服务程序将在指定连接（ATCH）的中断事件发生时执行。中断事件可能发生在扫描过程的任一时刻，这时候 CPU 将会先暂时中断主循环，转去执行中断服务程序，中断服务程序执行完成后，再从刚才的断点处重新进入主循环。如下图。

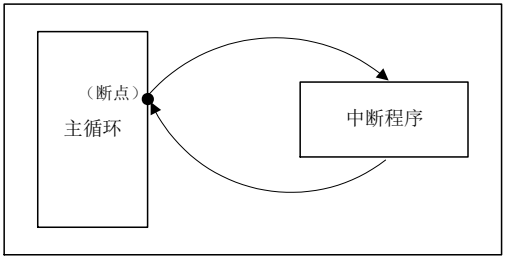


图 2-8 中断服务程序的执行

2.5 如何连接计算机与 PLC

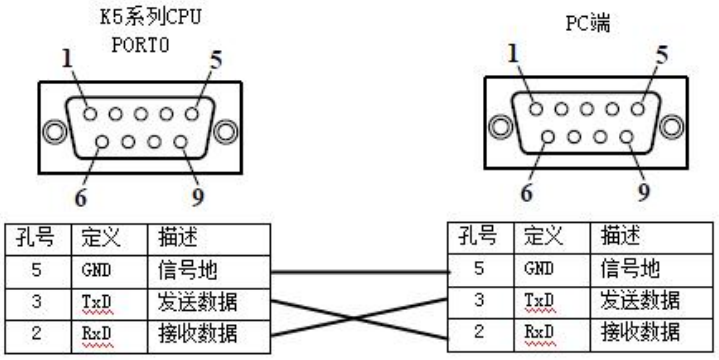
CPU 模块上集成了 USB 口、RS232、RS485 或者以太网通信口，用户可以使用这些通信口与其它设备进行通信。此处介绍的是在如何开始建立计算机与 CPU 之间的通信。

下表列出了各系列 CPU 支持编程协议的通信口。

产品系列	编程口				
	USB	PORT0（RS232）	PORT1（RS485）	PORT2（RS485）	Ethernet
K5 系列	/	√	√	/	/
K2 系列	√	仅 CPU209EA	√	√	仅 K204ET
KS 系列	仅 KS101M	√	√	/	仅 KS101M
KW 系列	√	√	√	/	/
HP 系列	√	/	√	/	/
MK 系列	√	/	√	√	/
K6 系列	/	/	√	√	√

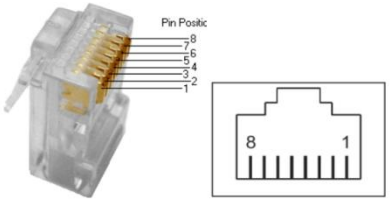
1) 使用适当的编程电缆连接好计算机和 CPU。

A：K5、KS、KW、K2（CPU209EA）系列 PLC 可以使用 PORT0（RS232）接口，通过如下编程电缆连接计算机的串行通信口（PC 端无 RS232 口的需另外使用 USB/RS232 转换器）。



KS/KW/K209 PORT0
RS232（位于RJ45接口）

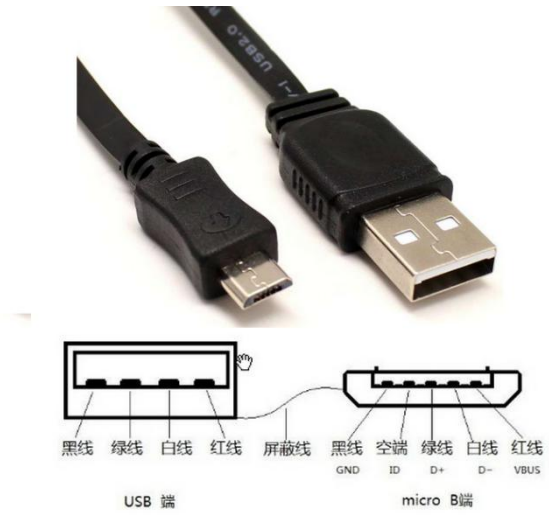
	管脚	描述
RS232	6	RXD
	3	TXD
	4	GND



⚠由于 RS232 的电气标准不支持带电插拔，因此在插拔电缆之前必须先断掉至少一方（CPU 或者计算机）的电，否则容易损坏通信口。

B：KS、KW、K2 系列 PLC 和 MK 系列 PLC 部分使用根据丝印上的 RS485 引脚制作编程电缆连接计算机的串行通信口(计算机的串行通信口需加上RS232转RS485的转换器)和CPU的PORT1口(RS485)相连。（PC 端无串行通信口的需另外使用 USB/RS485 转换器）

C：K2、KW、KS（仅 KS101M）系列 PLC 可以使用 Micro USB 编程电缆（常用作于安卓手机数据线）连接计算机。



在电脑上，USB 编程口作为一个虚拟串口，首次使用时必须先安装驱动程序。当安装完成最新版本的 Kincobuilder 编程软件后，驱动程序存放在 Kincobuilder 编程软件安装目录下面的 \Drivers 目录下面，目前支持 Windows XP、Windows 7、Windows 8 和 Windows 10 系统。如下图示例：

当第一次使用 USB 数据线连接 PLC 和电脑时，Windows 系统会自动检测到新硬件并提示安装驱动程序，此时用户根据自己的 Windows 版本选择相应目录下的驱动程序即可。

D：MK 系列一体机，PLC 和 HMI 共用同一个 USB 编程口，这种接口形式与 Kinco HMI 所用的一致，用户可以直接使用同样接口的数据线作为 PLC 的编程电缆。

在电脑上，PLC 编程口作为一个虚拟串口，首次使用时必须先安装驱动程序。当安装完成最新版本的 Kincobuilder 编程软件后，在安装目录下面的 \Drivers 目录下面存放着编程口针对各个版本 Windows 系统的驱动程序，目前支持 Windows XP、Windows 7、Windows 8 和 Windows 10 系统。当第一次使用编程数据线连接一体屏和电脑时，Windows 系统会自动检测到新硬件并提示安装驱动程序，此时用户根据自己的 Windows 版本选择相应目录下的驱动程序即可。

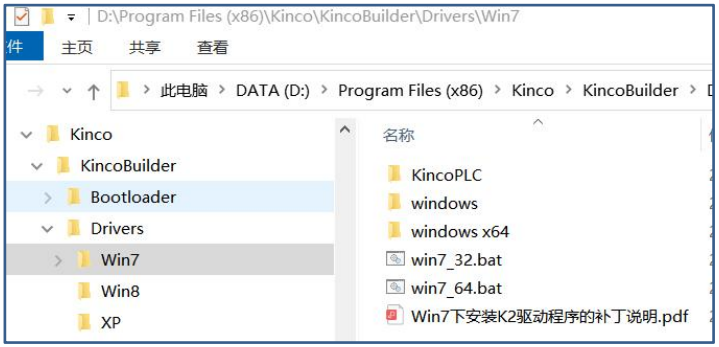
- **在 Windows 7 系统下，无法安装驱动程序怎么办？**

这是因为电脑上安装的是精简过的 Windows7 系统，系统缺少必要的文件，从而导致了无法安装虚拟串口。此时可采用如下方法解决：

方法一：

在 Kincobuilder 的安装目录的\Drivers\Win7 目录中，我们汇总了所需要的系统文件，并提供了批处理文件以方便用户使用：如果用户是 32 位的操作系统，则选择“win7_32.bat”；如果用户是 64 位的操作系统，则选择“win7_64.bat”。用户直接运行相应的批处理文件，即可把缺少的系统文件都安装好，然后就能正常安装 usb 驱动程序了。

此方法能解决大部分用户遇到的问题。



方法二：

如果用以上方法仍依然无法安装驱动程序时，用户可使用第三方驱动程序安装软件（如驱动精灵）进行自动搜索安装。

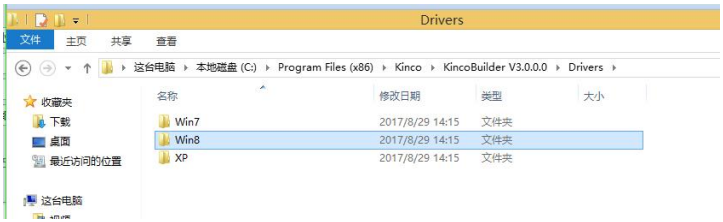
➢ 在 Windows 8 系统下，如何安装驱动程序？

如果能联网，那么可以选择 win8、 win10 的自动搜索更新功能，自动安装好驱动程序。

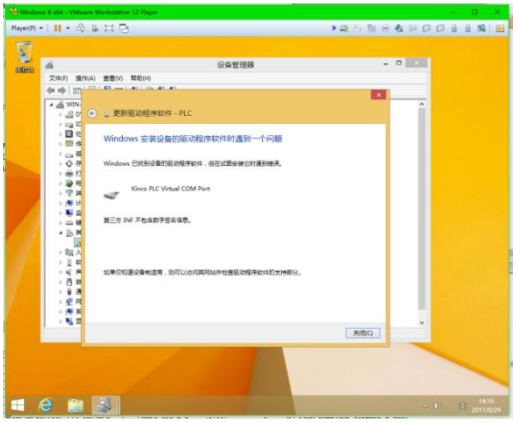
如果没法联网搜索更新，请按照以下图文指示来安装。

下面的截图是 win8 下高级启动的截图，win10 与 win8 类似，也是从【设置】里面找到高级启动然后选【7 禁止驱动程序强制签名】。

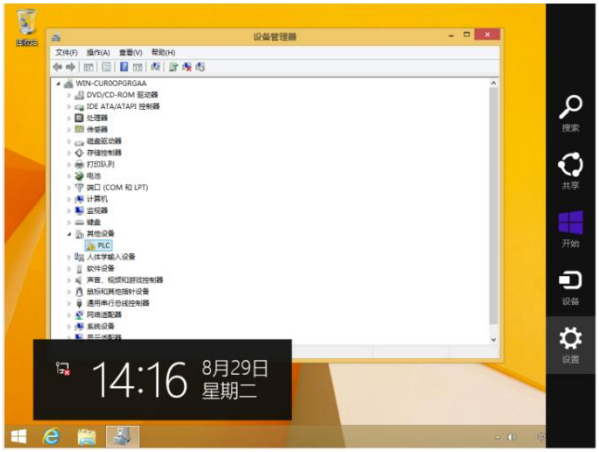
（1） 安装 PLC 驱动， 按 windows 的提示， 选择 win8 里面的驱动文件。



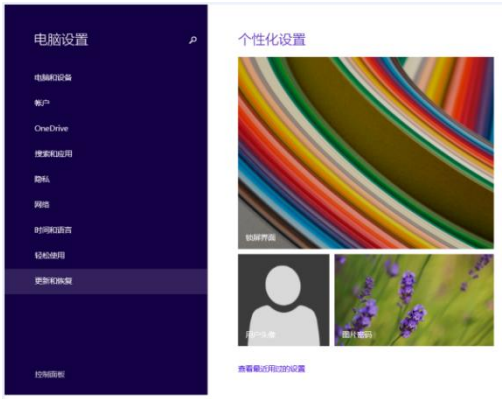
此时可能会有错误提示，如下图。



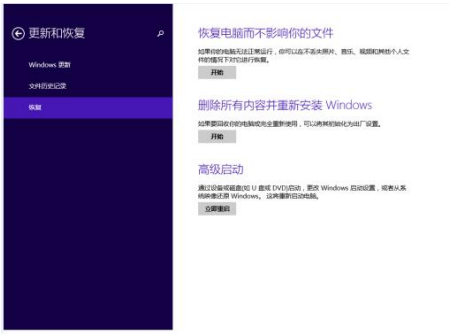
（2） 鼠标指向 win8 的右下角，出来设置，点设置



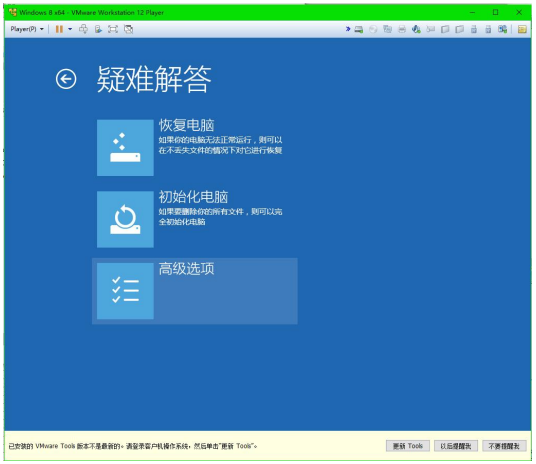
（3） 在设置里面，点【更新和恢复】



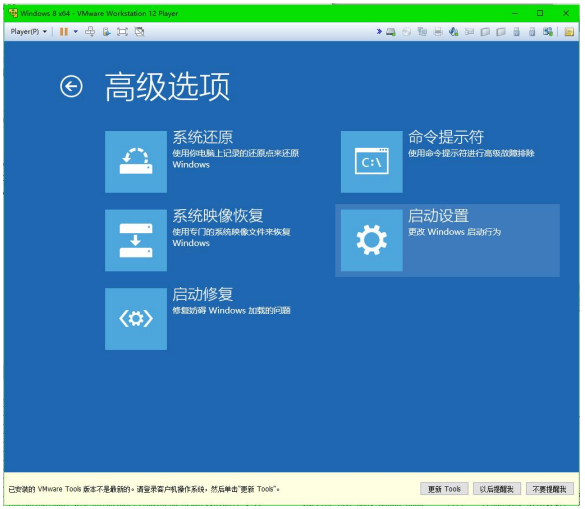
（4） 在里面点【更新】，然后点【高级启动】下面的立即启动。



（5）点【高级选项】

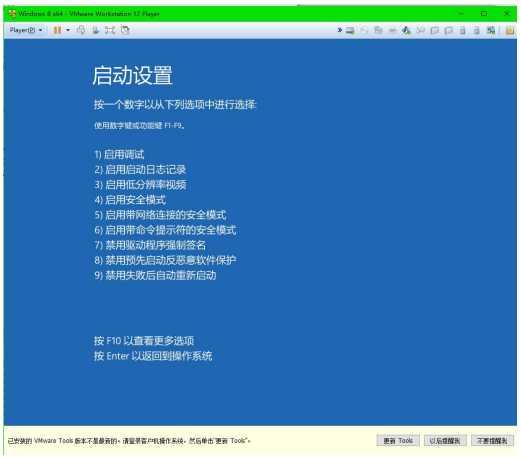


（6）点【启动设置】

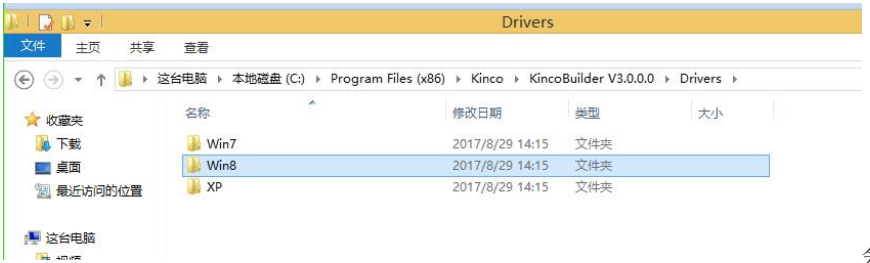


（7） 点【重启】

（8） 这是电脑重启后的界面， 选【7）禁用驱动程序强制签名】，电脑就开始启动

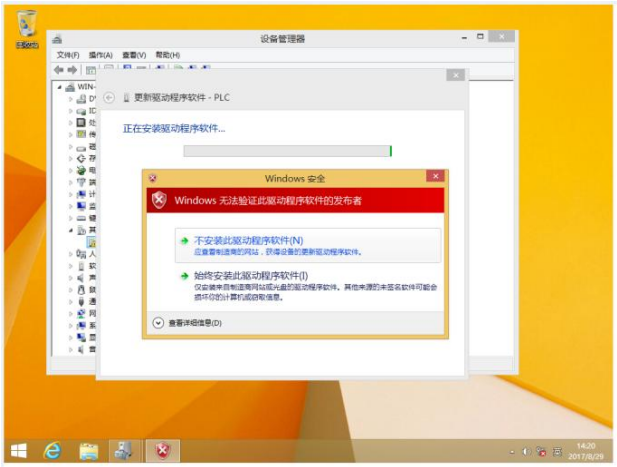


（9）重新安装 PLC 驱动， 按 windows 的提示， 选择 win8 里面的驱动文件。

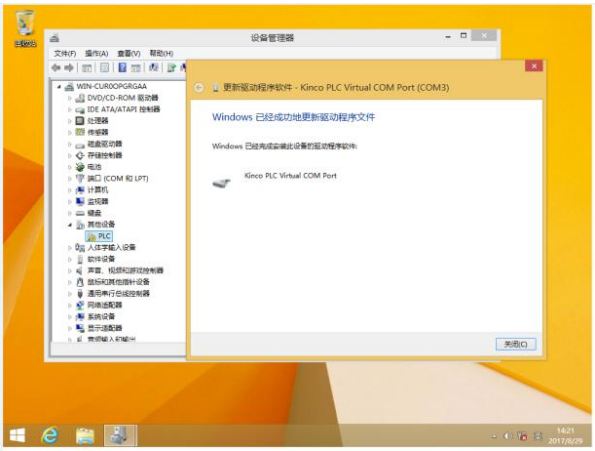


会

弹出如下提示，选【始终安装此驱动程序软件】。



（10）安装成功显示如下图，并且在设备管理器的【端口（COM 和 LPT）】里会显示出新的 USB 虚拟 COM 口号。



E：K204ET-16DT、KS101M-04DX、K6 提供了符合标准 IEEE802.3 规范的 Ethernet 接口。该接口支持编程协议，可以作为编程口使用，另外也支持 ModBus TCP Sever，即通常说的“从站”功能。

通信电缆采用直通电缆（直连线）或者交叉电缆（交叉线）均可。PLC 的以太网接口提供了“自动协商”功能，当插入电缆后，PLC 会跟通信对方自动进行协商并自动适应所用电缆。

连接 PC 时要使 PC 的 IP 与 PLC 的 IP 在同一网段（IP 的前三个数字相同，最后一位数字不同）。

2）启动 KincoBuilder，新建一个工程或者打开一个已经存在的工程。

3）PC 机通信设置

A：如果使用串行通信口或者 USB 口，则需要设置计算机串行通信口的通信参数。这些参数必须与 CPU 通信口的串行通信参数完全一致才能正常地进行通信。步骤如下：

a） 在工程管理器中，双击**【PC 机通信设置】**节点，或者在**【PC 机通信设置】**节点上单击鼠标右键，执行弹出菜单中的**【打开…】**命令，或者执行**【工具】→【PC 机通信设置…】**菜单命令，均可进入“PC 机通信设置”对话框，如图 2-9。

图 2-9“PC 机通信设置”对话框

b） 在**【目标 PLC】**中选择将要连接的目标 PLC 站号；在**【COM 口】**中选择当前计算机所用的 COM 口；依据目标 PLC 的串行通信参数来设置所选 COM 口的通信参数：如果使用的是 PLC 的串口，那么【波特率】、【奇偶校验】、【数据位】、【停止位】这 4 个参数必须与目标 PLC 的相应参数完全一致；如果是使用 PLC 的 USB 口，因为虚拟成了串口使用，所以上述通信参数可以任意设置。设置好后，单击**【确定】**按钮即可。

若使用 PLC 的串口，而用户不知道所用 CPU 的通信参数，那么该如何进行设置呢？此时有两

种方法：

- 方法一

先选择将要使用的计算机【COM 口】，然后单击【自动检测】按钮，KincoBuilder 将自动检测当前所连 CPU 的通信参数，检测所需要的时间可能是数秒到数分钟，若检测成功，那么 KincoBuilder 将依据检测到的 CPU 通信参数自动对当前所用计算机 COM 口的参数进行设置。

若用户使用的是 USB 转 RS232 的转换器，那么在安装好转换器驱动程序第一次使用时，如果不能自动检测到串口参数，用户就需要重新启动计算机以便新驱动程序生效。

- 方法二

先将所连 CPU 断电，将其运行开关拨至“STOP”位置，然后对 CPU 重新上电，此时 CPU 会将某些串口的通信参数自动设置为如下默认值：站号为 1，波特率 9600，无校验，8 位数据位，1 位停止位。用户依据此参数设置好计算机 COM 口的通信参数即可进行各种通信操作。注意：在 CPU 的通信参数被修改之前不要改变运行开关的位置！

如果 CPU 具有 PORT0（RS232）口，则 PORT0 将会被默认设置；如果 CPU 不具有 PORT0（RS232）口，则 PORT1（RS485）将会被默认设置。

B：如果使用以太网网口，则需要设置计算机的 TCP/IP 通信参数（包括 IP 地址和子网掩码）。

a） 在设置 TCP/IP 通信参数之前先确定所连 PLC 的接口参数，先使用串口通信查看接口参数，查看方法：软件菜单栏【工具】--【TCP/IP 参数配置】--点击【读取】，出现提示框“读取参数成功”，即可获得所连 PLC 的 TCP/IP 接口参数。

PLC 网口出厂时的默认参数如下：

b）选中【PC 通信设置】使用 TCP/IP，将 TCP/IP 通信设置下的 IP 地址和端口号与上一步读取到的接口参数设置相同。

注意：网口下载时，PC 的 IP 与 PLC 的 IP 一定要在同一网段。前面 3 个数字相同，最后 1 个数字不同！

4）现在设置好了计算机串口或网口的通信参数，可以接着对所连 PLC 进行编程、调试了。

2.6 如何修改 CPU 的通信参数？

- **修改 CPU 的串行通信参数**

用户连接好计算机和 CPU 之后就可以使用 KincoBuilder 来检查或者修改该 CPU 的通信参数了。

1） 首先，使用如下任一方法进入硬件配置窗口：

- 双击工程管理器中【**资源**】组下的【**PLC 硬件配置**】节点；
- 在【**PLC 硬件配置**】节点上单击鼠标右键，然后执行弹出的【**打开**...】菜单命令。

在硬件配置窗口的上半部分，以表格形式列出了工程中用到的 PLC 模块，我们称之为模块列表。PLC 模块列表表达了一个真实的 PLC 控制系统：各个模块在表格中的排列次序应当与实际模块在扩展总线上的连接次序一致。

在硬件配置窗口的下半部分，是 PLC 模块列表中选中模块的所有配置参数，我们称之为模块参数配置窗口。

2) 鼠标单击并选中模块列表中的 CPU 模块，然后在下部的模块参数配置窗口中选择【通信设置】页面，如下图，用户可以在此对其通信参数进行设置。

图 2-10 串行口通信参数配置页面

3) 通信参数配置完成后，必须将当前工程下载至 CPU 中，然后新的通信参数才会生效。

- **修改 CPU 的网口通信参数**

在修改 TCP/IP 通信参数之前先确定所连 PLC 的 TCP/IP 接口参数，先使用串口、USB 口、网口（已知 PLC 的 IP 地址所在网段时）通信查看接口参数，查看方法：软件菜单栏【工具】--【TCP/IP 参数配置】--点击【读取】，出现提示框“读取参数成功”，即可获得所连 PLC 的 TCP/IP 接口参数。然后在下面的红框中可以修改 IP 地址等参数，修改完成后点击【写入】即可。

PLC 网口的默认参数如下：

注意：在局域网内，位于同一网段（即 IP 地址的前三段数字要相同，最后一段不同）的 PLC、

PC 之间才能相互通信。

2.7 CPU 状态及指示灯

CPU 模块有两种状态：运行（RUN）状态和停止（STOP）状态。

在运行状态下，CPU 模块正常地循环执行主扫描任务和各种中断任务。

在停止状态下，CPU 模块仅处理部分通信请求(包括来自于 KincoBuilder 编程软件的编程、调试等命令，以及作为 Modbus RTU 从站响应主站的通信命令)，同时将所有的输出点（DO、AO）立即输出用户工程的【硬件配置】中定义的“停机输出”值。

➢ 改变 CPU 状态

用户可以通过如下方法来改变 CPU 状态：使用运行/停止(RUN/STOP)开关；在 KincoBuilder 软件中，执行执行【调试】->【启动…】或者【停止…】菜单命令。

这两种方法的组合操作结果如下表：

运行/停止开关位置	执行一次 KincoBuilder 菜单命令后	PLC 实际状态
RUN	启动	运行状态
	停止	停止状态
STOP	启动	停止状态
	停止	停止状态

另外，在下列情况下，CPU 会自动改变状态，而不受 RUN/STOP 开关和【启动…】、【停止…】菜单命令的影响。

① 若 CPU 在运行过程中若检测到严重错误，则会立即进入 STOP 状态。CPU 在运行中会实时检测各种错误并将错误分为三个等级：致命错误、严重错误、一般错误。当检测到致命错误时，CPU 自动进入独立的安全子系统运行，此时上述改变 CPU 状态的方式均失效；当检测到严重错误时，CPU 会立即进入 STOP 状态。请参阅 13.1 错误类型 了解更多关于错误的信息。

② 在下载程序时，CPU 先自动进入 STOP 状态，然后执行整个下载过程。当下载成功后，CPU 会依据 RUN/STOP 开关的位置自动转入相应的状态。若下载失败或者用户中途终止了下载过程，则 CPU 会继续维持 STOP 状态不变。

③ 在用户程序中调用了“STOP”指令且指令被执行了，则 CPU 会立即转入 STOP 状态。

➢ CPU 指示灯

在 CPU 模块上提供了不同的指示灯，用于指示 CPU 当前的工作情况。

由于不同系列的 CPU 功能、体积均有差别，因此指示灯的数量、种类也不一样，但具有相同标识（不区分大小写字母）的指示灯的功能都是相同的。

下面汇总列出了 KPLC 提供的所有功能的指示灯。

- 【Run】：若 CPU 正处于运行状态，则 RUN 灯点亮，否则熄灭。
- 【Stop】：若 CPU 正处于停止状态，则 STOP 灯点亮，否则熄灭。
- 【Comm】：任意一个串行通信口接收、发送数据时，Comm 灯闪烁。
- 【Err】：若 PLC 在运行时曾经发生过错误，则 Err 灯点亮，否则熄灭。
- 【NET】：无线通信指示，当无线通信接口接收、发送数据时，NET 灯闪烁。

第三章 PLC 编程基础

本章详细描述了对 Kinco-K 系列 PLC 编程的基础知识,同时也介绍了 IEC61131-3 标准中的一些基本概念，这些概念对于用户使用任何一种 IEC61131-3 软件都是非常有用的。本章的目的是帮助用户开始编程的学习和实践，达到“知其然并知其所以然”的程度。

在初次阅读时，并不需要用户对每个环节都理解得非常透彻，但建议用户采取“边阅读，边实验”的方式，这样会有助于对本章内容的理解。同时建议用户阅读后续章节时也采用这种方式。

3.1 程序组织单元（POU，Programme Organization Unit）

IEC61131-3 引入 POU 的概念。POU 包含有程序代码，是独立的、最小的软件单元，它是程序和工程的基本单元。传统的 PLC 制造商在编程方面为各自的 PLC 定义了各种类型的块,IEC61131-3 将这些块统一为 3 种类型的 POU。

用户程序由一个或者多个 POU 组成，这些 POU 可以是用户自己创建的。POU 之间可以相互调用，但是不允许递归调用，在 IEC61131-3 中明确规定了 POU 禁止直接或者间接调用自身。
下面描述了标准的 POU 类型。

- 程序（Programme）

  关键字：PROGRAMME。

  这种类型的 POU 代表了“主程序”，用于执行一定的任务。

  它可以有输入和输出参数，但是无返回值。

- 功能（Function）

  关键字：FUNCTION 。

  这种类型的 POU 可以有输入参数，只有一个返回值，其返回值是通过功能名带回的。当以相同的输入参数调用功能时，其返回值总是相同的。

  它主要用于代码重用，可被其它 POU 调用。

- 功能块（Function Block）

关键字：FUNCTION_BLOCK。

这种类型的 POU 简称 FB。它可以有输入、输出参数，并具有静态变量（即能够记忆 FB 以前的状态）。FB 的输出值通过其输出参数传递。FB 的输出不仅取决于其输入值，而且也取决于在其静态变量中储存的状态值。

FB 也主要用于代码重用，可被其它 POU 调用。

3.2 数据类型

数据类型定义了数据的位长度、取值范围及其初始化值。

在 IEC61131-3 中定义了一组最常用的基本数据类型,因此在 PLC 领域内这些数据类型的含义以及使用方式是开放、统一的。目前 K 系列 PLC 支持的基本数据类型如下表所示。

数据类型	描述	长度（位）	取值范围	缺省初始值
BOOL	布尔型	1	true, false	false
BYTE	8 位位串	8	0 ~ (2^8-1)	0
WORD	16 位位串	16	0 ~ $(2^{16}-1)$	0
DWORD	32 位位串	32	0 ~ $(2^{32}-1)$	0
INT	整型，有符号	16	-2^{15} ~ （$2^{15}-1$）	0
DINT	双整型，有符号	32	-2^{31} ~ （$2^{31}-1$）	0
REAL	实型	32	采用 ANSI/IEEE754-1985 标准； $1.18*10^{-38}$ ~ $3.40*10^{38}$， 0， $-3.40*10^{38}$ ~ $-1.18*10^{-38}$	0.0

表 3-1　Kinco-K 系列支持的基本数据类型

PLC 中的实数类型遵循 ANSI/IEEE754-1985 标准，也就是 C 语言中的 float（单精度）类型。

- **关于 REAL 型数据的舍入误差**

实数的二进制表示是不可能很精确的。因为 REAL 数据在存储时占用 4 个字节的空间，能够表

示的有效数字最多是 6-7 位，在有效位之外的数字将被舍入，因此会产生舍入误差。

所谓的有效数字，是从一个数据最左边第一个不是 0 的数字起，到最后一位数字止，这中间所有的数字都叫做这个数的有效数字。比如，3.3、0.33 的有效数字位数都是 2 位，3.30、0.330 的有效数字都是 3 位。

根据上面的描述，我们举个例子，1.23456 在 KPLC 中是能够精确表示的，而 1.2345678 就不能精确表示。

- **关于实数“0.0”**

由于存在舍入误差，“0.0”在 PLC 中可能不会精确表示出来。因此，我们根据实数能够表示的最大有效数字位数，做如下规定：凡是位于[-0.000001, 0.000001]范围内的实数，PLC 都会作为“0.0”进行处理。

- **关于实数的比较**

当使用比较指令（GT、GE、EQ、NE、LT、LE）时，我们需要注意：由于存在舍入误差，因此两个实数不可能进行精确的比较。根据前面对“0.0”的描述，当两个实数的绝对值的差值位于[-0.000001, 0.000001]之内，PLC 就认为这两个实数是相等的，否则才是不相等的。

3.3 标识符

标识符是由数字、字母、下划线字符组成的字符串，它必须以一个字母或者下划线开始。编程人员可以使用标识符来为变量、程序等定义名称。

3.3.1 标识符的定义

标识符的定义和使用必须依据如下原则：

- 必须以一个字母或者一个单一的下划线字符开始，随后是一定数量的数字、字母或者下划线。
- 标识符是大小写无关的。比如，abc、ABC、aBC 是同一个标识符。
- 标识符的长度仅受各编程系统的限制。在 KincoBuilder 中，标识符的最大长度是 16 个字符。
- 用户自定义的标识符不允许使用关键字。关键字是标准的标识符，其拼写形式和使用目的均由 IEC61131-3 明确规定。

3.3.2 标识符的使用

下面列出了在 KincoBuilder 中可使用标识符的语言元素：

- 程序、功能、功能块名称
- 变量
- 语句标号等

3.4 常量介绍

在程序运行的过程中，其值不能改变的量称为常量。常量的特性由它的值和数据类型来描述。根据数据类型的不同，常量的书写格式各不相同。下表列出了 Kinco-K 系列支持的各种类型常量的定义及示例。

数据类型	格 式[1]	取值范围	示例
BOOL	true、false	true 代表真，false 代表假	false
BYTE	B#十进制数字	B#0 ~ B#255	B#129
	B#2#二进制数字		B#2#10010110
	B#8#八进制数字		B#8#173
	B#16#十六进制数字		B#16#3E
WORD	W#十进制数字	W#0~ W#65535	W#39675
	2#二进制数字		2#100110011
	W#2#二进制数字		W#2#110011
	8#八进制数字		8#7432
	W#8#八进制数字		W8#174732
	16#十六进制数字		16#6A7D
	W#16#十六进制数字		W#16#9BFE
DWORD	DW#十进制数字	DW#0 ~ DW#4294967295	DW#547321
	DW#2#二进制数字		DW#2#10111

	DW#8#八进制数字		DW#8#76543
	DW#16#十六进制数字		DW#16#FF7D
INT	十进制数字	-32768~32767	12345
	I#十进制数字		I#-2345
	I#2#二进制数字(2)		I#2#1111110
	I#8#八进制数字(2)		I#8#16732
	I#16#十六进制数字(2)		I#16#7FFF
DINT	DI#十进制数字	DI#-2147483648~DI#2147483647	DI#8976540
	DI#2#二进制数字(2)		DI#2#101111
	DI#8#八进制数字(2)		DI#8#126732
	DI#16#十六进制数字(2)		DI#16#2A7FF
REAL	带小数点的十进制数字	$1.18*10^{-38}$ ~ $3.40*10^{38}$， 0， $-3.40*10^{38}$ ~ $-1.18*10^{-38}$	1.0，-243.456
	指数形式：xEy x：带小数点的十进制数字 y：整数。		-2.3E-23

表 3-2 常量的定义

提示：

⑴在 IEC61131-3 中，标识符的使用是大小写无关的。因此在程序中将格式字符写为大写或者小写均合法，比如 W#234，dw#12345 均为合法常量。

具体请参见关于标识符的定义部分。

⑵ INT、DINT 型常量的二进制、八进制、十六进制表示方法均采用了通用计算机中标准的补码表示法，其最高有效位(MSB)是符号位：MSB 为 1 则代表负数，MSB 为 0 则代表正数。比如 I#16#FFFF＝-1，I#16#7FFF＝32767，I#16#8000＝-32768 等。

3.5 变量介绍

在程序的运行过程中，其值可以改变的量称为变量。

一个变量必须有一个名字，占据一定的存储单元，在该存储单元中存放着变量的值。在 IEC61131-3 中，变量的存储位置可以由用户自行指定一个有效的 PLC 内存地址，也可以由编程系统自行分配。

3.5.1 变量声明

变量的使用必须遵循“先声明，后使用”的原则。变量的命名请参阅 3.3.1 标识符的定义。在声明变量时，必须为它指定一个确定的数据类型，同时也必须指定它的变量类型。

在 IEC61131-3 中定义了各种变量类型。比如，变量可以被定义为一个 POU 的形式参数；也可以在一个 POU 内定义，仅作为本 POU 的局部变量；也可以在 POU 外定义，在整个工程范围内作为全局变量使用。下表描述了 Kinco-K 系列支持的标准变量类型。

变量类型	存储权限		描述
	外部	内部	
VAR	---	读写	局部变量。 只能在定义它的 POU 内使用，在此 POU 外部是不可见的。
VAR_INPUT	写	读	输入变量。在定义它的 POU 内作为 POU 的输入参数仅可读，在调用此 POU 时此变量仅可写。
VAR_OUTPUT	读	读写	输出变量。在定义它的 POU 内作为 POU 的输出参数可读写，在调用此 POU 时此变量仅可读。
VAR_IN_OUT	读写	读写	输入/输出变量，是 VAR_INPUT 和 VAR_OUTPUT 的组合类型。在定义它的 POU 内以及在调用此 POU 时此变量均可读写。
VAR_GLOBAL	读写	读写	全局变量。可被所有的 POU 直接读写。

表 3-3　Kinco-K 系列支持的变量类型

3.5.2 在 KincoBuilder 中声明变量

在 KincoBuilder 中，各种变量的声明均在相应的表格中进行，这样既避免了让用户进行繁琐的输入，同时软件还能够对用户的输入进行严格的语法检查。

全局变量的声明在全局变量表中完成。POU 的局部变量、形式参数在该 POU 编辑界面的变量声明表格中完成。若全局变量与局部变量的名称相同，则在程序中局部变量的使用优先。
具体的使用方法请参见本手册中关于界面的详细介绍部分。

3.5.3 变量的检验

在编辑、编译程序时，KincoBuilder 可以自动对变量的使用情况进行检验，即判别该变量是否按照其变量类型、数据类型进行处理。这也是 IEC61131-3 的重要优点之一。比如，在程序中为一个 WORD 类型的变量赋一个 BOOL 类型的值，或者对一个 VAR_INPUT 型的变量进行赋值操作，KincoBuilder 就会向用户进行错误警告并提示修改。

由于变量的特性取决于其变量类型和数据类型，因此这种检验能够在很大程度上避免由于变量使用而引起的错误。

3.6 内存区域及寻址方式

内存区域中的每个单元都有明确、唯一的编号，这个编号就是内存单元的物理地址。物理地址是一个抽象的数值，不便于在程序中使用。所以为了方便用户，在 Kinco-K 系列中内存被进一步划分为不同类型的几个区域并对各区域重新按字节进行编址，为每个内存单元都分配了一个直接地址，例如%IO.0、%VW20 等。在这种情况下，直接地址就如同一个变量一样。实际上在本书后面的描述中，也经常用到诸如“直接地址变量”、“变量%VW100”这样的说法。

3.6.1 内存区域类型及其特性

I	
描述	DI（开关量输入）映像区 在每个扫描周期的开始，CPU 读取所有物理 DI 通道的状态并将这些状态写入 I 区中以供用户程序使用。
访问方式	可按位、字节、字、双字访问
存储权限	只读
其它	允许强制，不能掉电保持
Q	
描述	DO（开关量输出）映像区 在每个扫描周期的结束，CPU 将 Q 区中的值全部输出至物理 DO 通道。
访问方式	可按位、字节、字、双字访问
存储权限	可读、可写
其它	允许强制，不能掉电保持
AI	
描述	AI（模拟量输入）映像区 AI 扩展模块对模拟量输入信号进行采样并转换为整型数值。在每个扫描周期的开始，CPU 从所有 AI 扩展模块读取测量值并写入 AI 区中以供用户程序使用。
访问方式	可按字（INT 型）访问
存储权限	可读
其它	允许强制，不能掉电保持
AQ	
描述	AO（模拟量输出）映像区 在每个扫描周期的结束，CPU 将 AQ 区中的所有数值全部输出至物理 AO 通道。
访问方式	可按字（INT 型）访问
存储权限	可读、可写
其它	允许强制，不能掉电保持
HC	
描述	高速计数器区。用于存放各高速计数器的当前计数值。
访问方式	可按双字（DINT 型）访问
存储权限	可读

其它	不允许强制，不能掉电保持
V	
描述	变量存储区。该区域比较大，可用于存储大量的数据。
访问方式	可按位、字节、字、双字访问
存储权限	可读、可写
其它	允许强制，允许掉电保持
M	
描述	内部存储区。用于一些中间状态或者其它数据。 与 V 区相比，M 区的访问速度更快，更有利于位操作。
访问方式	可按位、字节、字、双字访问
存储权限	可读、可写
其它	允许强制，不允许掉电保持
SM	
描述	系统存储区。 用于存储数据。用户程序可以访问 SM 区中的一些地址来获取系统当前的状态信息，也可以利用 SM 区的一些地址来选择和控制 CPU 的某些特殊功能。
访问方式	可按位、字节、字、双字访问
存储权限	可读、可写
其它	不允许强制，不能掉电保持
L	
描述	局部变量区。 所有 POU 的局部变量、输入/输出参数均在 L 区内自动分配地址。 不建议用户直接操作 L 区。
访问方式	可按位、字节、字、双字访问
存储权限	可读、可写
其它	不允许强制，不能掉电保持

表 3-4　Kinco-K 系列 PLC 的内存区域分配

3.6.2 内存区域的直接寻址

用户可以使用直接地址来存取其中的数据，这种方式称为直接寻址。请务必注意“直接地址”和“直接地址中的数据”这两个概念的区别。

3.6.2.1 直接地址表示格式

IEC61131-3 规定，所有的直接地址都必须以“%”开始。用户在编程时输入直接地址的时候，可以不输入“%”，Kincobuilder 可以自动加上，比如，若用户输入“I0.0”，那么 Kincobuilder 可以自动转换为“%I0.0”。

各内存区域的直接寻址方式如下述列表。表中的 x、y 均代表十进制数字。

- I 区

位寻址	格式	**%I**x.y
	描述	x：字节地址，表示在 I 区中该内存单元所在字节的编号（地址）。 y：位地址，表示位于字节 x 的第几位。范围为 0~7。
	数据类型	BOOL
	示例	%I0.0 %I0.7 %I5.6
字节寻址	格式	%IBx
	描述	x：字节地址，即在 I 区中该内存单元所在字节的编号（地址）。
	数据类型	BYTE
	示例	%IB0 %IB1 %IB10
字寻址	格式	%IWx
	描述	x：字节地址，即在 I 区中该内存单元首字节的地址。 由于 WORD 类型的数据长度为 2 字节，因此 x 必须为偶数。
	数据类型	WORD、INT
	示例	%IW0 %IW2 %IW12
双字寻址	格式	%IDx
	描述	x：字节地址，即在 I 区中该内存单元首字节的地址。 由于 DWORD 类型的数据长度为 4 字节，因此 x 必须为偶数。
	数据类型	DWORD、DINT
	示例	%ID0 %ID4 %ID12

- **Q 区**

位寻址	格式	**%Q**x.y
	描述	x：字节地址，即在 Q 区中该内存单元所在字节的编号（地址）。 y：位地址，表示位于字节 x 的第几位。范围为 0~7。
	数据类型	BOOL
	示例	%Q0.0 %Q0.7 %Q5.6
字节寻址	格式	%QBx
	描述	x：字节地址，即在 Q 区中该内存单元所在字节的编号（地址）。
	数据类型	BYTE
	示例	%QB0 %QB1 %QB10
字寻址	格式	%QWx
	描述	x：字节地址，即在 Q 区中该内存单元首字节的地址。 由于 WORD 类型的数据长度为 2 字节，因此 x 必须为偶数。
	数据类型	WORD、INT
	示例	%QW0 %QW2 %QW12
双字寻址	格式	%QDx
	描述	x：字节地址，即在 Q 区中该内存单元首字节的地址。 由于 DWORD 类型的数据长度为 4 字节，因此 x 必须为偶数。
	数据类型	DWORD、DINT
	示例	%QD0 %QD4 %QD12

- **M 区**

位寻址	格式	**%M**x.y
	描述	x：字节地址，即在 M 区中该内存单元所在字节的编号（地址）。 y：位地址，表示位于字节 x 的第几位。范围为 0~7。
	数据类型	BOOL
	示例	%M0.0 %M0.7 %M5.6
字节寻址	格式	%MBx
	描述	x：字节地址，即在 M 区中该内存单元所在字节的编号（地址）。
	数据类型	BYTE
	示例	%MB0 %MB1 %MB10

字寻址	格式	%MWx
	描述	x：字节地址，即在 M 区中该内存单元首字节的地址。 由于字数据长度为 2 字节，因此 x 必须为偶数。
	数据类型	WORD、INT
	示例	%MW0 %MW2 %MW12
双字寻址	格式	%MDx
	描述	x：字节地址，即在 M 区中该内存单元首字节的地址。 由于双字数据长度为 4 字节，因此 x 必须为偶数。
	数据类型	DWORD、DINT
	示例	%MD0 %MD4 %MD12

- **V 区**

位寻址	格式	**%V**x.y
	描述	x：字节地址，即在 V 区中该内存单元所在字节的编号（地址）。 y：位地址，表示位于字节 x 的第几位。范围为 0~7。
	数据类型	BOOL
	示例	%V0.0 %V0.7 %V5.6
字节寻址	格式	%VBx
	描述	x：字节地址，即在 V 区中该内存单元所在字节的编号（地址）。
	数据类型	BYTE
	示例	%VB0 %VB1 %VB10
字寻址	格式	%VWx
	描述	x：字节地址，即在 V 区中该内存单元首字节的地址。 由于字数据长度为 2 字节，因此 x 必须为偶数。
	数据类型	WORD、INT
	示例	%VW0 %VW2 %VW12
双字寻址	格式	**%VD**x 或者**%VR**x
	描述	x：字节地址，即在 V 区中该内存单元首字节的地址。 由于双字数据长度为 4 字节，因此 x 必须为偶数。
	数据类型	DWORD、DINT（**%VD**x）；REAL（**%VR**x）
	示例	%VD0、%VD12；REAL 型：%VR0、%VR4

- SM 区

位寻址	格式	**%SM**x.y
	描述	x：字节地址，即在 SM 区中该内存单元所在字节的编号（地址）。 y：位地址，表示位于字节 x 的第几位。范围为 0~7。
	数据类型	BOOL
	示例	%SM0.0　%SM0.7　%SM5.6
字节寻址	格式	%SMBx
	描述	x：字节地址，即在 SM 区中该内存单元所在字节的编号（地址）。
	数据类型	BYTE
	示例	%SMB0　%SMB1　%SMB10
字寻址	格式	%SMWx
	描述	x：字节地址，即在 SM 区中该内存单元首字节的地址。 由于字数据长度为 2 字节，因此 x 必须为偶数。
	数据类型	WORD、INT
	示例	%SMW0　%SMW2　%SMW12
双字寻址	格式	%SMDx
	描述	x：字节地址，即在 SM 区中该内存单元首字节的地址。 由于双字数据长度为 4 字节，因此 x 必须为偶数。
	数据类型	DWORD、DINT
	示例	%SMD0　%SMD4　%SMD12

- L 区（注意：不建议用户使用直接地址来访问 L 区）

位寻址	格式	**%L**x.y
	描述	x：字节地址，即在 L 区中该内存单元所在字节的编号（地址）。 y：位地址，表示位于字节 x 的第几位。范围为 0~7。
	数据类型	BOOL
	示例	%L0.0　%L0.7　%L5.6
字节寻址	格式	%LBx
	描述	x：字节地址，即在 L 区中该内存单元所在的字节的编号（地址）。
	数据类型	BYTE
	示例	%LB0　%LB1　%LB10

字寻址	格式	%LWx
	描述	x：字节地址，即在 L 区中该内存单元首字节的地址。 由于字数据长度为 2 字节，因此 x 必须为偶数。
	数据类型	WORD、INT
	示例	%LW0　%LW2　%LW12
双字寻址	格式	%LDx
	描述	x：字节地址，即在 L 区中该内存单元首字节的地址。 由于双字数据长度为 4 字节，因此 x 必须为偶数。
	数据类型	DWORD、DINT、REAL
	示例	%LD0　%LD4　%LD12

- **AI 区**

字寻址	格式	%AIWx
	描述	x：字节地址，即在 AI 区中该内存单元首字节的地址。 由于字数据长度为 2 字节，因此 x 必须为偶数。
	数据类型	INT
	示例	%AIW0　%AIW2　%AIW12

- **AQ 区**

字寻址	格式	%AQWx
	描述	x：字节地址，即在 AQ 区中该内存单元首字节的地址。 由于字数据长度为 2 字节，因此 x 必须为偶数。
	数据类型	INT
	示例	%AQW0　%AQW2　%AQW12

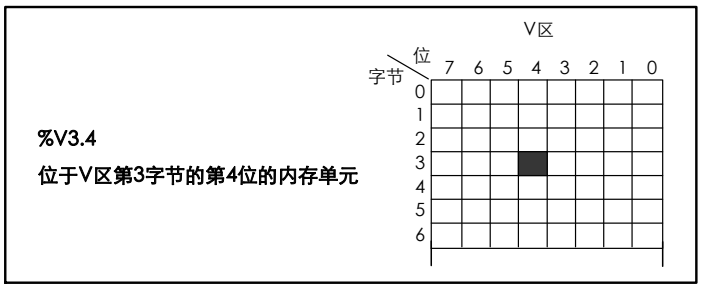
- **HC 区**

字寻址	格式	%HCx
	描述	x：高速计数器的编号。
	数据类型	DINT
	示例	%HC0　%HC1

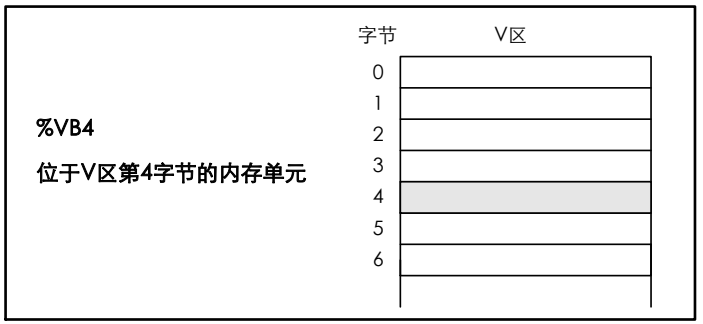
3.6.2.2 直接地址与内存单元之间的映射

每一个合法的直接地址都对应于 CPU 中的一个内存单元。在程序中对直接地址的操作就是对其对应的内存单元进行操作。下面将以 V 区为例图解直接地址与内存单元之间的映射关系。

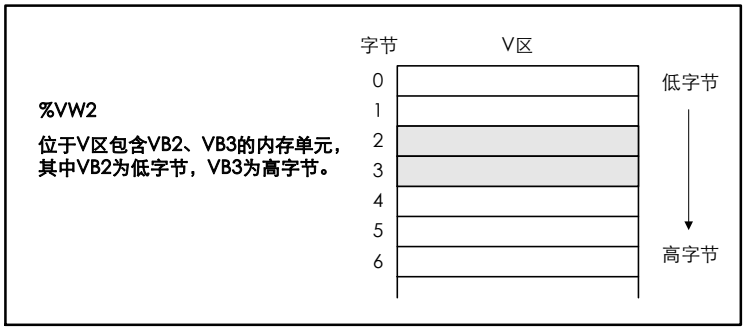
• 位地址：



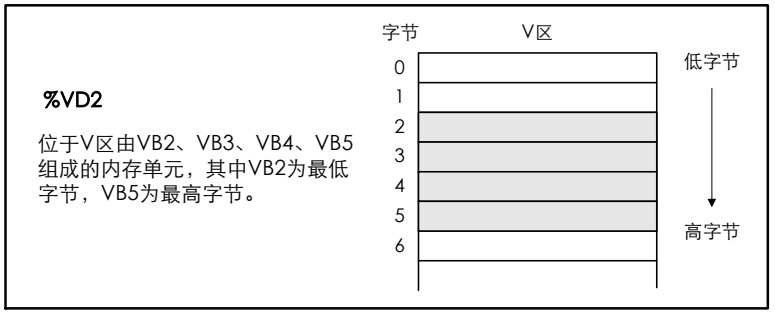
• 字节地址：



• 字地址：



• 双字地址：



以 V 区为例来说明字节、字、双字之间的关系：

地址	Addr	Addr+1	Addr+2	Addr+3
BYTE	VB0	VB1	VB2	VB3
BOOL	0...7	0...7	0...7	0...7
	V0.0...V0.7	V1.0...V1.7	V2.0...V2.7	V3.0...V3.7
WORD	VW0		VW2	
	0...7	8...15	0...7	8...15
DWORD	VD0			
	0...7	8...15	16...23	24...31
REAL	VR0			

如上图 K 系列 PLC 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）。采用小端模式的存储方式，即高地址存放变量的高位，低地址存放变量的低位。

要注意，按不同数据类型访问，数据存储区可能是重叠的。比如%VW0 的数值为 3， 则%VB0 的值也为 3，%V0.0 的值为 TRUE，%VX0.1 的值也为 TRUE。为了更好的理解存储方式，请看下图。

	内存地址	监视长度	显示格式	内存值	内存值	内存值
1	%VB0	4	十六进制	B#16#78	B#16#56	B#16#34
2	%VB3			B#16#12		
3						
4	%VW0	2	十六进制	W#16#5678	W#16#1234	
5						
6	%VD0	1	十六进制	DW#16#12345678		
7						
8	%VR0	1	十进制(无符号)	5.690e-028		
9						
10	%V0.0	32	十进制(有符号)	FALSE	FALSE	FALSE
11	%V0.3			TRUE	TRUE	TRUE
12	%V0.6			TRUE	FALSE	FALSE
13	%V1.1			TRUE	TRUE	FALSE
14	%V1.4			TRUE	FALSE	TRUE
15	%V1.7			FALSE	FALSE	FALSE
16	%V2.2			TRUE	FALSE	TRUE
17	%V2.5			TRUE	FALSE	FALSE
18	%V3.0			FALSE	TRUE	FALSE
19	%V3.3			FALSE	TRUE	FALSE
20	%V3.6			FALSE	FALSE	
21						
22						

3.6.3 内存区域的间接寻址

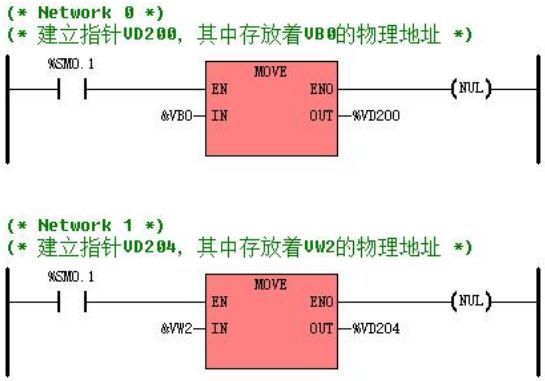
指针是一个 32 位长度的变量，用于存放一个内存单元的物理地址。用户可以利用指针来存取对应内存单元中的数据，这种方式称为间接寻址。

在 Kinco-K 系列中，只允许使用 V 区中的直接地址变量（双字长度）来作为指针。另外，只允许使用指针对 V 区进行间接寻址，但是不支持位变量的间接寻址。

3.6.3.1 建立指针

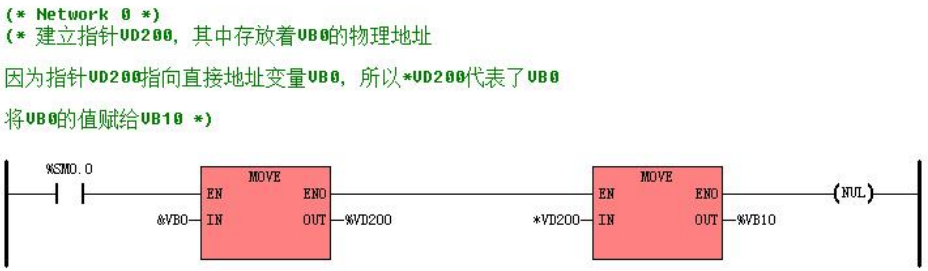
为了实现对某个内存单元的间接寻址，必须先建立起它的指针。这时候需要用到取地址运算符“&”，例如，&VB100 表示 VB100 的物理地址。

建立指针的方法为：使用取地址运算符获取某个内存单元的物理地址，并将其使用 MOVE 指令写入另一个直接地址变量来作为指针。例如：



3.6.3.2 使用指针存取数据

在指令中的操作数前加上“*”就表示该操作数是一个指针。“*”是指针运算符，在一个指针之前加“*”就代表了该指针所指向的直接地址变量。在使用指针作为指令的操作数时，需要注意指针所指向变量的数据类型。例如：



3.6.3.3 修改指针的值

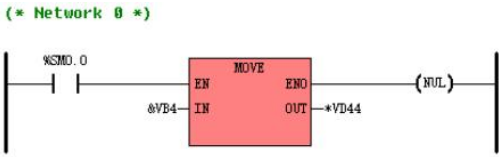
由于指针是一个 32 位的变量，因此可以使用指令修改它的值，比如加法、减法等指令。需要注意的是，指针的值每改变 1，那么它所指向的直接地址就相应改变了 1 个字节。在修改指针的值时，需要注意它所指向变量的数据类型：

- 若指向字节长度（BYTE 型）的变量，则指针值的改变量可以是任意的双整数。
- 若指向字长度（INT 或者 WORD 型）的变量，则指针值的改变量须是 2 的倍数。
- 若指向双字长度（DINT、DWORD 或者 REAL 型）的变量，则指针值的改变量须是 4 的倍数。

3.6.3.4 使用指针的注意事项

指针的有效性是由用户程序自己保证的。指针比较灵活，因此用户使用时必须小心，如果在程序中让指针指向了一个非法的变量地址，那么就会导致不可预期的结果。

Kinco-K 系列只支持一重的指针与地址，多重的应用是非法的。比如下面的指令就是非法的：



3.6.3.5 间接寻址示例

3.6.4 内存区域的地址范围

Kinco-K 系列有几种不同规格的 CPU。各型 CPU 中内存区域的地址范围有所不同，超出允许范围的地址是非法的。下面对各系列 PLC 内存区域进行了详细说明。

		K6 系列及 KM 系列（KS101M、K209M）
I	长度（字节）	32
	位地址	%I0.0 --- %I31.7

	字节地址	%IB0 --- %IB31
	字地址	%IW0 ---%IW30
	双字地址	%ID0 ---%ID28
Q	长度（字节）	32
	位地址	%Q0.0 --- %Q31.7
	字节地址	%QB0 --- %QB31
	字地址	%QW0 ---%QW30
	双字地址	%QD0 ---%QD28
AI	长度（字节）	128
	字地址	%AIW0 --- %AIW126
AQ	长度（字节）	128
	字地址	%AQW0 --- %AQW126
HC	长度（字节）	32
	双字地址	%HC0---%HC7
V	长度（字节）	16384
	位地址	%V0.0 ---%V16383.7
	字节地址	%VB0 --- %VB16383
	字地址	%VW0 --- %VW16382
	双字地址	%VD0 --- %VD16380 %VR0 --- %VR16380
M	长度（字节）	4096
	位地址	%M0.0 --- %M4095.7
	字节地址	%MB0 --- %MB4095
	字地址	%MW0 --- %MW4094
	双字地址	%MD0 --- %MD4092
SM	长度（字节）	2048
	位地址	%SM0.0 --- %SM2047.7
	字节地址	%SMB0 --- %SMB2047
	字地址	%SMW0 --- %SMW2046
	双字地址	%SMD0 --- %SMD2044
L	长度（字节）	2048
	位地址	%L0.0 --- %L2047.7

	字节地址	%LB0 --- %LB2047
	字地址	%LW0 --- %LW2046
	双字地址	%LD0 --- %LD2044

表 3-3 K6 系列及 KM 系列（KS101M、K209M）的内存范围

		MK 系列、KS 系列（KS101M 除外）、KW 系列
I	长度（字节）	32
	位地址	%I0.0 --- %I31.7
	字节地址	%IB0 --- %IB31
	字地址	%IW0 ---%IW30
	双字地址	%ID0 ---%ID28
Q	长度（字节）	32
	位地址	%Q0.0 --- %Q31.7
	字节地址	%QB0 --- %QB31
	字地址	%QW0 ---%QW30
	双字地址	%QD0 ---%QD28
AI	长度（字节）	128
	字地址	%AIW0 --- %AIW126
AQ	长度（字节）	128
	字地址	%AQW0 --- %AQW126
HC	长度（字节）	16
	双字地址	%HC0－－－%HC3
V	长度（字节）	4096
	位地址	%V0.0 ---%V4095.7
	字节地址	%VB0 --- %VB4095
	字地址	%VW0 --- %VW4094
	双字地址	%VD0 --- %VD4092 %VR0 --- %VR4092
M	长度（字节）	1024
	位地址	%M0.0 --- %M1023.7
	字节地址	%MB0 --- %MB1023

	字地址	%MW0 --- %MW1022
	双字地址	%MD0 --- %MD1020
SM	长度（字节）	300
	位地址	%SM0.0 --- %SM299.7
	字节地址	%SMB0 --- %SMB299
	字地址	%SMW0 --- %SMW298
	双字地址	%SMD0 --- %SMD296
L	长度（字节）	272
	位地址	%L0.0 --- %L271.7
	字节地址	%LB0 --- %LB271
	字地址	%LW0 --- %LW270
	双字地址	%LD0 --- %LD268

表 3-4　MK 系列、KS 系列（KS101M 除外）、KW 系列的内存范围

		K5 系列
I	长度（字节）	32
	位地址	%I0.0 --- %I31.7
	字节地址	%IB0 --- %IB31
	字地址	%IW0 ---% IW30
	双字地址	%ID0 --- %ID28
Q	长度（字节）	32
	位地址	%Q0.0 --- %Q31.7
	字节地址	%QB0 --- %QB31
	字地址	%QW0 --- %QW30
	双字地址	%QD0 --- %QD28
AI	长度（字节）	64
	字地址	%AIW0 --- %AIW62
AQ	长度（字节）	64
	字地址	%AQW0 -- %AQW62

HC	长度（字节）	16
	双字地址	%HC0－－－%HC3
V	长度（字节）	4096
	位地址	%V0.0 ---%V4095.7
	字节地址	%VB0 --- %VB4095
	字地址	%VW0 --- %VW4094
	双字地址	%VD0 --- %VD4092 %VR0 --- %VR4092
M	长度（字节）	1024
	位地址	%M0.0 --- %M1023.7
	字节地址	%MB0 --- %MB1023
	字地址	%MW0 --- %MW1022
	双字地址	%MD0 --- %MD1020
SM	长度（字节）	300
	位地址	%SM0.0 --- %SM299.7
	字节地址	%SMB0 --- %SMB299
	字地址	%SMW0 --- %SMW298
	双字地址	%SMD0 --- %SMD296
L	长度（字节）	272
	位地址	%L0.0 --- %L271.7
	字节地址	%LB0 --- %LB271
	字地址	%LW0 --- %LW270
	双字地址	%LD0 --- %LD268

表 3-5　K5 系列的内存范围

		K2 系列（K209M 除外）
I	长度（字节）	16

	位地址	%I0.0 --- %I15.7
	字节地址	%IB0 --- %IB15
	字地址	%IW0 ---%IW14
	双字地址	%ID0 ---%ID12
Q	长度（字节）	16
	位地址	%Q0.0 --- %Q15.7
	字节地址	%QB0 --- %QB15
	字地址	%QW0 ---%QW14
	双字地址	%QD0 ---%QD12
AI	长度（字节）	32
	字地址	%AIW0 --- %AIW30
AQ	长度（字节）	32
	字地址	%AQW0 --- %AQW30
HC	长度（字节）	16
	双字地址	%HC0－－－%HC3
V	长度（字节）	4096
	位地址	%V0.0 ---%V4095.7
	字节地址	%VB0 --- %VB4095
	字地址	%VW0 --- %VW4094
	双字地址	%VD0 --- %VD4092 %VR0 --- %VR4092
M	长度（字节）	1024
	位地址	%M0.0 --- %M1023.7
	字节地址	%MB0 --- %MB1023
	字地址	%MW0 --- %MW1022
	双字地址	%MD0 --- %MD1020
SM	长度（字节）	300
	位地址	%SM0.0 --- %SM299.7
	字节地址	%SMB0 --- %SMB299
	字地址	%SMW0 --- %SMW298
	双字地址	%SMD0 --- %SMD296
L	长度（字节）	272

	位地址	%L0.0 --- %L271.7
	字节地址	%LB0 --- %LB271
	字地址	%LW0 --- %LW270
	双字地址	%LD0 --- %LD268

表 3-6 K2 系列（K209M 除外）的内存范围

3.6.5 关于功能块以及功能块实例

3.6.5.1 IEC61131-3 中定义的标准功能块

在 IEC61131-3 中定义了如下标准的功能块：

- 定时器：TP --- 脉冲定时器；TON --- 接通延时定时器；TOF --- 断开延时定时器。
- 计数器：CTU --- 加计数器；CTD --- 减计数器；CTUD --- 加/减计数器。
- 双稳态触发器：SR --- SR 触发器；RS --- RS 触发器。
- 边沿检测： R_TRIG --- 上升沿检测； F_TRIG --- 下降沿检测。

3.6.5.2 FB 的实例化

在 IEC61131-3 中，“FB 实例化”的概念特别重要。

所谓实例化，就是用户在变量声明部分通过指定变量名称及其数据类型来建立一个变量。

FB 也需要如同变量那样首先进行实例化。在程序中不允许直接调用 FB，而只能调用 FB 的实例。形象地讲，在程序中不能读写一个数据类型（比如 INT 型），而只能读写声明为该数据类型（比如 INT 型）的具体的变量。如下图，在程序中只能调用、访问 T1。

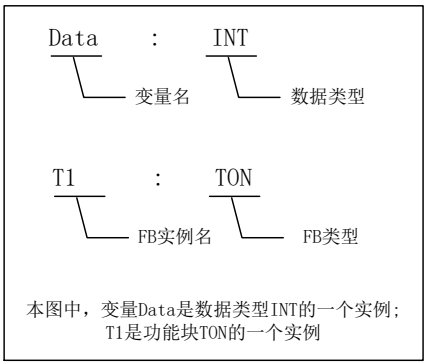


图 3-1 实例化举例

3.6.5.3 FB 实例存储区

Kinco-K 系列在内存中为每一种 FB 类型都分配了一个实例存储区域。当定义了一个 FB 实例时，KincoBuilder 会自动在对应的存储区域内为该实例分配一个独立的存储单元。

下表描述了 Kinco-K 系列的 FB 实例存储区域。

T	
描述	定时器区，可在该区域内分配 TON、TOF、TP 的实例。 用于存储所有定时器实例的状态值和当前计时值。
访问方式	直接访问定时器的状态值、当前计时值
存储权限	可读
其它	不允许掉电保持，不允许强制
C	
描述	计数器区，可在该区域内分配 CTU、CTD、CTUD 的实例。 用于存储所有计数器实例的状态值和当前计数值。
访问方式	直接访问计数器的状态值、计数值
存储权限	可读
其它	允许掉电保持，不允许强制
RS	
描述	RS 触发器区，可在该区域内分配 RS 的实例。 用于存储所有 RS 实例的状态值。
访问方式	直接访问 RS 状态值
存储权限	可读
其它	不允许掉电保持，不允许强制
SR	
描述	SR 触发器区，可在该区域内分配 SR 的实例。 用于存储所有 SR 实例的状态值。
访问方式	直接访问 SR 状态值
存储权限	可读
其它	不允许掉电保持，不允许强制

表 3-6　FB 实例的存储区

3.6.6 FB 实例的命名及使用

FB 的实例遵循“先声明，后使用”的原则。

为了方便用户，在 KincoBuilder 中特意作了如下处理：FB 实例的命名遵循传统 PLC 的常用方式，比如 T0、C3 等。用户不需要手工输入 FB 实例的声明语句，只需在程序中调用合法的功能块实例即可，软件将会在全局变量表中为用户调用的实例自动生成声明语句。

FB 实例存储区的直接寻址方式如下述列表。

- T

直接寻址	格式	Tx
	描述	x：定时器编号，十进制数字。
	数据类型	BOOL --- 定时器的状态值 INT --- 定时器的当前计时值。 Tx 兼具以上两种含义，但用户只需在程序中使用实例名即可，其含义将由软件自动识别。
	示例	T0、T5、T20

- C

直接寻址	格式	Cx
	描述	x：计数器编号，十进制数字。
	数据类型	BOOL --- 计数器的状态值 INT --- 计数器的当前计时值。 Cx 兼具以上两种含义，但用户只需在程序中使用实例名即可，其含义将由软件自动识别。
	示例	C0、C5、C20

- RS

直接寻址	格式	RSx
	描述	x：RS 触发器编号，十进制数字。
	数据类型	BOOL --- RS 触发器的状态值
	示例	RS0、RS5、RS10

- SR

直接寻址	格式	SRx
	描述	x：SR 触发器编号，十进制数字。
	数据类型	BOOL --- SR 触发器的状态值
	示例	SR0、SR5、SR10

3.6.7 FB 实例存储区的范围

Kinco-K 系列 CPU 的内存中为每一种 FB 类型都分配了一个存储区域。当定义了某种 FB 的实例时，KincoBuilder 会自动在该 FB 类型对应的存储区域内为每个实例分配一个独立的存储单元。

下表描述了 Kinco-K 系列 CPU 为每种 FB 分配的实例存储区域。

T	允许数量	256
	范围	T0 --- T255
	时基	T0 --- T3：1ms T4 --- T19：10ms T20 --- T255：100ms
	最大定时时间	32767*时基
C	允许数量	256
	范围	C0 --- C255
	最大计数值	32767
RS	允许数量	32
	范围	RS0 --- RS31
SR	允许数量	32
	范围	SR0 --- SR31

表 3-7　FB 实例存储区的分配

第四章 使用 KincoBuilder 软件的基本功能

本章对 KincoBuilder 软件各部分的功能、使用进行了详细的描述，用户在掌握上一章介绍的基本概念的基础上，通过阅读本章可以快速认识并理解 KincoBuilder 的具体功能以及操作。

LD 编辑器和 IL 编辑器的使用将涉及到 IEC61131-3 标准中的许多语法，在本章中未作介绍，相关内容以及语法将在下一章节进行详细描述。

4.1 KincoBuilder 软件设置

用户在正式使用 KincoBuilder 之前需要对一些软件默认值进行设置，设置结果将保存在当前计算机中，以后 KincoBuilder 在运行时将自动读取并使用这些设置值。执行【工具】→【软件设置…】菜单命令，将弹出“软件设置”对话框。

① 【通用】页面

➢ 工程属性默认值

此处允许对 KincoBuilder 工程的一些属性进行设置。

- **默认编程语言**：选择程序刚创建时的默认编程语言，IL 或者 LD。
- **默认 CPU 型号**：选择工程刚创建时其硬件配置中默认使用的 CPU 型号。

➢ **在线监视时整数显示格式**

选择在线监视时在编辑器中整数数据的显示格式。有如下三种选项：

- 混合：INT、DINT 型以十进制格式，BYTE、WORD、DWORD 型数据以十六进制显示。
- 十进制：所有整数都以十进制格式显示。
- 十六进制：所有整型数据都以十六进制格式显示。

➢ 其它

- **记录日志**

若选中此项，那么 KincoBuilder 运行时会在安装目录下新建立一个 KincoPlcLog 子目录，在这个目录中，每天都会生成一个日志文件，日志中记录了 KincoBuilder 所有的操作，便于发生问题时进行跟踪分析。若操作系统是 Win7 或者 Win8，那么必须以管理员权限运行 Kincobuilder 才会记录日志。日志只记录当天的操作，旧记录会被自动删除。

- **以同步方式打开串口**

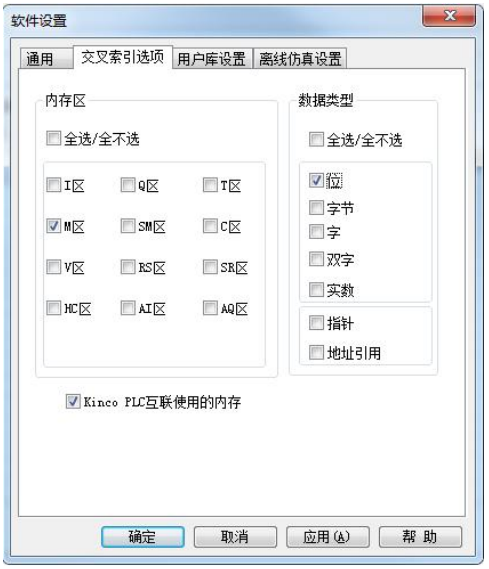
有些用户在 Windows7 或更高版本的操作系统下使用某些 USB 转 RS232 转换器时,可能会出现与 PLC 通信失败的问题。通常这是由该转换器驱动程序的兼容性引起的。

若出现这种问题，那么选中“以同步方式打开串口”，然后单击“确定”按钮退出。以后 KincoBuilder 将以同步方式操作串口，在大多数情况下能够解决这个问题。

- **加快串口通信过程**

当 PC 和 PLC 使用串口通信时此选项被勾选后，KincoBuilder 判断 PC 和 PLC 没有通信上给出提示的时间会缩短，没有勾选的话给出通信失败提示的时长会略微加长一些。

② 【交叉索引选项】页面

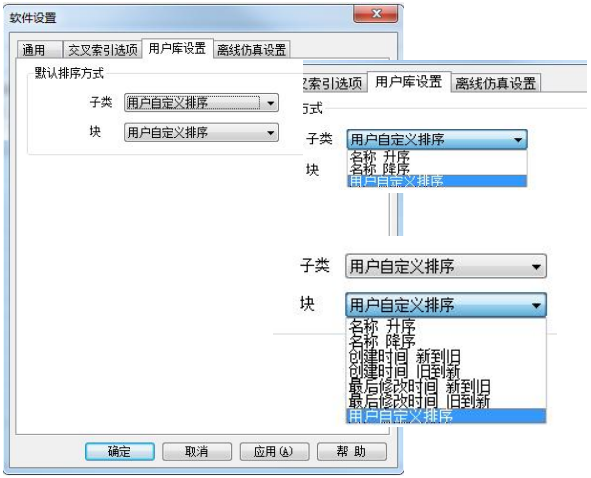


编译后，在交叉索引表中可以列表显示出用户工程中用到的变量及其被使用的位置。

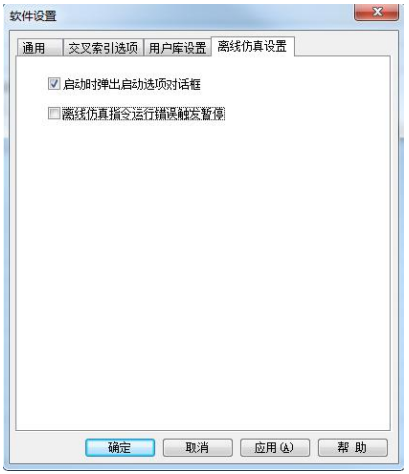
Kincobuilder 默认显示用到的所有变量。

在这个选项页面中，可以选择仅显示工程中用到的某个【内存区】及【数据类型】的变量，便于用户查询。比如上图中的选项，就指定了在交叉列表中只显示用户工程用到的 M 区的位变量。

③ 【用户库设置】页面



④ 【离线仿真设置】页面



4.2 浮动窗口

工程管理器、指令集窗口、信息输出窗口、PLC 模块列表窗口均被设计成浮动窗口，它们可以停靠在 KincoBuilder 主窗口的任意一边。

- 单击浮动窗口右上角的图标，即可让该窗口自动隐藏。在自动隐藏状态下，窗口自动收缩成一个图标并停留在主窗口边缘。此时将鼠标指向该图标并停留片刻，窗口就会自动出现，然后若将鼠标置于窗口之外，它又将收缩于屏幕边缘。
- 在自动隐藏状态下，单击窗口右上角的图标，窗口就会取消隐藏状态并重新停靠至上一次的停靠位置。

4.3 PLC 硬件配置

硬件配置是一个工程中必需的信息，因此建议用户在工程中首先完成硬件配置。

当用户新建一个工程时，KincoBuilder 将自动加入一个默认的 CPU 模块，它的型号是由用户在“软件设置”对话框中指定的**【默认 CPU 型号】**。用户可以按实际需求自行修改所有的配置信息。

硬件配置信息必须下载到 CPU 中以后才会生效。在每次冷启动（重新上电）时，CPU 会首先

检测实际所连模块的信息，并将检测到的实际信息与储存的配置信息进行比较，若用户配置的模块类型与实际连接的模块类型不一致，PLC 就会进入 STOP 状态，否则就进入正常运行状态。

硬件配置窗口的样式如下图。从图中可以看出，窗口可分为两部分：

图 4-2 硬件配置窗口

➢ 模块列表

在硬件配置窗口的上半部分，以表格形式列出了工程中用到的 PLC 模块，我们称之为模块列表。模块列表表达了一个真实的 PLC 控制系统：各个模块在表格中的排列次序应当与实际模块在扩展总线上的连接次序一致。

➢ 模块参数配置窗口

在硬件配置窗口的下半部分，是 PLC 模块列表中选中模块的所有配置参数，我们称之为模块参数配置窗口。

4.3.1 如何进入硬件配置窗口？

有如下两种方式可以进入硬件配置窗口：

- 双击工程管理器中【**资源**】组下的【**PLC 硬件配置**】节点；

- 在【PLC 硬件配置】节点上单击鼠标右键，然后执行弹出菜单的【打开…】命令。

4.3.2 在不同工程中复制和粘贴硬件配置信息

在 KincoBuilder 中，允许用户在不同工程中复制、粘贴【硬件配置】信息。注意，这个功能操作的是各种配置信息，比如 CANOpen 配置、通信口配置等，不会复制、粘贴具体的 CPU 型号，因此，在不同的 CPU 型号之间也可以进行操作。另外，待操作的各个工程必须分别使用 KincoBuilder 打开之后才可以进行操作。此功能与梯形图的复制粘贴，指令表的复制粘贴都不冲突，可以同时用。

用户希望移植或者继承旧工程中的 CANOpen 网络配置、通信口配置等信息，那么这个复制、粘贴【硬件配置】功能将会非常有用。

该功能的使用方法有如下 2 种：

- 在【编辑】菜单中执行【复制硬件配置】、【粘贴硬件配置】命令即可。
- 在【工程管理器】树中的【PLC 硬件配置】节点上单击右键，然后执行【复制硬件配置】、【粘贴硬件配置】命令

4.3.3 添加、删除模块

➢ 添加模块

用户可以按照如下步骤来添加一个模块：

① 鼠标单击模块列表中将要加入模块的位置，将焦点置于该行。

若该行已有模块存在，则必须首先删除已存在的模块，然后才能够加入新的模块。

② 在左侧的【PLC 模块列表】窗口中双击要加入的模块即可。

第 1 行（序号为 1 的行）只能加入 CPU 模块，其余各行只能加入扩展 I/O、功能模块。

各个模块之间不允许有空行存在。若有空行，则软件不允许在其后添加模块，并且在保存、编译时会提示错误。

➢ 删除模块

用户可以使用如下任一方法删除一个模块：

- 鼠标单击模块列表中将要删除的模块，然后敲 Delete 键即可删除。
- 鼠标右键单击模块列表中将要删除的模块，执行弹出菜单的**【删除模块】**命令也可。

4.3.4 配置模块参数

Kinco-K 系列的每种模块都提供了多种参数和选项设置以适应不同的具体应用，包括模块的 I/O 地址、模拟量模块各通道的信号类型等等，在 KincoBuilder 软件中允许用户自定义所有的这些参数和选项。

在模块列表中，鼠标单击任何一个模块并选中它，在下边就会出现该模块的参数配置窗口，用户可以在这个模块参数配置窗口中修改该模块的参数。当然，用户也可以在模块列表中使用键盘上的 Up 和 Down 箭头键来改变选中模块的位置。

在模块参数配置窗口的右侧有两个公用的按钮：**【缺省值】**、**【取消】**。

- **【缺省值】**：单击此按钮，将会取消当前页面用户的输入，软件为本模块自动分配参数。
- **【取消】**：单击此按钮，将会取消当前页面用户的输入，恢复本模块原有的配置。

注意：各模块在同一内存区域（I、Q、AI 或者 AQ）内的地址不允许重叠！

4.3.4.1 CPU 参数配置

①【I/O 设置】页面

在此页面中可以完成 CPU 本体上 I/O 点的相关配置。如下图。

➢ 输入区：用于配置 CPU 本体上 DI 点的参数。

- **起始地址**：DI 部分在 I 区中所占用地址的起始字节，固定为 0。
- **输入滤波**：定义 DI 信号采样时所需的滤波时间，单位 ms。每 4 个点被分为一组进行定义。来自于现场的 DI 信号至少要保持所设定的滤波时间，然后 CPU 才会认为该输入有效并更新相应的 DI 映像区，否则 CPU 对该输入不响应。这样有助于滤除输入噪声，增强系统的抗干扰能力。滤波时间越短，CPU 对该输入的响应就越快。在实际应用中，用户应根据现场的具体情况来设定滤波时间。所有的 DI 点默认是不滤波的。

➢ 输出区：用于配置 CPU 本体集成的 DO 点的参数。

- **起始地址**：DO 部分在 Q 区中所占用地址的起始字节，固定为 0。
- **输出保持**：设置当 CPU 处于 STOP 状态时各 DO 点的输出状态。

若某 DO 点对应的复选框被选中，则当 CPU 处于 STOP 时，该点输出为 1。

若某 DO 点对应的复选框没有被选中，则当 CPU 处于 STOP 时，该点输出为 0。

此项功能对于 CPU 停机后所需要的安全连锁非常有意义。

②【通信设置】页面

用于配置 CPU 本体所带串行通信口的参数。如下图。

不同系列、型号的 CPU 本体提供的串口类型和数量各不相同，当用户添加了某个 CPU 后，在通信设置页面中会自动对本型号 CPU 具有的串口类型及数量进行调整。比如用户添加了 KS105-16DT，它有 1 个 RS232 口（PORT0）和 1 个 RS485 通信口（PORT1），所以在通信设置页面

中允许用户配置“PORT0（RS232）”和“PORT1（RS485）”这 2 个通信口的参数，而其它口的参数全部变灰，表示不能操作。

➢ PORT0

在文字“PORT0”后面的括号里有“RS232”，表示 PORT0 是一个 RS232 通信口。

- PLC **站号**：为 PORT0 指定一个唯一的站号。该站号是作为 Modbus RTU 从站时的站号。
- **波特率**：选择波特率：1200、2400、4800、9600、19200、38400、57600、115200。
- **奇偶校验**：选择奇偶校验方式：无校验、奇校验、偶校验。
- **数据位**：选择数据位：8。
- **停止位**：选择停止位：1。

➢ PORT1

在文字“PORT1”后面的括号里有“RS485”，表示 PORT1 是一个 RS485 通信口。

- PLC **站号**：为 PORT0 指定一个唯一的站号。该站号是作为 Modbus RTU 从站时的站号。
- **波特率**：选择波特率：1200、2400、4800、9600、19200、38400、57600、115200。
- **奇偶校验**：选择奇偶校验方式：无校验、奇校验、偶校验。
- **数据位**：选择数据位：8。
- **停止位**：选择停止位：1。
- **作为 Modbus 主站**：若选中此项，那么该 PORT 口就作为 Modbus RTU 的主站进行通信，同时上面的【PLC 站号】就会无效。
- **超 时**：设置 Modbus 主站通信的超时时间，单位 ms。
- **重 试**：当 Modbus 主站收到从站错误的应答后，继续重新尝试通信的次数。

当主站发出一个命令后，在下述情况下会产生通信错误：

- 在定义的超时时间内没有收到从站的应答，则会产生一个通信超时错误；
- 主站收到了从站错误的应答，则会重新尝试通信，最多重新发送“**重试**”次命令。若最后一次仍没有收到从站正确的应答，则主站继续等待超时时间后产生一个通信超时错误。

③【**数据保持**】页面

在此页面中配置数据保持区域。当 CPU 掉电时，数据保持区域中的数据将由后备电池供电来得到保护，以供再次上电时使用。在常温下保持的时间不低于 3 个月。

注意：“初始化数据表”、“数据保持”以及“数据备份”功能保持的内存区域要避免重叠。因为这些数据均在上电之后进入主循环之前进行恢复，其次序是：恢复“数据保持”中定义的内存数据、“初始化数据表”中定义的内存区域赋初始值、恢复用户使用指令永久保存的数据。

数据保持页面如下图。

图 4-5 【数据保持】参数配置页面

总共可以配置 4 个互相独立的数据保持区域。

- **数据区**：指定被保护的数据区所在的内存区，有 V 区、C 区两个选项。

  对于计数器（C 区），只有其当前计数值可以保持。

- **起始地址**：指定该保持区域的起始字节地址或者计数器编号。
- **长度**：指定该保持区域的长度，单位：字节或者是计数器的个数。

如图 4-5 中，区域 1（VB0 - VB9）、区域 2（VB100 - VB109）、区域 3（C0 - C9）和区域 4（C20 - C29）中的数据在 CPU 掉电时将会保持。

④ 【**本体 AI/AO**】页面

某些型号的 CPU 模块本体集成了 AI 通道，各通道的映像区地址分别固定为 AIW0、AIW2、AIW4 等等，依次类推。每通道的采样转换速度约为 30 次/秒。

另外，有的 CPU 本体还集成了 AO 通道，各通道的映像区地址分别固定为 AQW0、AQW2 等等，依次类推。每通道的转换速度约为 30 次/秒。

这些集成的 AI、AO 点需要在【PLC 硬件配置】硬件配置中来选择各通道的信号形式和滤波方式等参数，配置界面如下：

AI 的参数说明请参阅 4.3.4.4 AI 模块的参数配置，AI 的参数说明请参阅 4.3.4.5 AO 模块的参数配置。

⑤ **【CANOpen 主站】**页面（部分经济型 CPU 不具备此功能）

关于 CANOpen 主站功能见 10.5.4 CANOpen 主站功能章节介绍。

⑥ **【其他】**页面

1.因为 KPLC 在不断升级，各 CPU 的数据备份长度不完全相同，为方便程序移植，兼容之前的 K3、K5 系列等，那么需要在【PLC 硬件配置】中 CPU 模块的【其它】页面进行设置相应的永久存储区，如下图：

- 【兼容 K3 的永久存储设置】-都支持

  选中此项则表示 VB3648-3092 将作为数据备份区，该区域中的数据会自动写入永久存储器中。

- 【兼容 K5 的永久存储设置】-除 K3 外都支持

  选中此项则表示 VB3648-4095 将作为数据备份区，该区域中的数据会自动写入永久存储器中。

- 【兼容 K6 的永久存储设置】-只有 K6、KS101M、K209M 支持

  选中此项则表示 VB15360-16383 将作为数据备份区，该区域中的数据会自动写入永久存储器中。

- 【上面所有的区域全部自动永久存储】-只有 K6 支持

  选中此项则表示 VB3648-4095、VB15360-16383 同时生效作为数据备份区，该区域中的数据会自动写入永久存储器中。

- 【上面所有的区域全部不永久存储】-只有 K6 支持

  选中此项则表示不存在永久存储区，全部 V 区数据都不会写入永久存储器中。

2.将整个用户工程备份到 PLC 的永久存储中。

用户新建立的工程，默认都会选中此项。这个功能会将完整的用户工程（包括全局变量定义、各种注释等等）全部都下载至程序存储器中保存。此功能会占用一部分程序存储器空间，如果用户的程序很大，以至于超出了程序存储器的大小，那么 PLC 会自动忽略此选项。

4.3.4.2 DI 模块的参数配置

DI 模块的参数配置非常简单，如下图所示。

图 4-6　DI 模块参数配置

➢　地址

• **起始地址**：指定该模块在 I 区中占用地址空间的起始字节地址。

该模块所有通道的地址都由这个“起始地址”决定。

• **长度**：该模块占用地址空间的长度。这是一个固定值，取决于模块上 DI 通道的数量。

如图 4-6，该模块具有 8 个 DI 通道，模块地址是%IB2，它的通道的地址是 %I2.0 - %I2.7。

4.3.4.3 DO 模块的参数配置

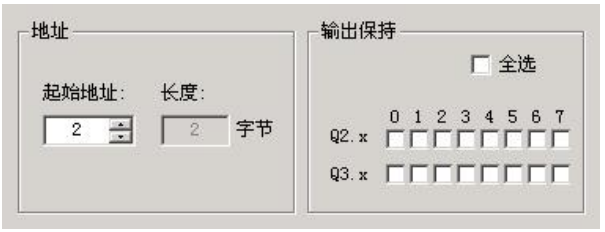


图 4-7　DO 模块参数配置

➢　地址

• **起始地址**：指定该模块在 Q 区中占用地址空间的起始字节地址。

• **长度**：该模块占用地址空间的长度。这是一个固定值，取决于模块上 DO 通道的数量。

如图 4-7,该模块具有 16 个 DO 通道,模块的起始地址是%QB2,它的通道的地址是%Q2.0 - %Q3.7。

➢ 输出保持

设置当 CPU 处于 STOP 状态时该模块各 DO 点的输出状态。

若某 DO 点对应的复选框被选中，则当 CPU 处于 STOP 时，该点输出为 1；若某 DO 点对应的复选框没有被选中，则当 CPU 处于 STOP 时，该点输出为 0。输出点在 CPU 处于 STOP 时的缺省状态是输出为 0。

此项功能对于 CPU 停机后所需要的安全连锁非常有意义。

4.3.4.4 AI 模块的参数配置

图 4-8 AI 模块参数配置

➢ 地址

• **起始地址**：指定该模块在 AI 区中占用地址空间的起始字节地址（也就是第一个通道的地址）。每个 AI 点在 AI 区占用 2 个字节。因此，该地址必须为偶数。

• **长度**：该模块占用地址空间的长度。这是一个固定值，取决于模块上 AI 通道的数量。

如图 4-8，模块的起始地址被指定为%AIW0，该模块具有 4 个 AI 通道，因此它的 4 个通道的地址依次是%AIW0、%AIW2、%AIW4、%AIW6。

➢ 通道设置

• **信号形式**：为各个通道选择输入信号的形式，如 4-20mA、1-5V 等。采样值将在 CPU 内自动进

行线性转换，关于数据转换格式请参见硬件手册中“测量范围和测量值表示格式”相关内容。

- **滤波方式**：为各个通道选择软件滤波器。

对于变化较快的模拟量信号使用滤波器可以让测量值变得比较稳定。

注意：若系统需要对某 AI 信号快速响应，那么就不应该启用该点的软件滤波器。

软件滤波器的输入采样均采用了滑动方式。软件滤波器有如下选项：

- ✓ *无* --- 不启用软件滤波器。
- ✓ *算术平均* --- 对一定数量的信号采样值取算术平均值。
- ✓ *中值平均* --- 将一定数量的信号采样值去掉最大、最小值后，对剩下的数取平均值。

4.3.4.5 AO 模块的参数配置

图 4-9 AO 模块参数配置

➢ 地址

- **起始地址**：指定该模块在 AQ 区中占用地址空间的起始字节地址（也就是第一个通道的地址）。

每个 AO 点在 AQ 区占用 2 个字节。因此，地址必须为偶数。

- **长度**：该模块占用地址空间的长度。这是一个固定值，取决于模块上 AO 通道的数量。

如图 4-9，模块的起始地址被指定为%AQW0，该模块具有 2 个 AO 通道，那么这 2 个通道的地址依次是%AQW0、%AQW2。

➢ 通道设置

- **信号形式**：为各个通道选择所输出信号形式，比如 4-20mA、1-5V 等。关于各种信号的输出值

表示格式的说明请参见硬件手册中“输出范围和输出值表示格式”相关内容。

- **停机保持**：指定当 CPU 处于 STOP 状态时，该点是否保持原来的输出。

若该点对应的复选框被选中，则该点采用停机保持方式。

若该点对应的复选框没有被选中，则该点不采用停机保持方式。

- **停机输出值**：若该点设置为停机保持方式，则在此设置停机时该点的输出值。此处应当输入经过线性转换之后的数值而非实际的信号值。例如，若用户选择信号形式为〔1,5〕V，停机时希望输出 1V，则应当在此处输入 1000。

4.4 初始化数据表

在初始化数据表中用户可以为 V 区中 BYTE、WORD、DWORD、INT、DINT、REAL 类型的数据指定初始值。在 CPU 上电时进入主循环之前，初始化数据表被处理一次，用户指定的初始值被赋给相应的地址。初始化数据表样式如下图：

初始化数据表

	起始地址	初始值	初始值	初始值	初始值
1	%VB0	B#1	B#2		
2	%VW10	2	3	4	
3	%VD100	DI#100	DI#200	DI#2000	DW#2456
4	%VD3840	3.45			
5					

图 4-10　初始化数据表

注意：“初始化数据表”、“数据保持”以及“数据备份”功能保持的内存区域要避免重叠。因为这些数据均在上电之后进入主循环之前进行恢复，其次序是：恢复“数据保持”中定义的内存数据、“初始化数据表”中定义的内存区域赋初始值、恢复用户使用指令永久保存的数据。

4.4.1 如何进入初始化数据表？

有如下两种方式可以进入“初始化数据表”窗口：

- 双击工程管理器中【**程序**】组下的【**初始化数据表**】节点；

• 在工程管理器中【**程序**】组下的【**初始化数据表**】节点上单击鼠标右键，然后执行弹出的【**打开**...】菜单命令。

4.4.2 在表格单元中输入数据

表格单元得到焦点则会自动进入编辑状态，或者鼠标单击某表格单元也可让其进入编辑状态。若某表格单元失去焦点，则其输入的数据将得到确认。

使用上、下、左、右箭头键可以改变具有焦点的表格单元。

使用 PageUp、PageDown 键可以翻页。

若输入的数据有错误，将会自动变成红色进行提示。错误的数据在保存时会被忽略掉。

4.4.3 定义初始化数据

初始化数据表中共有 5 列：1 个〔起始地址〕列和 4 个〔初始值〕列。

用户可以按如下步骤定义初始化数据：

① 在〔起始地址〕列中输入一个直接地址；

② 在〔初始值〕列中输入一个或者多个数值。若输入多个数值，则 KincoBuilder 将它们隐含地分配给从起始地址开始的连续多个同类型的直接地址变量。

如图 4-10，第 1 行的含义是为%VB0、%VB1 分别赋初始值 B#1、B#2；第 2 行的含义是为%VW10、%VW12、%VW14 分别赋初始值 2、3、4；第 3 行的含义是为%VD100、%VD104、%VD108、%VD112 分别赋初始值 DI#100、DI#200、DI#2000、DW#2456。

4.4.4 编辑初始化数据表

➢ 排序

单击〔起始地址〕列的列头即可让表格按照起始地址中字母的升序或者降序进行排序。

➢ 右键菜单

在表中任意一个单元单击鼠标右键，将会弹出如下菜单：

- **删除当前行**：删除焦点所在的行。
- **插入一行（上）**：在焦点所在行的上一行的位置插入一个新的空行。
- **插入一行（下）**：在焦点所在行的下一行的位置插入一个新的空行。

另外，使用粘贴命令时请注意：不同类型的表格之间不允许粘贴，比如全局变量表与初始化数据表；不同的列之间不允许粘贴。

4.5 全局变量表

在全局变量表中用户可以方便地完成全局变量的定义，在 IEC61131-3 中，全局变量的关键字是 VAR_GLOBAL。全局变量可以在所有的 POU 内读写。全局变量表窗口分为**【全局变量】**和**【功能块实例】**两个页面。

➢ **【全局变量】**页面

用于定义全局变量，也就是直接存取 PLC 内存地址的符号变量。

全局变量在程序中可以等效地代替对应的 PLC 直接内存地址使用，从而能够保证用户程序具有良好的可读性。每一个内存地址只允许赋予一个符号变量名，同样地，每一个符号变量名只允许有一个对应的内存地址。

符号变量的命名规则请参见 3.3.1 标识符的定义部分。

关于全局变量的更多信息请参见 3.5 变量。

在本书中，“全局变量表”通常就指这个页面。该页面样式如下图所示。


VAR_GLOBAL

	全局变量名	地 址	数据类型	注 释
1	MI_Guzhangl	%M0.2	BOOL	1#泵故障
2	MI_Jianxiul	%M0.5	BOOL	1#泵检修
3	MI_Yingji1	%M1.0	BOOL	1#消防应急
4	MQ_JY1	%M2.0	BOOL	1# 泵降压运行
5	MQ_QY1	%M2.4	BOOL	1#泵全压运行
6	T_JQ	%VW4	INT	降压后的延时
7	T_Beng	%VW6	INT	泵之间的启动延时
8				

全局变量 功能块实例

图 4-11　全局变量表页面

➢　**【功能块实例】**页面

在 3.6.6FB 实例的命名及使用中提到过，为了方便用户的使用，功能块实例的定义是由 KincoBuilder 自动完成的，因此在**【功能块实例】**页面中，所有的表格单元都是不可编辑的，其信息仅供用户进行参考。页面如下图


VAR_GLOBAL

	FB实例名称	FB类型	FB实例定义位置
1	T5	TON	SBR_运行
2	T6	TON	SBR_运行
3	T7	TON	SBR_运行
4	T8	TON	SBR_运行
5	T9	TON	SBR_运行

全局变量 功能块实例

图 4-12　**【功能块实例】**页面

4.5.1 如何进入全局变量表？

有如下三种方式可以进入全局变量表窗口：

•双击工程管理器中【配置】--**【资源】**组下的**【全局变量表】**节点；

• 在工程管理器中【配置】--**【资源】**组下的**【全局变量表】**节点上单击鼠标右键，然后执行弹出的**【打开...】**菜单命令。

• 执行**【工程】→【全局变量表】**菜单命令。

4.5.2 声明全局变量

全局变量表中共有 4 列：〔全局变量名〕、〔地址〕、〔数据类型〕和〔注释〕列。

用户可以按照如下步骤来声明一个全局变量：

① 进入全局变量表窗口并切换至【**全局变量**】页面；

② 在〔全局变量名〕列中输入一个变量名称并确认；

③ 在〔地址〕列中输入一个直接内存地址并确认；

④ 在〔数据类型〕列中选择一个为变量选择一个数据类型并确认；

⑤（可选）在〔注释〕列中为这个全局变量输入注释。

如图 4-11 的第 1 行的含义是：为地址“%M0.2”定义一个符号名称为“MI_Guzhang1”，其数据类型为“BOOL”。在程序中，“%M0.2”与“MI_Guzhang1”是等价的。

- **编辑全局变量表**

编辑办法：（如：M0.2 是 1#泵故障信号）

1、打开全局变量表，编辑变量名称：MI_Guzhang1

VAR_GLOBAL

	全局变量名	地 址	数据类型	注 释
1	MI_Guzhang1	%M0.2	BOOL	电机故障指示
2				
3				
4				
5				
6				

2、地址输入 M0.2，数据类型自动识别 BOOL

3、编辑注释是为了更清晰明了该寄存器的含义。光标放入对应的输入栏进行编辑。

- **修改全局变量名**

- 菜单栏【工程】-【显示全局变量名】，先将程序中的变量名切换寄存器状态
- 关闭所有程序窗口，修改完之后再打开菜单栏【工程】-【显示全局变量名】

全局变量表可以导出到 CSV 文件，也可以从 CSV 文件导入。右键选择进行操作。

4.6 交叉索引表

交叉索引表用于分析当前工程中用到的所有地址变量并列表显示详细信息。使用交叉索引表便于用户对当前的工程进行统计、分析。

交叉索引表中的信息在第一次编译之后才能生成，并在以后的编译过程中自动刷新。

交叉索引表如下图所示：

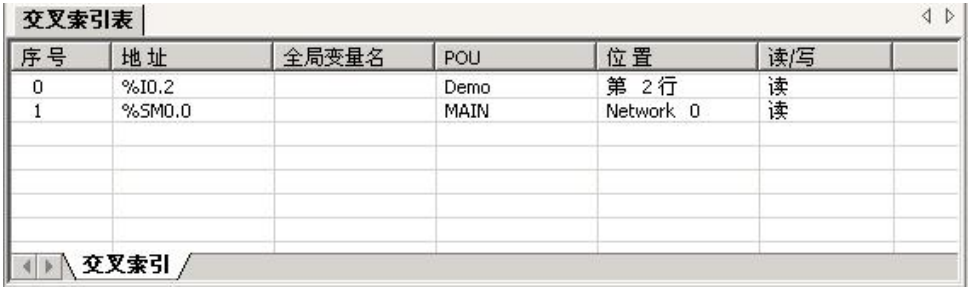


图 4-13　交叉索引表

- 〔地址〕：显示了当前工程中用到的所有内存地址。
- 〔全局变量名〕：显示本行〔地址〕所对应的全局变量名。
- 〔POU〕：显示本行〔地址〕所在的 POU 名称。
- 〔位置〕：指明本行〔地址〕在〔POU〕中的具体位置。

若 POU 用 IL 编写，则显示的是行号；若用 LD 编写，则显示的是网络号。

- 〔读/写〕：指明本行〔地址〕在所处的〔位置〕上被进行了读或者是写操作。

如图 4-13，表格中第一行的意思是：在本工程 Demo 程序的第 2 行用到了一次%I0.2，此处对%I0.2 进行的是读操作。

4.6.1 如何进入交叉索引表？

有如下三种方式可以进入交叉索引表窗口：

- 执行【**工程**】→【**交叉索引表**】菜单命令；
- 鼠标单击工具栏上的  图标。

4.6.2 在交叉索引表中进行操作

➢ 右键菜单

在表格任何一行上单击鼠标右键，将弹出如下右键菜单：

- **刷新**：刷新显示交叉索引数据。
- **变量定位**：打开本行的〔POU〕并定位到〔地址〕所在的具体〔位置〕上。
- **过滤**：弹出【软件设置】对话框中的【交叉索引选项】页面，用户可以选择仅显示工程中用到的某个内存区且数据类型相符的变量。

4.7 变量状态表

用户可以利用变量状态表来对 PLC 中的的内存变量进行在线监视和强制。在变量状态表中，提供了内存监视表和变量状态表两个页面。

➢ 内存监视表

用于监测 PLC 的任意内存地址。

内存监视表样式如下图：

表中共分如下：

- 〔内存地址〕：输入被监测的内存区域的起始地址。
- 〔监视长度〕：输入从“内存地址”开始要监测的数据个数，最大允许个数为 150。

  表格中每行最多显示 3 个数据，若输入的长度超过 3，那么软件会自动分行显示。
- 〔显示格式〕：选择内存值的显示格式，包括十进制、16 进制。
- 〔内存值〕 ：显示所监视的内存值。每行最多显示 3 列，其中第一列是起始内存的值。

  若监视的是位内存，那么只显示 TRUE、FALSE，而“显示格式”就无效了。

➢ 变量状态表

变量状态表用于对当前工程中用到的任意地址变量进行在线监视和强制。

变量状态表的样式如下图所示。

变量状态表

	地 址	全局变量名	显示格式	当前值	强制值
1	%M0.0	MI_Gaoshuiwei	BOOL	2#0	
2	%M0.1	MI_Dishuiwei	BOOL	2#0	
3	%VW4	T_JQ	无符号整数	W#2000	
4	%VW6	T_Beng	有符号整数	300	
5					
6					

图 4-14 变量状态表

表中共分如下五列：

- 〔地址〕 ：输入将要被监测和强制的直接地址。
- 〔全局变量名〕 ：显示本行〔地址〕所对应的全局变量名。
- 〔显示格式〕 ：选择当前值和强制值的数据显示格式。

  可选的格式有 BOOL、REAL、有符号数、无符号数、二进制、16 进制等。
- 〔当前值〕 ：在线监视时将显示监测到的本行〔地址〕的值。

  若复选框处于选中状态则表示本行的地址正处于强制状态。
- 〔强制值〕 ：在线监视时可以在此输入本行〔地址〕将要被强制为的值。

注意：为了提高效率，在变量状态表只允许对当前工程中用到的变量进行监测和强制。若用户输入了未被用到的变量，KincoBuilder 虽然不提示错误，但当前值、强制值均不会起作用。

4.7.1 如何进入变量状态表？

有如下三种方式可以进入变量状态表窗口：

- 双击工程管理器中的**【变量状态表】**节点；
- 在工程管理器中的**【变量状态表】**节点上单击鼠标右键，然后执行弹出的**【打开...】**菜单命令；
- 执行**【调试】→【变量状态表】**菜单命令。

4.7.2 监视变量的值

用户可以按如下步骤来利用变量状态表监视用户程序中用到的变量的值：

① 在〔地址〕列中输入要监视的直接内存地址；

② 使用如下方法之一进入在线监视状态：

- 执行【**调试**】->【**在线监视**】菜单命令；
- 单击工具栏上的  图标；
- 使用快捷键 **F6**

③ 用户可以在任何时候改变监视值的〔显示格式〕。

4.7.3 关于强制功能

用户可以使用强制功能来强制修改 PLC 中 I、Q、M、V、AI、AQ 区内的变量值，其中 I、Q、M 和 V 区中的变量可以按位、字节、字或者双字方式进行强制，AI、AQ 区中的变量可以按字方式进行强制。当 CPU 冷启动（重新上电）后，所有变量的强制状态都会被取消。

Kinco-K 系列 PLC 允许同时强制最多 32 个变量。立即指令不允许强制。

在扫描周期中的某一时刻，一个变量的取值会存在如下可能：外部的输入信号（I、AI）或者用户程序的执行结果（Q、AQ、M、V），用户指定的强制值。因此规定了如下变量取值的原则：

- 对于 M、V 区中的变量，强制值与程序执行结果处于同一优先级：若用户进行了强制，则在每个扫描周期的开始，强制值将会生效，之后程序执行结果将会生效。
- 对于 I、AI 区中的变量，强制值优先于外部信号输入值。若用户进行了强制，则在扫描过程中，强制值将会生效。
- 对于 Q、AQ 区中的变量，在扫描过程中以程序执行结果优先，但在扫描周期最后的输出任务中将会以强制值优先。

4.7.4 右键菜单

- **强制**：将〔强制值〕写入 PLC 内的直接内存〔地址〕中。
- **取消强制**：取消针对单击的行中 PLC 直接〔地址〕的强制状态。
- **全部强制**：将〔强制值〕列中的全部强制值都写入〔地址〕列中对应的 PLC 直接内存中。
- **全部取消强制**：取消针对〔地址〕列中所有 PLC 直接内存的强制状态。
- **读取全部强制**：读取所连 CPU 中所有被强制的地址及其强制值并显示于变量状态表中。

4.7.5 强制、取消强制

在变量状态表中，用户可以采用如下步骤来对一个内存变量进行强制或者取消强制：

① 在〔地址〕列中输入要强制的直接内存地址。

② （可选）选择数值的〔显示格式〕。当然显示格式也可以随时修改。

③ 在〔强制值〕列中输入想要的强制值。用户可以直接输入十进制的整数，KincoBuilder 会按照〔显示格式〕对数值的格式自动进行调整。

④ 使用如下方法之一进行强制：

- 在本行单击鼠标右键，执行弹出菜单的**【强制】**命令；
- 鼠标单击本行任何一处，然后单击工具栏上的 图标；
- 鼠标单击本行任何一处，然后执行**【调试】→【强制】**菜单命令。
- 使用如下方法之一取消针对本行中 PLC 直接〔地址〕的强制状态：

⑤ 在本行单击鼠标右键，执行弹出菜单的**【取消强制】**命令；

- 鼠标单击本行任何一处，然后单击工具栏上的图标；
- 鼠标单击本行任何一处，然后执行**【调试】→【取消强制】**菜单命令。

变量状态表的编辑方式与初始化数据表一样，请参考初始化数据表中的相关内容。

第五章 使用 KincoBuilder 编写用户程序

本章对 KincoBuilder 中 LD 和 IL 编辑器的功能、使用进行了详细的描述，同时也阐述了 IEC61131-3 标准中关于 LD、IL 语言的相关语法、规定。通过阅读本章，用户可以轻松地使用 KincoBuilder 编写出符合 IEC 标准的应用程序。

针对 PLC 应用程序的编写，IEC61131-3 中规定了 3 种文本化语言和 3 种图形化语言。文本化语言包括：指令表（IL）、结构化文本（ST）、顺序功能图（SFC，文本化版本）；图形化语言包括：梯形图（LD）、功能块图（FBD）、顺序功能图（SFC，图形化版本）。

KincoBuilder 目前支持 IL 语言和 LD 语言编程。在一个工程中，IL 和 LD 语言可以混用，但是在一个 POU 中只允许使用一种语言。用户可以随时执行【工程】→【IL 语言（指令表）】或者【工程】→【LD 语言（梯形图）】菜单命令切换当前程序的语言。注意：任何 LD 程序都可以转换为 IL 程序，但只有按照一定规则编写的 IL 程序才可以转换成 LD 程序。

5.1 IL 编程

5.1.1 IL 的背景

IL 语言是一种低级语言，与汇编语言非常相似，是在借鉴、吸收世界上各著名 PLC 厂商的指令表语言的基础上而形成的一种标准语言。

IL 接近于机器码，因此使用 IL 编写的程序效率更高、更加紧凑。IL 很适合经验丰富的编程人员使用，有时候使用 IL 语言可以解决 LD 等图形化语言难以解决的问题。

5.1.2 IL 的语法规定

5.1.2.1 IL 语句格式

IL 程序面向行，每行语句只能有一条指令或者一个标号。IL 程序中允许有空白行存在。

IL 程序中一个语句的基本格式如下图：

```
标号：
操作符/功能/功能块      操作数（表）      (* 注释 *)
```

- **标号（可选）**

  使用标号的目的就在于实现程序的跳转。标号的命名格式如同变量。

- **操作符/功能/功能块**

  即 PLC 指令。

- **操作数（表）**

  请参见下一章中对于各操作符、功能、功能块的详细说明，其中也包含了相应操作数的描述。

  在操作符、操作数之间至少有一个空格。

- **注释（可选）**

  在所有的语言中，注释的格式都是相同的，由(* *)进行界定。

  每行只允许有一个注释。注释也可以单独占用一行。

  注释不允许嵌套，嵌套的注释编译器将认为是错误。

IL 语句举例如下：

```
(* NETWORK 0 *)
Run:                (* 标号，供跳转时使用 *)
LD      %I1.0
TP      T2, 168     (* 若 I1.0 为 true，启动定时器 T2，T2 被声明为 TP 型 *)
```

5.1.2.2 关于 CR

IL 中提供了一个叫做“Current Result（CR）”的通用累加器，在 CR 中存储了用户程序的当前执行结果。用户程序中的每一行语句执行后，CR 都会被刷新。依据后续语句的不同，CR 值可能作为下一条语句的执行条件，也可能作为下一条语句的操作数之一。

操作符不同，执行之后对 CR 值的影响也不同。下表根据对 CR 值不同的影响将 KincoBuilder 中的操作符进行了初步的分组。更详细的描述请参见下一章对于各指令的介绍。

操作符	分组缩写	对 CR 值的影响
LD，LDN	C	CR 值重新建立
逻辑指令，比较指令等	P	CR 值被更新为操作结果
ST，R，S，JMP，JMPCN，JMPC 等	U	CR 值保持不变

表 5-1　各操作符执行之后对于 CR 的影响

注意：在 IEC61131－3 中并没有完善地定义各种操作符对于 CR 的影响，因此在不同的编程系统中这些定义可能有所不同。

5.1.2.3 网络

在 IL 程序中也存在着网络（Network）的概念，以网络作为基本的段落，一个 POU 的代码部分就是由若干个网络组成。

一个典型的网络由网络标号和网络中的语句这两部分组成。在 KincoBuilder 中，关于网络中的格式有如下规定：

- 在一个网络中可以只有一个语句标号。例如：

```
(* NETWORK 0 *)
MRun:     (* 可以只有一个标号 *)
```

- 在一个网络中可以只有程序语句。

在 5.1.2.2 关于 CR 中，我们将所有指令分为了三组：“C”、“P”、“U”。

网络中的程序必须以“C”组中的指令开始，以“P”、“U”组中的指令结束。

举例如下：

```
(* NETWORK 0 *)
LD  %M3.5  (* 以 LD 指令开始 *)
ST  %Q2.3  (* 以允许的指令结束 *)
```

- 在一个网络中可以有语句标号和程序语句。

网络中的程序必须以语句标号或者“C”组中的指令开始，以“P”、“U”组中的指令结束。

举例如下：

```
    (* NETWORK 0 *)
    MRun:       (* 以语句标号开始 *)
LD %M3.5
    ST %Q2.3   (* 以允许的指令结束 *)
```

5.1.3KincoBuilder 中的 IL 编辑器

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

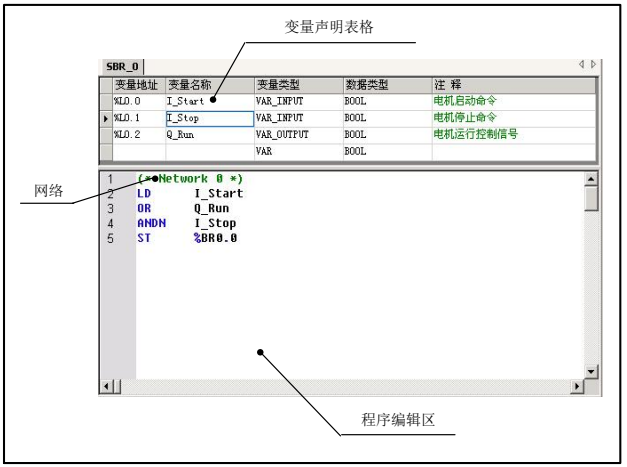


图 5-1　IL 编辑器

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

- 变量声明表格：用于声明该 POU 的输入/输入参数和局部变量。支持右键菜单。
- 程序编辑区：这个区域类似于一个文本编辑器，用户可以在此直接编辑自己的应用程序。

5.1.3.1 IL 程序编辑

➢ 添加一个网络

可以使用如下任何一种方法来加入一个新网络：

- 使用 Ctrl+Q 快捷键。
- 在程序编辑区内单击鼠标右键，执行【**加入新网络**】弹出菜单命令。

➢ 输入 IL 语句

程序编辑区类似于一个文本编辑器，用户可以在此直接输入语句，编辑自己的应用程序。这里支持通常的键盘操作，比如删除（Delete），退格（BackSpace），使用上、下、左、右方向键移动光标等等。

IL 编辑器能够自动地对用户输入的语句进行格式化，并将语句中的关键字用蓝色来显示，注释采用绿色来显示。

在编辑语句时，如果光标切换到其它行，IL 编辑器就会对刚离开的行进行语法检查，如果有语法错误，则会在该行的开头显示一个红色的问号（？）。只有检查正确的语句才会格式化显示。

➢ 选中/删除/复制/剪切/粘贴

在 IL 编辑器中可以使用【**编辑**】菜单中的所有编辑命令，包括全选、复制、剪切、粘贴等。另外，在程序编辑区中单击鼠标右键将会弹出右键菜单，用户也可以使用右键菜单命令进行编辑。

执行【**全选**】命令将会选中编辑区内所有的文本内容。使用鼠标在编辑区中拖动，或者使用上、下、左、右方向键移动光标同时按下 SHIFT 键，将会选中所经过的文本内容。选中的区域将采用黑色作为底色，文字呈高亮状态显示。

使用 Delete 键或者执行【**删除**】命令，可以删除选中的内容。

使用快捷键 Ctrl+C 或者执行【**复制**】命令，可以将选中的内容复制到 Windows 的剪贴板中。被复制的内容可以在任何文本编辑器中粘贴，比如 Windows 记事本、Word 等。

剪切的操作相当于复制并删除选中的内容。使用快捷键 Ctrl+X 或者执行【**剪切**】命令，可以将选中的内容剪切到 Windows 剪贴板中。

复制或者箭切完毕后，将光标定位到想要粘贴的位置，然后使用快捷键 Ctrl+V 或者执行【**粘贴**】命令，即可将剪贴板中的内容粘贴到光标所在的位置。

➢ 查找/替换

IL 编辑器提供了标准的查找和替换命令。

- 查找

使用快捷键 Ctrl+F 或者执行【**编辑**】→【**查找...**】菜单命令，将会弹出如下的查找窗口：

在【查找内容】输入框中输入要查找的字符串，然后单击【查找下一个】按钮即可开始查找，找到的内容会被选中并高亮显示。

其它选项均采用了 Windows 标准的查找窗口中的含义，此处不再赘述。

- 替换

使用快捷键 Ctrl+R 或者执行【**编辑**】→【**替换...**】菜单命令，将会弹出如下的替换窗口：

在【查找内容】输入框中输入要查找的字符串，在【替换为】输入框中输入要替换的字符串，

然后单击【替换】按钮，IL 编辑器将自动查找下一个符合查找内容的字符串并将该字符串替换。单击【全部替换】按钮，IL 编辑器将自动查找当前程序中所有符合查找内容的字符串并全部替换为指定的字符串。

其它选项均采用了 Windows 标准的查找窗口中的含义，此处不再赘述。

5.1.3.2 IL 程序示例

```
(* NETWORK 0 *)
LDN     %M0.0
TON     T0, 1000        (* 依靠 T1 的输出来启动 T0，定时为 1000×1ms *)
ST      %M0.1
(* NETWORK 1 *)
LD      %M0.1
TON     T1, 1000        (* 依靠 T0 的输出来启动 T1，定时为 1000×1ms *)
ST      %M0.0
(* NETWORK 2 *)
LD      %M0.1
ST      %Q0.0           (* 在 Q0.0 输出方波 *)
```

5.1.4 IL 程序转换为 LD 程序

执行【工程】→【LD 语言（梯形图）】菜单命令可以将当前 POU 的语言切换为 LD。

并非所有的 IL 程序都可以转换为 LD 格式，IL 程序要转换成 LD 程序须满足如下条件：

①IL 程序本身没有任何错误；

②IL 程序中的各个网络必须严格符合如下规则，包括：

- 若网络中包含有语句标号，则只允许有一个语句标号，除此之外不能有任何其它指令。
- 若网络中不包含语句标号，则需满足如下条件：
  - 网络必须以“C”组指令（LD 类指令）开始；

- 作为网络开始行的“C”组指令（LD 类指令）在一个网络中只能出现一次；
- 网络中的程序必须以“P”、“U”组指令结束。

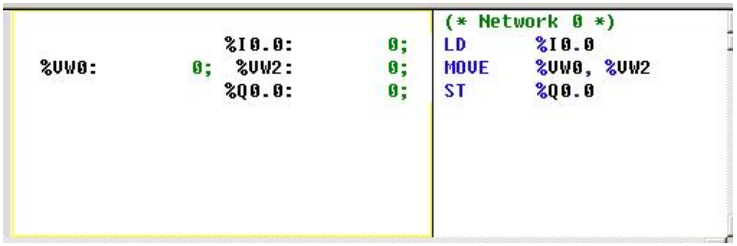
5.1.5 调试和监视程序

5.1.5.1 在线监视 IL 程序

若当前窗口是 IL 编辑器，则可以使用如下任一方法进入在线监视状态。

- 执行【**调试**】→【**在线监视**】菜单命令。
- 单击工具栏上的  图标。
- 使用 F6 快捷键。

在线监视状态下，在原程序编辑区中将分为两栏，右边显示程序，左边显示相应的变量值，中间以一条竖线分割开。将光标置于竖线上，待其形状变为  后，即可拖动该线改变左、右区域的大小。



 **注意：在线监视状态下不允许对程序进行编辑。**

5.1.5.2 强制指定变量

在前面 4.7.3 关于强制功能 中对 KINCO-K 系列的强制功能进行了详细的描述。

在线监视 IL 程序时，用户可以在 IL 编辑器中直接对指定变量进行强制、取消强制等操作：在程序中某个变量上单击鼠标右键，将弹出如下右键菜单（注：若右键单击的是非开关量变量，则菜单中的【**强制为 TRUE**】和【**强制为 FALSE**】命令将是无效的状态）。

- **强制为 TRUE**：将该变量（开关量）的值强制为 1（TRUE）。
- **强制为 FALSE**：将该变量（开关量）的值强制为 0（FALSE）。
- **强制...**：执行该命令后，将弹出如下对话框

在【强制值】输入框中输入要强制的值（相应数据类型的常量），然后单击【强制】按钮即可。

关于常量的表示格式请参阅 3.4 常量。

- **取消强制**：取消对该变量的强制。
- **全部取消强制**：取消对所连 CPU 中当前所有变量的强制。

5.2 LD 编程

5.2.1 LD 的背景

LD（梯形图）语言是 IEC61131-3 标准中规定的五种编程语言之一，是 PLC 编程中被最广泛使用的一种图形化语言。

LD 语言源于机电一体化领域中图形表示的继电器逻辑，用它编写的程序能够与实际的电气操作原理图相对应，非常直观，易于学习和掌握。LD 语言的长处在于布尔逻辑的处理。

5.2.2 网络

在 LD 中也使用“网络（Network）”的概念，以网络作为基本的段落。一个典型的网络由网络标号、注释和网络中的程序这三部分组成。如下图。

一个 LD 网络的边界是位于左、右两侧的电源线（Power Rails）。左侧电源线的状态一直为“1”，右侧电源线的状态没有定义。我们可以对一个 LD 网络作如下形象的描述：电能（能量流）自左侧的电源线沿着连接线流经线上所有的元件（包括触点、线圈、功能、功能块等），这些元件依据自身的逻辑状态，或是中断能量流，或是将电能传输到后继的元件并最终到达右侧的电源线。

5.2.3 标准化图形对象

- 连接

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

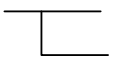
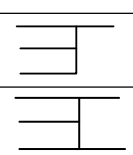
图形对象	名称	描述
	水平连接	可以将其理解为一根理想的导线。 水平连接将线上任一点左侧的值传递到右侧。
	垂直连接 （含有水平连接）	垂直连接将它左侧所有水平连接的值首先进行逻辑“或”运算，然后再将运算结果传递到它右侧所有的水平连接上。
		

表 5-2　LD 中的连接

➢ 触点

触点有两种类型：常开触点和常闭触点。

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

图形对象	名称	描述
变量名 ─┤ ├─	常开接点	若变量值为 TRUE，则触点闭合，它左侧连接的值被传递到右侧连接；否则触点断开，它右侧连接的值为 FALSE。
变量名 ─┤/├─	常闭接点	若变量的值为 TRUE，则触点断开，它右侧连接的值为 FALSE；否则触点闭合，它左侧连接的值被传递到右侧连接。

表 5-3　LD 中的接点

➢ 线圈

线圈是用于给布尔型变量赋值的图形元素。

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

表 5-4　LD 中的线圈

➢ 调用功能和功能块

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

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

功能且 ENO 也将被置为 0。

下图是一个调用功能和功能块的示例。

5.2.4 KincoBuilder 中的 LD 编辑器

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

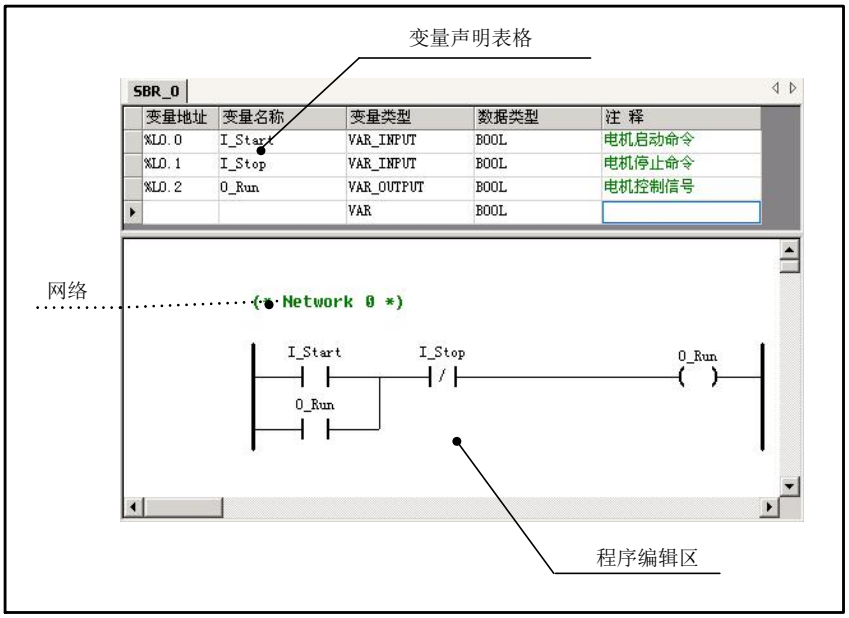


图 5-2 LD 编辑器

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

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

5.2.4.1 LD 程序的限制

程序编辑区被划分为许多个单元格，用户可以通过鼠标单击或使用键盘的方向键来选择单元格，选中的单元格会被虚线矩形框包围以提示当前位置。各种 LD 元件根据自身大小不同，分别占用一个或多个单元格。由于画布的大小限制，因此在每一个 LD 程序中也有如下限制：

- 一个程序（主程序，子程序或者中断程序）中最多允许 200 个网络。
- 一个工程中，子程序和中断程序允许的最大数量分别为 99 个，且所有程序中用到的元件总数量也有限制：KS101M、K209M、K6 最大允许 8192 条指令，K2、K5、KS、MK、KW 等系列最大允许 4096 条指令。
- 每条连接线路径上串联的元件数量限制：对于线圈、触点，则不超过 35 个触点加 1 个线圈；若只有功能/功能快，则建议不超过 12 个块加 1 个线圈加 1 个触点。
- 一个并联关系不允许超过 16 个分支。

5.2.4.2 LD 程序编辑

➢ 标记元件

鼠标左键单击某个元件可以选中并标记该元件。被标记的元件会被虚线矩形框所包围，表示该元件获得了当前的焦点。另外，也可以使用键盘上的方向键来移动焦点，从而改变标记的元件。

被标记的元件将作为基准，用户添加新元件需要根据这个基准位置来进行。

另外，用户可以对被标记的元件进行各种编辑操作，比如修改属性、删除、复制、剪切等。

➢ 添加一个网络

① 在期望添加的位置上标记一个元件作为基准。

若标记的是网络注释，那么将在标记元件所在网络的上方加入新网络；若标记的是其它元件，那么将在标记元件所在网络的下方加入新网络。

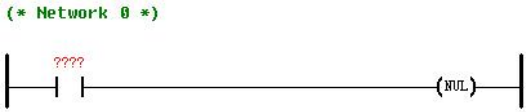
若当前是一个没有任何元件的空程序，那么无需标记元件，将在编辑区的最上部加入新网络。

② 使用如下任一方法在程序中加入一个新网络：

- 执行**【梯形图】→【新网络】**菜单命令。
- 单击工具栏上的 图标。

- 使用 Ctrl+W 快捷键。
- 鼠标右键单击任何一个元件，执行右键菜单中的【**新网络**】命令。

新网络加入后如同下图的形式：



➢ 输入注释

双击网络标号部分，即可弹出“注释”对话框，如下图所示。

用户可以在左侧的输入框中输入该网络的注释，然后单击【**确认**】按钮即可。

➢ 添加一个串联触点

① 在期望添加的位置上标记一个元件作为基准。

② 使用如下任一方法在程序中加入一个串联触点：

- 执行【**梯形图**】→【**左触点**】或者【**右触点**】菜单命令。
- 单击工具栏上的 图标（左触点）或者 图标（右触点）。
- 使用 Ctrl+L（左触点）或者 Ctrl+E（右触点）快捷键。
- 鼠标右键单击基准元件，执行右键菜单中的【**左触点**】或者【**右触点**】命令。
- 鼠标双击窗口右侧“**LD 指令集**”中所需要的触点指令。

➢ 添加一个并联触点

① 在期望添加的位置上标记一个触点元件作为基准。

② 使用如下任一方法在程序中加入一个并联触点：

- 执行【**梯形图**】→【**并联触点**】菜单命令。

- 单击工具栏上的 图标。
- 使用 Ctrl+D 快捷键。
- 鼠标右键单击基准元件，执行右键菜单中的【**并联触点**】命令。

➢ 添加一个并联线圈

① 在期望添加的位置上标记一个线圈元件作为基准。

② 使用如下任一方法在程序中加入一个并联线圈：

- 执行【**梯形图**】→【**并联线圈**】菜单命令。
- 单击工具栏上的 图标。
- 使用 Ctrl+D 快捷键。
- 鼠标右键单击基准元件，执行右键菜单中的【**并联线圈**】命令。
- 鼠标双击窗口右侧“**LD 指令集**”中所需要的线圈指令。

➢ 添加一个串联的功能/功能块

① 在期望添加的位置上标记一个元件作为基准。

② 使用如下任一方法在程序中加入一个串联的功能/功能块：

- 执行【**梯形图**】→【**功能/功能块**】菜单命令；或者单击工具栏上的 图标；或者使用 Ctrl+B 快捷键；或者鼠标右键单击基准元件，执行右键菜单中的【**功能/功能块**】命令。然后都会弹出如下的“功能/功能块/子程序”窗口：

用户选好所需的【功能/功能块】或者【子程序】，然后单击【确认】按钮即可。

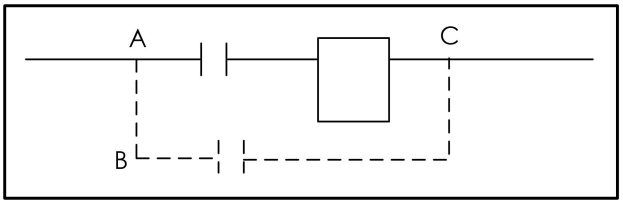
- 鼠标双击窗口右侧“**LD 指令集**”中所需要的功能、功能块指令或者子程序。

➢ 建立一个并联支路

LD 编辑器提供了并联支路编辑模式，从而让用户能够快速创建一个复杂的并联支路。进入并联支路编辑模式后，鼠标指针形状将变为，另外，用户可以随时使用 ESC 键来退出该模式。用户建立一个并联支路的步骤如下：

① 执行【**梯形图**】→【**并联支路**】菜单命令；或者单击工具栏上的图标；或者使用 Ctrl+H 快捷键；或者鼠标右键单击一个元件，执行右键菜单中的【**并联支路**】命令。然后都会进入并联支路编辑模式。

② 使用鼠标继续按如下步骤来画一个并联支路：



1) 在期望的并联起始位置处（如上图中A）单击；

2) 向上或者向下移动鼠标至任意位置（如B）并单击；

3) 移动鼠标至期望的并联结束位置处（如C）并单击，完成一个并联支路的建立。

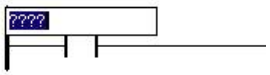
注：上面提到的三个位置并不需要太精确，用户在“大约”的位置处单击即可

➢ 修改元件的参数

加入一个元件后，它的实际参数以红色的问号（????）来表示，这些红色问号必须被修改为适当的常数、直接地址、全局变量或者局部变量名称。

用户可以采用如下任一方法来修改元件的参数：

- 鼠标单击将要修改的元件参数所在的位置，将会在此参数区域出现一个输入框，如下图。



用户直接在此输入框内输入期望的常数、直接地址、全局变量或者局部变量名称，然后按 Enter 键确认即可。LD 编辑器会自动对用户输入的直接地址进行格式化，因此用户在输入时可以不输入百分号（%）。在输入时，也可以按 ESC 键取消当前的修改。

- 标记要修改的元件，然后按 Enter 键，就会在被标记元件的第一个参数的区域出现一个

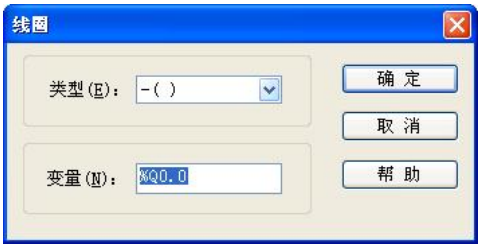
输入框，然后按照上面一个方法进行输入即可。

- 鼠标双击将要修改的元件，将会弹出该元件的属性对话框，

✓ 触点的属性对话框

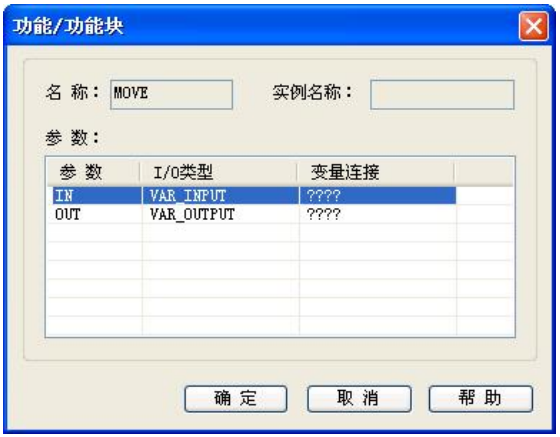
用户可以修改该元件的【**类型**】、连接的【变量】，然后单击【**确定**】按钮即可。

✓ 线圈的属性对话框



用户可以修改该元件的【**类型**】、连接的【变量】，然后单击【**确定**】按钮即可。

✓ 功能/功能块的属性对话框



双击【**参数**】列表中的某项参数，就会在该参数的【**变量连接**】处出现一个输入框，用户直接在此框内输入期望的常数、直接地址、全局变量或者局部变量名称，然后按 Enter 键确认即可。

用户也可以鼠标单击或者使用上、下方向键选择某项参数，然后按 Enter 键进入输入框修改该变量。

KincoBuilder 将对用户的输入进行严格的语法检查，不接受错误的输入，包括非法的变量、数据类型错误、内存区域类型错误等。

所有输入完成后，单击【**确定**】按钮即可。

➢ 删除一个元件

① 标记期望删除的元件。

② 使用如下任一方法删除标记的元件：

- 执行【**梯形图**】→【**删除元件**】菜单命令。
- 单击工具栏上的 ✕ 图标。
- 单击鼠标右键，执行右键菜单中的【**删除元件**】命令。
- 执行【**编辑**】→【**删除**】菜单命令。
- 使用 Delete 键。

➢ 删除一个网络

① 在期望删除的网络中标记任何一个元件。

② 使用如下任一方法删除标记元件所在的网络：

- 执行【**梯形图**】→【**删除网络**】菜单命令。
- 单击工具栏上的图标。
- 单击鼠标右键，执行右键菜单中的【**删除网络**】命令。
- 使用 Ctrl+Del 快捷键。

➢ 选中/删除/复制/剪切/粘贴

在 LD 编辑器中可以使用【**编辑**】菜单中的编辑命令，包括全选、复制、剪切、粘贴等。另外，在程序编辑区中单击鼠标右键将会弹出右键菜单，用户也可以使用右键菜单命令进行编辑。

执行【**全选**】命令将会选中编辑区内所有的内容。使用鼠标在编辑区中拖动，或者使用上、下、左、右方向键移动光标同时按下 SHIFT 键，将会选中所经过的内容。选中的区域将采用黑色作为底色，图形元件和文字呈高亮状态显示。

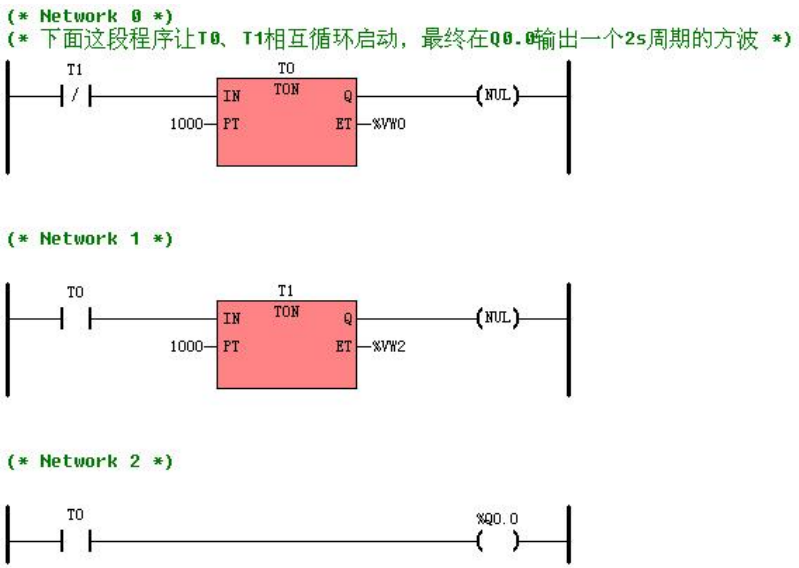
使用 **Delete** 键或者执行【**删除**】命令，可以删除选中的内容。

使用快捷键 **Ctrl+C** 或者执行【**复制**】命令，可以复制选中的内容。被复制的内容可以在 LD 编辑器中进行粘贴。

剪切的操作相当于复制并删除选中的内容。使用快捷键 **Ctrl+X** 或者执行【**剪切**】命令，可以剪切选中的内容。

复制或者箭切完毕后，在想要粘贴的位置标记一个元件，然后使用快捷键 **Ctrl+V** 或者执行【**粘贴**】命令，即可将复制的内容粘贴到相应的位置：若标记的是网络注释，那么将粘贴至标记元件所在网络的上方；若标记的是其它元件，那么将粘贴至标记元件所在网络的下方。

5.2.4.3 LD 程序示例



5.2.5 监视和调试程序

5.2.5.1 在线监视 LD 程序

注意：在线监视状态下不允许对程序进行编辑。

若当前窗口是 LD 编辑器，则可以使用如下任一方法进入在线监视状态：

- 执行【调试】→【在线监视】菜单命令。
- 单击工具栏上的 图标。
- 使用 F6 快捷键。

在线监视状态下，程序中的变量状态显示方式为：

- %I0.0 表示该触点断开，%I0.0 表示该触点闭合。

-  表示该线圈断开，  表示该线圈闭合。
- 程序中非开关量变量的值将直接在该变量的右侧显示出来。

下图是一个在线监视的程序示例：

注：黄色的锁代表强制值

5.2.5.2 强制指定变量

在前面 4.7.3 关于强制功能那一节中对 KINCO-K 系列的强制功能进行了详细的描述。

在线监视 LD 程序时，在程序中某个变量上单击鼠标右键，将弹出如下右键菜单（注：若右键单击的是非开关量变量，则菜单中的**【强制为 TRUE】**和**【强制为 FALSE】**命令将是无效的状态）。

- **强制为 TRUE**：将该变量（开关量）的值强制为 1（TRUE）。
- **强制为 FALSE**：将该变量（开关量）的值强制为 0（FALSE）。
- **强制…**：执行该命令后，将弹出如下对话框：

在【强制值】输入框中输入要强制的值（相应数据类型的常量），然后单击【强制】按钮即

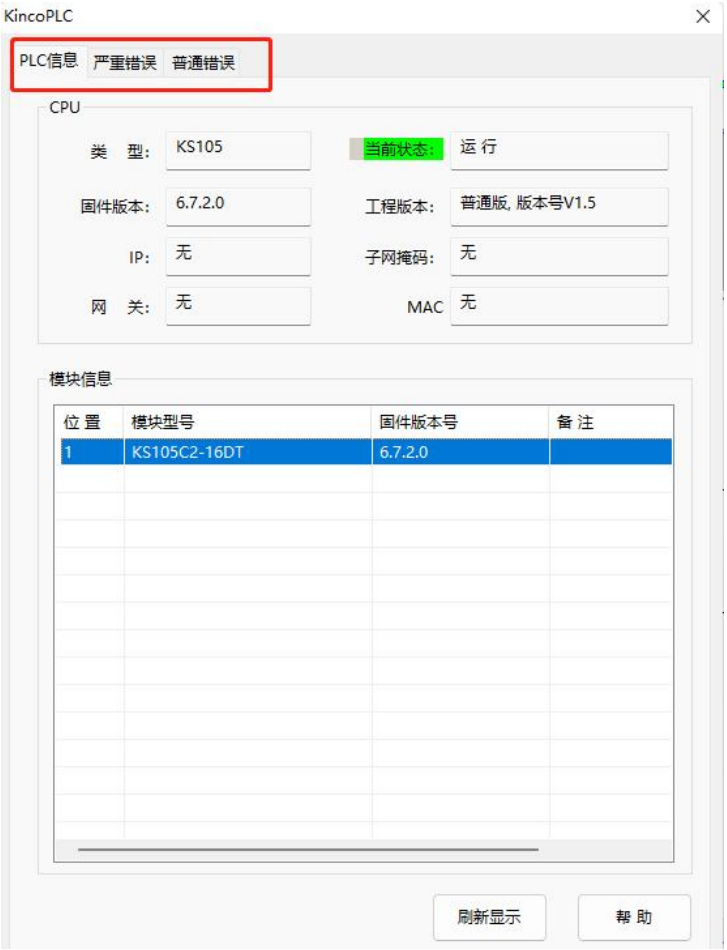
可。

关于常量的表示格式请参阅 3.4 常量。

- **取消强制**：取消对该变量的强制。
- **全部取消强制**：取消对所连 CPU 中当前所有变量的强制。

5.2.6 查看 PLC 错误和故障

点击 KincoBuilder 编程软件的【PLC】菜单下面【PLC 信息】出现如下窗口：



如上图所示包含 3 块内容：PLC 信息、严重错误、普通错误。

PLC 信息：包含 CPU 的类型、当前运行状态、固件版本、IP 地址等。

严重错误：已经发生过的严重错误的错误码和错误描述。

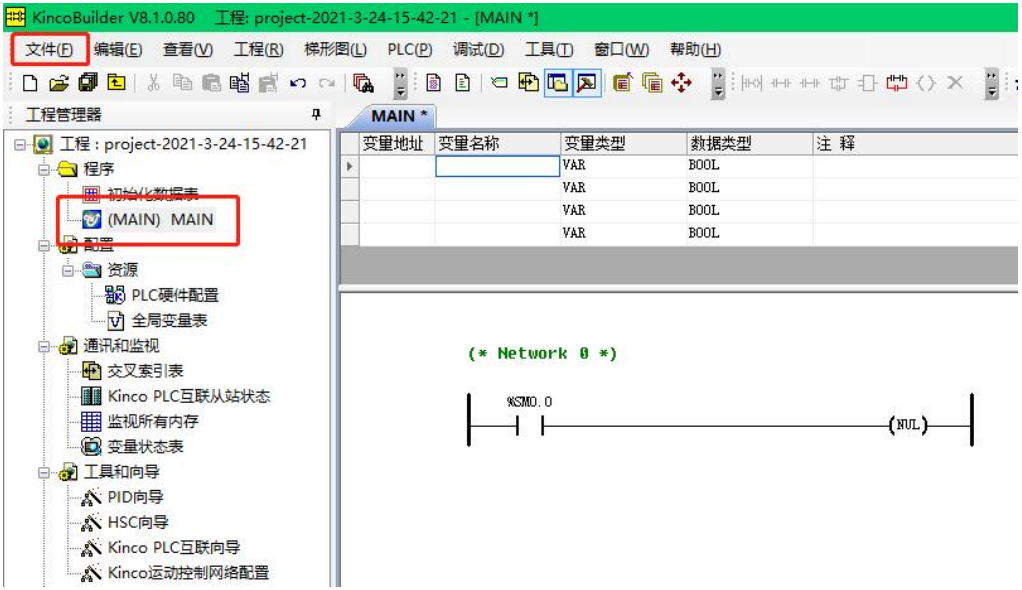
普通错误：已经发生过的普通错误的错误码和错误描述。

关于严重错误、普通错误的介绍请参考第十三章故障处理的介绍。

5.3 如何使用 KincoBuilder 建立主程序、子程序、中断程序

- **建立主程序**

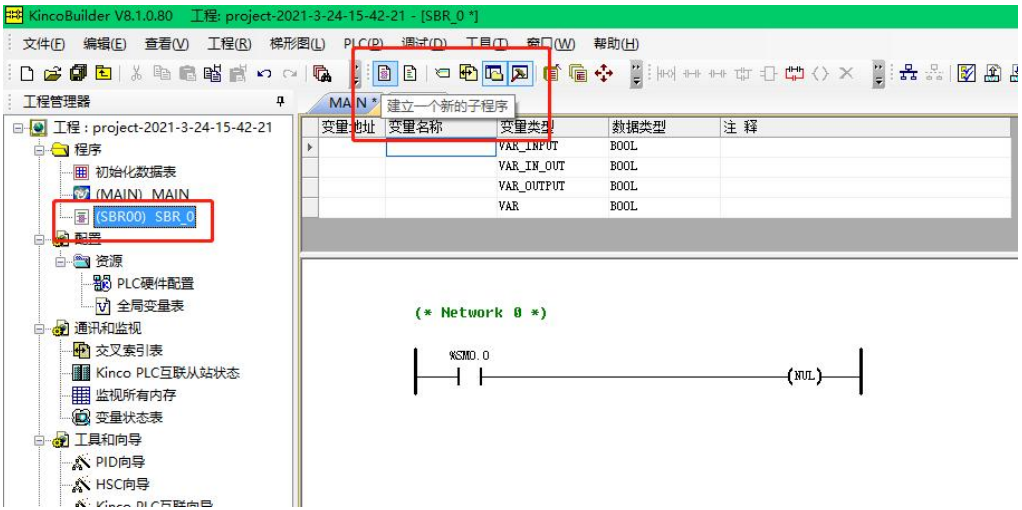
打开 KincoBuilder 软件，点击【文件】菜单下面的【新建工程】，软件会自动生成一个叫做 MAIN 的主程序，工程中所有程序的执行只能由 MAIN 调用，用户可以在 MAIN 主程序中进行编程，如果想要程序结构更有条理性，可以建立不同的子程序由主程序 MAIN 进行调用。



- **建立子程序**

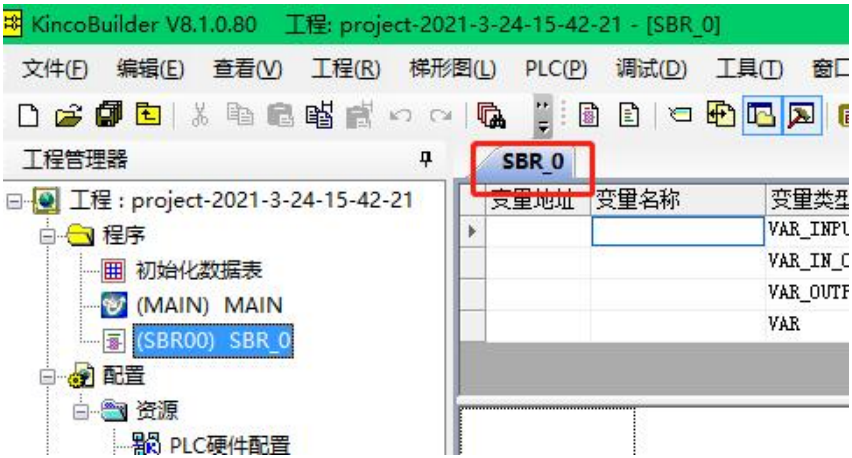
1）新建子程序

点击下图的图标会出现提示“建立一个新的子程序”，点击后自动新建了一个名叫 SBR_0 的子程序。

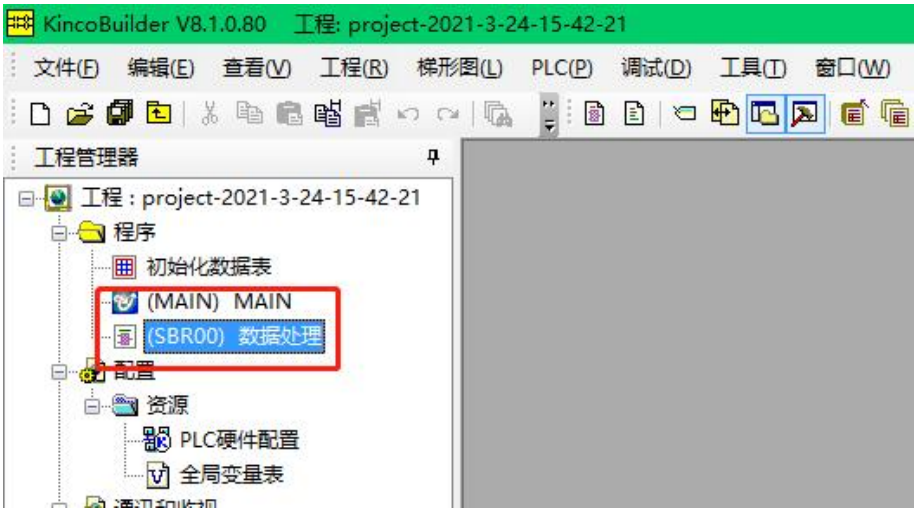


2）修改子程序的名字

右键下图红框处，点击关闭窗口。

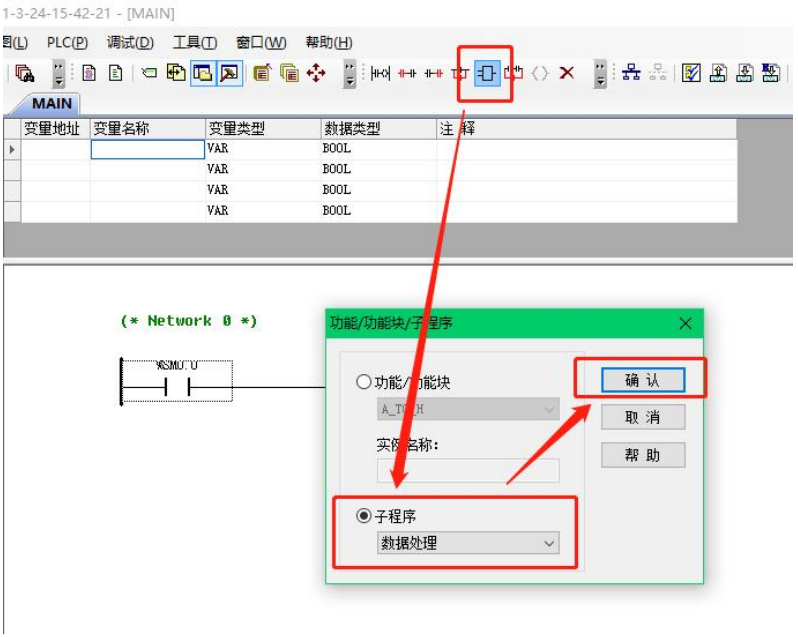


右键下图红框处，点击重命名，输入想要命名的名字即可。

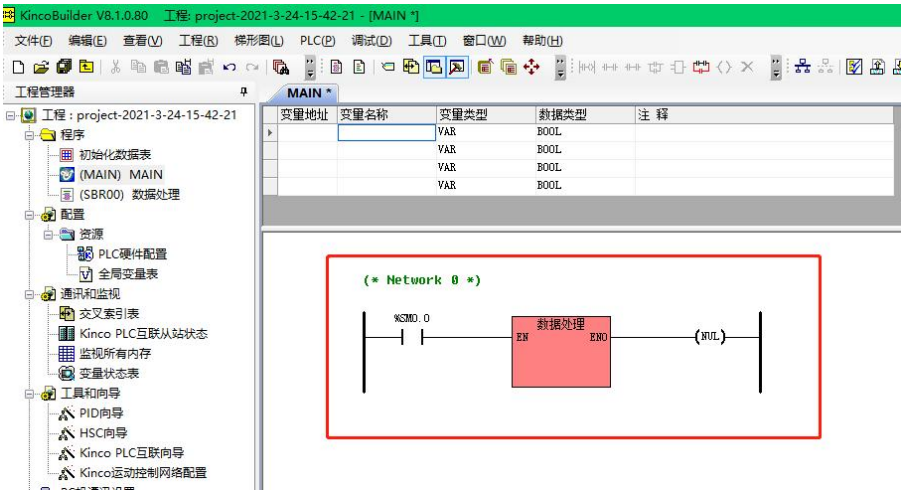


3）调用子程序

子程序需要在主程序 MAIN 中进行调用，否则子程序不会执行。首先将鼠标点到程序的网络中 SM0.0 处，然后按下图箭头的顺序进行操作即可。



下图为主程序调用子程序的网络。

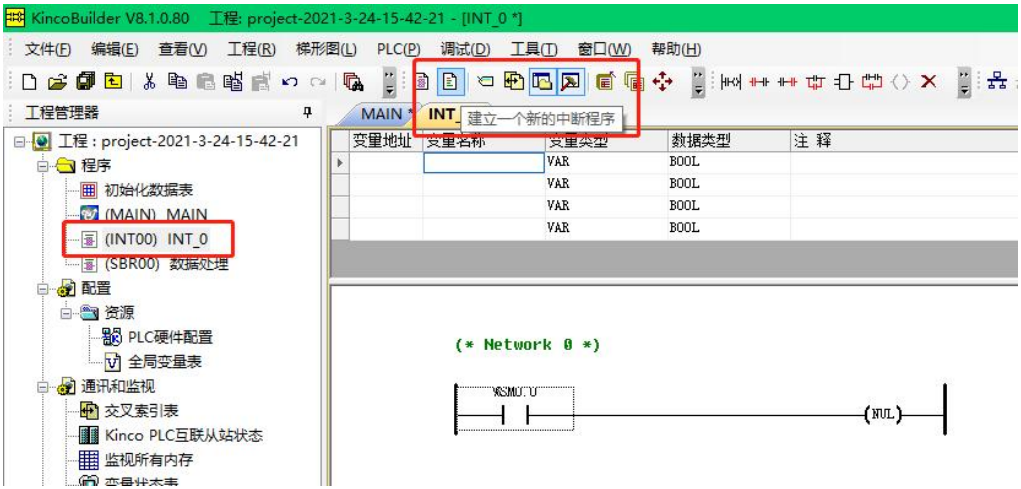


- **建立中断程序**

1）新建中断程序

点击下图的图标会出现提示“建立一个新的中断程序”，点击后自动新建了一个名叫 INT_0

的子程序。

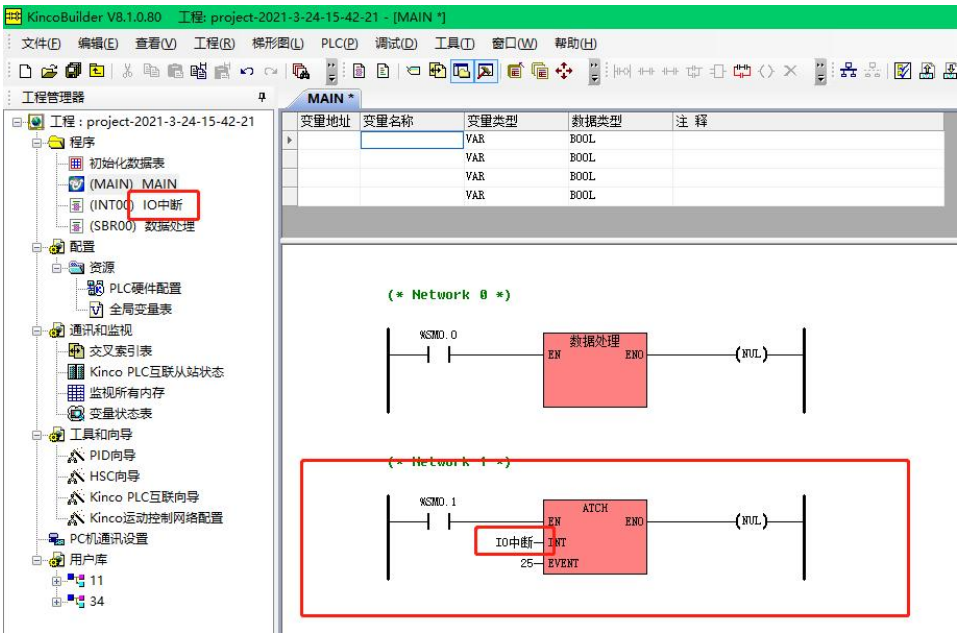


2）修改中断程序的名字

过程和修改子程序名字一样，在此不再累述。

3）连接中断程序

要使用中断程序，需要在主程序 MAIN 中通过相关指令连接中断。



关于中断的使用介绍请参考 6.12 中断指令章节。

第六章 基本应用指令集

K 系列 PLC 支持 IEC 61131-3 标准的基本指令及其大部分功能/功能块，编程风格符合 IEC 61131-3 标准要求，同时，根据实际的需要，对标准指令作了适当的扩充，可以满足不同用户、多应用领域工程实际的需要。

6.1 指令集综述

本章对指令集中的所有指令进行了详细的说明,同时对大多数指令也提供了具体的使用实例。对于指令的说明包括了 LD、IL 两种格式。

在 LD 格式中，下文的描述没有具体提及能量流，能量流的含义是固定的，一个网络上各指令的状态控制着能量流在本网络上的流动。我们可以认为它是部分指令（无 EN、ENO 操作数的指令）的虚拟输入，并且其类型是 BOOL 型。为了便于描述，在下文中我们将能量流是否到达某点简称为在该点能量流的值为 1 或者 0。

另外，在 LD 格式中，下文对 EN、ENO 操作数和其数据类型也没有说明，因为它们均为 BOOL 型，且含义也是固定的：EN 的意思为“使能”，若某功能/功能块的 EN 值为 1，则该功能/功能块才被执行，能量流继续向前流动，在 ENO 端输出为 1。若 EN 值为 0，则该功能/功能块不被执行，在 ENO 端输出为 0。

在 5.1.2.2 关于 CR 中提到，在 IL 程序中，每一条指令执行完之后都会对 CR 值产生相应的影响。本章对每条 IL 指令对 CR 值的影响都作了说明，在说明时采用了在 5.1.2.2 中规定的“分组缩写”符号。

6.2 位指令

6.2.1 标准触点

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	常开触点			☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
	常闭触点			
IL	LD	LD *bit*	C	
	AND	AND *bit*	P	
	OR	OR *bit*		
	LDN	LDN *bit*	C	
	ANDN	ANDN *bit*	P	
	ORN	ORN *bit*		

操作数	输入/输出	数据类型	允许的内存区
bit	输入	BOOL	I、Q、V、M、SM、L、T、C、RS、SR、常量

- LD

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

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

- IL

常开触点由 *LD*、*AND*、*OR* 指令提供。

LD 指令将 *bit* 值装载于 CR 中并作为新的 CR 值；

AND 指令将 CR 值和 *bit* 值进行“与”运算，并将运算结果作为新的 CR 值；

OR 指令将 CR 值和 *bit* 值进行“或”运算，并将运算结果作为新的 CR 值。

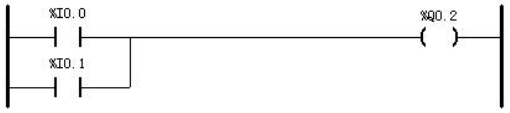
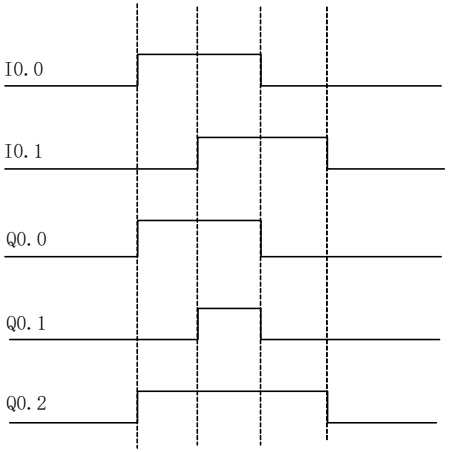
常闭触点由 *LDN*、*ANDN*、*ORN* 指令提供。

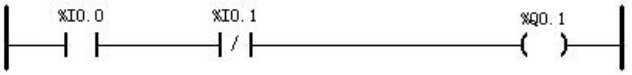
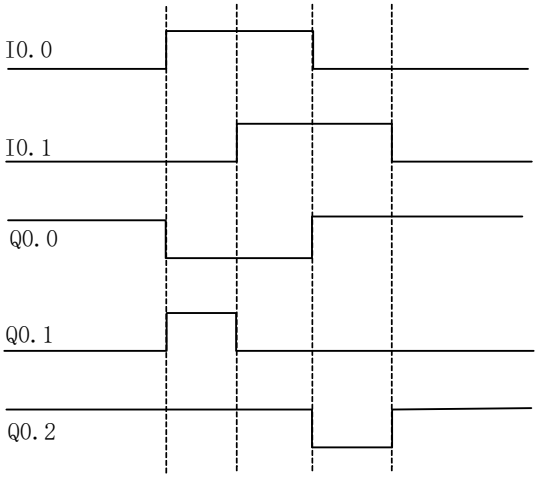
LDN 指令将 *bit* 值取反，然后将结果装载于 CR 中并作为新的 CR 值；

ANDN 指令将 *bit* 值取反，然后将结果和 CR 值进行“与”运算，并将运算结果作为新 CR 值；

ORN 指令将 *bit* 值取反，然后将结果和 CR 值进行“或”运算，并将运算结果作为新的 CR 值。

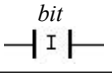
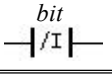
➢ 指令使用举例

LD	IL
	LD %I0.0 ST %Q0.0
	LD %I0.0 AND %I0.1 ST %Q0.1
	LD %I0.0 OR %I0.1 ST %Q0.2
时序图	
	

LD	IL
	LDN %I0.0 ST %Q0.0
	LD %I0.0 ANDN %I0.1 ST %Q0.1
	LD %I0.0 ORN %I0.1 ST %Q0.2
时序图	
	

6.2.2 立即触点

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	立即常开触点			☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
	立即常闭触点			
IL	LDI	LDI *bit*	C	
	ANDI	ANDI *bit*	P	
	ORI	ORI *bit*		
	LDNI	LDNI *bit*	C	
	ANDNI	ANDNI *bit*	P	
	ORNI	ORNI *bit*		

操作数	输入/输出	数据类型	允许的内存区
bit	输入	BOOL	I（CPU 本体）

在指令执行时，立即触点直接读取 *bit* 的物理输入通道的实际状态，但是不更新输入映像区。与普通触点指令相比，立即触点指令读取的不是输入映像区的数据，因此不会受到 CPU 扫描周期的影响，能够快速对输入信号做出响应。

立即触点指令只能用于 CPU 本体的 DI 点。在用户工程的【PLC 硬件配置】中，CPU 模块的【I/O 设置】>【输入滤波】的设置对立即触点指令没有影响。

- LD

立即常开触点的作用是：若 *bit* 的物理输入通道的实际状态值为 1，则触点闭合，能量流继续向后传递，否则触点断开，能量流也被切断。

立即常闭触点的作用是：若 *bit* 的物理输入通道的实际状态值为 0，则触点闭合，能量流继

续向后传递，否则触点断开，能量流也被切断。

- IL

  立即常开触点由 *LDI*、*ANDI*、*ORI* 指令提供。

  LDI 指令将 *bit* 的物理输入通道的实际状态值装载于 CR 中并作为新的 CR 值；

  ANDI 指令将 CR 值和 *bit* 的物理输入通道的实际状态值进行“与”运算，并将运算结果作为新的 CR 值；

  ORI 指令将 CR 值和 *bit* 的物理输入通道的实际状态值进行“或”运算，并将运算结果作为新的 CR 值。

  立即常闭触点由 LDNI、ANDNI、ORNI 指令提供。

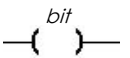
  LDNI 指令将 *bit* 的物理输入通道的实际状态值取反，然后将结果装载于 CR 中并作为新的 CR 值；

  ANDNI 指令将 *bit* 的物理输入通道的实际状态值取反，然后将结果和 CR 值进行“与”运算，并将运算结果作为新的 CR 值；

  ORNI 指令将 *bit* 的物理输入通道的实际状态值取反，然后将结果和 CR 值进行“或”运算，并将运算结果作为新的 CR 值。

6.2.3 普通输出

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	线圈			☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
	取反线圈			
	空线圈			
IL	ST	ST *bit*	U	
	STN	STN *bit*		

操作数	输入/输出	数据类型	允许的内存区
bit	输出	BOOL	Q、V、M、SM、L

- LD

线圈的作用是：将线圈左侧能量流的值赋给 *bit* 。

取反线圈的作用是：将线圈左侧能量流的值取反并赋给 *bit* 。

空线圈的作用是：在 LD 程序中指示一个网络的结束。该指令仅为方便用户的编程，并不执行具体的操作。

- IL

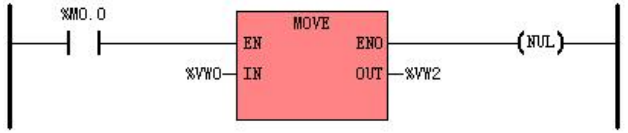
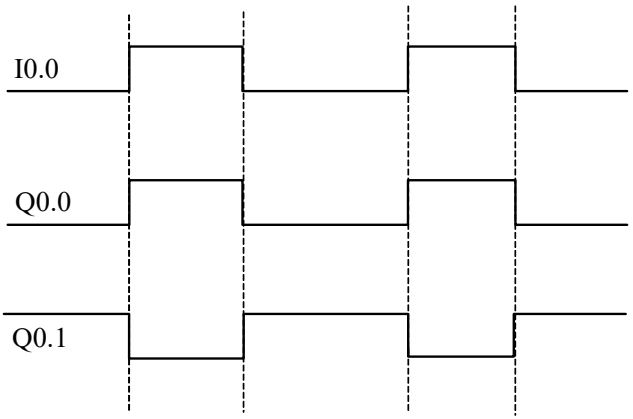
线圈由 *ST* 指令提供，取反线圈由 STN 指令提供。

ST 指令用于将 CR 值赋给 *bit* 。

STN 指令用于将 CR 值取反并赋给 *bit* 。

ST、*STN* 指令的执行对 CR 值无影响。

➢ 指令使用举例

<table>
<tr><td>LD</td><td>IL</td></tr>
<tr><td> 
</td><td>LD %I0.0
ST %Q0.0
STN %Q0.1</td></tr>
<tr><td> 
</td><td>LD %M0.0
MOVE %VW0, %VW2</td></tr>
<tr><td colspan="2">时序图</td></tr>
<tr><td colspan="2"> 
</td></tr>
</table>

6.2.4 立即输出

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	立即线圈	*bit* —(I)—		
IL	STI	STI*bit*	U	

操作数	输入/输出	数据类型	允许的内存区
bit	输出	BOOL	Q（CPU 本体）

立即输出指令只能用于 CPU 本体的 DO 点。

- LD

在指令执行时，立即线圈左侧能量流的值被直接写入 *bit* 的物理输出点进行输出，同时更新相应的输出映像区。

- IL

立即输出由 *STI* 指令提供。

STI 指令将 CR 值直接写入 *bit* 的物理输出点进行输出，同时更新相应的输出映像区。

STI 指令的执行对 CR 值无影响。

6.2.5 置位与复位

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	复位	*bit* —(R)—		☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
	置位	*bit* —(S)—		
IL	复位	R *bit*	U	
	置位	S *bit*		

操作数	输入/输出	数据类型	允许的内存区
bit	输出	BOOL	Q、V、M、SM、L

- LD

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

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

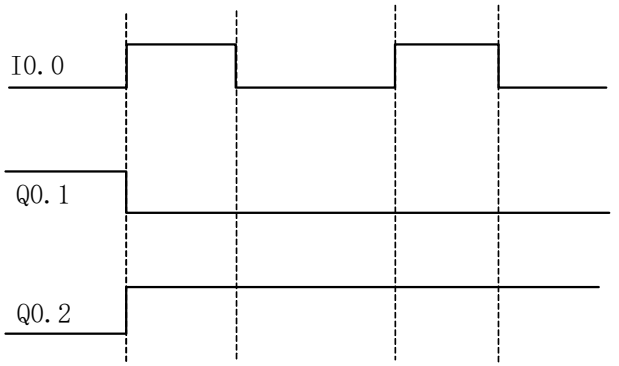
- IL

  复位指令的作用是：若 CR 值为 1，则 *bit* 值被置为 0，否则 *bit* 值保持不变。

  置位指令的作用是：若 CR 值为 1，则 *bit* 值被置为 1，否则 *bit* 值保持不变。

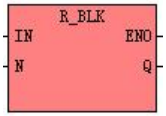
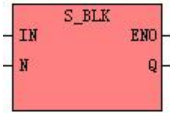
  R、S 指令的执行不影响 CR 值。

➢ 指令使用举例

<table>
<tr><td>LD</td><td>IL</td></tr>
<tr><td> 
</td><td>LD %I0.0
R %Q0.1
S %Q0.2</td></tr>
<tr><td colspan="2">时序图</td></tr>
<tr><td colspan="2">假定执行该指令之前Q0.1和Q0.2的值分别为1和0
 
</td></tr>
</table>

6.2.6 块置位与块复位

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	块复位			☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
	块置位			
IL	块复位	R_BLK *N,Q*	U	
	块置位	S_BLK *N,Q*		

操作数	输入/输出	数据类型	允许的内存区
IN	输入	BOOL	能流
N	输入	INT	L、M、V、常量
Q	输出	BOOL	Q、V、M、SM、L

参数 N 的最大数量为 1024。参数 Q 为一个内存块的起始地址，内存块长度为 N，注意整个内存块都必须位于合法的地址范围， 否则结果不可预期。

- LD

R_BLK 的作用：若 *IN* 值为 1，则从指定的位地址 *Q* 开始的连续 *N* 个位都被置为 0，否则这些位保持不变。

S_BLK 的作用：若 *IN* 值为 1，则从指定的位地址 *Q* 开始的连续 *N* 个位都被置为 1，否则这些位保持不变。

- IL

R_BLK 的作用：若 CR 值为 1，则从指定的位地址 *Q* 开始的连续 *N* 个位都被置为 0，否则这些位保持不变。

S_BLK 的作用：若 CR 值为 1，则从指定的位地址 *Q* 开始的连续 *N* 个位都被置为 1，否则这些位保持不变。

R_BLK、S_BLK 指令的执行不影响 CR 值。

6.2.7 立即置位与立即复位

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	立即复位	*bit* —(RI)—		
	立即置位	*bit* —(SI)—		
IL	立即复位	RI*bit*	U	
	立即置位	SI*bit*		

操作数	输入/输出	数据类型	允许的内存区
bit	输出	BOOL	Q（CPU 本体）

立即置位与立即复位指令只能用于 CPU 本体的 DO 点。

- LD

立即复位线圈的作用是：若线圈左侧能量流的值为 1，则 *bit* 对应的输出映像寄存器和物理输出点均被置为 0，否则这两者的值保持不变。

立即置位线圈的作用是：若线圈左侧能量流的值为 1，则 *bit* 对应的输出映像寄存器和物理输出点均被置为 1，否则这两者的值保持不变。

- IL

立即复位指令的作用是：若 CR 值为 1，则 *bit* 对应的输出映像寄存器和物理输出点均被置为 0，否则这两者的值保持不变。

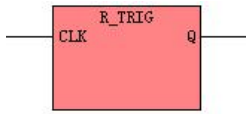
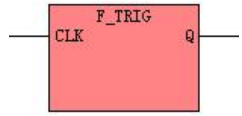
立即置位指令的作用是：若 CR 值为 1，则 bit 对应的输出映像寄存器和物理输出点均被置为

1，否则这两者的值保持不变。

RI、SI 指令的执行不影响 CR 值。

6.2.8 边沿检测

➢ 指令及其参数说明

	名称	指令	影响 CR 值	
LD	上升沿检测			☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
	下降沿检测			
IL	上升沿检测	R_TRIG	P	
	下降沿检测	F_TRIG		

参数	输入/输出	数据类型	允许的内存区
CLK（LD）	输入	BOOL	能量流
Q（LD）	输出	BOOL	能量流

- LD

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

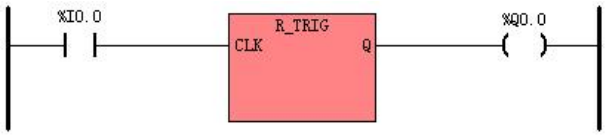
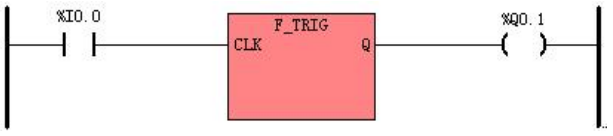
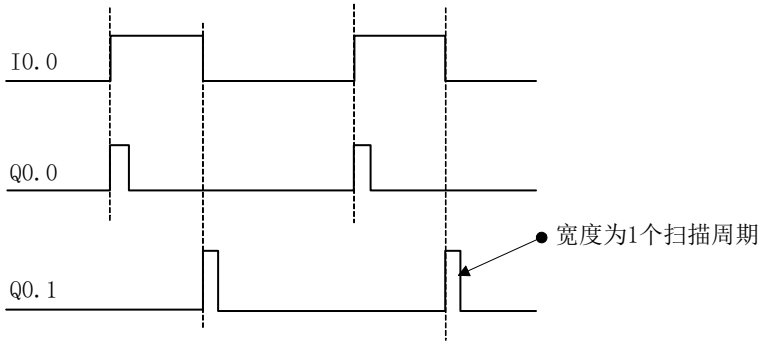
F_TRIG 用于检测 *CLK* 输入的下降沿跳变：如果 *CLK* 值产生了由 1 到 0 的跳变，则 *Q* 输出为 1 并保持一个扫描周期，然后 *Q* 将输出为 0。

- IL

R_TRIG 用于检测当前点 CR 值的上升沿跳变：如果当前点的 CR 值产生了由 0 到 1 的跳变，则立即将 CR 值设置为 1，否则将 CR 值设置为 0。

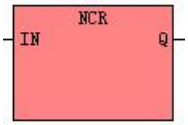
F_TRIG 用于检测当前点 CR 值的下降沿跳变：如果当前点的 CR 值产生了由 1 到 0 的跳变，则立即将 CR 值设置为 1，否则将 CR 值设置为 0。

➢ 指令使用举例

<table>
<tr><td>LD</td><td>IL</td></tr>
<tr><td> 
</td><td>LD %I0.0
R_TRIG
ST %Q0.0</td></tr>
<tr><td> 
</td><td>LD %I0.0
F_TRIG
ST %Q0.1</td></tr>
<tr><td colspan="2">时序图</td></tr>
<tr><td colspan="2"> 
</td></tr>
</table>

6.2.9 NCR（取反）

➢ 指令及其参数说明

	名称	指令	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	取反			
IL	取反	NCR	P	

参数	输入/输出	数据类型	允许使用的内存区
IN	输入	BOOL	能量流
Q	输出	BOOL	能量流

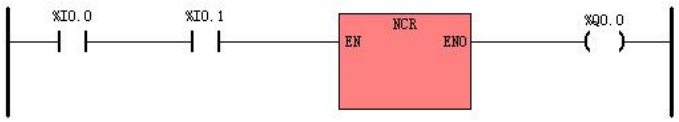
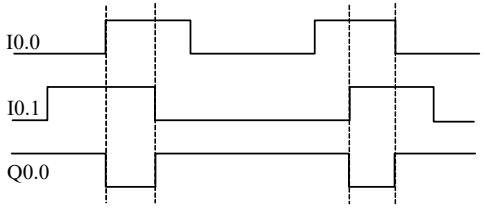
- LD

  该指令用于改变能量流的状态：将输入端能量流的值取反并输出。

- IL

  该指令用于改变 CR 值：将 CR 值取反，并将结果作为新的 CR 值。

➢ 指令使用举例

LD	IL
	LD %I0.0 AND %I0.1 NCR ST %Q0.0
时序图	
	

6.2.10 双稳态触发器

双稳态触发器是 IEC61131-3 标准中定义的功能块之一，共有 SR 触发器（置位优先触发器）和 RS 触发器（复位优先触发器）两种。

关于功能块及其实例的使用请参阅 3.6.5 关于功能块以及功能块实例中的描述。

6.2.10.1 SR（置位优先触发器）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	置位优先触发器	*SRx* S1 SR Q1 R		☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
IL	置位优先触发器	LD *S1* SR *SRx*, *R*	P	

参数	输入/输出	数据类型	允许使用的内存区
SRx	-	SR 触发器实例	SR
S1	输入	BOOL	能量流
R	输入	BOOL	I、Q、M、V、L、SM、T、C、RS、SR
OUT	输出	BOOL	能量流

SR 触发器的置位端输入是 *S1*，复位端输入是 *R*。其真值表如下：

S1	*R*	输出 *Q1* 及 *SRx* 状态值
0	0	保持前一状态
0	1	0
1	0	1
1	1	1

6.2.10.2 RS（复位优先触发器）

➢ 指令及其操作数说明

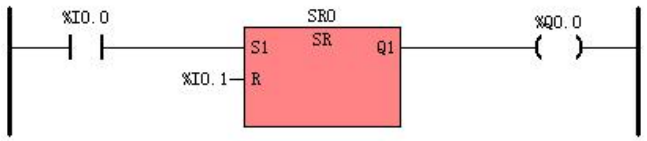
	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	复位优先 触发器			
IL	复位优先 触发器	LD *S* RS *RSx*, *R1*	P	

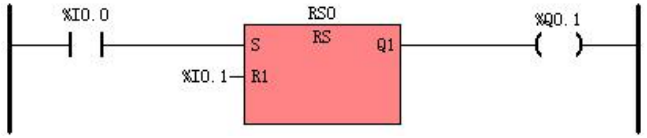
参数	输入/输出	数据类型	允许使用的内存区
RSx	-	RS 触发器实例	RS
S	输入	BOOL	能量流
R1	输入	BOOL	I、Q、M、V、L、SM、T、C、RS、SR
OUT	输出	BOOL	能量流

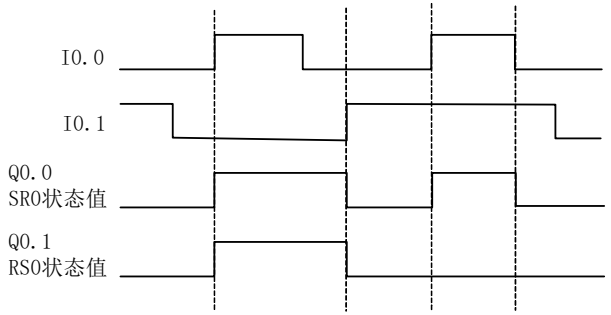
RS 触发器的置位端输入是 *S*，复位端输入是 *R1*。RS 触发器的真值表如下：

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

6.2.10.3 RS、SR 使用举例

<table>
<tr><th>LD</th><th>IL</th></tr>
<tr><td>(* Network 0 *)
 

(* Network 1 *)
 
</td><td>(* Network 0 *)
LD %I0.0
SR SR0, %I0.1
ST %Q0.0

(* Network 1 *)
LD %I0.0
RS RS0, %I0.1
ST %Q0.1</td></tr>
<tr><td colspan="2">时序图</td></tr>
<tr><td colspan="2"> 
</td></tr>
</table>

6.2.11 ALT（反转）

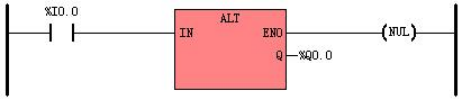
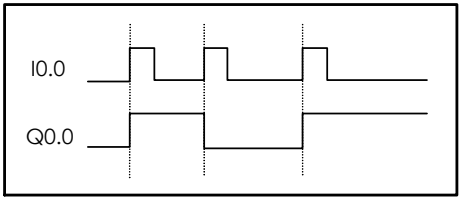
➢ 指令及其参数说明

	名称	指令	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	取反			
IL	取反	ALT *Q*	U	

参数	输入/输出	数据类型	允许的内存区
IN（LD）	输入	BOOL	能量流
Q	输出	BOOL	Q、V、M、SM、L

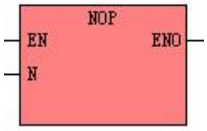
- LD
  如果输入 *IN*产生了上升沿跳变，则 *Q*立即被取反，否则 *Q*保持不变。
- IL
  如果当前点的 CR 值产生了上升沿跳变，则 *Q*立即被取反，否则 *Q*保持不变。
  该指令的执行不影响 CR 值。

➢ 指令使用举例

LD	IL
	LD %I0.0 ALT %Q0.0
时序图	
	

6.2.12 NOP（空操作）

➢ 指令及其参数说明

	名称	指令	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	空操作			
IL	空操作	NOP *N*	U	

操作数	输入/输出	数据类型	允许的内存区
N	输入	INT	常数（正数）

NOP 指令仅仅让 CPU 产生 *N* 次空操作，不会影响用户程序的执行结果。参数 *N* 指定了执行空操作的次数，它必须是一个大于 0 的 INT 型常数。

用户可以在程序中调用 NOP 指令进行不精确的延时，具体的延时效果需要用户自己调试确定。

6.2.13 括号修饰符

➢ 指令及其参数说明

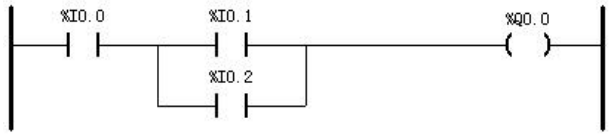
	名称	指令格式	影响 CR 值	☑ K5
IL	AND(	AND(	U	☑ K2 ☑ KS ☑ KW
	OR(	OR(		
	)	)	P	☑ MK ☑ K6

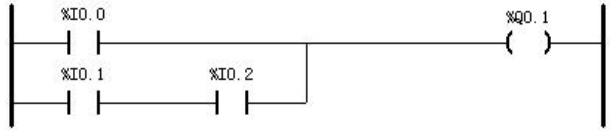
括号修饰符只有在 IL 中提供。在 IL 语言中只有简单的表达式，而不可能象在 LD、ST 等语言中那样采用复杂的表达式作为操作数,因此 IEC61131-3 标准中定义了括号来处理一些复杂的表达式。“AND(”或者“OR(”都是和“)”配对使用的。

在 IL 程序中，执行“AND(”和“)”之间的指令之前，先将 CR 值暂存，再执行括号中的指令，执行完之后将结果与刚才暂存的 CR 值进行“与”运算，并将运算结果作为新的 CR 值。

类似地，执行“OR(”和“)”之间的指令之前，首先将 CR 值暂存，再执行括号中的指令，执行完之后将结果与刚才暂存的 CR 值进行“或”运算，并将运算结果作为新的 CR 值。

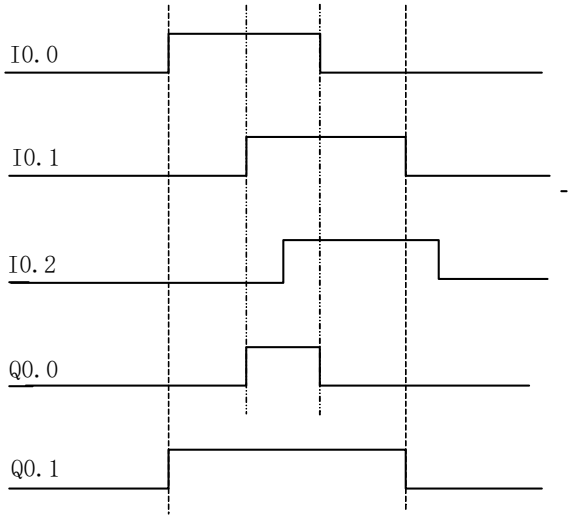
➢ 指令使用举例

LD	IL
	LD %I0.0 AND(LD %I0.1 OR %I0.2) ST %Q0.0



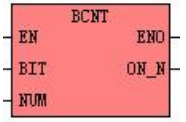

```
LD      %I0.0
OR(
LD      %I0.1
AND     %I0.2
)
ST      %Q0.1
```

时序图



6.2.14 BCNT（置位统计）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	BCNT			☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
IL	BCNT	BCNT BIT, ON_N, NUM	U	

参数	输入/输出	数据类型	允许的内存区
BIT	输入	BIT	I、Q、M、V、L
NUM	输入	WORD	I、Q、M、V、L、常量
ON_N	输出	WORD	I、Q、M、V、L

该指令用于统计从 *BIT* 地址开始的连续 *NUM* 个位中，值等于 TRUE 的位的数量，并将统计数量将输出到 *ON_N* 中。**注意：1 <= *NUM* <=256。**

- LD

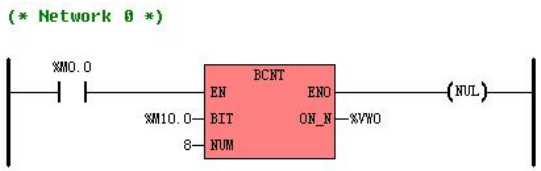
  如果 *EN* 为 1，则该指令进行置位统计，并将结果输出到 ON_N 中。

- IL

  如果 CR 值为 1，则该指令进行置位统计，并将结果输出到 ON_N 中。

  该指令的执行对 CR 值无影响。

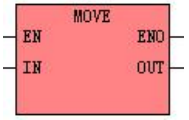
➢ 指令使用举例

LD		若 M0.0 为 TRUE，则执行 BCNT 指令：统计 M10.0 开始的 8 个位中有多少位等于 TRUE，并将结果存放在 VW0 中
IL	LD %M0.0 BCNT %M10.0, 8, %VW0	

6.3 赋值指令

6.3.1 MOVE（赋值）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	MOVE			
IL	MOVE	MOVE IN, OUT	U	

参数	输入/输出	数据类型	允许的内存区
IN	输入	BYTE、WORD、DWORD、INT、DINT、REAL	I、Q、M、V、L、SM、AI、AQ、T、C、HC、常量、指针
OUT	输出	BYTE、WORD、DWORD、INT、DINT、REAL	Q、M、V、L、SM、AQ、指针

该指令执行赋值操作，其参数 *IN* 与 *OUT* 的数据类型必须一致。

- LD

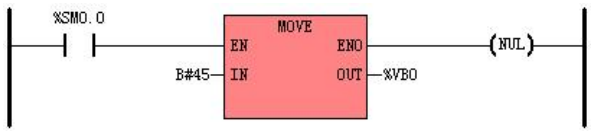
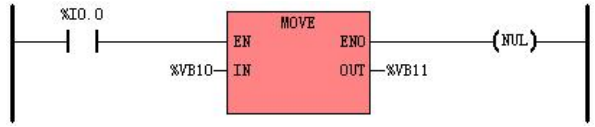
  如果 *EN* 为 1，则该指令将输入变量 *IN* 的值赋给输出变量 *OUT*。

- IL

  如果 CR 值为 1，则该指令将输入变量 *IN* 的值赋给输出变量 *OUT*。

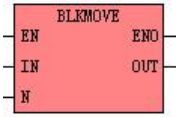
  该指令的执行对 CR 值无影响。

➢ 指令使用举例

<table>
<tr><td rowspan="2">LD</td><td> 
</td><td>SM0.0 固定为 1，因此 MOVE 指令总是执行：BYTE 类型常量 B#45 被赋给 VB0。</td></tr>
<tr><td> 
</td><td>I0.0 为 0：不执行 MOVE 指令。
I0.0 为 1：VB10 的值被赋给 VB11。</td></tr>
<tr><td>IL</td><td colspan="2">LD %SM0.0 (* 建立 CR，其值为 1 *)
MOVE B#45, %VB0 (* VB0 被赋值为 B#45 *)</td></tr>
<tr><td></td><td colspan="2">LD %I0.0 (* 建立 CR，其值为 I0.0 的值 *)
MOVE %VB10, %VB11 (* 若 CR 为 1：VB10 的值被赋赋给 VB11 *)
(* 若 CR 为 0：不执行 MOVE 指令，VB11 的值不变 *)</td></tr>
</table>

6.3.2 BLKMOVE（块传送）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	BLKMOVE			
IL	BLKMOVE	BLKMOVE *IN*, *OUT*, *N*	U	

参数	输入/输出	数据类型	允许的内存区
IN	输入	BYTE、WORD、DWORD、INT、DINT、REAL	I、Q、M、V、L、AI、AQ、T、C、HC
N	输入	BYTE	I、Q、M、V、L、SM、常量
OUT	输出	BYTE、WORD、DWORD、INT、DINT、REAL	Q、M、V、L、AQ

***IN*、*OUT*为可变长度的内存块的起始地址，整个内存块都不允许落在非法内存区域， 否则结果不可预期。**

参数 *IN*、*OUT* 的数据类型必须一致。该指令用于将从地址 *IN* 开始的连续 *N* 个变量的值传送给从地址 *OUT* 开始的连续 *N* 个变量，这些变量的数据类型与 *IN*、*OUT* 一致。

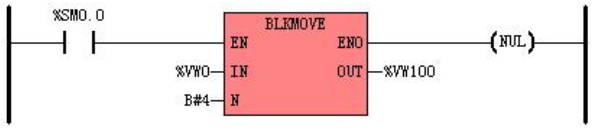
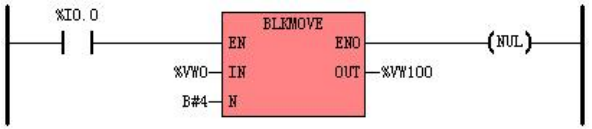
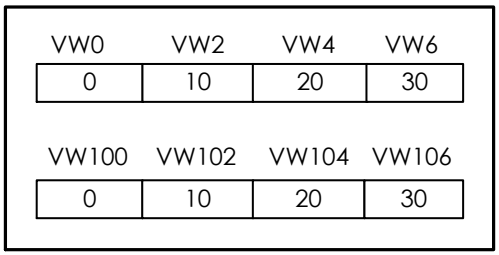
- LD

  如果 *EN* 为 1，则该指令被执行。

- IL

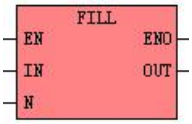
  如果 CR 值为 1，则该指令被执行。该指令的执行对 CR 值无影响。

➢ 指令使用举例

<table>
<tr><td rowspan="2">LD</td><td> 
</td><td>SM0.0 恒为 1，因此 BLKMOVE 指令总是执行：VW0 至 VW6 中的数据被依次传送到 VW100 至 VW106 中。</td></tr>
<tr><td> 
</td><td>I0.0 为 0：不执行 BLKMOVE。
I0.0 为 1：VW0 至 VW6 中的数据被依次传送到 VW100 至 VW106 中。</td></tr>
<tr><td rowspan="2">IL</td><td colspan="2">LD %SM0.0 (* 建立 CR，其值为 1 *)
BLKMOVE %VW0, %VW100, B#4 (* VW0 至 VW6 中的数据被传送到 VW100 至 VW106 中 *)</td></tr>
<tr><td colspan="2">LD %I0.0 (* 建立 CR，其值为 I0.0 的值 *)
BLKMOVE %VW0, %VW100, B#4 (* 若 CR 为 1：VW0 至 VW6 中的数据被传送到 VW100 至 VW106 中 *)
(* 若 CR 为 0：不执行 BLKMOVE 指令 *)</td></tr>
<tr><td>结果</td><td colspan="2">若 BLKMOVE 被执行，则结果举例如下：
 
</td></tr>
</table>

6.3.3 FILL（块赋值）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	FILL			
IL	FILL	FILL *IN*, *OUT*, *N*	U	

参数	输入/输出	数据类型	允许的内存区
IN	输入	BYTE	常量
N	输入	BYTE	常量
OUT	输出	BYTE	M、V、L

***OUT* 参数为一个可变长度的内存块的起始地址，整个块内存都不可以落在非法内存区域，否则结果不可预期。**

该指令用于将从地址 *OUT* 开始的连续 *N* 个字节变量赋值为常数 *IN*。

注意， *OUT* 为一个可变长度的内存块的起始地址，整个内存块都不可以落在非法内存区域，否则结果不可预期。

- LD

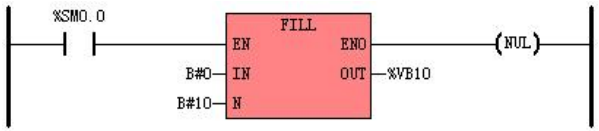
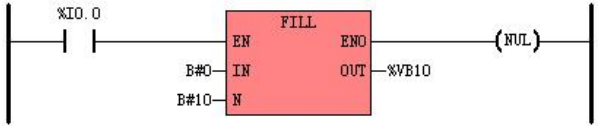
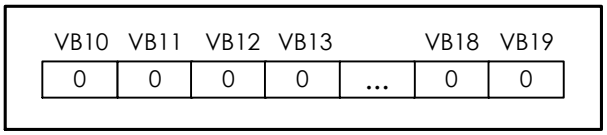
  如果 *EN* 为 1，则该指令被执行。

- IL

  如果 CR 值为 1，则该指令被执行。

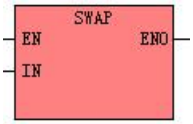
  该指令的执行对 CR 值无影响。

➢ 指令使用举例

<table>
<tr><td rowspan="2">LD</td><td> 
</td><td>SM0.0 恒为 1，因此 FILL 指令总是执行：VB10 至 VB19 总计 10 个变量均被赋值为 B#0。</td></tr>
<tr><td> 
</td><td>I0.0 为 0：不执行 FILL 指令。
I0.0 为 1：VB10 至 VB19 总计 10 个变量均被赋值为 B#0。</td></tr>
<tr><td rowspan="2">IL</td><td colspan="2">LD %SM0.0 (* 建立 CR，其值为 1 *)
FILL B#0, %VB10, B#10 (* VB10 至 VB19 总计 10 个变量均被赋值为 B#0 *)</td></tr>
<tr><td colspan="2">LD %I0.0 (* 建立 CR，其值为 I0.0 的值 *)
FILL B#0, %VB10, B#10 (* 若 CR 为 1：VB10 至 VB19 总计 10 个变量均被赋值为 B#0 *)
(* 若 CR 为 0：不执行 FILL 指令 *)</td></tr>
<tr><td>结果</td><td colspan="2">若 FILL 被执行，则结果如下：
 
</td></tr>
</table>

6.3.4 SWAP（交换）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	SWAP			
IL	SWAP	SWAP *IN*	U	

参数	输入/输出	数据类型	允许的内存区
IN	输入/输出	WORD、DWORD	Q、M、V、L、SM

若参数 *IN* 是 WORD 型，则该指令用于交换 *IN* 的高字节与低字节；

若参数 *IN* 是 DWORD 型，则该指令用于交换 *IN* 的高字与低字。

- LD

  如果 *EN* 为 1，则该指令被执行。

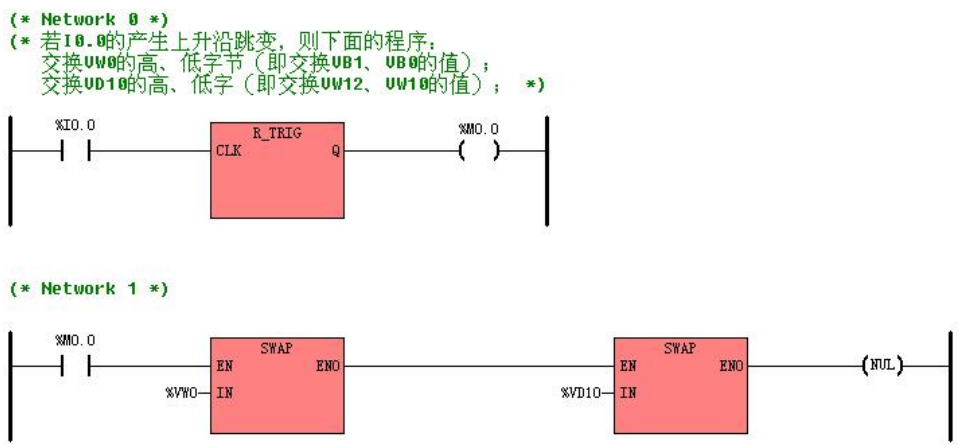
- IL

  如果 CR 值为 1，则该指令被执行。

  该指令的执行对 CR 值无影响。

➢ 指令使用举例

LD

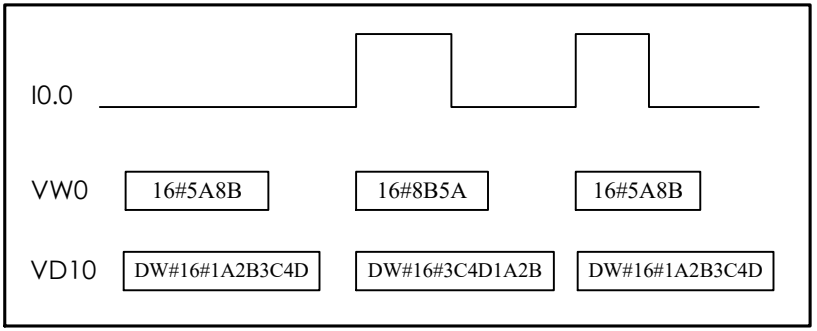


IL

```
(* Network 0 *)
LD      %I0.0
R_TRIG                  (* 若 I0.0 的产生上升沿跳变 *)
SWAP    %VW0        (* 则交换 VW0 的高、低字节 *)
SWAP    %VD10       (* 则交换 VD10 的高、低字 *)
```

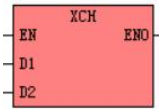
结果

假定 VW0 的初始值是 W#16#5A8B，VD10 的初始值是 DW#16#1A2B3C4D。



6.3.5 XCH（两个数据交换）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K2 ☑ KS ☑ MK ☑ K6
LD	XCH			
IL	XCH	XCH *D1,D2*	U	

参数	输入/输出	数据类型	允许的内存区
D1	输入/输出	BYTE、WORD、DWORD、INT、DINT、REAL	Q、M、V、L
D2	输入/输出	BYTE、WORD、DWORD、INT、DINT、REAL	Q、M、V、L

该指令用于将 *D1*、*D2* 中的值相互交换。

注意：D1、D2 的数据类型必须相同！

- LD

  如果 *EN* 为 1，则该指令被执行。

- IL

  如果 CR 值为 1，则该指令被执行。

  该指令的执行对 CR 值无影响。

➢ 指令使用举例

LD	
IL	(* Network 0 *) LD %M0.0 R_TRIG (* 若 M0.0 的产生上升沿跳变 *) XCH %VW0, %VW2 (* 则将 VW0 和 VW2 的值相互交换 *)
结果	假定执行前 VW0 的值为 1234，VW2 的值为 5678。 则执行一次后的结果为：VW0 的值为 5678，VW2 的值为 1234。

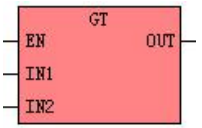
6.4 比较指令

对于所有的比较指令，BYTE、WORD、DWORD 型采用了无符号数据进行比较，其余数据类型采用了有符号数据进行比较。

若输入参数为实数（REAL 型）时，由于存在舍入误差，因此两个实数不可能进行精确的比较。当两个实数的绝对值的差值位于[-0.000001, 0.000001]之内，PLC 就认为这两个实数是相等的，否则才是不相等的。关于实数误差请参见 3.2 数据类型 中的相关描述。

6.4.1 GT（大于）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	GT			
IL	GT	GT *IN1*, *IN2*	P	

参数	输入/输出	数据类型	允许的内存区
IN1	输入	BYTE、WORD、DWORD、INT、DINT、REAL	I、Q、M、V、L、SM、AI、AQ、T、C、HC、常量、指针
IN2	输入	BYTE、WORD、DWORD、INT、DINT、REAL	I、Q、M、V、L、SM、AI、AQ、T、C、HC、常量、指针
OUT（LD）	输出	BOOL	能量流

参数 *IN1*、*IN2* 的数据类型必须一致。

- LD

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

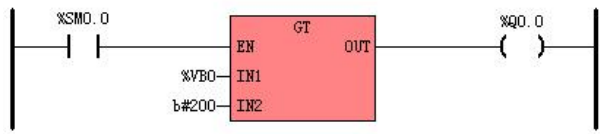
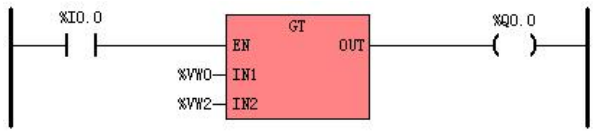
如果 *EN* 为 0，该指令不执行，在 *OUT* 端输出为 0。

- IL

  如果 CR 值为 1，该指令被执行：若 *IN1* 大于 *IN2*，则将 CR 置为 1，否则将 CR 置为 0；

  如果 CR 值为 0，该指令不执行，CR 值保持为 0。

➢ 指令使用举例

<table>
<tr><td rowspan="2">LD</td><td> 
</td><td>由于 SM0.0 固定为 1，因此 GT 总是执行：若 VB0 的值大于 B#200，则 Q0.0 被置为 1，否则 Q0.0 就置为 0。</td></tr>
<tr><td> 
</td><td>若 I0.0 为 0：不执行 GT，Q0.0 被置为 0。
若 I0.0 为 1：若 VW0 的值大于 VW2 的值，则 Q0.0 被置为 1，否则 Q0.0 被置为 0。</td></tr>
<tr><td rowspan="2">IL</td><td colspan="2">LD %SM0.0 (* 建立 CR，其值为 1 *)
GT %VB0, b#200 (* 若 VB0 大于 B#200，则 CR 被置为 1，否则 CR 被置为 0 *)
ST %Q0.0 (* 将 CR 值赋给 Q0.0 *)</td></tr>
<tr><td colspan="2">LD %I0.0 (* 建立 CR，其值为 I0.0 的值 *)
GT %VW0, %VW2 (* 若 CR 为 1：若 VW0 大于 VW2，则 CR 被置为 1，否则 CR 被置为 0 *)
(* 若 CR 为 0：不进行 GT 比较，CR 保持为 0 *)
ST %Q0.0 (* 将 CR 值赋给 Q0.0 *)</td></tr>
</table>

6.4.2 GE（大于等于）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	GE			
IL	GE	GE *IN1*, *IN2*	P	

参数	输入/输出	数据类型	允许的内存区
IN1	输入	BYTE、WORD、DWORD、INT、DINT、REAL	I、Q、M、V、L、SM、AI、AQ、T、C、HC、常量、指针
IN2	输入	BYTE、WORD、DWORD、INT、DINT、REAL	I、Q、M、V、L、SM、AI、AQ、T、C、HC、常量、指针
OUT（LD）	输出	BOOL	能量流

参数 *IN1*、*IN2* 的数据类型必须一致。

- LD

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

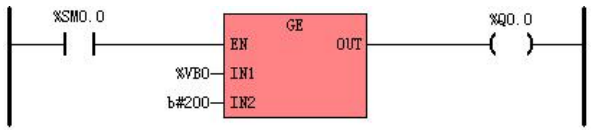
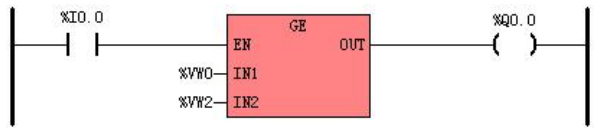
  如果 *EN* 为 0，该指令不执行，在 *OUT* 端输出为 0。

- IL

  如果 CR 值为 1，该指令被执行：若 *IN1* 大于等于 *IN2*，则将 CR 值置为 1，否则将 CR 值置为 0。

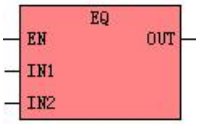
  如果 CR 值为 0，该指令不执行，CR 值保持为 0。

➢ 指令使用举例

<table>
<tr><td rowspan="2">LD</td><td> 
</td><td>由于 SM0.0 固定为 1，因此 GE 总是执行：若 VB0 的值大于等于 B#200，则 Q0.0 被置为 1，否则 Q0.0 就置为 0。</td></tr>
<tr><td> 
</td><td>若 I0.0 为 0：不执行 GE，Q0.0 被置为 0。
若 I0.0 为 1:若 VW0 的值大于等于 VW2 的值，则 Q0.0 被置为 1，否则 Q0.0 被置为 0。</td></tr>
<tr><td rowspan="2">IL</td><td colspan="2">LD %SM0.0 (* 建立 CR，其值为 1 *)
GE %VB0, b#200 (* 若 VB0 大于等于 B#200，则 CR 被置为 1，否则 CR 被置为 0 *)
ST %Q0.0 (* 将 CR 值赋给 Q0.0 *)</td></tr>
<tr><td colspan="2">LD %I0.0 (* 建立 CR，其值为 I0.0 的值 *)
GE %VW0, %VW2 (* 若 CR 为 1：则若 VW0 大于等于 VW2，则 CR 被置为 1，
(* 否则 CR 被置为 0 *)
(* 若 CR 为 0：则不进行 GE 比较，CR 保持为 0 *)
ST %Q0.0 (* 将 CR 值赋给 Q0.0 *)</td></tr>
</table>

6.4.3 EQ（等于）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	EQ			
IL	EQ	EQ IN1, IN2	P	

参数	输入/输出	数据类型	允许的内存区
IN1	输入	BYTE、WORD、DWORD、INT、DINT、REAL	I、Q、M、V、L、SM、AI、AQ、T、C、HC、常量、指针
IN2	输入	BYTE、WORD、DWORD、INT、DINT、REAL	I、Q、M、V、L、SM、AI、AQ、T、C、HC、常量、指针
OUT（LD）	输出	BOOL	能量流

参数 *IN1*、*IN2* 的数据类型必须一致。

- LD

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

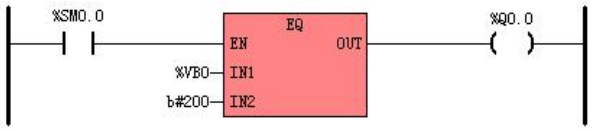
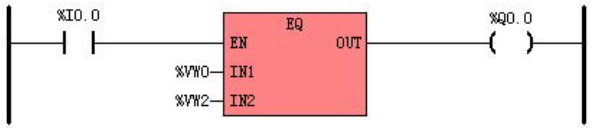
  如果 *EN* 为 0，该指令不执行，在 *OUT* 端输出为 0。

- IL

  如果 CR 值为 1，该指令被执行：若 *IN1* 等于 *IN2*，则将 CR 置为 1，否则将 CR 置为 0；

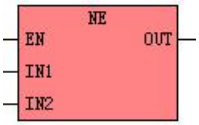
  如果 CR 值为 0，该指令不执行，CR 值保持为 0。

➢ 指令使用举例

<table>
<tr><td rowspan="2">LD</td><td> 
</td><td>由于 SM0.0 固定为 1，因此 EQ 总是执行：若 VB0 的值等于 B#200，则 Q0.0 被置为 1，否则 Q0.0 就置为 0。</td></tr>
<tr><td> 
</td><td>若 I0.0 为 0：不执行 EQ，Q0.0 被置为 0。
若 I0.0 为 1：若 VW0 的值等于 VW2 的值，则 Q0.0 被置为 1，否则 Q0.0 被置为 0。</td></tr>
<tr><td>IL</td><td colspan="2">LD %SM0.0 (* 建立 CR，其值为 1 *)
EQ %VB0, b#200 (* 若 VB0 等于 B#200，则 CR 被置为 1，否则 CR 被置为 0 *)
ST %Q0.0 (* 将 CR 值赋给 Q0.0 *)</td></tr>
<tr><td>IL</td><td colspan="2">LD %I0.0 (* 建立 CR，其值为 I0.0 的值 *)
EQ %VW0, %VW2 (* 若 CR 为 1：则若 VW0 等于 VW2，则 CR 被置为 1，否则 CR 被置为 0 *)
(* 若 CR 为 0：则不进行 EQ 比较，CR 保持为 0 *)
ST %Q0.0 (* 将 CR 值赋给 Q0.0 *)</td></tr>
</table>

6.4.4 NE（不等于）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	NE			
IL	NE	NE IN1, IN2	P	

参数	输入/输出	数据类型	允许的内存区
IN1	输入	BYTE、WORD、DWORD、INT、DINT、REAL	I、Q、M、V、L、SM、AI、AQ、T、C、HC、常量、指针
IN2	输入	BYTE、WORD、DWORD、INT、DINT、REAL	I、Q、M、V、L、SM、AI、AQ、T、C、HC、常量、指针
OUT（LD）	输出	BOOL	能量流

参数 *IN1*、*IN2* 的数据类型必须一致。

- LD

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

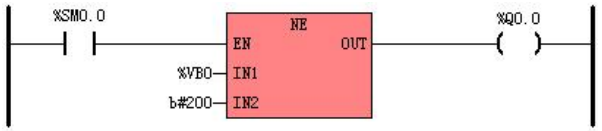
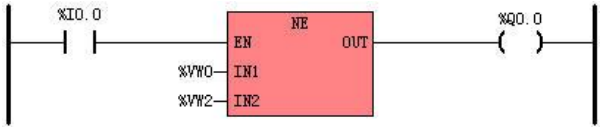
  若 *EN* 为 0，该指令不执行，在 *OUT* 端输出为 0。

- IL

  若 CR 值为 1，该指令被执行：若 *IN1* 不等于 *IN2*，则将 CR 置为 1，否则将 CR 置为 0；

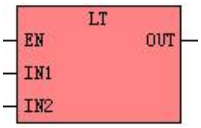
  若 CR 值为 0，该指令不执行，CR 值保持为 0。

➢ 指令使用举例

<table>
<tr><td rowspan="2">LD</td><td> 
</td><td>由于 SM0.0 固定为 1，因此 NE 总是执行：若 VB0 的值不等于 B#200，则 Q0.0 被置为 1，否则 Q0.0 就置为 0。</td></tr>
<tr><td> 
</td><td>若 I0.0 为 0：不执行 NE，Q0.0 被置为 0。
若 I0.0 为 1：若 VW0 的值不等于 VW2 的值，则 Q0.0 被置为 1，否则 Q0.0 被置为 0。</td></tr>
<tr><td>IL</td><td colspan="2">LD %SM0.0 (* 建立 CR，其值为 1 *)
NE %VB0, b#200 (* 若 VB0 不等于 B#200，则 CR 被置为 1，否则 CR 被置为 0 *)
ST %Q0.0 (* 将 CR 值赋给 Q0.0 *)</td></tr>
<tr><td></td><td colspan="2">LD %I0.0 (* 建立 CR，其值为 I0.0 的值 *)
NE %VW0, %VW2 (* 若 CR 为 1：则若 VW0 不等于 VW2，则 CR 被置为 1 *)
(* 否则 CR 被置为 0 *)
(* 若 CR 为 0：则不进行 NE 比较，CR 保持为 0 *)
ST %Q0.0 (* 将 CR 值赋给 Q0.0 *)</td></tr>
</table>

6.4.5 LT（小于）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	LT			
IL	LT	LT IN1, IN2	P	

参数	输入/输出	数据类型	允许的内存区
IN1	输入	BYTE、WORD、DWORD、INT、DINT、REAL	I、Q、M、V、L、SM、AI、AQ、T、C、HC、常量、指针
IN2	输入	BYTE、WORD、DWORD、INT、DINT、REAL	I、Q、M、V、L、SM、AI、AQ、T、C、HC、常量、指针
OUT（LD）	输出	BOOL	能量流

参数 *IN1*、*IN2* 的数据类型必须一致。

- LD

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

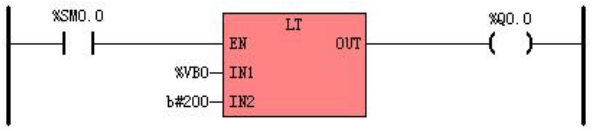
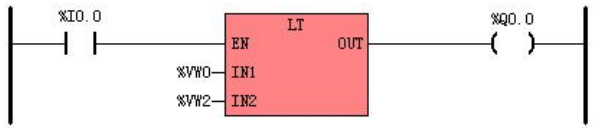
  如果 *EN* 为 0，该指令不执行，在 *OUT* 端输出为 0。

- IL

  如果 CR 值为 1，该指令被执行：若 *IN1* 小于 *IN2*，则将 CR 置为 1，否则将 CR 置为 0；

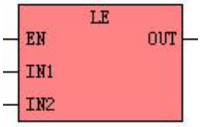
  如果 CR 值为 0，该指令不执行，CR 值保持为 0。

➢ 指令使用举例

<table>
<tr><td rowspan="2">LD</td><td> 
</td><td>由于 SM0.0 固定为 1，因此 LT 总是执行：若 VB0 的值小于 B#200，则 Q0.0 被置为 1，否则 Q0.0 就置为 0。</td></tr>
<tr><td> 
</td><td>若 I0.0 为 0：不执行 LT，Q0.0 被置为 0。
若 I0.0 为 1：若 VW0 的值小于 VW2 的值，则 Q0.0 被置为 1，否则 Q0.0 被置为 0。</td></tr>
<tr><td>IL</td><td colspan="2">LD %SM0.0 (* 建立 CR，其值为 1 *)
LT %VB0, b#200 (* 若 VB0 小于 B#200，则 CR 被置为 1，否则 CR 被置为 0 *)
ST %Q0.0 (* 将 CR 值赋给 Q0.0 *)</td></tr>
<tr><td></td><td colspan="2">LD %I0.0 (* 建立 CR，其值为 I0.0 的值 *)
LT %VW0, %VW2 (* 若 CR 为 1：则若 VW0 小于 VW2，则 CR 被置为 1 *)
(* 否则 CR 被置为 0 *)
(* 若 CR 为 0：则不执行 LT，CR 保持为 0 *)
ST %Q0.0 (* 将 CR 值赋给 Q0.0 *)</td></tr>
</table>

6.4.6 LE（小于等于）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	LE			
IL	LE	LE IN1, IN2	P	

参数	输入/输出	数据类型	允许的内存区
IN1	输入	BYTE、WORD、DWORD、INT、DINT、REAL	I、Q、M、V、L、SM、AI、AQ、T、C、HC、常量、指针
IN2	输入	BYTE、WORD、DWORD、INT、DINT、REAL	I、Q、M、V、L、SM、AI、AQ、T、C、HC、常量、指针
OUT（LD）	输出	BOOL	能量流

参数 *IN1*、*IN2* 的数据类型必须一致。

- LD

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

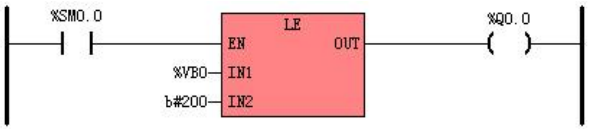
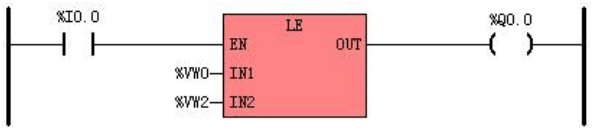
  如果 *EN* 为 0，该指令不执行，在 *OUT* 端输出为 0。

- IL

  如果 CR 值为 1，该指令被执行：若 *IN1* 小于等于 *IN2*，则将 CR 值置为 1，否则将 CR 值置为 0；

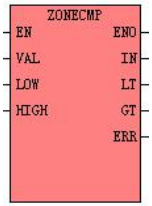
  如果 CR 值为 0，该指令不执行，CR 值保持为 0。

➢ 指令使用举例

<table>
<tr><td rowspan="2">LD</td><td> 
</td><td>由于 SM0.0 固定为 1，因此 LE 总是执行：若 VB0 的值小于等于 B#200，则 Q0.0 被置为 1，否则 Q0.0 就置为 0。</td></tr>
<tr><td> 
</td><td>若 I0.0 为 0：不执行 LE，Q0.0 被置为 0。
若 I0.0 为 1：若 VW0 的值小于等于 VW2 的值，则 Q0.0 被置为 1，否则 Q0.0 被置为 0。</td></tr>
<tr><td rowspan="2">IL</td><td colspan="2">LD %SM0.0 (* 建立 CR，其值为 1 *)
LE %VB0, b#200 (* 若 VB0 小于等于 B#200，则 CR 被置为 1，否则 CR 被置为 0 *)
ST %Q0.0 (* 将 CR 值赋给 Q0.0 *)</td></tr>
<tr><td colspan="2">LD %I0.0 (* 建立 CR，其值为 I0.0 的值 *)
LE %VW0, %VW2 (* 若 CR 为 1：则若 VW0 小于等于 VW2，则 CR 被置为 1 *)
(* 否则 CR 被置为 0 *)
(* 若 CR 为 0：则不执行 LE，CR 保持为 0 *)
ST %Q0.0 (* 将 CR 值赋给 Q0.0 *)</td></tr>
</table>

6.4.7 ZONECMP（数据区间比较）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	ZONECMP			☑ K2 ☑ KS ☑ MK ☑ K6
IL	ZONECMP	ZONECMP VAL, LOW, HIGH , IN , LT , GT, ERR	U	

参数	输入/输出	数据类型	允许的内存区
VAL	输入	BYTE、WORD、DWORD、 INT、DINT、REAL	I、Q、M、V、L、SM、AI、AQ、HC
LOW	输入	BYTE、WORD、DWORD、 INT、DINT、REAL	I、Q、M、V、L、SM、AI、AQ、HC、常量
HIGH	输入	BYTE、WORD、DWORD、 INT、DINT、REAL	I、Q、M、V、L、SM、AI、AQ、HC、常量
IN	输出	BOOL	Q、M、V、L、SM
LT	输出	BOOL	Q、M、V、L、SM
GT	输出	BOOL	Q、M、V、L、SM
ERR	输出	BOOL	Q、M、V、L、SM

注意， 参数 *VAL*、*LOW*、*HIGH* 的数据类型必须一致，并且 *LOW* <= *HIGH*。

该指令用于将 VAL 与 LOW、HIGH 进行比较，并将结果输出到相应的输出参数中：

- 若 *LOW* > *HIGH*，则 *ERR* 输出为 1，其它参数输出为 0；
- 若 *LOW* <= *VAL* <= *HIGH*，则 *IN* 输出为 1，其它参数输出为 0；

- 若 *VAL* < *LOW*，则 *LT* 输出为 1，其它参数输出为 0；
- 若 *VAL* > *HIGH*，则 *GT* 输出为 1，其它参数输出为 0。

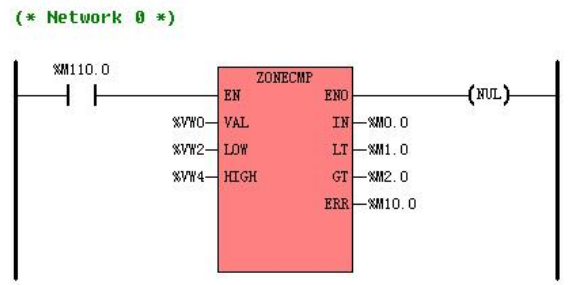
- LD

  如果 *EN* 为 1，该指令被执行。

- IL

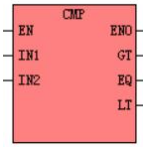
  如果 CR 值为 1，该指令被执行。该指令的执行不影响 CR 值。

➢ 指令使用举例

<table>
<tr><td>LD</td><td> 
</td><td>若 M110.0 为 0：不执行 ZONECMP，输出端保持原有状态。
若 M110.0 为 1:若 VW2<=VW0<=VW4 则 M0.0=TRUE；若 VW0<VW2 则 M1.0=TRUE；若 VW0>VW4 则 M2.0=TRUE；</td></tr>
<tr><td>IL</td><td colspan="2">LD %M110.0 (* 建立 CR，其值为 M110.0 的值 *)
ZONECMP %VW0, %VW2, %VW4, %M0.0, %M1.0, %M2.0, %M10.0
(*若 M110.0 为 0，则指令不执行，输出端保持原有状态。
若 M110.0 为 1，则指令执行：若 VW2<=VW0<=VW4，则 M0.0=TRUE；若 VW0<VW2 则 M1.0=TRUE；若 VW0>VW4 则 M2.0=TRUE *)</td></tr>
</table>

6.4.8 CMP（比较）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	CMP			☑ K2 ☑ KS ☑ MK ☑ K6
IL	CMP	CMP IN1，IN2，GT，EQ，LT	U	

参数	输入/输出	数据类型	允许的内存区
IN1	输入	BYTE、WORD、DWORD、 INT、DINT、REAL	I、Q、M、V、L、AI、AQ、HC
IN2	输入	BYTE、WORD、DWORD、 INT、DINT、REAL	I、Q、M、V、L、AI、AQ、HC、常量
GT	输出	BOOL	Q、M、V、L
EQ	输出	BOOL	Q、M、V、L
LT	输出	BOOL	Q、M、V、L

注意， 参数 *IN1*、*IN2*、*HIGH* 的数据类型必须一致。

该指令将 *IN1* 与 *IN2* 进行比较，并将结果输出到相应的输出参数中：

- 若 *IN1* 大于 *IN2*，则 *GT* 输出为 1，其它参数输出为 0。
- 若 *IN1* 等于 *IN2*，则 *EQ* 输出为 1，其它参数输出为 0。
- 若 *VAL* 小于 *LOW*，则 *LT* 输出为 1，其它参数输出为 0；

- LD

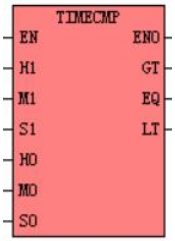
  如果 *EN* 为 1，该指令被执行。

- IL

  如果 CR 值为 1，该指令被执行。该指令的执行不影响 CR 值。

6.4.9 TIMECMP（时间比较）

➢ 指令及其操作数说明

	名称	指令格式	影响CR值	
LD	TIMECMP			☑ K2 ☑ KS ☑ MK ☑ K6
IL	TIMECMP	TIMECMP H1,M1,S1,H0,M0,S0,GT,EQ,LT	U	

参数	输入/输出	数据类型	允许的内存区
H1	输入	BYTE	M、V、L
M1	输入	BYTE	M、V、L
S1	输入	BYTE	M、V、L
H0	输入	BYTE	M、V、L、常量
M0	输入	BYTE	M、V、L、常量
S0	输入	BYTE	M、V、L、常量
GT	输出	BOOL	Q、M、V、L
EQ	输出	BOOL	Q、M、V、L
LT	输出	BOOL	Q、M、V、L

注意： 参数 *H0*、*M0*、*S0* 的必须同时采用变量或者常量。

该指令按 24 小时格式将 *H1*（时）、*M1*（分）、*S1*（秒）组成一个时间值（假定为 *Time1*），将 *H0*（时）、*M0*（分）、*S0*（秒）组成另一个时间值（假定为 *Time0*），然后比较 *Time1* 和 *Time0*，并将结果输出到相应的输出参数中。**用户必须输入合法的时间值，即时的取值范围为[0,23]，分的取值范围为[0,59]，秒的取值范围为[0,59]。**

- 若任何一个输入参数值非法，则所有参数输出为 0；
- 若 *Time1* 大于 *Time0*，则 *GT* 输出为 1，其它参数输出为 0；
- 若 *Time1* 等于 *Time0*，则 *EQ* 输出为 1，其它参数输出为 0；
- 若 *Time1* 小于 *Time0*，则 *LT* 输出为 1，其它输出参数为 0。

- LD

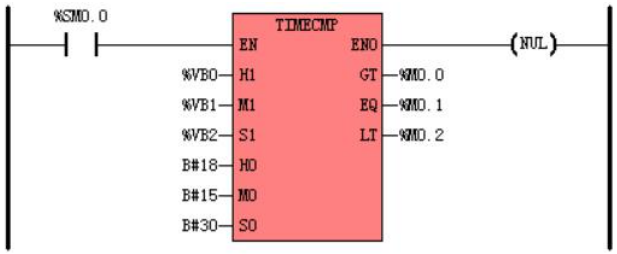
  如果 *EN* 为 1，则该指令被执行，否则不执行。

- IL

  如果 CR 值为 1，则该指令被执行，否则不执行。

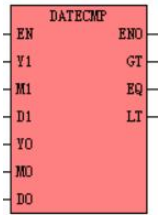
  该指令的执行不影响 CR 值。

➢ 指令使用举例

<table>
<tr><td rowspan="2">LD</td><td>(* Network 0 *)
(*用户可以利用 RTC_R 指令读取当前的时间值，并用于下面的比较中。*)
 
</td></tr>
<tr><td>SM0.0 恒为 TRUE，因此 TIMECMP 指令一直执行：用 VB0（时）、VB1（分）、VB2（秒）组成的时间值与时间 18:15:30 进行比较。假定 VB0=B#8,VB1=B#23,VB2=B#34,那么组成的时间值为 8:23:34，该时间小于 18:15:30，因此 M0.2 输出为 1，M0.0 和 M0.1 均输出为 0.</td></tr>
</table>

6.4.10 DATECMP（日期比较）

➢ 指令及其操作数说明

	名称	指令格式	影响CR值	
LD	DATECMP			☑ K2 ☑ KS ☑ MK ☑ K6
IL	DATECMP	DATECMP Y1, M1,D1,Y0,M0,D0,GT,EQ,LT	U	

参数	输入/输出	数据类型	允许的内存区
Y1	输入	BYTE	M、V、L
M1	输入	BYTE	M、V、L
D1	输入	BYTE	M、V、L
Y0	输入	BYTE	M、V、L、常量
M0	输入	BYTE	M、V、L、常量
D0	输入	BYTE	M、V、L、常量
GT	输出	BOOL	Q、M、V、L
EQ	输出	BOOL	Q、M、V、L
LT	输出	BOOL	Q、M、V、L

注意： 参数 *Y0*、*M0*、*D0* 的必须同时采用变量或者常量。

输入参数 *Y1* 和 *Y0* 均表示以 2000 年作为基准的年份值，即：真实的年份值 = 2000 + 参数值。例如，假定 *Y1* 值为 26，则表示 2026 年；假定 *Y0* 值为 16，则表示 2016 年。

该指令将 *Y1*（年）、*M1*（月）、*D1*（日）组成一个日期值（假定为 *Date1*），将 *Y0*（年）、*M0*（月）、*D0*（日）组成另一个时间值（假定为 *Date0*），然后比较 *Date1* 和 *Date0*，并将结果输出到相应的输出参数中。**用户必须输入合法的日期值，即月的取值范围为[1,12]，日的取值范围**

为[1,31]，但本指令仅判断各个单一的数值是否合法，不会判断一个完整的日期是否合法，比如，“2021 年 2 月 30 日”这个日期虽然不存在，但本指令也不会判错。

- 若任何一个输入参数值非法，则所有参数输出为 0；
- 若 *Date1* 大于 *Date0*，则 *GT* 输出为 1，其它参数输出为 0；
- 若 *Date1* 等于 *Date0*，则 *EQ* 输出为 1，其它参数输出为 0；
- 若 *Date1* 小于 *Date0*，则 *LT* 输出为 1，其它输出参数为 0。

- LD

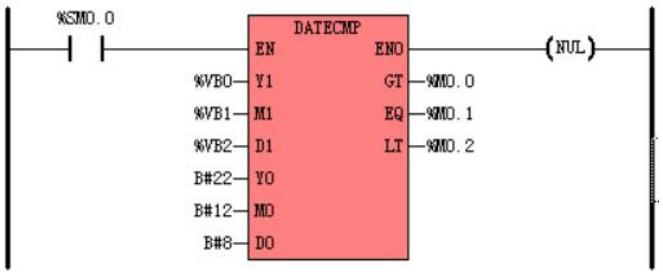
  如果 *EN* 为 1，则该指令被执行，否则不执行。

- IL

  如果 CR 值为 1，则该指令被执行，否则不执行。

  该指令的执行不影响 CR 值。

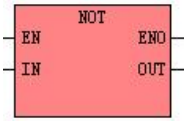
➢ 指令使用举例

<table>
<tr><td>LD</td><td>(* Network 0 *)
(*用户可以利用 RTC_R 指令读取当前的日期值，并用于下面的比较中。*)
 
</td></tr>
<tr><td></td><td>SM0.0 恒为 TRUE，因此 DATECMP 指令一直执行：用 VB0（年）、VB1（月）、VB2（日）组成的日期值与日期“2022 年 12 月 8 日”进行比较。假定 VB0=B#8,VB1=B#11,VB2=B#21,那么组成的日期“2008 年 11 月 21 日”，该日期小于“2022 年 12 月 8 日”，因此 M0.2 输出为 1，M0.0 和 M0.1 均输出为 0.</td></tr>
</table>

6.5 逻辑运算

6.5.1 NOT（按位取反）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	NOT			
IL	NOT	NOT *OUT*	U	

参数	输入/输出	数据类型	允许的内存区
IN	输入	BYTE、WORD、DWORD	I、Q、M、V、L、SM、常量、指针
OUT	输出	BYTE、WORD、DWORD	Q、M、V、L、SM、指针

- LD

  参数 *IN*、*OUT* 的数据类型必须一致。

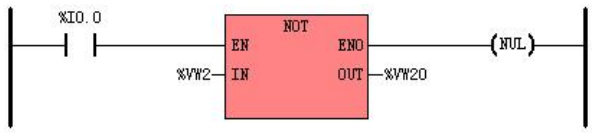
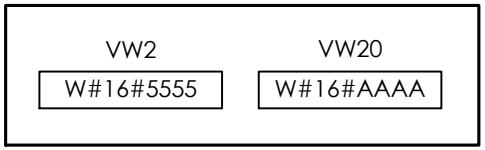
  如果 *EN* 为 1，则该指令被执行：将 *IN* 的每一个二进制位都取反，然后将结果赋给 *OUT*。

- IL

  如果 CR 值为 1，则该指令被执行：将 *OUT* 的每一个二进制位都取反，结果仍旧赋给 *OUT*。

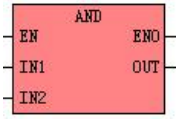
  该指令的执行不影响 CR 值。

➢ 指令使用举例

<table>
<tr><td>LD</td><td> 
</td><td>若 I0.0 为 0：不执行 NOT 指令。
若 I0.0 为 1：将 VW2 的值按位取反，并将结果赋给 VW20。</td></tr>
<tr><td>IL</td><td colspan="2">LD %I0.0 (* 建立 CR，其值为 I0.0 的值 *)
NOT %VW20 (* 若 CR 为 1：将 VW20 按位取反，结果仍放于 VW20 中 *)
(* 若 CR 为 0：不执行 NOT 指令 *)</td></tr>
<tr><td>结果</td><td colspan="2">参照上面的 LD 例子，若 NOT 指令被执行，则结果举例如下：
 
</td></tr>
</table>

6.5.2 AND（按位与）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	AND			
IL	AND	AND IN1, OUT	U	

参数	输入/输出	数据类型	允许的内存区
IN1	输入	BYTE、WORD、DWORD	I、Q、M、V、L、SM、常量、指针
IN2	输入	BYTE、WORD、DWORD	I、Q、M、V、L、SM、常量、指针
OUT	输出	BYTE、WORD、DWORD	Q、M、V、L、SM、指针

- LD

  参数 *IN1*、*IN2*、*OUT* 的数据类型必须一致。

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

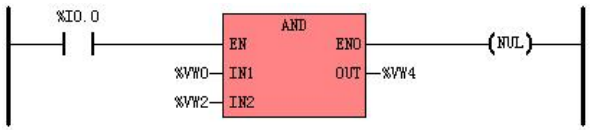
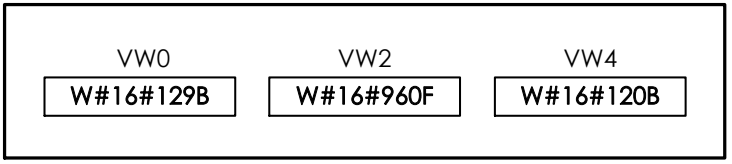
- IL

  参数 *IN1*、*OUT* 的数据类型必须一致。

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

  该指令的执行不影响 CR 值。

➢ 指令使用举例

<table>
<tr><td>LD</td><td> 
</td><td>若 I0.0 为 0：不执行 AND 指令。
若 I0.0 为 1：将 VW0 和 VW2 的值按位进行“与”运算，并将结果赋给 VW4。</td></tr>
<tr><td>IL</td><td colspan="2">LD %I0.0 (* 建立 CR，其值为 I0.0 的值 *)
AND %VW0, %VW2 (* 若 CR 为 1：将 VW0 和 VW2 的值按位相“与”，结果仍放于 VW2 中 *)
(* 若 CR 为 0：不执行 AND 指令 *)</td></tr>
<tr><td>结果</td><td colspan="2">参照上面的 LD 例子，若 AND 指令被执行，则结果举例如下：
 
</td></tr>
</table>

6.5.3 ANDN（按位与非）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	ANDN			
IL	ANDN	ANDN *IN1, OUT*	U	

参数	输入/输出	数据类型	允许的内存区
IN1	输入	BYTE、WORD、DWORD	I、Q、M、V、L、SM、常量、指针
IN2	输入	BYTE、WORD、DWORD	I、Q、M、V、L、SM、常量、指针
OUT	输出	BYTE、WORD、DWORD	Q、M、V、L、SM、指针

- LD

参数 *IN1*、*IN2*、*OUT* 的数据类型必须一致。

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

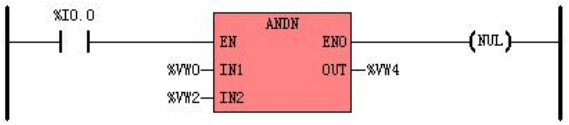
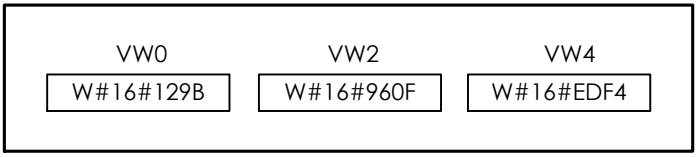
- IL

参数 *IN1*、*OUT* 的数据类型必须一致。

若 CR 值为 1，则该指令被执行：将 *IN1* 和 *OUT* 按二进制位进行“与”运算并取反，然后将结果赋给 *OUT*。

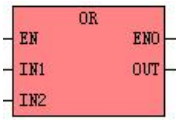
该指令的执行不影响 CR 值。

➢ 指令使用举例

<table>
<tr><td>LD</td><td> 
</td><td>若 I0.0 为 0：不执行 ANDN 指令。
若 I0.0 为 1：将 VW0 和 VW2 的值按位进行“与”运算并取反，最终的结果赋给 VW4。</td></tr>
<tr><td>IL</td><td colspan="2">LD %I0.0 (* 建立 CR，其值为 I0.0 的值 *)
ANDN %VW0, %VW2 (* 若 CR 为 1:将 VW0 和 VW2 的值按位相与并取反,结果仍放于 VW2 中 *)
(* 若 CR 为 0：不执行 ANDN 指令 *)</td></tr>
<tr><td>结果</td><td colspan="2">参照上面的 LD 例子，若 ANDN 指令被执行，则结果举例如下：
 
</td></tr>
</table>

6.5.4 OR（按位或）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	OR			
IL	OR	OR IN1, OUT	U	

参数	输入/输出	数据类型	允许的内存区
IN1	输入	BYTE、WORD、DWORD	I、Q、M、V、L、SM、常量、指针
IN2	输入	BYTE、WORD、DWORD	I、Q、M、V、L、SM、常量、指针
OUT	输出	BYTE、WORD、DWORD	Q、M、V、L、SM、指针

- LD

  参数 *IN1*、*IN2*、*OUT* 的数据类型必须一致。

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

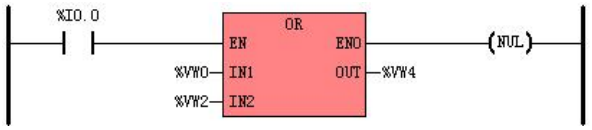
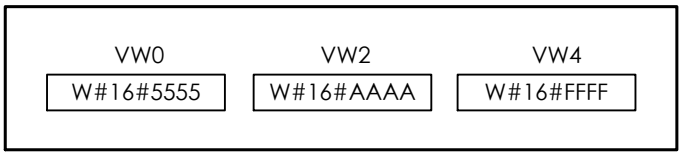
- IL

  参数 *IN1*、*OUT* 的数据类型必须一致。

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

  该指令的执行不影响 CR 值。

➢ 指令使用举例

<table>
<tr><td>LD</td><td> 
</td><td>若 I0.0 为 0：不执行 OR 指令。
若 I0.0 为 1：将 VW0 和 VW2 的值按位进行“或”运算，并将结果赋给 VW4。</td></tr>
<tr><td>IL</td><td colspan="2">LD %I0.0 (* 建立 CR，其值为 I0.0 的值 *)
OR %VW0, %VW2 (* 若 CR 为 1：将 VW0 和 VW2 的值按位相或，结果仍放于 VW2 中 *)
(* 若 CR 为 0：不执行 OR 指令 *)</td></tr>
<tr><td>结果</td><td colspan="2">参照上面的 LD 例子，若 OR 指令被执行，则结果举例如下：
 
</td></tr>
</table>

6.5.5 ORN（按位或非）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	ORN			
IL	ORN	ORN　　*IN1, OUT*	U	

参数	输入/输出	数据类型	允许的内存区
IN1	输入	BYTE、WORD、DWORD	I、Q、M、V、L、SM、常量、指针
IN2	输入	BYTE、WORD、DWORD	I、Q、M、V、L、SM、常量、指针
OUT	输出	BYTE、WORD、DWORD	Q、M、V、L、SM、指针

- LD

参数 *IN1*、*IN2*、*OUT* 的数据类型必须一致。

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

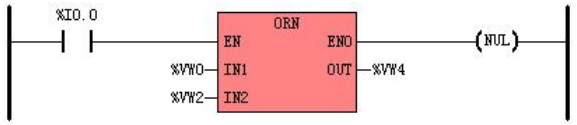
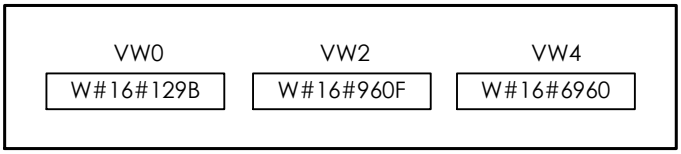
- IL

参数 *IN*、*OUT* 的数据类型必须一致。

若 CR 值为 1，则该指令被执行：将 *IN1* 和 *OUT* 按二进制位进行“或”运算并取反，然后将结果赋给 *OUT*。

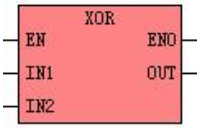
该指令的执行不影响 CR 值。

➢ 指令使用举例

<table>
<tr><td>LD</td><td> 
</td><td>若 I0.0 为 0：不执行 ORN 指令。
若 I0.0 为 1:将 VW0 和 VW2 的值按位进行“或”运算并取反，最终的结果赋给 VW4。</td></tr>
<tr><td>IL</td><td colspan="2">LD %I0.0 (* 建立 CR，其值为 I0.0 的值 *)
ORN %VW0, %VW2 (* 若 CR 为 1：将 VW0 和 VW2 的值按位相或并取反，结果仍放于 VW2 中 *)
(* 若 CR 为 0：不执行 ORN 指令 *)</td></tr>
<tr><td>结果</td><td colspan="2">参照上面的 LD 例子，若 ORN 指令被执行，则结果举例如下：
 
</td></tr>
</table>

6.5.6 XOR（按位异或）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	XOR			
IL	XOR	XOR IN1, OUT	U	

参数	输入/输出	数据类型	允许的内存区
IN1	输入	BYTE、WORD、DWORD	I、Q、M、V、L、SM、常量、指针
IN2	输入	BYTE、WORD、DWORD	I、Q、M、V、L、SM、常量、指针
OUT	输出	BYTE、WORD、DWORD	Q、M、V、L、SM、指针

- LD

  参数 *IN1*、*IN2*、*OUT* 的数据类型必须一致。

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

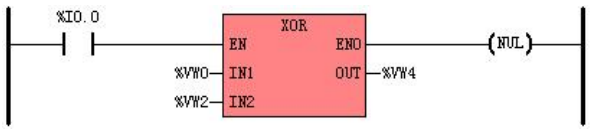
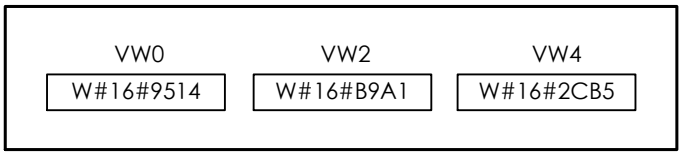
- IL

  参数 *IN1*、*OUT* 的数据类型必须一致。

  若 CR 值为 1，则该指令被执行：将 *IN1* 和 *OUT* 按二进制位进行“异或”运算后，将结果赋给 *OUT*。

  该指令的执行不影响 CR 值。

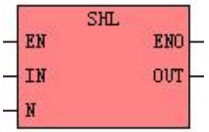
➢ 指令使用举例

<table>
<tr><td>LD</td><td> 
</td><td>若 I0.0 为 0：不执行 XOR 指令。
若 I0.0 为 1：将 VW0 和 VW2 的值按位进行“异或”运算，并将结果赋给 VW4。</td></tr>
<tr><td>IL</td><td colspan="2">LD %I0.0 (* 建立 CR，其值为 I0.0 的值 *)
XOR %VW0, %VW2 (* 若 CR 为 1：将 VW0 和 VW2 的值按位相“异或”，结果仍放于 VW2 中 *)
(* 若 CR 为 0：不执行 XOR 指令 *)</td></tr>
<tr><td>结果</td><td colspan="2">参照上面的 LD 例子，若 XOR 指令被执行，则结果举例如下：
 
</td></tr>
</table>

6.6 移位指令

6.6.1 SHL（左移）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	SHL			
IL	SHL	SHL *OUT, N*	U	

参数	输入/输出	数据类型	允许的内存区
IN	输入	BYTE、WORD、DWORD	I、Q、M、V、L、SM、常量、指针
N	输入	BYTE	I、Q、M、V、L、SM、常量、指针
OUT	输出	BYTE、WORD、DWORD	Q、M、V、L、SM、指针

- LD

参数 *IN*、*OUT* 的数据类型必须一致。

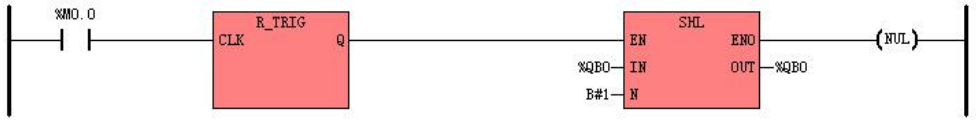
若 *EN* 为 1，则该指令被执行：将 *IN* 的全部二进制位向左移动 *N* 位，移出的高位被舍弃并且低位补 0，最终的结果赋给 *OUT*。

- IL

若 CR 值为 1，则该指令被执行：将 *OUT* 的全部二进制位向左移动 *N* 位，移出的高位被舍弃并且低位补 0，最终的结果仍旧被赋给 *OUT*。

该指令的执行不影响 CR 值。

➢ 指令使用举例

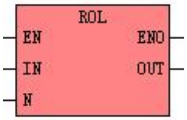
<table>
<tr><td rowspan="2">LD</td><td> 
</td></tr>
<tr><td>在 M0.0 的每个上升沿，都执行一次 SHL 指令，将 QB0 值的全部二进制位左移 1 位，并将结果仍赋给 QB0。</td></tr>
<tr><td>IL</td><td>LD %M0.0
R_TRIG (* 取 M0.0 的上升沿，并将结果作为 CR 值 *)
SHL %QB0, B#1 (* 若 CR 为 1：将 QB0 值的全部二进制位左移 1 位，结果仍放于 QB0 中 *)
(* 若 CR 为 0：不执行 SHL 指令 *)</td></tr>
<tr><td>结果</td><td>若 SHL 指令被执行，则结果举例如下：

QB0初值： B#2#10000001

第一次移位后 第二次移位后 第三次移位后 第四次移位后
QB0值： B#2#00000010 B#2#00000100 B#2#00001000 B#2#00010000</td></tr>
</table>

6.6.2 ROL（循环左移）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	ROL			
IL	ROL	ROL *OUT, N*	U	

参数	输入/输出	数据类型	允许的内存区
IN	输入	BYTE、WORD、DWORD	I、Q、M、V、L、SM、常量、指针
N	输入	BYTE	I、Q、M、V、L、SM、常量、指针
OUT	输出	BYTE、WORD、DWORD	Q、M、V、L、SM、指针

- LD

参数 *IN*、*OUT* 的数据类型必须一致。

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

- IL

若 CR 值为 1，则该指令被执行：将 *OUT* 的全部二进制位向左移动 *N* 位，移出的高位被依次移进低位位置，最终的结果仍旧被赋给 *OUT*。

该指令的执行不影响 CR 值。

➢ 指令使用举例

<table>
<tr><td>LD</td><td> 
</td></tr>
<tr><td></td><td>在 M0.0 的每个上升沿，都执行一次 ROL 指令，将 QB0 值的全部二进制位每次左移 1 位，移出的高位依次补到低位位置，并将结果仍赋给 QB0。</td></tr>
<tr><td>IL</td><td>LD %M0.1
R_TRIG (* 取 M0.0 的上升沿，并将结果作为 CR 值 *)
ROL %QB0, B#1 (* 若 CR 为 1：将 QB0 值的全部二进制位循环左移 1 位，结果仍放于 QB0 中 *)
(* 若 CR 为 0：不执行 SHL 指令 *)</td></tr>
<tr><td>结果</td><td>若 ROL 指令被执行，则结果举例如下：

QB0初值： B#2#10100001

第一次移位后 第二次移位后 第三次移位后 第四次移位后
QB0值： B#2#01000011 B#2#10000110 B#2#00001101 B#2#00011010</td></tr>
</table>

6.6.3 SHR（右移）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	SHR			
IL	SHR	SHR *OUT, N*	U	

参数	输入/输出	数据类型	允许的内存区
IN	输入	BYTE、WORD、DWORD	I、Q、M、V、L、SM、常量、指针
N	输入	BYTE	I、Q、M、V、L、SM、常量、指针
OUT	输出	BYTE、WORD、DWORD	Q、M、V、L、SM、指针

- LD

参数 *IN*、*OUT* 的数据类型必须一致。

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

- IL

若 CR 值为 1，则该指令被执行：将 *OUT* 的全部二进制位向右移动 *N* 位，移出的低位被舍弃并且高位补 0，最终的结果仍旧被赋给 *OUT*。

该指令的执行不影响 CR 值。

➢ 指令使用举例

<table>
<tr><td rowspan="2">LD</td><td> 
</td></tr>
<tr><td>在 M0.0 的每个上升沿，都执行一次 SHR 指令，将 QB0 值的全部二进制位右移 1 位，并将结果仍赋给 QB0。</td></tr>
<tr><td>IL</td><td>LD %M0.0
R_TRIG (* 取 M0.0 的上升沿，并将结果作为 CR 值 *)
SHR %QB0, B#1 (* 若 CR 为 1：将 QB0 值的全部二进制位右移 1 位，结果仍放于 QB0 中 *)
(* 若 CR 为 0：不执行 SHL 指令 *)</td></tr>
<tr><td>结果</td><td>若 SHR 指令被执行，则结果举例如下：

QB0初值： B#2#10000001

第一次移位后 第二次移位后 第三次移位后 第四次移位后
QB0值： B#2#01000000 B#2#00100000 B#2#00010000 B#2#00001000</td></tr>
</table>

6.6.4 ROR（循环右移）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	ROR			
IL	ROR	ROR *OUT, N*	U	

参数	输入/输出	数据类型	允许的内存区
IN	输入	BYTE、WORD、DWORD	I、Q、M、V、L、SM、常量、指针
N	输入	BYTE	I、Q、M、V、L、SM、常量、指针
OUT	输出	BYTE、WORD、DWORD	Q、M、V、L、SM、指针

- LD

参数 *IN*、*OUT* 的数据类型必须一致。

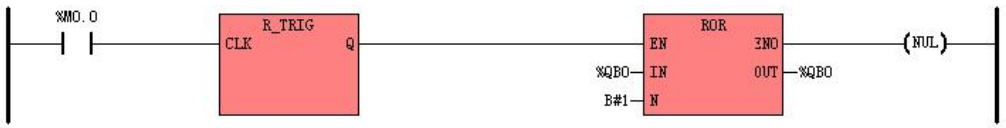
若 *EN* 为 1，则该指令被执行：将 *IN* 的全部二进制位向右移动 *N* 位，移出的低位被依次移进高位位置，最终的结果赋给 *OUT*。

- IL

若 CR 值为 1，则该指令被执行：将 *OUT* 的全部二进制位向右移动 *N* 位，移出的低位被依次移进高位位置，最终的结果仍旧被赋给 *OUT*。

该指令的执行不影响 CR 值。

➢ 指令使用举例

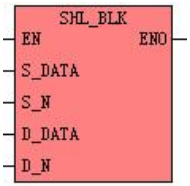
<table>
<tr><td rowspan="2">LD</td><td> 
</td></tr>
<tr><td>在 M0.0 的每个上升沿，都执行一次 ROR 指令，将 QB0 值的全部二进制位每次右移 1 位，移出的低位被依次补到高位位置，并将结果仍赋给 QB0。</td></tr>
<tr><td>IL</td><td>LD %M0.1
R_TRIG (* 取 M0.0 的上升沿，并将结果作为 CR 值 *)
ROR %QB0, B#1 (* 若 CR 为 1：将 QB0 值的全部二进制位循环右移 1 位，结果仍放于 QB0 中 *)
(* 若 CR 为 0：不执行 SHL 指令 *)</td></tr>
<tr><td>结果</td><td>若 ROR 指令被执行，则结果举例如下：

QB0初值： B#2#10100001

第一次移位后 第二次移位后 第三次移位后 第四次移位后
QB0值： B#2#11010000 B#2#01101000 B#2#00110100 B#2#00011010</td></tr>
</table>

6.6.5 SHL_BLK（位串左移）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	SHL_BLK			
IL	SHL_BLK	SHL_BLK S_DATA, S_N, D_DATA, D_N	U	

参数	输入/输出	数据类型	允许的内存区
S_DATA	输入	BOOL	I、Q、M、V、L
S_N	输入	INT	I、Q、V、M、L、SM、T、C、AI、AQ、常量、指针
D_DATA	输入/输出	BOOL	Q、M、V、L
D_N	输入	INT	I、Q、V、M、L、SM、T、C、AI、AQ、常量、指针

S_N, D_N 参数最大值为 1024，当输入大于 1024 时，取 1024。当 S_N 大于 D_N，取 S_N = D_N。 S_N, D_N 都必须大于 0。

注意， S_DATA, D_DATA 参数为一个可变长度的块内存参数， 整个块内存都不可以落在非法内存区域， 否则结果不可预期。

该指令执行时，将从 *D_DATA* 开始的连续 *D_N* 个二进制位向左移动 *S_N* 位，移出的高位被丢弃，同时将从 *S_DATA* 开始的连续 *S_N* 个二进制位补入 *D_DATA* 的最右端。

- LD

  若 *EN* 为 1，则该指令被执行。

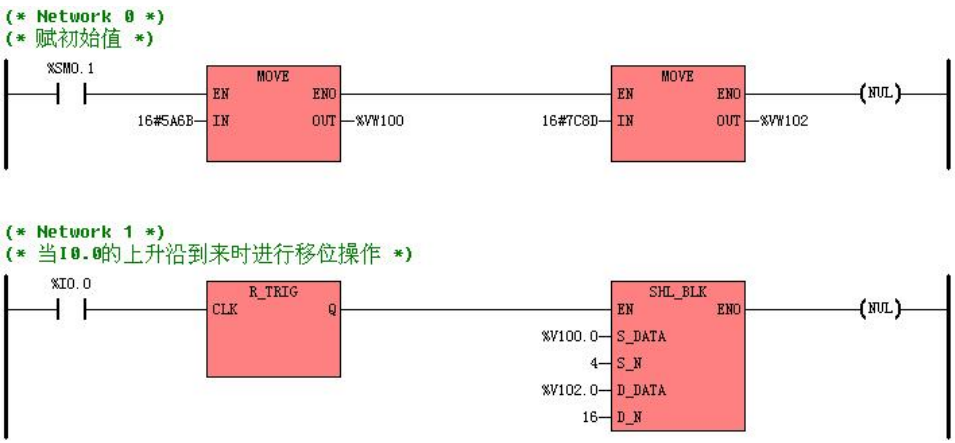
- IL

  若 CR 值为 1，则该指令被执行。

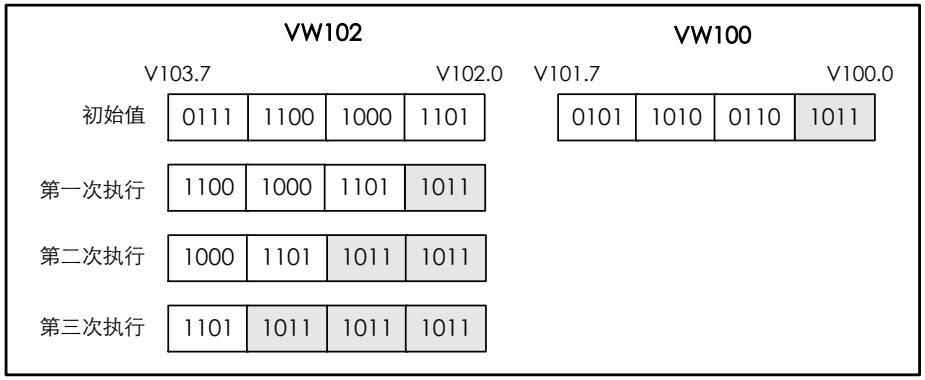
  该指令的执行不影响 CR 值。

➢ 指令使用举例

LD

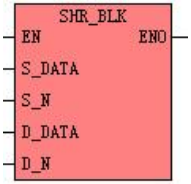


结果



6.6.6 SHR_BLK（位串右移）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	SHR_BLK			
IL	SHR_BLK	SHR_BLK S_DATA, S_N, D_DATA, D_N	U	

参数	输入/输出	数据类型	允许的内存区
S_DATA	输入	BOOL	I、Q、M、V、L
S_N	输入	INT	I、Q、V、M、L、SM、T、C、AI、AQ、常量、指针
D_DATA	输入/输出	BOOL	Q、M、V、L
D_N	输入	INT	I、Q、V、M、L、SM、T、C、AI、AQ、常量、指针

S_N, D_N 参数最大值为 1024， 当输入大于 1024 时， 取 1024。 当 S_N 大于 D_N 时， 取 S_N = D_N。 S_N, D_N 都必须大于 0。

注意， S_DATA, D_DATA 参数为一个可变长度的块内存参数， 整个块内存都不可以落在非法内存区域， 否则结果不可预期。

该指令执行时，将从 *D_DATA* 开始的连续 *D_N* 个二进制位向右移动 *S_N* 位，移出的低位被丢弃，同时将从 *S_DATA* 开始的连续 *S_N* 个二进制位补入 *D_DATA* 的最左端。

- LD

  若 *EN* 为 1，则该指令被执行。

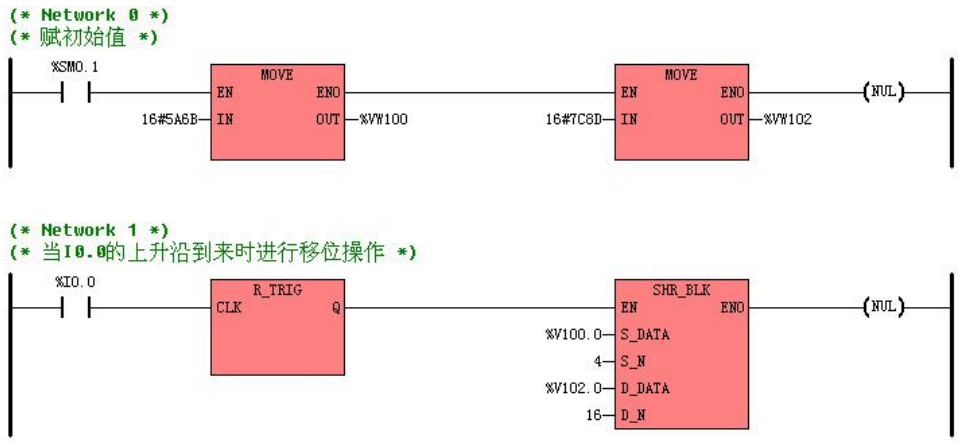
- IL

  若 CR 值为 1，则该指令被执行。

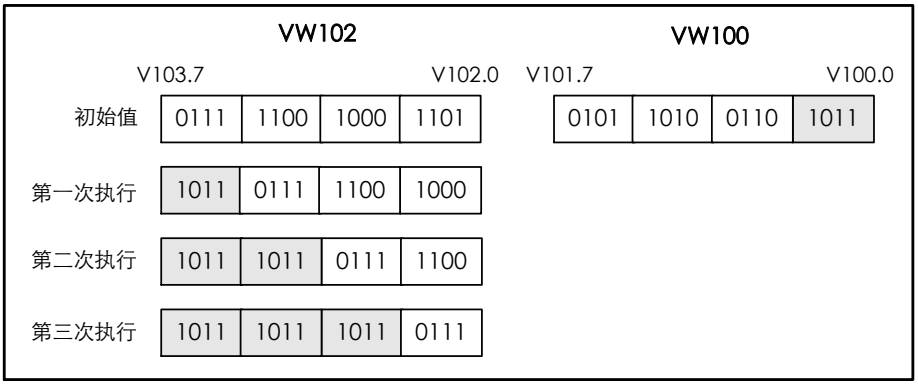
  该指令的执行不影响 CR 值。

➢ 指令使用举例

LD

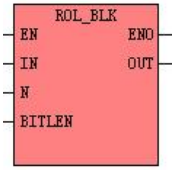


结果



6.6.7 ROL_BLK（位串循环左移）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	ROL_BLK			
IL	ROL_BLK	ROL_BLK *IN, N, BITLEN , OUT*	U	

参数	输入/输出	数据类型	允许的内存区
IN	输入	BOOL	I、Q、M、V、L、SM
N	输入	WORD	I、Q、M、V、L、SM、常量、指针
BITLEN	输入	WORD	I、Q、M、V、L、SM、常量、指针
OUT	输出	BOOL	Q、M、V、L、SM

- LD

参数 *IN*、*OUT* 的数据类型必须一致。

若 *EN* 为 1，则该指令被执行：将从 *IN* 开始的 *BITLEN* 个位向左移动 *N* 位，移出的高位被依次移进低位位置，最终的结果依次赋值给从 *OUT* 开始的 *BITLEN* 个位。

- IL

若 CR 值为 1，则该指令被执行：将从 *IN* 开始的 *BITLEN* 个位向左移动 *N* 位，移出的高位被依次移进低位位置，最终的结果依次赋值给从 *OUT* 开始的 *BITLEN* 个位。

该指令的执行不影响 CR 值。

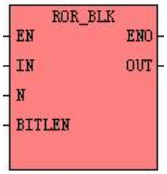
➢ 指令使用举例

<table>
<tr><td rowspan="2">LD</td><td> 
</td></tr>
<tr><td>在 M0.0 的每个上升沿，都执行一次 ROL_BLK 指令：将从 V0.0 开始的 8 个位（即 V0.0 至 V0.7）数据左移 1 位，移出的高位被依次补到低位位置，然后将结果仍赋给从 V0.0 开始的 8 个位（即 V0.0 至 V0.7）。</td></tr>
<tr><td>结果</td><td>假定从 V0.0 开始的 8 个的位数据初值：2#10010001

第一次移位后 第二次移位后 第三次移位后
结果从 V0.0 开始的 8 个位数据值：2#00100011 2#01000110 2#10001100</td></tr>
</table>

6.6.8 ROR_BLK（位串循环右移）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	ROR_BLK			
IL	ROR_BLK	ROR_BLK *IN, N, BITLEN , OUT*	U	

参数	输入/输出	数据类型	允许的内存区
IN	输入	BOOL	I、Q、M、V、L、SM
N	输入	WORD	I、Q、M、V、L、SM、常量、指针
BITLEN	输入	WORD	I、Q、M、V、L、SM、常量、指针
OUT	输出	BOOL	Q、M、V、L、SM

- LD

参数 *IN*、*OUT* 的数据类型必须一致。

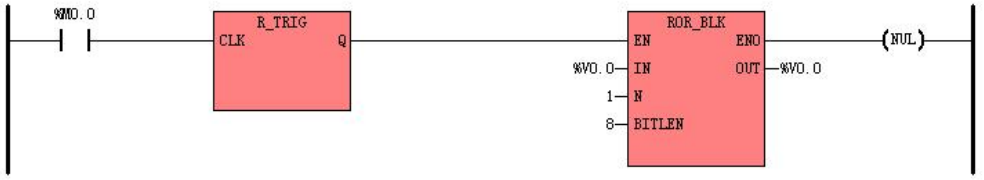
若 *EN* 为 1，则该指令被执行：将从 *IN* 开始的 *BITLEN* 个位向右移动 *N* 位，移出的低位被依次移进高位位置，最终的结果依次赋值给从 *OUT* 开始的 *BITLEN* 个位。

- IL

若 CR 值为 1，则该指令被执行：将从 *IN* 开始的 *BITLEN* 个位向右移动 *N* 位，移出的低位被依次移进高位位置，最终的结果依次赋值给从 *OUT* 开始的 *BITLEN* 个位。

该指令的执行不影响 CR 值。

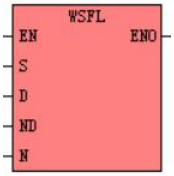
➢ 指令使用举例

<table>
<tr><td rowspan="2">LD</td><td> 
</td></tr>
<tr><td>在 M0.0 的每个上升沿，都执行一次 ROR_BLK 指令：将从 V0.0 开始的 8 个位（即 V0.0 至 V0.7）数据右移 1 位，移出的低位被依次补到高位位置，然后将结果仍赋给从 V0.0 开始的 8 个位（即 V0.0 至 V0.7）。</td></tr>
<tr><td>结果</td><td>假定从 V0.0 开始的 8 个的位数据初值：2#10010001

第一次移位后 第二次移位后 第三次移位后
结果从 V0.0 开始的 8 个位数据值：2#11001000 2#01100100 2#10110010</td></tr>
</table>

6.6.9 WSFL（字左移）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	WSFL			☑ K2 ☑ KS ☑ MK ☑ K6
IL	WSFL	WSFL *S*, *D*, *ND*,*N*	U	

参数	输入/输出	数据类型	允许的内存区
S	输入	WORD、INT	V、M、L
D	输入/输出	WORD、INT	V、M、L
ND	输入	INT	V、M、L、常量
N	输入	INT	V、M、L、常量

_S_和 _D_的数据类型必须一致；_ND_和 _N_必须同时为常量或者变量。

参数 _ND_和 _N_的值必须满足如下条件：0 < _N_ <= _ND_ <= 512。

本指令用于将以 *D* 为起始地址的连续 *ND* 个字与以 *S* 为起始地址的连续 *N* 个字联合向左（即高地址方向）移动 *N* 次（每次移动 1 个字），高地址移出的字将被舍弃。即，以 *D* 为起始地址的连续 *ND* 个字向左移动 *N* 次，高地址的 *N* 个字将被移出丢弃，低地址的 *N* 个字将依次用以 *S* 为起始地址的连续 *N* 个字补上。**注意：参与移动的所有内存变量都必须位于合法的地址范围内！**

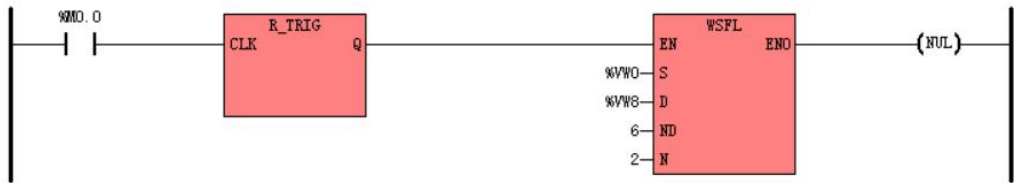
- LD

  若 *EN* 为 1，则该指令被执行。

- IL

  若 CR 值为 1，则该指令被执行。该指令的执行不影响 CR 值。

➢ 指令使用举例

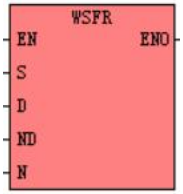
LD	
	当检测到 M0.0 的上升沿时，WSFL 被执行一次：从 VW8 开始得连续 6 个字向左（高地址方向）移动 2 次，最高的 2 个字（VW18 和 VW16）的值将被丢弃，最低的 2 个字（VW8 和 VW10）的值将分别用 VW0、VW2 的值替换。

执行结果举例如下，表格中的数字表示相应内存地址中的值。其中指令执行第 1 次的结果用不同的颜色来说明了数值来源。

指令执行次数	D						S	
	VW18	VW16	VW14	VW12	VW10	VW8	VW2	VW0
初始值	18	16	14	12	10	8	2	1
第 1 次	14	12	10	8	2	1	2	1
第 2 次	10	8	2	1	2	1	2	1

6.6.10 WSFR（字右移）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	WSFR			☑ K2 ☑ KS ☑ MK ☑ K6
IL	WSFR	WSFR *S, D, ND,N*	U	

参数	输入/输出	数据类型	允许的内存区
S	输入	WORD、INT	V、M、L
D	输入/输出	WORD、INT	V、M、L
ND	输入	INT	V、M、L、常量
N	输入	INT	V、M、L、常量

***S*和 *D*的数据类型必须一致；*ND*和 *N*必须同时为常量或者变量。**

参数 *ND*和 *N*的值必须满足如下条件：0 < *N* <= *ND* <= 512。

本指令用于将以 *D*为起始地址的连续 *ND*个字与以 *S*为起始地址的连续 *N*个字联合向右（即低地址方向）移动 *N*次（每次移动 1 个字），低地址移出的字将被舍弃。即，以 *D*为起始地址的连续 *ND*个字向右移动 *N*次，低地址的 *N*个字将被移出丢弃，高地址的 *N*个字将依次用以 *S*为起始地址的连续 *N*个字补上。**注意：参与移动的所有内存变量都必须位于合法的地址范围内！**

- LD

  若 *EN*为 1，则该指令被执行。

- IL

  若 CR 值为 1，则该指令被执行。该指令的执行不影响 CR 值。

➢ 指令使用举例

LD	
	当检测到 M0.0 的上升沿时，WSFL 被执行一次：从 VW8 开始得连续 6 个字向右（低地址方向）移动 2 次，最低的 2 个字（VW10 和 VW8）的值将被丢弃，最高的 2 个字（VW18 和 VW16）的值将分别用 VW2、VW0 的值替换。

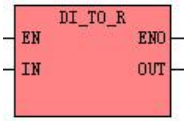
执行结果举例如下，表格中的数字表示相应内存地址中的值。其中指令执行第 1 次的结果用不同的颜色说明了数值来源。

指令执行次数	D						S	
	VW18	VW16	VW14	VW12	VW10	VW8	VW2	VW0
初始值	18	16	14	12	10	8	2	1
第 1 次	2	1	18	16	14	12	2	1
第 2 次	2	1	2	1	18	16	2	1

6.7 类型转换

6.7.1 DI_TO_R（双整型转实型）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	DI_TO_R			
IL	DI_TO_R	DI_TO_R *IN, OUT*	U	

参数	输入/输出	数据类型	允许的内存区
IN	输入	DINT	I、Q、M、V、L、SM、HC、常量
OUT	输出	REAL	V、L

该指令将输入的双整数 *IN*转换为实数并赋给 *OUT*。

- LD

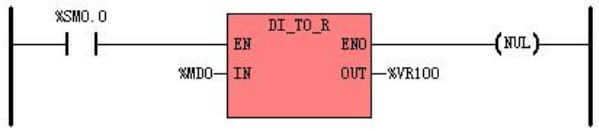
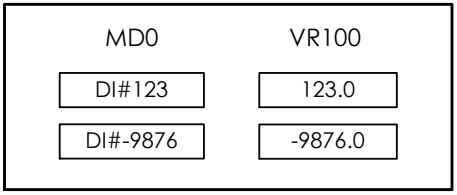
  如果 *EN*为 1，则该指令被执行。

- IL

  如果 CR 值为 1，则该指令被执行。

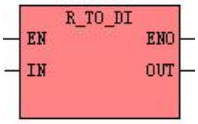
  该指令的执行不影响 CR 值。

➢ 指令使用举例

<table>
<tr><td>LD</td><td> 
</td><td>SM0.0 恒为 1，因此 DI_TO_R 指令总是执行：将 MD0 的值转换为实数并赋给 VR100。</td></tr>
<tr><td>IL</td><td colspan="2">LD %SM0.0 (* 建立 CR，其值为 1 *)
DI_TO_R %MD0, %VR100 (* 将 MD0 的值转换为实数并赋给 VR100 *)</td></tr>
<tr><td>结果</td><td colspan="2">结果举例如下：
 
</td></tr>
</table>

6.7.2 R_TO_DI（实型转双整型）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	R_TO_DI			
IL	R_TO_DI	R_TO_DI *IN, OUT*	U	

参数	输入/输出	数据类型	允许的内存区
IN	输入	REAL	V、L 、常量
OUT	输出	DINT	M、V、L、SM

该指令将输入的实数 *IN*转换为双整数（小数部分四舍五入）并赋给 *OUT*。

- LD

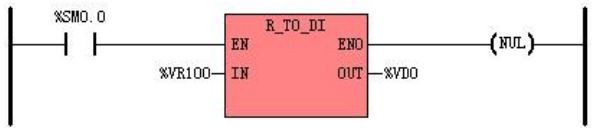
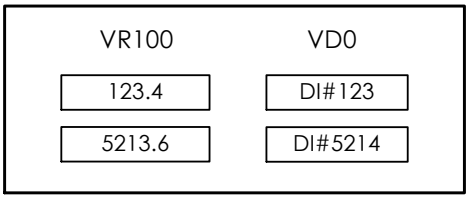
  如果 *EN*为 1，则该指令被执行。

- IL

  如果 CR 值为 1，则该指令被执行。

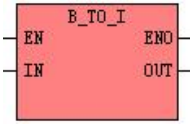
  该指令的执行不影响 CR 值。

➢ 指令使用举例

<table>
<tr><td>LD</td><td> 
</td><td>SM0.0 恒为 1，因此 R_TO_DI 指令总是执行：将实数 VR100 转换为双整数并赋给 VD0。</td></tr>
<tr><td>IL</td><td colspan="2">LD %SM0.0 (* 建立 CR，其值为 1 *)
R_TO_DI %VR100, %VD0 (* 将实数 VR100 转换为双整数并赋给 VD0 *)</td></tr>
<tr><td>结果</td><td colspan="2">结果举例如下：
 
</td></tr>
</table>

6.7.3 B_TO_I（字节型转整型）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	B_TO_I			
IL	B_TO_I	B_TO_I *IN, OUT*	U	

参数	输入/输出	数据类型	允许的内存区
IN	输入	BYTE	I、Q、M、V、L、SM、常量
OUT	输出	INT	Q、M、V、L、SM、AQ

该指令将输入的字节型数据 *IN* 转换为整数并赋给 *OUT*。

- LD

  如果 *EN* 为 1，则该指令被执行。

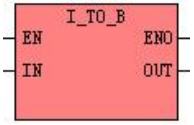
- IL

  如果 CR 值为 1，则该指令被执行。

  该指令的执行不影响 CR 值。

6.7.4 I_TO_B（整型转字节型）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	I_TO_B			
IL	I_TO_B	I_TO_B *IN*, *OUT*	U	

参数	输入/输出	数据类型	允许的内存区
IN	输入	INT	I、Q、M、V、L、SM、AI、AQ、T、C、常量
OUT	输出	BYTE	Q、M、V、L、SM

该指令将整数 *IN*转换为字节型数值并赋给 *OUT*。

字节型的取值范围为[0,255]，若参数 *IN*的值超过了这个范围，则转换结果为其低字节值，同时 PLC 会记录下溢出错误。

- LD

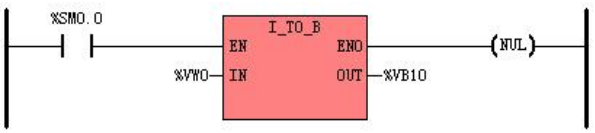
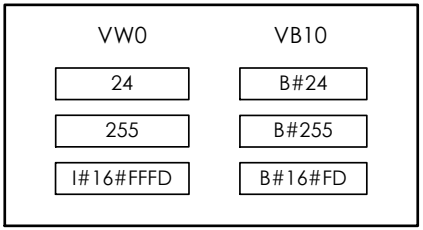
  如果 *EN*为 1，则该指令被执行。

- IL

  如果 CR 值为 1，则该指令被执行。

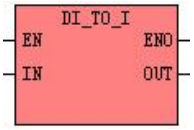
  该指令的执行不影响 CR 值。

➢ 指令使用举例

<table>
<tr><td>LD</td><td> 
</td><td>SM0.0 恒为 1，因此 I_TO_B 指令总是执行：将 VW0 的低字节赋给 VB10。</td></tr>
<tr><td>IL</td><td colspan="2">LD %SM0.0 (* 建立 CR，其值为 1 *)
I_TO_B %VW0, %VB10 (* 将 VW0 整数值转换为字节型值并赋给 VB10 *)</td></tr>
<tr><td>结果</td><td colspan="2">结果举例如下：
 
</td></tr>
</table>

6.7.5 DI_TO_I（双整型转整型）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	DI_TO_I			
IL	DI_TO_I	DI_TO_I *IN, OUT*	U	

参数	输入/输出	数据类型	允许的内存区
IN	输入	DINT	I、Q、M、V、L、SM、HC、常量
OUT	输出	INT	Q、M、V、L、SM、AQ

该指令将双整数 *IN*转换为整数并赋给 *OUT*。

整数的取值范围为 [-32768,32767]，若参数 *IN*的值超过了这个范围，则转换结果为其低字值，同时 PLC 会记录下溢出错误。

- LD

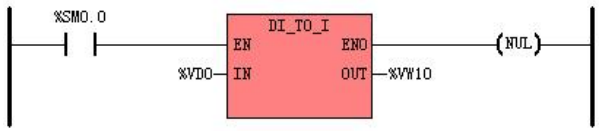
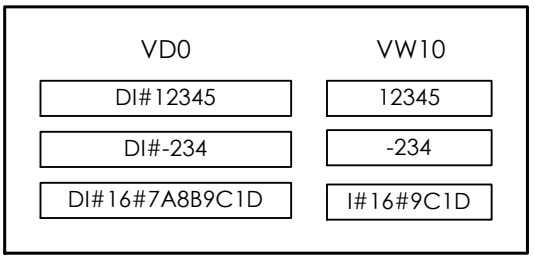
  如果 *EN*为 1，则该指令被执行。

- IL

  如果 CR 值为 1，则该指令被执行。

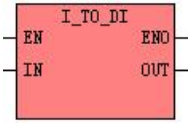
  该指令的执行不影响 CR 值。

➢ 指令使用举例

<table>
<tr><td>LD</td><td> 
</td><td>SM0.0 恒为 1，因此 DI_TO_I 指令总是执行：将 VD0 的低字赋给 VW10。</td></tr>
<tr><td>IL</td><td colspan="2">LD %SM0.0 (* 建立 CR，其值为 1 *)
DI_TO_I %VD0, %VW10 (* 将 VD0 双整数值转换为整数并赋给 VW10 *)</td></tr>
<tr><td>结果</td><td colspan="2">结果举例如下：
 
</td></tr>
</table>

6.7.6 I_TO_DI（整型转双整型）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	I_TO_DI			
IL	I_TO_DI	I_TO_DI *IN, OUT*	U	

参数	输入/输出	数据类型	允许的内存区
IN	输入	INT	I、Q、M、V、L、SM、AI、AQ、T、C、常量
OUT	输出	DINT	Q、M、V、L、SM

该指令将输入的整数 *IN*转换为双整数并赋给 *OUT*。

- LD

  如果 *EN*为 1，则该指令被执行。

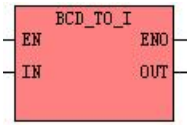
- IL

  如果 CR 值为 1，则该指令被执行。

  该指令的执行不影响 CR 值。

6.7.7 BCD_TO_I（BCD 码转整型）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	BCD_TO_I			
IL	BCD _TO_I	BCD _TO_I *IN, OUT*	U	

参数	输入/输出	数据类型	允许的内存区
IN	输入	WORD	I、Q、M、V、L、SM、常量
OUT	输出	INT	Q、M、V、L、SM、AQ

该指令将输入的 BCD 码 *IN*转换为整数并赋给 *OUT*。

注意：BCD 码采用 8421 码。*IN*的有效范围是 0∽9999 BCD。

- LD

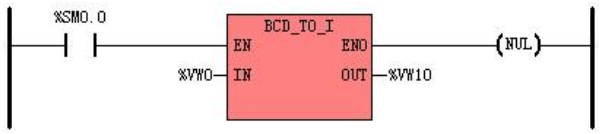
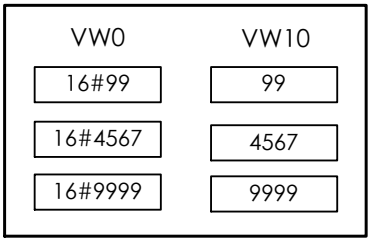
  如果 *EN*为 1，则该指令被执行。

- IL

  如果 CR 值为 1，则该指令被执行。

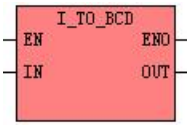
  该指令的执行不影响 CR 值。

➢ 指令使用举例

<table>
<tr><td>LD</td><td> 
</td><td>SM0.0 恒为 1，因此 BCD_TO_I 指令总是执行：将 VW0 的值由 BCD 转换为整数并赋给 VW10。</td></tr>
<tr><td>IL</td><td colspan="2">LD %SM0.0 (* 建立 CR，其值为 1 *)
BCD_TO_I %VW0, %VW10 (* 将 VW0 的值由 BCD 转换为整数并赋给 VW10 *)</td></tr>
<tr><td>结果</td><td colspan="2">结果举例如下：
 
</td></tr>
</table>

6.7.8 I_TO_BCD（整型转 BCD 码）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	I_TO_BCD			
IL	I_TO_BCD	I _TO_BCD*IN, OUT*	U	

参数	输入/输出	数据类型	允许的内存区
IN	输入	INT	I、Q、M、V、L、SM、AI、AQ、T、C、常量
OUT	输出	WORD	Q、M、V、L、SM

该指令将输入的整数 *IN*转换为 BCD 码并赋给 *OUT*。

注意：BCD 码采用 8421 码,因此 *IN*的有效范围是 0∽9999，若超过这个范围，则 PLC 会记录下溢出错误，同时按如下方式处理：若 *IN* ≤ 0，则转换结果保持为 0，若 *IN*≥ 9999，则转换结果保持为 9999 BCD。

- LD

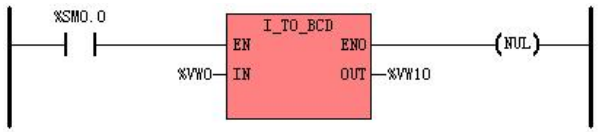
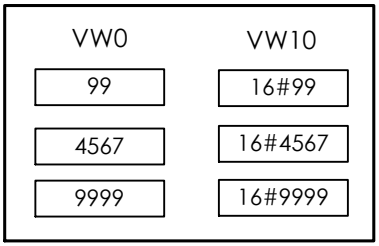
  如果 *EN*为 1，则该指令被执行。

- IL

  如果 CR 值为 1，则该指令被执行。

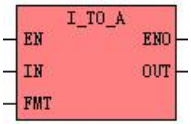
  该指令的执行不影响 CR 值。

➢ 指令使用举例

<table>
<tr><td>LD</td><td> 
</td><td>SM0.0 恒为 1，因此 I_TO_BCD 指令总是执行：将 VW0 的值由整数转换为 BCD 并赋给 VW10。</td></tr>
<tr><td>IL</td><td colspan="2">LD %SM0.0 (* 建立 CR，其值为 1 *)
I_TO_BCD %VW0, %VW10 (* 将 VW0 的值由整数转换为 BCD 并赋给 VW10 *)</td></tr>
<tr><td>结果</td><td colspan="2">结果举例如下：
 
</td></tr>
</table>

6.7.9 I_TO_A（整型转 ASCII）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	I_TO_A			
IL	I_TO_A	I_TO_AIN, OUT, FMT	U	

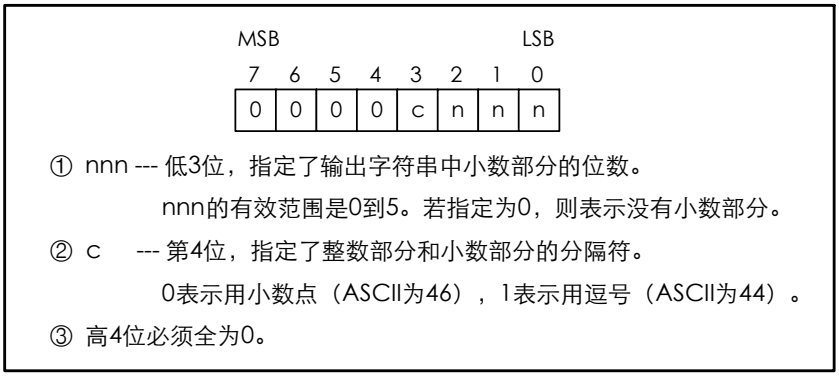
参数	输入/输出	数据类型	允许的内存区
IN	输入	INT	I、Q、M、V、L、SM、AI、AQ、T、C、常量
FMT	输入	BYTE	I、Q、M、V、L、SM
OUT	输出	BYTE	Q、M、V、L、SM

注意： ***OUT*** **参数为一个可变长度内存块的起始地址，用户需保证该内存块全部都是合法的地址范围，否则结果不可预期。**

该指令将输入的整数 *IN*转换成一个 ASCII 字符串，并将转换结果格式化输出到输出缓冲区中。正数转换后不带符号，负数转换后带负号。

参数 *OUT* 定义了输出缓冲区的起始地址，该区域占用连续 8 个字节的内存空间。字符串在缓冲区中右对齐，缓冲区中未使用的地址被赋值为空格符（ASCII 值为 32）。

参数 *FMT* 定义了输出字符串的表示形式，它的组成如下图：



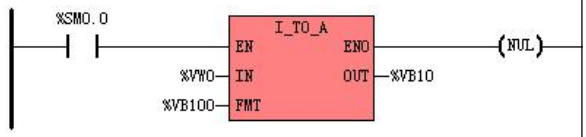
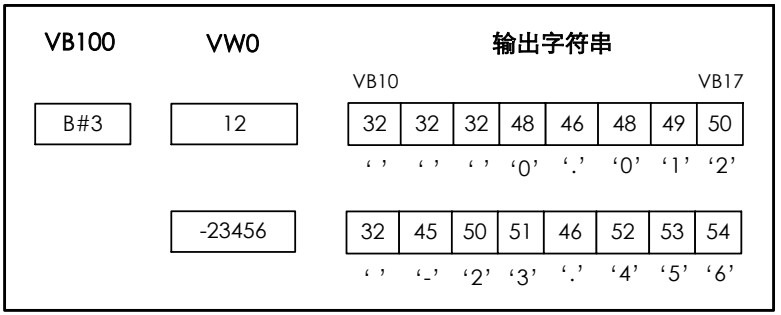
- LD

  如果 *EN* 为 1，则该指令被执行。

- IL

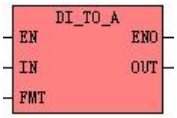
  如果 CR 值为 1，则该指令被执行。该指令的执行不影响 CR 值。

➢ 指令使用举例

<table>
<tr><td>LD</td><td> 
</td><td>SM0.0 恒为 1，因此 I_TO_A 指令总是执行：将 VW0 的值转换为字符串并格式化输出到 VB10 开始的连续 8 个字节中。</td></tr>
<tr><td>IL</td><td colspan="2">LD %SM0.0
I_TO_A %VW0, %VB10, %VB100</td></tr>
<tr><td>结果</td><td colspan="2">结果举例如下：
 
</td></tr>
</table>

6.7.10 DI_TO_A（双整型转 ASCII）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	DI_TO_A			
IL	DI_TO_A	DI_TO_AIN, OUT, FMT	U	

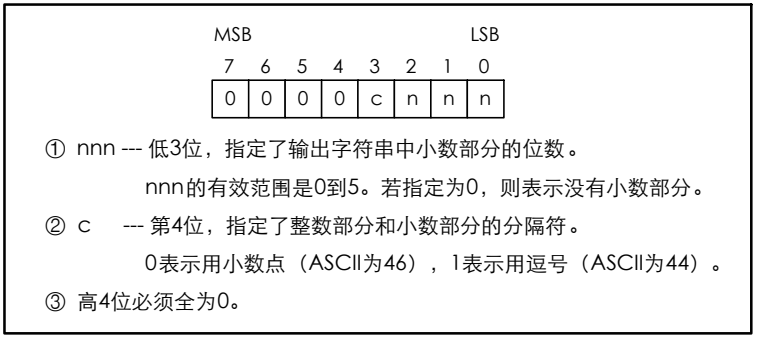
参数	输入/输出	数据类型	允许的内存区
IN	输入	DINT	I、Q、M、V、L、SM、HC、常量
FMT	输入	BYTE	I、Q、M、V、L、SM
OUT	输出	BYTE	Q、M、V、L、SM

注意： ***OUT*** **参数为一个可变长度内存块的起始地址，用户需保证该内存块全部都是合法的地址范围，否则结果不可预期。**

该指令将输入的双整数 *IN* 转换成一个 ASCII 字符串并将转换结果格式化输出到输出缓冲区中。正数转换后不带符号，负数转换后带负号。

参数 *OUT* 定义了输出缓冲区的起始地址，该区域占用连续 12 个字节的内存空间。字符串在缓冲区中右对齐，缓冲区中未使用的地址被赋值为空格符（ASCII 为 32）。

参数 *FMT* 定义了输出字符串的表示形式，它的组成如下图：



- LD

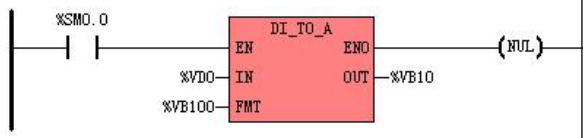
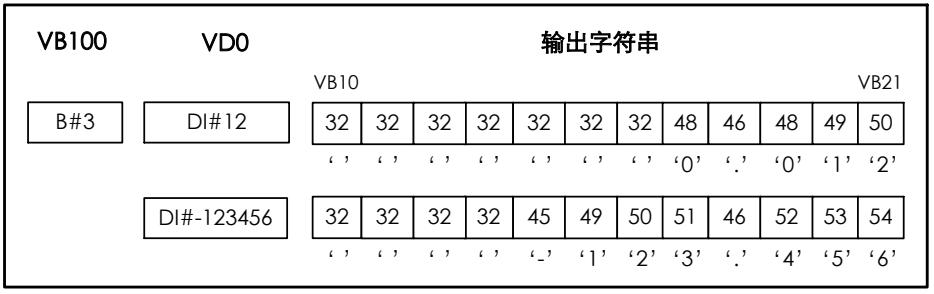
  如果 *EN* 为 1，则该指令被执行。

- IL

  如果 CR 值为 1，则该指令被执行。

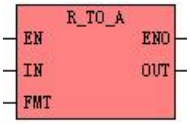
  该指令的执行不影响 CR 值。

➢ 指令使用举例

<table>
<tr><td>LD</td><td> 
</td><td>SM0.0 恒为 1，因此 DI_TO_A 指令总是执行：将 VD0 的值转换为字符串并格式化输出到 VB10 开始的连续 12 个字节中。</td></tr>
<tr><td>IL</td><td colspan="2">LD %SM0.0
DI_TO_A %VD0, %VB10, %VB100</td></tr>
<tr><td>结果</td><td colspan="2">结果举例如下：
 
</td></tr>
</table>

6.7.11 R_TO_A（实型转 ASCII）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	R_TO_A			
IL	R_TO_A	R_TO_AIN, OUT, FMT	U	

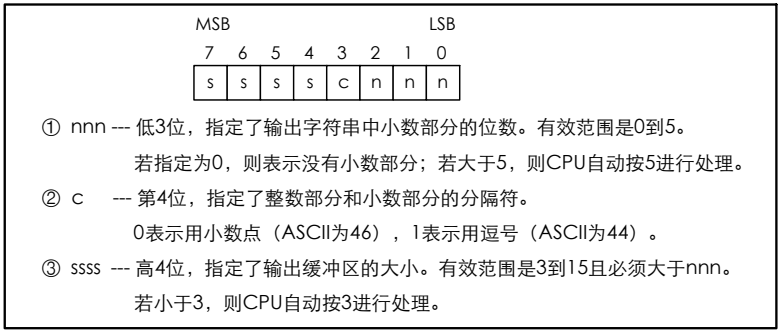
参数	输入/输出	数据类型	允许的内存区
IN	输入	REAL	V、L、常量
FMT	输入	BYTE	I、Q、M、V、L、SM
OUT	输出	BYTE	Q、M、V、L、SM

注意： ***OUT*** **参数为一个可变长度内存块的起始地址，用户需保证该内存块全部都是合法的地址范围，否则结果不可预期。**

该指令将输入的实数 *IN* 转换成一个 ASCII 字符串并将转换结果格式化输出到输出缓冲区中。正数转换后不带符号，负数转换后带负号。若 *IN* 的小数部分的位数大于参数 *FMT* 中 nnn 指定的小数位数，则转换之前首先将 *IN* 四舍五入，若小于则将 *IN* 的小数位数不足的部分补 0。

参数 *OUT* 定义了输出缓冲区的起始地址，该区域的大小在 *FMT* 中指定。字符串在缓冲区中右对齐，缓冲区中未使用的地址被赋值为空格符（ASCII 为 32）。

参数 *FMT* 定义了输出字符串的表示形式，它的组成如下图：



该指令使用时需要注意：输入 *IN*的允许范围是[-2147480000.0, 4294960000.0],若 *IN*超出这个范围或者转换结果的长度超出了输出缓冲区的长度，则输出缓冲区全部用空格符（ASCII 值为 32）填充。

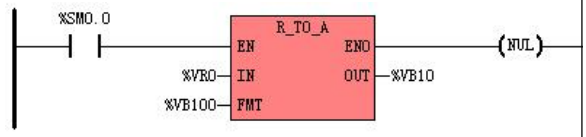
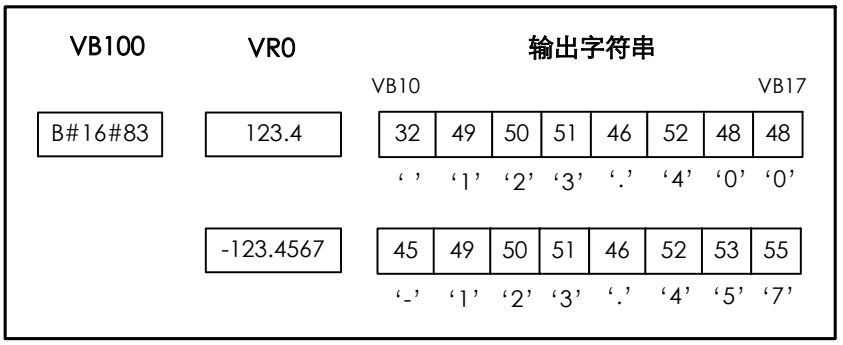
- LD

  如果 *EN*为 1，则该指令被执行。

- IL

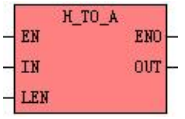
  如果 CR 值为 1，则该指令被执行。该指令的执行不影响 CR 值。

➢ 指令使用举例

<table>
<tr><td>LD</td><td> 
</td><td>SM0.0 恒为 1，因此 R_TO_A 指令总是执行：将 VR0 的值转换为字符串并格式化输出到 VB10 开始的缓冲区中。</td></tr>
<tr><td>IL</td><td colspan="2">LD %SM0.0
R_TO_A %VR0, %VB10, %VB100</td></tr>
<tr><td>结果</td><td colspan="2">结果举例如下：

</td></tr>
</table>

6.7.12 H_TO_A（16 进制数转 ASCII）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	H_TO_A			
IL	H_TO_A	H_TO_AIN, OUT, LEN	U	

参数	输入/输出	数据类型	允许的内存区
IN	输入	BYTE	I、Q、M、V、L、SM
LEN	输入	BYTE	I、Q、M、V、L、SM、常量
OUT	输出	BYTE	Q、M、V、L、SM

注意： *IN*、*OUT* 参数均表示为一个可变长度内存块的起始地址，用户需保证该内存块全部都是合法的地址范围，否则结果不可预期。

该指令将地址 *IN* 开始的连续 *LEN* 个字节的十六进制数转换成一个 ASCII 字符串并输出到地址 *OUT* 开始的输出缓冲区中。注意：由于每 4 个二进制位表示一个十六进制数，因此每个输入字节包含了 2 个十六进制数，输出缓冲区共占用了 *LEN*×*2* 个字节的空间。

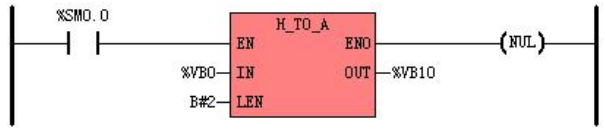
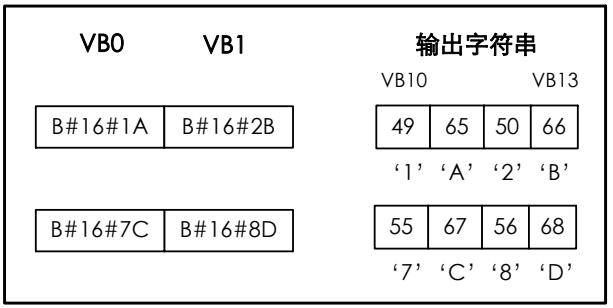
- LD

  如果 *EN* 为 1，则该指令被执行。

- IL

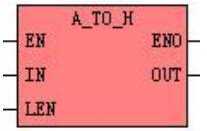
如果 CR 值为 1，则该指令被执行。该指令的执行不影响 CR 值。

- 指令使用举例

<table>
<tr><td>LD</td><td> 
</td><td>SM0.0 恒为 1，因此 H_TO_A 指令总是执行：将 VB0 开始连续 2 个字节的十六进制数转换为字符串并输出到 VB10 开始的连续 4 个字节中。</td></tr>
<tr><td>IL</td><td colspan="2">LD %SM0.0
H_TO_A %VB0, %VB10, B#2</td></tr>
<tr><td>结果</td><td colspan="2">结果举例如下：
 
</td></tr>
</table>

6.7.13 A_TO_H（ASCII 转 16 进制数）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	A_TO_H			
IL	A_TO_H	A_TO_HIN, OUT, LEN	U	

参数	输入/输出	数据类型	允许的内存区
IN	输入	BYTE	I、Q、M、V、L、SM
LEN	输入	BYTE	I、Q、M、V、L、SM、常量
OUT	输出	BYTE	Q、M、V、L、SM

注意： *IN*、*OUT* 参数均表示为一个可变长度内存块的起始地址，用户需保证该内存块全部都是合法的地址范围，否则结果不可预期。

该指令将地址 *IN* 开始的连续 *LEN* 个 ASCII 字符转换成十六进制数并输出到地址 *OUT* 开始的输出缓冲区中。注意：由于一个十六进制数采用 4 个二进制位来表示，因此每个输入字节（表示一个 ASCII 字符）转换后在输出缓冲区中占用 4 个二进制位（半个字节）的空间。

有效的 ASCII 输入范围是：B#16#30∽B#16#39（表示字符 0∽9），B#16#41∽B#16#46（表示字符 A∽F），B#16#61∽B#16#66（表示字符 a∽f）。

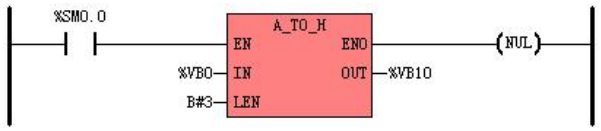
- LD

  如果 *EN* 为 1，则该指令被执行。

- IL

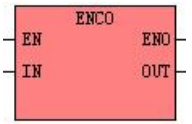
  如果 CR 值为 1，则该指令被执行。该指令的执行不影响 CR 值。

➢ 指令使用举例

<table>
<tr><td>LD</td><td> 
</td><td>SM0.0 恒为 1，因此 A_TO_H 指令总是执行：将 VB0 开始的连续 3 个字节的 ASCII 字符转换成十六进制数并输出到 VB100 开始的输出缓冲区中。</td></tr>
<tr><td>IL</td><td colspan="2">LD %SM0.0
A_TO_H%VB0, %VB10, B#3</td></tr>
<tr><td>结果</td><td colspan="2">结果举例如下：
VB0 VB1 VB2 VB10 VB11
51 56 54 B#16#38 B#16#6x
‘3’ ‘8’ ‘6’
55 65 49 B#16#7A B#16#1x
‘7’ ‘A’ ‘1’
注：上面的x表示这半个字节（4位）中内容保持原有值不变。</td></tr>
</table>

6.7.14 ENCO（编码）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	ENCO			
IL	ENCO	ENCOIN, OUT	U	

参数	输入/输出	数据类型	允许的内存区
IN	输入	WORD	I、Q、M、V、L、SM、常量
OUT	输出	BYTE	Q、M、V、L、SM

该指令从输入字 *IN*的最低位开始计算，将第一个为 1 的位的位号写入输出字节 *OUT*。注意：若 *IN*的值等于 0，那么计算结果是无意义的。

- LD

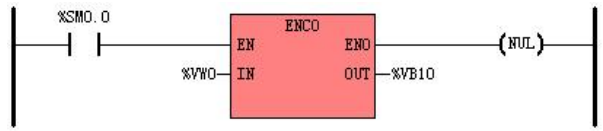
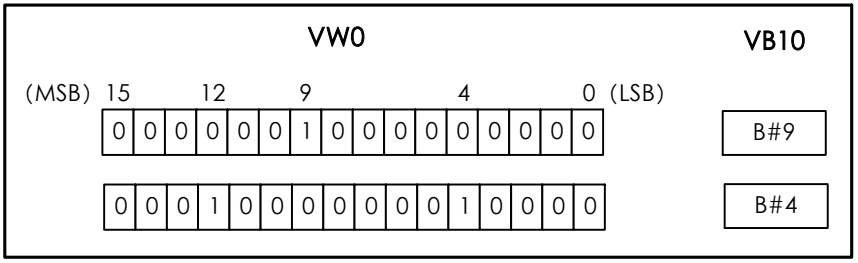
  如果 *EN*为 1，则该指令被执行。

- IL

  如果 CR 值为 1，则该指令被执行。

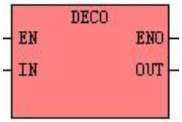
  该指令的执行不影响 CR 值。

➢ 指令使用举例

<table>
<tr><td>LD</td><td> 
</td><td>SM0.0 恒为 1，因此 ENCO 指令总是执行：从 VW0 中的最低位开始计算，将第一个等于 1 的位的位号写入 VB10 中。</td></tr>
<tr><td>IL</td><td colspan="2">LD %SM0.0
ENCO %VW0, %VB10</td></tr>
<tr><td>结果</td><td colspan="2">结果举例如下图，图中 VW0 的值采用了二进制来表示。
 
</td></tr>
</table>

6.7.15 DECO（解码）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	DECO			
IL	DECO	DECOIN, OUT	U	

参数	输入/输出	数据类型	允许的内存区
IN	输入	BYTE	I、Q、M、V、L、SM、常量
OUT	输出	WORD	Q、M、V、L、SM

该指令用输入字节 *IN* 的低四位所表示的数值来指定一个位号，然后根据这个位号将输出字 *OUT* 中对应的位赋值为 1，其它位全部赋值为 0。

- LD

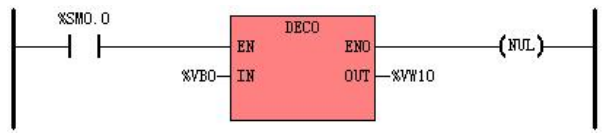
  如果 *EN* 为 1，则该指令被执行。

- IL

  如果 CR 值为 1，则该指令被执行。

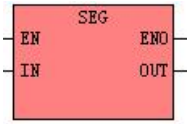
  该指令的执行不影响 CR 值。

➢ 指令使用举例

<table>
<tr><td>LD</td><td> 
</td><td>SM0.0 恒为 1，因此 DECO 指令总是执行：VB0 的低四位所表示的数值代表了位号，根据这个位号将 VW0 中的相应位赋值为 1，其它为全部赋值为 0。</td></tr>
<tr><td>IL</td><td colspan="2">LD %SM0.0
DECO %VB0, %VW10</td></tr>
<tr><td>结果</td><td colspan="2">结果举例如下图，图中 VW10 的值采用了二进制来表示。
 
</td></tr>
</table>

6.7.16 SEG（七段码显示）

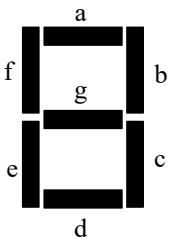
➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	SEG			
IL	SEG	SEGIN, OUT	U	

参数	输入/输出	数据类型	允许的内存区
IN	输入	BYTE	I、Q、M、V、L、SM、常量
OUT	输出	BYTE	Q、M、V、L、SM

输入字节 *IN*的低四位表示了要显示的数值，SEG 指令根据该数值来产生七段码的段码值并输出至 *OUT*中。

IN (低四位)	显示	*OUT* (- g f e d c b a)
0	0	0 0 1 1 1 1 1 1
1	1	0 0 0 0 0 1 1 0
2	2	0 1 0 1 1 0 1 1
3	3	0 1 0 0 1 1 1 1
4	4	0 1 1 0 0 1 1 0
5	5	0 1 1 0 1 1 0 1
6	6	0 1 1 1 1 1 0 1
7	7	0 0 0 0 0 1 1 1



IN (低四位)	显示	*OUT* (- g f e d c b a)
8	8	0 1 1 1 1 1 1 1
9	9	0 1 1 0 0 1 1 1
A	A	0 1 1 1 0 1 1 1
B	B	0 1 1 1 1 1 0 0
C	C	0 0 1 1 1 0 0 1
D	D	0 1 0 1 1 1 1 0
E	E	0 1 1 1 1 0 0 1
F	F	0 1 1 1 0 0 0 1

- LD

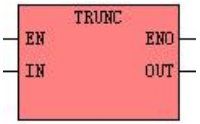
  如果 *EN*为 1，则该指令被执行。

- IL

  如果 CR 值为 1，则该指令被执行。该指令的执行不影响 CR 值。

6.7.17 TRUNC（取整）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	TRUNC			
IL	TRUNC	TRUNC *IN*, *OUT*	U	

参数	输入/输出	数据类型	允许的内存区
IN	输入	REAL	V、L 、常量
OUT	输出	DINT	M、V、L、SM

该指令将输入的实数 *IN* 转换为双整数（小数部分被舍弃）并赋给 *OUT*。

- LD

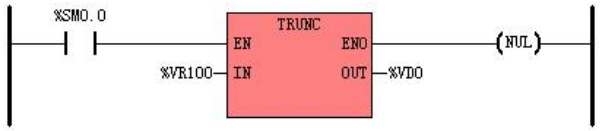
  如果 *EN* 为 1，则该指令被执行。

- IL

  如果 CR 值为 1，则该指令被执行。

  该指令的执行不影响 CR 值。

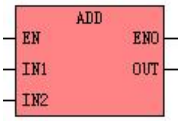
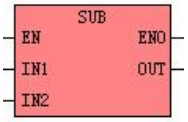
➢ 指令使用举例

<table>
<tr><td>LD</td><td> 
</td><td>SM0.0 恒为 1，因此 TRUNC 指令总是执行：将实数 VR100 转换为双整数（小数部分被舍弃）并赋给 VD0。</td></tr>
<tr><td>IL</td><td colspan="2">LD %SM0.0 (* 建立 CR，其值为 1 *)
TRUNC %VR100, %VD0 (* 将实数 VR100 舍弃小数部分后转换为双整数并赋给 VD0 *)</td></tr>
<tr><td>结果</td><td colspan="2">结果举例如下：
VR100 VD0
123.4 DI#123
5213.6 DI#5213</td></tr>
</table>

6.8 数学运算

6.8.1 ADD（加法）、SUB（减法）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	ADD			☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
	SUB			
IL	ADD	ADD *IN1, OUT*	U	
	SUB	SUB *IN1, OUT*		

参数	输入/输出	数据类型	允许的内存区
IN1	输入	INT、DINT、REAL、BYTE、WORD、DWROD	I、Q、AI、AQ、M、V、L、SM、T、C、HC、常量、指针
IN2	输入	INT、DINT、REAL、BYTE、WORD、DWROD	I、Q、AI、AQ、M、V、L、SM、T、C、HC、常量、指针
OUT	输出	INT、DINT、REAL、BYTE、WORD、DWROD	Q、AQ、M、V、L、SM、指针

参数 *IN1*、*IN2*、*OUT* 的数据类型必须一致。

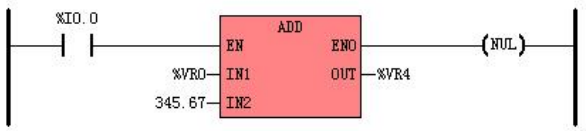
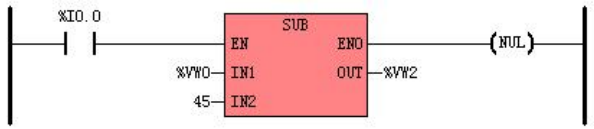
- LD

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

- IL

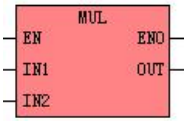
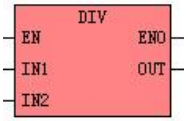
若 CR 值为 1，则指令被执行。其中，*ADD* 指令的功能是：*OUT*= *OUT* + *IN1*；*SUB* 指令的功能是：*OUT*= *OUT*－*IN1*。ADD、SUB 指令的执行不影响 CR 值。

➢ 指令使用举例

<table>
<tr><td rowspan="2">LD</td><td> 
</td><td>若 I0.0 为 0：
不执行 ADD 指令。
若 I0.0 为 1：
VR4 = VR4 + 345.67。</td></tr>
<tr><td> 
</td><td>若 I0.0 为 0：
不执行 SUB 指令。
若 I0.0 为 1：
VW2 = VW2 - 45。</td></tr>
<tr><td rowspan="2">IL</td><td colspan="2">LD %I0.0 (* 建立 CR，其值为 I0.0 的值 *)
ADD 345.67, %VR4(* 若 CR 为 1：VR4 = VR4 + 345.67 *)
(* 若 CR 为 0：不执行 ADD 指令 *)</td></tr>
<tr><td colspan="2">LD %I0.0 (* 建立 CR，其值为 I0.0 的值 *)
SUB 45, %VW2(* 若 CR 为 1：VW2 = VW2－45 *)
(* 若 CR 为 0：不执行 SUB 指令 *)</td></tr>
</table>

6.8.2 MUL（乘法）、DIV（除法）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	MUL			☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
	DIV			
IL	MUL	MUL *IN1, OUT*	U	
	DIV	DIV *IN1, OUT*		

参数	输入/输出	数据类型	允许的内存区
IN1	输入	INT、DINT、REAL、BYTE、WORD、DWROD	I、Q、AI、AQ、M、V、L、SM、T、C、HC、常量、指针
IN2	输入	INT、DINT、REAL、BYTE、WORD、DWROD	I、Q、AI、AQ、M、V、L、SM、T、C、HC、常量、指针
OUT	输出	INT、DINT、REAL、BYTE、WORD、DWROD	Q、AQ、M、V、L、SM、指针

参数 *IN1*、*IN2*、*OUT* 的数据类型必须一致。

若 *DIV* 的除数为 0，则指令的输出值维持上一次的结果，同时 PLC 记录下“被 0 除”错误。

- LD

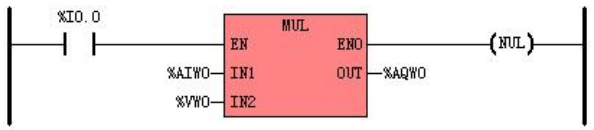
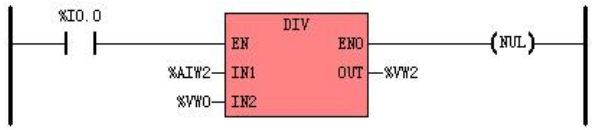
若 *EN* 值为 1，则指令被执行。其中，*MUL* 指令的功能是：*OUT*＝*IN1*×*IN2*；*DIV* 指令的功能是：*OUT*＝*IN1*÷*IN2*。

- IL

如果 CR 值为 1，则指令被执行。其中，*MUL* 指令的功能是：*OUT*＝*OUT*× *IN1*；*DIV* 指令的功能是：*OUT*＝*OUT*÷ *IN1*。

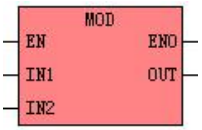
MUL、DIV 指令的执行不影响 CR 值。

➢ 指令使用举例

<table>
<tr><td rowspan="2">LD</td><td> 
</td><td>若 I0.0 为 0:不执行 MUL 指令。
若 I0.0 为 1：将 AIW0 与 VW0 相乘，并将结果赋给 AQW0。</td></tr>
<tr><td> 
</td><td>若 I0.0 为 0:不执行 DIV 指令。
若 I0.0 为 1:将 AIW2 除以 VW0，并将结果赋给 VW2。</td></tr>
<tr><td rowspan="2">IL</td><td colspan="2">LD %I0.0 (* 建立 CR，其值为 I0.0 的值 *)
MUL %AIW0, %VW0 (* 若 CR 为 1：VW0 = VW0×AIW0 *)
(* 若 CR 为 0：不执行 MUL 指令 *)</td></tr>
<tr><td colspan="2">LD %I0.0 (* 建立 CR，其值为 I0.0 的值 *)
DIV %AIW2, %VW0 (* 若 CR 为 1：VW0 = VW0÷AIW2 *)
(* 若 CR 为 0：不执行 DIV 指令 *)</td></tr>
</table>

6.8.3 MOD（求余数）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	MOD			
IL	MOD	MOD *IN1*, *OUT*	U	

参数	输入/输出	数据类型	允许的内存区
IN1	输入	INT、DINT、 BYTE、WORD、DWROD	I、Q、AI、AQ、M、V、L、SM、T、C、HC、 常量、指针
IN2	输入	INT、DINT、 BYTE、WORD、DWROD	I、Q、AI、AQ、M、V、L、SM、T、C、HC、 常量、指针
OUT	输出	INT、DINT、 BYTE、WORD、DWROD	Q、AQ、M、V、L、SM、指针

参数 *IN1*、*IN2*、*OUT* 的数据类型必须一致。

若除数 *(IN2)* 为 0，则指令的输出值维持上一次的结果，同时 PLC 记录下“被 0 除”错误。

- LD

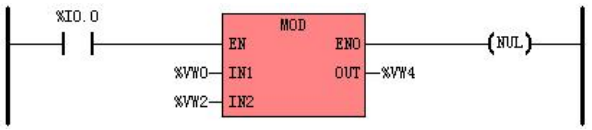
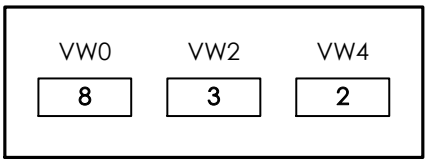
  若 *EN* 值为 1，则该指令被执行：将 *IN1* 除以 *IN2*，并将余数赋给 *OUT*。

- IL

  如果 CR 值为 1，则该指令被执行：将 *OUT* 除以 *IN1*，并将余数赋给 *OUT*。

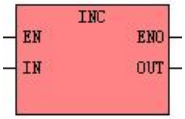
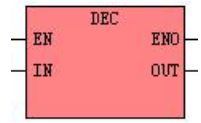
  MOD 指令的执行不影响 CR 值。

➢ 指令使用举例

<table>
<tr><td>LD</td><td> 
</td><td>若 I0.0 为 0：不执行 MOD 指令。
若 I0.0 为 1：将 VW0 除以 VW2，得到的余数赋给 VW4。</td></tr>
<tr><td>IL</td><td colspan="2">LD %I0.0 (* 建立 CR，其值为 I0.0 的值 *)
MOD %VW0, %VW4 (* 若 CR 为 1：将 VW4 除以 VW0，得到的余数赋给 VW4 *)
(* 若 CR 为 0：不执行 MOD 指令 *)</td></tr>
<tr><td>结果</td><td colspan="2">若上述 LD 例子中 MOD 指令得以执行，则结果举例如下：
 
</td></tr>
</table>

6.8.4 INC（加 1）、DEC（减 1）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	INC			☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
	DEC			
IL	INC	INC *OUT*	U	
	DEC	DEC *OUT*		

参数	输入/输出	数据类型	允许的内存区
IN	输入	INT、DINT、 BYTE、WORD、DWROD	I、Q、AI、AQ、M、V、L、SM、T、 C、HC、常量、指针
OUT	输出	INT、DINT、 BYTE、WORD、DWROD	Q、AQ、M、V、L、SM、指针

参数 *IN*、*OUT* 的数据类型必须一致。

- LD

若 *EN* 值为 1，则指令被执行。其中，*INC* 指令的功能是：*OUT*＝*IN*＋1；*DEC* 指令的功能是：*OUT*＝*IN*－1。

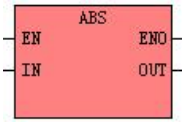
- IL

如果 CR 值为 1，则指令被执行。其中，*INC* 指令的功能是：*OUT*＝*OUT*＋1；*DEC* 指令的功能是：*OUT*＝*OUT*－1。

INC、DEC 指令的执行不影响 CR 值。

6.8.5 ABS（绝对值）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	ABS			
IL	ABS	ABS *IN, OUT*	U	

参数	输入/输出	数据类型	允许的内存区
IN	输入	INT、DINT、REAL	I、Q、V、M、L、SM、T、C、AI、AQ、HC、常量、指针
OUT	输出	INT、DINT、REAL	Q、V、M、L、SM、AQ、指针

参数 IN、OUT 的数据类型必须一致。

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

- LD

  若 *EN* 值为 1，则该指令被执行。

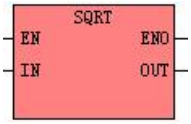
- IL

  如果 CR 值为 1，则该指令被执行。

  该指令的执行不影响 CR 值。

6.8.6 SQRT（平方根）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	SQRT			
IL	SQRT	SQRT *IN*, *OUT*	U	

参数	输入/输出	数据类型	允许的内存区
IN	输入	REAL	V、L、常量、指针
OUT	输出	REAL	V、L、指针

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

注意：若输入参数 *IN*为负数，则指令的输出值维持上一次的结果，同时 PLC 记录下错误。

- LD

  若 *EN*值为 1，则该指令被执行。

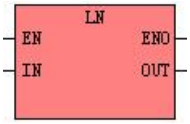
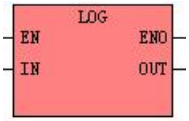
- IL

  如果 CR 值为 1，则该指令被执行。

  该指令的执行不影响 CR 值。

6.8.7 LN（自然对数）、LOG（常用对数）

➢ 指令及其操作数说明

<table>
<tr><th></th><th>名称</th><th>指令格式</th><th>影响 CR 值</th><th rowspan="2"></th></tr>
<tr><td rowspan="2">LD</td><td>LN</td><td> 
</td><td rowspan="2"></td></tr>
<tr><td>LOG</td><td> 
</td><td rowspan="3">☑ K5
☑ K2
☑ KS
☑ KW
☑ MK
☑ K6</td></tr>
<tr><td rowspan="2">IL</td><td>LN</td><td>LN IN, OUT</td><td rowspan="2">U</td></tr>
<tr><td>LOG</td><td>LOG IN, OUT</td></tr>
</table>

参数	输入/输出	数据类型	允许的内存区
IN	输入	REAL	V、L、常量、指针
OUT	输出	REAL	V、L、指针

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

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

注意：若输入参数 *IN* 为 0 或负数，则指令的输出值维持上一次的结果，同时 PLC 记录下错误。

- LD

  若 *EN* 值为 1，则指令被执行。

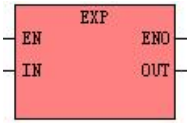
- IL

  如果 CR 值为 1，则指令被执行。

  LN、LOG 指令的执行不影响 CR 值。

6.8.8 EXP（以 e 为底的指数）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	EXP			
IL	EXP	EXP *IN*, *OUT*	U	

参数	输入/输出	数据类型	允许的内存区
IN	输入	REAL	V、L、常量、指针
OUT	输出	REAL	V、L、指针

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

- LD

  若 *EN*值为 1，则该指令被执行。

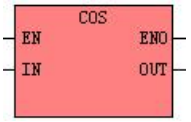
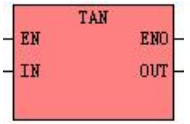
- IL

  如果 CR 值为 1，则该指令被执行。

  该指令的执行不影响 CR 值。

6.8.9 SIN（正弦）、COS（余弦）、TAN（正切）

➢ 指令及其操作数说明

<table>
<tr><th></th><th>名称</th><th>指令格式</th><th>影响 CR 值</th><th rowspan="2"></th></tr>
<tr><td rowspan="3">LD</td><td>SIN</td><td> 
</td><td rowspan="3"></td></tr>
<tr><td>COS</td><td> 
</td><td rowspan="3">☑ K5
☑ K2
☑ KS
☑ KW
☑ MK
☑ K6</td></tr>
<tr><td>TAN</td><td> 
</td></tr>
<tr><td rowspan="3">IL</td><td>SIN</td><td>SIN IN, OUT</td><td rowspan="3">U</td><td rowspan="3"></td></tr>
<tr><td>COS</td><td>COS IN, OUT</td></tr>
<tr><td>TAN</td><td>TAN IN, OUT</td></tr>
</table>

参数	输入/输出	数据类型	允许的内存区
IN	输入	REAL	V、L、常量、指针
OUT	输出	REAL	V、L、指针

输入 *IN*表示的是弧度值。

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

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

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

- LD

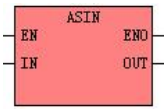
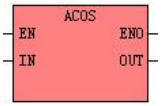
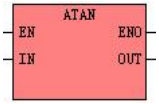
  若 *EN* 值为 1，则指令被执行。

- IL

  如果 CR 值为 1，则指令被执行。指令的执行不影响 CR 值。

6.8.10 ASIN（反正弦）、ACOS（反余弦）、ATAN（反正切）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	ASIN			☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
	ACOS			
	ATAN			
IL	ASIN	ASIN *IN, OUT*	U	
	ACOS	ACOS *IN, OUT*		
	ATAN	ATAN *IN, OUT*		

参数	输入/输出	数据类型	允许的内存区
IN	输入	REAL	V、L、常量、指针
OUT	输出	REAL	V、L、指针

ASIN 指令将 *IN* 求反正弦并将结果赋给 *OUT*，如下式所示：*OUT*=ARCSIN(*IN*)。

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

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

注意：结果是弧度值。

- LD

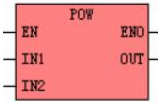
  若 *EN*值为 1，则指令被执行。

- IL

  如果 CR 值为 1，则指令被执行。指令的执行不影响 CR 值。

6.8.11 POW（指数运算）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	POW			☑ K2 ☑ KS ☑ MK ☑ K6
IL	POW	*POW IN1, IN2, OUT*	U	

参数	输入/输出	数据类型	允许的内存区
IN1	输入	REAL	V、L
IN2	输入	REAL	V、L、常量
OUT	输出	REAL	V、L

该指令用于求 *IN1* 为底数的 *IN2* 次方值并将结果赋给 *OUT*，如下式所示：$OUT = IN1^{IN2}$。

注意：若底数 *IN1* 为负数且指数 *IN2* 不为整数，则无法计算，将输出一个错误的结果；若 *IN1* 和 *IN2* 均为 0，本指令输出结果为 1，该结果仅有数学上的极限意义，但在实际应用中并无意义。

- LD

  若 *EN* 值为 1，则该指令被执行。

- IL

  如果 CR 值为 1，则该指令被执行。

  该指令的执行不影响 CR 值。

6.9 数组指令

新款 KPLC 提供了一组数组指令以方便用户的使用。支持数组指令的具体型号以及软件版本如下表。

CPU 型号	固件版本	编程软件版本
K209M	出厂初始任何版本均支持	KincoBulider V8.1.0.0 及以上版本
KS101M		
K6 系列		

6.9.1 数组指令综述

在实际应用编程中，我们会遇到对大量数据进行数学处理或者对某些单一逻辑多次重复调用等需求，如果使用基础逻辑指令和数学运算指令来处理则需要编写大量程序，使用不方便而且也占用程序空间。使用数组指令则可以有效解决此类问题，让程序变得简单明了。

数组是指用于存储多个具有**相同数据类型的数据**的集合，是在程序设计中，为了处理方便，把具有相同类型的若干元素组织在一起的一种形式。

KPLC 支持多个数组，每个数组均分配了一个唯一的编号，用户在程序中通过编号去操作相应的数组。在下文中，我们通常以“数组 N”来表示编号为 N 的数组。例如，K6 提供了 16 个数组，它们的编号依次为从 0 至 15 的整数，在用户程序中输入参数“*ARRAY*”为“*1*”的所有数组指令都是去操作“数组 1”（即编号为“1”的数组）。

数组内存放的各个数据称为数组的元素，每个数组内最多允许存放 1024 个元素。在一个数组内，为了区分各个元素，每个元素位置均分配了一个唯一的数字编号，这个编号称为下标，用户在程序中通过下标去操作本数组内相应的元素。例如，KPLC 单个数组内的元素最大为 1024 个，各元素的下标依次从 0 到 1023。

数组具有数据类型，当为一个数组指定了数据类型后，意味着本数组内的所有元素都具有相同的数据类型。例如，“数组 0”的数据类型为 INT，那么这个数组里所有元素的数据类型都是 INT 型。

下表举例描述了数组的构成。

数组编号	下标 0	下标 1	下标 2	下标 3	说明
8	DI#252	DI#-3	DI#16	DI#678	编号为 8 的数组（数组 8），其数据类型是 DINT，包含了下标为 0-3 总共 4 个元素。在本例中，数组 8 里下标为 2 的元素值为 DI#16。

为方便使用数组，KincoBuilder 提供了下述数组相关指令，均位于指令集的【数组指令】组中。

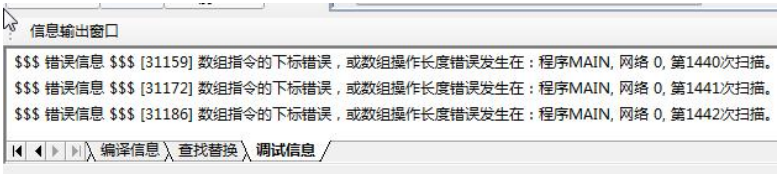
名称	功能描述
A_READ	读数组内的元素
A_WRITE	写数组内的元素
A_FILL	填充数组
A_GETSIZE	获取数组的长度（大小）
A_SETSIZE	设置数组的长度（大小）
A_GETTYPE	获取数组元素的数据类型
A_SETTYPE	设置数组元素的数据类型
A_MIN	求数组内指定范围内的数据最小值
A_MAX	求数组内指定范围内的数据最大值
A_AVE	求数组内指定范围内的数据平均值
A_SUM	求数组内指定范围内的数据总和值
A_SORT	对数组内指定范围内的数据排序

1） 注意事项

用户在使用这些指令时，需要注意如下几点：

- 只支持一维数组，不支持二维数组，三维数组等。
- 在一个用户工程中，可使用的数组编号共 16 个，编号依次为 0-15。而每个数组中最大元素数量为 1024，元素下标编号依次为 0-1023。
- 数组的元素个数可以用 A_SETSIZE 指令来指定。如果未指定，则默认支持最大 1024 个元素。但建议用户在使用时首先指定数组元素个数。
- 数组支持所有数据类型(BOOL,BYTE,WORD,DWORD,INT,DINT,REAL)，用户可以调用 A_SETTYPE 指令来指定数组的数据类型。如果未指定数据类型，则数组默认允许存放所有数据类型的元素，但建议用户首先为数组指定数据类型。A_MIN,A_MAX,A_AVE,A_SUM 等运算指令使用前必须用 A_SETTYPE 指令先指定数组的数据类型。

- 如果未对数组内的元素进行初始化赋值(可用 A_FILL 指令)，则各元素将按数据类型采用缺省的初始值,例如 BOOL 型初始值是 FALSE，REAL 型初始值是 0.0。
- 数组占用的内存空间是独立的，不占用 V、M 等基本内存空间。**数组不支持掉电保存！**
- 数组指令提供了执行错误等信息输出（如下图），但暂未提供指令执行错误输出等参数，建议用户使用前合理规划好数组的数据类型，元素，数组编号，长度等，以免出错。

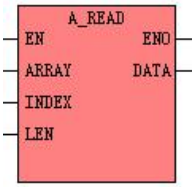


- 所有使用数组指令的输入参数需要注意不要超出允许的范围，如数组编号、元素下标等。数组指令执行出错后，会生成一个错误码，可用使用 KincoBuilder 读取。

错误码	描述
600	数组指令的数组编号错
601	数组指令的下标错误，或数组操作长度错误
602	数组指令操作了不支持的数据类型，或者与用户明确指定的数组数据类型不一致
603	数组操作长度错误，或者超出了用户明确指定的数组长度限制
606	数组指令的输入输出的内存区域不足够，例如超出了输出内存区的后边界
607	不支持的排序顺序

6.9.2 数组指令

6.9.2.1 A_READ（读数组元素）

	名称	指令格式	适用产品
LD	A_READ		☑ K209M ☑ KS101M ☑ K6

参数	输入/输出	数据类型	数值范围	允许使用的内存区
ARRAY	输入	INT	0-15	I、Q、V、M、L、常量
INDEX	输入	INT	0-1023	I、Q、V、M、L、常量
LEN	输入	INT	1-1024	I、Q、V、M、L、常量
DATA	输出	BOOL,BYTE，WORD,DWORD INT,DINT，REAL	----	V、M、L

各个参数的具体使用说明，见下表：

参数	功能
EN	使能端。若 *EN* 为 1 时，则该指令被使能，允许执行。
ARRAY	使用的数组编号，范围为 0-15
INDEX	待访问对象在数组中的起始索引地址，范围 0-1023
LEN	读取数组元素的个数，范围 1-1024
DATA	读取数据存放的起始地址

注意：*ARRAY*，*INDEX*，*LEN* 必须同时为常量或同时为变量,且这些参数组成了一个长度可变的内存块，此内存块必须全部位于合法的内存区域，否则数组指令执行错误。

该指令用于将编号为 *ARRAY* 的数组内从 *INDEX* 参数起始元素起连续 *LEN* 个元素复制传送到地址 *DATA* 开始的连续 *LEN* 个变量，传送的数据类型保持原有不变，传送的最大长度不应该超过最

大有效元素范围 1024，也不应该超过由 A_SETSIZE 指令指定的该数组的有效元素范围。

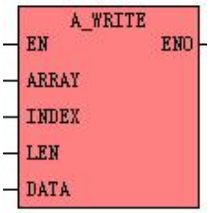
- LD 格式指令说明

如果 EN 为 1，则该指令被执行。

如果 EN 为 0，则指令不被扫描，也不会被执行。

- 指令用法，详情参见 6.9.3 数组使用举例。

6.9.2.2 A_WRITE（写数组元素）

	名称	指令格式	适用产品
LD	A_WRITE		☑ K209M ☑ KS101M ☑ K6

参数	输入/输出	数据类型	数值范围	允许使用的内存区
ARRAY	输入	INT	0-15	I、Q、V、M、L、常量
INDEX	输入	INT	0-1023	I、Q、V、M、L、常量
LEN	输入	INT	1-1024	I、Q、V、M、L、常量
DATA	输入	BOOL,BYTE， WORD,DWORD INT,DINT， REAL	----	I、Q、V、M、L

注意：ARRAY，INDEX，LEN 必须同时为常量或同时为变量,且这些参数组成了一个长度可变的内存块，此内存块必须全部位于合法的内存区域，否则数组指令执行错误。

注意：VWxxx 默认按 WORD 类型记录，VDxxx 默认按 DWORD 类型记录，如果希望记录成 INT 或

DINT 然后进行各种运算，则需要先在全局变量表声明称 INT 或 DINT,然后在 A_WRITE 的 DATA 参数输入。

各个参数的具体使用说明，见下表：

参数	功能
EN	使能端。若 *EN* 为 1 时，则该指令被使能，允许执行。
ARRAY	使用的数组编号，范围为 0-15
INDEX	待访问对象在数组中的起始索引地址，范围 0-1023
LEN	写入数组元素的个数，范围 1-1024
DATA	写入数据存放的起始地址

该指令用于将地址 *DATA* 开始的连续 *LEN* 个变量写入编号为 *ARRAY* 的数组内从 *INDEX* 参数起始元素开始的连续 *LEN* 个元素，写入的数据类型需跟 A_SETTYPE 指令指定该数组的数据类型一致。传送的最大长度不应该超过最大有效元素范围 1024，也不应该超过由 A_SETSIZE 指令指定的该数组的有效元素范围。

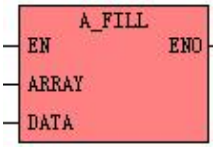
- LD 格式指令说明

如果 EN 为 1，则该指被执行。

如果 EN 为 0，则指令不被扫描，也不会被执行。

- 指令用法，详情参见 6.9.3 数组使用举例。

6.9.2.3 A_FILL（填充数组元素）

	名称	指令格式	适用产品
LD	A_FILL		☑ K209M ☑ KS101M ☑ K6

参数	输入/输出	数据类型	数值范围	允许使用的内存区
ARRAY	输入	INT	0-15	I、Q、V、M、L、常量
DATA	输入	BOOL,BYTE，WORD,DWORD INT,DINT，REAL	----	I、Q、V、M、L、常量

各个参数的具体使用说明，见下表：

参数	功能
EN	使能端。若 *EN* 为 1 时，则该指令被使能，允许执行。
ARRAY	使用的数组编号，范围为 0-15
DATA	填充数组元素的数值

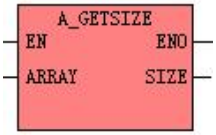
该指令用于将编号为 *ARRAY* 的数组内的所有有效元素赋值为 *DATA* 的数值，注意 *DATA* 的数据类型需跟该数组的数据类型一致。

- LD 格式指令说明

如果 EN 为 1，则该指令被执行。如果 EN 为 0，则指令不被扫描，也不会被执行。

- 指令用法，详情参见 6.9.3 数组使用举例。

6.9.2.4 A_GETSIZE（获取数组元素的有效范围）

	名称	指令格式	适用产品
LD	A_GETSIZE		☑ K209M ☑ KS101M ☑ K6

参数	输入/输出	数据类型	数值范围	允许使用的内存区
ARRAY	输入	INT	0-15	I、Q、V、M、L、常量
SIZE	输出	INT	1-1024	V、M、L

各个参数的具体使用说明，见下表：

参数	功能
EN	使能端。若 *EN* 为 1 时，则该指令被使能，允许执行。
ARRAY	使用的数组编号，范围为 0-15
SIZE	数组元素的有效长度，范围为 1-1024

该指令用于获取编号为 *ARRAY* 的数组的大小（即元素个数），并将结果写入到 SIZE。

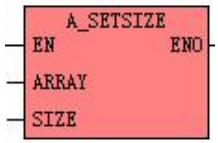
- LD 格式指令说明

如果 EN 为 1，则该指令执行。

如果 EN 为 0，则指令不被扫描，也不会被执行。

- 指令用法，详情参见 6.9.3 数组使用举例。

6.9.2.5 A_SETSIZE（设置数组元素的有效范围）

	名称	指令格式	适用产品
LD	A_SETSIZE		☑ K209M ☑ KS101M ☑ K6

参数	输入/输出	数据类型	数值范围	允许使用的内存区
ARRAY	输入	INT	0-15	I、Q、V、M、L、常量
SIZE	输入	INT	1-1024	I、Q、V、M、L

各个参数的具体使用说明，见下表：

参数	功能
EN	使能端。若 *EN* 为 1 时，则该指令被使能，允许执行。
ARRAY	使用的数组编号，范围为 0-15
SIZE	数组元素的有效长度，范围为 1-1024

该指令用于将编号为 *ARRAY* 的数组最大允许的元素个数设定为 *SIZE*。

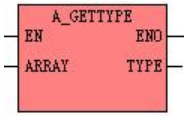
- LD 格式指令说明

如果 EN 为 1，则该指令被执行。

如果 EN 为 0，则指令不被扫描，也不会被执行。

- 指令用法，详情参见 6.9.3 数组使用举例。

6.9.2.6 A_GETTYPE（获取数组元素的数据类型）

	名称	指令格式	适用产品
LD	A_GETTYPE		☑ K209M ☑ KS101M ☑ K6

参数	输入/输出	数据类型	数值范围	允许使用的内存区
ARRAY	输入	INT	0-15	I、Q、V、M、L、常量
TYPE	输出	INT		V、M、L

各个参数的具体使用说明，见下表：

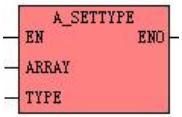
参数	功能
EN	使能端。若 *EN* 为 1 时，则该指令被使能，允许执行。
ARRAY	使用的数组编号，范围为 0-15
TYPE	数组元素的数据类型，具体值如下： 默认的类型 0 BOOL_TYPE 1 BYTE_TYPE 2 WORD_TYPE 3 INT_TYPE 4 DWORD_TYPE 5 DINT_TYPE 6 REAL_TYPE 7

该指令用于获取编号为 *ARRAY* 的数组的数据类型，并将结果写入到 *TYPE*。

- LD 格式指令说明

如果 EN 为 1，则该指令被执行。如果 EN 为 0，则指令不被扫描，也不会被执行。

6.9.2.7 A_SETTYPE（设定数组元素的数据类型）

	名称	指令格式	适用产品
LD	A_SETTYPE		☑ K209M ☑ KS101M ☑ K6

参数	输入/输出	数据类型	数值范围	允许使用的内存区
ARRAY	输入	INT	0-15	I、Q、V、M、L、常量
TYPE	输入	INT	0-7	I、Q、V、M、L

各个参数的具体使用说明，见下表：

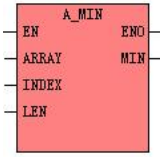
参数	功能
EN	使能端。若 *EN* 为 1 时，则该指令被使能，允许执行。
ARRAY	使用的数组编号，范围为 0-15
TYPE	数组元素的数据类型： 默认的类型 0 BOOL_TYPE 1 BYTE_TYPE 2 WORD_TYPE 3 INT_TYPE 4 DWORD_TYPE 5 DINT_TYPE 6 REAL_TYPE 7

该指令用于将编号为 *ARRAY* 的数组支持的数据类型设置为 *TYPE*。

- LD 格式指令说明

如果 EN 为 1，则该指令执行。如果 EN 为 0，则指令不被扫描，也不会被执行。

6.9.2.8 A_MIN（求数组内指定范围内的数据最小值）

	名称	指令格式	适用产品
LD	A_MIN		☑ K209M ☑ KS101M ☑ K6

参数	输入/输出	数据类型	数值范围	允许使用的内存区
ARRAY	输入	INT	0-15	I、Q、V、M、L、常量
INDEX	输入	INT	0-1023	I、Q、V、M、L、常量
LEN	输入	INT	1-1024	I、Q、V、M、L、常量
MIN	输出	BYTE，WORD,DWORD，INT,DINT，REAL	----	V、M、L

各个参数的具体使用说明，见下表：

参数	功能
EN	使能端。若 *EN* 为 1 时，则该指令被使能，允许执行。
ARRAY	使用的数组编号，范围为 0-15
INDEX	待访问对象在数组中的起始索引地址，范围 0-1023
LEN	进行数据运算的数组元素的个数，范围 1-1024
MIN	运算得出的最小值存放的内存地址

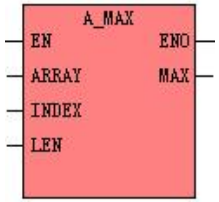
注意：*ARRAY, INDEX, LEN* 必须同时为常量或同时为变量！ 该指令使用前必须对数组用 A_SETTYPE 指令先设置数据类型，且该指令不支持 BOOL 型数组的运算。

该指令用于对编号为 *ARRAY* 的数组内从 *INDEX* 开始的连续 *LEN* 个元素进行取最小值的运算，得出的最小值存放到 *MIN* 指定的内存地址。

- LD 格式指令说明

如果 EN 为 1，则该指令执行。如果 EN 为 0，则指令不被扫描，也不会被执行。

6.9.2.9 A_MAX（求数组内指定范围内的数据最大值）

	名称	指令格式	适用产品
LD	A_MAX		☑ K209M ☑ KS101M ☑ K6

参数	输入/输出	数据类型	数值范围	允许使用的内存区
ARRAY	输入	INT	0-15	I、Q、V、M、L、常量
INDEX	输入	INT	0-1023	I、Q、V、M、L、常量
LEN	输入	INT	1-1024	I、Q、V、M、L、常量
MAX	输出	BYTE，WORD,DWORD INT,DINT，REAL	----	V、M、L

各个参数的具体使用说明，见下表：

参数	功能
EN	使能端。若 *EN* 为 1 时，则该指令被使能，允许执行。
ARRAY	使用的数组编号，范围为 0-15
INDEX	待访问对象在数组中的起始索引地址，范围 0-1023
LEN	进行数据运算的数组元素的个数，范围 1-1024
MAX	运算得出的最大值存放的内存地址

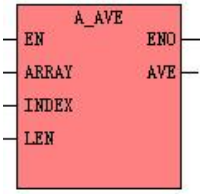
注意：*ARRAY, INDEX, LEN* 必须同时为常量或同时为变量！ 该指令使用前必须对数组用 A_SETTYPE 指令先设置数据类型，且该指令不支持 BOOL 型数组的运算。

该指令用于对编号为 *ARRAY* 的数组内从 *INDEX* 开始的连续 *LEN* 个元素进行取最大值的运算，得出的最小值存放到 *MAX* 指定的内存地址。

- LD 格式指令说明

如果 EN 为 1，则该指令执行。如果 EN 为 0，则指令不被扫描，也不会被执行。

6.9.2.10 A_AVE（求数组内指定范围内的数据平均值）

	名称	指令格式	适用产品
LD	A_AVE		☑ K209M ☑ KS101M ☑ K6

参数	输入/输出	数据类型	数值范围	允许使用的内存区
ARRAY	输入	INT	0-15	I、Q、V、M、L、常量
INDEX	输入	INT	0-1023	I、Q、V、M、L、常量
LEN	输入	INT	1-1024	I、Q、V、M、L、常量
AVE	输出	BYTE，WORD,DWORD INT,DINT，REAL	----	V、M、L

各个参数的具体使用说明，见下表：

参数	功能
EN	使能端。若 *EN* 为 1 时，则该指令被使能，允许执行。
ARRAY	使用的数组编号，范围为 0-15
INDEX	待访问对象在数组中的起始索引地址，范围 0-1023
LEN	进行数据运算的数组元素的个数，范围 1-1024
AVE	运算得出的平均值存放的内存地址

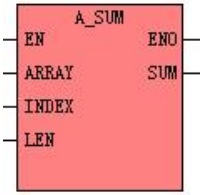
注意：*ARRAY, INDEX, LEN* 必须同时为常量或同时为变量！ 该指令使用前必须对数组用 A_SETTYPE 指令先设置数据类型，且该指令不支持 BOOL 型数组的运算。

该指令用于将编号为 *ARRAY* 的数组内从 *INDEX* 开始的连续 *LEN* 个元素进行求平均值的运算，得出的平均值存放到 *AVE* 指定的内存地址

- LD 格式指令说明

如果 EN 为 1，则该指令执行。如果 EN 为 0，则指令不被扫描，也不会被执行。

6.9.2.11 A_SUM（求数组内指定范围内的数据总和值）

	名称	指令格式	适用产品
LD	A_SUM		☑ K209M ☑ KS101M ☑ K6

参数	输入/输出	数据类型	数值范围	允许使用的内存区
ARRAY	输入	INT	0-15	I、Q、V、M、L、常量
INDEX	输入	INT	0-1023	I、Q、V、M、L、常量
LEN	输入	INT	1-1024	I、Q、V、M、L、常量
SUM	输出	BOOL，BYTE，WORD,DWORD，INT,DINT，REAL	----	V、M、L

注意：*ARRAY, INDEX, LEN* 必须同时为常量或同时为变量！ 该指令使用前必须对数组用 A_SETTYPE 指令先设置数据类型，且该指令不支持 BOOL 型数组的运算。

各个参数的具体使用说明，见下表：

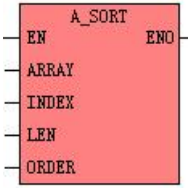
参数	功能
EN	使能端。若 *EN* 为 1 时，则该指令被使能，允许执行。
ARRAY	使用的数组编号，范围为 0-15
INDEX	待访问对象在数组中的起始索引地址，范围 0-1023
LEN	进行数据运算的数组元素的个数，范围 1-1024
SUM	运算得出的总和值存放的内存地址

该指令用于将编号为 *ARRAY* 的数组内从 *INDEX* 开始的连续 *LEN* 个元素进行求和的运算，计算结果存放到 *SUM* 指定的内存地址

- LD 格式指令说明

如果 EN 为 1，则该指令执行。如果 EN 为 0，则指令不被扫描，也不会被执行。

6.9.2.12 A_SORT（对数组内指定范围内的数据排序）

	名称	指令格式	适用产品
LD	A_SORT		☑ K209M ☑ KS101M ☑ K6

参数	输入/输出	数据类型	数值范围	允许使用的内存区
ARRAY	输入	INT	0-15	I、Q、V、M、L、常量
INDEX	输入	INT	0-1023	I、Q、V、M、L、常量
LEN	输入	INT	1-1024	I、Q、V、M、L、常量
ORDER	输入	INT	升序 1 降序 2	I、Q、V、M、L、常量

各个参数的具体使用说明，见下表：

参数	功能
EN	使能端。若 *EN* 为 1 时，则该指令被使能，允许执行。
ARRAY	使用的数组编号，范围为 0-15
INDEX	待访问对象在数组中的起始索引地址，范围 0-1023
LEN	进行数据运算的数组元素的个数，范围 1-1024
ORDER	数据排序的顺序，1 为升序，2 为降序

注意：*ARRAY, INDEX, LEN* 必须同时为常量或同时为变量！ 该指令使用前必须对数组用 A_SETTYPE 指令先设置数据类型！

该指令用于将编号为 *ARRAY* 的数组内从 *INDEX* 开始的连续 *LEN* 个元素进行排序，*ORDER* 参数值指明了升序（从小到大）或者降序（从大到小）。

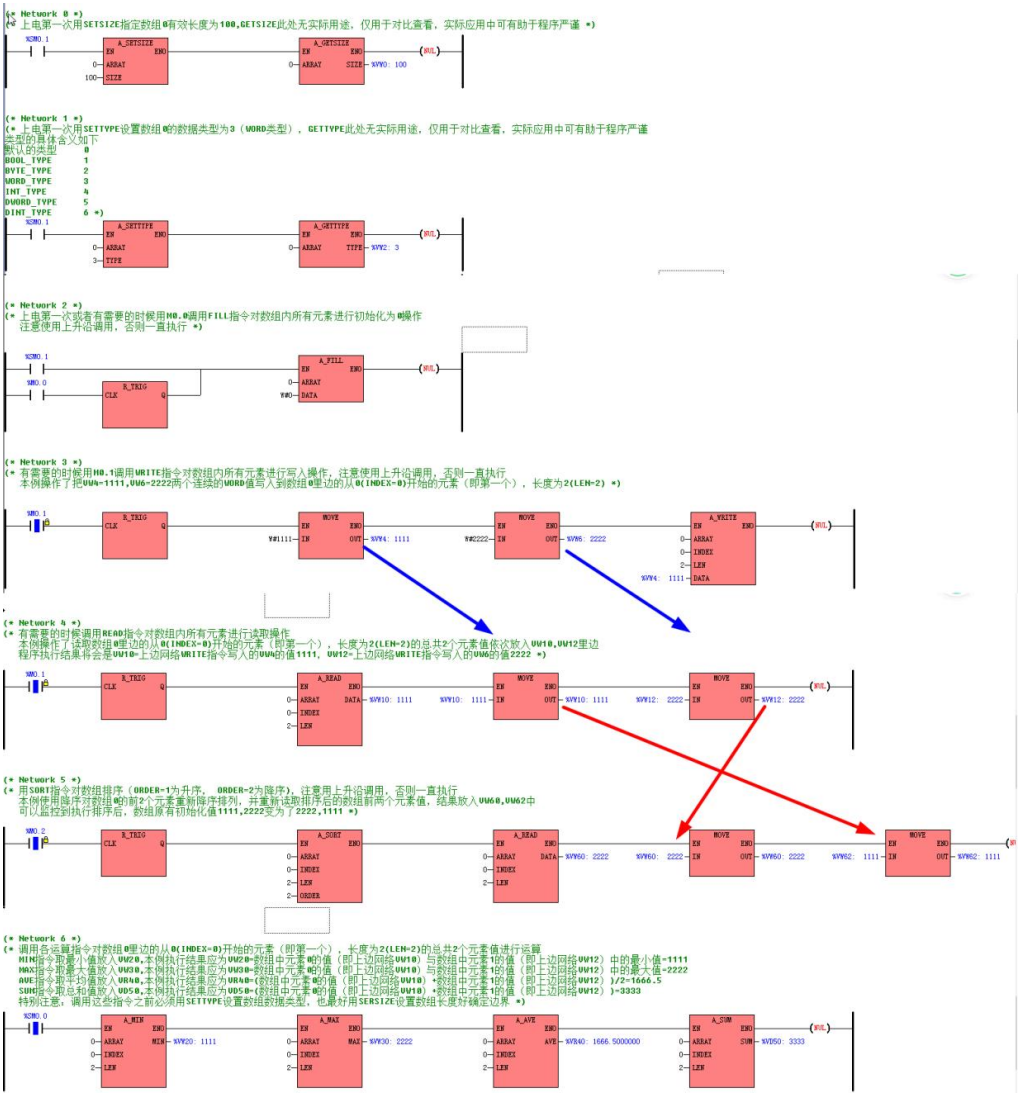
- LD 格式指令说明

如果 EN 为 1，则该指令执行。如果 EN 为 0，则指令不被扫描，也不会被执行。

6.9.3 数组使用举例

6.9.3.1 各指令基础说明演示

（一）：LD 格式（梯形图）程序（程序例子可在 www.kinco.cn 网站下载）



IL 格式（指令）程序 可以直接粘贴到软件中转换成梯形图。

```
(* Network 0 *)
(*上电第一次用 SETSIZE 指定数组 0 有效长度为 100,GETSIZE 此处无实际用途,仅用于对比查看，实际应用中可有助于程序严谨*)
LD      %SM0.1
A_SETSIZE 0, 100
A_GETSIZE 0, %VW0
(* Network 1 *)
(*上电第一次用 SETTYPE 设置数组 0 的数据类型为 3（WORD 类型），GETTYPE 此处无实际用途，仅用于对比查看,实际应用中可有助于程序严谨类型的具体含义如下默认的类型      0BOOL_TYPE 1BYTE_TYPE      2WORD_TYPE      3INT_TYPE      4DWORD_TYPE      5DINT_TYPE      6*)
LD      %SM0.1
A_SETTYPE 0, 3
A_GETTYPE 0, %VW2
(* Network 2 *)
(*上电第一次或者有需要的时候用 M0.0 调用 FILL 指令对数组内所有元素进行初始化为 0 操作注意使用上升沿调用，否则一直执行*)
LD      %SM0.1
OR(
LD      %M0.0
R_TRIG
)
A_FILL  0, W#0
(* Network 3 *)
(*有需要的时候用 M0.1 调用 WRITE 指令对数组内所有元素进行写入操作，注意使用上升沿调用，否则一直执行    本例操作了把 VW4=1111,VW6=2222 两个连续的 WORD 值写入到数组 0 里边的从 0(INDEX=0)开始的元素（即第一个），长度为 2(LEN=2)*)
```

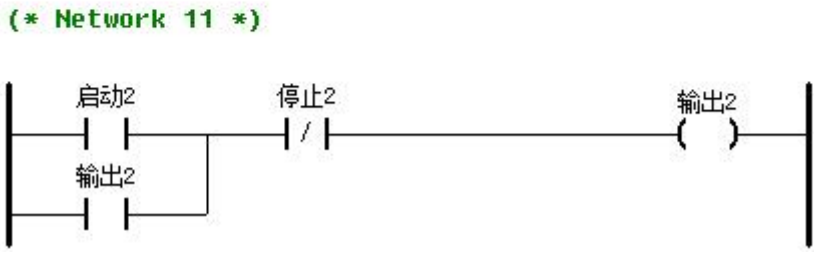
```
LD      %M0.1
R_TRIG
MOVE    W#1111, %VW4
MOVE    W#2222, %VW6
A_WRITE 0, 0, 2, %VW4
(* Network 4 *)
(*有需要的时候调用 READ 指令对数组内所有元素进行读取操作本例操作了读取数组 0 里边的从 0(INDEX=0)开始的元素（即第一个），长度为 2(LEN=2)的总共 2 个元素值依次放入 VW10,VW12 里边程序执行结果将会是 VW10=上边网络 WRITE 指令写入的 VW4 的值 1111，VW12=上边网络 WRITE 指令写入的 VW6 的值 2222*)
LD      %M0.1
R_TRIG
A_READ  0, 0, 2, %VW10
MOVE    %VW10, %VW10
MOVE    %VW12, %VW12
(* Network 5 *)
(*用 SORT 指令对数组排序（ORDER=1 为升序， ORDER=2 为降序)，注意用上升沿调用，否则一直执行    本例使用降序对数组 0 的前 2 个元素重新降序排列,并重新读取排序后的数组前两个元素值，结果放入 VW60,VW62 中可以监控到执行排序后，数组原有初始化值 1111,2222 变为了 2222,1111*)
LD      %M0.2
R_TRIG
A_SORT  0, 0, 2, 2
A_READ  0, 0, 2, %VW60
MOVE    %VW60, %VW60
MOVE    %VW62, %VW62
(* Network 6 *)
```

(*调用各运算指令对数组 0 里边的从 0(INDEX=0)开始的元素（即第一个），长度为 2(LEN=2)的总共 2 个元素值进行运算　MIN 指令取最小值放入 VW20,本例执行结果应为 VW20=数组中元素 0 的值（即上边网络 VW10）与数组中元素 1 的值（即上边网络 VW12）中的最小值=1111　MAX 指令取最大值放入 VW30,本例执行结果应为 VW30=数组中元素 0 的值（即上边网络 VW10）与数组中元素 1 的值（即上边网络 VW12）中的最大值=2222　AVE 指令取平均值放入 VR40,本例执行结果应为 VR40=(数组中元素 0 的值（即上边网络 VW10）+数组中元素 1 的值（即上边网络 VW12）)/2=1666.5 SUM 指令取总和值放入 VD50,本例执行结果应为 VD50=(数组中元素 0 的值（即上边网络 VW10）+数组中元素 1 的值（即上边网络 VW12）)=3333　特别注意：调用这些指令之前必须用 SETTYPE 设置数组数据类型，也最好用 SERSIZE 设置数组长度好确定边界*)

```
LD      %SM0.0
A_MIN   0, 0, 2, %VW20
A_MAX   0, 0, 2, %VW30
A_AVE   0, 0, 2, %VR40
A_SUM   0, 0, 2, %VD50
```

6.9.3.2 数组应用演示一：逻辑处理

在实际编程中，经常需要对某些动作逻辑进行同样的处理，如下边梯形图所示常见手动启停电路以及走马灯输出之类的。



这些动作都是单一重复的，如果用基本指令处理，需要有多少个动作就复制一模一样的程序，如 32 组以上，这样程序就看起来比较繁琐，同理对于走马灯输出，如果是规律的单个轮换输出还比较好处理，但如果比如每隔 4 个输出，或者每次规则都不一样的输出等用传统指令都不好处理，而利用数组则可以有效解决这个问题。下文以利用数组结合循环指令等来解决 32 组手动启停输出的演示例子说明，只需要简单几行程序即可完成原有的需要复制 32 行程序，且该程序理论上增加到最大 1024 组都是同样的语句，无需再增加语句。

本演示例子仅供纯粹演示数组结合循环这一类的最常使用方式，实际应用场景还有很多其他因素需要考虑，也可以用子程序，指针等其他方式来解决此类问题，本例仅供参考而不能做实际应用。

本例需求：共有 32 组输入启动和停止信号，完成输出

程序思路：

- 分配 ID0 连续的 32 位给 32 组输入启动信号，同理分配 ID4 给停止信号。

- 分配三个数组 1,2,3，数据类型本例都是对 BOOL 动作处理，所以为 BOOL 类型。数组 1 给启动信号用，数组 2 给停止信号用，数组 3 给结果输出用。
- 编程思路为：把输入信号都用 A_WRITE 指令统一存放到输入数组中，这样用 A_READ 指令读取出来就可以分解为单独一个个的元素，从而利用 FOR 循环依次对 32 组一次性处理，处理完的结果同样需要用 A_WRITE 指令统一依次连续存放到输出数组中，最后同样可以用 A_READ 指令读取输出数组中的各元素值供程序其他地方使用。

（一）：LD 格式（梯形图）程序（程序例子可在 www.kinco.cn 网站下载）

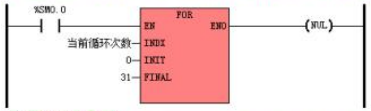
(* Network 0 *)
(* 初始化，设定三个数组全部为BOOL型数组类型（TYPE=1）
数组1为启动输入数组，数组2为停止输入数组，数组3为结果输出数组 *)

(* Network 1 *)
(* 设置数组的一些固定值，主要是因为READ和WIRTE指令限制了指令输入参数必须同时为变量类型 *)

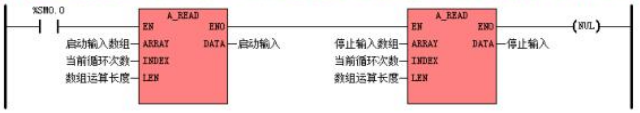
(* Network 2 *)
(* 设置数组的一些固定值，主要是因为READ和WIRTE指令限制了指令输入参数必须同时为变量类型 *)

(* Network 3 *)
(* 把启动输入信号ID0全部32个输入点用WIRTE指令传送到数组1，为启动输入数组
把停止输入信号ID4全部32个输入点用WIRTE指令传送到数组2，为停止输入数组， *)

(* Network 4 *)
(* 执行一个周期循环32次（从INIT=0到FINAL=31)的FOR循环指令，INDX是实际循环次数 *)



(* Network 5 *)
(* 读取启动输入数组（存放的是ID0）和停止输入数组（存放的是ID4）的当前元素值（共0-31的32个元素），好方便后续逻辑运算 *)



(* Network 6 *)
(* 执行逻辑运算，启动且非停止时执行输出 *)

(* Network 7 *)
(* 把运算的每一次结果单次一个个顺序存入到结果输出数组中 *)

(* Network 8 *)
(* 结束循环运算 *)

(* Network 9 *)
(* 结果输出数组的单个元素的BOOL值就是最终运算的结果，取出来供程序别的地方利用。比如直接给输出点 *)

IL 格式（指令）程序 可以直接粘贴到软件中转换成梯形图。

```
(* Network 0 *)
(*初始化，设定三个数组全部为 BOOL 型数组类型（TYPE=1） 数组 1 为启动输入数组，数组 2 为停止输入数组，数组 3 为结果输出数组*)
LD      %SM0.1
A_SETTYPE 1, 1
A_SETTYPE 2, 1
A_SETTYPE 3, 1
(* Network 1 *)
(*设置数组的一些固定值,主要是因为 READ 和 WIRTE 指令限制了指令输入参数必须同时为变量类型*)
LD      %SM0.1
MOVE    1, %VW0
MOVE    3, %VW20
(* Network 2 *)
(*设置数组一些固定值，因为 READ 和 WIRTE 指令限制了指令输入参数必须同时为变量类型*)
LD      %SM0.1
MOVE    2, %VW10
MOVE    1, %VW4
(* Network 3 *)
(*把启动输入信号 ID0 全部 32 个输入点用 WIRTE 指令传送到数组 1，为启动输入数组   把停止输入信号 ID4 全部 32 个输入点用 WIRTE 指令传送到数组 2，为停止输入数组，*)
LD      %SM0.0
A_WRITE 1, 0, 32, %I0.0
A_WRITE 2, 0, 32, %I4.0
(* Network 4 *)
(*执行一个周期循环 32 次（从 INIT=0 到 FINAL=31)的 FOR 循环指令,INDX 是实际循环次数*)
```

```
LD      %SM0.0
FOR     %VW30, 0, 31
(* Network 5 *)
(*读取启动输入数组（存放的是 ID0）和停止输入数组（存放的是 ID4）的当前元素值（共 0-31 的 32 个元素），好方便后续逻辑运算*)
LD      %SM0.0
A_READ  %VW0, %VW30, %VW4, %M10.0
A_READ  %VW10, %VW30, %VW4, %M20.0
(* Network 6 *)
(*执行逻辑运算，启动且非停止时执行输出*)
LD      %M10.0
ANDN    %M20.0
ST      %M30.0
(* Network 7 *)
(*把运算的每一次结果单次一个个顺序存入到结果输出数组中*)
LD      %SM0.0
A_WRITE %VW20, %VW30, %VW4, %M30.0
(* Network 8 *)
(*结束循环运算*)
LD      %SM0.0
NEXT
(* Network 9 *)
(*结果输出数组的单个元素的 BOOL 值就是最终运算的结果，取出来供程序别的地方利用。比如直接给输出点*)
LD      %SM0.0
A_READ  3, 0, 32, %M40.0
MOVE    %MD40, %QD0
```

6.9.3.3 数组应用演示二：报表输出

在实际编程中，经常需要对一些连续的数据需要记录下来，然后需要对记录的数据进行数学处理或者查找符合某一条件的数据，这类的报表应用用传统指令处理即使结合指针使用程序也比较麻烦，而数组则最适合这一类的应用。

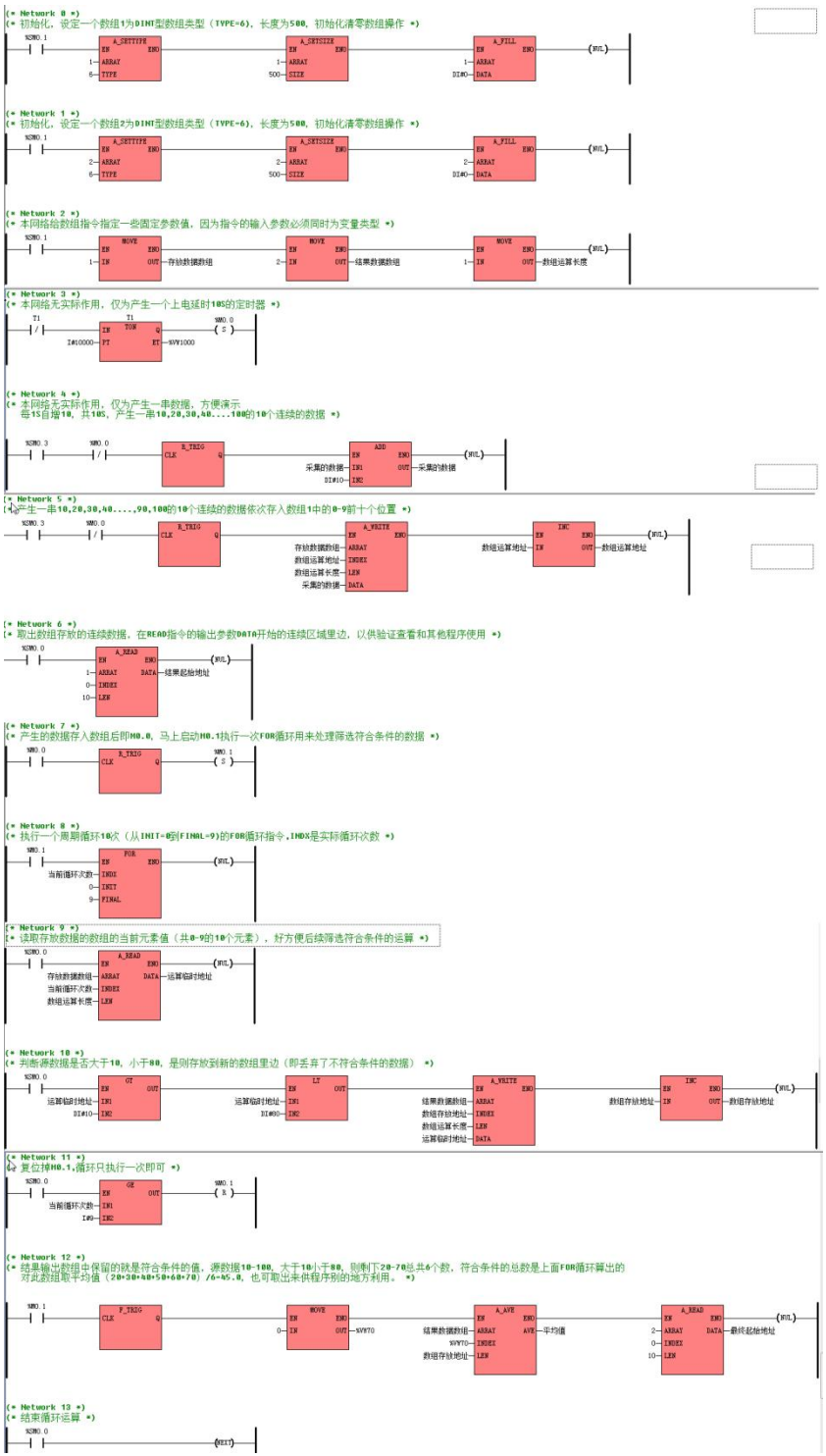
本演示例子仅供纯粹演示数组结合循环这一类的最常使用方式，实际应用场景还有很多其他因素需要考虑，也可以用子程序，指针等其他方式来解决此类问题，本例仅供参考而不能做实际应用。

本例需求：采集一组参数，去掉小于 10 以及大于 80 的值后，求其平均值

程序思路：

- 分配二个数组 1,2，数据类型本例都是 DINT 类型，数组 1 给存放源数据用，数组 2 给存放处理符合条件后的数据用。
- 编程思路为：把源数据用 A_WRITE 指令统一依次存放到输入数组中，这样用 A_READ 指令读取出来就可以分解为单独一个个的元素，从而利用 FOR 循环依次取出来每一个元素跟条件进行比对（大于 10 小于 80），处理完的结果同样需要用 A_WRITE 指令统一依次连续存放到输出数组中，最后同样可以用 A_READ 指令读取输出数组中的各元素值取平均值和供程序其他地方使用。

（一）：LD 格式（梯形图）程序（程序例子可在 www.kinco.cn 网站下载）。


(* Network 0 *)
(* 初始化，设定一个数组1为DINT型数组类型（TYPE=6），长度为500，初始化清零数组操作 *)
%M0.1
A_SETTYPE
EN ENO
1—ARRAY
6—TYPE
A_SETSIZE
EN ENO
1—ARRAY
500—SIZE
A_FILL
EN ENO
1—ARRAY
DI#0—DATA
(NUL)
(* Network 1 *)
(* 初始化，设定一个数组2为DINT型数组类型（TYPE=6），长度为500，初始化清零数组操作 *)
%M0.1
A_SETTYPE
EN ENO
2—ARRAY
6—TYPE
A_SETSIZE
EN ENO
2—ARRAY
500—SIZE
A_FILL
EN ENO
2—ARRAY
DI#0—DATA
(NUL)
(* Network 2 *)
(* 本网络给数组指令指定一些固定参数值，因为指令的输入参数必须同时为变量类型 *)
%M0.1
MOVE
EN ENO
1—IN
OUT—存放数据数组
MOVE
EN ENO
2—IN
OUT—结果数据数组
MOVE
EN ENO
1—IN
OUT—数组运算长度
(NUL)
(* Network 3 *)
(* 本网络无实际作用，仅为产生一个上电延时10S的定时器 *)
T1
T1
TON
IN Q
I#10000—PT ET—%W1000
%M0.0
(S)
(* Network 4 *)
(* 本网络无实际作用，仅为产生一串数据，方便演示
每1S自增10，共10S，产生一串10,20,30,40....100的10个连续的数据 *)
%M0.3
%M0.0
R_TRIG
CLK Q
ADD
EN ENO
采集的数据—IN1 OUT—采集的数据
DI#10—IN2
(NUL)
(* Network 5 *)
(* 产生一串10,20,30,40....,90,100的10个连续的数据依次存入数组1中的0-9前十个位置 *)
%M0.3
%M0.0
R_TRIG
CLK Q
A_WRITE
EN ENO
存放数据数组—ARRAY
数组运算地址—INDEX
数组运算长度—LEN
采集的数据—DATA
INC
EN ENO
数组运算地址—IN OUT—数组运算地址
(NUL)
(* Network 6 *)
(* 取出数组存放的连续数据，在READ指令的输出参数DATA开始的连续区域里边，以供验证查看和其他程序使用 *)
%M0.0
A_READ
EN ENO
1—ARRAY DATA—结果起始地址
0—INDEX
10—LEN
(NUL)
(* Network 7 *)
(* 产生的数据存入数组后即M0.0，马上启动M0.1执行一次FOR循环用来处理筛选符合条件的数据 *)
%M0.0
R_TRIG
CLK Q
%M0.1
(S)
(* Network 8 *)
(* 执行一个周期循环10次（从INIT=0到FINAL=9）的FOR循环指令，INDX是实际循环次数 *)
%M0.1
FOR
EN ENO
当前循环次数—INDX
0—INIT
9—FINAL
(NUL)
(* Network 9 *)
(* 读取存放数据的数组的当前元素值（共0-9的10个元素），好方便后续筛选符合条件的运算 *)
%M0.0
A_READ
EN ENO
存放数据数组—ARRAY DATA—运算临时地址
当前循环次数—INDEX
数组运算长度—LEN
(NUL)
(* Network 10 *)
(* 判断源数据是否大于10，小于80，是则存放到新的数组里边（即丢弃了不符合条件的数据） *)
%M0.0
GT
EN ENO
运算临时地址—IN1
DI#10—IN2
LT
EN ENO
运算临时地址—IN1
DI#80—IN2
A_WRITE
EN ENO
结果数据数组—ARRAY
数组存放地址—INDEX
数组运算长度—LEN
运算临时地址—DATA
INC
EN ENO
数组存放地址—IN OUT—数组存放地址
(NUL)
(* Network 11 *)
(* 复位掉M0.1，循环只执行一次即可 *)
%M0.0
GE
EN ENO
当前循环次数—IN1
19—IN2
%M0.1
(R)
(* Network 12 *)
(* 结果输出数组中保留的就是符合条件的值，源数据10-100，大于10小于80，则剩下20-70总共6个数，符合条件的总数是上面FOR循环算出的
对此数组取平均值（20+30+40+50+60+70）/6=45.0，也可取出来供程序别的地方利用。 *)
%M0.1
F_TRIG
CLK Q
MOVE
EN ENO
0—IN
OUT—%VW70
A_AVE
EN ENO
结果数据数组—ARRAY
%VW70—INDEX
数组存放地址—LEN
AVE—平均值
A_READ
EN ENO
2—ARRAY
0—INDEX
10—LEN
DATA—最终起始地址
(NUL)
(* Network 13 *)
(* 结束循环运算 *)
%M0.0
(NEXT)

IL 格式（指令）程序 可以直接粘贴到软件中转换成梯形图。

```
(* Network 0 *)
(*初始化，设定一个数组 1 为 DINT 型数组类型（TYPE=6），长度为 500，初始化清零数组操作*)
LD      %SM0.1
A_SETTYPE 1, 6
A_SETSIZE 1, 500
A_FILL  1, DI#0
(* Network 1 *)
(*初始化，设定一个数组 2 为 DINT 型数组类型（TYPE=6），长度为 500，初始化清零数组操作*)
LD      %SM0.1
A_SETTYPE 2, 6
A_SETSIZE 2, 500
A_FILL  2, DI#0
(* Network 2 *)
(*本网络给数组指令指定一些固定参数值，因为指令的输入参数必须同时为变量类型*)
LD      %SM0.1
MOVE    1, %VW10
MOVE    2, %VW40
MOVE    1, %VW20
(* Network 3 *)
(*本网络无实际作用，仅为产生一个上电延时 10S 的定时器*)
LDN     T1
TON     T1, I#10000
__CR_EQ_1
MOVE    T1, %VW1000
__CR_RESTORE
S       %M0.0
```

```
(* Network 4 *)
(*本网络无实际作用，仅为产生一串数据，方便演示每 1S 自增 10，共 10S，产生一串 10,20,30,40....100 的 10 个连续的数据*)
LD      %SM0.3
ANDN    %M0.0
R_TRIG
ADD     DI#10, %VD0
(* Network 5 *)
(*产生一串 10,20,30,40....,90,100 的 10 个连续的数据依次存入数组 1 中的 0-9 前十个位置*)
LD      %SM0.3
ANDN    %M0.0
R_TRIG
A_WRITE %VW10, %VW4, %VW20, %VD0
INC     %VW4
(* Network 6 *)
(*取出数组存放的连续数据，在 READ 指令的输出参数 DATA 开始的连续区域里边，以供验证查看和其他程序使用*)
LD      %SM0.0
A_READ  1, 0, 10, %VD100
(* Network 7 *)
(*产生的数据存入数组后即 M0.0，马上启动 M0.1 执行一次 FOR 循环用来处理筛选符合条件的数据*)
LD      %M0.0
R_TRIG
S       %M0.1
(* Network 8 *)
(*执行一个周期循环 10 次（从 INIT=0 到 FINAL=9)的 FOR 循环指令,INDX 是实际循环次数*)
```

```
LD      %M0.1
FOR     %VW30, 0, 9
(* Network 9 *)
(*读取存放数据的数组的当前元素值（共 0-9 的 10 个元素），好方便后续筛选符合条件的运算*)
LD      %SM0.0
A_READ  %VW10, %VW30, %VW20, %VD50
(* Network 10 *)
(*判断源数据是否大于 10，小于 80，是则存放到新的数组里边（即丢弃了不符合条件的数据）*)
LD      %SM0.0
GT      %VD50, DI#10
LT      %VD50, DI#80
A_WRITE %VW40, %VW6, %VW20, %VD50
INC     %VW6
(* Network 11 *)
(*复位掉 M0.1,循环只执行一次即可*)
LD      %SM0.0
GE      %VW30, I#9
R       %M0.1
(* Network 12 *)
(*结果输出数组中保留的就是符合条件的值，源数据 10-100，大于 10 小于 80，则剩下 20-70 总共 6 个数,符合条件的总数是上面 FOR 循环算出的对此数组取平均值(20+30+40+50+60+70)/6=45.0，也可取出来供程序别的地方利用。*)
LD      %M0.1
F_TRIG
MOVE    0, %VW70
A_AVE   %VW40, %VW70, %VW6, %VR60
A_READ  2, 0, 10, %VD200
```

```
(* Network 13 *)
(*结束循环运算*)
LD      %SM0.0
NEXT
```

6.10 堆栈指令

新款 KPLC 提供了一组堆栈指令以方便用户的使用。具体支持指令的型号以及相关软件版本如下表。

CPU 型号	固件版本	编程软件版本
K209M	出厂初始任何版本均支持	KincoBulider V8.1.0.0 及以上版本
KS101M		
K6 系列		

6.10.1 堆栈指令综述

堆栈是常用的一种数据结构，一般用于保存数据。所谓“栈”，可以理解为一列有着规定最大层数的货架。在程序设计中，可以使用压栈指令（又称装栈，PUSH）压入数据到“栈”内的各层内（即存放货物），存放顺序是可以指定先入先出（FIFO）或者后入先出(LIFO)，系统后台自动按规则把数据存放到各层。而当需要使用“栈”内数据时可以使用弹栈指令（又称出栈，POP）取出数据（即取出货物），取出的顺序为存入时设定的顺序。为了更加清楚堆栈构成，可参阅下图举例的描述。

FIFO 先入先出队列	初始 状态	装载数据1 第1次执行 PUSH指令后	装载数据2 第2次执行 PUSH指令后	装载数据N 第N次执行 PUSH指令后	装载数据1024 第1024次执行 PUSH指令后
S1层(栈顶)	空	1	1	1	1
S2层(栈中)	空	空	2	2	2
S3层(栈中)	空	空	空	3	3
S4层(栈中)	空	空	空	4	4
(括号中字母公式表规律) SN层(栈中) N=5至1023	空	空	空	(N) 5 . . . N	(N) 5 . . . 1023
S1024层(栈底)	空	空	空	空	1024
执行FIFO_SIZE 读取的有效层数	0	1	2	N	1024
综上描述，数据压入栈列始终是从栈底进入					

FIFO 先入先出队列	初始 状态	RLEN=1 第1次执行 POP指令后 指令读取的 DATA是1	RLEN=1 第2次执行 POP指令后 指令读取的 DATA是2	RLEN=1 第N次执行 POP指令后 指令读取的 DATA是N	RLEN=1 第1024次执行 POP指令后 指令读取的 DATA是1024
S1层(栈顶)	1	2	3	N+1	空
S2层(栈中)	2	3	4	N+2	空
S3层(栈中)	3	4	5	N+3	空
S4层(栈中)	4	5	6	N+4	空
(括号中字母公式表规律) SN层(栈中) N=5至1023	(N) 5 1023	(N+1) 6 1024	(N+2) 7 1024	(N+N) N+5 . . . 1024	空
			空	空	空
			空	空	空
S1024层(栈底)	1024	空	空	空	空
执行FIFO_SIZE 读取的有效层数	1024	1023	1022	1024-N	0
综上描述，读取栈列数据始终是从栈顶出列					

图一：先入先出 FIFO 栈列操作和后台数据存放

LIFO 后入先出队列	初始 状态	装载数据1 第1次执行 PUSH指令后	装载数据2 第2次执行 PUSH指令后	装载数据N 第N次执行 PUSH指令后	装载数据1024 第1024次执行 PUSH指令后
S1层(栈顶)	空	1	2	N	1024
S2层(栈中)	空	空	1	N-1	1023
S3层(栈中)	空	空	空	N-2	1022
S4层(栈中)	空	空	空	N-3	1021
(括号中字母公式表规律) SN层(栈中) N=5至1023	空	空	空	N-4 . . . 1	(1025-N) 1020 . . . 2
S1024层(栈底)	空	空	空	空	1
执行LIFO_SIZE 读取的有效层数	0	1	2	N	1024
综上描述，数据压入栈列始终是从栈顶进入					

LIFO 后入先出队列	初始 状态	RLEN=1 第1次执行 POP指令后 指令读取的 DATA是1024	RLEN=1 第2次执行 POP指令后 指令读取的 DATA是1023	RLEN=1 第N次执行 POP指令后 指令读取的 DATA是1025-N	RLEN=1 第1024次执行 POP指令后 指令读取的 DATA是1
S1层(栈顶)	1024	1023	1022	1024-N	空
S2层(栈中)	1023	1022	1021	1023-N	空
S3层(栈中)	1022	1021	1020	1022-N	空
S4层(栈中)	1021	1020	1019	1021-N	空
(括号中字母公式表规律) SN层(栈中) N=5至1023	(1025-N) 1020 2	(1024-N) 1019 1	(1023-N) 1018 . . . . 1 空	(1020-N) 1020-N . . . 1 空 空	空 空 空
S1024层(栈底)	1	空	空	空	空
执行LIFO_SIZE 读取的有效层数	1024	1023	1022	1024-N	0
综上描述，读取栈列数据始终是从栈顶出列					

图二：后入先出 LIFO 栈列操作和后台数据存放

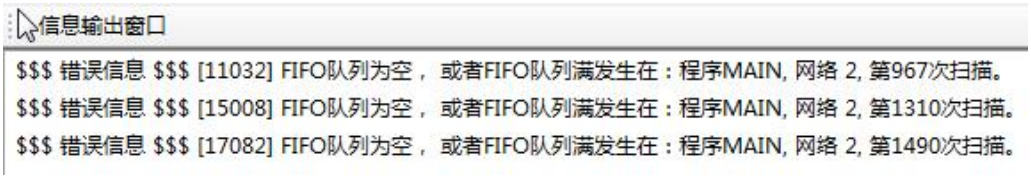
为方便使用堆栈，KincoBuilder 提供了下述堆栈相关指令，均位于指令集的【堆栈指令】组中。

名称	功能描述
FIFO_INIT	初始化(清空)先入先出栈列
FIFO_PUSH	装载数据到先入先出栈列
FIFO_POP	从先入先出栈列取出数据
FIFO_SIZE	获取先入先出栈列的有效高度（大小）
LIFO_INIT	初始化(清空)后入先出栈列
LIFO_PUSH	装载数据到后入先出栈列
LIFO_POP	从后入先出栈列取出数据
LIFO_SIZE	获取后入先出栈列的有效高度（大小）

2) 注意事项

用户在使用这些指令时，需要注意如下几点：

- 在一个用户工程中，可使用的栈列共 2 个，分别为遵循先入先出顺序的 FIFO 栈列以及遵循后入先出顺序的 LIFO 栈列。
- 两个栈列默认都支持最大 1024 个数据，也都支持装载任意 K 系列 PLC 支持的数据类型即（BOOL,BYTE,WORD,DWORD,INT,DINT,REAL），无需用特殊指令规定数据类型，系统自动兼容，即原则上用户可以装载混合数据类型的栈列，但取出数据后解析也应该用对应数据类型来解析。
- FIFO 和 LIFO 为独立两个不同栈列，他们的存放顺序不一样，所以其相关指令应配套对应使用。另外 PUSH 装载与 POP 弹出指令也应该配套使用，即栈列初始化(可用 INIT 指令)状态为空，必须使用 PUSH 指令装载数据才有意义（当前栈列的数据实际有效层数可用 SIZE 指令读出），***需要特别注意的是 0 和 0.0 也是有效数据***。
- 栈列指令单独占用 SRAM 内存区，即不占用通常所说的 V 区等基本存储区空间，***不支持掉电保存。***
- 栈列指令提供了执行错误等信息输出（如下图），但暂未提供指令执行错误输出等参数，建议用户使用前合理规划好栈列，以免出错！如栈列未执行 PUSH 指令装载压入数据就执行 POP 出栈指令自然是不对的。



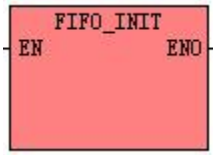
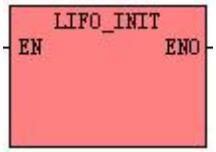
栈列占用了一个固定长度 1024 的内存块，所有的操作都必须处于此合法的内存区域内且应该首先用 PUSH 指令装载有效数据，否则堆栈指令执行错误。比如装载超过 1024 长度的数据、未装载数据就执行读取，或者读取范围超出实际装载的有效层次。弹出数据类型与装载类型不一致会造成用户意义的错误，比如把一个 INT 弹出到 BYTE，会导致高字节数值丢失。

堆栈指令执行出错后，会生成一个错误码，可用使用 KincoBuilder 读取。

错误码	描述
604	FIFO 队列为空，或者 FIFO 队列满
605	LIFO 队列为空，或者 LIFO 队列满

6.10.2 堆栈指令

6.10.2.1 FIFO_INIT 和 LIFO_INIT（初始化栈列）

	名称	指令格式	适用产品
LD	FIFO_INIT		☑ K209M ☑ KS101M ☑ K6
	LIFO_INIT		

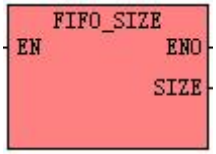
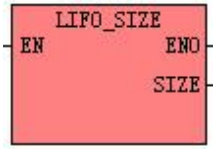
• LD 格式指令说明

如果 EN 为 1，则该指令每个扫描周期启动执行一次，指令每次执行则将 FIFO 栈列（或 LIFO 栈列）全部清空。

如果 EN 为 0，则指令不被扫描，也不会被执行。

• 指令用法，详情参见 6.10.3 堆栈使用举例。

6.10.2.2 FIFO_SIZE 和 LIFO_SIZE（获取栈列的有效装载范围）

	名称	指令格式	适用产品
LD	FIFO_SIZE		☑ K209M ☑ KS101M ☑ K6
	LIFO_SIZE		

参数	输入/输出	数据类型	数值范围	允许使用的内存区
SIZE	输出	INT	0-1024	V、M、L

各个参数的具体使用说明，见下表：

参数	功能
EN	使能端。若 *EN* 为 1 时，则该指令被使能，允许执行。
SIZE	获取栈列有效长度的存放地址

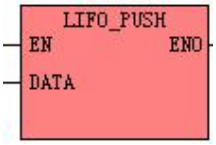
• LD 格式指令说明

如果 EN 为 1，则该指令每个扫描周期启动执行一次，指令每次执行则将 FIFO 栈列（或 LIFO 栈列）当前实际有效长度写入到 SIZE。注意是有效长度，栈列初始化时是全部为空，其有效长度为 0，每执行一次 PUSH 压栈装载数据操作，长度增加 1（最大 1024），而每执行一次 POP 出栈操作，则减去响应的本次出栈长度（最小值为 0）。

如果 EN 为 0，则指令不被扫描，也不会被执行。

• 指令用法，详情参见 6.10.3 堆栈使用举例。

6.10.2.3 FIFO_PUSH 和 LIFO_PUSH（装载数据到栈列）

	名称	指令格式	适用产品
LD	FIFO_PUSH		☑ K209M ☑ KS101M ☑ K6
	LIFO_PUSH		

参数	输入/输出	数据类型	数值范围	允许使用的内存区
DATA	输入	BOOL, BYTE WORD, DWORD INT, DINT REAL	----	I、V、M、L、常量

各个参数的具体使用说明，见下表：

参数	功能
EN	使能端。若 *EN* 为 1 时，则该指令被使能，允许执行。
DATA	存入栈列的数据地址

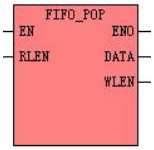
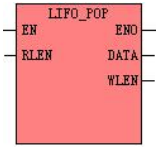
- LD 格式指令说明

如果 EN 为 1，则该指令每个扫描周期启动执行一次，指令每次执行则 DATA 的数据存放到将 FIFO 栈列（或 LIFO 栈列）中，同时系统后台自动给栈列长度增加 1，且根据先入先出（或后入先出）的规则决定存放的位置。详见 6.10.1　堆栈指令综述关于栈列的详细介绍。

如果 EN 为 0，则指令不被扫描，也不会被执行。

- 指令用法，详情参见 6.10.3　堆栈使用举例。

6.10.2.4 FIFO_POP 和 LIFO_POP（从数据栈列中取出数据）

	名称	指令格式	适用产品
LD	FIFO_POP		☑ K209M ☑ KS101M ☑ K6
	LIFO_POP		

参数	输入/输出	数据类型	数值范围	允许使用的内存区
RLEN	输入	INT	1-1024	I、V、M、L、常量
WLEN	输出	INT	0-1024	V、M、L
DATA	输出	BOOL, BYTE WORD, DWORD INT, DINT REAL	----	V、M、L

各个参数的具体使用说明，见下表：

参数	功能
EN	使能端。若 *EN* 为 1 时，则该指令被使能，允许执行。
RLEN	希望读取栈列数据的长度
WLEN	读取成功的数据实际长度
DATA	读取栈列中的数据存放起始地址

- LD 格式指令说明

如果 EN 为 1，则该指令每个扫描周期启动执行一次，指令每次执行从 FIFO 栈列（或 LIFO 栈

列）中的**栈顶**起始连续 RLEN 个**有效数据**复制传送到地址 DATA 开始的连续 WLEN 个变量，传送的数据类型保持原有不变，传送的最大长度不应该超过最大有效元素范围 1024，也不应该超过由 FIFO_SIZE LIFO_SIZE 指令获取到的当前栈列有效数据范围。执行完出栈操作后系统后台自动给栈列长度减去 WLEN，且自动把栈列内剩余数据全部依次向栈顶上升 WLEN 位以便保证栈内数据出入的统一规则（先入先出或后入先出）。详见 6.10.1 堆栈指令综述关于栈列的详细介绍。

注意，这里操作的是有效数据，假如栈内当前有效数据长度小于 RLEN 长度，则只能取出实际剩余的有效数据，即实际的 WLEN 读成功的数量会小于 RLEN 设定读取的数量。

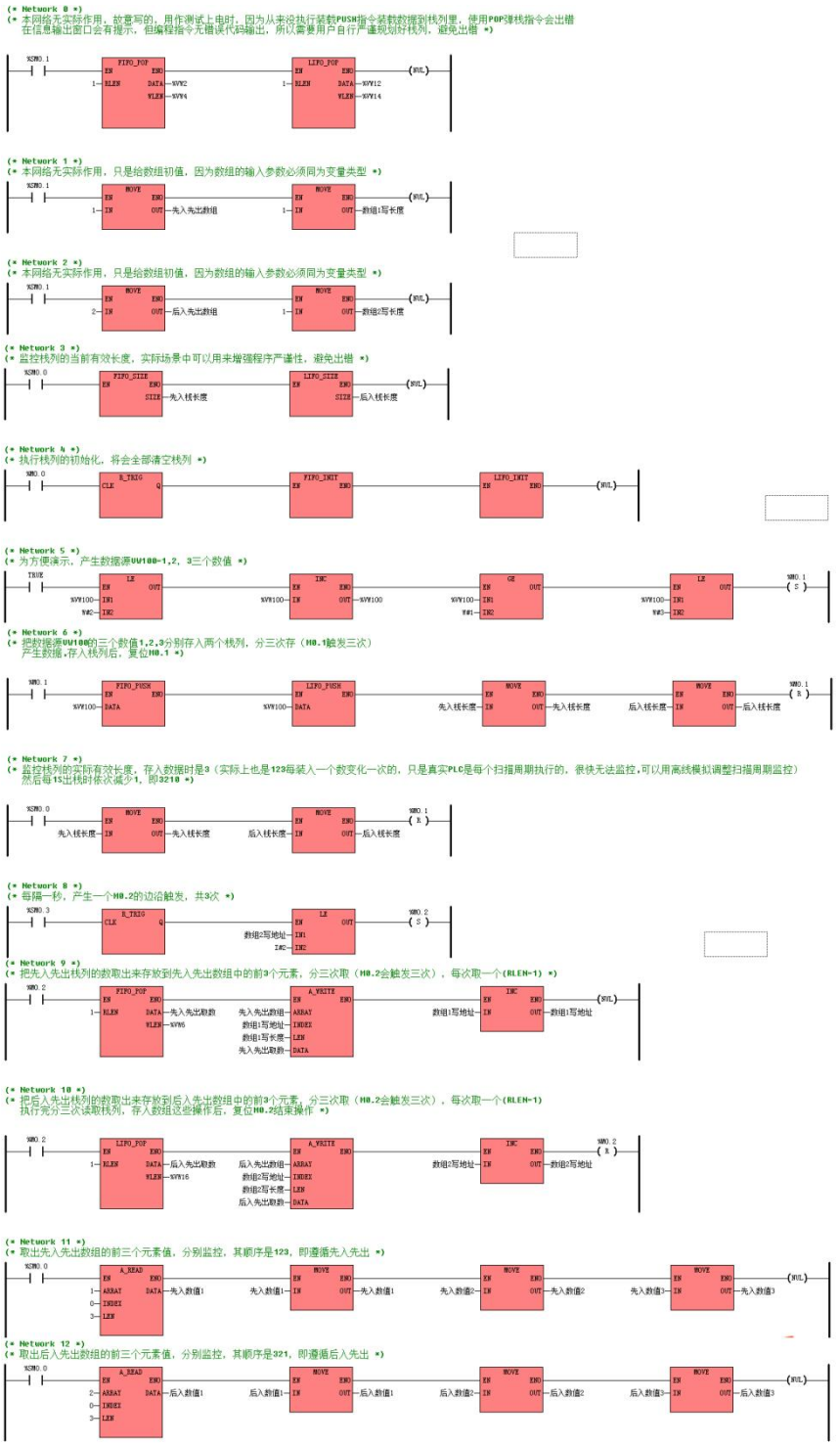
如果 EN 为 0，则指令不被扫描，也不会被执行。

- 指令用法，详情参见 6.10.3 堆栈使用举例。

6.10.3 堆栈使用举例

6.10.3.1 各指令基础说明演示

（一）：LD 格式（梯形图）程序（程序例子可在 www.kinco.cn 网站下载）


(* Network 0 *)
(* 本网络无实际作用，故意写的，用作测试上电时，因为从未没执行装载PUSH指令装载数据到栈列里，使用POP弹栈指令会出错
在信息输出窗口会有提示，但编程指令无错误代码输出，所以需要用户自行严谨规划好栈列，避免出错 *)
%M0.1
FIFO_POP
LIFO_POP
EN
ENO
BLEN
DATA
WLEN
%W2
%W4
%W12
%W14
NUL
(* Network 1 *)
(* 本网络无实际作用，只是给数组初值，因为数组的输入参数必须同为变量类型 *)
%M0.1
MOVE
IN
OUT
先入先出数组
数组1写长度
(* Network 2 *)
(* 本网络无实际作用，只是给数组初值，因为数组的输入参数必须同为变量类型 *)
%M0.1
后入先出数组
数组2写长度
(* Network 3 *)
(* 监控栈列的当前有效长度，实际场景中可以用来增强程序严谨性，避免出错 *)
%M0.0
FIFO_SIZE
LIFO_SIZE
SIZE
先入栈长度
后入栈长度
(* Network 4 *)
(* 执行栈列的初始化，将会全部清空栈列 *)
%M0.0
R_TRIG
CLK
Q
FIFO_INIT
LIFO_INIT
(* Network 5 *)
(* 为方便演示，产生数据源VW100-1,2, 3三个数值 *)
TRUE
LE
INC
GE
%W100
%W2
%W1
%M0.1
S
(* Network 6 *)
(* 把数据源VW100的三个数值1,2,3分别存入两个栈列，分三次存（M0.1触发三次）
产生数据,存入栈列后，复位M0.1 *)
%M0.1
FIFO_PUSH
LIFO_PUSH
先入栈长度
后入栈长度
R
(* Network 7 *)
(* 监控栈列的实际有效长度，存入数据时是3（实际上也是123每装入一个数变化一次的，只是真实PLC是每个扫描周期执行的，很快无法监控,可以用离线模拟调整扫描周期监控）
然后每1S出栈时依次减少1，即3210 *)
(* Network 8 *)
(* 每隔一秒，产生一个M0.2的边沿触发，共3次 *)
%M0.3
%M0.2
(* Network 9 *)
(* 把先入先出栈列的数取出来存放到先入先出数组中的前3个元素，分三次取（M0.2会触发三次），每次取一个(RLEN-1) *)
%M0.2
FIFO_POP
A_WRITE
先入先出取数
先入先出数组
数组1写地址
数组1写长度
先入先出取数
ARRAY
INDEX
LEN
DATA
(* Network 10 *)
(* 把后入先出栈列的数取出来存放到后入先出数组中的前3个元素，分三次取（M0.2会触发三次），每次取一个(RLEN-1)
执行完分三次读取栈列，存入数组这些操作后，复位M0.2结束操作 *)
%M0.2
LIFO_POP
A_WRITE
后入先出取数
后入先出数组
数组2写地址
数组2写长度
后入先出取数
(* Network 11 *)
(* 取出先入先出数组的前三个元素值，分别监控，其顺序是123，即遵循先入先出 *)
%M0.0
A_READ
先入数值1
先入数值2
先入数值3
(* Network 12 *)
(* 取出后入先出数组的前三个元素值，分别监控，其顺序是321，即遵循后入先出 *)
%M0.0
后入数值1
后入数值2
后入数值3

IL 格式（指令）程序 可以直接粘贴到软件中转换成梯形图。

```
(* Network 0 *)
(*本网络无实际作用，故意写的，用作测试上电时，因为从来没执行装载 PUSH 指令装载数据到栈列里，使用 POP 弹栈指令会出错在信息输出窗口会有提示，但编程指令无错误代码输出，所以需要用户自行严谨规划好栈列，避免出错*)
LD      %SM0.1
FIFO_POP 1, %VW2, %VW4
LIFO_POP 1, %VW12, %VW14
(* Network 1 *)
(*本网络无实际作用，只是给数组初值，因为数组的输入参数必须同为变量类型*)
LD      %SM0.1
MOVE    1, %VW40
MOVE    1, %VW42
(* Network 2 *)
(*本网络无实际作用，只是给数组初值，因为数组的输入参数必须同为变量类型*)
LD      %SM0.1
MOVE    2, %VW50
MOVE    1, %VW52
(* Network 3 *)
(*监控栈列的当前有效长度，实际场景中可以用来增强程序严谨性，避免出错*)
LD      %SM0.0
FIFO_SIZE %VW0
LIFO_SIZE %VW10
(* Network 4 *)
(*执行栈列的初始化，将会全部清空栈列*)
LD      %M0.0
R_TRIG
```

```
FIFO_INIT
LIFO_INIT
(* Network 5 *)
(*为方便演示，产生数据源 VW100=1,2，3 三个数值*)
LD      TRUE
LE      %VW100, W#2
INC     %VW100
GE      %VW100, W#1
LE      %VW100, W#3
S       %M0.1
(* Network 6 *)
(*把数据源 VW100 的三个数值 1,2,3 分别存入两个栈列，分三次存（M0.1 触发三次）产生数据，
存入栈列后，复位 M0.1*)
LD      %M0.1
FIFO_PUSH %VW100
LIFO_PUSH %VW100
MOVE    %VW0, %VW0
MOVE    %VW10, %VW10
R       %M0.1
(* Network 7 *)
(*监控栈列的实际有效长度，存入数据时是 3（实际上也是 123 每装入一个数变化一次的，只是
真实 PLC 是每个扫描周期执行的，很快无法监控,可以用离线模拟调整扫描周期监控）然后每 1S
出栈时依次减少 1，即 3210*)
LD      %SM0.0
MOVE    %VW0, %VW0
MOVE    %VW10, %VW10
R       %M0.1
```

```
(* Network 8 *)
(*每隔一秒，产生一个 M0.2 的边沿触发，共 3 次*)
LD      %SM0.3
R_TRIG
LE      %VW18, I#2
S       %M0.2
(* Network 9 *)
(*把先入先出栈列的数取出来存放到先入先出数组中的前 3 个元素,分三次取(M0.2 会触发三次),
每次取一个(RLEN=1)*)
LD      %M0.2
FIFO_POP 1, %VW20, %VW6
A_WRITE %VW40, %VW8, %VW42, %VW20
INC     %VW8
(* Network 10 *)
(*把后入先出栈列的数取出来存放到后入先出数组中的前 3 个元素,分三次取(M0.2 会触发三次),
每次取一个(RLEN=1)   执行完分三次读取栈列，存入数组这些操作后，复位 M0.2 结束操作*)
LD      %M0.2
LIFO_POP 1, %VW30, %VW16
A_WRITE %VW50, %VW18, %VW52, %VW30
INC     %VW18
R       %M0.2
(* Network 11 *)
(*取出先入先出数组的前三个元素值，分别监控，其顺序是 123，即遵循先入先出*)
LD      %SM0.0
A_READ  1, 0, 3, %VW22
MOVE    %VW22, %VW22
MOVE    %VW24, %VW24
```

```
MOVE    %VW26, %VW26
(* Network 12 *)
(*取出后入先出数组的前三个元素值，分别监控，其顺序是 321，即遵循后入先出*)
LD      %SM0.0
A_READ  2, 0, 3, %VW32
MOVE    %VW32, %VW32
MOVE    %VW34, %VW34
MOVE    %VW36, %VW36
```

6.10.3.2 堆栈应用演示一：电话排列

堆栈也是对一组连续数组的储存，但区别于数组，它可以指定顺序，即有了排队的概念，所以在实际编程中常用来处理排队的时间，如客服电话接入，银行叫号排队，电梯调度，AGV 调度等都是以所有总需求为一个队列，然后遵循先入先出的顺序，所以此时堆栈指令的作用得以体现。

本演示例子仅供纯粹演示堆栈结合循环这一类的最常使用方式，实际应用场景还有很多其他因素需要考虑，本例仅供参考而不能做实际应用。

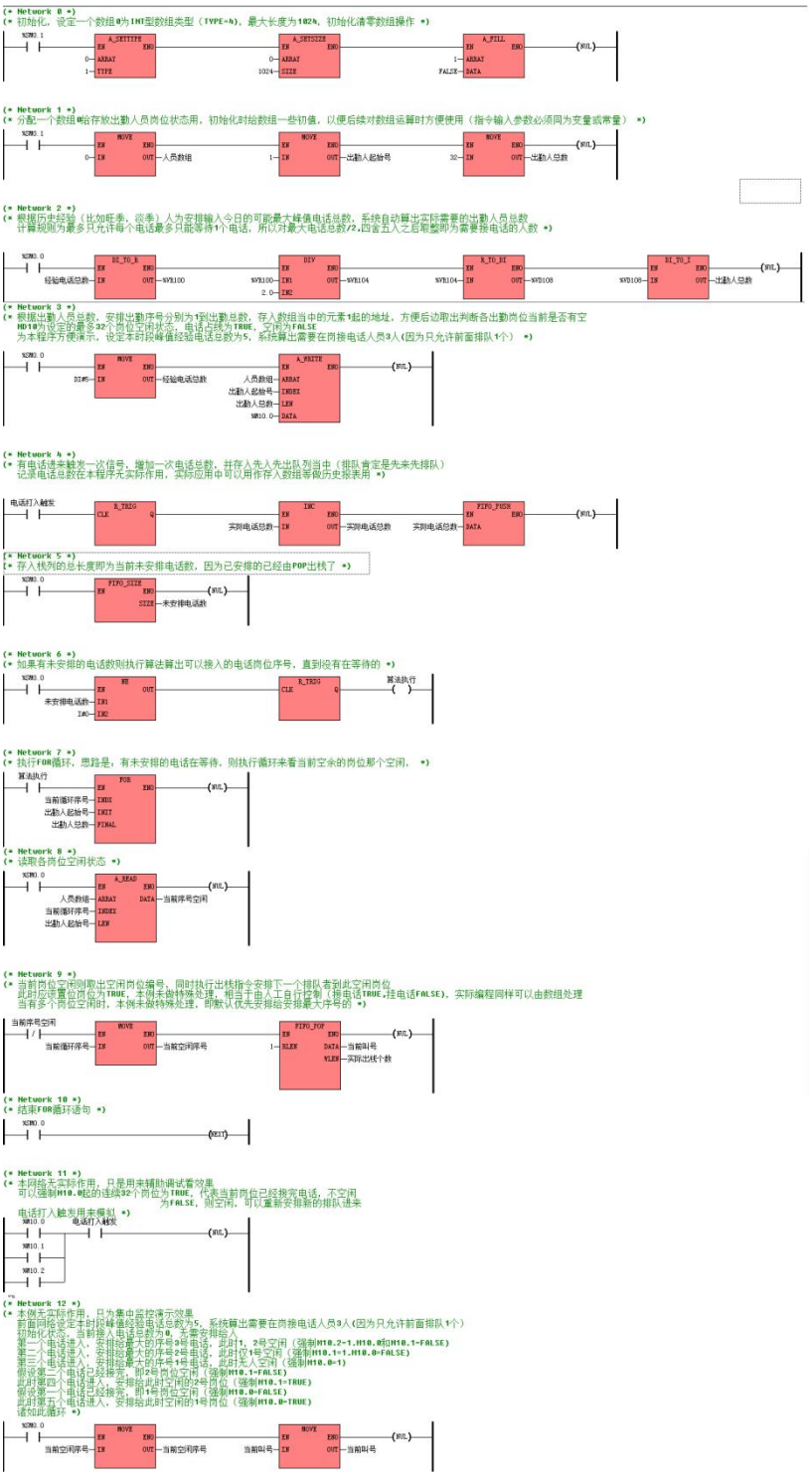
本例需求：现有一客服电话中心，负责接听电话，共有 32 部电话由 32 个接线员负责，设计程序完成以下需求：

- 根据历史经验得出的每日最大可能峰值电话数，以及最多只允许排队一个电话，自动计算得出需要安排的实际接线电话数。
- 电话进来后，按先入先接排队顺序安排给各空闲电话，即每个电话接入后安排给对应的空闲岗位号。

程序思路：

- 分配一个数组 0，数据类型本例是 BOOL 类型，数组长度为 32，即数组 0 用来存放当前岗位是否空闲的状态。
- 编程思路为：把接入的电话按序号用 PUSH 指令存入 FIFO 先入先出队列中，每个扫描周期都判断栈列长度是否为 0，不为 0 则代表有等待的电话，从而利用 FOR 循环依次取出来岗位状态数组中每一个元素（即岗位状态）跟条件进行比对（是否空闲），符合空闲的则可以安排，同时用 POP 指令从栈列中出掉当前已经处理完的电话序号，周而复始。

（一）：LD 格式（梯形图）程序（程序例子可在 www.kinco.cn 网站下载）


(* Network 0 *)
(* 初始化，设定一个数组0为INT型数组类型（TYPE=4），最大长度为1024，初始化清零数组操作 *)
A_SETTYPE
A_SETSIZE
A_FILL
ARRAY
TYPE
SIZE
DATA
(* Network 1 *)
(* 分配一个数组4给存放出勤人员岗位状态用，初始化时给数组一些初值，以便后续对数组运算时方便使用（指令输入参数必须同为变量或常量） *)
MOVE
人员数组
出勤人起始号
出勤人总数
(* Network 2 *)
(* 根据历史经验（比如旺季，淡季）人为安排输入今日的可能最大峰值电话总数，系统自动算出实际需要的出勤人员总数
计算规则为最多只允许每个电话最多只能等待1个电话，所以对最大电话总数/2,四舍五入之后取整即为需要接电话的人数 *)
R_TO_R
DIV
R_TO_DI
DI_TO_I
经验电话总数
(* Network 3 *)
(* 根据出勤人员总数，安排出勤序号分别为1到出勤总数，存入数组当中的元素1起的地址，方便后边取出判断各出勤岗位当前是否有空
MD10为设定的最多32个岗位空闲状态，电话占线为TRUE，空闲为FALSE
为本程序方便演示，设定本时段峰值经验电话总数为5，系统算出需要在岗接电话人员3人(因为只允许前面排队1个） *)
A_WRITE
(* Network 4 *)
(* 有电话进来触发一次信号，增加一次电话总数，并存入先入先出队列当中（排队肯定是先来先排队）
记录电话总数在本程序无实际作用，实际应用中可以用作存入数组等做历史报表用 *)
电话打入触发
R_TRIG
INC
FIFO_PUSH
实时电话总数
(* Network 5 *)
(* 存入栈列的总长度即为当前未安排电话数，因为已安排的已经由POP出栈了 *)
FIFO_SIZE
未安排电话数
(* Network 6 *)
(* 如果有未安排的电话数则执行算法算出可以接入的电话岗位序号，直到没有在等待的 *)
NE
算法执行
(* Network 7 *)
(* 执行FOR循环，思路是，有未安排的电话在等待，则执行循环来看当前空余的岗位那个空闲， *)
FOR
当前循环序号
出勤人起始号
出勤人总数
(* Network 8 *)
(* 读取各岗位空闲状态 *)
A_READ
当前循环序号
(* Network 9 *)
(* 当前岗位空闲则取出空闲岗位编号，同时执行出栈指令安排下一个排队者到此空闲岗位
此时应设置岗位为TRUE，本例未做特殊处理，相当于由人工自行控制（接电话TRUE，挂电话FALSE），实际编程同样可以由数组处理
当有多个岗位空闲时，本例未做特殊处理，即默认优先安排给号最大序号的 *)
当前序号空闲
FIFO_POP
当前循环序号
当前叫号
实际出栈个数
(* Network 10 *)
(* 结束FOR循环语句 *)
(* Network 11 *)
(* 本网络无实际作用，只是用来辅助调试看效果
可以强制M10.0起的连续32个岗位为TRUE，代表当前岗位已经接完电话，不空闲
为FALSE，则空闲，可以重新安排新的排队进来
电话打入触发用来模拟 *)
电话打入触发
(* Network 12 *)
(* 本例无实际作用，只为集中监控演示效果
前面网络设定本时段峰值经验电话总数为5，系统算出需要在岗接电话人员3人(因为只允许前面排队1个）
初始化状态：当前接入电话总数为0，无需安排岗人
第一个电话进入，安排给最大的序号8号电话，此时1，2号空闲（强制M10.2-1,M10.0和M10.1-FALSE)
第二个电话进入，安排给最大的序号2号电话，此时仅1号空闲（强制M10.1-1,M10.0-FALSE)
第三个电话进入，安排给最大的序号1号电话，此时无人空闲（强制M10.0-1)
假设第二个电话已经挂完，则2号岗位空闲（强制M10.1-FALSE)
此时第四个电话进入，安排给此时空闲的2号岗位（强制M10.1-TRUE)
假设第一个电话已经挂完，即1号岗位空闲（强制M10.0-FALSE)
此时第五个电话进入，安排给此时空闲的1号岗位（强制M10.0-TRUE)
诸如此循环 *)
当前空闲序号
当前叫号

IL 格式（指令）程序 可以直接粘贴到软件中转换成梯形图。

```
(* Network 0 *)
(*初始化，设定一个数组 0 为 INT 型数组类型（TYPE=4），最大长度为 1024，初始化清零数组操作*)
LD      %SM0.1
A_SETTYPE 0, 1
A_SETSIZE 0, 1024
A_FILL  1, FALSE
(* Network 1 *)
(*分配一个数组 0 给存放出勤人员岗位状态用，初始化时给数组一些初值，以便后续对数组运算时方便使用（指令输入参数必须同为变量或常量）*)
LD      %SM0.1
MOVE    0, %VW0
MOVE    1, %VW2
MOVE    1024, %VW4
(* Network 2 *)
(*根据历史经验（比如旺季，淡季）人为安排输入今日的可能最大峰值电话总数，系统自动算出实际需要的出勤人员总数   计算规则为最多只允许每个电话最多只能等待 1 个电话,所以对最大电话总数/2,四舍五入之后取整即为需要接电话的人数*)
LD      %SM0.0
DI_TO_R %VD8, %VR100
MOVE    %VR100, %VR104
DIV     2.0, %VR104
R_TO_DI %VR104, %VD108
DI_TO_I %VD108, %VW4
(* Network 3 *)
(*根据出勤人员总数，安排出勤序号分别为 1 到出勤总数，存入数组当中的元素 1 起的地址，方
```

```
便后边取出判断各出勤岗位当前是否有空   MD10 为设定的最多 32 个岗位空闲状态，电话占线为
TRUE，空闲为 FALSE   为本程序方便演示，设定本时段峰值经验电话总数为 5，系统算出需要在
岗接电话人员 3 人(因为只允许前面排队 1 个）*)
LD      %SM0.0
MOVE    DI#5, %VD8
A_WRITE %VW0, %VW2, %VW4, %M10.0
(* Network 4 *)
(*有电话进来触发一次信号，增加一次电话总数，并存入先入先出队列当中（排队肯定是先来先
排队）   记录电话总数在本程序无实际作用，实际应用中可以用作存入数组等做历史报表用*)
LD      %M0.0
R_TRIG
INC     %VW6
FIFO_PUSH %VW6
(* Network 5 *)
(*存入栈列的总长度即为当前未安排电话数，因为已安排的已经由 POP 出栈了*)
LD      %SM0.0
FIFO_SIZE %VW12
(* Network 6 *)
(*如果有未安排的电话数则执行算法算出可以接入的电话岗位序号，直到没有在等待的*)
LD      %SM0.0
NE      %VW12, I#0
R_TRIG
ST      %M0.1
(* Network 7 *)
(*执行 FOR 循环，思路是：有未安排的电话在等待，则执行循环来看当前空余的岗位那个空闲，
*)
LD      %M0.1
```

```
FOR     %VW14, %VW2, %VW4
(* Network 8 *)
(*读取各岗位空闲状态*)
LD      %SM0.0
A_READ  %VW0, %VW14, %VW2, %M20.0
(* Network 9 *)
(*当前岗位空闲则取出空闲岗位编号，同时执行出栈指令安排下一个排队者到此空闲岗位，此时应该置位岗位为 TRUE，本例未做特殊处理，相当于由人工自行控制（接电话 TRUE,挂电话 FALSE），实际编程同样可以由数组处理当有多个岗位空闲时，本例未做特殊处理，即默认优先安排给安排最大序号的*)
LDN     %M20.0
MOVE    %VW14, %VW16
FIFO_POP 1, %VW18, %VW20
(* Network 10 *)
(*结束 FOR 循环语句*)
LD      %SM0.0
NEXT
(* Network 11 *)
(*本网络无实际作用，只是用来辅助调试看效果可以强制 M10.0 起的连续 32 个岗位为 TRUE，代表当前岗位已经接完电话，不空闲为 FALSE，则空闲，可以重新安排新的排队进来电话打入触发用来模拟*)
LD      %M10.0
OR      %M10.1
OR      %M10.2
AND     %M0.0
ST      %BR0.0
(* Network 12 *)
```

(*本例无实际作用，只为集中监控演示效果前面网络设定本时段峰值经验电话总数为 5，系统算出需要在岗接电话人员 3 人(因为只允许前面排队 1 个）初始化状态，当前接入电话总数为 0，无需安排给人第一个电话进入，安排给最大的序号 3 号电话，此时 1，2 号空闲（强制 M10.2=1.M10.0 和 M10.1=FALSE)第二个电话进入，安排给最大的序号 2 号电话，此时仅 1 号空闲（强制 M10.1=1.M10.0=FALSE)第三个电话进入，安排给最大的序号 1 号电话，此时无人空闲(强制 M10.0=1)假设第二个电话已经接完，即 2 号岗位空闲（强制 M10.1=FALSE)此时第四个电话进入，安排给此时空闲的 2 号岗位（强制 M10.1=TRUE)假设第一个电话已经接完，即 1 号岗位空闲（强制 M10.0=FALSE)此时第五个电话进入，安排给此时空闲的 1 号岗位（强制 M10.0=TRUE)诸如此循环 *)

```
LD      %SM0.0
MOVE    %VW16, %VW16
MOVE    %VW18, %VW18
```

6.10.3.3 堆栈应用演示二：逆序输出

堆栈常用的场景是逆序输出（注意与数字的排序指令区分，数组的排序指令是指按数值大小排序，队列的逆序则是颠倒顺序，与具体数值无关，比如输入一串字符原有顺序为 ABCD，逆序则为 DCBA）。

实际应用中，经常碰到逻辑电路逆序输出，比如启动电机输出顺序为正，停止时要逆序输出。如果是紧挨着输出，比如 Q0.0-Q0.1-Q0.2-Q0.7 这样的标准紧挨着顺序还可以使用移位指令处理，但如果是无规则的比如 Q0.0-Q0.2-Q0.1 这样的顺序，则使用堆栈 LIFO 指令处理更为合适。下面例子演示了这一处理，实际上在通信当中，对收到的结果进行逆序处理也极为常见。

本演示例子仅供纯粹演示堆栈逆序输出的最常使用方式，实际应用场景还有很多其他因素需要考虑，本例仅供参考而不能做实际应用。

本例需求：启动后，按 Q0.0-Q0.2-Q0.1 顺序输出，停止时，按 Q0.1-Q0.2-Q0.0 逆序输出

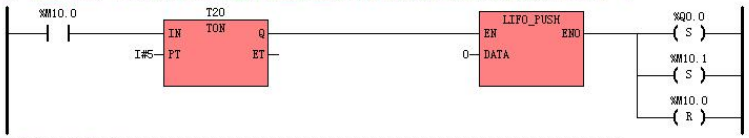
程序思路：

- 分配一个数组 0，数据类型本例是 BOOL 类型，数组长度为 8，即数组 0 用来存放当前输出点的状态。
- 编程思路为：每次输出时，把当前输出点的在整个输出队列中的顺序号用 PUSH 指令存入 LIFO 后入先出队列中，输出点总的状态按正常紧挨着顺序存放到数组中。需要逆序输出时，先执行 POP 出栈指令，得出本次出栈的输出点的顺序号，然后写给数组复位对应顺序号，但需要注意还要执行一次读出数组全部元素的结果给到输出点内存中才能复位（之前执行的只是把数组中的结果复位了），详见程序注释。

（一）：LD 格式（梯形图）程序（程序例子可在 www.kinco.cn 网站下载）

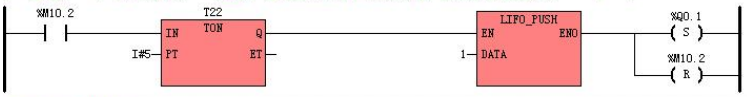
(* Network 0 *)
(* 本网络开始模拟产生按Q0.0.Q0.2.Q0.1相隔500毫秒的输出 *)

(* Network 1 *)
(* 输出Q0.0，同时存入Q0.0在整个输出数组的中的顺序位0到后入先出队列LIFO中 *)

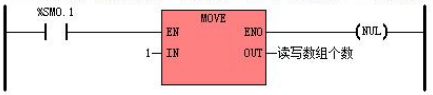


(* Network 2 *)
(* 输出Q0.2，同时存入Q0.2在整个输出数组的中的顺序位2到后入先出队列LIFO中 *)

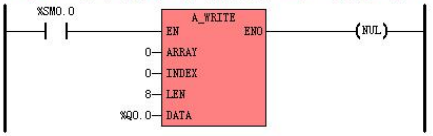
(* Network 3 *)
(* 输出Q0.1，同时存入Q0.1在整个输出数组的中的顺序位1到后入先出队列LIFO中 *)



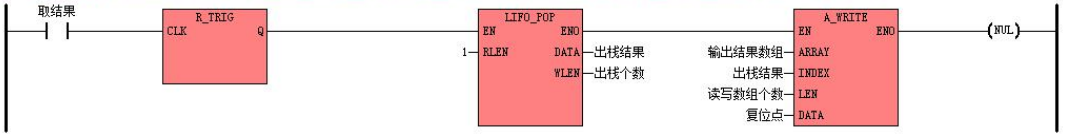
(* Network 4 *)
(* 本网络无实际作用，只给数组读写个数初值1，因为数组读写指令输入参数必须同时为变量 *)



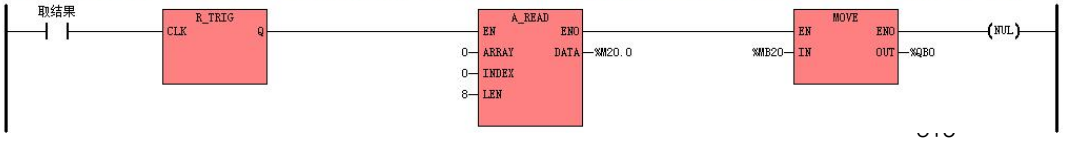
(* Network 5 *)
(* 把Q0.0开始的输出8个值存放到数组中，好方便运算 *)



(* Network 6 *)
(* 取结果时，先从LIFO中按后入先出的逆序取出存放的输出点顺序后，然后写零值给数组中对应的那个顺序点复位 *)



(* Network 7 *)
(* 因为上网络写入数组指令的值还只是数组内存区的值发生了改变，所以需要执行读出来一次全部数组内存值，赋值给真正的Q点输出 *)



IL 格式（指令）程序 可以直接粘贴到软件中转换成梯形图。

```
(* Network 0 *)
(*本网络开始模拟产生按 Q0.0,Q0.2,Q0.1 相隔 500 毫秒的输出*)
LD      %SM0.1
S       %M10.0
(* Network 1 *)
(*输出 Q0.0，同时存入 Q0.0 在整个输出数组的中的顺序位 0 到后入先出队列 LIFO 中*)
LD      %M10.0
TON     T20, I#5
LIFO_PUSH 0
S       %Q0.0
S       %M10.1
R       %M10.0
(* Network 2 *)
(*输出 Q0.2，同时存入 Q0.2 在整个输出数组的中的顺序位 2 到后入先出队列 LIFO 中*)
LD      %M10.1
TON     T21, I#5
LIFO_PUSH 2
S       %Q0.2
S       %M10.2
R       %M10.1
(* Network 3 *)
(*输出 Q0.1，同时存入 Q0.1 在整个输出数组的中的顺序位 1 到后入先出队列 LIFO 中*)
LD      %M10.2
TON     T22, I#5
LIFO_PUSH 1
S       %Q0.1
```

```
R       %M10.2
(* Network 4 *)
(*本网络无实际作用，只给数组读写个数初值 1，因为数组读写指令输入参数必须同时为变量*)
LD      %SM0.1
MOVE    1, %VW8
(* Network 5 *)
(*把 Q0.0 开始的输出 8 个值存放到数组中，好方便运算*)
LD      %SM0.0
A_WRITE 0, 0, 8, %Q0.0
(* Network 6 *)
(*取结果时，先从 LIFO 中按后入先出的逆序取出存放的输出点顺序后，然后写零值给数组中对应
的那个顺序点复位*)
LD      %M0.1
R_TRIG
LIFO_POP 1, %VW12, %VW10
A_WRITE %VW6, %VW12, %VW8, %M0.3
(* Network 7 *)
(*因为上网络写入数组指令的值还只是数组内存区的值发生了改变，所以需要执行读出来一次全
部数组内存值，赋值给真正的 Q 点输出*)
LD      %M0.1
R_TRIG
A_READ  0, 0, 8, %M20.0
MOVE    %MB20, %QB0
```

6.11 程序控制

6.11.1 标号及跳转指令

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	LBL	lbl (LBL)		☑ K5 ☑ K2 ☑ KS ☑ KW ☑ HP ☑ MK ☑ K6
	JMP	lbl (JMP)		
	JMPC	lbl (JMPC)		
	JMPCN	lbl (MPCN)		
IL	标号	lbl:	U	
	JMP	JMP *lbl*		
	JMPC	JMPC *lbl*		
	JMPCN	JMPCN *lbl*		

参数	描 述
lbl	合法的标识符

注意，跳转指令中指定的 lbl 标号必须存在而且必须与该指令在同一程序中。

注意，此指令耗时可能较长， 会触发看门狗， 可以加入 WDR 指令来延长看门狗的触发时间，且，注意 JMP 的条件不要造成无法跳出的死循环。

- LD

  LBL 指令用于在当前位置定义一个标号，该标号将作为跳转指令的目的地。标号不允许重复

定义。LBL 指令是无条件执行的，不建议用户在 LBL 指令的左端加入任何元件。实际上，在编译时编译器会将 *LBL* 指令的左端的所有元件忽略掉。

JMP 指令用于无条件地将程序跳转到 *lb1* 标号处继续执行。

JMPC 指令的作用是：若指令左侧能量流的值为 1，则将程序跳转到 *lb1* 标号处继续执行，否则该指令不起作用，程序继续向下依次执行。

JMPCN 指令的作用是：若指令左侧能量流的值为 0，则将程序跳转到 *lb1* 标号处继续执行，否则该指令不起作用，程序继续向下依次执行。

- IL

标号的定义格式是：*lb1*：。标号的定义占用独立的一行。标号不允许重复定义。

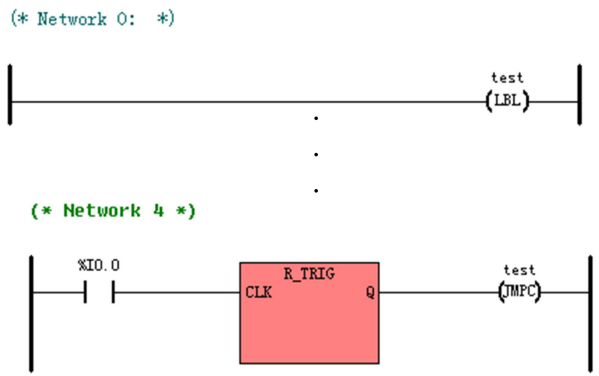
JMP 指令用于无条件地将程序跳转到 *lb1* 标号处继续执行。

JMPC 指令的作用是：若 CR 值为 1，则将程序跳转到 *lb1* 标号处继续执行，否则该指令不起作用，程序继续向下依次执行。

JMPCN 指令的作用是：若 CR 值为 0，则将程序跳转到 *lb1* 标号处继续执行，否则该指令不起作用，程序继续向下依次执行。

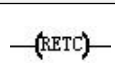
注意：跳转指令的执行不影响 CR 值，因此必要时用户应该在跳转的目的标号之后重新建立新的 CR 值，以免程序执行出现错误。

➢ 指令使用举例

LD	IL
	(* NETWORK 0 *) test: ... (* NETWORK 4 *) LD %I0.0 R_TRIG JMPC test

6.11.2 返回指令

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ HP ☑ MK ☑ K6
LD	RETC			
	RETCN			
IL	RETC	RETC	U	
	RETCN	RETCN		

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

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

- LD

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

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

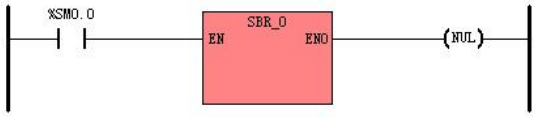
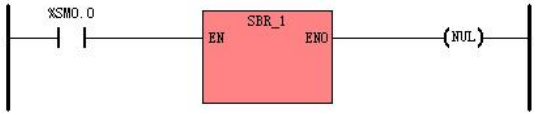
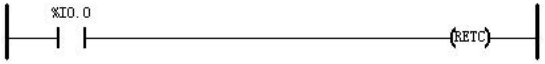
- IL

RETC 指令：若 CR 值为 1，则该指令被执行，否则该指令不起作用，程序继续向下依次执行。

RETCN 指令：若 CR 值为 0，则该指令执行，否则该指令不起作用，程序继续向下依次执行。

注意：另外，返回指令的执行不影响 CR 值，因此必要时用户应该在程序的返回点之后重新建立新的 CR 值，以免程序执行出现错误。

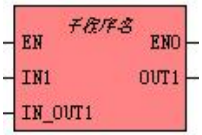
➢ 指令使用举例

<table>
<tr>
<td>LD</td>
<td>主程序：
(* Network 0 *)
 
(* Network 1 *)
 
子程序 SBR_0：
(* Network 0 *)
 
(* Network 1 *)
  </td>
<td>在子程序 SBR_0 中：
若 I0.0 为 0，则继续依次执行下面的指令。
若 I0.0 为 1，则返回到主程序中 SBR_0 的调用点处继续向下执行 Network 1 中的指令。</td>
</tr>
<tr>
<td>IL</td>
<td colspan="2">主程序：
LD %SM0.0 (* 建立 CR，其值恒为 I *)
CAL SBR_0 (* 调用子程序 SBR_0 *)
CAL SBR_1 (* 调用子程序 SBR_1 *)

子程序 SBR_0：
LD %I0.0 (* 建立 CR，其值为 I0.0 的值 *)
RETC (* 若 CR 为 1：则返回到主程序中并继续执行 CAL SBR_1 指令 *)
LD %I0.1 (* 若 RETC 指令没起作用，则该指令及后续指令得以执行 *)
ANDN %I0.2
ST %Q0.0</td>
</tr>
</table>

6.11.3 CAL（调用子程序）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ HP ☑ MK ☑ K6
LD	CAL			
IL	CAL	CAL 子程序名，子程序实参 1，子程序实参 2，…	U	

该指令用于调用并执行指定名称的子程序，子程序执行完成后，将返回至 CAL 之后的指令继续执行。CAL 指令调用的子程序在用户工程中必须已经存在。

在 CAL 指令中输入的实参必须与被调用子程序的形参（在该子程序的局部变量表中定义）的数据类型和变量类型相匹配。用户必须依照下列次序来输入子程序的实参：所有的输入参数、所有的输入/输出参数、所有的输出参数。

- LD

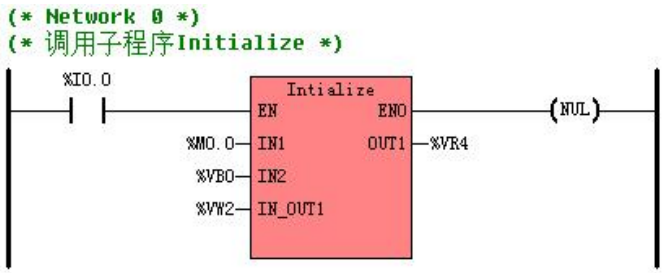
若用户在工程中编写了一个子程序，则该子程序的名字将出现在指令树的【子程序】组中，双击这个名字就可以在程序中相应的位置加入该子程序的调用指令。若调用指令左侧的能量流的值为 1，则该子程序就被调用并执行。

- IL

若 CR 值为 1，则调用并执行指定的子程序，否则该指令不起作用，程序继续向下依次执行。

CAL 指令的执行不影响 CR 值，但是需要注意的是：CR 值可能在子程序中发生了变化。

➢ 指令使用举例

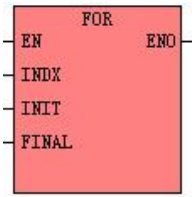
<table>
<tr>
<td>LD</td>
<td>主程序：


子程序 Initialize 中的局部变量表：
<table>
<tr><th></th><th>变量地址</th><th>变量名称</th><th>变量类型</th><th>数据类型</th></tr>
<tr><td>▶</td><td>%L0.0</td><td>IN1</td><td>VAR</td><td>BOOL</td></tr>
<tr><td></td><td>%LB16</td><td>IN2</td><td>VAR</td><td>BYTE</td></tr>
<tr><td></td><td>%LW22</td><td>IN_OUT1</td><td>VAR</td><td>INT</td></tr>
<tr><td></td><td>%LD18</td><td>OUT1</td><td>VAR</td><td>REAL</td></tr>
</table>
</td>
<td>在主程序中：
若 I0.0 为 0，则继续依次执行下面的指令。
若 I0.0 为 1，则调用并执行子程序 Initialize。</td>
</tr>
<tr>
<td>IL</td>
<td colspan="2">
主程序：
(* Network 0 *)
LD %I0.0(* 建立 CR，其值为 I0.0 的值 *)
CAL Intialize %M0.0, %VB0, %VW2, %VR10 (* 若 CR 为 1，则调用并执行 Intialize *)

子程序 Initialize 中的局部变量表：
<table>
<tr><th></th><th>变量地址</th><th>变量名称</th><th>变量类型</th><th>数据类型</th></tr>
<tr><td>▶</td><td>%L0.0</td><td>IN1</td><td>VAR</td><td>BOOL</td></tr>
<tr><td></td><td>%LB16</td><td>IN2</td><td>VAR</td><td>BYTE</td></tr>
<tr><td></td><td>%LW22</td><td>IN_OUT1</td><td>VAR</td><td>INT</td></tr>
<tr><td></td><td>%LD18</td><td>OUT1</td><td>VAR</td><td>REAL</td></tr>
</table>
</td>
</tr>
</table>

6.11.4 FOR/NEXT（循环指令）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	FOR			☑ K5 ☑ K2 ☑ KS ☑ KW ☑ HP ☑ MK ☑ K6
	NEXT			
IL	FOR	FOR INDX, INIT, FINAL	U	
	NEXT	NEXT		

参数	输入/输出	数据类型	允许的内存区
INDX	输入	INT	M、V、L、SM
INIT	输入	INT	M、V、L、SM、T、C、常量
FINAL	输出	INT	M、V、L、SM、T、C、常量

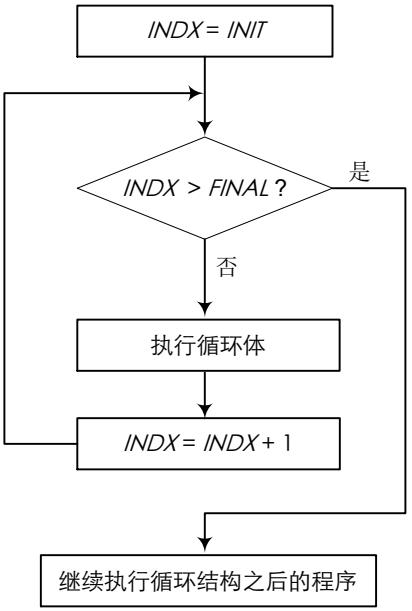
FOR/NEXT 用来实现循环，适用于循环次数已经确定的情况。

FOR 指令和 NEXT 指令必须成对使用，其中 FOR 标记了循环的开始，NEXT 标记了循环的结束。在 FOR 和 NEXT 之间的语句被称为循环体。FOR 指令和与其匹配的 NEXT 指令以及两者之间的循环体共同组成了一个完整的循环结构。

FOR/NEXT 指令反复执行循环体内的语句直至达到指定的循环次数，其中，循环次数的计数值存放在参数 *INDX* 内，而 *INIT* 指定了 *INDX* 的初始值，*FINAL* 指定了 *INDX* 的终值。

一个循环体内又包含另一个完整的循环结构，称为循环的嵌套。FOR/NEXT 循环支持嵌套，嵌套深度最大为 8 层。

FOR/NEXT 循环的执行过程如下图：



使用 FOR/NEXT 循环时，用户需要注意如下方面：

✧ FOR 指令必须是一个网络（Network）中的第二条指令；

NEXT 指令必须单独占用一个网络（Network）。

✧ 在循环体内，用户可以改变终值 *FINAL*，从而改变循环的结束条件。

✧ **若循环次数比较多，则程序的执行时间可能就会超过系统看门狗的定时值**，从而导致 CPU 扫描超时的故障。用户可以在循环体中使用 WDR 指令来延长看门狗的触发时间。

✧ ***FINAL* 参数不允许等于 32767， *INIT* 必须小于等于 *FINAL*，否则 PLC 不会执行此循环体。**

- LD

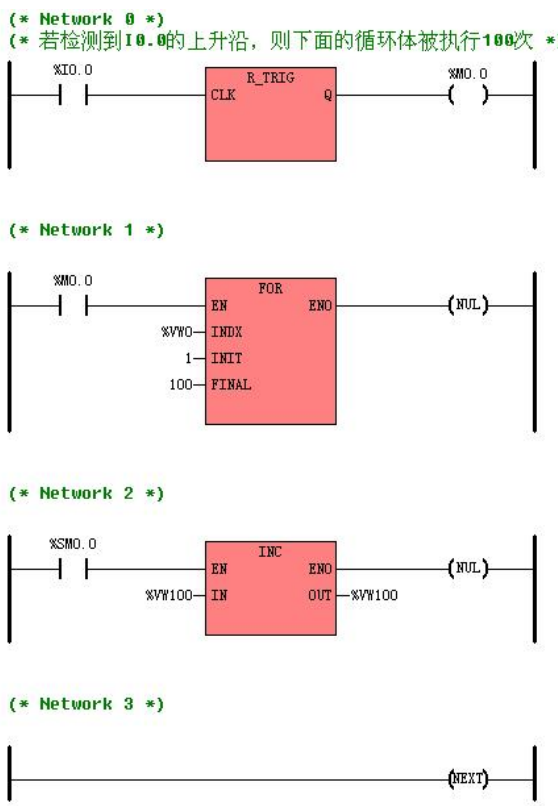
若 FOR 指令左侧能量流的值为 1，则该 FOR/NEXT 循环被执行，否则将跳过该循环，继续向下执行循环结构之后的程序。

- IL

若 FOR 指令处的 CR 值为 1，则该 FOR/NEXT 循环被执行，否则将跳过该循环，继续向下执行循环结构之后的程序。

- 指令使用举例

LD



IL	(* Network 0 *) (*若检测到 I0.0 的上升沿，则下面的循环体被执行 100 次*) LD %I0.0 R_TRIG ST %M0.0 (* Network 1 *) LD %M0.0 FOR %VW0, 1, 100 (* Network 2 *) LD %SM0.0 INC %VW100 (* Network 3 *) LD TRUE NEXT

6.11.5 END（终止主程序）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ HP ☑ MK ☑ K6
LD	END	—(END)—		
IL	END	END	U	

该指令只能在主程序中使用，用于终止主程序的执行。

在主程序的结尾，KincoBuilder 软件自动地隐含调用了 END 指令。

- LD

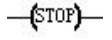
若指令左侧能量流的值为 1，则该指令被执行，否则该指令将不起作用，程序继续向下依次执行。

- IL

  若 CR 值为 1，则该指令被执行，否则该指令不起作用，程序继续向下依次执行。

6.11.6 STOP（停止 CPU）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ HP ☑ MK ☑ K6
LD	STOP			
IL	STOP	STOP	U	

该指令将 CPU 从运行（RUT）状态立即转变为停止（STOP）状态。

- LD

若指令左侧能量流的值为 1，则该指令被执行，CPU 立即被转入停止（STOP）状态，否则该指令将不起作用，程序继续向下依次执行。

- IL

若 CR 值为 1，则该指令被执行，CPU 立即被转入停止（STOP）状态，否则该指令不起作用，程序继续向下依次执行。

6.11.7 WDR（看门狗复位）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ HP ☑ MK ☑ K6
LD	WDR			
IL	WDR	WDR	U	

该指令将系统的看门狗复位并重新启动看门狗定时器。

使用 WDR 指令可以增加一次扫描所允许的时间，从而使耗时较长的程序能够顺利执行下去。但需要用户注意的是，如果程序耗时过长，那么下述任务就有可能不会被及时完成

✧ CPU 自诊断

✧ 读输入（读取物理输入通道的信号并刷新输入映像区）

✧ 通信

✧ 写输出（此处指的是将输出映像区中的数据写至物理输出通道，不包括立即输出指令）

✧ 10ms 和 100ms 定时器的计时

- LD

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

- IL

若 CR 值为 1，则该指令被执行，否则该指令不起作用，程序继续向下依次执行。

6.12 中断指令

采用中断技术的目的是提高 CPU 的执行效率，实现对内部或者外部各种预定义中断事件的快速响应。Kinco-K 系列 PLC 定义了数十种中断事件，每一种中断都被指定了唯一的事件号以供 CPU 识别。

6.12.1 Kinco-K 系列如何处理中断事件?

用户可以在程序中调用 ENI、DISI 指令来全局允许、禁止 CPU 处理中断事件。Kinco-K 系列默认全局允许处理中断事件。

若用户需要 CPU 响应某个中断事件，则必须在程序中使用 ATCH（中断连接）指令将该中断事件（用事件号进行标识）与一个中断服务程序连接起来。这样当该中断事件发生时， CPU 将自动调用一次它连接的中断服务程序进行处理。一旦中断服务程序中的最后一条指令执行完成，CPU 将返回到主循环的断点处继续执行。用户也可以在中断服务程序中调用 RETC 或者 RETCN 指令来退出该程序。

一个中断事件只能对应一个中断服务程序，但一个中断服务程序可以对应多个中断事件。中断服务程序没有参数，但允许使用局部变量。

由于有快速响应的要求，因此用户应当尽量优化中断服务程序，使其短小精干。

6.12.2 中断优先级和队列

中断事件各有不同的优先级。当中断事件发生时会按照优先级和发生的时序进行排队：队列中优先级高的中断事件优先得到处理；优先级相同的中断按照发生的时序进行处理，先发生的就先进行处理。

任何时刻都只允许有一个中断服务程序在执行。一旦一个中断服务程序开始执行，它会一直执行完成，而不会被其它任何中断服务程序打断，在它执行期间发生的任何中断事件都会进入队列等候处理。

6.12.3 中断事件分类

- 通信口中断

用于自由协议通信。

发送完成中断在发送完用户指定的数据时发生。

接收完成中断在接收到用户指定数量的数据时发生。

- I/O 中断

包含了上升沿或下降沿中断、高速计数器中断。

上升沿、下降沿中断只能由 CPU 本体集成 DI 的第 0 至 3 通道（地址为 I0.0---I0.3）捕获。

高速计数器中断在计数器复位、改变计数方向或者计数值等于预设值时发生。

- 时间中断

包含了定时中断和定时器中断。

定时中断会周期性的产生，周期单位为 0.1ms，可以利用它来完成周期性的任务。定时中断不受 PLC 扫描周期的影响，可以用于精确的定时。**(这类中断是可以产生不受 PLC 扫描周期影响的定时)**

定时器中断在 T2、T3 的当前值等于预设值时发生，可以利用它来及时响应定时事件。定时器中断会受到 PLC 扫描周期的影响。**(这类中断受 PLC 扫描周期影响)**

6.12.4 中断事件列表

下表是 Kinco-K 系列的中断事件的完整列表。需要注意的是，对于具体的 CPU 型号来说，由于不具备某些功能，所以相应的中断事件也就不支持。比如 CPU504 只有 PORT0 一个通信口，那么关于 PORT1 的中断事件就不支持。又比如 K5 系列 CPU 只支持 HSC0，HSC1，那么 HSC2 和 HSC3 的中断事件就不支持。

事件号	中断描述	分类
191	HSC3 使用多段 PV 值时第 32 个“CV=PV”	I/O 中断
…	…（依次加 1）	
161	HSC3 使用多段 PV 值时第 2 个“CV=PV”	

160	HSC3 使用多段 PV 值时第 1 个“CV=PV”	
159	HSC2 使用多段 PV 值时第 32 个“CV=PV”	
…	…（依次加 1）	
129	HSC2 使用多段 PV 值时第 2 个“CV=PV”	
128	HSC2 使用多段 PV 值时第 1 个“CV=PV”	
127	HSC1 使用多段 PV 值时第 32 个“CV=PV”	
…	…（依次加 1）	
97	HSC1 使用多段 PV 值时第 2 个“CV=PV”	
96	HSC1 使用多段 PV 值时第 1 个“CV=PV”	
95	HSC0 使用多段 PV 值时第 32 个“CV=PV”	
…	…（依次加 1）	
65	HSC0 使用多段 PV 值时第 2 个“CV=PV”	
64	HSC0 使用多段 PV 值时第 1 个“CV=PV”	
63-33	保留	
34	PORT 2：发送数据完成	通信中断
33	PORT 2：接收数据完成	
32	PORT 1：发送数据完成	
31	PORT 1：接收数据完成	
30	PORT 0：发送数据完成	
29	PORT 0：接收数据完成	
28--27	保留	I/O 中断
26	I0.0 检测到下降沿	
25	I0.0 检测到上升沿	
24	I0.1 检测到下降沿	
23	I0.1 检测到上升沿	
22	I0.2 检测到下降沿	
21	I0.2 检测到上升沿	
20	I0.3 检测到下降沿	
19	I0.3 检测到上升沿	
18	HSC0 使用单个 PV 值时 CV=PV（当前值=预设值）	

17	HSC0 输入方向改变	
16	HSC0 外部复位	
15	HSC1 使用单个 PV 值时 CV=PV（当前值=预设值）	
14	HSC1 输入方向改变	
13	HSC1 外部复位	
12	HSC2 使用单个 PV 值时 CV=PV（当前值=预设值）	
11—10	保留	
9	HSC3 使用单个 PV 值时 CV=PV（当前值=预设值）	
8—5	保留	
4	定时中断 1。周期在 SMD16 中指定，单位 0.1ms。**（不受扫描周期影响）**	时间中断
3	定时中断 0。周期在 SMD12 中指定，单位 0.1ms。**（不受扫描周期影响）**	
2	定时器 T3 ET=PT（当前值=预设值）**（受扫描周期影响）**	
1	定时器 T2 ET=PT（当前值=预设值）**（受扫描周期影响）**	

表 6-1　K 系列 PLC 的中断事件列表

事件号为 3，4 的定时中断 1， 定时中断 2， 不受 PLC 扫描的影响， 但受更高优先级的硬件中断影响， 所定时的时间大多偏长， 用户可以在测转速，流量时， 根据自己的应用程序， 加以补偿。

事件号为 1，2 的定时器中断，受 PLC 扫描的影响， 精确定时建议用 3，4 中断。

6.12.5 ENI（允许中断）、DISI（禁止中断）指令

- 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5

LD	ENI	—(ENI)—		☑ K2 ☑ KS ☑ KW ☑ HP ☑ MK ☑ K6
	DISI	—(DISI)—		
IL	ENI	ENI	U	
	DISI	DISI		

ENI、DISI 指令在程序中执行一次即可。PLC 默认是允许程序处理中断事件的。

- LD

ENI 指令的作用是：若指令左端能量流的值为 1，则全局允许处理中断事件（即当中断发生时 CPU 允许调用中断服务程序），否则不执行该指令。

DISI 指令的作用是：若指令左端能量流的值为 1，则全局禁止处理中断事件，否则不执行该指令。

- IL

ENI 指令的作用是：若 CR 值为 1，则全局允许处理中断事件，否则不执行该指令。

DISI 指令的作用是：若 CR 值为 1，则全局禁止处理中断事件，否则不执行该指令。

ENI、DISI 指令的执行均不影响 CR 值。

6.12.6 ATCH（中断连接）、DTCH（中断分离）指令

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	ATCH	ATCH EN ENO INT EVENT		☑ K5 ☑ K2 ☑ KS

	DTCH			☑ KW ☑ HP ☑ MK ☑ K6
IL	ATCH	ATCH INT, EVENT	U	
	DTCH	DTCH EVENT		

参数	输入/输出	数据类型	参数描述
INT	输入	标识符	用户工程中已存在的中断服务程序的名称
EVENT	输入	INT 型常量	中断事件号

ATCH 指令的功能是：将 *EVENT* 指定的中断事件与 *INT* 指定的中断服务程序连接起来，这样若 CPU 允许处理中断事件，则当该中断事件发生时，将自动调用该中断服务程序。多个中断事件可以连接一个中断服务程序，但一个中断事件不允许与多个中断服务程序连接。

DTCH 指令实现的功能是：取消参数 *EVENT* 指定的中断事件与所有中断服务程序的连接。

- LD

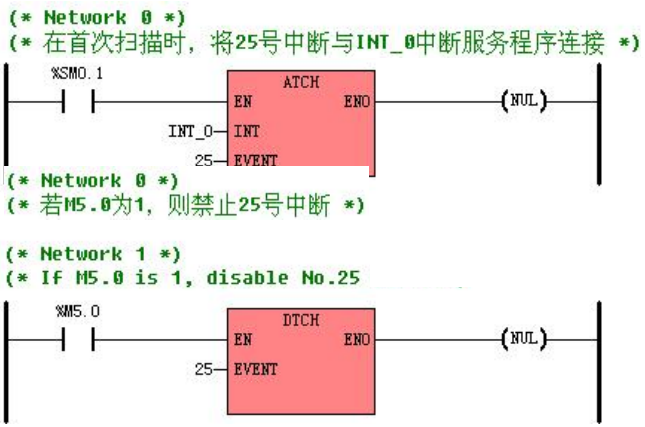
  若 *EN* 值为 1，则 *ATCH*、*DTCH* 指令被执行，否则不执行。

- IL

  如果 CR 值为 1，则 *ATCH*、*DTCH* 指令被执行，否则不执行。它们的执行不影响 CR 值。

➢ 指令使用举例

LD



IL

```
(* NETWORK 0 *)
LD      %M5.0       (* 建立 CR，其值为 M5.0 *)
ENI                 (* 若 CR 值为 1，则允许进行中断处理 *)

LDN     %M5.0       (* 建立 CR，其值为 M5.0 取反后的值 *)
DISI                (* 若 CR 值为 0，则禁止进行中断处理 *)

LD      %SM0.1      (* 在首次扫描时，将 25 号中断与 INT_0 中断服务程序连接 *)
ATCH    INT_0, 25

LD      %I1.0       (* 建立 CR，其值为 I1.0 *)
DTCH    25          (* 若CR值为1,则取消 25 号中断与所有中断服务程序的连接 *)
```

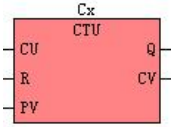
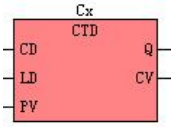
6.13 计数器

计数器是 IEC61131-3 标准中定义的功能块之一，共有 CTU、CTD 和 CTUD 三种。

关于功能块及其实例的使用请参阅 3.6.5 关于功能块以及功能块实例中的描述。注意，计数器实例不允许重复使用，在用户工程中不允许将同一个计数器实例分配给多个计数器指令。

6.13.1 CTU（增计数器）、CTD（减计数器）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	CTU			☑ K5 ☑ K2 ☑ KS ☑ KW ☑ HP ☑ MK ☑ K6
	CTD			
IL	CTU	CTU Cx, R, PV	P	
	CTD	CTD Cx, LD, PV		

参数	输入/输出	数据类型	允许使用的内存区
Cx	-	计数器实例（*x* 表示编号）	C
CU	输入	BOOL	能量流
R	输入	BOOL	I、Q、M、V、L、SM、T、C
CD	输入	BOOL	能量流
LD	输入	BOOL	I、Q、M、V、L、SM、T、C
Q	输出	BOOL	能量流
CV	输出	INT	Q、M、V、L、SM、AQ

- LD

CTU 指令对 *CU* 输入端的上升沿进行递增计数（每次加 1）并把 *Cx* 的计数值赋给 *CV*。*Cx* 将一直计数直至达到并保持在最大值 32767。当 *CV* 大于等于预设值 *PV* 时，*Q* 及 *Cx* 的状态值均被置为 1，否则均被置为 0。若 *R* 输入为 1，则 *Cx* 被复位，*CV* 及其计数值全被清零。

CTD 指令对 *CD* 输入端的下降沿进行递减计数（每次减 1）并把 *Cx* 的计数值赋给 *CV*。当 *CV* 等于 0 时，*Cx* 停止计数，*Q* 及 *Cx* 的状态值均被置为 1，否则均被置为 0。若 *LD* 输入为 1，则 *Cx* 被复位，将 *PV* 重新赋给 *Cx* 的计数值及 *CV*。

- IL

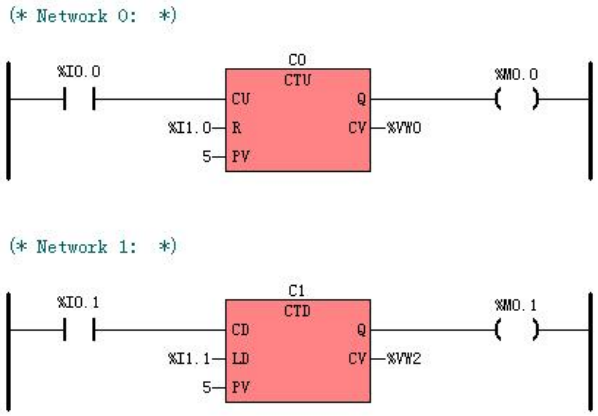
CTU 指令对 CR 值的上升沿进行递增计数（每次加 1）。*Cx* 将一直计数直至达到并保持在最大值 32767。当计数值大于等于预设值 *PV* 时，*Cx* 的状态值被置为 1，否则被置为 0。若 *R* 输入为 1，则 *Cx* 被复位，其计数值被清零。

CTD 指令对 CR 值的下降沿进行递减计数（每次减 1）。当计数值等于 0 时，*Cx* 停止计数，其状态值被置为 1，否则被置为 0。若 *LD* 输入为 1，则 *Cx* 被复位，将 *PV* 重新赋给 *Cx* 的计数值。

每次扫描 CTU、CTD 指令后，CR 值就被更新为 *Cx* 的状态值。

➢ CTU、CTD 使用举例

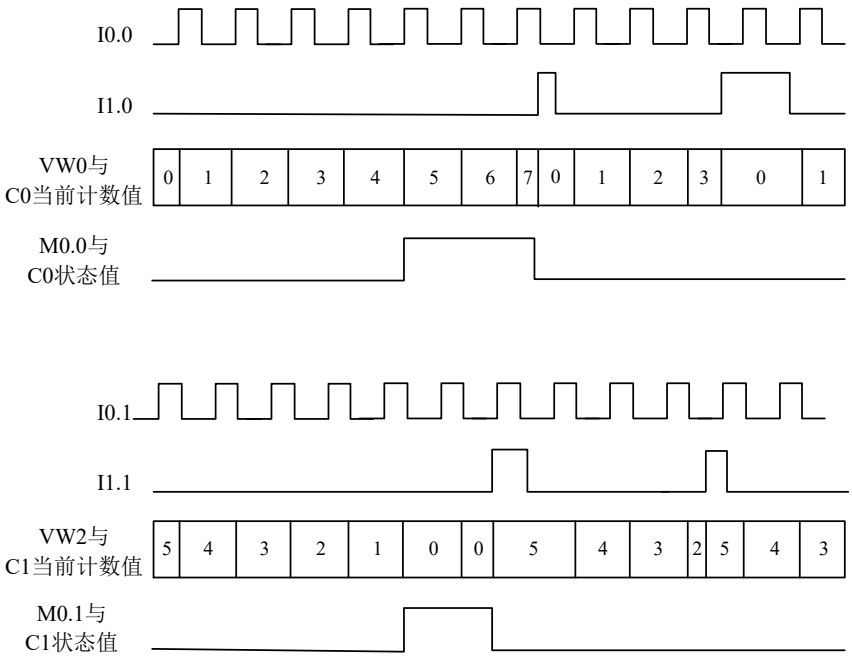
LD



IL

```
(* NETWORK 0 *)
LD      %I0.0
CTU     C0, %I1.0, 5
ST      %M0.0

(* NETWORK 1 *)
LD      %I0.1
CTD     C1, %I1.1, 5
ST      %M0.1
```

时序图



6.13.2 CTUD（增/减计数器）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ HP ☑ MK ☑ K6
LD	CTUD	Cx CTUD CU QU CD QD R CV LD PV		
IL	CTUD	CTUDCx, CD, R, LD, PV, QD	P	

参数	输入/输出	数据类型	允许使用的内存区
Cx	-	计数器实例（*x* 表示编号）	C
CU	输入	BOOL	能量流
CD	输入	BOOL	I、Q、M、V、L、SM、T、C、RS、SR
R	输入	BOOL	I、Q、M、V、L、SM、T、C、RS、SR
LD	输入	BOOL	I、Q、M、V、L、SM、T、C、RS、SR
PV	输入	INT	I、Q、M、V、L、SM、AI、AQ、常量
QU	输出	BOOL	能量流
QD	输出	BOOL	Q、M、V、L、SM
CV	输出	INT	Q、M、V、L、SM、AQ

- LD

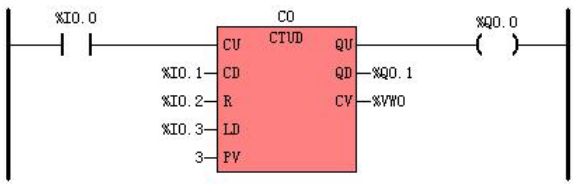
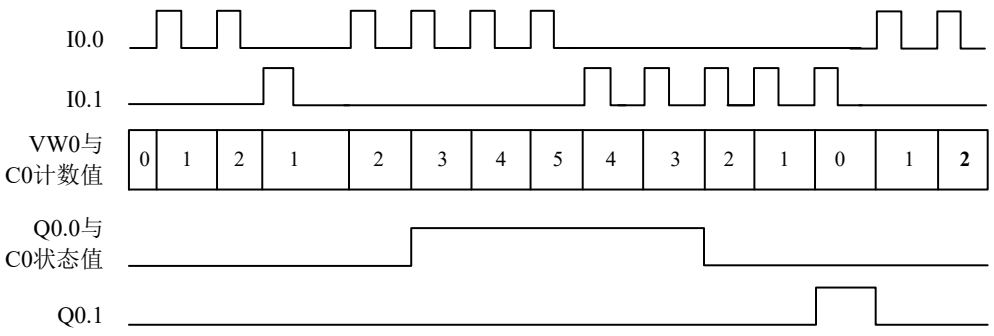
CTUD 指令对 *CU* 输入端的上升沿进行递增计数（每次加 1）并对 *CD* 输入端的上升沿进行递减计数（每次减 1），*Cx* 的计数值被赋给 *CV*。当 *CV* 大于等于预设值 *PV* 时，*QU* 及 *Cx* 的状态值均被置为 1，否则均被置为 0。当 *CV* 等于 0 时，*QD* 被置为 1，否则被置为 0。若 *R* 输入为 1，则 *Cx* 被复位，其计数值及 *CV* 均被清零。若 *LD* 输入为 1，则将 *PV* 重新赋值给 *Cx* 的计数值及 *CV*。当 *R* 及 *LD* 同时输入为 1 时，*R* 优先。

- IL

CTUD 指令对 CR 值的上升沿进行递增计数（每次加 1）并对 *CD* 输入端的上升沿进行递减计数（每次减 1）。当计数值大于等于预设值 *PV* 时，*Cx* 的状态值被置为 1，否则被置为 0。当计数值等于 0 时，*QD* 被置为 1，否则被置为 0。若 *R* 输入为 1，则 *Cx* 被复位，其计数值被清零。若 *LD* 输入为 1，则将 *PV* 重新赋值给 *Cx* 的计数值。当 *R* 及 *LD* 同时输入为 1 时，*R* 优先。

每次扫描 CTUD 指令后，CR 值就被更新为 *Cx* 的状态值。

➢ CTUD 使用举例

<table>
<tr><th>LD</th><th>IL</th></tr>
<tr><td> 
</td><td>LD %I0.0
CTUD C1,%I0.1,%I0.2,%I0.3, 4 ,%Q0.1
ST %Q0.0</td></tr>
<tr><th colspan="2">时序图</th></tr>
<tr><td colspan="2"> 
</td></tr>
</table>

6.14 定时器

定时器是 IEC61131-3 标准中定义的功能块之一，共有 TON、TOF 和 TP 三种。

关于功能块及其实例的使用请参阅 3.6.5 关于功能块以及功能块实例一节。

6.14.1 定时器的时基

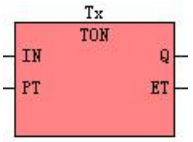
KPLC 提供了三种时基的定时器，定时器号决定了其时基，其中 T0-T3 的时基为 1ms，T4-T19 的是 10ms，T20-T255 的是 100ms。

定时器的最大定时时间为 32767×时基。定时器的预设值和计时值均是其时基的倍数。比如，对于 10ms 时基的定时器，100 就代表 1000ms。

只有定时器指令正在执行时，PLC 才对定时器的 ET 值进行更新，因此会受到扫描周期的影响。如需精确定时，请使用定时中断。

6.14.2 TON（接通延时定时器）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	TON			
IL	TON	TON Tx, PT	P	

参数	输入/输出	数据类型	允许使用的内存区
Tx	-	定时器实例	T
IN	输入	BOOL	能量流
PT	输入	INT	I、AI、AQ、M、V、L、SM、常量
Q	输出	BOOL	能量流
ET	输出	INT	Q、M、V、L、SM、AQ

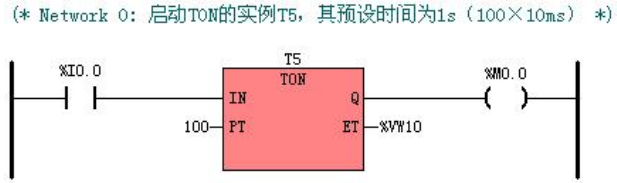
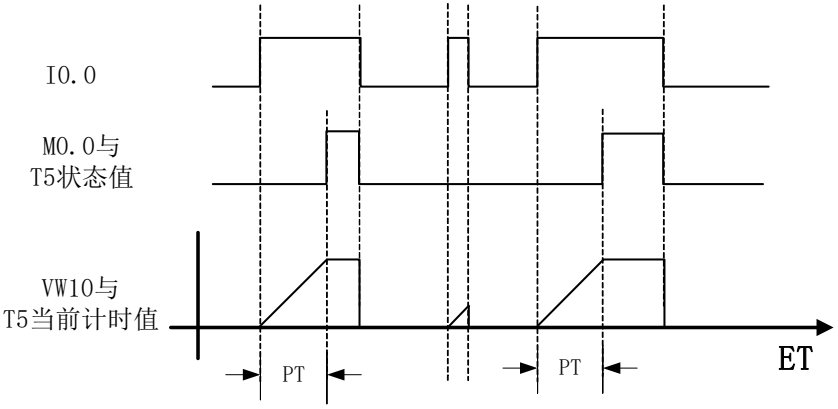
- LD

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

- IL

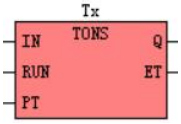
若检测到 CR 值的上升沿，则 *Tx* 开始启动定时，当计时值大于等于预设值 *PT* 时，*Tx* 停止，其状态值被置为 1。若 CR 值变为 0，则 *Tx* 被复位，其状态值及计时值均被清零。每次扫描 *TON* 后，CR 值均被设置为 *Tx* 的状态值。

➢ TON 使用举例

<table>
<tr><th>LD</th><th>IL</th></tr>
<tr><td> 
</td><td>(* NETWORK 0 *)
LD %I0.0
TON T5, 100
ST %M0.0</td></tr>
<tr><td colspan="2">时序图</td></tr>
<tr><td colspan="2"> 
</td></tr>
</table>

6.14.3 TONS（接通延时累积定时器）

➢ 指令及其操作数说明

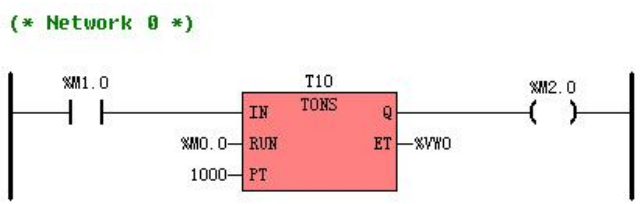
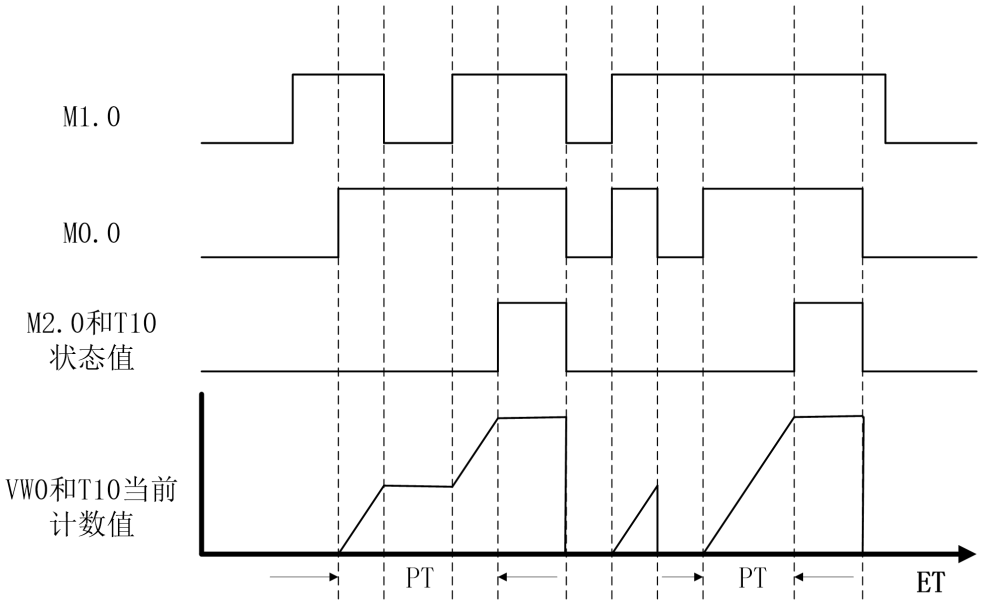
	名称	指令格式	影响 CR 值	☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	TONS			
IL	TONS	TONS Tx, RUN, PT	P	

参数	输入/输出	数据类型	允许使用的内存区
Tx	-	定时器实例	T
IN	输入	BOOL	能量流
RUN	输入	BOOL	I、Q、M、V、L、SM
PT	输入	INT	I、AI、AQ、M、V、L、SM、常量
Q	输出	BOOL	能量流
ET	输出	INT	Q、M、V、L、SM、AQ

- LD

若检测到输入端 *IN* 和 *RUN* 均变为 1 时，则 *Tx* 开始启动定时，当计时值 *ET* 大于等于预设值 *PT* 时，*Tx* 停止，其输出 *Q* 及其状态值均被置为 1。若输入 *IN* 变为 0，则 *Tx* 被暂停计时，其输出 *Q*、状态值、计时值 *ET* 都不会发生改变。若输入 *RUN* 变为 0，其输出 *Q* 及状态值均被置为 0，同时计时值 *ET* 也被清零。

➢ TONS 使用举例

<table>
<tr><th>LD</th><th>说明</th></tr>
<tr><td> 
</td><td>当 M1.0 和 M0.0 同时为 1 时，T10 开始计时，预设时间 10s。当 M1.0 变为 0 时，定时器暂停计时，ET 值保持不变；当 M1.0 重新变为 1 时，定时器继续计时，ET 继续累积。当计数值大于等于预设时间后输出 Q 和 T10 均被置 1，当 M0.0 为 0 时其输出 Q 及状态值均被置为 0，同时计时值 ET 也被清零。</td></tr>
<tr><td colspan="2">时序图</td></tr>
<tr><td colspan="2"> 
</td></tr>
</table>

6.14.4 TOF（断开延时定时器）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	TOF			
IL	TOF	TOF Tx, PT	P	

参数	输入/输出	数据类型	允许使用的内存区
Tx	-	定时器实例	T
IN	输入	BOOL	能量流
PT	输入	INT	I、AI、AQ、M、V、L、SM、常量
Q	输出	BOOL	能量流
ET	输出	INT	Q、M、V、L、SM、AQ

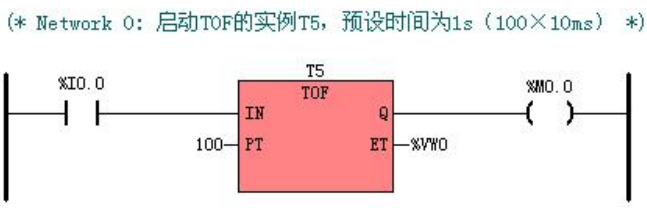
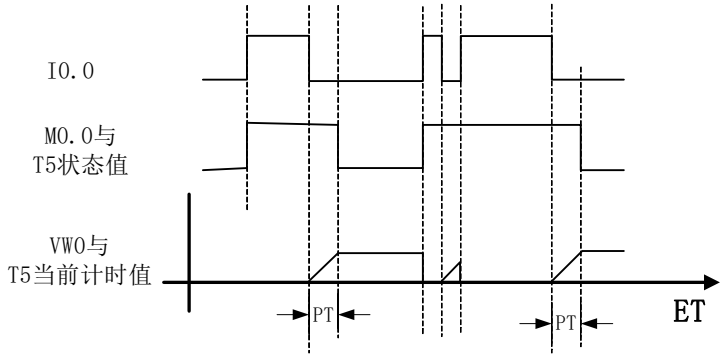
- LD

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

- IL

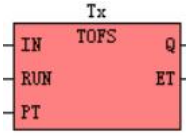
若检测到 CR 值的下降沿，则 *Tx*开始启动定时，当计时值大于等于预设值 *PT*时，*Tx*停止，其状态值被置为 0。若 CR 值变为 1，则 *Tx*被复位，其状态值被置为 1，且计时值被清零。每次扫描 *TOF*后，CR 值均被设置为 *Tx*的状态值。

➢ TOF 使用举例

<table>
<tr><td>LD</td><td>IL</td></tr>
<tr><td> 
</td><td>(* NETWORK 0 *)
LD %I0.0
TOF T5, 100
ST %M0.0</td></tr>
<tr><td colspan="2">时序图</td></tr>
<tr><td colspan="2"> 
</td></tr>
</table>

6.14.5 TOFS（断开延时累积定时器）

➢ 指令及其操作数说明

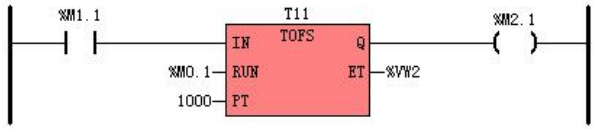
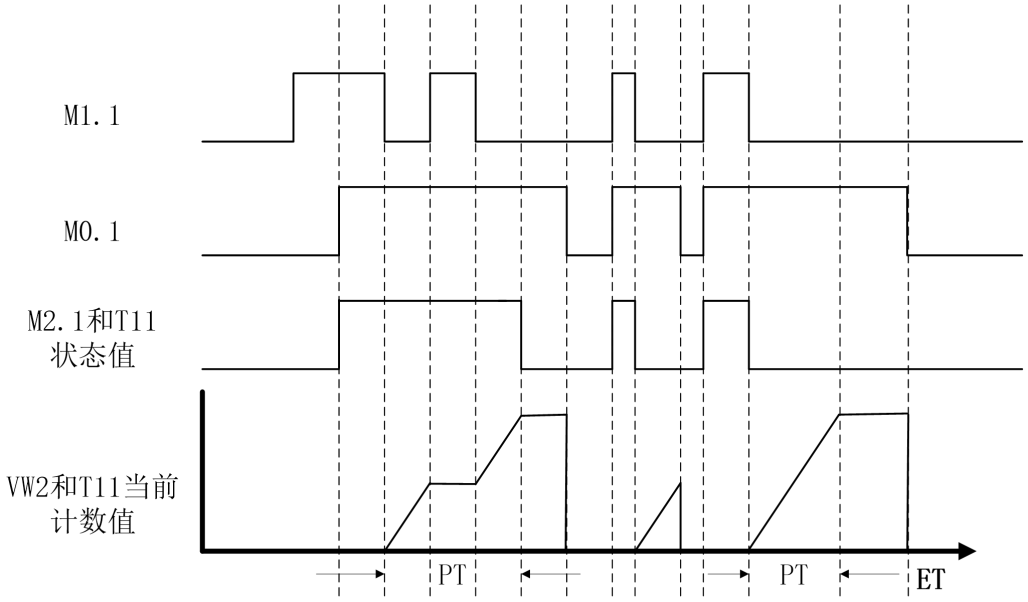
	名称	指令格式	影响 CR 值	☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	TOFS			
IL	TOFS	TOFS Tx, RUN, PT	P	

参数	输入/输出	数据类型	允许使用的内存区
Tx	-	定时器实例	T
IN	输入	BOOL	能量流
RUN	输入	BOOL	I、Q、M、V、L、SM
PT	输入	INT	I、AI、AQ、M、V、L、SM、常量
Q	输出	BOOL	能量流
ET	输出	INT	Q、M、V、L、SM、AQ

- LD

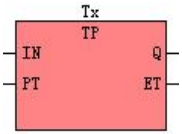
若检测到输入端 *IN* 和 *RUN* 均变为 1 后，其输出 *Q* 及其状态值均被置为 1，输入端 *IN 再变为 0* 则 *Tx* 开始启动定时，当计时值 *ET* 大于等于预设值 *PT* 时，*Tx* 停止，其输出 *Q* 及其状态值均被置为 0。若输入 *IN 计数中途*变为 1，则 *Tx* 被暂停计时，其输出 *Q*、状态值、计时值 *ET* 都不会发生改变。若输入 *RUN* 变为 0，其输出 *Q* 及状态值均被置为 0，同时计时值 *ET* 也被清零。

➢ TOFS 使用举例

LD	说明
	当 M1.1 和 M0.1 同时为 1 时输出 Q 和 T11 均被置 1，当 M1.1 变为 0 后开始计时，预设时间 10s，当 M1.1 变为 1 时定时器暂停计时，当 M1.1 重新变为 1 时在上次累积的时间上继续计时，当计数值大于等于预设时间后输出 Q 和 T11 均被置 0，当 M0.1 为 0 时其输出 *Q* 及状态值均被置为 0，同时计时值 *ET* 也被清零。
时序图	
	

6.14.6 TP（脉冲定时器）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
LD	TP			
IL	TP	TP Tx, PT	P	

参数	输入/输出	数据类型	允许使用的内存区
Tx	-	定时器实例	T
IN	输入	BOOL	能量流
PT	输入	INT	I、AI、AQ、M、V、L、SM、常量
Q	输出	BOOL	能量流
ET	输出	INT	Q、M、V、L、SM、AQ

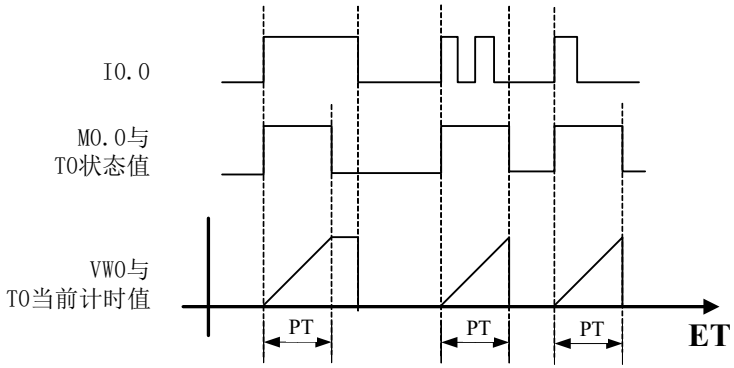
- LD

若检测到在输入端 *IN* 的上升沿，则 *Tx* 开始启动定时，在其输出端 *Q* 及其状态值输出一个恒定宽度的脉冲，脉宽值为预设时间 *PT*。参数 *ET* 中存放着 *Tx* 的计时值。

- IL

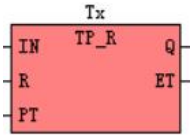
若检测到 CR 值的上升沿，则 *Tx* 开始启动定时，它的状态值输出一个恒定宽度的脉冲，脉宽值为预设时间 *PT*。每次扫描 *TP* 后，CR 值均被设置为 *Tx* 的状态值。

➢ TP 使用举例

LD	IL
(* Network 0: 启动TP的实例T20，预设时间为1s（1000×1ms） *) %I0.0 T0 TP IN Q 1000 PT ET %VW0 %M0.0	(* NETWORK 0 *) LD %I0.0 TP T0, 1000 ST %M0.0
时序图	
	

6.14.7 TP_R（可复位的脉冲定时器）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	TP_R			☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
IL	TP_R	TP_R Tx, R, PT	P	

参数	输入/输出	数据类型	允许使用的内存区
Tx	-	定时器实例	T
IN	输入	BOOL	能量流
R	输入	BOOL	I、Q、M、V、L、SM
PT	输入	INT	I、AI、AQ、M、V、L、SM、常量
Q	输出	BOOL	能量流
ET	输出	INT	Q、M、V、L、SM、AQ

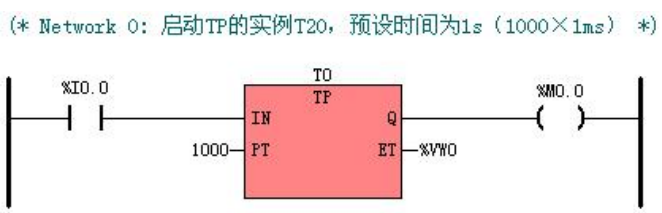
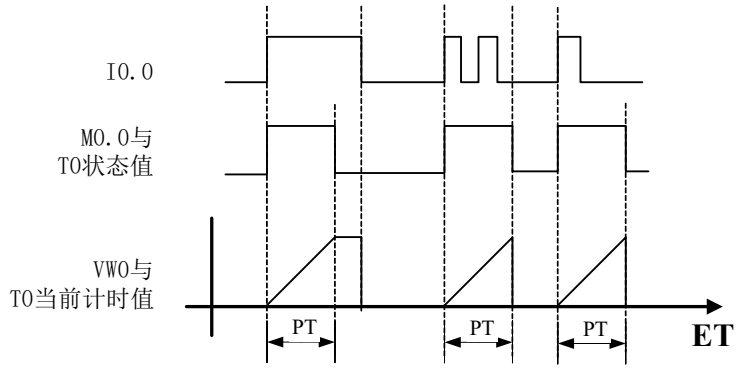
- LD

输入端 *R* 用于复位定时器。若 *R* 为 0，当检测到在输入端 *IN* 的上升沿，*Tx* 开始启动定时，在输出端 *Q* 及其状态值会输出一个恒定宽度的脉冲，脉宽值为预设时间 *PT*。参数 *ET* 中存放着 *Tx* 的计时值。若 *R* 为 1，则 *Tx* 被复位，输出端 *Q* 及其状态值会变为 0，计时值 *ET* 也会被清零。

- IL

输入端 *R* 用于复位定时器。若 *R* 为 0，当检测到 CR 值的上升沿，则 *Tx* 开始启动定时，它的状态值会输出一个恒定宽度的脉冲，脉宽值为预设时间 *PT*。若 *R* 为 1，则 *Tx* 被复位，其状态值会变为 0，计时值也会被清零。PLC 每次扫描 *TP* 后，CR 值均被设置为 *Tx* 的状态值。

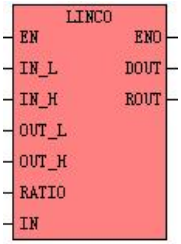
➢ TP_R 使用举例

<table>
<tr><th>LD</th><th>IL</th></tr>
<tr><td> 
</td><td>(* NETWORK 0 *)
LD %I0.0
TP T0, 1000
ST %M0.0</td></tr>
<tr><td colspan="2">时序图</td></tr>
<tr><td colspan="2"> 
</td></tr>
</table>

6.15 附加指令

6.15.1 LINCO（线性变换）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	适用于
LD	LINCO			☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
IL	LINCO	LINCO IN_L, IN_H, OUT_L, OUT_H, RATIO, IN, DOUT, ROUT	U	

参数	输入/输出	数据类型	允许的内存区
IN_L	输入	INT	I、Q、V、M、L、SM、T、C、AI、AQ、常量
IN_H	输入	INT	I、Q、V、M、L、SM、T、C、AI、AQ、常量
OUT_L	输入	REAL	V、L、常量
OUT_H	输入	REAL	V、L、常量
RATIO	输入	REAL	常量
IN	输入	INT	I、Q、V、M、L、SM、T、C、AI、AQ
DOUT	输出	DINT	Q、M、V、L、SM
ROUT	输入	REAL	V、L

注意：参数 IN_L,、IN_H、OUT_L 和 OUT_H 必须全为常量或者全为变量。

LINCO 指令将输入 *IN* 按照指定的线性关系进行计算，并将计算结果乘以系数 *RATIO* 后赋给 *ROUT*，

然后将 *ROUT* 取整（舍弃小数）后赋给 *DOUT*。所用的线性关系如下：输入下限 *IN_L* 和输出下限 *OUT_L*、输入上限 *IN_H* 和输出上限 *OUT_H*，将这些参数按照“两点定一直线”的方法进行计算，从而得到其中的线性关系。

LINCO 指令的作用可以用如下公式描述：

ROUT ＝ RATIO×(k×IN + b)

DOUT ＝ TRUNC(ROUT)

其中 $k = \frac{OUT\_H - OUT\_L}{IN\_H - IN\_L}$ ； $b = OUT\_L - k \times IN\_L$ 。

- LD

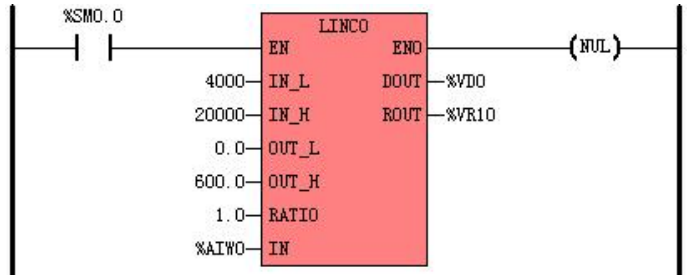
  如果 *EN* 为 1，则该指令被执行。

- IL

  如果 CR 值为 1，则该指令被执行。该指令的执行不影响 CR 值。

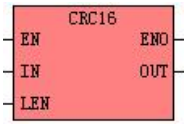
➢ 指令使用举例

假定现场一个温度变送器的测量范围是 0∽600℃，信号输出范围是 4∽20mA。变送器的输出信号接至 PLC 的 AIW0 通道。要求 PLC 计算出实际的温度值。

LD	
IL	LD %SM0.0 LINCO 4000, 20000, 0.0, 600.0, 1.0, %AIW0, %VD0, %VR10

6.15.2 CRC16（16 位 CRC 校验码）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	适用于
LD	CRC16			☑ K5 ☑ K2 ☑ KS
IL	CRC16	CRC16 IN, OUT, LEN	U	☑ KW ☑ MK ☑ K6

参数	输入/输出	数据类型	允许的内存区
IN	输入	BYTE	I、Q、M、V、L、SM
LEN	输入	BYTE	I、Q、M、V、L、SM、常量
OUT	输出	BYTE	Q、M、V、L、SM

该指令用于计算指定数据的 16 位 CRC 校验码。被校验的数据存放在从 *IN* 开始连续 *LEN* 个字节长度的区域内。计算结果存放在从 *OUT* 开始连续 2 个字节长度的区域内，其中，*OUT* 存放着 CRC 码的高字节，*OUT* 的下一个字节存放着 CRC 码的低字节。

注意所用内存区域不能超出有效的内存范围，否则执行结果不可预期。

- LD

  若 *EN* 值为 1，则该指令被执行，否则不执行。

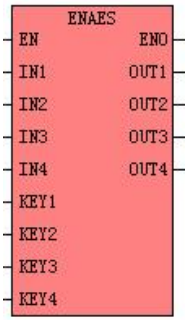
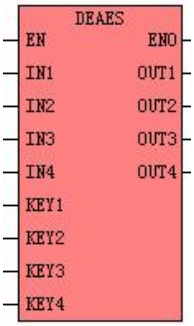
- IL

  如果 CR 值为 1，则该指令被执行，否则不执行。

  该指令的执行不影响 CR 值。

6.15.3 ENAES（AES-128 加密） DEAES（AES-128 解密）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	适用于
LD	ENAES			☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
	DEAES			
	ENAES	ENAES IN1, IN2, IN3, IN4, KEY1, KEY2, KEY3, KEY4, OUT1, OUT2, OUT3, OUT4	U	
	DEAES	DEAES IN1, IN2, IN3, IN4, KEY1, KEY2, KEY3, KEY4, OUT1, OUT2, OUT3, OUT4		

参数	输入/输出	数据类型	允许的内存区
IN1	输入	DWORD	I、Q、L、M、V、SM、常量

IN2	输入	DWORD	I、Q、L、M、V、SM、常量
IN3	输入	DWORD	I、Q、L、M、V、SM、常量
IN4	输入	DWORD	I、Q、L、M、V、SM、常量
KEY1	输入	DWORD	I、Q、L、M、V、SM、常量
KEY2	输入	DWORD	I、Q、L、M、V、SM、常量
KEY3	输入	DWORD	I、Q、L、M、V、SM、常量
KEY4	输入	DWORD	I、Q、L、M、V、SM、常量
OUT1	输出	DWORD	Q、SM、L、M、V
OUT2	输出	DWORD	Q、SM、L、M、V
OUT3	输出	DWORD	Q、SM、L、M、V
OUT4	输出	DWORD	Q、SM、L、M、V

IN1, IN2, IN3, IN4, KEY1, KEY2, KEY3, KEY4 必须同时为常量类型或同时为内存类型。

ENAES、DEAES 指令采用 AES-128 方式的加密、解密指令。

其中，参数 *IN1*、*IN2*、*IN3*、*IN4* 是待加密/解密的数据，*KEY1*、*KEY2*、*KEY3*、*KEY4* 是用户指定的密钥，*OUT1*、*OUT2*、*OUT3*、*OUT4* 是加密/解密后的数据。

- LD

  如果 *EN* 为 1，则该指令被执行。

- IL

  如果 CR 值为 1，则该指令被执行。该指令的执行不影响 CR 值。

6.15.4 特殊数据区读写指令

K 系列 PLC 在永久存储器中提供了一片 128 字节长度的特殊数据区域，并以 4 字节为单位划分为 32 个相互独立的块，用户可以对任意块进行读写。

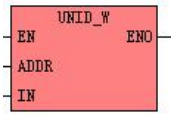
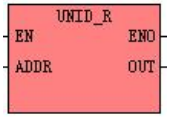
该区域内的数据在 PLC 出厂时已经被全部清零，**用户只能对每个块写入一次非 0 的数据。PLC 会将具有非 0 数据的块锁定，不允许再次写入，但可以任意读取**。注意：在写入时，待写入的 4 个字节

数据不可以全部为 0，否则会忽略此次写入。

在 KincoBuillder 中执行【PLC】->【清除…】菜单命令，可以将 PLC 内的所有数据清零，包括用户工程、特殊区域内的数据等。**此特殊数据区只能使用清除命令清除，无法单独清除。**

设备制造商可以利用此区域，在 PLC 中写入 S/N 之类的数据。

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	适用于
LD	UNID_W			☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
	UNID_R			
	UNID_W	UNID_W ADDR, IN	U	
	UNID_R	UNID_W ADDR, OUT		

参数	输入/输出	数据类型	允许的内存区
ADDR	输入	INT	I、Q、V、L、M、T、C、SM、AI、AQ、常量
OUT	输出	DWORD	Q、L、M、V、SM
IN	输入	DWORD	I、Q、L、M、V、SM、常量

UNID_W 用于将参数 IN 指定的数据写入特殊数据区域内某个块中。参数 ADDR 指定了将要写入的块编号，范围是 0-31。

UNID_R 用于读取特殊数据区域内某个块的数据并存放在参数 OUT 中。参数 ADDR 指定了将要读取的块编号，范围是 0-31

- LD

  如果 *EN* 为 1，则该指令被执行。

- IL

  如果 CR 值为 1，则该指令被执行。

  该指令的执行不影响 CR 值。

第七章 实时时钟介绍

7.1 实时时钟

Kinco-K系列PLC CPU本体内都集成了实时时钟(RTC)，可提供实时的时间/日历表示（CPU504不支持实时时钟）。实时时钟/日历的秒至年采用BCD格式编码，自动进行闰年调整。断电时，实时时钟使用后备电容供电。常温下，后备的时间不低于3年。

Kinco-K系列PLC允许使用特定规格的的锂电池作为后备电池。当断电时，后备电池用于给实时时钟供电来维持时钟的运行，同时也给RAM供电来进行数据保持。常温下，电池典型寿命为5年,断电电保持的时间累计不小于3年。

后备电池可以拆卸，在模块做侧面上部是电池盒的位置，电池就安装在电池盒中。当后备电池耗完电后，用户可以打开电池盒自行更换一个新的电池。电池为CR2032带连接器的3V锂电池，形状如下图，用户可以单独订购电池。

如果电池电量耗尽，则PLC断电后实时时钟不能保持。KPLC针对电池电量提供了如下报警寄存器，用户可以在程序中使用。

SM0.7	只读。SM0.7 为 TRUE，电池电压低， SM0.7 为 FALSE，电池电压正常。
SMW10	只读。存放后备电池的电压值，单位：0.01V。（仅用于 K5） 若后备电池的电源持续低于 2.6V 时，PLC 会产生“后备电池带电量低”报警。

注意：MK系列一体机的后备电池给HMI和PLC部分的RTC供电来维持时钟的运行，同时也给RAM供电来进行数据保持。屏上的实时时钟（RTC），也可提供实时的时间/日历表示，用户可以在系统设置画面修改时间或者通过特殊寄存器LW10000-LW10006修改时间，HMI的时钟设置方法参考Kinco DTOOLS使用手册章节2.7时钟设置（2.7.3系统时间与PLC CPU时间同步）。

实时时钟的运行误差约为3-5分钟/月，因此用户若需要使用精确的时间，需要将时钟定期设置为标准时间。

PLC调整实时时钟的方式有两种：一是通过KincoBuilder软件对CPU的实时时钟进行调整，二是使

用读写实时时钟的指令（READ_RTC、SET_RTC、RTC_W、RTC_R），实现实时时钟的调整和相关的控制应用。

7.1.1 调整 CPU 时钟

实时时钟在正式使用之前必须进行至少一次时间调整，将其时间调整为当前的实际时间。在调整之前，PLC 内的时间值可能是一个随机值。

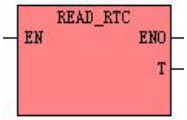
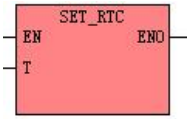
执行【PLC】→【调整 CPU 时钟…】菜单命令进入“调整 CPU 时钟”对话框，如下图。

- **系统当前时间**：显示编程所用 PC 机当前的日期、时间。
- **PLC 当前时间**：显示所连 CPU 内实时时钟的当前时间。若背景色是绿色，代表成功地从 CPU 内读到了实时时钟。若背景色是黄色，则代表从 CPU 内读取实时时钟失败。
- **将 PLC 时间调整为**：用户可以在此输入将要设置的 CPU 实时时钟的新时间值。用户既可以在输入框内直接键盘输入，也可以使用鼠标左键单击输入框右端的箭头然后进行选择、调整。
- **使用夏时制**：在使用夏时制的国家或者地区，需要选中此项。
- 单击【修改 PLC 时钟】按钮，则用户输入的新的时间值将被写入 CPU 内的实时时钟中。

注意：修改时区或夏时制之后， 必须重启 kincoBuilder（这是 windows 的机制）。

7.1.2 READ_RTC（读实时时钟）、SET_RTC（设置实时时钟）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	READ_RTC			☑ K5（CPU504 除外） ☑ K2 ☑ KS ☑ KW ☑ HP ☑ MK ☑ K6
	SET_RTC			
IL	READ_RTC	READ_RTC *T*	U	
	SET_RTC	SET_RTC *T*		

参数	输入/输出	数据类型	允许的内存区
T	输入（SET_RTC）	BYTE	V
	输出（READ_RTC）		

注意， T 参数为一个可变长度的块内存参数， 整个块内存都不可以落在非法内存区域， 否则结果不可预期。

READ_RTC 指令用于从硬件时钟中读取当前的日期和时间并放入时间缓冲区中。

SET_RTC 指令用于将时间缓冲区指定的日期和时间写入硬件时钟。

参数 *T* 定义了时间缓冲区的起始地址，该区域占用连续 8 个字节的内存空间。缓冲区内所有的数值均以 BCD 格式编码。注意该缓冲区不能超出有效的内存范围，否则执行结果不可预期。

时间缓冲区内的数据存储形式如下表所示。

内存字节	数据的含义	备注
T	星期	范围：1~7，其中 1 代表星期一，7 代表星期日。
T+1	秒	范围：0~59
T+2	分	范围：0~59
T+3	时	范围：0~23
T+4	日	范围：1~31
T+5	月	范围：1~12
T+6	年	范围：0~99
T+7	世纪	固定为 20

表 6-2 实时时钟指令的时间缓冲区

提示：

（1）执行 KincoBuilder 中的【PLC】→【调整 CPU 时钟…】菜单命令也可以读写 CPU 时钟。建议用户在使用实时时钟之前首先使用菜单命令为 CPU 设置好正确的时钟。

（2）CPU 不检查用户输入的日期、时间是否有效，无效的日期（比如 2 月 30 日）也会被 CPU 接受，因此用户在设置实时时钟时必须确保输入有效的日期、时间。

- LD

  若 *EN* 值为 1，则执行 READ_RTC、SET_RTC 指令，否则不执行。

- IL

  若 CR 值为 1，则执行 READ_RTC、SET_RTC 指令，否则不执行。

  READ_RTC、SET_RTC 指令的执行不影响 CR 值。

7.1.3 RTC_R（读实时时钟）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	RTC_R			☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
IL	RTC_R	RTC_R*FMT, WEEK, SECOND, MINUTE, HOUR, DAY, MONTH, YEAR, CENTURY*	U	

参数	输入/输出	数据类型	允许的内存区
FMT	输入	BYTE	L、M、V、常量
WEEK	输出	BYTE	L、M、V
SECOND	输出	BYTE	L、M、V
MINUTE	输出	BYTE	L、M、V
HOUR	输出	BYTE	L、M、V
DAY	输出	BYTE	L、M、V
MONTH	输出	BYTE	L、M、V
YEAR	输出	BYTE	L、M、V
CENTURY	输出	BYTE	L、M、V

下表详细描述了各个参数的作用。

参数	描　述
EN	使能端。
FMT	输出格式，0 表示十进制，1 表示 BCD 码。
WEEK	星期，范围：1~7，其中 1 代表星期一，7 代表星期日。
SECOND	秒，范围：0~59
MINUTE	分，范围：0~59
HOUR	小时，范围：0~23
DAY	日，范围：1~31
MONTH	月，范围：1~12
YEAR	年，范围：0~99
CENTURY	世纪，固定为 20

RTC_R 用于从实时时钟中读取当前的日期和时间值并写入各个输出参数。

FMT 指定了各参数的格式， 若 FMT 为 0， 则为十进制；若 FMT 为 1， 则为 BCD 码。

- LD

  若 *EN* 值为 1，则执行 RTC_R 指令，否则不执行。

- IL

  若 CR 值为 1，则执行 RTC_R 指令，否则不执行。

  RTC_R 指令的执行不影响 CR 值。

7.1.4 RTC_W（写实时时钟）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	RTC_W			☑ K5（CPU504 除外） ☑ K2 ☑ KS ☑ KW ☑ HP ☑ MK ☑ K6
IL	RTC_W	RTC_W *FMT, WEEK, SECOND, MINUTE, HOUR, DAY, MONTH, YEAR, CENTURY*	U	

参数	输入/输出	数据类型	允许的内存区
FMT	输入	BYTE	L、M、V、常量
WEEK	输入	BYTE	L、M、V 、常量
SECOND	输入	BYTE	L、M、V、常量
MINUTE	输入	BYTE	L、M、V、常量
HOUR	输入	BYTE	L、M、V、常量
DAY	输入	BYTE	L、M、V、常量
MONTH	输入	BYTE	L、M、V、常量
YEAR	输入	BYTE	L、M、V、常量
CENTURY	输入	BYTE	L、M、V、常量

下表详细描述了各个参数的作用。FMT,WEEK,SECOND,MINUTE,HOUR,DAY,MONTH,YEAR,CENTURY 参数必须同时为常量或同时为变量。

参数	描　述
EN	使能端。
FMT	输出格式，0 表示十进制，1 表示 BCD 码。
WEEK	星期，范围：1~7，其中 1 代表星期一，7 代表星期日。
SECOND	秒，范围：0~59
MINUTE	分，范围：0~59
HOUR	小时，范围：0~23
DAY	日，范围：1~31
MONTH	月，范围：1~12
YEAR	年，范围：0~99
CENTURY	世纪，固定为 20

RTC_W 用于根据各参数指定的时间来修改实时时钟。

FMT 指定了各参数的格式， 若 FMT 为 0， 则为十进制；若 FMT 为 1， 则为 BCD 码。

- LD

  若 *EN* 值为 1，则执行 RTC_R 指令，否则不执行。

- IL

  若 CR 值为 1，则执行 RTC_R 指令，否则不执行。

  RTC_R 指令的执行不影响 CR 值。

第八章 高速计数器功能的使用

本章主要介绍 Kinco-K 系列 PLC 的高速计数功能，内容包括了高速计数工作模式、输入信号、状态寄存器、控制寄存器、PV 中断等。

8.1 功能概述

CPU 型号	单相		AB 相/双相	
	HSC0 和 HSC1	HSC2 和 HSC3	HSC0 和 HSC1	HSC2 和 HSC3
K5 系列	60K	/	20K	/
K2 系列（K205）	50K	20K	50K	10K
K2 系列（K204ET）	200K	200K	200K	200K
K2 系列（K209EA）	200K	20K	100K	10K
K2 系列（K209M）	200K	/	200K	/
KS 系列（KS105）	200K	200K	200K	200K
KS 系列（KS101M）	200K	/	200K	/
KW 系列	200K	200K	200K	200K
MK 系列	50K	50K	50K	50K
K6 系列	200K	20K	200K	10K

Kinco-K 系列 PLC 的提供了最多 4 路高速计数器，编号为 HSC0 至 HSC3，最高计数频率视具体型号而定。

高速计数器具有多种工作模式，可以进行单相、双相（Up/Down）、AB 相（1 倍频和 4 倍频）等计数。通过选择不同的工作模式可以实现针对测量传感器、旋转编码器、光栅尺等高速输入信号的测定。在使用高速计数器之前，必须用 HDEF 指令或使用 HSC 向导（K5 不支持）为其指定一种工作模式。所有的高速计数器在相同的工作模式下均具有相同的功能。

所有高速计数器（K5 系列除外）均允许指定最大 32 个预置值（PV），每个 PV 值均支持“计数

值（CV）=预置值（PV）”中断。PV 值可以指定为相对值或者绝对值方式，若选择为相对值方式，那么“计数值=预置值”中断允许选择循环发生。

8.2 高速计数器工作模式和输入信号

高速计数器的输入信号包括如下几种：时钟（即输入脉冲）、方向、启动和复位信号。

在不同的工作模式下，所需要的输入信号也有所不同。下面各表详细描述了各个高速计数器所支持的工作模式及其输入信号的分配。

HSC 0				
模式	**描述**	**I0.1**	**I0.0**	**I0.5**
0	带内部方向控制的单相增/减计数器 方向控制位：SM37.3	时钟		
1			复位	
2			复位	启动
3	带外部方向控制的单相增/减计数器	时钟		方向
4			复位	方向
6	带增/减计数时钟输入的双相计数器	时钟（减）	时钟（增）	
9	A/B 相正交计数器	时钟 A 相	时钟 B 相	

HSC1					
模式	**描述**	**I0.4**	**I0.6**	**I0.3**	**I0.2**
0	带内部方向控制的单相增/减计数器 方向控制位：SM47.3			时钟	
1		复位			
2		复位	启动		
3	带外部方向控制的单相增/减计数器			时钟	方向
4		复位			方向
6	带增/减计数时钟输入的双相计数器			时钟（减）	时钟（增）
7		复位			
9	A/B 相正交计数器			时钟 A	时钟 B
10		复位			

HSC 2			
模式	**描述**	**I0.4**	**I0.5**
0	带内部方向控制的单相增/减计数器。 方向控制位：SM57.3		时钟
9	A/B 相正交计数器	时钟 B 相	时钟 A 相

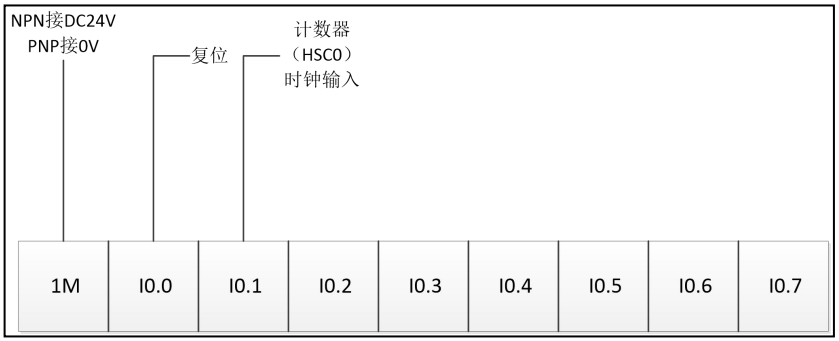
HSC 3			
模式	**描述**	**I0.6**	**I0.7**
0	带内部方向控制的单相增/减计数器。 方向控制位：SM127.3		时钟
9	A/B 相正交计数器	时钟 B 相	时钟 A 相

8.3 高速计数值范围

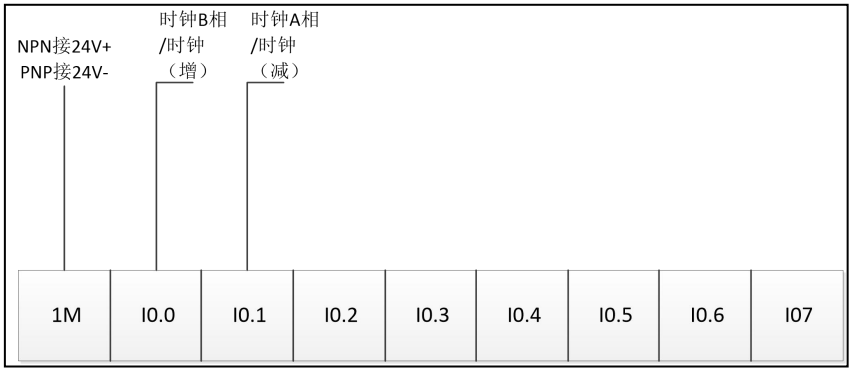
高速计数器的计数值为 DINT 型双整数，其范围为：-2,147,483,648 ~ +2,147,483,647。当计数值超出此范围时，则产生上溢或下溢现象。 所谓产生上溢，就是计数值从+2,147,483,647 跳转为-2,147,483,648，并继续计数；而当产生下溢时，计数值从-2,147,483,648 跳转为+2,147,483,647，并继续计数。

8.4 高速计数器输入端接线

高速计数器的输入端接线和其使用的计数器及工作模式有关，下面两个图以 HSC0 为例介绍。



计数器 HSC0 单相脉冲输入、工作模式 1



计数器 HSC0 双相/AB 相脉冲输入、工作模式 6/9

8.5 高速计数器的时序图

为了让用户更好地理解高速计数器，如下各图举例详细描述了高速计数器的工作时序。

图8-1和图8-2适用于所有具有复位输入信号和启动输入信号的工作模式。图8-3至图8-6描述了高速计数器各个工作模式的时序图。

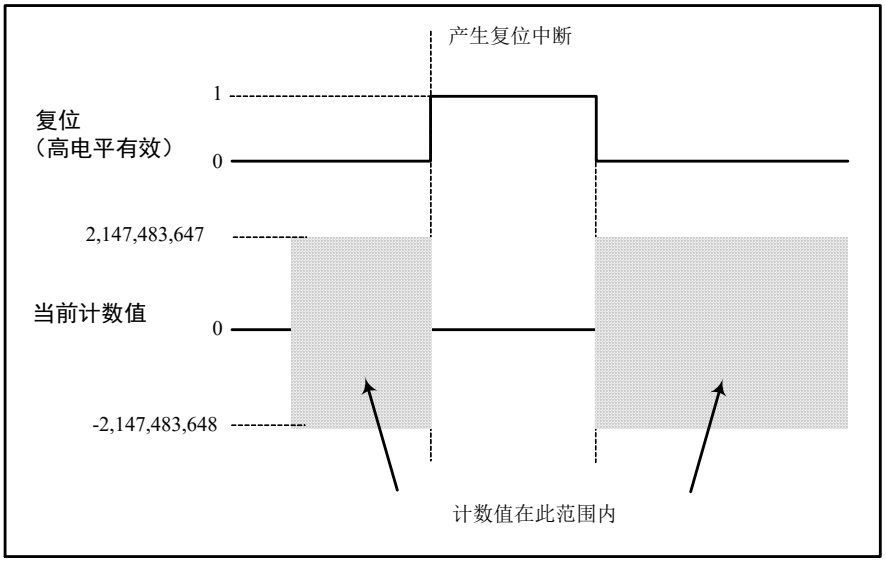


图 8-1 有复位信号无启动信号的高速计数器时序图

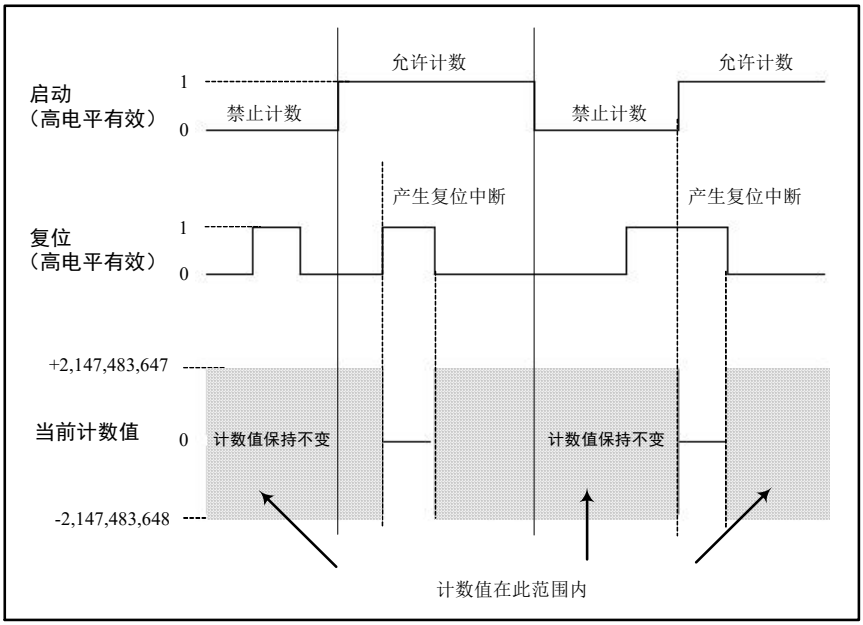


图 8-2 有复位信号和启动信号的高速计数器时序图

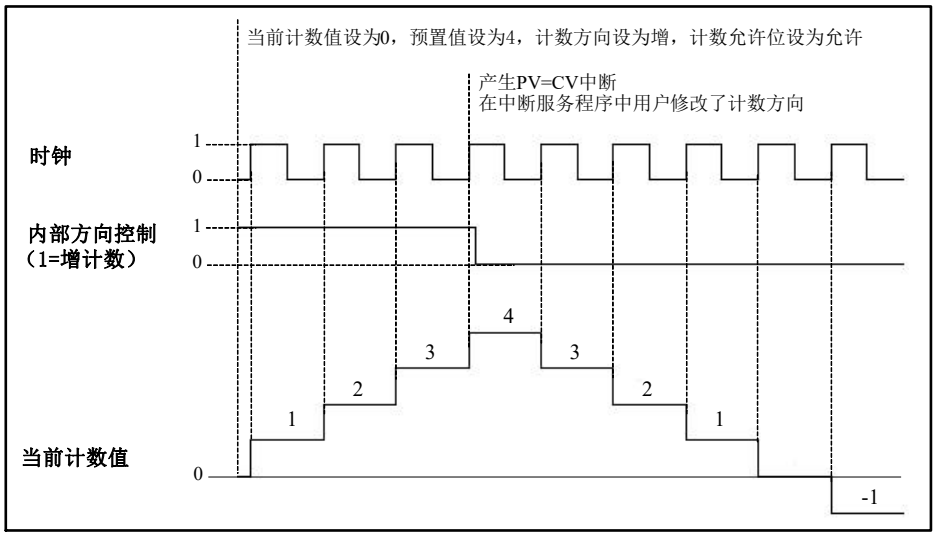


图 8-3 模式 0、1、2 的时序图

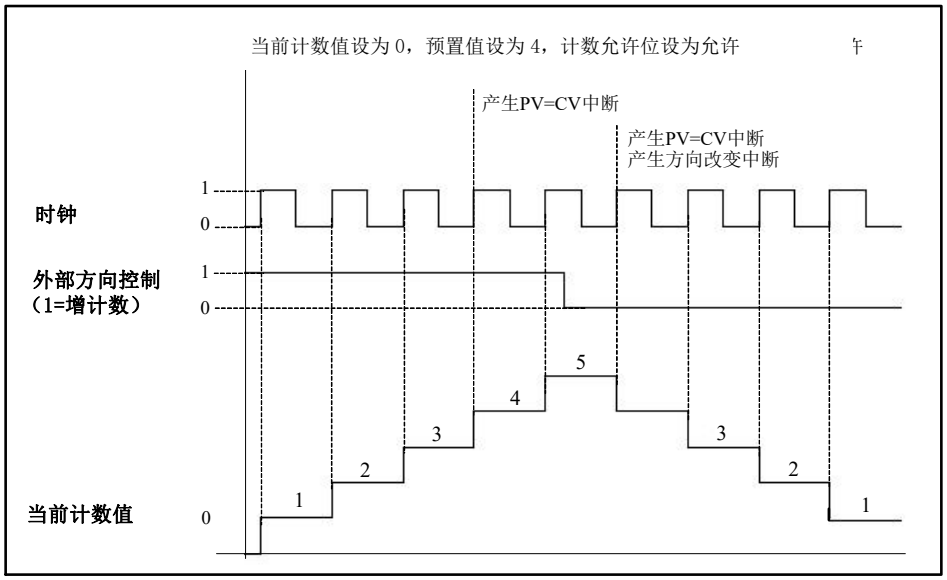


图 8-4 模式 3、4 的时序图

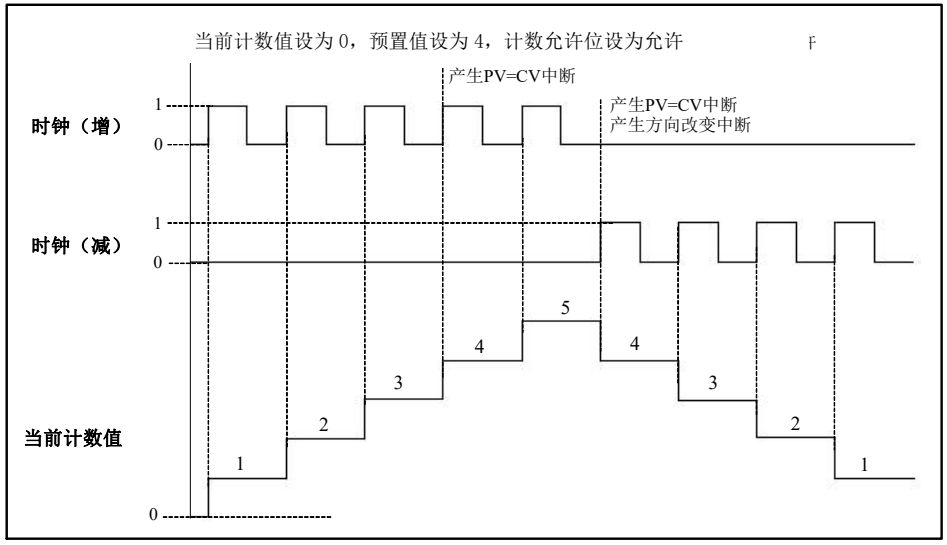


图 8-5 模式 6、7 的时序图

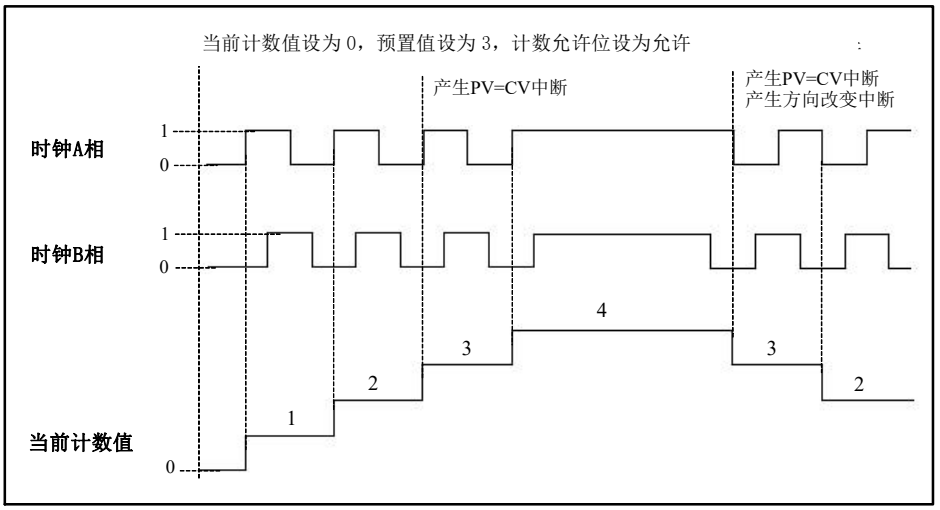


图 8-6 模式 9、10 的时序图（正交，1x）

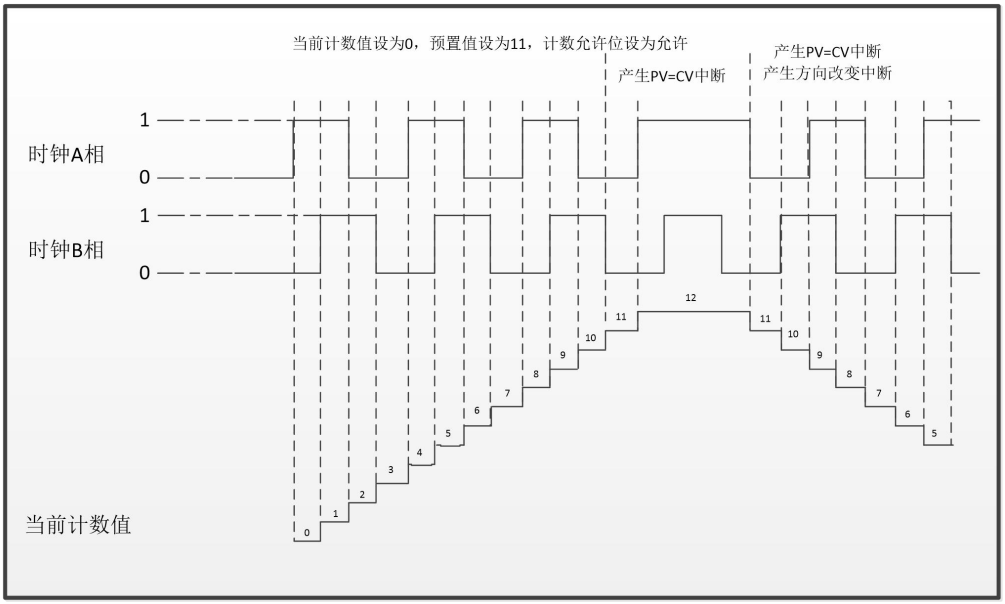


图 8-7 模式 9、10 的时序图（正交，4x）

8.6 控制寄存器和状态寄存器

➢ **控制寄存器**

在 SM 区中为每个高速计数器均提供了如下控制寄存器用于存放其配置数据：一个控制字节（8 位），当前值和预置值各一个（均为 32 位有符号双整数）。当前值指定计数的起始值，若将当前值写入高速计数器，那么高速计数器就会立即从这个数值开始计数。下表详细描述了这些寄存器。

HSC0	HSC1	HSC2	HSC3	描述
SM37.0	SM47.0	SM57.0	SM127.0	复位信号的有效电平：0=高电平；1=低电平
SM37.1	SM47.1	SM57.1	SM127.1	启动信号的有效电平：0=高电平；1=低电平
SM37.2	SM47.2	SM57.2	SM127.2	正交计数器速率：0=1x 速率；1=4x 速率*
SM37.3	SM47.3	SM57.3	SM127.3	计数方向：0=减计数；1=增计数。
SM37.4	SM47.4	SM57.4	SM127.4	是否向 HSC 中写入计数方向：0=否；1=是。
SM37.5	SM47.5	SM57.5	SM127.5	是否向 HSC 中写入新预置值：0=否；1=是。
SM37.6	SM47.6	SM57.6	SM127.6	是否向 HSC 中写入新当前值：0=否；1=是。
SM37.7	SM47.7	SM57.7	SM127.7	是否允许该高速计数器：0=禁止；1=允许。
HSC0	**HSC1**	**HSC2**	**HSC3**	**描述**
SMD38	SMD48	SMD58	SMD128	新的当前值
SMD42	SMD52	SMD62	SMD132	新的预置值

另外除 K5 系列 CPU 外的所有高速计数器均允许指定最大 32 个预置值（PV），其控制寄存器描述如下：

HSC0	HSC1	HSC2	HSC3	描述
SM141.0	SM151.0	SM161.0	SM171.0	是否使用多段预置值：0=否；1=是
SM141.1	SM151.1	SM161.1	SM171.1	预置值是相对值还是绝对值：0 =绝对；1=相对
SM141.2	SM151.2	SM161.2	SM171.2	预置值比较（“CV=PV”）中断是否循环产生： 0=否；1=是。 注意：只有相对值方式才允许设定为循环产生。
SM141.3	SM151.3	SM161.3	SM171.3	保留
SM141.4	SM151.4	SM161.4	SM171.4	是否更新段数及预置值：0=否；1=是
SM141.5	SM151.5	SM161.5	SM171.5	是否复位中断变量：0=是；1=否

SM141.6	SM151.6	SM161.6	SM171.6	保留
SM141.7	SM151.7	SM161.7	SM171.7	保留
HSC0	**HSC1**	**HSC2**	**HSC2**	**描述**
SMW142	SMW152	SMW162	SMW172	预置值表的起始位置（用相对于 VB0 的字节偏移来表示），必须为奇数。

需要注意的是，控制字节中并非所有的控制位都适用于所有的工作模式。比如，“计数方向”和“是否向 HSC 中写入计数方向”这两个控制位就只用于模式 0、1 和 2（带内部方向控制的单相增/减计数器），若高速计数器所用的工作模式是采用外部的方向控制信号，那么这两个控制位就会被忽略。

控制字节的当前值和预置值上电后的缺省值均为 0。

- **状态寄存器**

每个高速计数器都在 SM 区中提供了状态寄存器用于指明高速计数器当前的状态信息。

HSC0	HSC1	HSC2	HSC3	描述
SM36.0	SM46.0	SM56.0	SM126.0	保留
SM36.1	SM46.1	SM56.1	SM126.1	保留
SM36.2	SM46.2	SM56.2	SM126.2	保留
SM36.3	SM46.3	SM56.3	SM126.3	多段 PV 值表设置是否有错误：0=否，1=是。
SM36.4	SM46.4	SM56.4	SM126.4	保留
SM36.5	SM46.5	SM56.5	SM126.5	当前计数方向：0=减；1=增。
SM36.6	SM46.6	SM56.6	SM126.6	当前计数值是否等于预置值：0=否；1=是。
SM36.7	SM46.7	SM56.7	SM126.7	当前计数值是否大于预置值：0=否；1=是。
HSC0	HSC1	HSC2	HSC3	描述（K5 系列不支持）
SMB140	SMB150	SMB160	SMB170	正在运行的 PV 值段序号（从 0 开始）。

- **读取高速计数器的当前计数值**

高速计数器的计数值是 32 位的双整数，并且是只读的。

Kinco-K 系列在内存区域中提供了高速计数器区（HC 区）用于存放各高速计数器的计数值。这个区域的寻址方式是内存区域类型（HC）和高速计数器的编号。比如，HC2 表示 HSC2 的当前计数值。请

参阅 3.6.2.1 直接地址表示格式了解更多信息。下图描述了这种寻址方式。

图8-7 读取高速计数器的计数值示例

8.7 高速计数器中断

各高速计数器均支持中断：当计数值等于预置值时就会产生“PV（预置值）=CV（计数值）”中断；若是使用外部复位信号的高速计数器模式，那么在复位时就会产生“外部复位”中断；若是使用外部方向控制信号的计数器模式，那么在计数方向改变时就会产生“方向改变”中断。

高速计数器的中断事件列表如下：

事件号	中断描述	分类
191	HSC3 使用多段 PV 值时第 32 个“CV=PV”	I/O 中断
…	…（依次加 1）	
161	HSC3 使用多段 PV 值时第 2 个“CV=PV”	
160	HSC3 使用多段 PV 值时第 1 个“CV=PV”	
159	HSC2 使用多段 PV 值时第 32 个“CV=PV”	
…	…（依次加 1）	
129	HSC2 使用多段 PV 值时第 2 个“CV=PV”	
128	HSC2 使用多段 PV 值时第 1 个“CV=PV”	
127	HSC1 使用多段 PV 值时第 32 个“CV=PV”	
…	…（依次加 1）	
97	HSC1 使用多段 PV 值时第 2 个“CV=PV”	
96	HSC1 使用多段 PV 值时第 1 个“CV=PV”	
95	HSC0 使用多段 PV 值时第 32 个“CV=PV”	

…	…（依次加 1）	
65	HSC0 使用多段 PV 值时第 2 个“CV=PV”	
64	HSC0 使用多段 PV 值时第 1 个“CV=PV”	
18	HSC0 使用单个 PV 值时 CV=PV（当前值=预设值）	
17	HSC0 输入方向改变	
16	HSC0 外部复位	
15	HSC1 使用单个 PV 值时 CV=PV（当前值=预设值）	
14	HSC1 输入方向改变	
13	HSC1 外部复位	
12	HSC2 使用单个 PV 值时 CV=PV（当前值=预设值）	
9	HSC3 使用单个 PV 值时 CV=PV（当前值=预设值）	

8.8 PV 中断的使用

Kinco-K 系列 PLC 的 PV 设定方式有两种：单 PV 设定方式和多 PV 设定方式。

单 PV 设定方式仅允许每个高速计数器设定 1 个 PV 值。

多 PV 设定方式允许每个高速计数器设定最多 32 个 PV 值，允许选择 PV 之间的关系是相对值或者绝对值。相对值的方式允许“CV=PV”中断循环产生。

Kinco-K 系列 PLC 支持的 PV 设定方式如下表所示：

型号	PV 设定方式		
	单 PV 设定	多 PV 设定（32 个 PV 值）	
		绝对值（不支持中断循环）	相对值（支持中断循环）
K5 系列	支持	不支持	
K2 系列	支持	支持	
KS 系列			
KW 系列			
MK 系列			
K6 系列			

下面将以 HSC0 为例来对 PV 值的功能和设定方法进行详细的描述。

8.8.1 预置值（PV 值）设定

- **如何选择单段和多段 PV 值**

每个高速计数器的控制寄存器里都提供了一个控制位，用于选择是否使用多段预置值。

HSC0 的这个控制位是 SM141.0。

若 SM141.0 为 0，则表示采用单 PV 值方式，与 K5 的用法一致：SMD42 指定了新 PV 值，SM37.5 指明是否使用这个新的 PV 值。

若 SM141.0 为 1，则表示采用多段 PV 值方式，此时 SM37.5、SMD42 无效。各 PV 值存放于 PV 值表中（SMW142 为表起始地址），SM141.4 指明是否使用 PV 值表中的数据。若 SM141.4 为 1，则表示本次启动后，高速计数器时将采用 PV 值表中的数据。若 SM141.4 为 0，则表示本次启动后，高速计数器将采用上一次的 PV 值数据，而忽略 PV 值表中的数据。

- **多段 PV 值表**

若使用多段 PV 值，那么各 PV 值将采用 PV 值表中的数据。

每个高速计数器的控制寄存器里都提供一个控制字，用于存放 PV 值表的起始地址，**表的起始地址必须为 V 区中的奇数地址，例如 301（表示 VB301）。**

PV 值表的格式如下。

字节偏移[1]	数据类型	描述
0	BYTE	PV 值个数
1	DINT	第 1 个 PV 值
5	DINT	第 2 个 PV 值
…	DINT	…

- 所有偏移量均是相对于表起始位置的偏移字节数。
- 当采用相对值方式时，PV 值的数学绝对值必须大于 1，否则 PLC 认为段数到此结束，并以此统计 PV 值个数（优先于个数设定值）；
- 当采用绝对值方式时，相邻两个 PV 值之间的差值的数学绝对值必须大于 1，否则 PLC 认为段数到此结束，并以此统计 PV 值个数（优先于个数设定值）；

- 用户设定 PV 值时需要注意：“CV=PV”中断必须依次产生。也就是说，当计数值达到第 1 个 PV 值并产生中断后，接下来 PLC 将会与第 2 个 PV 值进行比较，依次类推。
- PV 值设置必须合理。以相对值为例，当增计数时，PV 值必须大于 0，否则该值对应的“CV=PV”中断也许永远不会产生；当减计数时，PV 值必须小于 0，否则该值对应的“CV=PV”中断也许永远不会产生。

8.8.2 相对值和绝对值方式

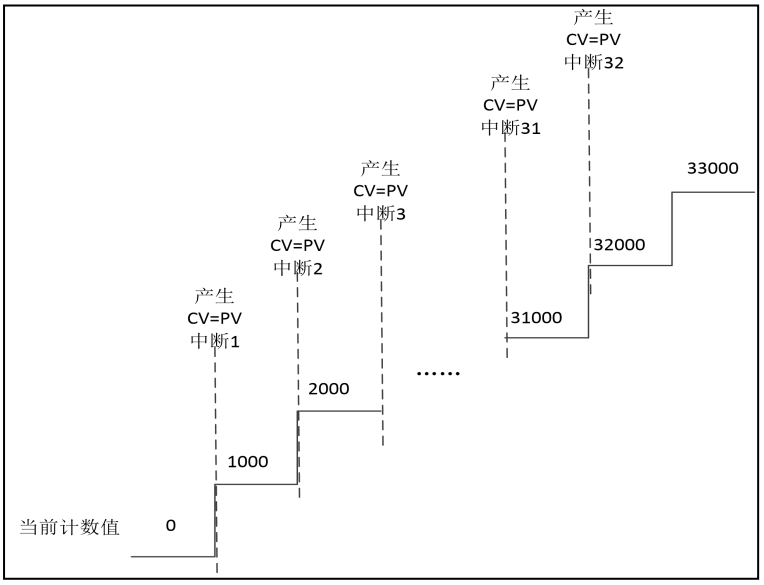
每个高速计数器的控制寄存器里都提供了一个控制位，用于选择 PV 值是相对值或者绝对值方式。HSC0 的这个控制位是 SM141.1。

- **绝对值方式**

若 SM141.1 为 0，则表示 PV 值是绝对值方式。当计数值等于 PV 值时，将会产生相应的“CV=PV”中断。例如，高速计数器启动时的计数值是 0，若设定 32 个 PV 值，依次是 1000、2000、3000 ……32000，那么计数值达到 1000 时，将产生第 1 个“CV=PV”中断；当计数值达到 2000 时，将产生第 2 个“CV=PV”中断；后面依次类推。

计数器	CV（计数值）=PV（预置值）中断标记				
	第 1 段	第 2 段	……	第 N 段	第 32 段
HC0	1000	2000	……	N*1000	32000

由表中可以看出 HC0 计数值等于预置值时就会产生一次中断，可以理解为运动控制中的绝对定位的过程，HC0 计数值是相对于零点的值。如下图：

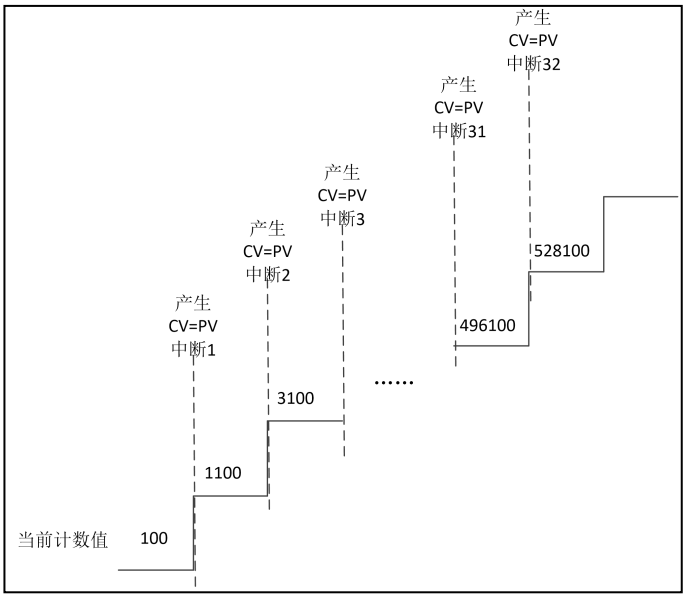


- **相对值方式**

若 SM141.1 为 1，则表示 PV 值是相对值方式，若计数器以当前的计数值为基准，继续计数使得差值等于 PV 值时，将会产生相应的“CV=PV”中断。例如，若设定 32 个 PV 值，分别为 1000、2000、3000 ……32000，而且在高速计数器启动时的计数值是 100，那么当计数值分别达到 1100、3100、6100……528100 时，将分别产生“CV=PV”中断。

计数器	CV（计数值）=PV（预置值）中断标记				
	第 1 段	第 2 段	……	第 N 段	第 32 段
HC0	1100	3100	……	100+N*（N+1）*1000/2	528100

由表中可以看出 HC0 计数值等于前面 N 次预置值之和时就会产生一次中断，可以理解为运动控制中的相对定位的过程，HC0 下一次产生中断的计数值是相对预置值的增加。如下图：



- **相对值方式的“CV=PV”中断循环产生**

只有 PV 值是相对值方式时，才允许设定为循环产生中断，否则无效。

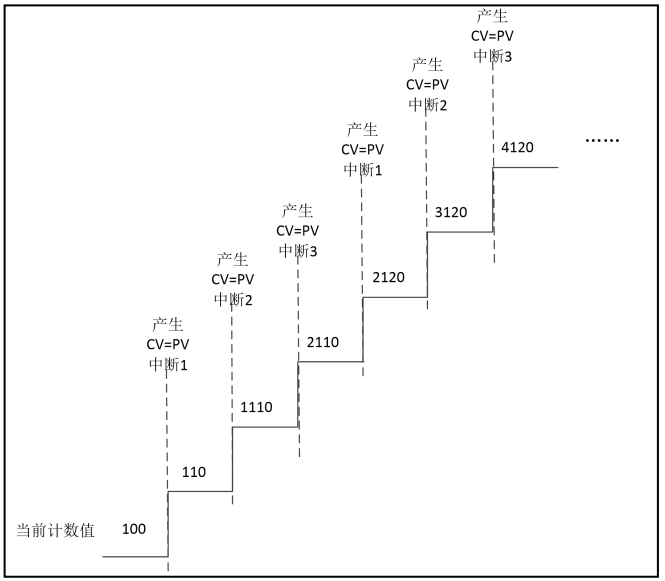
若 SM141.0 为 0，则表示“CV=PV”中断只产生一次。当所有 PV 值对应的中断都发生完成后就会停止。若要继续产生，则必须修改相应的寄存器值并再次调用 HSC 指令。

若 SM141.0 为 1，则表示“CV=PV”中断会循环产生。当最后一个 PV 值对应的中断发生完成时，PLC 将以当前的计数值为基准，与各个 PV 值再次相加，得到新的中断所需数值，然后继续与计数值进行比较，产生相应的“CV=PV”中断。这个过程会一直循环进行，永不停止。

例如，若设定 3 个 PV 值，分别为 10、1000、1000，而且在高速计数器启动时的计数值是 100，则各个中断每次产生所需数值如下：

当前计数值	中断次数	第 1 个值	第 2 个值	第 3 个值
100	第 1 次	110	1110	2110
2110	第 3 次	2120	3120	4120
4120	第 6 次	4130	5130	6130
…	第 n 次	…	…	…

如下图：

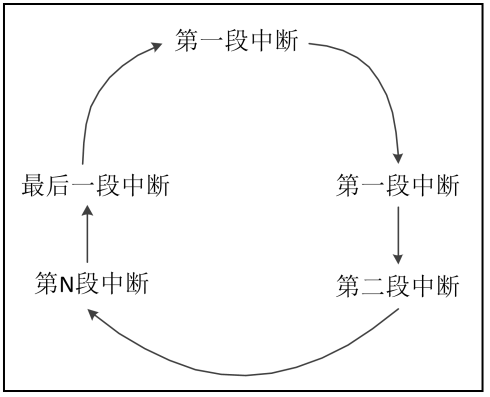


中断循环仅适用于相对值计数模式。在中断循环模式下，中断结束之后将自动重新开始。该模式特别 适用于以下场合：

（1）连续往复运动；

（2）按指定脉冲产生周期中断。

中断循环的顺序如下图：



8.8.3 CV=PV 中断编号

当采用单 PV 值方式时，其他系列 PLC 高速计数器完全兼容 K5，包括“CV=PV”中断的编号与 K5 中的一致。

高速计数器	中断事件号	描述
HSC0	18	HSC0 使用单个 PV 值时 CV=PV（当前值=预设值）
	17	HSC0 输入方向改变
	16	HSC0 外部复位
HSC1	15	HSC1 使用单个 PV 值时 CV=PV（当前值=预设值）
	14	HSC1 输入方向改变
	13	HSC1 外部复位
HSC2	12	HSC2 使用单个 PV 值时 CV=PV（当前值=预设值）
	11—10	保留
HSC3	9	HSC3 使用单个 PV 值时 CV=PV（当前值=预设值）
	8—5	保留

当采用多段 PV 值方式时，高速计数器为 32 个 PV 值均分配了一个新的中断编号，如下表。

高速计数器	中断事件号	描述
HSC0	64	第 1 个 PV 值的“CV=PV”中断
	65	第 2 个 PV 值的“CV=PV”中断
	…	…（依次加 1）
	95	第 32 个 PV 值的“CV=PV”中断
HSC1	96	第 1 个 PV 值的“CV=PV”中断
	97	第 2 个 PV 值的“CV=PV”中断
	…	…（依次加 1）
	127	第 32 个 PV 值的“CV=PV”中断
HSC2	128	第 1 个 PV 值的“CV=PV”中断
	129	第 2 个 PV 值的“CV=PV”中断
	…	…（依次加 1）
	159	第 32 个 PV 值的“CV=PV”中断

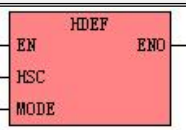
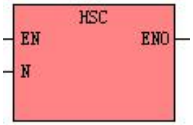
HSC3	160	第 1 个 PV 值的“CV=PV”中断
	161	第 2 个 PV 值的“CV=PV”中断
	…	…（依次加 1）
	191	第 32 个 PV 值的“CV=PV”中断

8.9 HDEF（高速计数器定义）、HSC（高速计数器）指令

高速计数器的运行不受 CPU 扫描周期的影响，因此可以用于对高速的脉冲输入进行计数。

HDEF、HSC 指令位于【指令集】->【计数器】组中。

- 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	HDEF			☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
	HSC			
IL	HDEF	HDEF HSC, MODE	U	
	HSC	HSC *N*		

参数	输入/输出	数据类型	描述
HSC	输入	INT	常量（高速计数器编号）
MODE	输入	INT	常量(0-10，高速计数器的工作模式)
N	输入	INT	常量（高速计数器编号）

HDEF 指令的作用是：为参数 *HSC* 指定的高速计数器定义一种工作模式 *MODE*。

HSC 指令的作用是：先读取 HDEF 指令指定的工作模式以及 SM 区中相应控制寄存器的值并配置高速计数器 *N* 的特性，然后启动高速计数器 *N*。

注意：只在需要的时候触发执行一次 HSC 指令就可以配置并启动高速计数器！建议使用上升沿、下降沿之类的条件来触发 HSC 指令，若触发条件长时间保持为“1”（比如使用 SM0.0），则高速计数器会一直被初始化，无法正常使用。

- LD

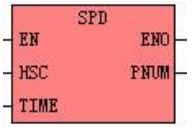
  若 *EN* 值为 1，则执行 HDEF、HSC 指令，否则不执行。

- IL

  若 CR 值为 1，则执行 HDEF、HSC 指令，否则不执行。HDEF、HSC 的执行不影响 CR 值。

8.10 SPD（脉冲密度）指令

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	适用于
LD	SPD			☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
IL	SPD	SPD HSC, TIME, PNUM	U	

参数	输入/输出	数据类型	允许的内存区
HSC	输入	INT	常量（高速计数器编号）
TIME	输入	WORD	I、Q、M、V、L、SM、常量
PNUM	输出	DINT	Q、M、V、L、SM

SPD 指令用于计算高速计数器 *HSC* 在时间 *TIME*（单位：ms）内接收到的脉冲个数，并将结果写入 *PNUM* 中。

- LD

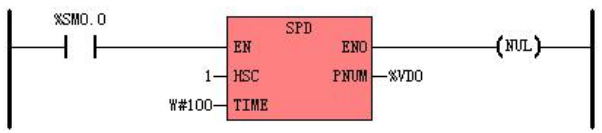
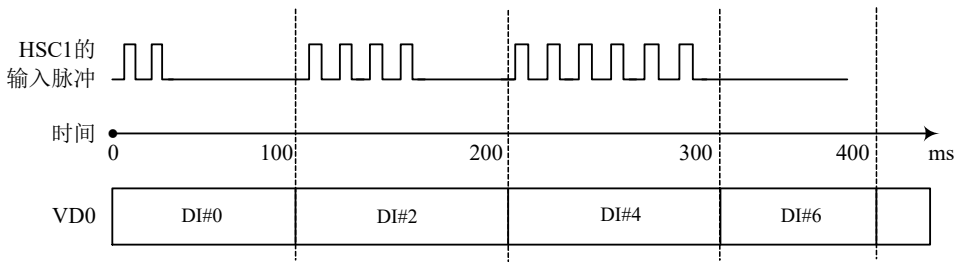
  如果 *EN* 为 1，则该指令被执行。

- IL

  如果 CR 值为 1，则该指令被执行。

  该指令的执行不影响 CR 值。

➢ 指令使用举例

<table>
<tr><td>LD</td><td> 
</td><td>SM0.0 恒为 1，因此 SPD 指令总是执行：对 HSC1 的脉冲计数值进行计算，每 100ms 就将这个时间内计算的脉冲个数值赋给 VD0。</td></tr>
<tr><td>结果</td><td>结果举例如下： 
</td><td></td></tr>
</table>

8.11 高速计数器的使用方法

高速计数器的使用方法有两种：

1、使用相关指令进行编程：这种方式需要在程序中设置相应控制寄存器，并调用 HDEF 指令和 HSC 指令进行编程，适合所有的 Kinco-K 系列 PLC,适用于单段 PV 设定方式和多段 PV 设定方式（K5 系列仅支持单段 PV 设定方式）。

2、使用 HSC 向导设置：这种方式简单直观只需要按向导中提供的内容进行勾选设置即可，适合除 K5 系列之外的所有 PLC,适用于单段 PV 设定方式和多段 PV 设定方式，建议使用 K5 系列之外的 PLC 使用向导的方式进行设置，简单方便节省编程时间。

8.11.1 使用相关指令进行编程

用户可以按照如下步骤来编程使用一个高速计数器：

1）配置该高速计数器的控制字节，并指定当前值（也就是计数的起始值）和预置值；

2）使用 HDEF 指令来定义一个高速计数器及其工作模式；

3）使用 HSC 指令来配置并启动高速计数器；

4）（可选）使用 ATCH 指令为高速计数器中断连接相应的中断服务程序；

注意：在 CPU 进入 RUN 模式后，只允许为每个高速计数器执行一次 HDEF 指令。

下面将以 HSC0 为例来详细说明如何编程使用高速计数器。

建议用户在工程中尽量编写单独的初始化子程序，其中包含有 HDEF 指令和其它的初始化指令，这样可以使整个用户工程具有良好的结构。

8.11.1.1 使用高速计数器

- **使用高速计数器**

下面将以模式 9 为例进行描述，但是描述的步骤在原理上同样适用于其它工作模式，用户根据实际需要稍作调整即可。

1) 根据期望的操作来设置控制字节 SMB37。

例如，（1x 计数方式），SMB37 = B#16#F0 表明了：

- 允许HSC0计数；
- 允许向HSC0中写入新的当前值和预置值
- 设置启动信号和复位信号为高电平有效。

2) 将期望的当前值（32 位双整数，也就是计数的起始值）赋给 SMD38。

3) 将期望的预置值（32 位双整数）赋给 SMD42。

4) （可选）使用ATCH指令为“PV = CV”中断事件（事件号18）连接一个中断服务程序以实现对该中断事件的快速响应。

5) （可选）使用ATCH指令为“方向改变”中断事件（事件号17）连接一个中断服务程序以实现对该中断事件的快速响应。（部分模式不支持）

6) （可选）使用 ATCH 令为“外部复位”中断事件（事件号 16）连接一个中断服务程序以实现对该中断事件的快速响应。（部分模式不支持）

7) 执行 HDEF 指令，在 *HSC* 端输入为 0，*MODE* 端输入为 9（本例采用模式 9）；

8) 执行 HSC 指令来配置并启动 HSC0。

- **改变计数方向（模式 0、1、2）**

下面描述了如何来改变 HSC0 的计数方向（模式 0、1、2）。

1) 根据期望的操作来设置控制字节 SMB37。

例如，SMB37 = B#16#90 表明了：

- 允许计数；
- 向 HSC0 中写入新的计数方向；
- 设置计数方向为减计数。

2) 执行 HSC 指令来配置 HSC0。

- **改变当前值（适用于所有模式）**

下面描述了如何改变 HSC0 的当前值（也就是计数的起始值）。

1) 根据期望的操作来设置控制字节 SMB37。

SMB37 = B#16#C0 表明了：

- 允许计数；
- 允许向 HSC0 中写入新的当前值。

2) 将期望的当前值（32 位双整数）赋给 SMD38。

3) 执行 HSC 指令来配置 HSC0。

➢ **改变预置值（适用于所有模式）**

下面描述了如何改变 HSC0 的预置值。

1) 根据期望的操作来设置控制字节 SMB37。

SMB37= B#16#A0 表明了：

- 允许计数；
- 允许更新 HSC0 的预置值。

2) 将期望的预置值（32 位双整数）赋给 SMD42。

3) 执行 HSC 指令来配置 HSC0。

➢ **禁止高速计数器（适用于所有模式）**

下面描述了如何禁止 HSC0。

1) 根据期望的操作来设置控制字节 SMB37。

SMB37= B#16#00 表明了：禁止该高速计数器。

2) 执行 HSC 指令以使 CPU 禁止 HSC0。

注意：修改控制字节的值时（比如 HSC0 的 SMB37），该字节的 8 个位都会被修改，而每个位代表的含义不同，所以需要注意避免出现误修改某控制位的情况发生，比如初始设定 SMB37= B#16#84 允许高速计数 4 倍频计数，如果用户为了改变当前值而修改为 SMB37= B#16#C0，那就把计数倍数也改了！为避免这种情况，用户可以直接操作相应的功能位。

8.11.1.2 使用举例

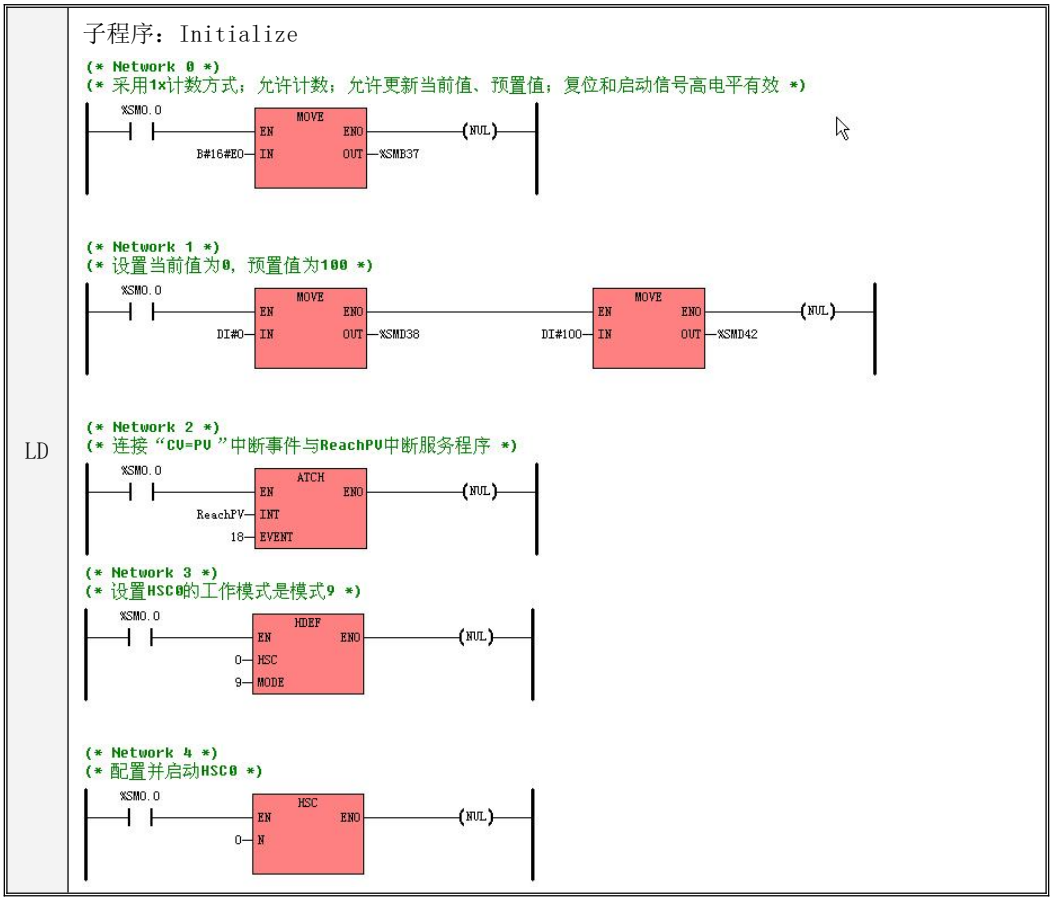
1、下面以 HSC0 为例来进行一个单段 PV 设定方式的编程。

示例中包含三段程序：主程序 MAIN，子程序 Initialize，中断服务程序 ReachPV。

主程序 MAIN：调用子程序 Initialize 对高速计数器 HSC0 进行初始化设置。

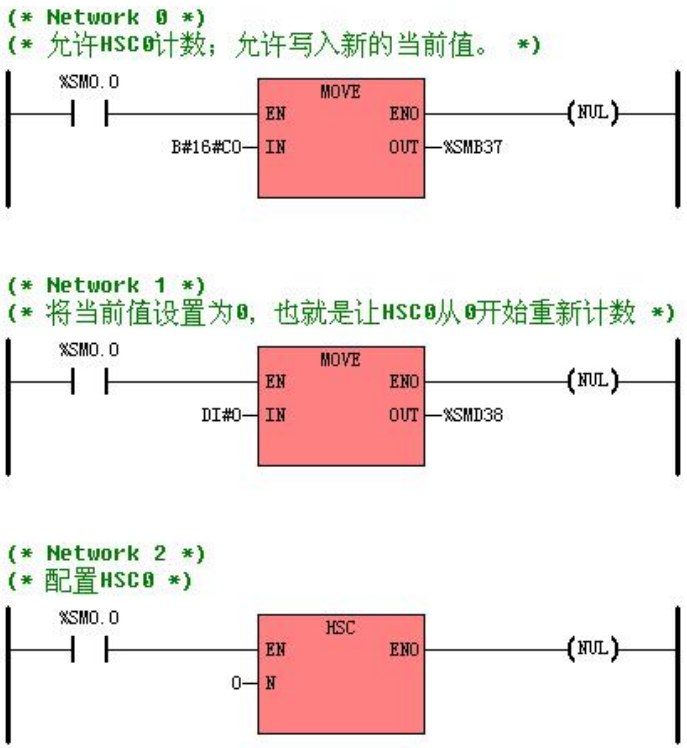
子程序 Initialize：设置计数器 HSC0 的工作模式、计数方式、当前值 CV、预置值 PV、连接 CV=PV 中断产生时连接的中断子程序等，并启动计数器。

中断服务程序 ReachPV：当 CV=PV 中断产生时执行此程序，将当前值 CV 清零重启计数器开始新的计数，当 CV=PV 时继续产生中断进入一个往复循环的过程。

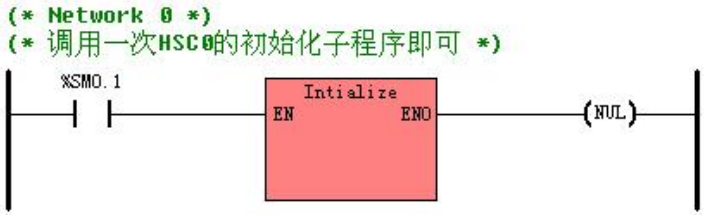


LD

中断服务程序：ReachPV



主程序：MAIN



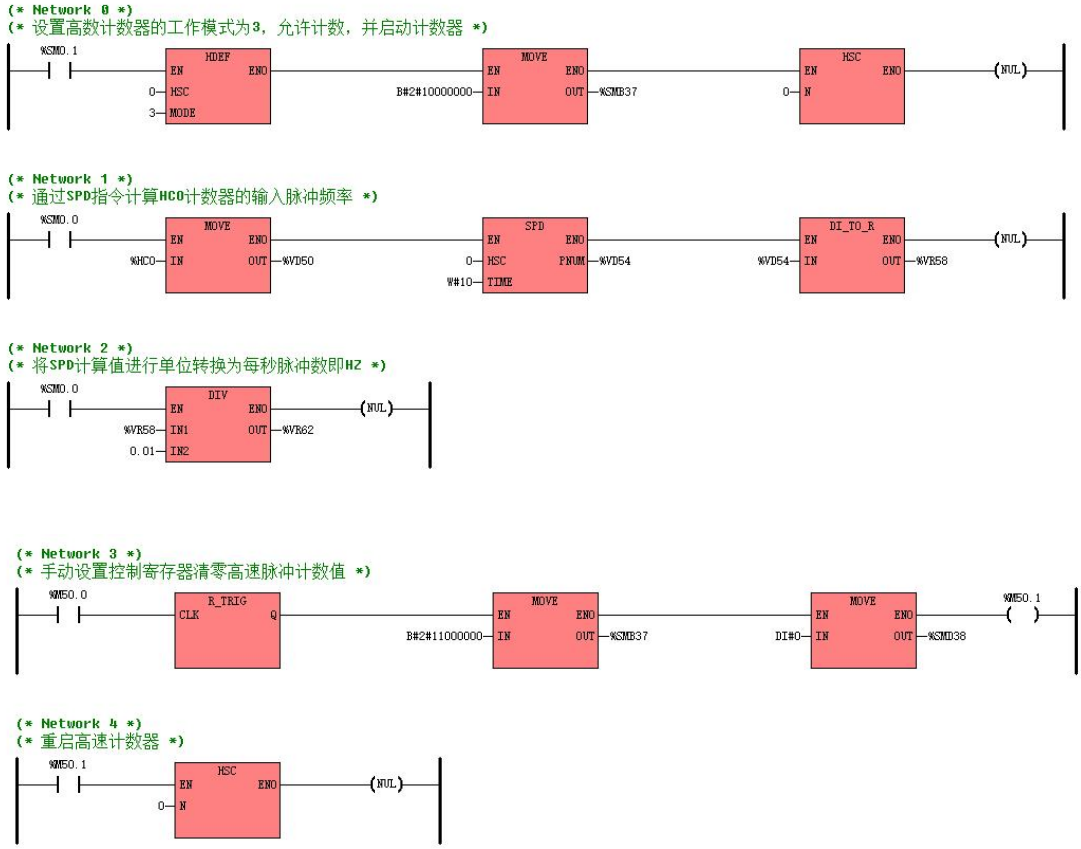
IL

```
子程序：Initialize

(* Network 0 *)
LD      %SM0.0
MOVE    B#16#E0, %SMB37      (* 配置控制字节 *)
MOVE    DI#0, %SMD38         (* 设置当前值为 0 *)
MOVE    DI#100, %SMD42       (* 设置预置值为 100 *)
ATCH    ReachPV, 18          (* 连接“CV=PV”中断与 ReachPV 中断程序 *)
HDEF    0, 9                 (* 设置 HSC0 的工作模式是模式 9 *)
HSC     0                    (* 配置并启动 HSC0 *)

中断服务程序：ReachPV

(* Network 0 *)
LD      %SM0.0
MOVE    B#16#C0, %SMB37      (* 允许 HSC0 计数；允许更新当前值 *)
MOVE    DI#0, %SMD38         (* 将当前值设置为 0 *)
HSC     0                    (* 配置 HSC0 *)


主程序：MAIN

LD      %SM0.1
CAL     Intialize            (* 调用一次 HSC0 的初始化子程序即可 *)
```

2、下面还是以 HSC0 为例计算高速脉冲输入频率和手动清零计数值。

LD

主程序：Main



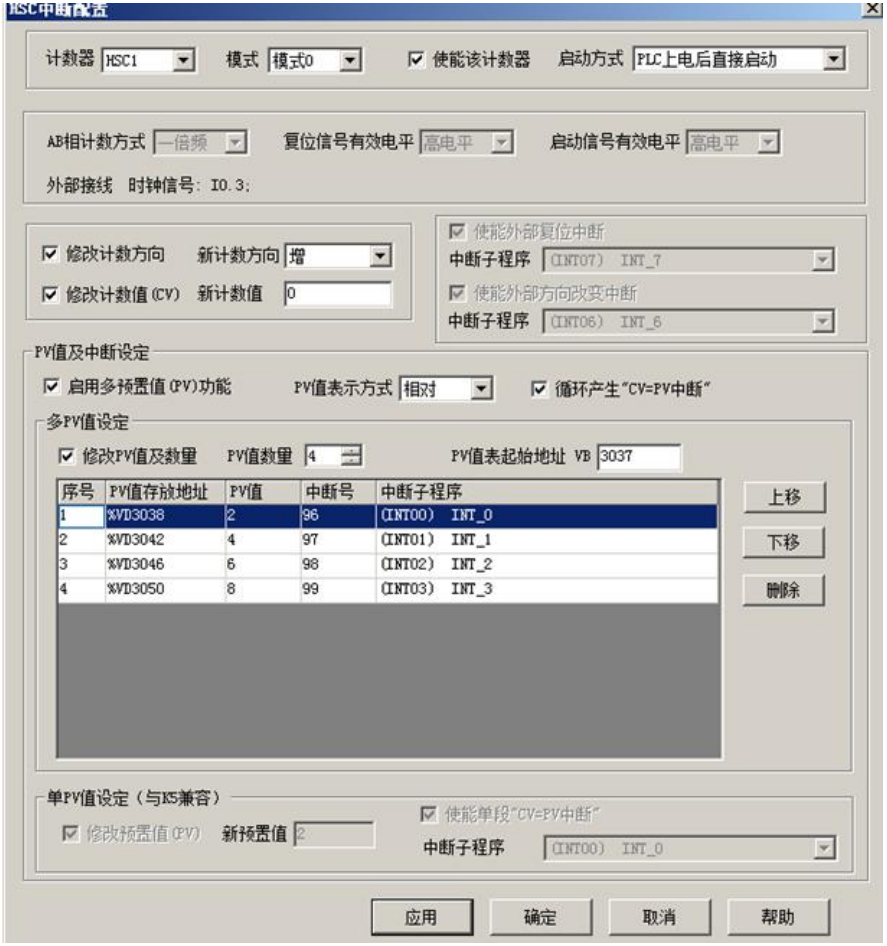
IL	(* Network 0 *)
	(*设置高数计数器的工作模式为 3，允许计数，并启动计数器*)
	LD %SM0.1
	HDEF 0, 3
	MOVE B#2#10000000, %SMB37
	HSC 0
	(* Network 1 *)
	(*通过 SPD 指令计算 HC0 计数器的输入脉冲频率*)
	LD %SM0.0
	MOVE %HC0, %VD50
	SPD 0, W#10, %VD54
	DI_TO_R %VD54, %VR58
	(* Network 2 *)
	(*将 SPD 计算值进行单位转换为每秒脉冲数即 HZ*)
	LD %SM0.0
	MOVE %VR58, %VR62
	DIV 0.01, %VR62
	(* Network 3 *)
	(*手动设置控制寄存器清零高速脉冲计数值*)
	LD %M50.0
	R_TRIG
	MOVE B#2#11000000, %SMB37
	MOVE DI#0, %SMD38
	ST %M50.1
	(* Network 4 *)
	(*重启高速计数器*)
	LD %M50.1
	HSC 0

8.11.2 使用 HSC 向导

8.11.2.1 HSC 向导设置

除 K5 系列之外的所有 PLC 用户都可以直接利用 HSC 向导的方式进行高速计数器的配置，无需再进行复杂的编程。向导如下图。

即使通过向导对 HSC 进行了配置，也只是对高速计数器进行了初始配置，用户可以在程序中按照“方法一”来随时对高速计数器进行参数修改、启动、停止。



HSC 向导的使用方法如下：

- 在【计数器】中，选择将要使用的计数器对应前面介绍的四路高速脉冲计数器。
- 选中【使能该计数器】，然后将会允许进行后续的配置对应控制器寄存中允许计数（以 HSC0 为例可以认为是对应控制寄存器 SM37.7）。
- 在【模式】中，选择将要使用的计数器模式对应 8.2 高速计数器工作模式和输入信号章节的工作模式。
- 在【启动方式】中，选择该高速计数器的启动方式。

  启动方式有如下两种：

  “在程序中调用 HSC 指令”：若选择这种方式，那么在用户程序中，通过调用 HSC 指令来启动该计数器。在调用 HSC 指令之前，无需再配置各寄存器和调用 HDEF 指令。

  “PLC 上电后直接启动”：若选择这种方式，那么该高速计数器在 PLC 上电后就自动运行，无需调用任何指令。
- 修改计数方向、修改计数值对应 8.6 控制寄存器和状态寄存器章节中控制寄存器的计数方向和当前值修改。
- 若要使用多段 PV 值方式，则选中【启用多预置值（PV）功能】，然后可以对 PV 值、数量、关联的中断子程序等进行配置。若选中【修改 PV 值及数量】，则可以调整【PV 值数量】中的数值，从而修改 PV 值个数。
- 若要使用单 PV 值方式，则首先选中“单 PV 值设定（与 K5 兼容）”中的【修改 PV 值】，然后可以修改 PV 值及关联的中断子程序。
- 其它的配置项，请参考前文的描述，按实际需求进行配置。

8.11.2.2 使用举例

下面的示例中采用多段 PV 的设定方式，PV 值表示方式为相对值，循环产生 CV=PV 中断，在向导中直接设定 PV 表，关联中断子程序即可，省去了之前设置状态寄存器、多段 PV 表和调用 HDEF、HSC 指令的部分程序等环节，设置好向导直接进行编程即可，简单易用。

下面分别举例说明了如何利用向导来配置高速计数器，并编写相关中断程序。

1）使用向导，

使用计数器 HSC0, 模式 0（带内部方向控制的单相增/减计数器），且使能该计数器。

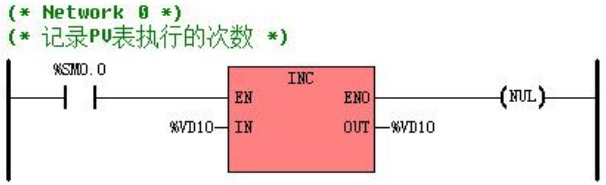
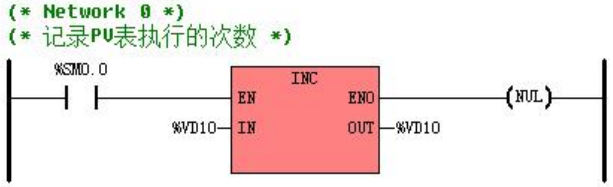
启动方式选择“PLC 上电后启动”，即不需要再去调用 HSC 指令启动计数器。

设定初始计数方向为增，当前计数值为 0（即从 0 开始计数）。

启用 PV 中断功能，设置 PV 表关联中断子程序（需要提前建立中断子程序才能进行关联），关于 PV 中断的说明请参考 8.8 PV 中断的使用。

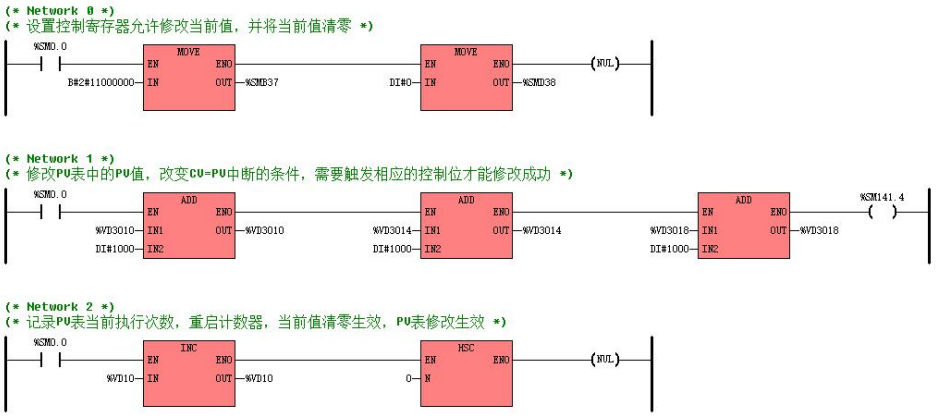
完成上述配置后，单击【确定】按钮退出，则 HSC0 的配置信息就会被保存在用户工程程序中。用户下载程序，PLC 重新运行后，上述配置立即生效，HSC0 将按照配置的方式启动。

2）编写所需的中断程序（已在向导中与计数中断进行了关联），并在 MAIN 程序中监视高速计数值。

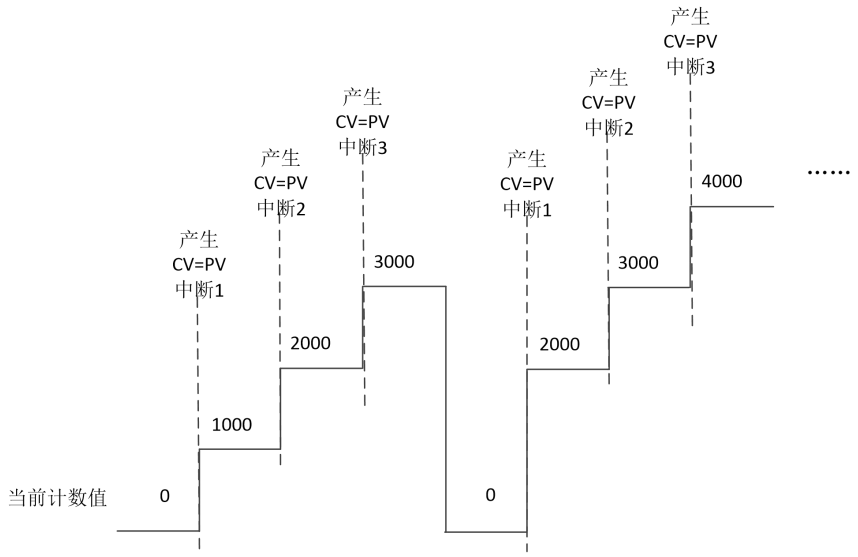
<table>
<tr><td>LD</td><td>主程序：MAIN
 
</td></tr>
<tr><td>LD</td><td>中断子程序 1：PV 表第一段计数到
 
</td></tr>
<tr><td></td><td>中断子程序 2：PV 表第二段计数到
 
</td></tr>
</table>

中断子程序 3：PV 表第三段计数到，清零当前计数值，修改 PV 表重启计数器生效

注意：向导只是设置了高速计数器的初始设置，如果需要修改高速计数器需要在程序中设置相应的寄存器并重启高速计数器才能生效。



时序图如下：



第九章 高速脉冲输出功能的使用

本章主要介绍 Kinco-K 系列 PLC 的高速脉冲输出功能，内容包括脉冲输出指令用法及编程应用、输出端子的接线、注意事项以及相关寄存器等。

9.1 功能概述

Kinco-K 系列 PLC 具有两路及两路通道以上的高速脉冲输出，每路最高输出频率也不完全相同，通过使用不同的指令编程方式，可以实现单段、多段脉冲串输出、脉宽调制、正反向输出、定位控制等功能。所有通道都支持 PTO（脉冲串）和 PWM（脉宽调制）方式输出。

脉冲输出的详细配置表如下：

PLC 型号	通道数	最高输出频率				输出方式
		Q0.0	Q0.1	Q0.4	Q0.5	
K5 系列	2	200K	200K			集电极开路输出（PNP）
K2 系列（K204ET）	3	200K	200K	200K		
K2 系列（K205）	3	50K	10K	10K		
K2 系列（K209EA）	3	200K	200K	10K		
K2 系列（K209M）	3	200K	200K	200K		
KS 系列(KS101M 无高速脉冲输出口）	4	200K	200K	200K	10K	
KW 系列	2	200K	200K			
MK 系列	4	50K	50K	50K	10K	
K6 系列	4	200K	200K	200K	10K	

注意：继电器输出型号的 CPU 模块（订货号最后位是“R”，比如 K205-16DR）不支持高速脉冲输出功能！

Q0.0 和 Q0.1、Q0.4 通道的最高输出频率时要求负载电阻不大于 3KΩ。

9.2 高速脉冲输出指令种类

Kinco-K 系列 PLC 的指令集中提供了如下 2 种指令用于高速输出功能：

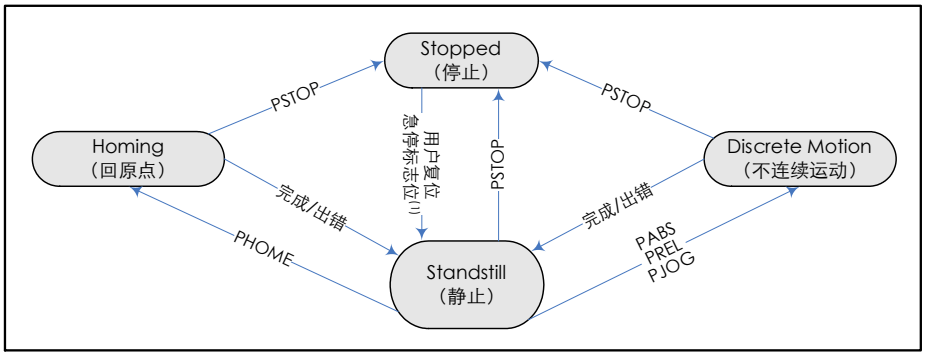
- 定位控制指令：包括 PREL（相对运动）、PABS（绝对运动）、PHOME（回原点）、PJOG（点动）、PJOGA（带加减速的点动）、PREL_M（多段相对运动）、PSTOP（急停）、PFLO_F（可变频率的脉冲串输出）指令，用户能够很方便地实现简单的定位控制功能。**注意：当使用定位控制指令时，输出脉冲频率不能低于 80Hz！**
- PLS 指令：可以实现 PTO（脉冲串输出单段或者多段）和 PWM（脉宽调制）输出功能。

9.3 定位控制指令的使用

本章节中描述的定位控制指令是高速脉冲输出功能的一种便利使用方法。与 PLS 指令相比，定位控制指令更适合于定位控制应用，用户能够使用它们很方便地实现常见的定位控制功能。需要注意的是，当使用定位控制指令时，输出脉冲频率的允许范围是 125Hz~200KHz（某些CPU的范围是125HZ-10KHZ）详情查看 9.1 功能概述各系列 PLC 高速脉冲输出口的最高输出脉冲频率，用户指定的输出频率必须在这个范围之内。

9.3.1 定位控制模型图

下图是定位控制的模型图。用户从图中可以了解如下信息：当定位控制指令执行后，相应高速输出通道的状态；在各个状态下，PLC 允许执行的定位控制指令。



(1) 急停标志位分别为 SM201.7/ SM231.7/SM251.7/SM221.7 依次对应四路高速脉冲输出通道，当 PSTOP 指令执行时，该位将自动置 1，在急停后，必须复位此标志位才可继续使用运动控制指令。这些特殊寄存器的说明请参阅 9.4.2.2 控制寄存器和状态寄存器。

9.3.2 定位控制指令的相关变量

9.3.2.1 电机方向控制信号

针对定位控制指令，Kinco-K 系列 PLC 为每路高速输出均指定了一个方向输出通道，同时还在 SM 区中提供了一个方向使能控制位。如下表。

脉冲输出通道	Q0.0	Q0.1	Q0.4	Q0.5
方向输出通道	Q0.2	Q0.3	Q0.6	Q0.7
方向使能控制位	SM201.3	SM231.3	SM251.3	SM221.3

方向输出通道用于输出电机的方向控制信号，正转时输出为 0，反转时输出为 1。

方向使能控制位用来禁止或者允许使用相应的方向输出通道。方向使能控制位具有最高的优先级，若设置为禁止，那么定位控制指令执行时将不会输出方向控制信号，相应的输出通道就可以作为普通的 DO 点使用。

注意：各系列 PLC 支持的高速脉冲输出通道数不完全相同，所以上述相对应寄存器只对具有高速脉冲输出口的 CPU 有效。

9.3.2.2 控制寄存器和状态寄存器

针对定位控制指令，Kinco-K 系列在 SM 区中为每路高速输出均分配了一个控制字节，在应用中用户需要注意设置该控制字节。另外，还分配了一个当前值（DINT 型）寄存器，用于存放当前已经输出的脉冲个数（正转时增加，反转时减少）。下表详细描述了这些寄存器。

Q0.0	Q0.1	Q0.4	Q0.5	描述
SMD212	SMD242	SMD262	SMD226	只读。当前值（正转时增加，反转时减少），表示当前已经输出的脉冲个数。
SMD208	SMD238	SMD258	SMD222	读写。 新当前值。与相应标志位配合，用于修改当前值。

Q0.0	Q0.1	Q0.4	Q0.5	描述
SM201.7	SM231.7	SM251.7	SM221.7	读写。急停标志位。若该位为 1，则表示处于急停状态，不执行任何定位控制指令。 当 PSTOP（急停）指令执行时，该位将自动置 1。用户需要使用程序将该位清 0。
SM201.6	SM231.6	SM251.6	SM221.6	读写。用于决定是否复位当前值 1 - 将当前值清零。 0 - 当前值保持不变。
SM201.5	SM231.5	SM251.5	SM221.5	保留
SM201.4	SM231.4	SM251.4	SM221.4	读写。用于决定是否修改当前值 1 - 将当前值清零。 0 - 当前值保持不变。
SM201.3	SM231.3	SM251.3	SM221.3	方向使能控制位。 1 禁止方向输出，方向通道作为普通 D0。 0 - 使能方向输出。
SM201.0 ~ SM201.2	SM231.0 ~ SM231.2	SM251.0 ~ SM251.2	SM221.0 ~ SM221.2	保留

- **如何修改当前值**

4 路高速输出通道各有一个当前值寄存器，分别为 SMD212、SMD242、SMD262 和 SMD226，其中存放着相应通道已经输出的脉冲个数。当前值寄存器是只读值，不允许在程序中直接进行修改。若需要修改当前值，则可以采取如下方法：

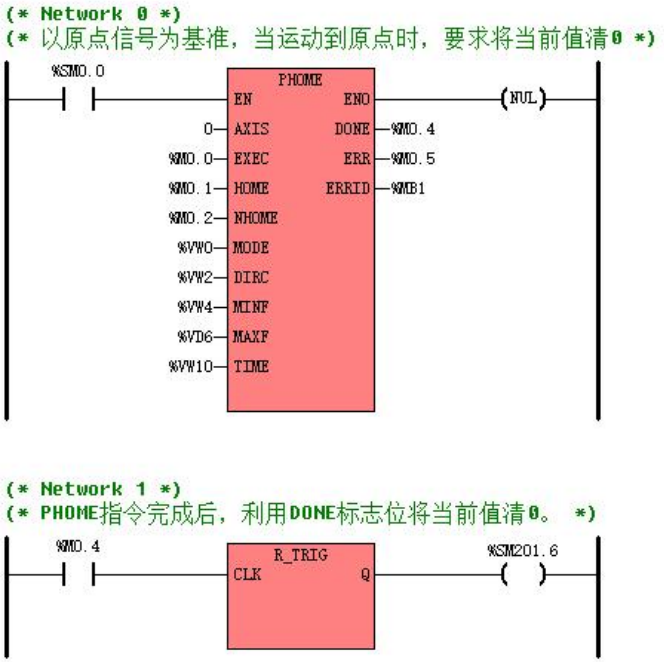
✧ **方法一**

利用复位控制位来将当前值清除为 0。

4 个通道的复位控制位分别为 SM201.6、SM231.6、SM251.6 和 SM221.6。

只要复位控制位为 1， PLC 就会将相应的当前值寄存器清 0。因此复位控制位仅需要保持一个扫描周期即可发挥作用，使用时注意避免复位控制位长时间保持为 1，也尽量避免在运动过程中（包括 PHOME、PREL、PABS、PJOG、PFLO_F 指令正在执行）来复位当前值，以免计数出现误差。

下面以通道 0 为例来说明如何复位当前值：



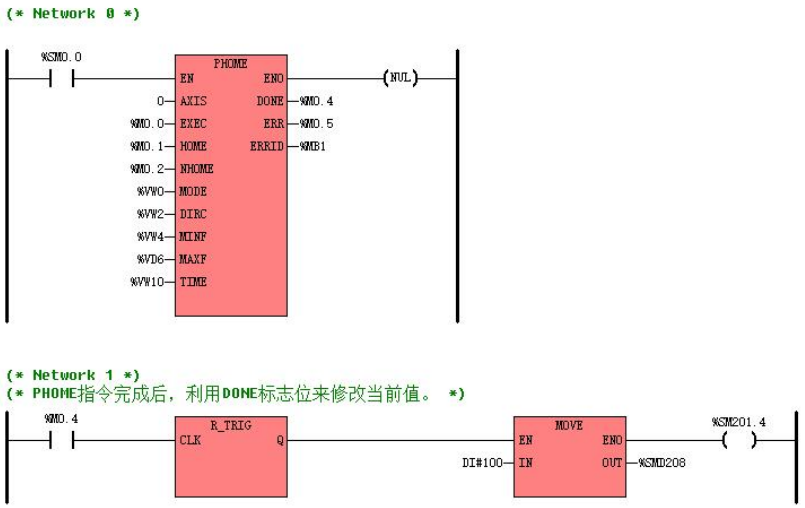
✧ **方法二**

利用下述寄存器，可以将当前值设置为任意值。

Q0.0	Q0.1	Q0.4	Q0.5	描述
SMD208	SMD238	SDM258	SMD222	读写。新当前值。与相应标志位配合，用于修改当前值。
SM201.4	SM231.4	SM251.4	SM221.4	读写。用于决定是否修改当前值 1 - 将当前值清零。 0 - 当前值保持不变。

以通道 0 为例来说明使用方法：若 SM201.4 为 0，则保持当前值 SMD212 不变。若 SM201.4 为 1，则将 SMD208 中的值赋值给当前值 SMD212。注意尽量避免在运动过程中（包括 PHOME、PREL、PABS、PJOG、PFLO_F 指令正在执行）来修改当前值寄存器，以免当前值计数出现误差。

下面的示例程序是以通道 0 为例，说明如何修改当前值：



9.3.3 定位控制指令

9.3.3.1 定位控制指令综述

下述指令位于指令集的【定位控制】组中。

名称	功能描述
PHOME	回原点
PABS	绝对运动
PREL	相对运动
PJOG	点动
PJOGA	带加减速的点动
PSTOP	急停
PFLO_F	可变频率的脉冲串输出
PREL_M	多段相对运动

1) 注意事项

1、CPU 的输出频率范围限制

- 当使用定位控制指令时，输出脉冲频率不能低于 125Hz 且不能高于 CPU 的最高输出频率，否则将导致 CPU 无法正常输出脉冲。输出低于最低频率指令 125HZ 或者高于最高 200K HZ，执行指令会直接报错且输出错误代码，但需要注意的是有部分经济型 CPU(如 K205，一体机等系列)本身 CPU 硬件规格即无法达到最高 200K，此时指令不会报错，但实际使用时应该以 CPU 规定最高频率为主（详情参见 9.1 功能概述），否则脉冲输出将是错误的！

2、急停标志位的使用方法

- KPLC 最多有四路高速脉冲输出通道，对应四个急停标志位分别为 SM201.7/ SM231.7/SM251.7 /SM221.7，当 PSTOP 指令执行时，该位将自动置 1，表示处于急停状态，不执行任何定位控制指令，用户如果想再次执行定位指令需要在程序中将该位清 0。

3、方向使能控制位的使用方法

- KPLC 的四路高速脉冲输出通道对应四路方向控制信号，这四路方向控制信号可以通过方向使能控制位用来禁止或者允许使用相应的方向输出通道，如果禁止可以使方向输出通道作为普通 DO 使用，详细说明参见 9.3.2.1 电机方向控制信号章节。

4、如何清零脉冲计数值？

- 详细说明参见 9.3.2.2 控制寄存器和状态寄存器章节。

5、如何修改定位控制指令中的最高输出频率？

- PREL（相对运动）和 PABS（绝对运动）指令在运行过程中不允许修改输出频率 *MINF*、*MAXF* 值，即使修改了也不会生效。
- PHOME（回原点）指令在运行到最高频率段之后、在回原点和原点信号产生之前，会实时读取最高频率参数（*MAXF*）值，并根据新的频率值自动计算加速或者减速段数，然后加速或者减速运行到新的频率值再维持匀速输出。
- PJOG（点动）指令在执行过程中会实时读取输入频率参数（*MAXF*）值，并根据新的频率值来调整输出脉冲的频率。
- PJOGA（带加减速的点动）指令在执行过程中不能通过修改最高频率（*MAXF*）值的方式来达到实时调整输出脉冲频率的效果。

- PFLO_F（可变频率的脉冲串输出）提供随时可变频率的输出。具体见指令介绍。
- PREL_M（多段相对运动）用户可以根据实际情况自定义 PV 曲线（速度位置关系），最多可以定义 8 段，具体见指令介绍。

2) 指令输出参数 ERRID

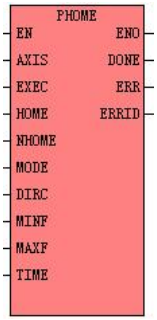
定位控制指令在执行时可能会产生非致命错误，这时候，CPU 就会产生一个错误码并输出至指令的 *ERRID* 参数中。下表列出了这些错误码及其描述。

错误码	描 述
0	正常
1	加/减速时间太短或初始速度太低，导致初始脉冲周期超过了每段的时间。
2	初始速度 MINF 超过最高允许速度（200KHz），部分 CPU 本身硬件限制无法达到 200K，指令不会报错，但实际应以 9.1 功能概述脉冲输出规格为准。
3	初始速度 MINF 低于最低允许速度（125Hz）。
4	加速和减速过程所需的脉冲个数超过了总脉冲个数。
5	初始速度 MINF 超过了最高速度 MAXF。
6	在 PREL_M 指令中，段数值为 0 或者超过了最大允许的段数。
7	在 PREL_M 指令中，方向错误，即所有段的运行方向不完全相同。
8	在 PREL_M 指令中，各种参数输入的地址有错误。
9	在 PREL_M 指令中，某相邻两段之间存在多个加/减速过程，即第 n 段 ADTIM2 的值与第 n+1 段的 ADTIM1 的值都不为 0.
10	在 PREL_M 指令中，无法根据各输入参数的值来分配成功全部的加/减速过程。
11	PREL_M 指令在执行过程中出现异常。
12	本通道有脉冲输出指令正在执行，所以无法启动 PREL_M 指令。
13	本通道的急停标志位为 1，所以无法启动 PREL_M 指令。

9.3.3.2 定位控制指令

9.3.3.2.1 PHOME（回原点）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	适用于
LD	PHOME			☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
IL	PHOME	PHOME *AXIS, EXEC, HOME, NHOME, MODE, DIRC, MINF, MAXF, TIME, DONE, ERR, ERRID*	U	

参数	输入/输出	数据类型	允许的内存区
AXIS	输入	INT	常量（0、1、2、3）
EXEC	输入	BOOL	I、Q、V、M、L、SM、RS、SR
HOME	输入	BOOL	I 、Q、V、M、L、SM、RS、SR
NHOME	输入	BOOL	I 、Q、V、M、L、SM、RS、SR
MODE	输入	INT	I、Q、V、M、L、SM、T、C、AI、AQ、常量
DIRC	输入	INT	I、Q、V、M、L、SM、T、C、AI、AQ、常量
MINF	输入	WORD	I、Q、M、V、L、SM、常量
MAXF	输入	DWORD	I、Q、M、V、L、SM、常量
TIME	输入	WORD	I、Q、M、V、L、SM、常量
DONE	输出	BOOL	Q、M、V、L、SM
ERR	输出	BOOL	Q、M、V、L、SM
ERRID	输出	BYTE	Q、M、V、L、SM

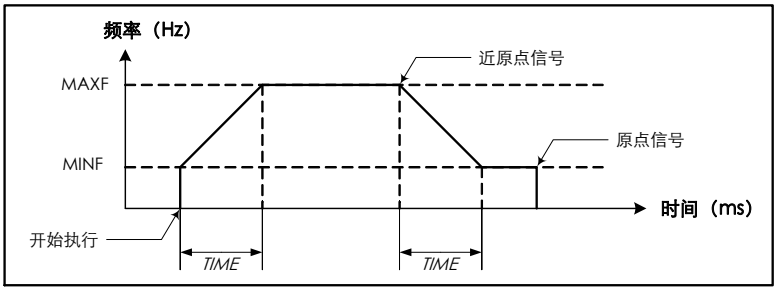
MODE, DIRC, MINF, MAXF, TIME, 必须同时为常量类型或同时为内存类型

下表对各个参数的作用进行了详细的描述。

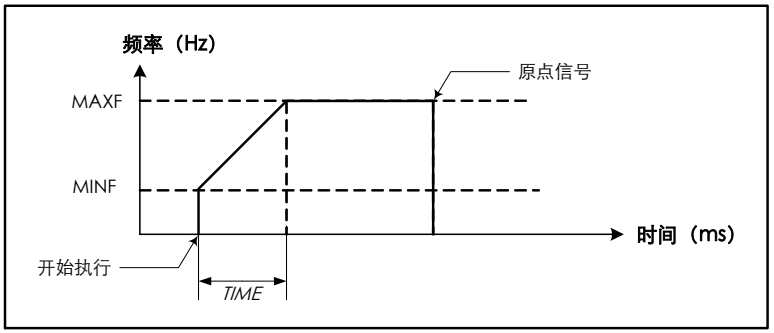
参数	描　述
AXIS	所用的高速输出通道。0 表示使用 Q0.0，1 表示使用 Q0.1，2 表示使用 Q0.4，3 表示使用 Q0.5。
EXEC	若检测到 *EXEC* 的上升沿跳变，则 PHOME 指令被触发执行。
HOME	原点输入信号。
NHOME	近原点输入信号。
MODE	回原点的控制方式： 0 表示利用近原点和原点输入信号进行控制；1 表示只利用原点输入信号进行控制
DIRC	电机的运转方向：0 表示正转；1 表示反转。 关于方向控制信号的输出请参阅 9.4.2.1 电机方向控制信号中的描述。
MINF	输出脉冲的初始速度（即初始频率），单位：Hz。 *MINF* 不允许低于 125Hz，也不允许超过 *MAXF*。
MAXF	输出脉冲的最高速度（即最高频率），单位：Hz。 *MAXF* 的允许范围是 125Hz 到 CPU 允许输出的最大脉冲频率。*MAXF* 必须大于等于 *MINF*。
TIME	加/减速时间，单位：ms。本指令采用相同的加速时间和减速时间。 加速时间是由 *MINF* 加速到 *MAXF* 所需的时间，减速时间是由 *MAXF* 减速到 *MINF* 所需的时间。
DONE	完成标志位。当指令正常执行完成时，*DONE* 由 0 跳变到 1。
ERR	出错标志位。若指令执行时发生错误，则该标志位被置 1。
ERRID	错误码。

PHOME 指令利用近原点和原点输入信号进行返回原点的控制，参数 *MODE* 定义了控制方式：

① 若利用近原点和原点信号进行控制，则当检测到近原点信号时开始减速，当检测到原点信号时停止脉冲输出。时序图如下：



② 若只利用原点信号进行控制，则当检测到原点信号时停止脉冲输出。时序图如下：



PHOME 指令执行时，若 *DIRC* 设定为正转，则当前脉冲计数值将会增加，若 *DIRC* 设定为反转，则当前脉冲计数值将会减少。**注意：当回原点运动完成后，当前脉冲计数值寄存器并不自动清零，用户需要根据实际要求自行改变当前值。详见 9.3.2.2 控制寄存器和状态寄存器中的描述。**

- LD

  当 *EN* 为 1 时，若检测到 *EXEC* 输入端的上升沿，则该指令被触发执行。

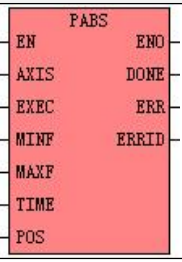
- IL

  当 CR 值为 1 时，若检测到 *EXEC* 输入端的上升沿，则该指令被触发执行。

  该指令的执行不影响 CR 值。

9.3.3.2.2 PABS（绝对运动）

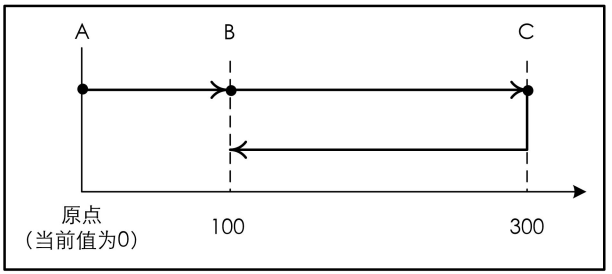
➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	适用于
LD	PABS			☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
IL	PABS	PABS AXIS, EXEC, MINF, MAXF, TIME, POS, DONE, ERR, ERRID	U	

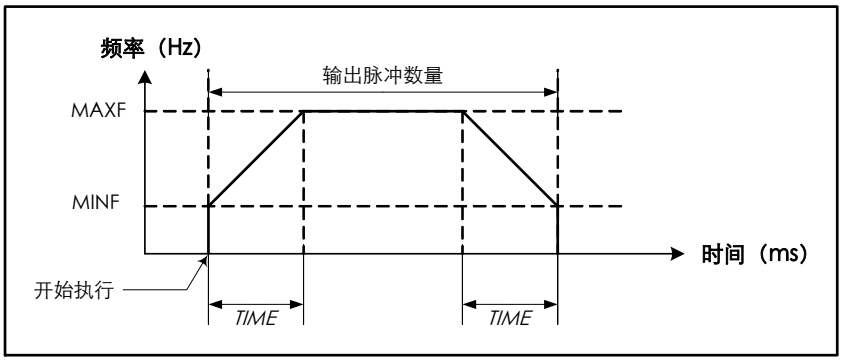
参数	输入/输出	数据类型	允许的内存区
AXIS	输入	INT	常量（0、1、2、3）
EXEC	输入	BOOL	I、Q、V、M、L、SM、RS、SR
MINF	输入	WORD	I、Q、M、V、L、SM、常量
MAXF	输入	DWORD	I、Q、M、V、L、SM、常量
TIME	输入	WORD	I、Q、M、V、L、SM、常量
POS	输入	DINT	I、Q、M、V、L、SM、HC、常量
DONE	输出	BOOL	Q、M、V、L、SM
ERR	输出	BOOL	Q、M、V、L、SM
ERRID	输出	BYTE	Q、M、V、L、SM

MINF,MAXF,TIME,POS, 必须同时为常量类型或同时为内存类型

下表对各个参数的作用进行了详细的描述。

参数	描　述
AXIS	所用的高速输出通道。0 表示使用 Q0.0，1 表示使用 Q0.1，2 表示使用 Q0.4，3 表示使用 Q0.5。
EXEC	若检测到 *EXEC* 的上升沿跳变，则 PABS 指令被触发执行。
MINF	输出脉冲的初始速度（即初始频率），单位：Hz。 *MINF* 不允许低于 125Hz，也不允许超过 *MAXF*。
MAXF	输出脉冲的最高速度（即最高频率），单位：Hz。 *MAXF* 的允许范围是 125Hz 到 CPU 允许输出的最大脉冲频率。*MAXF* 必须大于等于 *MINF*。
TIME	加/减速时间，单位：ms。本指令采用相同的加速时间和减速时间。 加速时间是由 *MINF* 加速到 *MAXF* 所需的时间，减速时间是由 *MAXF* 减速到 *MINF* 所需的时间。
POS	目标值，也就是原点到要移动到的位置之间所需的脉冲个数。 如下图，若将物体由 A 处移到 B 处，则目标值应设定为“100”；若从 B 处移到 C 处，则目标值应设定为“300”；若从 C 处移到 B 处，则目标值设定为“100”。 
DONE	完成标志位。当指令正常执行完成时，*DONE* 由 0 跳变到 1。
ERR	出错标志位。若指令执行时发生错误，则该标志位被置 1。
ERRID	错误码。

PABS 指令采用绝对式定位，根据当前值与目标值 *POS* 之间的差值来输出脉冲。当前值与目标值之间的差值就是输出脉冲的数量。PABS 指令执行的时序图如下：



若方向使能控制位被设置为0,那么 PABS 指令将在相应的方向输出通道输出电机的方向控制信号：当目标值>当前值时，输出正转信号，此时当前脉冲计数值将会增加；当目标值<当前值时，输出反转信号，此时当前脉冲计数值将会减少。

- LD

  当 *EN* 为 1 时，若检测到 *EXEC* 输入端的上升沿，则该指令被触发执行。

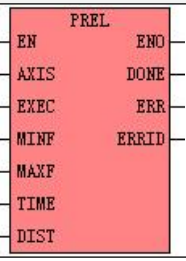
- IL

  当 CR 值为 1 时，若检测到 *EXEC* 输入端的上升沿，则该指令被触发执行。

  该指令的执行不影响 CR 值。

9.3.3.2.3 PREL（相对运动）

➢ 指令及其操作数说明

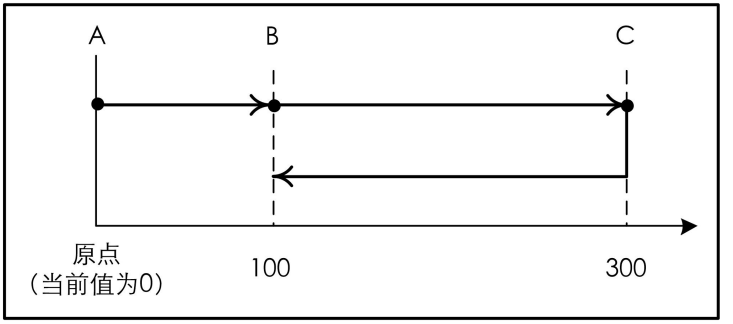
	名称	指令格式	影响 CR 值	适用于
LD	PREL			☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
IL	PREL	PREL AXIS, EXEC, MINF, MAXF, TIME, DIST, DONE, ERR, ERRID	U	

参数	输入/输出	数据类型	允许的内存区
AXIS	输入	INT	常量（0、1、2、3）
EXEC	输入	BOOL	I、Q、V、M、L、SM、RS、SR
MINF	输入	WORD	I、Q、M、V、L、SM、常量
MAXF	输入	DWORD	I、Q、M、V、L、SM、常量
TIME	输入	WORD	I、Q、M、V、L、SM、常量
DIST	输入	DINT	I、Q、M、V、L、SM、HC、常量
DONE	输出	BOOL	Q、M、V、L、SM
ERR	输出	BOOL	Q、M、V、L、SM
ERRID	输出	BYTE	Q、M、V、L、SM

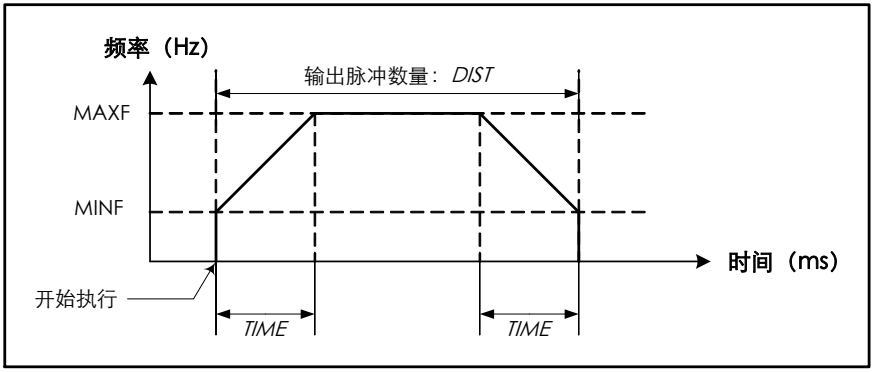
MINF，MAXF，TIME，DIST 必须同时为常量类型或同时为内存类型

下表对各个参数的作用进行了详细的描述。

参数	描　述
AXIS	所用的高速输出通道。 0 表示使用 Q0.0，1 表示使用 Q0.1，2 表示使用 Q0.4，3 表示使用 Q0.5。
EXEC	若检测到 *EXEC* 的上升沿跳变，则 PREL 指令被触发执行。

MINF	输出脉冲的初始速度（即初始频率），单位：Hz。 *MINF* 不允许低于 125Hz，也不允许超过 *MAXF*。
MAXF	输出脉冲的最高速度（即最高频率），单位：Hz。 *MAXF* 的允许范围是 125Hz 到 CPU 允许输出的最大脉冲频率。*MAXF* 必须大于等于 *MINF*。
TIME	加/减速时间，单位：ms。本指令采用相同的加速时间和减速时间。 加速时间是由 *MINF* 加速到 *MAXF* 所需的时间，减速时间是由 *MAXF* 减速到 *MINF* 所需的时间。
DIST	移动量，也就是当前位置到要移动到的位置之间所需的脉冲个数。 如下图，若将物体由 A 处移到 B 处，则移动量应设定为“100”；若从 B 处移到 C 处，则移动量应设定为“200”；若从 C 处移到 B 处，则移动量设定为“-200”。 
DONE	完成标志位。当指令正常执行完成时，*DONE* 由 0 跳变到 1。
ERR	出错标志位。若指令执行时发生错误，则该标志位被置 1。
ERRID	错误码。

PREL 指令采用相对式定位，输出脉冲的数量就是移动量 *DIST*。PREL 指令执行的时序图如下：



若方向使能控制位被设置为0,那么 PREL 指令将在相应的方向输出通道输出电机的方向控制信号：当移动量为正数时，输出正转信号，此时当前脉冲计数值将会增加；当移动量为负数时，输出反转信号，此时当前脉冲计数值将会减少。

- LD

  当 *EN* 为 1 时，若检测到 *EXEC* 输入端的上升沿，则该指令被触发执行。

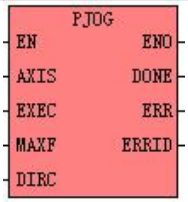
- IL

  当 CR 值为 1 时，若检测到 *EXEC* 输入端的上升沿，则该指令被触发执行。

  该指令的执行不影响 CR 值。

9.3.3.2.4 PJOG（点动）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	适用于
LD	PJOG			☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
IL	PJOG	PJOG AXIS, EXEC, MAXF, DIRC, DONE, ERR, ERRID	U	

参数	输入/输出	数据类型	允许的内存区
AXIS	输入	INT	常量（0、1、2、3）
EXEC	输入	BOOL	I、Q、V、M、L、SM、RS、SR
MAXF	输入	DWORD	I、Q、M、V、L、SM、常量
DIRC	输入	INT	I、Q、V、M、L、SM、T、C、AI、AQ、常量
DONE	输出	BOOL	Q、M、V、L、SM
ERR	输出	BOOL	Q、M、V、L、SM
ERRID	输出	BYTE	Q、M、V、L、SM

MAXF，DIRC 必须同时为常量类型或同时为内存类型

下表对各个参数的作用进行了详细的描述。

参数	描　述
AXIS	所用的高速输出通道。 0 表示使用 Q0.0，1 表示使用 Q0.1，2 表示使用 Q0.4，3 表示使用 Q0.5。

EXEC	若 *EXEC* 为 1，则持续输出脉冲；若为 0，则停止输出。
MAXF	输出脉冲的频率，单位：Hz。允许范围是 125Hz 到 CPU 允许输出的最大脉冲频率。
DIRC	电机的运转方向：0 表示正转；1 表示反转。 关于方向控制信号的输出请参阅 9.3.2.1 电机方向控制信号中的描述。
DONE	完成标志位。当指令正常执行完成时，*DONE* 由 0 跳变到 1。
ERR	出错标志位。若指令执行时发生错误，则该标志位被置 1。
ERRID	错误码。

若 *EXEC* 输入为 1，则 PJOG 指令从指定的通道 *AXIS* 持续输出脉冲串，频率为 *MAXF*，若 *MAXF* 是变量，那么在脉冲输出过程中可以随时改变这个输出频率值。若 *EXEC* 输入为 0，则立即停止输出。在指令运行过程中，用户可以修改输出频率值 *MAXF*，PJOG 指令会立即按新的 *MAXF* 频率值来输出脉冲串。

PJOG 指令执行时，若 *DIRC* 设定为正转，则当前脉冲计数值将会增加，若 *DIRC* 设定为反转，则当前脉冲计数值将会减少。

- LD

  当 *EN* 为 1 时，若 *EXEC* 输入为 1，则该指令被执行：

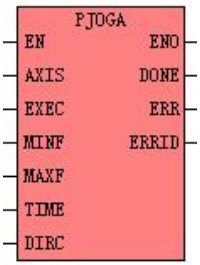
- IL

  当 CR 值为 1 时，若 *EXEC* 输入为 1，则该指令被执行。

  该指令的执行不影响 CR 值。

9.3.3.2.5 PJOGA（点动带加减速）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	适用于
LD	PJOGA			☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
IL	PJOGA	PJOGA AXIS, EXEC, MINF, MAXF, TIME, DIRC, DONE, ERR, ERRID	U	

参数	输入/输出	数据类型	允许的内存区
AXIS	输入	INT	常量（0、1、2、3）
EXEC	输入	BOOL	I、Q、V、M、L、SM、RS、SR
MINF	输入	WORD	I、Q、M、V、L、SM、常量
MAXF	输入	DWORD	I、Q、M、V、L、SM、常量
TIME	输入	WORD	I、Q、M、V、L、SM、常量
DIRC	输入	INT	I、Q、V、M、L、SM、T、C、AI、AQ、常量
DONE	输出	BOOL	Q、M、V、L、SM
ERR	输出	BOOL	Q、M、V、L、SM
ERRID	输出	BYTE	Q、M、V、L、SM

MINF, MAXF, TIME, DIST 必须同时为常量类型或同时为内存类型。

注意：与 PJOG 不同的是 PJOGA 是带加减速控制的点动指令，另外一点就是 PJOGA 执行过程中不允许通过修改 MAXF 的值来达到实时修改运行速度的效果。

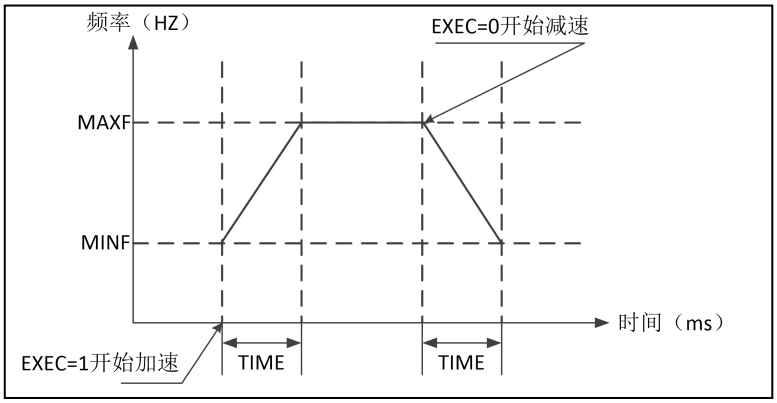
下表对各个参数的作用进行了详细的描述。

参数	描　述
AXIS	所用的高速输出通道。 0 表示使用 Q0.0，1 表示使用 Q0.1，2 表示使用 Q0.4，3 表示使用 Q0.5。
EXEC	若 *EXEC* 为 1，则持续输出脉冲；若为 0，则停止输出。
MINF	输出脉冲的初始速度（即初始频率），单位：Hz。 MINF 不允许低于 125Hz，也不允许超过 MAXF。
MAXF	MAXF 的允许范围是 125Hz 到 CPU 允许输出的最大脉冲频率。 MAXF 必须大于等于 MINF。
TIME	加/减速时间，单位：ms。本指令采用相同的加速时间和减速时间。 加速时间是由 MINF 加速到 MAXF 所需的时间,减速时间是由 MAXF 减速到 MINF 所需的时间。
DIRC	电机的运转方向：0 表示正转；1 表示反转。 关于方向控制信号的输出请参阅 9.3.2.1 电机方向控制信号。
DONE	完成标志位。当指令正常执行完成时，*DONE* 由 0 跳变到 1。
ERR	出错标志位。若指令执行时发生错误，则该标志位被置 1。
ERRID	错误码。

若 *EXEC* 输入为 1，则 *PJOGA* 指令从指定的通道 *AXIS* 持续输出脉冲串，首先经历一个加速过程由最小频率 *MINF* 增加到最大频率 *MAXF*，加速度大小由加减速时间 *TIME* 决定。若 *EXEC* 输入为 0，则根据设定的加减速时间 *TIME* 进行减速直到最小频率停止输出，如果减速过程中出现 *EXEC* 输入重新为 1，则从当前的速度重新转入加速启动过程中。

PJOGA 指令执行时，若 *DIRC* 设定为正转，则当前脉冲计数值将会增加，若 *DIRC* 设定为反转，则当前脉冲计数值将会减少。

PJOGA 指令执行的时序图如下：



- LD

  当 *EN* 为 1 时，若 *EXEC* 输入为 1，则该指令被执行：

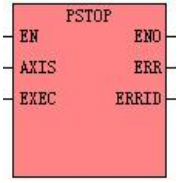
- IL

  当 CR 值为 1 时，若 *EXEC* 输入为 1，则该指令被执行。

  该指令的执行不影响 CR 值。

9.3.3.2.6 PSTOP（急停）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	适用于
LD	PSTOP			☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
IL	PSTOP	PSTOP AXIS, EXEC, ERR, ERRID	U	

参数	输入/输出	数据类型	允许的内存区
AXIS	输入	INT	常量（0、1、2、3）
EXEC	输入	BOOL	I、Q、V、M、L、SM、RS、SR
ERR	输出	BOOL	Q、M、V、L、SM
ERRID	输出	BYTE	Q、M、V、L、SM

下表对各个参数的作用进行了详细的描述。

参数	描　述
AXIS	所用的高速输出通道。 0 表示使用 Q0.0，1 表示使用 Q0.1，2 表示使用 Q0.4，3 表示使用 Q0.5。
EXEC	若检测到 *EXEC* 的上升沿跳变，则 PSTOP 指令被触发执行。
ERR	出错标志位。若指令执行时发生错误，则该标志位被置 1。
ERRID	错误码。

PSTOP 指令用于立即停止 *AXIS* 通道的脉冲输出，从而使当前的运动紧急停止，同时将急停标志位置位。KPLC 最多有四路高速脉冲输出通道，对应四个急停标志位分别为 SM201.7/SM231.7/SM251.7/SM221.7。**用户需要使用程序将急停标志位清 0，否则 CPU 不会再执行任何定位控制指令。**

- LD

  当 *EN* 为 1 时，若检测到 *EXEC* 输入端的上升沿，则该指令被触发执行。

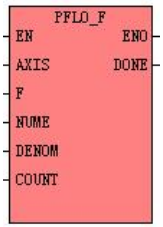
- IL

  当 CR 值为 1 时，若检测到 *EXEC* 输入端的上升沿，则该指令被触发执行。

  该指令的执行不影响 CR 值。

9.3.3.2.7 PFLO_F（可变频率的脉冲串输出）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	适用于
LD	PFLO_F			☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
IL	PFLO_F	PFLO_F AXIS, F, NUME, DENOM,COUNT, DONE	U	

参数	输入/输出	数据类型	允许的内存区
AXIS	输入	INT	常量（0、1、2、3）
F	输入	DINT	L、M、V、常量
NUME	输入	INT	L、M、V、常量
DENOM	输入	INT	L、M、V、常量
DONE	输出	BOOL	Q、M、V、L
COUNT	输入	DWORD	L、M、V、常量

F, NUME, DENOM, COUNT 必须同时为常量类型或同时为内存类型。

下表对各个参数的作用进行了详细的描述。

参数	描 述
EN	使能端。若 EN 为 1，则执行跟随脉冲输出，否则停止脉冲输出。
AXIS	所用的高速输出通道。0 表示使用 Q0.0，1 表示使用 Q0.1，2 表示使用 Q0.4，3 表示使用 Q0.5。
F	输入频率。单位：Hz
NUME	输出脉冲频率的电子齿轮分子
DENOM	输出脉冲频率的电子齿轮分母
DONE	完成标志位。若有脉冲输出，则为 0；若停止脉冲输出则为 1。

若 *EN* 为 1，则执行指令，从指定的通道 *AXIS* 持续输出脉冲，输出频率等于输入频率 *F* 乘以电子齿轮（*NUME*/ *DENOM*），总共输出 *COUNT* 个脉冲。当输出脉冲个数达到 *COUNT* 个则完成输出，将标志位 *DONE* 置 1。若输入频率 *F* 小于 0，则表示反转，对应的方向通道输出为 1；若 F 大于 0，则表示正转，对应的方向通道输出为 0。

注意：若计算得到输出频率小于最低允许频率 125Hz，PLC 则停止输出，当输出频率再次超过最低频率后才会继续跟随输出，输出频率不允许超过 CPU 的最高输出频率，请设置适合的输入频率和电子齿轮比。

- LD

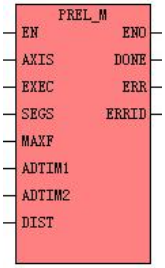
  当 *EN* 为 1 时，则该指令被执行。若 EN 输入为 0，则立即停止输出。

- IL

  当 CR 值为 1 时，则该指令被执行。该指令的执行不影响 CR 值。

9.3.3.2.8 PREL_M（多段相对运动）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	适用于
LD	PREL_M			☑ MK ☑ K6
IL	PREL_M	PREL_M AXIS, EXEC, SEGS, MAXF, ADTIM1, ADTIM1, DIST, DONE, ERR, ERRID	U	

参数	输入/输出	数据类型	允许的内存区
AXIS	输入	INT	常量（0、1、2、3）
EXEC	输入	BOOL	I、Q、V、M、L、SM、RS、SR
SEGS	输入	WORD	M、V、L、SM、常量
MAXF	输入	DINT	M、V、L、SM
ADTIM1	输入	WORD	M、V、L、SM
ADTIM1	输入	WORD	M、V、L、SM
DIST	输入	DINT	M、V、L、SM
DONE	输出	BOOL	Q、M、V、L、SM
ERR	输出	BOOL	Q、M、V、L、SM
ERRID	输出	BYTE	Q、M、V、L、SM

下表对各个参数的作用进行了详细的描述。

参数	描　述
AXIS	所用的高速输出通道。 0 表示使用 Q0.0，1 表示使用 Q0.1，2 表示使用 Q0.4，3 表示使用 Q0.5。
EXEC	若检测到 *EXEC* 的上升沿跳变，则 PREL_M 指令被触发执行。
SEGS	自定义曲线的总段数（1-8）。
MAXF	输出脉冲的最高速度（即最高频率），单位：Hz。 *MAXF* 的允许范围是 125Hz 到 CPU 允许输出的最大脉冲频率。 本参数为各段的频率的起始地址，即第 1 段的地址，后续各段在内存中连续存放。
ADTIM1	起始加/减速时间，单位：ms。0 代表没有加减速时间。 本参数为各段的起始加/减速时间的起始地址，即第 1 段的地址，后续各段在内存中连续存放。
ADTIM2	结束加/减速时间，单位：ms。0 代表没有加减速时间。 本参数为各段的结束加/减速时间的起始地址，即第 1 段的地址，后续各段在内存中连续存放。
DIST	移动量（包含加减速过程），单位：脉冲。正数代表正方向、负数代表负方向。 本参数为各段的移动量的起始地址，即第 1 段的地址，后续各段在内存中连续存放。
DONE	完成标志位。当指令正常执行完成时，*DONE* 由 0 跳变到 1。
ERR	出错标志位。若指令执行时发生错误，则该标志位被置 1。

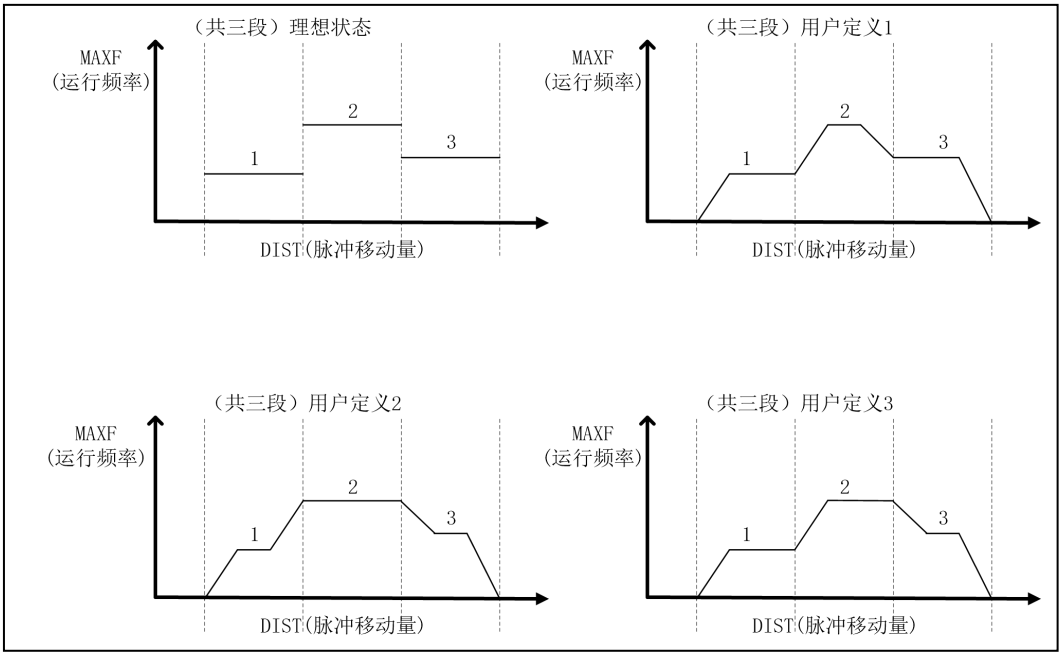
ERRID	错误码。

注意：本指令默认从静止开始启动，用户也可以修改下述 SM 寄存器来任意设定启动速度：0 通道 SMD204， 1 通道是 SMD234，2 通道是 SMD254，3 通道是 SMD180。若用户设定的启动速度小于 100Hz 则默认采用 100Hz，而且启动速度不允许超过第 1 段的最高速度（MAXF）。

本指令支持连续输出 1-8 段脉冲串。用户程序中可以定义期望的段数，以及每段的脉冲个数、频率、加速时间和减速时间。当本指令执行时，将根据用户设定的参数连续输出脉冲，其中第 1 段从静止或者用户设定的启动速度来启动，同时第 1 段的速度将作为第 2 段的启动速度，依次类推。每段的速度可以高于或者低于前一段速度，指令将自动判断并生成加速或者减速过程。

该指令的所有曲线段都只能同时为正方向或者同时为负方向。

注意：相邻两段之间只允许 1 个加速或者减速过程，也就是说，第 n 段 ADTIM2 的值与第 n+1 段的 ADTIM1 的值至少有一个必须为 0。下图以 3 段为例说明了几种可以自定义的输出时序图：



- LD

  当 *EN* 为 1 时，若检测到 *EXEC* 输入端的上升沿，则该指令被触发执行。

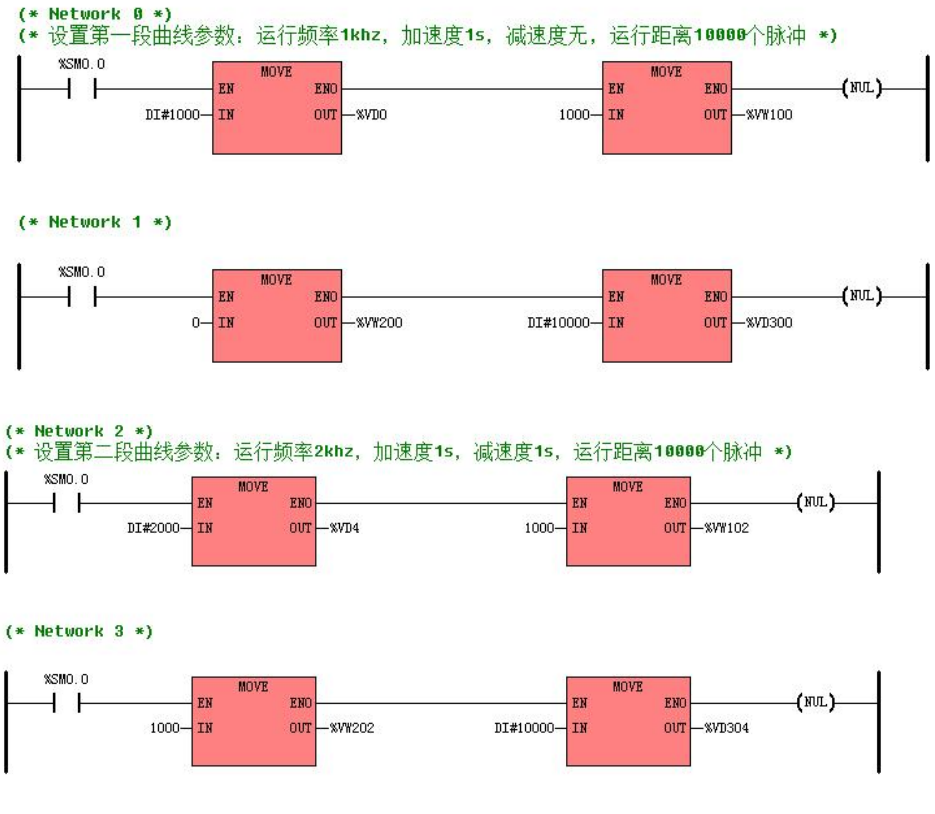
- IL

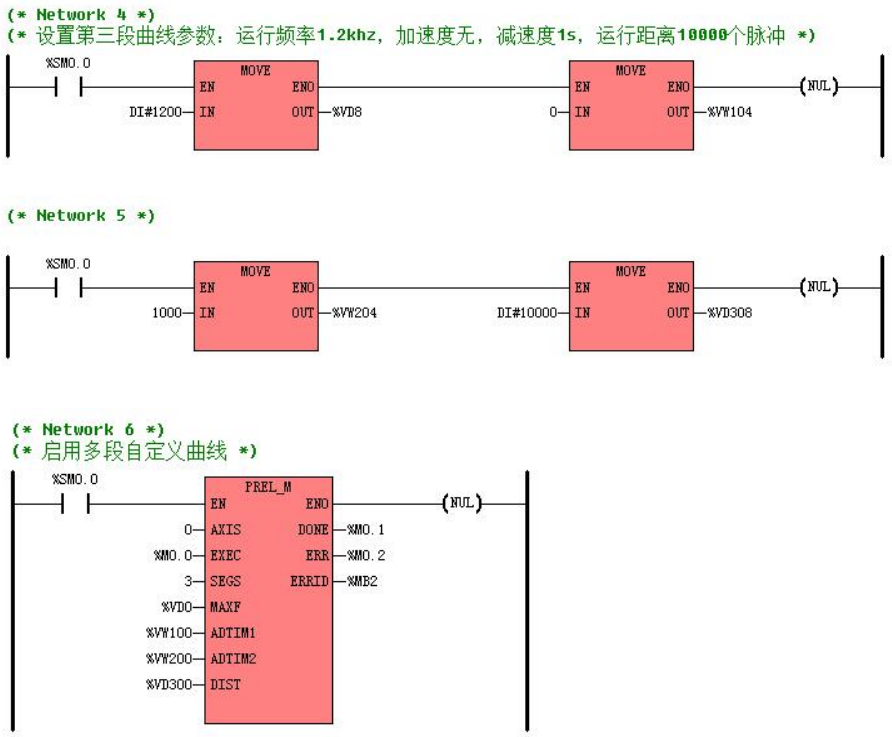
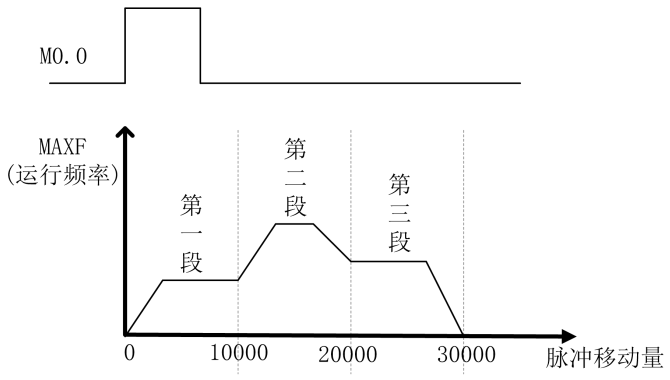
  当 CR 值为 1 时，若检测到 *EXEC* 输入端的上升沿，则该指令被触发执行。

  该指令的执行不影响 CR 值。

➢ PREL_M 指令使用举例

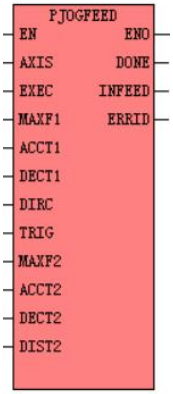
LD



LD	
描述	设定三段曲线，M0.0 上升沿启动执行本指令，按着设定好的速度位置关系运行。 

9.3.3.2.9 PJOGFEED（中断定长）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	适用于
LD	PJOGFEED			☑ MK ☑ K6

参数	输入/输出	数据类型	允许的内存区
AXIS	输入	INT	常量（0、1、2、3）
EXEC	输入	BOOL	I、Q、V、M、L、SM
MAXF1	输入	DWORD	M、V、L
ACCT1	输入	WORD	M、V、L
DECT1	输入	WORD	M、V、L
DIRC	输入	INT	M、V、L
TRIG	输入	BOOL	I、Q、V、M、L、SM
MAXF2	输入	DWORD	M、V、L
ACCT2	输入	WORD	M、V、L
DECT2	输入	WORD	M、V、L
DIST2	输入	DINT	M、V、L
DONE	输出	BOOL	Q、M、V、L
INFEED	输出	BOOL	Q、M、V、L
ERRID	输出	BYTE	Q、M、V、L

下表对各个参数的作用进行了详细的描述。

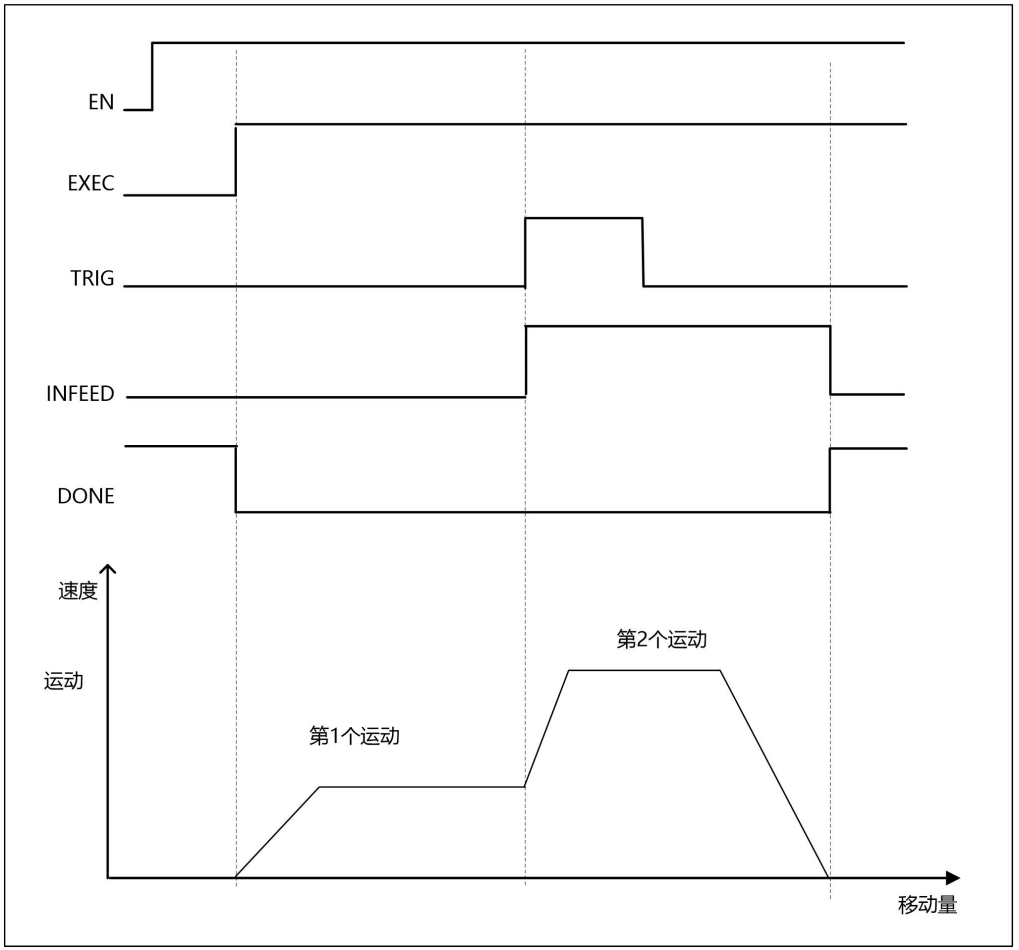
参数	描　述
AXIS	所用的高速输出通道。 0 表示使用 Q0.0，1 表示使用 Q0.1，2 表示使用 Q0.4，3 表示使用 Q0.5。
EXEC	*EXEC* 的上升沿触发本指令开始执行；*EXEC* 的下降沿触发停止运动。
MAXF1	第 1 个运动（速度模式）的最高速度。单位：Hz。
ACCT1	第 1 个运动（速度模式）的加速时间。单位：ms。
DECT1	第 1 个运动（速度模式）的减速时间。单位：ms。
DIRC	运动方向，0 表示正转，非 0 值表示反转。
TRIG	第 2 个运动的触发信号。在执行第 1 个运动的过程中，若检测到 TRIG 的上升沿，则本指令立即开始执行第 2 个运动（定长运动）。
MAXF2	第 2 个运动（定长运动）的最高速度。单位：Hz。
ACCT2	第 2 个运动（定长运动）的加速时间。单位：ms。
DECT2	第 2 个运动（定长运动）的减速时间。单位：ms。
DIST2	第 2 个运动（定长运动）的移动量，也就是输出的脉冲个数。
DONE	完成标志位。当指令正常执行完成时，*DONE* 由 0 跳变到 1。
INFEED	定长运动标志位。1 表示正在执行定长运动；0 表示未执行或已执行完成定长运动。
ERRID	错误码

如果 *EN* 为 1，则本指令被扫描。当检测到 *EXEC* 的上升沿时，本指令锁存 *MAXF1*、*ACCT1*、*DECT1*、*MAXF2*、*ACCT2*、*DECT2*、*DIST2* 等输入参数，并立即开始执行第 1 个运动（速度模式），在加速时间 *ACCT1* 内加速至最高速度 *MAXF1* 并以 *MAXF1* 维持匀速运动。在第 1 个运动期间，如果检测到 *TRIG* 的上升沿，则本指令立即开始执行第 2 个运动（定长运动），从当前位置开始走完指定的 *DIST2* 个脉冲之后就停止运动，第 2 个运动的加速时间为 *ACCT2*，最高速度为 *MAXF2*，减速时间 *DECT2*。在第 2 个运动开始执行时，*INFEED* 参数立即输出为 1，在第 2 个运动执行完成后，*INFEED* 参数立即输出为 0。两个运动的方向都由参数 *DIRC* 指定，0 表示正转，非 0 值表示反转。

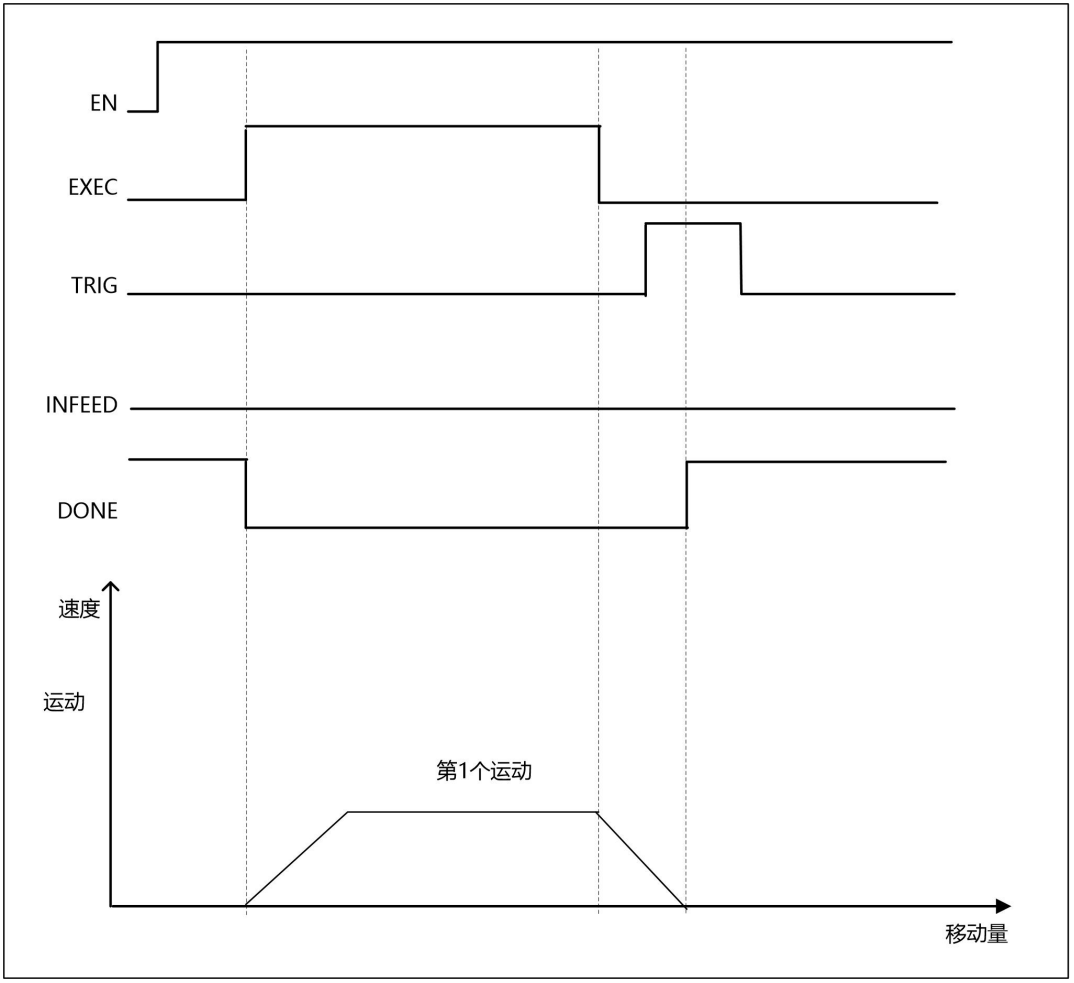
当 *EN* 为 1 期间检测到 *EXEC* 的下降沿时，或者 *EN* 变为 0 时，则本指令立即停止运动，此时若正在执行第 1 个运动，则转到减速段按减速时间 *DECT1* 减速停止；若正在执行第 2 个运动，则转到减速段按减速时间 *DECT2* 减速停止。

下面将列出常见情况下运动的时序图。

- **PJOGFEED 时序图 --- 完整的执行过程**

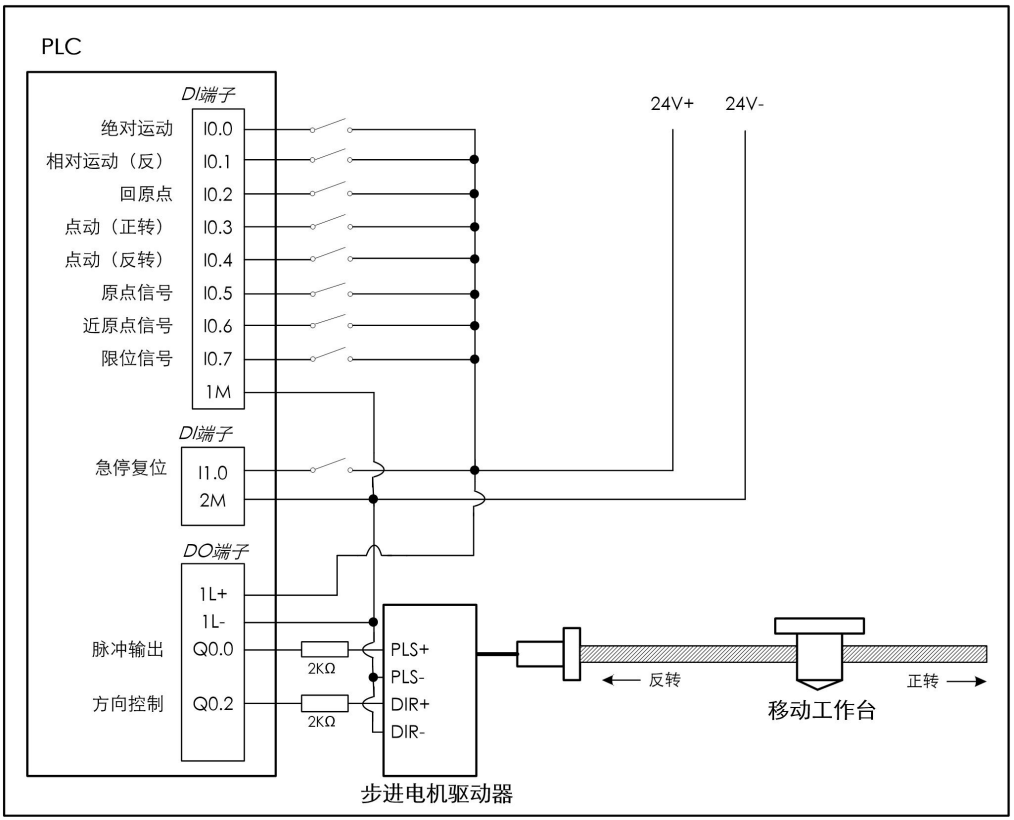


➢ PJOGFEED 时序图 --- 在第 1 个运动期间，EXEC 变为 0 导致停止运动



9.3.3.3 定位控制指令示例

➢ 接线示意图

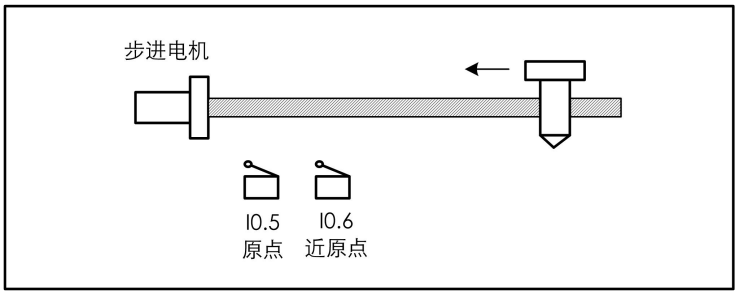
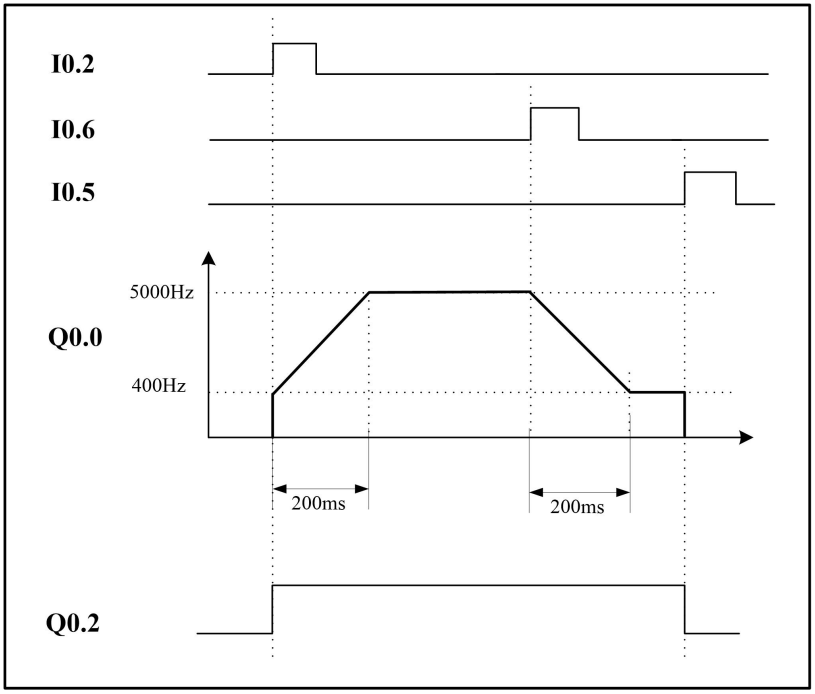


下面我们将依据这个系统来举例描述 PHOME、PREL、PABS、PJOG 和 PSTOP 指令的使用方式。

本例使用的是 Q0.0 输出通道，请注意相关特殊寄存器的使用，详见 9.3.2.2 控制寄存器和状态寄存器

➢ 回原点

1、I0.2 所接的“回原点”信号用于启动回原点运动。需要注意的是：当回原点运动完成后，当前值寄存器（SMD212）并不自动清零，用户需要根据实际要求自行改变当前值。

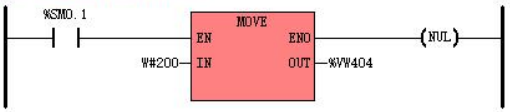
描述	假定当前工作台处于如下状态：  若检测到 I0.2 的上升沿，则在 Q0.0 输出脉冲，Q0.2 输出为 1 表示反转，电机反向找原点。 

LD

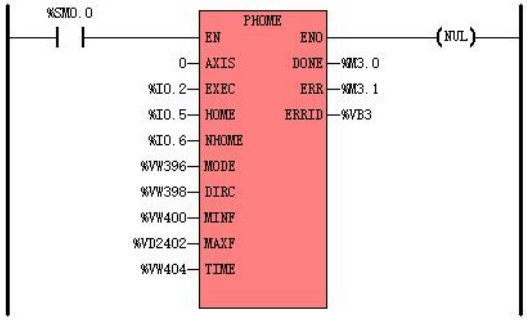
```
(* Network 0 *)
(* 同时利用近原点和原点信号，让电机反转 *)
```


```
(* Network 1 *)
(* 设置初始频率、运行频率 *)
```


```
(* Network 2 *)
(* 设置加减速时间 *)
```

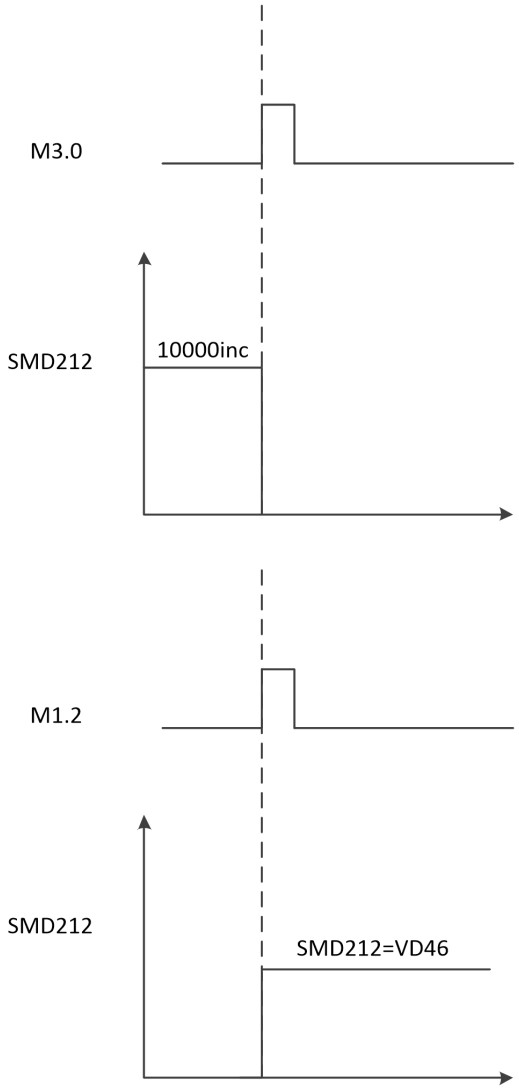


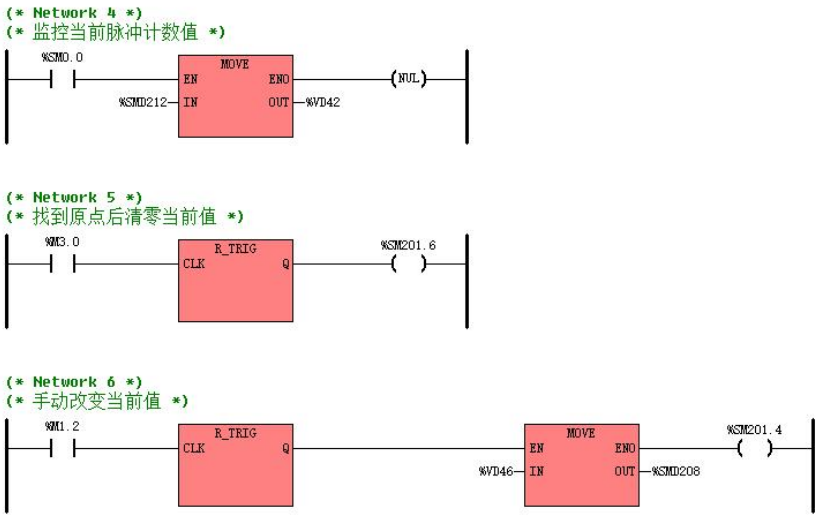

```
(* Network 3 *)
(* 调用回原点指令 *)
```



IL	(* Network 0 *)
	(*同时利用近原点和原点信号，让电机反转*)
	LD %SM0.1
	MOVE 0, %VW396
	MOVE 1, %VW398
	(* Network 1 *)
	(*设置初始频率、运行频率*)
	LD %SM0.1
	MOVE W#400, %VW400
	MOVE DW#5000, %VD2402
	(* Network 2 *)
	(*设置加减速时间*)
	LD %SM0.1
	MOVE W#200, %VW404
	(* Network 3 *)
	(*调用回原点指令*)
	LD %SM0.0
	PHOME
	0, %I0.2, %I0.5, %I0.6, %VW396, %VW398, %VW400, %VD2402, %VW404, %M3.0, %M3.1, %VB3

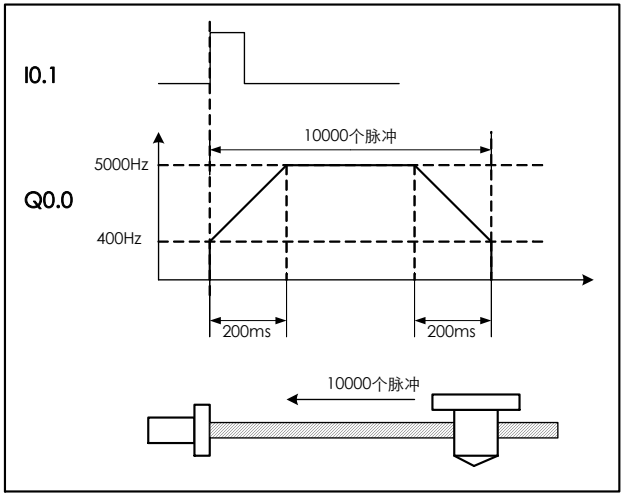
2、在回原点完成后可以对当前脉冲计数值清零或者赋予想要给定的值（当然修改当前脉冲计数值并非只能在回原点后修改）。

描述	假设找到原点前当前脉冲计数值为 10000，找到原点后瞬间清零。M1.2 为手动修改当前值，请结合下面程序观看此时序图。 

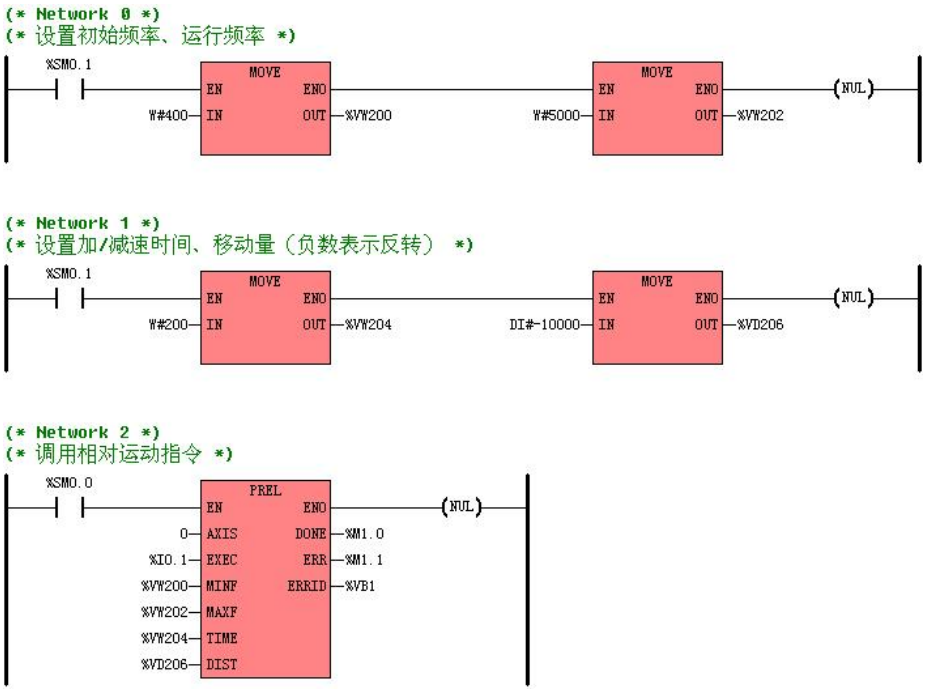
<table>
<tr><td>LD</td><td>监控当前脉冲计数值，找到原点后当前值自动清零，也可以使用 M1.2 配合相应的寄存器手动对当前值进行修改为想设定的值。
 
</td></tr>
<tr><td>IL</td><td>(* Network 4 *)
(*监控当前脉冲计数值*)
LD %SM0.0
MOVE %SMD212, %VD42
(* Network 5 *)
(*找到原点后清零当前值*)
LD %M3.0
R_TRIG
ST %SM201.6
(* Network 6 *)
(*手动改变当前值*)
LD %M1.2
R_TRIG
MOVE %VD46, %SMD208
ST %SM201.4</td></tr>
</table>

➢ 相对运动（反转）

I0.1 所接的“相对运动（反）”信号用于启动相对运动（反转）。

描述	若检测到 I0.1 的上升沿，则在 Q0.0 输出脉冲，Q0.2 输出为 1 表示反转。 

LD

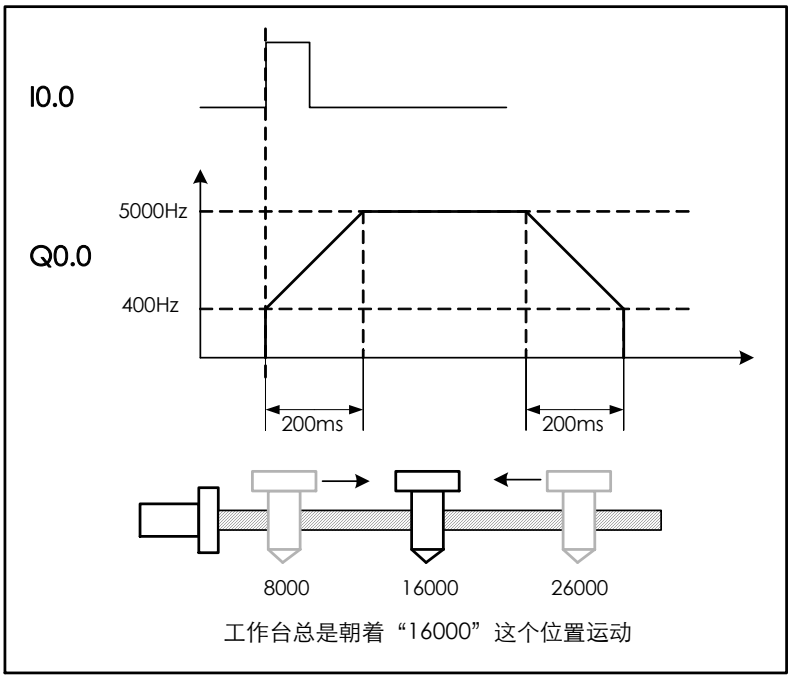


IL

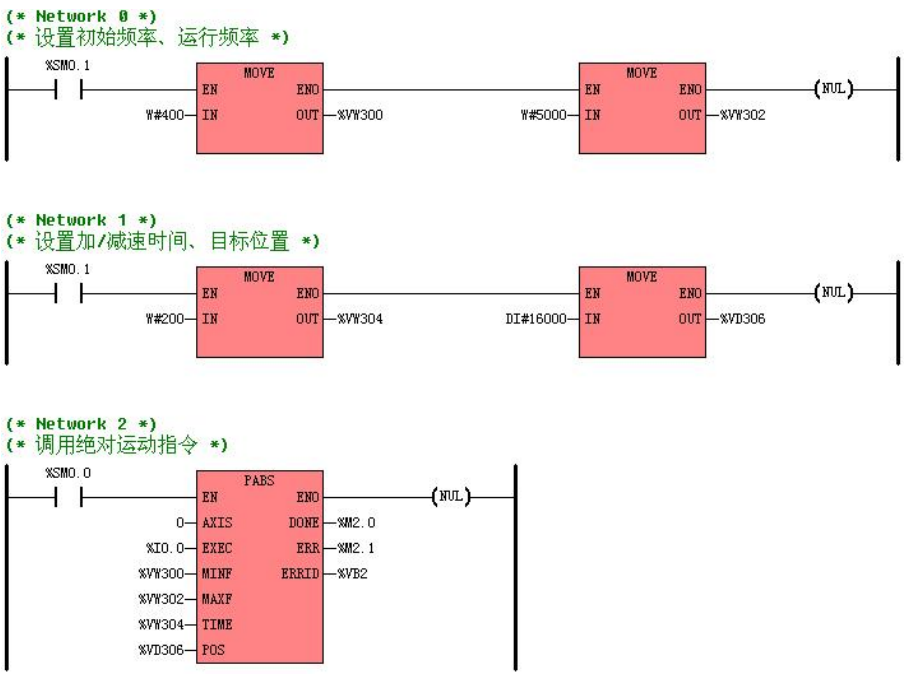
```
(* Network 0 *)
(*设置初始频率、运行频率*)
LD      %SM0.1
MOVE    W#400, %VW200
MOVE    W#5000, %VW202
(* Network 1 *)
(*设置加/减速时间、移动量（负数表示反转）*)
LD      %SM0.1
MOVE    W#200, %VW204
MOVE    DI#-10000, %VD206
(* Network 2 *)
(*调用相对运动指令*)
LD      %SM0.0
PREL    0, %I0.1, %VW200, %VW202, %VW204, %VD206, %M1.0, %M1.1, %VB1
```

➢ 绝对运动

I0.0 所接的“绝对运动”信号用于启动绝对运动。

描述	若检测到 I0.0 的上升沿，则在 Q0.0 输出脉冲。若 Q0.2 输出为 0 表示正转，输出为 1 表示反转。  工作台总是朝着“16000”这个位置运动

LD

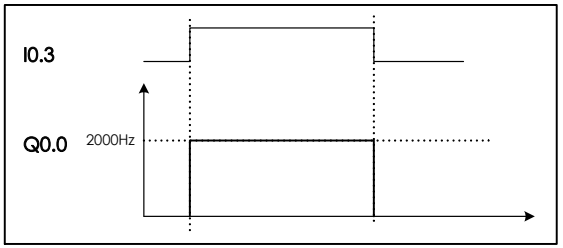
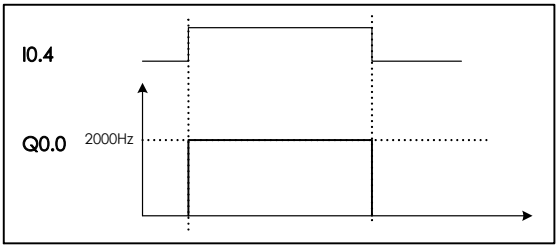


IL

```
(* Network 0 *)
(*设置初始频率、运行频率*)
LD      %SM0.1
MOVE    W#400, %VW300
MOVE    W#5000, %VW302
(* Network 1 *)
(*设置加/减速时间、目标位置*)
LD      %SM0.1
MOVE    W#200, %VW304
MOVE    DI#16000, %VD306
(* Network 2 *)
(*调用绝对运动指令*)
LD      %SM0.0
PABS    0, %I0.0, %VW300, %VW302, %VW304, %VD306, %M2.0, %M2.1, %VB2
```

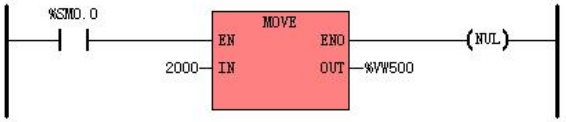
➢ 点动

I0.3 所接的“点动（正转）”信号用于点动（电机正转）。I0.4 所接的“点动（反转）”信号用于点动（电机反转）。若 I0.3 和 I0.4 同时接通，则采用上一次的电机转向。

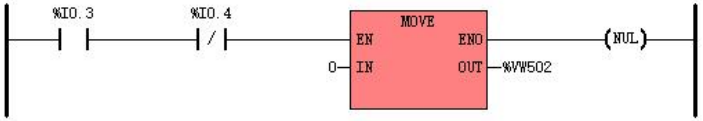
描述	若 I0.3 为 1，则 Q0.0 持续输出脉冲串。Q0.2 输出为 0 表示电机正转。若 I0.3 为 0，则 Q0.0 立即停止输出。  若 I0.4 为 1，则 Q0.0 持续输出脉冲串。Q0.2 输出为 1 表示电机反转。若 I0.4 为 0，则 Q0.0 立即停止输出。 

LD

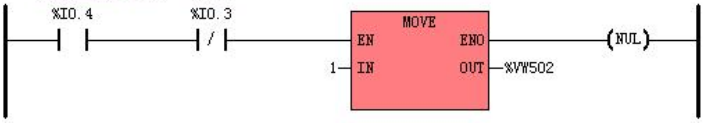
```
(* Network 0 *)
(* 设置运行频率2000HZ *)
```




```
(* Network 1 *)
(* 设置电机运行方向正转 *)
```

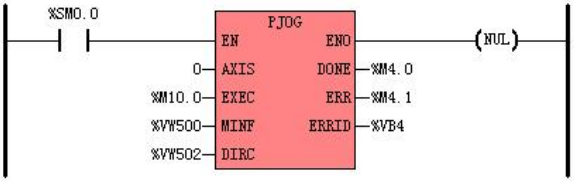



```
(* Network 2 *)
(* 设置电机方向：反转 *)
```




```
(* Network 3 *)
(* 点动 *)
```

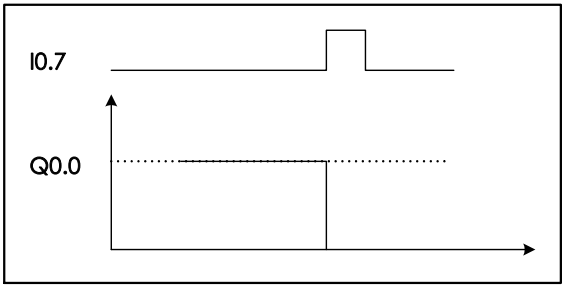
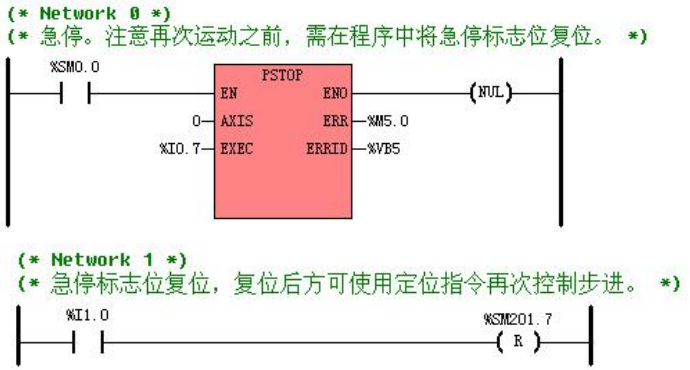

```
(* Network 4 *)
```



IL	(* Network 0 *)
	(*设置运行频率 2000HZ*)
	LD %SM0.1
	MOVE 2000, %VW500
	(* Network 1 *)
	(*设置电机方向正转*)
	LD %I0.3
	ANDN %I0.4
	MOVE 0, %VW502
	(* Network 2 *)
	(*设置电机方向：反转*)
	LD %I0.4
	ANDN %I0.3
	MOVE 1, %VW502
	(* Network 3 *)
	(*点动*)
	LD %I0.3
	OR %I0.4
	ST %M10.0
	(* Network 4 *)
	LD %SM0.0
	PJOG 0, %M10.0, %VW500, %VW502, %M4.0, %M4.1, %VB4

➢ 急停和急停复位

在丝杠两端有两个限位开关,这两个开关并联接入 I0.7 作为急停信号。程序中调用 PSTOP 指令后，如果需要再次使用定位指令控制步进电机，需要对急停标志位进行复位，I1.0 为急停复位信号。

<table>
<tr><td>描述</td><td>若检测到 I0.7 的上升沿，则 Q0.0 立即停止输出。
 
</td></tr>
<tr><td>LD</td><td> 
</td></tr>
<tr><td>IL</td><td>(* Network 0 *)
(*急停。注意再次运动之前，需在程序中将急停标志位复位。*)
LD %SM0.0
PSTOP 0, %I0.7, %M5.0, %VB5
(* Network 1 *)
(*急停标志位复位，复位后方可使用定位指令再次控制步进。*)
LD %I1.0
R %SM201.7</td></tr>
</table>

9.4 PLS 指令的使用

PLS 指令可以实现 PTO 或者 PWM 输出功能。

- PTO：Pulse Train Output，脉冲串输出，固定 50%的占空比，不可调节，但脉冲数和周期可以控制，通过控制 PTO 输出的脉冲数和周期可以达到控制电机转速和定位的效果。
- PWM：Pulse-Width Modulation，脉宽调制，可以调节脉宽（占空比 0-100%）和周期，通过调节脉宽和周期可以达到控制输出电压的效果，可以用来控制电机转速转矩、调节灯光亮度等，也可以配合 PID 指令调节占空比达到控温控压等效果。

9.4.1 高速脉冲输出功能（PTO/PWM）

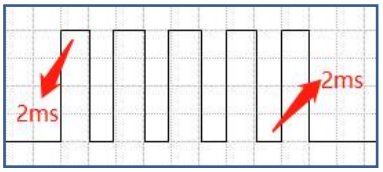
Kinco-K 系列 PLC 最多支持 4 路高速脉冲输出，相应地就提供了 4 个 PTO/PWM 脉冲发生器用于产生 PTO/PWM 输出。其中，第一个脉冲发生器分配在 Q0.0，称为 PWM0 或者 PTO0；第二个分配在 Q0.1，称为 PWM1 或者 PTO1；第三个分配在 Q0.4，称为 PWM2 或者 PTO2；第四个分配在 Q0.5，称为 PWM3 或者 PTO3。

PTO/PWM 发生器和 DO 映像寄存器共同使用内存地址 Q0.0、Q0.1 和 Q0.4、Q0.5。如果用户程序中调用了某个通道的高速输出指令，那么 PTO/PWM 发生器将控制输出通道，并禁止普通 DO 的输出。

注意：若Q0.0、Q0.1、 Q0.4、Q0.5是继电器类型的，则避免使用高速脉冲输出功能！

(1) PTO

PTO 功能能够产生指定脉冲个数的脉冲串方波（可以理解为 50%占空比的 PWM 波）。用户可以控制输出方波的周期和输出脉冲的个数。脉冲周期的单位是微秒（μs ）或者毫秒（ms），最大周期值为 65535。脉冲个数的范围是：2～4,294,967,295。如果指定脉冲数小于 2，则 CPU 将设置相应的错误标志位并禁止输出。



上图为周期 4ms，脉冲个数为 5 的脉冲串方波波形图

PTO 功能提供了单段操作和多段操作两种模式。

- **单段操作**

在单段操作模式下，每次执行 PLS 指令后仅会进行一次脉冲串输出，所以需要为下一个脉冲串更新特殊寄存器。一旦启动了起始 PTO 段，就必须按照第二个信号波形的要求改变特殊寄存器，在第一段执行完成后再次触发 PLS 指令执行下一个脉冲串。

- **多段操作**

在多段操作模式下，CPU 自动从 V 区的包络表中读出每个 PTO 段的设定值并依据设定值执行该段 PTO。

各段在包络表中的设置均占用 8 个字节，包括一个周期值（16 位无符号整数）、保留值（暂时未用到，16 位符号整数）和一个脉冲个数值（32 位无符号双整数）。也就是说，在同一段中，所有脉冲的输出频率是相同的。多段操作使用 PLS 指令来配置并启动。

包络表的起始位置存储在 SMW168（对应 PTO0）、SMW178（对应 PTO1）、SMW218（对应 PTO2）和 SMW248（对应 PTO3）中，时基通过 SM67.3（对应 PTO0）、SM77.3（对应 PTO1）、SM97.3（对应 PTO2）、SM107.3（对应 PTO3）设置，可以选择微秒或毫秒。包络表中的所有周期值必须使用同一个时基，并且在包络执行时不能改变。

包络表的格式如下表所示。

字节偏移[1]	长度	段数	描述
0	8 位		段数（1 到 64）
1	16 位	第 1 段	初始周期（2 到 65535 时基）
3	16 位		保留
5	32 位		脉冲个数（1 到 4,294,967,295）
9	16 位	第 2 段	初始周期（2 到 65535 时基）
11	16 位		保留
13	32 位		脉冲数（1 到 4,294,967,295）
…		…	…

所有偏移量均是相对于包络表起始位置的偏移字节数。

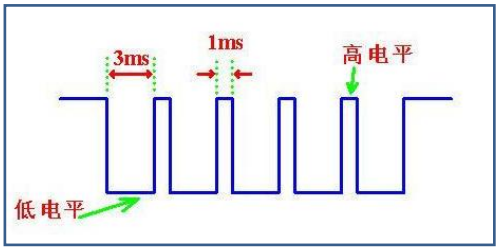
注意:包络表的起始位置必须为 V 区中的奇数地址，如 VB3001。且应注意最大范围不应超出内存

允许最大区域，否则产生不可预知错误

(2) PWM

PWM 功能提供占空比可调的连续脉冲输出。用户可以控制输出的周期和脉宽。

周期和脉宽的单位可以选择微秒（μs）或毫秒（ms），最大周期值为 65535。当脉宽大于等于周期时，占空比自动地被设为 100%，输出一直接通。当脉宽为 0 时，占空比为 0%，输出断开。



上图为周期 4ms，高电平（脉宽）1ms，低电平 3ms，占空比为 25%的波形图

有两个方法改变 PWM 信号波形的特性：

1、同步更新：如果不要求改变时间基准，则可以使用同步更新。利用同步更新，信号波形特性的变化发生在周期边沿，提供平滑转换。

2、异步更新：通常，对于 PWM 操作，脉冲宽度在周期保持不变时变化，所以不要求改变时间基准。但是，如果需要改变 PTO/PWM 发生器的时间基准，就要使用异步更新。异步更新会造成 PTO/PWM 功能被瞬时禁止，和 PWM 信号波形不同步。这会引起被控设备的振动。由于这个原因，建议采用 PWM 同步更新。选择一个适合于所有周期时间的时间基准。

9.4.2 PTO/PWM 控制寄存器和状态寄存器

在 SM 区中为每个 PTO/PWM 发生器均提供了一些控制寄存器用于存放其配置数据。如下表。

Q0.0	Q0.1	Q0.4	Q0.5	描述	
SM67.0	SM77.0	SM97.0	SM107.0	PTO/PWM	是否更新周期值：0=否；1=是
SM67.1	SM77.1	SM97.1	SM107.1	PWM	是否更新脉宽值：0=否；1=是
SM67.2	SM77.2	SM97.2	SM107.2	PTO	是否更新脉冲个数：0=否；1=是

SM67.3	SM77.3	SM97.3	SM107.3	PTO/PWM	时基：0=1μs；1=1ms
SM67.4	SM77.4	SM97.4	SM107.4	PWM	更新方法：0=异步更新；1=同步更新
SM67.5	SM77.5	SM97.5	SM107.5	PTO	操作方式：0=单段操作；1=多段操作
SM67.6	SM77.6	SM97.6	SM107.6	功能选择：0= PTO；1=PWM	
SM67.7	SM77.7	SM97.7	SM107.7	PTO/PWM	允许或禁止此功能：0=禁止；1= 允许
Q0.0	**Q0.1**	**Q0.4**	**Q0.5**	**描述**	
SMW68	SMW78	SMW98	SMW108	PTO/PWM	周期值，范围 2~65535
SMW70	SMW80	SMW100	SMW110	PWM	脉宽值，范围 0~65535
SMD72	SMD82	SMD102	SMD112	PTO	脉冲个数，范围 1~4,294,967,295
SMW168	SMW178	SMW218	SMW248	包络表的起始位置(用相对于 VB0 的字节偏移来表示)，仅用于 PTO 多段操作。	

所有控制字节、周期、脉冲数的缺省值都是 0。用户修改 PTO/PWM 波形的特性的方法是：首先设置相应的控制寄存器，如果是 PTO 多段操作，包络表也得先设置好，然后再执行 PLS 指令。

在 SM 区中也为每个 PTO/PWM 发生器均提供了一个状态字节，用户可以通过访问状态字节来了解 PTO/PWM 发生器的当前状态信息。如下表。

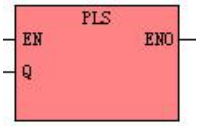
Q0.0	**Q0.1**	**Q0.4**	**Q0.5**	**描述**
SM66.0	SM76.0	SM96.0	SM106.0	保留
SM66.1	SM76.1	SM96.1	SM106.1	保留
SM66.2	SM76.2	SM96.2	SM106.2	保留
SM66.3	SM76.3	SM96.3	SM106.3	PWM 是否空闲：0=否；1=是
SM66.4	SM76.4	SM96.4	SM106.4	PTO 周期值、脉冲个数设置是否有错误：0=否；1=是 **注：周期值、脉冲个数值必须大于 1。**
SM66.5	SM76.5	SM96.5	SM106.5	PTO 是否由于用户命令而终止：0=否；1=是
SM66.6	SM76.6	SM96.6	SM106.6	保留
SM66.7	SM76.7	SM96.7	SM106.7	PTO 是否空闲：0＝忙；1＝空闲

PTO 空闲位、PWM 空闲位指明了 PTO 输出、PWM 输出是否已经结束。

9.4.3 PLS 指令

PLS 指令位于【指令集】->【计数器】组中。

指令及其操作数说明

	名称	指令格式	影响 CR 值	适用于
LD	PLS			☑ K5 ☑ K2 ☑ KS ☑ KW ☑ HP ☑ MK ☑ K6
IL	PLS	PLS *Q*	U	

参数	输入/输出	数据类型	允许的内存区
Q	输入	INT	常量（0、1、2、3）

PLS 指令的作用是：读取 SM 区中相应控制寄存器的值并配置高速脉冲输出的特性，然后启动高速脉冲输出，直到完成指定的脉冲输出功能。脉冲输出通道由参数 *Q* 指定，0 表示使用 Q0.0 输出，1 表示使用 Q0.1 输出，2 表示使用 Q0.4 输出，3 表示使用 Q0.5 输出。

注意：用户程序中，仅在需要时执行一次 PLS 指令即可，建议利用边沿指令的输出结果来调用 PLS 指令。若 PLS 的 EN 端一直保持为 1，那么 PLS 指令将无法正常输出。

(1) **LD**

若 *EN* 值为 1，则 PLS 指令被执行。

(2) **IL**

若 CR 值为 1，则 PLS 指令被执行。该指令的执行不影响 CR 值。

9.4.4 使用 PTO 功能

下面以 PTO0 为例来介绍如何编程使用 PTO 功能。

总体上，使用 PTO 包括两个步骤：

1、设置相关的控制寄存器，初始化 PTO；

2、执行 PLS 指令。

建议用户在工程中尽量编写单独的初始化子程序，这样可以使整个用户工程具有良好的结构。另外，若有可能的话，尽量在主程序中以 SM0.1 为条件来调用这个初始化子程序，这样该子程序将只在 CPU 上电后的首次扫描中调用并执行一次，可以减少 CPU 的扫描时间。

9.4.4.1 PTO（单段操作）

9.4.4.1.1 执行 PTO

1) 根据期望的操作来设置控制字节 SMB67。

   例如，SMB67 = B#16#85 表明了：

   - 允许 PTO/PWM 功能；
   - 选择使用 PTO 功能，单段操作；
   - 时基选择为 1μs；
   - 允许更新脉冲个数和周期值。

2) 将期望的周期值赋给 SMW68。

3) 将期望的脉冲个数赋给 SMD72。

4) （可选）使用 *ATCH* 指令为“PTO0 完成”中断事件（事件号 27）连接一个中断服务程序以实现对该中断事件的快速响应。

5) 调用 m0.0 上升沿根据初始化 PTO 参数执行 PLS 指令启动 PTO0。

9.4.4.1.2 改变 PTO 周期

按照如下步骤来改变 PTO0 周期值：

1） 根据期望的操作来设置控制字节 SMB67:

例如，SMB67 = B#16#81 表明了：

- 允许 PTO/PWM 功能；
- 选择使用 PTO 功能，单段操作；
- 时基选择为 1μs；
- 允许更新周期值。

2） 将期望的周期值赋给 SMW68。

3） 执行 PLS 指令来配置并启动 PTO0，具有新周期值的 PTO 就会立即接着启动。

9.4.4.1.3 改变 PTO 脉冲个数

按照如下步骤来改变 PTO0 输出的脉冲个数：

1） 根据期望的操作来设置控制字节 SMB67:

例如，SMB67 = B#16#84 表明了：

允许 PTO/PWM 功能；

选择使用 PTO 功能，单段操作；

时基选择为 1μs；

允许更新脉冲个数。

2） 将期望的脉冲个数赋给 SMD72。

3） 执行 PLS 指令来配置并启动 PTO0，就会立即接着输出新指定个数的脉冲。

9.4.4.1.4 PTO 使用举例

通过不同的条件选择不同的 PTO 输出脉冲和周期：

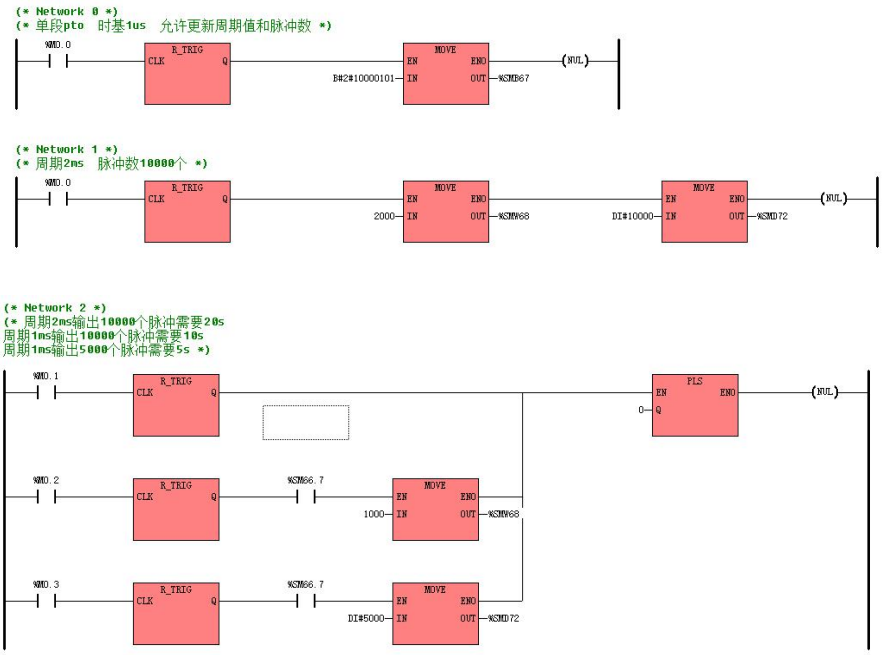
M0.0 首次初始化特殊寄存器设置周期和脉冲。

M0.1 启动 PTO 输出。

M0.2 修改周期重新启动 PTO 输出生效。

M0.3 修改输出脉冲数重新启动 PTO 输出生效。

LD


(* Network 0 *)
(* 单段pto 时基1us 允许更新周期值和脉冲数 *)
%M0.0
R_TRIG
CLK
Q
MOVE
EN
ENO
2#10000101
IN
OUT
%SMB67
(NUL)
(* Network 1 *)
(* 周期2ms 脉冲数10000个 *)
%M0.0
R_TRIG
CLK
Q
MOVE
EN
ENO
2000
IN
OUT
%SMW68
MOVE
EN
ENO
DI#10000
IN
OUT
%SMD72
(NUL)
(* Network 2 *)
(* 周期2ms输出10000个脉冲需要20s
周期1ms输出10000个脉冲需要10s
周期1ms输出5000个脉冲需要5s *)
%M0.1
R_TRIG
CLK
Q
PLS
EN
ENO
0
Q
(NUL)
%M0.2
R_TRIG
CLK
Q
%SM66.7
MOVE
EN
ENO
1000
IN
OUT
%SMW68
%M0.3
R_TRIG
CLK
Q
%SM66.7
MOVE
EN
ENO
DI#5000
IN
OUT
%SMD72

IL	(* Network 0 *)
	(*单段 pto 时基 1us 允许更新周期值和脉冲数*)
	LD %M0.0
	R_TRIG
	MOVE B#2#10000101, %SMB67
	(* Network 1 *)
	(*周期 2ms 脉冲数 10000 个*)
	LD %M0.0
	R_TRIG
	MOVE 2000, %SMW68
	MOVE DI#10000, %SMD72
	(* Network 2 *)
	(*周期 2ms 输出 10000 个脉冲需要 20s 周期 1ms 输出 10000 个脉冲需要 10s 周期 1ms 输出 5000 个脉冲需要 5s*)
	LD %M0.1
	R_TRIG
	OR(
	LD %M0.2
	R_TRIG
	AND %SM66.7
	MOVE 1000, %SMW68
	)
	OR(
	LD %M0.3
	R_TRIG
	AND %SM66.7
	MOVE DI#5000, %SMD72
	)
	PLS 0

9.4.4.2 PTO（多段操作）

9.4.4.2.1 执行 PTO

1） 根据期望的操作来设置控制字节 SMB67。

例如，SMB67 = B#16#A0 表明了：

- 允许 PTO/PWM 功能；
- 选择使用 PTO 功能
- 选择多段操作；
- 时基选择为 1μs；

2） 将包络表的起始位置（奇数，表示包络表起始地址相对于 VB0 的字节偏移）赋给 SMW168。

设置包络表中的相关数值。

3） （可选）使用 *ATCH* 指令为“PTO0 完成”中断事件（事件号 27）连接一个中断服务程序以实现对该中断事件的快速响应。

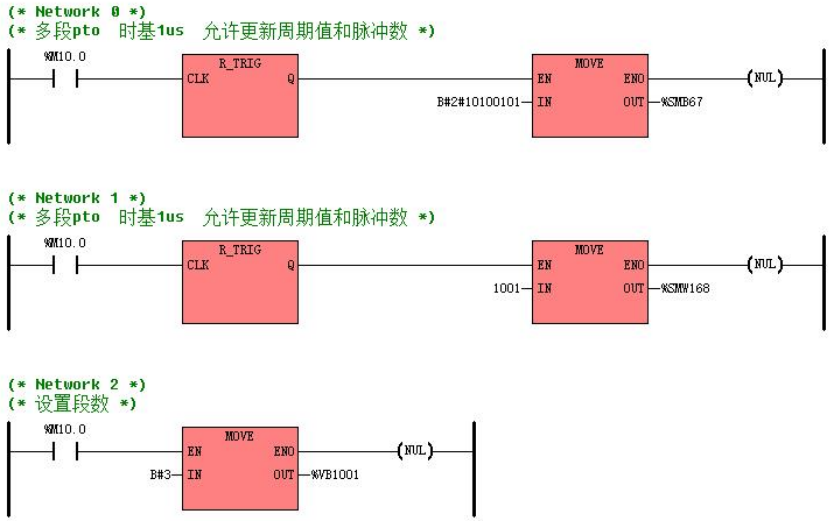
4） 执行 PLS 指令来配置并启动 PTO0。

9.4.4.2.2 使用举例

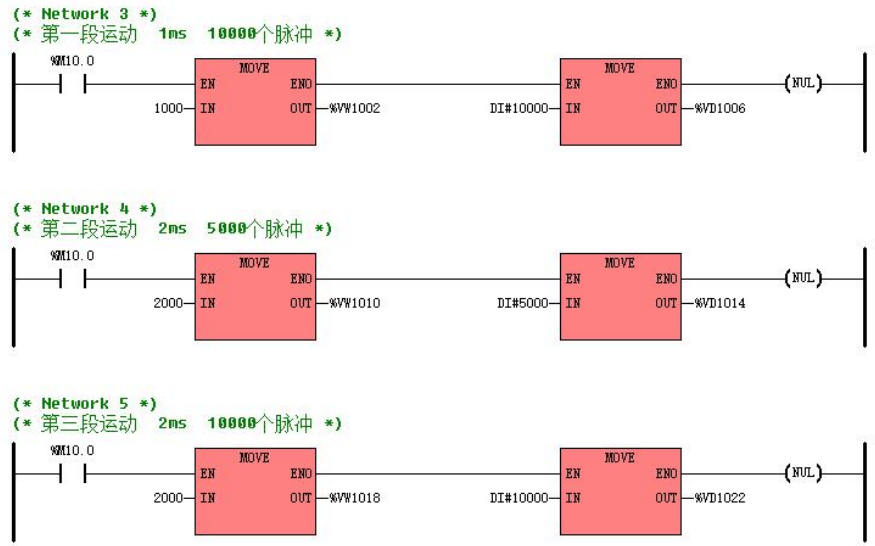
例子中在包络表设置了三段 PTO 的参数来实现步进电机的三段运动。包络表如下：

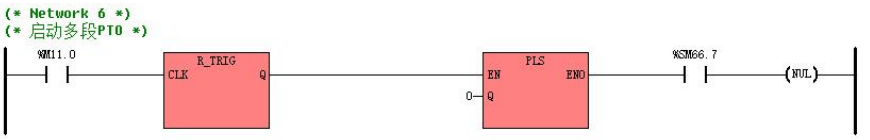
段数	描述	
第 1 段	周期	1ms
	保留	
	脉冲个数	10000
第 2 段	周期	2ms
	保留	
	脉冲数	5000
第 3 段	周期	2ms
	保留	
	脉冲数	10000

LD


(* Network 0 *)
(* 多段pto 时基1us 允许更新周期值和脉冲数 *)
%M10.0
R_TRIG
CLK
Q
MOVE
EN
ENO
B#2#10100101
IN
OUT
%SMB67
(NUL)
(* Network 1 *)
(* 多段pto 时基1us 允许更新周期值和脉冲数 *)
%M10.0
R_TRIG
CLK
Q
MOVE
EN
ENO
1001
IN
OUT
%SMW168
(NUL)
(* Network 2 *)
(* 设置段数 *)
%M10.0
MOVE
EN
ENO
B#3
IN
OUT
%VB1001
(NUL)

LD


(* Network 3 *)
(* 第一段运动 1ms 10000个脉冲 *)
%M10.0
MOVE
EN
ENO
1000
IN
OUT
%VW1002
MOVE
EN
ENO
DI#10000
IN
OUT
%VD1006
(NUL)
(* Network 4 *)
(* 第二段运动 2ms 5000个脉冲 *)
%M10.0
MOVE
EN
ENO
2000
IN
OUT
%VW1010
MOVE
EN
ENO
DI#5000
IN
OUT
%VD1014
(NUL)
(* Network 5 *)
(* 第三段运动 2ms 10000个脉冲 *)
%M10.0
MOVE
EN
ENO
2000
IN
OUT
%VW1018
MOVE
EN
ENO
DI#10000
IN
OUT
%VD1022
(NUL)

LD	
IL	(* Network 0 *) (*多段 pto　时基 1us　允许更新周期值和脉冲数*) LD　　%M10.0 R_TRIG MOVE　　B#2#10100101, %SMB67 (* Network 1 *) (*多段 pto　时基 1us　允许更新周期值和脉冲数*) LD　　%M10.0 R_TRIG MOVE　　1001, %SMW168 (* Network 2 *) (*设置段数*) LD　　%M10.0 MOVE　　B#3, %VB1001
IL	(* Network 3 *) (*第一段运动　1ms　10000 个脉冲*) LD　　%M10.0 MOVE　　1000, %VW1002 MOVE　　DI#10000, %VD1006 (* Network 4 *) (*第二段运动　2ms　5000 个脉冲*) LD　　%M10.0 MOVE　　2000, %VW1010 MOVE　　DI#5000, %VD1014

IL	(* Network 5 *) (*第三段运动　2ms　10000 个脉冲*) LD　　%M10.0 MOVE　　2000, %VW1018 MOVE　　DI#10000, %VD1022 (* Network 6 *) (*启动多段 PTO*) LD　　%M11.0 R_TRIG PLS　　0 AND　　%SM66.7 ST　　%BR0.0

9.4.5 使用 PWM 功能

总体上，使用 PWM 包括两个步骤：

1、设置相关的控制寄存器，初始化 PWM 功能；

2、执行 PLS 指令。

建议用户在工程中尽量编写单独的初始化子程序，这样可以使整个用户工程具有良好的结构。另外，若有可能的话，尽量在主程序中以 SM0.1 为条件来调用这个初始化子程序，这样该子程序将只在 CPU 上电后的首次扫描中调用并执行一次，可以减少 CPU 的扫描时间。

9.4.5.1 执行 PWM

1）根据期望的操作来设置控制字节 SMB67。

例如，SMB67 = B#16#D3 表明了：

- 允许 PTO/PWM 功能；
- 选择使用 PWM 功能；
- 选择使用同步更新方式；
- 时基选择为 1μs；

- 允许更新脉冲个数和周期值。

2） 将期望的周期值赋给 SMW68。

3）将期望的脉宽值赋给 SMW70。

4） 执行 PLS 指令来配置并启动 PWM0。

9.4.5.2 改变 PWM 脉宽

下面描述了如何改变 PWM0 的脉宽：

1） 根据期望的操作来设置控制字节 SMB67。

例如，SMB67 = B#16#D2 表明了：

- 允许 PTO/PWM 功能；
- 选择使用 PWM 功能；
- 选择使用同步更新方式；
- 时基选择为 1μs；
- 允许更新脉宽值。

2) 将期望的脉宽值赋给 SMW70。

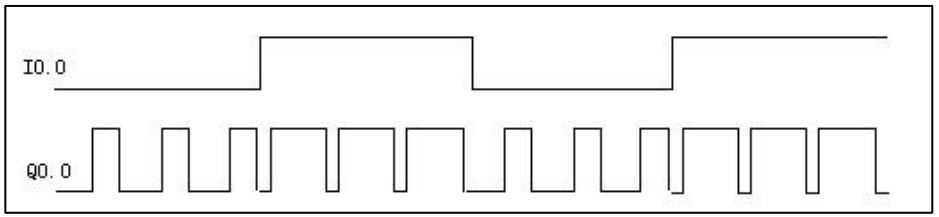
3) 执行 PLS 指令来配置并启动 PWM0。

9.4.5.3 使用举例

示例中使用了 PWM1， Q0.1 输出。周期为 10ms。

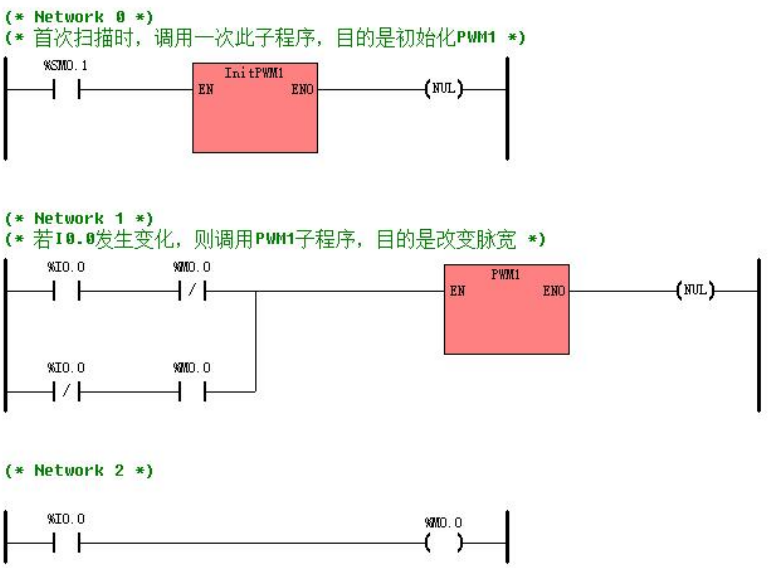
通过 I0.0 信号来改变输出脉冲的占空比，若 I0.0 为 0，则占空比为 40%;若 I0.0 为 1，则占空比为 80%;

时序示意图如下：



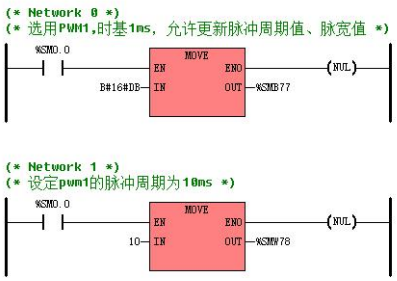
LD

主程序 MAIN：



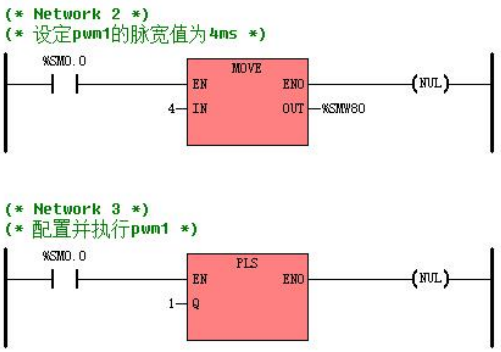
LD

子程序 InitPWM1：



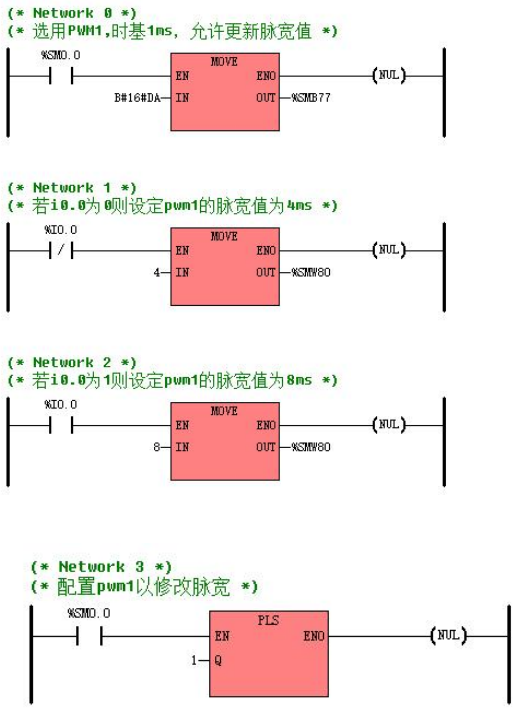
LD

子程序 InitPWM1：



LD

子程序 PWM1：



IL	主程序 MAIN： (* Network 0 *) (*首次扫描时，调用一次此子程序，目的是初始化 PWM1*) LD %SM0.1 CAL InitPWM1 (* Network 1 *) (*若 I0.0 发生变化，则调用 PWM1 子程序，目的是改变脉宽*) LD %I0.0 ANDN %M0.0 OR(LDN %I0.0 AND %M0.0) CAL PWM1 (* Network 2 *) LD %I0.0 ST %M0.0

IL	子程序 InitPWM1：
	(* Network 0 *)
	(*选用 PWM1,时基 1ms，允许更新脉冲周期值、脉宽值*)
	LD %SM0.0
	MOVE B#16#DB, %SMB77
	(* Network 1 *)
	(*设定 pwm1 的脉冲周期为 10ms*)
	LD %SM0.0
	MOVE 10, %SMW78
	(* Network 2 *)
	(*设定 pwm1 的脉宽值为 4ms*)
	LD %SM0.0
	MOVE 4, %SMW80
	(* Network 3 *)
	(*配置并执行 pwm1*)
	LD %SM0.0
	PLS 1

IL	子程序 PWM1：
	(* Network 0 *)
	(*选用 PWM1,时基 1ms，允许更新脉宽值*)
	LD %SM0.0
	MOVE B#16#DA, %SMB77
	(* Network 1 *)
	(*若 i0.0 为 0 则设定 pwm1 的脉宽值为 4ms*)
	LDN %I0.0
	MOVE 4, %SMW80
	(* Network 2 *)
	(*若 i0.0 为 1 则设定 pwm1 的脉宽值为 8ms*)
	LD %I0.0
	MOVE 8, %SMW80
	(* Network 3 *)
	(*配置 pwm1 以修改脉宽*)
	LD %SM0.0
	PLS 1

第十章 通信功能使用

10.1 功能概述

KPLC 支持多种类型的通信口，包括串行通信口（RS232 和 RS485）、CAN 通信口、以太网通信口和 LORA 无线通信口等，并且为每种类型的通信口提供了丰富的通信协议和功能。

下文将详细介绍各种通信口的功能和使用。

10.2 串行通信口的使用

CPU 模块本体提供了常用的 RS232 和 RS485 串行通信口。

下表详细说明了各系列 PLC 提供的串行通信口数量及其支持的通信协议。

系列		数量		通信协议	
		RS232	RS485	RS232	RS485
MK		--	2	编程协议、 自由通信、 Modbus RTU 从站 (3)	编程协议 (2)、 自由通信、 Modbus RTU 从站 (3)、 Modbus RTU 主站、 Kinco PLC 互联协议 (4)
K2	K209EA	1	2		
	其它	--	2		
K6		1 (1)	3 (1)		
KS		1	1		
KW		1	1		

注：（1）K6 的 CPU 模块本体提供了 2 个 RS485 通信口，用户可以使用 KB6-2COM 的 BD 板来增加 1 个 RS232 和 1 个 RS485 口。关于 BD 板的详细说明请参阅附录 G 扩展 BD 板的使用。

（2） K6 的 PORT1 和 PORT2 都支持编程协议，其它系列的仅 PORT1 支持编程协议。

（3） 所有的串行通信口都支持 Modbus RTU 从站协议，并且默认作为 Modbus RTU 从站，用户无需编程就可以直接使用。

（4） K6 的 PORT1 支持 Kinco PLC 互联协议主站及从站，MK 系列的 PORT1 支持互联协议从站，具体使用方法请参阅 10.4.4.2.2 Kinco PLC 互联协议及向导工具。

注意：在用户程序里，同一个串行通信口不允许同时使用多种协议！

CPU 模块的所有串口都统一进行编号、命名，其中 RS232 通信口编号为 0、命名为 PORT0，其它的 RS485 接口依次命名为 PORT1、PORT2 等，依此类推。另外，第 1 个串口（若有 PORT0 的话，则 PORT0 为第 1 个；若没有 PORT0 的话，则 PORT1 为第 1 个）都支持编程协议，并且为了便于处理异常情况，**第 1 个串口支持如下功能：将 CPU 模块的“RUN/STOP”开关拨至 STOP 位置，然后重新给 CPU 上电，则第 1 个串口的通信参数会自动设置为波特率 9600、无校验、8 数据位、1 停止位。**

10.2.1 自由通信

10.2.1.1 概述

所谓的自由通信，是指 CPU 通信口的通信过程及通信数据格式完全由用户程序进行控制。自由通信方式支持 ASCII 和二进制数据通信。用户可以使用自由通信方式来编写各种自定义的通信协议与其它设备进行通信。

如果通信口使用了自由通信功能后，则该通信口完全被自由通信占用，也就不能再使用其它通信协议，包括编程协议、Modbus 协议等，此时如果用户想用 KincoBuidler 软件来下载、上载程序等，则可以让 CPU 处于 STOP 状态，这样 CPU 就会停止执行自由通信指令。

10.2.1.2 自由通信指令

10.2.1.2.1 综述

下述指令均位于【指令集】->【通信指令】组中。

名称	功能描述	PLC 型号	通信口
XMT	发送数据	☑ K5 ☑ K2（K209M 除外） ☑ KS（KS101M 除外） ☑ KW（KW203 除外） ☑ MK	RS232 RS485 LORA
RCV	接收数据		

		☑ K6	
COM_XMT	发送数据	☑ K209M ☑ KS101M ☑ KW203 ☑ MK ☑ K6	RS232 RS485 LORA
COM_RCV、COM_RCV2	接收数据		
COM_RESET	复位通信口		

上表中的指令可对应分为两组：XMT、RCV 指令为一组；COM_XMT、COM_RCV、COM_RCV2 和 COM_RESET 指令为一组。两组指令适用不同型号的 CPU，下面介绍一下两组指令的异同。

XMT、RCV 指令和 COM_XMT、COM_RCV 指令优缺点对比：

1、XMT、RCV 指令使用过程较为复杂，首先需要设置自由通信寄存器以达到用户自定义协议的效果，然后需调用 XMT、RCV 指令配合通信中断进行编程。

2、使用 COM_XMT、COM_RCV、COM_RCV2 指令则不需要上述这些繁琐的过程，只需要直接调用指令并设置输入、输出参数即可，编程较为方便更易理解使用。这一组指令是后期开发的，仅一些新系列的 PLC 支持，包括 K6、MK、KM（KS101M 和 KM209）、KW203 等。

3、在通信逻辑要求校验严格的场合，使用 XMT、RCV 指令设置相应的控制寄存器可靠性更好，用户可以根据自己的实际需求进行选择。

因为串口和 LORA 无线通信口只是通信参数设置不同，指令的使用方式都是一样的，所以下面指令的使用介绍均按串口的硬件配置进行介绍，LORA 无线通信口的通信参数设置请参考 10.4 LPWAN 无线通信口的使用章节。

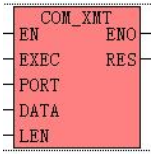
10.2.1.2.2 COM_XMT、COM_RCV 指令的使用

下面简要描述了用户使用 COM_XMT、COM_RCV 指令进行串口自由通信编程的总体步骤：

1) 配置所用串行通信口的通信参数（包括站号、波特率、奇偶校验等）。指令默认采用**【PLC 硬件配置】**中【通讯设置】定义的参数。如果操作的是 RS485 口，用户也可以使用 COM_WPARAS 指令来设置通信参数。

2) 在程序中直接调用这两个指令进行编程。需要注意的是自由通信指令均采用了半双工的方式来处理通信数据，所以接收、发送不能同时进行，运用这两个指令编程的时候请注意。

10.2.1.2.2.1 COM_XMT（发送数据）

	名称	指令格式	适用于
LD	COM_XMT		☑ KM（KS101M 和 K209M） ☑ KW203 ☑ MK ☑ K6

参数	输入/输出	数据类型	允许使用的内存区
EXEC	输入	BOOL	I、Q、V、M、L、SM
PORT	输入	INT	常量
DATA	输入	BYTE	V、M、L
LEN	输入	INT	V、M、L、常量
RES	输出	BYTE	V、M、L

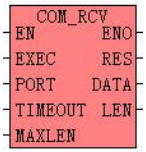
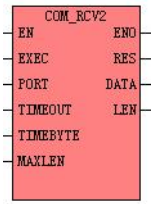
***DATA* 和 *LEN* 参数共同组成了一个内存块，该内存块内必须都是合法的变量地址。**

参数	描　述
EXEC	若检测到 *EXEC* 的上升沿跳变，则该指令被触发执行。
PORT	使用的通信口编号。0 表示 PORT0，1 表示 PORT1，2 表示 PORT2，依次类推。 若参数值指定了一个不存在的通信口，则为非法值，导致指令报错。
DATA	待发送数据的存放区域的首地址。
LEN	待发送数据的长度,单位：字节。**每次最多允许发送 248 个字节数据。**
RES	最新一次的执行结果。其组成如下： 第 7 位～指令状态。若指令正在执行则该位被置 0，当指令完成时该位立即被置 1。 第 4 位～发送错误。若发送过程中出现错误，则停止发送，该位被置 1。 第 3 位～通信口忙。若 *PORT* 口正在发送过程中，则本次不启动发送，该位被置 1。 第 2 位～数据长度错误。若 *LEN* 值等于 0 或者超过最大长度，则该位被置 1。 第 1 位～发送超时。若超过 2 秒钟没有发送完数据，则停止发送，该位被置 1。 第 0 位～非法的通信口。若 *PORT* 是非法值，则该位被置 1。 其它位～保留

- LD

当 *EN* 值为 1 时，若检测到 *EXEC* 输入端的上升沿，则该指令被触发执行一次，将用户指定的数据通过 *PORT* 口发送出去。当指令执行时，*RES* 被置为 0。当指令完成后（无论成功或者失败），*RES* 的第 7 位都会立即被置为 1。用户可以在程序中根据 *RES* 的 7 位的上升沿来判断指令是否完成，然后根据其它位表示的错误值来判断是否发送成功。

10.2.1.2.2.2 COM_RCV 和 COM_RCV2（接收数据）

	名称	指令格式	适用于
LD	COM_RCV		☑ KM（KS101M 和 K209M） ☑ KW203 ☑ MK ☑ K6
LD	COM_RCV2		

参数	输入/输出	数据类型	允许使用的内存区
EXEC	输入	BOOL	I、Q、V、M、L、SM
PORT	输入	INT	常量
TIMEOUT	输入	INT	V、M、L、常量
TIMEBYTE	输入	INT	V、M、L、常量
MAXLEN	输入	INT	V、M、L、常量
RES	输出	BYTE	V、M、L
DATA	输出	BYTE	V、M、L
LEN	输出	INT	V、M、L

注意：COM_RCV 指令的 *TIMEOUT* 和 *MAXLEN* 必须同为常量或者同为内存类型。COM_RCV2 指令的 *TIMEOUT*、*MAXLEN* 和 *TIMEBYTE* 必须同为常量或者同为内存类型。并且，*DATA* 和 *LEN* 参数共同组成了一个内存块，该内存块内必须都是合法的变量地址！

参数	描　述
EXEC	若检测到 *EXEC* 的上升沿跳变，则该指令被触发执行。
PORT	使用的通信口编号。0 表示 PORT0，1 表示 PORT1，2 表示 PORT2，依次类推。 若参数值指定了一个不存在的通信口，则为非法值，导致指令报错。
TIMEOUT	指明本次接收允许的最长时间。单位：ms。 若超过 *TIMEOUT* 值，则退出接收状态并设置 *RES* 相应的标志位。
TIMEBYTE	字节间超时。从收到第一个字节开始生效，任何两个相邻字节之间的间隔时间超过这个超时时间，就会结束一次接收，反之则收到任何字节后，重新开始计时。单位：ms。
MAXLEN	指明本次接收最多允许的字节数，***MAXLEN 值最大为 248***。单位：字节。 若 PLC 收到的字节数超过了 *MAXLEN*，则只会保留并处理最前面 *MAXLEN* 个字节。
RES	最新一次的执行结果。其组成如下： 第 7 位～指令状态。若指令正在执行则该位被置 0，当指令完成时该位立即被置 1。 第 4 位～接收错误。若接收过程中出现错误，则停止接收，该位被置 1。 第 3 位～通信口忙。若 *PORT* 口正在接收过程中，则本次不启动接收，该位被置 1。 第 2 位～*MAXLEN* 参数值错误。若 *MAXLEN* 值大于 248 或小于 0，则该位被置 1。 第 1 位～接收超时。若接收过程时间达到了 *TIMOUT*，则停止接收，该位被置 1。 第 0 位～非法的通信口。若 *PORT 是*非法值，则该位被置 1。 其它位～保留
DATA	接收数据的存放区域的首地址。
LEN	接收数据的长度,单位：字节。

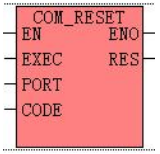
COM_RCV 和 COM_RCV2 这两条指令的功能基本一样，区别在于：COM_RCV2 可以通过 TIMEBYTE 参数来设定接收字符间的超时时间，即在接收到第一个字节之后，任意两个字节之间的超时时间，如果超过这个时间没有接收到下一个字符，则指令就会完成本次接收；而 COM_RCV 使用默认的 3.5 个字符长度的字符间超时时间。

- LD

当 *EN* 值为 1 时，若检测到 *EXEC* 输入端的上升沿，则该指令被触发执行：*PORT* 口进入接收状态，若接收到一帧报文，则停止接收并将接收的数据存放到接收缓冲区中（缓冲区的首地址为 *DATA*，长度为 *LEN*）。接收过程结束的条件是：若在“3.5 个字符长度”的时间内（指令将根据波特率自动计算，COM_RCV 后台默认使用 3.5 个字符长度，COM_RCV2 可以通过 *TIMEBYTE* 输入进行设置）没有继续接收到新的字符，则指令认为接收到了一帧完整报文并完成接收；若接收时间达到了 *TIMEOUT*，则指令也将立即完成接收。接收完成后，指令会判断接收到的字符个数，若超过 *MAXLEN*，那么指令只保留前面 *MAXLEN* 个字符，其余的全部抛弃。

当指令执行时，*RES* 被置为 0。当指令完成后（无论成功或者失败），*RES* 的第 7 位都会立即被置为 1。用户可以在程序中根据 *RES* 第 7 位的上升沿来判断指令是否完成，然后根据其它位的值来判断是否接收成功：若各位的值都为 0、第 1 位为 1 或者第 2 位为 1，均可认为接收成功。

10.2.1.2.2.3 COM_RESET（复位通信口）

	名称	指令格式	适用于
LD	COM_RESET		☑ KM（KS101M 和 K209M） ☑ KW203 ☑ MK ☑ K6

参数	输入/输出	数据类型	允许使用的内存区
EXEC	输入	BOOL	I、Q、V、M、L、SM
PORT	输入	INT	常量
CODE	输入	BYTE	V、M、L、常量
RES	输出	BYTE	V、M、L

参数	描　述
EXEC	若检测到 *EXEC* 的上升沿跳变，则该指令被触发执行。

PORT	要复位的通信口编号。0 表示 PORT0，1 表示 PORT1，2 表示 PORT2，依次类推。 若参数值指定了一个不存在的通信口，则为非法值，导致指令报错。
CODE	复位选项，其值的含义如下： 1 ～ 仅复位本通信口使用的缓存、状态等所有软件变量。 2 ～ 复位本通信口使用的所有软件变量，同时对芯片进行硬件复位和初始化。 其它值均为非法值，将导致指令报错。
RES	最新一次的执行结果。其组成如下： 第 7 位～指令状态。若指令正在执行则该位被置 0，当指令完成时该位立即被置 1。 第 1 位～非法的复位选项。若 *CODE* 是非法值，则该位被置 1。 第 0 位～非法的通信口。若 *PORT* 是非法值，则该位被置 1。

- LD

当 *EN* 值为 1 时，若检测到 *EXEC* 输入端的上升沿，则该指令被触发执行一次,对 *PORT* 通信口按 *CODE* 选项进行复位。当指令执行时，*RES* 被置为 0。当指令完成后（无论成功或者失败），*RES* 的第 7 位都会立即被置为 1。用户可以在程序中根据 *RES* 的第 7 位来判断指令是否完成，然后根据其它位表示的错误值来判断是否成功。

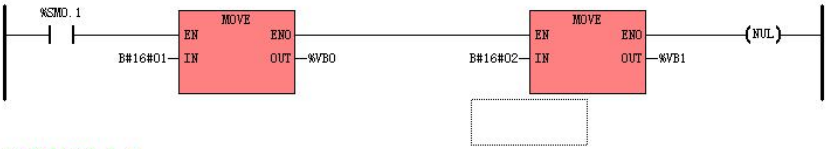
10.2.1.2.2.4 指令使用举例

下面将举例说明自由通信指令 COM_XMT 和 COM_RCV 的使用。

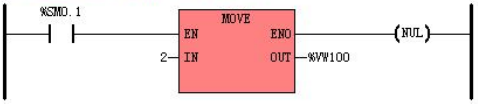
示例中为PLC的PORT1口和温湿度传感器进行通信读取温度值,PLC周期发送读取温度的报文01 02，发送完成后进入接收状态，将接收到的报文放入指定寄存器中，用户可以按照温湿度传感器自定义的报文协议将温度值存放的寄存器找到并进行解析。

LD

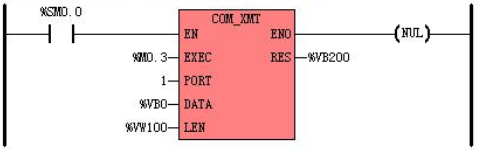
```
(* Network 0 *)
(* 读取某温度传感器的报文为 01 02
指定发送报文数据vb0=01 vb1=02 *)
```




```
(* Network 1 *)
(* 指定发送报文的数据长度 *)
```



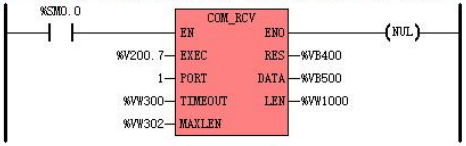

```
(* Network 2 *)
(* 每隔1s发送一次读取温度的报文 *)
```



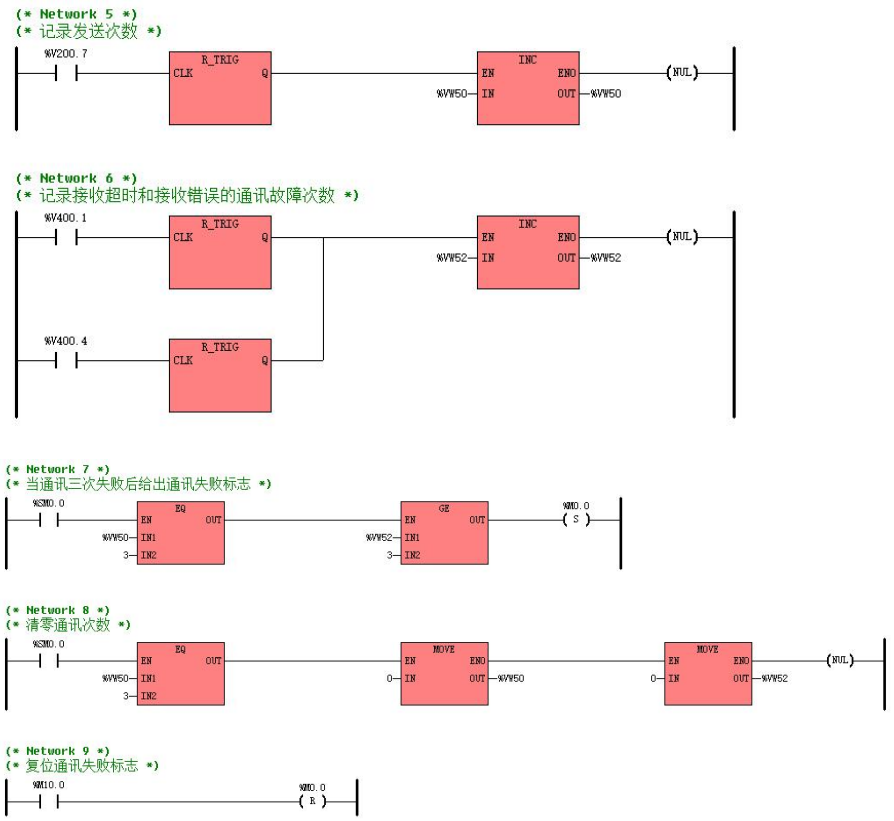
LD

```
(* Network 3 *)
(* 指定接收超时时间500ms，接收报文最大长度200个字节 *)
```


```
(* Network 4 *)
(* 将接收到的报文数据和报文长度放到指定的寄存器中 *)
```




```
(* Network 5 *)
(* 记录发送次数 *)
```

<table>
<tr><td></td><td> 
</td></tr>
<tr><td>IL</td><td>(* Network 0 *)
(*读取某温度传感器的报文为 01 02 指定发送报文数据 vb0=01 vb1=02*)
LD %SM0.1
MOVE B#16#01, %VB0
MOVE B#16#02, %VB1
(* Network 1 *)
(*指定发送报文的数据长度*)
LD %SM0.1
MOVE 2, %VW100</td></tr>
</table>

IL	(* Network 2 *)
	(*每隔 1s 发送一次读取温度的报文*)
	LD %SM0.0
	COM_XMT %M0.3, 1, %VB0, %VW100, %VB200
	(* Network 3 *)
	(*指定接收超时时间 500ms，接收报文最大长度 200 个字节*)
	LD %SM0.1
	MOVE 500, %VW300
	MOVE 200, %VW302
	(* Network 4 *)
	(*将接收到的报文数据和报文长度放到指定的寄存器中*)
	LD %SM0.0
	COM_RCV %V200.7, 1, %VW300, %VW302, %VB400, %VB500, %VW1000
	(* Network 5 *)
	(*记录发送次数*)
	LD %V200.7
	R_TRIG
	INC %VW50
	(* Network 6 *)
	(*记录接收超时和接收错误的通讯故障次数*)
	LD %V400.1
	R_TRIG
	OR(
	LD %V400.4
	R_TRIG
	)
	INC %VW52

IL	(* Network 7 *)
	(*当通讯三次失败后给出通讯失败标志*)
	LD %SM0.0
	EQ %VW50, 3
	GE %VW52, 3
	S %M0.0
	(* Network 8 *)
	(*清零通讯次数*)
	LD %SM0.0
	EQ %VW50, 3
	MOVE 0, %VW50
	MOVE 0, %VW52
	(* Network 9 *)
	(*复位通讯失败标志*)
	LD %M10.0
	R %M0.0

10.2.1.2.3 XMT、RCV 指令的使用

下面简要描述了用户使用 XMT、RCV 指令进行串口自由通信编程的总体步骤：

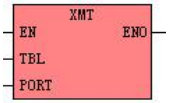
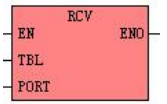
1) 在【PLC 硬件配置】设置所用通信口的串行通信参数（包括站号、波特率、奇偶校验等）。具体请参阅 4.3.4.1 CPU 参数配置中的描述。

2) 在程序中设置相应串口的自由通信控制寄存器（定义起始接收字符、结束接收字符、超时时间等），即自定义 PLC 和其他设备通信的协议，详情见 10.2.1.2.3.2 状态寄存器、控制寄存器中的寄存器定义说明。注意不同串口（PROT0、PROT1、PROT2）对应的控制寄存器地址不同。

3) 调用 XMT、RCV 指令并配合串口自由通信的状态寄存器并配合通信中断进行编程，指令的详细使用请参考下文的描述和示例程序。

10.2.1.2.3.1 XMT、RCV 指令（发送、接收数据）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	XMT			☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
	RCV			
IL	XMT	XMT TBL, PORT	U	
	RCV	RCV TBL, PORT		

参数	输入/输出	数据类型	允许使用的内存区
TBL	输入	BYTE	I、Q、M、V、L、SM
PORT	输入	INT	常量（0-2）

注意，TBL 参数为一个可变长度的内存块参数， 整个块内存都不允许超出有效的内存范围， 否则结果不可预期。

XMT 指令用于发送存放在数据缓冲区中的数据。参数 *PORT* 定义了所用通信口(0 表示 PORT0，1 表示 PORT1，依次类推)。参数 *TBL* 定义了数据缓冲区的起始地址，缓冲区的第一个字节中定义了本次将要发送的字节数（1--255），后边依次存放着待发送的数据字节。若发送字节数被设置为 0，则 XMT 指令不执行任何操作。

RCV 指令用于接收数据并将接收到的数据存放在数据缓冲区中。参数 *PORT* 定义了所用通信口(0 表示 PORT0，1 表示 PORT1，依次类推)。参数 TBL 定义了数据缓冲区的起始地址，缓冲区的第一个字节中存放着本次接收到的字节数，后边依次存放着接收到的有效数据字节。

- LD

  若 *EN* 值为 1，则执行 XMT、RCV 指令，否则不执行。

- IL

  若 CR 值为 1，则执行 XMT、RCV 指令，否则不执行。这些指令的执行不影响 CR 值。

10.2.1.2.3.2 通信中断

KPLC 提供了多种**串口自由通信中断，仅用于配合 XMT 和 RCV 指令使用**。若用户需要了解更多关于中断的信息，请参阅 6.12.1 Kinco-K 系列如何处理中断事件。

用户可以使用控制位 SM87.1、SM187.1 或 SM287.1 来禁止或允许 CPU 产生通信中断。若将中断控制位设置为 1，则允许生成通信中断：CPU 在发送完缓冲区中的最后一个字符时就会产生一个发送完成中断；CPU 在退出接收后（无论是正常还是异常退出）就会产生一个接收完成中断。

10.2.1.2.3.3 状态寄存器、控制寄存器

KPLC 在 SM 区中为自由通信提供了多个状态寄存器和控制寄存器。在编写通信程序时，用户必须对这些控制寄存器进行设置。另外，在通信过程中 CPU 会自动对通信状态进行检测，并将检测结果写入相关的状态寄存器，用户可以读取这些状态信息并在程序中进行相应的处理。

(1) 接收状态字节

位（只读）			值	含义
PORT 0	PORT 1	PORT 2		
SM86.0	SM186.0	SM286.0	1	保留。
SM86.1	SM186.1	SM286.1	1	终止接收：达到最大接收字节数。
SM86.2	SM186.2	SM286.2	1	终止接收：接收一个字符超时。
SM86.3	SM186.3	SM286.3	1	终止接收：系统接收超时。
SM86.4	SM186.4	SM286.4	-	保留。
SM86.5	SM186.5	SM286.5	1	终止接收：接收到了用户定义的结束字符。
SM86.6	SM186.6	SM286.6	1	终止接收：参数错误，无起始条件接收或无接收结束条件等。
SM86.7	SM186.7	SM286.7	1	终止接收：用户使用了禁止接收命令。

(2) 接收控制字节

位			值	描述
PORT 0	PORT 1	PORT 2		
SM87.0	SM187.0	SM287.0	-	保留。

SM87.1	SM187.1	SM287.1	0	当发送或者接收完成时禁止生成相应的中断。
			1	当发送或者接收完成时允许生成相应的中断。
SM87.2	SM187.2	SM287.2	0	忽略 SMW92/SMW192/ SMW292 中用户定义的接收字符超时值。
			1	使用 SMW92/SMW192/ SMW292 中用户定义的接收字符超时值。
SM87.3	SM187.3	SM287.3	-	保留。
SM87.4	SM187.4	SM287.4	0	忽略 SMW90/SMW190/SMW290 中用户定义的接收准备时间。
			1	使用 SMW90/SMW190/SMW290 中用户定义的接收准备时间。
SM87.5	SM187.5	SM287.5	0	忽略 SMB89/SMB189/SMB289 中用户定义的接收结束字符。
			1	使用 SMB89/SMB189/SMB289 中用户定义的接收结束字符。
SM87.6	SM187.6	SM287.6	0	忽略 SMB88/SMB188/SMB288 中用户定义的接收开始字符。
			1	使用 SMB88/SMB188/SMB288 中用户定义的接收开始字符。
SM87.7	SM187.7	SM287.7	0	禁止接收数据。此禁止条件优先于其它所有的接收控制字。
			1	允许接收数据。

(3) 其它控制寄存器

控制字（字节）			描　述
PORT0	PORT1	PORT2	
SMB88	SMB188	SMB288	用于存放用户定义的接收起始字符。 执行 RCV 指令后，CPU 收到了起始字符就开始进入有效接收状态，之前收到的数据都会被丢弃。CPU 将起始字符作为接收的第一个有效数据。 如果要使本设置值生效，需要将 SM87.6/SM187.6/SM287.6 置为 1。
SMB89	SMB189	SMB289	用于存放用户定义的接收结束字符。 CPU 将以该字符作为接收的最后一个字节。收到该字符后，无论其它的结束条件如何 CPU 都会立刻终止接收状态。 如果要使本设置值生效，需要将 SM87.5/SM187.5/SM287.5 置为 1。
SMW90	SMW190	SMW290	用于存放用户定义的接收准备时间（范围为 0~60000ms）。 执行 RCV 指令后，经过了该时间值，CPU 将自动进入有效接收状态而不管是否收到起始字符，此后接收到的数据将被认为是有效数据。 如果要使本设置值生效，需要将 SM87.4/SM187.4/SM287.4 置为 1。

SMW92	SMW192	SMW292	用于存放用户定义的接收字符超时值（范围为 1~60000ms）。 执行 RCV 指令并进入有效接收状态后，若在此超时时间内没有接收到任何字符，CPU 就会结束接收状态，无论其它的结束条件如何。 如果要使本设置值生效，需要将 SM87.2/SM187.2/SM287.2 置为 1。
SMB94	SMB194	SMB294	用于存放用户定义的每次接收字符数（1~255）。 CPU 只要接收到了此数量个有效字符，无论其它的结束条件如何，都会马上终止接收状态。本设置值总是有效。 若该值设置为 0，则 RCV 指令将直接退出。

在串口自由通信中，另外还有一个默认的系统接收超时，时间为 60 秒，此超时值的作用如下：在执行 RCV 指令后，若在此超时时间内串口上没有收到任何数据，则 CPU 将立刻终止接收并退出 RCV 指令；另外，CPU 在进入有效接收状态后（即接收到 SMB88 中定义的起始字符或者经过了 SMW90 中定义的接收准备时间后），将优先使用用户在 SMW92 中定义的接收字符超时值，若用户没有定义，则用该系统接收超时值来决定是否终止接收。

10.2.1.3.3.4 指令使用举例

下面将举例说明串口自由通信指令 XMT 和 RCV 的使用。

在示例中，CPU 将接收一串数据，以回车符作为接收结束字符。若接收正常完成，则把接收到的数据又发送回去并再次启动接收，若是异常退出接收状态（比如通信错误、接收超时等），则忽略接收到的数据并再次启动接收。

示例中包含主程序 MAIN、子程序 InitComn、接收完成中断子程序 EndReceive、发送完成中断子程序 EndSend。

主程序 MAIN：触发接收任务，调用子程序 InitComn。

子程序 InitComn：设置控制寄存器（接收起始、结束条件、超时时间）即自定义协议，初始化自由通信，同时将通信中断号和相应的中断子程序进行绑定用于产生相应的中断后，执行对应的发送、中断子程序。

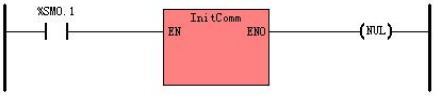
接收完成中断子程序 EndReceive：接收完成产生的中断，可以在此程序执行发送任务。

发送完成中断子程序 EndSend：发送完成产生的中断，重启接收任务。

LD

主程序（MAIN）：

```
(* Network 0 *)
(* 首次扫描时，执行一次初始化子程序，目的是初始化自由通讯。 *)
```

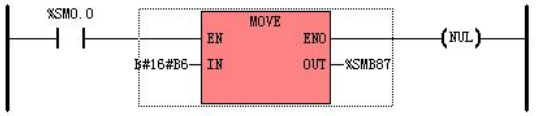



```
(* Network 1 *)
(* 利用I0.0来启动接收任务。 *)
```

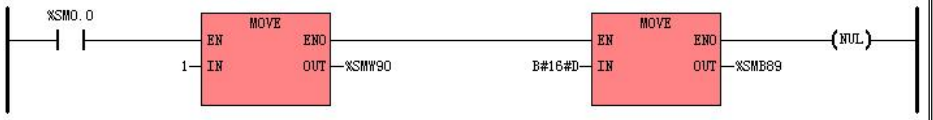
LD

InitComm（SBR00），初始化子程序：

```
(* Network 0 *)
(* 设定有效接收状态的起始条件和终止条件。 *)
```

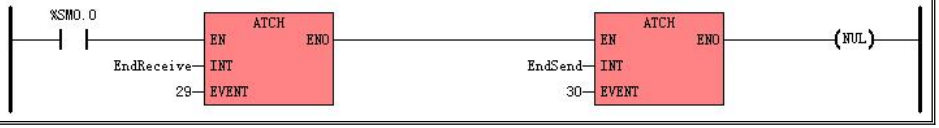



```
(* Network 1 *)
(* 设定接收准备时间为1ms，接收结束字符为回车符（ASCII为13）。 *)
```



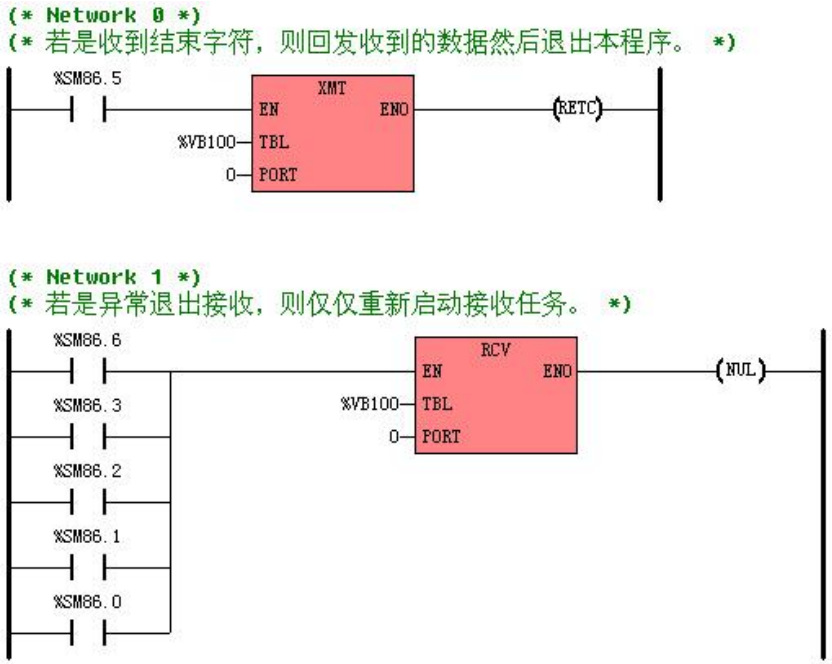

```
(* Network 2 *)
(* 设定接收字符超时为500ms，设定每次接收的字符数最大为100个。 *)
```


```
(* Network 3 *)
(* 将EndReceive程序绑定到接收完成中断，将EndSend程序绑定到发送完成中断。 *)
```

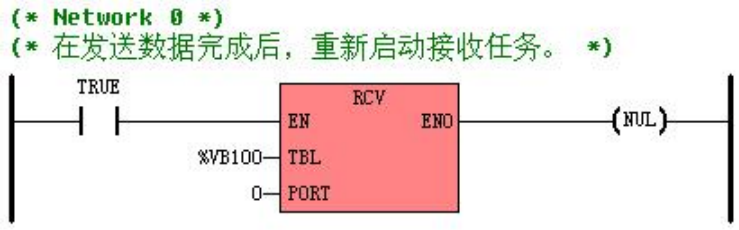


LD

EndReceive（INT00），接收完成中断服务程序：



EndSend（INT01），发送完成中断服务程序：



IL

主程序（MAIN）：

```
(* Network 0 *)
LD      %SM0.1
CAL     InitComm              (*首次扫描时，执行一次初始化子程序。*)
(* Network 1 *)
LD      %I0.0
R_TRIG
RCV     %VB100, 0             (*利用 I0.0 来启动接收任务。*)
```

InitComm（SBR00），初始化子程序：

```
(* Network 0 *)
LD      %SM0.0
MOVE    B#16#B6, %SMB87       (* 设定有效接收状态的起始条件和结束条件 *)
MOVE    1, %SMW90             (* 接收准备时间设定为 1ms *)
MOVE    B#16#D, %SMB89        (* 接收结束字符设定为回车符（ASCII 为 13）*)
MOVE    500, %SMW92           (* 接收字符超时设定为 500ms *)
MOVE    B#100, %SMB94         (* 每次接收的字符数设定为最大 100 个 *)
ATCH    EndReceive, 29        (* 将 EndReceive 程序绑定到接收完成中断 *)
ATCH    EndSend, 30           (* 将 EndSend 程序绑定到发送完成中断 *)
```

EndReive（INT00），接收完成中断服务程序：

```
(* NETWORK 0 *)
LD      %SM86.5
XMT     %VB100, 0             (* 收到结束字符则回发收到的数据 *)
RETC                          (* 然后退出本程序 *)

(* NETWORK 1 *)
LD      %SM86.0
OR      %SM86.1
OR      %SM86.2
OR      %SM86.3
OR      %SM86.6
RCV     %VB100, 0             (*若是异常退出接收状态，则重新启动接收任务*)
```

EndSend（INT01），发送完成中断服务程序：

```
(* NETWORK 0 *)
LD      TRUE
RCV     %VB100, 0             (*发送完成后重新启动接收任务*)
```

10.2.2 ModbusRTU 通信功能

10.2.2.1 概述

Modbus RTU 协议是目前使用最广泛的串行通信协议之一，它是一种主从通信协议。KPLC 的**所有串行通信口都默认作为 Modbus RTU 从站，用户无需编程即可使用**。如果如果用户需要 PLC 做 ModbusRTU 主站（仅 RS485 口支持），可以在硬件配置中进行勾选并设置通信参数。

10.2.2.2 Modbus RTU 主站可访问的 KPLC 内存区

Modbus RTU 主站可访问的 KPLC 内存区域分类如下：

类型	Modbus 功能码	对应的 PLC 内存区域
DO（开关量输出，0XXXX）	1，5，15	Q 区，M 区
DI（开关量输入，1XXXX）	2	I 区，M 区
AO（模拟量输出，4XXXX）	3，6，16	AQ 区，V 区
AI（模拟量输入，3XXXX）	4	AI 区，V 区
错误记录（16 位无符号整数）	3,4	PLC 错误记录区

一次命令最大访问的寄存器数量如下：

- 读取“位”，一次最大读取 1600 个位(200 字节)。(功能码 1、2)
- 写入“位”，一次最大写入 800 个位。(功能码 15)

3. 读取“字”，一次最大读取 100 个字。(功能码 3、4)

4. 写入“字”，一次最大写入 100 个字。(功能码 16)

注意：当操作的内存区大小小于上面的最大值时， 只允许一次操作整个内存区。比如对于 K506，外部主站一次读取 AI 区 90 个字，这是错误的，因为 AI 区最大 32 个字。

10.2.2.3 Modbus 寄存器编号

若 Modbus 主站所需的寄存器从 1 开始编号，那么将下表中的寄存器号直接加 1 即可，如果 Modbus 主站所需的寄存器也是从 0 开始编号，那么直接对应下表寄存器号。

- 适用于 K5、KS、MK 系列

内存区域	范围	类型	对应的 Modbus 寄存器号*
I	I0.0 --- I31.7	DI	0 --- 255
Q	Q0.0 --- Q31.7	DO	0 --- 255
M	M0.0 --- M1023.7	DI/DO	320 -- 8511
AI	AIW0 --- AIW62	AI	0 --- 31
AQ	AQW0 --- AQW62	AO	0 --- 31
V	VW0 --- VW4094	AI/AO	100 -- 2147

由于不同系列 PLC 的内存大小不同，所以允许访问的范围也不尽相同，比如 K6 的 V 区长度为 16K 字节，所以可以访问 VW0--VW16382；但 K2 的 V 区长度为 4K 字节，所以只可以访问 VW0--VW4094。

但是不同系列 PLC 对同一内存区域的起始 Modbus 寄存器编号是一致的，编号规律也一致，比如在所有系列 PLC 中，VW0 的寄存器号都为 100，VW2 的都为 101。

各系列 PLC 的内存区域范围请参考 3.6.4 内存区域的地址范围。

除以上内存区域外，KPLC 的错误记录的内存区域也支持通过 ModbusRTU 协议来读取，具体方式请参阅 13.3 如何读取 PLC 中曾经发生的错误。

10.2.2.4 Modbus RTU 报文基本格式

报文中的 CRC 校验码是高字节在前，低字节在后。

不小于 3.5 个字符长度的报文间隔时间	目标站号	功能码	数据	CRC 校验码
	1 字节	1 字节	N 字节	2 字节

10.2.2.5 Modbus RTU 命令简介

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

10.2.2.5.1 功能码 01：读线圈（开关量输出）

请求格式：

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

正确应答格式：

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

10.2.2.5.2 功能码 02：读输入状态（开关量输入）

请求格式：

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

正确应答格式：

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

10.2.2.5.3 功能码 03：读保持寄存器（模拟量输出）

请求格式：

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

正确应答格式：

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

10.2.2.5.4 功能码 04：读输入寄存器（模拟量输入）

请求格式：

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

正确应答格式：

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

10.2.2.5.5 功能码 05：写单线圈（开关量输出）

请求格式：

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

注：强制值＝ 0xFF00，则置线圈为接通；强制值＝0x0000，则置线圈为断开。

应答格式：若设置成功，则原报文返回

10.2.2.5.6 功能码 06：写单保持寄存器（模拟量输出）

请求格式：

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

应答格式：若设置成功，则原报文返回

10.2.2.5.7 功能码 15：写多线圈（开关量输出）

请求格式：

目标站号	功能码	起始地址		写入个数		强制值字节数	强制值第 1 字节	…	CRC
1 字节	15	高字节	低字节	高字节	低字节	1 字节	1 字节	…	2 字节

正确应答格式：

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

10.2.2.5.8 功能码 16：写多保持寄存器（模拟量输出）

请求格式：

目标站号	功能码	起始地址		写入个数		强制值字节数	强制值 1 高字节	强制值 1 低字节	…	CRC
1 字节	16	高字节	低字节	高字节	低字节	1 字节	1 字节	1 字节	…	2 字节

正确应答格式：

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

10.2.2.6 Modbus 协议中的 CRC 校验算法

在 Modbus RTU 协议中，使用 CRC 作为帧的校验方式。下面是用 C 编写的两种 CRC 算法。

10.2.2.6.1 直接计算 CRC

```
/* 参数：chData ―― const BYTE*，指向待校验数据存储区的首地址
              uNO  ――待校验数据的字节个数
   返回值：WORD 型，计算出的 CRC 值。                  */
WORD  CalcCrc(constBYTE* chData, WORDuNo)
```

```
{
WORDcrc=0xFFFF;
    WORDwCrc;
    UCHARi,j;
    for (i=0; i<uNo; i++)
    {
        crc ^= chData[i];
        for (j=0; j<8; j++)
        {
            if (crc&1)
            {
                crc>>= 1;
                crc ^= 0xA001;
            }
            else
                crc>>= 1;
        }
    }

    wCrc=( (WORD)LOBYTE(crc) )<<8;
    wCrc=wCrc|( (WORD)HIBYTE(crc) );
    return (wCrc);
}
```

10.2.2.6.2 查表快速计算 CRC

```
/* 高字节 CRC 表 */
const UCHAR auchCRCHi[] =
{
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
```

```
0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,
0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,
0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40
} ;

/* 低字节 CRC 表 */
const UCHAR auchCRCLo[] =
{
0x00, 0xC0, 0xC1, 0x01, 0xC3, 0x03, 0x02, 0xC2, 0xC6, 0x06,
0x07, 0xC7, 0x05, 0xC5, 0xC4, 0x04, 0xCC, 0x0C, 0x0D, 0xCD,
0x0F, 0xCF, 0xCE, 0x0E, 0x0A, 0xCA, 0xCB, 0x0B, 0xC9, 0x09,
0x08, 0xC8, 0xD8, 0x18, 0x19, 0xD9, 0x1B, 0xDB, 0xDA, 0x1A,
```

```
0x1E, 0xDE, 0xDF, 0x1F, 0xDD, 0x1D, 0x1C, 0xDC, 0x14, 0xD4,
0xD5, 0x15, 0xD7, 0x17, 0x16, 0xD6, 0xD2, 0x12, 0x13, 0xD3,
0x11, 0xD1, 0xD0, 0x10, 0xF0, 0x30, 0x31, 0xF1, 0x33, 0xF3,
0xF2, 0x32, 0x36, 0xF6, 0xF7, 0x37, 0xF5, 0x35, 0x34, 0xF4,
0x3C, 0xFC, 0xFD, 0x3D, 0xFF, 0x3F, 0x3E, 0xFE, 0xFA, 0x3A,
0x3B, 0xFB, 0x39, 0xF9, 0xF8, 0x38, 0x28, 0xE8, 0xE9, 0x29,
0xEB, 0x2B, 0x2A, 0xEA, 0xEE, 0x2E, 0x2F, 0xEF, 0x2D, 0xED,
0xEC, 0x2C, 0xE4, 0x24, 0x25, 0xE5, 0x27, 0xE7, 0xE6, 0x26,
0x22, 0xE2, 0xE3, 0x23, 0xE1, 0x21, 0x20, 0xE0, 0xA0, 0x60,
0x61, 0xA1, 0x63, 0xA3, 0xA2, 0x62, 0x66, 0xA6, 0xA7, 0x67,
0xA5, 0x65, 0x64, 0xA4, 0x6C, 0xAC, 0xAD, 0x6D, 0xAF, 0x6F,
0x6E, 0xAE, 0xAA, 0x6A, 0x6B, 0xAB, 0x69, 0xA9, 0xA8, 0x68,
0x78, 0xB8, 0xB9, 0x79, 0xBB, 0x7B, 0x7A, 0xBA, 0xBE, 0x7E,
0x7F, 0xBF, 0x7D, 0xBD, 0xBC, 0x7C, 0xB4, 0x74, 0x75, 0xB5,
0x77, 0xB7, 0xB6, 0x76, 0x72, 0xB2, 0xB3, 0x73, 0xB1, 0x71,
0x70, 0xB0, 0x50, 0x90, 0x91, 0x51, 0x93, 0x53, 0x52, 0x92,
0x96, 0x56, 0x57, 0x97, 0x55, 0x95, 0x94, 0x54, 0x9C, 0x5C,
0x5D, 0x9D, 0x5F, 0x9F, 0x9E, 0x5E, 0x5A, 0x9A, 0x9B, 0x5B,
0x99, 0x59, 0x58, 0x98, 0x88, 0x48, 0x49, 0x89, 0x4B, 0x8B,
0x8A, 0x4A, 0x4E, 0x8E, 0x8F, 0x4F, 0x8D, 0x4D, 0x4C, 0x8C,
0x44, 0x84, 0x85, 0x45, 0x87, 0x47, 0x46, 0x86, 0x82, 0x42,
0x43, 0x83, 0x41, 0x81, 0x80, 0x40
} ;

/* 参数：puchMsg  －－ const BYTE*，指向待校验数据存储区的首地址
usDataLen －－待校验数据的字节个数
返回值：WORD 型，计算出的 CRC 值。    */
WORD CKINCOSerialCom::CalCrcFast(const BYTE* puchMsg , WORD usDataLen)
{
    BYTE uchCRCHi = 0xFF ;  /* CRC 高字节初始化 */
    BYTE uchCRCLo = 0xFF ;  /* CRC 低字节初始化 */
     WORD uIndex ;            /* CRC 查表的索引*/
```

```
    while (usDataLen--)
    {
        uIndex = uchCRCHi ^ *puchMsg++ ;        /* 计算 CRC */
        uchCRCHi = uchCRCLo ^ auchCRCHi[uIndex];
        uchCRCLo = auchCRCLo[uIndex] ;
    }
    return (uchCRCHi << 8 | uchCRCLo) ;
}
```

10.2.2.7 Modbus RTU 主站指令

为了方便用户的应用，KPLC 提供了 Modbus RTU 主站指令（仅用于 RS485 口），用户调用这些指令就可以实现 Modbus RTU 主站的功能。

下面简要描述了用户进行 Modbus 主站编程的总体步骤：

1) 在【PLC 硬件配置】设置所用通信口的串行通信参数（包括站号、波特率、奇偶校验等），并选中【作为 Modbus 主站】项，设置超时时间，重试次数。具体请参阅 4.3.4.1 CPU 参数配置。

超 时：在此时间内，主站没有收到某从站的回应报文则认为该从站超时，单位 ms。

重 试：当主站收到某从站错误的应答或超时后，继续重新尝试与该从站进行通信的次数。

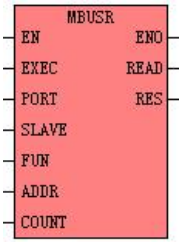
2) 在程序中直接调用 MBUSR、MBUSW 指令进行编程。

关于 Modbus RTU 协议的详细描述，请参阅 10.2.2.4 Modbus RTU 报文基本格式 。

本节描述的指令均位于【指令集】->【通信指令】组中。

10.2.2.7.1 MBUSR（Modbus 主站读）

- 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	MBUSR			☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
IL	MBUSR	MBUSR *EXEC, PORT, SLAVE, FUN, ADDR, COUNT, READ, RES*	U	

参数	输入/输出	数据类型	允许使用的内存区
EXEC	输入	BOOL	I、Q、V、M、L、SM、RS、SR
PORT	输入	INT	常量（0-2）
SLAVE	输入	BYTE	I、Q、M、V、L、SM、常量
FUN	输入	INT	常量（Modbus 功能码）
ADDR	输入	INT	I、Q、M、V、L、SM、AI、AQ、常量
COUNT	输入	INT	I、Q、M、V、L、SM、AI、AQ、常量
READ	输出	BOOL、WORD、INT	Q、M、V、L、SM、AQ
RES	输出	BYTE	Q、M、V、L、SM

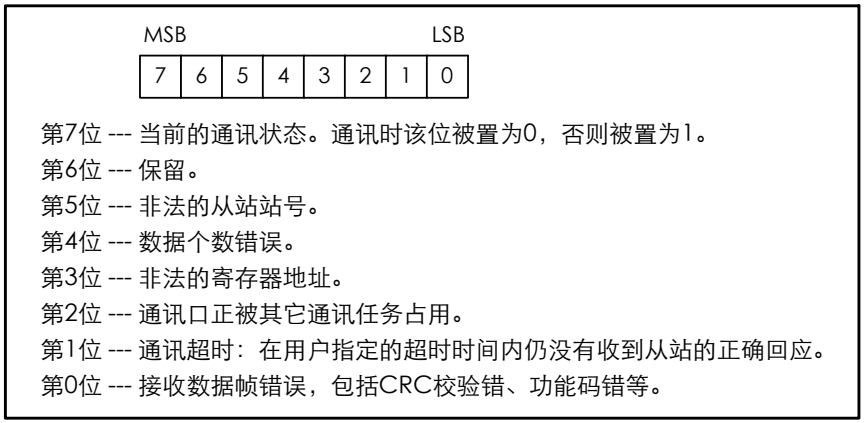
注意：参数 *SLAVE, ADDR, COUNT* 必须同时为常量类型或同时为内存类型；*READ*、*COUNT* 参数共同组成了一个内存块，整个块内的内存地址都必须为合法的地址。

KPLC 作为 Modbus RTU 主站时，MBUSR 指令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）。

参数 *PORT* 定义了所用的通信口。*SLAVE* 定义了目标从站的站号，允许的站号范围是 1~254。*FUN* 定义了功能码。*ADDR* 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个数，允许的最大值是 32。

*EXEC*输入端的上升沿跳变用于启动通信。MBUSR 指令执行时，若检测到 *EXEC*的上升沿跳变，MBUSR 就会进行一次通信：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、地址校验和功能码校验，若校验后证明报文正确，那么所需的数据就会被写入数据缓冲区中，否则接收到的报文会被丢弃。

参数 *READ* 定义了数据缓冲区的起始地址，读取的数据（个数为 *COUNT*）就存放在该区域内。*READ* 必须与功能码匹配，若功能码是 1、2，则输入 BOOL 型的地址变量；若功能码是 3、4，则输入 INT 或者 WORD 型的地址变量。参数 *RES* 用于存放当前的状态信息和最近一次通信的故障信息，它的组成如下图：



- LD

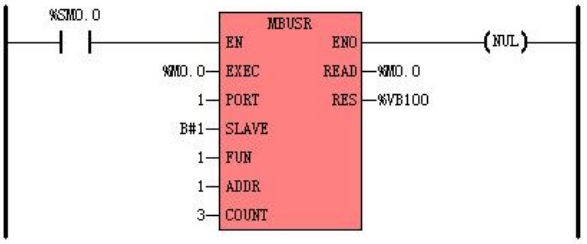
  如果 *EN* 为 1，则该指令被执行，否则不执行。

- IL

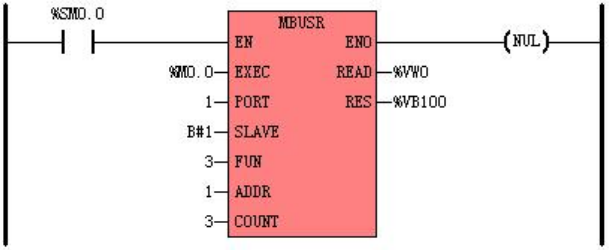
  如果 CR 值为 1，则该指令被执行，否则不执行。

  该指令的执行不影响 CR 值。

下面举例说明指令中功能码 FUN、个数 COUNT、READ 之间的关系：

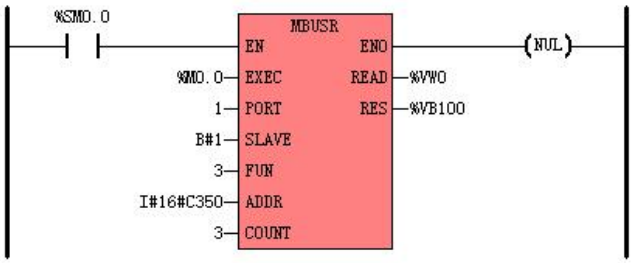


功能码 FUN 为 1、2 时，个数 COUNT 指的是读取的 bool 型变量的个数，读取到的数据存放到 READ 中，相应的 READ 要填入 bool 型的地址变量。



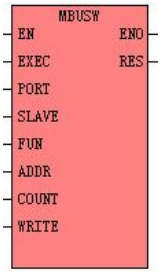
功能码 FUN 为 3、4 时，个数 COUNT 指的是读取的 INT 或者 WORD 型变量的个数，读取到的数据存放到 READ 中，相应的 READ 要填入 INT 或者 WORD 型的地址变量。

另外需要注意的是 ADDR 的输入为 INT，地址的输入范围为-32768-32767，当需要输入的地址超过 32767 后需要在输入端填写十六进制的数值，比如 ADDR 的地址给定为十进制 50000，需要按下图填写：



10.2.2.7.2 MBUSW（Modbus 主站写）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	MBUSW			☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
IL	MBUSW	MBUSW *EXEC, PORT, SLAVE, FUN, ADDR, COUNT, READ, RES*	U	

参数	输入/输出	数据类型	允许使用的内存区
EXEC	输入	BOOL	I、Q、V、M、L、SM、RS、SR
PORT	输入	INT	常量（0-2）
SLAVE	输入	BYTE	I、Q、M、V、L、SM、常量
FUN	输入	INT	常量（Modbus 功能码）
ADDR	输入	INT	I、Q、M、V、L、SM、AI、AQ、常量
COUNT	输入	INT	I、Q、M、V、L、SM、AI、AQ、常量
WRITE	输入	BOOL、WORD、INT	I、Q、RS、SR、V、M、L、SM、T、C、AI、AQ
RES	输出	BYTE	Q、M、V、L、SM

注意：参数 *SLAVE, ADDR, COUNT* 必须同时为常量类型或同时为内存类型；*READ*、*COUNT* 参数共同组成了一个内存块，整个块内的内存地址都必须为合法的地址。

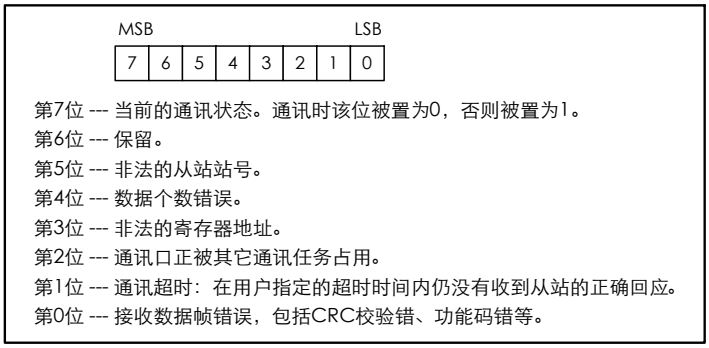
KPLC 作为 Modbus RTU 主站时，MBUSW 指令用于将数据写入从站内。该指令适用的功能码有 5（写一个 DO）、6（写一个 AO）、15（写多个 DO）和 16（写多个 AO）。

参数 *PORT* 定义了所用的通信口。*SLAVE* 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。*ADDR* 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，允许的最大值是 32。

参数 *WRITE* 定义了数据缓冲区的起始地址，要写入从站的数据就存放在该区域内。*WRITE* 必须与功能码匹配。若功能码是 5、15，则输入 BOOL 型的地址变量；若功能码是 6、16，则输入 INT 或者 WORD 型的地址变量。

EXEC 输入端的上升沿跳变用于启动通信。MBUSW 指令执行时，若检测到 *EXEC* 的上升沿跳变，MBUSW 就会进行一次通信：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、地址校验和功能码校验，判读从站是否正确执行了刚才的写命令。

参数 *RES* 用于存放当前的状态信息和最近一次通信的故障信息，它的组成如下图：



注意：如果从站站号 *SLAVE* 的值是 255，则表示本次是一次广播通信，网络中所有从站都接收到主站发出去的报文并且进行相应的“写”寄存器操作，但是不进行回应，因此主站也不会等待接收回应报文。

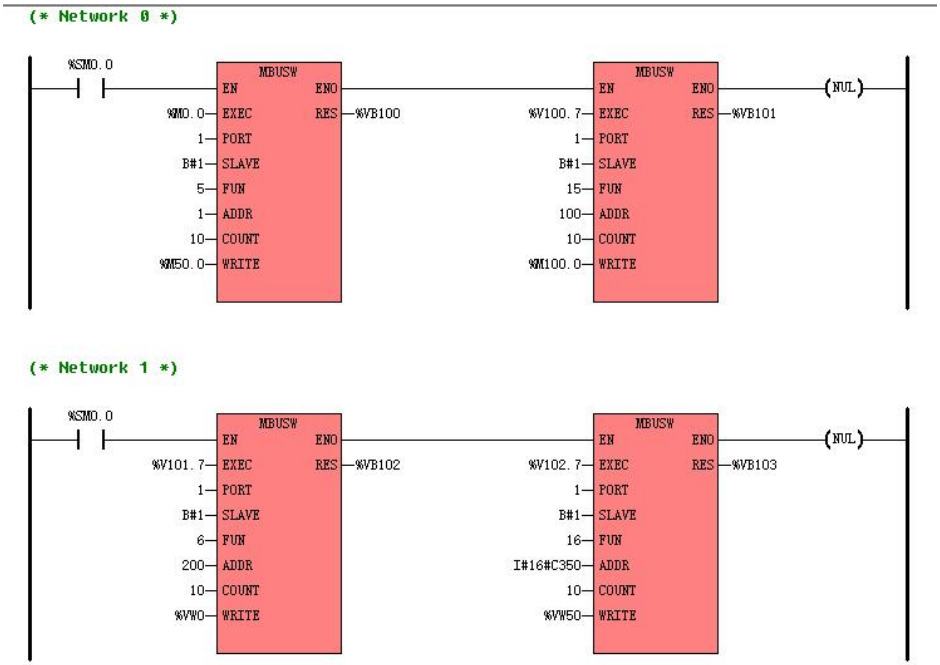
- LD

  如果 *EN* 为 1，则该指令被执行，否则不执行。

- IL

如果 CR 值为 1，则该指令被执行，否则不执行。该指令的执行不影响 CR 值。

下面举例说明指令中功能码 FUN、个数 COUNT、WRITE 之间的关系：



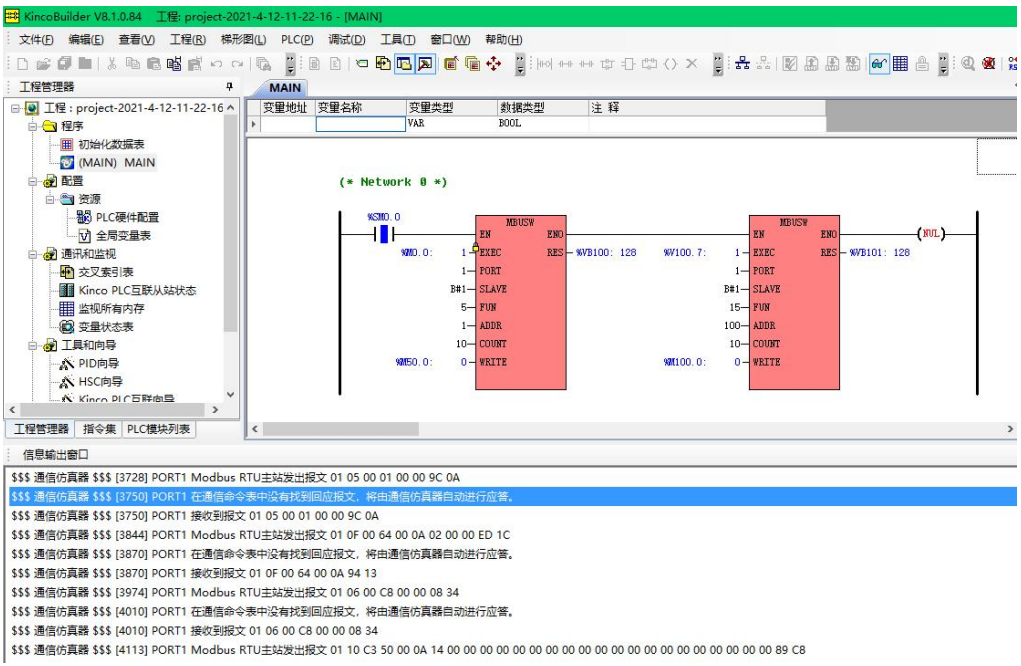
功能码 FUN 为 5、6 时，个数 COUNT 不起作用，个数固定为 1。功能码 FUN 为 15 时，个数 COUNT 指的是写入 BOOL 型变量的个数。功能码 FUN 为 16 时，个数 COUNT 指的是写入的 INT 或者 WORD 型变量的个数。

功能码 FUN 为 5、15 时，对应的 WRITE 的起始地址应为 BOOL 型的地址变量。

功能码 FUN 为 6、16 时，对应的 WRITE 的起始地址应为 INT 或者 WORD 型的地址变量。

另外需要注意的是 ADDR 的输入为 INT，地址的输入范围为-32768-32767，当需要输入的地址超过 32767 后需要在输入端填写十六进制的数值，比如 ADDR 的地址给定为十进制 50000，如上图中应写为 I#16#C350。

注意：使用 Modbus 指令与其他设备读写数据时，由于两边设备对数据的存储方式不一样，可能会需要对数据的高低字节进行交换，这样才能得到正确的结果，遇到类似情况可以通过串口调试工具，监控主从站的收发报文进行分析，KPLC 完全遵守标准 Modbus RTU 报文格式进行收发报文。也可以通过编程软件自带的离线模拟功能 4.4 通信仿真，查看指令正确的收发报文来查找问题，如下图所示：

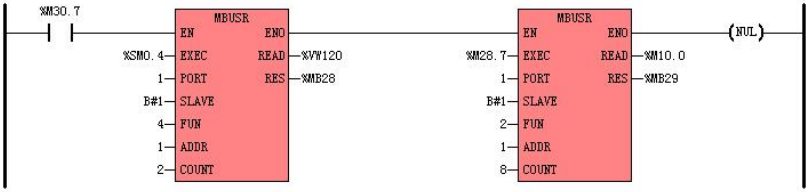


10.2.2.7.3 MBUSR、MBUSW 使用举例

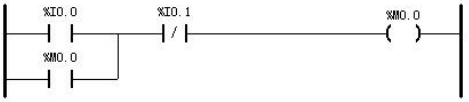
LD

```
(* Network 0 *)
(* M30.7用于标记MBUSW指令是否正在通讯，每次通讯之后会被更新。 *)
```


```
(* Network 1 *)
(* 若通讯口未被MBUSW指令占用，则执行MBUSR指令：
   每2秒钟从1#从站读取数据：先读1号和2号AI寄存器，完成后接着读1-8号DI寄存器。 *)
```

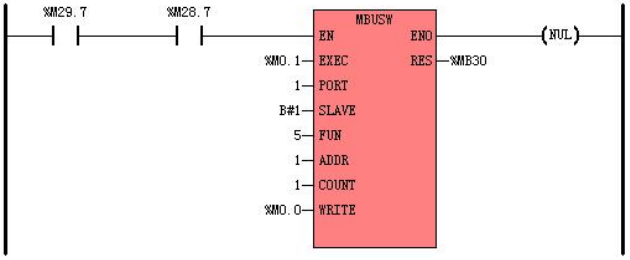



```
(* Network 2 *)
(* MBUSW指令举例 *)
```




```
(* Network 3 *)
```


```
(* Network 4 *)
(* 若通讯口未被MBUSR指令占用，则执行MBUSW指令：
   一旦I0.0或者I0.1接通，则马上写入1#从站的1#DO寄存器。 *)
```



IL	(* Network 0 *) (*M30.7 用于标记 MBUSW 指令是否正在通信，每次通信之后会被更新。*) LD %SM0.1 S %M30.7 (* Network 1 *) (*若通信口未被 MBUSW 指令占用，则执行 MBUSR 指令：*) (* 每 2 秒钟从 1#从站读取数据：*) (* 先读 1 号和 2 号 AI 寄存器，完成后接着读 1-8 号 DI 寄存器。*) LD %M30.7 MBUSR %SM0.4, 1, B#1, 4, 1, 2, %VW120, %MB28 MBUSR %M28.7, 1, B#1, 2, 1, 8, %M10.0, %MB29 (* Network 2 *) (*MBUSW 指令举例*) LD %I0.0 OR %M0.0 ANDN %I0.1 ST %M0.0 (* Network 3 *) LD %I0.0 OR %I0.1 ST %M0.1 (* Network 4 *) (*若通信口未被 MBUSR 指令占用，则执行 MBUSW 指令：*) (* 一旦 I0.0 或者 I0.1 接通，则马上写入 1#从站的 1#DO 寄存器。*) LD %M29.7 AND %M28.7 MBUSW %M0.1, 1, B#1, 5, 1, 1, %M0.0, %MB30

10.2.3 动态修改 RS485 通信口参数

10.2.3.1 概述

KPLC 默认需要在【PLC 硬件配置】中修改各个通信口的参数并下载到 PLC 中才能生效。

同时，KPLC 也提供了在用户程序中动态修改本体 RS485 口通信参数的功能。PORT0（RS232）作为编程口可能会经常使用，因此不允许动态调整通信参数。

- 允许动态修改【PLC 站号】、【波特率】和【奇偶校验】这三项参数。
- 动态修改的通信参数值存放在永久存储器中，永久有效。
- 动态修改的通信参数优先级要高于【PLC 硬件配置】中的通信参数。即使用户重新下载了新工程，PLC 也会优先采用已经存储的动态通信参数。用户可以使用【PLC】->【清除...】菜单命令或者下文提到的清除功能，来清除动态通信参数。
- 用户在程序中修改通信参数后，【PLC 站号】会立即生效，但是【波特率】和【奇偶校验】则不一定：若通信口当时正处于空闲状态，则这 2 个参数会立即生效；若通信口当时正处于通信状态，则这 2 个参数会被保存但不会立即生效。

  在 PLC 下一次上电启动时，所有修改过的通信参数都会生效。

目前修改 RS485 通信口参数有两种方式：一种是通过相关指令进行修改，另外一种是通过特殊寄存器进行修改。这两种方式适用于不同的 PLC 型号，且不同时支持。

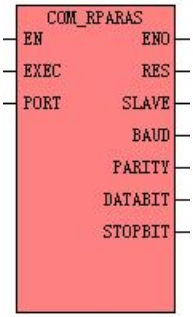
10.2.3.2 使用指令读写 RS485 通信口参数

下述指令均位于【指令集】->【通信指令】组中。

名称	功能描述	适用 PLC 型号
COM_RPARAS	读取 RS485 通信口参数	☑ K209M ☑ KS101M ☑ KW203 ☑ MK ☑ K6
COM_WPARAS	修改 RS485 通信口参数	

10.2.3.2.1 COM_RPARAS（读取 RS485 通信口参数）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	COM_RPARAS			☑ K209M ☑ KS101M ☑ KW203 ☑ MK ☑ K6
IL	COM_RPARAS	COM_RPARAS *EXEC, PORT, RES, SLAVE, BAUD, PARITY, DATABIT, STOPBIT*	U	

参数	输入/输出	数据类型	允许使用的内存区
EXEC	输入	BOOL	I、Q、V、M、L、SM
PORT	输入	INT	常量（1-2）
SLAVE	输出	BYTE	Q、M、V、L、SM
BAUD	输出	DINT	Q、M、V、L、SM
PARITY	输出	BYTE	Q、M、V、L、SM
DATABIT	输出	BYTE	Q、M、V、L、SM
STOPBIT	输出	BYTE	Q、M、V、L、SM
RES	输出	BYTE	Q、M、V、L、SM

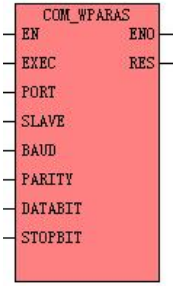
参数	描　述
EXEC	若检测到 *EXEC* 的上升沿跳变，则该指令被触发执行。
PORT	使用的通信口编号。1 表示 PORT1，2 表示 PORT2，依次类推。 若参数值指定了一个不存在的通信口，则为非法值，导致指令报错。
SLAVE	读取的站号存放的地址

BAUD	读取的波特率存放的地址
PARITY	读取的奇偶检验位存放的地址
DATABIT	读取的数据位存放的地址
STOPBIT	读取的停止位存放的地址
RES	RES 用于存放当前的状态信息和最近一次的故障信息： Bit7-指令执行正确 Bit0-不支持的 PORT 参数

*EXEC*输入端的上升沿跳变用于读取一次 RS485 口的通信参数，PORT 代表读取哪个 PORT 口的参数，读取的站号、波特率、奇偶校验、数据位、停止位依次存放在 SLAVE、BAUD、PARITY、DATABIT 、STOPBIT 对应的地址中。

10.2.3.2.2 COM_WPARAS（修改 RS485 通信口参数）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	COM_WPARAS			☑ K209M ☑ KS101M ☑ KW203 ☑ MK ☑ K6
IL	COM_WPARAS	COM_RPARAS *EXEC, PORT, SLAVE, BAUD, PARITY, DATABIT, STOPBIT, RES,*	U	

参数	输入/输出	数据类型	允许使用的内存区
EXEC	输入	BOOL	I、Q、V、M、L、SM
PORT	输入	INT	常量（1-2）
SLAVE	输入	BYTE	Q、M、V、L、SM

BAUD	输入	DINT	Q、M、V、L、SM
PARITY	输入	BYTE	Q、M、V、L、SM
DATABIT	输入	BYTE	Q、M、V、L、SM
STOPBIT	输入	BYTE	Q、M、V、L、SM
RES	输出	BYTE	Q、M、V、L、SM

参数	描　述
EXEC	若检测到 *EXEC* 的上升沿跳变，则该指令被触发执行。
PORT	使用的通信口编号。1 表示 PORT1，2 表示 PORT2，依次类推。 若参数值指定了一个不存在的通信口，则为非法值，导致指令报错。
SLAVE	需要修改的站号存放的地址，范围 1-127
BAUD	需要修改的波特率存放的地址。 波特率值可以为 1200、2400、4800、9600、19200、38400、57600、115200
PARITY	需要修改的奇偶检验位存放的地址，0-无校验、1-奇校验、2-偶校验
DATABIT	需要修改的数据位存放的地址，范围 7、8
STOPBIT	需要修改的停止位存放的地址，范围 1、2
RES	RES 用于存放当前的状态信息和最近一次的故障信息： Bit7-指令执行正确 Bit6-将这些参数写入永久存储时失败 Bit5-不支持的 STOPBIT 参数 Bit4-不支持的 DATABIT 参数 Bit3-不支持的 PARITY 参数 Bit2-不支持的 BAUD 参数 Bit1-不支持的 SLAVE 参数，目前只是不能为 0 Bit0-不支持的 PORT 参数

*EXEC*输入端的上升沿跳变用于修改一次 RS485 口的通信参数,PORT 代表修改哪个 PORT 口的参数，将存放在 SLAVE（站号）、BAUD（波特率）、PARITY（奇偶校验）、DATABIT （数据位）、STOPBIT（停止位）的参数写入到相应的 PORT 口中。

10.2.3.3 使用特殊寄存器读取修改 RS485 通信口参数

除 K209M、KS101M、KW203、MK、K6 外，其他所有型号 PLC 均需要使用特殊寄存器才能对 RS485 通信口参数进行读写。

10.2.3.3.1 寄存器说明

SMB20--SMB25 用于动态修改通信参数。下面详细描述了各个寄存器的含义。

- **参数值：SMB23、SMB24 和 SMB25**

SMB	描 述
SMB23	PLC 站号值。数值有效范围：1-31。 若执行写操作，则 SMB23 是将要写入的新 PLC 站号值；若执行读操作，则 SMB23 是读取到的当前使用的 PLC 站号值；若执行清除操作，则 SMB23 被忽略。
SMB24	波特率值。数值有效范围 0-5：0 表示 2400，1 表示 4800，2 表示 9600，3 表示 19200，4 表示 38400，5 表示 1200。 若执行写操作，则 SMB24 是将要写入的新波特率值；若执行读操作，则 SMB24 是读取到的当前使用的波特率值；若执行清除操作，则 SMB24 被忽略。
SMB25	奇偶校验值。数值有效范围 0-2，0 表示无检验，1 表示奇检验，2 表示偶检验。 若执行写操作，则 SMB25 是将要写入的新奇偶校验值；若执行读操作，则 SMB25 是读取到的当前使用的奇偶校验值；若执行清除操作，则 SMB25 被忽略。

- **控制字节：SMB20 和 SMB21**

位	描述
SMB20：指定将要操作的端口号和操作方式。	
SM20.7	值为 1 表示启动写操作。PLC 写入新参数完成后会自动将该位清 0。
SM20.6	值为 1 表示启动读操作。PLC 读取参数完成后会自动将该位清 0。
SM20.5	值为 1 表示启动清除操作。PLC 清除参数完成后会自动将该位清 0。
SM20.4	保留，必须赋值为 0。
SM20.3 SM20.0	这 4 位的组合值表示将要操作的端口号。 1 表示 PORT1，2 表示 PORT2，若设置为其它无效值则 PLC 将报错并退出操作。

SMB21：指定将要操作的通信参数。	
SM21.7 -- SM21.3	备用。必须赋值为 0。
SM21.2	值为 1 表示修改或者清除指定通信口的奇偶校验值。
SM21.1	值为 1 表示修改或者清除指定通信口的波特率值。
SM21.0	值为 1 表示修改或者清除指定通信口的 PLC 站号。

在同一时刻，SM20.5、SM20.6、SM20.7 只允许有一个位的值为 1，否则 PLC 将报错并退出操作。

当执行读操作时，SMB21 的值被忽略，PLC 将一次性读取所有的通信参数值。

当执行清除操作时，PLC 允许分别或者同时清除站号、波特率、奇偶校验值。某个参数被清除后，PLC 将会自动采用硬件配置信息中相应的通信参数值。

- **状态字节：SMB22**

SMB22 存放了本次动态调整通信参数的操作结果。

位（只读）	描述
SM22.7	值为 1 表示本次操作完成。 PLC 完成用户指定的操作后，无论成功还是失败，都会自动将 SM22.7 置 1。只有当 SM22.7 的值为 1 时，SMB22 中其它位的值才有实际意义。
SM22.6	若 SM22.7 值为 1，那么若 SM22.6 值为 1，则表示本次操作成功；若 SM22.6 值为 0，则表示本次操作因发生错误而失败。
SM20.5 SM20.0	若本次操作失败，那么这 6 位的组合值就表示发生的错误代码，含义如下表。

错误代码	错误描述
1	错误的操作命令。比如，SM20.7 和 SM20.6 被同时置为 1。
2	错误的端口号
3	SMB21（将要操作的通信参数）值错误。
4	SMB23（新 PLC 站号）值错误。
5	SMB24（新波特率）值错误。
6	SMB25（新奇偶校验）值错误。

10	从永久存储器中读取存放的 PORT1 动态 PLC 站号失败。
11	尚未设置过 PORT1 的动态 PLC 站号。
12	从永久存储器中读取存放的 PORT1 动态波特率值失败。
13	尚未设置过 PORT1 的动态波特率。
14	从永久存储器中读取存放的 PORT1 动态奇偶校验值失败。
15	尚未设置过 PORT1 的动态奇偶校验值。
20	从永久存储器中读取存放的 PORT2 动态 PLC 站号失败。
21	尚未设置过 PORT2 的动态 PLC 站号。
22	从永久存储器中读取存放的 PORT2 动态波特率值失败。
23	尚未设置过 PORT2 的动态波特率。
24	从永久存储器中读取存放的 PORT2 动态奇偶校验值失败。
25	尚未设置过 PORT2 的动态奇偶校验值。
61	动态通信参数值写入永久存储器失败。

10.2.3.3.2 使用方法

- 修改PLC的通信参数

1) 将SMB20的低4位设置为将要操作的端口号。

   比如，SMB20=B#1表示将要修改PORT1的参数。

2) 根据要修改的参数类型，将相应的值赋给 SMB21。

   比如，SMB21=B#16#03 表明了将要修改 PLC 站号和波特率值。

3) 将期望的新参数值赋给相应寄存器：SMB23 是新 PLC 站号值，SMB24 是新波特率值，SMB25 是新奇偶校验值。

   比如，SMB23=B#03 表示要将 PLC 站号修改为 3，SMB24=B#3 表示要将波特率修改为 19200。

4) （可选）若已经启动过一次通信参数的操作（读、写或者清除），那么必须先检查完成标志位SM22.7，只有SM22.7为1才可以启动本次的操作。

5) 将SM20.7赋值为1，启动本次的写操作。PLC在完成写入新参数后会自动将SM20.7清零。

6) （可选）检查标志位SM22.7和SM22.6，这两个标志位的值都为1表示修改参数成功。

➢ 读取PLC的通信参数

1) 将SMB20的低4位设置为将要操作的端口号。

比如，SMB20=B#1表示将要读取PORT1的参数。

2) （可选）若已经启动过一次通信参数的操作（读、写或者清除），那么必须先检查完成标志位SM22.7，只有SM22.7为1才可以启动本次的操作。

3) 将SM20.6赋值为1，启动本次的读操作。PLC在完成读取新参数后会自动将SM20.6清零。

4) 检查标志位SM22.7和SM22.6，这两个标志位的值都为1表示读取参数成功。读取成功后，参数值存放在如下寄存器中：SMB23是PLC站号值，SMB24新波特率值，SMB25是奇偶校验值。

➢ 清除PLC的通信参数

1) 将SMB20的低4位设置为将要操作的端口号。

比如，SMB20=B#1表示将要修改PORT1的参数。

2) 根据要清除的参数类型，将相应的值赋给 SMB21。

比如，SMB21=B#16#03 表明了将要清除永久存储器中存放的动态 PLC 站号和波特率值。

3) （可选）若已经启动过一次通信参数的操作（读、写或者清除），那么必须先检查完成标志位SM22.7，只有SM22.7为1才可以启动本次的操作。

4) 将SM20.5赋值为1，启动本次的清除操作。PLC在完成清除后会自动将SM20.5清零。

5) （可选）检查标志位SM22.7和SM22.6，这两个标志位的值都为1表示清除参数成功。

10.2.3.3.3 示例

下面例子是在HMI上修改PORT1和PORT2的动态站号，所使用PLC为KS。程序采用了IL语言，用户可以直接将其复制到KincoBuilder的编辑器中并执行【工程】->【LD语言】菜单命令转换为梯形图。

VW48是新站号值，可以在HMI上修改，同时VW48的值也被置于到KS的VW3690中永久存储（不同型号的PLC永久存储区不同具体参考12.1 数据保持和数据备份章节）。PLC程序检查VW48的实时值并与VW3690中存储的值进行比较，若数值发生变化且VW48是允许的合法值，则将VW48作为PORT1和PORT2的新站号值并启动修改。

```
(* Network 0 *)
(*上电启动时，利用永久存储的值来初始化 VW48。*)
LD      %SM0.1
MOVE    %VW3690, %VW48
(* Network 1 *)
(*判断 VB48 值是否变化，且值是否有效。然后保存 VW48 并启动修改。*)
LD      %SM0.0
GE      %VB48, B#1
LE      %VB48, B#31
NE      %VW48, %VW3690
MOVE    %VW48, %VW3690
ST      %M999.7
(* Network 2 *)
(*开始修改 PORT1 站号。*)
LD      %M999.7
R_TRIG
MOVE    B#1, %SMB20
MOVE    B#1, %SMB21
MOVE    %VB48, %SMB23
S       %SM20.7
S       %M999.6
(* Network 3 *)
(*PORT1 参数修改成功后，再修改 PORT2 站号。*)
LD      %M999.6
AND     %SM22.7
R_TRIG
AND     %SM22.6
MOVE    B#2, %SMB20
S       %SM20.7
R       %M999.6
```

10.3 以太网口的使用

10.3.1 概述

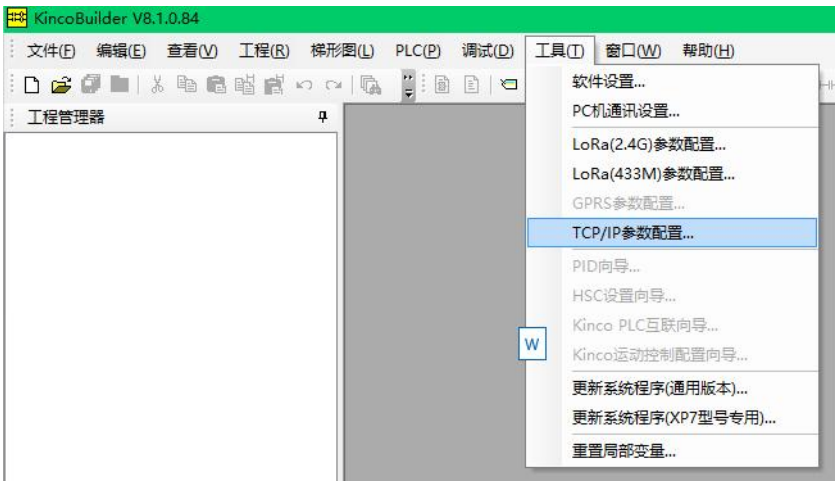
KPLC 以太网口目前支持的协议包括编程协议、ModbusTCP 协议（主从站）、TCP 和 UDP 自由通信。

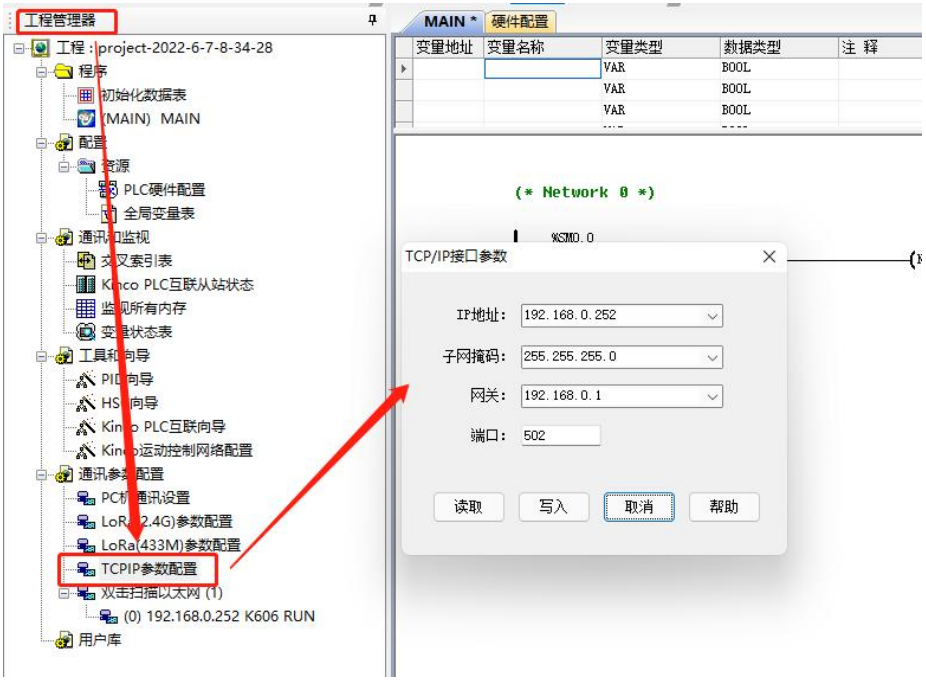
型号	ModbusTCP 从站协议	ModbusTCP 主站协议	自由通信
K204ET	支持	不支持	不支持
KS101M	支持	不支持	不支持
K6	支持	支持	支持

10.3.2 网口参数的设置

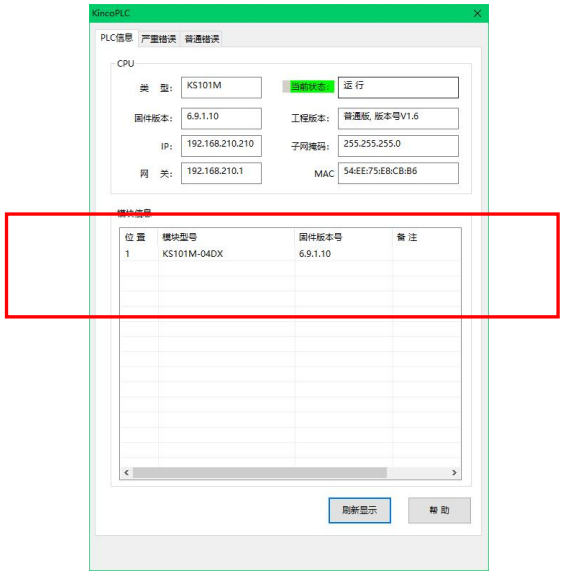
10.3.2.1 本机网口参数

【工具】->【TCP/IP 参数配置】，弹出 TCP/IP 接口参数窗口，或者在左侧【工程管理器】->【通讯参数】中进行点击也会弹出此窗口，在此窗口中可以读写 PLC 的 IP 地址、子网掩码、网关、端口这些参数。





上图所示即为网口 PLC 默认 IP 及网关，端口参数等。当前实时 PLC 的网络参数及 Mac 地址可用户可以通过编程口连接上 PLC 后，在 Kincobuilder 中执行【PLC】->【PLC 信息】菜单命令，在对话框窗口中查看，刷新显示，如下图示意图。需要注意的是 **Kinco PLC 的 Mac 地址暂不支持修改**。



10.3.2.2 以太网通信设置

带网口的 CPU 可以通过网口实现 PC 和 CPU 的通信，首先需要打开通信设置画面如图 1：

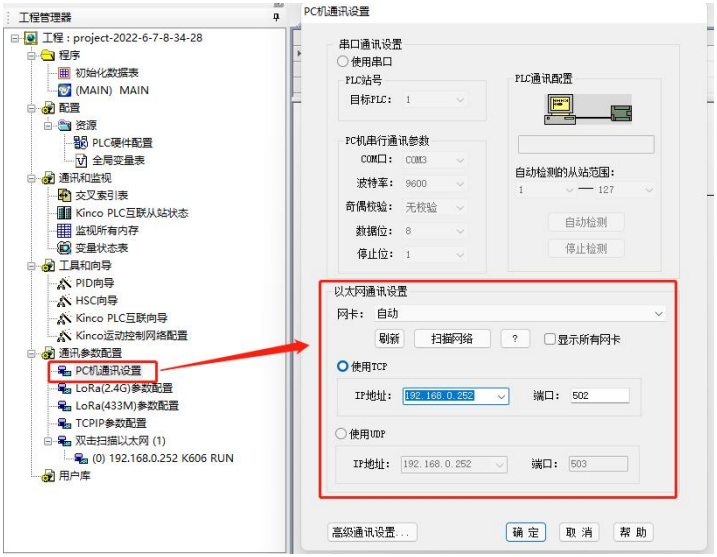
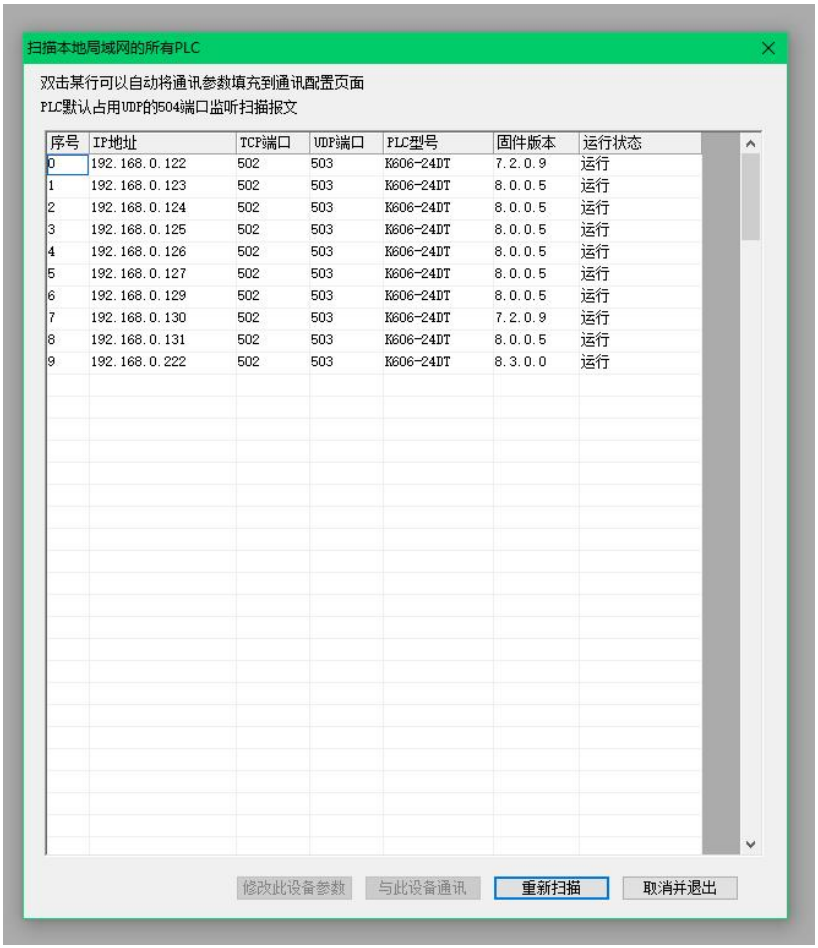


图 1

1、使用 TCP/UDP：在已知 CPU 的 IP 地址和端口号的情况下可以直接写入参数，与对应的 CPU 监控或者下载程序等，传输方式可以选择 TCP 或者 UDP。

注意：在同一网络里面有多个 CPU 的 IP 地址和端口号相同时不可以通过网口进行监控或者下载程序等，以免出现不可预期的错误。

2、扫描网络：在同一个局域网里面可以通过扫描网络的功能扫描到网络中所有的设备，此功能支持跨网段扫描。如图 2 所示：



序号	IP地址	TCP端口	UDP端口	PLC型号	固件版本	运行状态
0	192.168.0.122	502	503	K606-24DT	7.2.0.9	运行
1	192.168.0.123	502	503	K606-24DT	8.0.0.5	运行
2	192.168.0.124	502	503	K606-24DT	8.0.0.5	运行
3	192.168.0.125	502	503	K606-24DT	8.0.0.5	运行
4	192.168.0.126	502	503	K606-24DT	8.0.0.5	运行
5	192.168.0.127	502	503	K606-24DT	8.0.0.5	运行
6	192.168.0.129	502	503	K606-24DT	8.0.0.5	运行
7	192.168.0.130	502	503	K606-24DT	7.2.0.9	运行
8	192.168.0.131	502	503	K606-24DT	8.0.0.5	运行
9	192.168.0.222	502	503	K606-24DT	8.3.0.0	运行

图 2

图 3

3、修改此设备参数：左键选中上图中想要修改的设备序号点击右键选择修改此设备参数弹出如下画面，可以修改此设备的以太网参数。

图 4

4、与设备通信：点击与此设备通信则跳转回图 1，TCP 或者 UDP 的通信参数将会自动变为想要通信设备的 IP 地址和端口号。

图 5

5、在软件中直接批量修改 IP 和端口：用户可以使用此方式批量修改局域网中所有设备的参数。

① 在不勾选 以此IP为起始，所有IP设置为不同值 以此端口为起始，所有端口设置为不同值 时点击确定，网络中所有的设备都会修改为相同的 IP 地址、子网掩码、网关、端口号。如图 6。

② 在 以此IP为起始，所有IP设置为不同值 以此端口为起始，所有端口设置为不同值 时点击确定，网络中的设备以此 IP 为起始 IP、以此端口为起始端口号进行递增。图 6 中修改前网络中所有设备的 IP 地址和和端口号都相同，图 7 为批量修改后的效果。

注意：修改范围：开始序号-结束序号指的是扫描到的网络中的 PLC 序号，可以只修改指定序号 PLC 的参数。

序号	IP地址	TCP端口	UDP端口	PLC型号	固件版本	运行状态
0	192.168.0.253	502	503	K606	8.4.0.0	运行


扫描本地局域网的所有PLC
双击某行可以自动将通讯参数填充到通讯配置页面
PLC默认占用UDP的504端口监听扫描报文
序号 IP地址 TCP端口 UDP端口 PLC型号 固件版本 运行状态
0 192.168.0.252 502 503 K606-24DT 7.2.0.9 运行
1 192.168.0.252 502 503 K606-24DT 8.0.0.5 运行
2 192.168.0.252 502 503 K606-24DT 8.0.0.5 运行
3 192.168.0.252 502 503 K606-24DT 8.0.0.5 运行
4 192.168.0.252 502 503 K606-24DT 8.0.0.5 运行
5 192.168.0.252 502 503 K606-24DT 8.0.0.5 运行
6 192.168.0.252 502 503 K606-24DT 8.3.0.0 运行
7 192.168.0.252 502 503 K606-24DT 8.0.0.5 运行
8 192.168.0.252 502 503 K606-24DT 7.2.0.9 运行
9 192.168.0.252 502 503 K606-24DT 8.0.0.5 运行
修改此设备参数
与此设备通讯
重新扫描
取消并退出

图 6

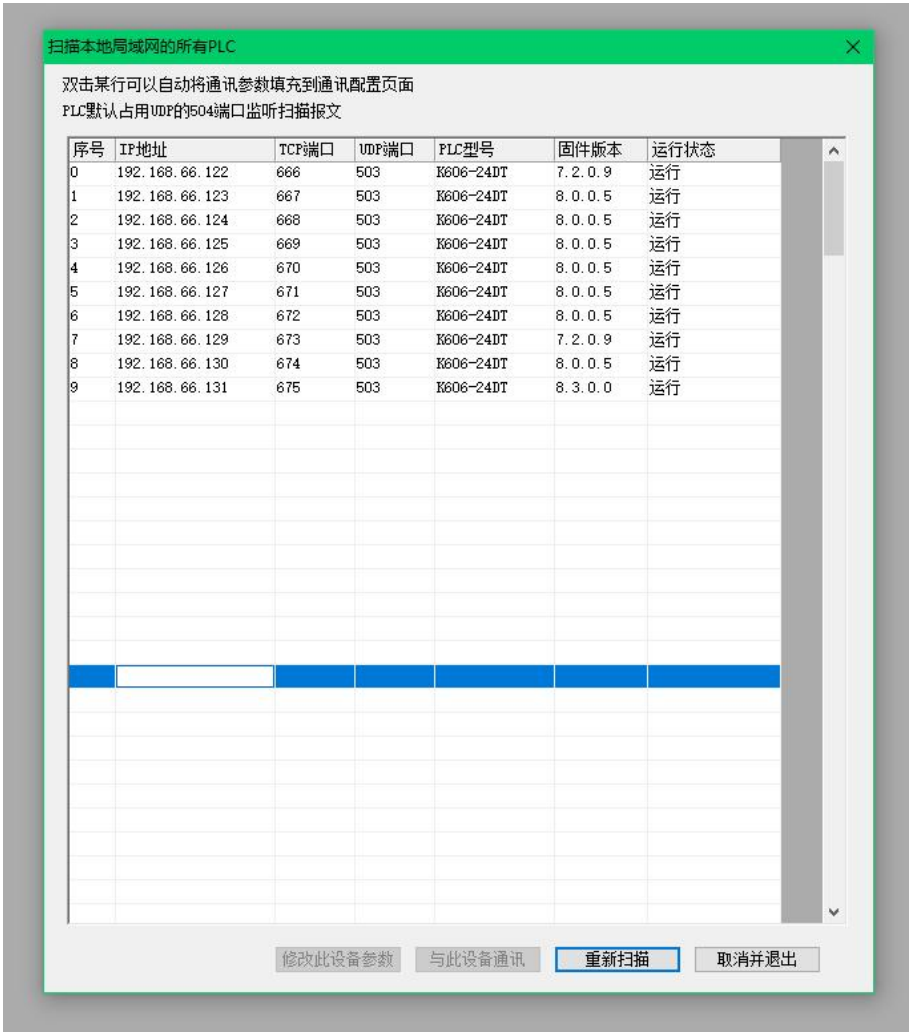


图 7

③ 在 ☑ 以此IP为起始，所有IP设置为不同值 ☐ 以此端口为起始，所有端口设置为不同值 时点击确定，网络中的设备以此 IP 为起始 IP 进行递增，所谓设备端口号变为相同的。

④ 在 ☐ 以此IP为起始，所有IP设置为不同值 ☑ 以此端口为起始，所有端口设置为不同值 时点击确定，网络中的设备以此端口为起始端口号进行递增，所有设备 IP 地址变为相同的。

6、通过 CSV 文件实现批量修改 IP、端口等参数。

①将网络中所有 PLC 参数导出列表到 CSV 文件，如图 1、图 2。

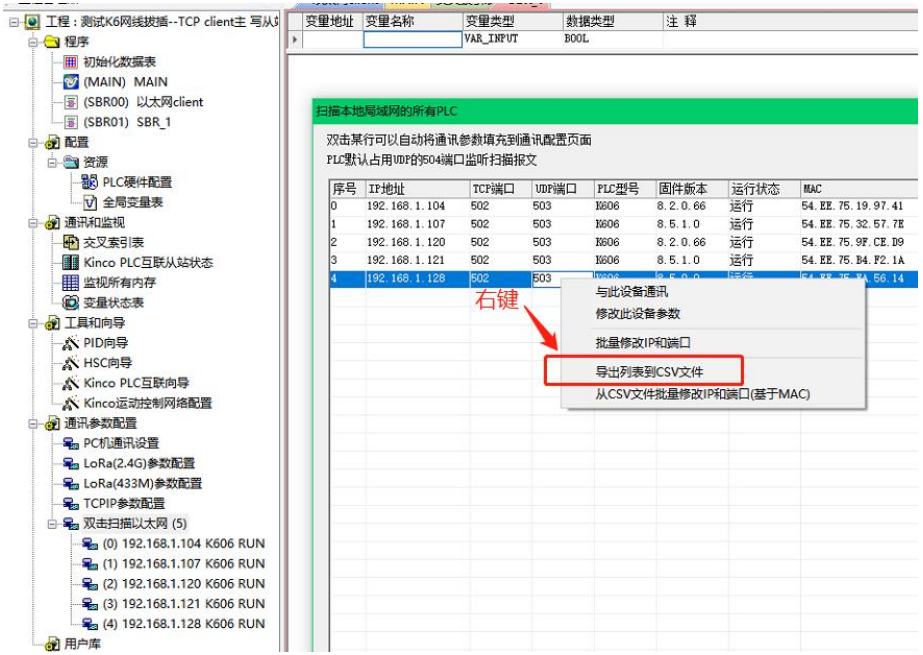
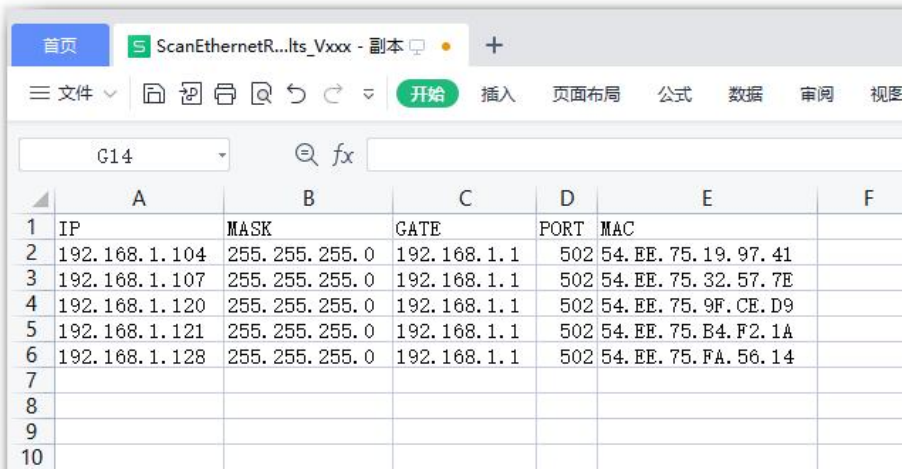


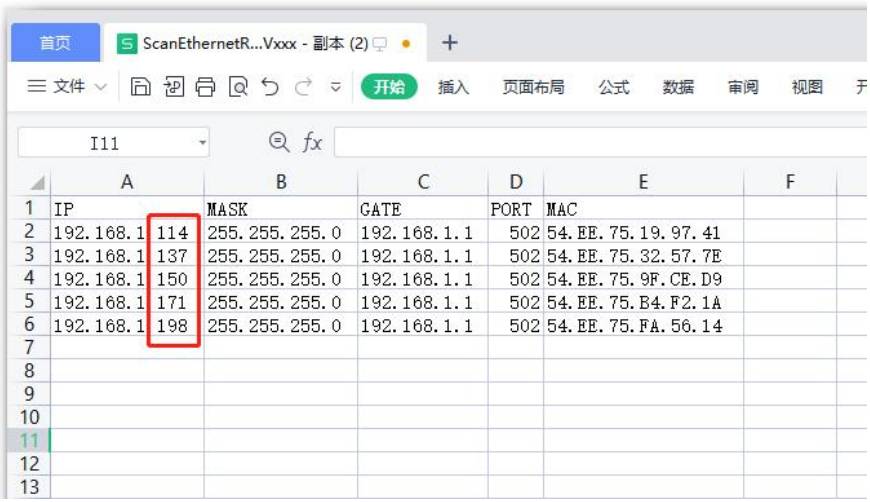
图 1



	A	B	C	D	E	F
1	IP	MASK	GATE	PORT	MAC	
2	192.168.1.104	255.255.255.0	192.168.1.1	502	54.EE.75.19.97.41	
3	192.168.1.107	255.255.255.0	192.168.1.1	502	54.EE.75.32.57.7E	
4	192.168.1.120	255.255.255.0	192.168.1.1	502	54.EE.75.9F.CE.D9	
5	192.168.1.121	255.255.255.0	192.168.1.1	502	54.EE.75.B4.F2.1A	
6	192.168.1.128	255.255.255.0	192.168.1.1	502	54.EE.75.FA.56.14	
7						
8						
9						
10						

图 2

②在导出之后的 csv 文件中修改相关参数。



	A	B	C	D	E	F
1	IP	MASK	GATE	PORT	MAC	
2	192.168.1.114	255.255.255.0	192.168.1.1	502	54.EE.75.19.97.41	
3	192.168.1.137	255.255.255.0	192.168.1.1	502	54.EE.75.32.57.7E	
4	192.168.1.150	255.255.255.0	192.168.1.1	502	54.EE.75.9F.CE.D9	
5	192.168.1.171	255.255.255.0	192.168.1.1	502	54.EE.75.B4.F2.1A	
6	192.168.1.198	255.255.255.0	192.168.1.1	502	54.EE.75.FA.56.14	
7						
8						
9						
10						
11						
12						
13						

图 3

③将修改后的 CSV 文件导入进来。

图 4

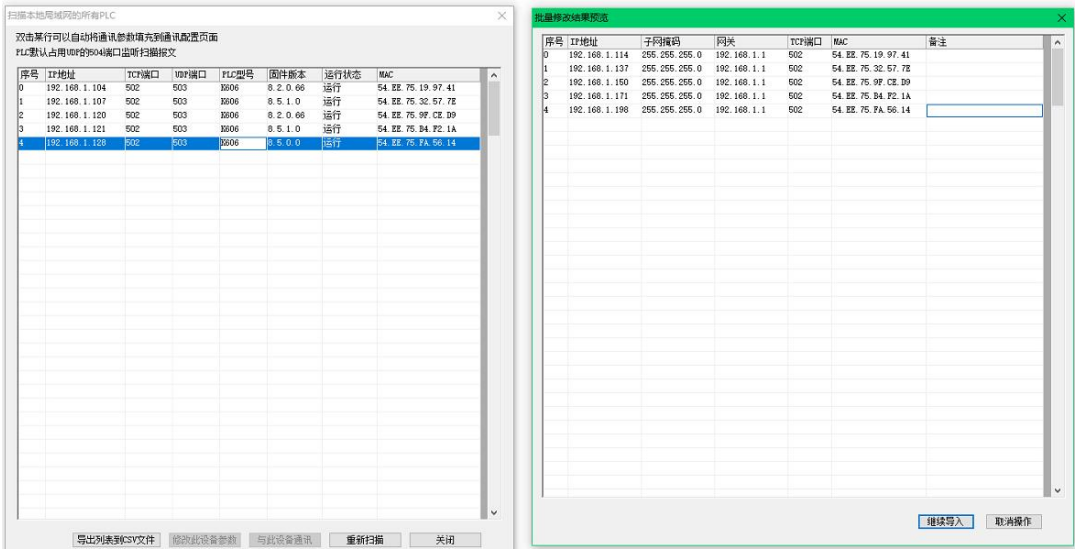


图 5

④导入完成之后会弹出需要重新扫描观察结果的窗口。

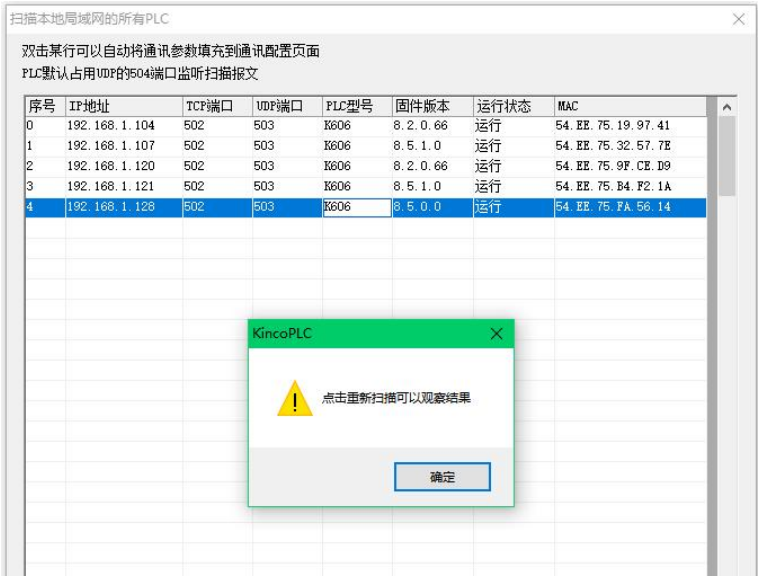


图 6

⑤修改成功之后的参数如下图。

图 7

注意：

a) 扫描网络中序号的排序规则为 IP 地址由小到大进行排序，当出现 IP 地址相同的时候以 MAC 地址由小到大进行排序。

图 6 为 IP 地址相同，以 MAC 地址由小到大进行排序。

图 7 为 IP 地址全不相同，以 IP 地址由小到大进行排序。

当部分 IP 地址相同时优先以 IP 地址对所有设备从小到大排序，对 IP 地址相同的设备再以 MAC 地址由小到大进行排序。

b) 当使用批量修改 IP 和端口时，起始 IP 地址和端口号需要保证是一个合理的值，否则可能导致地址和端口号递增后最终超过允许的最大值，这种情况不会修改成功并且软件会有报错提示。

c) 当搞不清楚网络中 IP 地址和实际 CPU 的对应关系时，可以单独连接某个 PLC 进行确定，或者通过将某个 PLC 置于停止状态通过观察网络中 CPU 的运行状态确认 IP 地址。

d) 扫描网络时 KincoBuilder 默认占用 UDP 的 504 端口号，另外通信连接默认为 503 端口号，所以当使用 UDP 指令时一定要避开这两个端口号以免造成不可预期的错误。

e) KincoBuilder 与 CPU 通过 TCP 通信的端口号，在程序中使用 TCP 指令时一定要避开以免造成不可预期的错误。

10.3.3 以太网指令

10.3.3.1 以太网指令综述

名称	功能描述	最大个数
TCP_MBUSW	ModbusTCP 主站读指令	32
TCP_MBUSR	ModbusTCP 主站写指令	32
TCP_SERVER	开启 TCP 本地服务器	16
TCPS_CLOSE	关闭 TCP 本地服务器	16
TCPS_XMT	作为 TCP 服务器时的发送指令	32
TCPS_RCV	作为 TCP 服务器时的接收指令	32
TCPS_XMTRCV	作为 TCP 服务器时的发送接收指令	32
TCP_CLIENT	开启 TCP 本地客户端	16
TCPC_CLOSE	关闭 TCP 本地客户端	16
TCPC_XMT	作为 TCP 客户端时的发送指令	32
TCPC_RCV	作为 TCP 客户端时的接收指令	32
TCPC_XMTRCV	作为 TCP 客户端时的发送接收指令	32
UDP_PEER	创建 UDP_PEER	16
UDP _XMT	作为 UDP_PEER 时的发送指令	32
UDP _RCV	作为 UDP_PEER 时的接收指令	32
UDP _XMTRCV	作为 UDP_PEER 时的发送接收指令	32
ETH_STATUS	监控当前以太网状态的指令	16
ETH_RPARAS	读取本机 CPU 的以太网参数（IP、子网掩码、网关、端口号）	16
ETH_WPARAS	修改本机 CPU 的以太网参数（IP、子网掩码、网关、端口号）	16
ETH_RESET	重置以太网功能	16

1) 注意事项

用户在使用这些指令时，需要注意如下几点：

- K 系列 CPU 提供 TCP 服务器资源 16 个、TCP 客户端资源 16 个、UDP 资源 16 个，实际使用时不可以超过相应的最大资源使用数量。另外，CPU 作为 ModbusTCP 从站的时候会占用 TCP 服务

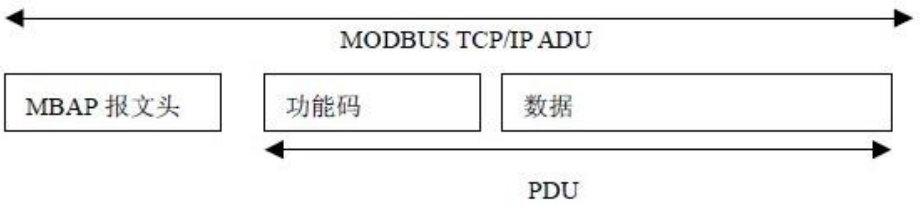
器资源，CPU 作为 ModbusTCP 主站的时候会占用 TCP 客户端资源。

- TCP 指令使用本地端口号时需要避开用于编程软件和 CPU 通信所使用的的端口号，以免影响通信。
- UDP 指令使用本地端口号时需要避开用于编程软件和 CPU 通信所使用的的端口号（503 和 504），以免影响通信。
- 以太网的所有指令都有个数限制，具体限制请看上表，当程序中使用个数超过上限值时 PLC 将会报错，在【PLC 信息】里面的【普通错误】可以看到错误代码。
- TCP_MBUSW,TCP_MBUSR，这两个指令，会与 TCPC_xxx 系类指令竞争 PLC 的客户端资源，尽量不要同时使用， 非要同时使用， 注意 PLC 一共 16 个客户端资源。
- TCP_MBUSW,TCP_MBUSR，这两个指令，启动时会自动配置占用哪个 PLC 客户端，多个指令运行时，会自动根据用户输入的 R_IP，R_PORT，分组连接。
- TCP_MBUSW,TCP_MBUSR，这两个指令，独立于其他所有的以太网指令， 每个指令可以单独使用，实现所有功能。
- xxx_RCV 和 xxx _XMTRCV 指令同时使用时，只有在程序中靠前的那条指令可以接收到报文。

10.3.3.2 ModbusTCP 指令

ModbusTCP 是基于以太网 TCP/IP 的 Modbus 协议，物理层是网口与 ModbusRTU 的物理层不一样，采用 master/slave 方式通信，目前 Kinco-K 系列 PLC 既支持 ModbusTCP 主站也支持 ModbusTCP 从站，默认为 ModbusTCP 从站。

简单的理解一下 Modbus TCP/IP 协议的内容，就是去掉了 modbus 协议本身的 CRC 校验，增加了 MBAP 报文头。TCP/IP 上的 MODBUS 的请求/响应如下图所示：



MBAP 为报文头，长度为 7 字节，组成如下：

事务处理标识	协议标识	长度	单元标识符
2 字节	2 字节	2 字节	1 字节

内容	解释
事务处理标识	可以理解为报文的序列号，一般每次通信之后就要加 1 以区别不同的通信数据报文。
协议标识	00 00 表示 ModbusTCP 协议。
长度	表示接下来的数据长度，单位为字节。
单元标识符	可以理解为设备地址。

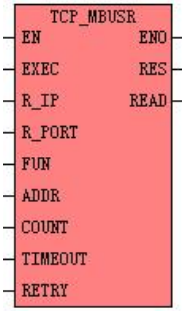
帧结构 PDU 由功能码+数据组成。与串行链路 Modbus 的功能码和数据域是一致的，可以参考串口 Modbus 功能的介绍。

单元标识符：K 系列 PLC 作为 ModbusTCP 主站时对外发送的报文，单元标识符始终为 1，也即忽略从站地址。PLC 作为 ModbusTCP 从站时回复的报文，单元标识符与主站发送的相同。

注意：网口 Modbus TCP 主站可访问的 PLC 内存区和寄存器的编号和串口 Modbus RTU 是一样的，在此不再累述，详情请参阅 10.2.2.2 Modbus RTU 主站可访问 K 系列 PLC 的内存区和 10.2.2.3 Modbus 寄存器编号。

10.3.3.2.1 TCP_MBUSR 指令

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	TCP_MBUSR			☑ K6
IL	TCP_MBUSR	TCP_MBUSR *EXEC, R_IP, R_PORT, FUN, ADDR, COUNT, TIMEOUT, RETRY, RES, READ*	U	

参数	输入/输出	数据类型	允许使用的内存区
EXEC	输入	BOOL	I、Q、V、M、L、SM、RS、SR
R_IP	输入	DWORD	M、V、L、SM、常量
R_PORT	输入	WORD	M、V、L、SM、常量
FUN	输入	INT	常量（Modbus 功能码）
ADDR	输入	WORD	M、V、L、SM、常量
COUNT	输入	INT	M、V、L、SM、常量

TIMEOUT	输入	INT	M、V、L、SM、常量
RETRY	输入	INT	M、V、L、SM、常量
RES	输出	BYTE	Q、M、V、L、SM
READ	输出	BOOL、WORD、INT	Q、M、V、L、SM

注意：参数 *R_IP*, *R_PORT*, *ADDR*, *COUNT*, *TIMEOUT*, *RETRY* 必须同时为常量类型或同时为内存类型。

注意， READ 参数为一个可变长度的块内存参数， 整个块内存都不可以落在非法内存区域， 否则结果不可预期。

参数	描　述
EXEC	若检测到 *EXEC* 的上升沿跳变，则该指令被触发执行。
R_IP	远端的服务器的 ip 地址,可以直接输入 IP 比如 192.168.210.100，也可以输入十六进制 DW#16#C0A8D264，也即 192==C0　168==A8　210==D2　100==64。
R_PORT	远端服务器的 port 端口，不支持 0。
FUN	Modbus 功能码。
ADDR	要访问的 modbusTCP 从站的首个寄存器地址。
COUNT	要访问多少个连续的寄存器， FUN 参数==1 或 2 时，COUNT 范围为 1-1600，其他输入触发错误；FUN 参数==3 或 4 时，COUNT 范围为 1-125，其他输入触发错误。
TIMEOUT	本次执行的超时时间，单位 ms，不能是 0ms， 不能大于 32767ms， 不能为负数。 例如输入 5000， 表示 5 秒内， 进行如下动作序列： a. 一直等待连接服务器成功，成功后立刻发送要发送的数据， 一直没有连接成功，返回失败并结束。 b. 发送成功后， 等待第一包接收数据， 一直没有等到， 返回失败并结束。 c. 发送成功后， 接收到了一包数据， 本指令成功结束。
RETRY	重试次数，范围 0-255,输入超过此范围，自动截取本参数低字节的值，发送失败后，重试的次数，比如重试次数为 1，假设每次发送都失败， 则启动发送后，一共发送 2 次。
READ	FUN 参数为 1，2 时， 必须为 BOOL 变量，为接收到的 bit 串的第一个 bit，接收到的其它 bit，依次摆放。 FUN 参数为 3，4 时， 必须为 WORD 或 INT 变量，为接收到的数据的第一个字，接收到的其它字，依次摆放。

RES	执行开始时， 此参数输出 0，执行结束时， bit7 == TRUE。 bit0-bit6 合起来的 byte 数值，表示错误码 0. 指令执行成功。 1. 无法自动分配客户端资源 2. 长度错误 3. 操作超时没有收到回执 4. 一直无法连接到服务器 5. TIMEOUT 参数错误 6. IP 或 PORT 参数错误 7. 功能码错误 8. 收到的回执存在错误 9. 有太多正在尝试连接中的 TCP 活动(当 ModbusTCP 从站快速访问不存在的从站或断线的从站时容易出现，放慢速度即可自动消失)

K 系列 PLC 作为 ModbusTCP 主站时，TCP_MBUSR 指令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）。

参数 R_IP、R_PORT 代表目标从站的 IP 地址和端口号，端口号范围是 1~65535。*FUN* 定义了功能码。*ADDR* 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个数，允许的最大值是读 AI 是 125、读 AO 是 125、读 DI 是 1600、读 DO 是 1600。

EXEC 输入端的上升沿跳变用于启动通信。TCP_MBUSR 指令执行时，若检测到 *EXEC* 的上升沿跳变，TCP_MBUSR 就会进行一次通信：按照用户输入的参数来组织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，校验证明报文是否正确，正确则将所需的数据写入数据缓冲区中，否则接收到的报文会被丢弃。

- LD

  如果 *EN* 为 1，则该指令被执行，否则不执行。

- IL

如果 CR 值为 1，则该指令被执行，否则不执行。

该指令的执行不影响 CR 值。

10.3.3.2.2 TCP_MBUSW 指令

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	TCP_MBUSW			☑ K6
IL	TCP_MBUSW	TCP_MBUSW *EXEC, R_IP, R_PORT, FUN, ADDR, COUNT, WRITE, TIMEOUT, RETRY, RES*	U	

参数	输入/输出	数据类型	允许使用的内存区
EXEC	输入	BOOL	I、Q、V、M、L、SM、RS、SR
R_IP	输入	DWORD	M、V、L、SM、常量
R_PORT	输入	WORD	M、V、L、SM、常量
FUN	输入	INT	常量（Modbus 功能码）
ADDR	输入	WORD	M、V、L、SM、常量
COUNT	输入	INT	M、V、L、SM、常量
WRITE	输入	BOOL、WORD、INT	Q、M、V、L、SM
TIMEOUT	输入	INT	M、V、L、SM、常量

RETRY	输入	INT	M、V、L、SM、常量
RES	输出	BYTE	Q、M、V、L、SM

注意：参数 *R_IP*, *R_PORT*, *ADDR*, *COUNT*, *TIMEOUT*, *RETRY* 必须同时为常量类型或同时为内存类型。

注意， WRITE 参数为一个可变长度的块内存参数， 整个块内存都不可以落在非法内存区域， 否则结果不可预期。

参数	描　述
EXEC	若检测到 *EXEC* 的上升沿跳变，则该指令被触发执行。
R_IP	远端的服务器的 ip 地址,可以直接输入 IP 比如 192.168.210.100，也可以输入十六进制 DW#16#C0A8D264，也即 192==C0　168==A8　210==D2　100==64。
R_PORT	远端服务器的 port 端口，不支持 0。
FUN	Modbus 功能码。
ADDR	要访问的 modbusTCP 从站的首个寄存器地址。
COUNT	要访问多少个连续的寄存器， FUN 参数==6 时，COUNT 只能等于 1，FUN 参数==16 时，COUNT 范围为 1-100，其他输入触发错误；FUN 参数==5 时，COUNT 只能等于 1，FUN 参数==15 时，COUNT 范围为 1-800，其他输入触发错误。
WRITE	FUN 参数为 6，16 时，必须为 WORD 或 INT 变量，为待发送数据的第一个字，待发送数据的其他字，依次摆放；FUN 参数为 5，15 时，必须为 BOOL 变量，为待发送数据的第一个 bit， 待发送数据的其他 bit，依次摆放。
TIMEOUT	本次执行的超时时间，单位 ms，不能是 0ms，不能大于 32767ms，不能为负数。 例如输入 5000， 表示 5 秒内， 进行如下动作序列： a. 一直等待连接服务器成功，成功后立刻发送要发送的数据， 一直没有连接成功，返回失败并结束。 b. 发送成功后， 等待第一包接收数据， 一直没有等到， 返回失败并结束。 c. 发送成功后， 接收到了一包数据， 本指令成功结束。
RETRY	重试次数，范围 0-255,输入超过此范围，自动截取本参数低字节的值，发送失败后，重试的次数，比如重试次数为 1，假设每次发送都失败， 则启动发送后，一共发送 2 次。

RES	执行开始时， 此参数输出 0，执行结束时， bit7 == TRUE。 bit0-bit6 合起来的 byte 数值，表示错误码 0. 指令执行成功。 1. 无法自动分配客户端资源 2. 长度错误 3. 操作超时没有收到回执 4. 一直无法连接到服务器 5. TIMEOUT 参数错误 6. IP 或 PORT 参数错误 7. 功能码错误 8. 收到的回执存在错误 9. 有太多正在尝试连接中的TCP活动(当ModbusTCP从站快速访问不存在的从站或断线的从站时容易出现，放慢速度即可自动消失)

K 系列 PLC 作为 ModbusTCP 主站时，TCP_MBUSW 指令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）。

参数 R_IP、R_PORT 代表目标从站的 IP 地址和端口号，端口号范围是 1~65535。*FUN* 定义了功能码。*ADDR* 定义了要写入的寄存器的起始地址。*COUNT* 定义了写入的寄存器个数，允许的最大值是写 AO 是 100、写 DO 是 800。

参数 *WRITE* 定义了数据缓冲区的起始地址，要写入从站的数据就存放在该区域内。*WRITE* 必须与功能码匹配。若功能码是 5、15，则输入 BOOL 型的地址变量；若功能码是 6、16，则输入 INT 或者 WORD 型的地址变量。

EXEC 输入端的上升沿跳变用于启动通信。TCP_MBUSW 指令执行时，若检测到 *EXEC* 的上升沿跳变，MBUSW 就会进行一次通信：按照用户输入的参数来组织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，判读从站是否正确执行了刚才的写命令。

- LD

  如果 *EN* 为 1，则该指令被执行，否则不执行。

- IL

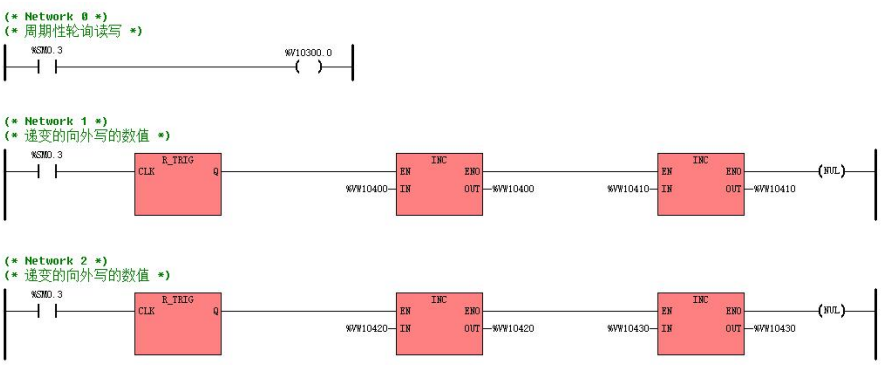
  如果 CR 值为 1，则该指令被执行，否则不执行。

  该指令的执行不影响 CR 值。

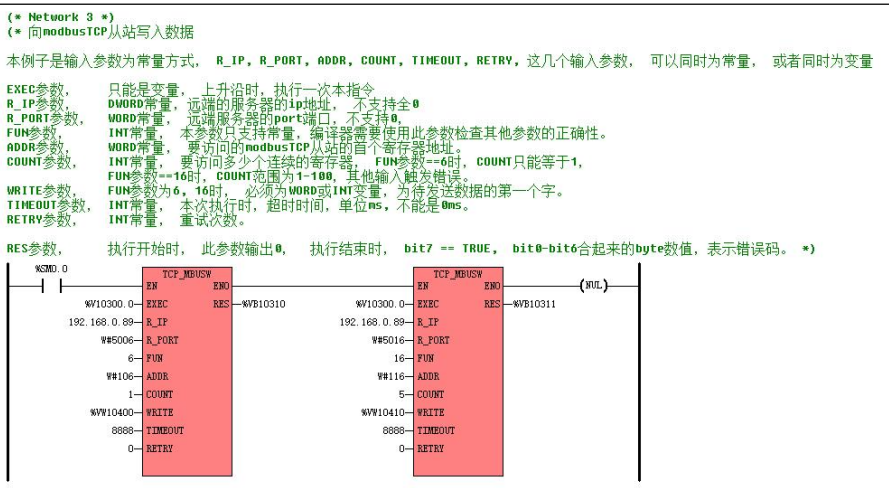
10.3.3.2.3 示例

本例通过网络调试助手监控 ModbusTCP 报文。

LD

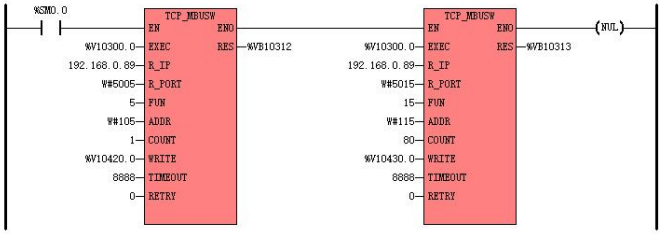


LD

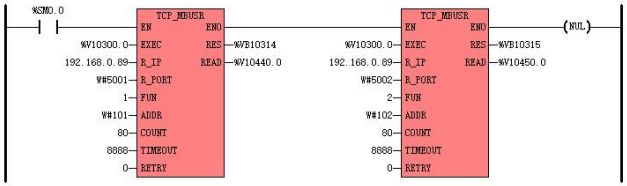


LD

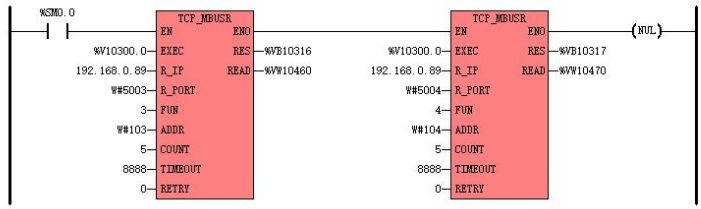
```
(* Network 4 *)
(* 向modbusTCP从站写入数据

COUNT参数,    INT常量, 要访问多少个连续的寄存器, FUN参数==5时, COUNT只能等于1,
              FUN参数==15时, COUNT范围为1-800, 其他输入触发错误
WRITE参数,    FUN参数为5, 15时, 必须为BOOL变量, 为待发送数据的第一个bit, 待发送数据的其他bit, 依次摆放

其他参数见上面的FUN==6, FUN==16的说明 *)
```




```
(* Network 5 *)
(* 从modbusTCP从站读取数据

本例子是输入参数为常量方式, R_IP, R_PORT, ADDR, COUNT, TIMEOUT, RETRY, 这几个输入参数, 可以同时为常量, 或者同时为变量

EXEC参数,     只能是变量, 上升沿时, 执行一次本指令
R_IP参数,     DWORD常量, 远端的服务器的ip地址, 不支持全0
R_PORT参数,   WORD常量, 远端服务器的port端口, 不支持0,
FUN参数,      INT常量, 本参数只支持常量, 编译器需要使用此参数检查其他参数的正确性。
ADDR参数,     WORD常量, 要访问的modbusTCP从站的首个寄存器地址。
COUNT参数,    INT常量, 要访问多少个连续的寄存器, FUN参数==1或2时, COUNT范围为1-1600, 其他输入触发错误
TIMEOUT参数,  INT常量, 本次执行时, 超时时间, 单位ms。
RETRY参数,    INT常量, 重试次数

RES参数,      执行开始时, 此参数输出0, 执行结束时, bit7 == TRUE, bit0-bit6合起来的byte数值, 表示错误码

READ参数,     FUN参数为1, 2时, 必须为BOOL变量, 为接收到的bit串的第一个bit, 接收到的其它bit, 依次摆放 *)
```




```
(* Network 6 *)
(* 从modbusTCP从站读取数据

COUNT参数,    INT常量, 要访问多少个连续的寄存器, FUN参数==3或4时, COUNT范围为1-125, 其他输入触发错误
READ参数,     FUN参数为3, 4时, 必须为WORD或INT变量, 为接收到的数据的第一个字, 接收到的其它字, 依次摆放

其他参数见上面的FUN==1, FUN==2的说明 *)
```




网络调试助手
网络设置
（1）协议类型
TCP Server
（2）本地主机地址
192.168.0.89
（3）本地主机端口
5005
关闭
接收设置
ASCII HEX
按日志模式显示
接收区自动换行
接收数据不显示
接收保存到文件...
自动滚屏 清除接收
自动应答 界面主题
发送设置
ASCII HEX
自动解析转义符
AT指令自动回车
自动发送附加位
数据日志
NetAssist V5.0.1
[2022-05-10 09:48:30.018]# RECV HEX FROM 192.168.0.252 :50155>
06 58 00 00 00 06 01 05 00 69 00 00
[2022-05-10 09:48:31.019]# RECV HEX FROM 192.168.0.252 :50155>
06 5C 00 00 00 06 01 05 00 69 FF 00
[2022-05-10 09:48:32.020]# RECV HEX FROM 192.168.0.252 :50155>
06 60 00 00 00 06 01 05 00 69 00 00
[2022-05-10 09:48:33.018]# RECV HEX FROM 192.168.0.252 :50155>
06 64 00 00 00 06 01 05 00 69 FF 00
[2022-05-10 09:48:34.018]# RECV HEX FROM 192.168.0.252 :50155>
06 68 00 00 00 06 01 05 00 69 00 00
[2022-05-10 09:48:35.018]# RECV HEX FROM 192.168.0.252 :50155>
06 6C 00 00 00 06 01 05 00 69 FF 00
[2022-05-10 09:48:36.019]# RECV HEX FROM 192.168.0.252 :50155>
06 70 00 00 00 06 01 05 00 69 00 00
[2022-05-10 09:48:37.018]# RECV HEX FROM 192.168.0.252 :50155>
06 74 00 00 00 06 01 05 00 69 FF 00
[2022-05-10 09:48:38.018]# RECV HEX FROM 192.168.0.252 :50155>
06 78 00 00 00 06 01 05 00 69 00 00
数据发送
客户端:
192.168.0.252 :50155
断开
清除
清除
PLC的IP和端口号
Modbus从站的IP和端口号


网络调试助手
网络设置
（1）协议类型
TCP Server
（2）本地主机地址
192.168.0.89
（3）本地主机端口
5016
关闭
接收设置
ASCII HEX
按日志模式显示
接收区自动换行
接收数据不显示
接收保存到文件...
自动滚屏 清除接收
自动应答 界面主题
发送设置
ASCII HEX
自动解析转义符
AT指令自动回车
数据日志
NetAssist V5.0.1
[2022-05-10 09:56:39.008]# RECV HEX FROM 192.168.0.252 :53747>
0D FB 00 00 00 11 01 10 00 74 00 05 0A 05 2B 00 00 00 00 00 00 00 00
[2022-05-10 09:56:40.008]# RECV HEX FROM 192.168.0.252 :53747>
0D FF 00 00 00 11 01 10 00 74 00 05 0A 05 2C 00 00 00 00 00 00 00 00
[2022-05-10 09:56:41.007]# RECV HEX FROM 192.168.0.252 :53747>
0E 03 00 00 00 11 01 10 00 74 00 05 0A 05 2D 00 00 00 00 00 00 00 00
[2022-05-10 09:56:42.007]# RECV HEX FROM 192.168.0.252 :53747>
0E 07 00 00 00 11 01 10 00 74 00 05 0A 05 2E 00 00 00 00 00 00 00 00
[2022-05-10 09:56:43.007]# RECV HEX FROM 192.168.0.252 :53747>
0E 0B 00 00 00 11 01 10 00 74 00 05 0A 05 2F 00 00 00 00 00 00 00 00
[2022-05-10 09:56:44.008]# RECV HEX FROM 192.168.0.252 :53747>
0E 0F 00 00 00 11 01 10 00 74 00 05 0A 05 30 00 00 00 00 00 00 00 00
[2022-05-10 09:56:45.007]# RECV HEX FROM 192.168.0.252 :53747>
0E 13 00 00 00 11 01 10 00 74 00 05 0A 05 31 00 00 00 00 00 00 00 00
[2022-05-10 09:56:46.007]# RECV HEX FROM 192.168.0.252 :53747>
0E 17 00 00 00 11 01 10 00 74 00 05 0A 05 32 00 00 00 00 00 00 00 00
[2022-05-10 09:56:47.007]# RECV HEX FROM 192.168.0.252 :53747>
0E 1B 00 00 00 11 01 10 00 74 00 05 0A 05 33 00 00 00 00 00 00 00 00

IL	(* Network 0 *) (*周期性轮询读写*) LD %SM0.3 ST %V10300.0 (* Network 1 *) (*递变的向外写的数值*) LD %SM0.3 R_TRIG INC %VW10400 INC %VW10410 (* Network 2 *) (*递变的向外写的数值*) LD %SM0.3 R_TRIG INC %VW10420 INC %VW10430 (* Network 3 *) LD %SM0.0 TCP_MBUSW %V10300.0, 192.168.0.89, W#5006, 6, W#106, 1, %VW10400, 8888, 0, %VB10310 TCP_MBUSW %V10300.0, 192.168.0.89, W#5016, 16, W#116, 5, %VW10410, 8888, 0, %VB10311 (* Network 4 *) LD %SM0.0 TCP_MBUSW %V10300.0, 192.168.0.89, W#5005, 5, W#105, 1, %V10420.0, 8888, 0, %VB10312 TCP_MBUSW %V10300.0, 192.168.0.89, W#5015, 15, W#115, 80, %V10430.0, 8888, 0, %VB10313

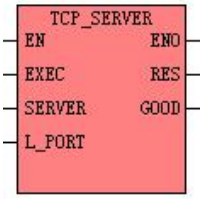
IL	(* Network 5 *) LD %SM0.0 TCP_MBUSR %V10300.0, 192.168.0.89, W#5001, 1, W#101, 80, 8888, 0, %VB10314, %V10440.0 TCP_MBUSR %V10300.0, 192.168.0.89, W#5002, 2, W#102, 80, 8888, 0, %VB10315, %V10450.0 (* Network 6 *) LD %SM0.0 TCP_MBUSR %V10300.0, 192.168.0.89, W#5003, 3, W#103, 5, 8888, 0, %VB10316, %VW10460 TCP_MBUSR %V10300.0, 192.168.0.89, W#5004, 4, W#104, 5, 8888, 0, %VB10317, %VW10470

10.3.3.3 TCP 服务器指令

10.3.3.3.1 TCP_SERVER 指令

当需要使用本机作为服务器时，首先使用此指令开启服务器连接。

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	TCP_SERVER			☑ K6
IL	TCP_SERVER	TCP_SERVER *EXEC, SERVER, L_PORT, RES, GOOD*	U	

参数	输入/输出	数据类型	允许使用的内存区
EXEC	输入	BOOL	I、Q、V、M、L、SM、RS、SR
SERVER	输入	INT	M、V、L、常量
L_PORT	输入	WORD	M、V、L、常量
RES	输出	BYTE	Q、M、V、L、SM
GOOD	输出	BOOL	Q、M、V、L、SM

注意：参数 *SERVER, L_PORT* 必须同时为常量类型或同时为内存类型。

➢ 参数说明

参数	描　述
EXEC	若检测到 *EXEC* 的上升沿跳变，则该指令被触发执行,开启服务器连接。
SERVER	服务器资源编号，范围 0--15，也即最大支持 16 个 TCP 服务器资源。
L_PORT	服务器的本地 port 端口，不支持 0。

RES	执行开始时，此参数输出 0，执行结束时，bit7 == TRUE, bit0-bit6 合起来的 byte 数值，表示错误码 0 连接成功 1 SERVER 参数错误 2 R_IP 或 R_PORT 参数错误 3 保留未使用。 4 已经连接了客户端，但客户端长时间无操作，因此结束
GOOD	每次执行 EXEC 时， GOOD 参数先会被赋值为 FALSE。 TRUE 表示有远端的客户端连接到本服务器，FALSE 表示没有。 此参数在指令执行后更新一次，不会动态更新，可通过 ETH_STATUS 指令来查看动态连接的变化。

注意：

1、每个 TCP_SERVER 只允许连接一个远端的客户端，不支持多连接。

2、服务器启动后会一直处于 listen 状态，不会超时，直到远端客户端连接了本服务器。

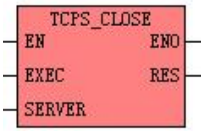
3、远端客户端连接了本服务器，然后远端客户端进行了断开操作，本服务器会再次自动进入 listen 状态，即使你没有再次执行本指令的 EXEC。此时远端客户端重新连接本服务器后，GOOD 不会更新，因为本指令已经在第一次连接成功时结束。

4、因为 PLC 本机只有 1 个网卡，所以不需要输入本地 ip， 默认取 PLC 的本地 ip。

10.3.3.3.2 TCPS_CLOSE 指令

此指令用于关闭本机作为服务器的连接。

- 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	TCPS_CLOSE			☑ K6
IL	TCPS_CLOSE	TCPS_CLOSE *EXEC, SERVER, RES*	U	

参数	输入/输出	数据类型	允许使用的内存区
EXEC	输入	BOOL	I、Q、V、M、L、SM、RS、SR
SERVER	输入	INT	M、V、L、常量
RES	输出	BYTE	Q、M、V、L、SM

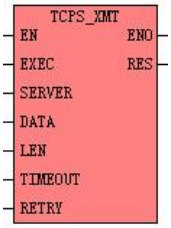
- 参数说明

参数	描　述
EXEC	若检测到 *EXEC* 的上升沿跳变，则该指令被触发执行,关闭 TCP 服务器连接。
SERVER	服务器资源编号，范围 0--15，也即最大支持 16 个 TCP 服务器资源。
RES	指令执行时，此参数输出 0。当执行结束时，bit7 变为 TRUE,bit0-bit6 的组合值表示错误码 0　已经执行了关闭服务器操作，不保证成功。 1　SERVER 参数错误.

10.3.3.3.3 TCPS_XMT 指令

此指令为本机作为服务器时的发送指令。

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	TCPS_XMT			☑ K6
IL	TCPS_XMT	TCPS_XMT *EXEC, SERVER, DATA, LEN, TIMEOUT, RETRY, RES*	U	

参数	输入/输出	数据类型	允许使用的内存区
EXEC	输入	BOOL	I、Q、V、M、L、SM、RS、SR
SERVER	输入	INT	M、V、L、常量
DATA	输入	BYTE	M、V、L
LEN	输入	INT	M、V、L、常量
TIMEOUT	输入	INT	M、V、L、常量
RETRY	输入	INT	M、V、L、常量
RES	输出	BYTE	Q、M、V、L、SM

注意：参数 *SERVER, LEN, TIMEOUT, TRTRY* 必须同时为常量类型或同时为内存类型。

➢ 参数说明

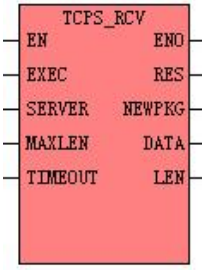
参数	描　述
EXEC	若检测到 *EXEC* 的上升沿跳变，则该指令被触发执行。
SERVER	已经开启的服务器资源编号，范围 0--15。
DATA	待发送数据字节的首地址。
LEN	发送数据的字节数，不能为 0，不能为负数，不能大于 1460 字节。

TIMEOUT	本次执行时的超时时间，单位 ms，不能大于 32767ms，不能为负数，0 表示不等待，有连接就发送，没连接直接失败。非 0 时，比如 5000，表示 5 秒内一直等待连接客户端成功，成功后立刻发送， 一直没有连接就返回超时失败。
RETRY	重试次数，范围 0-255,输入超过此范围，自动截取本参数低字节的值， 发送失败后重试的次数，比如重试次数为 1，假设每次发送都失败，则启动发送后，一共发送 2 次。
RES	执行开始时，此参数输出 0，执行结束时，bit7 变为 TRUE,bit0-bit6 的组合值表示错误码 0 发送完成，仅仅表示提交给了 TCP 协议栈，不代表远端已经收到，不代表远端必然收到 1 SERVER 参数错误 2 发送长度错误 3 操作超时还没有发送完成 4 还没有连接到客户端 5 TIMEOUT 参数, 输入错误 错误码 65--79，表示协议栈底层发送失败。

10.3.3.3.4 TCPS_RCV 指令

此指令为本机作为服务器时的接收指令。

- 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	TCPS_RCV			☑ K6

IL	TCPS_RCV	TCPS_RCV *EXEC, SERVER, MAXLEN, TIMEOUT, RES, NEWPKG, DATA, LEN*	U	

参数	输入/输出	数据类型	允许使用的内存区
EXEC	输入	BOOL	I、Q、V、M、L、SM、RS、SR
SERVER	输入	INT	M、V、L、常量
MAXLEN	输入	INT	M、V、L、常量
TIMEOUT	输入	INT	M、V、L、常量
RES	输出	BYTE	Q、M、V、L、SM
NEWPKG	输出	BOOL	Q、M、V、L、SM
DATA	输出	BYTE	Q、M、V、L、SM
LEN	输出	WORD	Q、M、V、L、SM

注意：参数 *SERVER, MAXLEN, TIMEOUT*,必须同时为常量类型或同时为内存类型。

- 参数说明

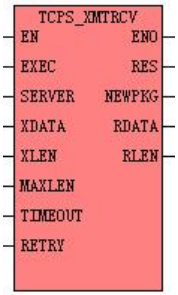
参数	描　述
EXEC	若检测到 *EXEC* 的上升沿跳变，则该指令被触发执行。
SERVER	已经开启的服务器资源编号，范围 0--15。
MAXLEN	1 条报文最大可以接收的数据字节数，不能输入 0，不能为负数，当 MAXLEN 大于 1460 时，会被忽略，最大接收 1460 个字节。
TIMEOUT	本次执行时的超时时间，单位 ms，不能大于 32767ms，不能为负数，0 表示永久等待，一直处于等待客户端报文的状态。非 0 时比如 5000，表示 5 秒内一直等待客户端发来的报文，超时后就不再接收报文。
RES	执行开始时，此参数输出 0，执行结束时，bit7 == TRUE,bit0-bit6 合起来的 byte 数值，表示错误码 0　未使用 1　SERVER 参数错误 2　MAXLEN 长度错误 3　操作超时，　结束接收 5　TIMEOUT 参数，输入错误

NEWPKG	服务器接收到了一个新报文，NEWPKG 就产生一个上升沿， 只维持 1 个扫描周期。
DATA	接收缓冲区的起始字节，注意这个缓冲区是和 MAXLEN 匹配使用的，比如，MAXLEN 是 100，DATA=VB0,此时 VB0--VB99 都是缓冲区，只能读取，不能做其他用处。缓冲区用户可自行清空，否则下次接收的报文长度小于上次时，缓冲区会残留上次报文数据。
LEN	接收报文的长度，当实际报文长度小于等于 MAXLEN 时，LEN 为实际报文长度。当实际报文长度超过 MAXLEN 时，LEN=MAXLEN。

10.3.3.3.5 TCPS_XMTRCV 指令

此指令为本机作为服务器时的收发指令。

- 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	TCPS_XMTRCV			☑ K6
IL	TCPS_XMTRCV	TCPS_RCV *EXEC，SERVER，XDATA，XLEN, MAXLEN, TIMEOUT, RETRY, RES, NEWPKG, RDATA, RLEN*	U	

参数	输入/输出	数据类型	允许使用的内存区
EXEC	输入	BOOL	I、Q、V、M、L、SM、RS、SR
SERVER	输入	INT	M、V、L、常量
XDATA	输入	BYTE	M、V、L
XLEN	输入	INT	M、V、L、常量
MAXLEN	输入	INT	M、V、L、常量
TIMEOUT	输入	INT	M、V、L、常量

RETRY	输入	INT	M、V、L、常量
RES	输出	BYTE	Q、M、V、L、SM
NEWPKG	输出	BOOL	Q、M、V、L、SM
RDATA	输出	BYTE	Q、M、V、L、SM
RLEN	输出	WORD	Q、M、V、L、SM

注意：参数 *SERVER, XLEN, MAXLEN, TIMEOUT, RETRY* 必须同时为常量类型或同时为内存类型。

➢ 参数说明

参数	描　述
EXEC	若检测到 *EXEC* 的上升沿跳变，则该指令被触发执行。
SERVER	已经开启的服务器资源编号，范围 0--15。
XDATA	待发送数据字节的首地址。
XLEN	发送数据的字节数，不能为 0，不能为负数，不能大于 1460 字节。
MAXLEN	1 条报文最大可以接收的数据字节数，不能输入 0，不能为负数，当 MAXLEN 大于 1460 时，会被忽略，最大接收 1460 个字节。
TIMEOUT	本次执行时的超时时间，单位 ms，不能是 0ms，不能大于 32767ms，不能为负数。 例如输入 5000， 表示 5 秒内， 进行如下动作序列: a. 一直等待连接客户端成功，成功后立刻发送要发送的数据，一直没有连接成功，返回失败并结束。 b. 发送成功后， 等待第一包接收数据，一直没有等到，返回失败并结束。 c. 发送成功后， 接收到了一包数据， 本指令成功结束。
RETRY	重试次数，范围 0-255,输入超过此范围，自动截取本参数低字节的值。 发送失败或者接收失败后的重试次数，比如重试次数为 1，假设每次发送都失败，则启动 EXEC 后，一共发送 2 次。
RES	执行开始时，此参数输出 0，执行结束时，bit7 == TRUE, bit0-bit6 合起来的 byte 数值，表示错误码 0　指令成功结束，发出报文，并接收到了一个报文 1　SERVER 参数错误 2　长度错误, MAXLEN 为 0,或者 XLEN 为 0，或者 XLEN 大于 1460 字节,当 MAXLEN 大于 1460 时， 会被忽略， 只接收最大 1460 长度报文

	3　操作超时，没有收到报文 4　发送失败，一直没有连接到客户端 5　TIMEOUT 参数，输入错误
NEWPKG	本指令 EXEC 启动时会自动变成 FALSE，成功接收到了一个报文变为 TRUE，直到下一次启动本指令。
RDATA	接收缓冲区的起始字节，注意这个缓冲区是和 MAXLEN 匹配使用的，比如，MAXLEN 是 100，DATA=VB0,此时 VB0--VB99 都是缓冲区，只能读取，不能做其他用处。缓冲区用户可自行清空，否则下次接收的报文长度小于上次时，缓冲区会残留上次报文数据。
RLEN	接收报文的长度，当实际报文长度小于等于 MAXLEN 时，LEN 为实际报文长度。当实际报文长度超过 MAXLEN 时，LEN=MAXLEN。

注意： 启动一次 EXEC 上升沿只能接收 1 个报文。 取 NEWPKG 的上升沿，读取报文。

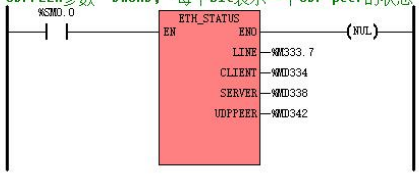
TCPS_RCV 和 TCPS_XMTRCV 指令同时使用时，只有在程序中靠前的那条指令可以接收到报文。

10.3.3.3.6 示例

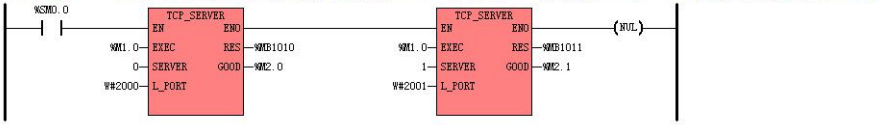
本例网络调试助手作为 TCP 客户端、PLC 作为 TCP 服务器进行说明。

LD

```
(* Network 0 *)
(* LINE参数     TRUE表示网线插着，   FALSE表示网线拔下了
CLIENT参数   DWORD，  每个bit表示一个TCP client的状态
SERVER参数   DWORD，  每个bit表示一个TCP server的状态
UDPPEER参数  DWORD，  每个bit表示一个UDP peer的状态 *)
```



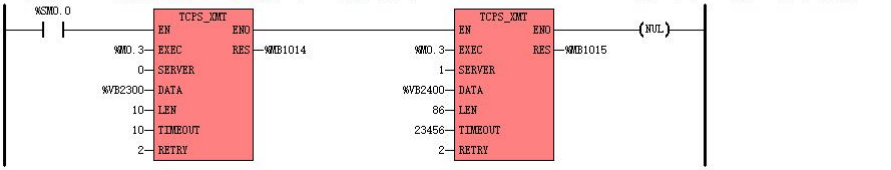

```
(* Network 1 *)
(* 本例子是输入参数为常量方式，  SERVER，L_PORT，这几个输入参数，  可以同时为常量，  或者同时为变量

开启TCP服务器

EXEC参数，      只能是变量，  上升沿时，执行一次本指令
SERVER参数，    INT常量，  范围0--15，也即最大支持16个tcp server
L_PORT参数，    WORD常量，  服务器的本地port端口，不支持0，

RES参数，       执行开始时，  此参数输出0，   执行结束时，  bit7 == TRUE，  bit0-bit6合起来的byte数值，表示错误码

GOOD参数，      TRUE表示有远端的客户端连接到本服务器，  FALSE表示没有，  每次执行EXEC时，  GOOD参数先会被赋值为FALSE *)
```



LD

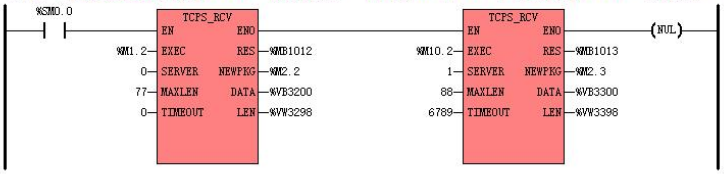
```
(* Network 2 *)
(* 输入参数为常量方式

关闭TCP服务器

EXEC参数,     只能是变量, 上升沿时, 执行一次本指令
SERVER参数,   INT常量, 范围0--15, 也即最大支持16个tcp server

RES参数,      执行开始时, 此参数输出0,  执行结束时, bit7 == TRUE, bit0-bit6合起来的byte数值, 表示错误码
   0   已经执行了关闭服务器操作, 不保证成功。
   1   SERVER参数错误 *)
```


```
(* Network 3 *)
(* 本例子是输入参数为常量方式, SERVER, LEN, TIMEOUT, RETRY, 这几个输入参数, 可以同时为常量, 或者同时为变量

数据发送

EXEC参数,     只能是变量, 上升沿时, 执行一次本指令
SERVER参数,   INT常量, 范围0--15, 也即最大支持16个tcp server
DATA参数,     BYTE变量, 待发送数据的第一个字节, 待发送数据的其他字节, 依次摆放
LEN参数,      INT常量, 发送数据的字节数, 不能为0, 不能为负数, 不能大于1460字节
TIMEOUT参数,  INT常量, 本次执行时, 超时时间, 单位ms
RETRY参数,    INT常量, 重试次数
RES参数,      执行开始时, 此参数输出0,  执行结束时, bit7 == TRUE, bit0-bit6合起来的byte数值, 表示错误码 *)
```




```
(* Network 4 *)
(* 发送的数据 *)
```


```
(* Network 5 *)
(* 本例子是输入参数为常量方式, SERVER, MAXLEN, TIMEOUT, 这几个输入参数, 可以同时为常量, 或者同时为变量

数据接收

备注:   在本例子中, 如果同时启动的话,  这个TIMEOUT为0的TCPS_RCV, 因为位于前面,
 所以, 总是会拦截并接收到server 0的新报文,  下面的TCPS_XMTRCV, 无法收到server 0新报文。 *)
```



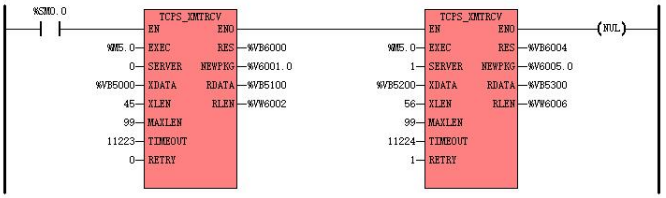
LD

(* Network 6 *)
(* 测试语句 观察TCPS_RCV指令从server 0接收的报文次数, *)

(* Network 7 *)
(* 测试语句 观察TCPS_RCV指令从server 1接收的报文次数, *)

(* Network 8 *)
(* 本例子是输入参数为常量方式, SERVER, XLEN, MAXLEN, TIMEOUT, RETRY, 这几个输入参数, 可以同时为常量, 或者同时为变量

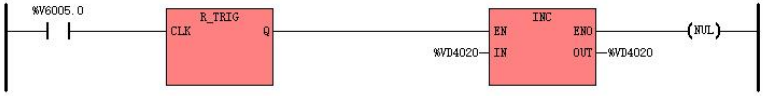
备注, 一次EXEC上升沿启动后, 只能接收1个报文。 取NEWPKG的上升沿, 读取报文。
备注, TCPC_XMTRCV, TCPS_XMTRCV, 用法相同。 *)

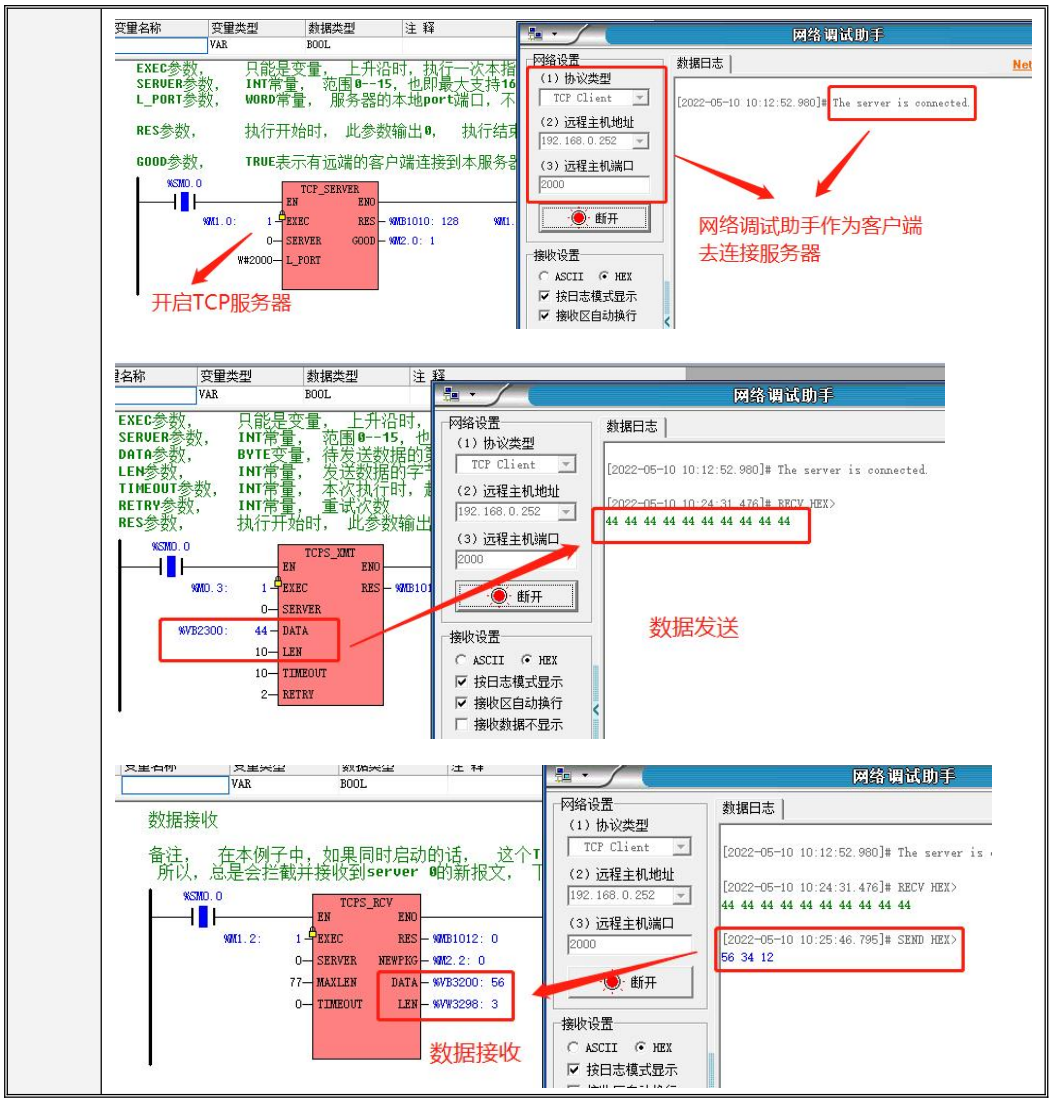


(* Network 9 *)
(* TCPS_XMTRCV用的发出字串 *)

(* Network 10 *)
(* 观察TCPS_XMTRCV 从server 0接收的报文次数, *)

(* Network 11 *)
(* 观察TCPS_XMTRCV 从server 1接收的报文次数, *)




网络调试助手
网络调试助手作为客户端
去连接服务器
开启TCP服务器
数据发送
数据接收

IL

```
(* Network 0 *)
(*LINE 参数 TRUE 表示网线插着， FALSE 表示网线拔下了 CLIENT 参数 DWORD， 每个 bit 表示一个 TCP client 的状态 SERVER 参数 DWORD， 每个 bit 表示一个 TCP server 的状态 UDPPEER 参数 DWORD， 每个 bit 表示一个 UDP peer 的状态*)
LD %SM0.0
ETH_STATUS %M333.7, %MD334, %MD338, %MD342
(* Network 1 *)
(*本例子是输入参数为常量方式， SERVER, L_PORT，这几个输入参数， 可以同时为常量， 或者同时为变量开启 TCP 服务器*)
LD %SM0.0
TCP_SERVER %M1.0, 0, W#2000, %MB1010, %M2.0
TCP_SERVER %M1.0, 1, W#2001, %MB1011, %M2.1
(* Network 2 *)
(*输入参数为常量方式关闭 TCP 服务器*)
LD %SM0.0
TCPS_CLOSE %M1.4, 0, %MB1016
TCPS_CLOSE %M1.5, 1, %MB1017
(* Network 3 *)
(*本例子是输入参数为常量方式*)
LD %SM0.0
TCPS_XMT %M0.3, 0, %VB2300, 10, 10, 2, %MB1014
TCPS_XMT %M0.3, 1, %VB2400, 86, 23456, 2, %MB1015
(* Network 4 *)
(*发送的数据*)
LD %SM0.1
FILL B#16#44, %VB2300, B#73
FILL B#16#55, %VB2400, B#83
```

IL

```
(* Network 5 *)
(*本例子是输入参数为常量方式*)
LD      %SM0.0
TCPS_RCV %M1.2, 0, 77, 0, %MB1012, %M2.2, %VB3200, %VW3298
TCPS_RCV %M10.2, 1, 88, 6789, %MB1013, %M2.3, %VB3300, %VW3398
(* Network 6 *)
(*测试语句　观察 TCPS_RCV 指令从 server 0 接收的报文次数*)
LD      %M2.2
R_TRIG
INC     %VD4008
(* Network 7 *)
(*测试语句　观察 TCPS_RCV 指令从 server 1 接收的报文次数*)
LD      %M2.3
R_TRIG
INC     %VD4012
(* Network 8 *)
(*本例子是输入参数为常量方式*)
LD      %SM0.0
TCPS_XMTRCV     %M5.0,     0,     %VB5000,     45,     99,     11223,
0, %VB6000, %V6001.0, %VB5100, %VW6002
TCPS_XMTRCV     %M5.0,     1,     %VB5200,     56,     99,     11224,
1, %VB6004, %V6005.0, %VB5300, %VW6006
(* Network 9 *)
(*TCPS_XMTRCV 用的发出字串*)
LD      %SM0.1
FILL    B#16#33, %VB5000, B#43
FILL    B#16#44, %VB5200, B#53
```

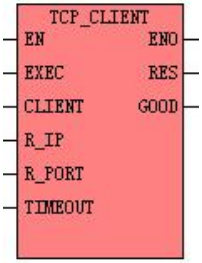
IL	(* Network 10 *) (*观察 TCPS_XMTRCV 从 server 0 接收的报文次数*) LD %V6001.0 R_TRIG INC %VD4016 (* Network 11 *) (*观察 TCPS_XMTRCV 从 server 1 接收的报文次数*) LD %V6005.0 R_TRIG INC %VD4020

10.3.3.4 TCP 客户端指令

10.3.3.4.1 TCP_CLIENT 指令

当需要使用本机作为 TCP 客户端时首先使用此指令开启客户端连接。

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	TCP_CLIENT			☑ K6
IL	TCP_ CLIENT	TCP_ CLIENT *EXEC, CLIENT, R_IP, R_PORT, TIMEOUT, RES, GOOD*	U	

参数	输入/输出	数据类型	允许使用的内存区
EXEC	输入	BOOL	I、Q、V、M、L、SM、RS、SR
CLIENT	输入	INT	M、V、L、常量
R_IP	输入	DWORD	M、V、L、常量
R_PORT	输入	WORD	M、V、L、常量
TIMEOUT	输入	INT	M、V、L、常量
RES	输出	BYTE	Q、M、V、L、SM
GOOD	输出	BOOL	Q、M、V、L、SM

注意：参数 *CLIENT, R_IP, R_PORT, TIMEOUT*,必须同时为常量类型或同时为内存类型。

➢ 参数说明

参数	描　述
EXEC	若检测到 *EXEC*的上升沿跳变，则该指令被触发执行,开启服务器连接。

CLIENT	客户端资源编号，范围 0--15，也即最大支持 16 个 TCP 客户端资源。
R_IP	远端的服务器的 ip 地址，不支持全 0，可以直接输入 IP 地址，例如：192.168.210.100
R_PORT	远端服务器的 port 端口，不支持 0。
TIMEOUT	本次执行时得我超时时间，单位 ms，不能大于 32767ms，不能为负数，0 表示一直等待连接成功。非 0 时，比如 5000， 表示 5 秒内没有连接服务器成功，就返回超时失败。
RES	执行开始时，此参数输出 0，执行结束时，bit7 == TRUE, bit0-bit6 合起来的 byte 数值，表示错误码 0 连接成功 1 CLIENT 参数错误 2 R_IP 或 R_PORT 参数错误 3 操作超时还没有连接成功 4 已经连接了服务器，无操作结束 5 TIMEOUT 参数错误 6 有太多正在尝试连接中的 TCP 活动(当 ModbusTCP 从站快速访问不存在的从站或断线的从站时容易出现，放慢速度即可自动消失)
GOOD	每次执行 EXEC 时，GOOD 参数先会被赋值为 FALSE。 TRUE 表示连接服务器成功，FALSE 表示连接失败。 此参数在指令执行后更新一次，不会动态更新，可通过 ETH_STATUS 指令来查看动态连接的变化。

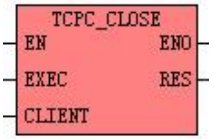
注意：当已经连接了一个远程服务器， 修改了 IP 或 PORT 指向了新的远程服务器，再次执行此指令，会自动关闭旧的连接，然后再连接新的远程服务器。

10.3.3.4.2 TCPC_CLOSE 指令

此指令用于关闭本机作为客户端的连接。

- 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K6

LD	TCPC_CLOSE			
IL	TCPC_CLOSE	TCPC_CLOSE *EXEC, SERVER, RES*	U	

参数	输入/输出	数据类型	允许使用的内存区
EXEC	输入	BOOL	I、Q、V、M、L、SM、RS、SR
CLIENT	输入	INT	M、V、L、常量
RES	输出	BYTE	Q、M、V、L、SM

➢ 参数说明

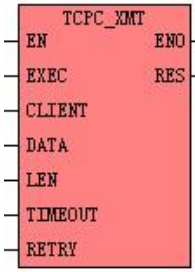
参数	描　述
EXEC	若检测到 *EXEC* 的上升沿跳变，则该指令被触发执行,关闭 TCP 客户端连接。
CLIENT	客户端资源编号，范围 0--15，也即最大支持 16 个 TCP 客户端资源。
RES	执行开始时，此参数输出 0，执行结束时，bit7 == TRUE,bit0-bit6 合起来的 byte 数值，表示错误码 0　已经执行了关闭客户端操作， 不保证成功。 1　CLIENT 参数错误。

10.3.3.4.3 TCPC_XMT 指令

此指令为本机作为客户端时的发送指令。

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	☑ K6

LD	TCPC_XMT			
IL	TCPC_XMT	TCPC_XMT *EXEC, CLIENT, DATA, LEN, TIMEOUT, RES*	U	

参数	输入/输出	数据类型	允许使用的内存区
EXEC	输入	BOOL	I、Q、V、M、L、SM、RS、SR
CLIENT	输入	INT	M、V、L、常量
DATA	输入	BYTE	M、V、L
LEN	输入	INT	M、V、L、常量
TIMEOUT	输入	INT	M、V、L、常量
RETRY	输入	INT	M、V、L、常量
RES	输出	BYTE	Q、M、V、L、SM

注意：参数 *CLIENT, LEN, TIMEOUT, TRTRY* 必须同时为常量类型或同时为内存类型。

- 参数说明

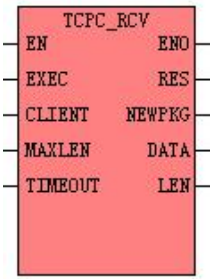
参数	描　述
EXEC	若检测到 *EXEC* 的上升沿跳变，则该指令被触发执行。
CLIENT	已经开启的客户端资源编号，范围 0--15。
DATA	待发送数据字节的首地址。
LEN	发送数据的字节数，不能为 0，不能为负数，不能大于 1460 字节。
TIMEOUT	本次执行时的超时时间，单位 ms，不能大于 32767ms，不能为负数，0 表示不等待，有连接就发送，没连接直接失败。非 0 时，比如 5000，表示 5 秒内一直等待连接客户端成功，成功后立刻发送， 一直没有连接就返回超时失败。
RETRY	重试次数，范围 0-255,输入超过此范围，自动截取本参数低字节的值， 发送失败后重试的次数，比如重试次数为 1，假设每次发送都失败，则启动发送后，一共发送 2 次。

RES	执行开始时，此参数输出 0，执行结束时，bit7 == TRUE,bit0-bit6 合起来的 byte 数值，表示错误码 0 发送完成，仅仅表示提交给了 TCP 协议栈，不代表远端已经收到，不代表远端必然收到 1 CLIENT 参数错误 2 发送长度错误 3 操作超时还没有发送完成 4 还没有连接到服务器 5 TIMEOUT 参数，输入错误 错误码 65--79，表示协议栈底层发送失败。

10.3.3.4.4 TCPC_RCV 指令

此指令为本机作为服务器时的接收指令。

- 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	TCPC_RCV			☑ K6
IL	TCPC_RCV	TCPC_RCV *EXEC, CLIENT, MAXLEN, TIMEOUT, RES, NEWPKG, DATA, LEN*	U	

参数	输入/输出	数据类型	允许使用的内存区
EXEC	输入	BOOL	I、Q、V、M、L、SM、RS、SR
CLIENT	输入	INT	M、V、L、常量
MAXLEN	输入	INT	M、V、L、常量

TIMEOUT	输入	INT	M、V、L、常量
RES	输出	BYTE	Q、M、V、L、SM
NEWPKG	输出	BOOL	Q、M、V、L、SM
DATA	输出	BYTE	Q、M、V、L、SM
LEN	输出	WORD	Q、M、V、L、SM

注意：参数 *CLENT, MAXLEN, TIMEOUT*,必须同时为常量类型或同时为内存类型。

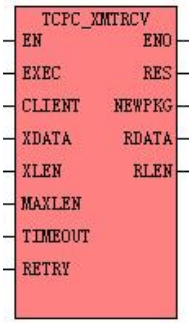
- 参数说明

参数	描　述
EXEC	若检测到 *EXEC*的上升沿跳变，则该指令被触发执行。
CLIENT	已经开启的客户端资源编号，范围 0--15。
MAXLEN	1 条报文最大可以接收的数据字节数，不能输入 0，不能为负数，当 MAXLEN 大于 1460 时，会被忽略，最大接收 1460 个字节。
TIMEOUT	本次执行时的超时时间，单位 ms，不能大于 32767ms，不能为负数，0 表示永久等待，一直处于等待客户端报文的状态。非 0 时比如 5000，表示 5 秒内一直等待客户端发来的报文，超时后就不再接收报文。
RES	执行开始时，此参数输出 0，执行结束时，bit7 == TRUE,bit0-bit6 合起来的 byte 数值，表示错误码 0　未使用 1　CLIENT 参数错误 2　MAXLEN 长度错误 3　操作超时，结束接收 5　TIMEOUT 参数，输入错误
NEWPKG	客户端接收到了一个新报文，NEWPKG 就产生一个上升沿，　只维持 1 个扫描周期。
DATA	接收缓冲区的起始字节，注意这个缓冲区是和 MAXLEN 匹配使用的，比如，MAXLEN 是 100，DATA=VB0,此时 VB0--VB99 都是缓冲区，只能读取，不能做其他用处。缓冲区用户可自行清空，否则下次接收的报文长度小于上次时，缓冲区会残留上次报文数据。
LEN	接收报文的长度，当实际报文长度小于等于 MAXLEN 时，LEN 为实际报文长度。当实际报文长度超过 MAXLEN 时，LEN=MAXLEN。

10.3.3.4.5 TCPC_XMTRCV 指令

此指令为本机作为客户端时的收发指令。

- 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	TCPC_XMTRCV			☑ K6
IL	TCPC_XMTRCV	TCPC_RCV *EXEC, CLIENT, XDATA, XLEN, MAXLEN, TIMEOUT, RETRY, RES, NEWPKG, RDATA, RLEN*	U	

参数	输入/输出	数据类型	允许使用的内存区
EXEC	输入	BOOL	I、Q、V、M、L、SM、RS、SR
CLIENT	输入	INT	M、V、L、常量
XDATA	输入	BYTE	M、V、L
XLEN	输入	INT	M、V、L、常量
MAXLEN	输入	INT	M、V、L、常量
TIMEOUT	输入	INT	M、V、L、常量
RETRY	输入	INT	M、V、L、常量
RES	输出	BYTE	Q、M、V、L、SM
NEWPKG	输出	BOOL	Q、M、V、L、SM
RDATA	输出	BYTE	Q、M、V、L、SM
RLEN	输出	WORD	Q、M、V、L、SM

注意：参数 *CLIENT, XLEN, MAXLEN, TIMEOUT, RETRY* 必须同时为常量类型或同时为内存类型。

➢ 参数说明

参数	描　述
EXEC	若检测到 *EXEC* 的上升沿跳变，则该指令被触发执行。
CLIENT	已经开启的客户端资源编号，范围 0--15。
XDATA	待发送数据字节的首地址。
XLEN	发送数据的字节数，不能为 0，不能为负数，不能大于 1460 字节。
MAXLEN	1 条报文最大可以接收的数据字节数，不能输入 0，不能为负数，当 MAXLEN 大于 1460 时，会被忽略，最大接收 1460 个字节。
TIMEOUT	本次执行时的超时时间，单位 ms，不能是 0ms，不能大于 32767ms，不能为负数。 例如输入 5000， 表示 5 秒内， 进行如下动作序列: a. 一直等待连接服务器成功，成功后立刻发送要发送的数据，一直没有连接成功，返回失败并结束。 b. 发送成功后， 等待第一包接收数据，一直没有等到，返回失败并结束。 c. 发送成功后， 接收到了一包数据， 本指令成功结束。
RETRY	重试次数，范围 0-255,输入超过此范围，自动截取本参数低字节的值。 发送失败或者接收失败后的重试次数，比如重试次数为 1，假设每次发送都失败，则启动 EXEC 后，一共发送 2 次。
RES	执行开始时，此参数输出 0，执行结束时，bit7 == TRUE, bit0-bit6 合起来的 byte 数值，表示错误码 0　指令成功结束，发出报文，并接收到了一个报文 1　CLIENT 参数错误 2　长度错误, MAXLEN 为 0,或者 XLEN 为 0，或者 XLEN 大于 1460 字节,当 MAXLEN 大于 1460 时，会被忽略，只接收最大 1460 长度报文 3　操作超时，没有收到报文 4　发送失败，一直没有连接到服务器 5　TIMEOUT 参数, 输入错误
NEWPKG	本指令 EXEC 启动时会自动变成 FALSE, 成功接收到了一个报文变为 TRUE，直到下一次启动本指令。
RDATA	接收缓冲区的起始字节，注意这个缓冲区是和 MAXLEN 匹配使用的，比如，MAXLEN 是 100，DATA=VB0,此时 VB0--VB99 都是缓冲区，只能读取，不能做其他用处。缓冲区用户可自行清空，否则下次接收的报文长度小于上次时，缓冲区会残留上次报文数据。

RLEN	接收报文的长度，当实际报文长度小于等于 MAXLEN 时，LEN 为实际报文长度。当实际报文长度超过 MAXLEN 时，LEN=MAXLEN。

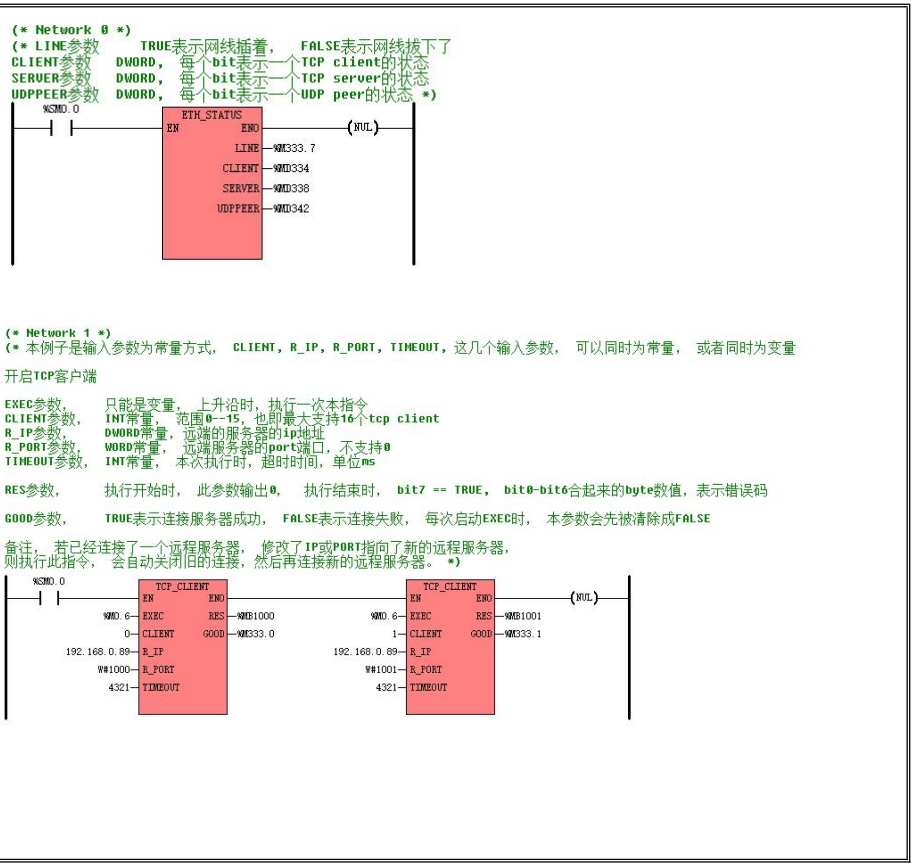
注意： 启动一次 EXEC 上升沿只能接收 1 个报文。 取 NEWPKG 的上升沿，读取报文。

TCPC_RCV 和 TCPC_XMTRCV 指令同时使用时，只有在程序中靠前的那条指令可以接收到报文。

10.3.3.4.6 示例

本例网络调试助手作为 TCP 服务器、PLC 作为 TCP 客户端进行说明。

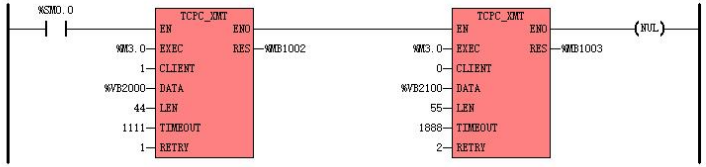
LD



LD

```
(* Network 2 *)
(* 输入参数为常量方式

关闭TCP客户端

EXEC参数,      只能是变量, 上升沿时, 执行一次本指令
CLIENT参数,    INT常量, 范围0--15, 也即最大支持16个tcp client

RES参数,       执行开始时, 此参数输出0,  执行结束时, bit7 == TRUE, bit0-bit6合起来的byte数值, 表示错误码 *)
```

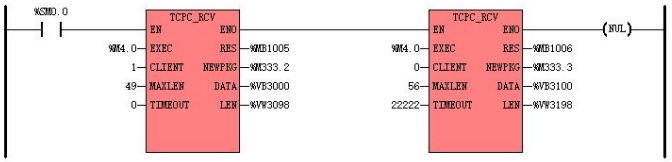

```
(* Network 3 *)
(* 本例子是输入参数为常量方式, CLIENT, LEN, TIMEOUT, RETRY, 这几个输入参数, 可以同时为常量, 或者同时为变量

发送数据

EXEC参数,      只能是变量, 上升沿时, 执行一次本指令
CLIENT参数,    INT常量, 范围0--15, 也即最大支持16个tcp client
DATA参数,      BYTE变量, 待发送数据的第一个字节, 待发送数据的其他字节, 依次摆放
LEN参数,       INT常量, 发送数据的字节数, 不能为0, 不能为负数, 不能大于1460字节
TIMEOUT参数,   INT常量, 本次执行时, 超时时间, 单位ms
RETRY参数,     INT常量, 重试次数

RES参数,       执行开始时, 此参数输出0,  执行结束时, bit7 == TRUE, bit0-bit6合起来的byte数值, 表示错误码

备注, TCPC_XMT, TCPS_XMT, 用法相同。 *)
```




```
(* Network 4 *)
(* TCP_XMT用的发出字串 *)
```

LD

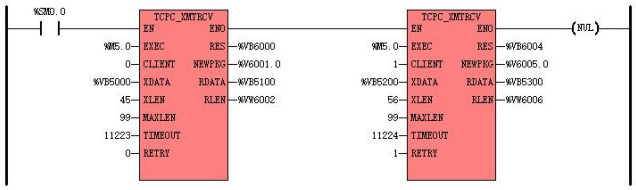
```
(* Network 5 *)
(* 本例子是输入参数为常量方式， CLIENT，MAXLEN，TIMEOUT，这几个输入参数， 可以同时为常量， 或者同时为变量

接收数据

EXEC参数，      只能是变量， 上升沿时，执行一次本指令
CLIENT参数，    INT常量， 范围0--15，也即最大支持16个tcp client
MAXLEN参数，    INT常量， 1个报文，最大可以接收的数据的字节数
TIMEOUT参数，   INT常量， 本次执行时，超时时间，单位ms

RES参数，       执行开始时， 此参数输出0，  执行结束时， bit7 == TRUE， bit0-bit6合起来的byte数值，表示错误码

NEWPKG参数，    客户端接收到了一个新报文， 就产生一个上升沿， 只维持1个扫描周期
DATA参数，      接收缓冲区的起始字节
LEN参数，       本次实际接收的报文大小

备注， 在本例子中，如果同时启动的话， 这个TIMEOUT为0的TCPC_RCV，因为位于前面，
所以，总是会拦截并接收到client 1的新报文， 下面的TCPC_XMTRCV，无法收到client 1新报文。 *)
```



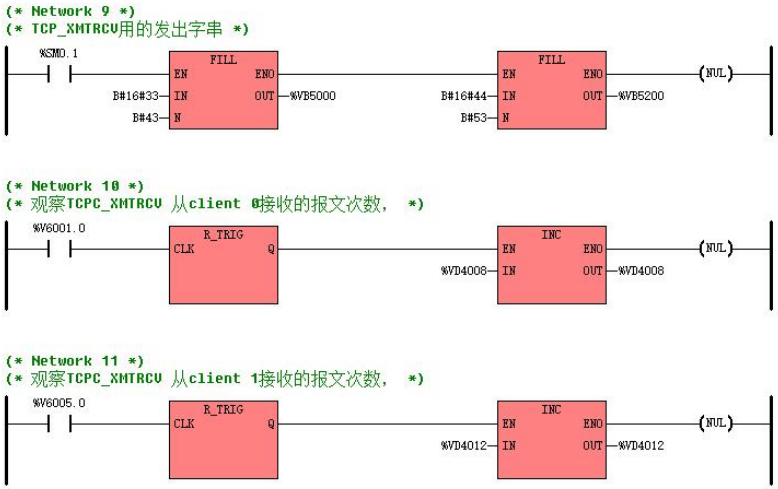

```
(* Network 6 *)
(* 观察TCPC_RCV指令， 从client 1接收的报文次数 *)
```

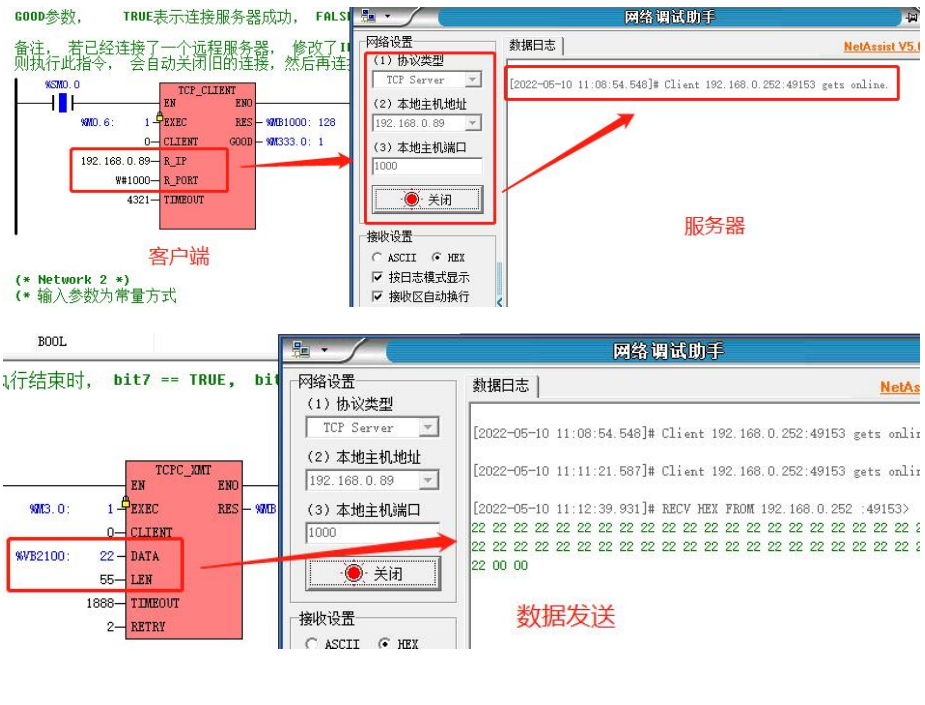

```
(* Network 7 *)
(* 观察TCPC_RCV指令， 从client 0接收的报文次数 *)
```

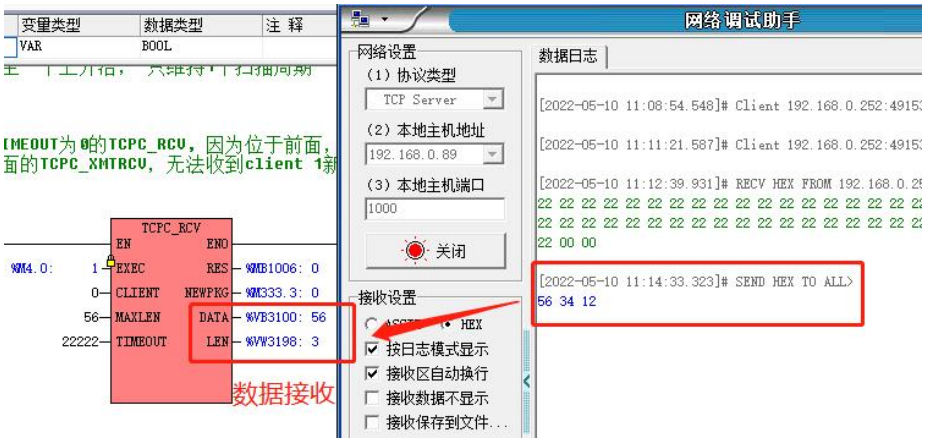

```
(* Network 8 *)
(* 本例子是输入参数为常量方式， CLIENT，XLEN，MAXLEN，TIMEOUT，RETRY，这几个输入参数， 可以同时为常量， 或者同时为变量

先发送数据再接收数据

EXEC参数，      只能是变量， 上升沿时，执行一次本指令
CLIENT参数，    INT常量， 范围0--15，也即最大支持16个tcp client
XDATA参数，     BYTE变量，待发送数据的第一个字节， 待发送数据的其他字节，依次摆放
XLEN参数，      INT常量， 发送数据的字节数， 不能为0， 不能大于1460字节， 不能为负数
MAXLEN参数，    INT常量， 一次启动本指令，最大可以接收数据的字节数， 最大为1460字节， 不能为负数
TIMEOUT参数，   INT常量， 本次执行时，超时时间，单位ms，
RETRY参数，     INT常量， 重试次数

RES参数，       执行开始时， 此参数输出0，  执行结束时， bit7 == TRUE， bit0-bit6合起来的byte数值，表示错误码

备注， TCPC_XMTRCV， TCPS_XMTRCV，用法相同。 *)
```



LD


(* Network 9 *)
(* TCP_XMTRCV用的发出字串 *)
%SM0.1
FILL
EN
ENO
B#16#33
IN
OUT
%VB5000
B#43
N
B#16#44
%VB5200
B#53
(NUL)
(* Network 10 *)
(* 观察TCPC_XMTRCV 从client 0接收的报文次数, *)
%V6001.0
R_TRIG
CLK
Q
INC
%VD4008
(NUL)
(* Network 11 *)
(* 观察TCPC_XMTRCV 从client 1接收的报文次数, *)
%V6005.0
%VD4012


GOOD参数, TRUE表示连接服务器成功, FALSE
备注, 若已经连接了一个远程服务器, 修改了IP
则执行此指令, 会自动关闭旧的连接, 然后再连
%SM0.0
TCP_CLIENT
%M0.6:
EXEC
CLIENT
192.168.0.89
R_IP
W#1000
R_PORT
4321
TIMEOUT
%MB1000: 128
%M333.0: 1
客户端
(* Network 2 *)
(* 输入参数为常量方式
网络调试助手
网络设置
(1) 协议类型
TCP Server
(2) 本地主机地址
192.168.0.89
(3) 本地主机端口
1000
关闭
接收设置
ASCII
HEX
按日志模式显示
接收区自动换行
数据日志
NetAssist V5.
[2022-05-10 11:08:54.548]# Client 192.168.0.252:49153 gets online.
服务器
BOOL
行结束时, bit7 == TRUE, bit
TCPC_XMT
%M3.0:
%VB2100:
DATA
LEN
1888
TIMEOUT
2
RETRY
[2022-05-10 11:11:21.587]# Client 192.168.0.252:49153 gets onlin
[2022-05-10 11:12:39.931]# RECV HEX FROM 192.168.0.252 :49153>
22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22
22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22
22 00 00
数据发送

<table>
<tr><td></td><td> 
</td></tr>
<tr><td>IL</td><td>(* Network 0 *)
(*LINE 参数 TRUE 表示网线插着， FALSE 表示网线拔下了 CLIENT 参数 DWORD， 每个 bit 表示一个 TCP client 的状态 SERVER 参数 DWORD， 每个 bit 表示一个 TCP server 的状态 UDPPEER 参数 DWORD， 每个 bit 表示一个 UDP peer 的状态*)
LD %SM0.0
ETH_STATUS %M333.7, %MD334, %MD338, %MD342
(* Network 1 *)
(*本例子是输入参数为常量方式备注， 若已经连接了一个远程服务器， 修改了 IP 或 PORT 指向了新的远程服务器， 则执行此指令， 会自动关闭旧的连接，然后再连接新的远程服务器。*)
LD %SM0.0
TCP_CLIENT %M0.6, 0, 192.168.0.89, W#1000, 4321, %MB1000, %M333.0
TCP_CLIENT %M0.6, 1, 192.168.0.89, W#1001, 4321, %MB1001, %M333.1</td></tr>
</table>

IL	(* Network 2 *) (*输入参数为常量方式关闭 TCP 客户端*) LD %SM0.0 TCPC_CLOSE %M0.5, 0, %MB1008 TCPC_CLOSE %M0.7, 1, %MB1009 (* Network 3 *) (*本例子是输入参数为常量方式， CLIENT, LEN, TIMEOUT, RETRY，这几个输入参数， 可以同时为常量， 或者同时为变量发送数据*) LD %SM0.0 TCPC_XMT %M3.0, 1, %VB2000, 44, 1111, 1, %MB1002 TCPC_XMT %M3.0, 0, %VB2100, 55, 1888, 2, %MB1003 (* Network 4 *) (*TCP_XMT 用的发出字串*) LD %SM0.1 FILL B#16#11, %VB2000, B#43 FILL B#16#22, %VB2100, B#53 (* Network 5 *) (*本例子是输入参数为常量方式， CLIENT, MAXLEN, TIMEOUT，这几个输入参数， 可以同时为常量， 或者同时为变量接收数据*) LD %SM0.0 TCPC_RCV %M4.0, 1, 49, 0, %MB1005, %M333.2, %VB3000, %VW3098 TCPC_RCV %M4.0, 0, 56, 22222, %MB1006, %M333.3, %VB3100, %VW3198 (* Network 6 *) (*观察 TCPC_RCV 指令， 从 client 1 接收的报文次数*) LD %M333.2 R_TRIG INC %VD4000

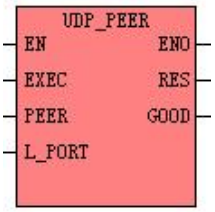
IL	(* Network 7 *) (*观察 TCPC_RCV 指令， 从 client 0 接收的报文次数*) LD %M333.3 R_TRIG INC %VD4004 (* Network 8 *) (*本例子是输入参数为常量方式，CLIENT, XLEN, MAXLEN, TIMEOUT, RETRY，这几个输入参数， 可以同时为常量， 或者同时为变量先发送数据再接收数据*) LD %SM0.0 TCPC_XMTRCV %M5.0, 0, %VB5000, 45, 99, 11223, 0, %VB6000, %V6001.0, %VB5100, %VW6002 TCPC_XMTRCV %M5.0, 1, %VB5200, 56, 99, 11224, 1, %VB6004, %V6005.0, %VB5300, %VW6006 (* Network 9 *) (*TCP_XMTRCV 用的发出字串*) LD %SM0.1 FILL B#16#33, %VB5000, B#43 FILL B#16#44, %VB5200, B#53 (* Network 10 *) (*观察 TCPC_XMTRCV 从 client 0 接收的报文次数*) LD %V6001.0 R_TRIG INC %VD4008 (* Network 11 *) (*观察 TCPC_XMTRCV 从 client 1 接收的报文次数*) LD %V6005.0 R_TRIG INC %VD4012

10.3.3.5 UDP 指令

10.3.3.5.1 UDP_PEER 指令

当需要使用本机 UDP 资源通信时首先使用此指令开启 UDP 通信。

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	UDP_PEER			☑ K6
IL	UDP_PEER	UDP_PEER *EXEC, PEER, L_PORT, RES, GOOD*	U	

参数	输入/输出	数据类型	允许使用的内存区
EXEC	输入	BOOL	I、Q、V、M、L、SM、RS、SR
PEER	输入	INT	M、V、L、常量
L_PORT	输入	WORD	M、V、L、常量
RES	输出	BYTE	Q、M、V、L、SM
GOOD	输出	BOOL	Q、M、V、L、SM

注意：参数 *PEER, L_PORT* 必须同时为常量类型或同时为内存类型。

➢ 参数说明

参数	描　述
EXEC	若检测到 *EXEC* 的上升沿跳变，则该指令被触发执行,开启 UDP 通信。
PEER	UDP PEER 资源编号，范围 0--15，也即最大支持 16 个 UDP PEER 资源。
L_PORT	本地 UDP PEER 的 port 端口，不支持 0。

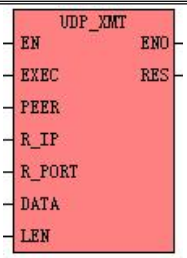
RES	执行开始时，此参数输出 0，执行结束时，bit7 == TRUE，bit0-bit6 合起来的 byte 数值，表示错误码 0 创建成功 1 PEER 参数错误 2 PORT 参数输入错误 3 创建失败 4 当前 PEER 已经创建过了
GOOD	每次执行 EXEC 时，GOOD 参数先会被赋值为 FALSE。 TRUE 表示连接服务器成功，FALSE 表示连接失败。 此参数在指令执行后更新一次，不会动态更新，可通过 ETH_STATUS 指令来查看动态连接的变化。

注意：因为 PLC 本机只有 1 个网卡，所以不需要输入本地 IP， 默认取 PLC 的本地 IP。

10.3.3.5.2 UDP_XMT 指令

此指令为本机 UDP 通信时的发送指令。

- 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	UDP_XMT			☑ K6

IL	UDP_XMT	UDP_XMT *EXEC, PEER, R_IP, R_PORT, DATA, LEN, RES*	U	

参数	输入/输出	数据类型	允许使用的内存区
EXEC	输入	BOOL	I、Q、V、M、L、SM、RS、SR
PEER	输入	INT	M、V、L、常量
R_IP	输入	DWORD	M、V、L、常量
R_PORT	输入	WORD	M、V、L、常量
DATA	输入	BYTE	M、V、L
LEN	输入	INT	M、V、L、常量
RES	输出	BYTE	Q、M、V、L、SM

注意：参数 *PEER, R_IP, R_PORT, LEN* 必须同时为常量类型或同时为内存类型。

- 参数说明

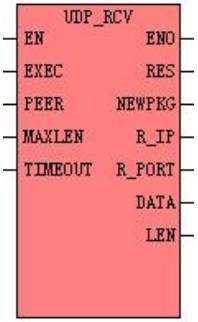
参数	描　述
EXEC	若检测到 *EXEC* 的上升沿跳变，则该指令被触发执行。
PEER	已经开启的 UDP PEER 资源编号，范围 0--15。
R_IP	远端的 UDP 的 ip 地址，不支持全 0,可以直接输入 IP 地址，比如：192.168.0.250。
R_PORT	远端 UDP 的 port 端口，不支持 0。
DATA	待发送数据字节的首地址。
LEN	发送数据的字节数，不能为 0，不能为负数，不能大于 1460 字节。

RES	执行开始时，此参数输出 0，执行结束时，bit7 == TRUE,bit0-bit6 合起来的 byte 数值，表示错误码 0 发送完成，仅仅表示提交给了 UDP 协议栈，不代表远端已经收到，不代表远端必然收到 1 PEER 参数错误 2 发送长度错误 3 备用 4 没有已经创建成功的 UDP PEER 可以用来发送数据

10.3.3.5.3 UDP_RCV 指令

此指令为本机作为 UDP 通信时的接收指令。

- 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	UDP_RCV			☑ K6
IL	UDP_RCV	UDP_RCV *EXEC, PEER, MAXLEN, TIMEOUT, RES, NEWPKG, R_IP, R_PORT, DATA, LEN*	U	

参数	输入/输出	数据类型	允许使用的内存区
EXEC	输入	BOOL	I、Q、V、M、L、SM、RS、SR

PEER	输入	INT	M、V、L、常量
MAXLEN	输入	INT	M、V、L、常量
TIMEOUT	输入	INT	M、V、L、常量
RES	输出	BYTE	Q、M、V、L、SM
NEWPKG	输出	BOOL	Q、M、V、L、SM
R_IP	输出	DWORD	Q、M、V、L、SM
R_PORT	输出	WORD	Q、M、V、L、SM
DATA	输出	BYTE	Q、M、V、L、SM
LEN	输出	WORD	Q、M、V、L、SM

注意：参数 *PEER, MAXLEN, TIMEOUT*,必须同时为常量类型或同时为内存类型。

- 参数说明

参数	描　述
EXEC	若检测到 *EXEC* 的上升沿跳变，则该指令被触发执行。
PEER	已经开启的 UDP PEER 资源编号，范围 0--15。
MAXLEN	1 条报文最大可以接收的数据字节数，不能输入 0，不能为负数，当 MAXLEN 大于 1460 时，会被忽略，最大接收 1460 个字节。
TIMEOUT	本次执行时的超时时间，单位 ms，不能大于 32767ms，不能为负数，0 表示永久等待，一直处于等待 UDP 报文的状态。非 0 时比如 5000，表示 5 秒内一直等待远端 UDP 发来的报文，超时后就不再接收报文。
RES	执行开始时，此参数输出 0，执行结束时，bit7 == TRUE,bit0-bit6 合起来的 byte 数值，表示错误码 0　备用 1　PEER 参数错误 2　MAXLEN 长度错误 3　操作超时，结束接收 4　备用 5　用户输入的 TIMEOUT 参数错误

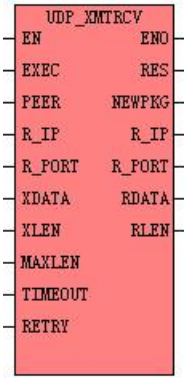
NEWPKG	接收到了一个 UDP 新报文，NEWPKG 就产生一个上升沿， 只维持 1 个扫描周期。
R_IP	NEWPKG 上升沿的瞬间，接收到那条消息的远端的 UDP 的 ip 地址。
R_PORT	NEWPKG 上升沿的瞬间，接收到那条消息的远端的 UDP 的 port 端口。
DATA	接收缓冲区的起始字节，注意这个缓冲区是和 MAXLEN 匹配使用的，比如，MAXLEN 是 100，DATA=VB0,此时 VB0--VB99 都是缓冲区，只能读取，不能做其他用处。缓冲区用户可自行清空，否则下次接收的报文长度小于上次时，缓冲区会残留上次报文数据。
LEN	接收报文的长度，当实际报文长度小于等于 MAXLEN 时，LEN 为实际报文长度。当实际报文长度超过 MAXLEN 时，LEN=MAXLEN。

注意：
1、一次 EXEC 上升沿启动后，可以接收任意多个报文。 注意观察 NEWPKG 即可。
2、NEWPKG 参数上升沿时，R_IP，R_PORT，DATA，LEN 这些参数的输出才有意义，其他时间维持不变。

10.3.3.5.4 UDP_XMTRCV 指令

此指令为本机 UDP 通信时的收发指令。

- 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	UDP_XMTRCV			☑ K6

IL	UDP_XMTRCV	UDP_RCV *EXEC，PEER，R_IP，R_PORT，XDATA，XLEN，MAXLEN，TIMEOUT，RETRY，RES，NEWPKG，R_IP，R_PORT，DATA，LEN*	U	

参数	输入/输出	数据类型	允许使用的内存区
EXEC	输入	BOOL	I、Q、V、M、L、SM、RS、SR
PEER	输入	INT	M、V、L、常量
R_IP	输入	DWORD	M、V、L、常量
R_PORT	输入	WORD	M、V、L、常量
XDATA	输入	BYTE	M、V、L
XLEN	输入	INT	M、V、L、常量
MAXLEN	输入	INT	M、V、L、常量
TIMEOUT	输入	INT	M、V、L、常量
RETRY	输入	INT	M、V、L、常量
RES	输出	BYTE	Q、M、V、L、SM
NEWPKG	输出	BOOL	Q、M、V、L、SM
R_IP	输出	DWORD	Q、M、V、L、SM
R_PORT	输出	WORD	Q、M、V、L、SM
DATA	输出	BYTE	Q、M、V、L、SM
RLEN	输出	WORD	Q、M、V、L、SM

注意：参数 *PEER，R_IP,R_PORT,XLEN，MAXLEN，TIMEOUT，RETRY* 必须同时为常量类型或同时为内存类型。

- 参数说明

参数	描　述
EXEC	若检测到 *EXEC* 的上升沿跳变，则该指令被触发执行。
PEER	已经开启的 UDP PEER 资源编号，范围 0—15。
R_IP	发送给远端的 UDP 的 ip 地址，不支持全 0,可以直接输入 IP 地址，比如：192.168.0.250。

R_PORT	发送给远端 UDP 的 port 端口，不支持 0。
XDATA	待发送数据字节的首地址。
XLEN	发送数据的字节数，不能为 0，不能为负数，不能大于 1460 字节。
MAXLEN	1 条报文最大可以接收的数据字节数，不能输入 0，不能为负数，当 MAXLEN 大于 1460 时，会被忽略，最大接收 1460 个字节。
TIMEOUT	本次执行时的超时时间，单位 ms，不能是 0ms，不能大于 32767ms，不能为负数。 例如输入 5000， 表示 5 秒内， 进行如下动作序列: a. 一直等待 UDP PEER 建立成功，成功后立刻发送要发送的数据，一直没有连接成功，返回失败并结束。 b. 发送成功后，等待第一包接收数据， 一直没有等到，返回失败并结束。 c. 发送成功后， 接收到了一包数据， 本指令成功结束。
RETRY	重试次数，范围 0-255,输入超过此范围，自动截取本参数低字节的值。 发送失败或者接收失败后的重试次数，比如重试次数为 1，假设每次发送都失败，则启动 EXEC 后，一共发送 2 次。
RES	执行开始时，此参数输出 0，执行结束时，bit7 == TRUE, bit0-bit6 合起来的 byte 数值，表示错误码 0 指令成功结束，发出报文，并接收到了一个报文 1 PEER 参数错误 2 长度错误, MAXLEN 为 0，或者 XLEN 为 0，或者 XLEN 大于 1460 字节,当 MAXLEN 大于 1460 时会被忽略，只接收最大 1460 长度报文 3 操作超时，没有收到报文 4 发送失败，一直没有 UDP PEER 建立成功 5 TIMEOUT 参数, 输入错误
NEWPKG	本指令 EXEC 启动时会自动变成 FALSE, 成功接收到了一个报文变为 TRUE，直到下一次启动本指令。
R_IP	NEWPKG 上升沿的瞬间， 接收到这条消息的远端的 UDP 的 ip 地址。因为 UDP 是无连接的，你可能发报文给 A 从站，等待回执过程中，收到了 B 从站的回执。
R_PORT	NEWPKG 上升沿的瞬间， 接收到这条消息的远端的 UDP 的 port 端口。因为 UDP 是无

	连接的，你可能发报文给 A 从站，等待回执过程中，收到了 B 从站的回执。
RDATA	接收缓冲区的起始字节，注意这个缓冲区是和 MAXLEN 匹配使用的，比如，MAXLEN 是 100，DATA=VB0,此时 VB0--VB99 都是缓冲区，只能读取，不能做其他用处。缓冲区用户可自行清空，否则下次接收的报文长度小于上次时，缓冲区会残留上次报文数据。
RLEN	接收报文的长度，当实际报文长度小于等于 MAXLEN 时，LEN 为实际报文长度。当实际报文长度超过 MAXLEN 时，LEN=MAXLEN。

注意： 启动一次 EXEC 上升沿只能接收 1 个报文。 取 NEWPKG 的上升沿，读取报文。

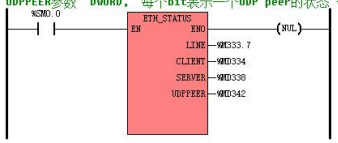
UDP_RCV 和 UDP_XMTRCV 指令同时使用时，只有在程序中靠前的那条指令可以接收到报文。

10.3.3.5.5 示例

本例网络调试助手和 PLC 通过 UDP 通讯。

LD

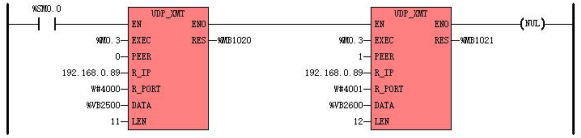
```
(* Network 0 *)
(* LINE参数     TRUE表示网线插着，   FALSE表示网线拔下了
CLIENT参数   DWORD，每个bit表示一个TCP client的状态
SERVER参数   DWORD，每个bit表示一个TCP server的状态
UDPPEER参数  DWORD，每个bit表示一个UDP peer的状态 *)
```




```
(* Network 1 *)
(* 本例子是输入参数为常量方式， PEER，L_PORT，这几个输入参数， 可以同时为常量， 或者同时为变量

开启UDP

EXEC参数，      只能是变量， 上升沿时，执行一次本指令
PEER参数，      INT常量， 范围0--15，也即最大支持16个UDP peer
L_PORT参数，    WORD常量，本地UDP peer的port端口，不支持0，

RES参数，       执行开始时， 此参数输出0，  执行结束时， bit7 == TRUE， bit0-bit6合起来的byte数值，表示错误码

GOOD参数，  当RES的bit7的上升沿瞬间，  TRUE表示创建成功，  FALSE表示创建失败

注意1  因为PLC本机只有1个网卡，所以不需要输入本地Ip，  默认取PLC的本地ip. *)
```

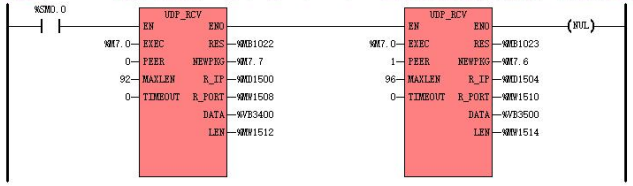

```
(* Network 2 *)
(* UPD_XMT用的发送字串 *)
```


```
(* Network 3 *)
(* 本例子是输入参数为常量方式， PEER，R_IP，R_PORT，LEN，这几个输入参数， 可以同时为常量， 或者同时为变量

发送数据

EXEC参数，      只能是变量， 上升沿时，执行一次本指令
PEER参数，      INT常量， 范围0--15，也即最大支持16个UDP peer
R_IP参数，      DWORD常量，远端的UDP的ip地址
R_PORT参数，    WORD常量， 远端UDP的port端口，不支持0，
DATA参数，      BYTE变量，待发送数据的第一个字节， 待发送数据的其他字节，依次摆放
LEN参数，       INT常量， 发送数据的字节数， 不能为0， 不能为负数， 不能大于1460

RES参数，       执行开始时， 此参数输出0，  执行结束时， bit7 == TRUE， bit0-bit6合起来的byte数值，表示错误码 *)
```



LD

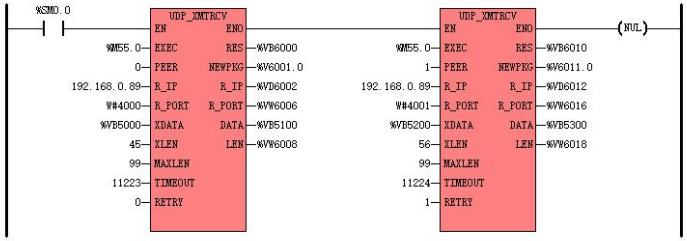
```
(* Network 4 *)
(* 本例子是输入参数为常量方式， PEER, MAXLEN, TIMEOUT 这几个输入参数， 可以同时为常量， 或者同时为变量

备注1,  当收到的报文大于MAXLEN时， 会截取MAXLEN的长度， 然后输出， 当收到的报文小于等于MAXLEN时， 按实际大小输出
备注2,  一次EXEC上升沿启动后， 可以接收任意多个报文。 注意观察NEWPKG即可。
备注3,  NEWPKG参数上升沿时， R_IP, R_PORT, DATA, LEN这些参数的输出才有意义， 其他时间维持不变 *)
```




```
(* Network 5 *)
(* 观察UDP_RCV指令从peer 0接收的报文次数， *)
```

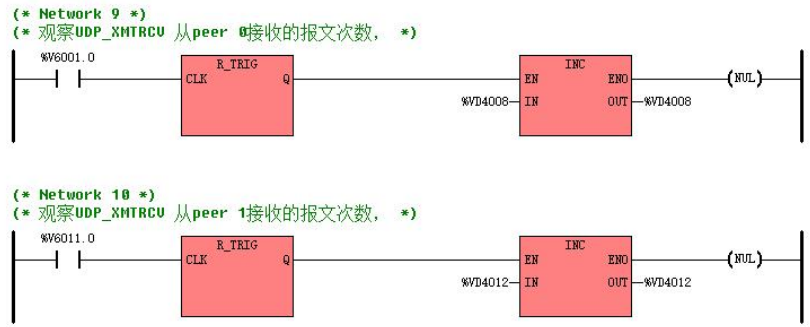

```
(* Network 6 *)
(* 观察UDP_RCV指令从peer 1接收的报文次数， *)
```

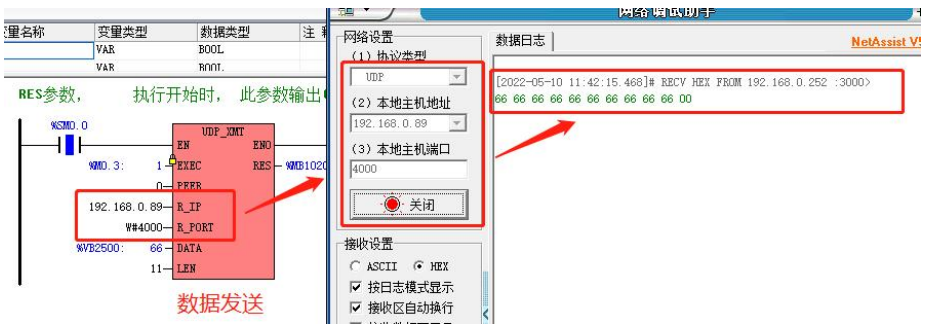

```
(* Network 7 *)
(* 本例子是输入参数为常量方式,
 PEER, R_IP, R_PORT, XLEN, MAXLEN, TIMEOUT, RETRY, 这几个输入参数， 可以同时为常量， 或者同时为变量 *)
```

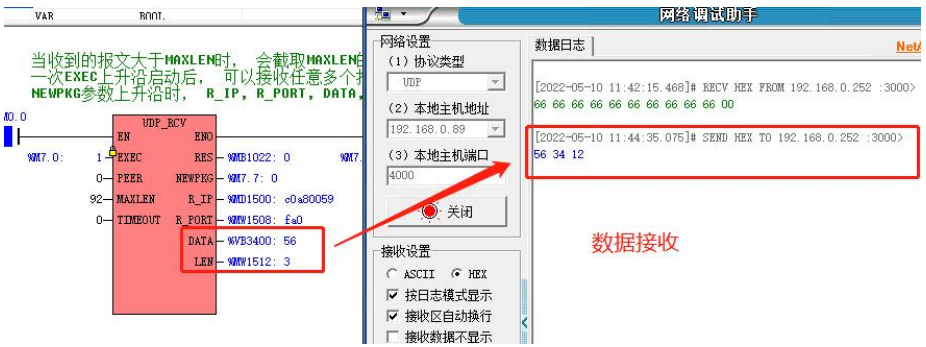



```
(* Network 8 *)
(* UDP_XMTRCV用的发出字串 *)
```

LD


(* Network 9 *)
(* 观察UDP_XMTRCV 从peer 0接收的报文次数， *)
%M6001.0
R_TRIG
CLK
Q
INC
EN
ENO
%VD4008
IN
OUT
%VD4008
(NUL)
(* Network 10 *)
(* 观察UDP_XMTRCV 从peer 1接收的报文次数， *)
%M6011.0
R_TRIG
CLK
Q
INC
EN
ENO
%VD4012
IN
OUT
%VD4012
(NUL)


RES参数，
执行开始时，
%SM0.0
UDP_XMT
EN
ENO
%M0.3:
EXEC
RES
PEER
192.168.0.89
R_IP
W#4000
R_PORT
%VB2500:
DATA
LEN
数据发送
网络调试助手
网络设置
(1) 协议类型
UDP
(2) 本地主机地址
192.168.0.89
(3) 本地主机端口
4000
关闭
接收设置
ASCII
HEX
按日志模式显示
接收区自动换行
数据日志
[2022-05-10 11:42:15.468]# RECV HEX FROM 192.168.0.252 :3000>
66 66 66 66 66 66 66 66 66 66 00


当收到的报文大于MAXLEN时，
一次EXEC上升沿启动后，
NEWPKG参数上升沿时，
R_IP, R_PORT, DATA,
%M0.0
UDP_RCV
EN
ENO
%M7.0:
EXEC
RES
%MB1022: 0
PEER
NEWPKG
%M7.7: 0
MAXLEN
R_IP
%MD1500: c0a80059
TIMEOUT
R_PORT
%MW1508: fa0
DATA
%VB3400: 56
LEN
%MW1512: 3
网络调试助手
网络设置
(1) 协议类型
UDP
(2) 本地主机地址
192.168.0.89
(3) 本地主机端口
4000
关闭
接收设置
ASCII
HEX
按日志模式显示
接收区自动换行
接收数据不显示
数据日志
[2022-05-10 11:42:15.468]# RECV HEX FROM 192.168.0.252 :3000>
66 66 66 66 66 66 66 66 66 66 00
[2022-05-10 11:44:35.075]# SEND HEX TO 192.168.0.252 :3000>
56 34 12
数据接收

IL

```
(* Network 0 *)
(*LINE 参数　　TRUE 表示网线插着，　FALSE 表示网线拔下了 CLIENT 参数 DWORD，每个 bit 表示一个 TCP client 的状态 SERVER 参数　DWORD，每个 bit 表示一个 TCP server 的状态 UDPPEER 参数　DWORD，每个 bit 表示一个 UDP peer 的状态*)
LD　　%SM0.0
ETH_STATUS %M333.7, %MD334, %MD338, %MD342
(* Network 1 *)
(*本例子是输入参数为常量方式，PEER, L_PORT，这几个输入参数，可以同时为常量，或者同时为变量开启 UDP*)
LD　　%SM0.0
UDP_PEER %M4.0, 0, W#3000, %MB1018, %M5.0
UDP_PEER %M4.0, 1, W#3001, %MB1019, %M5.1
(* Network 2 *)
(*UPD_XMT 用的发送字串*)
LD　　%SM0.1
FILL　　B#16#66, %VB2500, B#10
FILL　　B#16#77, %VB2600, B#11
(* Network 3 *)
(*本例子是输入参数为常量方式，PEER, R_IP, R_PORT, LEN，这几个输入参数，可以同时为常量，或者同时为变量发送数据*)
LD　　%SM0.0
UDP_XMT %M0.3, 0, 192.168.0.89, W#4000, %VB2500, 11, %MB1020
UDP_XMT %M0.3, 1, 192.168.0.89, W#4001, %VB2600, 12, %MB1021
(* Network 4 *)
(*本例子是输入参数为常量方式，PEER, MAXLEN, TIMEOUT 这几个输入参数，可以同时为常量，或者同时为变量*)
LD　　%SM0.0
UDP_RCV %M7.0, 0, 92,
0, %MB1022, %M7.7, %MD1500, %MW1508, %VB3400, %MW1512
UDP_RCV %M7.0, 1, 96,
```

IL

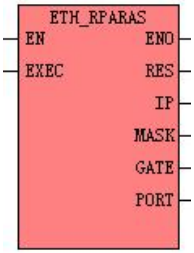
```
(* Network 5 *)
(*观察 UDP_RCV 指令从 peer 0 接收的报文次数*)
LD      %M7.7
R_TRIG
INC     %VD4016
(* Network 6 *)
(*观察 UDP_RCV 指令从 peer 1 接收的报文次数*)
LD      %M7.6
R_TRIG
INC     %VD4020
(* Network 7 *)
(*本例子是输入参数为常量方式*)
LD      %SM0.0
UDP_XMTRCV %M55.0, 0, 192.168.0.89, W#4000, %VB5000, 45, 99, 11223,
0, %VB6000, %V6001.0, %VD6002, %VW6006, %VB5100, %VW6008
UDP_XMTRCV %M55.0, 1, 192.168.0.89, W#4001, %VB5200, 56, 99, 11224,
1, %VB6010, %V6011.0, %VD6012, %VW6016, %VB5300, %VW6018
(* Network 8 *)
(*UDP_XMTRCV 用的发出字串*)
LD      %SM0.1
FILL    B#16#33, %VB5000, B#43
FILL    B#16#44, %VB5200, B#53
(* Network 9 *)
(*观察 UDP_XMTRCV  从 peer 0 接收的报文次数*)
LD      %V6001.0
R_TRIG
INC     %VD4008
(* Network 10 *)
(*观察 UDP_XMTRCV  从 peer 1 接收的报文次数*)
LD      %V6011.0
R_TRIG
INC     %VD4012
```

10.3.3.6 本机以太网参数读写指令

10.3.3.6.1 ETH_RPARAS 指令

此指令为读取本机以太网参数指令。

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	ETH_RPARAS			☑ K6
IL	ETH_RPARAS	ETH_RPARAS *EXEC, RES, IP, MASK, GATE, PORT*	U	

参数	输入/输出	数据类型	允许使用的内存区
EXEC	输入	BOOL	I、Q、V、M、L、SM、RS、SR
RES	输出	BYTE	Q、M、V、L、SM
IP	输出	DWORD	Q、M、V、L、SM
MASK	输出	DWORD	Q、M、V、L、SM
GATE	输出	DWORD	Q、M、V、L、SM
PORT	输出	WORD	Q、M、V、L、SM

➢ 参数说明

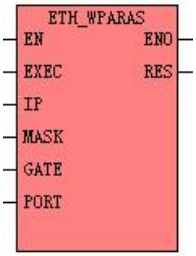
参数	描　述
EXEC	若检测到 *EXEC* 的上升沿跳变，则该指令被触发执行。

RES	执行开始时，此参数输出 0，执行结束时，bit7 == TRUE,bit0-bit6 合起来的 byte 数值，表示错误码 0 读取成功
IP	本机以太网的 IP 地址。
MASK	本机以太网的子网掩码。
GATE	本机以太网的网关。
PORT	本机以太网的端口号。

10.3.3.6.2 ETH_WPARAS 指令

此指令为修改本机以太网参数指令。

- 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	ETH_WPARAS			☑ K6
IL	ETH_WPARAS	ETH_WPARAS *EXEC, IP, MASK, GATE, PORT, RES*	U	

参数	输入/输出	数据类型	允许使用的内存区
EXEC	输入	BOOL	I、Q、V、M、L、SM、RS、SR
IP	输入	DWORD	M、V、L、SM、常量
MASK	输入	DWORD	M、V、L、SM、常量
GATE	输入	DWORD	M、V、L、SM、常量
PORT	输入	WORD	M、V、L、SM、常量
RES	输出	BYTE	M、V、L、SM、常量

注意：参数 *IP, MASK, GATE, PORT* 必须同时为常量类型或同时为内存类型。

- 参数说明

参数	描　述
EXEC	若检测到 *EXEC* 的上升沿跳变，则该指令被触发执行。
IP	需要修改的本机以太网的 IP 地址，不支持全 0。
MASK	需要修改的本机以太网的子网掩码。
GATE	需要修改的本机以太网的网关。
PORT	需要修改的本机以太网的端口号，不支持全 0。
RES	执行开始时，此参数输出 0，执行结束时，bit7 == TRUE,bit0-bit6 合起来的 byte 数值，表示错误码 0　写入成功 1　参数错误，有为 0 的输入参数，不支持。 2　将新参数写入永久存储时发生错误。

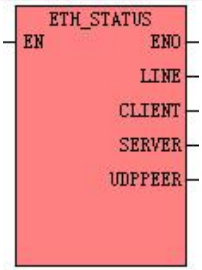
注意：写入新参数后， 整个以太网硬件将重启。

10.3.3.7 本机以太网状态监控指令和复位指令

10.3.3.7.1 ETH_STATUS 指令

此指令为监控本机以太网状态指令。

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	ETH_STATUS			☑ K6
IL	ETH_ STATUS	ETH_ STATUS *LINE, CLIENT, SERVER, UDPPEER*	U	

参数	输入/输出	数据类型	允许使用的内存区
EXEC	输入	BOOL	I、Q、V、M、L、SM、RS、SR
LINE	输出	DWORD	M、V、L、SM
CLIENT	输出	DWORD	M、V、L、SM
SERVER	输出	DWORD	M、V、L、SM
UDPPEER	输出	DWORD	M、V、L、SM

➢ 参数说明

参数	描　述
EXEC	若检测到 *EXEC* 的上升沿跳变，则该指令被触发执行。
LINE	TRUE 表示网线插着，FALSE 表示网线拔下了。
CLIENT	每个 bit 表示一个 TCP Client 的状态， 比如 bit0==TRUE,表示 CLIENT 0 处于连接

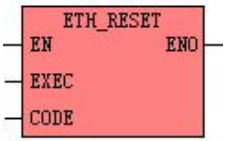
	好状态，bit0==FALSE,表示 CLIENT 0 处于不能收发数据状态。
SERVER	每个 bit 表示一个 TCP Server 的状态，比如 bit9==TRUE，表示 SERVER 9 处于连接好状态，bit9==FALSE,表示 SERVER 9 处于不能收发数据状态。
UDPPEER	每个 bit 表示一个 UDP peer 的状态， 比如 bit11==TRUE,表示 peer 11 已经创建，bit11==FALSE,表示 peer 11 未使用。

注意：LINE 为实时显示的状态，CLIENT、SERVER、UDPPEER 这三个参数都不是完全实时的，而是由 TCP 协议栈决定的。

10.3.3.7.2 ETH_RESET 指令

此指令为复位以太网指令。

- 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	ETH_RESET			☑ K6
IL	ETH_ RESET	ETH_ RESET *EXEC, CODE*	U	

参数	输入/输出	数据类型	允许使用的内存区
EXEC	输入	BOOL	I、Q、V、M、L、SM、RS、SR
CODE	输入	BYTE	常量

- 参数说明

参数	描　述

EXEC	若检测到 *EXEC* 的上升沿跳变，则该指令被触发执行，每次上升沿执行一次重置整个以太网功能，所有的连接都会失效，需要重连。
CODE	只能是 b#1，或 b#2 b#1 时，不清除 MCU 的以太网部分的硬件收发缓冲。 b#2 时，清除 MCU 的以太网部分的硬件收发缓冲。

注意：

1、这个指令设计用于无人值守的情形，一般不建议使用。

2、当设备仅仅用于通信，且无人值守时，如果 PLC 自己能判断出来必要的通信不正常了，那么可以尝试使用此指令重启以太网，以排除设备常年运行时的假死。

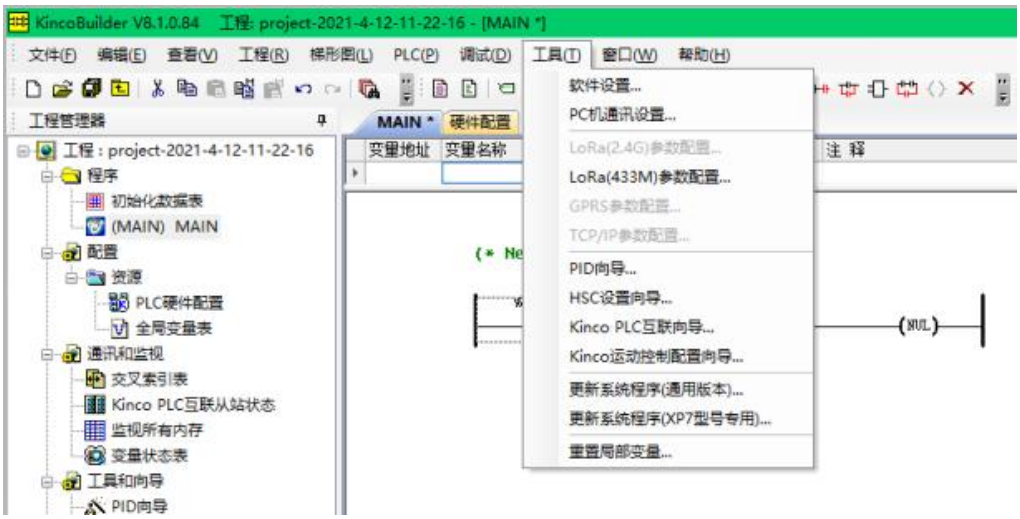
10.4 LPWAN 无线通信口的使用

支持 LoRa 无线通信口的 CPU 目前有三款，两个标准型 CPU（KW103、KW203），一个简易型 CPU（KW213），它们支持的工作频段如下表：

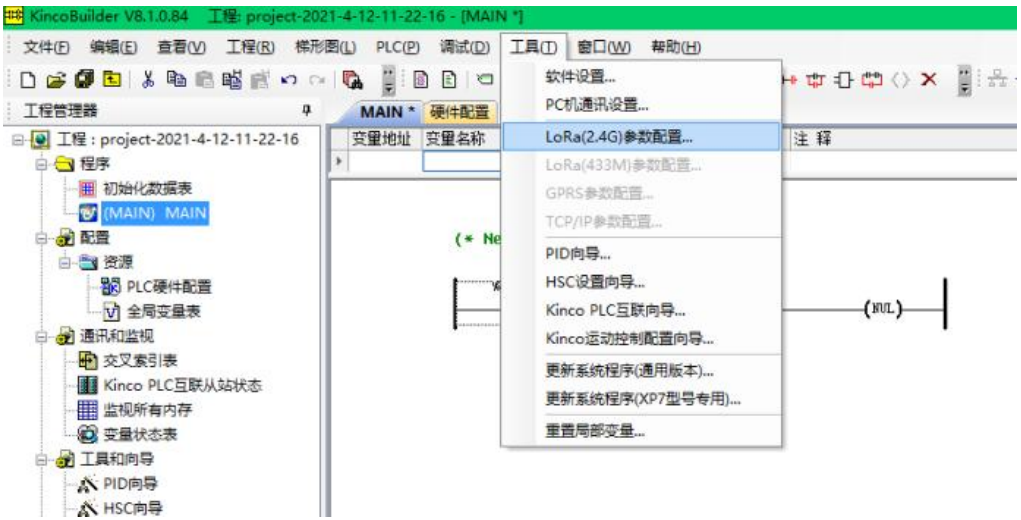
型号	工作频段
KW103	410～493MHz
KW203	2400～2500MHz
KW213	

KW103 和 KW203 除了工作频段不同外，支持的某些指令也不完全相同，自由通信指令 KW103 支持最早的 XMT、RCV 指令，而 KW203 则支持后续增加的 COM_XMT、COM_RCV 指令，还有 LORA 口专用的读写参数指令只有 KW203 可以使用，后续章节有相关介绍。

关于 KW103 的通信参数设定，用户执行【工具】->【LoRa（433M）参数设置】菜单命令，如下图，在弹出的窗口中可以对频率、空中传输速度等参数进行读写操作。



关于 KW203 的通信参数设定，用户执行【工具】->【LoRa（2.4G）参数设置】菜单命令，如下图，在弹出的窗口中可以对 LoRa 通信参数进行读写操作。



10.4.1 概述

LoRa 是 LPWAN（Low Power Wide Area Network，低功耗广域网网络）通信技术之一，是一种基于扩频技术、超远距离的窄带无线通信方案，具有覆盖范围广、抗干扰能力强、发射功率低等优点。LoRa 工作在免授权频段，且部署方便、灵活，用户可以自由组建自己的私有网络。

10.4.2 常用的无线通信术语

1） dBm（分贝毫瓦）

dBm（分贝毫瓦）是一个表征功率绝对值的值，是以 1mW 功率为基准的一个比值。它的计算公式是 10*lg(P)，其中 P 是功率（单位 mW）。对于 dBm，有几个比较简单直观的经验值：

- 1mW 的功率等于 0dBm；1W 的功率等于 30dBm。
- 增加 3dBm，表示功率乘以 2；减少 3dBm，表示功率除以 2。
- 增加 10dBm，表示功率乘以 10；减少 10dBm，表示功率除以 10。

2） dB（分贝）

dB 是一个表征相对值的值。

当考虑 A 的功率相比于 B 功率大或者小多少个 dB 时，按下面计算公式：10*lg(A/B)。

例如，甲功率比乙功率大一倍，那么 10*lg（甲功率/乙功率）=10*lg2=3dB。

3） 信道带宽（BW）

信道带宽是允许通过该信道的信号下限频率和上限频率之差，可以理解为一个频率通带。

LoRa 的带宽为双边带宽，假设一个 LoRa 信道的中心频率为 f，带宽为 BW，则该信道的实际频率范围为$[f-\frac{BW}{2},f+\frac{BW}{2}]$。

4） 编码率（CR）

LoRa 采用循环纠错编码进行前向检错与纠错，在干扰严重的情况下能有效提高链路的可靠性，但代价就是在编码时额外增加了一些冗余的信息。

编码率就是指通信数据流中有用（非冗余）部分的比例。如果编码率是 k/n，则对每 k 位有用信息，编码器总共产生 n 位的数据，其中 n-k 是多余的。

5） 接收灵敏度（Receiver Sensitivity）

接收灵敏度是指接收机能够正确接收到的信号的最小功率。当信号能量小于标称的接收灵敏度时，接收机将不会接收。

接收灵敏度是检验接收机接收微弱信号的能力，其数值越小，就意味着接收能力越强。例如，A 的接收灵敏度是 -85dBm，B 的是 -140dBm，那么显然 B 的接收能力要远远强于 A，使用 B 的无线系统也会比使用 A 的覆盖范围更广。

6） 链路预算（Link Budget）

在保持一定通信质量的前提下，通信链路所允许的最大传播损耗。

链路预算是评估无线通信系统覆盖能力的主要方法。在给定的环境中，链路预算值越大，说明无线通信的覆盖范围越大。

10.4.3 LoRa 通信口的参数设置

KW1 提供的 LoRa 通信口与 KW2 的工作频段不同，适用的场景也不尽相同，但是两者的使用方法基

本相同，因此下文将以 KW2 为例进行说明。

LoRa 通信口支持编程协议、Kinco PLC 互联协议、Modbus RTU 协议（主站及从站）以及自由通信功能。在编程软件中提供了各种使用向导、参数配置、网络状态监控等工具以方便用户的应用，另外也提供了各种通信指令，用户可以在程序中调用这些指令来实现对 LoRa 通信的在线操作。下文将详细描述这些功能。

10.4.3.1 配置 LoRa 参数

KW 提供了两种 LoRa 参数配置方法：

- 使用 KincoBuilder 软件提供的参数配置工具；
- 在用户程序中使用参数读写指令。

10.4.3.1.1 使用参数配置工具

在 KincoBuilder 软件中，执行【工具】->【LoRa(2.4GHz）参数配置…】菜单命令，将会进入配置工具窗口，用户可以在这里对 LoRa 接口进行配置,另外，也可以直观地看到采用不同通信参数时对于无线通信性能的预估结果，从而有助于选择最适合自己应用的参数组合。

➢ **操作目标**

选择要修改的接口或者设备。

不同型号的 KW 模块提供 1 或 2 个 LoRa 接口，并分配了不同的编号。若模块只有 1 个 LoRa 口，则这个口的编号为 1。若模块有 2 个 LoRa 口，则在模块丝印上标注了每个口的编号。

- 【本机口 1】：表示将要操作电脑所连模块的 LoRa 口 1 的参数。
- 【本机口 2】：表示将要操作电脑所连模块的 LoRa 口 2 的参数。
- 【口 1 网络所有节点】：此选项仅支持【写入】操作，表示将要修改电脑所连模块的 LoRa 口 1 所在无线网络中所有能够正常通信的模块 LoRa 接口参数。
- 【口 2 网络所有节点】：此选项仅支持【写入】操作，表示将要修改电脑所连模块的 LoRa 口 2 所在无线网络中所有能够正常通信的模块 LoRa 接口参数。

当选择修改“网络所有节点”的参数时，请注意：

1） 修改过程将持续 3 秒钟左右。

2） 修改参数可能会影响正常的数据通信，请在确保安全的前提下进行修改！

3） 若修改网络中所有节点的参数，则本网络中不允许有其它节点发送数据，否则可能导致修改失败！而且由于 LoRa 是半双工通信方式,配置软件无法判断是否有修改失败的节点！

建议用户通过网络中的主站来执行此功能以避免操作不成功。

➢ **通信参数**

- 【空中传输速率】：是指无线（在空气中）的数据传输速率，可以理解为与有线通信的波特率类似。空中速率高，则数据传输速度快，但通常情况下传输距离会变近。

  LoRa 提供了多种带宽，每个带宽下又提供了多种空中传输速率。在软件中，用户需首先选择一个带宽，然后软件将自动列出该组中所有的空中速率，用户即可进行选择。

  出厂默认值是 25.38Kbps。
- 【编码率】：LoRa 的编码率。所选值越大，则编码率越低。出厂默认值是 1。

  编码率越低，则通信的可靠性越高，但有效的数据传输速率相应就越低。在干扰严重的场合，建议降低编码率（即选择更大的数值）。
- 【频率信道】：LoRa 工作的中心频率。出厂默认值为 2450MHz。

2.4GHz 频段是全球免费频段，WiFi、蓝牙、Zigbee 等设备均使用该频段。LoRa 对于这些通信具有很强的免疫力，但极端情况下也可能会受到这些通信设备的干扰，若发生这种情况，建议用户调整 KW 的频率信道以避开现场范围内其它通信设备所用的频带。

- 【发射功率】：LoRa 的发射功率。出厂默认值是 16（-2dBm）。
- 【分包时间】：是指两帧报文之间的最大间隔时间。若 KW 模块在这个时间内没有再次接收到数据，则会将已经接收到的数据作为一帧完整报文进行处理。出厂默认值是 1ms。

  分包时间越长，则能够达到的通信频度就越低。

  在实际应用中，可能会因通信距离远、阻挡物多、干扰严重等原因引起无线信号传输不太稳定，从而导致 KW 对报文的分割错误，若发生这种情况，建议用户适当增大分包时间。
- 【使能芯片内置 CRC】：LoRa 芯片内置 CRC 功能，若启用 CRC，则发送端 LoRa 每次发送之前均会对数据进行 CRC 校验，并将校验码附加到报文中发送出去，当接收端 LoRa 收到报文后会首先进行 CRC 校验，若校验正确则存储报文，否则会将报文丢弃。

  KW 提供的 Kinco 互联协议、Modbus 通信协议中均自带 CRC 校验功能，因此出厂默认不使能芯片内置的 CRC。

➢ **性能预估**

根据用户所选的通信参数，本软件将自动预估计算可能达到的无线通信性能，并将结果显示在这里以供用户参考。

- 【链路预算】：若使用当前通信参数，LoRa 理论上能到达的链路预算。预算值越大，则表示无线覆盖范围越广。
- 【接收灵敏度】：若使用当前通信参数，LoRa 理论上的接收灵敏度。灵敏度值越小，则表示接受能力越强，因此发射端与接收端之间的距离也可以更远。
- 【有效空中速率】：若使用当前通信参数，LoRa 理论上传输用户有效数据的速率。

  有效速率跟空中传输速率、编码率有关：空中传输速率越高，则有效速率自然越高；编码率越低，意味着通信数据中的冗余信息越多，因此有效速率就越低。

➢ **时间预估**

选定通信参数后，用户可以在此处输入【传输数据长度】并单击【时间预估】按钮，本软件就会

自动计算这些数据从发送端到接收端所需要的【空中传输时间】。

10.4.3.1.2 使用参数读写指令

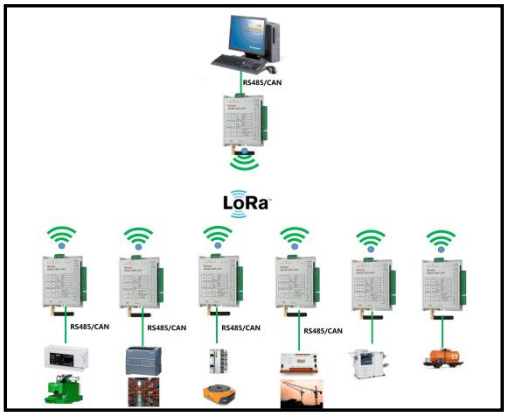
KW 提供了 LORA_RPARAS（读取 LoRa 参数）和 LORA_WPARAS（修改 LoRa 参数）指令。在用户工程中，可以调用这些指令实现对 LoRa 参数的在线读取和修改。指令中的参数值均与 LoRa 参数配置工具中的各参数选项值相对应。

指令的详细描述请参阅 10.4.5 LoRa 功能相关指令。

10.4.4 使用 LoRa 通信

在使用 LoRa 通信之前，用户需要先配置好 LoRa 通信口的参数。在同一个 LoRa 通信网络中，所有节点的下述参数必须一致：频率、空中传输速率、编码率、是否使能芯片内置 CRC。

10.4.4.1 组建 LoRa 通信网络



LoRa 是半双工通信方式，接收、发送不能同时进行。在一个 LoRa 网络中，任一时刻只能有一个节点处于发送状态。因此建议用户在实际应用中利用 LoRa 组建一种主从模式、星型拓扑的无线通信网络：在网络中有且只有一个主站节点，其余节点全部作为从站；主站负责调度、管理网络，并且轮询各个从站；从站只有收到主站的请求后才能进行回应。如图所示。

在实际应用中，用户可以通过设置不同的**通信频率信道或者空中传输速率**将同一区域内的模块划分成**不同的无线网络**。例如，假如一个车间内有 100 台设备，则可以将其中 50 台设备的通信频率设置为 2410MHz，另外 50 台的频率设置为 2430MHz，那么这些设备就可以组建 2 个相互独立的无线通信网络。相邻频段最好不要设置成规则偶数数列，而应无规律奇数相差为佳。

10.4.4.2 使用 LoRa 通信

LoRa 通信口支持编程协议（主从模式）、Kinco PLC 互联协议（主从模式）以及自由通信功能。

若用户程序中没有使用通信功能，则 KW 上电后默认会支持编程协议，同时也将自动作为 Kinco PLC 互联从站和 Modbus RTU 从站。

10.4.4.2.1 编程协议

LoRa 通信口支持 KincoBuilder 软件中的编程协议命令（位于【PLC】和【调试】菜单中），包括上载、下载、在线监测、强制等，同时也支持修改网络中所有节点的 LoRa 通信口参数。

Kinco 提供专用的调试模块以实现上述功能。使用方法如下：

1) 在电脑的 USB 或者 RS232 口连接一个 Kinco 专用调试模块。
2) 打开【PC 机通信设置】窗口，选择待操作的【目标 PLC】的站号以及电脑所用的【COM 口】。专用调试模块 RS232 口的通信参数为：波特率 115200，无校验，8 数据位，1 停止位。用户若使用 RS232 口，则需要在中将电脑 COM 口参数设置成上述参数。若使用 USB 口则无需设置。
3) 按前文所述方法配置好调试模块的 LoRa 口通信参数，确保与目标 PLC 能够正常通信。用户配置参数时无需关心【目标 PLC】站号，选择任何站号都不会影响对本模块参数的配置。
4) 最后，在 KincoBuilder 软件中执行所需的命令即可。

使用无线编程功能时需注意：目标 PLC 的 LoRa 通信口不能使用自由通信功能；所在的 LoRa 网络中不允许有调试模块之外的其它节点主动发送报文，即必须停止网络中主站的通信！

10.4.4.2.2 Kinco PLC 互联协议及向导工具

Kinco PLC 互联协议是专为 Kinco PLC 之间高效联网通信而设计的一种通信协议。该协议采用主从模式，在网络中必须且只允许存在一个主站，其它的节点都必须作为从站。主站按用户设定的周期来定时访问从站，而从站只能在收到主站的请求后进行回应。另外，主站支持发送“写数据”的广播报文，所有从站收到广播报文后都会进行处理，但从站不需对广播报文进行回应。

若没有启动其它通信协议功能，则 PLC 上电后会默认作为 Kinco 互联协议从站，用户无需另外编程。对于主站来说，Kinco 互联协议功能具有最高优先级，PLC 一旦作为互联协议主站运行，则该通信口将不再处理其它通信协议的报文

- **使用方法**

用户按照如下步骤即可启用 Kinco PLC 互联协议功能。

1) 按前文所述方法配置好各节点的 LoRa 口通信参数，确保能够正常通信。

2) 每个从站都需要在【硬件配置】->【通信设置】中为 LoRa 通信口指定一个站号，有效的站号范围是 1～63。在同一网络中，每个从站的站号都必须是唯一的，不允许出现重复的站号。

3) 在主站的用户工程中，使用 Kinco PLC 互联向导对各个从站进行配置，配置完成后将工程下载到一个 PLC 中，则该 PLC 将作为主站运行，启动并管理整个网络的通信。

- **Kinco PLC 互联向导**

在 KincoBuilder 软件中，用户双击【工程管理器】中的【Kinco PLC 互联向导】节点即可进入向导窗口，在这里可以设置网络中所有节点的通信数据、通信参数。

- 所有从站列表

编辑本网络中将要连接的从站，并以树状列表显示。

用户必须将网络中的所有从站都添加进来，主站将只会访问列表中存在的从站。

单击某个从站节点，窗口右侧就会显示本从站的通信数据和通信参数。**每个从站至少需要配置一个数据报文，没有任何数据报文的从站也将被忽略掉**。当修改了【从站地址】后，树节点的标题也将会根据新地址自动改变。

若从站的配置有问题，则在树节点标题前会显示一个红色的小“？”进行警告，通常的原因是多个从站读取的数据在主站中写入的地址区域重叠了。

- 通信设置

主站按用户设定的周期来定时访问从站，每个从站的访问周期都可以单独设置。

每个从站的各数据传输报文可分别设置为【定时触发】和【边沿触发】两种触发形式。当某个从

站的定时时间到了之后，主站会根据用户配置的触发形式来决定是否与该从站之间进行数据传输：对于定时触发的数据，主站会立即启动一次传输；对于边沿触发的数据，主站会先判断触发位，若触发位的值为“0”，则主站不会启动传输，若触发位的值为“1”，则主站会立即启动一次传输，完成后主站自动将触发位复位为“0”。

【从站地址】：当前从站的地址。

【访问周期】：主站定时访问本从站的定时时间，单位：ms。KW 允许的最长定时时间约为 49 天。用户根据实际需求来合理设置访问周期。假如需要快速刷新本从站数据，那么访问周期就可以设置得尽量短，但要注意的是，若访问周期比传输数据所需时间还短，那么实际的访问周期就是本站数据传输所需的时间。假如不需要频繁刷新数据，那么用户可以将访问周期尽量设置得长一些，降低本从站的通信频度，减轻网络压力。

【超时时间】：主站发出请求后，在这个时间内必须收到从站的回应报文，否则主站就会记录一次本从站的通信超时错误。

【重试次数】：主站发出请求后，若发生通信错误（超时无回应，或收到的报文有错误），则主站会再次重新尝试请求，直到通信成功或者达到用户配置的重试次数为止。

【触发读的上升沿】：指定一个主站中 M 区的变量，用于触发主站读取本从站中配置为“边沿触发”方式的数据。读取完成后主站会自动将该位清“0”，因此在主站用户程序中，在需要传输数据之前的时刻将该位置为“1”即可，避免一直将其保持置“1”。

【触发写的上升沿】：指定一个主站中 M 区的变量，用于触发主站写入本从站中配置为“边沿触发”方式的数据。写入完成后主站会自动将该位清“0”，因此在主站用户程序中，在需要传输数据之前的时刻将该位置为“1”即可，避免一直将其保持置“1”。

- 传输数据设置

用户在这里配置本从站需要传输的数据，供主站读取、写入的数据区域最大可以分别设置 6 组，每组数据可以分别设置触发方式（定时触发或者边沿触发）。

为了提高通信效率，主站会将用户配置的数据区域进行组合，其中“定时触发”的全部数据组合成一个报文，“边沿触发”的全部数据组合成另外一个报文。**LoRa 通信报文最大允许 246 字节长度！**用户在配置通信数据时，软件会自动显示组合报文长度，注意每种方式都不要超过 246 字节。一个报

文中包含了用户的有效数据、软件额外增加的协议数据等等，因此报文中**用户有效数据的最大长度约为 226 字节**。

- 广播报文

站号 64 被固定用于发送广播报文，用于同时修改网络中所有从站内的数据。

64 号从站在网络中实际上并不存在。用户在 64 号从站中配置的数据，主站将以广播报文形式发送出去，网络中所有从站接收到报文后都会进行处理，且无需进行回应。

➢ Kinco PLC 互联从站状态

“Kinco PLC 互联从站状态”用于监视当前互联协议网络中所有从站的运行信息。

在 KincoBuilder 软件的工程管理器中，用户双击【Kinco PLC 互联从站状态】节点即可打开。

【从站地址】：本行信息所属的从站站号。

【运行状态】：本从站通信的状态，包括：正常、离线、未配置成功。

在网络启动时，主站会向各从站发送报文配置通信数据区域等，配置成功后才会继续与该从站进行正常的数据交互。若配置失败，则主站记录该从站“未配置成功”，并且不断尝试重新配置该从站，直到配置成功为止。在正常交互数据时，若本从站响应错误，则主站会记录一次“离线”错误，下一次正常通信后会重新记录为“正常”。

【实际访问周期】：主站对本从站轮询访问的实际周期时间，单位 ms。

【信号强度】：主站每次接收到本从站的报文时，都会实际检测并记录无线信号的强度。信号强度值为负数，其数值越接近于 0 说明信号强度越高。在实际应用中，若用户发现某个从站通信错误率比较高，且信号强度相对比较低，则需要考虑改进该从站的安装，比如调整天线位置、换用更高增益的天线等等。

【通信正确率】：从主站上电运行开始，主站收到本从站的正确回应报文的比例。

$$通信正确率 = \frac{从站正确回应报文的总数量}{主站请求报文的总数量} \times 100\%$$

无线通信不能保证 100%的正确率，我们认为正确率在 95%以上就属于通信良好。

10.4.4.2.3 自由通信功能

所谓的自由通信，是指通信过程及通信数据完全由用户程序进行控制。用户可以使用自由通信方式来编写各种自定义的通信协议与其它设备进行通信。

当执行了用户程序中的自由通信指令时，自由通信方式就被激活，相应的通信口就完全被自由通信占用。当自由通信指令完成后，CPU 又自动将通信口切换到默认的协议。若 PLC 处于 STOP 状态，则自由通信程序被禁止，PLC 将恢复默认的编程协议和 Modbus RTU 从站功能。

指令的使用方法请参见 10.1 自由通信。

10.4.5 LoRa 功能相关指令

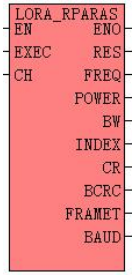
本节描述的指令均位于【指令集】->【通信指令】组中。

在 IL 语言编写的程序中，所有 LoRa 指令的功能均与 LD 程序中的一致，并且都遵循 IL 程序的执行原则：若 CR 值为 1，则该指令被扫描执行，并且执行结果不影响 CR 值。

因此，下文将仅描述 LD 格式的指令，对于 IL 格式不再赘述。

10.4.5.1 LoRa 口专用指令

10.4.5.1.1 LORA_RPARAS（读 LoRa 参数）

	名称	指令格式	适用于
LD	LORA_RPARAS		☑ KW203 ☑ KW213

参数	输入/输出	数据类型	允许使用的内存区
EXEC	输入	BOOL	I、Q、V、M、L、SM
CH	输入	INT	V、M、L、常量
RES	输出	BYTE	V、M、L
FREQ	输出	INT	V、M、L
POWER	输出	INT	V、M、L
BW	输出	INT	V、M、L
INDEX	输出	INT	V、M、L
CR	输出	INT	V、M、L
BCRC	输出	BOOL	V、M、L
FRAMET	输出	INT	V、M、L
BAUD	输出	REAL	V、L

该指令读取的 LoRa 参数值均与 Kincobuilder 软件的 LoRa 参数配置工具中各参数选项相对应。

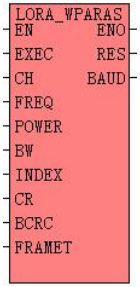
参数	描　述
EXEC	若检测到 *EXEC* 的上升沿跳变，则该指令被触发执行。
CH	待读取参数的 LoRa 口编号。0 表示本机口 1，1 表示使用本机口 2。

RES	最新一次的执行结果。其组成如下： 第 7 位～指令状态。若指令正在执行则该位被置 0，当指令完成时该位立即被置 1。 第 0 位～非法的 LoRa 口。若 CH 指定的 LoRa 口编号是非法值，则该位被置 1。 其它位～保留
FREQ	读取的频率信道选项值。
POWER	读取的发射功率选项值。
BW	读取的带宽选项值。该值对应着参数配置工具中的 BW 选项： 0 表示 BW203，1 表示 BW406，2 表示 BW812，3 表示 BW1625，4 表示 FLRC。 0-3 表示工作在 LoRa 模式下，4 表示工作在 FLRC 模式下。
INDEX	读取的空中传输速率选项值。该值对应着参数配置工具中当前 BW 选项下的“空中传输速率”列表中的选项：0 表示当前列表中的第 1 项，1 表示第 2 项，以此类推。
CR	读取的编码率值。该值对应着参数配置工具中“编码率”列表中的选项。 LoRa 模式：有效数值范围为 1-4。1 表示列表中第 1 项，依次类推。 FLRC 模式：有效数值范围为 0-2。0 表示列表中第 1 项，依次类推。
BCRC	读取的“使能芯片内置 CRC”选项值。
FRAMET	读取的分包时间选项值。
BAUD	根据读取的 *BW* 和 *INDEX* 选项值所计算得到的空中传输速率。

当 *EN* 值为 1 时，若检测到 *EXEC* 输入端的上升沿，则该指令被触发执行一次。当指令执行时，*RES* 被置为 0。当指令完成后（无论成功或者失败），*RES* 的第 7 位都会立即被置为 1。若参数读取成功，KW 会根据读取的 LoRa 参数值来更新本指令相应的输出参数，否则相关输出参数值保持不变。

用户可以在程序中根据 *RES* 第 7 位的上升沿来判断指令是否完成，然后根据其它位表示的错误值来判断是否读取成功。

10.4.5.1.2 LORA_WPARAS（修改 LoRa 参数）

	名称	指令格式	适用于
LD	LORA_WPARAS		☑ KW203 ☑ KW213

参数	输入/输出	数据类型	允许使用的内存区
EXEC	输入	BOOL	I、Q、V、M、L、SM
CH	输入	INT	V、M、L、常量
FREQ	输入	INT	V、M、L
POWER	输入	INT	V、M、L
BW	输入	INT	V、M、L
INDEX	输入	INT	V、M、L
CR	输入	INT	V、M、L
BCRC	输入	BOOL	V、M、L
FRAMET	输入	INT	V、M、L
RES	输出	BYTE	V、M、L
BAUD	输出	REAL	V、L

该指令中 LoRa 参数输入值均与 Kincobuilder 软件的 LoRa 参数配置工具中各参数选项相对应。

参数	描　述
EXEC	若检测到 *EXEC* 的上升沿跳变，则该指令被触发执行。
CH	要修改的 LoRa 口编号。输入值含义如下： 0 ～ 本机口 1； 2 ～ 口 1 网络所有节点； 1 ～ 本机口 2； 3 ～ 口 2 网络所有节点。

FREQ	新的频率信道选项值。
POWER	新的发射功率选项值。
BW	新的带宽选项值。该值对应着参数配置工具中的 BW 选项： 0 表示 BW203，1 表示 BW406，2 表示 BW812，3 表示 BW1625，4 表示 FLRC。 0-3 表示工作在 LoRa 模式下，4 表示工作在 FLRC 模式下。
INDEX	新的空中传输速率选项值。该值对应着参数配置工具中当前 BW 选项下的“空中传输速率”列表中的选项：0 表示当前列表中的第 1 项，1 表示第 2 项，以此类推。
CR	新的编码率值。该值对应着参数配置工具中“编码率”列表中的选项。 LoRa 模式：有效数值范围为 1-4。1 表示列表中第 1 项，依次类推。 FLRC 模式：有效数值范围为 0-2。0 表示列表中第 1 项，依次类推。
BCRC	新的“使能芯片内置 CRC”选项值。
FRAMET	新的分包时间选项值。
RES	最新一次的执行结果。其组成如下： 第 7 位～指令状态。若指令正在执行则该位被置 0，当指令完成时该位立即被置 1。 第 1 位～修改结果。若参数修改失败，则该位被置 1。 第 0 位～非法的 LoRa 口。若 CH 指定的 LoRa 口编号是非法值，则该位被置 1。 其它位～保留
BAUD	根据输入的 *BW* 和 *INDEX* 选项值所计算得到的空中传输速率。

当 *EN* 值为 1 时，若检测到 *EXEC* 输入端的上升沿，则该指令被触发执行一次，用新的参数值去更新 Lora 口 *CH* 的各个通信参数。

当指令执行时，*RES* 被置为 0。当指令完成后（无论成功或者失败），*RES* 的第 7 位都会立即被置为 1。用户可以在程序中根据 *RES* 第 7 位的上升沿来判断指令是否完成，然后根据其它位表示的错误值来判断是否修改成功。

用户使用本指令时需要注意如下几点：

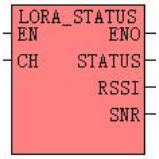
1）在同一时刻，只允许有一个 LORA_WPARAS 指令被执行！

2）修改参数可能会影响当前正常的数据通信，请在确保安全的前提下进行修改！

3）若修改网络中所有节点的参数，则建议用户通过网络中的主站来执行此指令，且需保证各节点都

能够正常通信！另外，由于 LoRa 是半双工通信方式，因此 KW 无法准确判断各节点的参数是否真正被修改成功！

10.4.5.1.3 LORA_STATUS（获取 LoRa 信号质量）

	名称	指令格式	适用于
LD	LORA_STATUS		☑ KW203

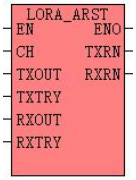
参数	输入/输出	数据类型	允许使用的内存区
CH	输入	INT	V、M、L、常量
STATUS	输出	WORD	V、M、L
RSSI	输出	INT	V、M、AQW、L
SNR	输出	INT	V、M、AQW、L

参数	描　述
CH	所用 LoRa 口的编号。0 表示本机口 1，1 表示使用本机口 2。
STATUS	LoRa 口的工作状态。该参数备用。
RSSI	接收信号的强度值。该值为负数，越接近于 0 说明信号强度越高。
SNR	接收信号的信噪比。信噪比越大，说明信号质量越好。

当 *EN* 值为 1 时，则该指令执行。该指令会获取最近一次接收数据时检测到的信号强度 *RSSI* 和信噪比 *SNR*。用户可以根据这些参数来判断无线通信的质量：RSSI 值越大表示接收到的信号强度越高，就意味着信号在传输过程中的衰减越小；信噪比值越大，说明信号质量越好，受到的干扰越小。

10.4.5.1.4 自动复位 LoRa 通信口

- LORA_ARST（发送和接收超时引起的自动复位）

	名称	指令格式	适用于
LD	LORA_ARST		☑ KW203 ☑ KW213

参数	输入/输出	数据类型	允许使用的内存区
CH	输入	INT	V、M、L、常量
TXOUT	输入	DWORD	V、M、L、常量
TXTRY	输入	INT	V、M、L、常量
RXOUT	输入	DWORD	V、M、L、常量
RXTRY	输入	INT	V、M、L、常量
TXRN	输出	WORD	V、M、L
SNR	输出	WORD	V、M、L

参数	描　述
CH	所用 LoRa 口的编号。0 表示本机口 1，1 表示使用本机口 2。
TXOUT	发送超时时间，单位 ms。
TXTRY	连续发送超时的允许次数。若连续发送超时的次数超过此值，LoRa 口会自动复位。
RXOUT	接收超时时间，单位 ms。
RXTRY	连续接收超时的允许次数。若连续接收超时的次数超过此值，LoRa 口会自动复位。
TXRN	本次上电后，由于发送超时失败而引起的 LoRa 口复位的次数。
RXRN	本次上电后，由于接收超时失败而引起的 LoRa 口复位的次数。

该指令用于指定 LoRa 通信口自动复位的条件。*EN*的上升沿触发执行一次本指令，指令执行后，PLC 会将输入参数指定的复位条件存储下来，在以后的 LoRa 通信过程中，PLC 会自动检测发送、接收的超时错误，若满足复位条件则 PLC 就会自动复位 LoRa 通信口。**因此，若不需要多次调整复位条件**

的话，本指令执行一次即可！

TXOUT、*TXTRY* 参数用于指定连续发送超时而引起复位的条件。在启动一次发送时，PLC 开始进行超时检测，若在 *TXOUT* 时间内没有检测到“发送成功”的信号，则认为是一次发送超时。如果连续出现发送超时，且超时次数达到了 *TXTRY* 值，则 PLC 会自动复位 LoRa 通信口,同时 *TXRN* 值加 1。

RXOUT、*RXTRY* 参数用于指定连续接收超时而引起复位的条件。在启动一次接收时，PLC 开始进行超时检测，若在 *RXOUT* 时间内接收到报文，则认为是一次接收超时。如果连续出现发接收超时，且超时次数达到了 *RXTRY* 值，则 PLC 会自动复位 LoRa 通信口,同时 *RXRN* 值加 1。

- **连续接收报文错误的自动复位**

SMW26 用于存放 LoRa 通信口（本机口 1）连续接收报文错误的最大允许次数。若值为 0，则不启用该功能。若其值大于 0，则 PLC 会自动检测报文错误（CRC 校验错误等）的次数，若连续出现接收报文错误且错误次数达到了 SMW26 的值，则 PLC 会自动复位 LoRa 口。

SMW28 用于存放由于连续接收报文错误而引起的 LoRa 口复位的次数。

本复位条件与 LORA_ARST 指令无关，可以单独使用。

- **定时自动复位**

SMW24 用于存放 LoRa 通信口（本机口 1）定时复位的周期。若值为 0，则不启用该功能。若其值大于 0，则 PLC 会自动启动一个定时器，当定时时间达到 SMW24 值后，LoRa 口就自动复位一次。

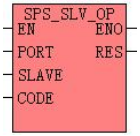
本复位条件与 LORA_ARST 指令无关，可以单独使用。

10.4.5.2 Kinco 互联协议专用指令

本组指令仅适用于“Kinco PLC 互联协议”。

在用户工程【PLC 硬件配置】中 CPU 的【通信设置】页面中，用户可以看到本机各个通信口的具体编号。

10.4.5.2.1 SPS_SLV_OP（主站暂停或者重启与从站的通信）

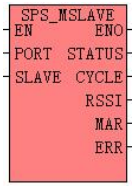
	名称	指令格式	适用于
LD	SPS_SLV_OP		☑ KW203 ☑ KW213

参数	输入/输出	数据类型	允许使用的内存区
PORT	输入	INT	常量
SLAVE	输入	INT	V、M、L、常量
CODE	输入	BYTE	V、M、L、常量
RES	输出	BYTE	V、M、L

参数	描　述
PORT	使用的通信口编号。0 表示 PORT0，1 表示 PORT1，2 表示 PORT2，依次类推。 若参数值指定了一个不存在的通信口，则为非法值，导致指令报错。
SLAVE	目标从站的站号。 该从站必须已经在网络中存在，即必须在【Kinco PLC 互联向导】中配置过。
CODE	操作选项，其值的含义如下： 1 ～ 主站启动与目标从站通信。 2 ～ 主站停止与目标从站通信。
RES	执行结果，其值的含义如下： 1 ～ 执行成功。2 ～ 所用的通信口不存在。 3 ～ 目标从站在网络中不存在。　4 ～ 非法的操作选项。

当 *EN* 值为 1 时，则该指令执行。对于在 *PORT* 通信口上运行的 Kinco 互联网络，用户可以在主站中调用该指令来暂停或者重启与目标从站 *SLAVE* 的通信。

10.4.5.2.2 SPS_MSLAVE（读取从站的通信情况）

	名称	指令格式	适用于
LD	SPS_MSLAVE		☑ KW203

参数	输入/输出	数据类型	允许使用的内存区
PORT	输入	INT	常量
SLAVE	输入	INT	V、M、L、常量
STATUS	输出	BYTE	V、M、L
CYCLE	输出	DWORD	V、M、L
RSSI	输出	INT	V、M、L
MAR	输出	WORD	V、M、L
ERR	输出	BYTE	V、M、L

参数	描　述
PORT	使用的通信口编号。0 表示 PORT0，1 表示 PORT1，2 表示 PORT2，依次类推。 若参数值指定了一个不存在的通信口，则为非法值，导致指令报错。
SLAVE	目标从站的站号。 该从站必须已经在网络中存在，即必须在【Kinco PLC 互联向导】中配置过。
STATUS	该从站的运行状态，其值的含义如下： 1 ～正常运行。 2 ～主站对该从站配置失败。 3 ～离线（即最近一次请求没有收到回应报文）。 4 ～主站调用 SPS_SLV_OP 指令暂停了与该从站之间的通信。

CYCLE	主站对该从站的真实轮询周期，单位：ms。
RSSI	主站最近一次接收到本从站的信号强度
MAR	本从站回应报文的正确率，该输出值是 16 位整数，含义：正确率×100。 正确率 = 从站正确回应报文总数量÷ 主站对该从站的请求报文总数量
ERR	本从站最近一次发生的通信错误。 0 ～无错误。 1 ～本从站超时没有回应。 2 ～“写数据”报文长度错误。 3 ～“读数据”报文长度错误。 4 ～错误的配置报文。 5～错误的通信区配置。

当 *EN* 值为 1 时，则该指令执行。对于在 *PORT* 通信口上运行的 Kinco 互联网络，用户可以在主站中调用该指令来读取从站 *SLAVE* 的通信情况。

在工程中的【PLC 硬件配置】中 CPU 的【通信设置】页面中，用户可以看到本机各个通信口的具体编号。

10.5 CAN 总线的使用

10.5.1 概述

KPLC 大多数系列产品都提供了 CAN 总线通信口，支持多种通信协议，方便用户连接各种仪器仪表、伺服驱动器等设备。如果只有 1 个 CAN 通信口，则命名为 CAN1；如果有 2 个 CAN 通信口，则分别被命名为 CAN1 和 CAN2。下表列出了各系列 PLC 提供的 CAN 通信口及其所支持的协议：

系列		提供的 CAN 口	通信协议					终端电阻（120Ω）
			扩展协议(3)	自由通信(4)	Kinco 运动控制	CANOpen 主站	CANOpen 从站	
K6(1)(2)		CAN2	--	支持	支持	支持	--	内置，用拨码开关选择
KS	KS105	CAN1	支持	支持	--	--	--	
	KS105C1	CAN1	--	支持	支持	支持	支持	
	KS105C2	CAN1	支持	支持	--	--	--	
	KS101M	CAN2	--	支持	支持	支持	--	
KW		CAN1	支持	支持	支持	支持	--	
MK		CAN1	支持	支持	--	--	--	无
K2	K209M	CAN1	支持	支持	--	--	--	
		CAN2	--	支持	支持	支持	--	
	其它	--	--	--	--	--	--	
K5(2)		--	--	--	--	--	--	

注：（1）K6 本体并不标配 CAN2，用户可以使用 KB6-CAN 的 BD 板来增加此接口。

（2）K6/K5 系列 CPU 均有专用的扩展总线接口，都可以连接 K6/K5 的各种扩展模块。此扩展总线接口不对外开放。具体使用方法请参阅附录 F 扩展模块的使用中的相关描述。

（3）此处的扩展协议是步科公司自定义的协议，可以连接 KS 系列扩展模块。KS 系列扩展模块采用了 RJ45 接口形式，用户可以使用直连网线连接，因此扩展模块可以分布式布置，从 CPU 到最后一个扩展模块的总长度最大允许 25 米。

（4）自由通信是指用户利用 KPLC 提供的发送、接收等指令进行编程，通信数据格式及通信过程完全由用户程序进行控制。CAN1 和 CAN2 均支持自由通信，且自由通信可以和扩展协议、CANOpen 主站协议同时使用，**注意：当与其它协议同时使用时，CAN 自由通信相对处于更低的优先级，此时 CAN_INIT 指令无效，通信速率将采用其它协议的速率。而且，除了自由通信之外，其它通信协议均不允许同时使用！**

10.5.2 硬件接线

CAN 总线接口通常使用 CAN_H、CAN_L、CAN_GND 这 3 个管脚。

CAN_H、CAN_L 是 CAN 总线的信号数据管脚。在使用时，用户需要将网络中所有节点的 CAN_H、CAN_L 管脚分别连接在一起。

CAN_GND 是 CAN 总线的信号地，为 CAN 总线提供了参考电位。在使用时，有时候不接 CAN_GND 也能让 CAN 总线正常工作，但是建议用户将网络中所有节点的 CAN_GND 连接在一起，这样可以使所有节点都具有一个统一的参考电位，从而能够消除或者减少共模电压，避免因共模干扰造成的总线通信故障。

在为 CAN 总线实际布线时，建议采用总线型的拓扑结构，并且为了减少通信电缆上的信号反射，建议在总线的首、末两端加入 120Ω 的终端电阻。当通信距离较长时，通信电缆推荐采用屏蔽双绞线且屏蔽层单端良好接地（控制地），并且通信电缆应远离强干扰源、各种大功率线（包括设备的动力电缆）、开关频繁的脉冲信号线等。

10.5.3 扩展总线功能

10.5.3.1 概述

通常 CPU 的 CAN1 通信口支持扩展总线协议，若在用户工程的【硬件配置】中配置了扩展模块，则 CAN1 接口将作为扩展总线接口工作。

在出厂默认的设置中，CPU 模块在上电时会为每个扩展模块自动分配一个唯一的 ID 并配置各种参数，因此要求 CPU 与所有扩展模块同时上电或者扩展模块全部都先于 CPU 模块上电，否则可能会

导致程序执行错误。

KS 系列扩展模块采用了 RJ45 接口形式，用户可以使用直连网线连接，因此 KS 扩展模块可以分布式安装，从 CPU 到最后一个扩展模块的总长度最大允许 25 米。但是当 KS 扩展模块分布式安装时，各个模块可能安装于不同设备上，现场无法满足上述的默认上电次序要求。针对这种情况，KPLC 为 KS 系列扩展模块提供了 EX_ADDR 指令，用户可以通过调用这条指令来使得 CPU 和 KS 扩展模块能够以任意的次序上电或者断电。

10.5.3.2 如何使用 EX_ADDR 指令实现扩展模块的分布式应用

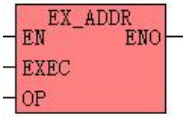
在实际应用中，若扩展模块与 CPU 模块相距很远或者安装于不同的设备上时，可能无法保证默认所需的上电次序。在这种情况下，用户可以按照下述步骤使用 EX_ADDR 指令来修改出厂默认的配置，使得各扩展模块可以在任意时刻上电或者断电而不会引起 PLC 执行程序错误。

1) 在用户工程中，在【硬件配置】中按所需次序依次加入各个扩展模块并按实际需求配置好，并在程序调用 EX_ADDR 指令（位于指令集的【CAN 指令】组中）。
2) 按【硬件配置】中的次序，将真实的 CPU 和所有扩展模块都连接好，然后按默认的次序上电（CPU 与所有扩展模块同时上电或者扩展模块全部都先于 CPU 模块上电）。
3) 将用户工程下载到 CPU 中。CPU 正常运行后，将 EX_ADDR 指令的参数值修改为 181（十进制），然后让 EX_ADDR 指令执行一次。指令成功执行后，各个扩展模块会自动保存好自己的 ID 和各种参数（比如信号形式、滤波方式等）。
4) 给本 PLC 系统断电。然后用户就可以将模块安装于所需的位置，**注意各模块的连接次序（从 CPU 开始）依然要与【硬件配置】中的次序一致**。此后，扩展模块再上电时就会自动读取保存好的数据并自动进入运行状态，无需 CPU 进行配置，因此可以独立于 CPU 在任意时间上电或断电。
5) 若用户需要恢复出厂默认的上电次序，则将程序中 EX_ADDR 指令的参数值修改为 99（十进制），然后让 EX_ADDR 指令执行一次。指令成功执行后，各个扩展模块会清除保存的 ID 和通道参数，以后再上电时就会等待 CPU 自动分配 ID 并配置参数。

10.5.3.3 扩展总线指令

扩展总线指令位于指令集的【CAN 指令】组中。

10.5.3.3.1 EX_ADDR（修改扩展模块配置）

	名称	指令格式	适用产品
LD	EX_ADDR		☑ KS ☑ KW ☑ K209M ☑ MK

参数	输入/输出	数据类型	允许的内存区
EXEC	输入	BOOL	M、V、L、SM
OP	输入	INT	M、V、L、常量

参数	描　述
EXEC	启动端。*EXEC*的上升沿触发本指令执行一次，需保证让 *EN*先于 *EXEC* 导通。
OP	操作码（十进制）。 181 --- 命令所有扩展模块保存好自身的 ID 和各种参数。 99 --- 命令所有扩展模块清除已保存的 ID 和各种参数。

本指令会根据 *OP*值向所连的扩展模块发送相应的命令：

- 若 *OP*值为 181，则扩展模块接收到命令后会自动保存好自己的 ID 和各种参数（比如信号形式、滤波方式等）。以后再上电时扩展模块会自动读取保存好的数据并进入运行状态，不需要 CPU 进行配置，因此可以独立于 CPU 在任意时间上电或断电。
- 若 *OP*值为 99，则扩展模块收到命令后会清除保存的 ID 和通道参数，以后再上电时就会等待 CPU 自动分配 ID 并配置参数。

➢ **LD 格式指令说明**

如果 EN 为 1，则该指令被扫描，若检测到 EXEC 的上升沿，则启动执行一次。

如果 EN 为 0，则指令不被扫描，也不会被执行。

10.5.4 CANOpen 主站功能

10.5.4.1 概述

CANOpen 总线具有开放性好、可靠性高、实时性较好、抗干扰能力强、成本低等优势，是工业控制中一种常用的现场总线，目前应用越来越广泛。

10.5.4.2 EDS 说明

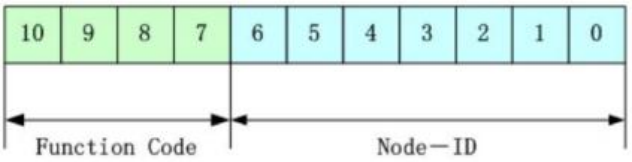
EDS（电子数据表格）文件是 PLC 所连接从站的标识文件或者类似码，通过该文件来辨认从站所属 的类型（是 401、402、403 中的何种类似，或者属于 402 中的哪一种设备）。该文件包含包含了从站的所有信息，比如生产厂家、序列号、软件版本、支持波特率种类、可以映射的 OD 及各个 OD 的属性等等参 数，类似于 Profibus 的 GSD 文件。因此在进行硬件配置前，我们首先需要把从站的 EDS 文件导入到上位组态软件中。

10.5.4.3 CANOpen 通信对象简介

CANOpen 应用层和通信规范（CiA DS301）是 CANOpen 协议的核心，适用于所有的 CANOpen 设备。在 DS301 中定义了多种 CANOpen 通信对象，同时也详细描述了这些对象的服务和协议。为了方便用户的应用，下面我们将介绍几种关键的对象及其通信协议。

10.5.4.3.1 COB-ID 说明

COB-ID 是 CANopen 通信协议的特有方式，它的全称是 Communication Object Identifier（通信对象 ID）， 这些 COB-ID 为 PDO 定义了相应的传输级别，有了这些传输级别后，PLC 编程软件和从站设备里配置里定义相同的传输级别和其里面的传输内容，这样控制器和从站设备都采用的同一个传输级别和传输内容后，数据的传输即透明化了，也就是双方都知道所要传输的数据内容了，也就不需要在传输数据时还需要对方回复数据是否传输成功。 缺省 ID 分配表是基于 CANopen 2.0A 定义的 11 位 CAN-ID（CANopen 2.0B 协议 COB-ID 是 27 位），包含一个 4 位的功能码部分和一个 7 位的节点 ID(Node-ID)部分，如图所示。



Node-ID 由系统集成商定义,即从站的站号,Node-ID 范围是 1～127(0 不允许被使用);Function Code 为数据传输的功能码，定义各种 PDO、SDO、管理报文的传输级别，功能码越小， 优先级越高。

下图为 CANopen 预定义的主/从连接集 COB-ID 分配表：

CANopen 预定义主/从连接集的广播对象			
对象	功能码 （ID-bits 10-7）	COB-ID	通讯参数在 OD 中的索引
NMT Module Control	0000	000H	-
SYNC	0001	080H	1005H，1006H，1007H
TIME SSTAMP	0010	100H	1012H，1013H
CANopen 主/从连接集的对等对象			
对象	功能码 功能码 （ID-bits 10-7）	COB-ID	通讯参数在 OD 中的索引
紧急	0001	081H-0FFH	1024H，1015H
PDO1(发送)	0011	181H-1FFH	1800H
PDO1(接收)	0100	201H-27FH	1400H
PDO2(发送)	0101	281H-2FFH	1801H
PDO2(接收)	0110	301H-37FH	1401H
PDO3(发送)	0111	381H-3FFH	1802H
PDO3(接收)	1000	401H-47FH	1402H
PDO4(发送)	1001	481H-4FFH	1803H
PDO4(接收)	1010	501H-57FH	1403H
SDO(发送/服务器)	1011	581H-5FFH	1200H
SDO(接收/客户)	1100	601H-67FH	1200H
NMT Error Control	1110	701H-77FH	1016H-1017H

注意：1、COB-ID 越小，优先级越高；

2、每一个 COB-ID 前面的功能码是固定格式；

3、图中 PDO 的 COB-ID 是 CANOpne 标准预定义的值。但在实际应用中，用户也可以修改 PDO 的 COB-ID，注意：00H、80H、100H、701H-77FH、081H-0FFH 均为系统占用，用户不允许使用这些值。

10.5.4.3.2 网络管理（NMT）

网络管理（NMT）面向 CANOpen 设备，采用了主从模式。NMT 服务可以初始化、启动、监视、复位或者停止 CANOpen 设备。在一个网络内必须存在一个 NMT 主站，主站拥有整个网络的控制权，即网络管理类（NMT）功能。下面介绍几种常用的 NMT 服务。

10.5.4.3.2.1 NMT 节点控制（NMT Node Control）

NMT 主站通过 NMT Node Control 报文来控制各从站的 NMT 状态（包括停止、预操作、操作和初始化）。从站设备必须支持 NMT 节点控制服务。NMT 节点控制报文格式如下：

COB-ID	Byte 0	Byte 1
0x000	CS(Command Specifier)	Node ID

其中， Node ID：目标从站的 ID。若 Node ID 为 0，则表示网络上所有的从站都需要执行本命令。

CS：命令字，不同数值的含义为： 1 表示启动目标节点；

2 表示停止目标节点；

128 表示目标节点进入预操作状态；

129 表示复位目标节点

130 表示目标节点复位通信参数

10.5.4.3.3 错误控制（NMT Error Control）

错误控制服务用于检测网络故障,包括节点保护（Node Guarding）和心跳（Heartbeat）两种方式。在实际应用中，必须为一个节点选择一种错误控制方式。

顺便提一下，心跳服务是在 DS301 后期的版本中新增加的，CiA 推荐使用。

- **NMT 节点保护（NMT Node Guarding）**

NMT 主站发送远程帧（无数据）：

COB-ID
0x700 + Node ID

NMT 从站发送如下应答报文：

COB-ID	Byte 0
0x700 + Node ID	Bit7：触发位，必须在每次节点保护应答中交替置“0”或者“1”。 Bit0-6：组合的数值表示从站状态。其中，0 表示 Boot-up，4 表示 STOPPED；5 表示 Operational；127 表示 Pre-Operational。

- **心跳（NMT Node Guarding）**

若一个节点被配置为心跳生产者，它会周期性地发送心跳报文。网络中另外一个或者多个节点作为心跳消费者，来处理各生产者的心跳报文。通常，主站作为心跳消费者，其它从站作为心跳生产者。心跳报文格式如下：

COB-ID	Byte 0
0x700 + Node ID	本节点的状态值。其中，0 表示 Boot-up，4 表示 STOPPED；5 表示 Operational；127 表示 Pre-Operational。

10.5.4.3.4 服务数据对象（SDO，Service Data Object）

SDO 通信是基于“客户机-服务器”模型。

通过使用索引（index）和子索引（sub-index），SDO 使一个 CANOpen 设备（作为客户机）可以直接访问其它 CANOpen 设备（作为服务器）的对象字典中的对象。通常，主站作为客户机。

SDO 有两种传输机制：加速传输，每次最多传输 4 字节数据；分段传输，允许分段传输超过 4 个字节的数据。下面简单介绍一下加速传输机制 SDO 的报文格式。

请求报文，Client -> Server：

COB-ID	Byte 0	Byte 1-2	Byte 3	Byte 4-7
0x600 + Node ID	SDO 命令字	对象索引	对象子索引	数据

应答报文，Server -> Client：

COB-ID	Byte 0	Byte 1-2	Byte 3	Byte 4-7
0x580 + Node ID	SDO 命令字	对象索引	对象子索引	数据

10.5.4.3.5 过程数据对象（PDO，Process Data Object）

PDO 用于传输实时的数据，一个 PDO 报文中最多包含 8 个字节的数据。

PDO 通信是基于“生产者-消费者”模型。以发送数据或者接收数据来区分，PDO 分为发送 PDO（TPDO）和接收 PDO（RPDO）。生产者支持 TPDO，消费者支持 RPDO。

PDO 通信没有协议规定，一个 PDO 报文中包含的内容是预先定义好的。在网络组态时，用户就定义了每个 PDO 的 COB-ID 和其中映射的对象，因此，生产者和消费者都能知道相应 PDO 的内容，从而对报文进行解析。

每个 PDO 在对象字典中由通信参数和映射参数来描述。下面介绍 PDO 的通信参数。

- **COB-ID**

  指明了该 PDO 使用的 COB-ID。

- **传输类型**

  指明了该 PDO 发送（或接收）的触发方式。它是一个 8 位无符号整数值。

  传输类型分为如下几类：

  - 同步方式：根据 SYNC 对象的计数值来触发发送（或接收）。传输类型的值为 0 表示“同步，非循环”方式，值为 1-240 表示“同步，循环”方式。
  - RTR-Only：仅适用于 TPDO，由接收到的 RTR 报文来触发 PDO 的发送。传输类型的值为 252 意味着接收到 SYNC 和 RTR 之后就发送 PDO。值为 253 意味着接收到 RTR 之后立即发送 PDO。
  - 事件驱动：当 CANOpen 设备内部事件发生之后就立即发送 PDO。传输类型值为 254 表示是设备制造商自定义的事件。值为 255 表示是设备子协议和应用层协议定义的事件，一般是指 PDO 内的数据值改变或者定时器定时时间到。

- **禁止时间**

禁止时间定义了该PDO 连续发送时的最小间隔时间。配置禁止时间是为了避免由于高优先级 PDO

发送过于频繁，始终占据总线，而使其它优先级较低的报文无法使用总线的问题。

- **事件定时器**

用于指定一个定时发送的周期值。它是一个 16 位无符号整数，单位是 ms。PDO 将以该定时值为周期来触发发送。若该数值为 0，则表示不使用事件定时器。

10.5.4.4 使用 CANOpen 主站功能

KPLC 的 CANOpen 主站功能具有如下特点：

- 采用 CAN2.0A 标准。符合 CANOpen 标准协议 DS301 V4.2.0。
- 支持 NMT 网络管理服务，包括 NMT Node Control 和 NMT Error Control，并作为 NMT 主站。
- KS 系列最大支持 8 个 CANOpen 从站，K6 系列最大支持 32 个 CANOpen 从站。
- 允许用户在 KincoBuilder 中为每个从站配置启动过程；
- 每个从站最多支持 8 个 TPDO 和 8 个 RPDO；总共最多支持 128 个 TPDO 和 128 个 RPDO。
- PDO 传输类型仅支持 254 或 255 异步传输方式（事件驱动方式）
- 支持客户端 SDO，并提供 SDO 读、写指令，SDO 指令支持标准的加速传输模式；

10.5.4.4.1 CANOpen 网络配置工具

在 KincoBuilder 中，进入【PLC 硬件配置】，在窗口上部的表格中选中 CPU 模块，然后在窗口下部的页面中单击【CANOpen 主站】，就进入了 CANOpen 的网络配置页面。

10.5.4.4.2 处理 EDS 文件

在【CANOpen 主站】->【网络配置】页面中，提供了如下按钮可以对 EDS 文件进行操作：

- 【导入 EDS】：单击此按钮，选择所需的 EDS 文件，就可以将其导入到 Kincobuilder 中并存储。导入一个 EDS 之后，相应的从站设备就在显示在【所有从站模块列表】中，之后才可以进行组态。
- 【删除】：在下方的【所有从站模块列表】中选择一个从站设备，然后单击【删除】按钮，就可以将该设备从列表中删除，同时也将它的 EDS 文件从 Kincobuilder 中删除。

- 【导出所有 EDS】：可以将 Kincobuilder 中已有的全部从站 EDS 文件合并导出到一个文件中（扩展名为 .ALLEDS）。这个功能在卸载 Kincobuilder 时会比较有用，用户在卸载前可以使用这个功能将所有从站的 EDS 文件备份，以后可以将备份的.ALLEDS 文件直接导入即可。
- 【导入所有 EDS】：可以将一个 EDS 备份文件（扩展名为 .ALLEDS ）导入到 Kincobuilder 中，导入之后该文件中包含的所有从站设备都将显示在下方的【所有从站模块列表】中。

10.5.4.4.3 CANOpen 网络配置过程

1) 配置全局参数

进入【主站及全局配置】页面，如下图：

- 【波特率】：选择主站所用的波特率。注意网络上所有节点的波特率必须一致。
- 【SDO 超时】：设定主站发送 SDO 请求报文之后的超时等待时间，若超过这个时间没有收到相应从站的应答报文，则会报告错误。SDO 超时值设定一般不需要超过 100ms。
- 【启动时配置各从站】：若选中此项，那么主站除了控制各个从站的 NMT 状态转换外，在启动时主站还会根据各个从站的参数组态情况依次发送相应的配置命令来对各个从站进行配置（如从站的错误控制方式、PDO 映射等）。若不选中此项，则主站仅仅控制各个从站的 NMT 状态转换。

2) 配置各从站

进入【网络配置】页面，继续配置网络上的从站节点及其参数，如下图：

页面中所有的功能按钮，都有相应的右键菜单命令。用户在相关位置单击鼠标右键，就会弹出相应的右键菜单，此时可以使用菜单命令。下面描述配置一个从站的常用过程。

a) 向网络中添加一个从站设备

从左侧的树形列表中双击需要加入网络的从站类型，就会添加一个该类型的从站设备到网络中去，并显示在右侧的表格中。

b) 配置从站设备的站号（ID）、监督类型等参数：

右侧表格中的【地址】列就是从站的站号（ID）。第 1 行是 1 号站的位置。

添加一个从站设备后，就会显示出它的默认配置参数。添加时，Kincobuilder 默认是将设备从上到下依次添加到表格中。用户可以鼠标单击表格中的行来选中一个从站，然后可以单击【上移】、【下移】按钮来调整它的站号，也可以单击【删除】按钮将此设备从网络中删除。

【监督类型】用于配置该节点的 NMT Error Control 方式，包括节点保护和心跳两种方式。若从站设备同时支持这两种方式，推荐优先选择使用心跳方式。

【监督时间】表示前面所选的节点保护方式或者心跳方式的周期值。建议在实际应用中，这个周期值设置不要太小，比如可以设置在 2000 以上。

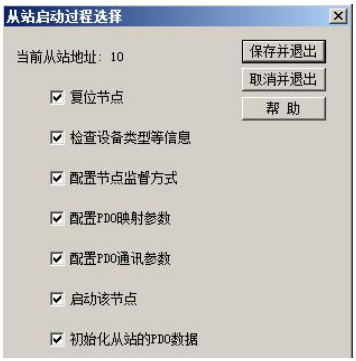
【心跳消费者时间】主站会定时查询是否收到从站的心跳报文，若超过这个“心跳消费者”时间仍然没有收到，则认为该从站已经离线并进行相应的故障处理。建议在实际应用中，这个周期值设置不要太小，比如可以设置在 2000 以上。

【故障处理】用于选择当主站检测到该从站故障后采用的处理方式，包括“无”、“停止节点”

和“停止网络”三种选项。主站能够检测的故障包括 SDO 命令超时没有应答、节点保护或者心跳报文超时、收到从站发送的部分类型的紧急报文等。

c) 配置从站的启动过程：

鼠标单击表格中的一个从站，然后单击【启动过程】，就可以选择在网络启动过程中主站需要对该从站进行何种配置。



【复位节点】：主站在向从站发送配置命令之前，是否先发送“复位节点”命令。

【检查设备类型】：主站在向从站发送配置命令之前，是否先读取设备信息进行检查。

【配置节点监督方式】：主站是否需要配置从站的监督类型及其参数。

【配置 PDO 映射参数】：主站是否需要配置从站的 PDO 映射参数。

【配置 PDO 通信参数】：主站是否需要配置从站的 PDO 通信参数。

【启动该节点】：配置完成后，主站是否需要向该从站发送“启动节点”命令。

【初始化从站的 PDO 数据】：在启动该从站后，主站是否需要把该从站全部 RPDO 中的数据清 0 并立即发送一次。

d) 配置各从站的 PDO：

进入【对象字典映射】页面，为网络中所有的从站配置 PDO。

页面左侧部分【已添加从站列表】显示了已经加入到网络的所有从站，以及各从站对象字典中允许映射到 PDO 中的对象。其中，【发送 PDO】中的对象只能映射到该从站的 TPDO 中，【接收 PDO】列表中的对象只能映射到该从站的 RPDO 中。

在【已添加从站列表】中鼠标单击一个从站，那么在右侧就会显示出该从站所有的 PDO，用户可以对各 PDO 的进行配置：

- 通信参数

在右侧的表格中，选中一个 PDO，可以修改其定时时间、禁止时间等通信参数。

其中，从站中前 4 个的 TPDO 和 RPDO 的 COB-ID 不允许修改，采用了 DS301 中预定义连接集中的默认值。后 4 个的 TPDO 和 RPDO 允许用户自己输入合法的 COB-ID 值。

- 映射参数

在左侧的对象列表中，双击一个对象，就会将这个对象加入到了当前 PDO 中，同时 Kincobuilder 会自动为这个对象分配一个 PLC 的 V 区地址，比如 VW1006，用户在程序中操作这个 V 区地址就相当于操作相应的对象了。

3）复制从站、粘贴从站

在【网络配置】页面中，提供了【复制从站】、【粘贴从站】和【粘贴从站（重分内存）】这 3 个按

钮。**该功能常用于更换 PLC 型号中，事先复制旧型号配置，粘贴到新型号**，如下图。

【复制从站】：选中一个已经配置好的从站，然后单击此按钮，就会复制该从站的所有信息（其中包括所有 PDO 的通信参数、映射参数等）。如果所选从站没有配置任何 PDO，那么复制失败并提示相应信息。

【粘贴从站】：复制成功一个从站后，单击选中表格中的一个空行，然后单击此按钮，就会将刚才复制的从站信息粘贴到该行并生成一个新从站。注意：新从站 PDO 中的各映射对象对应的 PLC 内存地址，还是保持与源从站中的一致，没有重新分配，用户需要自己修改。

【粘贴从站（重分内存）】：操作方法与【粘贴从站】一样，但是不同之处是新从站 PDO 中各映射对象的 PLC 内存地址会自动进行分配，无需用户修改。

10.5.4.5 CANopen 相关指令

10.5.4.5.1 CANopen 功能及相关指令

- **CANopen 支持的功能主要有：**

- 支持 NMT 管理报文；
- 支持 CANopen 预定义连接集模式，PDO 总数量为 128TxPDO 和 128RxPDO,每个从站可配置的 PDO 数量为 1-8TxPDO、 1-8RxPDO；
- 可以连接 8 个（KS 系列）或 64 个（K6 系列）从站,从站站号 1-126 任意设置；
- 支持紧急报文、节点保护、心跳报文；
- 可以对从站的启动过程进行设置；
- 支持 EDS 导入功能；

- 通过 SDO_WRITE 和 SDO_READ 命令字实现对 SDO 的操作；
- 支持 254、255 两种 PDO 传输模式，并具有定时发送 PDO 的功能，能同时发送 8 个 PDO 数据，且定时时间可设；
- 支持各种波特率通信：10K/20K/50K/125K/250K/500K/800K/1M；
- 通过系统字检查网络上主从站的状态；
- 具有多 PLC 通过 CAN 总线联网能力；
- 具有从站错误处理功能，当从站出错后，主站可以选择的处理方式有三种，即：停止节点、停止网络 、无 。

➢ **SDO 命令：**

SDO(服务数据对象，Servic Data Object)是 CANOpen 定义的一种通信对象（报文），主要用来在设备之间传输低优先级的数据，典型是用来对从设备进行配置、管理,比如用来修改伺服电流环、速度环、位置环的 PID 参数，PDO 配置参数等，这种数据传输跟 Modbus 的方式一样，即主站发出后，需要从站返回数据响应。这种数据只适合对参数的设置，不适合于对实时性要求较高的数据传输。SDO 的具体的报文格式请参见 10.5.4 CANOpen 主站功能。

➢ **其他命令：**

CO_STATE（获取从站的状态和错误）：CANOpen 主站在运行过程中会检测并记录所有从站的状态、故障以及发送的紧急报文。

CO_RESTART（重新配置并启动从站）：CANOpen 在主站正常运行过程中，用户可以调用本指令来重新配置并启动目标从站。

10.5.4.5.2 SDO 指令

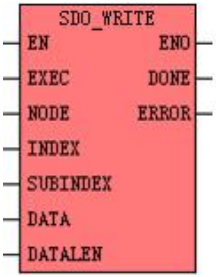
SDO 指令位于指令集的【CAN 指令】组中。

当使用 Kinco 运动控制功能或者 CANOpen 主站功能时，可以使用 SDO 指令。

在一个用户工程中，最多允许使用 64 个 SDO 指令。

10.5.4.5.2.1 SDO_WRITE（SDO 写操作）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	SDO_WRITE			☑ KS ☑ KW ☑ K209M ☑ K6
IL	SDO_WRITE	SDO_WRITE *EXEC, NODE, INDEX, SUBINDEX, DATA, DATALEN, DONE, ERROR*	U	

参数	输入/输出	数据类型	允许使用的内存区
EXEC	输入	BOOL	I、Q、V、M、L、SM、RS、SR
NODE	输入	BYTE	I、Q、V、M、L、SM、常量
INDEX	输入	WORD	I、Q、V、M、L、SM、常量
SUBINDEX	输入	BYTE	I、Q、V、M、L、SM、常量
DATA	输入	BYTE	I、Q、V、M、L、SM
DATALEN	输入	BYTE	I、Q、V、M、L、SM、常量
DONE	输出	BOOL	Q、M、V、L、SM
ERROR	输出	DWORD	Q、M、V、L、SM

注意：NODE, INDEX, SUBINDEX, DATALEN 必须同时为常量或同时为变量；DATA 与 DATALEN 参数共同组成了一个可变长度的内存块，该内存块内必须都为合法的内存地址。

各个参数的具体使用说明，见下表：

参数	功能
EN	使能端。若 *EN* 为 1 时，则该指令被使能，允许执行。

EXEC	启动端。*EXEC*的上升沿用于启动 SDO 通信。最好能够保证让 *EN*先于 *EXEC*导通。
NODEID	待访问节点的地址
Index	待访问对象在 OD 中的索引
SubIndex	待访问对象在 OD 中的子索引
Data	待发送数据存放的起始字节
DataLen	待发送数据的长度，单位：字节， 范围 1、2、4，与待访问的对象有关，如果待访问对象数据类型为字节则长度为 1，数据类型为字长度 2，数据类型为双字长度为 4
DONE	执行结果指示。 若 SDO 正在执行，DONE 为 0；若 SDO 通信结束（收到回应或者超时），DONE 为 1。
ERROR	错误信息。见下表

SDO 错误信息说明：

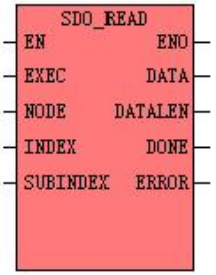
错误码	说明
0	无错误。
1	主站没有启用
2	目标节点不存在
3	输入参数值错误（比如数据长度）
4	目标节点上一次的命令尚未得到回应
5	PLC 的发送或者接收缓冲区已满
6	指令超时没有收到回应报文
7	收到的回应报文错误（不是期望的回应报文、长度错误等等）
8	收到终止报文
9	本工程中，SDO 指令数量超出限制
10	没有安装 KB6-CAN 的 BD 板

- LD

如果 EN 为 1，则该指令被执行，否则不执行。

10.5.4.5.2.2 SDO_READ（SDO 读操作）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	SDO_READ			☑ KS ☑ KW ☑ K209M ☑ K6
IL	SDO_READ	SDO_READ *EXEC, NODE, INDEX, SUBINDEX, DATA, DATALEN, DONE, ERROR*	U	

参数	输入/输出	数据类型	允许使用的内存区
EXEC	输入	BOOL	I、Q、V、M、L、SM、RS、SR
NODE	输入	BYTE	I、Q、V、M、L、SM、常量
INDEX	输入	WORD	I、Q、V、M、L、SM、常量
SUBINDEX	输入	BYTE	I、Q、V、M、L、SM、常量
DATA	输出	BYTE	I、Q、V、M、L、SM
DATALEN	输出	BYTE	I、Q、V、M、L、SM
DONE	输出	BOOL	Q、M、V、L、SM
ERROR	输出	DWORD	Q、M、V、L、SM

注意：NODE, INDEX 和 SUBINDEX 必须同时为常量或同时为变量；DATA 与 DATALEN 参数共同组成了一个可变长度的内存块，该内存块内必须都为合法的内存地址。

各个参数的具体使用说明，见下表：

参数	功能
EN	使能端。若 *EN* 为 1 时，则该指令被使能，允许执行。

EXEC	启动端。*EXEC*的上升沿用于启动 SDO 通信。最好能够保证让 *EN*先于 *EXEC*导通。
NODEID	待访问节点的地址
Index	待访问对象在 OD 中的索引
SubIndex	待访问对象在 OD 中的子索引
Data	接收到数据存放的起始字节
DataLen	接收到数据的长度，单位：字节，范围 1、2、4，与待访问的对象有关，如果为 4 代表读取的数据占用 Data 中存放的起始字节开始的 4 个字节的长度。
DONE	执行结果指示。 若 SDO 正在执行，DONE 为 0；若 SDO 通信结束（收到回应或者超时），DONE 为 1。
ERROR	错误信息。见下表

SDO 错误信息说明:

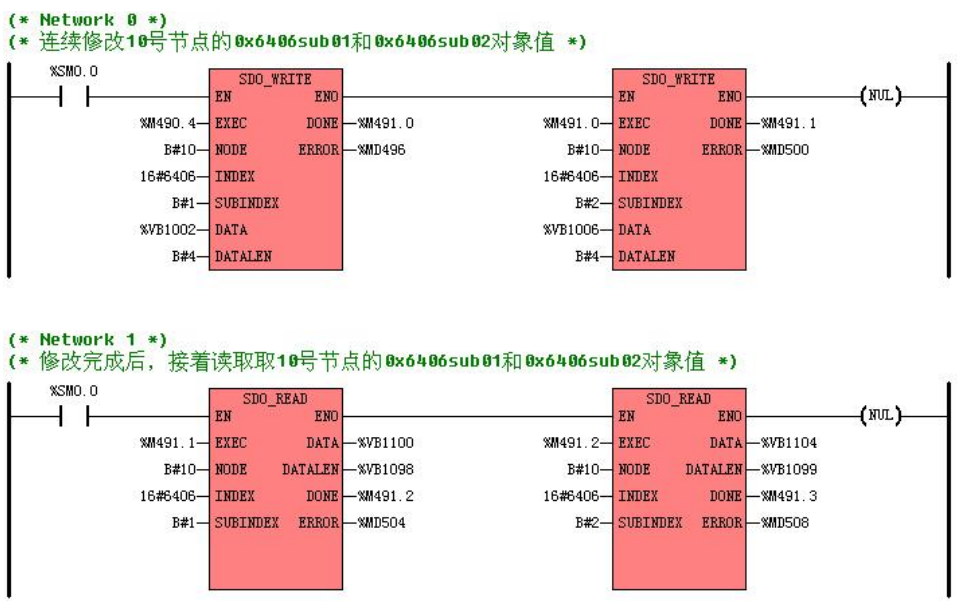
错误码	说明
0	无错误。
1	主站没有启用
2	目标节点不存在
3	输入参数值错误（比如数据长度）
4	目标节点上一次的命令尚未得到回应
5	PLC 的发送或者接收缓冲区已满
6	指令超时没有收到回应报文
7	收到的回应报文错误（不是期望的回应报文、长度错误等等）
8	收到终止报文
9	本工程中，SDO 指令数量超出限制
10	没有安装 KB6-CAN 的 BD 板

- LD

如果 EN 为 1，则该指令被执行，否则不执行。

10.5.4.5.2.3 SDO_WRITE、SDO_READ 使用举例

LD

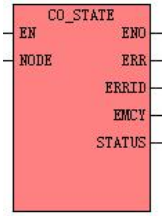


IL

```
(* Network 0 *)
(*连续修改 10 号节点的 0x6406sub01 和 0x6406sub02 对象值*)
LD      %SM0.0
SDO_WRITE %M490.4, B#10, 16#6406, B#1, %VB1002, B#4, %M491.0, %MD496
SDO_WRITE %M491.0, B#10, 16#6406, B#2, %VB1006, B#4, %M491.1, %MD500
(* Network 1 *)
(*修改完成后，接着读取取 10 号节点的 0x6406sub01 和 0x6406sub02 对象值*)
LD      %SM0.0
SDO_READ %M491.1, B#10, 16#6406, B#1, %VB1100, %VB1098, %M491.2, %MD504
SDO_READ %M491.2, B#10, 16#6406, B#2, %VB1104, %VB1099, %M491.3, %MD508
```

10.5.4.5.3 CO_STATE（获取从站的状态和错误）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	CO_STATE			☑ KS ☑ KW ☑ K209M ☑ K6
IL	CO_STATE	CO_STATE*NODE, ERROR, ERRID, EMCY, STATUS*	U	

参数	输入/输出	数据类型	允许使用的内存区
NODE	输入	INT	V、M、L、常量
ERR	输出	BOOL	Q、M、V、L
ERRID	输出	BYTE	V、M、L
EMCY	输出	WORD	V、M、L
STATUS	输出	BYTE	V、M、L

各个参数的具体使用说明，见下表：

参数	功能
EN	使能端。若 *EN* 为 1 时，则该指令被使能，允许执行。
NODE	待访问节点的地址
ERR	指明目标从站是否发生过故障。1 表示从站曾经发生过故障，0 表示没有发生过故障。
ERRID	指明从站故障的故障码。具体含义见后面列表。
EMCY	指明从站曾经发送过的紧急报文中的故障码。故障码具体含义请查阅 DS301 标准。

STATUS	指明从站目前的状态。该状态值来自于从站的心跳或者节点保护报文。状态值的含义请参阅 10.5.3.3.3 错误控制。

故障码（ERRID）的说明，见下表：

错误码	说明
0	无错误。
1	本从站对 SDO 请求报文回应了终止报文（命令码为 0x80）。
2	本从站对 SDO 请求报文的回应错误（比如报文长度错误、对象错误等）。
3	本从站曾经发送过紧急报文。
4	本从站心跳超时或者节点保护超时。
5	本从站对 SDO 请求报文没有进行回应。
6	本从站由于其它节点发生故障而被导致处于 STOP 状态。 注：如果用户在硬件配置中将某个从站的“故障处理”方式设置为“停止网络”，那么当主站检测到这个从站故障后会发送 STOP 命令让网络中所有从站进入 STOP 状态。

CANOpen 主站在运行过程中会检测并记录所有从站的状态、故障以及发送的紧急报文。 如果 EN 为 1，则本指令被执行,会读取目标从站（站号 NODE）的记录值，并输出到相应参数中（ERRID、 EMCY、 STATUS）。

- LD

如果 EN 为 1，则该指令被执行，否则不执行。

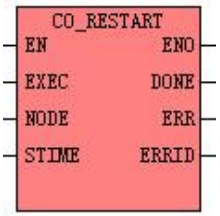
- IL

如果 CR 值为 1，则该指令被执行，否则不执行。

该指令的执行不影响 CR 值。

10.5.4.5.4 CO_RESTART（重新配置并启动从站）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	CO_RESTART			☑ KS ☑ KW ☑ K209M ☑ K6
IL	CO_RESTART	CO_RESTART*EXEC, NODE, STIME, DONE, ERR, ERRID*	U	

参数	输入/输出	数据类型	允许使用的内存区
EXEC	输入	BOOL	I、Q、V、M、L、SM
NODE	输入	INT	V、M、L、常量
STIME	输入	INT	V、M、L、常量
DONE	输出	BOOL	Q、M、V、L
ERR	输出	BOOL	Q、M、V、L
ERRID	输出	BYTE	V、M、L

注意：NODE 和 STIME 参数必须同时为常量或同时为变量。在一个用户工程中，最多允许使用 32 个 CO_Restart 指令。

各个参数的具体使用说明，见下表：

参数	功能
EN	使能端。若 *EN* 为 1 时，则该指令被使能，允许执行。
EXEC	启动端。EN 必须先于 EXEC 导通。 EXEC 的上升沿会启动本指令，执行一次重启从站的过程。 EXEC 的下降沿会停止本指令的执行，终止重启从站的过程。

NODE	待访问节点的地址
STIME	主站发送 SDO 请求报文后等待从站回应的最长时间。
DONE	指令执行结果标志位。若指令在执行过程中，DONE 输出为 0。若指令执行结束（无论 成功还是失败），则 DONE 立即变为 1。
ERR	指令执行错误标志位。若指令执行时发生错误，则被置 1。
ERRID	指令执行的错误码。如果指令执行时发生错误，则 ERRID 就是具体的错误信息。

故障码（ERRID）的说明，见下表：

错误码	说明
0	无错误。
1	目标从站站号错误。
2	目标从站正在执行重启从站的过程，在完成之前禁止再次执行。
3	主站发送 CAN 报文错误（可能是线路、硬件接口、软件缓冲区满等原因）。
4	主站发送的 SDO 请求报文，目标从站超时没有回应。
5	主站发送的 SDO 请求报文，目标从站回应错误。
6	用户主动停止本指令的执行（将 EXEC 设置为 0）。

- LD

如果 EN 为 1，则该指令被执行，否则不执行。

- IL

如果 CR 值为 1，则该指令被执行，否则不执行。

该指令的执行不影响 CR 值。

在主站正常运行过程中，用户可以调用本指令来重新配置并启动目标从站（站号为 NODE）。从站的各种参数会采用【硬件配置】->【CANOpen 主站】中对本从站已经配置好的参数，包括启动过程、 错误监督参数、PDO 的通信和映射参数等等。

注意：如果用户在【硬件配置】将某个从站的“故障处理”方式选择为“停止网络”，那么当该从站发生故障时可能会造成本指令不能正常执行！

若 EN 为 1，那么 EXEC 的上升沿会触发执行本指令，首先 DONE 输出为 0，然后主站将会读取【硬件配置】 中已经配置好的从站参数，对目标从站依次进行如下操作：

1）发送命令让目标从站进入预操作状态

2）根据用户在硬件配置中对本从站【启动过程】的配置，主站会继续进行操作　如果选中了“复位节点”，则主站将向从站发送“复位通信”命令。

- 如果选中了“配置节点监督方式”，则主站将发送 SDO 来配置目标从站的心跳或者节点保护参数。
- 如果选中了“配置 PDO 映射参数”，则主站将发送 SDO 来配置目标从站所有 PDO 的映射参数。
- 如果选中了“配置 PDO 通信参数”，则主站将发送 SDO 来配置目标从站所有 PDO 的通信参数。
- 如果选中了“启动该节点”，那么完成上述过程后，主站将向目标从站发送“启动节点”命令。

注意：无论是否有配置，本指令都不会向目标从站发送“PDO 的初始化数据”。

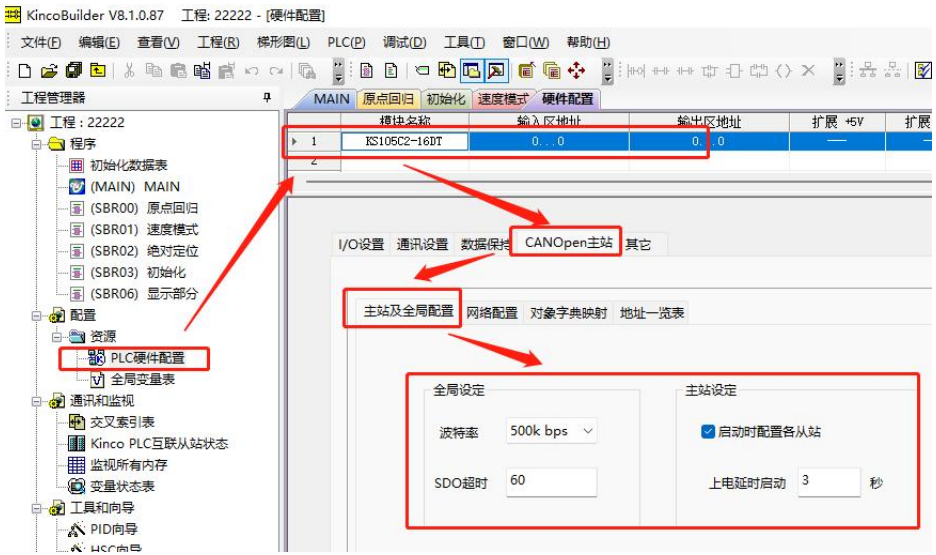
当上述过程成功执行完后，本指令会退出，并将 DONE 立即输出为 1，ERR 和 ERRID 均输出为 0。

如果指令在执行过程中发生任何错误，或者 EXEC 变为 0，则本指令也会退出，并将 DONE、ERR 立即输出为 1，ERRID 输出相应的错误代码。另外，对于特定的错误（包括 SDO 超时无回应、SDO 响应错误），本指令将依据【硬件配置】中用户选择的“故障处理”方式进行处理：如选择为“无”，则不进行处理；如果选择“停止节点”，则向目标节点发送“停止节点”命令；如果选择“停止网络”，则向网络中所有节点发送“停止节点”命令。

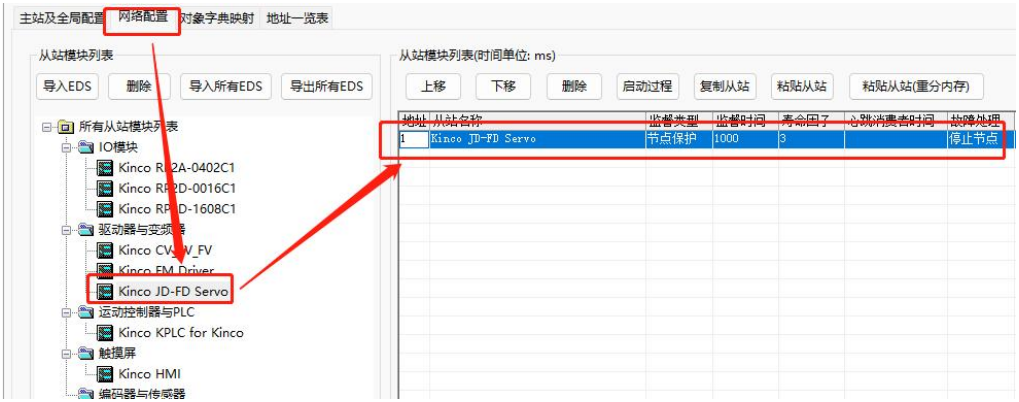
10.5.5 CANOpen 应用案例

下面以 KS105C2-16DT 带 Kinco 伺服为例进行说明整个配置过程和应用：

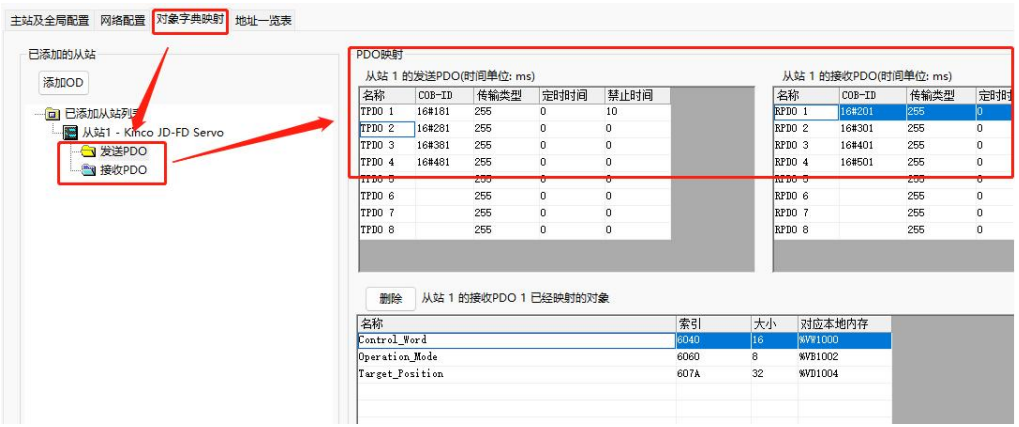
1、如图所示设置 CANOpen 主站的通信参数等（波特率主从站需要保持一致）。



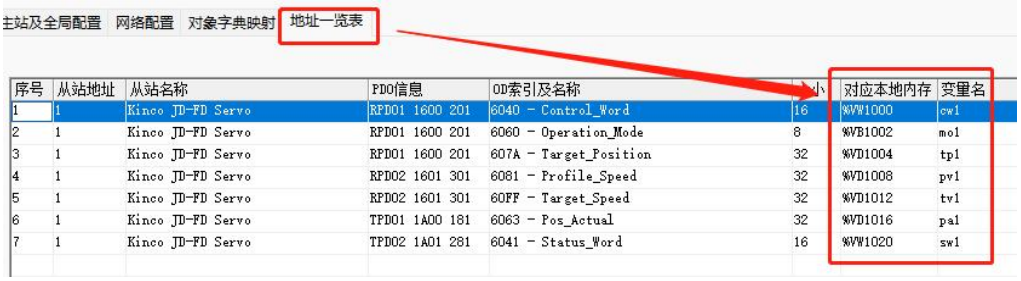
2、设置网络配置，添加 CANOpen 从站设备，同时设置监督类型和故障处理方式。



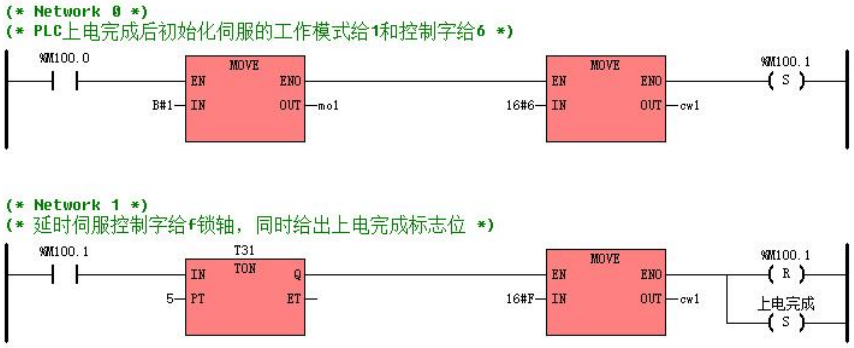
3、对象字典映射，将需要用到对象配置到相应发送 PDO 和接收 PDO 里面，实际位置、实际电流等实时变化的对象需要配置相应的禁止时间否则会导致 CAN 总线负荷太大出现通信故障。



4、配置完成的地址一览表，伺服对象对应的 PLC 内存地址如右侧所示，可以命名相应的全局变量名以便编程使用。

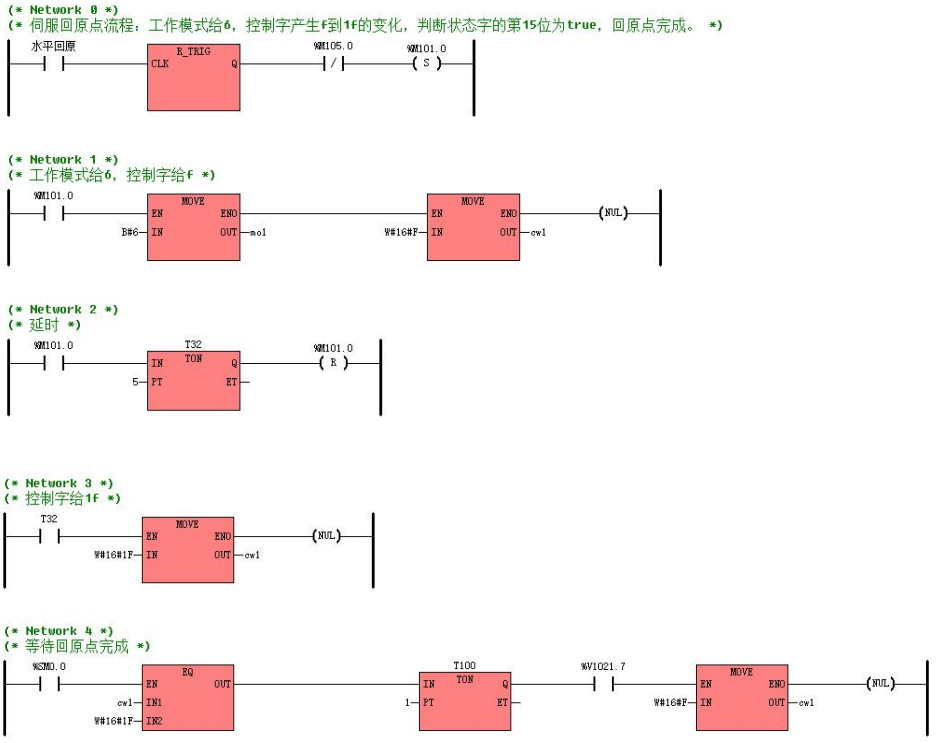


5、编写控制伺服的程序，包括上电初始化、原点回归、速度控制、绝对定位、显示部分子程序和 MAIN 主程序，主程序的作用主要是调用子程序。程序如下：

LD	上电初始化子程序：PLC 上电给伺服初始化的过程，工作模式给 1、控制字给 6，延时控制字给 F 伺服锁轴同时给出上电完成标志。 

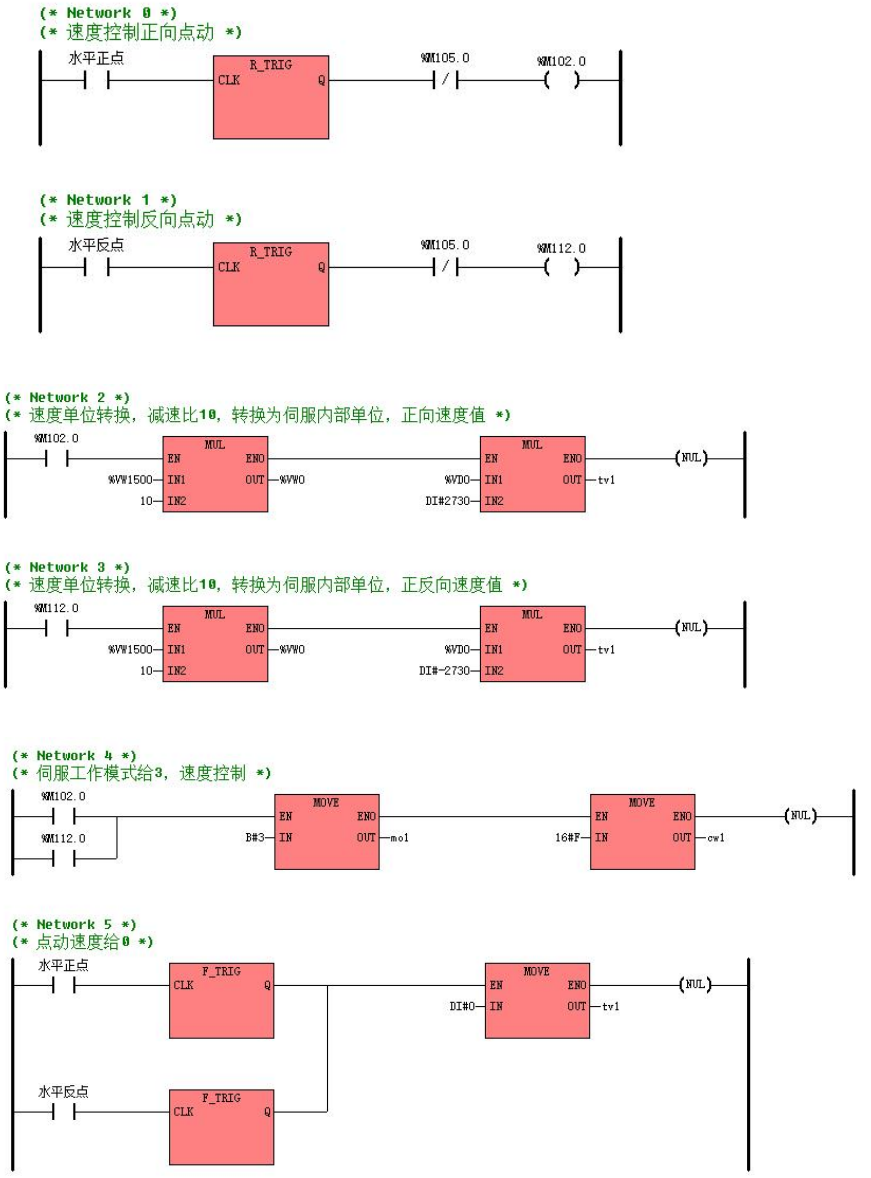
LD

回原点子程序：工作模式给 6，控制字产生 f 到 1f 的变化，等待状态字的第 15 位由 false 变为 true，回原点完成。



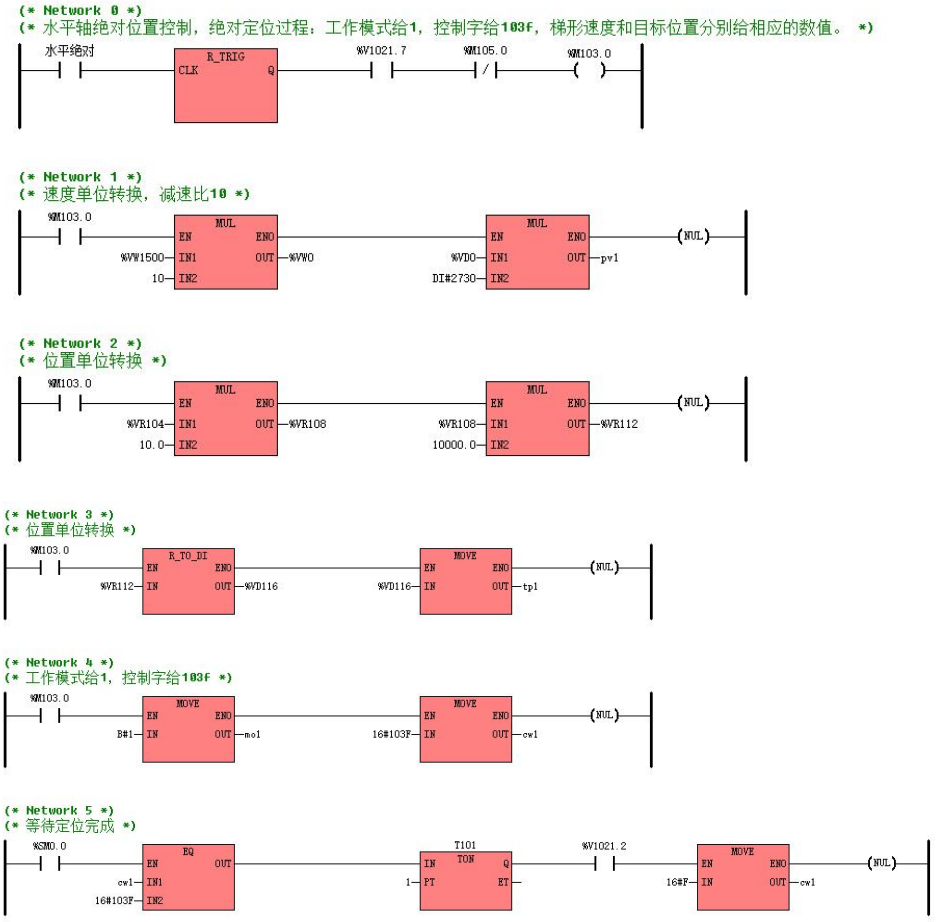
LD

速度控制子程序：点动控制伺服以速度模式运动，工作模式给 3、控制字给 f、目标速度需要根据工程单位转换为伺服内部值。



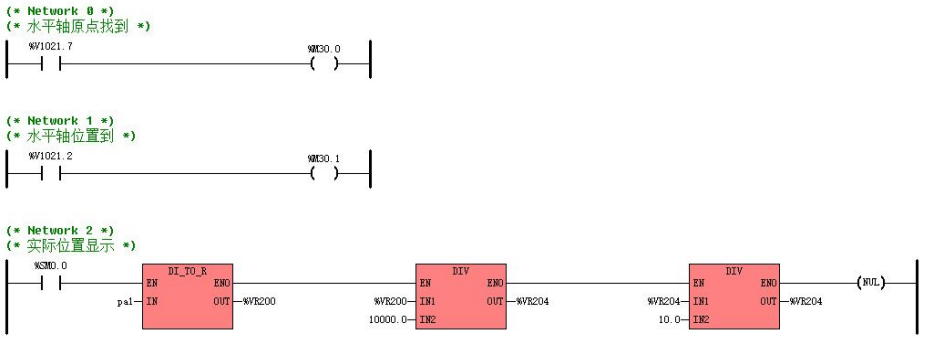
LD

绝对定位子程序：工作模式给 1、控制字给 103f，梯形速度和目标位置需要由工程单位转换为伺服内部单位，等待状态字的第 10 位由 false 变为 true 定位完成。



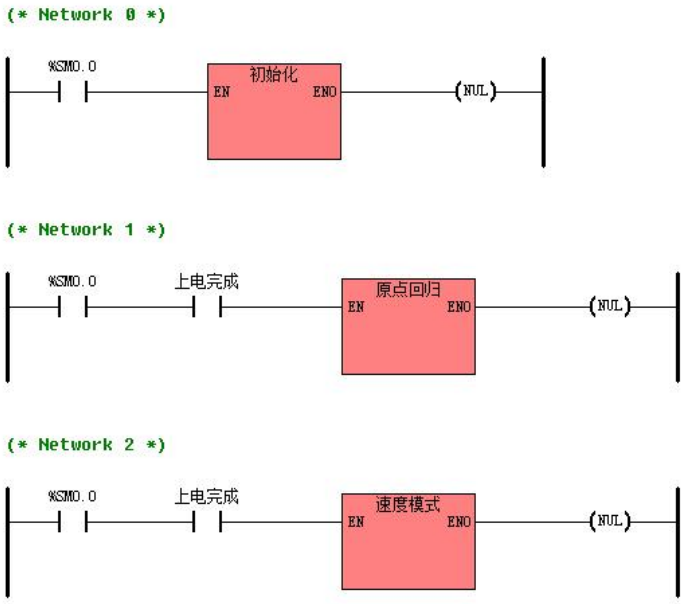
LD

显示部分：显示原点找到标志、定位完成标志、实际位置。

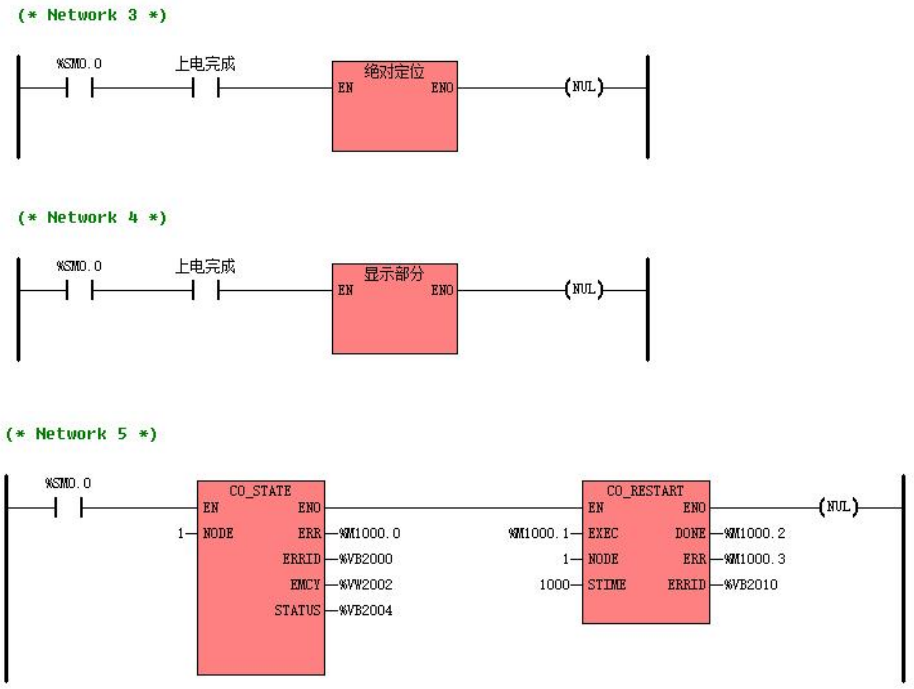


LD

主程序 MAIN：调用各个子程序



LD



CO_STATE:监控从站状态：是否掉线、紧急报文、节点保护状态。

CO_RESTART:重启节点，当节点因为故障停止或主站先于从站上电，可以通过此指令重启节点使从站进入操作状态。

IL	上电初始化子程序：PLC 上电给伺服初始化的过程，工作模式给 1、控制字给 6，延时控制字给 F 伺服锁轴同时给出上电完成标志。
	(* Network 0 *)
	(*PLC 上电完成后初始化伺服的工作模式给 1 和控制字给 6*)
	LD %M100.0
	MOVE B#1, mo1
	MOVE 16#6, cw1
	S %M100.1
	(* Network 1 *)
	(*延时伺服控制字给 f 锁轴，同时给出上电完成标志位*)
	LD %M100.1
	TON T31, 5
	MOVE 16#F, cw1
	R %M100.1
	S 上电完成

IL	回原点子程序：工作模式给 6，控制字产生 f 到 1f 的变化，等待状态字的第 15 位由 false 变为 true，回原点完成。
	(* Network 0 *)
	(*伺服回原点流程:工作模式给 6,控制字产生 f 到 1f 的变化,判断状态字的第 15 位为 true，回原点完成。*)
	LD 水平回原
	R_TRIG
	ANDN %M105.0
	S %M101.0
	(* Network 1 *)
	(*工作模式给 6，控制字给 f*)
	LD %M101.0
	MOVE B#6, mo1
	MOVE W#16#F, cw1
	(* Network 2 *)
	(*延时*)
	LD %M101.0
	TON T32, 5
	R %M101.0
	(* Network 3 *)
	(*控制字给 1f*)
	LD T32
	MOVE W#16#1F, cw1
	(* Network 4 *)
	(*等待回原点完成*)
	LD %SM0.0
	EQ cw1, W#16#1F
	TON T100, 1
	AND %V1021.7
	MOVE W#16#F, cw1

IL	速度控制子程序：点动控制伺服以速度模式运动，工作模式给 3、控制字给 f、目标速度需要根据工程单位转换为伺服内部值。
	(* Network 0 *)
	(*速度控制正向点动*)
	LD 水平正点
	R_TRIG
	ANDN %M105.0
	ST %M102.0
	(* Network 1 *)
	(*速度控制反向点动*)
	LD 水平反点
	R_TRIG
	ANDN %M105.0
	ST %M112.0
	(* Network 2 *)
	(*速度单位转换，减速比 10，转换为伺服内部单位，正向速度值*)
	LD %M102.0
	MOVE %VW1500, %VW0
	MUL 10, %VW0
	MOVE %VD0, tv1
	MUL DI#2730, tv1
	(* Network 3 *)
	(*速度单位转换，减速比 10，转换为伺服内部单位，正反向速度值*)
	LD %M112.0
	MOVE %VW1500, %VW0
	MUL 10, %VW0
	MOVE %VD0, tv1
	MUL DI#-2730, tv1

IL	(* Network 4 *)
	(*伺服工作模式给 3，速度控制*)
	LD %M102.0
	OR %M112.0
	MOVE B#3, mo1
	MOVE 16#F, cw1
	(* Network 5 *)
	(*点动速度给 0*)
	LD 水平正点
	F_TRIG
	OR(
	LD 水平反点
	F_TRIG
	)
	MOVE DI#0, tv1

IL	绝对定位子程序：工作模式给 1、控制字给 103f，梯形速度和目标位置需要由工程单位转换为伺服内部单位，等待状态字的第 10 位由 false 变为 true 定位完成。 (* Network 0 *) (*水平轴绝对位置控制，绝对定位过程：工作模式给 1，控制字给 103f，梯形速度和目标位置分别给相应的数值。*) LD 水平绝对 R_TRIG AND %V1021.7 ANDN %M105.0 ST %M103.0

IL	(* Network 1 *)
	(*速度单位转换，减速比 10*)
	LD %M103.0
	MOVE %VW1500, %VW0
	MUL 10, %VW0
	MOVE %VD0, pv1
	MUL DI#2730, pv1
	(* Network 2 *)
	(*位置单位转换*)
	LD %M103.0
	MOVE %VR104, %VR108
	MUL 10.0, %VR108
	MOVE %VR108, %VR112
	MUL 10000.0, %VR112
	(* Network 3 *)
	(*位置单位转换*)
	LD %M103.0
	R_TO_DI %VR112, %VD116
	MOVE %VD116, tp1
	(* Network 4 *)
	(*工作模式给 1，控制字给 103f*)
	LD %M103.0
	MOVE B#1, mo1
	MOVE 16#103F, cw1
	(* Network 5 *)
	(*等待定位完成*)
	LD %SM0.0
	EQ cw1, 16#103F
	TON T101, 1
	AND %V1021.2
	MOVE 16#F, cw1

IL	显示部分：显示原点找到标志、定位完成标志、实际位置。
	(* Network 0 *)
	(*水平轴原点找到*)
	LD %V1021.7
	ST %M30.0
	(* Network 1 *)
	(*水平轴位置到*)
	LD %V1021.2
	ST %M30.1
	(* Network 2 *)
	(*实际位置显示*)
	LD %SM0.0
	DI_TO_R pa1, %VR200
	MOVE %VR200, %VR204
	DIV 10000.0, %VR204
	DIV 10.0, %VR204
	(* Network 3 *)
	LD %M666.0
	MOVE 16#86, cw1

IL	主程序 MAIN：调用各个子程序 (* Network 0 *) LD　　%SM0.0 CAL　　初始化 (* Network 1 *) LD　　%SM0.0 AND　　上电完成 CAL　　原点回归 (* Network 2 *) LD　　%SM0.0 AND　　上电完成 CAL　　速度模式 (* Network 3 *) LD　　%SM0.0 AND　　上电完成 CAL　　绝对定位 (* Network 4 *) LD　　%SM0.0 AND　　上电完成 CAL　　显示部分 (* Network 5 *) LD　　%SM0.0 CO_STATE 1, %M1000.0, %VB2000, %VW2002, %VB2004 CO_RESTART %M1000.1, 1, 1000, %M1000.2, %M1000.3, %VB2010

报文解析

1、PLC 第一次上电需要对伺服进行配置发送 NMT 管理报文使伺服进入预操作状态并发送 SDO 映射 PDO 里面的对象，发送的配置报文如下：

系统时间	时间标识	CAN通道	传输方向	ID号	帧类型	帧格式	长度	数据
18:03:23.	0x3AB480(	ch2	接收	0x0701	数据帧	标准帧	0x01	x 00
18:03:26.	0x3ABC20(	ch2	接收	0x0000	数据帧	标准帧	0x02	x 80 00
18:03:26.	0x3ABC26E	ch2	接收	0x0000	数据帧	标准帧	0x02	x 81 01
18:03:26.	0x3ABC273	ch2	接收	0x0701	数据帧	标准帧	0x01	x 00
18:03:26.	0x3ABC45E	ch2	接收	0x0000	数据帧	标准帧	0x02	x 80 01
18:03:26.	0x3ABC45E	ch2	接收	0x0601	数据帧	标准帧	0x08	x 2B 0C 10 00 E8 03 00 00
18:03:26.	0x3ABC462	ch2	接收	0x0581	数据帧	标准帧	0x08	x 60 0C 10 00 E8 03 00 00
18:03:26.	0x3ABC464	ch2	接收	0x0601	数据帧	标准帧	0x08	x 2F 0D 10 00 03 00 00 00
18:03:26.	0x3ABC46A	ch2	接收	0x0581	数据帧	标准帧	0x08	x 60 0D 10 00 03 00 00 00
18:03:26.	0x3ABC46(	ch2	接收	0x0601	数据帧	标准帧	0x08	x 23 00 18 01 81 01 00 80
18:03:26.	0x3ABC470	ch2	接收	0x0581	数据帧	标准帧	0x08	x 60 00 18 01 81 01 00 80
18:03:26.	0x3ABC472	ch2	接收	0x0601	数据帧	标准帧	0x08	x 2F 00 1A 00 00 00 00 00
18:03:26.	0x3ABC477	ch2	接收	0x0581	数据帧	标准帧	0x08	x 60 00 1A 00 00 00 00 00
18:03:26.	0x3ABC479	ch2	接收	0x0601	数据帧	标准帧	0x08	x 23 00 1A 01 20 00 63 60
18:03:26.	0x3ABC47E	ch2	接收	0x0581	数据帧	标准帧	0x08	x 60 00 1A 01 20 00 63 60
18:03:26.	0x3ABC480	ch2	接收	0x0601	数据帧	标准帧	0x08	x 2F 00 1A 00 01 00 00 00
18:03:26.	0x3ABC484	ch2	接收	0x0581	数据帧	标准帧	0x08	x 60 00 1A 00 01 00 00 00
18:03:26.	0x3ABC486	ch2	接收	0x0601	数据帧	标准帧	0x08	x 2F 00 18 02 FF 00 00 00

先看第一条红框的报文 0000 80 01，0x0000 是 NMT 管理报文的 COB-ID，数据 0x80 代表使节点进入预操作状态，数据 0x01 代表 1 号节点即从站站号为 1 的设备。

第二条红框的报文的 COB-ID 为 0x601 和 0x581，0x601 为 PLC 发送的 SDO 报文，0x581 为伺服应答的报文，截图中发送的 SDO 报文是为了配置 TPDO 和 RPDO，将 PLC 硬件配置中的 PDO 映射到伺服中。

第二条红框中的报文代表的含义是将 606300 实际位置这个对象映射到伺服中。下图为映射到伺服的 TPDO 和 RPDO。


TPDOSet

TPDO1 | TPDO2 | TPDO3 | TPDO4 | TPDO5 | TPDO6 | TPDO7 | TPDO8

N	Index	Type	Name	Value	Unit
0	1A0000	uint8	TPDO1映射组	1	DEC
1	1A0001	uint32	TPDO1映射1	60630020	HEX
2	1A0002	uint32	TPDO1映射2	00000000	HEX
3	1A0003	uint32	TPDO1映射3	00000000	HEX
4	1A0004	uint32	TPDO1映射4	00000000	HEX
5	1A0005	uint32	TPDO1映射5	00000000	HEX
6	1A0006	uint32	TPDO1映射6	00000000	HEX
7	1A0007	uint32	TPDO1映射7	00000000	HEX
8	1A0008	uint32	TPDO1映射8	00000000	HEX
9	180001	uint32	TPDO1站号	00000181	HEX
10	180002	uint8	TPDO1传输类型	255	DEC
11	180003	uint16	TPDO1禁止时间	10	DEC
12	180005	uint16	TPDO1事件时间	0	DEC


RPDOSet

RPDO1 | RPDO2 | RPDO3 | RPDO4 | RPDO5 | RPDO6 | RPDO7 | RPDO8

N	Index	Type	Name	Value	Unit
0	160000	uint8	RPDO1映射组	3	DEC
1	160001	uint32	RPDO1映射1	60400010	HEX
2	160002	uint32	RPDO1映射2	60600008	HEX
3	160003	uint32	RPDO1映射3	607A0020	HEX
4	160004	uint32	RPDO1映射4	00000000	HEX
5	160005	uint32	RPDO1映射5	00000000	HEX
6	160006	uint32	RPDO1映射6	00000000	HEX
7	160007	uint32	RPDO1映射7	00000000	HEX
8	160008	uint32	RPDO1映射8	00000000	HEX
9	140001	uint32	RPDO1站号	00000201	HEX
10	140002	uint8	RPDO1传输类型	255	DEC
11	140003	uint16	RPDO1禁止时间	0	DEC

2、完成以上配置后 PLC 发送 NMT 管理报文启动节点，启动节点后就可以在 PLC 程序中通过操作 PDO

来控制伺服了，启动节点之前的报文均为 PLC 在后台自动发送配置报文不需要用户进行任何操作。

0x3ABC52A	ch2	接收	0x0601	数据帧	标准帧	0x08	x	23 01 14 01 01 03 00 00
0x3ABC52E	ch2	接收	0x0581	数据帧	标准帧	0x08	x	60 01 14 01 01 03 00 00
0x3ABC950	ch2	接收	0x0000	数据帧	标准帧	0x02	x	01 01
0x3ABC956	ch2	接收	0x0181	数据帧	标准帧	0x04	x	FF FF FF FF
0x3ABC957	ch2	接收	0x0281	数据帧	标准帧	0x04	x	31 42 00 00
0x3ABC95F	ch2	接收	0x0000	数据帧	标准帧	0x02	x	01 00
0x3ABC962	ch2	接收	0x0181	数据帧	标准帧	0x04	x	00 00 00 00
0x3ABC964	ch2	接收	0x0281	数据帧	标准帧	0x04	x	31 42 00 00
0x3ABC974	ch2	接收	0x0201	数据帧	标准帧	0x07	x	00 00 00 00 00 00 00
0x3ABC97E	ch2	接收	0x0281	数据帧	标准帧	0x04	x	70 42 00 00
0x3ABC983	ch2	接收	0x0301	数据帧	标准帧	0x08	x	00 00 00 00 00 00 00 00
0x3ABC9CE	ch2	接收	0x0181	数据帧	标准帧	0x04	x	FF FF FF FF
0x3ABD561	ch2	接收	0x0181	数据帧	标准帧	0x04	x	00 00 00 00
0x137712	ch2	接收	0x0181	数据帧	标准帧	0x04	x	01 00 00 00
0x1378C9	ch2	接收	0x0181	数据帧	标准帧	0x04	x	FF FF FF FF
0x137937	ch2	接收	0x0181	数据帧	标准帧	0x04	x	01 00 00 00
0x137B4C	ch2	接收	0x0201	数据帧	标准帧	0x07	x	0F 00 03 00 00 00 00
0x137B4E	ch2	接收	0x0301	数据帧	标准帧	0x08	x	00 00 00 00 68 2A 04 00
0x137B52	ch2	接收	0x0281	数据帧	标准帧	0x02	x	37 52
0x137B5C	ch2	接收	0x0181	数据帧	标准帧	0x04	x	02 00 00 00
0x137B5E	ch2	接收	0x0281	数据帧	标准帧	0x02	x	37 56
0x137B84	ch2	接收	0x0281	数据帧	标准帧	0x02	x	37 52
0x137BC0	ch2	接收	0x0281	数据帧	标准帧	0x02	x	37 42
0x137BCA	ch2	接收	0x0181	数据帧	标准帧	0x04	x	1E 00 00 00

第一个红框中的报文 0000 01 01 为启动 1 号节点的意思，使设备进入操作状态。

第二个红框的报文的 COB-ID 为 0x181、0x281、0x201、0x301 为 PDO 的 COB-ID，这些报文里面的数据对应硬件配置里面 PDO 配置的对象，0x181、0x281 为伺服的发送 PDO，0x201、0x301 为伺服的接收 PDO，我们在 0x281 中配置的是状态字从报文中可以看到状态字的值为 0x4231，PDO 里面的数据需要倒着读。

第三个红框的报文为 PLC 发送给伺服的数据，0x201 里面配置的依次是控制字、工作模式、目标位置，0x301 里面配置的是梯形速度、目标速度，我们来看 0x201 里面的报文数据为 0F 00 03 00 00 00 00 00，这条报文为我在 PLC 程序里面触发速度控制子程序发送的报文，控制字占用一个字的长度对应 00 0F，工作模式占用一个字节的长度 03，目标位置占用两个字的长度 00 00 00 00 因为是速度控制所以目标位置没有给值。

3、节点保护报文

0x3ABEC8E	ch2	接收	0x0181	数据帧	标准帧	0x04	x	FE FF FF FF
0x3ABECFC	ch2	接收	0x0181	数据帧	标准帧	0x04	x	FF FF FF FF
0x3ABF04A	ch2	接收	0x0701	远程帧	标准帧	0x00		
0x3ABF04D	ch2	接收	0x0701	数据帧	标准帧	0x01	x	05
0x3ABF5FF	ch2	接收	0x0181	数据帧	标准帧	0x04	x	FD FF FF FF
0x3ABF66D	ch2	接收	0x0181	数据帧	标准帧	0x04	x	FF FF FF FF
0x3ABFA49	ch2	接收	0x0181	数据帧	标准帧	0x04	x	FD FF FF FF
0x3ABFAB7	ch2	接收	0x0181	数据帧	标准帧	0x04	x	FF FF FF FF
0x3AC0127	ch2	接收	0x0181	数据帧	标准帧	0x04	x	00 00 00 00
0x3AC0195	ch2	接收	0x0181	数据帧	标准帧	0x04	x	FF FF FF FF
0x3AC0428	ch2	接收	0x0181	数据帧	标准帧	0x04	x	FD FF FF FF
0x3AC0496	ch2	接收	0x0181	数据帧	标准帧	0x04	x	FF FF FF FF
0x3AC0873	ch2	接收	0x0181	数据帧	标准帧	0x04	x	00 00 00 00
0x3AC08E1	ch2	接收	0x0181	数据帧	标准帧	0x04	x	FF FF FF FF
0x3AC094F	ch2	接收	0x0181	数据帧	标准帧	0x04	x	FE FF FF FF
0x3AC09BC	ch2	接收	0x0181	数据帧	标准帧	0x04	x	FF FF FF FF
0x3AC0B06	ch2	接收	0x0181	数据帧	标准帧	0x04	x	00 00 00 00
0x3AC0B74	ch2	接收	0x0181	数据帧	标准帧	0x04	x	FF FF FF FF
0x3AC109A	ch2	接收	0x0181	数据帧	标准帧	0x04	x	00 00 00 00
0x3AC1108	ch2	接收	0x0181	数据帧	标准帧	0x04	x	FF FF FF FF
0x3AC14E5	ch2	接收	0x0181	数据帧	标准帧	0x04	x	00 00 00 00
0x3AC15C1	ch2	接收	0x0181	数据帧	标准帧	0x04	x	FF FF FF FF
0x3AC1744	ch2	接收	0x0701	远程帧	标准帧	0x00		
0x3AC1746	ch2	接收	0x0701	数据帧	标准帧	0x01	x	85
0x3AC1778	ch2	接收	0x0181	数据帧	标准帧	0x04	x	00 00 00 00

主站根据设定的监督时间和寿命因子周期性发送 COB-ID 为 0x701 的远程帧，正常运行时伺服循环回复状态 05、85。

4、紧急报文

16:03:21.0	0x11BA7F2	ch2	接收	0x0081	数据帧	标准帧	0x08	x 31 73 20 00 02 40 00 00
16:03:21.0	0x11BA7F5	ch2	接收	0x0281	数据帧	标准帧	0x02	x 38 02
16:03:21.0	0x11BA7F6	ch2	接收	0x0000	数据帧	标准帧	0x02	x 02 01
16:03:21.0	0x11BA7F9	ch2	接收	0x0081	数据帧	标准帧	0x08	x 31 73 20 00 02 40 00 00
16:03:21.0	0x11BA7FE	ch2	接收	0x0000	数据帧	标准帧	0x02	x 02 01

应急报文说明

当设备内部出现致命错误将触发应急报文，由应用设备以最高优先级发送到其他设备。一条应急报文由 8 字节组成。

表 10-22 应急报文格式

COB-ID	Byte 0-1	Byte2	Byte4-5	Byte6-7
紧急报文站号 0x101400	应急错误代码 0x603F00	错误寄存器(0x100100)	错误状态 0x260100	错误状态 0x260200

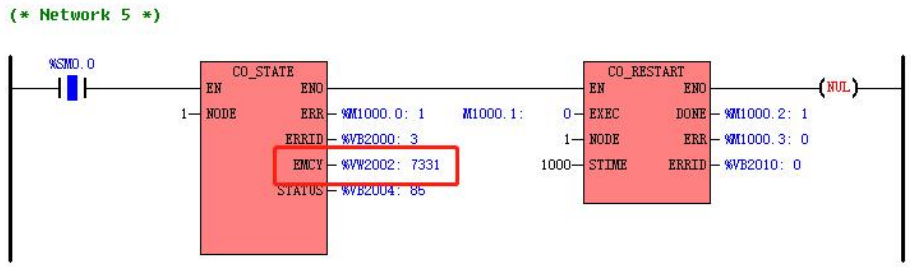
表 10-23 应急错误代码 0x603F00

报警内容	应急错误代码(Hex)	报警内容	应急错误代码(Hex)
通讯式编码器没有连接	0x7331	电流传感器故障	0x5210
通讯式编码器多圈错误	0x7320	软件看门狗复位	0x6010
通讯式编码器校验错误	0x7330	异常中断	0x6011
驱动器温度过高	0x4210	MCU 故障	0x7400
驱动器总线电压过高	0x3210	电机型号配置错误	0x6320
驱动器总线电压过低	0x3220	电机动力线缺相	0x6321
驱动器功率部分短路或电机短路	0x2320	预使能报警	0x5443
电流采样饱和	0x2321	正限位报错	0x5442
驱动器制动电阻异常	0x7110	负限位报错	0x5441
实际跟随误差超过允许	0x8611	SPI 故障	0x6012
逻辑电压低	0x5112	总线通讯错误	0x8100
电机或驱动器过载	0x2350	总线通讯超时	0x81FF
输入脉冲频率过高	0x8A80	全闭环检查错误	0x8A81

第一个图为拔掉伺服编码器线抓取到的报文，0x81 对应的 1 号从站的紧急报文的 COB-ID，数据为 31 73 20 00 02 40 00 00。

第二个图为 kinco 伺服紧急报文的说明（各个厂家对紧急报文的定义不完全相同请以实际应用为准），由数据可知紧急报文的数据 0x7331 对应报警内容通信式编码器没有连接。

通过下图的 CO_STATE 指令可以监控曾经发生过的紧急报文代码和节点保护状态。另外，可以通过 CO_RESTART 指令可以重新启动节点，适用于因报警、通信中断、主站先于从站上电等情况从站处于非操作状态，可以使用此指令重启节点使从站重新进入操作状态。



注意：关于报文解析的讲解需结合前面章节介绍的 CANopen 主站功能中的 NMT、PDO、SDO、节点保护等内容去理解。

10.5.6 CANOpen 从站功能

10.5.6.1 概述

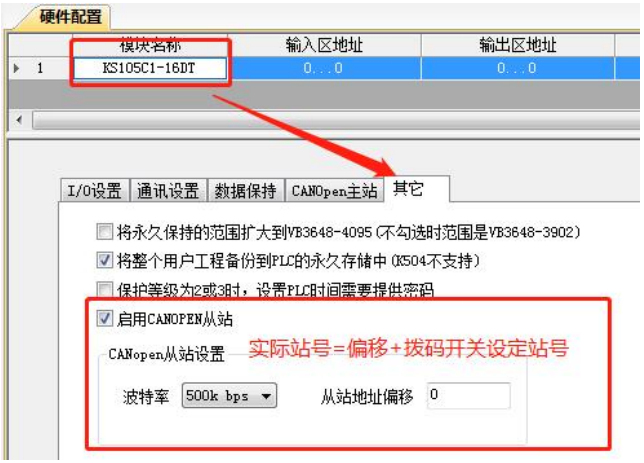
KS105C1-16DT 支持 CANOpen 从站功能，具有如下特点：

- CAN 接口采用 CAN2.0A 标准。
- 符合 DS301 V4.2.0 和 DS405 V2.0 协议
- 支持 NMT 网络管理服务，支持心跳协议。
- 支持 SDO Server，加速传输模式。
- 支持最多 16 个 TPDO 和 16 个 RPDO，PDO 的通信参数和映射参数均可自由配置。

注意：使用时，PDO的序号必须连续，比如，使用TPDO1、TPDO2、TPDO3 是合法的，但是若配置为使用TPDO1、TPDO3、TPDO4则是非法的。

10.5.6.2 启用 CANOpen 从站功能

在用户工程中，进入【PLC 硬件配置】，在 CPU 模块的【其它】页面中，用户可以配置 CANOpen 从站功能。



【启用 CANOPEN 从站】：若选中此项，则本 PLC 将作为 CANOpen 从站。

【波特率】：选择 CAN 接口所用的波特率。一个网络中所有节点的波特率都必须一致。

【从站地址偏移】：本从站的真实站号等于“第 1 至 3 位拨码开关的组合值”加上“从站地址偏移”。例如，假如第 1 到 3 位拨码开关的组合值是 5，从站地址偏移设置为 8，则本从站站号就是 13。

10.5.6.3 对象字典(EDS 文件)

用户可以到步科公司的官网免费下载本从站设备的 eds 文件。

在 KincoBuilder 软件中已经集成了本设备的 EDS 文件：在硬件配置的【CANOpen 主站】->【网络配置】页面中，本设备就显示在“所有从站模块列表”的“运动控制器和 PLC”组中，名字显示为“Kinco KPLC for Kinco”。用户若使用 Kinco PLC 作为主站及从站，那么无需再次导入 eds 文件。

本设备暂时开放了如下内存区域作为 CANOpen 通信对象，可以映射到 PDO 中使用。若需要使用更大的内存区域，请联系步科公司提供另外的 eds 文件即可。

对象索引	数据类型	PLC 内存区域	数量	读写属性
0xA040	BYTE	IB0～IB9	10	只读
0xA0C0	INT16	AIW0～AIW18	10	只读
0xA4C0	BYTE	QB0～QB9	10	可读可写
0xA4C1	BYTE	MB0～MB9	10	可读可写
0xA4D2	BYTE	VB508～VB517	10	可读可写

0xA540	INT16	AQW0～AQW18	10	可读可写
0xA550	INT16	VW0～VW18	10	可读可写
0xA641	INT32	VD1016～VD1052	10	可读可写
0xA6C2	FLOAT	VR2032～VR2068	10	可读可写

10.5.7 CAN 自由通信功能

10.5.7.1 概述

KPLC 里提供了一组 CAN 通信指令，可以初始化 CAN 口、通过 CAN 口收发数据等，用户可以使用这些指令来自由编写通信程序与其它设备进行通信。CAN 自由通信功能可以与扩展总线功能和 CANOpen 主站功能同时使用。

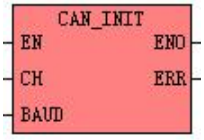
CAN 通信指令支持 CAN2.0A 和 CAN2.0B 标准，另外这些指令只支持数据帧，不支持远程帧。CAN 数据帧格式如下：

ID	字节 1-8
11 位（CAN2.0A，标准帧）或 29 位（CAN2.0B，扩展帧）	1-8 字节长度的数据

10.5.7.2 CAN 自由通信指令

10.5.7.2.1 CAN_INIT（初始化 CAN 接口）

- 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	CAN_INIT			☑ KS ☑ KW ☑ K209M ☑ K6
IL	CAN_INIT	CAN_INIT CH, BAUD	U	

参数	输入/输出	数据类型	允许的内存区
EN	输入	BOOL	I、Q、V、M、L、SM
CH	输入	INT	常量
BAUD	输入	INT	L、M、V、常量
ERR	输出	BOOL	L、M、V、常量

参数	描　述
EN	使能端。
CH	所用的 CAN 接口。0 表示 CAN1，1 表示 CAN2，2 表示 K541 模块
BAUD	CAN 的波特率。 8 --- 1000K 7 --- 800K 6 --- 500K 5 --- 250K 4 --- 125K 3 --- 50K 2 --- 20K 1--- 10K
ERR	指令执行是否成功。0 表示成功，1 表示有错误（比如参数错误）

*EN*输入端的上升沿跳变会触发执行该指令，用于初始化指定的 CAN 接口（*CH*），并将 CAN 波特率设置为 BAUD 代表的数值。

- LD

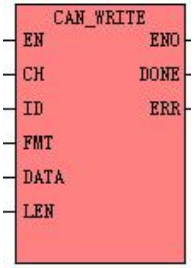
  *EN*的上升沿会触发该指令执行一次，否则不执行。

- IL

  CR 的上升沿会触发该指令执行一次，否则不执行。该指令的执行不影响 CR 值。

10.5.7.2.2 CAN_WRITE（发送一次 CAN 报文）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	CAN_WRITE			☑ KS ☑ KW ☑ K209M ☑ K6
IL	CAN_WRITE	CAN_WRITE CH, ID, FMT, DATA, LEN, DONE, ERR	U	

参数	输入/输出	数据类型	允许的内存区
CH	输入	INT	常量
ID	输入	DWORD	L、M、V、常量
FMT	输入	BYTE	L、M、V、常量
DATA	输入	BYTE	L、M、V
LEN	输入	BYTE	L、M、V、常量
DONE	输出	BOOL	L、M、V
ERR	输出	BOOL	L、M、V

参数	描　述
EN	使能端。
CH	所用的 CAN 接口。0 表示 CAN1，1 表示 CAN2，2 表示 K541 模块
FMT	报文格式。0 表示标准帧，1 表示扩展帧。
DATA	待发送数据存放的起始字节地址。

LEN	待发送数据的长度。单位：字节。
DONE	报文是否发送完成。在执行时 DONE 被置 0，发送完成则 DONE 被置 1。
ERR	报文发送是否出错。若发送失败（通常由于发送缓冲区满），则 ERR 被置 1。

注意：ID, FMT, LEN, 必须同时为常量类型或同时为内存类型。DATA 与 LEN 参数共同组成了一个一个可变长度的内存块，整个内存块内的地址都必须是合法地址。

待发送的 CAN 报文由 *ID* 号、格式（*FMT*，指明扩展帧或者标准帧）、数据（*DATA*，指明报文数据存放的起始地址)、长度 *LEN* 来指定。

EN 输入端的上升沿跳变会触发执行一次该指令，将待发送的报文写入 PLC 内部的发送缓冲区中，再由 PLC 调度通过指定的 CAN 接口 *CH* 发送出去。 若指令成功将报文发送出去，则将 *DONE* 置为 1，*ERR* 置为 0。若发送缓冲区已满或者报文发送失败，则同时将 *DONE* 和 *ERR* 置为 1。

在一个工程中，CAN_TX 指令和 CAN_WRITE 指令的总条数最多允许 64 条！

- LD

  EN 的上升沿会触发该指令执行一次，否则不执行。

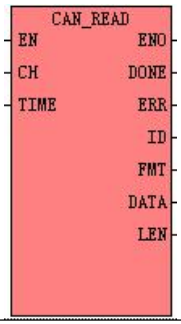
- IL

  CR 的上升沿会触发该指令执行一次，否则不执行。

  该指令的执行不影响 CR 值。

10.5.7.2.3 CAN_READ（接收一次 CAN 报文）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	CAN_READ			☑ KS ☑ KW ☑ K209M ☑ K6
IL	CAN_READ	CAN_READ CH, TIME, DONE, ERR, FMT, DATA, LEN	U	

参数	输入/输出	数据类型	允许的内存区
CH	输入	INT	常量
TIME	输入	INT	L、M、V、常量
DONE	输出	BOOL	L、M、V
ERR	输出	BOOL	L、M、V
ID	输出	DWORD	L、M、V
FMT	输出	BYTE	L、M、V
DATA	输出	BYTE	M、V
LEN	输出	BYTE	L、M、V

参数	描　述
EN	使能端。
CH	所用的 CAN 接口。0 表示 CAN1，1 表示 CAN2，2 表示 K541 模块

TIME	超时时间。启动接收后，若在指定的这个时间内没有收到任何报文，则超时退出接收状态，并将 ERR 置 1
ID	接收到的报文的 ID 号。
FMT	接收到的报文格式。0 表示标准帧，1 表示扩展帧。
DATA	接收到的报文数据存放的起始字节地址。
LEN	接收到的报文数据长度。单位：字节。
DONE	是否接收完成。在执行时 DONE 被置 0，发送完成则 DONE 被置 1。
ERR	接收报文是否出错。若接收失败则 ERR 被置 1。

注意：DATA 与 LEN 参数共同组成了一个一个可变长度的内存块，整个内存块内的地址都必须是合法地址。

*EN*输入端的上升沿跳变会触发执行该指令：启动接收状态，接收任何一条来自指定 CAN 接口 *CH* 的报文。

启动接收后，若在指定的超时时间 *TIME* 内接收到一条 CAN 报文，那么 PLC 就将报文相关数据分别置于输出参数 *ID*（ID 号）、*FMT*（格式，指明接收到的是扩展帧或者标准帧）、*DATA*（接收到的报文数据存放的起始地址)、LEN（长度）中，同时将 DONE 置为 1，退出接收。若在指定的超时时间 *TIME*）内没有收到任何报文，则超时退出接收状态，并将 *DONE* 和 ERR 置 1。

注意：1） CAN_READ 指令的优先级低于 CAN_RX 指令。

2）当 CAN_READ 指令启动之后，总线上的任何一条报文都会被其接收，因此若程序中调用了多条 CAN_READ 指令，那么最后启动的指令将会生效。

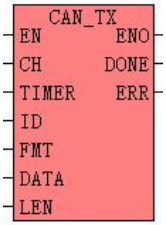
- LD

  *EN*的上升沿会触发该指令执行一次，否则不执行。

- IL

  CR 的上升沿会触发该指令执行一次，否则不执行。该指令的执行不影响 CR 值。

10.5.7.2.4 CAN_TX（自动发送 CAN 报文）

	名称	指令格式	适用产品
LD	CAN_TX		☑ KS ☑ KW209M ☑ K6

参数	输入/输出	数据类型	允许的内存区
CH	输入	INT	常量
TIMER	输入	INT	L、M、V、常量
ID	输入	DWORD	L、M、V、常量
FMT	输入	INT	L、M、V、常量
DATA	输出	BYTE	M、V
LEN	输出	BYTE	L、M、V
DONE	输出	BOOL	L、M、V
ERR	输出	BOOL	L、M、V

参数	描　述
EN	使能端。
CH	所用的 CAN 接口。0 表示 CAN1，1 表示 CAN2，2 表示 K541 模块
TIMER	待发送报文定时发送的周期，单位 ms。0 表示不启用定时发送功能。
ID	待发送报文的 ID
FMT	待发送报文的格式。0 表示标准帧，1 表示扩展帧。
DATA	待发送报文数据存放的首地址。
LEN	待发送报文的数据长度，单位：字节。
DONE	发送完成标志位。每次发送成功后，DONE 自动置 1 并维持至少一个扫描周期的时间。
ERR	发送错误标志位。1 表示本次发送失败。

注意：ID，FMT，TIMER 和 LEN 参数必须同时为常量或同时为变量；DATA 与 LEN 参数组成了一个长度可变的内存块，此内存块必须全部位于合法的内存区域，否则结果不可预期。

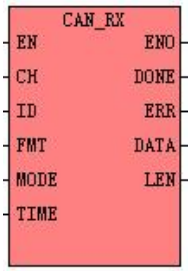
PLC 内部维护着一个报文自动发送列表，当表中的报文满足发送条件后，PLC 会将此报文自动发送出去。某报文自动发送的条件为：若报文中的数据发生了变化，则立即发送一次；若设置的定时发送周期时间到了，则立即发送一次。报文发送完成后，输出参数 *DONE* 自动置 1 并维持一个扫描后自动变为 0，若发送失败（原因是发送缓冲区已满或者报文发送失败），则输出参数 *ERR* 自动置 1。该发送报文列表最大长度为 64 条。

CAN_TX 指令用于向该发送报文列表中添加一条报文，报文由报文 *ID* 号、格式（*FMT*，指明扩展帧或者标准帧）、数据（*DATA*，指明报文数据存放的起始地址）、长度 *LEN* 来指定。*TIMER* 参数值指明了定时发送的周期（ms），若 *TIMER* 值为 0，则表示不会定时发送。

EN 输入端的上升沿跳变会触发执行该指令。本指令执行后，则 PLC 立即将该指令指定的报文加入到自动发送列表中。**因此在一个工程中，一条 CAN_TX 指令仅需要执行一次即可，无需多次执行。另外在一个工程中，CAN_TX 指令和 CAN_WRITE 指令的总条数最多允许 64 条！**

10.5.7.2.5 CAN_RX（自动接收特定 ID 的 CAN 报文）

➢ 指令及其操作数说明

	名称	指令格式	影响CR值	
LD	CAN_RX			☑ KS ☑ KW ☑ K209M ☑ K6
IL	CAN_RX	CAN_RX CH, ID, FMT, MODE, TIME, DONE, ERR, FMT, DATA, LEN	U	

参数	输入/输出	数据类型	允许的内存区
CH	输入	INT	常量
ID	输入	DWORD	L、M、V、常量
FMT	输入	INT	L、M、V、常量
MODE	输入	INT	L、M、V、常量
TIME	输入	INT	L、M、V、常量
DONE	输出	BOOL	L、M、V
ERR	输出	BOOL	L、M、V
DATA	输出	BYTE	M、V
LEN	输出	BYTE	L、M、V

参数	描　述
EN	使能端。
CH	所用的 CAN 接口。0 表示 CAN1，1 表示 CAN2，2 表示 K541 模块
ID	待接收报文的 ID
FMT	待接收报文的格式。0 表示标准帧，1 表示扩展帧。

MODE	接收模式。0 表示永久接收模式，1 表示单次接收模式
TIME	接收超时时间。单位 ms
DONE	在单次接收模式下，DONE 是接收成功标志位。
ERR	接收超时标志位。
DATA	最近一次接收的报文数据存放的首地址。
LEN	最近一次接收的报文的数据长度，单位：字节。

注意：ID，FMT，MODE 和 TIME 参数必须同时为常量或同时为变量；DATA 与 LEN 参数组成了一个长度可变的内存块，此内存块必须全部位于合法的内存区域，否则结果不可预期。

PLC 内部自动维护着一个接收报文过滤列表。CPU 会对接收的的 CAN 报文进行过滤，只有报文 ID 和格式（标准帧或扩展帧）符合列表值的报文才会被本指令接收下来。

CAN_RX 指令用于向该接收报文过滤列表中添加一条等待接收的报文，报文由指定的 ID 号（*ID*，CAN 报文的 ID）和格式（*FMT*，指明扩展帧或者标准帧）来确定。

MODE 参数指明了接收方式，若 *MODE* 为 1，则为单次接收模式，指令只接收一次指定报文，收到后则退出；若 *MODE* 为 0，则为永久接收模式，指令会一直接收指定报文。

EN 输入端的上升沿跳变会触发执行该指令。本指令执行后，指定的 ID 号（*ID*）和格式（*FMT*）值会立即加入到接收过滤列表中，并且 PLC 立即进入接收状态，同时将 *DONE*、*ERR* 清 0。若为单次接收模式，那么如果在时间 *TIME* 内接收到指定报文，则将 *DONE* 置 1，指令退出接收状态，如果在时间 *TIME* 内没有收到指定报文，那么 *DONE* 和 *ERR* 被置 1，指令退出接收状态；若为永久接收模式，那么本指令启动后就会一直监视 CAN 接口 *CH* 并接收所有的指定报文，若在一次成功接收之后，在 *TIME* 内没有再次接收到指定报文，则将 ERR 置 1，之后若再次成功接收，那么就会将 ERR 清 0。

在一个工程中，一条 CAN_RX 指令仅需要执行一次即可。另外在一个工程中，调用的 CAN_RX 指令最多允许 64 条！

- LD

EN 的上升沿会触发执行该指令并进入接收状态，否则不执行。

10.5.7.2.6 使用举例

下面将举例说明 CAN 自由通信功能的使用。

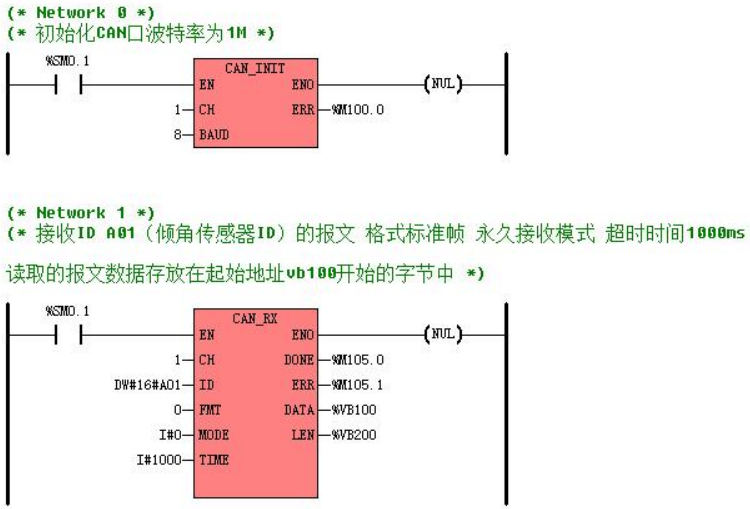
一、PLC 和倾角传感器进行 CAN 自由通信读取倾斜角度值的示例

1、上电初始化，使用 CAN_INIT 指令初始化 CAN 口设置波特率。

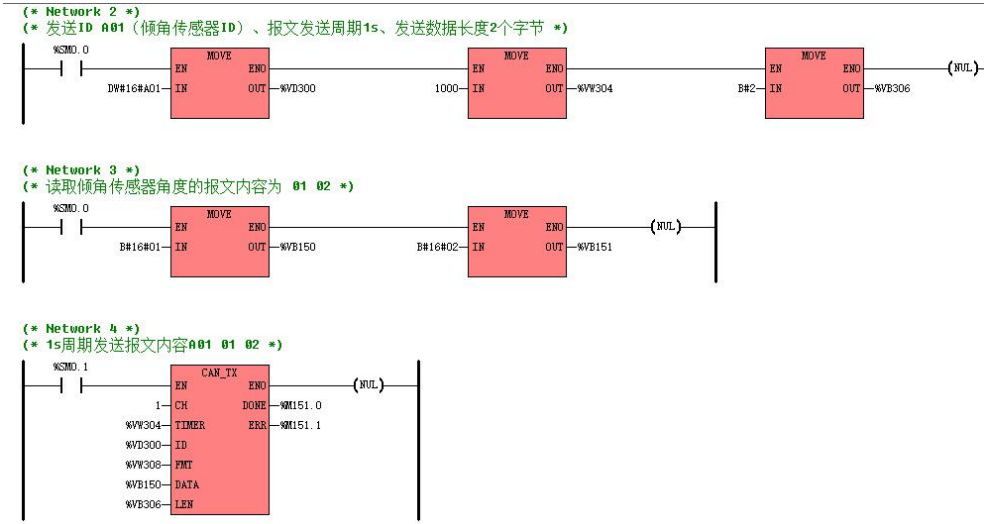
2、设置 CAN_RX 指令参数，指定接收报文的 ID 为 0xA01（倾角传感器 ID），进入永久接收模式，设定超时时间等，CPU 正常工作时会一直接收 ID 为 0xA01 的报文，报文的数据信息存放到起始地址 vb100 开始的字节中，用户可以根据倾角传感器的自定义协议将接收到的数据信息进行解析。

3、设置 CAN_TX 指令参数，发送周期 1s、发送报文 ID 为 0xA01（倾角传感器 ID），发送数据长度 2 个字节，发送报文内容 01 02（代表含义即读取倾角传感器角度值），即每隔 1s 发送一次读取倾角传感器的角度值的报文。

LD



LD


(* Network 2 *)
(* 发送ID A01（倾角传感器ID）、报文发送周期1s、发送数据长度2个字节 *)
%SM0.0
MOVE
EN
ENO
DW#16#A01
IN
OUT
%VD300
1000
%VW304
B#2
%VB306
(NUL)
(* Network 3 *)
(* 读取倾角传感器角度的报文内容为 01 02 *)
B#16#01
%VB150
B#16#02
%VB151
(* Network 4 *)
(* 1s周期发送报文内容A01 01 02 *)
%SM0.1
CAN_TX
1
CH
DONE
%M151.0
ERR
%M151.1
%VW304
TIMER
%VD300
ID
%VW308
FMT
%VB150
DATA
%VB306
LEN

```
IL
(* Network 0 *)
(*初始化 CAN 口波特率为 1M*)
LD      %SM0.1
CAN_INIT 1, 8, %M100.0
(* Network 1 *)
(*接收 ID A01 报文 超时时间 1000ms*)
LD      %SM0.1
CAN_RX  1, DW#16#A01, 0, I#0, I#1000, %M105.0, %M105.1, %VB100, %VB200
(* Network 2 *)
(*发送 ID 确认、发送周期 1s、数据长度 2 个字节*)
LD      %SM0.0
MOVE    DW#16#A01, %VD300
MOVE    1000, %VW304
MOVE    B#2, %VB306
(* Network 3 *)
(*读取倾角传感器角度报文内容为 01 02*)
LD      %SM0.0
MOVE    B#16#01, %VB150
MOVE    B#16#02, %VB151
(* Network 4 *)
(*1s 周期发送报文内容 A01 01 02*)
LD      %SM0.1
CAN_TX  1, %VW304, %VD300, %VW308, %VB150, %VB306, %M151.0, %M151.1
```

二、KS101M 和 KS105C2 二者通过 CAN 自由通信组网的示例

1、KS101M 中首先对 CAN1 口进行初始化设置波特率设置 1M，上电后进入永久接收模式接收指定 ID201、202 的报文（KS105C2 发送的），并发送指定 ID101、102 的报文，ID101 报文中数据包含 8 个字节长度即两个双字 VD100、VD104，ID102 报文中数据包含 4 个字节长度即一个双字 VD120，其中 VD100 自加 1、VD104 自加 2、VD120 自加 3。通过 USB 转 CAN 工具监控到的报文如下：

序号	系统时间	时间标识	CAN通道	传输方向	ID号	帧类型	帧格式	长度	数据
00000	09:29:05.672	0x505CF	ch2	接收	0x0101	数据帧	标准帧	0x08	x 01 00 00 00 02 00 00 00
00001	09:29:05.672	0x505D0	ch2	接收	0x0102	数据帧	标准帧	0x04	x 03 00 00 00
00002	09:29:06.513	0x525DD	ch2	接收	0x0101	数据帧	标准帧	0x08	x 02 00 00 00 04 00 00 00
00003	09:29:06.513	0x525DE	ch2	接收	0x0102	数据帧	标准帧	0x04	x 06 00 00 00
00004	09:29:07.502	0x54CD4	ch2	接收	0x0101	数据帧	标准帧	0x08	x 02 00 00 00 04 00 00 00
00005	09:29:07.502	0x54CD5	ch2	接收	0x0102	数据帧	标准帧	0x04	x 06 00 00 00
00006	09:29:07.502	0x54CE2	ch2	接收	0x0101	数据帧	标准帧	0x08	x 03 00 00 00 06 00 00 00
00007	09:29:07.502	0x54CE3	ch2	接收	0x0102	数据帧	标准帧	0x04	x 09 00 00 00
00008	09:29:08.493	0x573D6	ch2	接收	0x0101	数据帧	标准帧	0x08	x 03 00 00 00 06 00 00 00
00009	09:29:08.493	0x573D7	ch2	接收	0x0102	数据帧	标准帧	0x04	x 09 00 00 00
00010	09:29:08.493	0x573E4	ch2	接收	0x0101	数据帧	标准帧	0x08	x 04 00 00 00 08 00 00 00
00011	09:29:08.493	0x573E5	ch2	接收	0x0102	数据帧	标准帧	0x04	x 0C 00 00 00
00012	09:29:09.511	0x59ADB	ch2	接收	0x0101	数据帧	标准帧	0x08	x 04 00 00 00 08 00 00 00
00013	09:29:09.511	0x59ADC	ch2	接收	0x0102	数据帧	标准帧	0x04	x 0C 00 00 00
00014	09:29:09.511	0x59AE7	ch2	接收	0x0101	数据帧	标准帧	0x08	x 05 00 00 00 0A 00 00 00

通过报文可以清晰看到指定 ID 号报文中数据的变化。

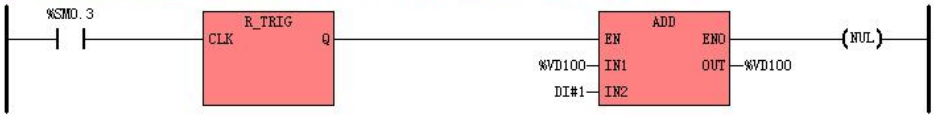
2、KS101M 的程序如下：

LD

```
(* Network 0 *)
(* 初始化can1口，波特率1M *)
%SM0.1 ──┤ ├──┬─ CAN_INIT ─┬──(NUL)
                │ EN      ENO │
              0─┤ CH      ERR ├─%M0.0
              8─┤ BAUD        │
                └─────────────┘

(* Network 1 *)
(* plc上电后进入永久接收模式，后台自动接收指定ID为201、202的标准帧报文，
不需要用户再重新触发此指令 *)
%SM0.1 ──┤ ├──┬─ CAN_RX ──┬─ CAN_RX ──┬──(NUL)
                │ EN    ENO │ EN    ENO │
              0─┤ CH    DONE├─%M0.1   0─┤ CH    DONE├─%M0.3
DW#16#201─┤ ID    ERR ├─%M0.2 DW#16#202─┤ ID    ERR ├─%M0.4
              0─┤ FMT   DATA├─%VB0    0─┤ FMT   DATA├─%VB20
              0─┤ MODE  LEN ├─%VB10   0─┤ MODE  LEN ├─%VB30
           1000─┤ TIME      │       1000─┤ TIME      │
                └───────────┘            └───────────┘
```

LD

(* Network 3 *)
(* vd100和vd104是发送ID101中的8个字节长度，二者分别自加1和自加2 *)



(* Network 4 *)

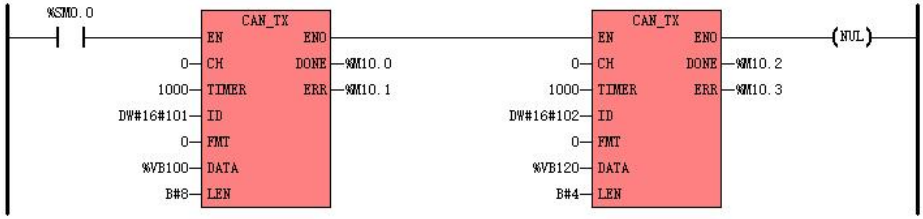
(* Network 5 *)
(* vd120是发送ID102中的4个字节长度，自加3 *)

(* Network 6 *)
(* 以1s周期发送指定ID101、102的标准帧报文

plc后台会自动对需要发送的报文按着ID大小的优先级进行排队发送，ID越小优先级越高

不需要用户再进行其他处理，用户只需要设置好合理的发送周期即可

避免出现周期太短，需要发送的报文太多，造成总线拥堵的情况 *)



IL

```
(* Network 0 *)
(*初始化 can1 口，波特率 1M*)
LD          %SM0.1
CAN_INIT 0, 8, %M0.0
(* Network 1 *)
(*plc 上电后进入永久接收模式，后台自动接收指定 ID 为 201、202 的标准帧报文，不需要用户再重新触发此指令*)
LD          %SM0.1
CAN_RX    0, DW#16#201, 0, 0, 1000, %M0.1, %M0.2, %VB0, %VB10
CAN_RX    0, DW#16#202, 0, 0, 1000, %M0.3, %M0.4, %VB20, %VB30
(* Network 2 *)
LD          %SM0.0
ST          %BR0.0
(* Network 3 *)
(*vd100 和 vd104 是发送 ID101 中的 8 个字节长度，二者分别自加 1 和自加 2*)
LD          %SM0.3
R_TRIG
ADD         DI#1, %VD100
(* Network 4 *)
LD          %SM0.3
R_TRIG
ADD         DI#2, %VD104
(* Network 5 *)
(*vd120 是发送 ID102 中的 4 个字节长度，自加 3*)
LD          %SM0.3
R_TRIG
ADD         DI#3, %VD120
```

IL	(* Network 6 *) (*以 1s 周期发送指定 ID101、102 的标准帧报文 plc 后台会自动对需要发送的报文按着 ID 大小的优先级进行排队发送，ID 越小优先级越高不需要用户再进行其他处理，用户只需要设置好合理的发送周期即可避免出现周期太短，需要发送的报文太多，造成总线拥堵的情况*) LD %SM0.0 CAN_TX 0, 1000, DW#16#101, 0, %VB100, B#8, %M10.0, %M10.1 CAN_TX 0, 1000, DW#16#102, 0, %VB120, B#4, %M10.2, %M10.3

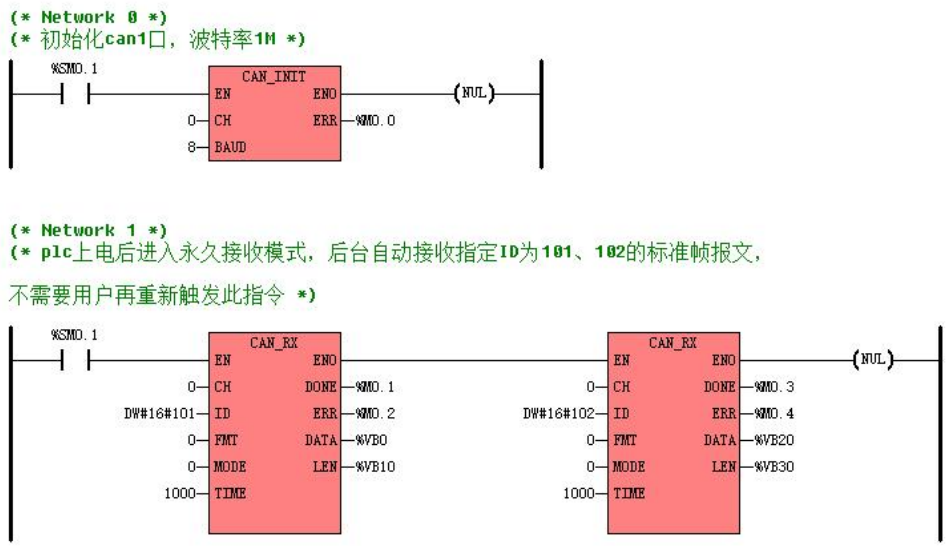
3、KS105C2 中首先对 CAN1 口进行初始化设置波特率设置 1M，上电后进入永久接收模式接收指定 ID101、102 的报文（KS101M 发送的），并发送到指定 ID201、202 的报文，ID201 报文中数据包含 8 个字节长度即两个双字 VD100、VD104，ID202 报文中数据包含 4 个字节长度即一个双字 VD120，其中 VD100 自加 1、VD104 自加 2、VD120 自加 3。通过 USB 转 CAN 工具监控到的报文如下：

序号	系统时间	时间标识	CAN通道	传输方向	ID号	帧类型	帧格式	长度	数据
00000	09:46:10.143	0xE3402	ch2	接收	0x0201	数据帧	标准帧	0x08	x 01 00 00 00 02 00 00 00
00001	09:46:10.143	0xE3403	ch2	接收	0x0202	数据帧	标准帧	0x04	x 03 00 00 00
00002	09:46:11.133	0xE5AF7	ch2	接收	0x0201	数据帧	标准帧	0x08	x 01 00 00 00 02 00 00 00
00003	09:46:11.133	0xE5AF8	ch2	接收	0x0202	数据帧	标准帧	0x04	x 03 00 00 00
00004	09:46:11.133	0xE5B07	ch2	接收	0x0201	数据帧	标准帧	0x08	x 02 00 00 00 04 00 00 00
00005	09:46:11.133	0xE5B08	ch2	接收	0x0202	数据帧	标准帧	0x04	x 06 00 00 00
00006	09:46:12.121	0xE81F9	ch2	接收	0x0201	数据帧	标准帧	0x08	x 02 00 00 00 04 00 00 00
00007	09:46:12.121	0xE81FA	ch2	接收	0x0202	数据帧	标准帧	0x04	x 06 00 00 00
00008	09:46:12.121	0xE8209	ch2	接收	0x0201	数据帧	标准帧	0x08	x 03 00 00 00 06 00 00 00
00009	09:46:12.121	0xE820A	ch2	接收	0x0202	数据帧	标准帧	0x04	x 09 00 00 00
00010	09:46:13.144	0xEA8FD	ch2	接收	0x0201	数据帧	标准帧	0x08	x 03 00 00 00 06 00 00 00
00011	09:46:13.144	0xEA8FE	ch2	接收	0x0202	数据帧	标准帧	0x04	x 09 00 00 00
00012	09:46:13.144	0xEA909	ch2	接收	0x0201	数据帧	标准帧	0x08	x 04 00 00 00 08 00 00 00
00013	09:46:13.144	0xEA90A	ch2	接收	0x0202	数据帧	标准帧	0x04	x 0C 00 00 00
00014	09:46:14.132	0xED002	ch2	接收	0x0201	数据帧	标准帧	0x08	x 04 00 00 00 08 00 00 00

通过报文可以清晰看到指定 ID 号报文中数据的变化。

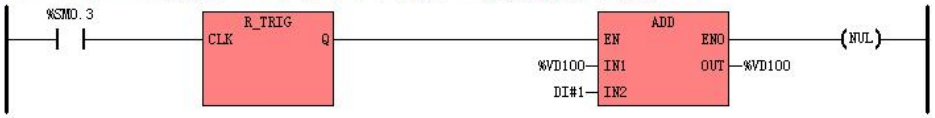
4、KS105C2 的程序如下：

LD


(* Network 0 *)
(* 初始化can1口，波特率1M *)
%SM0.1
CAN_INIT
EN
ENO
NUL
0
CH
ERR
%M0.0
8
BAUD
(* Network 1 *)
(* plc上电后进入永久接收模式，后台自动接收指定ID为101、102的标准帧报文，
不需要用户再重新触发此指令 *)
%SM0.1
CAN_RX
EN
ENO
0
CH
DONE
%M0.1
DW#16#101
ID
ERR
%M0.2
0
FMT
DATA
%VB0
0
MODE
LEN
%VB10
1000
TIME
CAN_RX
EN
ENO
NUL
0
CH
DONE
%M0.3
DW#16#102
ID
ERR
%M0.4
0
FMT
DATA
%VB20
0
MODE
LEN
%VB30
1000
TIME

LD

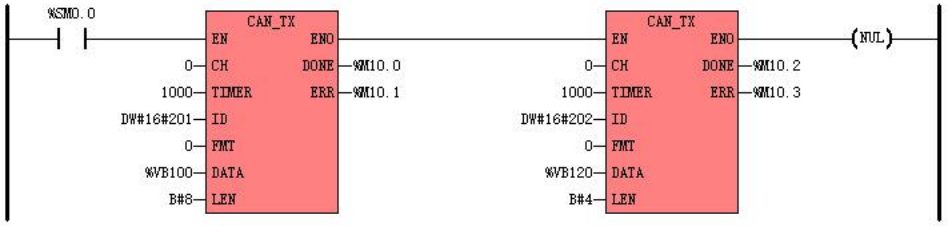
(* Network 3 *)
(* vd100和vd104是发送ID201中的8个字节长度，二者分别自加1和自加2 *)



(* Network 4 *)

(* Network 5 *)
(* vd120是发送ID202中的4个字节长度，自加3 *)

(* Network 6 *)
(* 以1s周期发送指定ID201、202的标准帧报文
plc后台会自动对需要发送的报文按着ID大小的优先级进行排队发送，ID越小优先级越高
不需要用户再进行其他处理，用户只需要设置好合理的发送周期即可
避免出现周期太短，需要发送的报文太多，造成总线拥堵的情况 *)



IL

```
(* Network 0 *)
(*初始化 can1 口，波特率 1M*)
LD          %SM0.1
CAN_INIT 0, 8, %M0.0
(* Network 1 *)
(*plc 上电后进入永久接收模式，后台自动接收指定 ID 为 101、102 的标准帧报文，不需要用户再重新触发此指令*)
LD          %SM0.1
CAN_RX    0, DW#16#101, 0, 0, 1000, %M0.1, %M0.2, %VB0, %VB10
CAN_RX    0, DW#16#102, 0, 0, 1000, %M0.3, %M0.4, %VB20, %VB30
(* Network 2 *)
LD          %SM0.0
ST          %BR0.0
(* Network 3 *)
(*vd100 和 vd104 是发送 ID201 中的 8 个字节长度，二者分别自加 1 和自加 2*)
LD          %SM0.3
R_TRIG
ADD         DI#1, %VD100
(* Network 4 *)
LD          %SM0.3
R_TRIG
ADD         DI#2, %VD104
(* Network 5 *)
(*vd120 是发送 ID202 中的 4 个字节长度，自加 3*)
LD          %SM0.3
R_TRIG
ADD         DI#3, %VD120
```

IL	(* Network 6 *) (*以 1s 周期发送指定 ID201、202 的标准帧报文 plc 后台会自动对需要发送的报文按着 ID 大小的优先级进行排队发送，ID 越小优先级越高不需要用户再进行其他处理，用户只需要设置好合理的发送周期即可避免出现周期太短，需要发送的报文太多，造成总线拥堵的情况*) LD %SM0.0 CAN_TX 0, 1000, DW#16#201, 0, %VB100, B#8, %M10.0, %M10.1 CAN_TX 0, 1000, DW#16#202, 0, %VB120, B#4, %M10.2, %M10.3

5、当出现 CAN 通信问题的时候，首先排查外部原因：接线是否正确，首尾两端节点是否都加上了终端电阻，还有外部干扰等一些因素。外部没有问题可以再去查找程序问题，通信的几个设备使用初始化 CAN 口指令时是否选择的对应的物理接线的 CAN 口，波特率是否相同等等。

6、CAN 自由通信中的接收指令 CAN_RX 在执行一次后在后台自动进入接收状态，不需要用户再去做其他处理。发送指令 CAN_TX，周期发送指定的 ID 报文，后台会对所有需要发送的报文以 ID 优先级进行排队发送（遵循 ID 越小优先级越高的原则），需要用户设置合理的发送周期和报文数量以免造成总线拥堵。

10.5.8 Kinco 运动控制功能

10.5.8.1 概述

Kinco 运动控制功能用于控制 Kinco 公司具有 CAN 接口的运动控制产品（伺服和步进驱动器）。它基于 CANOpen 协议，将与驱动器的 CANOpen 通信细节等进行了封装，并结合实际应用需求，为用户提供了提供了一组运动控制指令和相应的网络配置工具。本功能使用简便，用户即使不熟悉 CANOpen 协议细节，也可以很方便地实现与驱动器的通信并进行定位控制。

本功能支持对运动控制产品进行参数上传（下载）、电机锁轴、松轴、回原点、点动（速度模式）、绝对定位、相对定位等操作，暂不支持力矩模式和主从跟随模式等操作。另外，本功能原则上可以用于所有支持标准 CANopen 协议的第三方运动控制产品，**使用前请咨询步科技术人员**。

本功能不同型号 CPU 可以控制的运动控制轴数量不同，请参考下表。在实际应用中，用户可根据需要程序空间、网络负荷率等决定实际连接台数。

型号	连接的轴最大数量	用户程序中调用的 MC 指令的最大数量
K209M	16	192
KS101M		
K6 系列		
KS 系列（KS101M 除外）	16	192
KW 系列		

- **CAN 总线负荷率**

CAN 总线负载率是指在单位时间内总线上实际传输的位数和理论上能够传输的最大的位数的比值。

负荷率越高，表示 CAN 总线越繁忙，单位时间内传输的数据越多。通常，在实际应用中，业内一般要求 CAN 总线的负荷率不超过 30%，当超过这个值时，低优先级报文发生传输延迟的概率就会加大--甚至有可能一直得不到发送的机会，从而可能导致总线网络中某些设备节点控制异常。但“30%”这个值也只是一个历史悠久的经验数值，如果能通过优化网络及节点等手段以满足每个节点对响应时间的需求，那么当超过 70%的负荷率时 CAN 总线依然能够保证正常通信。

KPLC 中为 CAN2 接口提供了如下负荷率寄存器：

SMW122	CAN2 负荷率最小值	负荷率 = 寄存器值×100
SMW124	CAN2 负荷率最大值	

用户在程序中直接读取寄存器值就可以得到 CAN2 接口的实时总线负荷率。如果用户在实际应用中发现 CAN 网络中某些节点设备偶尔出现动作异常的现象，则可以读取实时负荷率来判断是否可能是总线负荷过高而导致了低优先级报文丢失，如果总线负荷率过高，则应当减少网络中连接的节点。

注意:CAN 总线具有位填充机制。CAN 控制器芯片在发送时只要检测到在位流中有连续 5 个相同位，就会自动在后面插入一个补码位，接收方会自动删除这个插入的位。当 CAN 发送不同的数据时，可能会出现位填充的现象，也可能不出现，这是由 CAN 控制器芯片自动完成的，外部软件无法得知这些信息，因此在计算负荷率时，KPLC 按照每一帧报文无位填充和有位填充这两种情况分别计算出一个负荷

率的最小值和最大值，真实的负荷率应当位于这两个值之间。

➢ **Kinco 运动控制功能的使用方法**

用户按照如下步骤使用 Kinco 运动控制功能：

1) 在用户工程中，进入【Kinco 运动控制网络配置】向导窗口并完成网络和轴参数的基本配置。
2) 根据实际需求调用运动控制指令进行编程。
3) 将工程下载到 PLC 中，则该 PLC 启动后将作为主站运行，管理整个网络的通信，并且执行定位控制程序。

10.5.8.2 Kinco 运动控制网络配置

Kinco 运动控制功能基于 CANOpen 协议，由 PLC 作为主站，各个驱动器作为从站。在调用指令之前，用户必须先对实际所用的 CANOpen 网络进行配置。按现场应用的习惯，我们在软件中把从站称为“轴”。

在 Kincobuilder 软件的【工程管理器】中，双击【Kinco 运动控制网络配置】节点即可进入配置窗口，在该窗口中完成网络的配置。

在窗口分了三部分区域:网络节点的树状列表、主站参数和轴（从站）的参数。

1) 网络节点树的操作

在网络节点树中，根节点是【CANOpen 主站】，下面的各个子节点是网络中的轴（从站）。

下方提供了【添加】、【删除】、【复制】和【删除】4 个按钮，同时软件也提供了相应的快捷键和右键菜单功能。用户可以利用这些功能对网络节点进行操作。

- 添加一个新的轴

单击【添加】按钮；或者在任一节点上单击右键，然后执行【添加】菜单命令；或者使用 ALT+N 快捷键。使用上述 3 种方法新增的轴在初始时均采用默认的参数。

- 复制、粘贴

用户可以先复制一个已有的轴，然后粘贴到网络中生成新的轴，新轴除了轴号（从站地址）之外，其它参数都跟被复制的轴保持一致。对于那种所有轴的功能都一样的应用来说，这个功能很方便。

先单击树中的某个轴选中它，然后单击【复制】按钮，或使用 Ctrl+C 快捷键；或者在某个轴上单击右键，执行【复制】菜单命令。这几种方法都可以复制这个轴。

复制完成后，再单击【粘贴】按钮，或者使用 Ctrl+P 快捷键，或者在任一轴上单击右键并执行【粘贴】菜单命令，都可以在网络中生成新的轴。

- 删除一个轴

先单击某个轴选中它，然后单击【删除】按钮，或者使用 DELETE 快捷键，都可以删除这个轴。

在某个轴上单击右键并执行【删除】菜单命令，也可以删除这个轴

2) 主站参数

单击【CANOpen 主站】节点，主站的所有参数将会可以修改，轴（从站）的所有参数将会变灰且不能修改。

- 【波特率】：选择主站所用的波特率。注意网络上所有节点（主站及从站）的波特率都必须一致。
- 【SDO 超时】：主站 PLC 发送 SDO 请求报文之后的超时等待时间，若超过这个时间没有收到相应从站的应答报文，则会报告超时错误。当选择不同波特率时，软件会自动推荐一个 SDO 超时时间，用户可以在这个值基础上修改。

3) 轴（从站）的参数

单击某个轴节点，该轴的所有参数将会可以修改，主站的所有参数将会变灰且不能修改。

（1）【轴号】：轴的 CANOpen 从站地址，**本系统中从站站号必须从 1 开始连续分配。**

（2）【类型】：根据轴的功能不同，用户可选择直线轴或者旋转轴。

（3）【编码器分辨率】：轴或者步进驱动器的编码器的分辨率，即编码器旋转一圈所发出的脉冲个数。

（4）【每圈机械当量】：电机轴每转动一圈，机械负载所移动的长度（直线轴，mm）或者转动的角度（旋转轴，°）。

（5）【节点保护】：设置该轴的节点保护时间。用户可以采用默认值，也可以点击“高级”自行修改。

（6）【PDO 禁止时间】：PLC 内为各轴自动建立了多个 PDO 用于传输位置、速度、状态等信息，因轴的位置、速度等变化很快，所以 PDO 发送非常频繁，必须设置 PDO 禁止时间。用户可以采用默认值，也可以自行修改。

4) 其它操作

（1）【确定】：保存当前界面所配置的参数并退出界面

（2）【取消】：只保存当前界面所配置并且已经点击应用的参数，然后退出界面

（3）【应用】：保存当前界面所配置的参数

（4）【轴一览表】：轴一览表主要是用来方便查看所有已经配置且使能的轴配置的参数，以便核对


轴一览表

轴号	轴类型	编码器分辨率	机械当量	节点保护时间	节点保护因子	PDO禁止时间
1	直线轴	10000	10.000000毫米	1000 ms	3	10 ms
2	旋转轴	10000	45.000000度	2000 ms	3	20 ms

10.5.8.3 Kinco 运动控制指令

10.5.8.3.1 运动控制指令综述

下述指令位于指令集的【Kinco 运动控制】组中。

名称	功能描述
MC_RPARAS	读取轴驱动器内的参数（具体见下文的参数表）
MC_WPARAS	修改轴驱动器内的参数

MC_POWER	控制锁轴、松轴
MC_RESET	复位轴上的错误信息，将轴状态置为静止等待状态
MC_HOME	控制轴回原点
MC_JOG	控制轴点动
MC_MABS/ MC_MABSE	控制轴进行绝对定位运动
MC_MREL/ MC_MRELE	控制轴进行相对定位运动
MC_STATE	读取驱动器的各状态数值
MC_RESTART	重新配置并启动从站
MC_MIOT	读取目标轴的序列号、软件版本、IIT、温度等设备信息

2) 注意事项

用户在使用这些指令时，需要注意如下几点：

- 在一个用户工程中，允许的最大轴数及专用指令使用的最大数量限制请参见 10.5.8.1 概述。
- 对于同一个轴，当某个专用指令正在执行、尚未完成时，不允许再启动执行另外的专用指令。假如此时用户程序启动了另外一个专用指令，那么这个指令会直接结束并报告错误。
- 对于同一个轴，用户程序执行运动指令（不含读写参数指令）之前，必须首先执行 MC_POWER 指令进行锁轴，在锁轴成功后才可以继续执行回原点、相对运动、绝对运动或者点动指令。若没有锁轴，那么执行这几种指令时会直接结束并报告错误。
- 对于同一个轴，用户程序使用 MC_RESET 指令进行复位，复位成功后，轴将处于松轴的静止等候状态，需先执行 MC_POWER 指令进行锁轴才可以继续执行回原点、相对运动、绝对运动或者点动指令。
- 对于回原点、相对运动、绝对运动或者点动指令，其使用的加减速度为驱动器内部设定的加减速度，用户也可通过 MC_WPARAS 指令设定。
- 在用户程序中调用各个运动控制指令的输出结果互不相干。若某条指令执行出错，则它的输出参数 ERRID 将给出错误代码，且这个错误结果直到下一次该条指令再次执行后才会再次刷新，其它指令执行的结果不会影响到该条指令的执行结果！
- 节点掉线后（MC_STATE 指令的 ONLINE 输出结果为 1），为安全考虑，PLC 不会自动重连本节点！用户必须排除错误后，断电重启 PLC 或者调用 MC_RESTART 指令来重新建立连接！

- 对于同一个轴，本功能不支持多个运控动作同时执行。同时本组指令的 DONE 信号输出可能会存在一定的延时，如果用户以 DONE 信号作为判断各指令相互动作联锁的依据，则需考虑并处理可能存在的延时，否则可能出现与预期工艺不相符的结果！

3) 指令输出参数 ERRID

每个指令均提供了 ERRID 输出参数。若指令执行成功，则 ERRID 输出为 0。若指令执行失败，则 ERRID 会被设置为不同的错误码值来说明错误的原因。

下面是各错误码值的说明。注意，**此处的错误码不适合于 MC_RPARAS、MC_WPARAS 和 MC_RESTART 指令，这几个指令的错误码另有特殊含义，**请另外参考指令说明。

错误码	描述
0	无错误
1	目标轴没有使能，或者网络中不存在该轴
2	目标轴没有处于锁轴状态。
3	目标轴正在执行其它运动控制指令，没有处于静止状态。
4	PLC 内部的 CAN 报文发送缓冲区满，不能发送 CAN 报文
5	PLC 向目标轴发送了 SDO 请求报文，但超时没有收到回应
6	PLC 向目标轴发送了 SDO 请求报文，但是收到了错误的回应报文
7	指令正常执行了，但是 PLC 持续检测目标轴返回的状态，最终没有检测到正确的状态值

在运动控制中，将轴分为若干个逻辑状态，而逻辑状态之间的转移则需要特定的条件或指定的 MC 运控指令。这样划分处理的好处是便于轴按运动模式分类控制，轴在一个时候只能处于一种逻辑状态，而逻辑状态的转移需要按规则进行，不会因不同 MC 的误触发而带来运行的混乱。

轴的逻辑状态共分为如下六种：

0：Power_off（Disabled）：轴未上电，或未使能，需执行 MC_Power 指令

1：Errorstop:———————— 轴处于故障停止状态，需执行 MC_Reset 指令

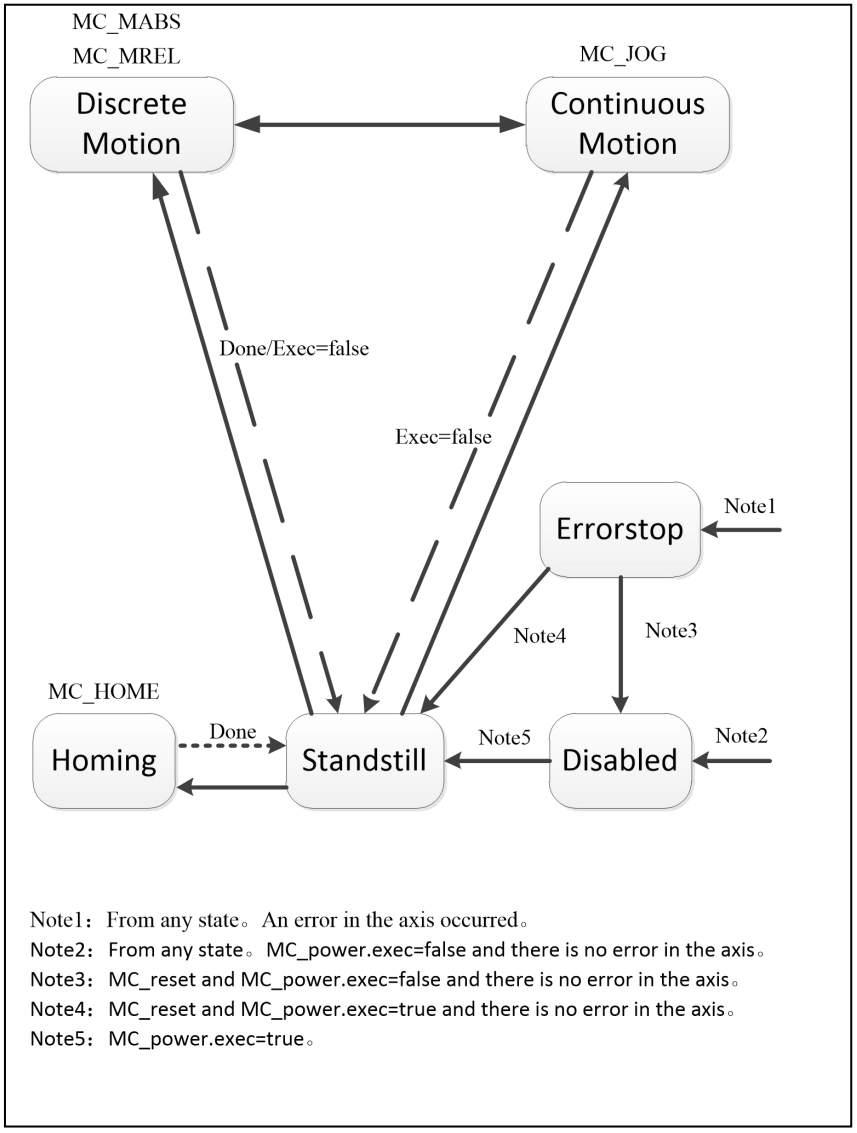
2：Standstill:————————轴处于静止状态

3：Discrete_Motion:———— 轴处于离散运行状态

4：Continuous_Motion:—— 轴处于连续运行状态

5：Homing:————————— 轴处于回零运行状态，等待归零操作执行完成

轴状态转移图如下：



10.5.8.3.2 运动控制指令

10.5.8.3.2.1 MC_RPARAS（读取参数）和 MC_WPARAS（修改参数）

本组指令的目的是方便用户批量操作驱动器参数，比如用户可以在调试初期一次性设定驱动器的参数。具体参数如何设置请查询驱动器操作手册，**设置不当则有可能运转异常，请谨慎操作。**

注意：下面列表中没有的参数可通过调用 SDO 指令发送 SDO 报文的方式进行读写，SDO 指令使用说明请参考 10.5.4.5.2 SDO 指令。

1) 可操作的驱动器参数列表

通过驱动器读写指令，可对驱动器的下列参数进行操作，所有参数均可读可写。每个指令一次最多操作 32 个参数。表中工艺数据类型，REAL 表示单精度浮点数，UINT32 表示无符号 32 位数，INT32 表示有符号 32 位数，其它以此类推。

表中的“序号”值是**固定的**，每个参数均有一个序号，用户在指令中输入序号就可以操作相应的参数。“工艺单位”指的是在指令参数中采用的单位，“驱动器取值范围”指的是驱动器内部的取值范围（本指令在内部会自动将用户需要的实际工艺参数值转换为驱动器内部使用的数据格式，比如加速度、速度、位置等）。

序号	参数名称	CANOpen 对象	工艺数据类型	工艺单位	驱动器内取值范围
0	**梯形加速度**	0x60830020	REAL	直线轴：mm/s² 旋转轴：1/s²	[0,268435455]
1	**梯形减速度**	0x60840020			
2	**找原点速度**	0x60990120	REAL	直线轴：mm/min 旋转轴：度/min	[-2147483648, 2147483647]
3	**找原点模式**	0x60980008	INT8	DEC	[-128,127]
4	**速度环比例增益 0**	0x60F90110	UINT16	DEC	[0,32767]
5	**速度环积分增益 0**	0x60F90210	UINT16	DEC	[0,32767]
6	**位置环比例增益 0**	0x60FB0110	REAL	HZ	[0,32767]
7	**位置环速度前馈**	0x60FB0210	REAL	%	[0,1024]
8	**速度环比例增益 1**	0x23400410	UINT16	DEC	[0,32767]

9	**速度环积分增益 1**	0x23400510	UINT16	DEC	[0,32767]
10	**位置环比例增益 1**	0x23400610	REAL	HZ	[0,32767]
11	**目标电流限制**	0x60730010	UINT16	DEC	[0,2048]
12	**最大速度限制**	0x607F0020	REAL	直线轴：mm/min 旋转轴：度/min	[-2147483648, 2147483647]
13	**原点偏移模式**	0x60990508	UINT8	无单位	[0,255]
14	**电机方向**	0x607E0008	UINT8	无单位	0 和 1
15	**电机型号**	0x64100110	UINT16	无单位	[0,65535]
16	**软限位正设置**	0x607D0120	REAL	直线轴：mm 旋转轴：度	[-2147483648, 2147483647]
17	**软限位负设置**	0x607D0220			
18	**平滑滤波**	0x60FB0510	UINT8	无单位	[0,255]
19	**最大跟随误差**	0x60650020	UINT32	DEC	[0,268435455]
20	**目标位置窗口**	0x60670020	UINT32	DEC	[0,268435455]
21	**位置窗口时间**	0x60680010	UINT16	DEC	[0,32767]
22	**速度反馈滤波**	0x60F90508	REAL	HZ	[0,45]
23	**速度反馈模式**	0x60F90608	UINT8	无单位	[0,85]
24	**输入口极性**	0x20100110	UINT8	无单位	[0,255]
25	**输入口 1 功能**	0x20100310	UINT16	无单位	[0,65535]
26	**输入口 2 功能**	0x20100410	UINT16	无单位	[0,65535]
27	**输入口 3 功能**	0x20100510	UINT16	无单位	[0,65535]
28	**输入口 4 功能**	0x20100610	UINT16	无单位	[0,65535]
29	**输入口 5 功能**	0x20100710	UINT16	无单位	[0,65535]
30	**输入口 6 功能**	0x20100810	UINT16	无单位	[0,65535]
31	**输入口 7 功能**	0x20100910	UINT16	无单位	[0,65535]
32	**输入口 8 功能**	0x20101D10	UINT16	无单位	[0,65535]
33	**输出口极性**	0x20100D10	UINT8	无单位	[0,255]

34	**输出口 1 功能**	0x20100F10	UINT16	无单位	[0,65535]
35	**输出口 2 功能**	0x20101010	UINT16	无单位	[0,65535]
36	**输出口 3 功能**	0x20101110	UINT16	无单位	[0,65535]
37	**输出口 4 功能**	0x20101210	UINT16	无单位	[0,65535]
38	**输出口 5 功能**	0x20101310	UINT16	无单位	[0,65535]
39	**输出口 6 功能**	0x20101E10	UINT16	无单位	[0,65535]
40	**输出口 7 功能**	0x20101F10	UINT16	无单位	[0,65535]
41	**齿轮前脉冲数据**	0x25080420	INT32	DEC	[-2147483648, 2147483647]
42	**脉冲模式**	0x25080308	UINT8	无单位	[0,255]
43	**保存参数**	0x10100120	UINT32	无单位	仅 16#65766173 有效
44	**初始化参数**	0x10110120	UINT32	无单位	仅 16#64616f6C 有效

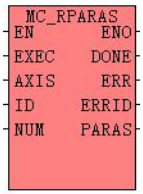
2) ERRID 参数说明

读、写参数指令均提供了 *ERRID*（DWORD 型）输出参数。

这个参数值是错误码，表示了指令执行过程中发生过的错误。

错误码	含义
0xFFFFFFFF	发生过导致指令无法执行的错误，包括： 1）用户输入的轴号错误、参数数量错误 2）有其它 Kinco 专用指令正在运行 3）指令要操作 32 个参数，结果这 32 个参数均操作失败
其它值	ERRID 的每个位均表示了对应参数的操作结果，每个位与 *ID* 参数序号表中指定的参数一一对应：bit0 表示本次操作第 1 个参数的结果，bit1 表示第 2 个参数的操作结果，以此类推。某位值为 1，表示对应参数操作失败，否则表示对应参数操作成功。

3) MC_RPARAS（读取参数）

	名称	指令格式	适用于
LD	MC_RPARAS		☑ KS ☑ KW ☑ K209M ☑ K6

参数	输入/输出	数据类型	允许的内存区	描述
EXEC	输入	BOOL	M、V、L、SM	若检测到 *EXEC* 的上升沿，则指令被触发执行。
AXIS	输入	INT	V、M、L、常量	目标轴的轴号（也就是 CANOpen 从站的地址）
ID	输入	BYTE	V、M、L	要读取的参数的序号表的起始地址。
NUM	输入	INT	V、M、L、常量	要读取的参数数量
DONE	输出	BOOL	M、V、L	完成标志位。指令执行完成时，*DONE* 由 0 跳变到 1。
ERR	输出	BOOL	M、V、L	错误标志位。若指令执行时发生错误，则被置 1。
ERRID	输出	DWORD	V、M、L	错误码
PARAS	输出	DWORD	V、M、L	读取到的所有参数值的存放起始地址。

注意：AXIS 和 NUM 必须同时为常量类型或同时为内存类型，另外，ID 和 PAPAS 参数共同组成了一个长度可变的内存块，此内存块必须全部位于合法的内存区域，否则结果不可预期。

ID、*PARAS*、*NUM* 这 3 个参数共同组成了一个参数表。其中 *ID* 是序号表的起始地址，从这个地址开始连续依次存储着待操作的各个参数的序号（即前文参数列表中的“序号”），每个序号占用 **1 个字节**；*PARAS* 是参数值表的起始地址，从这个地址开始连续依次存储着读取到的各个参数的数值，每个数值均占用 **4 个字节**；*NUM* 是待操作的参数的数量。例如下例中，假定 *ID* 参数是 VB100，*PARAS* 参数是 VD1200，*NUM* 参数是 3，那么 VB100、VB101、VB102 分别存放着本次要操作的 3 个参数的序号，当指令执行完成后，读取的 3 个参数值分别存放在 VD1200、VD1204、VD1208 中。

对于 *PARAS* 要注意，虽然参数值表统一采用了 DWORD 地址，但各个工艺参数的实际数据类型并不相同，因此表中在用户程序中用户要根据实际数据类型来处理参数表中的数据。

- ✧ 若实际工艺数据类型为 REAL 型，那么直接以浮点数地址来操作参数内存即可。比如下例中，第一个参数值的序号 0 是一个 REAL 参数，存放于 VD1200，那么可以直接操作 VR1200。因为 VD1200 与 VR1200 在 PLC 中实际占用的是同一片内存地址。
- ✧ 若实际工艺数据类型是 REAL 之外的其它数据类型，并且相应的参数内存没有在全局变量表中强制定义数据类型，那么直接读取参数内存即可，因为指令会自动处理各种有符号和无符号整数。比如下例中，第三个参数值的序号是 8 存放于 VD1208，而实际类型是 INT32 或者 UINT32，那么直接操作 VD1208。

1) LD 格式指令说明

若 *EN* 为 1，那么在 *EXEC* 输入的上升沿，该指令被触发执行，指令根据 *ID*、*NUM* 指定的待读取的参数表，依次发送 SDO 给驱动器来读取相应的对象，并将读到的数据依次放入 *PARAS* 指定的数值表中，同时将 *ERRID* 相应位设置为 0。若某个参数的 SDO 响应错误或者超时无响应，则 *PARAS* 中相应地址的数据保持不变，同时将 *ERRID* 相应位设置为 1，然后继续读取下一个参数。当读取完成所有参数后，DONE 被置 1，ERR、ERRID 根据执行结果来设置为不同的值。

若 EN 为 0，则指令不执行。在指令执行过程中 EXEC 变为 0，则指令会停止读取尚未完成的参数，并将 DONE 置 1，ERR、ERRID 维持已经执行的结果。

若在指令启动时，PLC 检测到错误（比如轴未使能、轴正在执行其它指令等），则直接退出，将 DONE、ERR 置 1，ERRID 设置为相应的错误码。

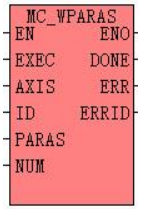
➢ **示例**

本例子采用 IL 格式。在 Kincobuilder 中，先在【工程】菜单中选择【IL】格式，然后可以将示例复制粘贴到编辑器中，然后再选择【LD 格式】，程序就可以显示为 LD 格式了。

```
(* Network 0 *)
(*设置参数表，指明本次要读取参数 0、3 和 8.*)
LD          %SM0.0
MOVE        B#0, %VB100
MOVE        B#3, %VB101
MOVE        B#8, %VB102
(* Network 1 *)
```

```
(*调用指令。此次，AXIS、NUM 参数均为常量，它们也支持全内存地址的格式。*)
LD          %M0.0
MC_RPARAS %M0.1, 1, %VB100, 3, %M0.2, %M0.3, %MD8, %VD1200
(* Network 2 *)
(*读取的参数值依次存放于 PARAS 参数起始的参数值表（本例是 V 区 1200 起）中。表中第 1 个数据是读取的第 1 个参数值，即表中参数 0，因为它是 REAL 型，所以读取浮点型内存地址。*)
LD          %SM0.0
MOVE        %VR1200, %VR300
(* Network 3 *)
(*表中第 2 个数据是读取的第 2 个参数值，即表中参数 3。这个参数是有符号 8 位数，因为 PLC 中没有提供这种数据类型，所以按整数进行处理。*)
LD          %SM0.0
DI_TO_I %VD1204, %VW304
(* Network 4 *)
(*表中第 3 个数据是读取的第 3 个参数值，即参数 8。这个参数是无符号 16 位数，但最大范围是 32767，所以程序中按 INT 或者 WORD 型处理均可，但最好先判断一下数值是否在允许范围内。*)
LD          %SM0.0
DI_TO_I %VD1208, %VW308
NE          %VW308, 0
ST          %M3.0
```

4) MC_WPARAS（修改参数）

	名称	指令格式	适用于
LD	MC_WPARAS		☑ KS ☑ KW ☑ K209M ☑ K6

参数	输入/输出	数据类型	允许的内存区	描述
EXEC	输入	BOOL	M、V、L、SM	若检测到 *EXEC* 的上升沿，则指令被触发执行。
AXIS	输入	INT	V、M、L、常量	目标轴的轴号（也就是 CANOpen 从站的地址）
ID	输入	BYTE	V、M、L	要修改的参数的序号表的起始地址。
PARAS	输出	DWORD	V、M、L	读修改的所有参数值的存放起始地址。
NUM	输入	INT	V、M、L、常量	要修改的参数数量
DONE	输出	BOOL	M、V、L	完成标志位。指令执行完成时，*DONE* 由 0 跳变到 1。
ERR	输出	BOOL	M、V、L	错误标志位。若指令执行时发生错误，则被置 1。
ERRID	输出	DWORD	V、M、L	错误码

注意：AXIS 和 NUM 必须同时为常量类型或同时为内存类型，另外，ID 和 PAPAS 参数共同组成了一个长度可变的内存块，此内存块必须全部位于合法的内存区域，否则结果不可预期。

ID、*PARAS*、*NUM* 这 3 个参数共同组成了一个参数表。其中 *ID* 是序号表的起始地址，从这个地址开始连续依次存储着待操作的各个参数的序号（即前文参数列表中的“序号”），每个序号占用 1 个字节；*PARAS* 是参数值表的起始地址，从这个地址开始连续依次存储着各个参数的数值，每个数值均占用 4 个字节；*NUM* 是待操作的参数的数量。例如下例中，假定 *ID* 参数是 VB200，*PARAS* 参数是 VD2000，*NUM* 参数是 3，那么 VB200、VB201、VB202 分别存放着本次要操作的 3 个参数的序号，VD2000、VD2004、VD2008 分别存放着待修改的参数值。

对于 *PARAS* 要注意，虽然参数值表统一采用了 DWORD 地址，但各个工艺参数的实际数据类型并不相同，因此表中在用户程序中用户要根据实际数据类型来给参数表中相应地址赋值。

- ✧ 若实际工艺数据类型为 REAL 型，那么直接以浮点数地址来操作参数内存即可。比如，参数值期望存放于 VD2000，那么可以直接操作 VR2000。VD2000 与 VR2000 在 PLC 中实际占用的是同一片内存地址，该指令会自动做类型转换。
- ✧ 若实际工艺数据类型是 REAL 之外的其它数据类型，那么直接操作参数内存即可，指令会根据参数的数据类型自动做类型转换。比如本例中，要操作的参数 3 和 8 数据类型是 UINT8 与 UINT16，那么就直接赋一个合法的数值给 VD2004 和 VD2008 即可。

1) LD 格式指令说明

若 *EN* 为 1，那么在 *EXEC* 输入的上升沿，该指令被触发执行，指令根据 *ID*、*PARAS*、*NUM* 指定的参数表，依次将 PARAS 中的数值通过 SDO 发送给驱动器来修改相应的对象，同时将 *ERRID* 相应位设置为 0。若某个参数的 SDO 响应错误或者超时无响应，则将 *ERRID* 相应位设置为 1，然后继续写入下一个参数。当写入完成所有参数后，DONE 被置 1，ERR、ERRID 根据执行结果来设置为不同的值。

若 EN 为 0，则指令不执行。若在指令执行过程中 EN 变为 0，则指令会停止写入尚未完成的参数，并将 DONE 置 1，ERR、ERRID 维持已经执行的结果。

若在指令启动执行时，PLC 检测到错误（比如轴未使能、轴正在执行其它指令等），则直接退出，将 DONE、ERR 置 1，ERRID 设置为相应的错误码。

➢ **示例**

本例子采用 IL 格式。在 Kincobuilder 中，先在【工程】菜单中选择【IL】格式，然后可以将示例复制粘贴到编辑器中，然后再选择【LD 格式】，程序就可以显示为 LD 格式了。

```
(* Network 0 *)
(*设置参数表，指明本次要写入参数 0、3 和 8.*)
LD          %SM0.0
MOVE        B#0, %VB200
MOVE        B#3, %VB201
MOVE        B#8, %VB202
(* Network 1 *)
(*设置各个参数待写入的数值。注意数据类型与每个参数自动占用 32 位地址。本例相当于给表中参数 0 写入 1200.0，参数 3 写入 8，参数 8 写入 2000*)
LD          %SM0.0
MOVE        1200.0, %VR2000
MOVE        DI#8, %VD2004
MOVE        DI#2000, %VD2008
(* Network 2 *)
(*调用指令*)
```

LD %SM0.0

MC_WPARAS %M1.1, 1, %VB200, %VD2000, 8, %M1.2, %M1.3, %MD14

10.5.8.3.2.2 MC_POWER（锁轴和松轴）

	名称	指令格式	适用于
LD	MC_POWER		☑ KS ☑ KW ☑ K209M ☑ K6

参数	输入/输出	数据类型	允许的内存区	描述
EXEC	输入	BOOL	M、V、L、SM	上升沿触发锁轴命令，下降沿触发松轴命令。
AXIS	输入	INT	V、M、L、常量	目标轴的轴号（也就是 CANOpen 从站的地址）
DONE	输出	BOOL	M、V、L	完成标志位。指令执行完成时，*DONE* 由 0 跳变到 1。
ERR	输出	BOOL	M、V、L	错误标志位。若指令执行时发生错误，则被置 1。
ERRID	输出	BYTE	V、M、L	错误码

- LD 格式指令说明

若*EN*为1，那么在*EXEC*的上升沿会触发执行锁轴命令，在*EXEC*的下降沿会触发执行松轴命令。

在指令执行时，PLC首先发送命令控制轴进入待操作状态，并在5S超时时间内检查驱动器实际返回状态，若成功执行则表示指令执行成功，则DONE被置1，ERR被置0，ERRID被置0。若发生错误（可能是指令本身执行错误，也可能是执行过程中驱动器未正确执行动作的错误，详见错误码）则指令执行失败，指令都会停止执行，同时将DONE被1，ERR置1，ERRID赋值为相应的错误代码。

若 *EN* 为 0，则指令不执行。

10.5.8.3.2.3 MC_RESET（复位驱动器报警）

	名称	指令格式	适用于
LD	MC_RESET		☑ KS ☑ KW ☑ K209M ☑ K6

参数	输入/输出	数据类型	允许的内存区	描述
EXEC	输入	BOOL	M、V、L、SM	上升沿触发本指令执行一次。
AXIS	输入	INT	V、M、L、常量	目标轴的轴号（也就是 CANOpen 从站的地址）
DONE	输出	BOOL	M、V、L	完成标志位。指令执行完成时，*DONE* 由 0 跳变到 1。
ERR	输出	BOOL	M、V、L	错误标志位。若指令执行时发生错误，则被置 1。
ERRID	输出	BYTE	V、M、L	错误码

当轴在运行过程中出错时，可以调用本指令将轴上的错误信息复位，同时将轴置为松轴静止等待状态。**复位成功后如需继续执行其他运动指令，则应首先调用 MC_POWER 指令锁轴！**

注意：本指令仅复位驱动器的报警错误信息，并不复位各指令的输出结果！

- LD 格式指令说明

若*EN*为1，那么在*EXEC*的上升沿会触发执行本指令。

在指令执行时，PLC首先发送指令复位驱动器报警，并在2秒钟超时时间内检查驱动器实际状态，若成功复位，则表示指令执行成功，则DONE被置1，ERR被置0，ERRID被置0。若发生错误（可能是指令本身执行错误，也可能是执行过程中驱动器未正确执行动作的错误，详见错误码）则指令执行失败，指令都会停止执行，同时将DONE被1，ERR置1，ERRID赋值为相应的错误代码。

若 *EN* 为 0，则指令不执行。

10.5.8.3.2.4 MC_HOME（回原点）

	名称	指令格式	适用于
LD	MC_HOME		☑ KS ☑ KW ☑ K209M ☑ K6

参数	输入/输出	数据类型	允许的内存区	描述
EXEC	输入	BOOL	M、V、L、SM	上升沿触发本指令执行一次；下降沿触发暂停运动
AXIS	输入	INT	V、M、L、常量	目标轴的轴号（也就是 CANOpen 从站的地址）
POS	输入	REAL	V、M、L、常量	原点的偏移位置，单位：mm 或者° 。
TIME	输入	DWORD	V、M、L、常量	超时时间，若在此时间内没有找到原点，则报错退出。
DONE	输出	BOOL	M、V、L	完成标志位。指令执行完成时，*DONE* 由 0 跳变到 1。
ERR	输出	BOOL	M、V、L	错误标志位。若指令执行时发生错误，则被置 1。
ERRID	输出	BYTE	V、M、L	错误码

执行本指令，可以使目标轴回原点。*POS*参数设置了原点坐标的偏移值。

注意：本指令使用驱动器内部回原点模式，需在驱动器首先设好 60980008 回原点模式及回原点速度等相关参数（也可通过 MC_WPARAS 指令写入），详情请参考驱动器使用手册。

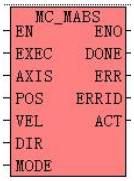
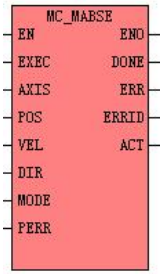
- LD 格式指令说明

若*EN*为1，那么在*EXEC*的上升沿会触发执行本指令。

在指令执行时，PLC首先发送命令让轴开始找原点；发送完成后，指令会持续检查驱动器返回的状态，检查时间为*TIME*（用户设定的超时时间，单位ms），若在此时间内轴成功找到原点，则表示指令执行成功，此时DONE被置1，ERR被置0，ERRID被置0。若发生错误（可能是指令本身执行错误，也可能是执行过程中驱动器未正确执行动作的错误，详见错误码）则指令执行失败，指令都会停止执行，同时将DONE被1，ERR置1，ERRID赋值为相应的错误代码。

若 *EN* 为 0，则指令不执行。若在执行过程中 *EN* 变为 0 或者 *EXEC* 变为 0，则指令将停止执行，轴处于静止锁轴等候状态。

10.5.8.3.2.5 MC_MABS（绝对运动）

	名称	指令格式	适用于
LD	MC_MABS		☑ KS ☑ KW ☑ K209M ☑ K6
	MC_MABSE		

参数	输入/输出	数据类型	允许的内存区	描述
EXEC	输入	BOOL	M、V、L、SM	上升沿触发本指令执行一次；下降沿触发暂停运动
AXIS	输入	INT	V、M、L、常量	目标轴的轴号（也就是 CANOpen 从站的地址）
POS	输入	REAL	V、M、L、常量	绝对目标位置，单位：mm 或者°
VEL	输入	REAL	V、M、L、常量	运动中增加的最大速度（>0），单位：mm/min 或° /min。
DIR	输入	INT	V、M、L、常量	运动方向。预留用，暂未实现功能，保持为 0 即可。
MODE	输入	INT	V、M、L、常量	运动模式：单次执行或者永久执行。 0 表示单次执行，轴执行完本次绝对定位后指令就退出。 1 表示永久执行，当轴执行完一次绝对定位后，指令并不退出，若发现新的目标位置就会发送命令让轴继续执行新一次绝对定位。
PERR	输入	REAL	V、M、L、常量	目标位置和实际位置的判断差值，当需要定位的目标位

				置减去实际位置的绝对值小于等于 RERR 时，*DONE* 由 0 跳变到 1。
DONE	输出	BOOL	M、V、L	完成标志位。指令执行完成时，*DONE* 由 0 跳变到 1。
ERR	输出	BOOL	M、V、L	错误标志位。若指令执行时发生错误，则被置 1。
ERRID	输出	BYTE	V、M、L	错误码
ACT	输出	BOOL	M、V、L	MODE=0,单次执行时，ACT 表示单次定位指令是否被正确激活。1 表示激活，0 表示未激活。 MODE=1,永久执行时，ACT 表示永久定位指令是否被正确激活。1 表示激活（单次定位完成时也会一直处于 1），0 表示未激活。

本指令用于控制目标轴运动到目标位置（绝对位置）。运动时，速度从 VEL 当前值开始，到达目标位置时速度为零。本指令允许在运动过程中暂停运动。

MC_MABS 和 MC_MABSE 指令的区别：MC_MABS 指令给出 DONE 信号的条件是判断伺服给过来的“位置到”状态字；MC_MABSE 指令给出 DONE 的条件是判断伺服给过来的“位置到”状态字，同时再去判断目标位置与实际位置之间差值的绝对值是否小于等于 RERR。**建议用户使用 MC_MABSE 指令，此指令给出的 DONE 信号更加可靠。**

注意：本指令使用驱动器内部加减速，需在驱动器首先设好 60830020 加速度及减速度等相关参数（也可通过 MC_WPARAS 指令写入），详情请参考驱动器使用手册。

- LD 格式指令说明

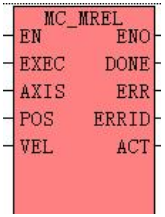
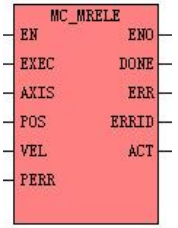
若*EN*为1，那么在*EXEC*的上升沿会触发执行本指令。

在指令执行时，PLC控制轴按照用户输入的目标位置（*POS*）、运动速度（*VEL*）参数值，让轴开始绝对定位。在运动过程中，指令将不停扫描目标位置和和目标速度参数值，若有变化则立即发送给轴，也就是随时可以接受新的速度参数和位置参数值（例如要执行暂停，在运动过程中将速度置为0即可暂停，重新给速度值则恢复运动）。同时，PLC将不停检查检查轴的返回状态，如果成功到达本次定位的目标位置，表示本次定位完成，则DONE被置1，ERR被置0，ERRID被置0。本次定位完成后，指令将判断模式（*MODE*）值，若设置为单次运行模式，那么指令直接退出；若设置为永久运行模式，那么指

令并不退出，随时扫描目标位置值，若目标位置发生变化就会将其发送给轴，让轴进行新一次绝对定位。若发生错误（可能是指令本身执行错误，也可能是执行过程中驱动器未正确执行动作的错误，详见错误码）则指令执行失败，指令都会停止执行，同时将DONE被1，ERR置1，ERRID赋值为相应的错误代码。

若 *EN* 为 0，则指令不执行。若在执行过程中 *EN* 变为 0 或者 *EXEC* 变为 0，则指令将停止执行，轴处于静止锁轴等候状态。

10.5.8.3.2.6 MC_MREL（相对运动）

	名称	指令格式	适用于
LD	MC_MREL		☑ KS ☑ KW ☑ K209M ☑ K6
	MC_MRELE		

参数	输入/输出	数据类型	允许的内存区	描述
EXEC	输入	BOOL	M、V、L、SM	上升沿触发本指令执行一次；下降沿触发暂停运动
AXIS	输入	INT	V、M、L、常量	目标轴的轴号（也就是 CANOpen 从站的地址）
POS	输入	REAL	V、M、L、常量	要运动的相对距离，单位：mm 或者°。 正数表示正方向运动；负数表示负方向运动。
VEL	输入	REAL	V、M、L、常量	运动中增加的最大速度（>0），单位：mm/min 或°/min。
PERR	输入	REAL	V、M、L、常量	目标位置和实际位置的判断差值，当需要定位的目标位置减去实际位置的绝对值小于等于 RERR 时，*DONE* 由 0 跳变到 1。

DONE	输出	BOOL	M、V、L	完成标志位。指令执行完成时，*DONE* 由 0 跳变到 1。
ERR	输出	BOOL	M、V、L	错误标志位。若指令执行时发生错误，则被置 1。
ERRID	输出	BYTE	V、M、L	错误码
ACT	输出	BOOL	M、V、L	该指令是否被正确激活。1 表示激活，0 表示未激活。

本指令控制目标轴来运动指定的距离 *POS*（以当前位置作为参考，也就是将当前位置做为起始位置）。运动时，速度从 VEL 当前值开始，到达目标位置时速度为零。本指令允许暂停。

MC_MREL 和 MC_MRELE 指令的区别：MC__指令给出 DONE 信号的条件是判断伺服给过来的“位置到”状态字；MC_MRELE 指令给出 DONE 的条件是判断伺服给过来的“位置到”状态字，同时再去判断目标位置与实际位置之间差值的绝对值是否小于等于 RERR。**建议用户使用 MC_MRELE 指令，此指令给出的 DONE 信号更加可靠。**

注意：本指令使用驱动器内部加减速，需在驱动器首先设好 60830020 加速度及减速度等相关参数（也可通过 MC_WPARAS 指令写入），详情请参考驱动器使用手册。

- LD 格式指令说明

若*EN*为1，那么在*EXEC*的上升沿会触发执行本指令。

在指令执行时，PLC控制轴按照用户输入的目标位置（*POS*）、运动速度（*VEL*）参数值，让轴开始相对定位（以当前位置作为参考）。在运动过程中，指令将不停扫描目标速度参数值，若有变化则立即发送给轴，也就是随时可以接受新的速度参数值（例如要执行暂停，在运动过程中将速度置为0即可，然后重新给速度值则恢复运动）。同时，PLC将不停检查检查轴的返回状态，如果成功到达本次定位的目标位置，表示本次定位完成，则*DONE*被置1，*ERR*被置0，*ERRID*被置0。若发生错误（可能是指令本身执行错误，也可能是执行过程中驱动器未正确执行动作的错误，详见错误码）则指令执行失败，指令都会停止执行，同时将*DONE*被1，*ERR*置1，*ERRID*赋值为相应的错误代码。

若 *EN* 为 0，则指令不执行。若在执行过程中 *EN* 变为 0 或者 *EXEC* 变为 0，则指令将停止执行，轴处于静止锁轴等候状态。

10.5.8.3.2.7 MC_JOG（点动）

	名称	指令格式	适用于
LD	MC_JOG		☑ KS ☑ KW ☑ K209M ☑ K6

参数	输入/输出	数据类型	允许的内存区	描述
EXEC	输入	BOOL	M、V、L、SM	上升沿触发本指令执行一次；下降沿触发暂停运动
AXIS	输入	INT	V、M、L、常量	目标轴的轴号（也就是 CANOpen 从站的地址）
VEL	输入	REAL	V、M、L、常量	运动速度，单位：mm/min 或°/min。 正数表示正方向，负数表示负方向。
DIR	输入	INT	V、M、L、常量	运动方向。预留用，暂未实现功能，保持为 0 即可。
DONE	输出	BOOL	M、V、L	完成标志位。指令执行完成时，*DONE* 由 0 跳变到 1。
ERR	输出	BOOL	M、V、L	错误标志位。若指令执行时发生错误，则被置 1。
ERRID	输出	BYTE	V、M、L	错误码
ACT	输出	BOOL	M、V、L	该指令是否被正确激活。1 表示激活，0 表示未激活。

本指令控制目标轴以 Vel 指定的目标速度运行。

注意：本指令使用驱动器内部加减速，需在驱动器首先设好 60830020 加速度及减速度等相关参数（也可通过 MC_WPARAS 指令写入），详情请参考驱动器使用手册。

- LD 格式指令说明

若*EN*为1，那么在*EXEC*的上升沿会触发执行本指令。

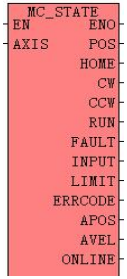
在指令执行时，PLC控制轴按照用户输入的运动速度（*VEL*）参数值，让轴开始点动运行。轴运动过程中，指令将不停扫描目标速度参数值，若有变化则立即发送给轴，也就是随时可以接受新的速度参数值。

若发生错误（可能是指令本身执行错误，也可能是执行过程中驱动器未正确执行动作的错误，详见

错误码）则指令执行失败，指令都会停止执行，同时将DONE被1，ERR置1，ERRID赋值为相应的错误代码。

若 *EN* 为 0，则指令不执行。若在执行过程中 *EXEC* 变为 0，则指令将停止执行，轴处于静止锁轴等候状态。

10.5.8.3.2.8 MC_STATE（读取驱动器的各状态数值）

	名称	指令格式	适用于
LD	MC_STATE		☑ KS ☑ KW ☑ K209M ☑ K6

参数	输入/输出	数据类型	允许的内存区	描述
AXIS	输入	INT	V、M、L、常量	目标轴的轴号（也就是 CANOpen 从站的地址）
POS	输出	BOOL	V、M、L	“位置到”信号
HOME	输出	BOOL	V、M、L	“原点找到”信号
CW	输出	BOOL	M、V、L	“电机正转”信号
CCW	输出	BOOL	M、V、L	“电机反转”信号
RUN	输出	BOOL	M、V、L	“电机运行中”标志
FAULT	输出	BOOL	M、V、L	“轴报警”标志
INPUT	输出	WORD	V、M、L	轴的开关量输入口状态，BIT0 对应轴的 DIN1，依次顺序存放，具体数字量输入口数量查询驱动器手册
LIMIT	输出	BOOL	M、V、L	“限位到”标志
ERRCODE	输出	WORD	V、M、L	轴的报警错误码
APOS	输出	REAL	M、V、L	机械的当前实际位置，mm 或者° 。
AVEL	输出	REAL	M、V、L	机械的当前实际速度，mm/min 或°/min。
ONLINE	输出	BYTE	M、V、L	“轴在线”标志。1 表示轴不在线，0 表示轴在线。

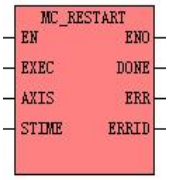
本指令一直扫描驱动器状态，获取各种不同状态的标志并输出到相应的输出参数中。**但可能存在一定的延时，用户使用各输出作为判断动作依据时需慎重**

注意：“位置到”和“原点找到”这两个信号在执行动作过程中（定位或找原点）会重新变为 0，直到动作正确执行完后才会重新置 1！

- LD 格式指令说明

若*EN*为1，则本指令执行。若*EN*为0，则指令不执行，也不会刷新各种输出参数。

10.5.8.3.2.9 MC_RESTRAT（重新配置并启动从站）

	名称	指令格式	适用于
LD	MC_RESTRAT		☑ KS ☑ KW ☑ K209M ☑ K6

参数	输入/输出	数据类型	允许的内存区	描述
EXEC	输入	BOOL	I、Q、V、M、L、SM	EXEC 的上升沿会启动本指令，执行一次重启从站的过程。 EXEC 的下降沿会停止本指令的执行，终止重启从站的过程。
AXIS	输入	INT	V、M、L、常量	目标轴的轴号（也就是 CANOpen 从站的地址）。
STIME	输入	INT	V、M、L、常量	主站发送 SDO 请求报文后等待从站回应的最长时间。
DONE	输出	BOOL	Q、M、V、L	指令执行结果标志位。若指令在执行过程中，DONE 输出为 0。 若指令执行结束(无论成功还是失败)，则 DONE 立即变为 1。
ERR	输出	BOOL	Q、M、V、L	指令执行错误标志位。若指令执行时发生错误，则被置 1。
ERRID	输出	BYTE	V、M、L	指令执行的错误码。如果指令执行时发生错误，则 ERRID 就是具体的错误信息。

注意：AXIS 和 STIME 参数必须同时为常量或同时为变量。在一个用户工程中，最多允许使用 32 个 MC_Restart 指令。

故障码（ERRID）的说明，见下表：

错误码	说明
0	无错误。
1	目标轴站号错误。
2	目标轴没有使能（必须在“Kinco 运动控制网络配置”窗口中使能）。
3	目标轴正在运动，禁止重启。
4	目标轴正在执行重启从站的过程，在完成之前禁止再次执行。
5	主站发送 CAN 报文错误（可能是线路、硬件接口、软件缓冲区满等原因）
6	主站发送的 SDO 请求报文，目标轴超时没有回应。
7	主站发送的 SDO 请求报文，目标轴回应错误。
8	用户主动停止本指令的执行（将 EXEC 设置为 0）。

- LD 格式指令说明

若*EN*为1，则本指令执行。若*EN*为0，则指令不执行，也不会刷新各种输出参数。

在主站正常运行过程中，用户可以调用本指令来重新配置并启动目标轴（站号为 AXIS）。轴的各种参数会采用【Kinco 运动控制网络配置】对本从站已经配置好的参数，包括错误监督参数等等。

若EN为1，那么EXEC的上升沿会触发执行本指令，首先DONE输出为0，然后主站将会读取已经配置好的轴参数，对目标轴依次进行如下操作：

1）发送命令让目标轴进入预操作状态 。

2）发送SDO来配置目标轴的节点保护参数。

3）发送SDO来配置目标轴所有PDO的映射参数。

4）发送SDO来配置目标轴所有PDO的通信参数。

5）向目标轴发送“启动节点”命令。

当上述过程成功执行完后，本指令会退出，并将DONE立即输出为1，ERR和ERRID均输出为0。

如果指令在执行过程中发生任何错误，或者EXEC 变为0，则本指令也会退出，并将 DONE、ERR 立即输出为1，ERRID 输出相应的错误代码。

第十一章 模拟量及 PID 使用

本章节重点介绍 Kinco-K 系列 PLC 的模拟量及 PID 指令的应用，包括指令的调用、参数的设定、使用注意点、 程序样例等。

11.1 功能概述

KPLC 部分 CPU 型号的本体提供了 AI（模拟量输入）和 AO（模拟量输出）通道，支持 0-20mA、0-10V 等信号形式。同时，KPLC 提供了支持各种模拟量输入信号（包括电流、电压、RTD、TC 等）和模拟量输出信号（包括电流、电压）的扩展模块。

PID 的使用通常伴随着模拟量的应用，简单的说 PID 就是根据给定值和实际输出值构成控制偏差，将偏差按比例、积分和微分通过线性组合构成控制量，对被控对象进行控制。

PID 指令就是一种数字 PID 控制器。在 Kinco-K 系列 PLC 中可以调用 PID 指令来实现 PID 控制功能。在一个 CPU 中用户可以同时使用多达 8 路 PID。另外 PID 控制可以脱离与扩展模块的联系，扩展了该功能的灵活性。

注意，PID 运算所需时间较长，建议避免在中断程序中调用 PID 功能。

11.2 模拟量的使用

11.2.1 模拟量介绍

模拟量是在一定范围连续变化的量，也就是在一定范围内可以取任意值。

数字量是离散量，只能取几个分立值，如二进制数字变量只能取 0 和 1 两个值。

我们通常所说的 DI、DO 指的是数字量，AI、AO 指的是模拟量。

目前 KPLC 支持如下类型的模拟量输入信号：电流信号（0-20mA/4-20mA)、电压信号（0-10V/1-5V/-10V-+10V)、热电阻信号（Pt100/Cu50/Pt1000)、热电偶信号（J 型/K 型/E 型/S 型）；支持如下类型的模拟量输出信号：电流信号（0-20mA/4-20mA)、电压信号（0-10V/1-5V/-10V-+10V)。用户可以参考各个系列的硬件手册来选择需要的模块。

11.2.2 模拟量的信号形式、测量范围和表现形式

11.2.2.1 模拟量输入的信号形式、测量范围和表现形式

模拟量通道的输入信号经过 ADC 采样、线性计算后，计算结果作为测量值经过扩展总线送往 CPU 模块的 AI 映像区中以供用户程序访问。

各种信号形式均有一定的测量范围，若被测值超出测量范围，则模块将会报警，同时通过扩展总线向 CPU 模块发送故障报文。**建议用户将未用通道的端子短接起来，并在编程软件中将其信号形式设置为【0-10V】、【0-20mA】（如果是 RD 模块信号形式设置为电阻），那么这些未用通道将不会引起报警。**

下表是输入模拟量的测量范围和测量值表示格式，其中 I 代表输入电流值，V 代表输入电压值。比如信号形式为电流，读取到的模拟量数值为 9965，实际的对应的电流值为 9.965mA。

信号形式	测量范围	测量值表示格式
4-20mA	3.92-20.4mA	I×1000
0-20mA	0-20.4mA	
1-5V	0.96-5.1V	V×1000
0-10V	0-10.2V	

下表是 RD 模块测量范围和测量值表示格式，RD 模块可以接入热电阻（Pt100、Pt1000、Cu50）来测量温度，也可以直接测量电阻的阻值。各通道可以混合接入不同的电阻型号，并且支持两线制和三线制两种接线形式，具体接线请参考硬件手册。下表中 T 代表被测温度值，R 代表被测电阻值。比如信号形式为 Pt100，读取到的模拟量数值为 365，实际的对应的温度值为 36.5℃。

信号形式	测量范围	测量值表现形式
Pt100	-200~850℃	T*10
Cu50	-50~150℃	
Pt1000	-50~300℃	R*10
电阻	0-2000Ω	

下表是 TC 模块测量范围和测量值表示格式，TC 模块可以接入不同形式的热电偶，支持两线制和三线制两种接线形式，具体接线请参考硬件手册。下表中 T 代表被测温度值。

信号形式	测量范围	测量值表现形式
J 型	-210~1200℃	T*10
K 型	-270~1300℃	
E 型	-270~1000℃	
S 型	-50~1600℃	

11.2.2.2 模拟量输出的信号形式、测量范围和表现形式

用户程序中指定的 AQ 输出值首先经过扩展总线送到相应的 AO 模块中，然后经过计算、变换并通过 DAC 在指定的通道输出。

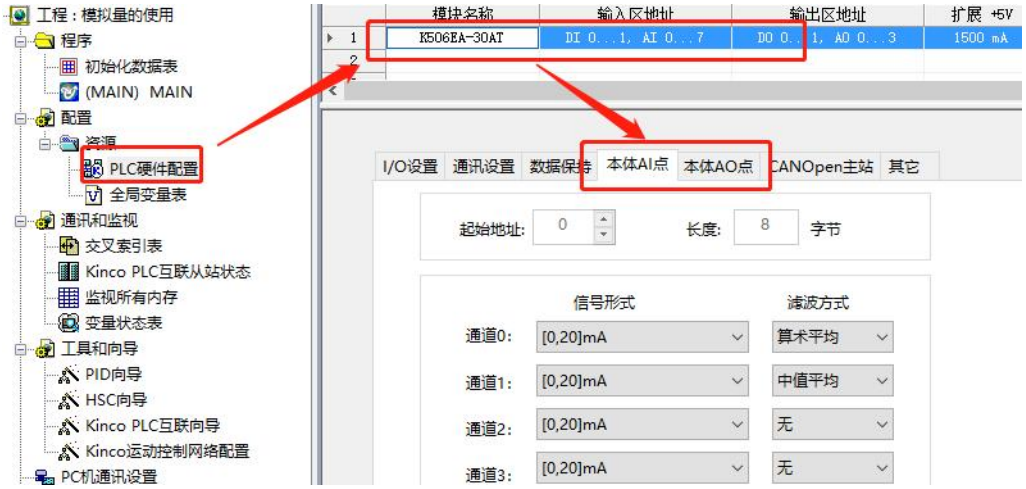
各种信号形式的输出范围是有限制的，若用户程序中指定的输出值超出了所选范围的上、下限，则的输出信号保持上、下限值不变。

下表是输出范围和输出值表示格式，其中 I 代表实际电流值，V 代表实际电压值。

信号形式	输出范围	用户程序中指定的输出值
4-20mA	3.92-20.4mA	I×1000
0-20mA	0-20.4mA	
1-5V	0.96-5.1V	V×1000
0-10V	0-10.2V	
-10-+10V	-10.2V-+10.2V	

11.2.3 模拟量在硬件配置中的设置

我们可以在 Kincobuilder 软件中 PLC 硬件设置中对 CPU 本体或者模拟量扩展模块模拟量信号的形式、滤波方式等进行选择。下图为支持电流、电压信号形式的配置示例：



➢ 地址

模拟量输入对应的内存区为%AIW 区、输出对应的%AQW 区，使用模拟量内存区的时候与硬件配置中起始地址和长度有关。

• **起始地址：**指定该模块在 AI 区中占用地址空间的起始字节地址（也就是第一个通道的地址）。

每个 AI 点在 AI 区占用 2 个字节。因此，该地址必须为偶数。

- **长度**：该模块占用地址空间的长度。这是一个固定值，取决于模块上 AI 通道的数量。

如上，模拟量输入的起始地址被指定为%AIW0，该 CPU 具有 4 个 AI 通道，因此它的 4 个通道的地址依次是%AIW0、%AIW2、%AIW4、%AIW6。模拟量输出的起始地址被指定为%AQW0，该 CPU 具有 2 个 AQ 通道，因此它的 2 个通道的地址依次是%AQW0、%AQW2。

➢ 通道设置

- **信号形式**：为各个通道选择输入信号的形式。采样值将在 CPU 内自动进行线性转换，关于数据转换格式请参见 11.2.3 模拟量的信号形式、测量范围和表现形式相关内容。
- **滤波方式**：为各个通道选择软件滤波器。

  对于变化较快的模拟量信号使用滤波器可以让测量值变得比较稳定。

  注意：若系统需要对某 AI 信号快速响应，那么就不应该启用该点的软件滤波器。

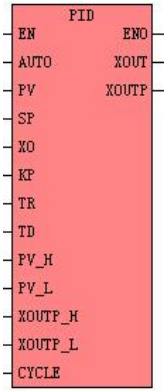
  软件滤波器的输入采样均采用了滑动方式。软件滤波器有如下选项：

  - ➢ 无 --- 不启用软件滤波器。
  - ➢ 算术平均 --- 对一定数量的信号采样值取算术平均值。
  - ➢ 中值平均 --- 将一定数量的信号采样值去掉最大、最小值后，对剩下的数取平均值。

11.3 PID 指令介绍

下述指令均位于【指令集】->【PID 回路】组中。

- 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	PID			☑ K5 ☑ K2 ☑ KS ☑ KW ☑ MK ☑ K6
IL	PID	PID AUTO, PV, SP, XO, KP, TR, TD, PV_H, PV_L, XOUTP_H, XOUTP_L, CYCLE, XOUT, XOUTP	U	

参数	输入/输出	数据类型	允许的内存区	描述
AUTO	输入	BOOL	I、Q、V、M、SM、L	手动/自动标志位。 0＝手动状态，1＝自动状态。
PV	输入	INT	AI、V	PV 值，即被控量的测量值。
SP	输入	INT	V	设定值，即被控量的目标值。
XO	输入	REAL	V	手动值，范围是 0.0～1.0。 在手动状态下，它直接是 PID 的输出值。
KP	输入	REAL	V	比例系数
TR	输入	REAL	V	积分时间，单位：秒。0 表示无积分。

TD	输入	REAL	V	微分时间，单位：秒。0 表示无微分。
PV_H	输入	INT	V	*PV* 值的上限
PV_L	输入	INT	V	*PV* 值的下限
XOUTP_H	输入	INT	V	XOUTP 值的上限
XOUTP_L	输入	INT	V	XOUTP 值的下限
CYCLE	输入	DINT	V	采样周期，单位：ms。 依据采样周期，PLC 循环对 PV 值进行采样并执行 PID 运算。
XOUT	输出	REAL:	V	PID 的输出值，范围是 0.0～1.0。
XOUTP	输出	INT	AQ、V	对 *XOUT* 进行线性化处理后的输出值。

该 PID 指令采用的是位置型算法，连续输出的控制方式。

- LD

若 EN 值为 1，则执行 PID 指令：每隔采样周期时间（*CYCLE*），PLC 就对 *PV* 值进行一次采样并进行 PID 运算、输出。

- IL

若 CR 值为 1，则执行 PID 指令：每隔采样周期时间（*CYCLE*），PLC 就对 *PV* 值进行一次采样并进行 PID 运算、输出。

PID 指令的执行不影响 CR 值。

➢ PID 详细使用说明

- 手动/自动状态

若手动/自动标志位 *AUTO* 的值为 0，则表示 PID 进入手动状态。在手动状态下，PID 指令不执行任何运算，直接将用户指定的手动值 *XO* 送至输出变量中。

若手动/自动标志位 *AUTO* 的值为 1，则表示 PID 进入自动状态。在自动状态下，PID 指令根据各输入参数的值进行正常的 PID 运算、输出。

在 PLC 控制系统正常运行时，应该让 PID 处于自动状态。

- PV 值和 SP 值的归一化

用户需要输入 *PV* 值、*SP* 值、PV 值的上限（*PV_H*）和 PV 值的下限（*PV_L*）参数。在进行 PID 运算之前，PLC 将依据这些参数自动对 *PV* 值和 *SP* 值进行归一化计算，从而方便了用户的使用。这些参数必须具有相同的量纲。

PV 值、*SP* 值参数均为 INT 型，用户可以灵活地输入与它们线性相关的数值。例如，假定对某个压力进行控制,测压所用压力变送器的量程为 0~40MPa,对应的输出为 4~20mA,自动控制的设定值为 25MPa；压力变送器的的输出直接接至 AI 模块的通道 AIW0（信号形式设置为 4~20mA，在 PLC 内的转换值为 4000~20000）。那么可以将 PID 指令的参数设置为：

	实际参数	描 述
PV	AIW0	AIW0 的测量值与实际压力值具有线性关系，因此可以直接作为 PV 值。
SP	14000	表示 14mA，因为 AIW0 测量 25MPa 得到的测量值为 14mA。
PV_L	4000	压力变送器的输出下限。
PV_H	20000	压力变送器的输出上限。

- PID 的输出值

PID 指令有两个输出值：*XOUT* 和 *XOUTP*。

XOUT 的范围是 0.0～1.0（也就是通常说的 0.0～100.0%）。

XOUTP 是根据用户指定的输出上限（*XOUTP_H*）和输出下限（*XOUTP_L*）进行线性化变换以后得到的整数值。*XOUTP* 的计算公式如下：

XOUTP ＝ （XOUTP_H － XOUTP_L ）× XOUT ＋ XOUTP_L

XOUTP 参数可以方便用户将 PID 的输出直接送至 AO 模块的某个通道。例如，假定 PID 的输出需要通过 AO 模块的通道 AQW0（信号形式设置为 4～20mA）送至某个调节阀，那么可以将 PID 指令的参数设置为：

	实际参数	描 述
XOUTP	AQW0	AQW0 直接控制调节阀,其值与阀门的开度具有线性关系,因此 AQW0 可以直接作为 PID 的输出参数。
XOUTP_L	4000	AQW0 的输出下限。
XOUTP_H	20000	AQW0 的输出上限。

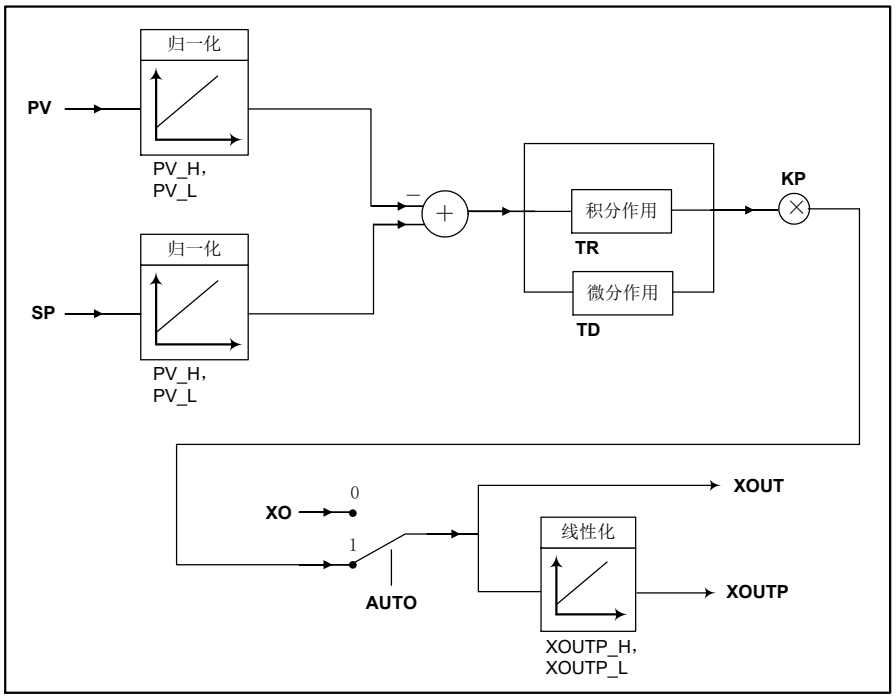
- 关于比例作用、积分作用和微分作用

比例作用是按照比例来反映被控量的偏差，一旦出现偏差，比例调节器将立即产生调节作用来减少偏差。用户通过 *KP* 参数来设置 PID 的比例系数。若 *KP* 大于 0，则 PID 是正作用；若 *KP* 小于 0，则 PID 是反作用。比例系数大，则调节速度快，但容易造成超调，使系统的稳定性下降。

积分作用能够消除被控量的偏差。只要存在偏差，就会进行积分调节，直至偏差消除。用户通过 *TR* 参数来设置 PID 的积分时间常数（若 *TR* 为 0 则表示取消积分作用）。积分时间常数越小，则积分作用就越强；积分时间常数越大，则积分作用就越弱。积分作用强也容易使系统的稳定性下降。积分作用通常与其它两种控制率相结合组成 PI 或者 PID 控制器。

微分作用反映了偏差的变化速率，能够预测偏差的变化趋势，在偏差变得很大之前产生一个有效的修正。因此微分作用有助于改善系统的动态性能，增加系统的稳定性。用户通过 *TD* 参数来设置 PID 的微分时间常数（若 *TD* 为 0 则表示取消微分作用）。微分时间常数越大，则微分作用就越强；微分时间常数越小，则微分作用就越弱。微分作用通常与其它两种控制率相结合组成 PD 或者 PID 控制器。一般在大滞后的系统中（比如温度控制）需要加入微分控制。

- PID 指令框图



11.4 PID 的使用方法

PID 的使用方法有两种：

1、使用相关指令进行编程：在程序中直接调用 PID 指令，设置好相应的参数进行编程即可。

2、使用 PID 向导设置：这种方式简单直观只需要按向导中提供的内容进行勾选设置参数即可，设置完向导自动生成相应的子程序，用户编程时直接调用即可，省去了 PID 指令调用繁琐的编程过程。

11.4.1 使用 PID 指令进行编程

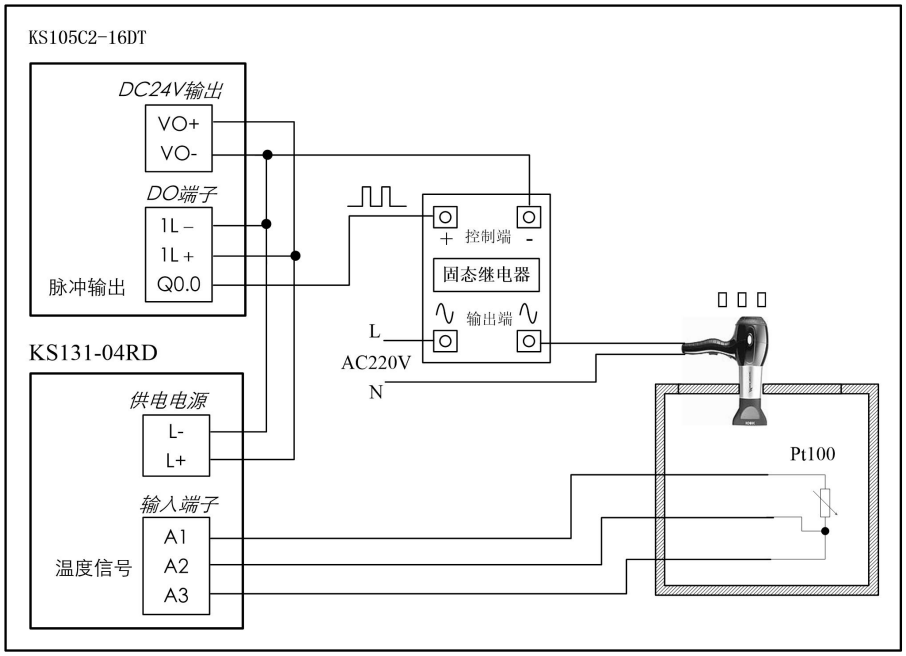
为了更好的帮助用户理解 PID 指令的使用方式，下面直接举例说明 PID 指令的使用。

11.4.1.1 PID 使用举例

我们建立如下模拟系统：对一个箱子内的温度进行控制，升温依靠电吹风加热，降温依靠自然冷却。PLC 的配置是：KS105C2-16DT +K S131-04RD。另外，SP、KP、TR、TD 等参数的修改通过一个 HMI 来完成。

在这个系统中，使用一个 Pt100 来测量温度，其信号接入 RD 模块的 AIW0 通道。Q0.0 输出脉冲串并接至固态继电器的控制端，利用 PID 的输出来相应调整脉冲宽度，从而控制电吹风的通断时间，实现温度的控制。

模拟系统如下图所示：

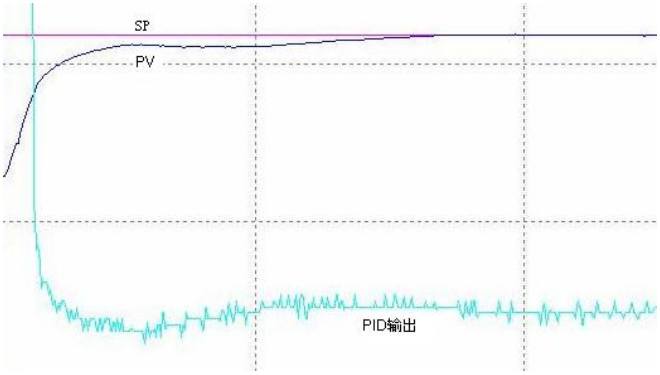


下面简单描述一下示例的编程思路：

1) 利用一个定时器实现了简单的脉宽调制输出。

2) 在程序中使用了如下控制方式：若 PV 值低于 SP 值超过 5℃，则使用 PID 手动方式并输出 100%；若 PV 值与 SP 值之间的偏差小于 5℃，则使用 PID 自动方式。这种方式的优点是：在偏差较大时

使用 PID 手动方式，避免了自动方式下由于积分累积值过大而造成的超调很大的问题；另外，也有利于减小调节时间。当 SP 设定为 50℃时，控制效果如下图。



程序如下：

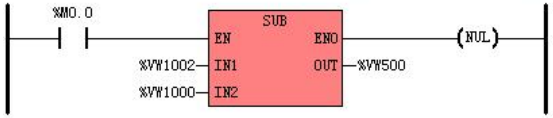
LD

```
(* Network 0 *)
(* 设定PV值、采样周期。
   采样周期是一个能够影响控制效果的重要参数，需要根据被控对象的实际特性来设定。 *)
```

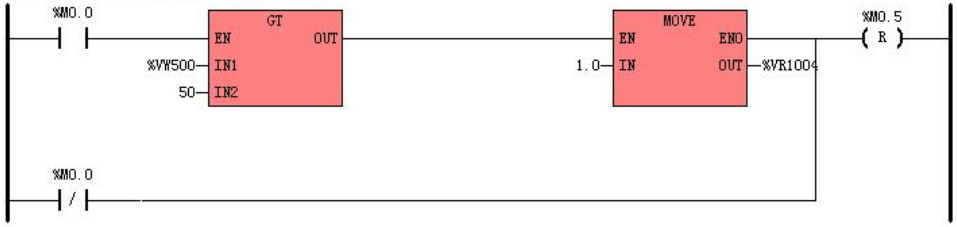

```
(* Network 1 *)
(* PV的最大、最小值 *)
```


```
(* Network 2 *)
(* XOUT_P的最大、最小值 *)
```


```
(* Network 3 *)
(* M0.0---HMI上的启动/停止自动控制按钮。
   控制方式：若PV值低于SP值超过5℃，则使用PID手动方式并输出100%；
             若PV值与SP值之间的偏差小于5℃，则使用PID自动方式。 *)
```

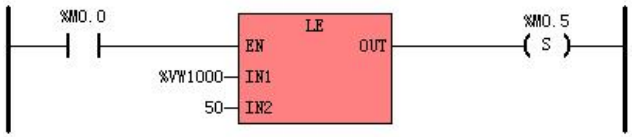



```
(* Network 4 *)
(* M0.5---PID手/自动方式参数 *)
```

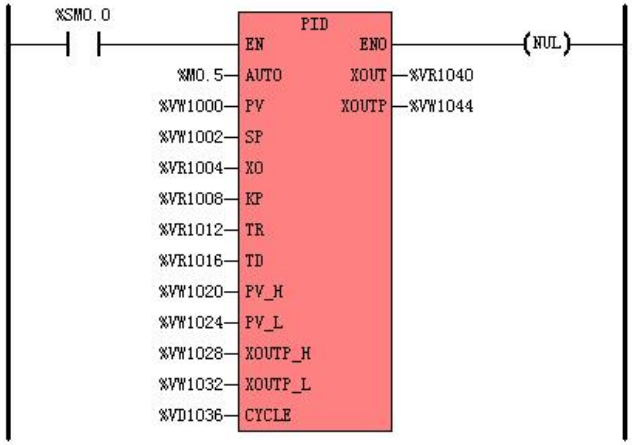


LD

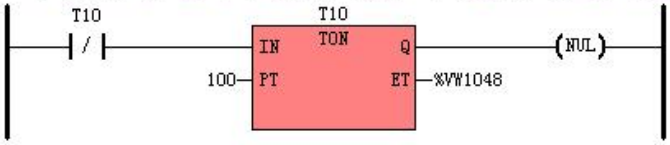
(* Network 5 *)



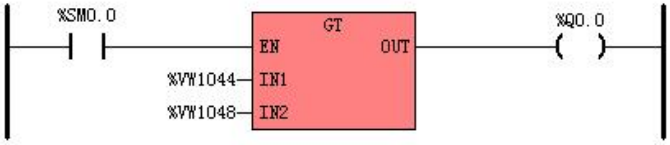
(* Network 6 *)
(* SP、X0、KP、TR、TD等参数需要通过人机界面来设定。 *)



(* Network 7 *)
(* 周期为1秒的脉冲串输出，利用PID的输出来调整脉冲宽度。 *)



(* Network 8 *)



IL

```
(* Network 0 *)
(*设定 PV 值、采样周期。*)
(*采样周期是一个能够影响控制效果的重要参数，需要根据被控对象的实际特性来设定。*)
LD      %SM0.0
MOVE    %AIW0, %VW1000
MOVE    DI#1000, %VD1036
(* Network 1 *)
(*PV 的最大、最小值*)
LD      %SM0.0
MOVE    1000, %VW1020
MOVE    0, %VW1024
(*XOUT_P 的最大、最小值*)
MOVE    100, %VW1028
MOVE    0, %VW1032
(* Network 3 *)
(*M0.0---HMI 上的启动/停止自动控制按钮。*)
(*控制方式：若 PV 值低于 SP 值超过 5℃，则使用 PID 手动方式并输出 100%；*)
(* 若 PV 值与 SP 值之间的偏差小于 5℃，则使用 PID 自动方式。*)
LD      %M0.0
MOVE    %VW1002, %VW500
SUB     %VW1000, %VW500
(* Network 4 *)
(*M0.5---PID 手/自动方式参数*)
LD      %M0.0
GT      %VW500, 50
MOVE    1.0, %VR1004
ORN     %M0.0
R       %M0.5
(* Network 5 *)
LD      %M0.0
LE      %VW1000, 50
S       %M0.5
```

11.4.2 使用 PID 向导

11.4.2.1 PID 向导设置及使用举例

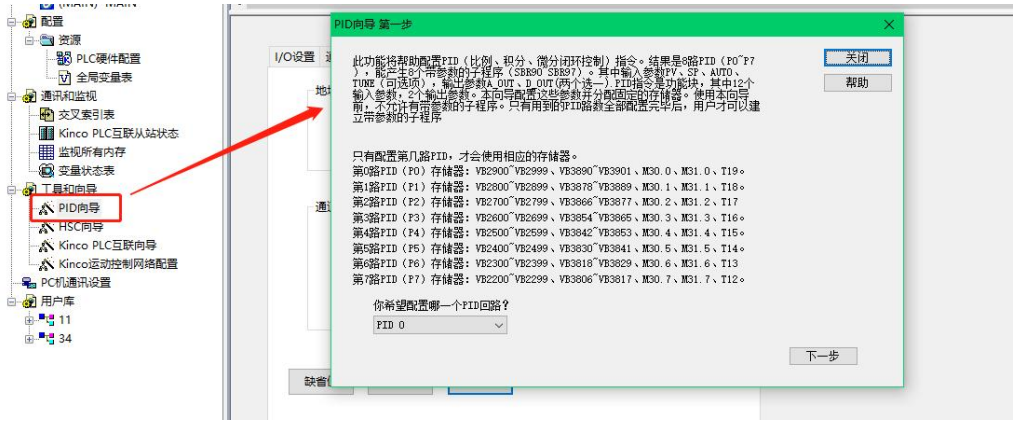
下面同样以 11.4.1.1 PID 使用举例中的模拟系统为例介绍 PID 向导的设置。

以下 PID 指令的参数请参考 11.3 PID 指令介绍章节的相关内容，在此不再详细介绍。

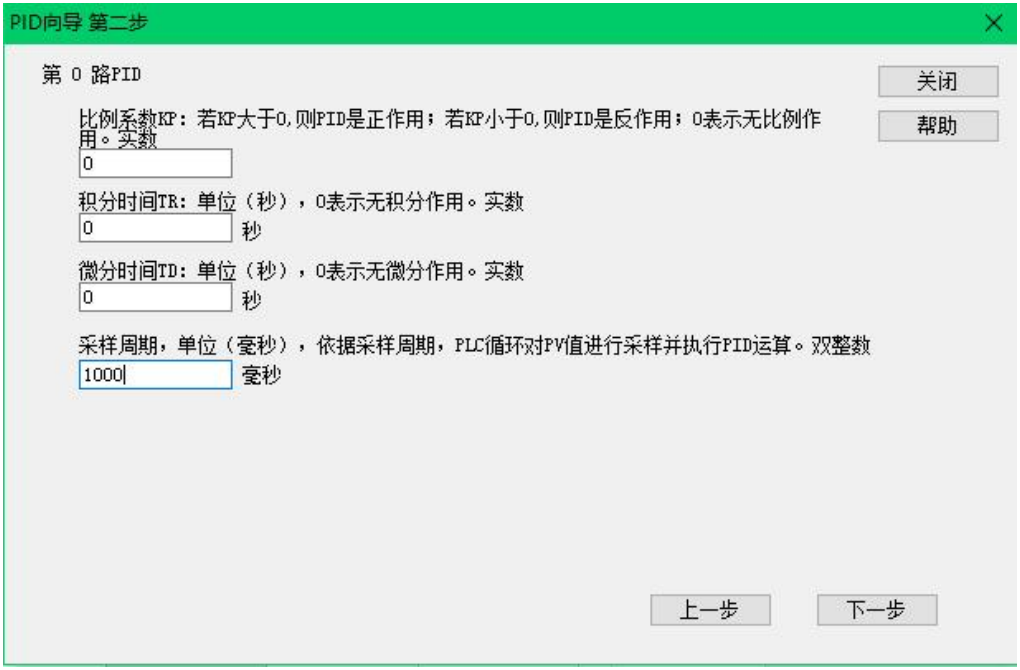
Kinco-K 系列 PLC 提供 8 路 PID 输出，设置 PID 需要占用一部分内存地址，下表为各路 PID 用到的部分内存地址（主要用于客户修改），自行设置的参数包括 PID 的输入端 X0、KP、TR、TD，分表代表手动输出值、P 比例、I 积分、D 微分。

PID 回路	手动值 X0	KP 比例	TR 积分时间	TD 微分时间
第 0 路 PID	%VR2916	%VR3898	%VR3894	%VR3890
第 1 路 PID	%VR2816	%VR3886	%VR3882	%VR3878
第 2 路 PID	%VR2716	%VR3874	%VR3870	%VR3866
第 3 路 PID	%VR2616	%VR3862	%VR3858	%VR3854
第 4 路 PID	%VR2516	%VR3850	%VR3846	%VR3842
第 5 路 PID	%VR2416	%VR3838	%VR3834	%VR3830
第 6 路 PID	%VR2316	%VR3826	%VR3822	%VR3818
第 7 路 PID	%VR2216	%VR3814	%VR3810	%VR3806

1、按向导提示选择想要使用的第几路 PID，设置 PID 需要占用一部分内存地址，选择不同的输出路数占用不同的内存地址，用户后面编程时需要注意这些被占用的内存区域不要再使用。



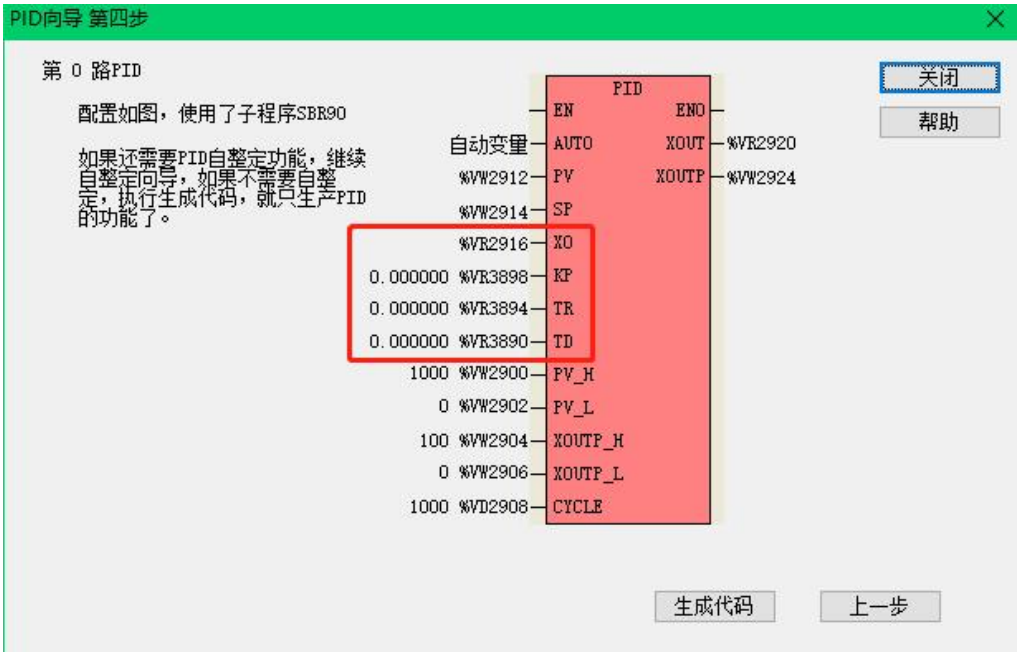
2、设置 PID 的采样周期和初始的 PID 参数，PID 参数的地址可以对应到 HMI 上方便后期调试调整参数用。

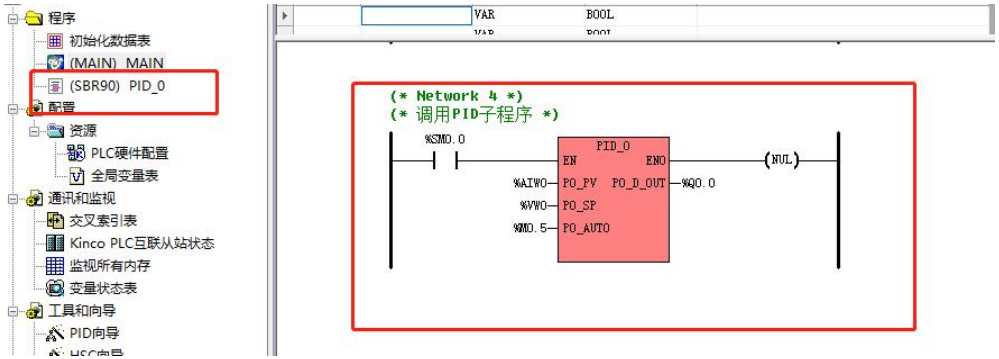


3、设置 PV 值的上下限，输出类型可以设置模拟量和数字量，模拟量即输出电压或电流的形式去控制外部的温度、压力、流量等被控对象，数字量输出指的以输出类似 PWM 波的形式，通过实时调节 PWM 波占空比的方式去控制外部的被控对象，占空比时间在此可以认为是 PWM 波的周期。

PV_L、PV_H、XOUTP_L、XOUTP_H 的设定值均为 PLC 内部值的表现形式，具体请参考 11.2.3 模拟量的信号形式、测量范围和表现形式相关内容。

4、向导生成的 PID 指令及其参数如下，确认无误后点击生成代码，软件中将生成相应的子程序如下图，我们需要自行设置的参数包括 PID 的输入端 X0、KP、TR、TD，分表代表手动输出值、P 比例、I 积分、D 微分，因为 8 路 PID 的具体对应的 PLC 地址不一样，所以请参考向导中生成的 PID 指令或者直接去生成的子程序中查看 PID 指令输入端的地址，以便将这些地址对应到 HMI 中方便调试。





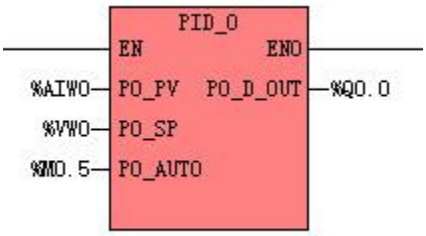
5、向导生成的 PID 子程序输入输出参数介绍：

PO_PV:外部被控对象的测量值。

PO_SP:被控对象的设定值。

PO_AUTO:PID 控制的手自动选择，0-手动，1-自动。

PO_D_OUT:PID 控制的输出（模拟量/数字量），与向导中的设置保持一致。



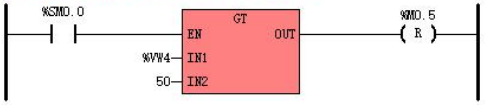
5、在MAIN主程序中编写相应的PID手自动切换程序并调用PID子程序PID_0即可实现和11.4.1.1 PID 使用举例中模拟系统同样的控温功能。

LD

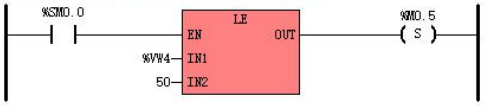
MAIN：

(* Network 0 *)
(* 求得测量的PV值与设定值SP的差值放入VW4中，VW0代表设定值 *)

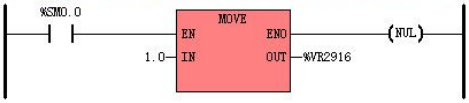
(* Network 1 *)
(* 当SP-PV值>5℃采用手动pid控制 *)



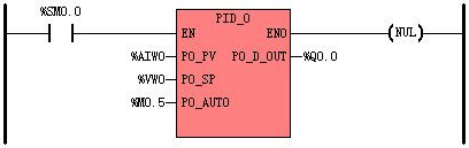
(* Network 2 *)
(* 当SP-PV值<5℃采用自动pid控制 *)



(* Network 3 *)
(* VR2916 手动PID的输出值，P、I、D参数都可以在HMI中给定，方便调试使用 *)



(* Network 4 *)
(* 调用PID子程序，M0.5手自动PID的选择使能位
Q0.0数字量输出用于产生PWM波以占空比的方式去控温 *)



第十二章 数据保持与数据备份

12.1 数据保持和数据备份

数据保持功能是 PLC 断电后利用后备电池来维持供电以保持 RAM 内存数据，因为操作的是内部 RAM，所以数据读写速度快，内存寿命也无限制。但是当电池电量耗尽后，数据将会全部丢失。

数据备份功能是 PLC 将数据保持至永久存储器中，可以永久保存，但永久存储器存在寿命和写入速度的限制，因此不能频繁、快速写入。

- **数据保持**

数据保持是指在 CPU 模块 RAM 中的数据在断电后保持为断电瞬间的状态，并供 CPU 在下一次上电的时候继续使用。CPU 模块内部均提供一个后备锂电池（不可充电）用于数据保持功能。在断电时，后备电池为 RAM 供电并保持 RAM 中的数据。用户需要使用 KincoBuilder 软件在用户工程的【PLC 硬件配置】中选择需保持的数据区类型（如 V 区、C 区等）及起止地址和长度，最大可以保持所有 V 区、C 区，如下图所示：

- **数据备份**

数据备份是指 CPU 模块在永久存储器中开辟一个区域，用于存放用户数据，该区域内的数据断电永久不会丢失，并供 CPU 在下一次上电的时候使用。**CPU 模块提供了 E2PROM 存储器用于数据备份功能，由于 E2PROM 只有 100 万次的写入寿命，因此用户注意尽量避免永久备份那些变化频繁的数据！**

K 系列 PLC 在 V 区中提供了数据备份区，该区域中的数据会自动写入永久存储器中，用户可以直接使用这些内存区域。各型号的 PLC 的数据保持区和数据备份区长度不完全相同，下表列出各型号的数据保持区和数据备份区的长度：

需要注意的是对于永久数据备份区域，若同时在【PLC 硬件配置】中也设置为数据保持，那么永久数据备份功能是优先的，也就是说永久备份存储器中的数据会覆盖电池供电的 RAM 中的数据。

因为 K 系列 PLC 在不断升级，由上表可以看出各 CPU 的数据备份长度不完全相同，在不改变程序的情况下为兼容之前的 K3、K5 系列等，那么需要在【PLC 硬件配置】中 CPU 模块的【其它】页面进行设置，如下图：

- 【兼容 K3 的永久存储设置】-都支持

选中此项则表示 VB3648-3092 将生效作为数据备份区，该区域中的数据会自动写入永久存储器中。

- 【兼容 K5 的永久存储设置】-除 K3 外都支持

选中此项则表示 VB3648-4095 将生效作为数据备份区，该区域中的数据会自动写入永久存储器中。

- 【兼容 K6 的永久存储设置】-只有 K6、KS101M、K209M 支持

选中此项则表示 VB15360-16383 将生效作为数据备份区，该区域中的数据会自动写入永久存储器中。

- 【上面所有的区域全部自动永久存储】-只有 K6 支持

选中此项则表示 VB3648-4095、VB15360-16383 同时生效作为数据备份区，该区域中的数据会自动写入永久存储器中。

- 【上面所有的区域全部不永久存储】-只有 K6 支持

选中此项则表示全部 V 区不设置有永久存储区。

- 【将整个用户工程备份到 PLC 的永久存储中】

K 系列默认在 PLC 中只保存用户工程中的硬件配置和用户程序信息，而变量名称（全局和局部）、程序名称、注释等均不保存。

若选中此项，则在 PLC 中保存完整的用户工程信息，包括硬件配置、用户程序、程序名称、所有变量名称、注释等。用户从 PLC 中上载后得到的工程将与下载的工程完全相同。

12.2 后备电池及电池电量指示

Kinco-K系列PLC使用特定规格的的锂电池作为后备电池。当断电时，后备电池用于给实时时钟供电来维持时钟的运行，同时也给RAM供电来进行数据保持。

后备电池可以拆卸，但需要开盖更换，打开外壳后即可看到如右图的电池，用户可以自行更换。电池为CR2032带连接器的3V锂电池，形状如右图，用户可以单独订购电池。

常温下，电池典型寿命为 5 年,断电电保持的时间累计不小于 3 年。

KPLC 在 SM 区中提供了如下寄存器用于指示“电池电量”的信息：

SM0.7	只读。SM0.7 为 TRUE 表示电池电压低， SM0.7 为 FALSE 表示电池电压正常。 **(适用于除 K5 之外的其它系列)**
SMW10	只读。存放后备电池的电压值，单位：0.01V。**（仅 K5 支持）** 若后备电池的电源持续低于 2.6V 时，PLC 会产生“后备电池带电量低”报警。

第十三章 故障处理

Kinco-K 系列 PLC 将运行时发生的错误分为三个等级：致命错误、严重错误、一般错误。当 PLC 在运行时发生错误时，会根据错误的等级采取不同的处理措施，同时将错误码根据发生的先后次序依次存储下来，以供用户读取并进行分析。

无论曾经发生过的错误等级是多少，我们都建议用户在发现有错误指示之后，及时进行分析、检查，以免造成不必要的损失！

只有当 PLC 能够运行时，PLC 才能够检测、分析各种发生的错误！如果 PLC 硬件被严重损坏，导致其根本无法运行各种软件程序，则所有的功能都会失效。

13.1 错误类型

13.1.1 致命错误

致命错误发生的原因是 PLC 检测到硬件芯片发生了未知的、可能危及正确运行的故障，或者 PLC 的看门狗被触发（比如程序中 FOR 循环量过大、JMP 指令造成了死循环）等等。致命错误可能会导致 PLC 无法继续工作，并且我们也无法确认 PLC 是否会再次发生不可预期的、危险的错误，因此，处理致命错误的目标是立即让 PLC 进入安全状态。

当发生致命错误后，PLC 会自动采取如下措施：立即退出正常的扫描状态，并根据 SM2.0 的值来直接复位或者进入独立的安全子系统运行：若 SM2.0 值为 0，则发生致命错误时，PLC 会进入独立的安全子系统运行；若 SM2.0 值为 1，则发生致命错误时，PLC 直接复位重新启动。**SM2.0 上电后默认为 0，建议用户非必要时不去修改它的值，让 PLC 能够进入安全子系统！**

13.1.1.1 关于安全子系统

当发生致命错误后，PLC 已无法正常运行，此时若 SM2.0 的值为 0，PLC 会自动进入一个独立的安全子系统运行。在安全子系统里，PLC 已不能执行正常的扫描，但是会让所有的输出通道输出用户指

定的【停机保持】值，同时提供一些方法以便用户能尽量清除软件方面的错误。

➢ **状态指示**

进入安全子系统后，PLC 的指示灯会用特殊状态进行指示。

下表列出了各系列 PLC 在安全子系统下各指示灯的状态，用户可以根据这些灯的状态来判断 PLC 是否因致命错误而进入了安全子系统中运行。

系列	指示灯			
	Run	Stop	Comm	Err
K6/K2/K5	常灭	常亮	常灭	闪烁
KS/KW	常灭	--	--	闪烁
MK 系列	闪烁	--	--	闪烁

注：“--”表示没有这个指示灯或者忽略其状态。

➢ **功能**

当进入安全子系统后，PLC 提供了如下功能：

- 所有的输出点（DO、AO）立即输出用户在“PLC 硬件配置”中定义的“停机输出”值。
- PORT1（RS485）能够工作，可以接收特殊的“清除”命令来清除 PLC 内的数据，从而让 PLC 恢复至出厂状态。此时 PORT1 的通信参数为：波特率 9600，无校验，8 数据位，1 停止位。
- 控制各指示灯的状态（如上面所述），通知用户 PLC 刚出现了致命错误。

13.1.1.2 如何解决致命错误

致命错误发生的可能原因及解决方案如下：

1) 关键的硬件芯片（比如 MCU、RAM 等）被损坏，导致 PLC 完全无法正常运行。如果是这种原因引发的致命错误，那么用户无法自己解决，只能将 PLC 返厂维修。

2) PLC 系统程序存在 bug，如果在应用过程中该 bug 被触发，则可能会导致 PLC 发生致命错误。解决方法：联系步科公司，由步科解决 PLC 系统程序的 bug，并将解决之后的新版本程序提供给用户，用户可以自行将故障 PLC 升级为新版本系统程序，升级方法请参阅附录 B 更新系统程序；

或者在确定了触发 bug 的原因之后，用户修改程序，避免使用相关的功能，然后将新程序下载至 PLC 中运行。

3) 在用户程序内存在严重的错误，比如 FOR 循环量过大、调用 JMP 指令错误而造成了死循环等等而引发的看门狗故障；使用指针时操作了非法的地址；使用某些需要操作内存块的指令（如 S_BLK、BLKMOVE、ROL_BLK、MBUSR 等）时参数值不合理，导致访问了非法的内存地址等等。解决方法：确定了原因之后，用户改正程序中的错误并重新下载至 PLC 中运行。

当发生致命错误之后，PLC 会自动进入安全子系统运行，用户无论要更新系统程序还是修改用户程序之后重新下载，都需要先让 PLC 脱离安全子系统并进入正常的系统运行过程中。

用户可以清除 PLC 内的所有数据，让 PLC 恢复至出厂初始状态，然后 PLC 即可正常运行，用户就可以将修改好的用户程序下载至 PLC 中运行或者将 PLC 更新为新版本系统程序。请参阅 13.5 如何将 CPU 恢复至出厂的初始状态？ 了解更多的信息。清除 PLC 有如下几种操作方法：

13.1.1.2.1 对于具有“RUN/STOP”开关的 PLC

用户可以先为 PLC 断电，将 RUN/STOP 开关拨至 STOP 位置，然后为 PLC 重新上电。上电后用户需要先观察 PLC 的指示灯状态，按如下情况分别进行处理：

- 如果指示灯依然指示上面所述的安全子系统的状态，则用户可以先尝试使用下面所述的第 3 种方法，即在 KincoBuilder 软件中，使用【工具】→【清除（仅用于致命错误时）】菜单命令来清除 PLC。如果清除失败，则表明本次致命错误可能是由核心芯片故障引起的，因此用户无法咨询解决。
- 如果指示灯状态为：RUN 灯熄灭；若有 STOP 灯的话，则 STOP 灯同时常亮；ERR 灯常亮或者常灭。这种指示灯状态表明 PLC 已经正常进入了 STOP 状态，本次致命错误是由软件引起的，因此用户可以修复。此时 PLC 各通信口的参数如下表：

通信口	通信参数
USB	虚拟为串口，在 Windows 中保持之前的串口号不变。 【PLC 站号】为 1，其它通信参数可忽略。

以太网口	所有的通信参数（包括 IP 地址、端口号等）都保持之前的值不变。
串行通信口	第 1 个串口（若标配有 PORT0，则是 PORT0；若没有 PORT0，则是 PORT1）的参数被 PLC 自动设置为：波特率 9600，无校验，8 数据位，1 停止位。

用户可任选一个通信口将 PLC 和电脑连接起来，按上表中的参数将电脑相应通信口的参数配置好，然后在 KincoBuilder 软件中执行【PLC】→【清除】菜单命令，将 PLC 恢复至出厂状态，然后重新断电重启 PLC 即可。

13.1.1.2.2 对于具有“CLR”清除按钮的 PLC

KPLC 有部分系列产品（包括 MK 系列）提供了 CLR 清除按钮。在安全子系统中，用户可以直接操作 CLR 按钮来清除 PLC 内的所有数据并恢复至出厂状态。

下面以 MK 系列一体机为例来完整描述清除的步骤：

① 第 1 步：按下 CLR 按钮，并保持 5 秒钟以上，ERR 指示灯将开始以约 2 秒钟的周期闪烁。

② 第 2 步：然后松开 CLR 按钮，约 2 秒钟后，ERR 指示灯将开始以约 600ms 的周期快速闪烁。此时 PLC 处于等待状态，用户需要在 60 秒钟内进行下一步操作，否则 PLC 将结束本次过程并恢复到操作 CLR 按钮之前的状态。

③ 第 3 步：按下 CLR 按钮，并保持 5 秒钟以上，然后 PLC 将进入清除过程，此时 ERR 灯常亮，用户可以松开按钮。不同系列的 PLC，清除过程所需的时间会有差别，通常需要几秒钟至约 20 秒钟。在清除过程中，PLC 不会再响应任何其它命令。当清除过程结束后，如果清除成功，则 ERR 灯会以 1 秒钟的周期闪烁 2 次，然后熄灭；如果清除失败，则 ERR 灯会以 600ms 周期闪烁 4 次，然后熄灭。

④ 第 4 步：将 PLC 断电重启，PLC 将恢复至出厂状态并运行。

13.1.1.2.3 对于所有的 PLC

在安全子系统中，所有系列 PLC 的 PORT1（RS485）口都会启动工作，用户可以利用 PORT1 口来清除 PLC 内的数据。

在 KincoBuilder 软件中，用户执行【工具】→【清除（仅用于致命错误时）】菜单命令，将弹出

如下图所示的窗口，用户按界面上的提示信息一步步操作即可。

13.1.2 严重错误

严重错误会导致 PLC 无法执行某项或者几项重要功能，不能继续正确运行用户程序，但是其结果是可以预期的。当发生严重错误时，PLC 会自动采取如下措施：

1) 立即将 PLC 置于 STOP 状态，所有输出点（DO、AO）立即输出用户设置的“停机输出”值。
2) ERR 指示灯常亮，RUN 指示灯熄灭，如果有 STOP 指示灯的话则 STOP 灯常亮。
3) 依次记录故障代码，并允许用户通过 KincoBuilder 和 Modbus RTU 协议读取这些记录。

13.1.3 一般错误

一般错误是 PLC 执行某项功能时发生错误，但 PLC 能够容忍这种错误，可以继续正确运行其它部分用户程序，其结果是可以预期的。当发生一般错误时，PLC 会自动采取如下措施：

1) PLC 继续运行。
2) ERR 指示灯常亮。
3) 依次记录故障代码，并允许用户通过 KincoBuilder 和 Modbus RTU 协议读取这些记录。

13.2 错误代码

代码	描述

严重错误	
20	在“PLC 硬件配置”中的 CPU 类型与实际连接的 CPU 类型不一致。
21	在“PLC 硬件配置”中存在错误的扩展模块（使用了其它系列的模块）。
25	上电时，读取 PLC 保护类型失败。
26	上电时，读取目标文件（明文）失败。
27	上电时，读取目标文件（密文）失败。
28	上电时，目标文件 CRC 校验错误。
29	上电时，检查 PLC 程序中有未知指令。
30	上电时，检查 PLC 程序的参数个数超过限制。
35	上电时，读取永久存储区数据失败。
40	运行时，JMP 指令跳转失败。
41	运行时，子程序调用失败。
42	运行时，中断子程序调用失败。
60	上电时，第 1 个扩展模块超时没有响应。
61	上电时，第 1 个模块响应错误。
62	第 1 个扩展模块与硬件配置中的类型不一致。
65	上电时，第 2 个扩展模块超时没有响应。
66	上电时，第 2 个模块响应错误。
67	第 2 个扩展模块与硬件配置中的类型不一致。
70	上电时，第 3 个扩展模块超时没有响应。
71	上电时，第 3 个模块响应错误。
72	第 3 个扩展模块与硬件配置中的类型不一致。
75	上电时，第 4 个扩展模块超时没有响应。
76	上电时，第 4 个模块响应错误。
77	第 4 个扩展模块与硬件配置中的类型不一致。
80	上电时，第 5 个扩展模块超时没有响应。
81	上电时，第 5 个模块响应错误。
82	第 5 个扩展模块与硬件配置中的类型不一致。
85	上电时，第 6 个扩展模块超时没有响应。
86	上电时，第 6 个模块响应错误。

87	第 6 个扩展模块与硬件配置中的类型不一致。
90	上电时，第 7 个扩展模块超时没有响应。
91	上电时，第 7 个模块响应错误。
92	第 7 个扩展模块与硬件配置中的类型不一致。
100	上电时，第 8 个扩展模块超时没有响应。
101	上电时，第 8 个模块响应错误。
102	第 8 个扩展模块与硬件配置中的类型不一致。
105	上电时，第 9 个扩展模块超时没有响应。
106	上电时，第 9 个模块响应错误。
107	第 9 个扩展模块与硬件配置中的类型不一致。
110	上电时，第 10 个扩展模块超时没有响应。
111	上电时，第 10 个模块响应错误。
112	第 10 个扩展模块与硬件配置中的类型不一致。
115	上电时，第 11 个扩展模块超时没有响应。
116	上电时，第 11 个模块响应错误。
117	第 11 个扩展模块与硬件配置中的类型不一致。
120	上电时，第 12 个扩展模块超时没有响应。
121	上电时，第 12 个模块响应错误。
122	第 12 个扩展模块与硬件配置中的类型不一致。
95	上电时，CPU 模块发送扩展通信报文失败。
96	上电时，CPU 模块扩展总线进入错误被动状态。
97	上电时，CPU 模块扩展总线进入总线关闭状态。
一般错误	
136	上电时，AI 通道的校准值读取失败。
137	上电时，AO 通道的校准值读取失败。
138	校准时，AI 通道的校准值写入失败。
139	校准时，AO 通道的校准值写入失败。
150	运行时，第 1 个扩展模块心跳超时，该模块的扩展总线通信可能中断。
151	运行时，收到第 1 个扩展模块的故障报文。
154	运行时，第 2 个扩展模块心跳超时，该模块的扩展总线通信可能中断。

155	运行时，收到第 2 个扩展模块的故障报文。
158	运行时，第 3 个扩展模块心跳超时，该模块的扩展总线通信可能中断。
159	运行时，收到第 3 个扩展模块的故障报文。
162	运行时，第 4 个扩展模块心跳超时，该模块的扩展总线通信可能中断。
163	运行时，收到第 4 个扩展模块的故障报文。
166	运行时，第 5 个扩展模块心跳超时，该模块的扩展总线通信可能中断。
167	运行时，收到第 5 个扩展模块的故障报文。
170	运行时，第 6 个扩展模块心跳超时，该模块的扩展总线通信可能中断。
171	运行时，收到第 6 个扩展模块的故障报文。
174	运行时，第 7 个扩展模块心跳超时，该模块的扩展总线通信可能中断。
175	运行时，收到第 7 个扩展模块的故障报文。
178	运行时，第 8 个扩展模块心跳超时，该模块的扩展总线通信可能中断。
179	运行时，收到第 8 个扩展模块的故障报文。
182	运行时，第 9 个扩展模块心跳超时，该模块的扩展总线通信可能中断。
183	运行时，收到第 9 个扩展模块的故障报文。
186	运行时，第 10 个扩展模块心跳超时，该模块的扩展总线通信可能中断。
187	运行时，收到第 10 个扩展模块的故障报文。
190	运行时，第 11 个扩展模块心跳超时，该模块的扩展总线通信可能中断。
191	运行时，收到第 11 个扩展模块的故障报文。
194	运行时，第 12 个扩展模块心跳超时，该模块的扩展总线通信可能中断。
195	运行时，收到第 12 个扩展模块的故障报文。
300	运行时，本体 AI 通道曾经发生过 DMA 错误。
301	运行时，本体 AI 通道的采样转换过程曾经停止过。
320	运行时，扩展总线通信曾经发生过帧格式错误。
321	运行时，扩展总线通信曾经进入错误错误主动状态。
322	运行时，扩展总线通信曾经进入错误错误被动状态。
323	运行时，扩展总线曾经关闭过。注：总线关闭是由于总线上通信错误过多造成的。
324	运行时，扩展通信故障：接收缓冲区满。
325	运行时，扩展通信故障：发送缓冲区满。
326	运行时，扩展通信故障：发送报文失败。
327	运行时，检测到 CANOpen 从站故障（心跳超时或者节点保护超时、SDO 无回应等）。

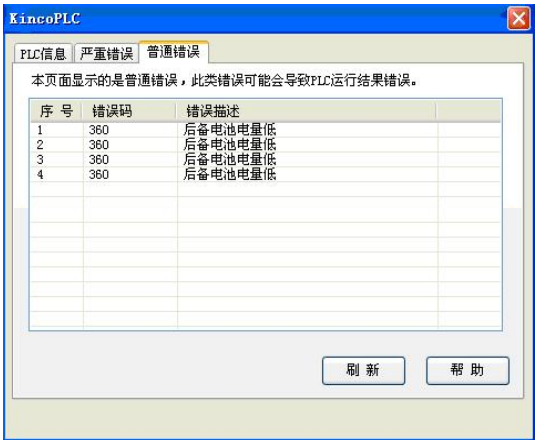
329	运行时，发生过如下错误：被 0 除。
330	运行时，发生过如下错误：类型转换指令（I_TO_B、DI_TO_I）溢出。
331	运行时，发生过如下错误：LN 指令输入值为 0 或者负数。
332	运行时，发生过如下错误：LOG 指令输入值为 0 或者负数。
333	运行时，发生过如下错误：SQRT 指令输入值为负数。
334	运行时，发生过如下错误：I_TO_BCD 指令无效的输入值。
335	运行时，发生过如下错误：A_TO_H 指令无效的输入值。
336	运行时，发生过如下错误：R_TO_A 指令无效的输入值。
341	运行时，发生过如下错误：FOR 指令无效的输入值。
350	运行时，发生过如下错误：保存永久存储数据失败。
351	上电时，检测到 RAM 中的掉电保持数据丢失。
360	后备电池电量低报警。

13.3 如何读取 PLC 中曾经发生的错误

当 PLC 发生错误之后，会依次自动记录下错误代码，用户可以利用如下方法来读取一般错误和严重错误信息。致命错误的处理方法请参阅 13.1.1.2 如何解决致命错误。

1) 使用 KincoBuilder

用户可以在 Kincobuilder 中执行【PLC】->【PLC 严重错误…】或者【PLC 普通错误…】菜单命令，在对话框窗口中查看，如右图。在对话框中，用户可以单击【刷新】按钮，刷新显示 PLC 内最新的错误记录信息。



2) 使用 Modbus RTU 通信

用户可以使用 Modbus RTU 通信命令（功能码 3、4），通过 Port0、Port1 或者 Port2 通信口来读取错误记录信息。用户读取到的是 PLC 记录的错误码，以字的形式来读取，一次可以读取一条或者多条记录。

错误记录的 Modbus 寄存器信息如下表：

Modbus 寄存器	描述
9000-9127	PLC 本次上电后，最近发生的 128 个一般错误的代码。 其中，9000 为最新一次的错误，9001 为次新的错误，依此类推。
9128-9255	PLC 本次上电后，最近发生的 128 个严重错误的代码。 其中，9128 为最新一次的错误，9129 为次新的错误，依此类推。
9256-9383	PLC 在上一次上电过程中，最后发生的 128 个一般错误的代码。 其中，9256 为最新一次的错误，9257 为次新的错误，依此类推。
9384-9511	PLC 在上一次上电过程中，最后发生的 128 个严重错误的代码。 其中，9384 为最新一次的错误，9385 为次新的错误，依此类推。

13.4 错误寄存器

针对错误处理，K 系列 PLC 在 SM 区提供了一个控制字节和各种错误状态寄存器。当 PLC 发生错误时会自动设置相应错误寄存器的值。用户程序可以直接读取这些寄存器值并进行相应的处理。

- SMB2：控制字节

位（可读写）	功能描述
SM2.0	其值决定了当发生致命错误时 PLC 的动作。上电后初始值为 0。 若值为 0：发生致命错误时，PLC 进入独立的安全子系统运行。 若值为 1：发生致命错误时，PLC 直接复位。

- SMB0 和 SMB1：内存、指令错误

SM	功能描述
SMB0（只读）	
SM0.2	若上电后 PLC 检测到 RAM 中的掉电保持数据丢失，则将该位置 1，否则置 0。
SMB1（只读）	
SM1.0	1 表示发生过如下错误：DIV、MOD 指令被 0 除。
SM1.1	1 表示发生过如下错误：LN、LOG、SQRT 指令参数为非法值（0 或者负数）。

SM1.2	1 表示发生过如下错误：I_TO_B、DI_TO_I 指令转换结果溢出。
SM1.3	1 表示发生过如下错误：I_TO_BCD 指令无效的输入 BCD 码。
SM1.4	1 表示发生过如下错误：A_TO_H 指令输入字符串中有不可识别的字符。
SM1.5	1 表示发生过如下错误：R_TO_A 指令转换结果溢出。
SM1.6	1 表示发生过如下错误：FOR 指令输入参数值无效。

- SMB3，SMB5 和 SMB96-SMB110：扩展模块故障(不支持扩展的 CPU 不支持）

若 PLC 检测到扩展总线通信故障或者收到扩展模块发送的故障报文，则 PLC 会设置相应的扩展扩展总线故障标志位，同时将故障代码存放在相应的扩展总线故障寄存器中供查询。若没有检测到故障，则故障标志位和故障寄存器被设置为 0。

注意：若 PLC 在启动过程中检测到扩展总线或者扩展扩展模块故障，那么将进入 STOP 状态，同时点亮 ERR 灯，但不设置相应的寄存器，因为此时 CPU 不执行用户程序，就无法读取寄存器值。

SM	功能描述
SMB3，SMB5（只读）：扩展总线错误标志	
SMB3.0	若第 1 个扩展模块发生故障，则该位被置 1。
SMB3.1	若第 2 个扩展模块发生故障，则该位被置 1。
SMB3.2	若第 3 个扩展模块发生故障，则该位被置 1。
SMB3.3	若第 4 个扩展模块发生故障，则该位被置 1。
SMB3.4	若第 5 个扩展模块发生故障，则该位被置 1。
SMB3.5	若第 6 个扩展模块发生故障，则该位被置 1。
SMB3.6	若第 7 个扩展模块发生故障，则该位被置 1。
SMB3.7	若第 8 个扩展模块发生故障，则该位被置 1。
SMB5.0	若第 9 个扩展模块发生故障，则该位被置 1。
SMB5.1	若第 10 个扩展模块发生故障，则该位被置 1。
SMB5.2	若第 11 个扩展模块发生故障，则该位被置 1。
SMB5.3	若第 12 个扩展模块发生故障，则该位被置 1。
SMB5.7	若 CPU 检测到扩展总线通信故障，则该位被置 1。
SMB96 - SMB110（只读）：扩展总线错误代码	

SMB96	若第 1 个扩展模块发生故障，则存放其故障码。含义见表 1。
SMB97	若第 2 个扩展模块发生故障，则存放其故障码。含义见表 1。
SMB98	若第 3 个扩展模块发生故障，则存放其故障码。含义见表 1。
SMB99	若第 4 个扩展模块发生故障，则存放其故障码。含义见表 1。
SMB100	若第 5 个扩展模块发生故障，则存放其故障码。含义见表 1。
SMB101	若第 6 个扩展模块发生故障，则存放其故障码。含义见表 1。
SMB102	若第 7 个扩展模块发生故障，则存放其故障码。含义见表 1。
SMB103	若第 8 个扩展模块发生故障，则存放其故障码。含义见表 1。
SMB104	若第 9 个扩展模块发生故障，则存放其故障码。含义见表 1。
SMB105	若第 10 个扩展模块发生故障，则存放其故障码。含义见表 1。
SMB106	若第 11 个扩展模块发生故障，则存放其故障码。含义见表 1。
SMB107	若第 12 个扩展模块发生故障，则存放其故障码。含义见表 1。
SMB110	若 CPU 端的扩展总线发生故障，则存放其故障码。含义见表 2。

故障码	描 述
0	无故障。
6	在运行过程中，扩展模块心跳报文超时。扩展模块会定时向 CPU 模块发送心跳报文，若 CPU 超时未收到该模块的心跳报文，则说明该模块可能通信异常。
10	AI 通道（电压、电流输入）的 ADC 转换出现一次错误。
11	模块保存校准值错误。
12	模块读取校准值错误。
14	模拟量输入模块第 1 个通道输入信号超出设定的测量范围。
15	模拟量输入模块第 2 个通道输入信号超出设定的测量范围。
16	模拟量输入模块第 3 个通道输入信号超出设定的测量范围。
17	模拟量输入模块第 4 个通道输入信号超出设定的测量范围。

表 1 扩展模块故障码含义

故障码	描 述
0	无故障。
1	通信帧格式错误。

2	扩展总线进入错误报警状态。
3	扩展总线进入错误被动状态。
4	扩展总线进入了总线关闭状态，并刚刚自动恢复。
5	扩展总线接收缓冲区满。
6	扩展总线发送缓冲区满。
7	CPU 发送报文失败。

表 2 CPU 模块扩展总线通信故障码含义

13.5 如何将恢复至出厂的初始状态？

用户可以清除 PLC 内的所有数据，让 PLC 恢复至出厂初始状态。清除方法有如下几种：

1) 对于所有系列的 PLC，用户在 KincoBuilder 编程软件中执行【PLC】→【清除…】菜单命令，即可清除 CPU 模块内的所有数据，包括用户程序、配置数据、密码等，从而将 CPU 恢复成出厂的初始状态。

2) 对于具有“CLR”清除按钮的 PLC，比如 MK 系列一体机，在安全子系统中，用户可以直接使用 CLR 硬件按钮来进行清除。具体的操作方法请参阅 13.1.1.2.2 对于具有“CLR”清除按钮的 PLC。

清除完成后，CPU 内部状态如下：

- 之前的用户程序、所有内存区（包括数据保持和数据备份区）、密码、用户使用 COM_WPARAS 和 LORA_WPARAS 指令设置的通信口参数、用户使用 UNID_W 指令写入的特殊存储区数据等等，全部被清除。
- 根据本机型号，CPU 在内部自动生成一个“空工程”，其中所有串行通信口的参数被配置为：PLC 站号为 1，波特率 9600，无校验，数据位 8 位，停止位 1 位。
- 原则上，用户设置的以太网通信口参数保持不变。在清除过程完成后，CPU 将会检查之前存储的以太网通信参数，如果是合法的数据，则保持不变；如果数据非法，则 CPU 将参数自动配置为出厂默认值：IP 地址为 192.168.0.252，子网掩码为 255.255.255.0，网关地址为 192.168.0.1，端口为 502。
- 实时时钟值（日期和时间）仍然保持不变。

第十四章 知识产权保护

14.1 密码保护

Kinco-K 系列 PLC 提供了密码保护功能，允许用户通过对 CPU 进行加密来限制特定功能的使用。

KPLC 密码长度允许 8-14 个字符，可以由数字、字母、下划线组成，字母区分大小写。

若用户将 PLC 保护等级设置为“等级 3：最高保护”并设置了密码， 或者启用了“禁止上载”功能，PLC 存储的用户程序将全部进行加密并以密文的形式进行存储，以防止被用直接读芯片数据的方法破解。

若 CPU 被加密，则用户在使用受限功能之前会被要求输入密码。此时若用户输入的密码正确，则 CPU 允许该操作；若用户输入的密码有误，则 CPU 将禁止该操作。输入的密码只对当次操作有效，以后再使用受限功能时仍然需要输入密码。密码输入错误次数受到限制，当密码输入错误到达限定次数后，必须重启 PLC 才能继续执行带密码的操作，以避免暴力破解。

14.1.1 保护等级

CPU 的密码保护提供了如下 3 个等级：

- **等级 1**：无保护。对 CPU 功能的使用没有限制。这是缺省的等级。
- **等级 2**：部分保护。用户在执行下载功能时需要提供密码。
- **等级 3**：最高保护。用户在执行上载、下载功能时需要提供密码。设置密码后，PLC 存储的用户程序将全将全字节进行加密，以密文的形式进行存储，以防止被用直接读硬件的方法破解。

14.1.2 修改密码和保护等级

执行【PLC】→【**修改密码**...】菜单命令，就可以进入“CPU 密码保护”窗口中进行修改。如下图：

1） 旧密码验证

若所连 CPU 原来已经被设置了密码保护，则必须在此正确输入原有的旧密码以供验证。

若原来没有设置过密码保护，则让输入框保持为空即可。

2） 新保护等级和新密码：在此可以为所连 CPU 设置新的保护等级和密码。

- **保护等级**：可以从等级 1、等级 2、等级 3 中任选一种。
- **新密码**：在此输入新的密码。

  密码允许 8-14 个字符，可以由数字、字母、下划线组成，字母区分大小写。

  该输入框当保护等级选择为等级 2 或者等级 3 时生效。
- **新密码确认**：在此用户需要再次输入新的密码。

用户全部完成上述设置后，单击【**应用**】按钮，新的设置就将被写入所连 CPU 中，而且 KincoBuilder 将会有对话框弹出来提示修改的结果。

14.1.3 忘记密码后的措施

如果用户忘记了 CPU 中的密码，那么所有受限的功能都将不能使用。

此时用户可以执行【PLC】→【清除...】菜单命令来清除 CPU 存储器中的所有内容，然后就可以继

续使用这个 PLC 了。执行完【清除...】命令后，用户程序、配置数据、密码等等将全部被清除，CPU 将恢复成出厂的初始状态。

附录 A 常用系统存储器 SM 区的定义

本附录详细描述了常用的系统存储区 SM 区中的相关定义。系统存储区是用来辅助 Kinco-K 系列 PLC 实现特定的指令功能，用户也可以通过使用系统存储器来读取 PLC 的某些状态。需要注意的是不是所有 CPU 都支持下边所有的 SM 去，比如某些 CPU 本体不自带模拟量，自然也就不支持 SM2.1 的功能

1、SMB0：系统状态字节

SM0.0-SM0.7 由 CPU 的运行软件进行赋值，不受用户程序控制，在用户程序中只能对这些位调用（只读）。详细的功能描述请参见下表。

位(只读)	描述
SM0.0	总是为“1”
SM0.1	首次扫描位。 在 CPU 首次扫描时为“1”，之后清“0”。通常用于用户程序初始化。
SM0.2	若 RAM 中的掉电保持数据丢失，则在首次扫描中 PLC 将该位置 1，之后清 0。
SM0.3	周期为 1s 的连续脉冲串，占空比为 50%。
SM0.4	周期为 2s 的连续脉冲串，占空比为 50%。
SM0.5	周期为 4s 的连续脉冲串，占空比为 50%。
SM0.6	周期为 60s 的连续脉冲串，占空比为 50%。
SM0.7	SM0.7 为 TRUE，电池电压低； SM0.7 为 FALSE，电池电压正常

2、错误寄存器

针对错误处理，KPLC 在 SM 区提供了一个控制字节和各种错误状态寄存器。当 PLC 发生错误时会自动设置相应错误寄存器的值。用户程序可以直接读取这些寄存器值并进行相应的处理。

- **SMB2：控制字节**

位（可读写）	功能描述
SM2.0	其值决定了当发生致命错误时 PLC 的动作。上电后初始值为 0。 若值为 0：发生致命错误时，PLC 进入独立的安全子系统运行。 若值为 1：发生致命错误时，PLC 直接复位。
SM2.1	其值决定了系统中 AI、AO 通道的工作状态。上电后初始值为 0。 若值为 0：本体的 AI、AO 通道正常工作。 若值为 1：本体的 AI、AO 通道进入校准状态。

- **SMB1（只读）：指令错误**

SM	功能描述
SM1.0	1 表示发生过如下错误：DIV、MOD 指令被 0 除。
SM1.1	1 表示发生过如下错误：LN、LOG、SQRT 指令参数为非法值（0 或者负数）。
SM1.2	1 表示发生过如下错误：I_TO_B、DI_TO_I 指令转换结果溢出。
SM1.3	1 表示发生过如下错误：I_TO_BCD 指令无效的输入 BCD 码。
SM1.4	1 表示发生过如下错误：A_TO_H 指令输入字符串中有不可识别的字符。
SM1.5	1 表示发生过如下错误：R_TO_A 指令转换结果溢出。
SM1.6	1 表示发生过如下错误：FOR 指令输入参数值无效。

- **SMB3，SMB5 和 SMB96-SMB110：扩展模块故障(仅用于支持扩展的 CPU)**

若 PLC 检测到扩展总线通信故障或者收到扩展模块发送的故障报文，则 PLC 会设置相应的扩展扩展总线故障标志位，同时将故障代码存放在相应的扩展总线故障寄存器中供查询。若没有检测到故障，则故障标志位和故障寄存器被设置为 0。

注意：若 PLC 在启动过程中检测到扩展总线或者扩展扩展模块故障，那么将进入 STOP 状态，同时点亮 ERR 灯，但不设置相应的寄存器，因为此时 CPU 不执行用户程序，用户就无法读取寄存器值。

SM	功能描述
SMB3，SMB5（只读）：扩展总线错误标志	
SMB3.0	若第 1 个扩展模块发生故障，则该位被置 1。
SMB3.1	若第 2 个扩展模块发生故障，则该位被置 1。
SMB3.2	若第 3 个扩展模块发生故障，则该位被置 1。
SMB3.3	若第 4 个扩展模块发生故障，则该位被置 1。
SMB3.4	若第 5 个扩展模块发生故障，则该位被置 1。
SMB3.5	若第 6 个扩展模块发生故障，则该位被置 1。
SMB3.6	若第 7 个扩展模块发生故障，则该位被置 1。
SMB3.7	若第 8 个扩展模块发生故障，则该位被置 1。
SMB5.0	若第 9 个扩展模块发生故障，则该位被置 1。
SMB5.1	若第 10 个扩展模块发生故障，则该位被置 1。
SMB5.2	若第 11 个扩展模块发生故障，则该位被置 1。
SMB5.3	若第 12 个扩展模块发生故障，则该位被置 1。
SMB5.7	若 CPU 检测到扩展总线通信故障，则该位被置 1。
SMB96 - SMB110（只读）：扩展总线错误代码	
SMB96	若第 1 个扩展模块发生故障，则存放其故障码。含义见表 1。
SMB97	若第 2 个扩展模块发生故障，则存放其故障码。含义见表 1。
SMB98	若第 3 个扩展模块发生故障，则存放其故障码。含义见表 1。
SMB99	若第 4 个扩展模块发生故障，则存放其故障码。含义见表 1。
SMB100	若第 5 个扩展模块发生故障，则存放其故障码。含义见表 1。
SMB101	若第 6 个扩展模块发生故障，则存放其故障码。含义见表 1。
SMB102	若第 7 个扩展模块发生故障，则存放其故障码。含义见表 1。
SMB103	若第 8 个扩展模块发生故障，则存放其故障码。含义见表 1。
SMB104	若第 9 个扩展模块发生故障，则存放其故障码。含义见表 1。
SMB105	若第 10 个扩展模块发生故障，则存放其故障码。含义见表 1。
SMB106	若第 11 个扩展模块发生故障，则存放其故障码。含义见表 1。
SMB107	若第 12 个扩展模块发生故障，则存放其故障码。含义见表 1。
SMB110	若 CPU 端的扩展总线发生故障，则存放其故障码。含义见表 2。

3、SMD12 和 SMD16：定时中断的周期

KPLC 提供了两个 0.1ms 时基的定时中断：定时中断 0，中断号为 3；定时中断 1，中断号为 4。

SMD12 用于指定定时中断 0 的周期值，单位 0.1ms。若将 SMD12 设置为 0，则定时中断 0 被禁止。SMD12 的缺省值为 0。

SMD16 用于指定定时中断 1 的周期值，单位 0.1ms。若将 SMD16 设置为 0，则定时中断 1 被禁止。SMD16 的缺省值为 0。

定时中断会周期性的产生，可以利用它来完成周期性的任务。定时中断不受 PLC 扫描周期的影响，可以用于精确的定时。

4、高速计数器的控制寄存器和状态寄存器

- **控制寄存器**

在 SM 区中为每个高速计数器均提供了如下控制寄存器用于存放其配置数据：一个控制字节（8 位），当前值和预置值各一个（均为 32 位有符号双整数）。当前值指定计数的起始值，若将当前值写入高速计数器，那么高速计数器就会立即从这个数值开始计数。下表详细描述了这些寄存器。

HSC0	HSC1	HSC2	HSC3	描述
SM37.0	SM47.0	SM57.0	SM127.0	复位信号的有效电平：0=高电平；1=低电平
SM37.1	SM47.1	SM57.1	SM127.1	启动信号的有效电平：0=高电平；1=低电平
SM37.2	SM47.2	SM57.2	SM127.2	正交计数器速率：0=1x 速率；1=4x 速率*
SM37.3	SM47.3	SM57.3	SM127.3	计数方向：0=减计数；1=增计数。
SM37.4	SM47.4	SM57.4	SM127.4	是否向 HSC 中写入计数方向：0=否；1=是。
SM37.5	SM47.5	SM57.5	SM127.5	是否向 HSC 中写入新预置值：0=否；1=是。
SM37.6	SM47.6	SM57.6	SM127.6	是否向 HSC 中写入新当前值：0=否；1=是。
SM37.7	SM47.7	SM57.7	SM127.7	是否允许该高速计数器：0=禁止；1=允许。
HSC0	**HSC1**	**HSC2**	**HSC3**	**描述**
SMD38	SMD48	SMD58	SMD128	当前值
SMD42	SMD52	SMD62	SMD132	预置值

另外除 KPLC 所有高速计数器均允许指定最大 32 个预置值（PV），其控制寄存器描述如下：

HSC0	HSC1	HSC2	HSC3	描述
SM141.0	SM151.0	SM161.0	SM171.0	是否使用多段预置值：0=否；1=是
SM141.1	SM151.1	SM161.1	SM171.1	预置值是相对值还是绝对值：0 =绝对；1=相对
SM141.2	SM151.2	SM161.2	SM171.2	预置值比较（“CV=PV”）中断是否循环产生： 0=否；1=是。 注意：只有相对值方式才允许设定为循环产生。
SM141.3	SM151.3	SM161.3	SM171.3	保留
SM141.4	SM151.4	SM161.4	SM171.4	是否更新段数及预置值：0=否；1=是
SM141.5	SM151.5	SM161.5	SM171.5	是否复位中断变量：0=是；1=否
SM141.6	SM151.6	SM161.6	SM171.6	保留
SM141.7	SM151.7	SM161.7	SM171.7	保留
HSC0	**HSC1**	**HSC2**	**HSC2**	**描述**
SMW142	SMW152	SMW162	SMW172	预置值表的起始位置（用相对于 VB0 的字节偏移来表示），必须为奇数。

- **状态寄存器**

每个高速计数器都在 SM 区中提供了状态寄存器用于指明高速计数器当前的状态信息。

HSC0	HSC1	HSC2	HSC3	描述
SM36.0	SM46.0	SM56.0	SM126.0	保留
SM36.1	SM46.1	SM56.1	SM126.1	保留
SM36.2	SM46.2	SM56.2	SM126.2	保留
SM36.3	SM46.3	SM56.3	SM126.3	多段 PV 值表设置是否有错误：0=否，1=是。
SM36.4	SM46.4	SM56.4	SM126.4	保留
SM36.5	SM46.5	SM56.5	SM126.5	当前计数方向：0=减；1=增。
SM36.6	SM46.6	SM56.6	SM126.6	当前计数值是否等于预置值：0=否；1=是。
SM36.7	SM46.7	SM56.7	SM126.7	当前计数值是否大于预置值：0=否；1=是。
HSC0	HSC1	HSC2	HSC3	描述（K5 系列不支持）
SMB140	SMB150	SMB160	SMB170	正在运行的 PV 值段序号（从 0 开始）。

关于高速计数器功能的使用详情请参考第八章 高速计数器功能的使用。

5、高速脉冲输出的控制寄存器和状态寄存器

5.1 使用 PLS 指令

关于高速脉冲输出 PTO/PWM 功能的使用详情请参考章节 9.3 PLS 指令的使用。

- **PTO/PWM 控制寄存器**

在 SM 区中为每个 PTO/PWM 发生器均提供了一些控制寄存器用于存放其配置数据。如下表。

Q0.0	Q0.1	Q0.4	Q0.5	描述	
SM67.0	SM77.0	SM97.0	SM107.0	PTO/PWM	是否更新周期值：0=否；1=是
SM67.1	SM77.1	SM97.1	SM107.1	PWM	是否更新脉宽值：0=否；1=是
SM67.2	SM77.2	SM97.2	SM107.2	PTO	是否更新脉冲个数：0=否；1=是
SM67.3	SM77.3	SM97.3	SM107.3	PTO/PWM	时基：0=1μs；1=1ms
SM67.4	SM77.4	SM97.4	SM107.4	PWM	更新方法：0=异步更新；1=同步更新
SM67.5	SM77.5	SM97.5	SM107.5	PTO	操作方式：0=单段操作；1=多段操作
SM67.6	SM77.6	SM97.6	SM107.6	功能选择：0= PTO；1=PWM	
SM67.7	SM77.7	SM97.7	SM107.7	PTO/PWM	允许或禁止此功能：0=禁止；1= 允许
Q0.0	**Q0.1**	**Q0.4**	**Q0.5**	**描述**	
SMW68	SMW78	SMW98	SMW108	PTO/PWM	周期值，范围 2~65535
SMW70	SMW80	SMW100	SMW110	PWM	脉宽值，范围 0~65535
SMD72	SMD82	SMD102	SMD112	PTO	脉冲个数，范围 1~4,294,967,295
SMW168	SMW178	SMW218	SMW248	包络表的起始位置（用相对于 VB0 的字节偏移来表示），仅用于 PTO 多段操作。	

- **PTO/PWM 状态寄存器**

在 SM 区中也为每个 PTO/PWM 发生器均提供了一个状态字节，用户可以通过访问状态字节来了解 PTO/PWM 发生器的当前状态信息。如下表。

Q0.0	Q0.1	Q0.4	Q0.5	描述
SM66.0	SM76.0	SM96.0	SM106.0	保留
SM66.1	SM76.1	SM96.1	SM106.1	保留

SM66.2	SM76.2	SM96.2	SM106.2	保留
SM66.3	SM76.3	SM96.3	SM106.3	PWM 是否空闲：0=否；1=是
SM66.4	SM76.4	SM96.4	SM106.4	PTO 周期值、脉冲个数设置是否有错误：0=否；1=是 **注：周期值、脉冲个数值必须大于 1。**
SM66.5	SM76.5	SM96.5	SM106.5	PTO 是否由于用户命令而终止：0=否；1=是
SM66.6	SM76.6	SM96.6	SM106.6	保留
SM66.7	SM76.7	SM96.7	SM106.7	PTO 是否空闲：0＝忙；1＝空闲

5.2 使用定位控制指令

关于定位指令的功能使用详情请参考 9.4 定位控制指令的使用。

针对定位控制指令，KPLC 在 SM 区中为每路高速输出均分配了一个控制字节，在应用中用户需要注意设置该控制字节。另外，还分配了一个当前值（DINT 型）寄存器，用于存放当前已经输出的脉冲个数（正转时增加，反转时减少）。下表详细描述了这些寄存器。

Q0.0	**Q0.1**	**Q0.4**	**Q0.5**	**描述**
SMD212	SMD242	SMD262	SMD226	只读。当前值，表示当前已经输出的脉冲个数。正转时增加，反转时减少。
SMD208	SMD238	SDM258	SMD222	读写。 新当前值。与相应标志位配合，用于修改当前值。
Q0.0	**Q0.1**	**Q0.4**	**Q0.5**	**描述**
SM201.7	SM231.7	SM251.7	SM221.7	读写。急停标志位。若该位为 1，则表示处于急停状态，不执行任何定位控制指令。 当 PSTOP（急停）指令执行时，该位将自动置 1。用户需要使用程序将该位清 0。
SM201.6	SM231.6	SM251.6	SM221.6	读写。用于决定是否复位当前值 1 - 将当前值清零。 0 - 当前值保持不变。
SM201.5	SM231.5	SM251.5	SM221.5	保留
SM201.4	SM231.4	SM251.4	SM221.4	读写。用于决定是否修改当前值 1 - 将当前值修改为上面“新当前值”寄存器中的值。

				0 - 当前值保持不变。
SM201.3	SM231.3	SM251.3	SM221.3	方向使能控制位。 1 禁止方向输出，方向通道作为普通 D0。 0 - 使能方向输出。
SM201.0 ~ SM201.2	SM231.0 ~ SM231.2	SM251.0 ~ SM251.2	SM221.0 ~ SM221.2	保留

6、自由通信指令 XMT/RCV 相关的控制寄存器和状态寄存器

关于自由通信 XMT/RCV 指令的功能使用详情请参考章节 10.2.1.2 自由通信指令。

KPLC 在 SM 区中为自由通信提供了多个状态寄存器和控制寄存器。在使用 XMT/RCV 指令编写通信程序时，用户必须对这些控制寄存器进行设置。另外，在通信过程中 CPU 会自动对通信状态进行检测，并将检测结果写入相关的状态寄存器，用户可以读取这些状态信息并在程序中进行相应的处理。

- 接收状态字节

位（只读）			值	含义
PORT 0	PORT 1	PORT 2		
SM86.0	SM186.0	SM286.0	1	接收到的字符存在奇偶校验错误，但不会停止接收。
SM86.1	SM186.1	SM286.1	1	终止接收：达到最大接收字节数。
SM86.2	SM186.2	SM286.2	1	终止接收：接收一个字符超时。
SM86.3	SM186.3	SM286.3	1	终止接收：系统接收超时。
SM86.4	SM186.4	SM286.4	-	保留。
SM86.5	SM186.5	SM286.5	1	终止接收：接收到了用户定义的结束字符。
SM86.6	SM186.6	SM286.6	1	终止接收：参数错误，无起始条件接收或无接收结束条件等。
SM86.7	SM186.7	SM286.7	1	终止接收：用户使用了禁止接收命令。

➢ 接收控制字节

位			值	描述
PORT 0	PORT 1	PORT 2		
SM87.0	SM187.0	SM287.0	-	保留。
SM87.1	SM187.1	SM287.1	0	当发送或者接收完成时禁止生成相应的中断。
			1	当发送或者接收完成时允许生成相应的中断。
SM87.2	SM187.2	SM287.2	0	忽略 SMW92/SMW192/ SMW292 中用户定义的接收字符超时值。
			1	使用 SMW92/SMW192/ SMW292 中用户定义的接收字符超时值。
SM87.3	SM187.3	SM287.3	-	保留。
SM87.4	SM187.4	SM287.4	0	忽略 SMW90/SMW190/SMW290 中用户定义的接收准备时间。
			1	使用 SMW90/SMW190/SMW290 中用户定义的接收准备时间。
SM87.5	SM187.5	SM287.5	0	忽略 SMB89/SMW189/SMW289 中用户定义的接收结束字符。
			1	使用 SMB89/SMW189/SMW289 中用户定义的接收结束字符。
SM87.6	SM187.6	SM287.6	0	忽略 SMB88/SMW188/SMW288 中用户定义的接收开始字符。
			1	使用 SMB88/SMW188/SMW288 中用户定义的接收开始字符。
SM87.7	SM187.7	SM287.7	0	禁止接收数据。此禁止条件优先于其它所有的接收控制字。
			1	允许接收数据。

➢ 其它控制寄存器

控制字（字节）			描　述
PORT0	PORT1	PORT2	
SMB88	SMB188	SMB288	用于存放用户定义的接收起始字符。 执行 RCV 指令后，CPU 收到了起始字符就开始进入有效接收状态，之前收到的数据都会被丢弃。CPU 将起始字符作为接收的第一个有效数据。 如果要使本设置值生效，需要将 SM87.6/SM187.6/SM287.6 置为 1。
SMB89	SMB189	SMB289	用于存放用户定义的接收结束字符。 CPU 将以该字符作为接收的最后一个字节。收到该字符后，无论其它的结束条件如何 CPU 都会立刻终止接收状态。 如果要使本设置值生效，需要将 SM87.5/SM187.5/SM287.5 置为 1。

SMW90	SMW190	SMW290	用于存放用户定义的接收准备时间（范围为 1~60000ms）。 执行 RCV 指令后，经过了该时间值，CPU 将自动进入有效接收状态而不管是否收到起始字符，此后接收到的数据将被认为是有效数据。 如果要使本设置值生效，需要将 SM87.4/SM187.4/SM287.4 置为 1。
SMW92	SMW192	SMW292	用于存放用户定义的接收字符超时值（范围为 1~60000ms）。 执行 RCV 指令并进入有效接收状态后，若在此超时时间内没有接收到任何字符，CPU 就会结束接收状态，无论其它的结束条件如何。 如果要使本设置值生效，需要将 SM87.2/SM187.2/SM287.2 置为 1。
SMW94	SMW194	SMW294	用于存放用户定义的每次接收字符数（1~255）。 CPU 只要接收到了此数量个有效字符，无论其它的结束条件如何，都会马上终止接收状态。本设置值总是有效。 若该值设置为 0，则 RCV 指令将直接退出。

在串口自由通信中，另外还有一个默认的系统接收超时，时间为 90 秒，此超时值的作用如下：在执行 RCV 指令后，若在此超时时间内串口上没有收到任何数据，则 CPU 将立刻终止接收并退出 RCV 指令；另外，CPU 在进入有效接收状态后（即接收到 SMB88 中定义的起始字符或者经过了 SMW90 中定义的接收准备时间后），将优先使用用户在 SMW92 中定义的接收字符超时值，若用户没有定义，则用该系统接收超时值来决定是否终止接收。

7、动态修改 RS485 通信口参数的相关寄存器

关于动态修改 RS485 通信口参数的功能使用详情请参考 10.2.3 动态修改 RS485 通信口参数。

除 K209M、KS101M、KW203、MK、K6 外，其它系列 PLC 支持使用 SMB20--SMB25 来动态修改通信参数。下面详细描述了各个寄存器的含义。

- **参数值：SMB23、SMB24 和 SMB25**

SMB	描 述
SMB23	PLC 站号值。数值有效范围：1-31。 若执行写操作，则 SMB23 是将要写入的新 PLC 站号值；若执行读操作，则 SMB23 是读取到的当前使用的 PLC 站号值；若执行清除操作，则 SMB23 被忽略。

SMB24	波特率值。数值有效范围 0-5：0 表示 2400，1 表示 4800，2 表示 9600，3 表示 19200，4 表示 38400，5 表示 1200。 若执行写操作，则 SMB24 是将要写入的新波特率值；若执行读操作，则 SMB24 是读取到的当前使用的波特率值；若执行清除操作，则 SMB24 被忽略。
SMB25	奇偶校验值。数值有效范围 0-2，0 表示无检验，1 表示奇检验，2 表示偶检验。 若执行写操作，则 SMB25 是将要写入的新奇偶校验值；若执行读操作，则 SMB25 是读取到的当前使用的奇偶校验值；若执行清除操作，则 SMB25 被忽略。

- **控制字节：SMB20 和 SMB21**

位	描述
SMB20：指定将要操作的端口号和操作方式。	
SM20.7	值为 1 表示启动写操作。PLC 写入新参数完成后会自动将该位清 0。
SM20.6	值为 1 表示启动读操作。PLC 读取参数完成后会自动将该位清 0。
SM20.5	值为 1 表示启动清除操作。PLC 清除参数完成后会自动将该位清 0。
SM20.4	保留，必须赋值为 0。
SM20.3 SM20.0	这 4 位的组合值表示将要操作的端口号。 1 表示 PORT1，2 表示 PORT2，若设置为其它无效值则 PLC 将报错并退出操作。
SMB21：指定将要操作的通信参数。	
SM21.7 -- SM21.3	备用。必须赋值为 0。
SM21.2	值为 1 表示修改或者清除指定通信口的奇偶校验值。
SM21.1	值为 1 表示修改或者清除指定通信口的波特率值。
SM21.0	值为 1 表示修改或者清除指定通信口的 PLC 站号。

在同一时刻，SM20.5、SM20.6、SM20.7 只允许有一个位的值为 1，否则 PLC 将报错并退出操作。

当执行读操作时，SMB21 的值被忽略，PLC 将一次性读取所有的通信参数值。

当执行清除操作时，PLC 允许分别或者同时清除站号、波特率、奇偶校验值。某个参数被清除后，

PLC 将会自动采用硬件配置信息中相应的通信参数值。

- **状态字节：SMB22**

SMB22 存放了本次动态调整通信参数的操作结果。

位（只读）	描述
SM22.7	值为 1 表示本次操作完成。 PLC 完成用户指定的操作后，无论成功还是失败，都会自动将 SM22.7 置 1。只有当 SM22.7 的值为 1 时，SMB22 中其它位的值才有实际意义。
SM22.6	若 SM22.7 值为 1，那么若 SM22.6 值为 1，则表示本次操作成功；若 SM22.6 值为 0，则表示本次操作因发生错误而失败。
SM20.5 ～ SM20.0	若本次操作失败，那么这 6 位的组合值就表示发生的错误代码，含义如下表。

错误代码	错误描述
1	错误的操作命令。比如，SM20.7 和 SM20.6 被同时置为 1。
2	错误的端口号
3	SMB21（将要操作的通信参数）值错误。
4	SMB23（新 PLC 站号）值错误。
5	SMB24（新波特率）值错误。
6	SMB25（新奇偶校验）值错误。
10	从永久存储器中读取存放的 PORT1 动态 PLC 站号失败。
11	尚未设置过 PORT1 的动态 PLC 站号。
12	从永久存储器中读取存放的 PORT1 动态波特率值失败。
13	尚未设置过 PORT1 的动态波特率。
14	从永久存储器中读取存放的 PORT1 动态奇偶校验值失败。
15	尚未设置过 PORT1 的动态奇偶校验值。
20	从永久存储器中读取存放的 PORT2 动态 PLC 站号失败。
21	尚未设置过 PORT2 的动态 PLC 站号。
22	从永久存储器中读取存放的 PORT2 动态波特率值失败。
23	尚未设置过 PORT2 的动态波特率。

24	从永久存储器中读取存放的 PORT2 动态奇偶校验值失败。
25	尚未设置过 PORT2 的动态奇偶校验值。
61	动态通信参数值写入永久存储器失败。

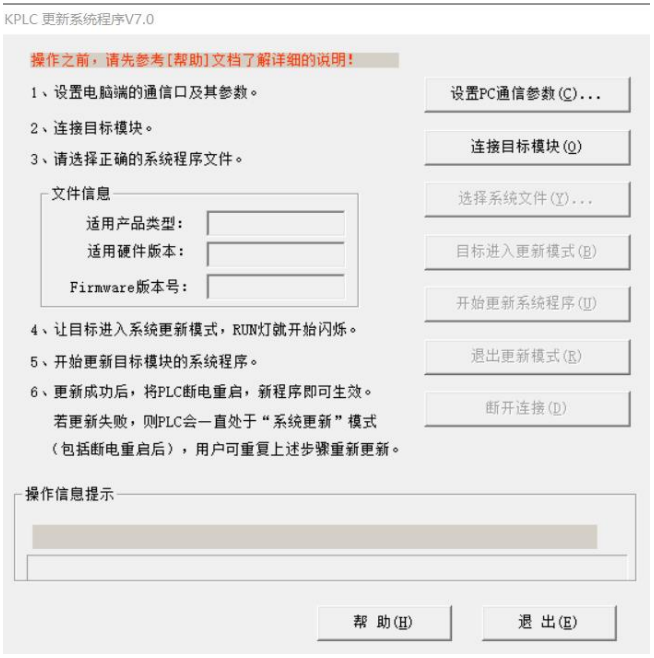
8、其它常用功能变量

SM	描述	
SMB6	只读。存放最近一次的 PLC 扫描时间，单位：ms。	
SMW10	只读。存放后备电池的电压值，单位：0.01V。（仅适用于 K5） 若后备电池的电源持续低于 2.6V 时，PLC 会产生“后备电池电量低”报警。	
SMW122	CAN2 口总线负荷率最小值。	负荷率 = SMW 的值÷1000
SMW124	CAN2 口总线负荷率最大值。	
SMB274-SMB285	这 12 个字节的组合值表示本 CPU 模块的 ID 值。 出厂时每个 CPU 模块均被赋予一个唯一的 ID 值，各个模块 ID 值不会相同。	

附录 B 更新系统程序

KPLC 提供 bootloader，允许用户自行更新系统程序。步科公司在持续对 KPLC 进行升级、改进，并发布最新的 PLC 系统程序。若用户需要最新版本的 PLC 系统程序，可以联系步科公司获取。

在 KincoBulider 软件中，用户执行【工具】->【更新系统程序】菜单命令即可进入更新系统程序的小工具软件。在软件界面中简要指明了更新 PLC 系统程序的步骤，用户也可以单击【帮助】按钮查看更详细的说明。



PLC 具有正常工作和系统更新两种模式。用户首先要让 PLC 进入系统更新模式，然后才可以更新其系统程序。

因各个系列 PLC 提高的通信口种类和数量均有不同，所以不同型号产品可以使用的通信口有所不同。另外，因历史原因，部分型号需要进入 STOP 状态才能进行更新。这些信息在更新软件的帮助中均有详细说明，此处也列表汇总如下：

型号	PORT0（RS232）	PORT1（RS485）	PORT2（RS485）	USB	以太网	是否需要先进入 STOP 状态
K5 系列	支持	/	/	/	/	是
K2 系列	/	/	/	支持	/	是
KS 系列	支持	支持	/	/	/	是
K209M KS101M	支持	支持	/	支持	/	否
KW 系列	支持	支持	/	支持	/	否
MK 系列	/	支持	/	支持	/	否
K6 系列	/	支持	/	/	支持	否

1、更新 PLC 系统程序的步骤及注意事项

用户也可以在更新系统程序软件的【帮助】中查阅相关的说明。

1）使用通信电缆连接 PLC 与电脑，然后给 PLC 上电。

注：除了 MK 系列，其它具有 USB 接口的产品支持 USB 数据线直接供电，在更新时无需外接电源。

2）单击【设置 PC 通信参数】按钮，进入配置窗口，用户可以选择电脑通信口并配置通信参数。

注：若使用 USB 口更新，因为是虚拟串口，所以无需修改通信参数；若使用串口更新，则注意采用波特率 115200，无校验，8 位数据位，1 位停止位，请先将用户工程【硬件配置】中的相应 PORT 的通信参数更改为上述值，然后将新工程下载到 PLC 中，等 PLC 成功运行后新参数就会生效。

3）（可选）首先将 PLC 置于“STOP”状态。 **注：K6、MK、KM、KW 系列不需要先进入 STOP 状态**。

4） 单击【连接目标模块】按钮。若用户使用串口或者 USB 口更新，则软件会打开相应的通信口；若使用以太网口更新，则软件会与 PLC 建立一个 TCP 连接。

5） 单击【选择系统程序文件...】按钮，在弹出的对话框里选择合适的文件并打开。

若是合法文件，那么在【文件信息】中会提示适用的 PLC 型号和硬件版本，同时会提示该文件中系统程序的软件版本号。

6）单击【PLC 进入更新模式】按钮，在延时约 2 秒钟后，PLC 就会进入系统更新模式。

在系统更新模式下，PLC 的“RUN”指示灯会快速闪烁，其它指示灯熄灭。

若使用 USB 口更新，进入更新模式后，原有的 USB 虚拟串口会被破坏，软件将会自动重新建立虚拟通信口。若自动建立失败，用户可先拔掉 USB 电缆，重新插上后再单击【PLC 进入更新模式】按钮。

若使用以太网口更新，进入更新模式后，原有的 TCP 连接会被破坏，软件将会自动与 PLC 重新建立 TCP 连接。若自动连接失败，用户可再次单击【PLC 进入更新模式】按钮。

7） 单击【开始更新系统程序】按钮，即可启动 PLC 的系统更新过程。

更新过程包括擦除 PLC 的系统程序数据、写入新程序数据、对写入数据进行校验等等。在窗口下部的【操作信息提示】中，软件会提示正在进行的操作和进度。

若更新过程中发生错误，软件将停止更新并提示失败原因。**在更新失败后，PLC 将会永久停留在系统更新模式（此时 RUN 指示灯闪烁），用户可以尝试再次单击【开始更新系统程序】按钮来重新开始更新，或者将 PLC 断电至少 10 秒钟后重启（注意保持之前的 RUN/STOP 状态），然后重新执行一遍前述的过程。**

8）提示更新成功后，将 PLC 断电重启，PLC 即可运行新的系统程序。

其它注意事项：

1) 系统程序文件、待更新的 PLC 的型号以及硬件版本必须匹配！
2) PLC 的系统程序文件扩展名为“.kfw”，具有自己特定的编码格式，注意不要该文件进行任何修改！如果向 PLC 中写入了非法的系统程序，PLC 将可能运行异常且用户无法修复！
3) 若 PLC 的系统更新失败，则无论用户是否断电重启，PLC 都会保持在系统更新模式（此时 RUN 指示灯闪烁），用户需要对 PLC 的系统进行正确的更新！只有更新成功后，PLC 才会退出系统更新模式。

附录 C PLC 离线仿真器的使用

一、简介

在 KincoBuilder 软件中提供了 KPLC 的离线仿真器，可以在 KincoBuilder 中模拟真实的 PLC 运行、调试用户程序。KPLC 的离线仿真器功能强大，提供了断点、单步执行、单网络执行、暂停/继续运行、串行通信仿真器等高级调试功能。

在正式使用 PLC 之前需要反复的调试和修改工程，才能达到理想效果，所以调试、测试是十分重要的工作，真正的调试需要有硬件基础，在不具备硬件条件的时，用户可以使用 PLC 编程软件 KincoBuilder 自带的离线仿真功能，进行初步的调试，模拟 PLC 运行效果。此说明介绍了一些具体的使用方法。

二、运行离线仿真器

2.1 启动步骤

1、打开 KincoBuilder 软件，并打开需要调试的工程

2、点击菜单栏图标  编译工程，然后在屏幕下方的【信息输出窗口】提示编译成功。**注意：一部分指令暂不支持离线仿真，所以离线仿真的程序中不能包含此类指令。**

3、编译成功后，单击菜单栏中的图标  ，启动离线仿真器。或者直接启动离线仿真器，在启动时仿真器会自动进行编译。

2.2 不支持仿真的指令

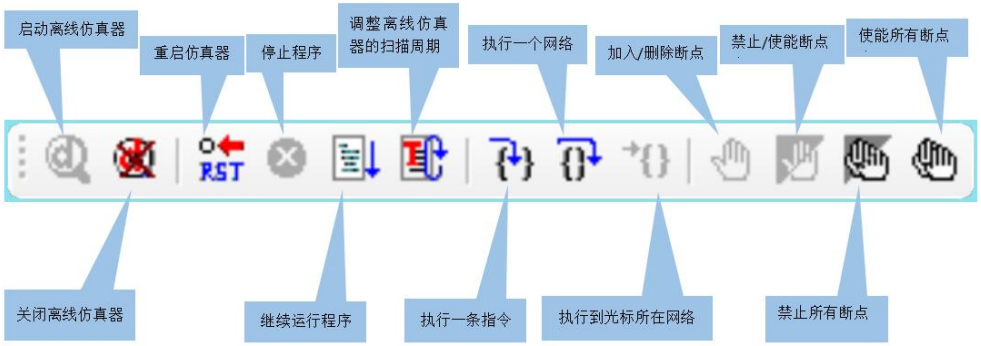
离线仿真暂时不支持一部分指令。

若程序中含有不支持仿真的指令，则会弹出如下图的提示框，用户单击【确定】按钮后，仿真器将忽略不支持的指令并继续运行。

暂时不支持离线仿真的指令有：

- 各种高速脉冲输出功能及指令，包括 PLS、定位控制指令等。
- 各种高速脉冲输入功能及指令，包括 HDEF、HSC、SPD、高速计数器向导等。
- 各种 CAN 通信功能及指令，包括 CAN 自由通信、CANOpen 主站、Kinco 运动控制等。
- 各种以太网功能及相关指令，包括以太网自由通信、Modbus TCP Client 等
- Kinco 互联协议功能及相关指令。
- LoRa 通信功能及相关指令。

2.3 认识仿真运行工具



三、离线仿真启动选项

软件中菜单栏【调试】--【启动离线仿真器】或者单击快捷图标  ，默认会弹出界面【离线仿真启动选项】

启动仿真器时可以设置启动选项，三个选项，选中的选项的某项就会执行该功能

1、仿真器启动时状态：连续运行或者暂停在第 1 个网络

（1）连续运行，进入仿真器之后程序就会依次循环进行直到执行别得操作或者退出仿真

（2）暂停在第 1 个网络，进入仿真器之后程序就会暂停在第 1 个网络的第一条指令

如果设置暂停在第一网络，例如扫描周期设置 5 秒时可看到，启动仿真器 5 秒后，开始第一次扫描并暂停在第一网络，（扫描周期设置在 1 秒或者更大可以看到此效果）。

2、存取区域：启动时选择是否将 V 区中的数据保持区和永久数据备份区清零

所清零的是用户在“PLC 硬件配置”中“数据保持”部分有关 V 区的设置

3、其他选项：

使用最近一次的断点设置：指的是使用最近一次的断点生效的情况，不是删除断点；不使用最近一次断电设置，启动之后，所有的断点都处于无效状态，用户可根据需要修改。

使用最近一次的 modbus 主站预定义通信报文表：使用最近一次的 modbus 主站预定义通信报文及其生效情况，不使用报文不会删除，只是处于无效状态。

使用最近一次的自由协议预定义通信报文表：使用最近一次的自由协议预定义通信报文及其生效情况，

不使用报文不会删除，只是处于无效状态。

备注：下一次启动时不再弹出本对话框，设置之后会跟随用户工程存盘，想最次打开对话框，在菜单栏--工具--软件设置--离线仿真设置，将启动时弹出启动选项对话框前的方向重新勾选，并确定。再启动仿真器时就会弹出。**注意：【软件设置】-【离线仿真设置】里的两个选项都是跟随用户工程存盘。**

四、离线仿真主要包括四部分功能

4.1 基本仿真

4.1.1 运行状态通过节点颜色变化区分

a、大红色：表示正在处理中还没有结果的节点

b、蓝色：表示执行过的节点（执行时 CR 为真是一个色）

c、橙色：表示执行过的节点（执行时 CR 为假是一个色，其实就是没有真执行）

d、绿色：表示网络在单步运行时执行中和执行过的状态，左右粗竖线的颜色变绿

4.1.2 基本仿真操作指令介绍

1、【启动离线仿真器】：单击图标 

2、【关闭离线仿真器】：单击图标 

3、【重启仿真器】：单击图标 

4、【停止程序】：单击图标 

5、【继续运行程序】当程序状态处在暂停状态时，单击  图标可以继续运行程序。

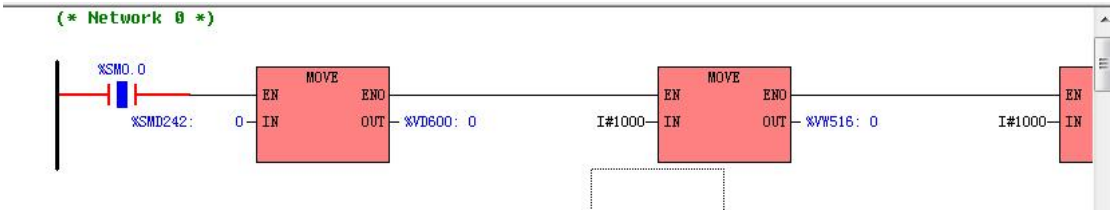
6、【调整离线仿真器的扫描周期】：单击图标 

调整离线仿真器的扫描周期，若不调整自动选择默认值。扫描周期只能在启动离线仿真器时修改，

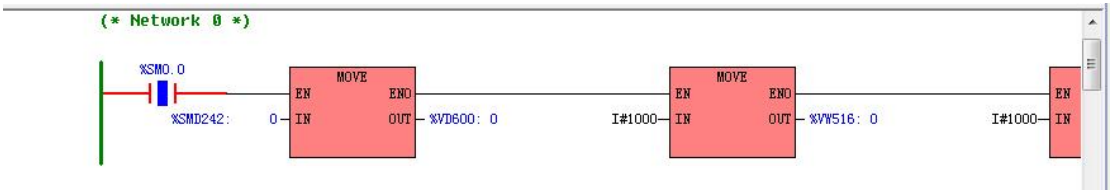
不能自主输入时间，可以拖动方块选择设置时间，时间设置可选：最快（是不是由硬件决定，一般是多少？）、默认（20ms？）、50ms、100ms、200ms、500ms、1000ms、2000ms、5000ms、10000ms。此扫描周期只针对仿真器，不影响真实 PLC 的运行。

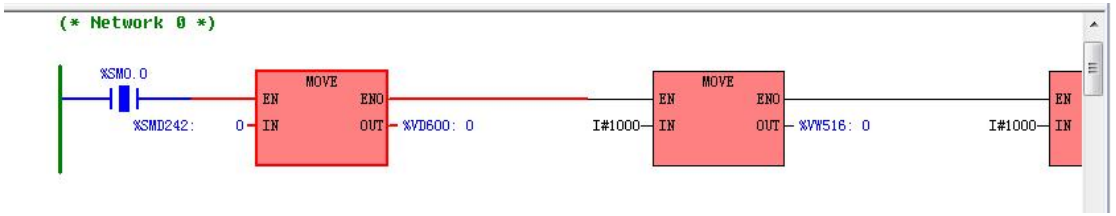
7、【执行一条指令】：单击图标 

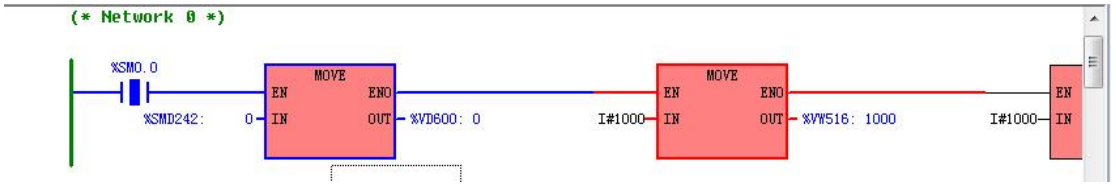
例如：先暂停在第一网络，然后单击执行一条指令，效果运行如下，若前面开关量导通，后面的 MOVE 指令执行之后，功能块的颜色变为蓝色



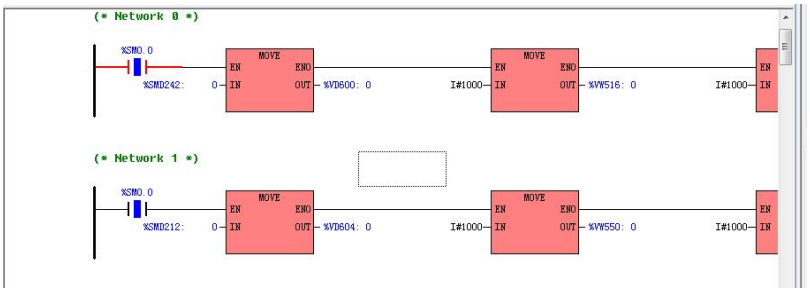
图示为：暂停在第一网络







8、【执行一个网络】：单击 图标，执行当前所在位置未执行完的一个网络。



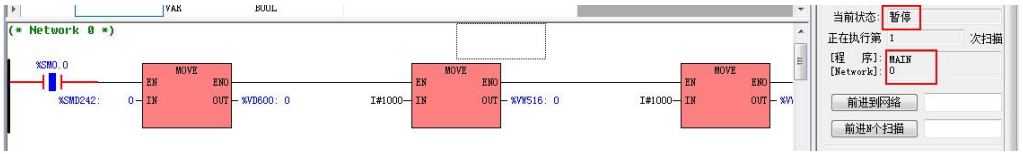
9、【执行到光标所在网络】：单击 图标，执行到光标所在网络，只能向下执行。当光标所在位置没有处在任何一个网络时，此图标为灰色，不能进行任何操作；当光标放在当前执行位置在同一个网络，点击执行到光标所在网络，不会有动作；若光标放在当前执行位置的下一个网络，程序就会执行到该网络；若将光标放置在当前执行位置的上一个网络，点击执行的到光标所在的网络，不会动作。

10、仿真器界面程序调试部分包含：当前状态、正在执行的 N 次扫描、程序 Network、前进到网络、前进 N 个扫描。

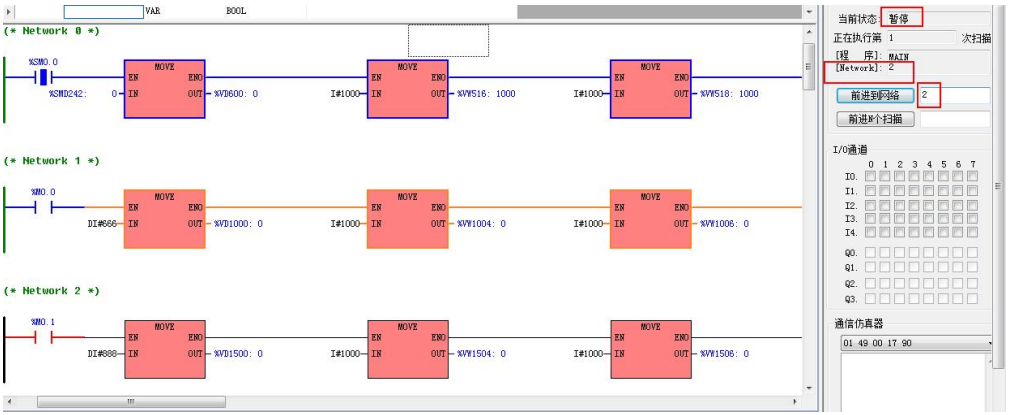
当前状态、正在执行的 N 次扫描、程序 Network，都作为状态指示方便用户查看。

前进到网络（暂停状态下有效）：在暂停的状态下，前进的网络数，可以自己在后面的空白处输入，然后点前进到网络，程序就会执行到指定的网络并暂停。

例如：现在网络暂停在 Network 0，现在我们在“前进到网络”后的空白处输入 2，也就是要前进到 Network 2，运行效果如下图：



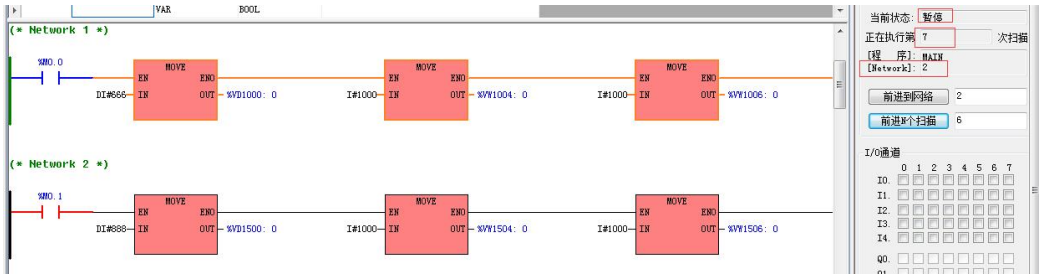
上图示为第一次扫描，暂停在 Network 0



上图示为设置前进到网络之后，暂停在 Network 2

前进 N 个扫描（暂停状态下有效）：在暂停的状态下，前进 N 个扫描，然后“前进 N 个扫描”后的空白处输入，然后点前进 N 个扫描，程序显示正在第 N+1 次扫描并暂停。

例如：上面示例是暂停在第一次扫描的 Network 2，如果再设置一下前进 N 个扫描，在“前进 N 个扫描”后的空白处输入 6，当执行该操作之后就会暂停在第 7 次扫描的 Network 2。效果如下图：



4.2 断点调试

4.2.1 网络中断点及其状态变化指示

网络左竖线左上角深红色圆圈：有表示该网络中有断点，没有表示该网络中没有断点。

圆圈有两个状台：（1）、若圆圈是空心  ，表示该网络中断点被禁止

（2）、若圆圈是实心  ，表示该网络中断点使能

4.2.2 断点操作

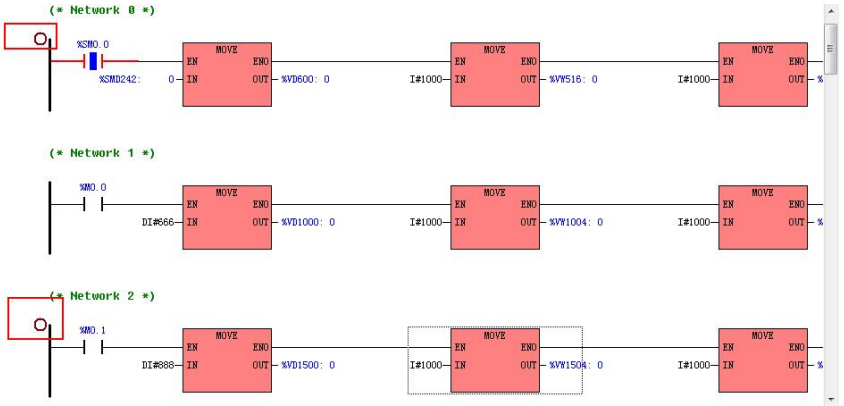
1、【加入/删除断点】：单击  图标，加入/删除断点，光标不在网络中的任意节点时  图标为灰色，不能进行操作。将光标放置任一网络中，即可进行加入/删除断点操作。

将光标放在网络中的任意位置，单击加入/删除断点，加入和删除是交替执行，第一次点击为加入使能断点，第二次点击是删除断点，判断加入和删除断点的标志在操作网络的左竖线左上角，会出现一个实心的圆圈，或者是没有圆圈。有实心的圆圈表示加入使能断点，没有圆圈表示此网络没有断点。

2、【禁止/使能断点】：单击  图标，禁止/使能断点，交替起作用，且只能作用于当前光标所在的且有断点的网络使能或者禁止断点，可根据网络左竖线的左上角圆圈状态判断是使能还是禁止。

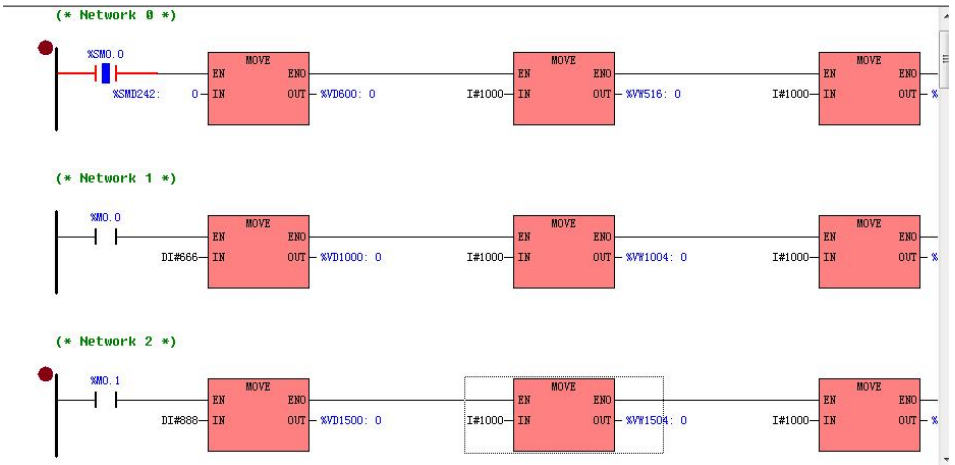
3、【禁止所有断点】：单击  图标，禁止所有断点

禁止所有断点是禁止程序中所有的断点，禁止之后可以看到，有断点的网左竖线的左上角有一个空心圆，如下图中红框中标注出的



4、【使能所有断点】：单击  图标，使能所有断点

使能所有断点是使能程序中所有的断点，点击图标之后可以看到，有断点的网络前端的左上角有一个实心圆，如下图：



5、断点及暂停条件

此操作在菜单栏中【调试】--【断电及暂停条件】打开，点开此界面可以统一查看和编辑暂停条件和断点列表。暂停条件中内存地址需要自输入，与内存地址相对应设置数据类型和数值，运算符可以选择，有：大于、大于等于、小于、小于等于、等于、不等于。是否生效根据可以单个设置，也可以使用右侧的全部使能或者全部禁用。

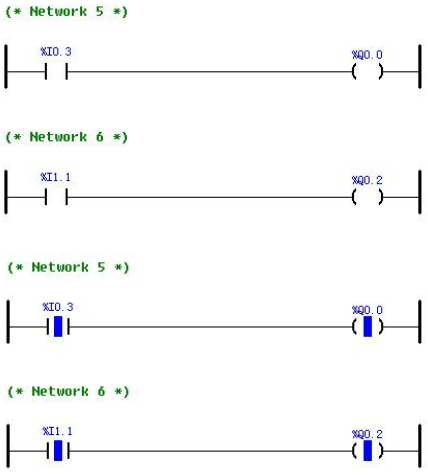
4.3 IO 操作

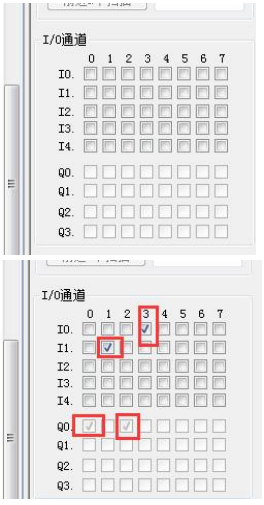
此处模拟看效果需要连续运行才有，暂停时单步运行看不到效果。

模拟 I/O 通道的输入信号和输出信号，手动勾选输入 I，模拟输入信号。输出 Q 随程序控制变化

示例：当 I0.3、I1.1 导通时，Q0.0、Q0.2 也随之导通；当 I0.3、I1.1 关断时，Q0.0、Q0.2 关闭

效果图：





4.4 通信仿真

报文格式可以参考软件菜单栏中【帮助】--【帮助主题】--附录 A 使用 Modbus RTU 协议与第三方设备通信--2、 Modbus RTU 报文基本格式，此说明中不再讲解报文格式。

1、浏览或选择最近发送过的报文

2、填写报文处

3、点击自动生成 2 处填写报文的 CRC 校验码

4、将 2 内填写的报文发送至 PORT0 口

5、将 2 内填写的报文发送至 PORT1 口

6、将 2 内填写的报文发送至 PORT2 口

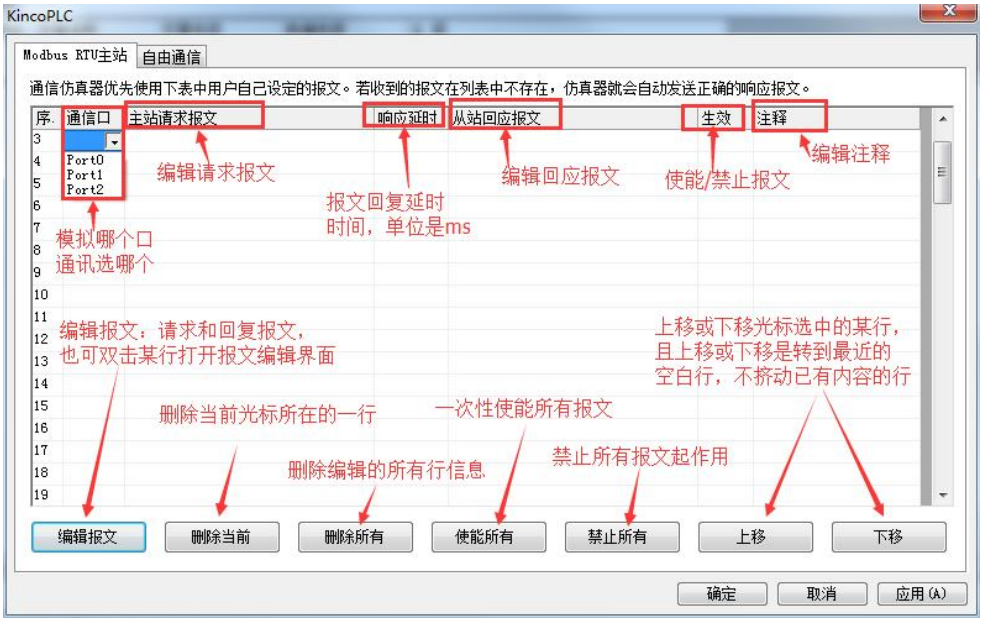
7、打开通信报文表

通信报文表里的报文可以认为是与 PLC 通信的设备发送给 PLC 的报文。

通信报文表可以设置 Modbus RTU 主站和自由通信（此处编辑的报文会随工程存盘）

通信仿真器优先使用下表中用户自己设定的报文。若收到的报文在列表中不存在，仿真器就会自动发送正确的响应报文。

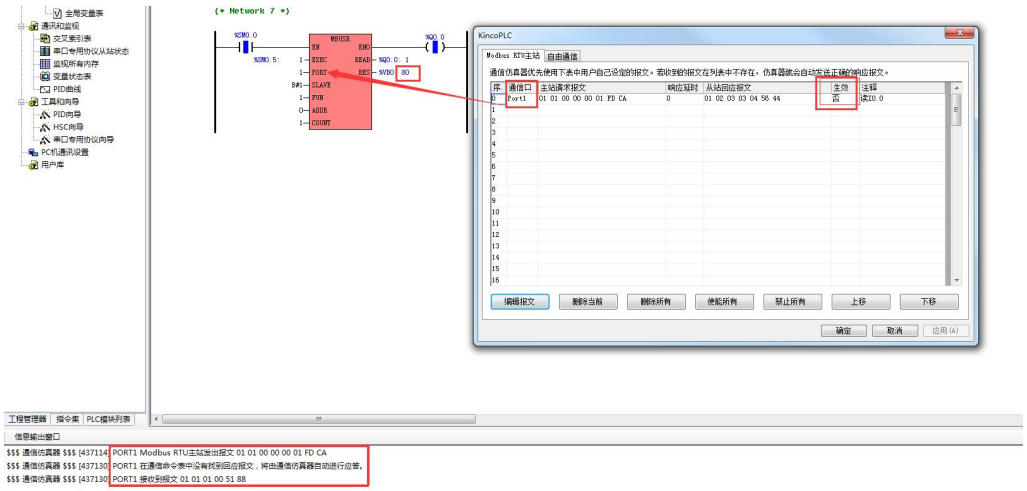
打开通信报文表优先进入自由通信表



（1） Modbus RTU 主站

例如：PLC 硬件设置中将 PORT1 作为 Modbus 主站，程序中读取 1 号从站的 I0.0 的状态

a、报文表编辑的报文设置无效，可以看到【信息输出窗口】



$$$ 通信仿真器 $$$ [437114] PORT1 Modbus RTU 主站发出报文 01 01 00 00 00 01 FD CA

$$$ 通信仿真器 $$$ [437130] PORT1 在通信命令表中没有找到回应报文，将由通信仿真器自动进行应答。

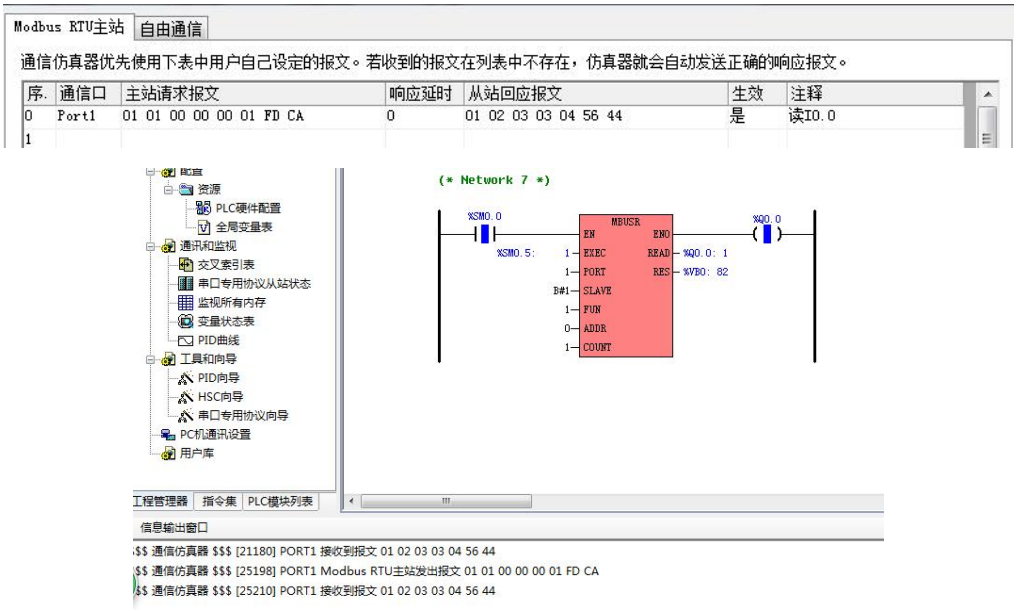
$$$ 通信仿真器 $$$ [437130] PORT1 接收到报文 01 01 01 00 51 88

同时，MBUSR 指令的返回值 VB0 位 80，指示通信成功。

备注：【信息输出窗口】$$$ 通信仿真器 $$$ [437114] PORT1 Modbus RTU 主站发出报文 01 01 00 00 00 01 FD CA，其中的[437114]表示距离本次开始离线仿真的时间，毫秒计数。

[437114]--[437130] 离线仿真处理一个通信报文，用时 16 毫秒

b、报文表编辑的报文设置有效



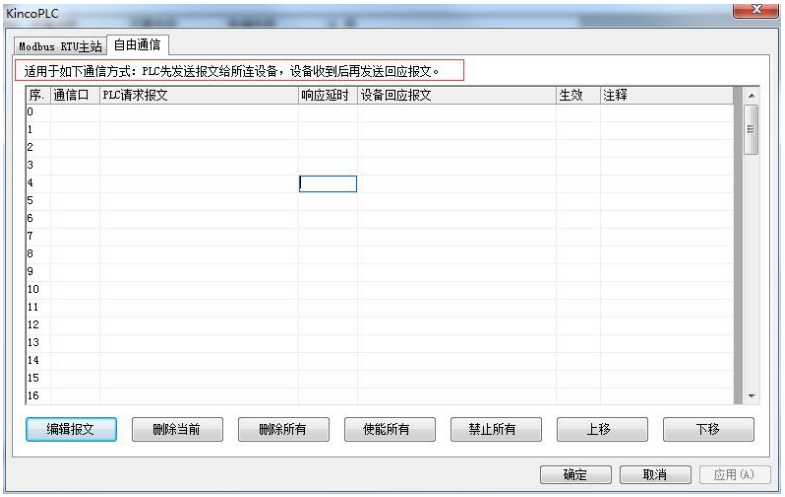
可以看到【信息输出窗口】

$$$ 通信仿真器 $$$ [25198] PORT1 Modbus RTU 主站发出报文 01 01 00 00 00 01 FD CA

$$$ 通信仿真器 $$$ [25210] PORT1 接收到报文 01 02 03 03 04 56 44

同时，MBUSR 指令的返回值 VB0 位 82，指示通信超时。

（2）自由通信（只支持 XMT　RCV 指令，目前暂不支持 COM_XMT　COM_RCV）



使用方法同 Modbus RTU 一样，只不过程序中的 Modbus 指令改为 XMT 指令，在此不再累述。

附录 D 一体机中屏部分介绍

Kinco 一体式控制器分为两个系列：HP 系列和 MK 系列，二者 HMI 部分的组态软件不是同一个软件，下面将分别对两个系列的 HMI 部分的使用做简要介绍。

一、HP 系列中 HMI 部分使用说明

1.1 HMI 的使用

HMI 编程软件为：Kinco HPBuilder。下载请登录官网--资料下载目录下【软件】--【PLC 编程软件】中选择一体机 HMI 编程软件进行下载，官网网址：www.kinco.cn。

1.1.1 工程制作

下面介绍使用 Kinco HPBuilder 制作工程的步骤（以 HP043 为例）。

一、创建工程

启动 Kinco HPBuilder

1、新建工程：

（1）菜单栏【文件】--【新建工程】；

（2）输入工程名称；

（3）选择工程文件夹保存路径；

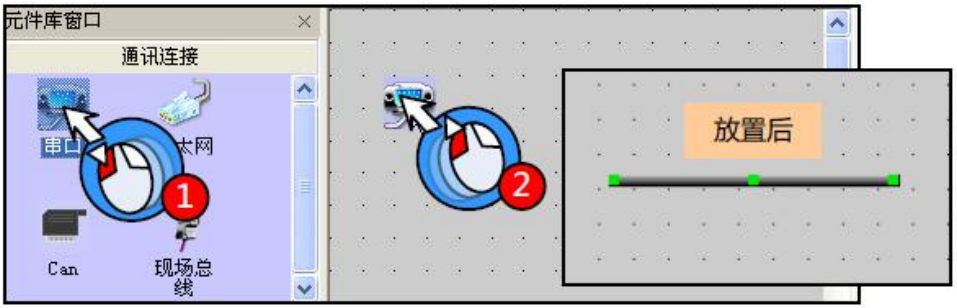
（4）单击【建立】完成工程创建；

2、设备选择、连接和参数设置

（1）设备选择

①设备选择--选择通信方式

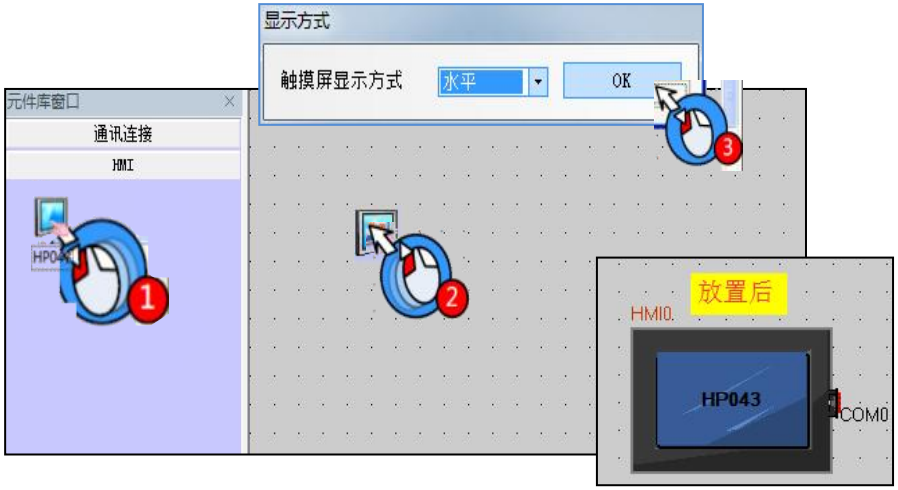
从【元件库窗口】--【通信连接】中选择“串口”往拓扑结构窗口拖曳并放置



②设备选择--选择 HMI 型号

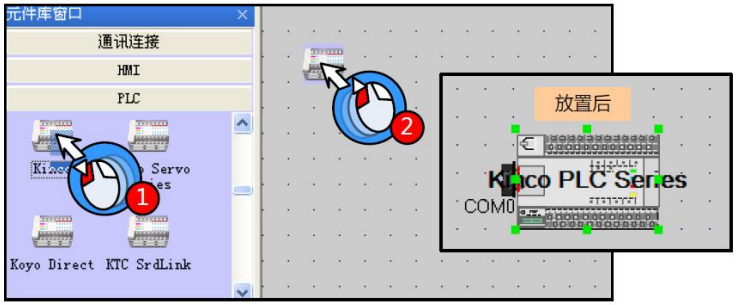
从【元件库串口】--【HMI】中选择“HP043”后往拓扑结构窗口拖曳并放置

拖放 HMI 是系统会弹出下图所示的【显示方式】对话框，要求用户选择触摸屏的显示方式，系统提供“水平”和“垂直”两个选项（本例选择“水平”），单击【OK】



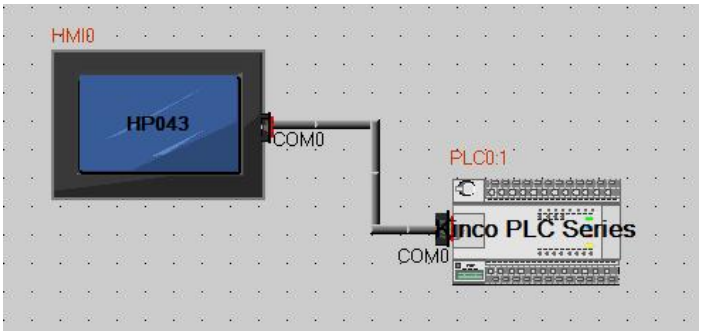
③设备选择--选择 PLC 型号（通信协议）

从【元件库窗口】--【PLC】中选择“Kinco PLC Series”后往拓扑结构窗口拖曳并放置



（2）设备连接

选中HMI并拖曳使其“COM”口靠近通通信连接的左端，直到拖曳HMI时通信连接线的一端也跟着“COM口”移动，此时 HMI 与串口电缆连接成功；同样的方法连接 PLC 和串口电缆



（3）参数设置--HMI 设置

①双击 HMI，系统弹出【HMI 属性】属性框

②切换到【任务栏】属性页

③在【任务栏】属性页中将“显示任务栏”选项前的勾选取消

④切换单【串口 0 设置】属性页根据实际 PLC 的通信参数来配置串口 0 的参数，其他选项均采用默认设置

（3）HMI 串口 0 设置

实际 PLC 的通信参数

二、编辑组态画面的编辑，可以参考 Kinco HMIware 使用手册。

三、HMI 对外提供 USB 主从口，具体使用参考 Kinco DTools 组态编辑软件使用手册。

1.1.2 HMI 使用手册下载链接

下载链接：

https://www.kinco.cn/Download/usermanual/HMI/Kinco%20DTools%E7%BB%84%E6%80%81%E7%BC%96%E8%BE%91%E8%BD%AF%E4%BB%B6%E4%BD%BF%E7%94%A8%E6%89%8B%E5%86%8C20200330.pdf

二、MK 系列中 HMI 部分使用说明

2.1 HMI 的使用

HMI 编程软件为：Kinco Dtools。下载请登录官网--资料下载目录下【软件】--【HMI 编程软件】中选择 Kinco Dtools 进行下载，官网网址：www.kinco.cn。

2.1.1 工程制作

下面介绍使用 Kinco Dtools 制作工程的步骤（以 MK070E 为例）。

一、创建工程

启动 Kinco Dtools

1、新建工程：

（1）菜单栏【文件】--【新建工程】；

（2）输入工程名称；

（3）选择工程文件夹保存路径；

（4）选择 HMI 型号 MK070E

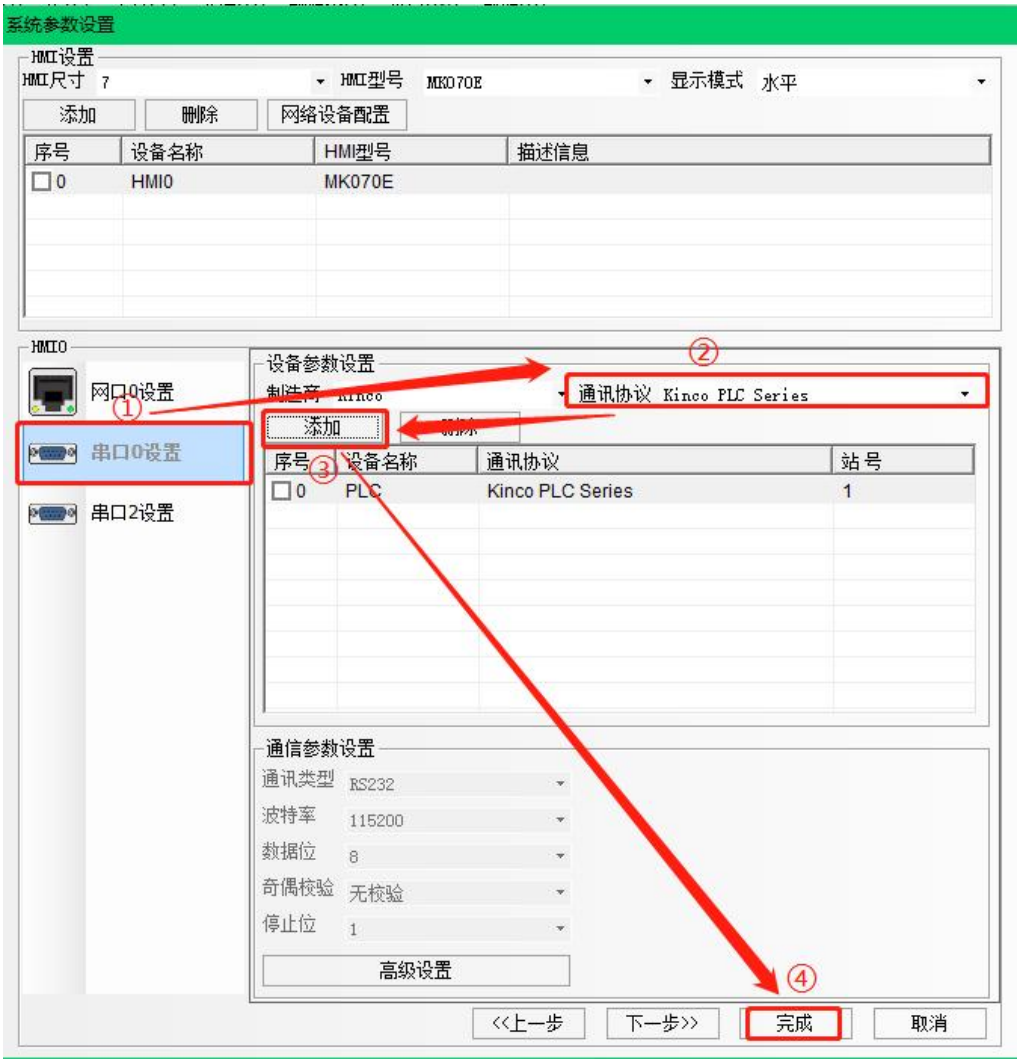
（5）点击下一步进入如下系统参数设置画面：

① 点击串口 0 设置（注意务必是 COM0。COM2 不可用）

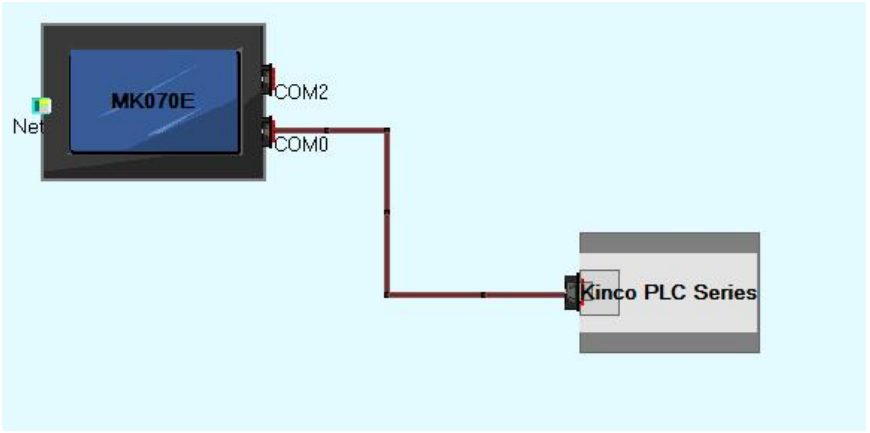
② 选择 Kinco PLC Series 通信协议

③ 点击添加（通信参数已固化不需要再手动进行设置）

④ 点击完成



完成后的硬件组态连接图如下：



注意：

实际 PLC 部分的通信参数如下图，PLC 内部已将通信参数内部固化不需要在 KincoBuilder 的工程硬件配置中再进行设置。

二、编辑组态画面的编辑，可以参考 Kinco DTools 组态编辑软件使用手册。

三、HMI 对外提供网口，USB 主从口，具体使用参考 Kinco DTools 组态编辑软件使用手册

2.1.2 HMI 使用手册下载链接

下载链接：

https://www.kinco.cn/Download/usermanual/HMI/Kinco%20DTools%E7%BB%84%E6%80%81%E7%BC%96%E8%BE%91%E8%BD%AF%E4%BB%B6%E4%BD%BF%E7%94%A8%E6%89%8B%E5%86%8C20200330.pdf

附录 E 入门案例（建立一个工程的步骤）

假定用户已准备好所需的硬件。

为了帮助用户迅速了解和掌握 Kinco 小型 PLC，下面我们将一步步地举例说明如何创建、调试一个具体的工程。请注意这些只是常见步骤的描述，若用户需要详细地了解某一方面的具体功能，请参阅相关章节部分。

假定要编写如下工程：

- 工程名称：project
- 硬件：一个 Kinco-KS105-16DTCPU 模块、一个开关量模块 KS122-12XR、一个模拟量模块 KS133-06IV。
- 实现功能：依次启动 Q0.0 --- Q0.7，并且循环执行这个过程，输出 0-10V 模拟量控制伺服以固定速度运动，输入 I0.0 控制整个过程的启停。

为了保证良好的程序结构，我们在工程中使用了三个 POU：

一个名称为“Demo”的子程序，目的是实现要求的控制功能，循环启动 Q0.0 --- Q0.7，控制伺服运动；

一个名称为“INT_0”的中断程序，控制 Q0.0 --- Q0.7 循环移位的启停和伺服的启停；

一个名称为“Main”的主程序，这是 CPU 的主循环任务，在主程序中将调用“Demo”子程序，执行中断程序。

1) 启动 KincoBuilder。

2) 若需要的话，使用如下方式来设置 KincoBuilder 软件中用到的一些默认值。

执行**【工具】→【软件设置…】**菜单命令，进入“软件设置”对话框。在这个对话框中，用户可以对一些软件默认值进行设置，比如**【默认编程语言】**、**【默认 CPU 型号】**等。单击**【确认】**按钮后，设置的默认值将生效并由 KincoBuilder 自动保存，不需要每次都进行设置。

在此我们设置【**默认编程语言**】是“梯形图 LD”。

3) 使用如下任一方法建立一个新工程：

- 执行【**文件**】->【**新建工程..**】菜单命令；
- 单击工具栏上的 图标。

然后就会出现“新建工程”对话框。用户在此为新工程命名并选择存放路径。路径、工程名称确定以后，单击【**保存**】按钮，创建新工程。本例中我们选择 D:\temp 作为工程存放目录，工程名称为“project”。

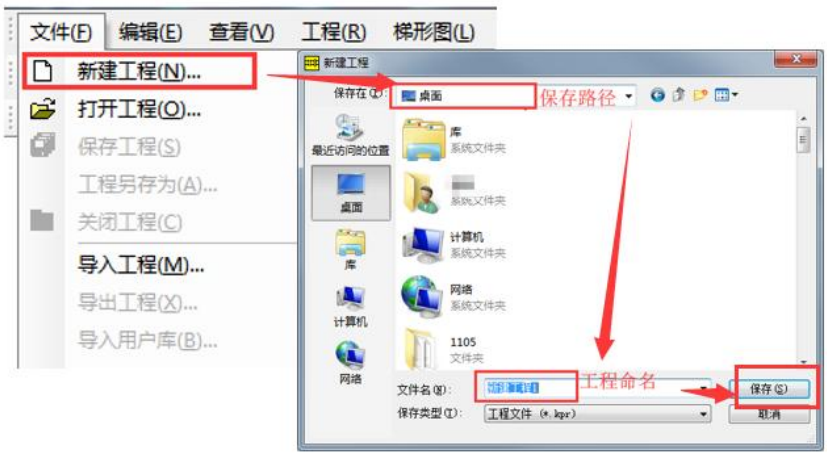


图 2-11“新建工程”对话框

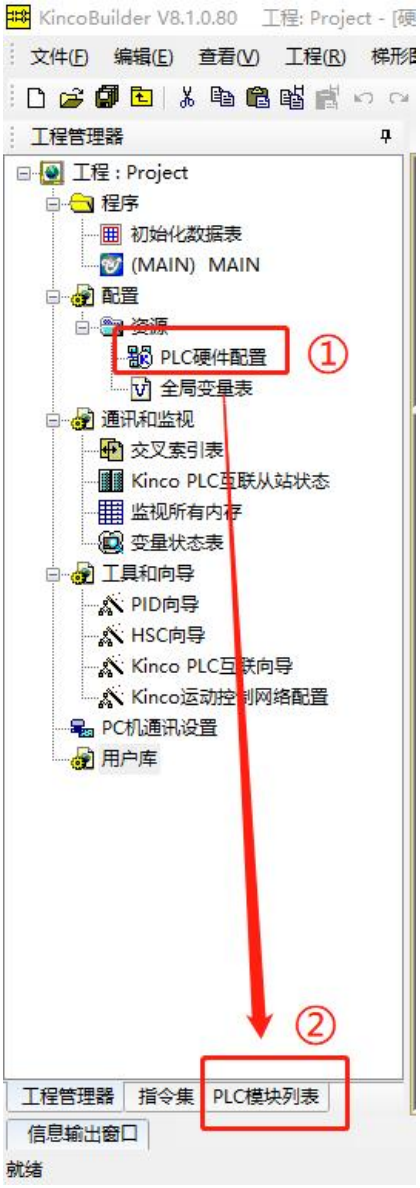
4) 进行 PLC 硬件配置（新建工程后自动生成一个叫做 MAIN 的主程序）

当用户新建一个工程时，KincoBuilder 将自动加入一个用户在“软件设置”对话框中指定的【**默认 CPU 型号**】的 CPU 模块。用户可以在任何时候修改硬件配置参数，但由于硬件配置是一个工程中必需的信息，因此建议用户在工程中首先完成硬件配置。

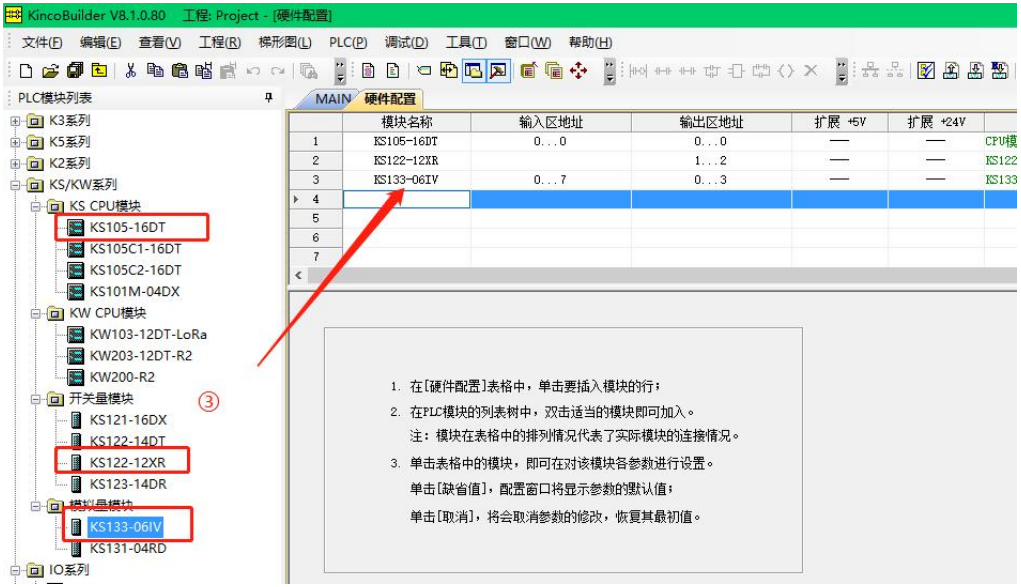
下图展示此工程中硬件配置的过程。

① 打开硬件配置

② 打开 PLC 模块列表

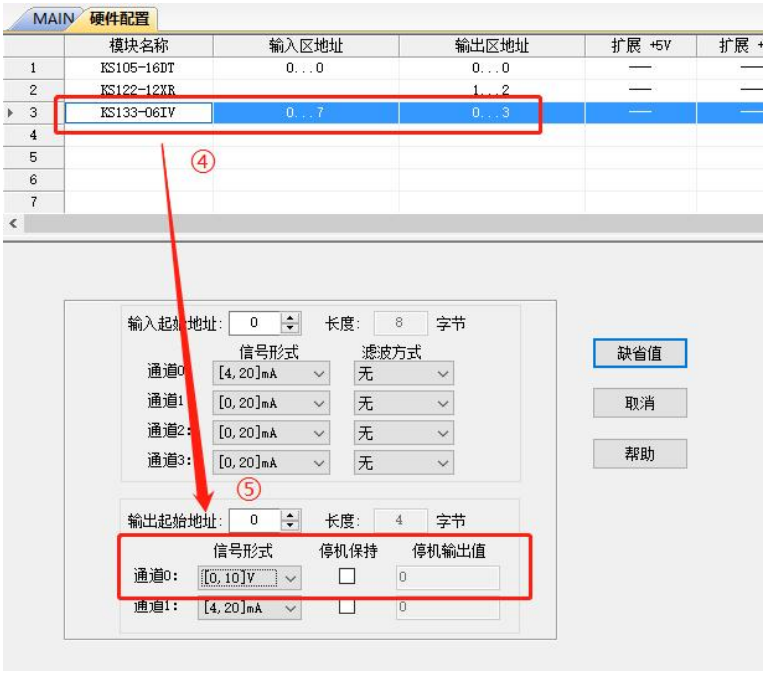


③ 在 PLC 模块列表中双击选择需要用到的 CPU 模块和扩展模块，在硬件配置中选中相关 CPU 模块和扩展模块可以右键进行删除。



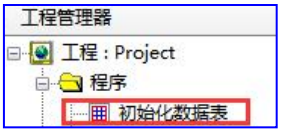
④ 选中需要设置的模块或 CPU 对硬件参数进行设置

⑤ 设置模拟量输出通道 0 信号形式为 0-10V，用于控制伺服。



上述仅设置了需要用到的配置，详细说明请参阅 4.3 PLC 硬件配置来了解进行硬件配置的具体步骤，在此不再一 一累述。

5) 关于初始化数据表



根据实际需要，用户可以随时填写初始化数据表。在初始化数据表中用户可以为 V 区中的 BYTE、WORD、DWORD、INT、DINT、REAL 类型的数据指定初始值。在 CPU 上电时进入主循环之前，初始化数据表将被处理一次，用户指定的初始值被赋给相应的地址，使用参照 4.4 初始化数据表。

注意：“初始化数据表”、硬件配置的“数据保持”以及用户程序中使用指令永久保存的内存区域均会在上电之后进入主循环之前进行恢复或者赋值，其次序是：恢复“数据保持”中定义的内存数据、“初始化数据表”中定义的内存区域赋初始值、恢复用户使用指令永久保存的数据。

6) 关于全局变量表

为了便于用户编写程序，增强程序的可读性，用户可进入全局变量表中对需要用到的变量进行定义，使用参照 4.5 全局变量表。

① 打开全局变量表

② 定义变量名和地址等

7) 根据实际的功能需求来编写程序

用户编程可以使用两种语言：IL 或者 LD。用户可以随时执行**【工程】→【IL 语言（指令表）】**

或者【工程】→【LD 语言（梯形图）】菜单命令切换当前程序的语言。注意：任何 LD 程序都可以转换为 IL 程序，但只有按照一定规则编写的 IL 程序才可以转换成 LD 程序。

下面我们将针对本示例来编写一个主程序“Main”、一个子程序“Demo”和一个中断程序“INT_0”。

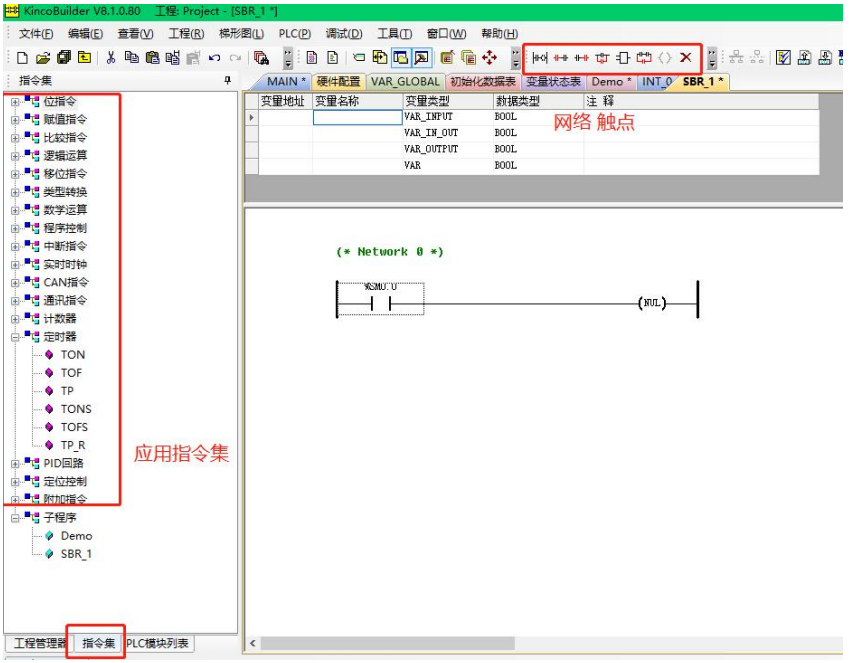
① 主程序

当用户建立一个新工程时，软件将会自动创建一个主程序，其名字为“MAIN”。

② 由于在主程序中只需要调用子程序“Demo”，但该子程序尚未建立，所以现在我们先不管主程序了，继续建立子程序“Demo”。

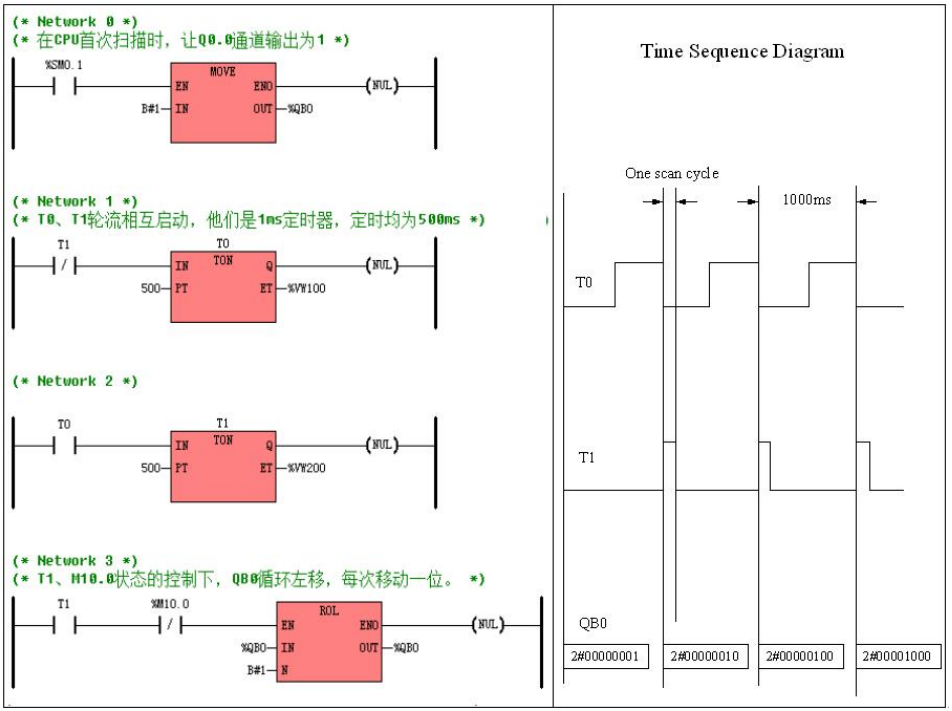
- 先使用如下任一方法建立一个新的子程序：
- 执行【工程】→【新建子程序】菜单命令；
- 单击工具栏上的  图标；
- 在工程管理器中的【程序】组节点上单击右键，执行弹出菜单中的【新建子程序】命令。

然后就会建立起一个新的子程序，其名字默认是“SBR_0”。

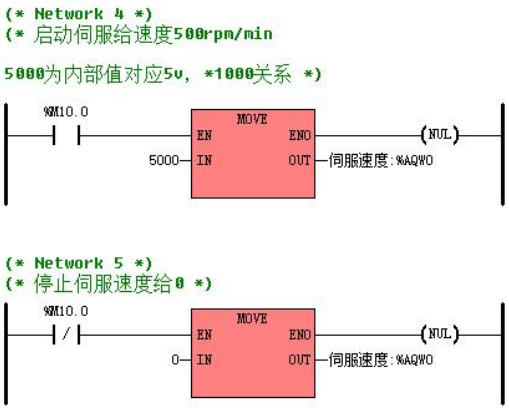


在红框的地方可以选择添加网络，增加触点，并联触点及添加相应的指令等，详细说明请参阅 5.2 LD 编程，在此不再详细描述。

现在可以先输入如下图所示的程序，图中右侧部分是时序图。



循环启动 Q0.0 --- Q0.7 部分程序



控制伺服启停部分程序

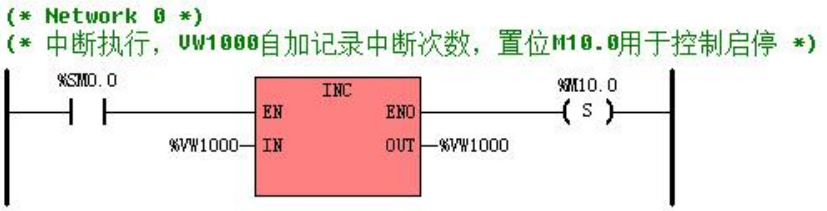
程序写完后，**关闭该子程序**，然后在工程管理器【**程序**】组中的【（SBR00）SBR_0】节点上单击

鼠标右键将会弹出菜单，执行【**重命名**】命令将名字改为“Demo”，或者执行【**属性**...】命令，在该程序的“属性”对话框中进行修改也可以。

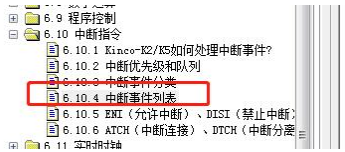
③中断程序

- 先使用如下任一方法建立一个新的中断程序：
- 执行【**工程**】→【**新建中断程序**】菜单命令；
- 单击工具栏上的  图标；
- 在工程管理器中的【**程序**】组节点上单击右键，执行弹出菜单中的【**新建中断程序**】命令。

然后就会建立起一个新的中断程序，其名字默认是“INT_0”。



中断事件可参照帮助了解对应的事件分类、列表等。



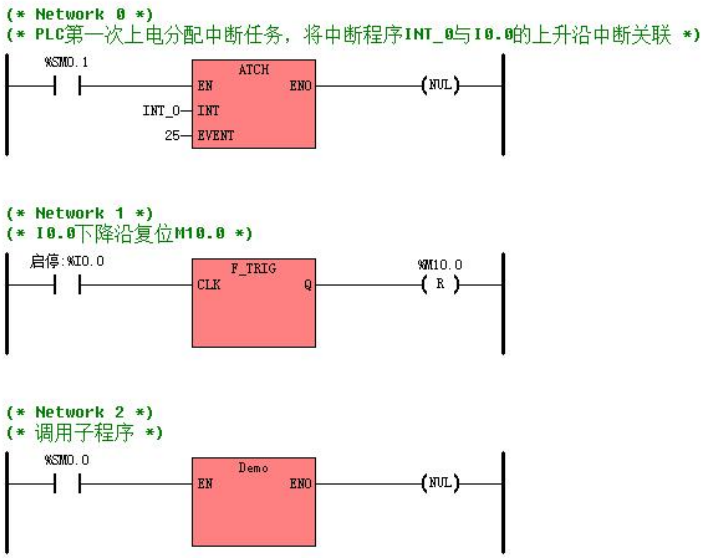
29	PORT 0：接收数据完成
28	PTO 1 完成
27	PTO 0 完成
26	I0.0检测到下降沿
25	I0.0检测到上升沿
24	I0.1检测到下降沿
23	I0.1检测到上升沿

本示例没有重命名该中断程序，需要重命名中断程序的用户，重命名的方法参照子程序即可。

④ 重新修改主程序

现在我们编完了子程序“Demo”，需要重新返回到主程序中加入对它的调用指令。

切换到主程序的编辑窗口，输入以下程序：



8) 编译工程

编写完工程后，就可以对它进行编译了。在编译之前，软件会自动保存本工程，以确保编译的是用户最新的输入。但是，仍然建议用户注意随时保存自己的工作，以免有意外发生。

用户可以使用如下任一方法来启动编译过程：

- 执行【PLC】→【**编译当前工程**】菜单命令；
- 单击工具栏上的 图标；
- 使用快捷键 **F7**。

编译的结果将会在下部信息输出窗口中的“编译信息”页面中。如果工程中有错误，双击编译信息中的错误信息就会自动定位到该错误所在的位置。用户需要根据错误信息提示对工程进行修改，直至编译成功。

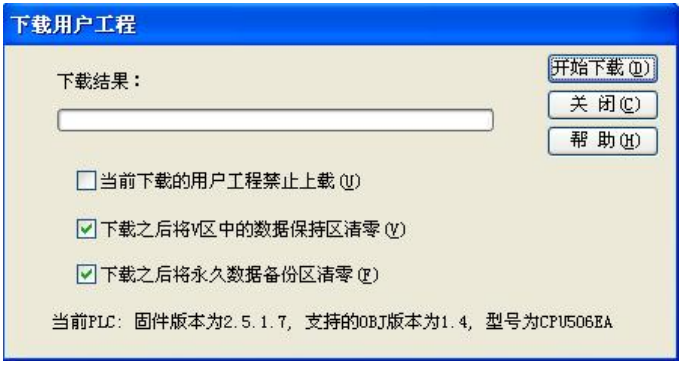
9) 下载工程

编译成功后，就可以将当前工程下载至 PLC 中了。

补充一下：若需要指定 PC 机串口的通信参数，请参阅 2.5 如何连接计算机与 PLC。

- 用户可以使用如下任一方法来打开“下载用户工程”对话框：
- 执行【PLC】→【下载..】菜单命令；
- 单击工具栏上的  图标；
- 使用快捷键 F8。

➢ “下载用户工程”对话框：



✧ 【当前下载的用户工程禁止上载】

若选中此项，则在本次下载之后 CPU 将禁止任何人上载工程。

若不选中此项，则在本次下载之后 CPU 将允许用户按照原有的权限进行上载。

✧ 【下载之后将 V 区中的数据保持区清零】

若选中此项，则在本次下载之后，数据保持区域 V、C 区中的数据将全部清零。

若不选中此项，则在本次下载之后，数据保持区域 V、C 区中的数据将保持不变。

✧ 【下载之后将永久数据备份区清零】

若选中此项，则在本次下载之后，永久数据备份区中的数据将全部清零。

若不选中此项，则在本次下载之后，永久数据备份区中的数据将保持不变。

在“下载用户工程”对话框中设置完成后，单击**【开始下载】**按钮即可将工程下载至 PLC 中。现在可以将 CPU 的运行开关放至“RUN”位置，看看程序的运行情况吧。

10）程序调试

用户可以使用 KincoBuilder 提供的在线监视、强制等功能对程序进行调试。

➢ 在线监视

在线监视有两种模式：

- 在变量状态表中进行监视。用户可以在表中输入想要监视的变量进行监视。
- 在程序中进行监视。用户可以观察到当前程序的运行情况。在线监视时不允许修改程序。

在线监视命令只有在打开变量状态表、LD 程序或者 IL 程序之后方能生效。注意在线监视命令是一种复选命令，若要脱离在线状态，再执行一次在线监视命令即可。

用户可以使用如下任一钟方法进入在线监视状态：

- 执行【**调试**】->【**在线监视**】菜单命令；
- 单击工具栏上的  图标；
- 使用快捷键 **F6**。

➢ 强制

用户可以使用强制功能来强制修改 PLC 中 I、Q、M、V、AI 或者 AQ 区内变量的值，其中 I、Q、M 和 V 区中的变量可以进行位、字节、字或者双字方式强制，AI、AQ 区中的变量可以进行字强制。

KincoBuilder 允许同时强制最多 32 个变量。立即指令不允许强制。

用户可以通过如下任一种方式来执行强制功能：

- 在变量状态表中进行强制。用户可以在表中输入所要监测的变量及其强制值，然后执行菜单命令（或者右键菜单命令）【**强制**】、【**全部强制**】等实现相应的功能。
- 进入在线监视状态之后,直接在程序中对用到的变量进行强制。

  开关量：在触点、线圈上单击右键，执行弹出菜单中的【**强制为 0**】、【**强制为 1**】或者【**强制…**】命令；

  非开关量：在变量名称上单击右键，执行弹出菜单中的【**强制…**】命令。

在同一时刻，一个变量或许会存在如下取值的可能：外部的输入信号（I、AI）或者用户程序中指令执行的结果（Q、AQ、M、V），用户指定的强制值。因此规定了如下强制值生效的原则：

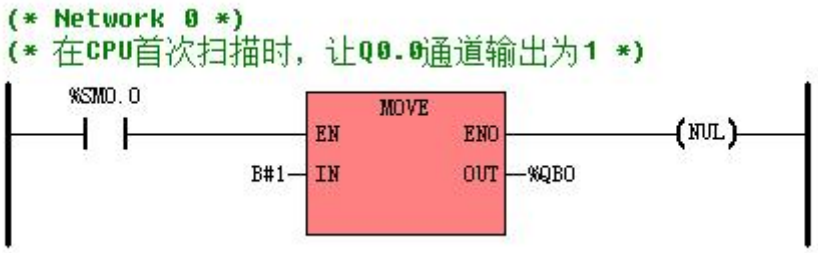
- ✓ 对于 M 区、V 区中的变量，强制值与指令执行结果处于同一优先级：用户在强制时，强制值将会生效；但强制值在一个扫描周期中只会生效一次，之后指令执行结果将会生效。
- ✓ 对于 I 区、AI 区中的变量，强制值优先于外部信号输入值。若用户指定了强制值，则强制值将会优先在用户程序中生效。
- ✓ 对于 Q 区、AQ 区中的变量，在程序执行过程中以指令执行结果优先，但在扫描周期最后的输出任务中将会以强制值优先。

用户可以执行**【取消强制】**菜单命令来取消对某个变量的强制状态，或者执行**【全部取消强制】**命令来取消全部变量的强制状态。

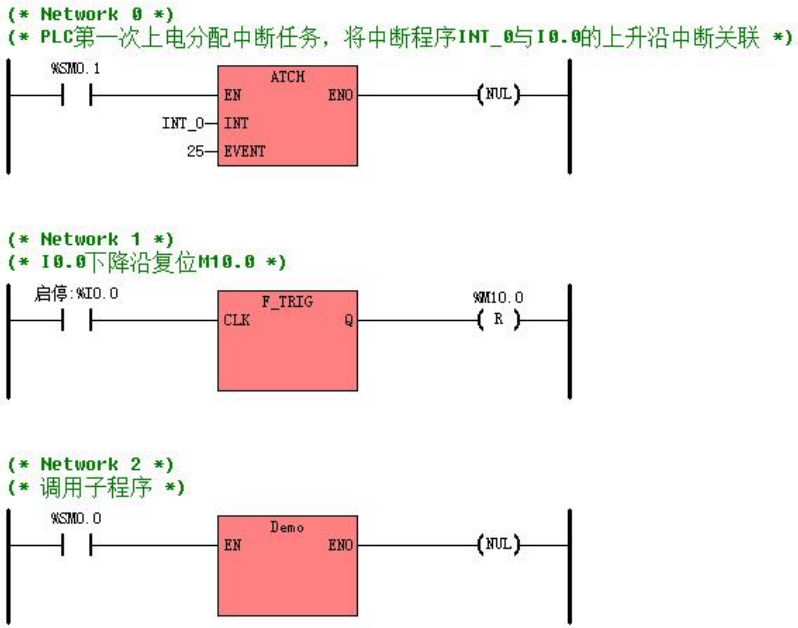
当 CPU 冷启动（重新上电）后，所有变量的强制状态都会被取消。

上面以摘要的形式描述了用户创建一个新工程以及进行调试等常用的步骤，为初次使用的用户提供了一个指南。关于各个部分更详细的描述请参见相关章节。

注意：程序指令中的常量请注明相关数据类型，不注明的话编译时会报错。比如下图需要注明常量 1 为字节，所以需要注明 B#1。相关说明请参阅 3.4 常量介绍。



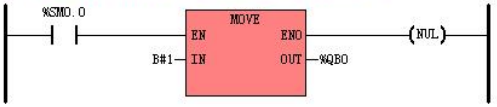
例程中的完整程序如下：

LD	MAIN：主程序用于分配中断任务，调用子程序，I0.0 上升沿产生中断置位 M10.0 下降沿复位，控制输出 QB0 循环移位，控制模拟量输出给定伺服速度。 

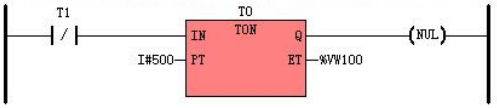
LD

子程序 Demo：自制 1s 脉冲，每隔 1S QB0 左移一次，M10.0 控制是否执行移位，M10.0 由 I0.0 控制，同时控制模拟量输出进而控制伺服启停。

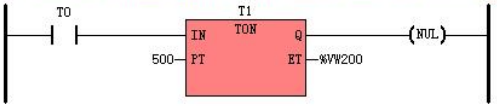
```
(* Network 0 *)
(* 在CPU首次扫描时，让Q0.0通道输出为1 *)
```



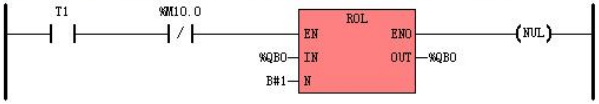

```
(* Network 1 *)
(* T0、T1轮流相互启动，他们是1ms定时器，定时均为500ms *)
```



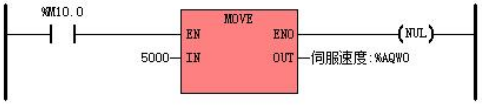

```
(* Network 2 *)
(* T0、T1轮流相互启动，他们是1ms定时器，定时均为500ms *)
```



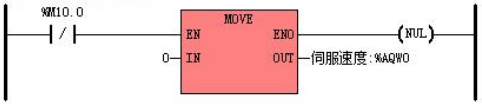

```
(* Network 3 *)
(* T1、M10.0状态的控制下，QB0循环左移，每次移动一位 *)
```

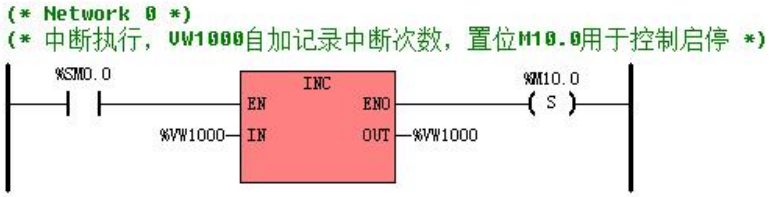



```
(* Network 4 *)
(* 启动伺服给速度500rpm/min
5000为内部值对应5v，*1000关系 *)
```




```
(* Network 5 *)
(* 停止伺服速度给0 *)
```



<table>
<tr><td>LD</td><td>中断程序 INT_0：记录中断次数，同时置位 M10.0，用于控制程序功能的启停。
 
</td></tr>
<tr><td>IL</td><td>MAIN：主程序用于分配中断任务，调用子程序，I0.0 上升沿产生中断置位 M10.0 下降沿复位，控制输出 QB0 循环移位，控制模拟量输出给定伺服速度。
(* Network 0 *)
(*PLC 第一次上电分配中断任务，将中断程序 INT_0 与 I0.0 的上升沿中断关联*)
LD %SM0.1
ATCH INT_0, 25
(* Network 1 *)
(*I0.0 下降沿复位 M10.0*)
LD 启停
F_TRIG
R %M10.0
(* Network 2 *)
(*调用子程序*)
LD %SM0.0
CAL Demo</td></tr>
</table>

IL	子程序 Demo：自制 1s 脉冲，每隔 1S QB0 左移一次，M10.0 控制是否执行移位，M10.0 由 I0.0 控制，同时控制模拟量输出进而控制伺服启停。
	(* Network 0 *)
	(*在 CPU 首次扫描时，让 Q0.0 通道输出为 1*)
	LD %SM0.0
	MOVE B#1, %QB0
	(* Network 1 *)
	(*T0、T1 轮流相互启动，他们是 1ms 定时器，定时均为 500ms*)
	LDN T1
	TON T0, I#500
	__CR_EQ_1
	MOVE T0, %VW100
	__CR_RESTORE
	ST %BR0.0
	(* Network 2 *)
	(*T0、T1 轮流相互启动，他们是 1ms 定时器，定时均为 500ms*)
	LD T0
	TON T1, 500
	__CR_EQ_1
	MOVE T1, %VW200
	__CR_RESTORE
	ST %BR0.0
	(* Network 3 *)
	(*T1、M10.0 状态的控制下，QB0 循环左移，每次移动一位*)
	LD T1
	ANDN %M10.0
	ROL %QB0, B#1
	(* Network 4 *)
	(*启动伺服给速度 500rpm/min5000 为内部值对应 5v，*1000 关系*)
	LD %M10.0
	MOVE 5000, 伺服速度

IL	(* Network 5 *) (*停止伺服速度给 0*) LDN %M10.0 MOVE 0, 伺服速度 中断程序 INT_0：记录中断次数，同时置位 M10.0，用于控制程序功能的启停。 (* Network 0 *) (*中断执行，VW1000 自加记录中断次数，置位 M10.0 用于控制启停*) LD %SM0.0 INC %VW1000 S %M10.0

附录 F 扩展模块的使用

1、综述

KPLC 各个系列产品均提供了相应的 CPU 模块和各种类型的扩展模块。

CPU 模块是 PLC 系统的核心，用于循环执行主扫描的工作任务。由于 CPU 模块本体集成了通讯口和一定数量的 IO 点数，因此可以独立使用。若系统规模较大，CPU 模块的 IO 点数不能满足系统需求时，用户就可以使用 CPU 模块和扩展模块灵活组合出适应各种应用的中小型控制系统。

下表列出了各类型 PLC 可以连接的扩展模块类型及数量。

系列	扩展模块类型	最大数量
K6	K6 及 K5 扩展模块	14
K5		4（K504EX）、14（其它 CPU）
MK	KS 扩展模块	14
KW		
KS（除 KS105C1 外）		
K209M		
K2（K209M 除外）	不支持	0
KS105C1		

注意：表格中的数量为各系列 PLC 所能连接的最大模块数量，实际可以连接单一型号的数量同时也受到相应的 IO 映像区大小的限制，比如 K508 的 AI 区长度是 64 字节，最多支持 32 个 AI 点，所以最多只能连接 8 个 K531-04IV 模块。各系列 PLC 内存区大小请查阅 3.6.4 内存区域的地址范围章节。

2、使用方法

1）硬件配置

在实际应用中，用户需要首先在用户工程的【硬件配置】中将用到 CPU 及模块按实际型号、排列次序依次添加到模块列表中并配置相关的参数，而且，实际使用的模块型号和排列次序必须与【硬件

配置】模块列表中的一致，否则 PLC 将会提示严重错误并进入 STOP 状态！

硬件配置

	模块名称	输入区地址	输出区地址	扩展 +5V	扩展 +24V	备 注
1	KS105-16DT	0...0	0...0	---	---	CPU模块, DC24V供电, DI 8*DC24V, DO 8*DC24V
2	KS123-14DR	1...1	1...1	---	---	KS123, DI 8*DC24V, DO 6*继电器
3	KS133-06IV	0...7	0...3	---	---	KS133, AI*4, AO*2, 4-20mA/0-20mA/1-5V/0-10V

如上图所示，用户工程的【硬件配置中】配置了上述 3 个模块型号，且 KS123-14DR 在 CPU 模块之后的第一个位置，KS133-06IV 在 CPU 模块的第二个位置（也就是紧随 KS123-14DR 之后的位置），因此在实际的模块连接时也必须按照同样的产品型号和连接顺序。

1）实际连接方式

- **K6 及 K5 系列**

K5/K6 系列是 KPLC 中的标准品系列。K6 是 K5 的升级版本，且保持了兼容性以方便用户进行维护和升级：K6、K5 的 CPU、扩展模块可任意搭配使用，即 K6 的 CPU 可以连接 K5 的扩展模块，K5 的 CPU 可以连接 K6 的扩展模块。

K6/K5 的扩展模块采用专用的扁平软线缆连接，如下图：

- **KS、KW 系列及 K209M**

KS、KW 系列和 K209M 的 CAN1 口，可作为扩展总线接口连接 KS 系列扩展模块。

KS 扩展模块采用了 RJ45 接口形式，用户可使用屏蔽的超五类双绞直连网线来连接扩展模块。扩展模块提供一进一出的两个接口，方便用户连接多个扩展模块时使用。具体连接可参考下图：

- **MK 系列一体机**

受限于成本和体积需求，MK 系列一体机在一个 DB9 接头内提供了 CAN1 和 2 个 RS485 通信口，所以由 MK 本体连接第一个 KS 扩展模块时用户需要自己制作一根通信电缆。

CAN1 接口的引脚定义及连接关系如下表格所示，注意 CAN_H、CAN_L、CAN_GND 管脚需要一一对应连接好。

MK 本体 DB9	KS 扩展模块 RJ45	定义
引脚 7	引脚 1	CAN_H
引脚 2	引脚 2	CAN_L
引脚 5	引脚 3	CAN_GND

附录 G 扩展 BD 板的使用

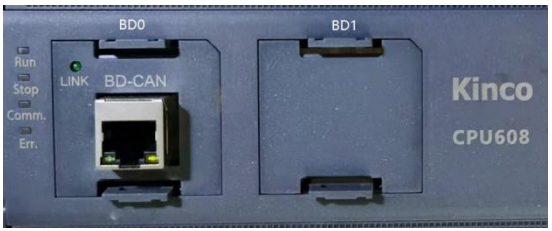
扩展 BD 板（Basic Unit Expansion Board）是一种提供少量 IO 通道或者通信口的电路板，可以插在 CPU 模块的 BD 板插槽上使用，为系统增加 IO 通道或者通信口数量。BD 板可以进一步丰富 CPU 模块的功能，而且与扩展模块相比，BD 板的价格更低，从而为用户提供了更高性价比的扩展方式。

1、综述

KPLC 提供了多种型号的 BD 板。从总体功能上，这些 BD 板可以划分为如下 2 类：

- 通信类：提供了串行通信（RS232/RS485)或者 CAN 等通信口。
- IO 类：仅提供了一定数量的 DI、DO、AI 或者 AO 等 IO 通道。

在支持此功能的 CPU 模块上提供了一个或多个用于安装 BD 板的插槽，如下图所示。用户使用时可以将 BD 板安装到这些插槽中。在 BD 板及插槽上有卡扣装置，以防止安装好的 BD 板脱落。



各 BD 板插槽都有唯一的位置编号，其规则是：从左侧开始，第 1 个插槽编号为 BD0，第 2 个编号为 BD1，以此类推。

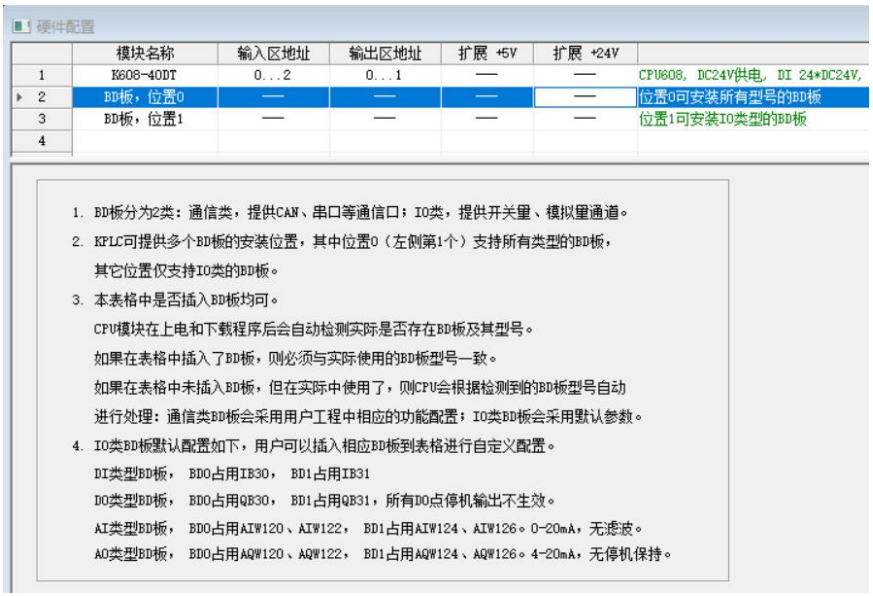
注意：1）通信类的 BD 板只能安装在 BD0 中；IO 类的 BD 板可以安装到所有插槽中。

2）用户必须先将 CPU 模块断电，然后才能安装或拆除 BD 板，禁止带电操作！

2、BD 板的使用

在 Kincobuilder 编程软件的【硬件配置】中提供了加入并配置 BD 板的功能，并且当用户单击表

格中某 BD 板位置行时，在下方窗口中将显示相关的说明信息：如果本行尚未加入 BD 板，则显示 BD 板总体的使用说明；如果本行已经加入了 BD 板，则显示相应型号 BD 板的使用说明。如下图。



CPU 模块在上电启动和下载程序后均会自动检测各 BD 插槽中是否安装了 BD 板及其型号。

因此，在实际应用中，用户是否在程序的【硬件配置】表格中加入 BD 板均可。如果用户在程序的【硬件配置】表格中加入了 BD 板，则表格中配置的型号必须与实际安装的 BD 型号一致，否则 CPU 模块会报错并进入 STOP 状态。如果用户在【硬件配置】表格中未加入 BD 板，但在 CPU 模块上实际安装了，那么 CPU 模块会根据检测到的 BD 板型号自动进行处理：对于通信类的 BD 板，CPU 模块将采用用户工程中相应通信口的参数及功能配置；对于 IO 类的 BD 板，CPU 模块将采用默认的地址及信号类型等。

2.1 各种 BD 板的功能及参数说明

1）串行通信口型号 KB6-2COM

该型号 BD 板提供了 1 路 RS232 通信口和 1 路 RS485 通信口。其中 RS232 口编号为 PORT0，RS485 口编号为 PORT3，通信参数均在【硬件配置】中 CPU 的【通信设置】中进行配置。

PORT0（RS232 口）支持编程协议、Modbus RTU 从站和自由通信功能。PORT3（RS485 口）支持 Modbus RTU 主站、Modbus RTU 从站和自由通信功能。

2）CAN 总线通信口型号 KB6-CAN

该型号 BD 板提供了 1 路 CAN 总线通信口，编号为 CAN2。

CAN2 支持 Kinco 运动控制、CANOpen 主站和 CAN 自由通信功能，其中，CANOpen 主站和 CAN 自由通信功能可以同时使用。各种 CAN 通信功能的使用及配置方法请参阅前文 10.5 CAN 总线的使用。

3）各种 IO 类型号

KB6-4DI 提供了 4 路晶体管型 DI 通道，额定输入电压 DC24V，可同时作为源型或漏型输入。

KB6-4DO 提供了 4 路光继电器型 DO 通道，额定输出电压 DC24V，可同时作为源型或漏型输出。每通道额定输出电流 300mA，最大输出电流 500mA，具有通道短路保护功能。

在不同插槽位置的 BD 板上的各种 IO 通道默认占用的内存地址及其它通道信息如下表。如果程序的【硬件配置】表格中加入了某型号的 BD 板，则用户可以对其内存地址、通道信息进行修改。

类型	BD 板通道默认内存地址		其它
	BD0	BD1	
DI	IB30	IB31	
DO	QB30	QB31	所有通道的停机保持值为 0
AI	AIW120	AIW124	信号形式默认为 0-20mA，无滤波。
AO	AQW120	AQW124	信号形式默认为 4-20mA，无停机保持。

例如，如果用户在 BD0 位置安装了 KB-4DO，则各通道默认的内存地址依次为 Q30.0、Q30.1、Q30.2 和 Q30.3；如果用户在 BD1 位置安装了 KB-4DI，则各通道默认的内存地址依次为 I31.0、I31.1、I31.2 和 I31.3。

附录 H KS 扩展模块作为 ModBus RTU 从站用法

KS 系列扩展模块，都提供一个标准 RS485 接口，目前出厂均可作为 ModBus RTU 从站供主站访问。（可通过查看序列号******19150****的中间 5 位数是否在此之后，仅在此之后的支持）。

- KS 扩展模块出厂默认通信参数：

站号	1
波特率	38400
奇偶校验	无校验
数据位	8
停止位	1

- 修改通信参数：

使用小工具（Kinco_Modbus_Module_Config_V10 请找厂家获取）通过 RS485 口修改。

注：更改完之后不能一键恢复出厂值，所以要求必须记住更改之后的参数！

使用方法：

打开小工具（如图一），左侧选择电脑的串口并设置通信参数为当前所连模块相同的参数（如默认：站号为 1，波特率 38400，无校验），右侧选择读取，提示读取成功，右侧框内显示的即为当前所连模块的通信参数，更改右侧框内的站号（站号最大可设置为 128）或波特率或校验方式，点击修改，提示修改成功，即将当前所连模块的 RS485 通信参数修改完成。

图一：小工具界面

➢ 扩展模块 ModBus 寄存器内存区域对照：

规格型号	内存区域	范围	类型	Modbus 功能码	对应的 Modbus 寄存器号（十进制）
KS121-16DX	I	I0.0-I1.7	DI	02	0-15
KS122-12XR	Q	Q0.0-Q1.3	DO	01，05，15	0-11
KS122-14DT	Q	Q0.0-Q1.5	DO	01，05，15	0-13
KS123-14DR	I	I0.0-I0.7	DI	02	0-7
	Q	Q0.0-Q0.5	DO	01，05，15	0-5
KS131-04RD	AI	AIW0-AIW6	AI	04	0-3
KS133-06IV	AI	AIW0-AIW6	AI	04	0-3
	AQ	AQW0-AQW2	AQ	03，06，16	0-1

➢ 简单应用示例：

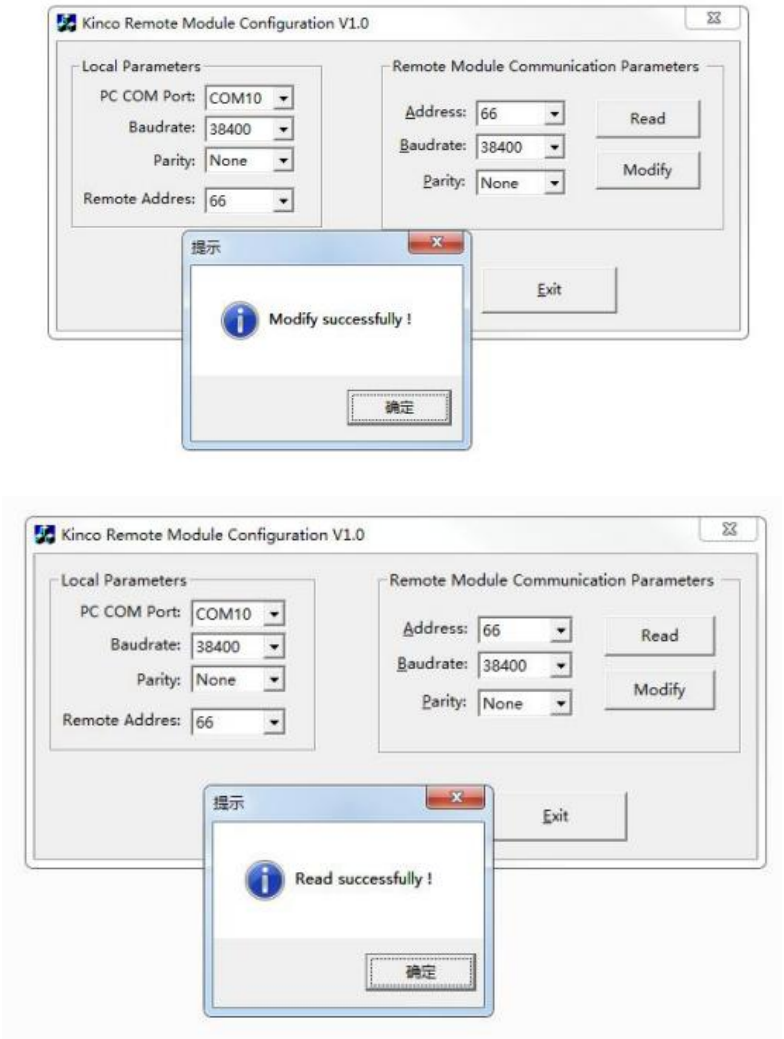
说明：

ModBus RTU 主站（K209M-56DT）通过 RS485 访问 3 个扩展模块（KS121-16DX、KS122-14DT、KS133-06IV）。

配置：

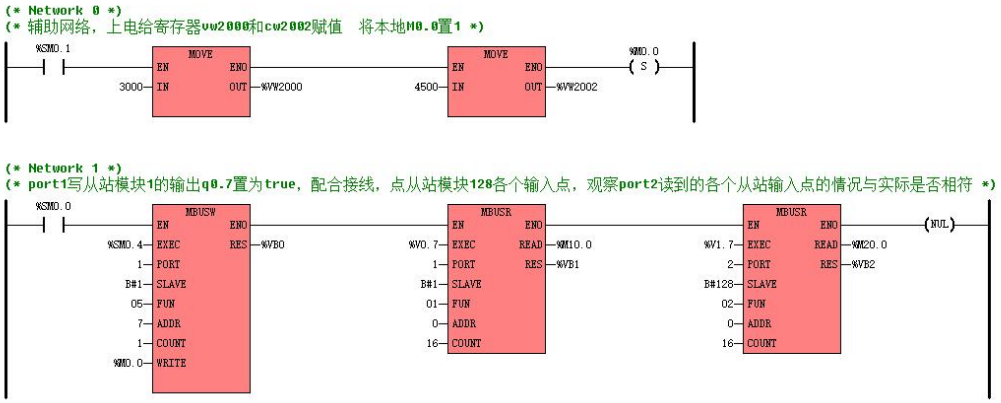
K209M-56DT Port1（参数：波特率 38400，无校验，8 数据位，1 停止位）+1*KS122-14DT（站号为 1）+1*KS133-06IV（站号为 66）；

Port2（参数：波特率 9600，无校验，8 数据位，1 停止位）+1*KS121-16DX（站号为 128）；



图二示例：修改模块站号为 66 后读取成功

图三：ModBus RTU 主站通信参数配置



图四：ModBus RTU 主站程序配置

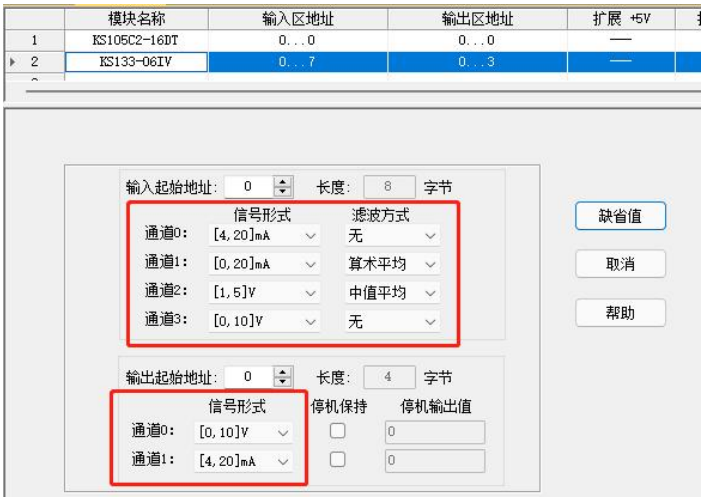
	内存地址	监视长度	显示格式	内存值	内存值	内存值
1	%M10.0	16	十进制(有符号)	FALSE	FALSE	FALSE
2	%M10.3			FALSE	FALSE	FALSE
3	%M10.6	从站1		FALSE	TRUE	FALSE
4	%M11.1			FALSE	FALSE	FALSE
5	%M11.4			FALSE	FALSE	FALSE
6	%M11.7			FALSE	从站1的Q0.7接从站128的10.4	
7						
8	%M20.0	16	十进制(有符号)	FALSE	FALSE	FALSE
9	%M20.3			TRUE	FALSE	FALSE
10	%M20.6	从站128		FALSE	FALSE	FALSE
11	%M21.1			FALSE	FALSE	FALSE
12	%M21.4			FALSE	FALSE	FALSE
13	%M21.7			FALSE		
14						
15	%VW1000	3	十进制(有符号)	3000	4500	0
16		从站66			本体模拟林输出接输入	
17	%VW1010	5	十进制(有符号)	0	4502	30
18	%VW1016			31	0	
19						
20						

图五：监控状态

注意：KS 扩展模块未使用扩展口连接 CPU 时，上电 RUN 灯（绿色）闪烁，为正常。

另外由于小工具（Kinco_Modbus_Module_Config_V10）只能修改通信参数，对模块的其他参数无法设置，比如模拟量模块的信号形式、滤波方式。所以需要通过 CPU 模块通过扩展总线的方式连接该模块提前设置好相关参数并通过 EX_ADDR 指令进行保存，然后再通过 RS485 进行连接。

硬件配置以 KS105C2-16DT 带模块 KS133-06IV 为例进行说明：



如图所示，假设模块的信号形式和滤波方式需要进行如上修改，编写保存参数的程序。

程序如下：