

BECKHOFF GUANGZHOU

TwinCAT NC I

实用教程

(Based on TcMc2.lib)

LizzyChen

History Version:

TwinCAT NC I 实用教程 Version 1.01	2016.10.16
TwinCAT NC I 实用教程 Version 1.02	2016.10.23
TwinCAT NC I 实用教程 Version 1.03	2016.11.03
TwinCAT NC I 实用教程 Version 1.04	2016.11.19
TwinCAT NC I 实用教程 Version 1.05	2016.12.24
TwinCAT NC I 实用教程 Version 1.06	2017.06.07

注意:

第 3、5、7 章的例程是功能递增的关系，可以直接参考功能最完整的第 7 章例程。

Version 1.07

(最新版本:)

前 言

本书是在《TwinCAT NC PTP 实用教程》的基础上，讲述 TwinCAT 运动控制软件的插补功能，即控制多个伺服轴在空间上的插补运动。

我们先回顾运动控制的功能分类。在运动控制领域，按联动关系可以分为单轴点位运动、主从跟随运动、多轴插补运动。最简单的单轴点位运动就是控制电机匀速运动、绝对定位、相对定位，再复杂一点就是模长内定位、用外部位置发生器控制电机走一个自定义的位置曲线。最简单的主从联动是速度跟随和位置跟随，力矩跟随也算一种跟随，但不是标准的运动控制功能，要通过 PLC 编程实现。速度跟随就是电子齿轮 (Gear)，位置跟随就是电子凸轮 (CAM)，在速度跟随、位置跟随、力矩跟随的基础上，针对专门的应用场合，又开发了飞剪、飞锯、张力控制等功能，用到具体的设备上，又常用工艺表达为：横切、纵切、旋切、收放卷控制等等。

单轴点位运动和主从跟随运动用 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。这里所说的插补联动，是指插补轴的运动方向在空间上有正交关系，比如 X、Y、Z 轴，并且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，而是运动机构在空间上的坐标轨迹。在三维空间里，最简单的轨迹是一维线段，比如只在 X 方向移动一段距离。最常用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，实现运动机构在空间上任意的坐标轨迹，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 支持两种插补指令的接口：G 代码文件和 FeedTable。

G 代码文件是若干行 G 代码的集合，G 代码有一套规范，常用的是 G 指令和 M 指令。最简单的直线插补指令 G01，圆弧插补指令 G02/G03。M 指令是在 G 代码文件执行过程中需要触发的开关状态。

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。

FeedTable 的区别是，G 代码不是写在 G 代码文件中，而是从 PLC 程序临时填入插补指令表。可以填入插补指令表的指令与 G 代码文件中的指令类型大致相当，也包括直线插补、圆弧插补、M 指令等等，但不再出现 G01、G02 等字样，而是以插补指令的类型枚举值来区分。

TwinCAT NCI 做插补运动时，所有轴的物理层都是在 PTP 轴中配置的。

0. 1 本书读者对象

本书的目的是教您如何尽可能快捷地编写 TwinCAT NC I 应用程序，并假定您已经熟练掌握了 TwinCAT PLC、TwinCAT NC PTP 编程。

本书适合于以下情况：

1) 习惯使用 G 代码的 CNC 数控系统的用户。

这类用户需要了解 TwinCAT NCI 支持的 G 代码语法与其原来所用系统的 G 代码语法的细微差别。虽然都是 DIN66025 标准，在一些细微的地方，还是有所不同的。

2) 习惯使用插补指令的运动控制器的用户

在 PLC 库提供的原始插补指令功能块中，并没有一个现成的 FB 用于直线插补或者圆弧插补，插补指令是一条一条地填入列表，批量执行。当然，如果填一条执行一条，完成后才填入下一条，就等效于单个的插补动作了。指令表的形式更加灵活、强大，但使用起来毕竟不同于以往的习惯，所以要个适应过程。还有一种做法是，自己封装一个 FB，接口类似第三方运动控制器的插补功能块。

0. 2 本书主要内容

第 1 章，TwinCAT NC I 的系统概述

第 2 章，在 System Manager 中测试 NCI 功能

第 3 章，使用 G 代码的插补运动项目

第 4 章，使用 FeedTable 的插补运动项目

第 5 章，关于 M 函数

第 6 章，G 代码文件中的指令简介

第 7 章，回溯和单步执行

第 8 章，插补运动中常见需求（样条过渡，限速，限加速度，寻参，JumpVelo）

第 9 章，在 TwinCAT 3 下使用 NCI 功能的差别说明

0.3 版本说明

本书所提供的操作截图、程序代码都基于 TwinCAT 2.11(Build 2254)。所述例程基于 TcMc2.Lib, 该库兼容于 PLC Open Motion Control Version2.0。截至目前, 由于 BECKHOFF 公司的 TwinCAT 软件仍然会持续升级和更新, 我们不排除后续版本的操作界面会发生变化, 而例程中的代码也有可能不适用于后续版本。

在 TwinCAT 3 下使用 NC I 功能, 与 TwinCAT 2 下几乎没有区别。绝大部分规则仍然适用, 只是界面上有所不同。

开发及模拟调试 TwinCAT NC I 程序需要 Windows NT/2000/XP/Win 7 32 位操作系统。

0.4 勘误表

尽管我们竭尽所能来确保在正文和代码中没有错误, 但也难免会发生错误。如果您在本书中发现了错误 (例如拼写错误或者代码错误), 我们将非常感谢您的反馈。发送勘误表将节省其它读者的时间, 同时也会帮助我们提供更高质量的信息。

[请发邮件至 L.Chen@Beckhoff.com.cn](mailto:L.Chen@Beckhoff.com.cn), 该邮箱由作者本人查收, 我会检查您的反馈信息。如果是正确的, 将在本书的后续版本中使用。

0.5 感谢

用户的需求是我们成长的动力, 我本人使用 TwinCAT NCI 的时间并不长, 经验不算丰富。本书只是我在支持 NCI 项目中的一些知识总结。通过梳理这些零散的知识, 令其系统有序, 并将实际应用中常见的工艺要求、常见的问题以及解决办法一一列举出来, 希望可以帮助后来的同事和用户, 至少在我遇到过的这些问题上可以少走弯路。

在此必须感谢长期给予我支持的 TwinCAT CNC 专家, 倍福中国应用部高级 CNC 工程师史晓云。他编写的《TwinCAT CNC 简明调试教程》第三章 CNC 系统编程指令, 对 NCI 用户也适用。

作者 2016-12-24 于广州

修订记录(V1.01)

2017-06-06, 2.3 NCI 通道参数设置

应华为肖工请求, 希望详细了解通道每个参数的作用。尤其是 DXD 页面那些。

2016-10-16, 周末整理

列提纲。

目 录

前 言	2
目 录	7
1 TwinCAT NC I 系统概述	10
1.1 插补通道	11
1.2 PLC 控制 NCI 插补通道的几个途径	12
1.3 插补指令的两种形式	13
1.3.1 G 代码	14
1.3.2 FeedTable	15
1.4 M 函数：插补运动与逻辑动作的协调	15
1.5 R 参数：NC I 通道与 PLC 的浮点数接口	16
2 在 System Manager 中测试 NCI 功能	17
2.1 创建 TwinCAT NC PTP 轴和 NCI 通道	17
2.1.1 创建 NC 任务和 PTP 轴	17
2.1.2 创建 NC I 通道	18
2.1.3 存盘和激活配置	20
2.1.4 PTP 轴调试界面	22
2.2 NCI 通道调试界面	23
2.2.1 配置 NC I 通道的插补轴所对应的 PTP 轴	23
2.2.2 配置 NC I 通道的输出倍率	23
2.2.3 定位和编辑 G 代码文件	24
2.2.4 测试 G 代码文件	26
2.2.5 解散插补通道，恢复 PTP 轴	30
2.3 NCI 通道的运动参数设置	34
2.3.1 插补曲线转折的分类	34
2.3.2 参数详解	34
3 使用 G 代码的插补运动项目	40
3.1 在 PLC 中新建 NCI 程序	40
3.1.1 准备工作	40
3.1.2 新建 NCI 通道控制的基本 FB 及其接口变量	41
3.1.3 编写基本代码	43
3.1.4 编写 NCI 通道控制 FB 的触发逻辑	46
3.2 在 System Manager 中引用 NCI 程序	46
3.3 确认控制器上的 CNC 文件路径与 PLC 程序中一致	48
3.4 PLC 程序下载运行，强制变量 bExecute 执行各种指令	48
3.5 NCI 通道 G 代码控制 FB 封装示例：FB_NCI_GCode	55
3.5.1 功能块的调用	55
3.5.2 控制对象和 Interface	57
3.5.3 FB_NCI_GCode 的源代码	60

3.5.4	PTP 控制程序.....	61
3.5.5	其它程序.....	62
3.5.6	System Manager 配置文件.....	66
3.5.7	调试画面.....	67
3.5.8	测试 NCI 插补功能的操作顺序.....	68
4	使用 FeedTable 的插补运动项目.....	69
4.1	在 PLC 中新建 NCI 程序.....	69
4.1.1	准备工作.....	70
4.1.2	新建 FeedTable 控制的基本 FB 及其接口变量.....	72
4.1.3	编写基本代码.....	73
4.1.4	编写 FeedTable 通道控制 FB 的触发逻辑.....	78
4.2	在 System Manager 中引用 NCI 程序.....	78
4.3	PLC 程序下载运行, 强制变量 bExecute 执行各种指令.....	80
4.4	FeedTable 控制 FB 封装示例: FB_MC_FeedTable.....	86
4.4.1	功能块的调用.....	87
4.4.2	控制对象和 Interface.....	88
4.4.3	FB_MC_FeedTable 的源代码.....	91
4.4.4	PTP 控制程序.....	91
4.4.5	测试程序说明.....	92
4.4.6	System Manager 配置文件.....	99
4.4.7	调试画面.....	100
4.4.8	测试 FeedTable 控制 NCI 通道的操作顺序.....	101
4.5	FeedTable 可以填充的非运动指令.....	101
5	关于 M 函数.....	106
5.1	M 函数的定义.....	107
5.2	显示和复位 M 函数的状态.....	108
5.3	用 PLC 函数获取 M 函数的状态及复位.....	109
5.4	使用 M 函数的 NCI 项目举例.....	110
5.4.1	G 代码文件.....	110
5.4.2	M 函数的设置.....	111
5.4.3	PLC 代码.....	111
5.4.4	测试画面.....	114
5.4.5	测试 NCI 插补功能的操作顺序.....	114
6	G 代码文件中的指令简介.....	115
6.1	G 指令.....	116
6.2	SHT 指令.....	118
6.2.1	S 指令.....	118
6.2.2	H 指令.....	119
6.2.3	T 指令和 D 指令.....	120
6.3	R 参数.....	122
6.4	@指令.....	123
6.4.1	跳转指令.....	124

6.4.2	分支指令.....	125
6.4.3	循环指令.....	126
6.4.4	子程序.....	127
6.4.5	读取实际轴的位置.....	129
6.5	#Set 参数设置命令.....	130
6.6	Command 命令.....	133
7	回溯 Retrace 和单步 SingleBlock.....	137
7.1	回溯 Retrace.....	137
7.1.1	什么是回溯.....	137
7.1.2	回溯的程序处理.....	137
7.1.3	启用回溯功能的 NCI 例程.....	139
7.1.4	测试 Retrace 功能的操作顺序.....	143
7.2	单步执行 Single Block.....	144
7.2.1	什么是单步执行 Single Block.....	144
7.2.2	单步执行的程序处理.....	144
7.2.3	启用单步功能的例程.....	145
7.2.4	测试单步功能 SingleBlock 的操作顺序.....	147
8	插补运动中常见需求.....	148
8.1	曲线平滑.....	148
8.2	速度限制.....	149
8.3	加速度限制.....	149
8.4	寻参.....	149
8.5	速度平滑.....	149
8.6	坐标偏置.....	152
8.7	主轴.....	152
8.8	刀具补偿.....	153
8.8.1	刀具补偿参数.....	153
8.8.2	刀具补偿的 G 代码.....	155
8.8.3	刀具示例 1: 长度和半径补偿.....	155
8.8.4	刀具示例 2: 刀具偏置.....	157
9	在 TwinCAT 3 下使用 NCI.....	158

1 TwinCAT NC I 系统概述

我们首先回顾一下运动控制的功能分类：按联动关系可以分为单轴点位运动、主从跟随运动、多轴插补运动。最简单的单轴点位运动就是控制电机匀速运动、绝对定位、相对定位，再复杂一点就是模长内定位、用外部位置发生器控制电机走一个自定义的位置曲线。最简单的主从联动是速度跟随和位置跟随，力矩跟随也算一种跟随，但不是标准的运动控制功能，要通过 PLC 编程实现。速度跟随就是电子齿轮（Gear），位置跟随就是电子凸轮(CAM)，在速度跟随、位置跟随、力矩跟随的基础上，针对专门的应用场合，又开发了飞剪、飞锯、张力控制等功能，用到具体的设备上，又常用工艺表达为：横切、纵切、旋切、收放卷控制等等。

单轴点位运动和主从跟随运动用 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。TwinCAT NCI 中的“T”，就是 Interplation（插补）的首字母。这里所说的插补联动，是指插补轴的运动方向在空间上有正交关系，比如 X、Y、Z 轴，并且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，而是运动机构在空间上的坐标轨迹。在三维空间里，最简单的轨迹是一维线段，比如只在 X 方向移动一段距离。最常用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，实现运动机构在空间上任意的坐标轨迹，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。

1.1 插补通道

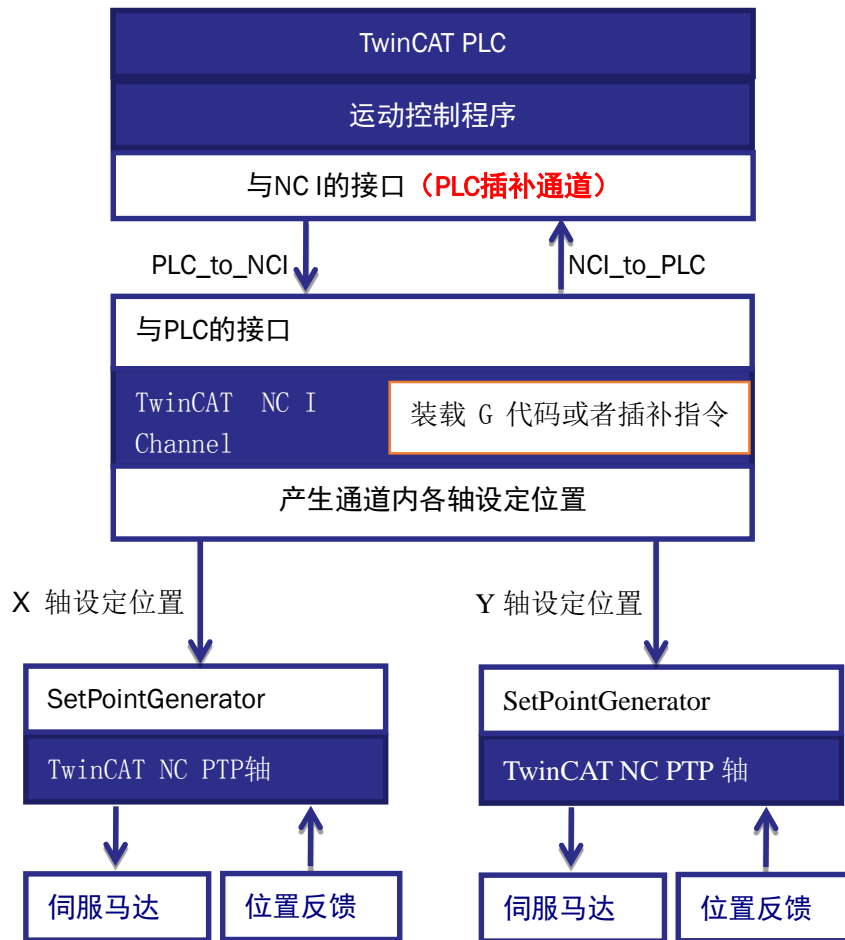
TwinCAT NC I 做插补运动时，是完全基于 TwinCAT NC PTP 的，所有轴的物理层都是在 PTP 轴中配置的，而它们在 PLC 程序中的接口仍然是类型为 `Axis_Ref` 的变量。所以 PTP 控制中的 3 个轴类型：PLC 轴、NC 轴、物理轴在 NC I 中仍然适用，`TcMc2.lib` 中的所有功能块也仍然适用于这些轴。

TwinCAT NC I 是作为一种 PTP 轴的联动关系来定义和使用的。建立联动关系之前，每个轴可以独立运动。如果联动关系是运动方向在空间上的正交关系，比如机械上已经安装成一个整体，一个轴控制 X 方向的运动，另一个轴控制 Y 方向的运动，另一个轴控制 Z 方向的运动。为了控制这个机构的整体运动，专门建立一个 NC I Channel 即插补通道作为它的模型，也就是软件上的控制对象。换句话说，与 TwinCAT NC PTP 中建立一个 NC 轴作为伺服电机的控制对象一样，TwinCAT NC I 中建立一个 NC I 插补通道作为三维正交联动机构的控制对象。

一个 NC I 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，通常命名为 X、Y、Z 轴，进给速度就是指这三个轴的合成速度（没有正交关系的轴是无法确定合成速度的）。5 个辅助轴与进给轴之间没有严格的空空间关系，需要同时达到预定位置的其它轴，可以添加到 NC I 通道中作为辅助轴控制。

TwinCAT NC I 可以包含多少个插补通道呢？理论上，只要所有 PTP 轴、NC I 通道、其它通道的总数不超过 255 即可。假如每个 NC I 通道都只有 XY 两轴插补，没有辅助轴，那么理论上可以有 170 个 PTP 轴和 85 个插补通道。假如每个 NC I 通道有 3 个插补轴，5 个辅助轴，理论上可以有 224 个 PTP 轴和 28 个插补通道。实际上，一个机床上当然不会有这么多需要联动机构。

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。而 TwinCAT NC I 把一个联动机构的控制分为三层：PLC 插补通道、NC I 插补通道、NC PTP 轴。



1.2 PLC 控制 NCI 插补通道的几个途径

ADS 接口：PLC 通过 ADS 接口可以控制 NCI 插补通道：

- 组建插补通道、解散插补通道
- 装载 G 代码文件到插补通道
- 插补运动的启动、停止、复位
- 读取缓存的插补指令数量及其它标记

接口变量 PLCTONC：每个 PLC 周期更新 NCI 通道的控制信号

倍率修改：

接口变量 NCTOPLC：每个 PLC 周期更新 NCI 通道的状态

通道状态，故障代码，当前进给速度，正在运行的插补指令编号。

R 参数：可以从 PLC 读写的浮点型参数，在 G 代码中也可以使用和设置 R 参数。

M 函数：在 NCI 通道运动控制中触发的 Bool 型状态变量。PLC 可以读取它的状态。

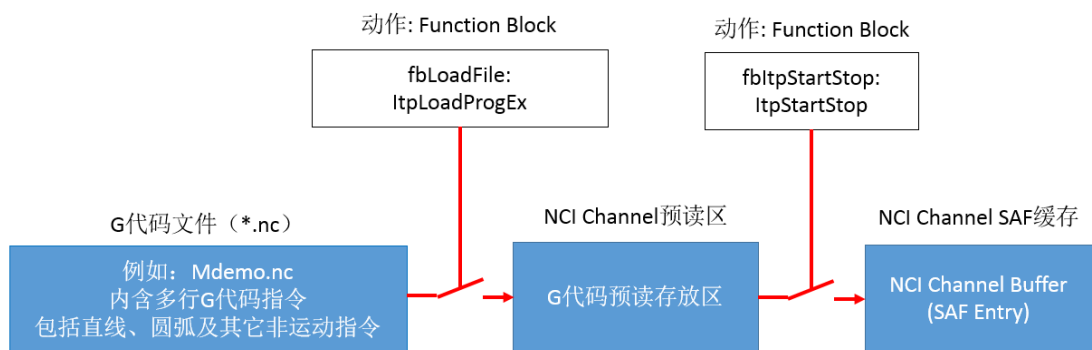
1.3 插补指令的两种形式

让一个插补通道运动之前，必须定义好，它的 X、Y、Z 轴分别对应哪个 PTP 轴，然后就可以控制它运动了。我们可以让它执行哪些运动呢？比起 PTP 轴，NCI 通道能够执行插补运动种类极少：仅包含直线、圆弧、螺旋三种，甚至螺旋都很少用。每一个插补运动指令都必须配有对应的参数，比如直线插补指令包含终点坐标和进给速度，而圆弧插补指令包含圆心、半径、弧长或者终点坐标等参数。

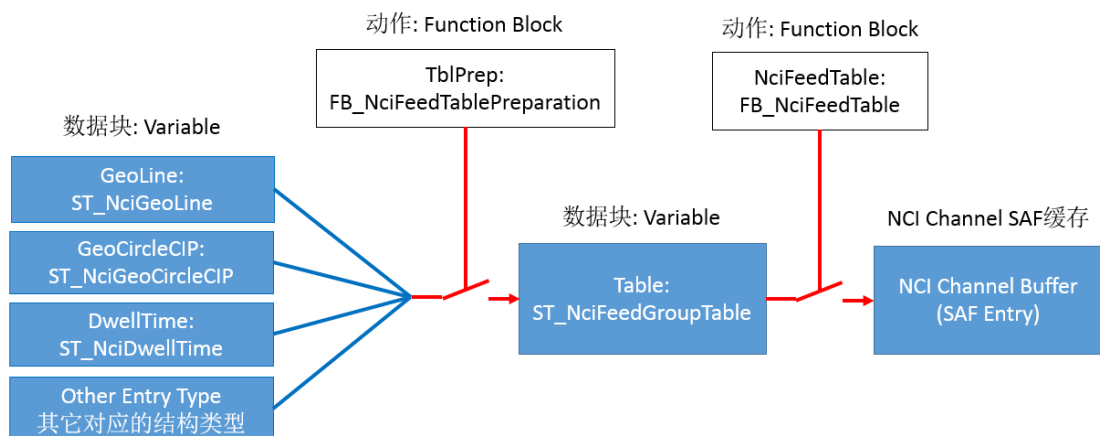
如前所述，插补运动控制的目标不再是单个轴的终点位置，而是运动机构在空间上的坐标轨迹。换句话说，插补运动要求机构到达某个空间位置，但可能并不要求它在那个位置停下来，对于连续加工来说，最好能够保持进给速度的稳定。所以插补指令不能给一条，执行一条，必须有相当的 Buffer，才能预读或者前瞻。在前一条指令未结束时，其实后一条指令的路径规划已经完成了，才可能让运动连贯。

TwinCAT NCI 支持两种形式的插补指令：G 代码文件和 FeedTable。可以从数据流向来理解二者的区别：

执行 G 代码文件的方式：



执行 FeedTable 单条指令填充的方式：

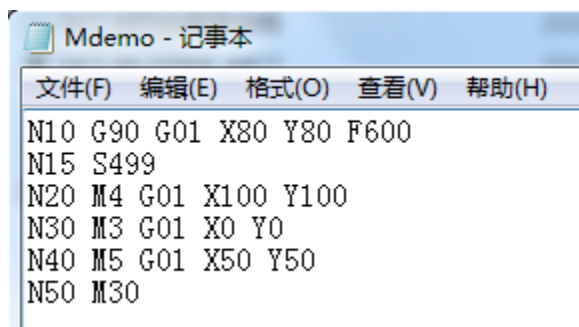


1.3.1 G 代码

G 代码文件是若干行 G 代码的集合，而每一行 G 代码就是一个动作命令。G 代码有一套规范，常用的是 G 指令和 M 指令。最简单的直线插补指令 G01，圆弧插补指令 G02/G03。

TwinCAT NCI 包含了 G 代码预读者，在执行行 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。

G 代码文件以.nc 为后缀名，可以用记事本编辑，一个最简单的 G 代码文件如下图所示：



```

Mdemo - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
N10 G90 G01 X80 Y80 F600
N15 S499
N20 M4 G01 X100 Y100
N30 M3 G01 X0 Y0
N40 M5 G01 X50 Y50
N50 M30
  
```

说明：新建 G 代码文件如果格式不方便写入，可以用示例 G 代文件来修改。

图中每行的第 1 列，比如 N10、N15，表示行号。这个不是必须的，但是行号可以增加可读性。行号还有其它辅助用途，这里就不细细展开了。

G90 是坐标切换指令，与 G91 是一对互锁指令。G90 表示切换到绝对坐标，G91 表示切换到相对坐标。如果 G 代码中从来没有出现 G90 或者 G91，默认使用绝对坐标。

S499 表示主轴速度 499 mm/min，在 NCI 中因为插补通道并没有主轴，实际上这个值是从 Proess Data 插补通道的过程变量 NCItOPLC 中的一个变量传递经 PLC 程序。PLC 程序再用它来控制主轴（PTP 轴）的速度。只有机加工的设备才有主轴速度，如果是激光切割或者其它简单走个路线的插补，就没有主轴这个概念，也就无须 S 指令了。

G01 这是最简单的直线插补指令，G01 X80 Y80 F600 的意思是说，下一个目标位置是 X80,Y80，进给速度是 600mm/min。有兴趣的读者可以试算一下，如果当前位置分别是 X0,Y0 及 X200,Y0 时，接下来 X 和 Y 轴的速度分别应该是多少。注意 F600 表示进给速度，在 G 代码中出现下一个 Fxxx 改变进给速度之前一直有效。

M3、M4、M5 这是自定义的 M 指令。当 M 指令与插补运动指令写在同一行时，需要在 NCI 通道参数中先设置好，是运动之前还是运动之后触发 M 函数，以及它的复位机制。假定 M4 的属性为 AM(After Motion)的 Handshake 型，M4 G01 X100 Y100，表示插补轴运动到 X100,Y100 的坐标位置后，M4 状态为 TRUE，插补通道的运动就暂停在这一行。这时 PLC 就得知 M4 的状态，根据 PLC 代码执行相关的逻辑，并复位 M4 函数。插补通道在 M4 函数复位后再继续下一行 G 代码的运动。

M30 G 代码规范约定的结束指令。

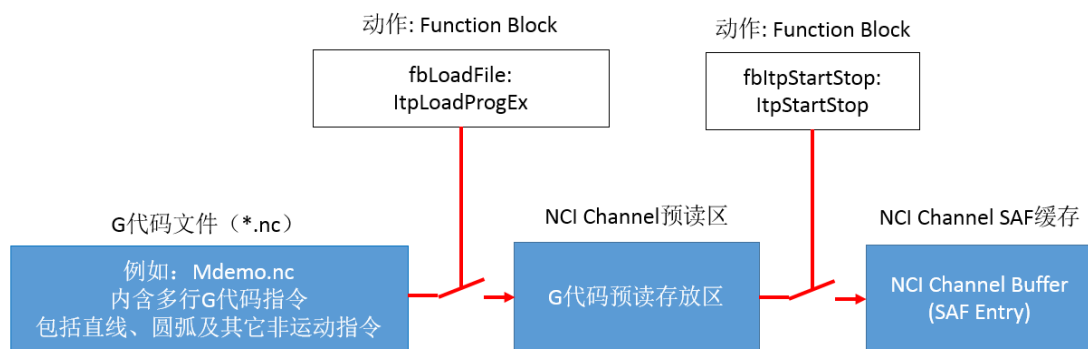
在 PLC 程序中，将 PTP 轴组合进 NCI 插补通道之后，装载 G 代码文件时只要确定 G

代码文件的路径和要装入的插补通道，然后就可以发启动命令让通道内各轴逐行执行 G 代码了。我们将在第 6 章详细介绍 G 代码的规则。

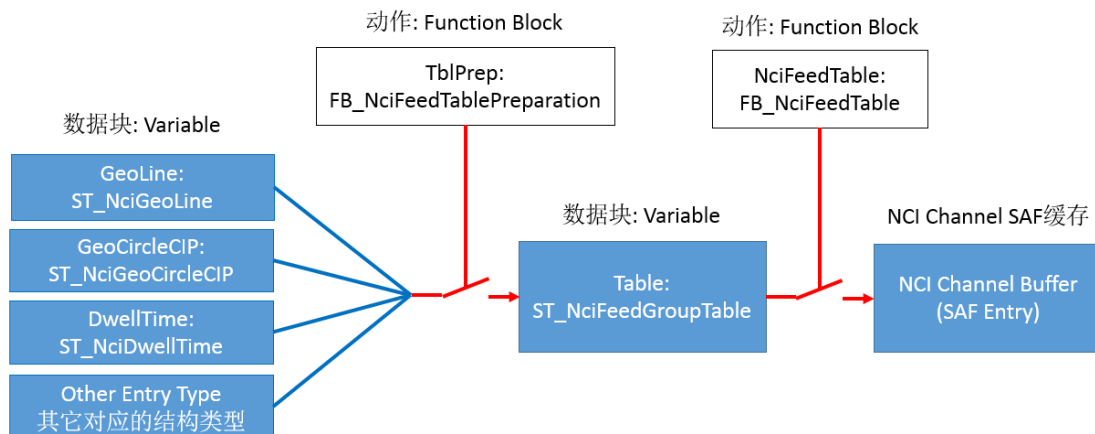
1.3.2 FeedTable

FeedTable 与 G 代码的区别是，插补指令不是写在 G 代码文件中，而是从 PLC 程序临时填入插补指令表。可以填入插补指令表的指令与 G 代码文件中的指令类型大致相当，也包括直线插补、圆弧插补、M 指令等等，但不再出现 G01、G02 等字样，而是以插补指令的类型枚举值来区分。

可以理解为，NCI 自带一个 G 代码解释器，装载 G 代码文件后，解释器就把它分解了一个一个的插补指令。



而使用 FeedTable 的时候，不是用 G 代码解释器，而是从 PLC 程序通过功能块 FB_NciFeedTable 往 NCI 的执行区（SAF Entry）里填充指令。对于熟悉 PTP 中的 TwinCAT NC Fifo 的用户，这点比较容易理解。



1.4 M 函数：插补运动与逻辑动作的协调

M 指令是在 G 代码文件执行过程中需要触发的开关状态。这个开关状态可以是自恢复

的，也可以是等待 PLC 确认恢复的。如果是自恢复的，插补运动在 M 代码行只是通知给 PLC 的 M 函数状态为 True，动作继续执行。如果是等待 PLC 确认才能恢复的，插补运动在 M 代码行通知给 PLC 的 M 函数状态为 True，插补运动就会停下来，PLC 收到 M 函数为 True 之后执行相应的动作，动作完成后复位 M 函数，才继续下一行 M 指令。

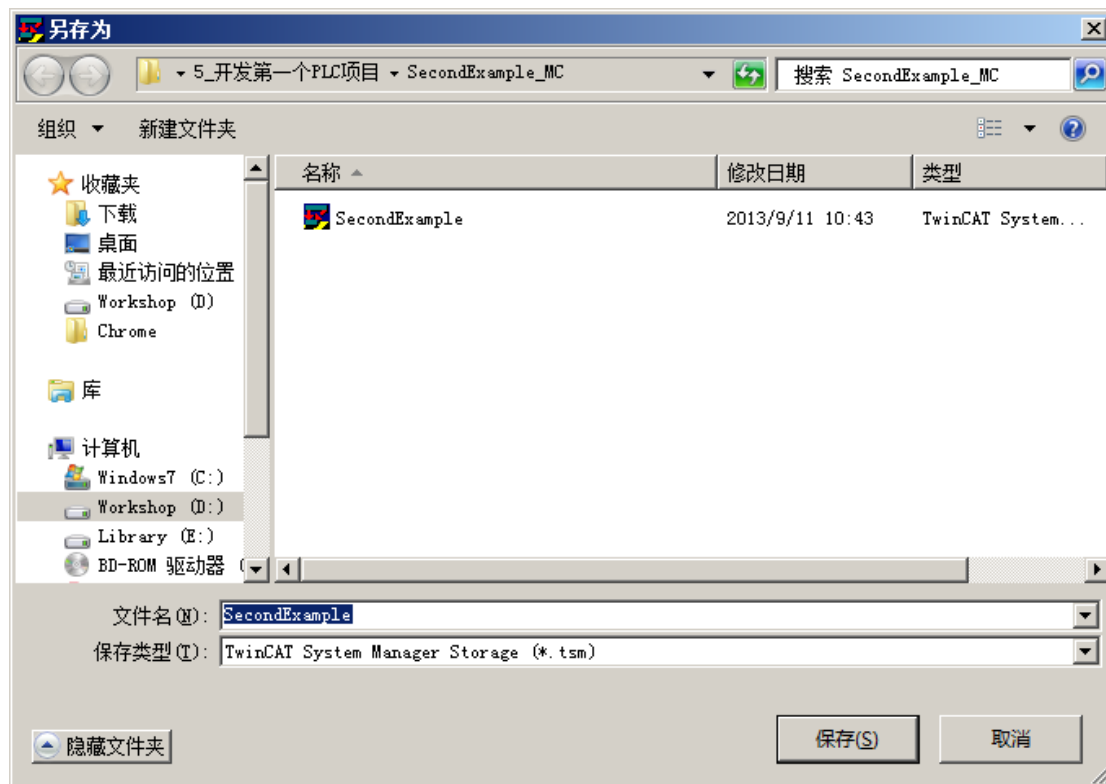
NCI 通道用到哪些 M 函数，分别是什么类型，需要在 NCI 配置文件中事先定义，否则系统默认为需要等待的 M 函数。如果是 MFast 型的，即自恢复型，那么它在什么时间恢复，也是可以设置的。

注意：在国际标准中，有一些 M 函数是有固定用途的，比如 M30 用于 G 代码结束。

1.5 R 参数：NC I 通道与 PLC 的浮点数接口

在 G 代码中，表达一个插补运动指令时，在给终点坐标位置或者进给速度的时候，可以用常量，也可以用变量。如果使用变量，并且希望这个变量可以在 PLC 程序中访问，就可以使用 R 参数。

每个通道都有 R0-R44 共 45 个 R 参数。R 参数都是实数型的，可以在 G 代码中赋值，也可以在 PLC 程序中通过功能块写入或者读取。所以可以把 R 参数作为 NC I 通道与 PLC 程序的浮点数接口，只是它不是每个 PLC 周期刷新的。



2 在 System Manager 中测试 NCI 功能

\配套例程\第 2 章 在 System Manager 中测试 NCI 功能

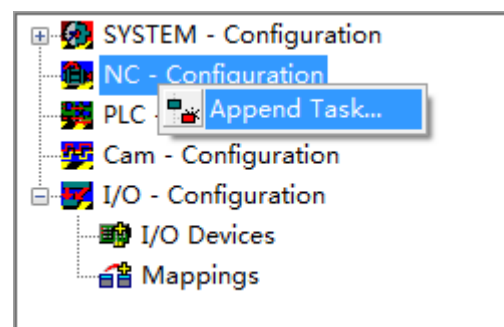
下册补充 NCI ▶ 配套例程 ▶ 第2章 在System Manager中测试NCI功能

名称	修改日期	类型	大小
Mdemo	2016/10/23 15:49	NC 文件	1 KB
NCL_Test	2016/10/23 17:59	TSM 文件	178 KB

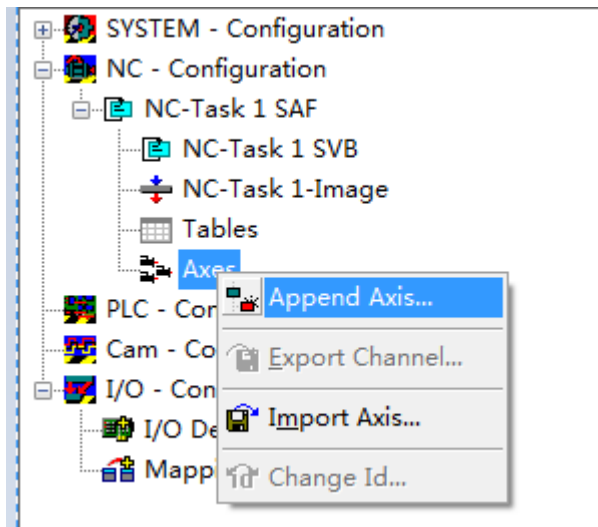
2.1 创建 TwinCAT NC PTP 轴和 NCI 通道

2.1.1 创建 NC 任务和 PTP 轴

添加 NC 任务

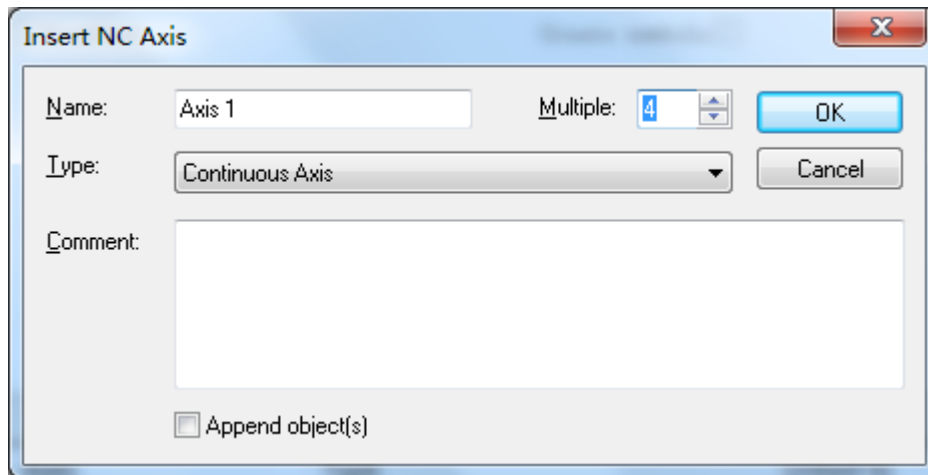


添加 PTP 轴



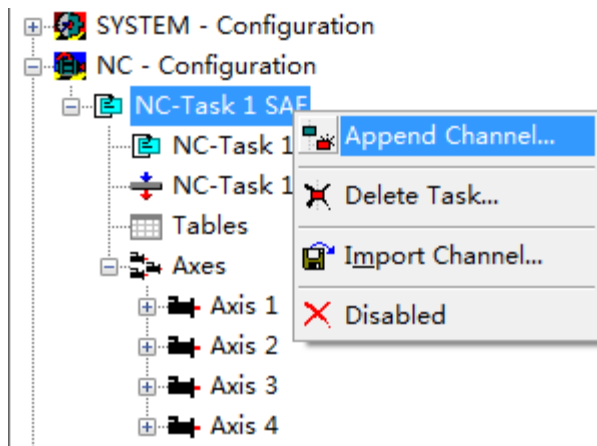
选择轴的类型

使用默认设置, 不做任何硬件配置和关联, 就创建了一个可以在任何 PC 上仿真运行的虚轴。

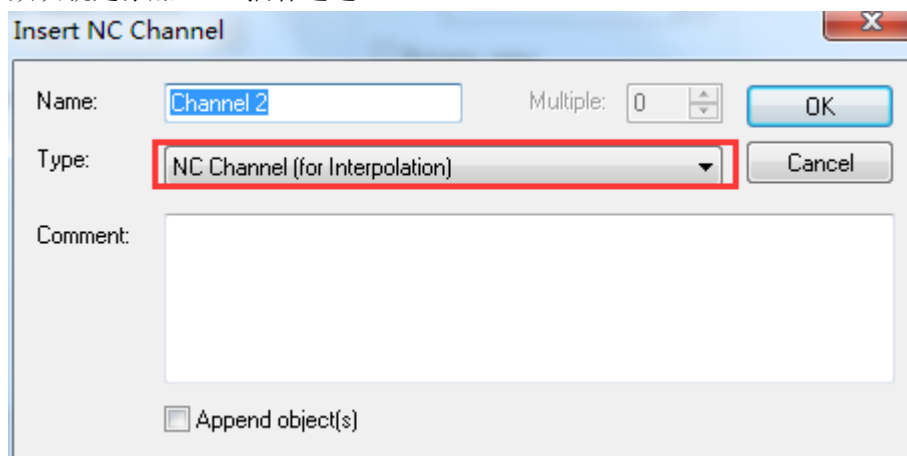


2.1.2 创建 NC I 通道

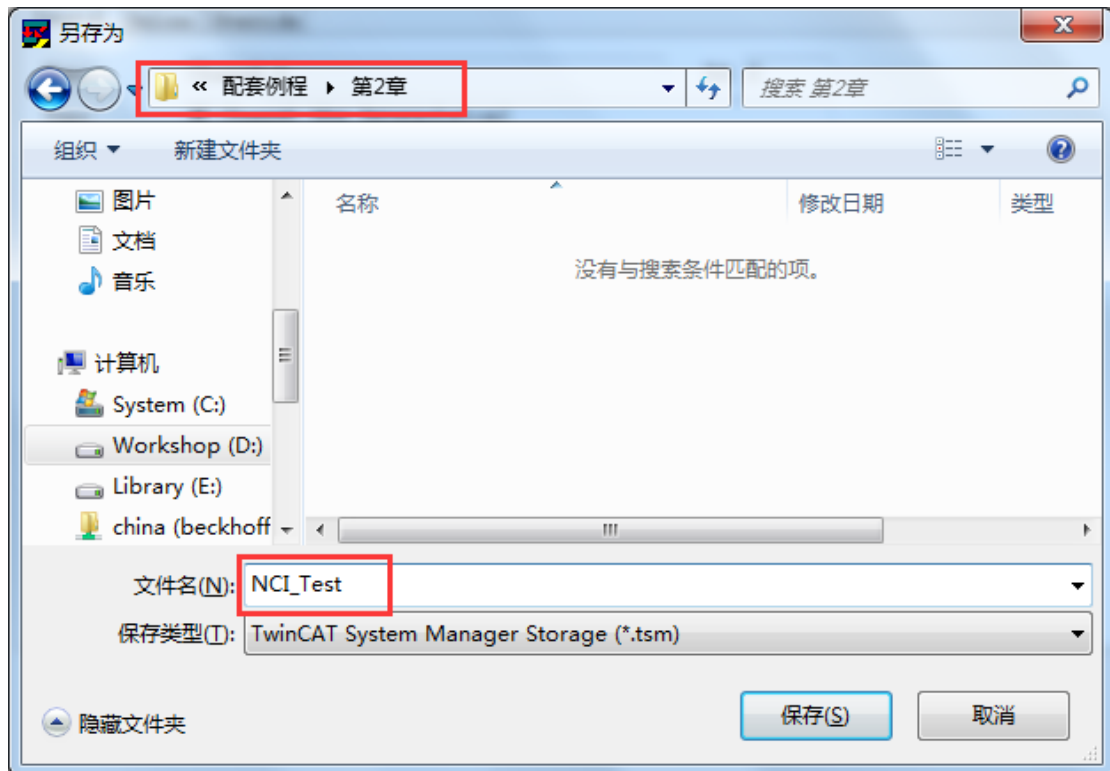
已经有一个 NC 任务的项目里, 再次添加 Channel:



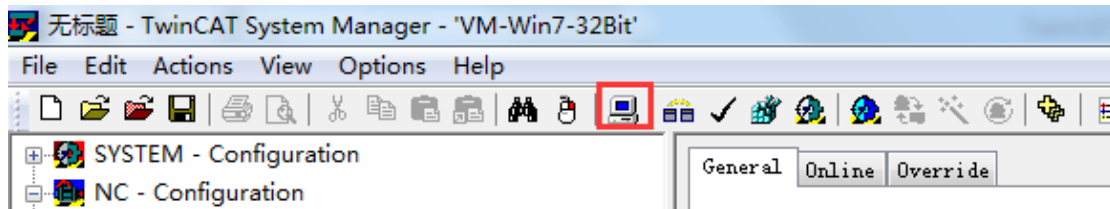
默认就是添加 NC I 插补通道：



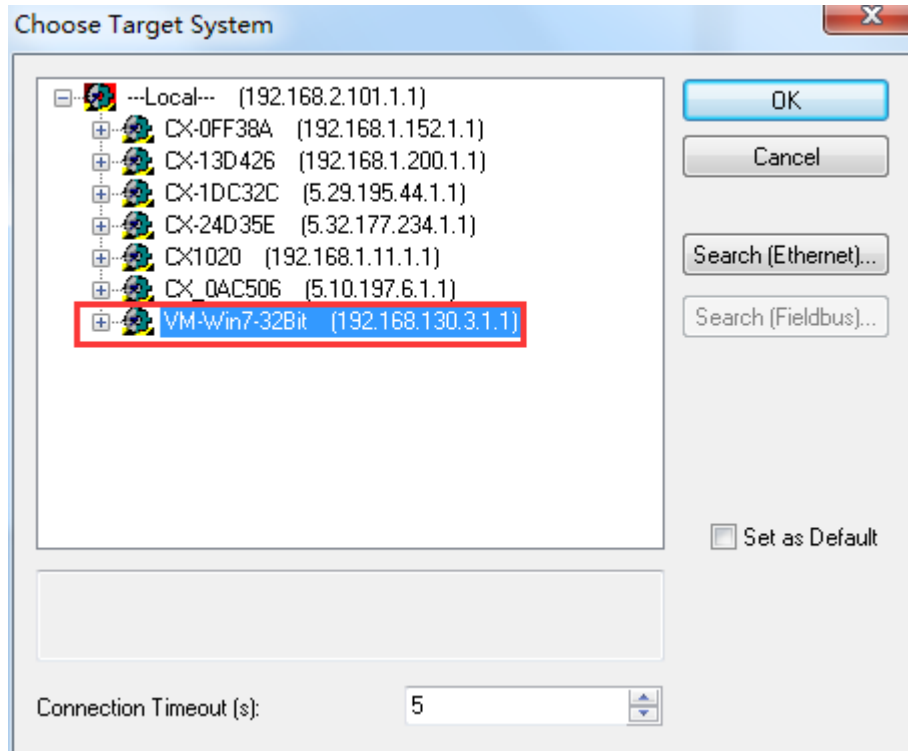
2.1.3 存盘和激活配置



选择目标系统

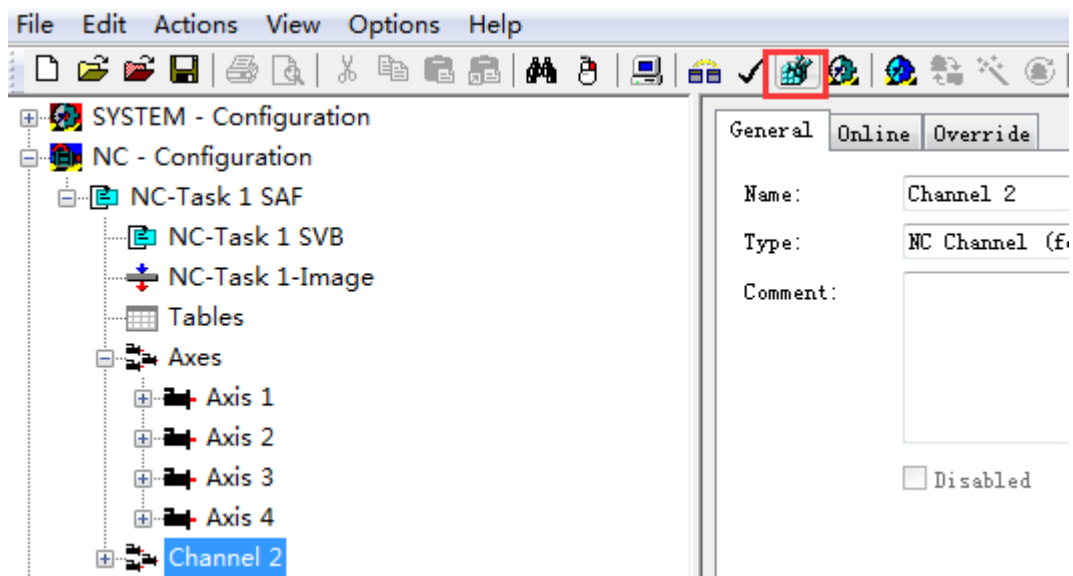


点击上图中的目标选择图标，就会弹出开发 PC 的路由表：



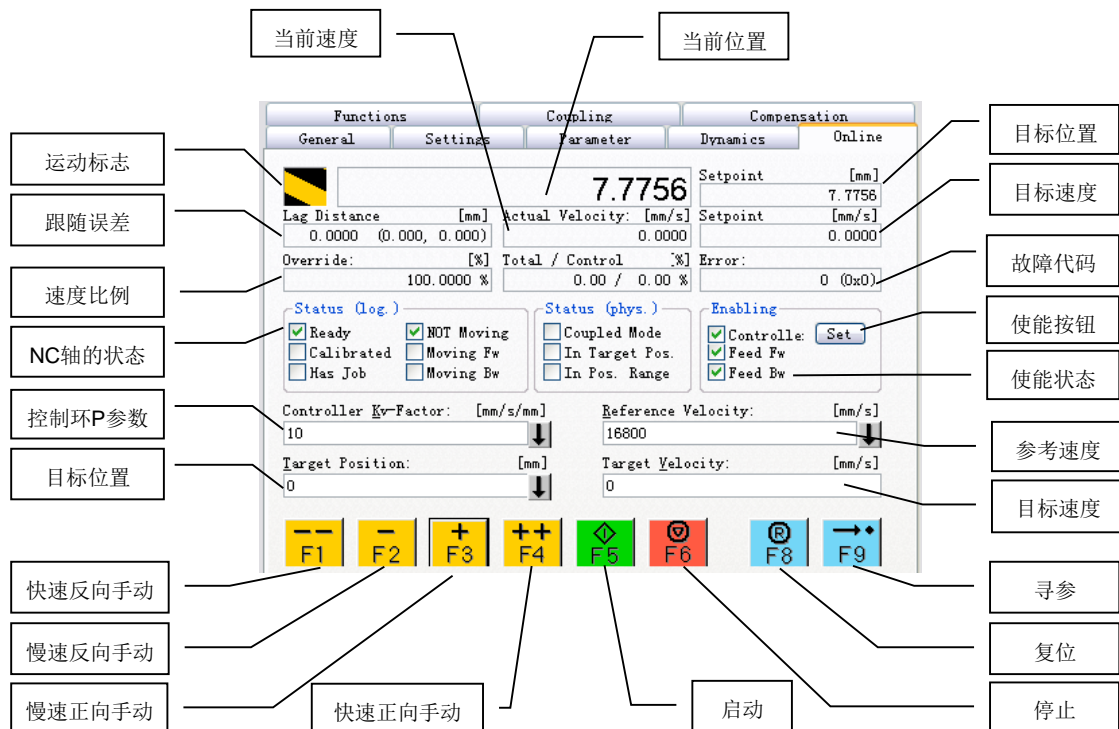
如果目标机器不在上图的列表就，就需要点击 Search 按钮进行添加。
对于 PTP 的用户这个步骤已经非常熟悉。此处不再重述。

激活配置：



这样一个带 NCI 插补通道的 TwinCAT 运动控制项目就建立起来了。以后的章节就是讲述分别从 PLC 程序或者开发环境控制 NCI 插补通道按 G 代码文件或者插补指令表执行规定动作。

2.1.4 PTP 轴调试界面



这里的 F1-F9，与 PC 键盘的功能键 F1-F9 是等效的，用户可以用鼠标点击界面上的按钮，也可以直接按下键盘上的对应的功能键。

这里的速度、位置、跟随误差等，默认都是以“mm”为单位。如果关联了伺服驱动器，就表示这些参数都是根据编码器的脉冲当量“Scaling Factor”换算之后的值。

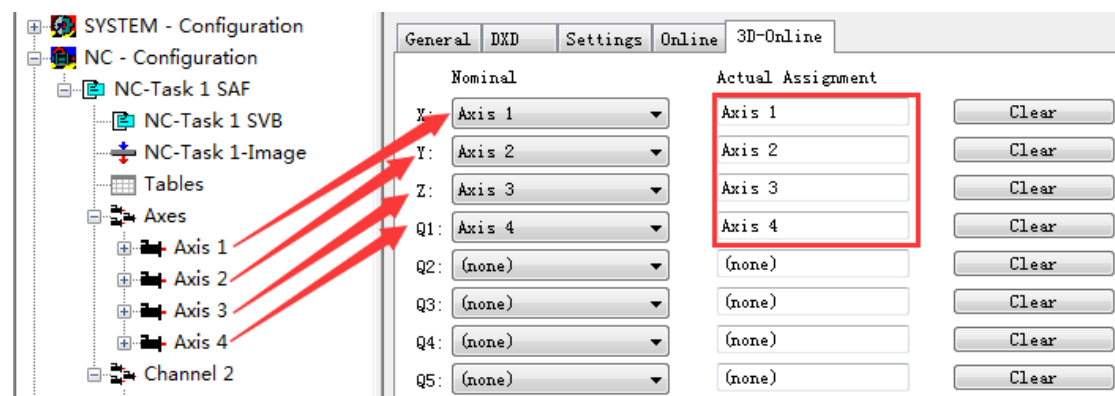
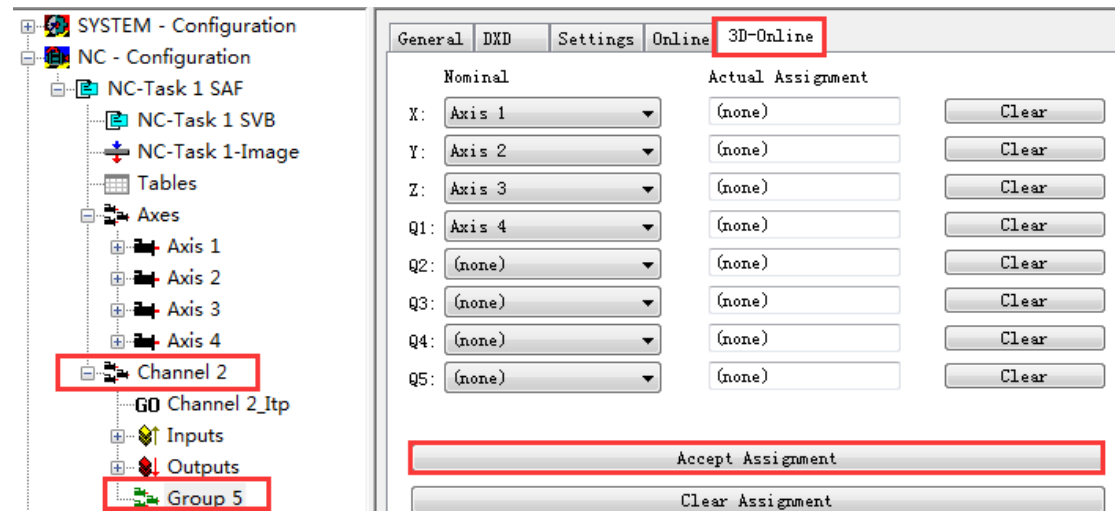
第 1 步：让轴使能。

第 2 步：点击 F1-F4 按钮，试试点动功能。

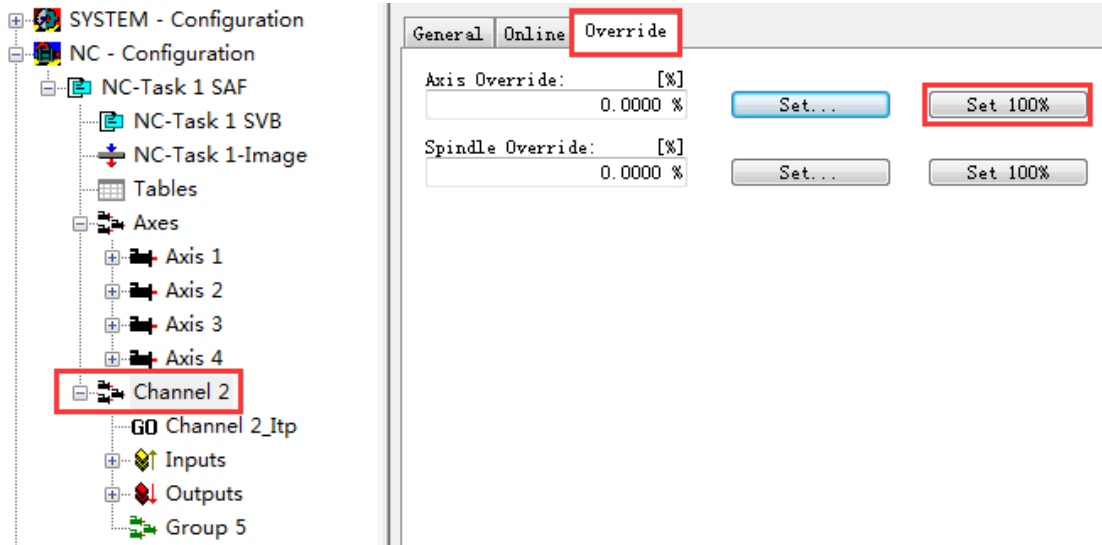
其它测试不再重述，鉴于 NC I 的用户必须先熟悉 PTP 操作，所以在此假设用户已经熟悉 PTP 调试界面和步骤。

2.2 NCI 通道调试界面

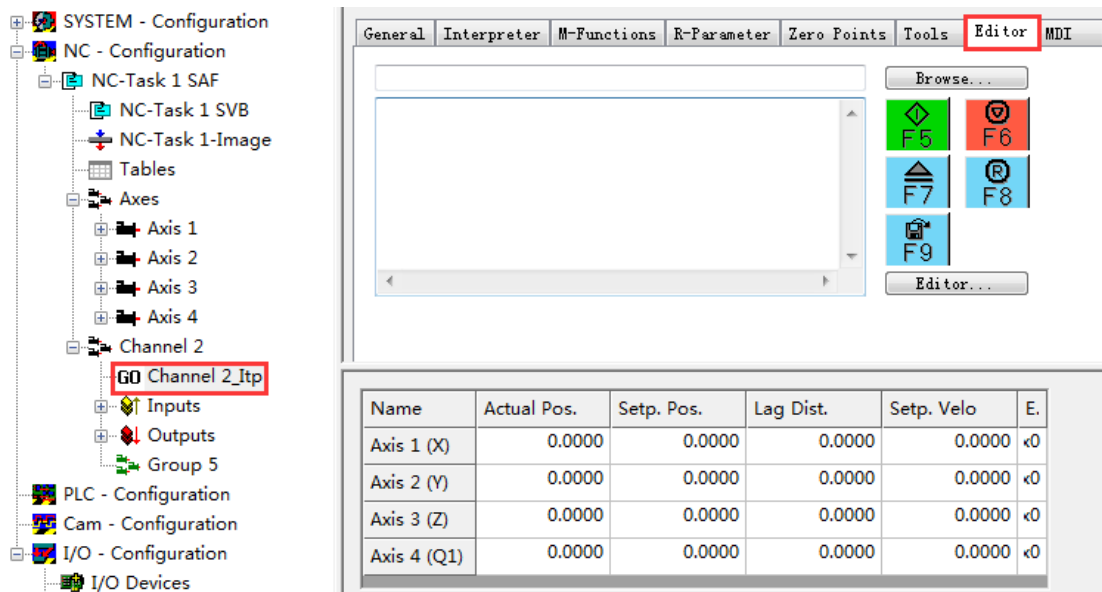
2.2.1 配置 NC I 通道的插补轴所对应的 PTP 轴

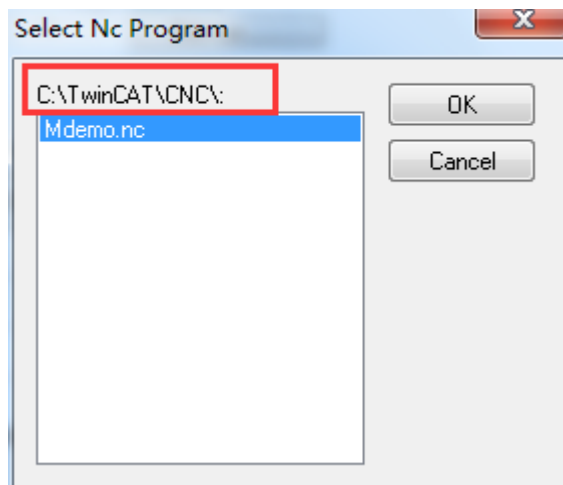
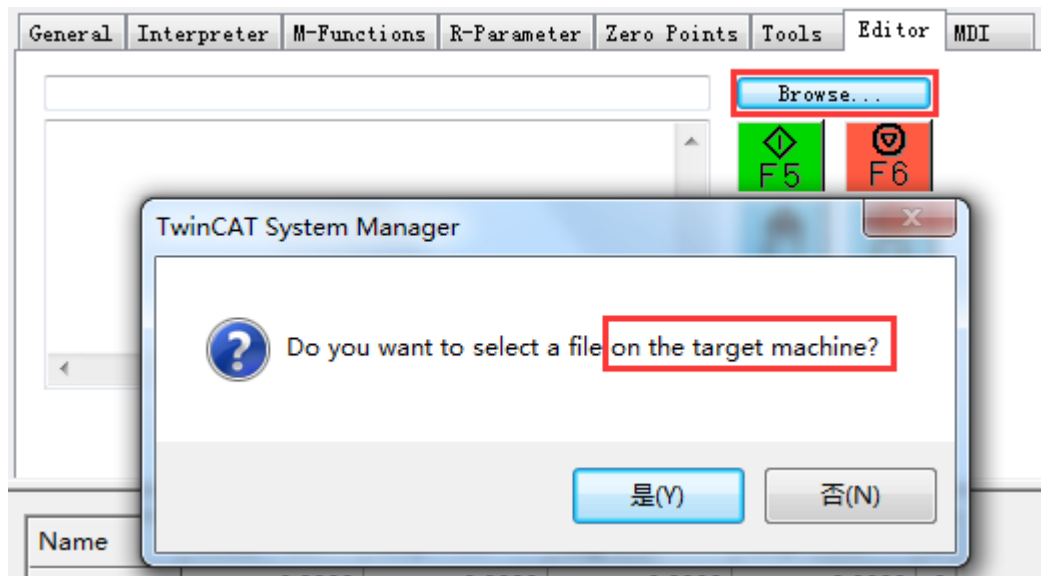


2.2.2 配置 NC I 通道的输出倍率



2.2.3 定位和编辑 G 代码文件





Target: C:\TwinCAT\CNC\demo.nc

```

N10 G01 X0 Y0 F1200
N20 G01 X0 Y100
N30 G01 X100 Y100
N40 G01 X100 Y0
N50 G01 X0 Y0
N50 M30

```

Name	Actual Pos.	Setp. Pos.	Lag Dist.	Setp. Velo	E.
Axis 1 (X)	0.0000	0.0000	0.0000	0.0000	<0
Axis 2 (Y)	0.0000	0.0000	0.0000	0.0000	<0
Axis 3 (Z)	0.0000	0.0000	0.0000	0.0000	<0
Axis 4 (Q1)	0.0000	0.0000	0.0000	0.0000	<0

NC Editor

```

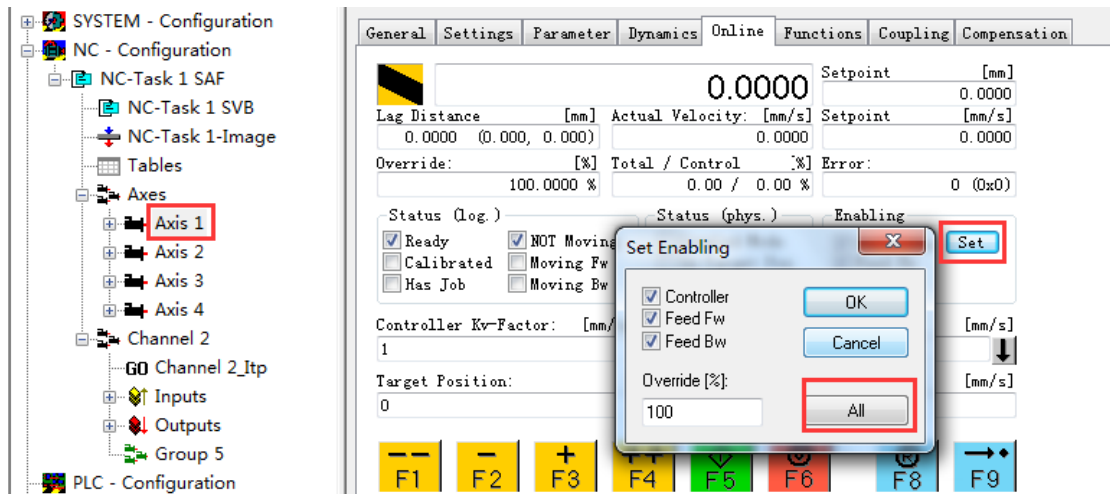
N10 G01 X0 Y0 F900
N20 G01 X0 Y100
N30 G01 X100 Y100
N40 G01 X100 Y0
N50 G01 X0 Y0
N50 M30

```

OK
Cancel

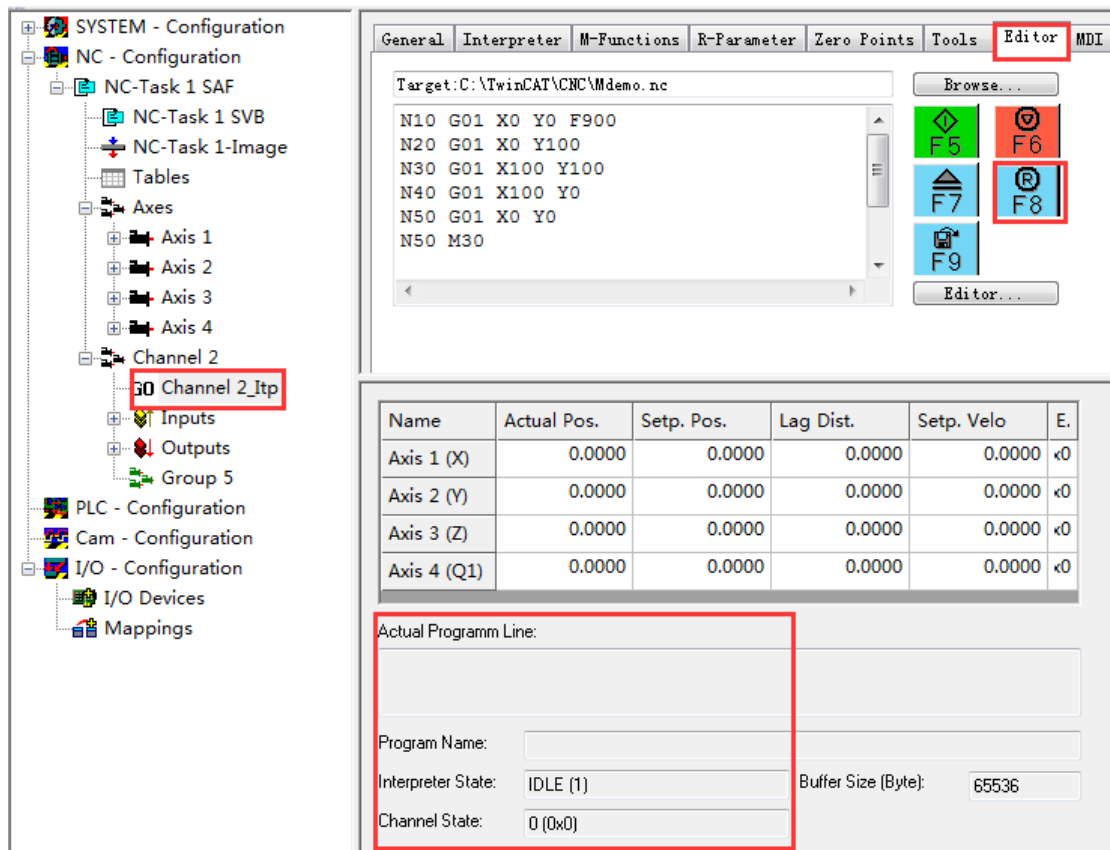
2.2.4 测试 G 代码文件

先使能各个 PTP 轴。



依次点击 F8(复位)—F7(装载文件)—F5(运行)，结果如下：

F8(复位)：程序行：空，插补状态：Idle，通道状态：0



F7(装载文件)：

Target: C:\TwinCAT\CNC\Mdemo.nc

```
N10 G01 X0 Y0 F900
N20 G01 X0 Y100
N30 G01 X100 Y100
N40 G01 X100 Y0
N50 G01 X0 Y0
N50 M30
```

Name	Actual Pos.	Setp. Pos.	Lag Dist.	Setp. Velo	E.
Axis 1 (X)	0.0000	0.0000	0.0000	0.0000	κ0
Axis 2 (Y)	0.0000	0.0000	0.0000	0.0000	κ0
Axis 3 (Z)	0.0000	0.0000	0.0000	0.0000	κ0
Axis 4 (Q1)	0.0000	0.0000	0.0000	0.0000	κ0

Actual Programm Line:

Program Name: Mdemo.nc

Interpreter State: READY (2) Buffer Size (Byte): 65536

Channel State: 0 (0x0)

F5(运行)

General Interpreter M-Functions R-Parameter Zero Points Tools Editor MDI

Target: C:\TwinCAT\CNC\Mdemo.nc

```

N10 G01 X0 Y0 F900
N20 G01 X0 Y100
N30 G01 X100 Y100
N40 G01 X100 Y0
N50 G01 X0 Y0
N50 M30

```

Name	Actual Pos.	Setp. Pos.	Lag Dist.	Setp. Velo	E.
Axis 1 (X)	0.0000	0.0000	0.0000	0.0000	κ0
Axis 2 (Y)	19.1942	19.1942	0.0000	14.9940	κ0
Axis 3 (Z)	0.0000	0.0000	0.0000	0.0000	κ0
Axis 4 (Q1)	0.0000	0.0000	0.0000	0.0000	κ0

Actual Programm Line:
N20 G01 X0 Y100

Program Name: Mdemo.nc

Interpreter State: **RUNNING (5)** Buffer Size (Byte): 65536

Channel State: 0 (0x0)

开始运行后，插补状态 Interpreter State 会显示为 Running(5)。

The screenshot shows the MDI (Manual Data Input) window in TwinCAT. The top menu bar includes General, Interpreter, M-Functions, R-Parameter, Zero Points, Tools, Editor, and MDI. The main area displays a CNC program with the following lines:

```

Target: C:\TwinCAT\CNC\Mdemo.nc
N10 G01 X0 Y0 F900
N20 G01 X0 Y100
N30 G01 X100 Y100
N40 G01 X100 Y0
N50 G01 X0 Y0
N50 M30

```

Below the program is a table showing the status of the axes:

Name	Actual Pos.	Setp. Pos.	Lag Dist.	Setp. Velo	Err...
Axis 1 (X)	100.0000	100.0000	0.0000	0.0000	0x0
Axis 2 (Y)	20.7367	20.7367	0.0000	-14.9940	0x0
Axis 3 (Z)	0.0000	0.0000	0.0000	0.0000	0x0
Axis 4 (Q1)	0.0000	0.0000	0.0000	0.0000	0x0

Below the table, the "Actual Programm Line:" section shows the current line being executed, which is highlighted with a red box:

```

N20 G01 X0 Y100
N30 G01 X100 Y100
N40 G01 X100 Y0

```

At the bottom, there are fields for Program Name (Mdemo.nc), Interpreter State (RUNNING (5)), Buffer Size (Byte) (65536), and Channel State (0 (0x0)).

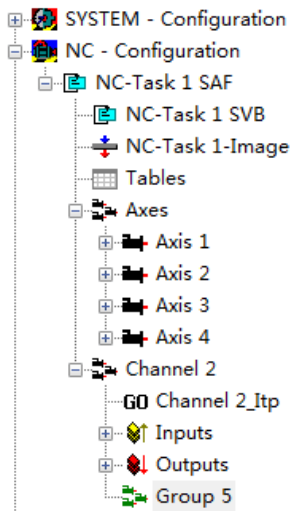
最新执行的代码行会显示在“Actual Programm Line”区域。

待所有代码都执行完毕，插补轴不再动作，重新依次点击 F8(复位)、F7(装载)、F5(运行)，插补轴就可以继续动作了。

2.2.5 解散插补通道，恢复 PTP 轴

对比 Axis 和 NCI Channel 的 Online 页面。

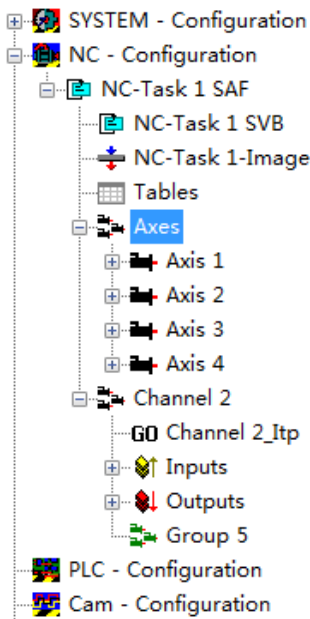
在 PTP 轴分配给插补通道时，在 Axes 的 Online 页面不显示分配到 NCI 通道的轴信息。而 Channel 的 Online 页面显示各个插补轴信息。



General DXD Settings Online **3D-Online**

	Nominal	Actual Assignment	
X:	Axis 1	Axis 1	Clear
Y:	Axis 2	Axis 2	Clear
Z:	Axis 3	Axis 3	Clear
Q1:	(none)	Axis 4	Clear
Q2:	(none)	(none)	Clear
Q3:	(none)	(none)	Clear
Q4:	(none)	(none)	Clear
Q5:	(none)	(none)	Clear

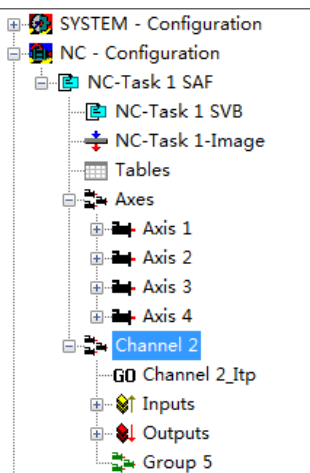
Accept Assignment
 Clear Assignment



General Online

Name	Actual Pos.	Setp. Pos.

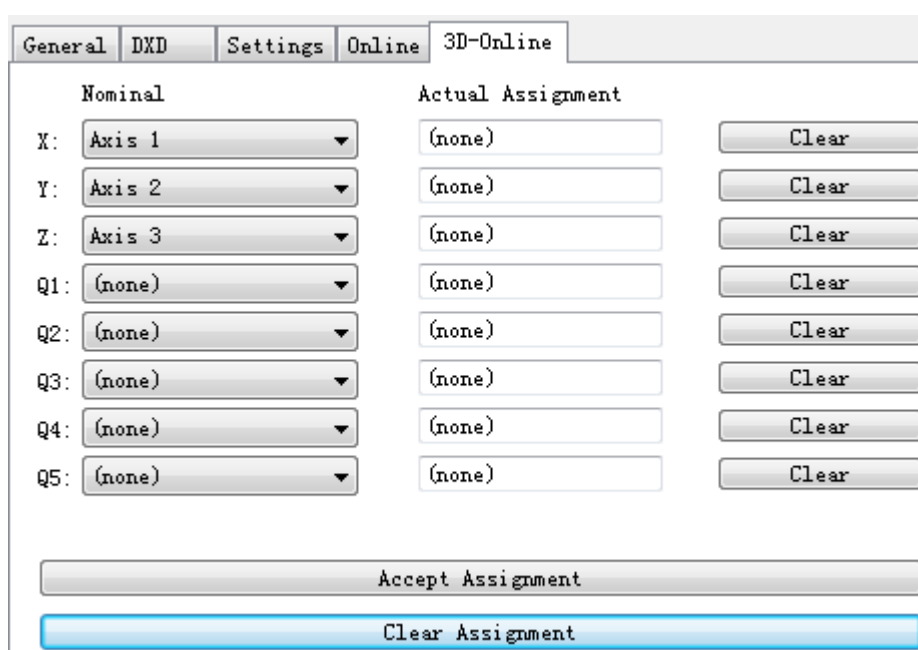
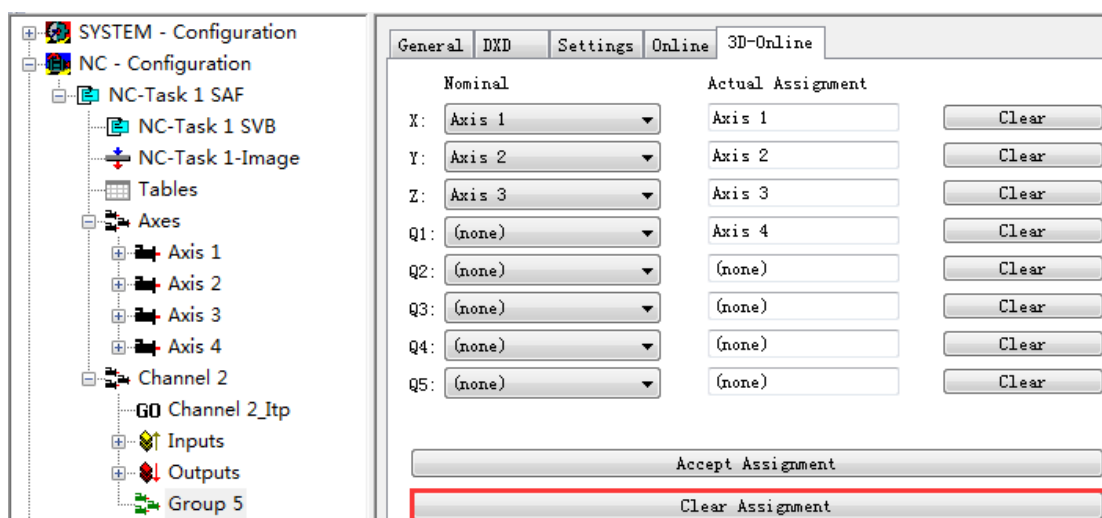
-- F1 - F2 + F3 ++ F4 ® F8



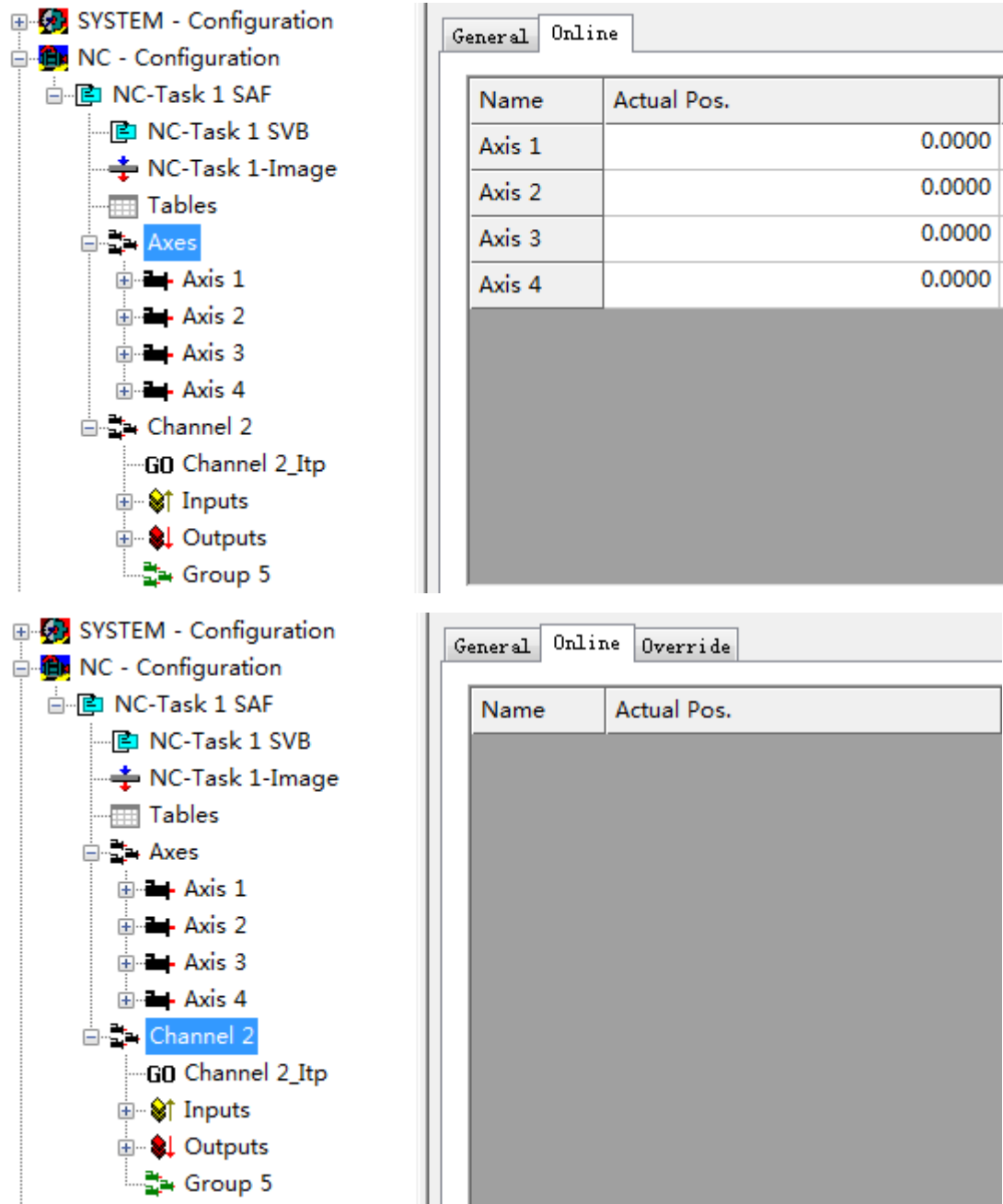
General Online **Override**

Name	Actual Pos.	Setp. Pos.
Axis 1 (X)	0.0000	0.0000
Axis 2 (Y)	0.0000	0.0000
Axis 3 (Z)	0.0000	0.0000
Axis 4 (Q...	0.0000	0.0000

点击 Clear，解散插补通道，XYZ 等插补轴还原成为 PTP 轴。

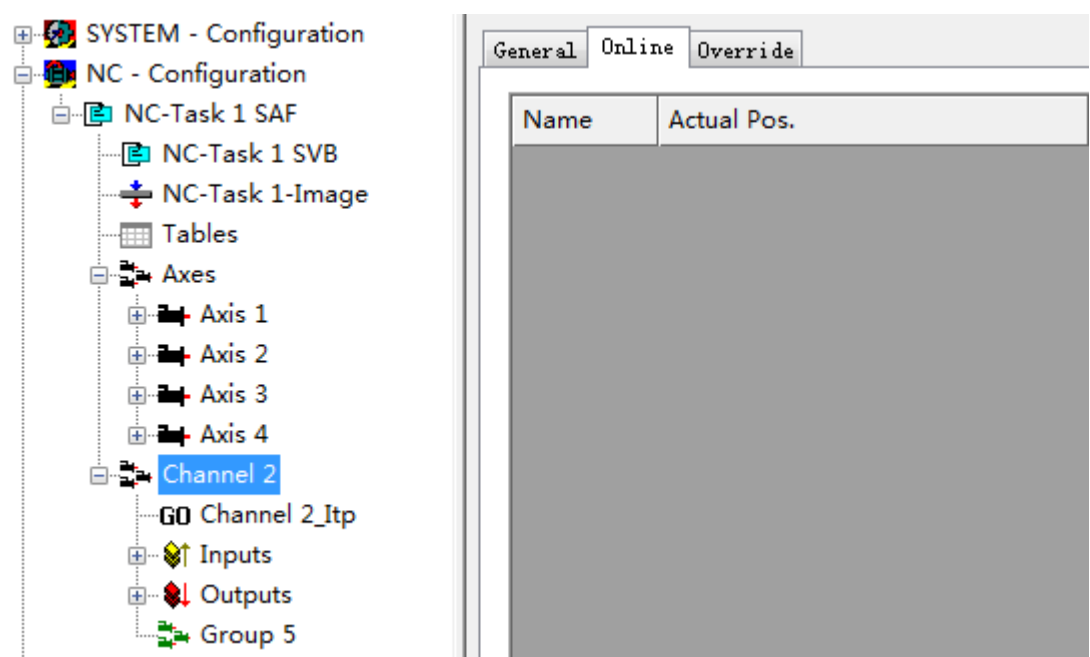


在 NCI 通道解散后，Axis1、Axis2、Axis3 恢复为 PTP 轴，在 Axes 的 Online 页面各轴信息，而 Channel 的 Online 页面就没有信息显示。



The screenshot shows the TwinCAT configuration tree on the left and the 'Online' tab on the right. The tree is expanded to show 'Axis 1' under the 'Axes' folder. The 'Online' tab displays a table with the following data:

Name	Actual Pos.
Axis 1	0.0000
Axis 2	0.0000
Axis 3	0.0000
Axis 4	0.0000



The second screenshot shows the same configuration tree, but with 'Channel 2' selected under the 'Channel 2' folder. The 'Override' tab is currently empty.

特别提示：除了轴的使能以前，插补轴不能执行任何 PTP 指令，包括 MC_Reset，MC_Home、MC_SetPosition 等指令。也不能作为 PTP 电了齿轮或者凸轮的主轴或者从轴。（主轴不知是否可以，从轴肯定不行）

2.3 NCI 通道的运动参数设置

2.3.1 插补曲线转折的分类

通常来讲，从一段曲线到下一段曲线的连接不是绝对平滑的。所以在两段曲线连接的地方必须降低速度以避免加速度跳变引起的冲击。为此把两段曲线的转折方式按几何特性分为 C0, C1, C2:

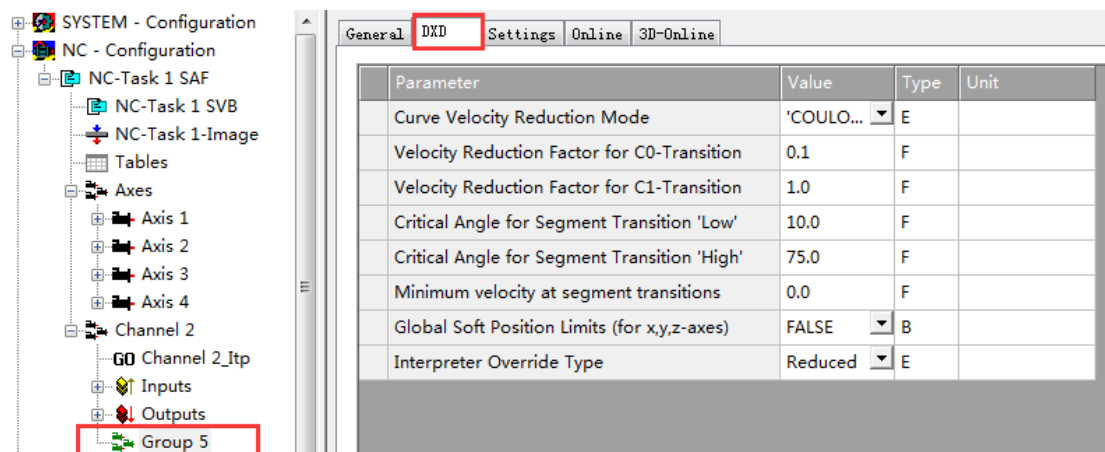
C0 转折: 位置连续速度不连续。两段几何曲线的连接处是一个拐点。

C1 转折: 速度连续加速度不连续。比如体育馆跑道的直线与半圆的转折点，切线是连续的，但是二次求导后的曲线是不连续的。如果均速经过这段曲线，必须有加速度阶跃。

C2 转折: 加速度连续加加速度不连续。如果均速经过这段曲线，加速度连续，不会产生冲击。超过 2 阶连续的曲线更加不会冲击。

2.3.2 参数详解

本节作为字典可查，但与实际项目要求的结合，请参考第 7 章“插补运动中的常见需求”



Parameter	Value	Type	Unit
Curve Velocity Reduction Mode	'COULO...	E	
Velocity Reduction Factor for C0-Transition	0.1	F	
Velocity Reduction Factor for C1-Transition	1.0	F	
Critical Angle for Segment Transition 'Low'	10.0	F	
Critical Angle for Segment Transition 'High'	75.0	F	
Minimum velocity at segment transitions	0.0	F	
Global Soft Position Limits (for x,y,z-axes)	FALSE	B	
Interpreter Override Type	Reduced	E	

在相邻曲线平滑过渡时的减速模式，由 Curve Velocity Reduction Mode 控制。

插补轴的 PTP 软限位设置是否生效，由 Global Soft Position Limits 控制。

曲线过渡时允许降到的最低速度，由 Minimum Velocity at Segment transitions，单位是运动指令速度的百分比。

在 DXD 属性页面可以设置 NCI 组的参数：

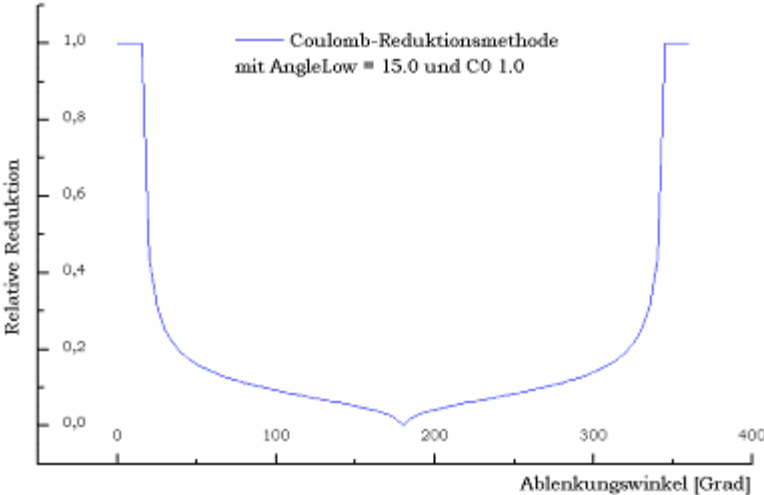
- Curve velocity reduction method (曲线减速方式)

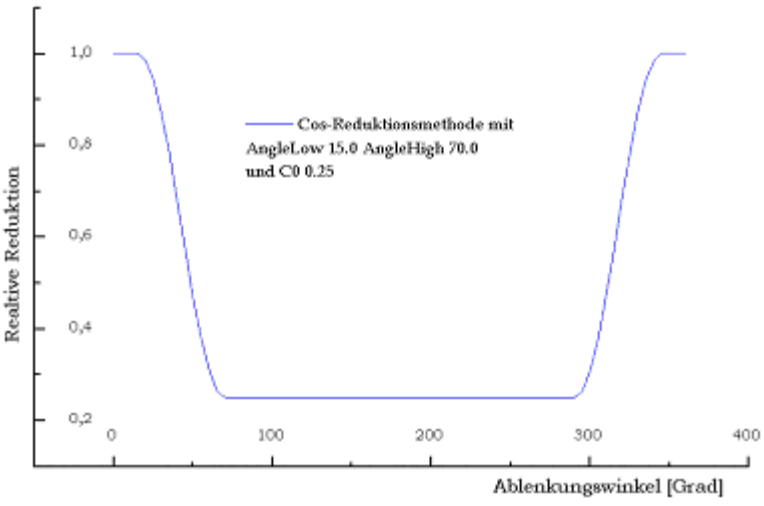
这个选项仅在 C0 转折时生效，即前后两段曲线有一个硬的拐角。C0 转折指切线斜率不连续的点，又称拐点。

可选项包括：

- 0 Coulomb (库仑减速)
- 1 Cosine (余弦减速)
- 2 VeloJump (速度跃变)

3 Deviation Angle (偏差角, 此功能未发布)

Method	描述
Coulomb 库仑减速	<p>库仑减速是一个类似库仑散射的动态过程。</p> <p>(百度“库仑散射”: 带电粒子入射方向与散射方向的夹角称为散射角 θ。库仑散射的特点是散射粒子按角度的分布, 与 $\sin^4(\theta/2)$成反比, 即散射粒子多集中在前向小角度区域。)</p> <p>前后两段曲线在转折点的切线方向的夹角, 称为偏转角 φ。如果前后两段曲线完全反向, $\varphi=180$, 速度就会减到 0。库仑散射的特点是偏转角越小, 减速曲线越陡。可以设置减速段的起始角度 φ_{low}。</p> <p>库仑减速的因素包括:</p> <p>C0 factor, 界于 0.0 到 1.0 , 即 0%到 100%。</p> <p>φ_{low}, 界于 0.0 到 180.0 度</p> <p>减速原则 (假定指令进给速度为 V_k, 减速后的设定速度 V_{res}):</p> <p>$\varphi < \varphi_{low}$: $V_{res} = V_k$</p> <p>$\varphi_{low} < \varphi < 180$: $V_{res} =$ 根据 φ 值按库仑曲线从 V_k 减到 $V_k * (1 - C0 \text{ factor})$</p> <p>如图所示:</p>  <p>上图是比较典型的设置, $\varphi_{low}=15$, C0 factor =1.0, 表示转角在 15 度以内都不减速, 而在反向时, 速度会减少 100%, 即速度降为 0。</p> <p>库仑减速的特点是: 偏转角一旦达到需要减速的程度, 速度就大幅下降。从图中可以看出, 15 度开始减速。如果偏转角继续加大, 到在 30 到 170 度就以一个很低的速度运行, 下降幅度很缓慢了, 超过 170 度即接近反向, 又开始急速减到 0。</p>

<p>Cosine 余弦减速</p>	<p>COS 减速，这是一个纯几何减速过程： 因素包括： C0 factor，介于 0.0 到 1.0 ，即 0% 到 100%。 ϕ_{low}，介于 0.0 到 180.0 度 ϕ_{high}，介于 0.0 到 180.0 度，且大于 ϕ_{low}</p> <p>减速原则（假定指令进给速度为 V_k，减速后的设定速度 V_{res}）： $\phi < \phi_{low}$: $V_{res} = V_k$ $\phi_{low} < \phi < \phi_{high}$: $V_{res} =$ 根据 ϕ 值按 Cosine 曲线从 V_k 减到 $V_k * C0 \text{ factor}$ $\phi_{high} < \phi$: $V_{res} = V_k * C0 \text{ factor}$。</p>  <p>上图中，C0 factor = 0.25（即 25%），$\phi_{low}=15$ 度，$\phi_{high}=75$ 度</p> <p>与库仑曲线相反，库仑减速是先快再慢再快，余弦减速是先慢后快再慢，并且余弦减速有底线的，减到最小值 $V_k * C0 \text{ factor}$ 就不再减了。而库仑减速可以减到 0 速。</p>
<p>VeloJump 速度跃变</p>	<p>这是决定 C0 转折速度的几何过程，这个过程按要求降低轨迹速度，以使速度阶跃在设定的范围以内。允许的速度阶跃最大值计算公式如下：</p> $DV = \text{VeloJump factor} * \min(A+, A-) * DT$ <p>一个 NC 周期的速度差 = 速度跃变因子 * 加速度和减速度的最小值 * NC 周期</p> <p>VeloJump factor 在插补轴的 NC PTP 轴参数中设置，如图 Parameter NCI Parameter Velo Jump Factor</p>

-	NCI Parameter:	
	Rapid Traverse Velocity (G0)	2000.0
	Velo Jump Factor	100.0
	Tolerance ball auxiliary axis	0.0
	Max. position deviation, aux. axis	0.0

从 Beckhoff Information System 中查到该参数的解释如下：

轴参数 Offset 0x0106	读 写	单位: 1	范围: 0.0-10 ⁶	NCI 动态减速时允许的速度跃变 最大值的作用因子: DV = factor * min(A+, A-) * DT	默认值: 0
-------------------------	--------	-------	----------------------------	---	-----------

对比 PTP 运动，一个 NC 周期允许的最大速度跃变就是加速度乘以时间，即 $DV = Acc * DT$ ，即 $VeloJump\ Factor = 1$ 。

NCI 中可以在 PTP 模式的 DV 基础上，迭加一个作用因子。小于 1 就是限制它的速度跃变，大于 1 就是放大它的速度跃变。对于放大速度跃变的情况，还受限于其它条件：电机功率、伺服驱动中的加速度限制、通道内其它轴的允许速度跃变等。所以通常是设置为小于 1，以限制速度跃变，牺牲轨迹精度，使运动更加平稳。

说明：上述 3 种方式中，速度跃变都会受限于轴的 NCI 参数 VeloJump Factor 的限制，只是库仑减速和余弦减速时，除受这个限制之外，还对何时开始减速、减到多少做出了规划，实际的 NCI 给定速差取规划速差和 VeloJump Factor 计算出的速差之最小值。而曲线过渡时如果选择 VeloJump 方式减速，则仅受限于根据 VeloJump Factor 计算出的速差，在 3 种减速模式中，这种减速最少，运动最猛。

DXD 页面的其它选项：

Parameter	Value	Type	Unit
Curve Velocity Reduction Mode	'COULO...	E	
Velocity Reduction Factor for C0-Transition	0.1	F	
Velocity Reduction Factor for C1-Transition	1.0	F	
Critical Angle for Segment Transition 'Low'	10.0	F	
Critical Angle for Segment Transition 'High'	75.0	F	
Minimum velocity at segment transitions	0.0	F	
Global Soft Position Limits (for x,y,z-axes)	FALSE	B	
Interpreter Override Type	Reduced	E	

- Velocity reduction factor C0 transition

C0 转折（速度不连续）的减速因子，其作用依赖于减速方式。C0 的取值范围 [0.0, 1] 在 Coulumb 减速模式时，表示减速的幅度。0.1 表示 100%。在 COS 减速模式下，表示减速的最小值。0.1 表示 100%。

- Velocity reduction factor C1 transition

C1 转折（加速度不连续）的减速因子，其作用依赖于减速方式。C1 的取值范围 [0.0, 1] 首先假定 V_Link 为前后两段进给速度中较小的那个值。V_link = min(V_in, V_out)

曲线过渡时加速度变化的绝对值 AccJump 计算取决于在速度 V_link 时几何类型 G_in 、G_out 和 G_in 、G_out 所连接曲线的平面选择。如果 AccJump 大于轨迹加速度 AccPathReduced 的 C1 倍，V_link 就会减小，直到加速度跃变的绝对值 AccJump 与 AccPathReduced 相等或者直到 V_link 等于 V_min（即 DXD 参数中的 minimum velocity at the segment transitions）

注意：修改 Dynamic 参数时，允许的空间和平面轨迹加速度也会自动地相应变化。

Reduction factor for C1 transitions: $C1 \geq 0.0$

- Critical angle, segment transition 'low' 曲线过渡处理的起始偏转角'low', 单位 Deg ϕ_{low} . 曲线过渡模式为 Coulumb 或者 Cosine 时，表示相邻曲线的夹角大于此角度后，才开始降速处理。
- Critical angle, segment transition 'high' 曲线过渡处理的截止偏转角'high', 单位 Deg ϕ_{high} . 曲线过渡模式为 Cosine 时，表示相邻曲线的夹角大于此角度后，速度不再继续下降，而是按 进给速度*C0 factor 运动。
- Minimum velocity at the segment transitions 曲线过渡时的最小速度
每个 NCI 通道都有“最小进给速度 (V_min)”这个参数， $V_{min} \geq 0.0$ 。实际速度应当保持在此速度之上，但以下情况除外：在曲线过渡或者路径末端的合法停止，以及低于 V_min 的 Override 倍率请求，以及系统产生的例外情况，比如使用 DEVIATIONANGLE 减速模式时的反转运动（轨迹反转），如果偏转角 $\phi \geq \phi_h$ 的条件满足，最小轨迹速度就可以忽略。每一段曲线的进给速度都必须大于 V_min（前进方向）。（DEVIATIONANGLE 模式尚未发布，蓝色字体可忽略）。
最小速度 V_min 的单位是 mm/sec，NC 代码中可以随时修改这个值。

Channel 2	Global Soft Position Limits (for x,y,z-axes)	FALSE	B
GO Channel 2_Itp	Interpreter Override Type	Reduced	E

- Global software limit positions for the path:

详见 Beckhoff Information System

TwinCAT2 | TwinCAT NC | TwinCAT NCI | Appednix | [Parameterisation](#)

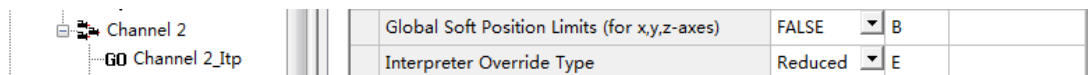
本选项提供两种监视限位的方式：

第一种，在 SAF 任务中监视软件限位：如果插补轴 Axis 的 Parameter 页面中打开了的软件限位，那么本选项也必须打开。这样 SAF 任务就会自动监视软限位。如果插补运动时超过了限位，进给速度就会立即设置为 0，整个插补通道就会报错。这种监视是通过轴参数来激活的，并不是本节描述的重点。

第二种, 路径软限位: 为了避免运动到路径终点时速度突然变 0, 本选项“Global software limit positions for the path” 必须启用。如果启用了此功能, 当一个插补运动可能会导致某个轴超过限位时, 该动作就会在达到限位之前提前减速并安全停止。

注意:

- 此监视功能只针对开启了软限位功能插补轴, 所以轴参数中的 Software Limit 功能必须打开。
- 监视功能在标准的几何曲线轨迹运动中并列运行, 包括: 直线、圆弧、螺旋。从 TwinCAT V2.10 B1258 开始, 辅助轴也可以被监视。
- 此功能不能监视样条曲线。与样条曲线有关的设定值由误差允许球面范围所限制, 如果超出该范围, 将由 SAF 任务监视软限位。
- 由于软限位的有效监视只能在预读之前在 NC 程序的 Run-time 中处理, 所以 NCI 通道有可能会执行一行明知会超出软限位的 G 代码, 直到进给轴移动到接近限位的位置才停下来。
- 如因某些原因, 轴处在限位之外, 允许用直线插补指令让它回到正常区域。



● Path override type | Interpreter override type

详见 Beckhoff Information System

TwinCAT2 | TwinCAT NC | TwinCAT NCI | Appednix | [Parameterisation](#)

Path override 是速度倍率。修改 Override 就等于修改速度, 但并不影响加速度和加加速度。由于有关的动态参数 (刹车距离, 加速度等), 不可能在每一段曲线都能达到 G 代码给定的目标速度, 因此 NCI 会计算出每一段空间曲线的最大速度 v_{max} , 有可能低于 G 代码给定的速度 v_{path} 。Path override type 决定了实际设定进给速度 v_{res} 的计算公式:

选项 “Reduced” – 默认选项, 基于 v_{max} ,

$$v_{res} = \min(v_{path}, v_{max}) * \text{Override}$$

无论指令速度是否超过最大曲线速度, Override 都起倍率作用。

选项 “Original” – 基于 v_{path} ,

$$v_{res} = \min(v_{path} * \text{Override}, v_{max})$$

指令速度乘以 Override 超过最大曲线速度, Override 就不起倍率作用。

选项 “Reduced [0 ... >100%]” – 类似 Reduced 模式, 且可以定义大于 100% 的倍率。

比如 120% 的速度倍率。但是 v_{path} 也得受限于每个插补轴的 PTP 参数: G0 速度及最大加减速设置。这种模式下, 必须通过 PLC 程序来写参数才能限制最大倍率, 比如 150%。实际进给速度的计算公式为:

$$v_{res} = \min(v_{path}, v_{max}) * \min(\text{Override}, 150)$$

无论指令速度是否超过最大曲线速度, Override 都起倍率作用, 但 Override 不能超过最大倍率, 比如本例中的 150。

3 使用 G 代码的插补运动项目

本章使用 PLC 程序来控制上一章上建立的 NC PTP 轴和插补通道。

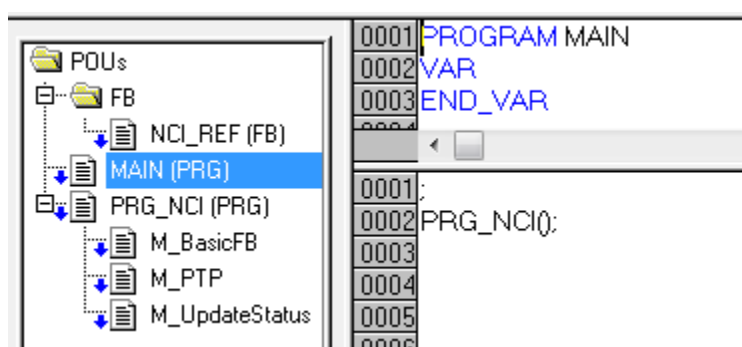
例程：\配套例程\第 3 章 使用 G 代的 NCI 项目\

LIB	2016/10/23 18:08	文件夹	
Mdemo	2016/10/23 15:49	NC 文件	1 KB
NCI_GCode	2016/10/23 23:47	PRO 文件	468 KB
NCI_Test	2016/10/23 19:45	TSM 文件	247 KB
第 3 章 使用G代的NCI项目	2016/10/24 18:38	WinRAR 压缩文件	530 KB

在第 2 章, 通道的控制都通过 TwinCAT System Manager 中的测试界面来实现, 接下来用 PLC 程序来实现这些控制, 包括通道的组合、解散、G 代码文件的装载、运行、停止、复位等。所以 TSM 文件继续使用第 2 章中新建的 NCI_Test.tsm。

3.1 在 PLC 中新建 NCI 程序

为了方便导出到其它程序使用, 所以代码并不写在 MAIN 中, 而是另外取名, 比如本例中叫做 PRG_NCI。需要在 MAIN 中引用 PRG_NCI, 程序才会执行。如图所示:



3.1.1 准备工作

- 准备工作 1: 新建 Pro 项目, 保存命名为“NCI_GCode_New.pro”, 并添加 PTP 程序所需要的库 TcMc2.lib 以及 NCI 程序所需要的库 TcNcCfg.lib 和 TcNci.lib。


```

TcMath.lib 23.9.04 14:15:30
TcNC.lib 10.10.08 16:55:34
TcNcCfg.lib 17.5.13 13:15:06
TcNci.lib 19.8.15 11:42:26
TcBase.lib 14.5.09 12:14:08
TcSystem.lib 21.1.15 09:22:54
TcBaseMath.lib 27.7.04 11:07:56
TcMC2.lib 19.10.15 14:49:44
STANDARD.LIB 5.6.98 12:03:02

```

手动加了这 3 个库之后，图中的其它库都会自动添加进来。

4) 准备工作 2: 新建功能块 NCI_Ref

这一步不是必须的，而是为了符合 TcMc2.lib 的习惯，将通道的控制对象约定为一个变量。在 TcMc2.lib 的 PTP 指令中，所有 FB 的控制对象都是 Axis_Ref，所以在 NCI 程序中我们定义一个 FB 叫做 NCI_Ref，如图所示：

```

FUNCTION_BLOCK NCI_REF
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR
    PlcToNci    AT    %Q*    :    NciChannelFromPlc;
    NciToPlc    AT    %I*    :    NciChannelToPlc;
END_VAR

```

变量名可以修改，而变量类型是所有 NCI 相关库中定义的，不能修改。

3.1.2 新建 NCI 通道控制的基本 FB 及其接口变量

在 PRG_NCI 的局部变量中声明以下变量或者功能块实例。

通道控制的基本动作包括：

- 组合通道，CfgBuildExt3DGroup;
- 解散通道：CfgReconfigGroup;
- 装载 G 代码：ItpLoadProgEx;
- 启动停止：ItpStartStop;
- 通道复位：ItpResetEx2

另外，通道的速度倍率是在 PLC 的接口变量 NciChannelFromPlc 中周期性刷新的，既可

以直接 PLC 赋值，也可以用函数（FC）ItpSetOverridePercent 来控制。

为此，我们新建以下局部变量：

VAR

(*基本动作功能块：通道组合，通道解散，装载 G 代码文件，启动停止，复位*)

```
CfgBuildExt3DGroup      : CfgBuildExt3DGroup;
fbClearGrp              : CfgReconfigGroup;

fbLoadFile              : ItpLoadProgEx;
fbItpStartStop         : ItpStartStop;
fbItpReset              : ItpResetEx2;
```

(*通道 ID 和轴组 ID*)

```
GroupID                 : UDINT;
ChannelID               : UDINT;
```

(*通道控制信号，对应 5 个基本动作功能块的触发命令：使能，复位，装载，启动，停止，其中使能为 True 即组合通道，为 False 则解散通道*)

```
bGroupEnable           : BOOL;
bGroupReset            : BOOL;
bLoad                  : BOOL;
bStart                 : BOOL;
bStop                  : BOOL;

bSetOverride           : BOOL;
rOverride              : LREAL:=100;
sFileName              : STRING(80):= 'c:\twincat\cnc\mdemo.nc';
```

(*NCI 通道的 Axis 和 Channel，与 TwinCAT NC 的接口变量*)

```
AxisX                  : AXIS_REF;
AxisY                  : AXIS_REF;
AxisZ                  : AXIS_REF;
AxisQ1                 : AXIS_REF;
AxisQ2                 : AXIS_REF;
ItpChannel             : NCI_REF;
```

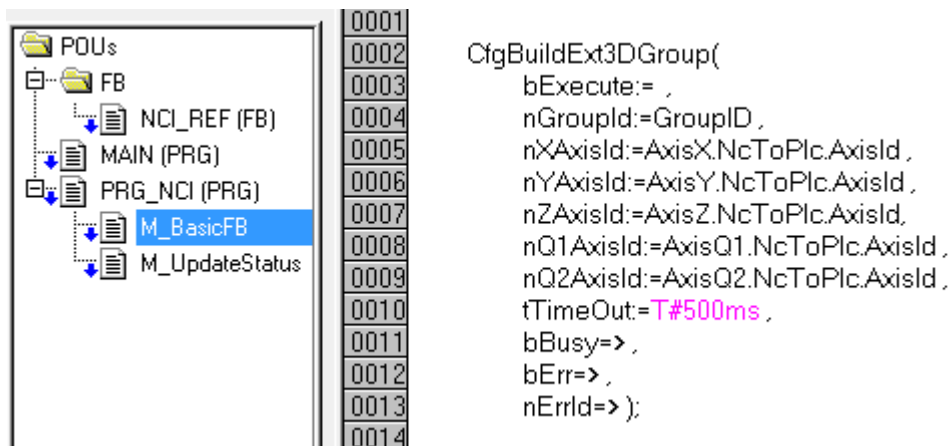
END_VAR

3.1.3 编写基本代码

基本代码分为控制命令的执行，和通道状态的刷新。由于轴的使能属于 PTP 代码，为了示例简单，PTP 轴使能的代码也写在这里。

1) 控制命令的执行

实际上，控制命令的执行就是罗列 5 个 FB 的 Instance，但是把每个 FB 的 bExecute 都悬空不填，这样就可以在程序的其它地方控制这些 bExecute 条件了。为了程序清晰，把以上代码放到 Action 里面，名为 M_BasicFB，如图所示：



代码如下：

```

CfgBuildExt3DGroup(
    bExecute:= ,
    nGroupId:=GroupID ,
    nXAxisId:=AxisX.NcToPlc.AxisId ,
    nYAxisId:=AxisY.NcToPlc.AxisId ,
    nZAxisId:=AxisZ.NcToPlc.AxisId ,
    nQ1AxisId:=AxisQ1.NcToPlc.AxisId ,
    nQ2AxisId:=AxisQ2.NcToPlc.AxisId ,
    tTimeOut:=T#500ms ,
    bBusy=> ,
    bErr=> ,
    nErrId=> );

```

```

fbClearGrp(
    bExecute:= ,
    nGroupId:= GroupID ,
    tTimeOut:= T#500MS);

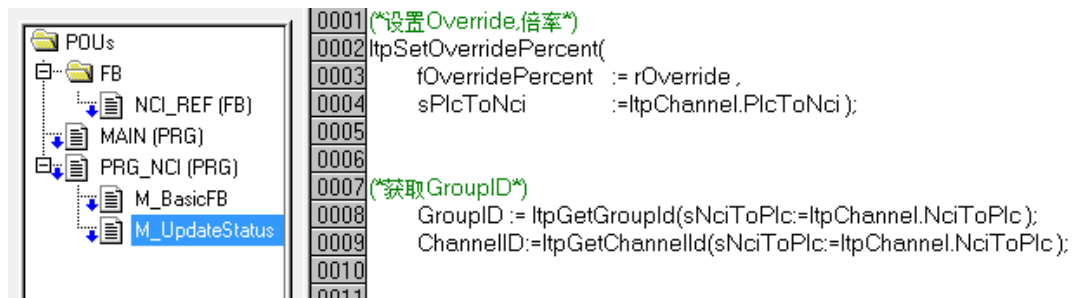
```

```
fbLoadFile(  
    bExecute:= ,  
    sPrg:= sFileName,  
    nLength:=LEN(sFileName) ,  
    tTimeOut:=T#500MS ,  
    sNciToPlc:=ItpChannel.NciToPlc ,  
    bBusy=> ,  
    bErr=> ,  
    nErrId=> );  
  
fbItpStartStop(  
    bStart:= ,  
    bStop:= ,  
    nChnId:=ChannelID,  
    tTimeOut:=T#500MS ,  
    bBusy=> ,  
    bErr=> ,  
    nErrId=> );  
  
fbItpReset(  
    bExecute:=,  
    tTimeOut:= T#500MS,  
    sNciToPlc:= ItpChannel.NciToPlc,  
    bBusy=> ,  
    bErr=> ,  
    nErrId=> );
```

以上功能块的接口变量，其功能几乎可以一目了然。对基类型不明确的，打开本文所附的例程也就清楚了。需要提示的是，装载 G 代码文件时，指的是控制器上的路径，所以 G 代码一定要复制到控制器上，并且在程序里提供正确的路径，包括文件夹和文件名。

2) 通道状态刷新

为了尽快看到 PLC 控制 NCI 轴的结果，本例并没有完整的状态，但是可以先设置这个功能的 Action，以后再增加状态变量和相应的代码。现在，只是刷新倍率设置和通道及轴组 ID 号。代码放到 Action 里面，名为 M_UpdateStatus，如图所示：



代码如下:

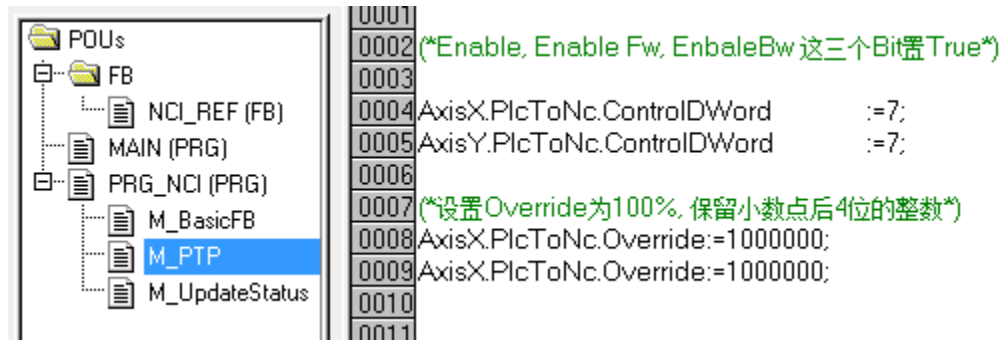
```
(*设置 Override,倍率*)
ItpSetOverridePercent(
    fOverridePercent := rOverride ,
    sPlcToNci       :=ItpChannel.PlcToNci );
```

```
(*获取 GroupID*)
    GroupID := ItpGetGroupId(sNciToPlc:=ItpChannel.NciToPlc);
    ChannelID:=ItpGetChannelId(sNciToPlc:=ItpChannel.NciToPlc );
```

同样道理, 以上

3) PTP 命令, 使能和 Override 设置

代码放到 Action 里面, 名为 M_PTP:



代码如下:

```
(*Enable, Enable Fw, EnbaleBw 这三个 Bit 置 True*)
AxisX.PlcToNc.ControlDWord :=7;
AxisY.PlcToNc.ControlDWord :=7;
```

```
(*设置 Override 为 100%, 保留小数点后 4 位的整数*)
AxisX.PlcToNc.Override:=1000000;
```

```
AxisX.PlcToNc.Override:=1000000;
```

3.1.4 编写 NCI 通道控制 FB 的触发逻辑

为了实现 System Manager 中的功能，最简单的触发方式就是手动。上一节我们定义了 5 个变量即分别用不同的 Bool 变量，计划用来触发这 5 个功能块的动作。在本节我们可以不用变量，而是直接手动强制 FB 的 bExecute 变量来触发控制命令。

为此，先把这些 bExecute 罗列在 NCI 程序代码区，比 Login 之后展开 FB 实例去找变量要方便快捷。

(*准备手动控制 NC 通道的触发命令*)

```
rOverride;  
CfgBuildExt3DGroup.bExecute;  
fbClearGrp.bExecute;  
fbItpStartStop.bStart;  
fbItpStartStop.bStop;  
fbLoadFile.bExecute;  
sFileName;  
fbItpReset.bExecute;
```

另外，NCI 程序代码区还要增加前面所建的 3 个 Action 的引用：

(*引用 3 个 Action，实现不同的功能*)

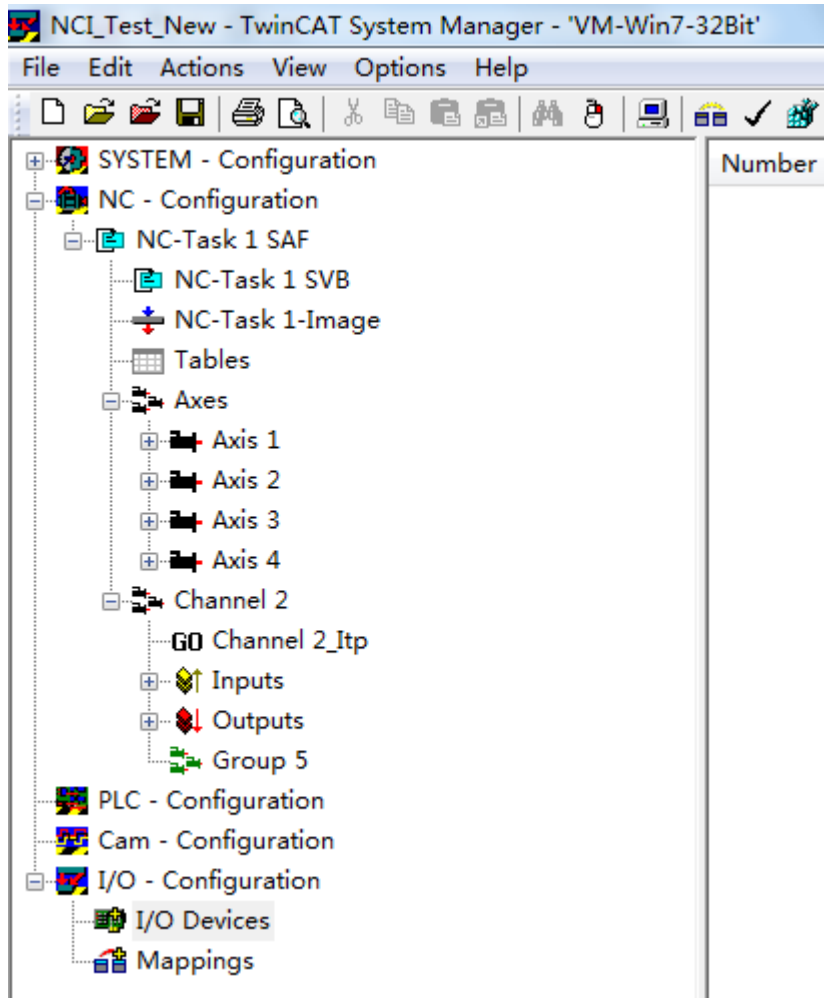
```
M_PTP;          (*轴使能*)  
M_BasicFB;     (*NCI 通道控制*)  
M_UpdateStatus; (*NCI 通道状态刷新*)
```

至此，控制 NCI 通道的基本程序就写成了。

然后就可以编译了，如果没有 Error 报错，这部份工作就算完成了。

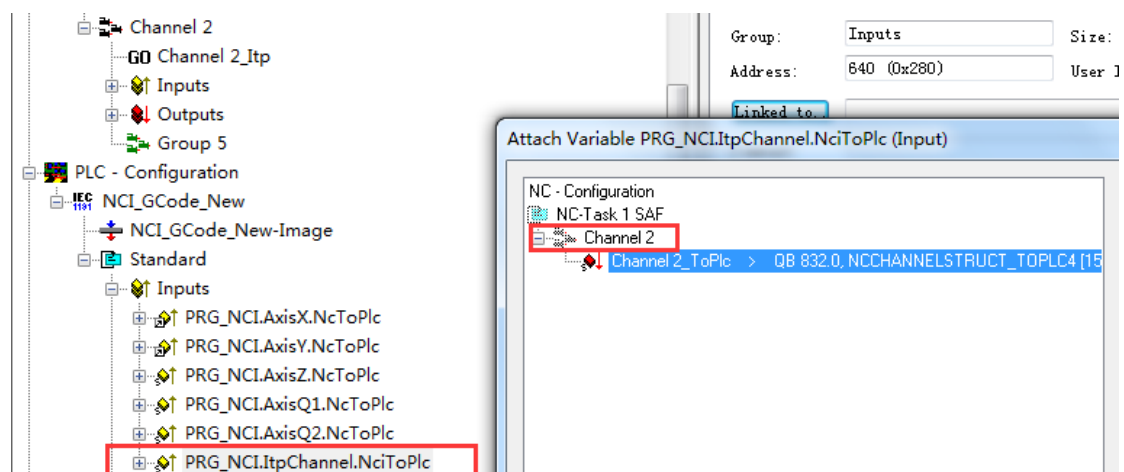
3.2 在 System Manager 中引用 NCI 程序

- 1) 准备工作：在 System Manger 中打开第 2 章的例程 NCI_Test.tsm，另存为 NCI_Test_new.tsm。



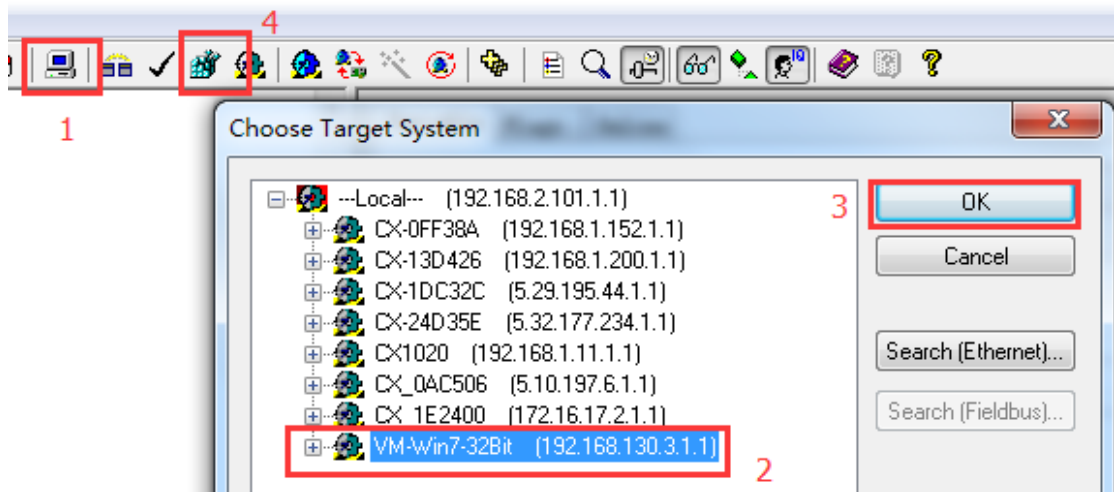
2) 导入 PLC 程序，并链接 Axis 及 Channel 变量

Axis 的链接对于 PTP 的用户来说已经很熟悉了，NCI 的链接也是同样道理。Channel 的接口变量是一对 150 字节的结构：NciChannelFromPlc 和 NciChannelToPlc。

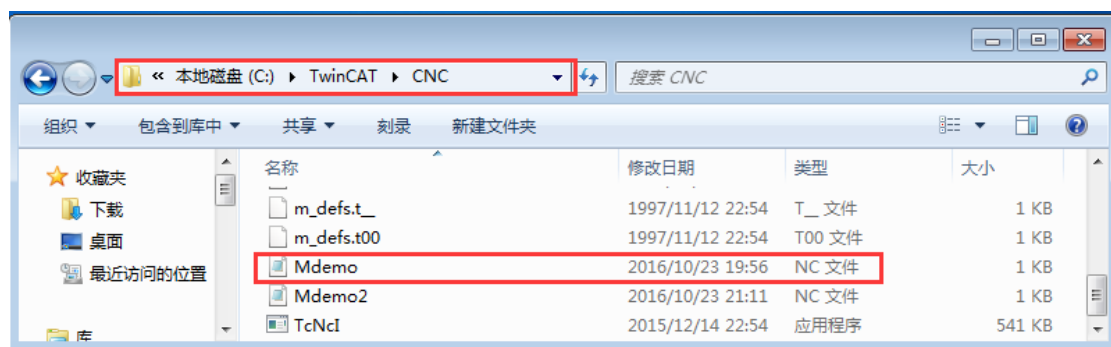
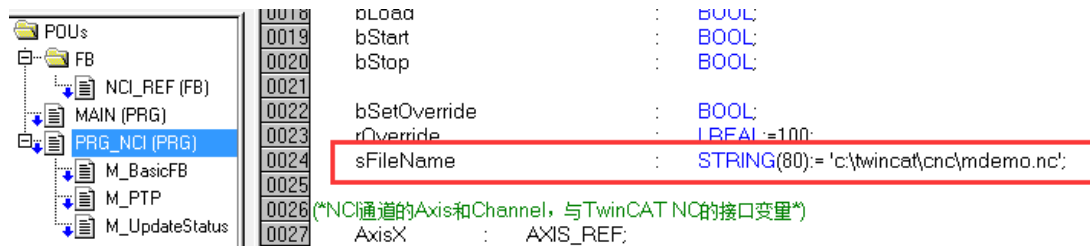


最简单的插补运动只有 X 和 Y 轴，所以其它轴可以不必链接。

3) 选中目标系统，激活配置。

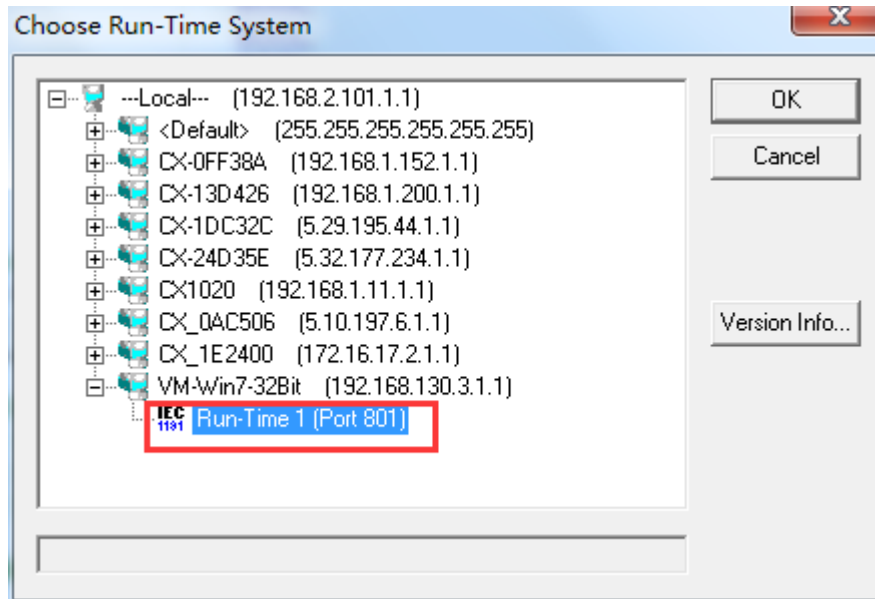


3.3 确认控制器上的 CNC 文件路径与 PLC 程序中一致



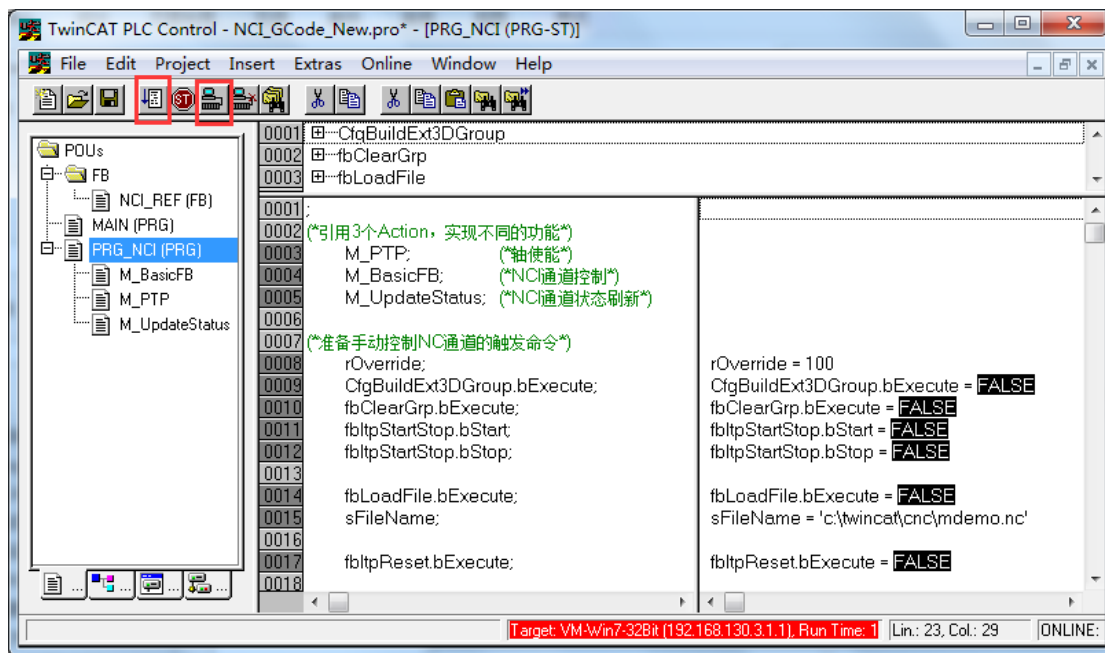
3.4 PLC 程序下载运行，强制变量 bExecute 执行各种指令

4) Choose Runtime System

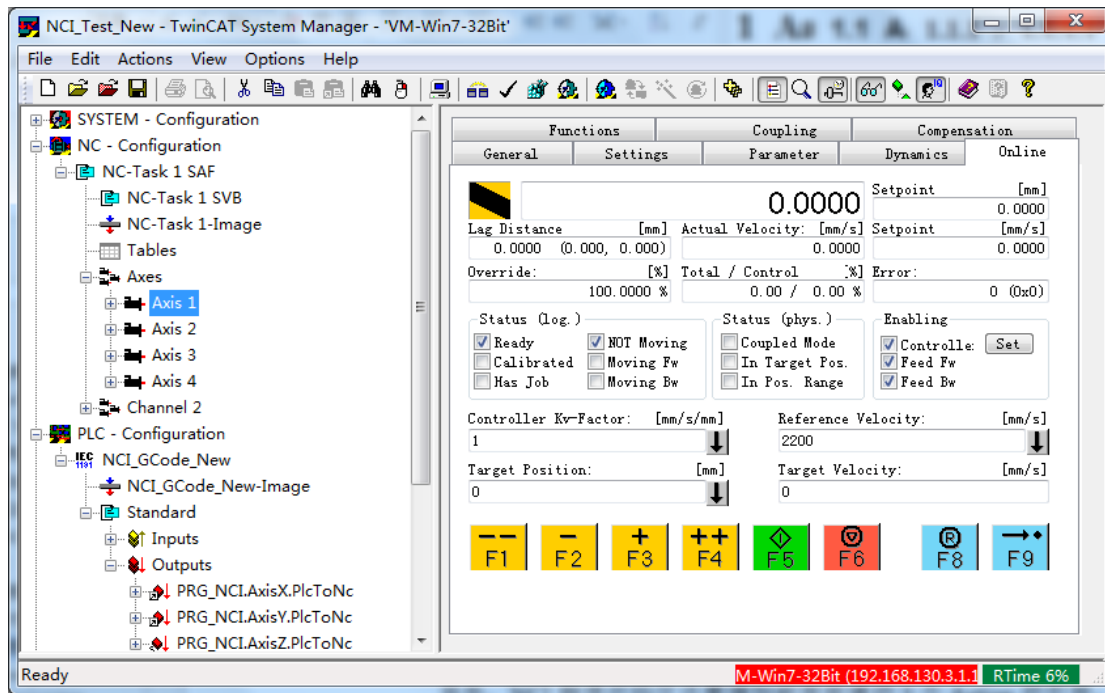


5) 程序运行之初，NC 轴的状态

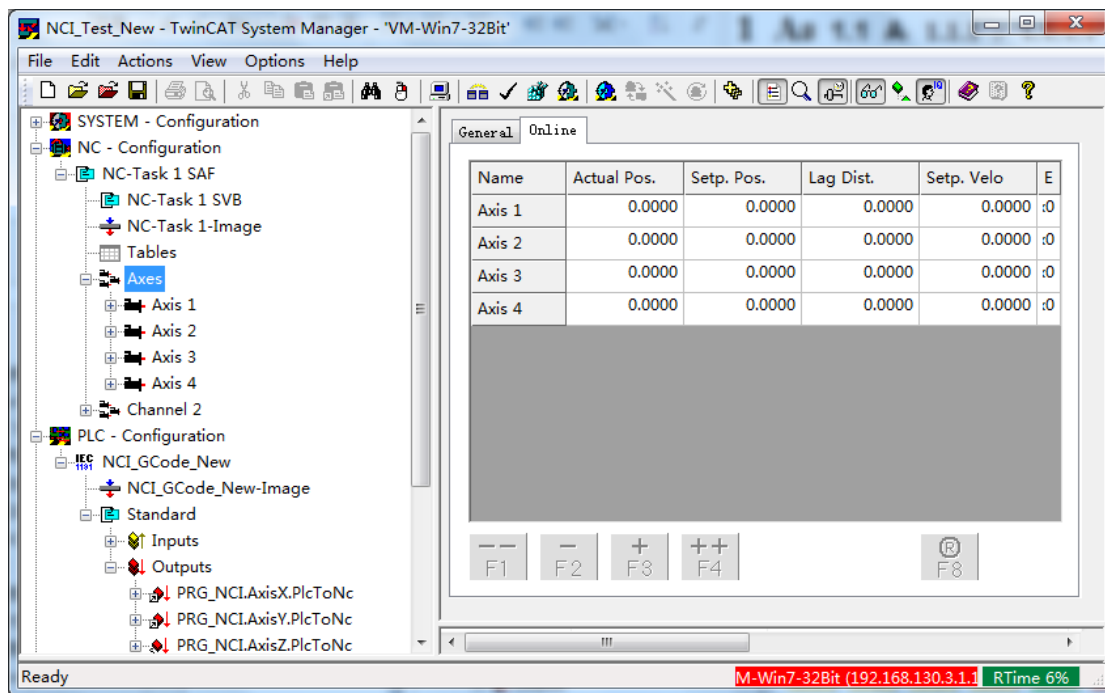
Login 并 Run，定位到 PRG_NCI 的代码区



可以看 Axis X 和 Axis Y 所对应的轴 1、轴 2 已经使能了。



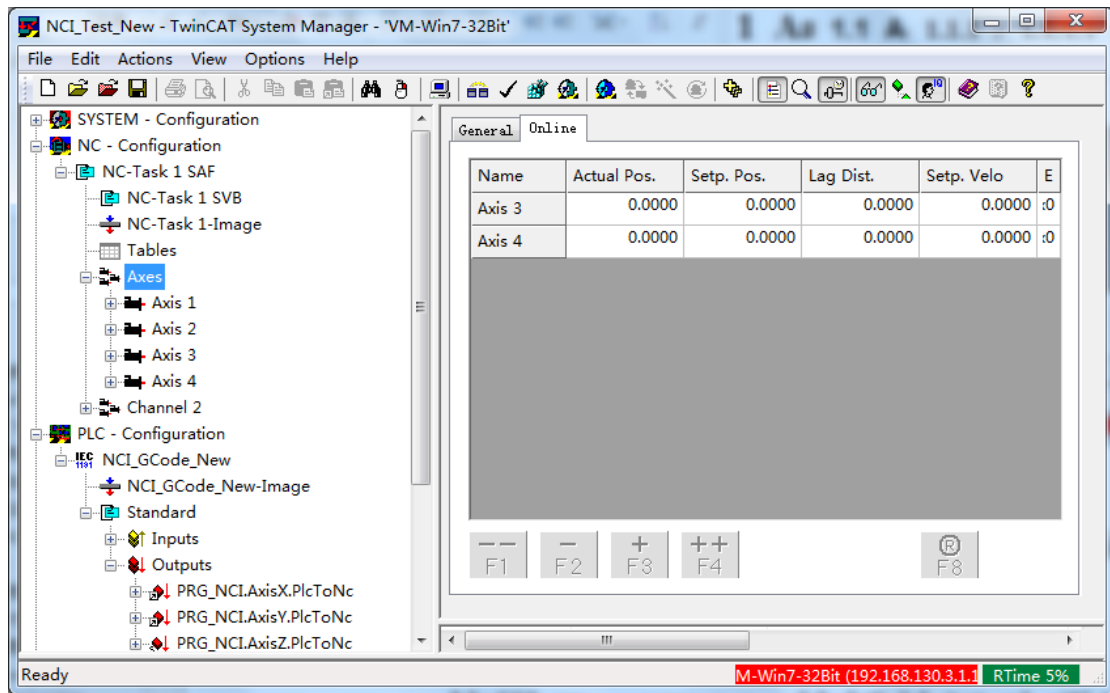
并且各轴是处在 PTP 状态下，可以从 Axes 树形结构的 Online 页面看到全部轴的信息：



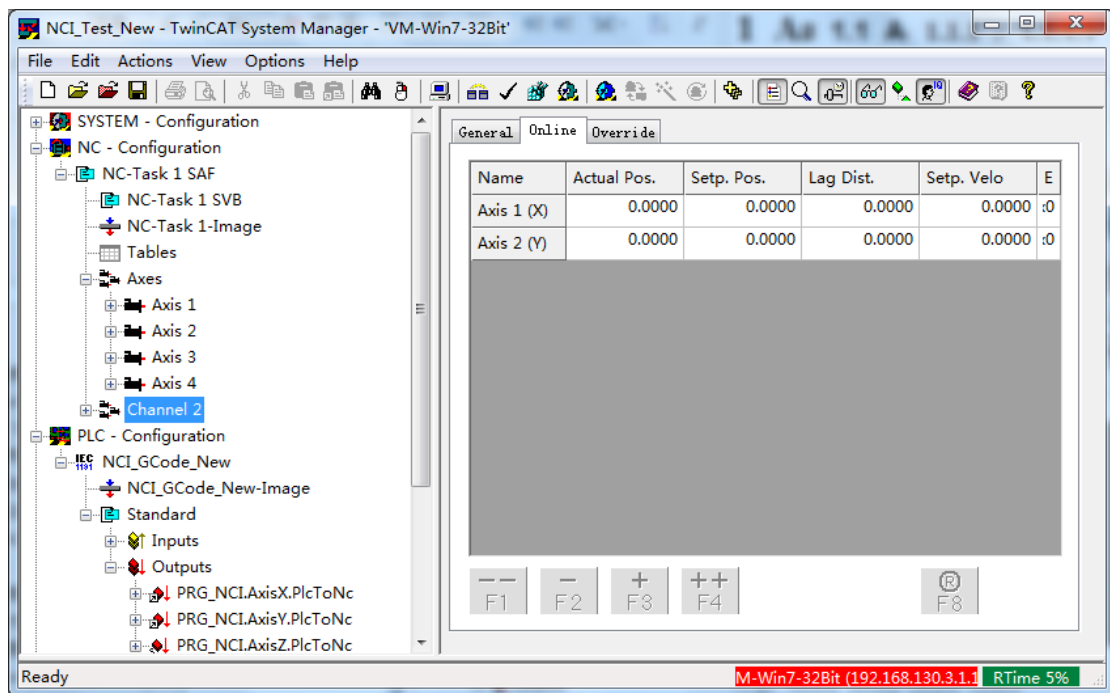
6) 组合通道

`0009` CfgBuildExt3DGroup.bExecute; `CfgBuildExt3DGroup.bExecute = TRUE`

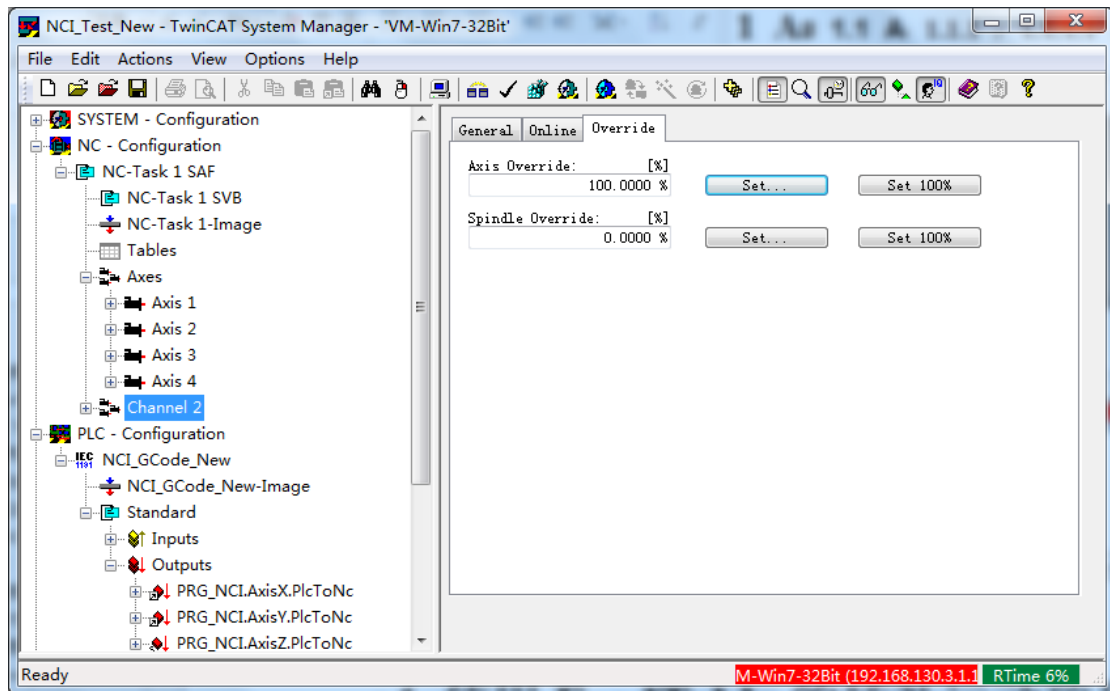
Axes 中少了轴 1 和轴 2 的信息：



而 Channel 中有了轴 1 和轴 2 的信息：



并且，通道的 Override 也成功设置为 100%了。

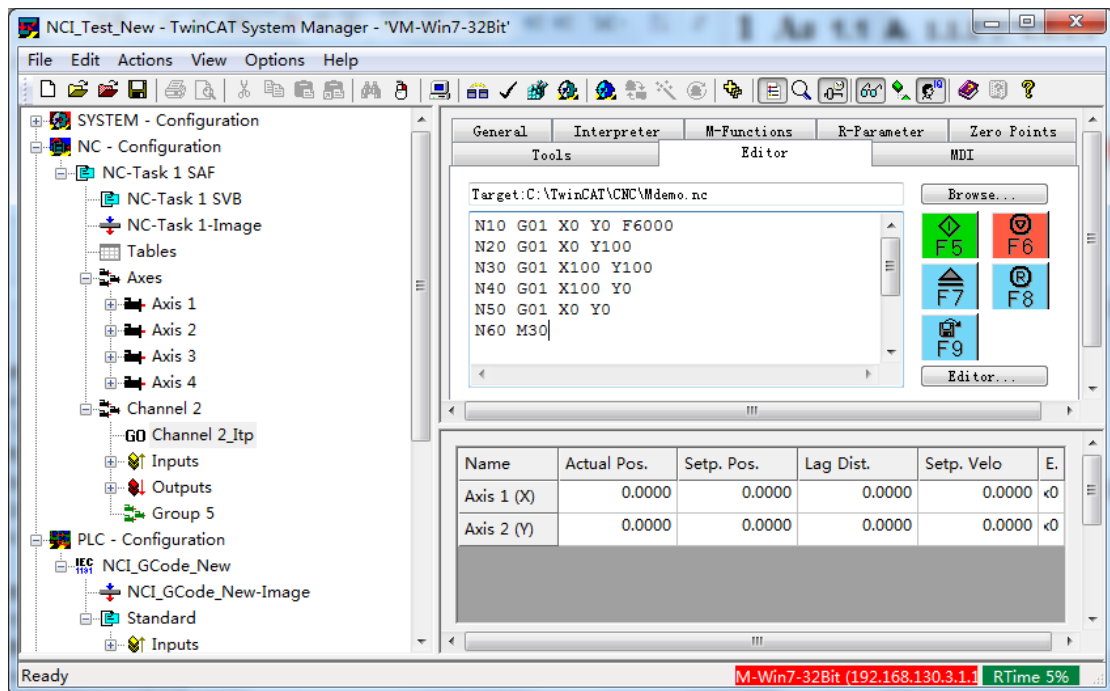


这表明这两个轴现在归 NCI 通道控制，不能再做 PTP 动作了。

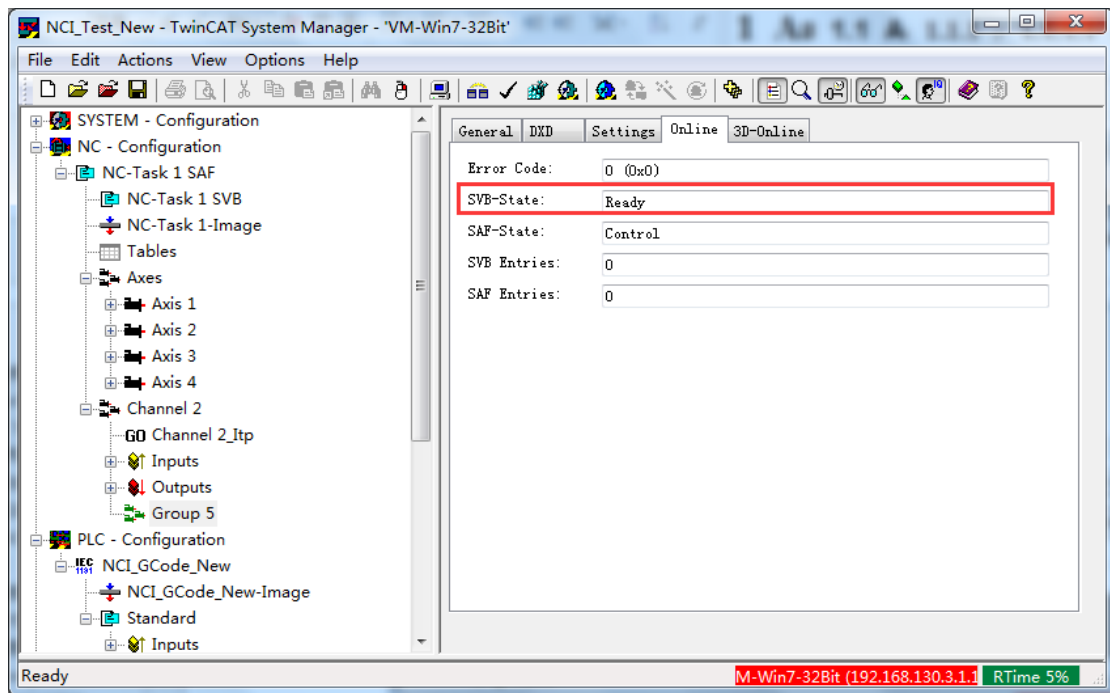
7) 装载 G 代码



在 Editor 页面可以看到，G 代码已经成功装载。



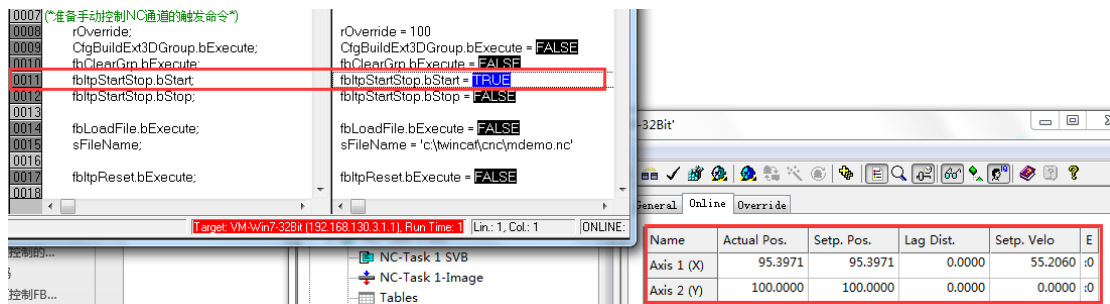
而 Group 的状态也到了 Ready。



表示 X、Y 轴随时可以按照 G 代码指定的轨迹来运行。

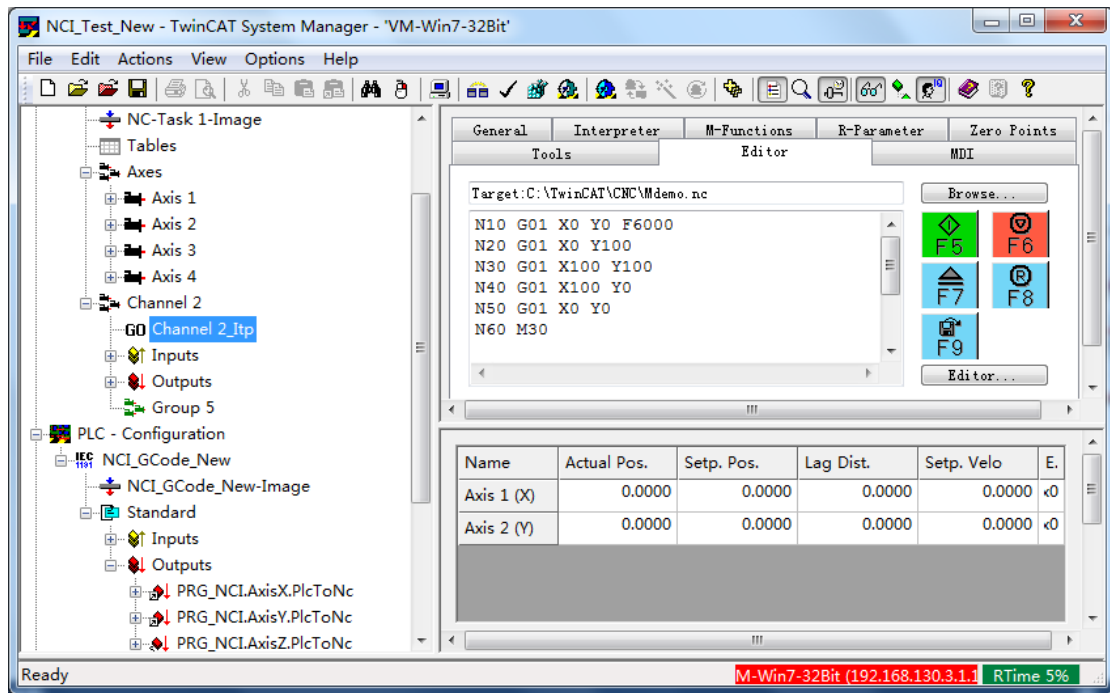
8) 通道启动运行

Group 状态为 Ready, 表示插补通道已经储存了一些数据, 可以运行。这时只要一个 Start 指令, 就可以开始动作了。



上图可见, 已经开始动作。

最终 Axis 1 (X) 和 Axis 2 (Y) 都停在了位置 0, 这是由于 G 代码的最后一行动作代码的终点坐标为 X0 Y0。



9) 通道停止、复位

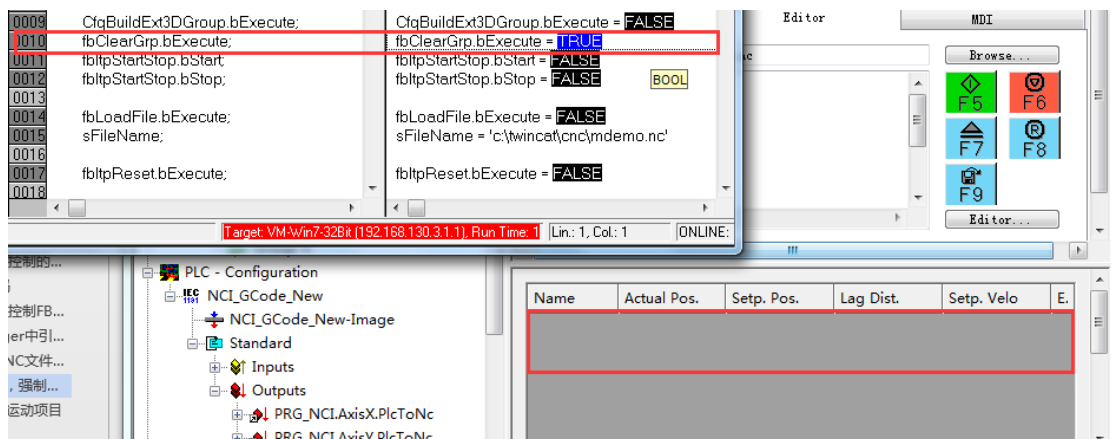
停止和复位要用录屏软件才能记录效果，用户可以自行测试。

停止和复位对通道的影响是类似的，当前运动马上停止。之后必须重新装载 G 代码才能从头运动。只是当轴出现报警之外，必须用 Reset 才能清除。

组合成通道的 NCI 轴，无法用 PTP 功能块 MC_Reset 进行单个轴的复位。

如果是 NCI 动作的过程中只是要暂停再继续运动，最简单的方式是，Override 设置为 0，需要继续运动时再恢复为 100%。

10) 通道解散



3.5 NCI 通道 G 代码控制 FB 封装示例：FB_NCI_GCode

\配套例程\第 3 章 使用 G 代的 NCI 项目\Demo Self Defined NCI FB\

下册补充 NCI > 配套例程 > 第 3 章 使用G代的NCI项目 > Demo Self Defined NCI FB

名称	修改日期	类型	大小
DEFAULT.DFR	2016/10/29 18:18	DFR 文件	1 KB
NCI_GCode	2016/10/29 16:58	PRO 文件	287 KB
NCI_Test	2016/10/23 19:45	TSM 文件	247 KB

例程用到 Lib 和 G 代码在上一层目录：

下册补充 NCI > 配套例程 > 第 3 章 使用G代的NCI项目 >

文件夹

名称	修改日期	类型	大小
20161020	2016/10/24 18:40	文件夹	
Demo Self Defined NCI FB	2016/10/29 18:18	文件夹	
First GCode Pro	2016/10/29 18:16	文件夹	
GCode	2016/10/29 18:16	文件夹	
LIB	2016/10/23 18:08	文件夹	
第 3 章 使用G代的NCI项目	2016/10/29 18:18	WinRAR 压缩文件	563 KB

这个例程是在前面这个 Pro 的基础上，补充了以下功能：

将 Prg_NCI 封装成 FB，

并完善了通道状态信息，以及缓存 G 代码数量。

引用 FB 做 NCI 控制时还集成了 PTP 基本功能。

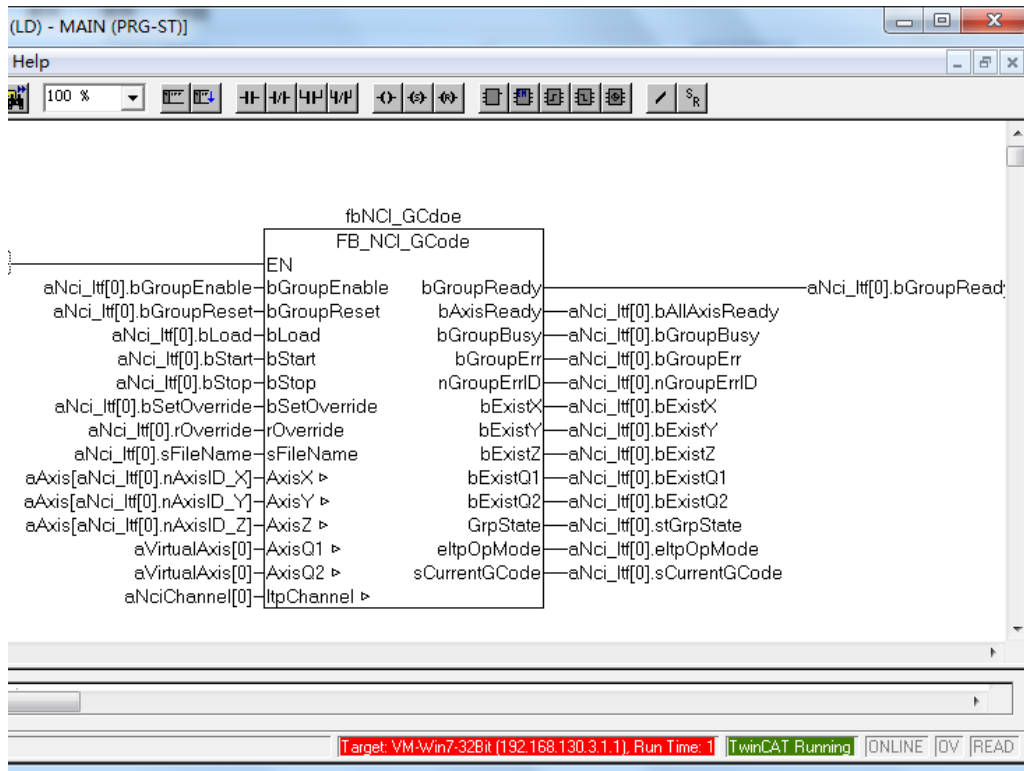
轴和通道的控制信号和状态信息都分别放在 Interface 的结构体中。

增加了调试画面。

以下为这个例程的说明

3.5.1 功能块的调用

在 MAIN 程序的 Action 子程序“Act_Call_GCode”中可以直观地看到功能块的输入变量和输出输出变量，



其中输入变量

- bGroupEnable**: 通道组合和解散;
- bGroupReset** : 通道复位;
- bLoad** : G 代码装载;
- bStart** : 插补运动启动;
- bStop** : 插补运动停止;

是为了执行 3.1 节中用到的 5 个功能块: 通道组合, 通道解散, G 代码装载, 启动停止, 通道复位。

- bSetOverride** : 修改倍率;
- rOverride** : 通道倍率, 单位 (%);
- sFileName** : G 代码文件的路径;

接口变量 5 个 PTP 轴 (Axis X 到 Axis Q2) 和 1 个 NCI 通道 (ItpChannel) 定义了 NCI 通道的组成。

输出变量:

- bGroupReady** : 通道正常, 已装载好代码, 随时可以开始动作;
- bAxisReady** : 通道内每个轴都能成功, 没有报错;
- bGroupBusy** : 通道正在执行动作;
- bGroupErr** : 通道错误;
- nGroupErrID** : 通道的错误代码;
- bExistX** : X 轴已启用;

bExistY : Y 轴已启用;
bExistZ : Z 轴已启用;
bExistQ1 : Q1 轴已启用;
bExistQ2 : Q2 轴已启用;
GrpState : NCI 通道的指令执行状态, 包括 **Buffer** 数目, 当前执行行号等;
eItpOpMode : NCI 通道的运行模式, 最常见的是 **Idle**、**Ready** 和 **IsRuning**;
 组合后装载 G 代码前为 **Idle**, 成功装载 G 代码后启动前为 **Ready**,
 启动后结束前为 **IsRuning**。动作完成后又恢复为 **Ready** 状态。
sCurrentGCode : 当前执行的 G 代码; (备用, 现在读不出来)

从图上可以看出, **FB** 的接口做成了单个的变量, 而调用 **FB** 并给它的接口赋值时, 统一使用了 **aNci_Itf[0]** 中的元素, 并且调用时使用了梯形图。这是为了使程序监视更直观, 可以在线显示每个接口变量的值, 也可以随时强制某个变量。

在全局变量文件 **Globale_Variabls** 中定义了数组 **aNci_Itf**:

```
aNci_Itf      :   ARRAY[0..nMaxChannel] OF GCode_Chn_Interface ;
```

这是为 **NCI** 通道专门建的接口数组, 一个 **NCI** 通道对应一数组里的一个结构。每个结构中包含了一个 **NCI** 通道的所有状态和控制信息。下一节会就此详展开细介绍。

另外 **aVirtualAxis** 是为纯粹为了补齐 5 个轴, 避免 **FB** 报错而建的辅助轴。实际使用是不要链接到任何 **NC** 轴即可。

3.5.2 控制对象和 Interface

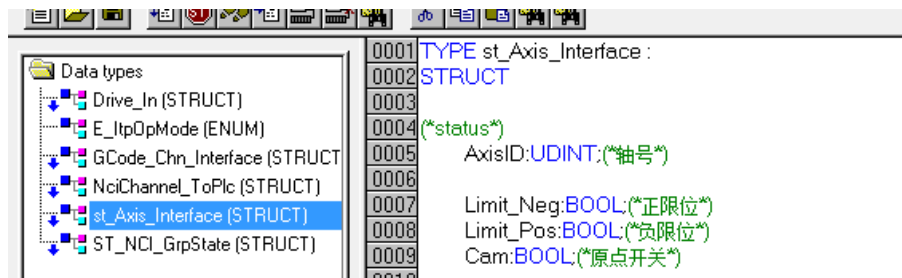
控制对象包括 **NC** 轴和 **NCI** 通道。本例程的基本思路是:

为每个 **PTP** 轴建一个 **Interface** 结构体 **st_Axis_Interface**, 为每个 **NCI** 插补通道也建一个 **Interface** 结构体 **GCode_Chn_Interface**。

所以在全局变量中: **aAxis** 和 **Axis_Interface** 数组的下标数字相同, 而 **aNciChannel** 和 **aNci_Itf** 数组的下标数字相同。如图所示:

0001	VAR_GLOBAL		
0002	aAxis	:	ARRAY[0..nMaxAxis] OF AXIS_REF;
0003	Axis_Interface	:	ARRAY[0..nMaxAxis] OF st_Axis_Interface;
0004			
0005	aNciChannel	:	ARRAY[0..nMaxChannel] OF NCI_REF;
0006	aNci_Itf	:	ARRAY[0..nMaxChannel] OF GCode_Chn_Interface ;
0007			
0008	(*与硬件相关*)		
0009	Brake	AT %Q*	: BOOL;
0010	MainPower	AT %Q*	: BOOL;
0011	aDriveInput	AT %I*	: ARRAY[0..nMaxAxis] OF Drive_In;
0012			

其中 **Axis** 的接口定义, 来自原先的 **TcMc2.lib** 例程:



由于结构体元素较多，不方便截图，以下为文字版：

```

TYPE st_Axis_Interface :
STRUCT

(*status*)
    AxisID:UDINT;(*轴号*)

    Limit_Neg:BOOL;(*正限位*)
    Limit_Pos:BOOL;(*负限位*)
    Cam:BOOL;(*原点开关*)

    Drive_Alarm:BOOL;(*驱动器报警*)
    Drive_Ready:BOOL;(*驱动器准备好*)
    Drive_Torque:REAL;(*1000 对应 100% 额定转矩*)
    DC_Voltage:REAL;(*驱动器直流母线电压*)

    Current_Pos:LREAL;(*轴当前位置*)

    Axis_Ready:BOOL;(*NC 轴准备好*)
    Axis_Err:BOOL;(*NC 轴故障*)
    Axis_ErrID:UDINT;(*NC 轴故障代码*)
    Axis_IsCalibrated:BOOL;(*NC 轴寻参完成*)
    Axis_IsNotMoving:BOOL;(*NC 轴静止*)
    Axis_IsPTP :BOOL;

(*manual control*)
    bEnable:BOOL;(*NC 轴使能*)
    bEnableFw:BOOL;(*NC 轴允许正转*)
    bEnableBw:BOOL;(*NC 轴允许反转*)
    Jog_Pos:BOOL;(*NC 轴正向点动*)
    Jog_Neg:BOOL;(*NC 轴反向点动*)
    bStop:BOOL;(*NC 轴正常停止*)
    bMoveAbs:BOOL;(*NC 轴绝对定位开始*)
    bMoveRel:BOOL;(*NC 轴相对定位开始*)

```

```
bHome:BOOL;(*NC 轴寻参开始*)
bReset:BOOL;(*复位 NC 轴*)
bSetPos:BOOL;(*NC 轴重设位置*)

(*virtual Neg*)
bVirtualAxis:BOOL;(*NC 轴虚轴标记*)

(*Parameter*)
AbsMove_TargetPos:REAL:=10;(*NC 轴绝对定位位置*)
AbsMove_Velo:REAL:=10;(*NC 轴绝对定位的速度*)
RelMove_Distance:REAL:=10;(*NC 轴相对定位距离*)
RelMove_Velo:REAL:=10;(*NC 轴相对定位速度*)
Jog_Velo:REAL:=10;(*NC 轴点动速度*)
RefPos:REAL;(*NC 轴寻参完成时的原点位置*)
SetPosition:REAL;

END_STRUCT
END_TYPE
```

而 NCI 通道的接口定义如下:

<pre> Data types ├── Drive_In (STRUCT) ├── E_ltpOpMode (ENUM) ├── GCode_Chn_Interface (STRUCT) ├── NciChannel_ToPlc (STRUCT) ├── st_Axis_Interface (STRUCT) └── ST_NCI_GrpState (STRUCT) </pre>	<pre> 0001 TYPE GCode_Chn_Interface : 0002 STRUCT 0003 0004 bGroupEnable : BOOL; 0005 bGroupReset : BOOL; 0006 bStart : BOOL; 0007 bStop : BOOL; 0008 bLoad : BOOL; 0009 bSetOverride : BOOL; 0010 rOverride : REAL:=100; 0011 sFileName : STRING(80)='c:\twinCAT\cnc\demo.nc'; 0012 0013 bAllAxisReady : BOOL; 0014 bGroupReady : BOOL; 0015 bGroupBusy : BOOL; 0016 bGroupErr : BOOL; 0017 nGroupErrID : DWORD; 0018 0019 bExistX : BOOL; 0020 bExistY : BOOL; 0021 bExistZ : BOOL; 0022 bExistQ1 : BOOL; 0023 bExistQ2 : BOOL; 0024 0025 nAxisID_X : USINT:=0; 0026 nAxisID_Y : USINT:=1; 0027 nAxisID_Z : USINT:=2; 0028 nAxisID_Q1 : USINT; 0029 nAxisID_Q2 : USINT; 0030 0031 stGrpState : ST_NCI_GrpState; 0032 eltpOpMode : E_ltpOpMode; 0033 sCurrentGCode : STRING(80); 0034 END_STRUCT 0035 END_TYPE </pre>
---	---

以上只是例程的思路，用户可以根据实际情况再增减或者修改。

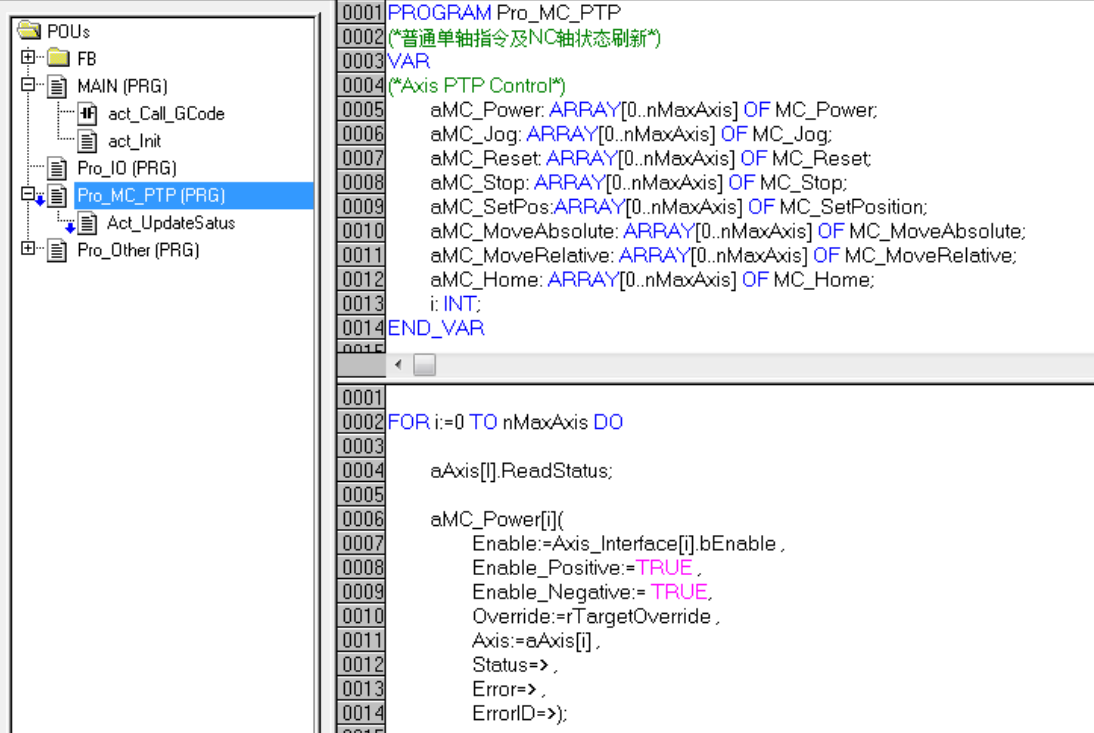
3.5.3 FB_NCI_GCode 的源代码

<pre> POUs ├── FB │ └── FB_NCI_GCode (FB) │ ├── M_BasicFB │ ├── M_UpdateStatus │ └── NCI_REF (FB) ├── MAIN (PRG) │ ├── act_Call_GCode │ ├── act_Init │ ├── Pro_ID (PRG) │ ├── Pro_MC_PTP (PRG) │ └── Pro_Other (PRG) </pre>	<pre> 0001 FUNCTION_BLOCK FB_NCI_GCode 0002 (*NCI通道控制封装FB_NCI_GCode *) 0003 (**Version 1.0.0 By LizzyChen 2016.10.29**) 0004 0005 VAR 0006 CfgBuildExt3DGroup : CfgBuildExt3DGroup; 0007 fbClearGrp : CfgReconfigGroup; 0008 0004 CASE iStep OF 0005 0: 0006 bGroupReady:=FALSE; 0007 bGroupBusy:=FALSE; 0008 0009 IF bGroupEnable AND bAxisReady THEN </pre>
---	--

如果当前封装的功能不能满足需求，用户可以在此基础上修改。如果涉及到接口变量的增加，就需要同时修改结构体 GCode_Chn_Interface，并在 FB_NCI_GCode 增加这些变量的控制代码。

3.5.4 PTP 控制程序

虽然这是 NCI 插补程序的例程，但是在实际项目中，必然涉及到 PTP 指令，比如使能、复位、寻参（回零）、点动等等。因此本例程直接集成了 TcMc2 PTP 的例程中的代码和 Axis Interface 结构体。



```

0001 PROGRAM Pro_MC_PTP
0002 (*普通单轴指令及NC轴状态刷新*)
0003 VAR
0004 (*Axis PTP Control*)
0005   aMC_Power: ARRAY[0..nMaxAxis] OF MC_Power;
0006   aMC_Jog: ARRAY[0..nMaxAxis] OF MC_Jog;
0007   aMC_Reset: ARRAY[0..nMaxAxis] OF MC_Reset;
0008   aMC_Stop: ARRAY[0..nMaxAxis] OF MC_Stop;
0009   aMC_SetPos: ARRAY[0..nMaxAxis] OF MC_SetPosition;
0010   aMC_MoveAbsolute: ARRAY[0..nMaxAxis] OF MC_MoveAbsolute;
0011   aMC_MoveRelative: ARRAY[0..nMaxAxis] OF MC_MoveRelative;
0012   aMC_Home: ARRAY[0..nMaxAxis] OF MC_Home;
0013   i: INT;
0014 END_VAR

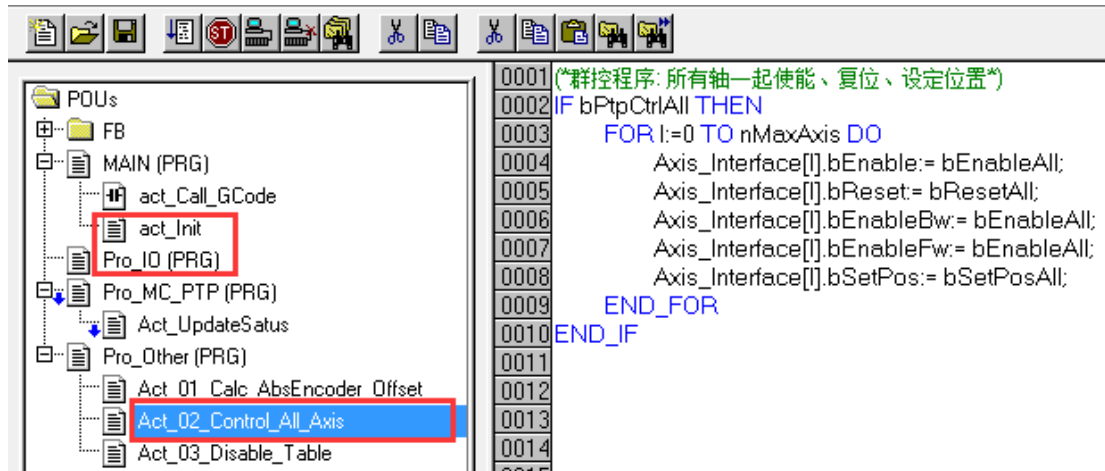
0001
0002 FOR i:=0 TO nMaxAxis DO
0003
0004   aAxis[i].ReadStatus;
0005
0006   aMC_Power[i](
0007     Enable:=Axis_Interface[i].bEnable,
0008     Enable_Positive:=TRUE,
0009     Enable_Negative:=TRUE,
0010     Override:=rTargetOverride,
0011     Axis:=aAxis[i],
0012     Status=>,
0013     Error=>,
0014     ErrorID=>);

```

这部分代码用到的结构体 `aAxis` 和 `Axis_Interface` 已经在 3.5.2 控制对象和 `Interfacec` 中介绍过了，PTP 的用户对这些代码也已经相当熟悉，这里不再详述。

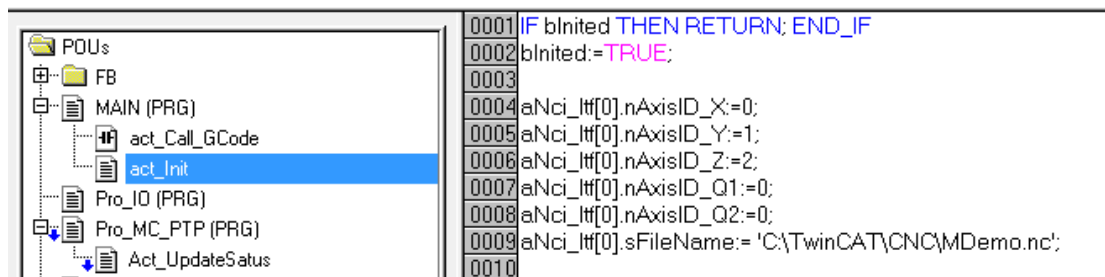
在本例中，还是通过程序 `Pro_MC_PTP` 中的 MC 功能块数组配合 For 语句来完成的。如果客户有兴趣，可以把一个 Axis 的功能也封装成一个 FB。

3.5.5 其它程序

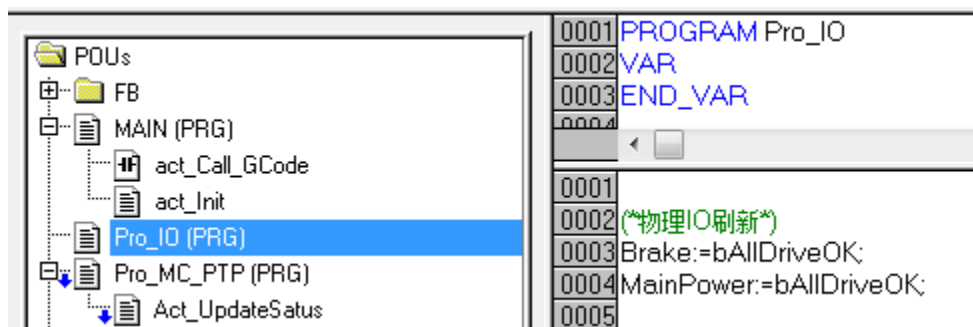


上图中红线框住的，都是无关核心功能的辅助程序，目的是为了调试方便。

11) Main 的 Act_Init 中，是为了初始化 NCI 通道接口变量中的轴号，也可以初始化



12) Pro_IO 中是与硬件 IO 相关的程序



在正式的应用项目中，可以把全部 IO 模块的变量赋值都放在这里。

在 Pro_Other 中集中放置辅助功能代码，本例中包括：

13) NciToPlc 结构体的转换赋值

```

0001 PROGRAM Pro_Other
0002
0003 (*其它辅助程序*)
0004
0005 VAR
0006   bCalc:BOOL;
0007   aOffset:ARRAY[0..11] OF REAL;
0008   aActPos:ARRAY[0..11] OF REAL;
0009   aSetPos:ARRAY[0..11] OF REAL;
0010
0011 (*计算绝对编码器的零位偏移量*)
0012 Act_01_Calc_AbsEncoder_Offset ;
0013
0014 (*料控程序:所有轴一起使能、复位、设定位置*)
0015 Act_02_Control_All_Axis ;
0016
0017 (* Table 异常处理*)
0018 Act_03_Disable_Table;
0019
0020 (*复制NCI TO PLC的结构体, 以方便查看*)
0021 MEMCPY(ADR(aNciToPlc[0]),ADR( aNciChannel[0].NciToPlc), sizeof(aNciToPlc[0]));
0022
0023 (*产生1s脉冲*)
0024 T1(IN:= NOT T2.Q, PT:=T#100MS , Q=>gbPulse1s , ET=>);
0025
0026 T2(IN:=T1.Q , PT:=T#50MS , Q=> , ET=>);
0027

```

其中 aNciToPlc 的定义如下:

```
aNciToPlc : ARRAY[0..nMaxChannel] OF NciChannel_ToPlc;
```

这是因为 TcNci.lib 中的 NCI 接口变量 NciChannelToPlc 与 System Manager 中 NCI 通道的输出变量结构体的元素顺序不一致,

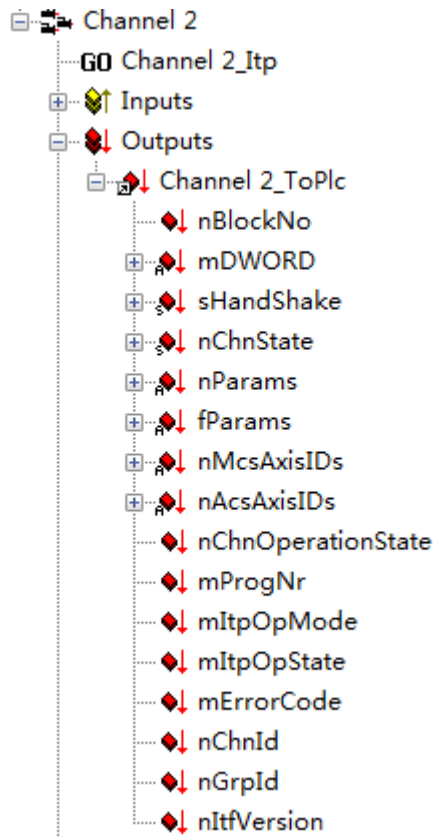
在 TcNci.lib 中的结构体定义如下:

```

TYPE NciChannelToPlc :
STRUCT
  nJobNo      : DWORD;
  nFastMFuncMask : ARRAY[1..5] OF DWORD; (* Mask to evaluate fast M-functions *)
  nHskMFuncNo  : WORD; (* evaluate M-function with handshake *)
  nHskMFuncReq : WORD;
  nHFuncValue  : DINT;
  nSpindleRpm  : WORD;
  nTool        : WORD;
  nReserved1   : ARRAY[37..132] OF BYTE; (* 37..40 nChnState *)
  (* 41..56 nParams *)
  (* 57..88 fParams *)
  (* 89..132 reserved *)
  nLoadedProg  : DWORD; (* loaded program number if exist *)
  nntpMode     : WORD; (* Interpreter mode *)
  nntpState    : WORD; (* Interpreter status *)
  nntpErrCode  : WORD; (* Interpreter-Channel Error Code *)
  nReserved2   : ARRAY[143..150] OF BYTE; (*143..144 still reserved *)
  (*145..146 nChnId *)
  (*147..148 nGrpId *)
  (*149..150 nntfVersion *)
END_STRUCT
END_TYPE

```

而在 System Manager 看到的 NCI 通道到 PLC 的结构体如下:



二者不一致, 导致在 System Manager 中看到的状态信息, 在 PLC 程序中无法一一对应, 所以根据 System Manager 中的变量顺序, 新建了结构体 NciChannel_ToPlc:

TYPE NciChannel_ToPlc :

(*2016.10.23 by LizzyChen*)

STRUCT

```
nJobNo          : UDINT;
nFastMFuncMask  : ARRAY[1..5] OF UDINT;
                 (* Mask to evaluate fast M-functions *)
```

(*sHandShake*)

```
nHskMFuncNo     : WORD;   (* evaluate M-function with handshake *)
nHskMFuncReq    : USINT;
nReserved_1     : BYTE;
nHFuncValue     : DINT;
nSpindleRpm     : WORD;
nTool           : WORD;
```

(*nChnState*)

```
nChnState       : DWORD;
```



```

nParams      : ARRAY[1..4] OF UDINT;
fParams      : ARRAY[1..4] OF LREAL;
nMcsAxisIDs  : ARRAY[1..8] OF USINT;
nAcsAxisIDs  : ARRAY[1..8] OF USINT;

nReserved_2  : ARRAY[1..24] OF BYTE;

nChnOpState  : UDINT;
nProgNr      : UDINT;
nItpOpMode   : WORD;      (* Interpreter mode *)
nItpOpState  : WORD;      (* Interpreter status *)
nItpErrCode  : DWORD;     (* Interpreter-Channel Error Code *)
nChnID       : UINT;
nGrpID       : UINT;
nItfVersion  : UINT;

END_STRUCT
END_TYPE

```

然后在 Pro_Other 中用内存拷贝指令 MEMCPY，将 IO 映射过来的类型为 NciChannelToPlc 结构体转送到与 TSM 中一致的类型为 NciChannel_ToPlc 的结构体中。

```

0009
0010 (*复制NCI TO PLC的结构体，以方便查看*)
0011 MEMCPY(ADR(aNciToPlc[0]),ADR( aNciChannel[0].NciToPlc), SIZEOF(aNciToPlc[0]));
0012

```

14) PTP 轴的群控

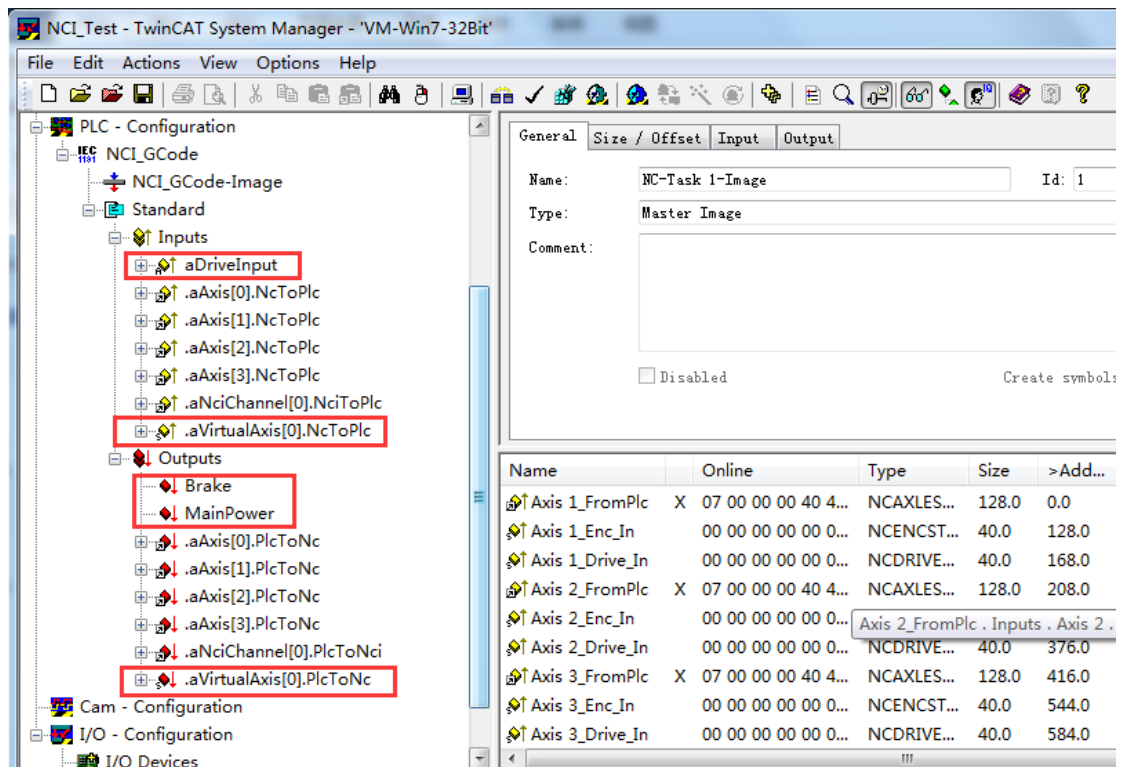
这是因为轴数特别多的时候，在调试界面上单个点击每个轴的接口变量，比如使能、复位、位置置零时，会相当耗时。所以在界面上做了几个群控的按钮，一旦群控模式 (bPtpCtrlAll) 开启，这个几个变量的值就直接赋到每个轴的 Interface 里了。

```

0001 (*群控程序: 所有轴一起使能、复位、设定位置*)
0002 IF bPtpCtrlAll THEN
0003   FOR I:=0 TO nMaxAxis DO
0004     Axis_Interface[I].bEnable:= bEnableAll;
0005     Axis_Interface[I].bReset:= bResetAll;
0006     Axis_Interface[I].bEnableBw:= bEnableAll;
0007     Axis_Interface[I].bEnableFw:= bEnableAll;
0008     Axis_Interface[I].bSetPos:= bSetPosAll;
0009   END_FOR
0010 END_IF
0011
0012
0013
0014
0015

```

3.5.6 System Manager 配置文件



注意：

aDriveInput 和 Brake/MainPower 都是在实际项目中使用，与伺服驱动器硬件相关的变量。用虚轴测试功能时，不必链接。

aVirtualAxis 是辅助轴，类似为了字对齐而在结构体中增加的 Reserve 或者 Dummy 变量，仅仅是为了程序不报错。不必连接 NC 轴。

3.5.7 调试画面

TwinCAT NC PTP 及 NCI 插补通道控制界面
2016.10.23

群控PTP	ID	Ready	故障	使能	Jog +	Jog -	当前位置	点动速度
	1	绿色	0	绿色			99.5	10.0
全部使能	2	绿色	0	绿色			100.0	10.0
全部复位	3	绿色	0	绿色			0.0	10.0
全部置0位	4	绿色	0	绿色			0.0	10.0

G代码文件 C:\TwinCAT\CNC\MDemo.nc

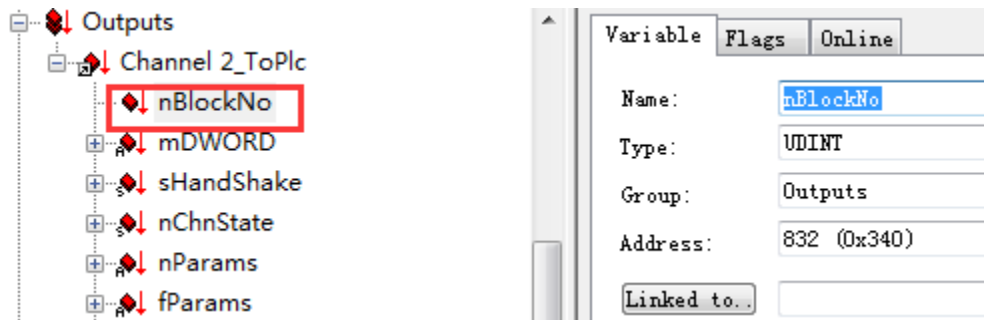
插补使能 1	插补复位	通道运行状态	IsRunning
插补启动	装载G代码 2	未执行指令数	8条
插补停止		当前指令行号	N 30

画面的上半部是 PTP 调试界面，包括群控的使能、复位、位置置 0 按钮。右边的列表中有轴的 Ready、Error、ErrorID 等状态，以及使能、正反向点动 Jog+、Jog-等控制信号。显示当前位置，可单独设置点动速度。

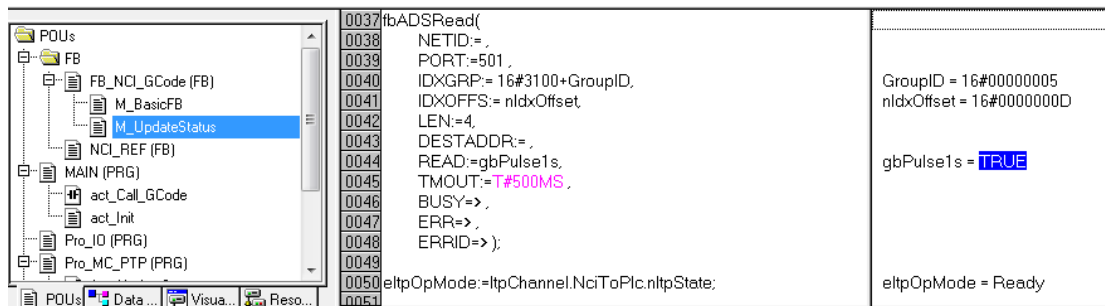
下半部的 NCI 调试界面，除了 5 个命令按钮之外，还可以显示通道运行状态，这是显示的通道 Interface 中的 eItpOpMode 中的值，说明如下：

eItpOpMode : NCI 通道的运行模式，最常见的是 Idle、Ready 和 IsRunning;
组合后装载 G 代码前为 Idle，成功装载 G 代码后启动前为 Ready，启动后结束前为 IsRunning。动作完成后又恢复为 Ready 状态。

当前指令的行号：假如 G 代码行首包含约定的行号，例如 N10、N20 等，执行这样的代码行时，行号就会在这里显示出来。这个行号其实来自 NciToPlc 的第一个变量 nBlockNo:



未执行指令数：指从 G 代码文件预读到 NCI 编译器但还没有执行的 G 代码行数。执行完毕，该值为 0。由于 NCI 缓存的容量（SAF Entry）有限，该值最大为 128。也就是说 NCI 最多缓存 128 条指令。虽然 SVB Entry 中也可以存 64 条，但是测试结果表明，把 SAF Entry 用尽偶尔会引起动作异常，原因有待研究。这个参数不能从 NciToPlc 的结构体中获得，必须用 ADS 通讯从 NCI 设备中读回来，这些代码在 FB_NCI_GCode 的 M_UpdateStatus 中：



3.5.8 测试 NCI 插补功能的操作顺序

- 1) 选择目标控制器，激活 NCI_Test.tsm
- 2) 打开 NCI_GCode.pro，下载到目标 PLC，运行。
- 3) 进入 HMI，查看 G 代码文件，确认控制器上的 CNC 文件路径与之一致
- 4) 使能 NC 轴。使用群控功能，可以节约时间。
- 5) 如有需要可以全部位置置 0
- 6) 插补使能——装载 G 代码——插补启动
- 7) 插补停止，或者插补复位。
- 8) 可以修改 G 代码文件，或者选择其它 G 代码文件。重新执行。

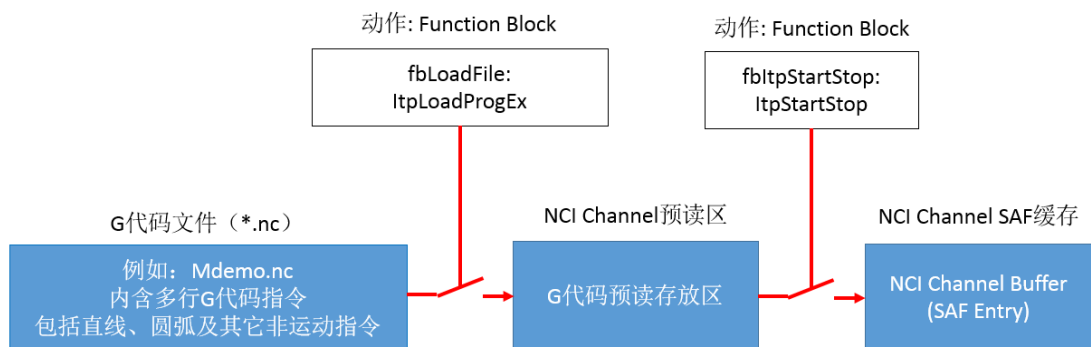
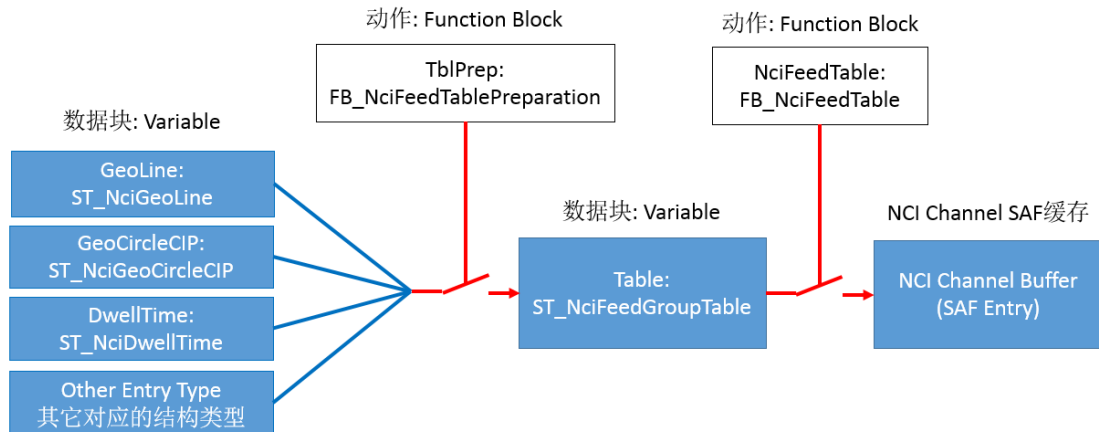
执行 Step 6 和 7 时，观察轴的位置变化和通道的状态、未执行指令数及当前行号的变化。

注意：

停止和复位后必须重新装载 G 代码，更换 G 代码文件也要重新装载。

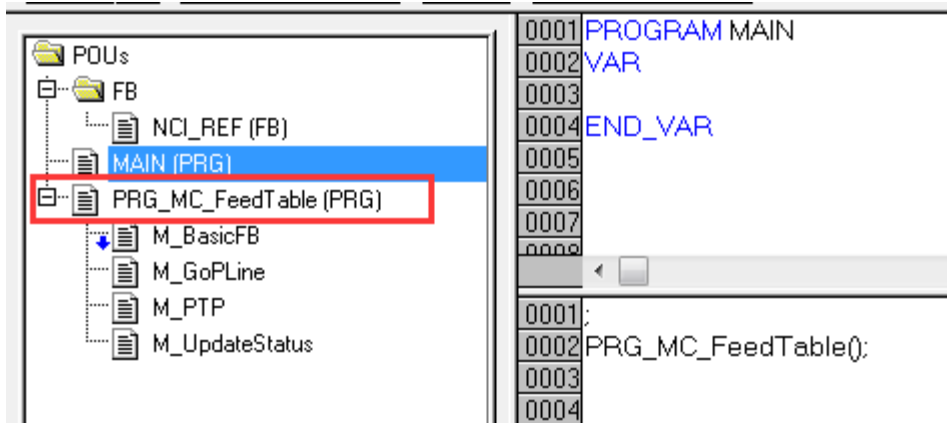
本例程中配了一个用于对比的 G 代码文件 MDemo2.nc，与 MDemo.nc 的唯一区别就是进给速度的不同。用户可以在界面上修改 G 代码文件名，然后重新装载，对比运动效果。

4 使用 FeedTable 的插补运动项目



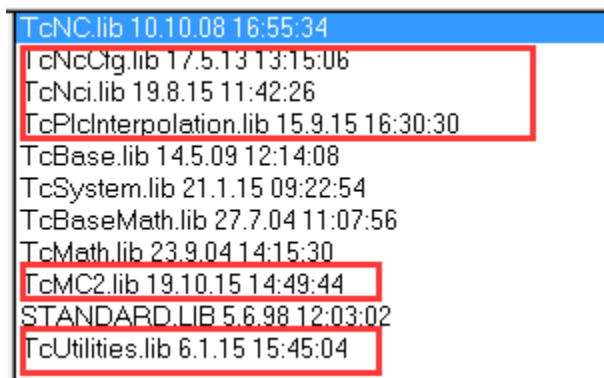
4.1 在 PLC 中新建 NCI 程序

为了方便导出到其它程序使用，所以代码并不写在 MAIN 中，而是另外取名，比如本例中叫做 PRG_MC_FeedTable。需要在 MAIN 中引用 PRG_MC_FeedTable，程序才会执行。如图所示：



4.1.1 准备工作

- 1) 准备工作 1: 新建 Pro 项目，保存命名为“NCI_FeedTable_New.pro”，并添加 PTP 程序所需要的库 TcMc2.lib 以及 NCI 程序所需要的库 TcNcCfg.lib、TcNci.lib、TcPlcInterpolation.lib。



手动加了这 4 个库之后，图中的其它库都会自动添加进来。

- 2) 准备工作 2: 新建功能块 NCI_Ref

这一步不是必须的，而是为了符合 TcMc2.lib 的习惯，将通道的控制对象约定为一个变量。在 TcMc2.lib 的 PTP 指令中，所有 FB 的控制对象都是 Axis_Ref，所以在 NCI 程序中我们定义一个 FB 叫做 NCI_Ref，如图所示：

```

FUNCTION_BLOCK NCI_REF
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR
    PlcToNci    AT    %Q*    :    NciChannelFromPlc;
    NciToPlc    AT    %I*    :    NciChannelToPlc;
END_VAR

```

变量名可以修改，而变量类型是所有 NCI 相关库中定义的，不能修改。

3) 准备工作 3：新建结构体 NCI_SingleEntry

这一步不是必须的，而是为了统一直线和圆弧插补的准备数据，填充插补指令时总是从这个结构中取数据。如图所示：

```

0001 TYPE NCI_SingleEntry :
0002 STRUCT
0003     SyncVelo:REAL;
0004     SyncAcc:REAL;
0005     fX:REAL;
0006     fY:REAL;
0007     fZ:REAL;
0008
0009     fQ1:REAL;
0010     fQ2:REAL;
0011     fQ3:REAL;
0012     fQ4:REAL;
0013     fQ5:REAL;
0014
0015     fCenterX:REAL;
0016     fCenterY:REAL;
0017     fCenterZ:REAL;
0018     iPlane:UDINT;
0019     Reserve1:UDINT;
0020
0021     Reserve2:UDINT;
0022 END_STRUCT
0023 END_TYPE
0024

```

圆弧还是直线插补，用 `iPlane` 来区分。如果该变量为 0，表示直线插补，否则为圆弧插补。这个逻辑在 4.1.3 编写基本代码的第 2 项“准备待填充到 Table 的插补指令的数据”的 `M_GoPLine` 子程序中实现。

4.1.2 新建 FeedTable 控制的基本 FB 及其接口变量

在 PRG_MC_FeedTable 的局部变量中声明以下变量或者功能块实例。

基本功能块 FB

通道控制的基本动作包括：

- 组合通道, CfgBuildExt3DGroup;
- 解散通道: CfgReconfigGroup;
- 插补指令准备: FB_NciFeedTablePreparation;
- 插补指令填充: FB_NciFeedTable;
- 插补运动启停: ItpStartStop;
- 通道复位: ItpResetEx2

使用过 G 代码方式的用户可以这样理解, 插补指令准备 (FB_NciFeedTablePreparation) 相当于装载 G 代码文件, 而插补指令填充 (FB_NciFeedTable) 相当于插补运动启动, 如果运动过程中要停止, 仍然是通过插补运动启停功能块 (ItpStartStop)

另外, 通道的速度倍率是在 PLC 的接口变量 NciChannelFromPlc 中周期性刷新的, 既可以直接 PLC 赋值, 也可以用函数 (FC) ItpSetOverridePercent 来控制。

为此, 我们新建以下局部变量:

VAR

(*NCI 通道的 Axis 和 Channel, 与 TwinCAT NC 的接口变量*)

AxisX : AXIS_REF;

AxisY : AXIS_REF;

ItpChannel : NCI_REF;

(*基本动作功能块: 通道组合, 通道解散, 装载 G 代码文件, 启动停止, 复位*)

CfgBuildExt3DGroup : CfgBuildExt3DGroup;

fbClearGrp : CfgReconfigGroup;

fbItpStartStop : ItpStartStop;

fbItpReset : ItpResetEx2;

TblPrep : FB_NciFeedTablePreparation;

NCI_Entry : NCI_SingleEntry ;

(*自定义结构体: 准备赋给待填充指令 GeoCircleCIP 和 GeoLine 的数据*)

GeoCircleCIP : ST_NciGeoCircleCIP;

(*待填充到 Table 的圆弧插补指令*)

GeoLine : ST_NciGeoLine;

(*待填充到 Table 的直线插补指令*)


```

NciFeedTable      :   FB_NciFeedTable;
Table             :   ST_NciFeedGroupTable;
                  (*插补通道要执行的插补指令序列*)

TableDisplayIndex :   UDINT := 1;
                  (*插补指令在 Table 中的索引号*)
rOverride         :   LREAL := 100;
                  (*插补运动的倍率*)

(*通道 ID 和轴组 ID*)
GroupID           :   UDINT;
ChannelID        :   UDINT;

(*仅在数据准备的上升沿, 才把 Entry 中的数据填充到 Table*)
bTablePrepare     :   BOOL;
fbRTrigTablePrepare :   R_TRIG;
END_VAR

```

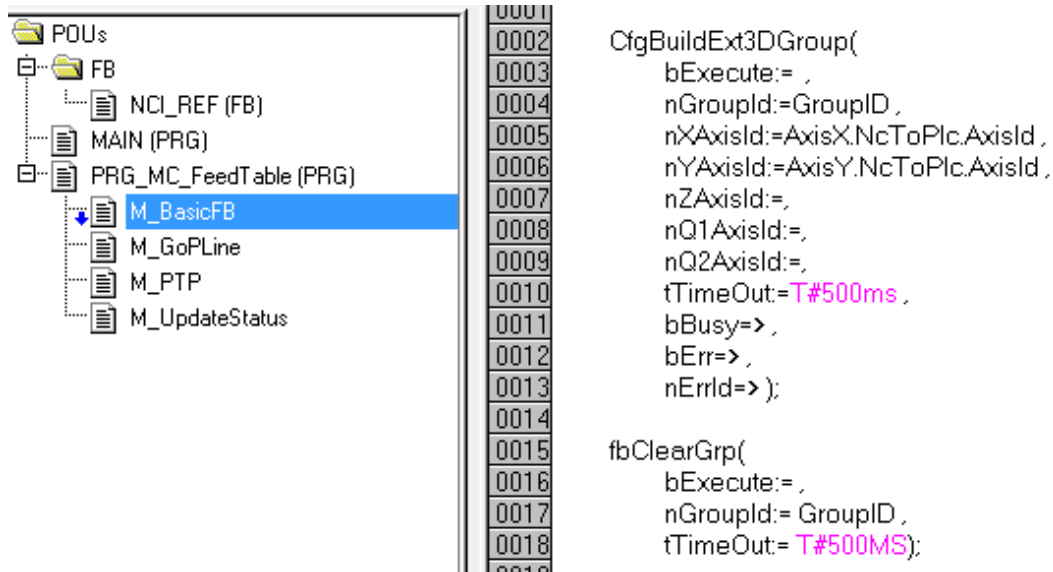
其中 NCI_Entry 的类型 NCI_SingleEntry 在 4.1.1 准备工作的第 3 项中定义。

4.1.3 编写基本代码

基本代码分为控制命令的执行, 和通道状态的刷新。由于轴的使能属于 PTP 代码, 为了示例简单, PTP 轴使能的代码也写在这里。

1) 控制命令的执行

实际上, 控制命令的执行就是罗列 5 个 FB 的 Instance, 但是把每个 FB 的 bExecute 都悬空不填, 这样就可以在程序的其它地方控制这些 bExecute 条件了。为了程序清晰, 把以上代码放到 Action 里面, 名为 M_BasicFB。如图所示:



```

0001 CfgBuildExt3DGroup(
0002     bExecute:= ,
0003     nGroupId:=GroupID ,
0004     nXAxisId:=AxisX.NcToPlc.AxisId ,
0005     nYAxisId:=AxisY.NcToPlc.AxisId ,
0006     nZAxisId:=,
0007     nQ1AxisId:=,
0008     nQ2AxisId:=,
0009     tTimeOut:=T#500ms ,
0010     bBusy=> ,
0011     bErr=> ,
0012     nErrId=> );
0013
0014
0015 fbClearGrp(
0016     bExecute:= ,
0017     nGroupId:= GroupID ,
0018     tTimeOut:= T#500MS);
0019

```

完整的基本代码为:

```

CfgBuildExt3DGroup(
    bExecute:= ,
    nGroupId:=GroupID ,
    nXAxisId:=AxisX.NcToPlc.AxisId ,
    nYAxisId:=AxisY.NcToPlc.AxisId ,
    nZAxisId:=,
    nQ1AxisId:=,
    nQ2AxisId:=,
    tTimeOut:=T#500ms ,
    bBusy=> ,
    bErr=> ,
    nErrId=> );

```

```

fbClearGrp(
    bExecute:= ,
    nGroupId:= GroupID ,
    tTimeOut:= T#500MS);

```

```

fbItpStartStop(
    bStart:= ,
    bStop:= ,
    nChnId:=ChannelID,
    tTimeOut:=T#500MS ,
    bBusy=> ,
    bErr=> ,
    nErrId=> );

```

```

fbItpReset(
    bExecute:= ,
    tTimeOut:= T#500MS,
    sNciToPlc:= ItpChannel.NciToPlc,
    bBusy=> ,
    bErr=> ,
    nErrId=> );

```

```

TblPrep(
    nEntryType := ,
    pEntry := ,
    bResetTable:= ,
    stFeedGroupTable:= Table,
    nFilledRows=> ,
    bError      => ,
    nErrorId    => );

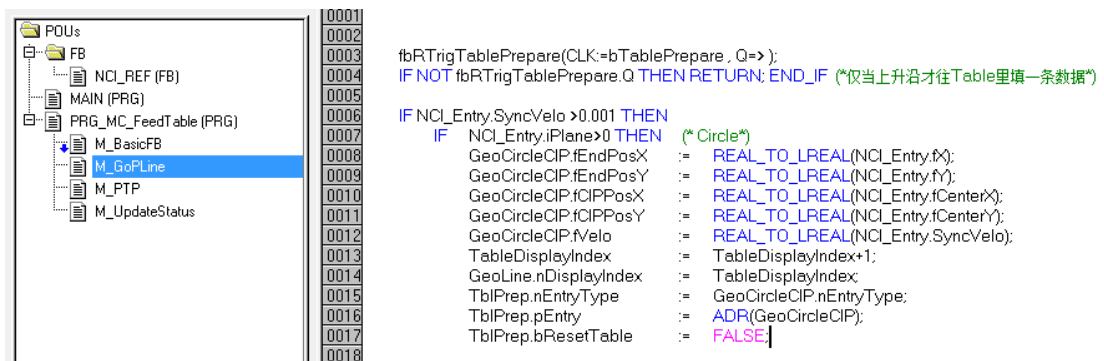
```

```

NciFeedTable(
    bExecute:= ,
    bReset:= ,
    bLogFeederEntries:= FALSE ,
    stFeedGroupTable:= Table,
    stNciToPlc:=ItpChannel.NciToPlc );

```

- 2) 准备待填充到 Table 的插补指令的数据
代码放到 Action 里面，名为 M_GoPLine:



```

0001
0002
0003 fbRTrigTablePrepare(CLK:=bTablePrepare , Q=> );
0004 IF NOT fbRTrigTablePrepare.Q THEN RETURN; END_IF (*仅当上升沿才往Table里填一条数据*)
0005
0006 IF NCI_Entry.SyncVelo > 0.001 THEN
0007     IF NCI_Entry.iPlane > 0 THEN (* Circle*)
0008         GeoCircleCIP.fEndPosX := REAL_TO_LREAL(NCI_Entry.fX);
0009         GeoCircleCIP.fEndPosY := REAL_TO_LREAL(NCI_Entry.fY);
0010         GeoCircleCIP.fCIPPosX := REAL_TO_LREAL(NCI_Entry.fCenterX);
0011         GeoCircleCIP.fCIPPosY := REAL_TO_LREAL(NCI_Entry.fCenterY);
0012         GeoCircleCIP.fVelo := REAL_TO_LREAL(NCI_Entry.SyncVelo);
0013         TableDisplayIndex := TableDisplayIndex+1;
0014         GeoLine.nDisplayIndex := TableDisplayIndex;
0015         TblPrep.nEntryType := GeoCircleCIP.nEntryType;
0016         TblPrep.pEntry := ADR(GeoCircleCIP);
0017         TblPrep.bResetTable := FALSE;
0018

```

代码如下:

```

fbRTrigTablePrepare(CLK:=bTablePrepare , Q=> );
IF NOT fbRTrigTablePrepare.Q THEN RETURN; END_IF (*仅当上升沿才往
Table 里填一条数据*)

```

```

IF NCI_Entry.SyncVelo >0.001 THEN
  IF NCI_Entry.iPlane>0 THEN    (* Circle*)
    GeoCircleCIP.fEndPosX:= REAL_TO_LREAL(NCI_Entry.fX);
    GeoCircleCIP.fEndPosY:= REAL_TO_LREAL(NCI_Entry.fY);
    GeoCircleCIP.fCIPPosX:= REAL_TO_LREAL(NCI_Entry.fCenterX);
    GeoCircleCIP.fCIPPosY:= REAL_TO_LREAL(NCI_Entry.fCenterY);
    GeoCircleCIP.fVelo    := REAL_TO_LREAL(NCI_Entry.SyncVelo);
    TableDisplayIndex    := TableDisplayIndex+1;
    GeoLine.nDisplayIndex := TableDisplayIndex;
    TblPrep.nEntryType   := GeoCircleCIP.nEntryType;
    TblPrep.pEntry       := ADR(GeoCircleCIP);
    TblPrep.bResetTable  := FALSE;

  ELSE      (* Line*)
    GeoLine.fEndPosX    := REAL_TO_LREAL(NCI_Entry.fX);
    GeoLine.fEndPosY    := REAL_TO_LREAL(NCI_Entry.fY);
    GeoLine.fEndPosZ    := REAL_TO_LREAL(NCI_Entry.fZ);
    GeoLine.fEndPosQ1   := REAL_TO_LREAL(NCI_Entry.fQ1);
    GeoLine.fEndPosQ2   := REAL_TO_LREAL(NCI_Entry.fQ2);

    GeoLine.fVelo       := REAL_TO_LREAL(NCI_Entry.SyncVelo);
    TableDisplayIndex   := TableDisplayIndex+1;
    GeoLine.nDisplayIndex := TableDisplayIndex;

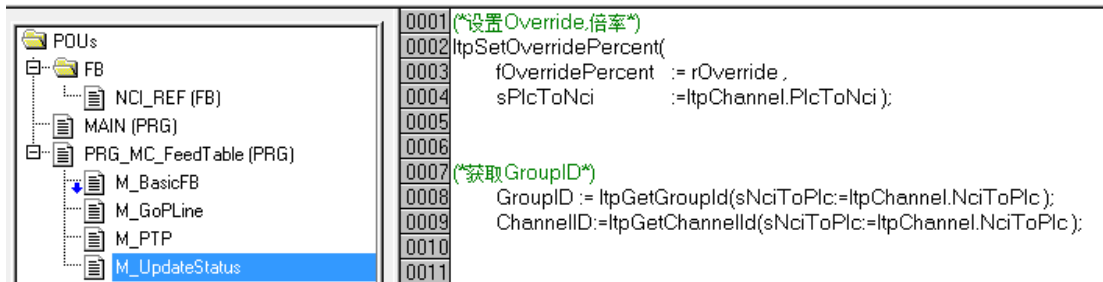
    TblPrep.nEntryType := GeoLine.nEntryType;
    TblPrep.pEntry     := ADR(GeoLine);
    TblPrep.bResetTable := FALSE;

  END_IF
END_IF

```

3) 通道状态刷新

为了尽快看到 PLC 控制 NCI 轴的结果，本例并没有完整的状态，但是可以先设置这个功能的 Action，以后再增加状态变量和相应的代码。现在，只是刷新倍率设置和通道及轴组 ID 号。代码放到 Action 里面，名为 M_UpdateStatus:



代码如下:

(*设置 Override,倍率*)

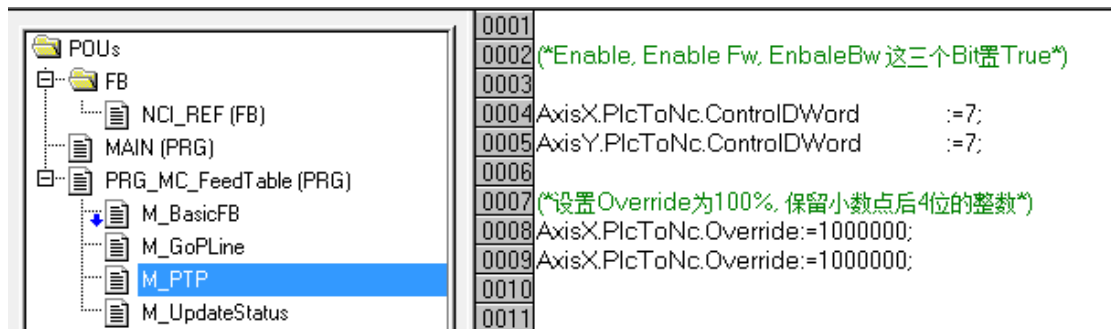
```
ItpSetOverridePercent(
    fOverridePercent := rOverride ,
    sPlcToNci       :=ItpChannel.PlcToNci );
```

(*获取 GroupID*)

```
GroupID := ItpGetGroupId(sNciToPlc:=ItpChannel.NciToPlc );
ChannelID:=ItpGetChannelId(sNciToPlc:=ItpChannel.NciToPlc );
```

4) PTP 命令, 使能和 Override 设置

代码放到 Action 里面, 名为 M_PTP, 如图所示:



代码如下:

(*Enable, Enable Fw, EnbaleBw 这三个 Bit 置 True*)

```
AxisX.PlcToNc.ControlIDWord :=7;
AxisY.PlcToNc.ControlIDWord :=7;
```

(*设置 Override 为 100%, 保留小数点后 4 位的整数*)

```
AxisX.PlcToNc.Override:=1000000;
AxisX.PlcToNc.Override:=1000000;
```

4.1.4 编写 FeedTable 通道控制 FB 的触发逻辑

为了直观的感受各个 FB 动作的先后顺序，以及相互的关联，最简单的触发方式就是手动。所以我们可以不用变量，而是直接手动强制 FB 的 bExecute 变量来触发控制命令。

为此，先把这些 bExecute 罗列在 NCI 程序代码区，比 Login 之后展开 FB 实例去找变量要方便快捷。

(*准备手动控制 NC 通道的触发命令*)

```
rOverride;
CfgBuildExt3DGroup.bExecute;
fbClearGrp.bExecute;
fbItpReset.bExecute;

NCI_Entry.fX;
NCI_Entry.fY;
NCI_Entry.SyncVelo;

bTablePrepare;

TblPrep.bResetTable;
NciFeedTable.bExecute;
```

另外，NCI 程序代码区还要增加前面所建的 4 个 Action 的引用：

(*引用 4 个 Action，实现不同的功能*)

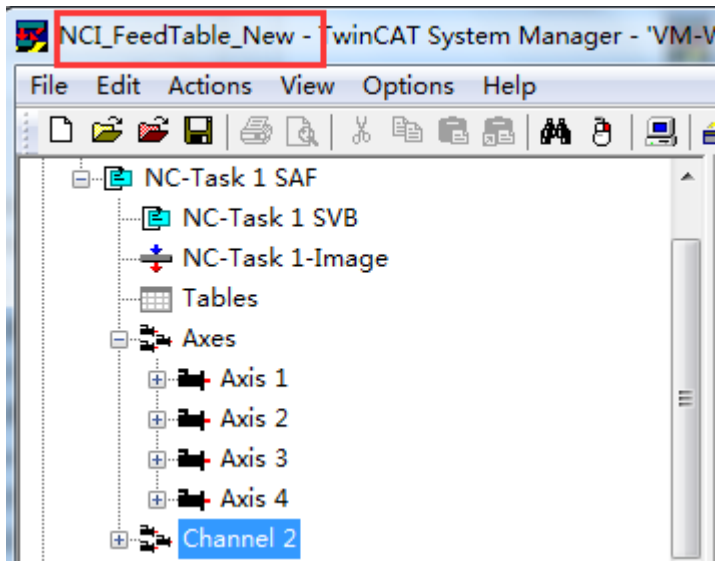
```
M_PTP;           (*轴使能*)
M_BasicFB;       (*NCI 通道控制*)
M_UpdateStatus; (*NCI 通道状态刷新*)
M_GoPLine;       (*数据填充*)
```

至此，控制 NCI 通道的基本程序就写成了。

然后就可以编译了，如果没有 Error 报错，这部份工作就算完成了。

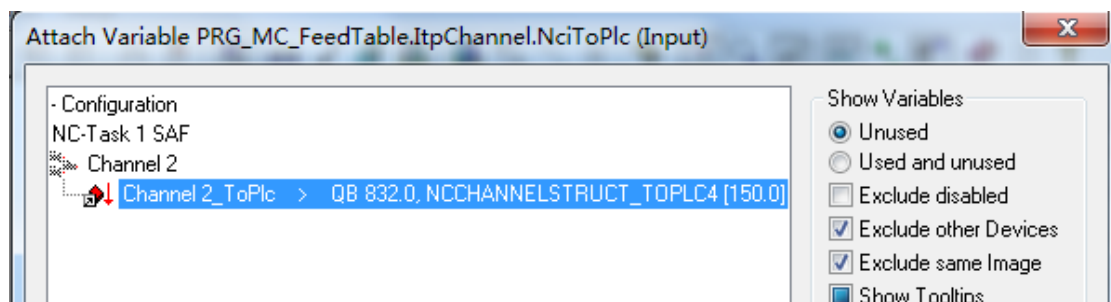
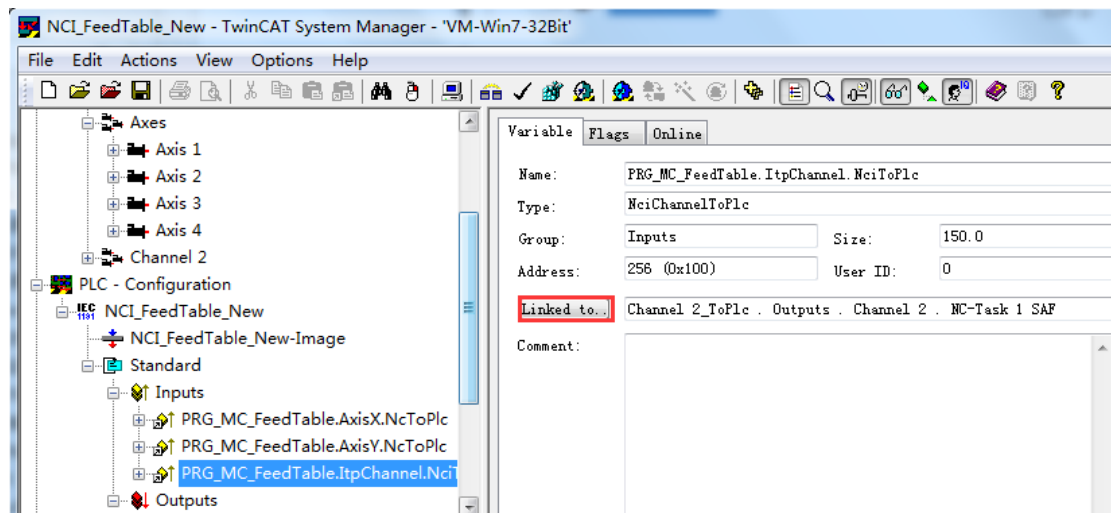
4.2 在 System Manager 中引用 NCI 程序

- 1) 准备工作：在 System Manger 中打开第 2 章的例程 NCI_Test.tsm，另存为 NCI_FeedTable_new.tsm。



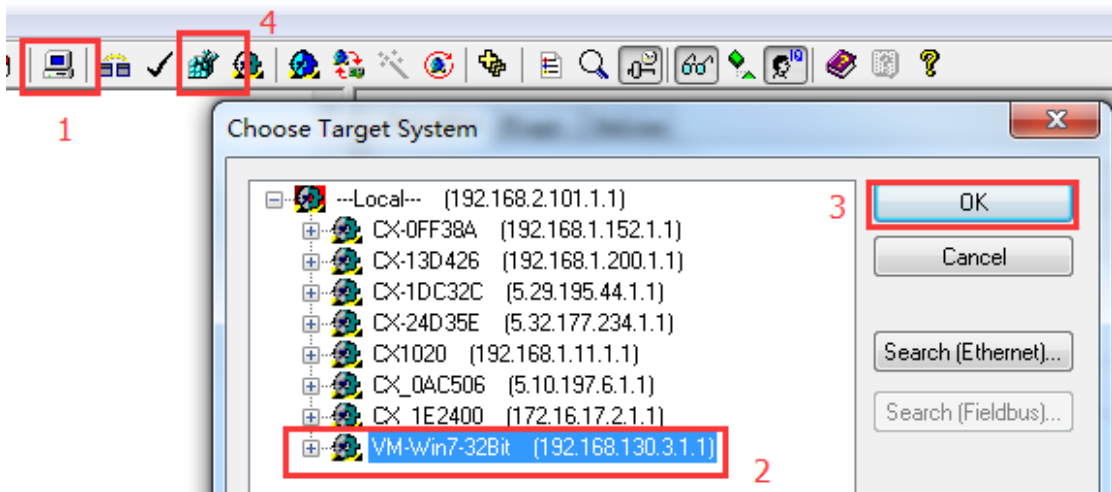
2) 导入 PLC 程序，并链接 Axis 及 Channel 变量

Axis 的链接对于 PTP 的用户来说已经很熟悉了，NCI 的链接也是同样道理。Channel 的接口变量是一对 150 字节的结构：NciChannelFromPlc 和 NciChannelToPlc。



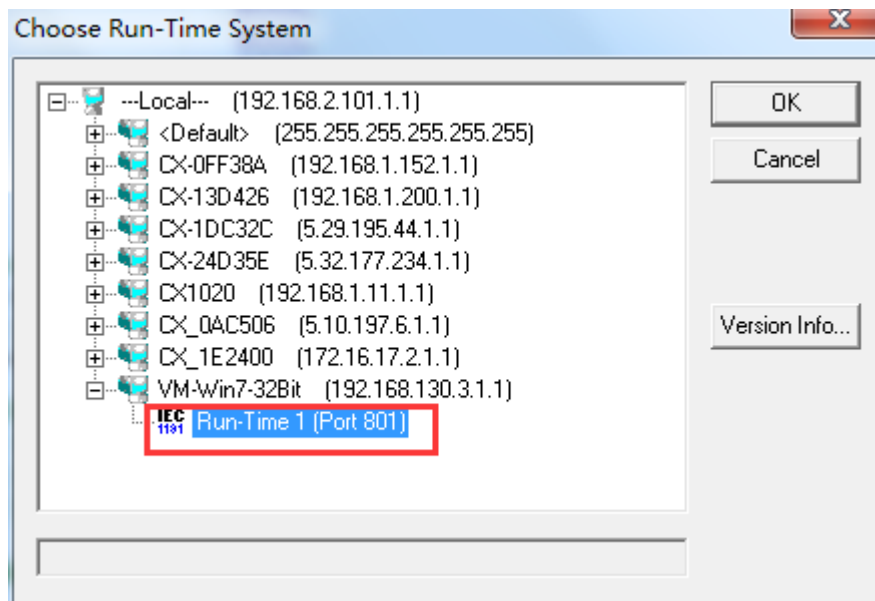
最简单的插补运动只有 X 和 Y 轴，所以其它轴可以不必链接。

3) 选中目标系统，激活配置。



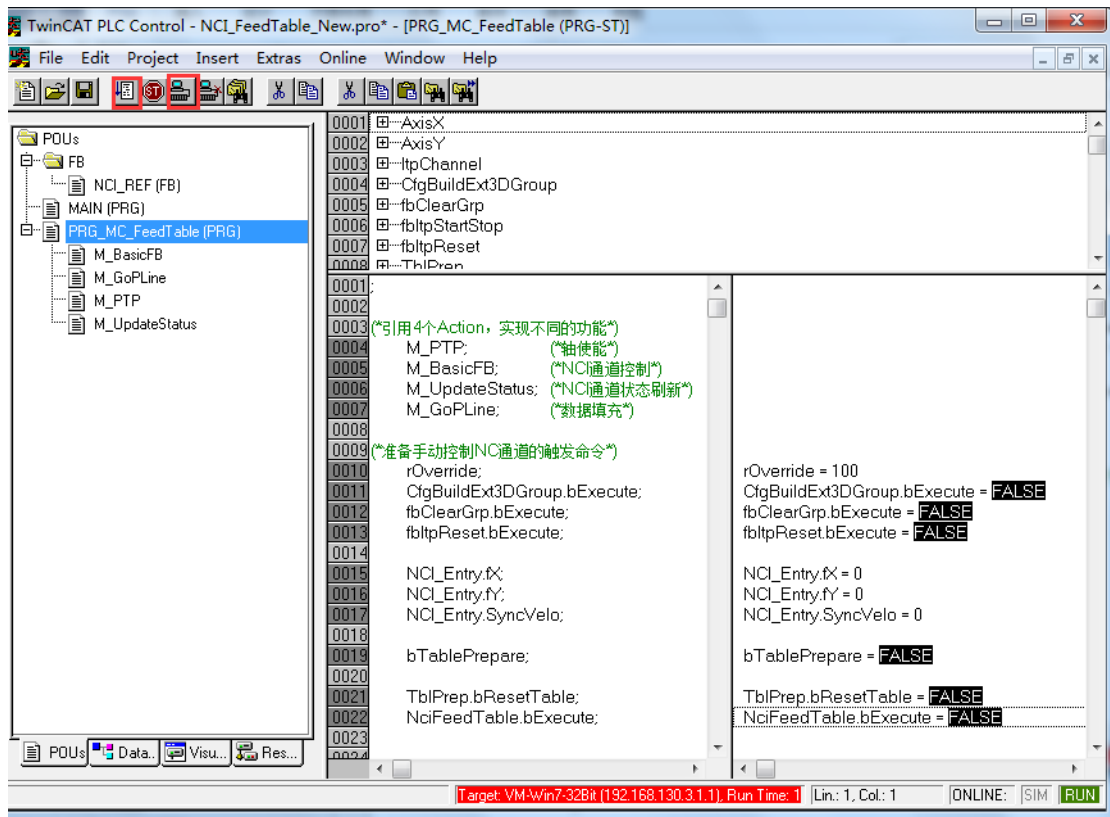
4.3 PLC 程序下载运行，强制变量 bExecute 执行各种指令

1) Choose Runtime System

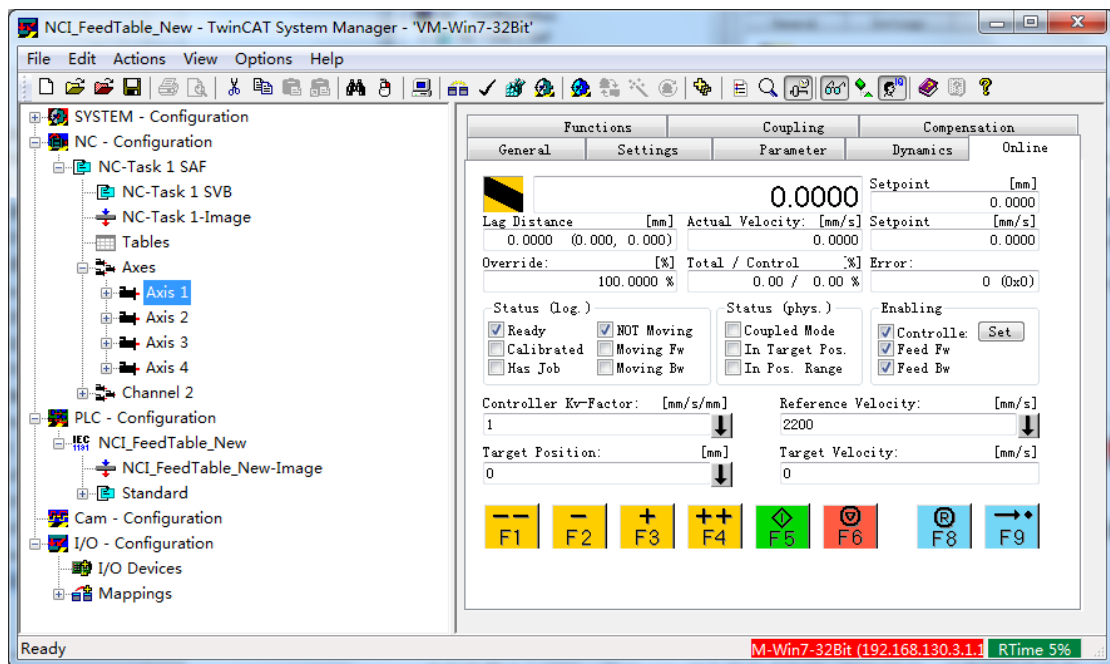


2) 程序运行之初，NC 轴的状态

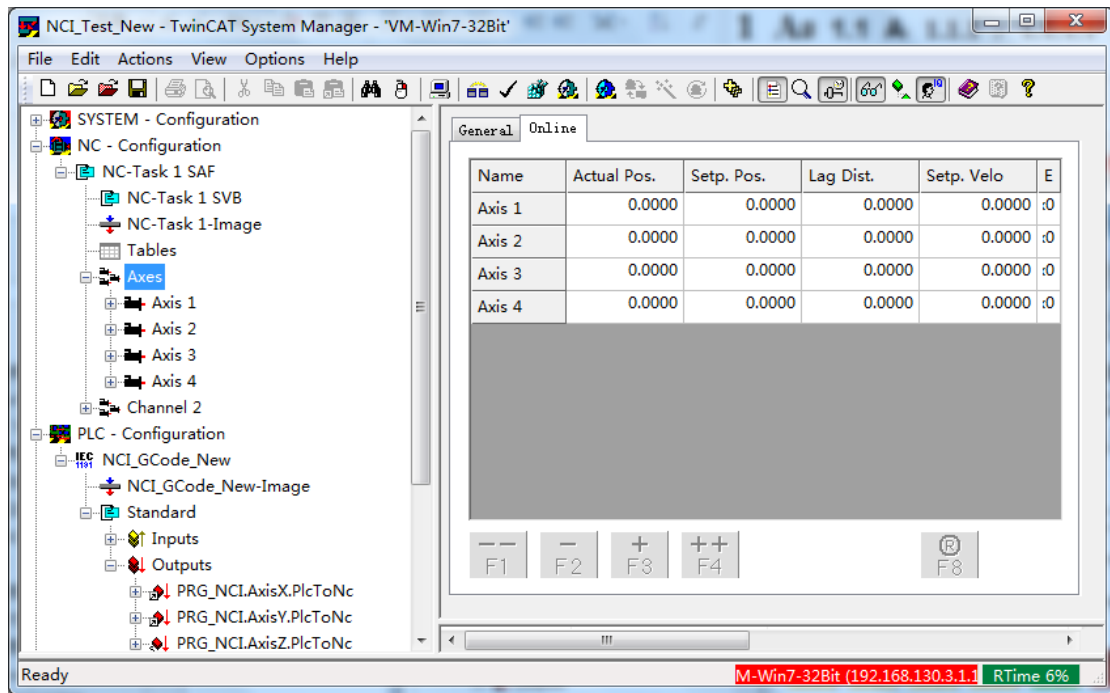
Login 并 Run，定位到 PRG_MC_FeedTable 的代码区



可以看 Axis X 和 Axis Y 所对应的轴 1、轴 2 已经使能了。



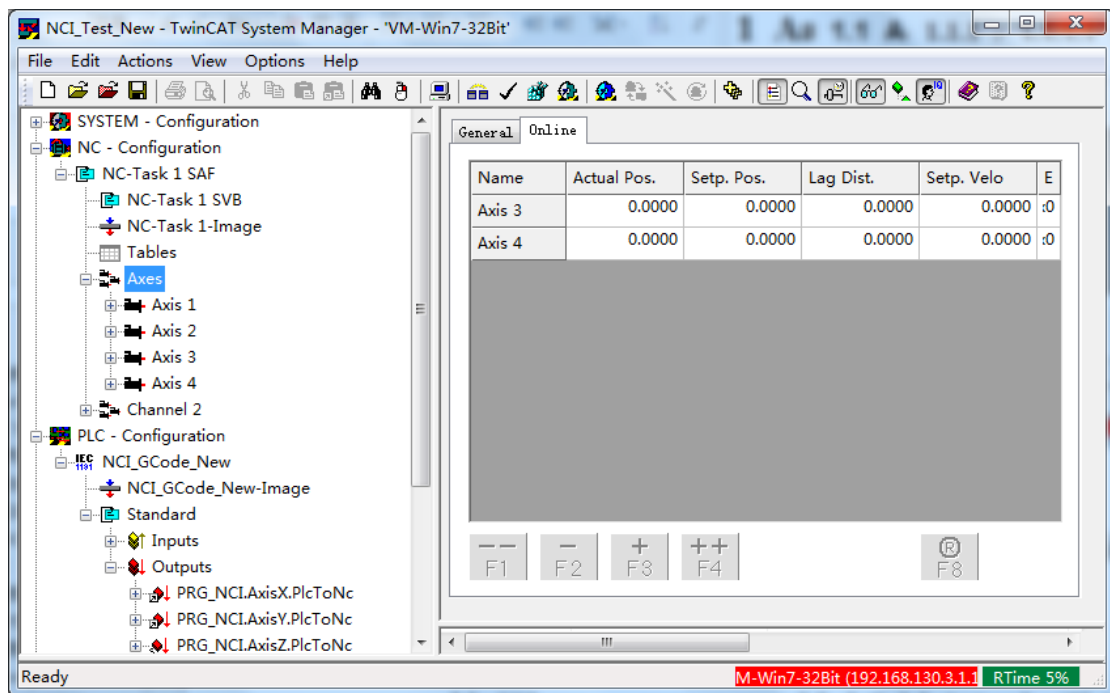
并且各轴是处在 PTP 状态下，可以从 Axes 树形结构的 Online 页面看到全部轴的信息：



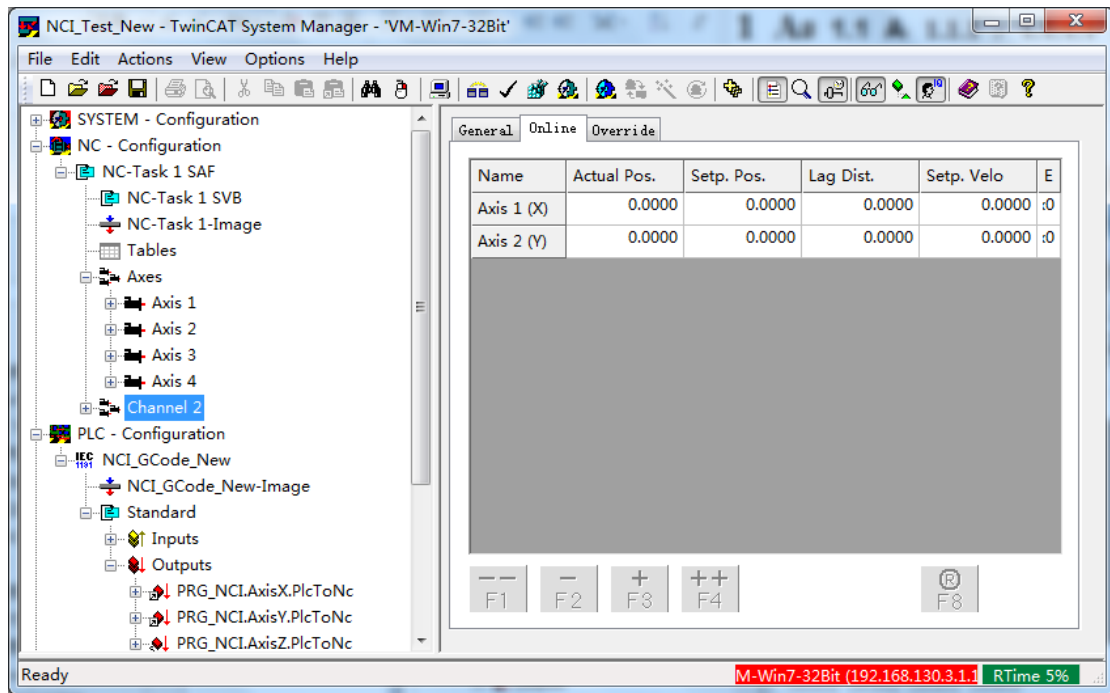
3) 组合通道

0009 CfgBuildExt3DGroup.bExecute; CfgBuildExt3DGroup.bExecute = TRUE

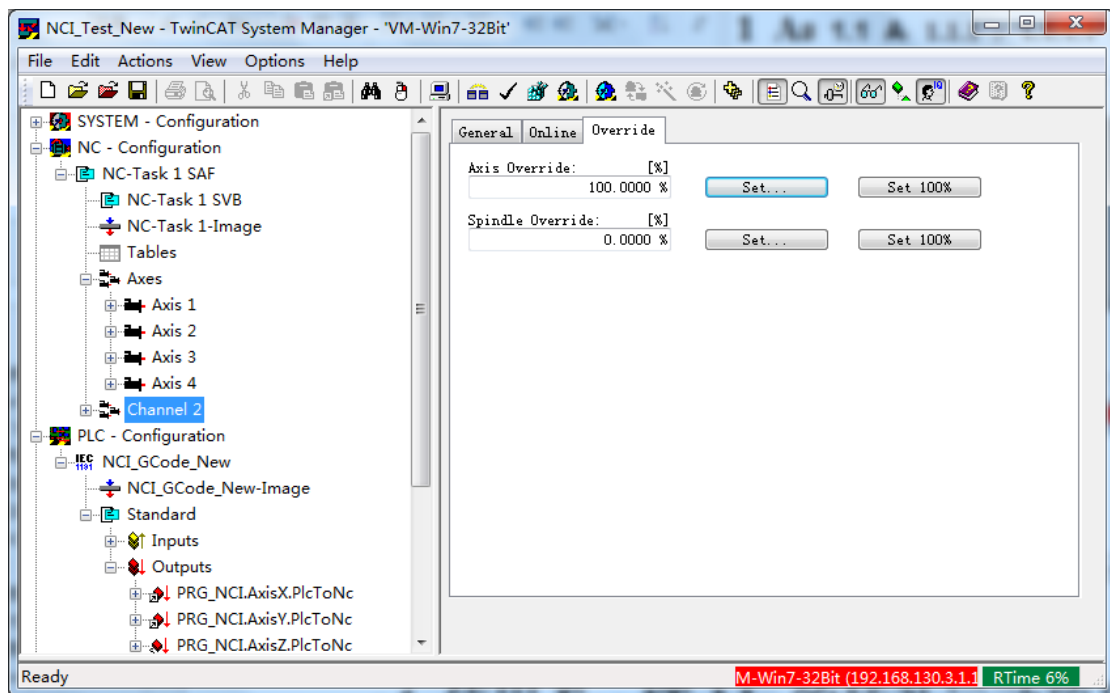
Axes 中少了轴 1 和轴 2 的信息:



而 Channel 中有了轴 1 和轴 2 的信息:



并且，通道的 Override 也成功设置为 100%了。



这表明这两个轴现在归 NCI 通道控制，不能再做 PTP 动作了。

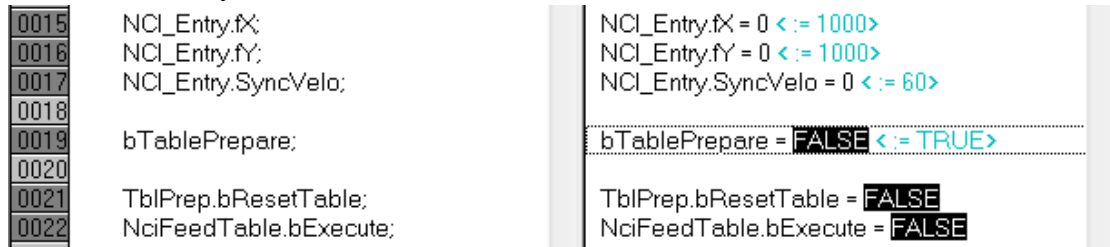
4) 复位 Table

由于不清楚 Table 中是否有以前填充的数据，所以最保险的做法是，往里面填充数据之前，总是执行一下复位动作，即 `TbiPrep.bResetTable` 置 `True`。注意，如果要修改 `NCI_Entry` 的终点位置，重新开始插补动作，也要求 `TbiPrep.bResetTable` 置 `True`，让 Table 清空，

然后置 False，再重复“准备数据——启动运行”。

5) 准备填充数据

在用变量 bPrepareTable 触发填充数据之前，NCI_Entry 中应该填好目标位置(Fx、fY)和进给速度 (SyncVelo)。

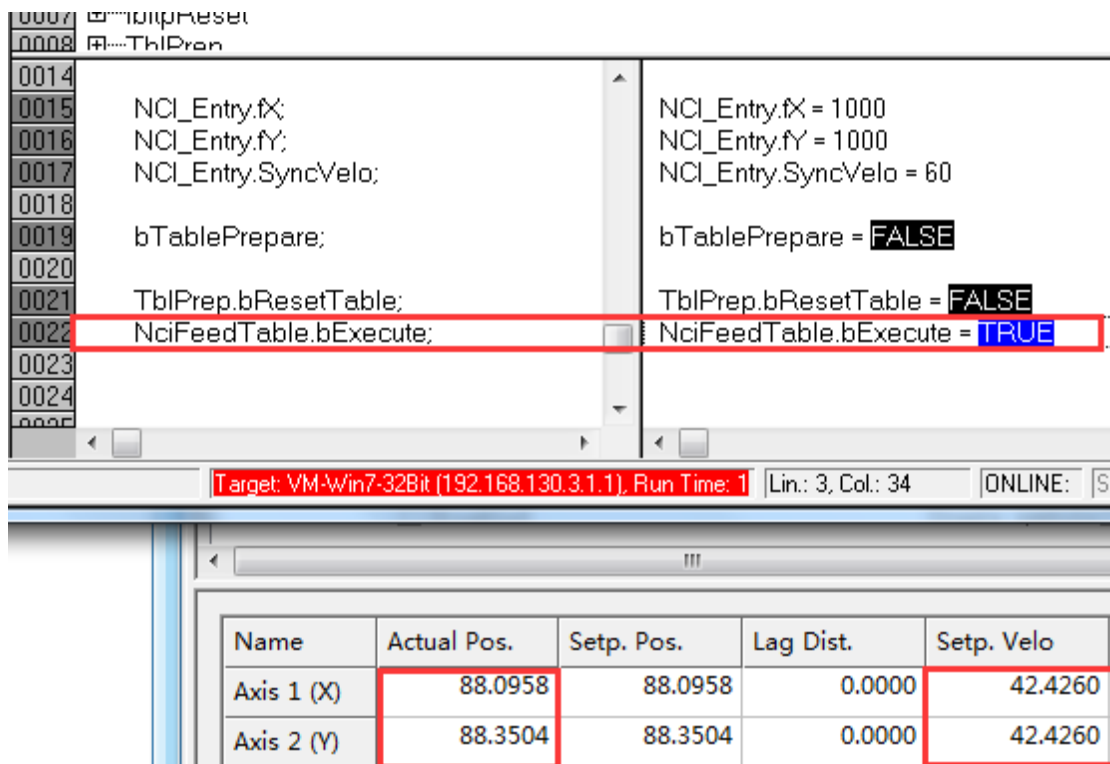


双击变量，准备以上数据，然后按“Ctrl+F7”，在线强制变量，把这些值写入控制器。

6) 通道启动运行

Group 状态为 Ready，表示插补通道已经储存了一些数据，可以运行。

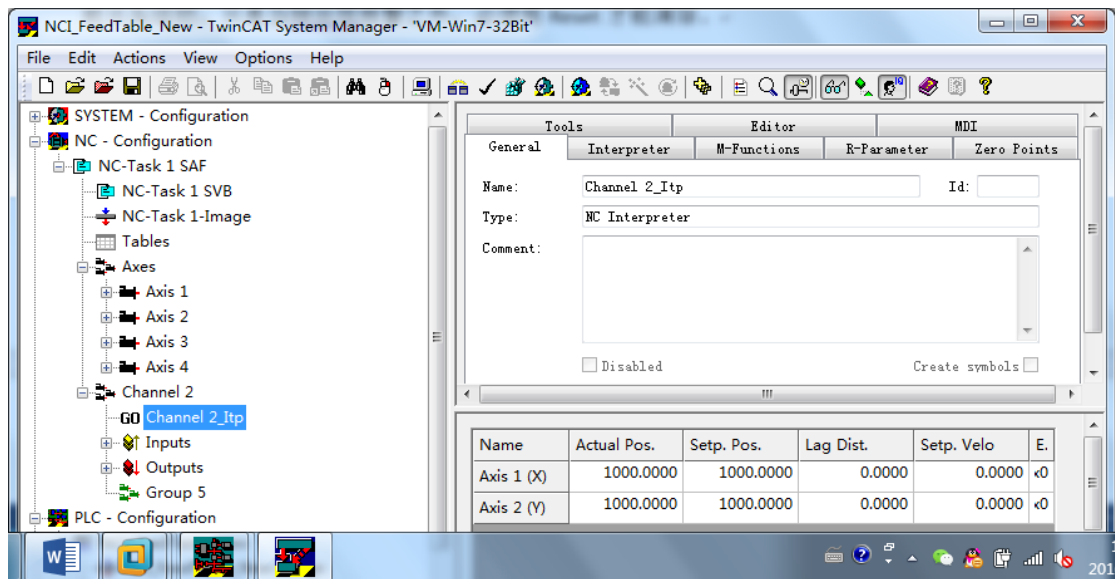
把 NciFeedTable.bExecute 强制为 True，插补通道就开始运动了，如图所示：



X、Y 轴的设定速度为什么都是 42.426 呢？这是由于 X、Y 轴在动作之前的位置都是 0，而 NCI_Entry 中设定的 SyncVelo 是 60，指合成速度 60mm，所以两轴速度应该是 $60/1.414=42.4286$ 。由于 NC 中采用 LReal 来运算，所以根号 2 的值会保留更多有效位数，运算的结果更准确。

最终 Axis 1 (X) 和 Axis 2 (Y) 都停在了位置 1000，这是由于刚才填入的插补指令终点

坐标为 fX 是 1000, fY 也是 1000。



7) 通道停止、复位

停止和复位要用录屏软件才能记录效果，用户可以自行测试。

停止和复位对通道的影响是类似的，当前运动马上停止。之后必须重新装载 G 代码才能从头运动。只是当轴出现报警之外，必须用 Reset 才能清除。

组合成通道的 NCI 轴，无法用 PTP 功能块 MC_Reset 进行单个轴的复位。

如果是 NCI 动作的过程中只是要暂停再继续运动，最简单的方式是，Override 设置为 0，需要继续运动时再恢复为 100%。

8) 通道解散

Name	Actual Pos.	Setp. Pos.	Lag Dist.	Setp. Velo	E.
Axis 1 (X)	1000.0000	1000.0000	0.0000	0.0000	<0
Axis 2 (Y)	1000.0000	1000.0000	0.0000	0.0000	<0

4.4 FeedTable 控制 FB 封装示例：FB_MC_FeedTable

\配套例程\第 4 章 使用 FeedTable 的 NCI 项目\Demo Self Defined FeedTable FB

下册补充 NCI ▶ 配套例程 ▶ 第 4 章 使用FeedTable的NCI项目 ▶ Demo Self Defined FeedTable FB ▶

名称	修改日期	类型	大小
History	2016/10/30 15:15	文件夹	
LatestVersion	2016/10/30 15:16	文件夹	
NCI_FeedTable.BAK	2016/10/30 15:13	BAK 文件	464 KB
NCI_FeedTable	2016/10/30 15:15	PRO 文件	464 KB
NCI_FeedTable____r.ci	2016/10/30 15:15	CI 文件	2,498 KB
NCI_FeedTable_SimuPC	2016/10/30 15:16	TSM 文件	244 KB
NCI_FeedTable_SimuPC.tsm.bak	2016/10/30 15:15	BAK 文件	243 KB

例程用到 Lib 在上一层目录：

下册补充 NCI ▶ 配套例程 ▶ 第 4 章 使用FeedTable的NCI项目 ▶

名称	修改日期	类型
Demo Self Defined FeedTable FB	2016/10/30 15:16	文件夹
First FeedtTable Pro	2016/10/30 14:02	文件夹
LIB	2016/10/23 18:08	文件夹

这个例程是在前面这个“\First FeedtTable Pro\ NCI_FeedTable_New.pro”的基础上，补充了以下功能：

将 PRG_MC_FeedTable 封装成 FB_MC_FeedTable，

并完善了通道状态信息，以及缓存插补指令的数量。

引用 FB 做 NCI 控制时还集成了 PTP 基本功能。

轴和通道的控制信号和状态信息都分别放在 Interface 的结构体中。

为了循环测试插补动作，增加了目标位置自动赋值的程序 pro_MC_FeedTable

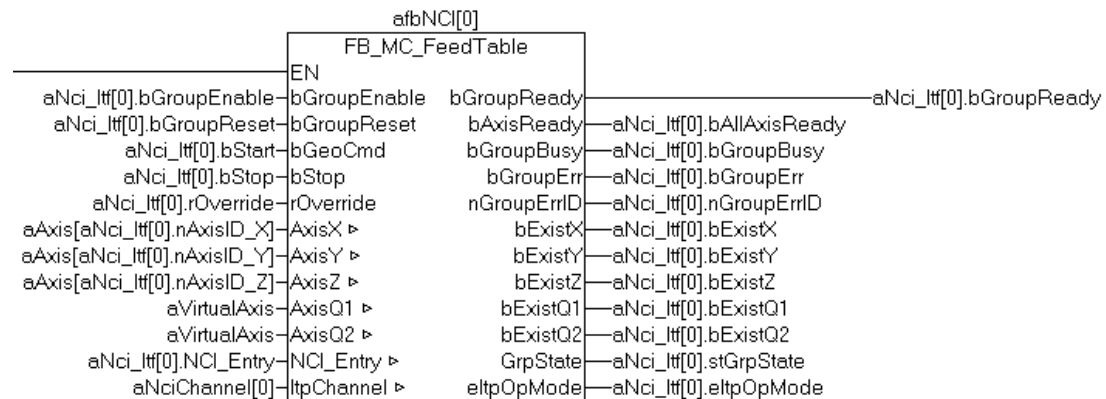
增加了调试画面。

注意：这个封装的 FB 仅供参考，用户也可以集成到自己的项目中使用。而本节除了 FB 源代码之外的所有程序和说明，都仅仅是为了演示例程，与实际应用项目无关。

以下为这个例程的说明

4.4.1 功能块的调用

在 pro_MC_FeedTable 程序的 Action 子程序“Act_02_Interplation_Channel”中可以直观地看到功能块的输入变量和输出输出变量，



其中输入变量

- bGroupEnable: 通道组合和解散;
- bGroupReset : 通道复位;
- bGeoCmd : 装载插补指令;
- bStop : 插补运动停止;
- rOverride : 通道倍率, 单位 (%);

接口变量 5 个 PTP 轴 (Axis X 到 Axis Q2) 和 1 个 NCI 通道 (ItfChannel) 定义了 NCI 通道的组成。

接口变量 NCI_Entry 是自定义结构体, 用于暂存准备赋给待填充指令 GeoCircleCIP 和 GeoLine 的数据

输出变量:

- bGroupReady : 通道正常, 已装载好代码, 随时可以开始动作;
- bAxisReady : 通道内每个轴都使能成功, 没有报错;
- bGroupBusy : 通道正在执行动作;
- bGroupErr : 通道错误;
- nGroupErrID : 通道的错误代码;

- bExistX : X 轴已启用;
- bExistY : Y 轴已启用;
- bExistZ : Z 轴已启用;
- bExistQ1 : Q1 轴已启用;
- bExistQ2 : Q2 轴已启用;
- GrpState : NCI 通道的指令执行状态, 包括 Buffer 数目, 当前执行行号等;

eItpOpMode : NCI 通道的运行模式，最常见的是 Idle、Ready 和 IsRuning; 组合后装载 G 代码前为 Idle，成功装载 G 代码后启动前为 Ready，启动后结束前为 IsRuning。动作完成后又恢复为 Ready 状态。

从图上可以看出，FB 的接口做成了单个的变量，而调用 FB 并给它的接口赋值时，统一使用了 aNci_Itf[0]中的元素，并且调用时使用了梯形图。这是为了使程序监视更直观，可以在线显示每个接口变量的值，也可以随时强制某个变量。

在全局变量文件 Globale_Variabls 中定义了数组 aNci_Itf:

```
aNci_Itf      :   ARRAY[0..nMaxChannel] OF GCode_Chn_Interface ;
```

这是为 NCI 通道专门建的接口数组，一个 NCI 通道对应一数组里的一个结构。每个结构中包含了一个 NCI 通道的所有状态和控制信息。下一节会就此详展开细介绍。

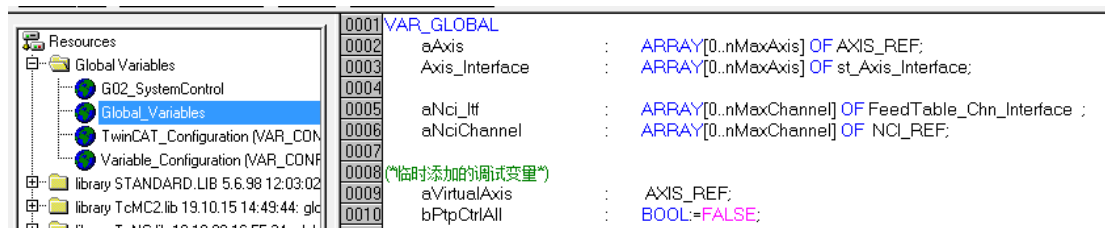
另外 aVirtualAxis 是为纯粹为了补齐 5 个轴，避免 FB 报错而建的辅助轴。实际使用是不要链接到任何 NC 轴即可。

4.4.2 控制对象和 Interface

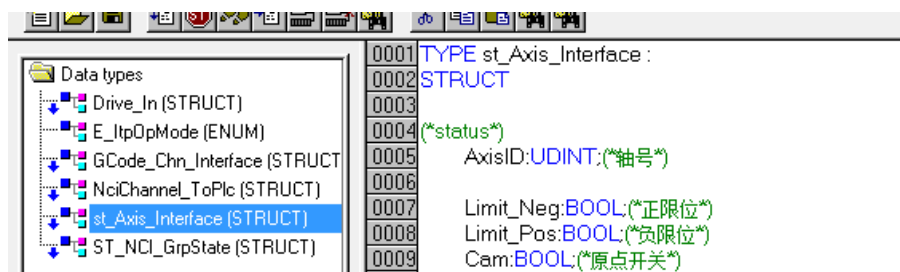
控制对象包括 NC 轴和 NCI 通道。本例程的基本思路是:

为每个 PTP 轴建一个 Interface 结构体 st_Axis_Interface，为每个 NCI 插补通道也建一个 Interface 结构体 GCode_Chn_Interface。

所以在全局变量中：aAxis 和 Axis_Interface 数组的下标数字相同，而 aNciChannel 和 aNci_Itf 数组的下标数字相同。如图所示:



其中 Axis 的接口定义，来自原先的 TcMc2.lib 例程:



由于结构体元素较多，不方便截图，以下为文字版:

TYPE st_Axis_Interface :

STRUCT

(*status*)

AxisID:UDINT>(*轴号*)

Limit_Neg:BOOL>(*正限位*)

Limit_Pos:BOOL>(*负限位*)

Cam:BOOL>(*原点开关*)

Drive_Alarm:BOOL>(*驱动器报警*)

Drive_Ready:BOOL>(*驱动器准备好*)

Drive_Torque:REAL>(*1000 对应 100% 额定转矩*)

DC_Voltage:REAL>(*驱动器直流母线电压*)

Current_Pos:LREAL>(*轴当前位置*)

Axis_Ready:BOOL>(*NC 轴准备好*)

Axis_Err:BOOL>(*NC 轴故障*)

Axis_ErrID:UDINT>(*NC 轴故障代码*)

Axis_IsCalibrated:BOOL>(*NC 轴寻参完成*)

Axis_IsNotMoving:BOOL>(*NC 轴静止*)

Axis_IsPTP :BOOL;

(*manual control*)

bEnable:BOOL>(*NC 轴使能*)

bEnableFw:BOOL>(*NC 轴允许正转*)

bEnableBw:BOOL>(*NC 轴允许反转*)

Jog_Pos:BOOL>(*NC 轴正向点动*)

Jog_Neg:BOOL>(*NC 轴反向点动*)

bStop:BOOL>(*NC 轴正常停止*)

bMoveAbs:BOOL>(*NC 轴绝对定位开始*)

bMoveRel:BOOL>(*NC 轴相对定位开始*)

bHome:BOOL>(*NC 轴寻参开始*)

bReset:BOOL>(*复位 NC 轴*)

bSetPos:BOOL>(*NC 轴重设位置*)

(*virtual Neg*)

bVirtualAxis:BOOL>(*NC 轴虚轴标记*)

(*Parameter*)

```

AbsMove_TargetPos:REAL:=10;(*NC 轴绝对定位位置*)
AbsMove_Velo:REAL:=10;(*NC 轴绝对定位的速度*)
RelMove_Distance:REAL:=10;(*NC 轴相对定位距离*)
RelMove_Velo:REAL:=10;(*NC 轴相对定位速度*)
Jog_Velo:REAL:=10;(*NC 轴点动速度*)
RefPos:REAL;(*NC 轴寻参完成时的原点位置*)
SetPosition:REAL;

```

```

END_STRUCT
END_TYPE

```

而 NCI 通道的接口定义如下:

Line	Code	Comment
0001	TYPE FeedTable_Chn_Interface :	
0002	STRUCT	
0003		
0004	bGroupEnable	: BOOL;
0005	bGroupReset	: BOOL;
0006	bStart	: BOOL;
0007	bStop	: BOOL;
0008	bSetOverride	: BOOL;
0009	rOverride	: REAL:=100;
0010		
0011	NCI_Entry	: NCI_SingleEntry;
0012		
0013	bAllAxisReady	: BOOL;
0014	bGroupReady	: BOOL;
0015	bGroupBusy	: BOOL;
0016	bGroupErr	: BOOL;
0017	nGroupErrID	: DWORD;
0018		
0019	bExistX	: BOOL;
0020	bExistY	: BOOL;
0021	bExistZ	: BOOL;
0022	bExistQ1	: BOOL;
0023	bExistQ2	: BOOL;
0024		
0025		
0026	nAxisID_X	: USINT:=0;
0027	nAxisID_Y	: USINT:=1;
0028	nAxisID_Z	: USINT:=2;
0029	nAxisID_Q1	: USINT;
0030	nAxisID_Q2	: USINT;
0031		
0032	stGrpState	: ST_NCI_GrpState;
0033	eltpOpMode	: E_ltpOpMode;
0034		
0035	END_STRUCT	
0036	END_TYPE	

以上只是例程的思路，用户可以根据实际情况再增减或者修改。

4.4.3 FB_MC_FeedTable 的源代码

POUs	0001	FUNCTION_BLOCK FB_MC_FeedTable
FB	0002	(*NC通道FeedTable控制封装FB_MC_FeedTable *)
FB_MC_FeedTable (FB)	0003	(*Version 1.0.0 By LizzyChen 2016.10.29*)
M_BasicFB	0004	VAR
M_GoPLine	0005	
M_UpdateStatus	0006	(*基本动作功能块: 通道组合, 通道解散, 装载G代码文件, 启动停止, 复位*)
NCI_REF (FB)	0007	CfgBuildExt3DGroup : CfgBuildExt3DGroup;
MAIN (PRG)	0008	fbClearGrp : CfgReconfigGroup;
Pro_IO (PRG)	0009	fbItpStartStop : ItpStartStop;
pro_MC_FeedTable (PRG)	0010	fbItpReset : ItpResetEx2;
Act_01_UpdateStatus	0011	
Act_02_Interplation_Channel	0012	TblPrep : FB_NciFeedTablePreparation;
Act_40_PrepereData	0013	GeoCircleCIP : ST_NciGeoCircleCIP; (*待填充到Table的圆弧插补指令*)
Act_41_Init_Data	0014	GeoLine : ST_NciGeoLine; (*待填充到Table的直线插补指令*)
Pro_MC_PTP (PRG)	0015	
	0016	NciFeedTable : FB_NciFeedTable;
	0017	Table : ST_NciFeedGroupTable; (*插补通道要执行的插补指令序列*)
	0018	

如果当前封装的功能不能满足需求，用户可以在此基础上修改。如果涉及到接口变量的增加，就需要同时修改结构体 FeedTable_Chnn_Interface，并在 FB_MC_FeedTable 增加这些变量的控制代码。

4.4.4 PTP 控制程序

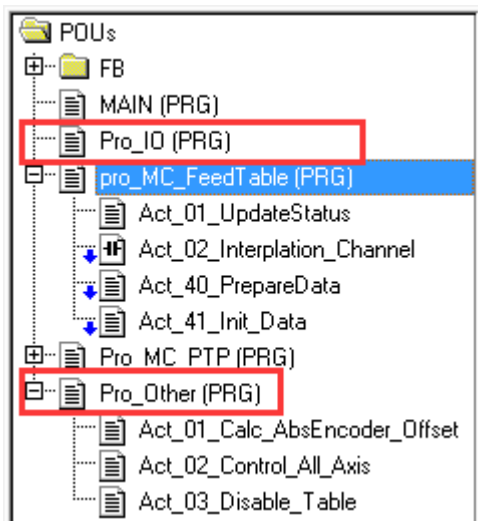
虽然这是 NCI 插补程序的例程，但是在实际项目中，必然涉及到 PTP 指令，比如使能、复位、寻参（回零）、点动等等。因此本例程直接集成了 TcMc2 PTP 的例程中的代码和 Axis Interface 结构体。

POUs	0001	PROGRAM Pro_MC_PTP
FB	0002	(*普通单轴指令及NC轴状态刷新*)
MAIN (PRG)	0003	VAR
Pro_IO (PRG)	0004	aMC_Power: ARRAY[0..nMaxAxis] OF MC_Power;
pro_MC_FeedTable (PRG)	0005	aMC_Jog: ARRAY[0..nMaxAxis] OF MC_Jog;
Pro_MC_PTP (PRG)	0006	aMC_Reset: ARRAY[0..nMaxAxis] OF MC_Reset;
Pro_Other (PRG)	0007	aMC_Stop: ARRAY[0..nMaxAxis] OF MC_Stop;
	0008	aMC_SetPos: ARRAY[0..nMaxAxis] OF MC_SetPosition;
	0009	aMC_MoveAbsolute: ARRAY[0..nMaxAxis] OF MC_MoveAbsolute;
	0010	aMC_MoveRelative: ARRAY[0..nMaxAxis] OF MC_MoveRelative;
	0011	aMC_Home: ARRAY[0..nMaxAxis] OF MC_Home;
	0012	i: INT;
	0013	END_VAR
	0014	
	0001	
	0002	FOR i:=0 TO nMaxAxis DO
	0003	
	0004	aAxis[i].ReadStatus;
	0005	
	0006	aMC_Power[i](
	0007	Enable:=Axis_Interface[i].bEnable ,
	0008	Enable_Positive:=TRUE ,
	0009	Enable_Negative:= TRUE,
	0010	Override:=rTargetOverride ,
	0011	Axis:=aAxis[i] ,
	0012	Status=> ,
	0013	Error=> ,
	0014	ErrorID=>);
	0015	

这部分代码用到的结构体 `aAxis` 和 `Axis_Interface` 已经在 4.4.2 控制对象和 `Interfacec` 中介绍过了，PTP 的用户对这些代码也已经相当熟悉，这里不再详述。

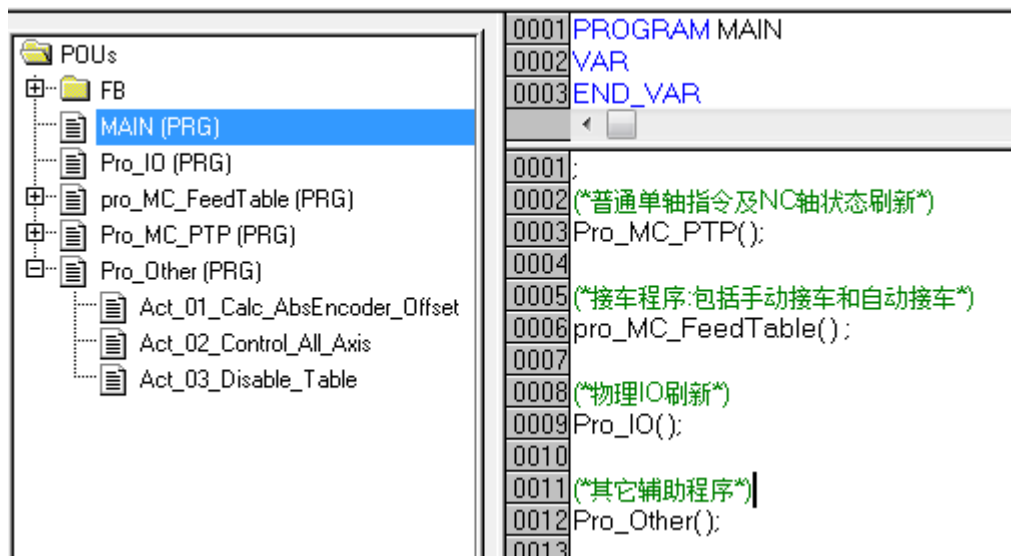
在本例中，还是通过程序 `Pro_MC_PTP` 中的 MC 功能块数组配合 `For` 语句来完成的。如果客户有兴趣，可以把一个 `Axis` 的功能也封装成一个 `FB`。

4.4.5 测试程序说明



上图中红线框住的，都是无关核心功能的辅助程序，目的是为了调试方便。

1) Main 程序入口



2) Pro_MC_FeedTable 也只提供子程序调用

```

0001 PROGRAM pro_MC_FeedTable
0002 VAR
0003
0004 (*Act 02 Interplation Channel, 插补运动*)
0005   afbNCI          : ARRAY[0..nMaxChannel] OF FB_MC_FeedTable ;
0006
0007
0008
0009
0010
0011
0012
0013
0014
0015
0016
0017
0018
0019
0020
0021
0022
0023
0024

```

主要代码在 Act_40_PrepareData;

Act_01_UpdateStatus 是为用户留的状态刷新接口，暂无代码

Act_02_Interplation_Channel 中只有 FB_MC_FeedTable 实例调用，在 4.4.1 中已详细分析。

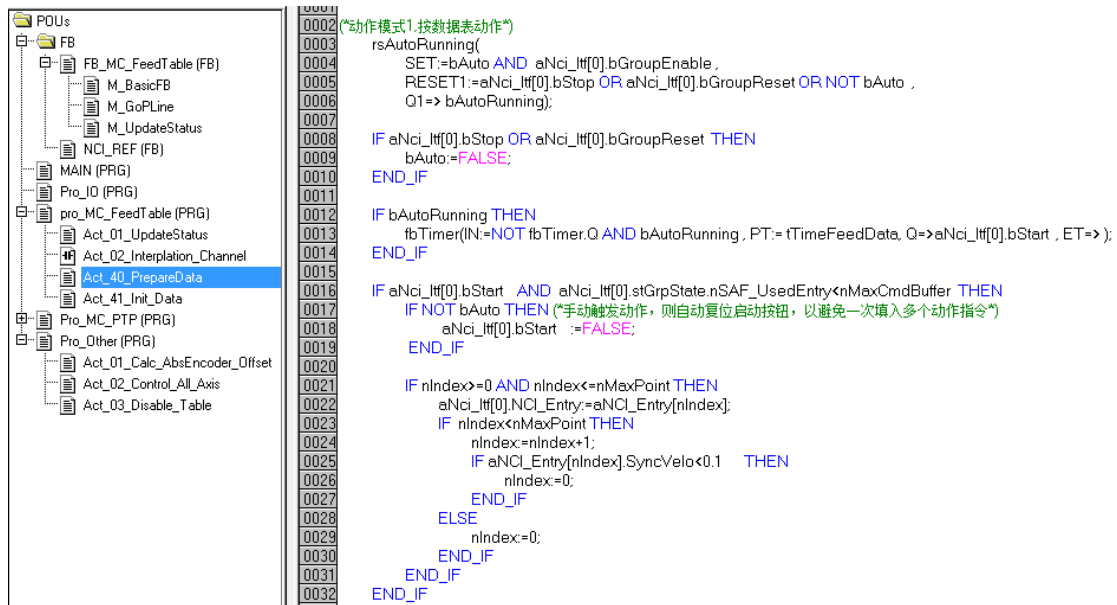
3) Act_41_Init_Data 中，是为了初始化 NCI 通道接口变量中的轴号和位置序列列表 Table

```

0001 IF bInited THEN RETURN; END_IF
0002   bInited:=TRUE;
0003
0004 (*初始化NCI通道的轴与aAxis数组各轴的对应关系*)
0005
0006   aNci_Itf[0].nAxisID_X      :=0;
0007   aNci_Itf[0].nAxisID_Y      :=1;
0008   aNci_Itf[0].nAxisID_Z      :=2;
0009   aNci_Itf[0].rOverride      :=100;
0010
0011 (*初始化位置序列列表Table*)
0012
0013   Index:=0;
0014   aNCI_Entry[Index].fX:=200;
0015   aNCI_Entry[Index].fY:=120;
0016   aNCI_Entry[Index].fZ:=20;
0017   aNCI_Entry[Index].SyncVelo:=300;
0018
0019   Index:=1;
0020   aNCI_Entry[Index].fX:=200;
0021   aNCI_Entry[Index].fY:=120;
0022   aNCI_Entry[Index].fZ:=160;
0023   aNCI_Entry[Index].SyncVelo:=300;
0024

```

4) Act_40_PrepareData 用于准备 NCI 通道接口 aNci_Itf[0] 中的插补指令 NCI_Entry 的数据。



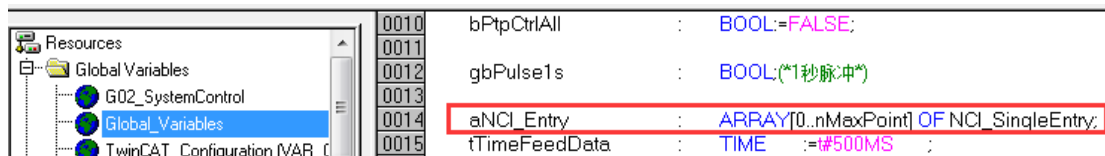
```

0002 (*动作模式1,按数据表动作*)
0003 rsAutoRunning(
0004   SET:=bAuto AND aNci_Itf[0].bGroupEnable,
0005   RESET1:=aNci_Itf[0].bStop OR aNci_Itf[0].bGroupReset OR NOT bAuto,
0006   Q1=> bAutoRunning);
0007
0008 IF aNci_Itf[0].bStop OR aNci_Itf[0].bGroupReset THEN
0009   bAuto:=FALSE;
0010 END_IF
0011
0012 IF bAutoRunning THEN
0013   fbTimer(IN:=NOT fbTimer.Q AND bAutoRunning, PT:=tTimeFeedData, Q=>aNci_Itf[0].bStart, ET=>);
0014 END_IF
0015
0016 IF aNci_Itf[0].bStart AND aNci_Itf[0].stGrpState.nSAF_UsedEntry<nMaxCmdBuffer THEN
0017   IF NOT bAuto THEN (*手动触发动作, 则自动复位启动按钮, 以避免一次填入多个动作指令*)
0018     aNci_Itf[0].bStart :=FALSE;
0019   END_IF
0020
0021   IF nIndex>=0 AND nIndex<=nMaxPoint THEN
0022     aNci_Itf[0].NCI_Entry:=aNci_Entry[nIndex];
0023     IF nIndex<nMaxPoint THEN
0024       nIndex:=nIndex+1;
0025       IF aNci_Entry[nIndex].SyncVelo<0.1 THEN
0026         nIndex:=0;
0027       END_IF
0028     ELSE
0029       nIndex:=0;
0030     END_IF
0031   END_IF
0032 END_IF

```

说明:

插补指令来源于全局变量，数组 aNCI_Entry:



0010	bPtpCtrlAll	: BOOL:=FALSE;
0011		
0012	gbPulse1s	: BOOL:(*1秒脉冲*)
0013		
0014	aNCI_Entry	: ARRAY[0..nMaxPoint] OF NCI_SingleEntry;
0015	tTimeFeedData	: TIME :=#500MS ;

子程序 Act_40_PrepareData 的作用，就是从这个数组中挑选适当的元素，赋给 NCI 通道接口 aNci_Itf[0]中的插补指令 NCI_Entry。

在 aNci_Itf[0].bStart 的上升沿，就会填充一条插补指令。Start 的触发逻辑如下：

如果 Auto 为 False，表示单次触发插补运动，需要手动控制变量 bStart；

如果 Auto 为 True，则表示自动循环触发，触发的时间间隔由变量 tTimeFeedData 指定，默认为 500ms。停止和复位信号会关闭自动循环触发，使用者需要重新在界面上点击按钮“自动连续运动”。

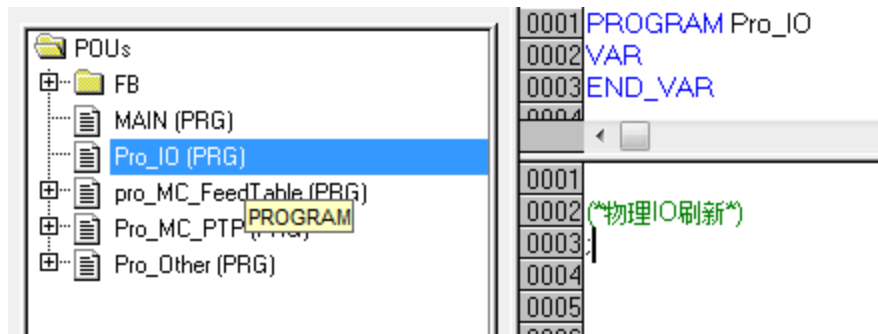
如果触发时间间隔大于动作实际执行时间，执行完一条就会停下来，等着下一次触发。如果触发时间间隔小于动作实际执行时间，指令就会缓存在 NCI 通道的 Buffer 中，时间长了 Buffer 可能会满，以至后面插入的指令丢失。为了解决这个 Buffer 满而指令丢失的问题，在填充指令时增加了条件“aNci_Itf[0].stGrpState.nSAF_UsedEntry<nMaxCmdBuffer”。

其中 aNci_Itf[0].stGrpState.nSAF_UsedEntry 是由 1 秒钟的脉冲（全局变量 gbPulse1s）触发 ADSRead，从 NCI 通道读取的当前缓存指令数。常数 nMaxCmdBuffer 用于限制最大的 Buffer 指令数量。NCI 规定 Buffer 条数不能大于 128。实际上该值设为 10 就能保证动作连贯了。限制了 Buffer 指令数量之后，触发 Start 变量的时间 tTimeFeedData 就不重要了，用默认值即可。

条件 aNCI_Entry[nIndex].SyncVelo<0.1，指通过进给速度 SynVelo 的值判断 aNCI_Entry 中的最后一条有效插补指令。如果判断为最后一条有效指令，就自动切回到第一条，开始第

二个循环。

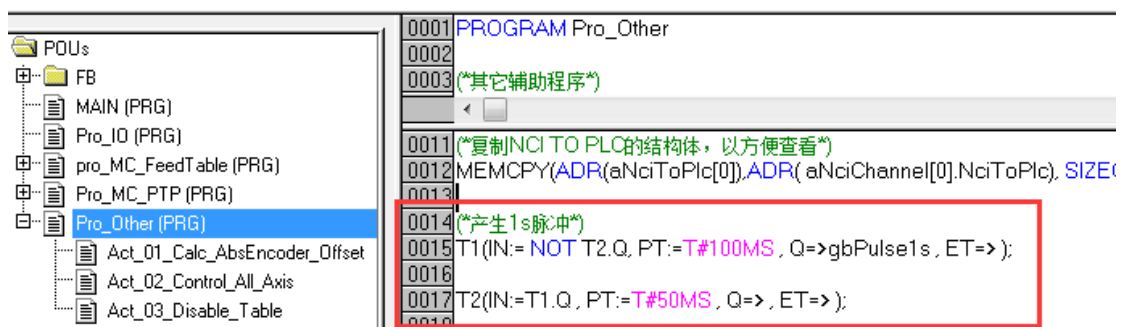
5) Pro_IO 中是与硬件 IO 相关的程序



在正式的应用项目中，可以把全部 IO 模块的变量赋值都放在这里。

在 Pro_Other 中集中放置辅助功能代码，本例中包括：

6) 产生 1s 脉冲

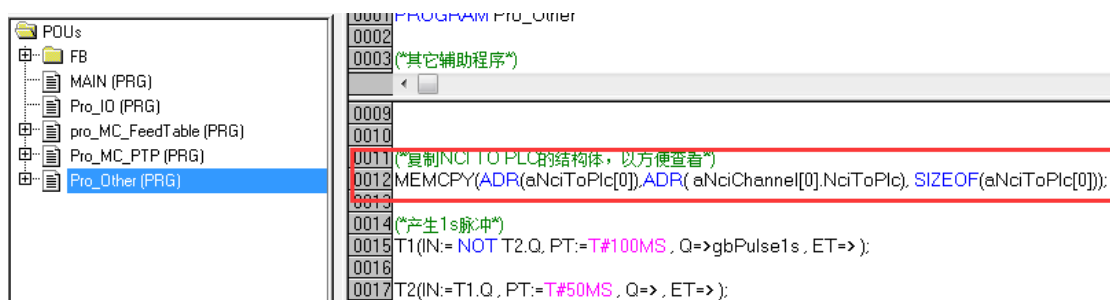


前面提到，这个信号在 FB_MC_FeedTable 中用于触发 ADSRead，读取 NCI 通道的 Buffer 指令数量。当然实际项目中，这个信号还可以用于闪灯等其它公共用途。

提示：在其它品牌的 PLC 中，通常有固定的 PLC 变量，提供 1s 或者 100ms 或者更长周期的脉冲。但在 TwinCAT 中，需要自己用代码实现。

7) NciToPlc 结构体的转换赋值

本部分与 3.5.5 中的相关说明一致，在这里重复一遍，是因为有的用户只用 FeedTable 功能，而不使用 G 代码，可能不会阅读第 3 章。



其中 aNciToPlc 的定义如下：

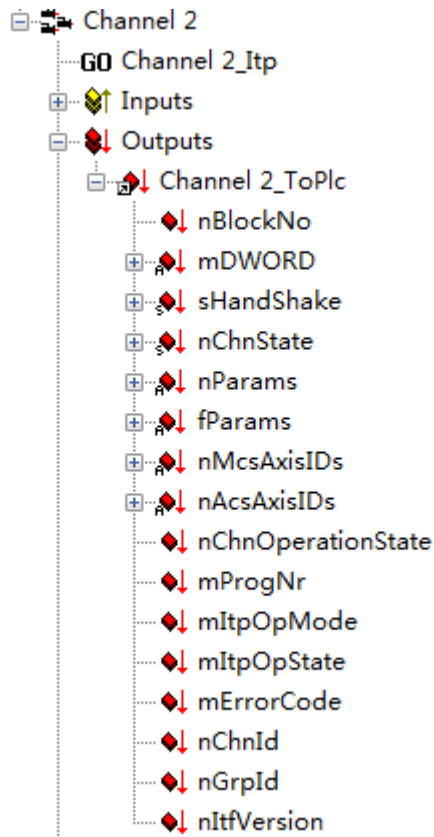
```
aNciToPlc : ARRAY[0..nMaxChannel] OF NciChannel_ToPlc;
```

这是因为 TcNci.lib 中的 NCI 接口变量 NciChannelToPlc 与 System Manager 中 NCI 通道的输出变量结构体的元素顺序不一致，

在 TcNci.lib 中的结构体定义如下：

```
TYPE NciChannelToPlc:
STRUCT
  nJobNo      : DWORD;
  nFastMFuncMask : ARRAY[1..5] OF DWORD; (* Mask to evaluate fast M-functions *)
  nHskMFuncNo  : WORD; (* evaluate M-function with handshake *)
  nHskMFuncReq : WORD;
  nHFuncValue  : DINT;
  nSpindleRpm  : WORD;
  nTool        : WORD;
  nReserved1   : ARRAY[37..132] OF BYTE; (* 37..40 nChnState *)
                (* 41..56 nParams *)
                (* 57..88 fParams *)
                (* 89..132 reserved *)
  nLoadedProg  : DWORD; (* loaded program number if exist *)
  nItpMode     : WORD; (* Interpreter mode *)
  nItpState    : WORD; (* Interpreter status *)
  nItpErrCode  : WORD; (* Interpreter-Channel Error Code *)
  nReserved2   : ARRAY[143..150] OF BYTE; (*143..144 still reserved *)
                (*145..146 nChnId *)
                (*147..148 nGrpId *)
                (*149..150 nItpVersion *)
END_STRUCT
END_TYPE
```

而在 System Manager 看到的 NCI 通道到 PLC 的结构体如下：



二者不一致, 导致在 System Manager 中看到的状态信息, 在 PLC 程序中无法一一对应, 所以根据 System Manager 中的变量顺序, 新建了结构体 NciChannel_ToPlc:

TYPE NciChannel_ToPlc :

(*2016.10.23 by LizzyChen*)

STRUCT

```
nJobNo          : UDINT;
nFastMFuncMask  : ARRAY[1..5] OF UDINT;
                 (* Mask to evaluate fast M-functions *)
```

(*sHandShake*)

```
nHskMFuncNo    : WORD;   (* evaluate M-function with handshake *)
nHskMFuncReq   : USINT;
nReserved_1    : BYTE;
nHFuncValue    : DINT;
nSpindleRpm    : WORD;
nTool          : WORD;
```

(*nChnState*)

```
nChnState      : DWORD;
```

```

nParams      : ARRAY[1..4] OF UDINT;
fParams      : ARRAY[1..4] OF LREAL;
nMcsAxisIDs  : ARRAY[1..8] OF USINT;
nAcsAxisIDs  : ARRAY[1..8] OF USINT;

nReserved_2  : ARRAY[1..24] OF BYTE;

nChnOpState  : UDINT;
nProgNr      : UDINT;
nItpOpMode   : WORD;      (* Interpreter mode *)
nItpOpState  : WORD;      (* Interpreter status *)
nItpErrCode  : DWORD;     (* Interpreter-Channel Error Code *)
nChnID       : UINT;
nGrpID       : UINT;
nItfVersion  : UINT;

END_STRUCT
END_TYPE

```

然后在 Pro_Other 中用内存拷贝指令 MEMCPY，将 IO 映射过来的类型为 NciChannelToPlc 结构体转送到与 TSM 中一致的类型为 NciChannel_ToPlc 的结构体中。

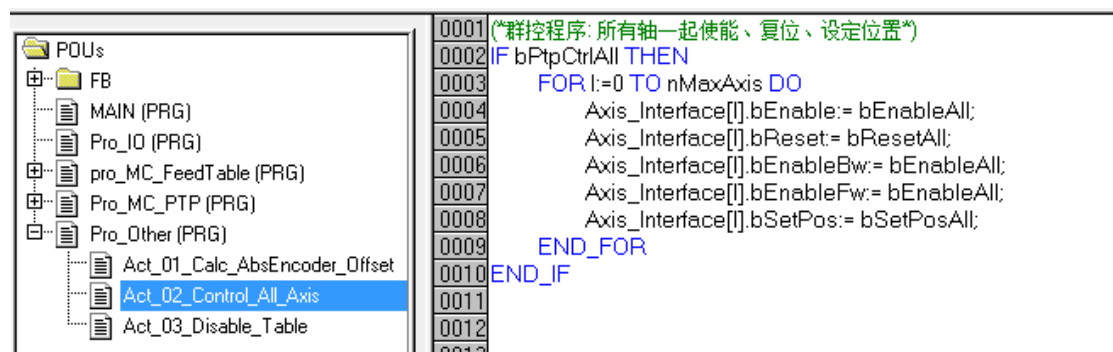
```

0009
0010 (*复制NCI TO PLC的结构体，以方便查看*)
0011 MEMCPY(ADR(aNciToPlc[0]),ADR( aNciChannel[0].NciToPlc), SIZEOF(aNciToPlc[0]));
0012

```

8) PTP 轴的群控

这是因为轴数特别多的时候，在调试界面上单个点击每个轴的接口变量，比如使能、复位、位置置零时，会相当耗时。所以在界面上做了几个群控的按钮，一旦群控模式（bPtpCtrlAll）开启，这个几个变量的值就直接赋到每个轴的 Interface 里了。

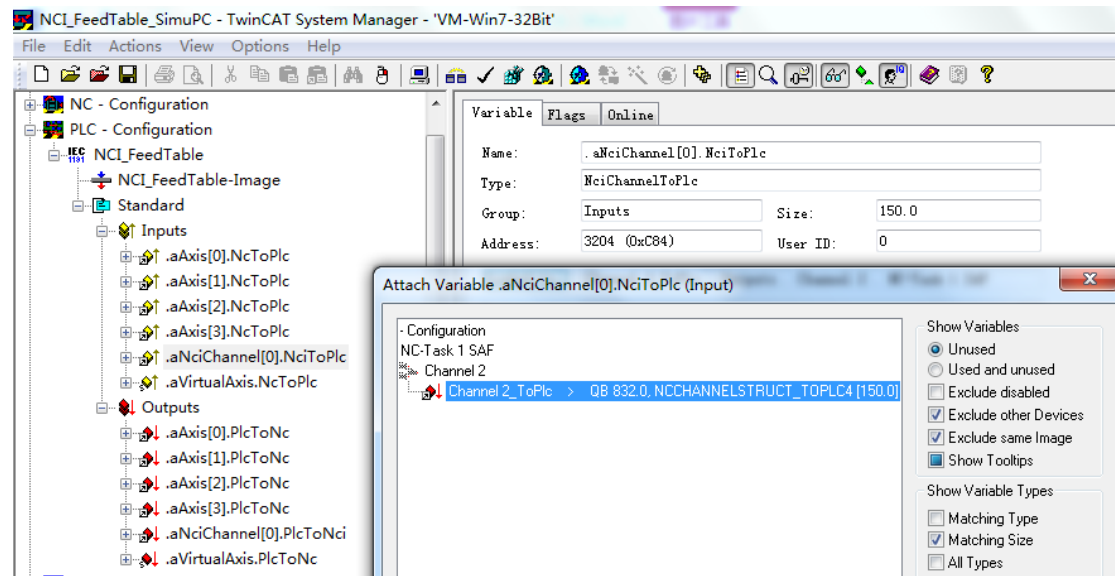


```

0001 (*群控程序: 所有轴一起使能、复位、设定位置*)
0002 IF bPtpCtrlAll THEN
0003   FOR I:=0 TO nMaxAxis DO
0004     Axis_Interface[].bEnable:= bEnableAll;
0005     Axis_Interface[].bReset= bResetAll;
0006     Axis_Interface[].bEnableBw:= bEnableAll;
0007     Axis_Interface[].bEnableFw:= bEnableAll;
0008     Axis_Interface[].bSetPos:= bSetPosAll;
0009   END_FOR
0010 END_IF
0011
0012
0013

```

4.4.6 System Manager 配置文件



注意:

.aVirtualAxis 是辅助轴，类似为了字对齐而在结构体中增加的 Reserve 或者 Dummy 变量，仅仅是为了程序不报错。不必连接 NC 轴。

4.4.7 调试画面

TwinCAT NC PTP 及FeedTable 插补通道控制界面 2016.10.23									示例： 循环位置序列				
群控PTP	ID	Ready	故障	使能	Jog +	Jog -	当前位置	点动速度		IX	IY	IZ	Velo
全部使能	1	1	0	1			200.0	10.0		200.0	120.0	20.0	300.0
全部复位	2	1	0	1			120.0	10.0		200.0	120.0	160.0	300.0
全部置0位	3	1	0	1			165.0	10.0		200.0	120.0	165.0	50.0
	4	1	0	1			0.0	10.0		200.0	120.0	290.0	300.0
插补使能	插补复位			通道运行状态		Idle							
插补启动	自动连续运动			允许最大缓存指令数		10条							
插补停止	间隔时间 T#500ms			未执行指令数		0条							
										100.0	50.0	290.0	300.0
										280.0	20.0	290.0	300.0
										200.0	120.0	290.0	300.0
										200.0	120.0	161.0	300.0
										200.0	120.0	158.0	50.0
										200.0	120.0	158.0	0.0
										0.0	0.0	0.0	0.0
										0.0	0.0	0.0	0.0
										0.0	0.0	0.0	0.0
										0.0	0.0	0.0	0.0

画面的上半部是 PTP 调试界面，包括群控的使能、复位、位置置 0 按钮。右边的列表中有轴的 Ready、Error、ErrorID 等状态，以及使能、正反向点动 Jog+、Jog- 等控制信号。显示当前位置，可单独设置点动速度。

下半部的 NCI 调试界面，除了 5 个命令按钮之外，还有 3 个文本框，说明如下：

通道运行状态： NCI 通道的运行模式 eItpOpMode，最常见的是 Idle、Ready 和 IsRuning；组合后装载 G 代码前为 Idle，成功装载 G 代码后启动前为 Ready，启动后结束前为 IsRuning。动作完成后又恢复为 Ready 状态。

允许最大缓存指令数： 默认为 10，可以设置，但最大不要超过 120。

未执行指令数： 指从 G 代码文件预读到 NCI 编译器但还没有执行的 G 代码行数。执行完毕，该值为 0。由于 NCI 缓存的容量（SAF Entry）有限，该值最大为 128。也就是说 NCI 最多缓存 128 条指令。虽然 SVB Entry 中也可以存 64 条，但是测试结果表明，把 SAF Entry 用尽偶尔会引起动作异常，原因有待研究。这个参数不能从 NciToPlc 的结构体中获得，必须用 ADS 通讯从 NCI 设备中读回来，这些代码在 FB_NCI_GCode 的 M_UpdateStatus 中：

<pre> 0037 fbADSRead(0038 NETID:=, 0039 PORT:=501, 0040 IDXGRP:= 16#3100+GroupID, 0041 IDXOFFS:= nIdxOffset, 0042 LEN:=4, 0043 DESTADDR:=, 0044 READ:=gbPulse1s, 0045 TMOUT:=T#500MS, 0046 BUSY=>, </pre>	<pre> GroupID = 16#00000005 nIdxOffset = 16#0000000D gbPulse1s = TRUE </pre>
---	---

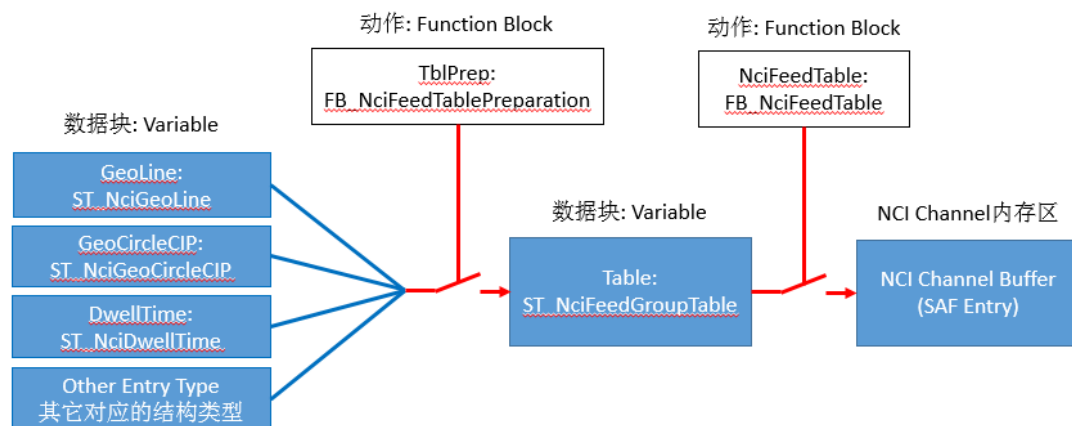
4.4.8 测试 FeedTable 控制 NCI 通道的操作顺序

- 1) 选择目标控制器，激活 NCI_FeedTable_SimuPC.tsm
- 2) 打开 NCI_FeedTable.pro，下载到目标 PLC，运行。
- 3) 进入 HMI
- 4) 使能 NC 轴。使用群控功能，可以节约时间。
- 5) 如有需要可以全部位置置 0
- 6) 插补使能——自动连续运动（或插补启动）
- 7) 插补停止，或者插补复位。

执行 Step 6 和 7 时，观察轴的位置变化和通道的状态、未执行指令数及当前行号的变化。

4.5 FeedTable 可以填充的非运动指令

前面的程序在向 Table 填充指令时，只用了运动指令。但是在实际应用中，还有一些辅助指令，比如延时、M 函数、改变加减速等等。实际上利用 FeedTable 功能可能填充的非运动指令也是很丰富的。



在 Beckhoff Infomaton System 帮助文件中，可以找到功能块 FB_NciFeedTablePreparation 的定义：

```
FUNCTION_BLOCK FB_NciFeedTablePreparation
VAR_INPUT
    nEntryType: E_NciEntryType;
    pEntry: POINTER TO ST_NciGeoLine;
    bResetTable: BOOL := FALSE;
    bResetAll: BOOL := FALSE;
END_VAR
```

```

VAR_IN_OUT
  stFeedGroupTable: ST_NciFeedGroupTable;
END_VAR
VAR_OUTPUT
  nFilledRows: INT := 0;
  bError:      BOOL  := FALSE;
  nErrorId:    UDINT := 0;
END_VAR

```

其中可以填充的运动与非运动指令类型就在 `nEntryType` 的枚举类型 `E_NciEntryType` 中定义,而调用功能块时 `nEntryType` 的枚举类型必须与指针 `pEntry` 所指向的变量类型相匹配。可以理解为 `nEntryType` 决定了类型,而 `pEntry` 所指向的结构变量中就包含了这个类型对应的参数。最典型的,如果 `nEntryType` 用了直线插补类型, `pEntry` 所指向的结构变量中就应该包含终点坐标和进给速度,如果 `nEntryType` 用了延时等待类型, `pEntry` 所指向的结构变量中就应该等待时间,以此类推。

下表列出 `nEntryType` 的枚举种类和各自对应的结构类型

Enum	Matching structure
E_NciEntryTypeNone	
E_NciEntryTypeGeoStart	ST_NciGeoStart
E_NciEntryTypeGeoLine	ST_NciGeoLine
E_NciEntryTypeGeoCirclePlane	ST_NciGeoCirclePlane
E_NciEntryTypeGeoBezier3	ST_NciGeoBezier3
E_NciEntryTypeGeoBezier5	ST_NciGeoBezier5
E_NciEntryTypeMFuncHsk	ST_NciMFuncHsk
E_NciEntryTypeMFuncFast	ST_NciMFuncFast
E_NciEntryTypeResetAllFast	ST_NciMFuncResetAllFast
E_NciEntryTypeHParam	ST_NciHParam
E_NciEntryTypeSParam	ST_NciSParam
E_NciEntryTypeTParam	ST_NciTParam
E_NciEntryTypeDynOvr	ST_NciDynOvr
E_NciEntryTypeVertexSmoothing	ST_NciVertexSmoothing
E_NciEntryTypeBaseFrame	ST_NciBaseFrame
E_NciEntryTypePathDynamics	ST_NciPathDynamics
E_NciEntryTypeAxisDynamics	ST_NciAxisDynamics
E_NciEntryTypeDwellTime	ST_NciDwellTime
E_NciEntryTypeTfDesc	ST_NciTangentialFollowingDesc

E_NciEntryTypeEndOfTables

[ST_NciEndOfTables](#)

在帮助文件中有每种结构类型的参数，在此不再详述。

```

TYPE E_NciEntryType :
(
  E_NciEntryTypeNone := 0,
  E_NciEntryTypeGeoStart := 1,
  E_NciEntryTypeGeoLine := 2,
  E_NciEntryTypeGeoCirclePlane := 3,
  E_NciEntryTypeGeoBezier3 := 10,
  E_NciEntryTypeGeoBezier5 := 11,
  E_NciEntryTypeMFuncHsk := 20,
  E_NciEntryTypeMFuncFast := 21,
  E_NciEntryTypeMFuncResetAllFast := 23,
  E_NciEntryTypeHParam := 24,
  E_NciEntryTypeSParam := 25,
  E_NciEntryTypeTParam := 26,
  E_NciEntryTypeDynOvr := 50,
  E_NciEntryTypeVertexSmoothing := 51,
  E_NciEntryTypeBaseFrame := 52,
  E_NciEntryTypePathDynamics := 53,
  E_NciEntryTypeAxisDynamics := 55,
  E_NciEntryTypeDwellTime := 56,
  E_NciEntryTypeTfDesc := 100,
  E_NciEntryTypeEndOfTables := 1000
);
END_TYPE

```

Enum	Matching structure
E_NciEntryTypeNone	
E_NciEntryTypeGeoStart	ST_NciGeoStart
E_NciEntryTypeGeoLine	ST_NciGeoLine

如果对比 G 代码的列表，会发现 G 代码中可以使用的非运动指令比以上所列要多得多，所以只有相对简单的插补运动才会用 FeedTable 方式。

举例一：延时

```
E_NciEntryTypeDwellTime := 56,
```

与之对应的 ST 是：

```
TYPE ST_NciDwellTime :
```

```
STRUCT
```

```
  nEntryType      : E_NciEntryType := E_NciEntryTypeDwellTime;
```

```
  nDisplayIndex : UDINT;
```

```
  fDwellTime     : LREAL;
```

```
END_STRUCT
```

```
END_TYPE
```

变量声明：

```
NciDwellTime : ST_NciDwellTime
```

```
TblPrep      : FB_NciFeedTablePreparation;
Table        : ST_NciFeedGroupTable;
TableDisplayIndex : UDINT := 1; (*插补指令在 Table 中的索引号*)
```

代码区:

```
TableDisplayIndex := TableDisplayIndex+1;
NciDwellTime.NDisplayIndex:= TableDisplayIndex;
NciDwellTime.FDwellTime :=20;(*Unit:s*)
TblPrep(
  nEntryType      := NciDwellTime.nEntryType,
  pEntry          := ADR(NciDwellTime),
  bResetTable     := FALSE,
  stFeedGroupTable := Table,
  nFilledRows     => ,
  bError          => ,
  nErrorId       => );
```

举例二：修改加減速度

```
E_NciEntryTypePathDynamics := 53,
```

与之对应的 ST 是:

```
TYPE ST_NciPathDynamics :
STRUCT
  nEntryType: E_NciEntryType := E_NciEntryTypePathDynamics;
  nDisplayIndex: UDINT;
  fAcc: LREAL;
  fDec: LREAL;
  fJerk: LREAL;
END_STRUCT
END_TYPE
```

变量声明:

```
NciPathDynamics : ST_NciPathDynamics;
TblPrep      : FB_NciFeedTablePreparation;
Table        : ST_NciFeedGroupTable;
TableDisplayIndex : UDINT := 1; (*插补指令在 Table 中的索引号*)
```

代码区:

```
TableDisplayIndex := TableDisplayIndex+1;
NciPathDynamics.NDisplayIndex:= TableDisplayIndex;
NciPathDynamics.fAcc :=10000; (*Unit:mm/s2*)
NciPathDynamics.fDec :=10000; (*Unit:mm/s2*)
```



```
NciPathDynamics.fJerk      :=200000;    (*Unit:mm/s3*)
TblPrep(
  nEntryType               := NciPathDynamics.nEntryType,
  pEntry                   := ADR(NciPathDynamics),
  bResetTable              := FALSE,
  stFeedGroupTable        := Table,
  nFilledRows              => ,
  bError                   => ,
  nErrorId                 => );
```

Note: 2016.10.30 By LizzyChen,

如果要使用非运动指令，4.4 中封装的 FB 需要升级，在 NCI_FeedTable_Interface 中的 NCI_Entry 变量不应该包括实际的参数，而应改为类型和指针，即 nEntryType 和 pEntry。

5 关于 M 函数

M 函数是 G 代码中的一种特殊指令，用于触发一个可供 PLC 周期访问的开关状态。NCI 中每个通道最多可以用 160 个 M 函数，除了 M2, M17, M30 之外，其余都可以自由使用。对绝大多数设备来说已经足够了。

M 函数	用途
0..159	自由使用的 M-functions (除了 2, 17, 30)
2	程序结束
17	子程序结束
30	程序结束，清除所有 fast M 函数

例如：

```
N10    G01 X0 Y0 F3600
N20    G01 X0 Y100
N30    M13
N40    G01 X100 Y0 M4
N50    G01 X0 Y0
N60    M30
```

执行这段 G 代码，在 N30 行触发 M13，而 N40 行触发 M4。PLC 检测到 M13 或者 M4 触发后，就可以执行相应的处理程序。

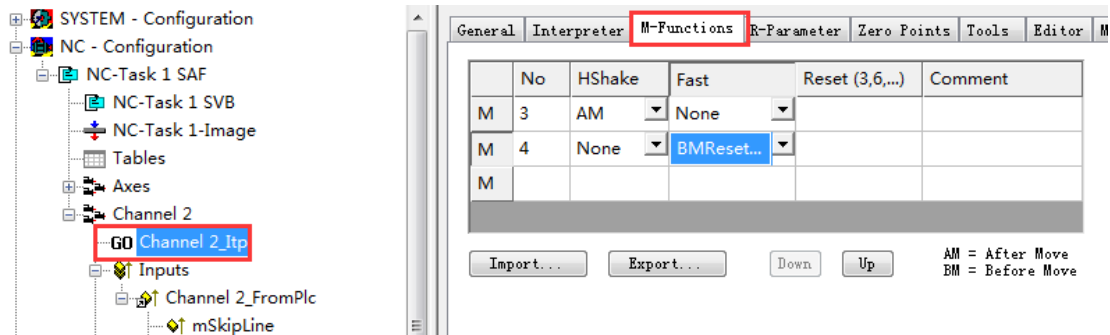
根据 M 函数是否打断 G 代码预读，可以分为握手型 M 函数 (MFuncShk) 和快速 M 函数 (MFuncFast)。顾名思义，握手型 M 函数需要 NCI 与 PLC 握手，M 函数在 NCI 通道中触发，在 PLC 中复位，然后才能执行后面的 G 代码。快速 M 函数不中断 G 代码预读，它触发以后，并不阻止后面的 G 代码运行。快速 M 函数可以不需要 PLC 复位，而在 NCI 通道中定义它的复位方式。

NCI 通道用到哪些 M 函数，分别是什么类型，需要在 NCI 配置文件中事先定义，未经定义则默认为握手型 M 函数 (MFuncShk)。如果是快速 M 函数 (MFuncFast)，那么它在什么时间恢复，也是可以设置的。

注意：在国际标准中，有一些 M 函数是有固定用途的，比如 M30 用于 G 代码结束，这些 M 函数在第 6 章 G 代码中描述。

5.1 M 函数的定义

在 TwinCAT NCI Channel 中，可以配置 M 函数的类型和参数，如图所示：

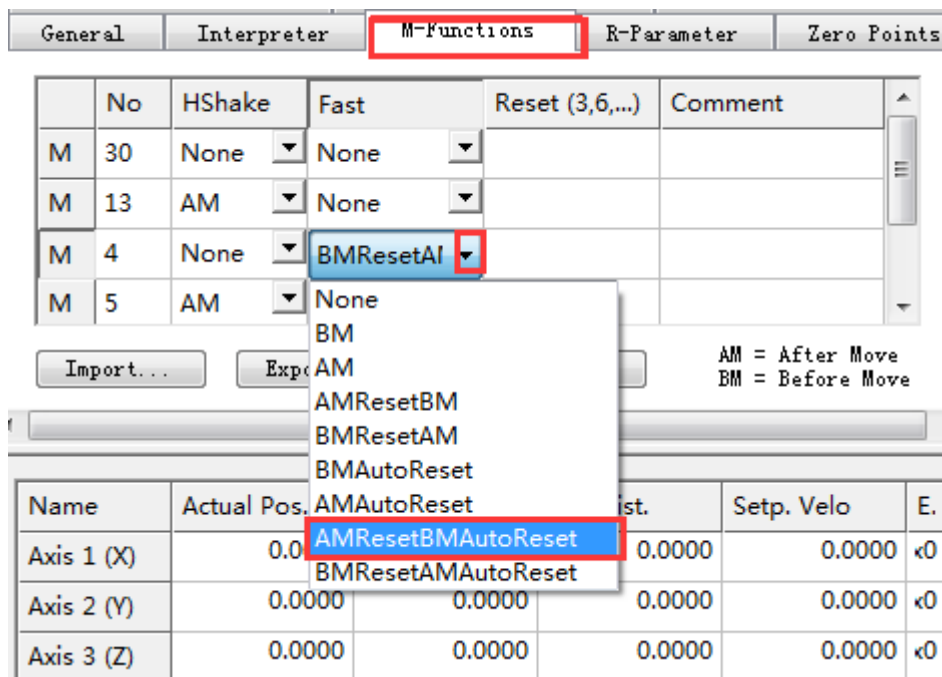


上图中：

No: M 函数的编号，最多 159 个。

HShake: 握手方式。当 M 函数与运动指令在同一行时，这个参数用于决定动作前还是动作后触发 M 函数。AM 表示 After Motion，BM 表示 Before Motion。None 表示只能独立成行？待测试。

Fast: 是否 Fast 函数。此处为 None，则表示握手型 M 函数。



在下拉菜单中的其它选项定义了 M 函数的触发类型和复位方式。

BM

AM

AMResetBM

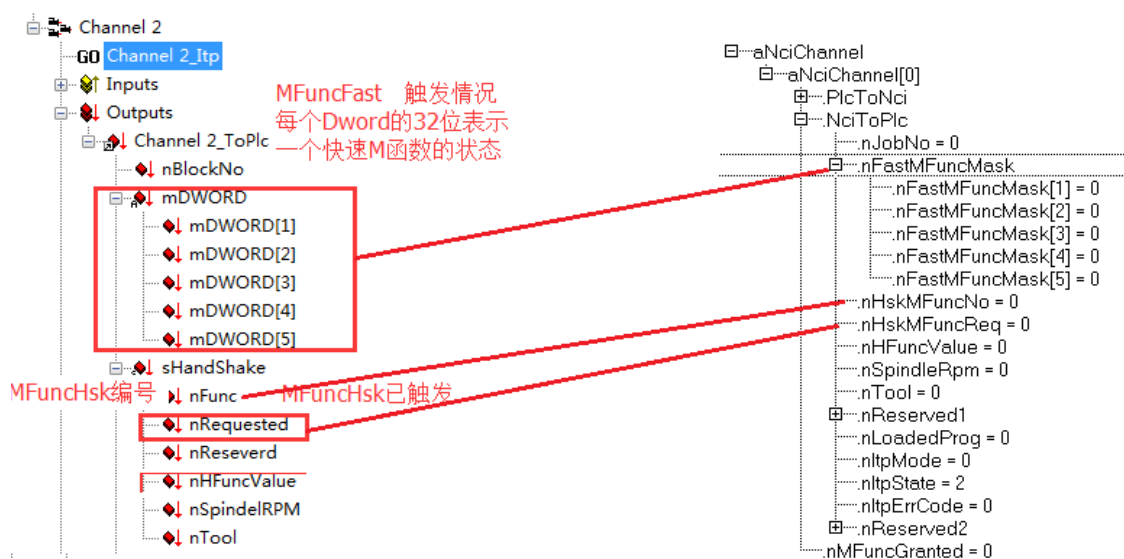
BMRResetAM
 BMAutoReset
 AMAutoReset
 AMResetBMAutoReset
 BMRResetAMAutoReset

通过 ScopeView 可以监视不同选项时 M 函数触发和复位的时间，以分辨它们的不同。如果包含了 AutoReset 字样，表示由前后的动作指令复位该函数。如否则需要在“Reset”这一列指定用另一个 M 函数来复位这一个 M 函数。

大多数情况下，如果只关心触发时间，而不关心复位的时间，就可以用 AutoReset 方式。如果直接 AM 或者 BM，就需要用 PLC 的功能块 ItpResetFastMFuncEx 来复位它了。与复位 MFuncShk 的方式一样。

5.2 显示和复位 M 函数的状态

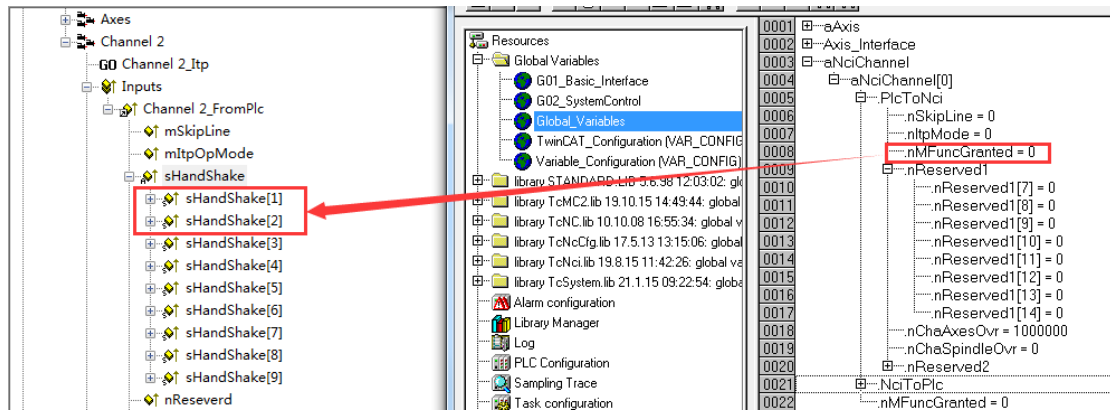
在 PLC 和 NCI 的接口结构变量中，可以查看 M 函数的状态：



快速 M 函数的状态用 5 个 DWord 来表示，每个 DWord 的 32Bit 表示 32 个 M 函数的状态，总共可以表达 160 个 M 函数。由 M 函数的最小编号是 M1，所以实际上第 1 个 DWord 的 Bit0 不对应任何 M 函数，所以 M 函数的最大值不超过 159。

握手型 M 函数同时最多只有一个触发，所以在一个结构体 sHandShake 中表示。nRequested 表示是否有触发，1 代表有，0 代表没有。而 nFunc 则表示触发的 M 函数的编号。

通过 PLC 和 NCI 的接口结构变量，可以复位 M 函数的状态：



nMFuncGranted 为 1 表示复位所有 M 函数，为 0 表示维持原状。其它数值无效。

5.3 用 PLC 函数获取 M 函数的状态及复位

根据上节内容，用户可以在 PLC 程序中直接访问与 NCI 的接口结构变量，同时为了使用方便，TcNci.lib 还提供 3 个 Function 来实现这些功能：

检查是否有 HandShake 类型的 M 函数触发：

```
FUNCTION ItpIsHskMFunc : BOOL
VAR_IN_OUT
    sNciToPlc:  NciChannelToPlc;
END_VAR
```

获取当前触发的 M 函数编号：

```
FUNCTION ItpGetHskMFunc : INT
VAR_INPUT
    sNciToPlc:  NciChannelToPlc;
END_VAR
```

检查指定的 MFast 函数有没有触发：

```
FUNCTION ItpIsFastMFunc : BOOL
VAR_INPUT
    nFastMFuncNo:  INT;
END_VAR
VAR_IN_OUT
    sNciToPlc:  NciChannelToPlc;
END_VAR
VAR CONSTANT
    nMFuncMin:  INT := 0;
    nMFuncMax:  INT := 159;
END_VAR
```

5.4 使用M函数的 NCI 项目举例

例程路径：“\配套例程\第 5 章 M 函数\NCI FB MFunc”

97 教材 ▶ 下册补充 NCI ▶ 配套例程 ▶ 第 5 章 M 函数 ▶ NCI FB MFunc

名称	修改日期	类型	大小
NCI_Test	2016/11/1 14:00	TSM 文件	247 KB
NCI_GCode_MFunc	2016/11/1 12:27	PRO 文件	291 KB

本例的目的是用简单的方式演示 M 函数的功能：M13 用于一个 BOOL 变量置位，M4 用于复位。

5.4.1 G 代码文件

97 教材 ▶ 下册补充 NCI ▶ 配套例程 ▶ 第 5 章 M 函数 ▶ GCode

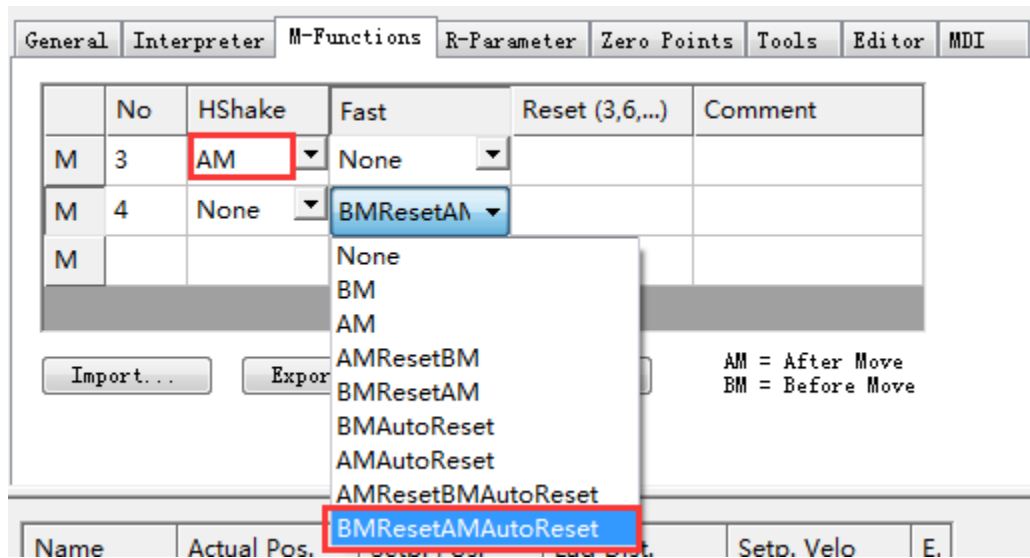
名称	修改日期
Mdemo	2016/10/23 15:49
Mdemo 2	2016/11/1 16:16

Mdemo 2 - 记事本

```
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
N10 G01 X0 Y0 F6000
N20 G01 X0 Y100 M3
N30 G01 X100 Y100 |
N40 G01 X100 Y0 M4
N50 G01 X0 Y0
N50 M30
```

G 代码文件 MDemo2.nc 应复制到控制器的“C:\TwinCAT\CNC”路径下。

5.4.2 M 函数的设置



为了对比 HandShake 与 Fast 类型的 M 函数的区别，

M3: HandShake, AM (After Move)

M4: Fast (BMResetAMAAutoReset)

5.4.3 PLC 代码

在第 3 章例程：“配套例程\第 3 章 使用 G 代的 NCI 项目\Demo Self Defined NCI FB\NCI_GCode.pro”的基础上，增加 POU“Pro_MFunc”，并在 Main 中调用。

变量声明：

VAR

State_M: WORD:=0;

nMFuncGranted: WORD;

tWait:TIME:=T#3s;

fbWait:TON;

I: INT;

bWait: BOOL;(*功能测试开关 : TRUE 表示 MFuncHsk 会等待一段时间复位*)

END_VAR

VAR_IN_OUT

ItpChannel : NCI_REF;

END_VAR

VAR_OUTPUT

```

    bLaser      :   BOOL   ;
END_VAR

```

代码实现:

```

CASE State_M OF

0: (*完成 M 函数的动作以后，复位确认信号，等待下一个 M 函数*)
    ItpChannel.nMFuncGranted:=1;
    ItpChannel.M_MFuncGrant;

    IF ItpIsHskMFunc(ItpChannel.NciToPlc) THEN (*检查是否有 M 函数发生*)
        State_M:= ItpGetHskMFunc(ItpChannel.NciToPlc);
    ELSE
        State_M:=0; (*检查 M1-M100 预定的 Fast MFunc 是否有发生*)
        FOR I:= 1 TO 100 DO
            IF ItpIsFastMFunc(I, ItpChannel.NciToPlc) THEN
                State_M:=I;
                EXIT;
            END_IF
        END_FOR
    END_IF;

3: (*Handle M3 MFunc Handshake*)
    bLaser := TRUE ;
    State_M := 89;

4: (*Handle M4, MFunc fast *)
    bLaser := FALSE ;
    State_M := 89;

89: (*Clear All MFunc *)
    IF bWait THEN
        act_Wait; (*延时 3 秒再复位 M 函数*)
    ELSE
        ItpChannel.nMFuncGranted:=1;
        ItpChannel.M_MFuncGrant;
        State_M:=99;
    END_IF

99: (*Wait one PLC cycle*)
    State_M:=0;

```



```

ELSE (*其它 M 函数触发了, 也直接复位*)
    State_M := 89;

END_CASE;

```

说明：最后两行，专门建个中间变量 `nMFuncGranted` 来赋给 `ItpChannel`，是因为 `ItpChannel` 是一个 `Function Block`，最初我们只是用它来把 `NCI` 和 `PLC` 的接口结构变量合二为一，方便链接。但在处理 `M` 函数的复位时，发现 `FB` 的局部变量不能从外部赋值。

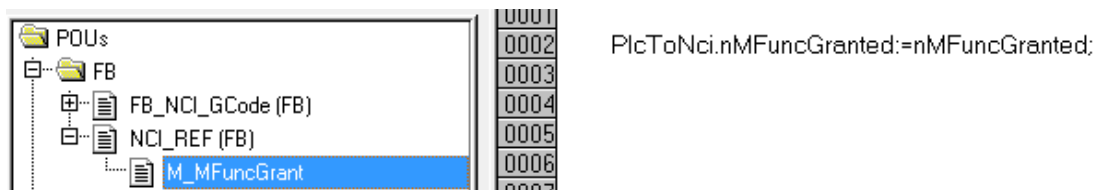
因为“`ItpChannel.PlcToNci.nMFuncGranted:=1;`”编译报错，所以替换为：

```

ItpChannel.nMFuncGranted:=0;
ItpChannel.M_MFuncGrant;

```

相应的 `ItpChannel` 的模型功能块 `NCI_REF` 中增加了
 输入变量 `nMFuncGranted`
 Action 子程序 `M_MFuncGrant`”



这个 Action 的唯一作用就是：把 `nMFuncGranted` 输出到 `PLC` 到 `NCI` 的接口结构。

5.4.4 测试画面

TwinCAT NC PTP 及 NCI 插补通道控制界面
2016.10.23

群控PTP	ID	Ready	故障	使能	Jog +	Jog -	当前位置	点动速度
	1		0				71.5	10.0
全部使能	2		0				100.0	10.0
全部复位	3		0				0.0	10.0
全部置0位	4		0				0.0	10.0

G代码文件 C:\TwinCAT\CNC\MDemo2.nc M函数 Laser

插补使能	插补复位	通道运行状态	IsRunning
插补启动	装载G代码	未执行指令数	11条
插补停止	M函数延时复位	当前指令行号	N 30

为了让动作停顿更明显，画面上增加了“M 函数延时复位”的开关。“M 函数 Laser”框就是显示用 M3 和 M4 控制的 BOOL 变量状态。

如果修改 M3 为 Fast 类型，则这个开关无论 On 还是 Off，动作都不会停顿了。

5.4.5 测试 NCI 插补功能的操作顺序

- 1) 选择目标控制器，激活 NCI_Test.tsm
- 2) 打开 NCI_GCode_MFunc.pro，下载到目标 PLC，运行。
- 3) 进入 HMI，确认 G 代码文件 MDemo2.nc 与控制器上的 CNC 文件路径一致。
- 4) 使能 NC 轴。使用群控功能，可以节约时间。
- 5) 如有需要可以全部位置置 0
- 6) 插补使能——装载 G 代码——插补启动
- 7) 插补停止，或者插补复位。
- 8) 可以修改 G 代码文件，或者选择其它 G 代码文件。重新执行。
- 9) 切换“M 函数延时复位”按钮状态，再“插补启动”，对比 M 函数触发时的效果。

执行 Step 6 和 7 时，观察轴的位置和通道的状态、未执行指令数及当前行号的变化。

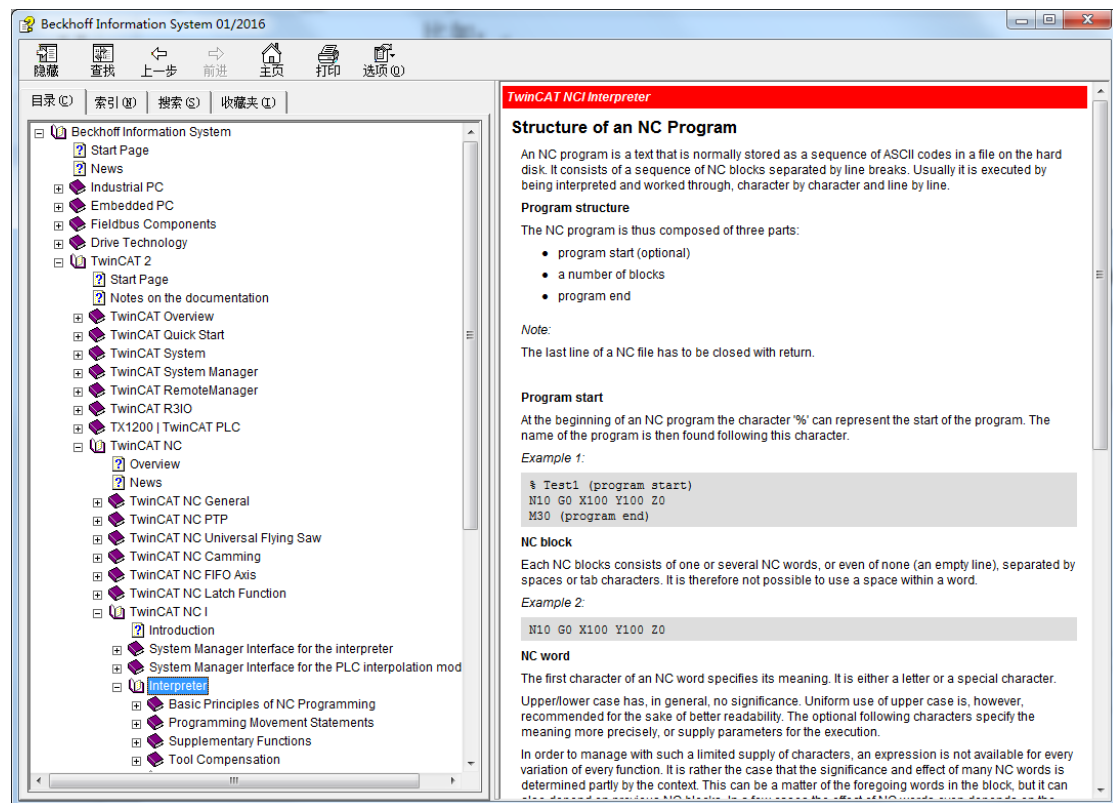
注意：停止和复位后必须重新装载 G 代码，更换 G 代码文件也要重新装载。

6 G 代码文件中的指令简介

TwinCAT NCI 的 G 代码系统符合 DIN66025 (Siemens Dialect) 规则, TwinCAT CNC 也是遵循 DIN66025, 所以本章只介绍 NCI 的 G 代码与 CNC 不同的部份。共同部分请参考附件: \配套例程\第 6 章 G 代码规则\

TwinCAT CNC 简明调试教程 第三章 CNC 编程指令.pdf

本章对这些指令简要介绍, 详细而完整的指令说明, 请参考 Beckhoff Information System 中的相关章节:



一个实际项目上用的 G 代码文件包括多种指令, 其中最核心的当然是以“G”字母打头的控制动作的指令, 比如常用的: G01 (直线插补), G02/G03 (圆弧插补)。执行动作的 G 代码同行通常会指定进给速度, 即 F 指令。比如:

```
G01 X100 Y100 F6000
```

就表示以 6000 mm/min 的速度, 即 100mm/s 的合成速度, 走直线去到 X100 Y100 的点。

除了执行动作之外, 还有配合插补动作:

- 逻辑动作 (M 指令)
- 主轴速度 (S 指令)
- Help 变量 (H 指令)
- 刀具选择 (T 指令)

PLC 程序每个周期都会通过 NciChannel_ToPLC 结构体刷新这些指令的值，用户需要编程实现特定的功能。其中 M 指令影响一个 BOOL 量，而 S、H、T 分别影响一个整型变量。

G 代码中给定动作参数时可以使用固定的值，比如 X100 Y100，也可以使用 LReal 型变量。但是 NCI 中不允许自定义变量名，而只能使用系统提供的 R 参数，从 R0-R999。

比如：

```
R0=100
G01 X=R Y=2*R
```

除了这些基本动作相关的指令，还有

控制 G 代码执行顺序 : @+数字，比如跳转、循环、子程序等；

设置运动特性的参数 : #Set Parameter(Value)#， Parameter 是可选参数之一。

Command 指令 : DIN66025 规则的命令字，比如 ROT 表示坐标旋转。

以上是我个人对 G 代码的一些认识，非官方非百度非参考任何资料。下面简单介绍 M 指令之外的常用功能及注意事项。M 指令的用法见第 5 章。

6.1 G 指令

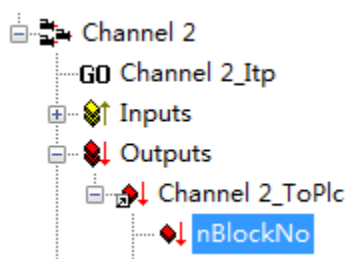
一段典型的 G 指令：

```
N10 G00 X0 Y0 (回零点)
N20 G01 X0 Y100 Q1=0 F3600 (以 60mm/s 的进给速度去座标 100,100)
N30 G01 X100 Y100 Q1=100
N40 G01 X100 Y0
N50 G01 X0 Y0 Q1=0
N60 M30
```

这段 G 代码包含了以下几种元素：

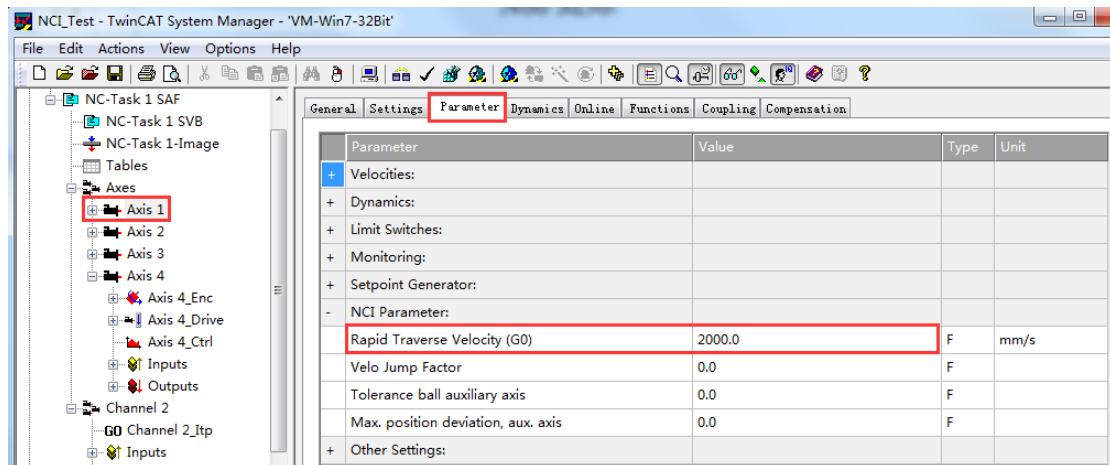
- 行号：N+数字。

可以有也可以没有。如果有行号，在 nBlockNo 中就可以发送当前正在执行的行号给 PLC。



G00: 快速回退

快速回退时没有插补，每个轴按自己的最大 G0 速度和加减速度运行到指定位置。



- **G01:** 直线插补
即从上次运动结束的位置走直线运动到 XYZ 坐标指定的目标位置。
- **G02/G03:** 圆弧插补
G02 是顺时针，G03 是逆时针。即从上次运动结束的位置沿顺时针或者逆时针走一段圆弧。圆弧特征参数是圆心、半径和终点坐标。
- **终点坐标: X100 Y100**
X\Y\Z 轴的坐标值可以紧跟轴号，比如 X100，也可以写 X=100
辅助轴 Q1 的坐标必须用“=”来赋值，比如 Q1=100。
- **进给速度: F3600**
指插补轴的合成速度，单位 mm/min，所以 F3600 即 60mm/s
进给速度给过一次之后对后续的动作也有效，直到下一次 F 指令出现才改变。
实际的轨迹速度并没有包括在 NciChannel to Plc 的结构体中，需要用 PLC 处理（**没有找到现成的 FB 或者 FC，只能用 X\Y\Z 三个方向的速度来合成**）
- **注释: (以 60mm/s 的进给速度去坐标 100,100)**
以英文括号括起来的部分是注释，支持中文。

常用的 G 代码还包括:

G00	Rapid Traverse	快速回退	m
G01	Linear Interpolation	直线插补	m
G02	Clockwise Circular Interpolation	顺时针圆弧插补	m
G03	Anticlockwise Circular Interpolation	逆时针圆弧插补	m
G04	Dwell Time	等待	b
G09	Accurate Stop	精确停止	b

G53	Zero Offset Shift Suppression	停用零点偏置	m
G54	1. Adjustable Zero Offset Shift	启用零点偏置	m
G58	1. Programmable Zero Offset Shift	启用零点偏置	m
G60	Accurate Stop	精确停止	m
G74	Referencing	寻找参考点	b
G90	Absolute Dimensions	启用绝对坐标	m
G91	Incremental Dimensions	启用相对坐标	m

完整的 G 代码请参考 [Beckhoff Information System](#).

6.2 SHT 指令

这 3 个指令都只影响 Channel_ToPlc 中的 1 个变量
3600mm/min 就对应速度 60mm/s

6.2.1 S 指令

由于 NCI 中并没有主轴的概念，所以 S 指令并不能让通道内某个轴的转速。但是可以影响 Channel_ToPlc 中的变量 nSpindleRPM，PLC 可以用这个值来调节一个 PTP 轴的转速。这个变量的功能完全取决于 PLC 逻辑。

The screenshot displays the TwinCAT NC I software interface. On the left, a tree view shows the project structure, including 'NC-Task 1 SVB', 'NC-Task 1-Image', 'Tables', 'Axes', and 'Channel 2'. Under 'Channel 2', the 'Outputs' folder is expanded, showing variables like 'nBlockNo', 'mDWord', 'sHandShake', 'nFunc', 'nRequested', 'nReseverd', 'nHFuncValue', 'nSpindelRPM', 'nTool', 'nChnState', 'bIsInterpolationChanne', 'bIsKinematicChannel', 'bIsEStopRequested', 'bIsFeedFromBackupLis', 'bIsMovingBackward', 'nParams', and 'fParams'. The 'nSpindelRPM' variable is highlighted with a red box.

The MDI editor on the right shows the target file 'C:\TwinCAT\NC\demo2.nc' and the following program code:

```
N10 G01 X0 Y0 F3600
N20 G01 X0 Y100 M3
N30 G01 X100 Y100 S2000
N40 G01 X100 Y0 M4
N50 G01 X0 Y0
N60 M30
```

The 'Actual Program Line' section shows the current line being executed:

```
N20 G01 X0 Y100 M3
N30 G01 X100 Y100 S2000
```

The 'Program Name' is 'Mdemo2.nc', the 'Interpreter State' is 'RUNNING (5)', and the 'Buffer Size (Byte)' is '65536'. The 'Channel State' is '0 (0x0)'.

Below the MDI editor, a table shows the current state of the 'nSpindelRPM' variable:

Name	Actual Pos.	Setp. Pos.	Lag Dist.	Setp. Velo	E.
Axis 1 (X)	48.9849	48.9849	0.0000	59.9940	<0
Axis 2 (Y)	100.0000	100.0000	0.0000	0.0000	<0
Axis 3 (Z)	0.0000	0.0000	0.0000	0.0000	<0

At the bottom, a table shows the variable 'nSpindelRPM' with its online value '0x07D0 (2000)' and source 'nSpindelRPM . sHandShake . Channel 2_ToPlc . Outputs . Channel 2 . NC-...'.

当执行到 S 指令的时候，nSpindelRPM 的值改变了。

6.2.2 H 指令

NCI 中的 H 指令，也只能影响 Channel_ToPlc 中的 1 个变量 nHFuncValue，这个变量的功能完全取决于 PLC 逻辑。

The screenshot displays the TwinCAT NC I software interface. On the left is a project tree showing the hierarchy: NC-Task 1 SVB, NC-Task 1-Image, Tables, Axes, Channel 2, GO Channel 2_Itp, Inputs, Channel 2_FromPlc, and Outputs. The main window shows the 'M-Functions' tab with a program editor. The target is set to 'C:\TwinCAT\NCI\Mdemo2.nc'. The program code is as follows:

```

N10 G01 X0 Y0 F3600 H100
N20 G01 X0 Y100 M3 H200
N30 G01 X100 Y100 S2000 H300
N40 G01 X100 Y0 M4 T3 H400
N50 G01 X0 Y0 H500
N60 M30

```

Below the editor is a table showing the current status of the axes:

Name	Actual Pos.	Setp. Pos.	Lag Dist.	Setp. Velo	E.
Axis 1 (X)	67.9430	67.9430	0.0000	59.9940	<0
Axis 2 (Y)	100.0000	100.0000	0.0000	0.0000	<0
Axis 3 (Z)	0.0000	0.0000	0.0000	0.0000	<0

The 'Actual Programm Line' section shows the current line being executed:

```

N20 G01 X0 Y100 M3 H200
N30 G01 X100 Y100 S2000 H300

```

Below the editor is a table showing the current status of the variables:

Variable	Online	Source
nSpindelRPM	0x07D0 (2000)	nSpindelRPM . sHandShake . Channel 2_ToPlc . Outputs . Channel 2 . NC-...
nTool	0x0000 (0)	nTool . sHandShake . Channel 2_ToPlc . Outputs . Channel 2 . NC-Task 1 S...
nHFuncValue	0x000000C8 (200)	nHFuncValue . sHandShake . Channel 2_ToPlc . Outputs . Channel 2 . NC-T...

说明：与 M 函数不一样，H 函数与动作指令写在同一行时，总是按它所在的位置来决定何时改变 nHFuncValue 的值。比如上图中，正在执行 N30，但是必须动作完成了，nHFuncValue 才会变成 300，把动作期间，它的值还是维持之前设置的值 200。

6.2.3 T 指令和 D 指令

由于 NCI 中的 T 指令也不能立即调用换刀程序，但是可以影响 Channel_ToPlc 中的变量 nTools，PLC 可以用这个值来调用换刀程序。这个变量的功能完全取决于 PLC 逻辑。

The screenshot shows the TwinCAT NC I interface. On the left, the project tree shows 'NC-Task 1 SVB' with 'Channel 2' selected. The 'Outputs' folder is expanded, showing 'Channel 2_ToPlc' with 'nTool' highlighted. The MDI editor shows the following G-code program:

```

Target: C:\TwinCAT\NC\Mdemo2.nc
N10 G01 X0 Y0 F3600
N20 G01 X0 Y100 M3
N30 G01 X100 Y100 S2000
N40 G01 X100 Y0 M4 T3
N50 G01 X0 Y0
N60 M30
  
```

The 'Actual Program Line' section shows the current line: 'N40 G01 X100 Y0 M4 T3'. The 'Program Name' is 'Mdemo2.nc', 'Interpreter State' is 'RUNNING (5)', and 'Buffer Size (Byte)' is '65536'. The 'Channel State' is '0 [0x0]'. The variable list at the bottom shows 'nTool' with a value of '0x0003 (3)'.

当执行到 T 指令的时候，nTool 的值改变了。

下图可以设置每把刀的补偿信息，启用刀补功能之后，NCI 执行动作时就会把刀具半径考虑进去，在 G 代码设定的轮廓线基础上叠加一个合适的偏置。

The screenshot shows the 'Tools' tab in the TwinCAT NC I software. The table below lists the tool compensation parameters for tools D 1 through D 5.

	TNr.(P0)	Typ(P1)	Geom.(P2)
D 1	0	0	0.000000
D 2	0	0	0.000000
D 3	0	0	0.000000
D 4	0	0	0.000000
D 5	0	0	0.000000

6.3 R 参数

G 代码中给定动作参数时可以使用固定的值，比如 X100 Y100，也可以使用 LReal 型变量。但是 NCI 中不允许自定义变量名，而只能使用系统提供的 R 参数。

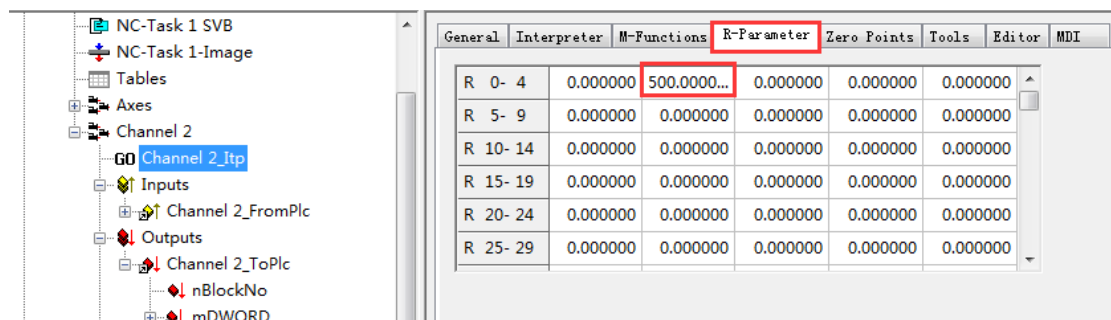
比如：

```
R0=100
G01 X=R Y=2*R
```

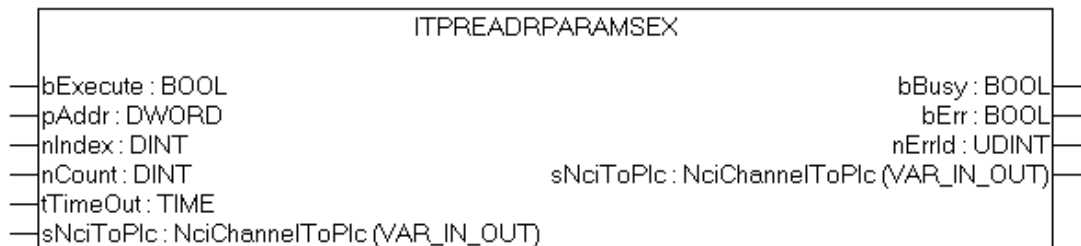
每个 NCI 通道有 1000 个 R 参数，从 R0-R999，都是 8 字节的 LReal 类型。它们与 PLC 并不能每个周期映射的。PLC 可以调用 Function Block 来读写 R 参数，而 G 代码文件中也可以设置或者使用 R 参数，这样就实现了 PLC 与 NCI 通道之间的实数类型的数据交换。

R 参数可以在 System Manager 的调试界面访问，也可以从 PLC 程序访问，也可以在 G 代码中访问：

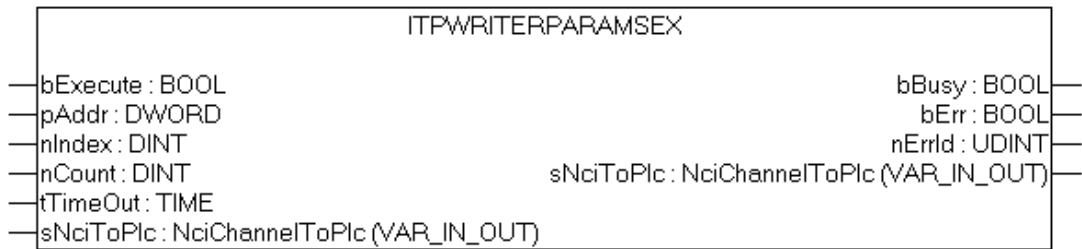
- 在 System Manager 的调试界面上修改



- 在 PLC 程序中访问
引用 TcNci.lib，就可以使用以下功能块：
读取 R 参数：ItpReadRParamsEx



写 R 参数: ItpWriteRParamsEx



- 在 G 代码中访问
给 R 参数赋值:

N50 R0=X

使用 R 参数:

N60 G01 X=R0+R1

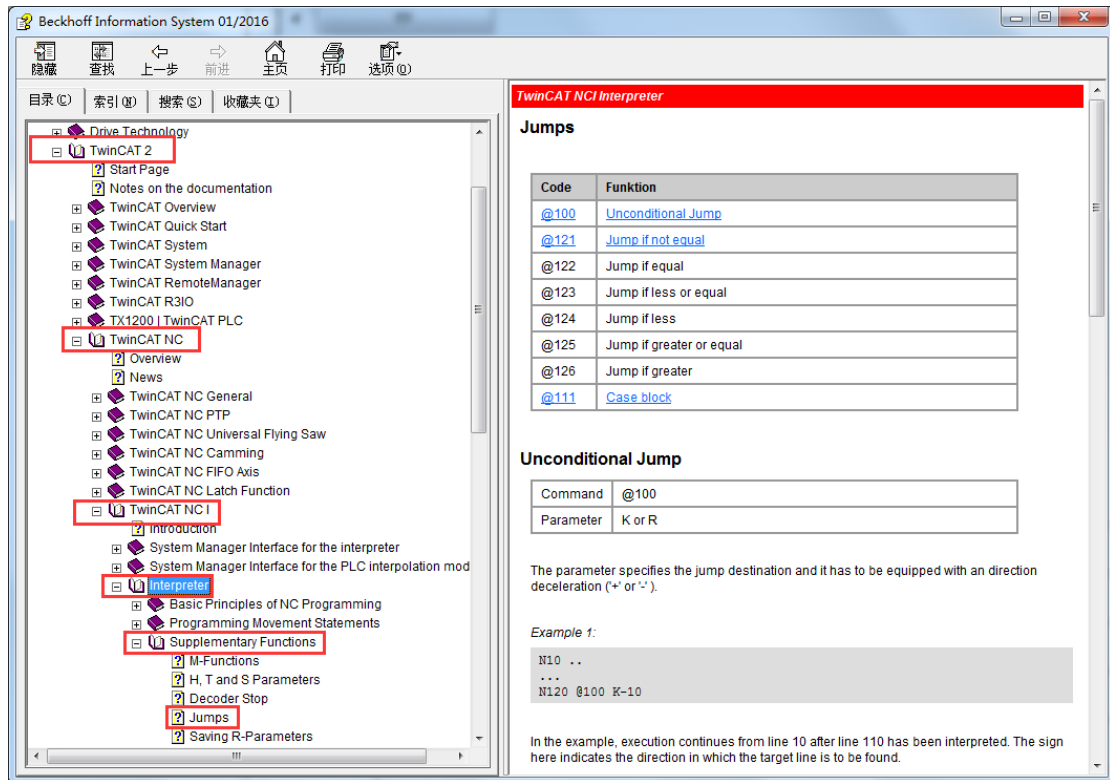
6.4 @指令

所有@指令以“@数字”开头，在 Beckhoff Information System 中有详细描述。

Command	Variation
@40	@40 Kn Rn Rm ...
@41	@41 Rn Rm
@42	@42 Kn ... Rm Rn
@43	@43 Rm Rn
@100	@100 K+n @100 Rm
@111	@111 Rn K/Rn Km ...
@121	@121 Rn K/Rn Kn
@122	@122 Rn K/Rn Kn
@123	@123 Rn K/Rn Kn
@124	@124 Rn K/Rn Kn
@125	@125 Rn K/Rn Kn
@126	@126 Rn K/Rn Kn

这里只挑常用的简述。

6.4.1 跳转指令



指令 功能

@100 无条件跳转

@100 K±n

@100 Rm

@121 条件跳转，如果不等于，测跳至

@122 Rn K/Rn Kn

- 无条件跳转

命令 @100

参数 K or R

例如：

N10 ..

...

N120 @100 K-10

表示在 N120 这一行，往回跳至 N10 这一行。

('+' 或 '-')表示跳转的方向，往下跳是“+”，往上跳是“-”。

- 条件跳转：如果不等于则跳转

命令	@121	
参数 1	R<n>	与之比较的值, >,.<符号后面那个值
参数 2	K or R<m>	比较的值, >,.<符号前面那个值
参数 3	K	跳转目标行

例如:

```
N10 ..
...
R1=14
N120 @121 R1 K9 K-10
N130 ...
```

表示如果 R1 不等于 9, 则往上跳转到 N10 这行。

注意: 字符 K 纯粹是一个常数的前置符。K9 所在的位置应该是一个数值, 所以 K9 表示值为 9, K-10 所在的位置应该是行号, 所以 K-10 表示返回到 N10 这行。

6.4.2 分支指令

@111 Case 分支 @111 Rn K/Rn Km ...

分支指令

命令	@111	
参数 1	R<n>	与之比较的变量
参数 2	K or R<m>	1. 比较值 1
参数 3	K	1. 跳转目标行
参数 4	K or R<m>	2. 比较值 2

...

例如:

```
N100 R2=12 (R2=13) (R2=14)
N200 @111 R2
K12 K300 (如果 R2 等于 12 就跳到 N300 这行)
K13 K400 (如果 R2 等于 13 就跳到 N400 这行)
K14 K500 (如果 R2 等于 14 就跳到 N500 这行)

N300 R0=300
N310 @100 K5000(无条件跳到 N5000, 即程序结尾 M30)

N400 R0=400
```

N410 @100 K5000(无条件跳到 N5000, 即程序结尾 M30)

N500 R0=500

N510 @100 K5000(无条件跳到 N5000, 即程序结尾 M30)

N5000 M30

本例中字符 K 也是一样, 仅仅上常数的前置符。K12\K13\K14 所在的位置应该是一个数值, 所以分别表示值为 12、13、14, 而 K300\K400\K500\K5000 所在的位置应该是行号, 所以分别表示 N300\N400\N500\N5000。

这段代码的功能是: 执行到 N200 时, 如果 R2 等于 12 就跳到 N300 这行, 如果 R2 等于 13 就跳到 N400 这行, 如果 R2 等于 14 就跳到 N500 这行。如果 R12 不等 3 个值中的任何一个, 则继续执行下一行, 即 N300。

6.4.3 循环指令

命令	循环类型	退出条件
@131	While 循环	while equal
@151	For-To 循环	

循环可以嵌套

● While 循环

命令	@13<n>	$1 \leq n \leq 6$, 即@131 到@136
参数 1	R<m>	对比的值, R 参数
参数 2	K or R<k>	比较的值, 参数 1 如果等于常数或另一 R 参数, 则执行循环
参数 3	K	参数 1 不等于参数 2, 则跳转到参数 3 指定的行

只要条件满足就一直执行, 不满足则跳到参数 3 指向的行。
While 循环的末尾必须有一个无条件跳转指令@100, 指向这个 While 循环的首行。
循环退出条件由<n>来定义。

例如:

```

N100 R6=4
N200 @131 R6 K4 K600 (如果条件不满足, 即如果 R6 不等于 4 就跳到 N600)

N210 ...
N220 @100 K-200

N600 ...

```

N5000 M30

只要 R6=4，循环体 (N200 至 N220)就一直重复。一旦条件不满足则跳转到 N600。

- For-To 循环

命令 @151 <变量> <值> <常数>

for-to 循环是一个计数循环，在变量等于这个值之前一直执行。

如果条件满足，跳转到常数指定的行。

在循环体的末尾，变量必须递增(@620)，而在循环开始必须有一个无条件跳转。

例如：

```
N190 R6=0
N200 @151 R6 K20 K400 (R6<20 往下执行循环体，R6=20 则跳转到 N400)
N210 ...
N290 @620 R6 (R6 递增 1)
```

```
N300 @100 K-200
```

例中 N210 到 N290 之间的代码会执行 20 次 (R6 依次等于 0、1、2……19)，然后执行 N400。

6.4.4 子程序

子程序的文体和调用它的主程序写在同一个.nc 文件，此时装载主程序就同时装载了子程序，如果这个子程序还要在其它文件中调用，它就必须写在一个单独的文件里面，并且放到 “\TwinCAT\CNC\” 目录下。

子程序名字应以“L”打头，加上一串数字。而这个数字与子程序中首行“子程序标记”的数字应该完全一致。子程序中的首行标记之后立即跟随插补指令。子程序必须以 M17 结尾。

定义子程序：

```
(file L2000.NC)
L2000
N100...
N110...
...
N5000 M17 (return command)
```

调用子程序:

N100 L2000 (call)

N100 L2000 P5

6.4.5 读取实际轴的位置

- 打断预读，获取轴位置

在 G 代码中读取实际轴的值，赋给指定的 R 参数

命令	@361	
参数 1	R<n>	读取的结果赋给 R 参数编号
参数 2	K<m>	常数，用于指定要读取的轴号，
		0: X axis
		1: Y axis
		2: Z axis
		3: Q1 axis
		4: Q2 axis
		...
		7: Q5 axis

例如：

```
N10 G0 X0 Y0 Z0 F24000
N30 G01 X1000
N40 @361 R1 K0 (序取 X 轴的位置，赋给 R1)
N50 R0=X
N60 G01 X=R0+R1
N70 M30
```

@361 命令隐含了打断 G 代码预读的功能，在本例中确保 N30 执行完毕才读取 X 轴位置。实际项目上可以与“取消剩余动作”结合使用。如果不想打断预读，而在执行过程中读取轴位置，则要使用指令 `#get PathAxesPos(R<a>; R; R<c>)#`

- 不中断预读，获取轴位置

命令	#get PathAxesPos(R<a>; R; R<c>)#	
参数 1	R<a>	X 轴位置赋给的 R 参数
参数 2	R	Y 轴位置赋给的 R 参数
参数 3	R<c>	Z 轴位置赋给的 R 参数

命令 `#get PathAxesPos()# reads` 读取插补轴(X, Y & Z) 的当前位置。功能类似@361，区别在于这个指令不会打断预读。这样工程师必须自己确认执行这个指令的时候轴是静止的，否则读回来的位置可能不准。要么就先用@714 打断预读，然后再读取位置。

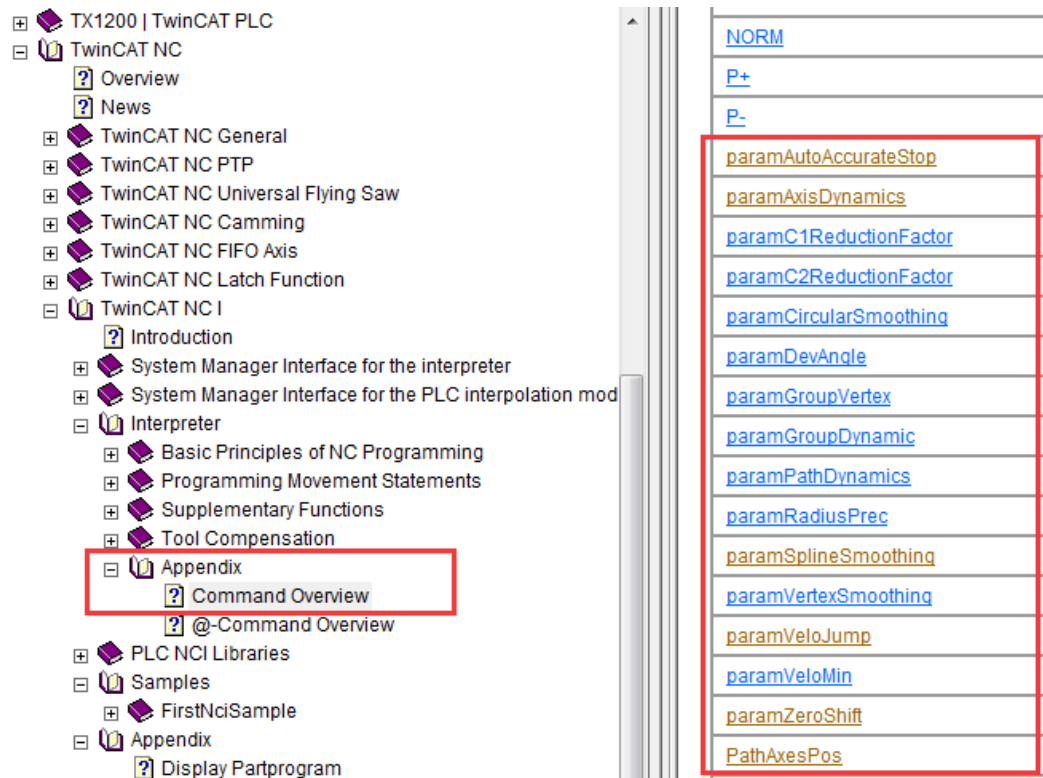
`#get PathAxesPos()#` 是@361 的变体，它需要特定的执行条件。例如：

```
@714 (可选)
N27 #get PathAxesPos( R0; R1; R20 )# (注意是以分号间隔)
```

注意：如果某个进给轴并没有绑定某个 NC 轴，最常见的是平面运动没有绑定 Z 轴，那么参数 3 指定的 R 参数值就为 0。

6.5 #Set 参数设置命令

在帮助系统中可以看到这些可设置参数的完整信息：



在 G 代码中，可以使用 `#Set 参数名(参数值)#` 来设置参数。

paramAutoAccurateStop	自动精确停车参数	m
paramAxisDynamics	轴的动态参数设置	m
paramC1ReductionFactor	C1 减速参数	m
paramC2ReductionFactor	C2 减速参数	m
paramCircularSmoothing	圆弧过渡	m
paramDevAngle	C0 减速 Deviation Angle	m
paramGroupVertex	速度平滑参数	m
paramGroupDynamic	切换组内各轴的动态参数	m
paramPathDynamics	切换合成运动的动态参数	m
paramRadiusPrec	小圆功能	m
paramSplineSmoothing	贝塞尔曲线平滑	m
paramVertexSmoothing	动作之间的平滑	m

paramVeloJump	C0 减速-最大速度跳变	m
paramVeloMin	最小速度	m
paramZeroShift	可调零偏的设置	m

例如:

Bezier 曲线平滑

使用平滑功能，可以自动在两段运动曲线之间插入一段 Bezier 平滑曲线平滑。只需要在指令中设置最大的允许偏差范围，即两段曲线连接处实际轮廓线偏离目标位置的最大允许值。启用平滑功能的好处是两段曲线过渡时没有加速度跳变，减少冲击。在 G 代码中可以随时改变允许的偏离值，把它设置为 0 就等于关闭了曲线平滑功能。

命令 `#set paramSplineSmoothing(<半径>)#`

参数 <半径> 允许偏离的球面半径

例 1:

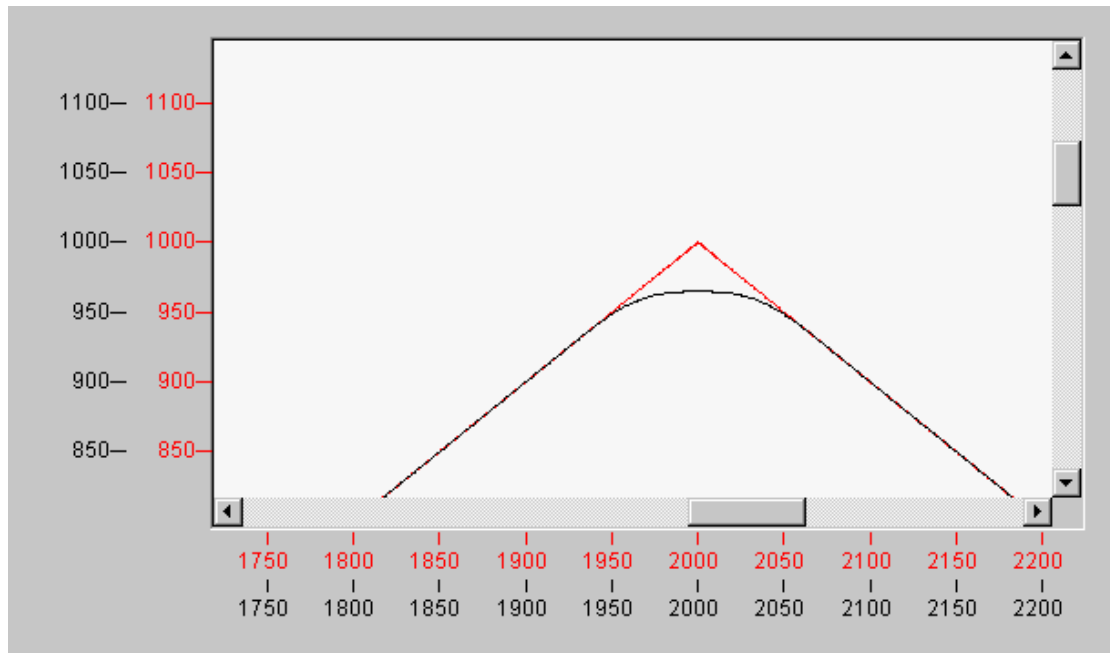
```
N10 R57=100
#set paramSplineSmoothing(R57)#
```

例 2:

```
N10 G01 X0 Y0 F6000
N20 X1000
N30 X2000 Y1000
#set paramSplineSmoothing(100)#
N40 X3000 Y0
M30
```

新的参数在这个指令的前后动作切换时就生效。在例 2 中，N30 和 N40 之间的参数就生效了。

Figure: 曲线切换时，启用或者关闭贝兹曲线平滑功能，加工轮廓线的对比



注意：G 代码文件中设置的误差半径持续有效，直到下一句设置该半径的代码执行，或者 TwinCAT 重启。

即使在一个非常尖锐的转角处也能产生平滑曲线，这种情况下为了避免加速度超限，必须适当降低速度。如此一来，加减速度保持恒定，经过这段平滑曲线的速度就可能很慢。这时现实的做法通常是在一个确定的位置启动曲线切换。为了避免手动计算夹角，可以使用“AutoAccurateStop”指令，这个指令也可以在 G 代码中初始化。（待测试）

6.6 Command 命令

在 G 代码中除了 G 开头语句外还有一些 Command 命令，和数学运算符一样，可以直接使用。详情请参考 Beckhoff Information System

Command Overview

Command	Meaning
ANG	Programming contour with angle definition
AROT	Additive Rotation
CalcInvRot	Calculate invers rotation of vector
CalcRot	Calculate rotation of vector
CDOF	Bottle neck detection off
CDON	Bottle neck detection on
CFC	Constant Feed at the Contour
CFIN	Constant Feed at the Internal Radius
CFTCP	Constant Feed of the Tool Centre Point
CIP	Circular Interpolation
CPCOF	Centerpoint correction off
CPCON	Centerpoint correction on
DelDTG	Delete distance to go
DYNVR	Dynamic Override
FCONST	Constant Feedrate Interpolation
FLIN	Linear Feedrate Interpolation
Mirror	Mirror co-ordinate system
MOD	Modulo movement
MSG	Messages from NC program
NORM	Orthogonal Contour Approach/Departure
P+	Feed direction positive
P-	Feed direction negative

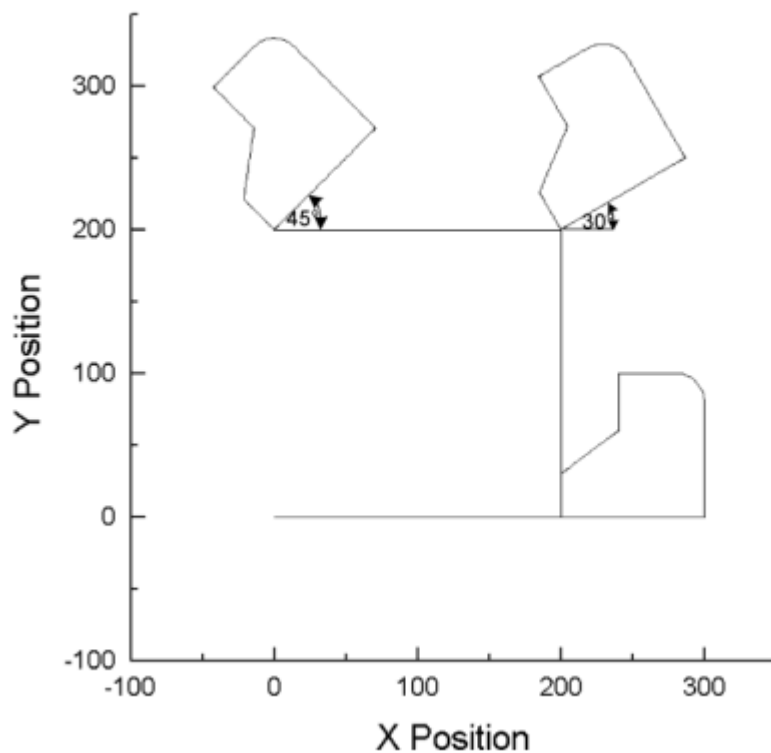
PathAxesPos	Reads current actual position
ROT	Absolute Rotation
RotExOff	Extended Rotation function off
RotExOn	Extended Rotation function on
RotVec	Calculating routing for rotation of a vector
RParam	Initialisation of R parameters
RToDwordGetBit	Changes a R parameter to DWord and checks if a defined bit is set.
SEG	Contour programming (Segment length)

例如旋转指令：ROT

Example:

```
N10 G01 G17 X0 Y0 Z0 F60000
N20 G55
N30 G58 X200 Y0
N50 L47
N60 G58 X200 Y200
N65 ROT Z30
N70 L47
N80 G58 X0 Y200
N90 AROT Z15
N100 L47
N50 M30
```

```
L47
N47000 G01 X0 Y0 Z0 (movements for zero shift & rotation)
N47010 G91 (incremental dimensions)
N47020 G01 X100
N47030 G01 Y80
N47040 G03 X-20 Y20 I-20 J0
N47050 G01 X-40
N47060 G01 Y-40
N47070 G01 X-40 Y-30
N47080 G01 Y-30
N47090 G90
N47100 M17
```



In this example, the same contour is traversed under different rotations. Since the contour (L47) is

programmed in incremental dimensions, and the starting point is described by means of the programmed zero offset shift, the rotation is clear to see.

Note:

After programming the ROT resp. AROT command the complete path vector (X, Y & Z) has to be allocated (always).

7 回溯 Retrace 和单步 SingleBlock

7.1 回溯 Retrace

7.1.1 什么是回溯

在 NCI 执行 G 代码动作的过程中，出现某些情况需要暂停。暂停之后可能要从当前位置按照刚才执行过的 G 代码路径和速度往回退，这个就叫回溯。

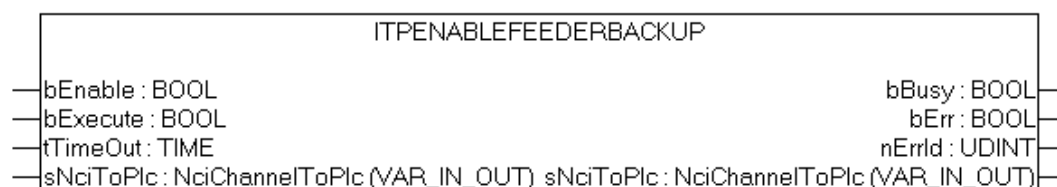
回溯的过程中如果没有干预，就会一直回溯到执行第一行 G 代码的起点位置。回溯过程中可以暂停，暂停之后可以继续回退，或者向前执行，向前执行到先前的暂停点时并不停下来，而是继续执行 G 代码直到文件结束。

7.1.2 回溯的程序处理

a) 打开和关闭回溯功能

可以用 PLC 程序打开和关闭回溯功能。如果要在执行 G 代码的过程中回退，必须在开始动作之前打开回溯功能。

功能块 (Function Block) ItpEnableFeederBackup

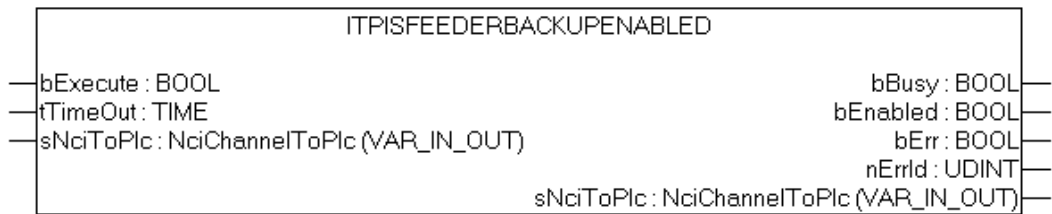


bEnable 是回溯功能的目标状态，为 True 启用，为 Off 关闭。但仅当 bExcute 的上升沿执行回溯功能的状态切换。

b) 检测回溯功能是否开启

PLC 程序也可以检测当前是否打开了回溯功能，作为是否启动 G 代码执行的条件之一。

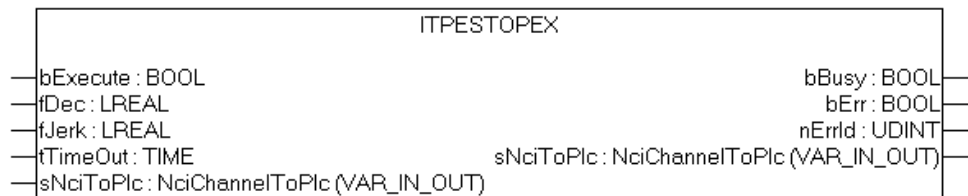
功能块 (Function Block) ItpIsFeederBackupEnabled



这是一个功能块，而不是函数。仅当 **bExecute** 的上升沿才会攻取回溯功能的开关状态。

c) G 代码暂停执行

功能块（Function Block） **ItpEStopEx**

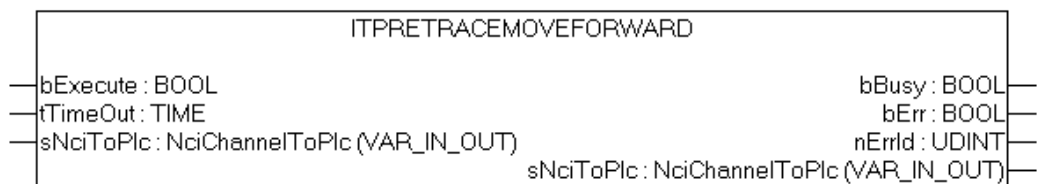


只有通过 PLC 用暂停功能停止的 NCI 通道，在确认动作完全停止以后，才可以执行回退动作。如果不需要回退的停止，可以用另一个功能块 **ItpStartStop**。这是一个功能块，仅当 **bExecute** 的上升沿才会触发暂停。

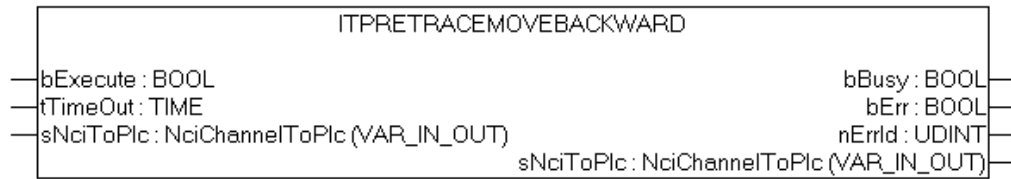
d) 触发回退或者前进动作

PLC 程序可以触发回退或者前进的动作。

功能块（Function Block） **ItpRetraceMoveForward**



功能块（Function Block） **ItpRetraceMoveBackward**

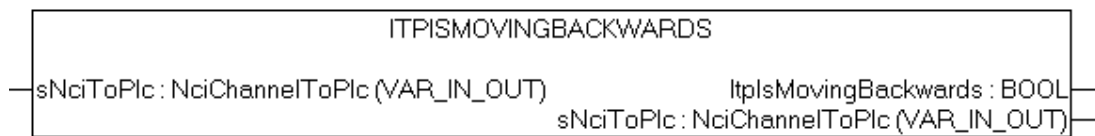


这两个是功能块，仅当通道完全停止后 bExcute 的上升沿才会触发回退或者前进。

e) 检测是否正在回退

检测当前设备是否在回退，可以用在界面上显示，PLC 也可以做相应的逻辑。

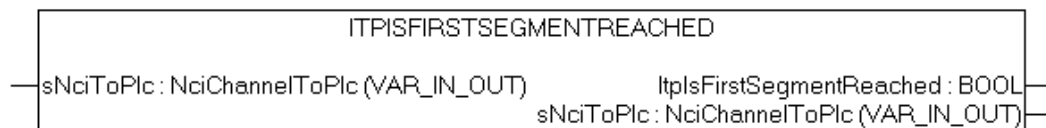
函数 (Function) ItpIsMovingBackwards



这个函数每次执行都会返回是否回退的状态，是一个 Bool 型函数。

f) 检测是否已退到 G 代码首行

函数 (Function) ItpIsFirstSegmentReached



这个函数每次执行都会返回是否回退至首行的状态，是一个 Bool 型函数。PLC 才能据此做下一步动作。

7.1.3 启用回溯功能的 NCI 例程

在第 5 章例程“NCI_GCode_MFunc.pro”的基础上，做了如下修改：

- 接口结构体 GCode_ChN_InterfaceEx

在 GCode_ChN_Interface 的基础上增加了以下功能

```

0001 TYPE GCode_Chn_InterfaceEx:
0002 STRUCT
0003
0004     bGroupEnable   :   BOOL;
0005     bGroupReset    :   BOOL;
0006     bStart         :   BOOL;
0007     bStop          :   BOOL;
0008     bEStop         :   BOOL;
0009     bLoad          :   BOOL;
0010
0011     bEnRetrace     :   BOOL;
0012     bMoveFw        :   BOOL;
0013     bMoveBw        :   BOOL;
0014
0015     bSetOverride   :   BOOL;
0016     rOverride      :   REAL:=100;
0017     sFileName      :   STRING(80):= 'C:\TwinCat\CNC\MDemo2.nc';
0018
0019     bAllAxisReady  :   BOOL;
0020     bEStoped       :   BOOL;
0021     bRetraceEnabled :   BOOL;
0022     IsFeedFromBackupList :   BOOL;
0023     IsMoveBw       :   BOOL;

```

命令接口变量

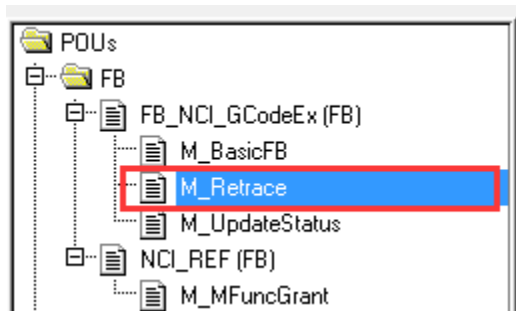
bEStop : 暂停;
 bEnRetrace : 回溯功能开关;
 bMoveFw : 前进;
 bMoveBw : 后退;

状态接口变量

bEStoped : 通道已暂停;
 bRetraceEnabled : 回溯功能已打开;
 IsFeedFromBackupList : 当前动作来自回溯指令表;
 IsMoveBw : 正在回退;

g) 功能块 FB_NCI_GCodeEx

在 FB_NCI_GCode 的基础上增加了专门处理回溯相关代码的 Action，即“M_Retrace”。



而在主程序分支中，只在 Case 30 处增加了对 M_Retrace 的引用：

```

30:(*插补运行阶段*)
  fbItpStartStop.bStart:=bStart;
  fbItpStartStop.bStop:=bStop;

  M_Retrace:(*回溯功能*)

  IF ItpChannel.NciToPlc.nJobNo = 0 THEN
    bGroupBusy:=FALSE; (*通道闲置*)
  END_IF

  IF NOT bGroupEnable THEN
    IF ItpChannel.NciToPlc.nJobNo = 0 THEN
      bGroupBusy:=FALSE; (*通道闲置*)
      bGroupReady:=FALSE;
      iStep:=40;
    END_IF
  END_IF

```

M_Retrace 的代码如下：

(*回溯相关的状态刷新：当前插补动作是否前来自 BackupList*)

```
IsFeedFromBackupList:=ItpIsFeedFromBackupList(aNciChannel[0].NciToPlc);
```

(*回溯相关的状态刷新：正在回退*)

```
IsMoveBw:=ItpIsMovingBackwards(aNciChannel[0].NciToPlc);
```

(*回溯相关的状态刷新：回退功能是否启用成功*)

```
fbIsRetraceEnabled(
  bExecute:=gbPulse1s ,
  tTimeOut:=T#500MS ,
  sNciToPlc:=ItpChannel.NciToPlc ,
  bBusy=> ,
  bEnabled=> bRetraceEnabled,
  bErr=> ,
  nErrId=> );
```

(*回溯相关的状态刷新：是否处在暂停状态*)

```
fbIsEStoped(
  bExecute:=gbPulse1s ,
  nGrpId:=GroupID ,
  tTimeOut:=T#500MS ,
  bBusy=> ,
  bEStop=>bEStoped ,
  bErr=> ,
  nErrId=> );
```

(*回溯功能的启用和禁用处理*)

```

R_TRIG_Retrace(CLK:=bEnRetrace , Q=> );
F_TRIG_Retrace(CLK:=bEnRetrace , Q=> );
TP_Retrace(IN:=R_TRIG_Retrace.Q OR F_TRIG_Retrace.Q , PT:=t#500ms , Q=>
bAcceptRetrace, ET=> );
fbRetraceEn(
  bEnable:=bEnRetrace ,
  bExecute:=bAcceptRetrace ,
  tTimeOut:=T#500MS ,
  sNciToPlc:=ItpChannel.NciToPlc ,
  bBusy=> ,
  bErr=> ,
  nErrId=> );

```

(*从暂停状态：后退*)

```

fbMoveBw(
  bExecute:=bMoveBw AND bRetraceEnabled AND bEStoped ,
  tTimeOut:=T#500MS ,
  sNciToPlc:=ItpChannel.NciToPlc ,
  bBusy=> ,
  bErr=> ,
  nErrId=> );

```

(*从暂停状态：前进*)

```

fbMoveFw(
  bExecute:=bMoveFw AND bRetraceEnabled AND bEStoped ,
  tTimeOut:= T#500MS ,
  sNciToPlc:=ItpChannel.NciToPlc ,
  bBusy=> ,
  bErr=> ,
  nErrId=> );

```

(*从回溯或者正常 G 代码执行状态：暂停*)

```

fbItpEStop(
  bExecute:= bEStop ,
  fDec:= ,
  fJerk:= ,
  tTimeOut:=t#500ms ,
  sNciToPlc:=ItpChannel.NciToPlc,
  bBusy=> ,
  bErr=> ,

```

```
nErrId=> );
```

h) 调试画面 RetracePH

TwinCAT NC PTP 及 NCI 插补通道控制界面
2016.10.23

群控PTP	ID	Ready	故障	使能	Jog +	Jog -	当前位置	点动速度
全部使能	1		0				0.0	10.0
	2		0				90.7	10.0
	3		0				0.0	10.0
	4		0				0.0	10.0

全部复位

全部置0位

G代码文件 C:\TwinCAT\CNC\MDemo2.nc M函数
Laser

插补使能	插补复位	通道运行状态	IsRunning
插补启动	装载G代码	未执行指令数	20条
插补停止	M函数延时复位	当前指令行号	N 20

回溯使能	前进
插补急停	后退
Retraced Enabled	EStoped
Is From BackList	Is Moving Backward

回溯功能相关

7.1.4 测试 Retrace 功能的操作顺序

- 1) 选择目标控制器，激活 NCI_GCode_Retrace.tsm
- 2) 打开 NCI_GCode_RetraceFB.pro，下载到目标 PLC，运行。
- 3) 进入 HMI，确认 G 代码文件 MDemo2.nc 与控制器上的 CNC 文件路径一致。
- 4) 使能 NC 轴。使用群控功能，可以节约时间。
- 5) 如有需要可以全部位置置 0
- 6) 插补使能——装载 G 代码——回溯使能——插补启动——插补急停——后退——前进
- 7) 插补停止，或者插补复位。

执行第 6、7 步的时候，注意观察状态标签

“Retrace Enabled”（回溯功能已启用）

“EStoped”（当前处于暂停状态）

“Is From BackList”（当前动作来自回溯列表）

“Is Moving Backward”（正在回退）

尤其注意回退和前进过程中，标签“M 函数 Laser”的变化。在回退过程中，M 函数的状态是不切换的，在前进阶段才会切换。至于实际项目中，回退的时候要不要处理 M 函数，需要视项目需求编程实现。

注意：

在按“插补启动”前，按“回溯使能”

在 G 代码完成前，按“插补急停”

可以在后退的过程中，再按“插补急停”

后退完成到首行，就不能再按“前进”了。

7.2 单步执行 Single Block

7.2.1 什么是单步执行 Single Block

在 NCI 执行 G 代码动作的项目中，正常生产的时候通常是一个 G 代码文件从头到尾连续执行。而设备调试或者生产维护的时候，为了便于检查机械或者其它外围信号是否正常，可能需要走一步停下来，按继续再走下一步，这个功能就是单步执行 SingleBlock。

单步执行，顾名思义就是触发一次指令只执行一行 G 代码，在 NCI 中又称一行 G 代码为一个 Block。

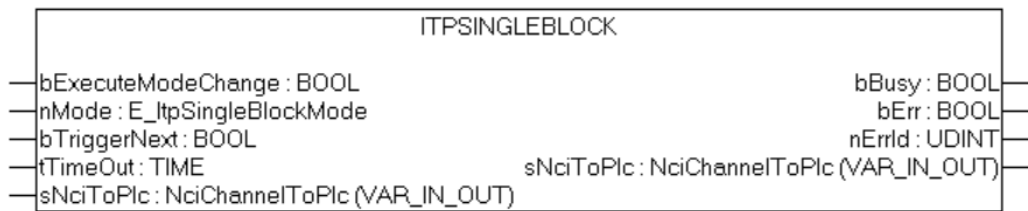
单步模式可以在 G 代码运行的过程中开启或者关闭。如果触发启动命令的时候，未启用单步模式，G 代码文件就连续运行。如果触发启动命令的时候，单步模式已启用，G 代码文件就一行一行地单步运行。

7.2.2 单步执行的程序处理

1) 打开单步功能

可以用 PLC 程序打开和关闭单步功能，引用的功能块在 TcNci.lib 中。

功能块（Function Block） ItpSingleBlock



bExecuteModeChange 上升沿触发模式切换。切换的目标模式在 nMode 中定义
nMode 是一个枚举类型 E_ItpSingleBlockMode:

ItpSingleBlockOff := 0, (*single set off*)
 ItpSingleBlockNck := 1, (* single set in the NC kernel *)
 ItpSingleBlockIntp := 16#4000 (*single set in the interpreter *)

nMode 等于 1 或者 16#4000 的时候，bExecuteModeChange 的上升沿就可以打开单步功能。

在 NC 内核 kernel 或者在插补器 interpreter 中实现单步的区别在于，使用在 NC 内核中插补时，G 代码中的指令仍然是提前预读多条缓存在插补器中，如果用 ADS 读取未执行指令数量，可以看到仍然存在多条。而插补器 interpreter 中实现单步，G 代码中的指令就是读入一行到插补器就执行一行，如果用 ADS 读取未执行指令数量结果不会超过 1 条。虽然缓存形式不同，但执行结果并没有发现明显区别：G 代码执行过程中单步功能关闭后，都可以连续执行后面的 G 代码直到结束。

2) 关闭单步功能

在上述功能块中，nMode 等于 0 的时候，bExecuteModeChange 的上升沿就可以关闭单步功能。G 代码执行过程中关闭单步功能，后续的 G 代码可以连接执行直到结束。

3) 触发单步命令

ItpSingleBlock 的输入变量 bTriggerNext 虽然可以在单步模式下触发单步命令，但实际上它的作用和 G 代码启动命令类似，甚至可以互相替换。如果关闭单步模式，它就等效于 fbItpStartStop(bStart:=)中的 bStart。如果打开单步模式，fbItpStartStop(bStart:=)中的 bStart 也可以代替 bTriggerNext 来触发单步动作。

7.2.3 启用单步功能的例程

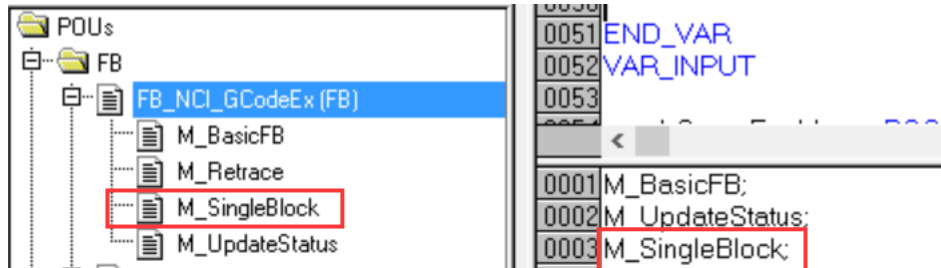
“\配套例程\第 7 章 回溯和单步执行\NCI FB SingleBlock\NCI_GCode_SingleBlockFB.pro”

它在“\第 7 章 回溯和单步执行\NCI FB Retrace\NCI_GCode_RetraceFB.pro”的基础上，做了如下修改：

- i) 接口结构体 GCode_Chn_InterfaceEx 增加了以下功能

```
(*Single Block*)
bTriggerNext      :   BOOL   := FALSE;
bSingleBlock      :   BOOL;
nMode              :   E_ItpSingleBlockMode := ItpSingleBlockOff; (*用于显示当前是否启用单步模式*)
```

j) FB_NCI_GCodeEx 中增加了 Action，专门处理接口变量。

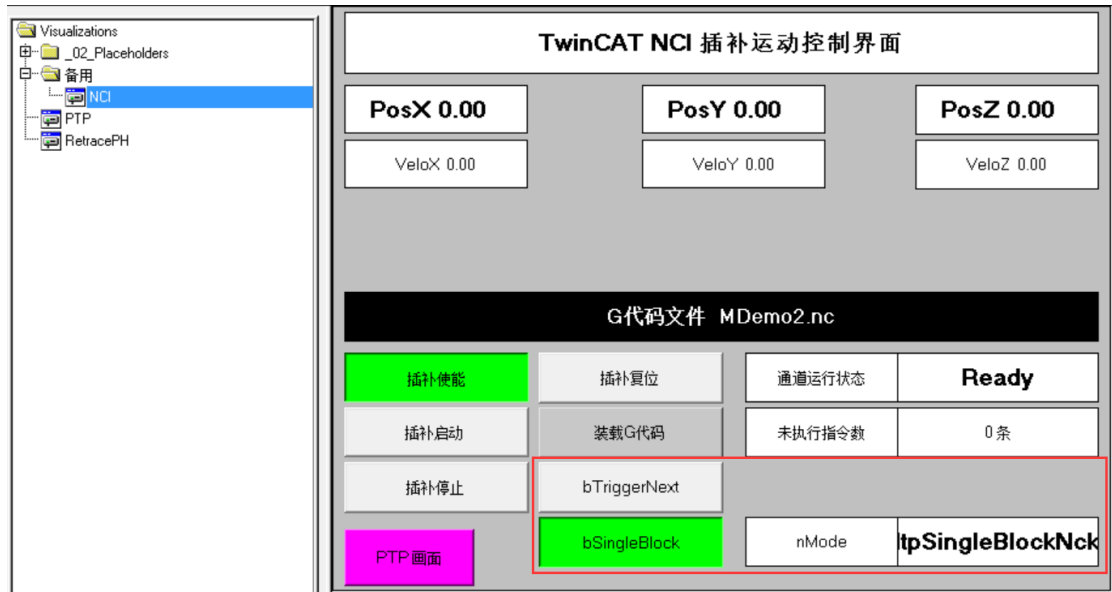


代码如下：

```
(*bSingleBlock 为 True，切换到单步模式，为 False，则禁用单步模式*)
fbRTrigSingle(CLK:=bSingleBlock , Q=> );
fbFTrigSingle(CLK:= bSingleBlock, Q=> );
fbTPSingle(IN:=fbRTrigSingle.Q OR fbFTrigSingle.Q , PT:=T#500MS , Q=>, ET=> );

nMode:=SEL(bSingleBlock,0,1);
fbSingleBlock(
    bExecuteModeChange:= fbTPSingle.Q,
    nMode:= nMode,
    bTriggerNext:=bTriggerNext,
    tTimeOut:= ,
    sNciToPlc:= ItpChannel.NciToPlc,
    bBusy=> ,
    bErr=> ,
    nErrId=> );
```

k) 调试画面 NCI



7.2.4 测试单步功能 SingleBlock 的操作顺序

- 1) 选择目标控制器，激活 NCI_GCode_Retrace.tsm
- 2) 打开 NCI_GCode_SingleBlockFB.pro，下载到目标 PLC，运行。
- 3) 进入 HMI，确认 G 代码文件 M Demo2.nc 与控制器上的 CNC 文件路径一致。
- 4) 使能 NC 轴。使用群控功能，可以节约时间。
- 5) 如有需要可以全部位置置 0
- 6) 插补使能——装载 G 代码——bSingleBlock 按下
- 7) 插补启动（重复按）——松开 bSingleBlock——插补启动
- 8) bTriggerNext（重复按）——松开 bSingleBlock——bTriggerNext
- 9) 插补停止，或者插补复位。

执行第 6、7、8 步的时候，注意观察状态标签

“nMode”的值（字符串）会在 ItpSingleBlockOff 和 ItpSingleBlockNck 之间切换。

当它为 OFF 的时候，按启动或者 bTriggerNext 都可以触发 G 代码连续执行，而它为 Nck 的时候，一次就只执行一个动作。之后切换到 OFF，不用重按启动，后续 G 代码就会继续完成了。

8 插补运动中常见需求

8.1 曲线平滑

l) 曲线过渡时的平滑处理

如果设定轨迹中相邻两段曲线的连接处是速度不连续的拐点，就会引起冲击，除非在拐点处把进给速度降到 0。为了以限定速度平稳地经过拐点，可以对拐点点用 **Bezier** 曲线进行平滑处理，轻微调整设定轨迹，以确保整条路径上的速度连续。

m) 公差球面

每个曲线过渡点都存在公差球面。为了达到平滑的目的，实际的轨迹可能会偏离预定的几何位置。在空间上允许偏离的最大值，就是公差球面。公差球面半径由用户在参数设置中预定义，并以模态参数的形式生效，它作用于所有不带精确定位或停止的曲线过渡处。公差球面半径可以自适应，以避免在加工细小线段时导致重迭。

命令 `#set paramSplineSmoothing(<半径>)#`

参数 <半径> 允许偏离的球面半径

n) 动态参数

平滑过程允许更大的动态参数。用户通过修改系统参数“减速因子 C2”，可以影响系统定义的最大曲线过渡速度 **Velo Link**，这个参数不能在 **DXD** 页面中设置，只能在线修改。

命令 `#set paramC2ReductionFactor(<C2Factor>)#`

参数 <C2Factor> C2 reduction factor

o) 曲线过渡时的共性

轨迹进入公差球面后，加速度为 0，进给速度等于曲线过渡速度。在公差球面内一直保持这个速度和加速度。修改 **Override** 引起的速度改变在公差球面内失效，轨迹走出公差球面后 **Override** 恢复生效。

8.2 速度限制

8.3 加速度限制

[paramAxisDynamics](#)

轴的动态参数设置

m

8.4 寻参

通常应该在把 PTP 轴指定给 NCI 通道之前用 PTP 指令 MC_Home 进行寻参，也可以用 G 代码来寻参，但是在 G 代码中只能同时执行一个轴的寻参动作，而在 PTP 模式就可以几个轴同时寻参。

命令 G74

结束条件 End of block

例如：

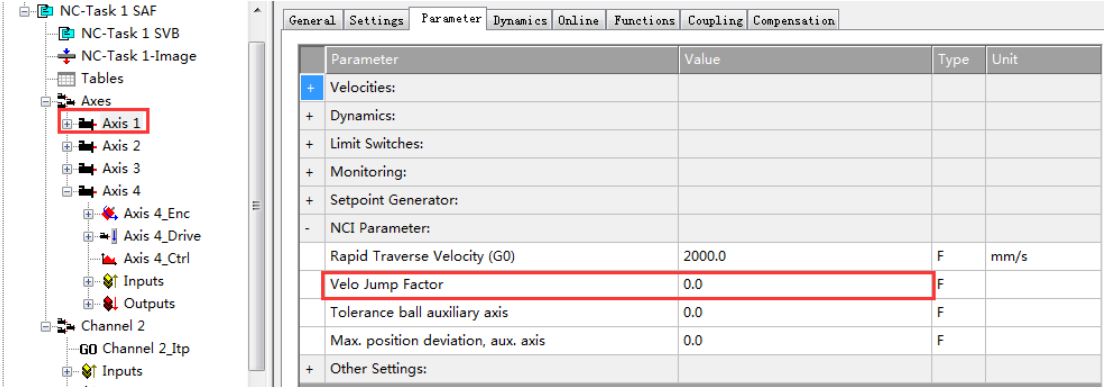
N10 G74 X

N20 G74 Y

注意：只能让插补轴寻参，即 X、Y、Z 轴。寻参指令必须在主程序中。

实际测试发现 G74 并不能控制任何轴动作，而 NCI 通道中也没有寻参参数的配置项，所以，**必须在 PTP 模式寻参完成再组合到 NCI 通道。**

8.5 速度平滑



Parameter	Value	Type	Unit
+ Velocities:			
+ Dynamics:			
+ Limit Switches:			
+ Monitoring:			
+ Setpoint Generator:			
- NCI Parameter:			
Rapid Traverse Velocity (G0)	2000.0	F	mm/s
Velo Jump Factor	0.0	F	
Tolerance ball auxiliary axis	0.0	F	
Max. position deviation, aux. axis	0.0	F	
+ Other Settings:			

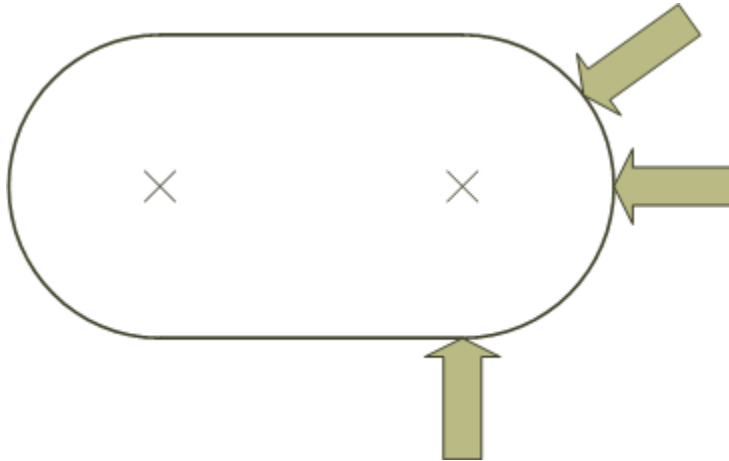
[paramVertexSmoothing](#) 动作之间的平滑

m

Path velocity at segment transitions

曲线过渡处的轨迹速度

下面操场的轮廓线为例说明进给速度的减速处理过程，其目的是为了让刀头的方向与加工轮廓的切线时刻保持一个固定的角度。



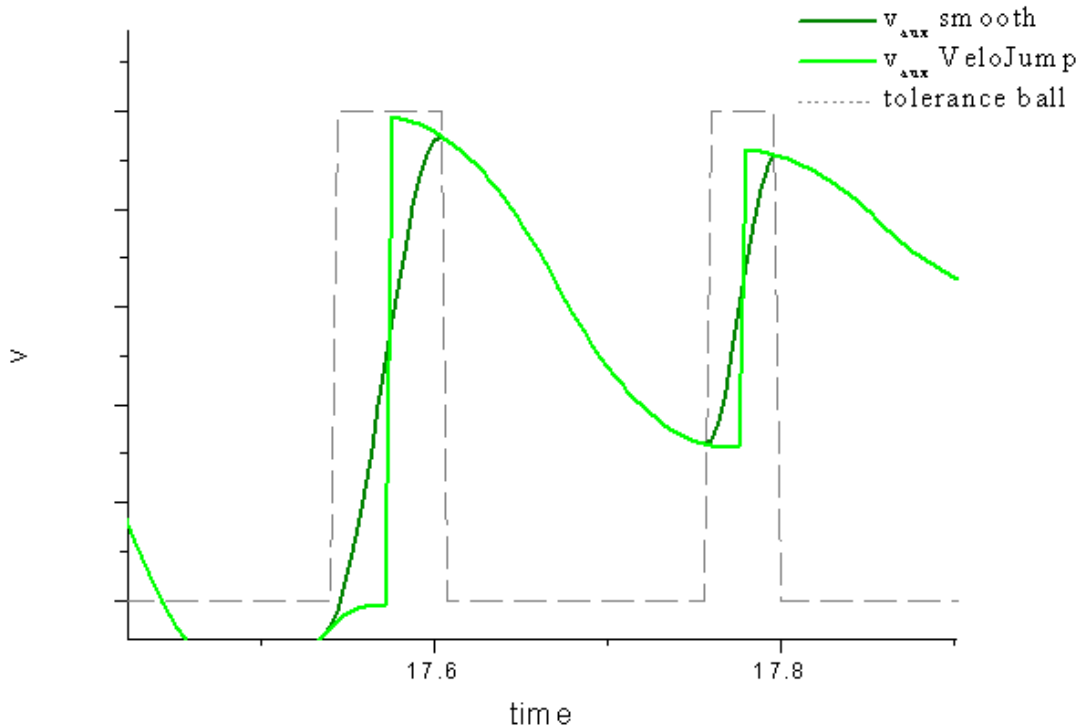
在上图的直线段刀具的方向不变。相比之下，相对于基准坐标系的方向必须在圆周内连续变化。假设直线和圆弧过渡点的轨迹速度不为 0，旋转轴就不可避免会产品一个速度跃变，而轨迹插补轴不会跃变。

辅助轴的速度跃变可以自定义，与设备相关。特殊情况是在这种曲线过渡处的轨迹速度会减到 0，或者根本不减速。可以独立设置每个插补轴的轴参数“VeloJumpFactor”，用于计算曲线过渡处实际给定的轨迹速度。

曲线过渡处的速度平滑

如上所述，在曲线过渡处可能会产生速度跃变，跃变的幅度可以在参数 VeloJump Factor 中进行配置。此外，还可以设置每个辅助轴的公差球面。这个球面是以曲线过渡点的路径为对称轴的。进入这个球面时，辅助轴的速度连续变化，以达到走出这个球面时的设定速度。换句话说，速度跃变就被限制住了。这意味着辅助轴在公差球面内会产生一个位置误差。轨迹走出公差球面时，立即切换到新的目标速度。这样可以避免位置过冲，在公差球面的边界上位置又恢复精确了。

如果定义的公差球面大于轨迹的 1/3，其半径就会自动缩小到该值。



选择和取消选择

辅助轴的公差球面是一个轴参数(IO: 0x0108)，可以从 System Manager 的轴参数页面设置或者从 PLC 经过 ADS 读入。

注意：此处所述的参数只对插补通道中的辅助轴(Q1..Q5)有效。对于插补轴(x,y,z)，参数“Veloc. discontinuity factor”、“Tolerance sphere auxiliary axis”、“Max. positional deviation, auxiliary axis”都没有影响。

p) 诊断

为了诊断的目的，可以记录每个辅助轴的公差球面和由此导致的位置偏差。还可以通过 ADS 访问这些变量，在 [group status](#) 中可以找到这些变量(IO: 0x54n and 0x56n)。

Index offset (Hex)	Access	Group type	Data type	Phys. Unit	Definition range	Description	Note
0x0000054n	Read	DXD group	REAL64	1	0/1	within the tolerance ball of the auxiliary axis n = 1.5 number of the auxiliary axis (not axis ID)	from TC V2.9 B932
0x0000056n	Read	DXD group	REAL64	1	±∞	current position error of the auxiliary axis within the tolerance ball (set value side only) only for auxiliary axes n = 1.5 number of the auxiliary axis (not axis ID)	from TC V2.9 B932

q) 公差球面半径缩小时，对速度跃变的影响

如果由于加工路径几何尺寸的原因要缩小公差球面的半径，那么这段曲线允许的速度跃变参数也相应减小，比如过渡处的轨迹速度大幅降低，以使辅助轴在更小的公差球面内动态特性不会超限。

r) 公差球面缩小时辅助轴的位置偏差

参数“maximum permitted positional deviation of the auxiliary axis（辅助轴允许的最大位置偏差）”只在公差球面由于加工路径几何尺寸的原因必须缩小的时候才起作用，目的是为了在加工较短的路径时也维持相对较高的轨迹速度，并且把产生的位置误差限定一定范围内。直到一段曲线的终点，辅助轴的速度都维持不变而同时计算位置误差。如果误差小于允许值，在这一段曲线就继续维持当前速度而产生的误差在下一段曲线补偿回来，相当于在这个曲线过渡点公差球面不起作用；如果误差大于允许值，缩小后的公差球面就生效，包括 VeloJump Factor，必要时还会降低轨迹速度。

8.6 坐标偏置

见 6.5 #Set 参数

8.7 主轴

见 6.2.1 S 指令

8.8 刀具补偿

在 Information System 中的系统描述如下：

The screenshot shows the Beckhoff Information System (BIS) interface. The left sidebar contains a tree view of the system's documentation, with the following items highlighted in a red box:

- Tool Compensation
 - Tool Data
 - Selecting and Deselecting the Le
 - Cartesian Tool Translation
 - Cutter Radius Compensation
 - Miller/Cutter Radius Compens
 - Miller/cutter radius compensa
 - Miller/cutter radius compensa
 - Departure and approach beha
 - Orthogonal Contour Approach/De**
 - Path Velocity in Arcs
 - Bottle Neck Detection

The main window displays the documentation for the **Orthogonal Contour Approach/Departure** command, from TwinCAT V2.7, Build 427.

Command	NORM
Cancellation	End of block
Programmable with	G40 G41 G42

The 'NORM' command has the effect that the contour is approached orthogonally when cutter radius compensation is switched on. The actual position of the cutter is irrelevant. When de-selecting, the last segment with active compensation is also left orthogonally.

Sample:

```
N10 G17
N20 G01 X0 Y0 Z0 F6000
N30 G42 NORM X100 Y0 D5
N40 X200
N50 G40 NORM X220 Y0
N60 M30
```

The diagram shows a circular tool (N30) approaching a rectangular contour (N40) orthogonally. The contour is shown with a hatched area representing the material being removed. The tool's path is shown as a dashed line, and the resulting contour is shown as a solid line. The tool's position is labeled N60.

Note 1:
The Norm command has hitherto only been implemented for straight line/straight line transitions.

本节只是作者的学习心得，仅涉及最基本的概念，需要在实际应用中积累更多的 Knowhow。对于图中内红线框中的内容，还没有研究透彻，有待补充。

8.8.1 刀具补偿参数

刀具补偿参数在下图中设置

	Verschl.(P5)	Verschl.(P6)	Verschl.(P7)	P8	p9
D 1	0.010000	0.000000	0.010000	20.00...	20.000...
D 2	0.000000	0.000000	0.000000	0.000...	0.000000
D 3	0.000000	0.000000	0.000000	0.000...	0.000000
D 4	0.000000	0.000000	0.000000	0.000...	0.000000
D 5	0.000000	0.000000	0.000000	0.000...	0.000000

Name	Actual Pos.	Setp. Pos.	Lag Dist.	Setp. Velo	E.
Axis 1 (X)	20.0000	20.0000	0.0000	0.0000	<0
Axis 2 (Y)	20.0000	20.0000	0.0000	0.0000	<0

最多可以有 255 套刀具补偿参数，用 D1 到 D255 来区分。每套参数包括 P0 到 P15，但目前只用到 P0 到 P10，说明如下：

- a) 第 1 列“Tnr.(P0)”值表示刀具号，G 代码执行到 Dn 刀补指令时，可以把该值赋给 Channel_ToPlc 中的 nTool 变量，可传输到 PLC。

	Tnr.(P0)	Typ(P1)	Geom.(P2)
D 1	7	20	10.000000
D 2	0	0	1.700000
D 3	0	0	0.000000
D 4	0	0	0.000000
D 5	0	0	0.000000

Name	Actual Pos.	Setp. Pos.	Lag Dist.	Setp. Velo	E.
Axis 1 (X)	-100.0000	-100.0000	0.0000	0.0000	<0
Axis 2 (Y)	-100.0000	-100.0000	0.0000	0.0000	<0

Variable	Online	Source
nTool	0x0007 (7)	nTool . sHandShake . Channel 2_ToPlc . Outputs . Channel 2 . NC-Task 1 S...

虽然这个值可以任意设置，但是通常还是把 Dn 的刀具号设置为 n，这样 G 代码的刀补指令 Dn，就包含了刀具指令 Tn。

- b) 第 2 列“Typ(P1)”指刀具的类型。目前只有两种类型：10 表示 Drill（钻头），20 表示 Shaft Cutter（切削）。分别有不同的参数，包括 X\Y\Z 偏置。
- c) P2 和 P4 参数，刀具几何尺寸的长度（P2）和半径（P4）
- d) P5 和 P7 参数，刀具磨损量的长度方向（P5）和刀头磨损（P7），只有切削类刀具才有刀头磨损。

- e) P8、P9、P10：刀具偏置参数。该参数引起 XYZ 轴的实际轮廓在 G 代码命令的基础上整体偏移。

8.8.2 刀具补偿的 G 代码

- a) T 指令：见 6.2.3 T 指令和 D 指令
如果需要 PLC 做相应的动作，才在 G 代码中添加 T1、T2 等指令。否则可以不写。
- b) D 指令：

刀具长度补偿的开启：Dn，

刀具长度补偿的关闭：D0

并确定工作平面，刀具长度补偿就是在垂直于工作平面的进给方向上进行补偿。

工作平面 G17 (XY 平面)、G18 (ZX 平面)、G19 (YZ 平面)。

示例代码：

```
N10 G17 G01 X0 Y0 Z0 F6000 (必须在 G0 或者 G1 指令同时给定工作平面)
N20 D1 X10 Y10 Z
N30 ...
N90 M30
```

刀具半径补偿的开启：G41 Dn (左补，加工内轮廓)、G42 Dn (右补，加工外轮廓)

刀具半径补偿的关闭：G40

*为了避免跳变，通常在运动指令中带补偿的开启或者关闭的选项。所以在正式加工之前，要把刀具移动到可以加工的位置并且开启补偿，类似“引线”功能。

示例代码：

```
N10 G17 G01 X0 Y0 Z0 F6000
N20 G41 X10 Y20 Z D1
N30 X30
N40 G40 X10 Y10 Z
N50 M30
```

8.8.3 刀具示例 1：长度和半径补偿

刀具参数：

	TNr.(P0)	Typ(P1)	Geom.(P2)	Geom.(P3)	Geom.(P4)	Verschl.(P5)	Verschl.(P6)	Verschl.(P7)	P8	P9
D 1	1	20	10.000000	0.000000	10.000000	0.010000	0.000000	0.010000	0.000...	0.000000
D 2	0	0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000...	0.000000

P0=1，表示 D1 调用时 PLC 可以从变量 nTool 中得到值“1”

P1=20，表示这是切削刀具 Shart Cutter。

P2=10，P4=10，表示刀架长度 10，刀具半径 10

P5=0.01，P7=0.01，仅供测试，忽略刀具长度和半径方向的磨损。(P7 为 0 会报错)

执行 G 代码：

```
N10 G0 X-50 Y-50 F6000
```

```

N20 G42 D1 X0 Y0 (启动 D1 刀具参数, G42 表示即加工轨迹的外沿)
N25 G01 X100
N30 G01 Y100
N40 G01 X0
N45 G01 Y0
N50 G40 X-50 Y-50 (此处 G40 表示关闭刀具半径补偿, 但长度补偿仍有效)
N60 D0 (关闭刀具补偿)

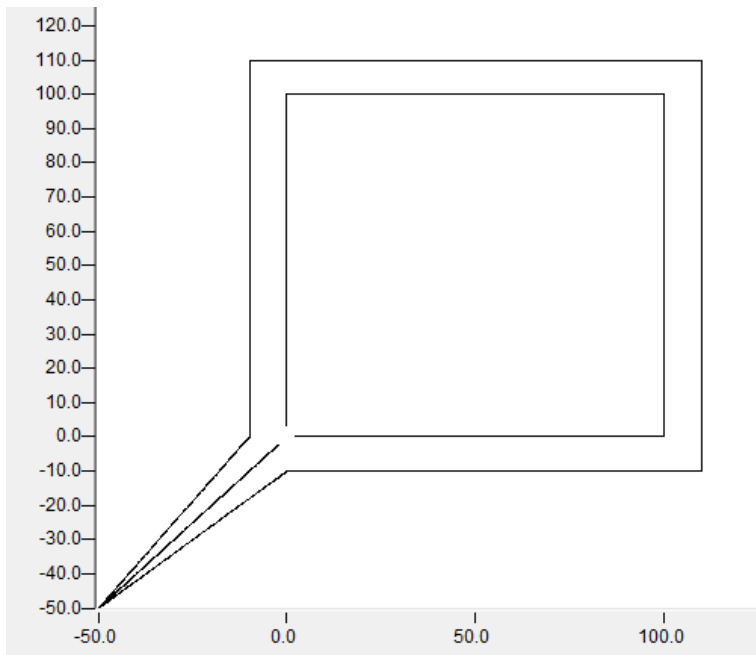
```

```

N20 G01 D1 X0 Y0
N25 G01 X100
N30 G01 Y100
N40 G01 X0
N45 G01 Y0
N100 M30

```

实测效果对比如下:



上图中, 外圈为启用刀具补偿的 X、Y 轴轨迹。内圈为禁用刀具补偿的 X、Y 轴轨迹。

- 刀具尺寸、刀具磨损参数 P2、P4、P5、P7 在 G41 和 G42 时对实际插补轴的影响测试
测试条件: G 代码画正方形, 对角坐标分别是 (0, 0) 和 (100, 100)

Geometry		Wear		补偿方向	左下角	右上角
Length	Radius	Length	Radius			
P2	P4	P5	P7			
0	0	0	0	G41	0,0	100,100
10					0,10	100,110
	10				20,20	100,100
		10			0,10	100,110

			10		20,20	100,100
10	10				20,30	100,110
0	0	0	0	G42	0,0	100,100
10					0,10	100,110
	10				0,0	120,120
		10			0,10	100,110
			10		0,0	120,120
10	10				0,10	120,130

可见，Length 只影响进给方向的轴（本例中为 Y 轴），而 Radius 影响 X 和 Y 轴。

G42 时，实际轮廓比加工路径向外 Offset 幅度为 Radius。

G41 时，实际轮廓比加工路径向内 Offset 幅度为 Radius。

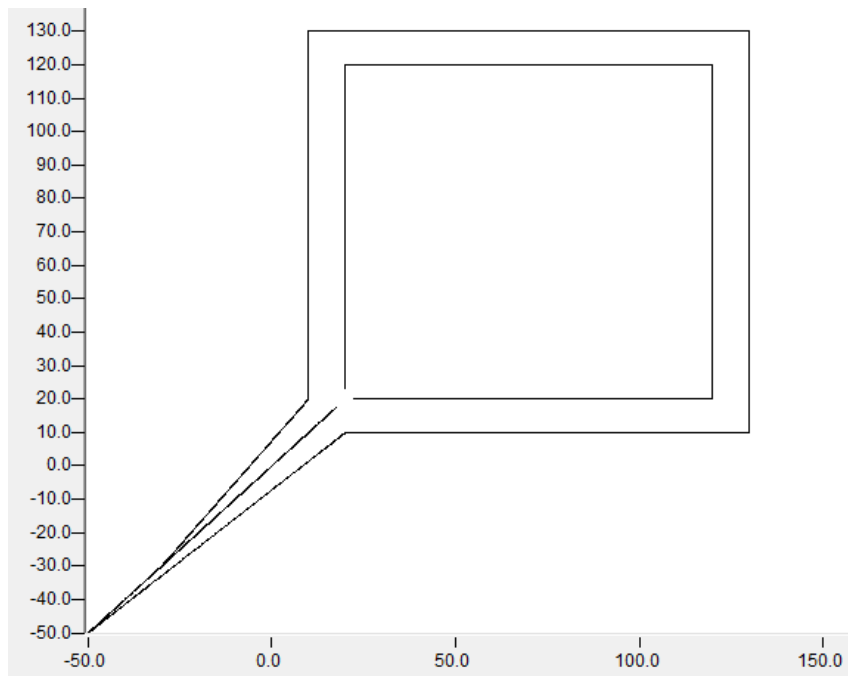
注意：实际上磨损量的取值应为**负值**，上例中仅仅是取相同的值以便计算和做关系对比。

8.8.4 刀具示例 2：刀具偏置

在上个例子的基础上，同样的 G 代码，修改 P8 和 P9，即 X、Y 偏置为 20:

	TNr.(P0)	Typ(P1)	Geom.(P2)	Geom.(P3)	Geom.(P4)	Verschl.(P5)	Verschl.(P6)	Verschl.(P7)	P8	P9
D 1	1	20	10.000000	0.000000	10.000000	0.010000	0.000000	0.010000	20.00...	20.000...
D 2	0	0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000...	0.000000

可以看到 XY 轴行走的轨迹为：



与前例相比，整个轨迹区域向 X、Y 的正方向各平移了 20 个单位。

9 在 TwinCAT 3 下使用 NCI