VECTOR
威科达科技

VEC-VA运动控制器
编程手册

©VAMM101C

WWW.szvector.com

目录

第一章 MULTIPROG 概述

MULTIPROG 是针对中大型控制应用场合开发的通用 PLC 编程系统，被广泛应用于机械制造、汽车和过程自动化行业。该工具基于微软的 COM/DCOM 技术架构，适用于 XP、Vista、win 7、win8 和 win10 视窗操作系统。其工程结构完全符合 IEC61131-3 标准，支持标准定义的 5 种编程语言，且允许用户自定义库和数据结构，支持第三方开发工具。该编程工具既可以和科维软件推出的运行时系统（ProConOS eCLR）配套使用，也可以运用于现有的控制系统，并且能对多个分布式 PLC 控制器进行统一配置、编程和下装程序。

MULTIPROG 提供了丰富的操作命令和优异的人机交互界面，支持拖拽和全键盘操作，提供了变量在线监视、强制和覆盖功能，允许对程序设置断点和单步调试，同时自带了逻辑分析器，可方便地记录输入输出波形。对于一些特殊场合，提供了源代码保护和不停机在线下装功能。针对不同国家的编程人员的使用习惯，提供了包括变量名称在内的多语言支持。

第二章 软件的特点以及硬件要求

该手册叙述了如何使用 MULTIPROG Express5.51 编程软件。用户根据本手册可在威科达运动控制器上编写自己的程序。

2.1 基础知识要求

用户要熟悉本手册，需要具有自动化技术的一般知识，了解 Windows 操作系统， 以及已经阅读了威科达公司的 《VA 运动控制器编程手册》。

2.2 MULTIPROG Express5.51 的特点

- 🕮 支持 IEC61131-3 编程语言——FBD、 LD、 IL、 ST 和 SFC。
- 🕮 工程结构清晰， 编程语言直观。
- 🕮 支持 FBD、LD 和 IL 之间的交叉编译，支持混合编程。
- 🕮 多用户编程，缩短工程编程时间。
- 🕮 向导、交叉参考等资源可进行高效编程。
- 🕮 兼容可进行统一版本管理。

2.3 计算机硬件要求

设备	最低	推荐
CPU	500MHz	1GHz
内存	256MB	1GB
硬盘	500MB	1GB
监视器	1 024×768	
操作系统	WindowsXP Pro、2000、 Vista、Win7、Win8、Win10， IE5.0 以上	
通讯	TCP/IP、 RS232	

2.4 MULTIPROG Express5.51 支持

内容	数量
工程树中的节点	8000
工程树中的配置/资源	1/1
每个资源的程序实例	1000
每个资源的任务	1
每个任务的程序实例	500
每个 POU 的全局变量/局部变量	1 5000/1 5000
包含的库	32
一个工程中的 POU 数量（包括库中的多个）	2000
一个工程支持的 I/O 数量	256 Byte
I/O 组	200

第三章 PLC 的工作原理

3.1 PLC 如何执行用户编写的程序

用户把程序下装到 PLC 然后运行 PLC，运动控制器就能循环执行用户的程序了。CPU 停止时，程序不执行。PLC 每执行一次用户的程序称为一个扫描周期，在一个扫描周期中将执行以下工作：

A】 读输入：将数字量、模拟量输入信号读取到输入映射区中。

B】 执行程序：执行用户的程序指令并将中间、最终数据存储在各种存储区中。

C】 处理通讯请求：执行通讯任务。

D】 CPU 自诊断：检查固件、程序存储器工作状态。

E】 写输出：在输出映射区中存储的数据被复制到物理输出点。

注：如果要实现一个每隔一段时间执行一段程序时要用周期任务，而不要在循环扫描任务中用计时器触发一个周期脉冲来实现，前者要准确些。

3.2 PLC 数据存取

1）用户的数据可存放在 PLC 的不同存储器单元，每个单元都有唯一的地址。 下表列出了不同长度的数据所能表示值范围；

数值	布尔（X）	字节（B）	字（W）	双字（DW）
无符号整数	0 到 1	0 到 255	0 到 65535	0 到 4294967295
有符号整数	—	-128 到 127	-32768 到 +32767	-2147483648 到+2147483647
32 位浮点数	—	—	—	+1.175495E-38 到 34022823（正数） -1.175495E-38 到 34022823（负数）

2）对于数据的存取必须指定地址，地址以 %开头，后续是位置前缀，大小前缀，用整数表示字节地址，用小数点 “.” 加整数表示位，如%IX0.0 表示输入映射区第 0 个字节的第 0 个位，下表是数据的地址特性

序号	前缀		定义	约定数据类型
1	位置前缀	I	输入映射区	
2		Q	输出映射区	
3		M	中间变量映射区	
4	大小前缀	X	位	BOOL
5		B	字节（8 位）	BYTE
6		W	字（16 位）	WORD
7		D	双字（32）	DWORD
8		L	长形（64 位）	LREAL

变量地址的示例

DI/DO 输入输出位操作：

%IX0.0 表示数字量输入映射区中第 0 个字节中的第 0 位，表示定义运动控制器中输入端子 DI0；

%IX1.0 表示数字量输入映射区中第 1 个字节中的第 0 位，表示定义运动控制器中输入端子 DI7；

%QX0.0 表示数字量输出映射区中第 0 个字节中的第 0 位，表示定义运动控制器中输入端子 DO0；

%QX1.0 表示数字量输出映射区中第 1 个字节中的第 0 位，表示定义运动控制器中输入端子 DO10；

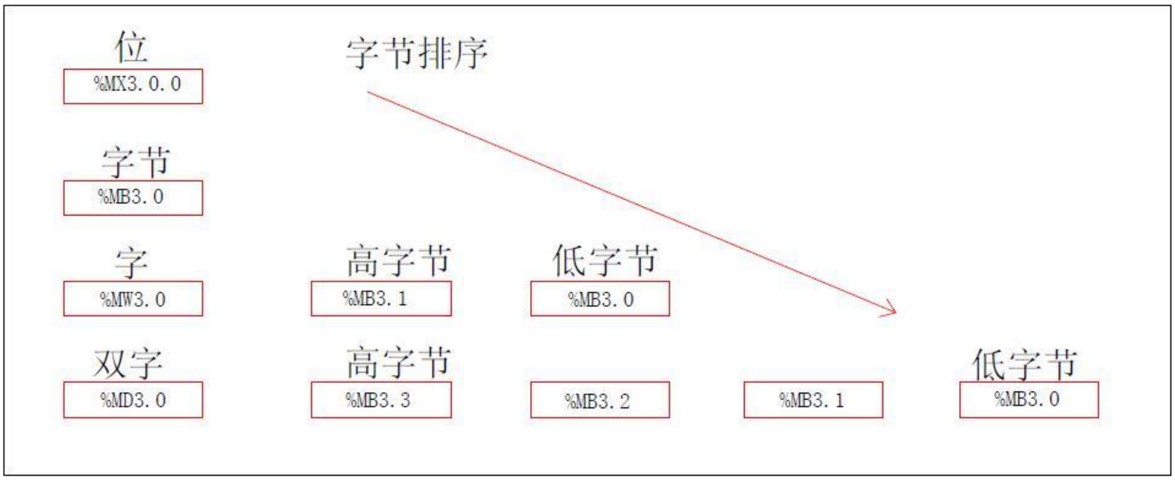
DI/DO 输入输出字节操作

%IB0 表示数字量输入映射区中第 0 个字节中的 8 个位状态，表示运动控制器中输入端子 DI0~DI7；

3）运动控制器地址之间的关系

运动控制器为了便于与外设（例如：人机，驱动器，其他控制器）进行数据交换，开放%MB3.0000~%MB3.9999 这 1 万字节地址与外设通讯,其中%MB3.2000~%MB3.3000 是掉电保存的数据地址。用户需要注意的是，不需要与外设交换的数据无需填写%MB3.XXXX 地址，只需填写变量名和选择正确的数据类型即可软件自动分配地址。

字节、字和双字之间的地址关系是一个双字包含两个字，或者说包含四个字节，下面以%MX3.0.0、%MB3.0、%MW3.0 和%MD3.0 为例说明字节、字和双字之间的地址关系和数据排列 如下图所示：



如：一个十六进制数 16#1234 存放在%MW3.0 中，则 16#34 存在%MB3.0 中，16#12 存放在%MB3.1 中。程序中若对位操作，则会影响到位所在字节、字和双字，反过来也一样。

变量地址的示例

%MX3.0.0 表示中间变量区中第 0 个字节的第 0 位；

%MB3.0 表示中间变量区中第 0 个字节；

%MW3.0 表示中间变量区中第 0 个字；

%MD3.4 表示中间变量区中第 4 个字节开始的 1 个双字

注：（5.3 版本的 1 个字节只能定义 1 个位，如%MX3.0.0，（表示已经占用一个字节），用户不能再使用%MX3.0.1~%MX3.0.7，如需定义下一个位，用户需从 %MX3.1.0 开始。）

PLC 支持常量值的输入，常量可以是二进制数、十进制数、十六进制数、字符串、ASCII 码或者实数，输入格式见下表。

数制	格式	举例
二进制	2#___	2#001
十进制	_____	67
十六进制	16#__	16#AC43
字符串	'___'	'VECTOR'
ASCII 码	ASCII 码值	16#30
实数	REAL#___	REAL#3.1415926

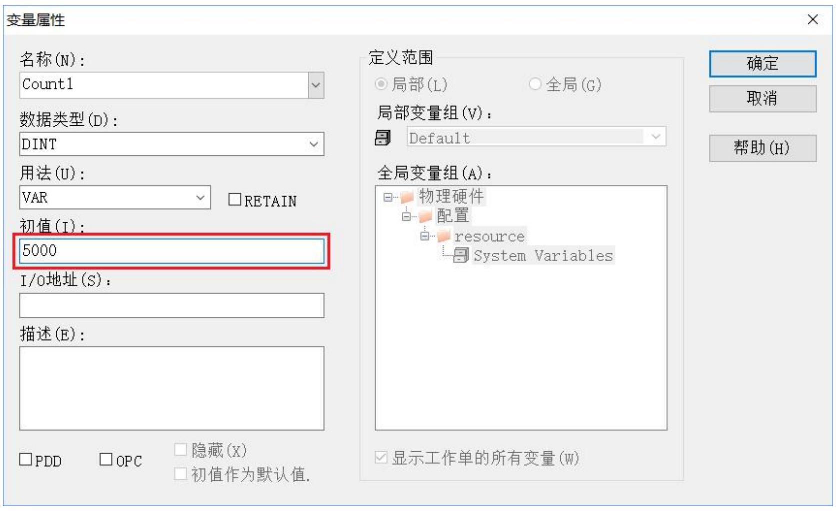
3.3 PLC 保存数据

1）首先，让我们了解一下运动控制器内部存储器的大概工作情况。运动控制器内部存储器分为 RAM 随机存储器和铁电永久存储器两种。我们都知道 RAM 存储器的数据必须靠供电维持，当存储芯片供电中断时，其中所存数据也不复存在。RAM 存储器主要用作运动控制器程序运行时程序、程序数据的实时存取空间。铁电存储器为带电可擦写存储器，其数据写入后可在完全断电情况下长期保存。运动控制器在每次程序下装时，将下装的程序、所需保存的数据（可选）、资源配置（可选）载入 RAM 存储区，同时 CPU 自动将其拷贝到铁电永存储器中，以实现永久保存。在运动控制器的使用过程中，每次上电时 PLC 会从铁电存储区向 RAM 存储区中恢复程序和资源配置。

🕮 运动控制器程序数据的断电保存方式；

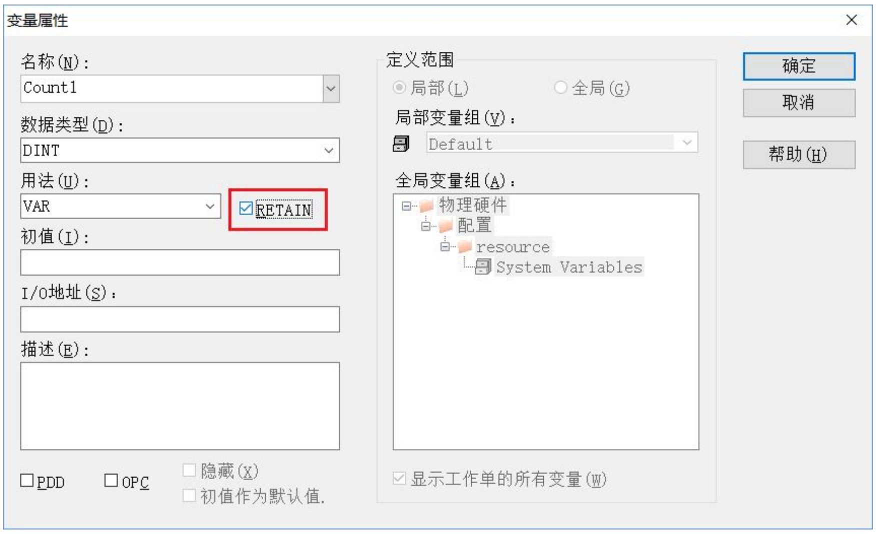
1）变量赋初值保存数据

编程时在需要保持数据的变量 “初值”下填写相应初值，当运动控制器重新上电后，该变量保存初值的数据 如下图所示；



2）变量设置保存数据

编程时用户可以选择需要断电后保持数据的变量，在变量属性“用法”下勾选“RETAIN”，表示该变量在运动控制器断电后，重新运行时能够保持断电前该变量的当前值。一般在不与外设通讯时使用（用户需要注意在变量勾上“RETAIN”时，“初值”上不能填写任何自然数，否者重新上电后变量的值还是保持初值的数据，而不是掉电前修改的数据）如图所示



注意 ： 这两者的区别是“变量赋初值保存数据”，当运动控制器重新上电后，只会保持该变量的初始值即使在程序运行过程有修改过该变量的数值；而“变量设置保存数据”是保持断电前该变量的数据。

3）特殊掉电保存数据地址

为了便于与外设进行数据交换，%MB3.2000~%MB3.3000 数据地址是掉电保存数据的，用户无需要任何的设置，只需填写地址即可实现，特别注意不需要在变量勾上“RETAIN”，否者编译通不过。（更多特殊寄存器说明请参考：附件五 寄存器说明）

第四章 数据类型

4.1 基本数据类型

程序包括代码和数据两部分，代码可以是 IL、ST、FBD、LD、 SFC 五种编程语言中的任何一种，或几种语言的组合，数据分为三种类型:基本数据类型、派生数据类型和用户自定义数据类型，数据必须通过变量的方式存在，数据类型决定了变量的格式、位数、可能值的范围初始值。

基本数据类型中声明

数据类型	描述	位长度	范围	默认初值
BOOL	布尔型	1	0 或 1（ture/falase）	0
SINT	短整型	8	-128 到+127	0
INT	整	16	-32768 到+32767	0
DINT	双整型	32	-2147483648 到 +2147483647	0
USINT	无符号短整型	8	0 到 255	0
UINT	无符号整型	16	0 到 65535	0
UDINT	无符号双整型	32	0 到 4294967295	0
REAL	实数	32	-3.402823466 E+38 到 -1.175494351 E-38 以及 +1.175494351 E-38 到 +3.402823466 E+38 注：科学计数法 7 位小数	0.0
LREAL	长实数	64	-11.798E+308 到 -2.225 E-308 以及 +2.225E-308 到+1.798 E+308	0.0
TIME	时间	32	0 到 4294967295 毫秒	T#0S
BYTE	字节	8	0 到 255（16#00..16#FF）	0
WORD	字	16	0 到 65535（16#00...16#FF）	0
DWORD	双字	32	0 到 4294967295 （16#00..16#FFFFFFFF）	0

4.2 类属数据类型

类属数据类型是把基本数据类型分级分组，以 ANY 作为数据类型的前缀，如，ANY_INT 表示包括 SINT、INT、DINT、USINT、UINT 和 UDINT 等全部整型数据。如果一个功能块的输入或输出与 ANYINT 相连，则意味着这个功能块可以处理 SINT、INT、DINT、USINT、UINT 和 UDINT 等整型数据的变量。

类属数据类型以如下形式组织:

<table>
<tr><td colspan="5">ANY</td></tr>
<tr><td colspan="2">ANY_NUM</td><td>ANY_BIT</td><td>STRING</td><td>TIME</td></tr>
<tr><td>ANY_REAL</td><td>ANY_INT</td><td rowspan="2">BOOL
BYTE
WORD
DWORD</td><td rowspan="2"></td><td rowspan="2"></td></tr>
<tr><td>REAL
LREAL</td><td>SINT
USINT
INT
UINT
DINT
UDINT</td></tr>
</table>

4.3 用户自定义数据类型

用户自定义的数据类型必须在工程“数据类型”中插入用户自定义数据类型，必须以 TYPE ...END_TYPE 声明块来完成，声明块中间部分是定义的衍生数据，衍生数据类型可以是结构，或者是数组。

◆ 数组

数组是一个单一数据类型对象的集合，同基本数据一样它具有唯一的名字，其中单个的对象并没有被命名，但用户可以通过它在数组中的位置对它进行访问，声明一个数组的例子如下:

```
TYPE
  graph:ARRAY [0...23] OF INT;
END_TYPE
```

注:ARRAY 型数组 graph 的最低字节是 graph[0]

◆ 结构

结构是多个不同数据类型对象的集合，同基本数据一样它有唯一的名字，结构的成员是基本数据类型或数组类型，也可以是另一个结构，或者嵌套。声明一个结构的例子如下:

```
TYPE
    machine:
    STRUCT
    x_pos：INT;
    y_pos：INT;
    depth：INT;
```

```
            rp：INT;
        END_ STRUCT;
        END _TYPE
```

◆ 字符串

字符串是由多个字符组成的有限序列，每个字符占用一个字节，字符串的数据类型是STRING，声明一个字符串时，其长度在数据类型后面的括号内设定，声明一个字符串的例子:

```
      TYPE
        STRING10:STRING（10）；
END_TYPE
```

在这个例子中，字符串的长度是 10，即 STRING10 是一个包含 10 个字符的字符串。最短的字符串长度为 1，最长的字符串长度为 32766。

4.4 常量数据的表示

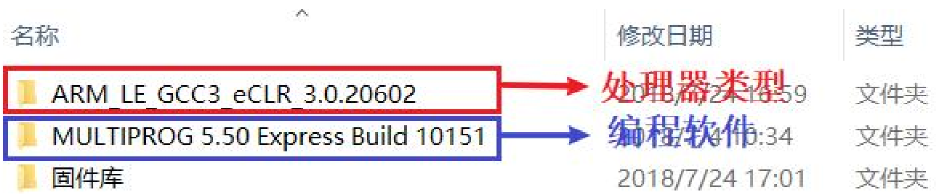
数据类型	描述	位长度	表示方法举例
BOOL	布尔	1	BOOL#0
SINT	短整型	8	SINT#- 128
INT	整型	16	INT#-32768
DINT	双整型	32	DINT#-2147483648
USINT	无符号短整型	8	US I NT#255
UINT	无符号整型	16	UINT#65535
UDINT	无符号双整型	32	UDINT#4294967295
REAL	实数	32	REAL#3.1415629
LREAL	长实数	64	LREAL#3.1415629
TIME	时间	32	T#10MS 、 T#10S 、 T#10M 、 T#10H,T#10D,T#1D_10H
DATE	日期	~	D#2011-07-24
TIME OF DATE	时刻		TOD#15:23:4555
TIME AND DATE	日期和时刻		DT#2011-07-24 一 15:23:4555
BYTE	字节	8	BYTE#16#FF
WORD	字	16	WORD#16#FFFF）
DWORD	双字	32	DWORD#16#FFFFFFFF）
STRING	字符串		‘VECTOR’

第五章 软件安装及介绍

得益于 MULTIPROG 优异的人机交互界面，只需简单的几步操作即可创建一个工程。本节将描述从如何安装 MULTIPROG 软件、软件界面的介绍到最终使用 MULTIPROG 的所有配置过程介绍。

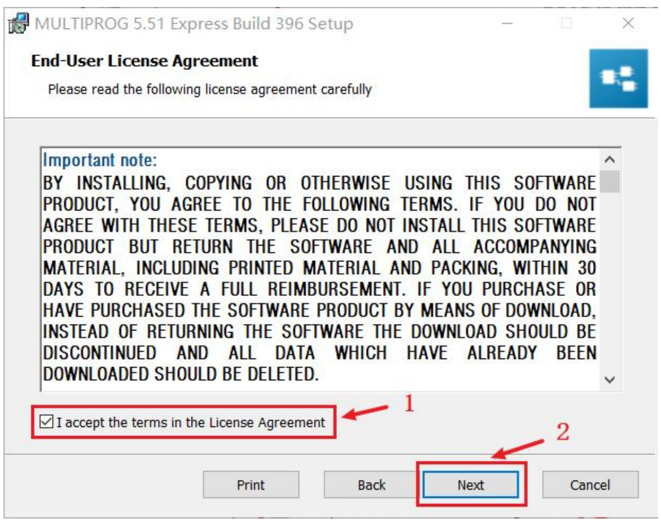
5.1 MULTIPROG 软件的安装与启动

1）将 MULTIPROG 的安装压缩文件解压缩，文件夹名中的“X”和“Y”都是数字，解压后会产生一个名为“MULTIPROGX.XXBuildYYY”的文件，表示该安装包的版本号。打开该文件夹，其中会有如图所示

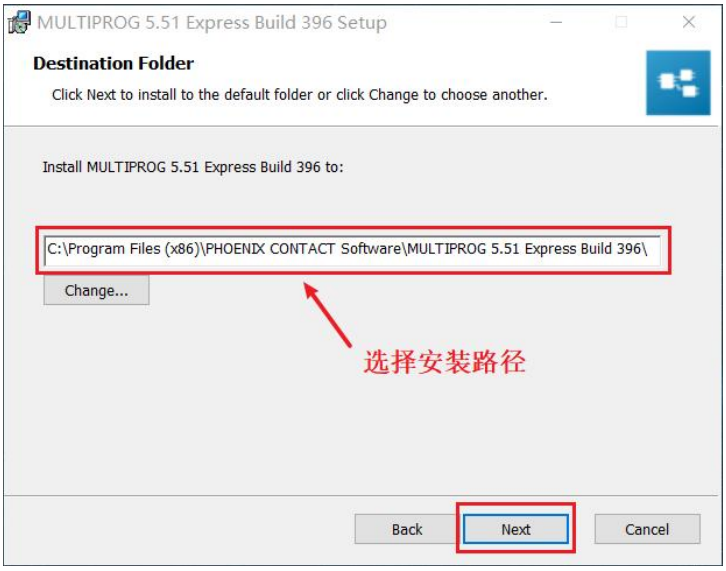


2）双击打开“MULTIPROG 5.50 EXPRESS Build 10151”，双击“setup.exe”，如图所示

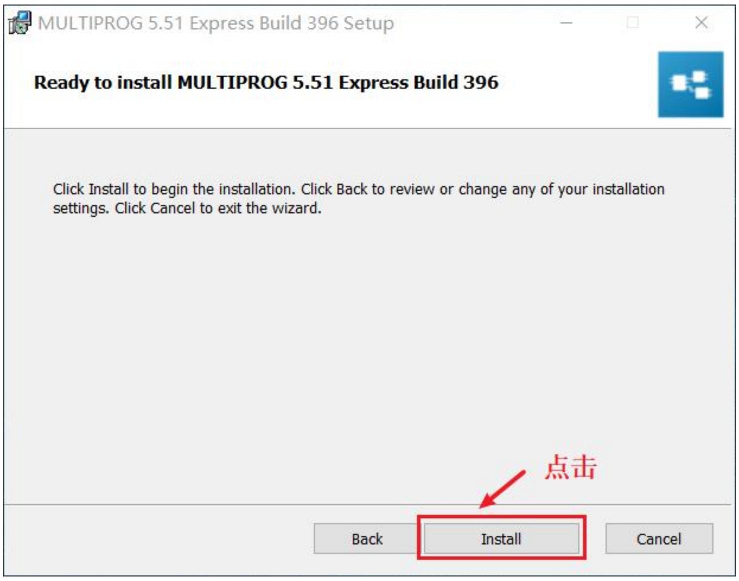
3）在弹出来的对话框中，勾上 “I accpet.........” 如图所示



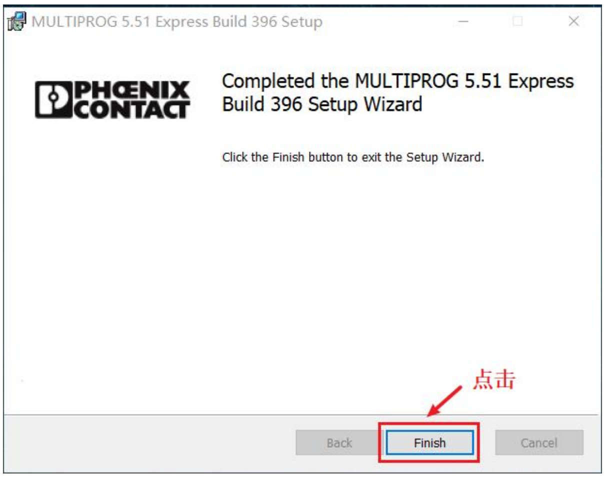
4）选择安装路径，用户可以自行选择，这里选择默认，单击“Next”如图所示



5）点击 Install，然后等待安装完成，安装过程可能需要一定的时间，如图所示



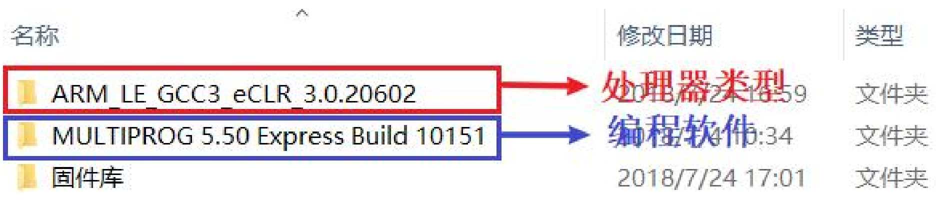
6）单击“Finish” 即可完成 MULTIPROG 编程软件的安装如图所示。



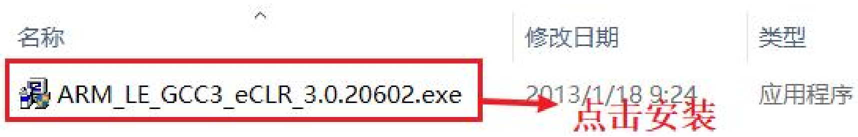
7）双击该图标即可启动 MULTIPROG。若桌面上未出现该图标，也可以使用“开始菜单”→“程序”→“PHOENIX CONTACT Software”→“MULTIPROG 5.51 Express”启动。

5.2 处理器类型软件安装

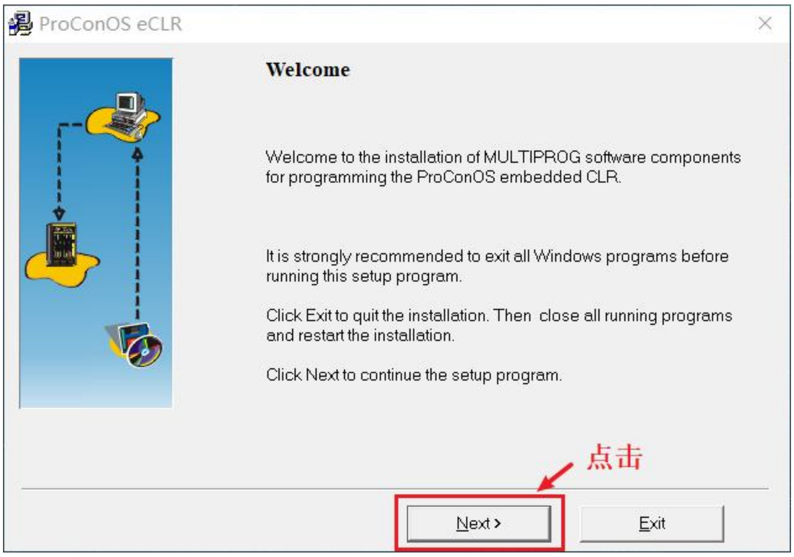
1）实物的 VA 运动控制器需要安装处理器类型才能使用，处理器类型“ARM_LE_GCC3”与实务的运动控制器的处理器一致（本公司的处理器类型为“ARM_LE_GCC3”所以需要安装该软件版本方可使用 VA 运动控制器），打开 ARM_LE_GCC3_eCLR_3.0.20602，如图所示



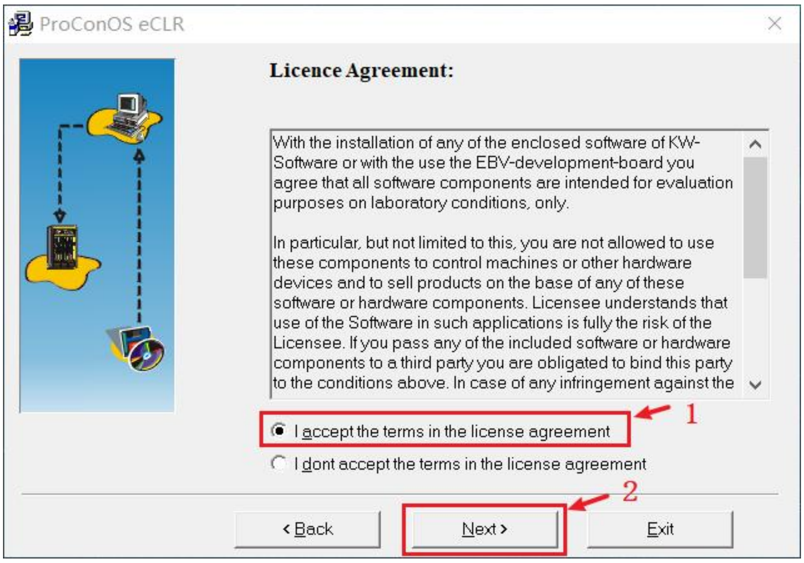
2）在安装前，（最好把安全软件关闭，否则安装过程会出现提示）双击安装，如图所示



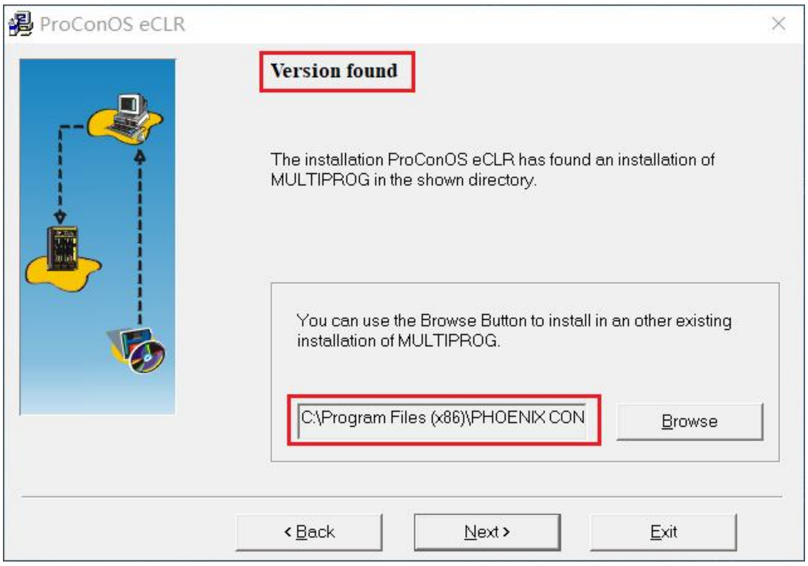
3）点击安装后，出现安装对话框，单击“Next”如图所示



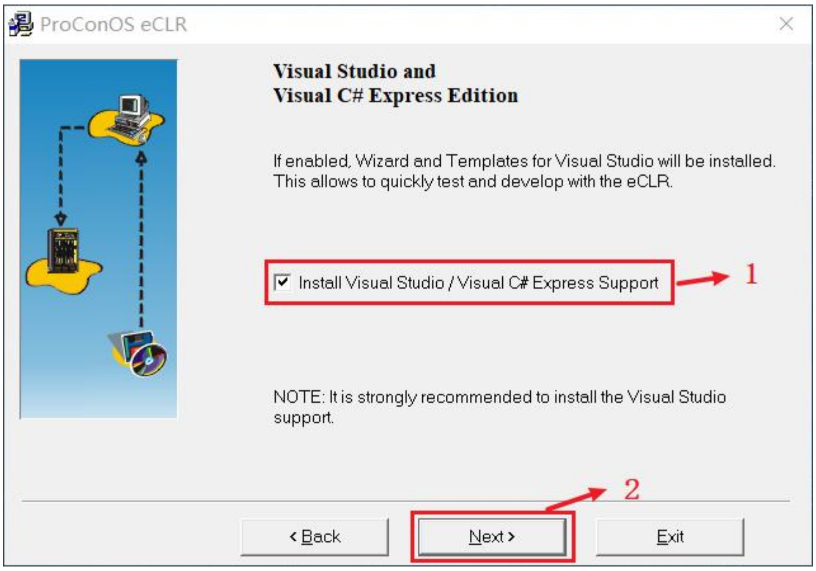
4）选择“I accept ……”，点击“Next” 如图所示



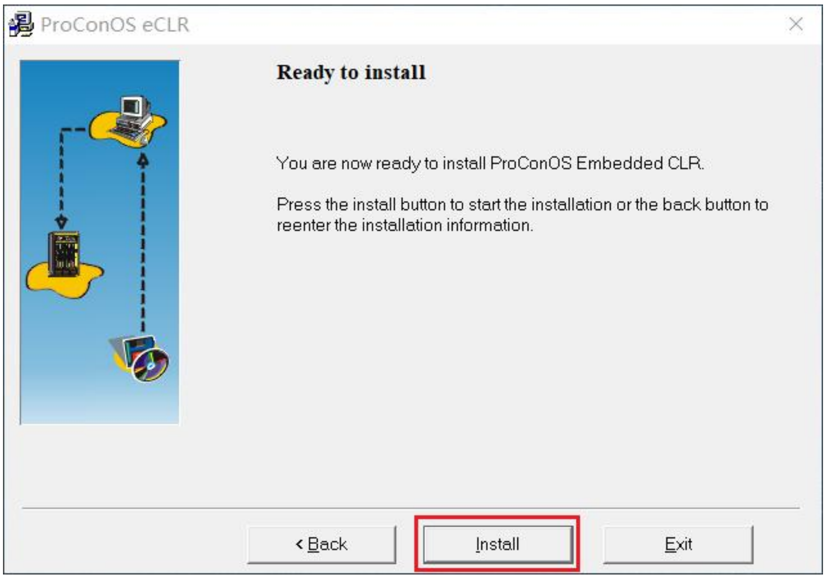
5）选择安装路径，这里选择默认，显示“Version found”，单击“Next”如图所示（特别注意；处理器类型软件一定要与 MULTIPROG 编程软件的安装路径一致，安装时处理器类型软件默认跟 MULTIPROG 编程软件安装路径一样，无需要更改，如果显示“Version not found”，则需要更改安装路径与 MULTIPROG 编程软件安装路径一样才能安装）



6）选择“Istall Visual Studio..” 单击“Next”如图所示



7）点击“Install”安装，安装过程需要一定时间，如图所示



8） 安装完成后，需要重启电脑，处理器类型软件便会和编程软件建立起关系，但需要一些参数的设置。

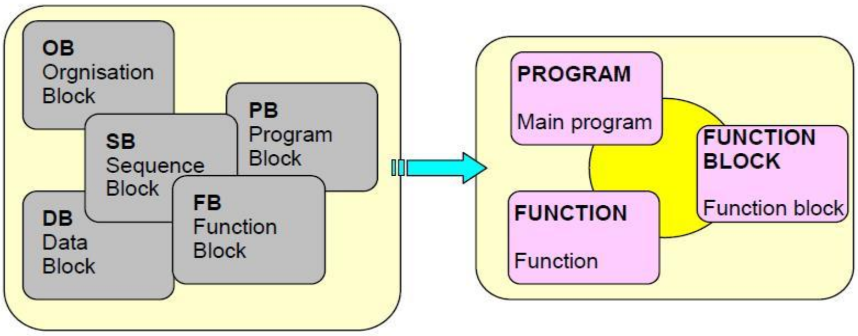
9）使用前用户必须把固件库文件夹下面的 system 文件夹下的 4 个文件复制到以下的路径：

名称	修改日期	类型
ARM_LE_GCC3.INI	2015/7/31 8:30	配置设置
eCLRDeviceProfiles.xml	2015/5/13 14:06	XML 文档
IO_DRIVER_LIB.dll	2015/3/10 14:26	应用程序扩展
SHM_LIB.dll	2015/3/24 13:41	应用程序扩展

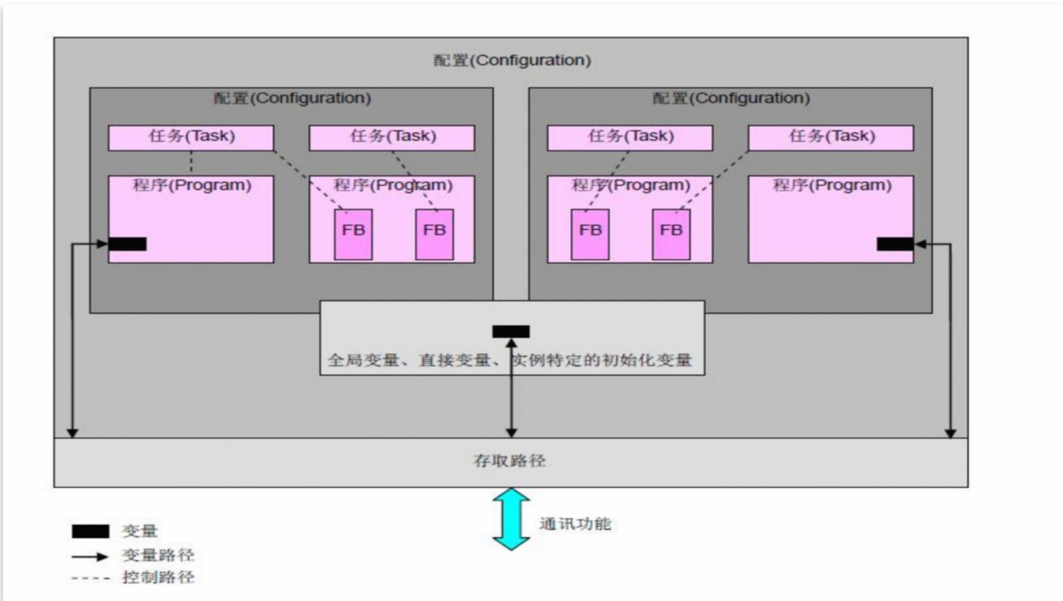
找到软件的安装路径，编者的安装路径是“D:\Program Files (x86)\PHOENIX CONTACT Software\MULTIPROG 5.35 Express Build 559\plc\eCLR\ARM_LE_GCC3”，将固件库文件下的 System 文件下的 4 个文件拷贝上述路径，选择“覆盖”原来的文件，完成后退出。

5.3 编程模型与 IEC61131-3 标准

1） VA 运动控制器使用的 MULTIPROG 编程软件，其编程语言和程序结构符合 IEC 61131-3 编程系统，在 IEC 61131-3 中，程序和工程的建立是在程序组织单元（POU）中完成的，程序组织单元 POU 包括程序（PROGRAM）、功能块（FUNCTIONBLOCK）和功能（FUNCTION）共三部分，代替了传统 PLC 编程语言的 OB、PB、DB、SB 和 FB 五种功能块，如下图，使得编程变得更加高效，更加简洁 如图所示。

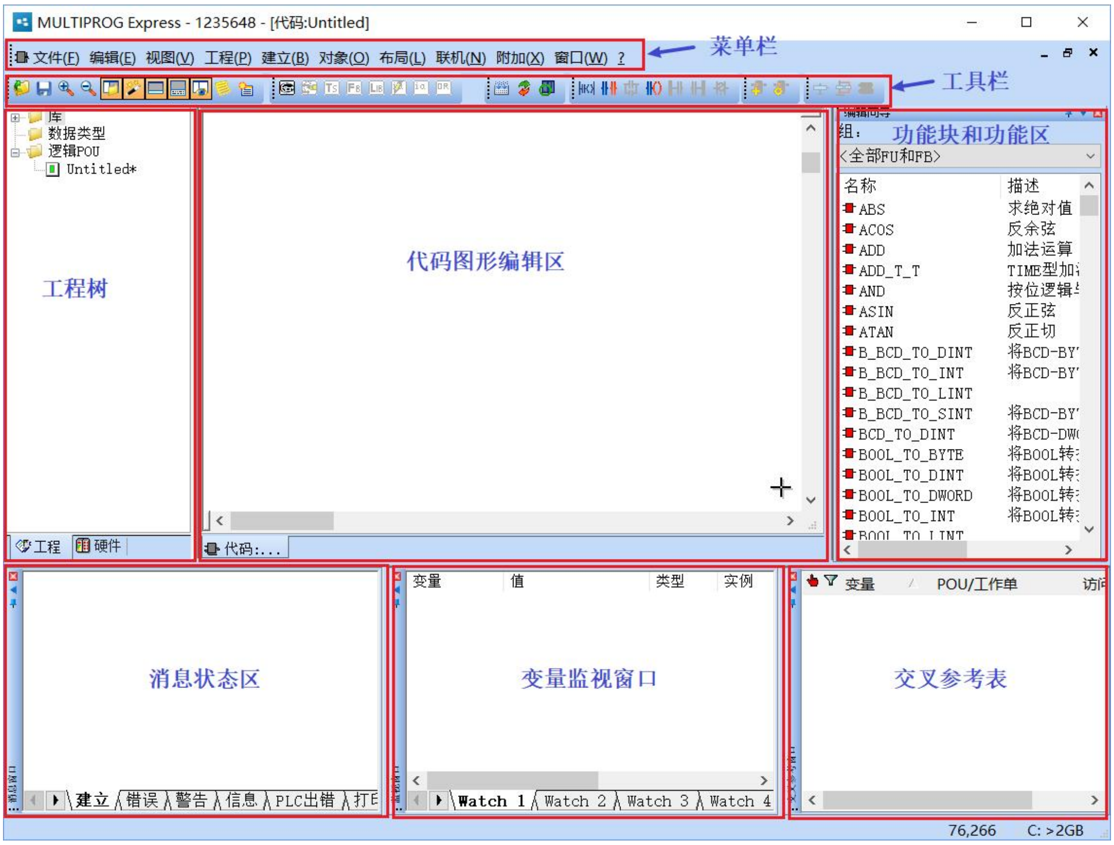


2）IEC 61131-3 编程的软件模型用分层结构表示，如下图，软件模型描述了各部分之间的相互关系，包括配置、资源、任务、程序组织单元、全局变量、I/O 配置等。编程过程中可将一个复杂的程序分解为若干个小程序或模块，也可将多个独立的程序同时下装、运行，或将程序分成多个任务去执行，提高了程序的模块化和运行效率如图所示。



5.4 MULTIPROG 编程界面介绍

打开 MULTIPROG 后，可以看到其只有一个主界，根据功能的不同，被划分成不同的区域，如图所示。



5.4.1 功能分区介绍

（1）工具栏区域里集合了用于代码编辑、调试特殊功能的命令;

（2）工程树则用于显示工程的结构和硬件的配置属性等; 工程树包括“硬件”和“工程”两部分，分别对应所建立工程的硬件和软件部分。

（3）代码图形编辑区在编辑状态下用于编辑文本或者图形代码，在调试模式下用于在线显示变量的值和程序的运行状态;

（4）消息状态区用于打开制作工程、联机调试、运行程序时各种信息的显示；

（5）交叉参考表，可看到变量的当前状态，地址；

（6）变量监视窗口，在大的工程项目中可能有很多的变量，可以添加你所需要的监视的变量，便于快速查看变量的当前状态，方便用户调试;

5.4.2 硬件

打开已经建立的工程，在左侧的“工程树”窗口，单击“硬件”选项卡，可以看到“物理硬件”，“物理硬件”能够显示软件模型的结构，用户可对各层进行查看、设置。

（1）物理硬件

“物理硬件”树能够反映符合 IEC 61131-3 的程序结构，是整个工程的全部配置文件，负责管理它的下一层——“配置”，目前，MULTIPROG Express 版本仅支持一个“配置”，不能插入多个配置，但能删除“配置”，也可从另一个工程中拷贝一个“配置”过来。

（2）配置

“配置”是软件模型中的第一层，在“物理硬件”的下一层，相当于可编程控制器系统，负责管理它的下一层“资源”，目前，MULTIPROG Express 版本仅支持一个“资源”，不能插入多个资源，但可以删除“资源”，也可从另一个工程中拷贝一个“资源”过来。可编程控制器的类型，可通过右键“配置”选择属性查看，在 PLC/处理选项卡中的 PLC 类型下拉列表中为“eCLR”，不可更改。

（3）资源

“资源”是软件模型中的第二层，在“配置”的下一层，相当于可编程控制器的处理器，负责管理它的下一层——“Tasks”，“Globales_Variables ”，“I/0_ Configuration”，这三个均不可删除。 处理器的类型，可通过右键”资源”，选择属性查看，在 PLC/处理选项卡中的处理器类型下拉列表中选择“ARM_LE GCC3”，当没有连接 PLC 只是仿真时，可选择，“eCLR Simulation”处理器的设置，可通过右键“资源”，选择“设置”查看，可以看到可编程控制器的通讯协议、IP 地址、处理器版本等。

（4）Tasks

1）“Tasks”是软件模型中的第三层，在“资源”的下一层，“Tasks”下可插入多个任务，可以是循环扫描的，也可是周期扫描的，如果插入的任务是周期扫描的，可设置不同的扫描周期和优先级，这是 MULTIPROG 的多任务特性。

2）在创建一个工程时，MULTIPROG 自动声明了一个“Tasks”和这个目录下的第一个任务，用户可以右键“Tasks”插入一个新的任务，或从另一个工程中拷贝一个任务过来，插入的任务类型默认为“DEFAULT”，即循环扫描，也可以选择“CYCLIC”，即周期扫描，选择“CYCLIC”后，需要设置时间间隔、优先权和监视定时时间。右键“任务”，在弹出的对话框中分别选择“属性”和“设置”，可查看修改任务的类型和扫描周期。

3）插入新的“任务”后，右键新插入的任务名，选择“程序实例”指定程序实例名和实例类型，实例类型是在 POU 中插入的程序（PROGRAM）。

一个工程中，只允许有一个 DEFAULT 类型的任务，其它都默认为 CYCLIC 类型任务；在一个任务下，可插入多个程序实例，多个程序实例的执行顺序按照它们出现在任务下的顺序来执行。

（5）Global_Variables——全局变量

“Global_Variables”是软件模型中的第三层，在“资源”的下一层，与“Tasks”并列，Global_Variables 不可复制、粘贴。全局变量是一个变量表，包括 MULTIPROG 提供的系统变量和用户建立的变量，用户建立的变量只有指定为 VAR_GLOBAL 时，才会出现在这个表中。

（6）IO_Configuration——IO 配置

“IO_Configuration”是软件模型中的第三层，在”资源”的下一层，与”Tasks”、”Globa。l_Variables”并列，IO_Configuration 不可复制、粘贴。双击“IO_Configuration”打开 I/O 配置对话框，它用于编辑 I/O 组态工作单，包括了 INPUT（输入）、OUTPUT（输出）、VARCONF 的属性设置，用户只需设置 INPUT、OUTPUT 即可，在 INPUT 中，有 INPUT 的名称，默认“IN”，所属的任务，逻辑起始地址，板卡/IO 模块的驱动程序参数，在驱动程序参数中，用户需要指定驱动程序名称，默认为 DUMMYIO，用户需要改为“KWIO”。

详细配置过程见“7.4 IO 配置”。

5.4.3 工程

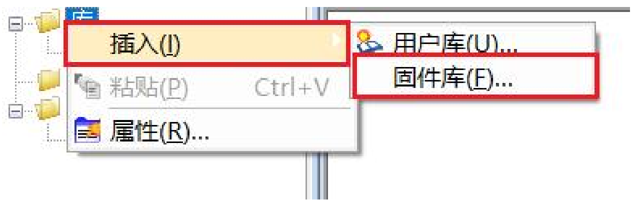
工程包含库、数据类型、逻辑 POU 三个部分，组成一个完整而又强大的程序。

（1）库

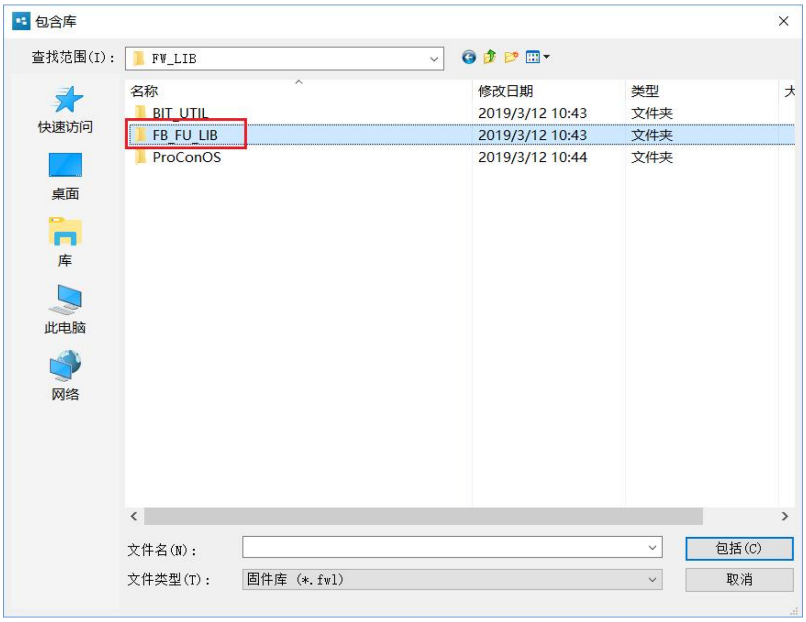
1）库提供了功能块、功能、程序和数据类型。在插入一个库之后，用户可以像使用 IEC 功能块一样使用库中所带的功能、功能块。右键“库”，可以插入“用户库”和“固件库”。这两种库不是必须的,用户要根据自己程序的需要选择插入。用户库是用户创建的其他工程，用户库的文件扩展名为*.mwto 固件库是特殊功能的功能、功能块，需要用户单独插入到工程中，固件库的文件扩展名为*.fwl，

下面举例如何插入一个固件库。

1>　在工程树窗口下右键“库”，选择“插入”选择“固件库”　如图所示



2> 在弹出来“包括库”的窗口中，打开“FB_FU_LIB”文件夹（在此操作之前，用户需要将该文件夹复制到该默认目录下），然后点击文件“FB_FU_LIB.FWL”，再点击包括，如图操作即可完成固件库插入。



（2）数据类型

1）如果用户要定义自己的数据类型（如:数组、结构，等等），则这些数据类型必须要在“数据类型”中声明。右键“数据类型”，选择插入“数据类型”，指定数据类型工作单的名称，双击生成的数据类型工作单，进入编辑区域，键入以下字符如图所示

例子：

```
TYPE
    DATA1:ARRAY [1..100] OF INT;
END_TYPE
```

以上代码定义了一个包含 100 个 INT 型变量的数组，数组名 DATA1。

（3）逻辑 POU

1）程序组织单元 POU 是 PLC 程序的语言元素。它们是包含了程序代码 最小的、独立的软件单位。POU 的名称在工程内必须是唯一的，右键“逻辑 POU”可插入以下三种程序组织单元:

A:〗程序（PROGRAM ）

B:〗功能块（FUNCTION BLOCK）

C:〗功能（FUNCTION ）

2）每个 POU 都由两个不同部分组成:变量工作单和代码本体，在变量工作单中是这个 POU 中出现的所有变量。一个 POU 的代码工作单是用户用 IL, ST、FBD, LD, SFC 五种编程语言中的一种编写的，其中 IL 是指令表编程语言（Instruction List） ST 是结构化文本编程语言（Structured Text）; FBD 是功能块图编程语言（Function Block Diagram ）, LD 是梯形图编程语言（Ladder Diagram）; SFC 是顺序功能表编程语言（Sequential Function Chart ）。

🕮 **功能**

“功能”，缩写为 FU，是带有多个输入和一个输出的程序组织单元 POU，类似于高级编程语言中的函数，“功能”的返回值可以是 BOOL, INT 等简单数据类型，一个“功能”内部可以调用另外的“功能”，但不能调用“功能块”或“程序”，不允许递归调用。声明一个“功能，时，必须在这个”功能”的变量工作单中声明输入输出变量、中间变量、外部变量。

MULTIPROG 支持的功能列表:

类型转换功能	如:IN_TO_REAL
数值功能	如:ABS 和 LOG
标准算术运算功能	如:ADD 和 MUL
位串功能	如:AND 和 SHL
选择和比较功能	如:SEL 和 GE
字符串功能	如:RIGHT 和 INSERT
时间数据类型功能	如带有 TIME 数据类型的 SUB

🕮 **功能块**

“功能块”，缩写为 FB，是带有多个输入和多个输出的程序组织单元 POU,“功能块”内可以调用另外的“功能块”或“功能”，但不能调用“程序”，不允许递归调用。所有的“功能块，”（IEC 定义的，库件库 FB 和用户定义的 FB）可以很容易地被插入到用户的“功能块”或“程序”中。声明一个“功能块”时，必须在这个“功能块”的变量工作单中声明输入输出变量、中间变量、外部变量。

MULTIPROG 支持的功能块列表:

双稳态元素，	如 SR 和 RS
边沿检测功能块	如:R_RIG 和 F_TRIG
计数器	如:CTU 和 CTD
定时器功能块	如:TON 和 TOF

📖 **程序**

“程序”是包含了功能、功能块的一个程序代码组合，“程序”的行为和用途类似于功能块，它可以具有输入和输出参数，可以具有内部存储区，但不允许递归调用。在创建一个工程时，MULTIPROG 自动声明了一个“程序”，声明一个新的“程序”时，MULTIPROG 同时生成这个“程序”的变量工作单（双击这个新声明的程序，然后单击变量工作单即可进入这个程序的变量工作单），并把它加载到 Tasks 目录下的第一个任务中，可通过剪切粘贴把这个程序放到其他任务中。如前所述，在 Tasks→任务下插入一个程序实例时，要输入程序名和程序实例类型，这个程序实例类型是在程序→逻辑 POU 下插入的“程序”，这样，可在多个任务中插入多个程序实例，这些程序实例的名字可以不相同，但可以是同一个程序实例类型，既是说，一个程序实例类型可以在多个任务中被执行。“程序”一定要被关联到任务上。

第六章 MULTIPROG 编程语言

MULTIPROG 支持 IL, ST、FBD, LD, SFC 五种编程语言，其中 IL 和 ST 属于文本类编程语言，FBD, LD 和 SFC 属于图形类编程语言。一个具有独立功能的程序分为代码部分和数据部分，代码要使用 IL, ST、FBD, LD, SFC 中的一种或几种语言的组合来编写，数据要在变量工作单中声明。本章介绍如何声明变量以及如何使用这五种编程语言编程。

〖1〗IL 是 Instruction List 指令表的缩写；
〖2〗ST 是 Structured Text 结构化文本的缩写；
〖3〗FBD 是 Function Block Diagram 功能图表的缩写；
〖4〗LD 是 Ladder Diagram 梯形图的缩写；
〖5〗SFC 是 SequentialFunction Chart 是顺序功能图的缩写；

图形类编程语言中程序是从上到下、从左到右扫描的，文本类编程语言中程序是从上到下扫描的。

6.1 变量工作单

IL, ST、FBD, LD, SFC 五种编程语言需要在各程序组织单元 POU 所对应的变量工作单中声明变量，用户选中工程树中的 POU 目录下的程序、功能或功能块，单击菜单栏上的变量工作单，进入变量工作单，如下选中第一行，选择追加变量，MULTIPROG 自动插入一个缺省变量，（也可创建变量集，名称下填字母加#；如 m# ；填入起始和终止的地址和选择数据类型，单击确定即可创建多个变量）如图所示

	名称	类型	用法	描述	地址	初值	保持	PDD	OPC	隐藏	初值作为隐藏值	默认的隐藏值
	⊟Default											
	NewVar1	BOOL	VAR				☐	☐	☐	☐	☐	

其中名称栏是变量的名称，默认 NewVarl，用户可修改变量名，变量名必须以字母开头，可包含字母、数字和下划线;类型栏是数据类型，用户可直接键入用户名，或通过下拉菜单选择;用法栏，表示插入的变量的作用域:

1➢对于插入的程序，只有 VAR 和 VAR EXTERNAL 两个选项，VAR 表示内部变量，VAR_EXTERNAL 表示外部变量;

2➢对于插入的功能，只有 VA R 和 VAR_INPUT 两个选项，VAR 表示内部变量，VAR_INPUT 表示输入变量;

3➢对于插入的功能块，有 VAR, VAR_EXTERNAL, VARIN_OUT, VAR_ INPUT 和 VAR _OUTPUT, VAR 表示内部变量，VAR_EXTERNAL 表示外部变量，VAR_IN OUT 表示输入输出变量；

⅄描述栏,:是用户输入的文字描述;地址栏是该变量的输入、输出、中间变量;

⅄地址栏:标明该变量的地址;

⮙初值栏:在 PLC 程序中，第一次使用这个变量时，会使用在这里标明的初始值;

⮙保持栏:在 PLC 断电的情况下，该变量的值仍被保存，暖启动之后，将使用变量的最后值;

⮙PDD 栏:指明变量已写入过程数据目录（PDD）中，仅当用户访问 PLC 上的某个地址对应的变量名称时，才勾选上它。

⮙OPC 栏:指明变量已写入 OPC 服务器文件中，仅当用户希望通过 OPC 客户端访问该变量时才勾选上它。

FBD, LD 和 SFC 编程语言编程时，也可在编辑区插入变量，插入后的变量自动列入变量工作单中。VAR_INPUT 表示输入变量，VAR_OUTPUT 表示输出变量。

6.2 IL 指令表编程语言

指令表编程语言的基本语句是指令表，它是一种底层语言，采用面向机器的操作符，比较容易转换为可编程控制器的机器代码，因为缺乏有效的工具，指令表编程语言比较适合小型的控制程序，不适合大型复杂的控制任务。

6.2.1 创建一个 IL 程序

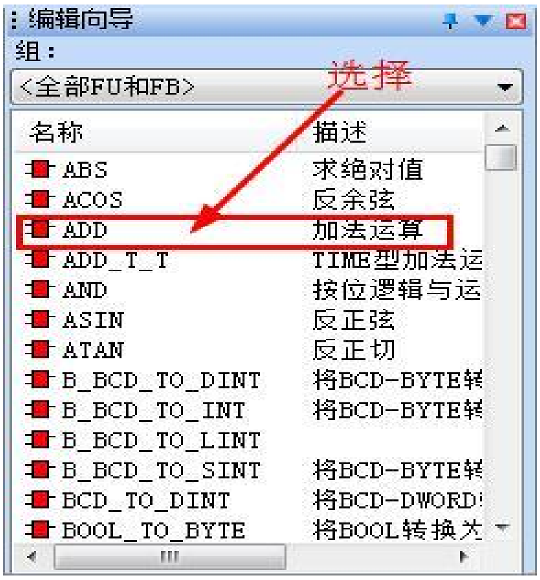
使用 IL 编程，用户可在编辑区键入代码，对于指令可直接键入或在“编辑向导”中拖拽指到编辑区。下面以写一个 A+B=OUT（*加法运算*）的程序为例说明 IL 编程工程的创建:

1> 创建一个工程

2> 右键工程树中的“逻辑 POU”选择插入“程序”弹出对话框，在对话框中输入“IL_Test”，在语言栏选择 IL，单击确定 如图所示。

3> 完成上步骤后，在工程树中出现了一个程序 POU 一 IL _Test，如图所示。

4> 双击“逻辑 POU”下的 IL _Test，然后在编辑向导区选择“全部 FU 和 FB”出现全部功能和功能块，如图所示。



5> 双击“ADD 模块”，或拖拽“ADD 模块”到编辑区，“ADD 模块”的功能即可出现在编辑区，如图所示。

6> 将图中的绿色字体部分分别改为 A,B 和 OUT（用户可自己选择变量名，不一定是 IN 和 OUT），如图所示。

```
LD    A
ADD   B
ST    OUT
```

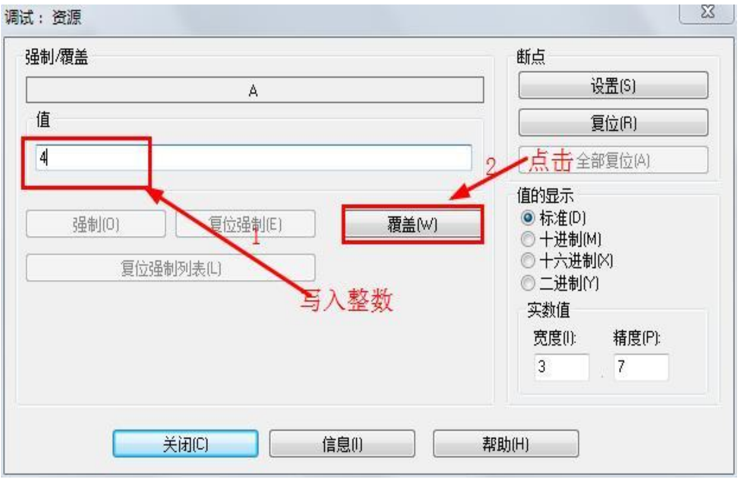
7> 单击菜单栏上的“变量工作单”，声明三个变量 A,B 和 OUT，类型为 INT，用法为 VAR（局部变量）完成后,返回 IL 的编程界面如图所示。

名称	类型	用法	描述
⊟ Default			
A	INT	VAR	
B	INT	VAR	
OUT	INT	VAR	

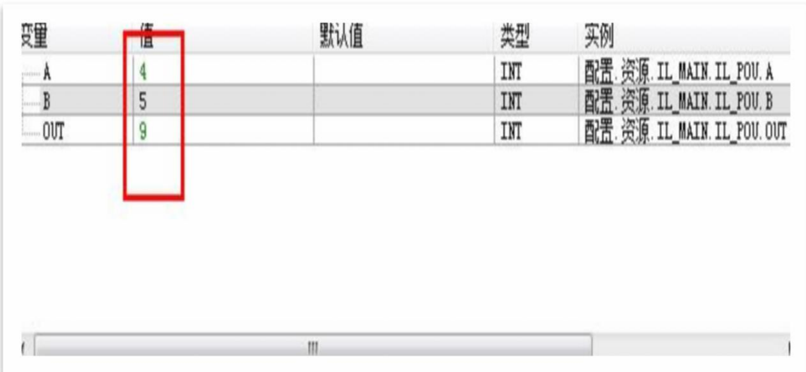
8> 单击“制作”无错误提示后，单击“下装”，然后单击“调试开关”，如图所示。

9> 单击变量工作表，用鼠标选中三个操作数，右健选择“添加到监视窗口”，用户可以在线调试和监视变量 如图所示。

10> 分别双击监视窗口中的 A 和 B 变量，分别赋值 4 和 5，然后单击覆盖，如图所示



11 > 在编辑区左边视窗口均可显示变量的状态值，如图所示。



6.2.2 IL 的语句

在指令表语言 IL 的编辑区，可逐行写入 IL 编程语言的语句，每条语句都占据一行，每行语句的前面（非编辑区）是一个行号，每行指令后不带分号。

IL 的语句格式如下：

行号	操作符号/指令	操作数	（*　*）
1 . . N.	IL 操作符或指令	操作符可带有 0,1 或者多个常数变量，或指令参数表	注释

IL 语句中以不同颜色来区分各部分:操作符/功能为蓝色，操作数为黑色，注释为绿色。

指令举例:以 ADD 指令为例进一步说明

求 IN 与 IN2 的和代码，如下：

LD　　IN（*IN1 作为 ANY_ NUM*）

ADD　IN2（*IN2 作为 ANY_NUM*）

ST　　OUT（*结果作为 ANY_ NUM*）

LD 是个操作符，LD 后面的变量 IN 是一个操作数，LD 的作用是把其后的操作数 IN 装载到累加器;ADD 是加法指令，加完值后再将结果装载到累加器;ST 后面的变量 OUT 也是一个操作数，作用是把累加器中的数据赋给其后的操作数;绿色字体部分是注释，提示用户声明一个变量作为操作数，数据类型是 ANY_NUM。 表示用户声明一个变量作为操作数，数据类型是 ANY_NUM。

6.2.3 IL 的操作符

IL 编程语言除了指令集中的指令可用以外，IEC 61131-3 标准将以下 24 种指令定为标准指令。

操作符	修饰符	操作数	说明
LD	N	ANY	将其后的操作数装载到累加器
ST	N	ANY	将累加器中的数据赋给其后的操作数
S		BOOL	将操作数置 1
R		BOOL	将操作数复 0
AND	N,（	ANY_BIT	逻辑与
&	N,（	ANY_BIT	逻辑与
OR	N,（	ANY_BIT	逻辑或
XOR	N,（	ANY_BIT	逻辑异或
NOT	（	ANY_BIT	逻辑取反
ADD	（	ANY_NUM	加
SUB	（	ANY_NUM	减
MUL	（	ANY_NUM	乘
DIV	（	ANY_NUM	除
MOD	（	ANY_INT	取模
GT	（	ANY_NUM, ANY_BIT	比较，大于，>
GE	（	ANY_NUM, ANY_BIT	比较，大于等于，>=
EQ	（	ANY_NUM, ANY_BIT	比较，等于，=
NE	（	ANY_NUM, ANY_BIT	比较，不等于，<>
LE	（	ANY_NUM, ANY_BIT	比较，小于等于<=
LT	（	ANY_NUM, ANY_BIT	比较，小于<
JMP	C,N	LABAL	跳转到标号处的指令

CAL	C,N	NAME	调用功能块
RET	C,N		从被调用的函数、功能块处返回
）			延迟结束

注:

1> 修饰符 N 表示取反操作，如 ANDN 表示将操作数取反后与;

2> 修饰符 C 表示只有在当前运算结果为真时才执行，如:

```
LD        IN1
AND      IN2
ST    OUT1
JMPC    M2         //  当 M2 为真的时候执行跳转
M2:
LD    IN3
ST OUT2
```

6.3 ST 结构化文本编程语言

结构化文本编程语言 ST 是高层语言，类似于 Pascal 编程语言，它不采用低层的面向机器的操作符，而是使用类似于日常语言的语句来描述控制命令，能够描述复杂的算法。结构化文本语言的程序由语句组成，语句由表达式和关键字组成，具有以下特点:

- ◆ 没有跳转语句，用条件语句实现程序的分支;
- ◆ 每条语句用分号“;”结束;
- ◆ 用（**）为程序添加注释，注释不能嵌套，如（*（**）*）;
- ◆ 需要在 POU 对应的变量工作单中申明输入、输出、内部、外部、全局变量;

6.3.1 创建一个 ST 程序

使用 ST 语言编程，用户可在编辑区键入一条语句，也可直接在“辑向导”中拖拽或双击指令到编辑区，再键入操作数和结束符“;”下面以一个 A*B=OUT（乘法运算）程序为例说明 ST 编程过程:

1> 创建一个工程

2> 右键工程树中的“辑 POU”，选择插入一程序，弹出对话框，在对话框中键入程序名，本例中为 ST_Text，在语言栏选择 ST，单击“ 确定 ”如图所示

3> 在工程树中出现了一个新的程序 POU，如图所示。

4> 双击“逻辑 POU”下的“ST_Text”，然后在编辑向导区选择“全部 FU 和 FB”，出现全部功能和功能块，如图所示

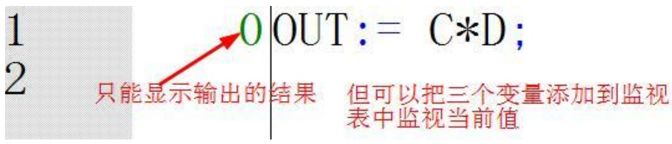
5> 双击“MUL”，或拖拽 MUL 到编辑区，MUL 功能就出现在编辑区了，如图所示

```
1  (* Result as ANY_NUM *) := (* IN1 as ANY_NUM *) * (* IN2 as ANY_NUM *);
2
```

6> 将图中的绿色字体部分分别改为 OUT 和 C 和 D；（用户可自己选择变量名，不一定是 IN 和 OUT），如图所示。

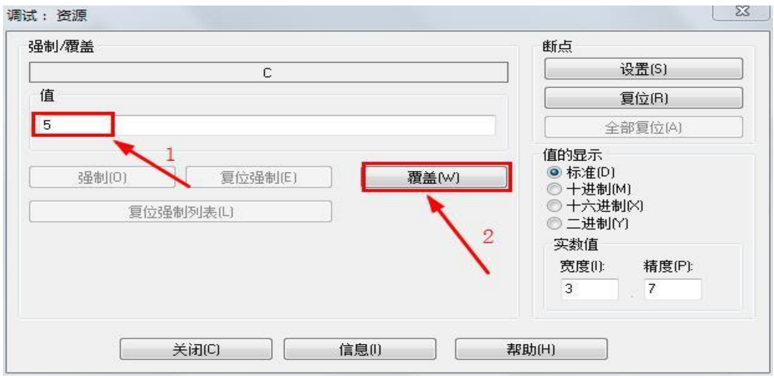
名称	类型	用法	描述
⊟ **Default**			
C	INT	VAR	
D	INT	VAR	
OUT	INT	VAR	

7> 单击“制作”，并“下装”，然后单击“调试开关”，在编辑的界面左边可以看到当前的结果，如图所示



8> 用鼠标分别选中三个操作数，右键选择添加到监视窗口，监视窗口如图所示

9> 分别双击监视窗口中的 C 和 D 变量，分别写入 5 和 3，然后单击“覆盖”，如图所示



10> 在编辑区和监视窗口均可显示变量的状态值，如图所示

至此，一个完整的简单的 ST 程序完成了。

6.3.2 ST 的语句

在 ST 编程语言中，程序由语句组成，语句由表达式和关键字组成。在结构化文本编程语言 ST 的编辑区，可逐行写入 ST 编程语言的语句，每条语句都以分号结束，多条语句可占

据一行，每行语句的前面（非编辑区）是一个行号。 如一条赋值语句，它由变量、赋值关

说明	关键字	举例	描述
赋值操作符	:=	OUT:=IN	将 IN 赋值给 OUT
返回	RETURN	RETURN;	退出被调用的功能、功能块或程序，返回到调用它的语句处。
选择	IF	IF a<b THEN c:=1; ELSIF a=b THEN c:=2; ELSE c:=3; END_IF:	当 IF 后的表达式‘a<b’为 TRUE 时，执行 THEN 后的一条语句（以分号为界），否则不执行 THEN 后的语句，而继续判断 ELSIF 或 ELSE 。
	CASE	CASE f OF	根据 CASE 关键字后面的表达式的值，执

键字和表达式组成，作用是将表达式的运算结果赋给

赋值关键字左侧的变量:
变量名:=表达式;
赋值关键字两侧的数据类型必须一致。

6.3.3 ST 的表达式

表达式由操作数和运算符组成，操作数可以是直接量、变量或功能名称。可以使用的运算符如下表:
可以使用的运算符如下表

操作符	举例	例子的值	描述	优先级
（ ）	（2+3）*（4+5）	45	括号	最高
**	3.0**4	81.0	乘幂运算	
-	-10	-10	求相反数	
NOT	NOT TRUE	false	按位取反	
*	10*3	30	乘法运算	
/	6/2	3	除法运算	
MOD	17 MOD 10	7	模数运算	
+	2+3	5	加法运算	
-	4-2	2	减法运算	
<，>，<=，>=	4>12	false	比较	
=	T#26h 二 T#1 d2h	true	相等	
<>	8 <>16	true	不相等	
&，AND	TRUE&FALSE	false	布尔与	
XOR	TRUE XOR FALSE	true	布尔异或	
OR	TRUE OR FALSE	true	布尔或	最低

		1:a:=3;
		2: a:=4;
		3: a:=2;
		ELSE
		a:=0;
		END_CASE;
			行一组语句。变量或表达式‘f’必须为 INT 数据类型。

循环	FOR	FOR a:=1 TO 10 BY 3 DO
		f[a] :=b;
		END_FOR;
			变量‘a’从 1 开始，FOR 和 END_FOR 之间的语句被反复执行，每执行一次，a 增加 3，到 10 结束。所有值必须具有 ANY_INT 数据类型。

	WHILE	WHILE b>1 DO
		b:=b/2;
		END_WHILE;
			当表达式`b>1‘的值为 TRUE 时，WHILE 和 END_WHILE 之间的语句被反复执行，直到‘b>1’的值为 FALSE。

	REPEAT	REPEAT
		a:=a*b;
		UNTIL a<10000
		END_REPEAT;
			REPEAT 和 END_REPEAT 之间的语句被反复执行，直到所表达式‘a<10000’的值为 TRUE。

循环结束	EXIT	FOR a:=1 TO 2 DO
		IF flag THEN EXIT;
		END IF
		SUM:= SUM+a
		END_FOR
			退出语句可被用于中止循环语句的执行。

语句结束	;		放在语句后面，表示语句的结束，也可单独存在。

ST 语言常用关键字

6.4 FBD 功能块图编程语言

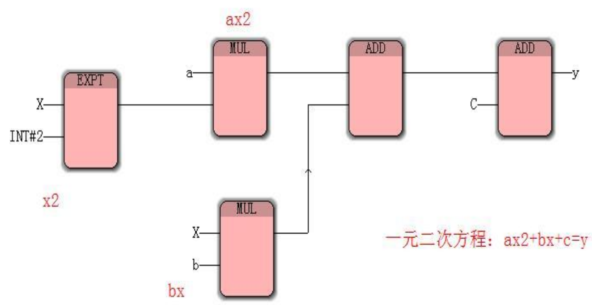
功能块图编程语言源于信号处理领域，它是 IEC 61499 标准的基础，一个功能块图编程语言的程序是将各种功能块连接起来，编程语言的元素是函数、功能块和连接符号。

6.4.1 创建一个 FBD 程序

1）创建一个新工程后，右键工程树中的“逻辑 POU”，选择插入→程序，在弹出的对话框中键入程序名 FBD_Test、选择类型和编程语言，如图所示

2）工程树中创建的 FBD_Test，在编辑向导中分别找到 ADD, MUL 和 EXPT 功能块，将他们拖拽到 FBD_TesT 的编辑区，我们要用它们完成一个一元二次函数的计算功能；

$y=ax^2+bx+c$，建立的 FBD　程序　如图所示



3）功能块 EXPT 完成 x 的平方运算，功能块 MUL 完成 a 与 X^2 的乘法，另一个 MUL 完成 b 与 x 的乘法，功能块 ADD 完成 ax^2 与 bx 与 c 的加法，单击“制作”，然后“下装”，再单击“调试开关”，将变量 a,b,c,x,y 添加到监视窗口，将 a,b,c,x 变量分别赋值为，2.0,5.0,6.0,3.0 则程序自动计算出 $ax^2+bx+c=y$ 的结果，如图所示

在 FBD 中，是不需要额外在变量工作单中建立变量的，所以，在插入变量并双击变量后，这些变量就出现在变量工作单中了。FBD 功能块图编程语言与 LD 梯形图编程语言相似。
至此，完成了一个 FBD 语言编程过程 。

6.4.2 EN 和 ENO 说明

1：逻辑指令带 EN 和 ENO 与不带 EN 和 ENO 的区别；

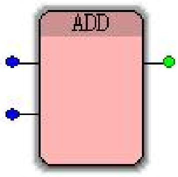
（1）当用户使用的指令带有 EN 和 ENO 时，若 EN 引脚的参数值为 FALSE（0），那么指令定义的功能将不会被执行，指令的输出引脚输出值不会被刷新。相反，若指令定义的 EN 引脚参数值为 TRUE（1），那么指令定义的功能将会被执行，指令输出引脚的值也会被刷新。

（2）ENO 引脚输出和 EN 引脚的输入保持一致的，EN 引脚为 TRUE，ENO 引脚同时变为 TRUE； EN 引脚为 FALSE，ENO 引脚同时变为 FALSE。当指令为功能块（FB）时，如果功能块（FB）执行后，EN 由 TRUE 变为 FALSE，该功能块（FB）仍继续执行，只是该功能块（FB）输出引脚的值不会被刷新。

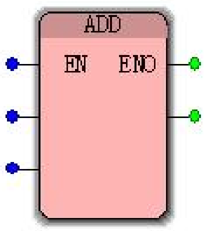
2：逻辑指令如何设置带 EN 和 ENO 与不带 EN 和 ENO

（1）鼠标放到编程界面下，此时菜单栏将会自动增加“对象”菜单，单击->“对象”-> 选中”插入带有 EN/ENO 的块”，以后从“编辑向导”中拖拽出来的功能模块将自 EN/ENO ；当不使用该功能时，重复操作上述操作即可。如下图所示

（2） 已经调用没有带 EN/ENO 的功能块，如果想要增加 EN/ENO 功能，鼠标选中编程界面，此时菜单栏将会自动增加“对象”菜单，单击-> “对象”-> 选中”插入带有 EN/ENO 的块”，然后选中需要添加 EN/ENO 的功能块，右键-> 选择->更新 FB/FU（E），此时功能块将自动添加带 EN/ENO 引脚。如下图所示



执行前



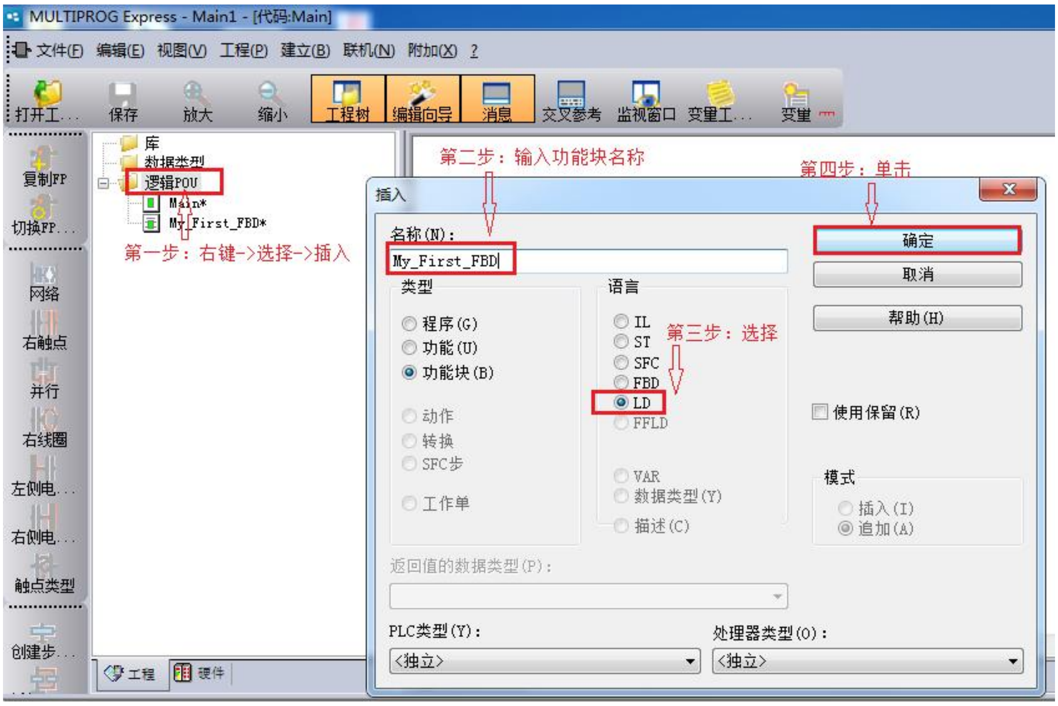
执行后

6.4.3 创建用户库

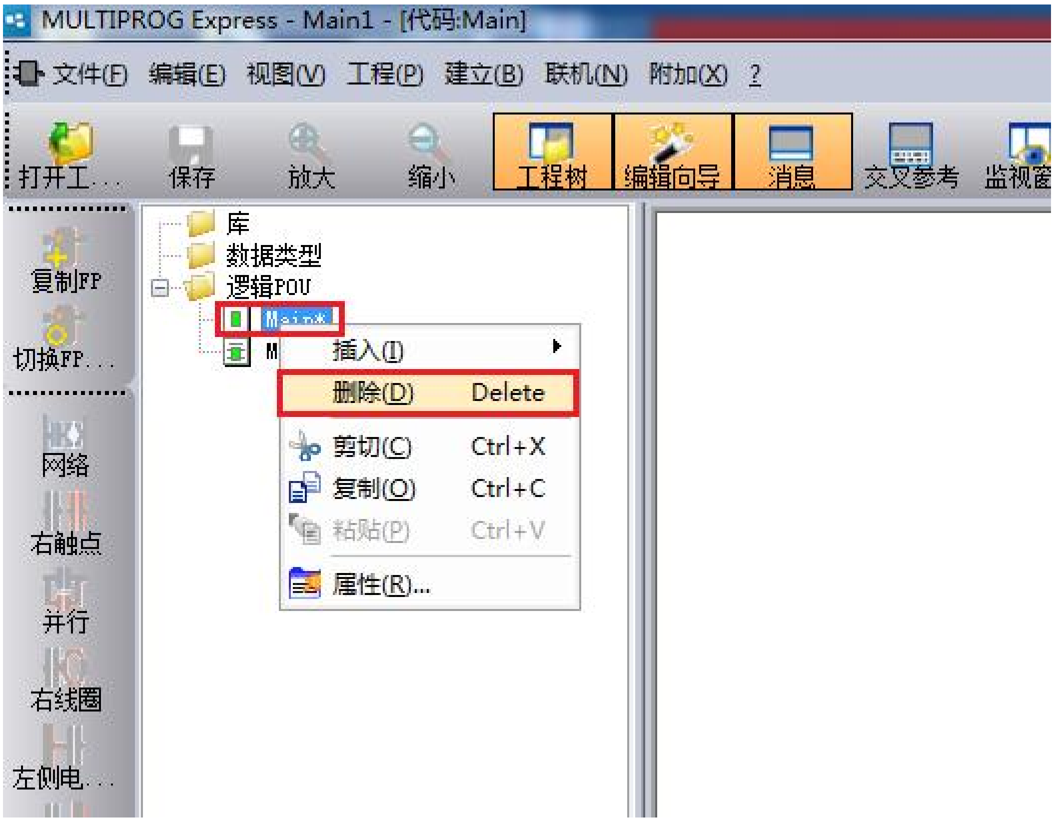
用户库功能：一般建立用户库是将特定的功能程序段将其封装成功能块，在其他工程下也需要相同特定功能，用户直接调用已封装的功能块即可，无需重新再编写一遍，可缩短程序开发周期，也方便程序模块化的管理。

创建用户库和调用步骤

1. 创建工程项目　用户参考（第三章 3.3 创建工程项目） 这里将不再重复说明。
2. 在工程树下选中->逻辑 POU->右键->插入->功能块（E），此时弹出一个“插入”对话框，为即将插入的功能块命名。（本案例命名为 My_First_FBD），选择编程语言（本案例选择为 LD 语言）单击->确定， 逻辑 PUO 文件下将创建一个名为 My_First_FBD 的功能块，如下图所示



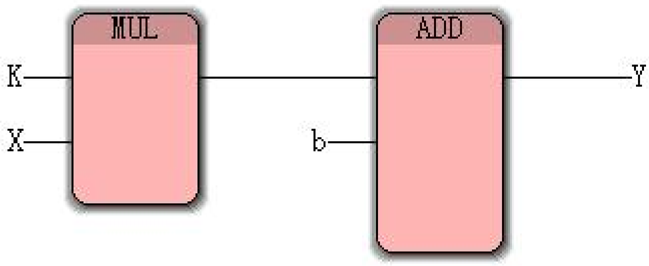
3. 删除 POU 逻辑下程序块，仅保留 功能块名为 My_First_FBD 选中要删除的程序块->右键->删除，如下图所示



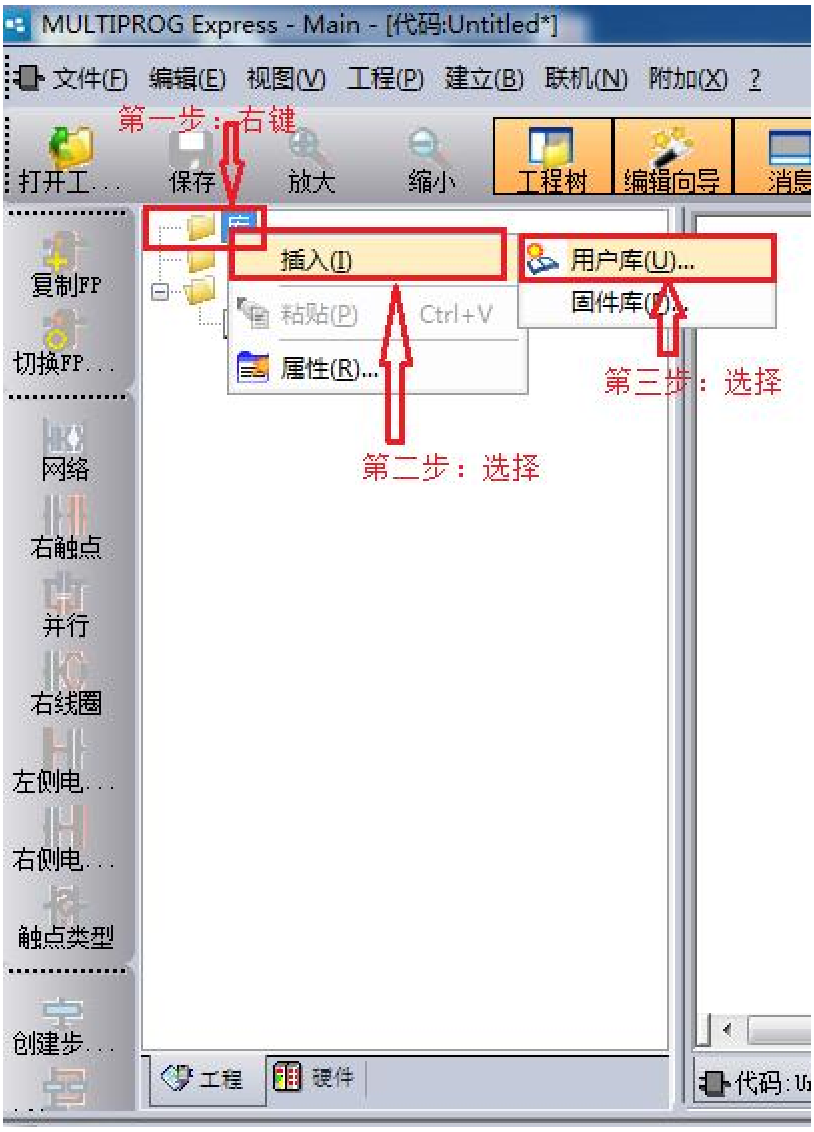
4. 删除硬件下的配置信息，单击->硬件->选中配置->右键->选中->删除 只保留 物理硬件如下图所示

5. 完成后工程树下的“工程”和“硬件”如下图所示

6．完成上述操作后，开始进行功能块的编程和封装
例：封装一个一元一次函数指令；
调用一个乘法指令，一个加法指令，然后设置四个变量名称分别为 K,X,b,Y，数据类型选择为 INT，（本例选择使用 INT 类型）其中 K,X,b 用法中三个变量选择类型为 VAR_INPUT， Y 输出结果的变量选择为 VAR_OUTPUT。完成后单击“制作”没有错误警告提示后。单击保存，然后关闭工程。至此编程和封装全部完成。 如下图 （特别提醒：封装的功能块变量尽可能地使用内部变量，可提高封装块使用范围）

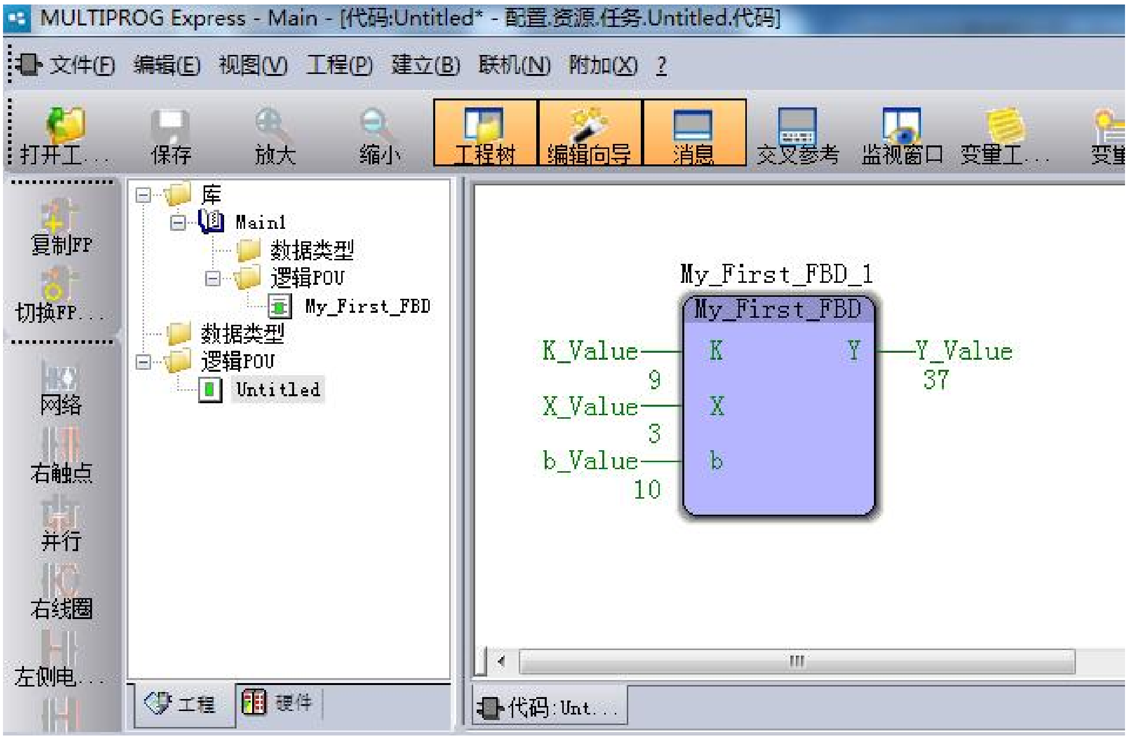


7.打开需要调用该功能的项目工程，然后选中“库”右键->插入->用户库-> 找到存储上述功能块封装的工程路径（.mwe 文件，文件的名称不允许跟现工程的名称相同）->单击->抱括，完成后，库中将添加了上述封装的功能块如下图所示


MULTIPROG Express - Main - [代码:Untitled*]
文件(F) 编辑(E) 视图(V) 工程(P) 建立(B) 联机(N) 附加(X) ?
第一步：右键
打开工... 保存 放大 缩小 工程树 编辑向导
插入(I)
粘贴(P) Ctrl+V
属性(R)...
用户库(U)...
固件库(F)...
第二步：选择
第三步：选择
复制FP
切换FP...
网络
右触点
并行
右线圈
左侧电...
右侧电...
触点类型
创建步...
工程
硬件


库
Main1
数据类型
逻辑POU
My_First_FBD
数据类型
逻辑POU
Untitled*
工程
硬件

8： 在工程项目主程序下，将 My_First_FBD 功能块拖拽到主程序中 然后为其设置变量名称，数据类型为 INT 型，用法为内部变量（VAR）。完成后单击“制作”没有编译错误提示后，单击“下装”，然后单击“冷启”，选择打开“调试开关”，在线赋值变量 K_Value=9，X_Value=3，b_Value=10 最终输出结果 Y_Value=27 。如下图所示（从颜色可区分用户库和固件库，粉色为固件库，蓝色为用户库，绿色为在本工程项目下封装的功能块）



至此一个用户库封装和调用已经完成。

特别说明：

（1）被调用的用户库 “双击”功能块可进行查看内部编程内容，但是不允许修改或者添加新的功能。

6.5 LD 梯形图编程语言

梯形图编程语言是历史最久远的一种编程语言，梯形图源于电气系统的逻辑控制图，逻辑图采用继电器、触点、线圈和逻辑关系图等来表示它们之间逻辑关系，梯形图编程语言可采用的图形元素有梯形图网络、电源轨线、连接导线、触点、线圈,功能和功能块等，数据类型可以是 BOOL, BYTE, WORD 和 DWORD。

6.5.1 创建一个 LD 程序

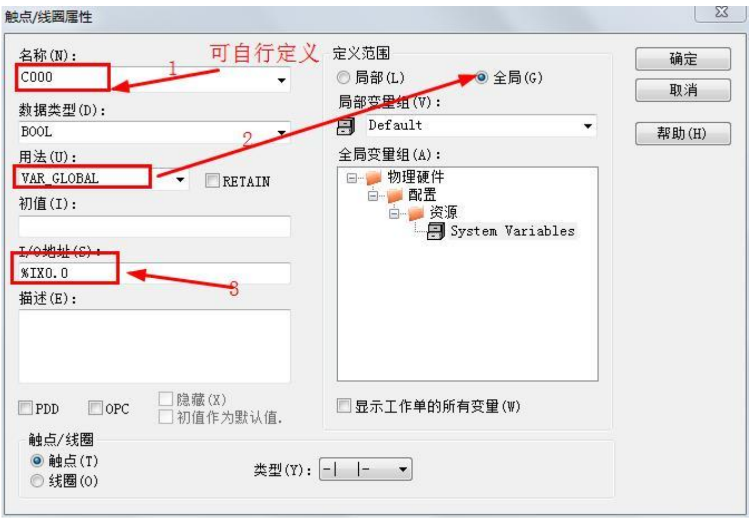
1> 创建一个工程后，右键工程树中的“逻辑 POU”，选择插入→程序，在弹出的对话框中键入程序名 LD_Test、选择类型和编程语言，如图 8-25 所示

图 8-25 LD 工程

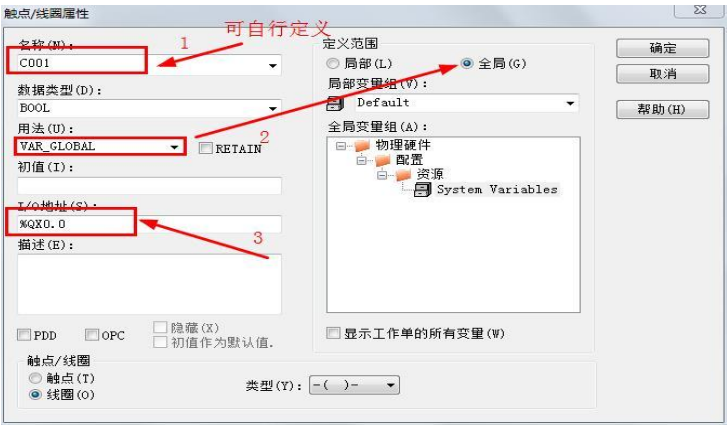
2> 单击 MULTIPROG 编程软件的编辑区域，然后单击编辑区域左侧的工具栏上的“  ”

3> 在编辑区域中出现了一个梯形图的简单网络，左侧是左侧电源轨线 001 和一个常开触点 C000，右侧是一个线圈 C001 和右侧电源轨线 如图所示

4> 双击常开触点 C000，出现触点/线圈属性对话框，在 I/O 地址栏，输入%IX0.0，表示 PLC 本机的第一个数字量输入通道，单击“确定”，如图所示



5> 双击梯形图中线圈 C001，出现触点/线圈属性对话框，在 I/O 地址栏，输入%QX0.0，表示 PLC 本机的第一个数字量输出通道，单击“确定”，如图所示。

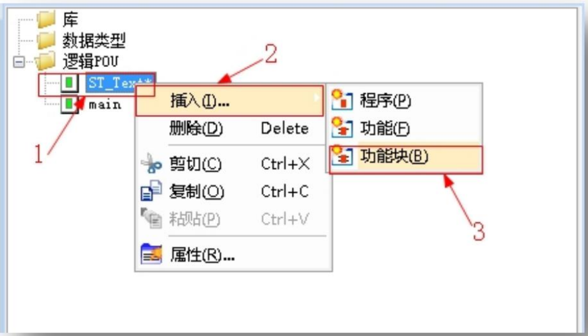


6> 在 LD 中，是不需要额外在变量工作单中建立变量的，所以，在插入一个梯形图网络，并分别双击触点和线圈后，这两个变量就出现在变量工作单中，也可以在编辑区右键鼠标，选择”变量（V）”插入一个变量，这个插入的变量必须要连接到功能块引脚端。 在上图中，C000 和 C001 的变量工作范围（用法）设为 VAR GLOBAL，说明这两个变量是全局变量，可在本工程中的其他程序里使用。

至此，一个完整的 LD 程序就完成了，当连接到第一个数字量输入通道%IX0.0 的触点闭合时，与第一个数字量输出通道%QX0.0 连接的线圈导通。

6.5.2 在 LD 中创建一个 FB

➢ 使用 LD 编程时，有时会需要一个特殊的功能块，而编辑向导中没有集成，此时用户可以在 LD 中创建用户自己的功能块，下面说明如何在 LD 中创建一个 MOVE 功能块。在工程树中右键用创建的 LD 程序名，选择插入→功能块，如图所示

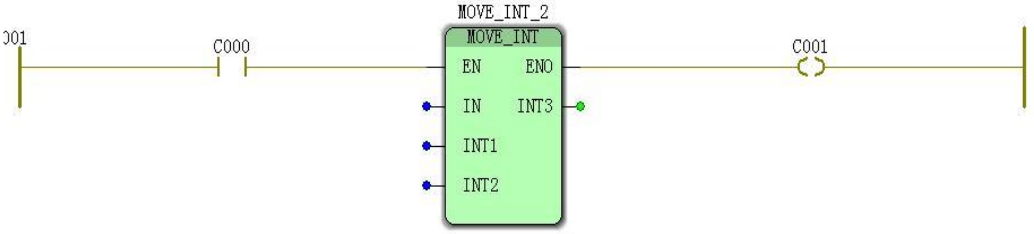


➢ 弹出一个对话框，在这个对话框中键入要创建的 FB 的名称、制定类型和所用编程语言，这里分别是 MOVE _INT、功能块和 ST，单击确定后，就进入 ST 语言编程了 如图所示。

➢ 创建的 MOVE 功能块，有输入引脚 EN, IN,INT1,INT2 有输出引脚 ENO,和 INT3，这个功能块要在前面所创建的 LD 程序中使用。 与使用 ST 创建程序时一样，首先要在变量工作单中建立 MOVE_INT 功能块所用到的变量包括变量的名称，数据类型，引脚功能等，完成后返回 ST 的编程界面进行如下编程 如图所示

➢ 完成后，单击“制作”编译通过后，回到 LD 编辑区，在编辑向导中将创建的 MOVE_INT 功能块拖拽到 LD 的编辑区，如图所示

➢ 把这个 MOVE_INT 功能块进行的下图的连接 如图所示



📖 例子：控制一个设备的起停，它有两种工作模式，触点“C000”控制启动停，触点“C002”控制运行方式（如不同的速度运行）

设备:

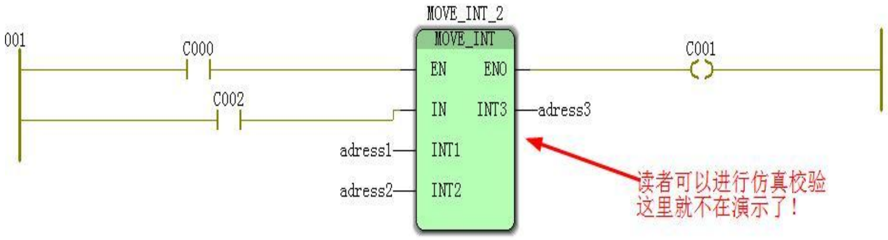
C000 为“TRUE”设备启动，“FALSE”为停止

C002 为“FALSE”运行方式 选择 “adress2”

C002 为“TRUE”运行方式选择 “adress1 ”

adress3 用于监视当前运行方式;

最终的梯形图如图所示



6.6 SFC 顺序功能图编程语言

SFC 顺序功能图编程语言可将一个复杂的控制程序分成若干个小的任务，每个小任务以顺序执行。 SFC 顺序功能图编程语言中将每个小任务称为一个“步”，“步”与“步”之间的承接关系称为“转换”，每个“步”带有一个动作，“步”、“转换”和“动作”之间用“连线”连接。 一个“步”可以关联多个动作。动作由一个动作本体和动作限定符组成，动作限定符说明动作是如何与步关联的，当 SFC 步变为活动状态时，所关联的动作会根据动作限定符而被执行。动作既可以连接一个布尔变量，也可以是一个 IL, ST、LD, FBD 程序（被称为‘明细’）。转换代表了通过下一步继续处理的情况。如果一个转换变为 TRUE，则前一个步被再执行一次，并且后一个的步成为活动状态。转换既可以是一个布尔变量，也可以是使用 FBD 或 LD 编写的直接连接的布尔表达式。也可以在另一个被称为明细的程序中，编辑将要执行的代码。

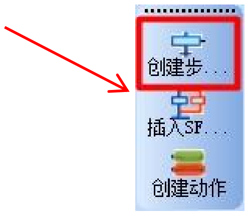
所连接对象的集合被称为 SFC 网络。一个 SFC 网络必须具有一个初始步，当调用该 SFC POU 时，这个初始步是第一个要被执行的步。可以在 SFC 网络内插入并行分支（同步地执行）或选择分支。

6.6.1 创建一个 SFC 程序

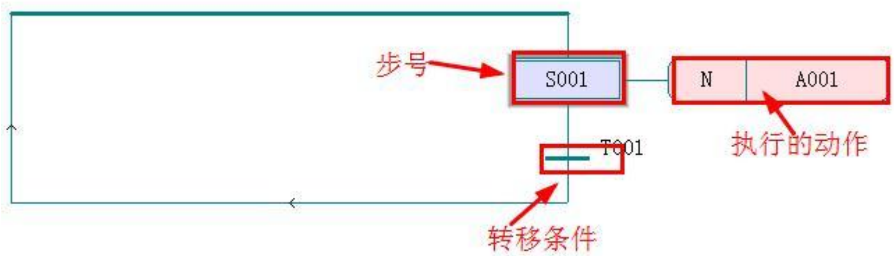
下面我们创建一个交通灯的控制程序。

6.6.2 创建 SFC 网络

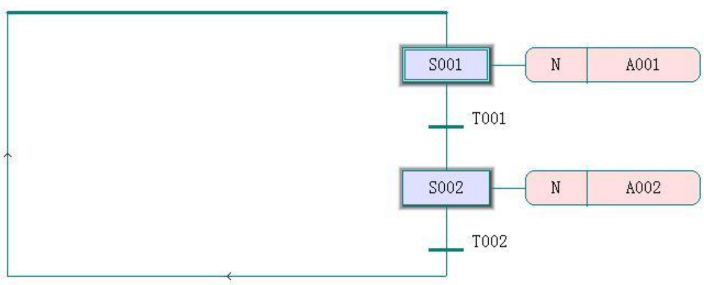
（1）创建一个新的工程，在创建过程中，选择程序语言是 SFC，程序名是 TrafficLight。进入 TrafficLight 的编辑区，然后单击编辑区左侧的“创建步转换序列” 如图所示



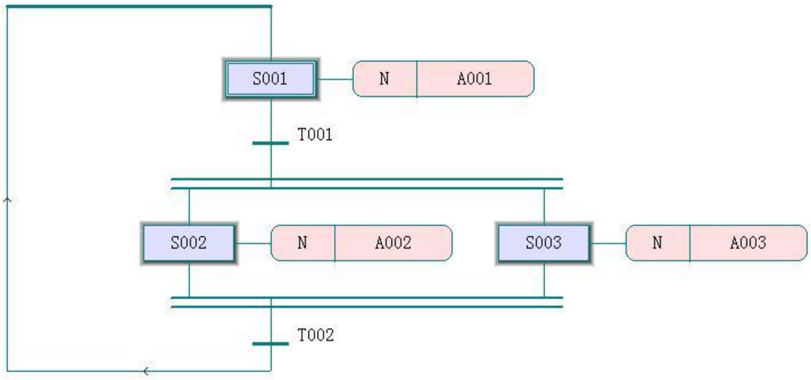
（2）在编辑区出现一个 SFC 的“步”S001 ,“动作” A001,“转换”T001 如图所示



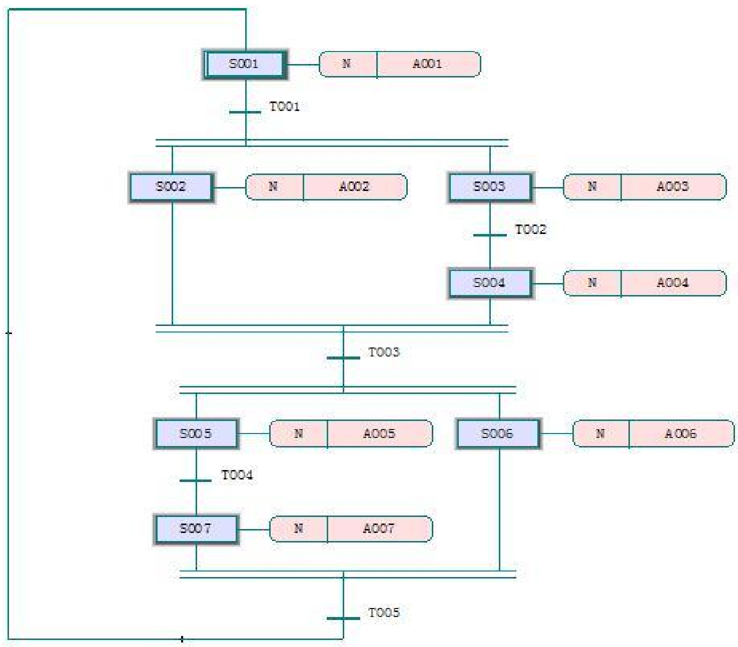
（3）上图中，每个“步”、“转换”和“动作”都有唯一的名称，接着，单击上图中的“转换”T001，再单击“创建步转换序列”，则在这个步的下面增加了一个“步”S002，如图所示



（4）单击“步”S002，再单击“插入 SFC 分支”，则在“步”S002 的右侧插入了一个分支，分支带有一个“步”S002，如图所示

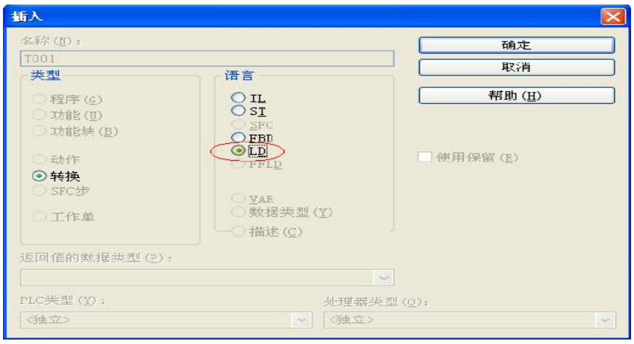


✧ 重复上描述的操作最终的交通灯 SFC 功能图如图所示



至此，一个 SFC 网络创建完毕了，其中，S001 是起始步 ，用户可以用初始化数据，如计数器清零赋初值等操作，接下来根据要求进行对“步”，“转换”进行编写代码以及属性的设置。

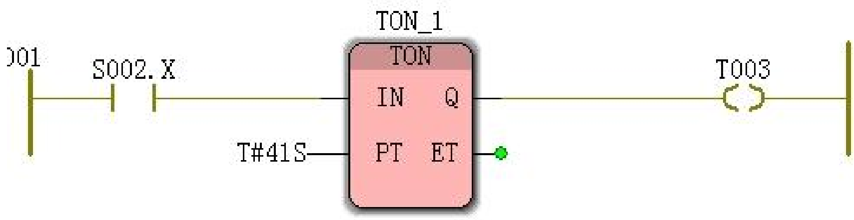
1）转换条件:双击“转换”T001，在弹出的对话框中选择 LD 编程语言，如图



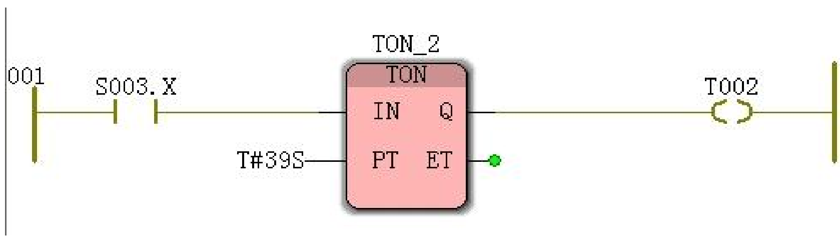
2）单击确定，进入 T001 的编程，在编辑区插入一个网络，并双击 C001 的变量名，线圈的变量名改为 T001，如下图

这个转换条件是启动 SFC 程序，因为“步”S001 是起始步，始终为 1，当全局变量 C000 置 1 时，“转换”T001 为 1，激活 “步”S002 和 S003。

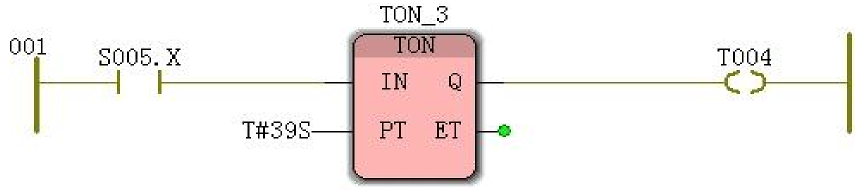
3）双击“转换”T003，选择 LD 编程语言，这个转换条件用于计时，当“步”S002 激活 41 秒后，“转换”T003 置 1 激活“步”S005 和 S006 插入如下程序



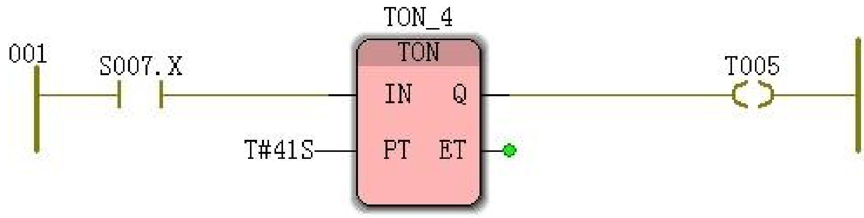
4）双击“转换”T002，当选择 LD 编程语言，这个转换条件用于计时，“步”S003 被激活后开始计时，当计时到达 39 秒后“转换”T002 置 1，激活“步”S004 插入如下程序



5）双击“转换”T004，选择 LD 编程语言，插入如下程序，这个转换条件用于计时，当“步”S005 激活 39 秒后，“转换”T004 置 1，进入“步”S007



6）双击“转换”T005，选择 LD 编程语言，插入如下程序，这个转换条件用于计时，当“步”S006 激活 41 秒后，“转换”T005 置 1，重新跳转到“步“S001，如此反复。



至此，SFC 的转换条件编程完毕。

动作

将“动作”A001 名称改为初始化；

将“动作”A002 名称改为南北_红灯；

将“动作”A003 名称改为东西_绿灯；

将“动作”A004 名称改为东西_黄灯；

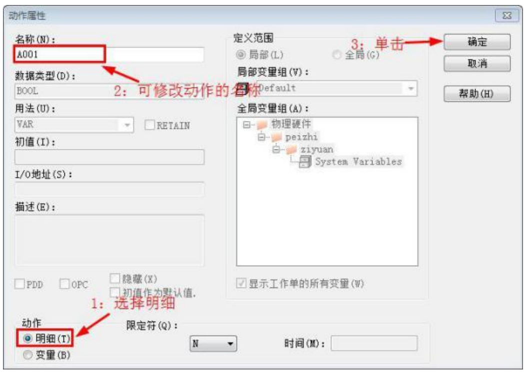
将“动作”A005 名称改为东西_红灯；

将“动作”A006 名称改为南北_绿灯；

将“动作”A007 名称改为南北_黄灯；

1）双击“步”S001 在弹出来的“SFC 步”对话框可以对“步”进行命名，以及类型选择，这里选择“初始步”

2）双击“动作”A001 在弹出来的“动作属性”选择“明细”，将 A001 名称修改初始化，如下图所示



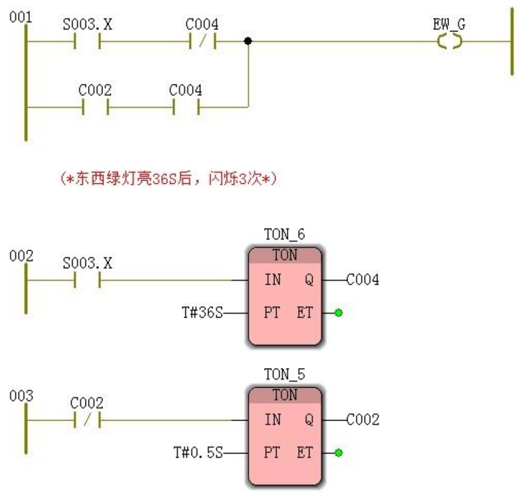
3）完成后，此时的“动作”名称为“初始化” 颜色由原来粉色变成了绿色，再次双击“初

始化” 在弹出来的“插入”框选择编程语言这里选择 LD 语言，单击“确定”，弹出一个空白的 LD 编程界面用户可以在此编写初始化的“动作”代码；这里选择空白。

4） 重复 2~3 的步骤，将 A001~A007 每一个动作的名称进行修改；

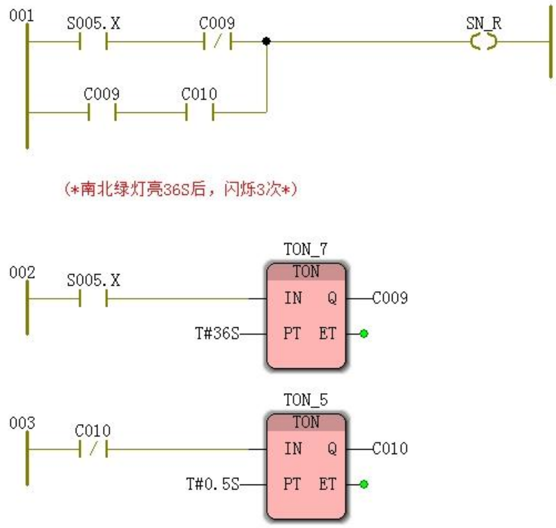
5） 双击“南北_红灯”进入‘南北_红灯’动作的编程，在编辑区插入如下程序其中，线圈 SN_R 的 I/O 地址为：%QX0.0，表示“步”S002 激活时南北方向亮红灯。

6）双击“东西_绿灯”进入‘东西_绿灯’动作的编程，在编辑区插入程序线圈 EW_G 的 I/O 地址为：%QX0.1，表示“步”S003 激活时东西方向绿灯点亮。

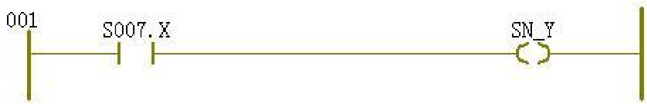


7）双击“东西_黄灯 ”进入‘东西_黄灯’动作的编程，在编辑区插入程序，线圈 EW_Y 的 I/O 地址为：%QX0.2，表示“步”S004 激活时东西方向亮黄灯。

8）双击“南北_绿灯”进入“南北_绿灯”动作的编程，在编辑区插入程序,线圈 SN_G 的 I/O 地址为：%QX0.3，表示“步“S005 激活时南北方向绿灯点亮。



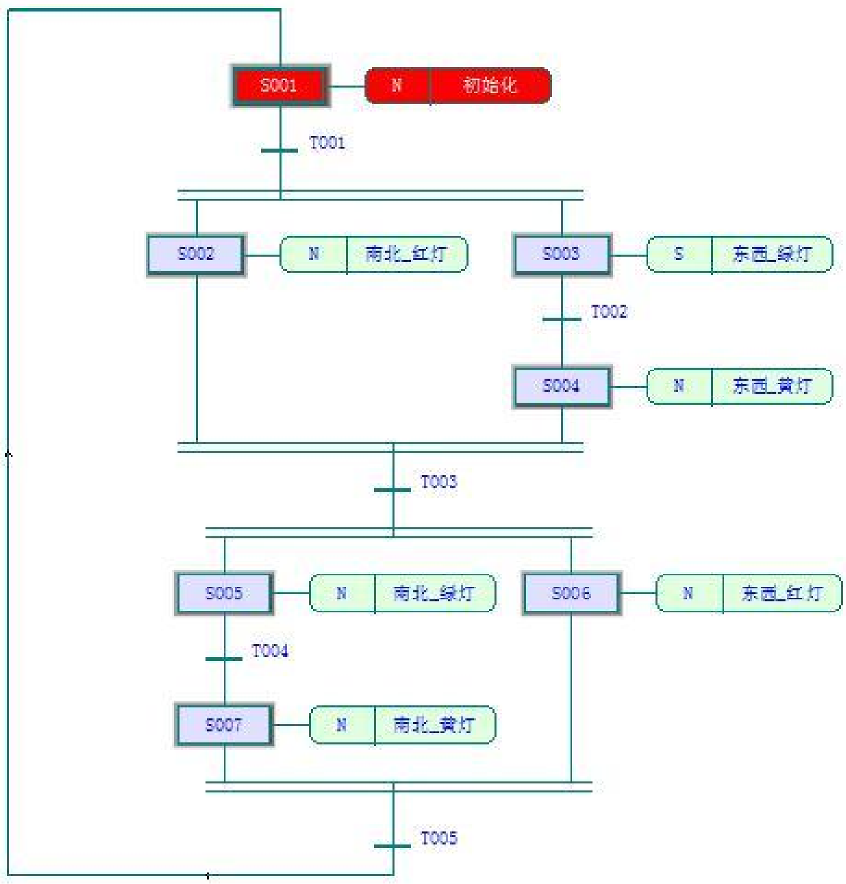
9）双击“南北_黄灯 ”进入‘南北_黄灯’动作的编程，在编辑区插入程序，线圈 SN_Y 的 I/O 地址为：%QX0.4，表示“步”S007 激活时南北方向亮黄灯。



10）双击“东西_红灯”进入‘东西_红灯’动作的编程，在编辑区插入如下程序其中，线圈 EW_R 的 I/O 地址为：%QX0.5，表示“步”S006 激活时东西方向亮红灯。

至此，一个完整的交通灯控制程序完成了，单击“制作”，无错误提示后进行下装到 PLC 后，单击“冷启”程序开始工作。这个程序是模拟的交通十字路口的红绿灯的自动控制，即当南北方向红灯亮时，保持 41 秒；东西方向绿灯亮，保持 36 秒，再闪烁 3 秒，然后黄灯亮，保持 2 秒整个周期为 41 秒；

最终的模拟仿真效果如图所示



6.6.3 SFC 的动作限定符

SFC 动作包括动作限定符和动作本体，动作限定符说明动作是如何与步关联的，以下的动作限定符都是可用的。

限定符	描述	功能
N	不保存	只要步是活动的，就执行动作代码本体或置位布尔变量。
R	超越复位	不再执行动作代码本体或者复位布尔变量。在使用‘S’限定符之前，必须要置位动作。
S	置位（保存的）	执行动作代码本体或者置位布尔变量。一旦（相关联的）步变为活动状态，就保存这个（置位）状态。该置位状态，仅能通过使用‘R’限定符，将该动作关联到另一个不同的步上，来显式地复位。
L	限时的	只要步是活动的，就执行动作代码本体或置位布尔变量，但最多能持续所设定的时间间隔这样一段时间。
D	延时的	在所设定的延迟时间流逝之后，再执行动作代码本体或置位布尔变量。只要步是活动的，动作就保持活动状态。如果步处于活动状态的时间比所设定的延迟时间短，则动作不会变为活动状态。

P	脉冲	步一变为活动状态，动作代码本体就执行一个操作周期，或布尔变量就置位一个操作周期。
SD	保存并延时	在步被激活之后，当又流逝了所设定的延迟时间之时，就执行动作代码本体或保存并置位布尔变量，即使步又变为不活动状态。该动作在被复位之前，将一直保持活动状态。如果步处于活动状态的时间比所设定的延迟时间还短，则该动作无论如何也会变为活动状态的。
DS	延时并保存	在步被激活之后，当又流逝了所设定的延迟时间之时，就执行动作代码本体或保存
SL	保存并限时	只要步是活动的，就在一个固定时间间隔内，执行动作代码本体或置位并保存布尔变量。如果步处于活动状态的时间比该时间间隔短，则该动作也无论如何会在整个时间间隔内处于活动状态。如果在该时间间隔内，动作被复位，该动作马上就会变为不活动状态。

第七章 工程创建及配置

7.1 创建工程项目

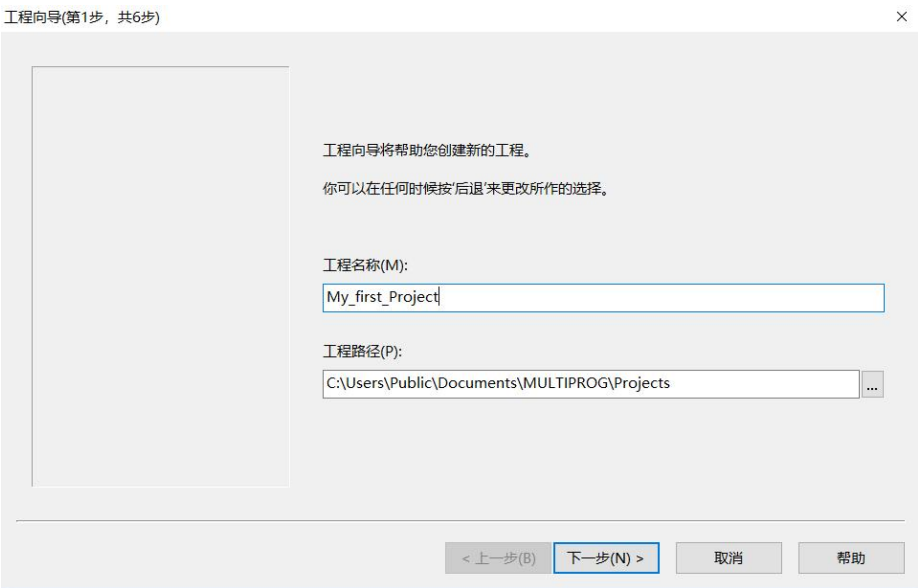
1）这一节我们对于使用 MULTIPROG 软件开发、编辑和运行一个梯形图（LD）示例程序的循序渐进的指导。将程序的开发分为几个阶段，如下图所示:

创建	编辑	编译	下装	调试	保存
（阶段 1 ）	（阶段 2）	（阶段 3）	（阶段 4）	（阶段 5）	（阶段 6）

使用工程向导将指导您创建新工程项目，在这里，用户必须定义工程的名称和路径、编程语言、以及所使用的 PLC 类型。

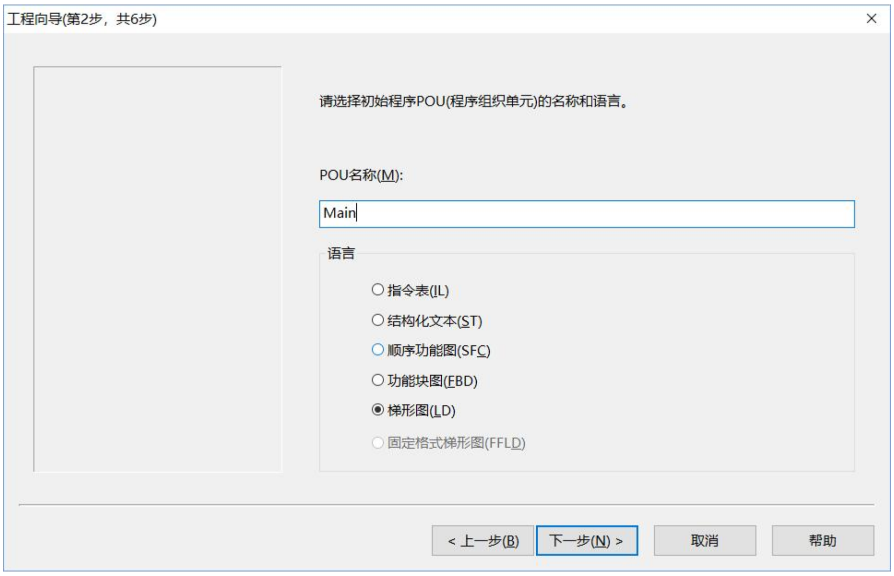
（1）单击“文件” ⟶ “新建工程”；

（2）在向导窗口的“工程名称”框中输入“My_first_Project”，如图 6-1 所示；根据工程的命名规则，工程的名称和路径一定不能含有空格或特殊字符，“工程路径”输入框指明了工程保存的路径，初始状态下为默认路径，用户可以自行制定，完成后单击“下一步”按钮；



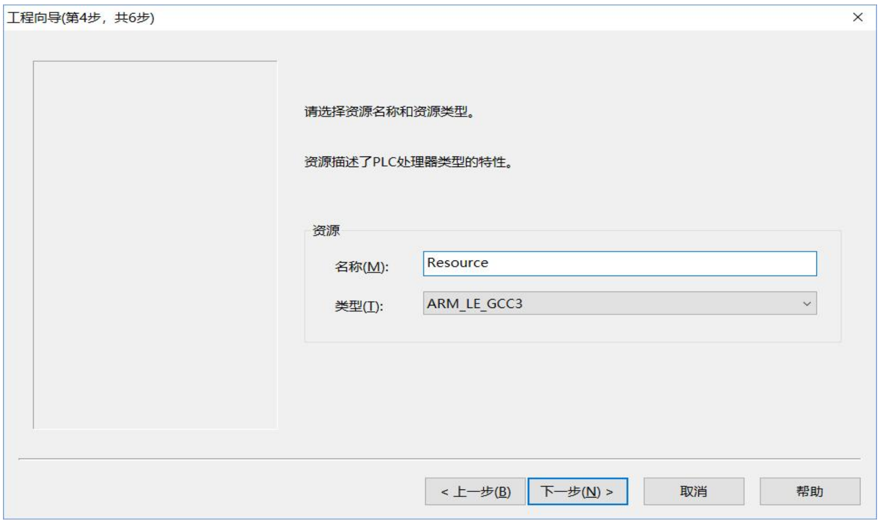
***注意**：工程名称和路径中不能包含特殊字符，否则无法成功创建工程。

（3）工程向导第二步对话框如图所示。将第一个 POU 取名为“Main”，编程语言选择“梯形图（LD）”单击“下一步”



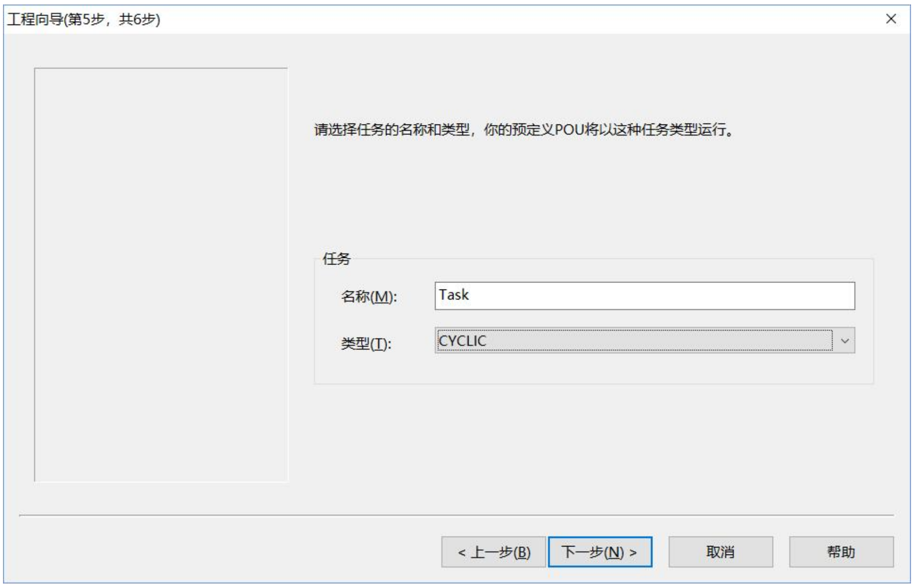
（4） 工程向导第三步用于确定配置的名称和类型，对话框如图所示。在“名称”输入框中填入配置的名称，这里保持默认的“配置”。在“类型”列表框中选择 PLC 类型为“eCLR”，单击“下一步”。（注！ 软件默认选择 “eCLR”，所以新建工程时，自动跳到下工程向导第四步）

（5） 工程向导第四步用于选择所要使用的“资源”，对话框如图所示。“名称”保持默认的“资源（Resource）”，“类型”列表框中选择“ARM_LE_GCC3”（真实的 PLC），（如果是使用 MULTIPROG 自带的模拟 PLC 进行仿真，选择“eCLR_Simulation”，单击“下一步”继续。）

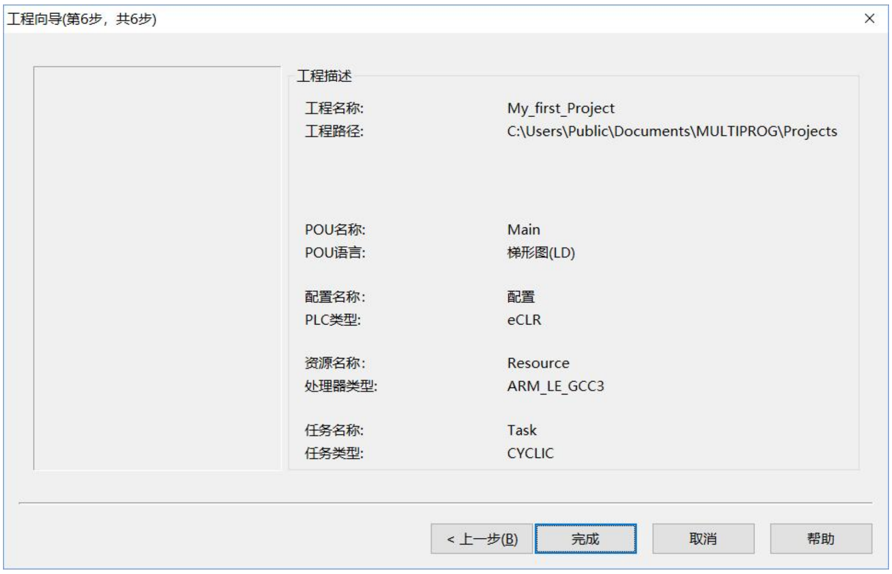


注意：不同的类型选择代表不同的硬件平台，因为在工程制作时，系统会根据资源的类型生成特定于硬件的机器码。

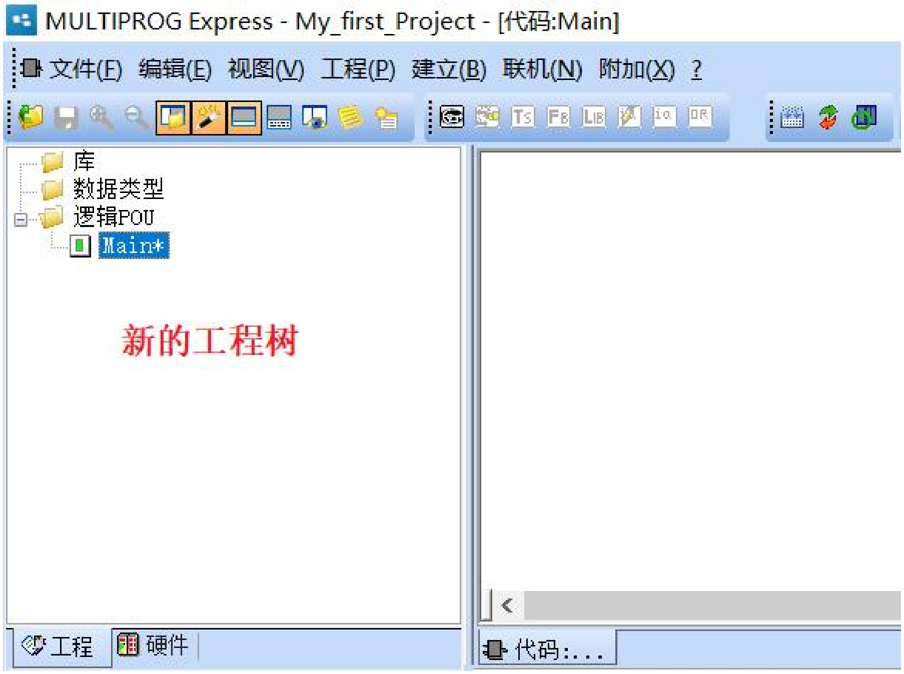
（6） 向导第五步用于指定任务的名称和类型，对话框如图所示。这里仍旧保持默认的名称“任务（Task）”，类型选择为“CYCLIC”，单击“下一步”。



（7）最后一步，在向导弹出的对话框对前面设置进行了小结，工程名称，工程路径，POU 名称，PLC 类型配置，处理器的类型，任务类型，如图所示。

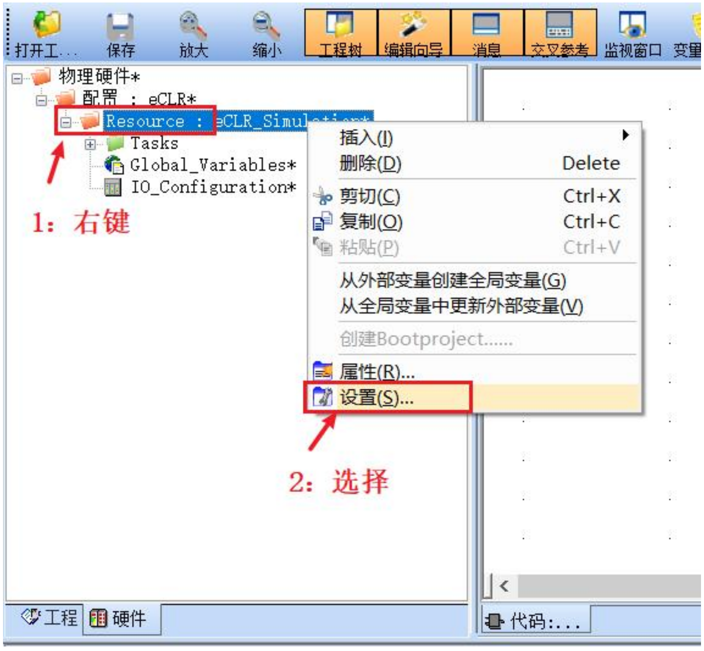


（8） 如果没有错误提示，单击“完成”，就可以在工程树窗口中看到新生成的工程树。如图所示。其中的“逻辑 POU”节点是有关于算法实现的部分，而“物理硬件”则是和实际的控制器的类型和设置相关联。

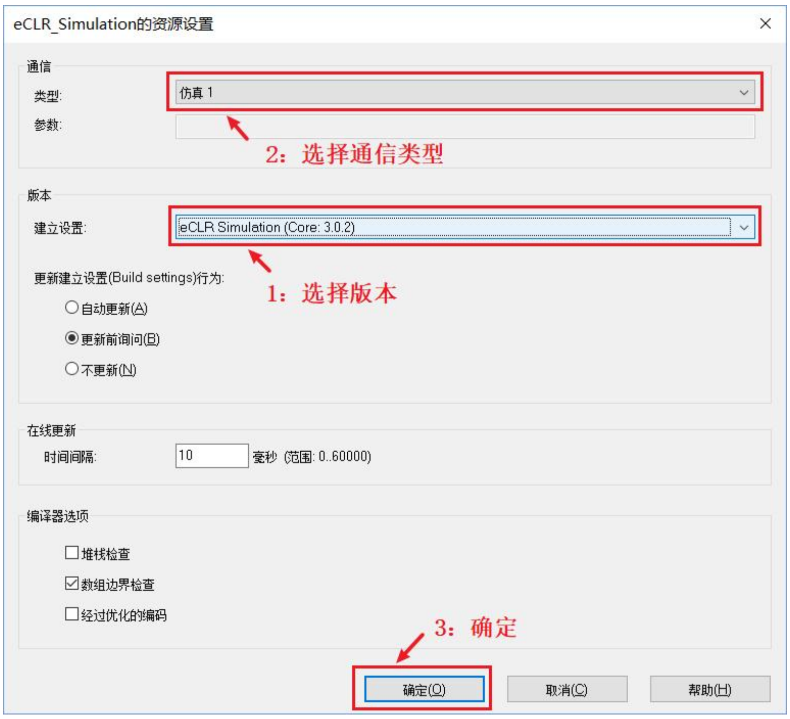


7.2 仿真通讯参数设置

1）安装完成后，打开已建好的工程项目，进行仿真通讯设置（前提是“资源”（Resource）选择“eCLR_Simulation”），在“工程树窗口”下，右键单击“资源”选择“设置” 如图所示



2）在弹出来的“eCLR_Simulation 的资源设置”窗口中进行选型，在“类型”下选择“仿真 1” 或者“仿真 2”；在“建立设置”下选择仿真处理器版本型号，这里选择“eCLR_3.0.2”；单击“确定”，再次重新单击“制作”按钮，没有错误提示后将程序下装到仿真的 PLC 中。



7.3 实物通讯参数设置

1） 将运动控制器进行上电通讯线连接完成后，进行 PC 机与控制器通讯参数设置，在“工程树窗口”下，右键单击“资源”选择“属性” 如图所示

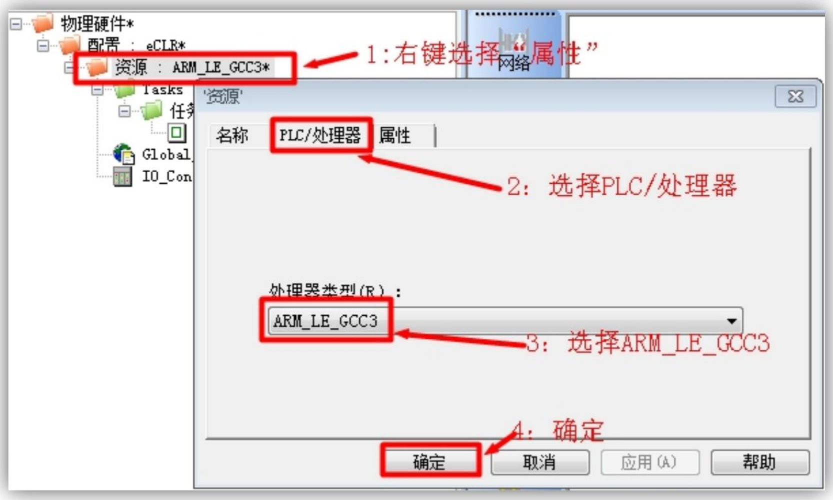


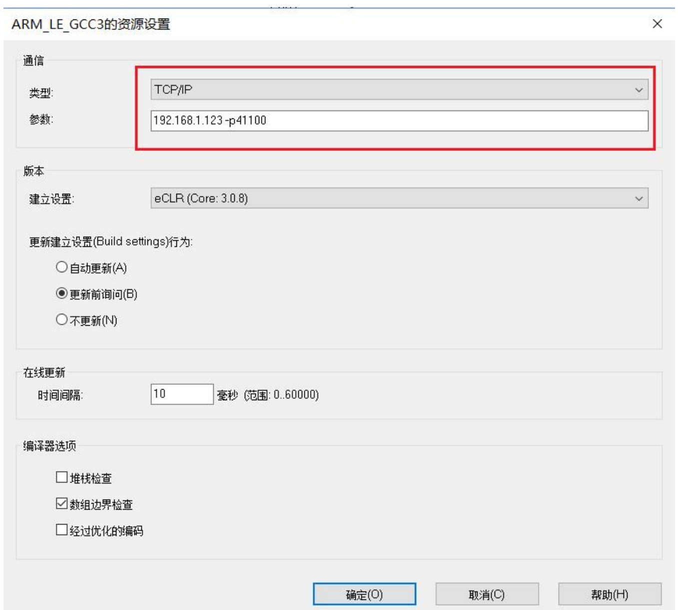
图 6-10 资源设置

（注意 ！ 在新建立工程向导时“PLC 处理器类型”已选择，仿真时处理器类型选择“eCLR_Simulation”；实物 PLC 时处理器类型选择“ARM_LE_GCC3”；此项用于仿真工程项目和实物的工程项目之间的切换）

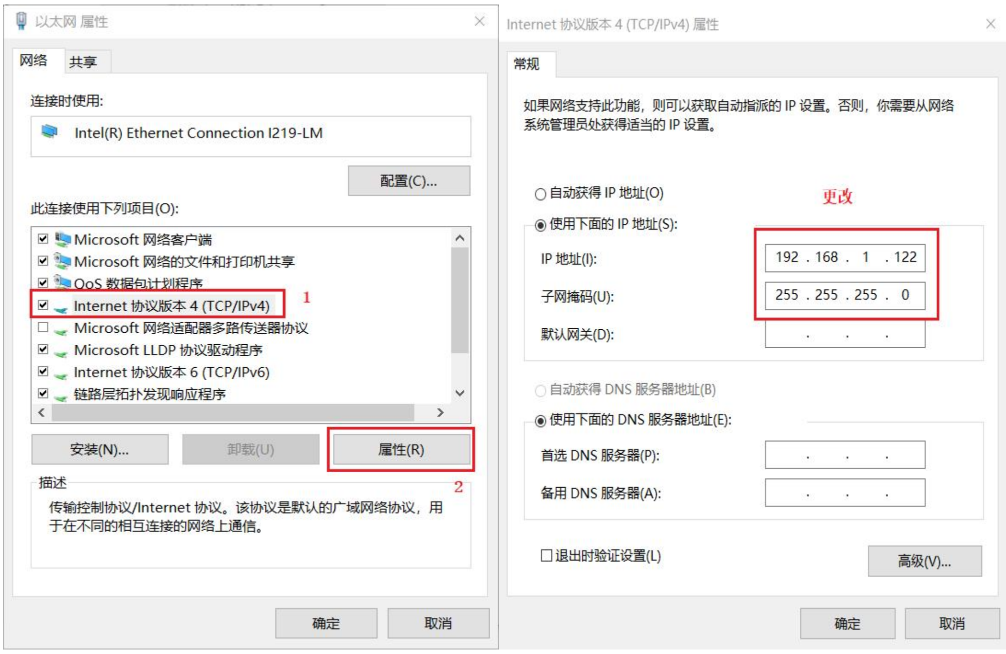
2） 在“工程树窗口”下右键单击“资源” 选择“设置”在弹出来的“ARM_LE_GCC3 资源设置”窗口中进行设置：

在通讯“类型”下选择“TCP/IP”；

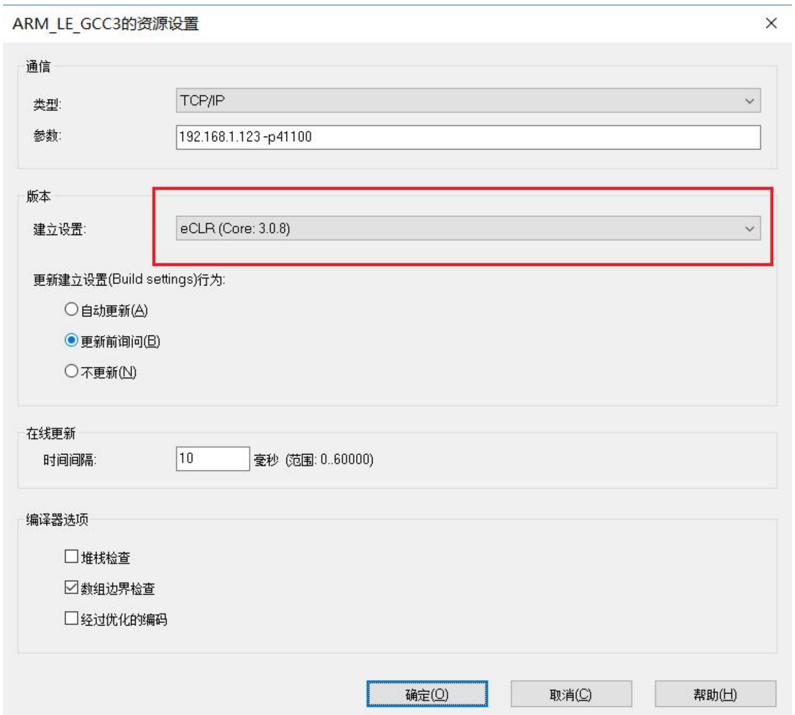
在通讯“参数”下设置 IP 地址为“192.168.1.123-p41100”



确认 PC 机的端口，PC 端口设置网络属性，IP 地址为：“192.168.1.122”，如下图



在版本建立设置选择“eCLR（Core.3.0.8）”，如图所示。



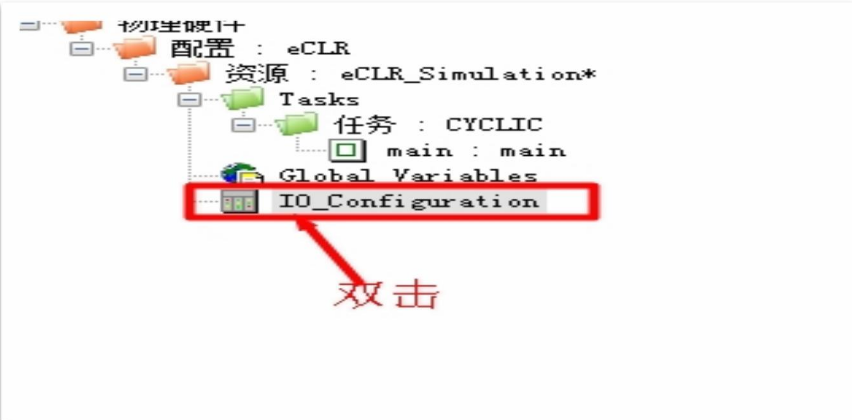
完成后，单击“确定”再次重新单击“制作”按钮；没有错误提示后程序方可下装到真实的 PLC 中。

7.4 IO 配置

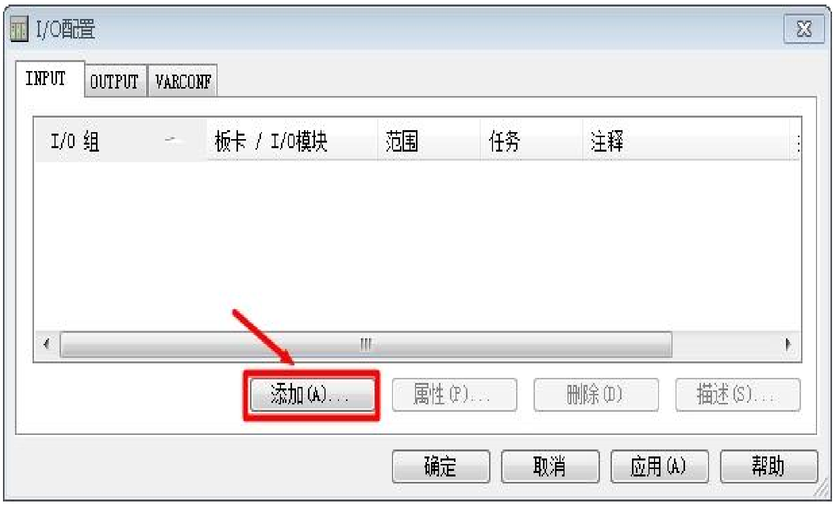
程序执行时，控制器通过 I/O 来接收来自现场设备的信号、发送控制命令到现场设备，所以用户必须指定逻辑起始地址，驱动程序名称是指定 I/O 的驱动，否者编译将出现“ I/O 变量'xxx'的地址与任何 I/O 组都不匹配！ ”的错误提示。

以下进行 I/O 驱动设置；

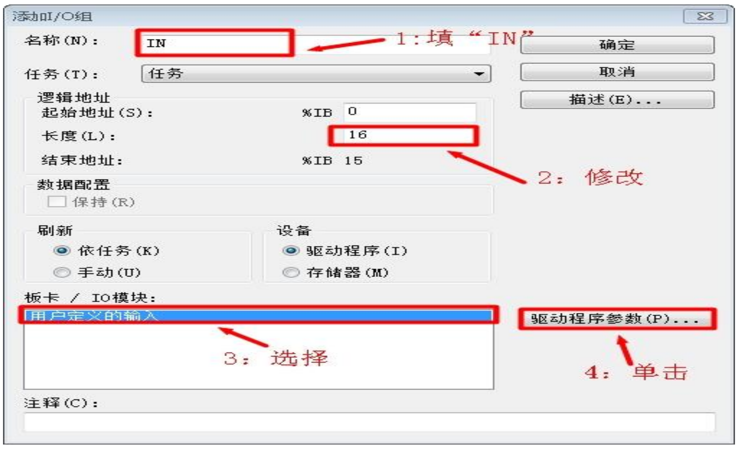
1、双击“IO_Configuration”打开 I/O 配置对话框，它用于编辑 I/O 组态工作单，如下图所示



2、 双击出现“I/O 配置”对话框，选择“INPUT”单击“添加”如图所示



3、 在名称填“IN”按照实际需要对 I/O 配置进行修改这个配置，例如：我们想在现有组中定义 16Byte 个输入点，在“长度”域中输入 16，表示输入位地址为：%IB0--%IB15 ，16 个输入字节。如图所示



4、单击<驱动程序参数>,“对驱动程序名称”修改为“KWIO” 如图所示

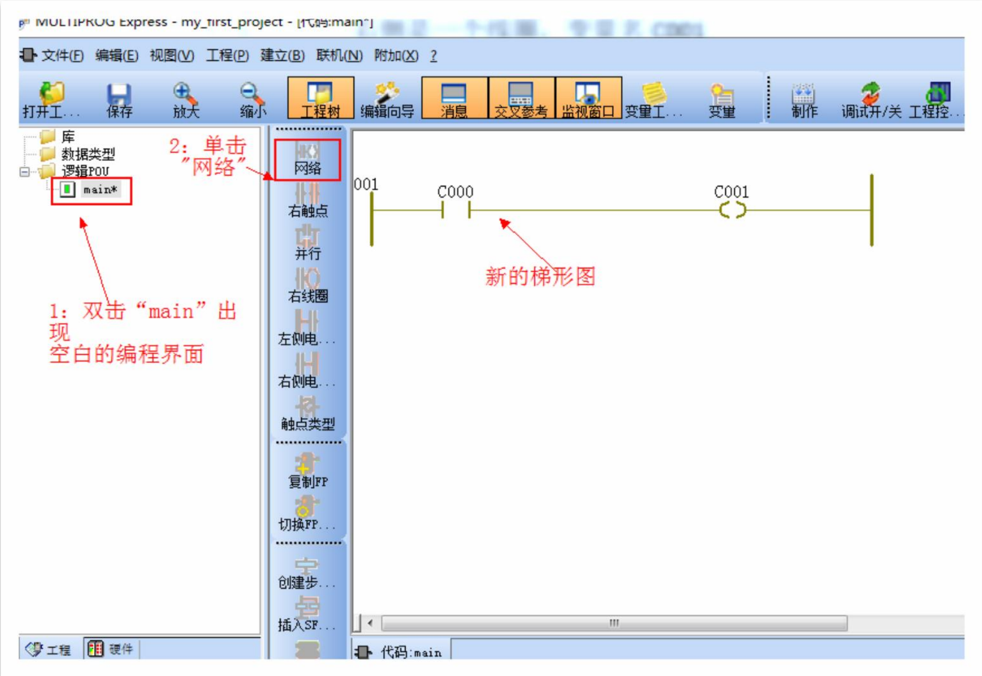
5、重复上述的步骤对输出端进行同样的设置,选中“OUTPUT”，单击“添加”将输出的“名称”改为“OUT”，逻辑地址“长度”修改为 16，同样“驱动程序名称”修改为“KWIO”单击“确定”，即可完成 I/O 驱动设置。

注意：用户每次新建工程时都需要进行 I/O 设置。

7.5 编写梯形图代码

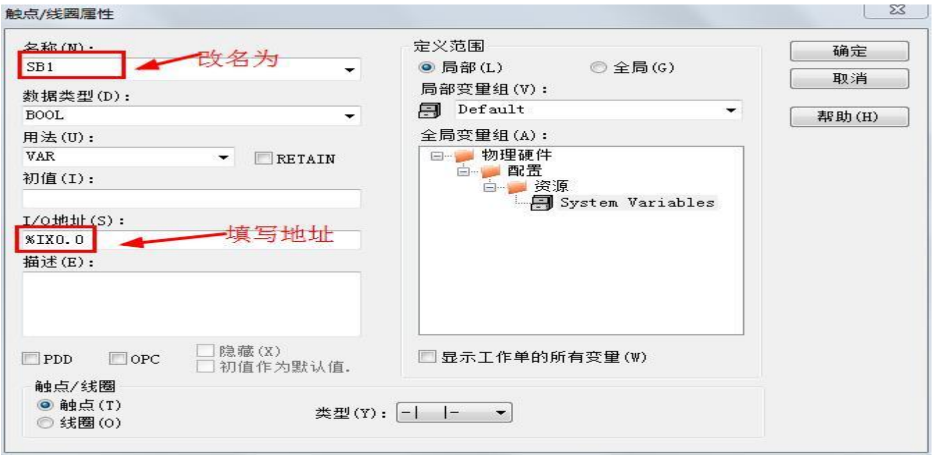
7.5.1 插入新的网络

1）如下图所示双击工程树中“main”工程名，此时出现空白的编辑窗口；单击“网络图标”，在编辑区域中出现一个梯形图网络，左侧是常开触点，变量名 C000　右侧是一个线圈，变量名 C001。

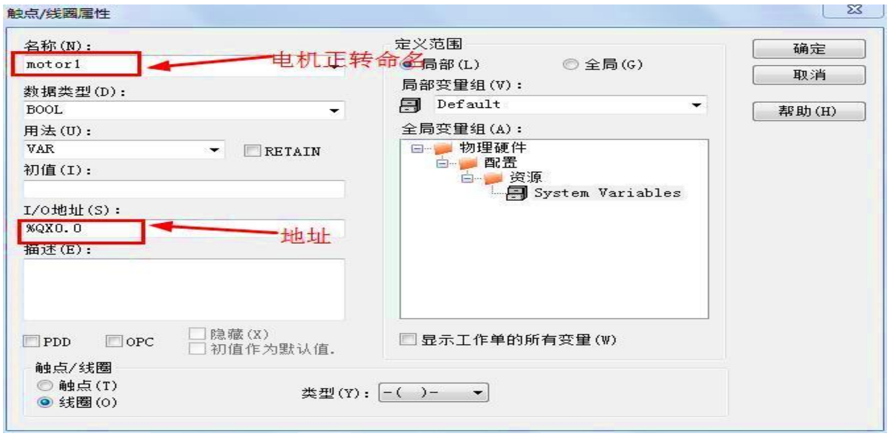


7.5.2 修改变量的属性

1）　双击常开触点“C000”，出现触点/线圈属性对话框，在 I/O 地址（S）栏，输入%IX0.0，表示 PLC 本机的第一个数字量输入通道，单击“确定”，如图所示

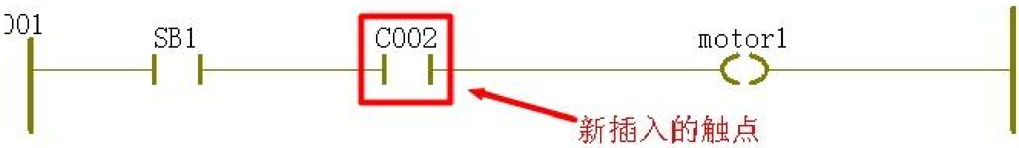


2） 双击梯形图中线圈 C001,出现触点/线圈属性对话框，在 I/O 地址 （S）栏，输入栏输入 %QX0.0 表示 ，表示 PLC 本机的第一个数字量输出通道，单击 “确定 ”，如图所示

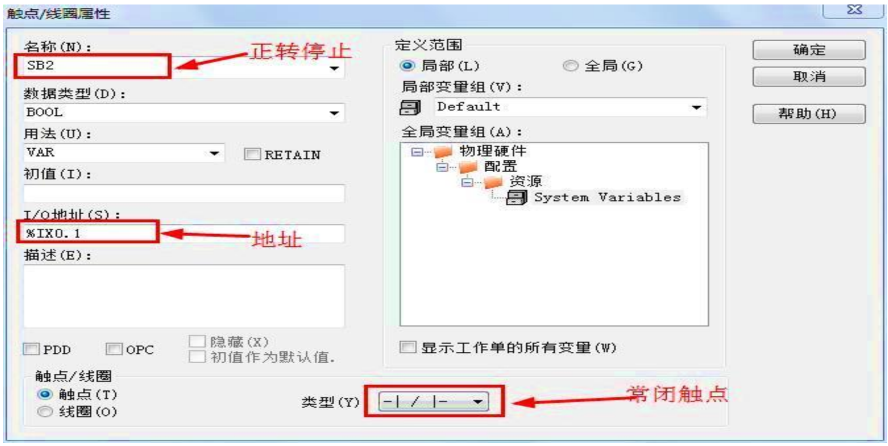


7.5.3 插入新触点

1）单击“SB1”，在选中的情况，单击工具栏上的触点如图所示，即在“SB1”后插入常开触点 C02，如图所示；



2）双击“C002”修改触点其属性如图所示



同理插入热继电器的常闭触点：

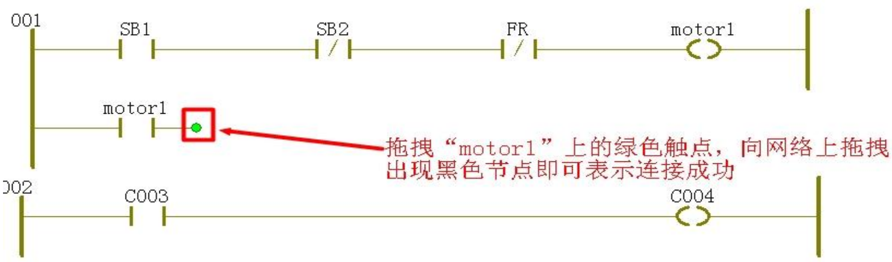
7.5.4 插入第二个 LD 网络

1） 把鼠标移到原有的网络下方，单击工具“网络”按钮即可插入新的网络，如下图所示

7.5.5 画线的连接和注释

1）画线的连接

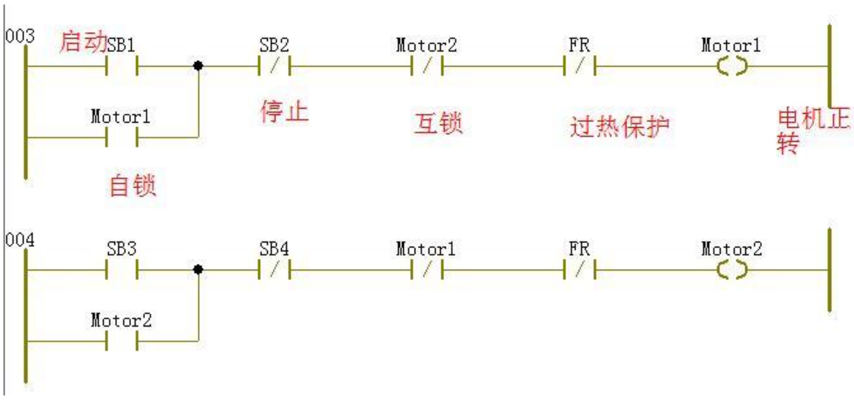
MULTIPROG 软件，提供了方便的拖拽功能，单击工具栏上的触点“”，此时出现一个新的“常开触点”，双击新的触点修改其属性，然后把触点拖拽要连接的网络上出现黑节点即可，如图所示



2）梯形图注释

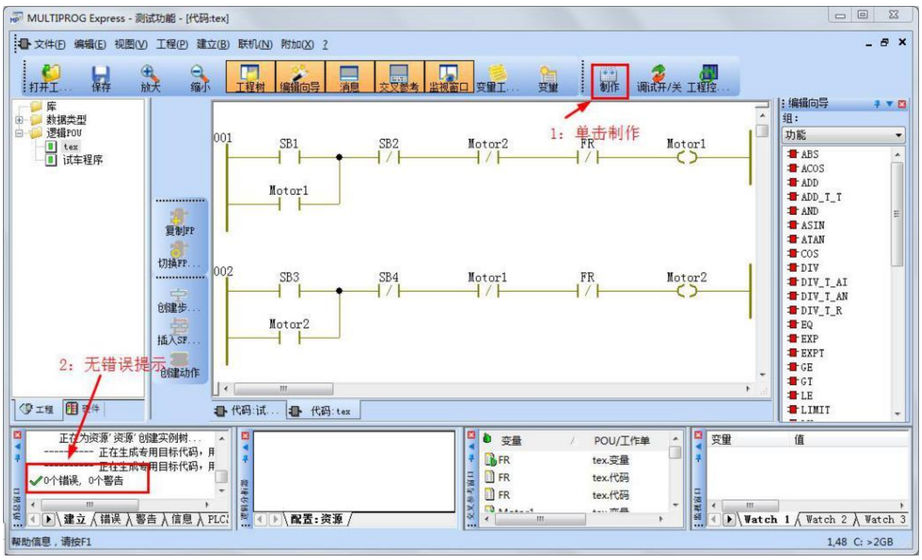
双击网络的左母线标号，在弹出来的“注释”对话框填写的程序注释，以便日后的理解如图所示

重复上步骤：最终按题目要求的梯形图如图所示



7.6 工程的制作与编译

1）单击编译“制作”，信息状态区没有错误的提示后进行下一步的仿真阶段，如有提示，双击消息区下的“错误”提示，软件自动跳到错误的地方，修改完后再单击“制作”，直到信息状态区没有错误提示后进行“下装”，如图所示



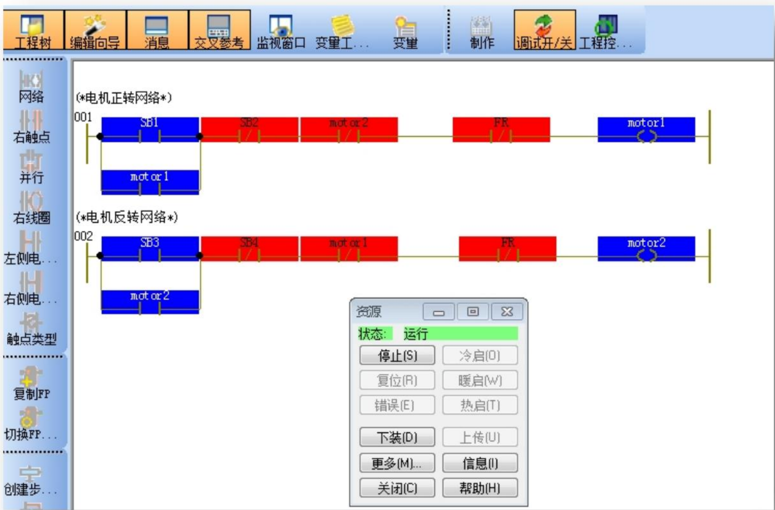
到目前创建的工程项目基本完成。

7.7 程序下装到 PLC/仿真

1）在通讯设置完成后，单击工具栏“工程控制对话框窗口”，弹出来的“资源”面板 ，单击下装，把已建好的工程项目下装到仿真器上，下装过程有下装进度，如图所示

2）程序“下装”完成，后单击资源面板上“冷启”程序才能运行，同时状态显示“运行”表明用户程序在不断地循环执行；用户可以在线调试，监视变量的状态，单击工具栏上“调试开光”，如图所示

4）仿真时画面效果，如图所示



📖　资源面板功能介绍

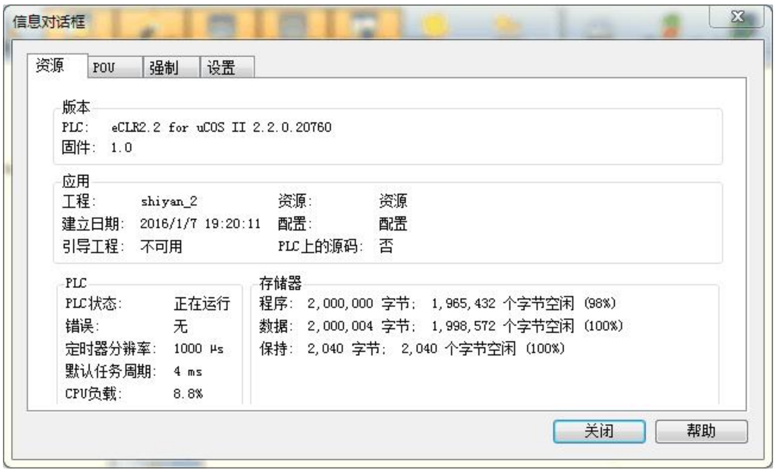
◆ 冷启：表示 PLC 从初始状态开始执行程序，所有的变量在启动时刻都是定义的初始值；

◆ 暖启：表示程序中的保持型变量维持上一次停止时的状态，其他变量则是初始状态。

◆ 提示：在“运行”状态单击“停止”按钮只是暂停 PLC 的运行，程序中所有的变量都保持停止前一刻的值，此时“暖启”按钮可用，单击该按钮可以继续程序的运行。

◆ 更多：用于选择是否保存用户下装的程序（在 VA 运动控制器掉电后），在用户程序下装前需要单击“更多”弹出来的窗口勾选“作为引导工程永久驻留”然后单击“关闭”如图所示 （注意:如果不勾选时，PLC 重新上电后将不能自行运行上一次下装的程序，即运动控制器不保存上一次的下装程序，需要重新下装程序）

◆ 信息： 用于查看 PLC 运行状态，程序存储器占用的字节，当前扫描周期设置多少等等；如图所示



注意：当 PLC 运行期间发生异常（例如除数为零）时，PLC 会自动停止，同时“状态”显示为“错误”，背景色为红色。此时“错误按钮”变为可用，单击该按钮，出错原因就会显示在 MULTIPROG 消息窗口的“PLC 出错”标签页中。

第八章 在线调试和监控程序

8.1 强制和覆盖

在联机模式下，可以“强制”或“覆盖”变量。两种情况下，都有一个新的值被赋给相应的变量。

✍ 强制：将一个值赋给一个变量（通常是一个触点或线圈）。该值将一直保持到复位强制时。

✍ 覆盖：由用户将一个值临时赋给一个变量。该值将一直保持到程序在下一个程序循环中用原值再次覆盖掉这个值的时候。强制和覆盖一个变量的必要步骤几乎是相同的。当 PLC 运行时，要特别小心地进行强制或覆盖变量。强制和覆盖变量意味着用强制的或覆盖的变量值执行 PLC 程序。

使用“强制和覆盖”

1➢ 要确保工作单处于联机模式。否则，按工具栏中的“调试开/关”图标：

2➢ 双击程序中的“变量”出现“调试：资源”对话框 如下图所示：

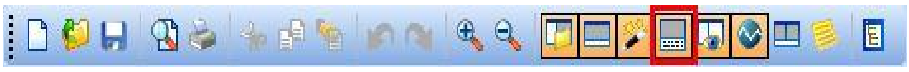
3➢ 选择单选按钮“TRUE”然后单击“强制”结果，该变量将被强制为“接通”，并在联机工作单中被加亮为红色。

8.2 变量监视窗口

变量监视窗口是一个功能强大的工具，允许用户很容易地将不同变量插入到列表中，并观察其运行期行为。一旦某个变量被添加到监视窗口，则不必打开相应的工作单，即可监视其当前值。用户可以专注于调试和观察所需变化的变量。如果还不是这种情况，请将工作单切换到联机模式，这要通过按下“调试开/关”图 标来进行。用鼠标右键单击工作单内部，并从上下菜单中选择“打开监视窗口”或者单击工具栏上 监视窗口按钮，在梯形图，或者在变量工作表中，选择要监视的变量右键“添加到监视窗口” 如图所示

8.3 交叉参考窗口

1）交叉参考列表包含了当前工程中使用的所有变量、功能块、跳转、标号和连接符。这个工具对于调试和错误隔离特别有帮助，单击工具栏中的“交叉参考窗口”图标，以打开交叉参考窗口如下图所示。



2）将光标放于交叉参考窗口中，并用鼠标右键单击窗口背景，以打开其上下文菜单。选择“建立交叉参考”菜单项，将创建交叉参考列表，如图所示。

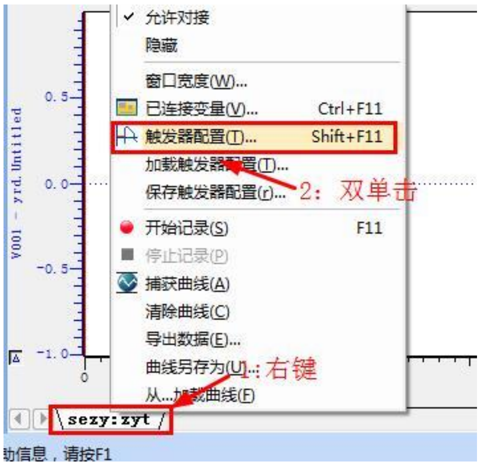
3）双击交叉参考窗口中的某个变量，将会打开使用了这个变量的工作单，并加亮此变量。另外，如果您在工作单中标记一个变量，交叉参考窗口中的相应变量也将被标记。交叉参考列表包含了当前工程中使用的所有变量、功能块、跳转、标号和连接符。这个工具对于调试和错误隔离特别有帮助。

8.4 逻辑分析器

1）逻辑分析器能监视多个模拟量（比如温度，液位，压力等）的趋势变化，可以设置采样的时间，方便用户调试在工具栏上选择“逻辑分析器”，在软件下方弹出“逻辑分析器”如图所示

2）用户在使用逻辑分析器前，需要将工程树“物理硬件”下的“配置”，“资源”，“任务”的属性名修改为英文名或者拼音，不能出现汉字否者不能正确使用逻辑分析器如下图设置，如图所示

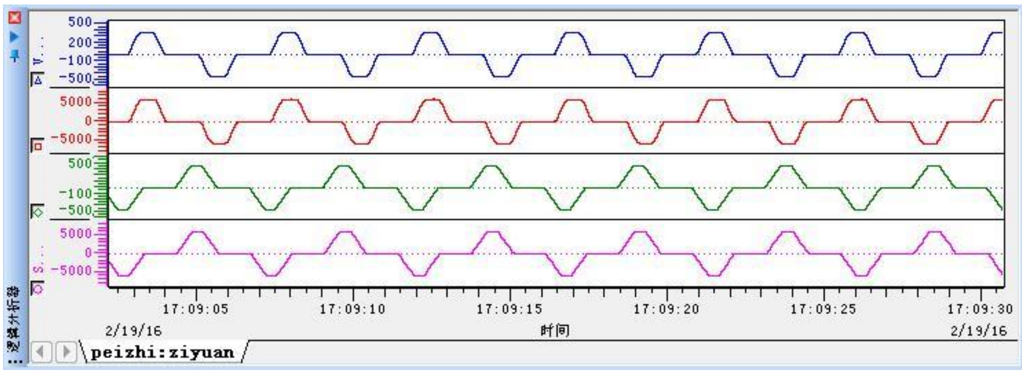
3）在“调试开/关”打开状态下，右键添加所需要监视的“变量”选择“添加到逻辑分析器”，此时“逻辑分析器”出现不同颜色的刻度线，表示用户添加了多个不同的变量，在弹出来的窗口左下方进行如下设置



4）在弹出来的“触发器配置”下 “勾选”连续记录，然后单击“确定”，如图所示

5）完成上述设置后 “右键”逻辑分析器名称，单击“开始记录”如图所示

6） 注意在单击“开始记录”时，尽量不要进行其他操作，否者会造成通讯不流畅或者死机，如果要进行其他操作时，先停止记录仪，再进行其他操作，最终监视的结果如下图所示。如需停止监视重复第五步操作选择“停止记录”即可。



8.5 I/O 面板控制

1）对于 MULTIPROG 自带的模拟 PLC，同样有类似于实际 PLC 的 I/O 面板。单击 Windows 系统通知栏区域的隐藏箭头，右键“EasySim eCLR 1”图标，在弹出的快捷菜单中选择“显示 I/O”菜单项，如图所示。

2）在程序中定义的 I/O 变量都会在该面板中列出，绿色表示的是输入量，而红色表示的是输出量。单击“值”列里面 SB3 对应的绿色小圆点，就可以到“值”列里面“Motor2”对应的红色小圆点也亮起，如图所示，这和程序设计的功能相符合。按下 SB4 之后，红色小圆点熄灭，说明程序正常运行。

8.6 断点调试

1） 像其他高级语言的开发工具一样，MULTIPROG 支持对 PLC 程序设置断点，用于调试程序。设置断点之后，对程序的调试有两种方式：单步和跟踪。

单步：PLC 执行当前指令的下一条指令，如果是功能或者功能块的调用，则会执行完整个调用过程得到结果。

跟踪：PLC 执行下一条指令，如遇到用户自定义功能或功能块调用，则会打开相应的代码本体。跟踪过程每次只执行一条指令。

1 ➢ 打开调试模式；

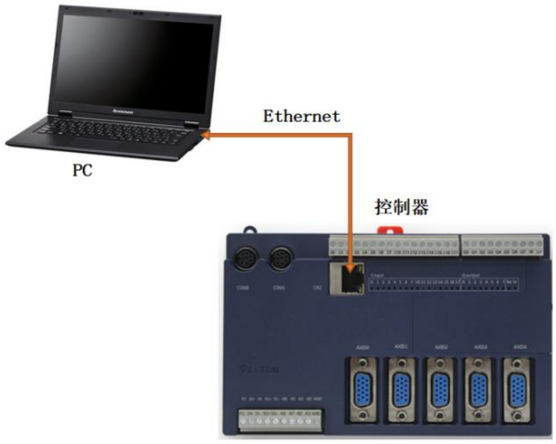
2 ➢ 在代码工作单中双击“SB1”变量，并在“调试:资源”对话框中单击“设置”按钮；在联机工作单中，“SB1”被橙色加亮，如图所示，此时工程控制对话框的状态也会变为“暂停[调试]”，背景为橙色，按钮也相应地编程调试的功能，如图所示；

2） MULTIPROG 支持两种标准的通信方式，一是串口，另一种是基于下 TCP/IP 方式的以太网。为了方便编程人员进行编程和调试，MULTIPROG 内置了两个模拟的控制器，“仿真 1”和“仿真 2”，其都是运行于和 MULTIPROG 同一个系统上的应用程序。注意:如果运行仿真控制器的操作系统是非实时的,则仿真只能用于程序的功能性验证,不能用于测试实时性“建立设置”是用于定义将 IEC 代码编译生成下装到控制器里执行的机器码的一整套库定义的文件,不同的版本生成的机器码可能不同,所以一般需要选择最新版的“建立设置”文件。“在线更新”的“时间间隔” 指的是在使用 MULTIPROG 的 Debug 模式时，控制器内部变量的值在 MULTIPROG 界而上显示的更新周期。

第九章 快速入门

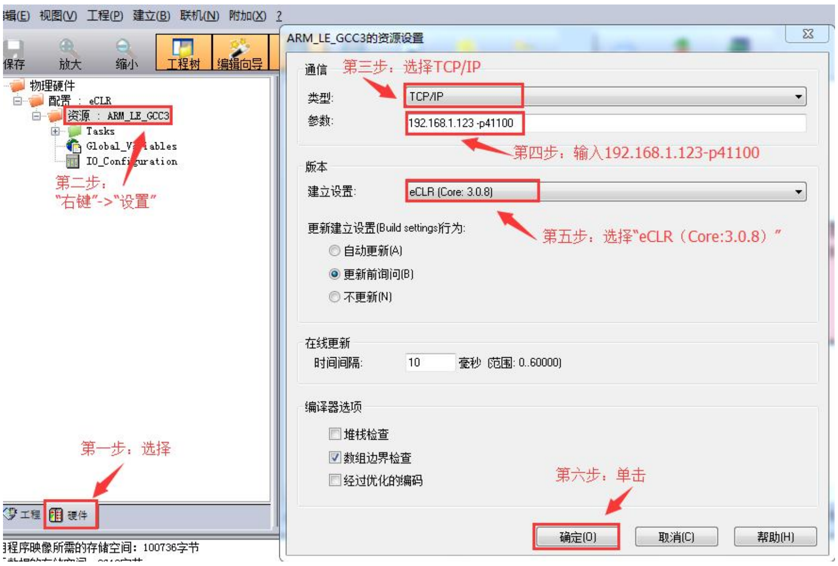
9.1 软件和运动控制器建立连接（网口通讯）

1： 通讯连接示意图



2：工程通讯配置

工程创建完成后，通讯快速配置如下。（参考：[7.1 创建工程项目]、[7.3 实物通讯参数设置]、[7.4 IO 配置]）



第一步：工程创建完成后，在工程树中选择“硬件”；

第二步：选中“Resource：ARM_LEGCC3”右键-> 设置；

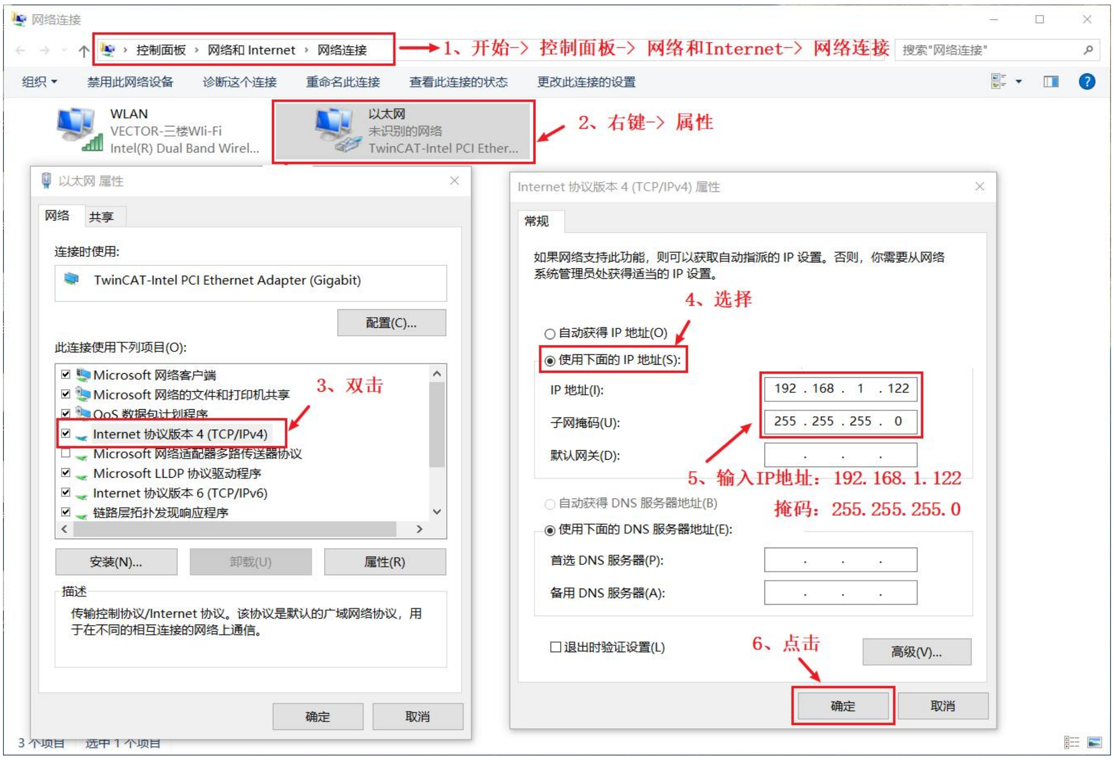
第三步：通讯类型： 选择“TCP/IP” ；

第四步：IP 地址“192.168.1.123-p41100”（控制器上的 IP 地址不允许更改 固定不变 ） ；

第五步：版本建立：选择 eCLR（Core.3.0.8）；

第六步：单击“确定”。

PC 电脑设置 如下图所示



电脑上的设置：

第一步：单击->开始->控制面板->网络和 Internet->网络连接 ；

第二步：右键“本地连接”–>属性；

第三步：双击“Internet 协议版本 4（TCP/IPV4）”；

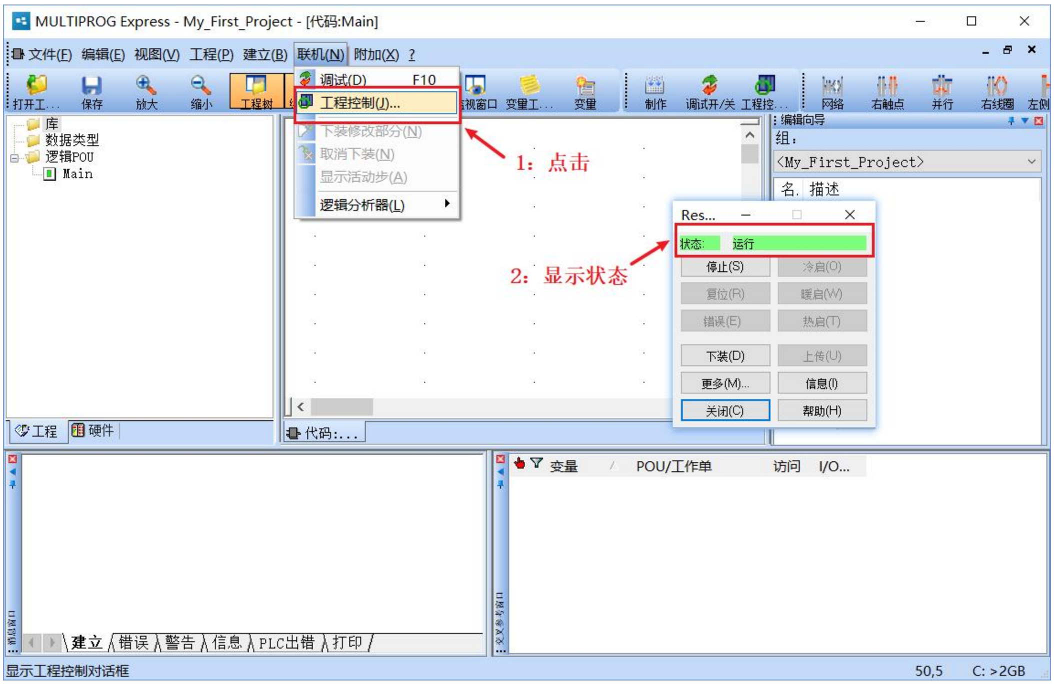
第四步： 选择“使用下面的 IP 地址（S）”；

第五步：输入 IP 地：192.168.1.122 子网掩码：255.255.255.0 ；

第六步：单击“确定” 设置完成。

3：检查配置是否成功

完成上述步骤后，单击“工程控制对话框”，此时将会弹出“资源”对话框 ，状态显示 “运行”，说明通讯设置成功，否则状态显示“超时”，此时检查 PC 端口号是否与软件设置一致，如下图。


MULTIPROG Express - My_First_Project - [代码:Main]
文件(F) 编辑(E) 视图(V) 工程(P) 建立(B) 联机(N) 附加(X) ?
调试(D) F10
工程控制(J)...
下装修改部分(N)
取消下装(N)
显示活动步(A)
逻辑分析器(L)
1：点击
2：显示状态
Res...
状态: 运行
停止(S)
冷启(O)
复位(R)
暖启(W)
错误(E)
热启(T)
下装(D)
上传(U)
更多(M)...
信息(I)
关闭(C)
帮助(H)
库
数据类型
逻辑POU
Main
编辑向导
组：
<My_First_Project>
名. 描述
建立 错误 警告 信息 PLC出错 打印
变量 POU/工作单 访问 I/O...
显示工程控制对话框
50,5 C: >2GB

9.2 控制器通过发模拟量控制伺服运动（单轴启停）

1： 通讯、控制连接示意图

按照上述系统，通过运动控制器控制伺服电机启动，运行和停止。

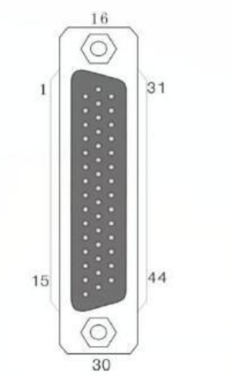
2： 设置伺服驱动器参数

威科达 VC 型伺服驱动器参数设置（VB 型伺服驱动器参数设置方法请咨询我司人员）

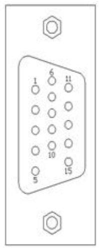
功能号	数值	描述
P02.01	1	速度控制模式
P04.01	0	速度来源于主速度 A
P04.02	1	主速度 A 来源于模拟量 AI1
P06.01	1	DI1 功能寄存器功能设置为：使能驱动器

3： 控制线接法

威科达 VC 型伺服驱动器 CN3 部分引脚定义

引脚	信号	描述	
37	OA+	编码器信号放大输出	
38	OA-		
39	OB+		
40	OB-		
12	AGND	模拟量地	
14	AI1	模拟量输入	
24	DI1	伺服 DI1 输入	
10、26	+24V	外接 DC24V 电源,供 伺服 DI、 DO 工作使用,备注： 25 、 26 脚供 NPN/PNP 跳线选择使用	
9、25	COM		

威科达运动控制器 AXIS 口部分引脚定义

引脚	信号	描述	
6	A+	编码器信号输入	
7	A-		
8	B+		
9	B-		
5	AO+	模量输出	
10	AGND	模拟量地	

控制线连接如下所示

伺服驱动器（CN3）		运动控制器（AXIS0）	
37	OA+	6	A+
38	OA-	7	A-
39	OB+	8	B+
40	OB-	9	B-
12	AGND	10	AGND
14	AI1	5	AO
26	+24V跳线		
27	SW-DI		
9	COM	0V	外接
10	+24V	+24V	外接
24	DI1	Q0	控制器外部端子输出

📖 说明：

1、为了通过控制运动控制器 Q0 输出，来控制伺服驱动器使能，需要给伺服驱动器外接 DC24V 电源给伺服的 DI 供电（如果伺服选择内部使能，则无需接 Q0 和 DI1）；

2、DIx 信号类型（NPN/PNP）选择：SW-DI（27 脚）与+24V（26 脚）短接为 NPN；

3、为了减少干扰，差分信号（OA+→A+）与（OA-→A-）、（OB+→B+）与（OB-→B-）分别用双绞线连接，外壳共地。

4：程序创建配置以及调试

📖 须知

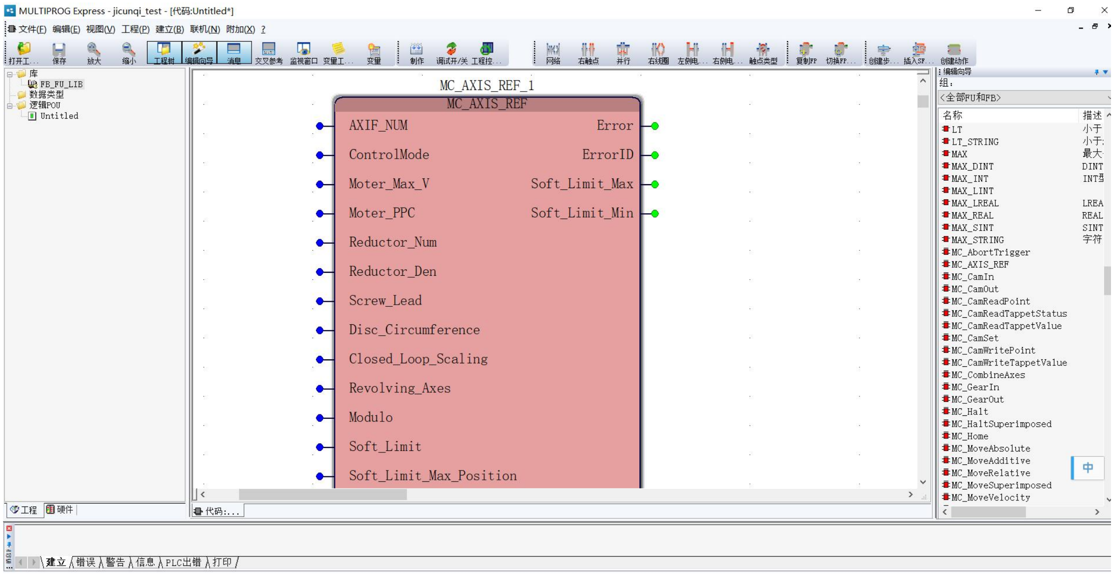
在上述 PC 与运动控制器通讯成功，系统搭建，伺服参数设置等完成后，接下来开始编程控制伺服电机运行和停止；在编程前默认用户已经阅读了《第十一章 运动指令》包括：11.1 插入 FB_FU_LIB 运动控制库、11.2 运动指令 、11.3 运动控制指令基本知识和 11.4 单轴指令。

（1）工程创建以及配置

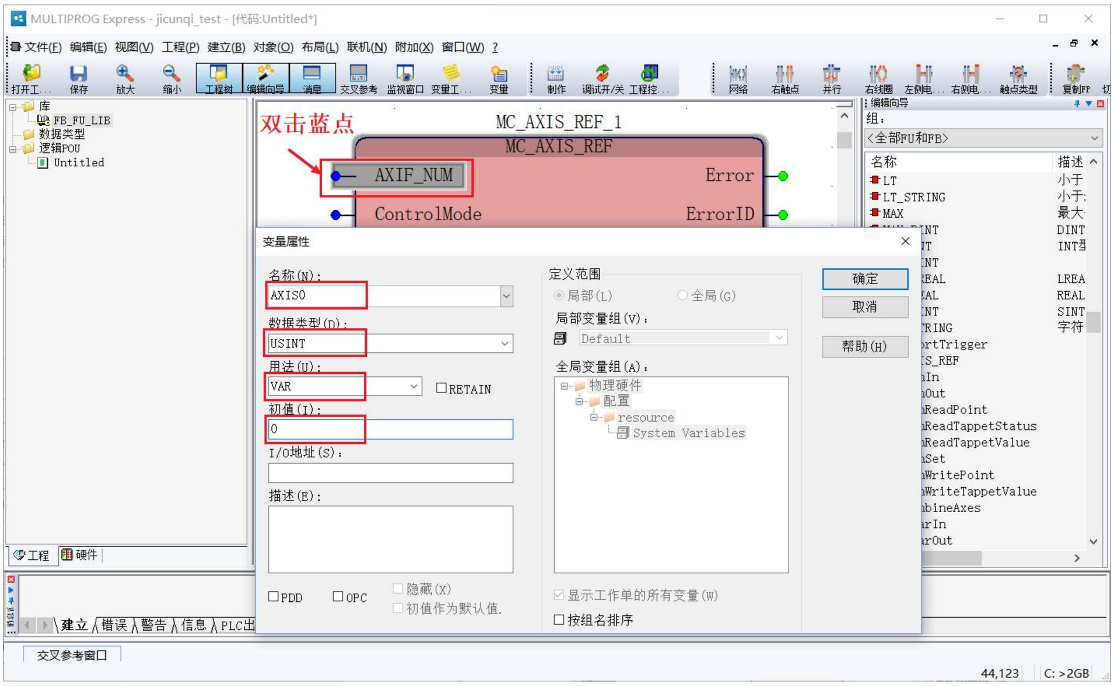
参考《第七章 工程的创建及配置》按照步骤完成：7.1 创建工程项目→ 7.3 实物通讯参数设置 → 7.4 IO 配置，在此不再重复赘述。

（2）编写程序

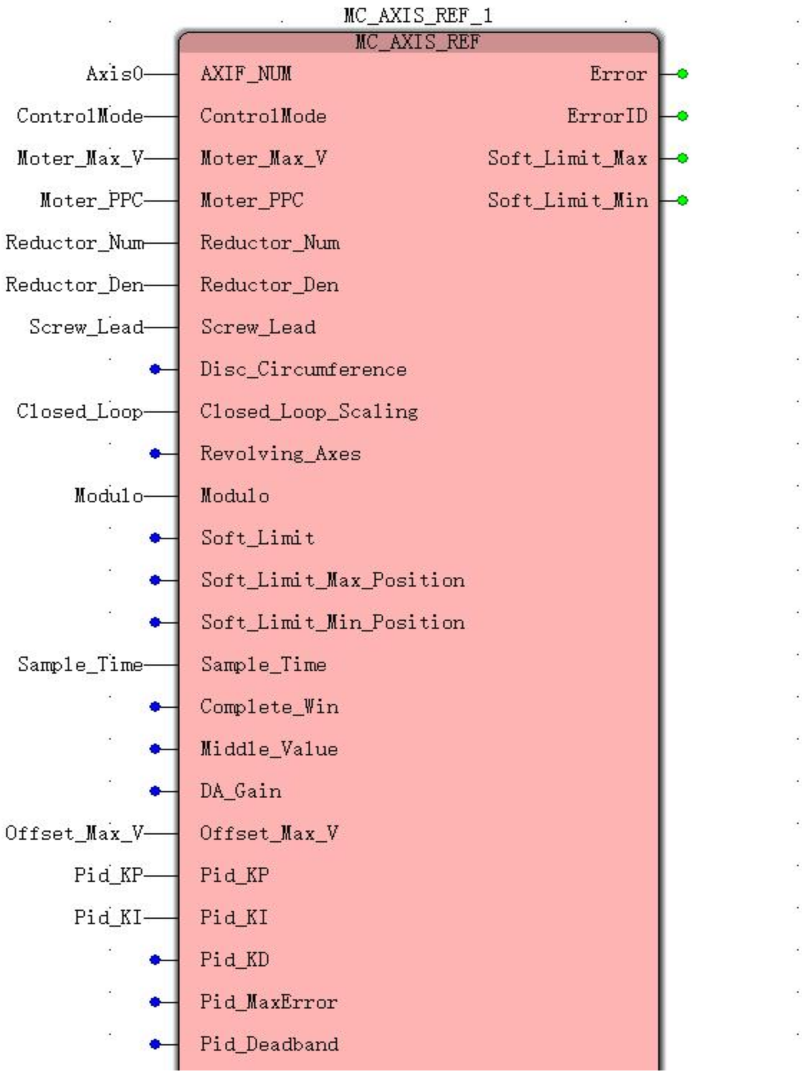
第一步：进入到编程界面，在 FB_FU_LI 运动控制库中选中“MC_AXIS_REF”块，按住鼠标左键拖拽到编程界面，然后松手，此时会弹出模块的属性，需要为模块命名，一般保持默认即可，单击“确定”，如下图所示；



第二步：双击模块输入引脚（蓝色点），此时弹出“变量属性”框，为其定义变量名，数据类型，用法，初值等，如图所示（这里使用轴 AXIS0 作为控制轴）；



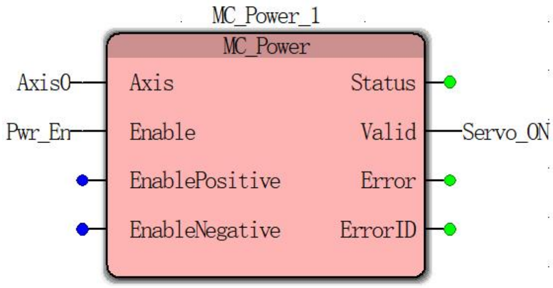
用户可以只需填写轴参数不可缺省的参数即可，可根据功能需要添加设置，参考 11.3.3 MC_AXIS_REF（轴参数设定），此处范例添加完成后，如下图所示；



变量名称和属性

变量名	数据类型	初始值
MC_AXIS_REF_1	MC_AXIS_REF	
Axis0	USINT	0
ControlMode	INT	0
Moter_Max_V	DINT	3000
Moter_PPC	DINT	10000
Reductor_Num	LREAL	1.0
Reductor_Den	LREAL	1.0
Screw_Lead	LREAL	60.0
Closed_Loop	LREAL	1.0
Modulo	LREAL	360.0
Sample_Time	WORD	20
Offset_Max_V	DINT	200
Pid_KP	DINT	80
Pid_KI	DINT	0

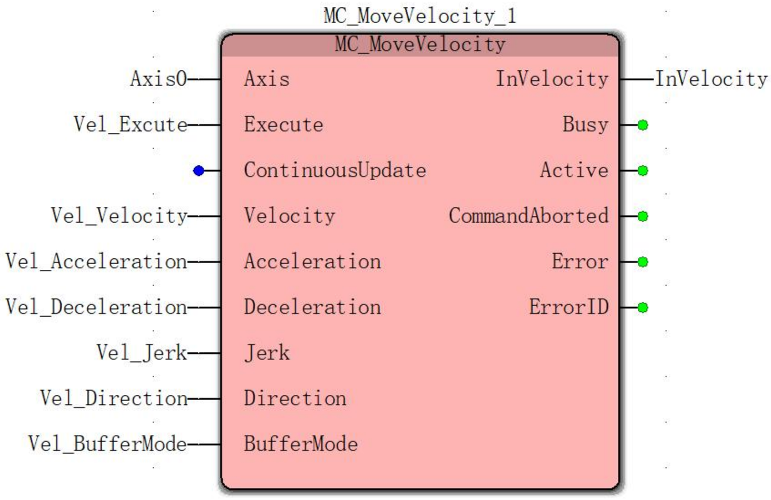
第三步：同理添加“MC_Power” 指令如下，使能运动控制器，同时通过 Servo_ON（I/O 地址：%QX0.0）输出控制伺服驱动器的使能，参考 11.4.1 MC_Power（使能指令）；



变量名称和属性

变量名	数据类型	初始值	地址
MC_Power_1	MC_Power		
Axis0	USINT	0	
Pwr_En	BOOL		
Servo_ON	BOOL		%QX0.0

第四步：同理添加“MC_MoveVelocity”（速度指令模块），用于控制伺服电机按设定的速度运转，参考 11.4.2 MC_MoveVelocity（速度指令）；

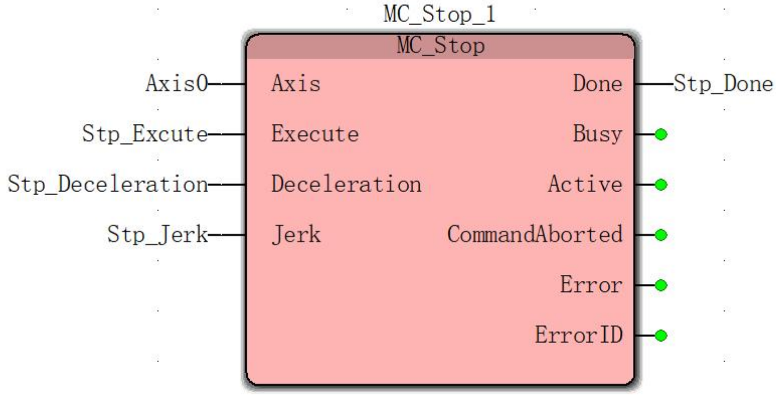


变量名称和属性

变量名	数据类型	初始值
MC_MoveVelocity_1	MC_MoveVelocity	
Axis0	USINT	0
Vel_Excute	BOOL	
Vel_Velocity	LREAL	500.0
Vel_Acceleration	LREAL	1000.0
Vel_Deceleration	LREAL	1000.0
Vel_Jerk	LREAL	1000.0
Vel_Direction	INT	1
Vel_BufferMode	INT	0

InVelocity	BOOL	

第五步：添加“MC_Stop”（停止指令），模块执行后，伺服电机开始减速停止，参考 11.4.10 MC_Stop（停止指令）；

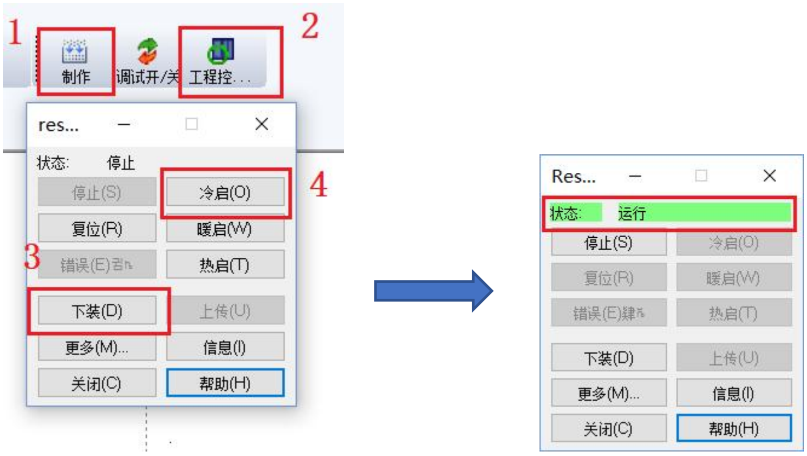


变量名称和属性

变量名	数据类型	初始值
MC_Stop_1	MC_Stop	
Stp_Excute	BOOL	
Stp_Deceleration	LREAL	1000.0
Stp_Jerk	LREAL	1000.0
Stp_Done	BOOL	

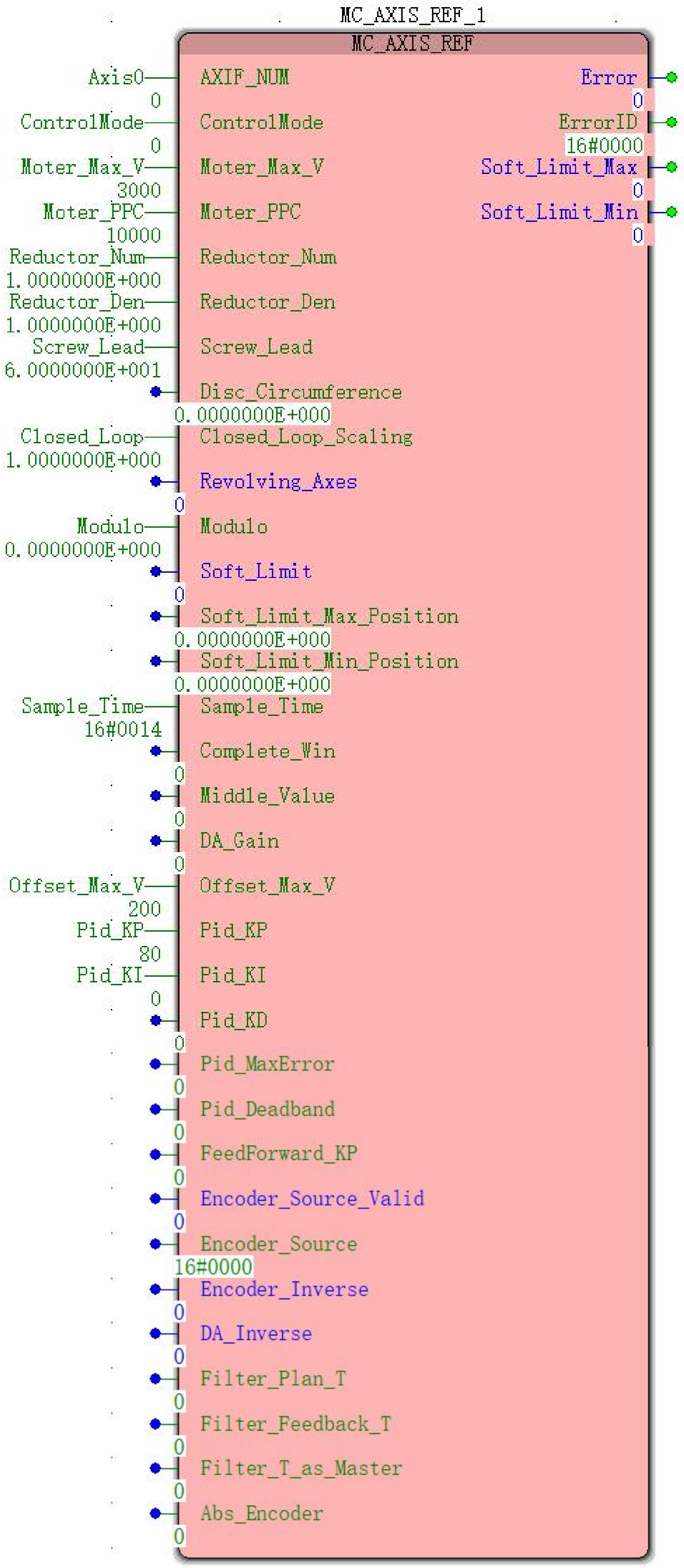
至此，程序编写完成。

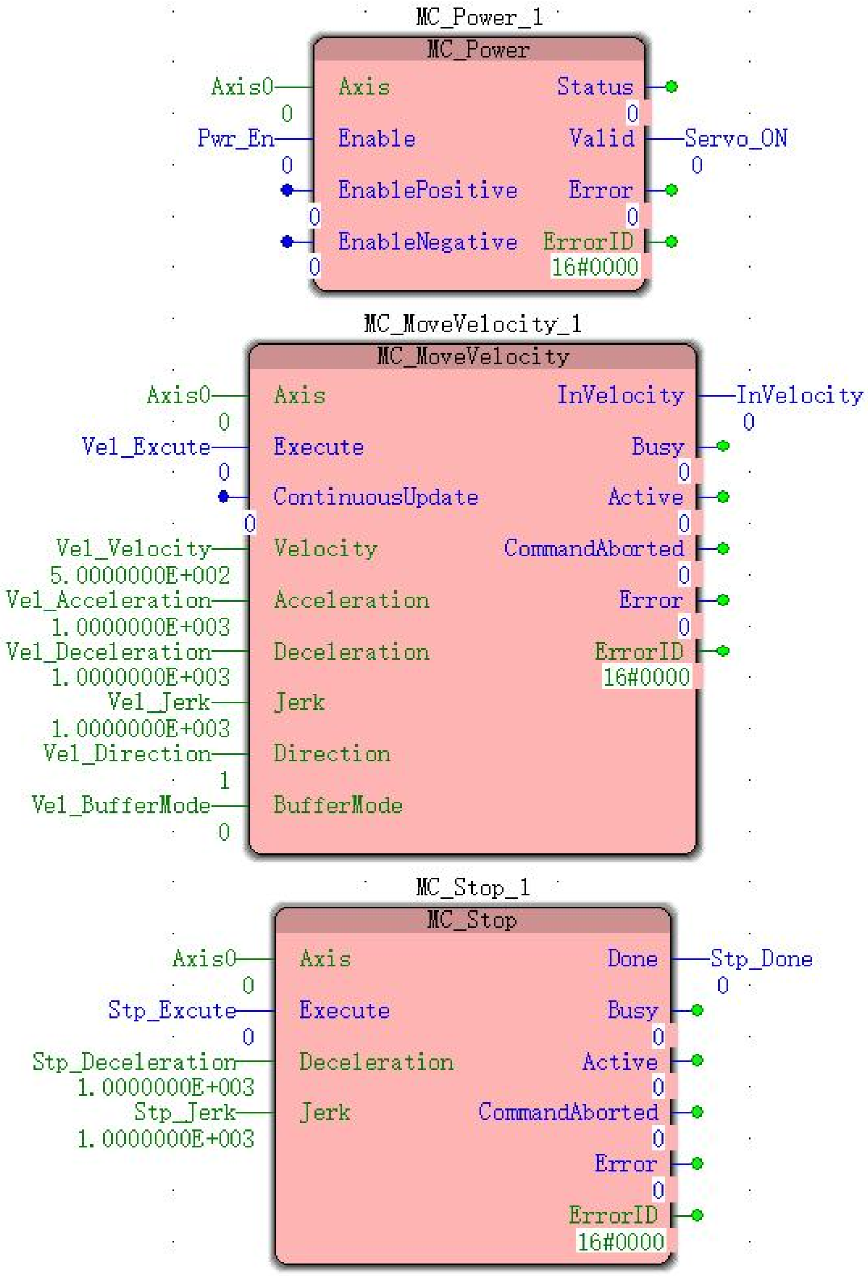
第六步：制作下装工程。在工具栏点击制作，确认程序无误后，点击下装程序，然后点击冷启，冷启成功后，状态显示为运行状态。



第七步：程序调试。单击工具栏上的在线图标 

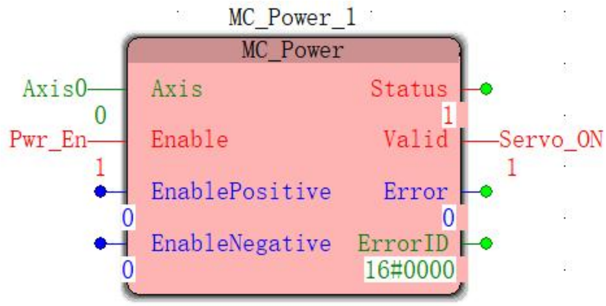
可以对程序进行调试和监控。在线监视如下图所示


MC_AXIS_REF_1
MC_AXIS_REF
Axis0 AXIF_NUM
0
ControlMode ControlMode
0
Moter_Max_V Moter_Max_V
3000
Moter_PPC Moter_PPC
10000
Reductor_Num Reductor_Num
1.0000000E+000
Reductor_Den Reductor_Den
1.0000000E+000
Screw_Lead Screw_Lead
6.0000000E+001
Disc_Circumference
0.0000000E+000
Closed_Loop Closed_Loop_Scaling
1.0000000E+000
Revolving_Axes
0
Modulo Modulo
0.0000000E+000
Soft_Limit
0
Soft_Limit_Max_Position
0.0000000E+000
Soft_Limit_Min_Position
0.0000000E+000
Sample_Time Sample_Time
16#0014
Complete_Win
0
Middle_Value
0
DA_Gain
0
Offset_Max_V Offset_Max_V
200
Pid_KP Pid_KP
80
Pid_KI Pid_KI
0
Pid_KD
0
Pid_MaxError
0
Pid_Deadband
0
FeedForward_KP
0
Encoder_Source_Valid
0
Encoder_Source
16#0000
Encoder_Inverse
0
DA_Inverse
0
Filter_Plan_T
0
Filter_Feedback_T
0
Filter_T_as_Master
0
Abs_Encoder
0
Error
0
ErrorID
16#0000
Soft_Limit_Max
0
Soft_Limit_Min
0

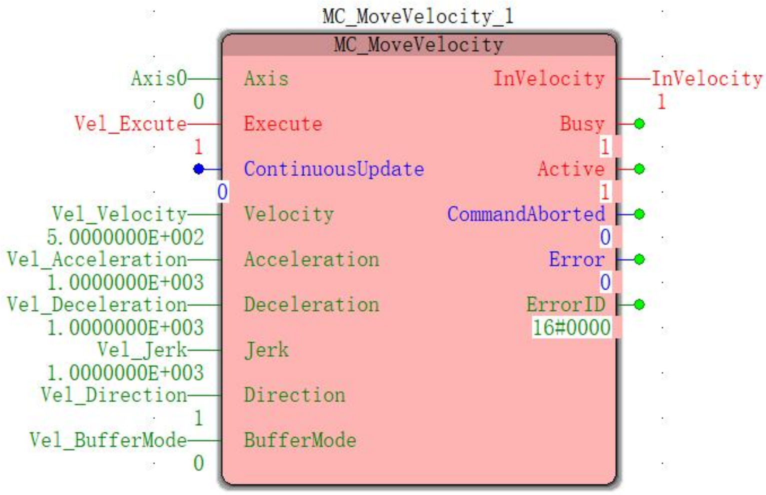


调试 1：双击输入功能位 Pwr_En，弹出调试：Resource 界面，变量值选择 Ture，然后点击覆盖，Pwr_En 将由 False 变为 Ture，如图所示；

当 Servo_ON 由 False 变为 Ture 时，说明轴 0 使能成功，并且通过运动控制器输出让伺服同时使能；

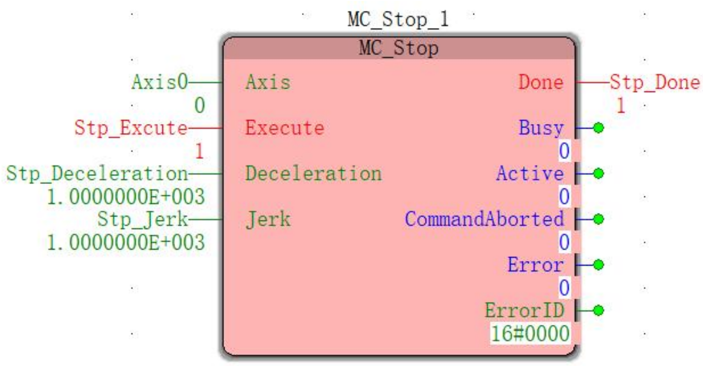


调试 2：同理，双击 Vel_Excute，使其值由 False 变为 Ture，控制器开始向伺服发送模拟量指令，电机开始向正方向加速运转，当 InVelocity 由 False 变为 Ture 时，速度到达预设值 500.0；



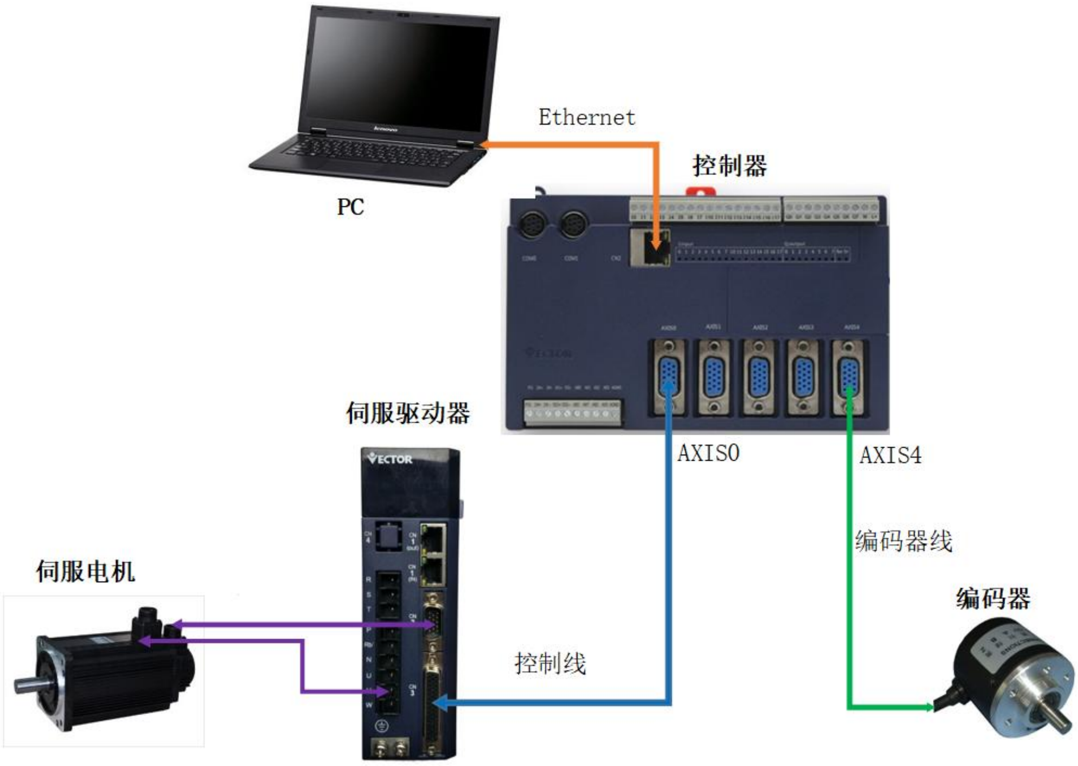
调试 3：双击 Vel_Velocity，在弹出的调试窗口重新赋值 1000.0，点击覆盖，然后再重新触发一次 Vel_Excute 使速度更新，电机转速将按照预设的加减速由 500.0 加速到 1000.0；

调试 4：双击 Stp_Excute，使其值由 False 变为 Ture，电机将按照预设的减速度减速直到停止，当 Stp_Done 由 False 变为 Ture 时，停止完成。



9.3 控制器通过发脉冲控制伺服运动（编码器带动伺服运转）

1： 通讯、控制连接示意图



按照上述系统，控制伺服电机跟随主轴（编码器）运行一个电子齿轮；

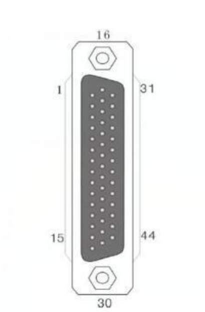
2： 设置伺服驱动器参数

威科达 VC 型伺服驱动器参数设置（VB 型伺服驱动器参数设置方法请咨询我司人员）

功能号	数值	描述
P02.01	0	位置控制模式
P03.01	0	位置命令来源于外部脉冲
P03.02	2	指令脉冲形态为 AB 脉冲
P06.01	1	DI1 功能寄存器功能设置为：使能驱动器

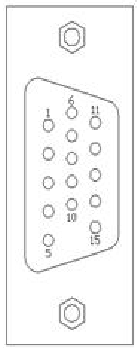
3： 控制线与编码器线接法

威科达 VC 型伺服驱动器 CN3 部分引脚定义

引脚	信号	描述	
31	X+	脉冲信号输入	
32	X-		
33	Y+		
34	Y-		
37	OA+	编码器信号放大输出	
38	OA-		
39	OB+		
40	OB-		

24	DI1	伺服 DI1 输入	
10、26	+24V	外接 DC24V 电源,供 DI、DO 工作使用,备注： 25、26 脚供 NPN/PNP 跳线选择使用	
9、25	COM		

威科达运动控制器 AXIS 口部分引脚定义

引脚	信号	描述	
1	X+	脉冲信号输出	
2	X-		
3	Y+		
4	Y-		
6	A+	编码器信号输入	
7	A-		
8	B+		
9	B-		
13	+5	DC 5V 输出	
15	GND	GND	

编码器引脚部分定义

引脚	信号	描述	
1	AO+	编码器信号输出	
2	AO-		
3	BO+		
4	BO-		
5	+5V	5V 输入	
6	GND	GND	

控制线连接

伺服驱动器（CN3）		运动控制器（AXIS0）	
31	X+	1	X+
32	X-	2	X-
33	Y+	3	Y+
34	Y-	4	Y-
37	OA+	6	A+
38	OA-	7	A-
39	OB+	8	B+
40	OB-	9	B-
26	+24V跳线		
27	SW-DI		
9	COM	0V	外接
10	+24V	+24V	外接
24	DI1	Q0	控制器外部端子输出

编码器线连接

编码器		运动控制器（AXIS4）	
1	OA+	6	A+
2	OA-	7	A-
3	OB+	8	B+
4	OB-	9	B-
5	5V+	13	5V+
6	GND	15	GND

说明：

1、为了通过控制运动控制器 Q0 输出，来控制伺服驱动器使能，需要给伺服驱动器外接 DC24V 电源给伺服的 DI 供电；

2、DIx 信号类型（NPN/PNP）选择：SW-DI（27 脚）与+24V（26 脚）短接为 NPN；

3、为了降低干扰，差分信号（OA+→A+）与（OA-→A-）、（OB+→B+）与（OB-→B-）以及 XY 脉冲信号分别用双绞线连接，外壳共地。

4： 程序的创建配置与调试

须知

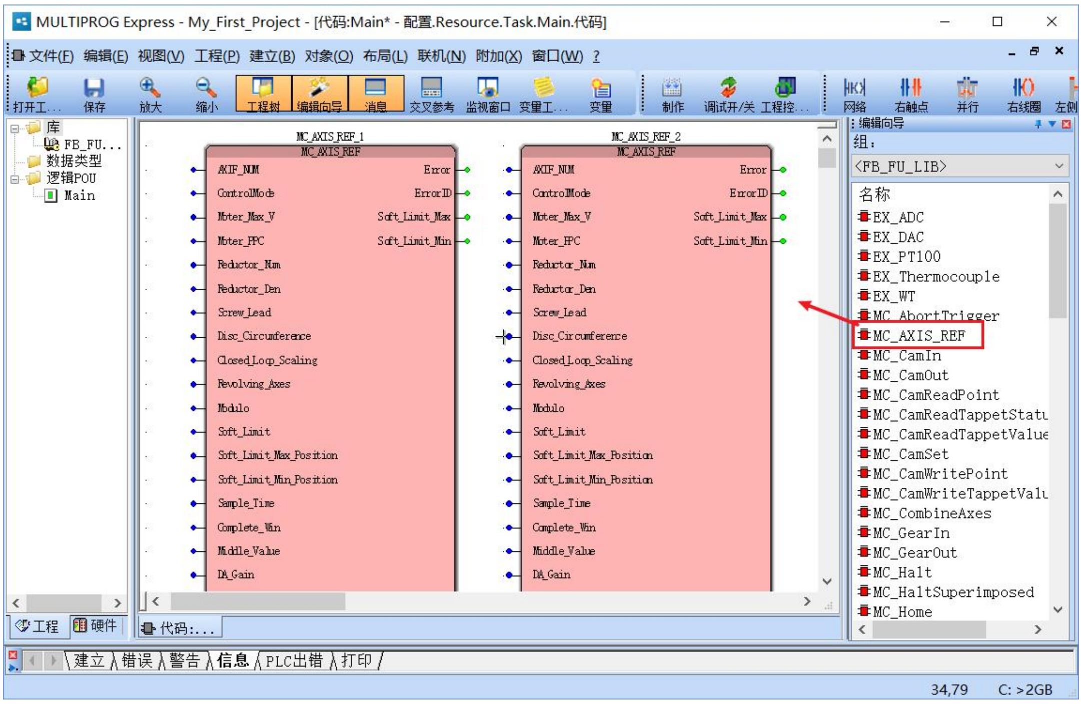
在上述 PC 与运动控制器通讯成功，系统搭建，伺服参数设置等完成后，接下来开始编程控制伺服电机运行和停止；在编程前默认用户已经阅读了《第十一章 运动指令》包括：11.1 插入 FB_FU_LIB 运动控制库、11.2 运动指令 、11.3 运动控制指令基本知识和 11.4 单轴指令。

（1）工程创建以及配置

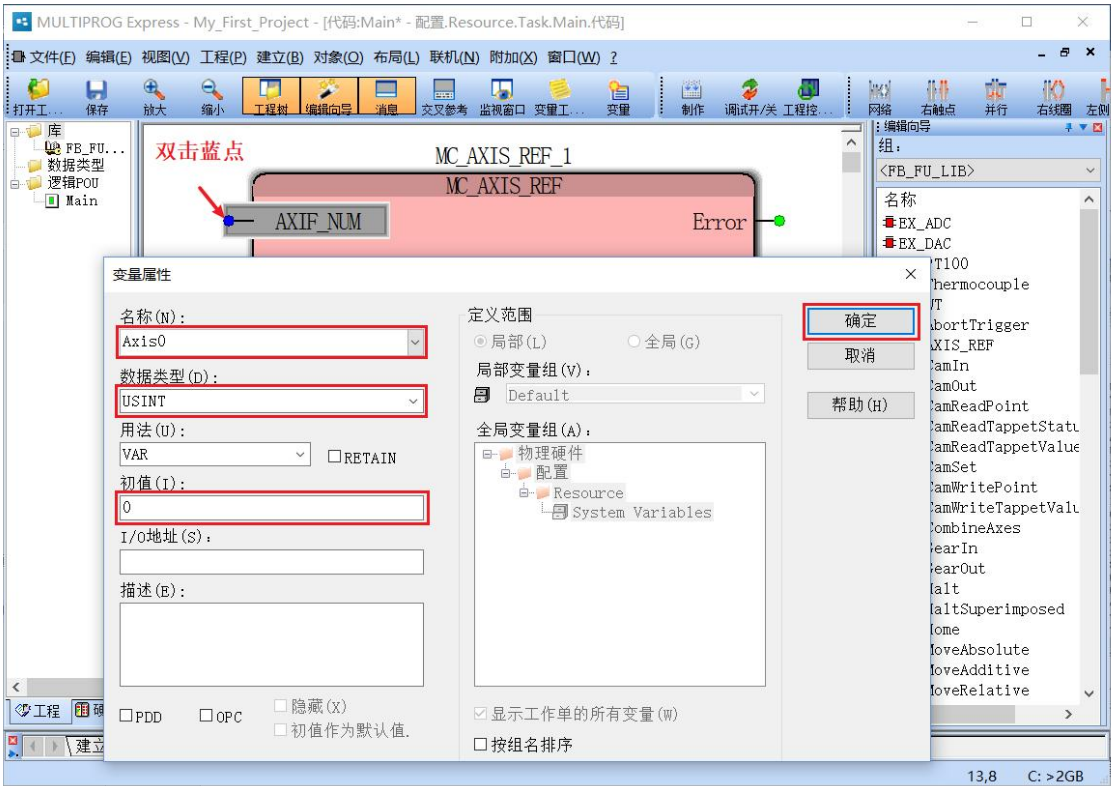
参考《第七章 工程的创建及配置》按照步骤完成：7.1 创建工程项目→ 7.3 实物通讯参数设置 → 7.4 IO 配置，在此不再重复赘述。

（2）编写程序

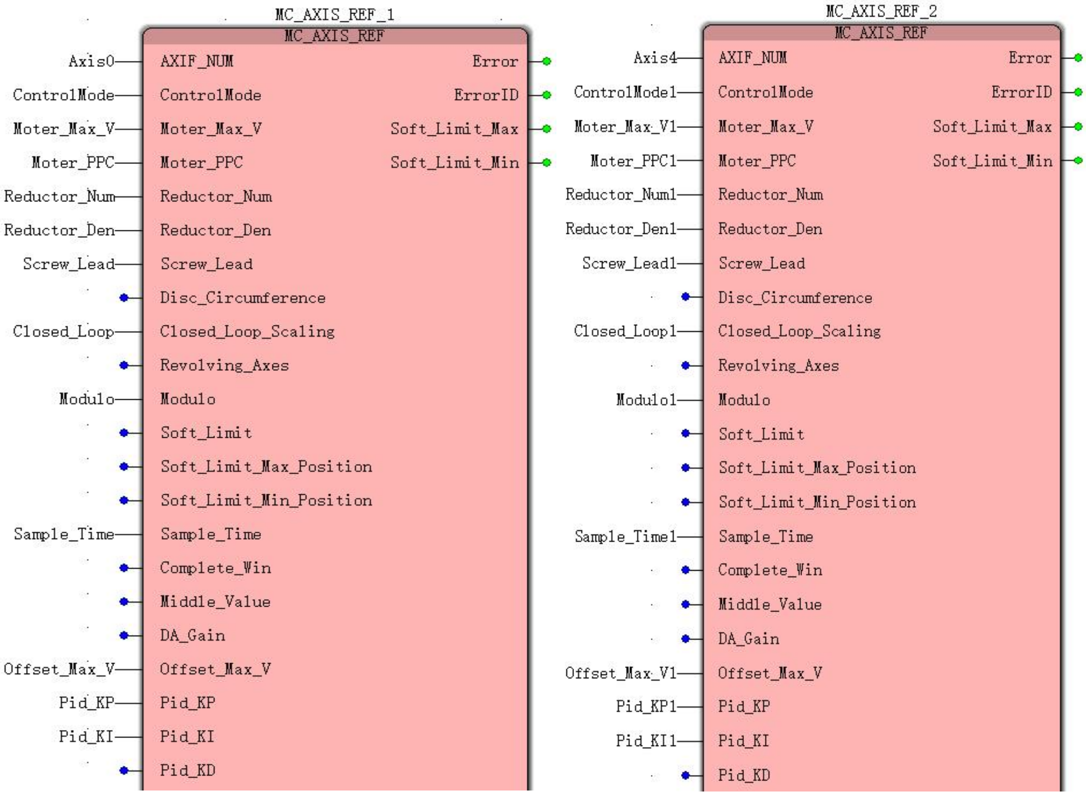
第一步：进入到编程界面，在 FB_FU_LI 运动控制库中选中“MC_AXIS_REF”块，按住鼠标左键拖拽到编程界面，然后松手，此时会弹出模块的属性，需要为模块命名，一般保持默认即可，单击“确定”，如下图所示，插入两个“MC_AXIS_REF”块；



第二步：双击模块输入引脚（蓝色点），此时弹出“变量属性”框，为其定义变量名，数据类型，用法，初值等，如图所示；



用户可以只需填写轴参数不可缺省的参数即可，可根据功能需要添加设置，参考 11.3.3 MC_AXIS_REF（轴参数设定），此处范例添加完成后，如下图所示；

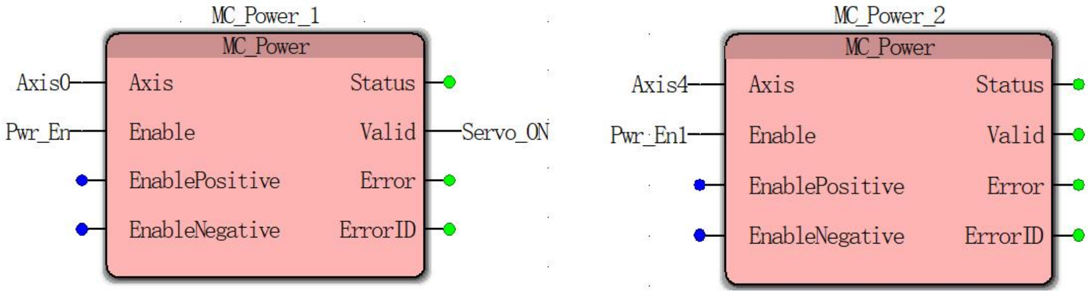


变量名称和属性

变量名	数据类型	初始值
MC_AXIS_REF_1	MC_AXIS_REF	
Axis0	USINT	0
ControlMode	INT	1
Moter_Max_V	DINT	3000
Moter_PPC	DINT	10000
Reductor_Num	LREAL	1.0
Reductor_Den	LREAL	1.0
Screw_Lead	LREAL	60.0
Closed_Loop	LREAL	1.0
Modulo	LREAL	360.0
Sample_Time	WORD	20
Offset_Max_V	DINT	200
Pid_KP	DINT	80
Pid_KI	DINT	0
MC_AXIS_REF_2	MC_AXIS_REF	
Axis4	USINT	4
ControlMode1	INT	1
Moter_Max_V1	DINT	3000
Moter_PPC1	DINT	10000
Reductor_Num1	LREAL	1.0
Reductor_Den1	LREAL	1.0

Screw_Lead1	LREAL	60.0
Closed_Loop1	LREAL	1.0
Modulo1	LREAL	360.0
Sample_Time1	WORD	20
Offset_Max_V1	DINT	200
Pid_KP1	DINT	80
Pid_KI1	DINT	0

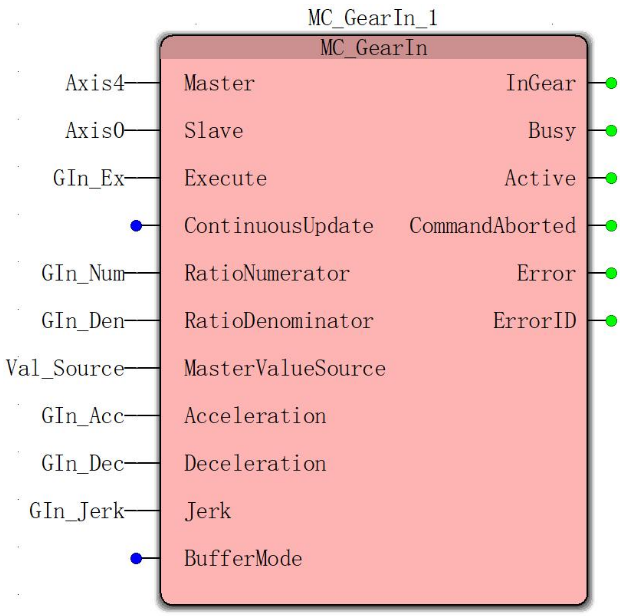
第三步：同理添加两个“MC_Power” 指令如下，使能运动控制器，同时伺服轴（AXIS0）通过 Servo_ON（I/O 地址：%QX0.0）输出控制伺服驱动器的使能，参考 11.4.1 MC_Power（使能指令）；



变量名称和属性

变量名	数据类型	初始值	地址
MC_Power_1	MC_Power		
Axis0	USINT	0	
Pwr_En	BOOL		
Servo_ON	BOOL		%QX0.0
MC_Power_2	MC_Power		
Axis4	USINT	4	
Pwr_En1	BOOL		

第四步：同理添加“MC_GearIn”（电子齿轮耦合指令），用于控制伺服跟随编码器轴走电子齿轮运动，参考 11.4.2 MC_GearIn（电子齿轮耦合指令）；

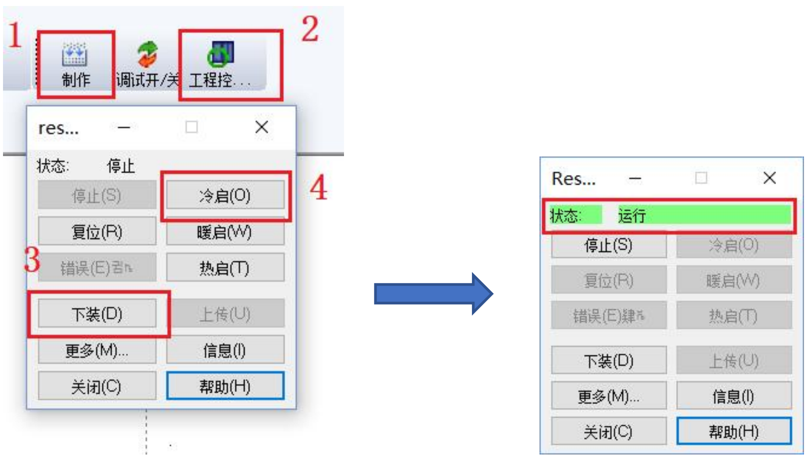


变量名称和属性

变量名	数据类型	初始值
MC_GearIn_1	MC_GearIn	
Axis4	USINT	4
Axis0	USINT	0
GIn_Ex	BOOL	
GIn_Num	LREAL	1.0
GIn_Den	LREAL	1.0
Val_Source	INT	1
GIn_Acc	LREAL	1000.0
GIn_Dec	LREAL	1000.0
GIn_Jerk	LREAL	1000.0

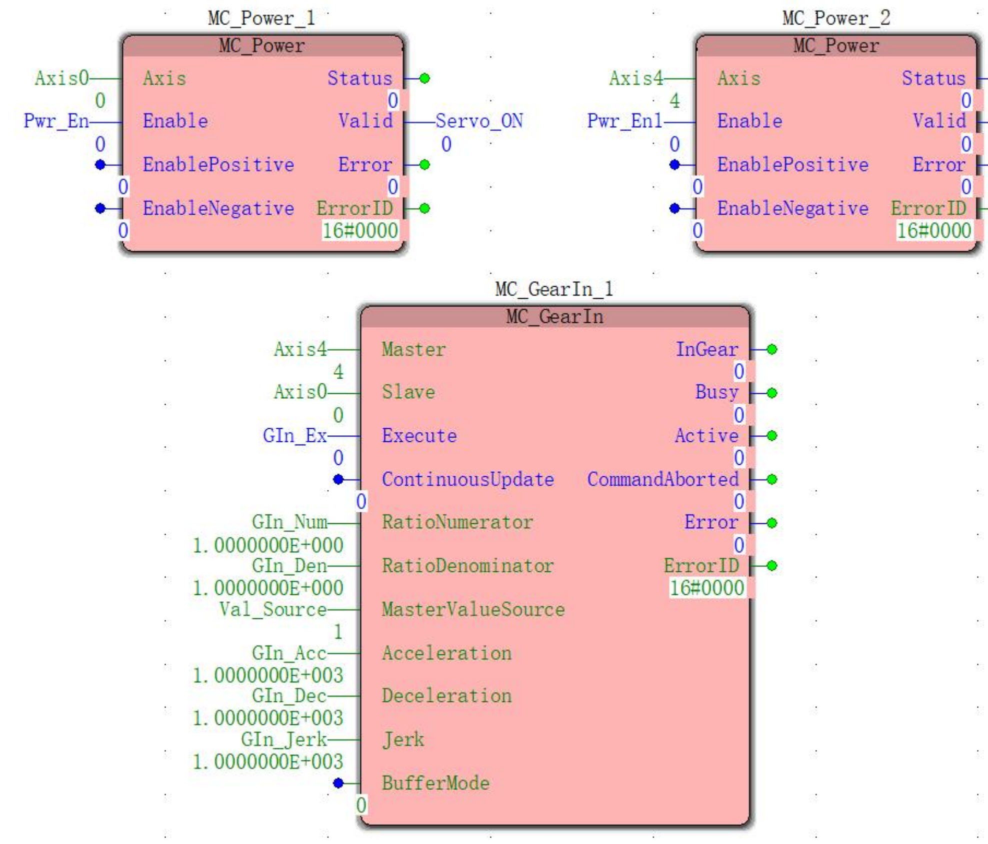
至此，程序编写完成。

第五步：制作下装工程。在工具栏点击制作，确认程序无误后，点击下装程序，然后点击冷启，冷启成功后，状态显示为运行状态。



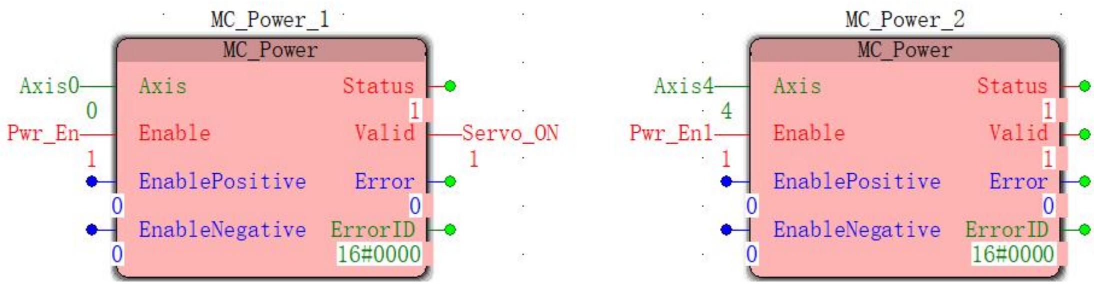
第六步：程序调试。单击工具栏上的在线图标 

可以对程序进行调试和监控，在线监视如下图所示；

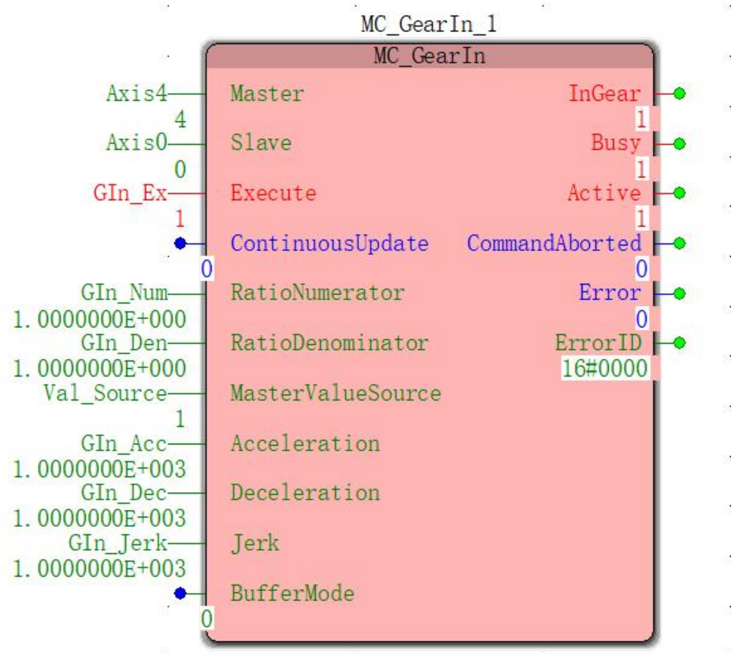


调试 1：双击输入功能位 Pwr_En，弹出调试：Resource 界面，变量值选择 Ture，然后点击覆盖，Pwr_En 将由 False 变为 Ture；同理使 Pwr_En1 值变为 Ture；

当 Servo_ON 由 False 变为 Ture 时，说明运动控制器轴使能成功，并且通过 Q0 输出同时让伺服使能；



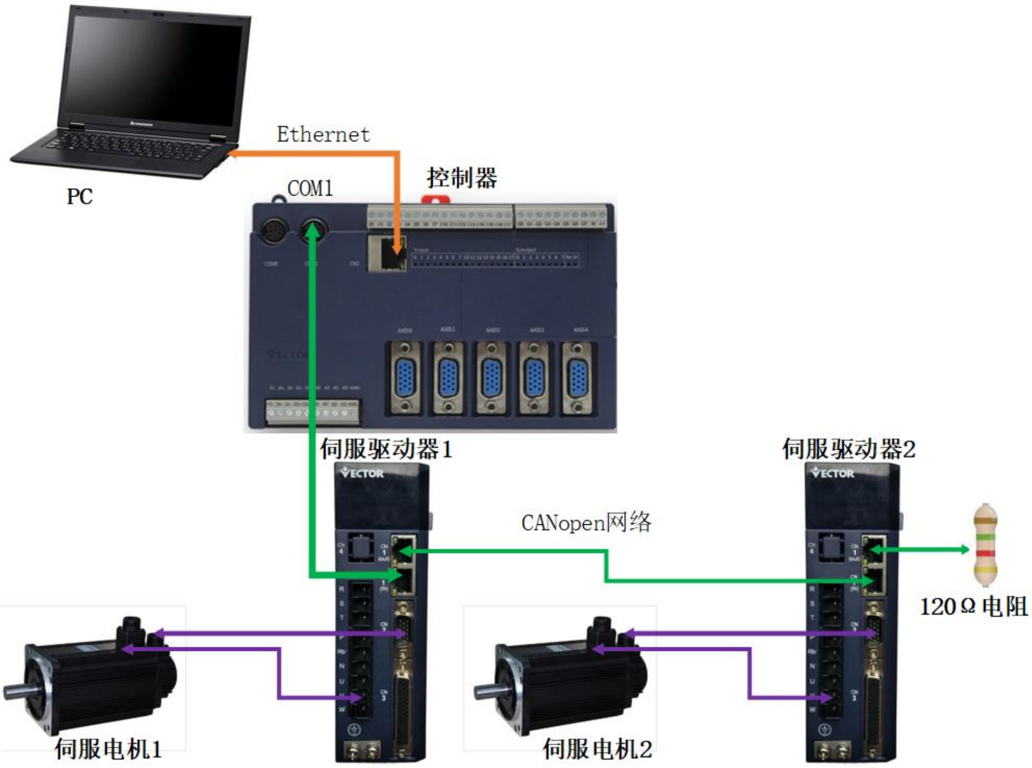
调试 2：同理，双击 GIn_Ex，使其值由 False 变为 Ture，使控制器 AXIS0 与 AXIS4 建立电子齿轮关系，AXIS4 为主轴，AXIS0 为从轴，齿轮比为 1:1；



调试 3：此时，拧动主轴（编码器），从轴（伺服轴）将跟着主轴按照 1:1 的电子齿轮比运动。

9.4 控制器通过 CANopen 通讯方式控制伺服运动（两轴运动）

1： 通讯、控制连接示意图



按照上述系统，运动控制器通过 CANopen 通信方式控制伺服电机运转，电机 1 走速度指令，电机 2 走相对位移指令。

2： 设置伺服驱动器参数

威科达 CANopen 型伺服驱动器 1 参数设置

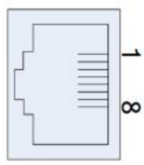
功能号	数值	描述
P08.40	800	CAN 波特率
P08.41	1	CAN 节点号

伺服驱动器 2 参数设置

功能号	数值	描述
P08.40	800	CAN 波特率
P08.41	2	CAN 节点号

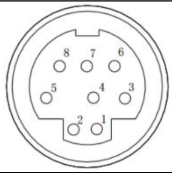
3： CANopen 网络通讯线接法

威科达 CANopen 型伺服驱动器 CN1 部分引脚定义

引脚	信号	描述	
1	CANH	CAN 总线的高信号	
2	CANL	CAN 总线的低信号	
3	GND	电源地	

威科达运动控制器 COM1 口部分引脚定义

引脚	信号	描述
6	CANL	CAN 总线的低信号
7	CANH	CAN 总线的高信号
8	GND	电源地



CANopen 网络通讯线连接

运动控制器(COM1)			伺服驱动器1(CN1)	
6	CANL	↔	1	CANL
7	CANH	↔	2	CANH
8	GND	↔	3	GND

伺服驱动器1(CN1)			伺服驱动器2(CN1)	
1	CANL	↔	1	CANL
2	CANH	↔	2	CANH
3	GND	↔	3	GND

伺服驱动器2(CN1)			
1	CANL	↔	120Ω电阻
2	CANH	↔	120Ω电阻
3	GND		

说明：

1、CANopen 通讯方式下，程序使用 MC_Power（使能指令)使能模块的同时，也会通过通信将伺服驱动器使能，因此无需另外接输出点来控制伺服驱动器使能；

2、为了增强 CANopen 通讯的稳定性，CANopen 总线网络的终端需接入 120Ω 的终端电阻。

4：　程序的创建配置与调试

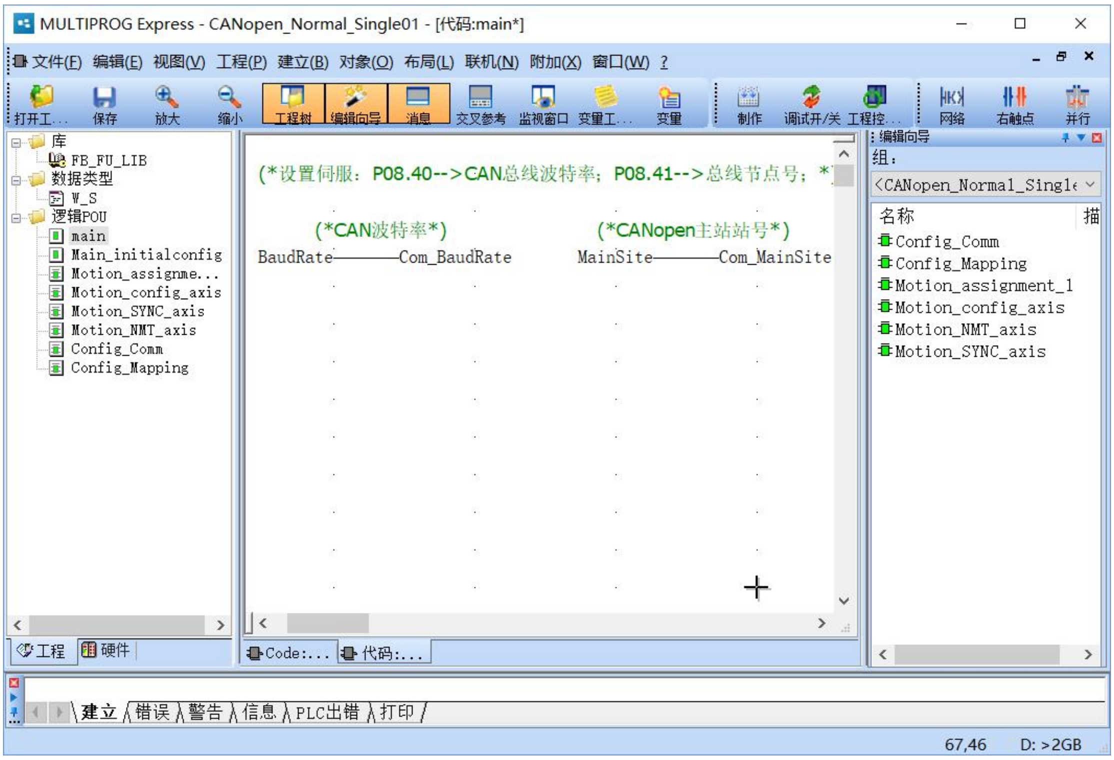
须知

在上述 PC 与运动控制器通讯成功，系统搭建，伺服参数设置等完成后，接下来开始编程控制伺服电机运行和停止；在编程前默认用户已经阅读了“《第十一章　运动指令》包括：11.1　插入 FB_FU_LIB 运动控制库、11.2 运动指令 、11.3 运动控制指令基本知识和 11.4 单轴指令”。

（1）工程创建以及配置

为了方便使用，我司已为用户配备了 CANopen 通讯配置的模版工程，用户可前往官网下载后直接在模版工程的基础上进行编程。（该模版默认配置了一个轴，最多可配置 16 个轴，可根据需要在配置模版中增添）

下载好模版后，解压打开，如下图所示，参考 第七章　工程的创建及配置 完成 PC 与运动控制器的通讯，并参考 11.1　插入 FB_FU_LIB 运动控制库完成库的添加，在此不再重复赘述。

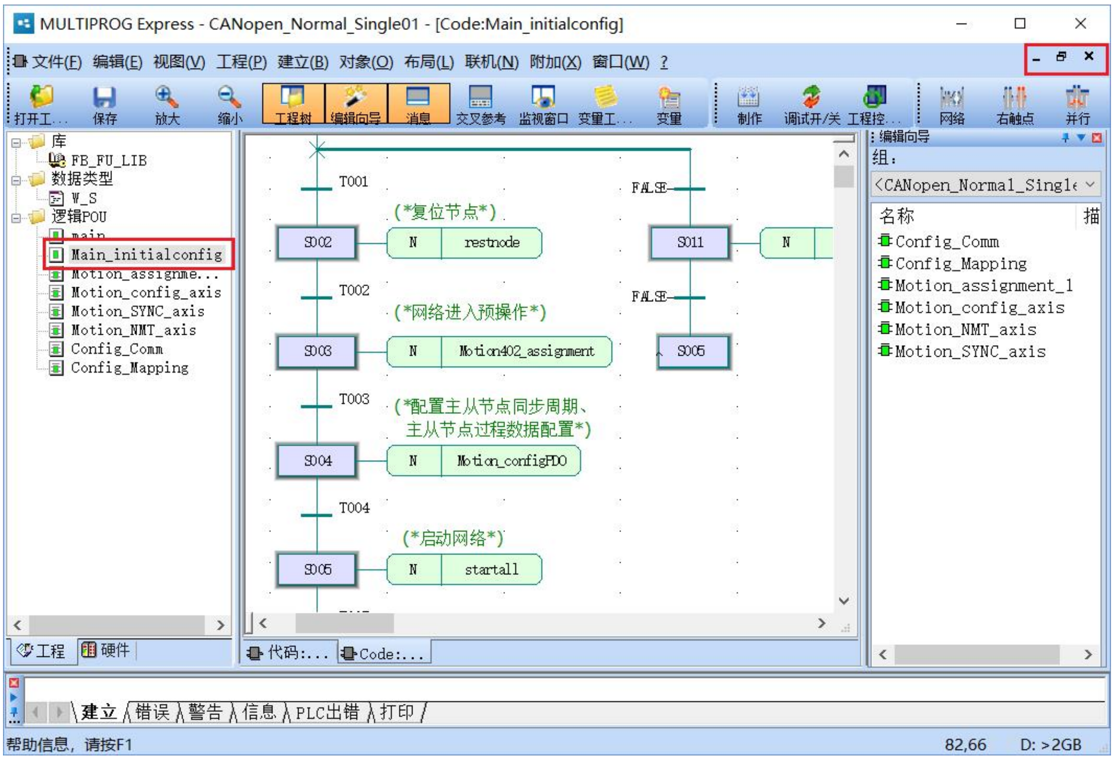


说明：模版默认 CANopen 主站站号为 18，CAN 总线波特率为 800（对应伺服驱动器 P08.40 设置的 CAN 总线波特率），可通过修改 BaudRate 的初值进行修改。

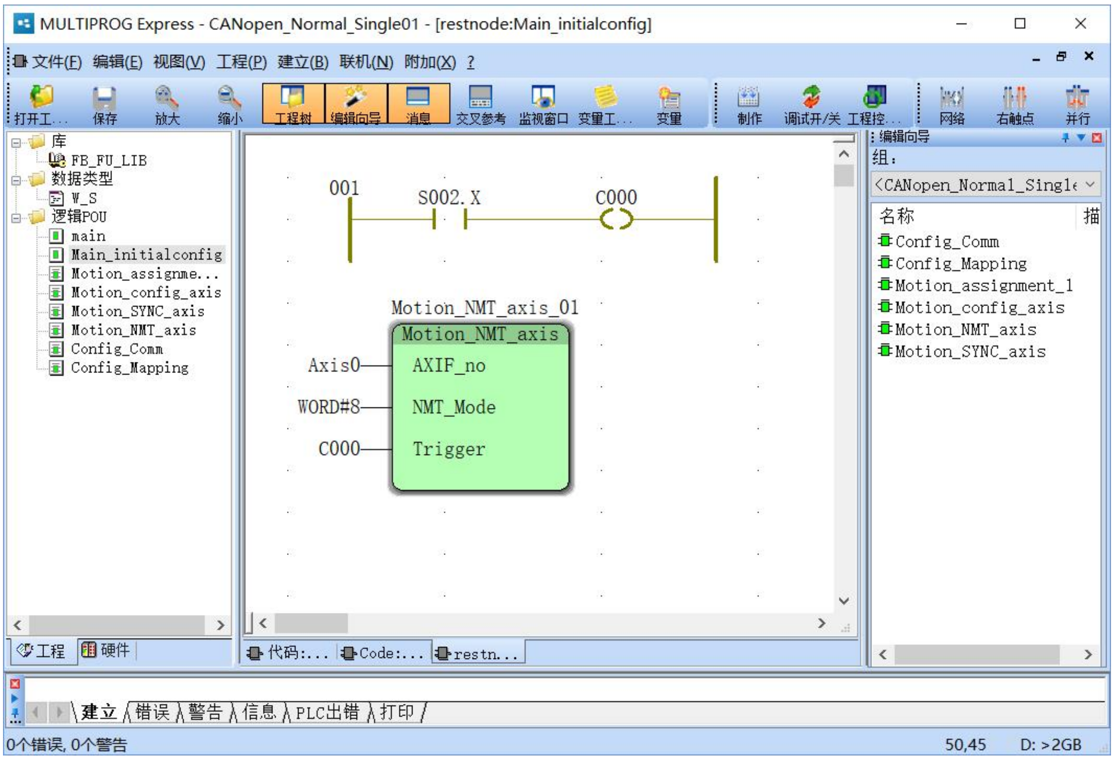
（2）节点配置

第一步：由于模版默认配置一个轴（节点号为 1），而此案例中用到了两个轴，因此我们需要在配置模版中手动再添加一个轴（节点号设定为 2）。双击工程树中“逻辑 POU”下“Main_initialconfig”，打开 CANopen 通讯的配置流程程序（需要关闭时可点击右上角窗口管理按键关闭），如下图

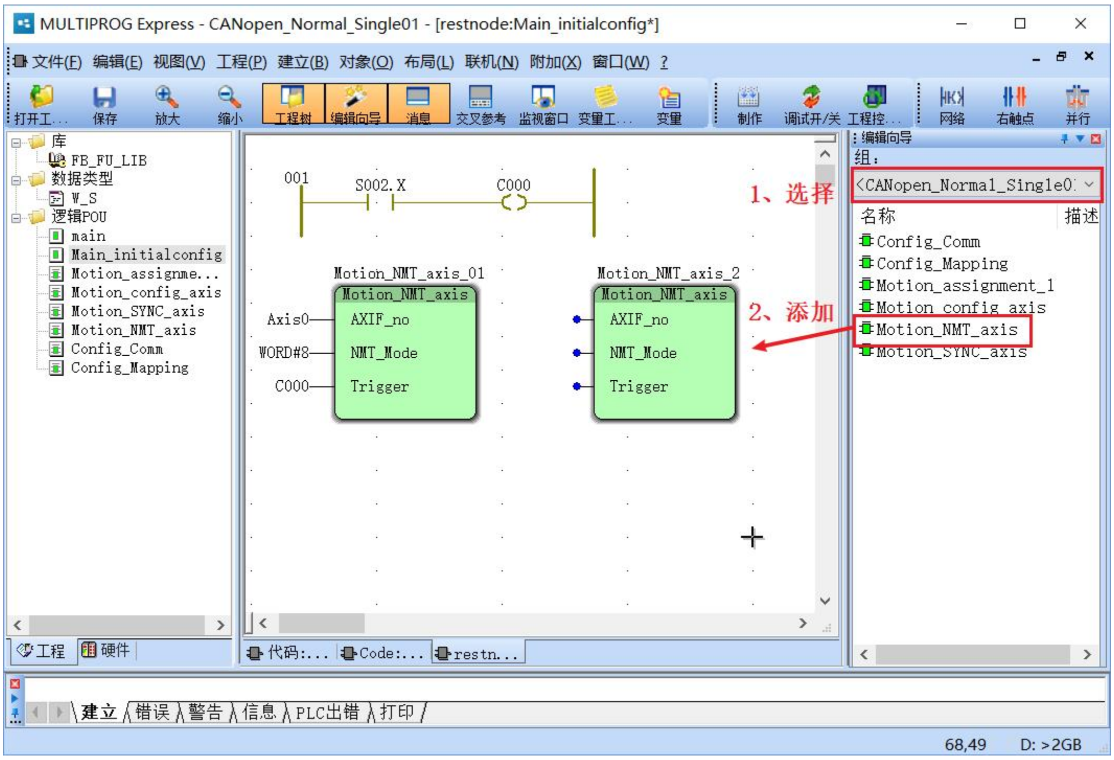
图中可以看到配置的主要步骤包括：主从节点复位（Restnode）、主从节点进入预操作模式（Motion402_assignment）、配置主从节点同步周期和主从节点过程数据配置（Motion_configPDO）、启动总线（Startall），详细步骤定义参考 附件四 CANopen 相关说明；



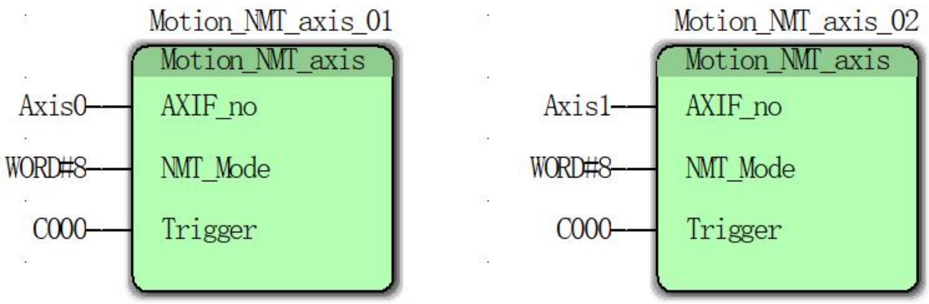
第二步：双击“restnode”，打开复位节点流程，可看到流程下有经过二次封装后的模块“Motion_NMT_axis”，如下图，其输入功能位定义参考附件四 4.1.2 主从节点复位，轴号 Axis0 初值为 0，其作用为对 CANopen 网络下的主站以及节点 1（节点号=轴号+1）进行一次复位操作，因此，我们还需要添加一个该模块，对节点 2 进行复位操作；



第三步：在用户自定义库中，找到“Motion_NMT_axis”块，按住鼠标左键拖拽到编程界面，然后松手，此时会弹出模块的属性，需要为模块命名，一般保持默认即可，单击“确定”，如下图；



第四步：双击模块输入引脚（蓝色点），此时弹出“变量属性”框，为其定义变量名，数据类型，用法，初值等；输入引脚位填写如下图，除了轴号不同（轴号处填写 Axis1，初值为 1，代表节点 2），其余变量填写与轴 0 相同；

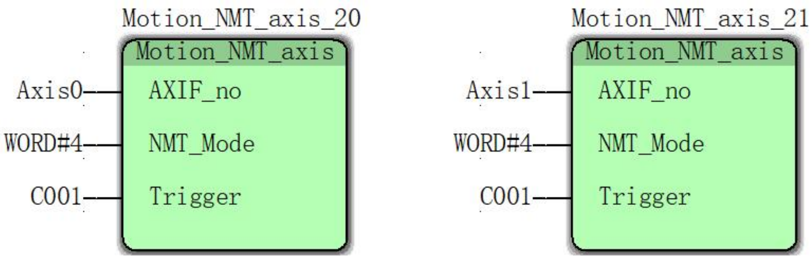


变量名	数据类型	初始值
Motion_NMT_axis_02	Motion_NMT_axis	
Axis1	USINT	1
WORD#8		
C000	BOOL	

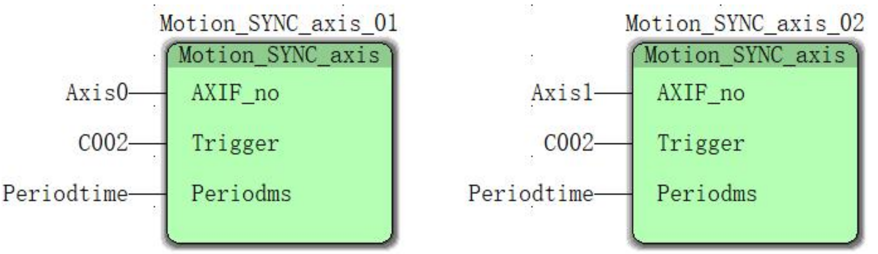
填写完成后，点击关闭该窗口，弹出对话框是否保存，选择“是”，如下图，节点 2 的复位节点配置（resetnode）流程已完成；

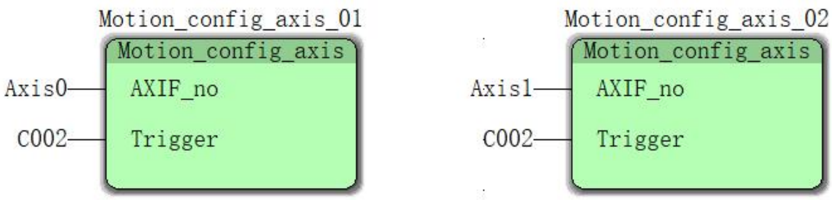
第五步：完成复位节点流程后，回到“Main_initialconfig”，重复第二、三、四步，为节点 2 完成剩余配置，包括：主从节点进入预操作模式（Motion402_assignment）、配置主从节点同步周期和主从节点过程数据配置（（Motion_configPDO））、启动总线（Startall）。添加的模块除了轴号不同（Axis1），其余变量填写与轴 0 相同；

预操作模式（Motion402_assignment）：

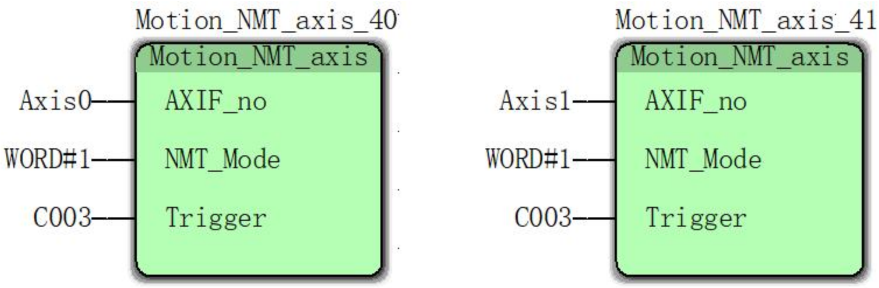


配置主从节点同步周期（Motion_configPDO）：





启动总线（Startall）：

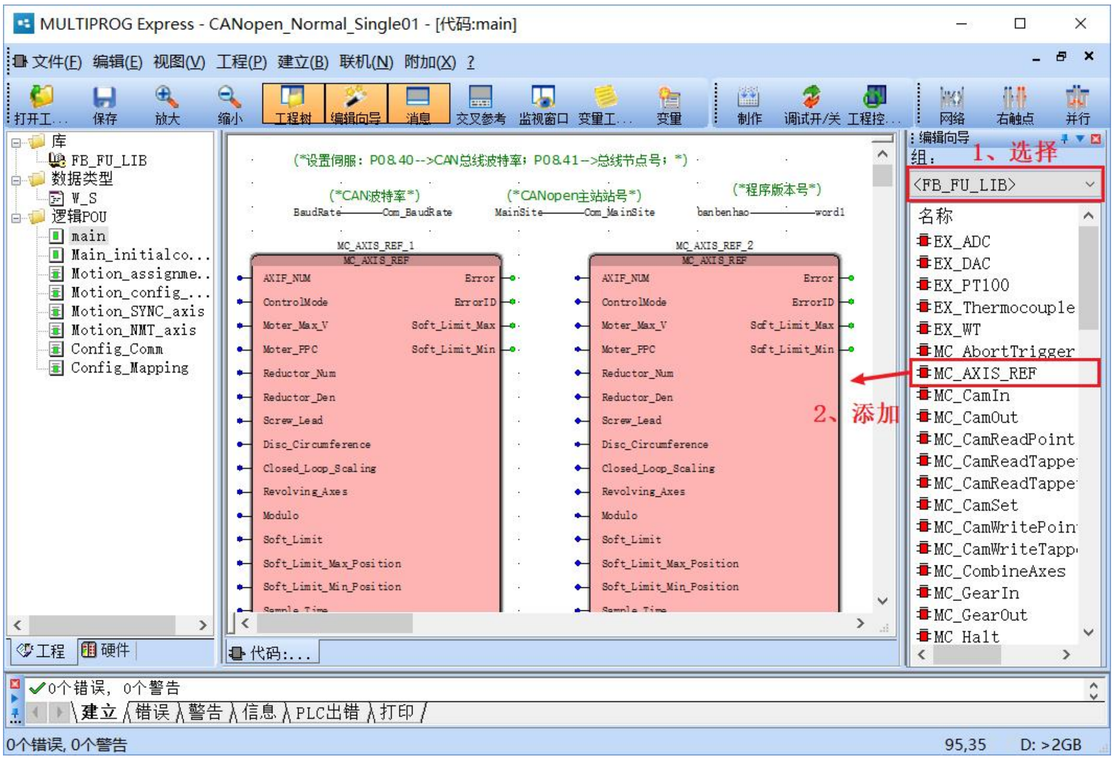


第六步：完成了所有的流程配置编写后，点击制作并确认无误，至此，节点 2（Axis1）的 CANopen 通讯配置已全部完成；

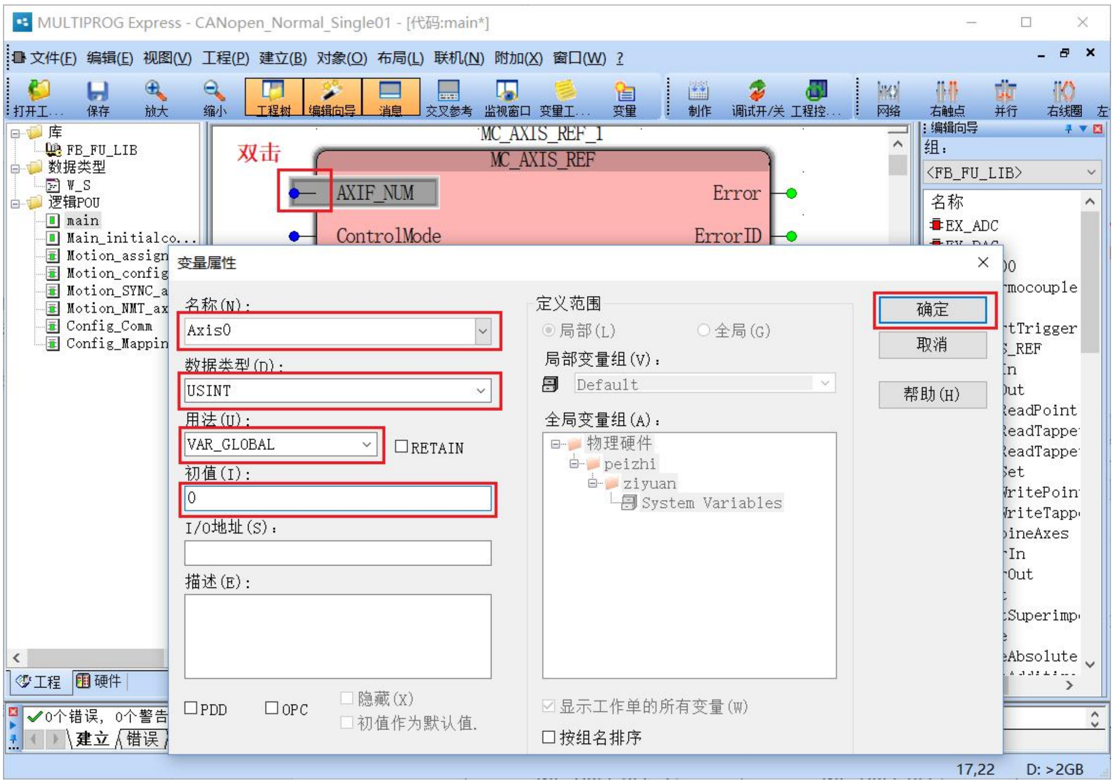
第七步：双击工程树中“逻辑 POU”下的“main”，回到主程序编程界面，便可开始运动程序的编写。

（3）编写程序

第一步：进入到“main”编程界面，在 FB_FU_LI 运动控制库中选中“MC_AXIS_REF”模块，按住鼠标左键拖拽到编程界面，然后松手，此时会弹出模块的属性，需要为模块命名，一般保持默认即可，单击“确定”，如下图所示，插入两个“MC_AXIS_REF”模块；



第二步：双击模块输入引脚（蓝色点），此时弹出“变量属性”框，为其定义变量名，数据类型，用法，初值等，填写完单击确定，如图所示；



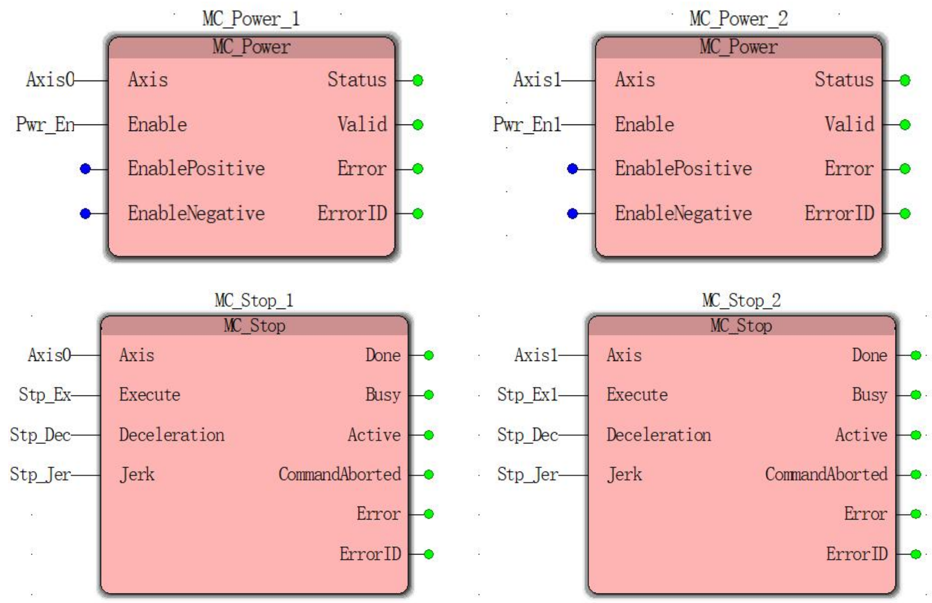
用户可以只需填写轴参数不可缺省的参数即可，可根据功能需要添加设置，参考 11.3.3 MC_AXIS_REF（轴参数设定），此处范例添加完成后，如下图所示：

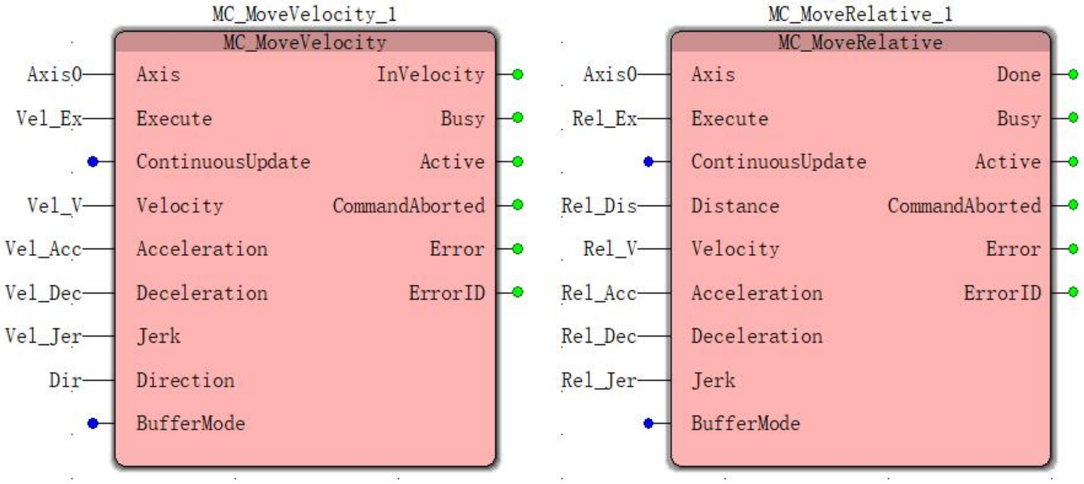
变量名称和属性

变量名	数据类型	初始值
MC_AXIS_REF_1	MC_AXIS_REF	

Axis0	USINT	0
ControlMode	INT	2
Moter_Max_V	DINT	3000
Moter_PPC	DINT	10000
Reductor_Num	LREAL	1.0
Reductor_Den	LREAL	1.0
Screw_Lead	LREAL	60.0
Closed_Loop	LREAL	1.0
Modulo	LREAL	360.0
Sample_Time	WORD	20
Offset_Max_V	DINT	200
MC_AXIS_REF_2	MC_AXIS_REF	
Axis1	USINT	1
ControlMode1	INT	2
Moter_Max_V1	DINT	3000
Moter_PPC1	DINT	10000
Reductor_Num1	LREAL	1.0
Reductor_Den1	LREAL	1.0
Screw_Lead1	LREAL	60.0
Closed_Loop1	LREAL	1.0
Modulo1	LREAL	360.0
Sample_Time1	WORD	20
Offset_Max_V1	DINT	200

第三步：按照上述方式，分别添加两个“MC_Power”模块、两个“MC_Stop”模块、一个“MC_MoveVelocity”模块、一个“MC_MoveRelative”模块，其变量名称和属性如下图，参考 11.4.1 MC_Power（使能指令）、11.4.2 MC_MoveVelocity（速度指令）、11.4.3 MC_MoveRelative（相对位移指令）、11.4.10 MC_Stop（停止指令）；



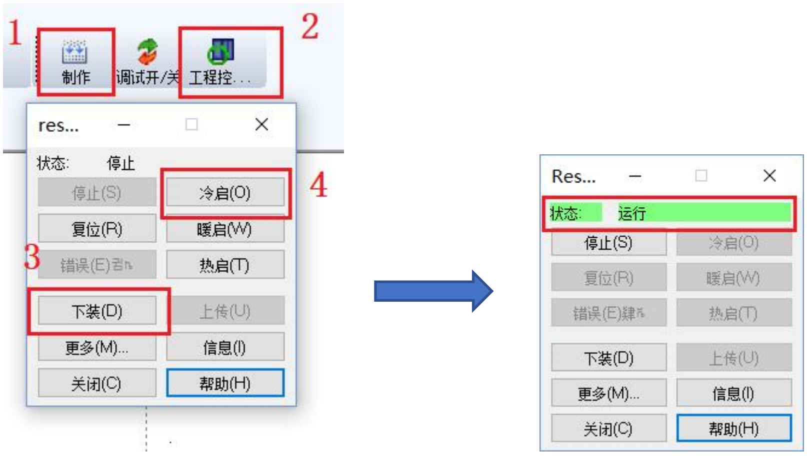


变量名称和属性

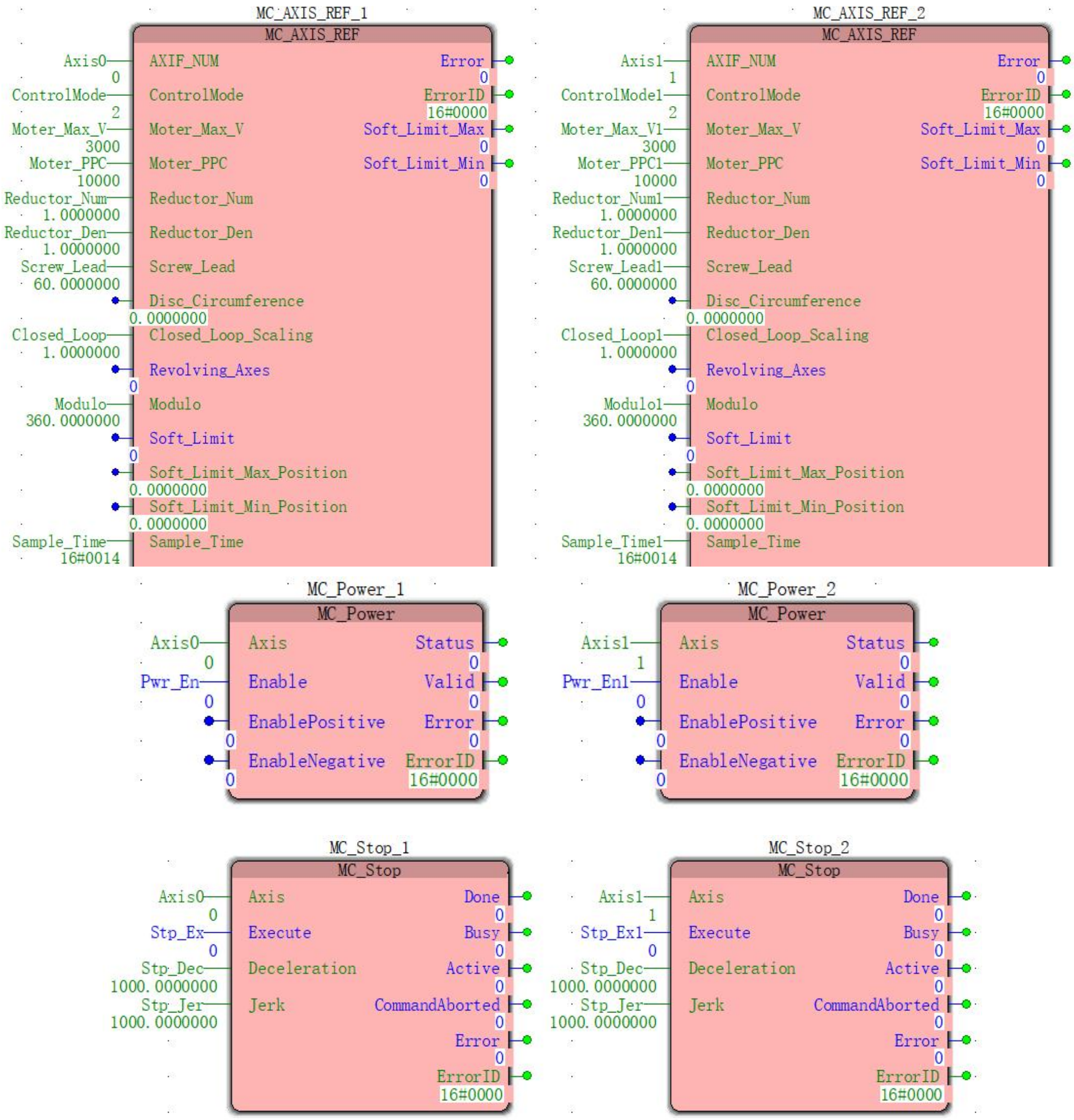
变量名	数据类型	初始值
MC_Power_1	MC_Power	
Pwr_En	BOOL	
MC_Power_2	MC_Power	
Pwr_En1	BOOL	
MC_Stop_1	MC_Stop	
Stp_Ex	BOOL	
Stp_Dec	LREAL	1000.0
Stp_Jer	LREAL	1000.0
MC_Stop_2	MC_Stop	
Stp_Ex1	BOOL	
MC_MoveVelocity_1	MC_MoveVelocity	
Vel_Ex	BOOL	
Vel_V	LREAL	200.0
Vel_Acc	LREAL	1000.0
Vel_Dec	LREAL	1000.0
Vel_Jer	LREAL	1000.0
Dir	INT	1
MC_MoveRelative_1	MC_MoveRelative	
Rel_Ex	BOOL	
Rel_Dis	LREAL	600.0
Rel_V	LREAL	100.0
Rel_Acc	LREAL	1000.0
Rel_Dec	LREAL	1000.0
Rel_Jer	LREAL	1000.0

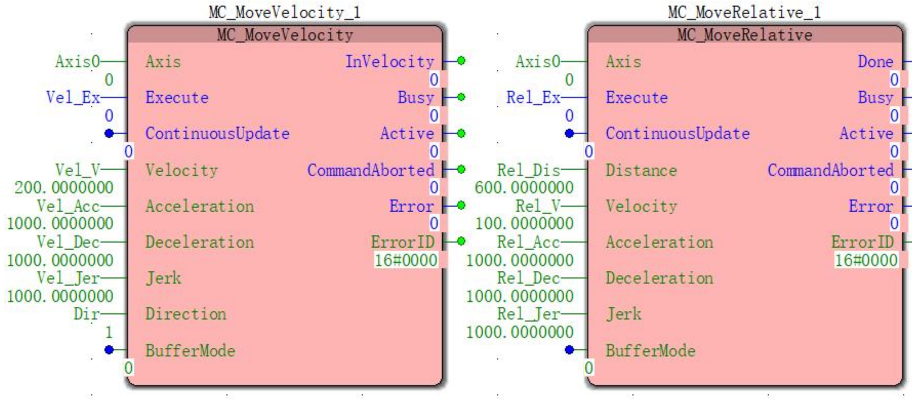
至此，程序编写完成。

第四步：制作下装工程。在工具栏点击制作，确认程序无误后，点击下装程序，然后点击冷启，冷启成功后，状态显示为运行状态；



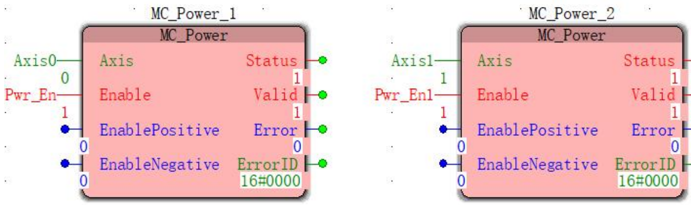
第五步：程序调试。单击工具栏上的在线图标 调试开/关 可以对程序进行调试和监控。在线监视如下图所示



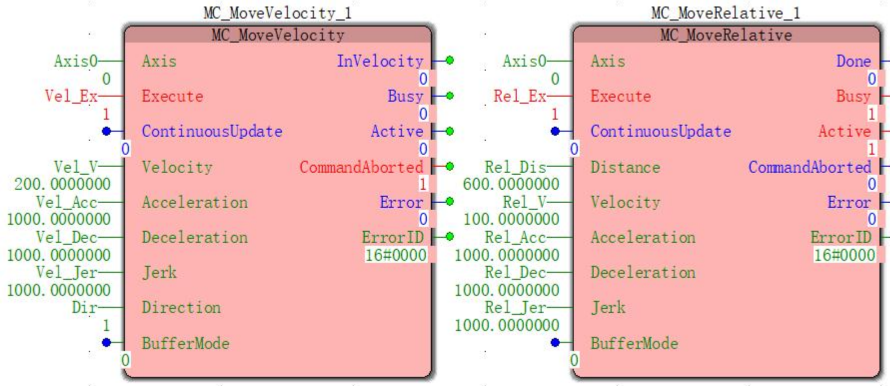


调试 1：双击输入功能位 Pwr_En，弹出调试：Resource 界面，变量值选择 Ture，然后点击覆盖，Pwr_En 将由 False 变为 Ture；同理使 Pwr_En1 值变为 Ture；

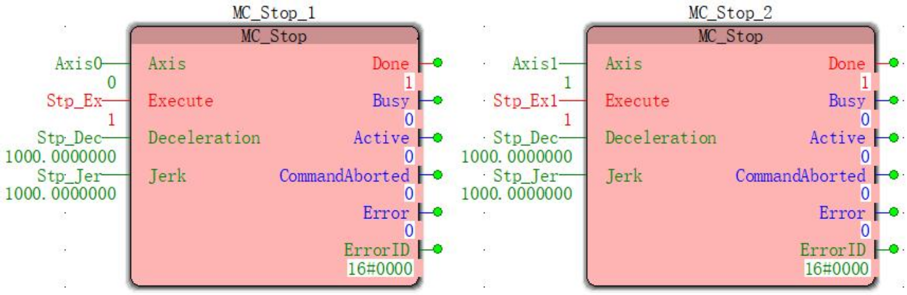
当 Valid 由 False 变为 Ture 时，说明运动控制器轴使能成功，并且通过通信让伺服同时使能，如图所示；



调试 2：同理，分别双击 Vel_Ex、Rel_Ex，使其值由 False 变为 Ture，控制器将控制 Axis0 按照设定的速度走速度模式，Axis1 将按照设定的位移量和速度走相对位移模式，如图所示；



调试 3：此时，双击 Stp_Ex，使其值由 False 变为 Ture，控制器将控制 Axis0 和 Axis1 按照设定的减速度减速，直到停止，如图所示。



第十章 逻辑指令

PLC 指令是封装好了程序块，每个指令能够完成一定的逻辑、运算操作，指令集是 PLC 指令的集合。在 MULTIPROG 编程中，为编程方便，将这些指令被分配到几个不同的功能区（或库）中，在 MULTIPROG 中的编辑向导中可分别罗列出这些功能块，本章将按照这几个不同功能区（或库）的划分，介绍下列指令

✍1：功能
✍2：功能块
✍3：类型转换 FU
✍4：字符串 FU
✍5：位操作函数 BIT_UTIL
✍6：P roConOS 功能

注:

IL 编程语言的指令用法说明中，经常用到 LD 和 ST 操作符，它们的用如下：

LD　IN　（*LD 表示将变量 IN 装入累加器*）
ABS　　（*ABS 表示将累加器的值求绝对值，结果再送累加器*）
ST　OUT　（*ST 表示将累加器的值赋给变量 OUT *）
ST　编程语言的指令用法说明中，“:=”为赋值操作符。

10.1 功能

功能是带有多个输入参数和一个输出参数的程序组织单元 POU，它们没有任何内部存储器，调用带有相同值的功能总是返回相同的结果。返回值是单变量，或多元素变量，如数组或结构。功能的缩写为 FU。

MULTIPROG 编程过程中，可以使用以下功能

▲ 类型转换功能
▲ 数值功能
▲ 算数运算功能
▲ 按位布尔运算功能
▲ 位串功能
▲ 选择运算功能
▲ 比较运算功能
▲ 字符串功能

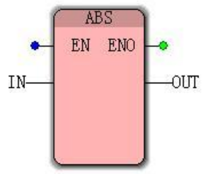
功能中包含的指令（在编辑向导中，从下拉列表选择“功能”）

名称	名称	名称	名称
ABS	DIV_T_R	MAX	ROL
ACOS	EQ	MIN	ROR
ADD	EXP	MOD	SEL
ADD_T_T	EXPT	MOVE	SHL
AND	GE	MUL	SHR

ASIN	GT	MUL_T_AI	SIN
ATAN	LE	MUL_T_AN	SORT
IL 编程语言	LD， FBD 编程语言		
功能：ACOS 指令用于求输入值的反余弦			
COS	LIMIT	MUL_T_R	SUB
DIV	LN	NE	SUB_T_T
DIV_T_AI	LOG	NOT	TAN
DIV_T_AN	LT	OR	XOR

在下面的 LD 和 FBD 的指令说明中，仅当输入引脚 EN 为 1 时，该指令激活，当该指令被顺利执行后，输出引脚 ENO 置 1，否则引脚 ENO 置 0

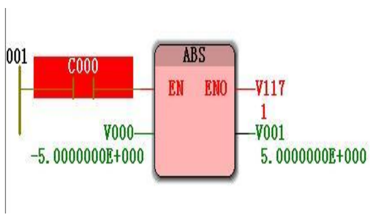
10.1.1 ABS（绝对值指令）

IL 编程语言	LD， FBD 编程语言
功能：ABS 指令用于求负数的绝对值	
LD IN	
ABS	
ST OUT	
ST 编程语言	
OUT:=ABS（IN）	
注：IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

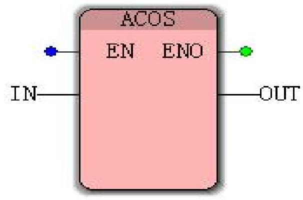
➢ **ABS 指令处理的数据类型**

输入和输出	操作数类型	描述
IN	ANY_NUM	输入
OUT	ANY_NUM	输出

➢ **程序示范**

求-5.0 的绝对值：	描述
	当触点 C000 位 ON 时， 把 V000 值取绝对值后装在 V001 中

10.1.2 ACOS（反余弦指令）

LD IN	
ACOS	
ST OUT	
ST 编程语言	
OUT:=ACOS (IN)	
注：IL,ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

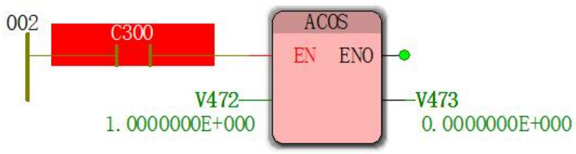
➢ **ACOS 指令处理的数据类型**

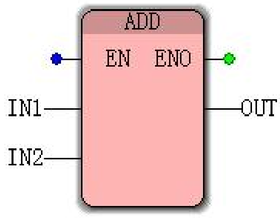
输入和输出	操作数类型	描述
IN	REAL	输入
OUT	REAL	输出，以弧度表示角度

➢ **程序示范**

10.1.3 ADD（加法指令）

IL 编程语言	LD, FBD 编程语言

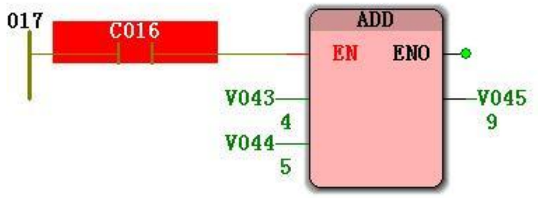
求 1.0 的反余弦的值：	描述
	当触点 C300 为 ON 时，执行该指令，将地址为 V472 的值； 反余弦后，存到地址 V473 中， 执行的结果：ACOS（1.0） = 0.0

功能： ADD 指令用于求两个数据的和	
LD IN1	
ADD IN2	
ST OUT	
ST 编程语言	
OUT:=IN1+IN2	
注：IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1, IN2 和 OUT 或使用常量	

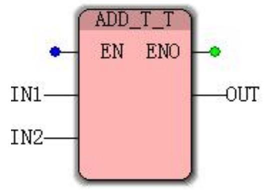
➢ **ADD 指令处理的数据类型**

输入和输出	操作数类型	描述
IN1	ANY_NUM	被加数
IN2	ANY_NUM	加数
OUT	ANY_NUM	和：OUT=IN1+IN2

➤程序示范

<table>
<tr><th>求整数 4 加 5 的值</th><th>描述</th></tr>
<tr><td></td><td>当触点 C016 为 ON 时，执行该指令，两个 16 位的整数进行相加，得到一个 16 位的整数；
执行的结果：VO43+V044=V045</td></tr>
</table>

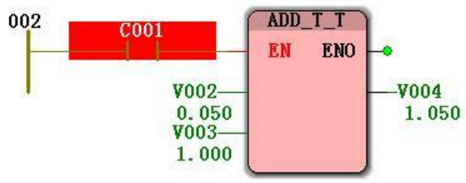
10.1.4 ADD_T_T（时间加法指令）

<table>
<tr><th>IL 编程语言</th><th>LD，FBD 编程语言</th></tr>
<tr><td colspan="2">功能：ADD_T_T 指令用于求两个时间数据的和</td></tr>
<tr><td>LD IN1</td><td rowspan="5"></td></tr>
<tr><td>ADD_T_T IN2</td></tr>
<tr><td>ST OUT</td></tr>
<tr><td>ST 编程语言</td></tr>
<tr><td>OUT:=ADD_T_T（IN1，IN2）</td></tr>
<tr><td colspan="2">注：IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1, IN2 和 OUT 或使用常量</td></tr>
</table>

➤ **ADD_T _T 指令处理的数据类型**

输入和输出	数据类型	描述
IN1	TIME	加数
IN2	TIME	被加数
OUT	TIME	和，OUT=IN1+IN2

➤ **程序示范**

<table>
<tr><th>求 T#1s+T#50ms 的值</th><th>描述</th></tr>
<tr><td></td><td>当 C001 为 ON 时，执行该指令，把量个时间数据类型，进行相加；
执行的结果：　V002+V003=V004</td></tr>
</table>

10.1.5 AND（逻辑与指令）

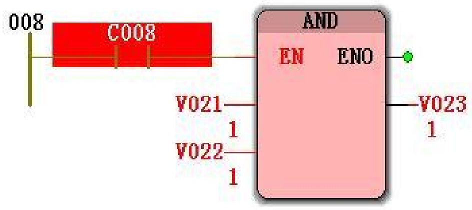
<table>
<tr><th>IL 编程语言</th><th>LD，FBD 编程语言</th></tr>
<tr><td colspan="2">功能：AND 指令用于两个数据的逻辑与运算</td></tr>
<tr><td>LD IN1</td><td rowspan="2"></td></tr>
<tr><td>AND IN2</td></tr>
</table>

ST OUT	
ST 编程语言	
OUT:=IN1&IN2	
注：IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1, IN2 和 OUT 或使用常量	

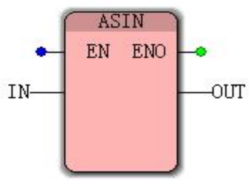
➢ **AND 指令处理的数据类型**

输入和输出	操作数类型	描述
IN1	ANY_BIT	数据 1
IN2	ANY_BIT	数据 2
OUT	ANY_BIT	结果： IN1=0, IN2=0,OUT=0; IN1=0,IN2=1,OUT=0; IN1=1,IN2=0,OUT=0; IN1=1,IN2=1,OUT=1;

➢ **程序示范**

求两个数据的逻辑与的结果	描述
	当触点 C008 位 ON 时，执行该指令，将两个操作数进行相与； 执行的结果：V021&V022=V023

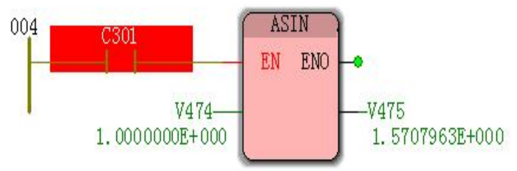
10.1.6 ASIN（反正弦指令）

IL 编程语言	LD, FBD 编程语言
功能：ASIN 指令用于求反正弦	
LD IN	
ASIN	
ST OUT	
ST 编程语言	
OUT:=ASIN（ IN）	
注：IL,ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

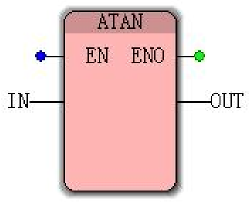
➢ **ASIN 指令处理的数据类型**

输入和输出	数据类型	描述
IN	REAL	输入
OUT	REAL	输出，以弧度表示角度

➢ 程序示范

求 1.0 的反正弦的值	描述
	当触点 C301 为 ON 时，执行该指令，将地址 V474 的值反正弦后，存放在地址 V475 里：执行的结果：ASIN（1.0）=1.57079763…

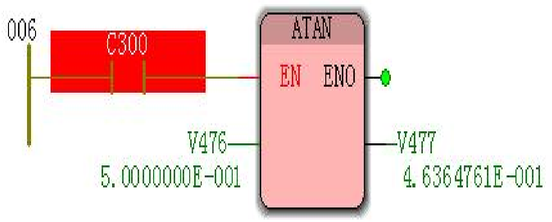
10.1.7 ATAN（反正切指令）

IL 编程语言	**LD， FBD 编程语言**
功能： ATAN 指令用于求反正切	
LD IN	
ATAN	
ST OUT	
ST 编程语言	
OUT:=ATAN（IN）	
注：IL,ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

➢ **ATAN 指令处理的数据类型**

输入和输出	数据类型	描述
IN	*REAL*	输入
OUT	*REAL*	输出，以弧度表示角度

➢ **功能和动作例子**

求 1/2 的反正切的值	**描述**
	当触点 C300 为 ON 时，执行该指令；将地址 V476 的值反正切，存到地址 V477 中：执行的结果：ATAN（1/2）=4.6364761E-001

10.1.8 COS（余弦指令）

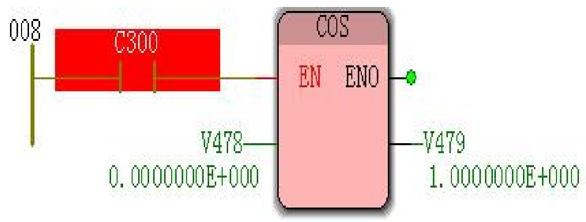
IL 编程语言	**LD， FBD 编程语言**
功能：OS 指令用于求输入值的余弦	
LD IN	
COS	
ST OUT	

ST 编程语言	
OUT:=COS（ IN）	
注：IL,ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

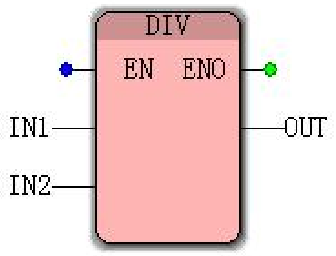
➢ **COS 指令处理的数据类型**

输入和输出	数据类型	描述
IN	REAL	输入，以弧度表示角度
OUT	REAL	输出

➢ **功能和动作例子**

求 0^0 的余弦值	描述
	当触点 C300 为 ON 时,执行该指令；将地址 V478 的值，余弦后存到 V479 中: 执行的结果：COS（0^0）=1.00000

10.1.9 DIV（除法指令）

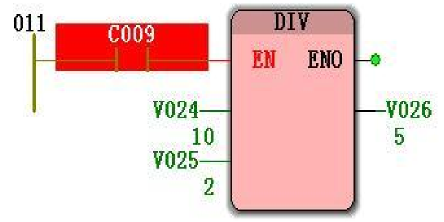
IL 编程语言	LD, FBD 编程语言
功能：DIV 指令用于除法运算	
LD IN1	
DIV IN2	
ST OUT	
ST 编程语言	
OUT:=IN1/IN	
注：IL,ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1, IN2 和 OUT 或使用常量	

➢ **DIV 指令处理的数据类型**

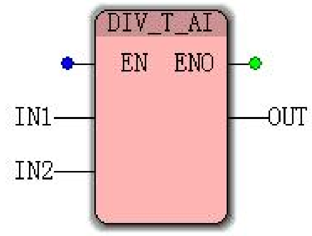
输入和输出	操作数类型	描述
IN1	ANY_NUM	被除数
IN2	ANY_NUM	除数
OUT	ANY_NUM	商

➢ 程序示范

求整数 10 除以 2 的商	描述

	当 C009 为 ON 时，执行该指令，V024 的除以 V025 的商放在 V026 中： 执行的结果：　V024/V025=V026

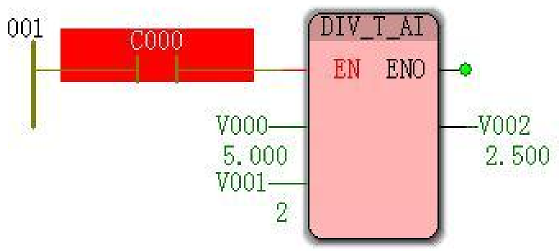
10.1.10 DIV_T_AI（除法（时间除以整数）指令）

IL 编程语言	LD，　FBD 编程语言
功能：DIV_T_AI 指令用于时间除以整数运算	
LD IN1	
DIV_T_AI IN2	
ST OUT	
ST 编程语言	
OUT:=DIV_T_AI（IN1，IN2）	
注：IL,ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1, IN2 和 OUT 或使用常量	

➢ **DIV_T_AI 指令处理数据类型**

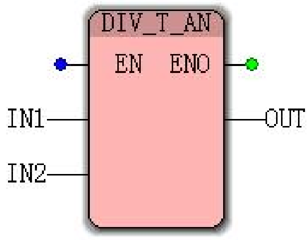
输入和输出	数据类型	描述
IN1	TIME	被除数
IN2	ANY_INT	除数
OUT	TIME	商

➢ **程序示范**

求时间 5s 除以整数 2 的商；	描述
	当触点 C000 位 ON 时，执行该指令，将地址 V000 的值除以地址 V001 的值，存放在 V002 中， 执行的结果：V000/V001=V002

10.1.11 DIV_T_AN（除法（时间除以整数、实数）指令）

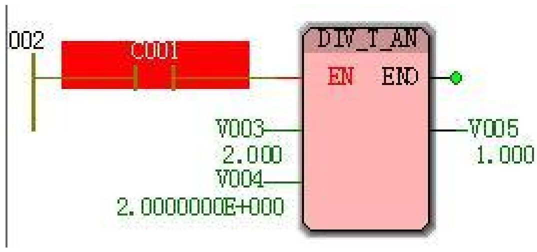
IL 编程语言	LD，　FBD 编程语言
功能：DIV_T_AN 指令用于时间除以整数或实数运算	

<table>
<tr><td>LD IN1</td><td rowspan="5">
</td></tr>
<tr><td>DIV_T_AN IN2</td></tr>
<tr><td>ST OUT</td></tr>
<tr><td>ST 编程语言</td></tr>
<tr><td>OUT:=DIV_T_AN（ IN1，IN2）</td></tr>
<tr><td colspan="2">注：IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1, IN2 和 OUT 或使用常量</td></tr>
</table>

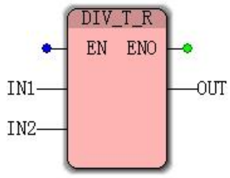
➢ **DIV_T_AN 指令处理的数据类型**

输入和输出	操作数类型	描述
IN1	TIME	被除数
IN2	ANY_NUM	除数
OUT	TIME	商

➢ **程序示范**

求时间 2s 除以整数 2 的商；	描述
	当触点 C001 为 ON，执行该指令，地址 V003 的值除以地址 V004 的商，存放在 V005 地址中； 执行的结果：V003/V004=V005

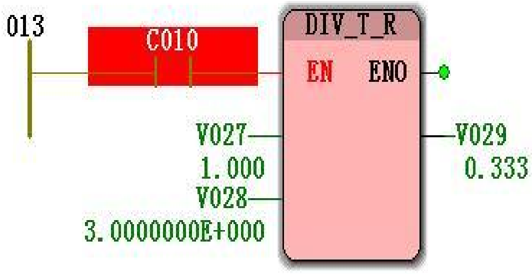
10.1.12 DIV_T_ R（除法（时间除以实数）指令）

<table>
<tr><th>IL 编程语言</th><th>LD, FBD 编程语言</th></tr>
<tr><td colspan="2">功能：DIV_T_R 指令用于时间除以实数运算。</td></tr>
<tr><td>LD IN1</td><td rowspan="5">
</td></tr>
<tr><td>DIV T R IN2</td></tr>
<tr><td>ST OUT</td></tr>
<tr><td>ST 编程语言</td></tr>
<tr><td>OUT:=DIV_T_R（IN1,IN2）</td></tr>
<tr><td colspan="2">注：IL,ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1, IN2 和 OUT 或使用常量</td></tr>
</table>

➢ **DIV_T _R 指令处理的数据类型**

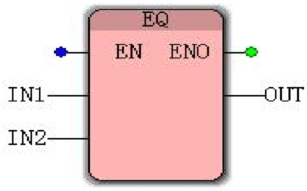
输入和输出	操作数类型	描述
IN1	TIME	被除数
IN2	REAL	除数
OUT	TIME	商

➢ **功能和动作例子**

<table>
<tr><th>求时间 1s 除以浮点数 3.0 的商：</th><th>描述</th></tr>
<tr><td>
</td><td>当触点 C010 为 ON 时执行该指令，地址 V027 的值除以地址 V028 值的商，存放在 V029

执行的结果：V027/V028=V029</td></tr>
</table>

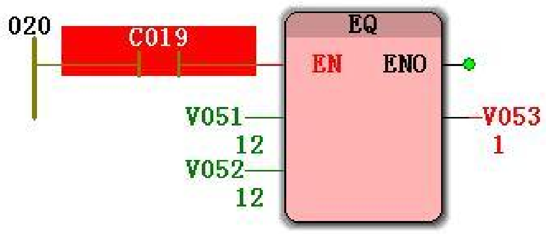
10.1.13 EQ（等于指令）

IL 编程语言	LD， FBD 编程语言
功能：EQ 指令用于判断两个数是否相等	
LD IN1	
EO IN2	
ST OUT	
ST 编程语言	
OUT:=IN1=IN2	
注：IL,ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1, IN2 和 OUT 或使用常量	

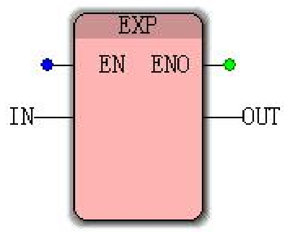
➢ **EQ 指令处理的数据类型**

输入和输出	操作数类型	描述
IN1	ELEMENTARY	数据 1
IN2	ELEMENTARY	数据 2
OUT	BOOL	输出 两数相等，为 TRUE; 两数不相等，为 FALSE

➢ **程序示范**

比较两个地址中的数值；	描述
	当触点 C019 为 ON 时，执行该指令；比较 V051 与 V052 两个地址中值的大小，如果相等则 V053 输出 1。

10.1.14 EXP（自然数 e 的指数函数指令）

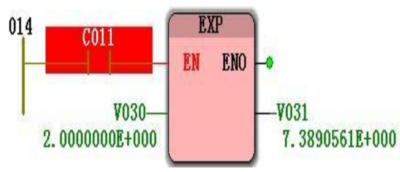
IL 编程语言	LD， FBD 编程语言
功能：指令用于计算自然常数 e 的 x 次方，在这里 x 是输入，其中 e ≈2.718281828 。	
LD IN	
EXP	
ST OUT	
ST 编程语言	
OUT:=EXP（IN）	

注：IL,ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量

➢ **EXP 指令处理的数据类型**

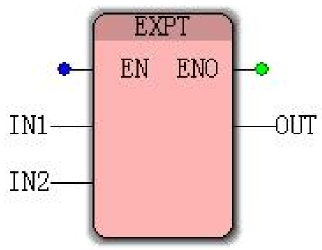
输入和输出	操作数类型	描述
IN	REAL	指数
OUT	REAL	结果，e 的 IN 次方

➢ **功能和动作例子**

<table>
<tr><th>求 e^2 的数值：</th><th>描述</th></tr>
<tr><td>
</td><td>当触点 C011 为 ON 时，执行该指令，其中地址 V030 中的值为 X 次方；

执行的结果：e^{V030} 的值存放在 V031 当中；</td></tr>
</table>

10.1.15 EXPT（幂（x 的 Y 次方）指令）

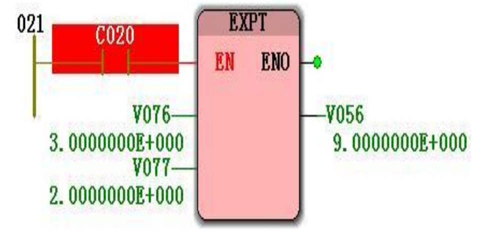
<table>
<tr><th>IL 编程语言</th><th>LD， FBD 编程语言</th></tr>
<tr><td colspan="2">功能：EXPT 指令用于计算 X 的 Y 次方，在这里 X 是第一个输入，Y 是第二个输入。</td></tr>
<tr><td>LD IN1</td><td rowspan="5">
</td></tr>
<tr><td>EXP IN2</td></tr>
<tr><td>ST OUT</td></tr>
<tr><td>ST 编程语言</td></tr>
<tr><td>OUT:=EXPT（ IN1，IN2）</td></tr>
<tr><td colspan="2">注：IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量</td></tr>
</table>

➢ **EXPT 指令处理的数据类型**

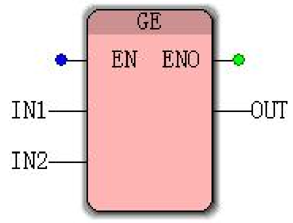
输入和输出	操作数类型	描述
IN1	ANY_REAL	基数
IN2	ANY_NUM	指数
OUT	REAL	结果，IN1 的 IN2 次方

➢ **功能和动作例子**

求 3 的 2 次方的值：	描述

	当触点 C020 为 ON 时，执行该指令，以 V076 为底数，V077 为指数，运算的结果存在 V056; 执行的结果： $V076^{V077}$=V056

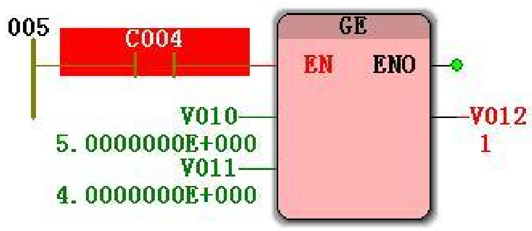
10.1.16 GE（大于等于指令）

IL 编程语言	LD, FBD 编程语言
功能：GE 指令用于比较两个数值的大小，当第一个输入大于或等于第二个时，输出为 1，其他为 0。	
LD IN1 GE IN2 ST OUT ST 编程语言 OUT:=IN1>=IN2;	
注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1, IN2 和 OUT 或使用常量	

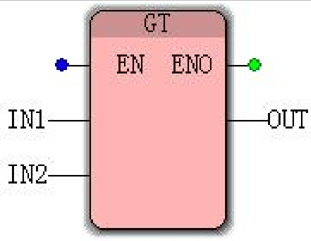
- **GE 指令处理的数据类型**

输入和输出	数据类型	描述
IN1	ANY	第一个输入
IN2	ANY	第二个输入
OUT	BOOL	结果，IN1 >= IN2 时，OUT 为 1

- **程序示范**

比较两个 V010 与 V011 地址中数值的大小：	描述
	当触点 C004 为 ON 时，执行该指令；比较 V010 数值与 V011 数值的大小，当 V010 地址中的数值大于等于 V011 时，输出 V012 为 1； 执行的结果： V010>V011=V012=1

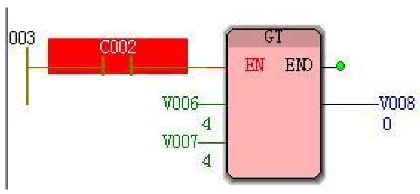
10.1.17 GT（大于指令）

IL 编程语言	LD， FBD 编程语言
功能：GT 指令用于比较两个数值的大小，当第一个输入大于第二 个时，输出为 1，其他为 0。	
LD IN1	
GT IN2	
ST OUT	
ST 编程语言	
OUT:=IN1>IN2	
注：IL,ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1, IN2 和 OUT 或使用常量	

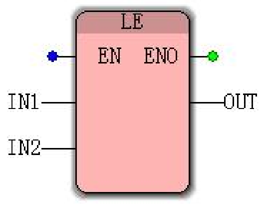
➢ **GT 指令处理的数据类型**

输入和输出	数据类型	描述
IN1	ANY	第一个输入
IN2	ANY	第二个输入
OUT	BOOL	结果，IN1 > IN2 时，OUT 为 1

➢ **程序示范**

比较两个 V010 与 V011 地址中数值的大小：	描述
	当触点 C002 为 ON 时，执行该指令，当地址 V006 的值大于地址 V007 的值，V008 为 V006

10.1.18 LE（小于等于指令）

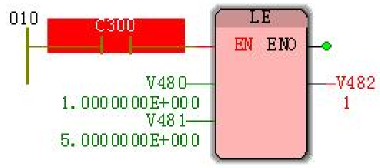
IL 编程语言	LD， FBD 编程语言
功能：LE 指令用于比较两个数值的大小，当第一个输入小于或等于第二个时，输出为 1，其他为 0	
LD IN1	
LE IN2	
ST OUT	
ST 编程语言	
OUT:=IN1<=IN2	
注：IL,ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1, IN2 和 OUT 或使用常量	

➢ **LE 指令处理的数据类型**

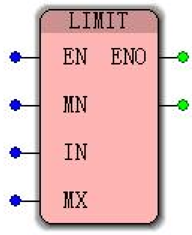
输入和输出	数据类型	描述

IN1	ANY	第一个输入
IN2	ANY	第二个输入
OUT	BOOL	结果 当 IN1<=IN2 时， OUT 为 1 当 IN1>IN2 时，OUT 为 0;

➢ **程序示范**

比较 V480 与 V481 中的值	描述
	当触点 C300 为 ON 时，执行该指令； 当地址 V480 的值小于地址 V481 的值时，V482 输出为 1： 当地址 V480 的值大于等于地址 V481 的值时，V482 输出为 0： 执行的结果：1.0<5.0 输出 1

10.1.19 LIMIT（极限选择指令）

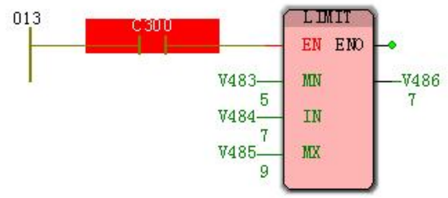
IL 编程语言	LD， FBD 编程语言
功能：LIMIT 指令用于把输入值限定到由最大值和最小值所确定的区间。	
LD IN1	
LIMIT IN，IN2	
ST OUT	
ST 编程语言	
OUT:= LIMIT（IN1，IN, IN2）	
注：IL,ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN, IN1, IN2 和 OUT 或使用常量	

➢ **LIMIT 指令处理的数据类型**

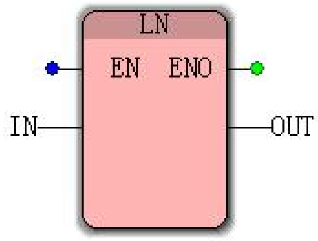
输入和输出	操作数类型	描述
IN1	ANY_INT	最小值
IN	ANY_INT	输入值
IN2	ANY_INT	最大值
OUT	ANY	输出值 当 IN1<=（IN）<=IN2 时， OUT=IN; 当 IN<IN1 时， OUT=IN1; 当 IN>IN2 时， OUT=IN2;

➢ **功能和动作例子**

选择数输出 5~9 之间的整数	描述

	当触点 C300 为 ON 时，执行该指令； 输入值 IN 在 MN~MX 之间时输出 IN, 当 IN 的值大于等于 MX 时输出 MX； 当 IN 小于等于 MN 时输出 MN: 执行的结果： 5<7<9 输出 1

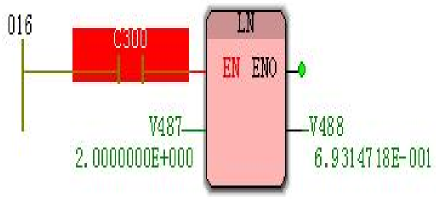
10.1.20 LN（自然对数指令）

<table>
<tr><th>IL 编程语言</th><th>LD, FBD 编程语言</th></tr>
<tr><td colspan="2">功能：LN 指令用于计算输入的自然对数。</td></tr>
<tr><td>LD IN</td><td rowspan="5"></td></tr>
<tr><td>LN</td></tr>
<tr><td>ST OUT</td></tr>
<tr><td>ST 编程语言</td></tr>
<tr><td>OUT:=LN（ IN）</td></tr>
<tr><td colspan="2">注：IL,ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量</td></tr>
</table>

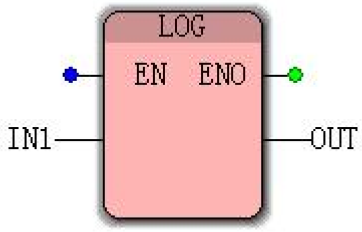
➢ **LN 指令处理的数据类型**

输入和输出	数据类型	描述
IN	REAL	输入值
OUT	REAL	结果，OUT = LOGeIN

➢ **功能和动作例子**

求以 $\log_e{}^2$ 的值	描述
	当触点 C300 为 ON 时，执行该指令，其中 V487 为指数； 执行的结果：$\log_e{}^2$=6.9314718E-001

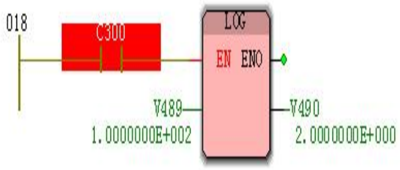
10.1.21 LOG（对数指令）

<table>
<tr><th>IL 编程语言</th><th>LD, FBD 编程语言</th></tr>
<tr><td colspan="2">功能：LOG 指令用于计算输入的以 10 为底的对数。</td></tr>
<tr><td>LD IN</td><td rowspan="5">
</td></tr>
<tr><td>LOG</td></tr>
<tr><td>ST OUT</td></tr>
<tr><td>ST 编程语言</td></tr>
<tr><td>OUT:=LOG（ IN）</td></tr>
<tr><td colspan="2">注：IL,ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量</td></tr>
</table>

➢ **LOG 指令处理的数据类型**

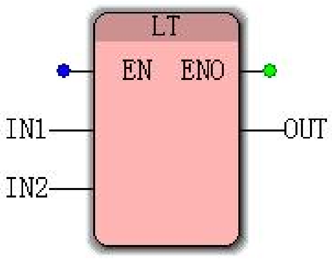
输入和输出	数据类型	描述
IN	REAL	输入值
OUT	REAL	结果，OUT = LOG10IN=lg（IN）

➢ **功能和动作例子**

<table>
<tr><th>求对数 log_{10}^{100} 值</th><th>描述</th></tr>
<tr><td>
</td><td>当触点 C300 为 ON 时，执行该指令；
其中地址 V489 的数值为指数

执行的结果：log_{10}^{100}=2.0</td></tr>
</table>

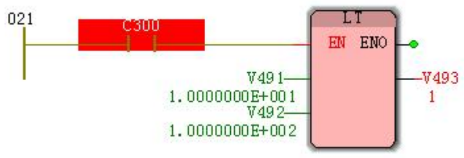
10.1.22 LT（小于指令）

<table>
<tr><th>IL 编程语言</th><th>LD， FBD 编程语言</th></tr>
<tr><td colspan="2">功能：LT 指令用于比较两个数值的大小，当第一个输入小于第二个时，输出为 1，其他为 0</td></tr>
<tr><td>LD IN1</td><td rowspan="5">
</td></tr>
<tr><td>LT IN2</td></tr>
<tr><td>ST OUT</td></tr>
<tr><td>ST 编程语言</td></tr>
<tr><td>OUT:=IN1<IN2</td></tr>
<tr><td colspan="2">注：IL,ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1, IN2 和 OUT 或使用常量</td></tr>
</table>

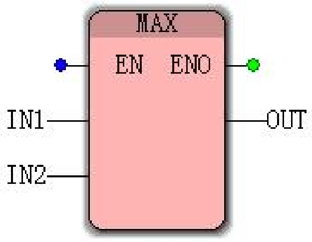
➢ **LT 指令处理的数据类型**

输入和输出	操作数类型	描述
IN1	ANY	第一个输入
IN2	ANY	第二个输入
OUT	BOOL	结果： 当 IN1 < IN2 时，OUT 为 1; 当 IN1 >= IN2 时，OUT 为 0;

➢ 功能和动作例子

比较 V491（10.0）与 V492（100.0）的值	描述
	当触点为 ON 时，执行该指令，当地址 V491 的值小于 V492 的值时，地址 V493 将输出 1 执行的结果：10.0< 100.0 =V493 输出 1

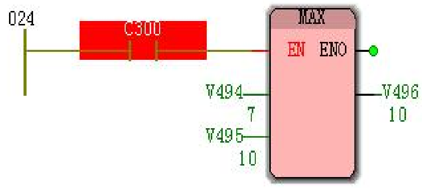
10.1.23 MAX（最大值指令）

IL 编程语言	LD， FBD 编程语言
功能：MAX 指令用于确定两个数值的最大值。	
LD IN1 MAX IN2 ST OUT ST 编程语言 OUT:=MAX（IN1，IN2）	
注：IL,ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1, IN2 和 OUT 或使用常量	

➢ **MAX 指令处理的数据类型**

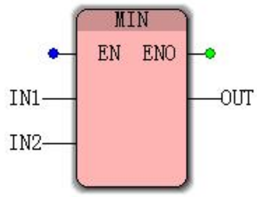
输入和输出	数据类型	描述
IN1	ANY_NUM	第一个输入
IN2	ANY_NUM	第二个输入
OUT	ANY	结果： 当 IN1 <= IN2 时，OUT 为 IN2; 当 IN1 >= IN2 时，OUT 为 IN1;

➢ 功能和动作例子

<table>
<tr><th>整数 7 与 10 输出最大的值</th><th>描述</th></tr>
<tr><td>
</td><td>当触点为 ON 时，执行该指令，比较地址 V494 与 V495 的值，较大者输出存在地址 V496 中：

执行结果：10>7　输出 10</td></tr>
</table>

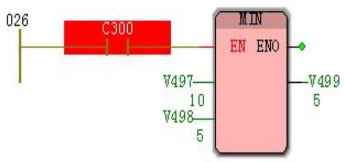
10.1.24 MIN（最小值指令）

<table>
<tr><th>IL 编程语言</th><th>LD，FBD 编程语言</th></tr>
<tr><td colspan="2">功能：指令用于确定两个数值的最小值。</td></tr>
<tr><td>LD IN1</td><td rowspan="5">
</td></tr>
<tr><td>MIN IN2</td></tr>
<tr><td>ST OUT</td></tr>
<tr><td>ST 编程语言</td></tr>
<tr><td>OUT:=MIN（IN1，IN2）</td></tr>
<tr><td colspan="2">注：IL,ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1, IN2 和 OUT 或使用常量</td></tr>
</table>

- **MIN 指令处理的数据类型**

输入和输出	数据类型	描述
IN1	ANY_NUM	第一个输入
IN2	ANY_NUM	第二个输入
OUT	ANY	结果： 当 IN1 <= IN2 时，OUT 为 IN1; 当 IN1 >= IN2 时，OUT 为 IN2;

- **功能和动作例子**

<table>
<tr><th>整数 5 与 10 输出最小的值</th><th>描述</th></tr>
<tr><td>
</td><td>当触点为 ON 时，执行该指令，比较地址 V497 与 V498 的值，较大者输出存在地址 V499 中：

执行结果：5<10　输出 5</td></tr>
</table>

10.1.25 MOD（取模指令）

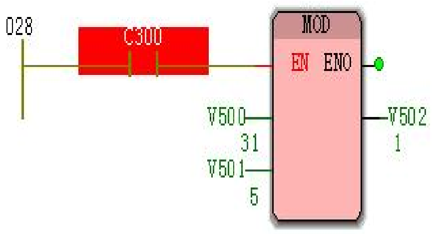
<table>
<tr><th>IL 编程语言</th><th>LD，FBD 编程语言</th></tr>
<tr><td colspan="2">功能：MOD 指令用于确定两个数值相除后的余数。</td></tr>
<tr><td>LD IN1</td><td rowspan="2">
</td></tr>
<tr><td>MOD IN2</td></tr>
</table>

ST OUT	
ST 编程语言	
OUT:=MOD（IN1，IN2）	
注：IL,ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1, IN2 和 OUT 或使用常量	

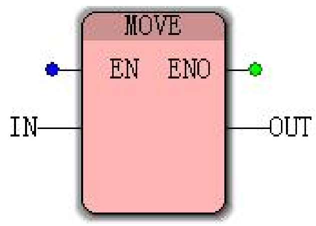
➢ **MOD 指令处理的数据类型**

输入和输出	数据类型	描述
IN1	ANY_INT	被除数
IN2	ANY_INT	除数
OUT	ANY_INT	结果，IN1 除以 IN2 后的余数

➢ **功能和动作例子**

求整数 31 除以 5 的余数的值	描述
	当触点 C300 为 ON 时，执行该指令； 将地址 V500 的值除以 V501 的值，余数存放在地址 V502 中 执行的结果：31/5 余数为 1

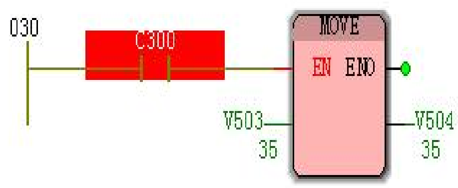
10.1.26 MOVE（赋值指令）

IL 编程语言	LD， FBD 编程语言
功能：MOVE 指令用于将输入值赋给输出值。	
LD IN	
MOVE	
ST OUT	
ST 编程语言	
OUT:=MOVE（IN）	
注：IL,ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

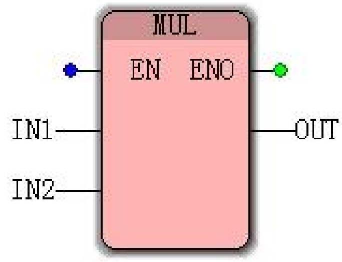
➢ **MOVE 指令处理的数据类型**

输入和输出	数据类型	描述
IN	ANY_NUM	输入值
OUT	ANY_NUM	输出值，OUT=IN

➢ 功能和动作例子

将地址 V503 的值传送到地址 V504	描述
	当触点 C300 为 ON 时，执行该指令，将地址 V503 的值传送到地址 V504 中： 执行的结果： V503 = V504

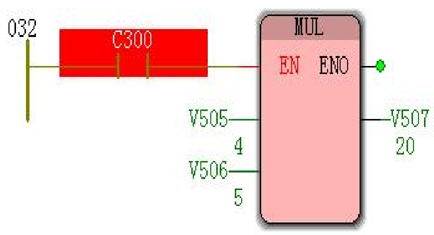
10.1.27 MUL（乘法指令）

IL 编程语言	LD, FBD 编程语言
功能：MUL 指令用于求两个数据的积。	
LD IN1	
MUL IN2	
ST OUT	
ST 编程语言	
OUT :=IN1*IN2	
注：IL,ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1, IN2 和 OUT 或使用常量	

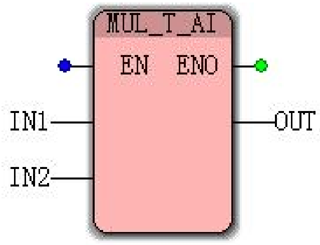
➢ **MUL 指令处理的数据类型**

输入和输出	数据类型	描述
IN1	ANY_NUM	第一个输入
IN2	ANY_NUM	第二个输入
OUT	ANY_NUM	积，OUT=IN1*IN2

➢ 功能和动作例子

求整数 4*5 的值	描述
	当触点 C300 为 ON 时，执行该指令，将地址 V505 的值乘以地址 V506 中的值积存放在 V507： 执行的结果： V505 *V506=V507

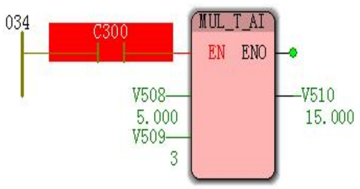
10.1.28 MUL_T_AI（乘法（时间乘以整数）指令）

<table>
<tr><th>IL 编程语言</th><th>LD, FBD 编程语言</th></tr>
<tr><td colspan="2">功能：MUL_T_AI 指令用于计算一个时间型数据与一个整型数据的积。</td></tr>
<tr><td>LD IN1</td><td rowspan="5">
</td></tr>
<tr><td>MUL_T_AI IN2</td></tr>
<tr><td>ST OUT</td></tr>
<tr><td>ST 编程语言</td></tr>
<tr><td>OUT:= MUL_T_AI（IN1，IN2）</td></tr>
<tr><td colspan="2">注：IL,ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1, IN2 和 OUT 或使用常量</td></tr>
</table>

➢ **MUL_T_AI 指令处理的数据类型**

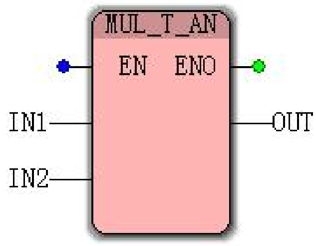
输入和输出	数据类型	描述
IN1	TIME	第一个输入，时间
IN2	ANY_INT	第二个输入，整数
OUT	TIME	积，OUT=IN1*IN2

➢ 功能和动作例子

<table>
<tr><th>求时间 T#5s 乘以整数 3 的值</th><th>描述</th></tr>
<tr><td>
</td><td>当触点 C300 为 ON 时，执行该指令，将地址 V508 的值乘以地址 V509 中的值积存放在 V510：

执行的结果： V508 *V509=V510</td></tr>
</table>

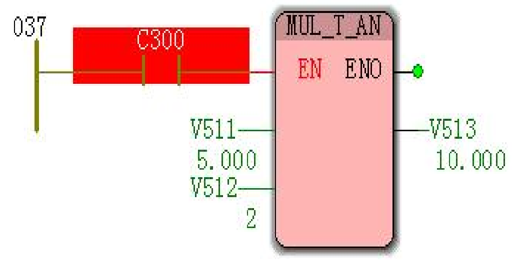
10.1.29 MUL_T_AN（乘法（时间乘以整数、实数）指令）

<table>
<tr><th>IL 编程语言</th><th>LD, FBD 编程语言</th></tr>
<tr><td colspan="2">功能：MUL_T_AN 指令用于计算一个时间型数据与一个整数或浮点数的积。</td></tr>
<tr><td>LD IN1</td><td rowspan="5">
</td></tr>
<tr><td>MUL_T_AN IN2</td></tr>
<tr><td>ST OUT</td></tr>
<tr><td>ST 编程语言</td></tr>
<tr><td>OUT:=MUL_T_AN（IN1，IN2）</td></tr>
<tr><td colspan="2">注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1, IN2 和 OUT 或使用常量</td></tr>
</table>

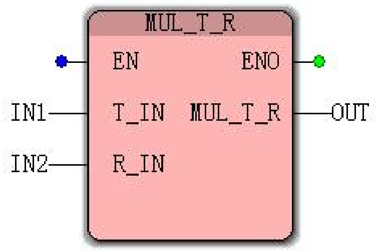
➢ **MUL_T_AN 指令处理的数据类型**

输入和输出	数据类型	描述
IN1	TIME	第一个输入，时间
IN2	ANY_NUM	第二个输入，整数或浮点数
OUT	TIME	积，OUT=IN1*IN2

➢**功能和动作例子**

求时间 T#5s 乘以整数 2 的值	描述
	当触点 C300 为 ON 时，执行该指令，将地址 V511 的值乘以地址 V512 中的值积存放在 V513： 执行的结果： V511 *V512=V513

10.1.30 MUL_T_R（乘法（时间乘以实数）指令）

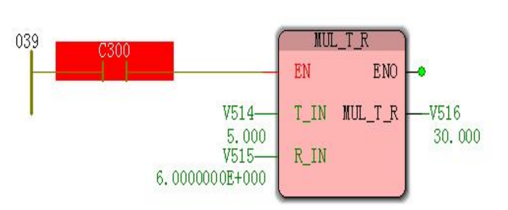
<table>
<tr><td>IL 编程语言</td><td>LD， FBD 编程语言</td></tr>
<tr><td colspan="2">功能：MUL_T_R 指令用于计算一个时间型数据与一个浮点型数据的积</td></tr>
<tr><td>LD IN1</td><td rowspan="5">
</td></tr>
<tr><td>MUL_T_R IN2</td></tr>
<tr><td>ST OUT</td></tr>
<tr><td>ST 编程语言</td></tr>
<tr><td>OUT:=MUL T-R(IN1， IN2)</td></tr>
<tr><td colspan="2">注：IL,ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1, IN2 和 OUT 或使用常量</td></tr>
</table>

➢ **MUL_T_R 指令处理的数据类型**

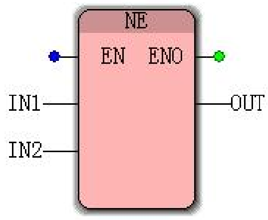
输入和输出	数据类型	描述
IN1 （T_IN）	TIME	第一个输入，时间
IN2（R_IN）	ANY_NUM	第二个输入，浮点数
OUT（MUL_T_R）	TIME	积，OUT=IN1*IN2

➢ **功能和动作例子**

求时间 T#5s 乘以浮点数 6.0 的值	描述

	当触点 C300 为 ON 时，执行该指令，将地址 V514 的值乘以地址 V515 中的值积存放在 V516： 执行的结果： V514 *V515=V516

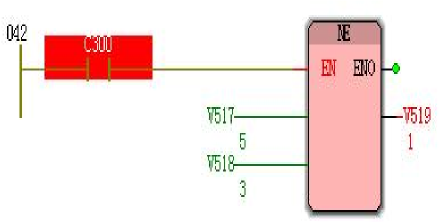
10.1.31 NE（不等于指令）

<table>
<tr><th>IL 编程语言</th><th>LD， FBD 编程语言</th></tr>
<tr><td colspan="2">功能：NE 指令用于判断两个数值的大小关系，当第一个输入不等于第二个时，输出为 1，其他为 0</td></tr>
<tr><td>LD IN1</td><td rowspan="5"></td></tr>
<tr><td>NE IN2</td></tr>
<tr><td>ST OUT</td></tr>
<tr><td>ST 编程语言</td></tr>
<tr><td>OUT:=IN1<>IN2</td></tr>
<tr><td colspan="2">注：IL,ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1, IN2 和 OUT 或使用常量</td></tr>
</table>

➢ **NE 指令处理的数据类型**

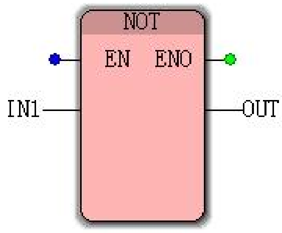
输入和输出	数据类型	描述
IN1	ANY	第一个输入
IN2	ANY	第二个输入
OUT	BOOL	结果： 当 IN1 <> IN2 时， OUT 为 1; 当 IN1 = IN2 时， OUT 为 0;

➢ **功能和动作例子**

比较地址 V517 与 V518 的值	描述
	当触点 C300 为 ON 时，执行该指令，地址 V517 的值与地址 V518 中的值比较，不相等时输出 1 执行的结果： V517 *V518=V519

10.1.32 NOT（逻辑非指令）

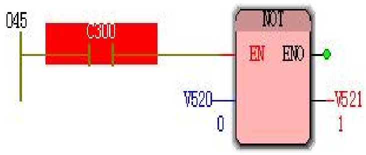
IL 编程语言	LD， FBD 编程语言

功能：NOT 指令用于将输入值按位取反，如 BYTE#2#11001100，经 NOT 运算得 BYTE#2#00110011 。	
LD IN NOT ST OUT ST 编程语 OUT:=NOT（IN）	
注：IL,ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

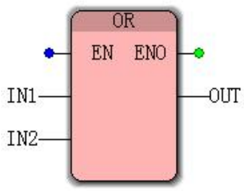
➢ **NOT 指令处理的数据类型**

输入和输出	数据类型	描述
IN	ANY_BIT	输入
OUT	ANY_BIT	结果

➢ **功能和动作例子**

取反地址 V520（BOOL）的值	描述
	当触点 C300 为 ON 时，执行该指令，取反地址 V520（BOOL）的值: 执行的结果：0（V520）取反输出 1（V521）

10.1.33 OR（逻辑或指令）

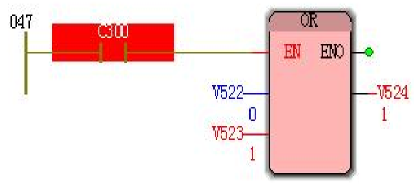
IL 编程语言	LD， FBD 编程语言
功能：OR 指令用于将输入值按位做逻辑或运算。	
LD IN1 OR IN2 ST OUT ST 编程语言 OUT:=（1N1） OR （1N2）	
注：IL,ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1, IN2 和 OUT 或使用常量	

➢ **OR 指令处理的数据类型**

输入和输出	数据类型	描述
IN1	ANY_BIT	第一个输入

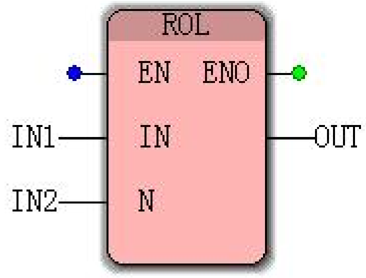
IN2	ANY_BIT	第二个输入
OUT	ANY_BIT	结果，逻辑或运算 IN1=0, IN2=0,OUT 为 0; IN1=0,IN2=1 , OUT 为 1; IN1=1， IN2=0， OUT 为 1; IN1=1， IN2=1， OUT 为 1;

➢ **功能和动作例子**

<table>
<tr><th>两个地址的数值进行或运算</th><th>描述</th></tr>
<tr><td>
</td><td>当触点 C300 为 ON 时，执行该指令，两个地址进行或运算，输出到地址 V524 中

执行的结果：V522^V523=V524</td></tr>
</table>

10.1.34 ROL（循环左移指令）

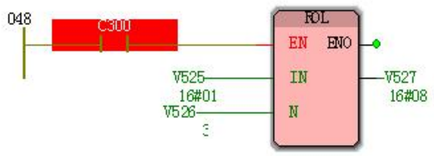
<table>
<tr><th>IL 编程语言</th><th>LD, FBD 编程语言</th></tr>
<tr><td colspan="2">功能：ROL 指令用于将输入值按位循环左移。</td></tr>
<tr><td>LD IN1</td><td rowspan="5">
</td></tr>
<tr><td>ROL IN2</td></tr>
<tr><td>ST OUT</td></tr>
<tr><td>ST 编程语言</td></tr>
<tr><td>OUT:=ROL（IN1，IN2）</td></tr>
<tr><td colspan="2">注：IL,ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1, IN2 和 OUT 或使用常量</td></tr>
</table>

➢ **ROL 指令处理的数据类型**

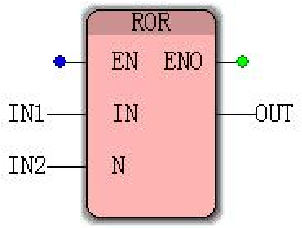
输入和输出	数据类型	描述
IN1	ANY_BIT	输入
IN2	ANY_INT	循环左移的位数

OUT	ANY_BIT	结果，当引脚 IN 或 N 的值有变化时，左移一次。 如下： 左移两位， 0 0 0 0 1 0 1 1 移动后的数值 1 1 0 0 0 0 1 0 移动前的数值

➢ **功能和动作例子**

数值 16#01（BYTE）向左移三位	描述
	当触点 C300 为 ON 时，执行该指令， 将地址 V525 所有位向左移 3 位： 执行的结果：16#01 向左移动 3 位存到 V527 中

10.1.35 ROR（循环右移指令）

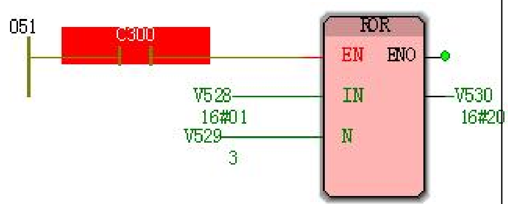
<table>
<tr><th>IL 编程语言</th><th>LD， FBD 编程语言</th></tr>
<tr><td colspan="2">功能：ROR 指令用于将输入值按位循环右移。</td></tr>
<tr><td>LD IN1</td><td rowspan="5"></td></tr>
<tr><td>ROR IN2</td></tr>
<tr><td>ST OUT</td></tr>
<tr><td>ST 编程语言</td></tr>
<tr><td>OUT:=ROR（IN1，IN2）</td></tr>
<tr><td colspan="2">注：IL,ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1, IN2 和 OUT 或使用常量</td></tr>
</table>

➢ **ROR 指令处理的数据类型**

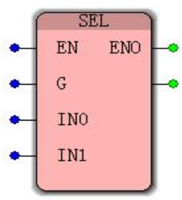
输入和输出	数据类型	描述
IN1（IN）	ANY_BIT	输入
IN2（N）	ANY_INT	循环右移的位数

OUT	ANY_BIT	结果，当引脚 IN 或 N 的值有变化时，右移一次。如下：右移两位 1　1　0　0　0　1　1　1 右移动后的数值 0　0　1　1　0　0　0　1 右移动前的数值

➤ **功能和动作例子**

将数值 16#01（BYTE）向右移三位	描述
	当触点 C300 为 ON 时，执行该指令， 将地址 V528 所有位向右移 3 位： 执行的结果：16#01 向右移动 3 位存到 V530 中；

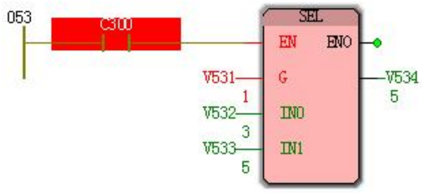
10.1.36 SEL（选择指令）

IL 编程语言	LD，FBD 编程语言
功能：SEL 指令用于根据一个布尔量的两个状态选择不同的输入值。	
LD IN SEL IN1，IN2 ST OUT ST 编程语言 OUT:=SEL（IN, IN1，IN2）	
注：IL,ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN, IN1, IN2 和 OUT 或使用常量	

➤ **SEL 指令处理的数据类型**

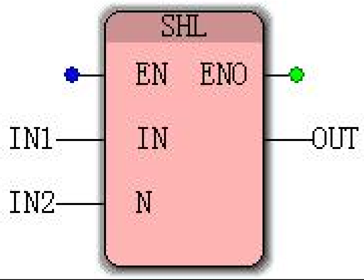
输入和输出	数据类型	描述
IN（G）	BOOL	选择输入端
IN1（INO）	ANY	第一个输入
IN2（IN1）	ANY	第二个输入
OUT	ANY	结果： 如果 IN=0, OUT=IN1; 如果 IN=1, OUT=IN2;

➢ 功能和动作例子

<table>
<tr><th>选择输出整数 3 或 5</th><th>描述</th></tr>
<tr><td>
</td><td>当触点 C300 为 ON 时，执行该指令，
当 G 为 1 时，输出地址 V533 中的值；当 G 为 0 时，输出地址 V532 中的值：

执行的结果：输出为 5</td></tr>
</table>

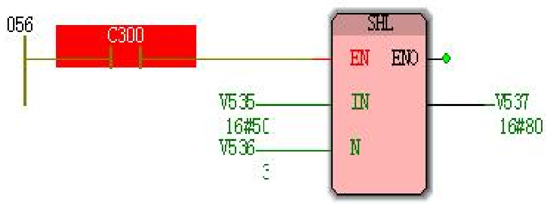
10.1.37 SHL（左移指令）

<table>
<tr><th>IL 编程语言</th><th>LD,　FBD 编程语言</th></tr>
<tr><td colspan="2">功能：SHL 指令用于将输入值按位左移，数据左端移出，右端补 0。</td></tr>
<tr><td>LD IN1</td><td rowspan="5">
</td></tr>
<tr><td>SHL IN2</td></tr>
<tr><td>ST OUT</td></tr>
<tr><td>ST 编程语言</td></tr>
<tr><td>OUT:=SHL(IN1，IN2)</td></tr>
<tr><td colspan="2">注：IL,ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1, IN2 和 OUT 或使用常量</td></tr>
</table>

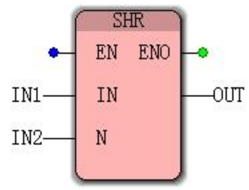
➢ SHL 指令处理的数据类型

输入和输出	数据类型	描述
IN1（IN）	ANY_BIT	输入
IN2（N）	ANY_INT	左移的位数
OUT	ANY_BIT	结果，当引脚 IN 或 N 的值有变化时，左移一次。左移两位，如下 左移动前的数值 1　1　1　0　0　0　1　1 ← 左移后的数值 ← 1　0　0　0　1　1　0　0

➢ **功能和动作例子**

<table>
<tr><th>将地址 V536（BYTE）的值，左移 3 位</th><th>描述</th></tr>
<tr><td>
</td><td>当触点 C300 为 ON 时，执行该指令，将地址 V535 所有位向左移 3 位：
执行的结果：16#01 向左移动 3 位输出 16#80</td></tr>
</table>

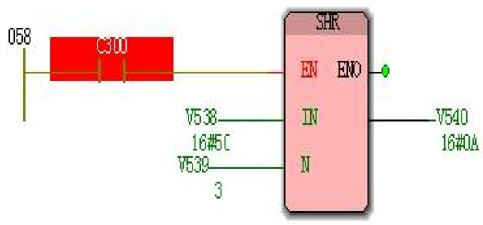
10.1.38 SHR（右移指令）

<table>
<tr><th>IL 编程语言</th><th>LD， FBD 编程语言</th></tr>
<tr><td colspan="2">功能：指令用于将输入值按位右移，数据右端移出，左端补 0。</td></tr>
<tr><td>LD IN1</td><td rowspan="5">
</td></tr>
<tr><td>SHR IN2</td></tr>
<tr><td>ST OUT</td></tr>
<tr><td>ST 编程语言</td></tr>
<tr><td>OUT:=SHR（IN1，IN2）</td></tr>
<tr><td colspan="2">注：IL,ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1, IN2 和 OUT 或使用常量</td></tr>
</table>

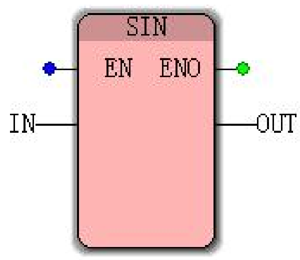
➢ **SHR 指令处理的数据类型**

输入和输出	数据类型	描述
IN1（IN）	ANY_BIT	输入
IN2（N）	ANY_INT	右移的位数
OUT	ANY_BIT	结果，当引脚 IN 或 N 的值有变化时，右移一次。右移两位，如下 右移动前的数值 0 0 0 0 0 0 1 1 → 右移动后的数值 → 0 0 0 0 1 1 0 0

➢ **功能和动作例子**

将数值 16#50（BYTE）向右移三位	描述
	当触点 C300 为 ON 时，执行该指令，将地址 V538 所有位向右移 3 位： 执行的结果：16#50 向右移动 3 位输出 16#0A：

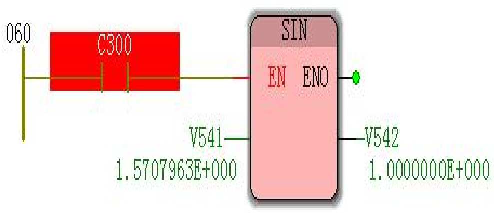
10.1.39 SIN（正弦指令）

IL 编程语言	LD, FBD 编程语言
功能：SIN 指令用于求输入值的正弦。	
LD IN SIN ST OUT ST 编程语言 OUT:=SIN（IN）	
注：IL,ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

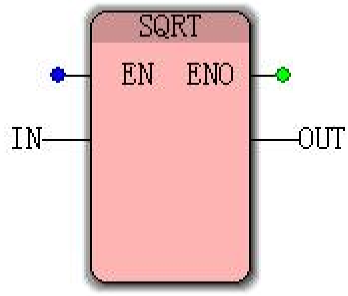
➢ **SIN 指令处理的数据类型**

输入和输出	数据类型	描述
IN	REAL	输入，以弧度表示角度
OUT	REAL	输出

➢ **功能和动作例子**

求 SIN（π/2）的值	描述
	当触点 C300 为 ON 时，执行该指令， 执行的结果：SIN（π/2）=1

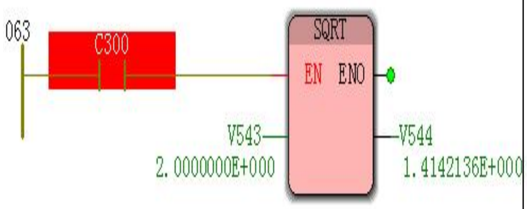
10.1.40 SQRT（平方根指令）

IL 编程语言	LD， FBD 编程语言
功能：SQRT 指令用于求输入值的平方根。	
LD IN	
SORT	
ST OUT	
ST 编程语言	
OUT:=SORT（ IN）	
注：IL,ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

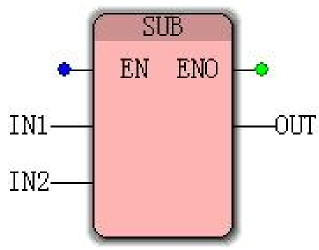
➢ **SQRT 指令处理的数据类型**

输入和输出	数据类型	描述
IN	REAL	输入
OUT	REAL	输出

➢ **功能和动作例子**

求√2 的值	描述
	当触点 C300 为 ON 时，执行该指令：执行的结果√2=1.4142136E+000

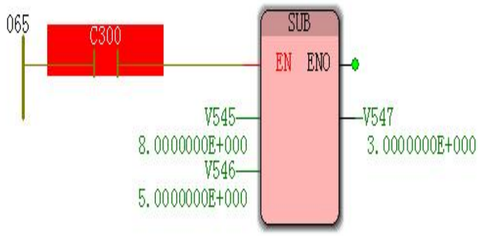
10.1.41 SUB（减法指令）

IL 编程语言	LD， FBD 编程语言
功能：SUB 指令用于求两个输入值的差。	
LD IN1	
SUB IN2	
ST OUT	
ST 编程语言	
OUT:=IN1-IN2	
注：IL,ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1, IN2 和 OUT 或使用常量	

➢ **SUB 指令处理的数据类型**

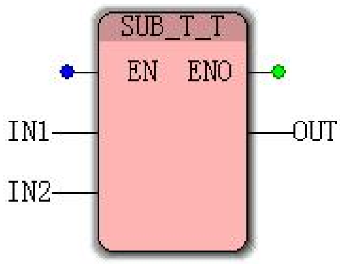
输入和输出	数据类型	描述
IN1	ANY_NUM	第一个输入
IN2	ANY_NUM	第二个输入
OUT	ANY_NUM	输出，OUT=IN1 一 IN2

➢ **功能和动作例子**

<table>
<tr><th>求浮点数 8.0 减 5.0 的值</th><th>描述</th></tr>
<tr><td>
</td><td>当触点 C300 为 ON 时，执行该指令：将地址 V545 的值减去 V546 的值，存放在 V547 的地址中：

执行的结果：V545-V546=V547</td></tr>
</table>

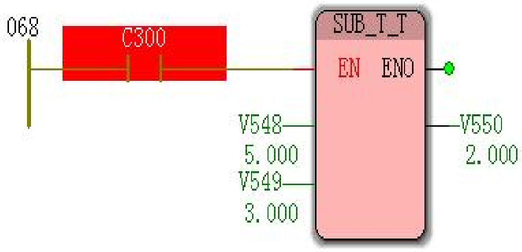
10.1.42 SUB_T_T（时间减法指令）

<table>
<tr><th>IL 编程语言</th><th>LD，FBD 编程语言</th></tr>
<tr><td colspan="2">功能：SUB_T_T 指令用于求两个时间输入值的差。</td></tr>
<tr><td>LD IN1</td><td rowspan="5">
</td></tr>
<tr><td>SUB IN2</td></tr>
<tr><td>ST OUT</td></tr>
<tr><td>ST 编程语言</td></tr>
<tr><td>OUT:=SUB_T_T（1N1，IN2）</td></tr>
<tr><td colspan="2">注：IL,ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1, IN2 和 OUT 或使用常量</td></tr>
</table>

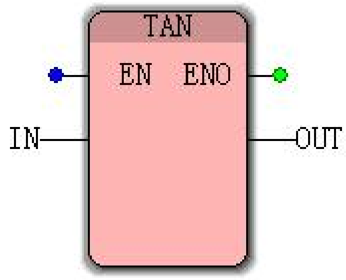
➢ **SUB_T_指令处理的数据类型**

输入和输出	数据类型	描述
IN1	TIME	第一个输入
IN2	TIME	第二个输入
OUT	TIME	输出，OUT=IN1-IN2

➢ **功能和动作例子**

时间 T#5s-T#3s 的值	描述
	当触点 C300 为 ON 时，执行该指令：将地址 V548 的值减去 V549 的值，存放在 V550 的地址中： 执行的结果：V548-V549=V550

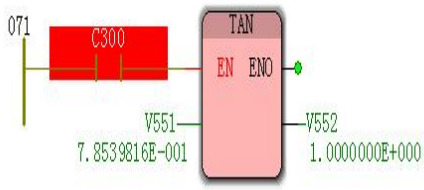
10.1.43 TAN（正切指令）

<table>
<tr><th>IL 编程语言</th><th>LD， FBD 编程语言</th></tr>
<tr><td colspan="2">功能：TAN 指令用于求输入值的正切。</td></tr>
<tr><td>LD IN</td><td rowspan="5"></td></tr>
<tr><td>TAN</td></tr>
<tr><td>ST OUT</td></tr>
<tr><td>ST 编程语言</td></tr>
<tr><td>OUT:=TAN（ IN）</td></tr>
<tr><td colspan="2">注：IL,ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量</td></tr>
</table>

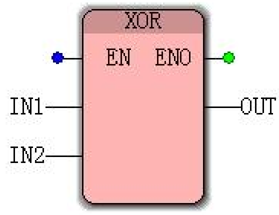
- **TAN 指令处理的数据类型**

输入和输出	数据类型	描述
IN	REAL	输入，以弧度表示角度
OUT	REAL	输出

- **功能和动作例子**

求 TAN（$\frac{\pi}{4}$）的值	描述
	当触点 C300 为 ON 时，执行该指令：执行的结果：TAN（$\frac{\pi}{4}$）=1

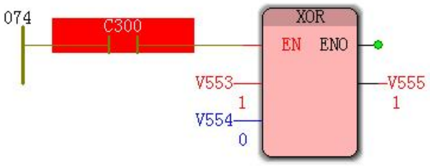
10.1.44 XOR（逻辑异或指令）

IL 编程语言	LD， FBD 编程语言
功能：XOR 指令用于将输入值按位做逻辑异或运算。	
LD IN1	
XOR IN2	
ST OUT	
ST 编程语言	
OUT:=（IN1）XOR （IN2）	
注：IL,ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1, IN2 和 OUT 或使用常量	

➢ **XOR 指令处理的数据类型**

输入和输出	数据类型	描述
IN1	ANY_BIT	第一个输入
IN2	ANY_BIT	第二个输入
0UT	ANY_BIT	结果，逻辑异或运算 IN1=0,IN2=0,OUT 为 0; IN1=0, IN2=1,OUT 为 1; IN1=1, IN2=0,OUT 为 1; IN1=1, IN2=1 ,OUT 为 0;

➢ **功能和动作例子**

求地址 V553（BOOL）与 V554（BOOL）异或的值	描述
	当触点 C300 为 ON 时，执行该指令：当 V553（BOOL）与 V554（BOOL）值相异时，输出为 1 执行的结果：V553 与 V554 相异输出 1

10.2 功能块

功能块是带有多个输入和输出参数的程序组织单元 POU，它们有内部存储器，功能块的返回值取决于其内部存储单元的值。功能块的缩写为 FBD 与前述的功能不同，功能块必须被实例化，实例名称可以是默认的名称，也可更改默认名，实例名称在 POU 内必须是独一无二的。在 FBD 和 LD 编程中，这个实例名称出现在功能块上部。

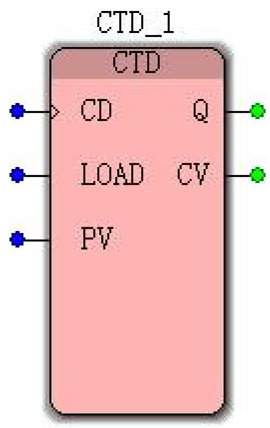
MULTIPROG 编程过程中，可以使用以下功能块

- 〇 双稳态功能块
- 〇 脉冲边沿检测功能块
- 〇 计数器功能块
- 〇 定时器功能块

功能块中包含的指令（在编辑向导中，从下拉列表选择”功能块”）

名称	描述
CTD	递减计数器
CTU	递增计数器
CTUD	增/减计数器
F_TRIG	下降沿检测
R_TRIG	上升沿检测
RS	复位优先
SR	置位优先
TOF	断开延迟时器
TON	接通延迟时器
TP	脉冲

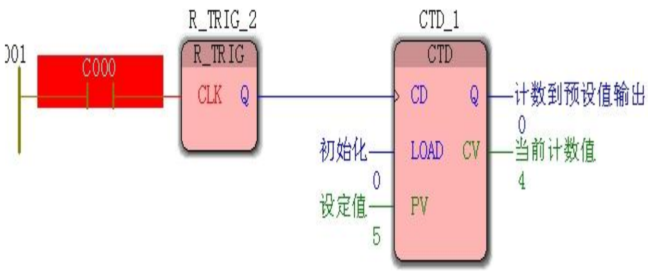
10.2.1 CTD（递减计数器指令）

IL 编程语言	LD, FBD 编程语言
功能：TD 指令用于对输入端做递减计数，LOAD 端为 FALSE 时，如果 CD 输入端出现一个上升沿，则 CV 端减 1。如果计数值 CV 达到了记数器的下限值 0，则在 Q 输出端发出 TRUE 信号，CTD 功能块停止记数。LOAD 端为 TRUE 时，计数器停止计数，并将 PV 输入端的数值赋给 CV 端。	
LD varl	
ST CTD_1.CD	
LD var2	
ST CTD_1.L0AD	
LD var3	
ST CTD_1.PV	
CAL CTD_1	
LD CTD_1.Q	
ST var4	
LD CTD_1.CV	
ST var5	
ST 编程语言	
CTD_1（CD:=var1，LOAD:=var2, PV:=var3）	
var4:=CTD_1.Q	
vary:=CTD_1.CV	
注：IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 varl1~var5 常量	

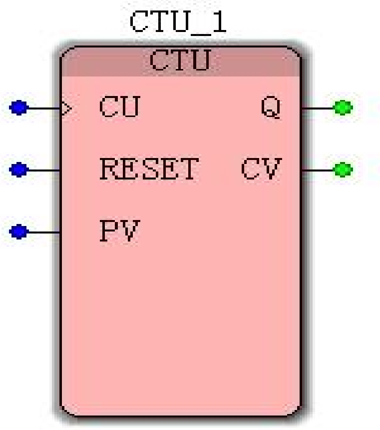
➢ **CTD 指令处理的数据类型**

输入	数据类型	描述
CD	BOOL	如果 CD 出现一个上升沿,CV 减 1
LOAD	BOOL	LOAD 为 FALSE 时，启动计数，为 TRUE，将 PV 赋给 CV，初始化计数器。
PV	INT	递减计数起始值
输出	数据类型	描述
Q	BOOL	当 C V=0 时，O=1
CV	INT	计数值

➢ 功能和动作例子

当触点 C000 由 OFF 到 ON 五次后 Q 输出 1	描述
	这个计数器功能块递减记数。假设在 CD 输入端有一个上升沿，且 LOAD=FALSE，则 CV 减一。 如果达到了计数器的最终值（PV），则在 Q 输出端发出一个 TRUE 信号，并且功能块停止计数。 如果 LOAD=TRUE，则按 PV 输入的值初始化计数器。 为了启用计数过程，LOAD 输入端必须为 FALSE。否则将重新初始化计数器。

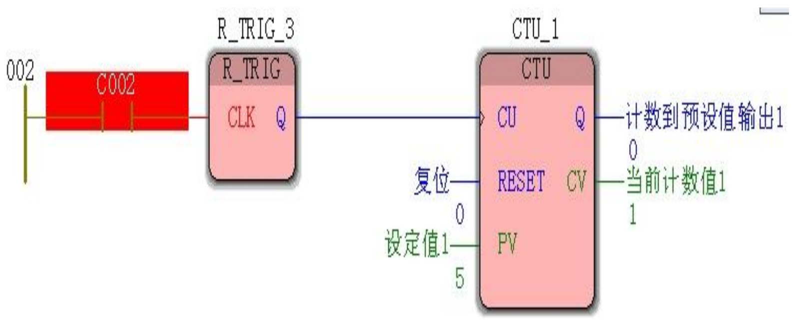
10.2.2 CTU（递增计数器指令）

IL 编程语言	LD, FBD 编程语言
功能：CTU 指令用于对输入端做递增计数， RESET 端为 FLSE 时，如果 CU 输入端出现一个上升沿，则 CV 端加 1。如果计数值 CV 达到了记数器的上限值 PV，则在 Q 输出端发出 TRUE 信号，CTU 功能块停止记数。RESET 端为 TRUE 时，计数器停止计数，CV 端清零。	
LD varl	
ST CTU_1.CU	
LD var2	
ST CTU_1.RESET	
LD var3	
ST CTU_1.PV	
CAL CTU_1	
LD CTU_1.Q	
ST var4	
LD CTU_1.CV	
ST var5	
ST 程语言	
CTU_1（CD:=var1，LOAD:=var2, PV:=var3）	
var4:=CTU_1.Q	
var5:=CTU_1.CV	
注：IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 var1 ~var5 或使用常量	

➢ **CTU 指令处理的数据类型**

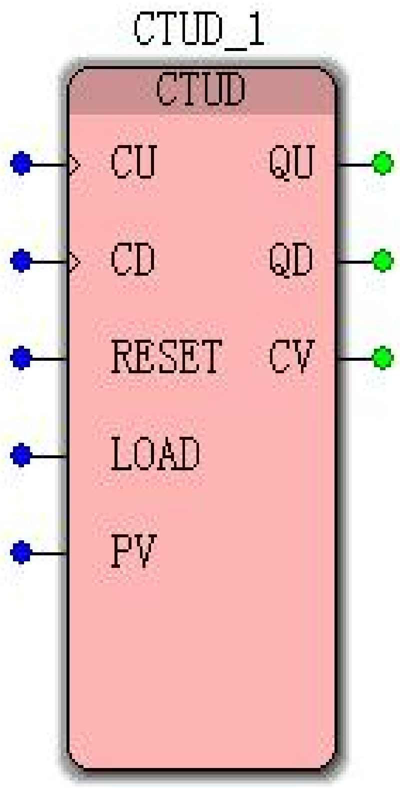
输入	数据类型	描述
CU	BOOL	如果 CU 出现一个上升沿,CV 加 1
RESET	BOOL	RESET 为 FALSE 时，启动计数; 为 TRUE 时，将 CV 清零，初始化计数器，Q 端复位。
PV	INT	递增计数上限值
输出	数据类型	描述
Q	BOOL	当 CV=PV 时，Q=1
CV	INT	计数值

➢ **功能和动作例子**

<table>
<tr><th>当触点 C002 由 OFF 到 ON 五次后 Q 输出 1</th><th>描述</th></tr>
<tr><td>


</td><td>这个记数器功能块递增计数。 假设在 CU 输入端有一个上升沿，并且 RESET=FALSE，则 CV 递增一。 如果达到了计数器的最终值（PV），则在 Q 输出端发出一个 TRUE 信号，并且功能块停止计数。
如果 RESET=TRUE，则用 0 初始化记数器。为了使能记数过程，RESET 输入端必须为 FALSE。 否则记数器将总被重新初始化。</td></tr>
</table>

10.2.3 CTUD（增减双向计数器指令）

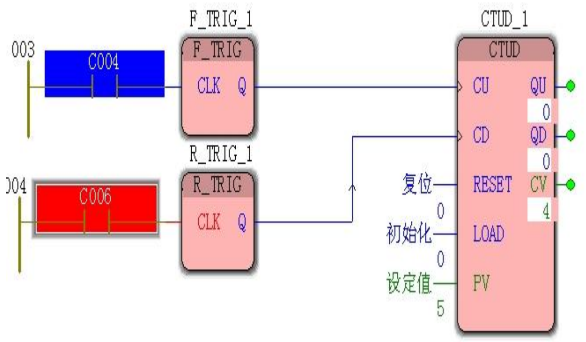
<table>
<tr><th>IL 编程语言</th><th>LD, FBD 编程语言</th></tr>
<tr><td colspan="2">功能：CTUD 指令用于对输入端做递增或者递减计数。 RESET 端和 LOAD 端同为 FALSE 时，允许计数:如果 CU 输入端出现一个上升沿，则 CV 端加 1;如果 CD 输入端出现一个上升沿，则 CV 端减 1;当 CV=PV 时，则 QU=1, CTUD 功能块停止递增计数;如果 CV=0，则 QD=1, CTUD 功能块停止递减记数。RESET 端为 TRUE 时，计数器停止递增、递减计数，CV 端清零; LOAD 端为 TRUE 时，计数器停止递增、递减计数，PV 值赋给 CV 端。</td></tr>
<tr><td>LD varl</td><td rowspan="24">
</td></tr>
<tr><td>ST CTUD_1.CU</td></tr>
<tr><td>LD var2</td></tr>
<tr><td>ST CTUD_1.CD</td></tr>
<tr><td>LD var3</td></tr>
<tr><td>ST CTUD_1.RESET</td></tr>
<tr><td>LD var4</td></tr>
<tr><td>ST CTUD_1.LOAD</td></tr>
<tr><td>LD var5</td></tr>
<tr><td>ST CTUD_1.PV</td></tr>
<tr><td>CAL CTUD_1</td></tr>
<tr><td>LD CTUD_1.QU</td></tr>
<tr><td>ST var6</td></tr>
<tr><td>LD CTUD_1.QD</td></tr>
<tr><td>ST var7</td></tr>
<tr><td>LD CTUD_1.CV</td></tr>
<tr><td>ST var8</td></tr>
<tr><td>ST 编程语言</td></tr>
<tr><td>CTUD_1（CU:=var1, CD:=var2, RESET:= var3, LOAD:=var4, PV:=var5）;</td></tr>
<tr><td>var6:=CTUD_ 1.QU;</td></tr>
<tr><td>var7:=CTUD_ 1.QD;</td></tr>
<tr><td>ar8:=CTUD_ 1.CV;</td></tr>
<tr><td colspan="2">注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 varl ~var8 或使用常量</td></tr>
</table>

➢ **CTU 指令处理的数据类型**

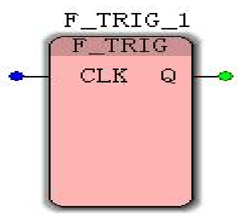
输入	数据类型	描述
CU	BOOL	如果 CU 出现一个上升沿,CV 加 1
CD	BOOL	如果 CD 出现一个上升沿,CV 减 1
RESET	BOOL	RESET 为 FALSE 时，启动计数，为 TRUE 时，将 CV 清零，初始化计数器。
LOAD	BOOL	LOAD 为 FALSE 时，启动计数，为 TRUE 时，将

		PV 赋给 CV，初始化计数器。
PV	INT	计数上限值或起始值
输出	数据类型	描述
QU	BOOL	当 CV=PV 时，QU=1
QD	BOOL	当 C V=0 时，QD=1
CV	INT	计数值

➤ **功能和动作例子**

当 CU 有一个上升沿来时当值前加 1，当 CD 有一个下降沿当前值减 1	描述
	这个计数器功能块递增或者递减记数。 假设在 CU 输入端有一个上升沿，则 CV 递增一。 假设在 CD 输入端有一个上升沿，则 CV 递减一。 如果 CV = PV，OU 设置为 TRUE。 如果 CV = PV，OU 设置为 TRUE。 如果 RESET = TRUE，计数器初始化为 0。如果 LOAD = TRUE，计数器初始化为 PV。为了启用计数过程，RESET 和 LOAD 输入端都必须为 FALSE。否则将重新初始化计数器。

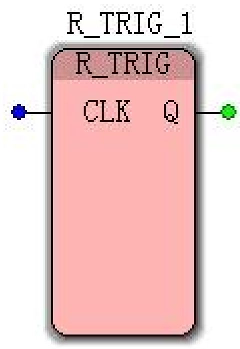
10.2.4 F_TRIG（下降沿检测指令）

IL 编程语言	LD, FBD 编程语言
功能：F_TRIG 指令用于检测输入端的下降沿，如果在输入端 CLK 检测到一个下降沿，则输出端 Q 由 FALSE 变为 TRUE，直到下一次扫描到这条指令，Q 输出端都将一直保持为 TRUE	
LD varl	
ST F_TRIG_1.CLK	
CAL F_TRIG_1	
LD F_TRIG_1.Q	
ST var2	
ST 编程语言	
F_TRIG_1（CLK:=var1）	
var2:=F_TRIG_1.Q	
注：IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 varl1~var2 或使用常量	

➤ **F_TRIG 指令处理的数据类型**

参数	数据类型	描述
CLK	BOOL	下降沿有效
Q	BOOL	当 CLK 出现下降沿时，Q=由 0 变为 1,直到下一次扫描到这条指令

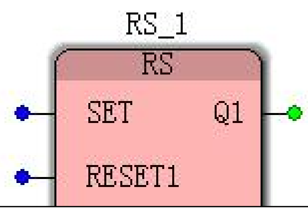
10.2.5 R_TRIG（上升沿检测指令）

<table>
<tr><th>IL 编程语言</th><th>LD， FBD 编程语言</th></tr>
<tr><td colspan="2">功能：R_TRIG 指令用于检测输入端的上升沿，如果在输入端 CLK 检测到一个上升沿，则输出端 Q 由 FALSE 变为 TRUE，直到下一次扫描到这条指令，Q 输出端都将一直保持为 TRUE。</td></tr>
<tr><td>LD var1</td><td rowspan="8">
</td></tr>
<tr><td>ST R_TRIG_1.CLK</td></tr>
<tr><td>CAL R_TRIG_1</td></tr>
<tr><td>LD R_TRIG_1.Q</td></tr>
<tr><td>ST var2</td></tr>
<tr><td>ST 编程语言</td></tr>
<tr><td>R_TRIG_1（CLK:=var1）</td></tr>
<tr><td>var2:=R_TRIG_1.Q</td></tr>
<tr><td colspan="2">注：IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 varl1~var2 或使用常量</td></tr>
</table>

➢ **CTU 指令处理的数据类型**

参数	数据类型	描述
CLK	BOOL	上升沿有效
Q	BOOL	当 CLK 出现上升沿时，Q=由 0 变为 1 直到下一次扫描到这条指令

10.2.6 RS（RS 触发器指令）

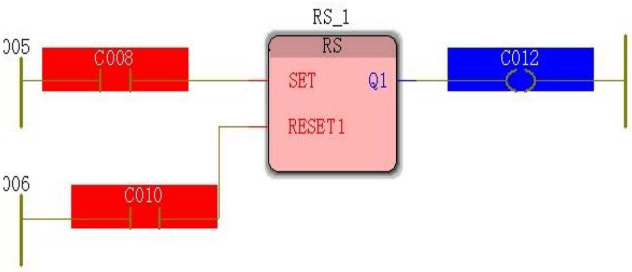
<table>
<tr><th>IL 编程语言</th><th>LD， FBD 编程语言</th></tr>
<tr><td colspan="2">功能：RS 指令用于实现 RS 触发器的功能，如果 SET 端为 TRUE 且 RESET 端为 FALSE，则输出端 Q1 被置位，即使 SET 变为 FALSE,Q1 仍然保持置位状态。如果 RESET1=TRUE，则不论 SET 端为 TRUE 还是 FALSE, Q1 端均被复位，即使 RESET1 变为 FALSE, Q1 仍然保持复位状态。</td></tr>
<tr><td>LD var1</td><td rowspan="5">
</td></tr>
<tr><td>ST RS_1.SET</td></tr>
<tr><td>LD var2</td></tr>
<tr><td>ST RS_1.RESETI</td></tr>
<tr><td>AL RS_1</td></tr>
</table>

LD RS_1.Q1	
ST var3	
ST 编程语言	
RS_1（SET:=var1，RESET:=var2）	
Xar3:=RS_1.Q1	
注：IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 varl1~var3 或使用常量	

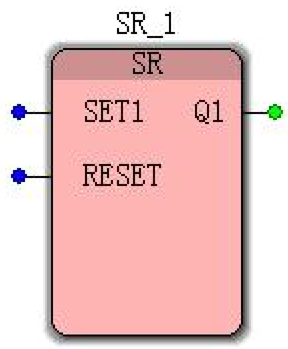
➤ **RS 指令处理的数据类型**

参数	数据类型	描述
SET	BOOL	置位
RESET1	BOOL	复位
Q1	BOOL	SET=0,RESET1=0,Q1 保持上次状态; SET=1,RESET1=0,Q1=1; SET=0,RESET1=1,Q1=0;

➤ **功能和动作例子**

当触点 C008 与 C010 同时为 ON 时，复位优先 输出 Q1 0	描述
	这个双稳态功能块实现了 Q1 输出端的优先复位。 如果输入端 SET=TRUE，则输出端 Q1 被置位。即使 SET 为 FALSE，Q1 仍然保持置位状态。 如果 RESET1=TRUE，则 Q1 被复位。 如果两个输入均为 TRUE，则由 RESET1 将 Q1 输出端设置为 FALSE。

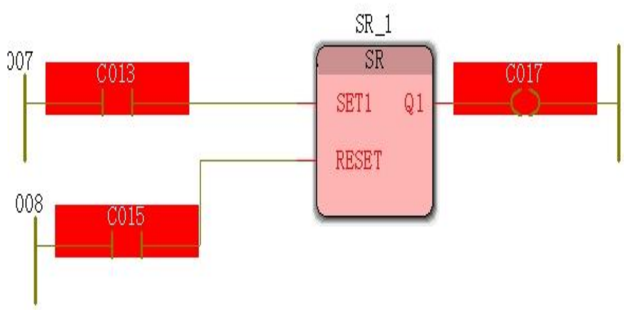
10.2.7 SR（SR 触发器指令）

<table>
<tr><th>IL 编程语言</th><th>LD， FBD 编程语言</th></tr>
<tr><td colspan="2">功能：RS 指令用于实现 RS 触发器的功能，如果 SET1 端为 TRUE，则不论 RESET 端为 TRUE 还是 FALSE，输出端 Q1 被置位，即使 SET1 变为 FALSE, Q1 仍然保持置位状态。如果 RESET 端为 TRUE 且 SET1 端为 FALSE,则 Q1 端均被复位，即使 RESET 变为 FALSE, Q1 仍然保持复位状态。</td></tr>
<tr><td>LD var1</td><td rowspan="10"></td></tr>
<tr><td>ST SR_1.SET1</td></tr>
<tr><td>LD var2</td></tr>
<tr><td>ST SR_1.RESET</td></tr>
<tr><td>CAL SR_1</td></tr>
<tr><td>LD SR_1.Q1</td></tr>
<tr><td>ST var3</td></tr>
<tr><td>ST 编程语言</td></tr>
<tr><td>SR_1（SET1:=var1，RESET:=var2）</td></tr>
<tr><td>var3:=SR_1.Q1</td></tr>
<tr><td colspan="2">注:IL， ST 语言编程时需要在当前 POU 的变量工作单中插入变量 varl1~var3 或使用常量</td></tr>
</table>

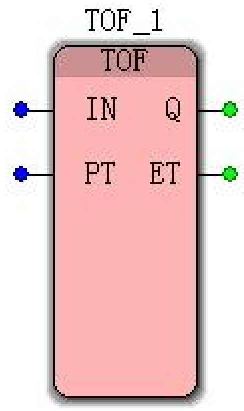
- **RS 指令处理的数据类型**

参数	数据类型	描述
SET1	BOOL	置位
RESET	BOOL	复位
Q1	BOOL	结果 SET1=0,RESET=0,Q1 保持上次状态; SET1=1,RESET=0,Q1=1; SET1=0,RESET=1,Q1=0; SET1=1,RESET=1,Q1=1;

- **功能和动作例子**

当触点 C013 与 C015 同时为 ON 时，置位优先 Q1 输出 1	描述
	这个双稳态功能块实现 Q1 输出端的优先置位。 如果输入 SET1=TRUE，则 Q1 输出端被置位。 即使 SET 为 FALSE，Q1 仍然保持置位状态。 如果 RESET=TRUE，则 Q1 输出端被复位。 如果两个输入均为 TRUE，则 Q1 输出端被 SET1 设置为 TRUE。

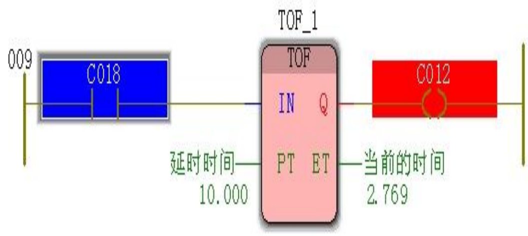
10.2.8 TOF（延迟断开定时器指令）

<table>
<tr><th>IL 编程语言</th><th>LD, FBD 编程语言</th></tr>
<tr><td colspan="2">功能：:TOF 指令用于实现延迟断开功能，如果输入端 IN 为 TRUE，则输出端 Q 立即变为 TRUE;如果输入端 IN 由 TRUE 变为 FALSE，输出端 Q 将延迟一定时间后再由 TRUE 变为 FALSE，这个延迟时间为 PT 的值，ET 端记录了从 IN 变为 FALSE 的时刻起到 Q 端变为 TRUE 之间的计时时间。</td></tr>
<tr><td>LD var1</td><td rowspan="13">
</td></tr>
<tr><td>ST TOF_1.IN</td></tr>
<tr><td>LD var2</td></tr>
<tr><td>ST TOF_1.PT</td></tr>
<tr><td>CAL TOF_1</td></tr>
<tr><td>LD TOF_1.Q</td></tr>
<tr><td>ST var3</td></tr>
<tr><td>LD TOF_1.ET</td></tr>
<tr><td>ST var4</td></tr>
<tr><td>ST 编程语言</td></tr>
<tr><td>TOF_1（IN:=var1，PT:=var2）</td></tr>
<tr><td>var3:=TOF_ 1.Q</td></tr>
<tr><td>var4:=TOF_1.ET</td></tr>
<tr><td colspan="2">注：IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 varl1~var4 或使用常量</td></tr>
</table>

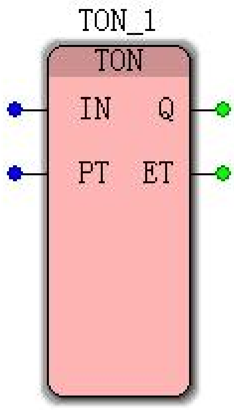
➢ **TOF 指令处理的数据类型**

参数	数据类型	描述
IN	BOOL	使能输入端
PT	TIME	要 Q 端延迟断开的时间
Q	BOOL	结果 IN=1, Q=1; IN=0，延迟 PT 时间后， Q 由 1 变为 0
ET	TIME	从 IN 变为 FALSE 的时刻起到 Q 端变为 TRUE 之间的计时时间

➢ **功能和动作例子**

当触点 C018ON 时，Q 立即输出 1 ，当触点由 ON 到 OFF 时，Q 延时 PV（设定值）后输出 0	描述
	如果输入 IN 从 TRUE 变为 FALSE，在延迟输入 PT 中的时长后关机。经过 PT 值的时长后，Q 值设置为 FALSE。 在输出端 ET 上显示过程时间间隔。

10.2.9 TON（延迟接通定时器指令）

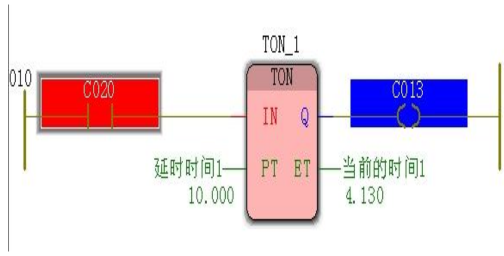
IL 编程语言	LD, FBD 编程语言
功能：TON 指令用于实现延迟接通功能，如果输入端 IN 为 FALSE，则输出端 Q 立即变为 FALSE;如果输入端 IN 由 FALSE 变为 TRUE，输出端 Q 将延迟一定时间后再由 FALSE 变为 TRUE，这个延迟时间为 PT 的值，ET 端记录了从 IN 变为 FALSE 的时刻起到 Q 端变为 TRUE 之间的计时时间。	
LD var1	
ST TON_1.IN	
LD var2	
ST TON_1.PT	
CALTON_1	
LD TON_1.Q	
ST var3	
LD TON_1.ET	
ST var4	
ST 编程语言	
TON_1（1N:=var1，PT:=var2）	
var3:=TON_1.Q	
var4:=TON_.ET	
注：IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 varl1~var4 或使用常量	

➢ **TON 指令处理的数据类型**

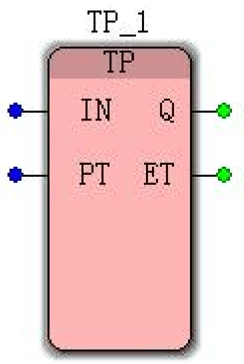
参数	数据类型	描述
IN	BOOL	使能输入端
PT	TIME	要 Q 端延迟接通的时间，如 T#5S
Q	BOOL	结果:如果 IN=0，则 Q=0;如果 IN=1，延迟 PT 时间后，Q 由 0 变为 1
ET	TIME	从 IN 变为 TRUE 的时刻起到 Q 端变为 TRUE 之间的计时时间

➢ **功能和动作例子**

当触点 C020 为 ON 时开始，Q 延时 PV（设定值）输出 1	描述

<table>
<tr><td>
</td><td>如果输入 IN 从 TRUE 变为 FALSE，在延迟输入 PT 中的时长后开机。经过 PT 值的时长后，Q 值设置为 TRUE。在输出端 ET 上显示过程时间间隔。</td></tr>
</table>

10.2.10 TP（脉冲指令）

<table>
<tr><th>IL 编程语言</th><th>LD, FBD 编程语言</th></tr>
<tr><td colspan="2">功能：:TP 指令用于实现一定宽度脉冲的功能，如果输入端 IN 由 FALSE 变为 TRUE，则输出端 Q 产生一个时间间隔为 PT 的脉冲。如果输入端 IN 在 PT 时间内又变为 FALSE，则输出端 Q 照样产生一个 PT 宽度的脉冲。ET 端记录了从 IN 变为 FALSE 的时刻起到 Q 端变为 TRUE 之间的计时时间。</td></tr>
<tr><td>LD var1</td><td rowspan="13">
</td></tr>
<tr><td>ST TP_1.IN</td></tr>
<tr><td>LD var2</td></tr>
<tr><td>ST TP_1.PT</td></tr>
<tr><td>CAL TP_1</td></tr>
<tr><td>LD TP_1.Q</td></tr>
<tr><td>ST var3</td></tr>
<tr><td>LD TP_1.ET</td></tr>
<tr><td>ST var4</td></tr>
<tr><td>ST 编程语言</td></tr>
<tr><td>TP_1（IN:=varl，PT:=var2）</td></tr>
<tr><td>var3:=TP_1.Q</td></tr>
<tr><td>var4:=TP_1.ET</td></tr>
<tr><td colspan="2">注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 varl1~var4 或使用常量</td></tr>
</table>

➢ **TON 指令处理的数据类型**

参数	数据类型	描述
IN	BOOL	IN 的上升沿有效
PT	TIME	脉冲时间间隔
Q	BOOL	结果，IN 上升沿时，Q 产生一个 PT 宽度的脉冲
ET	TIME	从 IN 变为 TRUE 的时刻起到 Q 端变为 TRUE 之间的计时时间，期间 IN 的状态变化对 Q 不起作用

10.3 类型转换 FU

类型转换功能，简称类型转换 FU，它将一个类型的数据转换为另一个类型的数据，所以它带有一个输入参数和一个输出参数。MULTIPROG 编程过程中，可以使用以下的类型转换 FU

- ●BYTE 型 BCD 数据的转换
- ●WORD 型 BCD 数据的转换
- ●DWORD 型 BCD 数据的转换
- ●BOOL 型数据的转换
- ●BYTE 型数据的转换
- ●WORD 型数据的转换
- ●DWORD 型数据的转换
- ●SINT 型数据的转换
- ●INT 型数据的转换
- ●DINT 型数据的转换
- ●USINT 型数据的转换
- ●UINT 型数据的转换
- ●UDINT 型数据的转换
- ●LREAL 型数据的转换
- ●REAL 型数据的转换
- ●TRUNC 小数取整

功能中包含的指令（在编辑向导中，从下拉列表选择”类型转换 FU”）

分类	功能		
BYTE 型 BCD 数据的转换	B_BCD_TO_SINT	B_BCD_TO_INT	B_BCD_TO_DINT
WORD 型 BCD 数据的转换	W_BCD_TO_SINT	W_BCD_TO_INT	W_BCD_TO_DINT
DWORD 型 BCD 数据的转换	D_BCD_TO_SINT	D_BCD_TO_INT	D_BCD_TO_DINT
BCD 型数据的转换	BCD_TO_DINT		
TIME 型数据的转换	TIME_TO_DINT		
BOOL 型数据的转换	BOOL_TO_BYTE	BOOL_TO_WORD	BOOL_TO_DWORD
	BOOL_TO_SINT	BOOL_TO_INT	BOOL_TO_DINT
	BOOL_TO_USINT	BOOL_TO_UINT	BOOL_TO_UDINT
	BOOL_TO_REAL	BOOL_TO_LREAL	
BYTE 型数据的转换	BYTE_TO_BOOL BYTE_TO_WORD BYTE_TO_DWORD	BYTE_TO_BOOL BYTE_TO_WORD BYTE_TO_DWORD	BYTE_TO_BOOL BYTE_TO_WORD BYTE_TO_DWORD
WORD 型数据的转换	WORD_TO_BOOL WORD_TO_BYTE WORD_TO_DWORD	WORD_TO_BOOL WORD_TO_BYTE WORD_TO_DWORD	WORD_TO_BOOL WORD_TO_BYTE WORD_TO_DWORD
DWORD 型数据的转	DWORD_TO-BOOL	DWORD_TO-BOOL	DWORD_TO-BOOL

换	DWORD_TO_BYTE DWORD_TO_WORD	DWORD_TO_BYTE DWORD_TO_WOR D	DWORD_TO_BYTE DWORD_TO_WORD
SINT 型数据的转换	SINT_TO_B_BCD	SINT_TO_W_BCD	SINT_TO_D_BCD
	SINT_TO_BOOL	SINT_TO_BYTE	SINT_TO_WORD
	SINT_TO_DWORD	SINT_TO_INT	SINT_TO_DINT
	SINT_TO_USINT	SINT_TO_UINT	SINT_TO_UDINT
	SINT_TO_REAL	SINT_TO_LREAL	
INT 型数据的转换	INT_TO_B_BCD	INT_TO_W BCD	INT_TO_D_BCD
	INT_TO_BOOL	INT_TO_BYTE	INT_TO_WORD
	INT_TO_DWORD	INT_TO_SINT	INT_TO_DINT
	INT_TO_USINT	INT_TO_UINT	INT_TO_UDINT
	INT_TO_REAL	INT_TO_LREAL	
DINT 型数据的转换	DINT_TO_B_BCD	DINT_TO_W_BCD	DINT_TO_D_BCD
	DINT_TO_BOOL	DINT_TO_BYTE	DINT_TO_WORD
	DINT_TO_DWORD	DINT_TO_SINT	DINT_TO _INT
	DINT_TO_USINT	DINT_TO_UINT	DINT_TO_UDINT
	DINT_TO_REAL	DINT_TO_LREAL	DINT_TO_BCD
	DINT_TO_TIME		
USINT 型数据的转换	USINT_TO_BOOL	USINT_TO_BYTE	USINT_TO_WORD
	USINT_TO_DWORD	USINT_TO_SINT	USINT_TO_INT
	USINT_TO_DINT	USINT_TO_UINT	USINT_TO_UDINT
	USINT_TO_REAL	USINT_TO_LREAL	
UINT 型数据的转换	UINT_TO_BOOL	UINT_TO_BYTE	UINT_TO_WORD
	UINT_TO_DWORD	UINT_TO_SINT	UINT_TO_INT
	UINT_TO_DINT	UINT_TO_USINT	UINT_TO_UDINT
	UINT_TO_REAL	UINT_TO_LREAL	
UDINT 型数据的转换	UDINT_TO_BOOL	UDINT_TO_BYTE	UDINT_TO_WORD
	UDINT_TO_DWORD	UDINT_TO_SINT	UDINT_TO_INT
	UDINT_TO _DINT	UDINT_TO_USINT	UDINT_TO_UINT
	UDINT_TO_REAL	UDINT_TO_LREAL	
LREAL 型数据的转换	LREAL_TO_BOOL	LREAL_TO_BYTE	LREAL_TO_WORD
	LREAL_TO_DWORD	LREAL_TO_SINT	LREAL_TO_INT
	LREAL_TO_DINT	LREAL_TO_USINT	LREAL_TO_UINT
	LREAL_TO_UDINT	LREAL_TO_REAL	
REAL 型数据的转换	REAL_TO_BOOL	REAL_TO_BYTE	REAL_TO_WORD
	REAL_TO_DWORD	REAL_TO_SINT	REAL_TO_INT

	REAL_TO_DINT	REAL_TO_USINT	REAL_TO_UINT
	REAL_TO_UDINT	REAL_TO_LREAL	
TRUNC 型数据的转换	TRUNC	TRUNC_SINT	TRUNC_INT
	TRUNC_DINT		

10.3.1 BYTE 型 BCD 数据的转换

类型转换 FU	功能
BYTE 型 BCD 数据的转换	BYTE 型 BCD 数据的转换包括以下三个指令:B_BCD_TO_SINT, B_BCD_TO_INT 和 B_BCD_TO_DINT。这三个指令能将一个 BYTE 数据类型的 BCD（二进制编码的十进制数）输入值分别转换为一个 SINT、INT 和 DINT 数据类型的输出值。

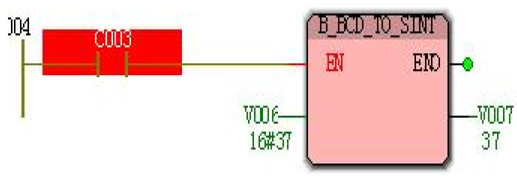
➢ **BYTE 型 BCD 数据的转换指令**

指令	输入值	输出值	描述
B_BCD_TO_SINT	BYTE 型 BCD 码	SINT	输入值取值范围 BCD 码 16#0099; 对应输出值 SINT, INT 和 DINT 均为 0~99。
B_BCD_TO_INT	BYTE 型 BCD 码	INT	
B_BCD_TO_DINT	BYTE 型 BCD 码	DINT	

➢ **用法（以 B_BCD_TO_SINT 为例）**

IL 编程语言	LD, FBD 编程语言
LD IN	
B_BCD_TO_SINT	
ST OUT	
ST 编程语言	
OUT:=B_ BCD_TO_ SINT （IN）	
注：IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

➢ **功能和动作例子**

16#37 转为短整型	描述
	当触点 C003 为 ON 时，执行该指令；将地址为 V006 中的 16 进制的数转为短整型，存放到 V007 中； 执行的结果：16#37----37

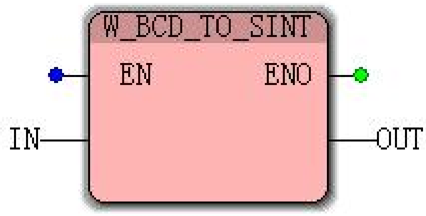
10.3.2 WORD 型 BCD 数据的转换

类型转换 FU	功能
WORD 型 BCD 数据的转换	WORD 型 BCD 数据的转换包括以下三个指令: W_BCD_TO_SINT, W_BCD_TO_INT 和 W_BCD_TO_DINT。 这三个指令能将一个 WORD 数据类型的 BCD（二进制编码的十进制数）输入值分别转换为一个 SINT、INT 和 DINT 数据类型的输出值。

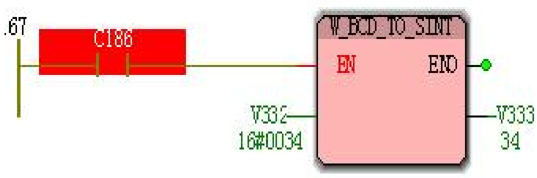
➢ **WORD 型 BCD 数据的转换指令**

指令	输入值	输出值	描述
W_BCD_TO_SINT	WORD 型 BCD 码	SINT	输入值 BCD 码 16#0127，对应输出值 SINT 为 127，输入再增加则输出为-1;输入值 BCD 码 16#0000~16#9999，对应输出值 INT, DINT 均为 0~9999。
W_BCD_TO_INT	WORD 型 BCD 码	INT	
W_BCD _TO_DINT	WORD 型 BCD 码	DINT	

➢ **用法（以 W_BCD_TO_SINT 为例）**

IL 编程语言	LD, FBD 编程语言
LD IN	
W BCD TO SINT	
ST OUT	
ST 编程语言	
OUT:=W_BCD_TO_SINT （IN）	
注：IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

➢ **功能和动作例子**

16#0010 转化为短整型	描述
	当触点 C186 为 ON 时，执行该指令；将地址为 V332 中的 16 进制的数转为短整型，存放到 V333 中； 执行的结果：16#0034 转为 34

10.3.3 DWORD 型 BCD 数据的转换

类型转换 FU	功能
DWORD 型 BCD 数据转换	DWORD 型 BCD 数据的转换包括以下三个指令:D_BCD_TO_SINT,

	D_BCD_TO_INT 和 D_BCD_TO_DINT。这三个指令能将一个 DWORD 数据类型的 BCD（二进制编码的十进制数）输入值分别转换为一个 SINT、INT 和 DINT 数据类型的输出值。

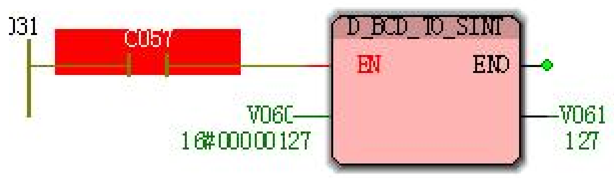
➢ **DWORD 型 BCD 数据的转换指令**

指令	输入值	输出值	描述
D_BCD_TO_SINT	DWORD 型 BCD 码	SINT	输入值 BCD 码 16#00000000 ~16#00000127，对应输出值 SINT 为 0~127，输入再增加则输出为-1;输入值 BCD 码 16#00000000~16#00032767，输出值 INT 为 032,767，输入再增加则输出为-1;输入值 BCD 码 16#00000000~16#99999999，输出值 DINT 0~99999999。
D_BCD_TO_INT	DWORD 型 BCD 码	INT	
D_BCD_TO_DINT	DWORD 型 BCD 码	DINT	

➢ 用法（以 **D_BCD_TO_SINT** 为例）

IL 编程语言	LD，FBD 编程语言
LD　IN	
D_BCD_TO_SINT	
ST　OUT	
ST 编程语言	
OUT:=D_ BCD_TO_ SINT　（IN）	
注：IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

➢ **功能和动作例子**

16#00000044 转为短整数	描述
	当触点 C057 为 ON 时；执行该指令;将地址为 V060 中的 16 进制的数转为短整型，存放到 V061 中； 执行的结果：16#0000127 转为短整数 127

10.3.4 BCD 型数据的转换

类型转换 FU	功能
CD 型数据的转换	BCD 型数据的转换指令 BCD_TO _DINT，用于将一个 BCD（二进制编码的十进制数）输入值分别转换为一个 DINT 数据类型的输出值。这条指令同 D_BCD_TO_DINT，详见 D_BCD_TO_DINT 的用法。

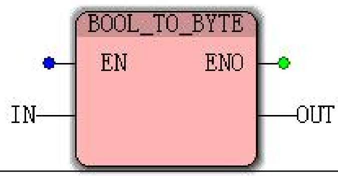
10.3.5 BOOL 型数据的转换

类型转换 FU	功能
BOOL 型数据的转换	BOOL 型数据的转换有 11 个指令，可以将 BOOL 型数据分别转换为 BYTE,WORD,DWORD, SINT, INT, DINT, USINT, UINT, UDINT, REAL 和 LREAL 等类型。

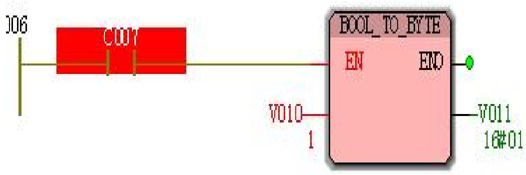
➢ **BOOL 型数据的转换指令**

指令	输入值	输出值	描述
BOOL_TO_BYTE	BOOL	BYTE	输入值取值范围 FALSE 或 TRUE 输入为 FALSE 时，输出为 0; 输入为 TRUE 时，输出为 1;
BOOL_TO_WORD	BOOL	WORD	
BOOL_TO_DWORD	BOOL	DWORD	
BOOL_TO_SINT	BOOL	SINT	
BOOL_TO_INT	BOOL	INT	
BOOL_TO_DINT	BOOL	DINT	
BOOL_TO_USINT	BOOL	USINT	
BOOL_TO_UINT	BOOL	UINT	
BOOL_TO_UDINT	BOOL	UDINT	
BOOL_TO_REAL	BOOL	REAL	
BOOL_TO_LREAL	BOOL	LREAL	

➢ **用法（以 BOOL_TO_BYTE 为例）**

IL 编程语言	LD, FBD 编程语言
LD IN	
BOOL_TO_BYTE	
ST OUT	
ST 编程语言	
OUT:=BOOL_TO_BYTE （IN）	
注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

➢ **功能和动作例子**

将 BOOL 转为字节	描述
	当触点 C007 为 ON 时，执行该指令；将地址 V010 为 TRUE 时，转为字节型，存放到 V011 中； 执行的结果; V010（TRUE）转 16#01 存到 V011 中；

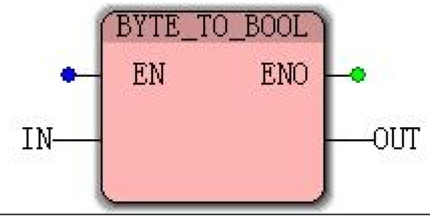
10.3.6 BYTE 型数据的转换

类型转换 FU	功能
BYTE 型数据的转换	BYTE 型数据的转换有 11 个指令，可以将 BYTE 型数据分别转换为 BOOL,WORD,DWORD, SINT, INT, DINT, USINT, UINT, UDINT, REAL 和 LREAL 等类型。

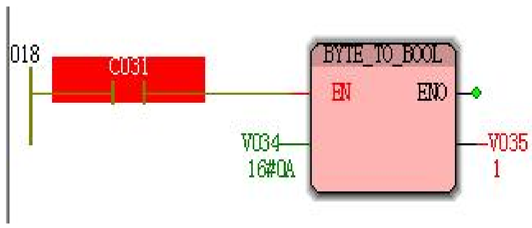
➢ **BYTE 型数据的转换指令**

指令	输入值	输出值	描述
BYTE_TO_BOOL	BYTE	BOOL	输入值取值范围 0~255; 输出为 BOOL 型:仅当输入为 0 时，输出为 FALSE，其它情况下输出均为 TRUE; 输出为 SINT 型:输入 0~127 对应输出 0~127，输入 128~255 对应输出 -128~-1; 输出为 WORD，DWORD 等其它类型时，输出等于 输入。
BYTE _O_WORD	BYTE	WORD	
BYTE_TO_DWORD	BYTE	DWORD	
BYTE_TO_SINT	BYTE	SINT	
BYTE_TO_INT	BYTE	INT	
BYTE_TO_DINT	BYTE	DINT	
BYTE_TO_USINT	BYTE	USINT	
BYTE_TO_UINT	BYTE	UINT	
BYTE_TO_UDINT	BYTE	UDINT	
BYTE_TO_REAL	BYTE	REAL	
BYTE_TO_LREAL	BYTE	LREAL	

➢ 用法（以 **BYTE_TO_BOOL** 为例）

IL 编程语言	LD， FBD 编程语言
LD IN	
BYTE_TO_BOOL	
ST OUT	
ST 编程语言	
OUT:=BYTE_TO_BOOL （IN）	
注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

➢ **功能和动作例子**

求：16#0A（16 进制数）转 BOOL	描述
	当触点 C031 为 ON 时；执行该指令； 将地址为 V034 中的 16 进制的数转为 BOOL，存放到 V035 中； 执行的结果：16#OA 转 BOOL 型 注意： 只要 V034 中的数不为零，V035 输出 1（TRUE）

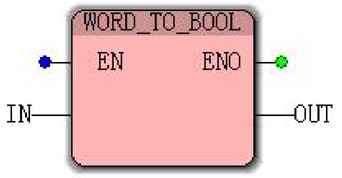
10.3.7 WORD 型数据的转换

类型转换 FU	功能
WORD 型数据的转换	WORD 型数据的转换有 11 个指令，可以将 WORD 型数据分别转换为 BOOL,BYTE,DWORD, SINT, INT, DINT, USINT, UINT, UDINT, REAL 和 LREAL 等类型。

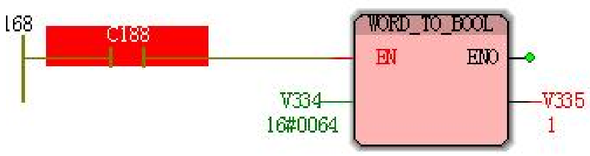
➢ **WORD 型数据的转换指令**

指令	输入值	输出值	描述
WORD_TO _BOOL	WORD	BOOL	输入值取值范围 0~65,535; 输出为 BOOL 型:仅当输入为 0 时，输出为 FALSE, 其它情况下输出均为 TRUE; 输出为 SINT 型:输入 0~127 对应输出 0~127，输入 128~255 对应输出-128~-1，输入再增加则输出将重复 0~127，-128~-1; 输出为 USINT、BYTE 型:输入 0~255 对应输出 0~ 255，输入再增加则输出将重复 0~255;输出为 INT 型:输入 0~32767 对应输出 0 ~32767, 输入 32768~65535 对应输出 -32768~-1;输出为 WORD, DWORD 等其它类型时，输出等于 输入。
WORD_TO_BYTE	WORD	BYTE	
WORD_TO_DWORD	WORD	DWORD	
WORD_TO_SINT	WORD	SINT	
WORD_TO_INT	WORD	INT	
WORD_TO_DINT	WORD	DINT	
WORD_TO_USINT	WORD	USINT	
WORD _TO_UINT	WORD	UINT	
WORD _TO_UDINT	WORD	UDINT	
WORD _TO_REAL	WORD	REAL	
WORD_TO_LREAL	WORD	LREAL	

➢ 用法（以 **WORD_TO_BOOL** 为例）

IL 编程语言	LD, FBD 编程语言
LD IN	
WORD_TO_BOOL	
ST OUT	
St 编程语言	
OUT:=WORD_ TO_BOOL （IN）	
注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

➢ **功能和动作例子**

16#0064 转 BOOL 型	描述
 	当触点 C188 为 ON 时，执行该指令，将地址为 V334 中的字转为 BOOL 型，存放到 V335 中； 执行的结果：16#0064 转 BOOL

	注意：当 V334 的值不为零时，V335 输出为 TRUE（1）

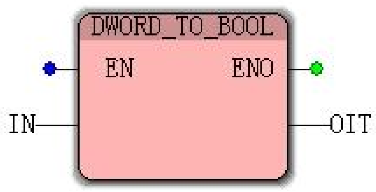
10.3.8 DWORD 型数据的转换

类型转换 FU	功能
DWORD 型数据的转换	DWORD 型数据的转换有 11 个指令，可以将 DWORD 型数据分别转换为 BOOL,BYTE,WORD, SINT, INT, DINT, USINT, UINT, UDINT, REAL 和 LREAL 等类型。

➢ **DWORD 型数据的转换指令**

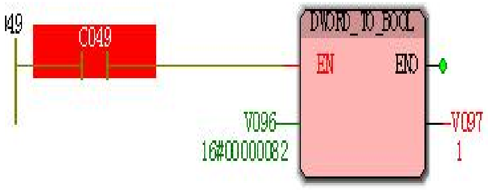
指令	输入值	输出值	描述
DWORD_TO_BOOL	DWORD	BOOL	输入值取值范围 0~4,294,967,295;输出为 BOOL 型:仅当输入为 0 时，输出为 FALSE,其它情况下输出均为 TRUE; .输出为 SINT 型:输入 0~127 对应输出 0~127， 输入 128~255 对应输出 -128~-1， 输入再增加则输出 将重复 0~127，-128~-1;输出为 USINT、BYTE 型:输入 0~255 对应输出 0~255，输入再增加则输出将重复 0~255;输出为 INT 型:输入 0~32767 对应输出 0~32767, 输入 32768~65535 对应输出-32768~-1，输入再增加则输出将重复 0~32767，-32768~-1;;输出为 UINT、WORD 型:输入 0~65535 对应输出 0 ~65535，输入再增加则输出将重复 0~65535;输出为 BYTE,WORD 等类型时，输出分别等于 输入的低 8 位、低 16 位数据。
DWORD_TO_BTTE	DWORD	BYTE	
DWORD_TO_WORD	DWORD	WORD	
DWORD_TO_SINT	DWORD	SINT	
DWORD_TO_INT	DWORD	INT	
DWORD_TO_DINT	DWORD	DINT	
DWORD_TO_USINT	DWORD	USINT	
DWORD_TO_UINT	DWORD	UINT	
DWORD_TO_UDINT	DWORD	UDINT	
DWORD_TO_REAL	DWORD	REAL	
DWORD_TO_LREAL	DWORD	LREAL	

➢ 用法（以 **DWORD_TO_BOOL** 为例）

IL 编程语言	**LD, FBD 编程语言**
LD IN	
DWORD_TO_BOOL	
ST OUT	
ST 编程语言	
OUT:=DWORD_TO_BOOL （IN）	
注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

➢ **功能和动作例子**

16#00000082 转 BOOL	描述

	当触点 C049 为 ON 时，执行该指令； 将地址为 V096 中的 DWORD 数转 BOOL 型，存放到 V097 中； 执行的结果：16#0000082 转为 BOOL 型 注意： 只要 V096 中的值不为零时，输出 1（TRUE）

10.3.9 SINT 型数据的转换

类型转换 FU	功能
SINT 型数据的转换	SINT 型数据的转换有 14 个指令，可以将 SINT 型数据分别转换为 B_BCD,W_BCD, D_BCD、BOOL, BYTE、WORD、DWORD, INT, DINT, USINT, UINT, UDINT, REAL 和 LREAL 等类型。

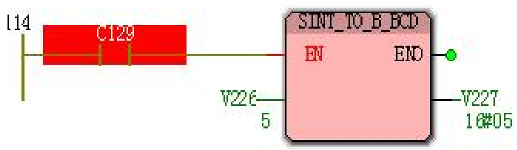
➢ **SINT 型数据的转换指令**

指令	输入值	输出值	描述 输入值取值范围-128~127;
SINT_TO_B_BCD	SINT	BYTE	输入 0~99 时，输出 16#0~99;输入其它值时，输出 16#FF
SINT_TO_W_BCD	SINT	WORD	输入 0~127 时,输出 16#0~127;输入其它值时，输出 16#FFFF
SINT_TO_D_BCD	SINT	DWORD	输入 0~127 时，输出 16#0~127，输入其它值时，输出 16#FFFFFFFF
SINT_TO_BOOL	SINT	BOOL	输入 0 时，输出 FALSE;输入其它值时，输出 TRUE
SINT_TO_BYTE	SINT	BYTE	输入 0~127 时，输出 0~127;输入-128~-1，输出 128~255
SINT_TO_WORD	SINT	WORD	输入 0~127 时，输出 0~127;输入-128~-1，输出 128~255
SINT_TO_DWORD	SINT	DWORD	输入 0~127 时，输出 0~127;输入-128~-1，输出 128~255
SINT_TO_INT	SINT	INT	输入 0~127 时，输出 0~127;输入-128~-1，输出 128~255
SINT_TO_DINT	SINT	DINT	输入 0~127 时，输出 0~127;输入-128~-1，输出 128~255
SINT_TO_USINT	SINT	USINT	输入 0~127 时，输出 0~127;输入-128~-1，输出 128~255
SINT_TO_UINT	SINT	UINT	输入 0~127 时，输出 0~127;输入-128~-1，输出 128~255
SINT _TO_UDINT	SINT	UDINT	输入 0~127 时，输出 0~127;输入-128~-1，输出 128~255
SINT _TO_REAL	SINT	REAL	输入 0~127 时，输出 0~127;输入-128~-1，输出-128~-1
SINT_TO_LREAL	SINT	LREAL	输入 0~127 时，输出 0~127;输入-128~-1，输出-128~-1

➢ 用法（以 **SINT_TO_B_BCD** 为例）

<table>
<tr><th>IL 编程语言</th><th>LD, FBD 编程语言</th></tr>
<tr><td>LD IN</td><td rowspan="5">
</td></tr>
<tr><td>SINT_TO_B_BCD</td></tr>
<tr><td>ST OUT</td></tr>
<tr><td>ST 编程语言</td></tr>
<tr><td>OUT:=SINT_TO_B_BCD （IN）</td></tr>
<tr><td colspan="2">注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量</td></tr>
</table>

➢**功能和动作例子**

短整型 5 转为 BCD16#05	描述
	这个类型转换功能将数据类型 SINT 的输入值转换为数据类型 BYTE 的 BCD 输出值（二进制码十进制数）。

10.3.10 INT 型数据的转换

类型转换 FU	功能
INT 型数据的转换	INT 型数据的转换有 14 个指令，可以将 INT 型数据分别转换为日 BCD,WBCD, D BCD,BOOL, BYTE、WORD、DWORD、SINT, DINT, USINT, UINT, UDINT, REAL 和 LREAL 等类型。

➢ **INT 型数据的转换指令**

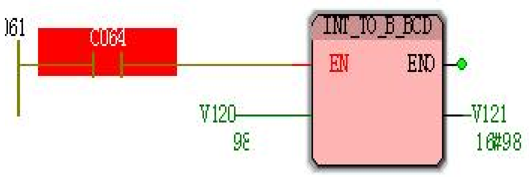
指令	输入值	输出值	描述 输入值取值范围一 32768~32767;
INT_TO_B_BCD	INT	BYTE	输入 0~99 时，输出 16#0~99;输入其它值时，输出 16#FF
INT_TO_W_BCD	INT	WORD	输入 0~9999 时，输出 16#0~9999;输入其它值时，输出 16#FFFF
INT_TO_D_BCD	INT	DWORD	输入 0~32767 时，输出 16#0 ~32767，输入其它值时，输 16#FFFFFFFF
INT _TO_BOOL	INT	BOOL	输入 0 时，输出 FALSE;输入其它值时，输出 TRUE
INT_TO_BYTE	INT	BYTE	输入 0~255 时，输出 0~255;输入再增加则输出将重复 0~255;输入-1~255，输出 255~0;输入再减小则输出重复 255~0
INT_TO_WORD	INT	WORD	输入 0~32767 时，输出 0~32767;输入-32768~-1，输出 32768~65535
INT_TO_DWORD	INT	DWORD	输入 0~32767 时，输出 0~32767;输入-32768-~1，输

			出 32768~65535
INT_TO_SINT	INT	SINT	输入 0~127 时， 输出 0~127;输入 128~255， 输出-128-1;输入再增加则输出重复 0~127, -128~-1;输入-1~-128，输出-1~-128;输入-129~-256，输出 127~输入再减小则输出重复-1~-128, 127~0
INT_TO_DINT	INT	DINT	输入-32768~32767 时， 输出-32768 ~32767
INT_TO_USINT	INT	USINT	输入 0~255 时，输出 0~255;输入再增加则输出重复 0 ~255
INT_TO_UINT	INT	UINT	输入 0~32767 时，输出 0~32767;输入-32768~-1，输出 32768~65535
INT_TO_UDINT	INT	UDINT	输入 0~32767 时，输出 0~32767;输入-32768~-1，输出 32768 一 65535
INT_TO_REAL	INT	REAL	输入 0~32767 时，输出 0~32767;输入-32768~-1，输出-32768~-1
INT_TO_LREAL	INT	LREAL	输入 0~32767 时，输出 0~32767;输入-32768~-1，输出-32768~-1

➢ 用法（以 **INT_TO_B_BCD** 为例）

IL 编程语言	LD, FBD 编程语言
LD IN	
INT_TO_B_BCD	
ST OUT	
St 编程语言	
OUT:=INT_TO_B_BCD （IN）	
注：IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

➢ **功能和动作例子**

整型 98 转为 BYTE	描述
	当触点 C064 为 ON 时，执行该指令，将地址为 V006 中的整数 98 转为 BYTE 型，存放到 V121 中； 执行的结果：98 整型转 16#98（BYTE）

10.3.11 DINT 型数据的转换

类型转换 FU	功能
DINT 型数据的转换	DINT 型数据的转换有 16 个指令，可以将 DINT 型数据分别转换为 B_BCD,W_BCD,D_BCD、BOOL, BYTE、WORD、DWORD, SINT、INT, USINT, UINT, UDINT, REAL,LREAL, BCD 和 TIME 等类型。

➢ **DINT 型数据的转换指令**

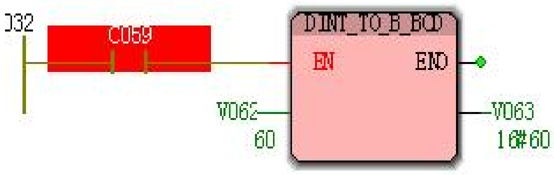
指令	输入值	输出值	描述：输入值取值范围 -2,147,483,648...2,147,483,647
DINT_TO_B_BCD	DINT	BYTE	输入 0~99 时，输出 16#0~99;输入其它值时，输出 16#FF
DINT_TO_W_BCD	DINT	WORD	输入 0~9999 时，输出 16#0~9999;输入其它值时，输出 16#FFFF
DINT_TO_D_BCD	DINT	DWORD	输入 0~99999999 时，输出 16#0~99999999，输入其它值时，输出 16#FFFFFFF
DINT_TO_BOOL	DINT	BOOL	输入 0 时，输出 FALSE;输入其它值时，输出 TRUE
DINT_TO_BYTE	DINT	BYTE	输入 0~255 时，输出 0~255;输入再增加则输出将重复 0~255;输入-1~-255，输出 255~0;输入再减小则输出重复 2550
DINT _TO_WORD	DINT	WORD	输入 0~65535 时，输出 0~65535;输入再增加则输出重复 0~65535;输入-1~-65536，输出 65535~0;输入再减小则输出重复 65535~0
DINT_TO_DWORD	DINT	DWORD	输入 0~2,147,483,647 时，输出 0~2,147,483,647;输入 -2,147,483,648~-1，输出 2,147,483,648~4,294,967,295
DINT_TO_SINT	DINT	SINT	输入 0~127 时，输出 0~127;输入 128~255，输出 -128~-1;输入再增加则输出重复 0~127, -128~-1;输入-1 ~-128，输出-1~-128;输入-129~-256，输出 127~0;输入再减小则输出重复-1~-128, 127~0
DINT _TO-INT	DINT	INT	输入-32768 ~32767 时，输出-32768 ~32767;输入大于 32767 时，输出重复-32768~32767;输入小于-32768 是，输出重复 32767-32768
DINT_TO_USINT	DINT	USINT	输入 0~255 时，输出 0~255;输入再增加则输出重复 0~255
DINT_TO_UINT	DINT	UINT	输入 0~65535 时，输出 0~65535;输入再增加则输出重复 0 ~65535;输入-65536~-1，输出 0~ 65535;输入再减小则输出重复 65535~0
DINT-TO_UDINT	DINT	UDINT	输入 0~2,147,483,647 时，输出 0~2,147,483,647;输入 -2,147,483,648~-1，输出 2,147,483,648~4.294.967.295
DINT_TO_REAL	DINT	REAL	输入 0~2,147,483,647 时，输出 0~2,147,483,647;

			输入-2,147,483,648~-1，输出-2,147,483,648~-1，精度会降低
DINT_TO_LREAL	DINT	LREAL	输入 0~2,147,483,647 时，输出 0~2,147,483,647;输入-2,147,483,648~-1，输出-2,147,483,648~-1，精度会降低
DINT_TO_BCD	DINT	BCD	输入值 DINT 0~99999999，输出值 BCD 码 16#00000000~16#99999999。
DINT_TO_TIME	DINT	TIME	输出值单位为秒;输入值 0~2,147,483,647，输出值 0~2147483.647 秒;输入值-2147483648~-1 秒，输出值 2147483.648~4294967.295 秒

➢ **用法（以 DINT_TO_B_BCD 为例）**

IL 编程语言	LD， FBD 编程语言
LD IN	
DINT_TO_B_BCD	
ST OUT	
ST 编程语言	
OUT:=DINT_TO_B_BCD （IN）	
注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

➢ **功能和动作例子**

双整数 60 转 BYTE	描述
	当触点 C059 为 ON 时，执行该指令，将地址为 V062 中的双整数 60 转为 BYTE 存放到 V063 中； 执行的结果：60（DINT）转 16#60（BYTE）

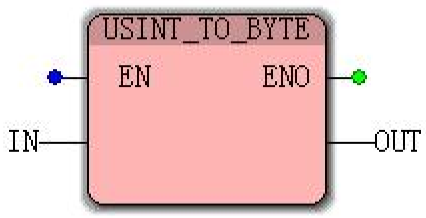
10.3.12 USINT 型数据的转换

类型转换 FU	功能
USINT 型数据的转换	USINT 型数据的转换有 11 个指令，可以将 USINT 型数据分别转换为 BOOL,BYTE, WORD, DWORD, SINT, INT, DINT, UINT, UDINT, REAL 和 LREAL 等类型。

➢ **USINT 型数据的转换指令**

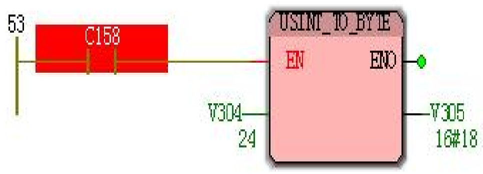
指令	输入值	输出值	描述输入值取值范围 0~255;
USINT_TO_BOOL	USINT	BOOL	输入 0 时，输出 FALSE;输入其它值时，输出 TRUE
USINT_TO_BYTE	USINT	BYTE	输入 0~255 时，输出 0~255
USINT_TO_WORD	USINT	WORD	输入 0~255 时，输出 0~255
USINT_TO_DWORD	USINT	DWORD	输入 0~255 时，输出 0~255
USINT_TO_SINT	USINT	SINT	输入 0~127 时，输出 0~127;输入 128~255，输出-128~-1
USINT_TO_INT	USINT	INT	输入 0~255 时，输出 0~255
USINT_TO_DINT	USINT	DINT	输入 0~255 时，输出 0~255
USINT_TO_UINT	USINT	UINT	输入 0~255 时，输出 0~255
USINT_TO_UDINT	USINT	UDINT	输入 0~255 时，输出 0~255
USINT_TO_REAL	USINT	REAL	输入 0~255 时，输出 0~255
USINT_TO_LREAL	USINT	LREAL	输入 0~255 时，输出 0~255

➢ 用法（以 **USINT_TO_BYTE** 为例） **IL** 编程语言

LD IN	LD, FBD 编程语言
USINT_TO_BYTE	
ST OUT	
ST 编程语言	
OUT:=USINT_TO_BYTE （IN）	
注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

➢ 功能和动作例子

24 （USINT）转 BYTE	描述

	当触点 C158 为 ON 时，执行该指令， 将地址为 V304 中的无符短整数 24 转为 BYTE，存放到 V305 中； 执行的结果：24（DINT）转 16#18（BYTE）

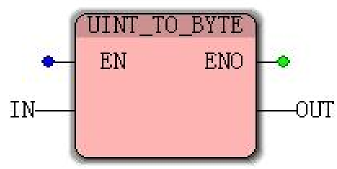
10.3.13 UINT 型数据的转换

类型转换 FU	功能
UINT 型数据的转换	UINT 型数据的转换有 11 个指令，可以将 UINT 型数据分别转换为 BOOL, BYTE,WORD,DWORD、SINT, INT, DINT, USINT, UDINT, REAL 和 LREAL 等类型。

➢ **UINT 型数据的转换指令**

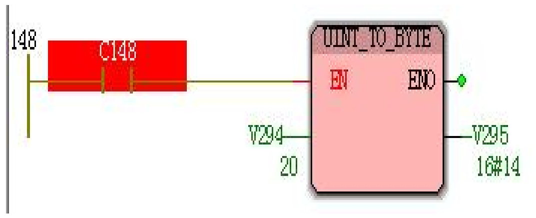
指令	输入值	输出值	描述输入值取值范围 0 65535
UINT _TO_BOOL	UINT	BOOL	输入 0 时，输出 FALSE;输入其它值时，输出 TRUE
UINT _TO_BYTE	UINT	BYTE	输入 0~255 时，输出 0~255;输入再增加则输出将重复 0~255
UINT_TO_WORD	UINT	WORD	输入 0~65535 时，输出 0~65535
UINT_TO_DWORD	UINT	DWORD	输入 0~65535 时，输出 0~65535
UINT _TO_SINT	UINT	SINT	输入 0~127 时，输出 0~127;输入 128~255，输出-128~-1;输入再增加则输出重复 0~127, -128~-1
UINT _TO_INT	UINT	INT	输入 0~32767 时，输出 0~32767;输入 32768~65535,输出-32768~-1
UINT _TO_DINT	UINT	DINT	输入 0~65535 时，输出 0~65535
UINT _TO_USINT	UINT	USINT	输入 0~255 时，输出 0~255;输入再增加则输出重复 0~255
UINT _TO_UDINT	UINT	UDINT	输入 0~65535 时，输出 0~65535
UINT_TO_REAL	UINT	REAL	输入 0~65535 时，输出 0~65535
UINT _TO_LREAL	UINT	LREAL	输入 0~65535 时，输出 0~65535

➢ 用法（以 **UINT_TO_BYTE** 为例）

IL 编程语言	LD, FBD 编程语言
LD IN	
UINT TO BYTE	
ST OUT	
St 编程语言	
OUT:=UINT_T0_BYTE （IN）	

注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量

➢ 功能和动作例子

无符整型 20 转 BYTE	描述
	当触点 C148 为 ON 时，执行该指令，将地址为 V294 中的无符整数 20 转为 BYTE 存放到 V295 中； 执行的结果：20（UINT）转 16#14（BYTE）

10.3.14 UDINT 型数据的转换

类型转换 FU	功能
UDINT 型数据的转换	UDINT 型数据的转换有 11 个指令，可以将 UDINT 型数据分别转换为 BOOL,BYTE,WORD, DWORD, SINT, INT, DINT, USINT, UINT, REAL 和 LREAL 等类型。

➢ **UDINT 型数据的转换指令**

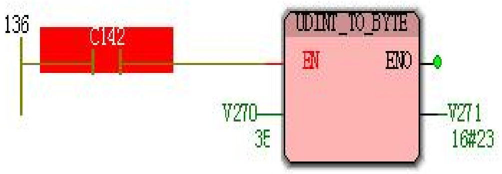
指令	输入值	输出值	描述输入值取值范围 04,294,967,295
UDINT_TO_BOOL	UDINT	BOOL	输入 0 时，输出 FALSE;输入其它值时，输出 TRUE
UDINT _TO_BYTE	UDINT	BYTE	输入 0~255 时，输出 0~255;输入再增加则输出将重复 0~255
UDINT _TO_WORD	UDINT	WORD	输入 0~65535 时，输出 0~65535，输入再增加则输出将重复 0~65535
UDINT_TO_DWORD	UDINT	DWORD	输入 0~4,294,967,295 时，输出 0~4,294,967,295
UDINT_TO_SINT	UDINT	SINT	输入 0~127 时，输出 0~127;输入 128~255，输出 -128~-1; 输入再增加则输出重复 0~127, -128~-1
UDINT_TO_INT	UDINT	INT	输入 0~32767 时，输出 0~32767; 输入 32768~65535 输出-32768~-1，输入再增加则输出重复 0~32767,-32768~-1
UDINT_TO_DINT	UDINT	DINT	输入 0~2,147,483,647 时，输出 0~2,147,483,647;入 2,147,483,648~4,294,967,295 ， 输 -2,147,483,648~-1
UDINT_TO_USINT	UDINT	USINT	输入 0~255 时，输出 0~255;输入再增加则输出重复 0~255
UDINT_TO_UINT	UDINT	UDINT	输入 0~65535 时，输出 0~65535;输入再增加则输出重复 0~65535
UDINT_TO_REAL	UDINT	REAL	输入 0~4,294,967,295 时，输出 0~4,294,967,295，精度会降低

UDINT _TO_LREAL	UDINT	LREAL	输入 0~4,294,967,295 时，输出 0~4,294,967,295，精度会降低

➢ 用法（以 **UDINT_TO_BYTE** 为例）

IL 编程语言	**LD, FBD 编程语言**
LD IN	
UDINT_TO_BYTE	
ST OUT	
ST 编程语言	
OUT:=UDINT_TO_BYTE （IN）	
注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

➢ 功能和动作例子

无符双整 35 转 BYTE	描述
	当触点 C142 为 ON 时，执行该指令，将地址为 V270 中的无符双整数 35 转为 BYTE，存放到 V271 中； 执行的结果：35（UDINT）转 16#23（BYTE）

10.3.15 REAL 型数据的转换

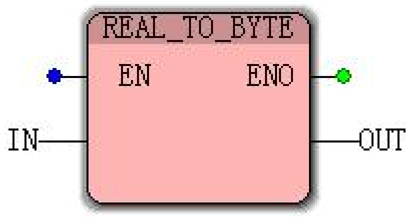
类型转换 FU	功能
REAL 型数据的转换	REAL 型数据的转换有 11 个指令，可以将 REAL 型数据分别转换为 BOOL,BYTE,WORD,DWORD、SINT, INT, DINT, USINT, UINT, UDINT 和 LREAL 等类型。

➢ **REAL 型数据的转换指令**

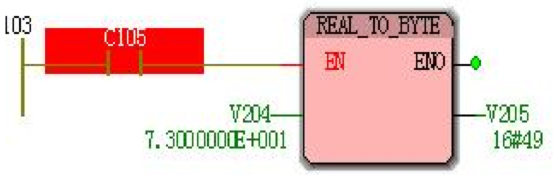
指令	输入值	输出值	描述 REAL 型的输入值其小数部分在转换时被舍去
REAL_TO_BOOL	REAL	BOOL	输入 0 时，输出 FALSE;输入其它值时，输出 TRUE
REAL_TO_BYTE	REAL	BYTE	输入 0~255 时，输出 0~255;输入再增加则输出将重复 0~255;输入-1~-255，输出 255~0;输入再减小则输出重复 255~0
REAL_TO_WORD	REAL	WORD	输入 0~65535 时，输出 0~65535，输入再增加则

			输出重复 0~65535; 输入 -1~-65536， 输出 65535~0，输入再减小则输出重复 65535~0
REAL_TO_DWORD	REAL	DWORD	输入 0~2,147,483,647 时，输出 0~2,147,483,647，输入再增加输出为 2,147,483,647; 输入 -1~-2,147,483,648 ， 输 出 4,294,967,295~2,147,483,648，输入再减小输出仍为 2,147,483,648;精度会降低
REAL_TO_SINT	REAL	SINT	输入 0~127 时，输出 0~127;输入 128~255，输出 -128~-1;输入再增加则输出重复 0~127, -128~-1;输入-1 ~-128，输出-1~-128;输入-129~-256，输出 127~0;输入再减小则输出重复-1~128, 127~0
REAL _TO_INT	REAL	DINT	输入 0~32767 时， 输出 0~32767; 输入 32768~65535}输出-32768~-1;输入再增加则输出重复 0~32767,-32768~-1;输入-1~-32768，输出 -1~-32768;输入-32769~-65536，输出 32767~0;输入再减小则输出重复-1~-32768, 32767~0
REAL _TO_DINT	REAL	DINT	输入 0~2,147,483,647 时，输出 0~2,147,483 入再增加输出仍为 2,147,483,647-1 ~-2,147,483,648，输出-1~-2,147,483,648,输入再减小输出仍为 -2,147,483,648
REAL_TO_USINT	REAL	USINT	输入 0~255 时，输出 0~255，输入再增加则输出重复 0~255;输入-1~-256，输出 255~0，输出再减小则输出重复 255~0
REAL_TO_UINT	REAL	UINT	输入 0~65535 时，输出 0~65535，输入再增加则输出重复 0~65535; 输入 -1~-65536， 输出 65535~0，输入再减小则输出重复 65535~0
REAL_TO_UDINT	REAL	UDINT	输入 0~2,147,483,647 时，输出 0~2,147,483,647，输入再增加输出为 2,147,483,647; 输入 -1 ~-2,147,483,648 ， 输 出 4,294,967,295~2,147,483,648，输入再减小输出仍为 2,147,483,648;精度会降低
REAL_TO_LREAL	REAL	LREAL	输入等于输出

➢ 用法（以 **REAL_TO_BYTE** 为例）

IL 编程语言	LD, FBD 编程语言
LD IN	
REAL_TO_BYTE	
ST OUT	
ST 编程语言	
OUT:=REAL_TO_BYTE （IN）	
注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

➢ 功能和动作例子

浮点数 73.0 转 BYTE	描述
	当触点 C105 为 ON 时，执行该指令，将地址为 V204 中的浮点数 73.0 转为 BYTE 存放到 V205 中； 执行的结果：73.0（REAL）转 16#49（BYTE）

10.3.16 LREAL 型数据的转换

类型转换 FU	说明
LREAL 型数据的转换	LREAL 型数据的转换有 11 个指令，可以将 LREAL 型数据分别转换为 BOOL,BYTE,WORD, DWORD, SINT, INT, DINT, USINT, UINT, UDINT 和 REAL 等类型。

➢ **LREAL 型数据的转换指令**

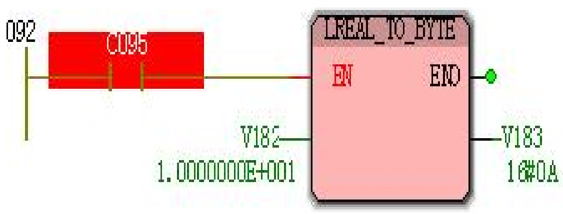
指令	输入值	输出值	描述 REAL 型的输入值其小数部分在转换时被舍去
LREAL_TO_BOOL	LREAL	BOOL	输入 0 时，输出 FALSE;输入其它值时，输出 TRUE
LREAL_TO_BYTE	LREAL	BYTE	输入 0~255 时，输出 0~255;输入再增加则输出将重复 0~255;输入-1~-255，输出 255~0;输入再减小则输出重复 255~0
LREAL_TO_WORD	LREAL	WORD	输入 0~65535 时，输出 0~65535，输入再增加则输出重复 0~65535;输入-1~-65536，输出 65535~0，输入再减小则输出重复 65535~0
LREAL_TO_DWORD	LREAL	DWORD	输入 0~2,147,483,647 时，输出 0~2,147,483,647，输入再增加输出为 2,147,483,647;输入-1~-2,147,483,648，输出 4,294,967,295~2,147,483,648，输入再减小输出仍为 2,147,483,648;精度会降低
LREAL_TO_SINT	LREAL	SINT	输入 0~127 时，输出 0~127;输入 128~255，输出-128~-1;输入再增加则输出重复 0~127, -128-1;输入-1~-128，输出-1~-128;输入-129~-256，输出 127~0;输入再减小则输出重复-1~-128, 127~0
LREAL_TO_INT	LREAL	DINT	输入 0~32767 时，输出 0~32767;输入 32768~65535 输出-32768~-1;输入再增加则输出重复 0~32767 -32768~-1;输入-1~-32768，输出

			-1~-32768;输入-32769~-65536，输出 32767~0;输入再减小则输出重复-1~-32768, 32767~0
LREAL_TO_DINT	LREAL	DINT	输入 0~2,147,483,647 时，输出 0~2,147,483647 入再增加输出仍为 2,147,483,647; 输入-1~-2,147,483,648，输出-1~-2,147,483,648 输入再减小输出仍为-2,147,483,648
LREAL_TO_USINT	LREAL	USINT	输入 0~255 时，输出 0~255，输入再增加则输出重复 0~255;输入-1~-256，输出 255~0，输出再减小则输出重复 255~0
LREAL_TO_UINT	LREAL	UINT	输入 0~65535 时，输出 0~65535，输入再增加则输出重复 0~65535;输入-1~-65536，输出 65535~0，输入再减小则输出重复 65535~0
LREAL_TO_UDINT	LREAL	UDINT	输入 0~2,147,483,647 时，输出 0~2,147,483,647，输入再增加输出为 2,147,483,647; 输入-1~-2,147,483,648，输出 4,294,967,295~2,147,483,648，输入再减小输出仍为 2,147,483,648;精度会降低
LREAL_TO_LREAL	LREAL	REAL	输入等于输出，精度会降低

➢ **用法（以 LREAL_TO_BYTE 为例）**

<table>
<tr><th>IL 编程语言</th><th>LD, FBD 编程语言</th></tr>
<tr><td>LD IN</td><td rowspan="5">
</td></tr>
<tr><td>LREAL_TO_BYTE</td></tr>
<tr><td>ST OUT</td></tr>
<tr><td>ST 编程语言</td></tr>
<tr><td>OUT:=LREAL_TO_BYTE （IN）</td></tr>
<tr><td colspan="2">注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量</td></tr>
</table>

➢**功能和动作例子**

<table>
<tr><th>长浮点数 10.0 转 BYTE</th><th>描述</th></tr>
<tr><td>
</td><td>当触点 C095 为 ON 时，执行该指令时，将地址为 V182 中的长浮点数 10.0 转为 BYTE，存放到 V183 中；
执行的结果：10.0（LREAL）转 16#0A（BYTE）</td></tr>
</table>

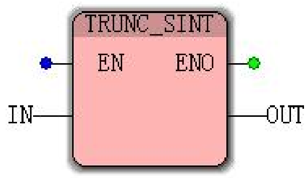
10.3.17 TRUNC 小数取整

类型转换 FU	功能
TRUNC 小数取整	TRUNC 小数取整有 4 个指令，可以将浮点型数据分别转换为 SINT、INT、DINT 等类型。

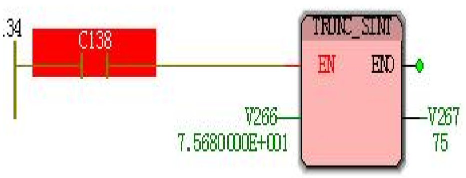
➢ **RUNC 小数取整指令**

指令	输入值	输出值	描述
TRUNC	REAL	无符号整型	截去位于输入值的小数部分，得到一个整型值
TRUNC_SINT	REAL	SINT	输入 0~127 时，输出 0~127;输入 128~255，输出-128~-1;输入再增加则输出重复 0~127, -128~-1；输入一1~-128，输出-1~-128;输入-129~-256，输出 127~0;输入再减小则输出重复-1~-128, 127~0
TRUNC_INT	REAL	INT	输入 0 ~32767 时,输出 0~32767;输入 32768~ 65535，输出-32768~-1;输入再增加则输出重复 0~32767，-32768~-1; 输入 -1~-32768，输出-1~-32768;输入-32769~-65536，输出 32767~0;输入再减小则输出重复-1~-32768, 32767~0
TRUNC _DINT	REAL	DINT	输入 0~2,147,483,647 时，输出 0~2,147,483,647,输入再增加输出仍为 2,147,483,647;输入-1~-2,147,483,648，输出-1~-2,147,483,648，输入再减小输出仍为-2,147,483,648

➢ **用法（以 TRUNC_SINT 为例）**

IL 编程语言	LD, FBD 编程语言
LD IN	
TRUNC_SINT	
ST OUT	
ST 编程语言	
OUT:=TRUNC_ SINT （IN）	
注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

➢ **功能和动作例子**

浮点数 75.68 取整	描述
	当触点 C138 为 ON 时，执行该指令，将地址为 V266 中的浮点数 75.68 取整 存放到 V267 中； 执行的结果：75.69 转为 75 （注意：该功能是取整，小数点后的数舍去）

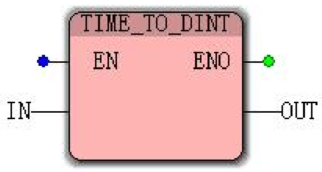
10.3.18 TIME 型数据的转换

类型转换 FU	功能
TIME_TO _DINT 类型转换	TIME_TO _DINT 类型转换功能将一个 TIME 型的输入值转换为一个 DINT 型的输出值(任何时间值都将换算为毫秒值再将毫秒值转换为 DINT）。TIME 型数据必须是以 T#为开始的一个无符号数，大于#2147483647 的时间值都将为负数，因为 DINT 类型为有符号数，其最大值为 2,147,483,647，例如，输入值 T#4294967295 毫秒将导致输出值为-1。

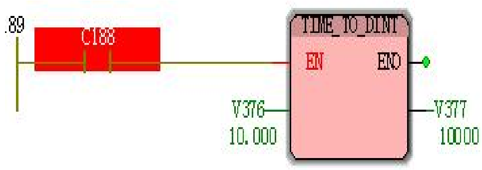
➢ **TIME_TO_DINT 转换指令处理的数据类型**

指令	输入值	输出值	描述
TIME_TO_DINT	TIME	DINT	输入值取值范围 T#0~2147483647M S（等于 T#0~2147483.647S，转换为 DINT 输出值为 0~2147483647; 输入值取值范围 T#2147483648~4294967295，转换为 DINT 输出值为 -2147483648~-1。

➢ 用法

IL 编程语言	LD, FBD 编程语言
LD IN	
TIME_TO_DINT	
ST OUT	
ST 编程语言	
OUT:=TIME_TO_DINT （IN）	
注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

➢ **功能和动作例子**

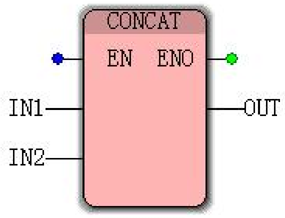
将 T#10s 转成双整数	描述
 	当触点 C188 为 ON 时，执行该指令，将地址为 V376 中的时间数值（T#10s）转为 DINT，存放到 V277 中； 执行的结果：T#10S（TIME）转成 10000（DINT） 注意（时间单位转 DINT 之间的转化为 1000；（例 1S---1000）

10.4 字符串 FU

字符串功能，简称字符串 FU，是对于字符串的比较、转换、查找、连接等操作，它带有多个输入参数和一个输出参数。MULTIPROG 编程过程中，可以在编辑向导中，从下拉列表选择”字符串 FU”，使用以下指令

分类	功能		
合并插入删除替换长度限制	CONCAT	INSERT	DELEtE
	REPLACE	LEN	LI MIT_STRING
	FIN D		
大小与位置	MAX_STRING	M IN_STRING	LEFT
	MID	RIGHT	SEL_STRING
比较	GT_STRING	GE_STRING	EQ_STRING
	NE_STRING	LE STRING	LT_STRING
字符串转换为其它	STRING_TO_BYTE	STRING_TO_WORD	STRING_TO_DWORD
	STRING_TO_SINT	STRING_TO_INT	STRING_TO_DINT
	STRING_TO_USINT	STRING_TO_UINT	STRING_TO_UDINT
	STRING_TO_LREAL	STRING_TO_REAL	STRING_TO_TIME
其它转换为字符串	BYTE_TO_STRING	WORD_TO_STRING	DWORD_TO_STRING
	SINT_TO_STRING	INT_TO_STRING	DINT_TO_STRING
	USINT_TO_STRING	UINT_TO_STRING	UDINT_TO_STRING
	REAL_TO_STRING	LREAL_TO_STRING	TIME_TO_STRING

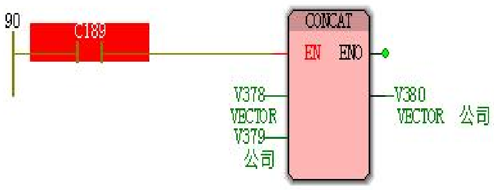
10.4.1 CONCAT（合并字符串）

<table>
<tr><th>IL 编程语言</th><th>LD, FBD 编程语言</th></tr>
<tr><td colspan="2">功能：CONCAT 合并字符串指令用于合并两个字符串。</td></tr>
<tr><td>LD IN</td><td rowspan="5">
</td></tr>
<tr><td>CONCAT</td></tr>
<tr><td>ST OUT</td></tr>
<tr><td>ST 编程语言</td></tr>
<tr><td>OUT:=CONCAT（ IN1，IN2）</td></tr>
<tr><td colspan="2">注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1， IN2 和 OUT 或使用常量</td></tr>
</table>

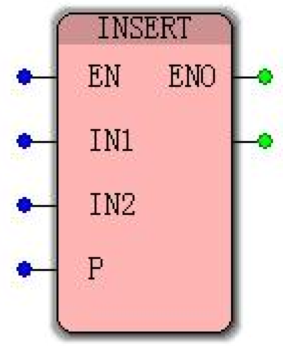
➢ **CONCAT 指令处理的数据类型**

输入	数据类型	描述
IN1	STRING	第一个输入
IN2	STRING	第二个输入
输出	数据类型	描述
OUT	STRING	输出，OUT=IN1 +IN2;第二个输入被添加到第一个输入的后面，输出不允许与输入同名

➢ **功能和动作例子**

<table>
<tr><th>“VECTOR 公司”</th><th>描述</th></tr>
<tr><td>
</td><td>当触点 C189 为 ON 时，执行该指令，将地址为 V378 中的字符串+地址 V379 中的字符串，存放到 V380 中；
执行的结果：VECTOR+公司=VECTOR 公司</td></tr>
</table>

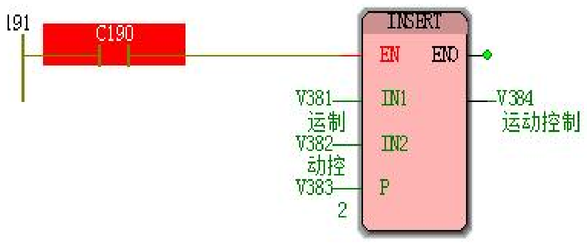
10.4.2 INSERT（插入字符串）

IL 编程语言	LD, FBD 编程语言
功能：INSERT 插入字符串指令用于将一个字符串或字符插入到另一个字符串中。 用法	
LD IN1 INSERT IN2, P ST OUT ST 编程语言 OUT:=INSERT（ IN1，IN2, P）	
注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1， IN2,P 和 OUT 或使用常量	

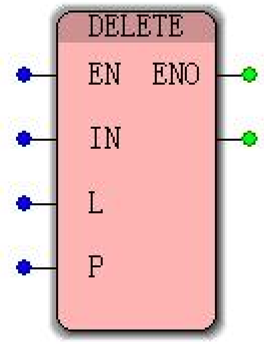
➢ **INSERT 指令处理的数据类型**

输入和输出	数据类型	描述
IN1	STRING	第一个输入字符串，使用变量时，变量必须设非空的初值，也可使用字符串常量
IN2	STRING	第二个输入字符串——将被插入到第一个输入中，使用变量时，变量必须设非空的初值，也可使用字符串常量
P	ANY_INT	在第一个输入中的插入点，P 必须是大于 0 的整数;字符串中的第一个字符位置为 1，后面的字符为 2, 3...,
OUT	STRING	输出第二个输入被插入到第一个输入中第 P 个字符的后面，输出不允许与输入同名

➢ **功能和动作例子**

将字符串”运控”插入到字符串”运制”中”运”字后	描述
	当触点 C190 为 ON 时，执行该指令，将地址为 V381 中的字符串插入到地址 V382 中的第一个字符串后，存放到 V384 中,P 表示插入第几个字符串后 执行的结果： 运制+动控=运动控制

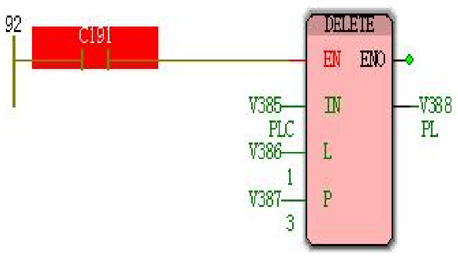
10.4.3 DELETE（册除字符串）

<table>
<tr><th>IL 编程语言</th><th>LD, FBD 编程语言</th></tr>
<tr><td colspan="2">功能：DELETE 删除字符串指令用于删除一个字符串中确定的几个相连的字符。</td></tr>
<tr><td>LD IN
DELETE L, P
ST OUT
ST 编程语言
OUT:=DELETE（ IN, L, P）</td><td>
</td></tr>
<tr><td colspan="2">注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN, L,P 和 OUT 或使用常量</td></tr>
</table>

➢ **DELETE 指令处理的数据类型**

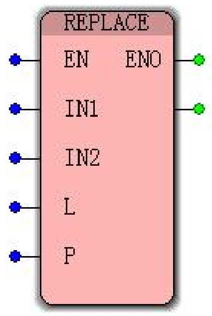
输入和输出	数据类型	描述
IN	STRING	输入字符串，也可使用常量字符串
L	ANY_INT	要删除的字符数，L 可以是 0, 1 , 2...
P	ANY_INT	在输入字符串中要删除的第一个字符的位置，P 必须是大于 0 的整数;字符串中的第一个字符位置为 1，后面的字符为 2, 3...
OUT	STRING	输出输入字符串不受删除的影响，实际上本指令是在输入字符串中挑选几个字符串形成一个新的字符串，输出不允许与输入同名

➢ **功能和动作例子**

<table>
<tr><th>将字符串”PLC”中第三个字符”C”删除</th><th>描述</th></tr>
<tr><td>
</td><td>当触点 C191 触点为 ON 时，执行该指令；删除地址 V385 的字符串中第三个字符，数量为 1 个；其中 L 为需要删除字符的个数，P 为要删除的第几个字符

执行的结果：PLC----PL</td></tr>
</table>

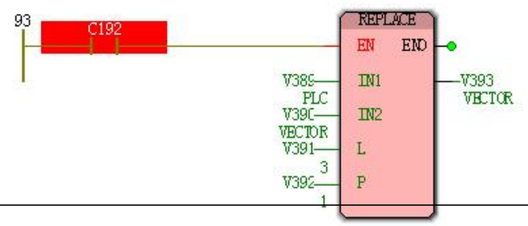
10.4.4 REPLACE（替换字符串）

<table>
<tr><th>IL 编程语言</th><th>LD， FBD 编程语言</th></tr>
<tr><td colspan="2">功能：REPLACE 替换字符串指令可用一个字符串替换一个字符串中确定的几个相连的字符。</td></tr>
<tr><td>LD IN
REPLACE L, P
ST OUT
ST 编程语言
OUT:=REPLACE（IN, L, P）</td><td>
</td></tr>
<tr><td colspan="2">注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1, IN2, L,P 和 OUT 或使用常量</td></tr>
</table>

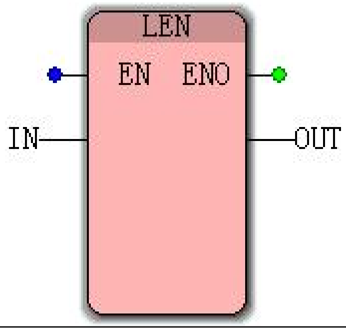
➢ **REPLACE 指令处理的数据类型**

输入和输出	数据类型	描述
IN1	STRING	第一个输入字符串
IN2	STRING	第二个输入字符串——将要替换第一个输入字符串中的某些字符
L	ANY_NT	要替换的字符数，L 可以是 0 ,1 , 2...
P	ANY_INT	在第一个输入字符串中要替换的第一个字符的位置，P 必须是大于 0 的整数;字符串中的第一个字符位置为 1，后面的字符为 2, 3...
OUT	STRING	输出第一个输入字符串不受替换的影响，实际上本指令是在第一个输入字符串中挑选出前后两部分字符与第二个字符串形成一个新的字符串，输出不允许与输入同名

➢ **REPLACE 功能和动作例子**

<table>
<tr><th>将字符串 “PLC”替换成字符串”VECTOR”</th><th>描述</th></tr>
<tr><td>
</td><td>当触点 C192 为 ON 时，执行该指令；将地址 V389 中的字符串，替换成地址 V390 的字符串，其中 P 为被替换字符串的第几个数开始被替换，L 为被替换字符个数。
执行的结果： PLC-----VECTOR</td></tr>
</table>

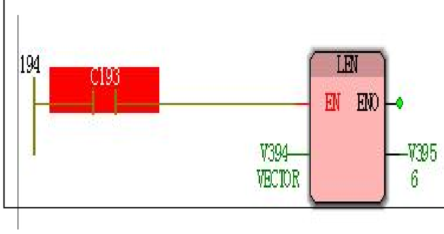
10.4.5 LEN（字符串长度）

IL 编程语言	LD, FBD 编程语言
功能：LEN 确定一个字符串的长度。	
LD IN LEN ST OUT	
ST 编程语言	
OUT:=LEN（IN）	
注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

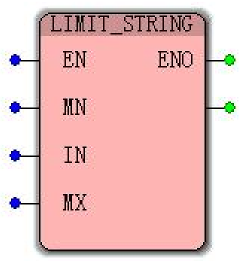
➢ **LEN 指令处理的数据类型**

输入和输出	数据类型	描述
IN	STRING	输入字符串
OUT	INT	输出，一个字符的长度为 1

➢ **功能和动作例子**

确定一个字符串的长度	描述
	当触点 C193 为 ON 时，执行该指令,将地址 V394 中的字符串的长度，以整数形式显示，存在地址 V395 中 执行的结果：VECTOR-------6

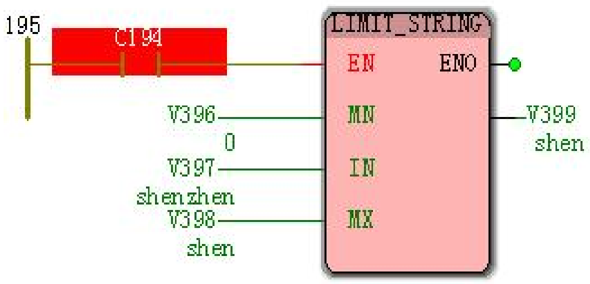
10.4.6 LIMIT_STRING（设置字符串极限）

IL 编程语言	LD, FBD 编程语言
功能：LIMIT_STRING 指令用于限制输入的字符或字符串在上下限所确定的区间。	
LD IN1 LIMIT_ STRING IN, IN2 ST OUT	
ST 编程语言	
OUT:=LIMIT_ STRING（IN1，IN, IN2）	
注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1， IN, IN2 和 OUT 或使用常量	

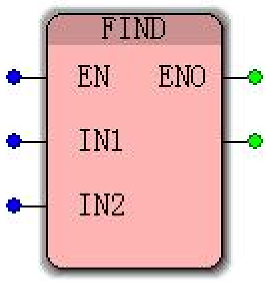
➢ **LIMIT_STRING 指令处理的数据类型**

输入和输出	数据类型	描述
IN1（MN）	STRING	字符下限
IN	STRING	输入字符或字符串
IN2（MX）	STRING	字符上限
OUT	STRING	输出，输入值中高于上限和低于下限的部分将分别等于上、下限;字符的大小均按其 ASCII 码值的大小判断;当输入字符串时，仅第一个字符参与比较，即如果第一个字符在上下限之间，则字符串全部输出，否则仅输出上限或下限

➢ **功能和动作例子**

限定字符串输出	描述
	当触点 C194 为 ON 时，执行该指令，当地址 V397 中的字符串长度在地址 V396，V398 之间时，输出本身，否者输出上下限。 执行的结果： shenzhen—shen

10.4.7 FIND（查找字符串中的出现的一个字符）

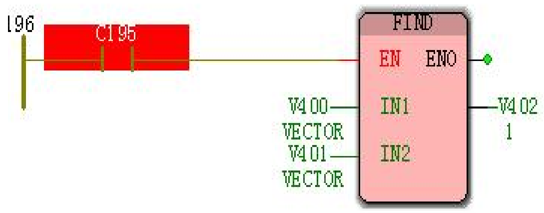
IL 编程语言	LD, FBD 编程语言
功能：FIND 指令用于确定在第一个字符串中出现的第二个字符串（以首字符为准）的位置。	
LD IN1 FIND IN2 ST OUT ST 编程语言 OUT:=FIND（IN1，IN2）	
注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1， IN2 和 OUT 或使用常量	

➢ **FIND 指令处理的数据类型**

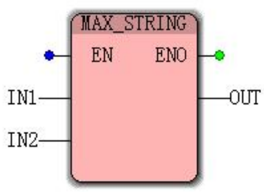
输入和输出	数据类型	描述
IN1	STRING	第一个输入字符或字符串，字符串中左起第一个字符的位置为 1

IN2	STRING	第二个输入字符或字符串，字符串中左起第一个字符的位置为 1
OUT	STRING	输出，第二个字符串中的第一个字符出现在第一个字符串中的位置，如果第一个字符串中不包含这个字符，输出为 O。

➢ **功能和动作例子**

查找相同的字符串	描述
	当触点为 ON 时，执行该指令，当地址 V400 中的字符串与地址 401A 的字符串相等时，地址 V402 输出 1； 执行的结果：VECTOR=VECTOR 输出 1；

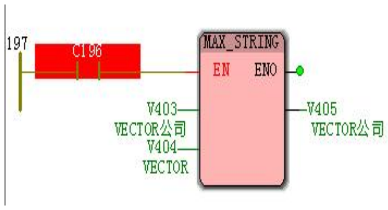
10.4.8 MAX_STRING（取较大的字符串）

IL 编程语言	LD， FBD 编程语言
功能：MAX_STRING 指令用于确定两个字符中的较大的一个。 用法	
LD IN1	
MAX_STRING IN2	
ST OUT	
ST 编程语言	
OUT:=MAX_ STRING（IN1，IN2）	
注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1， IN2 和 OUT 或使用常量	

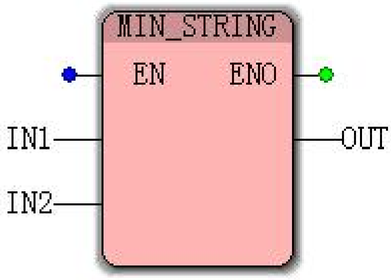
➢ **MAX_STRING 指令处理的数据类型**

输入和输出	数据类型	描述
IN1	STRING	第一个输入字符或字符串
IN2	STRING	第二个输入字符或字符串
OUT	STRING	输出，两个字符或字符串中较大的一个;字符的大小均按其 ASCII 码计算;当输入字符串时，仅第一个字符参与比较，即如果一个字符串的第一个字符较大，则本字符串全部输出

➢ **功能和动作例子**

<table>
<tr><td>比较最大的字符串</td><td>描述</td></tr>
<tr><td>
</td><td>当触点为 ON 时，执行该指令，比较地址 V403 的字符串与地址 V404 的字符串的长度，输出最大的字符串存在 V405 中：
执行的结果：VECTOR 公司----VECTOR
输出 ： VECTOR 公司</td></tr>
</table>

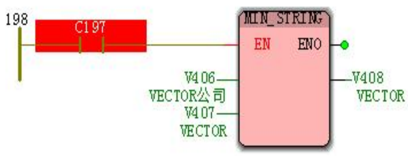
10.4.9 MIN_STRING（取较小的字符串）

<table>
<tr><td>IL 编程语言</td><td>LD, FBD 编程语言</td></tr>
<tr><td colspan="2">功能：MIN_STRING 指令用于确定两个字符中的较小的一个。</td></tr>
<tr><td>LD IN1</td><td rowspan="5">
</td></tr>
<tr><td>MIN_STRING IN2</td></tr>
<tr><td>ST OUT</td></tr>
<tr><td>ST 编程语言</td></tr>
<tr><td>OUT:=MIN_ STRING（IN1，IN2）</td></tr>
<tr><td colspan="2">注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1 , IN2 和 OUT 或使用常量</td></tr>
</table>

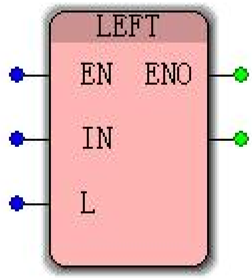
➢ **MIN_STRING 指令处理的数据类型**

输入和输出	数据类型	描述
IN1	STRING	第一个输入字符或字符串
IN2	STRING	第二个输入字符或字符串
OUT	STRING	输出，两个字符或字符串中较小的一个;字符的大小均按其 ASCII 码计算;当输入字符串时，仅第一个字符参与比较，即如果一个字符串的第一个字符较小，则本字符串全部输出

➢ **功能和动作例子**

<table>
<tr><td>比较最小字符串</td><td>描述</td></tr>
<tr><td>
</td><td>当触点为 ON 时，执行该指令，比较地址 V406 字符串与地址 V407 字符串的长度，输出最小字符串存在 V408 中
执行的结果：VECTOR 公司--VECTOR
输出 ： VECTOR</td></tr>
</table>

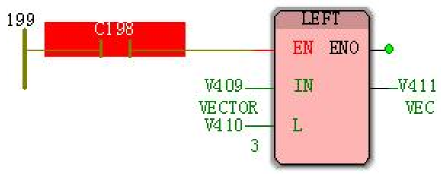
10.4.10 LEFT（取出字符串最左边的几个字符）

<table>
<tr><th>IL 编程语言</th><th>LD, FBD 编程语言</th></tr>
<tr><td colspan="2">功能：LEFT 指令用于取出输入字符串中最左边的几个连续的字符。</td></tr>
<tr><td>LD IN</td><td rowspan="5">
</td></tr>
<tr><td>LEFT L</td></tr>
<tr><td>ST OUT</td></tr>
<tr><td>ST 编程语言</td></tr>
<tr><td>OUT:=LEFT（IN, L）</td></tr>
<tr><td colspan="2">注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN, L 和 OUT 或使用常量</td></tr>
</table>

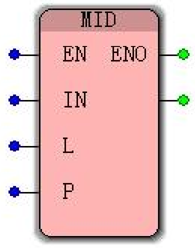
➢ **LEFT 指令处理的数据类型**

输入和输出	数据类型	描述
IN	STRING	输入字符串
L	ANY_INT	要取出的字符数
OUT	STRING	输出，取出输入字符串中从最左侧开始的 L 个连续的字符;L 必须大于 0，且小于等于输入字符串 IN 的字符个数

➢ **功能和动作例子**

<table>
<tr><th>将字符串”VECTOR”的前三个字符提取出来</th><th>描述</th></tr>
<tr><td>
</td><td>当触点为 ON 时，执行该指令，将地址 V409 的字符串前三个字符提取出来
执行的结果：VECTOR ----VEC</td></tr>
</table>

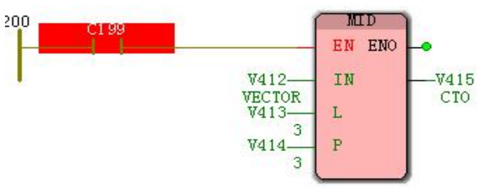
10.4.11 MID（取出字符串中的几个字符）

IL 编程语言	LD, FBD 编程语言
功能：MID 指令用于取出输入字符串中最左边的几个连续的字符。	
LD IN	
MID L,P	
ST OUT	
ST 编程语言	
OUT:=MID（IN, L, P）	
注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN, L,P 和 OUT 或使用常量	

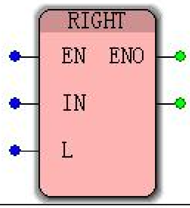
➢ **MID 指令处理的数据类型**

输入和输出	数据类型	描述
IN	STRING	输入字符串
L	ANY_INT	要取出的字符数
P	ANY_INT	要取出的字符的起始位置，即从输入字符串的第 P 个字符开始取字符
OUT	STRING	输出，取出输入字符串中从第 P 个至 P+L 个连续的字符; 注:L 必须大于 0;第 P+L 个字符必须在输入字符串中;P 必须大于 0 且小于等于输入字符串的最大字符数;IN 必须为非空字符串

➢ **功能和动作例子**

提取字符串中任意的字符	描述
	当触点 C199 为 ON 时，执行该指令，提取地址 V412 中字符串里的三个字符，其中 L 为需要提取的符号数，P 为提取字符的开始位置 执行的结果：VECTOR---CTO

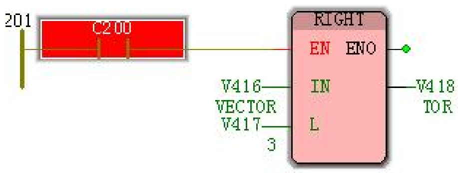
10.4.12 RIGHT（取出字符串最右边的几个字符）

IL 编程语言	LD, FBD 编程语言
功能：RIGHT 指令用于取出输入字符串中最右边的几个连续的字符。	
LD IN	
RIGHT L	
ST OUT	
ST 编程语言	
OUT:=RIGHT（IN, L）	
注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN, L 和 OUT 或使用常量	

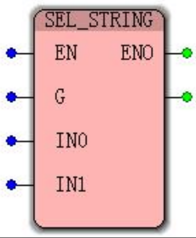
➢ **RIGHT 指令处理的数据类型**

输入和输出	数据类型	描述
IN	STRING	输入字符串
L	ANY_INT	要取出的字符数
OUT	STRING	输出，取出输入字符串中从最右侧开始的 L 个连续的字符;L 必须大于 0，且小于等于输入字符串 IN 的字符个数

➢**功能和动作例子**

提取字符串 VECTOR 最右边的 3 个字符	描述
	当触点 C200 为 ON 时，执行该指令，将地址 V416 的最右边的三个字符提取出来，输出 V418 执行的结果：VECTOR---TOR

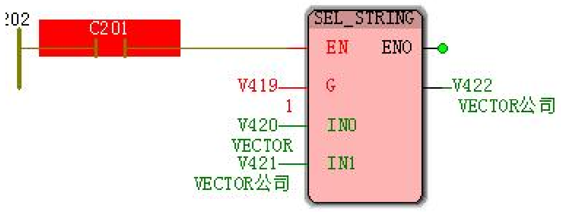
10.4.13 SEL_STRING（字符串的二进制选择）

IL 编程语言	LD, FBD 编程语言
功能：SEL_STRING 指令用于取出输入字符串中最左边的几个连续的字符	
LD IN	
SEL_STRING IN1，IN2	
ST OUT	
ST 编程语言	
OUT:=SEL_STRING（IN, IN1，IN2）	
注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN, IN1, IN2 和 OUT 或使用常量	

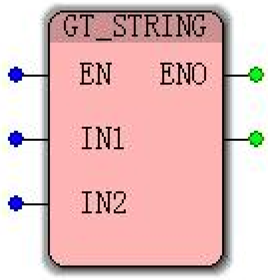
➢ **SEL_STRING 指令处理的数据类型**

输入和输出	数据类型	描述
IN（G）	BOOL	选择输入端
IN1（INO）	STRING	第一个输入
IN2（IN1）	STRING	第二个输入
OUT	STRING	输出 如果 IN=0,OUT=IN1;如果 IN=1, OUT=IN2

➢ **功能和动作例子**

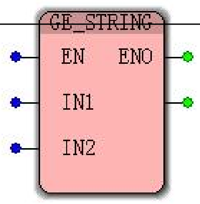
通过变量 V419 的值进行选择输出的字符串	描述
	当触点 C201 为 ON 时，执行该指令，当地址 V419 为 0 时，输出地址 V420 中的字符串；当地址 V419 为 1 时，输出地址 V421 中的字符串：

10.4.14 GT_STRIN（字符串大于）

IL 编程语言	LD, FBD 编程语言
功能：GT_STRING 指令用于确定第一个字符串是否大于第二个字符串。	
LD IN1	
GT_STRING IN2	
ST OUT	
ST 编程语言	
OUT:=GT_ STRING（IN1，IN2）	
注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1, IN2 和 OUT 或使用常量	

10.4.15 GE_STRING（字符串大于等于）

IL 编程语言	LD, FBD 编程语言
功能：GE_STRING 指令用于确定第一个字符串是否大于等于第二个字符串。	
LD IN1	
GE_STRING IN2	
ST OUT	
ST 编程语言	

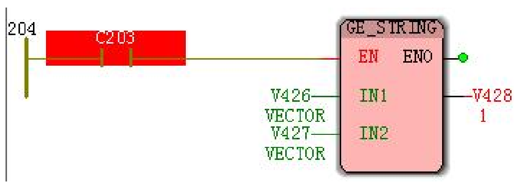


OUT:=GE_STRING（IN1，IN2）	
注:IL， ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1, IN2 和 OUT 或使用常量	

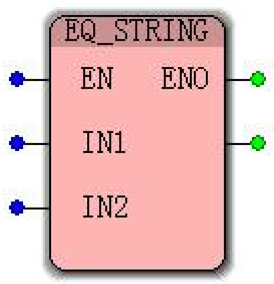
➢ **GE_STRING 指令处理的数据类型**

输入和输出	数据类型	描述
IN1	STRING	第一个输入
IN2	STRING	第二个输入
OUT	BOOL	输出，如果第一个输入字符串的第一个字符大于或等于第 二个输入字符串的第一个字符，则认为 IN1>=IN2,OUT=1;否则 OUT=0;字符的大小均按其 ASCII 码计算

➢ **功能和动作例子**

比较 V426 与 V427 两个字符串	描述
	当触点 C203 为 ON 时 ，执行该指令，当地址 V426 的字符串长度大于等于地址 V427 的字符串长度时；地址 V428 输出 1；否者输出 0 执行的结果； V426>=V426 输出 1

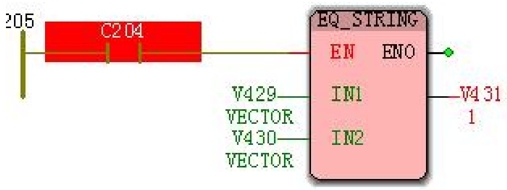
10.4.16 EQ_STRING（字符串等于）

IL 编程语言	LD， FBD 编程语言
功能：EQ_STRING 指令用于确定第一个字符串是否等于第二个字符串。	
LD IN1	
EQ_STRING IN2	
ST OUT	
ST 编程语言	
OUT:=EQ_STRING（IN1， IN2）	
注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1, IN2 和 OUT 或使用常量	

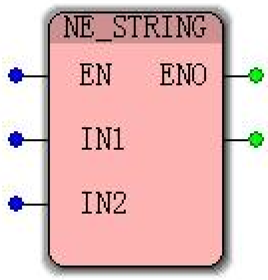
➢ **EQ_STRING 指令处理的数据类型**

输入和输出	数据类型	描述
IN1	STRING	第一个输入
IN2	STRING	第二个输入
OUT	BOOL	输出，如果第一个输入字符串与第二个输入字符串完全一样，则认为 IN1=IN2, OUT=1;否则 OUT=0

➢**功能和动作例子**

比较 V429 与 V430 两个字符串	描述
	当触点 C204 为 ON 时，执行该指令，当地址 V429 的字符串长度等于地址 V430 的字符串长度时；地址 V431 输出 1；否者输出 0 执行的结果； V429=V430 输出 1

10.4.17 NE_STRING（字符串不等于）

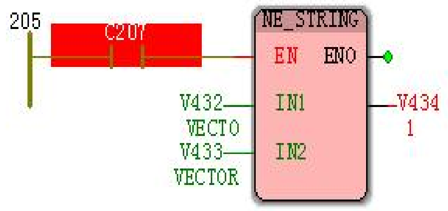
IL 编程语言	LD, FBD 编程语言
功能：NE_STRING 指令用于确定第一个字符串是否不等于第二个字符串。	
LD IN1 NE_STRING IN2 ST OUT ST 编程语言 OUT:=NE_STRING（IN1，IN2）	
注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1, IN2 和 OUT 或使用常量	

➢ **NE_STRING 指令处理的数据类型**

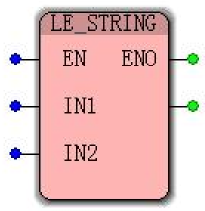
输入和输出	数据类型	描述
IN1	STRING	第一个输入
IN2	STRING	第二个输入
OUT	BOOL	输出，如果第一个输入字符串与第二个输入字符串不一样，则 OUT=1;否则 OUT=0;本指令与 EQ_STRING 相反

➢ **功能和动作例子**

比较 V432 与 V433 两个字符串	描述

	当触点 C207 为 ON 时，执行该指令，当地址 V432 的字符串长度不等于地址 V433 的字符串长度时；地址 V434 输出 1；否者输出 0 执行的结果；　V429< >V430 输出 1

10.4.18 LE_STRING（字符串小于等于）

IL 编程语言	LD，　FBD 编程语言
功能：LE_STRING 指令用于确定第一个字符串是否小于等于第二个字符串。	
LD IN1 LE_STRING IN2 ST OUT ST 编程语言 OUT:=LE_STRING（IN1，IN2）	
注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1, IN2 和 OUT 或使用常量	

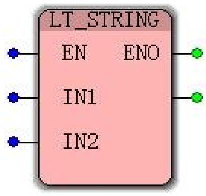
➤ **LE_STRING 指令处理的数据类型**

输入和输出	数据类型	描述
IN1	STRING	第一个输入
IN2	STRING	第二个输入
OUT	BOOL	输出，如果第一个输入字符串的第一个字符小于或等于第二个输入字符串的第一个字符，则认为 IN1<=IN2,OUT=1;否则 OUT=0;字符的大小均按其 ASCII 码计算

➤ **功能和动作例子**

比较 V435 与 V436 两个字符串	描述
	当触点 C206 为 ON 时，执行该指令，当地址 V435 的字符串长度小于等于地址 V436 的字符串长度时；地址 V437 输出 1；否者输出 0 执行的结果；　V435<=V436 输出 1

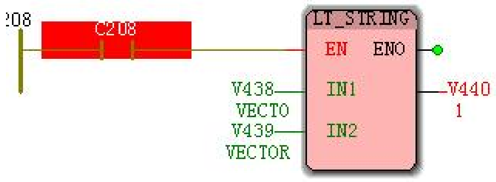
10.4.19 LT_STRING（字符串小于）

IL 编程语言	LD, FBD 编程语言
功能：LT_STRING 指令用于确定第一个字符串是否小于第二个字符串。	
LD IN1	
LT_STRING IN2	
ST OUT	
ST 编程语言	
OUT:=LT_STRING（IN1，IN2）	
注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1, IN2 和 OUT 或使用常量	

➢ **LT_STRING 指令处理的数据类型**

输入和输出	数据类型	描述
IN1	STRING	第一个输入
IN2	STRING	第二个输入
OUT	BOOL	输出，如果第一个输入字符串的第一个字符小于第二个输入字符串的第一个字符，则认为 IN1<IN2, OUT=1;否则 OUT=0;字符的大小均按其 ASCII 码计算

➢ **功能和动作例子**

比较 V438 与 V439 两个字符串	描述
	当触点 C206 为 ON 时，执行该指令，当地址 V438 的字符串长度小于地址 V439 的字符串长度时；地址 V440 输出 1；否者输出 0 执行的结果； V435<V436 输出 1

10.4.20 STRING_TO_*（字符串转换为其它类型）

字符串 FU	功能
STRING_TO_*指令	STRING_TO_*指令用于将一个 STRING 型数据转换一个有效的 ANY 数值类型，有 12 个指令，分别是将 STRING 型数据转换为 BYTE, WORD, DWORD, SINT, INT, DINT, USINT,UINT, UDINT, REAL, LREAL 和 TIME 等类型。

➢ **STRING_TO_*转换指令**

指令	输入值	输出值	描述

STRING_TO_BYTE	STRING	BYTE	输入格式 0~255*，输出为 BYTE 类型
STRING_TO_WORD	STRING	WORD	输入格式 0~65535*”，输出为 WORD 类型
STRING_TO_DWORD	STRING	DWORD	输入格式 0~4,294,967,295*，输出为 DWORD 类型
STRING_TO_SINT	STRING	SINT	输入格式-128~127*，输出为 SINT 类型
STRING_TO_INT	STRING	INT	输入格式-32768~-32767*，输出为 INT 类型
STRING_TO_DINT	STRING	DINT	输入格式-2,147,483,648~2,147,483,647”，输出 DINT
STRING_TO_USINT	STRING	USINT	输入格式 0~255”，输出为 USINT 类型
STRING_TO_UINT	STRING	UINT	输入格式 0~65535”，输出为 UINT 类型
STRING_TO_UDINT	STRING	UDINT	输入格式 0~4,294,967,295”，输出为 UDINT 类型
STRING_TO_REAL	STRING	REAL	输入格式小数*，输出为 REAL 类型
STRING_TO_LREAL	STRING	LREAL	输入格式小数*，输出为 LREAL 类型
STRING_TO_TIME	STRING	TIME	输入格式 T#0~T#4,294,967,295”毫秒，输出秒
*:除 STRING_TO_TIME 指令外，可带对应的 BYTE#, WORD#等类型前缀和”16#”前缀，也可不带前缀;十进制时输入字符有效值为正负号和数字 0~9;数字后出现的其它字符以及再出现的数字均无效。			

➢**用法（以 STRING_TO_BYTE 为例）**

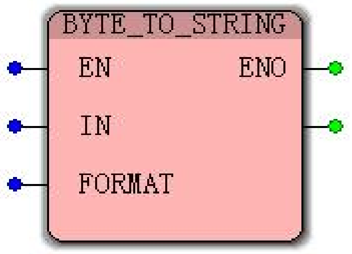
输入和输出	数据类型	描述
IN	STRING	输入字符或字符串，输入格式 0~255
OUT	BYTE	输出，把输入转换为 BYTE 类型

在数值转换过程中，可能会发生如下错误：

☞ 输入带有非法前缀，如 INT##123,　INT#是非法的前缀；

☞ 输入值超出输出数据类型的范围，如 STRING_TO_BYTE 指令的输入值 1024, 1024 大于 BYTE 类型的范围 0~255 。

10.4.21 *_TO_STRING（其它类型转换为字符串）

<table>
<tr><th>IL 编程语言</th><th>LD， FBD 编程语言</th></tr>
<tr><td colspan="2">功能：*_TO_STRING 分为 12 个指令，可分别将 BYTE, WORD, DWORD, SINT, INT, DINT,USINT, UINT, UDINT, REAL, LREAL 和 TIME 等类型转换为 STRING 类型。
用法（以 BYTE_TO_STRING 为例）</td></tr>
<tr><td>LD IN</td><td rowspan="5">
</td></tr>
<tr><td>BYTE_TO_STRING FORMAT</td></tr>
<tr><td>ST OUT</td></tr>
<tr><td>ST 编程语言</td></tr>
<tr><td>OUT:=BYTE_TO_ STRING （IN, IN1）</td></tr>
</table>

➢ **BYTE_TO_STRING 指令处理数据类型**

输入和输出	数据类型	描述
IN	BYTE	输入范围 0~255
FORMAT	STRING	有效的格式:%c, %x, %u,默认%x
OUT	STRING	输出，格式为%c 时，输出为输入的 ASCII 码;格式为%x 时，输出为输入的十六进制数;格式为%u 时，输出为输入的无符号十进制数

➢ **WORD_TO_STRING 指令处理数据类型**

输入和输出	数据类型	描述
IN	WORD	输入范围 0~65535
IN1（FORMAT）	字符串	有效的格式:%x, %u,默认%x
OUT	STRING	输出，格式为%X 时，输出为输入的十六进制数;格式为%U 时，输出为输入的无符号十进制数

➢ **DWORD_TO_STRING 指令处理的数据类型**

输入和输出	数据类型	描述
IN	DWORD	输入范围 0~4,294,967,295
IN1（FORMAT）	字符串	有效的格式:%x, %u,默认%x
OUT	STRING	输出，格式为%X 时，输出为输入的十六进制数;格式为%U 时，输出为输入的无符号十进制数

➢ **SINT_TO_STRING 指令处理的数据类型**

输入和输出	数据类型	描述
IN	SINT	输入范围-128~127
IN1（FORMAT）	字符串	有效的格式:%d,默认%d
OUT	STRING	输出，输出为输入的有符号十进制数

➢ **INT_TO_STRING 指令处理的数据类型**

输入和输出	数据类型	描述
IN	INT	输入范围一 32768~32767
IN1（FORMAT）	字符串	有效的格式:%d,默认%d
OUT	STRING	输出，输出为输入的有符号十进制数

➢ **DINT_TO_STRING 指令处理的数据类型**

输入和输出	数据类型	描述
IN	DINT	输入范围-2,147,483,648~2,147,483,647
IN1（FORMAT）	字符串	有效的格式:%d,默认%d
OUT	STRING	输出，输出为输入的有符号十进制数

➢ **USINT_TO_STRING 指令处理的数据类型**

输入和输出	数据类型	描述
IN	USINT	输入范围 0~255
FORMAT	字符串	有效的格式:%u,默认%u
OUT	STRING	输出，输出为输入的符号十进制数

➢ **UINT_TO_STRING 指令处理的数据类型**

输入和输出	数据类型	描述
IN	UINT	输入范围 0~65535
IN1（FORMAT）	字符串	有效的格式:%U,默认%U
OUT	STRING	输出，输出为输入的无符号十进制数

➢ **DINT_TO_STRING 指令处理的数据类型**

输入和输出	数据类型	描述
IN	UDINT	输入范围 0~4,294,967,295
IN1（FORMAT）	字符串	有效的格式:%u,默认%u
OUT	STRING	输出，输出为输入的无符号十进制数

➢ **REAL_TO_STRING 指令处理的数据类型**

输入和输出	数据类型	描述
IN	REAL	输入范围-3.402823466 E+38... -1.175494351 E-38 以及 +1.175494351 E-38... +3.402823466 E+38
IN1（FORMAT）	字符串	有效的格式:%e, %f,默认%e
OUT	STRING	输出，格式为%e 时，输出为输入的科学计数法表示的浮点数;格式为%f 时，输出为输入的浮点数

LREAL_T0_STRING 指令处理的数据类型与 REAL_TO_STRING 相同，只是输入范围不同。

➢ **TIME_TO_STRING 指令处理的数据类型**

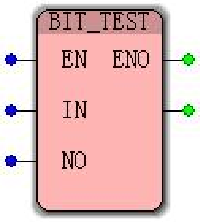
输入和输出	数据类型	描述
IN	TIME	输入 TIME 类型，范围 T#O~T#4,294,967,295”毫秒，如 T#1 MS
IN1（FORMAT）	字符串	有效的格式:%u,默认%u
OUT	STRING	输出，格式为%u 时，输出是将输入表示为不带前缀的 TIME 类型，单位毫秒，如输入 T#1 S，输出 1000;如不指定格式，输出是将输入表示为带前缀的 TIME 类型，如输入 T#1 S，输出为 T#1000MS

10.5 位操作函数 BIT_UTIL

位操作函数功能是带有多个输入参数和一个输出参数的程序组织单元 POU，能对输入的位串做读写操作，位操作函数的缩写为 BIT_UTIL。 位操作函数需要单独插入固件库。位操作函数包含的指令（在编辑向导中，从下拉列表选择”BIT_UTIL”）

BIT_UTIL 功能块			
BIT_TEST	GET_CHAR	GET_LSB	GET_MSB
I_BIT_IN_BYTE	I_BIT_IN_WORD	I_BIT_IN_DWORD	PARITY_BYTE
PARITY_WORD	PARITY_DWORD	R_BIT_IN_BYTE	R_BIT_IN_WORD
R_BIT_IN_DWORD	S_BIT_IN_BYTE	S_BIT_IN_WORD	S_BIT_IN_DWORD
SET_LSB	SET_MSB	STRING_TO_BUFFER	SWAP

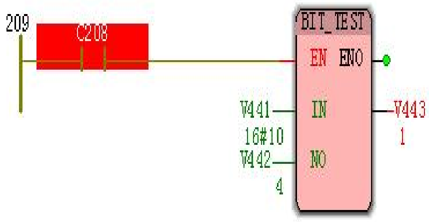
10.5.1 BIT_TEST（读取位串中的位值指令）

IL 编程语言	LD， FBD 编程语言
功能：BIT_TEST 指令用于读取输入位串中的单个位的值。	
LD IN	
BIT_TEST IN1	
ST OUT	
ST 编程语言	
OUT:=BIT_T0_EST（ IN, IN1）	
注:IL， ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN, IN1 和 OUT 或使用常量	

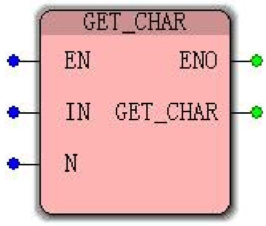
➢ **BIT_TEST 指令处理的数据类型**

输入和输出	数据类型	描述
IN（IN）	ANY_BIT	输入位串
IN1（NO）	SINT	要读取的位串中的第 NO 个位，取值范围:BYTE 0~7， WORD0~15， DWORD0~31
OUT	BOOL	输出，输入位串中第 NO 个位的值

➢ 功能和动作例子

读取地址 V441 字节中的第 4 位的值；	描述
	当触点 C208 为 ON 时 ，执行该指令，读取地址 V441 中的第 4 位的值，V442 地址输出其相应的位状态： 注意（第几位是从 0 开始计算的）

10.5.2 GET_ CHAR（取出字符串中的字符指令）

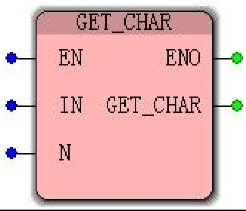
IL 编程语言	LD， FBD 编程语言
功能：GET_CHAR 指令用于从输入字符串中取得一个字符，并 ASCII 码表示该字符。	
LD IN	
GET_CHAR IN1	
ST OUT	
ST 编程语言	
OUT:=GET_CHAR（ IN, IN1）	
注:IL， ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN, IN1 和 OUT 或使用常量	

➢ **GET_CHAR 指令处理的数据类型**

输入和输出	数据类型	描述
IN（IN）	ANY_BIT	输入字符串
IN1（N）	INT	要去出的字符串中的第 N 个字符，取值范围 0~32767
OUT（GET_CHAR）	INT	输出，输入字符串中第 N 个字符的 ASCII 码值

注:目前的软件版本还不支持这个指令。

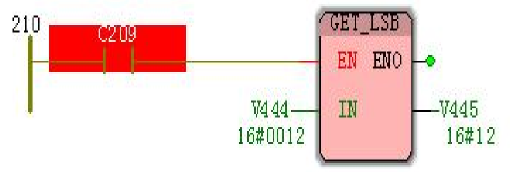
10.5.3 GET_LSB（取出位串中的低 8 位指令）

IL 编程语言	LD， FBD 编程语言
功能：GET_LSB 指令用于读取输入位串中的较低字节（the Less SignificantBYTE）的值。	
LD IN	
GET_LSB	
ST OUT	
ST 编程语言	
OUT:=GET_LSB（ IN）	
注:IL， ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN、和 OUT 或使用常量	

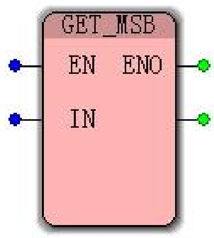
➢ **GET_LSB 指令处理的数据类型**

输入和输出	数据类型	描述
IN	WORD	输入位串
OUT	BYTE	输出，较低字节（低 8 位）的值

➢ **功能和动作例子**

读取地址 V444 低 8 位的值	描述
	当触点 C209 为 ON 时 ，执行该指令，将地址 V444 的低 8 位的值状态取出，存放在地址 V445 中； 执行的结果：16#0012（word）--16#12（byte）

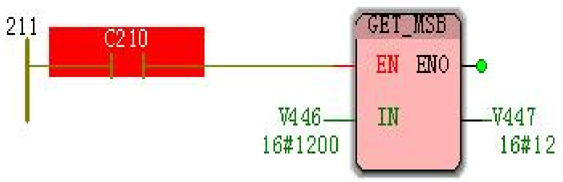
10.5.4 GET_MSB（取出位串中的高 8 位指令）

IL 编程语言	LD, FBD 编程语言
功能：GET_MSB 指令用于读取输入位串中的最高字节（the Most Significant BYTE）的值。	
LD IN GET_MSB ST OUT ST 编程语言 OUT:=GET_MSB（ IN）;	
注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN、和 OUT 或使用常量	

➢ **GET_MSB 指令处理的数据类型**

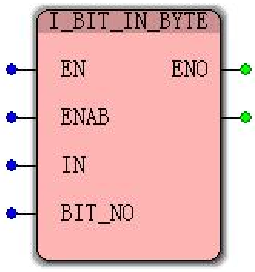
输入和输出	数据类型	描述
IN	WORD	输入位串
OUT	BYTE	输出，最高字节（高 8 位）的值

➢ **功能和动作例子**

读取地址 V446 高 8 位的值	描述
	当触点 C210 为 ON 时 ，执行该指令，将地址 V446 的高 8 位的值状态取出，存放在地址 V447 中； 执行的结果：16#1200（word）--16#12（byte）

10.5.5 I_BIT_IN*（将位串中单个位取反指令）

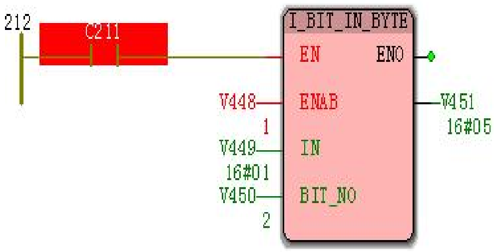
IL 编程语言	LD, FBD 编程语言

功能：I_BIT_IN*指令用于将输入位串中的单个位取反，包括 I_BIT_IN_WORD 和 I_BIT_IN_DWORD 指令，能够处理 BYTE, WORD 和 DWORD 类型的输入位串。	
LD ENAB	
I_ BIT_IN_BYTE IN， BIT_NO	
ST OUT	
ST 编程语言	
OUT:=I_ BIT_IN_ BYTE（IN）	
注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1, IN, IN2 和 OUT 或使用常量	

➢ **I_BIT_IN_BYTE 指令处理的数据类型**

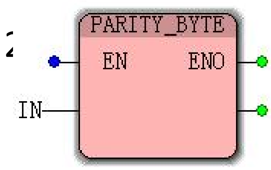
输入和输出	数据类型	描述
IN1（ENAB）	BOOL	使能
IN	BYTE WORD DWORD	输入位串
IN2（BIT_N O）	SINT	要操作的位串中的第 NO 个位，取值范围:BYTE 时 0~7,WORD 时 0~15, DWORD 时 0~31（其它值无效）
OUT	BYTE	输出，当 IN1 为 FALSE 时，输出 OUT 等于输入 IN;当 IN1 为 TRUE 时，输出 OUT 是将输入 IN 的第 NO 个位取反后的值

➢ **功能和动作例子**

取反地址 V449 中的第 2 位的值	描述
	当触点 C211 为 ON 时 ，执行该指令，且当 ENAB 为 ON 时，取反 V449 中的第 2 位，进行输出；当 ENAB 为 FALSE 时，不进行取反输出自已本身； 执行的结果：16#01（byte）---16#05（byte）

10.5.6 PARITY_*（位串的奇偶校验指令）

IL 编程语言	LD, FBD 编程语言
功能：PARITY_*指令用于检查输入位串中为 1 的位的数量是奇数还是偶数个时，如果是奇数则输出为 FALSE，如果为偶数则输出为 TRUE, PARITY_*指令包括队 RITY_BYTE, PARITY_WORD 和 PARITY _DWORD 指令，能够处理 BYTE, WORD 和 DWORD 类型的输入位串。	
LD IN	

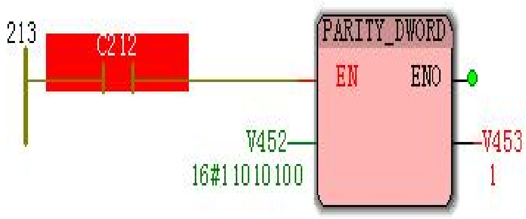


PARITY_BYTE	
ST OUT	
ST 编程语言	
OUT:=PARITY_BYTE（IN）	
注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

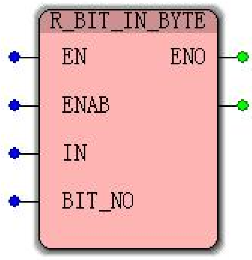
➢ PARITY_BYTE 指令处理的数据类型

输入和输出	数据类型	描述
IN	BYTE WORD DWORD	输入位串
OUT	BOOL	当输入位串中为 1 的位的数量是奇数个时，输出为 FALSE;为 1 的位的数量是偶数个时（包括 0 个为 1 的位），输出为 TRUE

➢ 功能和动作例子

校验地址 V452 中 1 的个数进行输出	描述
	当触点 C212 为 ON 时，执行该指令，将地址 V452 中为 1 的个数取出，当为 1 的个数为奇数时，地址 V453 输出为 0；当为 1 的个数为偶数时（或全 0 时）地址 V453 输出 1 执行的结果：16#11010100（DWORD）----1（BOOL）

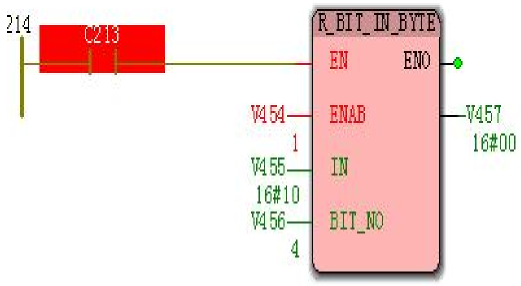
10.5.7 R _BIT_IN_*（将位串中单个位置 0 指令）

<table>
<tr><th>IL 编程语言</th><th>LD， FBD 编程语言</th></tr>
<tr><td colspan="2">功能： R _BIT_IN_* 指令用于将输入位串中的单个位置 0， 包括 R_BIT_IN_BYTE 和 R_BIT_IN_DWORD 指令，能够处理 BYTE, WORD 和 DWORD 类型的输入位串。</td></tr>
<tr><td>LD ENAB</td><td rowspan="5">
</td></tr>
<tr><td>R_BIT_IN_ BYTE IN，BIT_NO</td></tr>
<tr><td>ST OUT</td></tr>
<tr><td>ST 编程语言</td></tr>
<tr><td>OUT:=R_BIT_IN_ BYTE（IN）</td></tr>
<tr><td colspan="2">注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1, IN, IN2 和 OUT 或使用常量</td></tr>
</table>

➢ **R_BIT_IN_BYTE 指令处理的数据类型**

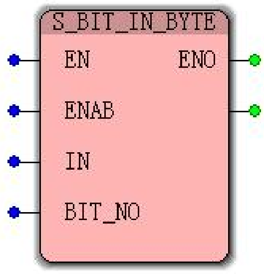
输入和输出	数据类型	描述
IN1（ENAB）	BOOL	使能
IN	BYTE WORD DWORD	输入位串
IN2 （BIT_NO）	SINT	要操作的位串中的第 NO 个位， 取值范围:BYTE 时 0~7,WORD 时 0~15 DWORD 时 0~31（其它值无效）
OUT	BYTE	输出，当 IN1 为 FALSE 时，输出 OUT 等于输入 IN;当 IN1 为 TRUE 时， 输出 OUT 是将输入 IN 的第 NO 个位置 0 后的值

➢**功能和动作例子**

<table>
<tr><th>复位地址 V455 中第 4 位的状态</th><th>描述</th></tr>
<tr><td>
</td><td>当触点 C213 为 ON 时 ，执行该指令， 当 ENAB 为 ON 时， 复位地址 V455 中的第 4 的状态；当 ENAB 为 FALSE 时 ，输出地址 V455 当前值；

执行的结果：16#10（BYTE）---16#00（BYTE）</td></tr>
</table>

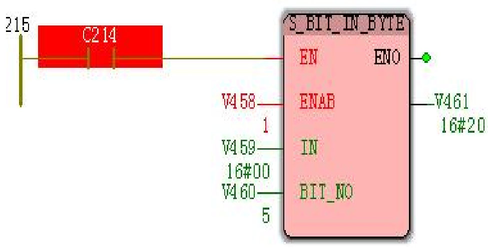
10.5.8 S_BIT_IN_*（将位串中单个位置 1 指令）

<table>
<tr><th>IL 编程语言</th><th>LD, FBD 编程语言</th></tr>
<tr><td colspan="2">功能：SBIT_IN_*指令用于将输入位串中的单个位置 1，包括 S_BIT_IN_BYTE, S_BIT_IN_WORD 和 S_BIT _IN_DWORD 指令，能够处理 BYTE, WORD 和 DWORD 类型的输入位串。</td></tr>
<tr><td>LD ENAB</td><td rowspan="5">
</td></tr>
<tr><td>S_BIT_IN_BYTE IN, BIT_NO</td></tr>
<tr><td>ST OUT</td></tr>
<tr><td>ST 编程语言</td></tr>
<tr><td>OUT:=S_BIT_IN_BYTE（IN）</td></tr>
<tr><td colspan="2">注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1, IN, IN2 和 OUT 或使用常量</td></tr>
</table>

➢ **S_BIT_IN_BYTE 指令处理的数据类型**

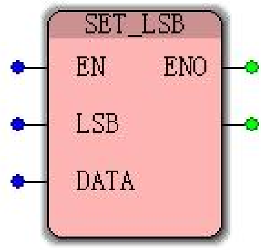
输入和输出	数据类型	描述
IN1（ENAB）	BOOL	使能
IN	BYTE WORD DWORD	输入位串
IN2（BIT_NO）	SINT	要操作的位串中的第 NO 个位，取值范围:BYTE 时 0~7,WORD 时 0~15,DWORD 时 0~31（其它值无效）
OUT	BYTE	输出，当 IN1 为 FALSE 时，输出 OUT 等于输入 IN;当 IN1 为 TRUE 时，输出 OUT 是将输入 IN 的第 NO 个位置 1 后的值

➢**功能和动作例子**

<table>
<tr><th>置位地址 V458 中第 5 位的状态</th><th>描述</th></tr>
<tr><td>
</td><td>当触点 C214 为 ON 时 ，执行该指令，当 ENAB 为 ON 时， 置位地址 V458 中的第 5 的状态；当 ENAB 为 FALSE 时 ，输出地址 V455 当前值；

执行的结果：16#00（BYTE）---16#20（BYTE）</td></tr>
</table>

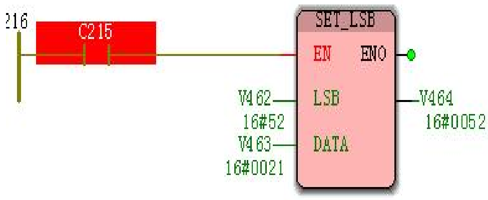
10.5.9 SET_LSB（向位串中的低 8 位写数指令）

IL 编程语言	LD,　FBD 编程语言
功能：SET_LSB 指令用于向输入位串中的较低字节（the Less Significant BYTE）写数值。	
LD IN1	
SET_LSB IN	
ST OUT	
ST 编程语言	
OUT:=SET_LSB（ IN1，IN）	
注:IL,　ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1, IN 和 OUT 或使用常量	

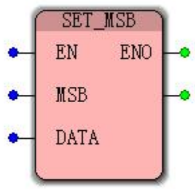
➢ **SET_LSB 指令处理的数据类型**

输入和输出	数据类型	描述
IN1（LSB）	BYTE	要写入的数值，范围 0~255
IN（DATA）	WORD	输入位串，0~65535
OUT	WORD	输出，将输入位串的较低字节（低 8 位）改为 IN1

➢ **功能和动作例子**

将地址 V463 的低 8 位的值改为 16#52	描述
	当触点 C215 为 ON 时，执行该指令，将地址 V463 的低 8 位的值改为 16#52，再输出： 执行的结果：16#0021-----16#0052

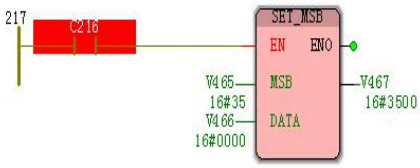
10.5.10 SET_MSB（向位串中的高 8 位写数指令）

IL 编程语言	LD,　FBD 编程语言
功能：SET_MSB 指令用于向输入位串中的最高字节（the Most Significant BYTE）写数值。	
LD IN1	
SET_MSB IN	
ST OUT	
ST 编程语言	
OUT:=SET_MSB（ IN1，IN）	
注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1, IN 和 OUT 或使用常量	

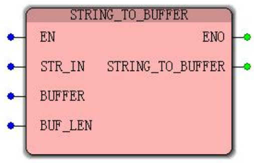
➢ **SET_MSB 指令处理的数据类型**

输入和输出	数据类型	描述
IN1（MSB）	BYTE	要写入的数值，范围 0~255
IN（DATA）	WORD	输入位串，0~65535
OUT	WORD	输出，将输入位串的最高字节（高 8 位）改为 IN1

➢ **功能和动作例子**

将地址 V466 的高 8 位的值改为 16#35	描述
	当触点 C216 为 ON 时，执行该指令，将地址 V466 的高 8 位的值改为 16#35，再输出： 执行的结果：16#0000-----16#3500

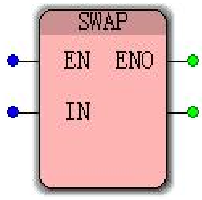
10.5.11 STRING_TO_BUFFER（复制字符串到缓冲区指令）

IL 编程语言	LD, FBD 编程语言
功能：STRING_TO_BUFFER 指令用于将字符串复制到缓冲区，缓冲区为一个字节数组。	
LD IN	
STRING_TO_BUFFER IN1，IN2	
ST OUT	
ST 编程语言	
OUT:=STRING_TO_ BUFFER（IN1,IN）	
注:IL,ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN, IN1, IN2 和 OUT 或使用常量	

➢ **STRING_TO_BUFFER 指令处理的数据类型**

输入和输出	数据类型	描述
IN（STR_IN）	STRING	输入字符串
IN1 [0]（BUFFER）	BYTE	缓冲区（字节数组）的一个元素，如 IN1 [0]或 IN1[1]
IN2（BUF_LEN）	INT	拷贝到缓冲区中的字符数
OUT（STRING_TO_BUFFER）	INT	输出，暂没有定义，实际上字符是被拷贝到缓冲区 IN1 中了，IN1 中的一个元素（字节）存放着被拷贝字符的 ASCII 码。

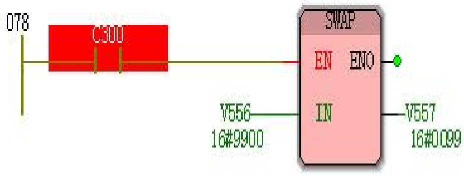
10.5.12 SWAP（交换高字节和低字节指令）

IL 编程语言	LD, FBD 编程语言
功能：SWAP 指令用于交换输入位串的最高字节（MSB）和较低字节（LSB）的位置。	
LD IN	
SWAP	
ST OUT	
ST 编程语言	
OUT:=SWAP（IN）	
注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

➢ **SWAP 指令处理的数据类型**

输入和输出	数据类型	描述
IN（IN）	WORD	输入位串
OUT（STRING_TO_BUFFER）	WORD	输出位串，将输入位串的高低字节交换后输出

➢ **功能和动作例子**

将地址 V556（16#9900）的高 8 位与低 8 位互换	描述
	当触点 C300 为 ON 时，执行该指令，将地址 V556 的高 8 位与低 8 位交换数值；执行的结果：16#9900 互换成 16#0099

10.6 ProConOS 功能

ProConOS 中的 FILE_OPEN,　FILE_CLOSE、FILE_SEEK、FILE_TELL、FILE_READ、FILE_WRITE, FILE_REMOVE 指令，不可用。

10.6.1 BUF 型转换为其它类型

BUF 型数据可分为 12 个功能块，可将基本数据类型从字节流复制到变量、数组或者用户自定义结构的元素中，即分别转换为 BYTE, WORD, DWORD, SINT, INT, DINT, USINT,UINT、UDINT、REAL, STRING, TIME 等数据，它主要用于数据传送或者在不同硬件平台上执行应用中的通讯。

ProConOS 中的 BUF 型数据转换指令需要在编辑向导中，从下拉列表选择”ProConOS”。源数据必须为字节数组（数据类型为 BYTE），也可是 ANY_BIT （BOOL 除外）或 ANY_INT 型（被转换的字节数不能超出存放目标的数据类型）。

➢ **BUF 型数据的转换指令**

<table>
<tr><th>指令</th><th>源数据</th><th>转换后数据</th><th colspan="2">描述</th></tr>
<tr><td>BUF _TO_BYTE</td><td>BUF</td><td>BYTE</td><td rowspan="9">缓冲区的数据格式 MOTOROLA 或 INTEL，根据需要，源数据也可以是 ANY_BIT（BOOL 除外）或 ANY_INT</td><td>被转换的字节数是 1</td></tr>
<tr><td>BUF_TO_WORD</td><td>BUF</td><td>WORD</td><td>被转换的字节数是 2</td></tr>
<tr><td>BUF_TO_DWORD</td><td>BUF</td><td>DWORD</td><td>被转换的字节数是 4</td></tr>
<tr><td>BUF _TO_SINT</td><td>BUF</td><td>SINT</td><td>被转换的字节数是 1</td></tr>
<tr><td>BUF _TO_INT</td><td>BUF</td><td>INT</td><td>被转换的字节数是 2</td></tr>
<tr><td>BUF_ TO_DINT</td><td>BUF</td><td>DINT</td><td>被转换的字节数是 4</td></tr>
<tr><td>BUF _TO_USINT</td><td>BUF</td><td>USINT</td><td>被转换的字节数是 1</td></tr>
<tr><td>BUF _TO_UINT</td><td>BUF</td><td>UINT</td><td>被转换的字节数是 2</td></tr>
<tr><td>BUF _TO_UDINT</td><td>BUF</td><td>UDINT</td><td>被转换的字节数是 4</td></tr>
<tr><td>BUF _TO_RAEL</td><td>BUF</td><td>REAL</td><td>缓冲区的数据格式为 MOTOROLA</td><td>输入为 IEEE 754 浮点数的十六进制码，被转换的字节数可以是 4</td></tr>
<tr><td>BUF_TO_STRING</td><td>BUF</td><td>STRING</td><td>缓冲区的数据格式为 MOTOROLA 或 INTEL</td><td>输入为字符的 ASCI I 码，被转换的字节数可以是 1, 2...等正整数</td></tr>
<tr><td>BUF _TO _TIME</td><td>BUF</td><td>TIME</td><td>缓冲区的数据格式为 INTEL</td><td>输入为以毫秒为单位的十六进制码，被转换的字节数是 4，输出单位为秒</td></tr>
</table>

➢ **BUF_TO_*指令处理的数据类型**

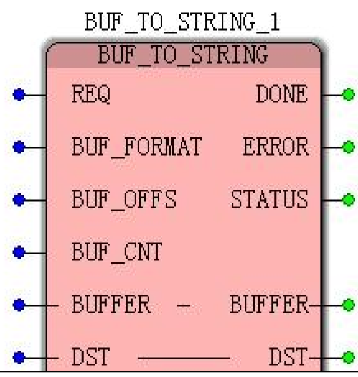
引脚	功能	数据类型	描述
REQ	输入	BOOL	上升沿有效
BUF_FORMAT	输入	BOOL	TRUE 表示缓冲区数据为 MOTOROLA 格式;FALSE

			表示 INTEL 格式
BUF_OFFS	输入	DINT	缓冲区中被转换的起始字节序号，0 表示缓冲区第一个字节
BUF_CNT	输入	DINT	缓冲区中被转换的字节数
BUFFER	输入-输出	ARRAY	缓冲区，为一个字节数组
DST	输入-输出	数组或 ANT	存放区，被转换的字节内容放在这里，DST 的类型应与具体的 BUF_TO_* 类型一致，如 BUF_TO_BYTE 指令的 DST 必须是 BYTE 类型
DONE	输出	BOOL	转换完成后，置 1，直到 REQ 为 0
ERROR	输出	BOOL	转换正常，为 0，否则置 1
STATUS	输出	INT	如果转换不正常，则给出错误代码，具体见下表

注:MOTOROLA 和 INTEL 微处理器的数据存放顺序不同，INTEL 格式为高低字节排列，MOTOROLA 格式为低高字节排列。

错误代码	描述
0	转换过程正常完成
1	BUFFER 和 DST 的输出一输出类型错误
2	超出缓冲区的长度，即将被复制的字节数 BUF_CNT 大于缓冲区 BUFFER 的可用字节数
3	超出存放区的长度，即将被复制的字节数 BUF CNT 超出存放区长度
4	不支持这个数据类型
5	将要转换的字节长度与存放区的字节长度不对应，前者字节数必须能被后者的字节数整除
6	转换 INTEL/MOTOROLA 失败
7	字符串长度不适当，对于数据类型串，有必要做额外的检查
8	存放区数据类型错误
9	BUF_OFFS 数值不正确
10	BUF_CNT 数值不正确
11	缓冲区与存放区地址相同

注:缓冲区 BUFFER 的可用字节数—在缓冲区中从第 BUF_OFFS 个字节开始至最后一个字

IL 编程语言	LD, FBD 编程语言
见帮助文件	
ST 编程语言	
见帮助文件	

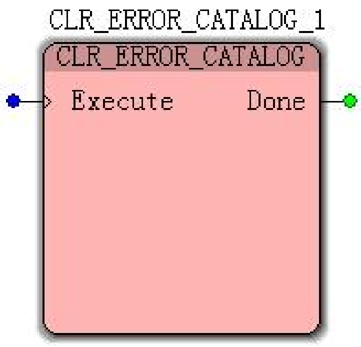
注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量

10.6.2 其它类型转换为 BUF 型

其它类型可转换为 BUF 型数据，可将变量、数组或者用户自定义结构的元素中的基本数据类型复制到字节流中，共有 12 个指令，分别将 BYTE,WORD, DWORD, SINT、INT、DINT, USINT, UINT, UDINT, REAL, STRING, TIME 等数据转换到 BUF 类型数据中，它主要用于数据传送或者在不同硬件平台上执行应用中的通讯。

ProConOS 中的其它类型转换为 BUF 型指令需要在编辑向导中，从下拉列表选择” ProConOS”。存放区必须为字节数组（数据类型为 BYTE），也可是 ANY_BIT （BOOL 除外）或 ANY_INT 型。这些指令的用法与上述，”BUF 型转换为其它类型”相似，这里不再描述。

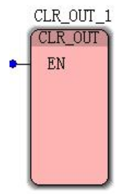
10.6.3 CLR_ERROR_CATALOG（册除完整的错误目录）

<table>
<tr><th>IL 编程语言</th><th>LD, FBD 编程语言</th></tr>
<tr><td colspan="2">功能：CLR_ERROR_CATALOG 指令用于删除完整的错误目录。</td></tr>
<tr><td>LD IN</td><td rowspan="5">
</td></tr>
<tr><td>CLR_ERROR_CATALOG</td></tr>
<tr><td>ST OUT</td></tr>
<tr><td>ST 编程语言</td></tr>
<tr><td>OUT:=CLR_ERROR_CATALOG（IN）</td></tr>
<tr><td colspan="2">注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量</td></tr>
</table>

➢ **CLR_ERROR_CATALOG 指令处理的数据类型**

输入和输出	数据类型	描述
IN（Execute）	BOOL	上升沿有效
OUT（Done）	BOOL	0:不能够删除错误目录。 1:成功删除错误目录。

10.6.4 CLR_OUT（将 I/O 映像的输出设置为 0 指针）

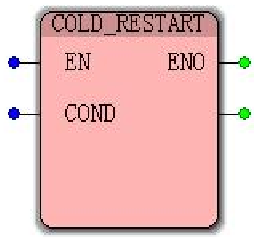
IL 编程语言	LD, FBD 编程语言
功能：CLR_OUT 指令用于将 I/O 映像区的输出设置为 0	
LD IN	
ST INCLR_OUT_1.EN	
CAL CLR_OUT_1	
ST 编程语言	
CLR_OUT_1（EN:=（IN））	
注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 或使用常量	

➢ **CLR_OUT 指令处理的数据类型**

输入和输出	数据类型	描述
IN（Execute）	BOOL	如果为 TRUE，则将 I/O 映像的所有输出都设置为零

注:CLR_OUT 指令暂时不能用。

10.6.5 COLD_RESTART（PLC 冷启动）

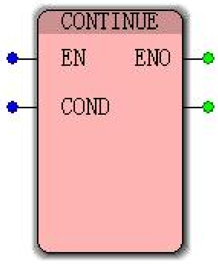
IL 编程语言	LD, FBD 编程语言
功能：COLD_RESTART 指令用于冷启动 PLC。在冷再启动过程中，初始化所有数据。如程序发生堆栈溢出、字符串错误或者被 0 除等问题时，可以调用这个指令自动重新启动程序的执行。	
LD IN	
COLD_RESTART	
ST OUT	
ST 编程语言	
COLD_RESTART_1（IN）	
注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

➢ **COLD_RESTART 指令处理的数据类型**

输入和输出	数据类型	描述
IN（COND）	BOOL	如果为 TRUE，则执行冷再启动
OUT	BOOL	如果 COND=TRUE，且可以冷启动，则 OUT 为 TRUE

注:COLD_RESTART 指令暂时不能用。

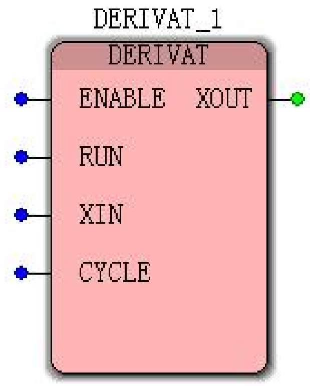
10.6.6 CONTINUE（继续运行程序）

<table>
<tr><th>IL 编程语言</th><th>LD，FBD 编程语言</th></tr>
<tr><td colspan="2">功能：CONTINUE 指令用于程序出现暂时错误时（如定时器错误），继续运行其它程序。这个功能不应该用于被 0 除、栈溢出、总线错误和界限错误等异常事件。</td></tr>
<tr><td>LD IN</td><td rowspan="5">
</td></tr>
<tr><td>CONTINUE</td></tr>
<tr><td>ST OUT</td></tr>
<tr><td>ST 编程语言</td></tr>
<tr><td>CONTINUE（IN）</td></tr>
<tr><td colspan="2">注:IL，ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量</td></tr>
</table>

➢ **CONTINUE 指令处理的数据类型**

输入和输出	数据类型	描述
IN（COND）	BOOL	如果为 TRUE，则继续程序的执行。
OUT	BOOL	如果 COND=TRUE，并且可以执行程序，则为 TRUE

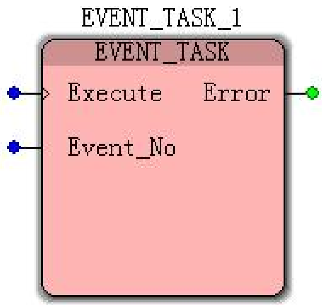
10.6.7 DERIVAT（微分）

IL 编程语言	LD, FBD 编程语言
功能：DERIVAT 指令用于对数据做时间的微分计算，使用微分指令时，需要把 POU 所在的任务类型设置为周期扫描（CYCLIC），并根据自己的需要设置扫描周期，任务类型的设置和扫描周期见本手册编程模型 硬件 Tasks.	
LD ENABLE	
ST DERIVAT_1.ENABLE	
LD RUN	
ST DERIVAT_1.RUN	
LD XIN	
ST DERIVAT_1.XIN	
LD CYCLE	
ST DERIVAT_1.CYCLE	
CAL DERIVAT_1	
LD DERIVAT_1.XOUT	
ST XOUT	
ST 编程语言	
DERIVAT_1（ ENABLE:= IN, RUN:=IN1, XIN:=IN2, CYCLE:=T）; OUT:=DERIVAT_1.XOUT;	
注:IL, ST 语言编程时需在当前 POU 的变量工作单中插入变量 ENABLE, RUN 等或使用常量	

➢ **DERIVAT 指令处理的数据类型**

输入和输出	数据类型	描述
ENABLE	BOOL	TRUE 时，执行功能块
RUN	BOOL	FALSE 时，功能块暂停，并且输出被置为 0
XIN	REAL	输入值
CYCLEX	TIME	时间常数，单位秒，本指令实际上是对 [XIN（n）-XIN（n-1）]/CYCLE 的微分
OUT	REAL	输出值，微分结果

10.6.8 EVENT_TASK （触发事件）

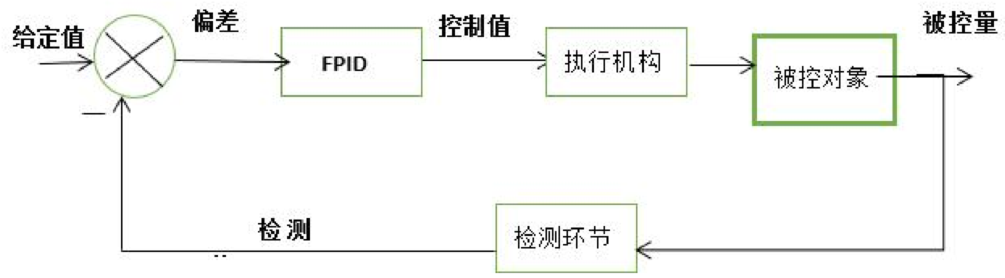
<table>
<tr><th>IL 编程语言</th><th>LD， FBD 编程语言</th></tr>
<tr><td colspan="2">功能：EVENT_TASK 指令用于触发一个事件任务，事件编号的可以定义，如果用户的一个事件任务存在该事件编号，那么它被激活，即程序被指派给该事件任务。</td></tr>
<tr><td>LD START</td><td rowspan="11">
</td></tr>
<tr><td>ST EVENT_TASK_1.Execute</td></tr>
<tr><td>LD IN</td></tr>
<tr><td>ST EVENT_TASK_1.Event No</td></tr>
<tr><td>CAL EVENT_TASK_1</td></tr>
<tr><td>LD EVENT_TASK_1.Erroe</td></tr>
<tr><td>ST OUT</td></tr>
<tr><td>ST 编程语言</td></tr>
<tr><td>EVENT_TASK_1
（Execute:=START,Event_No:=IN）;
OUT:=EVENT_TASK_1.Error;</td></tr>
<tr><td colspan="2">注:IL， ST 语言编程时需在当前 POU 的变量工作单中插入变量 START、IN 和 OUT 或使用常量</td></tr>
</table>

- **EVENT_TASK 指令处理的数据类型**

输入	数据类型	描述
START（Execute）	BOOL	上升沿有效
IN（Event_No）	UINT	事件编号
OUT（Error）	BOOL	错误码: 0 ——没有错误产生; 1 ——事件编号不在值的有效范围内

10.6.9 FPID

ProConOS 功能	功能
FPID	FPID 指令根据设定值与检测值之间的偏差自动计算控制值，使得检测值能够跟踪设定值，设定值是期望被控设备的一个行为保持的一个数值，检测值是通过仪表对被控设备的行为检测出的数值，而控制值是通过控制被控设备的这个行为或间接的行为的一个数值，这样就形成了一个闭环控制回路，如下图，FPID 是核心部分。



偏差=设定值一检测值。
FPID 指令在自动工作模式下，其输出值是经 PID 运算后的计算结果，如下

输出=kp$\left(e(t)+\frac{1}{Ti}\int_0^t e\,(\tau)_{d\tau}+T_d\frac{de(t)}{dt}\right)$FPID 指令在手动工作模式下，其输出值等于手动输出值

➢ **FPID 指令处理的数据类型**

输入和输出	类型	描述
REMOTE	BOOL	TRUE 时为远程设定，FALSE 时为本地设定
AUTO	BOOL	TRUE 时 FPID 为自动工作模式，FALSE 时为手动工作模式
DIRECTN	BOOL	TRUE 时为正作用（检测值高于设定值　控制值上升），FALSE 时为反作用　（检测值高于设定值　控制值下降）
INTLCK	BOOL	TRUE 时 FPID 的输出 Yout 被强制，强制值通过 FPID 的输入 INTLCKV 设置;FALSE 时输出不强制（输出 Yout 经比例积分微分计算值或手动值）
Tscan	REAL	时问常数，单位秒，一般可设为 REAL#0.1 o Tscan 值越大，控制作用越强
Yman	REAL	FPID 在手动工作模式下的输出值
SPR	REAL	远程设定值
SPL	REAL	本地设定值
X	REAL	检测值
KP	REAL	比例，用户可设一个初值，如 6.5
TI	REAL	积分，单位秒，用户可设一个初值，如 60
TD	REAL	微分，单位秒，用户可设一个初值，如 0
HIGH	REAL	输出 Yout 的上限
LOW	REAL	输出 Yout 的下限
INTLCKV	REAL	强制值，当 INTLCK 为 TRUE 时有效
输出	类型	描述
OUT	REAL	输出值

10.6.10 GET_ERROR（在错误目录中获得的错误的详细信息）

注:GET_ERROR 指令暂时不能用。

10.6.11 GET_ERROR_CATALOG（在错误目录中获得的当前内容的信息）

注:GET_ERROR_CATALOG 指令暂时不能用。

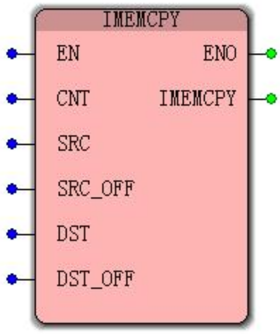
10.6.12 GET_SYM （搜索 PDD 变量的符号名称）

注:GET_SYM 指令暂时不能用。

10.6.13 HOT_RESTART（PLC 热启动）

注:HOT_RESTART 指令暂时不能用。

10.6.14 IMEMCPY（数据复制）

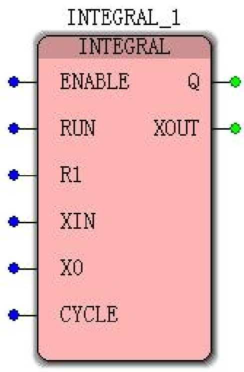
<table>
<tr><th>IL 编程语言</th><th>LD, FBD 编程语言</th></tr>
<tr><td colspan="2">功能：IMEMCPY 指令用于将数据从源数据区域复制到目标数据区域。</td></tr>
<tr><td>LD CNT</td><td rowspan="5">
</td></tr>
<tr><td>IMEMCPY SRC, SRC_OFF, DST,DST_OFF</td></tr>
<tr><td>ST OUT</td></tr>
<tr><td>ST 编程语言</td></tr>
<tr><td>OUT:=IMEMCPY （ CNT,SRC, SRC_OFF,DST,DST_OFF）;</td></tr>
<tr><td colspan="2">注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 CNT、SRC 等或使用常量</td></tr>
</table>

➢ **IMEMCPY 指令处理的数据类型**

输入和输出	数据类型	描述
CNT （CNT）	INT	将要复制的字节数

SRC[0]　（SRC）	BYTE	源数据区域的第一个字节，数据类型为 ARRAY 时写为 SRC[0]
SRC_OFF　（SRC OFF）	INT	源数据区域的起始字节序号，第一个字节的序号为 0
DST[0]　（DST）	BYTE	目标数据区域的第一个字节，数据类型为 ARRAY 时写为 DST[0]
DST_OFF　（DST OFF）	INT	目标数据区域的起始字节序号号，第一个字节的序号为 0
OUT　（IMEMCPY）	INT	错误代码: 0 复制了数据，无错误发生 14 缓冲区超过数据段 15 目标区域是一个输入组

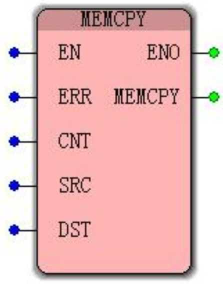
10.6.15 INTEGRAL（积分）

<table>
<tr><th>IL 编程语言</th><th>LD, FBD 编程语言</th></tr>
<tr><td colspan="2">功能：INTEG RAL 指令用于对数据做时间的积分计算，使用积分指令时，需要把 POU 所在的任务类型设置为周期扫描（CYCLIC ），并根据自己的需要设置扫描周期，任务类型的设置和扫描周期见本手册编程模型　硬件　Tasks.</td></tr>
<tr><td>见帮助文件</td><td rowspan="3">
</td></tr>
<tr><td>ST 编程语</td></tr>
<tr><td>INTEGRAL_1（ENABLE:=ENABLE,RUN:=RUN,R1:=R1,XIN:=XIN,XO:=X0, CYCLE:=CYCLE）;Q:=INTEGRAL_1.0;XOUT:=INTEGRAL_1.XOUT;</td></tr>
<tr><td colspan="2">注:IL,　ST 语言编程时需在当前 POU 的变量工作单中插入变量 E, RUN 等或使用常量</td></tr>
</table>

➢ **INTEGRAL 指令处理的数据类型**

输入和输出	数据类型	描述
ENABLE	BOOL	TRUE 时，执行功能块
RUN	BOOL	TRUE 时，开始积分，FALSE 时，积分暂停，并且输出保持最后的积分值
R1	BOOL	TRUE 时复位，复位值是 XO
XIN	REAL	输入值
XO	REAL	复位值
CYCLE	TIME	时间常数，单位秒，本指令实际上是对 XIN × CYCLE 进行积分
Q	BOOL	Q 等于 R1 取反
XOUT	REAL	输出值，积分结果

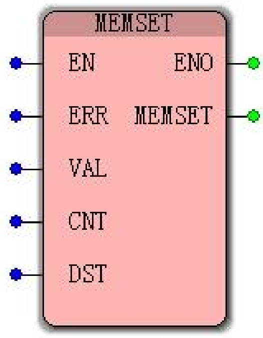
10.6.16 MEMCPY（数据复制指令）

IL 编程语言	LD, FBD 编程语言
功能：MEMCPY 指令用于将数据从源数据区域复制到目标数据区域。	
LD ERR	
MEMCPY CNT,SRC[0], DST[0]	
ST OUT	
ST 编程语言	
OUT:= MEMCPY（ERR, CNT,SRC[0],DST[0]）	
注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 ERR, CNT、SRC[0]等或使用常量	

> **MEMCPY 指令处理的数据类型**

输入和输出	数据类型	描述
ERR （ERR）	INT	错误代码:0 一一正确复制;14 一一缓冲区超过数据段;15—目标
CNT （CNT）	INT	区域是一个输入数组。注意:这是放置在左边的输出参数!
SRC[0] （SRC）	BYTE	要复制的字节数
DST[0] （DST）	BYTE	源数据区域的第一个字节，数据类型为 ARRAY 时写为 SRC[0]
OUT （IMEMCPY）	WORD	目标数据区域的第一个字节，数据类型为 ARRAY 时写为 SRC[0]

10.6.17 MEMSET（数据分发）

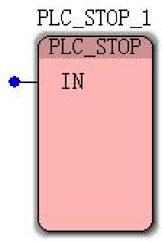
IL 编程语言	LD, FBD 编程语言
功能：MEMSET 指令用于将源数据分发到目标数据区域。	
LD ERR	
MEMSET VAL, CN 工 DST[0]	
ST OUT	
ST 编程语言	
OUT:=MEMSET （ERR, VAL, CNT,DST[0]）	

注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 ERR, VAL, CNT 等或使用常量

➢ **MEMSET 指令处理的数据类型**

输入和输出	数据类型	描述
ERR （ERR）	INT	错误代码:0 一一正确复制; 14 一一缓冲区超过数据段;15—目标区域是一个输入数组。注意:这是放置在左边的输出参数!
VAL （VAL）	BYTE	源数据一一要分发的数据
CNT （CNT）	DINT	要分发的份数,可把一个源数据分发到目标数据区中的 N 个字节中，一个字节一份
DST[0] （DST）	BYTE	目标数据区域的第一个字节，数据类型为 ARRAY 时写为 SRC[0]
OUT （IMEMCPY）	WORD	输出，暂没有定义，实际上字符是被拷贝到目标数据区 DST 了

10.6.18 PLC_STOP（PLC 停止）

<table>
<tr><th>IL 编程语言</th><th>LD, FBD 编程语言</th></tr>
<tr><td colspan="2">功能：PLC_STOP 指令用于停止 PLC，在冷再启动过程中，初始化所有数据。如程序发生堆栈溢出、字符串错误或者被 0 除等问题时，可以调用这个指令自动重新启动程序的执行。</td></tr>
<tr><td>LD IN</td><td rowspan="5">
</td></tr>
<tr><td>ST PLC_STOP_1.IN</td></tr>
<tr><td>CAL PLC_STOP_1</td></tr>
<tr><td>ST 编程语言</td></tr>
<tr><td>PLC_ STOP_ 1（IN）</td></tr>
<tr><td colspan="2">注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 或使用常量</td></tr>
</table>

➢ **P LC_STOP 指令处理的数据类型**

输入和输出	数据类型	描述
IN	BOOL	上升沿有效，PLC 停止

10.6.19 RD_*_BY_SYM （读 PDD 变量的值）

🕮 指令功能概述：RD_*_BY_SYM 包括以下指令，分别用于读取 PDD 不同数据类型的变量

RD_BOOL_BY_SYM	RD_BYTE _BY_SYM	RD_WORD_BY_SYM	RD_DWORD_BY_SYM
RD_SINT_BY_SYM	RD_INT_BY_SYM	RD_DINT_BY_SYM	RD_USINT_BY_SYM
RD_UINT_BY_SYM	RD_UDINT_BY_SYM	RD_REAL_BY_SYM	RD_STRING_BY_SYM
RD_TIME_BY_SYM	RD_INPUT_GROUP		

PDD 是通过变量的名称可以访问到该变量值的办法，是在控制器内核层访问 MULTIPROG 定义的 PLC 变量的值时所使用的一种方法，RD_*_BY_SYM 指令暂时不能使用，一般用户直接读写变量就可以了。

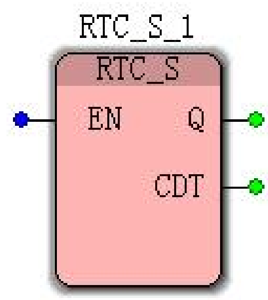
10.6.20 WR_*_BY_SYM（写 PDD 变量的值）

🕮 指令功能概述：WR_*_BY_SYM 包括以下指令，分别用于写 PDD 不同数据类型的变量

WR_BOOL_BY_SYM	WR_BYTE_BY_SYM	WR_WORD_BY_SYM	WR_DWORD_BY_SYM
WR_SINT_BY_SYM	WR_INT_BY_SYM	WR_DINT_BY_SYM	WR_USINT_BY_SYM
WR_UINT_BY_SYM	WR_UDINT_BY_SYM	WR_REAL_BY_SYM	WR_STRING_BY_SYM
WR_TIME_BY_SYM	WR_OUTPUT_GROUP		

PDD 是通过变量的名称可以访问到该变量值的办法，是在控制器内核层访问 MULTIPROG 定义的 PLC 变量的值时所使用的一种方法，WR_*_BY_SYM 指令暂时不能使用，一般用户直接读写变量就可以了。

10.6.21 RTC_S（读取 PLC 时钟）

<table>
<tr><th>IL 编程语言</th><th>LD, FBD 编程语言</th></tr>
<tr><td colspan="2">功能：RTC_S 指令用于读取 PLC 的时钟放在一个字符串变量中，读取的时钟为 GMT 格式。
IEC61131-3 规定的口期和时间输出字符串格式为:DT#1998-11-21-15:27:56.46。</td></tr>
<tr><td>LD IN</td><td rowspan="9">
</td></tr>
<tr><td>ST RTC_S_1.EN</td></tr>
<tr><td>CAL RTC_S_1</td></tr>
<tr><td>L D RTC_S_1.Q</td></tr>
<tr><td>ST Q</td></tr>
<tr><td>LD RTC_S_1.CDT</td></tr>
<tr><td>ST OUT</td></tr>
<tr><td>ST 编程语言</td></tr>
<tr><td>RTC_S_1 （ EN:= （ IN ） ） ;
Q:=RTC_S_1.Q;
OUT:=RTC_S_1.CDT;</td></tr>
<tr><td colspan="2">注:IL, ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN, O, OUT 或使用常量</td></tr>
</table>

➢ **RTC_S 指令处理的数据类型**

输入和输出	数据类型	描述
IN （EN）	BOOL	如果为 TRUE，则实际的日期和时间被写入所连结的输出字符串中
Q （Q）	BOOL	如果 EN 为 TRUE， Q 为 TRUE，否则 Q 为 FALSE

OUT （CTD）	STRING	取得的日期与时间，如 DT#2011-08-15-10:08:55.19

10.6.22 WARM_RESTART（PLC 暖启动）

注： WARM_RESTART 指令暂时不能用。

第十一章 运动指令

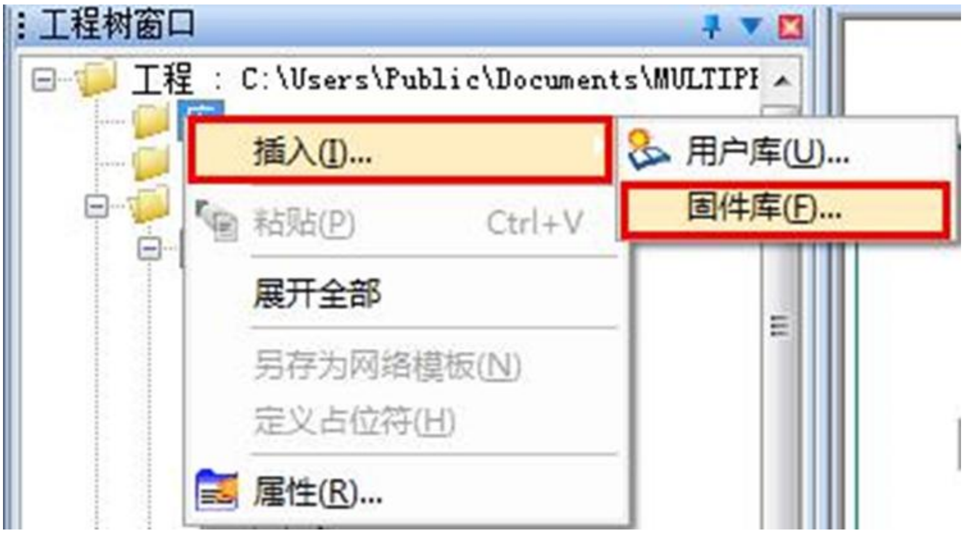
11.1 插入 FB_FU_LIB（运动控制固件库）

11.1.1 功能介绍

FB_FU_LIB 固件库是专为运动控制而提供的，作为一个库函数给用户使用，用户无需复杂的编程，只需调用和设置一些简单参数就可以使用。运动控制库包含丰富的运动指令，如单轴运动的绝对点位，相对点位功能，多轴运动的电子凸轮，电子齿轮，叠加功能等。

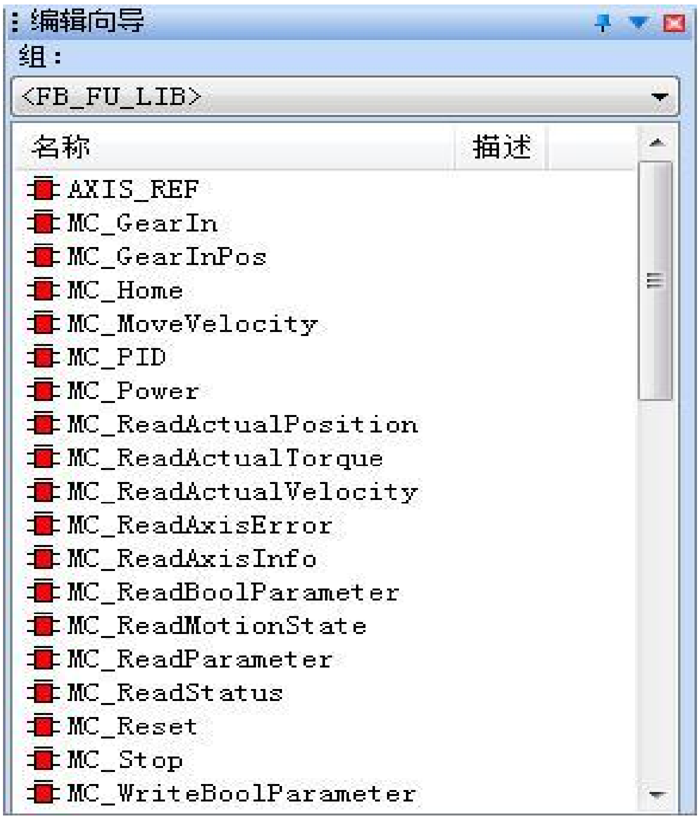
11.1.2 添加固件库

（1）在工程树窗口下右键“库”，选择“插入”选择“固件库” 如图所示。



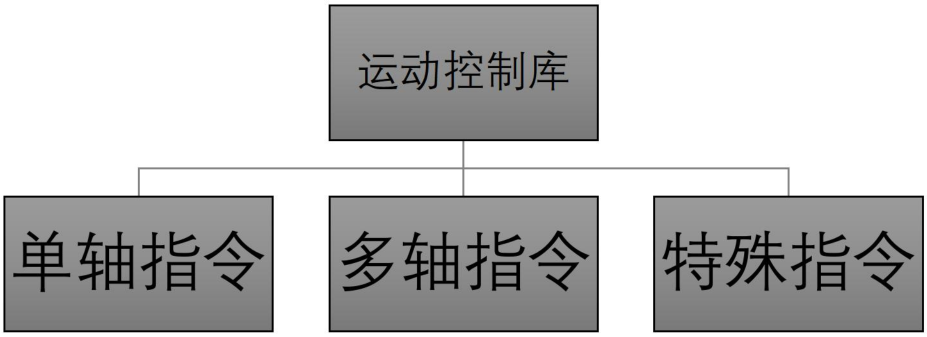
（2）查找存放的“FB_FU_LIB”文件的位置，找到此文件，单击“包括”按钮后，完成后如图所示

（3）在“编辑向导”组下单击“FB_FU_LIB”可查看到新插入运动控制库如下图所示


编辑向导
组：
<FB_FU_LIB>
名称
描述
AXIS_REF
MC_GearIn
MC_GearInPos
MC_Home
MC_MoveVelocity
MC_PID
MC_Power
MC_ReadActualPosition
MC_ReadActualTorque
MC_ReadActualVelocity
MC_ReadAxisError
MC_ReadAxisInfo
MC_ReadBoolParameter
MC_ReadMotionState
MC_ReadParameter
MC_ReadStatus
MC_Reset
MC_Stop
MC_WriteBoolParameter

11.2 运动指令

11.2.1 运动控制库分类



11.2.2 运动指令一览表

指令组	指令码	功能	页码
单轴指令	MC_Power	使能指令	11.4.1 MC_Power（使能指令）
	MC_MoveVelocity	速度指令	11.4.2 MC_MoveVelocity(速度指令)
	MC_MoveRelative	相对位移指令	11.4.3 MC_MoveRelative（相对位移指令）
	MC_MoveAdditive	附加位移指令	11.4.4 MC_MoveAdditive（附加位移指令）
	MC_MoveAbsolute	绝对位移指令	11.4.5 MC_MoveAbsolute（绝对位移指令）
	MC_MoveSuperimposed	追加位移指令	11.4.6 MC_MoveSuperimposed(追加位移指令)
	MC_HaltSuperimposed	暂停追加位移指令	11.4.7 MC_HaltSuperimposed(暂停追加位移)
	MC_Home	原点回归指令	11.4.8 MC_Home（原点回归指令）
	MC_SetOverride	速度超调指令	11.4.9 MC_SetOverride(速度超调指令)
	MC_Stop	停止指令	11.4.10 MC_Stop（停止指令）
	MC_Halt	暂停指令	11.4.11 MC_Halt（暂停指令）
	MC_SpecialMoveAbsolute	特殊绝对位	11.4.11

		移指令	MC_SpecialMoveAbsolute（特殊绝对位移指令）
	MC_ReadActualPosition	读取实时位置指令	11.4.12 MC_ReadActualPosition（读取实时位置指令）
	MC_ReadActualVelocity	读取实时速度指令	11.4.13 MC_ReadActualVelocity（读取实时速度）
	MC_ReadMotionState	读取轴运动状态指令	11.4.14 MC_ReadMotionState（读取轴运动状态指令）
	MC_ReadStatus	读取轴状态指令	11.4.15 MC_ReadStatus（读轴状态）
	MC_SetPosition	位置设置指令	11.4.16 MC_SetPosition（位置设置指令）
	MC_Phasing	偏移主轴指令	11.4.17 MC_Phasing（偏移主轴指令）
	MC_TouchProbe	位置捕获指令	11.4.18 MC_TouchProbe(位置捕获指令）
	MC_AbortTrigger	位置捕获打断指令	11.4.19 MC_AbortTrigger(位置捕获打断指令)
	NS_MC_Jog	点动指令	11.4.20 NS_MC_Jog(点动指令)
	NS_MC_StopByPos	指定模相位停止指令	11.4.21 NS_MC_StopByPos(指定模位置停止指令)
	NS_MC_ReadParameter	读取参数指令	11.4.22 NS_MC_ReadParameter（读取参数指令）
多轴指令	MC_GearIn	电子齿轮耦合指令	11.5.1 MC_GearIn（电子齿轮耦合指令）
	MC_GearOut	电子齿轮脱离指令	11.5.2 MC_GearOut（电子齿轮脱离指令）
	MC_CombineAxes	双主轴联合齿轮指令	11.5.3 MC_CombineAxes（双主轴联合齿轮指令）
	NS_MC_RotaryCutIn	旋切指令	11.5.9 NS_MC_RotaryCutIn（旋切指令）
	NS_MC_SpecialCamIn	特殊凸轮指令	11.5.10 NS_MC_SpecialCamin（特殊凸轮指令）
	NS_MC_SpecialCombineAxes	特殊双主轴联合齿轮指令	11.5.11 NS_MC_SpecialCombineAxes（特殊双主轴联合齿轮指令）
	MC_CamIn	电子凸轮关联指令	11.5.12 MC_CamIn（电子凸轮关联指令）

	MC_CamOut	电子凸轮脱离指令	11.5.13 MC_CamOut（电子凸轮脱离指令）
	MC_CamWritePoint	写入凸轮点信息指令	11.5.14 MC_CamWritePoint（写入凸轮点信息指令）
	MC_CamReadPoint	读取凸轮点信息指令	11.5.15 MC_CamReadPoint（读取凸轮点信息指令）
	MC_CamSet	更改凸轮点生效指令	11.5.16 MC_CamSet（更改凸轮点生效指令）
	MC_ReadTappetStatus	读取多个挺杆点状态指令	11.5.17 MC_ReadTappetStatus（读取多个挺杆点状态指令）
	MC_ReadTappetValue	读取多个挺杆点信息指令	11.5.18 MC_ReadTappetValue（读取单个挺杆点信息指令）
	MC_WriteTappetValue	修改挺杆点信息指令	11.5.19 MC_WriteTappetValue（修改挺杆点信息指令）
特殊指令	NS_CC_ADC	AD 指令	11.6.1 NS_CC_ADC(AD 指令)
	NS_CC_DAC	DA 指令	11.6.2 NS_CC_DAC(DA 指令)
	EX_ADC	AD 扩展指令	11.6.3 EX_ADC（AD 扩展指令）
	EX_DAC	DA 扩展指令	11.6.4 EX_DAC（DA 扩展模块）
	NS_CC_NOoutput	禁止 QXX 输出指令	11.6.5 NS_CC_NOoutput（禁止 QXX 输出指令）
	NS_CC_Counter	高速计数指令	11.6.6 NS_CC_Counter（高速计数器指令）
	NS_CC_CNTI	高速计数中断指令	11.6.7 NS_CC_CNTI(高速计数中断指令)
	NS_CC_CNT_Out	区间比较输出指令	11.6.8 NS_CC_CNT_Out（区间比较输出指令）
	NS_CC_DI_Counter	DI 高速计数指令	11.6.9 NS_CC_DI_Counter（DI 高速计数指令）
	NS_CC_EXTI	DI 中断指令	11.6.10 NS_CC_EXTI(DI 中断指令)
	NS_CC_ReadPulseVelocity	读取脉冲速率指令	11.6.11 NS_CC_ReadPulseVelocity（读取被控轴脉冲速率）
	RTC_S	时钟特殊寄存器	11.6.12 RTC_S（时钟特殊寄存器）
G 代码指令	NC_GroupEnable	轴组使能指令	11.7.1 NC_GroupEnable(轴组使能指令)
	NC_MoveLiner	直线插补指令	11.7.2 NC_MoveLiner（直线插补）
	NC_MoveCircula	圆弧插补指令	11.7.3 NC_MoveCircula（圆弧插补）

	NC_CartesianCoordinate	直角坐标系指令	11.7.4 NC_CartesianCoordinate(直角坐标机器人指令)
CANopen 代码指令	NS_CC_CANopen_NMT_Read	网络状态读取指令	3.1.1 NS_CC_CANopen_NMT_Read（网络状态读取指令）
	NS_CC_CANopen_NMT_Write	网络状态写入指令	3.1.2 NS_CC_CANopen_NMT_Write（网络状态写入指令）
	NS_CC_CANopen_PDO_Comm	配置 PDO 过程数据通信参数	3.1.3 NS_CC_CANopen_PDO_Comm（配置 PDO 过程数据通信参数）
	NS_CC_CANopen_PDO_Map	配置 PDO 过程数据映射参数	3.1.4 NS_CC_CANopen_PDO_Map（配置 PDO 过程数据映射参数）
	NS_CC_CANopen_RPDO	PDO 数据映射区读取指令	3.1.5 NS_CC_CANopen_RPDO（PDO 数据映射区读取指令）
	NS_CC_CANopen_TPDO	PDO 数据映射区赋值指令	3.1.6 NS_CC_CANopen_TPDO（PDO 数据映射区赋值指令）
	NS_CC_CANopen_SDO_Read	服务数据读取指令	3.1.7 NS_CC_CANopen_SDO_Read（服务数据读取指令）
	NS_CC_CANopen_SDO_Write	服务数据赋值指令	3.1.8 NS_CC_CANopen_SDO_Write（服务数据赋值指令）

11.3 运动控制指令基本知识

11.3.1 运动控制器通常采用指令方式

数字脉冲： 这种方式与步进电机的控制方式类似，运动控制器给伺服驱动器发送“脉冲/方向”或“CW/CCW”类型的脉冲指令信号，我司只支持 AB 脉冲；伺服驱动器工作在位置控制模式，位置闭环由伺服驱动器完成。日系伺服和国产伺服产品大都采用这种模式。其优点是系统调试简单，不易产生干扰，但缺点是伺服系统响应稍慢。

模拟信号： 这种方式下，运动控制器给伺服驱动器发送±10V 的模拟电压指令，同时接收来自电机编码器或直线光栅等位置检测元件的位置反馈信号；伺服驱动器工作在速度控制模式，位置闭环由运动控制器完成。欧美的伺服产品大多采用这种工作模式。其优点是伺服响应快，但缺点是对现场干扰较敏感，调试稍复杂。

CANopen 通信： 这种方式是通过通信协议来控制伺服驱动器运作，详细介绍见附录四。

以下介绍运动控制器用模拟量信号控制伺服轴的一般调试步骤：

（1）初始化参数

确认伺服驱动器接线无误后，先初始化伺服驱动器的参数（恢复出厂设置）。伺服驱动器完成出厂设置后：设置控制方式；设置使能由外部控制；编码器信号输出的齿轮比；设置控制信号与电机转速的比例关系（模拟量输出电压对应伺服轴的转速）。

（2）接线

连接运动控制器与伺服之间的信号线。以下的连线是必须的：运动控制器的模拟量输出线、伺服使能控制信号线、伺服输出的编码器信号线。（详情请参考 11.4.2 接线方式）

（3）试方向

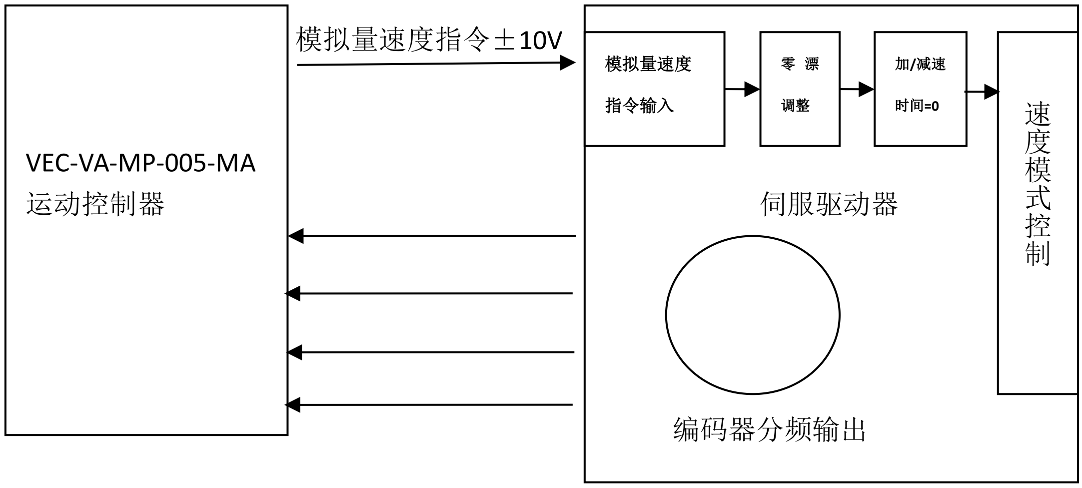
对于一个闭环控制系统，如果反馈信号的方向不正确，后果肯定是灾难性的。通过运动控制器编程输出 Q0 控制伺服驱动器的使能信号。此时伺服轴应该以一个较低的速度转动，这就是所谓的“零漂”。执行运动控制器指令模块（DA 模块）。使用这个指令模块，看电机的转速和方向是否可以通过该指令（参数）可控，并且监视编码器反馈信号是否是一致（即：确保模拟量给定为正电压时，编码器方馈为递增；模拟量给负电压时，编码器反馈为递减）。如果不能控制，或者编码器反馈不正确，检查模拟量接线，编码器反馈线和控制方式的参数设置。

（4）抑制零漂

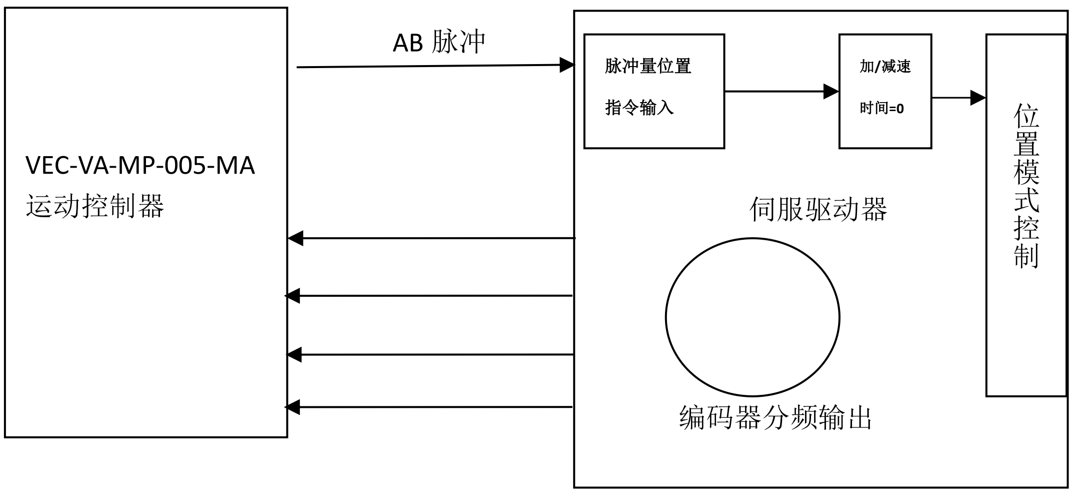
在闭环控制过程中，零漂的存在会对控制效果有一定的影响，最好将其抑制住。配合运动控制器上的指令模块或调整伺服上的零漂值来抑制零飘现象，即可实现（可参考模拟量零漂调整），使电机的转速趋近于零。由于零漂本身也有一定的随机性，所以，不必要求电机转速绝对为零。

11.3.2 运动控制原理

VEC-VA-MP-005-MA 运动控制器使用模拟量±10V 控制伺服驱动器运行于速度模式，通过伺服驱动器分频输出编码器信号反馈到运动控制器，位置闭环由运动控器器完成，如下图所示；

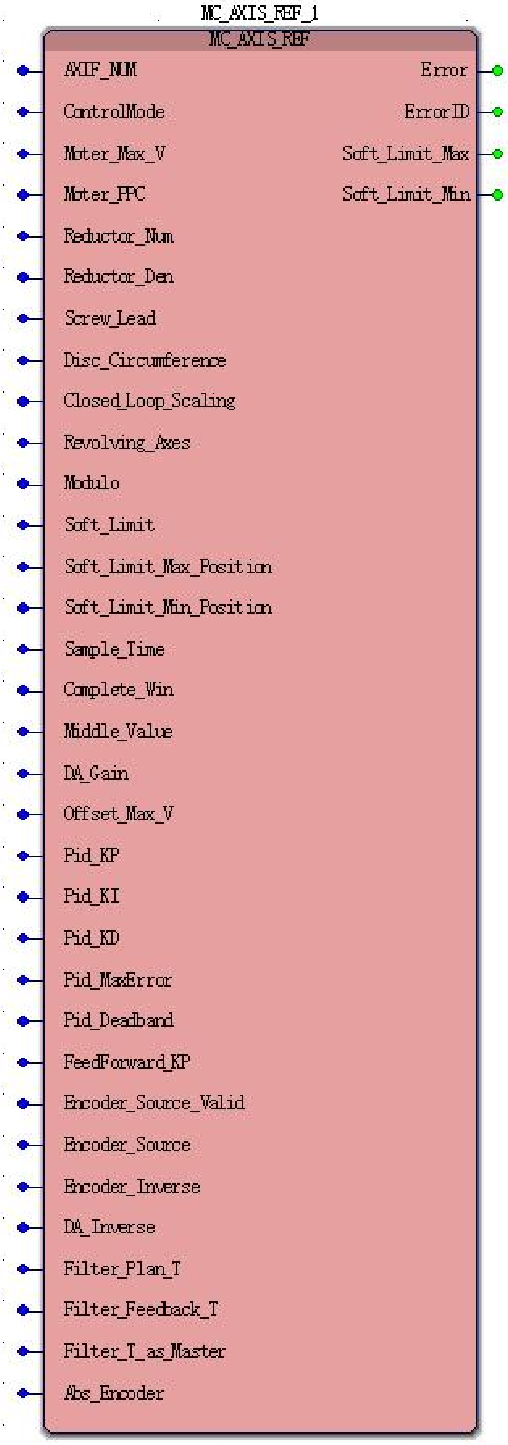


VEC-VA-MP-005-MA 运动控制器使用脉冲量控制伺服驱动器运行于位置模式，通过伺服驱动器分频输出编码器信号反馈到运动控制器，位置闭环由伺服驱动器完成，如下图所示；



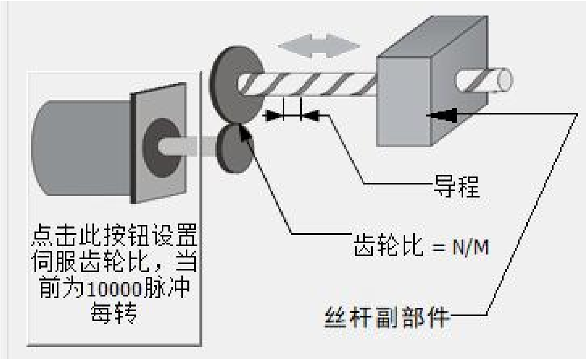
11.3.3 MC_AXIS_REF(轴参数设定)

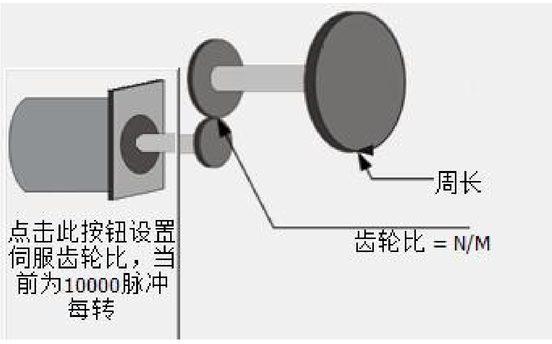
FB/FU	说明	适用机种
FB	此指令用于配置被控伺服轴 AXISXXX 的相关参数	VEC-VA-MP-005-MA



➢ **输入参数**

<table>
<tr><th>名称</th><th>功能</th><th>数据类型</th><th>设定范围（缺省值）</th></tr>
<tr><td>Axis_Num（轴号）</td><td>设定指令欲控制的轴</td><td>USINT</td><td>模拟量/脉冲：
0~4（实轴）
5~11（虚轴）
CANopen 模式：0~15
(实轴/虚轴)
(0)</td></tr>
<tr><td colspan="4">轴号说明：
Axis_Num 是被控轴的轴号，模拟量或脉冲方式控制下，轴号 0~4 为实轴，5~11 为虚轴，与实轴相比，虚轴没有实际的控制作用；
CANopen 模式下，轴号 0~15 均可作实轴或虚轴，配置了网络的为实轴，没有配置网络的为虚轴，在 CANopen 给定的模版中，控制的节点号=轴号+1。</td></tr>
<tr><td>ControlMode(控制模式)</td><td>选择运动控制器控制命令的输出模式
0：模拟量控制
1：脉冲量控制
2：CANopen 控制</td><td>INT</td><td>0：模拟量控制
1：脉冲量控制
2：CANOPEN 控制
(0)</td></tr>
<tr><td colspan="4"></td></tr>
<tr><td>Motor_Max_V（最大转速）</td><td>伺服驱动器允许电机最大转速
单位：转/分钟</td><td>DINT</td><td>正数</td></tr>
<tr><td colspan="4">运动控制器输出±10V 模拟量时对应伺服驱动器最大正反转速； 即每伏对应的伺服转速。 例如：伺服驱动器模拟量增益设为 300（r/min）/v，则该参数为 10*300=3000 r/min</td></tr>
<tr><td>Motor_PPC
（每转脉冲数）</td><td>电机的每转脉冲数</td><td>DINT</td><td>正数</td></tr>
<tr><td colspan="4">被控伺服驱动器每转分频输出的脉冲数乘以 4 倍频，不一定是电机编码器的分辨率*4 倍频（部分伺服可支持编码器分频输出设置）</td></tr>
<tr><td>Reductor_Num
（减速齿轮比分子）</td><td>电机轴到执行终端的减速比分子</td><td>LREAL</td><td>正数</td></tr>
<tr><td colspan="4">+</td></tr>
<tr><td>Reductor_Den
（减速齿轮比分母）</td><td>电机轴到执行终端的减速比分母</td><td>LREAL</td><td>正数</td></tr>
<tr><td colspan="4"></td></tr>
<tr><td>Screw_Lead
（丝杆导程）</td><td>丝杆导程，丝杆转一圈终端行进的距离
单位：单元</td><td>LREAL</td><td>正数</td></tr>
<tr><td colspan="4">当 Disc_Circumference 为 0 时，默认终端传动装置为丝杆。</td></tr>
</table>

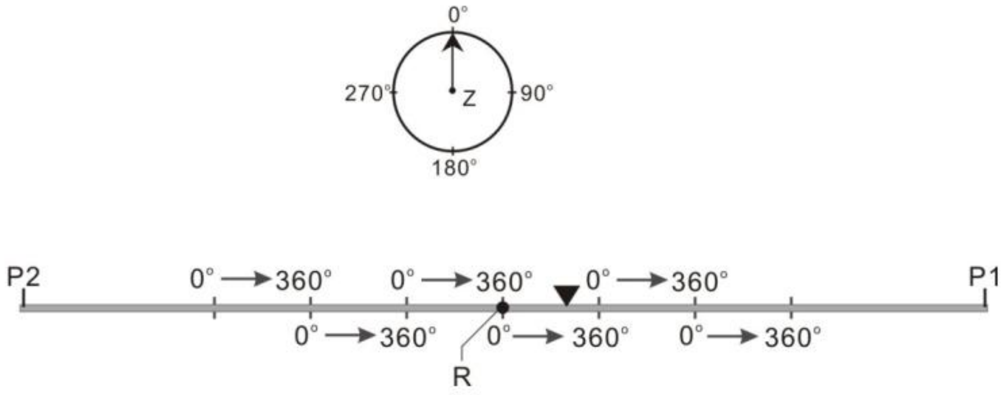
<table>
<tr><td colspan="4">

设置终端传动装置丝杆的导程，当减速比分子/减速比分母=1/2 时，表示伺服电机转两圈丝杆副部件移动的距离</td></tr>
<tr><td>Disc_Circumference
（圆盘周长）</td><td>终端的圆盘周长
单位：单元</td><td>LREAL</td><td>正数</td></tr>
<tr><td colspan="4">当 Disc_Circumference 不为零时，Screw_Lead 无效，默认终端传动装置为圆盘


设置终端传动装置圆盘的周长，当减速比分子/减速比分母=1/2 时，表示伺服电机转两圈圆盘转过的弧长为其周长</td></tr>
<tr><td>Closed_Loop_Scaling
（双闭环系数）</td><td>双闭环系数，终端机构每米电机脉冲数/外部编码器每米脉冲数，如果没有外部编码器此参数设 1.0</td><td>LREAL</td><td>正数</td></tr>
<tr><td colspan="4"></td></tr>
<tr><td>Revolving_Axes
（被控轴类型）</td><td>被控轴类型设定
0：直线轴
1：旋转轴</td><td>BOOL</td><td>0：直线轴
1：旋转轴</td></tr>
<tr><td colspan="4">直线轴：


直线轴模型
直线轴模型注解：
P1:正极限
P2:反极限
Origin：原点</td></tr>
</table>

旋转轴：



旋转轴模型（“模”设为 360）

旋转轴模型注解：

P1:正极限

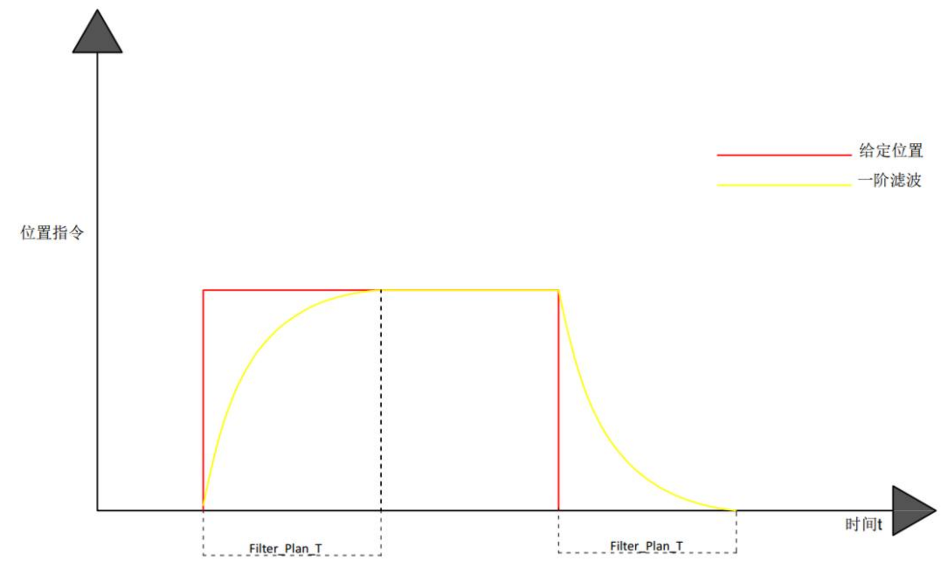
P2：反极限

直线轴和旋转轴的差异：

直线轴和旋转轴的差异主要在于旋转轴以模为周期。若直线轴终端执行机构的位置 500，对应旋转轴的位置为 140（模为 360 时），计算方法为 500 除以模后所得的余数。

Modulo （模）	设定旋转轴的模 单位：单元	LREAL	正数
用以平分终端执行机构实际位置的周期			
Soft_Limit （软限位）	当 Soft_Limit=1 时，开启软限位功能	BOOL	TRUE 或 FALSE
Soft_Limit_Max_Position (软限位最大值)	软限位最大值，最大位置超过此值则 Soft_Limit_Max 输出 TRUE	LREAL	正数、负数、零（0）
Soft_Limit_Min_Position （软限位最小值）	软限位最小值，最小位置低于此值则 Soft_Limit_Min 输出 TRUE	LREAL	正数、负数、零（0）
Sample_Time （采样时间）	设置采样编码器反馈脉冲的时间（单位：ms）	WORD	正数
设置采样编码器反馈脉冲的时间。当 Abs_Encoder=2 时，Sample_Time 的范围必须为[1,3]			
Complete_Win （点位完成窗口）	保留	DINT	保留

Middle_Value （模拟量零漂值）	模拟量零漂值	DINT	正数、负数、0 （0）
模拟量零漂值为 0 时，模拟量输出 0V。当选择 ControlMode=0 时，该值需与 11.3.5 模拟量零漂调整中 DAC_Value 的值相同。			
DA_Gain （模拟量增益）	设定模拟量增益	DINT	正数、负数、0 （0）
设定模拟量增益，一般设为 0。模拟量偏置，调整输出模拟量线性度，保证控制器输出的模拟量与伺服模拟量增益一致性，以提高控制精度。（1+DA_Gain/10000）*原模拟量输出。			
Offset_Max_V （最大补偿速度）	最大补偿速度 单位：转/分钟	DINT	正数、0 （0）
伺服电机最大的补偿速度，在模拟量闭环控制存在误差时，伺服电机将得到一个补偿速度，以达到精确地控制位置。这个补偿速度的最大值为 Offset_Max_V。			
Pid_KP	比例增益	DINT	正数、0 （0）
当选择控制模式为模拟量控制，即 ControlMode=0 时，此参数有效。比例增益与伺服驱动器的位置环比例增益类似，当位置闭环控制在运动控制器上完成，增大该值可缩短定位的时间；当该值为 0 时，控制器将不对位置闭环进行调节；（在电机不抖动的情况可增大该值）			
Pid_KI	积分增益	DINT	正数、0 （0）
当选择控制模式为模拟量控制，即 ControlMode=0 时，此参数有效。积分增益与伺服的位置环的积分增益类似，适当增加该值时可降低累积误差，该值设置过大时将会引起电机的抖动；			
Pid_KD	微分增益	DINT	正数、0 （0）
当选择控制模式为模拟量控制，即 ControlMode=0 时，此参数有效。积分增益与伺服的位置环的微分增益类似，一般不启用微分			
Pid_MaxError	PID 最大误差	DINT	正数、0 （0）
当误差脉冲数超过该值后，积分增益无用，一般设为零。			
Pid_Deadband	PID 死区	DINT	正数、0 （0）
PID 死区指跟随误差（误差脉冲数）在该数值以内，不进行 PID 调整.			
FeedForward_KP	前馈增益	DINT	正数、0 （0）
传统的 P 控制需要一个跟随误差（设定值-实际值）这样造成了轴的轮廓错误，动态响应性差，在执行轮廓上升期间， 适当增加前馈增益可以减小在运行过程中的跟随误差。			
Encoder_Source_Valid （编码器来源有效位）	编码器来源有效位	BOOL	TRUE/FALSE
当该值为 TRUE 时，选择 Encorder_Source 所设定的轴口作为编码器信号来源口。 当该值为 FALSE 时，选择轴参数功能块上 Axis_Num 所设定的轴口作为编码器信号来源口。			
Encoder_Source （编码器来源）	设定编码器信号来源	WORD	0~4

<table>
<tr><td colspan="4">配合 Encorder_Source_Valid，设定编码器信号来源。</td></tr>
<tr><td>Encoder_Inverse
（编码器反向有效位）</td><td>编码器反向有效位</td><td>BOOL</td><td>TRUE/FALSE</td></tr>
<tr><td colspan="4">当该值为 TRUE 时，将轴口接收到的位置脉冲进行取反计数</td></tr>
<tr><td>DA_Inverse
（模拟量反向）</td><td>DA 反向</td><td>BOOL</td><td>TRUE/FALSE</td></tr>
<tr><td colspan="4">DA_Inverse 有两种模式 0 和 1
1）DA_Inverse=0 时，控制器逆时针控制伺服电机，即模拟量输出为正电压;
2）DA_Inverse=1 时，控制器顺时针控制伺服电机，即模拟量输出为负电压;
特别注意：DA_Inverse 的两种模式，需要与编码器方向配和使用，否者无法形成闭环控制
例如：往下指令例子中，当 MC_Power 执行的时候，如果伺服轴能定位，则该值不需要更改，如果电机以一个设定的补偿速度 Offset_Max_V 运行 ，此时将该值由原来的 0 改为 1 或由原来的 1 改为 0，或者将编码器 A/B 线调换任意一项，也可通过 Encoder_Inverse 或寄存器 MB3.9654 设置修改编码器方向；特殊寄存器地址（%MB3.9654）数值对应被修改轴 AXISXX（XX 表示 0~4）关系如下
<table>
<tr><th>特殊寄存器地址</th><th>数值（二进制）</th><th>被修改轴</th></tr>
<tr><td>%MB3.9654</td><td>0000 0001（1 十进制）</td><td>AXIS0</td></tr>
<tr><td>%MB3.9654</td><td>0000 0010（2 十进制）</td><td>AXIS1</td></tr>
<tr><td>%MB3.9654</td><td>0000 0100（4 十进制）</td><td>AXIS2</td></tr>
<tr><td>%MB3.9654</td><td>0000 1000（8 十进制）</td><td>AXIS3</td></tr>
<tr><td>%MB3.9654</td><td>0001 0000（16 十进制）</td><td>AXIS4</td></tr>
</table>
例如：修改轴 AXIS0 反馈编码器计数方向；只需在特殊寄存器%MB3.9654 填 1 ，即可由原来的递增变为递减，或者由递减变为递增。如果需要修改多个轴将对应的位写 1 即可。</td></tr>
<tr><td>Filter_Plan_T
（给定滤波常数）</td><td>对给定位置和给定速度进行滤波</td><td>DINT</td><td></td></tr>
<tr><td colspan="4">Filter_Plan_T 的单位为 2ms，使能中可更改。下图黄色曲线为红色曲线经过一阶滤波后的结果

</td></tr>
<tr><td>Filter_Feedback_T
（反馈滤波）</td><td>对反馈速度进行滤波</td><td>DINT</td><td></td></tr>
</table>

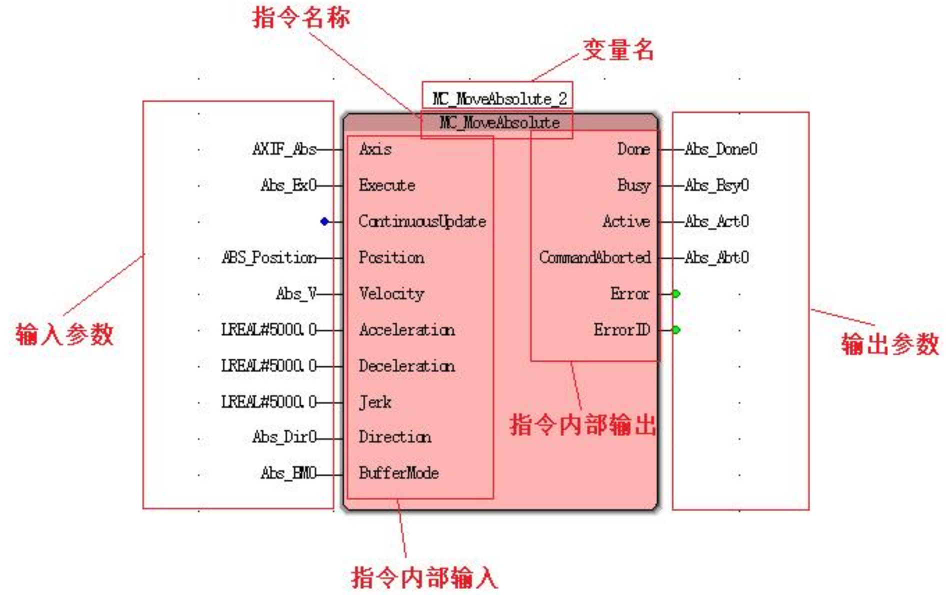
<table>
<tr><td colspan="4">Filter_Feedback_T 的单位为底层周期 2ms。使能中可更改。</td></tr>
<tr><td>Filter_T_as_Master</td><td>对主轴速度进行滤波</td><td>DINT</td><td></td></tr>
<tr><td colspan="4">设置 Filter_T_as_Master 表示将该轴作为主轴时，将其输出给从轴的实时速度进行滤波。</td></tr>
<tr><td>Abs_Encoder</td><td>设定绝对值编码器</td><td>USINT</td><td>0~6
（0）</td></tr>
<tr><td colspan="4">设定绝对值编码器类型，只有轴口 4 能接入绝对值编码器
SP 和 MP 型号的 VA 运动控制器：
0：不启用
1：启用 17 位绝对值编码器（多摩川）（按 2 的 17 次方溢出，不建议使用）
2：启用 24 位绝对值编码器（尼康）（按 2 的 24 次方溢出，不建议使用）
3：启动 23 位绝对值编码器（多摩川）（按 2 的 23 次方溢出，不建议使用）
4：启用 17 位绝对值编码器（多摩川）（按 32 位溢出，建议使用）
5：启用 24 位绝对值编码器（尼康）（按 32 位溢出，建议使用）
6：启动 23 位绝对值编码器（多摩川）（按 32 位溢出，建议使用）</td></tr>
</table>

模块说明：

- 使用伺服轴控制伺服电机时，必须根据机械参数正确配置该被控伺服轴的 AXIS_REF 模块，否者伺服轴无法正确控制伺服电机运行；
- 往下的单轴指令和多轴指令程序范例中，都需要调用 AXIS_REF 模块，并且正确配置相关的信息（往下的程序示范将不再重复说明）；
- 整个工程每个轴最多使用一个该模块

11.3.4 运动指令构成

运动指令构成如图所示



11.3.5 模拟量零漂调整

零漂定义：模拟量零漂指的是放大器在输入信号为零的时候，输出不为零的现象就叫零点漂移。也就是：当放大器的输入端短路时，在输出端有不规律的、变化缓慢的电压产生的现象。

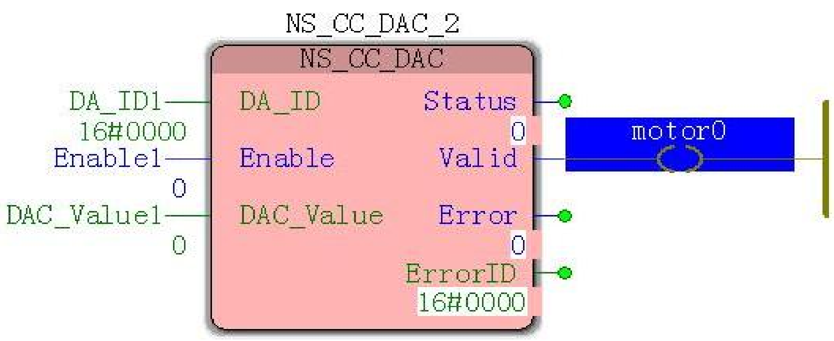
运动控制器指令模块零飘调节步骤；（这里以分别以威科达 VB 伺服驱动器和 VC 伺服驱动器进行说明）

➢ **威科达 VB 伺服驱动器：**

(1)参考 11.6.2 NS_CC_DAC(DA 指令)DA 模块的使用说明以及数模转换关系，完成后进行编程。

(2)在编辑向导中，调出“NS_CC_DAC”模块 ，这里是调节轴 0（AXIS0）的零漂，DA_ID 的初值填“0 ”其他轴依次类推（0~3），控制多轴时可同时调出多个“NS_CC_DAC”模块。

(3)根据模块输入端的数据类型填写变量，变量名可以自行命名，但要保证变量名不重复用户无需填写实际的物理地址，软件自动分配地址。完成后单击“制作”无错误提示后进行下装。



(4)在线调试模式下，Enable 由 FALSE 变为 TRUE 时（确保伺服已经处于使能状态下），如果

电机以零漂的速度在运行，此时调整 DAC_Value 的值，保证伺服驱动器处于静止状态，Enable 由 TRUE 变为 FALSE，零漂调整完成，然后在填写轴参数时将 DAC_Value 的值作为 MC_AXIS_REF（轴参数）模块 Middle_Value 的初值 。

➢ **威科达 VC 伺服驱动器：**

步骤（1）（2）（3）与 VB 伺服驱动器相同，步骤（4）在线调试模式下，Enable 由 FALSE 变为 TRUE 时（确保伺服已经处于使能状态下），如果电机以零漂的速度在运行，通过 BOP 面板对伺服的 P06.68（AI1 零漂 mV）/P06.73（AI2 零漂 mV）/P06.78（AI3 零漂 mV）进行调整，保证伺服驱动器处于静止状态，零漂调整完成。在填写轴参数时将 0 作为 MC_AXIS_REF（轴参数）模块 Middle_Value 的初值 。

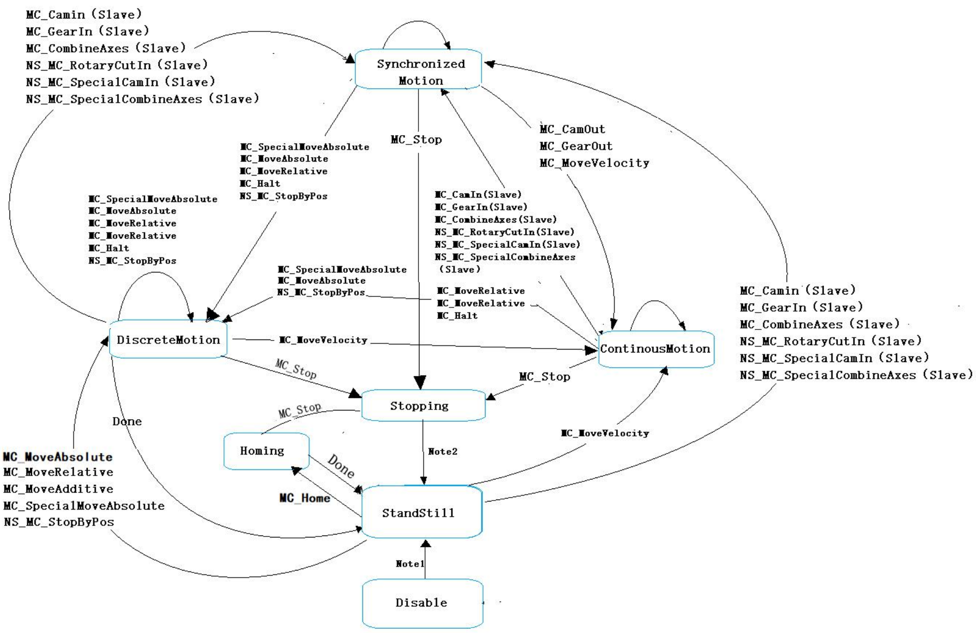
注意：

1）完成零漂调节后，确定该模块上的 Enable 状态为 False，不允许“NS_CC_DAC 模块”和“MC_Power 模块”在程序执行过程中输入参数 Enable 的状态同时为“TRUE”，否则运行功能模块时电机将会一顿一顿地运行。

2）往下介绍的例子都将是默认是零漂已经调整后的例子，往下的程序示范将不再重复说明。

11.3.6 状态机

VEC-VA-MP-005-MA 运动控制器在使用运动控制指令控制每个轴时，每个轴都有一个内部运行状态，被控轴的状态切换遵循下图所示的状态机。状态机定义了各个状态下可以执行的运动指令及执行运动指令后的状态，用户在使用运动指令时，可以通过状态机判断某一运动指令在当前状态下是否可以使用。VEC-VA-MP-005-MA 运动控制器的状态机如下图所示，箭头指示部分为轴的状态。



Note1：使用 MC_Power 指令使能并且 MC_Power. Status 为 True。

Note2：MC_Stop.Done 为 True 并且 MC_Stop.Execute 为 False。

序号	轴状态	说明
1	StandStill	准备执行状态
2	Disable	未执行状态
3	Stopping	停止状态
4	Homing	原点回归状态
5	Discrete Motion	离散运动状态
6	Continuous Motion	连续运动状态
7	Synchronized Motion	同步运动状态

轴的状态可以根据 MC_ReadStatus（读取轴状态指令）的输出引脚来进行判断，指令具体使用请参考“11.4.15 MC_ReadStatus（读轴状态）”。

11.3.7 BufferMode 功能介绍

对于同一轴，当有运动指令控制轴在运动过程中，可以启动其它运动指令，前后两个运动指令进行交接时，有 2 种可供选择的交接模式，交接模式可以根据后一个运动指令的 BufferMode 引脚参数设置来选择。BufferMode（交接模式）相关术语的含义如下：

1. 当前指令：当前控制轴的运动指令
2. 交接指令：等待执行的指令
3. 交接速度：当前指令切换到交接指令时的速度
4. 目标速度：指令中的 Velocity 引脚参数
5. 目标位置：位移相关指令中 Position 或 Distance 引脚参数

- **2 种交接模式**

交接模式	动作说明
0：mcAborting（打断）	立即打断当前指令的动作并执行交接指令
1：mcBuffered（等待）	等待当前指令的动作正常执行结束后，并立即执行交接指令的动作

- **注意**

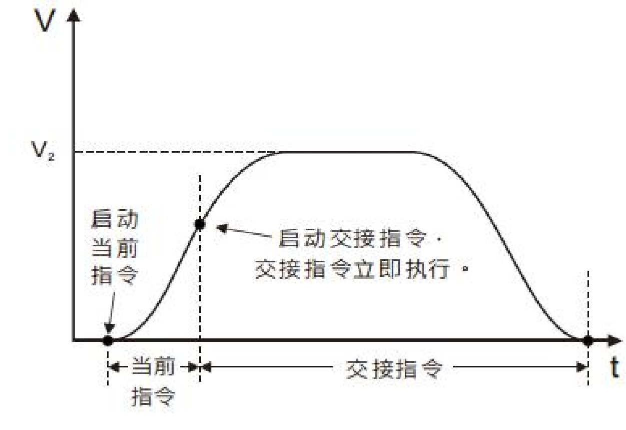
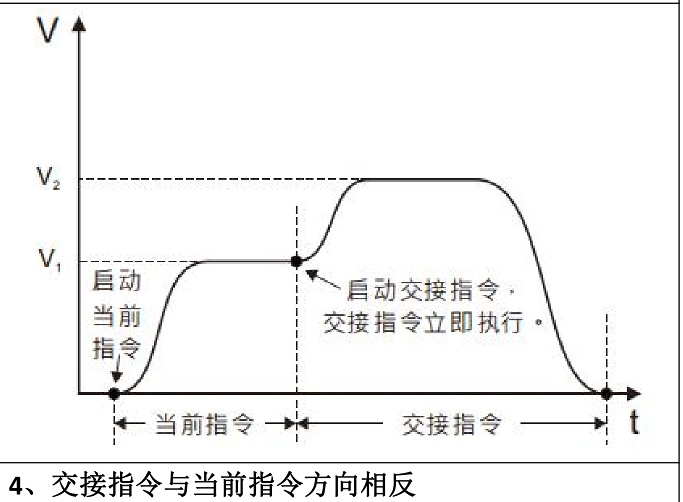
同一个轴仅支持一级 BufferMode 缓存：对于含有 BufferMode 的运动指令，若使用 BufferMode=1 的运动指令 2 交接运动指令 1 时，且运动指令 2 还未执行，此时执行 BufferMode=1 的运动指令 3 无效并报错，但不影响指令 1 和指令 2 的执行。

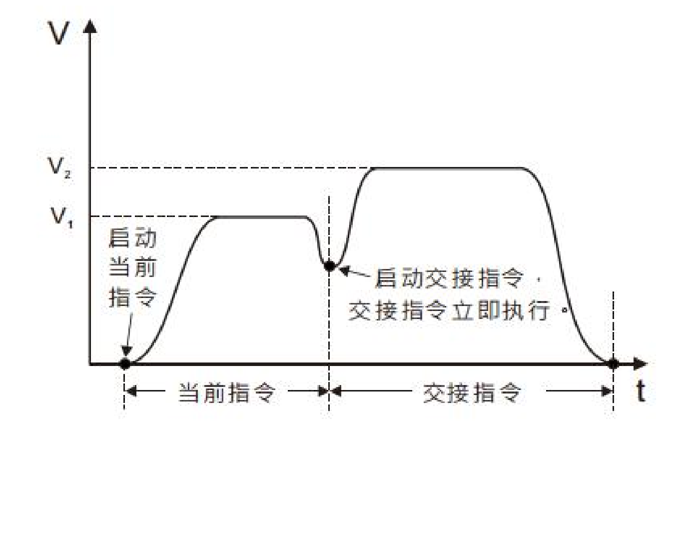
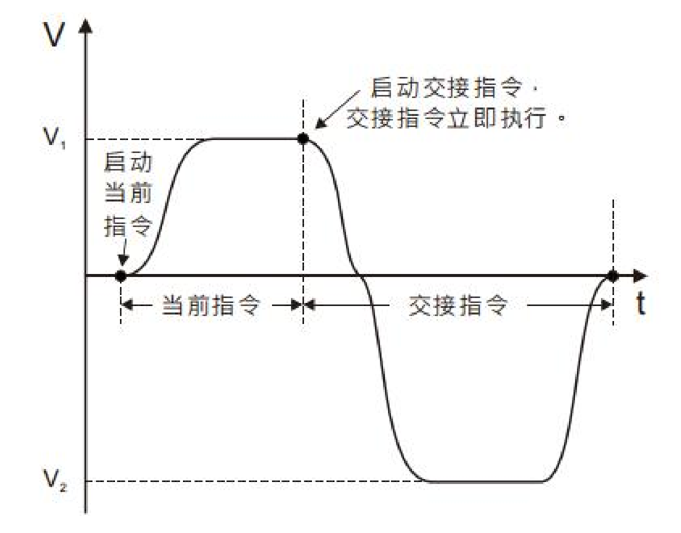
程序范例：

以两个相对位移指令对 BufferMode 进行简单说明

第一个相对位移指令速度最大是 v1，位移量为 S1，第二个相对位移指令速度最大是 v2，位移量是 S2。改变第二个位移指令的 BufferMode 使得这两个指令有不同的交接过程，如下说明：

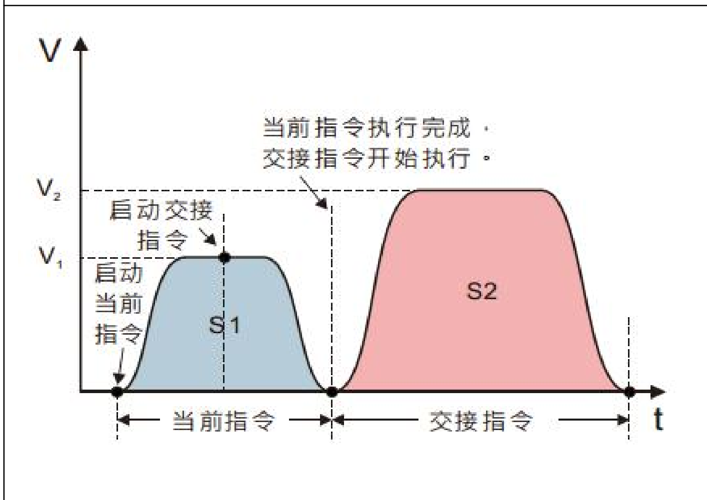
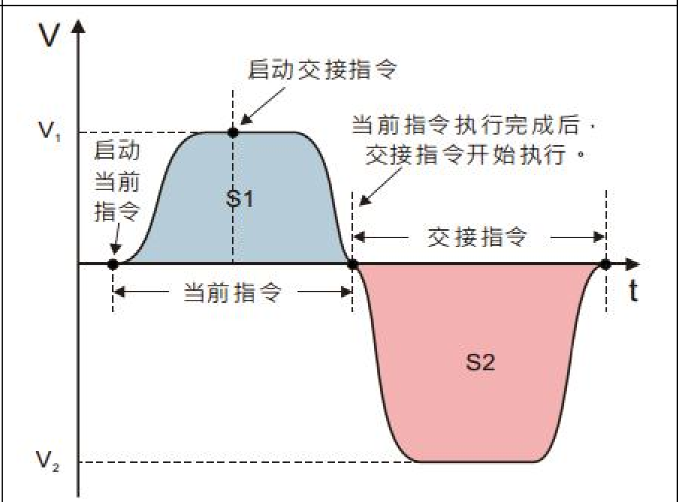
■ 当 BufferMode=0，针对以下四种情况进行说明：

<table>
<tr><td>1、当前指令加速时打断</td><td>2、当前指令运匀速阶段打断</td></tr>
<tr><td>
</td><td>
</td></tr>
<tr><td>3、当前指令减速阶段打断</td><td>4、交接指令与当前指令方向相反</td></tr>
</table>

注意：当前指令与交接指令进行交接时被控轴的加/减速度为交接指令的加/减速度

■ 当 BufferMode=1，针对以下两种情况进行说明：

1、交接指令与当前指令方向相同	2、交接指令与当前指令方向相反
	

11.4 单轴指令

注意事项：

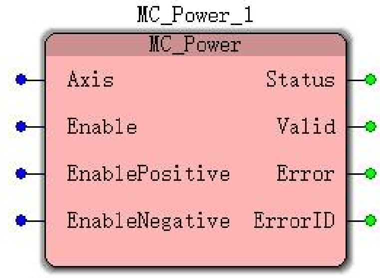
- 对于非运动指令，MC_ReadActualVelocity、MC_ReadActualPosition、MC_SetOverride、MC_ReadMotionState、MC_ReadStatus、NS_MC_ReadParameter、MC_SetPosition，在任何轴状态下均可以使用。
- 各指令在同一工程下对同一个轴的使用个数如下：

MC_AXIS_REF	整个工程每个轴最多 1 个该模块
MC_Power	整个工程每个轴最多 1 个该模块
MC_CamIn	整个工程每个轴最多 3 个该模块
MC_CamOut	整个工程每个轴最多 3 个该模块
MC_CombineAxes	整个工程每个轴最多 3 个该模块
MC_GearIn	整个工程每个轴最多 3 个该模块
MC_GearOut	整个工程每个轴最多 3 个该模块
MC_Halt	整个工程每个轴最多 3 个该模块
MC_Home	整个工程每个轴最多 3 个该模块
MC_MoveAbsolute	整个工程每个轴最多 3 个该模块
MC_MoveAdditive	整个工程每个轴最多 3 个该模块
MC_MoveRelative	整个工程每个轴最多 3 个该模块
MC_MoveVelocity	整个工程每个轴最多 3 个该模块
MC_Stop	整个工程每个轴最多 3 个该模块
NS_MC_StopByPos	整个工程每个轴最多 3 个该模块
MC_SpecialMoveAbsolute	整个工程每个轴最多 3 个该模块
NS_MC_RotaryCutIn	整个工程每个轴最多 1 个该模块
NS_MC_SpecialCamIn	整个工程每个轴最多 1 个该模块
NS_MC_SpecialCombineAxes	整个工程每个轴最多 3 个该模块
MC_HaltSuperimposed	整个工程每个轴最多 1 个该模块
MC_MoveSuperimposed	整个工程每个轴最多 1 个该模块
MC_Phasing	整个工程每个轴最多 3 个该模块
NS_MC_Jog	整个工程每个轴最多 1 个该模块
MC_SetOverride	整个工程每个轴最多 1 个该模块
MC_SetPosition	整个工程每个轴最多 1 个该模块
MC_TouchProbe	整个工程每个轴最多 1 个该模块
MC_AbortTrigger	整个工程每个轴最多 1 个该模块
NS_MC_CamReadPoint	整个工程每个轴最多 1 个该模块
NS_MC_CamReadTappetStatus	整个工程每个轴最多 3 个该模块
NS_MC_CamReadTappetValue	整个工程每个轴最多 1 个该模块
NS_MC_CamSet	整个工程每个轴最多 1 个该模块
NS_MC_CamWritePoint	整个工程每个轴最多 1 个该模块
NS_MC_CamWriteTappetValue	整个工程每个轴最多 1 个该模块

MC_ReadActualPosition	任意多个
MC_ReadActualVelocity	任意多个
MC_ReadMotionState	任意多个
MC_ReadStatus	任意多个
NS_MC_ReadParameter	任意多个

11.4.1 MC_Power（使能指令）

FB/FC	说明	适用机种
FB	此指令用于对相应的伺服轴使能或解除使能	VEC-VA-MP-005-MA



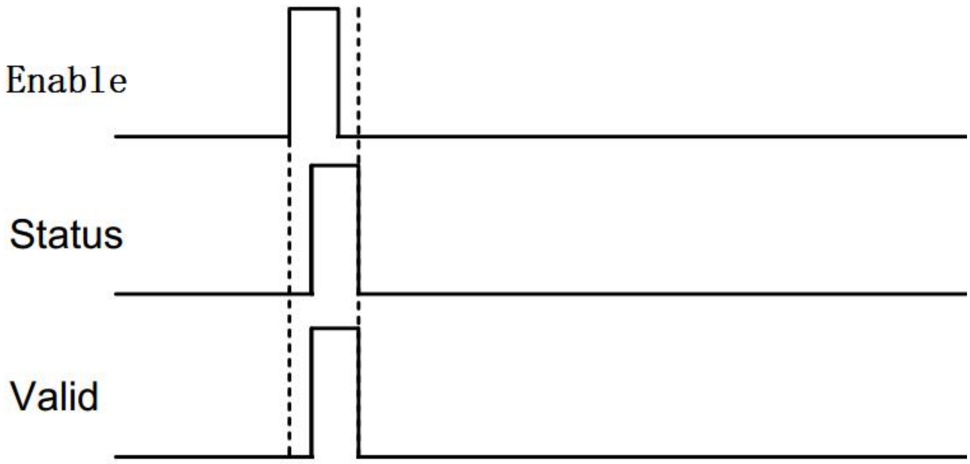
➢ **输入参数**

名称	功能	数据类型	设定范围（缺省值）	生效时机
Axis（轴号）	设定指令欲控制的轴	USINT	**模拟量/脉冲：** 0~4（实轴） 5~11（虚轴） **CANopen 模式：** 0~15 (实轴/虚轴) (0)	Enable 为 TRUE 时
Enable（执行位）	当 Enable 为 True 时，执行该指令。	BOOL	TRUE 或 FALSE(FALSE)	Enable 为 TRUE 时
EnablePositive（允许正转）	保留	BOOL	保留	保留
EnableNegative（允许反转）	保留	BOOL	保留	保留

➢ **输出参数**

名称	功能	数据类型	输出范围
Status	该参数为 TRUE 时表示指令正在控制轴	BOOL	TRUE 或 FALSE
Valid	该输出参数为 TRUE 时表示指令输出有效	BOOL	TRUE 或 FALSE
Error	该参数为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE
ErrorID	指令执行出错时的错误代码	WORD	-

➢ **输出参数时序变化图**

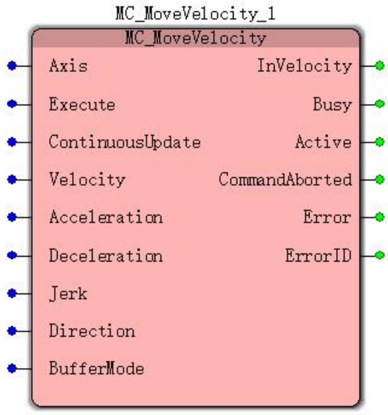


➢ **功能说明**

- 当 Enable 由 FALSE 变为 TRUE，延时一个周期后，Status、Valid 同时为 TRUE；
- 当 Enable 由 TRUE 变为 FALSE，延时一个周期后，Status、Valid 同时为 FALSE；
- 此指令用于使被控伺服轴使能或者解除使能；
- 模拟量或者脉冲模式时，该指令仅仅只是运动控制器对应控制伺服轴使能，不是伺服驱动器本身的使能， 伺服驱动器使能需根据不同伺服厂家进行设置；而 CANopen 模式下是直接使能伺服本身；
- 一个轴只能使用一个 MC_Power（使能指令）
- 单轴指令和多轴指令执行前，必须先执行 MC_Power 指令，执行顺序颠倒或者不执行
- 时，运动控制功能将无法被执行 。

11.4.2 MC_MoveVelocity(速度指令)

FB/FC	说明	适用机种
FB	此指令用于控制轴按照设定的加减速运动至设定速度并匀速运行	VEC-VA-MP-005-MA



➢ **输入参数**

名称	功能	数据类型	设定范围（缺省值）	生效时机
Axis（轴号）	设定指令欲控制的轴	USINT	**模拟量/脉冲：** 0~4（实轴） 5~11（虚轴） **CANopen 模式：** 0~15 (实轴/虚轴) (0)	Execute 由 FALSE 为 TRUE 时
Execute(执行位)	当 Execute 由 FALSE 变为 TRUE 时，指令开始执行	BOOL	TRUE 或 FALSE(FALSE)	-
ContinuousUpdata	保留	-	-	-
Velocity(速度)	设定的目标速度 (单位：单元/S)	LREAL	正数(不可缺省)	Execute 由 FALSE 为 TRUE 时
Accleration （加速度）	设定的目标加速度 (单位：单元/S^2)	LREAL	正数(不可缺省)	Execute 由 FALSE 为 TRUE 时
Decleration （减速度）	设定的目标减速度 (单位：单元/S^2)	LREAL	正数(不可缺省)	Execute 由 FALSE 为 TRUE 时
Jerk （加速度的变化率）	设定的目标加速度/减速度的变化率 (单位：单元/S^3)	LREAL	正数(不可缺省)	Execute 由 FALSE 为 TRUE 时
Direction （方向）	设定运转方向 1：正方向 3：负方向	INT	1：正方向 3：负方向 4：延续当前的方	Execute 由 FALSE 为 TRUE 时

	4：延续当前的方向		向 (不可缺省)	
BufferMode （交接模式）	设定两个指令间的交接模式 0：立即打断 1：等待	INT	0：立即打断 1：等待 (0)	Execute 由 FALSE 为 TRUE 时

说明：

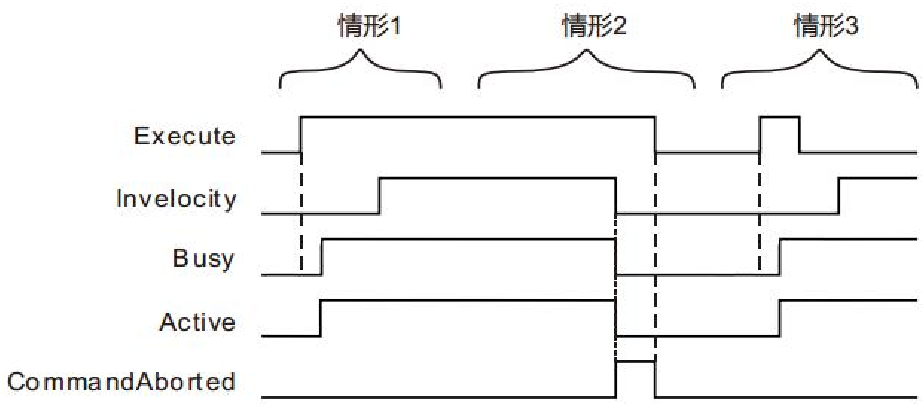
1、该指令在 Execute 由 FALSE 变为 TRUE 时开始执行。该指令正在执行，Execute 由 TRUE 变为 FALSE 时，对该指令执行无影响。

2、当指令正在执行中，Execute 再次由 FALSE 变为 TRUE 时，该指令可以重新执行，此时能够重新生效的引脚参数包括 Velocity、Acceleration、Deceleration、Jerk、Direction、BufferMode。

➢ **输出参数**

名称	功能	数据类型	输出范围
Invelocity（速度到达）	该参数输出为 TRUE 时表示速度到达	BOOL	TRUE 或 FALSE
Busy（执行中）	该参数输出为 TRUE 时表示指令正在执行中	BOOL	TRUE 或 FALSE
Active（控制中）	该参数输出为 TRUE 时表示指令正在控制轴	BOOL	TRUE 或 FALSE
CommandAborted（中断）	该参数输出为 TRUE 时表示指令被中断	BOOL	TRUE 或 FALSE
Error（错误）	该指令输出为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE
ErrorID（错误代码）	执行指令出错时的错误代码	WORD	-

➢ **输出参数时序变化图**



情形 1：当 Execute 由 FALSE 变为 TRUE 时，一个周期后，Busy、Active 变为 TRUE。当速度到达时，Invelocity 变为 TRUE，同时 Busy 和 Active 依然保持为 TRUE 状态。

情形 2：当 Execut 为 TRUE 时，该指令被其它指令中断时，CommandAborted 变为 TRUE，同时 Invelocity、Busy 和 Active 变为 FALSE，当 Execute 由 TRUE 变为 FALSE 时，CommandAborted 变为 FALSE。

情形 3：在指令执行过程中，当 Execute 由 TRUE 变为 FALSE 后，速度到达后，Invelocity 变为 TRUE，同时 Busy 和 Active 依然保持为 TRUE 状态。

➢ **功能说明**

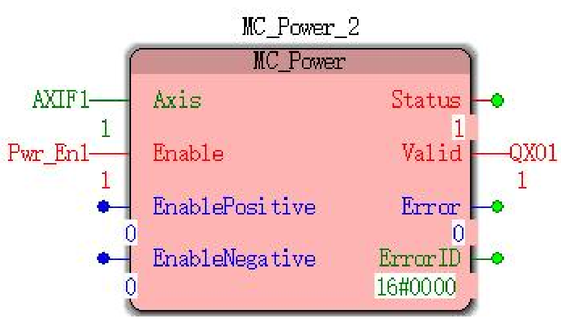
- 该指令在 Execute 由 FALSE 变为 TRUE 时开始执行。若该指令无交接指令，无论该指令是否执行完成，Execute 再次由 FALSE 变为 TRUE 时，该指令可以重新执行，此时能够重新生效的引脚参数包括 Velocity、Acceleration、Deceleration、Jerk、Direction、BufferMode;
- 修改被控轴的 Velocity 速度值时，需要重新触发 Execute，速度才能改变;
- 当该指令执行完成后，即 Invelocity 由 FALSE 变为 TRUE 后，即使通过 MC_SetOverride 指令改变目标速度，此时 Invelocity 依然保持为 TRUE。当 MC_MoveVelocity 未完成时，即 Invelocity 为 FALSE 时，通过 MC_SetOverride 指令改变目标速度，当到达新的目标速度后，Invelocity 才会由 FALSE 变为 TRUE。

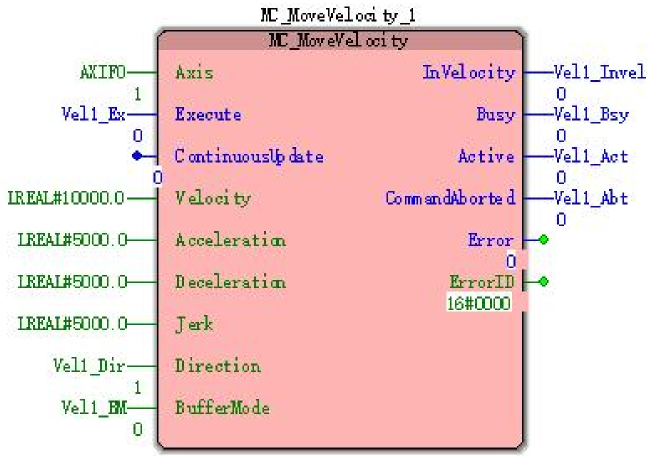
程序范例一

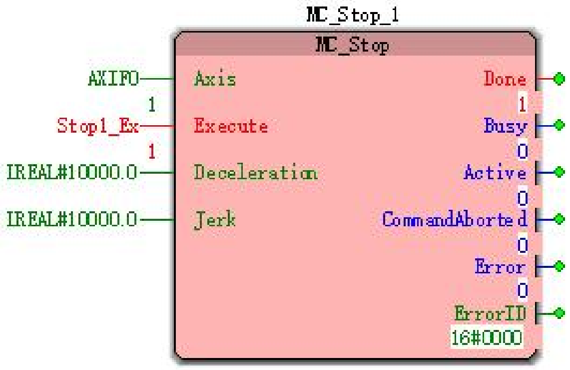
MC_MoveVelocity 指令单独执行时的范例如下所示。

1、变量和程序

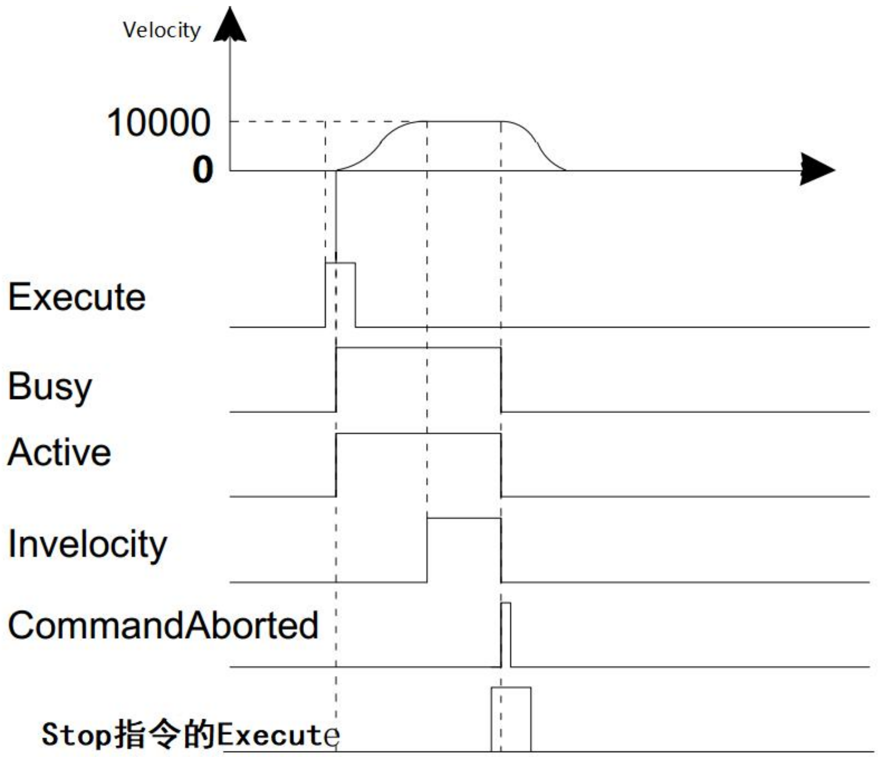
变量名	数据类型	初始值
MC_MoveVelocity1	MC_MoveVelocity	
AXIF0	USINT	1
Vel1_Ex	BOOL	FALSE
Vel1_Dir	INT	1
Vel1_BM	INT	0
Vel1_Invel	BOOL	
Vel1_Bsy	BOOL	
Vel1_Act	BOOL	
Vel_Abt	BOOL	
MC_Stop1	MC_Stop	
Stop_Ex	BOOL	FALSE







2、运动曲线和时序图



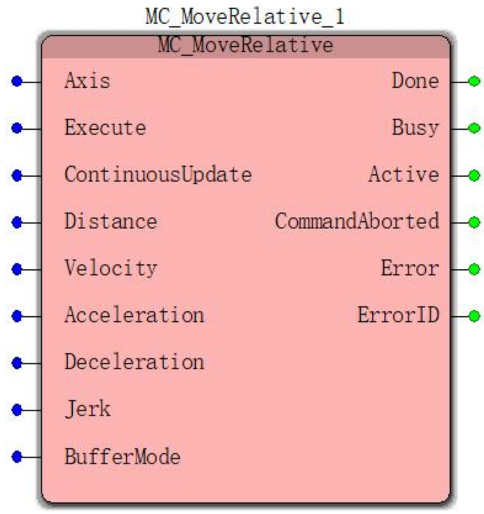
- 当 Vel1_Ex 由 FALSE 变为 TRUE 时，Vel1_Bsy、Vel_Act 同时变为 TRUE，开始执行速度指

令；当速度到达时，Vel1_Invel 变为 TRUE，同时 Vel1_Bsy 和 Vel1_Act 依然保持为 TRUE。

- 当 Stop_Ex1 由 FALSE 变为 TRUE 时，一个周期后，CommandAborted 由 FALSE 变为 TRUE(若 Vel1_Ex 此时为 FALSE,则在一个周期后 CommandAborted 变为 FALSE)，同时，Busy 和 Active 变为 FALSE。

11.4.3 MC_MoveRelative（相对位移指令）

FB/FC	说明	适用机种
FB	此指令用于控制轴以当前位置为起点，按设定的速度，加减速，加速度的变化率移动设定的距离	VEC-VA-MP-005-MA



➢ **输入参数**

名称	功能	数据类型	设定范围（缺省值）	生效时机
Axis（轴号）	设定指令欲控制的轴	USINT	**模拟量/脉冲：** 0~4（实轴） 5~11（虚轴） **CANopen 模式：** 0~15 (实轴/虚轴) (0)	Exexcute 由 FALSE 变为 TRUE
Execute	当 Exexcute 由 FALSE 变为 TRUE，指令开始执行。	BOOL	TRUE 或 FALSE (FALSE)	-
ContinuousUpdata	保留	-	-	-
Distance （距离）	设定的目标距离 （单位：单元）	LREAL	正数、负数、零 (0)	Exexcute 由 FALSE 变为 TRUE
Velocity （速度）	设定的目标速度 （单位：单元/S）	LREAL	正数(不可缺省)	Exexcute 由 FALSE 变为 TRUE
Acceleration （加速度）	设定的目标加速度 (单位：单元/S^2)	LREAL	正数(不可缺省)	Exexcute 由 FALSE 变为 TRUE
Deceleration （减速度）	设定的目标减速度 (单位：单元/S^2)	LREAL	正数(不可缺省)	Exexcute 由 FALSE 变为 TRUE
Jerk （加速度的	设定的目标加速度或减速度的变化率	LREAL	正数、零 (0)	Exexcute 由 FALSE 变为 TRUE

变化率）	(单位：单元/S³)			
BufferMode（交接模式）	设定两个指令之间交接模式 0:立即打断 1：等待	INT	0:立即打断 1：等待 (0)	Exexcute 由 FALSE 变为 TRUE

说明：

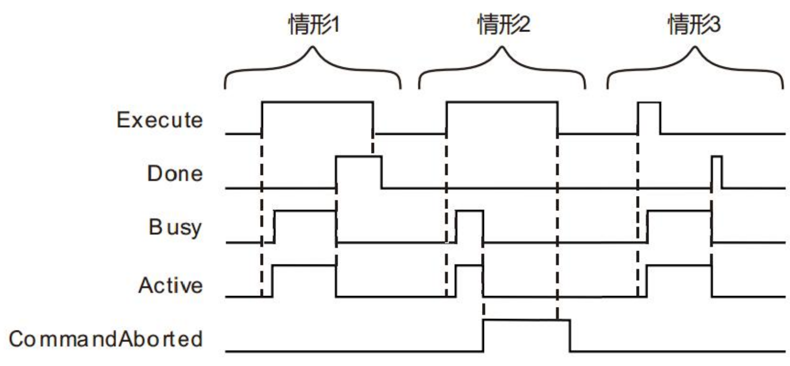
1、该指令在 Execute 由 FALSE 变为 TRUE 时开始执行。该指令正在执行，Execute 由 TRUE 变为 FALSE 时，对该指令执行无影响。

2、当指令正在执行中，Execute 再次由 FALSE 变为 TRUE 时，该指令可以重新执行，此时能够重新生效的引脚参数包括 Distance、Velocity、Acceleration、Deceleration、Jerk、BufferMode。

➢ **输出参数**

名称	功能	数据类型	输出范围
Done	该参数输出为 TRUE 时表示指令执行完成	BOOL	TRUE 或 FALSE
Busy（执行中）	该参数输出为 TRUE 时表示指令正在执行中	BOOL	TRUE 或 FALSE
Active（控制中）	该参数输出为 TRUE 时表示指令正在控制轴	BOOL	TRUE 或 FALSE
CommandAborted（中断）	该参数输出为 TRUE 时表示指令被中断	BOOL	TRUE 或 FALSE
Error（错误）	该指令输出为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE
ErrorID（错误代码）	执行指令出错时的错误代码	WORD	-

➢ **输出参数时序变化图**



情形 1： 当 Execute 由 FALSE 变为 TRUE 时，Busy 和 Active 同时变为 TRUE。当定位完成时，Done 变为 TRUE，同时 Busy 和 Active 变为 FALSE，若在定位完成后，Execute 由 TRUE 变为 FALSE，一个周期后，Done 变为 FALSE。

情形 2： 当 Execute 为 TRUE 时，该指令被其它指令中断后，CommandAborted 变为 TRUE，同时 Busy 和 Active 变为 FALSE；当 Execute 由 TRUE 变为 FALSE 时，一个周期后 CommandAborted 变为 FALSE。

情形 3：在指令执行过程中，Execute 由 TRUE 变为 FALSE 后，指令执行完成时，Done 变为 TRUE，同时 Busy 和 Active 变为 FALSE，且一个周期后，Done 变为 FALSE。

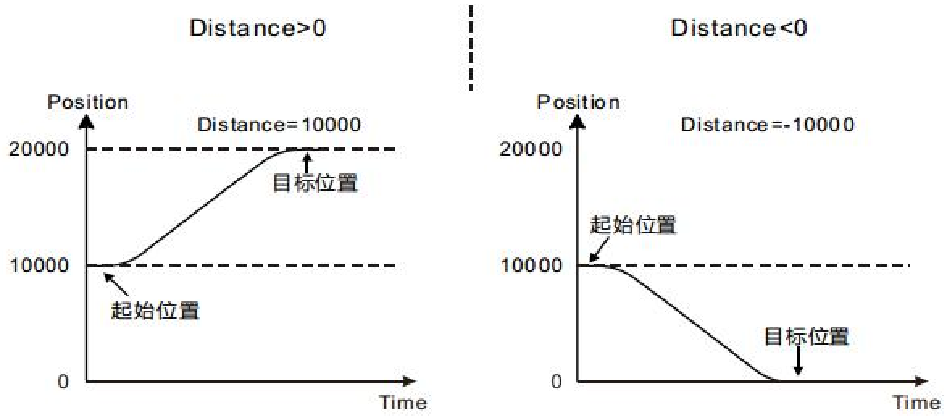
➢ **功能说明**

● 此指令用于控制轴按照设定速度，加减速以及加速度的变化率移动设定的距离，该距离的参考起点为指令开始执行时的轴位置。

● Distance 与参考起点共同决定轴在该指令控制下移动的目标位置，即目标位置= 参考起点位置+Distance。

● 当 Distance 为 0 时，轴运动的目标位置为当前位置，指令在开始执行的下一周期即执行完成，Done 变为 TRUE。

如下图所示，参考起点的位置为 10000，当 Distance>0（10000）时，轴将正向运动，目标位置为 20000（10000+10000），如左下图；当 Distance<0（-10000）时，轴将反转，目标位置为 0（10000-10000），如右下图。

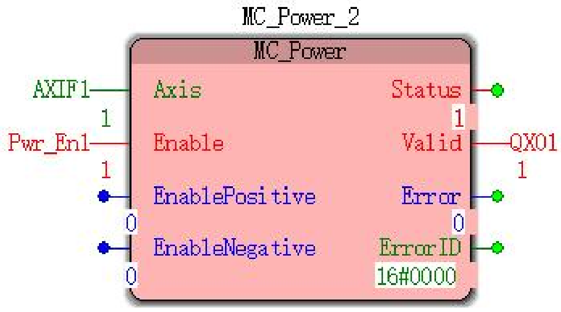


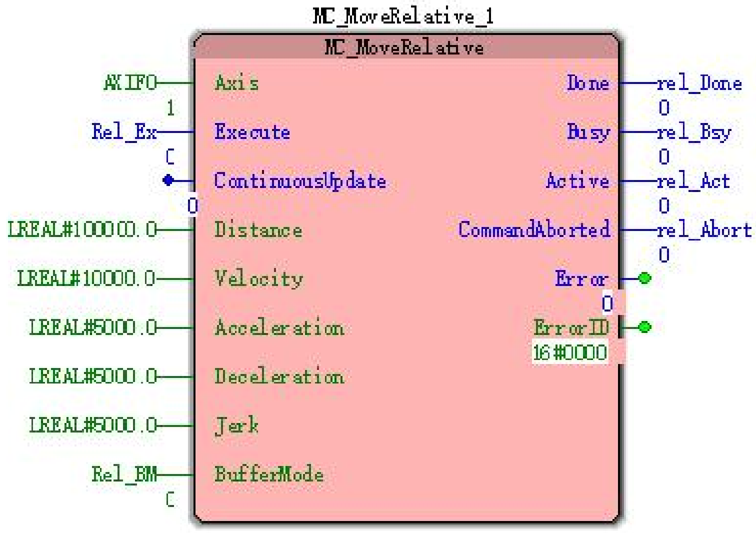
程序范例一

MC_MoveRelative 指令单独执行时的范例如下所示。

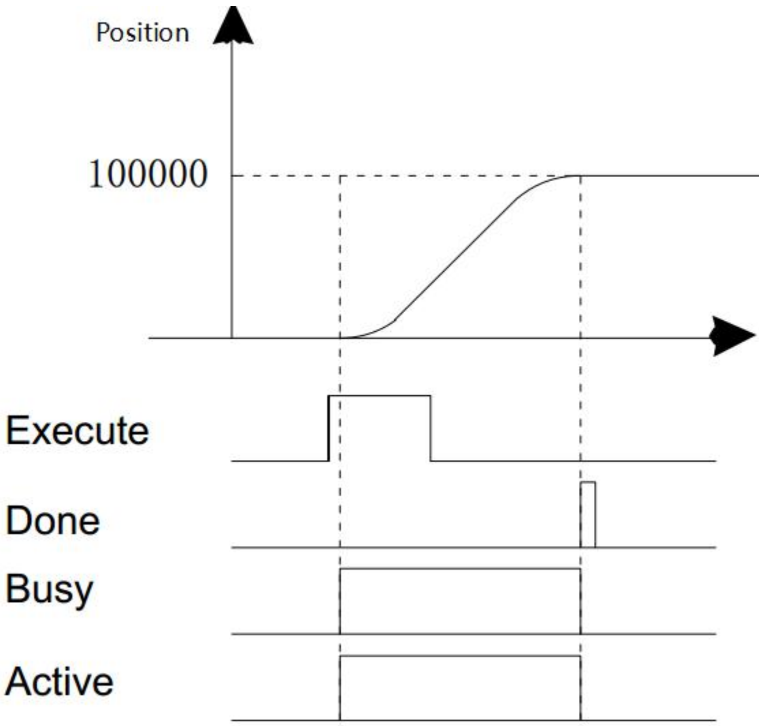
1、变量和程序

变量名	数据类型	初始值
MC_MoveRelativey1	MC_MoveRelative	
AXIF0	USINT	1
Rel_Ex	BOOL	FALSE
Rel_BM	BOOL	0
Rel_Done	BOOL	
Rel_Bsy	BOOL	
Rel_Act	BOOL	
Rel_Abort	BOOL	





2、运动曲线和时序图



- 当 Rel_Ex 由 FALSE 变为 TRUE，一个周期后，Busy 和 Active 同时为 TRUE，开始按照设定的参数执行相对位移指令，此时轴的当前位置为 0，目标位置为 100000。
- 当轴位置到达 100000，指令执行完成，输出 Done 为 TRUE。

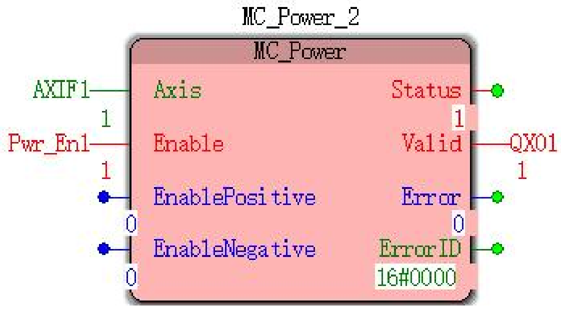
程序范例二

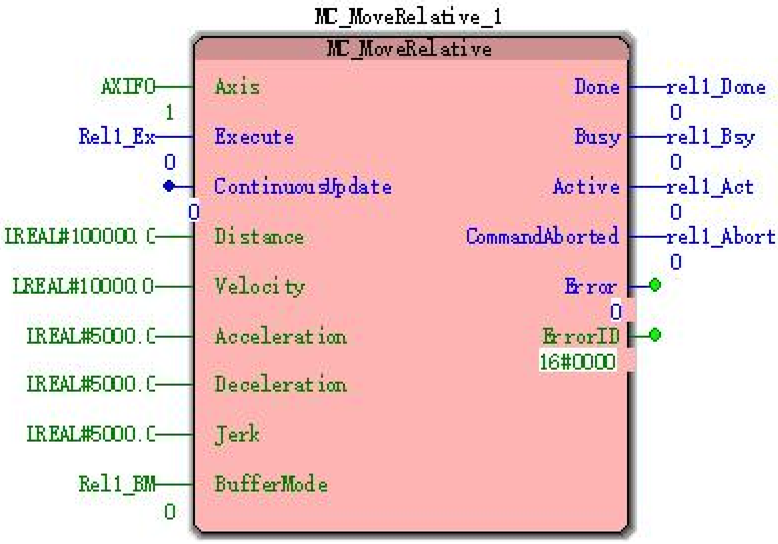
MC_MoveRelative 打断 MC_MoveRelative 的范例如下所示。

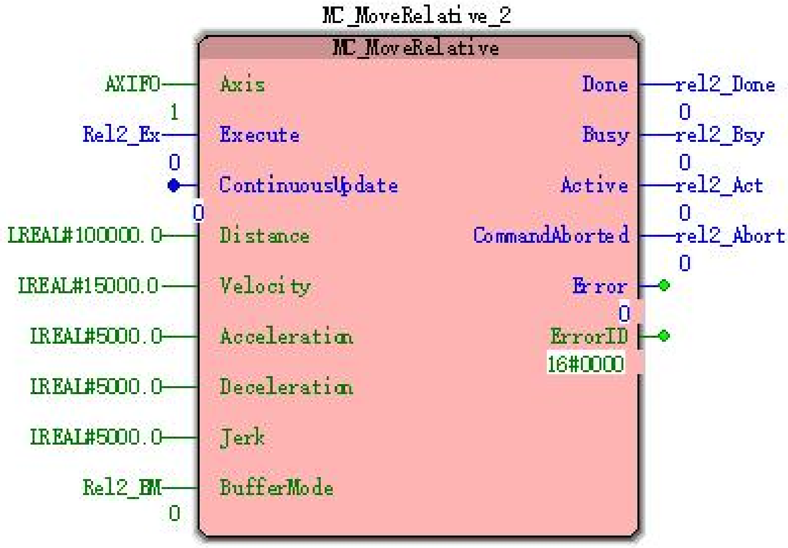
1、变量和程序

变量名	数据类型	初始值
MC_MoveRelativey1	MC_MoveRelative	
AXIF0	USINT	1

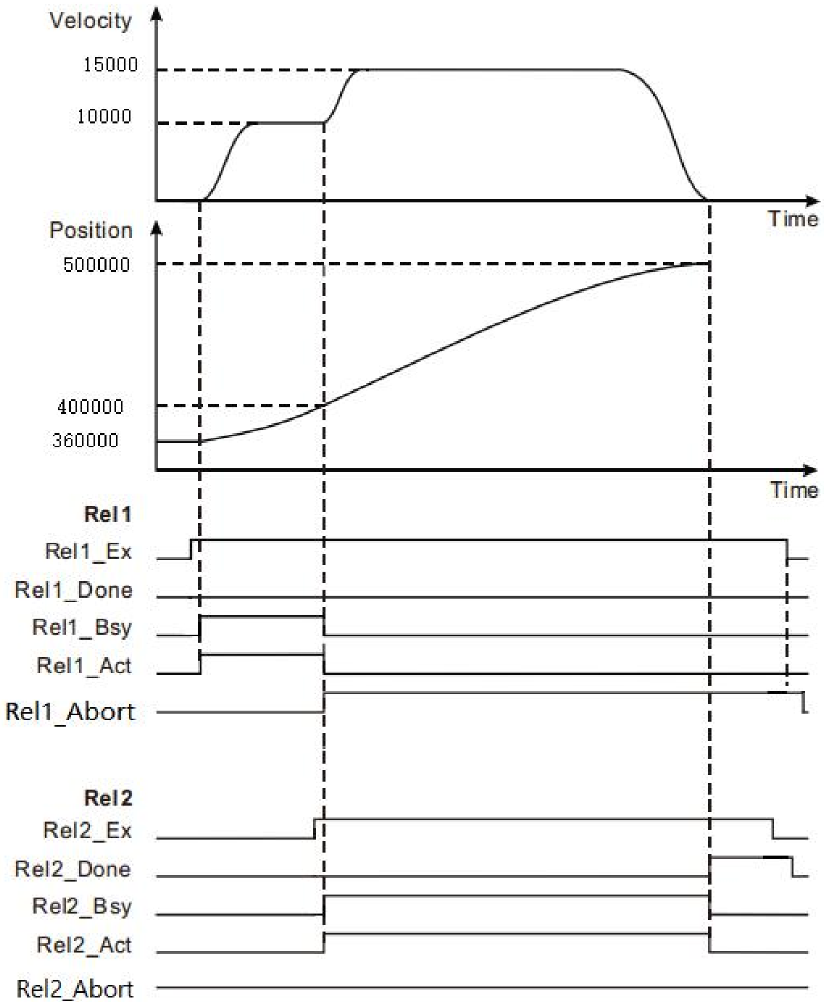
Rel_Ex	BOOL	FALSE
Rel_BM	BOOL	0
Rel_Done	BOOL	
Rel_Bsy	BOOL	
Rel_Act	BOOL	
Rel_Abort	BOOL	
MC_MoveRelativey2	MC_MoveRelative	
AXIF0	USINT	1
Rel2_Ex	BOOL	FALSE
Rel2_BM	BOOL	0
Rel2_Done	BOOL	
Rel2_Bsy	BOOL	
Rel2_Act	BOOL	
Rel2_Abort	BOOL	







2、运动曲线和时序图

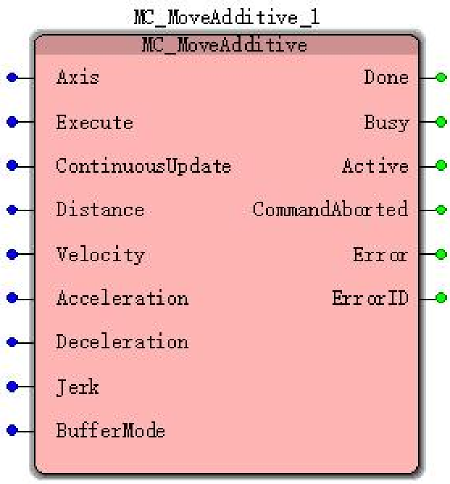


- 当 Rel1_Ex 由 FALSE 变为 TRUE 时， 第一个 MC_MoveRelative 指令开始执行，此时轴的当前位置为 360000，目标位置为（46000=36000+10000）。

- 当轴位置到达 40000 时，Rel2_Ex 由 FALSE 变为 TRUE，第二个 MC_MoveRelative 指令开始执行，且第一个 MC_MoveRelative 指令的执行被中断，其输出参数 Rel1_Abt 变为 TRUE。
- 当轴位置到达 50000（50000=40000+10000）时，第二个 MC_MoveRelative 执行完成，其输出参数 Rel2_Done 变为 TRUE。

11.4.4 MC_MoveAdditive（附加位移指令）

FB/FC	说明	适用机种
FB	此指令用于控制轴按照设定速度，加减速移动一段附加的距离	VEC-VA-MP-005-MA



➢ **输入参数**

名称	功能	数据类型	设定范围（缺省值）	生效时机
Axis（轴号）	设定指令欲控制的轴	USINT	**模拟量/脉冲：** 0~4（实轴） 5~11（虚轴） **CANopen 模式：** 0~15 (实轴/虚轴) (0)	Exexcute 由 FALSE 变为 TRUE
Execute	当 Exexcute 由 FALSE 变为 TRUE，指令开始执行。	BOOL	TRUE 或 FALSE (FALSE)	-
ContinuousUpdata	保留	-	-	-
Distance （距离）	设定的目标距离 （单位：单元）	LREAL	正数、负数、零 (0)	Exexcute 由 FALSE 变为 TRUE
Velocity （速度）	设定的目标速度 （单位：单元/S）	LREAL	正数(不可缺省)	Exexcute 由 FALSE 变为 TRUE
Acceleration （加速度）	设定的目标加速度 (单位：单元/S^2)	LREAL	正数(不可缺省)	Exexcute 由 FALSE 变为 TRUE
Deceleration （减速度）	设定的目标减速度 (单位：单元/S^2)	LREAL	正数(不可缺省)	Exexcute 由 FALSE 变为 TRUE
Jerk （加速度的	设定的目标加速度或减速度的变化率	LREAL	正数、零 (0)	Exexcute 由 FALSE 变为 TRUE

变化率）	(单位：单元/S³)			
BufferMode（交接模式）	设定两个指令之间交接模式 0:立即打断 1：等待	INT	0:立即打断 1：等待 (0)	Exexcute 由 FALSE 变为 TRUE

说明：

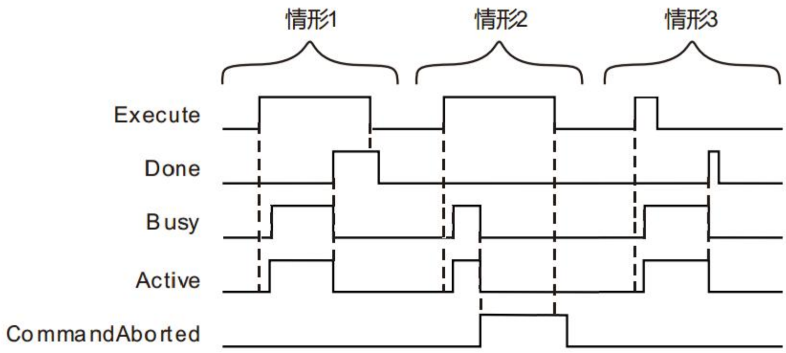
1、该指令在 Execute 由 FALSE 变为 TRUE 时开始执行。该指令正在执行，Execute 由 TRUE 变为 FALSE 时，对该指令执行无影响。

2、当指令正在执行中，Execute 再次由 FALSE 变为 TRUE 时，该指令可以重新执行，此时能够重新生效的引脚参数包括 Distance、Velocity、Acceleration、Deceleration、Jerk、BufferMode。

➢ **输出参数**

名称	功能	数据类型	输出范围
Done	该参数输出为 TRUE 时表示指令执行完成	BOOL	TRUE 或 FALSE
Busy（执行中）	该参数输出为 TRUE 时表示指令正在执行中	BOOL	TRUE 或 FALSE
Active（控制中）	该参数输出为 TRUE 时表示指令正在控制轴	BOOL	TRUE 或 FALSE
CommandAborted（中断）	该参数输出为 TRUE 时表示指令被中断	BOOL	TRUE 或 FALSE
Error（错误）	该指令输出为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE
ErrorID（错误代码）	执行指令出错时的错误代码	WORD	-

➢ **输出参数时序变化图**



情形 1： 当 Execute 由 FALSE 变为 TRUE 时，一个周期后 Buys 和 Active 同时变为 TRUE；当定位完成时，Done 变为 TRUE，同时 Busy 和 Active 变为 FALSE，在 Execute 由 TRUE 变为 FALSE 一个周期后，Done 变为 FALSE。

情形 2： 当 Execute 为 TRUE 时，该指令被其它指令中断后，CommandAborted 变为 TRUE，同时 Busy 和 Active 变为 FALSE；当 Execute 由 TRUE 变为 FALSE 时，一个周期后 CommandAborted 变为 FALSE。

情形 3： 在指令执行过程中，Execute 由 TRUE 变为 FALSE 后， 指令执行完成时，Done 变为 TRUE，同时 Busy 和 Active 变为 FALSE，且一个周期后，Done 变为 FALSE。

➢ **功能说明**

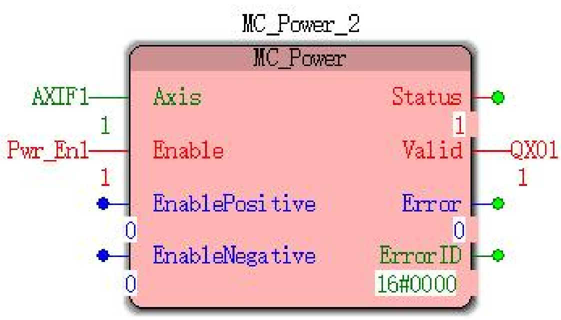
- 此指令用于控制终端执行机构按设定的速度，加速度移动一段附加的距离。
- 当前一个指令为位移相关指令时，如未完成，则此指令执行时终端执行机构移动的距离为前一个指令剩余的距离和此指令设定的距离总和，当此指令执行完成时，终端执行机构的最终位置为前一个位移指令和此指令设定的距离总和；当前一个指令为速度指令时，此指令执行时会终止速度指令执行，并按设定的速度，加减速移动设定的距离后停止
- 该指令单独执行时，其作用与 MC_MoveRelative 相同

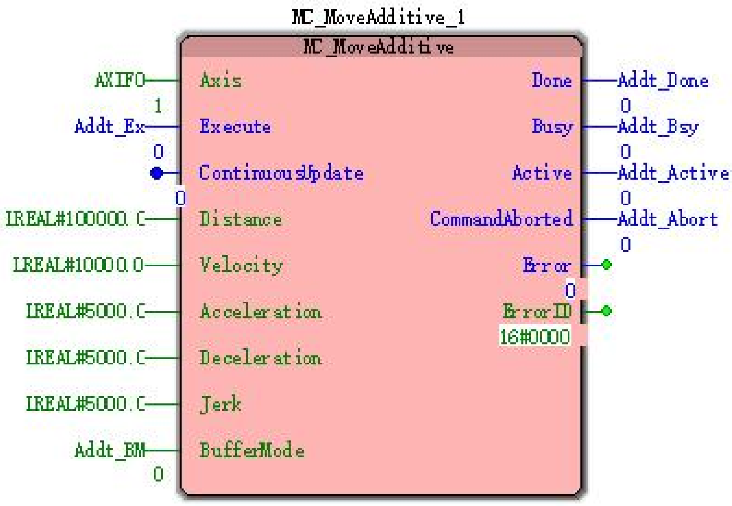
程序范例一

MC_MoveAdditive 指令单独执行时的范例如下所示

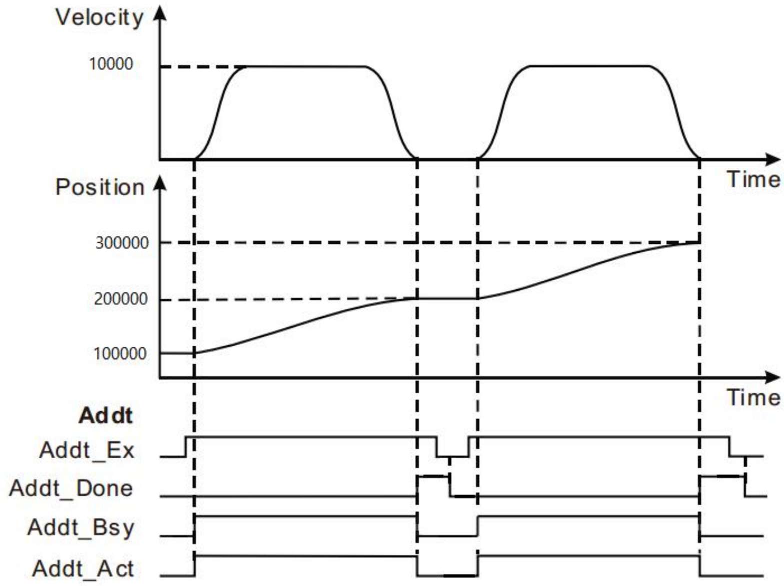
1、变量和程序

变量名	数据类型	初始值
MC_MoveAdditive1	MC_MoveAdditive	
AXIF0	USINT	1
Addt_Ex	BOOL	FALSE
Addt_BM	INT	0
Addt_Done	BOOL	
Addt_Bsy	BOOL	
Addt_Active	BOOL	
Addt_Abort	BOOL	





2、运动曲线和时序图



- 当 Addt_Ex 由 FALSE 变为 TRUE 时，运动控制器控制伺服电机以当前位置为参考点运转，一个周期后 Addt_Bsy、Addt_Act 变为 TRUE。当伺服电机完成设定距离后，完成位 Addt_Done 由 FALSE 变为 TRUE，同时 Addt_Bsy 和 Addt_Act 由 TRUE 变为 FALSE。
- 当 Addt_Ex 由 TRUE 变为 FALSE 时，一个周期后完成位 Addt_Done 复位。
- 伺服电机完成设定距离后，Addt_Ex 再次由 FALSE 变为 TRUE 时，运动控制器控制伺服电机运转，当伺服电机完成设定距离后，完成位 Addt_Done 再次由 FALSE 变为 TRUE。

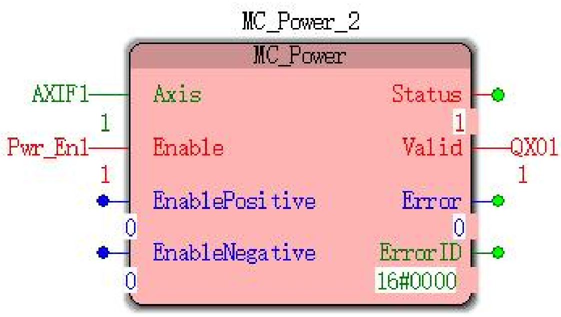
程序范例二

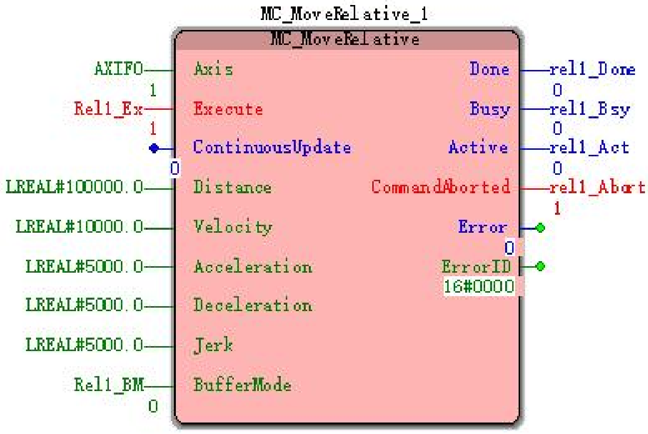
MC_MoveAdditive 指令打断 MC_MoveRelative 的范例如下所示

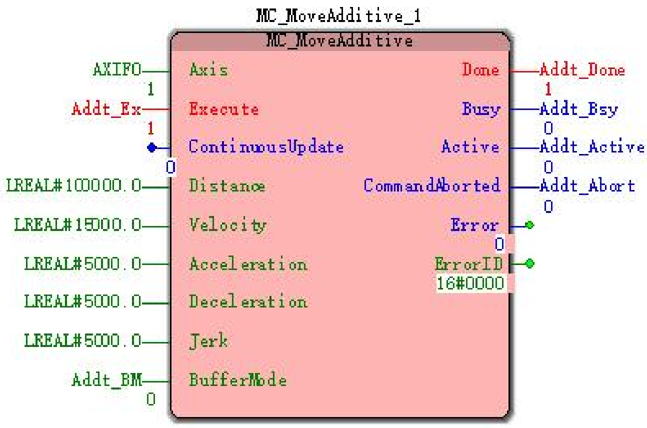
1、变量和程序

名变量	数据类型	初始值
MC_MoveRelative_1	MC_MoveRelative	
AXIF0	USINT	1

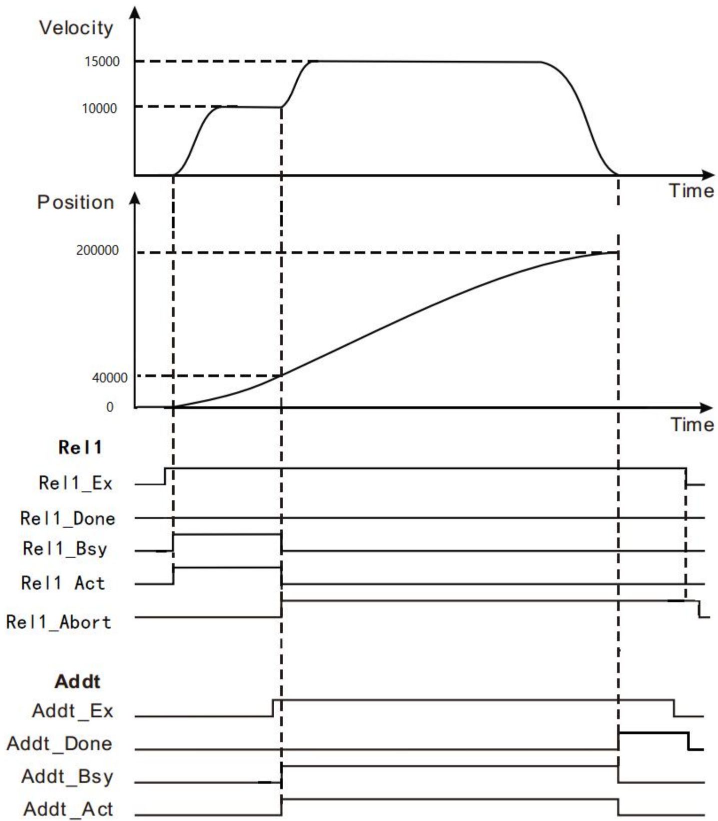
Rel1_Ex	BOOL	FALSE
Rel1_BM	INT	0
Rel1_Done	BOOL	
Rel1_Bsy	BOOL	
Rel1_Act	BOOL	
Rel1_Abort	BOOL	
MC_MoveAdditive1	MC_MoveAdditive	
Addt_Ex	BOOL	FALSE
Addt_BM	INT	0
Addt_Done	BOOL	
Addt_Bsy	BOOL	
Addt_Active	BOOL	
Addt_Abort	BOOL	







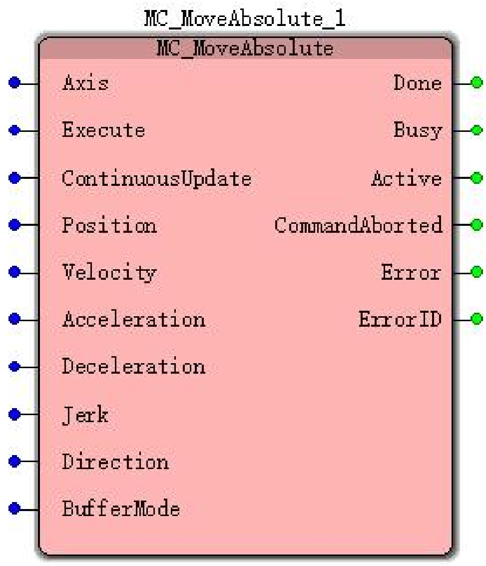
2、运动曲线和时序图



- 当 Rel1_Ex 由 FALSE 变为 TRUE 时，运动控制器控制伺服电机以当前位置为参考点运转，在轴位置 Position=40000 处，执行附加位移指令，Addt_Ex 由 FALSE 变为 TRUE，一个周期后，中断位 Rel1_Abt 由 FALSE 变为 TRUE。同时，伺服电机以第二个附加位置移动指令的参数进行运动。当伺服电机到达设定距离时（此时距离为两个指令设定距离总和），完成位 Addt_Done 由 FALSE 变为 TRUE。
- 当 Addt_Ex 由 TRUE 变为 FALSE 时，一个周期后完成位 Addt_Done 复位。

11.4.5 MC_MoveAbsolute（绝对位移指令）

FB/FC	说明	适用机种
FB	此指令用于控制轴按照设定速度，加减速移动到相对于零点的目标位置	VEC-VA-MP-005-MA



➢ **输入参数**

名称	功能	数据类型	设定范围（缺省值）	生效时机
Axis（轴号）	设定指令欲控制的轴	USINT	**模拟量/脉冲：** 0~4（实轴） 5~11（虚轴） **CANopen 模式：** 0~15 (实轴/虚轴) (0)	Exexcute 由 FALSE 变为 TRUE
Execute	当 Exexcute 由 FALSE 变为 TRUE，指令开始执行。	BOOL	TRUE 或 FALSE (FALSE)	-
ContinuousUpdata	保留	-	-	-
Position（位置）	设定的目标位置 旋转轴：0≤Position＜模 直线轴：无限制 （单位：单元）	LREAL	正数、负数、零 (0)	Exexcute 由 FALSE 变为 TRUE
Velocity（速度）	设定的目标速度 （单位：单元/S）	LREAL	正数(不可缺省)	Exexcute 由 FALSE 变为 TRUE
Acceleration（加速度）	设定的目标加速度 (单位：单元/S^2)	LREAL	正数(不可缺省)	Exexcute 由 FALSE 变为 TRUE
Deceleration	设定的目标减速度	LREAL	正数(不可缺省)	Exexcute 由 FALSE 变

（减速度）	(单位：单元/S^2)			为 TRUE
Jerk （加速度的变化率）	设定的目标加速度或减速度的变化率 (单位：单元/S^3)	LREAL	正数、零 （0）	Exexcute 由 FALSE 变为 TRUE
Direction （方向）	设定的运转方向（该参数仅在旋转轴是有效） 1：正向 2：最短距离 3：反向 4：当前方向	INT	1：正向 2：最短距离 3：反向 4：当前方向 (不可缺省)	Exexcute 由 FALSE 变为 TRUE 且轴为旋转轴模式
BufferMode （交接模式）	设定两个指令之间交接模式 0:立即打断 1：等待	INT	0:立即打断 1：等待 (0)	Exexcute 由 FALSE 变为 TRUE

说明：

1. 该指令在 Execute 由 FALSE 变为 TRUE 时开始执行。该指令正在执行，Execute 由 TRUE 变为 FALSE 时，对该指令执行无影响。

2、当指令正在执行中，Execute 再次由 FALSE 变为 TRUE 时，该指令可以重新执行，此时能够重新生效的引脚参数包括 Distance、Velocity、Acceleration、Deceleration、Jerk、Direction、BufferMode。

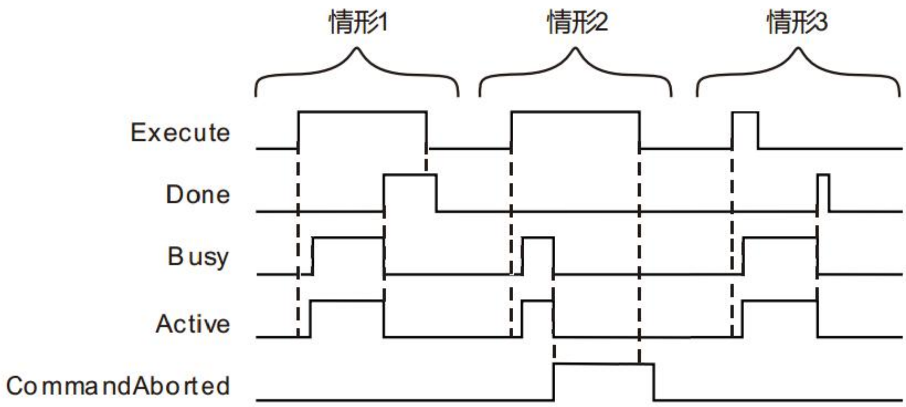
3. 当轴为旋转轴时，参数 Position 可以为 0 到模以内但不包括模的值，如果参数 Position 的绝对值大于或等于模，执行此指令会报错；当轴为直线轴时，参数 Position 的大小和模无关，可以设为任意一个常数。

4. 参数 Direction 仅在轴为旋转轴时有效，关于该参数的详细描述请参考该指令功能描述中 Direction 部分 。

➢ **输出参数**

名称	功能	数据类型	输出范围
Done	该参数输出为 TRUE 时表示指令执行完成	BOOL	TRUE 或 FALSE
Busy（执行中）	该参数输出为 TRUE 时表示指令正在执行中	BOOL	TRUE 或 FALSE
Active（控制中）	该参数输出为 TRUE 时表示指令正在控制轴	BOOL	TRUE 或 FALSE
CommandAborted（中断）	该参数输出为 TRUE 时表示指令被中断	BOOL	TRUE 或 FALSE
Error（错误）	该指令输出为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE
ErrorID（错误代码）	执行指令出错时的错误代码	WORD	-

➢ **输出参数时序变化图**



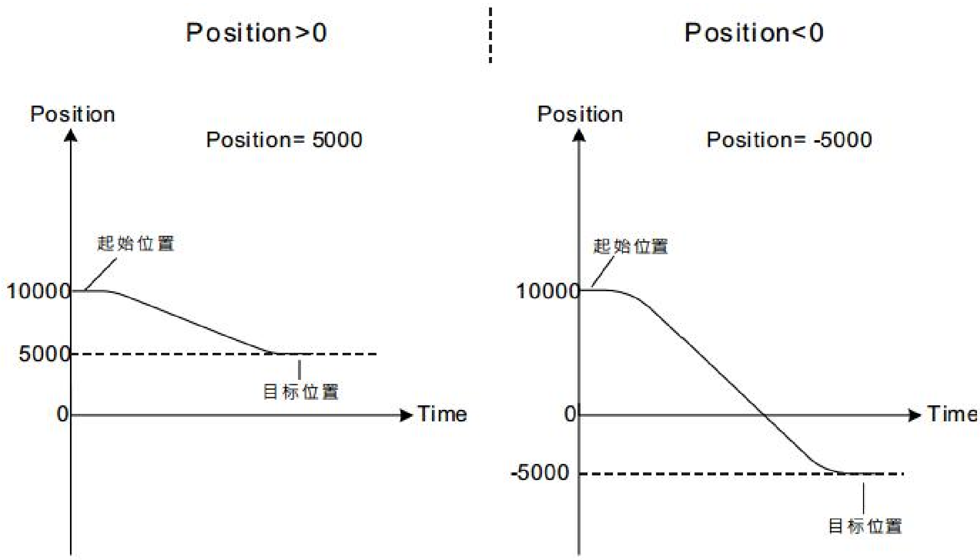
情形 1： 当 Execute 由 FALSE 变为 TRUE 时，一个周期后 Buys 和 Active 同时变为 TRUE；当定位完成时，Done 变为 TRUE，同时 Busy 和 Active 变为 FALSE，在 Execute 由 TRUE 变为 FALSE 一个周期后，Done 变为 FALSE。

情形 2： 当 Execute 为 TRUE 时，该指令被其它指令中断后，CommandAborted 变为 TRUE，同时 Busy 和 Active 变为 FALSE；当 Execute 由 TRUE 变为 FALSE 时，一个周期后 CommandAborted 变为 FALSE。

情形 3： 在指令执行过程中，Execute 由 TRUE 变为 FALSE 后， 指令执行完成时，Done 变为 TRUE，同时 Busy 和 Active 变为 FALSE，且一个周期后，Done 变为 FALSE。

➢ **功能说明**

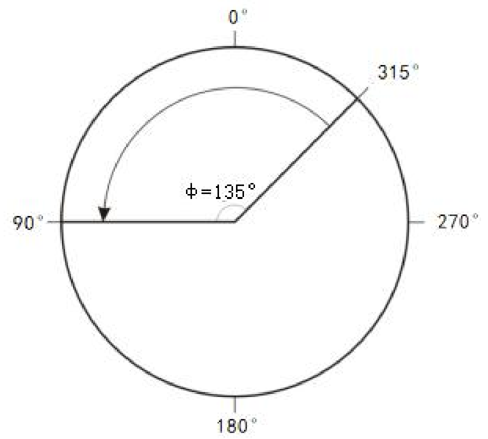
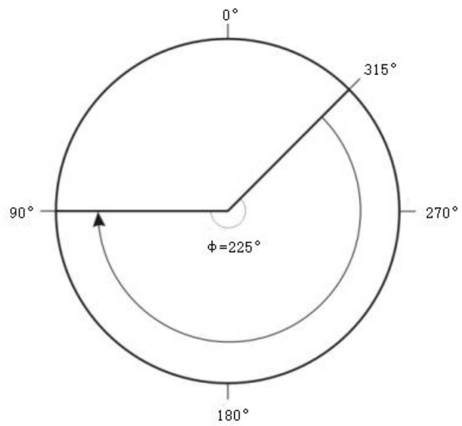
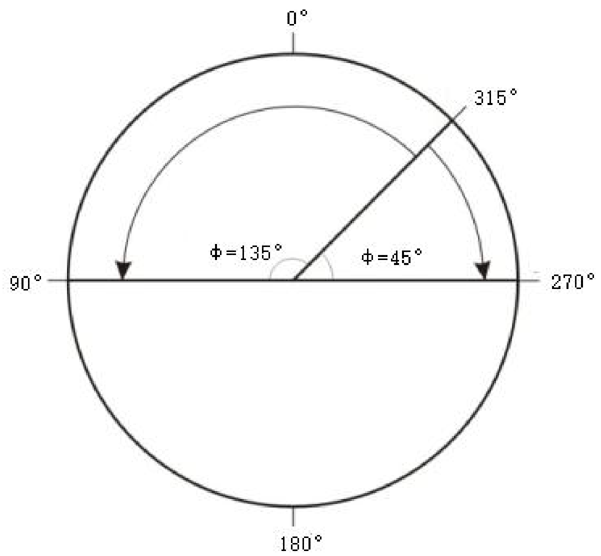
- 此指令用于控制轴按照设定速度，加减速以及加速度的变化率移动到相对于零点的目标位置。
- 绝对位移指令执行起点的轴位置为 10000，当 Position>0 （5000）时，轴将反向运动，目标位置为 5000，如左下图所示；当 Position<0（-5000）时，轴将反转，目标位置为 -5000，如右下图所示 。

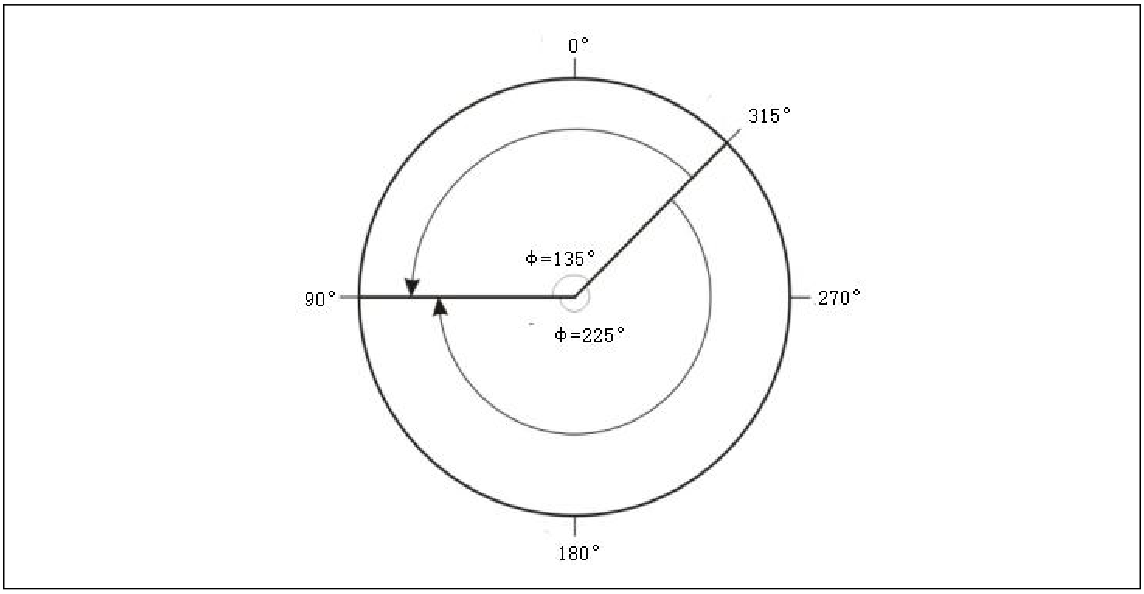


注意： 此指令在运行过程中一旦被其它指令终止，剩余未完成的距离将被丢弃，执行新的指令功能。

- **Direction**

参数 Direction 仅在轴为旋转轴时生效，其取不同值时，轴的运动方向如下表例子所示（模为 360）：

<table>
<tr><td>Direction：1(正方向)
当前位置：315
目标位置：90
移动角度：135°</td><td>Direction：3(反方向)
当前位置：315
目标位置：90
移动角度：225°</td></tr>
<tr><td>
</td><td>
</td></tr>
<tr><td>Direction：2(最短)
当前位置：315
目标位置：90
移动角度：135°</td><td>Direction：2(最短)
当前位置：315
目标位置：270
移动角度：45°</td></tr>
<tr><td colspan="2">
</td></tr>
<tr><td>Direction：4(延续当前的方向)
功能块执行前，旋转轴的状态；正在反转
当前位置：315
目标位置：90
移动角度：225°</td><td>Direction：4(延续当前的方向)
功能块执行前，轴的状态；静止、正在正转
当前位置：315
目标位置：90
移动角度：135°</td></tr>
</table>

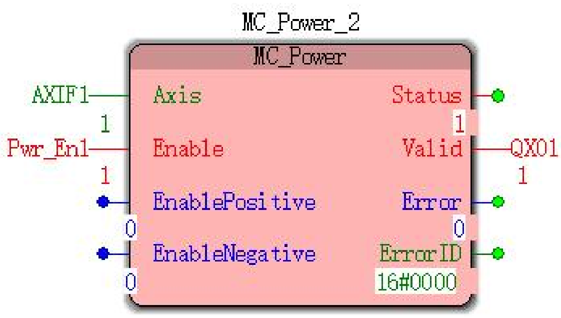


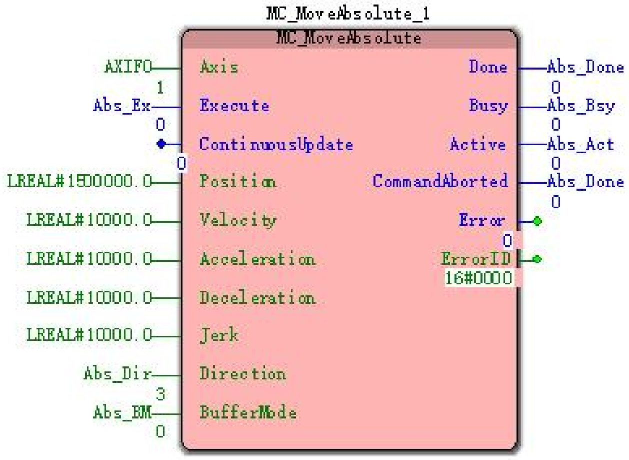
程序范例一

被控轴为直线轴，MC_MoveAbsolute 指令单独执行时的范例如下所示

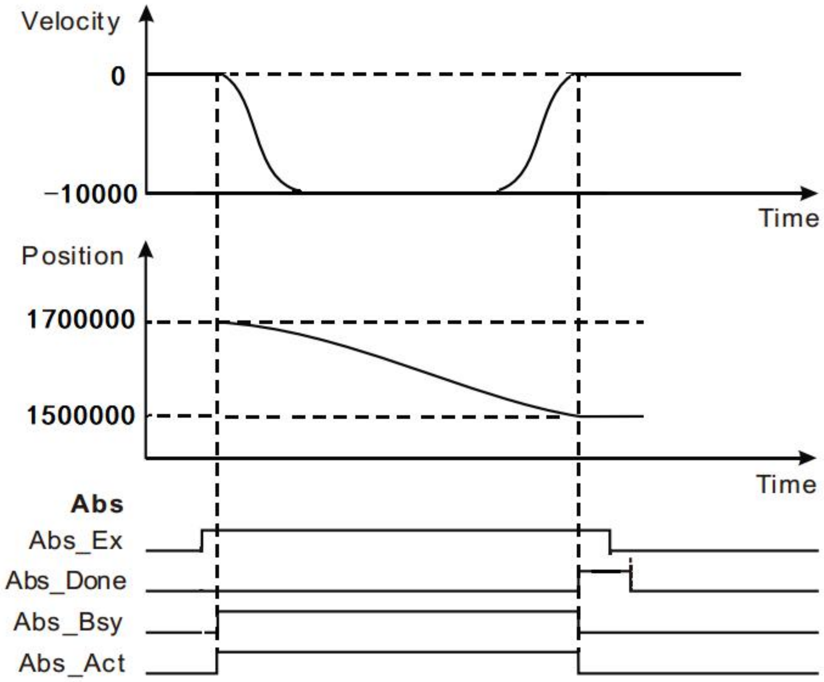
1、变量和程序

变量名	数据类型	初始值
MC_MoveAbsolue_1	MC_MoveAbsolute	
AXIF0	USINT	1
Abs_Ex	BOOL	FALSE
Abs_Dir	INT	1
Abs_BM	INT	0
Abs_Done	BOOL	
Abs_Bsy	BOOL	
Abs_Act	BOOL	
Abs_Done	BOOL	





2、运动曲线和时序图



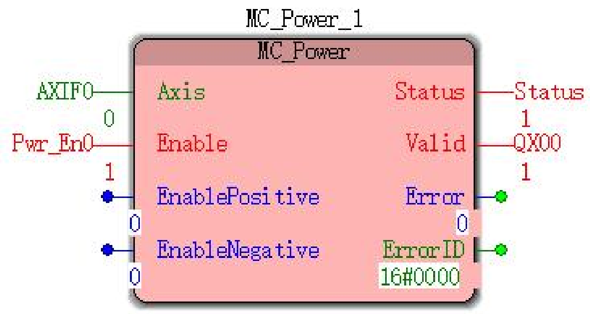
- 当 Abs_Ex 由 FALSE 变为 TRUE 时 MC_MoveAbsolute 指令开始执行，此时轴的当前位置为 1700000， 目标位置为 1500000。
- 当轴运动到 1500000 时，指令执行完成。

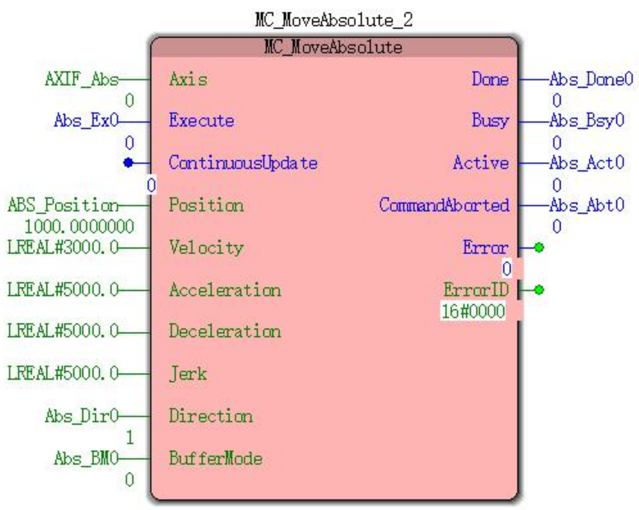
程序范例二

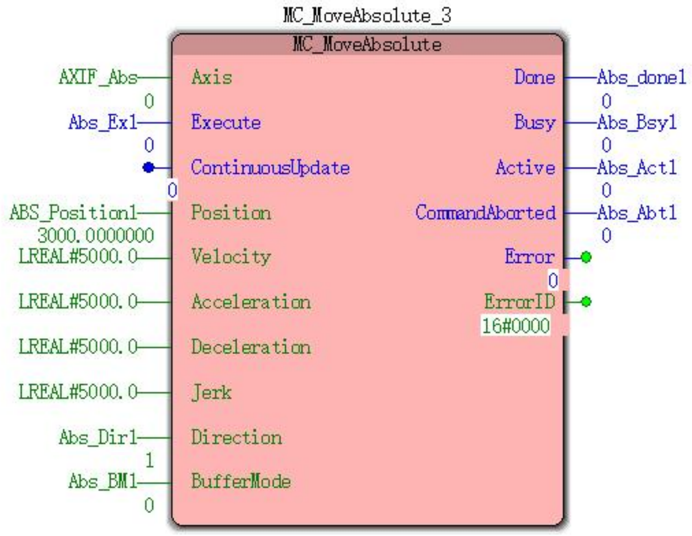
被控轴为直线轴，MC_MoveAbsolute 指令打断 MC_MoveAbsolute 指令的范例如下：

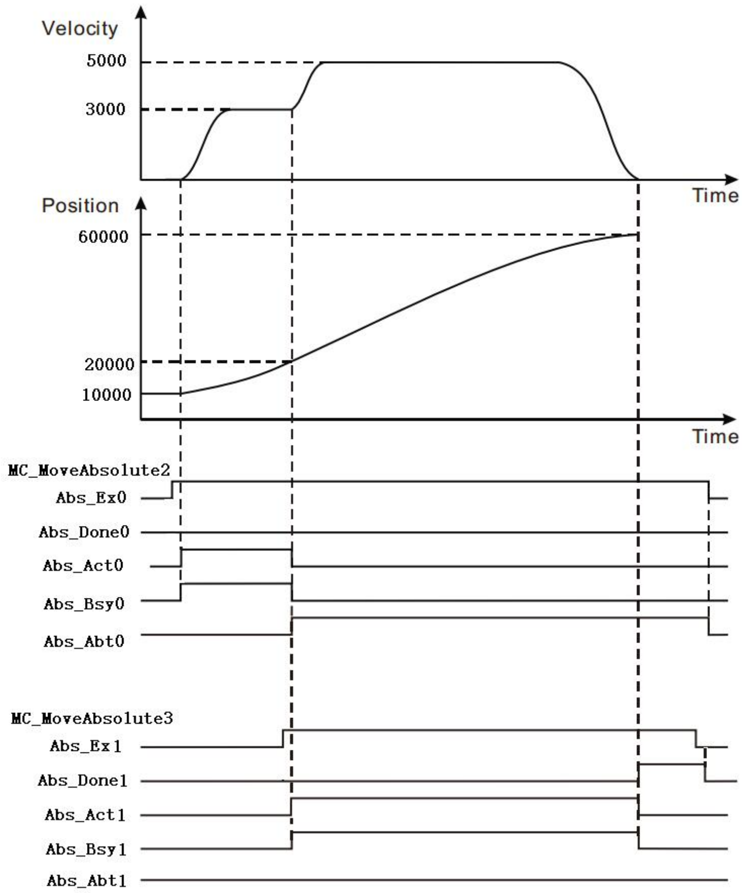
1、变量和程序

变量名	数据类型	初始值
MC_MoveAbsolue_2	MC_MoveAbsolute	-
Abs_Ex0	BOOL	FALSE
ABS_Position	LREAL	30000.0
Abs_Dir0	INT	-
Abs_BM0	INT	0
Abs_Done0	BOOL	FALSE
Abs_Bsy0	BOOL	FALSE
Abs_Act0	BOOL	FALSE
Abs_Abt0	BOOL	FALSE
MC_MoveAbsolue_3	MC_MoveAbsolute	-
Abs_Ex1	BOOL	FALSE
ABS_Position1	LREAL	60000.0
Abs_Dir1	INT	-
Abs_BM1	INT	0
Abs_Done0	BOOL	FALSE
Abs_Bsy0	BOOL	FALSE
Abs_Act0	BOOL	FALSE
Abs_Abt0	BOOL	FALSE





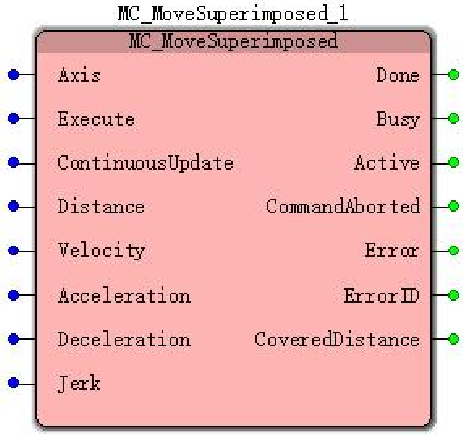




- 当 Abs_Ex0 由 FALSE 变为 TRUE 时，第一个 MC_MoveAbsolute 指令开始执行，此时轴的当前位置为 1000，目标位置为 3000。
- 当轴位置到达 20000 时，Abs_Ex1 由 FALSE 变为 TRUE，第二个 MC_MoveAbsolute 指令开始执行，且第一个 MC_MoveAbsolute 指令的执行被中断，其输出参数 Abs1_Abt 变为 TRUE。
- 当轴位置到达 60000 时，第二个 MC_MoveAbsolute 指令执行完成，其输出参数 Abs2_Done 变为 TRUE。

11.4.6 MC_MoveSuperimposed(追加位移指令)

FB/FC	说明	适用机种
FB	此指令用于控制轴在当前运动状态下按照设定速度，加减速独立的追加一段设定距离。	VEC-VA-MP-005-MA



➢ **输入参数**

名称	功能	数据类型	设定范围（缺省值）	生效时机
Axis（轴号）	设定指令欲控制的轴	USINT	**模拟量/脉冲：** 0~4（实轴） 5~11（虚轴） **CANopen 模式：** 0~15 (实轴/虚轴) (0)	Exexcute 由 FALSE 变为 TRUE
Execute (执行位)	当 Exexcute 由 FALSE 变为 TRUE，指令开始执行。	BOOL	TRUE 或 FALSE (FALSE)	-
ContinuousUpdata	保留	-	-	-
Distance （距离）	设定的目标距离 （单位：单元）	LREAL	正数、负数、零 (0)	Exexcute 由 FALSE 变为 TRUE
Velocity （速度）	设定的目标速度 （单位：单元/S）	LREAL	正数(不可缺省)	Exexcute 由 FALSE 变为 TRUE
Acceleration （加速度）	设定的目标加速度 (单位：单元/S^2)	LREAL	正数(不可缺省)	Exexcute 由 FALSE 变为 TRUE
Deceleration （减速度）	设定的目标减速度 (单位：单元/S^2)	LREAL	正数(不可缺省)	Exexcute 由 FALSE 变为 TRUE
Jerk （加速度的变化率）	设定的目标加速度或减速度的变化率 (单位：单元/S^3)	LREAL	正数、零（0）	Exexcute 由 FALSE 变为 TRUE

说明：

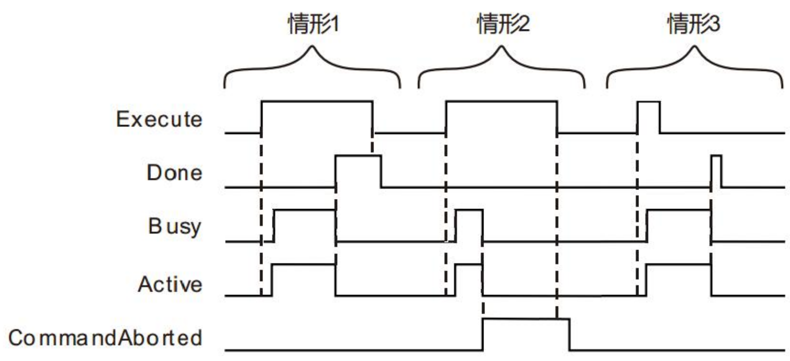
1、该指令在 Execute 由 FALSE 变为 TRUE 时开始执行。该指令正在执行，Execute 由 TRUE 变为 FALSE 时，对该指令执行无影响。

2、当指令正在执行中，Execute 再次由 FALSE 变为 TRUE 时，该指令可以重新执行，此时能够重新生效的引脚参数包括 Distance、Velocity、Acceleration、Deceleration、Jerk。

➢ **输出参数**

名称	功能	数据类型	输出范围
Done	该参数输出为 TRUE 时表示指令执行完成	BOOL	TRUE 或 FALSE
Busy（执行中）	该参数输出为 TRUE 时表示指令正在执行中	BOOL	TRUE 或 FALSE
Active（控制中）	该参数输出为 TRUE 时表示指令正在控制轴	BOOL	TRUE 或 FALSE
CommandAborted（中断）	该参数输出为 TRUE 时表示指令被中断	BOOL	TRUE 或 FALSE
Error（错误）	该指令输出为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE
ErrorID（错误代码）	执行指令出错时的错误代码	WORD	-
CoveredDistance（累计追加距离）	该指令自启动后所累计追加的距离，此参数输出范围极限按 int 型溢出处理（如果累计追加距离超过 2^32 时，此参数会取反）	LREAL	负数、正数、0

➢ **输出参数时序变化图**



情形 1：当 Execute 由 FALSE 变为 TRUE 时，一个周期后 Buys 和 Active 同时变为 TRUE；当追加位移完成时，Done 变为 TRUE，同时 Busy 和 Active 变为 FALSE，在 Execute 由 TRUE 变为 FALSE 一个周期后，Done 变为 FALSE。

情形 2：当 Execute 为 TRUE 时，该指令被其它指令中断后，CommandAborted 变为 TRUE，同时 Busy 和 Active 变为 FALSE；当 Execute 由 TRUE 变为 FALSE 时，一个周期后 CommandAborted 变为 FALSE。

情形 3： 在指令执行过程中，Execute 由 TRUE 变为 FALSE 后，指令执行完成时，Done 变为 TRUE，同时 Busy 和 Active 变为 FALSE，且一个周期后，Done 变为 FALSE。

➢ **功能说明**

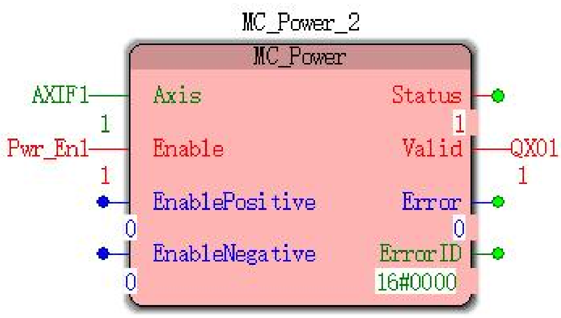
- MC_MoveSuperimposed 指令执行时，不会终止前一个指令（不包括 MC_MoveSuperimposed 和 MC_HaltSuperimposed 指令）的执行，两条指令同时执行，距离、速度、加减速会实时叠加（当某个指令到达设定速度时，其加速度为 0）。当前一个指令执行完毕，将不再叠加其速度、加减速，MC_MoveSuperimposed 指令仍然独立运行。
- 当轴状态为 Standstill 时，执行 MC_MoveSuperimposed 指令，MC_MoveSuperimposed 指令作用等同于 MC_MoveRelative 指令。
- MC_MoveSuperimposed 指令和运动指令共同控制轴时，再执行其它的运动指令（不包括 MC_MoveSuperimposed 和 MC_HaltSuperimposed 指令）。如果后执行的运动指令的 Buffermode=0，则 MC_MoveSuperimposed 指令和先执行的运动指令都会被中断；如果后执行的运动指令的 Buffermode 为其它值时，则 MC_MoveSuperimposed 指令和先执行的运动指令都不会被中断。
- MC_MoveSuperimposed 指令单独控制轴时，再执行另一个 MC_MoveSuperimposed 指令，则前一个 MC_MoveSuperimposed 指令被中断 。
- MC_MoveSuperimposed 指令执行中，再执行 MC_HaltSuperimposed 指令，则 MC_MoveSuperimposed 指令被中断 。
- 该指令不影响当前状态机

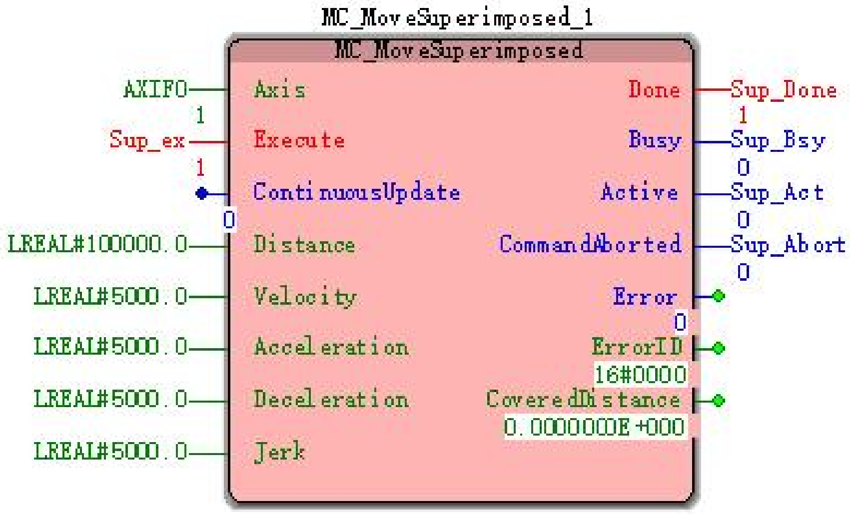
程序范例一

MC_MoveSuperimposed 指令单独执行时的范例如下所示

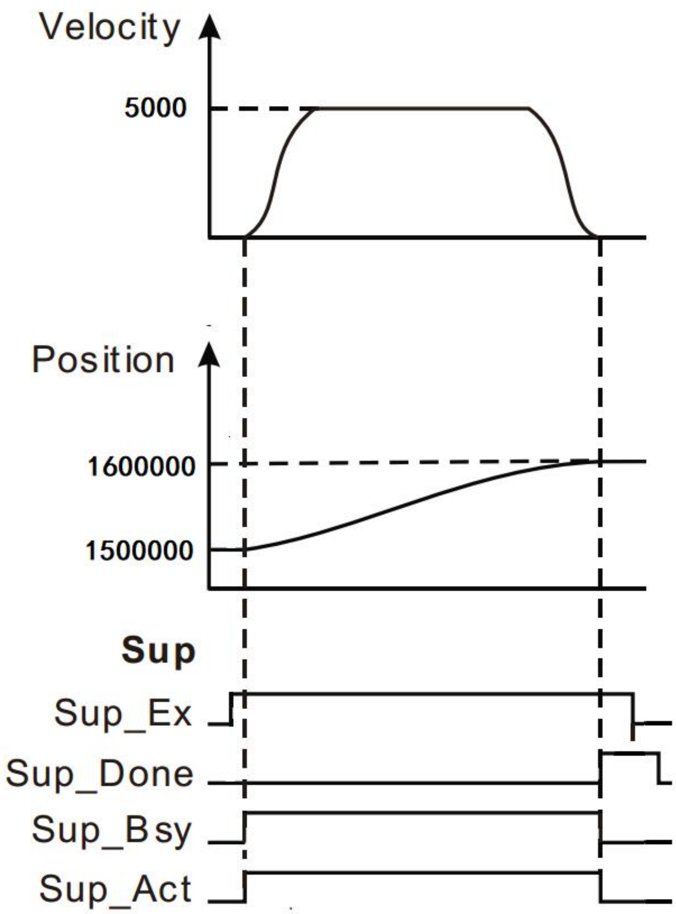
1、变量和程序

变量名	数据类型	初始值
MC_MoveSuperimposed_1	MC_MoveSuperimposed	
AXIF0	USINT	1
Sup_Ex	BOOL	FALSE
Sup_Done	BOOL	
Sup_Bsy	BOOL	
Sup_Act	BOOL	
Sup_Abort	BOOL	





2、运动曲线和时序图



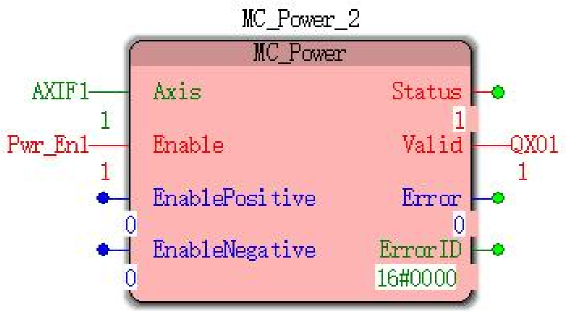
- 当 Sup_Ex 变为 TRUE 时，一个周期后，Sup_Bsy、Sup_Act 变为 TRUE，运动控制器控制伺服电机以当前位置为参考点运转。
- 伺服电机完成设定距离后，Sup_Done 变为 TRUE，同时 Sup_Bsy 和 Sup_Act 变为 FALSE。
- 当 Sup_Ex 变为 FALSE 时，Sup_Done 变为 FALSE。

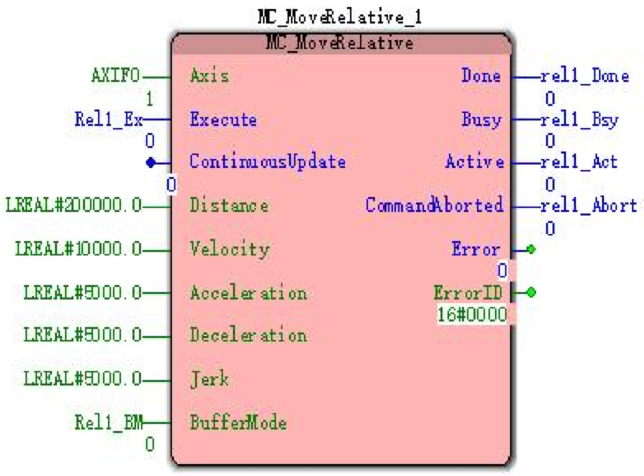
程序范例二

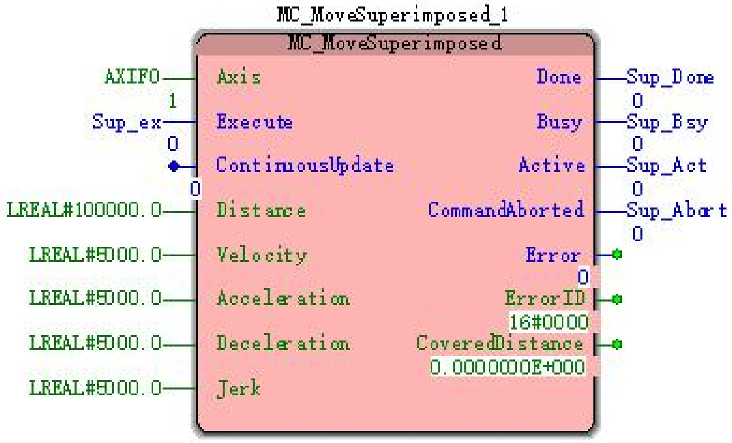
MC_MoveSuperimposed 和 MC_MoveRelative 指令搭配范例如下所示：

1、变量和程序

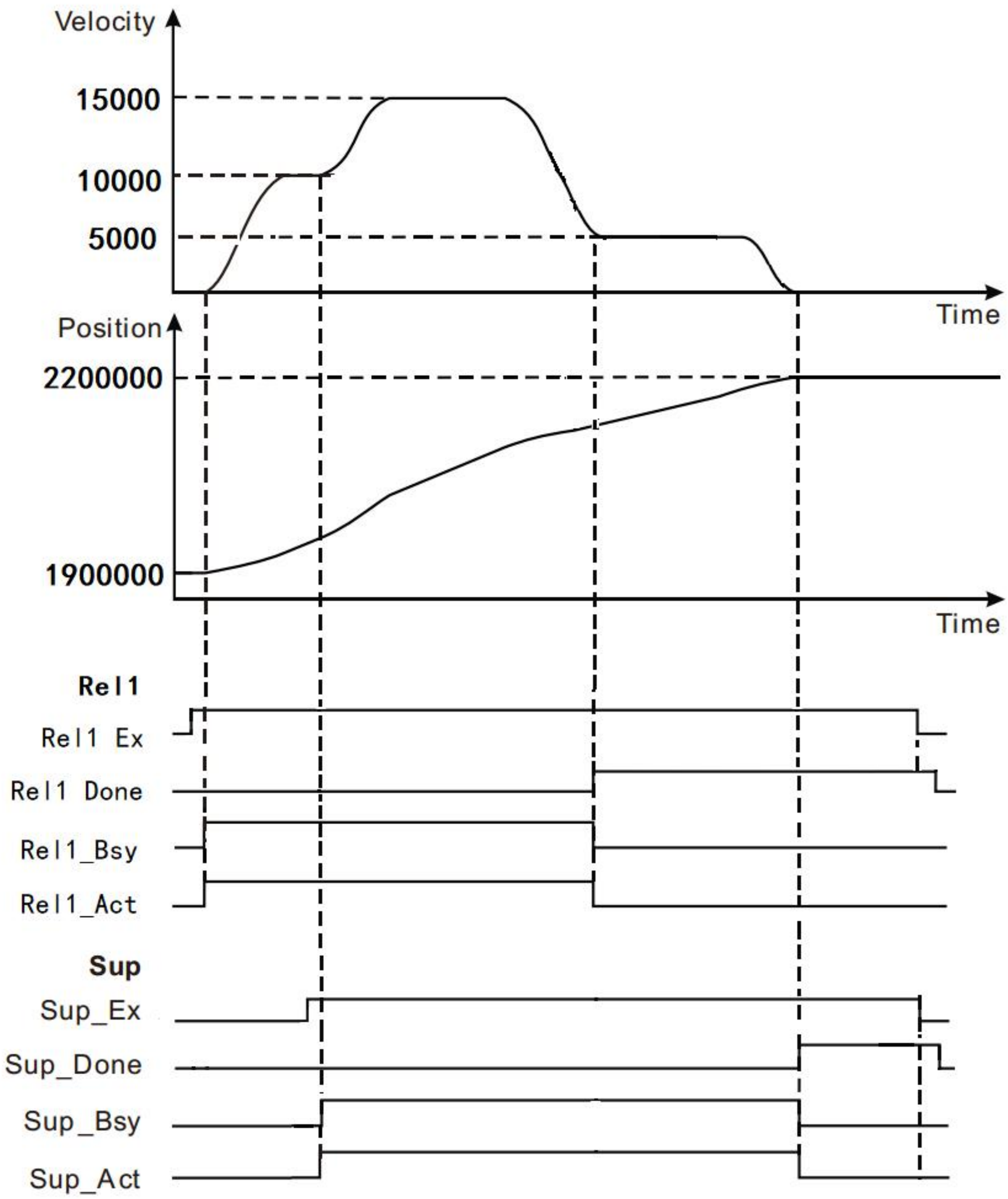
变量名	数据类型	初始值
MC_MoveRelative_1	MC_MoveRelative	
AXIF0	USINT	1
Rel1_Ex	BOOL	FALSE
Rel1_BM	INT	0
Rel1_Done	BOOL	
Rel1_Bsy	BOOL	
Rel1_Act	BOOL	
Rel1_Abort	BOOL	
MC_MoveSuperimposed_1	MC_MoveSuperimposed	
Sup_Ex	BOOL	FALSE
Sup_Done	BOOL	
Sup_Bsy	BOOL	
Sup_Act	BOOL	
Sup_Abort	BOOL	






MC_MoveSuperimposed_1
MC_MoveSuperimposed
AXIF0
Axis
Done
Sup_Done
1
Sup_ex
Execute
Busy
Sup_Bsy
ContinuousUpdate
Active
Sup_Act
LREAL#100000.0
Distance
CommandAborted
Sup_Abort
LREAL#5000.0
Velocity
Error
Acceleration
ErrorID
16#0000
Deceleration
CoveredDistance
0.0000000E+000
Jerk

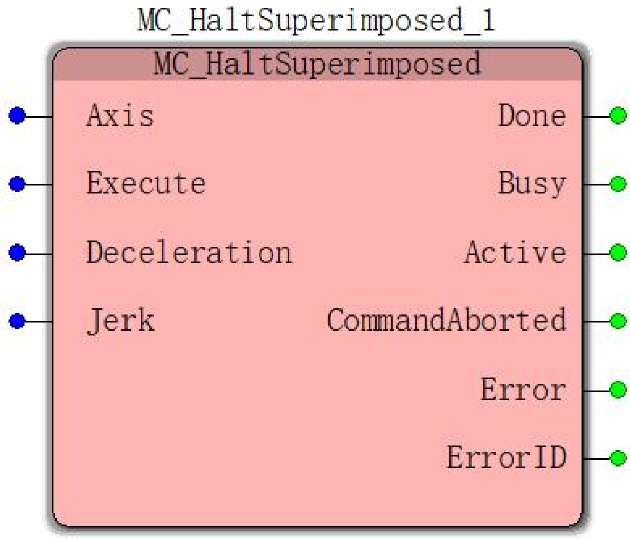
2、运动曲线和时序图


Velocity
15000
10000
5000
Time
Position
2200000
1900000
Time
Rel1
Rel1 Ex
Rel1 Done
Rel1_Bsy
Rel1_Act
Sup
Sup_Ex
Sup_Done
Sup_Bsy
Sup_Act

- 当 Rel_Ex 变为 TRUE 时，一个周期后 Rel_Act、Rel_Bsy 变为 TRUE，运动控制器控制伺服电机以当前位置为参考点运转。
- 当 Sup_Ex 变为 TRUE 时，一个周期后,Sup_Act、Sup_Bsy 变为 TRUE，MC_MoveSuperimposed 指令开始执行，伺服电机的速度以及加速度会（此时加速度为 0）叠加。
- 当 MC_MoveRelative 指令的位置完成时，Rel_Done 变为 TRUE，Rel_Bsy 和 Rel_Act 变为 FALSE。轴的最终位置为两个指令设定位置的总和加起始位置。
- 当 MC_MoveSuperimposed 指令追加距离完成时，Sup_Done 变为 TRUE，Sup_Bsy 和 Sup_Act 变为 FALSE。轴的最终位置为两个指令设定位置的总和加起始位置。
- 当 Rel_Ex 变为 FALSE 时，Rel_Done 变为 FALSE。当 Sup_Ex 变为 FALSE 时，Sup_Done
- 变为 FALSE。

11.4.7 MC_HaltSuperimposed(暂停追加位移)

FB/FC	说明	适用机种
FB	此指令用于暂停追加位移指令	VEC-VA-MP-005-MA



➢ **输入参数**

名称	功能	数据类型	设定范围（缺省值）	生效时机
Axis（轴号）	设定指令欲控制的轴	USINT	**模拟量/脉冲：** 0~4（实轴） 5~11（虚轴） **CANopen 模式：** 0~15 (实轴/虚轴) (0)	Exexcute 由 FALSE 变为 TRUE
Execute（执行位）	当 Execute 由 FALSE 变为 TRUE，指令开始执行	BOOL	TRUE 或 FALSE	
Acceleration（加速度）	设定的目标加速度 (单位：单元/S²)	LREAL	正数(不可缺省)	Exexcute 由 FALSE 变为 TRUE
Deceleration（减速度）	设定的目标减速度 (单位：单元/S²)	LREAL	正数(不可缺省)	Exexcute 由 FALSE 变为 TRUE
Jerk（加速度的变化率）	设定的目标加速度或减速度的变化率 (单位：单元/S³)	LREAL	正数、零（0）	Exexcute 由 FALSE 变为 TRUE

说明：

1、该指令在 Execute 由 FALSE 变为 TRUE 时开始执行。该指令正在执行，Execute 由 TRUE 变为 FALSE 时，对该指令执行无影响。

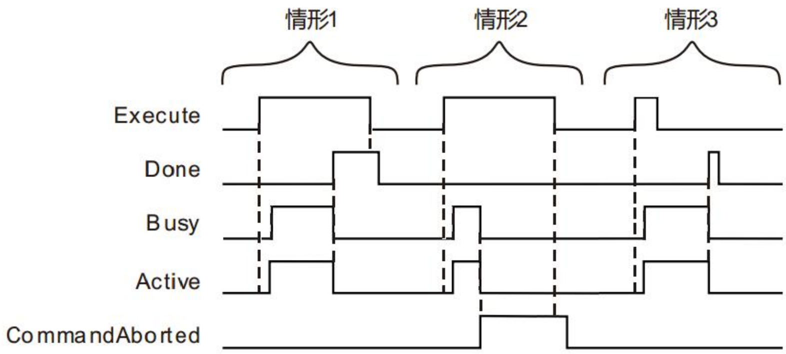
2、当指令正在执行中，Execute 再次由 FALSE 变为 TRUE 时，该指令可以重新执行，此时

能够重新生效的引脚参数包括 Deceleration、Jerk。

➢ 输出参数

名称	功能	数据类型	输出范围
Done	该参数输出为 TRUE 时表示指令执行完成	BOOL	TRUE 或 FALSE
Busy（执行中）	该参数输出为 TRUE 时表示指令正在执行中	BOOL	TRUE 或 FALSE
Active（控制中）	该参数输出为 TRUE 时表示指令正在控制轴	BOOL	TRUE 或 FALSE
CommandAborted（中断）	该参数输出为 TRUE 时表示指令被中断	BOOL	TRUE 或 FALSE
Error（错误）	该指令输出为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE
ErrorID（错误代码）	执行指令出错时的错误代码	WORD	-

➢ 输出参数时序变化图



情形 1： 当 Execute 由 FALSE 变为 TRUE 时，一个周期后 Buys 和 Active 同时变为 TRUE；当暂停追加位移完成时，Done 变为 TRUE，同时 Busy 和 Active 变为 FALSE，在 Execute 由 TRUE 变为 FALSE 一个周期后，Done 变为 FALSE。

情形 2： 当 Execute 为 TRUE 时，该指令被其它指令中断后，CommandAborted 变为 TRUE，同时 Busy 和 Active 变为 FALSE；当 Execute 由 TRUE 变为 FALSE 时，一个周期后 CommandAborted 变为 FALSE。

情形 3： 在指令执行过程中，Execute 由 TRUE 变为 FALSE 后，指令执行完成时，Done 变为 TRUE，同时 Busy 和 Active 变为 FALSE，且一个周期后，Done 变为 FALSE。

➢ 功能说明

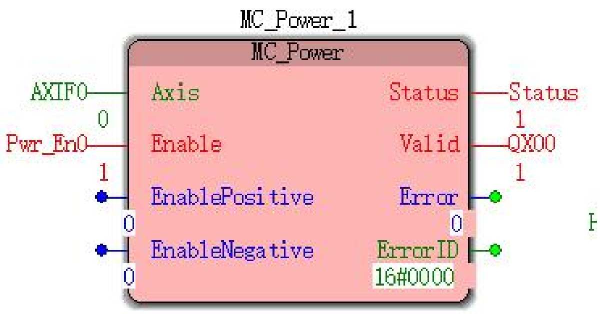
- MC_HaltSuperimposed 指令只能用于打断 MC_MoveSuperimposed 指令的执行。
- MC_MoveSuperimposed 和其它运动指令共同控制轴时，再执行 MC_HaltSuperimposed 指令，MC_HaltSuperimposed 指令会中断 MC_MoveSuperimposed 指令，但是其它运动指令的执行不受影响。

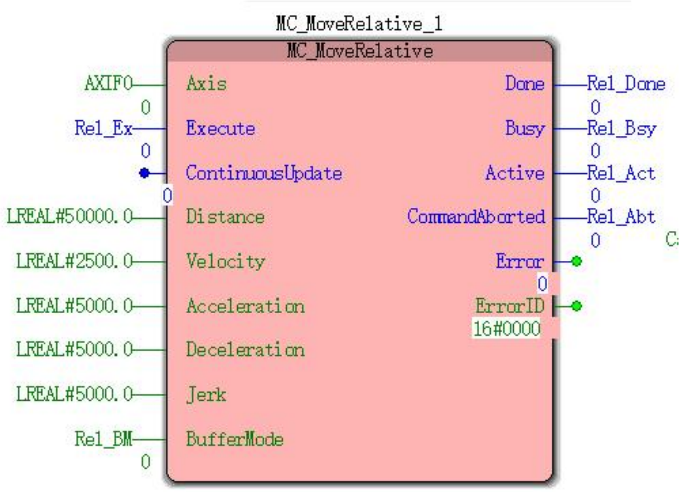
程序范例一

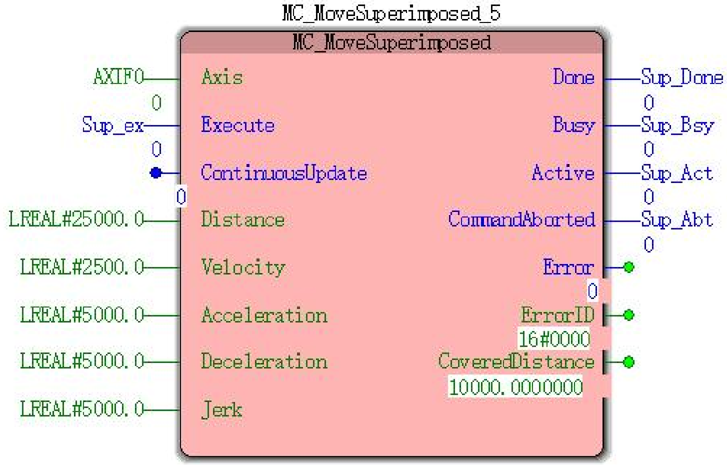
MC_HaltSuperimposed 指令暂停 MC_MoveSuperimposed 指令对 MC_MoveRelative 指令的追加的范例如下所示：

1、变量和程序

变量名	数据类型	初始值
MC_MoveRelative_1	MC_MoveRelative	-
AXIF0	USINT	1
Rel_Ex	BOOL	FALSE
Rel_BM	INT	0
Rel_Done	BOOL	FASLE
Rel_Bsy	BOOL	FALSE
Rel_Act	BOOL	FALSE
Rel_Abort	BOOL	FALSE
MC_MoveSuperimposed_5	MC_MoveSuperimposed	-
Sup_Ex	BOOL	FALSE
Sup_Done	BOOL	FALSE
Sup_Bsy	BOOL	FALSE
Sup_Act	BOOL	FALSE
Sup_Abort	BOOL	FALSE
MC_HaltSuperimposed_3	MC_HaltSuperimposed	-
HaltSup_Ex	BOOL	FALSE
HaltSup_Done	BOOL	FALSE
HaltSup_Bsy	BOOL	FALSE
HaltSup_Act	BOOL	FALSE
HaltSup_Abt	BOOL	FALSE

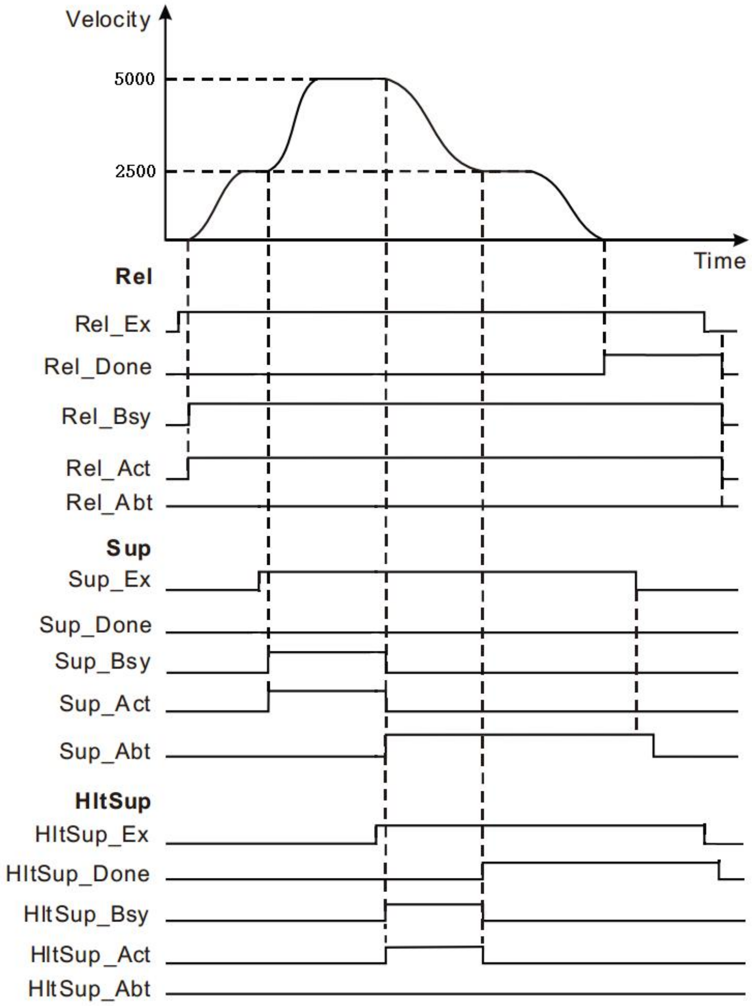



MC_MoveRelative_1
MC_MoveRelative
AXIF0
Rel_Ex
LREAL#50000.0
LREAL#2500.0
LREAL#5000.0
LREAL#5000.0
LREAL#5000.0
Rel_BM
Axis
Execute
ContinuousUpdate
Distance
Velocity
Acceleration
Deceleration
Jerk
BufferMode
Done
Busy
Active
CommandAborted
Error
ErrorID
16#0000
Rel_Done
Rel_Bsy
Rel_Act
Rel_Abt


MC_MoveSuperimposed_5
MC_MoveSuperimposed
AXIF0
Sup_ex
LREAL#25000.0
LREAL#2500.0
LREAL#5000.0
LREAL#5000.0
LREAL#5000.0
Axis
Execute
ContinuousUpdate
Distance
Velocity
Acceleration
Deceleration
Jerk
Done
Busy
Active
CommandAborted
Error
ErrorID
16#0000
CoveredDistance
10000.0000000
Sup_Done
Sup_Bsy
Sup_Act
Sup_Abt


MC_HaltSuperimposed_3
MC_HaltSuperimposed
AXIF0
HaltSup_Ex
LREAL#5000.0
LREAL#5000.0
Axis
Execute
Deceleration
Jerk
Done
Busy
Active
CommandAborted
Error
ErrorID
16#0000
HaltSup_Done
HaltSup_Bsy
HaltSup_Act
HaltSup_Abt

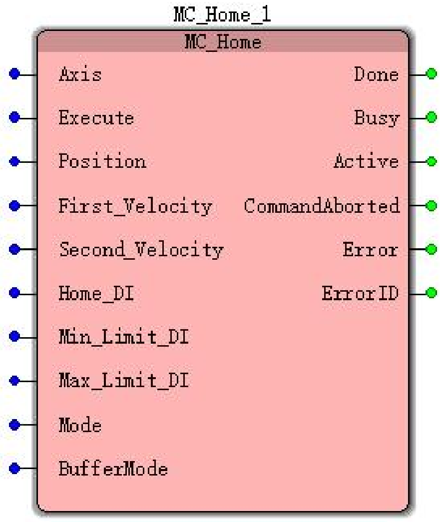
2、运动曲线和时序图



- 当 Rel_Ex 变为 TRUE 时，一个周期后，Rel_Bsy 和 Rel_Act 变为 TRUE，运动控制器控制伺服电机以当前位置为参考点运转。当 Sup_Ex 变为 TRUE 时，一个周期后，Sup_Bsy 和 Sup_Act 变为 TRUE，MC_MoveSuperimposed 指令开始执行，伺服电机的速度以及加速度会（此时轴加速度为 0）叠加。
- 当 Hltsup_Ex 变为 TRUE 时，一个周期后，Hltsup_Bsy 和 Hltsup_Act 变为 TRUE，MC_HaltSuperimposed 指令开始执行, MC_MoveSuperimposed 指令被中断，Sup_Bsy、Sup_Act 变为 FALSE ，同时 Sup_Abt 变为 TRUE 。MC_HaltSuperimposed 指令中断 MC_MoveSuperimposed 指令的执行。
- 当 Hltsup_Done 变为 TRUE 时，Hltsup_Bsy 和 Hltsup_Act 变为 FALSE。
- MC_HaltSuperimposed 指令的执行不影响 MC_MoveRelative 指令的执行。

11.4.8 MC_Home（原点回归指令）

FB/FC	说明	适用机种
FB	此指令用于控制轴按照参数设定的模式和速度执行回原点动作	VEC-VA-MP-005-MA



➢ **输入参数**

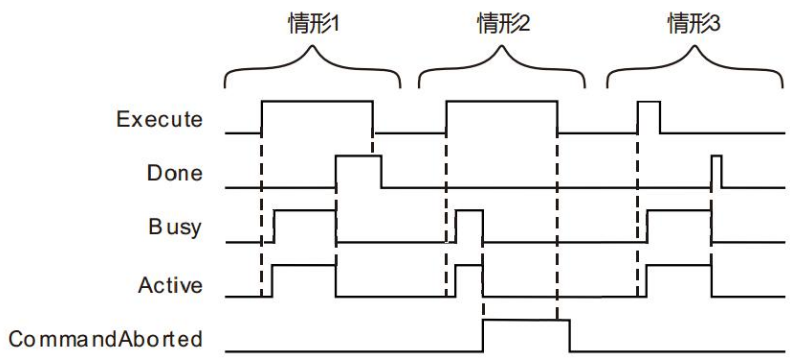
名称	功能	数据类型	设定范围（缺省值）	生效时机
Axis（轴号）	设定指令欲控制的轴	USINT	**模拟量/脉冲：** 0~4（实轴） 5~11（虚轴） **CANopen 模式：** 0~15 (实轴/虚轴) (0)	Exexcute 由 FALSE 变为 TRUE
Execute（执行位）	当 Execute 由 FALSE 变为 TRUE，指令开始执行。	BOOL	TRUE 或 FALSE	-
Position (位置)	被控轴原点偏移位置，单位：单元	LREAL	负数、正数、0	Exexcute 由 FALSE 变为 TRUE
First_Velocity（第一段速）	被控轴原点回归第一段速，单位：r/min	LREAL	正数(不可缺省)	由 Mode 决定
Second_Velocity（第二段速）	被控轴原点回归第二段速，单位：r/min	LREAL	正数(不可缺省)	由 Mode 决定
Home_DI（原点开关）	指定一个 DI 作为原点开关	INT	0~63	Exexcute 由 FALSE 变为 TRUE
Min_Limit_DI（反转极限）	指定一个 DI 作为反转极限开关	INT	0~63	Exexcute 由 FALSE 变为 TRUE
Max_Limit_DI	指定一个 DI 作为正转	INT	0~63	Exexcute 由 FALSE 变

（正转极限）	极限开关			为 TRUE
Mode（模式）	设定原点回归的模式	INT	17~30,35	Exexcute 由 FALSE 变为 TRUE
BufferMode（交接模式）	保留	-	-	-

➢ **输出参数**

名称	功能	数据类型	输出范围
Done	该参数输出为 TRUE 时表示指令执行完成	BOOL	TRUE 或 FALSE
Busy（执行中）	该参数输出为 TRUE 时表示指令正在执行中	BOOL	TRUE 或 FALSE
Active（控制中）	该参数输出为 TRUE 时表示指令正在控制轴	BOOL	TRUE 或 FALSE
CommandAborted（中断）	该参数输出为 TRUE 时表示指令被中断	BOOL	TRUE 或 FALSE
Error（错误）	该指令输出为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE
ErrorID（错误代码）	执行指令出错时的错误代码	WORD	-

➢ **输出参数时序变化图**



➢ **情形 1：** 当 Execute 由 FALSE 变为 TRUE 时，一个周期后 Buys 和 Active 同时变为 TRUE；

➢ 当定位完成时，Done 变为 TRUE，同时 Busy 和 Active 变为 FALSE，在 Execute 由 TRUE 变为 FALSE 一个周期后，Done 变为 FALSE。

➢ **情形 2：** 当 Execute 为 TRUE 时，该指令被其它指令中断后，CommandAborted 变为 TRUE，同时 Busy 和 Active 变为 FALSE；当 Execute 由 TRUE 变为 FALSE 时，一个周期后 CommandAborted 变为 FALSE。

➢ **情形 3：** 在指令执行过程中，Execute 由 TRUE 变为 FALSE 后， 当定位完成时，Done 变为 TRUE，同时 Busy 和 Active 变为 FALSE，且一个周期后，Done 变为 FALSE。

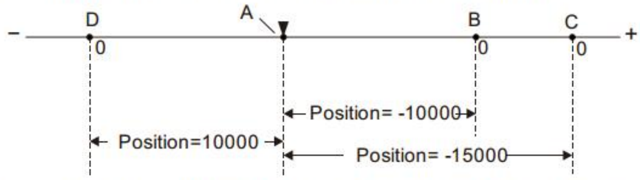
➢ **功能说明**

● 本指令根据所选回原点模式，将原点开关和正向或反向极限开关接到运动控制器的数字量输入点上以实现原点回归功能。

- 实轴在软件轴参数部分设置原点回归模式和原点回归的一二段速。原点回归模式的详细说明请见附录 A。
- 本指令只能在轴处于 StandStill 状态时才可以执行，在其它状态下执行时，本指令报错。
- 参数 Position 定义了返回原点的位置相对于伺服位置零点的偏移量。

A	机械原点，即光电开关所在的位置
▼	该指令执行成功后伺服的位置

Position 为不同数值时，在此指令的控制下，伺服最终都会停在机械原点 A 处。但是，伺服位置的参考零点却发生了变化，如下图所示。



当 Position=10000 时，伺服位置的参考零点在 D 点，A 点位置为 10000；

当 Position= -15000 时，伺服位置的参考零点在 C 点，A 点位置为 -15000；

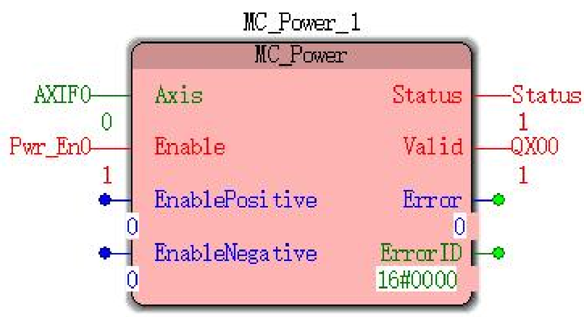
当 Position= -10000 时，伺服位置的参考零点在 B 点，A 点位置为 -10000；

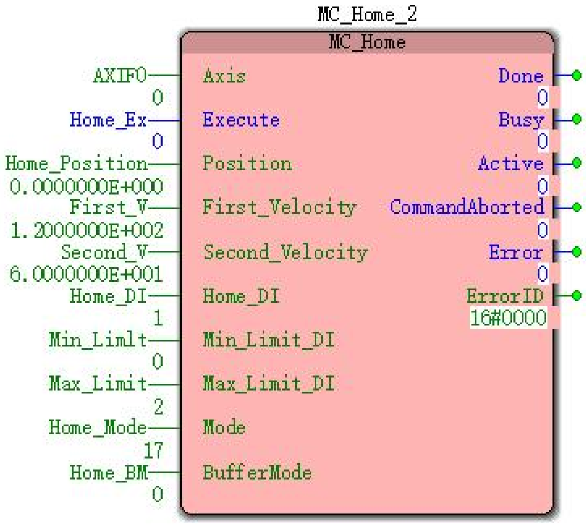
程序范例

通过机构和光电开关位置选择合适的原点回归模式，当 Home_Ex 由 FALSE 变 TRUE 时，运动控制器控制伺服电机运转，带动机构回到机械原点位置 A。

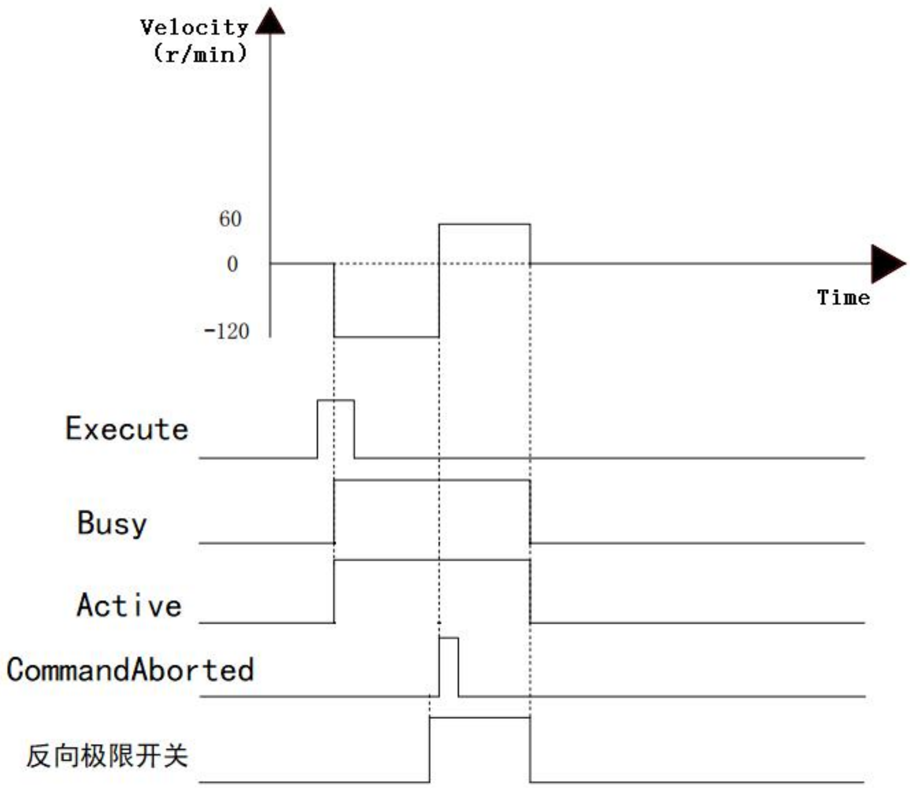
1、 **变量和程序**

变量名	数据类型	初始值
MC_Home_2	MC_Home	
Home_Ex	BOOL	FALSE
Home_Position	LREAL	0.0
First_V	LREAL	120.0
Second_V	LREAL	60.0
Home_DI	INT	1
Min_Limit	INT	0
Max_Limit	INT	2
Home_Mode	INT	17
Home_BM	INT	0





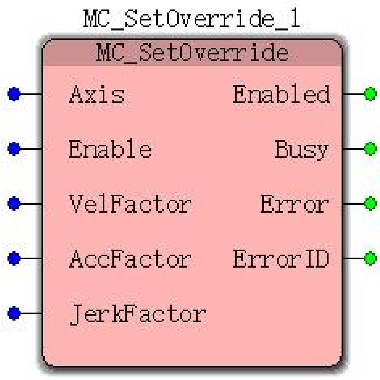
2、 运动曲线和时序图



- Mode=17，在 Home_Ex 由 FALSE 变 TRUE 时，运动控制器控制伺服电机运转，机构开始反转，到达反向极限开关后变为正转，然后在离开反向极限开关的位置停下，从而带动机构回到机械原点位置 A。

11.4.9 MC_SetOverride(速度超调指令)

FB/FC	说明	适用机种
FB	此指令用于按照百分比变更当前被控轴的目标速度	VEC-VA-MP-005-MA



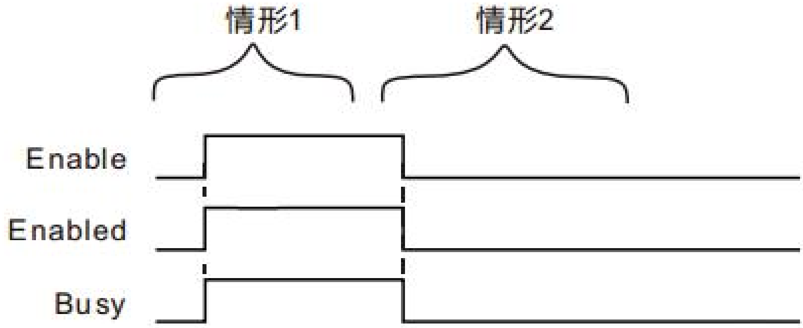
➢ **输入参数**

名称	功能	数据类型	设定范围（缺省值）	生效时机
Axis（轴号）	设定指令欲控制的轴	USINT	**模拟量/脉冲：** 0~4（实轴） 5~11（虚轴） **CANopen 模式：** 0~15 (实轴/虚轴) (0)	Enable 为 TRUE 时
Enable	当 Enable 为 TRUE 时，执行本指令	BOOL	TRUE 或 FALSE	-
VelFactor	速度超调值 （单位：%）	LREAL	0~500 （0）	Enable 为 TRUE 时
AccFactor	保留	LREAL	保留	-
JerkFactor	保留	LREAL	保留	-

➢ **输出参数**

名称	功能	数据类型	输出范围
Enabled（控制中）	该参数输出为 TRUE 时表示指令正在控制轴	BOOL	TRUE 或 FALSE
Busy（执行中）	该参数输出为 TRUE 时表示指令正在执行中	BOOL	TRUE 或 FALSE
Error（错误）	该指令输出为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE
ErrorID（错误代码）	执行指令出错时的错误代码	WORD	-

➢ **输出参数时序图**



情形 1：当 Enable 由 FALSE 变为 TRUE 时，Busy、Enabled 变为 TRUE
情形 2：当 Enable 由 TRUE 变为 FALSE 时，Enabled 和 Busy 同时变为 FALSE

➢ **功能说明**

此指令用于按百分比变更轴的目标速度

- 可变更目标速度的指令如下：

MC_MoveAbsolute（绝对位移指令）	MC_MoveAdditive（附加位移指令）
MC_MoveRelative（相对位移指令）	MC_SpecialMoveAbsolute（特殊绝对位移指令）
MC_MoveVelocity（速度指令）	

- 新的目标速度如下：
  变更后的目标速度=当前执行中指令的目标速度*VelFactor
- VelFactor 的单位为%。“100”表示“100%”。VelFactor 的有效范围为 0~500，超出有效范围时执行 MC_SetOverride 指令，指令会报错
- 对于 MC_MoveVelocity 指令，相对变更后的目标速度，轴会根据当前执行指令的 Acceleration（或 Deceleration）加速（或减速）到变更后的目标速度。
- 对于 MC_MoveAbsolute、MC_MoveAdditive、MC_MoveRelative、MC_SpecialMoveAbsolute 指令，相对变更后的目标速度，轴会根据当前执行指令的 Acceleration（或 Deceleration）加速（或减速）*$VelFactor^2$到变更后的目标速度。
- 将 VelFactor 设置为“0”时，目标速度变为“0”，轴的动作表现为减速，以速度“0”动作。
- 希望保持动作状态，但又想暂时停止轴动作时，将 VelFactor 设为“0”。此时，轴状态不会发生变化。
- 当运动指令执行时或者运动指令交接时，可以改变 VelFactor 来设定新的目标速度 Enable 为 TRUE 时，修改 VelFactor，VelFactor 立即生效，不需要重启 MC_SetOverride 指令。
- Enable 为 TRUE 时，修改 VelFactor，VelFactor 超出有效范围，MC_SetOverride 指令会报错，目标速度返回 100%。Enable 变为 FALSE 时，以 VelFactor=100 为目标进行加速或减。
- 在 MC_MoveVelocity 指令执行时，使用 MC_SetOverride 指令，当 MC_MoveVelocity 指令的 InVelocity 变为 TRUE 后，即使变更目标速度，InVelocity 也保持 TRUE 状态。
- 同一个轴只能使用一个该模块。

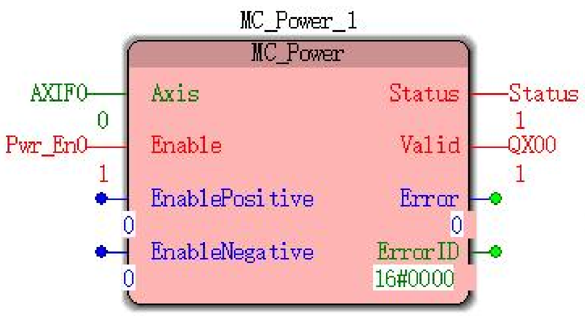
程序范例

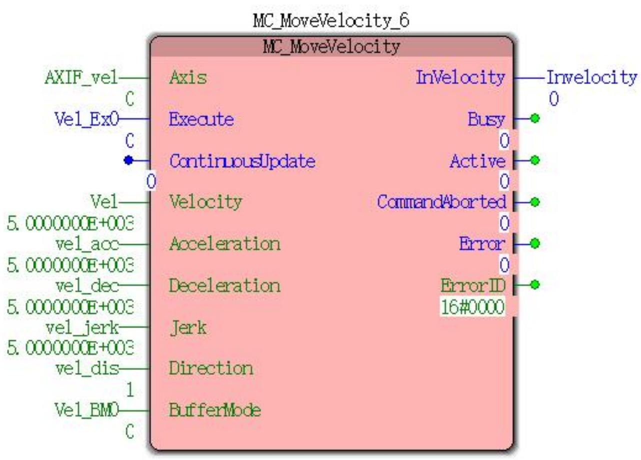
MC_SetOverride 指令执行时的范例如下所示：

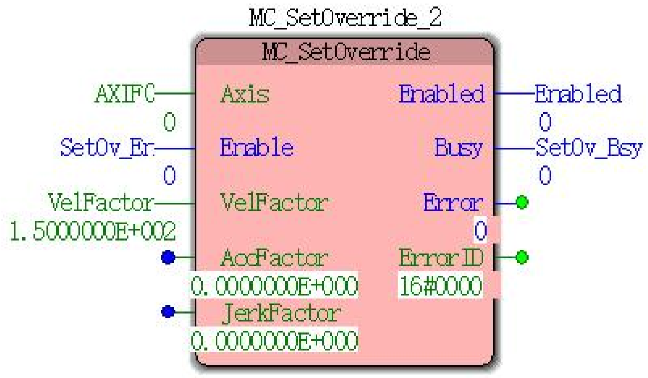
范例讲述 MC_SetOverride 指令的执行对 MC_MoveVelocity 指令执行结果的影响。

1、变量和程序名

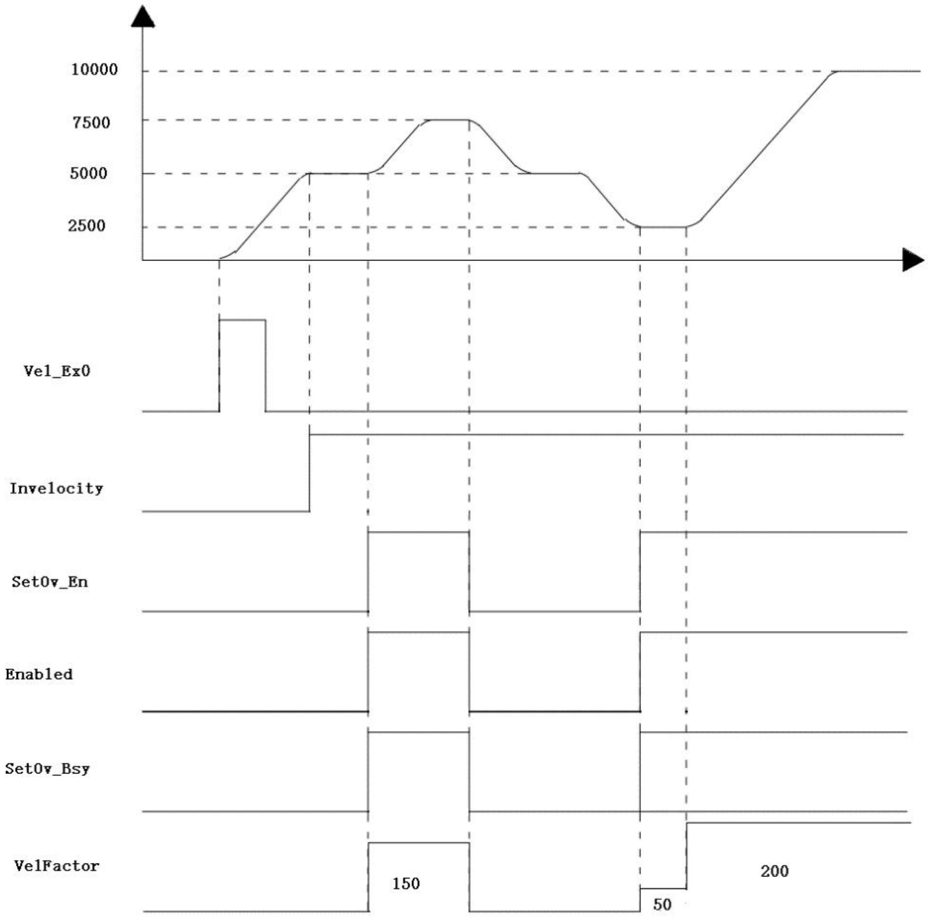
变量名	数据类型	初始值
MC_MoveVeleocity_6	MC_MoveVelocity	
AXIF0	USINT	0
Vel_EX0	BOOL	FALSE
Vel	LREAL	5000.0
Vel_acc	LREAL	5000.0
Vel_dec	LREAL	5000.0
Vel_Jerk	LREAL	5000.0
Vel_dis	INT	1
Vel_BM0	INT	0
Invelocity	BOOL	-
MC_SetOverride_2	MC_SetOverride	
SetOv_En	BOOL	FALSE
VelFactor	LREAL	150.0
Enabled	BOOL	
SetOv_Bsy	BOOL	







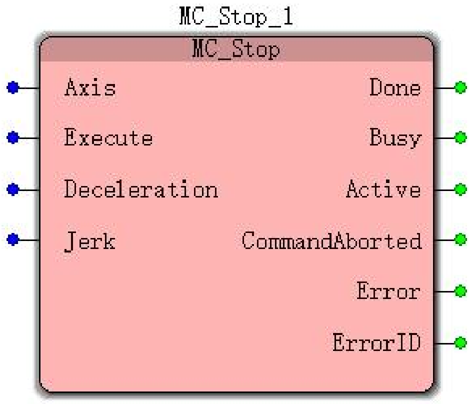
2、运动曲线和时序图



- 当 Vel_Ex0 变为 TRUE 时，一个周期后 Vel_Bsy 和 Vel_Act 变为 TRUE，轴开始正转，到达设定的速度后 Invelocity 输出 TRUE，然后将 SetOv_En 设为 TRUE，同时 Enbaled、SetOv_Bsy 变为 TRUE，MC_SetOverride 指令生效，被控轴根据 VelFactor 的值重新生成目标速度。
- 当 SetOv_En 变为 FALSE 时，相当于以 VelFactor=100 为目标速度进行减速。
- MC_SetOverride 指令执行过程中修改 VelFactor 的数值，VelFactor 的数值会立即生效，MC_MoveVelocity 指令的目标速度会相应的改变。

11.4.10 MC_Stop（停止指令）

FB/FC	说明	适用机种
FB	此指令用于控制轴按设定的减速度减速，直到停止。轴状态机进入 Stopping	VEC-VA-MP-005-MA



➢ **输入参数**

名称	功能	数据类型	设定范围（缺省值）	生效时机
Axis（轴号）	设定指令欲控制的轴	USINT	**模拟量/脉冲**： 0~4（实轴） 5~11（虚轴） **CANopen 模式**： 0~15 (实轴/虚轴) (0)	Execute 由 FALSE 变为 TRUE 时
Execute	当 Execute 由 FALSE 变为 TRUE 时，执行该指令	BOOL	TRUE 或 FALSE	-
Deceleration	减速度 (单位：单元/S^2)：	LREAL	正数(不可缺省)	
Jerk	减速度的变化率 (单位：单元/S^3)	LREAL	正数(不可缺省)	

说明：

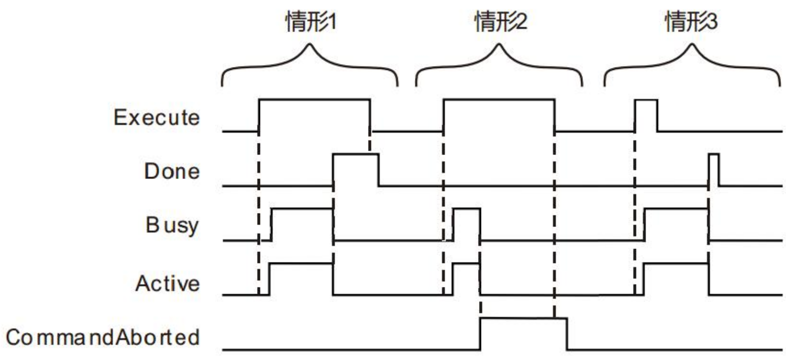
1、该指令在 Execute 由 FALSE 变为 TRUE 时开始执行。该指令正在执行，Execute 由 TRUE 变为 FALSE 时，对该指令执行无影响。

2、当指令正在执行中，Execute 再次由 FALSE 变为 TRUE 时，该指令可以重新执行，此时能够重新生效的引脚参数包括 Deceleration、Jerk。

➢ **输出参数**

名称	功能	数据类型	输出范围
Done	该参数输出为 TRUE 时表示指令执行完成	BOOL	TRUE 或 FALSE
Busy（执行中）	该参数输出为 TRUE 时表示指令正在执行中	BOOL	TRUE 或 FALSE
Active（控制中）	该参数输出为 TRUE 时表示指令正在控制轴	BOOL	TRUE 或 FALSE
CommandAborted（中断）	该参数输出为 TRUE 时表示指令被中断	BOOL	TRUE 或 FALSE
Error（错误）	该指令输出为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE
ErrorID（错误代码）	执行指令出错时的错误代码	WORD	-

➢ **输出参数时序变化图**



➢ **情形 1：** 当 Execute 由 FALSE 变为 TRUE 时，一个周期后 Buys 和 Active 同时变为 TRUE；当位置到达时，Done 变为 TRUE，同时 Busy 和 Active 变为 FALSE，在 Execute 由 TRUE 变为 FALSE 一个周期后，Done 变为 FALSE。

➢ **情形 2：** 当 Execute 为 TRUE 时，该指令被其它指令中断后，CommandAborted 变为 TRUE，同时 Busy 和 Active 变为 FALSE；当 Execute 由 TRUE 变为 FALSE 时，一个周期后 CommandAborted 变为 FALSE。

➢ **情形 3：** 在指令执行过程中，Execute 由 TRUE 变为 FALSE 后，定位完成时，Done 变为 TRUE，同时 Busy 和 Active 变为 FALSE，且一个周期后，Done 变为 FALSE。

➢ **功能说明**

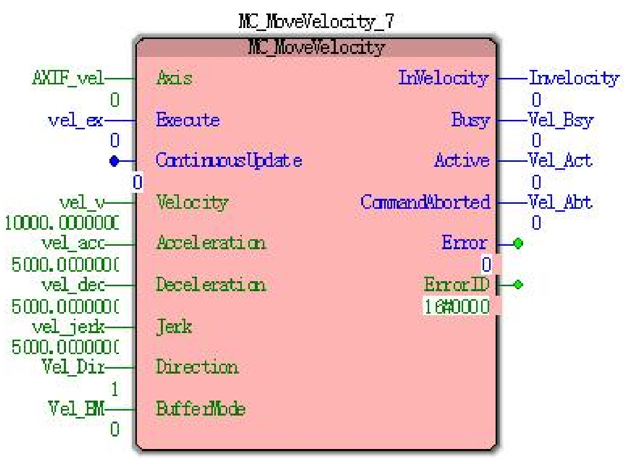
- MC_Stop 指令执行完成，轴速度降为 0 后，只要 Excute 为 TRUE，轴状态就一直为 Stopping，此时不能执行其它运动指令。
- 和 MC_Halt 指令相比较，MC_Stop 指令会锁定轴，控制器在执行 MC_Stop 指令期间不能执行其它的运动指令（不包括 MC_Stop 指令）。MC_Stop 指令执行完毕，轴已经停下来后，控制器也不可以执行其它的运动指令，只有当 MC_Stop 的 Excute 由 TRUE 变为 FALSE 后才能执行其它运动指令

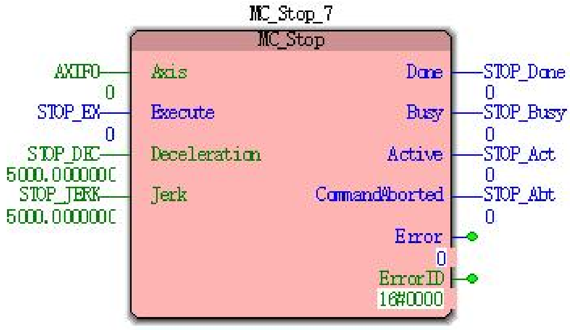
程序范例

MC_Stop 指令执行时的范例如下所示：

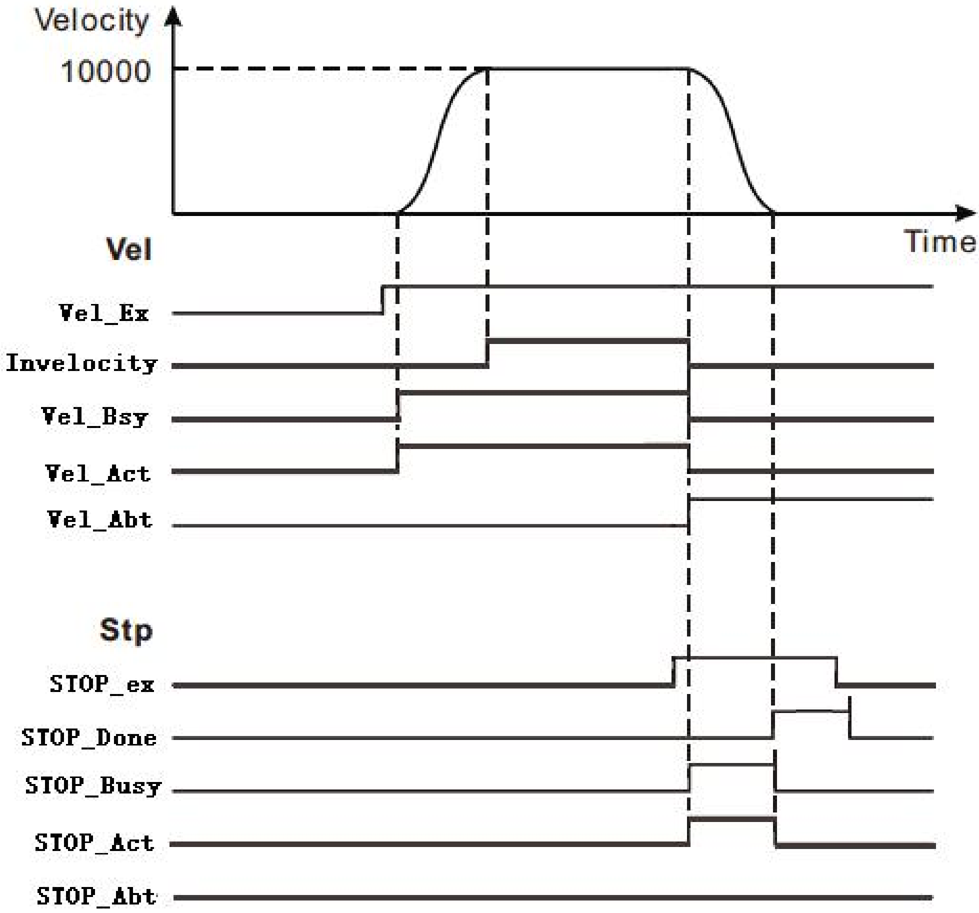
1、变量和程序名

变量名	数据类型	初始值
MC_MoveVeleocity_7	MC_MoveVelocity	-
AXIF_vel	USINT	0
vel_ex	BOOL	FALSE
Vel_v	LREAL	10000.0
Vel_acc	LREAL	5000.0
Vel_dec	LREAL	5000.0
Vel_jerk	LREAL	5000.0
Vel_Dir	INT	1
Vel_BM	INT	0
Invelocity	BOOL	
Vel_Bsy	BOOL	
Vel_Act	BOOL	
Vel_Abt	BOOL	
MC_Stop_7	MC_Stop	
AXIF0	USINT	0
STOP_EX	BOOL	FALSE
STOP_DEC	LREAL	5000.0
STOP_JERK	LREAL	5000.0
STOP_Done	BOOL	
STOP_Busy	BOOL	
STOP_Act	BOOL	
STOP_Abt	BOOL	





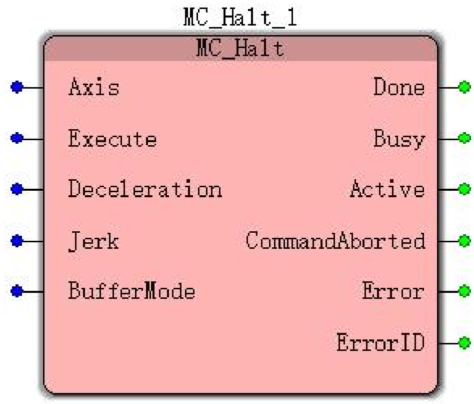
2、运动曲线和时序图



- 当 Vel_Ex 变为 TRUE 时，一个周期后 Vel_Bsy、Vel_Act 变为 TRUE，伺服电机开始正转。当伺服电机达到目标速度时，Invelocity 变为 TRUE。
- 当 STOP_ex 变为 TRUE 时，一个周期后 STOP_Busy、STOP_Act 变为 TRUE，同时 Invelocity 变为 FALSE，Vel_Abt 变为 TRUE，伺服电机开始减速。
- 当轴速度减为 0 时，STOP_Done 变为 TRUE，同时 STOP_Busy、STOP_Act 变为 FALSE。
- 当 STOP_ex 变为 FALSE 时，一个周期后 STOP_Done 变为 FALSE。

11.4.11 MC_Halt（暂停指令）

FB/FC	说明	适用机种
FB	此指令用于控制轴按设定的减速度减速，直到停止。	VEC-VA-MP-005-MA



➢ **输入参数**

名称	功能	数据类型	设定范围（缺省值）	生效时机
Axis（轴号）	设定指令欲控制的轴	USINT	**模拟量/脉冲：** 0~4（实轴） 5~11（虚轴） **CANopen 模式：** 0~15 (实轴/虚轴) (0)	Execute 由 FALSE 变为 TRUE 时
Execute（执行位）	当 Execute 由 FALSE 变为 TRUE 时，执行该指令	BOOL	TRUE 或 FALSE	-
Deceleration（减速度）	减速度 (单位：单元/S^2)：	LREAL	正数(不可缺省)	Execute 由 FALSE 变为 TRUE 时
Jerk（减速度的变化率）	减速度的变化率 (单位：单元/S^3)	LREAL	正数(不可缺省)	Execute 由 FALSE 变为 TRUE 时
BufferMode（交接模式）	设定两个指令之间交接模式 0:立即打断 1：等待	INT	0:立即打断 1：等待 (0)	Execute 由 FALSE 变为 TRUE 时

说明：

1、该指令在 Execute 由 FALSE 变为 TRUE 时开始执行。该指令正在执行，Execute 由 TRUE 变为 FALSE 时，对该指令执行无影响。

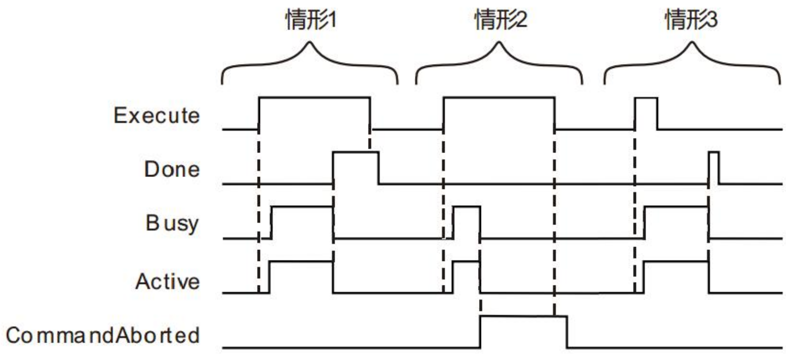
2、当指令正在执行中，Execute 再次由 FALSE 变为 TRUE 时，该指令可以重新执行，此时

能够重新生效的引脚参数包括 Deceleration、Jerk、BufferMode。

- **输出参数**

名称	功能	数据类型	输出范围
Done	该参数输出为 TRUE 时表示指令执行完成	BOOL	TRUE 或 FALSE
Busy（执行中）	该参数输出为 TRUE 时表示指令正在执行中	BOOL	TRUE 或 FALSE
Active（控制中）	该参数输出为 TRUE 时表示指令正在控制轴	BOOL	TRUE 或 FALSE
CommandAborted（中断）	该参数输出为 TRUE 时表示指令被中断	BOOL	TRUE 或 FALSE
Error（错误）	该指令输出为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE
ErrorID（错误代码）	执行指令出错时的错误代码	WORD	-

- **输出参数变化时序图**

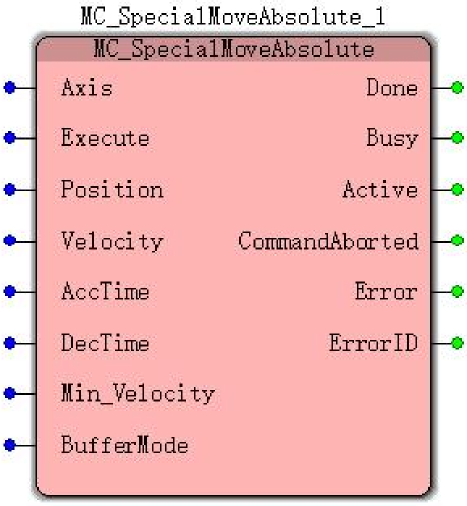


- **情形 1：** 当 Execute 由 FALSE 变为 TRUE 时，一个周期后 Buys 和 Active 同时变为 TRUE；
- 当位置到达时，Done 变为 TRUE，同时 Busy 和 Active 变为 FALSE，在 Execute 由 TRUE 变为 FALSE 一个周期后，Done 变为 FALSE。
- **情形 2：**当 Execute 为 TRUE 时，该指令被其它指令中断后，CommandAborted 变为 TRUE，同时 Busy 和 Active 变为 FALSE；当 Execute 由 TRUE 变为 FALSE 时，一个周期后 CommandAborted 变为 FALSE。
- **情形 3：** 在指令执行过程中，Execute 由 TRUE 变为 FALSE 后，定位完成时，Done 变为 TRUE，同时 Busy 和 Active 变为 FALSE，且一个周期后，Done 变为 FALSE。

- **功能说明**
  - MC_Halt 指令开始执行，状态机进入 DiscreteMotion，当轴速度降为 0 时，Done 变为 TRUE，同时状态机变为 Standstill。
  - 和 MC_Stop 指令相比较，MC_Halt 指令不会锁定轴，控制器可以执行其它的运动指令。在 MC_Halt 指令执行期间，轴在进行减速时，可以执行其它的运动指令来中断 MC_Halt 指令；在 MC_Halt 指令执行完毕，轴已经停下来后，控制器可以执行其它的运动指令来重新启动轴。

11.4.11 MC_SpecialMoveAbsolute（特殊绝对位移指令）

FB/FC	说明	适用机种
FB	此指令用于控制轴按照设定速度，加减速移动到相对于零点的目标位置	VEC-VA-MP-005-MA



➢ **输入参数**

名称	功能	数据类型	设定范围（缺省值）	生效时机
Axis（轴号）	设定指令欲控制的轴	USINT	**脉冲模式：** 0~4（实轴） 5~11（虚轴） (0)	Exexcute 由 FALSE 变为 TRUE
Execute (执行位)	当 Exexcute 由 FALSE 变为 TRUE，指令开始执行。	BOOL	TRUE 或 FALSE (FALSE)	-
Position （位置）	设定的目标位置 （单位：单元）	LREAL	正数、负数、零 (0)	Exexcute 由 FALSE 变为 TRUE
Velocity （速度）	设定的目标速度 （单位：单元/S）	LREAL	正数(不可缺省)	Exexcute 由 FALSE 变为 TRUE
AccTime (加速时间)	设定的目标加速时间 (单位：S)	LREAL	正数(不可缺省)	Exexcute 由 FALSE 变为 TRUE
DecTime (减速时间)	设定的目标减速时间 (单位:S)	LREAL	正数(不可缺省)	Exexcute 由 FALSE 变为 TRUE
Min_Velocity	设定的目标最小转动速度 (单位：单元/S)	LREAL	正数(不可缺省)	Exexcute 由 FALSE 变为 TRUE
BufferMode （交接模式）	设定两个指令之间交接模式 0:立即打断	INT	0:立即打断 1：等待 (0)	Exexcute 由 FALSE 变为 TRUE

	1：等待			

说明：

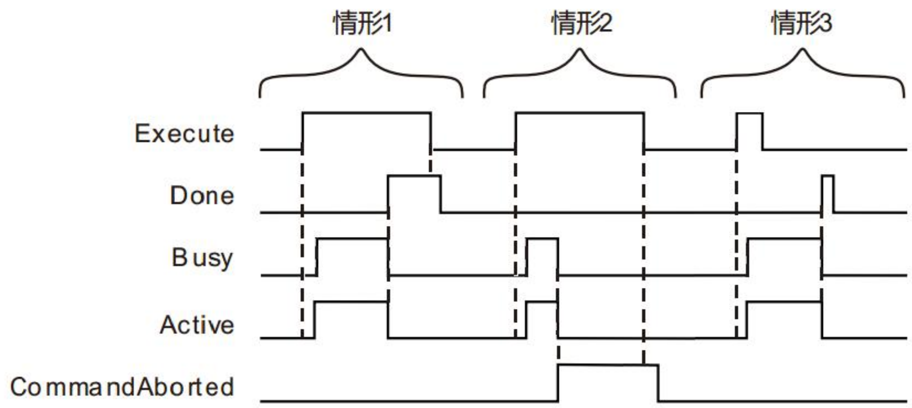
1、执行该指令时，被控轴以 Min_Velocity 启动，以 Min_Velocity 停止。

2、该指令应在 ControlModel=1 下执行，它是一种脉冲量半闭环控制，在还未执行该指令前，根据该模块上设置的输入参数，规划出给定位置和给定速度的对应表，运行时，根据当前反馈脉冲，决定给定速度。因此本模块无正反调节，用于大惯量控制较会比较稳定。

➢ **输出参数**

名称	功能	数据类型	输出范围
Done	该参数输出为 TRUE 时表示指令执行完成	BOOL	TRUE 或 FALSE
Busy（执行中）	该参数输出为 TRUE 时表示指令正在执行中	BOOL	TRUE 或 FALSE
Active（控制中）	该参数输出为 TRUE 时表示指令正在控制轴	BOOL	TRUE 或 FALSE
CommandAborted（中断）	该参数输出为 TRUE 时表示指令被中断	BOOL	TRUE 或 FALSE
Error（错误）	该指令输出为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE
ErrorID（错误代码）	执行指令出错时的错误代码	WORD	-

➢ **输出参数时序变化图**



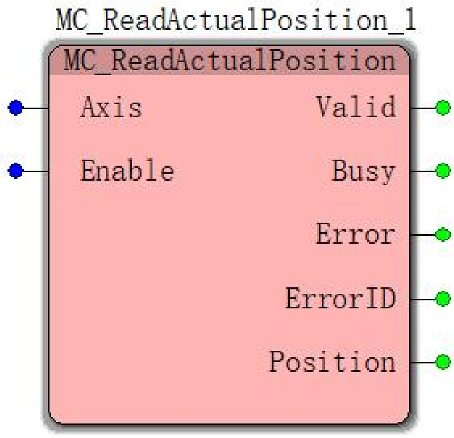
➢ **情形 1：**当 Execute 由 FALSE 变为 TRUE 时，一个周期后 Buys 和 Active 同时变为 TRUE；当定位完成时，Done 变为 TRUE，同时 Busy 和 Active 变为 FALSE，在 Execute 由 TRUE 变为 FALSE 一个周期后，Done 变为 FALSE。

➢ **情形 2：**当 Execute 为 TRUE 时，该指令被其它指令中断后，CommandAborted 变为 TRUE，同时 Busy 和 Active 变为 FALSE；当 Execute 由 TRUE 变为 FALSE 时，一个周期后 CommandAborted 变为 FALSE。

- **情形 3**： 在指令执行过程中，Execute 由 TRUE 变为 FALSE 后， 指令执行完成时，Done 变为 TRUE，同时 Busy 和 Active 变为 FALSE，且一个周期后，Done 变为 FALSE。

11.4.12 MC_ReadActualPosition（读取实时位置指令）

FB/FC	说明	适用机种
FB	此指令用于读取轴的实际位置	VEC-VA-MP-005-MA



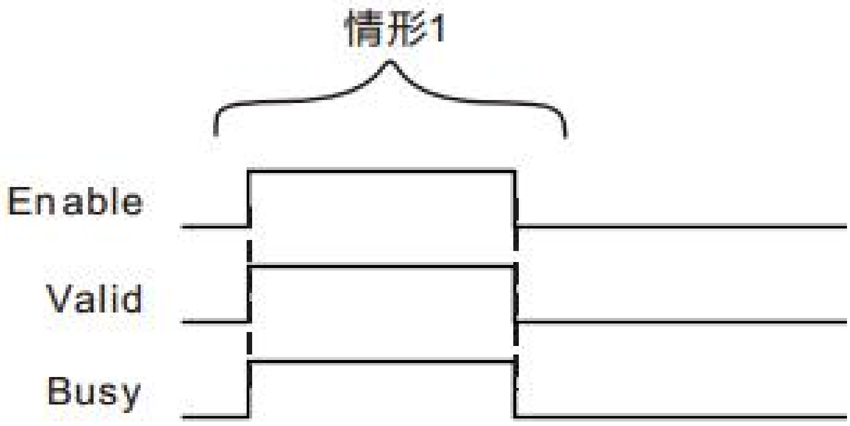
➢ **输入参数**

名称	功能	数据类型	设定范围（缺省值）	生效时机
Axis（轴号）	设定指令欲控制的轴	USINT	**模拟量/脉冲：** 0~4（实轴） 5~11（虚轴） **CANopen 模式：** 0~15 (实轴/虚轴) (0)	Enable 由为 TRUE
Enable（执行位）	当 Enable 为 TRUE 时，指令执行	BOOL	TRUE 或 FALSE	-

➢ **输出参数**

名称	功能	数据类型	输出范围
Valid（输出有效）	该参数输出为 TRUE 时表示指令输出有效	BOOL	TRUE
Busy（执行中）	该参数输出为 TRUE 时表示指令正在执行中	BOOL	TRUE 或 FALSE
Error（错误）	该指令输出为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE
ErrorID（错误代码）	执行指令出错时的错误代码	WORD	-
Position（实际位置）	轴实际位置	LREAL	实数

➢ **输出变化时序图**



情形 1：当 Enable 由 FALSE 变为 TRUE，Valid 和 Busy 同时变为 TRUE,。当 Enable 变为 FALSE 时，Valid、Busy 均变为 FALSE。

➢ **功能说明**

此指令用于读取轴的实际位置（包括实轴、虚轴与编码器轴）

- 实际位置

此指令读取的实际位置的单位为单元，而伺服驱动器反馈给控制器的位置的单位为脉冲，故此实际位置由伺服驱动器的位置反馈脉冲数经转换后得到，转换时需使用轴参数中的 Motor_PPC、Reductor_Num、Reductor_Den、机构导程/周长（Screw_Lead/Disc_Circumference）。转换关系如下公式所示：

$$\text{ActualPosition}=\frac{\text{机构导程/周长}}{(Motor\_PPC)*\frac{Reductor\_Num}{Reductor\_Den}}*\text{伺服位置反馈脉冲数}$$

若轴为直线轴，则指令执行时输出的 Position=ActualPosition；

若轴为旋转轴，则指令执行时输出的 Position=ActualPosition%Modulo（Position 为 ActualPosition 按轴参数设置的模做取余运算的结果），所以 Position 的值在 0~Modulo 之间变化。

- **实际位置更新时机**

由于此实际位置来源于伺服驱动器的位置反馈脉冲数，则实际位置的刷新时机由 MC_AXIS_REF 设置的 Sample_Time 采样编码器反馈脉冲的时间决定。在一个采样周期内，伺服反馈其位置脉冲数给控制器的动作仅发生一次。因此，此指令读取轴实际位置的实时性不及位置捕获，若需获取实时性更高的位置，请使用位置捕获功能。

- **MC_SetPosition 对实际位置的影响**

指令 MC_SetPosition 执行后，MC_ReadActualPosition 指令读取的实际位置须加上是 MC_SetPosition 指令引起的位置偏移量，换算关系式如下公式所示：

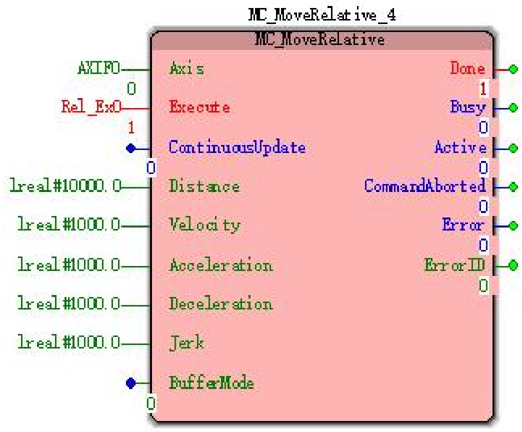
$$\text{ActualPosition}=\text{MC\_SetPosition 引起的位置偏移量}+\frac{\text{机构导程/周长}}{(Motor\_PPC)*\frac{Reductor\_Num}{Reductor\_Den}}*\text{伺服位置反馈脉冲数}$$

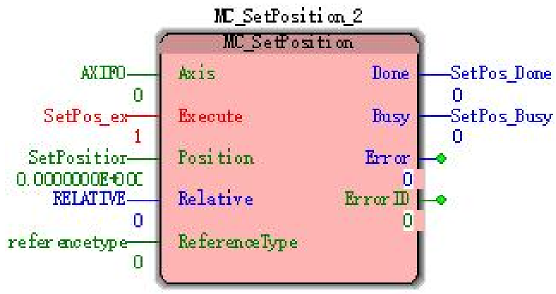
程序范例

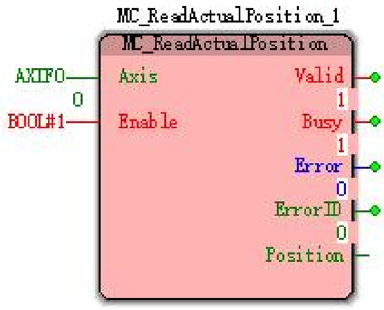
本例描述 MC_SetPosition 指令对 MC_ReadActualPosition 指令的影响，范例程序如下：

1、变量和程序

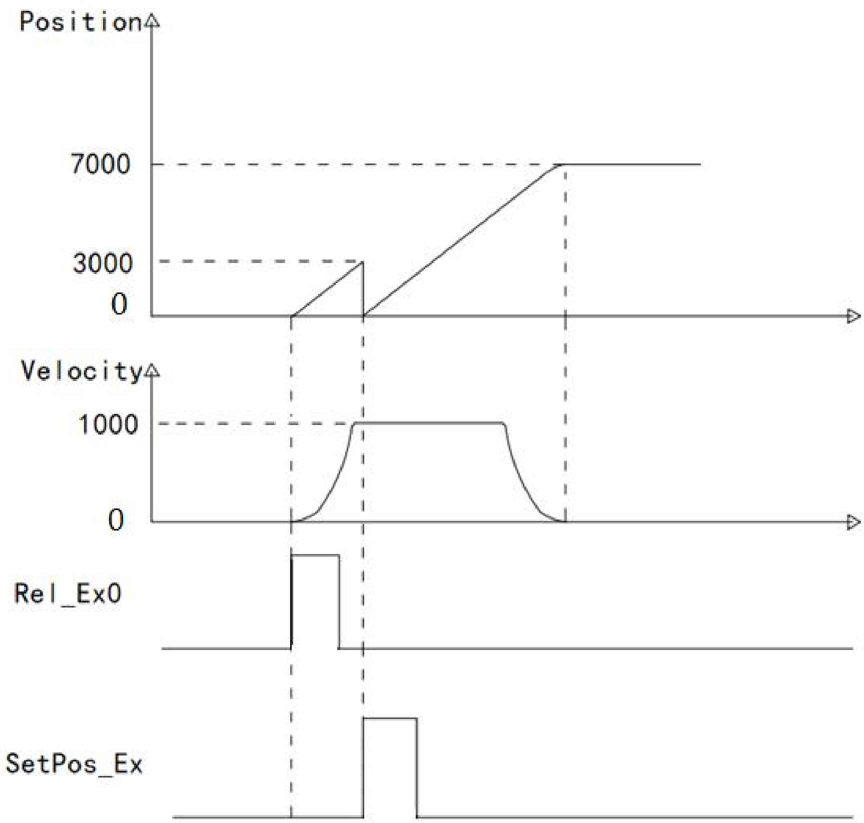
变量名	数据类型	初始值
MC_MoveRelative_4	MC_MoveRelative	-
AXIF0	USINT	0
Rel_Ex0	BOOL	-
MC_SetPosition_2	MC_SetPosition	-
SetPos_ex	BOOL	-
SetPosition	LREAL	0.0
RELATIVE	BOOL	FALSE
referencetype	BOOL	FALSE
Read_P0	LREAL	-







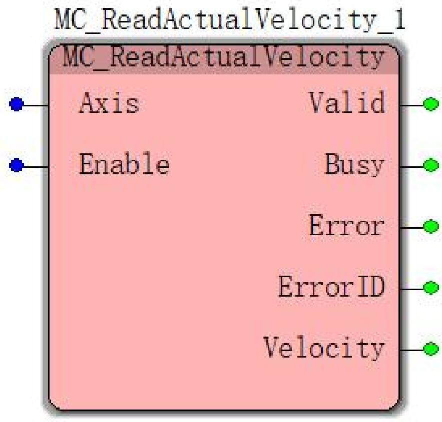
2、运动曲线和时序图



- 执行 MC_ReadActualPosition 指令读取实时位置，执行 MC_MoveRelative，被控轴以设定速度向目标位置 10000 运动。
- ActualPosition 为 3000 时，执行 MC_SetPosition 指令，由于 MC_SetPosition 指令选择为绝对位置，因此，执行完成后 ActualPosition=0，MC_MoveRelativel 指令执行完成时的 ActualPosition 为 7000。
- 通过上图中的速度变化曲线可以看出 MC_SetPosition 指令对正在执行的运动不会产生影响，但 ActualPosition 曲线却反映出 MC_RealActualPosition 读到的 ActualPosition 值会受到 MC_SetPosition 指令的影响。

11.4.13 MC_ReadActualVelocity（读取实时速度）

FB/FC	说明	适用机种
FB	此指令用于读取轴的实际速度	VEC-VA-MP-005-MA



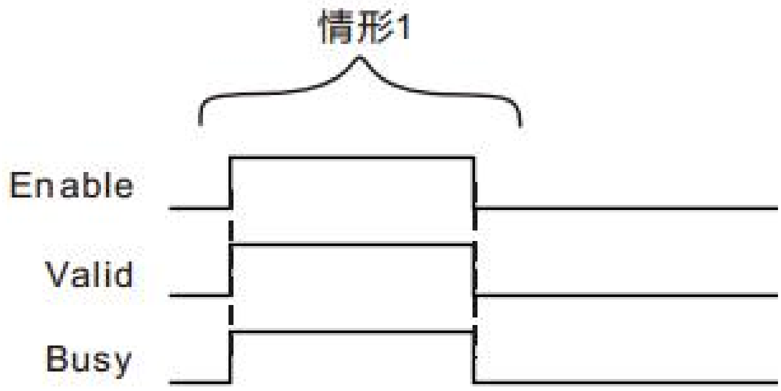
- **输入参数**

名称	功能	数据类型	设定范围（缺省值）	生效时机
Axis（轴号）	设定指令欲控制的轴	USINT	**模拟量/脉冲：** 0~4（实轴） 5~11（虚轴） **CANopen 模式：** 0~15 (实轴/虚轴) (0)	Enable 由为 TRUE
Enable（执行位）	当 Enable 为 TRUE 时，指令执行	BOOL	TRUE 或 FALSE	-

- **输出参数**

名称	功能	数据类型	输出范围
Valid（输出有效）	该参数输出为 TRUE 时表示指令输出有效	BOOL	TRUE
Busy（执行中）	该参数输出为 TRUE 时表示指令正在执行中	BOOL	TRUE 或 FALSE
Error（错误）	该指令输出为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE
ErrorID（错误代码）	执行指令出错时的错误代码	WORD	-
Velocity（实际速度）	轴实际速度 单位：单元/S	REAL	实数

- **输出变化时序图**



情形 1： 当 Enable 由 FALSE 变为 TRUE，Valid 和 Busy 同时变为 TRUE,。当 Enable 变为 FALSE 时，Valid、Busy 均变为 FALSE。

➢ **功能说明**

该指令读取的速度为终端执行机构的移动速度 Velocity，其单位为单元/S，折算到电机的每分钟转数：

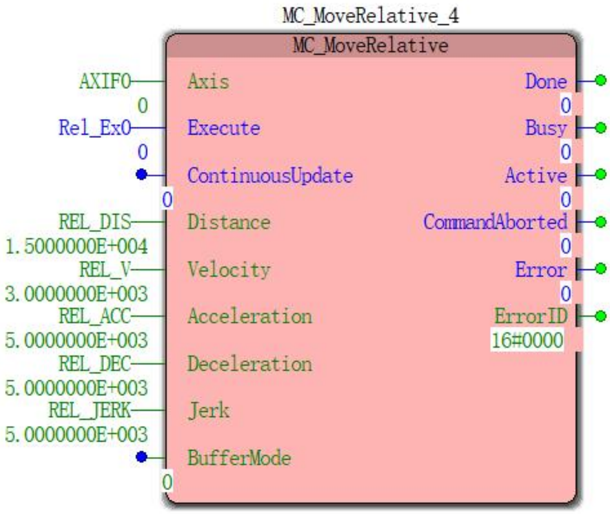
$$r/min=\frac{Velocity}{（Screw\_Lead 或 Disc\_Circumference）*\frac{Reductor\_Num}{Reductor\_Den}}*60$$

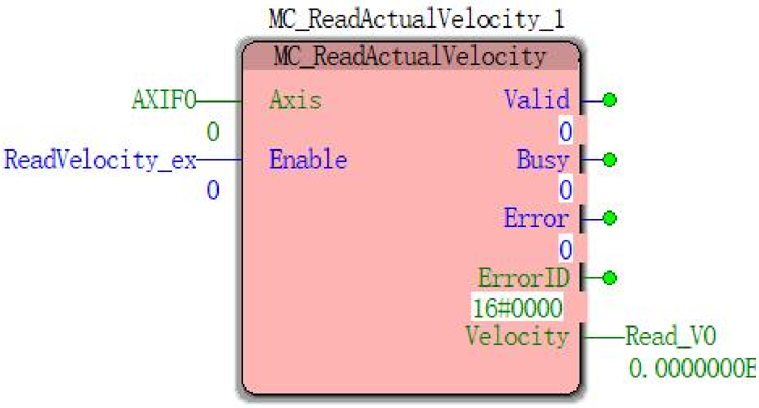
程序范例

执行 MC_MoveRelative， MC_ReadActualVelocity 读取被控轴实时速度。

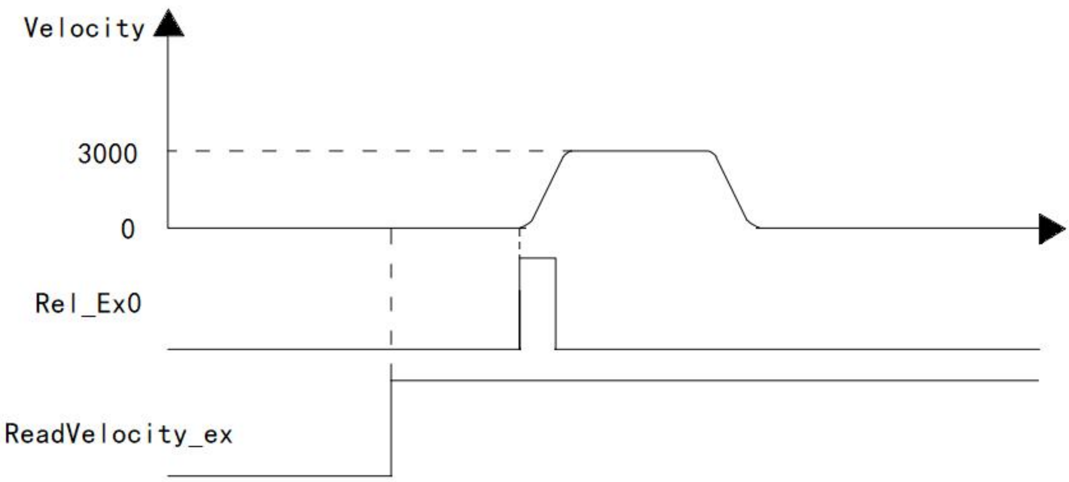
1、 变量和程序

变量名	数据类型	初始值
MC_MoveRelative_4	MC_MoveRelative	-
AXIF0	USINT	0
Rel_Ex0	BOOL	FALSE
REL_DIS	LREAL	15000.0
REL_V	LREAL	3000.0
REL_ACC	LREAL	5000.0
REL_DEC	LREAL	5000.0
REL_JERK	LREAL	5000.0
MC_ReadActualVelocity_1	MC_ReadActualVelocity	-
ReadVelocity_ex	BOOL	FALSE
Read_V0	LREAL	-


MC_MoveRelative_4
MC_MoveRelative
AXIF0
0
Rel_Ex0
0
0
REL_DIS
1.5000000E+004
REL_V
3.0000000E+003
REL_ACC
5.0000000E+003
REL_DEC
5.0000000E+003
REL_JERK
5.0000000E+003
0
Axis
Execute
ContinuousUpdate
Distance
Velocity
Acceleration
Deceleration
Jerk
BufferMode
Done
0
Busy
0
Active
0
CommandAborted
0
Error
0
ErrorID
16#0000

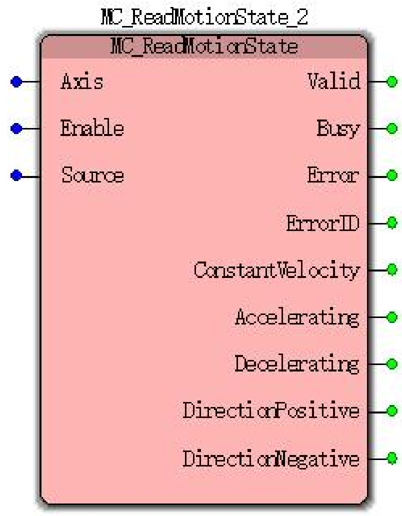

MC_ReadActualVelocity_1
MC_ReadActualVelocity
AXIF0
0
ReadVelocity_ex
0
Axis
Enable
Valid
0
Busy
0
Error
0
ErrorID
16#0000
Velocity
Read_V0
0.0000000E

2、 运动曲线和时序图


Velocity
3000
0
Rel_Ex0
ReadVelocity_ex

11.4.14 MC_ReadMotionState（读取轴运动状态指令）

FB/FC	说明	适用机种
FB	此指令用于读取被控轴的运动状态	VEC-VA-MP-005-MA



➢ **输入参数**

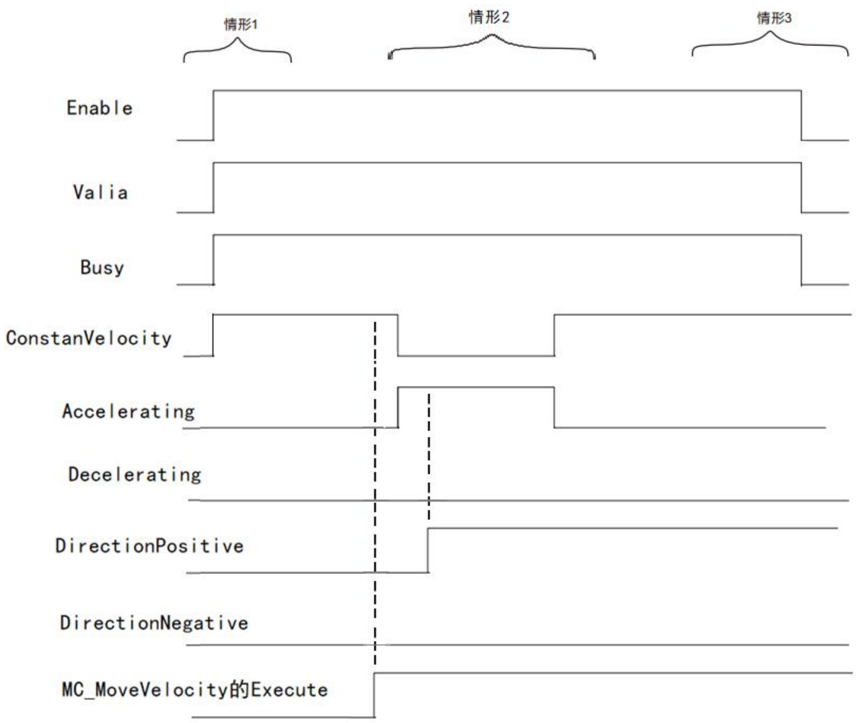
名称	功能	数据类型	设定范围（缺省值）	生效时机
Axis（轴号）	设定指令欲控制的轴	USINT	**模拟量/脉冲：** 0~4（实轴） 5~11（虚轴） **CANopen 模式：** 0~15 (实轴/虚轴) (0)	Enable 为 TRUE 时
Enable（执行位）	当 Enable 为 TRUE 时，指令执行	BOOL	TRUE 或 FALSE	-
Source（保留）	保留	INT	-	-

➢ **输出参数**

名称	功能	数据类型	输出范围
Valid（输出有效）	该参数输出为 TRUE 时表示指令输出有效	BOOL	TRUE
Busy（执行中）	该参数输出为 TRUE 时表示指令正在执行中	BOOL	TRUE 或 FALSE
Error（错误）	该指令输出为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE
ErrorID（错误代码）	执行指令出错时的错误代码	WORD	-
ConstanVelocity	该输出参数为 TRUE 时表示轴	BOOL	TRUE 或 FALSE

(匀速状态)	正在做匀速运动		
Accelerating （加速状态）	该输出参数为 TRUE 时表示轴速度的绝对值增大	BOOL	TRUE 或 FALSE
Declerating （减速状态）	该输出参数为 TRUE 时表示轴速度的绝对值减小	BOOL	TRUE 或 FALSE
DirectionPositive （轴正转）	该输出参数为 TRUE 时表示轴当前位置增大	BOOL	TRUE 或 FALSE
DirectionNegative （轴反转）	该输出参数为 TRUE 时表示轴当前位置减小	BOOL	TRUE 或 FALSE

➢ **输出参数时序变化图**



情形 1：在被控轴处于 Standstill 状态下，当 Enable 由 FALSE 变为 TRUE 时，Vaild 和 Busy 同时变为 TRUE，ConstantVelocity、Accelerating、Decelerating、DirectionPositive、DirectionNegative 引脚的输出结果根据轴状态变为 TRUE 或 FALSE。

情形 2：执行速度指令，被控轴加速，Accelerating 输出为 TRUE，ConstantVelocity 输出变为 FALSE，被控轴当前位置增大时 DirectionPositive 输出为 TRUE。

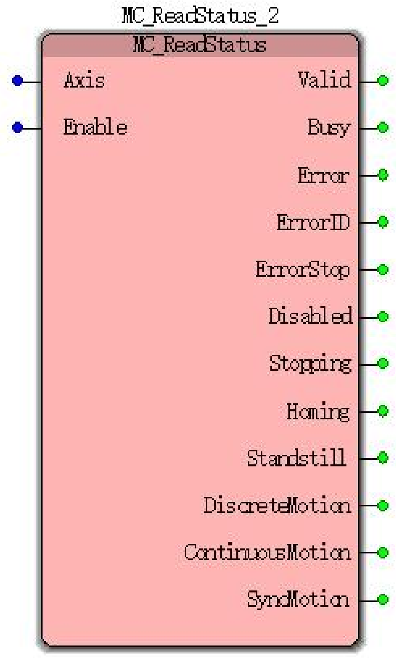
情形 3： 当 Enable 由 TRUE 变为 FALSE 时， Vaild 和 Busy 同时变为 FALSE，ConstantVelocity、Accelerating、Decelerating、DirectionPositive、DirectionNegative 引脚的输出结果保持 Enable 为 TRUE 时的状态不变。

➢ **功能说明**

此指令用于读取伺服轴当前的运动状态。伺服轴的运动状态包括：匀速运动、加速运动或减速运动，以及正转或反转。

11.4.15 MC_ReadStatus（读轴状态）

FB/FC	说明	适用机种
FB	此指令用于读取被控轴的状态信息	VEC-VA-MP-005-MA



➢ **输入参数**

名称	功能	数据类型	设定范围（缺省值）	生效时机
Axis（轴号）	设定指令欲控制的轴	USINT	**模拟量/脉冲：** 0~4（实轴） 5~11（虚轴） **CANopen 模式：** 0~15 (实轴/虚轴) (0)	Enable 为 TRUE 时
Enable（执行位）	当 Enable 为 TRUE 时，指令执行	BOOL	TRUE 或 FALSE	-

➢ **输出参数**

名称	功能	数据类型	输出范围
Valid（输出有效）	该参数输出为 TRUE 时表示指令输出有效	BOOL	TRUE
Busy（执行中）	该参数输出为 TRUE 时表示指令正在执行中	BOOL	TRUE 或 FALSE
Error（错误）	该指令输出为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE
ErrorID（错误代码）	执行指令出错时的错误代码	WORD	-
ErrorStop	请参考状态机说明 11.3.6 状态	BOOL	TRUE 或 FALSE

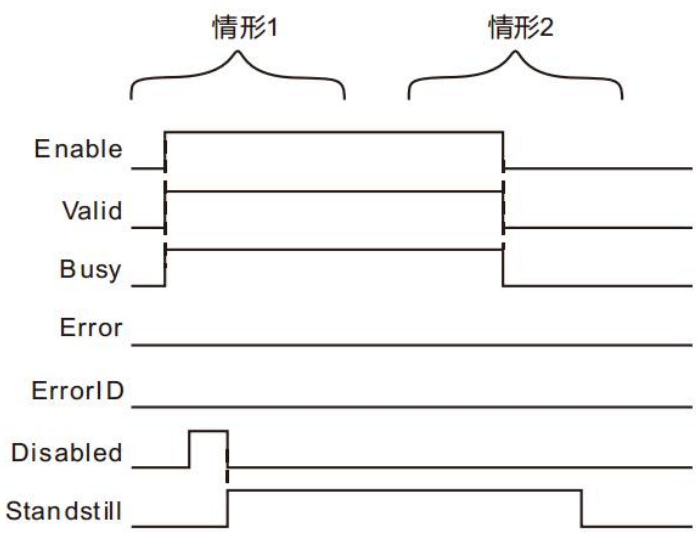
(异常停止)	机		
Disabled （未执行）		BOOL	TRUE 或 FALSE
Stopping （正常停止）		BOOL	TRUE 或 FALSE
Homing （原点回归）		BOOL	TRUE 或 FALSE
Standstill （准备执行）		BOOL	TRUE 或 FALSE
DiscreteMotion （离散运动）		BOOL	TRUE 或 FALSE
ContinousMotion （连续运动）		BOOL	TRUE 或 FALSE
SyncMotion （同步运动）		BOOL	TRUE 或 FALSE

说明：

1、 该指令在 Enable 为 TRUE 时执行，读取轴状态。

2、该指令在 Enable 由 TRUE 变为 FALSE 时，Vaild、Busy 变为 FALSE，ErrorStop、Disabled、Stopping、Homing、Standstill、DiscreteMotion、ContinuousMotion 和 SyncMotion 的输出结果保持 Enable 为 TRUE 时的状态不变。

- **输出参数变化时序图**



情形 1：当 Enable 由 FALSE 变为 TRUE 时，Vaild 和 Busy 同时变为 TRUE， ErrorStop、Disabled、Stopping、Homing、Standstill、DiscreteMotion、ContinuousMotion 和 SyncMotion 根据轴状态变为 TRUE 或 FALSE。

情形 2：当 Enable 由 TRUE 变为 FALSE 时，Vaild 和 Busy 同时变为 FALSE，Disabled、Stopping、Homing、Standstill、DiscreteMotion、ContinuousMotion 和 SyncMotion 引脚的输出结果

保持 Enable 为 TRUE 时的状态不变。

➤ **功能说明**

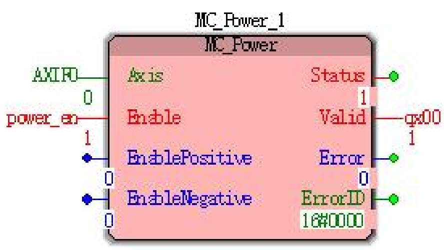
此指令用于读取被控轴的轴状态信息，关于轴状态的详细说明请参考状态机说明

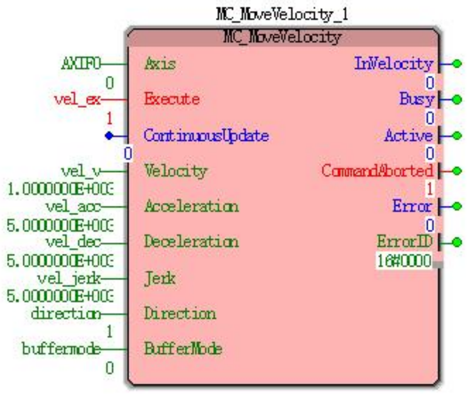
程序范例

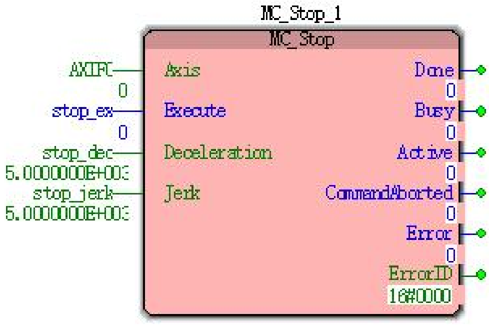
MC_ReadStatus 指令执行范例如下所示：

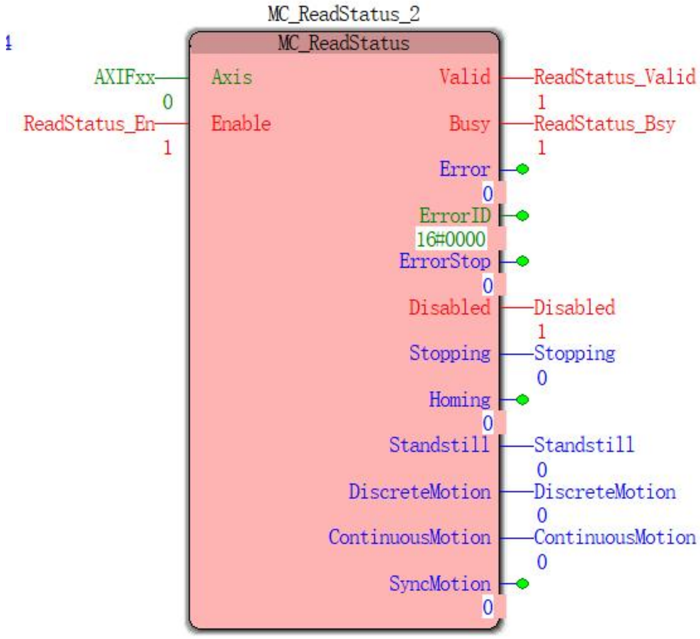
1、 变量和程序

变量名	数据类型	初始值
MC_Power_1	MC_Power	--
AXIF0	USINT	0
Power_en	BOOL	FALSE
MC_MoveVelocity_1	MC_MoveVelocity	-
Vel_ex	BOOL	FALSE
Vel_v	LREAL	1000.0
Vel_acc	LREAL	5000.0
Vel_dec	LREAL	5000.0
Vel_jerk	LREAL	5000.0
direction	INT	1
buffermode	INT	0
MC_Stop_1	MC_Stop	-
stop_ex	BOOL	FALSE
stop_dec	LREAL	5000.0
stop_jerk	LREAL	5000.0
MC_ReadStatus_2	MC_ReadStatus	-
AXIFxx	USINT	0
ReadStatus_En	BOOL	FALSE
ReadStatus_Valid	BOOL	FALSE
ReadStatus_Bsy	BOOL	FALSE
Disabled	BOOL	FALSE
Stopping	BOOL	FALSE
Standstill	BOOL	FALSE
DiscreteMotion	BOOL	FALSE
ContinuousMotion	BOOL	FALSE

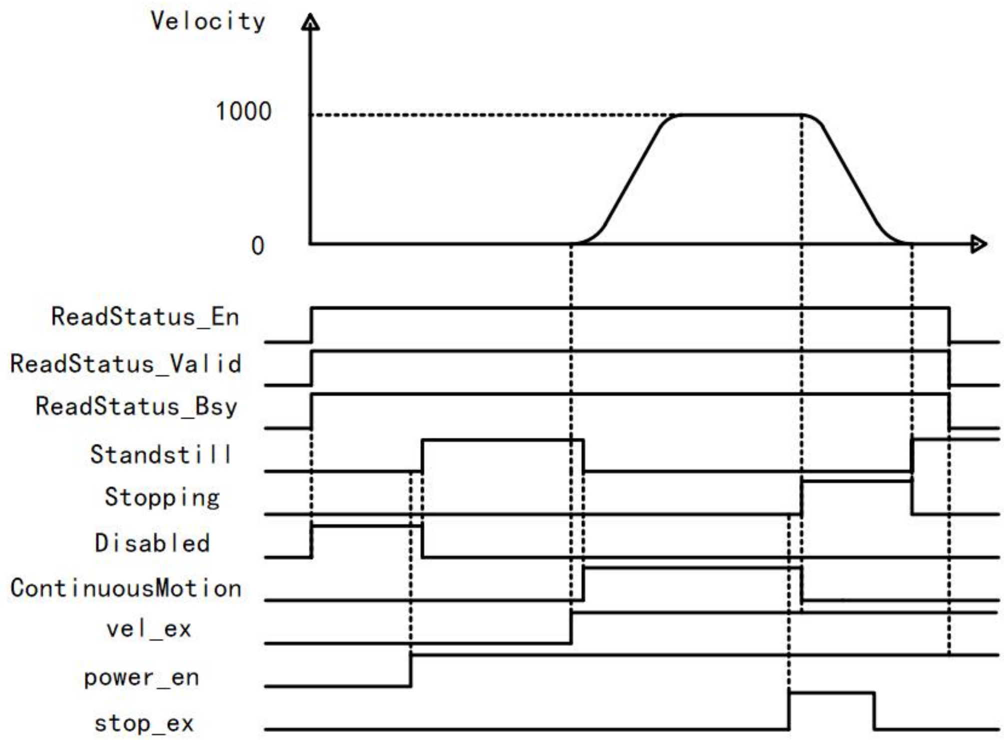



MC_MoveVelocity_1
MC_MoveVelocity
AXIF0
Axis
InVelocity
vel_ex
Execute
Busy
ContinuousUpdate
Active
vel_v
Velocity
CommandAborted
vel_acc
Acceleration
Error
vel_dec
Deceleration
ErrorID
16#0000
vel_jerk
Jerk
direction
Direction
buffermode
BufferMode


MC_Stop_1
MC_Stop
AXIF0
Axis
Done
stop_ex
Execute
Busy
stop_dec
Deceleration
Active
stop_jerk
Jerk
CommandAborted
Error
ErrorID
16#0000


MC_ReadStatus_2
MC_ReadStatus
AXIFxx
Axis
Valid
ReadStatus_Valid
ReadStatus_En
Enable
Busy
ReadStatus_Bsy
Error
ErrorID
16#0000
ErrorStop
Disabled
Disabled
Stopping
Stopping
Homing
Standstill
Standstill
DiscreteMotion
DiscreteMotion
ContinuousMotion
ContinuousMotion
SyncMotion

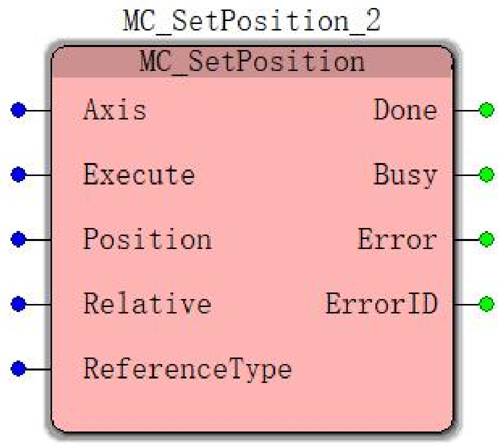
2、运动曲线及时序图



- MC_Power 指令的 Enable 由 FALSE 变为 TRUE，一个周期后 MC_ReadStatus 指令的 Disabled 由 TRUE 变为 FALSE，同时 MC_ReadStatus 指令的 Standstill 由 FALSE 变为 TRUE（即状态机由 Disabled 变为 Standstill）
- 速度指令的 Execute 由 FALSE 变为 TRUE，一个周期后 MC_ReadStatus 指令的 Standstill 由 TRUE 变为 FALSE，同时 MC_ReadStatus 指令的 ContinuousMotion 由 FALSE 变为 TRUE(即状态机由 Standstill 变为 ContinuousMotion)
- MC_Stop 指令的 Execute 由 FALSE 变为 TRUE，一个周期后 MC_ReadStatus 指令的 ContinuousMotion 由 TRUE 变为 FALSE，同时 MC_ReadStatus 指令的 Stopping 由 FALSE 变为 TRUE（即状态机由 ContinuousMotion 变为 Stopping）

11.4.16 MC_SetPosition（位置设置指令）

FB/FC	说明	适用机种
FB	此指令用于将轴的位置值设定成指定值，且不引起该轴产生实际运动。	VEC-VA-MP-005-MA



➢ **输入参数**

名称	功能	数据类型	设定范围（缺省值）	生效时机
Axis（轴号）	设定指令欲控制的轴	USINT	**模拟量/脉冲：** 0~4（实轴） 5~11（虚轴） **CANopen 模式：** 0~15 (实轴/虚轴) (0)	Execute 由 FALSE 为 TRUE 时
Execute（执行位）	当 Execute 由 FALSE 变为 TRUE 时，指令开始执行	BOOL	TRUE 或 FALSE（FALSE）	
Position（位置）	设定的目标位置（单位：单元）	LREAL	正数、负数、零（0）	Execute 由 FALSE 为 TRUE 时
Relative（相对模式）	设定目标位置与当前位置为相对模式或绝对模式	BOOL TRUE:Relative 相对模式 FALSE:Absolute 绝对模式	TRUE 或 FALSE（FALSE）	Execute 由 FALSE 为 TRUE 时
ReferenceType（参考位置的类型）	设定参考位置的类型	INT 0：命令位置 1：实际位置	0、1（0）	Execute 由 FALSE 为 TRUE 时

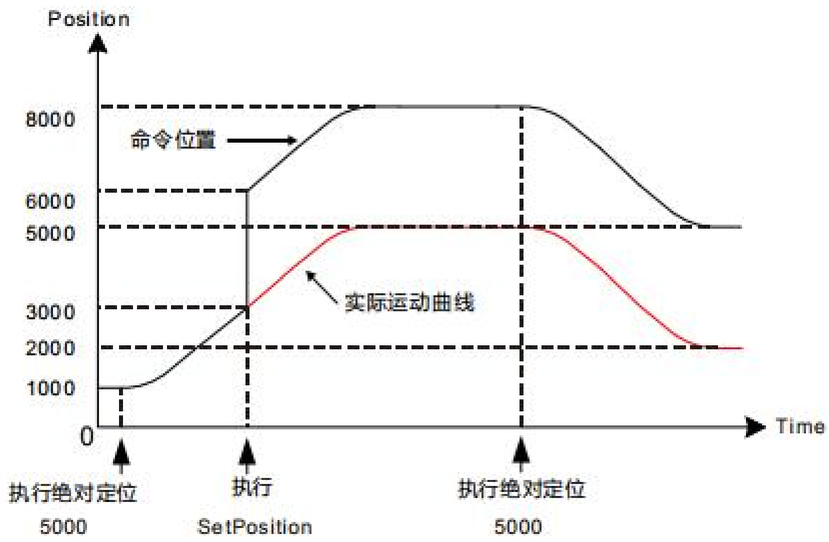
➢ **输出参数**

名称	功能	数据类型	输出范围

Done	该参数输出为 TRUE 时表示指令执行完成	BOOL	TRUE 或 FALSE
Busy（执行中）	该参数输出为 TRUE 时表示指令正在执行中	BOOL	TRUE 或 FALSE
Error（错误）	该指令输出为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE
ErrorID（错误代码）	执行指令出错时的错误代码	WORD	-

- **功能说明**

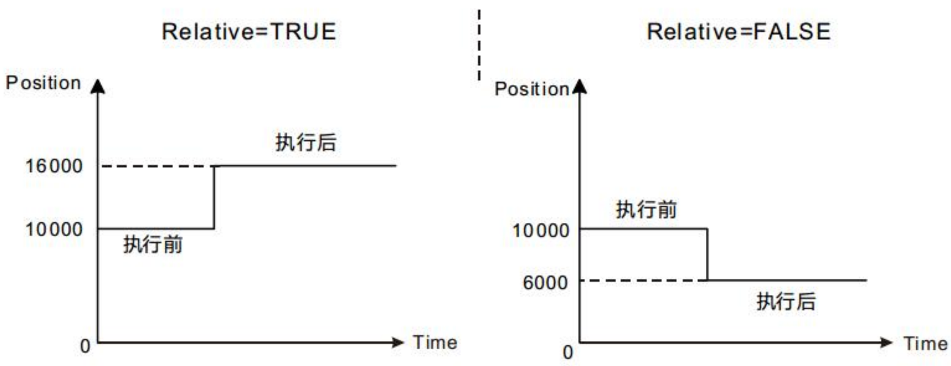
此指令用于将轴的位置值设定成指定值，且不引起该轴产生实际运动。此指令的执行对正在执行的运动不会产生实际影响，但对此指令执行完成后再开始执行的指令的实际执行效果有影响，如下图所示：



- **Position 与 Relative**

输入参数 Position、Relative 与指令开始执行时刻的轴位置（本文使用“参考位置”表示）共同决定欲设置的位置值。

输入参数 Relative 用于定义输入参数 Position 与参考位置之间的关系。当 Relative=TRUE 时，Position 与参考位置为相对关系，位置设置值=参考位置+Position；当 Relative=FALSE 时，Position 与参考位置为绝对关系，位置设置值=Position。如下图所示，指令执行时的参考位置为 10000，输入参数 Position 的值为 6000，当输入参数 Relative 为不同值时，对应的执行效果分别为左下图和右下图。



- **ReferenceType**

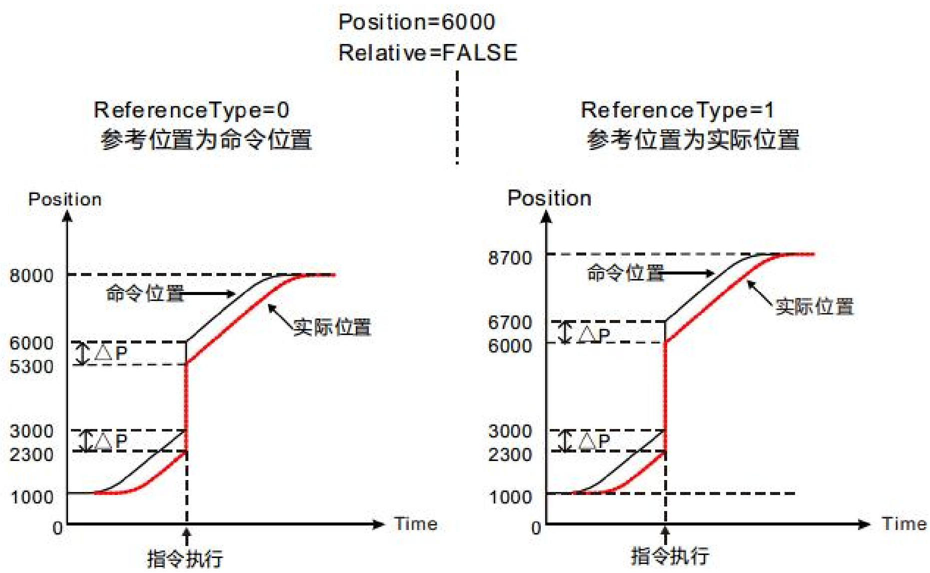
输入参数 ReferenceType 用于选择参考位置为命令位置或实际位置。ReferenceType=0 时，参考位置为该轴命令位置；ReferenceType=1 时，参考位置为该轴实际位置。

当参考位置选择为命令位置时，指令将根据当前的命令位置和 Position 计算出目标命令位置，并将命令位置的值修改为该目标位置的值；同时，该轴的实际位置也将随之变化，变化规律为：实际位置的变化量与命令位置的变化量相等，也即是命令位置与实际位置之间的差值在指令开始执行时刻和指令执行完成时刻保持不变。

参考位置选择为实际位置时的处理方式与参考位置选择为命令位置时的处理方式同理。

若执行 MC_SetPosition 指令时，轴处于静止状态，则参考位置分别选择命令位置和实际位置时的实际执行效果没有差别，原因在于轴静止时，其命令位置与实际位置之间不存在差值（差值为 0）；

若执行 MC_SetPosition 指令时，轴处于运动状态，其命令位置与实际位置之间存在差值（差值不为 0，由命令响应时间引起），则参考位置分别选择命令位置和实际位置时的实际执行效果存在差异，如下图范例所示：如下图曲线所示，轴正在作定位运动（目标位置为 5000）时执行 MC_SetPosition 指令（绝对模式，Position=6000），此时该轴的命令位置和实际位置分别为 3000 和 2300（差值△P =700）。若参考位置选择命令位置，则指令执行后，该轴的命令位置变为 6000，实际位置变为 5300（5300=6000-△P），如左下图所示；若参考位置选择实际位置，则指令执行后，该轴的实际位置变为 6000，命令位置变为 6700（6700=6000+△P），如右下图所示：



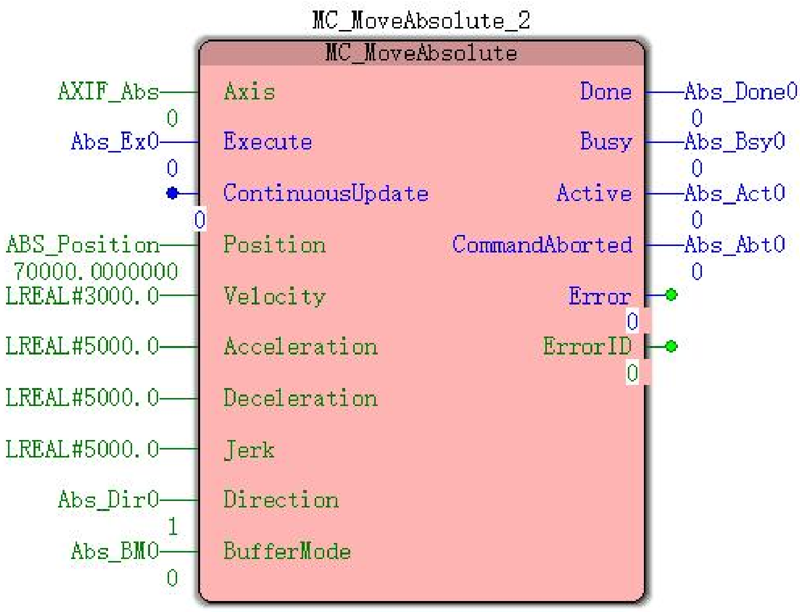
程序范例

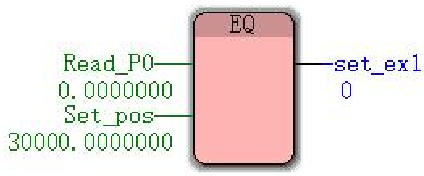
本例描述 MC_SetPosition 指令对正在执行的 MC_MoveAbsolute 执行的执行效果的影响：MC_SetPosition 对正在执行的 MC_MoveAbsolute 指令的实际执行效果无影响。

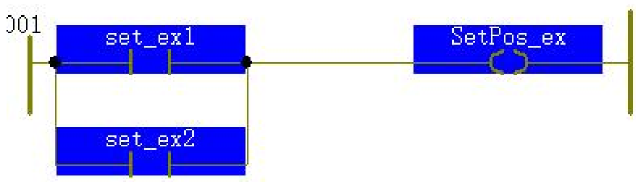
2、 变量和程序

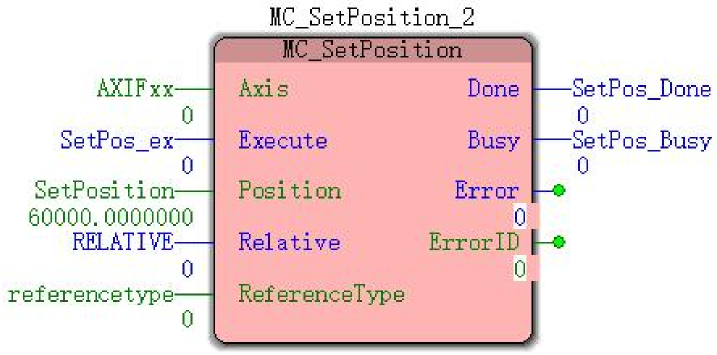
变量名	数据类型	初始值
MC_MoveAbsolute_2	MC_MoveAbsolute	--
AXIF_abs	USINT	0
Abs_Ex0	BOOL	FALSE
Abs_Position	LREAL	70000

Abs_BM0	INT	
Abs_Done0	BOOL	
Abs_Bsy0	BOOL	
Abs_Act0	BOOL	
Abs_Abt0	BOOL	
MC_SetPosition_2	MC_SetPosition	
AXIFxx	USINT	0
SetPos_ex	BOOL	FALSE
SetPosition	LREAL	60000
RELATIVE	BOOL	0
referencetype	INT	
SetPos_Done	BOOL	
SetPos_Busy	BOOL	

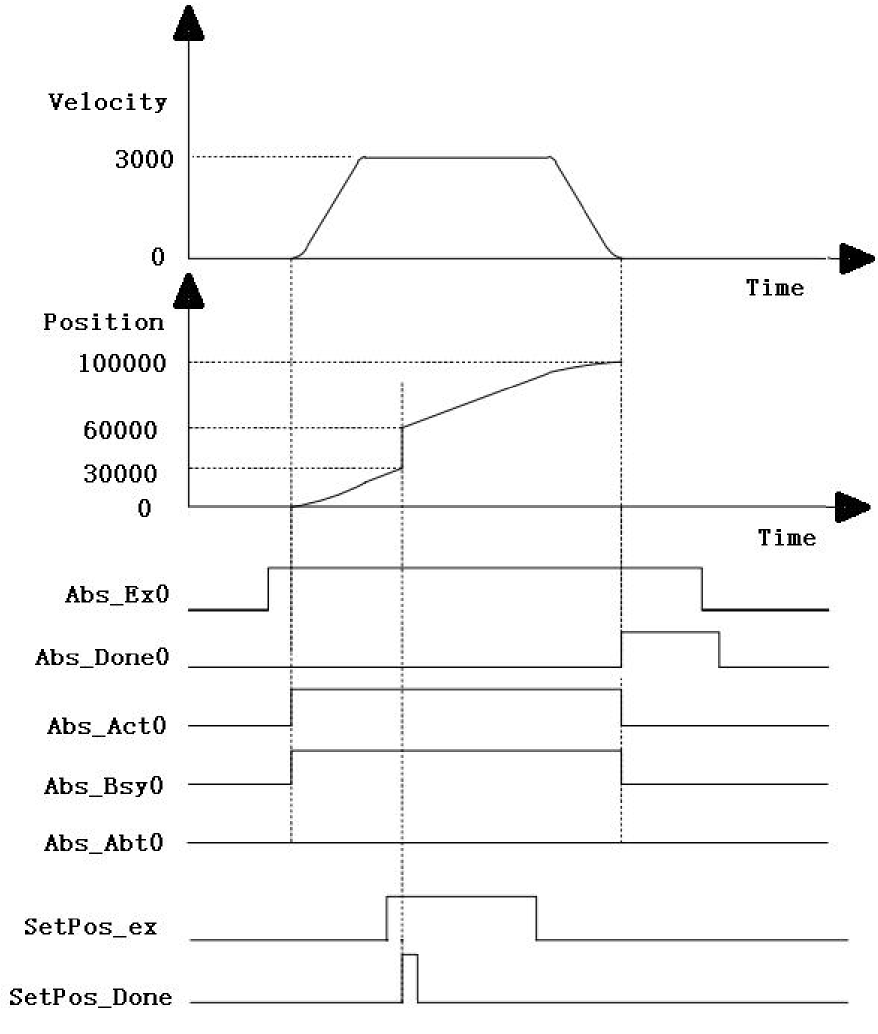








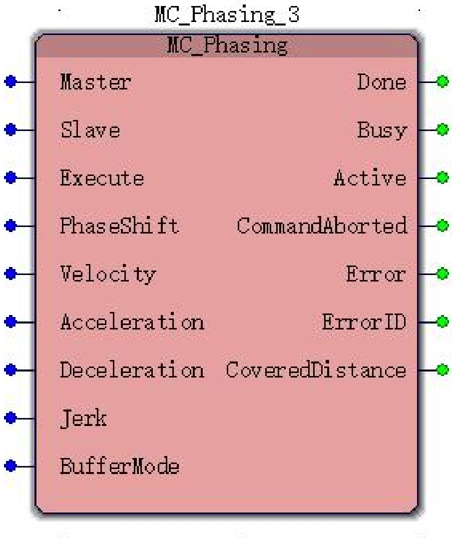
2、运动曲线及时序图



- 当 Abs_Ex0 由 FALSE 变为 TRUE 时，MC_MoveAbsolute 指令开始执行，当当前位置大于 30000 后 MC_SetPosition 指令执行。
- MC_SetPosition 指令开始执行时的命令位置为 30000，其执行完成后的命令位置为 60000，MC_MoveAbsolute 指令执行完成时的位置为 100000。
- 通过上图中的速度变化曲线可以看出：MC_SetPosition 指令对正在执行的 MC_MoveAbsolute 的实际执行效果不会产生影响

11.4.17 MC_Phasing（偏移主轴指令）

FB/FC	说明	适用机种
FB	此指令用于给主轴附加一段不影响主轴运动的位置偏移	VEC-VA-MP-005-MA



➢ **输入参数**

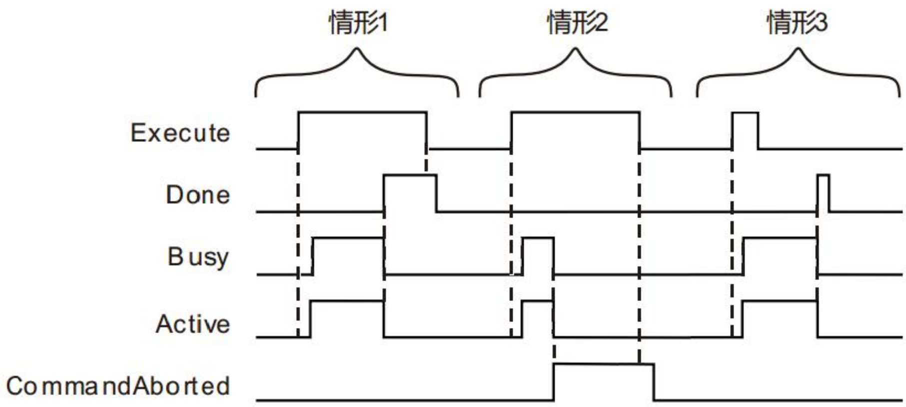
名称	功能	数据类型	设定范围（缺省值）	生效时机
Master（主轴）	设定指令欲控制的主轴	USINT	**模拟量/脉冲：** 0~4（实轴） 5~11（虚轴） **CANopen 模式：** 0~15 (实轴/虚轴) (0)	Exexcute 由 FALSE 变为 TRUE
Slave（从轴）	设定指令欲控制的从轴	USINT	0~4 实轴 5~11 虚轴 (0)	Exexcute 由 FALSE 变为 TRUE
Execute（执行位）	当 Execute 由 FALSE 变为 TRUE 时，执行该指令	BOOL	TRUE 或 FALSE	Exexcute 由 FALSE 变为 TRUE
PhaseShift（偏移量）	设定主轴的位置偏移量 （单位：单元）	LREAL	正数、负数、零（0）	Exexcute 由 FALSE 变为 TRUE
Velocity（速度）	设定偏移主轴速度	LREAL	正数 （不可缺省）	Exexcute 由 FALSE 变为 TRUE
Acceleration（加速度）	设定偏移主轴加速度	LREAL	正数 （不可缺省）	Exexcute 由 FALSE 变为 TRUE
Decleration	设定偏移主轴减速度	LREAL	正数	Exexcute 由 FALSE

（减速度）			（不可缺省）	变为 TRUE
Jerk （加/减速度的变化率）	设定偏移主轴加速度/减速度的变化率	LREAL	正数 （不可缺省）	Exexcute 由 FALSE 变为 TRUE
BufferMode （交接模式）	保留	-	-	-

➢ **输出参数**

名称	功能	数据类型	输出范围
Done	该参数输出为 TRUE 时表示指令执行完成	BOOL	TRUE 或 FALSE
Busy（执行中）	该参数输出为 TRUE 时表示指令正在执行中	BOOL	TRUE 或 FALSE
Active（控制中）	该参数输出为 TRUE 时表示指令正在控制轴	BOOL	TRUE 或 FALSE
CommandAborted（中断）	该参数输出为 TRUE 时表示指令被中断	BOOL	TRUE 或 FALSE
Error（错误）	该指令输出为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE
ErrorID（错误代码）	执行指令出错时的错误代码	WORD	-
CoveredDistance （累计偏移主轴距离）	该指令自启动后所累计追加的距离，此参数输出范围极限按 int 型溢出处理（如果累计追加距离超过 2^32 时，此参数会取反）	LREAL	-

➢ **输出参数时序变化图**



情形 1：当 Execute 由 FALSE 变为 TRUE 时，一个周期后 Buys 和 Active 同时变为 TRUE；当主轴完成偏移后，Done 变为 TRUE，同时 Busy 和 Active 变为 FALSE，在 Execute 由 TRUE 变为 FALSE 一个周期后，Done 变为 FALSE。

情形 2：当 Execute 为 TRUE 时，该指令被其它指令中断后，CommandAborted 变为 TRUE，

同时 Busy 和 Active 变为 FALSE；当 Execute 由 TRUE 变为 FALSE 时，一个周期后 CommandAborted 变为 FALSE。

情形 3： 在指令执行过程中，Execute 由 TRUE 变为 FALSE 后，指令执行完成时，Done 变为 TRUE，同时 Busy 和 Active 变为 FALSE，且一个周期后，Done 变为 FALSE。

➢ **功能说明**

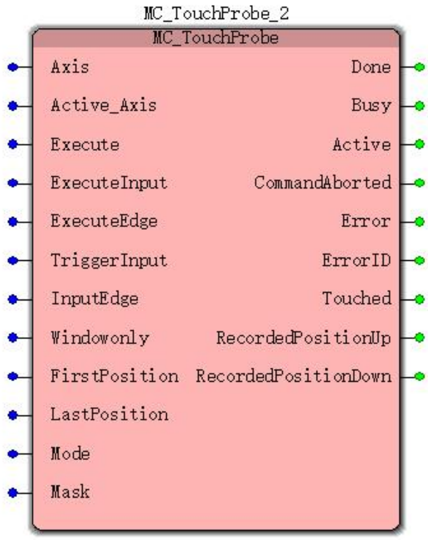
- 该指令用于给主轴叠加一段按设定距离、速度、加速度/减速度、减速度的变化率进行运动的假想位移，它不影响主轴的实际运动，而从轴获取到的主轴位置会在物理基础上偏移，从而导致从轴的跟随位置有所偏移。

- MC_Phasing 可作用的多轴指令如下：

MC_GearIn（电子齿轮）	NS_MC_RotaryCutIn（旋切指令）
MC_CamIn（电子凸轮）	NS_MC_SpecialCamin（特殊凸轮）
NS_MC_SpecialCombineAxes (特殊双主轴耦合)	

11.4.18 MC_TouchProbe（位置捕获指令）

FB/FC	说明	适用机种
FB	此指令用于轴的位置捕获	VEC-VA-MP-005-MA



➢ **输入参数**

名称	功能	数据类型	设定范围（缺省值）	生效时机
Axis（轴号）	在模式 0、1、2 下用于与高速计数器配对用，模式 3、4 用作计数轴号	USINT	**模拟量/脉冲：** 0~4（实轴） 5~11（虚轴） **CANopen 模式：** 0~15 (实轴/虚轴) (0)	Execute 由 FALSE 为 TRUE 时
Active_Axis (硬件轴号)	设定位置捕获来源的硬件轴号	USINT	0~4 实轴	Execute 由 FALSE 为 TRUE 时
Execute（执行位）	当 Execute 由 FASLE 变为 TRUE 时，执行该指令。	BOOL	TRUE 或 FALSE	
ExecuteInput (触发允许位)	输入点 I0~I7、I10~I17 中的一个为位置捕获的触发位，引脚输入值 0~7 对应输入点 I0~I7、8~15 对应输入点 I10~I17。该引脚在模式（Mode）等于 2 时有效。	WORD	0~15	Execute 由 FALSE 为 TRUE 时
ExecuteEdge（信号边沿）	FALSE 时，选择输入 DI 信号为下降沿，TRUE 时，选择输入 DI 信号为上升沿；	BOOL	TRUE 或 FALSE	Execute 由 FALSE 为 TRUE 时
TriggerInput (触发位)	输入点 I0~I7、I10~I17 中的一个为位置捕获的触发位，引脚输入	WORD	0~15 (0)	Execute 由 FALSE 为

	值 0~7 对应输入点 I0~I7、8~15 对应输入点 I10~I17。			TRUE 时
InputEdge（信号边沿）	设定信号触发边沿 0：下降沿 1：上升沿	BOOL	TRUE 或 FALSE	Execute 由 FALSE 为 TRUE 时
Windowly	保留	-	-	-
FirstPosition	保留	-	-	-
LastPosition	保留	-	-	-
Mode（模式）	位置捕获模式选择，详情见下模式说明	INT	0~6（0）	Execute 由 FALSE 为 TRUE 时
Mask	保留	-	-	-

➢ **输出参数**

名称	功能	数据类型	输出范围
Done	该参数输出为 TRUE 时表示指令执行完成，该参数仅在 Mode=0、3 时有效。	BOOL	TRUE 或 FALSE
Busy（执行中）	该参数输出为 TRUE 时表示指令正在执行中。	BOOL	TRUE 或 FALSE
Active（控制中）	该参数输出为 TRUE 时表示指令正在控制轴	BOOL	TRUE 或 FALSE
CommandAborted（中断）	该参数输出为 TRUE 时表示指令被中断	BOOL	TRUE 或 FALSE
Error（错误）	该指令输出为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE
ErrorID（错误代码）	执行指令出错时的错误代码	WORD	-
Touched（连续模式循环计数）	当选择的锁存位置模式为 1、2、4 时，每锁存一次位置，Touched 加 1，Toucheded0~3 循环计数。	INT	0~3
RecordedPositionUp（上升沿锁存位置）	当 InputEdge=TRUE 时，锁存位置存放于此。 单位：脉冲	LREAL	正数、负数、0
RecordedPositionDown（下降沿锁存位置）	当 InputEdge=FALSE 时，锁存位置存放于此。 单位：脉冲	LREAL	正数、负数、0

➢ **模式说明**

模式	位置捕捉触发方式
Mode=0	该模式为高速计数器单次锁存，在该模式下执行位置捕获指令： ①TriggerInput 只有在首次触发有效。 ②需要与 NS_CC_Counter 高数计数指令配合使用。Active_Axis 不起作用，输入参数 Axis 需要与 NS_CC_Counter 指令的输入参数 AXIF_no 一致。
Mode=1	该模式为高速计数器连续锁存，在该模式下执行位置捕获指令： ①每触发一次 TriggerInput 进行一次位置锁存的更新。 ②需要与 NS_CC_Counter 高数计数指令配合使用。Active_Axis 不起作用，输入参数 Axis 需要与 NS_CC_Counter 指令的输入参数 AXIF_no 一致。
Mode=2	该模式为高速计数器特殊连续锁存，在该模式下执行位置捕获指令： ①每次触发 TriggerInput 进行位置锁存的更新前，需要触发一次 ExecuteInput。 ②需要与 NS_CC_Counter 高数计数指令配合使用。在该模式下，Active_Axis 不起作用，输入参数 Axis 需要与 NS_CC_Counter 指令的输入参数 AXIF_no 一致。
Mode=3	该模式为编码器单次锁存。在该模式下，执行位置捕获指令： ①TriggerInput 只有在首次触发有效。 ②Axis 为锁存位置寄存轴号，Active_Axis 为实际编码器脉冲来源轴。
Mode=4	该模式为编码器连续锁存。在该模式下，执行位置捕获指令： ①每触发一次 TriggerInput 进行一次位置锁存的更新。 ②Axis 为锁存位置寄存轴号，Active_Axis 为实际编码器脉冲来源轴。
Mode=5	该模式为 CANopen 专用的单次锁存模式，需要配合伺服锁存外部 DI 信号来使用，在该模式下执行位置捕获指令： ①TriggerInput 只有在首次触发有效。 ②Axis 为锁存位置寄存轴号，Active_Axis 不起作用。
Mode=6	该模式为 CANopen 专用的连续锁存模式，需要配合伺服锁存外部 DI 信号来使用，在该模式下执行位置捕获指令： ①每触发一次 TriggerInput 进行一次位置锁存的更新。 ②Axis 为锁存位置寄存轴号，Active_Axis 不起作用。

➢ **功能说明**

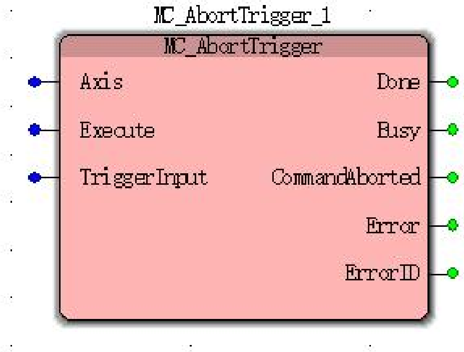
- 位置捕获指令捕获到的位置（RecordedPositionUp/ RecordedPositionDown） 是从轴/主轴伺服编码器的反馈脉冲数；
- 本指令属于高速位置捕捉指令，底层硬件计数，不受扫描周期影响。
- 在模式 0、1、2 下，该指令需要配合 NS_CC_Counter（高速计数器指令）使用，捕捉的位置才是伺服编码器反馈 A/B 脉冲的值，单独使用没有意义。
- 使用完连续锁存模式后，需要更换输入条件重新触发时，必须先用 MC_AbortTrigger(位置捕获打断指令)，退出当前的连续锁存模式，才能再进行触发；
- 在 CANopen 控制模式下，使用模式 5、6 时，需要下载我司的 CANopen 探针功能专用模版，同时配合 VEC 的 CANopen 型伺服驱动器使用。通过伺服自带的探针功能，锁存外部 DI 信号发生变化时的位置信息(编码器单位)，再传输到主站进行输出。
  ① VEC 支持 2 个探针同时使能，可同时记录每个探针信号的上升沿和下降沿对应的位置信息，即可同时锁存 4 个位置信息（上升沿和下降沿通过 InputEdge 切换）；
  ② 探针 1 选择 DI8 作为探针信号，探针 2 选择 DI9 作为探针信号，此处 DI8 和 DI9

指的是伺服上的 DI；

③ 当选择 DI8 为探针信号时，轴号 Axis=伺服节点号-1；当 DI9 位探针信号时，轴号 Axis=伺服节点号+31；

11.4.19 MC_AbortTrigger(位置捕获打断指令)

FB/FC	说明	适用机种
FB	此指令用于打断轴的位置捕获	VEC-VA-MP-005-MA



➢ **输入参数**

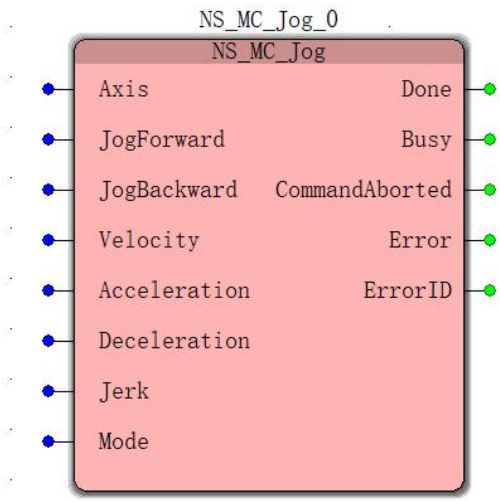
名称	功能	数据类型	设定范围 （缺省值）	生效时机
Axis （轴号）	该值设置与 MC_TouchProbe 指令的 Axis 值一致。用于指向需要被打断的计数轴号	USINT	**模拟量/脉冲：** 0~4（实轴） 5~11（虚轴） **CANopen 模式：** 0~15 (实轴/虚轴) (0)	Execute 由 FALSE 为 TRUE 时
Execute （执行位）	当 Execute 由 FALSE 为 TRUE 时，指令开始执行。	BOOL	TRUE 或 FALSE	-
Triggerinput	保留	-	-	-

➢ **输出参数**

名称	功能	数据类型	输出范围
Done	该参数输出为 TRUE 时表示指令执行完成	BOOL	TRUE 或 FALSE
Busy（执行中）	该参数输出为 TRUE 时表示指令正在执行中	BOOL	TRUE 或 FALSE
CommandAborted（中断）	该参数输出为 TRUE 时表示指令被中断	BOOL	TRUE 或 FALSE
Error（错误）	该指令输出为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE
ErrorID（错误代码）	执行指令出错时的错误代码	WORD	-

11.4.20 NS_MC_Jog(点动指令)

FB/FC	说明	适用机种
FB	此指令既可用于点动功能，也可做速度的叠加。	VEC-VA-MP-005-MA



➢ **输入参数**

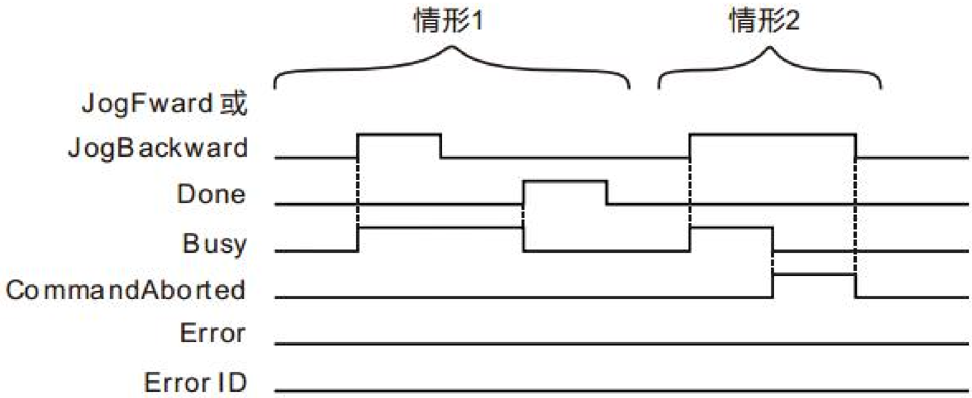
名称	功能	数据类型	设定范围（缺省值）	生效时机
Axis（轴号）	设定指令欲控制的轴	USINT	**模拟量/脉冲：** 0~4（实轴） 5~11（虚轴） **CANopen 模式：** 0~15 (实轴/虚轴) (0)	JogForward 或 JogBackward 为 TRUE 时
JogFoward (正向点动)	当 JogFoward 由 FALSE 变 TRUE 时，执行该指令	BOOL	TRUE 或 FALSE	-
JogBackward（反向点动）	当 JogBackward 由 FALSE 变 TRUE 时，执行该指令	BOOL	TRUE 或 FALSE	-
Velocity（速度）	设定的目标速度（单位：单元/S）	LREAL	正数（不可缺省）	JogForward 或 JogBackward 为 TRUE 时
Acceleration（加速度）	设定的目标加速度 (单位：单元/S^2)	LREAL	正数（不可缺省）	JogForward 或 JogBackward 为 TRUE 时
Deceleration（减速度）	设定的目标减速度 (单位：单元/S^2)	LREAL	正数（不可缺省）	JogForward 或 JogBackward 为 TRUE

				时
Jerk（加速度的变化率）	设定的目标加速度或减速度的变化率 (单位：单元/S³)	LREAL	正数（不可缺省）	JogForward 或 JogBackward 为 TRUE 时
Mode（模式）	点动模式选择： 0: 点动过程中更改点动速度，需重新触发 Excute 生效； 1: 点动过程中更改点动速度立即生效	INT	0 或 1	JogForward 或 JogBackward 为 TRUE 时

➢ **输出参数**

名称	功能	数据类型	输出范围
Done	该参数输出为 TRUE 时表示指令执行完成	BOOL	TRUE 或 FALSE
Busy（执行中）	该参数输出为 TRUE 时表示指令正在执行中	BOOL	TRUE 或 FALSE
CommandAborted（中断）	该参数输出为 TRUE 时表示指令被中断	BOOL	TRUE 或 FALSE
Error（错误）	该指令输出为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE
ErrorID（错误代码）	执行指令出错时的错误代码	WORD	-

➢ **输出参数时序变化图**



情形一：当 JogForward 或 JogBackward 由 FALSE 变为 TRUE 时，Busy 变为 TRUE。当点动停止，轴速度降为 0 后，Busy 变为 FALSE，同时 Done 变为 TRUE 并维持一个周期。
情形二：当 JogForward 或 JogBackward 由 FALSE 变为 TRUE，该指令被其它指令中断后，CommandAborted 变为 TRUE，同时 Busy 变为 FALSE；当 JogForward 或 JogBackward 由 TRUE 变为 FALSE 后，CommandAborted 变为 FALSE。

➢ **功能说明**

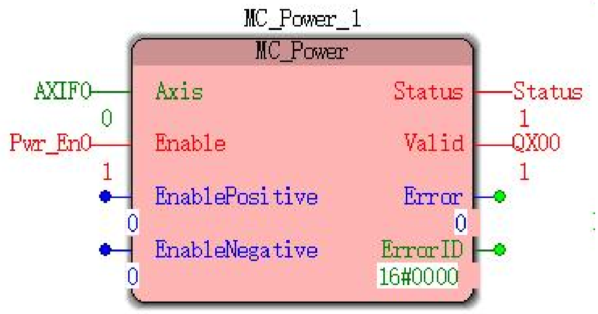
- 此指令用于给指定轴叠加一个指定的点动速度，JogForward 为 TRUE 则为被控轴叠加一个正向的点动速度，JogBackward 为 TRUE 则为被控轴叠加一个反向的点动速度。当叠加的点动速度速度降为 0 时，Done 为 TRUE 一个周期后变为 FALSE
- 该指令不影响当前状态机。

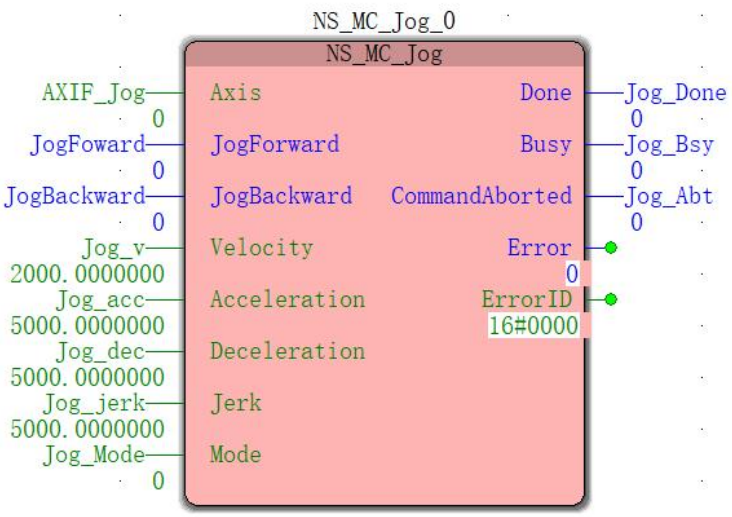
程序范例

NS_MC_Jog 指令单独执行时的范例如下：

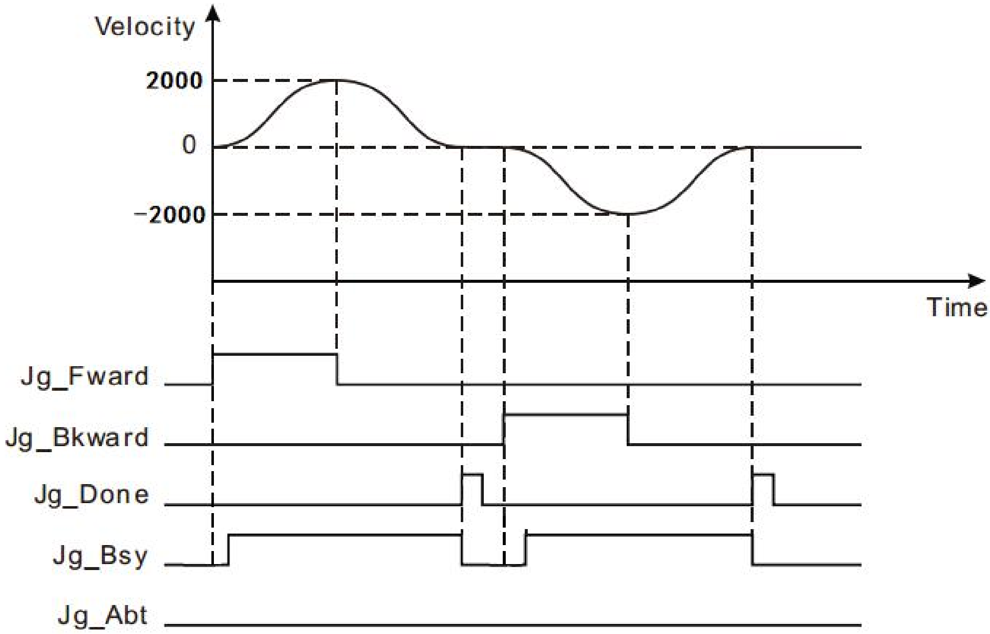
1、变量和程序

变量名	数据类型	初始值
NS_MC_Jog_2	NS_MC_Jog	-
JogForward	BOOL	FALSE
JogBackward	BOOL	FALSE
Jog_v	LREAL	2000.0
Jog_acc	LREAL	5000.0
Jog_dec	LREAL	5000.0
Jog_jerk	LREAL	5000.0
Jog_Done	BOOL	FALSE
Jog_Bsy	BOOL	FALSE
Jog_Abt	BOOL	FALSE





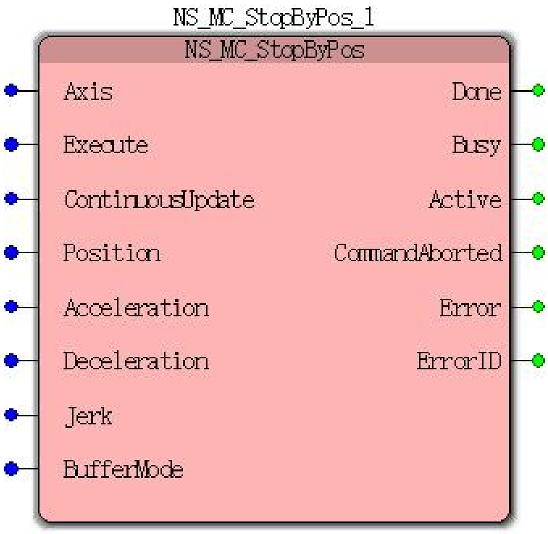
二、运动曲线和时序图



- 当 Jog_Forward 由 FALSE 变为 TRUE 时 NS_MC_Jog 指令开始执行，一个周期后 Jog_Bsy 由 FALSE 变为 TRUE，轴开始正向运转；将 Jog_Forward 由 TRUE 变为 FALSE，轴开始减速，当速度减为 0 时，Jog_Bsy 变为 FALSE，同时 Jog_Done 由 FALSE 变为 TRUE 一个周期后变为 FALSE。
- 当 Jog_Backward 由 FALSE 变为 TRUE 时，NS_MC_Jog 指令开始执行，一个周期后 Jog_Bsy 由 FALSE 变为 TRUE，轴开始反向运转；将 Jog_Backward 由 TRUE 变为 FALSE，轴开始减速，当速度减为 0 时，Jog_Bsy 位变为 FALSE，同时 Jog_Done 由 FALSE 变为 TRUE 一个周期后变为 FALSE。

11.4.21 NS_MC_StopByPos(指定模位置停止指令)

FB/FC	说明	适用机种
FB	此指令用于指定轴运转时停止在指定的模位置	VEC-VA-MP-005-MA



➢ **输入参数**

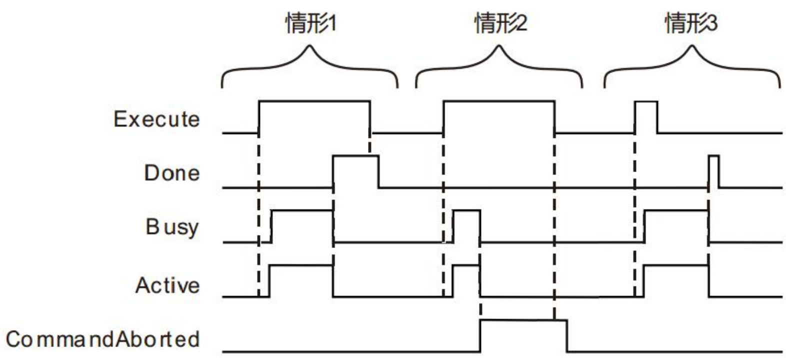
名称	功能	数据类型	设定范围（缺省值）	生效时机
Axis（轴号）	设定指令欲控制的轴	USINT	**模拟量/脉冲：** 0~4（实轴） 5~11（虚轴） **CANopen 模式：** 0~15 (实轴/虚轴) (0)	Exexcute 由 FALSE 变为 TRUE
Execute(执行位)	当 Execute 由 FALSE 变为 TRUE，指令开始执行	BOOL	FALSE 或 TRUE	-
ContinousUpdate	保留		保留	-
Position（位置）	设定的模位置	LREAL	0≤Position＜模	Exexcute 由 FALSE 变为 TRUE
Acceleration（加速度）	设定的目标加速度 (单位：单元/S^2)	LREAL	正数 （不可缺省）	Exexcute 由 FALSE 变为 TRUE
Deceleration（减速度）	设定的目标减速度 (单位：单元/S^2)	LREAL	正数 （不可缺省）	Exexcute 由 FALSE 变为 TRUE
Jerk（加速度的变化率）	设定的目标加速度或减速度的变化率 (单位：单元/S^3)	LREAL	正数 （不可缺省）	Exexcute 由 FALSE 变为 TRUE

BufferMode（交接模式）	设定两个指令之间交接模式 0:立即打断 1：等待	INT	正数（不可缺省）	Exexcute 由 FALSE 变为 TRUE

➢ **输出参数**

名称	功能	数据类型	输出范围
Done	该参数输出为 TRUE 时表示指令执行完成	BOOL	TRUE 或 FALSE
Busy（执行中）	该参数输出为 TRUE 时表示指令正在执行中	BOOL	TRUE 或 FALSE
Active（控制中）	该参数输出为 TRUE 时表示指令正在控制轴	BOOL	TRUE 或 FALSE
CommandAborted（中断）	该参数输出为 TRUE 时表示指令被中断	BOOL	TRUE 或 FALSE
Error（错误）	该指令输出为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE
ErrorID（错误代码）	执行指令出错时的错误代码	WORD	-

➢ **输出参数时序变化图**



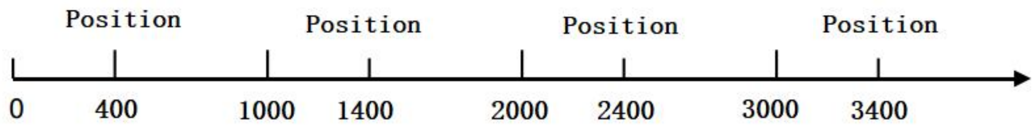
- **情形 1：**当 Execute 由 FALSE 变为 TRUE 时，一个周期后 Buys 和 Active 同时变为 TRUE；
- 当位置到达时，Done 变为 TRUE，同时 Busy 和 Active 变为 FALSE，在 Execute 由 TRUE 变为 FALSE 一个周期后，Done 变为 FALSE。
- **情形 2：**当 Execute 为 TRUE 时，该指令被其它指令中断后，CommandAborted 变为 TRUE，同时 Busy 和 Active 变为 FALSE；当 Execute 由 TRUE 变为 FALSE 时，一个周期后 CommandAborted 变为 FALSE。
- **情形 3：** 在指令执行过程中，Execute 由 TRUE 变为 FALSE 后，定位完成时，Done 变为 TRUE，同时 Busy 和 Active 变为 FALSE，且一个周期后，Done 变为 FALSE。

➢ **功能说明**

- 此指令用于使指定轴按设定的加、减速度、加速度的变化率停在指定的模位置上。Position 为指定模中的某一位置，该值要小于 MC_AXIS_REF 轴参数设定中 Modulo 的值，

将该指令的 Execute 从 FALSE 变为 TRUE 时，被控轴按照设定的加速度/减速度、加速度/减速度的变化率控制轴停止在 Position 设定的模位置，轴最终停止的位置为 Moulo 的整数倍+Position 的值

如下图所示，Modulo 为 1000，Position 为 400，可以通过该指令控制终端执行机构停止在 Position 指定的模位置，终端执行机构可能停在 400 单元、1400 单元、2400 单元、3400 单元……。

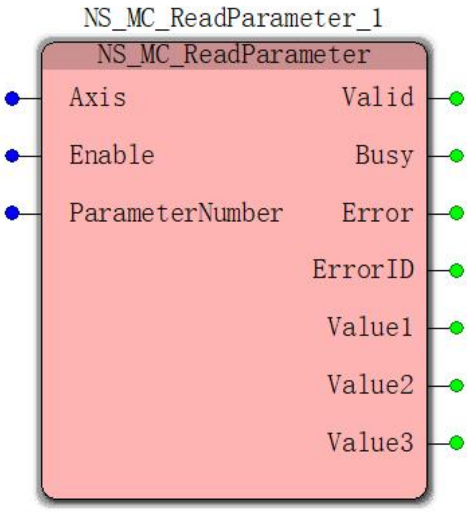


运动控制器中关于实时模位置的计算作如下解释：

模实时位置=终端执行机构位置%Modulo

11.4.22 NS_MC_ReadParameter（读取参数指令）

FB/FC	说明	适用机种
FB	此指令用于读取被控轴的相关参数	VEC-VA-MP-005-MA



➢ **输入参数**

名称	功能	数据类型	设定范围（缺省值）	生效时机
Axis（轴号）	设定指令欲控制的轴	USINT	**模拟量/脉冲:** 0~4（实轴） 5~11（虚轴） **CANopen 模式：** 0~15 (实轴/虚轴) (0)	Enable 为 TRUE 时
Enable（执行位）	当 Enable 为 TRUE 时，指令执行	BOOL	TRUE 或 FALSE	
ParameterNumber（监视功能号）	监视控制轴相应功能号的数值	INT	正数	Enable 为 TRUE 时

➢ **输出参数**

名称	功能	数据类型	输出范围
Valid	该参数输出为 TRUE 时表示指令的输出有效	BOOL	TRUE 或 FALSE
Busy	该输出参数为 TRUE 时表示指令正在执行中	BOOL	TRUE 或 FALSE
Error	该输出参数为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE
ErrorID	指令执行出错时的错误代码。	WORD	-
Value1	对应功能号所监视到的参数	DINT	-
Value2		REAL	-

Value3		LREAL	-

监视功能参数表

功能号	适用功能块	监视描述	生效时机
3	单轴指令/多轴指令	反馈编码器位置（单位：脉冲）	运行指令执行
4	单轴指令/多轴指令	给定位置（单位：脉冲）	运行指令执行
5	单轴指令/多轴指令	setposition 后的反馈编码器位置	运行指令执行
6	单轴指令/多轴指令	setposition 后的给定位置	运行指令执行
7	单轴指令/多轴指令	理论规划的给定速度（单位：r/min）	运行指令执行
8	单轴指令/多轴指令	位置环 PI 后的实时补偿速度	运行指令执行
9	单轴指令/多轴指令	实时给定速度（单位：r/min）	运行指令执行
10	单轴指令/多轴指令	每个底层周期给定位置增量（单位：脉冲）	运行指令执行
11	单轴指令/多轴指令	实时误差（单位：脉冲）	运行指令执行
12	单轴指令/多轴指令	实时模拟量输出	运行指令执行
16	单轴指令/多轴指令	当前给定的模位置	运行指令执行
17	单轴指令/多轴指令	当前给定的模位置对应的脉冲	运行指令执行
18	单轴指令/多轴指令	当前反馈的模位置	运行指令执行
19	单轴指令/多轴指令	当前反馈的模位置对应的脉冲	运行指令执行
20	单轴指令/多轴指令	采样时间 Sample_Time 内采集到脉冲个数	运行指令执行
24	多轴指令	读取绝对值编码器单圈的绝对位置	运行指令执行
40	多轴指令	主轴的滤波后速度	运行指令执行
41	多轴指令	主轴的位置	运行指令执行
42	多轴指令	第二主轴的滤波后速度	运行指令执行
43	多轴指令	第二主轴的位置	运行指令执行
44	单轴指令/多轴指令	直接读取编码器的值，用于检查编码器是否接错或接反	运行指令执行
45	单轴指令/多轴指令	直接读取虚轴编码器的值	运行指令执行
46	单轴指令	读取 MC_TouchProbe 模块进行脉冲锁存时刻其 Active_Axis 实际轴的实时控制误差（单位：脉冲）	运行指令执行
47	单轴指令/多轴指令	DI0 触发时锁存该轴的实时误差（单位：脉冲）	运行指令执行
48	单轴指令/多轴指令	DI1 触发时锁存该轴的实时误差（单位：脉冲）	运行指令执行
49	单轴指令	读取 MC_TouchProbe 模块进行脉冲锁存时刻其 Active_Axis 实际轴的实时转速（单位：r/min）	运行指令执行
63	多轴指令	读 MC_SpecialCamIn 的主轴位置，单位为用户单元	运行指令执行

70	多轴指令	读 MC_CamIn 的主轴位置，单位为用户单元	运行指令执行
180	拓展 IO 类型	扫描拓展类型次数（如读到有拓展，每次扫描后此参数都会叠加）	运行指令执行
181	拓展 IO 类型	第一个拓展的类型	运行指令执行
182	拓展 IO 类型	第二个拓展的类型	运行指令执行
183	拓展 IO 类型	第三个拓展的类型	运行指令执行
184	拓展 IO 类型	第四个拓展的类型	运行指令执行
185	拓展 IO 类型	第五个拓展的类型	运行指令执行
186	拓展 IO 类型	第六个拓展的类型	运行指令执行
187	拓展 IO 类型	第七个拓展的类型	运行指令执行
188	拓展 IO 类型	第八个拓展的类型	运行指令执行

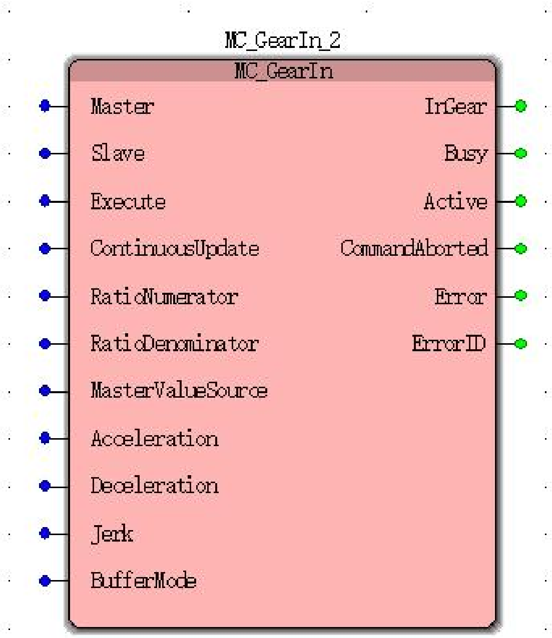
11.5 多轴指令

MasterValueSource 说明：

MasterValueSource=0 时，从轴跟随主轴的命令位置。
MasterValueSource=1 时，从轴跟随主轴的实际位置，实际位置是靠硬件轴口所检测到的脉冲数决定的。

11.5.1 MC_GearIn（电子齿轮耦合指令）

FB/FC	说明	适用机种
FB	此指令用于两个轴之间建立电子齿轮关系	VEC-VA-MP-005-MA



➢ **输入参数**

名称	功能	数据类型	设定范围（缺省值）	生效时机
Master (主轴)	设定指令欲控制的主轴	USINT	**模拟量/脉冲：** 0~4（实轴） 5~11（虚轴） **CANopen 模式：** 0~15 (实轴/虚轴) (0)	Exexcute 由 FALSE 变为 TRUE
Slave （从轴）	设定指令欲控制的从轴	USINT	**模拟量/脉冲：** 0~4（实轴） 5~11（虚轴） **CANopen 模式：** 0~15 (实轴/虚轴) (0)	Exexcute 由 FALSE 变为 TRUE
Execute （执行位）	当 Execute 由 FALSE 变为 TRUE，指令开始执行	BOOL	TRUE 或 FALSE	-
ContinousUpdate	保留	-	-	-
RatioNumerator （电子齿轮分子）	电子齿轮的分子	LREAL	正数、负数 （不可缺省）	Exexcute 由 FALSE 变为 TRUE
RatioDenominator （电子齿轮分母）	电子齿轮的分母	LREAL	正数 （不可缺省）	Exexcute 由 FALSE 变为 TRUE

MasterValueSource （位置来源选择）	从轴命令来源选择 0: 从轴跟随主轴的命令位置 1: 从轴跟随主轴的实际位置	INT	0 或 1	Exexcute 由 FALSE 变为 TRUE
Acceleration （加速度）	设定的目标加速度 (单位：单元/S^2)	LREAL	正数 （不可缺省）	Exexcute 由 FALSE 变为 TRUE
Deceleration （减速度）	设定的目标减速度 (单位：单元/S^2)	LREAL	正数 （不可缺省）	Exexcute 由 FALSE 变为 TRUE
Jerk （加速度的变化率）	设定的目标加速度或减速度的变化率 (单位：单元/S^3)	LREAL	正数、零 (0)	Exexcute 由 FALSE 变为 TRUE
BufferMode （交接模式）	设定两个指令之间交接模式 0：立即打断 1：等待	INT	0:立即打断 1：等待 (0)	Exexcute 由 FALSE 变为 TRUE

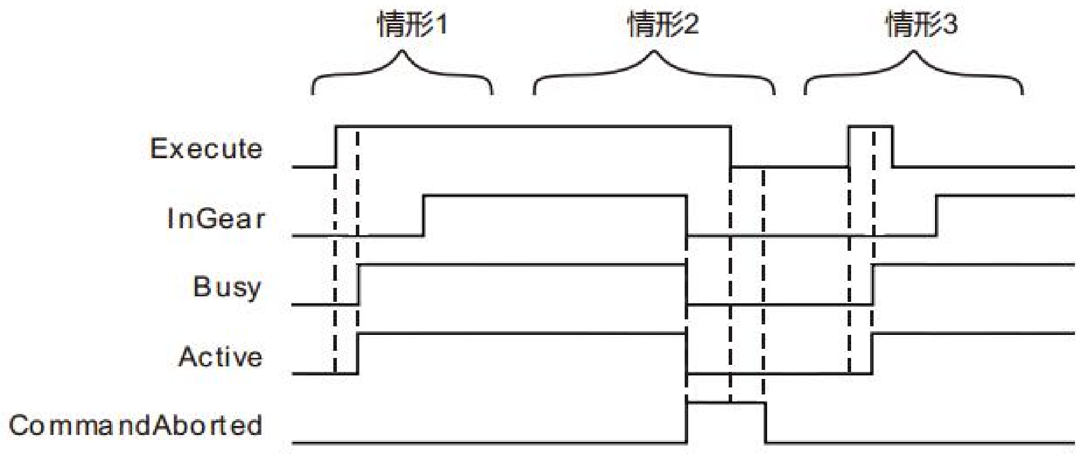
说明：

1. 该指令在 Execute 由 FALSE 变为 TRUE 时开始执行。无论该指令是否执行完成，Execute 再次由 FALSE 变为 TRUE 时，该指令可以重新执行，此时能够重新生效的引脚参数包括 RatioNumerator、RatioDenominator、MasterValueSource、Acceleration、Deceleration、Jerk、BufferMode，并且 CommandAborted 输出 TRUE。
2. 该指令执行时，该指令的从轴执行其它运动指令如 MC_MoveVelocity 能够打断该指令，主轴与从轴之间的齿轮关系会解除。可以执行 MC_Halt 或者 MC_Stop 停止从轴。
3、该指令跟随的是主轴的脉冲变化。

➢ **输出参数**

名称	功能	数据类型	输出范围
InGear（达到同步状态）	该参数输出为 TRUE 时表示从轴达到同步状态	BOOL	TRUE 或 FALSE
Busy（执行中）	该参数输出为 TRUE 时表示指令正在执行中	BOOL	TRUE 或 FALSE
Active（控制中）	该参数输出为 TRUE 时表示指令正在控制轴	BOOL	TRUE 或 FALSE
CommandAborted（中断）	该参数输出为 TRUE 时表示指令被中断	BOOL	TRUE 或 FALSE
Error（错误）	该指令输出为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE
ErrorID（错误代码）	执行指令出错时的错误代码	WORD	-

➢ **输出参数时序变化图**



情形 1： 当 Execute 由 FALSE 变为 TRUE 时，且一个周期后，Busy、Active 变为 TRUE。当同步状态到达时，InGear 变为 TRUE，同时 Busy 和 Active 依然保持为 TRUE。

情形 2： 当 Execute 为 TRUE 时，从轴被其它指令控制时，该指令被其它指令中断，CommandAborted 变为 TRUE，同时 Busy 和 Active 变为 FALSE；当 Execute 由 TRUE 变为 FALSE 时，一个周期后 CommandAborted 变为 FALSE。

情形 3：在指令执行过程中，当 Execute 由 TRUE 变为 FALSE 后，InGear 变为 TRUE，同时 Busy 和 Active 依然保持为 TRUE。

➢ **功能说明**

- 此指令用于两个轴之间建立电子齿轮关系。该指令执行后，从轴按照电子齿轮分子、电子齿轮分母、命令来源、加速度、减速度、加速度的变化率、交接模式与主轴进行齿轮动作。主轴可以为实轴、虚轴，或者编码器轴，从轴可以为实轴或者虚轴。
- 此指令执行时，从轴需要在使能状态下，主轴在使能或去使能状态下都可以。
- 两个轴没有建立电子齿轮关系时（即指令的 InGear 为 FALSE 时），执行该指令，从轴按照该指令指定的电子齿轮分子、电子齿轮分母、加速度、减速度、加速度的变化率到达目标速度（主轴的实时速度）。

①当主轴实时加速度不变时，从轴以指令上设定的加减速度、加速度的变化率到达目标速度
②当主轴实时加速度变化时，从轴速度根据以下公式朝目标速度变化。

$$\text{从轴加(减)速度} = \text{主轴加(减)速度} * \frac{\textit{电子齿轮比分子}}{\textit{电子齿轮比分母}}$$

两个轴建立电子齿轮关系后（该指令的 InGear 为 TRUE 时），从轴速度与电子齿轮分子、电子齿轮分母和主轴速度的关系如下所示：

$$\text{从轴目标速度} = \text{主轴速度} * \frac{\textit{电子齿轮比分子}}{\textit{电子齿轮比分母}}$$

- 电子齿轮比

$$\text{电子齿轮比} = \frac{\textit{电子齿轮比分子}}{\textit{电子齿轮比分母}}$$

电子齿轮比为正数时，从轴和主轴的运动方向相同。

电子齿轮比为负数时，从轴和主轴的运动方向相反。

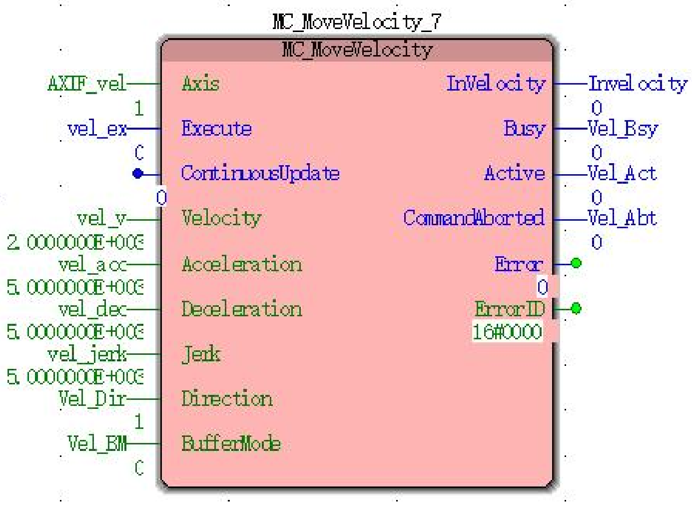
程序范例

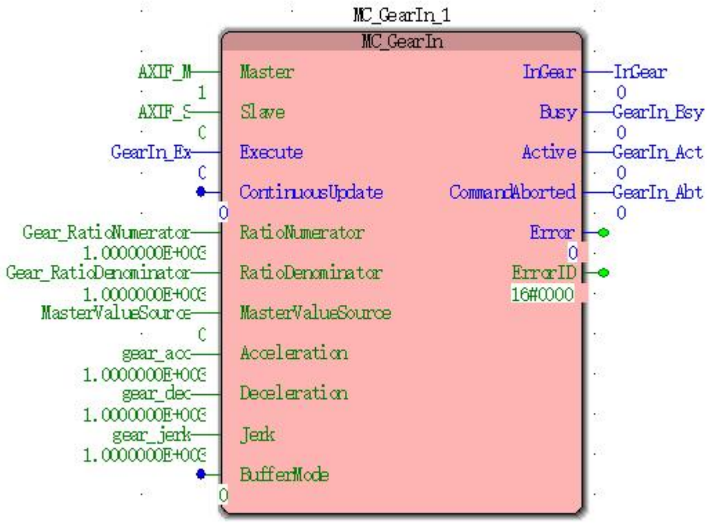
MC_GearIn 指令程序范例如下所示：

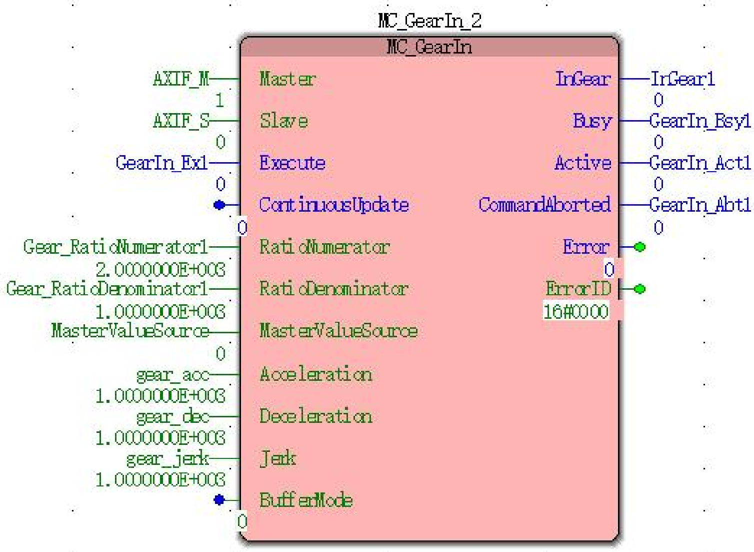
1、变量和程序

变量名	数据类型	初始值
MC_MoveVelocity_7	MC_MoveVelocity	
AXIF_vel	USINT	1
Vel_ex	BOOL	FALSE
Vel_v	LREAL	2000.0
Vel_acc	LREAL	5000.0
Vel_dec	LREAL	5000.0
Vel_jerk	LREAL	5000.0
Vel_Dir	INT	1
Vel_BM	INT	0
Invelocity	BOOL	
Vel_Bsy	BOOL	
Vel_Act	BOOL	
Vel_Abt	BOOL	
MC_GearIn_1	MC_GearIn	
AXIF_M	USINT	1
AXIF_S	USINT	0
GearIn_Ex	BOOL	FALSE
Gear_RatioNumerator	LREAL	1000.0
Gear_RatioDenominator	LREAL	1000.0
MasterValueSource	INT	0
Gear_acc	LREAL	1000.0
Gear_dec	LREAL	1000.0
Gear_Jerk	LREAL	1000.0
InGear	BOOL	
GearIn_Bsy	BOOL	
GearIn_Act	BOOL	
GearIn_Abt	BOOL	
MC_GearIn_2	MC_GearIn	
GearIn_Ex1	BOOL	FALSE
Gear_RatioNumerator1	LREAL	2000.0
Gear_RatioDenominator1	LREAL	1000.0
InGear1	BOOL	
GearIn_Bsy1	BOOL	
GearIn_Act1	BOOL	

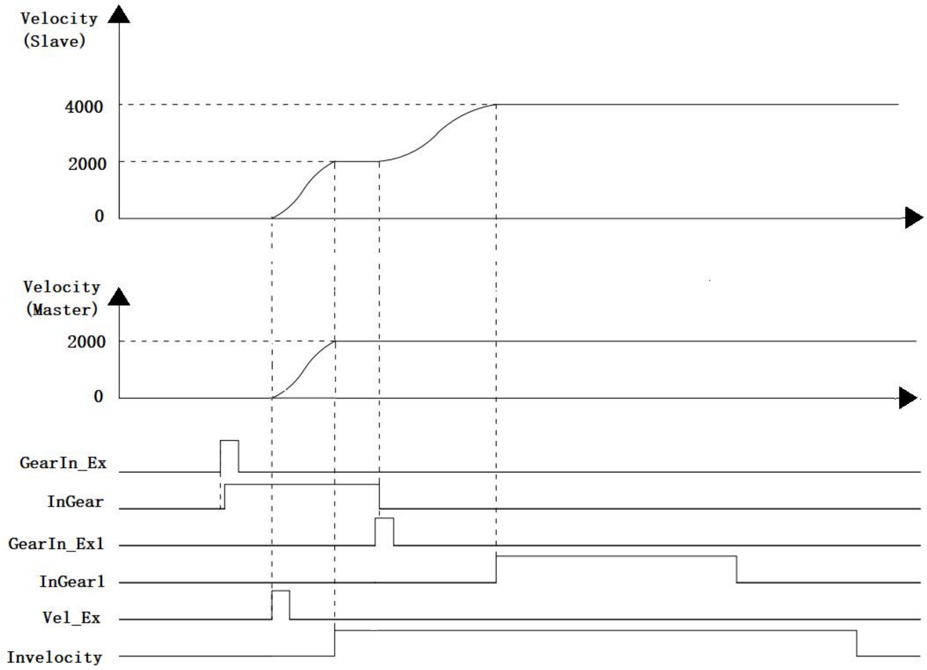
GearIn_Abt1	BOOL	







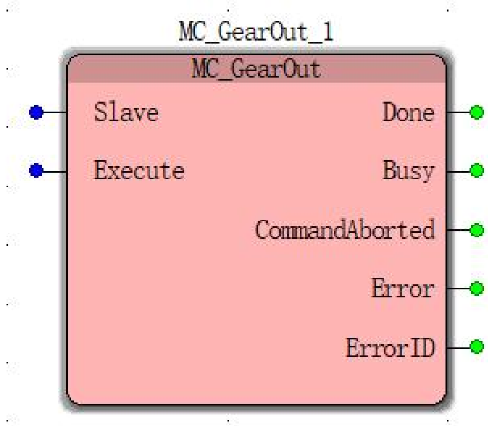
2、运动曲线和时序图



- MC_GearIn1 电子齿轮比分子和分母都为 1，GearIn_Ex 由 FALSE 变为 TRUE，一个周期后，Gear_Bsy、Gear_Act、InGear 变为 TRUE，主轴和从轴建立齿轮关系。
- 主轴和从轴建立电子齿轮关系后，Vel_Ex 由 FALSE 变为 TRUE，且一个周期后，Vel_Bsy、Vel_Act 变为 TRUE，主轴执行速度指令，从轴跟随主轴运转。
- MC_GearIn2 电子齿轮比分子和分母分别为 2 和 1，GearIn_Ex1 由 FALSE 变为 TRUE，且一个周期后，GearIn_Bsy1、GearIn_Act1 和 GearIn_Abt1 变为 TRUE，从轴根据 MC_GearIn2 指令设定的电子齿轮比、命令来源、加速度、加速度的变化率、交接模式到达目标速度。因为 GearIn2 电子齿轮比分子和分母分别为 2 和 1，所以从轴的目标速度为主轴速度的 2 倍。InGear1 变为 TRUE 时，从轴的速度为主轴速度的 2 倍。

11.5.2 MC_GearOut（电子齿轮脱离指令）

FB/FC	说明	适用机种
FB	此指令用于解除两个轴之间已建立的电子齿轮关系	VEC-VA-MP-005-MA



➢ **输入参数**

名称	功能	数据类型	设定范围 （缺省值）	生效时机
Slave （从轴）	设定指令欲控制的从轴	USINT	**模拟量/脉冲：** 0~4（实轴） 5~11（虚轴） **CANopen 模式：** 0~15 (实轴/虚轴) (0)	Exexcute 由 FALSE 变为 TRUE
Execute （执行位）	当 Execute 由 FALSE 变为 TRUE，指令开始执行	BOOL	TRUE 或 FALSE	-

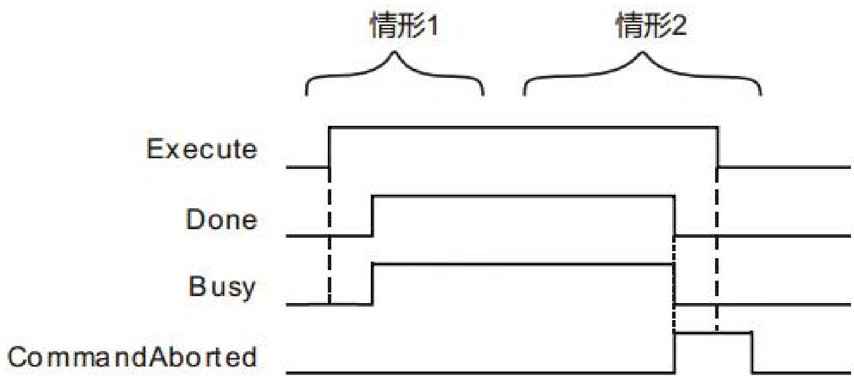
说明：

1. 两个轴建立电子齿轮关系（MC_GearIn）后，如果从轴通过 MC_GearOut 指令脱离电子齿轮关系，从轴将保持脱离时的速度继续运行。
2. 该指令执行完成后，从轴可以执行其它运动指令。
3. 两个轴脱离电子齿轮关系（MC_GearOut）后，如果想停止从轴，可以使用 MC_Halt、MC_Stop 或者 NS_MC_StopByPos 指令使从轴停止。

➢ **输出参数**

名称	功能	数据类型	输出范围
Done	该参数输出为 TRUE 时表示指令执行完成	BOOL	TRUE 或 FALSE
Busy（执行中）	该参数输出为 TRUE 时表示指令正在执行中	BOOL	TRUE 或 FALSE
CommandAborted（中断）	该参数输出为 TRUE 时表示指令被中断	BOOL	TRUE 或 FALSE
Error（错误）	该指令输出为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE
ErrorID（错误代码）	执行指令出错时的错误代码	WORD	-

➢ **输出参数时序变化图**



情形 1：当 Execute 由 FALSE 变为 TRUE 时，一个周期后，Done 变为 TRUE。Execute 由 TRUE 变为 FALSE 后，Busy 和 Done 依然保持为 TRUE。

情形 2：当 Execute 为 TRUE 时，若指令被其它指令中断，CommandAborted 变为 TRUE，同时 Busy 和 Done 变为 FALSE；当 Execute 由 TRUE 变为 FALSE 时，一个周期后，CommandAborted 变为 FALSE。

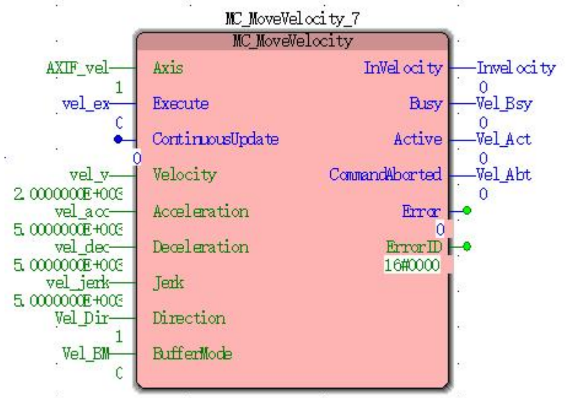
程序范例

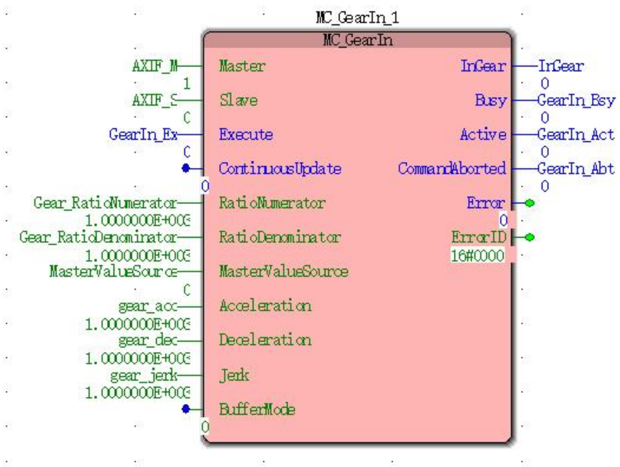
MC_GearOut 指令程序范例如下所示：

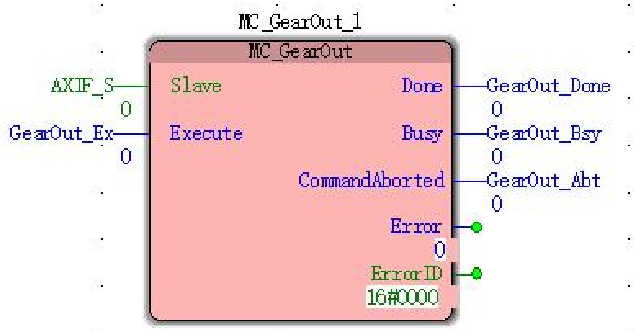
1、变量和程序

变量名	数据类型	初始值
MC_MoveVelocity_7	MC_MoveVelocity	
AXIF_vel	USINT	1
Vel_ex	BOOL	FALSE
Vel_v	LREAL	1000.0
Vel_acc	LREAL	5000.0
Vel_dec	LREAL	5000.0
Vel_jerk	LREAL	5000.0
Vel_Dir	INT	1
Vel_BM	INT	0
Invelocity	BOOL	
Vel_Bsy	BOOL	
Vel_Act	BOOL	
Vel_Abt	BOOL	
MC_GearIn_1	MC_GearIn	
AXIF_M	USINT	1
AXIF_S	USINT	0
GearIn_Ex	BOOL	FALSE
Gear_RatioNumerator	LREAL	1000.0
Gear_RatioDenominator	LREAL	1000.0
MasterValueSource	INT	0
Gear_acc	LREAL	1000.0
Gear_dec	LREAL	1000.0

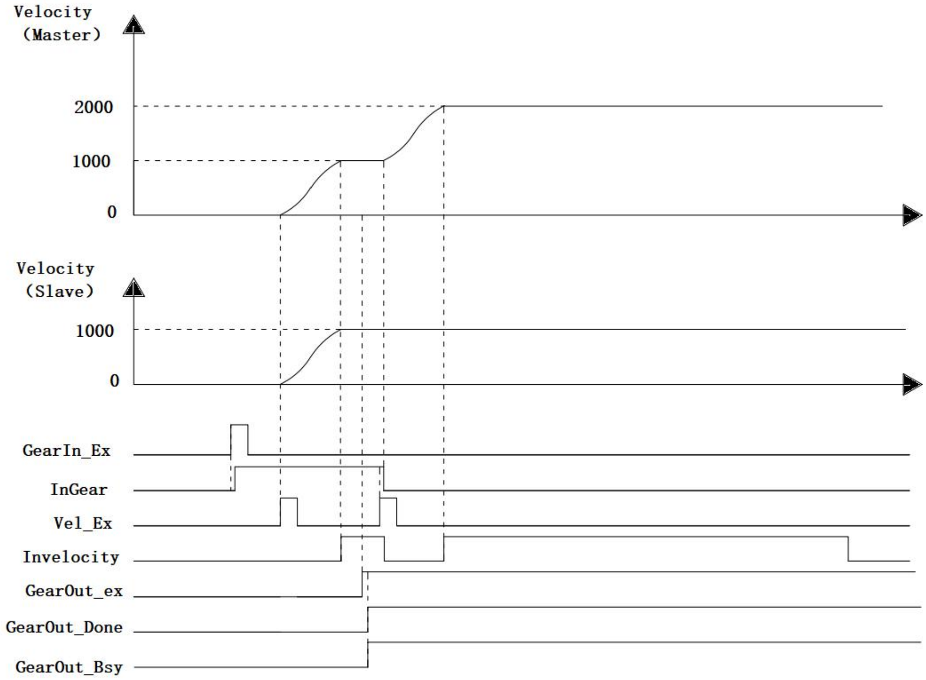
Gear_Jerk	LREAL	1000.0
InGear	BOOL	
GearIn_Bsy	BOOL	
GearIn_Act	BOOL	
GearIn_Abt	BOOL	
MC_GearOut_1	MC_GearOut	
GearOut_Ex	BOOL	FALSE
GearOut_Done	BOOL	
GearOut_Bsy	BOOL	
GearOut_Abt	BOOL	







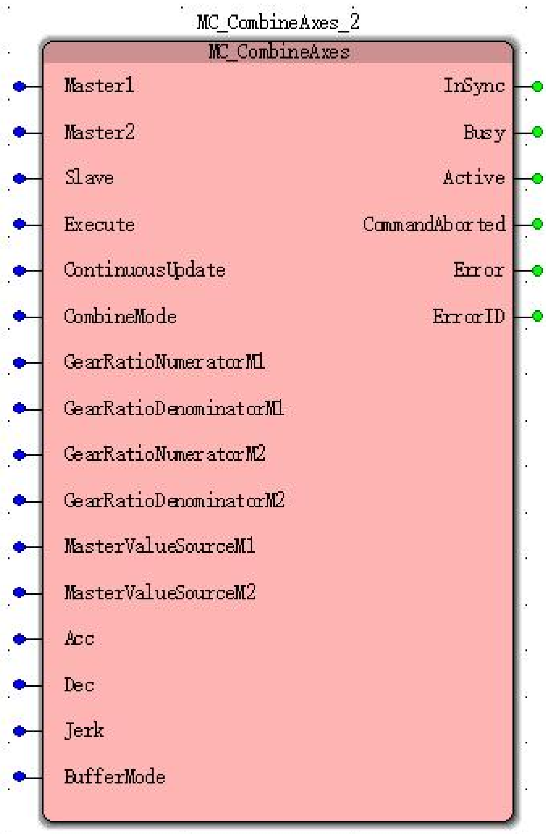
2、运动曲线和时序图



- MC_GearIn1 电子齿轮比分子和分母都为 1，GearIn_Ex 由 FALSE 变为 TRUE，一个周期后，Gear_Bsy、Gear_Act、InGear 变为 TRUE，主轴和从轴建立齿轮关系。
- 主轴和从轴建立电子齿轮关系后，Vel_Ex 由 FALSE 变为 TRUE，且一个周期后，Vel_Bsy、Vel_Act 变为 TRUE，主轴执行速度指令，从轴跟随主轴运转。
- 主轴执行速度指令时，GearOut_Ex 由 FALSE 变为 TRUE，一个周期后，GearOut_Bsy、GearOut_Done 和 GearIn_Abt 变为 TRUE，从轴以当前的速度继续运转。
- 修改主轴参数速度指令 Velocity 为 2000.0，再次执行，主轴速度增加的 2000.0，从轴不再受主轴影响。

11.5.3 MC_CombineAxes（双主轴联合齿轮指令）

FB/FC	说明	适用机种
FB	将两个主轴的位置相加或相减的值作为从轴位置输出	VEC-VA-MP-005-MA



➢ **输入参数**

名称	功能	数据类型	设定范围（缺省值）	生效时机
Master1 (主轴 1)	进行控制时，第一个轴的位置来源。	USINT	**模拟量/脉冲：** 0~4（实轴） 5~11（虚轴） **CANopen 模式：** 0~15 (实轴/虚轴) (0)	Exexcute 由 FALSE 变为 TRUE
Master2 （主轴 2）	进行控制时，第二个轴的位置来源。	USINT	0~4 实轴 5~11 虚轴 (CANopen 模式：0~15,可作实轴或虚轴) (0)	Exexcute 由 FALSE 变为 TRUE
Slave （从轴）	被控轴的轴	USINT	0~4 实轴 5~11 虚轴 (CANopen 模式：	Exexcute 由 FALSE 变为 TRUE

			0~15,可作实轴或虚轴) (0)	
Execute （执行位）	当 Execute 由 FALSE 变为 TRUE 时，指令开始执行	BOOL	TRUE 或 FALSE	-
ContinuousUpdate	保留	-	-	-
CombineMode （合成模式）	选择合成模式 0：两主轴各自变化位置相加 1：两主轴各自变化位置相减	INT	0 或 1	Exexcute 由 FALSE 变为 TRUE
GearRatioNumeratorM1 （主轴 1 齿轮比分子）	设定主轴 1 齿轮比分子	LREAL	正数或负数 （不可缺省）	Exexcute 由 FALSE 变为 TRUE
GearRatioDenominatorM1 （主轴 1 齿轮比分母）	设定主轴 1 齿轮比分母	LREAL	正数或负数 （不可缺省）	Exexcute 由 FALSE 变为 TRUE
GearRatioNumeratorM2 （主轴 2 齿轮比分子）	设定主轴 2 齿轮比分子	LREAL	正数或负数 （不可缺省）	Exexcute 由 FALSE 变为 TRUE
GearRatioDenominatorM2 （主轴 2 齿轮比分母）	设定主轴 2 齿轮比分母	LREAL	正数或负数 （不可缺省）	Exexcute 由 FALSE 变为 TRUE
MasterValueSourceM1 （主轴 1 的同步源）	设定主轴 1 的同步源 0：命令位置 1：实际位置	INT	0 或 1	Exexcute 由 FALSE 变为 TRUE
MasterValueSourceM2 （主轴 2 的同步源）	设定主轴 2 的同步源 0：命令位置 1：实际位置	INT	0 或 1	Exexcute 由 FALSE 变为 TRUE
Acc （加速度）	设定从轴的加速度 单位：单元/S^2	LREAL	正数 （不可缺省）	Exexcute 由 FALSE 变为 TRUE
Dec （减速度）	设定从轴的减速度 单位：单元/S^2	LREAL	正数 （不可缺省）	Exexcute 由 FALSE 变为 TRUE
Jerk （加速度的变化率）	设定从轴的加速度的变化率 单位：单元/S^3	LREAL	正数 （不可缺省）	Exexcute 由 FALSE 变为 TRUE

BufferMode（交接模式）	设定两个指令之间交接模式。 0：打断 1：等待	INT	0 或 1	Exexcute 由 FALSE 变为 TRUE

说明：

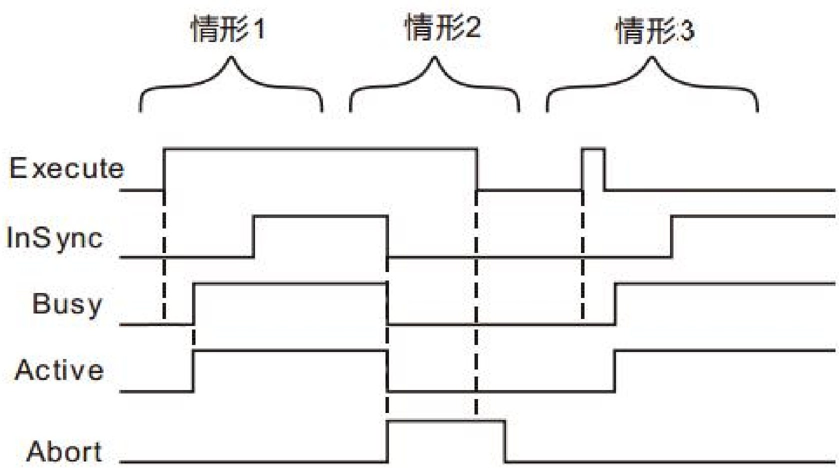
1. 该指令在 Execute 由 FALSE 变为 TRUE 时开始执行。无论该指令是否执行完成，Execute 再次由 FALSE 变为 TRUE 时，该指令可以重新执行，此时能够重新生效的引脚参数包括 CombineMode、RatioNumeratorM1、RatioDenominatorM1、RatioNumeratorM2、RatioDenominatorM2、MasterValueSourceM1、MasterValueSourceM2、Acceleration、Deceleration、Jerk、BufferMode。
2. 该指令执行时，该指令的从轴执行其它运动指令如 MC_MoveVelocity 能够打断该指令，主轴与从轴之间的位置关系会解除。可以执行 MC_Halt 或者 MC_Stop 停止从轴。

0

➢ **输出参数**

名称	功能	数据类型	输出范围
InSync（达到同步状态）	该参数输出为 TRUE 时表示从轴达到同步状态	BOOL	TRUE 或 FALSE
Busy（执行中）	该参数输出为 TRUE 时表示指令正在执行中	BOOL	TRUE 或 FALSE
Active（控制中）	该参数输出为 TRUE 时表示指令正在控制轴	BOOL	TRUE 或 FALSE
CommandAborted（中断）	该参数输出为 TRUE 时表示指令被中断	BOOL	TRUE 或 FALSE
Error（错误）	该指令输出为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE
ErrorID（错误代码）	执行指令出错时的错误代码	WORD	-

➢ **输出参数时序变化图**



情形一：当 Execute 由 FALSE 变为 TRUE 时，一个周期后，Busy、Active 变为 TRUE。当从轴已经和两个主轴同步，InSync 变为 TRUE，同时 Busy 和 Active 依然保持为 TRUE。

情形二： 当 Execute 为 TRUE 时， Busy 为 TRUE， Active 为 TRUE， 当从轴已经和两个主轴同步，InSync 为 TRUE，此时另一个指令打断本指令，CommandAborted 变为 TRUE，同时 Invelocity、Busy 和 Active 变为 FALSE，当 Execute 由 TRUE 变为 FALSE 时，一个周期后，CommandAborted 变为 FALSE。
情形三： 在执行的过程中，当 Execute 由 TRUE 变为 FALSE 后，指令仍然在执行，Busy 和 Active 不会改变状态。当从轴已经和两个主轴同步，InSync 变为 TRUE，同时 Busy 和 Active 依然保持为 TRUE。

- **功能说明**

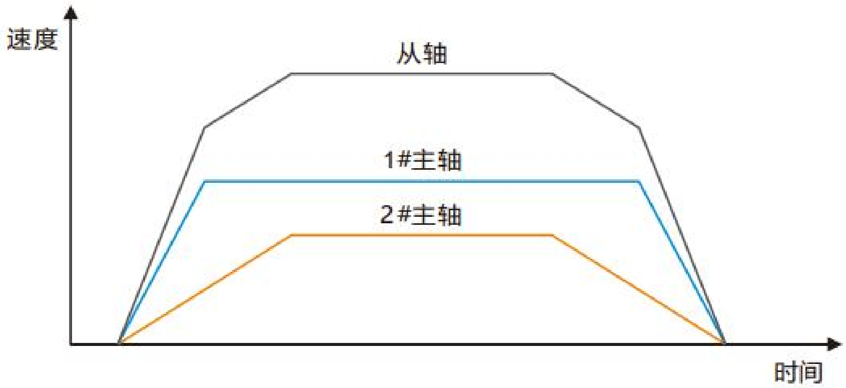
  此指令用于将两个主轴的位置相加或相减的值作为从轴位置输出。这里所说的位置其单位是脉冲。

  - **此指令合成方式分为两种：相加或相减**

  一号主轴的位置变化量和二号主轴的位置变化量进行相加或者相减，将计算出的值作为从轴的位置变化量输出。

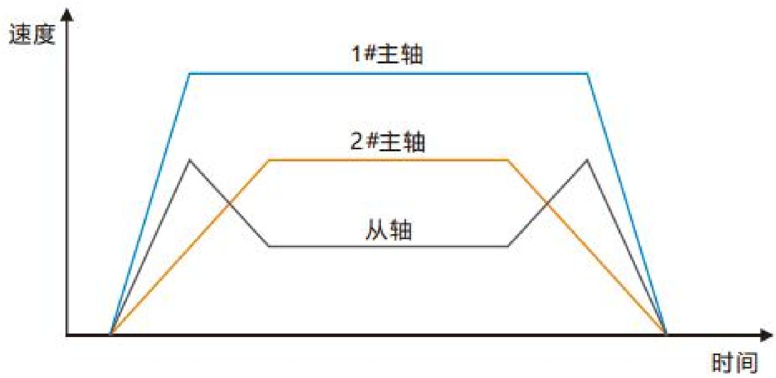
  - 当 CombineMode 值为 0 时

$$从轴位置变化量 = 主轴1位置变化量 * \frac{主轴1齿轮比分子}{主轴1齿轮比分母} + 主轴2位置变化量 * \frac{主轴2齿轮比分子}{主轴2齿轮比分母}$$



  - 当 CombineMode 值为 1 时

$$从轴位置变化量 = 主轴1位置变化量 * \frac{主轴1齿轮比分子}{主轴1齿轮比分母} - 主轴2位置变化量 * \frac{主轴2齿轮比分子}{主轴2齿轮比分母}$$



- 主轴齿轮比分子和分母设定是调节两个主轴位置变化量的因子，使用公式见上。
- 主轴的同步源可以设置为 0：命令位置；1：实际位置，用以确认位置变化量的来源。设置为 0 时是主轴命令位置的变化量进行相加或相减，设置为 1 时是主轴实际位置的变化量进行相加或相减。
- 加速度、减速度和加速度的变化率表示在执行本指令之前，主轴已经在运动状态，此时若执行本指令，从轴会根据加速度、减速度和加速度的变化率进行加速或者减速，以达到和主轴位置变化同步。同步后 InSync 为 TRUE，指令执行完成。
- 要结束本指令的主从轴关系，请使用其它运动指令控制从轴（如 MC_Stop）,BufferMode 输入引脚填 0，以打断本指令，解除主从轴关系。

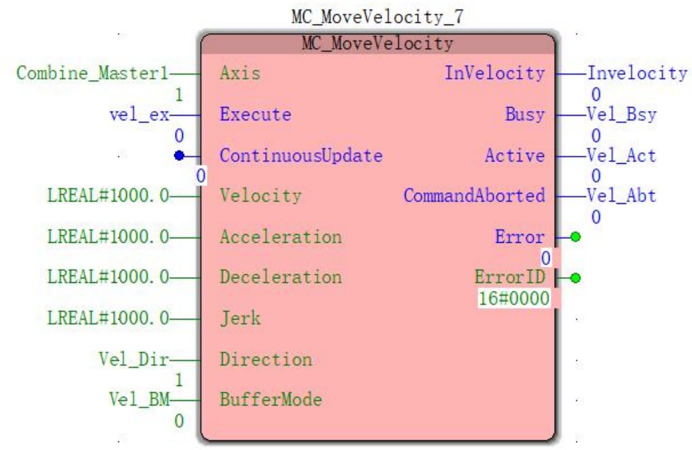
程序范例

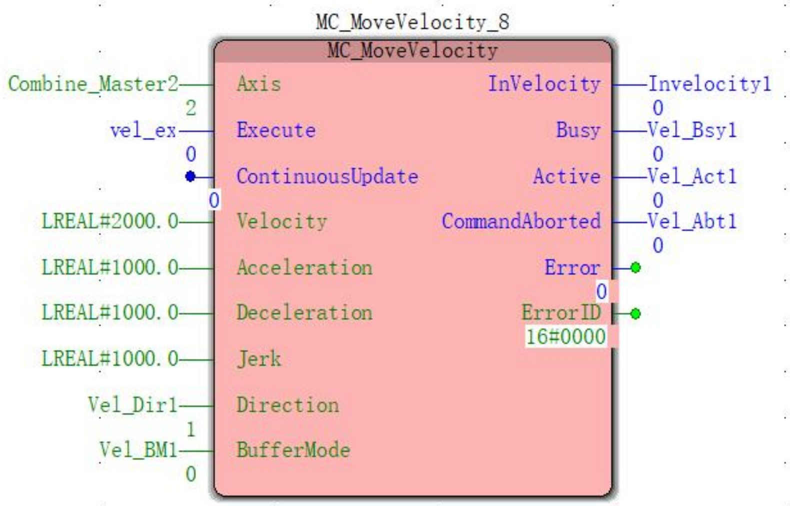
MC_CombineAxes 指令程序范例如下：

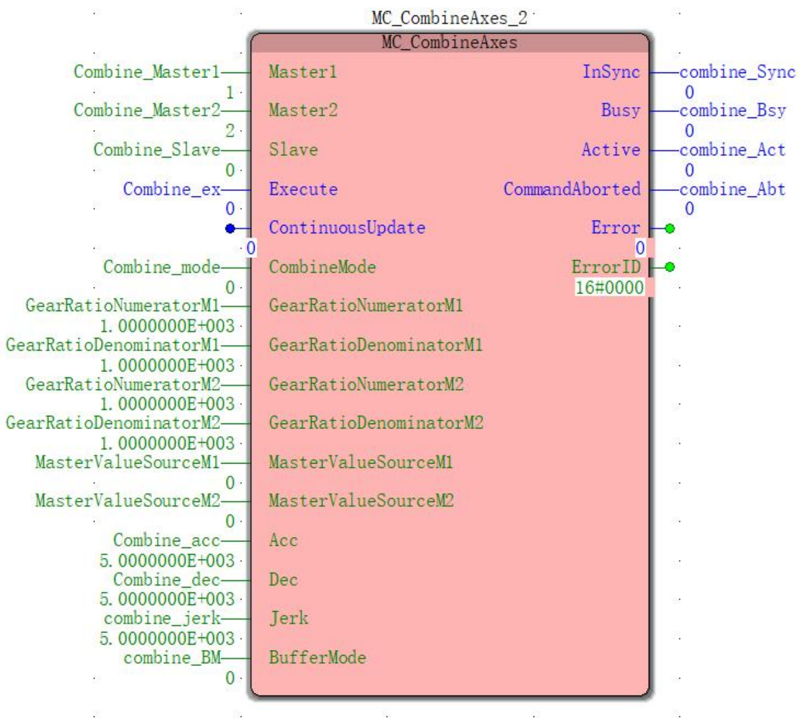
1、变量和程序

变量名	数据类型	初始值
MC_MoveVelocity_7	MC_MoveVelocity	
Combine_Master1	USINT	1
Vel_ex	BOOL	FASLE
Vel_Dir	INT	1
Vel_BM	INT	0
Invelocity	BOOL	
Vel_Bsy	BOOL	
Vel_Act	BOOL	
Vel_Abt	BOOL	
MC_MoveVelocity_8	MC_MoveVelocity	
Combine_Master2	USINT	2
Vel_ex1	BOOL	FASLE
Vel_Dir1	INT	1
Vel_BM1	INT	0
Invelocity1	BOOL	
Vel_Bsy1	BOOL	
Vel_Act1	BOOL	
Vel_Abt1	BOOL	
MC_CombineAxes_2	MC_CombineAxes	
Combine_Slave	USINT	1
Combine_ex	BOOL	FALSE
Combine_mode	INT	0
GearRatioNumeratorM1	LREAL	1000.0
GearRatioDenominatorM1	LREAL	1000.0
GearRatioNumeratorM2	LREAL	1000.0
GearRatioDenominatorM2	LREAL	1000.0
MasterValueSourceM1	INT	0

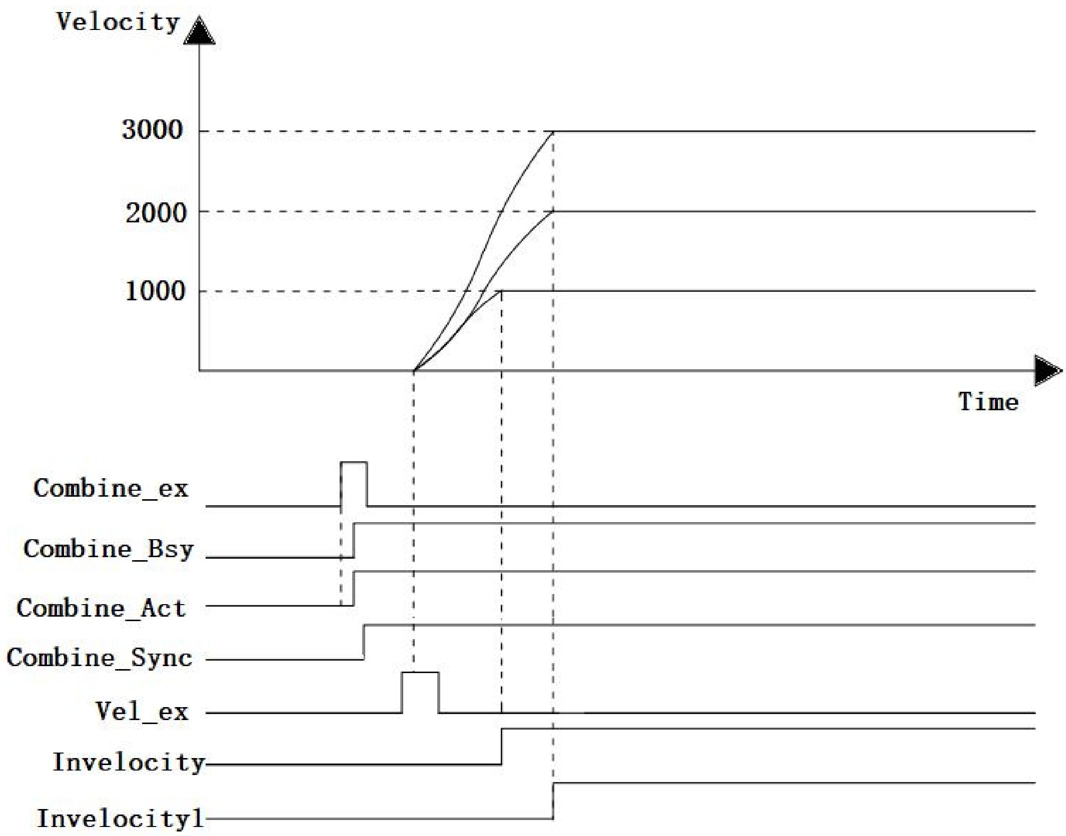
MasterValueSourceM2	INT	0
Combine_acc	LREAL	5000.0
Combine_dec	LREAL	5000.0
Combine_jerk	LREAL	5000.0
Combine_BM	INT	0
Combine_Sync	BOOL	
Combine_Bsy	BOOL	
Combine_Act	BOOL	
Combine_Abt	BOOL	







2、运动曲线和时序图



- 当 Combine_ex 由 FALSE 变为 TRUE，MC_CombineAxes 指令开始执行，一段时间后，指令执行成功，Combine_InSync 变为 TRUE，三个轴按照指令要求达到同步运动状态。

此时将两个主轴的 MC_MoveVelocity 指令的 Excute 变为 TRUE，两个主轴开始运动，此时从轴也开始根据两个主轴位置的变化量的和进行运动,单位时间内从轴的位置变化量为两主轴位置变化量的和。当主轴指令执行完成后，三个轴依然保持同步状态。若要打断三个轴的同步状态，请按照状态机转换图使用相应指令打断从轴，即可解除同步状态

11.5.4 旋切电子凸轮简介

简介

电子凸轮从根本上讲就是由计算机实现的凸轮的功能，整个系统一般由硬件和软件两大部分组成。硬件系统主要包括微处理器、存储装置、D/A 转换器、控制器和执行机构。微处理器根据编码器信号从存储装置中取得相应的位移信号或者根据公式计算出位移值，然后将凸轮的位移值输入 D/A 转换装置，转换后的信号经过控制器处理驱动执行机构实现相应的运动。

应用范围

电子凸轮在机械行业的材料定长剪切领域具有广泛的应用，例如定长切断钢管、木材， 对钢卷、铝带进行定长裁剪，将瓦楞纸进行裁剪，对包装袋进行横封分切、冲孔、压花等。电子凸轮应用中飞剪、旋切、追剪、停剪以及排料等统称为轮切，又叫横切。横切是相对于纵切而言的，是指在物料运输方向对物料进行垂直切割，通常为定长裁切。

(1) 旋切

在该机构中，辊筒上安装一把（或者数把剪切刀），运动中辊筒带动剪切刀旋转，辊筒运动一周对材料剪切一次（或者数次）。

(2) 飞剪

在部分定义中与旋切相同，但在某种特殊的剪切中，例如钢板剪切中，采用偏心轴安装方式，使剪切刀绕固定刀座作回转运动，达到定长剪切的目的[5]。

(3) 追剪

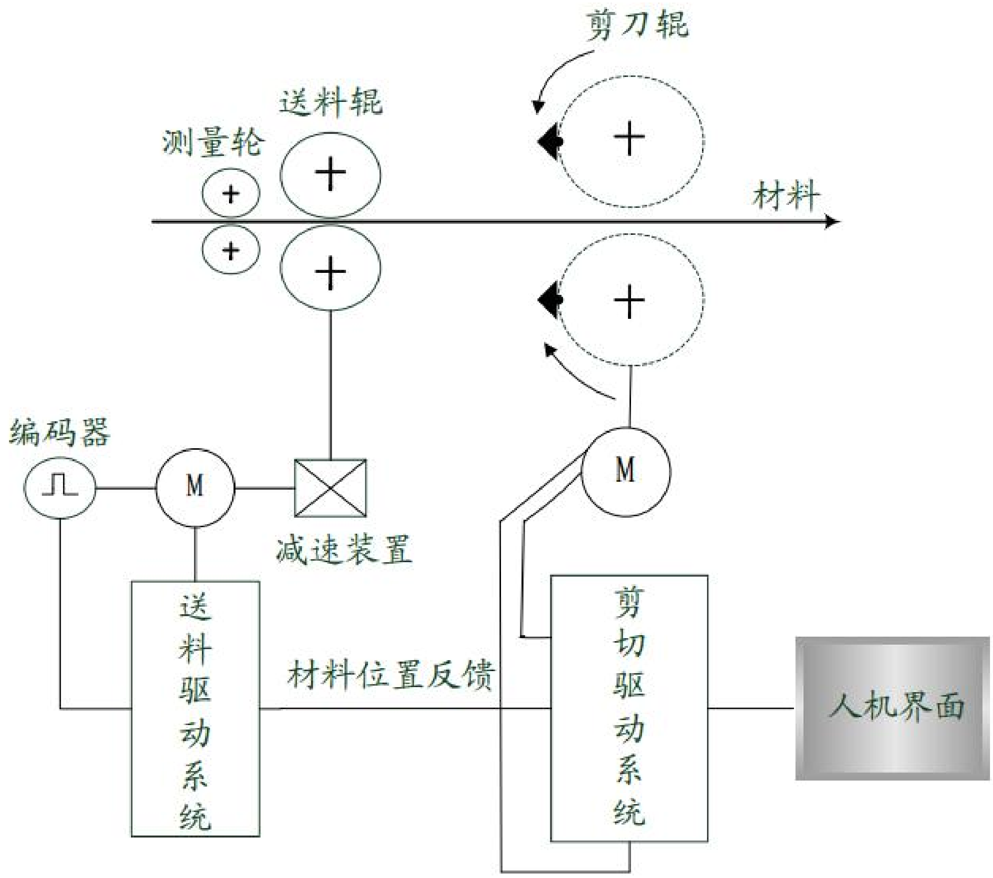
追剪的特点是：在设定的同步区，牵引剪切部件的速度和送料速度一致，在同步区完成剪切运动，而不同的切割长度则通过调节非同步区的速度来适应。 追剪与旋切、飞剪的的最大不同之处是：追剪为往复运动，而旋切、飞剪为同向运动。追剪的另一种应用方式为飞锯，飞锯是指当剪切机构与送料同步时，对物料进行锯切。

(4) 停剪

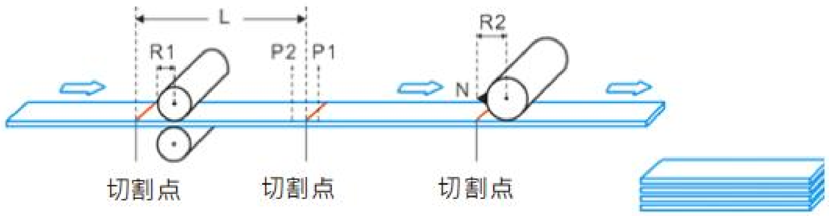
与上述剪切方式不同，停剪控制送料轴，是在剪切刀切断抬起的时间内迅速送料，并停下来剪断。与前面几种不同，停剪中的送料为间隙送料，故停剪的控制最简单，加工效率也最低。

11.5.5 旋切功能的系统构成

实现旋切功能的基本要素包括刀辊轴、进给轴、检测元件、控制单元及电机。另外根据加工产品的不同可能还需要色标。系统的构成如图 11.5.9 所示，刀辊每旋转一圈完成一次剪切，测量辊测量材料的长度，控制单元根据预先设计的凸轮曲线控制刀辊的运动，使得刀刃与材料进入啮合时通过的材料的长度正好是所需要的长度，从而完成一次精确的定长剪切。



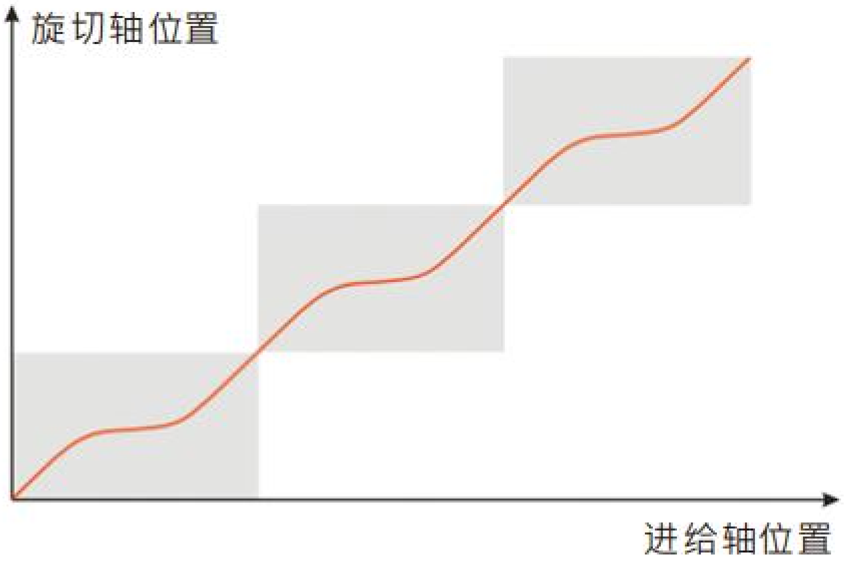
11.5.6 旋切工艺参数



<table>
<tr><th>图中参数</th><th>实际含义</th><th>指令中的名称</th></tr>
<tr><td>L</td><td>设定裁切长度（单位：unit（单元））</td><td>Cut_Length（裁切长度）</td></tr>
<tr><td>R1</td><td>进料辊轴半径（单位：unit（单元））</td><td>FeedAxisRadius（进给轴半径）</td></tr>
<tr><td>R2</td><td>旋切轴半径；</td><td>Cutter_Cir（切刀周长）=2*R2*3.1415</td></tr>
<tr><td>P1</td><td rowspan="2">同步区</td><td rowspan="2">Sync_Angle（同步区）</td></tr>
<tr><td>P2</td></tr>
<tr><td>N</td><td>旋切轴的刀头数</td><td>RotaryAxisKnifeNum</td></tr>
<tr><td colspan="3">特别说明：指令中的名称请参考 11.5.9 NS_MC_RotaryCutIn（旋切指令）</td></tr>
</table>

11.5.7 旋切功能控制特性

旋切功能是一种特殊的电子凸轮功能。连续切割时，其旋切曲线示意图如下：



功能特性

1.、用户可根据工艺要求自由设置切割长度，切割长度可以小于或大于等于切刀周长。

2.、在同步区内，进料轴与旋切轴按一定的速度比例运转（速度通常相等），并且材料的切割发生在同步区内。

3、旋切功能启动后，旋切轴跟随进料轴的相位动作，因此进料轴可以作匀速、加速、减速运动、不规则运动。

4、旋切功能支持多刀头的旋切辊。

5、旋切功能结束后，旋切刀停止在系统零点，即进入点。

11.5.8 旋切功能介绍

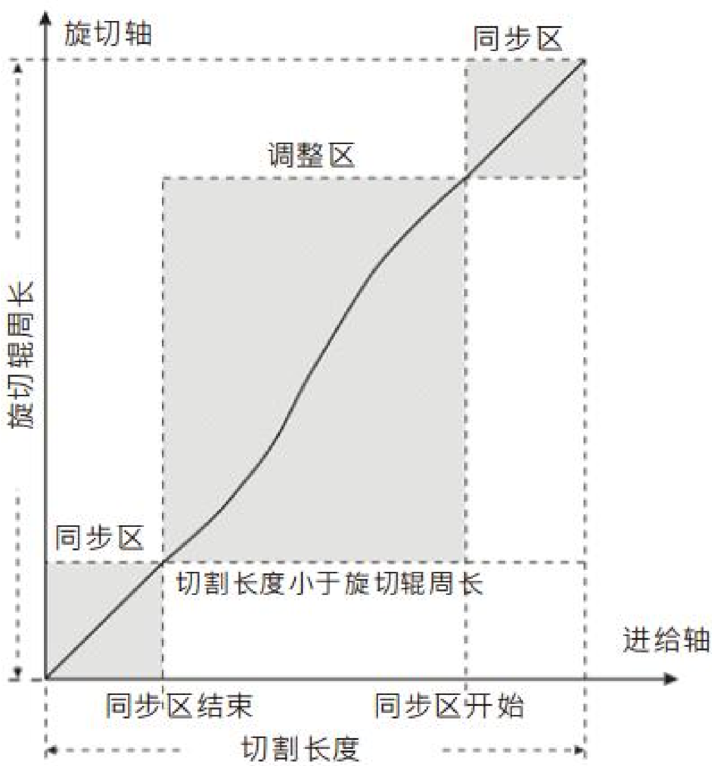
旋切曲线分为同步区与调整区。

同步区：此时进给轴与旋切轴按固定的速度比例运转（刀头的线速度与切割面的线速度通常相等），并且材料的切割发生在同步区内。

调整区：由于切割长度不同，需要做相应的位移调整。根据切割长度调整区可以分为下面三种情况。

1：短料切割

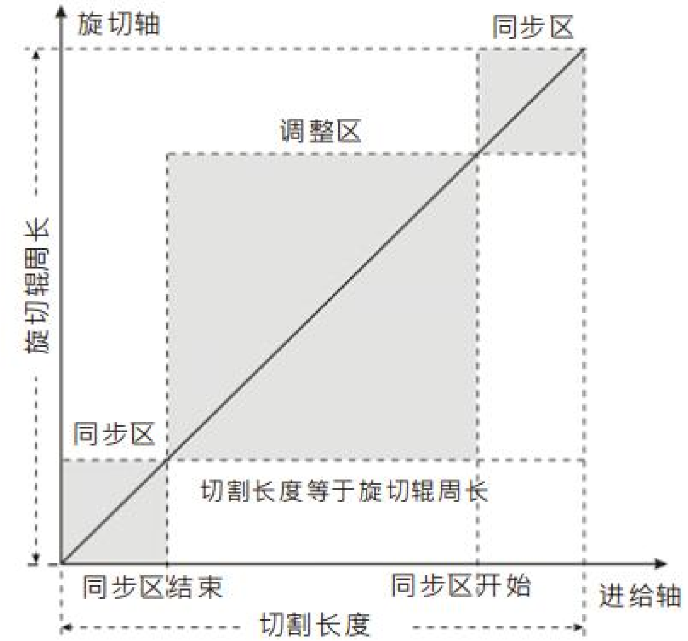
切割长度小于旋切辊周长时，任一周期的旋切曲线如下



短料切割时，旋切轴必须在调整区内先加速，然后再减速到同步速度。

2：等长切割

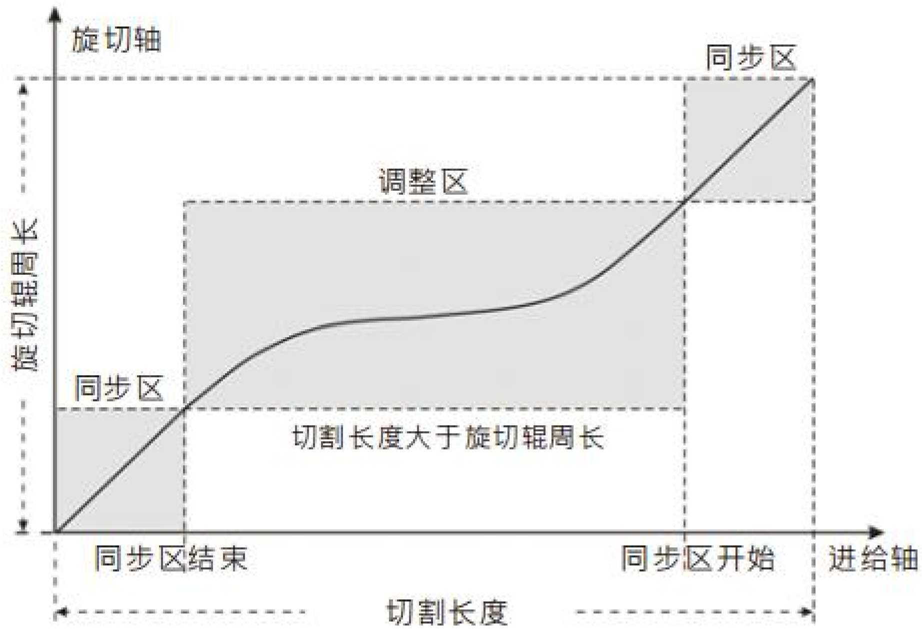
旋切长度等于刀辊周长时，任一周期的旋切曲线如下：



此情况下，同步区与非同步区进给轴和旋切轴一直保持速度同步，旋切轴不需要调整。

3:长料切割

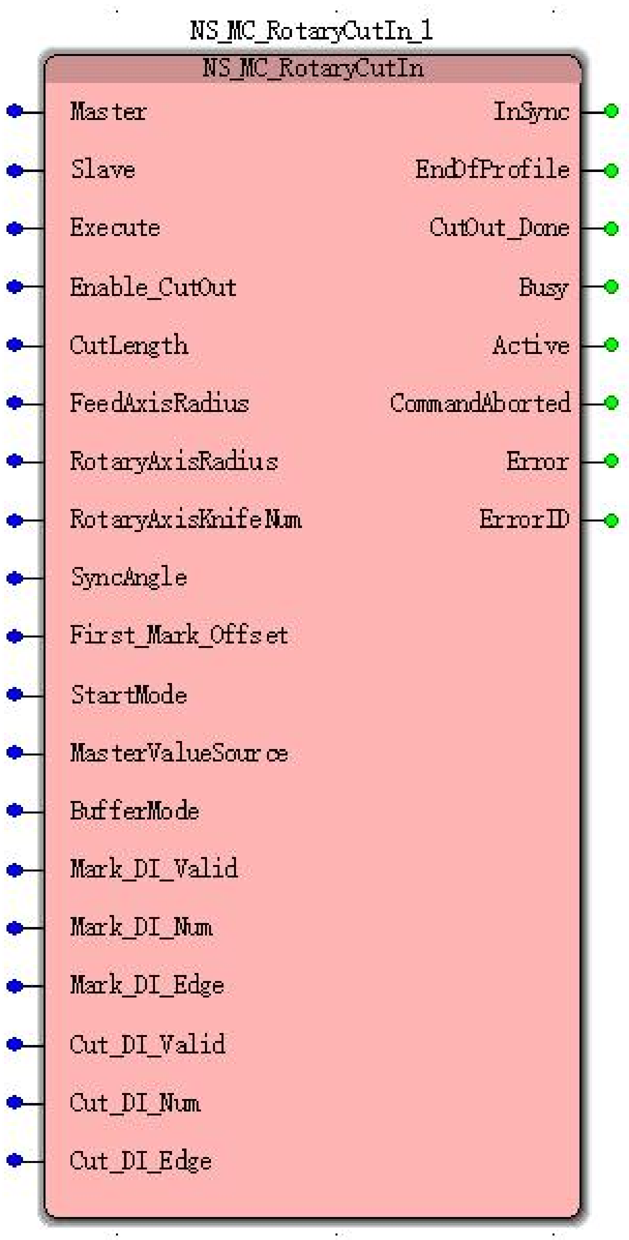
旋切长度大于刀辊周长时，任一周期的旋切曲线如下：



此情况下，旋切轴应该在调整区内先减速，然后再加速到同步速度。如果旋切长度远大于刀辊周长，则刀辊有可能减速到零，停留一段时间，然后再加速到同步速度。切割长度越长，停留的时间越长。

11.5.9 NS_MC_RotaryCutIn（旋切指令）

FB/FC	说明	适用机种
FB	此指令用于两个轴之间建立旋切关系	VEC-VA-MP-005-MA



➢ **输入参数**

名称	功能	数据类型	设定范围（缺省值）	生效时机
Master (主轴)	设定指令欲控制的主轴	USINT	**模拟量/脉冲:** 0~4（实轴） 5~11（虚轴） **CANopen 模式：** 0~15 (实轴/虚轴)	Exexcute 由 FALSE 变为 TRUE

			(0)	
Slave （从轴）	设定指令欲控制的从轴	USINT	0~4 实轴 5~11 虚轴 (CANopen 模式：0~15,可作实轴或虚轴) (0)	Exexcute 由 FALSE 变为 TRUE
Execute （执行位）	当 Execute 由 FALSE 变为 TRUE，执行该指令。	BOOL	TRUE 或 FALSE	-
Enable_CutOut （解除旋切位）	当 Enable_CutOut 为 TRUE 且凸轮处于周期末点，主从轴解除旋切关系，从轴停在切点处。	BOOL	TRUE 或 FALSE	-
CutLength (切长)	设定裁切长度（指的是进料主轴长度） 单位：单元	LREAL	正数	Exexcute 由 FALSE 变为 TRUE
FeedAxisRadius （进给轴半径）	设定进给轴的轴半径 单位：单元	LREAL	正数	Exexcute 由 FALSE 变为 TRUE
RotaryAxisRadius （旋切轴半径）	设定旋切轴半径，即旋切辊圆心到刀尖的距离 （单位：单元）	LREAL	正数	Exexcute 由 FALSE 变为 TRUE
RotaryAxisKnifeNum （旋切轴刀头数）	设定旋切轴的刀头数，即安装于旋切辊的刀头数量，当刀头数大于 1 时，需尽量确保两把刀之间的刀距一致。	USINT	正整数 （1~16）	Exexcute 由 FALSE 变为 TRUE
SyncAngle （同步角）	设定同步区范围（指的是旋切轴的角度） 单位：度	LREAL	0`360	Exexcute 由 FALSE 变为 TRUE
First_Mark_Offset （首个色标距离）	色标点到裁切点的距离，设置该值可实现首刀即可切到色标点上。 单位：单元	LREAL	正数、0	Exexcute 由 FALSE 变为 TRUE
StartMode （启动模式）	设定该指令的启动模式,详情见下模式说明.	INT	0~4	Exexcute 由 FALSE 变为 TRUE
MasterValueSource （主轴同步源）	从轴命令来源选择，当选择 StartMode=2、3、4 时，MasterValueSource 必须为 1。 0： 从轴跟随主轴的命令位置	INT	0、1	Exexcute 由 FALSE 变为 TRUE

	1：　从轴跟随主轴的实际位置			
BufferMode（交接模式）	设定两个指令之间交接模式 0：立即打断 1：等待	INT	0、1	Exexcute 由 FALSE 变为 TRUE
Mark_DI_Valid（色标信号有效位）	状态为 TRUE 色标信号有效；FALSE 色标信号无效；	BOOL	TRUE 或 FALSE	Exexcute 由 FALSE 变为 TRUE
Mark_DI_Num（色标点信号）	指定色标点端子号，输入值 0~7 对应输入点 I0.0~I0.7、8~15 对应输入 I1.0~I1.7	INT	0~15	Exexcute 由 FALSE 变为 TRUE
Mark_DI_Edge（色标点信号边沿）	设定色标点信号触发边沿 0：下降沿 1：上升沿	BOOL	TRUE 或 FALSE	Exexcute 由 FALSE 变为 TRUE
Cut_DI_Valid（裁切点信号有效位）	状态为 TRUE 裁切点信号有效；FALSE 裁切点信号无效；	BOOL	TRUE 或 FALSE	Exexcute 由 FALSE 变为 TRUE
Cut_DI_Num（裁切点信号）	指定裁切点点端子号，输入值 0~7 对应输入点 I0.0~I0.7、8~15 对应输入 I1.0~I1.7	INT	0~15	Exexcute 由 FALSE 变为 TRUE
Cut_DI_Edge（裁切点信号边沿）	设定裁切点信号触发边沿 0：下降沿 1：上升沿	BOOL	TRUE 或 FALSE	Exexcute 由 FALSE 变为 TRUE

说明：

- 对于进给轴和旋切轴，在设定 FeedAxisRadius（进给轴半径）、RotaryAxisRadius（旋切轴半径）前，应该在它们各自的轴参数模块 MC_AXIS_REF 上，将设置与其各自半径相符的 Disc_Circumference（圆盘周长）参数，以便于 MC_ReadActualPosition、MC_ReadActualVelocity 模块的使用。
- 当指令正在执行中，可以修改 CutLength、SyncAngle、FeedAxisRadius、RotaryAxisRaidus、RotaryAxisKnifeNum、First_Mark_Offset、Mark_DI_Valid、Cut_DI_Valid 的值后重新触发 Execute，修改的参数在触发 Execute 后的下一个凸轮周期生效。重新触发 Execute 并不会置 CommandAborted 为 TRUE；
- CANopen 模式下，带色标或切点功能无法使用。

➢ **输出参数**

名称	功能	数据类型	输出范围
InSync（达到同步状态）	该参数输出为 TRUE 时表示从轴达到同步状态	BOOL	TRUE 或 FALSE

EndOfProfile（旋切凸轮终点执行标志）	该参数输出为 TRUE 时表示凸轮终点被执行	BOOL	TRUE 或 FALSE
CutOut_Done(解除旋切完成)	该参数输出为 TRUE 时表示解除主从轴旋切关系完成	BOOL	TRUE 或 FALSE
Busy（执行中）	该参数输出为 TRUE 时表示指令正在执行中	BOOL	TRUE 或 FALSE
Active（控制中）	该参数输出为 TRUE 时表示指令正在控制轴	BOOL	TRUE 或 FALSE
CommandAborted（中断）	该参数输出为 TRUE 时表示指令被中断	BOOL	TRUE 或 FALSE
Error（错误）	该指令输出为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE
ErrorID（错误代码）	执行指令出错时的错误代码	WORD	-

模式说明

模式	说明
0	指令执行后，直接以当前点为切点同步速度启动跟随主轴。
1	指令执行后，以当前点为刀周一半的点（即切点的对立点）零速启动跟随主轴
2	指令执行后，触发色标信号，以当前点为刀周一半的点（即切点的对立点）零速启动跟随主轴。该模式下主轴必须为实轴，置 Mark_DI_Valid 为 Ture，且 Cut_DI_Valid 为 False
3	指令执行后，以当前点为刀周一半的点（即切点的对立点）零速启动跟随主轴，在同步区达到同步速度，等待切点触发，切点触发后，执行下一个凸轮周期。该模式下主轴必须为实轴，置 Cut_DI_Valid 为 Ture，且 Mark_DI_Valid 为 False
4	指令执行后，触发色标信号，以当前点为刀周一半的点（即切点的对立点）零速启动跟随主轴，等待切点触发，同时套色标，若色标的触发晚于切点的触发，即早切了，那么在色标触发的时候从轴会有一个回转补偿，确保下次能够套标。该模式下主轴必须为实轴，Cut_DI_Valid 和 Mark_DI_Valid 均置为 Ture

➢ **功能说明**

1、凸轮轴同样属于一种跟随运动，凸轮轴自身的位置是由我们设置的参数根据主轴位置自动规划出来的

2、轮切模块是连续型的凸轮曲线。只要参数规划正确，无需连续执行指令，可保持定长模式。（因为轮切位置清零点和曲线规划点是切点，而切点一般作用在从轴上）

3、在模式 3、4 下，切点信号不可缺少，否则从轴无法实现轮切电子凸轮的功能，切点既是从轴位置的起始点也是终点位置清零点，没有切点时，从轴将一直以同步速度运行，无法执行下一个凸轮周期；

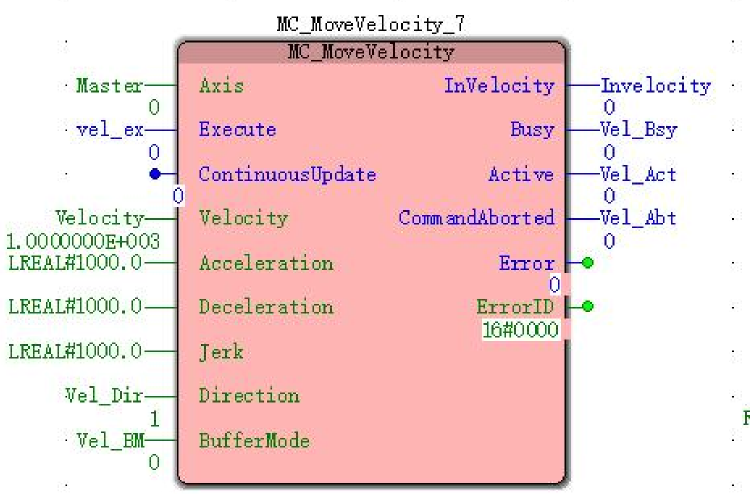
4、CANopen 模式下，只能使用模式 0、1。

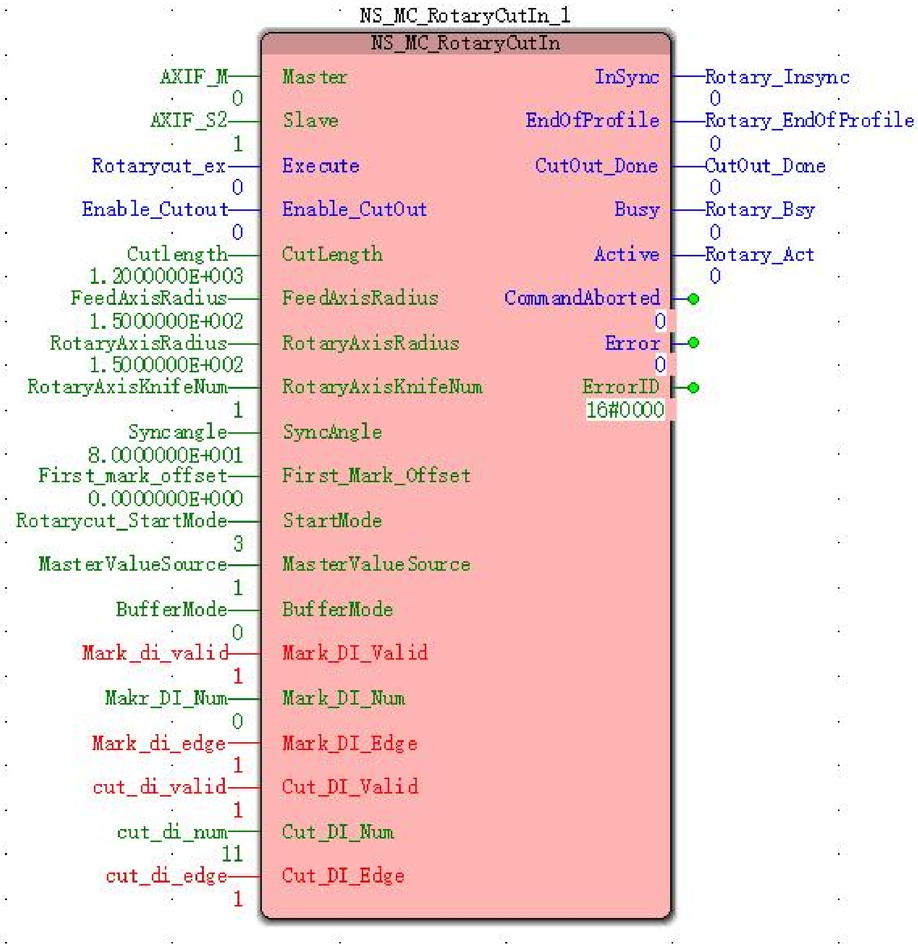
程序范例

1、变量和程序

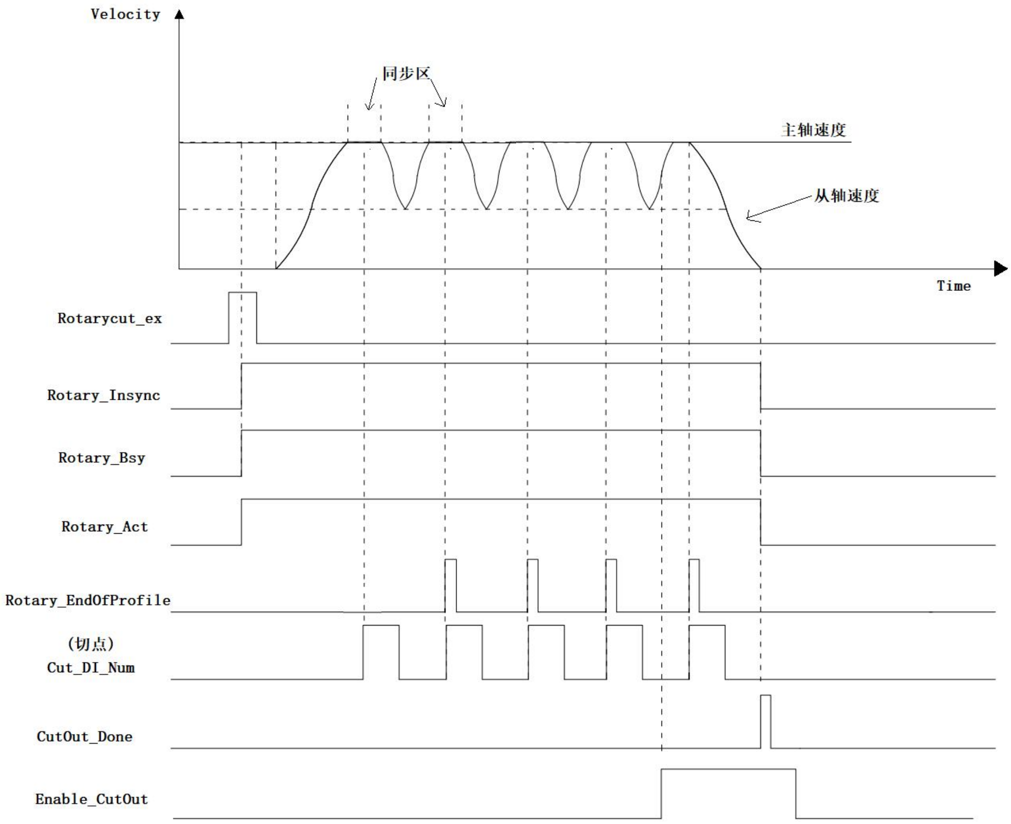
变量名	数据类型	初始值

MC_MoveVelocity_7	MC_MoveVelocity	
Master	USINT	0
Velocity	LREAL	1000.0
Vel_Dir	INT	1
Vel_BM	INT	0
NS_MC_RotaryCutIn_1	NS_MC_RotaryCutIn	
AXIF_M	USINT	0
AXIF_S2	USINT	1
Rotarycut_ex	BOOL	FALSE
Enable_Cutout	BOOL	FASLE
StartMode	INT	3
MasterValueSource	INT	1
BufferMode	INT	0
Mark_di_valid	BOOL	TRUE
Mark_DI_Num	INT	11
Mark_DI_Edge	BOOL	TRUE
Rotary_Insync	BOOL	
Rotary_EndOfProfile	BOOL	
CutOut_Done	BOOL	
Rotary_Bsy	BOOL	
Rotary_Act	BOOL	




NS_MC_RotaryCutIn_1
NS_MC_RotaryCutIn
AXIF_M
0
AXIF_S2
1
Rotarycut_ex
0
Enable_Cutout
0
Cutlength
1.2000000E+003
FeedAxisRadius
1.5000000E+002
RotaryAxisRadius
1.5000000E+002
RotaryAxisKnifeNum
1
Syncangle
8.0000000E+001
First_mark_offset
0.0000000E+000
Rotarycut_StartMode
3
MasterValueSource
1
BufferMode
0
Mark_di_valid
1
Makr_DI_Num
0
Mark_di_edge
1
cut_di_valid
1
cut_di_num
11
cut_di_edge
1
Master
Slave
Execute
Enable_CutOut
CutLength
FeedAxisRadius
RotaryAxisRadius
RotaryAxisKnifeNum
SyncAngle
First_Mark_Offset
StartMode
MasterValueSource
BufferMode
Mark_DI_Valid
Mark_DI_Num
Mark_DI_Edge
Cut_DI_Valid
Cut_DI_Num
Cut_DI_Edge
InSync
EndOfProfile
CutOut_Done
Busy
Active
CommandAborted
0
Error
0
ErrorID
16#0000
Rotary_Insync
0
Rotary_EndOfProfile
0
CutOut_Done
0
Rotary_Bsy
0
Rotary_Act
0

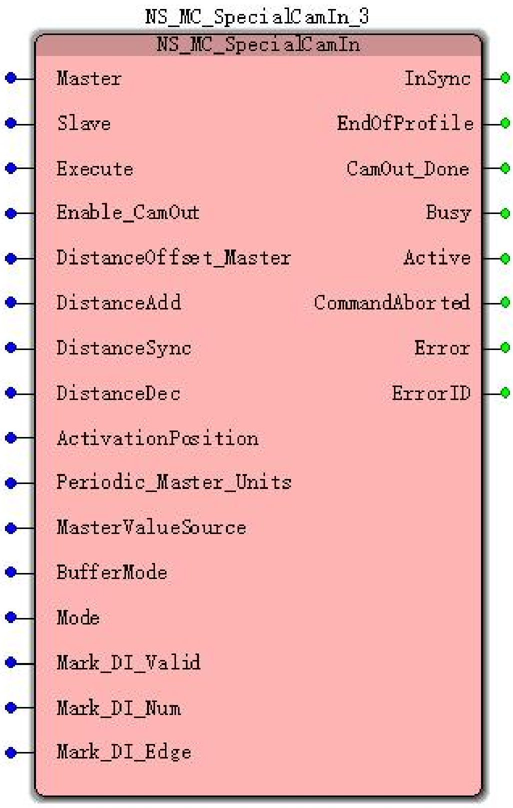
2、运动曲线和时序图



- 以 StartMode=3 执行指令，Execute 变为 TRUE，一个周期后，InSync、Busy、Active 变为 TRUE,从轴柔性启动到达同步区，触发切点信号，凸轮进入下一个周期。而后每触发一次切点信号，输出 EndOfProfile 信号 TRUE 一个周期，若切点信号一直未触发，则从轴以同步线速度一直运行。
- Enable_CutOut 变为 TRUE，触发切点信号从轴解除旋切关系并停在切点的对立点。

11.5.10 NS_MC_SpecialCamin（特殊凸轮指令）

FB/FC	说明	适用机种
FB	此指令用于两个轴之间建立特殊凸轮关系	VEC-VA-MP-005-MA



➢ **输入参数**

名称	功能	数据类型	设定范围 （缺省值）	生效时机
Master (主轴)	设定指令欲控制的主轴	USINT	**模拟量/脉冲：** 0~4（实轴） 5~11（虚轴） **CANopen 模式：** 0~15 (实轴/虚轴) (0)	Exexcute 由 FALSE 变为 TRUE
Slave （从轴）	设定指令欲控制的从轴	USINT	0~4 实轴 5~11 虚轴 (CANopen 模式：0~15,可作实轴或虚轴) (0)	Exexcute 由 FALSE 变为 TRUE
Execute （执行位）	当 Execute 由 FALSE 变为 TRUE，指令开始执行	BOOL	TRUE 或 FALSE	-

Enable_CamOut (特殊凸轮关系解除)	当 Enable_CamOut 为 TRUE，下一个凸轮周期到来即将主从轴特殊凸轮关系解除	BOOL	TRUE 或 FALSE	
DistanceOffset_Master （主轴位置偏移）	设定凸轮起点偏移主轴的距离	LREAL	正数	Exexcute 由 FALSE 变为 TRUE
DistanceAdd （加速距离）	从轴由静止达到同步区的加速距离	LREAL	正数	Exexcute 由 FALSE 变为 TRUE
DistanceSync （同步距离）	从轴在到同步区的同步运行距离	LREAL	正数、0	Exexcute 由 FALSE 变为 TRUE
DistanceDec （减速距离）	从轴由同步区达到静止的减速距离	LREAL	正数	Exexcute 由 FALSE 变为 TRUE
ActivationPosition （啮合启动位置）	设定啮合过程开始是的主轴位置，即当主轴通过该位置时，从轴开始执行啮合动作。模式 0、1 有效。（啮合启动位置为相对位置）	LREAL	正数、0	Exexcute 由 FALSE 变为 TRUE
Periodic_Master_Units （凸轮周期主轴单元数）	模式 0 下： Periodic_Master_Units = DistanceOffset_Master +DistanceAdd*30/16 + DistanceSync + DistanceDec*30/16 模式 1、2 下： 第一次触发色标信号，从轴执行完成凸轮指令后，在主轴未跑完此项设置的单元数期间触发色标信号，从轴不执行凸轮动作。	LREAL	正数、零	Exexcute 由 FALSE 变为 TRUE
MasterValueSource （位置来源选择）	从轴命令来源选择，当选择 Mode=1、2 时，MasterValueSource 必须为 1。 0：　从轴跟随主轴的命令位置 1：　从轴跟随主轴的实际位置	INT	0、1	Exexcute 由 FALSE 变为 TRUE
BufferMode （交接模式）	设定两个指令之间交接模式	INT	0、1	Exexcute 由 FALSE 变为 TRUE

	0：立即打断 1：等待			
Mode （指令执行模式）	设定该指令的启动模式，详情见下模式说明.	INT	0~2	Exexcute 由 FALSE 变为 TRUE
Mark_DI_Valid （色标信号有效位）	状态为 TRUE 色标信号有效；FALSE 色标信号无效；	BOOL	TRUE 或 FALSE	Exexcute 由 FALSE 变为 TRUE
Mark_DI_Num （色标信号）	指定色标点端子号，输入值 0~7 对应输入点 I0.0~I0.7、8~15 对应输入 I1.0~I1.7	INT	0~15	Exexcute 由 FALSE 变为 TRUE
Mark_DI_Edge （色标信号边沿）	设定色标点信号触发边沿 0：下降沿 1：上升沿	BOOL	TRUE 或 FALSE	Exexcute 由 FALSE 变为 TRUE

➢ **输出参数**

名称	功能	数据类型	输出范围
InSync（达到同步状态）	该参数输出为 TRUE 时表示从轴达到同步状态	BOOL	TRUE 或 FALSE
EndOfProfile（凸轮终点执行标志）	该参数输出为 TRUE 时表示凸轮终点被执行	BOOL	TRUE 或 FALSE
CutOut_Done(解除旋切完成)	该参数输出为 TRUE 时表示解除主从轴旋切关系完成	BOOL	TRUE 或 FALSE
Busy（执行中）	该参数输出为 TRUE 时表示指令正在执行中	BOOL	TRUE 或 FALSE
Active（控制中）	该参数输出为 TRUE 时表示指令正在控制轴	BOOL	TRUE 或 FALSE
CommandAborted（中断）	该参数输出为 TRUE 时表示指令被中断	BOOL	TRUE 或 FALSE
Error（错误）	该指令输出为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE
ErrorID（错误代码）	执行指令出错时的错误代码	WORD	-

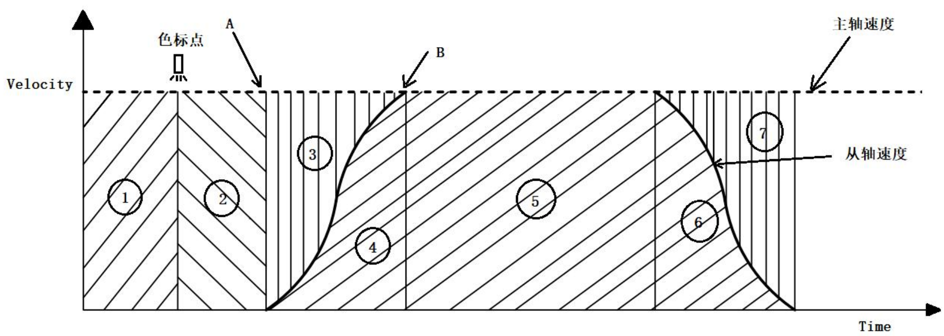
➢ **模式说明**

模式	说明
0	当指令执行后，在主轴到达 ActivationPosition 时开始啮合，从轴以 Periodic_Master_Units 为周期循环运动，该模式下需将 Mark_DI_Valid 置为 False
1	当指令执行后，在主轴到达 ActivationPosition 时开始啮合，以 Periodic_Master_Units 执行一个凸轮周期，而后每触发一个色标，从轴根据 DistanceOffset_Master、DistanceAdd、DistanceSync、DistanceDec 的设定完成一个凸轮周期，该模式下需将 Mark_DI_Valid 置为 Ture
2	当指令执行后，每触发一个色标，从轴根据 DistanceOffset_Master、

	DistanceAdd、DistanceSync、DistanceDec 的设定完成一个凸轮周期，该模式下需将 Mark_DI_Valid 置为 Ture
3	基于 0 模式，ActivationPosition 为绝对位置启动，参考 MC_CamIn 主轴绝对位置启动
4	基于 0 模式，ActivationPosition 为绝对模位置启动，绝对模位置启动，根据模的绝对位置启动

➢ **功能说明**

- 当指令正在执行中，可以修改 DistanceOffset_Master、DistanceAdd、DistanceSync DistanceDec、Periodic_Master_Units 的值后重新触发 Execute，修改的参数在触发 Execute 后的下一个凸轮周期生效。修改 Mark_DI_Valid、Mark_DI_Num、Mark_DI_Edge 的值后，修改的参数重新触发 Execute 立即生效。重新触发 Execute 并不会置 CommandAborted 为 TRUE；
- 模式 1、2 下，若上一个凸轮周期未执行完成，再次触发色标信号并不会立即开始下一个凸轮周期；
- CANopen 模式下，带色标点功能无法使用；
- MC_SpecialCamIn 指令用于控制从轴按照预先规划的凸轮关系与主轴实现凸轮同步运动；
- 电子凸轮的主轴来源可选择实轴（AXIS0~AXIS3） 或虚轴或主轴（AXIS4）；
- 电子凸轮指令根据参数 DistanceOffset_Master、AvtivationPosition，DistanceAdd、DistanceSync、DistanceDec，电子凸轮曲线将自动生成；凸轮位置与主轴关系如下图所示；



（1）上图表示主轴与电子凸轮时刻的速度位置关系：

（2）①表示 ActivationPosition，②表示 DistanceOffset_Master，④表示电子凸轮设定加速距离，⑤表示电子凸轮匀速距离（电子凸轮从轴与主轴线速度相等），⑥表示电子凸轮设定减速距离。其中，③与④、⑥与⑦的面积比皆为 14:16。

可以通过该比例求解一段从轴对主轴追速所需加速距离。
例：假设要求主轴在 A 运动到 B 的位置后（距离为
100 Units），电子凸轮从轴与主轴线速度同步，则电子凸轮的加速距离应设置为多少？ 解：设定
电子凸轮加速距离为 X,则 X=100*16/30；

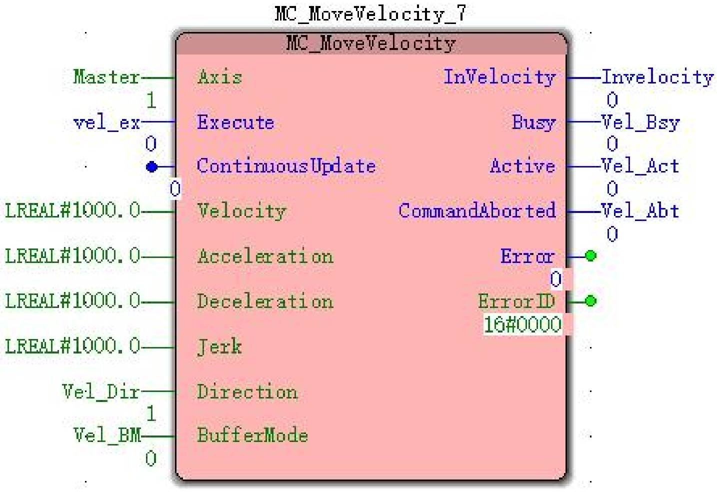
- NS_MC_SpecialCamIn 模式 1 和模式 2 硬件资源上实际只有 4 个轴的，分为 4 组。
  轴 0，4，8，12 为一组，
  轴 1，5，9，13 为一组，
  轴 2，6，10，14 为一组，
  轴 3，7，11，15 为一组。
  理论上同一组的 4 个轴的主轴不同时，同一时刻只能用一个；主轴相同时，同一时刻可以任意使用同组的一个或多个从轴。比如 0 轴的主轴为 5 轴，4 轴的主轴为 6 轴，则 0 轴和 4 轴作为从轴的凸轮指令不能同时触发。但是，当同组的轴作从轴时都是用同一个色标点、同一个主轴的情况下，可以同时用模式 1 和模式 2。例如前缘用的凸轮，凸轮从轴都是同一个色标点、同一个主轴，所以同一组的轴可以同时用模式 2。

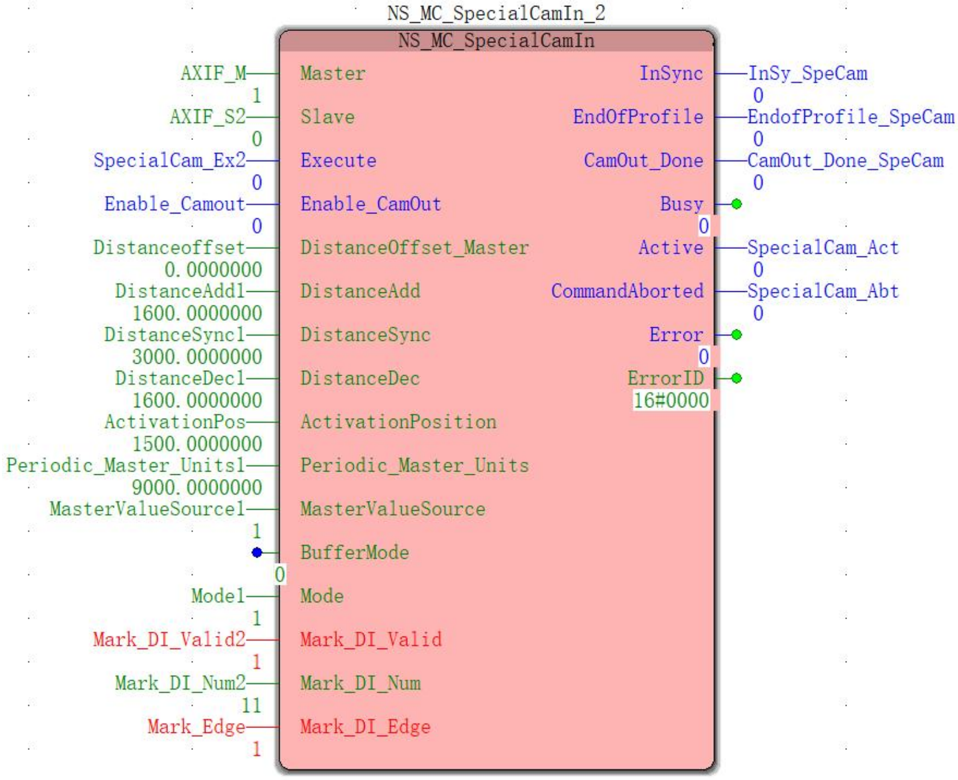
程序范例

以模式 1 执行 NS_MC_SpecialCamin 指令范例如下所示。

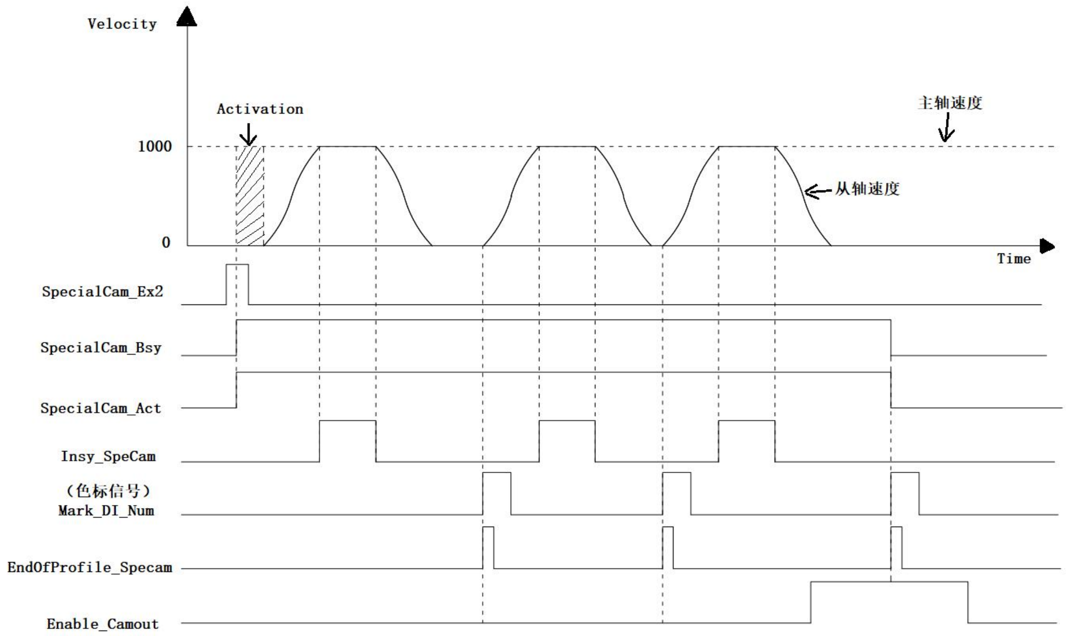
1、程序和变量

变量名	数据类型	初始值
MC_MoveVelocity_7	MC_MoveVelocity	
Master	USINT	1
Vel_Dir	INT	1
Vel_BM	INT	0
NS_MC_SpecialCamIn_3	NS_MC_SpecialCamIn	
AXIF_M	USINT	1
AXIF_S2	USINT	0
SpecialCam_Ex2	BOOL	FALSE
Enable_Camout	BOOL	FALSE
Distanceoffset	LREAL	0.0
DistanceAdd1	LREAL	1600.0
DistanceSync1	LREAL	3000.0
DistanceDec1	LREAL	1600.0
ActivationPos	LREAL	1500.0
Periodic_Master_Units1	LREAL	9000.0
MasterValueSource1	INT	1
Mode1	INT	1
Mark_DI_Valid2	BOOL	TRUE
Mark_DI_Num2	INT	11
Mark_DI_Edge2	BOOL	TRUE
Insy_SpeCam	BOOL	
endOfProfile_specam	BOOL	
CamOut_done_SpeCam	BOOL	
SpecialCam_Bsy	BOOL	
SpecialCam_Act	BOOL	
SpecialCam_Abt	BOOL	


MC_MoveVelocity_7
MC_MoveVelocity
Master
Axis
InVelocity
Invelocity
vel_ex
Execute
Busy
Vel_Bsy
ContinuousUpdate
Active
Vel_Act
LREAL#1000.0
Velocity
CommandAborted
Vel_Abt
LREAL#1000.0
Acceleration
Error
LREAL#1000.0
Deceleration
ErrorID
16#0000
LREAL#1000.0
Jerk
Vel_Dir
Direction
Vel_BM
BufferMode


NS_MC_SpecialCamIn_2
NS_MC_SpecialCamIn
AXIF_M
Master
InSync
InSy_SpeCam
AXIF_S2
Slave
EndOfProfile
EndofProfile_SpeCam
SpecialCam_Ex2
Execute
CamOut_Done
CamOut_Done_SpeCam
Enable_Camout
Enable_CamOut
Busy
Distanceoffset
DistanceOffset_Master
Active
SpecialCam_Act
0.0000000
DistanceAdd1
DistanceAdd
CommandAborted
SpecialCam_Abt
1600.0000000
DistanceSync1
DistanceSync
Error
3000.0000000
DistanceDec1
DistanceDec
ErrorID
1600.0000000
16#0000
ActivationPos
ActivationPosition
1500.0000000
Periodic_Master_Units1
Periodic_Master_Units
9000.0000000
MasterValueSource1
MasterValueSource
BufferMode
Mode1
Mode
Mark_DI_Valid2
Mark_DI_Valid
Mark_DI_Num2
Mark_DI_Num
11
Mark_Edge
Mark_DI_Edge

二、运动曲线和时序图



- 以 Mode=1 执行 NS_MC_SpecialCamIn 指令，Execute 变为 TRUE，一个周期后，Busy、Active 变为 TRUE，以此时为起点，当主轴到达 Activation，从轴按照设定的加速距离走向同步区，在同步区内，InSync 输出 TRUE，随后完成同步距离和减速距离结束第一个同步周期。
- 在完成第一个凸轮周期后，触发色标信号，从轴立即根据设定参数生成的凸轮表进行同步运动。
- 设 Enable_Camout 为 TRUE，在下一个色标信号到来时，解除主从轴的凸轮关系。

11.5.11 NS_MC_SpecialCombineAxes（特殊双主轴联合齿轮指令）

FB/FC	说明	适用机种
FB	此指令用于两个轴之间建立特殊双主轴联合齿轮关系	VEC-VA-MP-005-MA

➢ **输入参数**

名称	功能	数据类型	设定范围 （缺省值）	生效时机
Master (主轴)	设定指令欲控制的主轴	USINT	**模拟量/脉冲:** 0~4（实轴） 5~11（虚轴） **CANopen 模式：** 0~15 (实轴/虚轴)	Exexcute 由 FALSE 变为 TRUE

			(0)	
Slave （从轴）	设定指令欲控制的从轴	USINT	0~4 实轴 5~11 虚轴 (CANopen 模式：0~15,可作实轴或虚轴) (0)	Exexcute 由 FALSE 变为 TRUE
Execute （执行位）	当 Execute 由 FALSE 变为 TRUE，指令执行	BOOL	TRUE 或 FASLE	
Execute_Precaculate (预计算执行位)	当 Execute_Precaculate 为 TRUE 时，计算 Pos_Min 和 Pos_Max 的值	BOOL	TRUE 或 FASLE	
Precaculate_Pulse_Cycle （预计算主轴周期脉冲数）	该参数用于计算 Pos_Min 和 Pos_Max Precalculate_Pulse_Cycle=Periodic_Master_Units*Cam_Pulse_Per_Unit_M 单位：Pulse	LREAL	正数	当 Execute_Precaculate 由 FALSE 变为为 TRUE
RatioNumeratorM1 （主轴 1 齿轮比分子）	设定主轴 1 齿轮比分子	LREAL	正数或负数 （不可缺省）	Exexcute 由 FALSE 变为 TRUE
RatioDenominatorM1 （主轴 1 齿轮比分母）	设定主轴 1 齿轮比分母	LREAL	正数或负数 （不可缺省）	Exexcute 由 FALSE 变为 TRUE
RatioNumeratorM2 （主轴 2 齿轮比分子）	设定主轴 2 齿轮比分子	LREAL	正数或负数 （不可缺省）	Exexcute 由 FALSE 变为 TRUE
RatioDenominatorM2 （主轴 2 齿轮比分母）	设定主轴 2 齿轮比分母	LREAL	正数或负数 （不可缺省）	Exexcute 由 FALSE 变为 TRUE
Gear_RatioNumerator （电子齿轮分子）	MC_GearIn 的电子齿轮比分子	LREAL	正数、负数 （不可缺省）	Exexcute 由 FALSE 变为 TRUE
Gear_RatioDenominator （电子齿轮分母）	MC_GearIn 的电子齿轮比分母	LREAL	正数、负数 （不可缺省）	Exexcute 由 FALSE 变为 TRUE
Cam_DistanceOffset_Master （主轴位置偏差）	单位：单元	LREAL	正数、零 （0）	Exexcute 由 FALSE 变为 TRUE
Cam_DistanceAdd （加速距离）	NS_MC_SpecialCamIn 由静止达到同步区的加速距离	LREAL	正数 （不可缺省）	Exexcute 由 FALSE 变为 TRUE

Cam_DistanceSync （同步距离）	NS_MC_SpecialCamIn 在同步区的同步运行距离	LREAL	正数、零 （0）	Exexcute 由 FALSE 变为 TRUE
Cam_DistanceDec （减速距离）	NS_MC_SpecialCamIn 由同步区达到静止的减速距离	LREAL	正数 （不可缺省）	Exexcute 由 FALSE 变为 TRUE
Cam_Pulse_Per_Unit_M （主轴每单元脉冲数）	MC_SpecialCamIn 根据此值确定凸轮曲线的规划，如设置此值为 5，当主轴走 10000 个脉冲，此时认为主轴的终端执行机构移动了 2000 个单元的位置。	LREAL	正数	Exexcute 由 FALSE 变为 TRUE
Periodic_Master_Units （主轴凸轮周期单元数）	Periodic_Master_Units = DistanceOffset_Master +DistanceAdd*30/16 + DistanceSync+ DistanceDec*30/16	LREAL	正数 （不可缺省）	Exexcute 由 FALSE 变为 TRUE
MasterValueSource （位置来源选择）	从轴命令来源选择 0： 从轴跟随主轴的命令位置 1： 从轴跟随主轴的实际位置 （模式 1 下该值必须为 1）	INT	0、1	Exexcute 由 FALSE 变为 TRUE
BufferMode （交接模式）	设定两个指令之间交接模式 0：立即打断 1：等待	INT	0、1	Exexcute 由 FALSE 变为 TRUE
Mode （模式）	指令执行模式 模式 0：从轴对主轴 1 与主轴 2 的位置变化进行叠加追随 模式 1：咨询我司技术人员	INT	0、1	Exexcute 由 FALSE 变为 TRUE
Mark_DI_Valid （色标有效位）	状态为 TRUE 色标信号有效；FALSE 色标信号无效；	BOOL	TRUE 或 FALSE	Exexcute 由 FALSE 变为 TRUE
Mark_DI_Num （色标信号）	指定色标点端子号，输入值 0~7 对应输入点 I0.0~I0.7、8~15 对应输入 I1.0~I1.7	INT	0~15	Exexcute 由 FALSE 变为 TRUE
Mark_DI_Edge （色标信号边沿）	设定色标点信号触发边沿 0：下降沿 1：上升沿	BOOL	TRUE 或 FALSE	Exexcute 由 FALSE 变为 TRUE

➢ **功能说明：**

- 当指令正在执行中，可以修改 RatioNumeratorM1、RatioDenominatorM1、RatioNumeratorM2、RatioDenominatorM2、Gear_RatioNumerator、Gear_RatioDenominator、Cam_DistanceOffset_Master、Cam_DistanceAdd、Cam_DistanceSync、Cam_DistanceDec、Cam_Pulse_Per_Unit_M、Periodic_Master_Units
  的值后重新触发 Execute，修改的参数在触发 Execute 后的下一个凸轮周期生效。重新触发 Execute 并不会置 CommandAborted 为 TRUE；
- CANopen 模式下，带色标点功能无法使用；
- NS_MC_SpecialCombineAxes 指令可视为以下两部分的叠加：
  ①从轴通过 MC_GearIn 指令跟随主轴运动，本文称其为主轴 1
  ②从轴通过 NS_MC_SpecialCamIn 指令跟随主轴运动，本文称其为主轴 2
  因此，

$$\text{从轴位置变化量} = \text{主轴 1 位置变化量} * \frac{\text{主轴 1 齿轮比分子}}{\text{主轴 1 齿轮比分母}} + \text{主轴 2 位置变化量} * \frac{\text{主轴 2 齿轮比分子}}{\text{主轴 2 齿轮比分母}}$$

- **输出参数**

名称	功能	数据类型	输出范围
InSync（达到同步状态）	该参数输出为 TRUE 时表示从轴达到同步状态	BOOL	TRUE 或 FALSE
Busy（执行中）	该参数输出为 TRUE 时表示指令正在执行中	BOOL	TRUE 或 FALSE
Active（控制中）	该参数输出为 TRUE 时表示指令正在控制轴	BOOL	TRUE 或 FALSE
CommandAborted（中断）	该参数输出为 TRUE 时表示指令被中断	BOOL	TRUE 或 FALSE
Error（错误）	该指令输出为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE
ErrorID（错误代码）	执行指令出错时的错误代码	WORD	-
Pos_Min（最小位置）	一个执行周期内，从轴距离这个执行周期起点的最小位置 单位：Unit	LREAL	
Pos_Max（最大位置）	一个执行周期内，从轴距离这个执行周期起点的最大位置 单位：Unit	LREAL	

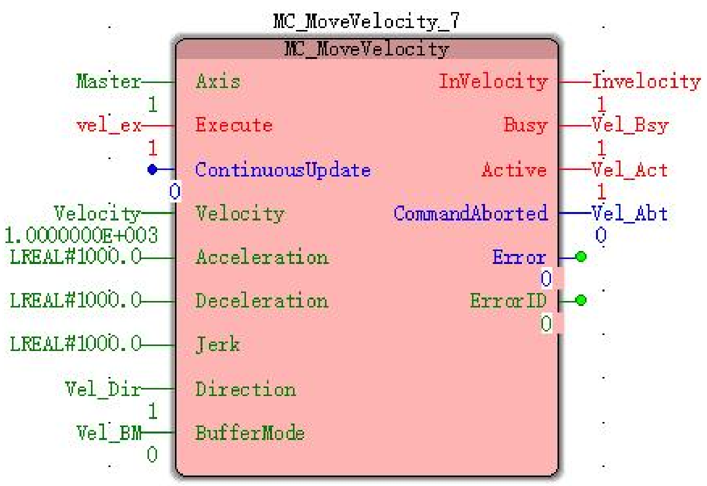
程序范例

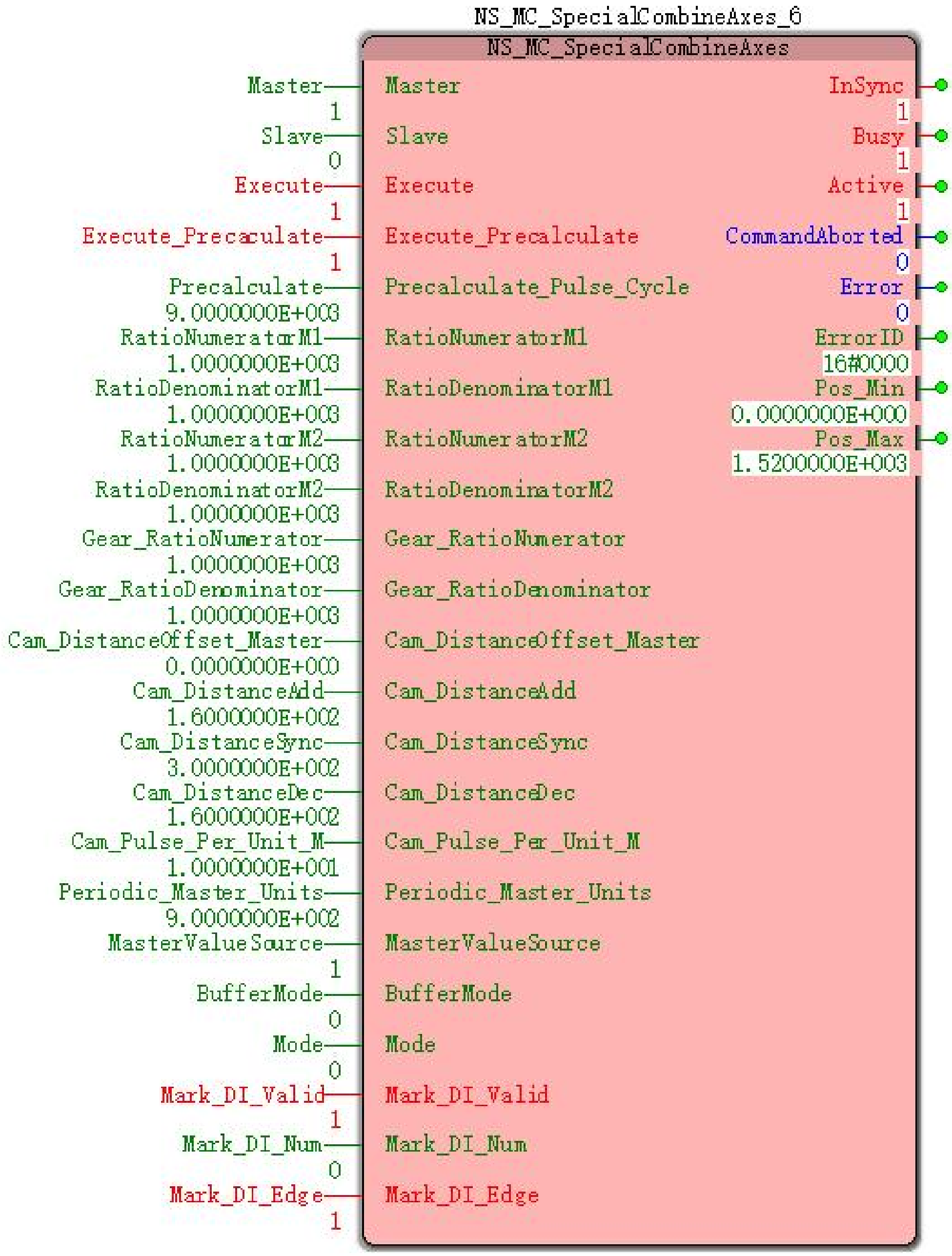
以模式 0 执行 NS_MC_SpecialCamIn 指令

1、程序和变量

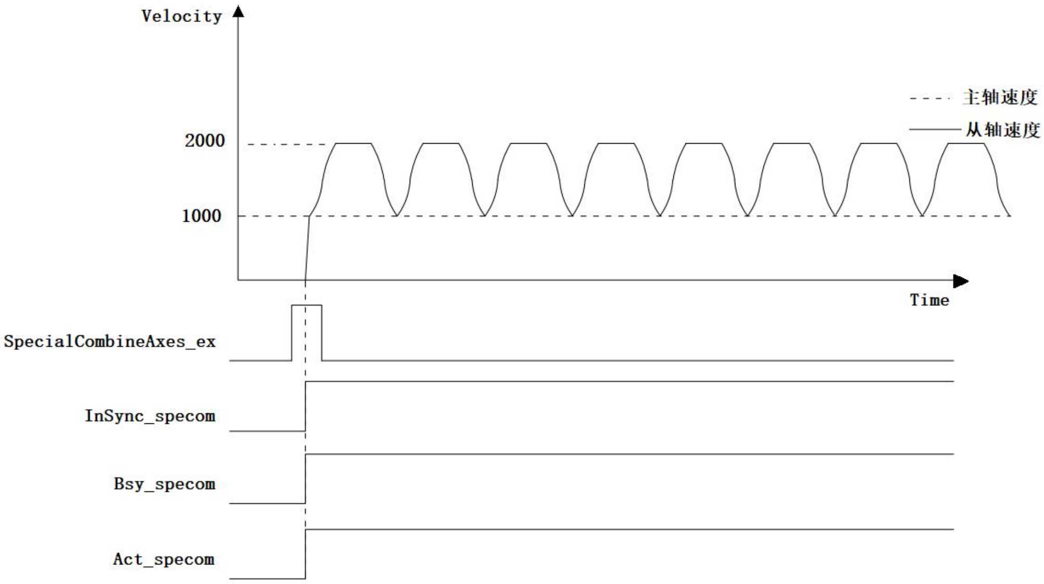
变量名	数据类型	初始值
MC_MoveVelocity_7	MC_MoveVelocity	
Master	USINT	1

Vel_Dir	INT	1
Vel_BM	INT	0
Master	USINT	1
Slave	USINT	0
Execute	BOOL	FASLE
Execute_Precalculate	BOOL	FALSE
Precalculate	LREAL	9000.0
RatioNumeratorM1	LREAL	1000.0
RatioDenominatorM1	LREAL	1000.0
RatioNumeratorM2	LREAL	1000.0
RatioDenominatorM2	LREAL	1000.0
Gear_RatioNumerator	LREAL	1000.0
Gear_RatioDenominator	LREAL	1000.0
Cam_DistanceOffset_Master	LREAL	0.0
Cam_DistanceAdd	LREAL	160.0
Cam_DistanceSync	LREAL	300.0
Cam_DistanceDec	LREAL	160.0
Cam_Pulse_Per_Unit_M	LREAL	10.0
Periodic_Master_Units	LREAL	900.0
MasterValueSource	INT	0
Mode_specom	INT	0
InSync_specom	BOOL	
Bsy_specom	BOOL	
Act_specom	BOOL	
Abt_specom	BOOL	




NS_MC_SpecialCombineAxes_6
NS_MC_SpecialCombineAxes
Master
1
Slave
0
Execute
1
Execute_Precaculate
1
Precalculate
9.0000000E+003
RatioNumeratorM1
1.0000000E+003
RatioDenominatorM1
1.0000000E+003
RatioNumeratorM2
1.0000000E+003
RatioDenominatorM2
1.0000000E+003
Gear_RatioNumerator
1.0000000E+003
Gear_RatioDenominator
1.0000000E+003
Cam_DistanceOffset_Master
0.0000000E+000
Cam_DistanceAdd
1.6000000E+002
Cam_DistanceSync
3.0000000E+002
Cam_DistanceDec
1.6000000E+002
Cam_Pulse_Per_Unit_M
1.0000000E+001
Periodic_Master_Units
9.0000000E+002
MasterValueSource
1
BufferMode
0
Mode
0
Mark_DI_Valid
1
Mark_DI_Num
0
Mark_DI_Edge
1
Master
Slave
Execute
Execute_Precalculate
Precalculate_Pulse_Cycle
RatioNumeratorM1
RatioDenominatorM1
RatioNumeratorM2
RatioDenominatorM2
Gear_RatioNumerator
Gear_RatioDenominator
Cam_DistanceOffset_Master
Cam_DistanceAdd
Cam_DistanceSync
Cam_DistanceDec
Cam_Pulse_Per_Unit_M
Periodic_Master_Units
MasterValueSource
BufferMode
Mode
Mark_DI_Valid
Mark_DI_Num
Mark_DI_Edge
InSync
1
Busy
1
Active
1
CommandAborted
0
Error
0
ErrorID
16#0000
Pos_Min
0.0000000E+000
Pos_Max
1.5200000E+003

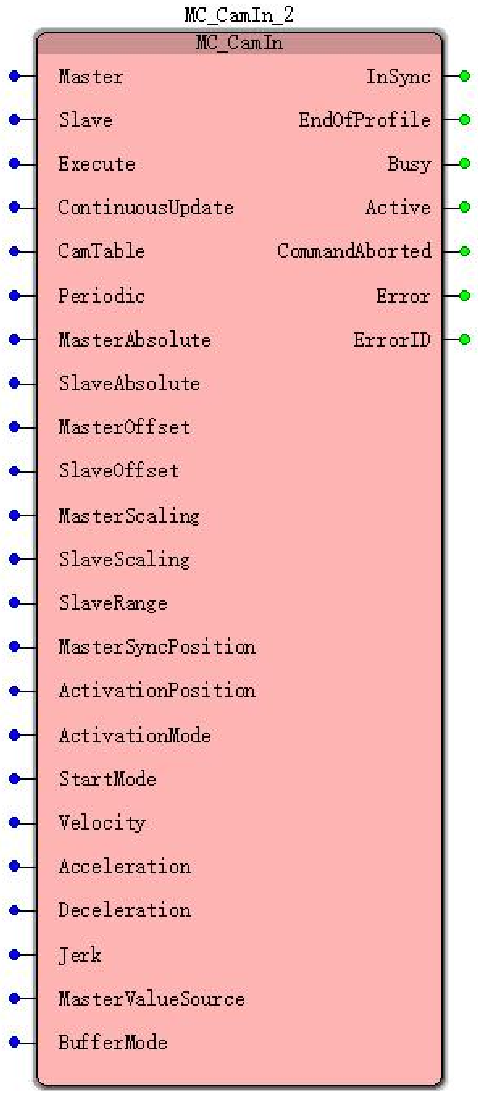
2、时序和曲线图



- 主轴速度 1000，执行 NS_MC_SpecialCombineAxes，从轴根据凸轮表跟随主轴位置变化做出运动。

11.5.12 MC_CamIn（电子凸轮关联指令）

FB/FC	说明	适用机种
FB	此指令用于两个轴之间建立电子凸轮关系	VEC-VA-MP-005-MA



➢ **输入参数**

名称	功能	数据类型	设定范围 （缺省值）	生效时机
Master (主轴)	设定电子凸轮的主轴	USINT	**模拟量/脉冲：** 0~4（实轴） 5~11（虚轴） **CANopen 模式：** 0~15 (实轴/虚轴) (0)	Exexcute 由 FALSE 变为 TRUE
Slvae （从轴）	设定电子凸轮的从轴	USINT	0~4 实轴 5~11 虚轴 (CANopen 模式：	Exexcute 由 FALSE 变为 TRUE

			0~15,可作实轴或虚轴) (0)	
Execute （执行位）	当 Execute 由 FALSE 变为 TRUE，指令开始执行	BOOL	TRUE 或 FALSE	
ContinousUpdate （保留）				
CamTable （电子凸轮表编号）	用于设定主从轴建立凸轮关系所依据的凸轮表	USINT	0~31	Exexcute 由 FALSE 变为 TRUE
Periodic （周期运动型）	设定电子凸轮为周期运行或仅运行一个周期	BOOL	TRUE 或 FALSE	Exexcute 由 FALSE 变为 TRUE
MasterAbsolute （主轴绝对）	设定主轴的位置模式：为 TRUE 时，主轴位置为绝对模式；为 FALSE 时，主轴位置为相对模式	BOOL	TRUE 或 FALSE	Exexcute 由 FALSE 变为 TRUE
SlaveAbsolute （从轴绝对）	设定从轴的位置模式：为 TRUE 时，主轴位置模式为绝对模式；为 FALSE 时，主轴位置模式为相对模式	BOOL	TRUE 或 FALSE	Exexcute 由 FALSE 变为 TRUE
MasterOffset （主轴位置偏移量）	设定主轴位置偏移量 （单位：单元）	LREAL	正数、负数、0（0）	Exexcute 由 FALSE 变为 TRUE
SlaveOffset （从轴位置偏移量）	设定从轴位置偏移量 （单位：单元）	LREAL	正数、负数、0（0）	Exexcute 由 FALSE 变为 TRUE
MasterScaling （主轴位置缩放比）	设定主轴位置的缩放比例	LREAL	正数 （不可缺省）	Exexcute 由 FALSE 变为 TRUE
SlaveScaling （从轴位置缩放比）	设定从轴位置的缩放比例	LREAL	正数 （不可缺省）	Exexcute 由 FALSE 变为 TRUE
SlaveRange （从轴凸轮相位范围）	设定从轴凸轮相位的范围	LREAL		
MasterSyncPosition （保留）	保留			
ActivationPosition （啮合启动位置）	设定啮合过程开始时的主轴位置，即当主轴经过该位置时，从轴开始执行啮合动作 （单位：单元）	LREAL	正数、负数、0（0）	Exexcute 由 FALSE 变为 TRUE
ActivationMode （启动模式的模式）	设定啮合启动位置的模式	INT	0：相对轴位置 1：绝对轴位置 2：绝对轴相位	Execute 由 FALSE 变为 TRUE

			3：绝对凸轮相位 （0）	
StartMode （啮合模式）	设定从轴执行啮合动作的方式	INT	0：最短距离 1：正向 2：反向	Exexcute 由 FALSE 变为 TRUE
Velocity （速度）	设定从轴执行啮合动作过程中允许的最大叠加速度 （单位：单元/S）	LREAL	正数 （不可缺省）	Exexcute 由 FALSE 变为 TRUE
Acceleration （加速度）	设定从轴执行啮合动作过程中允许的最大叠加加速度 （单位：单元/S²）	LREAL	正数 （不可缺省）	Exexcute 由 FALSE 变为 TRUE
Deceleration （减速度）	设定从轴执行啮合动作过程中允许的最大叠加减速度 （单位：单元/S²）	LREAL	正数 （不可缺省）	Exexcute 由 FALSE 变为 TRUE
Jerk （加/减速度的变化率）	保留			
MasterValueSource （主轴位置来源）	设定电子凸轮计算过程中主轴位置的类型	INT	0： 从轴跟随主轴的命令位置 1： 从轴跟随主轴的实际位置	Exexcute 由 FALSE 变为 TRUE
BufferMode （交接模式）	设定两个指令之间的交接模式	INT	0:立即打断 1：等待 (0)	Exexcute 由 FALSE 变为 TRUE

说明：

1、该指令在 Execute 由 FALSE 变为 TRUE 时开始执行。该指令正在执行，Execute 由 TRUe 变为 FALSE 时，对该指令执行无影响。

2、当指令正在执行中，BufferMode=0 下，Execute 再次由 FALSE 变为 TRUE 时，该指令可以自己立即打断自己。

- **输出参数**

名称	功能	数据类型	输出范围
InSync（达到同步状态）	该参数输出为 TRUE 时表示从轴达到同步状态	BOOL	TRUE 或 FALSE
EndOfProfile（凸轮终点执行标志）	该参数输出为 TRUE 时表示凸轮终点被执行	BOOL	TRUE 或 FALSE
Busy（执行中）	该参数输出为 TRUE 时表示指令正在执行中	BOOL	TRUE 或 FALSE

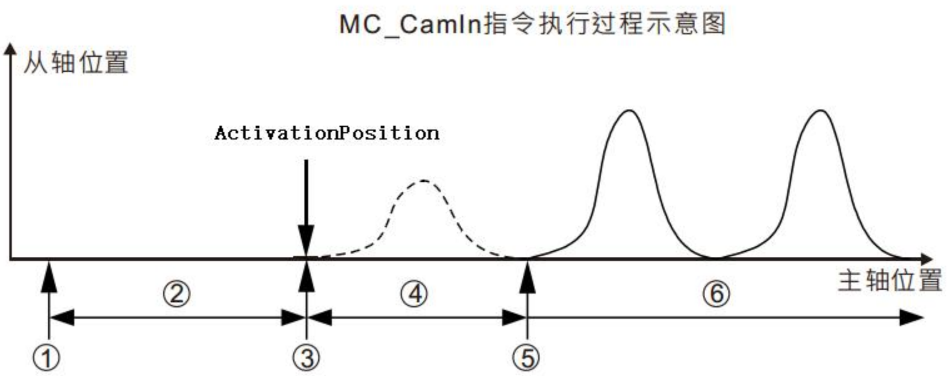
Active	该参数输出为 TRUE 时表示指令正在控制轴	BOOL	TRUE 或 FALSE
CommandAborted	该输出参数为 TRUE 时表示指令的执行被中断	BOOL	TRUE 或 FALSE
Error	该参数为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE
ErrorID	指令执行错误是的错误代码	WORD	

- **功能说明：**
- MC_CamIn 指令用于控制从轴按照预先规划的凸轮关系与主轴实现凸轮同步运动；
- MC_CamOut 指令则用于解除凸轮关系。

MC_CamIn 指令简述

✧ MC_CamIn 指令的执行流程

MC_CamIn 指令的执行流程如下图所示：



阶段①：触发 MC_CamIn 指令执行
阶段②：等待啮合开始
阶段③：主轴到达啮合开始位置，从轴开始执行啮合动作
阶段④：啮合过程中
阶段⑤：啮合完成，主从轴实现同步
阶段⑥：主从轴同步运动

<u>阶段①：</u>触发 MC_CamIn 指令执行

MC_CamIn 指令在此刻执行，从轴将立刻进入等待啮合开始状态。

注意：若 ActivationPosition 为 0 且 ActivationMode 为 0（相对轴位置）时，从轴将以当前速度朝同步速度运动，除此之外，从轴将立即停止运动！

MC_CamIn 指令的所用输入参数将在此刻读入指令并锁定，供指令在执行过程中使用。

<u>阶段②：</u>等待啮合开始

从轴在静止状态下，等待开始执行啮合动作时机的到来，即等待主轴经过参数 ActivationPosition 所指定的位置。从轴等待的时间在不同情形下会有不同，如果 MC_CamIn 指令开始执行时，主轴即处在参数 ActivationPosition 所指定的位置，则从轴立即开始执行啮合动作；如果主轴永远没有机会到达参数

ActivationPosition 所指定的位置，则从轴将永远无法开始执行啮合，永远无法实现凸轮同步。参数 ActivationPosition、ActivationMode 作用于此阶段。

<u>阶段③：</u>主轴到达啮合开始位置，从轴开始执行啮合动作

当主轴经过参数 ActivationPosition 所指定的位置时，从轴开始执行啮合动作。参数 MasterAbsolute、SlaveAbsolute、MasterOffset、SlaveOffset、MasterScaling、SlaveScaling 将在此刻开始作用，用于确定主从轴的轴位置与其凸轮相位之间的对应关系。

<u>阶段④：</u>啮合过程中

从轴按参数 StartMode 所指定的方式执行啮合动作。除参数 StartMode 外，参数 Velocity、Acceleration、Deceleration 也作用于此阶段，它们将决定啮合过程中，从轴速度、加/减速度这几项运动特性。

<u>阶段⑤：</u>啮合完成，主从轴实现同步

在从轴开始执行啮合动作后，如果主从轴对应的凸轮相位满足规划的凸轮关系，则啮合完成，从轴与主轴实现凸轮同步。

说明：上图仅表示啮合开始时的主轴位置大于 MC_CamIn 指令开始执行时刻的主轴位置的情形，等于以及小于的情形可同理推导。

■ ActivationPosition

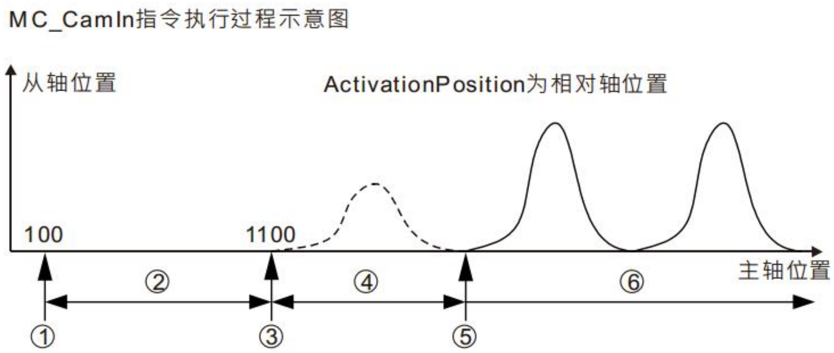
参数 ActivationPosition 为凸轮啮合的起始位置（该位置为主轴的“位置”），即在 MC_CamIn 指令正确触发执行且主轴到达 ActivationPosition 时，从轴开始执行啮合动作。

ActivationPosition 可以为：主轴位置、主轴相位、主轴凸轮相位，通过参数 ActivationMode 进行选择。

● **ActivationPosition 为相对轴位置**

当参数 ActivationMode=0 时，ActivationPosition 为轴位置，且与 MC_CamIn 指令开始执行时刻的主轴位置为相对关系，即实际啮合开始时的主轴位置为 MC_CamIn 指令开始执行时刻的主轴位置加上 ActivationPosition。

例如：MC_CamIn 指令开始执行时刻的主轴位置为 100，ActivationPosition 为 1000，则实际啮合开始时的主轴位置为 1100（1100=100+1000）。



阶段①：触发 MC_CamIn 指令执行，此时主轴绝对位置为 100

阶段②：等待啮合开始

阶段③：主轴到达啮合开始位置（1100），从轴开始执行啮合动作

阶段④：啮合过程中

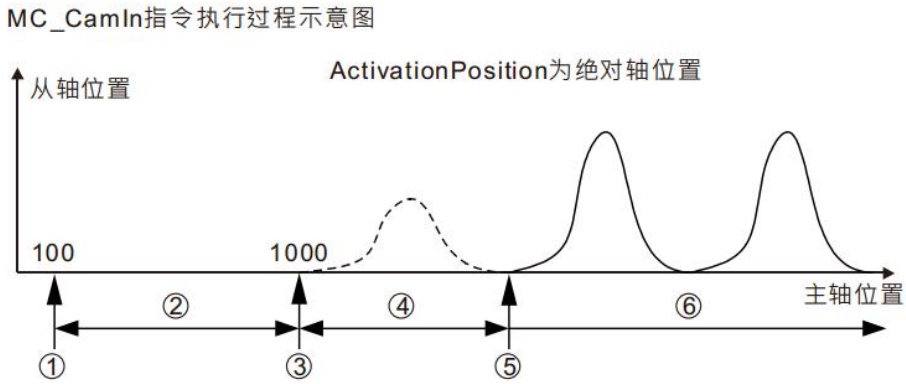
阶段⑤：啮合完成，主从轴实现同步

阶段⑥：主从轴同步运动

- **ActivationPosition 为绝对轴位置**

  当参数 ActivationMode =1 时，ActivationPosition 为轴位置，且与 MC_CamIn 指令开始执行时刻的主轴位置为绝对关系，即实际啮合开始时的主轴位为 ActivationPosition。

  例如：MC_CamIn 指令开始执行时刻的主轴位置为 100，ActivationPosition 为 1000， 则实际啮合开始时的主轴位置为 1000（1000=ActivationPosition）。



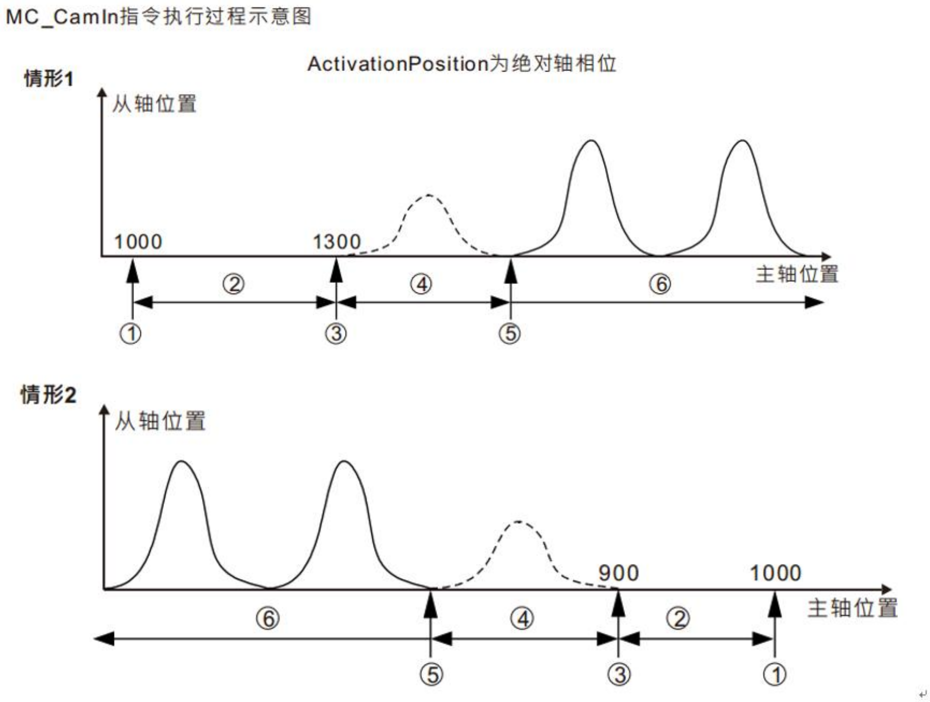
阶段①：触发 MC_CamIn 指令执行，此时主轴绝对位置为 100
阶段②：等待啮合开始
阶段③：主轴到达啮合开始位置（1000），从轴开始执行啮合动作
阶段④：啮合过程中
阶段⑤：啮合完成，主从轴实现同步
阶段⑥：主从轴同步运动

- **ActivationPosition 为绝对轴相位**

  当参数 ActivationMode =2 时，ActivationPosition 为绝对轴相位（绝对轴相位为轴绝对位置按模做取余运算后的结果）。当主轴绝对轴相位为 ActivationPosition 时，从轴开始执行啮合动作。

  绝对轴相位具有周期循环特性，在主轴运转过程中，其绝对轴相位可能多次出现等于 ActivationPosition 的情形，但仅在 MC_CamIn 指令开始执行后，主轴绝对轴相位第一次等于 ActivationPosition 时，从轴开始执行啮合动作。

  例如：主轴的模为 400， ActivationPosition=100， MC_CamIn 指令开始执行时刻的主轴位置为 1000，由于 MC_CamIn 指令开始执行时刻的主轴绝对轴相位为 200（200=1000%400），从轴不会执行啮合动作。此后，当主轴位置为 1300（绝对轴相位为 100=1300%400）或 900（绝对轴相位为 100=900%400）时，从轴开始执行啮合动作（%表示取余运算）



阶段①：触发 MC_CamIn 指令执行，此时主轴绝对位置为 1000（绝对轴相位为 200）

阶段②：等待啮合开始

阶段③：主轴到达啮合开始位置（情形 1 为 1300，情形 2 为 900），从轴开始执行啮合动作

阶段④：啮合过程中

阶段⑤：啮合完成，主从轴实现同步

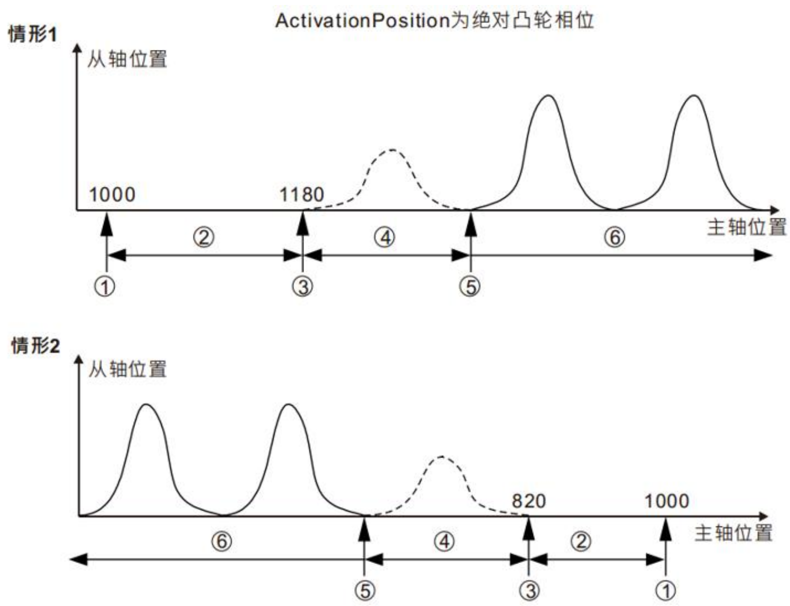
阶段⑥：主从轴同步运动

注意：当 ActivationPosition 为绝对轴相位时，参数 ActivationPosition 的有效范围为：0~模（不包括模）。如果参数 ActivationPosition 的值不在有效范围内，MC_CamIn 指令执行时，将报错且执行失败！

- **ActivationPosition 为绝对凸轮相位**

当参数 ActivationMode=3 时，ActivationPosition 为绝对凸轮相位（绝对凸轮相位为轴绝对位置按其凸轮周期做取余运算后的结果）。当主轴凸轮相位为 ActivationPosition 时，从轴开始执行啮合动作。

凸轮相位具有周期循环特性，在主轴运转过程中，其凸轮相位可能多次出现等于 ActivationPosition 的情形，但仅在 MC_CamIn 指令开始执行后，主轴凸轮相位第一次等于 ActivationPosition 时，从轴开始执行啮合动作。例如：凸轮表中主轴的最大范围值为 360，ActivationPosition=100，MC_CamIn 指令开始执行时刻的主轴位置为 1000，由于 MC_CamIn 指令开始执行时刻的主轴绝对凸轮相位为 280（280=1000%360），从轴不会执行啮合动作。此后，当主轴位置为 1180（绝对凸轮相位为 100=1180%360）或 820（绝对凸轮相位为 100=820%360）时，从轴开始执行啮合动作。



阶段①：触发 MC_CamIn 指令执行，此时主轴绝对位置为 1000（绝对凸轮相位为 280）

阶段②：等待啮合开始

阶段③：主轴到达啮合开始位置（情形 1 主轴位置为 1180，情形 2 主轴位置为 820），从轴开始执行啮合动作

阶段④：啮合过程中

阶段⑤：啮合完成，主从轴实现同步

阶段⑥：主从轴同步运动

注意： 当 ActivationPosition 为绝对凸轮相位时，参数 ActivationPosition 的有效范围为：0~凸轮周期（不包括周期值）。如果参数 ActivationPosition 的值不在有效范围内，MC_CamIn 指令执行时，将报错且执行失败！

■ **主从轴位置关系**

通过软件预先规划的凸轮关系是主从轴之间的位置关系，此处的“位置”为主从轴的凸轮相位，而非实际轴位置。若将预先规划的凸轮关系视为函数 CAM，则函数 CAM 的输入为主轴凸轮相位，输出为从轴凸轮相位，如下所示：

y=CAM(x)

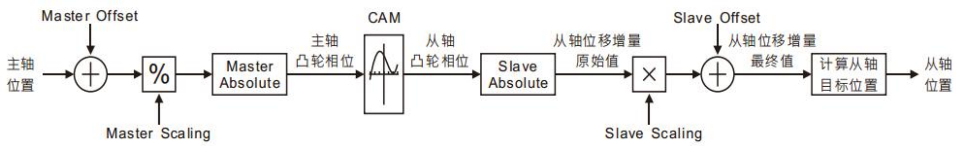
x：主轴凸轮相位

y：从轴凸轮相位

凸轮相位来源于轴位置，它们之间存在换算关系。轴位置与凸轮相位之间的换算关系与参数 MasterAbsolute、SlaveAbsolute、MasterOffset、SlaveOffset、MasterScaling、SlaveScaling 有关，请参阅相关详细内容。

从轴在 MC_CamIn 指令的作用下跟随主轴做凸轮同步运动。凸轮同步运动中，主从轴的轴位置对应关系建立在预先规划的凸轮关系（凸轮关系曲线或凸轮表）基础之上，由主轴轴位置计算从轴轴位置的

过程如下图所示：



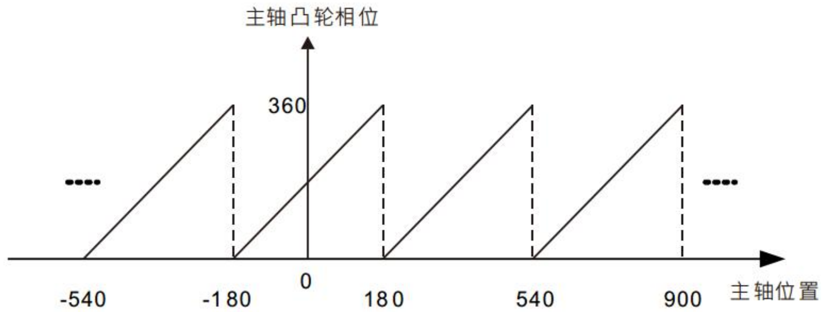
■ **MasterAbsolute 与 SlaveAbsolute**

参数 MasterAbsolute 用于指定主轴的轴位置与其凸轮相位之间的对应关系：当参数值为 TRUE 时，为绝对关系；当参数值为 FALSE 时，为相对关系。参数 SlaveAbsolute 与 MasterAbsolute 同理。

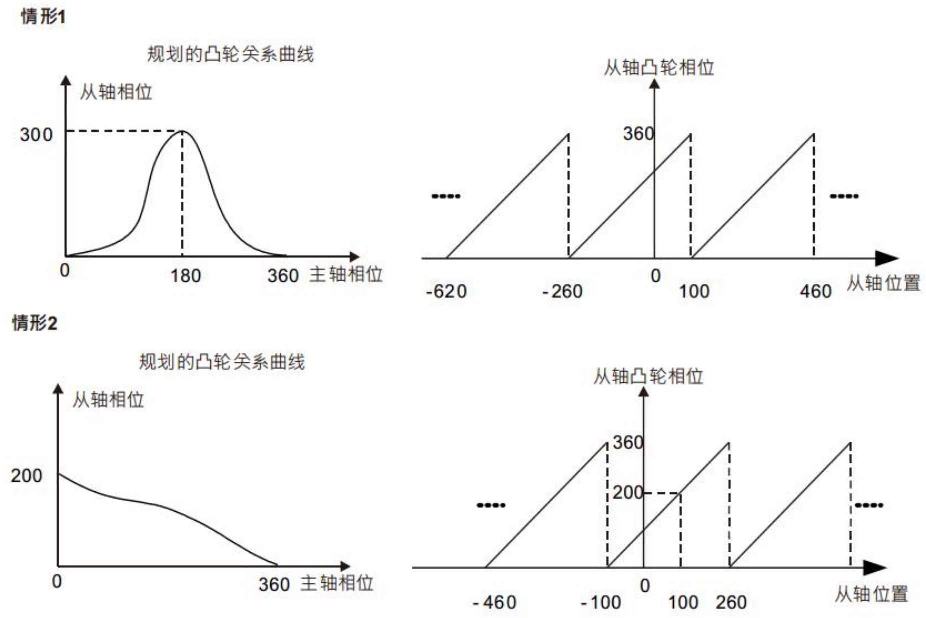
参数 MasterAbsolute 和 SlaveAbsolute 作用于啮合开始时刻，也就是轴位置与凸轮相位之间的对应关系在啮合开始时刻建立（注意：对应关系是在啮合动作开始时刻建立，而不是 MC_CamIn 指令开始执行时刻）。在此之后，凸轮相位的计算，将依照该对应关系。

● **相对模式**

当参数 MasterAbsolute 为 FALSE 时，主轴的轴位置与其凸轮相位之间为相对关系，即啮合开始时刻的主轴的轴位置对应其凸轮相位为 0，此后，主轴凸轮相位的计算，将依照该对应关系。例如：主轴为相对模式，凸轮关系中主轴最大范围值为 360，啮合开始时刻主轴的轴位置为 180，则主轴的轴位置 180 对应其凸轮相位为 0，轴位置 200 对应其凸轮相位为 20（20=（200-180）%360），并可以此类推。这种情形下，主轴的轴位置（主轴位置）与与其凸轮相位之间的关系如下图所示：

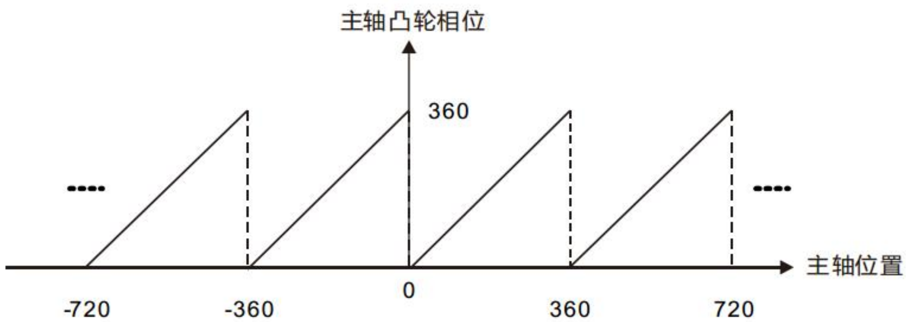


当参数 SlaveAbsolute 为 FALSE 时，从轴的轴位置与其凸轮相位之间为相对关系，即啮合开始时刻从轴的凸轮相位与此刻主轴的凸轮相位满足规划的凸轮关系。从轴为相对模式时，确定从轴凸轮相位的方法与主轴不同，在确定从轴凸轮相位时需满足条件：啮合开始时刻从轴的凸轮相位与此刻主轴的凸轮相位满足规划的凸轮关系。例如：从轴为相对模式，凸轮关系中从轴最大范围值为 360，啮合开始时刻从轴的轴位置为 100，若此刻主轴凸轮相位为 0（按凸轮关系要求从轴凸轮相位为 0），则从轴轴位置 100 对应其凸轮相位为 0，如下图情形 1 所示；若按凸轮关系要求从轴凸轮相位为 200，则从轴轴位置 100 对应其凸轮相位为 200，如下图情形 2 所示。



- **绝对模式**

当参数 MasterAbsolute 为 TRUE 时，主轴的轴位置与其凸轮相位之间为绝对关系，任意时刻，主轴的凸轮相位等于该时刻主轴的轴位置与凸轮关系中的主轴最大范围值做取余运算的结果。例如：主轴为绝对模式，凸轮关系中的主轴最大范围值为 360，则主轴的轴位置为 100 时，其凸轮相位为 100（100=100%360）；主轴的轴位置为 500 时，其凸轮相位为 140（140=500%360），并可以此类推，主轴轴位置与与其凸轮相位之间的关系如下图所示



当参数 SlaveAbsolute 为 TRUE 时，从轴的轴位置与其凸轮相位之间为绝对关系，任意时刻，从轴的凸轮相位等于该时刻从轴的轴位置与凸轮关系中的从轴最大范围值做取余运算的结果。从轴为绝对模式时，其轴位置与其凸轮相位之间的对应关系与主轴一致。

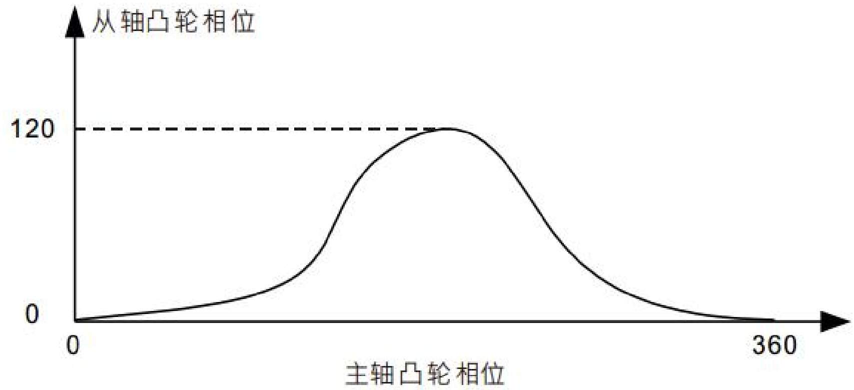
- **位置偏移和位置比例缩放（Offset 和 Scaling）**

主从轴的凸轮关系为预先规划，但在执行凸轮时，可通过参数”Offset”和“Scaling”在预先规划的凸轮关系基础上进行位置偏移或位置缩放，例如：加工的同一产品有几种不同尺寸，则只需规划一种凸轮关系，然后通过改变参数“Offset”和“Scaling”以适应不同尺寸产品之间的加工切换。

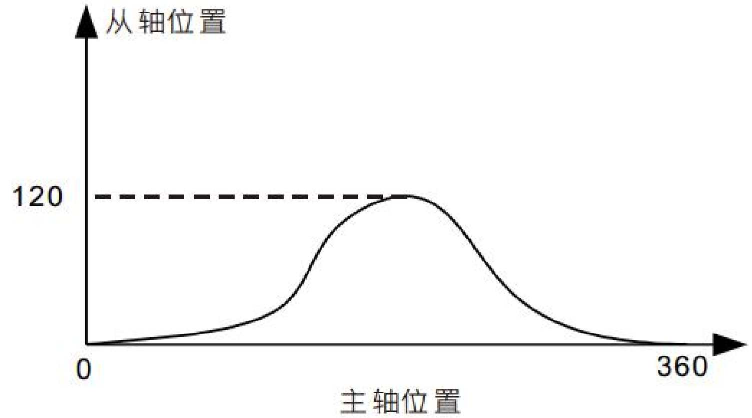
参数 MasterOffset 在主轴为相对或绝对模式均有效；参数 SlaveOffset 仅在从轴为绝对模式（SlaveAbsolute=TRUE）时有效，从轴为相对模式（SlaveAbsolute=FALSE）时无

效。

主从轴的位置偏移和比例缩放共同决定实际执行的凸轮关系，其作用效果通过以下范例进行描述。预先规划的凸轮关系如下图所示：

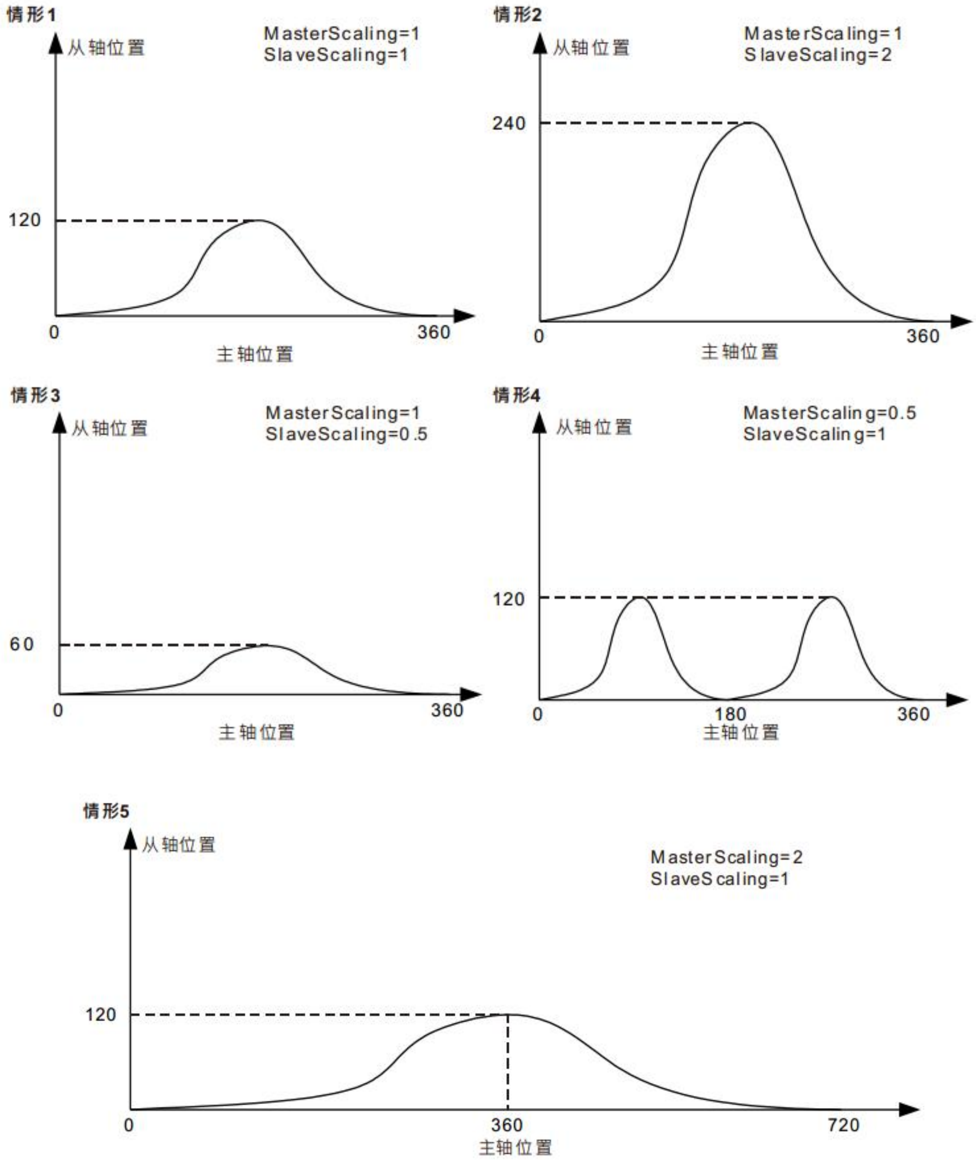


若主从轴均为绝对模式，且执行啮合动作时，主从轴的轴位置均为 0，在不使用偏移和缩放时（默认值），凸轮执行过程中的主从轴实际位置对应关系如下图所示：



当位置偏移量或缩放比例不为默认值时，其对凸轮执行过程中主从轴实际位置对应关系的影响如下：

- **主从轴偏移量为 0，主从轴缩放比例对实际执行的凸轮关系的影响**



情形 1：当主从轴缩放比例为 1，偏移量为 0 时，实际凸轮关系与预先规划的一致。

情形 2：当主轴缩放比例为 1，从轴缩放比例为 2，主从偏移量为 0 时，与主轴位置对应的从轴位置变为预先规划的两倍。

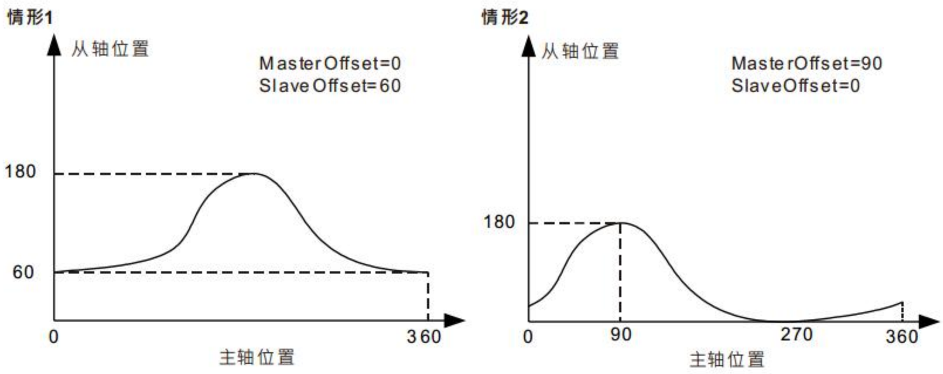
情形 3：当主轴缩放比例为 1，从轴缩放比例为 0.5，主从轴偏移量为 0 时，与主 轴位置对应的从轴位置变为预先规划的 1/2。

情形 4：当主轴缩放比例为 0.5，从轴缩放比例为 1，主从轴凸轮偏移量为 0 时，与从轴位置对应的主轴位置变为预先规划的 1/2。如果从凸轮相位的角度看，则是主轴的凸轮相位为预先规划的 1/2，即主轴凸轮周期从 360 变为 180（180=360*0.5），从轴凸轮相位不变。

情形 5：当主轴缩放比例为 2，从轴缩放比例为 1，主从轴偏移量为 0 时，与从轴位置对应的主轴位置变为预先规划的两倍。如果从凸轮相位的角度看，则是主轴凸轮相位为原来的两倍，即主轴凸轮周期从 360 变为 720（720=360*2），从轴凸轮相位不变。

- **主从轴缩放比例为 1，主从轴偏移实际执行的凸轮关系的影响**

主轴偏移相当于将执行凸轮时的实际轴位置关系曲线作横向移动；从轴偏移相当于将执行凸轮时的轴位置关系曲线作纵向移动。

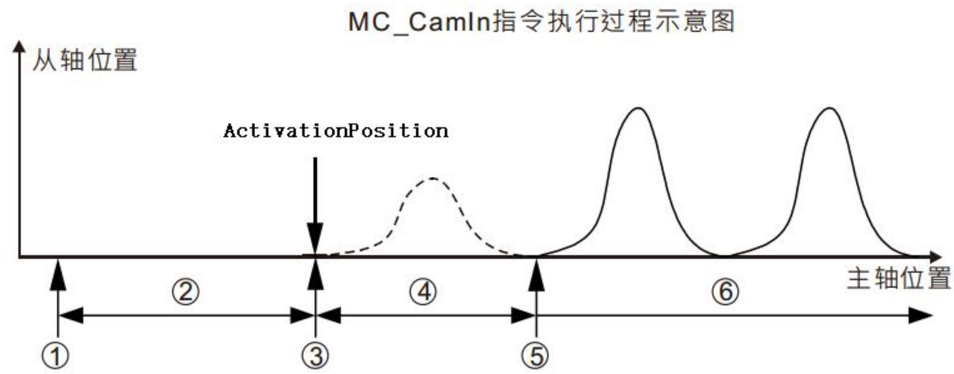


情形 1：当主从轴缩放比例为 1，主轴偏移量为 0，从轴偏移量为 60 时，与主轴位置对应的从轴位置均在预先规划的基础上加上 60。例如：规划的凸轮关系中，主轴轴位置 180 对应的从轴轴位置为 180，实际执行时，对应的从轴轴位置为 240（240=180+60）。

情形 2：当主从轴缩放比例为 1，主轴偏移量为 90，从轴偏移量为 0 时，与从轴轴位置对应的主轴轴位置在预先规划的基础上偏移 90（加上偏移量）。例如：规划的凸轮关系中，主轴轴位置 180 对应的从轴轴位置为 180，实际执行时，主轴轴位置 90 对应的从轴轴位置为 180，即预先规划的凸轮关系中主轴轴位置 180（180=90+90）所对应的从轴轴位置。

- **StartMode**

啮合过程中，从轴的动作方式可通过参数 StartMode 进行指定，即 StartMode 作用于 MC_CamIn 指令的阶段④，如下图所示：



阶段①：触发 MC_CamIn 指令执行

阶段②：等待啮合开始

阶段③：主轴到达啮合开始位置，从轴开始执行啮合动作

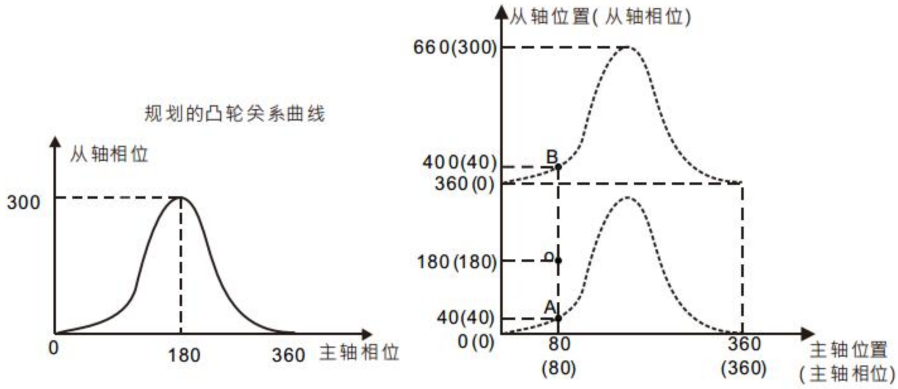
阶段④：啮合过程中

阶段⑤：啮合完成，主从轴实现同步

阶段⑥：主从轴同步运动

凸轮同步要求主从轴的凸轮相位满足定义的凸轮关系，啮合过程则为从轴朝同步相位运动的过程，该同步相位与主轴凸轮相位满足定义的凸轮关系。由于轴的凸轮相位具有周期循环特性，即每个凸轮相位均有多个轴位置与之对应，啮合时，期望的同步位置存在多种选择，

也就使得啮合方式有多种选择。例如：开始执行啮合时，主从轴的凸轮相位分别为 80 和 180（如右下图中的 O 点），但定义的凸轮关系要求从轴的凸轮相位为 40，则此刻从轴期望的同步位置为 40 或 400（如右下图中的点 A 和点 B），可通过参数 StartMode 选择啮合过程是从 O 到 A 还是 O 到 B。



StartMode 有三种模式供选择，分别为：最短距离渐变（StartMode=0）、正向渐变（StartMode=1）和反向渐变（StartMode= -1），用户可根据实际需求选择不同的啮合模式。

- **StartMode=0（最短距离渐变）**

若参数 StartMode=0，执行啮合动作时，从轴朝与同步位置距离最短的方向运动，此时从轴的运动受参数 Velocity、Acceleration、Deceleration 影响。

- **StartMode=1（正向渐变）**

若参数 StartMode=1， 执行啮合动作时，从轴正向朝同步位置运动， 此时从轴的运动受参数 Velocity、Acceleration、Deceleration 影响。

- **StartMode= -1（反向渐变）**

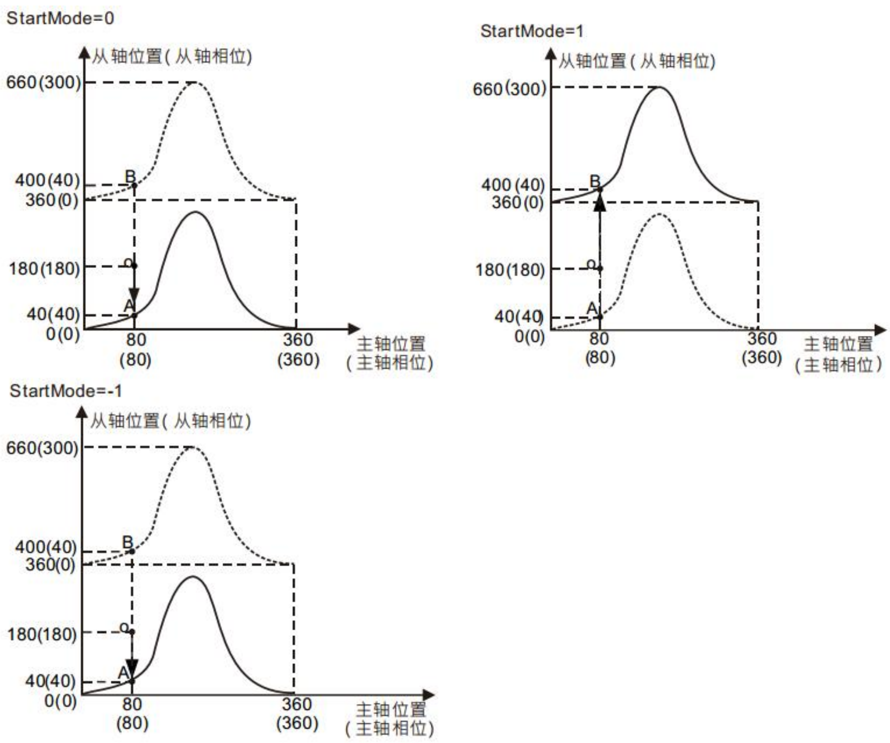
若参数 StartMode= -1，执行啮合动作时，从轴反向朝同步位置运动，此时从轴的运动受参数 Velocity、Acceleration、Deceleration 影响。

例如：开始执行啮合动作时，主从轴的凸轮相位分别为 80 和 180（如下图的 O 点），根据定义的凸轮关系，要求主轴凸轮相位为 80 时，从轴凸轮相位为 40（如下图中的 A 点或 B 点），则 StartMode 选择不同模式时，啮合过程中，从轴的动作方式如下图所示。

StartMode=0：从轴立即从 O 点渐变到 A 点，在 A 点实现同步，因为从 O 点到 A 点的距离小于 O 点到 B 的距离；

StartMode=1：从轴正向立即从 O 点渐变到 B 点；

StartMode= -1：从轴反向立即从 O 点渐变到 A 点；

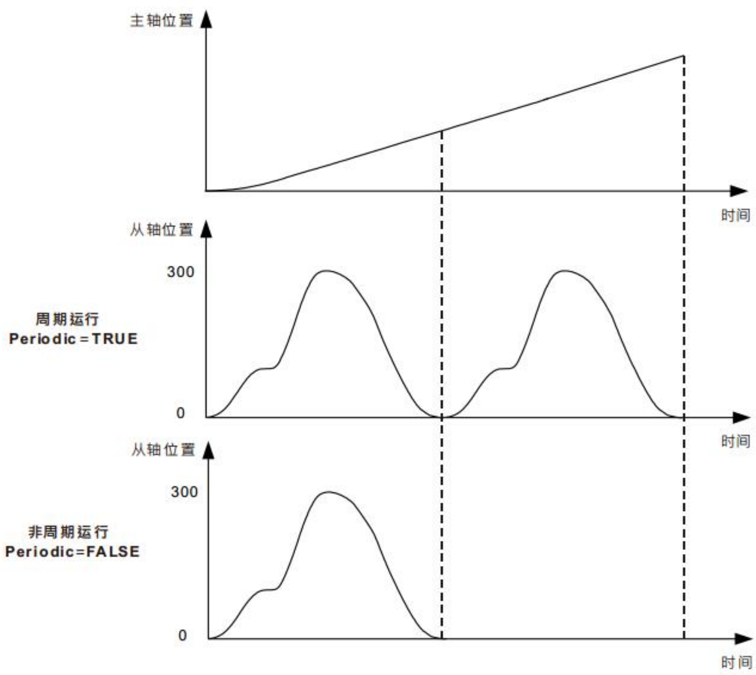


■ **周期/非周期执行凸轮（Periodic）**

在电子凸轮实际应用中，有的可能需要按周期循环运行，而有的则可能仅需要运行一个周期，参数 Periodic 即用于对这两种情形进行选择。

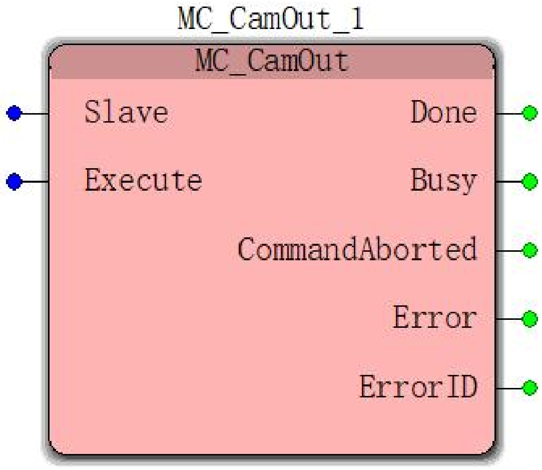
当参数 Periodic=TRUE 时，从轴跟随主轴按周期循环执行凸轮，直到凸轮关系解除；

当参数 Periodic=FALSE 时，从轴与主轴凸轮同步后，执行到凸轮周期结束点时，从轴与主轴的凸轮关系解除，且从轴立即停止运动。



11.5.13 MC_CamOut（电子凸轮脱离指令）

FB/FC	说明	适用机种
FB	此指令用于解除两个轴之间已建立的电子凸轮关系	VEC-VA-MP-005-MA



➢ **输入参数**

名称	功能	数据类型	设定范围（缺省值）	生效时机
Slvae（从轴）	设定电子凸轮的从轴	USINT	**模拟量/脉冲：** 0~4（实轴） 5~11（虚轴） **CANopen 模式：** 0~15 (实轴/虚轴) (0)	Exexcute 由 FALSE 变为 TRUE
Execute（执行位）	当 Execute 由 FALSE 变为 TRUE，指令开始执行	BOOL	TRUE 或 FALSE (FALSE)	

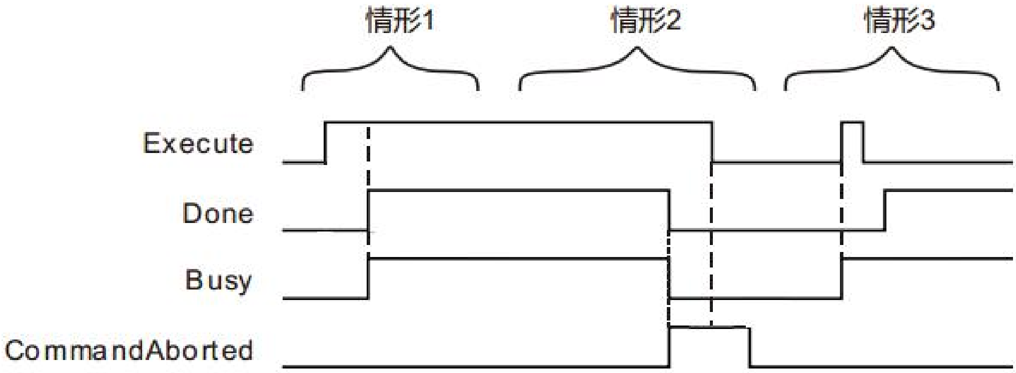
➢ **输出参数**

名称	功能	数据类型	输出范围
Done	该参数输出为 TRUE 时表示指令执行完成	BOOL	TRUE 或 FALSE
Busy（执行中）	该参数输出为 TRUE 时表示指令正在执行中	BOOL	TRUE 或 FALSE
CommandAborted（中断）	该参数输出为 TRUE 时表示指令被中断	BOOL	TRUE 或 FALSE
Error（错误）	该指令输出为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE
ErrorID（错误代码）	执行指令出错时的错误代码	WORD	-

功能说明

MC_CamOut 指令用于解除已建立的电子凸轮关系。指令作用于凸轮从轴，且从轴将保持脱离时的速度继续运行。若需控制凸轮从轴停止运动，可对从轴使用 MC_Halt 或者 MC_Stop 指令。MC_Halt 或者 MC_Stop 指令执行完成后，从轴停止运动且凸轮关系解除。

- **输出参数变化时序图**



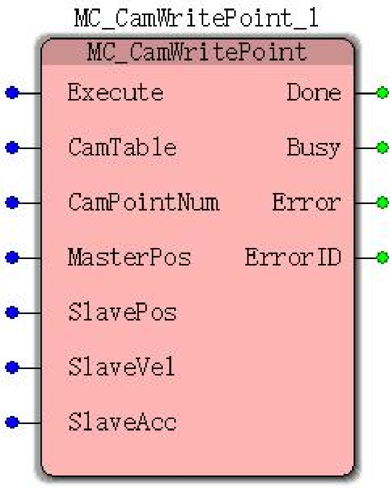
情形 1：当 Execute 由 FALSE 变为 TRUE 时，一个周期后，Busy、Done 变为 TRUE。Execute 由 TRUE 变为 FALSE 后，Busy 和 Done 依然保持为 TRUE。

情形 2：当 Execute 为 TRUE 时，若指令被其它指令中断，CommandAborted 变为 TRUE，同时 Busy 和 Done 变为 FALSE；当 Execute 由 TRUE 变为 FALSE 时，一个周期后 CommandAborted 变为 FALSE。

情形 3：在指令执行过程中，不到一个周期时，Execute 由 TRUE 变为 FALSE 后，当到达一个周期时，Done 变为 TRUE，且 Busy 仍然保持为 TRUE。

11.5.14 MC_CamWritePoint（写入凸轮点信息指令）

FB/FC	说明	适用机种
FB	此指令用于写入凸轮点信息	VEC-VA-MP-005-MA



➢ **输入参数**

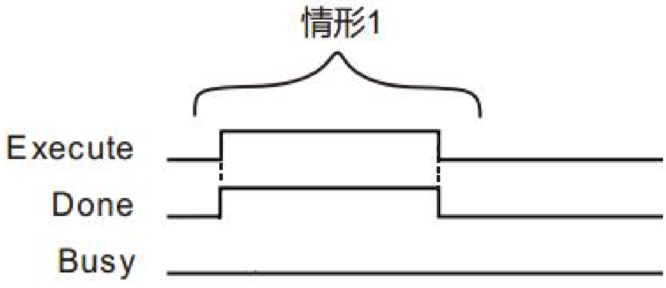
名称	功能	数据类型	设定范围（缺省值）	生效时机
Execute（执行位）	当 Execute 由 FALSE 变为 TRUE，指令开始执行	BOOL	TRUE 或 FALSE (FALSE)	
CamTable（电子凸轮表号）	用于设定主从轴建立凸轮关系所依据的凸轮表	USINT	0~31	Exexcute 由 FALSE 变为 TRUE
CamPointNum（凸轮点编号）	选择读取的凸轮点	UINT		Exexcute 由 FALSE 变为 TRUE
MasterPos	读取指令设置电子凸轮点的主轴位置	LREAL	正数、0	MasterPos
SlavePos	读取指令设置电子凸轮点的从轴位置	LREAL	正数、负数、0	SlavePos
SlaveVel	读取指令设置电子凸轮点的从轴速度	LREAL	正数、负数、0	SlaveVel
SlaveAcc	读取指令设置电子凸轮点的从轴加速度	LREAL	正数、负数、0	SlaveAcc

➢ **输出参数**

名称	功能	数据类型	输出范围
Done	该参数输出为 TRUE 时表示指令执行完成	BOOL	TRUE 或 FALSE
Busy（执行中）	该参数输出为 TRUE 时表示指	BOOL	TRUE 或 FALSE

	令正在执行中		
Error（错误）	该指令输出为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE
ErrorID（错误代码）	执行指令出错时的错误代码	WORD	-

➢ **输出参数变化时序图**



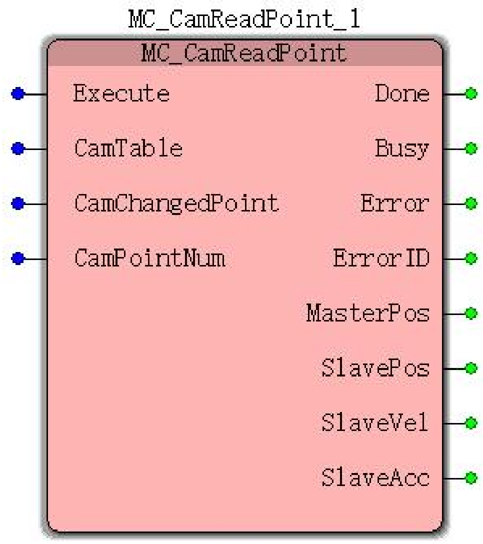
➢ 情形 1：当 Execute 由 FALSE 变为 TRUE 时，同时 Done 变为 TRUE，当 Execute 变为 FALSE 时，Done 变为 FALSE

功能说明

此指令用于写电子凸轮表中凸轮点的信息。写电子凸轮点成功后，更改后的曲线并不能立即生效，需要执行 MC_CamSet 指令更改后凸轮曲线才生效。

11.5.15 MC_CamReadPoint（读取凸轮点信息指令）

FB/FC	说明	适用机种
FB	此指令用于读取凸轮点信息	VEC-VA-MP-005-MA



➢ **输入参数**

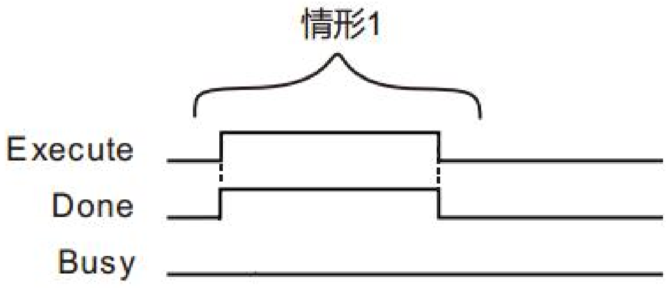
名称	功能	数据类型	设定范围（缺省值）	生效时机
Execute（执行位）	当 Execute 由 FALSE 变为 TRUE，指令开始执行	BOOL	TRUE 或 FALSE (FALSE)	
CamTable（电子凸轮表号）	用于设定主从轴建立凸轮关系所依据的凸轮表	USINT	0~31	Exexcute 由 FALSE 变为 TRUE
CamChangePoint（凸轮点更改前或后信息选择）	当为 FALSE 时，指令读取该凸轮点更改前信息；当为 TRUE 时，指令读取该凸轮点更改后信息。	BOOL	TRUE/FALSE	Exexcute 由 FALSE 变为 TRUE
CamPointNum（凸轮点编号）	选择读取的凸轮点	UINT		Exexcute 由 FALSE 变为 TRUE

➢ **输出参数**

名称	功能	数据类型	输出范围
Done	该参数输出为 TRUE 时表示指令执行完成	BOOL	TRUE 或 FALSE
Busy（执行中）	该参数输出为 TRUE 时表示指令正在执行中	BOOL	TRUE 或 FALSE
Error（错误）	该指令输出为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE

ErrorID（错误代码）	执行指令出错时的错误代码	WORD	-
MasterPos	读取指令设置电子凸轮点的主轴位置	LREAL	正数、0
SlavePos	读取指令设置电子凸轮点的从轴位置	LREAL	正数、负数、0
SlaveVel	读取指令设置电子凸轮点的从轴速度	LREAL	正数、负数、0
SlaveAcc	读取指令设置电子凸轮点的从轴加速度	LREAL	正数、负数、0

➢ **输出参数变化时序图**



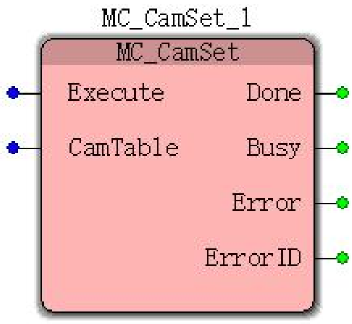
➢ 情形 1：当 Execute 由 FALSE 变为 TRUE 时，同时 Done 变为 TRUE，当 Execute 变为 FALSE 时，Done 变为 FALSE

■ **功能说明**

此指令用于读取电子凸轮表中凸轮点的信息。当 CamChangedPoint 为 FALSE 时，读取到的是 MC_CamSet 指令更改前的凸轮点信息，当 CamChangedPoint 为 TRUE 时，读取到的是 MC_CamSet 指令更改后的凸轮点信息。

11.5.16 MC_CamSet（更改凸轮点生效指令）

FB/FC	说明	适用机种
FB	此指令用于将更改的凸轮点生效	VEC-VA-MP-005-MA



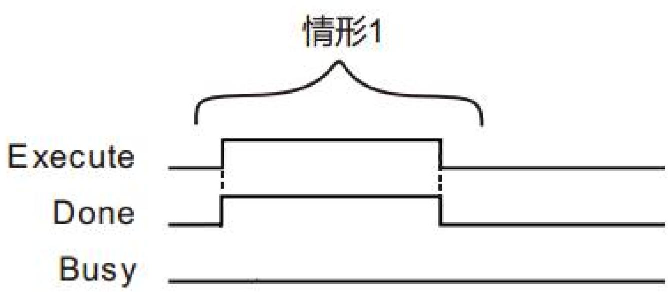
➢ **输入参数**

名称	功能	数据类型	设定范围（缺省值）	生效时机
Execute（执行位）	当 Execute 由 FALSE 变为 TRUE，指令开始执行	BOOL	TRUE 或 FALSE (FALSE)	
CamTable（电子凸轮表号）	用于设定主从轴建立凸轮关系所依据的凸轮表	USINT	0~31	Exexcute 由 FALSE 变为 TRUE

➢ **输出参数**

名称	功能	数据类型	输出范围
Done（完成位）	该参数输出为 TRUE 时表示指令执行完成	BOOL	TRUE 或 FALSE
Busy（执行中）	该参数输出为 TRUE 时表示指令正在执行中	BOOL	TRUE 或 FALSE
Error（错误）	该指令输出为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE
ErrorID（错误代码）	执行指令出错时的错误代码	WORD	-

➢ **输出参数变化时序图**



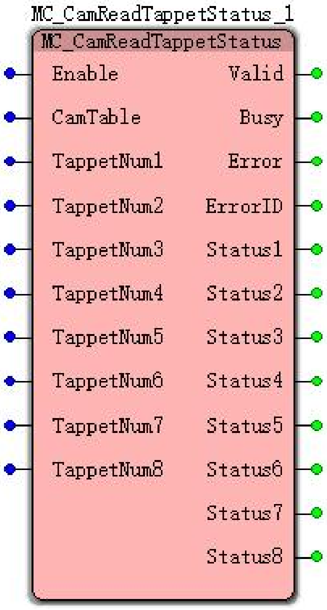
情形 1：当 Execute 由 FALSE 变为 TRUE 时，同时 Done 变为 TRUE，当 Execute 变为 FALSE 时，Done 变为 FALSE

■ **功能说明：**

此指令用于将更改的凸轮点生效。首先使用 MC_CamWritePoint 指令将电子凸轮表中对应的凸轮点信息写入，然后执行 MC_CamSet 指令，更改后凸轮点信息才生效。
执行 MC_CamSet 指令，更改后的凸轮曲线立即生效。

11.5.17 MC_ReadTappetStatus（读取多个挺杆点状态指令）

FB/FC	说明	适用机种
FB	此指令用于读取多个挺杆点状态	VEC-VA-MP-005-MA



➢ **输入参数**

名称	功能	数据类型	设定范围（缺省值）	生效时机
Execute（执行位）	当 Execute 由 FALSE 变为 TRUE，指令开始执行	BOOL	TRUE 或 FALSE (FALSE)	
CamTable（电子凸轮表号）	用于设定主从轴建立凸轮关系所依据的凸轮表	USINT	0~31	Exexcute 由 FALSE 变为 TRUE
TappetNum1（挺杆点编号）	设定挺杆点编号	UINT	(不可缺省)	Exexcute 由 FALSE 变为 TRUE
TappetNum2（挺杆点编号）	设定挺杆点编号	UINT	(不可缺省)	Exexcute 由 FALSE 变为 TRUE
TappetNum3（挺杆点编号）	设定挺杆点编号	UINT	(不可缺省)	Exexcute 由 FALSE 变为 TRUE
TappetNum4（挺杆点编号）	设定挺杆点编号	UINT	(不可缺省)	Exexcute 由 FALSE 变为 TRUE
TappetNum5（挺杆点编号）	设定挺杆点编号	UINT	(不可缺省)	Exexcute 由 FALSE 变为 TRUE
TappetNum6（挺杆点编号）	设定挺杆点编号	UINT	(不可缺省)	Exexcute 由 FALSE 变为 TRUE

TappetNum7 （挺杆点编号）	设定挺杆点编号	UINT	(不可缺省)	Exexcute 由 FALSE 变为 TRUE
TappetNum8 （挺杆点编号）	设定挺杆点编号	UINT	(不可缺省)	Exexcute 由 FALSE 变为 TRUE

➢ **输出参数**

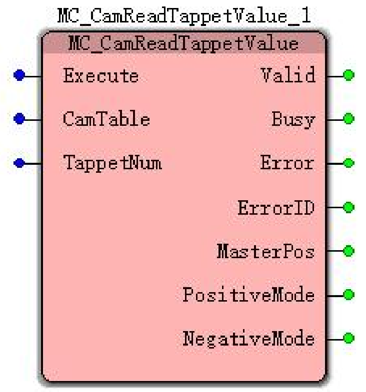
名称	功能	数据类型	输出范围
Valid	该参数输出为 TRUE 时表示指令执行有效	BOOL	TRUE 或 FALSE
Busy（执行中）	该参数输出为 TRUE 时表示指令正在执行中	BOOL	TRUE 或 FALSE
Error（错误）	该指令输出为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE
ErrorID（错误代码）	执行指令出错时的错误代码	WORD	-
Status1 （挺杆点 1 状态）	TappetNum1 指定挺杆点编号的状态	BOOL	TRUE 或 FALSE
Status2 （挺杆点 2 状态）	TappetNum2 指定挺杆点编号的状态	BOOL	TRUE 或 FALSE
Status3 （挺杆点 3 状态）	TappetNum3 指定挺杆点编号的状态	BOOL	TRUE 或 FALSE
Status4 （挺杆点 4 状态）	TappetNum4 指定挺杆点编号的状态	BOOL	TRUE 或 FALSE
Status5 （挺杆点 5 状态）	TappetNum5 指定挺杆点编号的状态	BOOL	TRUE 或 FALSE
Status6 （挺杆点 6 状态）	TappetNum6 指定挺杆点编号的状态	BOOL	TRUE 或 FALSE
Status7 （挺杆点 7 状态）	TappetNum7 指定挺杆点编号的状态	BOOL	TRUE 或 FALSE
Status8 （挺杆点 8 状态）	TappetNum8 指定挺杆点编号的状态	BOOL	TRUE 或 FALSE

■ **功能说明**

此指令用于读取八个挺杆点状态。每个挺杆点状态为主轴正向或者反向经过挺杆点时的状态，每个挺杆点的状态由每个挺杆点的设置决定。每个挺杆点的状态在主轴正向经过凸轮终点或者反向经过凸轮起点时变为 FALSE。

11.5.18 MC_ReadTappetValue（读取单个挺杆点信息指令）

FB/FC	说明	适用机种
FB	此指令用于读取单个挺杆点信息指令	VEC-VA-MP-005-MA



➢ **输入参数**

名称	功能	数据类型	设定范围（缺省值）	生效时机
Execute（执行位）	当 Execute 由 FALSE 变为 TRUE，指令开始执行	BOOL	TRUE 或 FALSE (FALSE)	
CamTable（电子凸轮表号）	用于设定主从轴建立凸轮关系所依据的凸轮表	USINT	0~31	Exexcute 由 FALSE 变为 TRUE
TappetNum（挺杆点编号）	读取挺杆点编号	UINT	(不可缺省)	Exexcute 由 FALSE 变为 TRUE

➢ **输出参数**

名称	功能	数据类型	输出范围
Valid	该参数输出为 TRUE 时表示指令执行有效	BOOL	TRUE 或 FALSE
Busy（执行中）	该参数输出为 TRUE 时表示指令正在执行中	BOOL	TRUE 或 FALSE
Error（错误）	该指令输出为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE
ErrorID（错误代码）	执行指令出错时的错误代码	WORD	-
MasterPos（主轴位置）	显示主轴位置	LREAL	
PositiveMode（正向经过模式）	轴正转经过时挺杆点选择的模式	INT	0:PositiveDisable 1：PositiveOn 2：PositiveOff 3：PositiveInvert
NegativeMode	轴反转经过时挺杆点选择的模	INT	0:NegativeDisable

（反向经过模式）	式		1：NegativeOn 2：NegativeOff 3：NegativeInvert

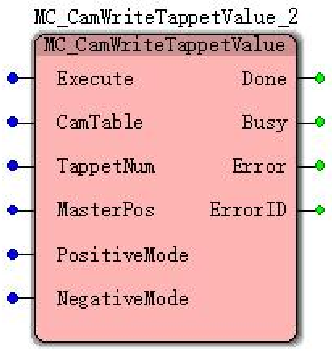
■ **功能说明**

挺杆点信息包括该挺杆点的主轴位置、正向经过模式和反向经过模式。当轴正转时挺杆点可以选择的模式有 PositiveDisable、PositiveOn、PositiveOff 或 PositiveIvert；当轴反转时挺杆点可以选择的模式有 NegativeDisable、NegativeOn、NegativeOff 或 NegativeIvert。各个模式代表的含义如下表所示：

模式	功能	含义
PositiveDisable	关闭	主轴正向经过该点时，读取到该挺杆点时的状态无变化
PositiveOn	置位	主轴正向经过该点时，读取到该挺杆点时的状态为置位状态。
PositiveOff	复位	主轴正向经过该点时，读取到该挺杆点时的状态为复位状态。
PositiveInvert	取反	主轴正向经过该点时， 正向经过该点前的状态为置位，则读取到该挺杆点时的状态为复位状态；正向经过该点前的状态为复位，则读取到该挺杆点时的状态为置位状态。
NegativeDisable	关闭	主轴反向经过该点时，读取到该挺杆点时的状态无变化
NegativeOn	置位	主轴反向经过该点时，读取到该挺杆点时的状态为置位状态。
NegativeOff	复位	主轴反向经过该点时，读取到该挺杆点时的状态为复位状态。
NegativeInvert	取反	主轴反向经过该点时， 反向经过该点前的状态为置位，则读取到该挺杆点时的状态为复位状态；反向经过该点前的状态为复位，则读取到该挺杆点时的状态为置位状态。

11.5.19 MC_WriteTappetValue（修改挺杆点信息指令）

FB/FC	说明	适用机种
FB	此指令用于写入、增加、删除挺杆点信息指令	VEC-VA-MP-005-MA



➢ **输入参数**

名称	功能	数据类型	设定范围（缺省值）	生效时机
Execute（执行位）	当 Execute 由 FALSE 变为 TRUE，指令开始执行	BOOL	TRUE 或 FALSE (FALSE)	
CamTable（电子凸轮表号）	用于设定主从轴建立凸轮关系所依据的凸轮表	USINT	0~31	Exexcute 由 FALSE 变为 TRUE
TappetNum（挺杆点编号）	读取挺杆点编号	UINT	(不可缺省)	Exexcute 由 FALSE 变为 TRUE
MasterPos（主轴位置）	设定挺杆点主轴位置	LREAL		Exexcute 由 FALSE 变为 TRUE
PositiveMode（正向经过模式）	轴正转时设定挺杆点模式 0：关闭 1：置位 2：复位 3：取反	INT	0~3 (0)	Exexcute 由 FALSE 变为 TRUE
NegativeMode（反向经过模式）	轴反转时设定挺杆点模式 0：关闭 1：置位 2：复位 3：取反	INT	0~3 (0)	Exexcute 由 FALSE 变为 TRUE

➢ **输出参数**

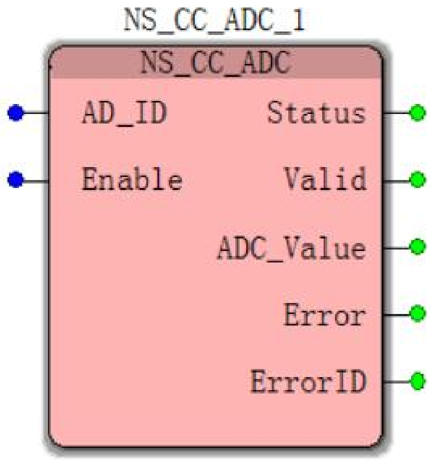
名称	功能	数据类型	输出范围

Done（完成位）	该参数输出为 TRUE 时表示指令执行完成	BOOL	TRUE 或 FALSE
Busy（执行中）	该参数输出为 TRUE 时表示指令正在执行中	BOOL	TRUE 或 FALSE
Error（错误）	该指令输出为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE
ErrorID（错误代码）	执行指令出错时的错误代码	WORD	-

11.6 特殊指令

11.6.1 NS_CC_ADC(AD 指令)

FB/FC	说明	适用机种
FB	此指令用于将模拟量转换成数字量进行输出	VEC-VA-MP-005-MA



➢ **输入参数**

名称	功能	数据类型	设定范围（缺省值）	生效时机
AD_ID（模拟量输入号）	选择需要转的模拟量口 AI0~AI3，其中数值 0~3 对应数模拟量输入口 AI0~AI3	WORD	0~3 (不可缺省)	Enable 为 TRUE
Enable（执行位）	当 Enable 为 TRUE 时，执行该指令	BOOL	TRUE 或 FALSE	

说明：

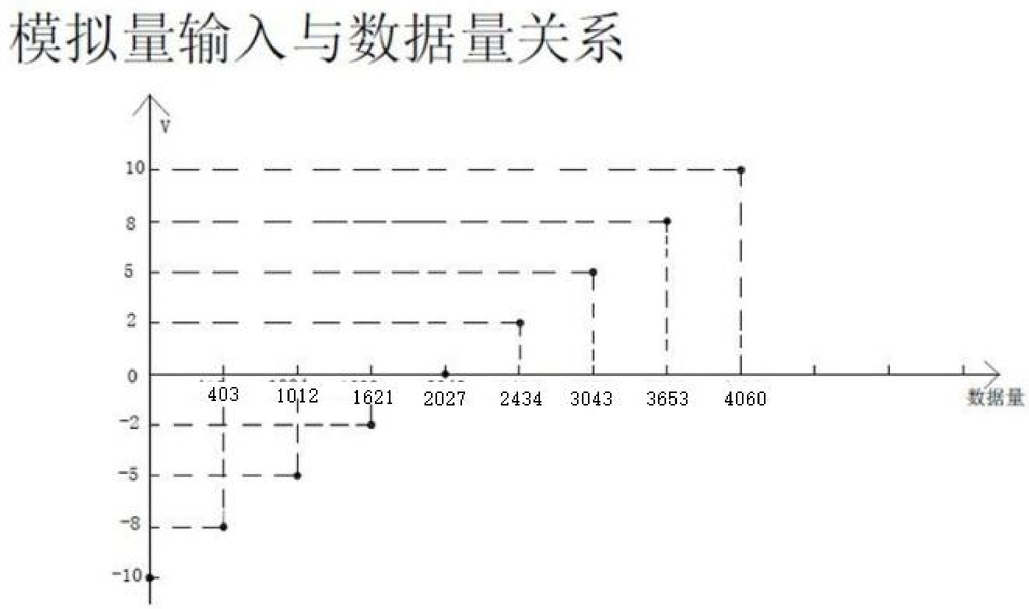
（1）控制器带有四路的模拟量输入（±10V）模拟量和数字量的转换关系如下所示：

名称	数值
模拟量输入电压（V）	-10 ~ 0 ~ + 10
数字量	0 ~ 2048 ~ 4096

（2）实验的结果（由于误差的存在，相应的数字量值上也会有波动仅供参考）

输入电压（V）	数字量
0	2028
2	2434
5	3043
8	3653
10	4060
-10	0
-8	403
-5	1012
-2	1621
0	2027

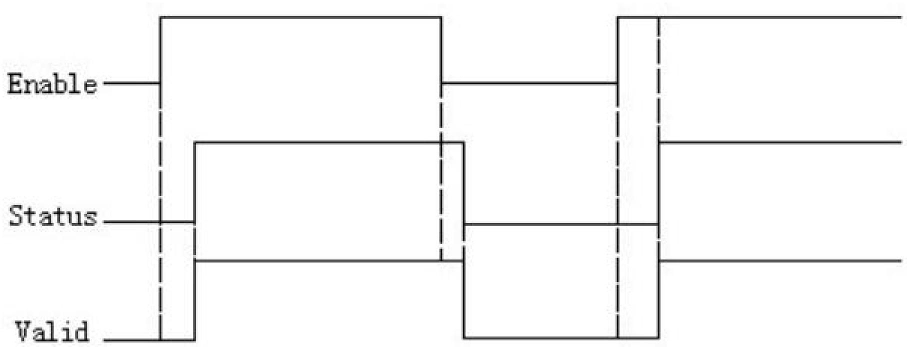
（3）由上述试验表格得出模拟量输入与数据量的关系图



➢ **输出参数**

名称	功能	数据类型	输出范围
Status （状态位）	该输出参数为 TRUE 时 表示指令正在执行中	BOOL	TRUE 或 FALSE
Valid （使能有效位）	该输出参数为 TRUE 时 表示指令正在控制轴	BOOL	TRUE 或 FALSE
ADC_Value （模拟量转数据量的当前值）	该输出为当前模拟量转化 成数据量的值	DINT	0~4096
Error（错误）	该指令输出为 TRUE 时表示指 令的执行出错	BOOL	TRUE 或 FALSE
ErrorID （错误代码 ID）	执行指令出错时的错误代码	WORD	-

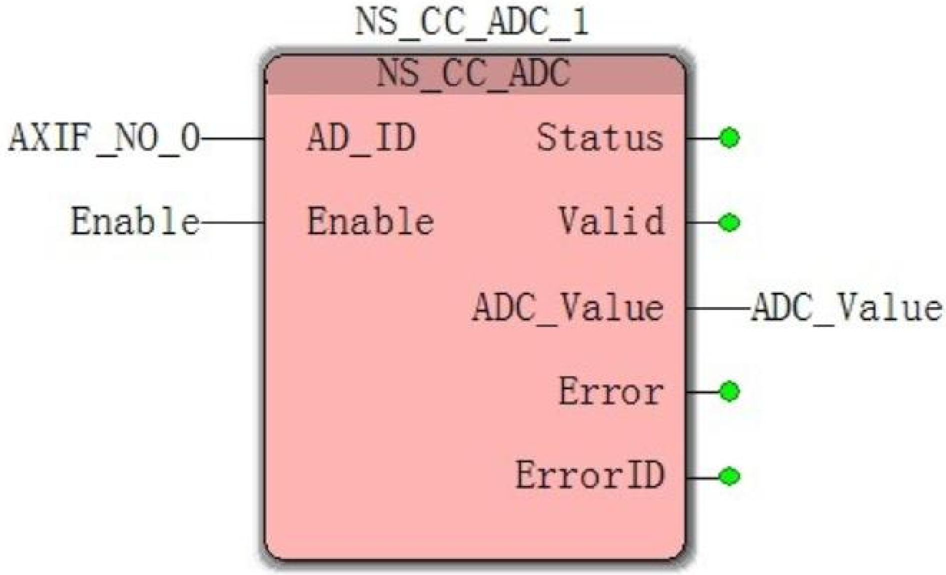
- **输出参数时序变化图**



程序范例

AI0 输入 5V，指令转换成数据量 ADC_Value 值

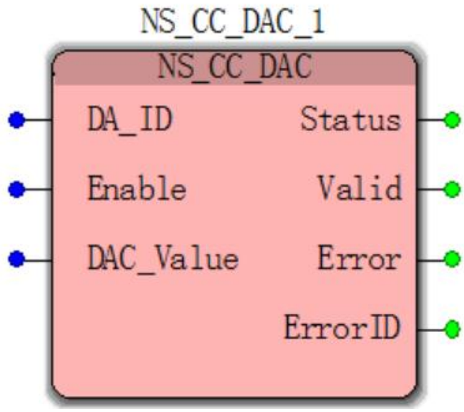
变量名	数据类型	初始值
NS_CC_ADC_1	AXIF_NO_0	
AXIF_NO_0	WORD	0
Enable	BOOL	



情形 1：当 Enable 由 FALSE 变为 TRUE 时，外部 AI0 输入 DC5V 时，控制器将 5V 电压转换成数字量输出 ADC_Vable 将显示 3070（可能存在误差，以实际为准）

11.6.2 NS_CC_DAC(DA 指令)

FB/FC	说明	适用机种
FB	该指令用于将数据量转换成模拟量进行输出 （输出电压范围±10V）	VEC-VA-MP-005-MA



➢ **输入参数**

名称	功能	数据类型	设定范围 （缺省值）	生效时机
AD_ID （模拟量输入号）	选择模拟量口作为输出， 数值 0~3 对应 AXIS0~AXIS3 轴	WORD	0~3 (不可缺省)	Enable 为 TRUE
Enable （执行位）	当 Enable 为 TRUE 时，执 行该指令	BOOL	TRUE 或 FALSE	
DAC_Value （数据量设置）	设定数据值	DINT	-2048~2047	Enable 为 TRUE

说明：

（1） 数字量转模拟量对应关系 如下：

数字量输入 -2048～0 0~2047

模拟量输出 -10V~0V 0V~10V

如果 VA 为经济型 VA，数字量转模拟量对应关系 如下：

数字量输入 -32768～0 0~32767

模拟量输出 -10V~0V 0V~10V

（2） 本指令执行时，变量 DAC_Value 值为 0 时，模拟量输出标准 0V。

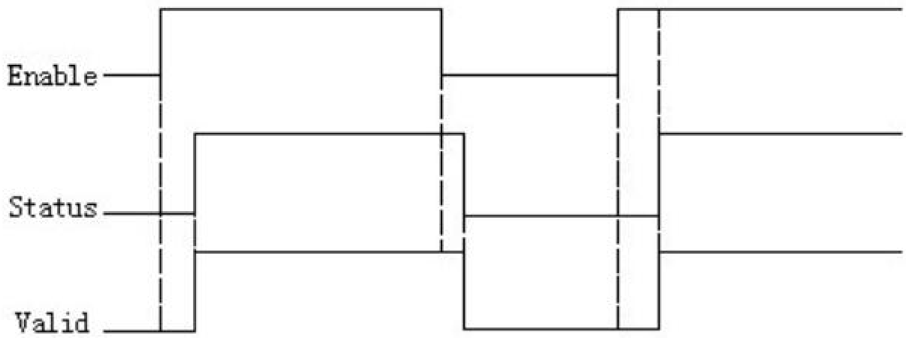
（3）本指令是开环控制， 模块执行后， 根据 DAC_Value 中数值输入相应的电压。

➢ **输出参数**

名称	功能	数据类型	输出范围
Status （状态位）	该输出参数为 TRUE 时 表示指令正在执行中	BOOL	TRUE 或 FALSE
Valid	该输出参数为 TRUE 时	BOOL	TRUE 或 FALSE

（使能有效位）	表示指令正在控制轴		
Error（错误）	该指令输出为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE
ErrorID （错误代码 ID）	执行指令出错时的错误代码	WORD	-

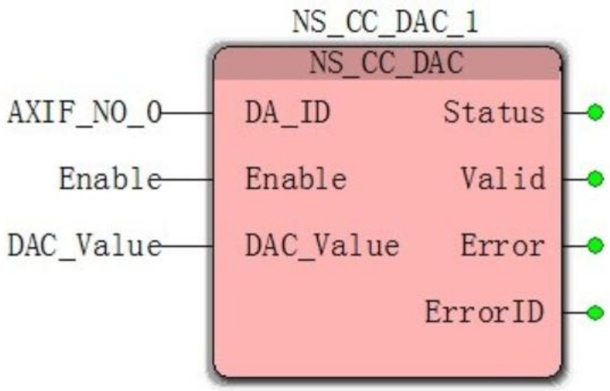
- **输出参数时序变化图**



程序范例

万用表测试第一路 DA 输出 5V

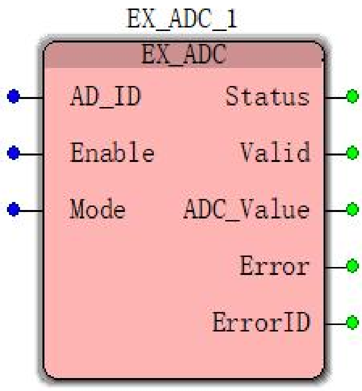
变量名称	数据类型	初始值
NS_CC_DAC_1	NS_CC_DAC	
AXIF_NO_0	WORD	0
Enable	BOOL	
DAC_Value	DINT	0



情形 **1**：当 Enable 由 FALSE 变为 TRUE 时，Valid 由 FALSE 变为 TRUE，且后一个周期，此时模拟量量输出 0V，修改变量 DAC_Value 为 1024 时，模拟量输出 5V。

11.6.3 EX_ADC（AD 扩展指令）

FB/FC	说明	适用机种
FB	此指令用于将 AD 扩展模块的模拟量输入转换成数据量进行输出	VEC-VA-MP-005-MA



➢ **输入参数**

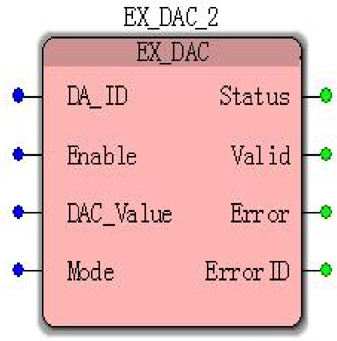
名称	功能	数据类型	设定范围（缺省值）	生效时机
AD_ID（模拟量输入号）	选择需要转的模拟量口 AI4~AI36，其中数值 4~7 对应第一个 AD 扩展模块的 V0~V4	WORD	4~36 (不可缺省)	Enable 为 TRUE
Enable（执行位）	当 Enable 为 TRUE 时，执行该指令	BOOL	TRUE 或 FALSE	
Mode（模式）	EX_ADC 不同模式下 0，输入电压 0~5 对应 0~4095 1，输入电压 0~10V 对应 0~4095 2，输入电压±10V 对应 0~4095 3，输入电流 4~20mA 对应 819~4095	WORD	0~3 (不可缺省)	

➢ **输出参数**

名称	功能	数据类型	输出范围
Status （状态位）	该输出参数为 TRUE 时 表示指令正在执行中	BOOL	TRUE 或 FALSE
Valid （使能有效位）	该输出参数为 TRUE 时 表示指令正在控制轴	BOOL	TRUE 或 FALSE
ADC_Value （模拟量转数据量的当前值）	该输出为当前模拟量转化 成数据量的值	DINT	0~4096
Error（错误）	该指令输出为 TRUE 时表示指 令的执行出错	BOOL	TRUE 或 FALSE
ErrorID （错误代码 ID）	执行指令出错时的错误代码	WORD	-

11.6.4 EX_DAC（DA 扩展模块）

FB/FC	说明	适用机种
FB	该指令用于将DA扩展模块的数据量转换成模拟量进行输出	VEC-VA-MP-005-MA



➢ **输入参数**

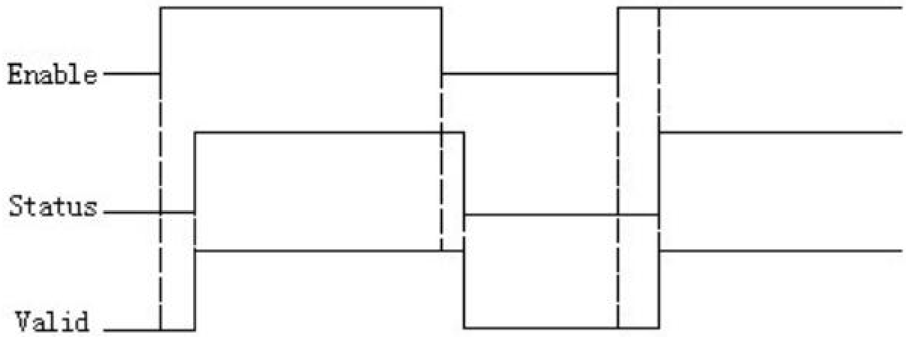
名称	功能	数据类型	设定范围（缺省值）	生效时机
DA_ID（模拟量输入号）	选择需要转的模拟量口 AO4~AO36，其中数值 4~7 对应第一个 DA 扩展模块的 AO0~AO4	WORD	4~36 (不可缺省)	Enable 为 TRUE
Enable（执行位）	当 Enable 为 TRUE 时，执行该指令	BOOL	TRUE 或 FALSE	
DAC_Value（数据量设置）	设定数据值	DINT	0~4095	Enable 为 TRUE
Mode（模式）	EX_DAC 不同模式下 0，0~4095 对应输出电压--10V~+10V 1，0~4095 对应输出电压-0V~+5V 2，0~4095 对应输出电压-0V~+10V	WORD	0~2 (不可缺省)	

➢ **输出参数**

名称	功能	数据类型	输出范围
Status（状态位）	该输出参数为 TRUE 时表示指令正在执行中	BOOL	TRUE 或 FALSE
Valid（使能有效位）	该输出参数为 TRUE 时表示指令正在控制轴	BOOL	TRUE 或 FALSE
Error（错误）	该指令输出为 TRUE 时表示指	BOOL	TRUE 或 FALSE

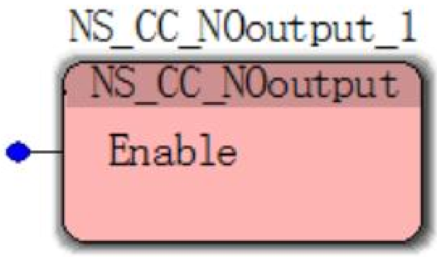
	令的执行出错		
ErrorID （错误代码 ID）	执行指令出错时的错误代码	WORD	-

- **输出参数时序变化图**



11.6.5 NS_CC_NOoutput（禁止 QXX 输出指令）

FB/FC	说明	适用机种
FB	该指令禁止全部 QX 输出（X:表示 0~128）	VEC-VA-MP-005-MA



- **输入参数**

名称	功能	数据类型	设定值范围（不可缺省值）	生效时机
Enable（禁止有效位）	TRUE 时禁止全部 QX 输出，FALSE 时允许 QX 输出。	BOOL	TRUE 或 FALSE	

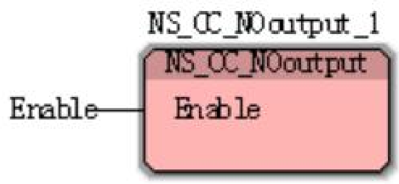
功能说明：

（1）该指令用于禁止全部的 QXX 输出，使用者根据现场需求进行选择。

（2）当 Enable 为 TRUE 的时候，全部 DO，将被禁止输出，当 Enable 为 FALSE 的时候，DO 将恢复执行该模块之前的状态（禁止是全部 QX，无法指定单个 QX.X）；

程序范例

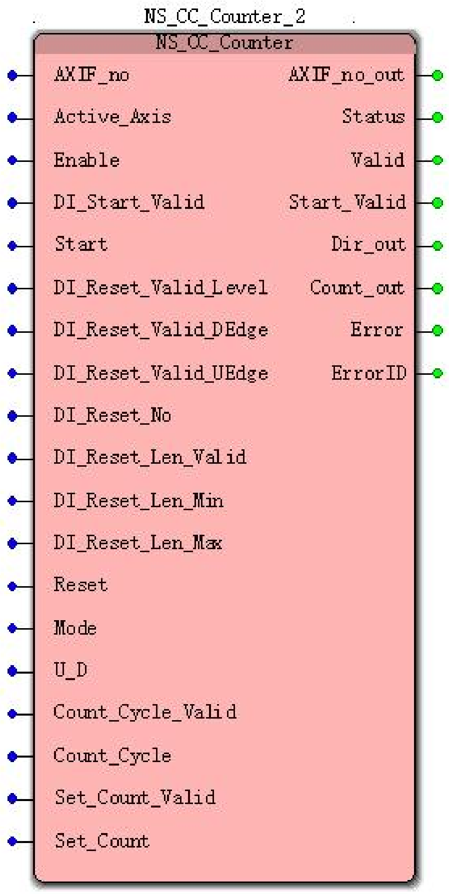
变量名称	数据类型	初始值
NS_CC_NOoutput	NS_CC_NOoutput	
Enable	BOOL	



情形 **1**：当 Enable 由 FALSE 变为 TRUE 时，全部数字量输出将被禁止输出，当 Enable 由 TRUE 变为 FALSE 时，QXX 将恢复之前的状态。

11.6.6 NS_CC_Counter（高速计数器指令）

FB/FC	说明	适用机种
FB	用于脉冲个数计数，计数方式为硬件计数不受扫描周期的影响，其最高计数频率为 1MHz	VEC-VA-MP-005-MA



➢ **输入参数**

名称	功能	数据类型	设定范围（缺省值）	生效时机
AXIF_no （计数轴号）	设定计数器轴号	WORD	0~6 (0)	Enable 由 FALSE 变为 TRUE 时
Active_Axis （当前计数轴）	设定脉冲计数来源: 0： AXIS0; 1： AXIS1; 2： AXIS2; 3： AXIS3 4： AXIS4	WORD	0~4	Enable 由 FALSE 变为 TRUE 时
Enable （执行位）	当 Enable 由 FALSE 变为 TRUE 时，执行该指令。	BOOL	TRUE 或 FALSE	-

DI_Start_Valid（外部 DI 启动有效位）	当 DI_Start_Valid 由 FALSE 变为 TRUE 时，允许指定外部 DI 有效时，启动高速计数器（DI 已经指定，详细往下说明）	BOOL	TRUE 或 FALSE	-
Start（启动位）	当 Start 为 TRUE 时，启动高速计数器	BOOL	TRUE 或 FALSE	-
DI_Rest_Valid_Level（外部 DI 清零高电平有效位）	当 DI_Reset_Valid_Level 为 TRUE 时，DI 清零为高电平清零	BOOL	TRUE 或 FALSE	-
DI_Rest_Valid_DEdge（外部 DI 清零下降沿有效位）	当 DI_Reset_Valid_DEdge 为 TRUE 时，DI 清零为下降沿清零	BOOL	TRUE 或 FALSE	-
DI_Rest_Valid_UEdge（外部 DI 清零上升沿有效位）	当 DI_Reset_Valid_UEdge 为 TRUE 时，DI 清零为上升沿清零	BOOL	TRUE 或 FALSE	-
DI_Reset_No（外部 DI 输入清零号）	指定外部清零信号端子，输入值 0~7 对应输入点 I0~I7、8~15 对应输入点 I10~I17	WORD	TRUE 或 FALSE	-
DI_Rest_Len_Valid（外部 DI 清零区间有效位）	当 DI_Rest_Len_Valid 为 TRUE 时，DI 清零信号只有在清零区间内才有效	BOOL	TRUE 或 FALSE	-
DI_Reset_Len_Min（外部 DI 清零区间下限）	设定 DI 清零信号有效的区间下限	WORD		-
DI_Reset_Len_Max（外部 DI 清零区间上限）	设定 DI 清零信号有效的区间上限	WORD		-
Reset（清零位）	当 Reset 为 TRUE 时，清除计数器当前值	BOOL	TRUE 或 FALSE	-
Mode（模式选择）	保留	DINT	保留	-
U_D（计数方向选	保留	DINT	保留	-

择）				
Count_Cycle_Valid （循环计数有效位）	当 Count_Cycle_Valid 为 TRUE 的时候，启动计数周期循环，当计数器值到达计数器 Count_Cycle 设定值时，自动清零从新计数，不断地循环。 Count_Cycle_Valid 为 FALSE 时不启用该功能。	BOOL	TRUE 或 FALSE	-
Count_Cycle （设定脉冲循环计数值）	当 Count_Cycle_Valid 为 TRUE 的时候，启动计数周期循环，当计数器值到达计数器 Count_Cycle 设定值时，自动清零从新计数，不断地循环。 Count_Cycle_Valid 为 FALSE 时不启用该功能。	DINT	正数、负数	--
Set_Count_Valid （设定当前计数脉冲值有效位）	当 Set_Count_Valid 由为 TRUE 时，将高速计数器当前值设置为 Set_Count 值	BOOL	TRUE 或 FALSE	-
Set_Count （设定当前脉冲计数值）	设定当前高速计数器值	DINT	正数、负数、0	-

说明：

（1)运动控制器集成有 5 路硬件高速计数器口分别为 AXIS0~AXIS4（目前只支持 AB 相脉冲），纯硬件计数不受程序扫描周期的影响，其最高计数频率为 1MHz。5 路计数器口可以调用 7 高速计数器模块 AXIF_no（计数器号）分别为 0~6，计数方式是相互对立，互不干扰。

（2）该指令集成了丰富的功能，包含有外部 DI 启动，清零计数器，变量启动和清零计数器，循环计数功能，置数功能，

（3）计数器脉冲来源可通过 Active_Axis（当前计数轴）进行选择，任意指定，而且可以实现跨轴计数功能。

（4）高速计数器功能集成丰富，可根据要求进行选用，未使用的功能输入参数保留空白即可。

（5）外部 DI 端子启动高速计数器内部已经指定，使用高数计数器的时候，当 DI_Start_Valid 为 TRUE 时，外部 DI 端子被占用做为启动高速计数器的输入端子，当 DI_Start_Valid 为 FALSE 时，可当普通 I/O 使用，具体分配如下

AXIF_no(轴号)	指定外部 DI 端子	DI_Reset_No

	作为启动高数计数器信号	
0	DI0.3	可选择（不允许跟启动的 DI 重复）
1	DI0.5	可选择（不允许跟启动的 DI 重复）
2	DI0.7	可选择（不允许跟启动的 DI 重复）
3	DI1.1	可选择（不允许跟启动的 DI 重复）
4	DI1.3	可选择（不允许跟启动的 DI 重复）
5	DI1.5	可选择（不允许跟启动的 DI 重复）
6	DI1.7	可选择（不允许跟启动的 DI 重复）

➢ **输出参数**

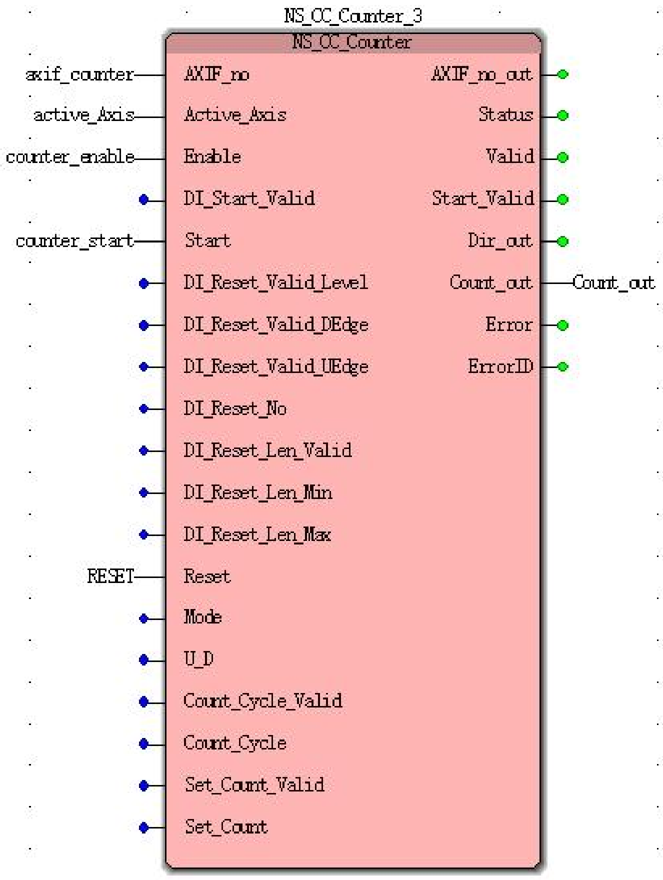
名称	功能	数据类型	输出范围
AXIF_no_out (轴号输出)	输出当前计数轴号	WORD	0~6
Status （状态位）	该输出参数为 TRUE 时表示指令正在执行中	BOOL	TRUE 或 FALSE
Valid （使能有效位）	该输出参数为 TRUE 时表示指令正在控制轴	BOOL	TRUE 或 FALSE
Start_Valid	保留	BOOL	-
Dir_out	保留	BOOL	-
Count_out	当前脉冲计数值	DINT	正数、负数、0
Error	该指令输出为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE
ErrorID	执行指令出错时的错误代码	WORD	-

程序示范 I

例：变量启动和清零计数 axif_counter 轴脉冲

1、变量和程序

变量名称	数据类型	初始值
NS_CC_Counter_3	NS_CC_Counter	
axif_counter	WORD	0
active_Axis	WORD	0
counter_enable	BOOL	1
counter_start	BOOL	1
RESET	BOOL	



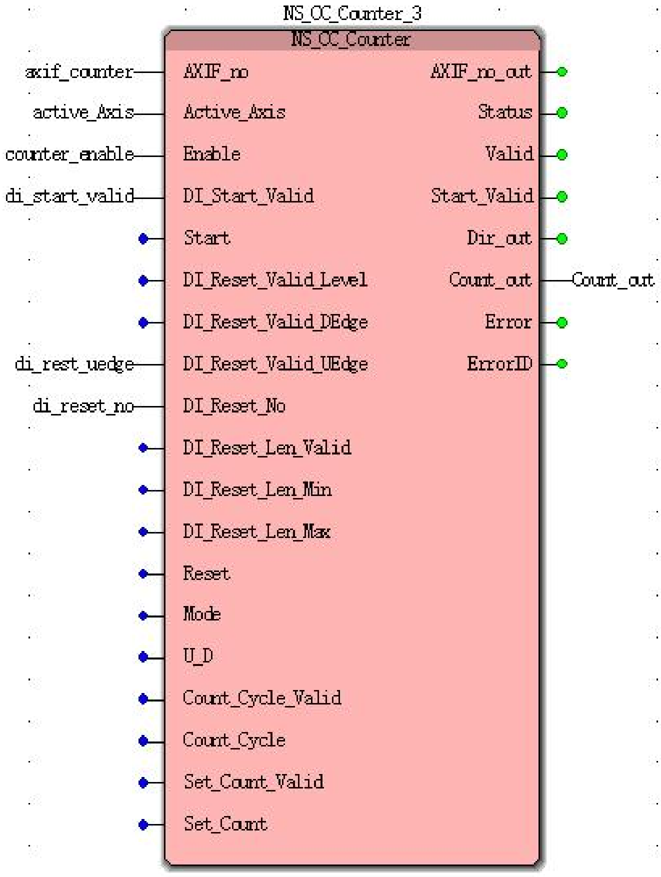
情形一：当 counter_enable 由 FALSE 变为 TRUE 时，且保持，counter_start 由 FALSE 变为 TRUE 开始允许高速计数器计数，Count_out 显示当前脉冲计数值。当 RESET 由 FALSE 变为 TRUE 时，Count_out 显示当前脉冲计数值为 0，此时脉冲数输入时，Count_out 显示当前脉冲计数值一直保持为 0，当 RESET 由 TRUE 变为 FALSE 时，继续开始允许计数。

程序示范 II

例：外部 DI3 端子启动和外部 DI0 清零计 axif_counter 轴脉冲数

1、变量和程序

变量名称	数据类型	初始值
NS_CC_Counter_3	NS_CC_Counter	
axif_counter	WORD	0
active_Axis	WORD	0
counter_enable	BOOL	1
di_start_Valid	BOOL	1
di_reset_uedge	BOOL	1
di_reset_no	BOOL	0



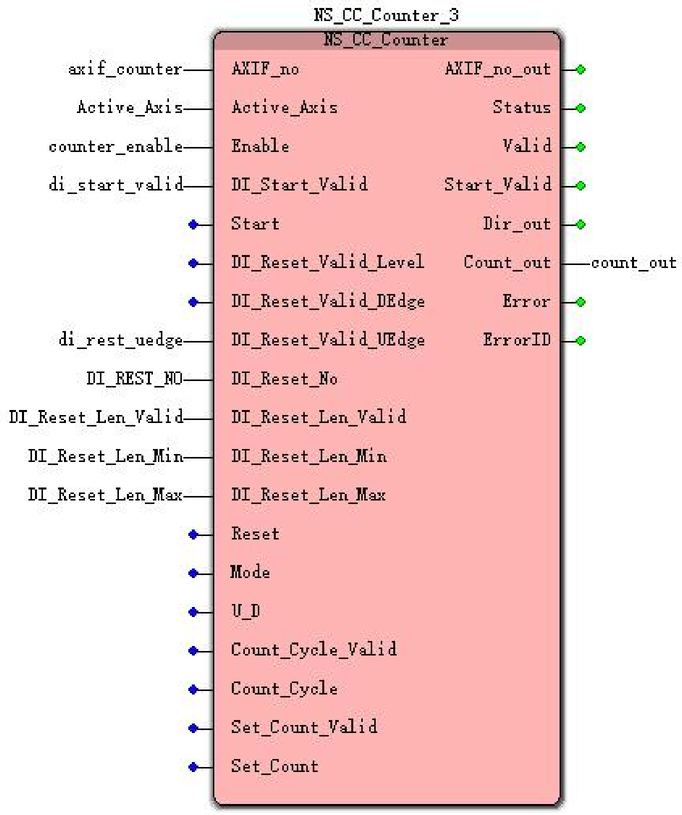
情形一：当 counter_enable 为由 FALSE 变为 TRUE 的时，且保持。di_start_Valid 变为 TRUE，当外部信号 DI3 信号有效，高数计数器 3 开始计数,Count_out 显示当前脉冲，当 di_reset_no 指定的 DI0 出现上升沿的时候，将清零高速计数器的当前值。

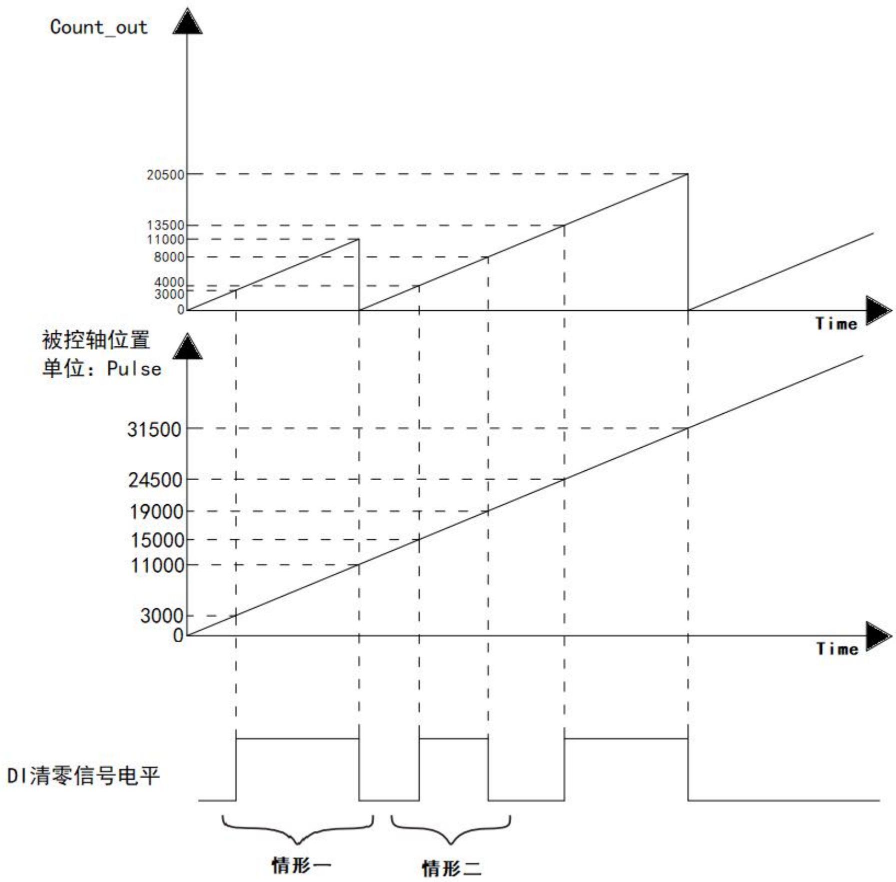
程序示范III

例：外部 DI3 端子启动和外部 DI0 清零，并且设定外部 DI 清零区间有效计 axif_counter 轴脉冲数

1、变量和程序

变量名称	数据类型	初始值
NS_CC_Counter_3	NS_CC_Counter	
axif_counter	WORD	0
active_Axis	WORD	0
counter_enable	BOOL	1
di_start_Valid	BOOL	1
di_reset_uedge	BOOL	1
di_reset_no	BOOL	0
DI_Rest_Len_Valid	BOOL	1
DI_Rest_Len_Min	WORD	5000
DI_Rest_Len_Max	WORD	10000





情形一：当 counter_enable 为由 FALSE 变为 TRUE 的时，且保持。di_start_Valid 变为 TRUE，当外部信号 DI3 信号有效，高数计数器 3 开始计数,Count_out 显示当前脉冲，当 di_reset_no 指定的 DI0 由 FALSE 变为 TRUE 直到 DI0 出现再次变为 FALSE，这期间被控轴走过的位移（脉冲）为 8000（11000-3000），在 DI_Rest_Len_Min~DI_Rest_Len_Max 所设定的区间内，因此 DI0 再次变为 FALSE 时，对 Count_out 进行清零操作。

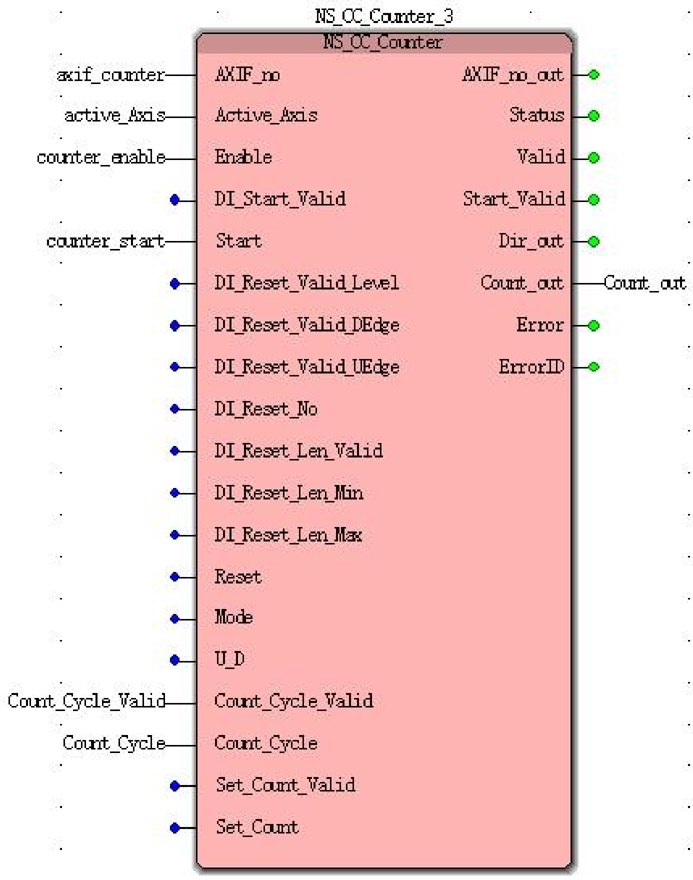
情形二：当 counter_enable 为由 FALSE 变为 TRUE 的时，且保持。di_start_Valid 变为 TRUE，当外部信号 DI3 信号有效，高数计数器 3 开始计数,Count_out 显示当前脉冲，当 di_reset_no 指定的 DI0 由 FALSE 变为 TRUE 直到 DI0 出现再次变为 FALSE，这期间被控轴走过的位移（脉冲）为 4000（19000-15000），不在 DI_Rest_Len_Min~DI_Rest_Len_Max 所设定的区间内，因此 DI0 再次变为 FALSE 时，不会对 Count_out 进行清零操作。

程序示范Ⅳ

例：循环计数模式计 axif_counter 轴脉冲数

1、变量和程序

变量名称	数据类型	初始值
NS_CC_Counter_3	NS_CC_Counter	
axif_counter	WORD	0
active_Axis	WORD	0
counter_enable	BOOL	1
counter_start	BOOL	1
Count_Cycle_Valid	BOOL	1
Count_Cycle	DINT	5000



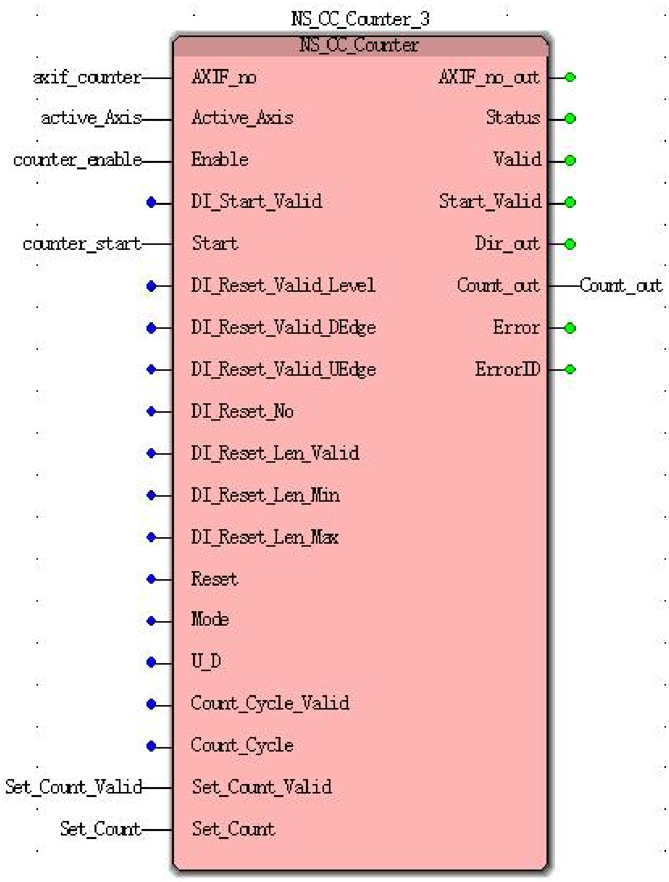
情形一：当 counter_enable 由 FALSE 变为 TRUE 时，且保持，counter_start 由 FALSE 变为 TRUE 开始允许高速计数器计数，Count_out 显示当前脉冲计数值。当高速计数器当前的值为 5000 时，计数器将自动清零重新计数，不断地循环。

程序范 V

置数模式计 axif_counter 轴脉冲数

1、变量和程序

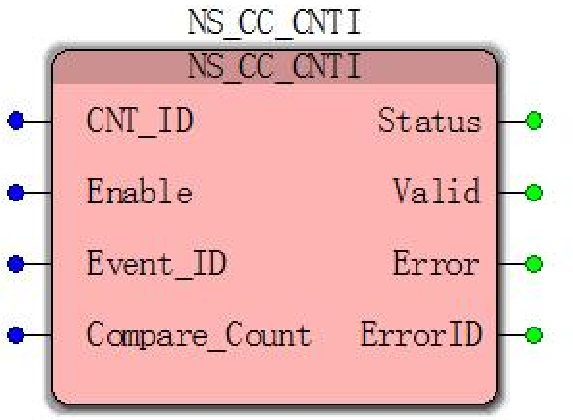
变量名称	数据类型	初始值
NS_CC_Counter_3	NS_CC_Counter	
Axif_counter	WORD	0
active_Axis	WORD	0
counter_enable	BOOL	1
Star	BOOL	1
Set_Count_Valid	BOOL	
Set_Count	DINT	10000



情形一：当 counter_enable 由 FALSE 变为 TRUE 时，counter_start 由 FALSE 变为 TRUE 开始允许高速计数器计数，Count_out 显示当前脉冲计数值。当 Set_Count_Valid 变量由 FALSE 变 TRUE 时，高速计数器 Count_out 当前值将被修改 10000，且一直保持 10000，当 Set_Count_Valid 变量由 TRUE 变 FALSE 时，当脉冲进来时，计数脉冲数将在 10000 的基础上继续递增或者递减。

11.6.7 NS_CC_CNTI(高速计数中断指令)

FB/FC	说明	适用机种
FB	此指令用于高速计数器当前值到达脉冲设定个数时产生中断，进入中断程序	VEC-VA-MP-005-MA



- **输入参数**

名称	功能	数据类型	设定值范围（不可缺省值）	生效时机
CNT_ID（关联计数器号）	指定关联高数计号，该值等于计数器号自动关联	WORD	0~1（不可或缺）	Execute 由 FALSE 变为 TRUE
Enable（执行位）	当 Enable 由 FALSE 变为 TRUE 时，执行该指令。	BOOL	TRUE 或 FALSE	Execute 由 FALSE 变为 TRUE
Event_ID（中断）	设定中断事件号（0-5）与中断程序事件号一致。	WORD	0~5	Execute 由 FALSE 变为 TRUE
Compare_Count（设定脉冲个数）	设定中断脉冲个数	DINT	正数（不可缺省）	Execute 由 FALSE 变为 TRUE

- **输出参数**

名称	功能	数据类型	输出范围
Status（状态位）	该输出参数为 TRUE 时表示指令正在执行中	BOOL	TRUE 或 FALSE
Valid（有效位）	该输出参数为 TRUE 时表示指令正在控制轴	BOOL	TRUE 或 FALSE

Error（误差）	保留	BOOL	TRUE 或 FALSE
ErrorID（错误代码）	保留	WORD	保留

说明：

（1）NS_CC_CNTI 指令（计数中断指令）需要关联 NS_CC_Counter 指令（高速计数器）一起使用，否者无法实现。

（2）NS_CC_CNTI 指令（计数中断指令）关联 NS_CC_Counter 指令（高速计数器）时，需要将 CNT_ID（关联计数器轴号）填写与高速计数器 AXIF_no（计数轴号）一致即可

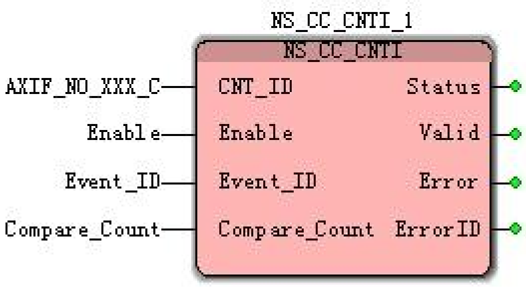
（3）计数器中断指令目前支持两路，**目前高速计数器号为 0 和 1 支持该功能**，剩下的 2-6 暂未支持计数中断功能。

程序范例（0 号高数计数器中断设置）

变量名称	数据类型	初始值
NS_CC_CNTI_1	NS_CC_CNTI	
AXIF_NO_XXX_C	WORD	0
Enable	BOOL	1
Event_ID	WORD	0
Compare_Count	DINT	5000

第一步.调用高速计数器 NS_CC_Counter_1（可参考 1.4.4 需要注意计数器和本指令轴号波保持一致，这里将不重复写高数计数器变量）

第二步.调用 NS_CC_CNTI_1 模块，如下



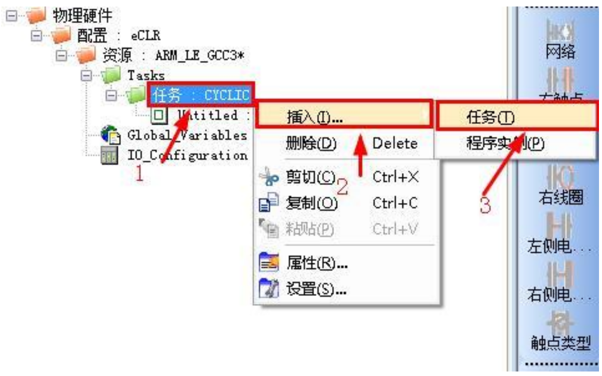
第三步.建立中断程序， 在“工程树窗口”右键“逻辑 POU” 选择“插入”然后选择“程序”在弹出来的窗口给中断程序命名为“EN1”（用户可以自行命名），以及选择编写中断程序的语言“LD 语言”如下图 10-10。

第四步：完成后，单击确定，“逻辑 POU”下将会自动添加一个以“EN1”为名称程序如图 10-11 所示

第五步：刚才插入的“EN1”程序还是属于主程序的范畴，因为新建的“POU”程序软件自动加载到新建工程时选择的“任务”CYCLIC 类型下；如图 10-13 所示

第六步：在“工程树窗口”下选择 “硬件”，在资源“任务”下，右键删除“EN1”程序，单击“确定”我们需要另外新建“任务类型”，如图图 10-14 所示

第七步： 新建“任务类型” 右键“任务”，单击“插入”，然后选择“任务”如图 10-15 所示



第八步：在弹出来的“插入”对话框在名称填写“ZD”（用户可以自行命名） 任务类型选择“EVENT”完成后单击“确定”。如图图 10-16 所示

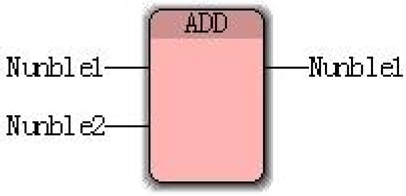
第九步：在弹出来的“ZD 的任务设置”选择事件号这里选择“事件 0”由中断模块上的“Event_ID”填写的数值决定，这里选择 0（程序示范的“Event_ID=0”）如图 10-17 所示（特别注意中断的优先权不能和主程序的优先级一样否者编译不通过或者是下载 97%报错误提示）

第十步：完后插入程序实例，右键任务下“EVENT”，选择“插入”，然后选择“程序实例”如图 10-18 所示

第十一步：在弹出来的“插入框”，为程序实例命名为“ZD1”，程序类型选择新建的“EN1”单击“确定”最终的结果如图 10-19 所示

第十二步：返回“工程树窗口”下的“工程”窗口双击 POU 下的 EN1 加入一个加法模块，（目的是验证中断程序是否被正确执行）

变量名	数据类型	初始值
ADD	ADD	
Nunble1	INT	
Nunble2	INT	1

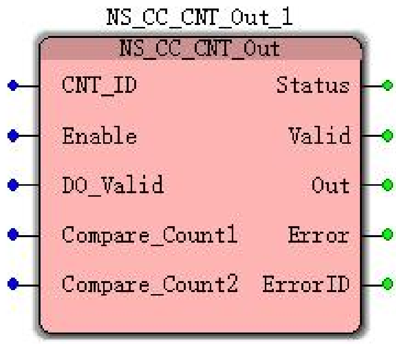


到此中断程序创建和关联已经完成，下面是分析中断程被执行的过程。

情形 1：启动高数计数器，允许其计数，当 NS_CC_CNTI_1 模块上 Enable 由 FALSE 变为 TRUE 时，允许中断打开，当高数计数器当前值到达中断设定值 5000 的时候，中断程序被执行一次，此时变量 Nunble1 值为 1， 完成后将跳出中断程序，返回主程序。高数计数器当前值再次变为 5000 时，中断程序再次被执行，此时变量 Nunble1 值为 2。

11.6.8 NS_CC_CNT_Out（区间比较输出指令）

FB/FC	说明	适用机种
FB	该指令用于高速计数器当前值到达设定区间内立即输出，DO 输出不受扫描周期影响	VEC-VA-MP-005-MA



- **输入参数**

名称	功能	数据类型	设定值范围（不可缺省值）	生效时机
CNT_ID（关联计数器轴号）	关联相应的高速计数器轴号	WORD	0~6（不可缺省）	-
Enable（执行位）	当 Enable 由 FALSE 变为 TRUE 时，执行该指令。	BOOL	TRUE 或 FALSE	-
DO_Valid（允许 Q 输出有效位）	该值为 TRUE 时，允许 Q 输出有效	BOOL	TRUE 或 FALSE	-
Compare_Count1（设定区间最小值 min）	设定区间脉冲 Min 最小值（单位：个）	DINT	正数	-
Compare_Count2 (设定值区间最大值 Max)	设定区间脉冲 Max 最大值（单位：个）	DINT	正数	-

说明：

（1）NS_CC_CNT_Out 指令（区间比较输出指令）需要关联 NS_CC_Counter 指令（高速计数器）一起使用，单独使用时没有任何意义。

（2）NS_CC_CNT_Out 指令（区间比较输出指令）关联 NS_CC_Counter 指令（高速计数器），设置需要将 CNT_ID（关联计数器轴号）填写与高速计数器 AXIF_no（计数轴号）一致即可。

（3）区间比较输出的 QXX 信号不受扫描周期影响，采用内部 FPAG 计数满足条件后输出。

（4）计数器区间输出的 DO_ID 号，内部已经指定，指定的关系如下（被指定的 DO_ID，不允许当普通的 DO 使用）

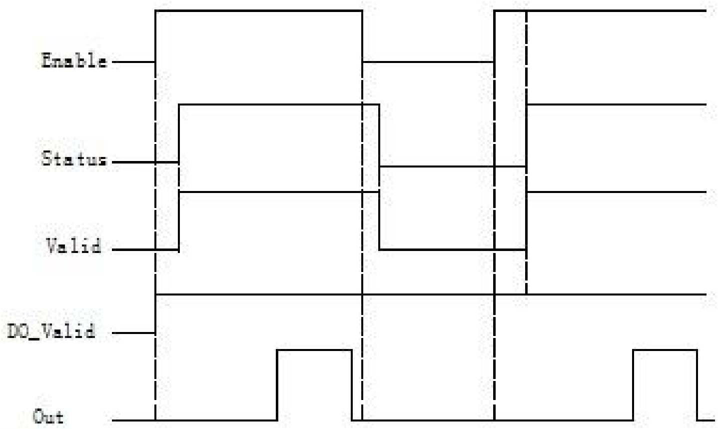
CNT_ID（关联计数器轴号）	QXX（输出）
0	Q0.0
1	Q0.1
2	Q0.2
3	Q0.3
4	Q0.4

5	Q0.5
6	Q0.6

- **输出参数**

名称	功能	数据类型	输出范围
Status（状态位）	该输出参数为 TRUE 时表示指令正在执行中	BOOL	TRUE 或 FALSE
Valid（有效位）	该输出参数为 TRUE 时表示指令正在控制轴	BOOL	TRUE 或 FALSE
Out（输出信号）	该参数输出 TRUE 时表示 Q 正在输出中	BOOL	TRUE 或 FALSE
Error（错误）	保留	BOOL	保留
ErrorID（错误代码）	保留	WORD	保留

- **输出参数时序变化图**



情形 1:当 Enable 由 FALSE 变为 TRUE 时，Status 和 Valid 变为 TRUE 且后一个周期，当 Enable 由 TRUE 变为时 FALSE，Status 和 Valid 由 FALSE 变为 TRUE 且后一个周期。
情形 2.当 DO_Valid 由 FALSE 变为 TRUE 时，条件满足后 Out 由 FALSE 变为 TRUE，同时 Q0.0 有输出,当条件不满足的时候 Out 自动由 TRUE 变为 FALSE 同时 Q0.0 无输出。

程序范例

例：高速计数器区间在 1000~5000 之间输出 Q0.0;

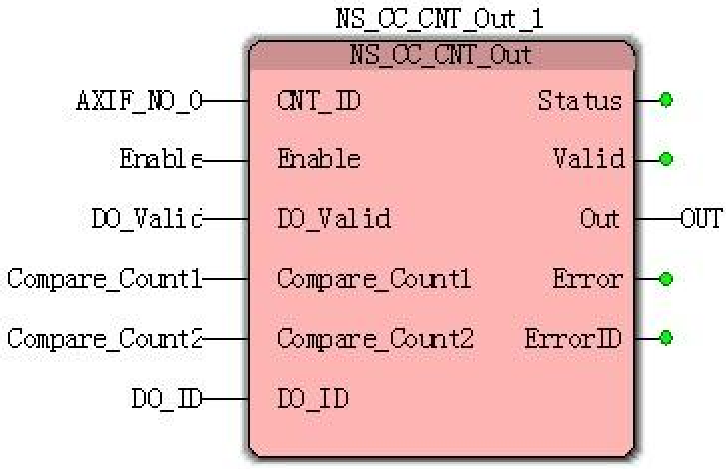
1、变量和程序

变量名	数据类型	初始值
AXIF_NO_0	WORD	0
Enable	BOOL	

DO_Valid	BOOL	1
Compare_Count1	DONT	1000
Compare_Count2	DONT	5000
DO_ID	WORD	0
OUT	BOOL	

第一步.调用高速计数器 NS_CC_Counter_1 （可参考 11.7.7 需要注意计数器和本指令轴号波保持一致，这里将不重复写计数器变量）

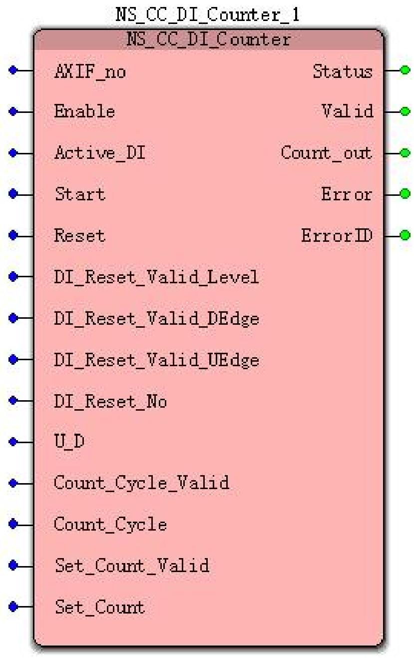
第二步.调出 NS_CC_OUT_1 模块，配置如下



情形 1：启动 NS_CC_Counter_1（高速计数器模块），当 Enable 由 FALSE 变为 TRUE 时，执行该指令模块，当高速计数器当前脉冲值在 1000 和 5000 时，Q0.0 将有输出，OUT 将由 FALSE 变为 TRUE,当前脉冲值在 1000 和 5000 之外，Q0.0 将不输出，OUT 将由 TRUE 变为 FALSE.

11.6.9 NS_CC_DI_Counter（DI 高速计数指令）

FB/FC	说明	适用机种
FB	此指令对 DI 口脉冲进行高速计数	VEC-VA-MP-005-MA



➢ **输入参数**

名称	功能	数据类型	设定范围（缺省值）	生效时机
AXIF_no（计数轴号）	设定计数器轴号	WORD	0~6	Enbale 为 TRUE
Enbale（执行位）	当 Enable 由 FALSE 变为 TRUE 时，执行该指令。	BOOL	TRUE 或 FALSE	
Active_DI（DI 输入通道号）	设定 DI 高速计数口	WORD	0~15	
Start（启动位）	当 Start 为 TRUE 时，启动 DI 高速计数器	BOOL	TRUE 或 FALSE	
Reset（清零位）	当 Reset 为 TRUE 时，清除计数器当前值	BOOL	TRUE 或 FALSE	
DI_Reset_Valid_Level（外部 DI 清零电平有效位）	当 DI_Reset_Valid_Level 为 TRUE 时，DI 清零为电平清零	BOOL	TRUE 或 FALSE	
DI_Reset_Valid_DEdge（外部清零下降沿有	当 DI_Reset_Valid_DEdge 为 TRUE 时，DI 清零为下降沿清零	BOOL	TRUE 或 FALSE	

效位）				
DI_Reset_Valid_UEdge （外部 DI 清零上升沿有效位）	当 DI_Reset_Valid_UEdge 为 TRUE 时，DI 清零为上升沿清零	BOOL	TRUE 或 FALSE	
DI_Reset_No （外部 DI 清零）	指定外部清零信号端子，输入值 0~7 对应输入点 I0~I7、8~15 对应输入点 I10~I17	WORD	0~15	
U_D （计数方向选择位）	指定计数器的计数方向 0:负方向计数 1：正方向计数	DINT	0 或 1	
Count_Cycle_Valid （循环计数有效位）	当 Count_Cycle_Valid 为 TRUE 的时候，启动计数周期循环，当计数器值到达计数器 Count_Cycle 设定值时，自动清零从新计数，不断地循环。 Count_Cycle_Valid 为 FALSE 时不启用该功能。	BOOL	TRUE 或 FALSE	
Count_Cycle （设定 DI 循环计数值）	当 Count_Cycle_Valid 为 TRUE 的时候，启动计数周期循环，当计数器值到达计数器 Count_Cycle 设定值时，自动清零从新计数，不断地循环。 Count_Cycle_Valid 为 FALSE 时不启用该功能。	DINT	正数、负数	
Set_Count_Valid （设定当前 DI 计数有效位）	当 Set_Count_Valid 由为 TRUE 时，将高速计数器当前值设置为 Set_Count 值	BOOL	TRUE 或 FALSE	
Set_Count （设定当前 DI 计数值）	设定当前高速计数器值	DINT	正数、负数、0	

说明：

（1）运动控制器本体有 16 个数字输入 DI，纯硬件计数不受程序扫描周期的影响，其最高输入频率为 5KHz。可以调用 8 个高速计数器模块 AXIF_no（计数器号）分别为 0~6，计数方式是相互对立，互不干扰。

（2）该指令集成了丰富的功能，包含有变量启动/复位、外部 DI 清零计数和计数方向选择、循环计数功能、置数功能，

（3）计数器脉冲来源可通过 Active_DI（DI 输入通道号）进行选择，在 0~15 任意指定。
（4）DI 高速计数器功能集成丰富，可根据要求进行选用，未使用的功能输入参数保留空白即可。

➢ **输出参数**

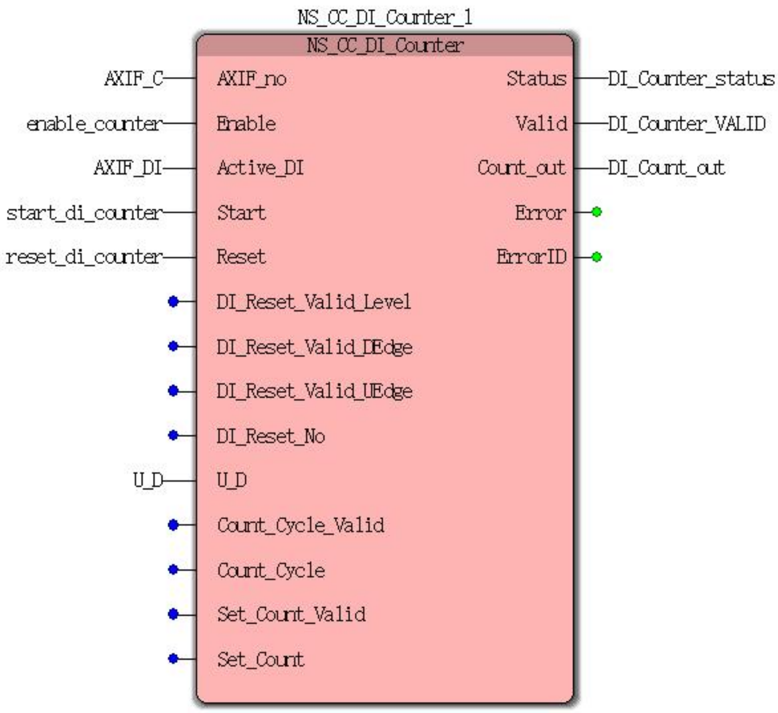
名称	功能	数据类型	输出范围
Status（状态位）	该输出参数为 TRUE 时表示指令正在执行中	BOOL	TRUE 或 FALSE
Valid（使能有效位）	该输出参数为 TRUE 时表示指令正在控制轴	BOOL	TRUE 或 FALSE
Count_out	当前脉冲计数值	DINT	正数、负数、0
Error	该指令输出为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE
ErrorID	执行指令出错时的错误代码	WORD	-

程序示范 I

变量启动和复位 DI 高速计数器

1、变量和程序

变量名称	数据类型	初始值
NS_CC_DI_Counter_1	NS_CC_DI_Counter	
AXIF_0	WORD	0
enbale_counter	BOOL	FALSE
AXIF_DI	WORD	11
start_di_counter	BOOL	FALSE
reset_di_counter	BOOL	FALSE
U_D	DINT	1
DI_Count_out	DINT	0



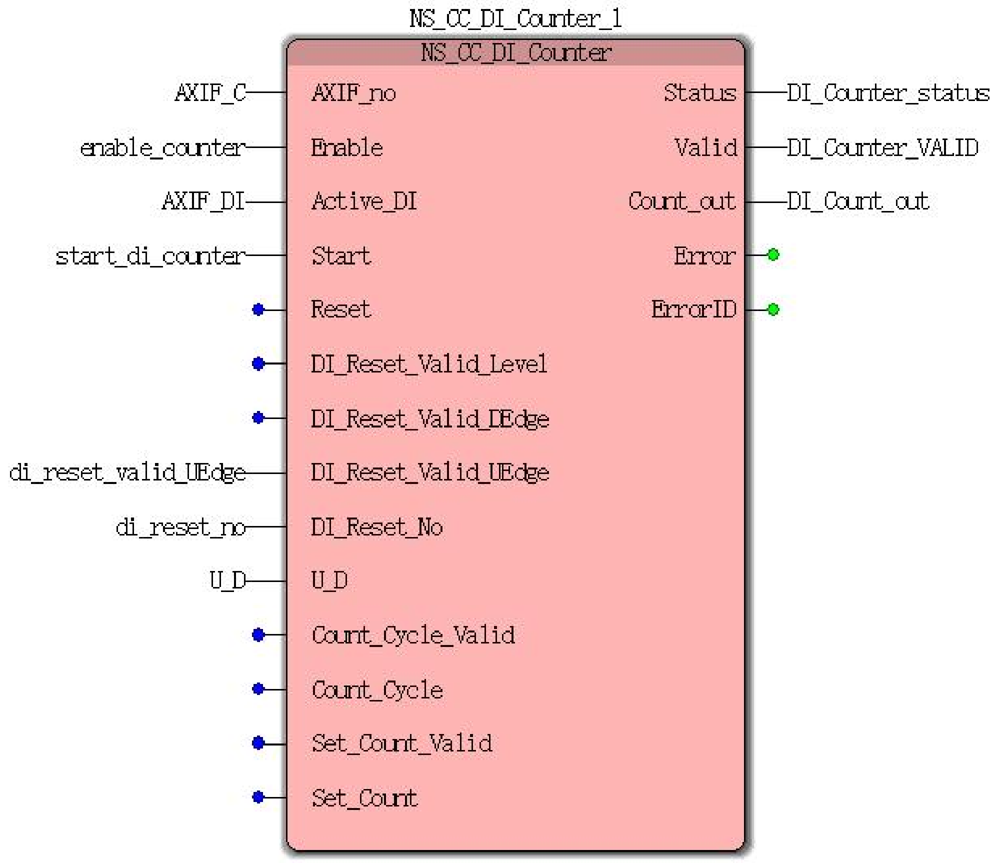
情形一：当 enable_counter 由 FALSE 变为 TRUE 时，且保持，start_di_counter 由 FALSE 变为 TRUE 开始允许 DI 高速计数器计数，DI_Count_out 显示当前脉冲计数值。当 reset_di_counter 由 FALSE 变为 TRUE 时，Count_out 显示当前脉冲计数值为 0，此时计数 DI 有脉冲输入，Count_out 显示当前脉冲计数值一直保持为 0，当 Reset 由 TRUE 变为 FALSE 时，继续开始允许计数。

程序示范 II

变量启动和外部 DI 复位执行 DI 高速计数器

1、变量和程序

变量名称	数据类型	初始值
NS_CC_DI_Counter_1	NS_CC_DI_Counter	
AXIF_0	WORD	0
enbale_counter	BOOL	FALSE
AXIF_DI	WORD	11
start_di_counter	BOOL	FALSE
di_reset_no	BOOL	0
U_D	DINT	1
DI_Count_out	DINT	0



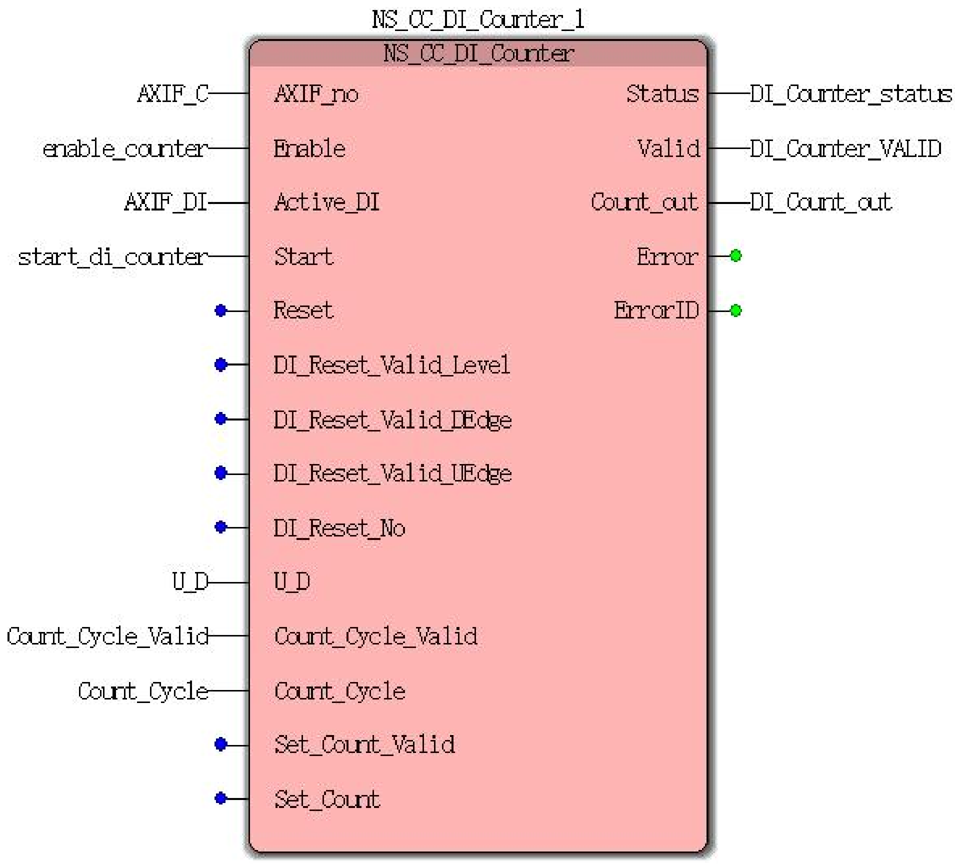
情形一：当 enable_counter 由 FALSE 变为 TRUE 时，且保持，start_di_counter 由 FALSE 变为 TRUE 开始允许 DI 高速计数器计数，DI_Count_out 显示当前脉冲计数值。当 di_reset_no 指定的 DI 出现上升沿时，Count_out 清零当前脉冲计数值。

程序示范III

循环计数模式执行 DI 高速计数器

1、变量和程序

变量名称	数据类型	初始值
NS_CC_DI_Counter_1	NS_CC_DI_Counter	
AXIF_0	WORD	0
enbale_counter	BOOL	FALSE
AXIF_DI	WORD	11
start_di_counter	BOOL	FALSE
U_D	DINT	1
Count_Cycle_Valid	BOOL	TRUE
Count_Cycle	DINT	1000
DI_Count_out	DINT	0



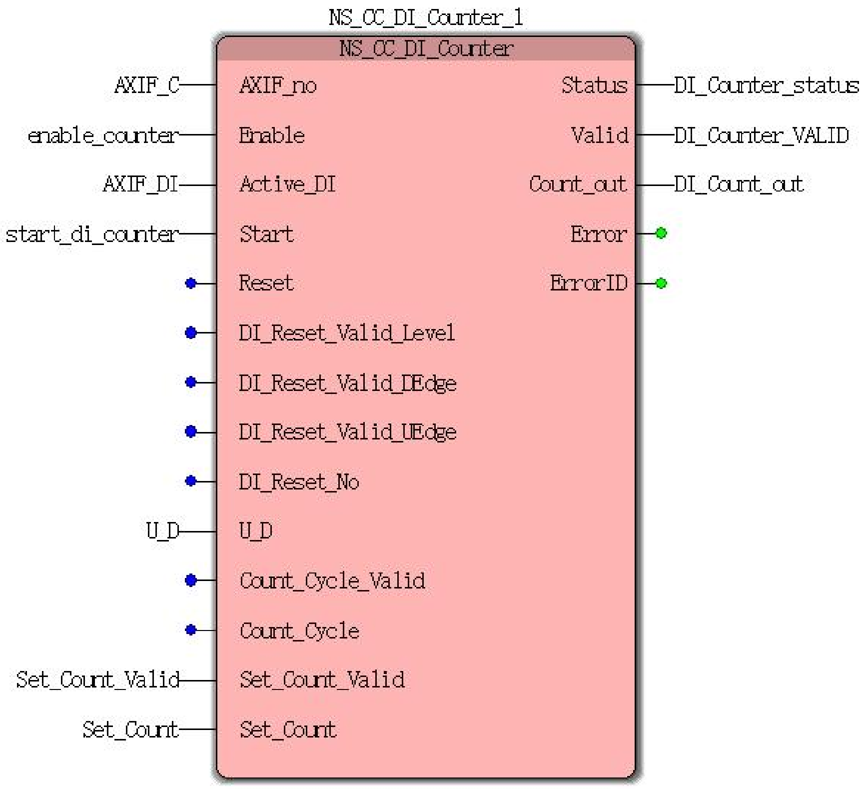
情形一：当 enable_counter 由 FALSE 变为 TRUE 时，且保持，start_di_counter 由 FALSE 变为 TRUE 开始允许 DI 高速计数器计数，DI11 来一个脉冲，DI_Count_out 显示当前脉冲计数值加 1，当 DI_Count_out 达到 1000 后自动清零重新计数。

程序示范Ⅳ

置数模式执行 DI 高速计数器

1、变量和程序

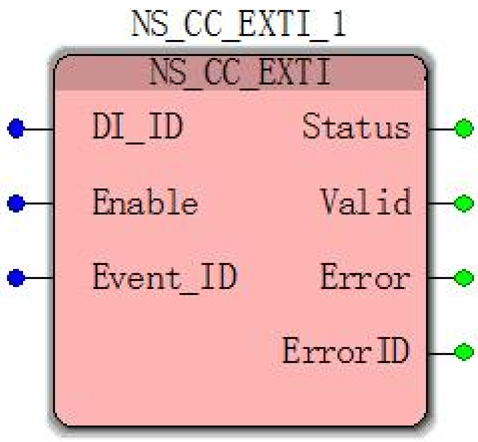
变量名称	数据类型	初始值
NS_CC_DI_Counter_1	NS_CC_DI_Counter	
AXIF_0	WORD	0
enbale_counter	BOOL	FALSE
AXIF_DI	WORD	11
start_di_counter	BOOL	FALSE
U_D	DINT	1
Set_Count_Valid	BOOL	TRUE
Set_Count	DINT	1000
DI_Count_out	DINT	0



情形一：当 counter_enable 由 FALSE 变为 TRUE 时，start_di_counter 由 FALSE 变为 TRUE 开始允许 DI 高速计数器计数，DI_Count_out 显示当前脉冲计数值。当 Set_Count_Valid 变量由 FALSE 变 TRUE 时，高速计数器 Count_out 当前值将被修改 1000，且一直保持 1000，当 Set_Count_Valid 变量由 TRUE 变 FALSE 时，当 DI11 收到脉冲时，计数脉冲数将在 1000 的基础上继续递增。

11.6.10 NS_CC_EXTI(DI 中断指令)

FB/FC	说明	适用机种
FB	指定 DI 有效时产生中断，关联事件程序将被执行一次	VEC-VA-MP-005-MA



- **输入参数**

名称	功能	数据类型	设定值范围（不可缺省值）	生效时机
DI_ID	指定 DI 有效时中断，输入值 0~7 对应输入点 I0~I7、8~15 对应输入点 I10~I17	WORD	目前仅支持 2 路中断 DI0 和 DI1 0~1 （不可缺省）	
Enable	当 Enable 由 FALSE 变为 TRUE 时，执行该指令（允许打开中断）。	BOOL	TRUE 或 FALSE	
Event_ID	设定中断事件号（0-5）与中断程序事件号一致。	WORD	0~5 （不可缺省）	

说明：

（1）目前仅支持两路 DI 中断，分别为外部端子 DI0 和 DI1

（2）NS_CC_EXTI 指令属于中断指令，当中断打开时，指定 DI0 或 DI1 有效的时候发生中断，程序将跳转到中断程序去执行，完成后返回到主程序继续循环运行。

- **输出参数**

名称	功能	数据类型	输出范围
Status（状态位）	该输出参数为 TRUE 时表示指令正在执行中	BOOL	FALSE 或 TRUE

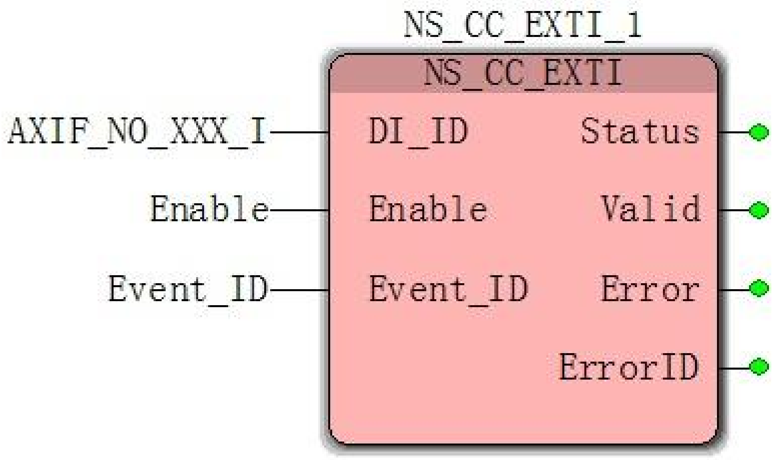
Valid（有效位）	该输出参数为 TRUE 时表示指令正在控制轴	BOOL	FALSE 或 TRUE
Error（误差）	保留	BOOL	FALSE 或 TRUE
ErrorID（错误代码）	保留	WORD	保留

程序范例

例：指定外部 DIO 中断设置

变量名称	数据类型	初始值
NS_CC_EXTI_1	NS_CC_EXTI	
AXIF_NO_XXX_I	WORD	0
Enable	BOOL	1
Event_ID	WORD	0

第一步.调用 NS_CC_EXTI 模块，编程如下



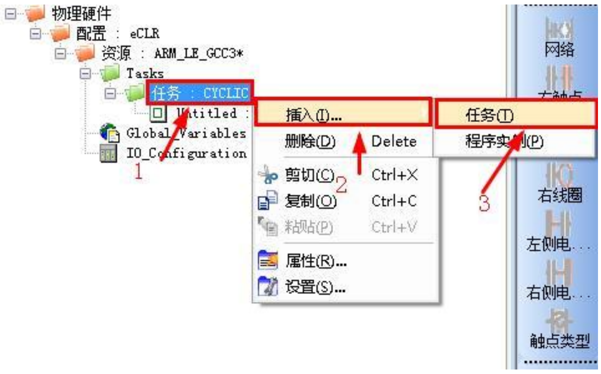
第二步.中断程序建立，在“工程树窗口”右键“逻辑 POU” 选择“插入”然后选择“程序”在弹出来的窗口给中断程序命名为“EN1”（用户可以自行命名），以及选择编写中断程序的语言“LD 语言”如下图所示。

第三步：完成后，单击确定，“逻辑 POU”下将会自动添加一个以“EN1”为名称程序如下图所示

第四步：刚才插入的“EN1”程序还是属于主程序的范畴，因为新建的“POU”程序软件自动加载到新建工程时选择的“任务”CYCLIC 类型下；如下图所示

第五步：在“工程树窗口”下选择 “硬件”，在资源“任务”下，右键删除“EN1”程序，单击“确定”我们需要另外新建“任务类型”，如下图所示

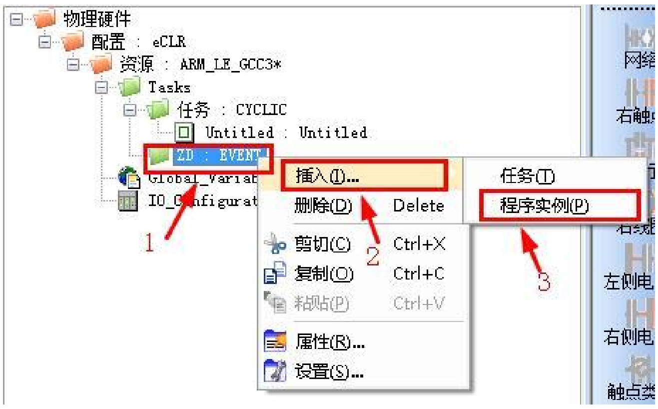
第六步： 新建“任务类型” 右键“任务”，单击“插入”，然后选择“任务”如下图所示



第七步：在弹出来的“插入”对话框在名称填写“ZD”（用户可以自行命名） 任务类型选择“EVENT”完成后单击“确定”。如下图所示

第八步：在弹出来的“ZD 的任务设置”选择事件号这里选择“事件 0”由中断模块上的“Event_ID”填写的数值决定，这里选择 0（程序示范的“Event_ID=0”）如图 10-17 所示（特别注意中断的优先权不能和主程序的优先级一样否者编译不通过或者是下载 97%报错误提示）

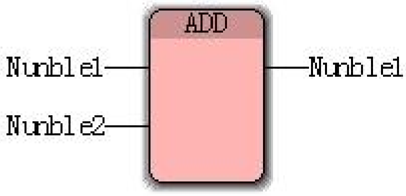
第九步：完后插入程序实例，右键任务下“EVENT”，选择“插入”，然后选择“程序实例”如下图所示



第十步：在弹出来的“插入框”，为程序实例命名为“ZD1”，程序类型选择新建的“EN1”单击“确定”最终的结果如下图所示

第十一步：返回“工程树窗口”下的“工程”窗口双击 POU 下的 EN1 加入一个加法模块，（目的是验证中断程序是否被正确执行）

变量名	数据类型	初始值
ADD	ADD	
Nunble1	INT	
Nunble2	INT	1

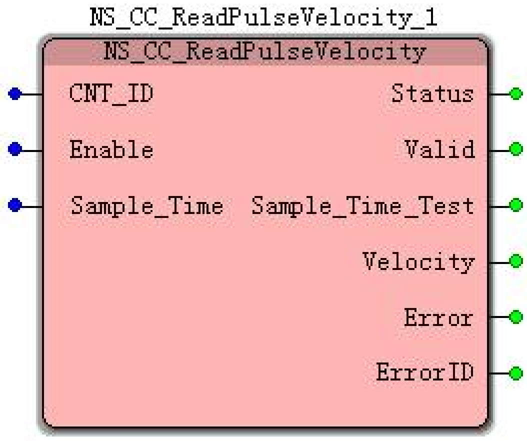


到此中断程序创建和关联已经完成，下面是分析中断程被执行的过程。

情形 1：当 NS_CC_EXTI 模块上 Enable 有 FALSE 变为 TRUE 时，允许中断打开，当外部 DI0 有效时候，:中断程序被执行一次，此时变量 Nunble1 值为 1， 完成后将跳出中断程序，返回主程序。当外部 DI 再次有效的时候，中断程序再次被执行，此时变量 Nunble1 值为 2。

11.6.11 NS_CC_ReadPulseVelocity（读取被控轴脉冲速率）

FB/FC	说明	适用机种
FB	此指令用于读取设定采样周期下的电机编码器脉冲数	VEC-VA-MP-005-MA



- **输入参数**

名称	功能	数据类型	设定值范围（不可缺省值）	生效时机
CNT_ID（轴号 ID）	指定欲采电机轴号速度（ 0-4 对应 AXIS0-AXIS4）	WORD	0-4（不可或缺）	Enable 由 FALSE 变 TRUE
Enable（执行位）	当 Execute 由 FALSE 变为 TRUE 时，指令开始执行	BOOL	TRUE 或 FALSE	-
Sample_Time（采样时间）	设定采样时间（单位：ms）该参数需与 MC_AXIS_REF(轴参数设定)的 Sample_Time 一致。	REAL	正数	实时生效

说明：

（1）该指令在 Enable 由 FALSE 变为 TRUE 时开始执行。

读取被控轴的速度值单位为 Pulse/s ,采样周期是 Sample_Time 参数，当前从轴速度计算为：假设：终端机构显示的转速为 n units/s，读取到的每秒脉冲数 Velocity=10000 Pulse/s ，电机每转脉冲数（moter_PPC）=10000 Pulse/rad，终端机构的 Screw_Lead 导程为 1000units，减速齿轮比 1:2

则：

$$n = \frac{Velocity}{motor\_PPC} * Screw\_Lead * \frac{1}{2} \text{ unit/s}$$

= 500 units/s

● **输出参数**

名称	功能	数据类型	输出范围
Status（状态）	保留	BOOL	TRUE 或 FALSE
Valid（有效）	保留	BOOL	TRUE 或 FALSE
Sample_Time_Test	保留	REAL	
Velocity（当前速度）	显示当前采样的速度（单位：Pulse/s）	REAL	实数
Error（保留）	保留	BOOL	保留
ErrorID（保留）	保留	WORD	保留

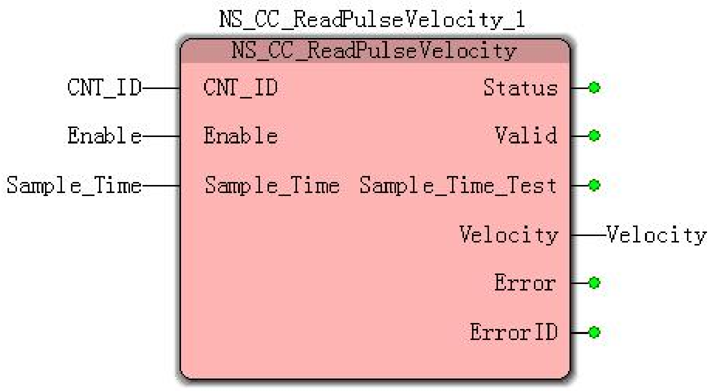
程序范例

例： 读取轴 AXIS0 当前速度；

1、变量和程序

变量名	数据类型	初始值
NS_CC_ReadPulseVelocity_1	NS_CC_ReadPulseVelocity	
CNT_ID	WORD	0
Enable	BOOL	1
Sample_Time	REAL	10
Velocity	REAL	

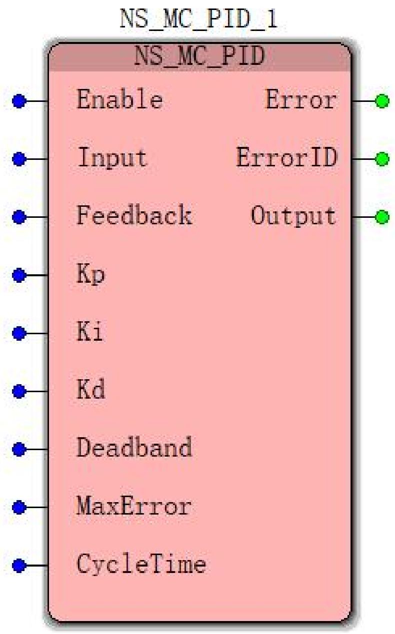
第一步：调用 NS_CC_ReadPulseVelocity_1 模块指令 编程如下



情形 1： 在伺服轴运行中，Enable 由 FALSE 变为 TRUE 时，模块 Velocity 将输出当前伺服轴的运行速度，如果想转换中断执行机构转速一致，需要按上述的公式进行转换。

11.6.12 MC_PID（PID 指令）

FB/FC	说明	适用机型
FB	该指令用于对指定的变量进行 PID 控制	VEC-VA-MP-005-MA



- **输入参数**

名称	功能	数据类型	设定值范围（不可缺省值）	生效时机
Enable（执行位）	当 Enable 由 FALSE 变为 TRUE 时，执行该指令。	BOOL	TRUE 或 FALSE	Enable 由 FALSE 变为 TRUE
Input（给定值）	设定给定值	LREAL	实数	-
Feedback（反馈值）	反馈值	LREAL	实数	-
Kp（比例系数）	比例控制	LREAL	实数	-
Ki（积分系数）	积分控制	LREAL	实数	-
Kd（微分系数）	微分控制	LREAL	实数	-
Deadband（PID 死区）	PID 死区 表示在该设定值下的值不进行 PID 运算；	LREAL	实数	-
MaxError（最大累积误差）	最大累积误差	LREAL	实数	-
CycleTime（PID 采样周期）	PID 采样周期单位（ms）	LREAL	实数	-

说明：

1）PID 控制原理

PID 调节是 Proportional（比例）、Integral（积分）、Differential（微分）三者的缩写，是连续系统中应用最为广泛的调节方式。PID 调节的实质是根据输入的偏差值，按比例、积分和微分的函数关系进行计算，其计算结果用以输出控制，以实现调节功能。

增量式 PID 调节算法

$$\Delta\mathrm{u(n)} = \mathrm{u(n)} - \mathrm{u(n-1)}$$

$$\Delta\mathrm{u(n)} = K_p * [e(n) - e(n-1)] + K_i * e(n) + K_d * [e(n) - 2 * e(n-1) + e(n-2)]$$

或者$\Delta\mathrm{u(n)} = P_{val} + I_{val} + D_{val}$

其中：

Kp：比例系数，现场习惯用比例 Kp=100；

Ki：积分系数，$K_i = K_p * T/T_i$；

Kd：微分系数，$K_d = K_p * T_d/T$；

Pval：比例作用， $P_{val} = K_p * [e(n) - e(n-1)]$；

Ival：积分作用， $I_{val} = K_i * e(n)$；

Dval：微分作用， $D_{val} = K_d * [e(n) - 2 * e(n-1) + e(n-2)]$；

式中 u(n-1)为 n-1 时刻的实际控制量，△u(n)为 n 时刻的控制量的增量，e(n)、e(n-1)和 e(n-2)分别是 n、n-1 和 n-2 时刻控制量与实际值的偏差，Ti、Td 和 T 分别是积分时间、微分时间和采样周期（CycleTime），其中偏差规定如下：偏差=设定值－测量值。

- **输出参数**

名称	功能	数据类型	输出范围
Output（输出）	PID 后输出结果	REAL	浮点数

程序示范

（略）

11.6.13 RTC_S（时钟特殊寄存器）

地址	说明	适用机型
特殊寄存器	用于修改和读取运动控制器内部时钟	VEC-VA-MP-005-MA

写时钟地址

特殊寄存器地址	写功能	数值范围
%MB3.9543	年	0~255
%MB3.9544	月	0~255
%MB3.9545	日	0~255
%MB3.9546	星期	0~255
%MB3.9547	时	0~255
%MB3.9548	分	0~255
%MB3.9549	秒	0~255
特别说明： 由于特殊寄存器地址为字节类型，最大值为 255 大于该值时寄存器将会溢出，如果年大于该值时需要进行取模运算。例如　今年为 2018 ，不能往%MB3.9543 地址填写 2018，需要将 2018%100（取模）后的值填入到年寄存器地址 也就是年地址中的值显示 18；		

读时钟地址

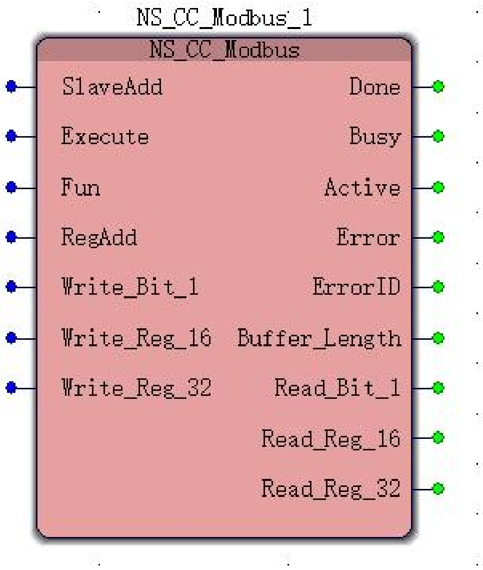
特殊寄存器地址	读功能	数值范围
%MB3.9550	年	0~255
%MB3.9551	月	0~255
%MB3.9552	日	0~255
%MB3.9553	星期	0~255
%MB3.9554	时	0~255
%MB3.9555	分	0~255
%MB3.9556	秒	0~255

执行修改时钟地址

特殊寄存器地址	功能	数值范围
%MB3.9542	数值为 1 的时候执行修改时钟（需要保持 1 大约 1s，时钟修改成功后需将该字节写为 0）	0~255

11.6.14 NS_CC_Modbus(VA 运动控制器作 Modbus 主站指令)

FB/FC	说明	适用机种
FB	此指令用于将 VA 控制器作为 Modbus 通讯主站控制从站运作	VEC-VA-MP-005-MA



➢ **输入参数**

名称	功能	数据类型	设定范围（缺省值）	生效时机
SlaveAdd（从站站号）	设定指令欲控制的从站（对应伺服 MODBUS 地址）	WORD	1~128 (不可缺省)	Execute 为 TRUE
Execute（上升沿触发位）	当 Execute 由 FALSE 变为 TRUE，指令开始执行。	BOOL	TRUE 或 FALSE	
Fun（读写参数的位长度选择）	写：为 1 写一个位； 为 2 写一个字； 为 3 写双字； 读：为 4 读一个位； 为 5 读一个字； 为 6 读双字。	WORD	1~6	Execute 为 TRUE
RegAdd	需要读写的从站参数的地址	WORD	0~65535	Execute 为 TRUE
Write_Bit_1	写入的位参数	BOOL	0，1	Execute 为 TRUE
Write_Reg_16	写入的 16 位参数	WORD	0，正数	Execute 为 TRUE
Write_Reg_32	写入的 32 位参数	DWORD	0，正数	Execute 为 TRUE

说明：

1、该指令在 Execute 由 FALSE 变为 TRUE 时开始执行。该指令正在执行，Execute 由 TRUE 变为 FALSE 时，对该指令执行无影响。

2、2、当指令正在执行中，Execute 再次由 FALSE 变为 TRUE 时，该指令可以重新执行，此时能够所有引脚参数重新生效。

3、Done 信号只会存在一个周期，数据输出也与 Done 信号相同只存在一个周期，所以读数据要外配逻辑锁存该数据。

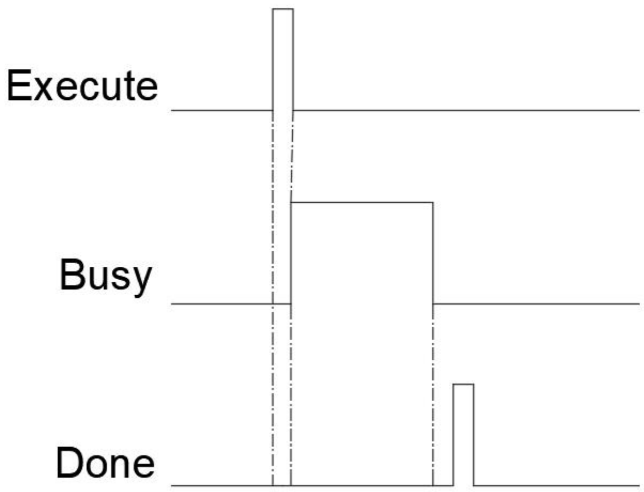
➢ **输出参数**

名称	功能	数据类型	输出范围
Done	该参数输出为 TRUE 时表示指令执行完成	BOOL	TRUE 或 FALSE
Busy（执行中）	该参数输出为 TRUE 时表示指令正在执行中	BOOL	TRUE 或 FALSE
Active（控制中）	该参数输出为 TRUE 时表示指令正在控制轴	BOOL	TRUE 或 FALSE
Error（错误）	该指令输出为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE
ErrorID（错误 ID）	该指令输出各参数情况： 0x2500：通讯成功； 0x2501：通讯错误； 0x2502：通讯错误； 0x2503：通讯错误； 0x2504：通讯错误； 0x2505：从机没回应； 0x2506：FUN 不支持该功能，只支持 1-6； 0x2500-0x2501：只有在 DONE 等于 1 的时候才会报错误 0x2510：触发 Execute，如果缓存溢出，则会报	UDINT	-
Buffer_Length（缓存深度）	可以占用的缓存值最多为 200，超过缓存深度上限后，通讯数据不能正常收发；	-	0~200
Read_Bit_1	读取到的位参数	BOOL	0，1
Read_Reg_16	读取到的 16 位参数	WORD	0，正数
Read_Reg_32	读取到的 32 位参数	DWORD	0，正数

注意：

（1）使用此功能时，需要将地址为%MB3.4024 的特殊寄存器初值设为 1，让 VA 运动控制器作主站（设 0 时作从站）；重上电后有效；
（2）使用此功能时，VA 运动控制器 MODBUS 通讯的波特率、停止位、校验位、数据位为默认参数，如伺服需要变动这些参数，则需要同步改动 VA 控制器的波特率（%MB3.4011），停止位（%MB3.4018），校验位（%MB3.4017），数据位（%MB3.4016）；此类寄存器相关详细说明请查阅“附件五 寄存器说明”。
（3）使用此功能时，需要在功能块的“Excute”前连接一个“R_TRIG（上升沿检测指令）”模块，以确保每次触发“Excute”都为上升沿触发。因为持续让“Excute”为 TRUE 会不断的占用此功能的“Buffer_Length（缓存深度）”，很有可能导致缓存溢出，导致参与通讯的数据无法正常收发。

- **输出参数时序变化图**

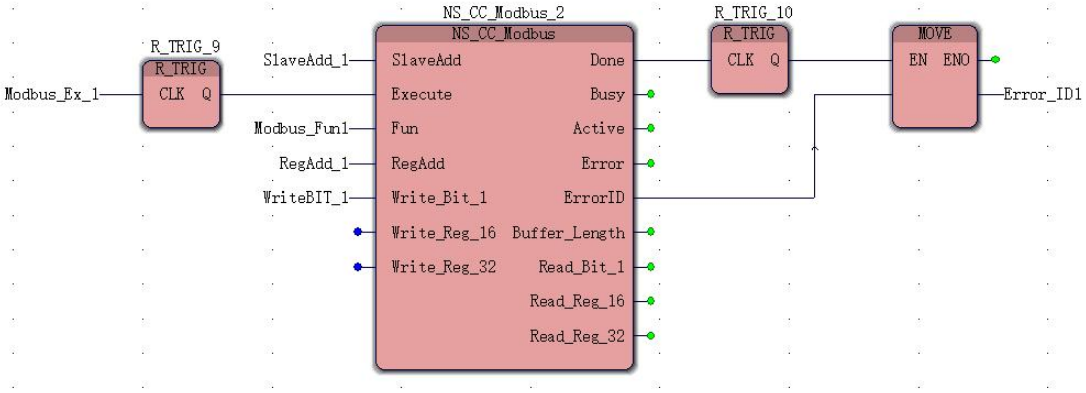


情形 1：Excute 由 FALSE 变为 TRUE 的一个周期后，Busy 变为 TRUE，当 Busy 由 TRUE 变 FALSE 一个周期后，Done 为 TRUE；

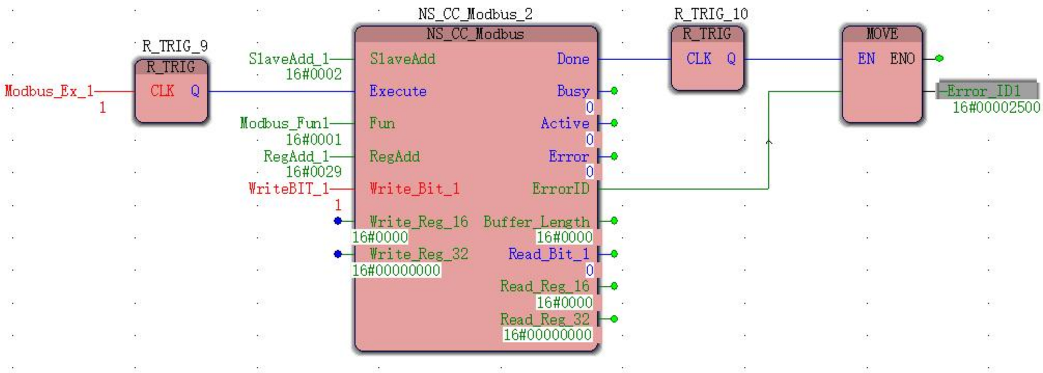
程序范例

写位参数：

变量名	数据类型	初始值
SlaveAdd_1	WORD	2
Modbus_Ex_1	BOOL	
Modbus_Fun1	WORD	1
RegAdd_1	WORD	41
WriteBIT_1	BOOL	1
Error_ID1	UDINT	

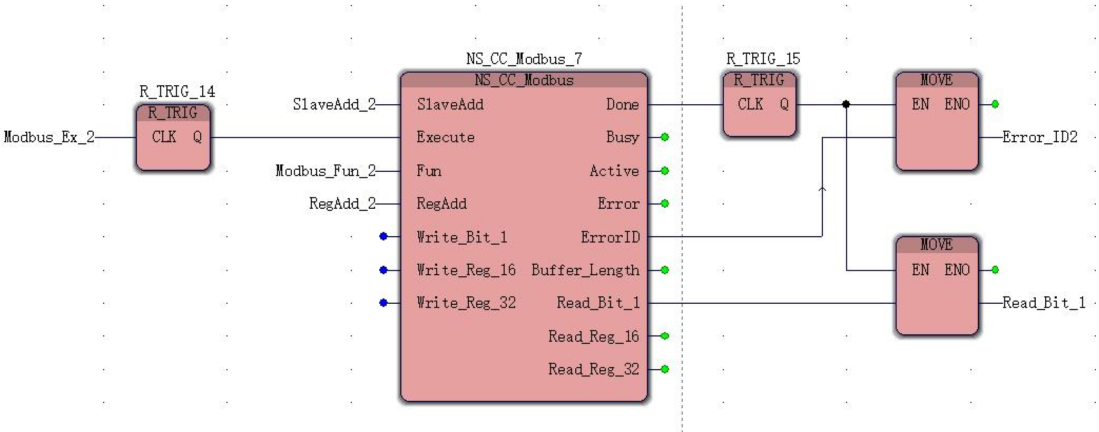


运行状态：

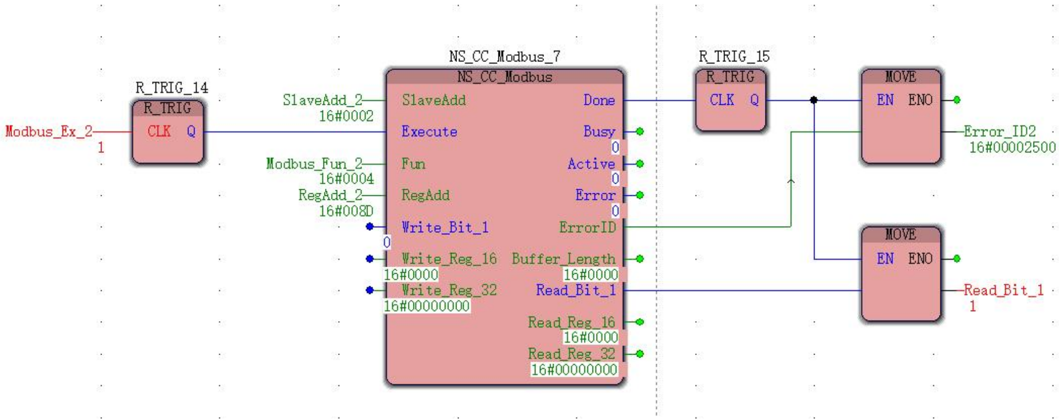


读位参数：

变量名	数据类型	初始值
SlaveAdd_2	WORD	2
Modbus_Ex_2	BOOL	
Modbus_Fun_2	WORD	2
RegAdd_2	WORD	141
Error_ID2	UDINT	
Read_Bit_1	BOOL	



运行状态：



由于 Done 信号与读出来的参数只保持一个周期，因此在读参数时需要按照以上示例的逻辑进行操作，即 Done 信号触发一次上升沿就把需要读的参数锁存出来。

11.7 G 代码指令

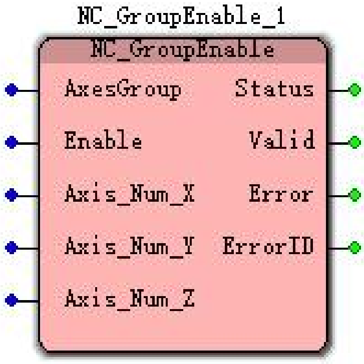
G 代码输入格式

我司 VA 运动控制器支持的 G 代码及输入格式如下表所示：

支持代码	功能描述	支持轴数
G0	快速定位	3 轴
G1	直线插补	3 轴
G2	顺时针方向圆弧插补	3 轴
G3	逆时针方向圆弧插补	3 轴
G4	定时暂停	
G17	加工 XY 平面	
G18	加工 XZ 平面	
G19	加工 YZ 平面	
G90	绝对尺寸	
G91	相对尺寸	
M0	程序停止	
M1	条件程序停止	
M2	程序结束	
M30	程序结束并返回程序头	

11.7.1 NC_GroupEnable(轴组使能指令)

FB/FC	说明	适用机种
FB	此指令用于将轴组使能	VEC-VA-MP-005-MA



➢ **输入参数**

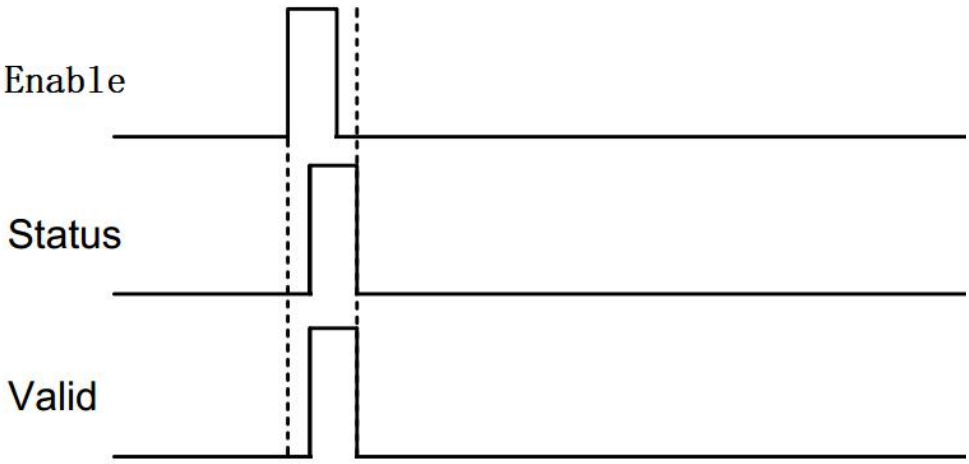
名称	功能	数据类型	设定范围（缺省值）	生效时机
AxesGroup（轴组编号）	设定欲使能的轴组	USINT	0	当 Enable 为 TRUE 时
Enable（使能位）	当 Enable 为 TRUE 时，执行该指令	BOOL	TRUE 或 FALSE	
Axis_Num_X（X 轴轴号）	设定 X 轴的轴号，必须设定为 0	USINT	0	当 Enable 为 TRUE 时
Axis_Num_Y（Y 轴轴号）	设定 Y 轴的轴号，必须设定为 1	USINT	1	当 Enable 为 TRUE 时
Axis_Num_Z (Z 轴轴号)	设定 Z 轴的轴号，必须设定为 2	USINT	2	当 Enable 为 TRUE 时

说明：轴组必须在使能后才能控制轴组作相应动作，当轴组未使能时，直线插补（NC_MoveLiner）、圆弧插补（NC_MoveCircula）、直角坐标机器人（NC_CartesianCoordinate）指令均不能执行。

➢ 输出参数

名称	功能	数据类型	输出范围
Status	该参数为 TRUE 时表示指令正在控制轴	BOOL	TRUE 或 FALSE
Valid	该输出参数为 TRUE 时表示指令输出有效	BOOL	TRUE 或 FALSE
Error	该参数为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE
ErrorID	指令执行出错时的错误代码	WORD	-

➢ **输出参数时序变化图**



➢ **功能说明**

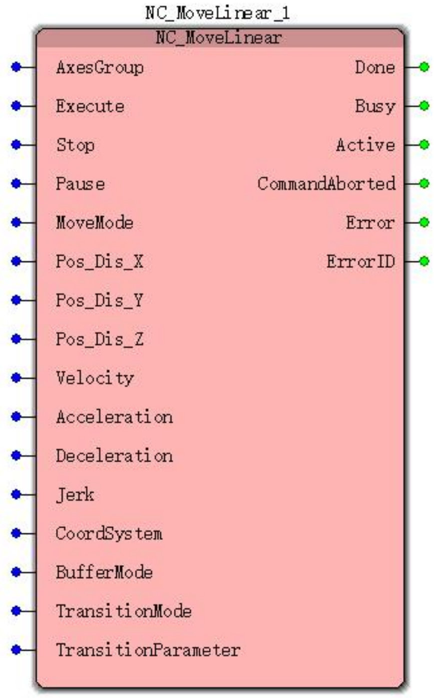
1、当 Enable 由 FALSE 变为 TRUE，延时一个周期后，Status、Valid 同时为 TRUE

2、当 Enable 由 TRUE 变为 FALSE，延时一个周期后，Status、Valid 同时为 FALSE

3、此指令用于使被控伺服轴组使能或者解除使能；

11.7.2 NC_MoveLiner（直线插补）

FB/FC	说明	适用机种
FB	此指令用于控制轴执行直线插补功能	VEC-VA-MP-005-MA



➢ **输入参数**

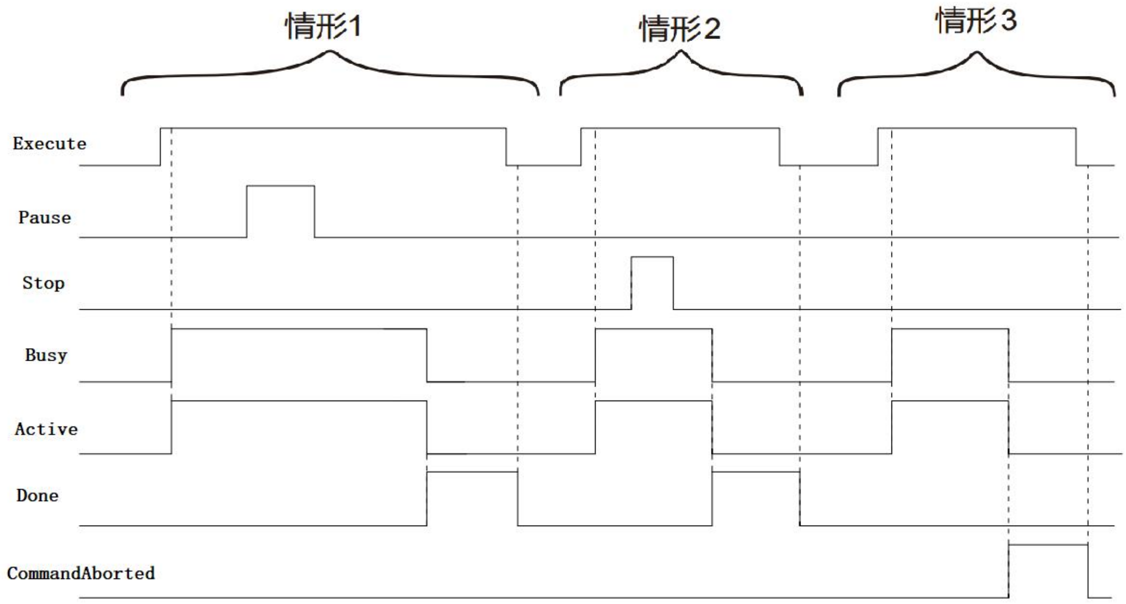
名称	功能	数据类型	设定范围（缺省值）	生效时机
AxesGroup（轴组编号）	设定欲使能的轴组	USINT	0	当 Execute 由 FALSE 变为 TRUE
Execute（执行位）	当 Execute 由 FALSE 变为 TRUE 时，执行指令	BOOL	TRUE 或 FALSE	-
Stop（停止位）	当 Stop 由 FALSE 变为 TRUE 时，停止该指令。	BOOL	TRUE 或 FALSE	-
Pause（暂停位）	当 Pause 为 TRUE 时，暂停该指令的执行	BOOL	TRUE 或 FALSE	-
MoveMode（移动模式）	当 MoveMode 为 TRUE 时，X/Y/Z 轴的目标位置为绝对位置 FALSE 时，X/Y/Z 轴的目标位置为相对位置	BOOL	TRUE 或 FALSE	当 Execute 由 FALSE 变为 TRUE
Pos_Dis_X（X 轴目标位置）	设定 X 轴目标位置 单位：单元	LREAL	正数、负数、0	当 Execute 由 FALSE 变

				为 TRUE
Pos_Dis_Y （Y 轴目标位置）	设定 Y 轴目标位置 单位：单元	LREAL	正数、负数、0	当 Execute 由 FALSE 变为 TRUE
Pos_Dis_Z （Z 轴目标位置）	设定 Z 轴目标位置 单位：单元	LREAL	正数、负数、0	当 Execute 由 FALSE 变为 TRUE
Velocity （速度）	设定三个轴合成速度的最大值 单位：单元/min	LREAL	正数	当 Execute 由 FALSE 变为 TRUE
Acceleration （加速度）	设定三个轴合成加速度的最大值 单位：单元/min^2	LREAL	正数	当 Execute 由 FALSE 变为 TRUE
Deceleration	保留	保留	保留	保留
Jerk	保留	保留	保留	保留
CoordSystem	保留	保留	保留	保留
BufferMode	指令交接模式，只能设置为 1	INT		
TransitionMode	保留	保留	保留	保留
TransitionParameter	保留	保留	保留	保留

➢ **输出参数**

名称	功能	数据类型	输出范围
Done	该参数输出为 TRUE 时表示指令执行完成	BOOL	TRUE 或 FALSE
Busy（执行中）	该参数输出为 TRUE 时表示指令正在执行中	BOOL	TRUE 或 FALSE
Active（控制中）	该参数输出为 TRUE 时表示指令正在控制轴	BOOL	TRUE 或 FALSE
CommandAborted（中断）	该参数输出为 TRUE 时表示指令被中断	BOOL	TRUE 或 FALSE
Error（错误）	该指令输出为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE
ErrorID（错误代码）	执行指令出错时的错误代码	WORD	-

➢ **输出参数时序变化图**



情形 1： 当 Execute 由 FALSE 变为 TRUE，一个周期后，Busy 和 Active 同时变为 TRUE，指令执行期间，Pause 由 FALSE 变为 TRUE，指令执行的动作暂停，但 Busy 和 Active 仍然为 TRUE，当 Pause 由 TRUE 变为 FALSE 后，被控轴继续未完成的动作。当指令执行完成时，Busy 和 Active 变为 FALSE，同时 Done 变为 TRUE。在 Execute 由 TRUE 变为 FALSE 后一个周期，Done 变为 FALSE。

情形 2： 当 Execute 由 FALSE 变为 TRUE，一个周期后，Busy 和 Active 同时变为 TRUE，指令执行期间，Stop 由 FALSE 变为 TRUE，结束指令的执行，但 Busy 和 Active 仍然为 TRUE 直到被控轴停止，Busy 和 Active 变为 FALSE，同时 Done 变为 TRUE。在 Execute 由 TRUE 变为 FALSE 后一个周期，Done 变为 FALSE。

情形 3： 当 Execute 由 FALSE 变为 TRUE，该指令被其它指令中断时，CommandAborted 变为 TRUE，同时 Busy 和 Active 变为 FALSE；当 Execute 由 TRUE 变为 FALSE 时 CommandAborted 变为 FALSE。

■ **功能说明**

此指令用于轴组进行直线插补，可以控制轴组中的一个轴或多个轴。

1、NC_MoveLiner 指令的参数 Velocity 为终端机构的目标速度，终端机构速度与各轴速度的关系如下：终端机构速度的平方=各轴速度的平方和。该指令参数 Acceleration、Deceleration 为终端机构的目标加速度和目标减速度，终端机构的加速度和减速度与各轴加、减速度的关系为：终端机构的加（减）速度=各轴加（减）速度的平方和。

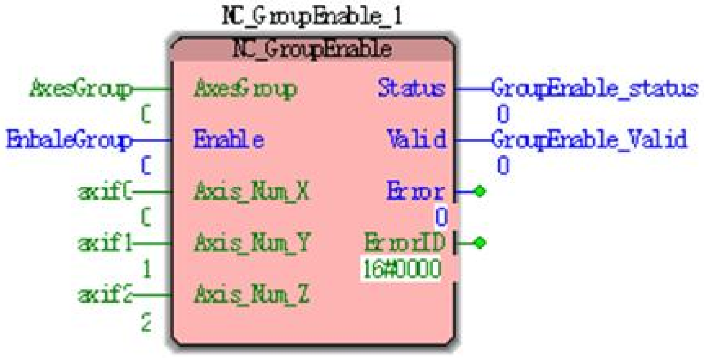
程序范例一

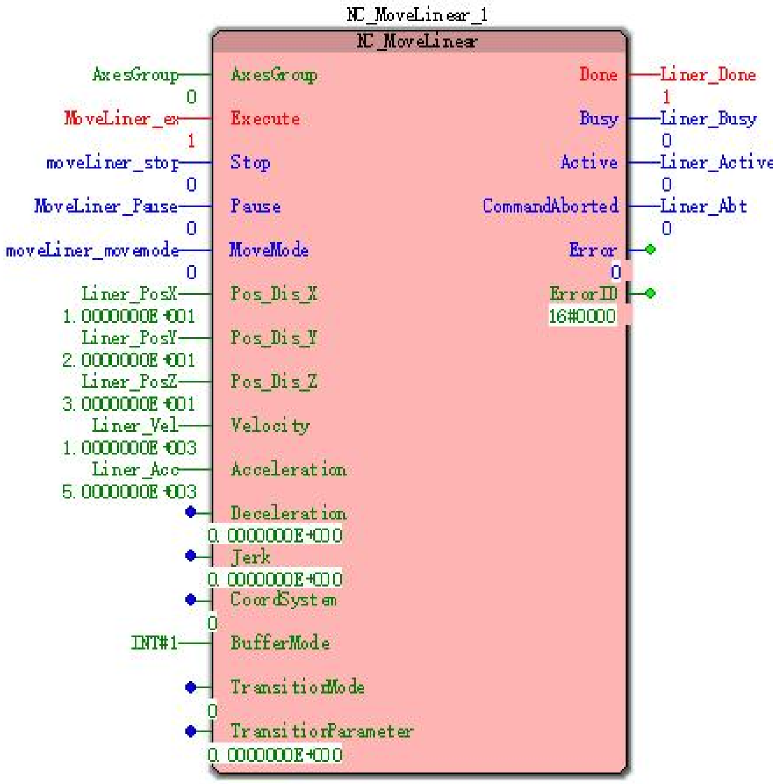
相对模式执行 NC_MoveLiner

1、变量和程序

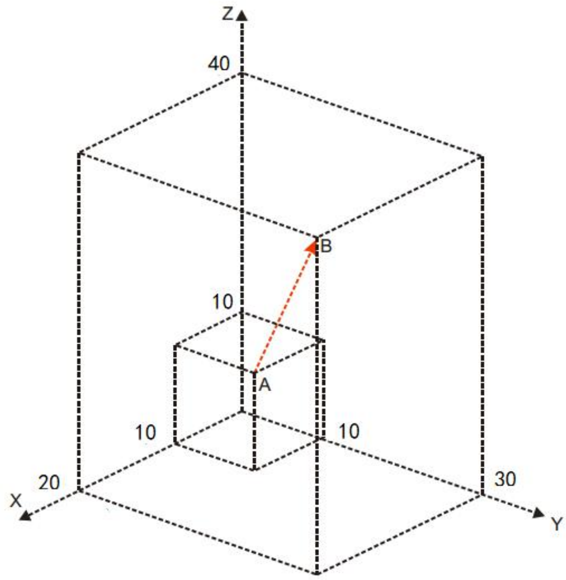
变量名	数据类型	初始值
NC_MoveLiner_1	NC_MoveLiner	-
AxesGroup	USINT	0
MoveLiner_ex	BOOL	

MoveLiner_stop	BOOL	
MoveLiner_Pause	BOOL	
MoveLiner_Movemode	BOOL	
Liner_PosX	LREAL	10.0
Liner_PosY	LREAL	20.0
Liner_Vel	LREAL	30.0
Liner_Acc	LREAL	1000.0
Liner_Done	BOOL	5000.0
Liner_Busy	BOOL	
Liner_Active	BOOL	
Liner_Abt	BOOL	



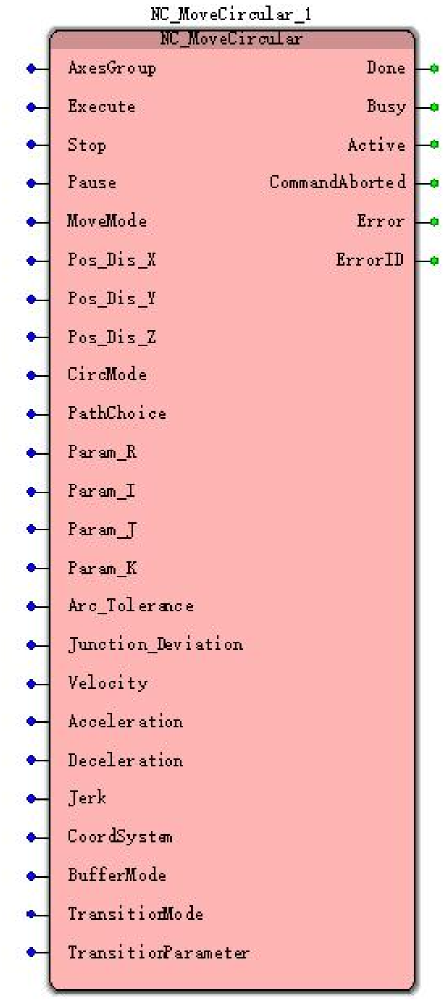


2、指令执行后，整个运动过程如下图所示：



11.7.3 NC_MoveCircula（圆弧插补）

FB/FC	说明	适用机种
FB	此指令用于控制轴执行直线插补功能	VEC-VA-MP-005-MA



➢ **输入参数**

名称	功能	数据类型	设定范围（缺省值）	生效时机
AxesGroup（轴组编号）	设定欲使能的轴组	USINT	0	当 Execute 由 FALSE 变为 TRUE
Execute（执行位）	当 Execute 由 FALSE 变为 TRUE 时，执行指令	BOOL	TRUE 或 FALSE	-
Stop（停止位）	当 Stop 由 FALSE 变为 TRUE 时，停止该指令。	BOOL	TRUE 或 FALSE	-
Pause（暂停位）	当 Pause 为 TRUE 时，暂停该指令的执行	BOOL	TRUE 或 FALSE	-
MoveMode（移动模式）	当 MoveMode 为 TRUE 时，X/Y/Z 轴的目标位	BOOL	TRUE 或 FALSE	当 Execute 由 FALSE 变

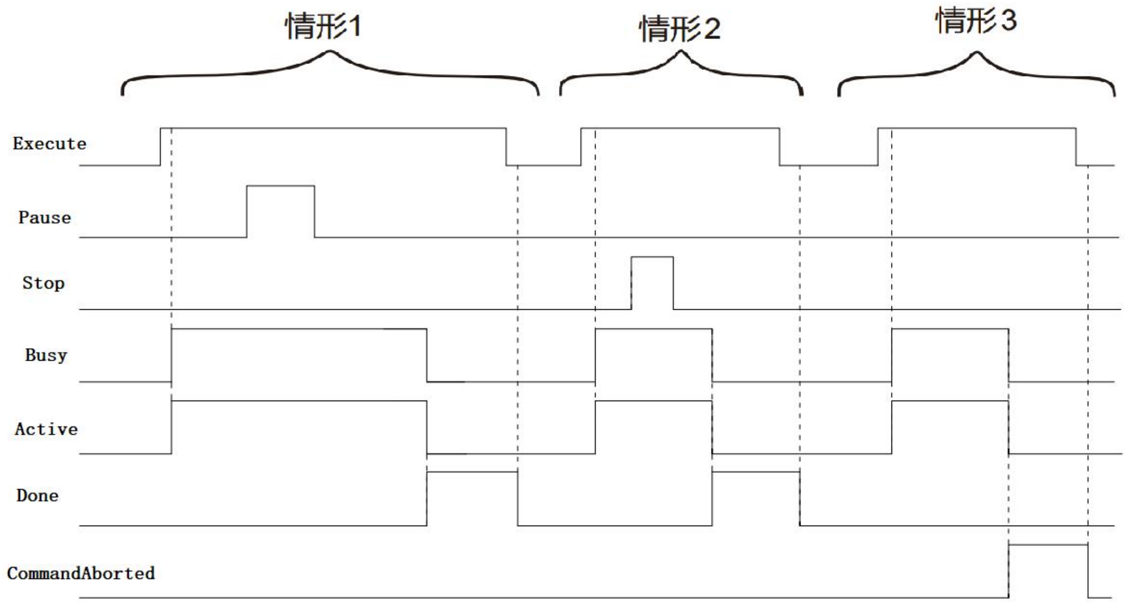
	置为绝对位置 FALSE 时，X/Y/Z 轴的目标位置为相对位置			为 TRUE
Pos_Dis_X （X 轴目标位置）	设定 X 轴目标位置 单位：单元	LREAL	正数、负数、0	当 Execute 由 FALSE 变为 TRUE
Pos_Dis_Y （Y 轴目标位置）	设定 Y 轴目标位置 单位：单元	LREAL	正数、负数、0	当 Execute 由 FALSE 变为 TRUE
Pos_Dis_Z （Z 轴目标位置）	设定 Z 轴目标位置 单位：单元	LREAL	正数、负数、0	当 Execute 由 FALSE 变为 TRUE
CircMode （圆弧插补方式）	设定圆弧插补方式 0：XY 平面画圆 1：ZX 平面画圆 2：YZ 平面画圆	INT	0~2	
PathChoice （圆弧方向）	圆弧插补的方向 0：顺时针方向 1：逆时针方向	INT	0、1	
Param_R (半径)	设定平面半径法画圆的半径并且当 半径为负值时选择优弧； 半径为正值时选择劣弧； 半径为 0 时选择圆心法画圆	LREAL	正数、负数、0 （0）	
Param_I （X 轴圆心偏移）	设定圆心法画圆时，圆心在 X 轴当前位置的偏移量	LREAL	正数、负数、0 (0)	
Param_J （Y 轴圆心偏移）	设定圆心法画圆时，圆心在 Y 轴当前位置的偏移量	LREAL	正数、负数、0 (0)	
Param_K （Z 轴圆心偏移）	设定圆心法画圆时，圆心在 Z 轴当前位置的偏移量	LREAL	正数、负数、0 (0)	
Arc_Tolerance （弦弧公差）	设定插补过程中允许的最大弦弧公差。 弦弧公差=每段插补弧长-每段插补弦长	LREAL		
Junction_Deviation （转角偏差）	设定每段圆弧插补最大转角偏差	LREAL		
Velocity （速度）	设定三个轴合成速度的最大值	LREAL	正数	当 Execute 由 FALSE 变

	单位：单元/min			为 TRUE
Acceleration（加速度）	设定三个轴合成加速度的最大值 单位：单元/min²	LREAL	正数	当 Execute 由 FALSE 变为 TRUE
Deceleration	保留	保留	保留	保留
Jerk	保留	保留	保留	保留
CoordSystem	保留	保留	保留	保留
BufferMode	指令交接模式，只能设置为 1	INT		
TransitionMode	保留	保留	保留	保留
TransitionParameter	保留	保留	保留	保留

➢ **输出参数**

名称	功能	数据类型	输出范围
Done	该参数输出为 TRUE 时表示指令执行完成	BOOL	TRUE 或 FALSE
Busy（执行中）	该参数输出为 TRUE 时表示指令正在执行中	BOOL	TRUE 或 FALSE
Active（控制中）	该参数输出为 TRUE 时表示指令正在控制轴	BOOL	TRUE 或 FALSE
CommandAborted（中断）	该参数输出为 TRUE 时表示指令被中断	BOOL	TRUE 或 FALSE
Error（错误）	该指令输出为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE
ErrorID（错误代码）	执行指令出错时的错误代码	WORD	-

➢ **输出参数时序变化图**



情形 1：当 Execute 由 FALSE 变为 TRUE，一个周期后，Busy 和 Active 同时变为 TRUE，指令执行期间，Pause 由 FALSE 变为 TRUE，指令执行的动作暂停，但 Busy 和 Active 仍然为 TRUE，当 Pause 由 TRUE 变为 FALSE 后，被控轴继续未完成的动作。当指令执行完成时，Busy 和 Active 变为 FALSE，同时 Done 变为 TRUE。在 Execute 由 TRUE 变为 FALSE 后一个周期，Done 变为 FALSE。

情形 2：当 Execute 由 FALSE 变为 TRUE，一个周期后，Busy 和 Active 同时变为 TRUE，指令执行期间，Stop 由 FALSE 变为 TRUE，结束指令的执行，但 Busy 和 Active 仍然为 TRUE 直到被控轴停止，Busy 和 Active 变为 FALSE，同时 Done 变为 TRUE。在 Execute 由 TRUE 变为 FALSE 后一个周期，Done 变为 FALSE。

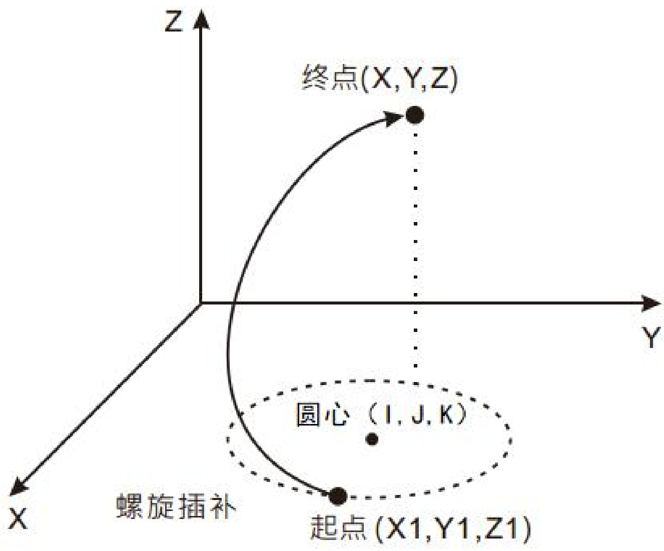
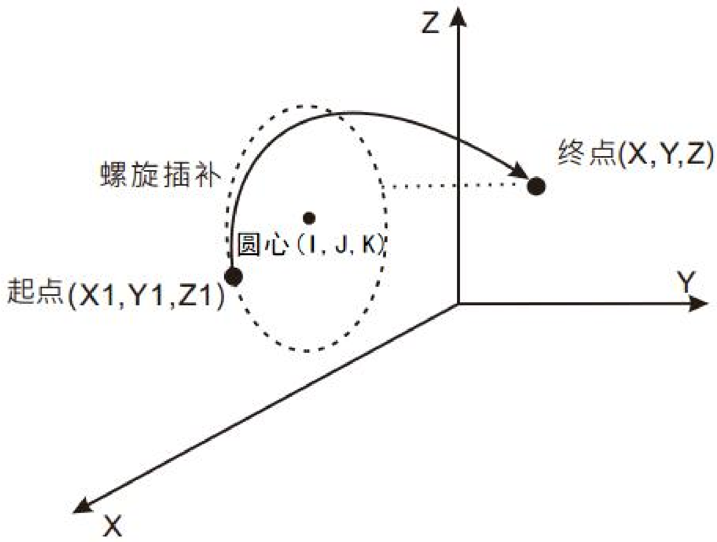
情形 3：当 Execute 由 FALSE 变为 TRUE，该指令被其它指令中断时，CommandAborted 变为 TRUE，同时 Busy 和 Active 变为 FALSE；当 Execute 由 TRUE 变为 FALSE 时 CommandAborted 变为 FALSE。

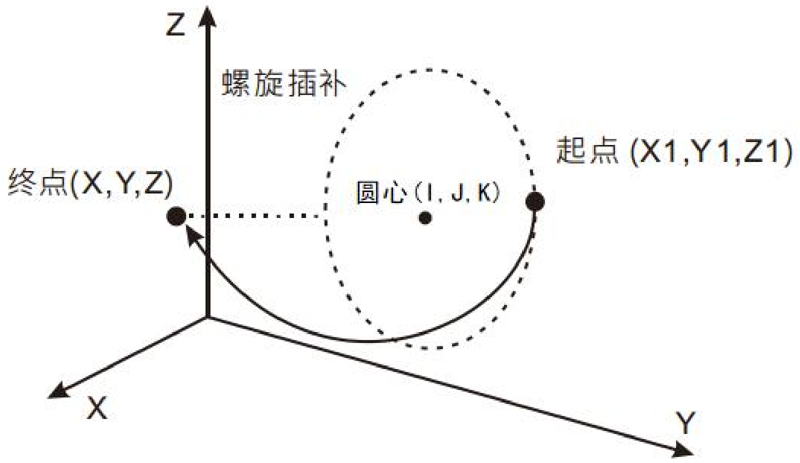
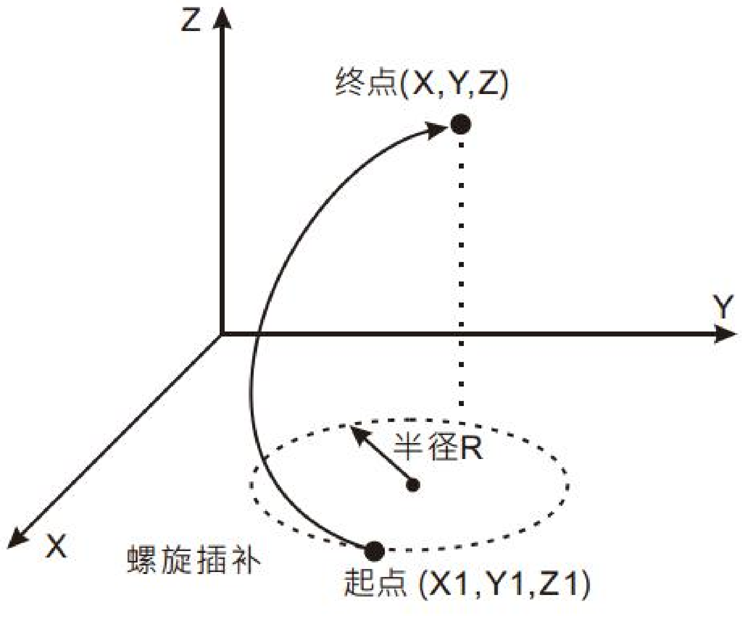
功能说明：

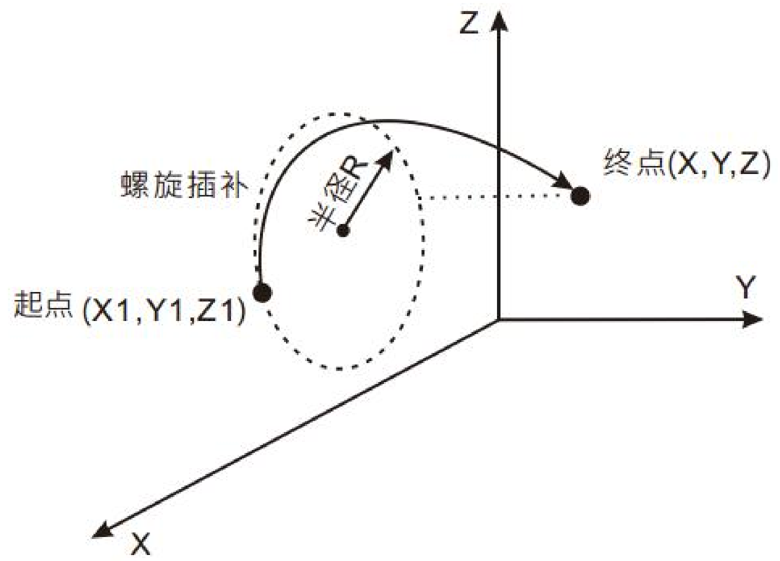
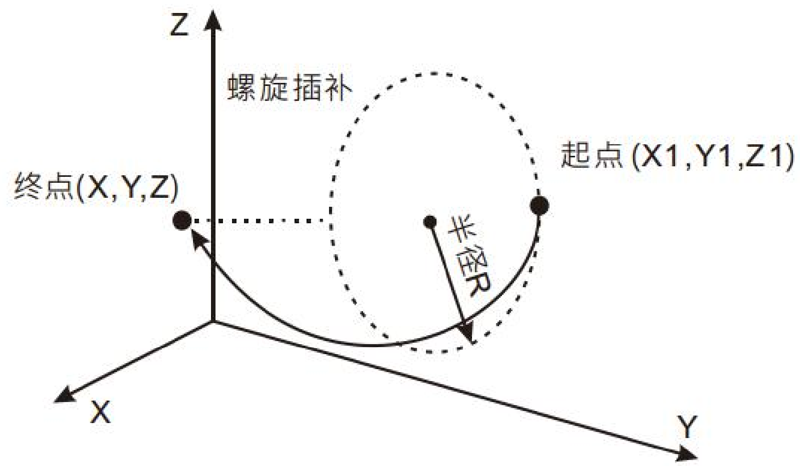
此指令用于轴组进行圆弧插补

■ **CirMode、Param_R、Param_I、Param_J、Param_K 共同决定圆弧插补模式**

组合方式	说明
CirMode=0，Param_R=0 Param_I= A，A≠0 Param_J=B，B≠0 Param_K=0	XY 平面圆心法画圆。使用该方法时，Param_I 代表圆心位置相对于起点位置的 X 轴偏移量，Param_J 代表圆心位置相对于起点位置的 Y 轴偏移量。

	 上图中，圆心坐标 I=X1+ Param_I，J= Y1+ Param_J，K=Z1 终点坐标 X=Pos_Dis_X，Y= Pos_Dis_Y，Z= Pos_Dis_Z。
CirMode=1，Param_R=0 Param_I= A，A≠0 Param_K=B，B≠0 Param_J=0	ZX 平面圆心法画圆。使用该方法时，Param_I 代表圆心位置相对于起点位置的 X 轴偏移量，Param_K 代表圆心位置相对于起点位置的 Z 轴偏移量。  上图中，圆心坐标 I=X1+ Param_I，J=Y1，K= Z1+ Param_K， 终点坐标 X=Pos_Dis_X，Y= Pos_Dis_Y，Z= Pos_Dis_Z。
CirMode=2，Param_R=0 Param_J= A，A≠0 Param_K=B，B≠0 Param_I=0	YZ 平面圆心法画圆。使用该方法时，Param_J 代表圆心位置相对于起点位置的 Y 轴偏移量，Param_K 代表圆心位置相对于起点位置的 Z 轴偏移量。

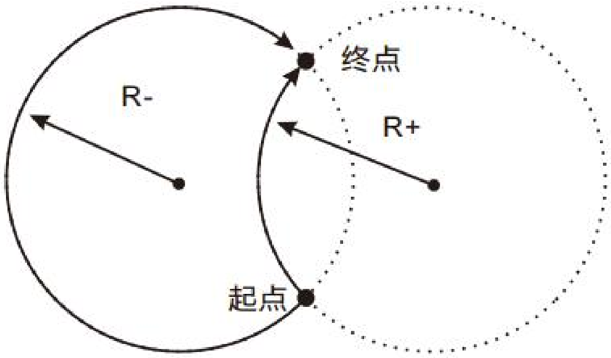
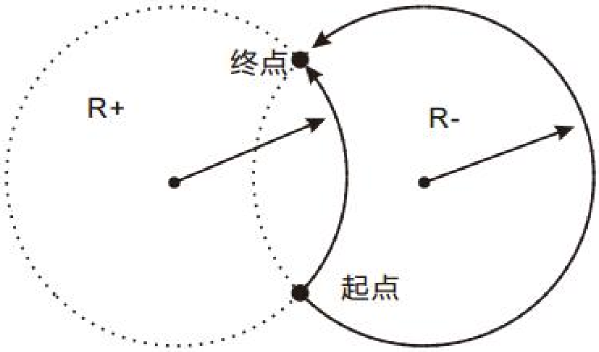
	 上图中，圆心坐,I=X1，J=Y1+ Param_J，K= Z1+ Param_K 终点坐标 X=Pos_Dis_X，Y= Pos_Dis_Y，Z= Pos_Dis_Z。
CirMode=0，Param_R=R，R≠0	XY 平面半径法画圆。使用该方法时，Param_R 的值表示 XY 平面上圆的半径，此时 Param_R 大于 0，圆弧为劣弧；Param_R 小于 0，圆弧为优弧  上图中圆的半径为 R。 终点坐标 X=Pos_Dis_X，Y= Pos_Dis_Y，Z= Pos_Dis_Z。
CirMode=1，Param_R=R，R≠0	ZX 平面半径法画圆。使用该方法时，Param_R 的值表示 ZX 平面上圆的半径，此时 Param_R 大于 0，圆弧为劣弧；Param_R 小于 0，圆弧为优弧

	 上图中圆的半径为 R。 终点坐标 X=Pos_Dis_X，Y= Pos_Dis_Y，Z= Pos_Dis_Z。
CirMode=2，Param_R=R，R≠0	YZ 平面半径法画圆。使用该方法时，Param_R 的值表示 YZ 平面上圆的半径，此时 Param_R 大于 0，圆弧为劣弧；Param_R 小于 0，圆弧为优弧  上图中圆的半径为 R。 终点坐标 X=Pos_Dis_X，Y= Pos_Dis_Y，Z= Pos_Dis_Z。

- **PathChoice**

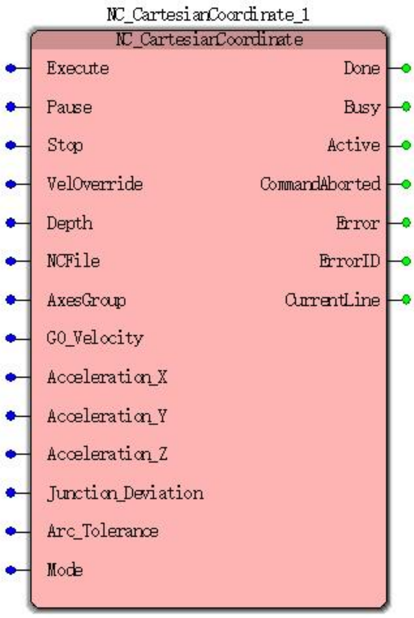
该参数决定圆弧插补的方向

参数值	说明
0	轴组在指定的平面及圆弧上进行顺时针圆弧插补。以半径法为例，圆弧运行轨迹如下图所示：

	 顺时针运转
1	轴组在指定的平面及圆弧上进行逆时针圆弧插补。以半径法为例，圆弧运行轨迹如下图所示：  逆时针运转

11.7.4 NC_CartesianCoordinate(直角坐标机器人指令)

FB/FC	说明	适用机种
FB	此指令用于控制直角坐标轴按照 G 代码进行插补运动	VEC-VA-MP-005-MA



➢ **输入参数**

名称	功能	数据类型	设定范围（缺省值）	生效时机
Execute（执行位）	当 Execute 由 FALSE 变为 TRUE 时，执行指令	BOOL	TRUE 或 FALSE	当 Execute 由 FALSE 变为 TRUE-
Pause（暂停位）	当 Pause 为 TRUE 时，暂停该指令的执行	BOOL	TRUE 或 FALSE	-
Stop（停止位）	当 Stop 由 FALSE 变为 TRUE 时，停止该指令。	BOOL	TRUE 或 FALSE	-
VelOverride（速度超调值）	速度超调值（%）	LREAL	1≤ VelOverride ≤500	
Depth（缓存深度）	缓存深度（设 2 表示 1，设 16 表示 15）。正常设置 16，如果每个小线段都想启停速度为 0 则设 2	UINT		
NCFile（NC 文件）	NC 文件选择	UINT		
AxesGroup（轴组）	轴组组号，必须设置为 0	USINT	0	
G0_Velocity	设置 G0 指令最大速度	LREAL		

(G0 速度)	单位：单元/min			
Acceleration_X （X 轴加速度）	设置 X 轴最大加速度 单位：单元/min^2	LREAL		
Acceleration_Y （Y 轴加速度）	设置 Y 轴最大加速度 单位：单元/min^2	LREAL		
Acceleration_Z （Z 轴加速度）	设置 Z 轴最大加速度 单位：单元/min^2	LREAL		
Junction_Deviation （转角偏差）	设定每段圆弧插补最大转角偏差	LREAL		
Arc_Tolerance （弦弧公差）	设定插补过程中允许的最大弦弧公差。 弦弧公差=每段插补弧长-每段插补弦长	LREAL		
Mode （末端处理模式）	当该值为 0 时，对每个插补小线段的末端不处理。 当该值为 1 时，对每个插补小线段的末端进行处理	INT	0、1	

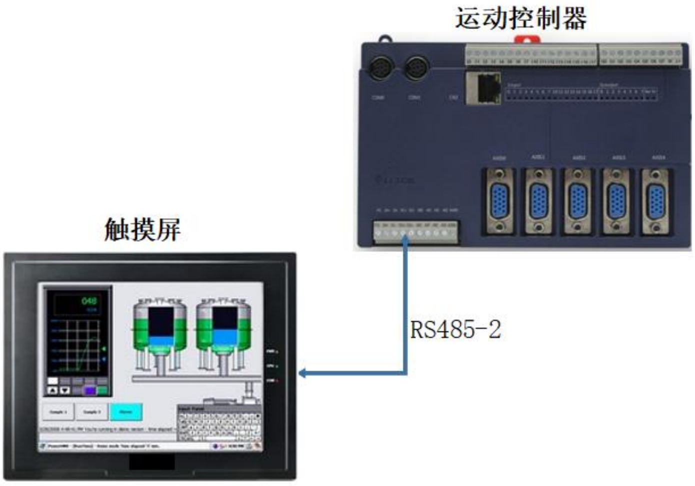
➢ **输出参数**

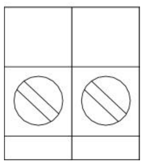
名称	功能	数据类型	输出范围
Done	该参数输出为 TRUE 时表示指令执行完成	BOOL	TRUE 或 FALSE
Busy（执行中）	该参数输出为 TRUE 时表示指令正在执行中	BOOL	TRUE 或 FALSE
Active（控制中）	该参数输出为 TRUE 时表示指令正在控制轴	BOOL	TRUE 或 FALSE
CommandAborted（中断）	该参数输出为 TRUE 时表示指令被中断	BOOL	TRUE 或 FALSE
Error（错误）	该指令输出为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE
ErrorID（错误代码）	执行指令出错时的错误代码	WORD	-
CurrentLine	当前正在执行的 G 代码所在行数	UDINT	

第十二章 功能设置

12.1 运动控制器与 HMI 通讯

12.1.1 运动控制器与人机接线方式 如下图所示



		端子	定义
RS-485		SG+	RS-485 信号正
		SG-	RS-485 信号负

12.1.2 人机与运动控制器通讯格式

功能名称	格式	出厂设置	
PLC 通讯协议	MODBUS RTU	MODBUS RTU	MODBUS RTU
通讯接口类型	RS 485-2	RS 485-2	RS 232
波特率	9600	9600	9600
数据位	8	8	8
奇偶校验	偶校验	偶校验	偶校验
停止位	1	1	1
站号	111	111	111

运行控制器与外设通讯时默认出厂设置，用户可通过特殊寄存器来修改站号和波特率；
特殊寄存器：%MB3.4010（站号）； %MB3.4011（波特率）；
特别说明： 波特率=输入值* 4800 （例如：波特率为 9600 则%MB3.4011 填 2 ；波特率为 19200 则%MB3.4011 填 4，依次类推 ）
例如： 设置站号为 10 波特率为 19200 其他格式保持不变。 编程如下 %MB3.4010 初值填 10；%MB3.4011 初值填 4；完成后编译下装，重新上电控制器后站号和波特率修改成功。

注意 ： 当使用 RS 232 通讯时，可以通过设置特殊寄存器%MB.4019 的值选择 RS 232 通讯

与 RS 485 是否共享内存地址，设 0 为否（MODBUS-232 通讯首地址人机 0 对应我们的共享内存地址 15000，也就是%Mx3.15000），设 1 为是。**如需使用此功能且%MB.4019 的值需设 0 时，旧版本软件需要更新 SHM_LIB.dll 文件，请联系威科达技术人员更新。**

12.1.3 运动控制器通讯地址与人机地址对应关系

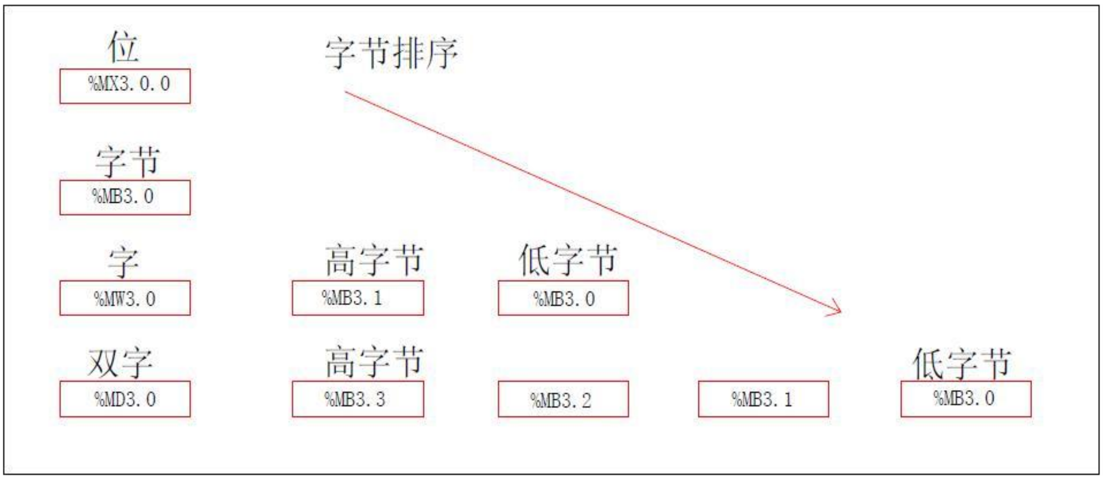
（1）运动控制器地址关系

数据的存取必须指定地址，地址以 %MX3.开头，其中“X”可以是位“%MX3.”；可以是字节“%MB3.”；可以是字“%MW3.”；也可以是双字“%MD3.”；用小数点 “.” 加整数表示存储的地址，如%MX3.0.0 表示数据内存映射区第 0 个字节的第 0 个位，下表是数据的地址特性

序号	前缀		定义约定	数据类型
1	位置前缀	I	输入映射区	
2		Q	输出映射区	
3		M	中间变量映射区	
4	大小前缀	X	位	BOOL
5		B	字节（8 位）	BYTE
6		W	字（16 位）	WORD
7		D	双字（32）	DWORD
8		L	长形（64 位）	LREAL

（2） 运动控制器地址之间的关系

字节、字和双字之间的地址关系是一个双字包含两个字，或者说包含四个字节，下面以%MX3.0.0、%MB3.0、%MW3.0 和%MD3.0 为例说明字节、字和双字之间的地址关系和数据排列 如下图所示：



如：一个十六进制数 16#1234 存放在%MW3.0 中，则 16#34 存在%MB3.0 中，16#12 存放在%MB3.1 中。程序中若对位操作，则会影响到位所在字节、字和双字，反过来也一样。

变量地址的示例

%IX1.3 表示数字量输入映射区中第 1 个字节中的第 3 位；
%QX0.0 表示数字量输出映射区中第 0 个字节中的第 0 位；
%MX3.0.0 表示中间变量区中第 0 个字节的第 0 位；
%MD3.4 表示中间变量区中第 4 个字节开始的 1 个双字；
（3）运动控制器与人机地址对应关系：

运动控制器地址=（人机地址-1）*2　（威纶通和繁易人机）

位操作：

PLC 地址类型	人机地址类型
%IX0.0	无
%QX0.0	无
%MX3.0.0	0X
例：　PLC 地址类型　%MX3.6.0　对应　人机地址类型　0X　地址　4；	

字节操作：

PLC 地址类型	人机地址类型
%IB0.0	无
%QB0.0	无
%MB3.0.0	4（3）X
例：　PLC 地址类型　%MB3.10　对应　人机地址类型　4（3）X　地址　6　（注意数据类型）	

字操作：

PLC 地址类型	人机地址类型
%IW0.0	无
%QW0.0	无
%MW3.0.0	4（3）X
例：　PLC 地址类型　%MW3.14　对应　人机地址类型　4（3）X　地址　8　（注意数据类型）	

双字操作

PLC 地址类型	人机地址类型
%ID0.0	无
%QD0.0	无
%MD3.0.0	4（3）X
例：　PLC 地址类型：　%MD3.20　对应　人机地址类型　4（3）X　地址　11	

12.2 更改拓展模块类型

12.2.1 自动识别拓展类型

地址类型	赋值
%MB3.9800	0

12.2.2 手动选择拓展类型

地址类型	赋值
%MB3.9800	1
%MB3.9801（第一个拓展的拓展类型）	设 3 代表 8DI8DO , 设 4 代表 16DI, 设 5 代表 16DO , 设 6 代表模拟量, 设 7 代表 PT100 , 设 8 代表称重 ， 设 9 代表热电偶 ， 设 10 代表物联
%MB3.9802（第二个拓展的拓展类型）	同%MB3.9801
%MB3.9803（第三个拓展的拓展类型）	同%MB3.9801
...	...
%MB3.9808（第八个拓展的拓展类型）	同%MB3.9801

附件一 编程注意事项

使用梯形图编程语言时，尽量避免用同一个指令对同一个变量进行多处操作。梯形图编程，程序的扫描顺序是从上到下，从左到右。编程时，不要在编辑区拷贝粘贴指令（功能块），否则编译出错，可以从右侧编辑向导中拖入编辑区，或在编辑区直接键入指令（功能块）的名称。

以下指令暂时不支持:

GET_CHAR

GET_ERROR

GET_ERROR CATALOG

GET_SYM

CLR_OUT

COLD_RESTART

CONTINUE

HO_ RESTART

WRITE_RETAIN

WARM_RESTART

写 PDD 变量的指令，如 WR_BOOL_BY_SYMFOR 循环:步数不能太长，如 15000，否则可能出错误，一个程序中，也不能出现多个较大的 FOR 循环（如 10000 步），否则也可能出错。数组 ARRAY 的最大长度是 32767

附件二 ASCII 码表

ASCII 的全称是美国信息交换标准码，是 American Standard Code for Information Interchange 的缩写，ASCII 码定义从 0 到 127 的一百二十八个数字所代表的英文字母或数字或符号，所有使用 ASCII 的计算机之间可以互相读取同一份文件而不会有不一样的结果与意义。由于只使用 7 个位（BIT）就可以表示从 0 到 127 的数字，大部分的计算机都使用 8 个位来存取字符集（CHARACTER SET），见表 1 和表 2。从 128 到 255 之间的数字可以用来代表另一组一百二十八个符号，称为 EXTENDED ASCII，见表 3。表 1 中的十进制的 0~31 和 127 为控制字符，表 2 中的 32~126 为可打印字符;表 3 中 128~255 为扩展 ASCII 码。

表 1 控制字符（表中 Hx 表示十六进制，Dec 表示十进制）

Dec	HX	字符	描述	Dec	Hx	描述
0	0	NULL	Null/空字符	17	11	Device Control 1/设备控制 1
1	1	SOX	Start Of Heading/标题开始	18	12	Device Control 1/设备控制 2
2	2	STX	Start Of Text/正文开始	19	13	Device Control 1/设备控制 3
3	3	ETX	End Of Text/正文结束	20	14	Device Control 1/设备控制 4
4	4	EOT	End Of Transmission/传输结束	21	15	Negtive Acknowledge/拒绝接收
5	5	ENO	Enquiry/请求	22	16	Synchronous Idle/同步空闲
6	6	ACK	Acknowledge/收到通知	23	17	End of Trans. Block/传输块结束
7	7	BEL	Bell/响铃	24	18	Cancel/取消
8	8	BS	Backspace/退格	25	19	End of Medium/介质中断
9	9	HT	Horizontal Tab/水平制表符	26	1A	Substitute/替补
10	OA	LF	NL line feed, new	27	1B	Escape/溢出

			line/换行			
11	OB	VT	Vertical Tab/垂自制表符	28	1C	File Separator/文件分割符
12	OC	FF	NP form feed; new page/换页	29	1D	Group Separator/分组符
13	OD	CR	Carriage Return/回车	30	1E	Record Separator/记录分离符
14	OE	SO	Shift Out/停 切换	31	1F	Unit Separator/单元分隔符
15	0F	SI	Shift In/启用切换	127	7F	Delete/删除
16	10	DLE	Data Link Escape/数据链路转义			

Dec	Hx	控制字符	Dec	Hx	控制字符	Dec	Hx	控制字符	Dec	Hx	控制字符
32	20	（space）	56	38	8	80	50	P	104	68	h
33	21	!	57	39	9	81	51	Q	105	69	i
34	22	“	58	3A	:	82	52	R	106	6A	j
35	23	#	59	3B	;	83	53	S	107	6B	k
36	24	$	60	3C	<	84	54	T	108	6C	l
37	25	%	61	3D	=	85	55	U	109	6D	m
38	26	&	62	3E	>	86	56	V	110	6E	n
39	27	,	63	3F	?	87	57	W	111	6F	o
40	28	（	64	40	@	88	58	X	112	70	p
41	29	）	65	41	A	89	59	Y	113	71	q
42	2A	*	66	42	B	90	5A	Z	114	72	r
43	2B	+	67	43	C	91	5B	[	115	73	s
44	2 C	,	68	44	D	92	5C	/	116	74	t
45	2D	-	69	45	E	93	5D	]	117	75	u
46	2E	.	70	46	F	94	5E	^	118	76	v
47	2F	/	71	47	G	95	5F	—	119	77	w
48	30	0	72	48	H	96	60	、	120	78	x
49	31	1	73	49	I	97	61	a	121	79	y
50	32	2	74	4A	J	98	62	b	122	7A	z

51	33	3	75	4B	K	99	63	c	123	7B	{
52	34	4	76	4C	L	100	64	d	124	7C	|
53	35	5	77	4D	M	101	65	e	125	7D	}
54	36	6	78	4E	N	102	66	f	126	7E	~
55	37	7	79	4F	O	103	67	g			

表 2 可打印字符（表中 Hx 表示十六进制，Dec 表示十进制）

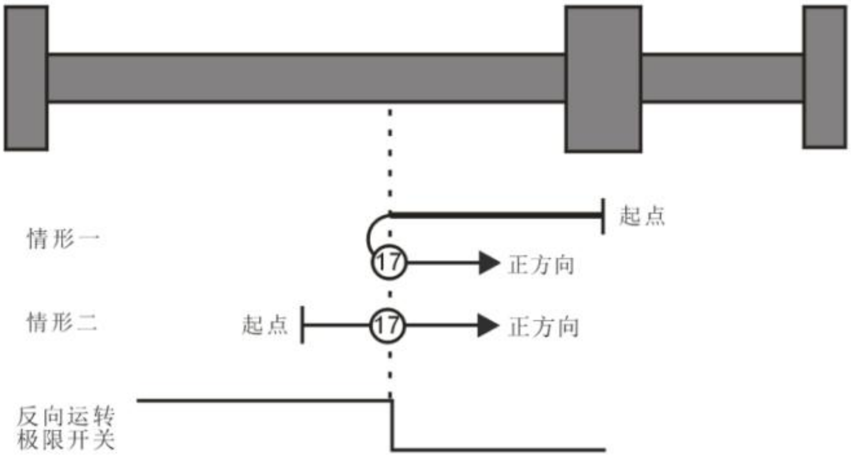
附件三 原点回归模式说明

我司运动控制器有多种原点回归模式。用户可根据现场条件及工艺要求选择合适的原点回零模式 。

- **原点回归模式 17 取决于反向运转极限开关的原点回归**

情形一：当反向极限开关状态处于低位时执行 MC_Home 指令，轴开始以第一段速反向运动，当遇到反向极限开关处于高位时，运动方向改变且以第二段速开始运动，当遇到反向极限开关处于低位时的位置就是原点位置。

情形二：当反向极限开关状态处于高位时执行 MC_Home 指令，轴开始以第二段速正向运动，当遇到反向极限开关处于低位时的位置就是原点位置。

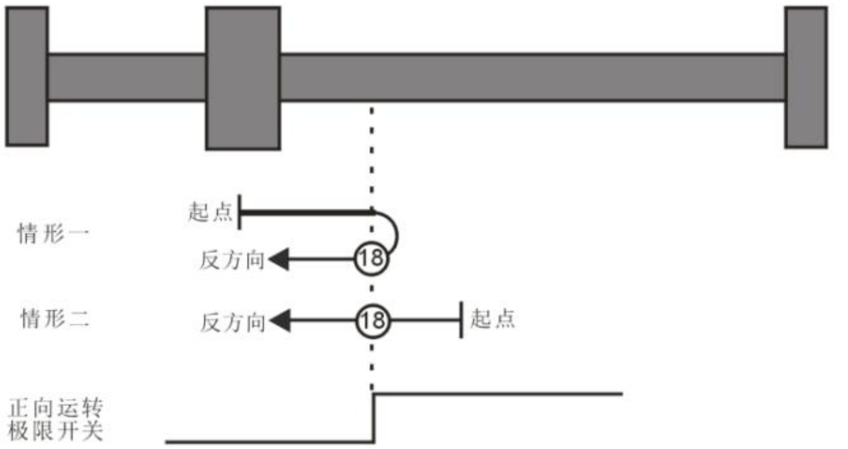


取决于反向运转极限开关的原点回零，上图中的表示17原点回零模式17

- **原点回归模式 18 取决于正向运转极限开关的原点回归**

情形一：当正向极限开关处于低位时执行 MC_Home 指令，轴开始以第一段速正向运动，当遇到正向运转极限开关处于高位时，运动方向改变且以第二段速开始运动，在正向运转极限开关状态处于低位时的位置就是原点位置。

情形二：当正向运转极限开关状态处于高位时执行 MC_Home 指令，轴直接以第二段速开始反向运动，在正向运转极限开关状态处于低位时的位置就是原点位置

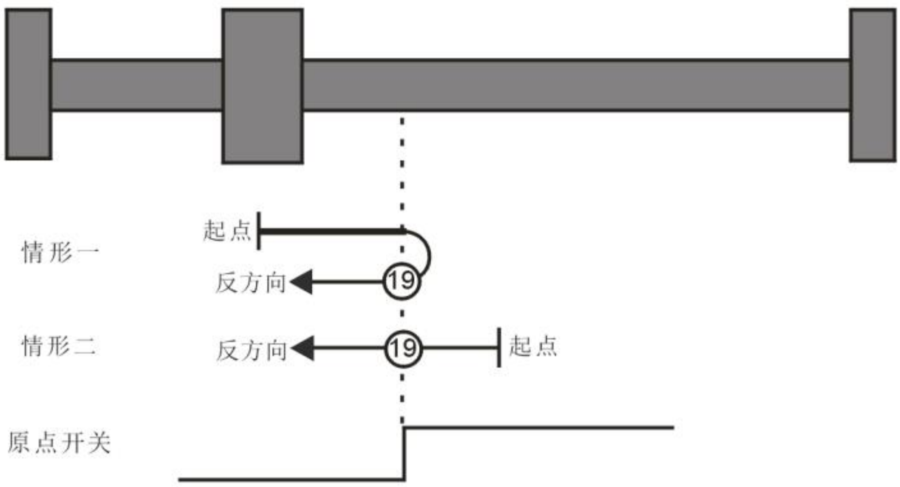


取决于正向运转极限开关的原点回零，上图中的表示18原点回零模式18

- **原点回归模式 19 取决于原点开关的原点回归**

情形一：当原点开关状态处于低位时执行 MC_Home 指令，轴开始以第一段速正向运动，当遇到原点开关处于高位时，运动方向改变且以第二段速开始运动，当遇到原点开关处于低位时的位置就是原点位置。

情形二：当原点开关状态处于高位时执行 MC_Home 指令，轴直接以第二段速开始反向运动，当遇到原点开关处于低位时的位置就是原点位置。

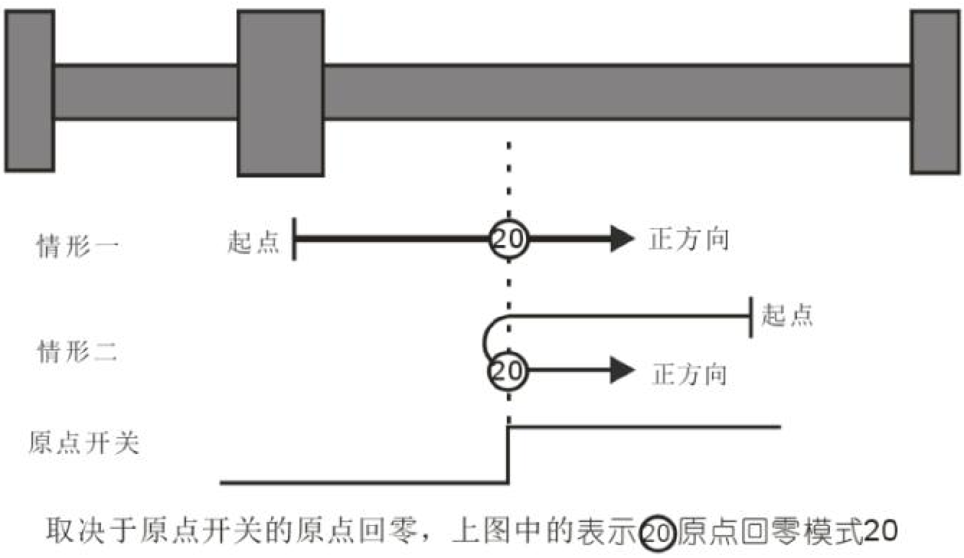


取决于原点开关的原点回零，上图中的表示19原点回零模式19

- **原点回归模式 20 取决于原点开关的原点回归**

情形一：当原点开关状态处于低位时执行 MC_Home 指令，轴开始以第一段速正向运动，当遇到原点开关处于高位时的位置就是原点位置。

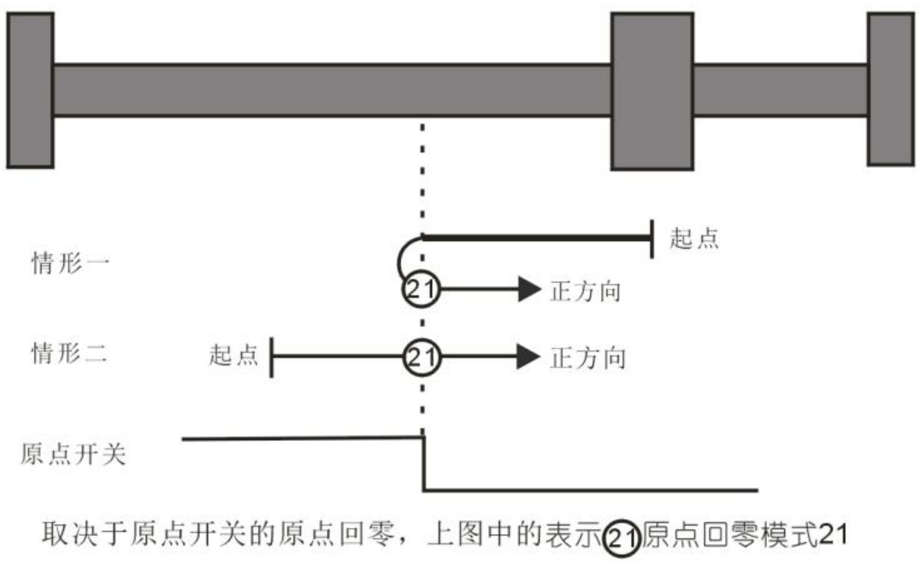
情形二：当原点开关状态处于高位时执行 MC_Home 指令，轴直接以第二段速开始反向运动，当遇到原点开关处于低位时运动方向改变且以第二段速开始运动。当遇到原点开关再次处于高位时的位置就是原点位置。



- **原点回归模式 21 取决于原点开关的原点回归**

情形一：当原点开关状态处于低位时执行 MC_Home 指令，轴开始以第一段速反向运动，当遇到原点开关处于高位时，运动方向改变且以第二段速开始运动，当遇到原点开关处于低位时的位置就是原点位置。

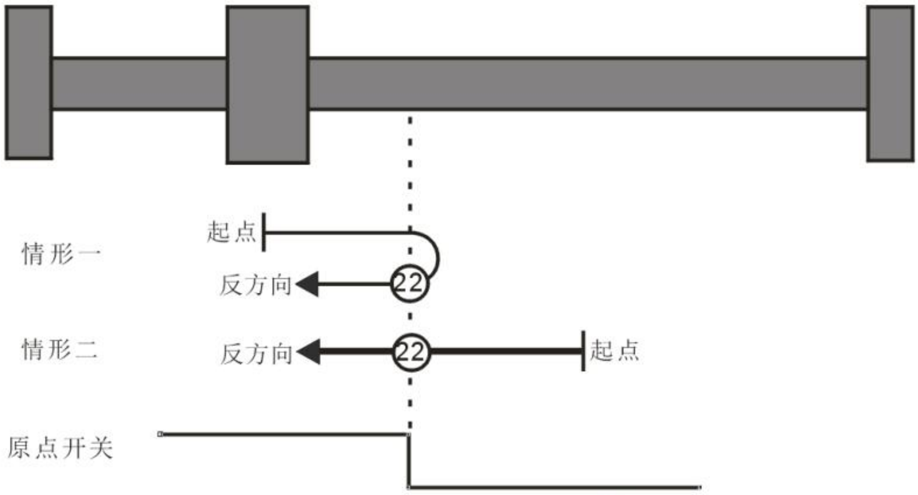
情形二：当原点开关状态处于高位时执行 MC_Home 指令，轴直接以第二段速开始正向运动，当遇到原点开关处于低位时的位置就是原点位置 。



- **原点回归模式 22 取决于原点开关的原点回归**

情形一：当原点开关状态处于高位时执行 MC_Home 指令，轴直接以第二段速开始正向运动，当遇到原点开关处于低位时运动方向改变且以第二段速开始运动。当遇到原点开关处于高位时的位置就是原点位置 。

情形二：当原点开关状态处于低位时执行 MC_Home 指令，轴开始以第一段速反向运动，当遇到原点开关处于高位时的位置就是原点位置。



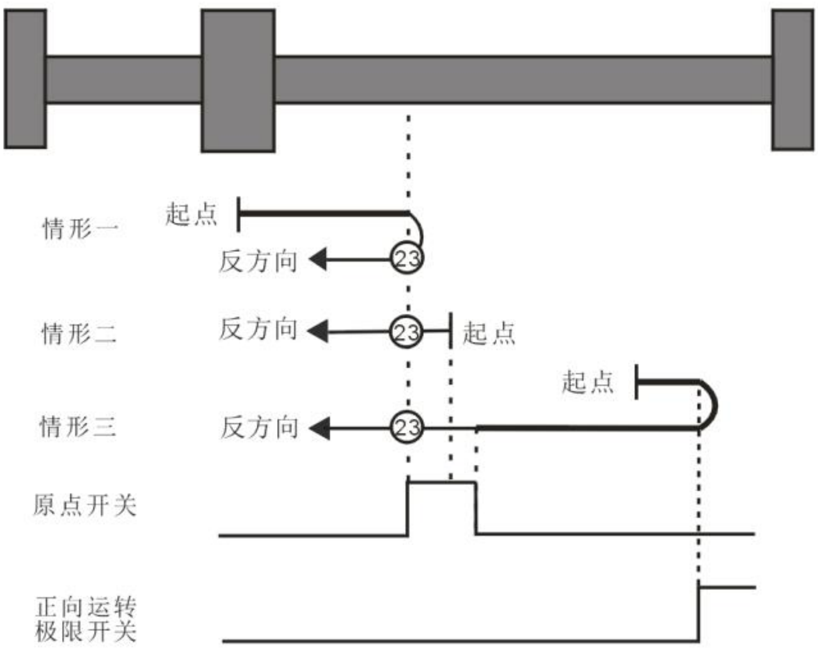
取决于原点开关的原点回零，上图中的表示22原点回零模式22

- **原点回归模式 23 取决于原点开关与正向运转极限开关的原点回归**

情形一：当原点开关状态处于低位时执行 MC_Home 指令，轴开始以第一段速正向运动，当遇到原点开关处于高位时，运动方向改变且以第二段速开始运动，在原点开关状态处于低位时的位置就是原点位置。

情形二：当原点开关状态处于高位时执行 MC_Home 指令，轴直接以第二段速开始反向运动，在原点开关状态处于低位时的位置就是原点位置。

情形三：当原点开关状态处于低位时执行 MC_Home 指令，轴开始以第一段速正向运动，当原点开关处于低位且遇到正向运转极限开关处于高位时，运动方向改变且以第一段速开始运动，当遇到原点开关处于高位时，以第二段速开始运动，在原点开关状态处于低位时的位置就是原点位置。



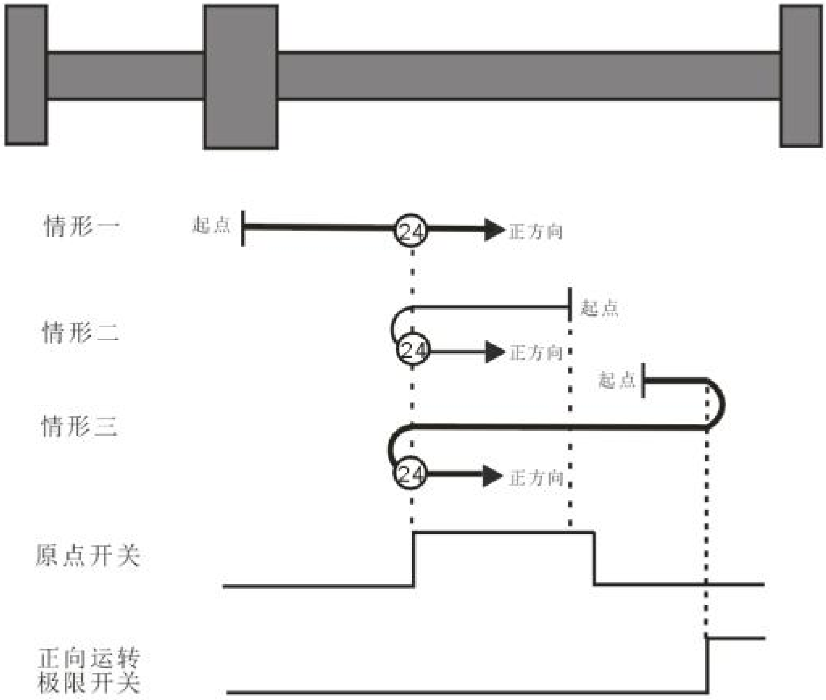
取决于原点开关、正向运转极限开关的原点回零，上图中的表示23原点回零模式23

- **原点回归模式 24 取决于原点开关与正向运转极限开关的原点回归**

情形一：当原点开关状态处于低位时执行 MC_Home 指令，轴开始以第一段速正向运动，当遇到原点开关处于高位时的位置就是原点位置。

情形二：当原点开关状态处于高位时执行 MC_Home 指令，轴直接以第二段速开始反向运动，当遇到原点开关处于低位时运动方向改变且以第二段速开始运动。当遇到原点开关再次处于高位时的位置就是原点位置。

情形三：当原点开关状态处于低位时执行 MC_Home 指令，轴开始以第一段速正向运动，当原点开关处于低位且遇到正向运转极限开关处于高位时，运动方向改变且以第一段速开始运动，当遇到原点开关处于高位时，仍以第一段速运动，在原点开关处于低位时，运动方向改变且以第一段速开始运动，在原点开关出现高位时的位置就是原点位置。



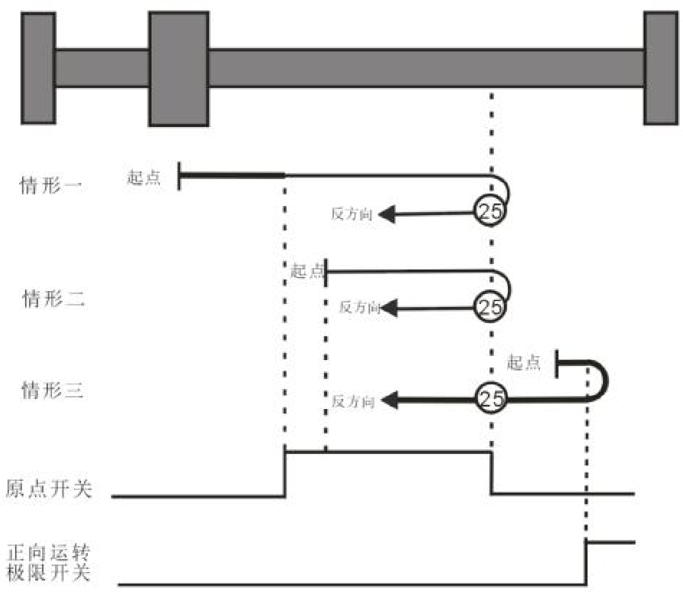
取决于原点开关、正向运转极限开关的原点回零，上图中的表示24原点回零模式24

- **原点回归模式 25 取决于原点开关与正向运转极限开关的原点回归**

情形一：当原点开关状态处于低位时执行 MC_Home 指令，轴开始以第一段速正向运动，当遇到原点开关处于高位时，以第二段速开始正向运动，当遇到原点开关处于低位时，运动方向改变且以第二段速开始运动，当遇到原点开关处于高位时的位置就是原点位置。

情形二：当原点开关状态处于高位时执行 MC_Home 指令，轴开始以第二段速正向运动，当遇到原点开关处于低位时，运动方向改变且以第二段速开始运动，当遇到原点开关处于高位时的位置就是原点位置。

情形三：当原点开关状态处于低位时执行 MC_Home 指令，轴开始以第一段速正向运动，当原点开关处于低位且遇到正向运转极限开关处于高位时，运动方向改变且以第一段速开始运动，当遇到原点开关处于高位时位置就是原点位置。



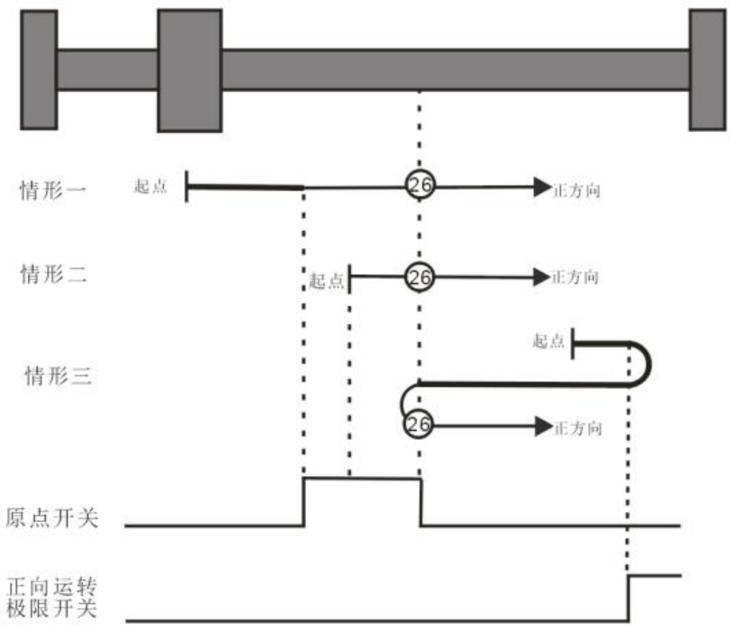
取决于原点开关、正向运转极限开关的原点回零，上图中的表示25原点回零模式25

- **原点回归模式 26 取决于原点开关与正向运转极限开关的原点回归**

情形一：当原点开关状态处于低位时执行 MC_Home 指令，轴开始以第一段速正向运动，当遇到原点开关处于高位时，以第二段速开始运动，当遇到原点开关处于低位时的位置就是原点位置。

情形二：当原点开关状态处于高位时执行 MC_Home 指令，轴开始以第二段速正向运动，当遇到原点开关处于低位时的位置就是原点位置。

情形三：当原点开关状态处于低位时执行 MC_Home 指令，轴开始以第一段速正向运动，当原点开关处于低位且遇到正向运转极限开关处于高位时，运动方向改变且以第一段速开始运动，当遇到原点开关处于高位时，运动方向再次改变且以第二段速开始运动，当原点开关处于低位时的位置就是原点位置。



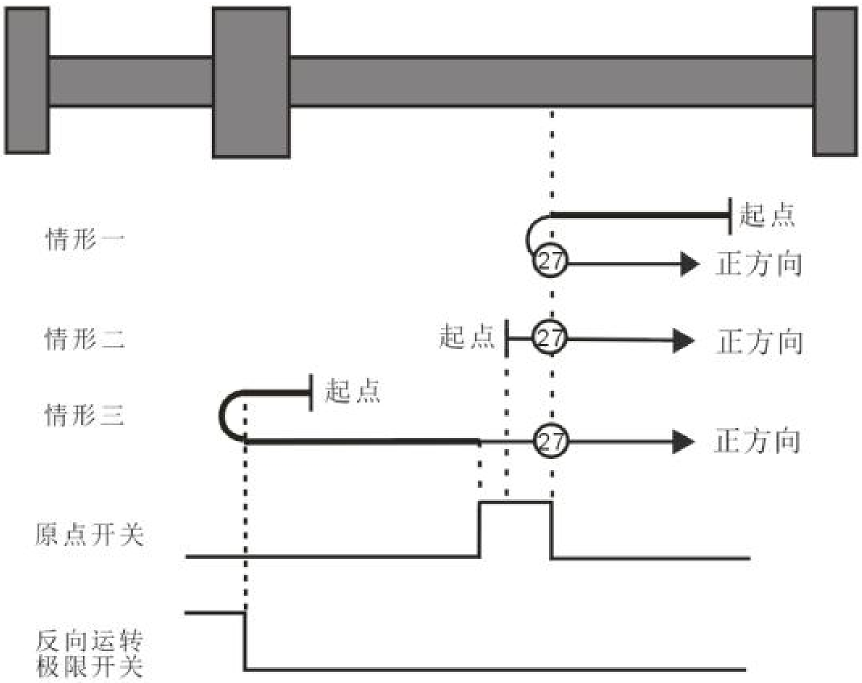
取决于原点开关、正向运转极限开关的原点回零，上图中的表示26原点回零模式26

- **原点回归模式 27 取决于原点开关与反向运转极限开关的原点回归**

情形一：当原点开关状态处于低位时执行 MC_Home 指令，轴开始以第一段速反向运动，当遇到原点开关处于高位时，运动方向改变且以第二段速开始运动，在原点开关状态处于低位时的位置就是原点位置。

情形二：当原点开关状态处于高位时执行 MC_Home 指令，轴直接以第二段速开始正向运动，在原点开关状态处于低位时的位置就是原点位置。

情形三：当原点开关状态处于低位时执行 MC_Home 指令，轴开始以第一段速反向运动，当原点开关处于低位且遇到反向运转极限开关处于高位时，运动方向改变且以第一段速开始运动，当遇到原点开关处于高位时，以第二段速开始运动，在原点开关状态处于低位时的位置就是原点位置。



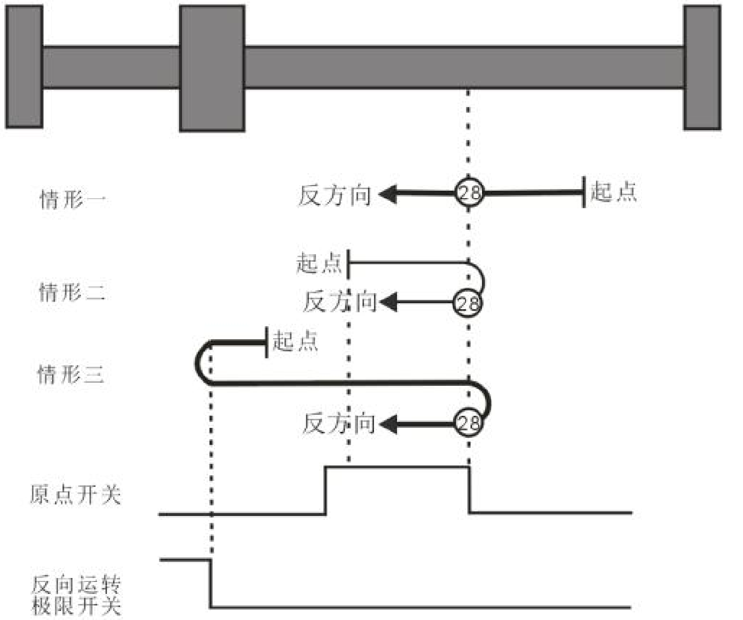
取决于原点开关、反向运转极限开关的原点回零，上图中的表示27原点回零模式27

- **原点回归模式 28 取决于原点开关与反向运转极限开关的原点回归**

情形一：当原点开关状态处于低位时执行 MC_Home 指令，轴开始以第一段速反向运动，当遇到原点开关处于高位时的位置就是原点位置。

情形二：当原点开关状态处于高位时执行 MC_Home 指令，轴直接以第二段速开始正向运动，在原点开关状态处于低位时改变运动方向且以第二段速度开始运动，当原点开关处于高位时的位置就是原点位置。

情形三：当原点开关状态处于低位时执行 MC_Home 指令，轴开始以第一段速反向运动，当原点开关处于低位且遇到反向运转极限开关处于高位时，运动方向改变且以第一段速开始运动，当遇到原点开关处于高位时，仍以第一段速运动，在原点开关状态处于低位时，运动方向改变且以第一段速开始运动，在原点开关状态处于高位时的位置就是原点位置。



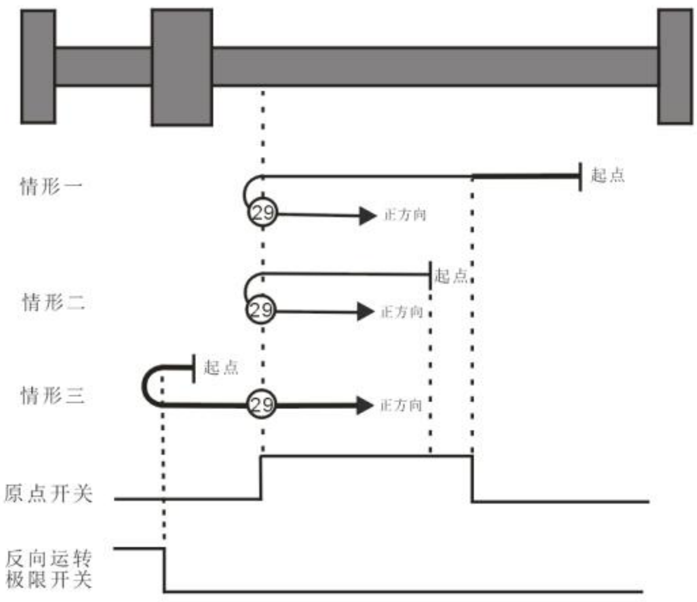
取决于原点开关、反向运转极限开关的原点回零，上图中的表示28原点回零模式28

- **原点回归模式 29 取决于原点开关与反向运转极限开关的原点回归**

情形一：当原点开关状态处于低位时执行 MC_Home 指令，轴开始以第一段速反向运动，当遇到原点开关处于高位时，以第二段速开始运动，当遇到原点开关处于低位时改变运动方向且以第二段速运行，当遇到原点开关再次处于高位时的位置就是原点位置。

情形二：当原点开关状态处于高位时执行 MC_Home 指令，轴开始以第二段速反向运动，当遇到原点开关处于低位时改变方向且以第二段速开始运动，当遇到原点开关处于高位的位置就是原点位置。

情形三：当原点开关状态处于低位时执行 MC_Home 指令，轴开始以第一段速反向运动，当原点开关处于低位且遇到反向运转极限开关处于高位时，运动方向改变且以第一段速开始运动，当遇到原点开关处于高位时的位置就是原点位置。



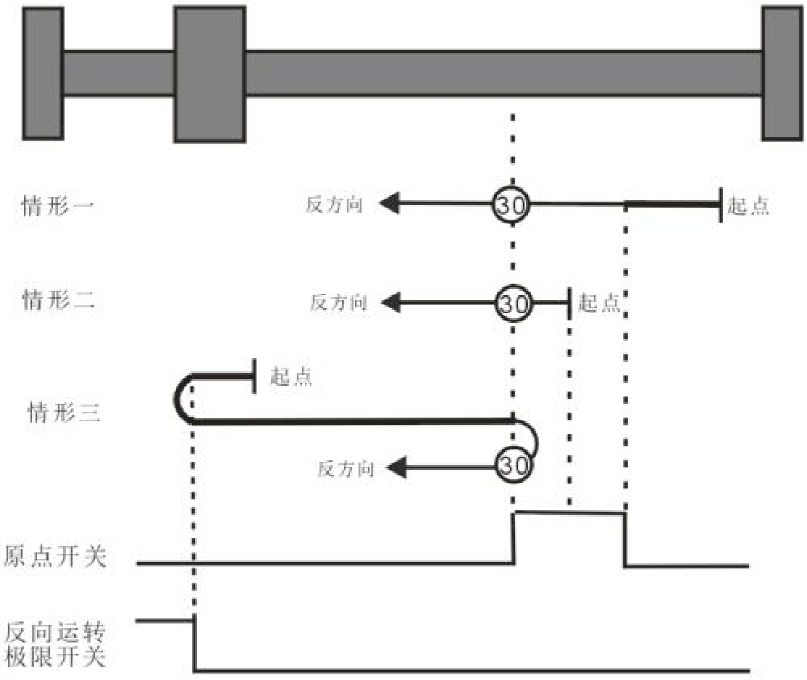
取决于原点开关、反向运转极限开关的原点回零，上图中的表示29原点回零模式29

- **原点回归模式 30 取决于原点开关与反向运转极限开关的原点回归**

情形一：当原点开关状态处于低位时执行 MC_Home 指令，轴开始以第一段速反向运动，当遇到原点开关处于高位时，以第二段速开始运动，当遇到原点开关处于低位时的位置就是原点位置。

情形二：当原点开关状态处于高位时执行 MC_Home 指令，轴开始以第二段速反向运动，当遇到原点开关处于低位时的位置就是原点位置。

情形三：当原点开关状态处于低位时执行 MC_Home 指令，轴开始以第一段速反向运动，当原点开关处于低位且遇到反向运转极限开关处于高位时，运动方向改变且以第一段速开始运动，当遇到原点开关处于高位时，运动方向再次改变且以第二段速开始运动，当遇到原点开关处于低位时的位置就是原点位置。



取决于原点开关、反向运转极限开关的原点回零，上图中的表示30原点回零模式30

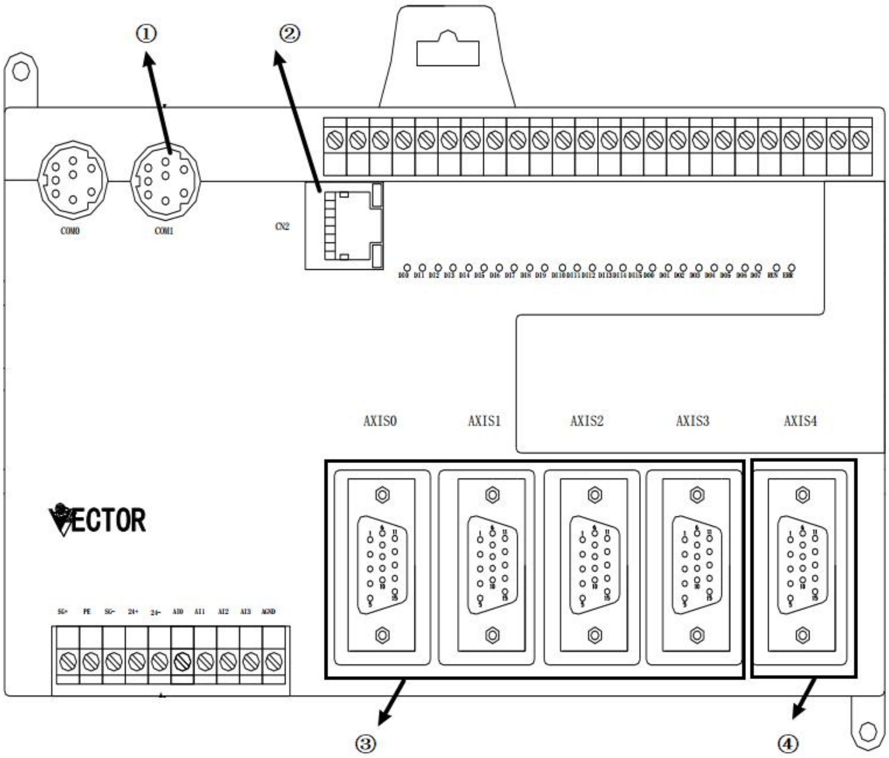
- **原点回归模式 35 轴的当前位置被认为是原点回归的位置**

在模式 35 下，执行 MC_Home 指令，轴不运动，轴的当前位置被认为是原点回归的位置。

附件四 CANopen 相关说明

1. CANopen 通讯连接

1.1 运动控制器接线端口说明



如图所示：

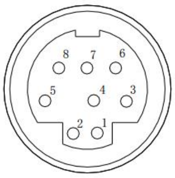
①CANopen 通信口，通过此接口与从站节点进行数据交换，传输同步信号以及收发同步报文；

②100M 以太网口，通过此接口上传下载程序及在线监控；

③AXIS0~AXIS3 对于 CANopen 型无效；

④AXIS4 为主轴接口，轴号为 16，仅仅用于做多轴指令的主轴（接编码器或脉冲发生器），注意的是，其他接口功能与模拟量相同。

1.2 CANopen 通讯口引脚定义

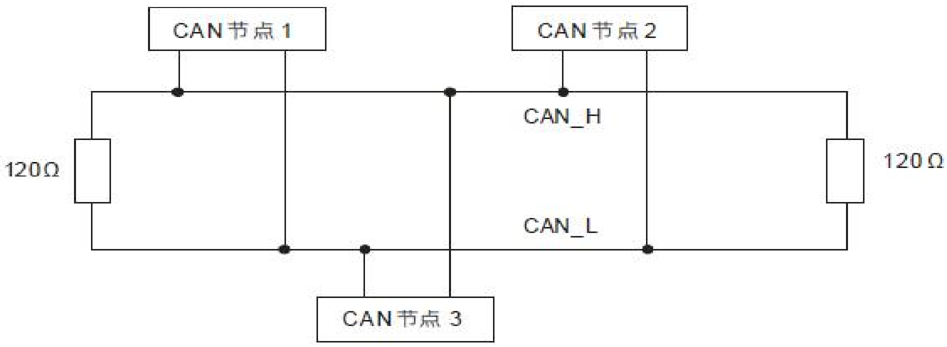
<table>
<tr><td rowspan="9">RS-232
（COM1）</td><td rowspan="9"></td><td>脚位</td><td>定义</td></tr>
<tr><td>1</td><td>NC</td></tr>
<tr><td>2</td><td>NC</td></tr>
<tr><td>3</td><td>NC</td></tr>
<tr><td>4</td><td>NC</td></tr>
<tr><td>5</td><td>NC</td></tr>
<tr><td>6</td><td>CANL</td></tr>
<tr><td>7</td><td>CANH</td></tr>
<tr><td>8</td><td>GND</td></tr>
</table>

CANL 对应信号负（-）；CANH 对应信号正(+)；GND 为信号地，需要与所连接的设备的信号地共通，焊线时请注意孔座和插针的方向。

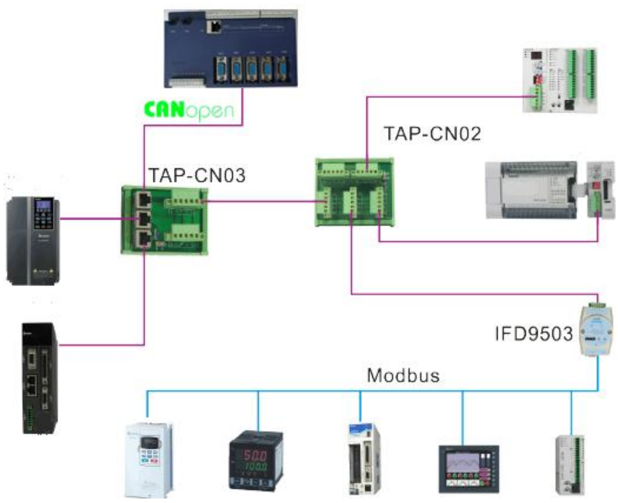
1.3 CANopen 通讯口网络连接

CANopen 总线终端和网络拓扑结构：

为了增强 CANopen 通讯的稳定性，CANopen 总线网络的两个终端需接入 120 欧姆的终端电阻。下图所示为基本的 CANopen 网络拓扑结构示意图。



CANopen 总线网络拓扑架构



1.4 CANopen 通讯口通讯速率与通讯距离

支持的 CANopen 通讯速率：20K,50K,125K,250K,500K,1Mbps，通信速率与通信距离，下图对应各个频率段的最大传输距离

传输速度（位/秒）	20K	50K	125K	250K	500K	1M
最大通讯距离（米）	2500	1000	500	250	100	25

2. CANopen 协议基础知识

2.1 网络管理（NMT）

支持 NMT（Network Management Object：网络管理对象）Master 服务,包括对网络的复位、停止、预操作、启动等。

支持 NMT 错误控制，NMT 错误控制用于监控从站是否掉线。NMT 错误控制分为 Heartbeat 和 NodeGuarding 两种,本机支持 Heartbeat。

2.2 服务数据（SDO）

支持在 PLC 梯形图中使用梯形图对从站进行非实时数据的读写服务，读写区域需参照制造商定义。

2.3 过程数据（PDO）

支持 PDO（Process Data Object：过程数据对象）服务：
RxPDO 最大支持 200 个，数据量最大支持 1000 个字节
TxPDO 最大支持 200 个，数据量最大支持 1000 个字节
每个从站最多可配置 4 个 TxPDO 和 4 个 RxPDO
PDO 传输类型：支持事件触发，时间触发，同步周期，同步非周期
PDO 映射：每个 PDO 最大可以映射 32 个字节
支持的映射数据类型：

存储空间	数据类型
1bit	BOOL
8bit	SINT USINT BYTE
16bit	INT UINT WORD
32bit	DINT UDINT REAL DWORD
64bit	LINT ULINT LREAL LWORD

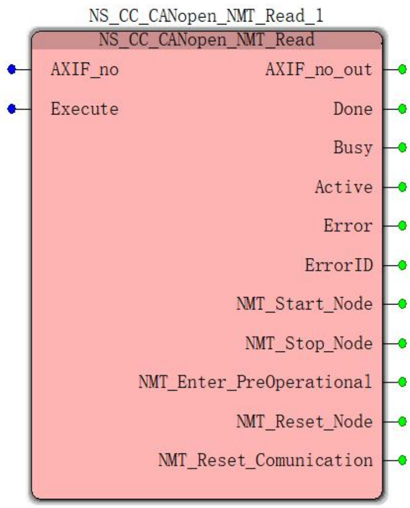
更多请参考 CANopen 标准协议 DS301v4.02 和 DS402 关于电机驱动子协议。

3. 软件功能

3.1 总线初始化配置模块

3.1.1 NS_CC_CANopen_NMT_Read（网络状态读取指令）

FB/FC	说明	适用机种
FB	此指令用于读取当前网络设备所处的网络状态	



➢ 输入参数

名称	功能	数据类型	设定范围（缺省值）	生效时机
AXIF_no（节点号）	设定指令欲控制的节点，该功能只支持读取主站（18）的网络状态	WORD	18	Excute 为 TRUE 时
Excute（执行位）	当 Excute 为 True 时，执行该指令。	BOOL	TRUE 或 FALSE(FALSE)	Excute 为 TRUE 时

➢ 输出参数

名称	功能	数据类型	输出范围
AXIF_no_out（节点号输出）	该参数为指令执行时的节点号输出	WORD	18
Done	该参数输出为 TRUE 时表示指令执行完成	BOOL	TRUE 或 FALSE

Busy（执行中）	该参数输出为 TRUE 时表示指令正在执行中	BOOL	TRUE 或 FALSE
Active（控制中）	该参数输出为 TRUE 时表示指令正在控制轴	BOOL	TRUE 或 FALSE
Error（错误输出）	该参数为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE
ErrorID（错误代码）	指令执行出错时的错误代码	UDINT	-
NMT_Start_Node（网络节点启动）	该参数为 TRUE 时表示节点网络状态为启动状态	BOOL	TRUE 或 FALSE
NMT_Stop_Node（网络节点停止）	该参数为 TRUE 时表示节点网络状态为停止状态	BOOL	TRUE 或 FALSE
NMT_Enter_Preoperational（网络进入预操作模式）	该参数为 TRUE 时表示节点网络状态为预操作状态	BOOL	TRUE 或 FALSE
NMT_Reset_Node（网络节点复位）	该参数为 TRUE 时表示节点网络状态为复位状态	BOOL	TRUE 或 FALSE
NMT_Reset_Communication（通信复位）	该参数为 TRUE 时表示网络通信状态为复位状态	BOOL	TRUE 或 FALSE

➢ **功能说明**

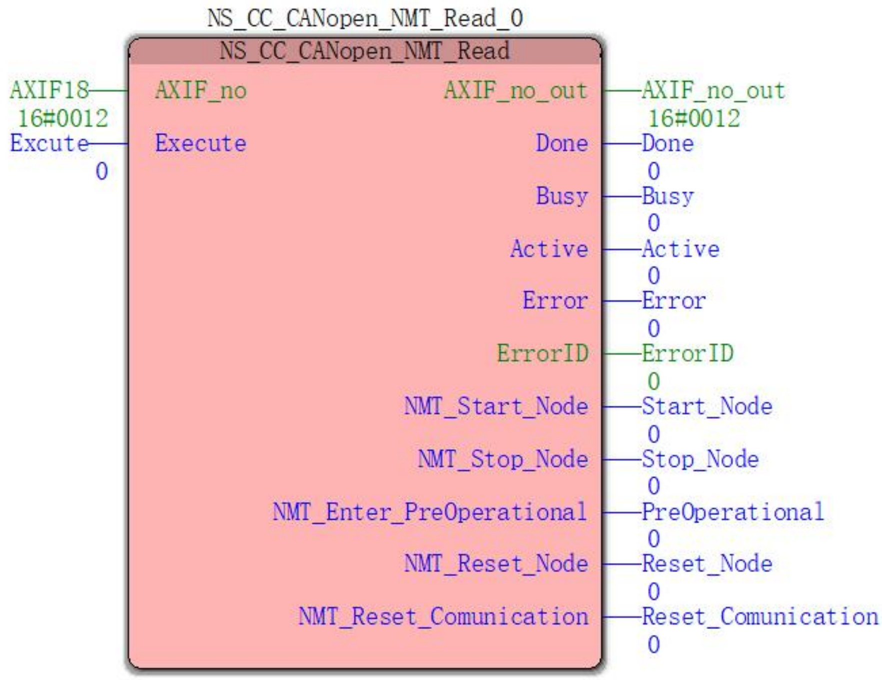
给定轴号（包括 Master 站号），触发模块可读取当前网络主站设备所处的网络状态，所读到的状态会反映到模块输出端，当对应的状态读取成功后，输出位 Done 变为 Ture，输出端对应当前状态的 BOOL 变量会变为 TRUE。

程序范例

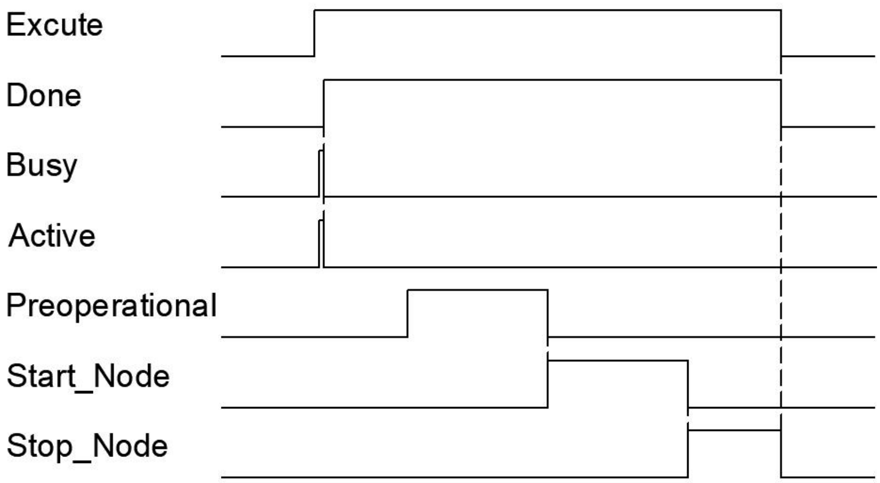
NS_CC_CANopen_NMT_Read 指令单独执行时的范例如下所示。

3、变量和程序

变量名	数据类型	初始值
NS_CC_CANopen_NMT_Read_0	NS_CC_CANopen_NMT_Read	
AXIF18	USINT	18
Excute	BOOL	FALSE
AXIF_no_out	WORD	1
Done	BOOL	0
Busy	BOOL	
Active	BOOL	
Error	BOOL	
ErrorID	USINT	
Start_Node	BOOL	
Stop_Node	BOOL	
PreOperational	BOOL	
Reset_Node	BOOL	
Reset_Comunication	BOOL	

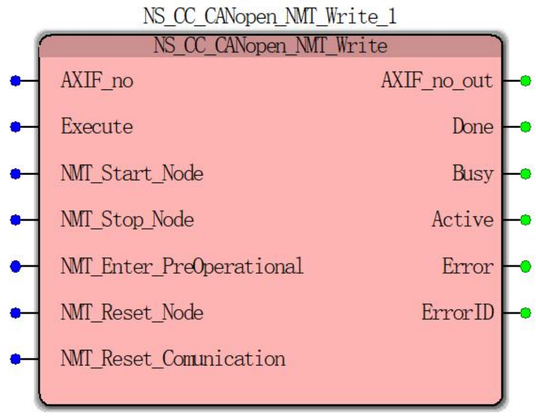

NS_CC_CANopen_NMT_Read_0
NS_CC_CANopen_NMT_Read
AXIF18
16#0012
Excute
0
AXIF_no
Execute
AXIF_no_out
Done
Busy
Active
Error
ErrorID
NMT_Start_Node
NMT_Stop_Node
NMT_Enter_PreOperational
NMT_Reset_Node
NMT_Reset_Comunication
AXIF_no_out
16#0012
Done
0
Busy
0
Active
0
Error
0
ErrorID
0
Start_Node
0
Stop_Node
0
PreOperational
0
Reset_Node
0
Reset_Comunication
0

4、时序图


Excute
Done
Busy
Active
Preoperational
Start_Node
Stop_Node

3.1.2 NS_CC_CANopen_NMT_Write（网络状态写入指令）

FB/FC	说明	适用机种
FB	此指令用于写入当前网络各个网络节点的设备所处的网络状态	



➢ **输入参数**

名称	功能	数据类型	设定范围（缺省值）	生效时机
AXIF_no（节点号）	设定指令欲控制的节点	WORD	1~16、18	Excute 为 TRUE 时
Excute（执行位）	当 Excute 为 True 时，执行该指令	BOOL	TRUE 或 FALSE(FALSE)	
NMT_Start_Node（启动节点）	该参数为 TRUE 时，启动网络节点	BOOL	TRUE 或 FALSE(FALSE)	
NMT_Stop_Node（停止节点）	该参数为 TRUE 时，停止网络节点	BOOL	TRUE 或 FALSE(FALSE)	
NMT_Enter_Preoperational（网络进入预操作模式）	该参数为 TRUE 时，网络状态进入预操作	BOOL	TRUE 或 FALSE(FALSE)	
NMT_Reset_Node（复位节点）	该参数为 TRUE 时，复位网络节点	BOOL	TRUE 或 FALSE(FALSE)	
NMT_Reset_Communication（复位通信）	该参数为 TRUE 时，复位网络通信	BOOL	TRUE 或 FALSE(FALSE)	

➢ 输出参数

名称	功能	数据类型	输出范围
AXIF_no_out（节点号输出）	该参数为指令执行时的节点号输出	WORD	1~16、18
Done	该参数输出为 TRUE 时表示指令执行完成	BOOL	TRUE 或 FALSE
Busy（执行中）	该参数输出为 TRUE 时表示指令正在执行中	BOOL	TRUE 或 FALSE
Active（控制中）	该参数输出为 TRUE 时表示指令正在控制轴	BOOL	TRUE 或 FALSE
Error	该参数为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE
ErrorID	指令执行出错时的错误代码	WORD	-

➢ 功能说明

1、给定节点号（包括 Master 站号），先触发节点状态，再触发 Excute（上升沿输入），可写入该网络节点对应设备的网络状态，写入完成后，输出位 Done 变为 Ture；

2、NMT_Start_Node（启动节点）、NMT_Stop_Node（停止节点）、NMT_Enter_Preoperational（网络进入预操作模式）、NMT_Reset_Node（复位节点）、NMT_Reset_Communication（复位通信），以上任意两个输入不能同时触发；

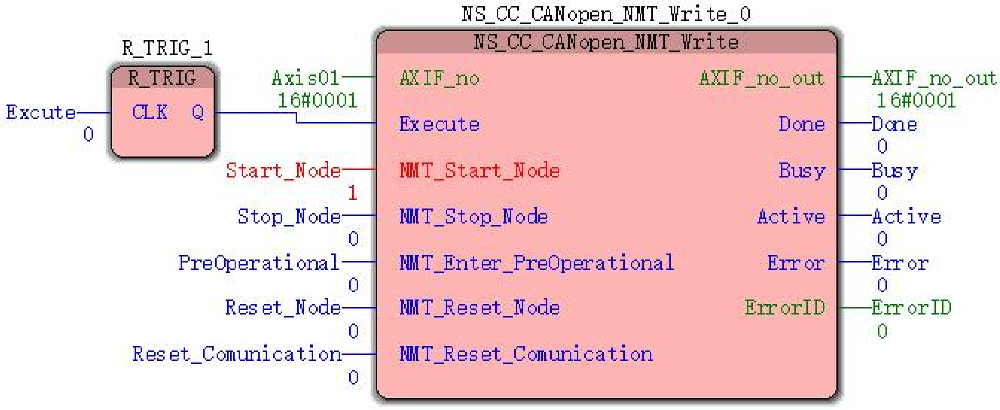
3、触发该指令必须用边沿模块触发，Excute 不能一直为 TURE；当 Done 信号输出，表示所选网络状态写入完成。

程序范例

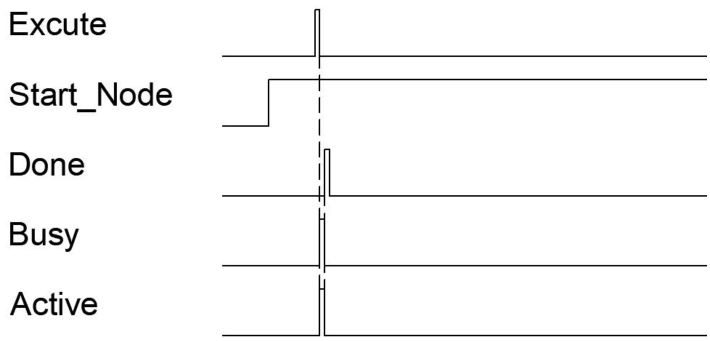
NS_CC_CANopen_NMT_Read 指令单独执行时的范例如下所示。

1、变量和程序

变量名	数据类型	初始值
NS_CC_CANopen_NMT_Read_0	NS_CC_CANopen_NMT_Read	
AXIF18	USINT	1
Excute	BOOL	FALSE
Start_Node	BOOL	FALSE
Stop_Node	BOOL	FALSE
PreOperational	BOOL	FALSE
Reset_Node	BOOL	FALSE
Reset_Comunication	BOOL	FALSE
AXIF_no_out	WORD	
Done	BOOL	
Busy	BOOL	
Active	BOOL	
Error	BOOL	
ErrorID	USINT	



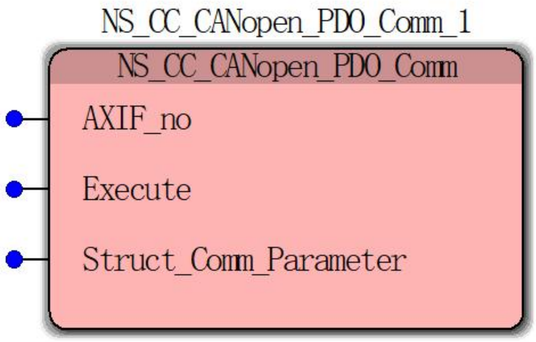
2、时序图



如图，给定一种操作模式，比如 Start_Node 接通，触发模块 Excute，主站会将节点 1 的网络进行一次启动操作，操作成功输出“Done”信号。

3.1.3 NS_CC_CANopen_PDO_Comm（配置 PDO 过程数据通信参数）

FB/FC	说明	适用机种
FB	此指令用于配置 PDO 过程数据通信参数	



➢ **输入参数**

名称	功能	数据类型	设定范围（缺省值）	生效时机
AXIF_no（节点号）	设定指令欲控制的节点	WORD	1~16	Excute 为 TRUE 时
Excute（执行位）	当 Excute 为 True 时，执行该指令	BOOL	TRUE 或 FALSE(FALSE)	Excute 为 TRUE 时
Struct_Comm_Parameter（结构体参数）	结构体参数，格式见功能说明	ANY		Excute 为 TRUE 时

➢ **功能说明**

1、触发该指令必须用边沿模块触发，Excute 不能一直为 TURE；

2、“Struct_Comm_Parameter”为结构体数据，其内容按照 DS301 协议规定的内容进行定义，定义完成后触发模块，即将通信参数写入对应从站的对象字典中，通信参数对应的对象字典为“0X1400H~0X14FF”或“0X1800H~0X18FF”，“CommPara”结构体数据结构如下，协议定义固定格式勿做更改。

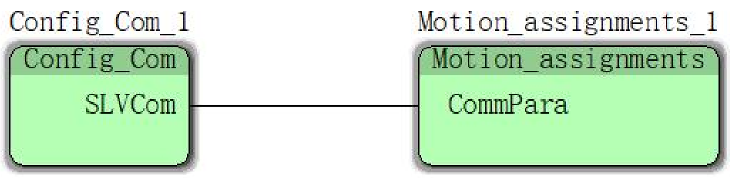
```
TYPE
  AIXComm:
    STRUCT
        Index:UINT ;
        Num_Of_SubIndex:BYTE ;
        CobID:UDINT ;
        Transmission_Type:BYTE ;
        Inhibit_Time:UINT ;
        Compatibility_Entry:BYTE ;
        Event_Timer:UINT ;
    END_STRUCT;
END_TYPE
```

程序范例

NS_CC_CANopen_PDO_Comm 指令单独执行时的范例如下所示。

1、变量和程序

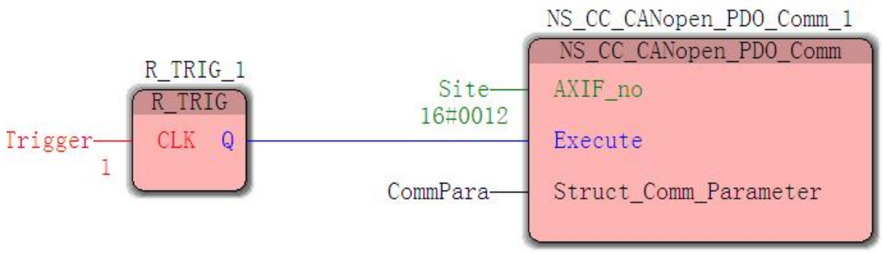
变量名	数据类型	初始值
Config_Com_1	Config_Com	
Motion_assignments_1	Motion_assignments	
R_TRIG_1	R_TRIG	
NS_CC_CANopen_PDO_Comm_1	NS_CC_CANopen_PDO_Comm	
SLVCom	VAR_OUTPUT	
CommPara	VAR_INPUT	



Config_Com_1 配置如下：

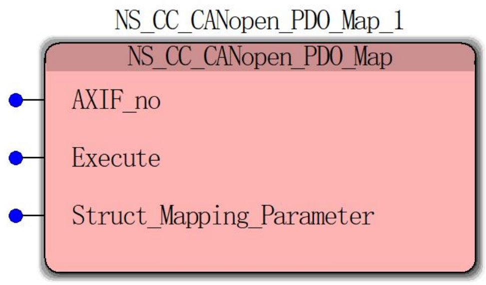
```
 TRUE |IF Trigger = TRUE THEN (*配置主站对从站的RPDO通信参数*)
 6144 |SLVCom.Index := UINT#16#1800;
16#02 |SLVCom.Num_Of_SubIndex := INT_TO_BYTE (2);
  385 |SLVCom.CobID := UDINT#16#180 + WORD_TO_UDINT(AXIF_no+WORD#1);
16#01 |SLVCom.Transmission_Type := INT_TO_BYTE (1);
    0 |SLVCom.Inhibit_Time := INT_TO_UINT (0);
16#00 |SLVCom.Compatibility_Entry := INT_TO_BYTE (0);
    0 |SLVCom.Event_Timer :=INT_TO_UINT (0);
```

Motion_ assignment_1 配置如下：



3.1.4 NS_CC_CANopen_PDO_Map（配置 PDO 过程数据参数映射）

FB/FC	说明	适用机种
FB	此指令用于配置 PDO 过程数据参数映射	



➢ **输入参数**

名称	功能	数据类型	设定范围（缺省值）	生效时机
AXIF_no（节点号）	设定指令欲控制的节点	WORD	1~16	Excute 为 TRUE 时
Excute（执行位）	当 Excute 为 True 时，执行该指令	BOOL	TRUE 或 FALSE(FALSE)	
Struct_Mapping_Parameter（结构体参数）	结构体参数，格式见功能说明	ANY		Excute 为 TRUE 时

➢ **功能说明**

1、触发该指令必须用边沿模块触发，Excute 不能一直为 TURE；

2、“Struct_Mapping_Parameter”为结构体数据，其内容按照 DS301 协议规定的内容进行定义，定义完成后出发模块，即将通信参数写入对应从站的对象字典中，通信参数对应的对象字典为“0X1600H~0X16FF”或“0X1A00H~0X1AFF”，“MapPara”结构体数据结构如下，协议定义固定格式勿做更改。

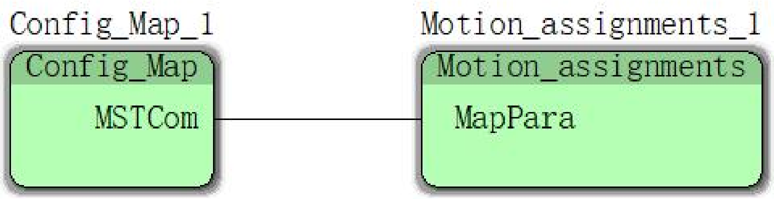
```
TYPE
    AIXMap:
    STRUCT
        Index:UINT;
        Num_Of_SubIndex:BYTE;
        SubIndex_Mapping_1:UDINT;
        SubIndex_Mapping_2:UDINT;
        SubIndex_Mapping_3:UDINT;
        SubIndex_Mapping_4:UDINT;
        SubIndex_Mapping_5:UDINT;
        SubIndex_Mapping_6:UDINT;
        SubIndex_Mapping_7:UDINT;
        SubIndex_Mapping_8:UDINT;
    END_STRUCT;
END_TYPE
```

程序范例

NS_CC_CANopen_PDO_Map 指令单独执行时的范例如下所示。

1、变量和程序

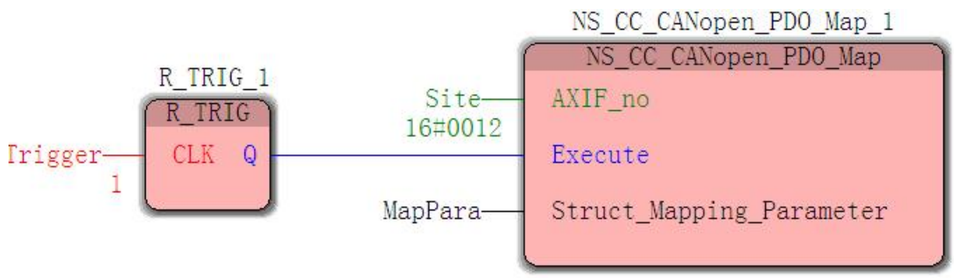
变量名	数据类型	初始值
Config_Map_1	Config_Map	
Motion_assignments_1	Motion_assignments	
R_TRIG_1	R_TRIG	
NS_CC_CANopen_PDO_Map_1	NS_CC_CANopen_PDO_Map	
MSTCom	VAR_OUTPUT	
MapPara	VAR_INPUT	



Config_Map_1 配置如下：

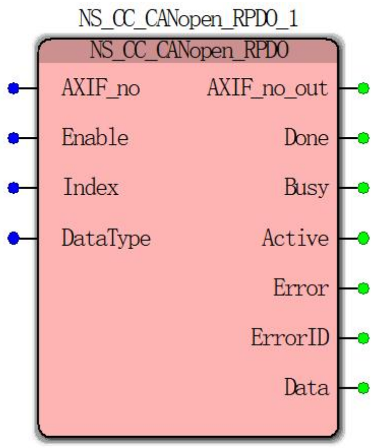
```
  TRUE |IF Trigger = TRUE THEN
  5120 |MSTCom.Index := UINT#16#1400 + WORD_TO_UINT(AXIF_no * WORD#2) ;
 16#02 |MSTCom.Num_Of_SubIndex := INT_TO_BYTE (2);
   385 |MSTCom.CobID := UDINT#16#180 + WORD_TO_UDINT(AXIF_no+WORD#1);
 16#01 |MSTCom.Transmission_Type := INT_TO_BYTE (1);
     0 |MSTCom.Inhibit_Time := INT_TO_UINT (0);
 16#00 |MSTCom.Compatibility_Entry := INT_TO_BYTE (0);
     0 |MSTCom.Event_Timer :=INT_TO_UINT (0);
```

Motion_assignments_1 配置如下：



3.1.5　NS_CC_CANopen_RPDO（PDO 数据映射区读取指令）

FB/FC	说明	适用机种
FB	此指令用于 PDO 数据映射区读取指令	



➢ **输入参数**

名称	功能	数据类型	设定范围（缺省值）	生效时机
AXIF_no（节点号）	设定指令欲控制的节点	WORD	1~16	Enable 为 TRUE 时
Enable（执行位）	当 Enable 为 True 时，执行该指令	BOOL	TRUE 或 FALSE(FALSE)	
Index（索引）	数据映射区索引	WORD	0000~FFFF	Enable 为 TRUE 时
DataType	数据类型	WORD	0000~FFFF	Enable 为 TRUE 时

➢ **输出参数**

名称	功能	数据类型	输出范围
AXIF_no_out（节点号输出）	该参数为指令执行时的节点号输出	WORD	0~16
Done（执行完成）	该参数输出为 TRUE 时表示指令执行完成	BOOL	TRUE 或 FALSE
Busy（执行中）	该参数输出为 TRUE 时表示指令正在执行中	BOOL	TRUE 或 FALSE
Active（控制中）	该参数输出为 TRUE 时表示指令正在控制轴	BOOL	TRUE 或 FALSE
Error（报错输出位）	该参数为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE
ErrorID（报错代码）	指令执行出错时的错误代码	WORD	0000~FFFF
Data（数据内容）	数据输出内容	WORD	0000~FFFF

➢ **功能说明**

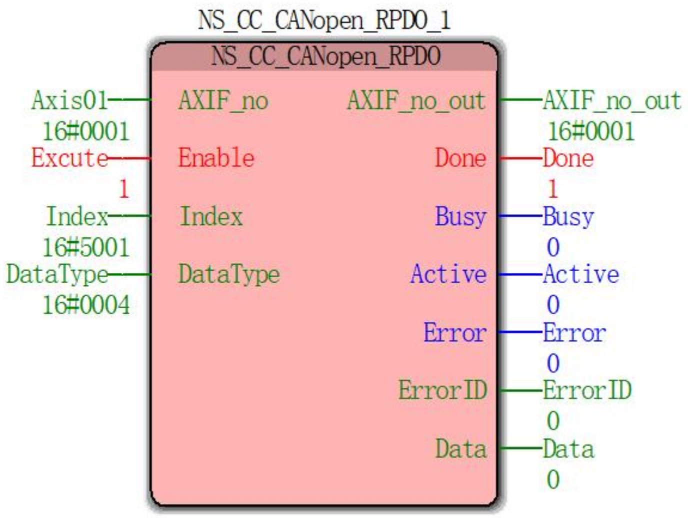
此模块的功能为，读取映射在某一从站节点 TPDO 数据映射区某一子索引所存放的地址数据的数据内容，当对应的状态读取成功后，输出位 Done 变为 Ture。

程序范例

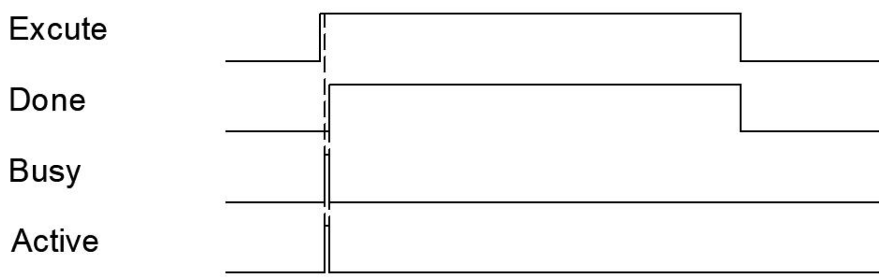
NS_CC_CANopen_RPDO 指令单独执行时的范例如下所示。

1、变量和程序

变量名	数据类型	初始值
NS_CC_CANopen_RPDO_1	NS_CC_CANopen_RPDO	
AXIF01	USINT	1
Excute	BOOL	FALSE
Index	WORD	16#5001
DataType	WORD	16#0004
AXIF_no_out	WORD	
Done	BOOL	
Busy	BOOL	
Active	BOOL	
Error	BOOL	
ErrorID	USINT	
Data	UDINT	



2、时序图



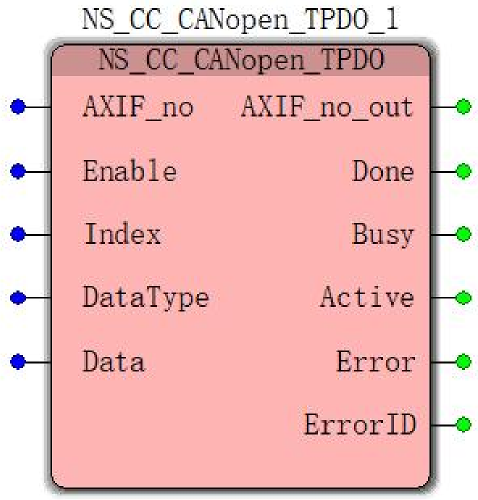
如图，0X60FF 映射在 TPDO 的数据映射区，索引为 0X5001（主站定义），那么“Index=16#5001”，　“DataType”所定义的类型如下：

```
数据类型表示：
02 signed8
03 signed16
04 signed32
05 unsigned8
06 unsigned16
07 unsigned32
```

因为 0X60FF 为 DINT 型，所以“DataType=16#04”，触发模块读取对应的轮廓速度的设定值。

3.1.6 NS_CC_CANopen_TPDO（PDO 数据映射区赋值指令）

FB/FC	说明	适用机种
FB	此指令用于 PDO 数据映射区赋值	



➢ 输入参数

名称	功能	数据类型	设定范围（缺省值）	生效时机
AXIF_no（节点号）	设定指令欲控制的节点	WORD	1~16	Enable 为 TRUE 时
Enable（执行位）	当 Enable 为 True 时，执行该指令	BOOL	TRUE 或 FALSE(FALSE)	Enable 为 TRUE 时
Index（索引）	数据映射区索引	WORD	0000~FFFF	Enable 为 TRUE 时
DataType（数据类型）	数据类型	WORD	0000~FFFF	Enable 为 TRUE 时
Data（数据内容）	数据内容	WORD	0000~FFFF	Enable 为 TRUE 时

➢ 输出参数

名称	功能	数据类型	输出范围
AXIF_no_out（节点号输出）	该参数为指令执行时的节点号输出	WORD	1~16
Done（执行完成）	该参数输出为 TRUE 时表示指令执行完成	BOOL	TRUE 或 FALSE
Busy（执行中）	该参数输出为 TRUE 时表示指令正在执行中	BOOL	TRUE 或 FALSE
Active（控制中）	该参数输出为 TRUE 时表示指令正在控制轴	BOOL	TRUE 或 FALSE

Error（报错输出位）	该参数为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE
ErrorID（报错代码）	指令执行出错时的错误代码	WORD	

➢ **功能说明**

1、此模块的功能为，向某一从站节点 RPDO 数据映射区某一子索引所存放的地址数据进行赋值；

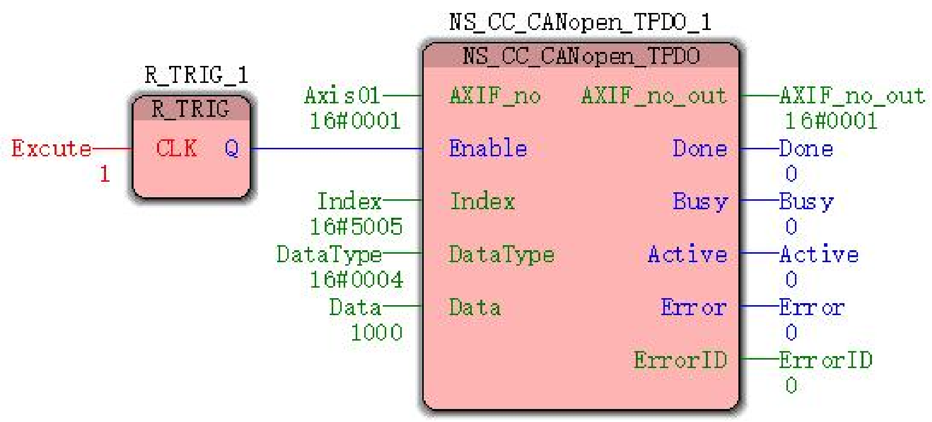
2、触发该指令必须用边沿模块触发，Excute 不能一直为 TURE。

程序范例

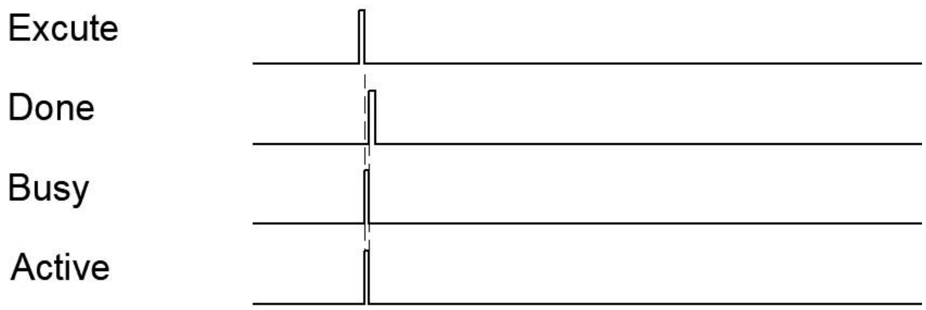
NS_CC_CANopen_TPDO 指令单独执行时的范例如下所示。

1、变量和程序

变量名	数据类型	初始值
NS_CC_CANopen_TPDO_1	NS_CC_CANopen_TPDO	
AXIF01	USINT	1
Excute	BOOL	FALSE
Index	WORD	16#5001
DataType	WORD	16#0004
Data	UDINT	1000
AXIF_no_out	WORD	
Done	BOOL	
Busy	BOOL	
Active	BOOL	
Error	BOOL	
ErrorID	USINT	



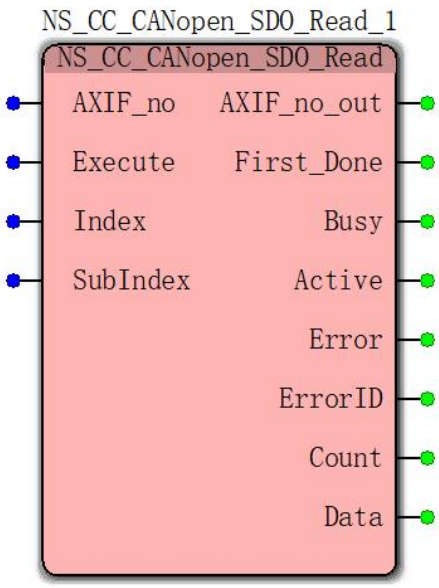
2、时序图



如图，0X60FF 映射在 RPDO 的数据映射区，索引为 0X5005（主站自定义），那么“Index=16#5005”，因 0X60FF 为 DINT 型，所以“DataType=16#04”，“Data=1000”，触发模块将 1000 赋值给节点地址 0X60FF，表示此节点轮廓速度当前的设定值设定为 1000。

3.1.7 NS_CC_CANopen_SDO_Read（服务数据读取指令）

FB/FC	说明	适用机种
FB	此指令用于读取服务数据	



➢ 输入参数

名称	功能	数据类型	设定范围（缺省值）	生效时机
AXIF_no（节点号）	设定指令欲控制的节点	WORD	1~16	Excute 为 TRUE 时
Excute（执行位）	当 Excute 为 True 时，执行该指令	BOOL	TRUE 或 FALSE(FALSE)	
Index（索引）	节点地址的索引	WORD	0~FFFF	Excute 为 TRUE 时
SubIndex（子索引）	节点地址的子索引	WORD	0~FFFF	Excute 为 TRUE 时

➢ 输出参数

名称	功能	数据类型	输出范围
AXIF_no_out（节点号输出）	该参数为指令执行时的节点号输出	WORD	0~16
First_Done（第一次执行完成）	该参数输出为 TRUE 时表示指令第一次执行完成，再次触发该参数仍为 Ture	BOOL	TRUE 或 FALSE
Busy（执行中）	该参数输出为 TRUE 时表示指令正在执行中	BOOL	TRUE 或 FALSE
Active（控制中）	该参数输出为 TRUE 时表示	BOOL	TRUE 或 FALSE

	指令正在控制轴		
Error（报错输出位）	该参数为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE
ErrorID（报错代码）	指令执行出错时的错误代码	WORD	
Count	字节长度	WORD	0000~FFFF
Data	输出数据	UDINT	0000~FFFF

➢ **功能说明**

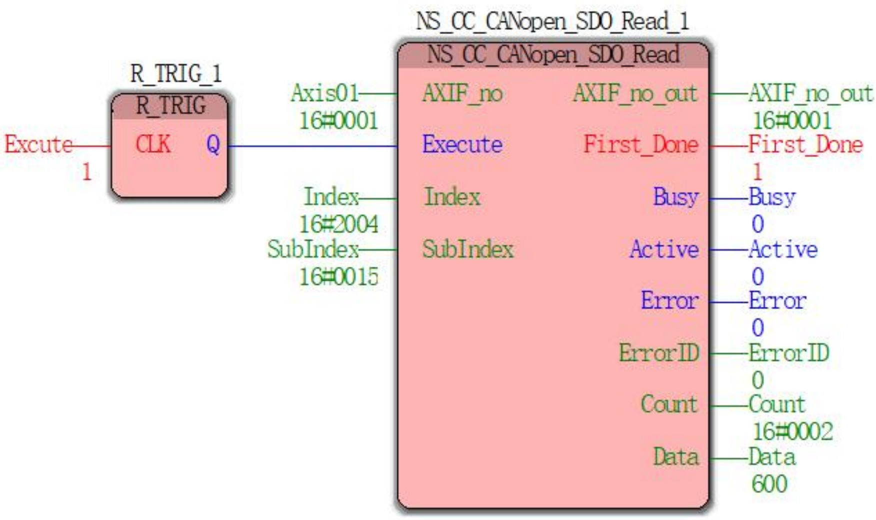
服务数据读取模块，通过指定某节点地址的索引和子索引的方式直接读取地址内容，有问有答，读取速慢，通常一次读取耗时为两个同步周期，应用于非实时数据的读取操作。

程序范例

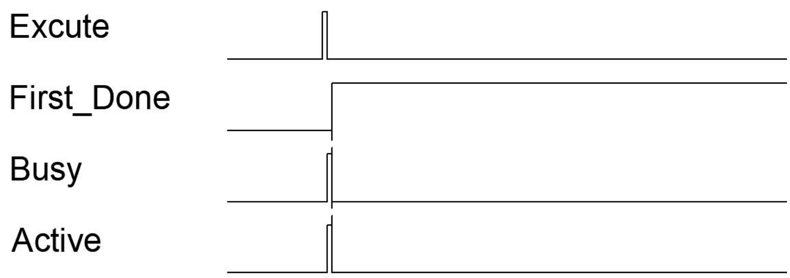
NS_CC_CANopen_SDO_Read 指令单独执行时的范例如下所示。

1、变量和程序

变量名	数据类型	初始值
NS_CC_CANopen_SDO_Read_1	NS_CC_CANopen_SDO_Read	
AXIF01	USINT	1
Excute	BOOL	FALSE
Index	WORD	16#2004
SubIndex	WORD	16#0015
AXIF_no_out	WORD	
Done	BOOL	
Busy	BOOL	
Active	BOOL	
Error	BOOL	
ErrorID	USINT	
Count	WORD	
Data	UDINT	



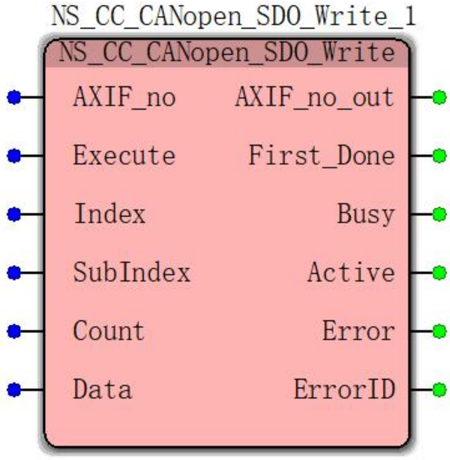
2、时序图



如图，读取 Vector 伺服参数 P04.21（当前转速，单位 r/min）时，对应的索引为：16#2000+16#4=16#2004，子索引为 21 对应 16#15，触发模块读取当前 1 号节点伺服转速为 600r/min。

3.1.8 NS_CC_CANopen_SDO_Write（服务数据赋值指令）

FB/FC	说明	适用机种
FB	此指令用于赋值服务数据	



➢ **输入参数**

名称	功能	数据类型	设定范围（缺省值）	生效时机
AXIF_no（节点号）	设定指令欲控制的节点	WORD	1~16	Excute 为 TRUE 时
Excute（执行位）	当 Excute 为 True 时，执行该指令	BOOL	TRUE 或 FALSE(FALSE)	
Index（索引）	节点地址的索引	WORD		
SubIndex（子索引）	节点地址的子索引	WORD		
Count	字节长度	WORD		
Data	数据输入	UDINT		

➢ **输出参数**

名称	功能	数据类型	输出范围
AXIF_no_out（节点号输出）	该参数为指令执行时的节点号输出	WORD	0~16
Done（执行完成）	该参数输出为 TRUE 时表示指令执行完成	BOOL	TRUE 或 FALSE
Busy（执行中）	该参数输出为 TRUE 时表示指令正在执行中	BOOL	TRUE 或 FALSE
Active（控制中）	该参数输出为 TRUE 时表示指令正在控制轴	BOOL	TRUE 或 FALSE
Error（报错输出位）	该参数为 TRUE 时表示指令的执行出错	BOOL	TRUE 或 FALSE
ErrorID（报错代码）	指令执行出错时的错误代码	WORD	-

➢ **功能说明**

1、服务数据赋值模块，通过指定某节点地址的索引和自索引的方式直接给地址赋值，有问有答，赋值速慢，通常一次读取耗时为两个同步周期，应用于非实时数据的写入操作。

2.、触发该指令必须用边沿模块触发，Excute 不能一直为 TURE；

3、与读取模块的不同点在于，多了一个字节变量“Count”（字节长度）和 “Data” （数据变量），字节长度的定义如下：

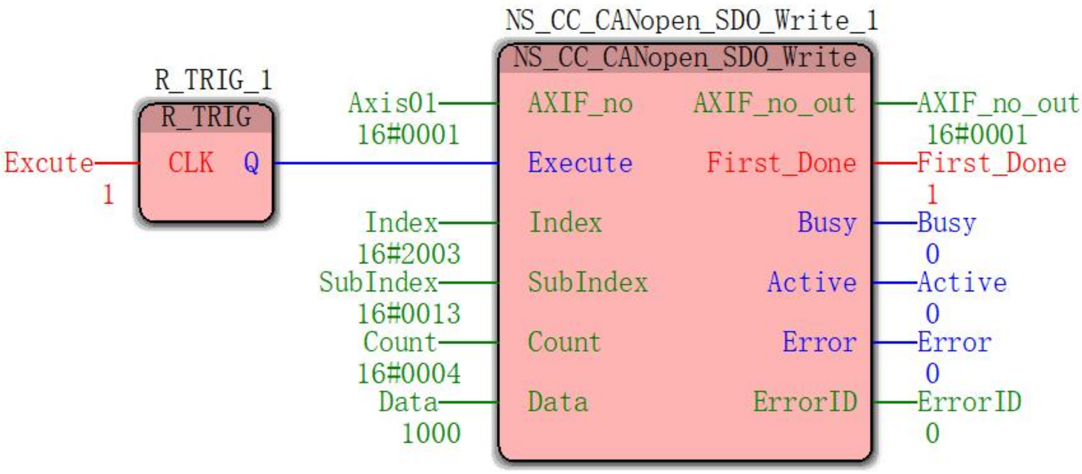
数据字节长度表示：
1,2,3,4 COUNT数

程序范例

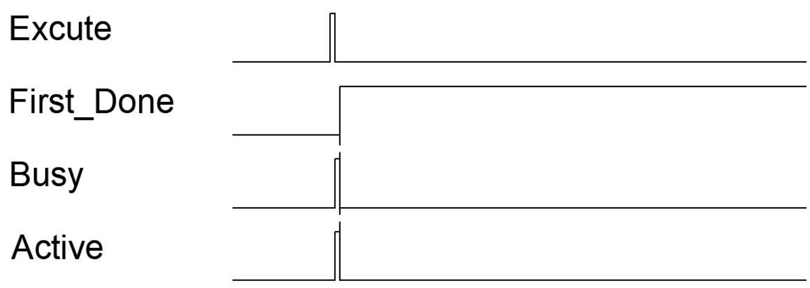
NS_CC_CANopen_SDO_Write 指令单独执行时的范例如下所示。

1、变量和程序

变量名	数据类型	初始值
NS_CC_CANopen_SDO_Write_1	NS_CC_CANopen_SDO_Write	
AXIF01	USINT	1
Excute	BOOL	FALSE
Index	WORD	16#2004
SubIndex	WORD	16#0015
Count	WORD	
Data	UDINT	
AXIF_no_out	WORD	
Done	BOOL	
Busy	BOOL	
Active	BOOL	
Error	BOOL	
ErrorID	USINT	



2、时序图



如上图，赋值 Vector 伺服参数 P03.19（位置误差过大值）时，对应的索引为：16#2000+16#3=16#2004，子索引为 19 对应 16#13，字节长度为 4，触发模块赋值当前 1 号节点的位置过大值。

3.2 运动控制模块

a.当伺服参数 P08.42=0 时，默认为威科达品牌伺服
VEC 运动控制器总线型支持的模块有 ：
MC_AXIS_REF
MC_Power
MC_CamIn
MC_CamOut
MC_CombineAxes
MC_GearIn
MC_GearOut
MC_Halt
MC_Home
MC_MoveAbsolute
MC_MoveAdditive
MC_MoveRelative
MC_MoveVelocity
MC_Stop
NS_MC_StopByPos
MC_SpecialMoveAbsolute
NS_MC_RotaryCutIn
NS_MC_SpecialCamIn
NS_MC_SpecialCombineAxes
MC_HaltSuperimposed
MC_MoveSuperimposed
MC_Phasing
NS_MC_Jog
MC_SetOverride
MC_SetPosition
MC_TouchProbe
MC_AbortTrigger
NS_MC_CamReadPoint
NS_MC_CamReadTappetStatus
NS_MC_CamReadTappetValue
NS_MC_CamSet
NS_MC_CamWritePoint
NS_MC_CamWriteTappetValue
MC_ReadActualPosition
MC_ReadActualVelocity
MC_ReadMotionState
MC_ReadStatus
NS_MC_ReadParameter

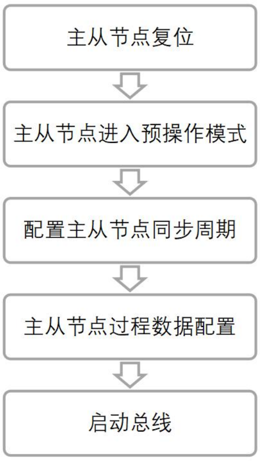
MC_Reset

共 39 个与运动控制相关的模块，最大控制 16 轴。
总线型运动控制器第一版是在模拟量版本的基础上增加了 CANopen 协议层而推出的一款升级版产品，本产品运动控制功能模块的用法基本与模拟量产品一致，模块功能的说明不再赘述，请参考《VA 系列运动控制器编程手册》一书详细了解。

b.当伺服参数 P08.42=1 时，默认为台达 A2 系列 CANopen 型伺服
通过台达 DVP15MC11T 运动控制器 Motion 测试，VC 总线伺服适用于台达控制器的大部分应用功能。Motion 口不能使用的功能共有六条：
MC_SetPosition（位置设置指令）；
MC_ReadAxisError（读取轴错误指令）；
MC_TouchProbe（位置捕获指令,当捕捉伺服位置时因 DI 端子定义不同而不能使用）；
DMC_NC(G 代码解析指令）；
DMC_ControlAxisByPos(NC 移位指令);
主要功能除 NC 功能不能用之外，其他未列出来的功能经过测试，都可使用。

4. 配置实例

介绍总线运动控制器使用 Vector 的 CANopen 型伺服时的软件配置过程,配置流程如下:



4.1 运动控制轴配置

主要工作是配置 PDO 过程数据，网络同步周期，下面通过二次封装后的模块逐步介绍运动轴的配置过程，本配置过程配置一台伺服，节点号为 1。

4.1.1 通信配置

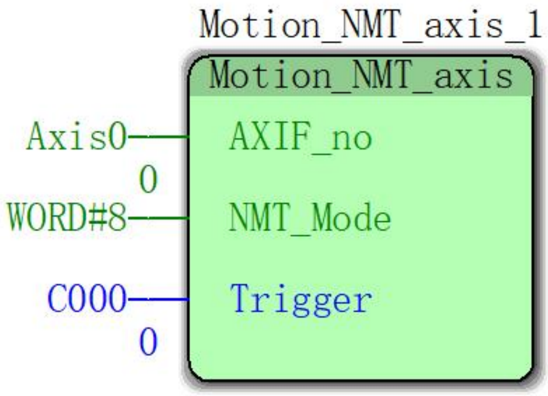
按照硬件拓扑结构搭建好网络后，开始建立通信：

1、从站设置，设置伺服 P08.41=1（节点号） ，P08.40=800（波特率）；

2、主站设置，运动控制器主站站号默认为 18，寄存器地址为：%MB3.4012,通信波特率寄存器地址为：%MW3.4013，按如下设置

变量名	数据类型	初始值	地址
BaudRate	WORD	800	
Com_ BaudRate	WORD		%MW3.4013
MainSite	Byte	18	
Com_ MainSite	Byte		%MB3.4012

4.1.2 主从节点复位



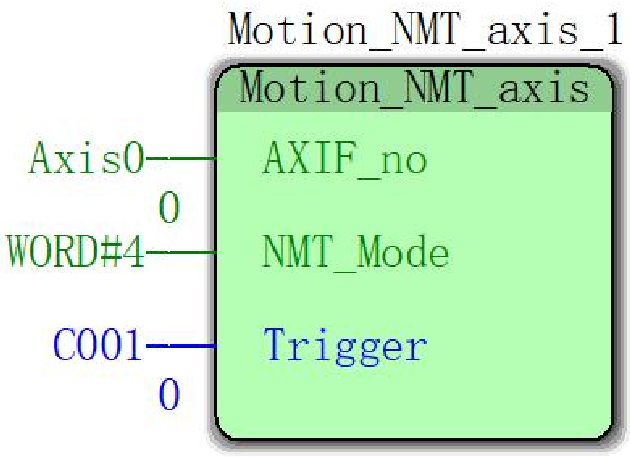
➢ **输入参数**

变量名	功能	数据类型	初始值
Axis0（轴号）	节点号=轴号+1	USINT	0
C000（执行位）	当 Execute 由 FALSE 变为 TRUE，执行该指令	BOOL	FALSE

1、NMT_Mode（网络模式）是可自定义的，模版定义为：1（启动网络）、2（停止网络）、4（网络进入预操作）、8（复位节点）、16（复位通信），此处选择 NMT_Mode=8，复位网络；

2、模块内部有两个 NS_CC_CANopen_NMT_Write 功能模块，一个对主站 18 进行复位，一个对从站节点 1 号进行复位，条件“C000”触发，延时等待大概 1s 的时间，然后 转到下一步。（详细见配置模版“Vector CANopen Configuration”）

4.1.3 主从节点进入预操作模式



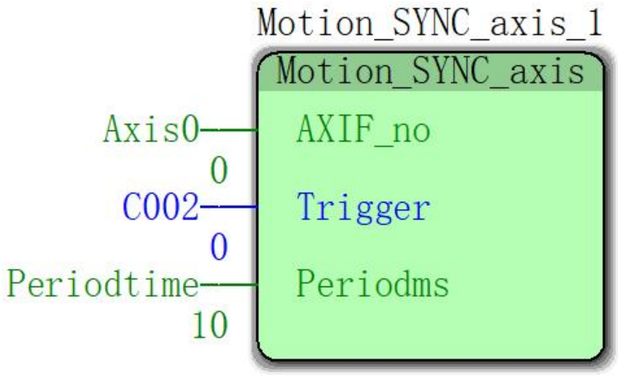
➢ **输入参数**

变量名	功能	数据类型	初始值
Axis0（轴号）	节点号=轴号+1	USINT	0
C001（执行位）	当 Execute 由 FALSE 变为 TRUE，执行该指令	BOOL	FALSE

1、NMT_Mode（网络模式）是可自定义的，模版定义为：1（启动网络）、2（停止网络）、4（网络进入预操作）、8（复位节点）、16（复位通信），此处选择 NMT_Mode=4，网络进入预操作；

2、模块内部有两个 NS_CC_CANopen_NMT_Write 功能模块，一个对主站 18 进行预操作，一个对从站节点 1 号进行预操作，条件“C001”触发，同样延时等待大概也是 1s 的时间，然后转到下一步。（详细见配置模版“Vector CANopen Configuration”）

4.1.4 配置主从节点同步周期



➢ **输入参数**

变量名	功能	数据类型	初始值
Axis0（轴号）	节点号=轴号+1	USINT	0
C002（执行位）	当 Execute 由 FALSE 变为 TRUE，执行该指令	BOOL	FALSE
Periodtime（同步周期）	设置主从节点的同步周期（单位：ms）	Time	10

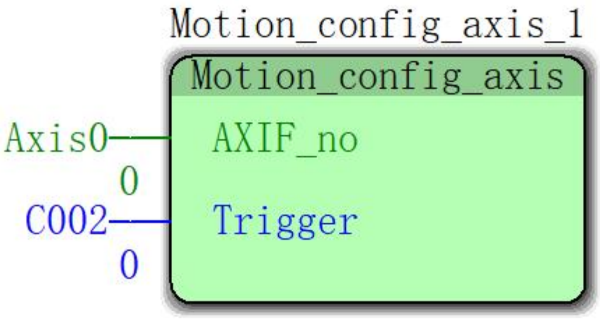
1、在主从站均进入预操作模式情况下，在此步进行同步周期的配置，模块内部是用 NS_CC_CANopen_SDO_Write 模块分别对主从站的 1006H 对象进行设置，主从站的同步周期设置必须一致，否则导致控制失误！

2、条件 C002 触发后，配置好同步周期即可进入下一步，无需延时；

3、设定 1006H 同步周期的参考标准：

Periodtime =（0.114*1.3*PDO 数量*1000/波特率+1+轴数*0.125），模版配置的 PDO 数量为 4；（详细见配置模版“Vector CANopen Configuration”）

4.1.5 主从节点过程数据配置

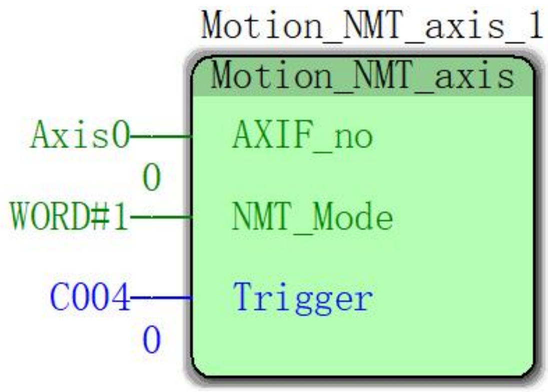


➢ **输入参数**

变量名	功能	数据类型	初始值
Axis0（轴号）	节点号=轴号+1	USINT	0
C002（执行位）	当 Execute 由 FALSE 变为 TRUE，执行该指令	BOOL	FALSE

此步骤是对 PDO 数据进行通信参数和映射参数的配置，必须按照伺服插值位置模式所要求的对象和格式进行配置，否则无法使用运动控制模块功能，此步的配置需要认真估算配置所消耗的时间进行延时，延时时间太短会使有些轴配置不成功，导致无法控制，时间太长影响效率。（详细见配置模版“Vector CANopen Configuration”）

4.1.6 启动总线



➢ **输入参数**

变量名	功能	数据类型	初始值
Axis0（轴号）	节点号=轴号+1	USINT	0
C000（执行位）	当 Execute 由 FALSE 变为 TRUE，执行该指令	BOOL	FALSE

1、NMT_Mode（网络模式）是可自定义的，模版定义为：1（启动网络）、2（停止网络）、4（网络进入预操作）、8（复位节点）、16（复位通信），此处选择 NMT_Mode=1，启动网络；

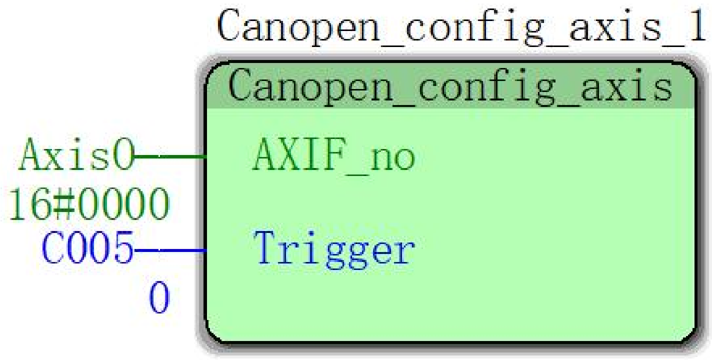
2、模块内部有两个 NS_CC_CANopen_NMT_Write 功能模块，分别对主站 18 和从站节点 1 号进行启动，条件“C004”触发，使总线进入运行模式，之后就可使用运动控制模块对轴进行控制了。（详细见配置模版“Vector CANopen Configuration”）

若有兴趣对配置过程进行更详细的了解，请参考配置模版程序。

4.2 张力控制轴配置

4.2.1 通信配置----------------同 4.1.1 通信配置
4.2.2 主从节点复位----------------同 4.1.2 主从节点复位
4.2.3 主从节点进入预操作模式----------------同 4.1.3 主从节点进入预操作模式
4.2.4 配置主从节点同步周期----------------同 4.1.4 配置主从节点同步周期
4.2.6 启动总线----------------同 4.1.6 启动总线

4.2.5 主从节点过程数据配置



5.关键注意事项

a. 设定 1006H 同步周期的参考标准：同步周期=（0.114*1.3*PDO 数量*1000/波特率+1+轴数*0.125）ms；

b. 设置软件 CYCLETIME,程序扫描周期 =（同步周期 +1 或 2）ms；

c. 配置数据的时间：张力轴耗时 =26 条*同步周期*2ms; 运动轴 =12 条*同步周期*2ms；

d. 总线 JITTER *3.56/1000 = 同步周期 ms；

e. 电子齿轮的速度来源%MB3.9690；CANopen 通信波特率%MW3.4013；

f. 运动控制器若绿灯闪烁，请检查控制器及各站的通信站号及波特率设置是否正确，或者检查线路是否断线，或者检查终端电阻有没有接，或者检查信号地有没有连通在一起，或者检查伺服是否接地良好；

g. 程序模块（除了没有被封装的总线的读、写、配置模块）轴号均比对应的站号少 1，例如模块为 0 轴，实际对站号 1 进行操作；

h.调用了没有被封装的总线的读、写、配置模块时，模块与轴号一一对应，1 轴对应 1 站以此类推；

j.主轴的站号默认为 1,即作为运动轴且轴号为 1 时，此节点可做主轴，张力轴都读取站号 1 的速度值存入 P14.63 用于张力控制使用，数据不经过主站，从发从收；

i.系统要求接地可靠，预防干扰。

附件五 寄存器说明

寄存器分类：

1、 0-1999 掉电不保存；
2、 2000-3999 掉电保存 ；
3、 4000-4095 掉电保存的特殊寄存器 ；
4、 4096-9499 掉电不保存 ；
5、 9500-9999 掉电不保存的特殊寄存器 ；
6、 10000-19999 掉电不保存。

特殊寄存器说明：

<table>
<tr><td>%MB3.4010</td><td>MODBUS 通讯站号</td></tr>
<tr><td>%MB3.4011</td><td>MODBUS 通讯波特率波特率为 4800*%MB3.4011 里的值</td></tr>
<tr><td>%MB3.4012</td><td>CAN 通讯站号</td></tr>
<tr><td>%MW3.4013</td><td>CAN 通讯波特率，例如 500K 设置为 500</td></tr>
<tr><td>%MB3.4015</td><td>EtherNET IP 地址最后一位 192.168.1.%MB3.4015 里的数 范围 93-124</td></tr>
<tr><td>%MB3.4016</td><td>MODBUS 数据长度 7 七位 8 八位</td></tr>
<tr><td>%MB3.4017</td><td>MODBUS 奇偶校验 0 偶 1 奇 2 无</td></tr>
<tr><td>%MB3.4018</td><td>MODBUS 停止位 0 为一位停止位 1 为二位停止位</td></tr>
<tr><td>%MB3.4019</td><td>为 0 MODBUS-232 通讯首地址人机 0 对应 VA 的共享内存地址 15000 也就是%MX3.15000
为 1 MODBUS-232 和 Modbus485 操控同样的共享内存地址。</td></tr>
<tr><td>%MB3.4020</td><td>MODBUS 通讯延时 设置范围为 2-100（单位：毫秒）；默认 2 毫秒</td></tr>
<tr><td>%MW3.4021</td><td>周期时间，单位：微秒（默认为 1000 微秒）</td></tr>
<tr><td>%MW3.4023</td><td>各轴绝对值编码器正反向；0-4 轴对应 bit 0-4；相应位 0 为正常计数，1 为反向计数</td></tr>
<tr><td>%MB3.4024</td><td>此特殊寄存器设 1 时 VA 运动控制器作主站，设为 0 为从站。重上电后有效；</td></tr>
<tr><td>%MB3.9536</td><td>默认，保留不动</td></tr>
<tr><td>%MB3.9538</td><td>将寄存器区间 2000~3999、4000~4095 内所有寄存器的值赋为 0</td></tr>
<tr><td>%MB3.9542~
%MB3.9556</td><td>时钟寄存器，详见 11.6.10 RTC_S（时钟特殊寄存器）</td></tr>
<tr><td>%MB3.9654</td><td>编码器方向，不能与轴信息上的一起用。相应位设 1 则相应的轴的编码器反向</td></tr>
<tr><td>%MW3.9690</td><td>0-4 位设编码器来源
<table>
<tr><td>轴口</td><td>Bit4</td><td>Bit3</td><td>Bit2</td><td>Bit1</td><td>Bit0</td></tr>
<tr><td>4</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>2</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr>
<tr><td>3</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td></tr>
</table>
8-12 位设 DI 来源(16 表示 Z 点)</td></tr>
</table>

	13 位设边沿 边沿位为 0 表示 DI 外部信号由无效变有效
%MW3.9692	0-4 位设编码器来源 8-12 位设 DI 来源(16 表示 Z 点) 13 位设边沿 边沿位为 0 表示 DI 外部信号由无效变有效
%MD3.9694	两次信号之间的脉冲数(该寄存器其脉冲来源与 DI 采样信号由 MW3.9690 设置，需与 MW3.9690 配合使用)
%MD3.9704	两次信号之间的脉冲数(该寄存器其脉冲来源与 DI 采样信号由 MW3.9692 设置，需与 MW3.9692 配合使用)
%MB3.9698	0 为正常 1 为超诺
%MB3.9702	0 为 DO 过电流保护，1 为 DO 不保护
%MB3.9710	0 为关闭虚轴虚拟脉冲功能，1 为开启虚轴虚拟脉冲功能
%MB3.9711	指定生成虚轴虚拟脉冲的轴号
%MB3.9720	执行一次运算所需周期数，单位：个（默认为 2）
%MB3.9721	用于参与运算的轴数，该值最低为 1，最高为 16
%MB3.9722	在特殊凸轮指令下，轴相应的位设为 1 表示该轴在该功能下，减速段位置环增益自动无效
%MB3.9723	延续%MB3.9722 说明，在减速段结束后，延时%MB3.9723 个底层周期后恢复为增益有效
%MB3.9724	0-4 轴对应位（0-4 位）设为 1，发脉冲有补偿
%MB3.9800	为 1 选择手动选择扩展 为 0 代表自动识别扩展类型
%MB3.9801	第一个扩展的扩展类型（设 3 代表 8ID8DO ,设 4 代表 16DI, 设 5 代表 16DO , 设 6 代表模拟量,设 7 代表 PT100 ,设 8 代表称重 ，设 9 代表热电偶 ，设 10 代表物联）
%MB3.9802	第二个扩展的扩展类型，同 9801
%MB3.9803	第三个扩展的扩展类型，同 9801
%MB3.9804	第四个扩展的扩展类型，同 9801
%MB3.9805	第五个扩展的扩展类型，同 9801
%MB3.9806	第六个扩展的扩展类型，同 9801
%MB3.9807	第七个扩展的扩展类型，同 9801
%MB3.9808	第八个扩展的扩展类型，同 9801
%MB3.9809	VA 控制器硬件 0~3 轴口脉冲收发控制位，0~3 轴对应 bit 为 0 表示 4~7 轴收脉冲；0~3 轴对应 bit 为 1 表示 4~7 轴发脉冲； （例：将%MB3.9809 的值改为 0，即“0000”，4~7 轴收脉冲；将%MB3.9809 的值改为 15，即“1111”，4~7 轴发脉冲）
%MW3.9995	ARM 版本号
%MW3.9997	FPGA 版本号

附件六 错误代码

0x1001	轴号 Axis 设定超出允许范围
0x1002	加速度 Acceleration 设定超出允许范围
0x1003	减速度 Deceleration 设定超出允许范围
0x1004	加速度变化率 Jerk 设定超出允许范围
0x1005	速度 Velocity 设定超出允许范围
0x1006	位置 Position 设定超出允许范围
0x1007	方向 Direction 设定超出允许范围
0x1008	BufferMode 设定超出允许范围
0x1009	SetPosition 功能的 ReferenceType 设定超出允许范围
0x100b	电子凸轮凸轮表超出允许范围
0x100c	主轴轴号 Master 设定错误
0x100d	电子凸轮启动模式 StartMode 设定超出允许范围
0x100e	电子凸轮 MasterScaling 设定超出允许范围
0x100f	电子凸轮 SlaveScaling 设定超出允许范围
0x1010	主轴信息来源 MasterValueSource 设定超出允许范围
0x1011	主从轴轴号冲突
0x1012	电子齿轮分子 RatioNumerator 设定超出允许范围（≥0）
0x1013	电子齿轮分母 RatioDenominator 设定超出允许范围（>0）
0x1014	MC_SetOverride 功能的 VelFactor 参数设定超出允许范围（0~500）
0x1015	电子凸轮 SlaveRange 设定范围错误
0x1018	MC_TouchProbe 功能的 TriggerInput 设定超范围（0~15）
0x1019	MC_TouchProbe 功能的 Mode 设定错误
0x1021	RotaryAxisRadius 设定超出允许范围（>0）
0x1022	FeedAxisRadius 设定超出允许范围（>0）
0x1023	CutLength 设定超出允许范围（>0）
0x1026	SyncAngle 设定超出允许范围（0~360）
0x1027	旋切功能无此参数
0x1028	RotaryAxisKnifeNum 设定超出允许范围（1-16）
0x1045	NS_MC_SpecialCamIn 的 Mode 等于 1/2 时，MaterValueSource 不能为 0
	NS_MC_RotaryCutIn 的 mode 等于 2/3/4 时，MaterValueSource 不能为 0
0x1046	MC_AXIS_REF，Sample_Time 设定超出允许范围
0x1047	MC_AXIS_REF,Closed_Loop_Scaling 设定超出允许范围
0x1048	MC_AXIS_REF，Reductor_Den 设定超出允许范围（>0）

0x1049	MC_AXIS_REF，Reductor_Num 设定超出允许范围（＞0）
0x1050	MC_AXIS_REF，Screw_Lead 和 Disc_Circumference 设定超出允许范围（＞0）
0x1051	MC_AXIS_REF，Revolving_Axes 为 1 时 Modulo 设定超出允许范围（＞0）
0x1052	MC_AXIS_REF，ControlMode 设定超出允许范围
0x1053	MC_AXIS_REF，Moter_Max_V 设定超出允许范围
0x1054	MC_AXIS_REF，Moter_PPC 设定超出允许范围
0x1055	MC_AXIS_REF，Offset_Max_V 设定超出允许范围
0x1056	MC_CamIn，ActivationMode 为 2 时，ActivationPosition 小于 0 或大于模
	NS_MC_SpecialCamIn， ActivationPosition 设定超出允许范围（≥0）
0x1057	NS_MC_SpecialCamIn，DistanceOffset_Master 设定超出允许范围（≥0）
0x1058	DistanceAdd 设定超出允许范围（≥0）
0x1059	DistanceSync 设定超出允许范围（≥0）
0x1060	DistanceDec 设定超出允许范围（≥0）
0x1061	NC_CartesianCoordinate 模块 Depth 设定超出允许范围
0x1062	NC_CartesianCoordinate 模块 Junction_Deviation 设定超出允许范围
0x1063	NC_CartesianCoordinate 模块 Arc_Tolerance 设定超出允许范围
0x1064	NS_MC_SpecialCombineAxes 模块 Cam_DistanceOffset_Master 设定超出允许范围（＞0）
0x1068	NS_MC_SpecialCombineAxes 模块 Cam_Pulse_Per_Unit_M 设定超出允许范围（＞0）
0x1069	未找到 NS_MC_SpecialCombineAxes 模块 NCFile 指定文件
0x1070	NC_MoveCircular，CircMode 设定超出允许范围（0~2）
0x1071	NC_MoveCircular，PathChoice 设定超出允许范围（0~1）
0x1072	NC_MoveCircular 模块，Param_R、Param_I 、Param_J 、Param_K 全部为 0
0x1073	NC_GroupEnable，当轴 0/1/2 中存在轴的当前状态不为 Standstill，则不能使能轴组
0x1074	NC_MoveLinear/NC_MoveCircular/ NC_CartesianCoordinate 指令在执行时，未使用 NC_GroupEnable 使能轴组
0x1075	NC_GroupEnable 模块 Axis_Num_X 设定超出允许范围

	（0）
0x1076	NC_GroupEnable 模块 Axis_Num_Y 设定超出允许范围（1）
0x1077	NC_GroupEnable 模块 Axis_Num_Z 设定超出允许范围（2）
0x1078	NS_MC_RotaryCutIn,切长 CutLength 的设定小于刀周长的$\frac{1}{10}$
0x1079	NS_MC_RotaryCutIn，Cut_DI_Num 设定超出允许范围（0~15）
0x1080	NS_MC_RotaryCutIn，Mark_DI_Num 设定超出允许范围（0~15）
0x1081	MC_CombineAxes，CombineMode 设定超出允许范围（0~1）
0x1082	NS_MC_SpecialCombineAxes，Periodic_Master_Units 输入超出允许范围（>0）
0x1083	本指令被控轴的轴信息有错误
0x1084	本指令对应的主轴轴信息有错误
0X2001	MC_Power，伺服从站返回个主机的状态字，报从站有故障
0x2002	MC_Power，总线上有错误，例如总线干扰，波特率不对等
0x2003	NS_CC_CANopen_NMT_Read，读到没有意义的状态
0x2004	NS_CC_CANopen_NMT_Write，写的状态没有意义
0x2005	NS_CC_CANopen_SDO_Read,NS_CC_CANopen_SDO_Write，超过 buffer
0x2006	NS_CC_CANopen_SDO_Write 数据类型错误，只有 1,2,4
0x2007	NS_CC_CANopen_SDO_Read,NS_CC_CANopen_SDO_Write 从机回复超时
0x2008	NS_CC_CANopen_TPDO,NS_CC_CANopen_RPDO 索引越界
0x2009	NS_CC_CANopen_TPDO,NS_CC_CANopen_RPDO 类型错误，正常范围为 2,3,4,5,6,7
0x2401	该功能块的 Axis_no 设定超出允许范围(0~6)
0x2402	该功能块的 Active_Axis 设定超出允许范围
0x2403	CNT_ID 设定超出允许范围
0x2404	Event_ID 设定超出允许范围
0x2405	DI_ID 设定超出允许范围
0X4000	同一个轴使用同一个模块超过规定个数，允许使用个数范围请翻阅注意事项：
0X4001	NS_NC_ReadParameter 模块的 ParameterNumber 设定超出允许范围
0x4101	该轴当前运行状态为 ErrorStop 或 Disabled，无法执行任何运动指令。

0x4102	该轴当前运行状态为 Stopping，无法执行除 MC_Stopping 以外任何运动指令。
0x4103	执行 MC_Home 指令需要该轴处于 StandStill 状态
0x4104	MC_CamOut 模块只有在当前运行指令为 MC_CamIn 时才能运行
0x4105	MC_GearOut 模块只有在当前运行指令为 MC_GearIn 时才能运行
0x4106	该轴当前运行状态为 Homing，无法执行除 MC_Stopping 以外任何运动指令
0x4107	当前 BufferMode 缓存超出允许范围，详情请翻阅 BufferMode 缓存说明
0x4108	RunCommand 下，没有命令指针，属于底层错误
0x4150	本轴当前状态为 ErrorStop 状态，无法执行 NS_MC_Jog
0x4151	本轴当前状态为 Disabled 状态，无法执行 NS_MC_Jog
0x4152	本轴当前状态为 Homing 状态，无法执行 NS_MC_Jog
0x4153	本轴当前状态为 Stopping 状态，无法执行 NS_MC_Jog
0x4201	本轴当前状态为 ErrorStop/Disabled，无法执行 MC_Phasing 指令
0x4202	本轴当前状态为 Stopping，无法执行 MC_Phasing 指令
0x4203	本轴在 MC_CamIn 指令下跟随主轴，MC_Phasing 指令的指定的主轴 Master 非本轴的跟随主轴
0x4205	本轴在 MC_GeanIn 指令下跟随主轴，MC_Phasing 指令的指定的主轴 Master 非本轴的跟随主轴
0x4207	本轴在 NS_MC_RotaryCutIn 指令下跟随主轴，MC_Phasing 指令的指定的主轴 Master 非本轴的跟随主轴
0x4209	MC_Phasing 指令设定的主轴 Master 和从轴 Slave 无主从跟随关系，本次指令执行无效
0x4210	本轴在 NS_MC_SpecialCamIn 指令下跟随主轴，MC_Phasing 指令的指定的主轴 Master 非本轴的跟随主轴
0x4211	本轴在 NS_MC_SpecialCombineAxes 指令下跟随主轴，MC_Phasing 指令指定的主轴 Master 非本轴的跟随主轴
0x4212	本轴在 Mode 为 1 的 NS_MC_SpecialCombineAxes 指令下跟随主轴，MC_Phasing 指令无法执行
0x4251	本轴当前状态为 ErrorStop/Disabled，无法执行 MC_MoveSuperimposed 指令
0x4252	本轴当前状态为 Stopping，无法执行 MC_MoveSuperimposed 指令
0x4351	MC_Home，FirstVecocity 设定超出范围（＞0）
0x4352	MC_Home，SecondVecocity 设定超出范围（＞0）
0x4353	MC_Home，Mode 设定超出范围
0x4400	在旋转轴模式（Revolving_Axes=TRUE）下， StopByPos

	设定位置超出范围
0x4500	该控制器为普通 PLC 功能，不能应用此功能

附件七 经济型与 MP 型号 VA 控制器的区别

1. 脉冲的使用

MP 型号 VA 控制器	SP 经济型 VA 控制器
0~4 轴收发脉冲；	**寄存器%MB3.9809 控制 0~3 轴口脉冲收发：** 0~3 轴的各对应 bit 值为 1 时（例：将%MB3.9809 的值改为 15，即“1111”），VA 控制器硬件 0~3 轴口的收脉冲引脚将会改变为发脉冲（软件上分别对应为 5~8 轴）。最多可以 8 轴（0~3，5~8 轴）发脉冲； 0~3 轴的各对应 bit 值为 0 时（例：将%MB3.9809 的值改为 0，即“0000”），0~3 轴为既可以发脉冲也可以收脉冲的状态； **注意：** （1）硬件上的最后一个轴口只收脉冲； （2）硬件上的最后一个轴口在软件上为轴 4；

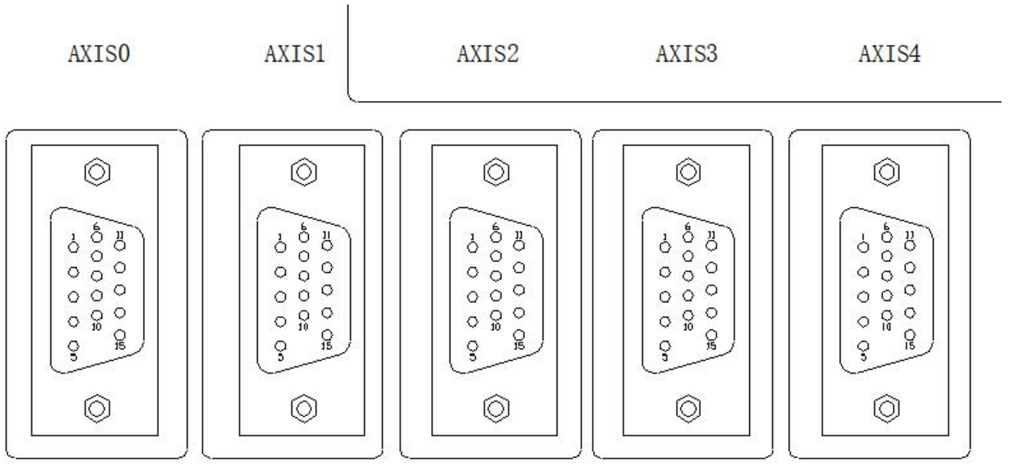
2. 模拟量的使用

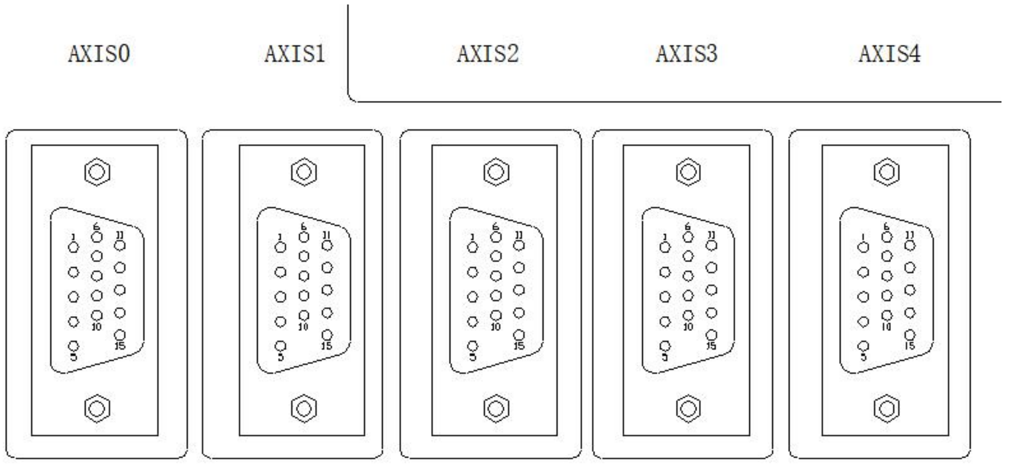
MP 型号 VA 控制器	SP 经济型 VA 控制器
（1）0~3 轴可以通过发模拟量控制轴，进行运动控制； （2）数字量输入 -2048～0 0~2047 对应模拟量输出 -10V~0V 0V~10V	（1）0~3 轴只能发模拟量（AO0~AO3），不能通过模拟量进行运动控制； （2）其中轴口输出的模拟量与数字量的对应关系为-10V~0V 和 0V~10V 分别对应数字量-32768~0 和 0~32767；

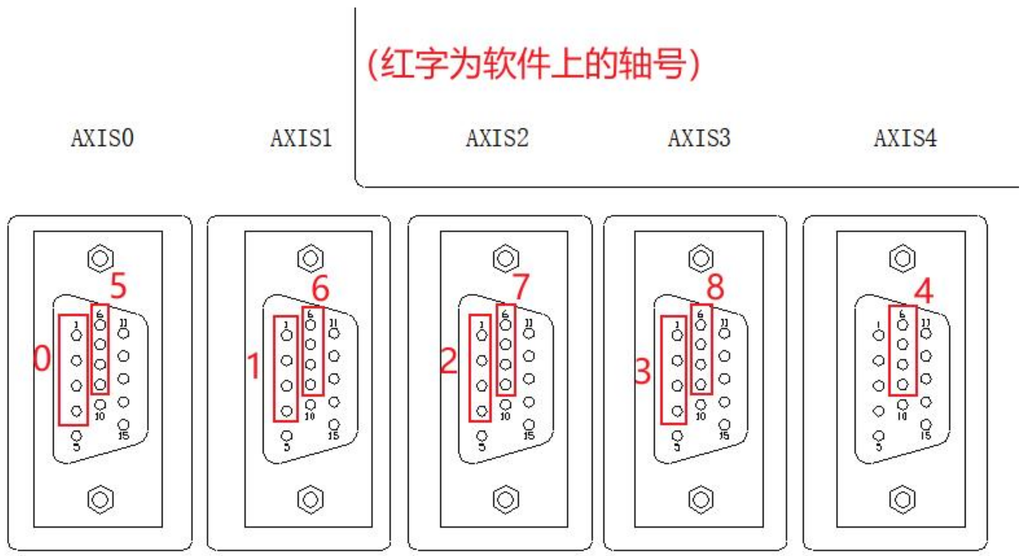
3. 拓展 IO 的使用

MP 型号 VA 控制器	SP 经济型 VA 控制器
可以自动识别拓展 IO 类型，也可以手动设置；	只能手动设置连接的拓展 IO 类型，详细操作及介绍见《12.2.2　手动选择拓展类型》

4. 硬件轴口脉冲控制的区别

MP 型号 VA 控制器

AXIS0~AXIS4 都是 1~4 引脚发脉冲，6~9 引脚收脉冲

SP 经济型 VA 控制器
寄存器%MB3.9809 对应各轴 bit 位为 0 时

（1）AXIS0~3 收发脉冲，AXIS4 只收脉冲；

（2）AXIS0~AXIS3 都是 1~4 引脚发脉冲，6~9 引脚收脉冲；AXIS4 的 6~9 引脚收脉冲，1 引脚为绝对值编码器输入信号负极,2 引脚为为绝对值编码器输入信号正极（可支持编码器类型详见《11.3.3 轴参数设定》） （3）AXIS0~4 分别对应软件上的轴 0~4；
寄存器%MB3.9809 对应各轴 bit 位为 1 时 
（1） 硬件上 AXIS0~3 的 0~4 引脚，6~9 引脚都可以发脉冲，一共可以发出 8 路脉冲，AXIS4 只收脉冲； （2） AXIS0 的 1~4 引脚对应软件上的 0 轴，AXIS0 的 6~9 引脚对应软件上的 5 轴；其他轴如上图以此类推；

5. 高速计数器的区别

MP 型号 VA 控制器	SP 经济型 VA 控制器
NS_CC_Counter：计数轴号为 0~6，计数轴为 AXIS0~AXIS4 **NS_CC_DI_Counter**：计数轴号为 0~6，有 16 个计数 DI	**NS_CC_Counter**： 计数轴号为 0~3； 计数轴： AXIS4 一直可以用作计数轴； AXIS0~AXIS3：当寄存器%MB3.9809 对应位为 0 时，可以用作计数轴，当寄存器%MB3.9809 对应位为 1 时，不能用作计数轴；

	NS_CC_DI_Counter： 计数轴号为 0~3，有 16 个计数 DI

6. 特殊功能块的使用

MP 型号 VA 控制器	SP 经济型 VA 控制器
模拟量控制模式下： 威科达特殊功能块的各种模式都可用； **脉冲和 CANOpen 控制模式下：** NS_MC_RotaryCutIn： 模式 2，模式 3，模式 4 不可用； NS_MC_SpecialCamin： 模式 1，模式 2 不可用； NS_MC_SpecialCombineAxes： 模式 1 不可用；	NS_MC_RotaryCutIn： 模式 2，模式 3，模式 4 不可用； NS_MC_SpecialCamin： 模式 1，模式 2 不可用； NS_MC_SpecialCombineAxes： 模式 1 不可用；

附件八 版本号

说明书版本	APP 版本	固件库版本
2.0	476	476
2.1	497	496

欢迎您宝贵的意见!

我们想竭诚为您服务，努力完善自己。由于编者水平有限，错漏在所难免恳请读者不吝指正。我们希望您阅读本书、使用产品时，如发现错误、论述不详或使用问题找不到相应解释时，请您致电我们或者填写意见表邮寄给我们，我们真诚期待您的宝贵意见。来电请致:

客服热线：40008-50004

专注行业 精于方案

威科达科技

深圳市威科达科技有限公司

SHENZHEN VECTOR TCHNOLOGY CO.LTD

地址：深圳市南山区留仙大道创客小镇13栋

电话：0755-26610452

研发大楼：广东省东莞市松山湖高新技术产业开发区南山路一号中集智谷12栋

电话：0769-22235716

关注公众号