

# BECKHOFF



## TwinCAT 3 运动控制教程

Version 1.15

毕孚自动化设备贸易（上海）有限公司

2019年4月



# 前言

TwinCAT3 是基于 PC 的控制软件并且它开启了一个新的时代，是倍福公司历史上又一个里程碑。

特别是在高效的工程领域中 TwinCAT3 将模块化思想以及其灵活的软件架构，融入到整个平台。

几乎每一种控制应用程序都能在 TwinCAT3 中实现。从印刷设备、木工设备、塑料机械或门窗设备、风力发电机和实验台，亦或是楼宇，诸如剧院，以及运动场，一切都可以通过 TwinCAT3 实现自动化。

用户可以选择不同的编程语言来实现这些应用。除了经典的 PLC 编程语言的 IEC 61131-3，用户现在也可以用高级语言 C 或 C++，以及 MATLAB®/ Simulink®。

整合了运动功能从而简化了工程项目，以及全新的安全应用编辑更加人性化。这些以及更多的特性都证明了为什么 TwinCAT3 也名为扩展的自动化。

本书针对任何想要学习倍福 TwinCAT3 软件如何实现基于 PC 控制编程的读者，阅读本书需要预先具备 IEC61131-3，C/C++或 MATLAB®/ Simulink®中至少一种编程语言的知识。

本书内容的框架安排如下：

第一章对 TwinCAT NC PTP 的系统进行概述，简单介绍 TwinCAT NC PTP 与 TwinCAT PLC 的关系、NC 轴的类型、控制周期等。

第二章介绍如何扫描驱动器以及 Axis 轴的重要参数以及如何实现单轴和多轴的调试。

第三章 NC PTP 功能所需要的库文件，如何利用功能块实现使能、点动、绝对定位、电子齿轮、寻参等。

第四章到第八章介绍电子凸轮、位置外部设定值发生器、位置补偿功能、飞锯功能、fifo 功能所适用的场合，创建方法以及程序编程相关功能块等。

第九章介绍如何使用 PLC 程序来修改 NC 轴参数。

本书所有的内容都会不间断更新，如果想获取更新的教材可以通过访问倍福虚拟学院获取到，当然本书所有配套的案例程序也会在此虚拟学院中供所有读者免费获取。

虚拟学院地址：<https://tr.beckhoff.com.cn/course/view.php?id=115>

本书的撰写过程中，尽量确保不出现错误，但难免有疏忽，如果您在阅读过程中发现错误，非常欢迎您反馈给我们，请发邮件至：

[lw.zhang@beckhoff.com.cn](mailto:lw.zhang@beckhoff.com.cn)

编者 张立文

2017 年 7 月

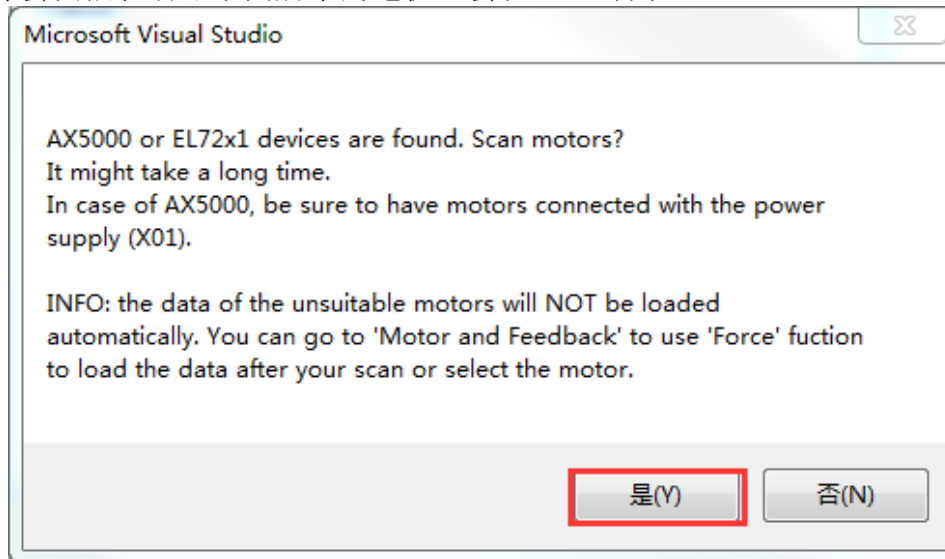
# 目录

一、	TwinCAT NC PTP 硬件配置 .....	2
二、	TwinCAT NC PTP 系统介绍 .....	15
三、	PLC Control 编程控制电机 .....	18
四、	电子凸轮表功能.....	37
五、	位置外部设定值发生器 .....	66
六、	位置补偿功能 .....	77
七、	飞锯功能.....	87
八、	FIFO 功能.....	100
九、	PLC 程序修改 NC 轴的参数设置.....	113
十、	NCI 入门介绍.....	121

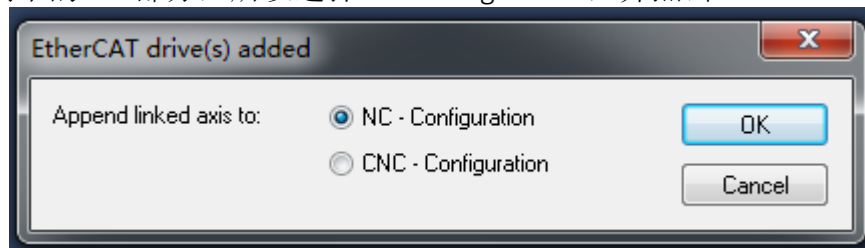
# 一、TwinCAT NC PTP 硬件配置

## 1. 硬件扫描

将培训器材上电后开始扫描硬件，培训室实验器材有两种，一种为：面板 PC+IO+AX5000 驱动+电机，简称器材 A，一种为：嵌入式 PC+IO+ AX5000 驱动+电机，简称器材 B，两种器材皆有 Beckhoff 的驱动和电机，在扫描硬件的时候会有下图提示，“AX5000 设备或 EL72x1 设备已经发现，是否要扫描驱动器所带的电机型号？”，点击“是”之后，软件会扫描驱动器下面所带的电机，会花一些时间。

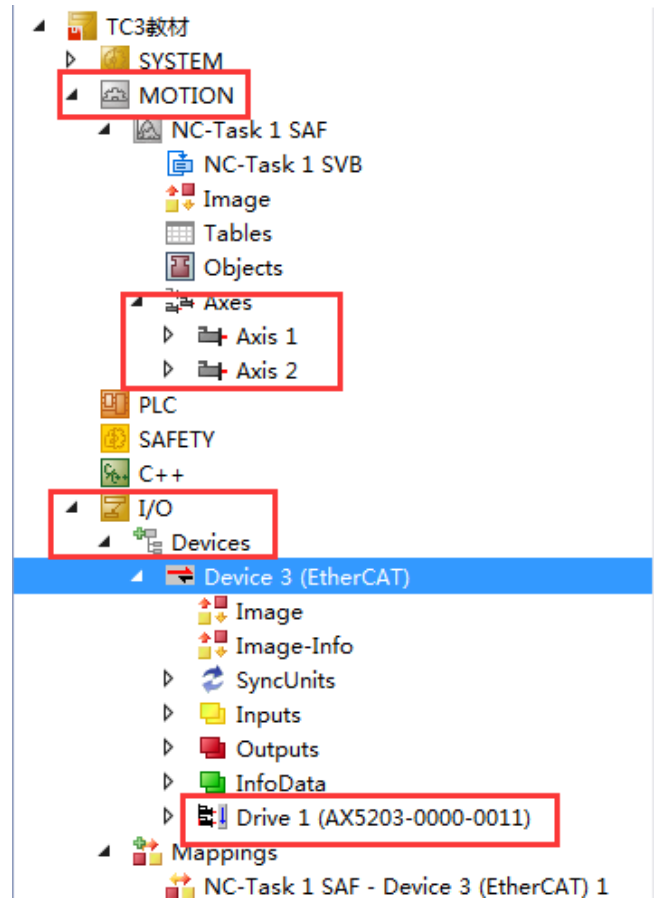


然后会弹出提示框询问，扫描到的实际轴将与 NC 轴还是 CNC 轴做链接，本教材中介绍的是运动控制中的 NC 部分，所以选择 NC-Configuration，并点击 OK。



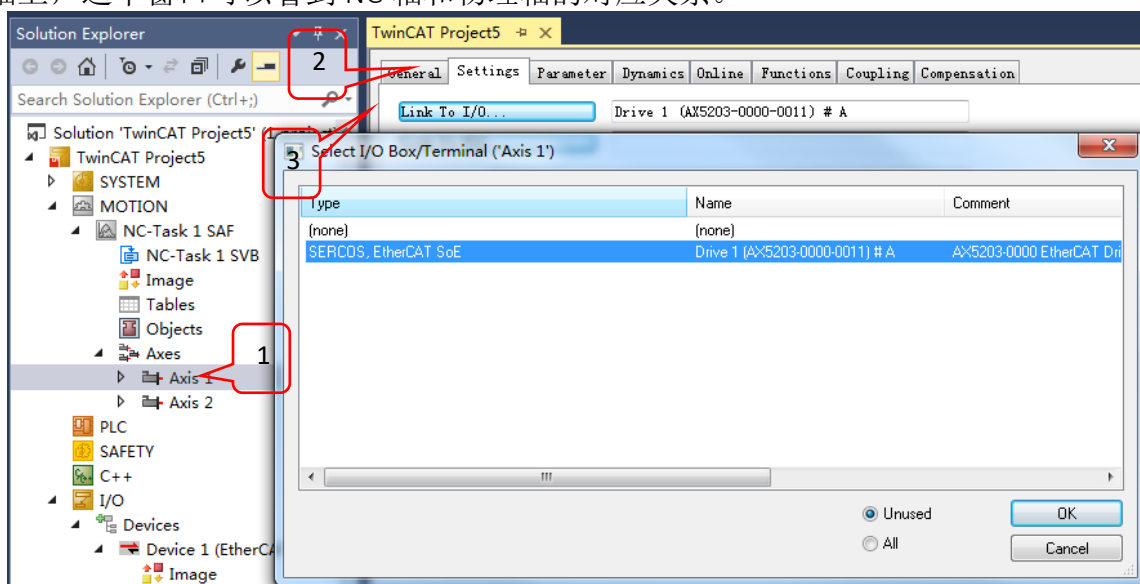
扫描完成后可以在 NC-Configuration 中看到 2 根轴，Axis1、Axis2 对应伺服驱动器控制的两台电机，IO-Configuration 中扫描到硬件驱动器 AX5203-0000-0011。





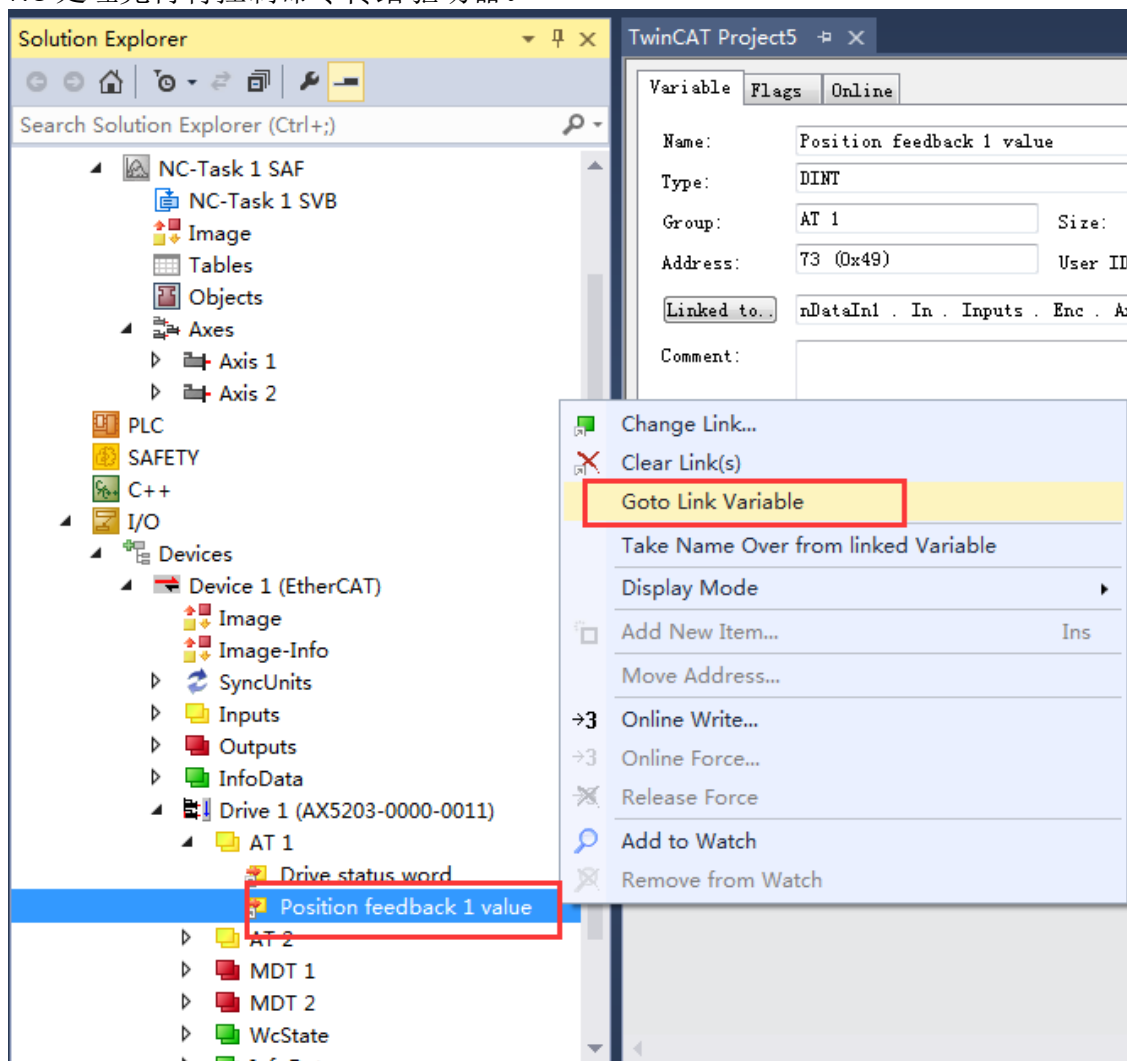
## 2. NC 轴和物理轴的关系

可以通过 Axis1-Settings-Link 来选择 NC 轴所关联的物理轴，这个链接在扫描硬件的时候自动添加，也可以手动右键 Axes，点击 Append axis 添加轴，将 NC 轴手动链接到物理轴上，这个窗口可以看到 NC 轴和物理轴的对应关系。



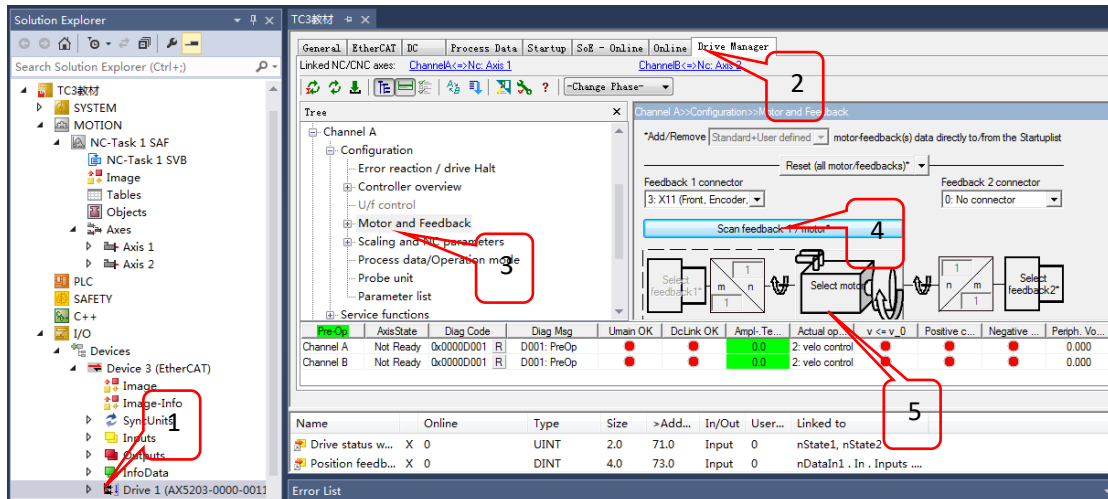
展开 I/O Devices 中 AX5203 下的变量，可以看到驱动器下面的变量都已经有关联了，

右键其中的一个变量，点击 Goto link variable，可以看到此变量和 NC 轴中的变量链接。NC 轴与物理轴就是通过这些变量来交换数据的，每个周期将驱动器的数据读取到 NC 中，NC 处理完再将控制命令传给驱动器。

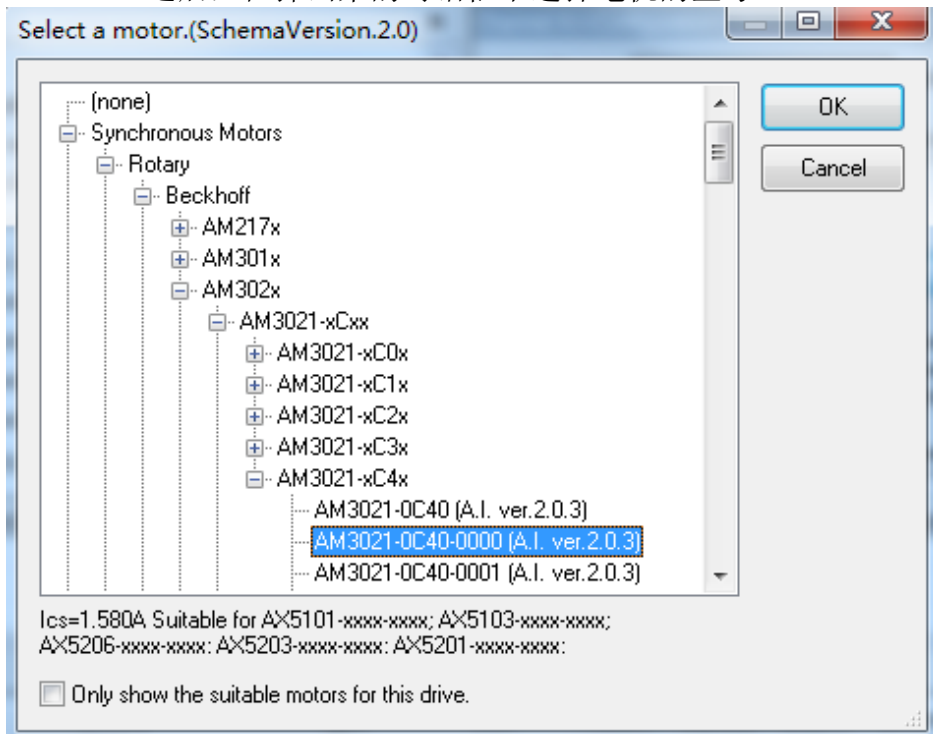


### 3. AX5000 的配置

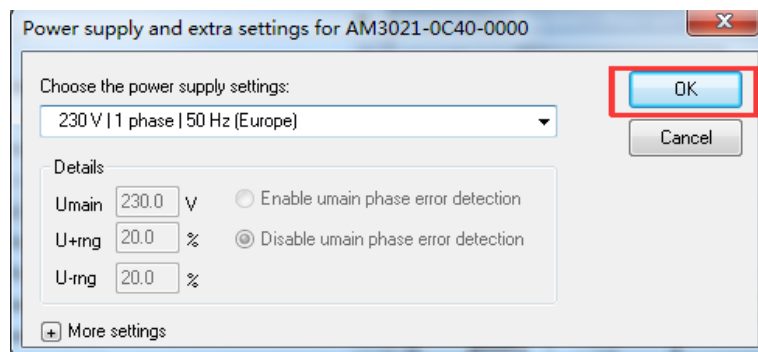
选中 Drive 4(AX5203-0000-0011)下的 Drive Manager 选项卡，Channel A 下面的 Motor and Feedback，然后点击 Select Motor 来手动添加驱动器所带的电机型号，也可以点击 Scan feedback 1/moter\*来自动获取电机型号，Drive Manager 是用来对 AX5000 驱动器进行配置的窗口，我们 AX5000 驱动器的配置软件没有额外的软件，直接通过 System Manager 软件可以配置，并且不需要专门的电缆，只需要网线即可完成配置，有些电机无法通过自动扫描的方式获取型号，因此只能通过 Select Motor 手动选择电机。



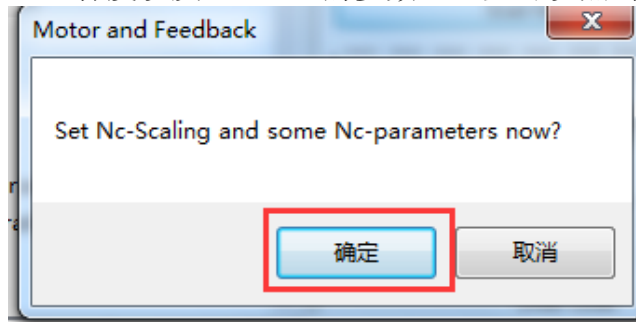
点击 select a motor 之后，在弹出来的对话框中选择电机的型号。



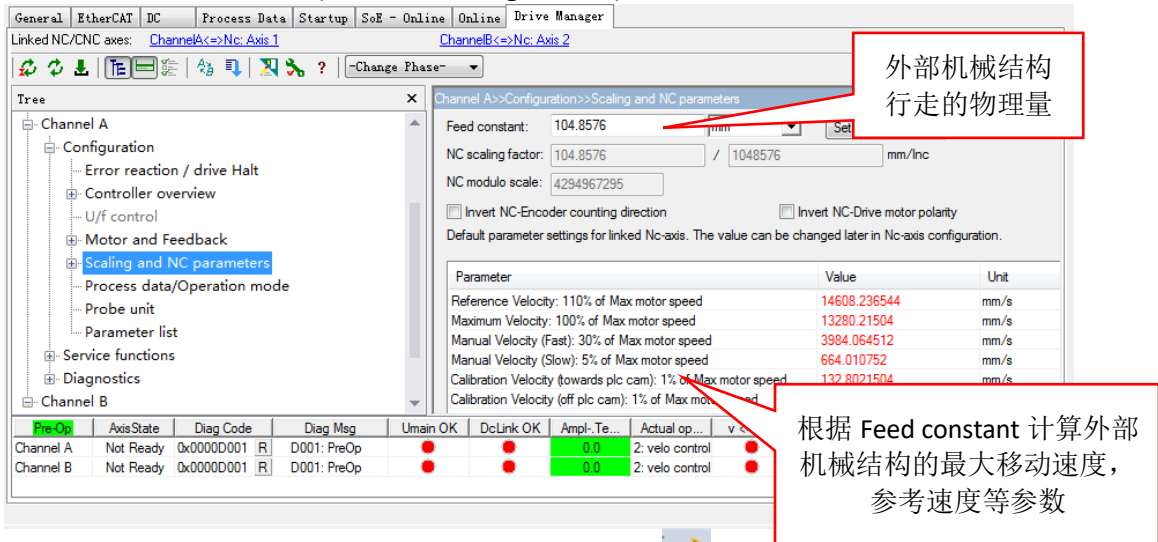
选择合适的电机型号之后点击 OK，接下来会提示如下窗口，选择驱动器实际的供电类型，点击 OK。



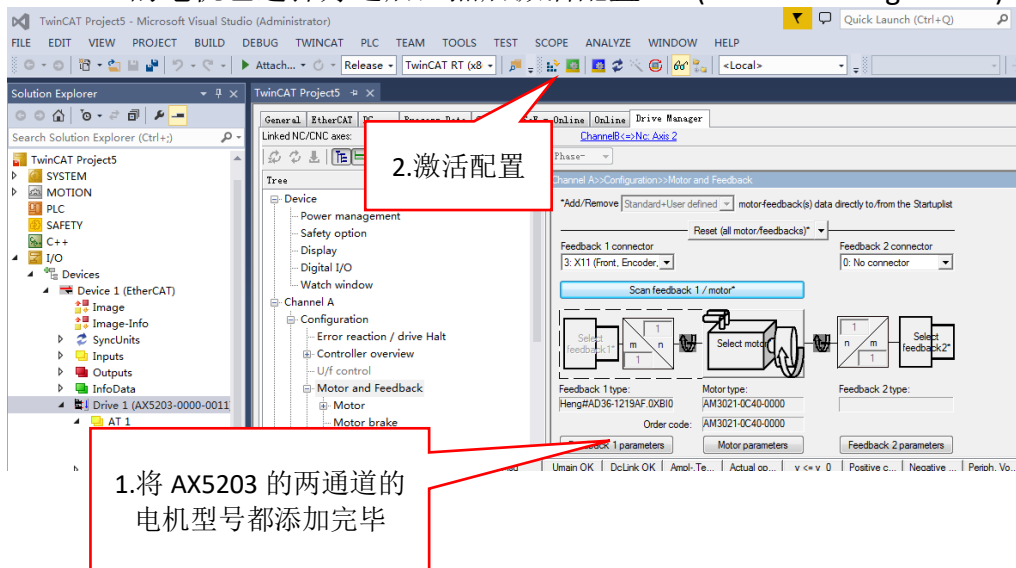
然后会提示是否设置 NC-标度以及一些 NC 的参数，这里可以点击确定或者取消。



如果点击确定，那么会提示如下窗口，这里主要是用来设置电机旋转一圈，实际外部机械结构行走的物理量，比如电机带了一个丝杠，如果电机旋转一圈，丝杠移动的位移为 20mm，那么就将 20mm 填入 Feed Constant 这个参数里面，然后软件会自动根据 Feed Constant 这个参数计算出丝杠移动的最大速度，参考速度等参数，点击 Set NC Parameters 即可保存，当然也可以不点 Set NC Parameters，用软件的默认参数，如果让这些参数生效，那么一定要激活配置 (Activate Configuration)。

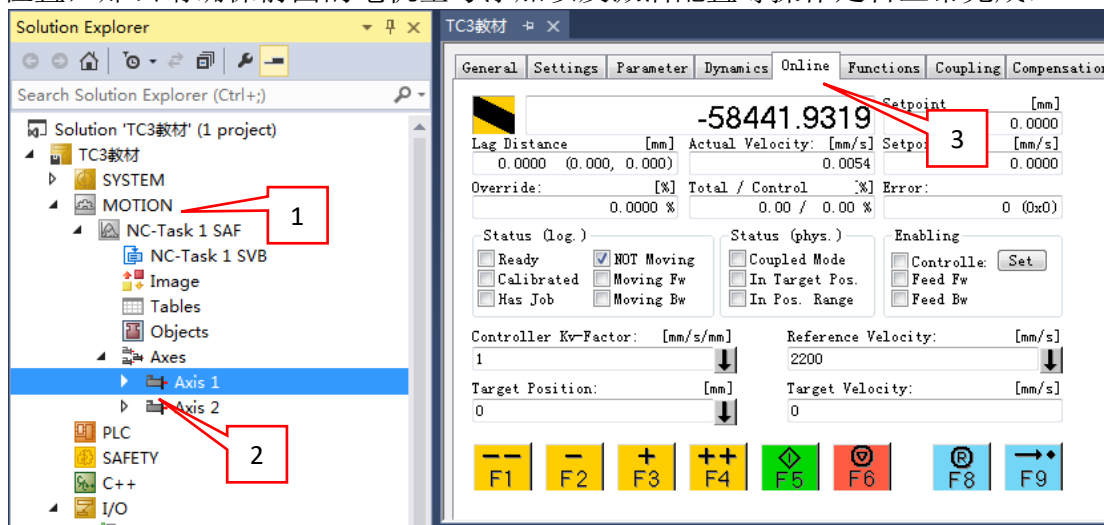


将 Channel B 的电机也选择好之后，然后激活配置 (Activate Configuration)

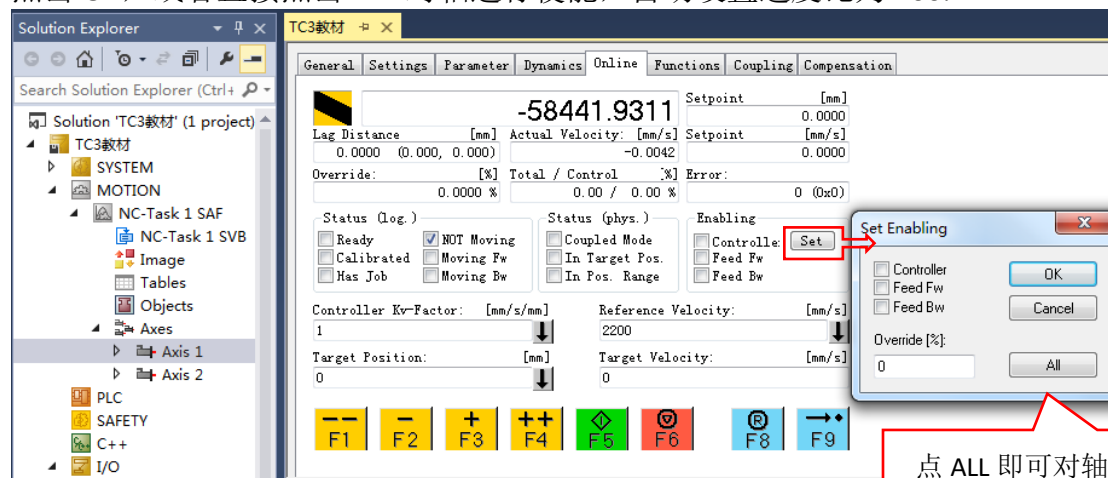


#### 4. AX5000 通过 System Manager 软件调试

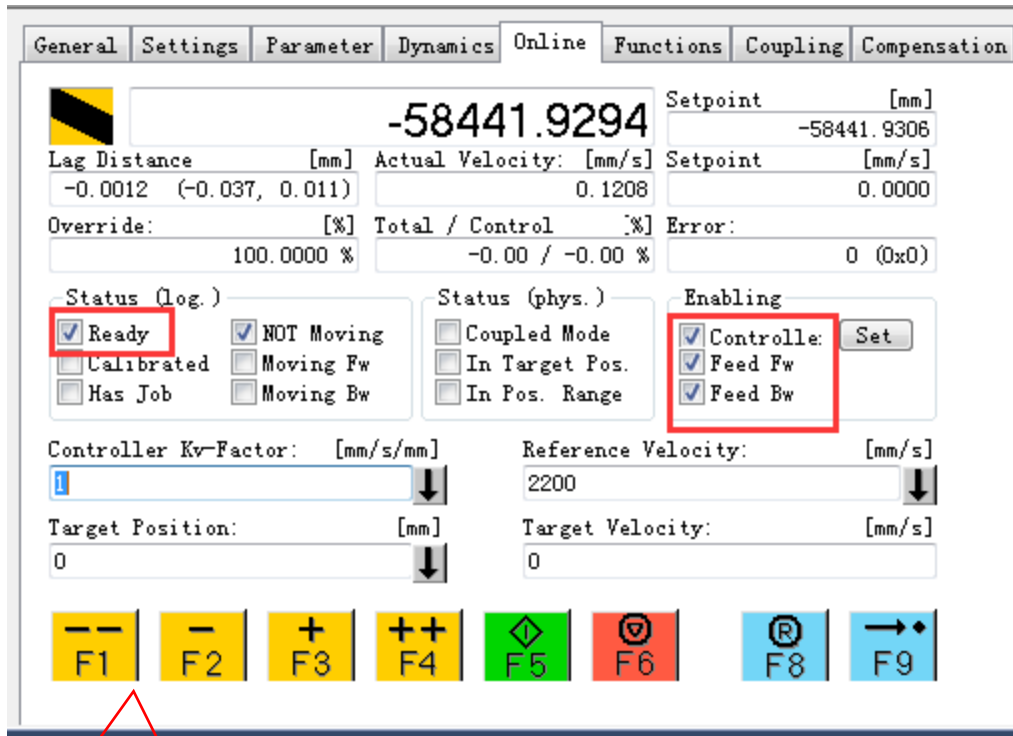
激活配置后，将 TwinCAT 切换到运行模式，然后单击 MOTION，单击 Axis1，单击 Online 选项卡，可以在这里对伺服轴进行调试（注：如果在 Online 选项卡里面看不到轴的当前位置，那么请确保前面的电机型号添加以及激活配置等操作是否正常完成）



单击 SET，手动勾选 Controller，Feed Fw，Feed Bw，并设置 Override（速度比），然后单击 OK，或者直接单击 ALL 对轴进行使能，自动设置速度比为 100%。

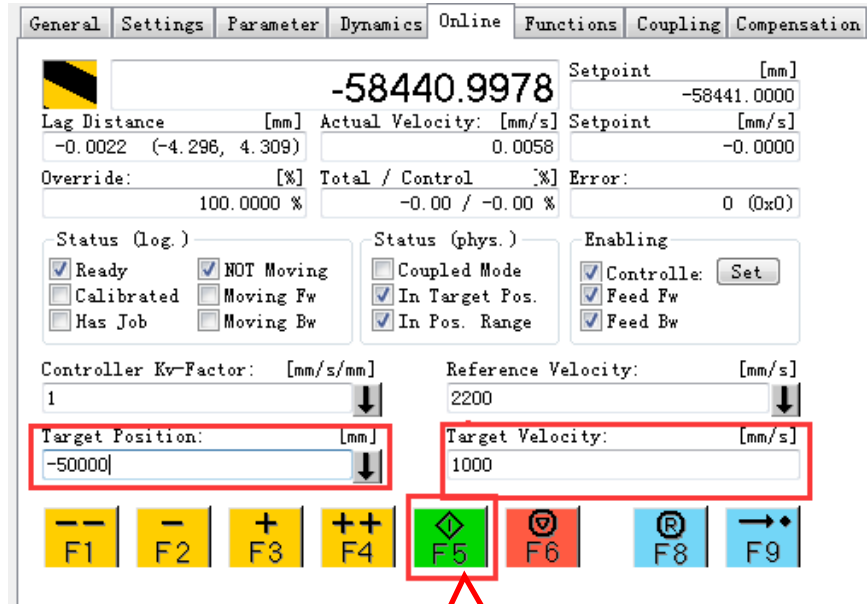


使能之后可以看到 Ready 状态会打勾，代表电机已使能，Controller, Feed Fw, Feed Bw 这些状态也会勾上，然后按下 F1 至 F4 即可对电机进行点动操作，按下 F1 点动，放开 F1 电机停止，点动速度在 Parameter 选项卡中的 Manual Velocity 中设置，默认速度为 100mm/s 与 600mm/s，分别对应慢速点动和快速点动。



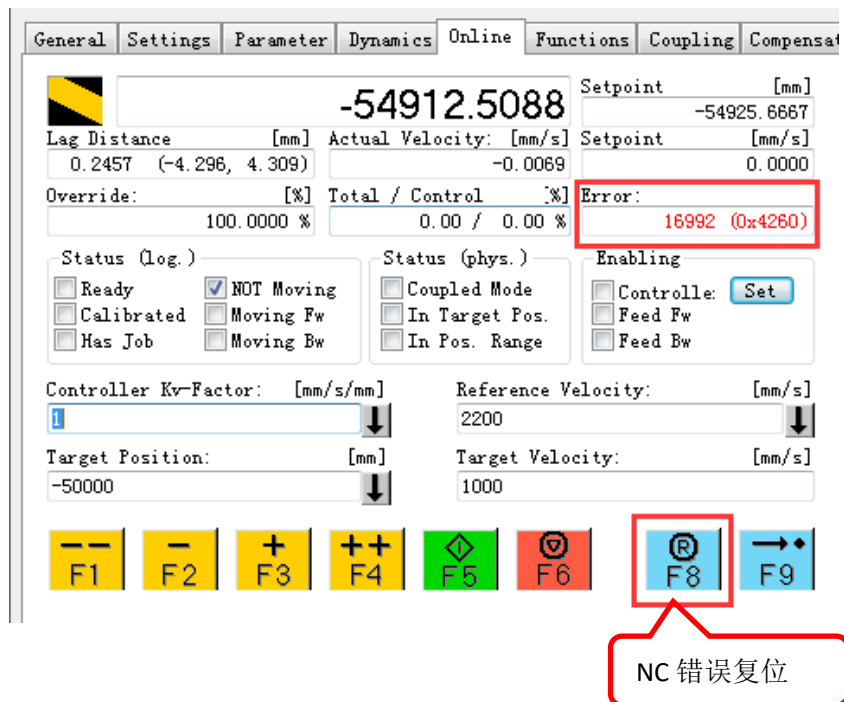
F1-F4 皆为点动按钮

设置完 Target Position 和 Target Velocity 后按下 F5，即可实现位置控制，电机会以设定的目标速度走到目标位置，如当前位置-58441，目标位置为-50000，那么触发 F5 后，电机就会从-58441 的位置移动到-50000，是绝对位置定位，定位的过程中可以使用 F6 停止。



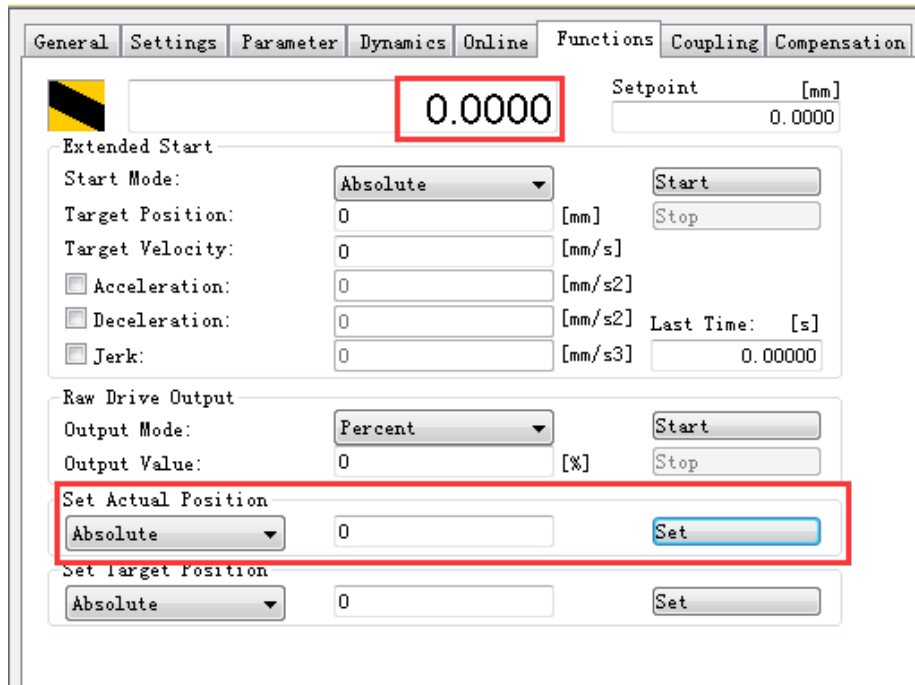
绝对位置定位

当 NC 报错之后，Error 中会有错误代码，需要通过 F8 来对错误进行复位，否则轴无法继续动作，F9 是找原点的按钮，按下 F9 之后，轴位置会变成 99999……，并慢速移动，但是找原点的过程中需要一个外部的硬件信号做为原点信号，这个原点信号在 Online 窗口中无法捕捉，因此一般不采用 F9 按钮进行寻参，而是通过程序中编程来实现找原点的功能。

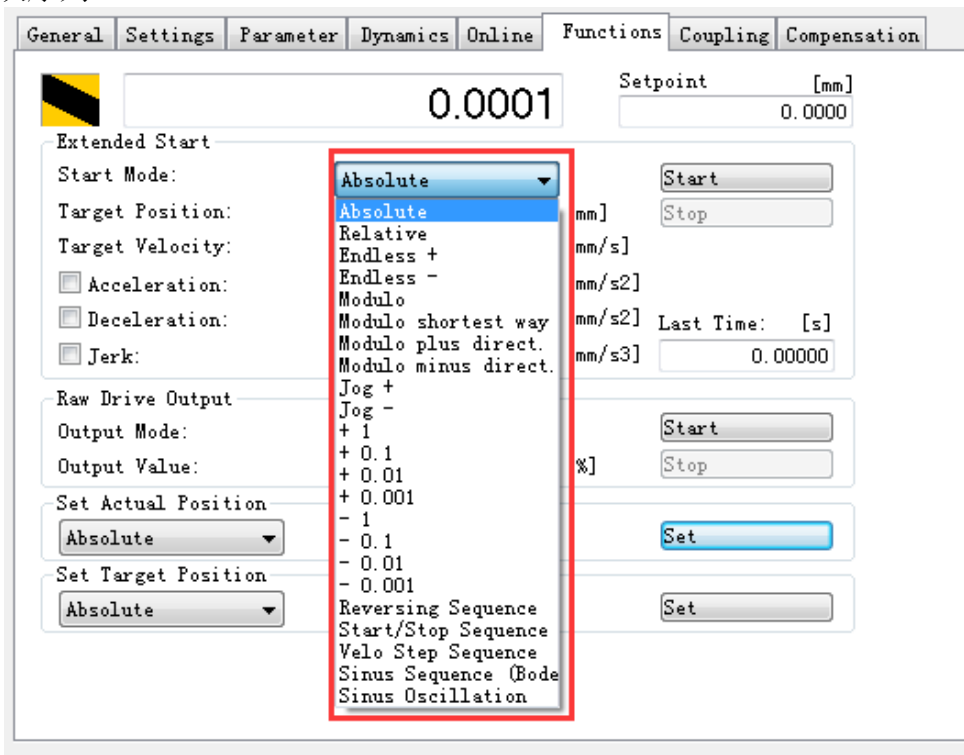


通过 Functions 里面的 Set Actual Position 可以修改轴的当前位置，如果将当前位置设置为 0，那么当前位置即为原点，此位置在 TwinCAT 重启之后会丢失，如果是绝对值编码器类型的反馈，那么重启之后以编码器的实际反馈位置为当前位置。



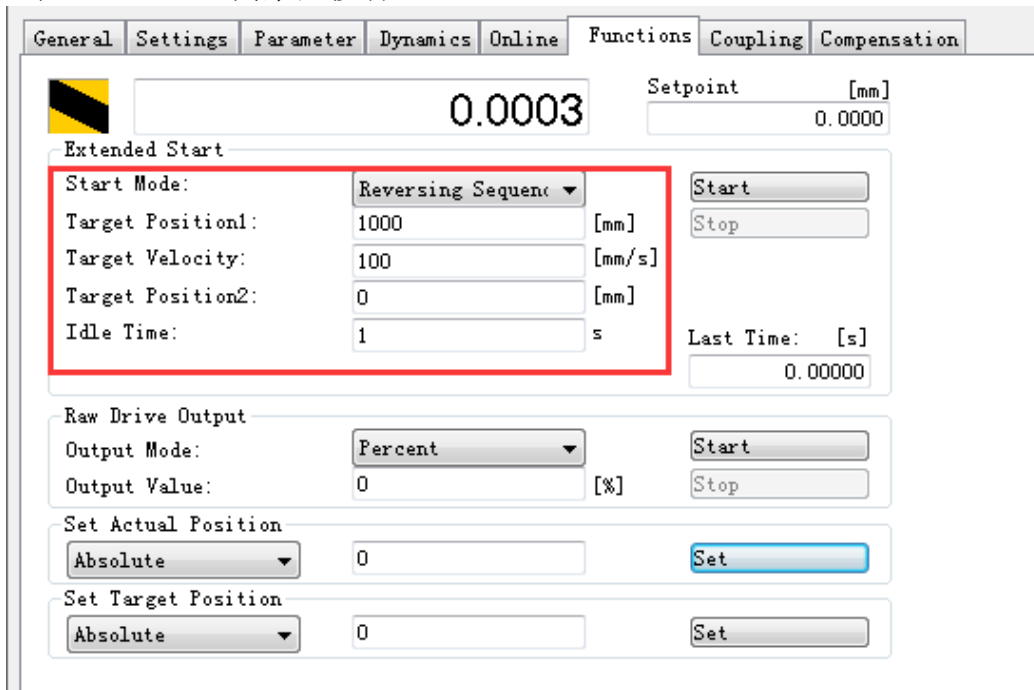


Functions——Start Mode 菜单中有很多对单轴的调试方法，常用的有 Absolute（绝对位置移动），Relative（相对位置移动），Endless+（无限正反转），Modulo（模值移动），Reversing Sequence（往返序列），Start/Stop Sequence（启停序列），Velo Step Sequence（速度阶跃序列）

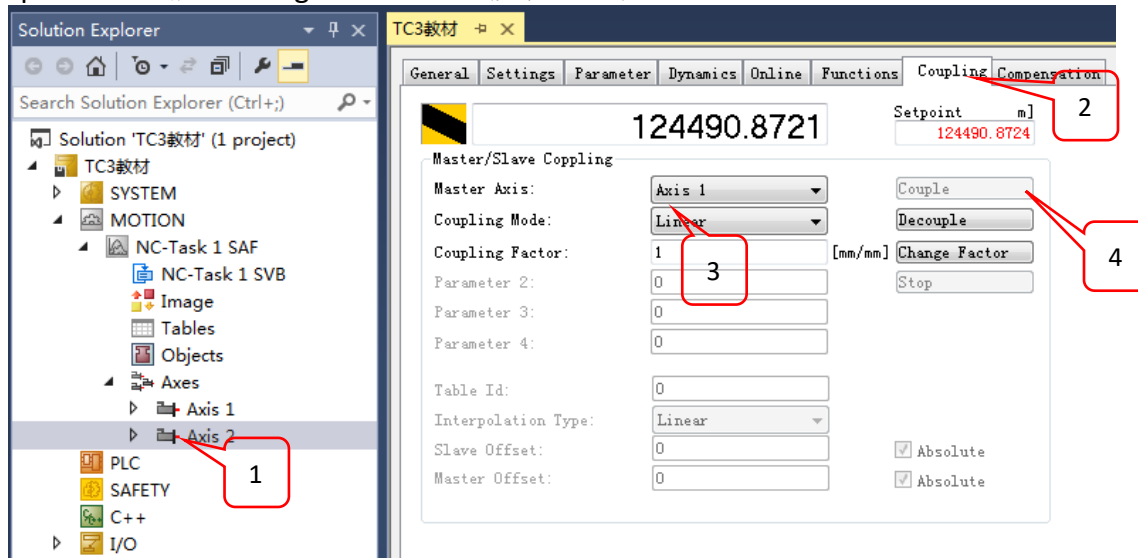


选择 Start Mode 为 Reversing Sequence，设置 Target Position1,Target Velocity,Target Position2,Idle Time（到达目标位置之后的等待时间）之后，点击 Start 即可让轴在

Position1 和 Position2 之间来回移动。



电子齿轮功能（主轴与从轴的速度保持比例关系，从轴跟随主轴移动）：首先将两根伺服轴都使能，然后选中 Axis2，Coupling 选项卡中，Master Axis 选择 Axis 1，Coupling mode 设置为 linear，Coupling Factor 设置为 1，然后点击 Couple 进行耦合，此时看到 Axis2 的 Setpoint 变为红色，代表 Axis2 已经作为从轴处于耦合状态，不能单独对 Axis2 进行控制了，此时控制 Axis1 轴动作的时候，Axis2 也会跟随动作，速度为 1:1，点击 Decouple 进行解耦，Change Factor 可以修改主从轴之间的速度比。



## 1. NC 参数设置

在 Parameter 选项卡中需要设置一些 NC 的参数，Reference Velocity 是参考速度，一般为 Maximum Velocity 的 110%，Maximum Velocity 是轴的最大速度，Default Dynamics 展开

可设置加减速，Manual Motion and Homing 是点动和寻参的速度，Fast Axis Stop 是设置快速停止，Limit Switches 可以设置开启软限位，Monitoring 可以设置跟随误差的监视。

Parameter	Offline Value	Online Value	Type	Unit
- Maximum Dynamics:				
Reference Velocity	2200.0	2200.0	F	mm/s
Maximum Velocity	2000.0	2000.0	F	mm/s
Maximum Acceleration	15000.0	15000.0	F	mm/s <sup>2</sup>
Maximum Deceleration	15000.0	15000.0	F	mm/s <sup>2</sup>
- Default Dynamics:				
Default Acceleration	1500.0	1500.0	F	mm/s <sup>2</sup>
Default Deceleration	1500.0	1500.0	F	mm/s <sup>2</sup>
Default Jerk	2250.0	2250.0	F	mm/s <sup>3</sup>
+ Manual Motion and Homing:				
+ Fast Axis Stop:				
+ Limit Switches:				

Axis1——Enc 中的 Parameter 有 Scaling Factor Numerator 和 Scaling Factor Denominator 两个参数，用来进行定标，此参数比较重要，必须要设置，默认值分别是是 0.0001 和 1.0。

Scaling Factor Numerator 是电机转一圈最终工件移动量

Scaling Factor Denominator 是编码器反馈脉冲数

例如：电机转一圈，带动丝杠移动 5mm，AX5000 的编码器反馈为一圈 1048576，那么

Scaling Factor Numerator =5，

Scaling Factor Denominator=1048576。

例如：电机转一圈，带动一个圆形负载移动 360°，

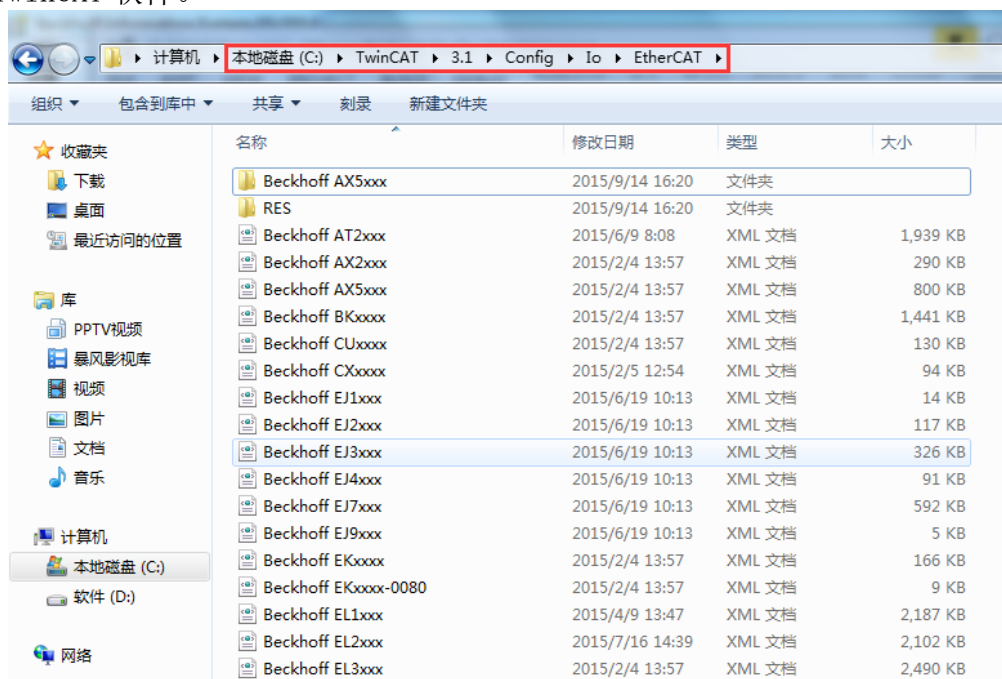
那么 Scaling Factor Numerator =360，Scaling Factor Denominator=1048576。（注：如用第三方伺服驱动器，那么编码器反馈不再是 1048576，需要根据第三方设备的实际反馈量来进行设置）

定标之后，NC 轴的位置和速度都是最终工件的位置和速度，用户可以直接通过 Motion 或者编程控制最终工件，而不需要关注中间电机的转速和位置。

Parameter	Offline Value	Online Value
- Encoder Evaluation:		
Invert Encoder Counting Direction	FALSE	
Scaling Factor Numerator	0.0001	
Scaling Factor Denominator (default: 1.0)	1.0	
Position Bias	0.0	
Modulo Factor (e.g. 360.0°)	360.0	
Tolerance Window for Modulo Start	0.0	
Encoder Mask (maximum encoder value)	0x00FFFFFF	
Noise level of simulation encoder	0.0	
- Limit Switches:		
Soft Position Limit Minimum Monitoring	FALSE	
Minimum Position	0.0	
Soft Position Limit Maximum Monitoring	FALSE	
Maximum Position	0.0	

## 2. 添加第三方 EtherCat 总线的伺服驱动器

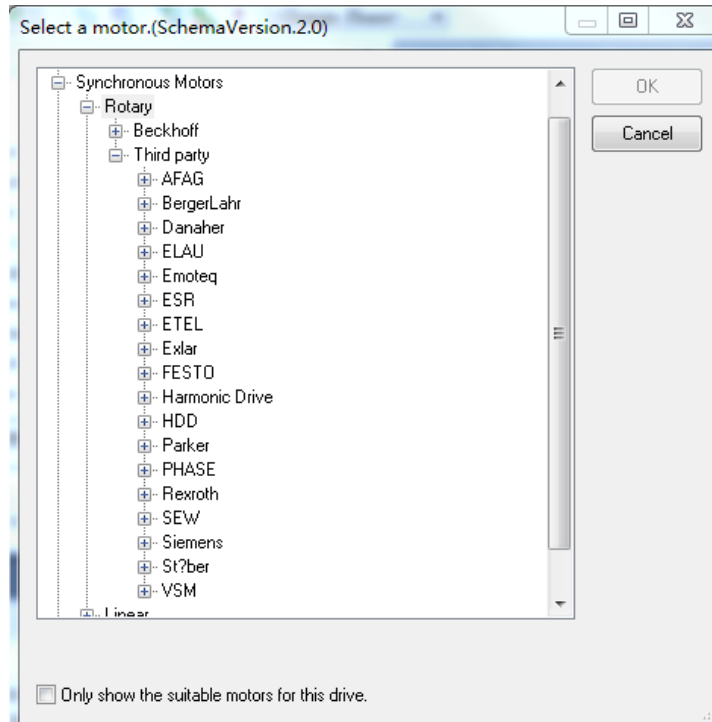
如果使用 BECKHOFF 的 PC 控制第三方的 EtherCAT 伺服驱动器，那么首先要将对方设备的从站描述文件——xml 文件拷贝到 C:\TwinCAT\3.1\config\Io\EtherCAT 路径中，然后重启 TwinCAT 软件。



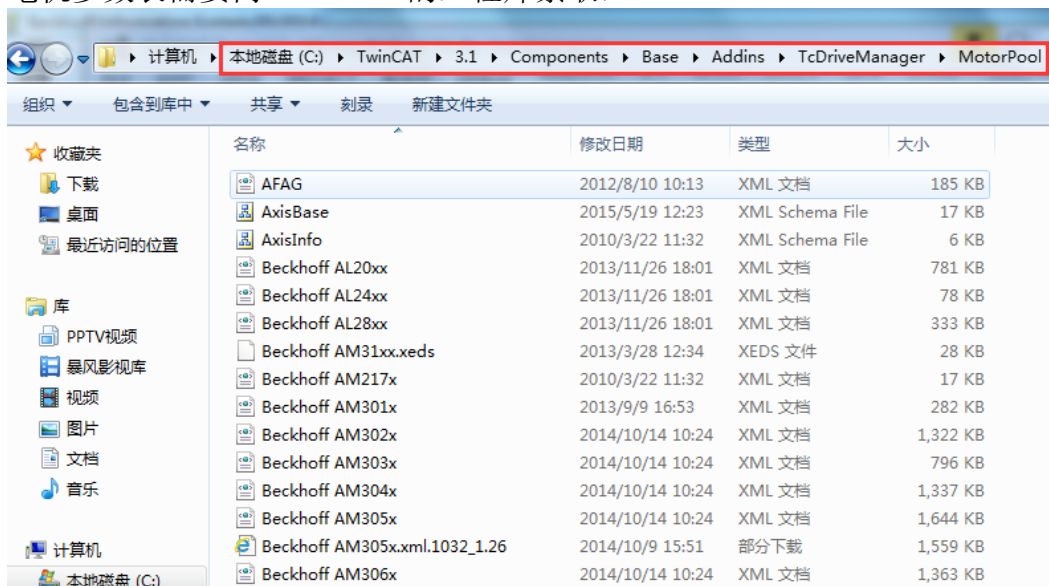
当 xml 文件拷贝到对应路径之后，可自动扫描到第三方的伺服驱动器，也可以手动添加第三方的伺服驱动器至 TwinCAT3 中进行配置。

## 3. 添加第三方厂家的伺服电机

如果使用 AX5000 带第三方的伺服电机，并且此伺服电机可以在手动添加窗口中找到，那么选中此电机点击 OK 即可添加。



如果在手动添加电机的窗口中找不到此第三方电机，就需要填写“电机参数表”并提供给 BECKHOFF 技术工程师，BECKHOFF 工程师会制作此电机的 xml 文件给客户，客户将此电机文件放 C:\TwinCAT\3.1\Components\Base\Addins\TcDriveManager\Motorpool 路径中，重启 TwinCAT 软件即可，此时在手动添加电机的窗口中就可以找到该型号的电机了。  
 （注：电机参数表需要问 BECKHOFF 的工程师索取）



## 二、TwinCAT NC PTP 系统介绍

TwinCAT NC PTP 是 Beckhoff 公司的运动控制软件的名称，TwinCAT 是 “The Windows Control and Automation Technology” 的缩写，即基于 Windows 操作系统的自动化控制技术，而 NC PTP 是 “Numerical Control Point To Point” 的缩写，NC (Numerical Control) 是自控领域的一个专业术语，类似 MC (Motion Control)，也指运动控制，NC PTP 就是点对点的运动控制。

TwinCAT NC 是基于 PC 的纯软件的运动控制，它的功能与传统的运动控制模块、运动控制卡类似。由于 TwinCAT NC 与 PLC 运行在同一个 CPU 上，运动控制和逻辑控制之间的数据交换更直接、快速，因此 TwinCAT NC 比传统的运动控制器更加灵活和强大。TwinCAT NC 的另一个特点是完全独立于硬件，用户可以选择不同厂家的驱动器和电机，而控制程序不变。程序的运动控制指令集遵循 PLCopen 组织关于运动控制功能块的定义规范 V1.0 和 V2.0。

TwinCAT NC 有 PTP 和 NC I 两个级别，PTP 即点对点控制方式，可控制单轴定位或者定速，也可以实现两轴之间的电子齿轮、电子凸轮同步。在此基础上，Beckhoff 还提供 Dancer Control (张力控制)、Flying Saw (飞锯)、FIFO (先入先出) 等多轴联动方式。此外，用户还可以在 PLC 程序中编写位置发生器，每个 PLC 周期都计算目标位置、速度和加速度，并发送给 TwinCAT NC 去执行。而 TwinCAT NC I 除了能够实现 TwinCAT NC PTP 的所有功能之外，还可以执行 G 代码，实现多轴之间的直线、圆弧和空间螺旋插补。

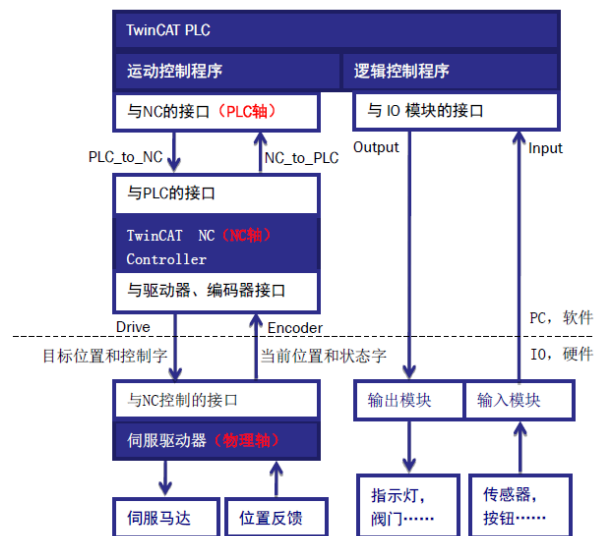
### 4. TwinCAT NC PTP 与 TwinCAT PLC 的关系

TwinCAT NC PTP 把一个电机的运动控制分为三层：PLC 轴、NC 轴和物理轴。

A、PLC 程序中定义的轴变量，叫做 PLC 轴。

B、在 NC 配置界面定义的 AXIS，叫做 NC 轴。

C、在 IO 配置中扫描或者添加的运动执行和位置反馈的硬件，叫做物理轴。它们的关系如图所示：



由图可见，PLC 程序对电机的控制，必须经过两个环节：PLC 轴到 NC 轴；NC 轴再

到物理轴。PLC 轴的控制，是指 PLC 程序中编程，调用运动控制库的功能块。

NC 轴不需要编程，它的运算分为轨迹规划、PID 运算和 IO 接口处理。其中轨迹规划和 PID 运算是固定的，与硬件无关。IO 接口处理随接口类型而不同。这些运算都在后台进行，用户只需要进行参数设置。这些参数可以固化在 TwinCAT System Manager 配置文件中，也可以在 PLC 程序中通过 ADS 指令读写。

物理轴，指驱动器、电机和编码器。物理轴的配置，主是对驱动器的设置。在驱动器中，要配置好正确型号的电机、编码器、电子齿轮比，还要调整位置环、速度环、电流环的 PID 参数。如果是总线接口，还要设置好接口变量和通讯参数。

TwinCAT NC 做轨迹规划，是指接收到 PLC 指令以某个速度运动到某个位置后，计算出每个 NC 周期（比如：2ms）伺服轴应该到达的位置。IO 接口处理，是指根据轴的硬件类型和相应的参数设置，进行单位换算，将 NC 运算得出的目的位置，换算成驱动器可接受的输出变量值。

## 5. TwinCAT NC PTP 控制的轴的类型和数量

和传统的硬件运动控制器和运动控制卡不同，TwinCAT NC PTP 是纯软件的运动控制。理论上，最多可以驱动 255 个伺服轴。在实际应用中，一个 EPC 或者 PC 上运行的 TwinCAT NC PTP 软件能够控制的伺服轴数量，与 PC 或者 EPC 的 CPU 速度、内存以及 NC 任务的周期有关。

TwinCAT NC 支持多种伺服轴类型，下面介绍几种常用类型：

### 总线接口

总线接口，又称数字接口，比如 Sercos，CanOpen (DS402)，Lightbus 等。由不同厂家生产的同一种总线协议的伺服驱动器，在 TwinCAT NC 中视作同一种驱动器。值得一提的是，对于 EtherCAT 接口的驱动器，其协议层通常使用 CanOpen，或者 Sercos。在 TwinCAT NC 中，EtherCAT 接口 CanOpen 协议的驱动器，与 CanOpen 接口 CanOpen 协议的驱动器，都视作同一种驱动器。同理，EtherCAT 接口 Sercos 协议的驱动器，与 Sercos 光纤接口 Sercos 协议的驱动器，也视作同一种驱动器。

### 紧凑型驱动模块

这里主要是指 Beckhoff 公司的步进电机驱动模块 KL2531/2541、EL7031/7041，伺服电机驱动模块 EL7201 等等。

### 高速脉冲接口

TwinCAT NC 通过控制脉冲输出模块 KL/EL2521 的输出频率，控制伺服驱动器或者步进电机驱动器。同时，TwinCAT NC 直接把 KL/EL2521 发出的脉冲数量，作为位置反馈信号。

### 模拟量控制

TwinCAT NC 通过控制电压输出模块 KL/EL4xxx 的电压，控制伺服驱动器和电机的速度。此时，必须配置编码器模块 KL/EL5xxx 作为位置反馈。

## 6. TwinCAT NC PTP 的控制周期

通常说的 NC 周期，是指轨迹规划和 PID 运算的周期，是 NC 与伺服驱动器交换数据的周期，目标位置、当前位置、控制字、状态字都以这个频率更新。在 TwinCAT System Manager 中，叫做 NC Task SAF 任务周期，默认值为 2ms，理论上最小设置为 50us。当连接硬件运动轴时，以 BECKHOFF 的伺服驱动器 AX5000 为例，位置环周期为 125us，所以 NC 周期设置为 50us 是没有意义的，实际上 250us 的 NC 周期已经是很高端的应用了。

另一个 NC 周期，是 NC 与 PLC 交换数据的周期，比如 NC 轴状态、当前位置、使能信



号等等，都是以这个周期刷新的。在 TwinCAT System Manager 中，叫做 NC Task SVB 任务周期，默认值为 10ms，与 PLC 程序中默认的任务周期一致。

## 7. TwinCAT NC PTP 的配置、编程、调试

在开发 PC 上安装 TwinCAT 时，如果选择 TwinCAT NC PTP 或者 TwinCAT NC I 级别，安装完成后，运行 TwinCAT System Manager，左边的树形结构中就包含了 TwinCAT NC Configuration 这一项。TwinCAT NC 任务和轴的配置调试就是在这一项下进行。

TwinCAT NC 任务的配置主要是设置任务周期，多数情况下，使用默认值即可。

TwinCAT NC 轴的配置包括：编码器（Enc）、驱动器（Drive）、NC 控制器（Ctrl）、与 PLC 的接口（Inputs 和 Outputs）。Enc 和 Drive 的配置决定了 NC 轴与哪个驱动器对应，而 Inputs 和 Outputs 则决定它对应 PLC 程序中的哪一个轴结构型变量。Ctrl 中的设置则决定了 PID 运算的模型和参数。

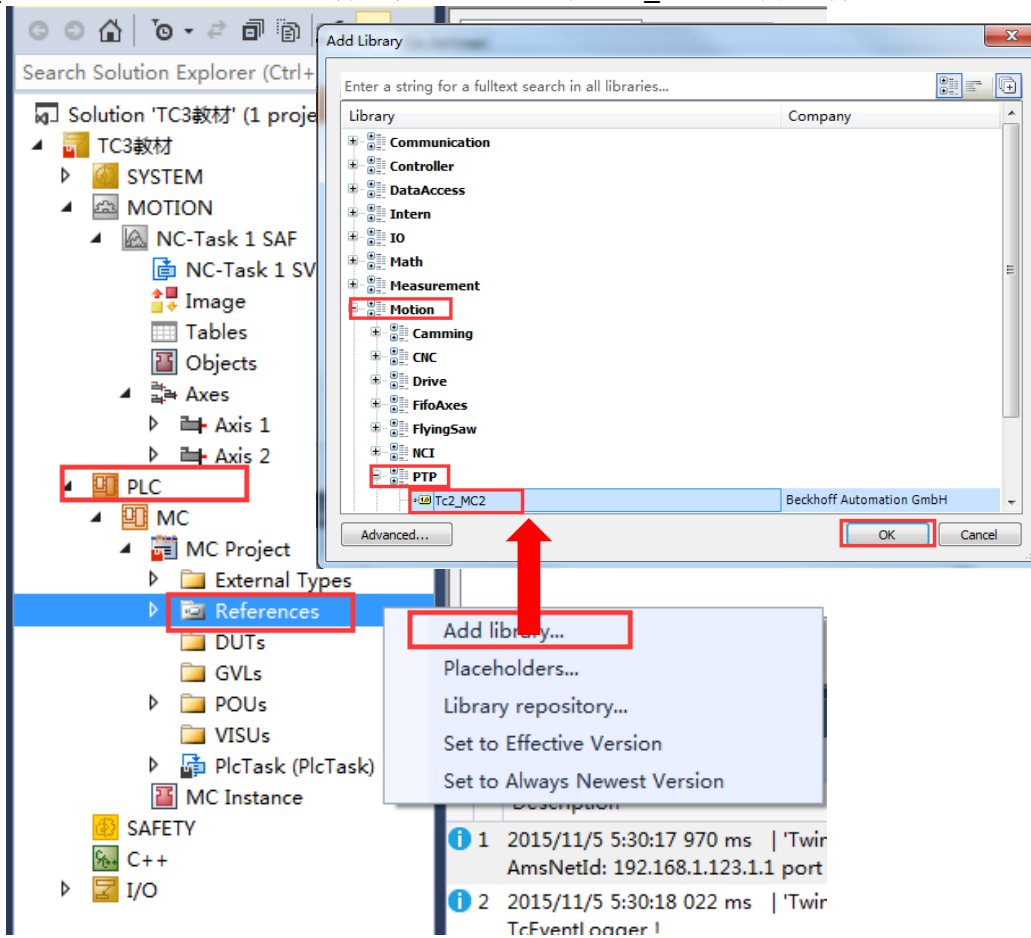
TwinCAT NC 轴的调试，分为单轴点动、指定方式动作和双轴齿轮或凸轮联动。这些动作都可以在 TwinCAT System Manager 的 NC Configuration 项下完成，不需要编写任何 PLC 程序。NC 轴调试的目标，是确保电机能够按要求走得准、走得稳，消除单位设置、PID 参数、传动机械方面的误差。

TwinCAT NC 轴的编程，在 TwinCAT PLC 中通过引用运动控制功能库 TcMC.Lib 或者 TcMC2.Lib，并调用其中功能块来实现。实际应用中，必须在 TwinCAT NC 轴调试完成后，才用 PLC 程序控制轴的动作，以达到设备的工艺要求。

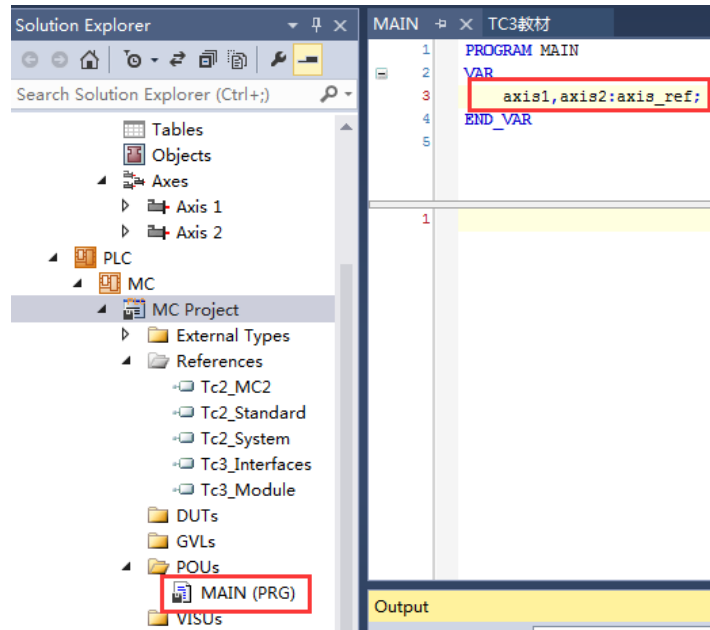
### 三、PLC Control 编程控制电机

#### 1. 添加运动控制库文件以及轴类型变量

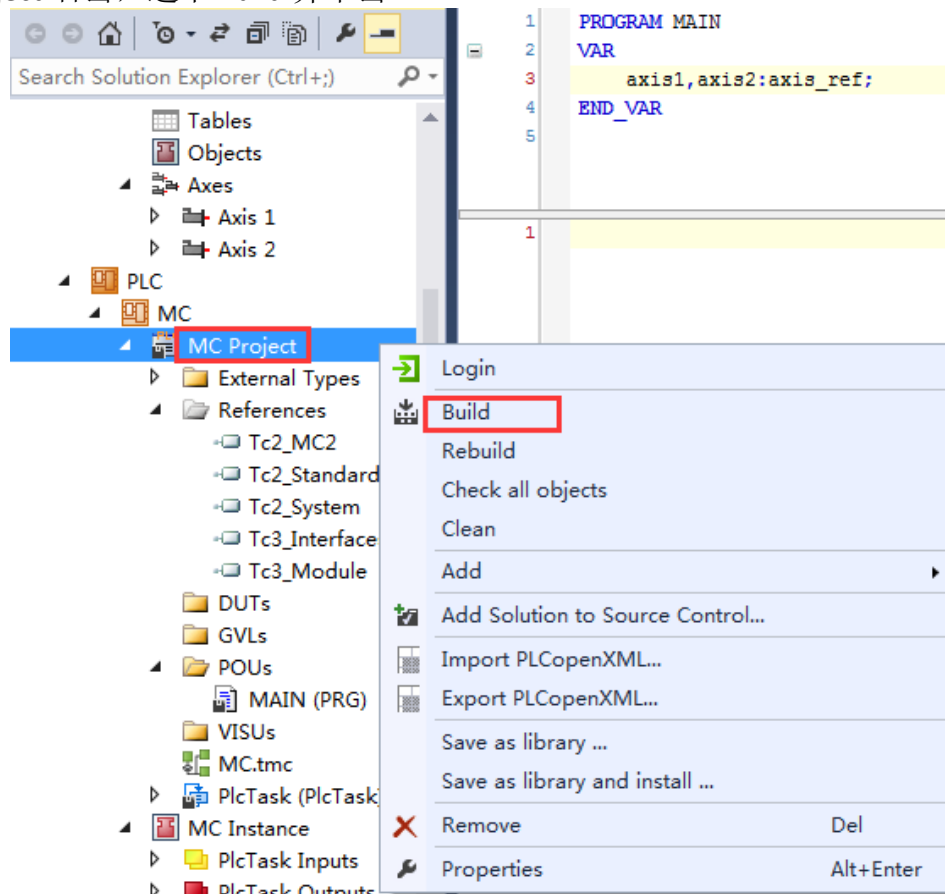
在 PLC 下新建一个项目(例如：本实例新建项目命名为 MC)，然后展开该项目，从下方找到 References 并右击，单击 Add library。从弹出的对话框中找到 Motion——PTP——Tc2\_MC2，选中单击 OK，这样就完成加载一个 TC2\_MC2 的库文件。



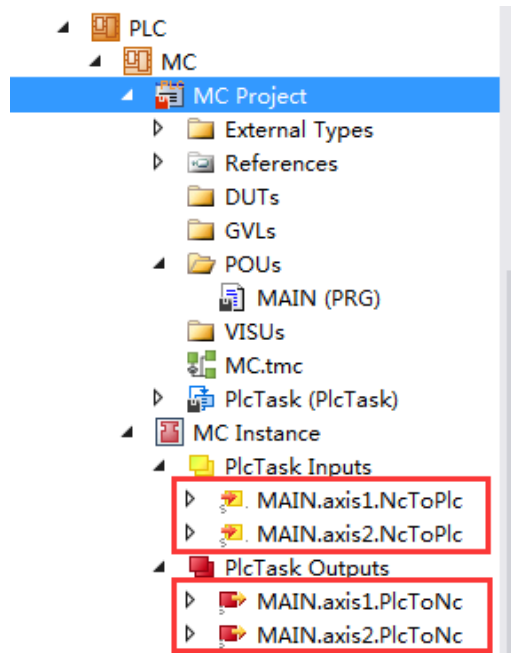
在 POU 中，主程序处新建两个 Axis\_ref 类型的变量，axis\_ref 是一个结构体，主要用来做 NC 和 PLC 数据交换用的，内部又嵌套了另外一些结构体，我们将 axis\_ref 类型的变量称之为轴类型的变量。



程序写完后，对它进行编译，查看是否编写错误。本实例的项目命名为 MC，所以找到 MC Project 右击，选中 Build 并单击。

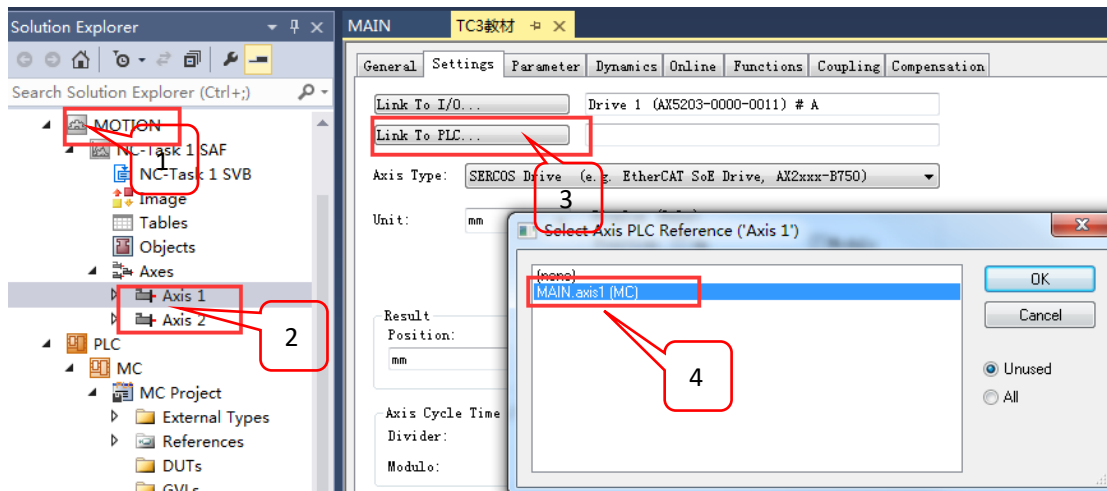


当 Build 编译成功后，可以在 MC Instance 下看 PlcTask Inputs 和 PlcTask Outputs 下分别绑定两个变量。



## 2. NC 与 PLC 的变量链接

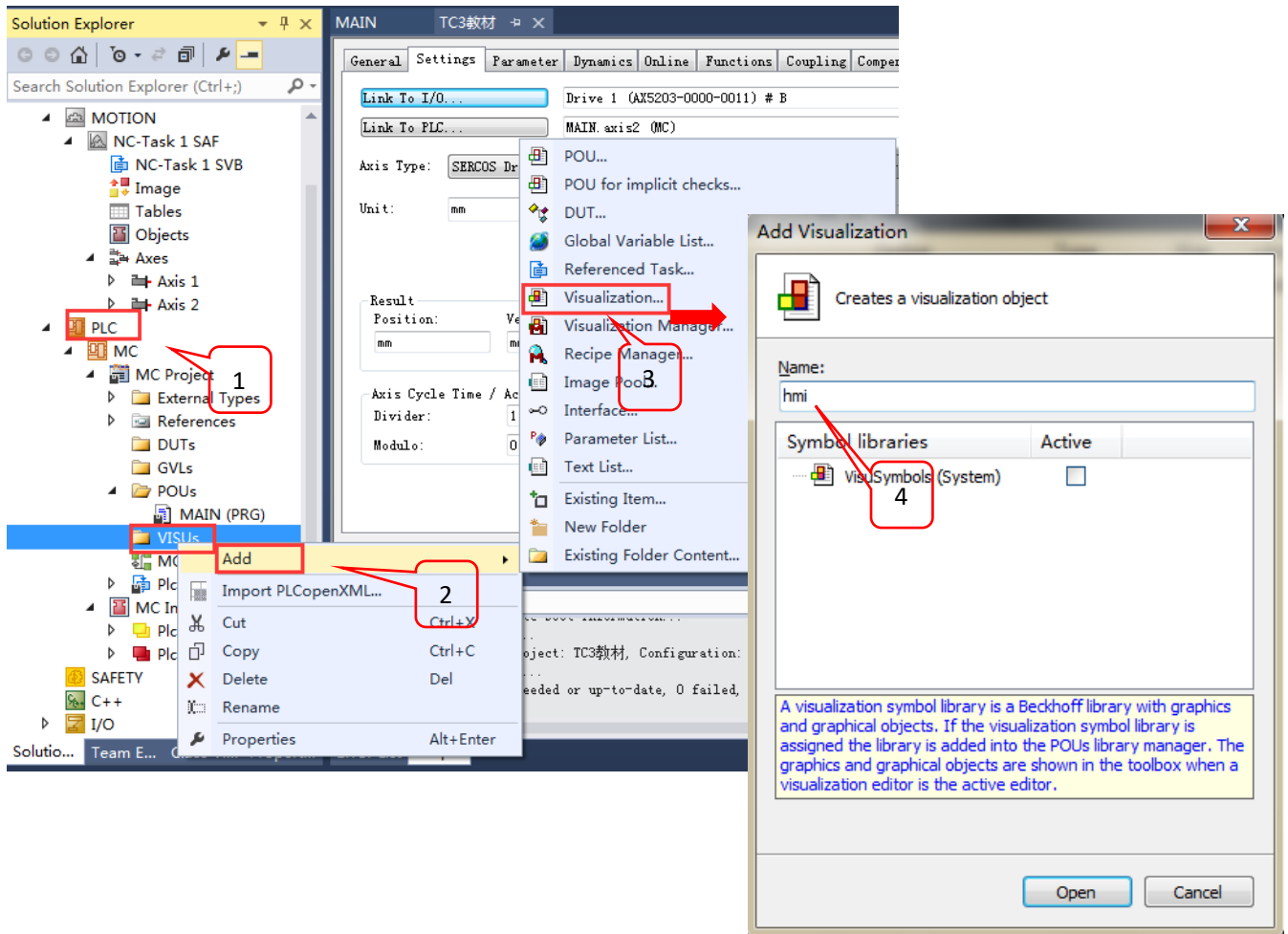
从 Motion 展开 Axes，找到 axis1,axis2 双击，从右边的界面中找到“Settings”选项卡下的 Link To PLC，将 axis1 链接上 MAIN.axis1(MC)，将 axis2 链接上 MAIN.axis2(MC)，NC 和 PLC 通过以上的链接交换数据，NC 将驱动器的状态位置反馈给 PLC，PLC 将功能块的控制数据写给 NC。



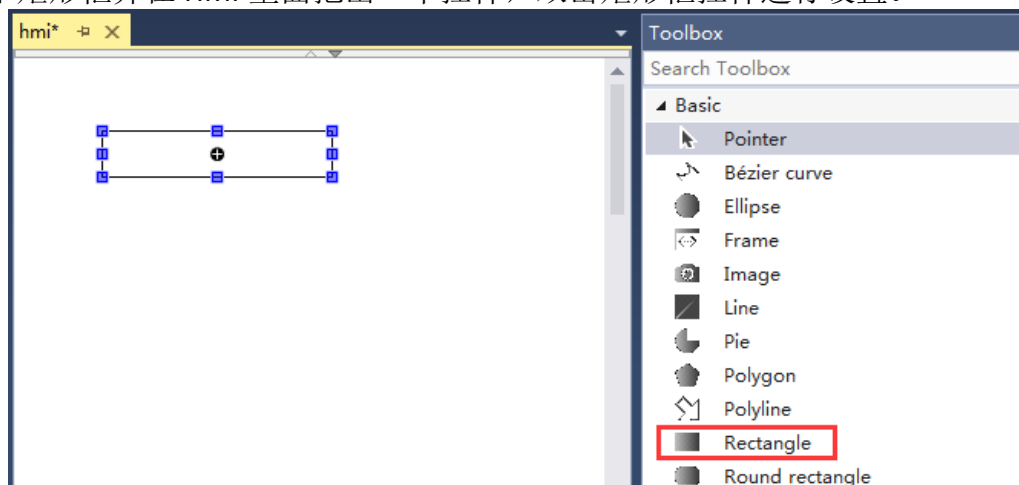
链接之后激活配置，并将 TwinCAT 重启为运行模式。

## 3. 调用功能块控制轴使能点动

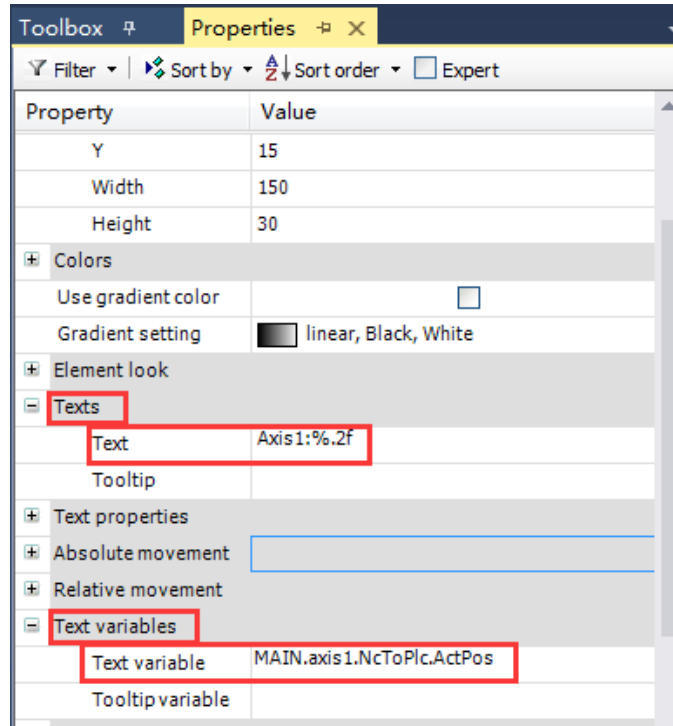
找到 PLC 下方的 VISUs 右击从弹出的菜单中单击 Add，再次从新的菜单中找到 Visualization，弹出对话框命名为 HMI。



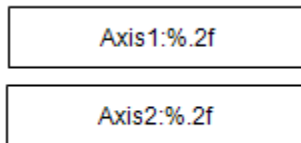
选中矩形框并在 HMI 里面拖出一个控件，双击矩形框控件进行设置。





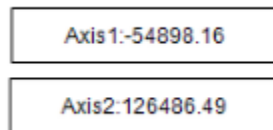
从右侧 Properties 选项卡下，将矩形框里面的参数设置如下：Texts——Text 里面输入 Axis1:%.2f, %.2f 代表以浮点数的数据类型显示关联变量（Text variables——Text variable 所指向的变量）的值，并只保留小数点后两位。Text variables 里面的 Text variable 中设置所关联的变量，这里选择轴 1 的实际位置。



再做一个矩形框控件用来显示轴 2 的实际位置。



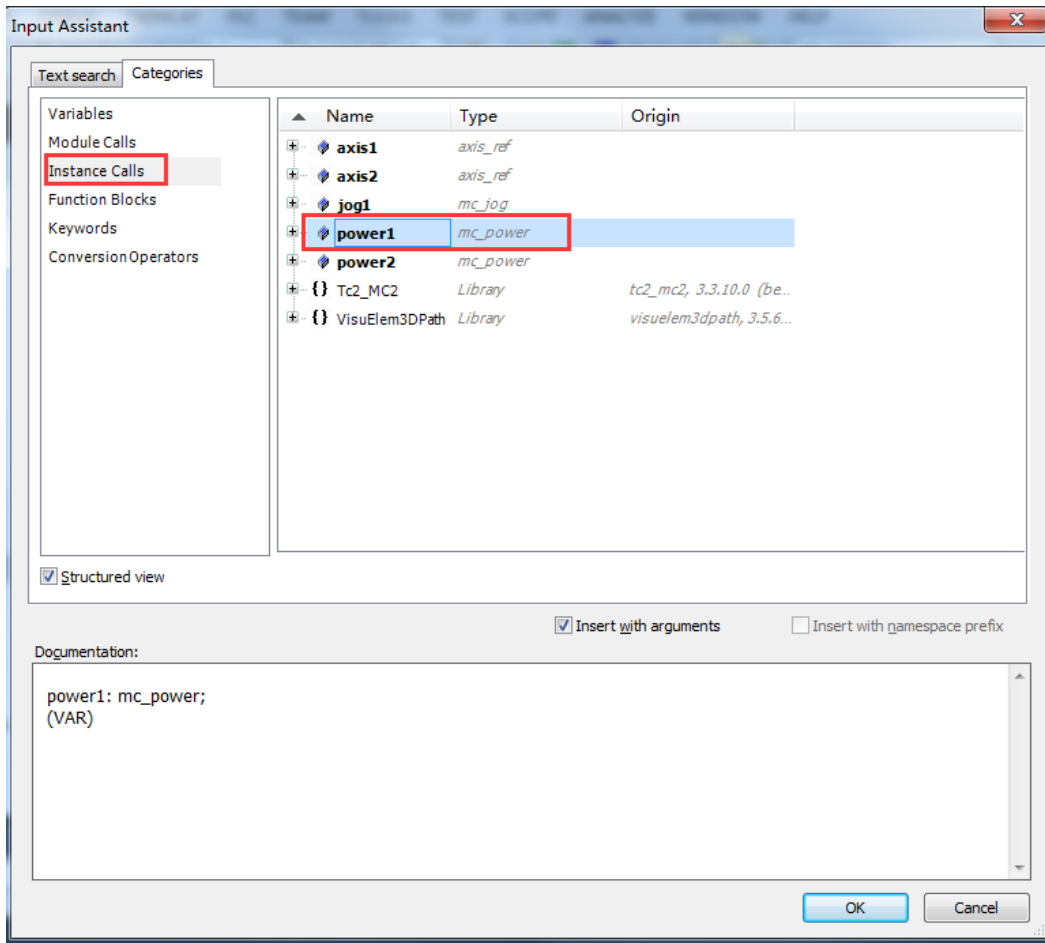
程序写好后需要 Login ，Login 之后把程序运行起来，点击运行按钮  即可看到轴 1 和轴 2 的当前位置显示在 HMI 上面。



回到 POU——MAIN(PRG)界面，声明两个 MC\_POWER 的功能块以及一个 MC\_JOG 的功能块，MC\_POWER 用来使能，MC\_JOG 用来点动。

```
PROGRAM MAIN
VAR
    axis1,axis2:axis_ref;
    power1,power2:mc_power;
    jog1:mc_jog;
END_VAR
```

在程序编写窗口中按 F2，在 Categories——Instance Calls 中选择 power1 点击 OK，将功能块调用到程序里面来，之后用同样的方法在程序中调用 Power2 与 Jog1 功能块。



将功能块里面的参数填写完整，Enable 代表使能触发位，Enable\_Positive 代表允许正转，Enable\_Negative 代表允许反转，Override 代表速度比，Axis 代表对哪个轴进行操作。JogForward 代表正向点动位，另外再声明两个 bool 类型变量（power\_do 和 jog\_for）做为使能与点动功能块的触发位。


```

power1(
  Axis:=axis1 ,
  Enable:= power_do,
  Enable_Positive:=TRUE ,
  Enable_Negative:= TRUE,
  Override:= ,
  BufferMode:= ,
  Options:= ,
  Status=> ,
  Busy=> ,
  Active=> ,
  Error=> ,
  ErrorID=> );

power2(
  Axis:= axis2,
  Enable:=power_do ,
  Enable_Positive:= TRUE,
  Enable_Negative:=TRUE ,
  Override:= ,
  BufferMode:= ,
  Options:= ,
  Status=> ,
  Busy=> ,
  Active=> ,
  Error=> ,
  ErrorID=> );

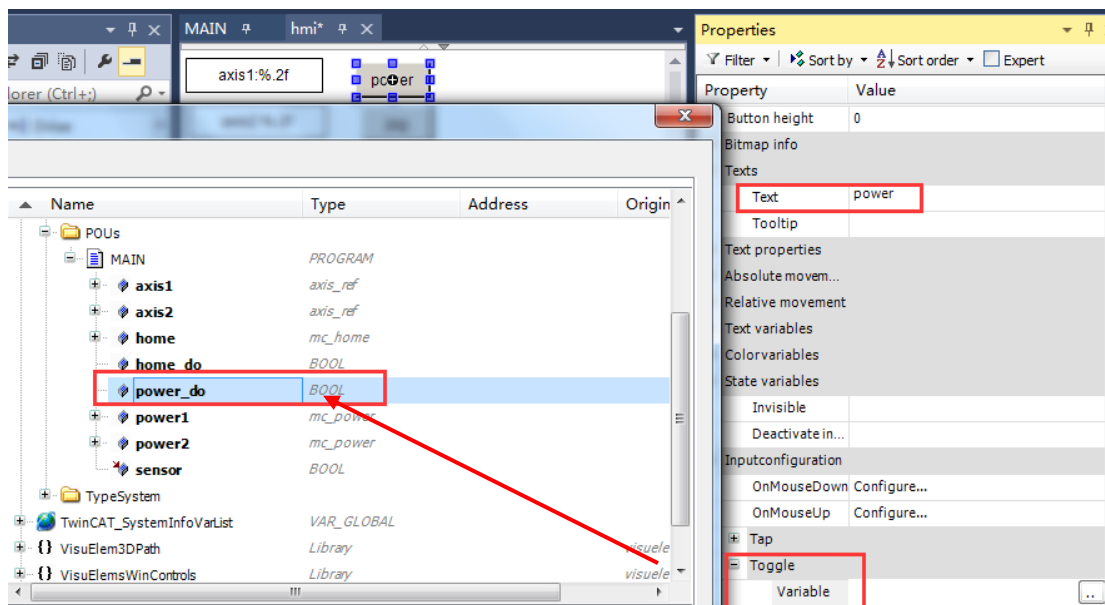
jog1(
  Axis:=axis1 ,
  JogForward:=jog_for
  JogBackwards:= ,
  Mode:= ,
  Position:= ,
  Velocity:= ,
  Acceleration:= ,
  Deceleration:= ,
  Jerk:= ,
  Done=> ,
  Busy=> ,
  Active=> ,
  CommandAborted=> ,
  Error=> ,
  ErrorID=> );

```


HMI 中加入两个按钮控件  Button ，用来对轴进行使能以及点动，两个按钮都设置成 inputconfiguration——Toggle（交替按钮），第一个按钮关联 MAIN.power\_do 变量，第二个按钮关联 MAIN.jog\_for 变量，在按钮的 Texts——Text 里面加上标签 Power 与

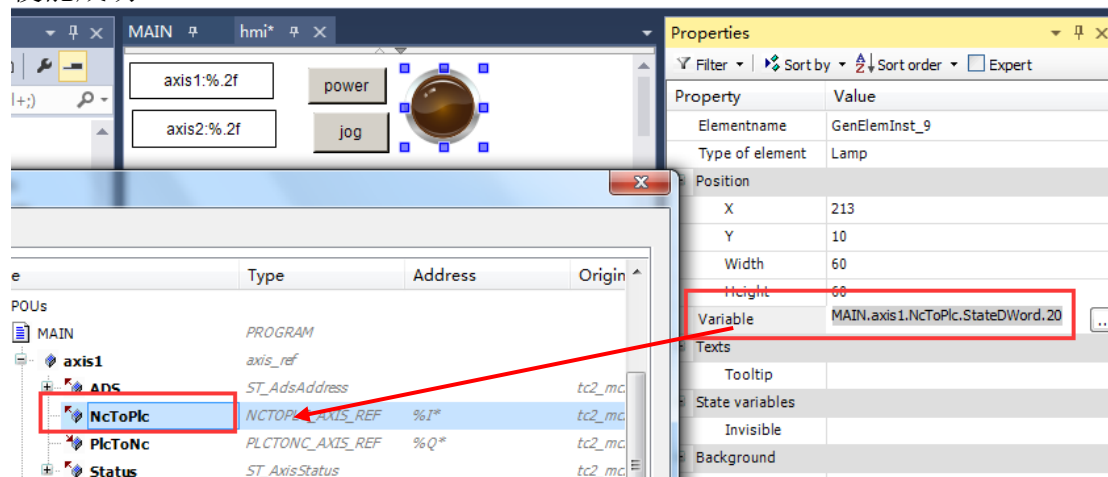


Jog。



注：由于 power 按钮按下后，需要返回到 Axis1,Axis2 的 Online 选项卡下查看轴是否成功使能（轴的 ready 状态有没有勾选来判断轴是否已经使能上了），这样操作十分麻烦，并且后续操作也会带来不便，所以在此，我们在 HMI 界面创建一个 LED 灯用以显示是否使能成功。

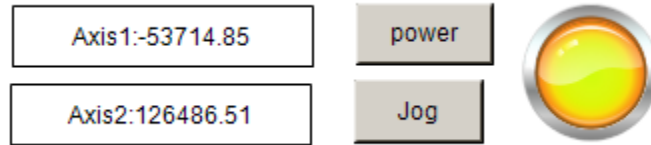
首先从右边的 Toolbox 拖出一个 LED 的图标  Lamp，然后将 Position——Variable 绑定 main.power1.NcToPlc.StateDWord.20 变量，用该变量显示 Power 功能块是否使能成功。



将 main.power1.NcToPlc.StateDWord 这个状态字转换成二进制后，它的第 20 位在成功使能之后会置 1。查询 Information System 软件，可知这一位的具体描述。

		20	0/1	ControlLoopClosed	ControlLoopClosed	Axis is ready for operation and axis control loop is closed (e.g. position control)
--	--	----	-----	-------------------	-------------------	---

Power 按钮按下，然后按下 Jog 即可看到轴在转动，再次按下 Jog 可以看到轴停止  
(注：先使能再点动，点动的时候不能撤除使能信号，否则轴会报错)

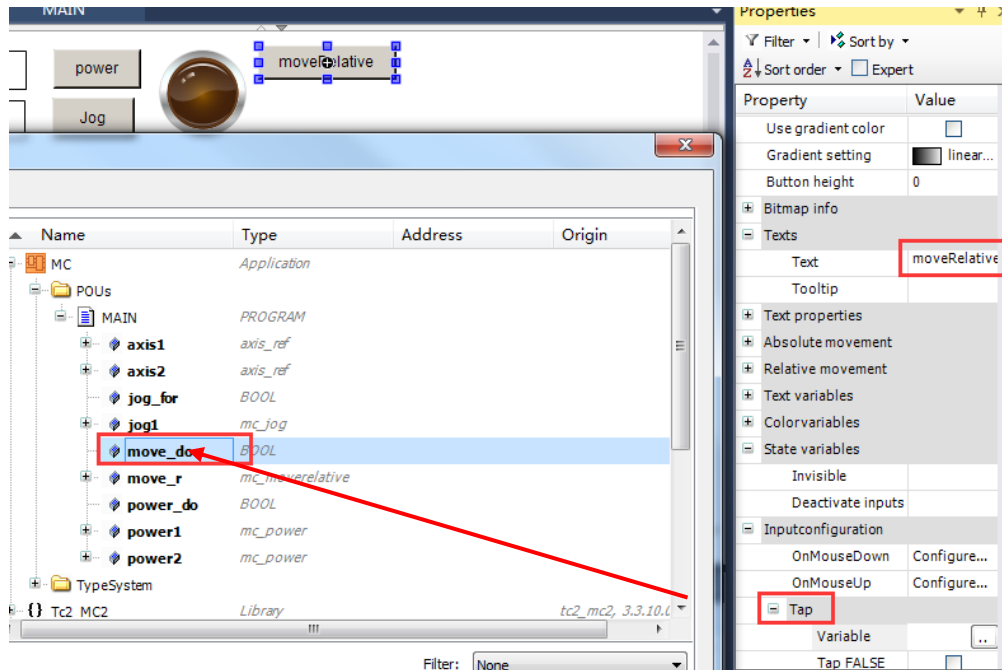


#### 4. 调用功能块控制轴走相对位置

声明一个 MC\_MoveRelative 功能块对轴进行位置控制，依然是按 F2 在程序编写窗口中调用功能块，然后将对应的参数填写完整，Execute 是功能块触发位，distance 和 velocity 是移动的距离以及速度，axis 代表控制哪根轴。

```
jog_for: BOOL;  
move_r:mc_moverelative;  
move_do: BOOL;  
  
move_r(  
  Axis:=axis1 ,  
  Execute:=move_do ,  
  Distance:=1000 ,  
  Velocity:=100 ,  
  Acceleration:= ,  
  Deceleration:= ,  
  Jerk:= ,  
  BufferMode:= ,  
  Options:= ,  
  Done=> ,  
  Busy=> ,  
  Active=> ,  
  CommandAborted=> ,  
  Error=> ,  
  ErrorID=> );
```

依然在 hmi 里面新建一个按钮，通过按钮来触发 Mc\_MoveRelative 功能块。



将程序 login 并 run，轴使能后按下 MoveRelative 按钮，即可让轴移动 1000 个位置，再次按下 MoveRelative，轴依然移动 1000 个位置。



## 5. 调用功能块控制轴改变当前位置

MC\_SetPosition 可以设置轴的当前位置，声明功能块后，在编程编写窗口中调用这个功能块，并将功能块的参数填写完整，Position 设置为 0，触发功能块即将当前位置改为原点。

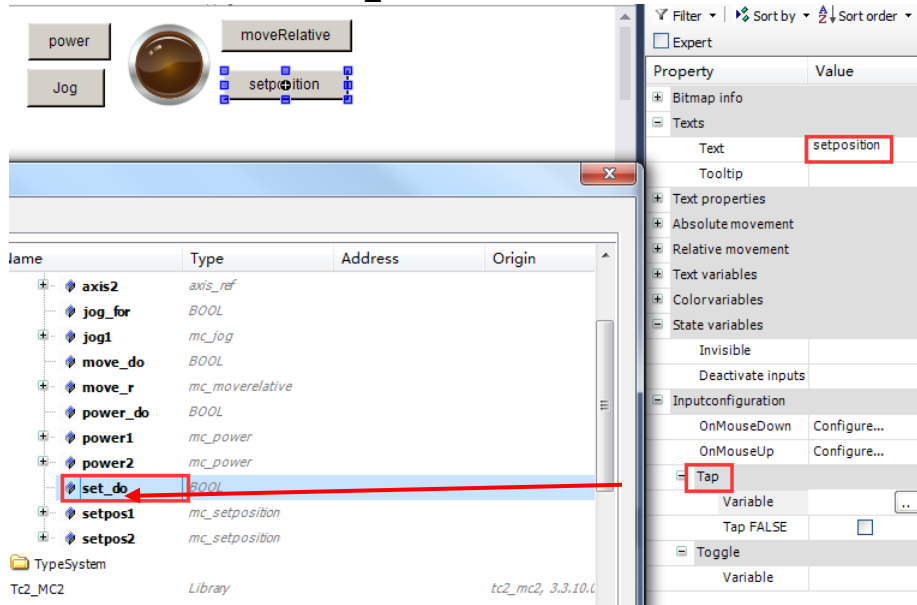
```

setpos1, setpos2: mc_setposition;
set_do: BOOL;

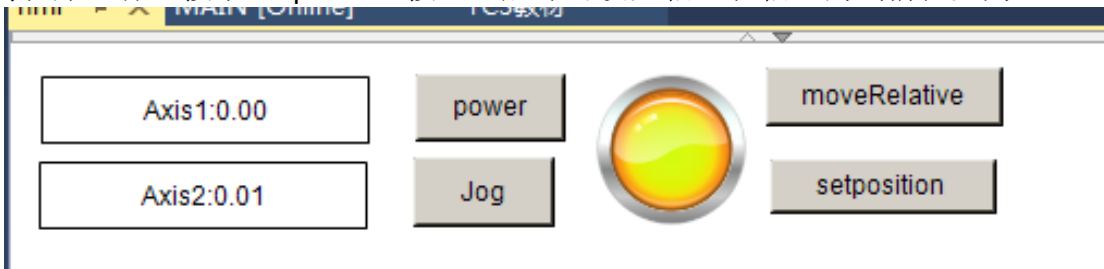
setpos1 (
  Axis:=axis1 ,
  Execute:=set_do ,
  Position:=0 ,
  Mode:= ,
  Options:= ,
  Done=> ,
  Busy=> ,
  Error=> ,
  ErrorID=> );

```

HMI 中加入一个按钮用来触发 Mc\_SetPosition 功能块。



将程序运行，按下 setposition 按钮之后即可设置轴 1 和轴 2 的当前位置为 0。



## 6. 调用功能块控制轴停止以及复位

MC\_Stop 对轴进行停止，MC\_Reset 对轴进行复位，声明之后在主程序中调用功能块，并将功能块的参数设置好。

```

set_do: BOOL;
stop:mc_stop;
reset:mc_reset;
stop_do: BOOL;
reset_do: BOOL;

```

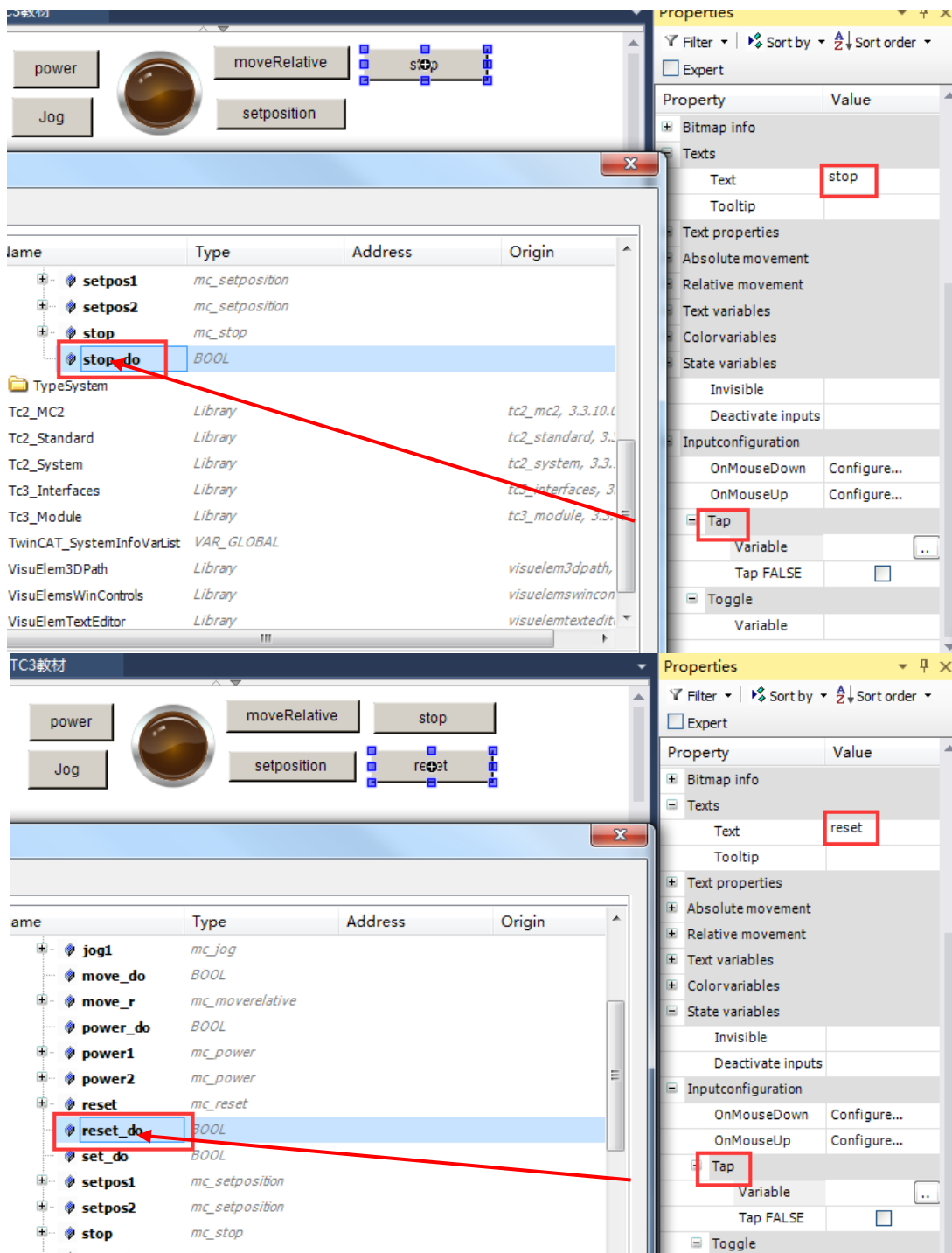
---

```

stop(
Axis:=axis1 ,
Execute:=stop_do ,
Deceleration:= ,
Jerk:= , |
Options:= ,
Done=> ,
Busy=> ,
Active=> ,
CommandAborted=> ,
Error=> ,
ErrorID=> );
reset(
Axis:=axis1 ,
Execute:=reset do ,
Done=> ,
Busy=> ,
Error=> ,
ErrorID=> );

```

HMI 里面加入两个按钮，一个对轴进行停止，一个对轴进行复位，当轴动作的时候可以通过 Stop 按钮来停止，当 NC 轴报错的时候可以通过 Reset 来复位。



## 7. 调用功能块控制两轴电子齿轮耦合

电子齿轮需要两个功能块，一个是耦合 MC\_GearIn,一个是解耦 MC\_GearOut, 分别将两个功能块调用到主程序中, RatioNumerator 代表从轴的速度, RatioDenominator 代表主轴的速度, 如 RatioDenominator 设为 2, RatioNumerator 设为 1, 那么主轴速度是从轴的两倍, MASTER 设置哪个轴为主轴, Slave 设置哪个轴为从轴。

```

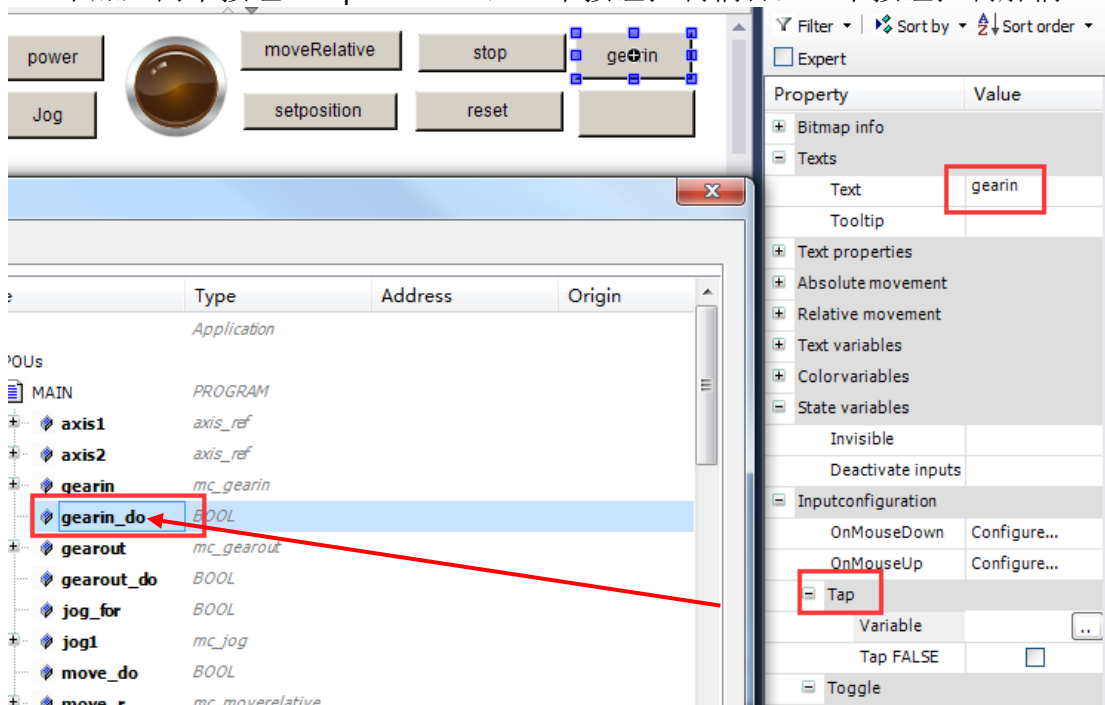
gearin:mc_gearin;
gearout:mc_gearout;
gearin_do: BOOL;
gearout_do: BOOL;

gearin(
  Master:= axis1,
  Slave:=axis2 ,
  Execute:= gearin_do,
  RatioNumerator:=1 ,
  RatioDenominator:=1 ,
  Acceleration:= ,
  Deceleration:= ,
  Jerk:= ,
  BufferMode:= ,
  Options:= ,
  InGear=> ,
  Busy=> ,
  Active=> ,
  CommandAborted=> ,
  Error=> ,
  ErrorID=> );

gearout(
  Slave:=axis2 ,
  Execute:=gearout_do ,
  Options:= ,
  Done=> ,
  Busy=> ,
  Error=> ,
  ErrorID=> );

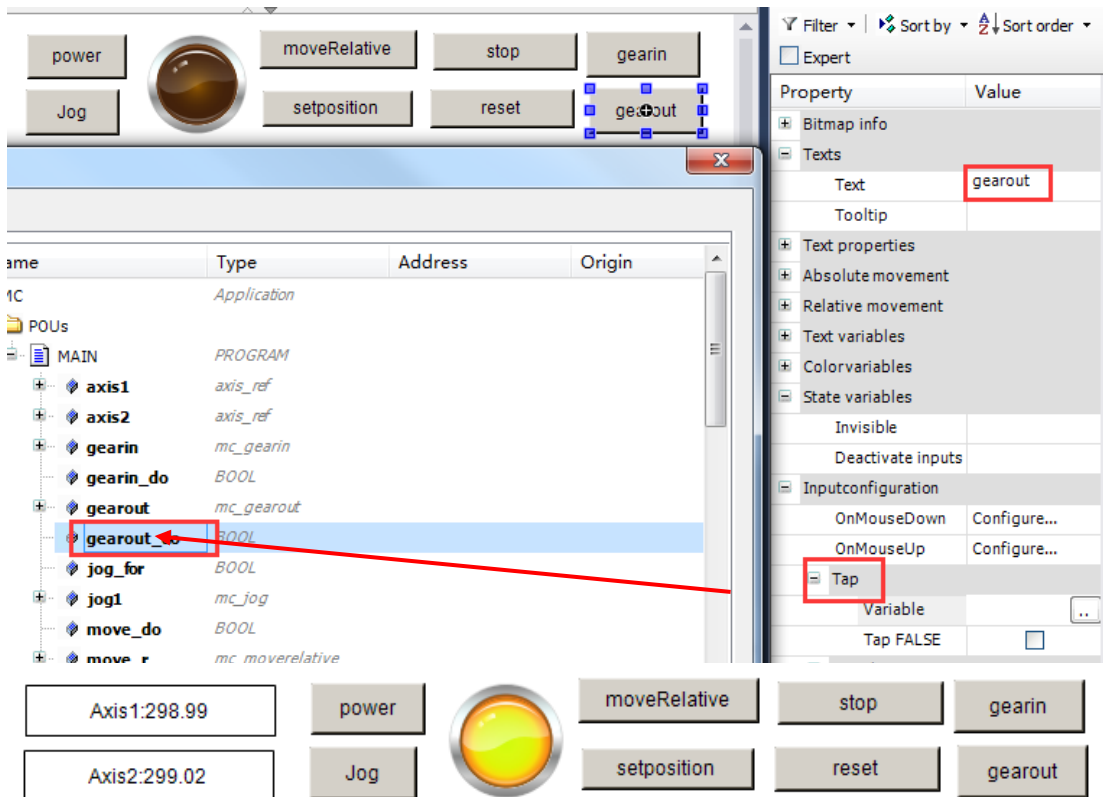
```

HMI 中加入两个按钮（Tap variable），一个按钮控制耦合，一个按钮控制解耦。



将程序登录进去并运行起来，首先通过 power 将两个轴使能，然后按下 GearIn 进行耦合，再按下 Jog 按钮后可以看到两个轴以 1:1 的速度转动，将 Jog 复位后，按下 GearOut 进行解耦。





### 8. 调用功能块控制轴寻参

MC\_home 功能块可以定位原点，home\_do 是功能块的触发位，sensor 是外部接近开关的触发信号，可以用 hmi 的按钮来代替，或者链接到外部输入点，当轴碰到接近开关信号之后，NC 轴的位置变为 Position 参数中设置的值。

```

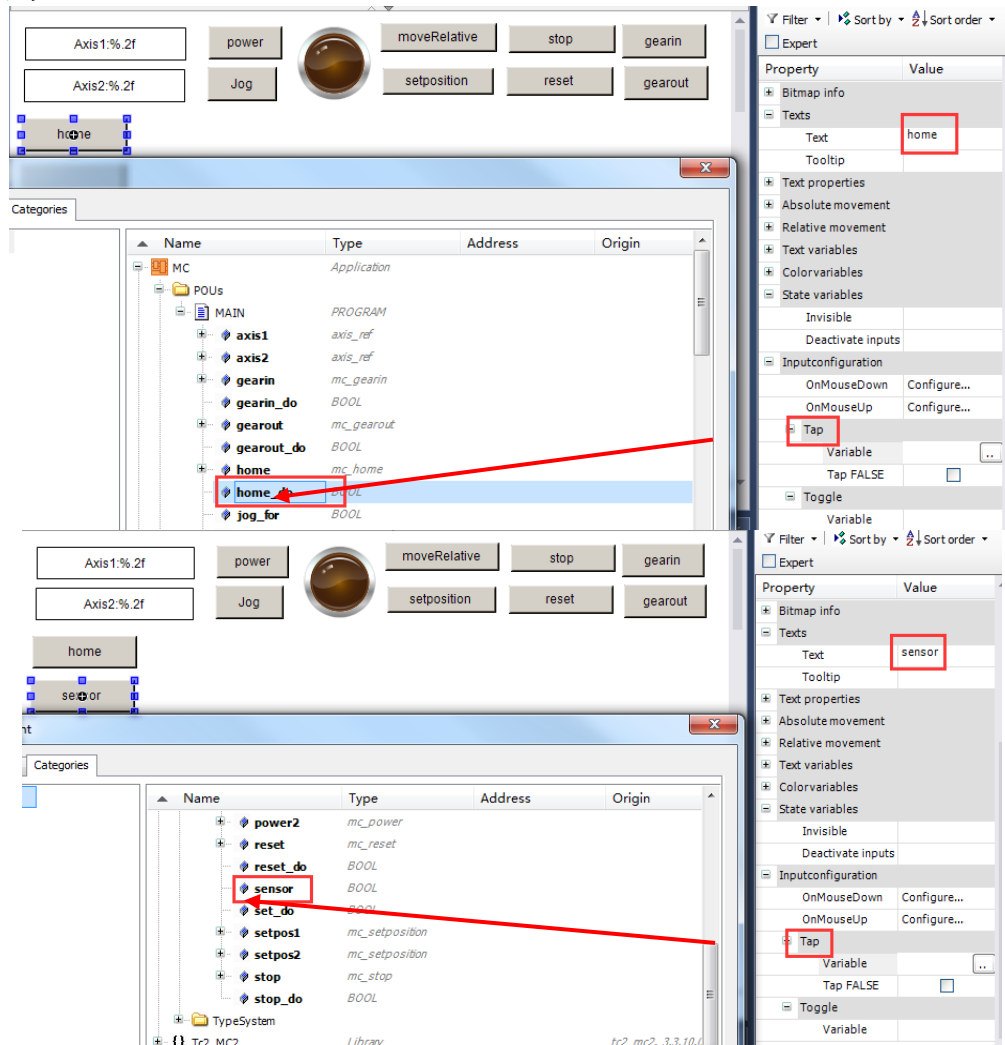
home:mc_home;
home_do: BOOL;
sensor: BOOL;

home(
  Axis:=axis1 ,
  Execute:=home_do ,
  Position:=0 ,
  HomingMode:= ,
  BufferMode:= ,
  Options:= ,
  bCalibrationCam:=sensor ,
  Done=> ,
  Busy=> ,
  Active=> ,
  CommandAborted=> ,
  Error=> ,
  ErrorID=> );

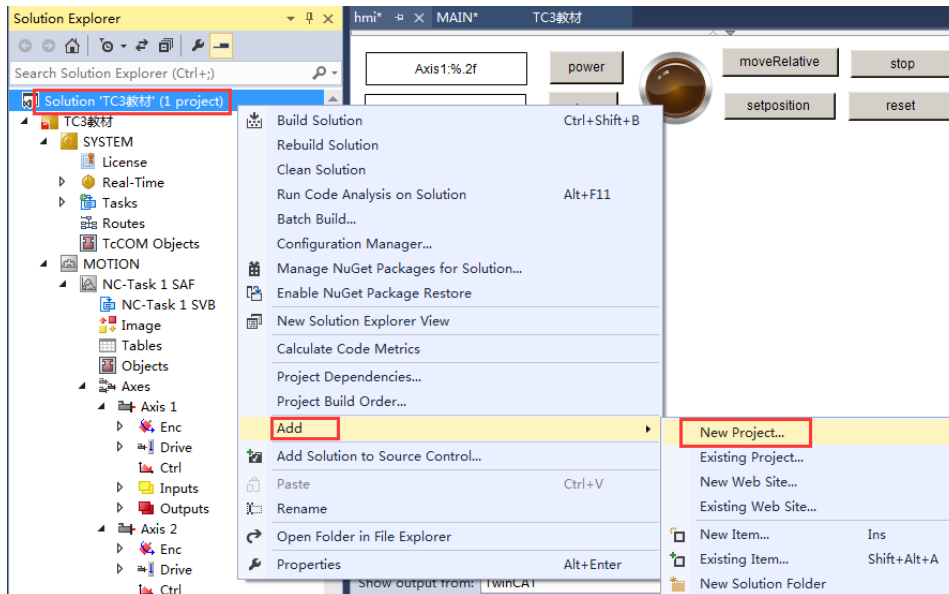
```

HMI 里面加入两个按钮（Tap variable），分别用来触发 MC\_home 功能块以及触发

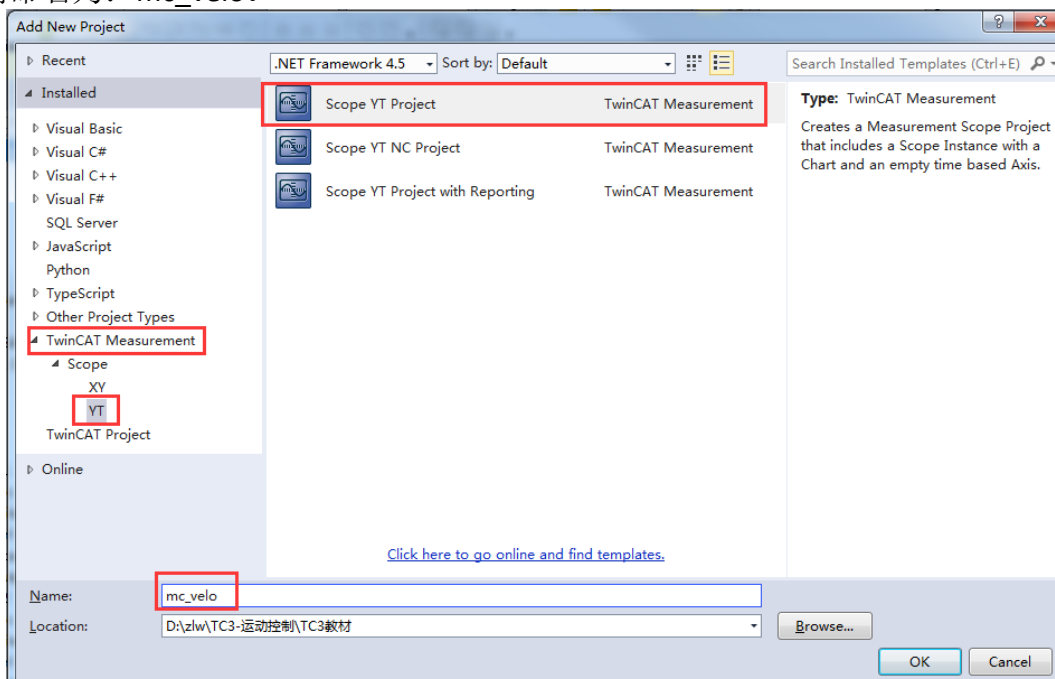
sensor 信号。



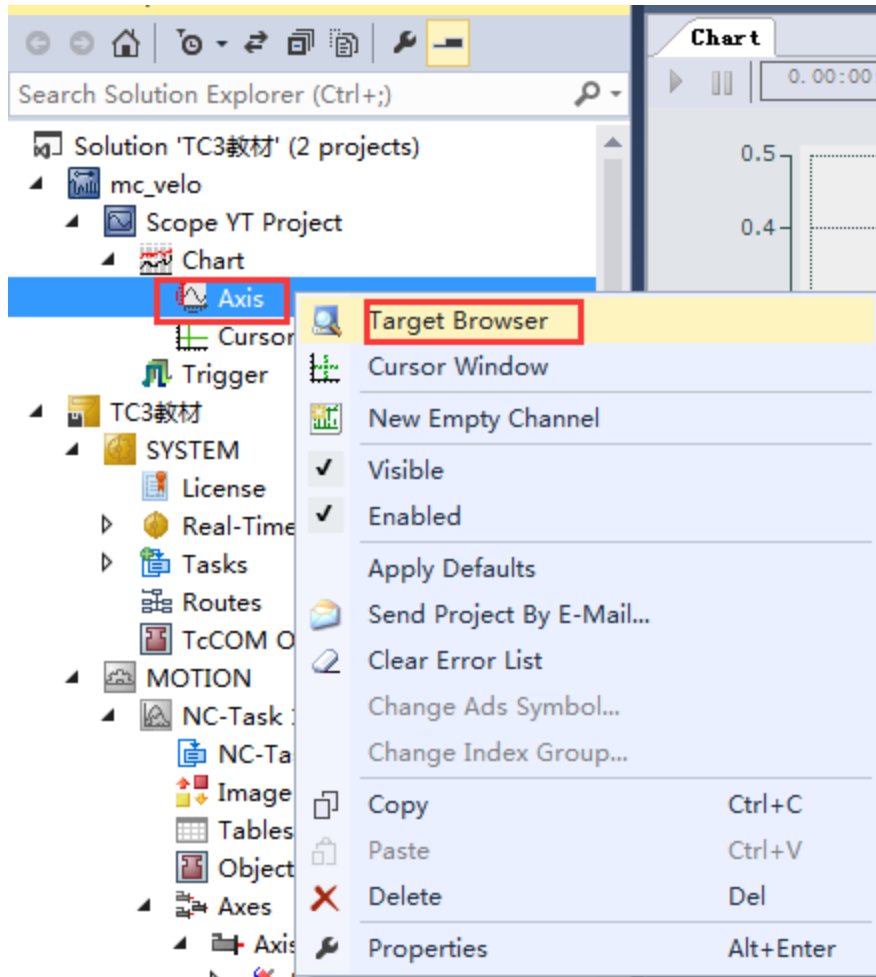
创建一个 Scope View，可以用来监视轴的速度变化。首先找到左侧对象管理器最上方“Solution\TC3 教材”处右击，找到 Add——New Project。



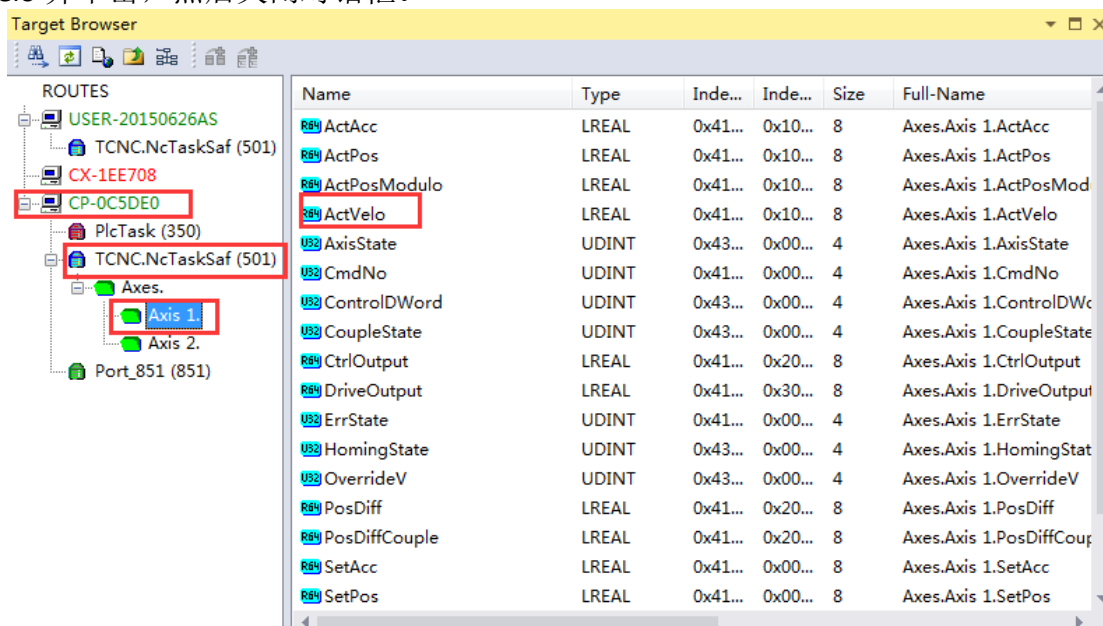
弹出的对话框找到 TwinCAT Measurement——YT——Scope YT Project，创建一个示波器，本实例命名为：mc\_velo。



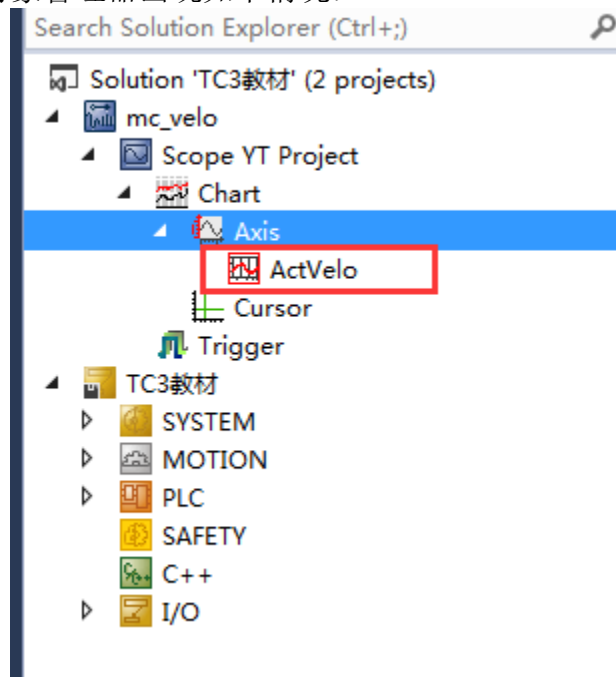
创建完成后，找到示波器下的 Axis 右击，单击 Target Browser。



从弹出的对话框中，找到所连接的控制器主机名（本实例控制器的主机名为 CP-0C5DE0），再找到 TCNC.NcTaskSaf(501)—Axis1，从右侧 NC 反馈回来的轴状态字中找到 ActVelo 并单击，然后关闭对话框。

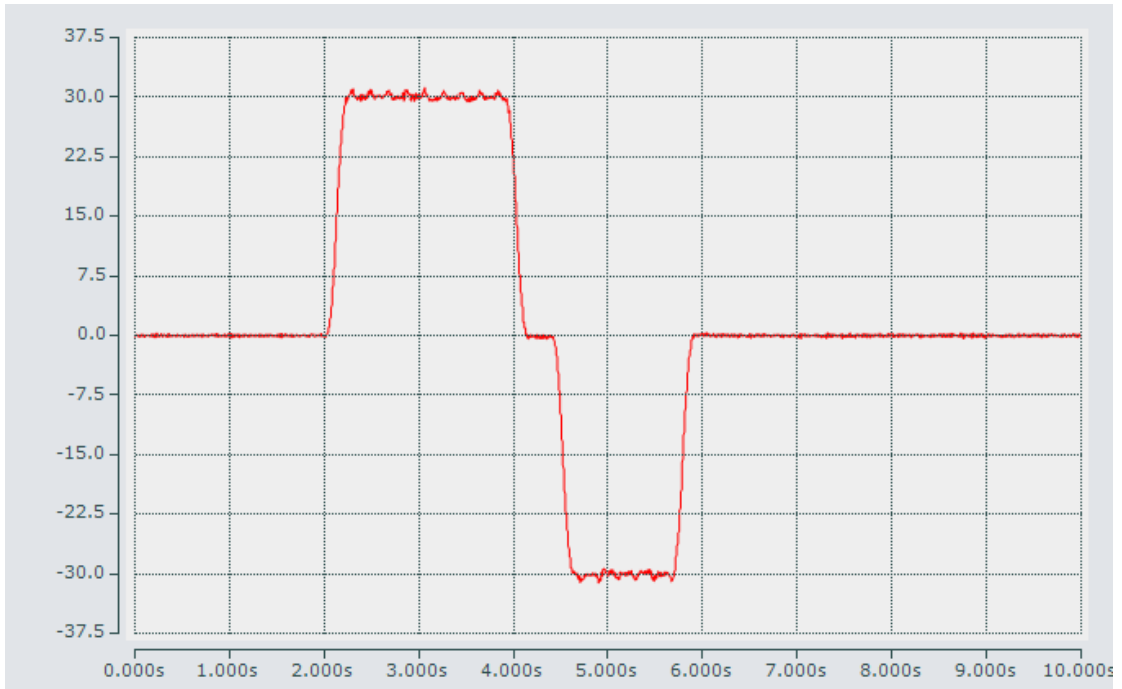


此时可以看见左侧对象管理器出现如下情况：



将程序 Login 之后，首先按下 Record 按钮，使用 Scope View 记录轴的速度，然后在 HMI 界面按下 power 按钮对轴进行使能，接着按下 Home 按钮，此时轴开始找寻原点，速度为 30 左右，当前位置变为-9999999999，然后按下 sensor 按钮不要放开，观察此时的轴会停止且反转，在放开 sensor 按钮的那一刻，将位置定为原点，找原点的流程如下：正向找原点=>碰到原点信号=>原点信号从 0 变为 1=>电机停止并反转=>脱离原点信号=>原点信号从 1 变为 0=>寻参完成且轴当前位置变成 0。





## 四、电子凸轮表功能

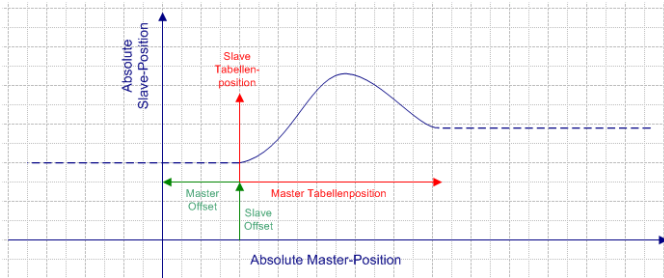
电子凸轮（Electronic CAM），是利用构造的凹轮曲线来模拟机械凸轮，以达到机械凸轮系统相同的凸轮轴与主轴之间相对运动的软件系统。在机械加工方面，用电子凸轮来代替笨重的机械凸轮。采用电子凸轮的系统具有更高的加工精度和灵活性，提高生产效率。

电子凸轮表是用来表示主轴与从轴的位置保持对应关系。这个对应关系是通过凸轮表（Cam Table）来表示的。在 Motion 下方提供了凸轮绘制界面（此功能需要 License，否则绘制的凸轮表在设备重启后会丢失，也可以通过 Mc\_CamTableSelect 功能块在程序中创建凸轮表）。

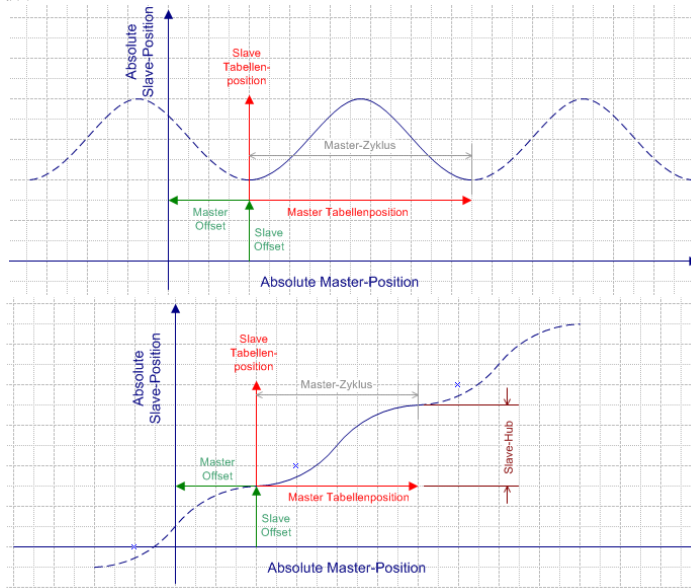
### （一）电子凸轮的分类：

电子凸轮表根据其周期性可以分成：

#### 1. 非循环性（单周期）



#### 2. 循环性（多周期）



电子凸轮表根据其灵活性可以分成：

#### 1. 固定的位置凸轮表（PositionTable）

该方式是通过外部工具进行计算，不受限制的位置曲线规划，相比较 Motion Function 缺少一定的灵活性，没有在线修改的能力。

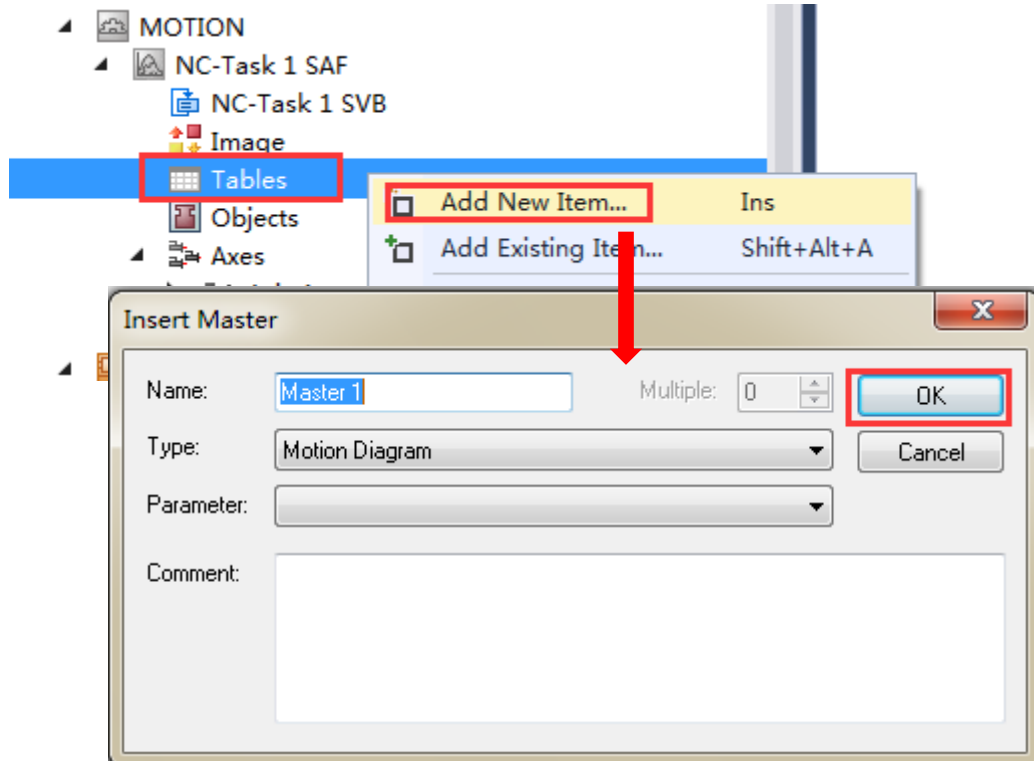
## 2. 可修改的位置凸轮点（MotionFunction）

该方式是非常灵活的；可以在运动的过程中进行在线修改由几个重要的凸轮点组成的凸轮曲线；两个凸轮点之间的曲线类型是固定的,不可变的。

### （二）电子凸轮的创建

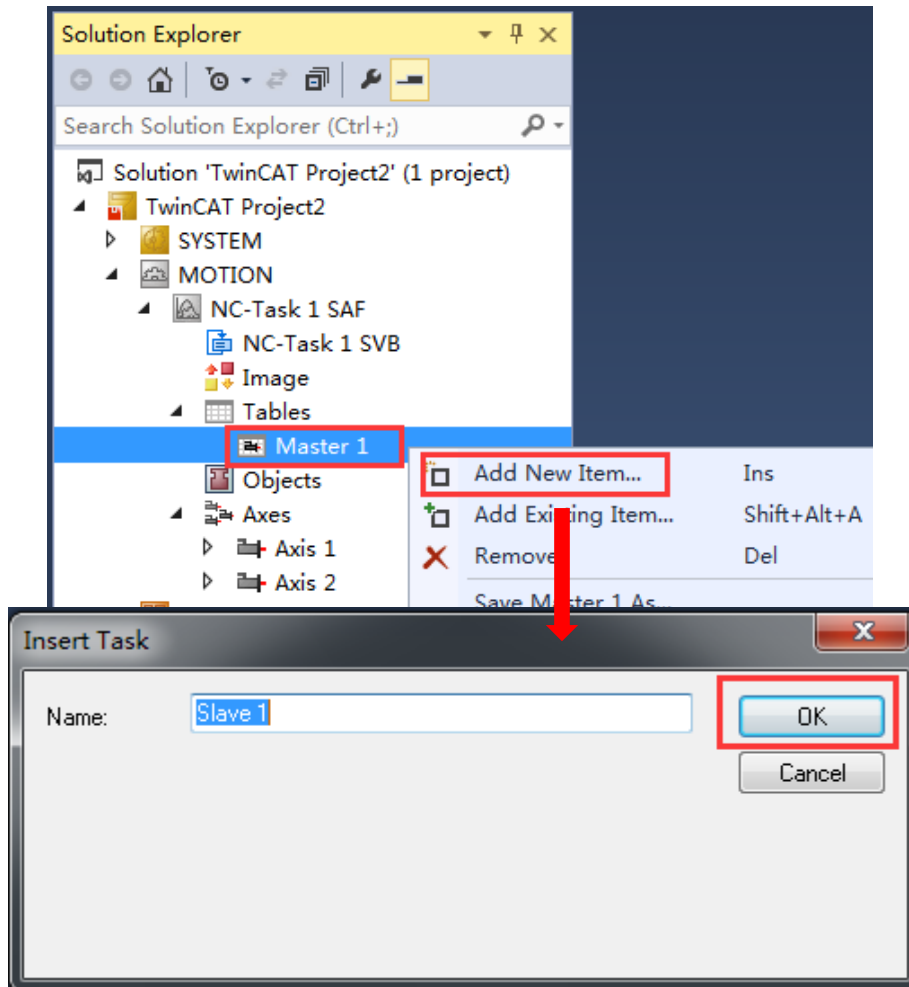
#### 第一种，通过 TwinCAT 凸轮编辑工具

1. 在 Motion 下方找到“Tables”右键，选择“Add New Item”，弹出的对话框点击“OK”。

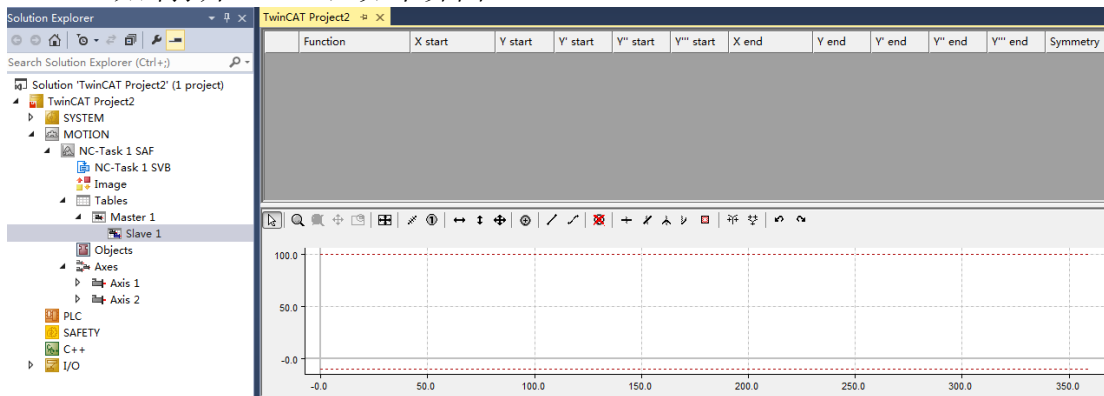


2. 找到刚才新建的“Master1”右键，选择“Add New Item”，弹出的对话框点击“OK”。







3. 双击打开 slave 1，如下界面：



图中按钮功能如下：

- ：添加关键点；
- ：删除关键点；



: 移动关键点;



: 以直线或者自动平滑曲线连接关键点;



: 视图缩放, 取消缩放, 视图平移, 左上角小窗口显示。

其中: 添加点是可定义点的属性,



Rest points 速度为 0, 加速度为 0



Velocity points 速度不为 0, 加速度为 0



Motion Points 速度不为 0, 加速度不为 0

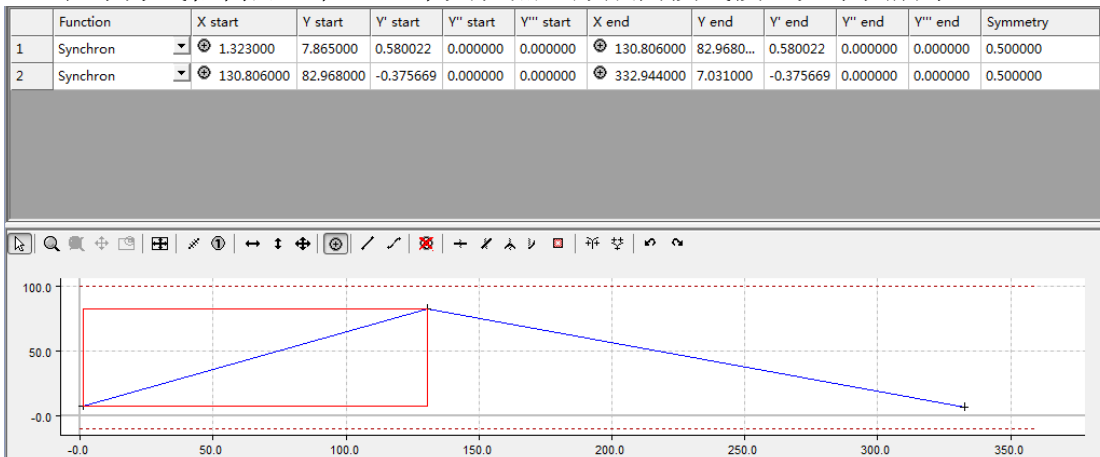


Turn points 速度为 0, 加速度不为 0

具体特性见下表

Transition		to			
from		Rest R	constant Velocity V	Turn T	Motion M
Rest R	R	 R-R RR-Laws, P5MM, P7MM	 R-V P5MM, P7MM	 R-T RT-Laws, P5MM, P7MM	 R-M P5MM, P7MM
constant Velocity V	V	 V-R P5MM, P7MM	 V-V ModSineLine_VV, P5MM, P7MM	 V-T VT-Laws, P5MM, P7MM	 V-M P5MM, P7MM
Turn T	T	 T-R TR-Laws, P5MM, P7MM	 T-V TV-Laws, P5MM, P7MM	 T-T SineSyncCombi, P5MM, P7MM	 T-M P5MM, P7MM
Motion M	M	 M-R P5MM, P7MM	 M-V P5MM, P7MM	 M-T P5MM, P7MM	 M-M P5MM, P7MM

在下方线框内加入任意三个关键点, 形成两段线段, 如下图所示。



4. 修改关键点的位置，如下图所示；（当主轴从 0->180->360 的过程中，从轴的位置 0->100->0）

	Function	X start	Y start	Y' start	Y'' start	Y''' start	X end	Y end	Y' end	Y'' end	Y''' end	Symmetry
1	Synchron	0.000000	0.000000	0.555556	0.000000	0.000000	180.000000	100.000...	0.555556	0.000000	0.000000	0.500000
2	Synchron	180.000000	100.000...	-0.555556	0.000000	0.000000	360.000000	0.000000	-0.555556	0.000000	0.000000	0.500000

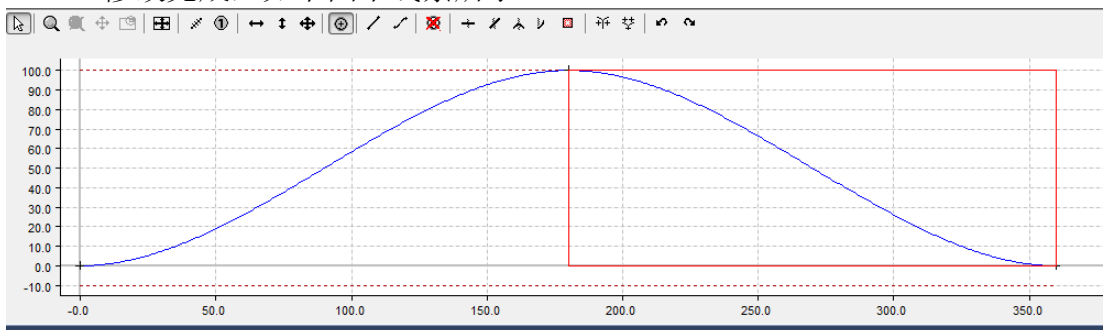


并将两段线段的线形修改成为三次多项式；

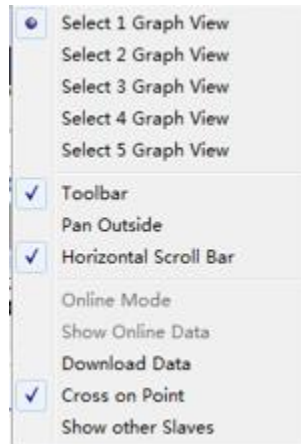
	Function	X start	Y start	Y' start	Y'' start	Y''' start	X end	Y end	Y' end	Y'' end	Y''' end	Symmetry
1	Synchron	0.000000	0.000000	0.555556	0.000000	0.000000	180.000000	100.000...	0.555556	0.000000	0.000000	0.500000
2	Synchron	180.000000	100.000...	-0.555556	0.000000	0.000000	360.000000	0.000000	-0.555556	0.000000	0.000000	0.500000

Polynom3

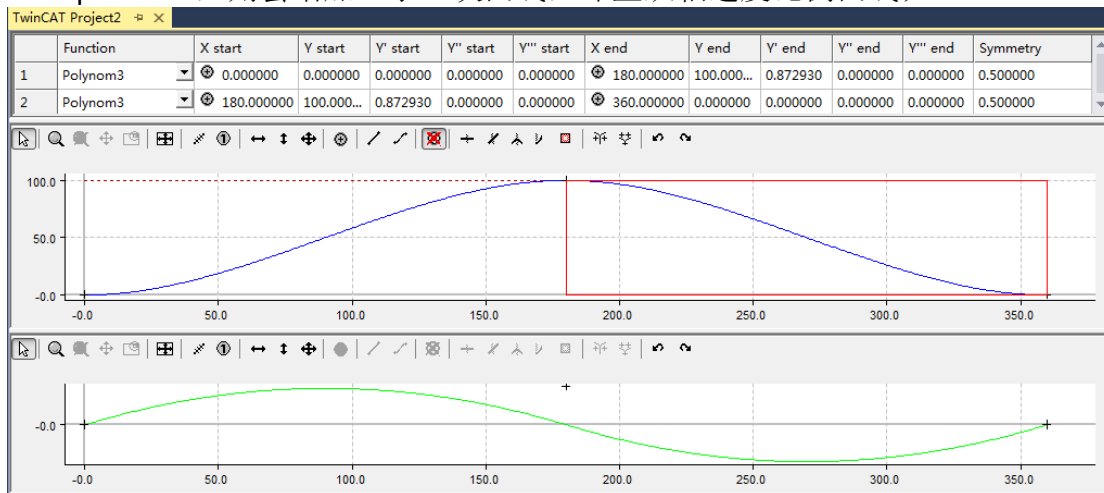
修改完成，如下图中线条所示：



在线框内右键，显示出的快捷菜单如下所示：



默认勾选“Select 1 Graph View”，显示一次曲线，即主从轴位置对应曲线；勾选“Select 2 Graph View”，则会增加显示二次曲线，即主从轴速度比例曲线；

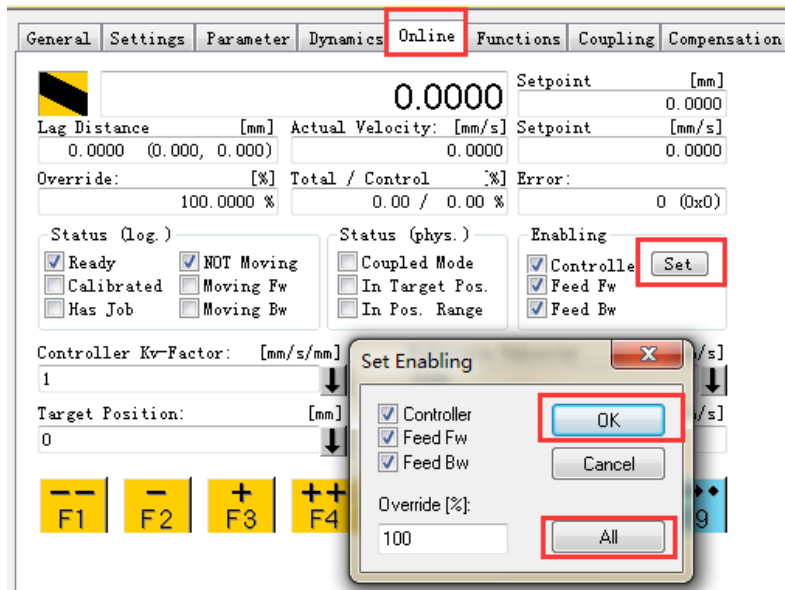


同理，如果勾选“Select 3 Graph View”，显示三次曲线，即主从轴加速度对应曲线，以此类推“Select 4 Graph View”，显示四次曲线加加速度曲线。

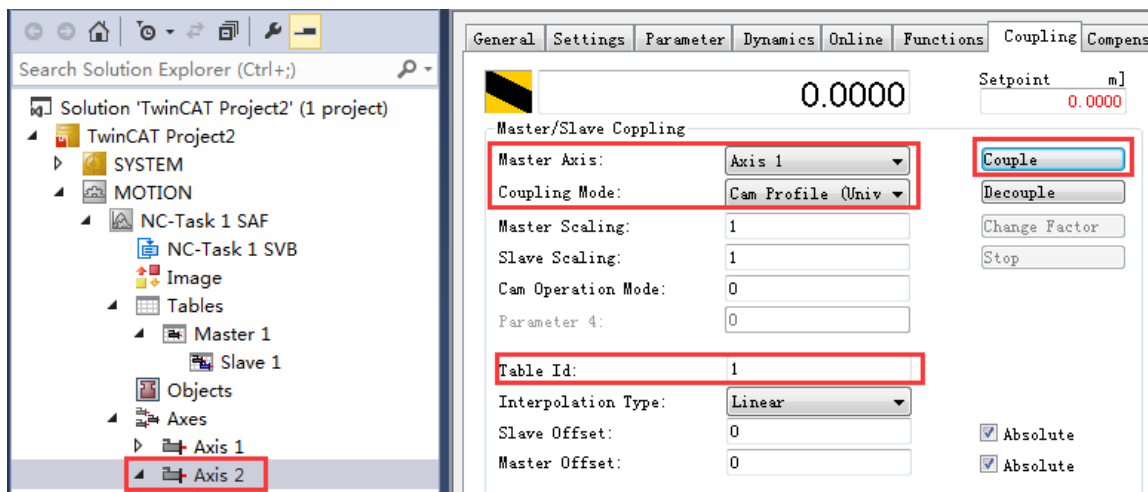
至此，一个凸轮表就建好了。现可以激活配置，将创建好的电子凸轮表写入配置，并且将 TwinCAT 切至运行模式。

#### 5. 调试操作：

首先将 Axis1,Axis2 使能，如下图：

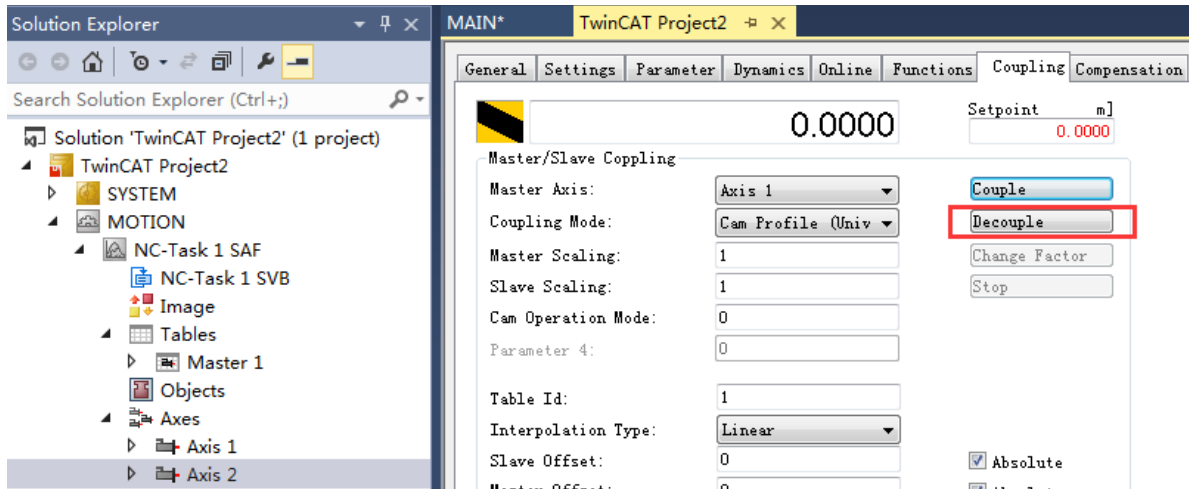


并将 Axis1 作为主轴，Axis2 作为从轴。单击 Axis2，然后找到 coupling 选项卡，Master Axis 为 Axis1，选择 Coupling Mode 耦合方式：Cam Profile 电子凸轮表，并将 Table ID 设置成 1，相当于选择 table-master 1-slave 1 中编辑的电子凸轮表，然后点击“Couple”（注意耦合成功后上方的 Setpoint 中当前位置值会变成红色，作为从轴的 Axis2 将不能再进行单独调试，只能跟随 Axis1 进行电子凸轮关系的运动）。

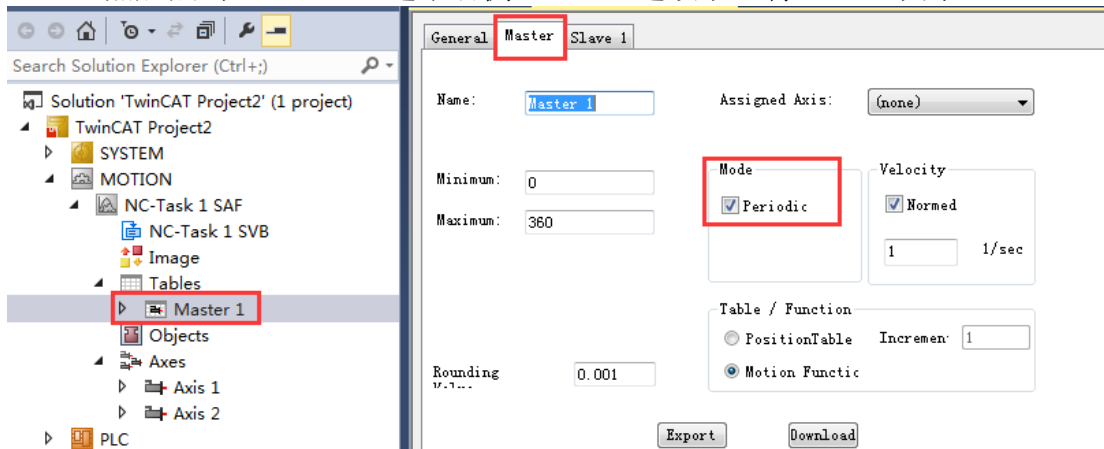


当对 Axis1 轴做正向点动，Axis2 轴的位置会从 0→100→0，Axis1 超过 360 后，Axis2 将停止。这是因为 Master 的属性未设置成周期性，如需要 Axis1 在正向点动或者其他运动的时候，Axis2 轴始终是循环的从 0→100→0，需做以下操作。

首先将其解耦，如下图所示：



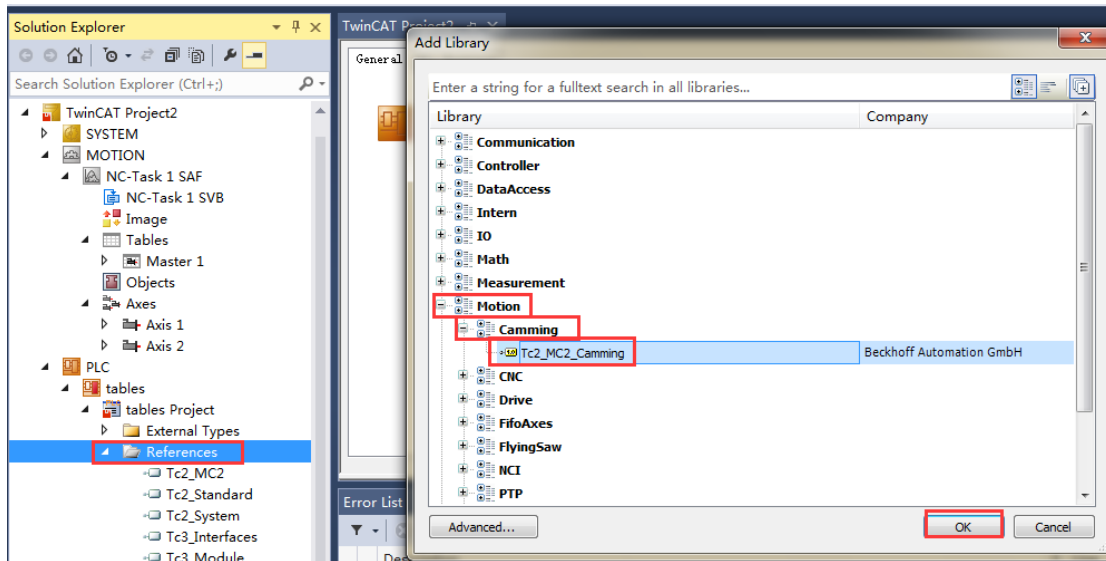
然后点击 Master 1，选中右侧“Master”选项卡，将 Mode 下的“Periodic”勾选上。



完成以上操作，激活配置，在对 Axis1 轴做正向点动时，Axis2 轴的位置会从 0→100→0 周期性运动，不再停止。

## 6. 编程操作：

首先利用 PLC 程序控制电子凸轮表运动，需要在基础的 Tc2\_MC2.lib 这个基础功能库上添加 Tc2\_MC2\_Camming.lib。



添加完功能库后，定义两个轴变量，两个使能功能块，一个点动功能块，以及电子凸轮表耦合和解耦功能。

```

PROGRAM MAIN
VAR
    axis1,axis2:    axis_ref;
    power1,power2: mc_power;
    jog1:          mc_jog;
    camin:         mc_camin;
    camout:        mc_camout;

```

对于使能功能块以及点动功能块的编程操作按照教材前面所述来操作，电子凸轮表耦合和解耦按照下图所示操作。

耦合功能块 **Mc\_Camin**: Master 绑定电子凸轮表的主轴，Slave 绑定电子凸轮表的从轴，变量 **camin\_do** (bool) 作为该功能块的触发位，**CamTableID** 是所选择的电子凸轮表的ID，此时选择的是 **Slave 1**。

```

camin(
  Master:=axis1 ,
  Slave:= axis2,
  Execute:= camin do,
  MasterOffset:= ,
  SlaveOffset:= ,
  MasterScaling:= ,
  SlaveScaling:= ,
  StartMode:= ,
  CamTableID:= 1,
  BufferMode:= ,
  Options:= ,
  InSync=> ,
  Busy=> ,
  Active=> ,
  CommandAborted=> ,
  Error=> ,
  ErrorID=> );

```

解耦功能块 Mc\_Camout: Slave 绑定从轴 Axis2, 变量 camout\_do (bool) 作为该功能块的触发位。

```

camout(
  Slave:= axis2,
  Execute:= camout_do,
  Options:= ,
  Done=> ,
  Busy=> ,
  Error=> ,
  ErrorID=> );

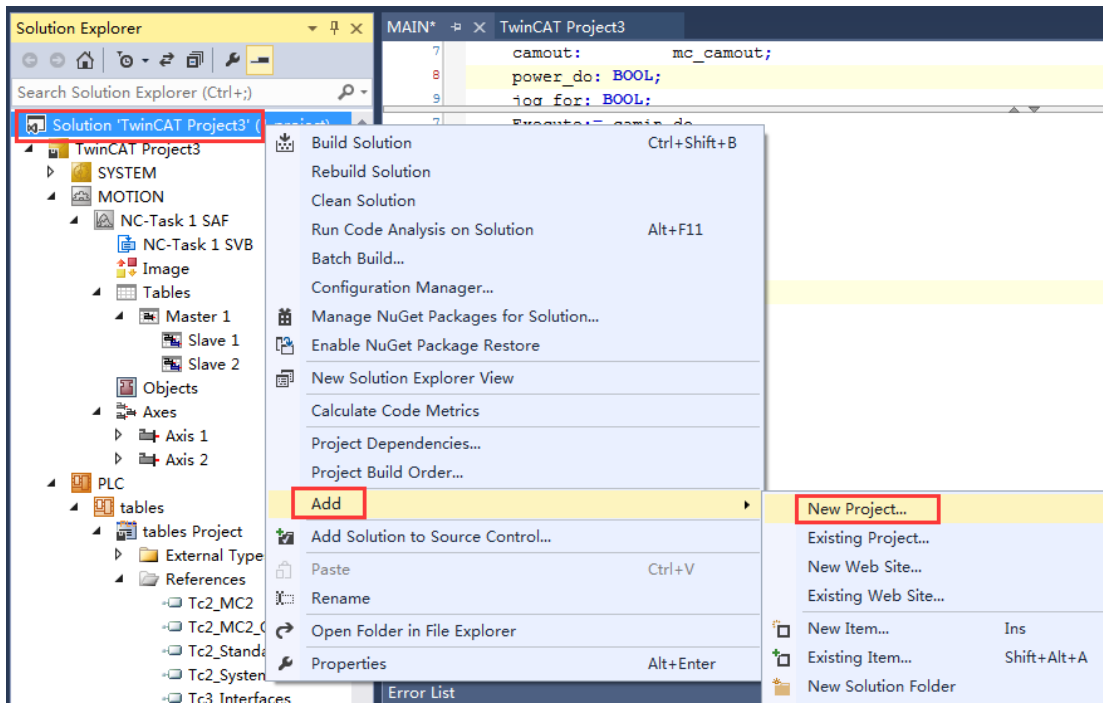
```

完成 PLC 编程，然后创建 Scope View 来监视 Axis1 和 Axis2 两轴在进行电子凸轮表功能时各自的位置变化曲线。

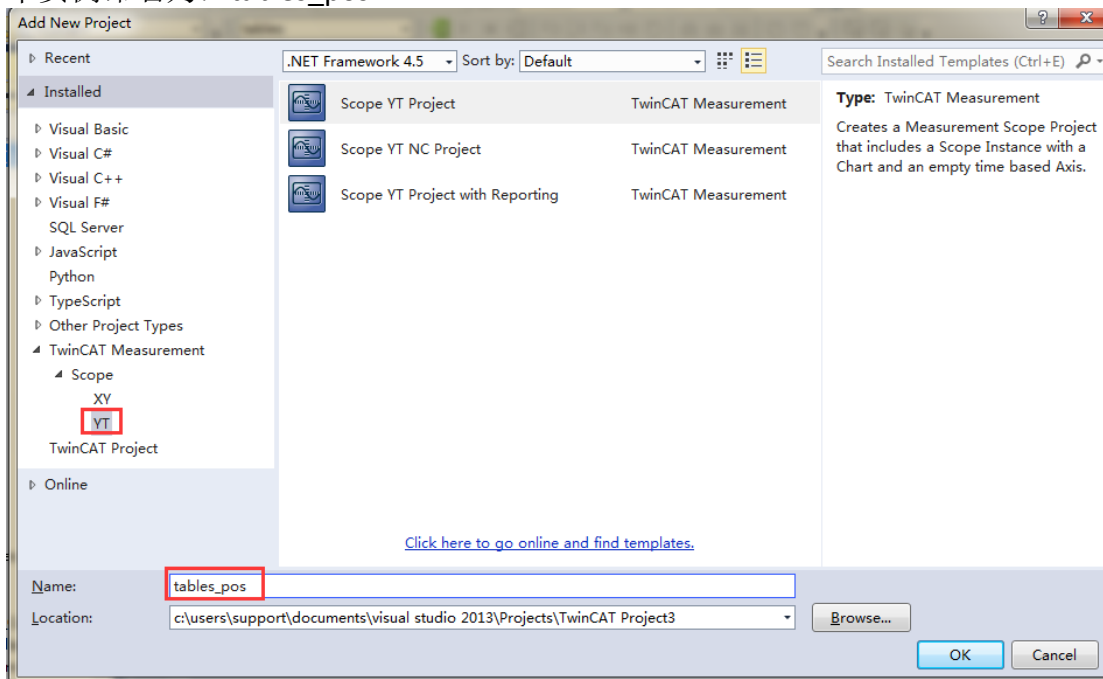
#### 7. Scope View 来监视 Axis1 和 Axis2 位置变化

首先找到左侧对象管理器最上方“Solution'TwinCAT Project3”处右击，找到 Add——New Project。

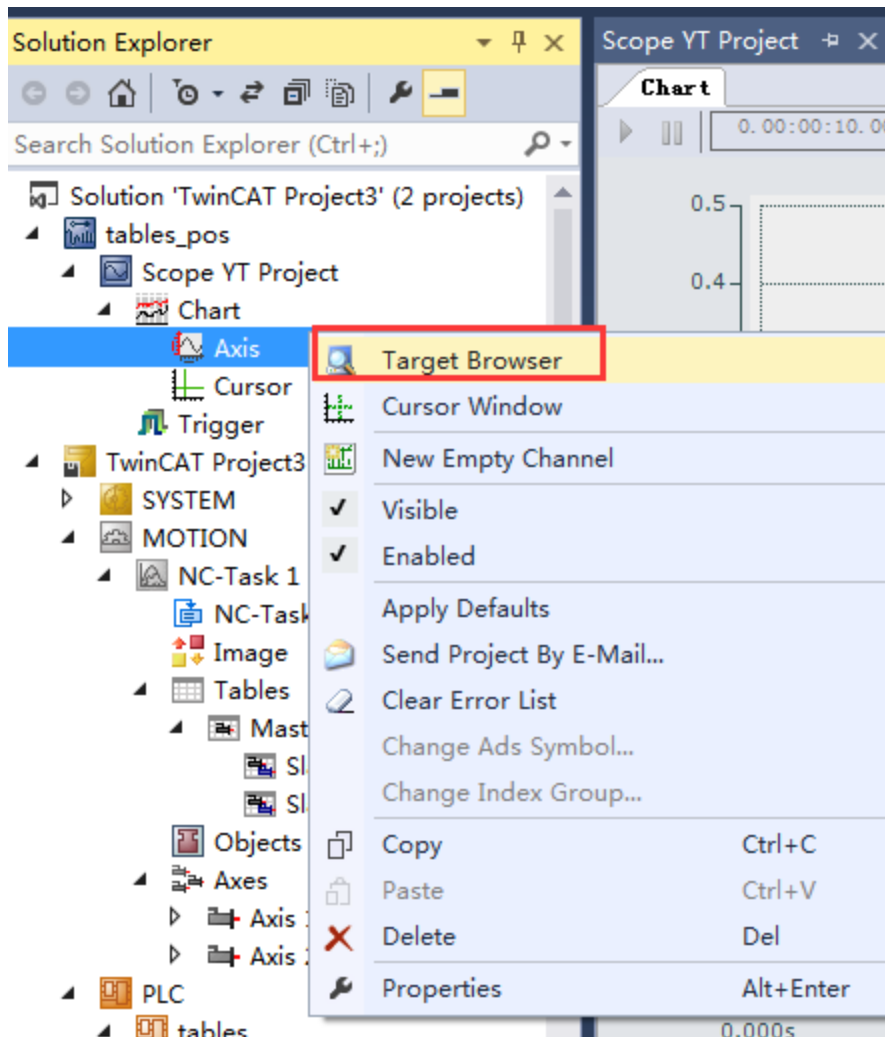




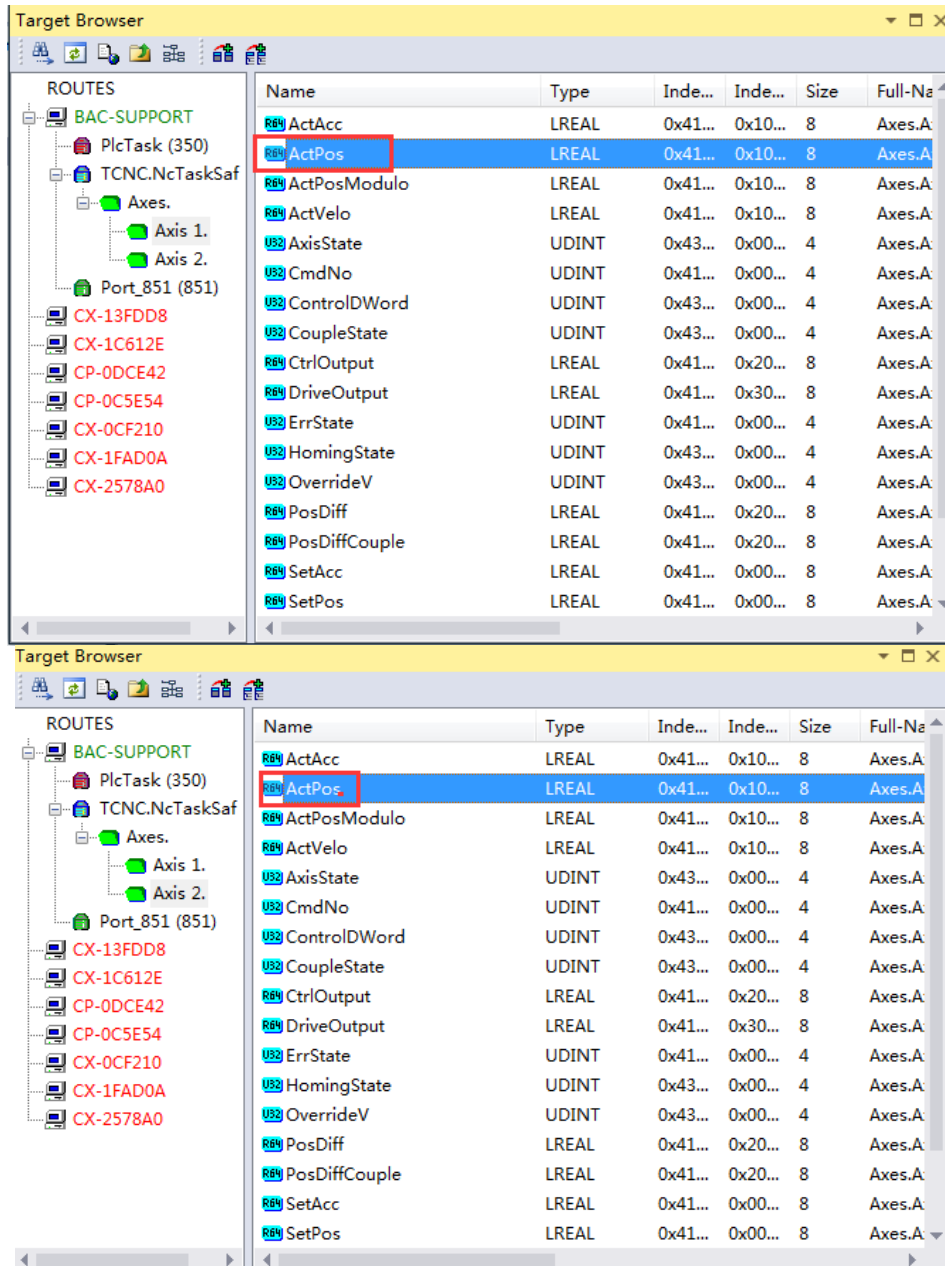
弹出的对话框找到 TwinCAT Measurement——YT——Scope YT Project，创建一个示波器，本实例命名为：tables\_pos。



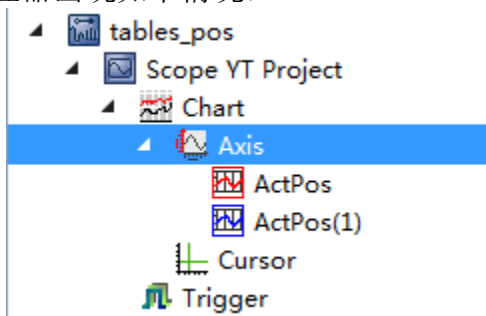
创建完成后，找到示波器下的 Axis 右击，单击 Target Browser。



从弹出的对话框中，找到所连接的控制器主机名（本实例是利用虚轴来做的，所以选择本地电脑），再找到 TCNC.NcTaskSaf(501)—Axis1，从右侧 NC 反馈回来的轴状态字中找到 Actpos 并双击，同理添加 Axis2 轴的 Actpos，然后关闭对话框。

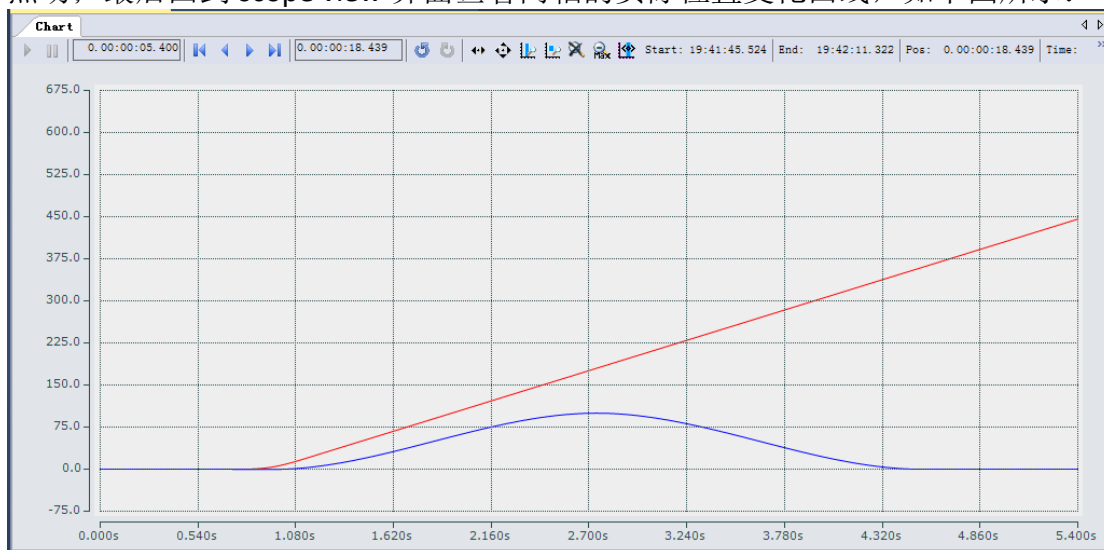


此时可以看见左侧对象管理器出现如下情况：

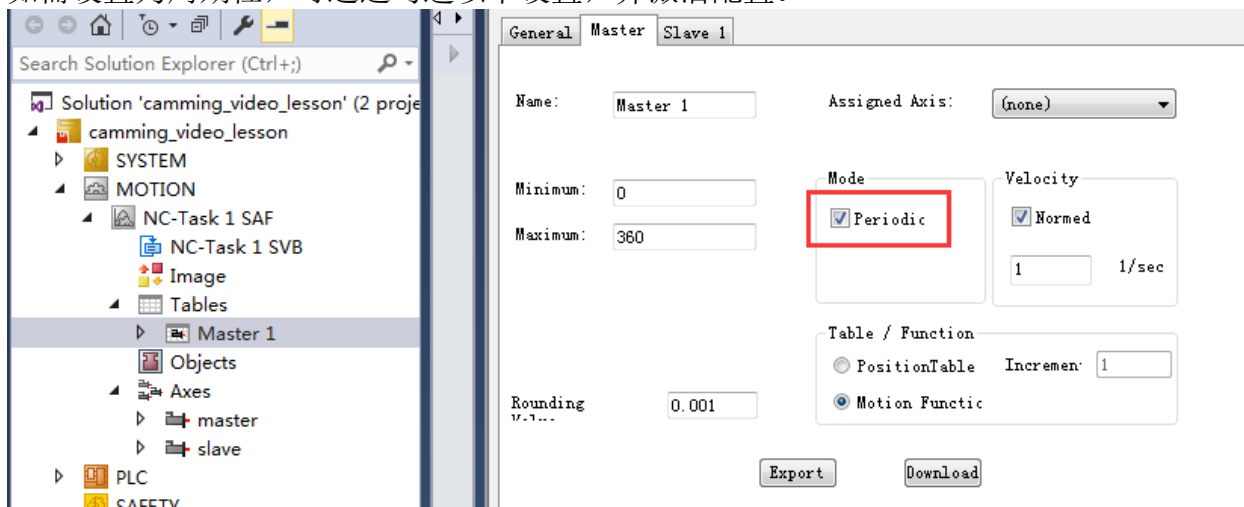


将程序 Login 之后，首先按下 Record 按钮，使用 Scope View 记录两轴的实际位置；然

后在程序区对 power\_do 变量置 TRUE，使得两轴使能；接下来将 camin\_do 置 TRUE 再置 FALSE,对两轴进行耦合；耦合后将 jog\_for 置为 true，触发 axis1 轴以正向 100 的速度进行点动；最后回到 scope view 界面查看两轴的实际位置变化曲线，如下图所示：



如需设置为周期性，可通过勾选以下设置，并激活配置。



## 第二种方法，通过外部设计工具：

通常使用该种方法是由于主从轴之间的位置变化曲线，由第三方优化软件生成出来的，如 MathCAD 软件等，生成出来的结果由文件格式.csv 的

### (1) 导入办法：

新建一张空白 slave，然后点击下图中的 import，通过对应路径他添加关键点表格

General Master Slave 1 Slave 2 Slave 3

Name: Slave 3 Table Id: 3

Assigned Axis: (none) Color

	Position	Velocity	Accelerati	Jerk
Maximum	100	2	0.2	0.2
Minimum	-10	-2	-0.2	-0.2

Rounding 0.001

Import Export Download Upload

另外，也可以将采用电子凸轮表编辑器绘制的凸轮表，导出成.csv 格式，作为备份保存。

General Master Slave 1 Slave 2 Slave 3

Name: Slave 3 Table Id: 3

Assigned Axis: (none) Color

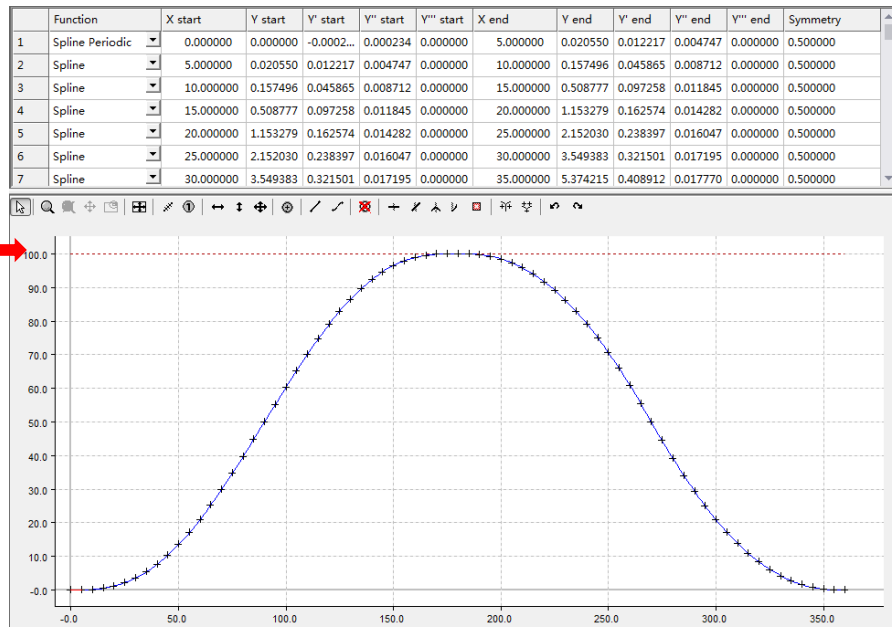
	Position	Velocity	Accelerati	Jerk
Maximum	100	2	0.2	0.2
Minimum	-10	-2	-0.2	-0.2

Rounding 0.001

Import Export Download Upload

(2) 下图分别是数据来源表格以及生成出来的曲线（可自定义一张凸轮表格，其中包含若干关键点）

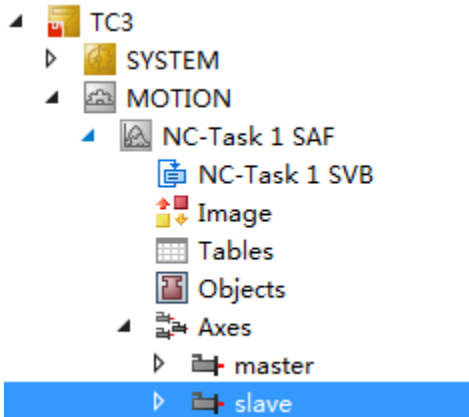
	A	B	C
1	0	0	
2	5	0.02055	
3	10	0.157496	
4	15	0.508777	
5	20	1.153279	
6	25	2.15203	
7	30	3.549383	
8	35	5.374215	
9	40	7.641112	
10	45	10.35156	
11	50	13.49515	
12	55	17.05073	
13	60	20.98765	
14	65	25.26692	
15	70	29.84238	
16	75	34.66194	
17	80	39.66875	
18	85	44.80237	
19	90	50	
20	95	55.19763	
21	100	60.33125	
22	105	65.33806	
23	110	70.15762	
24	115	74.73308	
25	120	79.01235	
26	125	82.94927	
27	130	86.50485	
28	135	89.64844	



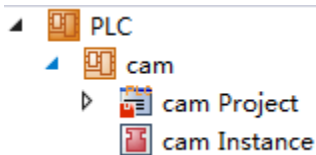
### 第三种，通过 PLC 程序来设计电子凸轮表

设计一个具有五个关键点的电子凸轮，具有多周期性，

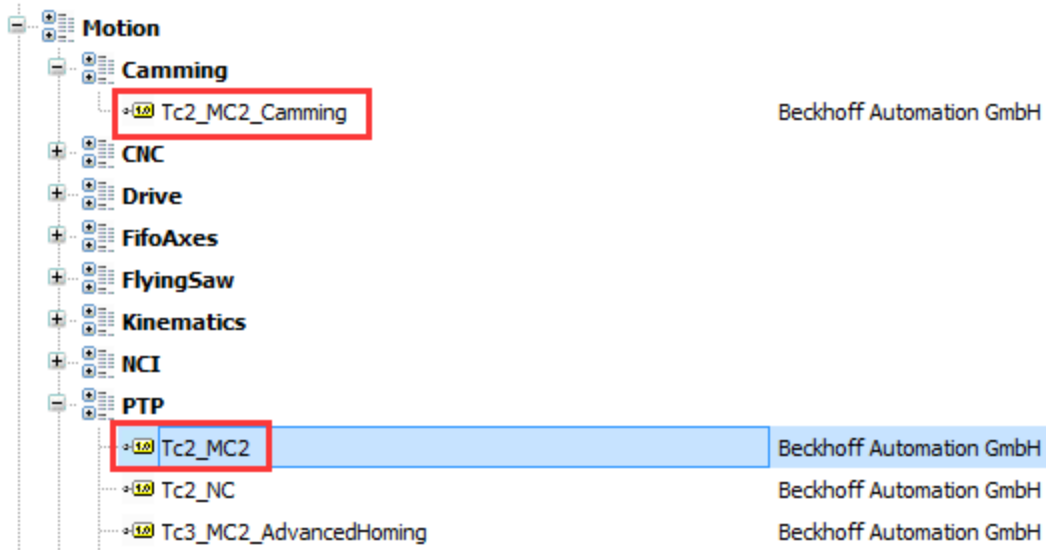
1. 建立两根虚轴，分别更名为 master 和 slave;



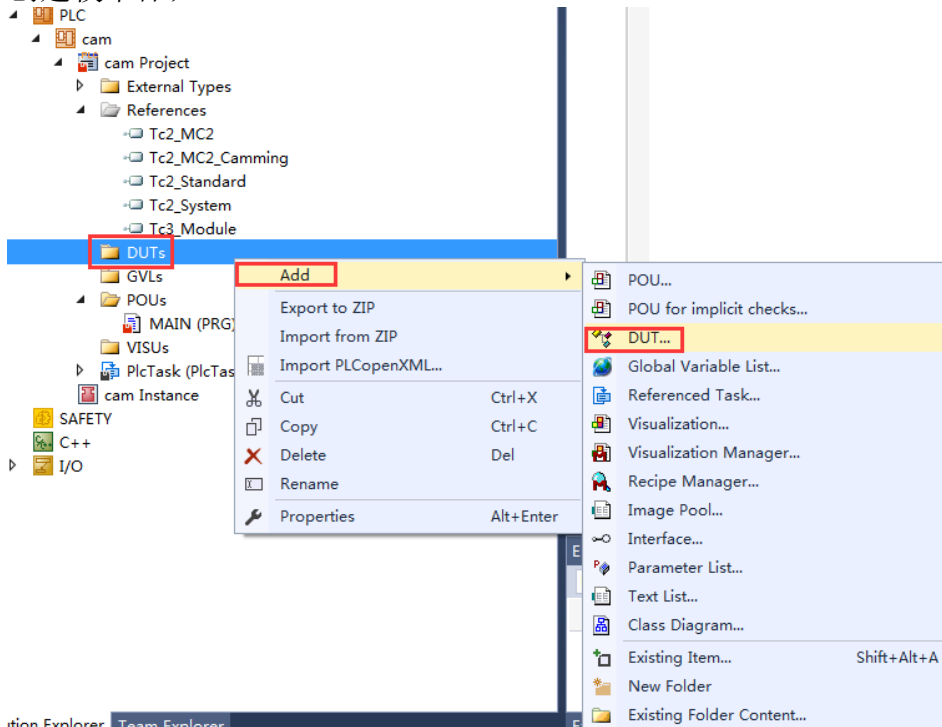
2. 建立 PLC 程序 cam



3. 添加以下库文件;



4. 创建枚举体;



命名为: camming\_state,

```

camming_state*  X
1
2  TYPE camming_state :
3  (
4      state_init,
5      state_power,
6      state_forward,
7      state_pre_table1,
8      state_camming1,
9      state_pre_table2,
10     state_table1,
11     state_error,
12     state_reset
13 ) INT;
14 END TYPE

```

5. MAIN 主程序区创建轴变量及定义变量调用枚举体

```

camming_state*  MAIN*  X  TC3
1  PROGRAM MAIN
2  VAR
3      master,slave:          axis_ref;
4      camming_state:        camming_state;
5  END_VAR

```

同时，程序区刷新轴状态

```

1  (*用于更新轴状态*)
2  master();
3  slave();

```

6. 创建一个 Action 命名为 power，用于对两轴进行使能，同时设置速比；在 power 中，编程如下：

```

MAIN.power  X  camming_state*  MAIN*
1  master.PlcToNc.ControlDWord:=7;
2  slave.PlcToNc.ControlDWord:=7;
3
4  master.PlcToNc.Override:=1000000;
5  slave.PlcToNc.Override:=1000000;

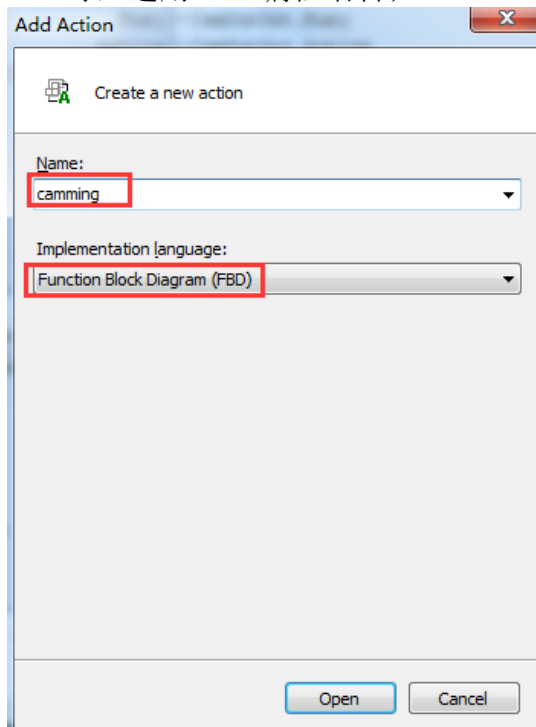
```

7. 定义电子凸轮所需要的数据结构和功能块；

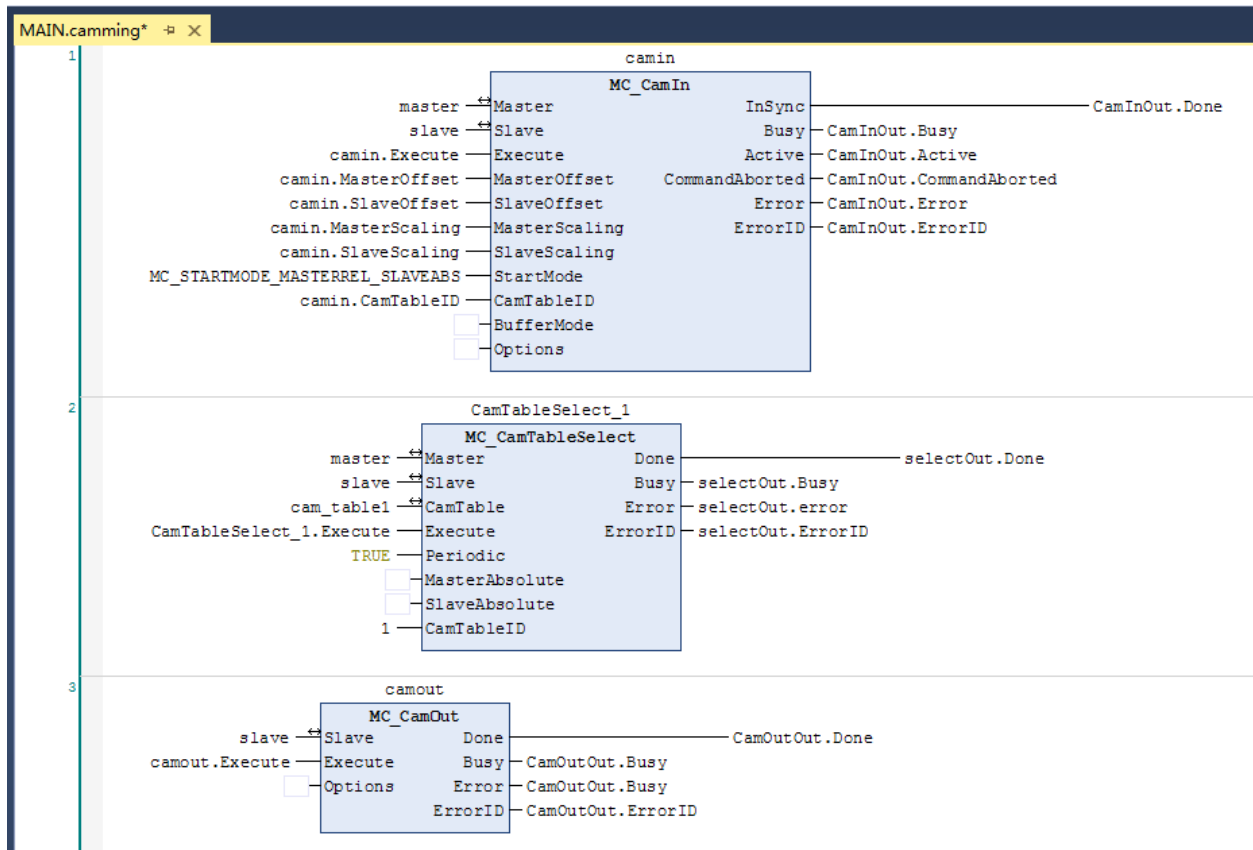


```
MAIN*  TC3  MAIN.pow
1  PROGRAM MAIN
2  VAR
3  master,slave:          axis_ref;
4  camming_state:        camming_state;
5
6  camin:                 mc_camin; (*用于电子凸轮耦合功能块*)
7  camout:                mc_camout; (*用于电子凸轮表解耦功能块*)
8  CamTableSelect_1:     MC_CamTableSelect; (*用于创建凸轮表*)
9  cam_table1:           MC_CAM_REF; (*用于定义凸轮表*)
10 table1:               ARRAY[1..5] OF MC_MotionFunctionPoint; (*用于定义电子凸轮表中的关键点, 此例子中包含五个关键点*)
11 CamInOut,CamOutOut,selectOut: ST_McOutputs; (*用于绑定相关功能块输出接口*)
..
```

8. 其次再创建一个 Action，并命名为 camming（方法同上一个 Action），在创建该 Action 时，选用 FBD 编程语言；



接下来，在 camming 中，调用以下功能块，并且绑定接口；



9. 创建一个 Action，并命名为 table1\_point（方法同上一个 Action），该 Action 用于定义凸轮表的类型及关键点；

```
MAIN.table1_point  X MAIN.camming*  MAIN*  TC3
1  cam_table1.ArraySize:=SIZEOF(table1);
2  cam_table1.NoOfColumns:=1;
3  cam_table1.NoOfRows:=5;
4  cam_table1.pArray:=ADR(table1);
5  cam_table1.TableType:=MC_TABLETYPE_MOTIONFUNCTION;
6
7
8  table1[1].FunctionType:=MOTIONFUNCTYPE_POLYNOM5;
9  table1[1].MasterPos:=0;
10 table1[1].SlavePos:=0;
11 table1[1].PointIndex:=1;
12 table1[1].PointType:=MOTIONPOINTTYPE_MOTION;
13
14 table1[2].FunctionType:=MOTIONFUNCTYPE_POLYNOM5;
15 table1[2].MasterPos:=90;
16 table1[2].SlavePos:=100;
17 table1[2].PointIndex:=2;
18 table1[2].PointType:=MOTIONPOINTTYPE_MOTION;
19
20 table1[3].FunctionType:=MOTIONFUNCTYPE_POLYNOM5;
21 table1[3].MasterPos:=180;
22 table1[3].SlavePos:=80;
23 table1[3].PointIndex:=3;
24 table1[3].PointType:=MOTIONPOINTTYPE_MOTION;
25
26 table1[4].FunctionType:=MOTIONFUNCTYPE_POLYNOM5;
27 table1[4].MasterPos:=270;
28 table1[4].SlavePos:=50;
29 table1[4].PointIndex:=4;
30 table1[4].PointType:=MOTIONPOINTTYPE_MOTION;
31
32 table1[5].FunctionType:=MOTIONFUNCTYPE_POLYNOM5;
33 table1[5].MasterPos:=360;
34 table1[5].SlavePos:=0;
35 table1[5].PointIndex:=5;
36 table1[5].PointType:=MOTIONPOINTTYPE_MOTION;
```

10. 在 MAIN 主程序区的申明区定义如下:

```

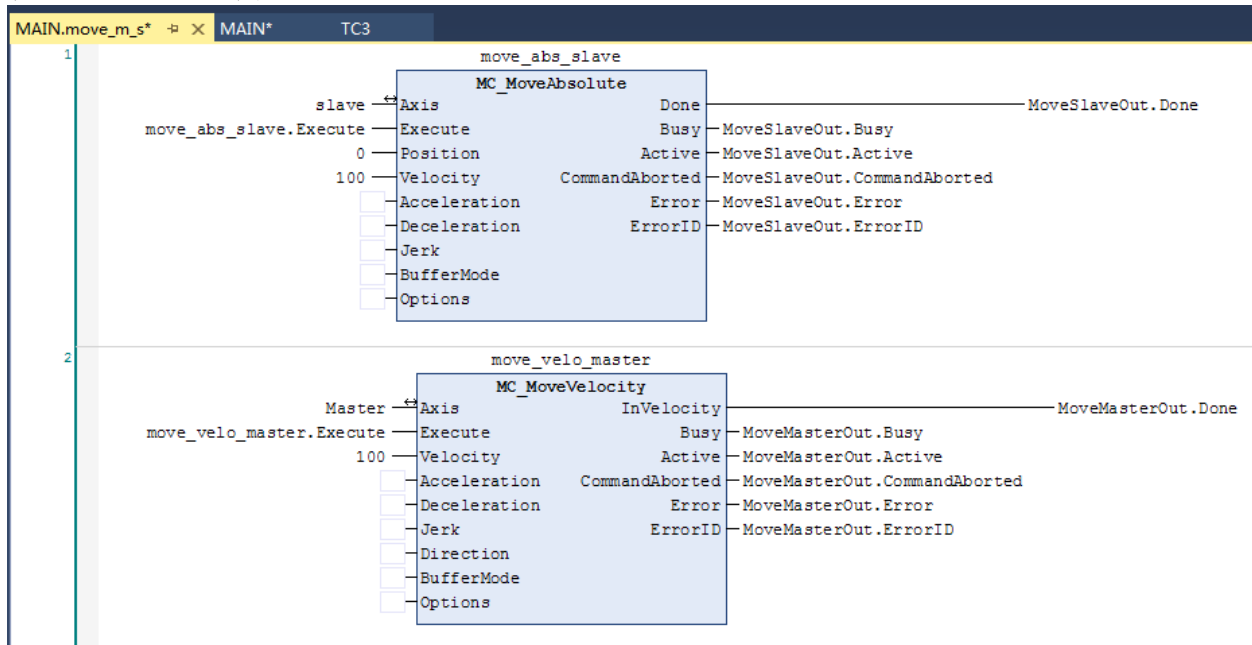
mming*   MAIN*  TC3
PROGRAM MAIN
VAR
  master,slave:          axis_ref;
  camming_state:        camming_state;

  camin:                 mc_camin; (*用于电子凸轮耦合功能块*)
  camout:                mc_camout; (*用于电子凸轮表解耦功能块*)
  CamTableSelect_1:     MC_CamTableSelect; (*用于创建凸轮表*)
  cam_table1:           MC_CAM_REF; (*用于定义凸轮表*)
  table1:               ARRAY[1..5] OF MC_MotionFunctionPoint;
  CamInOut,CamOutOut,selectOut: ST_McOutputs; (*用于绑定相关功能块输出接*)

  move_velo_master:     mc_movevelocity;
  move_abs_slave:       mc_moveabsolute;
  MoveMasterOut:        ST_McOutputs;
  MoveSlaveOut:         ST_McOutputs;

```

创建一个 Action，并命名为 move\_m\_s（方法同上），该 Action 用于定义从轴的回零，及主轴的匀速运动，编程语言用 FBD；



11. 在 MAIN 主程序区的申明区定义如下：

```

camming*   MAIN*  TC3
PROGRAM MAIN
VAR
  master,slave:          axis_ref;
  camming_state:        camming_state;

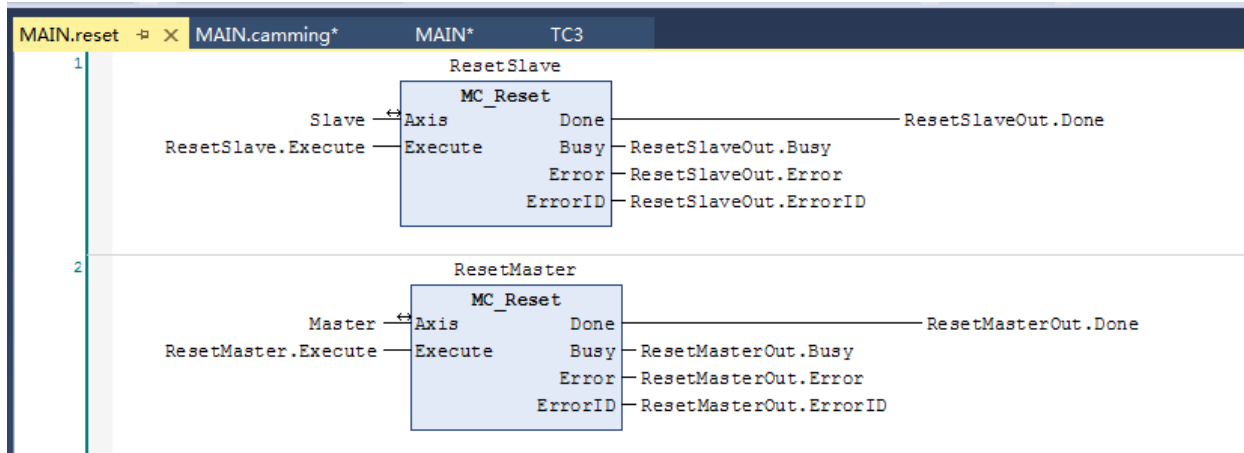
  camin:                 mc_camin; (*用于电子凸轮耦合功能块*)
  camout:                mc_camout; (*用于电子凸轮表解耦功能块*)
  CamTableSelect_1:     MC_CamTableSelect; (*用于创建凸轮表*)
  cam_table1:           MC_CAM_REF; (*用于定义凸轮表*)
  table1:               ARRAY[1..5] OF MC_MotionFunctionPoint; (*用于定
  CamInOut,CamOutOut,selectOut: ST_McOutputs; (*用于绑定相关功能块输出接口*)

  move_velo_master:     mc_movevelocity;
  move_abs_slave:       mc_moveabsolute;
  MoveMasterOut:        ST_McOutputs;
  MoveSlaveOut:         ST_McOutputs;

  ResetSlave,resetMaster: mc_reset;
  ResetMasterOut,ResetSlaveOut: ST_McOutputs;
END_VAR

```

创建一个 Action，并命名为 reset（方法同上），该 Action 用于如果轴一旦出现报错，对错误代码进行复位，编程语言用 FBD；



12. MAIN 主程序区调用定义好的 Action，程序区调用枚举 camming\_state，用 CASE 语句作为主体框架：

```

1  (*用于更新轴状态*)
2  master();
3  slave();
4
5  power();
6  camming();
7  reset();
8  move_m_s();
9  table1_point();
10

```

① 做好初始化;

```

CASE camming_state OF
state_init: (*程序初始化*)
    move_velo_master.Execute := FALSE;
    ResetMaster.Execute := FALSE;
    ResetSlave.Execute := FALSE;
    CamIn.Execute := FALSE;
    CamOut.Execute := FALSE;
    camming_state:=state_power;

```

② 对主从轴进行使能

```

state_power: (*主从轴使能*)
    IF master.NcToPlc.StateDWord.0 AND slave.NcToPlc.StateDWord.0 THEN
        camming_state := state_forward;
    ELSIF master.Status.Error OR slave.Status.Error THEN
        camming_state := state_error;
    END_IF

```

③ 从轴回到位置为 0 处，主轴以 100 的速度匀速正转

```

state_forward: (*主轴100正转，从轴回0*)
    move_velo_master.Execute:=TRUE;
    move_abs_slave.Execute:=TRUE;
    IF master.Status.Moving AND move_abs_slave.Done THEN
        move_velo_master.Execute:=FALSE;
        move_abs_slave.Execute:=FALSE;
        camming_state:=state_pre_table1;
    END_IF
    IF master.Status.Error OR slave.Status.Error THEN
        camming_state := state_error;
    END IF

```

④ 对 table1 表进行初始化，并导入关键点

```

state_pre_table1: (*table1初始化*)
  camtableselect_1.Execute:=TRUE;
  camin.CamTableID:=1;
  IF camtableselect_1.Done THEN
    camming_state:=state_camming1;
    camtableselect_1.Execute:=FALSE;
  END_IF
  IF master.Status.Error OR slave.Status.Error THEN
    camming_state := state_error;
  END_IF

```

⑤ 主从轴耦合，实现电子凸轮表；

```

state_camming1:
  camin.Execute:=TRUE;
  IF camin.InSync THEN
    camin.Execute:=FALSE;
    camming_state:=camming_status;
  END_IF
  IF master.Status.Error OR slave.Status.Error THEN
    camming_state := state_error;
  END_IF

```

⑥ 耦合成功后，通过 camming\_status 来读取主从轴设定位置；

```

60      camming_status:
61          master_actpos:=master.NcToPlc.SetPos;
62          slave_actpos:=slave.NcToPlc.SetPos;
63          IF master.Status.Error OR slave.Status.Error THEN
64              camming_state := state_error;
65          END_IF

```

⑦ 另外，若出现轴报错，需要执行电子凸轮表解耦和复位；通过 state\_error 和 state\_reset

```

67      state_error:
68          CamOut.Execute := Slave.Status.Coupled; (* 先解耦 *)
69          IF not Slave.Status.Coupled THEN
70              IF Master.Status.Error OR Slave.Status.Error THEN
71                  camming_state := state_reset; (* 轴报错需要reset*)
72              ELSE
73                  camming_state := state_init; (*若功能块报错，不需要reset*)
74              END_IF
75          END_IF
76
77      state_reset:
78          ResetMaster.Execute := TRUE;
79          CamTableSelect_1.Execute:=FALSE;
80          ResetSlave.Execute := TRUE;
81          IF ResetMaster.Done AND ResetSlave.Done THEN
82              camming_state := state_init;
83          ELSIF ResetMaster.Error OR ResetSlave.Error THEN
84              camming_state := state_init;
85          END IF

```

13. 创建 Scope 监测从轴的位置变化曲线，可看到如下显示；



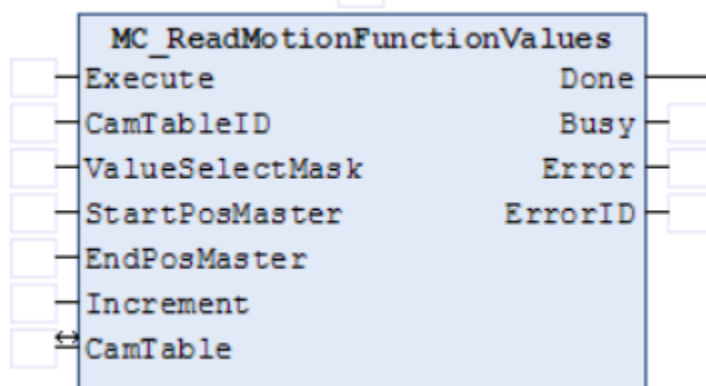
14. 该例子程序见 Motion 部分的“TC3-Camming”

### (三) 电子凸轮的转换

通常电子凸轮表一般有关键点、描点法两种实现方法。关键点方法一般只需要定义好几个关键点的坐标，该点的速度、加速度等属性，关键点之间的线型等即可，NC 即可帮忙规划好整个位置关系路径，但有时需要导出位置表数据做进一步的分析处理，这时就可以使用 MC\_ReadMotionFunctionValues 来实现程序动态导出成所需的数组。

本例程 TC3\_camming\_ReadMotionFunctionValues 以 TC3-Camming 为基础，实现此功能。

1. 需要使用到 FB: MC\_ReadMotionFunctionValues，关于此功能块的介绍可参考链接：  
[https://infosys.beckhoff.com/content/1033/tf5050\\_tc3\\_nc\\_camming/1003149323.html?id=2657705569318836145](https://infosys.beckhoff.com/content/1033/tf5050_tc3_nc_camming/1003149323.html?id=2657705569318836145)



VAR\_Input 部分:

Execute:作为该功能的触发位，上升沿触发

CamTableID:需要读取的凸轮表 Id



ValueSelectMask:若导出的数组只需要位置关系, 选择 MC\_VALUETYPE\_POSITION

```
TYPE MC_ValueSelectType :  
(  
  (* a bitmask can be created by adding the following values *)  
  MC_VALUETYPE_POSITION      := 1,  
  MC_VALUETYPE_VELOCITY      := 2,  
  MC_VALUETYPE_ACCELERATION  := 4,  
  MC_VALUETYPE_JERK          := 8  
) ;  
END TYPE
```

StartPosMaster: 凸轮表内部 master 的采集位置起点

EndPosMaster: 凸轮表内部 master 的采集位置重点

Increment: master 轴位置的采集间隔。

注意得到的数据点个数:  $\frac{\text{EndPosMaster} - \text{StartPosMaster}}{\text{Increment}} + 1$ , 需为整数

CamTable: 这里需定义成描点法的凸轮表类型, 用来缓存需记录的凸轮曲线

## 2. 程序实现方法:

(1) 主程序区定义功能块, 如下:

```
// ADD ReadMotionFunctionValues //  
ReadMotionFunctionValues:      MC_ReadMotionFunctionValues;
```

(2) 添加一个 Action, 命名为 ACT\_ReadMotionFunctionValues, 并在 MAIN 区调用

MAIN (PRG)

ACT\_ReadMotionFunctionValues

```
Act_ReadMotionFunctionValues ();
```

(3) 调用 ReadMotionFunctionValues 功能块

```
ReadMotionFunctionValues (  
  ① Execute:=bReadMotionFunctionValues ,  
  ② CamTableID:=1 ,  
  ③ ValueSelectMask:= MC_VALUETYPE_POSITION ,  
  ④ StartPosMaster:= StartPosMaster ,  
  ⑤ EndPosMaster:= EndPosMaster ,  
  ⑥ Increment:= Increment ,  
  Done=> ,  
  Busy=> ,  
  Error=> ,  
  ErrorID=> ,  
  ⑦ CamTable:=readCamTable );
```

① 触发条件 bReadMotionFunctionValues;

② 读取的是 ID 为 1 的凸轮表;

③ 只采集主从轴之间的位置对应关系;

- ④ 和导出从 Master 内部位置为 0 (StartPosMaster 赋初值 0) 为起点,
- ⑤ 至 Master 内部位置为 100 (EndPosMaster 赋初值 100) 为终点的位置数据;

```
EndPosMaster: LREAL := 100;
StartPosMaster: LREAL := 0;
```

- ⑥ 间隔量为 1;

```
Increment: LREAL := 1;
```

此处可以通过公式:  $\frac{\text{EndPosMaster} - \text{StartPosMaster}}{\text{Increment}} + 1$  得到位置对应点的个数为 101 个, 所以申明记录位置值得二维数组为:

```
table_test: ARRAY[0..100, 0..1] OF LREAL;
```

Table\_test[X,0]为 master 位置值, Table\_test[X,1]为 slave 位置值

- ⑦ 将需要导出段的凸轮曲线存在 readCamTable 凸轮表中

```
readCamTable: MC_CAM_REF;
```

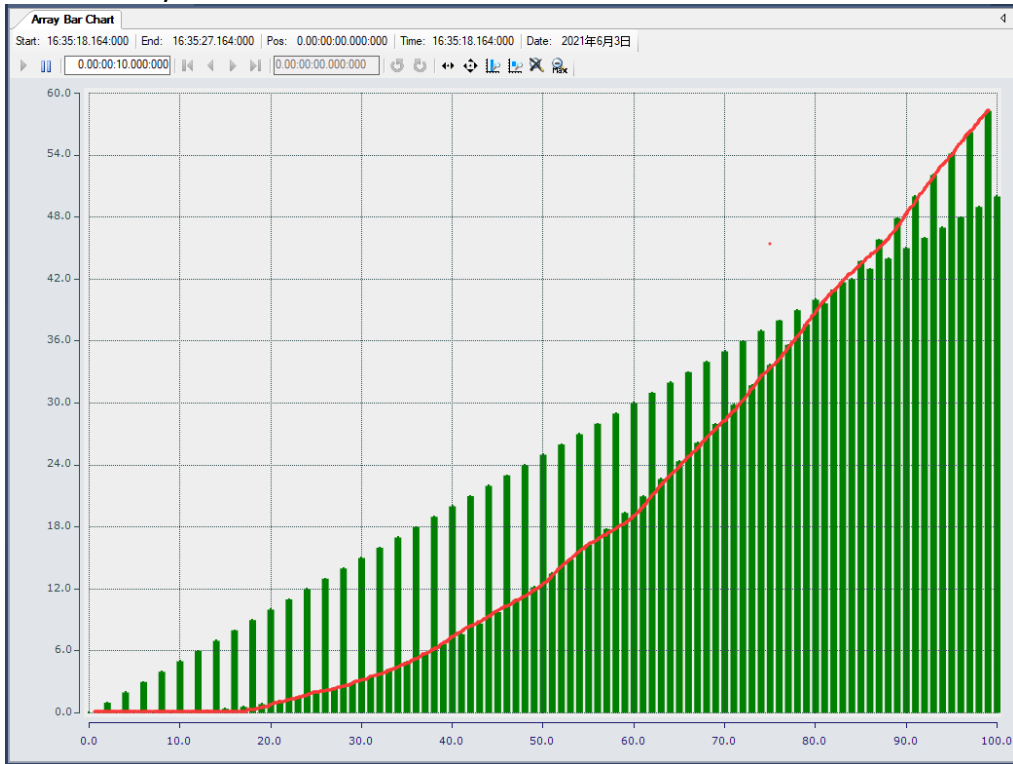
对该凸轮表进行定义, 如下 (关于 MC\_CAM\_REF 参见教材前面的介绍)

```
readCamTable.ArraySize:=SIZEOF(table_test);
readCamTable.NoOfColumns:=2;
temp_M:=LREAL_TO_UDINT((EndPosMaster- StartPosMaster)/ Increment);
readCamTable.NoOfRows:=100;
readCamTable.pArray:=ADR(table_test);
readCamTable.TableType:= MC_TABLETYPE_EQUIDISTANT;
```

- (4) 运行程序, 触发 bReadMotionFunctionValues 可读以下数据

table_test	ARRAY [0..100, 0....	
table_test[0, 0]	LREAL	0
table_test[0, 1]	LREAL	0
table_test[1, 0]	LREAL	1
table_test[1, 1]	LREAL	0.001348981354...
table_test[2, 0]	LREAL	2
table_test[2, 1]	LREAL	0.010611390540...
table_test[3, 0]	LREAL	3
table_test[3, 1]	LREAL	0.035209876543...
table_test[4, 0]	LREAL	4
table_test[4, 1]	LREAL	0.082042778031...
table_test[5, 0]	LREAL	5
table_test[5, 1]	LREAL	0.157496316618...
table_test[6, 0]	LREAL	6
table_test[6, 1]	LREAL	0.267456790123...
table_test[7, 0]	LREAL	7
table_test[7, 1]	LREAL	0.417322765838...
table_test[8, 0]	LREAL	8
table_test[8, 1]	LREAL	0.612017273789...
table_test[9, 0]	LREAL	9
table_test[9, 1]	LREAL	0.856000000000...
table_test[10, 0]	LREAL	10
table_test[10, 1]	LREAL	1.153279479754...
table_test[11, 0]	LREAL	11
table_test[11, 1]	LREAL	1.507425290860...
table_test[12, 0]	LREAL	12
table_test[12, 1]	LREAL	1.921580246913...

若使用 arraybar 采集，红色标记的为从轴坐标值

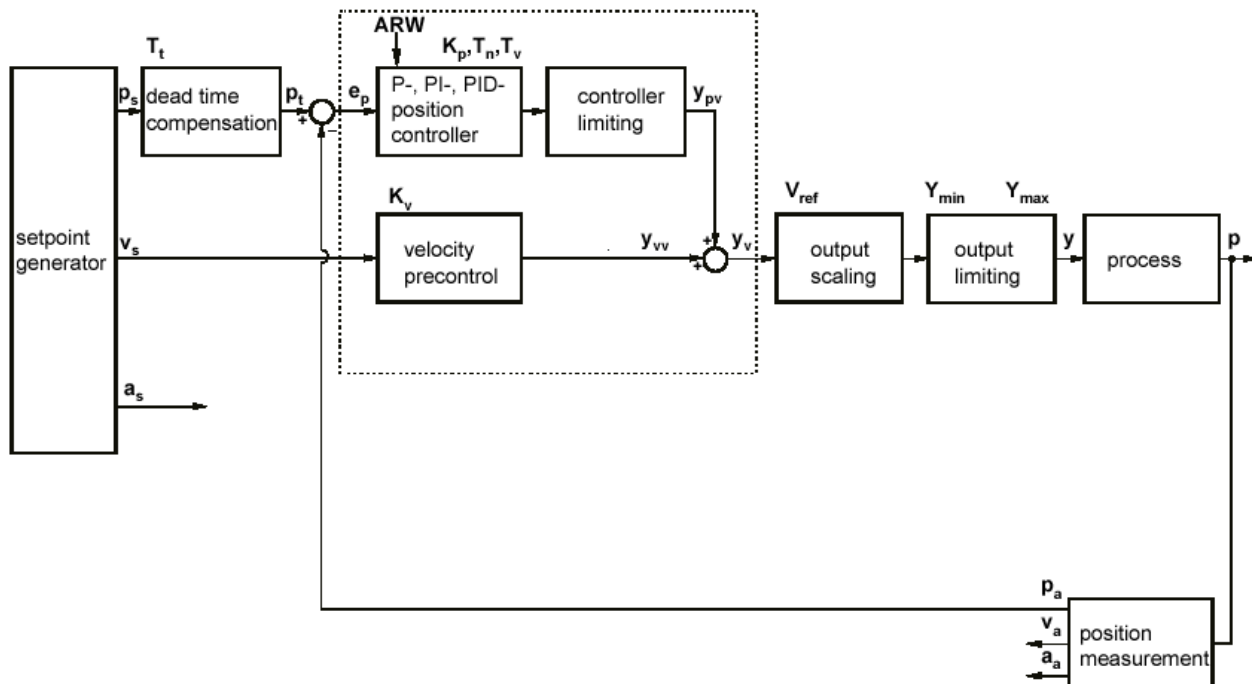


(5) 该例程参考“TC3\_camming\_ReadMotionFunctionValues.tszip”

## 五、位置外部设定值发生器

通常 TwinCAT NC 的设定位置 (SetPosition)、设定速度 (SetVelocity)、设定加速度 (SetAcceleration) 是由 NC 信号发生器 (即下图的 Setpoint Generator) 产生的。每个 NC 周期 (比如 2ms) 产生一套设定数据 Setpoint。如果驱动器工作在位置模式, Setpoint 中的位置信号, 就会换算后发给驱动器, 如果驱动器工作在速度模式, Setpoint 中的速度信号, 就会换算后发给驱动器。

如图所示:

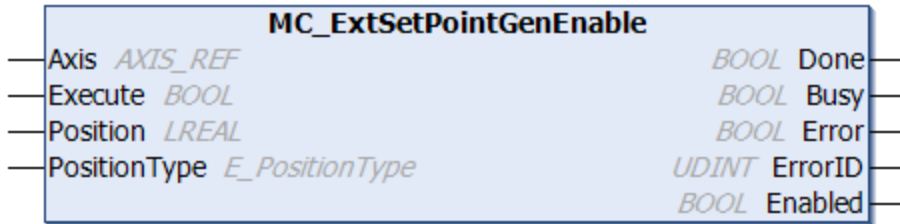


但在一些特殊情况下, 运动关系较为复杂, 那么用户需要自己的算法来给定每个 NC 周期的目标位置和目标速度。此时, 可以在 PLC 程序中使用一个独立的设定值发生器 (Setpoint Generator), 取代 NC 位置发生器的功能。这样提高 TwinCAT 轴的灵活性, 可以应用于更广泛的场合。比如: 电机转动与实际工件运动为非线性关系时, 或者需要多种运动迭加的时候。

使用外部设定值发生器, 需要三个步骤: 启用——位置给定——停用, 依次由功能块 MC\_ExtSetPointGenEnable、MC\_ExtSetPointGenFeed、MC\_ExtSetPointGenDisable (基于 MC 基本库 TC2\_MC2.lib) 实现。

### (一) 功能块及功能介绍 (基于 Tc2\_Mc2.lib)

1. 该功能块用于对外部位置发生器进行使能



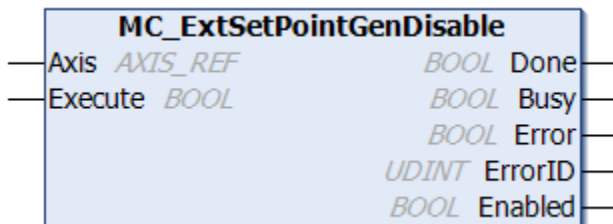
其中：PositionType：给定位置类型，有两种类型可选：

POS\_ABSOLUTE：绝对位置

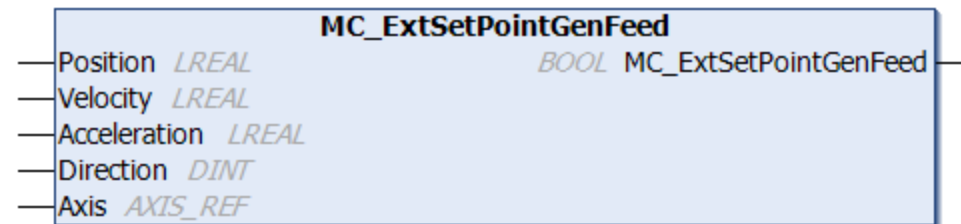
POS\_RELATIVE：相对位置

注意：输入 Position 并不是指让 NC 轴运动到该位置，而是到达该位置后，NC 轴标记位 InTargetPosition 置位

2. 该功能块用于关断外部位置发生器



3. 该功能用于给定位置发生器的目标位置，且只在外部位置发生器触发之后，才能将输入标量复制到 PLCTONC\_AXIS\_REF 结构体中



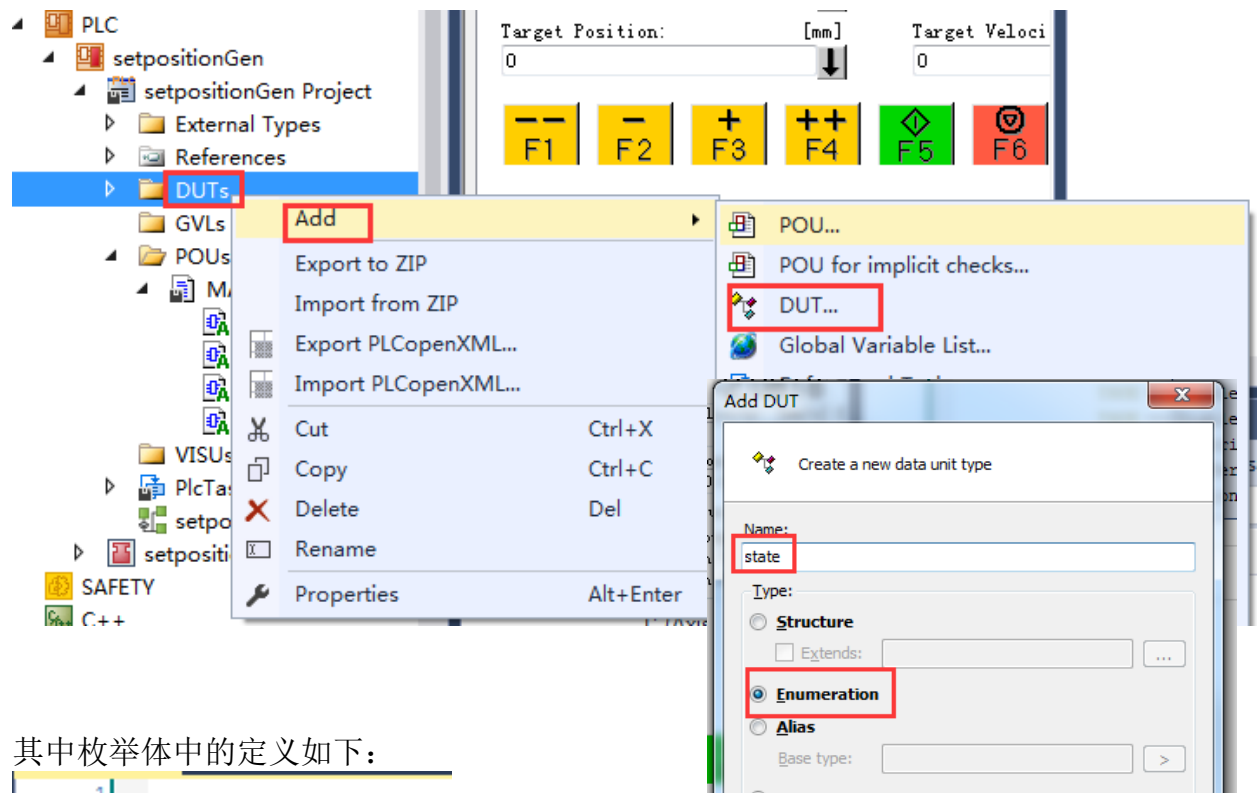
## (二) 外部位置给定发生器例子程序说明

首先该例子程序中，创建了一个枚举体和四个 Action，主程序去以一个 CASE 作为整个程序的框架；其中本例子中，外部位置给定发生器对轴按照以下函数关系发送位置值，关系式如下：

$$y(x) = \sin(2x) + 2 \cos(4x) + 3 \cos(2x) + 4 \cos(4x)$$

$$x = 0.01 \times N_{plc\ task} \text{ (其中 } N_{plc\ task} \text{ 为每个任务周期自加 1)}$$

1. 定义枚举体



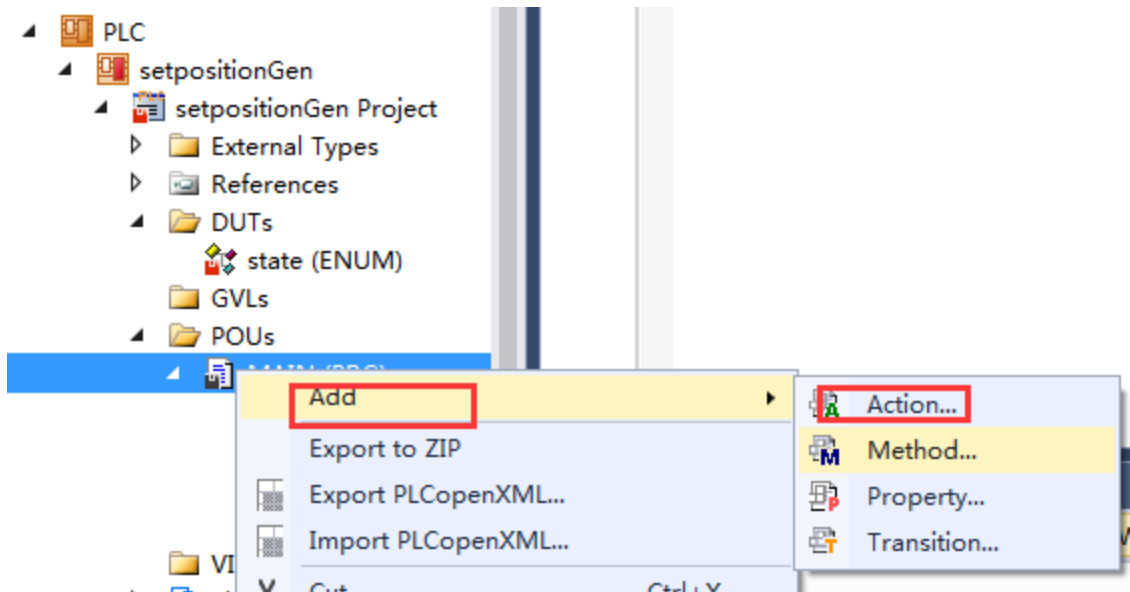
其中枚举体中的定义如下：

```

1
2  TYPE state :
3  (
4      init := 0,
5      gen_power,
6      gen_move,
7      gen_enable,
8      gen_error,
9      gen_reset
10 ) INT;
11 END_TYPE
12

```

2. 先定义第一个 Action 用于对轴进行使能，命名为 power，使用 FBD 语言；



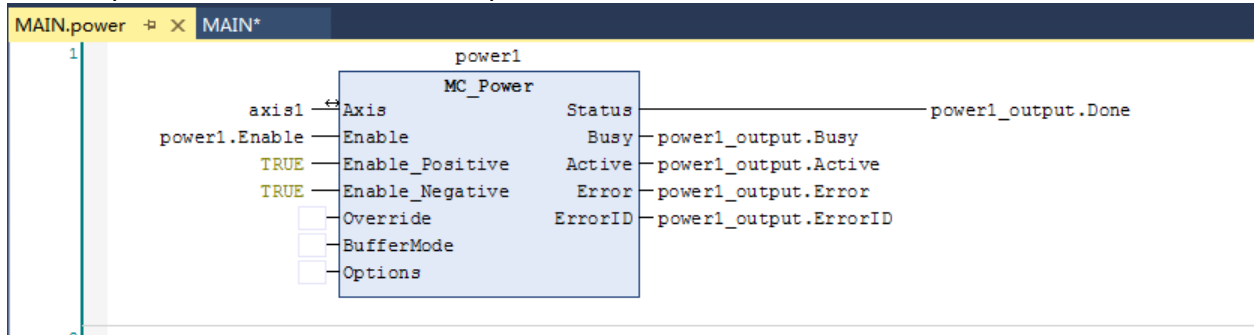
接下来，在 MAIN 主程序变量申明区，定义如下变量和功能块：

```

MAIN.power  MAIN*
1  PROGRAM MAIN
2  VAR
3      power1: MC_Power;
4      axis1: AXIS_REF;
5      power1_output:ST_McOutputs;

```

其次在 power 这个 ACTION 中添加 power1 功能块，并对接口做如下定义：



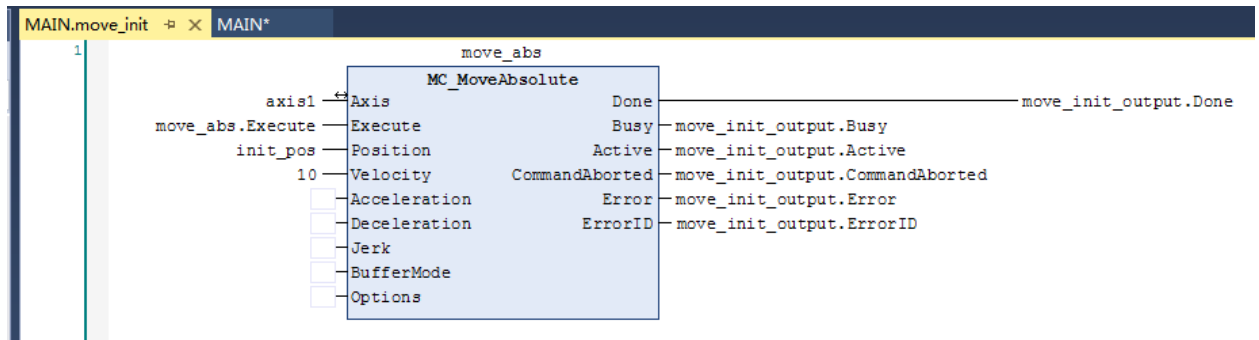
3. 定义第二个 Action，命名为 move\_init，该 Action 用于将轴先走到该段函数关系式的起始位；首先定义相关功能块和接口，在 MAIN 主程序申明区做如下申明：

```

MAIN.move_init  MAIN*
2  VAR
3      power1: MC_Power;
4      axis1: AXIS_REF;
5      power1_output:ST_McOutputs;
6
7      move_abs: MC_MoveAbsolute;
8      init_pos: LREAL;
9      move_init_output:ST_McOutputs;
10

```

Action 中的定义如下：



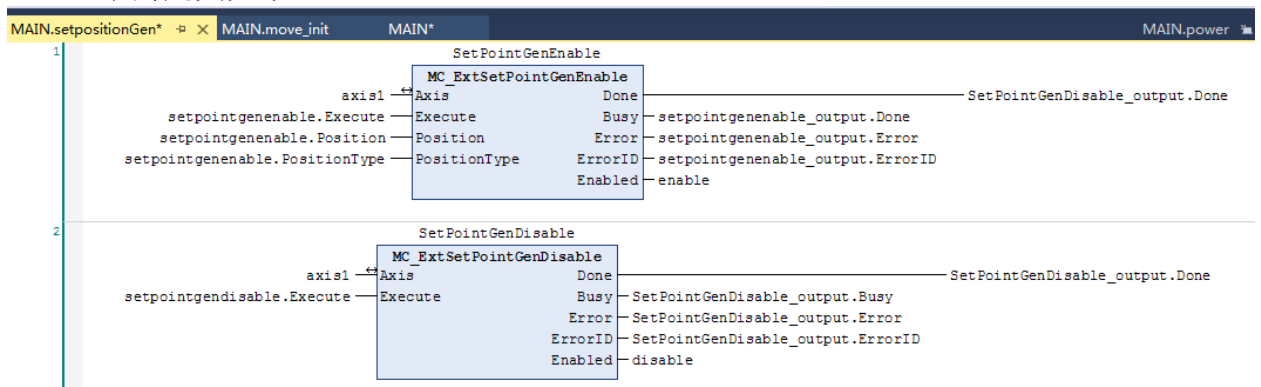
- 定义第三个 Action，命名为 setpositionGen，该 Action 用于定义开启/关闭外部位置发生器; 首先定义相关功能块和接口，在 MAIN 主程序申明区做如下申明：

```

MAIN.setpositionGen*   MAIN.move_init   MAIN*
1  PROGRAM MAIN
2  VAR
3      power1: MC_Power;
4      axis1: AXIS_REF;
5      power1_output:ST_McOutputs;
6
7      move_abs: MC_MoveAbsolute;
8      init_pos: LREAL;
9      move_init_output:ST_McOutputs;
10
11     SetPointGenEnable: MC_ExtSetPointGenEnable;
12     SetPointGenDisable: MC_ExtSetPointGenDisable;
13     SetPointGenEnable_output,SetPointGenDisable_output:ST_McOutputs;
14     enable,disable:BOOL;
15

```

Action 中的定义如下：



- 定义第四个 Action，命名为 reset，该 Action 用于如果程序中出现轴报错用于关闭外部发生器并且复位的; 首先定义相关功能块和接口，在 MAIN 主程序申明区做如下申明：

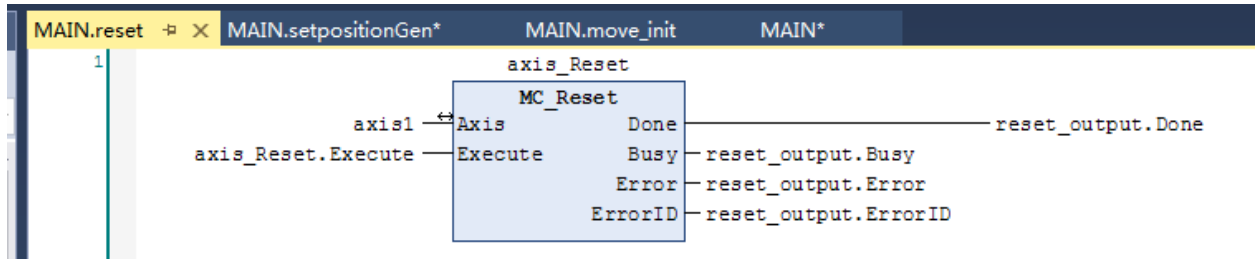


```

MAIN.setpositionGen*   MAIN.move_init   MAIN*
1  PROGRAM MAIN
2  VAR
3      power1: MC_Power;
4      axis1:  AXIS_REF;
5      power1_output:ST_McOutputs;
6
7      move_abs: MC_MoveAbsolute;
8      init_pos:  LREAL;
9      move_init_output:ST_McOutputs;
10
11     SetPointGenEnable: MC_ExtSetPointGenEnable;
12     SetPointGenDisable: MC_ExtSetPointGenDisable;
13     SetPointGenEnable_output,SetPointGenDisable_output:ST_McOutputs;
14     enable,disable:BOOL;
15
16     reset_output:ST_McOutputs;
17     axis Reset:  MC_Reset;

```

Action 中的定义如下:



6. 回到 MAIN 主程序区, 定义变量 gen\_state, 采用类型为 state, 如下:

```

MAIN*
1  PROGRAM MAIN
2  VAR
3      power1: MC_Power;
4      axis1:  AXIS_REF;
5      power1_output:ST_McOutputs;
6
7      move_abs: MC_MoveAbsolute;
8      init_pos:  LREAL;
9      move_init_output:ST_McOutputs;
10
11     SetPointGenEnable: MC_ExtSetPointGenEnable;
12     SetPointGenDisable: MC_ExtSetPointGenDisable;
13     SetPointGenEnable_output,SetPointGenDisable_output:ST_McOutputs;
14     enable,disable:BOOL;
15
16     reset_output:ST_McOutputs;
17     axis_Reset:  MC_Reset;
18
19     gen_state:state;
20

```

7. 在程序区，调用四个 Action，同时更新轴状态，程序如下：

```
MAIN*  ▸ ×
1  PROGRAM MAIN
2  VAR
3      power1: MC_Power;
4      axis1:  AXIS_REF;
5      power1_output:ST_McOutputs;
6
7      move_abs: MC_MoveAbsolute;
8      init_pos: LREAL;
9
10     axis1(); (*更新轴状态*)
11
12     power();
13     setpositionGen();
14     reset();
15     move_init();
```

8. 在 CASE 语句，第一步也就是 state 为 0，对应为 init 时，对功能块进行初始化，同时计算关系式起始位置，程序如下：

```
1  axis1(); (*更新轴状态*)
2
3  power();
4  setpositionGen();
5  reset();
6  move_init();
7
8  CASE gen_state OF
9      init:
10         power1.Enable:=FALSE;
11         SetPointGenEnable.Execute:=FALSE;
12         SetPointGenDisable.Execute:=FALSE;
13         axis_Reset.Execute:=FALSE;
14         gen_state:=gen_power;
15         init_pos:= SIN(0) + 2*COS(0) + 3*COS(0) + 4*COS(0);
16
```

9. gen\_state 在为 gen\_power 时，对轴进行使能，程序如下：

```

8 CASE gen_state OF
9     init:
10         power1.Enable:=FALSE;
11         SetPointGenEnable.Execute:=FALSE;
12         SetPointGenDisable.Execute:=FALSE;
13         axis_Reset.Execute:=FALSE;
14         gen_state:=gen_power;
15         init_pos:= SIN(0) + 2*COS(0) + 3*COS(0) + 4*COS(0);
16
17     gen_power:
18         power1.Enable:=TRUE;
19         IF power1_output.Done THEN
20             gen_state:=gen_move;
21         END_IF
22         IF axis1.Status.Error THEN
23             gen_state := gen_error;
24         END_IF

```

10. 使能完成之后，gen\_state 为 gen\_move 时，通过 ABS 的方式轴先走到关系式的起点，程序如下：

```

gen_power:
power1.Enable:=TRUE;
IF power1_output.Done THEN
    gen_state:=gen_move;
END_IF
IF axis1.Status.Error THEN
    gen_state := gen_error;
END_IF

```

```

gen_move:
move_abs.Execute:=TRUE;
IF move_abs.Done THEN
    move_abs.Execute:=FALSE;
    gen_state:=gen_enable;
END_IF

```

11. 走到起始位置之后，gen\_state 为 gen\_enable 时，开启外部位置给定发生器，同时让轴按照给定的关系式来运行，因用到 FUN 需做如下申明：

```
MAIN*  ▢ ×
16     reset_output:ST_McOutputs;
17     axis_Reset: MC_Reset;
18
19     gen_state:state;
20
21     position: LREAL;
22     velocity: LREAL;
23     direction: DINT;
24     acceleration: LREAL;
25     t:lreal;
26
```

程序如下:

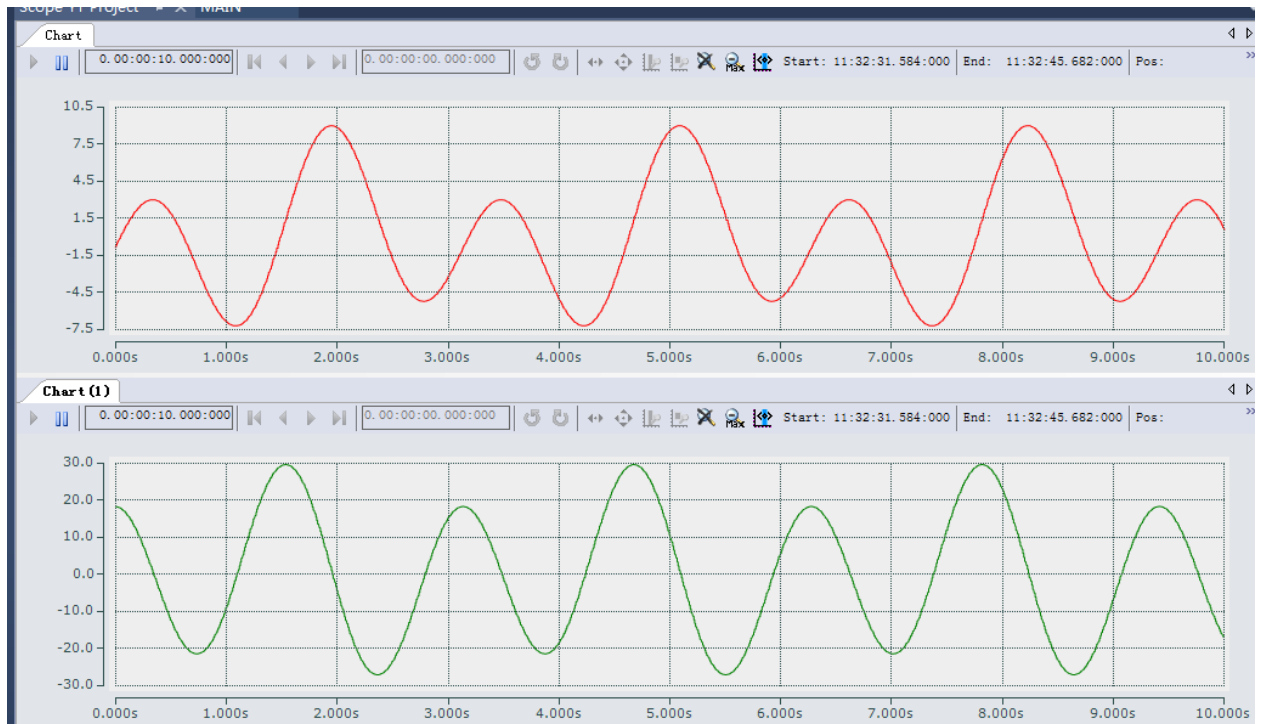
```
33     gen_enable:
34         setpointgenenable.Execute:=TRUE;
35         IF SetPointGenEnable.Enabled THEN
36             t:=t+0.01;
37             MC_ExtSetPointGenFeed(
38                 Position:= SIN(2*t) + 2*COS(4*t) + 3*COS(2*t) + 4*COS(4*t),
39                 Velocity:= 2*COS(2*t)-8*SIN(4*t)-6*SIN(2*t)-16*SIN(4*t),
40                 Acceleration:=-4*SIN(2*t)-32*COS(4*t)-12*COS(2*t)+64*COS(4*t) ,
41                 Direction:= 1,
42                 Axis:= axis1);
43         END_IF
44         IF axis1.Status.Error THEN
45             gen_state := gen_error;
46         END_IF
```

12. 若一旦出现报错, 进行复位操作, 程序如下:

```
43     END_IF
44     IF axis1.Status.Error THEN
45         gen_state := gen_error;
46     END_IF
47
48     gen_error:
49         SetPointGenEnable.Execute:=FALSE;
50         SetPointGenDisable.Execute := SetPointGenEnable.Enabled;
51     IF NOT axis1.Status.Coupled THEN
52         IF axis1.Status.Error THEN
53             gen_state := gen_reset;
54         ELSE
55             gen_state := init;
56         END_IF
57     END_IF
58
59     gen_reset:
60         axis_Reset.Execute := TRUE;
61         SetPointGenDisable.Execute:=FALSE;
62
63         IF axis_Reset.Done THEN
64             gen_state := init;
65         ELSIF axis_Reset.Error THEN
66             gen_state := init;
67         END_IF
68
69 END_CASE
```

13. 创建虚轴，并且连接 PLC 轴变量；

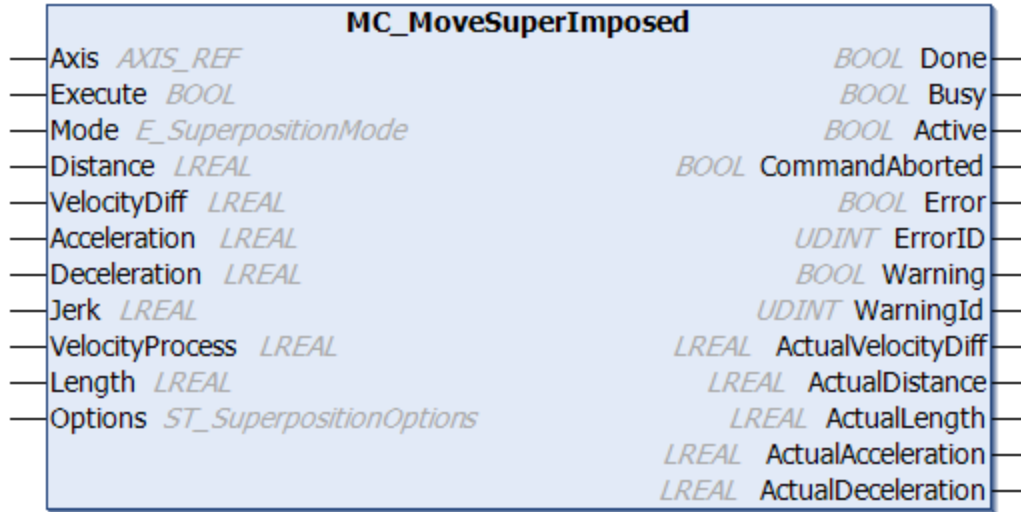
14. 运行程序，采用 Scope 来监测设定位置和设定速度，可看到如下显示：



## 六、位置补偿功能

### 1. 位置补偿功能块

TwinCAT NC PTP 提供一个用于位置补偿的功能块 MC\_MoveSuperimposed, 基于运动控制基本库 Tc2\_MC2.lib。该功能块使运动中的 NC 轴同时执行一个位置叠加的动作。如果 NC 轴是独立运动的轴、多轴联动的 Master, 或者电子凸轮 Cam 的 Slave, 都可以对轴进行补偿。



该功能块由输入变量 Execute 的上升沿触发。完成后输出变量 Done 置位。

VelocityProcess, 指补偿过程匀速阶段的最大限值。

位置补偿的功能块 MC\_MoveSuperImposedExt 的关键参数是补偿距离 Distance、最大速度差 VelocityDiff、补偿区间 Length 以及补偿模式 Mode。

补偿模式 Mode 用于选择生效的补偿速度差和补偿区间。一共有 4 种模式:

**SUPERPOSITIONMODE\_VELOREDUCTION\_ADDITIVEMOTION :**

规定的区间 Length+Distance 内完成 Distance 的补偿, 限定速度变化不超过 VelocityDiff。

**SUPERPOSITIONMODE\_VELOREDUCTION\_LIMITEDMOTION:**

规定的区间 Length 内完成 Distance 的补偿, 限定速度变化不超过 VelocityDiff。

**SUPERPOSITIONMODE\_LENGTHREDUCTION\_ADDITIVEMOTION,**

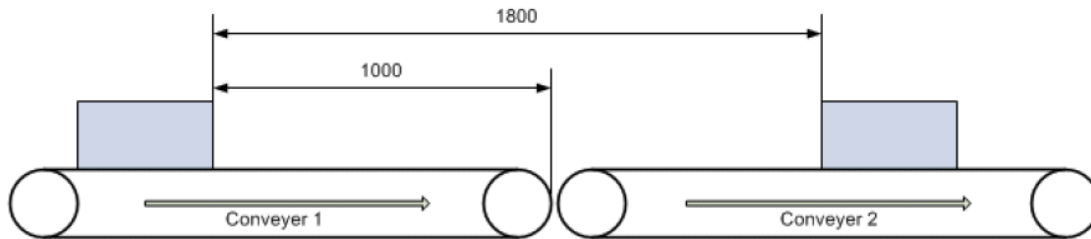
以规定的最大速度差 VelocityDiff 完成补偿, 补偿区间最短, 以 Length+Distance 为限。

**SUPERPOSITIONMODE\_LENGTHREDUCTION\_LIMITEDMOTION ,**

以规定的最大速度差 VelocityDiff 完成补偿, 补偿区间最短, 以 Length 为限。

产品传送带上的位置补偿

一条传送带分为若干段, 每段由一个伺服轴驱动。传送带用于传送包装箱, 包装箱之间必须保持正确的距离。如果不符合设定值, 就要增加或者减小, 包装箱必须在到达传送带终点之前, 比前段传送带走得更慢或者更快, 这就是位置补偿。



如图所示，当前测量距离是 1800mm，需要缩短至 1500mm。传送带 1 应加速，以缩短距离。距离补偿必须在传送带 1 到达终点之前完成，以免包装箱被推到速度更慢的传送带 2 上。

由于此时传送带 1 必须加速，传动系统要求给定速度差，在本例中假设为 500mm/s。实际应用中，该值取决于传送带的最大速度和当前设置速度之差。

功能块 MC\_MoveSuperImposed 的参数设置：

Distance = 1800 mm - 1500 mm = 300 mm (补偿距离)

Length = 1000 mm (补偿距离，此处用包装箱到传送带终点的距离)

Mode = SUPERPOSITIONMODE\_VELOREDUCTION\_LIMITEDMOTION

VelocityDiff = 500 mm/s

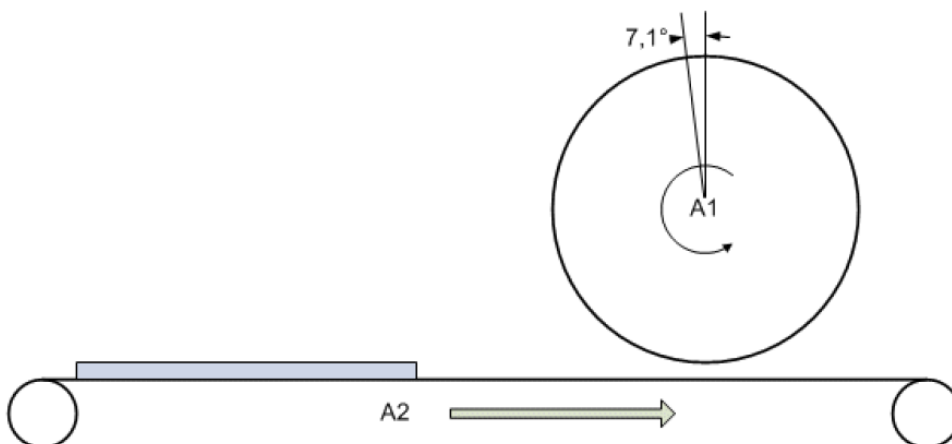
这种模式下，补偿距离为最大，以保持速度变化量为最小。此时速度差 VelocityDiff 的设定值是传送带 1 完成位置补偿的最大速度变化量。该值不能太小，以至传送带 1 用这个速度差走到终点还不能完成位置补偿。

另一种办法是让传送带 2 减速。此时，补偿位置 Distance 必须为负，而补偿距离 Length 为包装箱的右端到传送带 2 终点的距离。允许的最大速度差 VelocityDiff 相应改为传送带 2 的最大速度与当前速度之差。这样传送带 2 就可以减速，必要时甚至可以减为 0。

### (1) 印刷辊移相

印刷辊轮保持与印刷工件所在的传送带相同的速度匀速运动。如果辊轮上的印刷图案位置与工件上的设计印刷位置没有同步，印刷辊轮就必须补偿一个适当的角度（移相）。

如图所示：



移相可以有两种方式。

快速移相：在最短时间内修正相位角度，此时印刷辊轮必然发生速度冲击。

慢速移相：在尽可能长的距离内修正相位角度以减小速度冲击。比如，辊子转动完整



一圈。

功能块 MC\_MoveSuperImposedExt 的参数设置:

① 快速移相

Distance = 7.1°

Length = 360° (最大补偿距离)

Mode =SUPERPOSITIONMODE\_LENGTHREDUCTION\_LIMITEDMOTION

VelocityDiff = 30° /s(速度差)

此模式下, 补偿距离尽可能短。此时补偿距离 Length 的设定值是辊轮完成移相的最大距离。该值不能太小, 以至用最大速度也不能在这么短的距离内完成位置补偿。

也可选择模式 SUPERPOSITIONMODE\_LENGTHREDUCTION\_ADDITIVEMOTION。此时, 补偿距离为 367.1°。由于补偿距离都是尽可能短, 实际上对于这种应用, 两种模式结果相同。

② 慢速移相

Distance = 7.1°

Length = 360°(correction distance)

Mode =SUPERPOSITIONMODE\_VELOREDUCTION\_LIMITEDMOTION

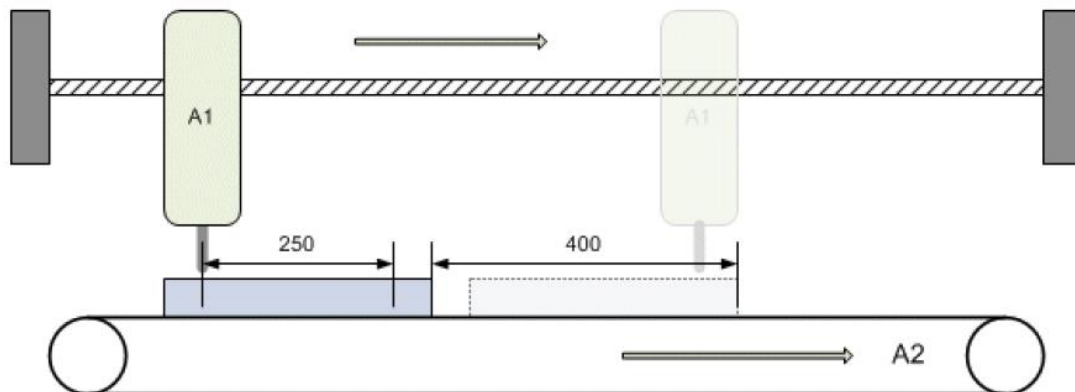
VelocityDiff = 30° /s(速度差)

这种模式下, 补偿距离为最大, 以保持速度变化量为最小。此时速度差 VelocityDiff 的设定值是辊轮完成移相的最大速度变化量。该值不能太小, 以至辊轮用这个速度差走完一圈还不能完成位置补偿。

(2) 钻削设备

钻头要在运动的工件上钻两个孔。第一个孔的同步是通过飞锯功能(MC\_GearInPos)实现的, 在此不再详述。完成第一个孔后, 钻头必须相对于运动工件移动一定的距离。

如图所示:



图中设备完成第一个孔后, 钻头必须相对于工件移动 250mm, 即两孔之间的距离。而在这段时间内, 工件本身移动的距离是 400mm。从这个位置开始, 钻头再次与工件同步, 然后钻第二个孔。

同样, 这里也可以有两种模式可供选择, 区别在于钻头的速度变化值。

功能块 MC\_MoveSuperImposed 的参数设置:

① 快速补偿

Distance = 250 mm

Length = 400 mm

Mode = SUPERPOSITIONMODE\_LENGTHREDUCTION\_ADDITIVEMOTION

VelocityDiff = 500 mm/s(钻头移动速度最大变化量)

在此模式下，补偿距离尽可能短。此时补偿距离 Length 的设定值是钻头完成补偿的最大距离。该值不能太小，以至用最大速度也不能在这么短的距离内完成补偿。由于补偿距离是工件走过的距离加上相对位移，所以钻头实际上要走一个更长的距离。

## ② 慢速补偿

Distance = 250 mm

Length = 400 mm

Mode = SUPERPOSITIONMODE\_VELOREDUCTION\_ADDITIVEMOTION

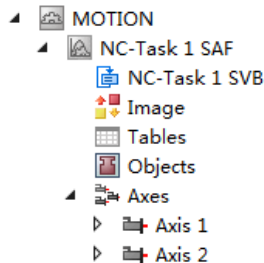
VelocityDiff = 500 mm/s(钻头移动速度最大变化量)

这种模式下，补偿距离为最大，以保持速度变化量为最小。此时速度差 VelocityDiff 的设定值是钻头完成补偿的最大速度变化量。该值不能太小，以至钻头用这个速度差走全程还不能完成位置补偿。在此过程中，工件走过距离 Length 为 400mm，钻头走过的距离就是 Length + Distance，即 650mm。

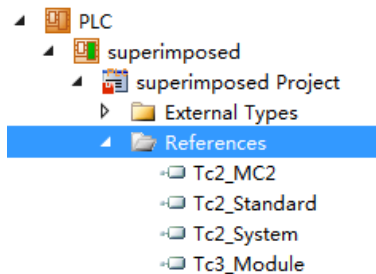
## 2. 实例程序

以适用场合中的案例一“产品传送带上的位置补偿”为例子。

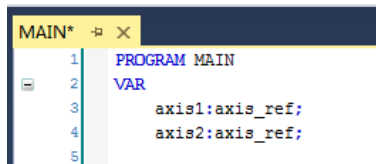
### (1) 首先在 MOTION 下方创建一个虚轴



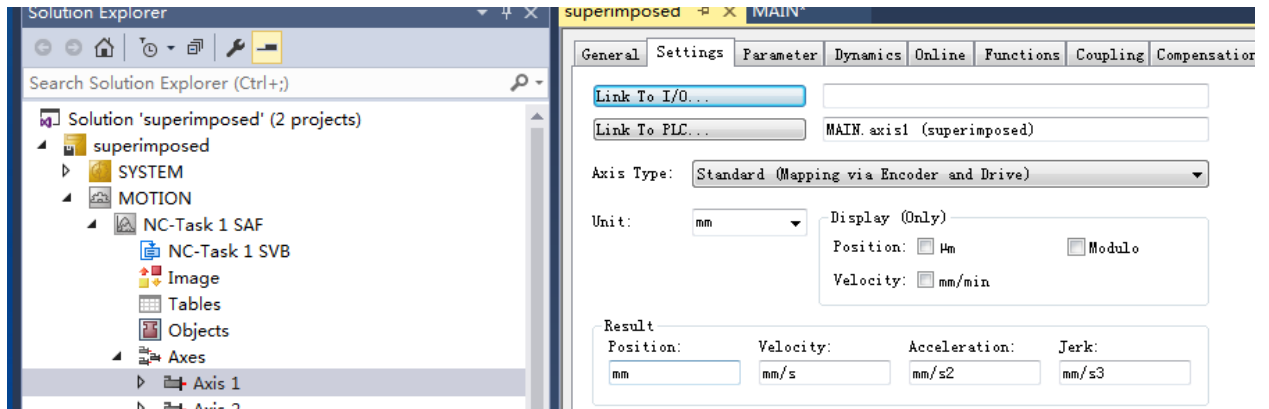
### (2) 在 PLC 下方创建项目，命名为 superimposed，添加库文件 Tc2\_MC2.lib



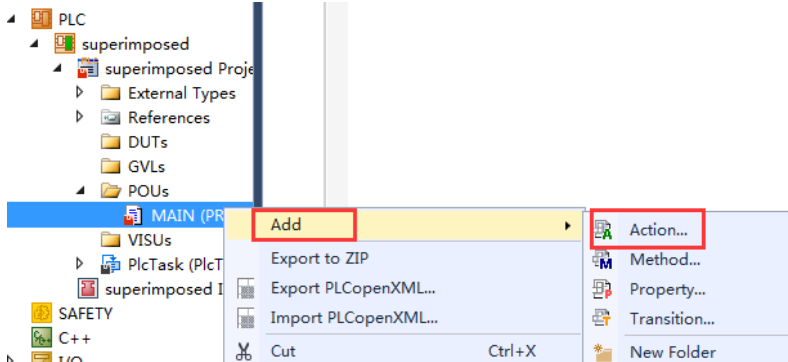
### (3) MAIN 区定义轴变量



并且和 NC 轴做关联，轴 1 如下，轴 2 同轴 1 类似操作（省略）



定义如下 ACTION，命名为“power”用于轴使能，使用 ST 语言编辑。



在 ACTION 中的编程如下：

```

MAIN.power  MAIN*
1  axis1.PlcToNc.ControlDWord:=7;
2  axis1.PlcToNc.Override:=1000000;
3  axis2.PlcToNc.ControlDWord:=7;
4  axis2.PlcToNc.Override:=1000000;

```

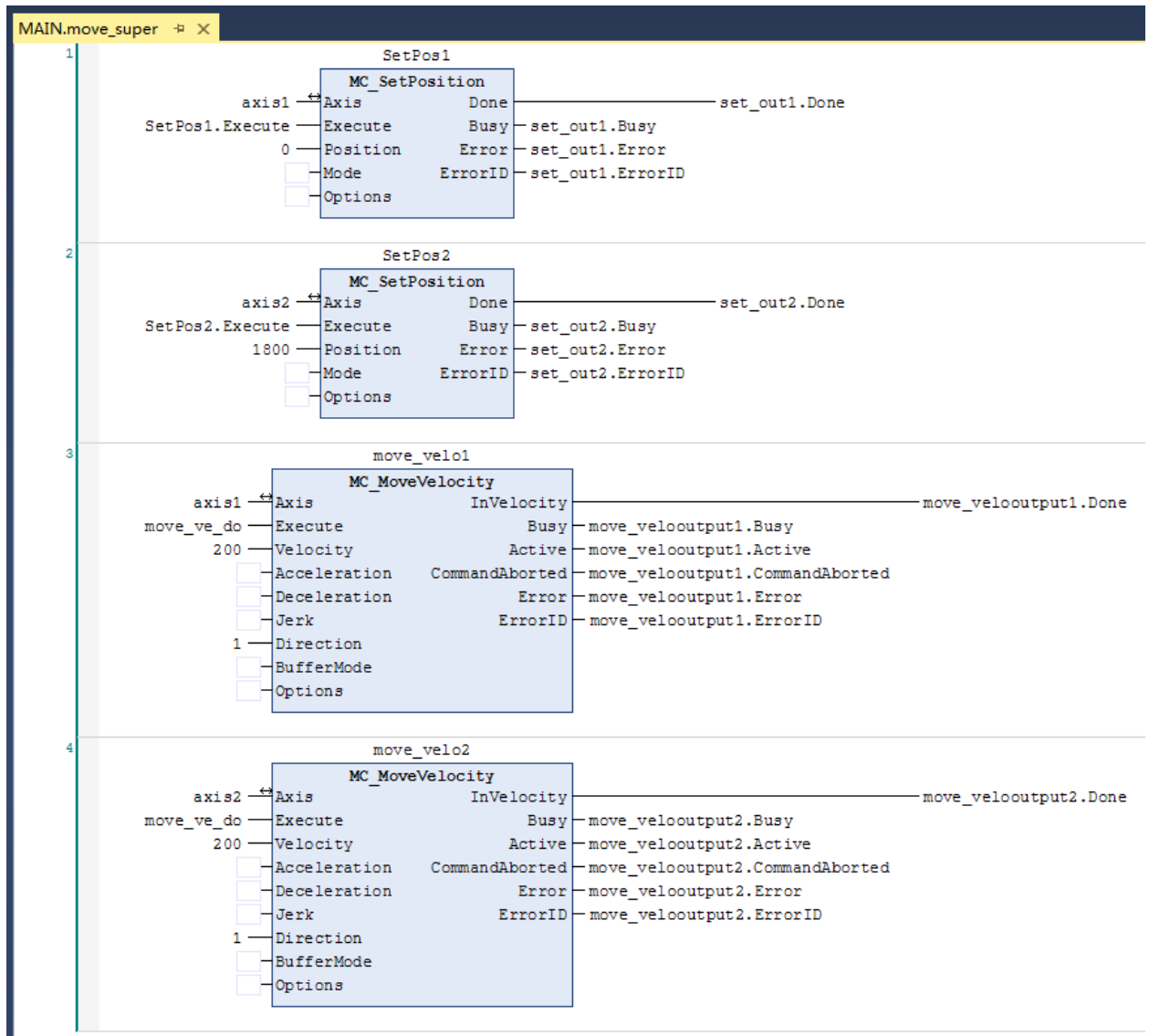
(4) 创建一个 ACTION，命名为 move\_super，使用 FBD 语言编程。为了模拟该实例，所以将轴 1 轴 2 的当前位置设成为 0 和 1800，同时调用轴匀速运动功能块。首先在 MAIN 区申明如下：

```

MAIN.power  MAIN*
1  PROGRAM MAIN
2  VAR
3      axis1:axis_ref;
4      axis2:axis_ref;
5
6      SetPos1,SetPos2: MC_SetPosition;
7      set_out1,set_out2:st_mcoutputs;
8      set_do: BOOL;
9
10     move_velo1: MC_MoveVelocity;
11     move_velo2: MC_MoveVelocity;
12     move_ve_do: BOOL;
13     move_velooutput1,move_velooutput2:st_mcoutputs;
14

```

其次在 ACTION 中的调用如下：



(5) 创建一个 ACTION 命名为 `superimposed`，采用 FBD 语言编程，该 ACTION 用于调用位置补偿功能，首先在 MAIN 区申明如下：

```

move_super    MAIN.superimposed    MAIN*  [X]
PROGRAM MAIN
VAR
  axis1:axis_ref;
  axis2:axis_ref;

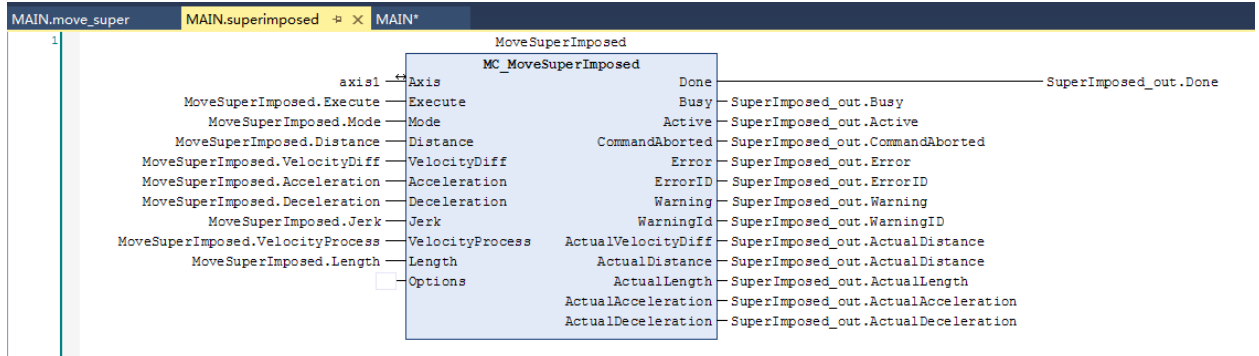
  SetPos1,SetPos2: MC_SetPosition;
  set_out1,set_out2:st_mcoutputs;
  set_do: BOOL;

  move_velo1: MC_MoveVelocity;
  move_velo2: MC_MoveVelocity;
  move_ve_do: BOOL;
  move_velooutput1,move_velooutput2:st_mcoutputs;

  MoveSuperImposed: MC_MoveSuperImposed;
  SuperImposed_out:superimposed_actural;
  SUPERPOSITIONMODE: E_SuperpositionMode:=SUPERPOSITIONMODE VELOREDUCTION LIMITEDMOTION;

```

其次在该 ACTION 中的编程如下：



- (6) 接下来在 MAIN 区做如下申明一个变量，用于计算轴 1 和轴 2 之间的位置差，定义如下：

```

Scope YT Project1*    MAIN*  [X]
6   SetPos1,SetPos2: MC_SetPosition;
7   set_out1,set_out2:st_mcoutputs;
8   set_do: BOOL;
9
10  move_velo1: MC_MoveVelocity;
11  move_velo2: MC_MoveVelocity;
12  move_ve_do: BOOL;
13  move_velooutput1,move_velooutput2:st_mcoutputs;
14
15  MoveSuperImposed: MC_MoveSuperImposed;
16  SuperImposed_out:superimposed_actural;
17  SUPERPOSITIONMODE: E_SuperpositionMode:=SUPERPOSITIONMODE_VELOREDUCTION LIMITEDMOTION;
18
19  posdiff: LREAL;
20  END_VAR

```

- (7) 在程序区首先调用轴变量用于更新轴状态；调用三个 ACTION，为后续编程做准备，如下：

```
MAIN* -> X
1 PROGRAM MAIN
2 VAR
3     axis1:axis_ref;
4     axis2:axis_ref;
5
6     SetPos1,SetPos2: MC_SetPosition;
7     set_out1,set_out2:st_mcoutputs;
8     set_do: BOOL;
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

- (8) 设置位置补偿根据应用，可知：  
当前测量距离是 1800mm，需要缩短至 1500mm。传送带 1 应加速，以缩短距离。距离补偿必须在传送带 1 到达终点之前完成，以免包装箱被推到速度更慢的传送带 2 上。  
由于此时传送带 1 必须加速，传动系统要求给定速度差，在本例中假设为 500mm/s。  
实际应用中，该值取决于传送带的最大速度和当前设置速度之差。

功能块 MC\_MoveSuperImposed 的参数设置：

Distance = 1800 mm-1500 mm = 300 mm(补偿距离)

Length = 1000 mm(补偿距离，此处用包装箱到传送带终点的距离)

Mode = SUPERPOSITIONMODE\_VELOREDUCTION\_LIMITEDMOTION

VelocityDiff = 500 mm/s

综上，对位置补偿功能在主程序区的程序编辑区设置如下：

```
1 axis1();
2 axis2();
3
4 power();
5 move_super();
6 superimposed();
7
8 MoveSuperImposed.Execute:=move_ve_do;
9 MoveSuperImposed.Mode:=2;
10 MoveSuperImposed.Distance:=300;
11 MoveSuperImposed.VelocityDiff:=500;
12 movesuperimposed.VelocityProcess:=700;
13 movesuperimposed.Length:=1000;
14
```

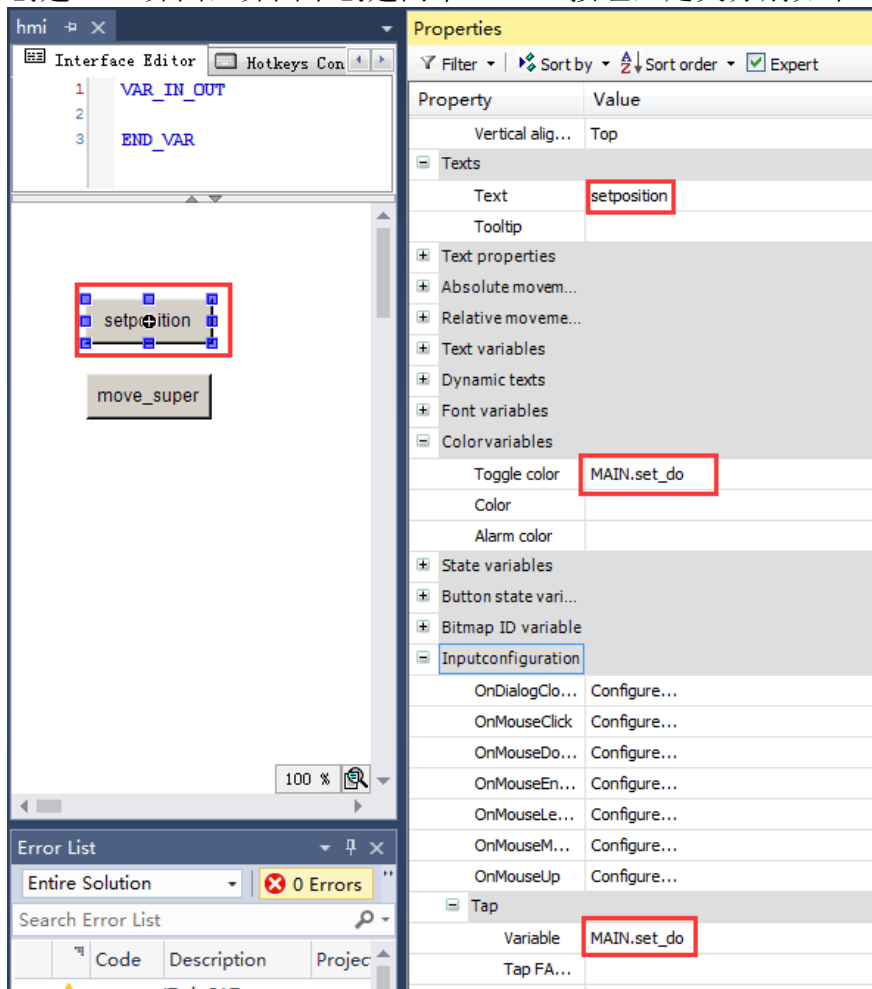
- (9) 接下来计算两轴的位置差值，并且对修改当前位置的触发位和匀速运动的触发位绑定变量

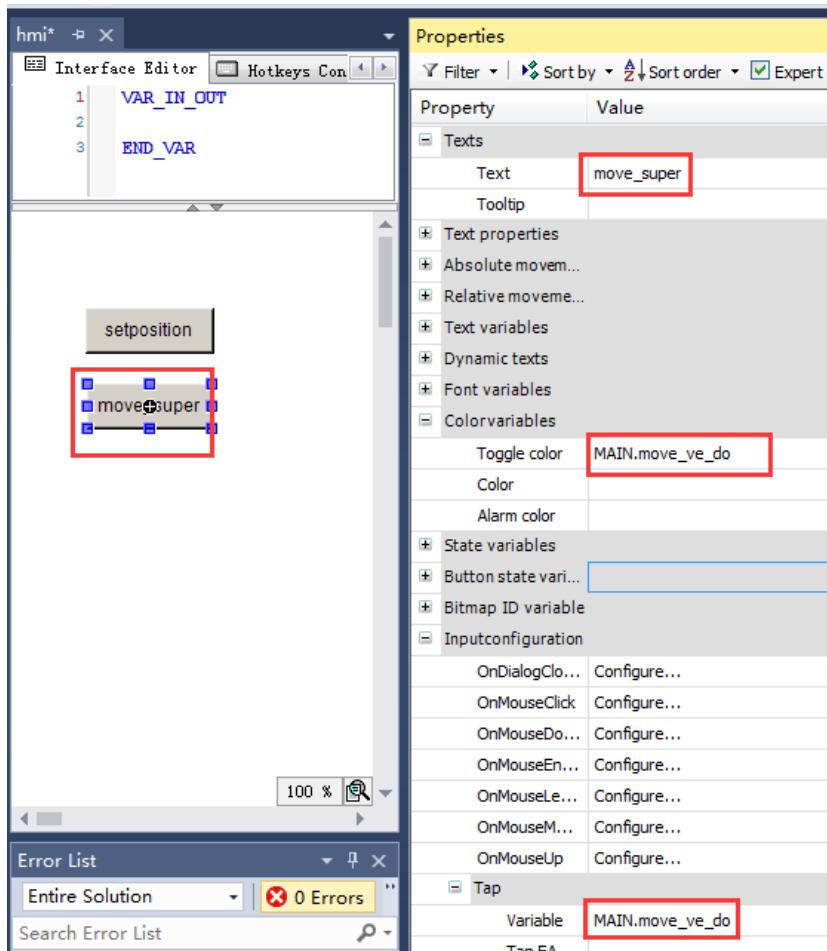
```

1  axis1();
2  axis2();
3
4  power();
5  move_super();
6  superimposed();
7
8  MoveSuperImposed.Execute:=move_ve_do;
9  MoveSuperImposed.Mode:=2;
10 MoveSuperImposed.Distance:=300;
11 MoveSuperImposed.VelocityDiff:=500;
12 movesuperimposed.VelocityProcess:=700;
13 movesuperimposed.Length:=1000;
14
15 posdiff:=axis2.NcToPlc.SetPos-axis1.NcToPlc.SetPos;
16 SetPos1.Execute:=set_do;
17 setpos2.Execute:=set_do;
18 move_velo1.Execute:=move_ve_do;
19 move_velo2.Execute:=move_ve_do;

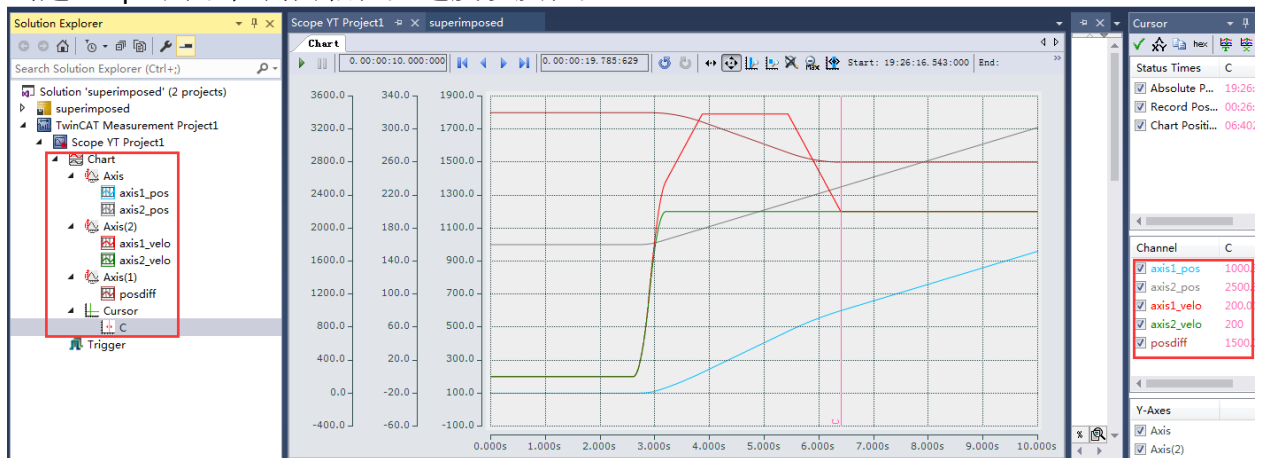
```

(10) 创建 HMI 界面，界面中创建两个 button 按钮，定义分别如下：





(11) 创建 scope 用于检测两轴位置速度以及位置差



可以看出轴 1 到 1000 的位置，轴 2 到达 2500，正好完成补偿，同时速度也降为 200，也就意味着正好轴 1 以 200 的速度上 2 号传送带。

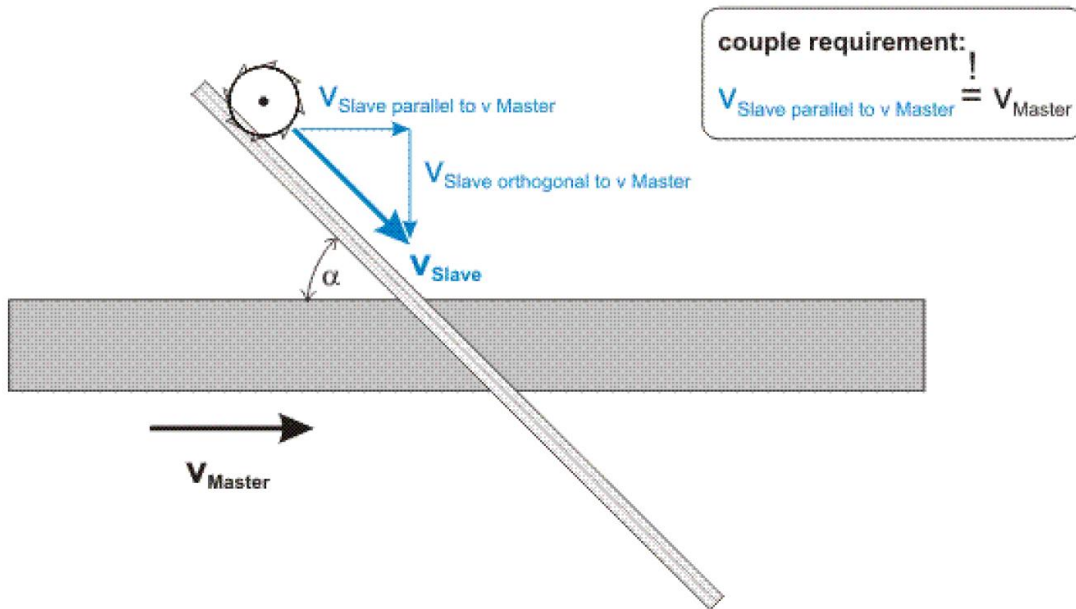


## 七、飞锯功能

### 1. 飞锯功能介绍

飞锯是指从轴可以同步到正在运动的主轴，并与主轴同步运行以完成一个加工周期。这种同步到主轴的运动，意味着工件可以在传输的过程中进行加工。在许多工厂的工件都需要在传输的过程中进行加工作业，为了实现这个目的，需要对传输过程中的工件和刀具进行位置和速度的同步，这样刀具就可以对相对静止的工件进行加工，为了实现此类功能，TwinCAT 提供了 FlyingSaw（飞锯），包含在库文件 Tc2\_Mc2\_FlyingSaw 中。

同步运行阶段主从轴速度之比，由参数“耦合系数”（coupling factor）给定。例如斜切时耦合系数不等于 1，于是同步阶段，从轴在主轴运动方向上的速度分量（ $v_{\text{slave parallel to } v_{\text{master}}}$ ）与主轴速度相等。（如下图所示）



飞锯有两种同步方式：速度同步和位置同步。速度同步时，从轴按照耦合系数 **coupling factor** 运算速度尽快同步到主轴，因此主从轴的耦合位置，就是各个参数允许的前提下最快达到同步的位置。位置同步时，用户通过参数设定主从轴的同步位置，主从轴将在最新指定的位置实现同步运行。

这两种方式都要求定义同步阶段的约束条件，即同步模式 **SyncMode**，以便根据工艺需求调整同步动作。

两种模式：

#### (1) 速度同步

速度同步时，从轴使用指定的最大加减速尽快同步到主轴。在同步阶段，从轴速度与主轴成正比，即：

$$V_{\text{Slave}} = F_{\text{CouplingFactor}} * V_{\text{Master}}$$

同步顺序：从轴到主轴的同步过程依次经过一下顺序：

①启动通用飞锯，和电子齿轮类似，飞锯启动的时间即为耦合时间。通用飞锯的耦合对于主从轴的停止或运行状态没有特殊要求，耦合过程可以在主从轴都运动的情况下进行。

- ②同步阶段：从轴从当前状态变速到主轴速度，同时监视从轴的运动不超过用户定义的约束条件（例如加速度不超过设定值）。从开始加速到同步运行的时间，就称为同步时间。
- ③ 同步运行阶段：从轴和主轴同步运行。
- ④通用飞锯解耦：这是在线修改。被耦合的从轴恢复为一个独立主轴，以解耦时的速度继续运动。
- ④ 解耦后的从轴可以重新启动或者停止，并且可以使用 TwinCAT NC PTP 中的任意功能块控制该轴。

## (2) 位置同步

位置同步时，要求从轴以指定的加减速速度在指定的同步位置达到与主轴同步。这意味着需要从轴需要恰好在同步位置达到同步速度。然后，从轴与主轴保持同步运行，在同步运行阶段，从轴速度与主轴速度成正比：位置同步时，要求从轴以指定的加减速速度在指定的同步位置达到与主轴同步。这意味着需要从轴需要恰好在同步位置达到同步速度。然后，从轴与主轴保持同步运行，在同步运行阶段，从轴速度与主轴速度成正比：

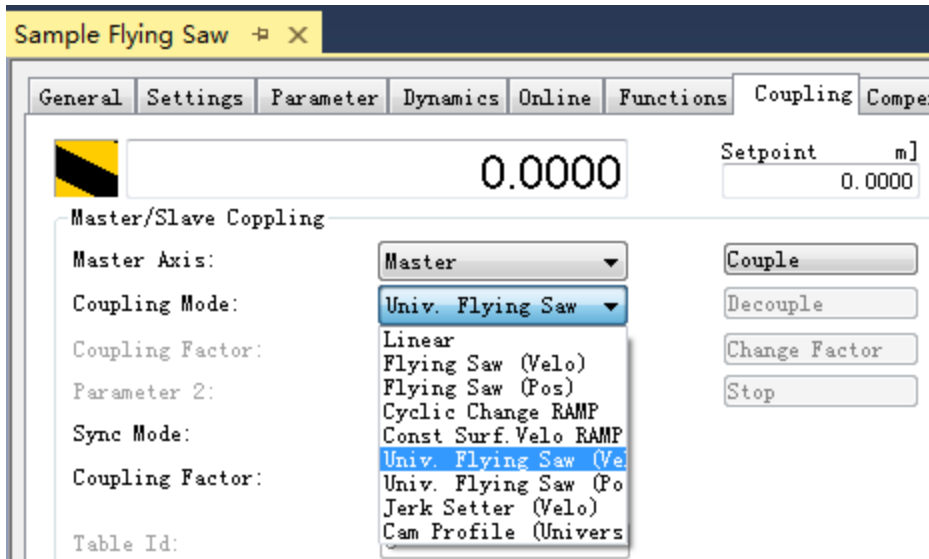
$$V_{Slave} = F_{CouplingFactor} * V_{Master}$$

- ①启动通用飞锯，和电子齿轮类似，飞锯启动的时间即为耦合时间。通用飞锯的耦合对于主从轴的停止或运行状态没有特殊要求，耦合过程可以在主从轴都运动的情况下进行。与速度模式不同，位置模式的飞锯启动后，从轴不会立即动作，而是根据加速度计算启动时刻，以保证在指定的位置达到和主轴匹配的速度。
- ②同步阶段：从轴从初始条件加速到主轴速度，在主轴同步位置以同步速度准确到达从轴同步位置。在此过程中，必须保证从轴的运动不超出用户定义的约束条件，（比如加速度不超过设定值）。从开始加速到同步运行的时间，就称为同步时间。
- ③ 同步运行阶段：从轴和主轴同步运行。
- ④ 通用飞锯解耦：这是在线修改。被耦合的从轴恢复为一个独立主轴，以解耦时的速度继续运动。
- ⑤ 解耦后的从轴可以重新启动或者停止，并且可以使用 TwinCAT NC PTP 中的任意功能块控制该轴

## 2. NC 界面中的 FlyingSaw 调试

在 TwinCAT3 Motion 中可以直接进行飞锯功能的调试，在这种情况下，实际上是 TwinCAT3 Motion 在后台通过 ADS 通讯控制 NC/PLC 接口，以接替 PLC 程序对 NC/PLC 接口的操作。

- (1) 首先在 TwinCAT3 左侧 Motion 下创建名称为 Master 和 Slave 的两根轴。创建完成后，点击 激活配置。
- (2) TwinCAT 切换到运行状态后，单击 Motion Axes Slave，找到 Slave 轴右边的 Coupling 选项卡，在 Coupling 选项卡下可以进行多轴耦合调试，首先在 Master Axis 下拉选项中选择飞锯的主轴 Master，选择 Coupling Mode，在 NC 调试界面的下拉选项中选择 Univ.FlyingSaw（Velo）速度模式的飞锯耦合，选择对应的模式后，点击右边 couple 按钮进行主从轴耦合。



### 3. 功能块介绍（基于 Tc2\_Mc2\_FlyingSaw 库）

在许多工厂中，都需要对传输过程中的工件进行加工，为此，进行加工的刀具的位置和速度必须和工件保持同步，刀具和工件之间相对静止才可以进行加工作业。为了实现此类应用，TwinCAT3 提供了飞锯功能，用户可以在 References 中加载 Tc2\_MC2\_FlyingSaw 库文件。

飞锯的同步模式有速度同步和位置同步，分别用功能块 MC\_GearInVelo 和 MC\_GearInPos 实现。库文件中的另一个功能块 MC\_ReadFlyingSawCharacteristics 用于读取同步阶段的从轴的特征参数。下面分别对这几个功能块的使用作简单说明。

#### (1) MC\_GearInVelo



MC\_GearInVelo 把一个从轴作为飞锯，以速度同步的方式耦合到主轴。速度耦合对实现同步的位置没有要求，按照设定要求尽快实现同步。这个功能块的接口和功能都是 PLCOpen 组织定义的。

飞锯的从轴可以通过功能块 MC\_GearOut 进行解耦。如果解耦时从轴还在运动状态，解耦后将会保持解耦时速度继续运行，可以使用 MC\_Halt 或者 MC\_Stop 来对从轴进行减速或停止动作。

Excute	BOOL		上升沿触发飞锯同步
RatioNumerator	LREAL	=1	耦合系数的分子
RatioDenominator	UINT	=1	耦合系数的分母

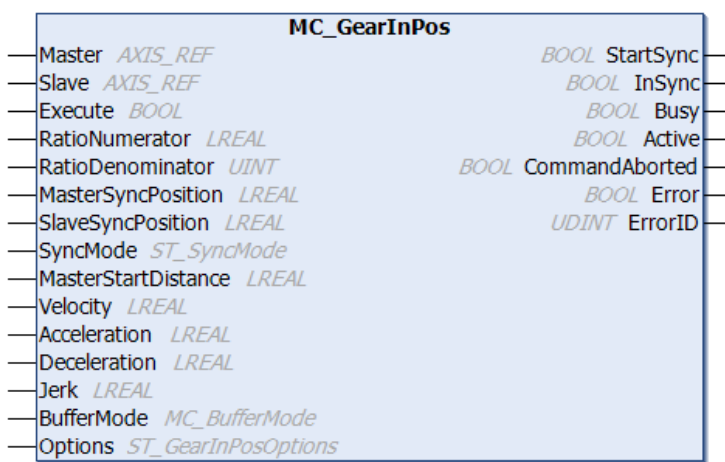
SyncMode	ST_SyncMode		同步模式，通过设置具体校验参数对从轴在实现同步过程中的临界状态作限定
Velocity	LREAL		对同步阶段的从轴的最大速度/加速度/减速度/抖动。此处未定义数值，则使用 System Manager 中的相应设定。 注意：此处给定的参数，仅当同步模式中激活了相应的校验选项，才会进行校验。 (GearInSync_CheckMask_MaxVel:=TRUE;) (GearInSync_CheckMask_MaxAcc:=TRUE;) (GearInSync_CheckMask_MaxDec:=TRUE;) (GearInSync_CheckMask_MaxJerk:=TRUE;)
Acceleration	LREAL		
Deceleration	LREAL		
Jerk	LREAL		
BufferMode	MC_BufferMode		功能目前暂未实现
Options	ST_GearInVelocityOption		

说明：如果耦合系数要设置成 1:4，则 RatioNumerator（分子）设置成 1，RatioDenominator（分母）应为 4。也可以选择将 RatioDenominator 设置成 1，此时 RatioNumerator 可以设置成浮点型数值 0.25。RatioNumerator（分子）可以是负数。

#### 输出变量

StartSync	同步开始后即置 TRUE
InSync	同步完成后即置 TRUE
Busy	Execute 触发后 Busy 会被置成 TRUE 直到同步过程执行完成后，Busy 才会重新编程 False，此时功能块可以执行第二次同步，与此同时输出变量 InSync,CommandAborted 和 Error 都会复位
Active	表示同步动作已经执行（目前 Active=Busy）
CommandAborted	耦合中断即置 TRUE
Error	发生错误即置 TRUE
ErrorID	Error 置 TRUE 后，此变量显示错误代码
Master	进行飞锯耦合的主轴
Slave	进行飞锯耦合的从轴

#### (2) MC\_GearInPos



MC\_GearInPos 把一个从轴作为飞锯，以位置同步方式耦合到主轴。在主轴和从轴的同

步位置实现精确地速度同步。这个功能块的接口和功能都是 PLCOpen 组织定义的。

飞锯的从轴可以通过功能块 MC\_GearOut 进行解耦。如果解耦时从轴还在运动状态，解耦后将会保持解耦时速度继续运行，可以使用 MC\_Halt 或者 MC\_Stop 来对从轴进行减速或停止动作。

输入变量

Excute	BOOL		上升沿触发飞锯同步
RatioNumerator	LREAL	=1	耦合系数的分子
RatioDenominator	UINT	=1	耦合系数的分母
SyncMode	ST_SyncMode		同步模式，通过设置具体校验参数对从轴在实现同步过程中的临界状态作限定
MasterSyncPosition	LREAL		主轴同步位置
SlaveSyncPosition	LREAL		从轴同步位置
Velocity	LREAL		对同步阶段的从轴的最大速度/加速度/减速度/抖动。此处未定义数值，则使用 System Manager 中的相应设定。 注意：此处给定的参数，仅当同步模式中激活了相应的校验选项，才会进行校验。 (GearInSync_CheckMask_MaxVel:=TRUE;) (GearInSync_CheckMask_MaxAcc:=TRUE;) (GearInSync_CheckMask_MaxDec:=TRUE;) (GearInSync_CheckMask_MaxJerk:=TRUE;)
Acceleration	LREAL		
Deceleration	LREAL		
Jerk	LREAL		
BufferMode	MC_BufferMode		功能目前暂未实现
Options	ST_GearInVelocityOption		

输出变量

StartSync	同步开始后即置 TRUE
InSync	同步完成后即置 TRUE
Busy	Excute 触发后 Busy 会被置成 TRUE 直到同步过程执行完成后，Busy 才会重新编程 False，此时功能块可以执行第二次同步，与此同时输出变量 InSync,CommandAborted 和 Error 都会复位
Active	表示同步动作已经执行（目前 Active=Busy）
CommandAborted	耦合中断即置 TRUE
Error	发生错误即置 TRUE
ErrorID	Error 置 TRUE 后，此变量显示错误代码
Master	进行飞锯耦合的主轴
Slave	进行飞锯耦合的从轴

其中：SyncMode（同步模式）

从轴运动自初始状态到同步运行状态的加速过程，必须严格遵守用户定义的约束条件，这些约束条件包括：限制从轴最大速度，防止位置过冲等等。无论是速度同步还是位置同步，都需要约定这个约束条件，这就是同步模式 SyncMode，其类型为结构体 ST\_SyncMode，包含以下参数：

(* mode *)		
GearInSyncMode	E_GearInSyncMode	耦合同步模式

(* 32 bit check mask ... *)		
GearInSync_CheckMask_MinPos	BOOL	是否校验最小位置
GearInSync_CheckMask_MaxPos	BOOL	是否校验最大位置
GearInSync_CheckMask_MaxVelo	BOOL	是否校验最大速度
GearInSync_CheckMask_MaxAcc	BOOL	是否校验最大加速度
GearInSync_CheckMask_MaxDec	BOOL	是否校验最大减速度
GearInSync_CheckMask_MaxJerk	BOOL	是否校验最大抖动
GearInSync_CheckMask_OvershootPos	BOOL	是否校验位置过冲
GearInSync_CheckMask_UndershootPos	BOOL	是否校验位置静差
GearInSync_CheckMask_OvershootVelo	BOOL	是否校验速度过冲
GearInSync_CheckMask_UndershootVelo	BOOL	是否校验速度静差
GearInSync_CheckMask_OvershootVeloZero	BOOL	是否校验零速过冲
GearInSync_CheckMask_UndershootVeloZero	BOOL	是否校验零速静差
(* operation masks ... *)		
GearInSync_OpMask_RollbackLock	BOOL	反转锁定
GearInSync_OpMask_InstantStopOnRollback	BOOL	反转时停止
GearInSync_OpMask_PreferConstVelo	BOOL	恒速优先
GearInSync_OpMask_IgnoreMasterAcc	BOOL	忽略主轴加速

其中 E\_GearInSyncMode 是一个枚举变量，包含以下两项

GEARINSYNCMODE_POSITIONBASED	基于主轴位置的同步，根据主轴加减速，从轴相应加速和减速。
GEARINSYNCMODE_TIMEBASED	基于时间的同步曲线，主从轴的加减速动态特性是独立的。

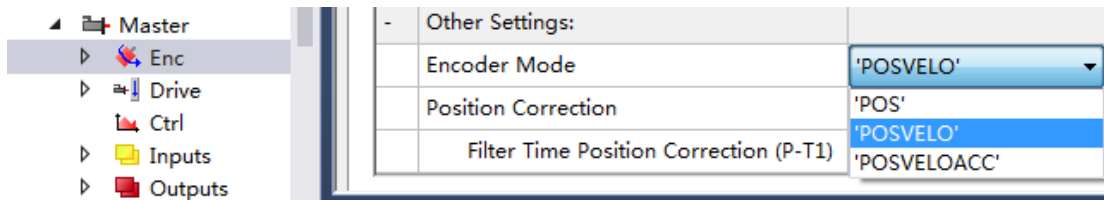
注意：基于时间的同步曲线（GEARINSYNCMODE\_TIMEBASED）仅适用于 MC\_GearInVelo 功能块中。

### (3) MC\_ReadFlyingSawCharacteristics 飞锯同步特征值

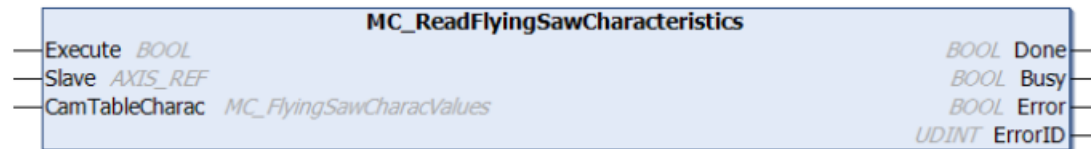
飞锯启动的过程中，系统会自动计算各项特征值，并根据同步模式 SyncMode 中设定的约束条件进行校验。原则上主从轴在任意状态下都可以计算这些特征值，但在实际操作中，因为耦合时不知道主轴将要执行的动作，所以假设主轴保持匀速运动，没有加速度，否则无法计算和校验各项特征值。耦合以后，如果主轴加速，从轴也会相应加速。所以即使是校验通过的参数，主轴加速度过大时也可能出现过冲或者静差。

通用飞锯启动后，用户可以使用 MC\_ReadFlyingSawCharacteristics 功能块访问从轴同步阶段使用的动作特征参数。读回的结果包括各种限值，例如最大从轴加速度，最大最小从轴位置等等。这些值都是在假定主轴速度不变的前提下计算的，仅在此条件下，这些校验值才正确。

因为通用飞锯启动时，主轴加速度对于曲线的计算和优化影响巨大，这意味着如果主轴是一个编码器，速度和加速度必须小心滤波，或者不要选中实际加速度的运算。



选择 Encoder Mode 为 POS 或者 POSVELO，不要设置为 POSVELO  
在 PLC 程序中读取飞锯耦合特征值的功能块是：



此功能块仅用于 PLC 程序读取特征值并显示。即使不调用，飞锯功能块 MC\_GearInVelo 和 MC\_GearInPos 也会自动计算这些特征参数，并依据同步模式 SyncMode 的设置同步阶段使用。

输入变量：

Execute:	上升沿时，从 TwinCAT NC 读取飞锯的特征值 注意：只有飞锯启动以后才能读取。
Slave	飞锯从轴
CamTableCharac	读取的结果，飞锯特征值。 这些特征值仅用于同步阶段

输出变量：

Done:	成功读取后置 True
Error:	发生错误后置 True
ErrorID:	Error 置 True 后提供错误代码

CamTableCharac 包含以下内容：

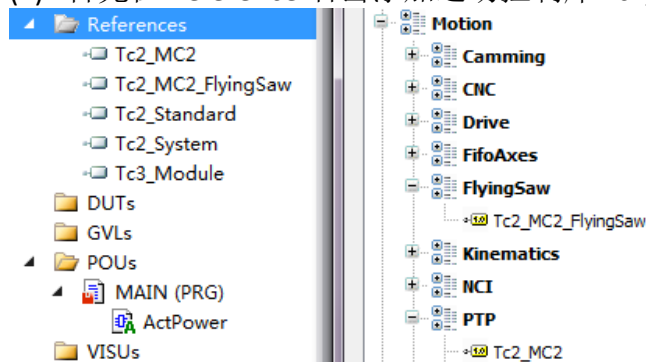
飞锯特征值	描述	是否受主轴加速度影响
fMasterVeloNom	飞锯启动时，主轴速度	no
fMasterPosStart	飞锯启动时，主轴位置	yes
fSlavePosStart	飞锯启动时，从轴位置	yes
fSlaveVeloStart	飞锯启动时，从轴速度	no
fSlaveAccStart	飞锯启动时，从轴加速度	no
fSlaveJerkStart	飞锯启动时，从轴抖动	no
fMasterPosEnd	同步阶段结束时，主轴位置	yes
fSlavePosEnd	同步阶段结束时，从轴位置	yes
fSlaveVeloEnd	同步阶段结束时，从轴速度	no
fSlaveAccEnd	同步阶段结束时，从轴加速度	no
fSlaveJerkEnd	同步阶段结束时，从轴抖动	no
fMPosAtSPosMin	从轴在最小位置时，主轴位置	no
fSlavePosMin	从轴最小的位置	yes
fMPosAtSVeloMin	从轴在最小速度时，主轴速度	no
fSlaveVeloMin	从轴最小的速度	no
fMPosAtSAccMin	从轴在最小加速度时，主轴位置	no
fSlaveAccMin	从轴最小的加速度	no
fSVeloAtSAccMin	从轴在最小加速度时，从轴速度	no



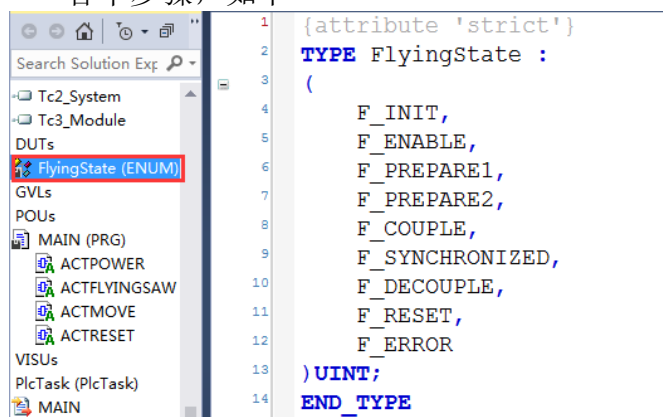
fSlaveJerkMin	从轴最小的抖动	no
fSlaveDynMomMin	最小从轴动态力矩（暂不支持）	no
fMPosAtSPosMax	从轴在最大位置时，主轴的位置	no
fSlavePosMax	从轴最大位置	yes
fMPosAtSVeloMax	从轴在最大速度时，主轴的位置	no

#### 4. 案例程序

(1) 首先在 Reference 右击添加运动控制库 Tc2\_mc2 和飞锯库 Tc2\_MC2\_FlyingSaw



(2) 按照位置模式下飞锯的实现流程，创建枚举体 FlyingState 来表示位置模式下的飞锯的各个步骤，如下



(3) 首先在 MAIN 程序中创建轴变量，命名为 Master 和 Slave，同时对创建好的枚举体作实例化

```

3      Master          :axis_Ref;
4      Slave           :axis_Ref;
5
6      state           : FlyingState;

```

(4) 创建使能功能块分别对主从轴进行使能

```

8      (*Power Part*)
9      PowerMaster     : MC_Power;
10     PowerMasterOut  : ST_McOutputs;
11     PowerSlave      : MC_Power;
12     PowerSlaveOut   : ST_McOutputs;
13     PowerEnable     : BOOL;

```

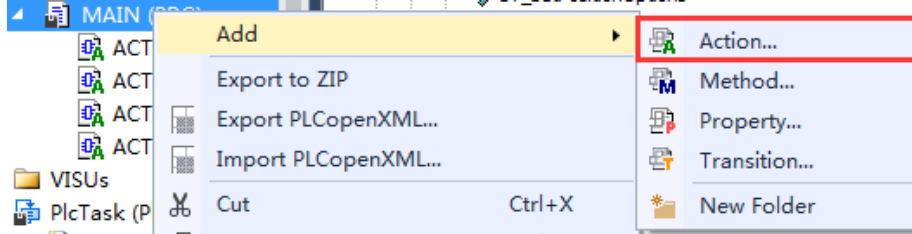
其中 ST\_McOutputs 是 Tc2\_MC2 库文件中的结构体，该结构体定义了功能块的通用输出变量接口，使用户可以对输出变量作统一实例化，无需声明单个变量，十分方便



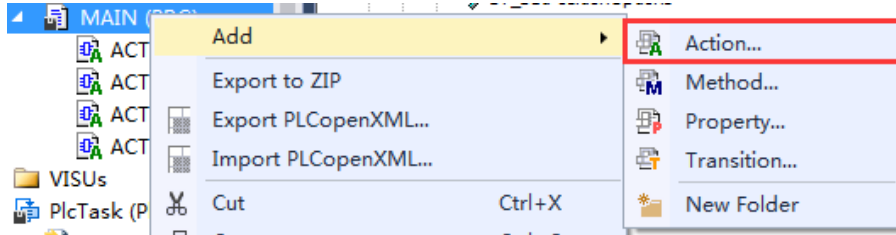
STRUCT ST\_McOutputs

Name	Type	Inherited from	Address	Initial	Comment
Done	BOOL				
Busy	BOOL				
Active	BOOL				
CommandAborted	BOOL				
Error	BOOL				
ErrorID	UDINT				

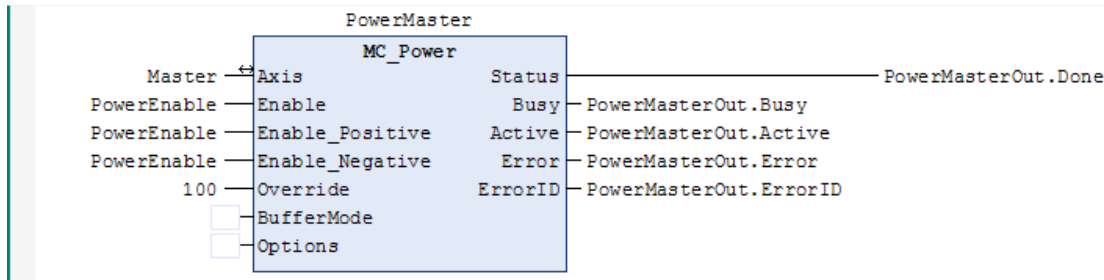
接下去在 MAIN 程序右击，创建 ACTION 来实现使能功能



接下去在 MAIN 程序右击，创建 ACTION 来实现使能功能



选择 ACTION 的编程语言为 FBD，以主轴为例如下定义输入输出接口



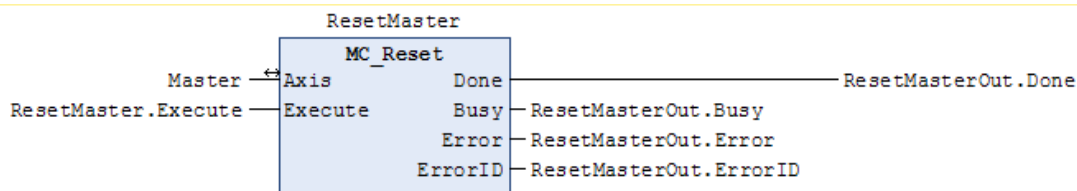
(5) 创建复位功能块，对轴报错进行复位

```

15 (*Reset Part*)
16 ResetMaster      : MC_Reset;
17 ResetMasterOut   : ST_McOutputs;
18 ResetSlave       : MC_Reset;
19 ResetSlaveOut    : ST_McOutputs;

```

同样在 MAIN 程序下创建 ACTION 来实现复位，接口定义如下



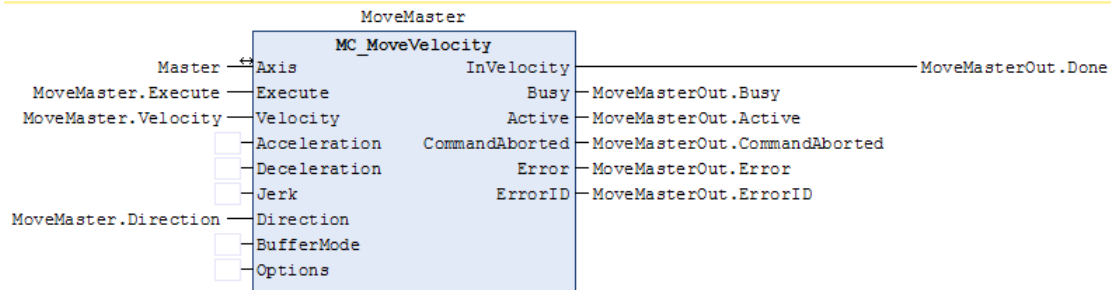
(6) 主从轴完成使能后，需要让主轴首先进行动作，从轴在飞锯启动后进行位置模式下的飞锯耦合，使用 MC\_MoveVelocity 让主轴以恒定速度运行

```

22      MoveMaster          : MC_MoveVelocity;
23      MoveMasterOut       : ST_McOutputs;

```

创建 ACTION 实现该功能



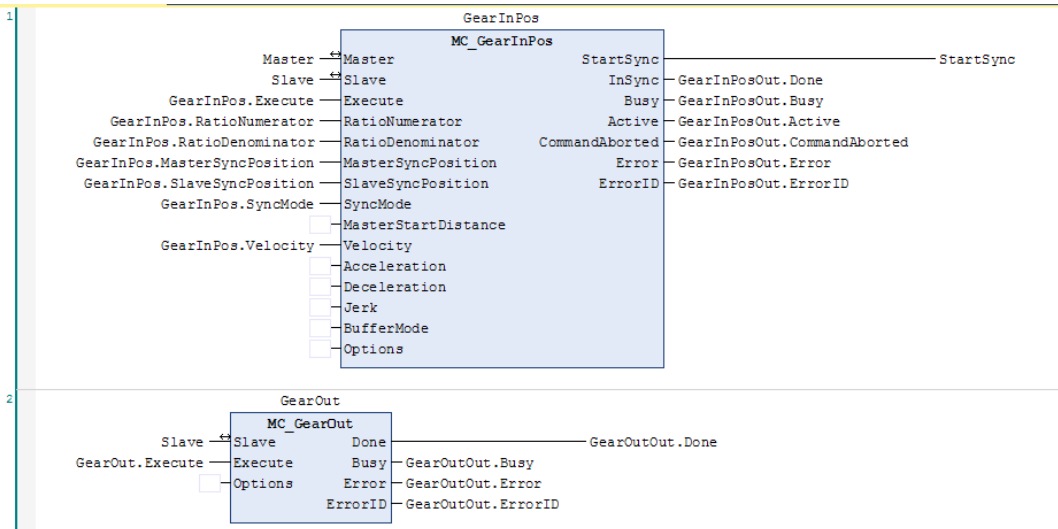
(7) 主轴匀速运动后，从轴执行飞锯耦合动作，在主程序中编辑位置模式下的飞锯耦合功能块，飞锯的解耦可以使用 MC\_GearOut 来实现

```

29      (*FlyingSaw Part*)
30      GearInPos          : MC_GearInPos;
31      GearInPosOut       : ST_McOutputs;
32      GearOut            : MC_GearOut;
33      GearOutOut         : ST_McOutputs;
34      StartSync          : BOOL;

```

创建 ACTION 来实现这一功能



(8) 在主程序区编写示例程序，完成一次飞锯耦合

(9) 更新轴状态

```

1      (*update all axes status*)
2      Master();
3      Slave();

```

调用 ACTION

```

88      ACTPOWER();
89      ACTFLYINGSAW();
90      ACTMOVE();
91      ACTRESET();

```

使用 CASE 语句完成整个流程，首先对所有功能块作初始化

```

5   F_INIT:
6   (*reset all funtion blocks*)
7   ResetMaster.Execute := FALSE;
8   ResetSlave.Execute := FALSE;
9   MoveMaster.Execute := FALSE;
10  MoveSlave.Execute := FALSE;
11  GearInPos.Execute := FALSE;
12  GearOut.Execute := FALSE;
13  state:=F_ENABLE;

```

接下去对轴进行使能，并且使用 IF 语句判断功能块是否成功执行

```

14  F_ENABLE:
15  PowerEnable:=TRUE;
16  IF PowerMaster.Status AND PowerSlave.Status THEN
17      state:=F_PREPARE1;
18  ELSIF PowerMaster.Error OR PowerSlave.Error THEN
19      state:=F_ERROR;
20  END_IF

```

需要注意的是功能块执行报错或者轴出现报错，处理方式都是跳转到 F\_ERROR 中，对轴进行解耦、复位或者对功能块进行初始化，分别通过 F\_ERROR 和 F\_RESET 来实现

```

70  F_ERROR:
71  GearOut.Execute:= Slave.Status.Coupled; (* gear out if done already *)
72  IF Master.Status.Error OR Slave.Status.Error THEN
73      state := F_RESET; (* axis error requires reset *)
74  ELSE
75      state := F_INIT; (* function block errors don't need a reset *)
76  END_IF
77
78  F_RESET :
79  ResetMaster.Execute := TRUE;
80  ResetSlave.Execute := TRUE;
81  IF ResetMaster.Done AND ResetSlave.Done THEN
82      state := F_INIT;
83  ELSIF ResetMaster.Error OR ResetSlave.Error THEN
84      state := F_INIT; (* can't do anything here *)
85  END_IF

```

使能完成后，启动主轴

```

21  F_PREPARE1:
22  (*Start Master Axis*)
23  MoveMaster.Velocity:=Master_V;
24  MoveMaster.Execute:=TRUE;
25  IF MoveMaster.InVelocity THEN
26      State:=F_PREPARE2;
27  ELSIF MoveMaster.CommandAborted OR MoveMaster.Error THEN
28      State:=F_ERROR;
29  END_IF

```

主轴匀速运动，从轴开始计算耦合时的各个参数，并开启相关参数校验

```

30 F_PREPARE2:
31 (*calculate synchronization parameters*)
32 GearOut.Execute:=FALSE;
33 GearInPos.Execute:=FALSE;
34 GearInPos.Velocity:=SlaveMax_V;
35 GearInPos.MasterSyncPosition:=MasterSync_P;
36 GearInPos.SlaveSyncPosition :=SlaveSync_P;
37 (* activate checks *)
38 GearInPos.SyncMode.GearInSync_OpMask_IgnoreMasterAcc := TRUE;
39 GearInPos.SyncMode.GearInSync_OpMask_PreferConstVelo := PREFERCONSTANTVELOCITY;
40 IF ENABLECHECKS THEN
41 GearInPos.SyncMode.GearInSync_CheckMask_MaxPos := TRUE;
42 GearInPos.SyncMode.GearInSync_CheckMask_MinPos := TRUE;
43 GearInPos.SyncMode.GearInSync_CheckMask_MaxVelo := TRUE;
44 GearInPos.SyncMode.GearInSync_CheckMask_MaxAcc := FALSE;
45 GearInPos.SyncMode.GearInSync_CheckMask_MaxDec := FALSE;
46 ; END_IF

```

参数计算完成后，触发飞锯耦合

```

48 F_COUPLE:
49 (*wait for master and slave to be synchronized*)
50 GearInPos.Execute:=TRUE;
51 IF GearInPos.InSync THEN
52 state:=F_SYNCHRONIZED;
53 ELSIF GearInPos.CommandAborted OR GearInPos.Error THEN
54 state:=F_ERROR;
55 END_IF

```

耦合完成后进入同步运行模式，此处以延时来表示同步运行阶段

```

56 F_SYNCHRONIZED:
57 (*master and slave in synchronized motion*)
58 Ton_SyncTime.IN:=TRUE;
59 Ton_SyncTime.PT:=SyncTime;
60 IF Ton_SyncTime.Q THEN
61 state:=F_DECOUPLE;
62 END_IF

```

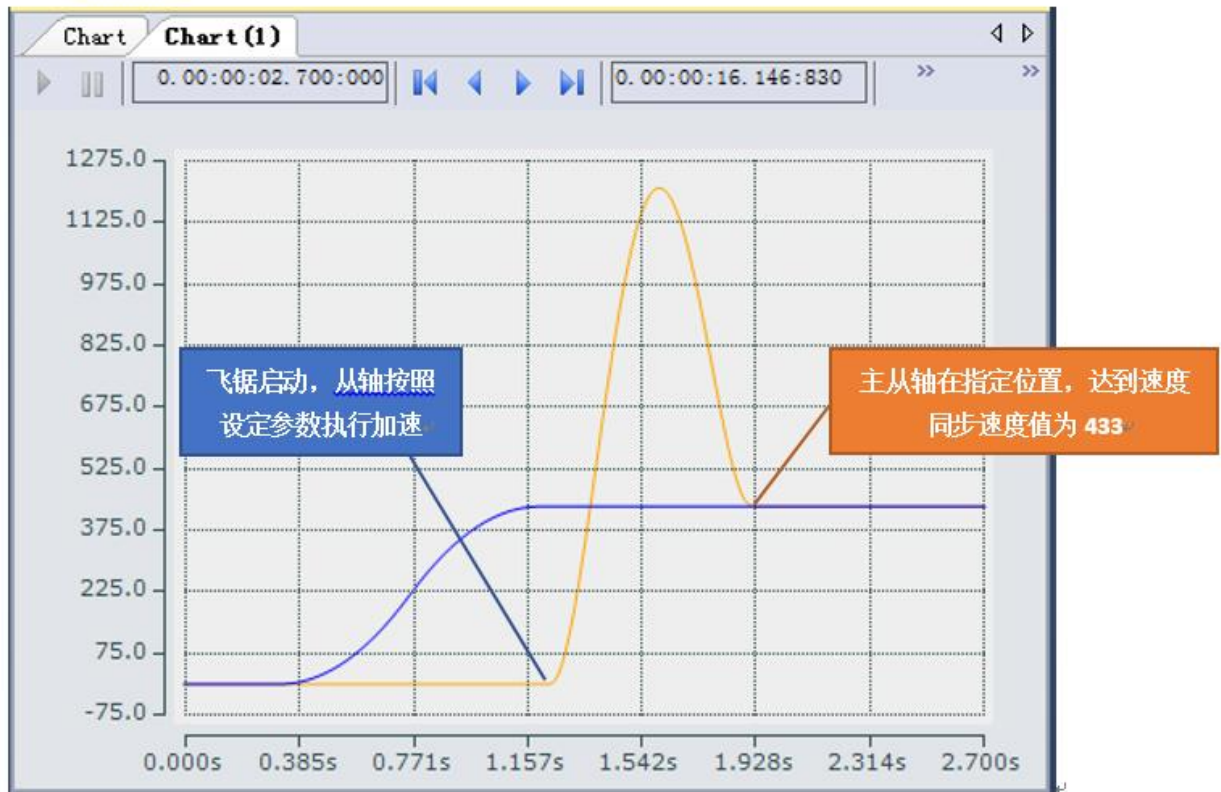
完成同步运行阶段后，主从轴解耦，从轴会维持解耦后的速度继续运动，整个飞锯耦合就完成了

```

63 F_DECOUPLE:
64 (*gear out-slave will keep its velocity *)
65 GearOut.Execute:= TRUE;
66 IF GearOut.Error THEN
67 GearOut.Execute:= TRUE;
68 state := F_ERROR;
69 END_IF

```

(10) 在 TwinCAT3 Measurement 中可以采集到主从轴在整个过程中的速度和位置变化曲线



(11) 该例子程序见 Motion 部分的“TC3-FlyingSaw”

## 八、FIFO 功能

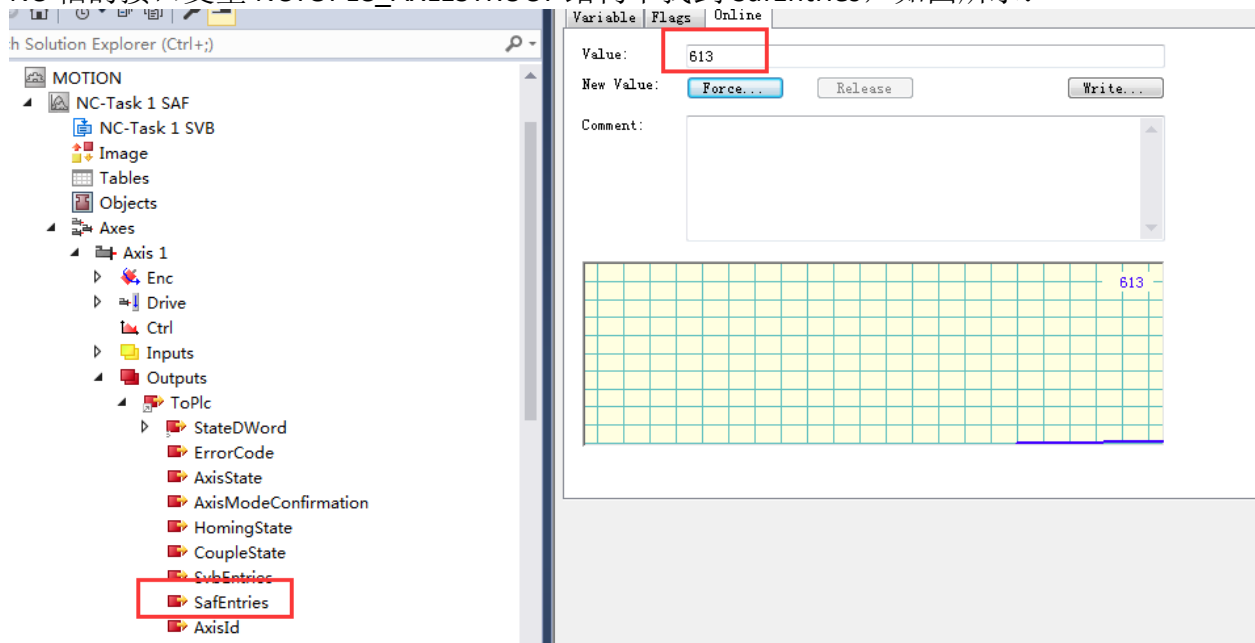
### 1. FIFO 功能介绍

FIFO 是 First Input First Output 的缩写。TwinCAT NC FIFO 是 TwinCAT NC 中的一个堆栈区。堆栈区中存放的是一个 n 维数组，数组中的值就是 n 个轴的位置序列。这些位置序列以先进先出的方式，依次作为设定位置发送给各个 NC 轴。

与单轴的 External set value generation 功能相比，这个功能类似于外部位置发生器（ExtSetpointGenerator），都是由用户自定义位置序列发送给 NC 轴作为设定位置，代替 NC 本身的位置发生器。区别有两点，一是 FIFO 功能允许同时给最多 16 个轴发送位置而 ExtSetpointGenerator 只能输出给一个轴；二是 FIFO 功能的位置序列允许自定义完成相邻两行位置之间的时间间隔，实际上就是 NC 会在相邻两行之间以 NC 周期插值。比如第 1 个点位置是 0.0，第 2 个点位置是 1.0，如果时间间隔为 10ms，而 NC 周期为 2ms，则 NC 会将 10ms 分为 5 段，每 2ms 发送一个设定位置，依次为 0.2, 0.4, 0.6, 0.8, 1.0，保证在第 10ms 的时候，位置达到 1.0。Fifo 可使运动更加平稳。

如果 FIFO 组中的轴数可以自定义，默认值为 1。TwinCAT NC FIFO 类似描点式（Fixed Table）电子凸轮表，但是 Fifo 消耗资源少，灵活性较差，不支持关键点的方式，也不能在线修改位置点。并且 Fifo 组内没有主从轴之分，不能根据主轴的速度变化调节从轴速度。FIFO 运动不能反转，从堆栈中完成的位置序列不再保留，如果要重复动作，只能重新装载数据。

FIFO 堆栈区数据的大小可以自定义，默认值为 1000 行。堆栈数据的 input，是从 PLC 中经功能块 FiFoWrite 写入的。而堆栈数据的 output，是从 PLC 中经功能块 FiFoStart 流出的。写入数据一次可以写入多行，流出数据就只能按照定义好的时间间隔。比如间隔为 10ms，则 1 秒种流出 100 行数据。至于堆栈中还剩作多少行数据，可以从 FIFO 组内任意 NC 轴的接口变量 NCTOPLC\_AXLESTRUCT 结构中找到 SafEntries，如图所示：

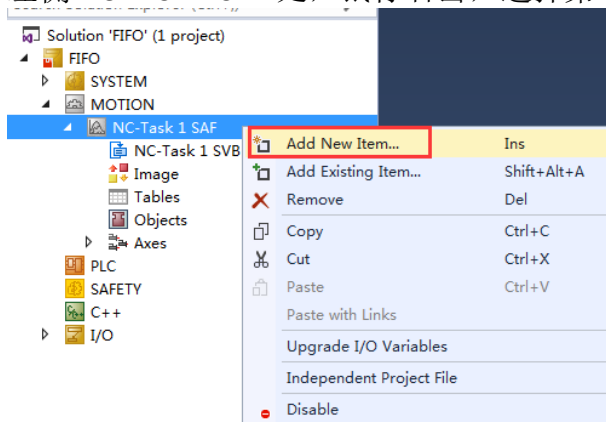


TwinCAT NC 中允许创建多个 FIFO 组，每个组的 ID 号是唯一的。在 PLC 程序中 FIFO 组

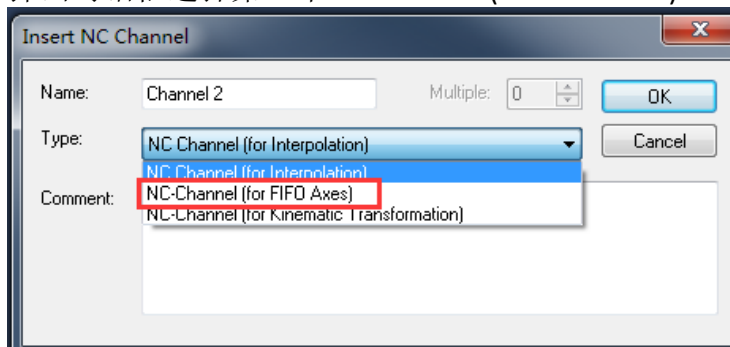
的操作，都是通过 ID 号识别的。

## 2. FIFO 功能的创建

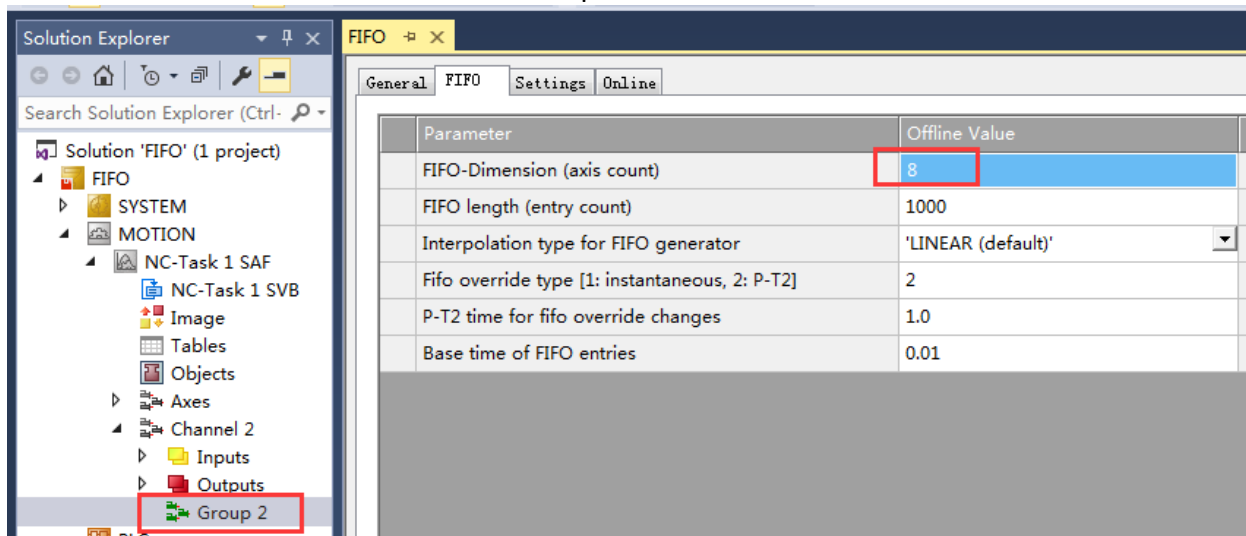
(1) 左侧 NC-TASK 1 SAF 处，鼠标右击，选择第一个 Add new item，如下图：



(2) 弹出对话框选择第二个 NC-Channel(for FIFO Axes)



(3) FIFO-Dimension 根据控制器的性能不同，一组 FIFO 组可以最多控制的轴数，有 8 个或者 16 个（设置该参数，在 Channel2-Group2-FIFO）



其中：

Fifo Length: Fifo 位置表的 Buffer 容量。以图中所示为例，Base time 为 0.02s，Fifo Length 为 1000，则 Buffer 中的数据能维持运行 20 秒。缓存越小，则数据从 PLC 传送



到 NC Fifo 就越频繁。Buffer 越大，消耗计算机内存越多。

**Fifo Override Type:** 通过 Override，可以调节运动速度。Override 的切换方式可以选择阶跃型（Instantaneous override）、平滑型（PT-2 override）或者三次型（CUBIC SPLINE(6P)）。

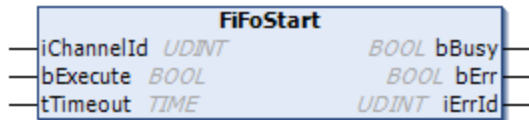
**P-T2 time for override changes:** 当 Fifo Override Type 选择平滑型（PT-2 override）切换方式时，在此处设置切换时间。时间越长，切换越平滑。

**Base Time of Fifo entries:** 连续两个位置点之间的时间间隔，Base Time 必须是 NC 周期的整数倍，NC 会在连续两个位置点之间进行插补，并自动计算运动速度。

- (4) 注意点：外部给定位置值列表，可以从文件中读取，也可以在 PLC 程序中在线生成。当 Fifo 表驱动实际硬件时，由于速度不是在程序中指定，而是由文件中位置表决定，所以建议先在虚轴上运行，观察各轴运行 Fifo 表的最大速度，然后确认伺服驱动器和电机确实能够支持该速度。这一过程，又称为曲线校验。

### 3. FIFO 功能包含的功能块（基于）

- (1) FiFoStart 启动 Fifo 组内各轴按照此前接收并存储在 Buffer 中的位置表运动



iChannelId: FIFO channel 的 ID 号。

bExecute: 上升沿触发本功能块动作。

tTimeout: ADS timeout (约 1 s)。

bBusy: bExecute 的上升沿 bBusy 置 True, 完成后为 False。

bErr: 指令执行过程中出错则置为 True。

bErrId: 错误代码(ADS 或 NC 错误代码)。

- (2) FiFoGroupIntegrate 把一个独立的 PTP 轴集成到一个 Fifo 组中。变量 iGroupPosition 指定了它在 Fifo 组中的序号



iChannelId: FIFO channel 的 ID 号。

iAxisId: 轴的 ID。

iGroupPosition: 将轴集成到 Fifo 组后，它在该组中的序号，首序号为 1)。

bExecute: 上升沿触发本功能块动作。

tTimeout: ADS timeout (约 1 s)。

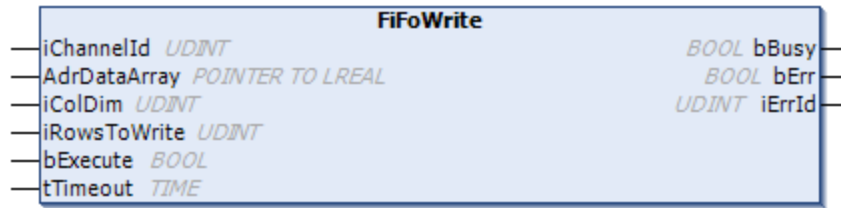
bBusy: bExecute 的上升沿 bBusy 置 True, 完成后为 False。

bErr: 指令执行过程中出错则置为 True。

bErrId: 错误代码(ADS 或 NC 错误代码)。

- (3) FiFoWrite 把指定数组中的数据，写到 TwinCAT NC Fifo 组的位置缓存表（Buffer）





iChannelId: FIFO channel的ID号。

AdrDataArray: 位置表数组的指针。该数组应该是一个二维数组。“列”表示轴的数量，“行”表示每个轴的位置点数量。

iRowsToWrite: 写入行数. 必须小于等于位置表数组的行数。

bExecute: 上升沿触发本功能块动作。

tTimeout: ADS timeout (约1 s)。

bBusy: bExecute的上升沿Busy置True, 完成后为False。

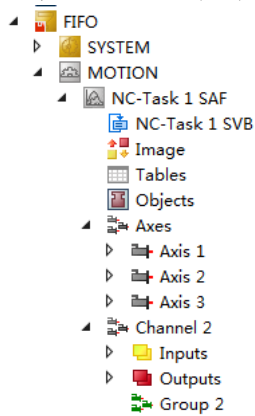
bErr: 指令执行过程中出错则置为True。

bErrId: 错误代码 ADS 或 NC 错误代码。

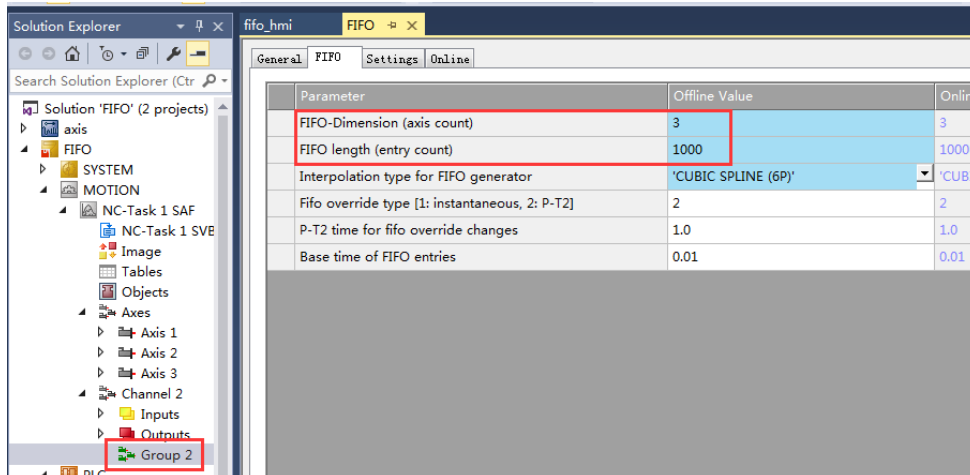
#### 4. 案例程序

本程序实现三轴 FIFO 功能，堆栈中的数据量为每个轴 1000 个数据，其中数据是通过 PLC 中计算出的数据写入 XML 文件，并再从 XML 文件中读出，简而言之，为读者以后用 XML 的格式提供思路。

(1) 首先创建三根虚轴，创建 FIFO 通道，如下图所示：

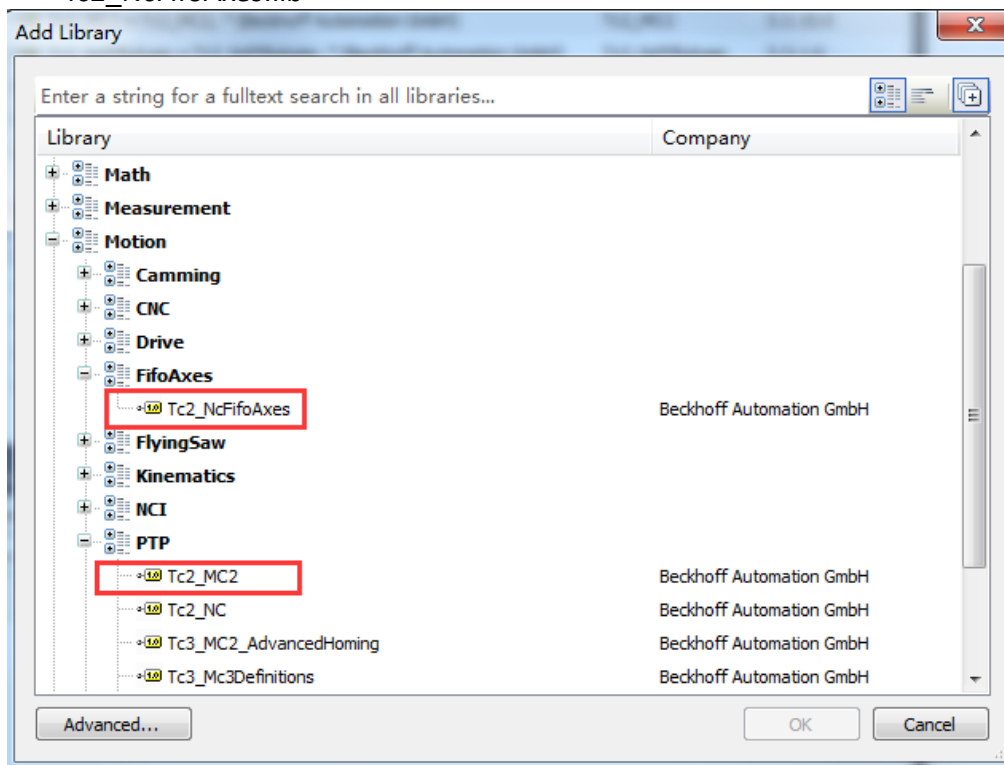


设置 FIFO 通道轴数为 3，堆栈数据量为：1000，设置如下：



完成以上设置，激活配置。

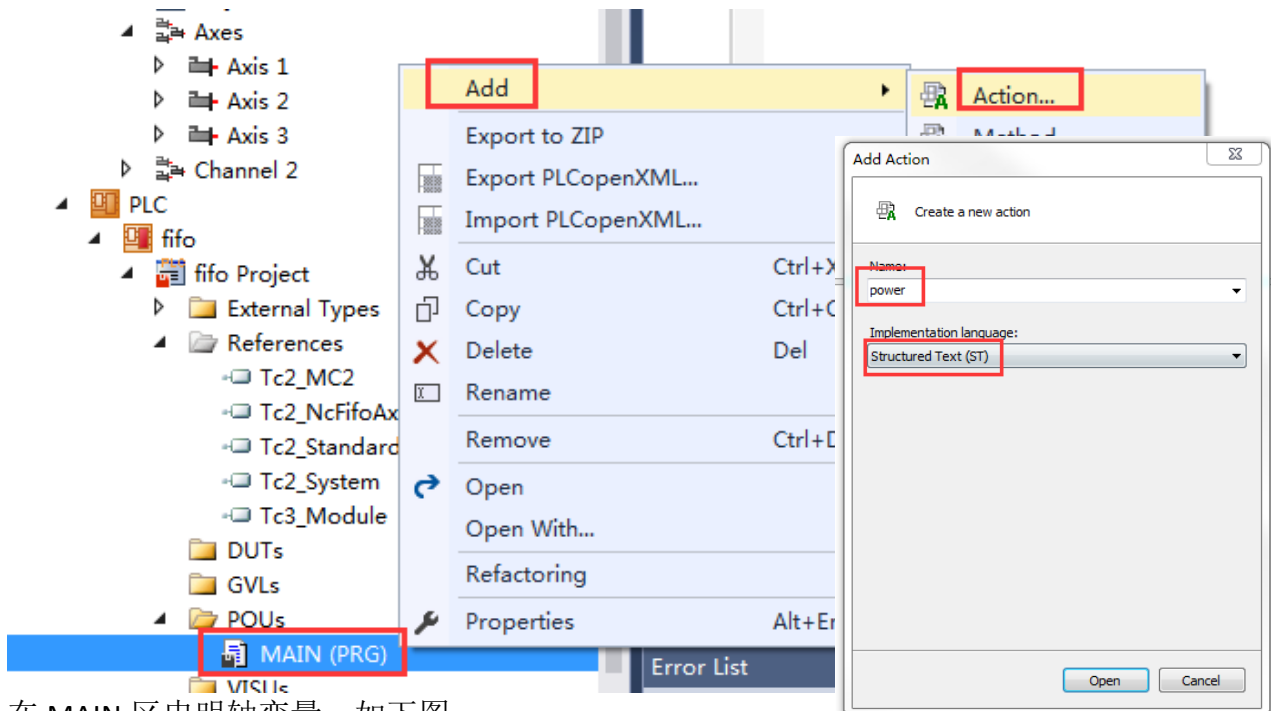
- (2) 在 PLC 下方创建项目，并命名为 `fifo`。同时在调用两个库：`Tc2_MC2.lib` 和 `Tc2_NcFifoAxes.lib`



- (3) 根据例程的功能，用五个 Action 来实现，分别是：对轴使能；编辑 FIFO 的位置表；XML 文件读写功能；FIFO 中集合/解散通道，设置通道速率，向通道写入位置表，开启/停止 FIFO。

① 对轴使能：

创建一个 Action，命名为 `power`，选用 ST 语言；



在 MAIN 区申明轴变量，如下图：

```

MAIN*  ▸ ×
1  PROGRAM MAIN
2  VAR
3      axis1,axis2,axis3:axis_ref;
4
5

```

接下来，在 power 这个 Action 对轴进行使能，程序如下：

```

MAIN.power  ▸ ×  MAIN*
1  axis1.PlcToNc.ControlDWord:=7;
2  axis2.PlcToNc.ControlDWord:=7;
3  axis3.PlcToNc.ControlDWord:=7;
4  axis1.PlcToNc.Override:=1000000;
5  axis2.PlcToNc.Override:=1000000;
6  axis3.PlcToNc.Override:=1000000;

```

## ② 编辑 FIFO 的位置表：

模拟程序，所以采用以下公式来设计位置表：

$$1 \text{ 轴: } y(x) = 10 \sin(0.02x) + 2 \cos(0.04x) + 3 \cos(0.02x) - 5 \cos(0.04x)$$

$$2 \text{ 轴: } y(x) = 10 \sin(0.01x) + 2 \cos(0.002x) + 3 \cos(0.001x) - 5 \cos(0.02x)$$

$$3 \text{ 轴: } y(x) = 10 \sin(0.03x) + 2 \cos(0.06x) + 3 \cos(0.03x) - 5 \cos(0.06x)$$

( $x$ 为从 0 累加到 999)

根据以上公式，可为每个轴提供 1000 个位置数据，创建 Action，命名为：fif\_pos；首先先在 MAIN 区申明如下：

```

MAIN.fifo_pos  MAIN.power  MAIN*  X
1  PROGRAM MAIN
2  VAR
3      axis1,axis2,axis3:axis_ref;
4
5      t:INT;
6      Pos_arr1,Pos_arr2: ARRAY [0..999,1..3] OF LREAL;
7

```

其中 t 相当于 x，Pos\_arr1 用于储存计算得到的位置表数据；Pos\_arr2 先申明好，稍后用于 XML 读写功能中。

在 fifo\_pos 这个 Action 中如下编程：

```

MAIN.fifo_pos  MAIN.power  MAIN*
1  FOR t:=0 TO 999 BY 1 DO
2      Pos_arr1[t,1]:=10*SIN(0.02*t) + 2*COS(0.04*t) + 3*COS(0.02*t) - 5*COS(0.04*t);
3      Pos_arr1[t,2]:=10*SIN(0.01*t) + 2*COS(0.002*t) + 3*COS(0.001*t) - 5*COS(0.02*t);
4      Pos_arr1[t,3]:=10*SIN(0.03*t) + 2*COS(0.06*t) + 3*COS(0.03*t) - 5*COS(0.06*t);
5  END_FOR
6

```

### ③ XML 文件读写功能：

该功能将已经计算得到的位置表数据 Pos\_arr1 先写成 XML 文件，再将其读出放置到 Pos\_arr2 中进行使用。读者可根据实际项目中的情况来酌情使用，例如：第三方软件优化好的轨迹存成 XML 文件，可直接通过 FB\_XmlSrvRead 读取到 TwinCAT 软件中等。若不需要 XML 文件读写的话，可跳过本步骤，并在下一个 Action 中，选择导入 FIFO 的位置表数据时，选择 Pos\_arr1 即可。

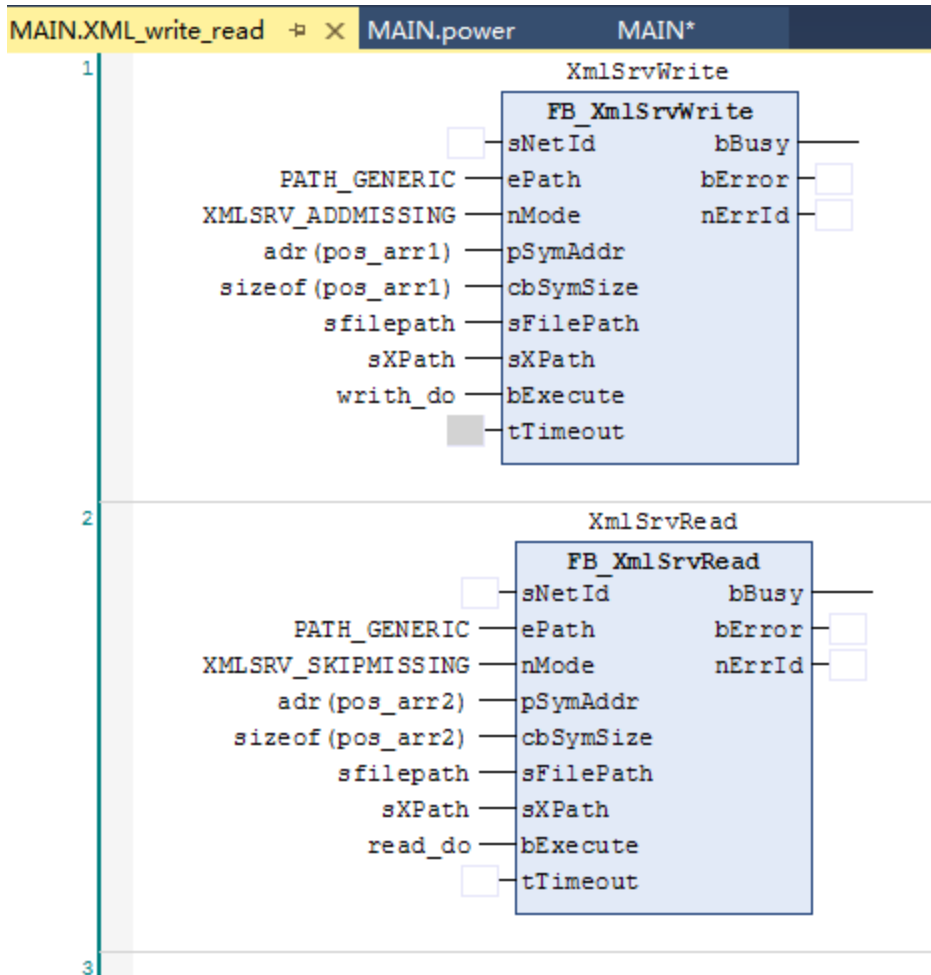
首先 MAIN 区定义如下，其中 FB\_XmlSrvWrite 功能块用于将程序中的 Pos\_arr1 的位置数据做成 XML 文件，命名为 Test3.xml，并写到路径：E:\Test3.xml 中；FB\_XmlSrvRead 功能块用来读取 XML 文件到程序中 Pos\_arr2 中：

```

MAIN.power  MAIN*  X
1  PROGRAM MAIN
2  VAR
3      axis1,axis2,axis3:axis_ref;
4
5      t:INT;
6      Pos_arr1,Pos_arr2: ARRAY [0..999,1..3] OF LREAL;
7
8      XmlSrvWrite: FB_XmlSrvWrite;
9      sfilepath: T_MaxString:='E:\Test3.xml';
10     sXPath: T_MaxString:='/dataentry/MAIN.Pos_arr1';
11     writh_do: BOOL;
12     XmlSrvRead: FB_XmlSrvRead;
13     read do: BOOL;
14

```

在创建一个 Action，命名为 XML\_write\_read，编程如下：



④ FIFO 中集合/解散通道，设置通道速率，向通道写入位置表，开启/停止 FIFO：  
首先 MAIN 区定义如下图：

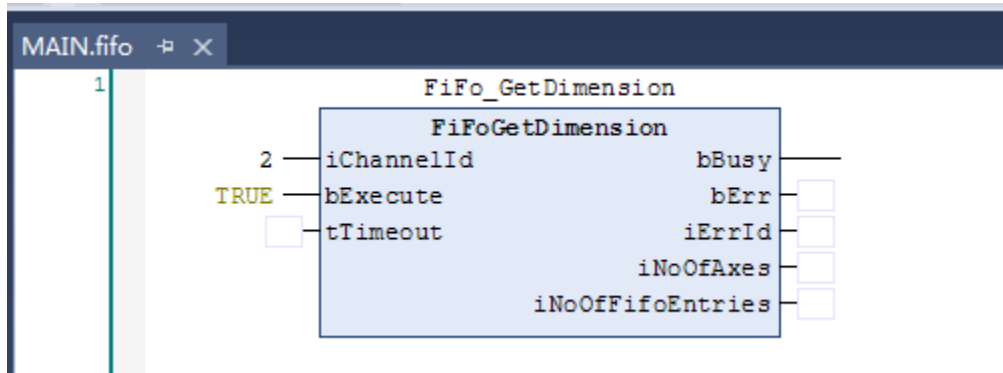
```

MAIN.XML_write_read  MAIN.power  MAIN*
14
15  FiFo_GetDimension: FiFoGetDimension;
16  fifo_display: STRING(255);
17
18  FiFo_GroupIntegrate1,FiFo_GroupIntegrate2,FiFo_GroupIntegrate3: FiFoGroupIntegrate;
19  FiFo_SetChannelOverride: FiFoSetChannelOverride;
20  FiFo_Groupout1,FiFo_Groupout2,FiFo_Groupout3,FiFo_Disintegrate,FiFo_Override_output:st_mcoutputs;
21  integrate_do: BOOL;
22  FiFo_GroupDisintegrate: FiFoGroupDisintegrate;
23
24  FiFo_Start: FiFoStart;
25  start_do: BOOL;
26  FiFo_Stop: FiFoStop;
27  stop_do: BOOL;
28  fifo_startoutput,fifo_stopoutput:st_mcoutputs;
29
30  FiFo_Write:FiFoWrite;
31  FIFO_write_do: BOOL;
32  FiFo_overWrite: FiFoOverwrite;
33  FIFO_overwrite_do: BOOL;
34  FIFO_write_output:st_mcoutputs;
35  END_VAR
36

```

接下来创建 Action 并命名为 fifo，在该 Action 中：

第一部分，是用于检查在 Motion 中设置与程序中所定义的轴数及位置表数据量是否一致，Action 中的定义如下，iNoOfAxes 将读出 FIFO 通道中的轴数，iNoOfFifoEntries 将读出单轴的数据量；



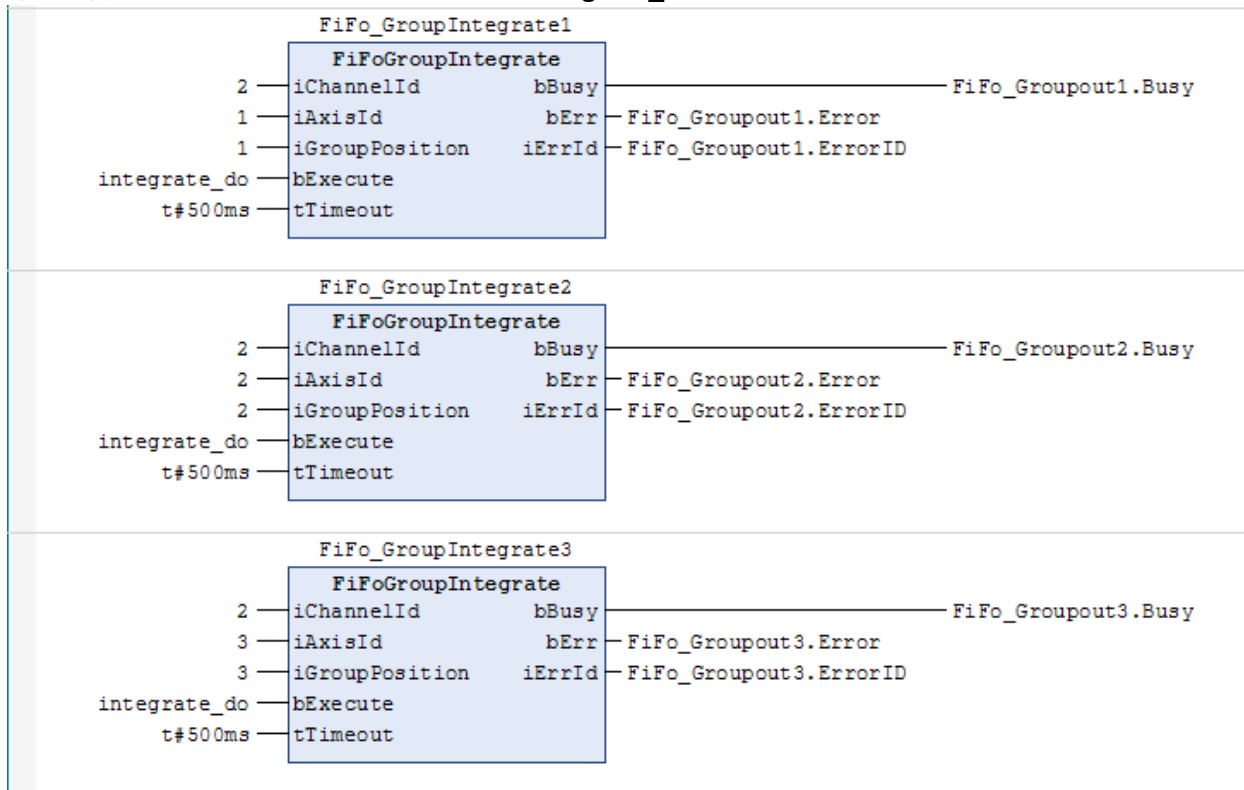
在 MAIN 区编程如下，检查是否满足参数设置，并将 fifo\_display 其做在 HMI 界面显示(本章节 HMI 界面不做详解，在最后简单介绍)。

```

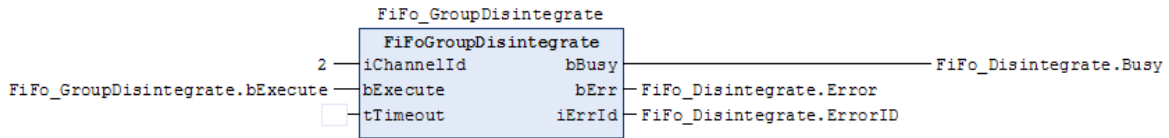
1 IF FiFo_GetDimension.iNoOfAxes=3 AND FiFo_GetDimension.iNoOfFifoEntries=1000 THEN
2   fifo_display:='Set Parameters Successfully,Please Continue';
3 ELSE
4   fifo_display:='Please,Check Parameters';
5 END_IF

```

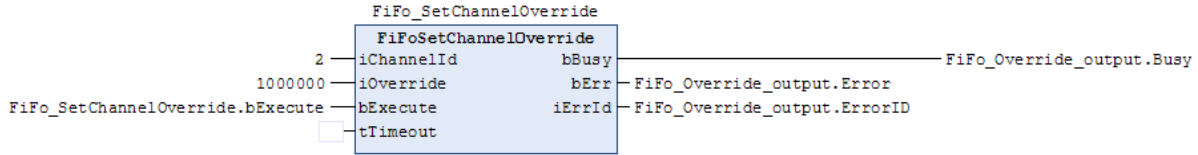
第二部分，在 Action 中调用如下，用 integrate\_do 变量控制三轴添加到 FIFO 的通道中。



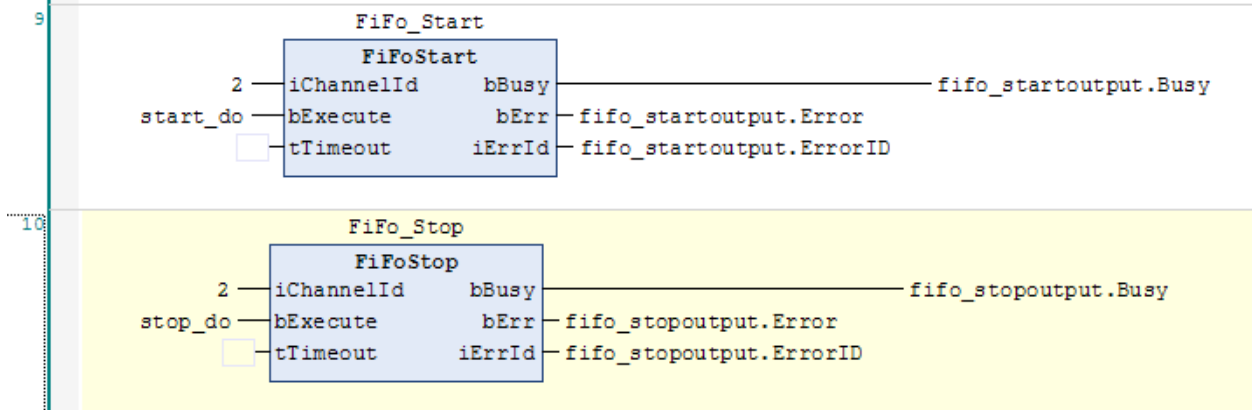
同时，如下功能块的调用，用 FiFo\_GroupDisintegrate.bExecute 来解散通道中的轴；



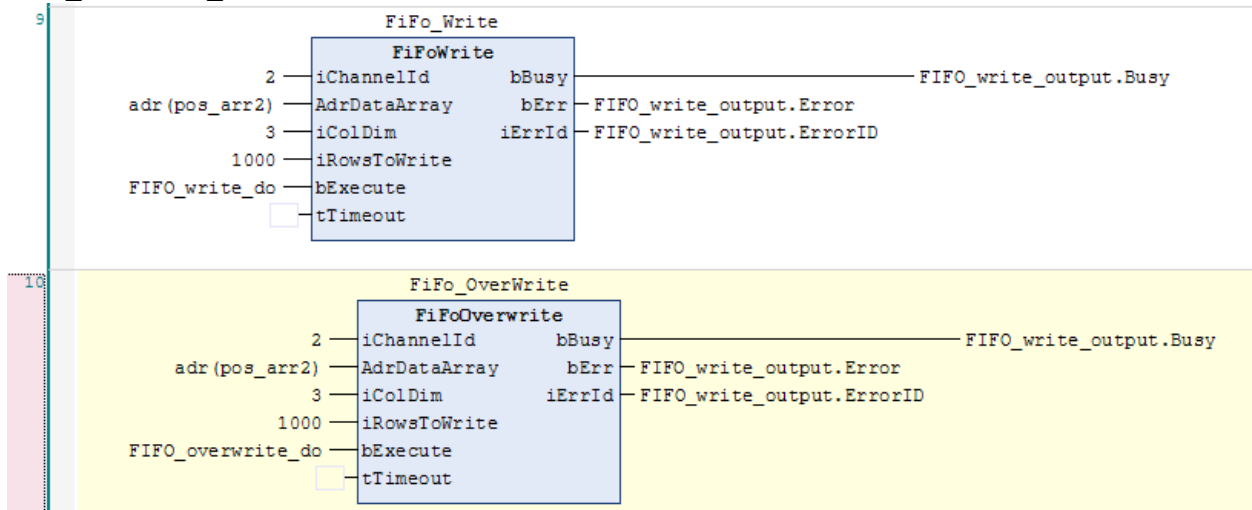
最后，设置通道倍率，具体调用如下，用来触发该功能块。



第三部分，用于启停 FIFO 功能，调用如下，启动采用 `start_do` 变量来触发，停止采用 `stop_do` 变量来触发。



第四部分，用于将位置数据写入到 FIFO 通道中，**FiFoWrite** 功能块用于写入空白的 FIFO 通道中，而 **FiFo\_OverWrite** 用于覆盖原有通道中的数据，分别用 `FIFO_write_do`、`FIFO_overwrite_do` 两个变量来触发。



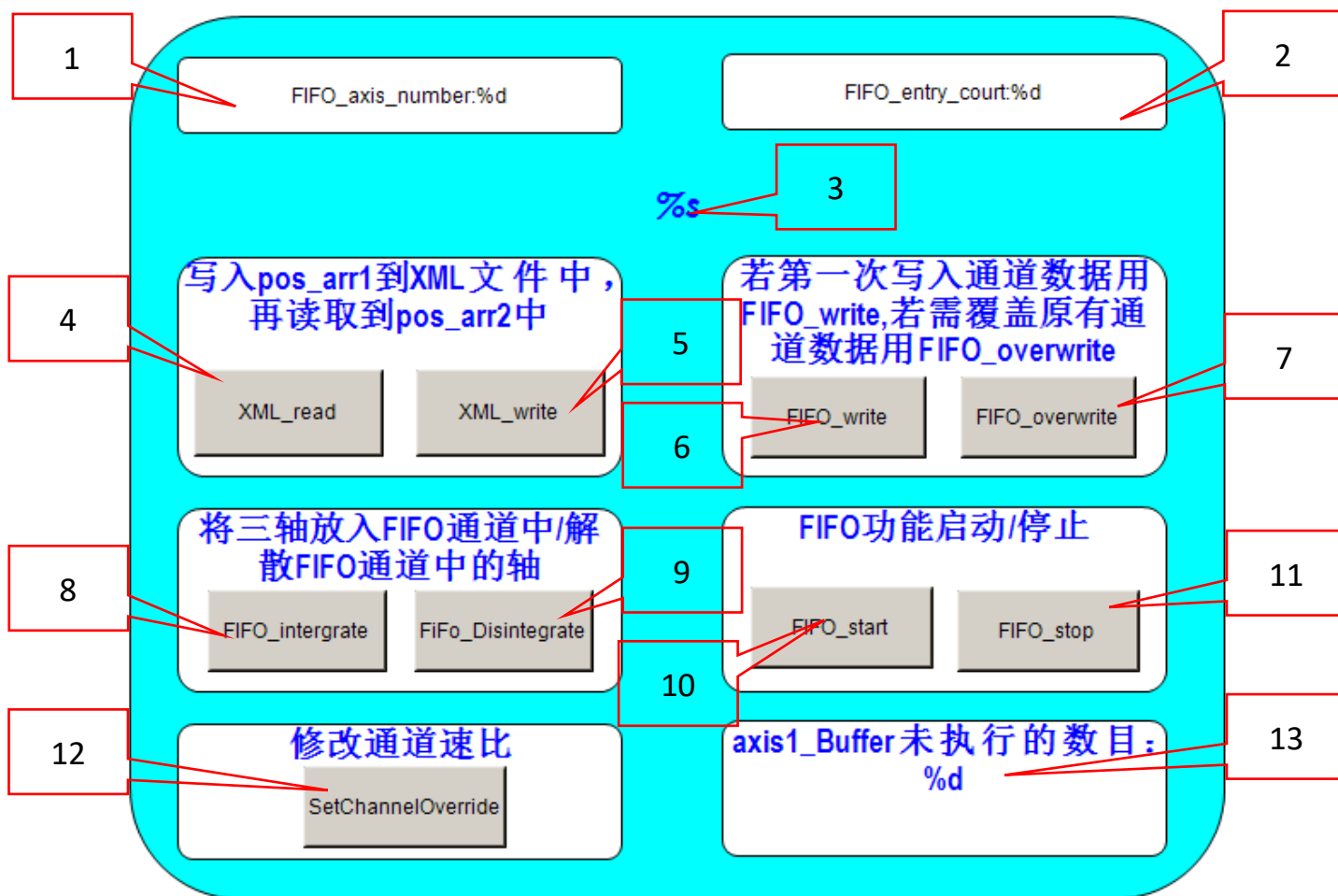
(4) 最后在 MAIN 程序区调用这四个 Action

```

1  IF FiFo_GetDimension.iNoOfAxes=3 AND FiFo_GetDimension.iNoOfFifoEntries=1000 THEN
2      fifo_display:='Set Parameters Successfully,Please Continue';
3  ELSE
4      fifo_display:='Please,Check Parameters';
5  END_IF
6
7  fifo();
8  fifo_pos();
9  power();
10 XML_write_read();

```

(5) 在此基础上，完善 HMI 界面，本例程中 HMI 界面，命名为 fifo\_hmi。

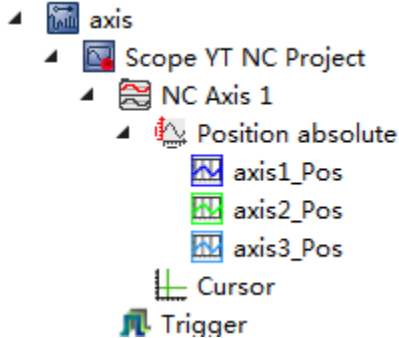


编号	所用变量	作用
1	MAIN.FiFo_GetDimension.iNoOfAxes	显示 FIFO 通道中所绑定的轴数
2	MAIN.FiFo_GetDimension.iNoOfFifoEntries	显示 FIFO 通道单轴的位置数据量
3	MAIN.fifo_display	显示是否 MOTION 和程序中设置的参数相统一
4	MAIN.read_do	触发 XML 文件读取到程序中
5	MAIN.writh_do	触发将程序中的数据写入到 XML 文件中
6	MAIN.FIFO_write_do	用于将位置数据写入到空白的 FIFO 通道位置表中



7	MAIN.FIFO_overwrite_do	用于覆盖 FIFO 通道中原有的位置表，替换成新的
8	MAIN.integrate_do	用于将三轴放置到 FIFO 通道中
9	MAIN.FiFo_GroupDisintegrate.bExecute	用于解散 FIFO 中的轴
10	MAIN.start_do	用于开启 FIFO 功能
11	MAIN.stop_do	用于停止 FIFO 功能
12	MAIN.FiFo_SetChannelOverride.bExecute	用于将 FIFO 通道的速率设为 100%
13	MAIN.axis1.NcToPlc.SafEntries	用于显示轴 1 未执行的位置数据量

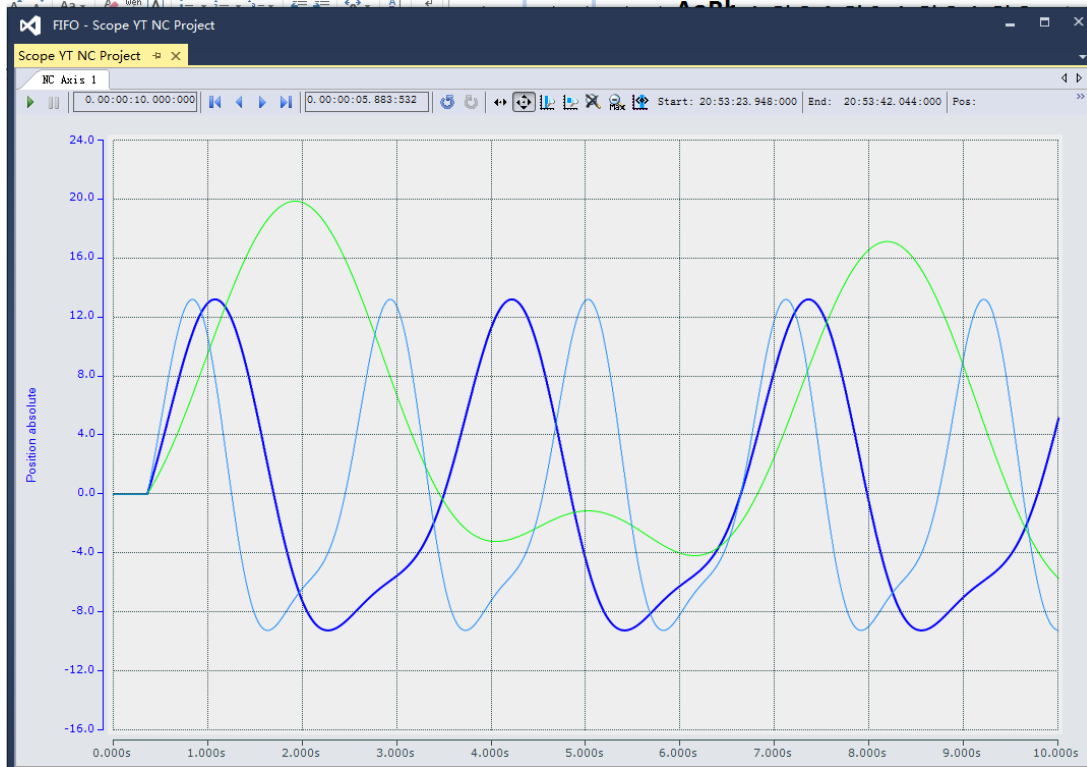
(6) 创建 Scope 用于监测三个轴的位置曲线，



(7) 当运行程序后，以此按下 HMI 界面中的按钮，顺序为：5-4-6-8-12-10；若参数设置正确，且在执行 FIFO 功能时，可看见 HMI 界面如下显示：



Scope 界面可监测到如下：



5. 该例子程序见 Motion 部分的“TC3-Fifo”

## 九、PLC 程序修改 NC 轴的参数设置

如果需要 PLC 程序动态地修改 NC 轴的参数，而不是驱动器参数，有两种方法：用专门的 MC 功能块，或者使用 ADS 通讯。

### 1. 读写 NC 轴参数的功能块

在 Tc2\_MC2.lib 库中提供了多个用于读写 NC 轴参数的功能块，以下介绍常用的功能块。

#### (1) 读写 BOOL 类型的功能块

##### ① 读 BOOL 型的功能块

### MC\_ReadBoolParameter



该模块在 enable 置为 true 时，读取 NC 轴的参数，具体读哪个参数取决于 ParameterNumber（例如读取 EnableLimitPos，那么 ParameterNumber 为 4），ReadMode 决定是读一次还是周期循环读，最终读取结果以 Value 输出结果。

其中可读的变量是 MC\_AxisParameter 中的 bool 型变量，以下是 MC\_AxisParameter 的部分变量：

## MC\_AxisParameter

The *MC\_AxisParameter* data type is used in conjunction with function blocks for reading and writing of axis p

```

TYPE MC_AxisParameter : (
(* PLCopen specific parameters *) (* Index-Group 0x4000 +ID*)
CommandedPosition :=1, (*lreal *) (* taken from NcToPlc *)
SWLimitPos, (*lreal *) (* IndexOffset= 16#0001_000E *)
SWLimitNeg, (*lreal *) (* IndexOffset= 16#0001_000D *)
EnableLimitPos, (*bool *) (* IndexOffset= 16#0001_000C *)
EnableLimitNeg, (*bool *) (* IndexOffset= 16#0001_000B *)
EnablePosLagMonitoring, (*bool *) (* IndexOffset= 16#0002_0010 *)
MaxPositionLag, (*lreal *) (* IndexOffset= 16#0002_0012 *)
MaxVelocitySystem, (*lreal *) (* IndexOffset= 16#0000_0027 *)
MaxVelocityAppl, (*lreal *) (* IndexOffset= 16#0000_0027 *)
ActualVelocity, (*lreal *) (* taken from NcToPlc *)
CommandedVelocity, (*lreal *) (* taken from NcToPlc *)
MaxAccelerationSystem, (*lreal *) (* IndexOffset= 16#0000_0101 *)
MaxAccelerationAppl, (*lreal *) (* IndexOffset= 16#0000_0101 *)
MaxDecelerationSystem, (*lreal *) (* IndexOffset= 16#0000_0102 *)
MaxDecelerationAppl, (*lreal *) (* IndexOffset= 16#0000_0102 *)
MaxJerkSystem, (*lreal *) (* IndexOffset= 16#0000_0103 *)
MaxJerkAppl, (*lreal *) (* IndexOffset= 16#0000_0103 *)

(* Beckhoff specific parameters *) (* Index-Group 0x4000 +ID*)
AxisId := 1000, (*lreal *) (* IndexOffset= 16#0000_0001 *)
AxisVeloManSlow, (*lreal *) (* IndexOffset= 16#0000_0008 *)
AxisVeloManFast, (*lreal *) (* IndexOffset= 16#0000_0009 *)
AxisVeloMax, (*lreal *) (* IndexOffset= 16#0000_0027 *)
AxisAcc, (*lreal *) (* IndexOffset= 16#0000_0101 *)
AxisDec, (*lreal *) (* IndexOffset= 16#0000_0102 *)
AxisJerk, (*lreal *) (* IndexOffset= 16#0000_0103 *)
MaxJerk, (*lreal *) (* IndexOffset= 16#0000_0103 *)
AxisMaxVelocity, (*lreal *) (* IndexOffset= 16#0000_0027 *)
AxisRapidTraverseVelocity, (* lreal *) (* IndexOffset= 16#0000_000A *)
AxisManualVelocityFast, (*lreal *) (* IndexOffset= 16#0000_0009 *)
AxisManualVelocitySlow, (*lreal *) (* IndexOffset= 16#0000_0008 *)
AxisCalibrationVelocityForward, (* lreal *) (* IndexOffset= 16#0000_0006 *)
AxisCalibrationVelocityBackward, (* lreal *) (* IndexOffset= 16#0000_0007 *)
AxisJogIncrementForward, (*lreal *) (* IndexOffset= 16#0000_0018 *)
AxisJogIncrementBackward, (*lreal *) (* IndexOffset= 16#0000_0019 *)
AxisEnMinSoftPosLimit, (*bool *) (* IndexOffset= 16#0001_000B *)
AxisMinSoftPosLimit, (*lreal *) (* IndexOffset= 16#0001_000D *)
AxisEnMaxSoftPosLimit, (*bool *) (* IndexOffset= 16#0001_000C *)
AxisMaxSoftPosLimit, (*lreal *) (* IndexOffset= 16#0001_000E *)
AxisEnPositionLagMonitoring, (* bool *) (*IndexOffset= 16#0002_0010 *)
AxisMaxPosLagValue, (*lreal *) (* IndexOffset= 16#0002_0012 *)
AxisMaxPosLagFilterTime, (*lreal *) (* IndexOffset= 16#0002_0013 *)
AxisEnPositionRangeMonitoring, (*bool *) (* IndexOffset= 16#0000_000F *)
AxisPositionRangeWindow, (*lreal *) (* IndexOffset= 16#0000_0010 *)
AxisEnTargetPositionMonitoring, (* bool *) (* IndexOffset= 16#0000_0015 *)
AxisTargetPositionWindow, (*lreal *) (* IndexOffset= 16#0000_0016 *)
AxisTargetPositionMonitoringTime. (* lreal *) (*IndexOffset= 16#0000_0017 *)

```

### ② 写 BOOL 型的功能块

## MC\_WriteBoolParameter



该模块在 Execute 置为 true 时，写入 NC 轴的参数，具体读哪个参数取决于 ParameterNumber。

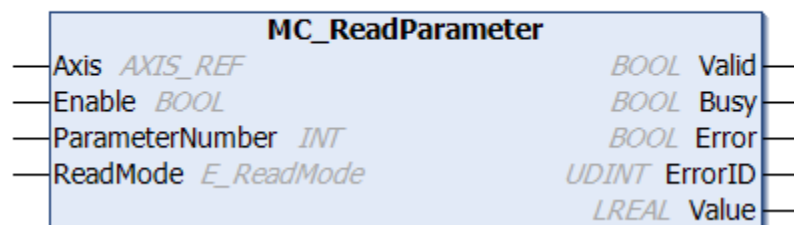
例：若需限制轴只能在位置数据为负值内移动，可以通过对 EnableLimitPos 这个变量置 true；

```
1 PROGRAM MAIN
2 VAR
3   axis:axis_ref;
4   power:mc power;
5   bool_writeparameter:mc_writeboolparameter;
6
7   bool_write_do: BOOL;
8   temp: BOOL:=TRUE;
9   number: INT:=4;
10
11
12
13
14 bool_writeparameter(
15   Axis:=axis ,
16   Execute:=bool_write_do ,
17   ParameterNumber:=number ,
18   Value:= temp,
19   Done=> ,
20   Busy=> ,
21   Error=> ,
22   ErrorID=> );
23
```

### (2) 读写 Lreal 型的功能块

#### ① 读 Lreal 型的功能块

## MC\_ReadParameter



该模块在 enable 置为 true 时，读取 NC 轴的参数，具体读哪个参数取决于 ParameterNumber（例如读取 NcSafCycleTime，那么 ParameterNumber 为 4000），ReadMode 决定是读一次还是周期循环读，最终读取结果以 Value 输出结果。

#### ② 写 Lreal 型的功能块

## MC\_WriteParameter



该模块在 Execute 置为 true 时，写入 NC 轴的参数，具体读哪个参数取决于 ParameterNumber。

例：需要通过功能块读取 NC-SAF 周期时间，ParameterNumber 取 NcSafCycleTime

```

TwinCAT Project26  MAIN*
1 PROGRAM MAIN
2 VAR
3     axis:axis_ref;
4     power:mc_power;
5
6     bool_writeparameter:mc_writeboolparameter;
7     bool_write_do: BOOL;
8     write_temp: BOOL:=TRUE;
9     number: INT:=4;
10
11     lreal_readparameter:mc_readparameter;
12     read_do: BOOL;
13     read_temp: INT :=4000;
14     SAF_TIME: LREAL;
15 END_VAR
16
23 lreal_readparameter(
24     Axis:=axis ,
25     Enable:=read_do ,
26     ParameterNumber:=read_temp ,
27     ReadMode:=2 ,
28     Valid=> ,
29     Busy=> ,
30     Error=> ,
31     ErrorID=> ,
32     Value=>SAF_TIME );
33
34

```

### (3) 其他功能块

- MC\_ReadActualPosition 用来读取轴的实际位置
- MC\_ReadActualVelocity 用来读取轴的实际速度
- MC\_ReadStatus 用来读取轴当前状态
- MC\_ReadAxisError 用来读取轴的报错信息
- MC\_ReadParameterSet 用来读取轴的相关参数，具体参数可以查看
- ST\_AxisParameterSet
- MC\_ReadAxisComponents 用来读取轴组成部分的 ID，如 encoder、controller、drive
- MC\_WriteBoolParameterPersistent 和 MC\_WriteParameterPersistent 用于设置初始配置的 NC 参数，用该 FB 设置的 NC 参数不能用 MC\_ReadBoolParameter 和 MC\_ReadParameter 修改

## 2. 采用 ADS 的方式来读写 NC 参数

在 Tc2\_System.lib 标准库中有两个功能块 ADSREAD 和 ADSWRITE，可采用这两个来进行读写 NC 轴参数。

用 ADS 功能块读写 NC 轴参数时，NetID 可以为空白（""），表示与 PLC 程序同一台控制器。

Port 填“500”，指 TwinCAT NC 端口。

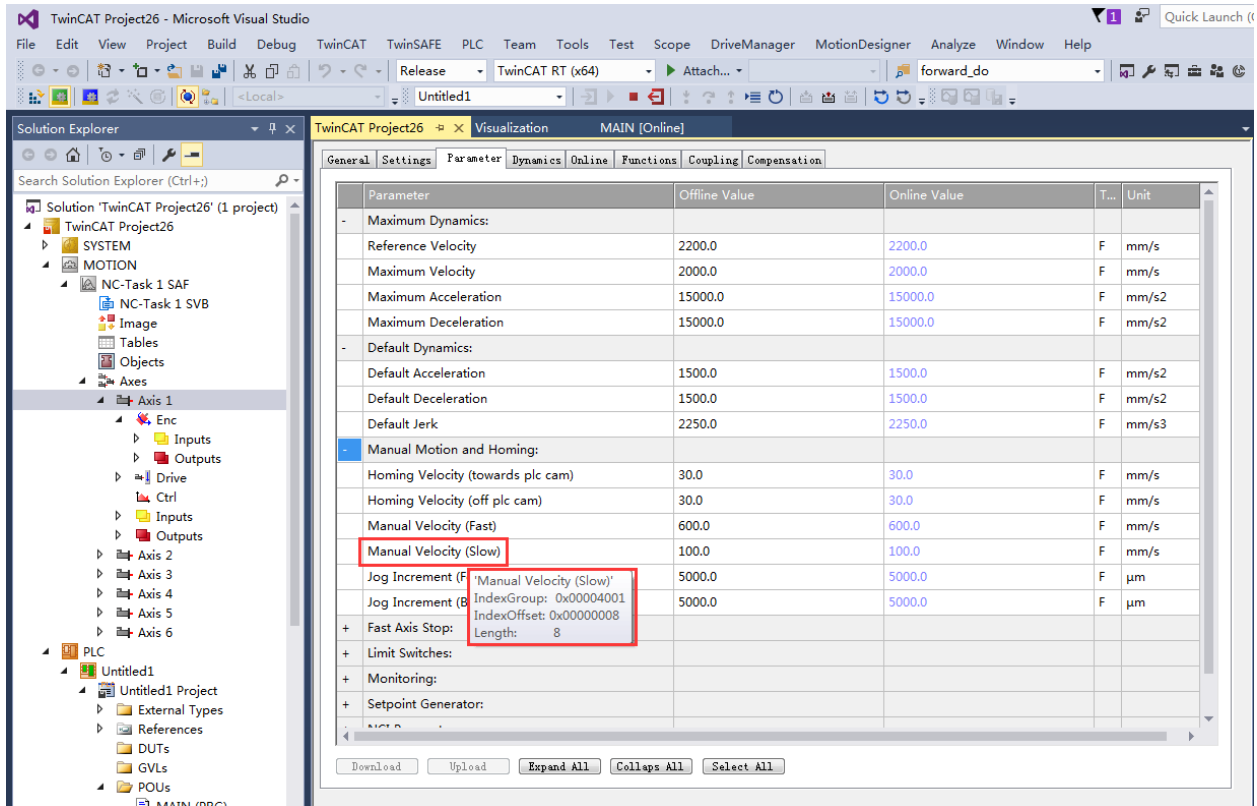
Index 和 Offset，则需要依据 NC 轴的 ID 号和要修改的参数而定。

如果要查询 NC 轴参数的 Index 和 Offset 说明，请打开 TC3 的 Help Viewer 帮助文件并定位到下图位置：

Index Offset (Hex)	Access	Axis type	Data type	Phys. unit	Definition range	Description	Remarks
0x00000000	Read	every (Structure for all axis parameters)	{ UINT32 UINT32[30+1+1] UINT32 ... ... ... } }	1		General AXIS PARAMETER STRUCTURE (NO/CNC), also includes subelements such as encoder, controller and drive (see MC_ReadParameterSet in TcM2 lib) Note: Size and alignment changed.	Modified from TC3
0x00000001	Read	every	UINT32	1		Axis ID	
0x00000002	Read	every	UINT32[30+1]	1		Axis name	
0x00000003	Read	every	UINT32	ENUM		Axis type	
0x00000004	Read	every	UINT32	us		cycle time axis (SAF)	
0x00000005	Read	every	UINT32[10+1]	1		physical unit	
0x00000006	Read / Write	every	REAL64	e.g. mm/s		ref. velocity in cam direction	
0x00000007	Read / Write	every	REAL64	e.g. mm/s		ref. velocity in sync direction	
0x00000008	Read / Write	every	REAL64	e.g. mm/s		velocity hand slow	
0x00000009	Read / Write	every	REAL64	e.g. mm/s		velocity hand fast	
0x0000000A	Read / Write	every	REAL64	e.g. mm/s	[0.0..1.0E20]	velocity rapid traverse	
0x0000000F	Read / Write	every	UINT16	1	0/1	position range monitoring?	
0x00000010	Read / Write	every	REAL64	e.g. mm	[0.0..1.0E6]	position range window	
0x00000011	Read / Write	every	UINT16	1	0/1	motion monitoring?	

如：velocity hand slow, Index Group: 16#4000+ID; Index Offset: 0x00000008（其中的 ID 为 AxisID）

同时也可以到 NC 的 parameter 中查看参数的 Index Group 和 Index Offset，如下图：



程序定义 ADSREAD 功能块用于读取轴 1 的点动慢速速度

```

30   read_set_do: BOOL;
31
32   read_manualvelo:adsread;
33   manualvelo: LREAL;           申明区
34   ADSread_do: BOOL;
35   END_VAR

84   read_manualvelo(
85     NETID:='',
86     PORT:=500,
87     IDXGRP:=16#4001,
88     IDXOFFS:=16#00000008,
89     LEN:=8,
90     DESTADDR:=ADR(manualvelo),
91     READ:= ADSread_do,
92     TMOUT:=,
93     BUSY=>,                    程序区
94     ERR=>,
95     ERRID=> );

```

程序运行，对 ADSread\_do 置 TRUE，可见到结果如下，manualvelo 读到数据，此时 axis1 的慢速点动速度为 100:



```
read_manualvelo(  
  NETID[ ]:='',  
  PORT[500]:=500,  
  IDXGRP[16385]:=16#4001,  
  IDXOFFS[8]:=16#00000008,  
  LEN[8]:=8,  
  DESTADDR[18446738026784143648]:=ADR(manualvelo[100]),  
  READ TRUE:= ADSread_do TRUE, |  
  TMOUT:= ,  
  BUSY=> ,  
  ERR=> ,  
  ERRID=> );
```



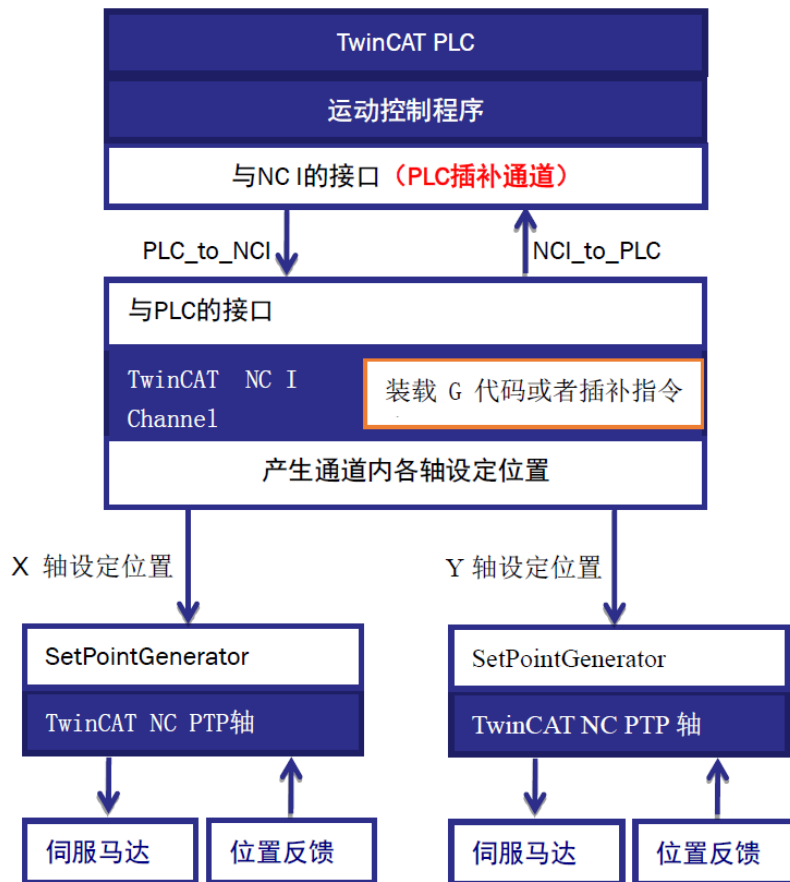
## 十、NCI 入门介绍

当用到单轴点位运动和主从跟随运动时，用 TwinCAT NC PTP 都可以实现，而插补联动就必须使用 TwinCAT NCI 或者 TwinCAT CNC 才能实现。插补联动即插补轴的运动方向上有正交关系，比如：X Y Z 轴，并在在机械上已经安装成一个整体，运动控制的目标不再是单个轴的终点位置，而是运动机构在控件上的坐标轨迹。在三维空间里，最简单的轨迹是一维线段，比如只是 X 方向移动一段距离。最常见的是二维平面上的线段，比如 XY 平面上一定斜率的直线段，以及二维平面上的圆和圆弧。直线和圆弧可以构成平面上任意的图形。TwinCAT NCI 可以实现 3 轴插补，实现机构在控件上任意的坐标轨迹，最常用的是螺旋插补，比如：XY 轴做圆弧插补的同时，Z 轴上下移动，就会在空间上形成一个螺旋轨迹。

TwinCAT NCI (TF5100) 支持 G 代码插补指令，G 代码文件是若干行 G 代码的指令，有一套规范，常用的是 G 代码和 M 代码。如直线插补指令 G01，圆弧插补指令 G02/G03。M 指令是在 G 代码指令执行过程中需要出发的开关状态。TwinCAT NCI 包含了 G 代码预读者，在执行 G 代码文件的时候，NCI 会预读 G 代码行，结合插补通道内每个轴的当前位置，分解出每个轴接下来在每个控制周期的设置位置。

TwinCAT NCI 在做插补运动时，所有轴的物理层都是在 PTP 轴中配置的。最多 31 个通道，一个 NCI 插补通道可以最多包含 3 个插补轴，5 个辅助轴。3 个插补轴的运动方向在空间上存在正交关系，通常会命名为 X Y Z 轴，进给速度就是指空间坐标系三轴的合成速度。5 个辅助轴与进给轴之间没有严格的空间关系，如果需要同时达到预定位置的其他轴，可以添加到 NCI 通道中作为辅助轴。

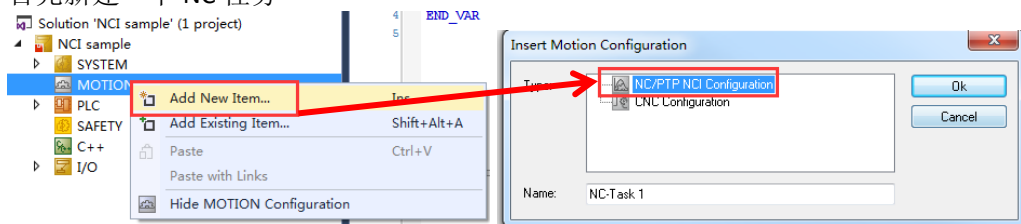
TwinCAT NC PTP 将一个电机的运动控制划分成：PLC 轴、NC PTP 轴和物理轴。而 TwinCAT NCI 把一个联动机构的控制分成三层：PLC 插补通道、NCI 插补通道、NC PTP 轴。



### 1. 在 Motion 中测试 NCI 功能

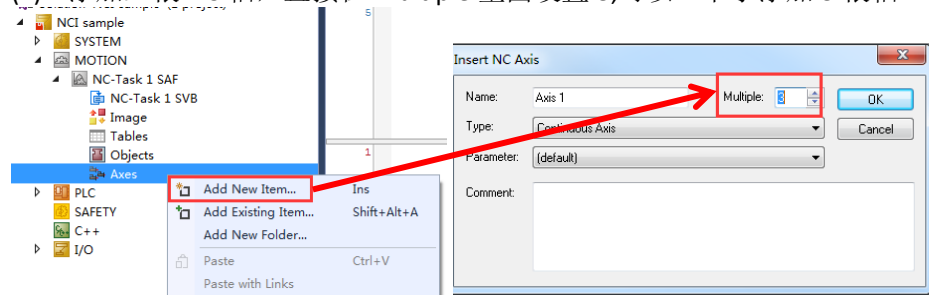
NCI 的配置方式和 NC PTP 有比较大的区别，NC PTP 可以直接通过 axis 的 online 窗口对物理轴进行调试，而 NCI 则必须通过调用 G 代码才可以让电机正反转，下面以虚轴为例，介绍如何在软件中对 NCI 功能进行调试。

(一) 首先新建一个 NC 任务

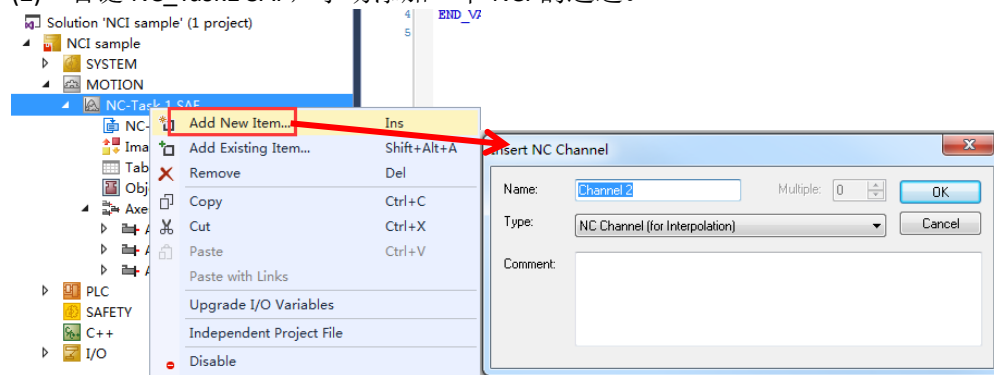


由于本次采用的是虚轴模拟的，所以需要先建三根虚轴：

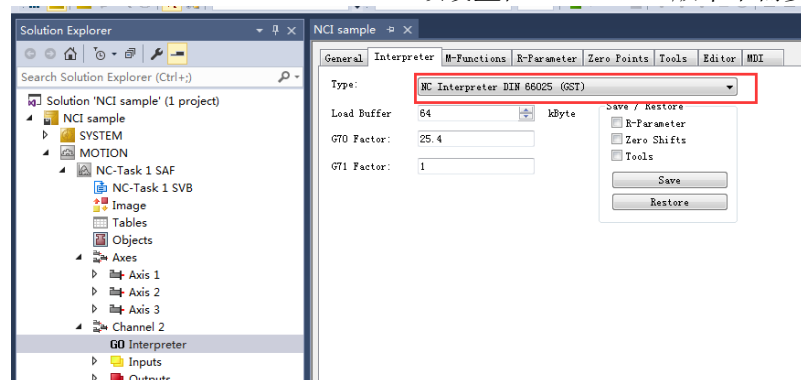
(1) 添加三根 NC 轴，直接在 Multiple 里面设置 3, 可以一下子添加 3 根轴。



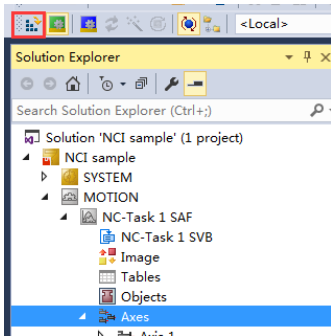
(2) 右键 NC\_Task1 SAF, 手动添加一个 NCI 的通道。



(3) 本教材采用 Simens 编程 G 代码指令，所以设置插补通道支持 G 代码指令格式为 NC Interpreter DIN66025 (Simens dialect)。该设置在 TC3 4022.2 版本中需要手动修改，其他版本默认就是该类型。

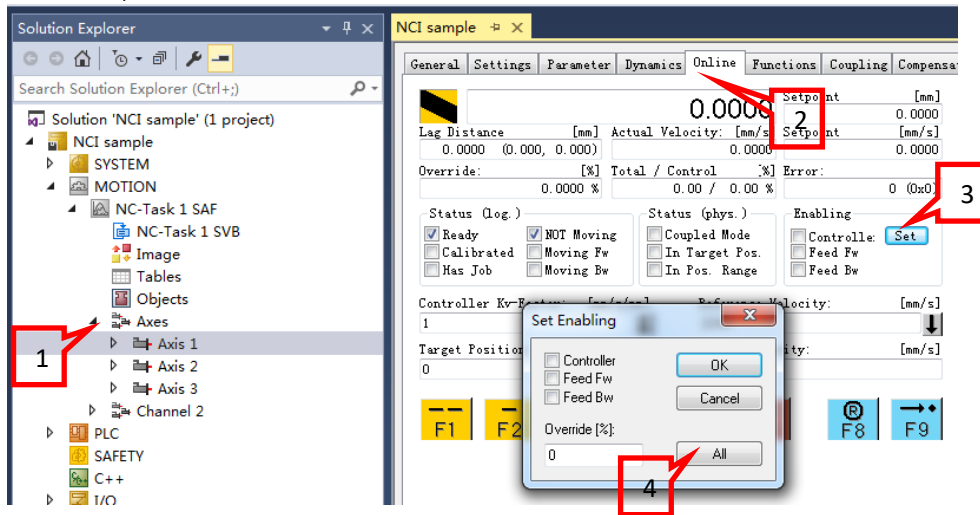


(4) 激活配置使前面添加的虚轴以及 NC 插补通道生效，输入验证码并切换 TwinCAT 至运行模式。

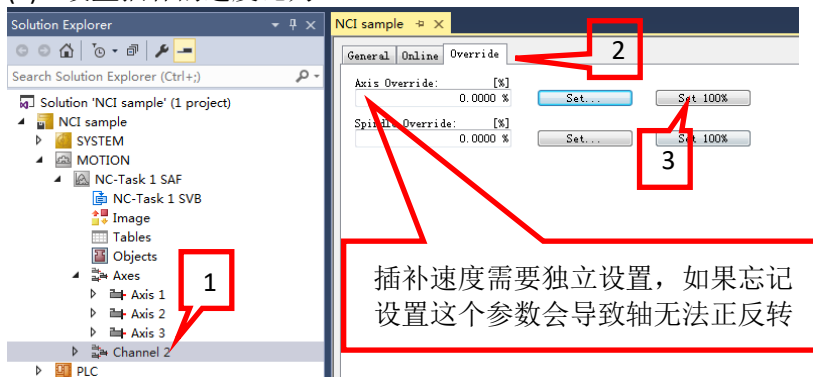


(二) 完成以上上述操作，就可以来尝试对轴进行初步调试工作：

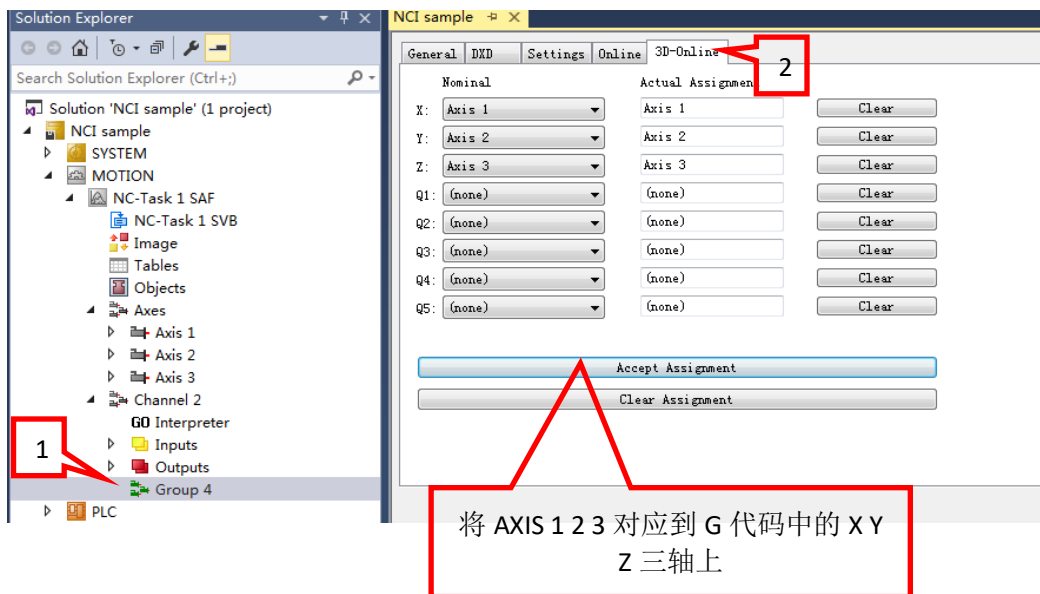
(1) 将三个虚轴使能，先选中 Axis 1，点击 Online 选项卡，点击 Set，点击 All 对轴进行使能，依次对 Axis2, Axis3 都使能上。



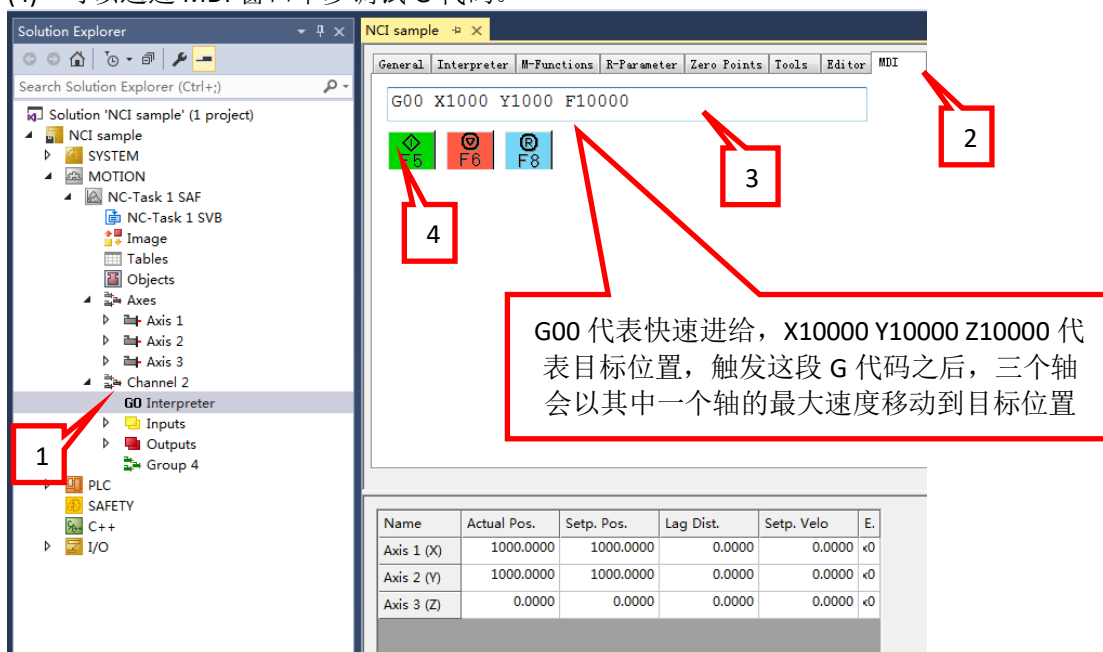
(2) 设置插补的速度比为 100%。



(3) 建立 G 代码中 XYZ 轴和 NC 轴的对应关系。



(4) 可以通过 MDI 窗口单步调试 G 代码。



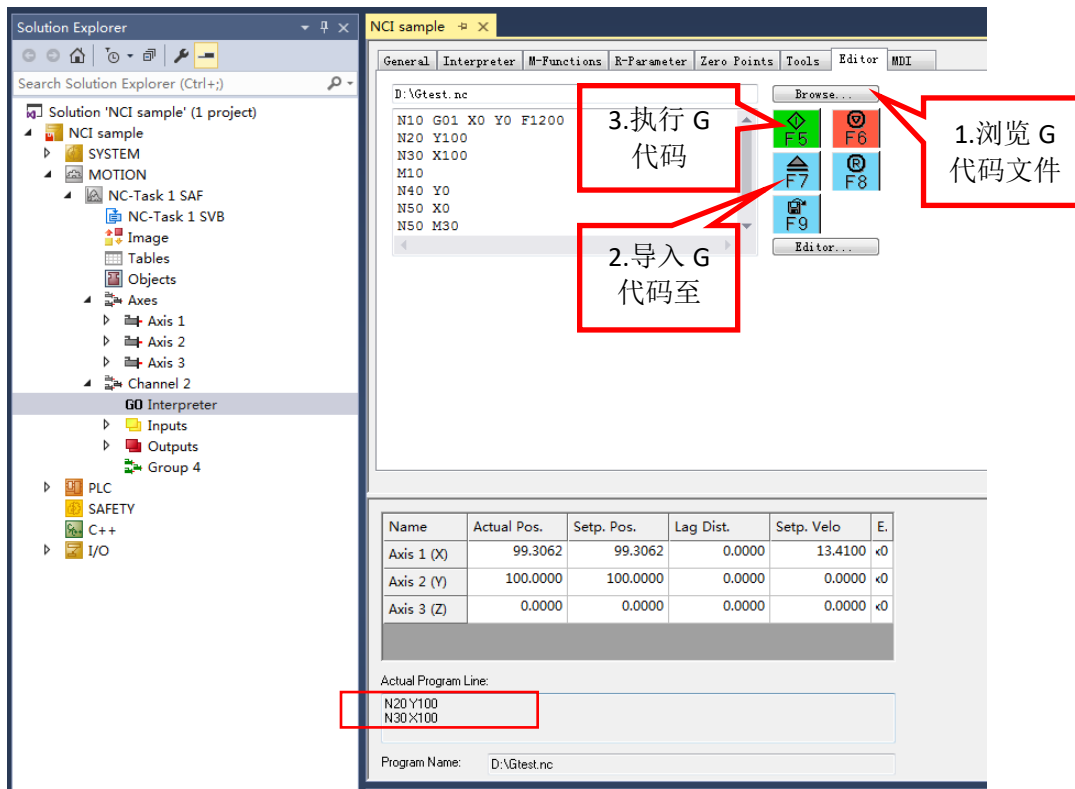
(5) Editor 窗口中可以对完整的 G 代码文件进行调试。

F5	执行 G 代码
F6	停止 G 代码
F7	加载 G 代码
F8	复位错误
F9	保存 G 代码至文件

调试步骤如下:

① 点击 Browse 选择 G 代码文件, 按 F7 将 G 代码文件导入到 NCI 中准备执行, 可以在 Program Name 中看到当前已经导入的 G 代码文件;

按 F5 执行 G 代码, 此时可以看到 Actual Program line 中当前正在执行的 G 代码行, X Y Z 轴会根据 G 代码的内容执行相应的动作。



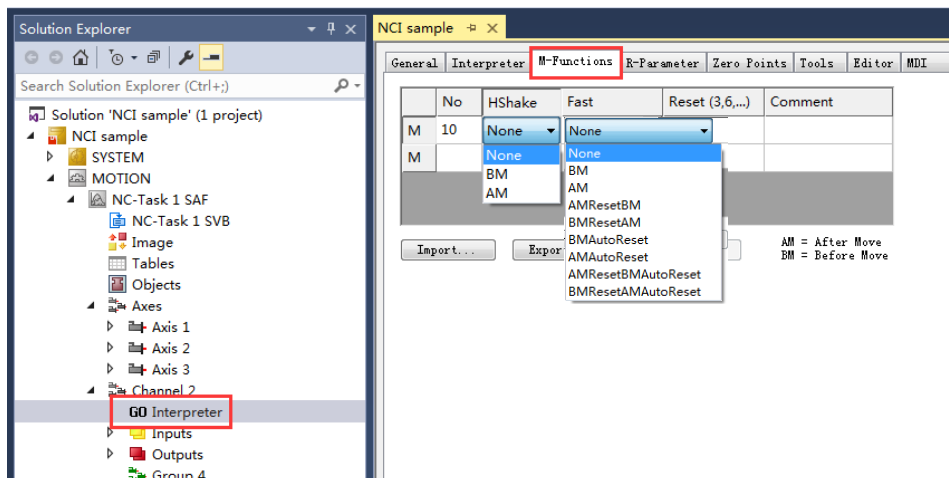
### (三) M 指令的使用

在 G 代码程序中，还会用到 M 指令。M 指令起到 G 代码与 PLC 程序交互的作用，比如 G 代码执行过程中，需要 PLC 里面做一些处理，比如换刀，吹气等操作，可以通过 M 代码来完成，除了部分根据国际标准已经指定好固定用途的 M 指令，根据 M 指令是否打断 G 代码预读，可以被分为握手型 M 指令 Hshake 和快速 M 指令 Fast。握手型 M 指令需要 NCI 与 PLC 握手，M 指令在 NCI 通道中被触发，在 PLC 程序确认这个 M 指令之后，才能继续执行后面的 G 指令。而快速 M 指令不需要 PLC 程序去确认，只是起到通知 PLC 的作用。

M 功能	含义
0..159	可以任意定义的 M 功能（除了 2，17，30）
2	程序结束
17	子程序结束
30	程序结束并删除所有快速信号位方式的 M 功能

#### (1) M 指令的定义

路径在：Channel 2\_ltp 下的 M-Functions:



其中：

**No:** M 指令的编号，最多可设置 159 个

**Hshake:** 用来设置握手方式的，AM 表示 After Motion，即当 M 指令和 G 代码同行时，M 代码会在 G 代码执行完毕后被激活；BM 表示 Before Motion，即当 M 指令和 G 代码同行时，M 代码会在 G 代码还未被执行之前被激活；None 不需要 PLC 握手确认。

**Fast:** 用于定义是否为 Fast 类型。其中有 autoreset 的代表前后动作指令复位该 M 指令，reset 则是用另一个 M 指令来复位这个指令，如果直接使用 AM 或者 BM，M 指令触发之后会一直保持为 true，复位需要使用 PLC 中的功能块 ItpresetMFuncEx，类似于 Hshake。

#### (四) R 参数的使用

G 代码中给定动作参数时可以使用固定的值，例如：X100 Y100 F1000，也可以使用 Lreal 类型的变量替代常量增加灵活性，NCI 中称为 R 参数。

如 G01 X100 Y200

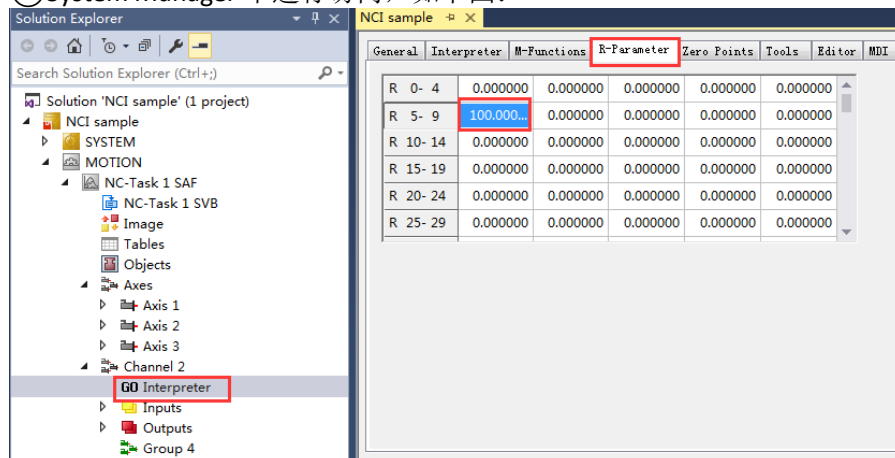
可改成：R5=100

G01 X=R5 Y=2\*R5

其中每个 NCI 通道可设置 1000 个 R 参数，从 R0 到 R999，类型均为 Lreal 型，要注意的是 NCI 会对 G 代码进行预读，因此 R 参数需要提前修改。

R 参数有三种访问方式：

① System Manager 中进行访问，如下图：

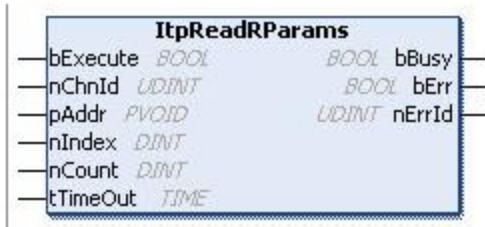


② 由 PLC 程序中进行访问，如下图：

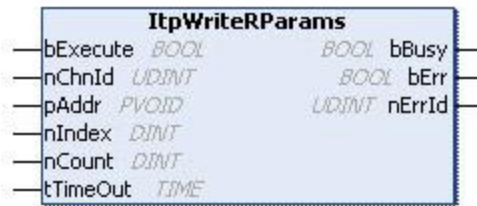
采用功能块，ItpReadRParams（读），ItpWriteRParams（写），在 Tc2\_NCI 库。



## ItpReadRParams



## ItpWriteRParams



③G 代码的方式进行访问:

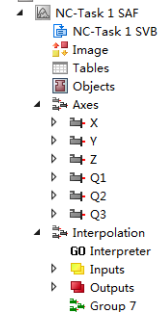
赋值 N10 R5=100

调用 G01 X=R5 Y=2\*R5

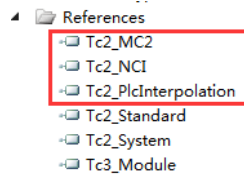
## 2. PLC 编程配合 NCI 的调试

在此部分的讲解中, 使用 Twincat3 帮助系统中的例子程序作为参考, 略有修改。

其中 NC 配置如下, 分别是插补轴 X Y Z 和辅助轴 Q1 Q2 Q3

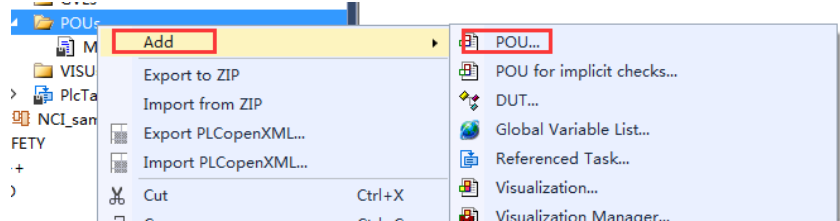


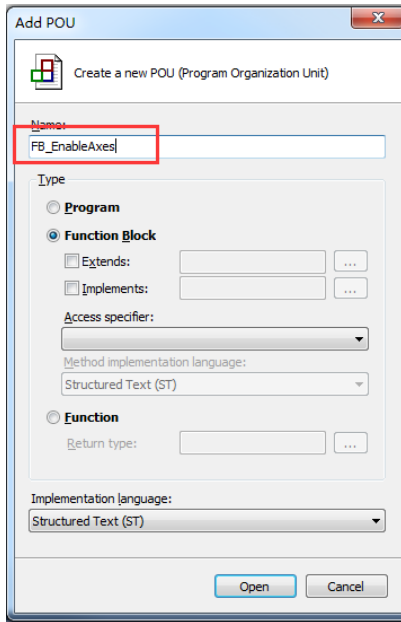
首先通过程序来做 NCI 控制, 需要添加三个库文件: ①Tc2\_MC2 ②Tc\_NCI.lib ③Tc2\_PlcInterpolation 下图所示的库文件, 程序的作用主要是配合 G 代码, 并不直接控制轴移动。



(一) 首先定义一个 FB 用于对多个轴进行使能

(1) 该功能块定义名称为 FB\_EnableAxes, 首先在 POU 下创建 FB 并进行命名





(2) 在变量申明区定义如下:

```

FUNCTION_BLOCK FB_EnableAxes
VAR_INPUT
    bEnableAxes: BOOL:= TRUE;
END_VAR
VAR_OUTPUT
    bAllAxesReady: BOOL:= FALSE;
END_VAR
VAR_IN_OUT
    stX:    AXIS_REF;
    stY:    AXIS_REF;
    stZ:    AXIS_REF;
    stM1:   AXIS_REF;
    stM2:   AXIS_REF;
    stM3:   AXIS_REF;
END_VAR
VAR
    fOverride: LREAL := 100.0;
    fbPowerX:  MC_Power;
    fbPowerY:  MC_Power;
    fbPowerZ:  MC_Power;
    fbPowerM1: MC_Power;
    fbPowerM2: MC_Power;
    fbPowerM3: MC_Power;
END_VAR

```

输入接口，默认值为true  
 输出接口，默认值为false  
 输入输出接口，  
 三个插补轴和三个辅助轴  
 中间变量，  
 轴速度比和六个用于使能的功能块

(3) 程序编程部分

首先在程序区刷新六个轴的状态，程序如下:

```

stX.ReadStatus();
stY.ReadStatus();
stZ.ReadStatus();
stM1.ReadStatus();
stM2.ReadStatus();
stM3.ReadStatus();

```

然后对每个轴进行使能，下图是对 x 轴进行使能，其他五个轴的使能与下图类似，但框出部分是不同的。

```

fbPowerX
  Enable:=bEnableAxes,
  Enable_Positive:=bEnableAxes,
  Enable_Negative:=bEnableAxes,
  Override:=fOverride,
  BufferMode:= ,
  Axis:=stX,
  Status=> ,
  Busy=> ,
  Active=> ,
  Error=> ,
  ErrorID=> );

```

- (4) 在程序区做一个赋值，若六个轴的使能均成功，则对功能块的输出接口 bAllAxesReady 置为 true，程序如下：

```

bAllAxesReady := fbPowerX.Status AND fbPowerY.Status AND fbPowerZ.Status AND
fbPowerM1.Status AND fbPowerM2.Status AND fbPowerM3.Status;

```

- (二) 定义一个 FB 用于建立插补通道，可以将用户自定义路径下的 G 代码导入插补通道并执行，可确认 H 握手指令。以 Case 作为整个框架。

- (1) 该功能块名为：FB\_SimpleNciSequence，在变量申明区做如下申明：

```

FUNCTION_BLOCK FB_SimpleNciSequence
VAR_INPUT
  bExecute           : BOOL;           (*触发位*)
  bConfirmHsk       : BOOL := FALSE;  (*确认H握手代码*)
  sPrgName          : STRING(255);    (*G代码所在地址*)
END_VAR
VAR_OUTPUT
  bBusy             : BOOL;           (*功能块在被执行*)
  bError            : BOOL;           (*出现报错*)
  bDone             : BOOL;           (*功能块被执行完毕*)
END_VAR
VAR_IN_OUT
  stX               : AXIS_REF;       (*通道的X轴*)
  stY               : AXIS_REF;       (*通道的Y轴*)
  stZ               : AXIS_REF;       (*通道的Z轴*)
  st_Q1             : AXIS_REF;       (*通道的辅助Q1轴*)
  st_Q2             : AXIS_REF;       (*通道的辅助Q2轴*)
  st_Q3             : AXIS_REF;       (*通道的辅助Q3轴*)
  stItpToPlc       : NCTOPLC_NCICHANNEL_REF; (*NCI通道反馈给PLC的*)
  stPlcToItp       : PLCTONC_NCICHANNEL_REF; (*PLC发送给NCI通道的*)
END_VAR
VAR
  nState           : UDINT := 0;
  nErrorState      : UDINT;

  fbBuildGroup     : CfgBuildExt3DGroup; (*创建通道，并将轴添加进通道*)
  fbLoadGCode      : ItpLoadProgEx;    (*装载G代码*)
  fbStartGCode     : ItpStartStopEx;   (*开始执行G代码*)
  fbClearGroup     : CfgReconfigGroup; (*清除NCI通道中的周，释放轴*)
  fbConfirmHsk     : ItpConfirmHsk;    (*确认H握手代码*)

  nInterpreterState : UDINT := 0;
END_VAR

```

- (2) nState 变量为 Case 的条件

- ① 当 nState=0 时：

```

CASE nState OF
0:
  IF bExecute THEN          (*功能块的触发位一旦触发后, 执行以下: *)
    bBusy := TRUE;         (*FB为执行状态*)
    bDone := FALSE;        (*对执行完毕功能快的引脚复位*)
    bError := FALSE;       (*报错引脚复位*)
    nErrorState := nState; (*state为0, 认为无报错*)
    nState := 10;          (*进入10步*)
  END_IF

```

(3) 当 nState=10 时:

```

10:
// 该步用来将NC轴添加进NCI通道中
fbBuildGroup(
  bExecute:=TRUE,
  nGroupId:=ItpGetGroupId(sNciToPlc:=stItpToPlc) ,
  nXAxisId:=stX.NcToPlc.AxisId,
  nYAxisId:=stY.NcToPlc.AxisId,
  nZAxisId:=stZ.NcToPlc.AxisId,
  nQ1AxisId:=st_Q1.NcToPlc.AxisId,
  nQ2AxisId:=st_Q2.NcToPlc.AxisId,
  nQ3AxisId:=st_Q3.NcToPlc.AxisId,
  nQ4AxisId:= 0,
  nQ5AxisId:= 0,
  tTimeout:= ,
  bBusy=> ,
  bErr=> ,
  nErrId=> );
IF NOT fbBuildGroup.bBusy THEN
  IF NOT fbBuildGroup.bErr THEN
    nState := 20; (*如果将NC轴添加进NCI通道中成功后, 进入20步*)
  ELSE
    (* add error handling *)
    nErrorState := nState; (*如果出现报错, 首先FB出现报错, 然后将错误状态定位在10步, 并且进入9999步*)
    bBusy := FALSE;
    bError := TRUE;
    nState := 9999;
  END_IF
  fbBuildGroup(bExecute:=FALSE); (*对fbBuildGroup的触发位复位*)
END_IF

```

(4) 当 nState=20 时:

```

20:
(*将G代码装在进通道, sPrgName有两种用法, 第一种写出路径+G代码程序的文件名, 第二种直接写G代码程序的文件名,但需要放置在安装盘中TwinCAT\MC\NCI的文件夹*)
// load g-code file
// pls. ensure that first.nc is available in TwinCAT\MC\NCI-folder
fbLoadGCode(
  sNciToPlc:=stItpToPlc,
  bExecute:=TRUE,
  sPrg:= sPrgName,
  nLength:= INT_TO_UDINT(LEN(sPrgName)),
  tTimeout:= ,
  bBusy=> ,
  bErr=> ,
  nErrId=> );
IF NOT fbLoadGCode.bBusy THEN
  IF NOT fbLoadGCode.bErr THEN
    nState := 30; (*如果装载G代码成功后, 进入20步*)
  ELSE
    (* add error handling *)
    nErrorState := nState; (*如果出现报错, 首先FB出现报错, 然后将错误状态定位在20步, 并且进入9999步*)
    bBusy := FALSE;
    bError := TRUE;
    nState := 9999;
  END_IF
  fbLoadGCode(sNciToPlc:=stItpToPlc, bExecute:=FALSE); (*对fbLoadGCode的触发位复位*)
END_IF

```

(5) 当 nState=30 时:

```

30:
(*检查通道状态, 如果处于ready, 那么说明装载G代码成功*)
// we have to check the status of the interpreter. It has to be in ready state, in case of loading the program successfully.
nInterpreterState := ItpGetStateInterpreter(sNciToPlc:=stItpToPlc); (*获取通道当前状态*)
IF nInterpreterState = Tc2_NCI.NCI_INTERPRETER_READY THEN (*判断通道是否为ready状态*)
    nState := 40;
ELSE
    (* add error handling *)
    nErrorState := nState; (*如果通道没有到ready状态, 将错误状态定位在30步, 并且进入9999步*)
    bBusy := FALSE;
    bError := TRUE;
    nState := 9999;
END_IF

```

(6) 当 nState=40 时:

```

40:
(*开始执行G代码*)
// start g-code file
fbStartGCode(
    bStart:=TRUE,
    bStop:=FALSE,
    tTimeOut:= ,
    sNciToPlc:= stItpToPlc,
    bBusy=> ,
    bErr=> ,
    nErrId=> );
IF NOT fbStartGCode.bBusy THEN
    IF NOT fbStartGCode.bErr THEN (*如果开始执行G代码了, 进入50步*)
        nState := 50;
    ELSE
        (* add error handling *)
        nErrorState := nState; (*如果未能成功执行, 将错误状态定位在40步, 并且进入9999步*)
        bBusy := FALSE;
        bError := TRUE;
        nState := 9999;
    END_IF
    fbStartGCode( bStart:=FALSE, sNciToPlc:= stItpToPlc ); (*对fbStartGCode的触发位复位*)
END_IF

```

(7) 当 nState=50 时:

```

50:
(*检查状态, 确认G代码是否执行完毕*)
// check state, again - we are at least not in ready state for several ticks
// this is to ensure that we don't indicate program has finished, before we have started
nInterpreterState := ItpGetStateInterpreter(sNciToPlc:=stItpToPlc);
IF nInterpreterState <> Tc2_NCI.NCI_INTERPRETER_READY THEN (*判断通道是否为ready状态*)
    // nci is running
    nState := 60; (*不为ready的状态, 进入60步*)
END_IF

```

(8) 当 nState=60 时:

```

60:
(*获取通道的当前状态*)
nInterpreterState := ItpGetStateInterpreter(sNciToPlc:=stItpToPlc);
IF nInterpreterState = Tc2_NCI.NCI_INTERPRETER_READY THEN (*判断是否回到ready状态, 回到说明G代码已经执行完毕, 进入70步*)
    // program has finished
    nState := 70;
ELIF nInterpreterState = Tc2_NCI.NCI_INTERPRETER_ABORTED THEN (*如果通道状态变成aborted的话, 说明通道此时出现报错, 具体需要通过通道报错的ID进行排查原因*)
    // a run-time error occurred - this could be a lag error or something else...
    // add error handling
    ;
END_IF

```

(9) 当 nState=70 时:

```

70:      (*G代码执行完毕, 释放NC轴, 解散NCI通道*)
      // program has finished
      // now clear interpolation group
      fbClearGroup(
        bExecute:=TRUE,
        nGroupId:=ItpGetGroupId(sNciToPlc:=stItpToPlc) ,
        tTimeout:= ,
        bBusy=> ,
        bErr=> ,
        nErrId=> );
      IF NOT fbClearGroup.bBusy THEN
        IF NOT fbClearGroup.bErr THEN (*如果释放完毕, 进入80步*)
          nState := 80;
        ELSE
          (* add error handling *)
          nErrorState := nState; (*如果未能成功执行, 将错误状态定位在70步, 并且进入9999步*)
          bBusy := FALSE;
          bError := TRUE;
          nState := 9999;
        END_IF
      fbClearGroup(bExecute:=FALSE); (*对fbClearGroup的触发位复位*)
    END_IF

```

(10) 当 nState=80 时:

```

80:      (*以上已经完成该功能块的目标, 对输出位DONE置为TRUE, 说明已完成G代码的执行*)
      bDone := TRUE;
      bBusy := FALSE;
      IF NOT bExecute THEN (*若此功能块的触发位没有复位, 那么nState回到0步*)
        nState := 0;
      END_IF

```

(11) 当 nState=9999 时:

```

9999:      // error state
      IF NOT bExecute THEN (*若此功能块的触发位没有复位,那么回到初始0步, 重新执行*)
        nState := 0;
        bError := FALSE;
      END_IF
    END_CASE

```

(12) 用于确认 M 握手指令

(\*一旦G代码中出现M握手型指令, 用于进行确认M握手指令\*)

```

fbConfirmHsk(
  bExecute:=bConfirmHsk ,
  sNciToPlc:=stItpToPlc ,
  sPlcToNci:=stPlcToItp ,
  bBusy=> ,
  bErr=> ,
  nErrId=> );

```

以上就完成了对功能块 FB\_SimpleNciSequence 的定义。

(三) 主程序 MAIN 中, 我们用以上两个功能块来执行 G 代码验证其可行性。

(1) 首先变量申明窗口

```

PROGRAM MAIN
VAR
  bUserEnableAxes:    BOOL      := TRUE;      (*触发使能功能块, 由于是模拟程序, 所以直接对轴组进行使能*)
  fUserOverride:      LREAL     := 100.0;     (*设置通道速比为100%*)

  io_X:               AXIS_REF;              (*定义各NC轴*)
  io_Y:               AXIS_REF;
  io_Z:               AXIS_REF;
  io_Q1:              AXIS_REF;
  io_Q2:              AXIS_REF;
  io_Q3:              AXIS_REF;

  in_stItpToPlc      AT %I*: NcToPlc_NciChannel_Ref; (*定义NCI通道反馈给PLC的一系列状态字等*)
  out_stPlcToItp     AT %Q*: PLCTONC_NCICHANNEL_REF; (*定义PLC发送给NCI通道的一系列状控制字等*)

  fbEnableAxes:      FB_EnableAxes;          (*调用轴组使能功能块*)
  bAllAxesReady:     BOOL;                   (*该变量用于标明轴组使能成功*)

  fbSimpleNciSequence: FB_SimpleNciSequence; (*调用生成NCI通道及执行G代码的功能块*)
  bExecSimpleNci:    BOOL      := FALSE;     (*用于触发生成NCI通道及执行G代码的功能块*)
  bExeConfirmHsk:    BOOL      := FALSE;     (*用于确认M握手指令*)
  sPrgName:           STRING(255) := 'Gtest.nc'; (*放置在安装盘中TwinCAT\MC\NCI的文件夹*)
END_VAR

```

(2) 主程序区进行如下编程:

```

fbEnableAxes(
  bEnableAxes:= bUserEnableAxes,
  stX:=io_X ,
  stY:=io_Y ,
  stZ:=io_Z ,
  stM1:=io_Q1 ,
  stM2:=io_Q2 ,
  stM3:=io_Q3 ,
  bAllAxesReady=>bAllAxesReady );

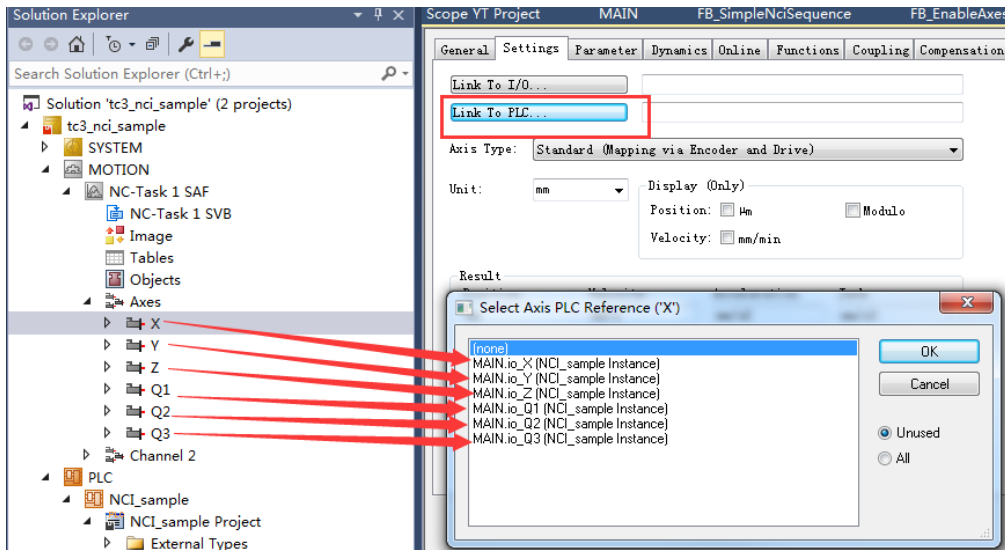
ItpSetOverridePercent(fOverridePercent:=fUserOverride , sPlcToNci:=out_stPlcToItp);

fbSimpleNciSequence(
  bExecute:=bExecSimpleNci ,
  bConfirmHsk:=bExeConfirmHsk ,
  sPrgName:=sPrgName ,
  bBusy=> ,
  bError=> ,
  bDone=> ,
  stX:=io_X ,
  stY:= io_Y,
  stZ:= io_Z,
  st_Q1:=io_Q1 ,
  st_Q2:=io_Q2 ,
  st_Q3:= io_Q3,
  stItpToPlc:=in_stItpToPlc,
  stPlcToItp:= out_stPlcToItp);

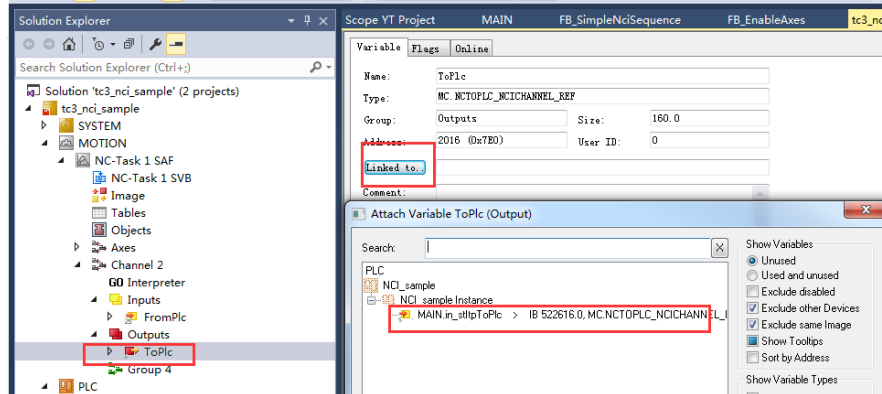
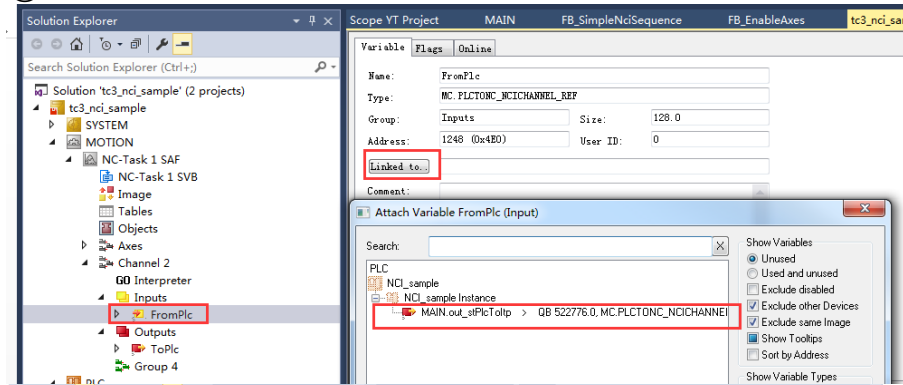
```

(3) 接下来需要对 NC 轴以及 NCI 通道进行绑定

① 对 X Y Z Q1 Q2 Q3 进行绑定



② 对 NCI 通道进行绑定



(4) 完成以上操作，激活配置，并且 Log in 运行程序对进行置为 TRUE，观察 interpreter 的状态如下，说明已经开始执行 G 代码

Name	Actual Pos.	Setp. Pos.	Lag Dist.	Setp. Velo
X (X)	9.1643	9.1643	0.0000	19.9980
Y (Y)	100.0000	100.0000	0.0000	0.0000
Z (Z)	0.0000	0.0000	0.0000	0.0000
O1 (O1)	0.0000	0.0000	0.0000	0.0000

Actual Program Line:



### 上海（中国区总部）

中国上海市静安区汶水路 299 弄 9号（市北智汇园）

电话：021-66312666

传真：021-66315696

邮编：200072

### 北京分公司

北京市西城区新街口北大街 3 号新街高和大厦 407 室

电话：010-82200036

传真：010-82200039

邮编：100035

### 广州分公司

广州市天河区珠江新城珠江东路16号高德置地G2603室

电话：020-38010300/1/2

传真：020-38010303

邮编：510623

### 成都分公司

成都市锦江区东御街18号 百扬大厦2305 房

电话：028-86202581

传真：028-86202582

邮编：610016



请用微信扫描二维码  
通过公众号与技术支持交流

倍福中文官网：

<http://www.beckhoff.com.cn/>

倍福虚拟学院：

<http://tr.beckhoff.com.cn/>

招贤纳士：[job@beckhoff.com.cn](mailto:job@beckhoff.com.cn)

技术支持：[support@beckhoff.com.cn](mailto:support@beckhoff.com.cn)

产品维修：[service@beckhoff.com.cn](mailto:service@beckhoff.com.cn)

方案咨询：[sales@beckhoff.com.cn](mailto:sales@beckhoff.com.cn)