运动控制卡函数库

编程手册

V2.1

深圳市纳尔时代技术有限公司

2026 年 6 月

修改记录

V2.0	初始版本	2025/7/28
V2.1	补充插补函数	2026/5/20

声明

感谢您选择纳尔时代的产品。在使用之前，请务必仔细阅读该手册，以便您能够正确、安全地使用本产品。本公司不对因使用本产品而造成的任何直接或间接损失承担责任。

本手册中的信息资料仅供参考。由于改进设计和功能等原因，纳尔时代保留对本资料的最终解释权，内容如有更改，不另行通知。

调试、运动中的机器有危险！用户有责任在机器中设计有效的出错处理和安全保护机制，纳尔时代没有义务或责任对由此造成的附带的或相应产生的损失负责。

目 录

1 产品概述 7
1.1 产品简介 7
1.2 典型系统架构 7
2 应用软件开发方法 8
2.1 基于 WINDOWS 平台的应用系统结构 8
2.2 采用 C#开发应用软件的方法 9
3 API 函数一览表 14
4 功能说明与实现 28
4.1 控制卡初始化 28
4.1.1 函数列表 28
4.1.2 重点说明 29
4.1.3 例程 30
4.2 系统配置 32
4.2.1 系统配置基本概念 32
4.2.2 配置工具 Motion 33
4.2.3 EtherCAT 配置 34
4.2.4 轴配置 46
4.2.5 功能测试 50
4.3 安全机制 52
4.3.1 参数与状态配置 52
4.3.2 软限位停止功能 53
4.3.3 硬触发停止和命令停止功能 55
4.4 轴参数配置及状态读取 55
4.4.1 轴基础参数配置 56
4.4.2 轴状态配置及读取 60
4.4.3 轴编码器操作 62
4.4.4 轴 IO 操作 63
4.5 硬件资源 67

4.5.1 数字 IO 操作 67
4.5.2 编码器操作 69
4.5.3 模拟量操作 71
4.5.4 手轮操作 72
4.6 单轴基本运动 75
4.6.1 运动描述 75
4.6.2 回零运动 77
4.6.3 点位运动 79
4.6.4 定速运动 83
4.6.5 Jog 运动 85
4.6.6 变速运动 87
4.6.7 CSV 模式 88
4.6.8 CST 模式 89
4.7 插补运动 90
4.7.1 插补配置 90
4.7.2 插补运动 94
4.7.3 插补中其他操作 102
4.7.4 矩形插补 105
4.7.5 椭圆插补 106
4.7.6 单段插补 107
4.8 补偿功能 108
4.8.2 反向间隙补偿 108
4.8.1 丝杠螺距补偿 109
4.9 位置锁存功能 110
4.9.1 软件锁存 110
4.9.2 高速锁存 112
4.10 位置比较功能 114
4.10.1 软件一维位置比较 114
4.10.2 软件二维位置比较 117
4.10.3 高速一维比较 119
4.10.4 高速二维位置比较 124
4.11 EtherCAT 总线操作 128

4.11.1 总线配置 128
4.11.2 总线轴和模块操作 130
4.11.3 对象字典和 PDO 操作 130
4.12 PWM 功能 131
指令列表 131
4.13 PVT 运动 132
4.14 电子齿轮 137
指令列表 137
4.15 电子凸轮 141
4.16 龙门功能 145
指令列表 145
5 函数详细说明 148
5.1 初始化函数 148
5.1.1 初始化控制卡 148
5.1.2 控制卡复位 149
5.1.3 关闭控制卡 150
5.1.4 配置恢复出厂设置 150
5.1.5 获取控制卡相关信息 151
5.2 系统配置函数 157
5.3 安全机制相关函数 157
5.3.1 参数与状态函数 157
5.3.2 软限位停止功能 159
5.3.3 命令触发停止功能 161
5.4 轴参数相关函数 162
5.4.1 轴基础参数 162
5.4.2 轴规划状态 167
5.4.3 轴运动状态 169
5.4.4 轴编码器参数 171
NV_AxisIoSetDoBit5.4.5 轴 IO 操作 173
5.5 硬件资源相关函数 183
5.5.1 通用 IO 操作 183
5.5.2 高速 IO 操作 188

5.5.3 编码器操作 191
5.5.4 模拟量操作 193
5.5.5 手轮操作 195
5.6 单轴基本运动 199
5.6.1 回零运动 199
5.6.2 点位运动 202
5.6.4 Jog 运动 207
5.6.5 变速运动 209
5.6.6 CSV 模式 210
5.6.7 CST 模式 211
5.7 插补运动 213
5.7.1 插补配置 213
5.7.2 插补运动 220
5.7.3 插补中其他操作 234
5.7.4 矩形插补 247
5.7.5 椭圆插补 248
5.7.6 单段插补 248
5.8 补偿功能 251
5.8.1 反向间隙补偿 251
5.8.2 丝杆螺距补偿 252
5.9 锁存功能 253
5.9.1 软件锁存 253
5.9.2 高速锁存 256
5.10 位置比较功能 259
5.10.1 软件一维位置比较 259
5.10.2 软件二维位置比较 262
5.10.3 高速一维位置比较 266
5.10.4 高速二维位置比较 273
5.11 EtherCAT 总线操作 279
5.11.1 总线配置 279
5.11.2 总线轴操作 281
5.11.3 总线模块操作 282

5.11.4 总线对象字典操作 284
5.12 PWM 功能 286
5.13 PVT 运动 288
5.14 电子齿轮 290
5.15 电子凸轮 293
5.16 龙门功能 297
5.17 看门狗功能 299
5.18 密码管理 301
附件 303
F1 错误码 303
F2 常用 PDO 对象表 314
F3 回零模式 315
模式 1：寻找负限位和 Z 脉冲 315
模式 2：寻找正限位和 Z 脉冲 316
模式 3：负向寻找原点开关和 Z 脉冲 316
模式 4：正向寻找原点开关和 Z 脉冲 317
模式 5：正向寻找原点开关和 Z 脉冲 318
模式 6：负向寻找原点开关和 Z 脉冲 318
模式 7：负向寻找原点开关 ON→OFF 位置和 Z 脉冲，正限位自动反向 319
模式 8：正向寻找原点开关 OFF→ON 位置和 Z 脉冲，正限位自动反向 319
模式 9：负向寻找原点开关 OFF→ON 位置和 Z 脉冲，正限位自动反向 320
模式 10：正向寻找原点开关 ON→OFF 位置和 Z 脉冲，正限位自动反向 321
模式 11：正向寻找原点开关 ON→OFF 位置和 Z 脉冲，负限位自动反向 322
模式 12：负向寻找原点开关 OFF→ON 位置和 Z 脉冲，负限位自动反向 322
模式 13：负向寻找原点开关 OFF→ON 位置和 Z 脉冲，负限位自动反向 323
模式 14：负向寻找原点开关 ON→OFF 位置和 Z 脉冲，负限位自动反向 324
模式 17：寻找负限位 324
模式 18：寻找正限位 325
模式 19：负向寻找原点开关 ON→OFF 位置 325
模式 20：正向寻找原点开关 OFF→ON 位置 326
模式 21：正向寻找原点开关 ON→OFF 位置 326
模式 22：负向寻找原点开关 OFF→ON 位置 326

模式 23：负向寻找原点开关 ON→OFF 位置，正限位自动反向.....327
模式 24：正向寻找原点开关 OFF→ON 位置，正限位自动反向.....327
模式 25：负向寻找原点开关 OFF→ON 位置，正限位自动反向.....328
模式 26：正向寻找原点开关 ON→OFF 位置，正限位自动反向.....329
模式 27：正向寻找原点开关 ON→OFF 位置，负限位自动反向.....329
模式 28：负向寻找原点开关 OFF→ON 位置，负限位自动反向.....330
模式 29：正向寻找原点开关 OFF→ON 位置，负限位自动反向.....330
模式 30：负向寻找原点开关 ON→OFF 位置，负限位自动反向.....331
模式 33：寻找负向运行时最近的 Z 脉冲.....331
模式 34：寻找正向运行时最近的 Z 脉冲.....332
F4 信息源.....332

1 产品概述

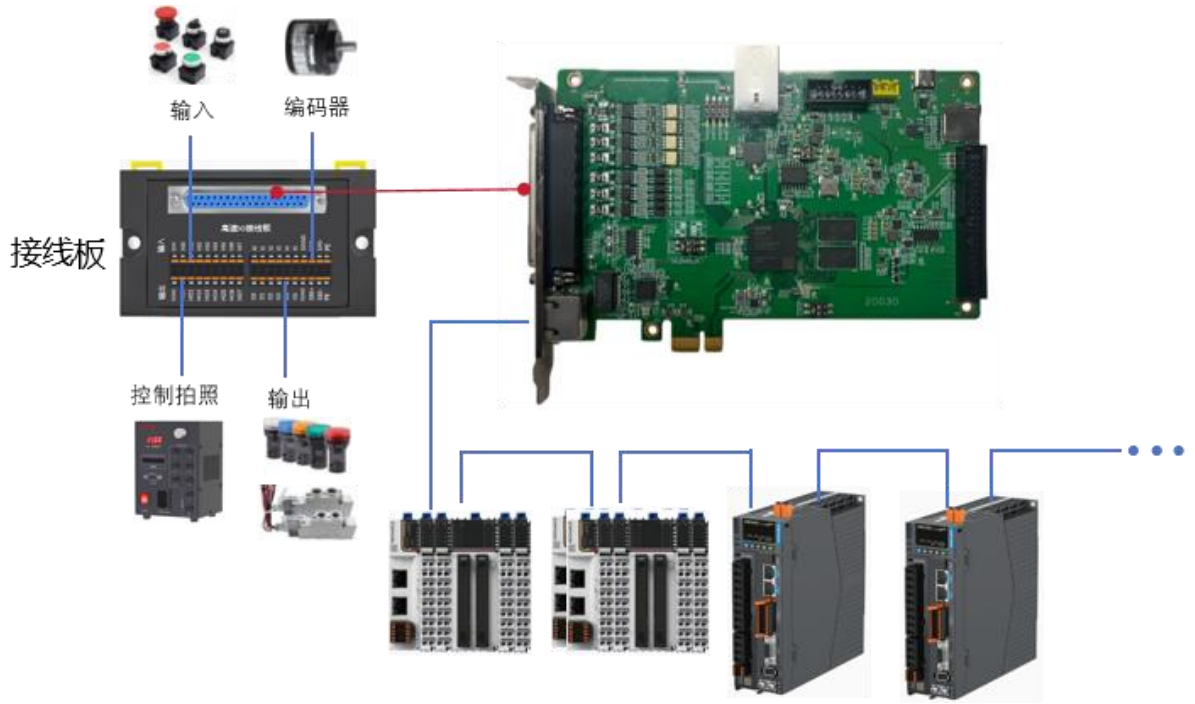
1.1 产品简介

纳尔时代系列运动控制卡支持与 PC 通过 PCI/PCIe、以太网的通讯形式，直接进行在线控制；通过提供 C#、VC、VB、PYTHON 等语言等函数库和 Windows 动态链接库，实现复杂的控制功能。用户能够将这些控制函数与自己控制系统所需的数据处理、界面显示、用户接口等应用程序模块集成在一起，构建符合特定应用需求的控制系统，以适应各种应用领域的要求。

本手册描述的功能，是针对纳尔时代系列产品，属于通用全功能版手册；具体型号产品支持的功能，请查阅对应产品的用户手册。

1.2 典型系统架构

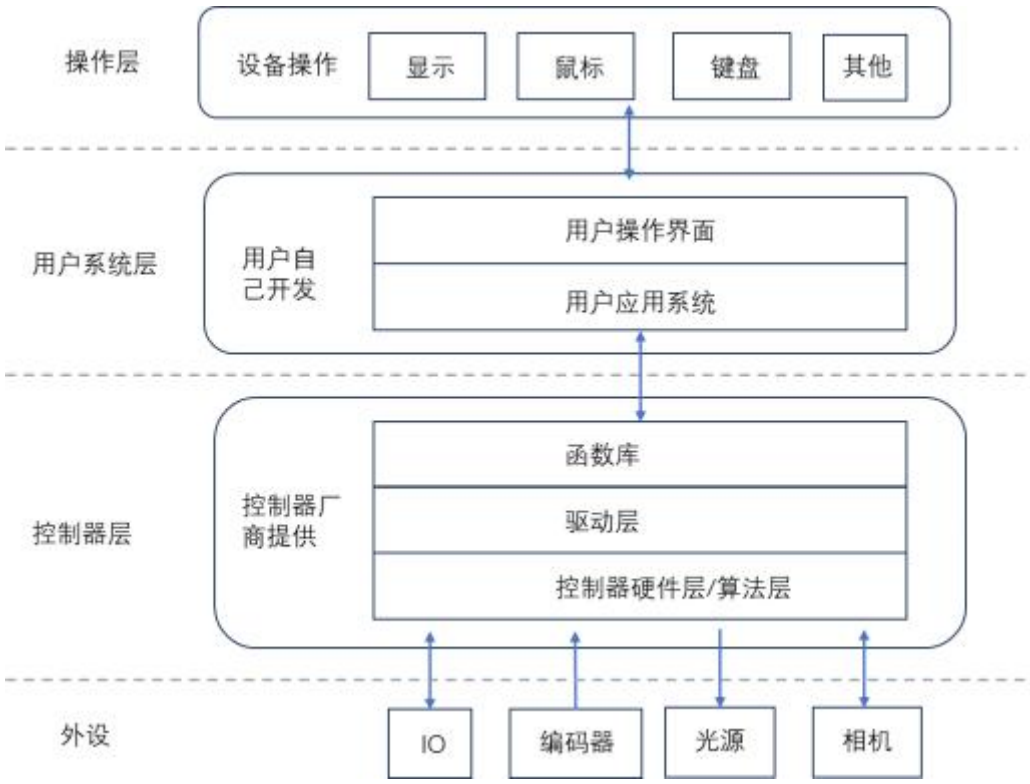
PCI/PCIe 类型控制卡典型系统架构:



2 应用软件开发方法

2.1 基于 WINDOWS 平台的应用系统结构

控制卡用户自己开发应用系统的软件架构如下图所示：



从上图可看出，整个用户的视觉系统可以分为以下四个部分：

（1）操作层：操作员通过终端设备（包括显示屏、鼠标、键盘等设备）与用户的应用系统交互数据；

（2）用户系统层：该部分为用户自己编写的程序，主要包括用户操作界面及逻辑、系统检测工艺、函数库调用等。

（3）控制器层：他为用户提供编写应用系统的软硬件平台。通过调用函数库的形式，为用户操作外部的检测、执行设备。

（4）外设：控制系统中用户需要检测、操作的设备。

因此，用户只需根据设备的检测工艺、人机交互要求就可以开发专用的控制系统；

2.2 采用 C#开发应用软件的方法

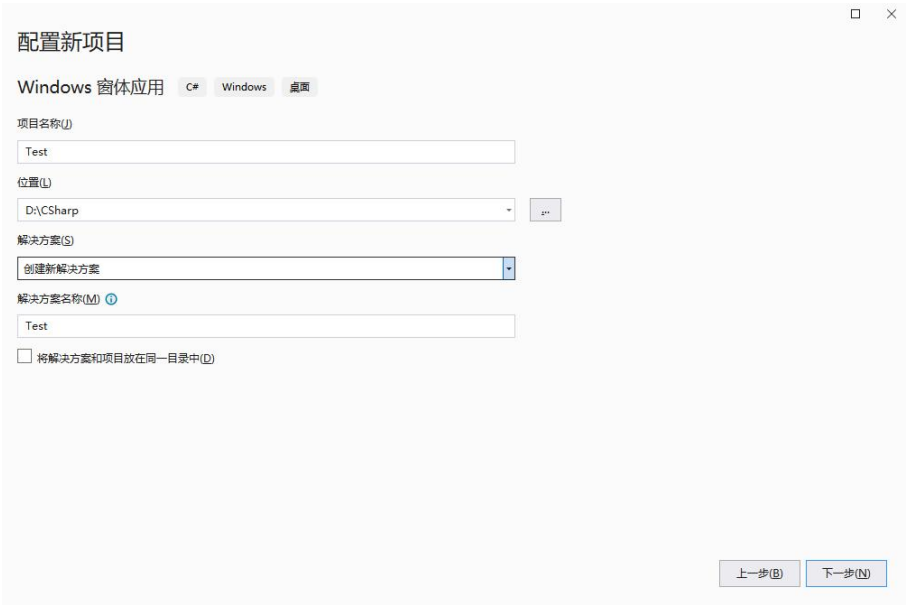
控制卡支持 windows 操作系统下 C#/VC/VB.NET/Python 环境的开发。

下面采用 C#语言，编写一个设置 Out0 输出的例程，讲解编写应用软件的一般方法。

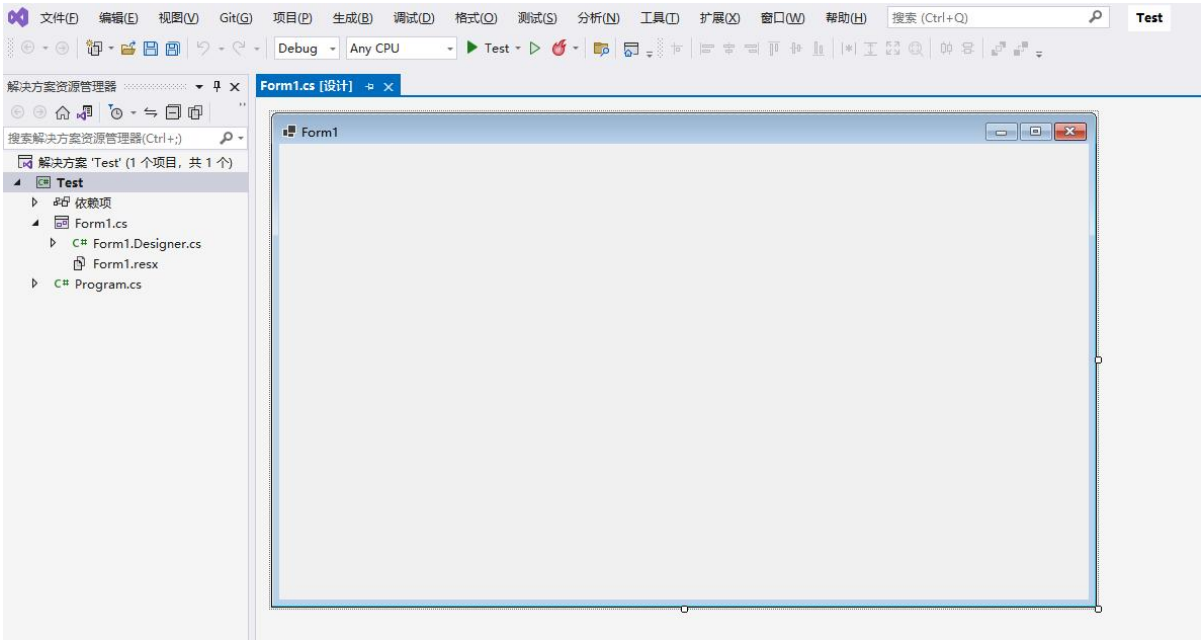
1）在磁盘上新建一个目录，如 D:\ CSharp

2）打开 VS2022，选择“windows 窗体应用　C#”工程，如下图：

点击“下一步”后，“配置新项目”，项目名称为“Test”，选择路径为新创建的路径，如下图：



再点击“下一步”，直到工程创建完成，如下图：



（3）在“Form1.cs”中，添加两个按钮，如下图：

将两个按钮添加单击响应事件，如下图：

```
9
                1 个引用
10              private void button1_Click(object sender, EventArgs e)
11              {
12
13              }
14
                1 个引用
15              private void button2_Click(object sender, EventArgs e)
16              {
17
18              }
```

（4）编译工程后，将需要的库文件拷贝到工程路径下：

将 NVCardAPI.cs 文件拷贝到 D:\CSharp\Test\Test 路径下。

将 NVCardAPI.dll 文件拷贝到 D:\CSharp\Test\Test\bin\Debug 路径下。

（5）在工程中添加库引用

在窗体的代码编辑器中，添加引用代码。

```
using CommonLib;
using static CommonLib.NVCardAPI;
```

（6）添加“启动连接”相关的响应代码：

```
private ushort m_nConnectNo = 0;
private bool m_bConnect = false;
ERROR_CODE_TYPE ret = 0;
private void button1_Click(object sender, EventArgs e)
{
    ret = NV_ConnectCloseAll();

    ScanResult_t pScanResult = new ScanResult_t();
    ret = NV_ConnectOpenAll(ref pScanResult);
    if (ret != 0)
    {
    MessageBox.Show($"NV_ConnectOpenAll连接失败，报错：{ret}");
    }
    else if (pScanResult.m_ConnectNum <= 0)
    {
    MessageBox.Show($"没有找到板卡，请确认正常插卡以及驱动成功安装！");
    }
    else
    {
    m_nConnectNo = pScanResult.m_ConnectInfo[0].m_ConnectNo;
    MessageBox.Show($"NV_ConnectOpenAll连接成功");
```

```
    }
}
```

（7）添加“输出 Out0”相关的响应代码：

```
bool OUT0_state = false;
private void button2_Click(object sender, EventArgs e)
{
if (m_nConnectNo < 0)
{
MessageBox.Show("未连接");
return;
}
if (OUT0_state == false)
{
    ret = NV_IoSetDoBit(m_nConnectNo, 0, 1);
    OUT0_state = true;
    MessageBox.Show($"设置OUT0输出TRUE");
    }
    else
    {
    ret = NV_IoSetDoBit(m_nConnectNo, 0, 0);
    OUT0_state = false;
    MessageBox.Show($"设置OUT0输出FALSE");
    }
    }
```

（8）编译工程，并运行。

点击“启动连接”按钮，连接扩展板，如果函数识别到了接口板，弹出“连接成功”消息框；否则弹出“没有找到板卡，请确认正常插卡以及驱动成功安装!”消息框。

点击“输出 Out0”按钮，输出低电平；后续，循环点击该按钮，输出高低电平来回切换。

3 API 函数一览表

控制卡支持的函数列表如下：

4.1 控制卡初始化		
控制卡扫描	NV_ConnectPCIScan	扫描 PCI/PCIe 卡
	NV_ConnectNETScan	扫描网络卡
	NV_ConnectScan	扫描工控机内所有卡
控制卡初始化	NV_ConnectOpenPCI	打开单张 PCI/PCIe 卡
	NV_ConnectOpenPCIAll	打开所有 PCI/PCIe 卡
	NV_ConnectOpenNET	打开单张网络卡
	NV_ConnectIsOnline	控制卡连接是否在线
控制卡关闭	NV_ConnectClose	关闭单张控制卡
	NV_ConnectCloseAll	关闭所有控制卡
控制卡复位	NV_ConnectColdReset	控制卡冷复位
	NV_ConnectWarmReset	控制卡软复位
	NV_ConfigRecovery	恢复出厂设置
控制卡信息	NV_ConfigGetProductInfo	获取产品信息
	NV_ConfigGetDllVersion	获取动态库版本及发布时间
	NV_ConfigGetSoftVersion	获取固件版本及发布时间
	NV_ConfigGetHardVersion	获取硬件版本及发布时间
	NV_ConfigGetAxisNum	获取控制轴数量
	NV_ConfigGetEncoderNum	获取编码器数量
	NV_ConfigGetIoNum	获取通用 IO 数量
	NV_ConfigGetHSIoNum	获取高速 IO 数量
	NV_ConfigGetCmpNum	获取位置比较器数量
	NV_ConfigGetLatchNum	获取位置锁存器数量
	NV_ConfigGetInputNumList	获取 IO 输入数量
	NV_ConfigGetOutputNumList	获取 IO 输出数量
	NV_ConfigGetAxisNumList	获取各种类型的轴的数量
	NV_ConfigGetEncoderNumList	获取各种类型的编码器的数量
	NV_ConfigGetFunNumList	获取各种功能通道的数量

4.3 安全机制		
参数与状态配置	NV_AxisSetStopParam	设置轴的异常停止参数
	NV_AxisGetStopParam	获取轴的异常停止参数
	NV_AxisGetStopReason	获取轴运动停止的原因
	NV_AxisClrStopReason	清除轴运动停止原因的记录
限位处理功能	NV_AxisSetSoftLimit	设置轴的软限位功能
	NV_AxisGetSoftLimit	获取轴的软限位功能配置
	NV_AxisSetCircleLimit	设置圆弧限位保护功能
	NV_AxisGetCircleLimit	获取圆弧限位保护功能
	NV_AxisSetErrorBand	设置超差停止功能
	NV_AxisGetErrorBand	获取超差停止功能状态
命令停止功能	NV_EmgStop	紧急停止所有轴
	NV_AxisStop	指定轴停止运动

4.4 轴参数配置及状态读取		
轴基础参数配置	NV_AxisSetEnable	设置轴使能
	NV_AxisGetEnable	获取轴使能状态
	NV_AxisSetBusMode	设置总线轴控制模式
	NV_AxisGetBusMode	获取总线轴控制模式
	NV_AxisSetType	设置轴类型
	NV_AxisGetType	获取轴类型
	NV_AxisSetStepPulse	设置轴脉冲类型
	NV_AxisGetStepPulse	获取轴脉冲类型
	NV_AxisSetUnit	设置轴的当量
	NV_AxisGetUnit	获取轴的当量
	NV_AxisSetMaxVel	设置轴的最大速度
	NV_AxisGetMaxVel	获取轴的最大速度
	NV_AxisSetMaxAcc	设置轴的最大加速度
	NV_AxisGetMaxAcc	获取轴的最大加速度
	NV_AxisGetMaxAcc	获取轴的最大加速度
轴状态配置	NV_AxisSetPrfPos	修改轴的当前规划位置

及读取	NV_AxisGetPrfPos	获取轴的当前规划位置
	NV_AxisGetPrfVel	获取轴的当前规划速度
	NV_AxisGetPrfAcc	获取轴的当前加速度
	NV_AxisGetPrfMode	获取轴的运动模式
	NV_AxisCheckDone	检测轴状态
	NV_AxisGetStatus	获取轴运动的各种状态
	NV_AxisSetReached	设置轴到位检测
	NV_AxisGetReached	获取轴到位检测状态
轴编码器配置	NV_AxisSetEncMapping	设置轴的编码器映射关系
	NV_AxisGetEncMapping	获取轴的编码器映射关系
	NV_AxisSetEncPos	修改轴的编码器位置
	NV_AxisGetEncPos	获取轴的编码器位置
轴 IO 操作	NV_AxisIoSetMapping	设置轴 IO 映射关系
	NV_AxisIoGetMapping	设置轴 IO 映射关系
	NV_AxisIoSetEnable	设置轴 IO 专用使能
	NV_AxisIoGetEnable	读取轴 IO 专用使能
	NV_AxisIoSetDiLogic	设置轴专用输入信号的有效电平
	NV_AxisIoGetDiLogic	读取轴专用输入信号的有效电平
	NV_AxisIoGetDi	读取轴专用 IN 信号（按照实际物理端口读取）
	NV_AxisIoGetDiStatus	读取轴专用 IN 信号（按照 IO 映射后的端口读取）
	NV_AxisIoSetDoBit	设置轴专用输出信号
	NV_AxisIoGetDoBit	读取轴专用输出信号
	NV_AxisIoSetOrgLogic	设置轴原点输入信号的有效电平
	NV_AxisIoGetOrgLogic	读取轴原点输入信号的有效电平
	NV_AxisIoSetEzLogic	设置轴 EZ 输入信号的有效电平
	NV_AxisIoGetEzLogic	读取轴 EZ 输入信号的有效电平
	NV_AxisIoSetHardLimit	设置轴的硬件限位功能
	NV_AxisIoGetHardLimit	获取轴的硬件限位功能状态
	NV_AxisIoSetDecStop	设置轴的硬件减速停止功能
	NV_AxisIoGetDecStop	获取轴的硬件减速停止功能状态

	NV_AxisIoSetEmgStop	设置轴的硬件急停功能
	NV_AxisIoGetEmgStop	获取轴的硬件急停功能状态
	NV_AxisIoSetAlarm	设置伺服报警功能
	NV_AxisIoGetAlarm	获取伺服报警功能状态
	NV_AxisIoSetInp	设置伺服到位功能
	NV_AxisIoGetInp	获取伺服到位功能状态
	NV_AxisIoSetRdy	设置伺服准备好功能
	NV_AxisIoGetRdy	获取伺服准备好功能状态

4.5 硬件资源		
通用 IO 操作函数	NV_IoGetDi	读取一批输入信号
	NV_IoGetDiBit	读取单个输入口状态
	NV_IoSetDiCountMode	设置输入的计数模式
	NV_IoGetDiCountMode	获取输入的计数模式
	NV_IoSetDiCountValue	修改当前输入的计数值
	NV_IoGetDiCountValue	获取当前输入的计数值
	NV_IoSetDiFilter	设置滤波时间
	NV_IoGetDiFilter	获取滤波时间
	NV_IoSetDo	设置批量输出
	NV_IoGetDo	获取批量输出
	NV_IoSetDoMask	有条件地设置批量输出
	NV_IoSetDoBit	设置单个输出
	NV_IoGetDoBit	获取单个输出的电平
	NV_IoSetDoBitReverse	设置单个输出按时翻转
	NV_IoSetDoBitPulse	控制输出信号产生脉冲波形
	NV_IoClrDoBitPulse	中断输出信号产生脉冲波形
高速 IO 操作函数	NV_HSIoGetDi	高速 IO 读取一批输入信号
	NV_HSIoGetDiBit	高速 IO 读取单个输入口状态
	NV_HSIoSetDo	高速 IO 设置批量输出
	NV_HSIoSetDoMask	高速 IO 有条件地设置批量输出
	NV_HSIoGetDo	高速 IO 获取批量输出
	NV_HSIoSetDoBit	高速 IO 设置单个输出

	NV_HSIoGetDoBit	高速 IO 获取单个输出的电平
	NV_HSIoSetDiFilter	设置输入的滤波时间
	NV_HSIoGetDiFilter	获取输入的滤波时间
编码器操作	NV_EncoderSetMode	设置编码器的工作模式
	NV_EncoderGetMode	获取编码器的工作模式
	NV_EncoderSetFilter	设置编码器的滤波参数
	NV_EncoderGetFilter	读取编码器的滤波参数
	NV_EncoderSetPulsePos	设置编码器的脉冲计数值
	NV_EncoderGetPulsePos	读取编码器的脉冲计数值
模拟量操作	NV_ADSetType	设置 AD 的采样类型
	NV_ADGetType	读取 AD 采样类型
	NV_ADGetValue	读取 AD 值
	NV_DASetEnable	设置 DA 使能
	NV_DAGetEnable	读取 DA 使能
	NV_DASetValue	设置 DA 值
	NV_DAGetValue	读取 DA 值
手轮操作	NV_HandwheelSetEncMapping	设置手轮编码器映射关系
	NV_HandwheelGetEncMapping	获取手轮编码器映射关系
	NV_HandwheelSetEncoderPos	设置手轮编码器值
	NV_HandwheelGetEncoderPos	读取手轮编码器值
	NV_HandwheelGetHardInStatus	获取手轮上的输入信号状态
	NV_HandwheelSetHardAxisSel	设置手轮硬件轴选挡位对应的控制轴
	NV_HandwheelGetHardAxisSel	获取手轮硬件轴选挡位对应的控制轴
	NV_HandwheelSetHardRatioSel	设置手轮硬件倍选挡位对应的倍率值
	NV_HandwheelGetHardRatioSel	获取手轮硬件倍选挡位对应的倍率值
	NV_HandwheelSetSoftParam	设置手轮软件控制参数
	NV_HandwheelGetSoftParam	获取手轮软件控制参数
	NV_HandwheelMove	启动手轮运动

	NV_HandwheelStop	退出手轮运动

4.6 单轴基本运动		
回零运动	NV_HomeSetProfile	设置回零的参数
	NV_HomeGetProfile	获取回零的参数
	NV_HomeStatus	获取回零的状态
	NV_HomeError	获取回零失败的原因
	NV_HomeMove	启动回零运动
点位运动	NV_PmoveSetProfile	设置定长运动的参数
	NV_PmoveGetProfile	获取定长运动的参数
	NV_PmoveAbsolute	设置绝对位置定长参数并启动运动
	NV_PmoveRelative	设置相对位置定长参数并启动运动
	NV_PmoveTwice	设置两段参数并启动运动
	NV_PmoveChangePos	修改点位运动的目标位置或者在停止的时候进行新的一个运动
	NV_PmoveGetPlanInfo	获取规划信息
定速运动	NV_VmoveSetProfile	设置定速运动的参数
	NV_VmoveGetProfile	获取定速运动的参数
	NV_VmoveMove	软件启动定速运动
Jog 运动	NV_JogSetProfile	设置 Jog 运动的参数
	NV_JogGetProfile	获取 Jog 运动的参数
	NV_JogSetDiConfig	设置 Jog 硬件信号，并启动 Jog 运动
	NV_JogGetDiConfig	读取 Jog 硬件信号
	NV_JogMove	启动 Jog 运动
在线变速	NV_AxisChangeSpeed	修改点位运动的速度
	NV_AxisChangeSpeedByRate	修改点位运动的速度的比例
	NV_AxisChangeSpeedAcc	修改点位运动的速度、加速度、减速度
	NV_AxisChangeSpeedAccByRate	修改点位运动的速度、加速度比例
CSV 模式	NV_AxisSetTargetVelocity	设置控制目标速度，需要切换到对应的模式
	NV_AxisGetTargetVelocity	获取设置的目标速度
	NV_AxisGetActualVelocity	获取实时速度

CST 模式	NV_AxisSetTarget	设置控制目标转矩，需要切换到对应的模式
	NV_AxisGetTargetTorque	获取设置的目标转矩
	NV_AxisGetActualTorque	获取转矩
	NV_AxisSetMaxTorque	设置最大转矩
	NV_AxisGetMaxTorque	获取最大转矩

4.7 插补运动		
插补配置	NV_CrdSetLookAhead	设置小线段前瞻功能
	NV_CrdGetLookAhead	获取设置小线段前瞻功能状态
	NV_CrdSetPrm	设置插补坐标系参数
	NV_CrdGetPrm	获取插补坐标系参数
	NV_CrdSetPrfMax	设置插补坐标系运动限制参数
	NV_CrdGetPrfMax	获取插补坐标系运动限制参数
	NV_CrdSetProfile	设置插补坐标系运动参数
	NV_CrdGetProfile	获取插补坐标系运动参数
	NV_CrdSetLinkMode	设置插补坐标系拐角参数
	NV_CrdGetLinkMode	获取插补坐标系拐角参数
	NV_CrdSetOverride	设置插补倍率
	NV_CrdGetOverride	获取插补倍率
	NV_CrdStatus	获取插补坐标系运动状态
	NV_CrdPosition	获取插补坐标系各轴位置
	NV_CrdOpen	插补开启
	NV_CrdClose	关闭连续插补缓冲区
	NV_CrdStart	连续插补启动
	NV_CrdPause	插补暂停
	NV_CrdStop	插补停止
直线插补	NV_CrdLnXY	两轴直线插补
	NV_CrdLnXYZ	三轴直线插补
	NV_CrdLnXYZA	四轴直线插补
	NV_CrdLnMove	多轴直线插补
圆弧插补	NV_CrdArcXYR	半径模式的 XY 平面圆弧插补

	NV_CrdArcXYC	圆心模式的 XY 平面圆弧插补
	NV_CrdArcYZR	半径模式的 YZ 平面圆弧插补
	NV_CrdArcYZC	圆心模式的 YZ 平面圆弧插补
	NV_CrdArcZXR	半径模式的 ZX 平面圆弧插补
	NV_CrdArcZXC	圆心模式的 ZX 平面圆弧插补
	NV_CrdArcXYZ	三点模式的空间圆弧插补
	NV_CrdHelixXYRZ	半径模式的 XY 平面螺旋线插补
	NV_CrdHelixXYCZ	圆心模式的 XY 平面螺旋线插补
	NV_CrdHelixYZRX	半径模式的 YZ 平面螺旋线插补
	NV_CrdHelixYZCX	圆心模式的 YZ 平面螺旋线插补
	NV_CrdHelixZXRY	半径模式的 ZX 平面螺旋线插补
	NV_CrdHelixZXCY	圆心模式的 ZX 平面螺旋线插补
	NV_CrdArcMoveC	圆心模式的多轴圆弧插补
	NV_CrdArcMoveR	半径模式的多轴圆弧插补
	NV_CrdArcMove3P	三点模式的多轴圆弧插补
插补中其他操作	NV_CrdBufDo	在插补中插入缓存按组 IO 输出
	NV_CrdBufDoBit	在插补中插入缓存按位 IO 输出
	NV_CrdBufDoBitDelayToStart	在插补中相对于轨迹起点 IO 滞后输出
	NV_CrdBufDoBitDelayToStop	在插补中相对于轨迹终点 IO 滞后输出
	NV_CrdBufDoBitAheadToStop	在插补中相对于轨迹终点 IO 提前输出
	NV_CrdBufDoBitClearAction	在插补中清除段内未执行完的 IO
	NV_CrdBufDoBitReverseOut	在插补中 IO 输出延时翻转
	NV_CrdBufDoBitDelayOut	在插补中 IO 延时输出
	NV_CrdDoSetPauseParam	配置在插补中暂停或者异常时 IO 的操作
	NV_CrdDoGetPauseParam	读取在插补中暂停或者异常时 IO 的操作配置值
	NV_CrdBufWaitIn	在插补中插入条件等待功能
	NV_CrdBufSetDoBitPulse	在插补中插入输出 IO 波形功能
	NV_CrdBufClrDoBitPulse	在插补中清除输出脉冲波形
	NV_CrdBufDelay	在插补中插入延时等待功能
	NV_CrdBufDA	在插补中插入 DA 输出功能

	NV_CrdBufPWM	在插补中插入 PWM 输出功能
	NV_CrdBufPWMAheadStop	在插补轨迹段终点 PWM 提前停止功能
	NV_CrdBufPmove	在插补中启动坐标系外的轴进行定长运动
	NV_CrdBufGear	在插补中跟随下一段轨迹运动一段距离
	NV_CrdBufFollow	在插补中跟随下一段轨迹运动开始按比例跟随插补速度运动
	NV_CrdBufVmove	在插补中开始定速运动
	NV_CrdBufStop	在插补中停止坐标系外的轴
矩形插补	NV_CrdRectangle	连续插补中矩形插补指令
椭圆插补	NV_CrdEllipse	启动椭圆插补运动

4.8 补偿功能		
反向间隙补偿	NV_AxisSetBacklash	设置反向间隙功能
	NV_AxisGetBacklash	获取反向间隙补偿功能状态
螺距补偿	NV_AxisSetPitchErr	设置丝杆螺距补偿功能
	NV_AxisGetPitchErr	获取丝杆螺距补偿功能状态

4.9 锁存功能		
软件锁存	NV_LatchSetMode	设置锁存功能参数
	NV_LatchGetMode	获取锁存功能参数
	NV_LatchSetSource	设置锁存的数值的来源
	NV_LatchGetSource	获取锁存设定的数值的来源
	NV_LatchClear	清除锁存数据
	NV_LatchStatus	获取锁存的锁存状态
	NV_LatchValue	获取锁存的数值
高速锁存	NV_HSLatchSetMode	设置锁存功能
	NV_HSLatchGetMode	获取锁存功能状态
	NV_HSLatchSetSource	设置锁存的数值的来源

	NV_HSLatchGetSource	获取锁存设定的数值的来源
	NV_HSLatchClear	清除锁存数据
	NV_HSLatchStatus	获取锁存的状态
	NV_HSLatchValue	获取锁存的数值

4.10 比较功能		
软件一维位置比较	NV_CmpSetEnable	设置软件一维位置比较器使能
	NV_CmpGetEnable	读取软件一维位置比较器使能
	NV_CmpSetSource	设置软件一维位置比较器
	NV_CmpGetSource	读取软件一维位置比较器设置
	NV_CmpClear	清除软件一维位置比较（比较点、比较状态等）
	NV_CmpAddPoint	添加软件一维位置比较点
	NV_CmpAddPoints	添加软件一维位置比较点（多个）
	NV_CmpStatus	读取当前软件一维比较状态
软件二维位置比较	NV_Cmp2DSetEnable	设置软件二维位置比较器使能
	NV_Cmp2DGeEnable	读取软件二维位置比较器使能
	NV_Cmp2DSetSource	设置软件二维位置比较器
	NV_Cmp2DGeSource	读取软件二维位置比较器设置
	NV_Cmp2DClear	清除软件二维位置比较（比较点、比较状态等）
	NV_Cmp2DAddPoint	添加软件二维位置比较点
	NV_Cmp2DAddPoints	添加软件二维位置比较点（多个）
	NV_Cmp2DStatus	读取当前软件二维比较状态
高速一维比较	NV_HSCmpSetMode	设置高速一维比较模式
	NV_HSCmpGetMode	读取高速一维比较模式
	NV_HSCmpSetSource	设置高速一维比较位置源
	NV_HSCmpGetSource	读取高速一维比较位置源
	NV_HSCmpSetOuputMode	设置高速一维比较输出模式
	NV_HSCmpGetOuputMode	读取高速一维比较输出模式
	NV_HSCmpSetOutPulse	配置高速一维比较输出信号及波形
	NV_HSCmpGetOutPulse	读取高速一维比较输出信号及波形

	NV_HSCmpStartPulse	启动比较器的脉冲输出
	NV_HSCmpStopPulse	停止比较器的脉冲输出
	NV_HSCmpClear	清除已添加的所有高速位置比较点
	NV_HSCmpSetPoint	设置高速一维比较位置
	NV_HSCmpGetPoint	读取高速一维比较位置
	NV_HSCmpAddPoints	设置高速一维比较的队列先入先出比较
	NV_HSCmpSetLinear	设置高速一维比较线性模式参数
	NV_HSCmpGetLinear	读取高速一维比较线性模式参数设置
	NV_HSCmpSetOffset	设置高速一维比较比较点的整体偏移位置值
	NV_HSCmpGetOffset	读取高速一维比较比较点的整体偏移位置值
	NV_HSCmpTableStart	高速一维比较列表比较启动
	NV_HSCmpStatus	获取高速一维比较的比较器的状态
高速二维位置比较	NV_HSCmp2DSetMode	设置高速二维比较模式
	NV_HSCmp2DGetMode	读取高速二维比较模式设置
	NV_HSCmp2DSetPSOPara	设置高速二维 PSO 输出模式参数
	NV_HSCmp2DGetPSOPara	读取高速二维 PSO 输出模式参数
	NV_HSCmp2DSetSource	设置高速二维比较位置源
	NV_HSCmp2DGetSource	读取高速二维比较器设置的位置源
	NV_HSCmp2DSetErrorBand	设置高速二维比较器误差带
	NV_HSCmp2DGetErrorBand	读取高速二维比较器误差带
	NV_HSCmp2DSetOuputMode	设置高速二维比较器输出模式
	NV_HSCmp2DGetOuputMode	读取高速二维比较器输出模式
	NV_HSCmp2DSetOutPulse	设置高速二维比较器的输出信号及波形
	NV_HSCmp2DGetOutPulse	读取高速二维比较器的输出信号及波形
	NV_HSCmp2DStartPulse	强制启动比较器的脉冲输出
	NV_HSCmp2DStopPulse	停止比较器的脉冲输出
	NV_HSCmp2DClearPoints	清除已添加的所有高速位置比较点

	NV_HSCmp2DAddPoints	设置高速二维比较器的队列先入先出比较点
	NV_HSCmp2DStatus	读取高速二维比较器的状态

4.11 EtherCAT 总线操作		
总线配置	NV_EcatReset	复位 EtherCAT 总线
	NV_EcatSetCycleTime	设置 EtherCAT 总线循环周期
	NV_EcatGetCycleTime	读取 EtherCAT 总线循环周期
	NV_EcatGetSlaveNum	获取 EtherCAT 从站总数
	NV_EcatGetLostHeartbeatNodes	获取心跳报文信息
	NV_EcatGetErrcode	获取总线错误码
	NV_EcatClrErrcode	清除总线错误码
总线轴	NV_AxisGetErrcode	获取总线轴错误码
	NV_AxisClrErrcode	清除总线轴的错误码
总线模块	NV_EcatGetSlaveIoNum	获取总线节点的 IO 数量
	NV_EcatGetDiBit	获取总线节点的单个输入状态
	NV_EcatGetDi	获取总线节点的单组输入状态
	NV_EcatSetDoBit	设置总线节点的单个输出电平
	NV_EcatGetDoBit	获取总线节点的单个输出电平
	NV_EcatSetDo	设置总线节点的一组输出电平
	NV_EcatGetDo	获取总线节点的一组输出电平
对象字典和 PDO 操作	NV_EcatSetSDO	设置总线节点的对象字典
	NV_EcatGetSDO	获取总线节点的对象字典
	NV_EcatSetPDO	设置总线节点的过程数据对象字典
	NV_EcatGetPDO	获取总线节点的过程数据对象字典

4.12 PWM 功能		
PWM 配置	NV_PWMSetEnable	设置 PWM 使能。
	NV_PWMGetEnable	读取 PWM 使能
PWM 输出	NV_PWMSetOutput	设置 PWM 输出
	NV_PWMGetOutput	获取 PWM 输出

4.13 PVT 运动		
PVT 数据表配置	NV_PvtAddTable	添加 PVT 列表数据
	NV_PtsAddTable	添加 PTS 列表数据
	NV_PvtsAddTable	添加 PVTS 列表数据
	NV_PttAddTable	添加 PTT 列表数据
PVT 运动指令	NV_PvtStart	同时启动多轴 PVT
	NV_PvtStatus	获取当前 PVT 状态

4.14 电子齿轮		
参数配置	NV_GearSetParam	设置电子齿轮的参数
	NV_GearGetParam	读取电子齿轮的参数
执行运动	NV_GearIn	进入电子齿轮模式
	NV_GearInPos	进入电子齿轮模式
	NV_GearOut	退出电子齿轮模式

4.15 电子凸轮		
Table 表配置及执行运动	NV_CAMAddTable	将凸轮数据添加到 Table 表
	NV_CAMGetTable	读取 Table 表数据
	NV_CAMGetTableSize	读取 Table 表中数据个数
	NV_CAM	电子凸轮运动（根据凸轮表运动）
	NV_CAMInAddTable	将凸轮主从轴数据添加到 Table 表
	NV_CAMIn	跟随凸轮表运动（根据凸轮表运动）
	NV_CAMOut	退出电子凸轮运动

4.16 龙门功能		
龙门配置及运动	NV_GantryIn	设置龙门的参数
	NV_GantryOut	退出龙门模式
	NV_GantryGetParam	设置龙门的参数
	NV_GantrySetProtect	设置龙门误差保护参数

	NV_GantryGetProtect	读取设置龙门误差保护参数

4 功能说明与实现

4.1 控制卡初始化

4.1.1 函数列表

分类	名称	功能
控制卡扫描	NV_ConnectPCIScan	扫描 PCI/PCIe 卡
	NV_ConnectNETScan	扫描网络卡
	NV_ConnectScan	扫描工控机内所有卡
控制卡初始化	NV_ConnectOpenPCI	打开单张 PCI/PCIe 卡
	NV_ConnectOpenPCIAll	打开所有 PCI/PCIe 卡
	NV_ConnectOpenNET	打开单张网络卡
	NV_ConnectIsOnline	控制卡连接是否在线
控制卡关闭	NV_ConnectClose	关闭单张控制卡
	NV_ConnectCloseAll	关闭所有控制卡
控制卡复位	NV_ConnectColdReset	控制卡冷复位
	NV_ConnectWarmReset	控制卡软复位
	NV_ConfigRecovery	恢复出厂设置
控制卡信息	NV_ConfigGetProductInfo	获取产品信息
	NV_ConfigGetDllVersion	获取动态库版本及发布时间
	NV_ConfigGetSoftVersion	获取固件版本及发布时间
	NV_ConfigGetHardVersion	获取硬件版本及发布时间
	NV_ConfigGetAxisNum	获取控制轴数量
	NV_ConfigGetEncoderNum	获取编码器数量
	NV_ConfigGetIoNum	获取通用 IO 数量
	NV_ConfigGetHSIoNum	获取高速 IO 数量
	NV_ConfigGetCmpNum	获取位置比较器数量
	NV_ConfigGetLatchNum	获取位置锁存器数量
	NV_ConfigGetInputNumList	获取 IO 输入数量
	NV_ConfigGetOutputNumList	获取 IO 输出数量
	NV_ConfigGetAxisNumList	获取各种类型的轴的数量

	NV_ConfigGetEncoderNumList	获取各种类型的编码器的数量
	NV_ConfigGetFunNumList	获取各种功能通道的数量

4.1.2 重点说明

（1）控制卡的连接

在操作控制卡之前，必须调用控制卡的初始化函数为卡分配资源；同样，当程序结束对控制卡的操作时，必须调用关闭函数释放接口板所占用的 PC 系统资源，使得所占资源可被其它设备使用。

控制卡与 PC 机通讯的方式有多种：有采用 PCI/PCIe 的方式，有采用网口的方式，在初始化控制卡（为控制卡建立通讯连接）前，可以通过扫描的方式，找出 PC 机连接了哪些卡。函数 ConnectPCIScan 用于扫描 PCI/PCIe 通讯类型的卡；ConnectNETScan 用于扫描网口通讯类型的卡；函数 ConnectScan 可以扫描上述两种类型的卡，但是扫描顺序是先找 PCI/PCIe 类型，让后再找网口类型，扫描出来的控制卡信息存放于 ScanResult_t 结构体信息中。ScanResult_t 结构体存放的控制卡信息有卡的类型、卡号（通过硬件拨码配置的卡号）、连接类型等信息。

扫描到卡以后，通过 Open 函数就可以连接到每一张卡。函数 ConnectOpenPCI 用于创建 PCI/PCIe 控制卡的连接；函数 ConnectOpenNET 用于创建网口卡的连接。运行过程中，通讯连接是否断线通过函数 ConnectIsOnline 判断。

（2）控制卡的复位

控制卡支持多种复位方式。

控制卡的冷复位：控制卡重新启动，当前所有的数据、状态都被清除。执行复位操作后，必须关闭控制卡，等待 15 秒方可执行初始化控制卡，否则会出错；出错后需要重新执行复位操作，再等待 15 秒后执行初始化控制卡。

控制卡的热复位：控制卡将总线复位并重新初始化、软件中的临时变量复位到初始值；但是不会改变用户已经配置的控制卡参数/功能，如 IO 的输出特性、高速锁存的配置信息等。执行热复位操作后，需要循环调用总线错误读取函数用来判断总线状态，如果总线状态返回正常，则可退出循环，热复位完成。

控制卡的总线复位：只重新初始化总线及总线相关变量，其他状态保持不变。

（3）控制卡的关闭

通过函数 NV_ConnectCloseAll 关闭控制卡，释放 PC 资源。

（4）控制卡信息获取

控制卡提供了多个获取信息函数，用户可以读取控制卡的硬件版本信息、软件版本信息、以及硬件资源等信息，如通过函数 NV_ConfigGetIoNum 可以获取 IO 数量，通过函数 NV_ConfigGetAxisNumList 获取轴数量，通过函数 NV_ConfigGetEncoderNumList 获取编码器数量等。

4.1.3 例程

例程：建立连接，断开连接，读取控制卡信息，请参考工程“IOSample”。

建立连接：

```
ERROR_CODE_TYPE ret = 0;
ret = NV_ConnectCloseAll();

ScanResult_t pScanResult = new ScanResult_t();
ret = NV_ConnectOpenAll(ref pScanResult);
if (ret != 0)
{
MessageBox.Show($"NV_ConnectOpenAll开卡失败，报错：{ret}");
}
else if (pScanResult.m_ConnectNum <= 0)
{
MessageBox.Show($"没有找到板卡，请确认正常插卡以及驱动成功安装！");
}
else
{
m_nConnectNo = pScanResult.m_ConnectInfo[0].m_ConnectNo;
```

```
comboBox_CONNECTID.Items.Clear();
for (int i = 0; i < pScanResult.m_ConnectNum; i++)
{
                comboBox_CONNECTID.Items.Add(pScanResult.m_ConnectInfo[
                  i].m_ConnectNo.ToString());
}
if (comboBox_CONNECTID.Items.Count > 0)
{
comboBox_CONNECTID.SelectedIndex = 0;
}
}
```

断开连接：

```
ERROR_CODE_TYPE ret = 0;
ret = NV_ConnectCloseAll();
if (ret != 0)
{
MessageBox.Show($"NV_ConnectCloseAll调用报错：{ret}");
return;
}
```

产品硬件信息、产品信息、版本信息读取：

```
      ERROR_CODE_TYPE ret = 0;
String pVersion = "";
String pVersionTime = "";
ret = NV_ConfigGetHardVersion(m_nConnectNo, ref pVersion, ref
pVersionTime);
String pProductName = "";
```

```
String pProductID = "";
ret = NV_ConfigGetProductInfo(m_nConnectNo, ref pProductName, ref pProductID);
String strVersion = "产品型号：";
strVersion += pProductName;
strVersion += "  硬件版本：";
strVersion += pVersion;
label_HardVersion.Text = strVersion;
ret = NV_ConfigGetSoftVersion(m_nConnectNo, ref pVersion, ref pVersionTime);
if (pVersionTime.Length > 0)
{
strVersion = "固件版本：";
strVersion += pVersion;
label_SoftVersion.Text = strVersion;
}
else
{
label_SoftVersion.Text = "";
}
```

4.2 系统配置

4.2.1 系统配置基本概念

在使用控制卡进行各种操作之前，需要对运动控制卡进行配置，使运动控制卡的状态和各种工作模式能够满足用户的要求，这个过程，叫做系统配置。

在控制卡配置工具 Motion 中，有对应的系统参数配置界面，用户可以利用这些界

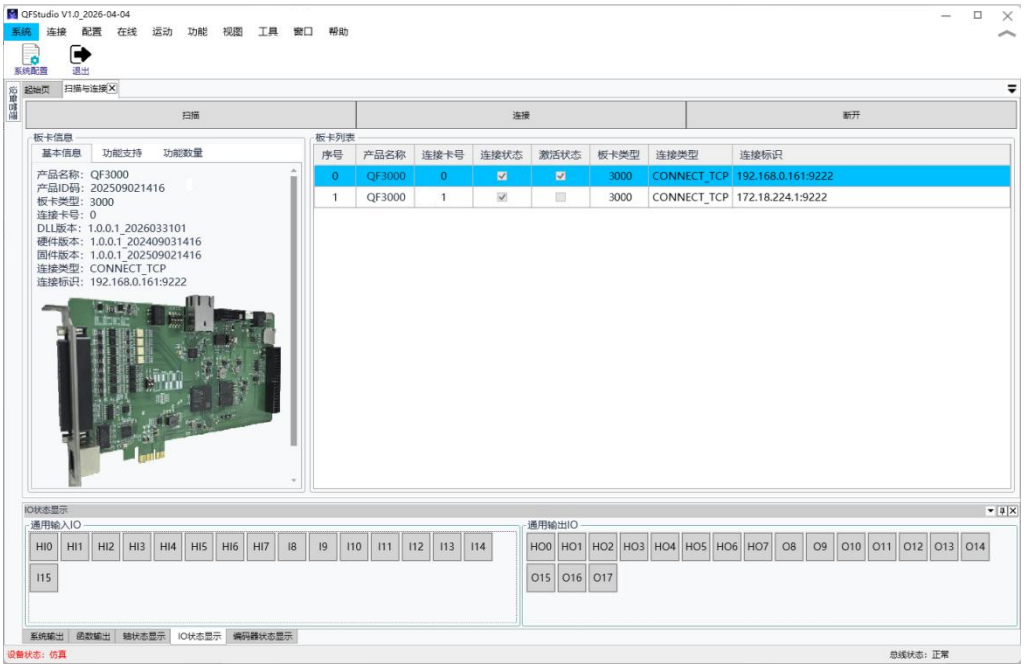
面对控制卡进行配置，配置完成之后，生成相应的操作或者配置文件，将配置文件下载到运动控制卡，即可完成控制卡的配置工作。除了配置文件，用户也可以直接调用配置的指令完成运动控制卡的配置。

4.2.2 配置工具 Motion

使用控制卡配套的 Motion 软件，能够方便地对系统进行配置。

该软件为绿色版，不需要安装。直接点击软件包中的执行文件，即可打卡软件。

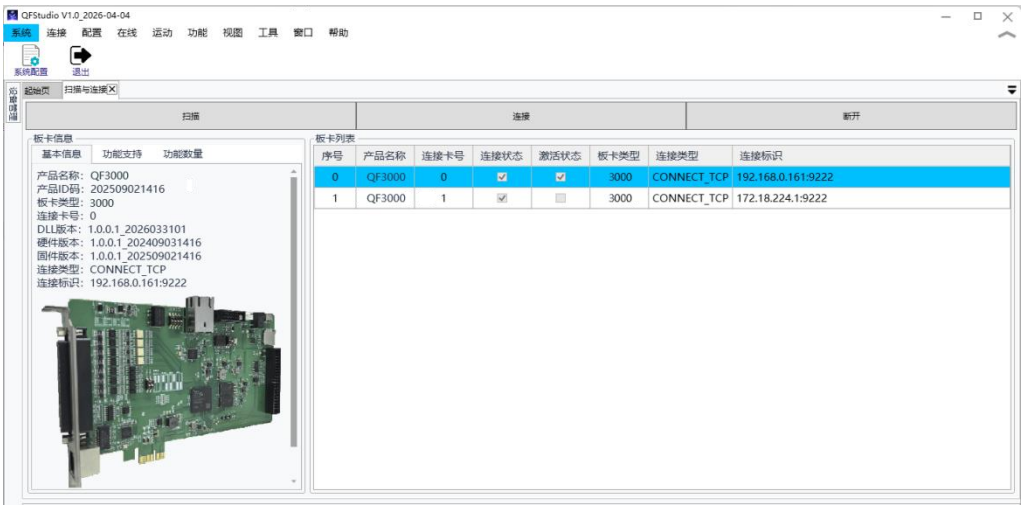
启动软件以后，主界面如下图所示。



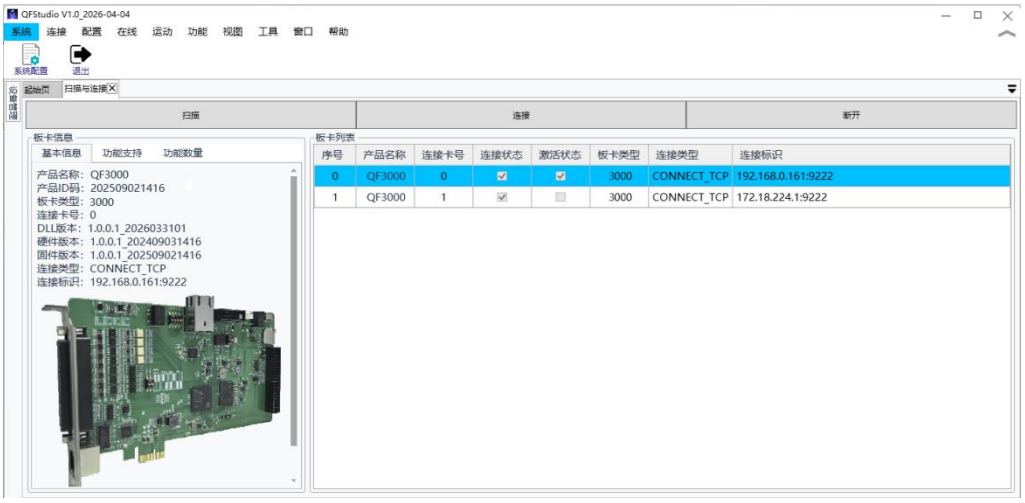
软件启动时，会自动识别电脑中已经插入的卡，如上图所示，识别到了一张卡；

如果手动执行扫描查看电脑中的卡，点击菜单“连接”，在快捷工具中点击“扫描与连接”。

如果电脑上有多张卡，那么将会显示按照卡号，显示多张卡，如下图，软件识别到了两张卡。



在右侧列表中，双击已经扫描出的卡，软件则与卡建立连接（在“板卡列表”中，“连接状态”和“激活状态”会被勾选），如下图所示：

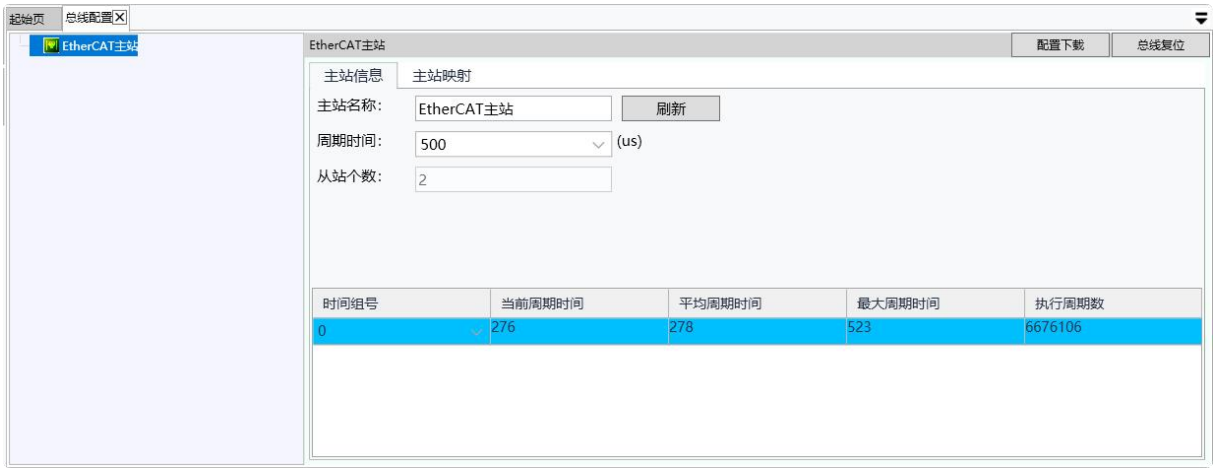


在左侧的“板卡信息”中，将会显示卡的基本信息、支持的功能及功能数量。

4.2.3 EtherCAT 配置

扫描从站

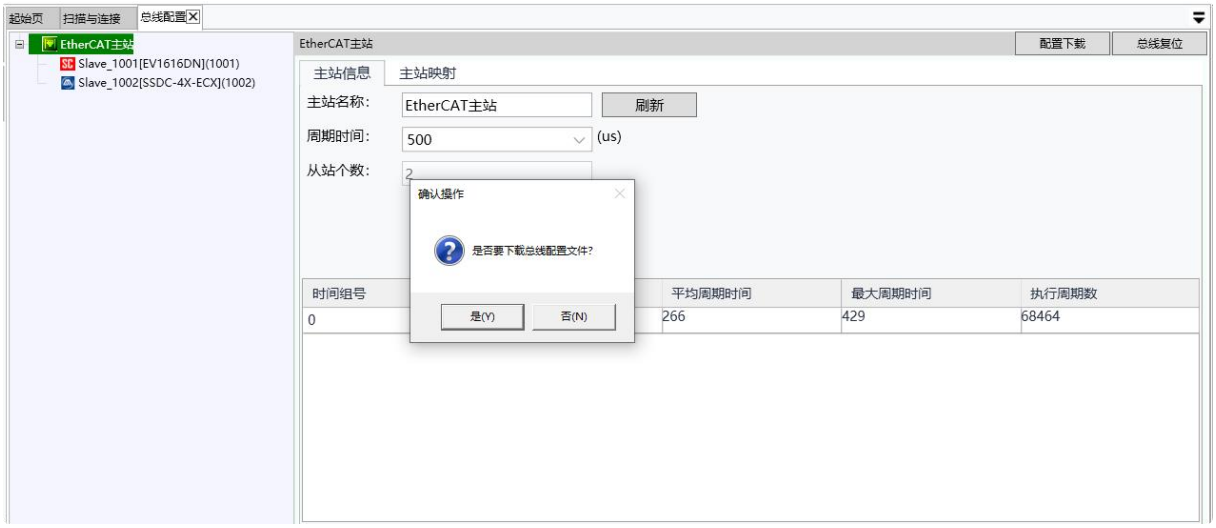
在菜单“配置”中，选择快捷按键“总线配置”，将会弹出“总线配置”界面，如下图所示：



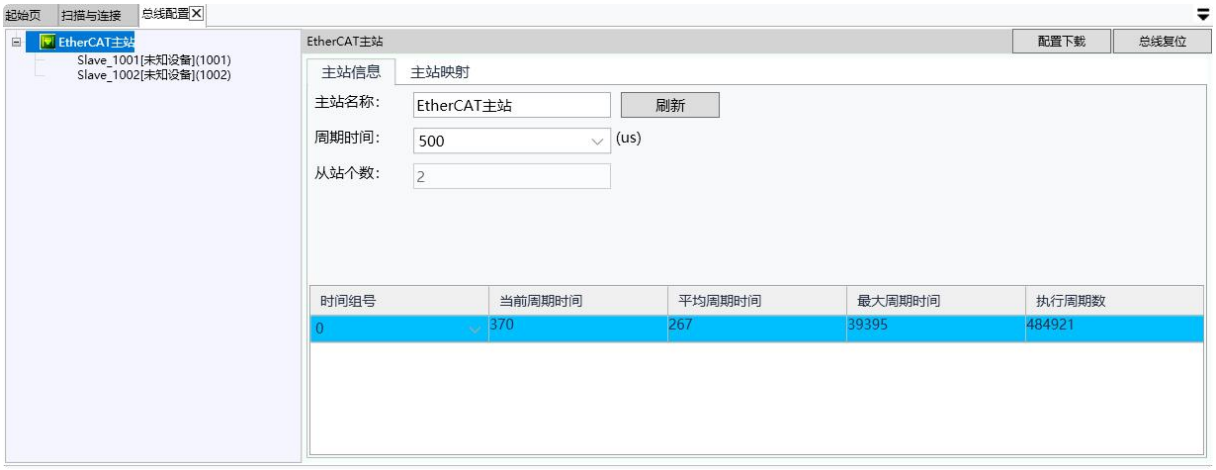
在左侧目录树上，右键点击“EtherCAT 总线”，弹出对话框，选择“扫描设备”，如下图所示：

在上图所示的弹出框中，用户也可以手动添加从站、删除从站，具体操作请查阅“手动添加/移除设备”章节内容。

控制卡将自动扫描出总线上连接的设备，并提示“是否下载总线配置文件”，请点击“是”，如下图所示，总线上连接有两个设备：

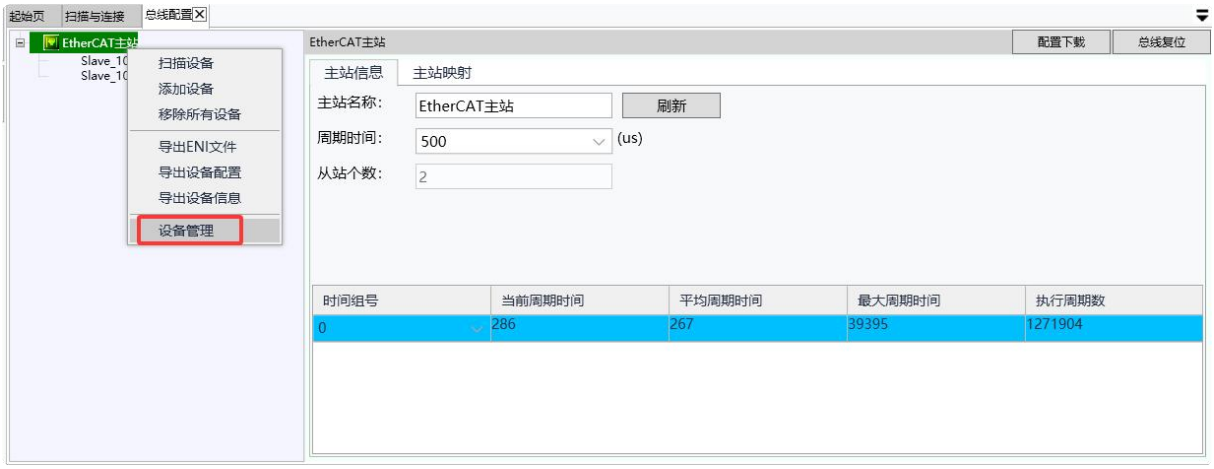


如果 Motion 软件中，没有添加从站设备的设备描述文件（XML 文件），那么从站设备将处于“未知设备状态”，如下图所示，这种情况下需要再 Motion 中添加从站设备的设备描述文件，具体操作请查阅 “设备管理”章节内容。



设备管理

在左侧目录树上，右键点击“EtherCAT 总线”，弹出对话框，选择“设备管理”，如下图所示：



点击“设备管理”选项后，将弹出“设备管理”对话框，界面如下图所示：

设备列表：显示 Motion 软件中已经添加的设备

安装设备：添加需要导入到 Motion 软件中的设备文件。

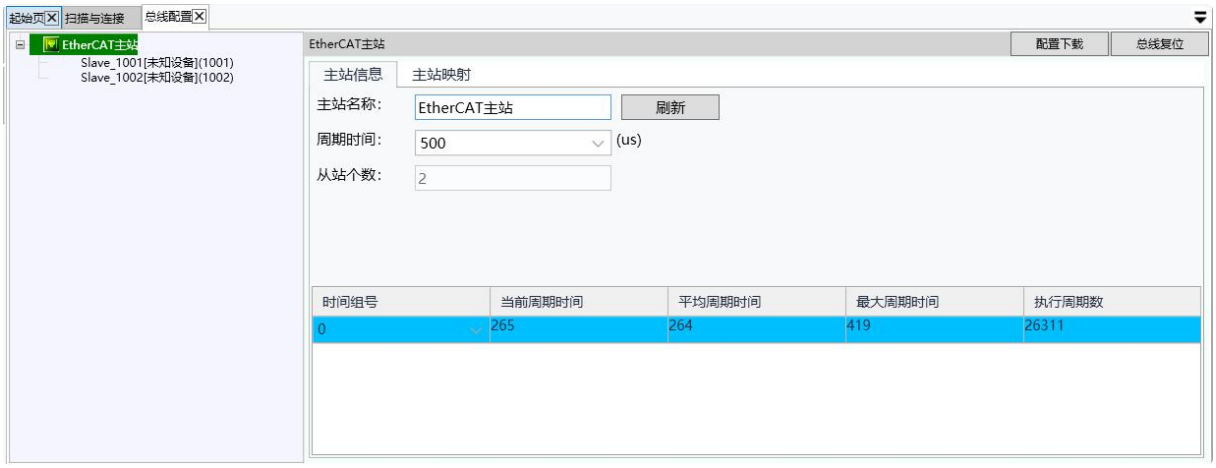
配置信息：手动配置从站设备的轴数、数字 IO 的 PDO、模拟 IO 的 PDO。

保存配置：将配置信息保存到 Motion 软件中。

现以添加一个 IO 模块和一个步进驱动为例，描述添加从站设备文件的步骤。

前提条件：现在总线上连接有两个从站设备，一个为 16IN16OUT 的数字 IO，一个为一拖四（一个驱动带四个电机）的步进闭环驱动，通过总线扫描后，不能识别出这两

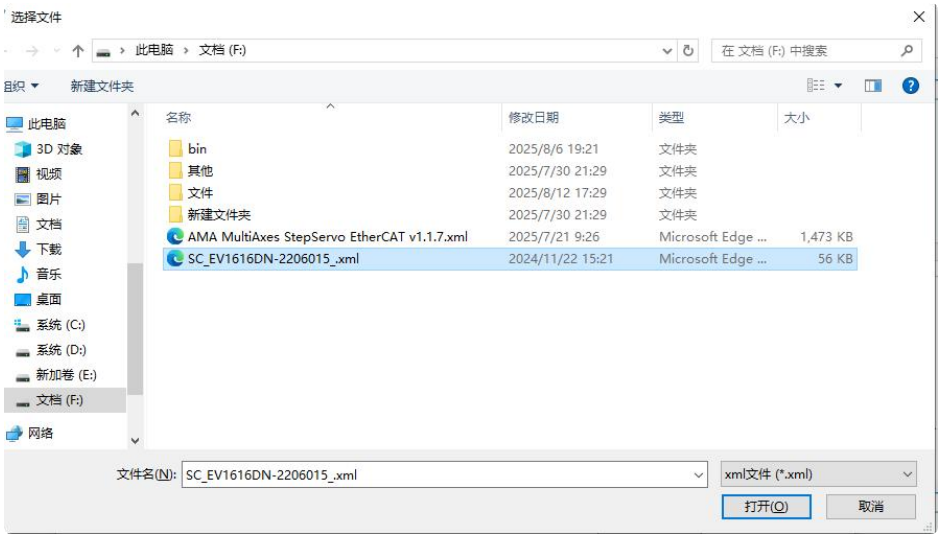
设备，如下图所示，因此需要往 Motion 中添加对应的设备描述文件。



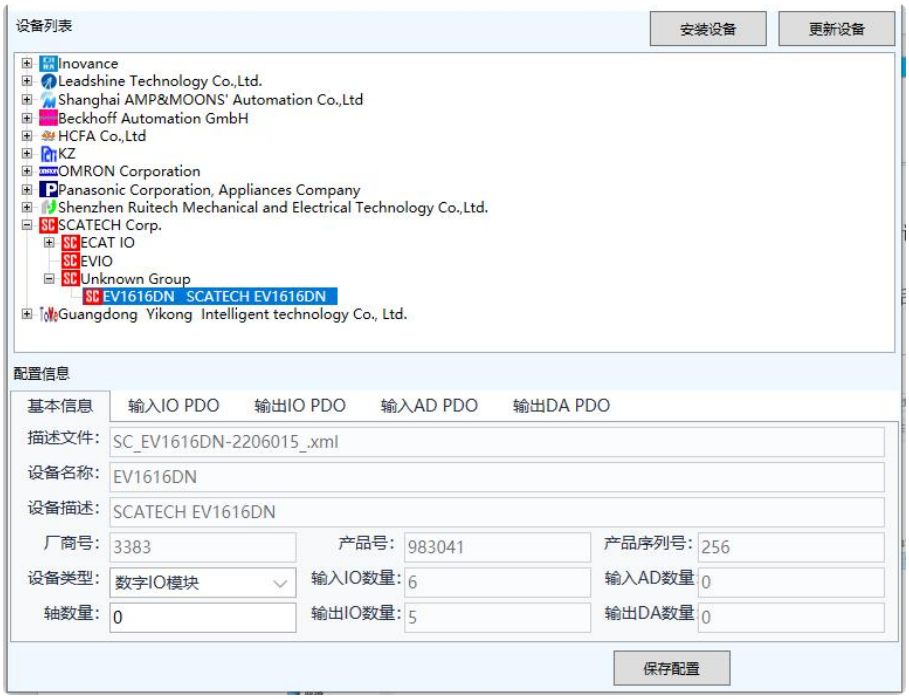
其操作步骤如下：

（1）进入设备管理界面。在左侧目录树上，右键点击“EtherCAT 总线”，弹出对话框，选择“设备管理”，弹出“设备管理”界面。

（2） 添加 IO 模块的设备描述文件。点击“安装设备”，在弹出的对话框中选择 IO 模块的设备文件，然后点击“打开”按钮，等待系统将文件加载到 Motion 中。



添加完成后，在设备列表中，可以看到新添加的 IO 设备，如下图：

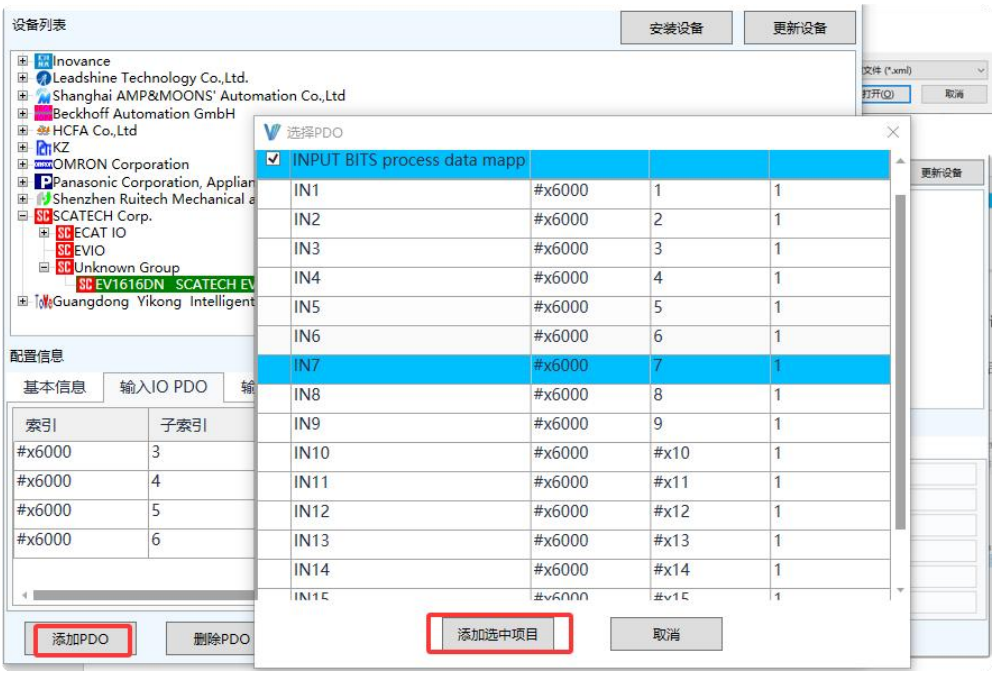


（3）配置 IO 从站的“配置信息”。

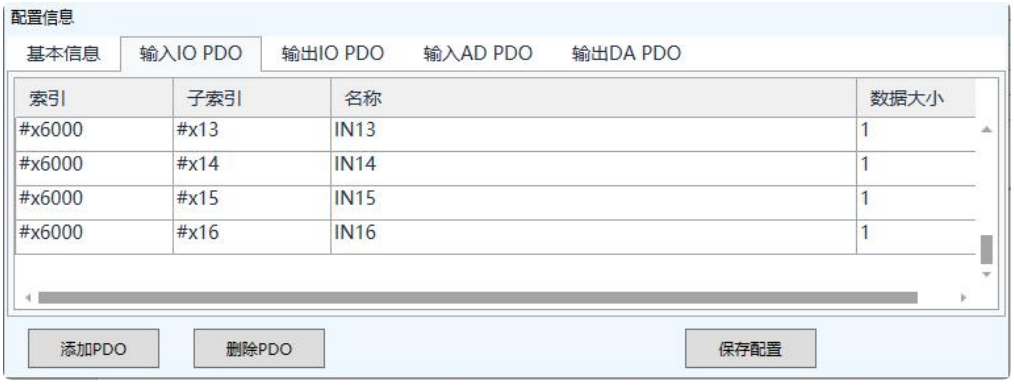
在基本信息的“设备类型”中，选择“数字 IO 模块”。

在输入 PDO 中，手动添加 PDO 的选项。

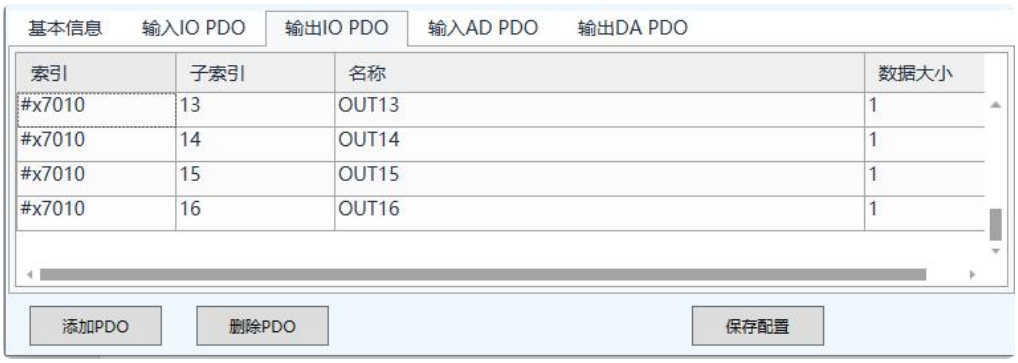
在“输入 IOPDO”界面中 -> 点击“添加 PDO” -> 在弹出的界面中，选择列表中的行 -> 点击“添加选中项目”。操作界面如下：



重复上述步骤，将需要在输入 PDO 中传输的项目都添加完毕，一共有 16 输入点，添加完后如下图所示：



在输出 PDO 中，手动添加输出 PDO 的选项，操作过程如输入 PDO；一共有 16 输出点，添加完后如下图所示：

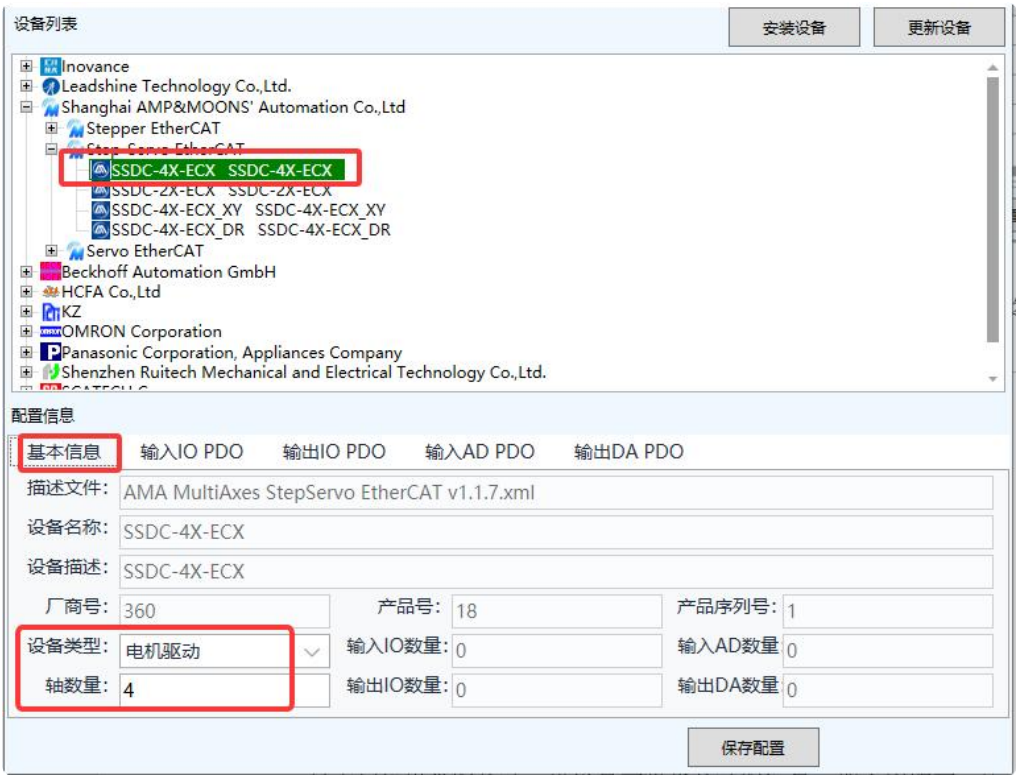


配置完成后，点击“保存配置”。

（4）添加驱动的设备描述文件。在“设备管理”中点击“安装设备”，选择文件，导入到中 Motion 软件中。

（5）配置驱动的“配置信息”。

在“设备列表”中选择连接的驱动设备；在“基本信息”的“设备类型”中，选择“电机驱动”，轴的数量，输入 4（驱动器 SSDC-4X-ECX 可以连接 4 个电机，因此轴数量填写 4；如果驱动器为 SSDC-2X-ECX，轴数量填写 2），显示界面如下图：



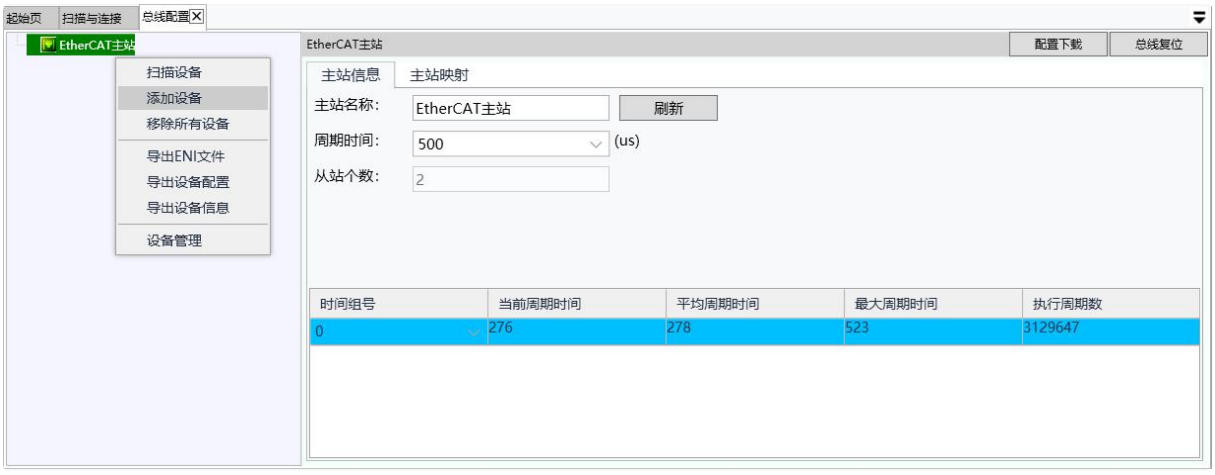
配置完成后，点击“保存配置”。

（6）关掉“设备管理”界面后，重新“扫描设备”，即可正确识别到设备。

手动添加/移除设备

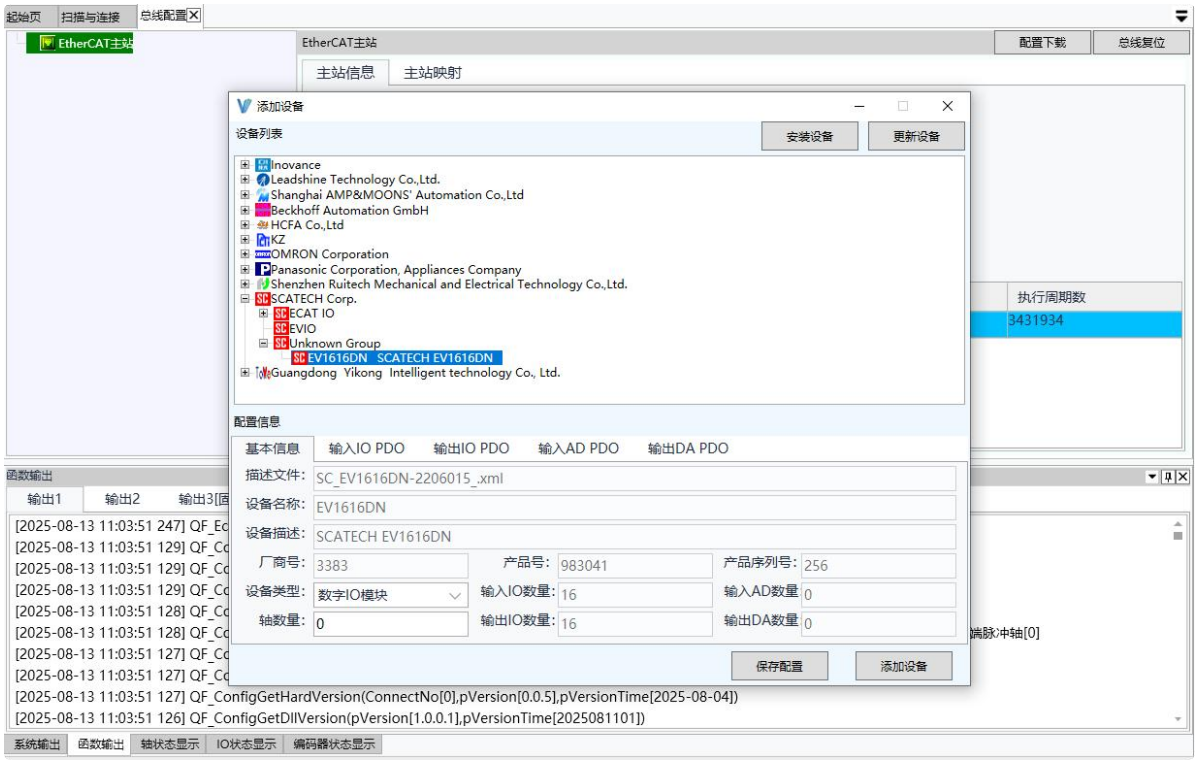
（1）手动添加设备

在左侧目录树上，右键点击“EtherCAT 总线”，弹出对话框，选择“添加设备”，如下图所示：



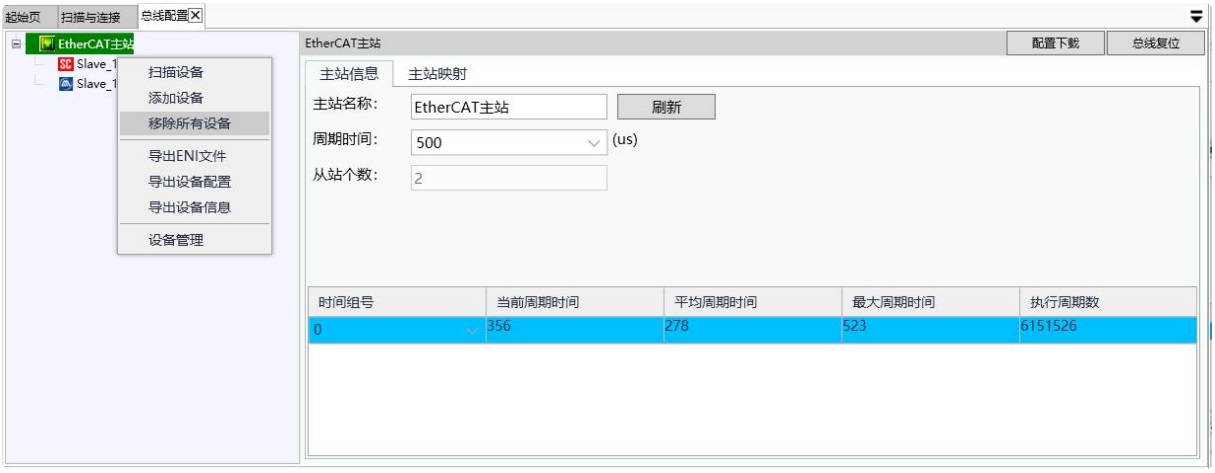
在弹出的“添加设备”对话框中，选择连接的从站设备，如下图，然后点击“添加

设备”。

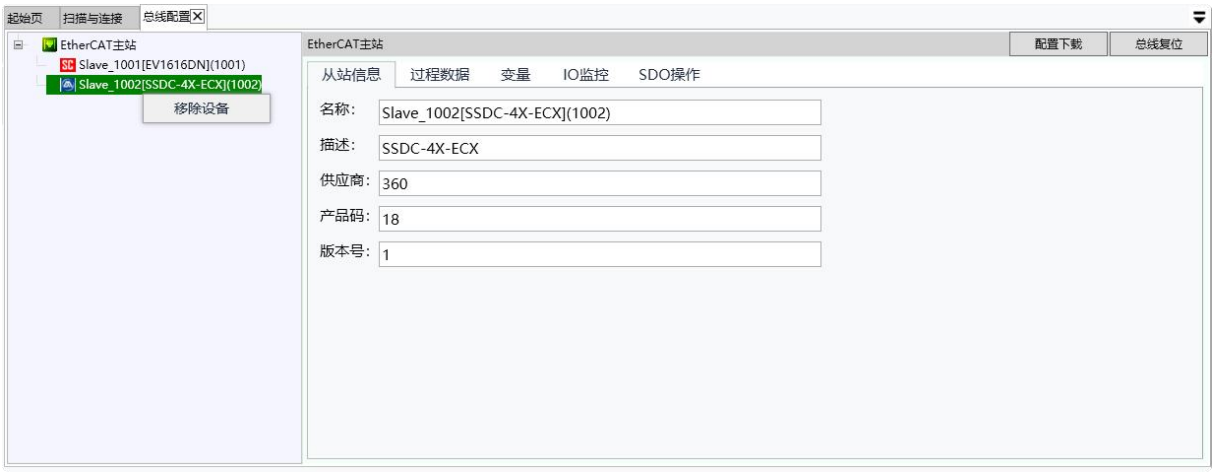


（2）手动移除设备

移除所有设备：在左侧目录树上，右键点击“EtherCAT 总线”，弹出对话框，选择“移除所有设备”，即可移除总线下所有设备，如下图所示：

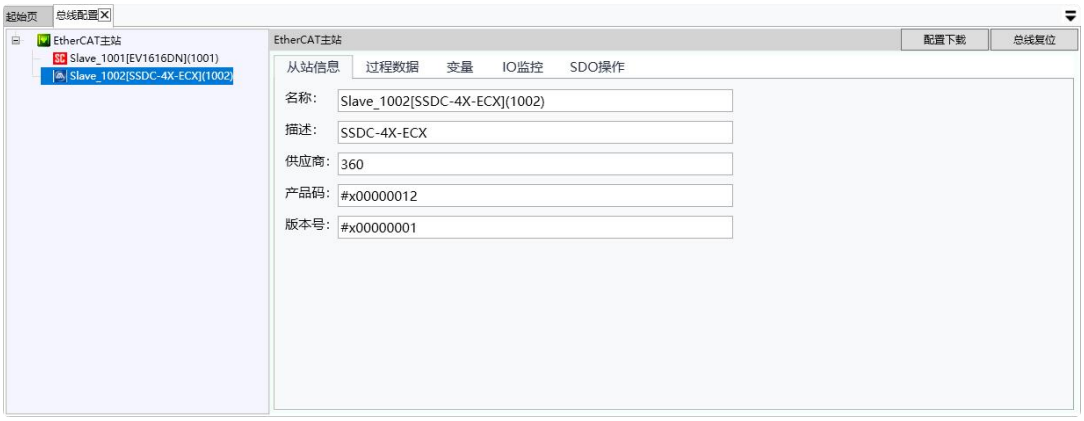


移除单个设备：在左侧目录树上，右键点击需要移除的设备，在弹出的对话框中，点击“移除设备”即可，如下图所示：

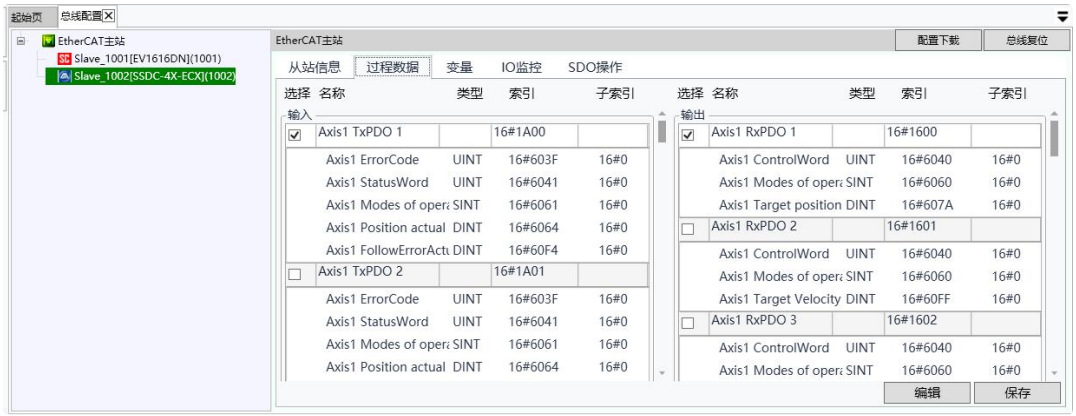


从站配置信息

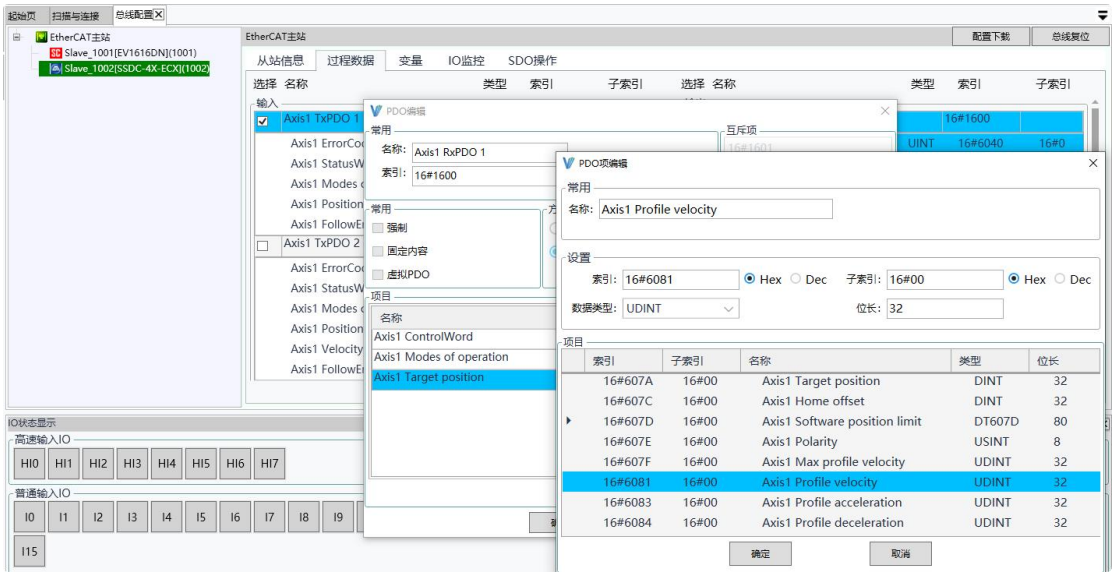
点击扫描出来的从站，可以显示每个从站的信息，如下图所示：



如果需要配置从站的 PDO，则需要点击“过程数据”菜单，如下图所示：

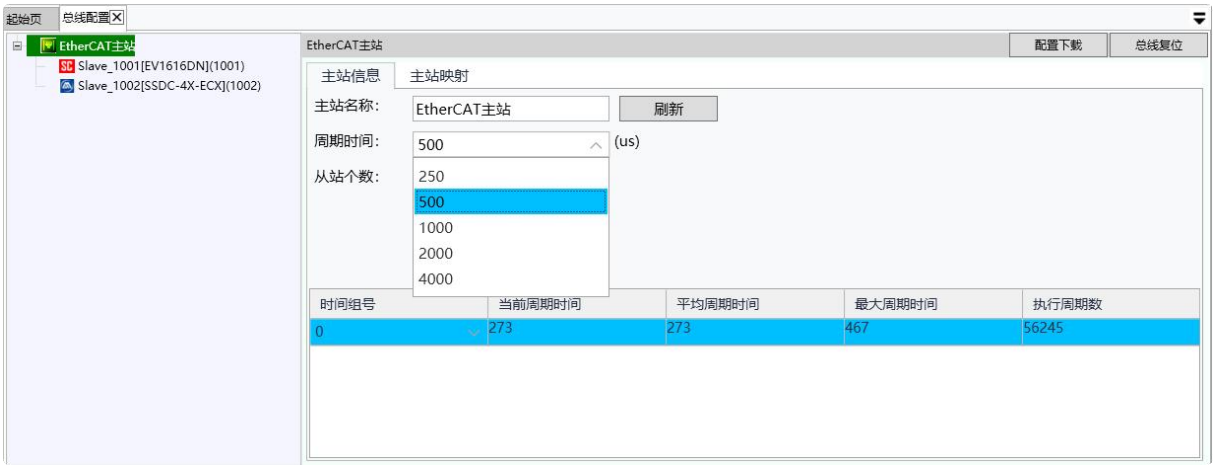


在上述界面中，用户选择需要配置的 PDO，点击“编辑”按键，在弹出的对话框中 PDO 参数进行配置。点击“添加”按钮，在弹出的对话框中，选择需要添加的对象字典，如下图所示；点击“删除”按钮，直接删除选择的对象参数。



主站配置信息

对于主站参数，用户主要是配置总线的“周期时间”，系统默认的时间为 500us，如下图。用户可以根据总线负载的大小，选择合适的周期时间。



EtherCAT 主站配置界面，主要可以进行如下操作

（1）配置总线周期。

（2）实时显示总线执行的周期状态信息。

（3）查看主站的映射信息，轴映射、映射。

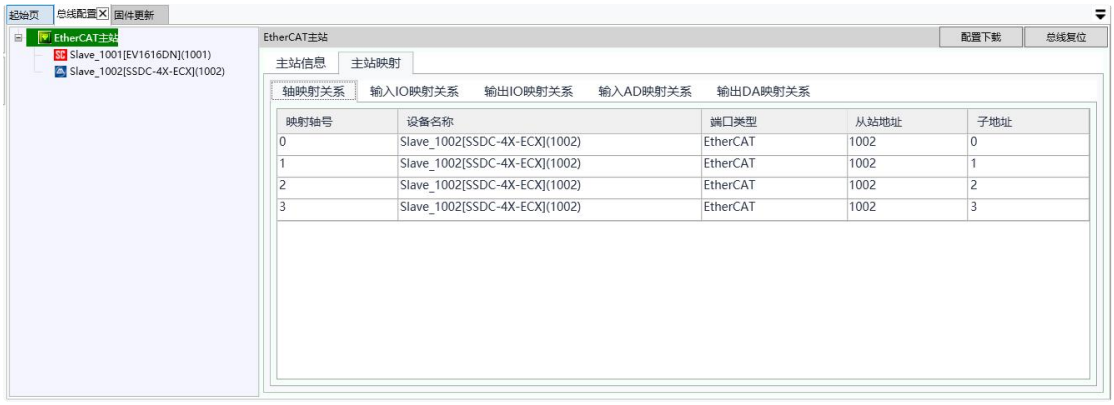
（4）将总线配置文件下载和执行总线复位。

主站映射

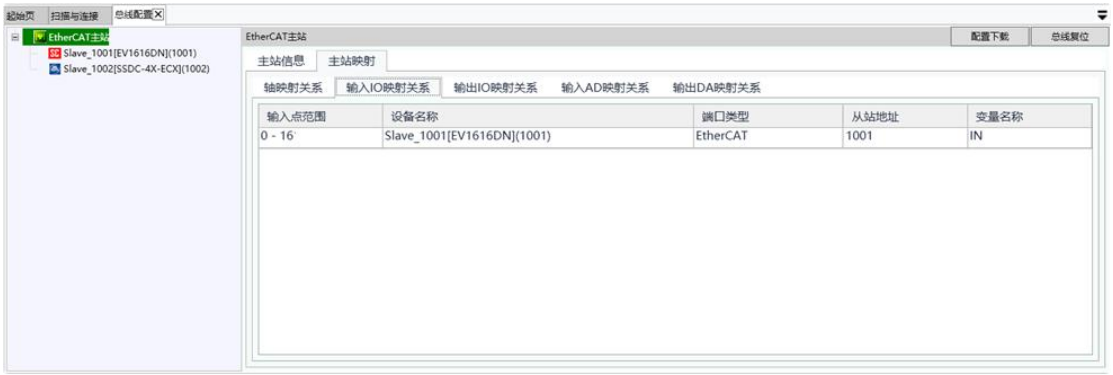
用于显示值各从站映射的映射关系，软件自动生成。总线扫描到从站设备后，会自动为从站设备分配访问资源。

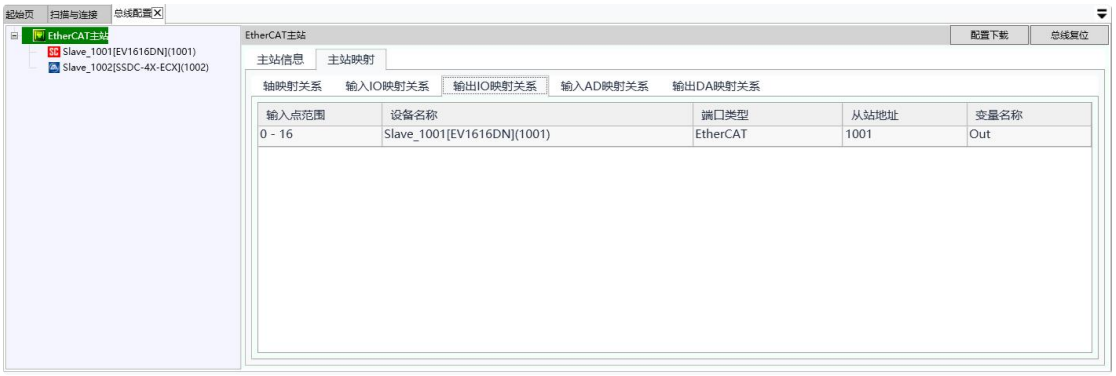
如果从站是 IO 设备，自动为每个模块的 IO 点分配 IO 编号；如果从站是轴设备，自动为每个轴分配轴编号。用户在编写应用程序中，直接操作分配好的 IO 编号和轴编号即可实现对扩展的 IO 和轴进行操作。

下图显示的为映射的轴号，如需要将第 2 个轴执行相对运动，从“映射轴号”中可以得知该轴的编号为 1（轴的编号从 0 号开始），那么执行相对运动函数 NV_PmoveRelative 的轴号参数 m_nAxisNo 值为 1。



下图分别显示的是映射的输入点和输出点，用户在应用程序中可以对每个从站的 IO 点根据编号进行操作。如需要将第 2 个 OUT 口输出状态设置为 1，那么设置输出的函数 NV_AxisIoSetDo 的从站节点号参数 SlaveId 值为 1001，IO 序号 DoIndex 值为 2，输出电平 Value 为 1。





配置文件下载及总线复位

经过上述的步骤后，总线配置已经完成，通过“配置下载”按钮，将配置的文件下载到控制卡中。该文件下载后，自动保存在卡中，下次启动会自动加载该文件。

如果总线连接发生变化，需要重新对总线进行配置，并将新配置后的文件下载到卡中。

新的配置下载后，需要执行“总线复位”，这样配置才会立即生效。

4.2.4 轴配置

轴映射配置

控制轴号：控制卡已经识别到的轴列表。

映射类型：配置当前轴号对应的类型。可以将其配置为虚拟轴、差分脉冲轴、EtherCAT 总线轴、单端脉冲轴。

映射编号：常规情况下系统自动生成，不需要修改。

映射子编号：有些驱动可以带多个轴，该参数标识一个驱动里面的多个轴的轴列表。

设置参数：将配置的参数写入控制卡中。

读取参数：将控制卡的参数读取显示到 Motion 中。

轴参数配置

在轴参数配置界面，可以配置轴的所有参数，配置界面如下图所示：

轴基本参数：脉冲输出类型（脉冲轴）、脉冲当量；

轴专用 IO：输入通道、有效电平、是否启用；

运动停止参数：软件限位停止参数、编码器超差停止参数、异常减速停止参数。

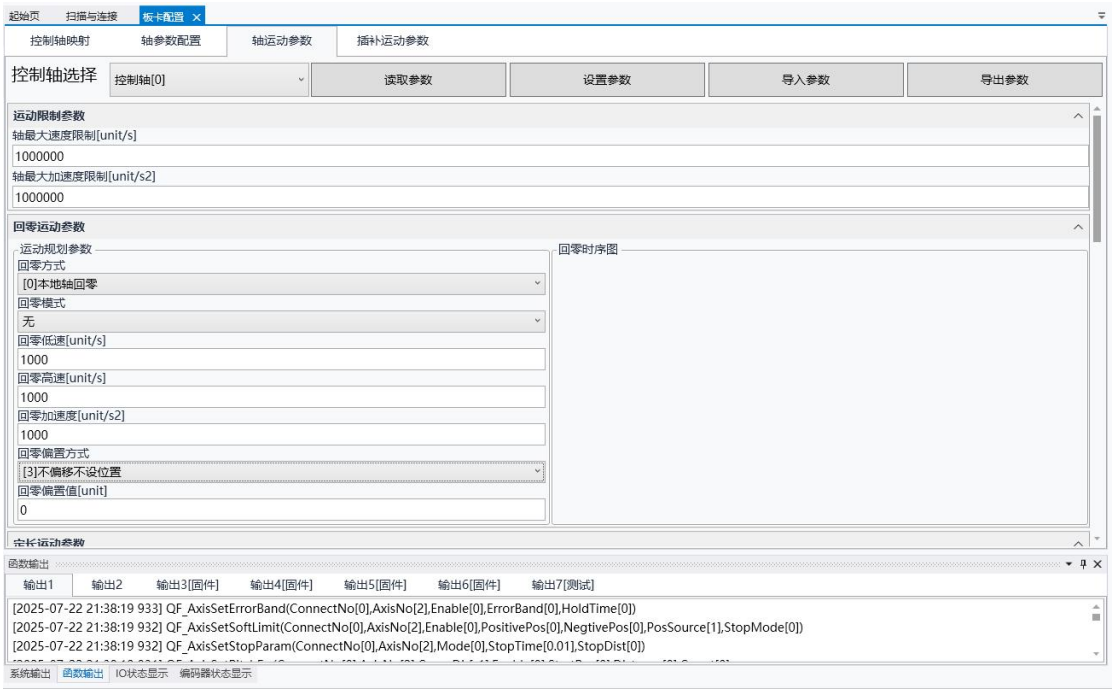
运动补偿：反向间隙补偿、丝杆螺距补偿；

参数配置完后，通过“设置参数”按钮，将参数写入到控制卡中；在软件的最下方“函数输出”信息框中，将把所有设置函数列出，以供用户查阅，如下图所示。

轴参数不会存储到卡中，卡重启后，需要重新配置这些信息。

轴运动参数

点击“轴运动参数”界面，进入配置轴的运动参数界面，其配置界面如下图所示：



运动限制参数：设置轴运动的最大速度限制、最大加速度限制。

回零运动参数：设置回零需要配置的参数，如回零方式、回零模式、回零的低速/高速/加速度、回零偏置方式等参数。

定长运动：配置定长运动时的起始速度、速度、停止速度、加减速方式等参数。

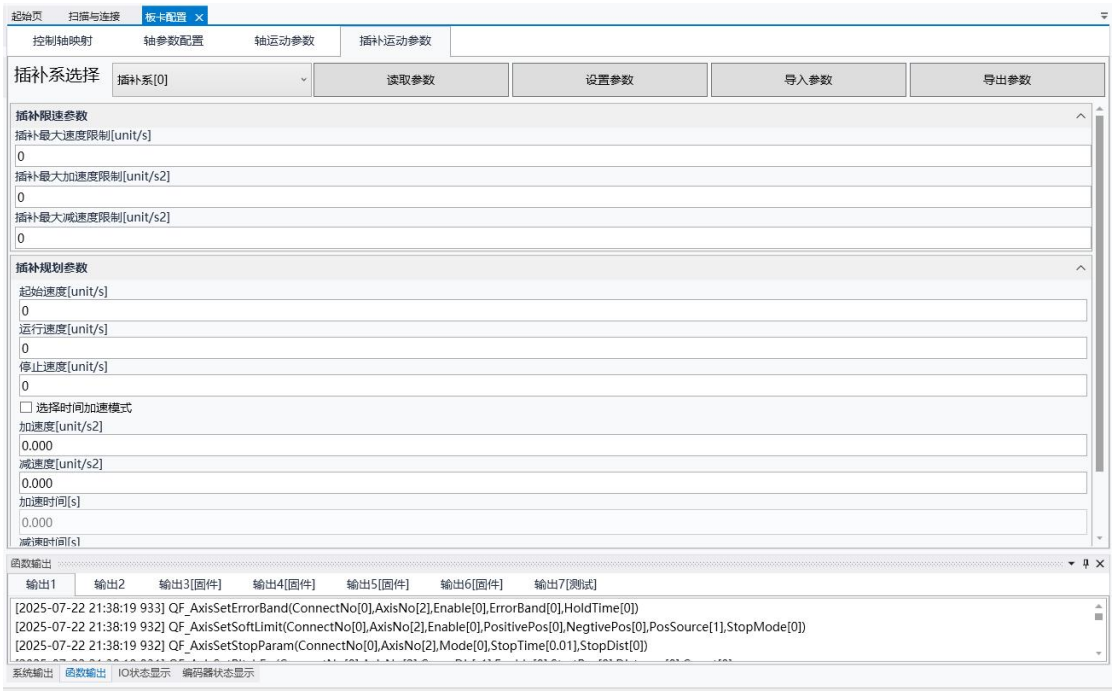
定速运动：类似配置定长运动的运动参数。

Jog 运动：类似配置定长运动的运动参数。

参数配置完毕后，通过“设置参数”按钮写入到卡中。这些参数不会储存到卡上，卡重启后，需要重新配置。

插补运动参数

点击“插补运动参数”界面，进入插补运动参数，其配置界面如下图所示：



插补限速参数：设置插补的最大速度、加速度、减速度限制。

插补规划参数：设置插补运动的起始速度、运动速度停止速度、加减速方式等参数。

参数配置完毕后，通过“设置参数”按钮写入到卡中。这些参数不会储存到卡上，卡重启后，需要重新配置。

4.2.5 功能测试

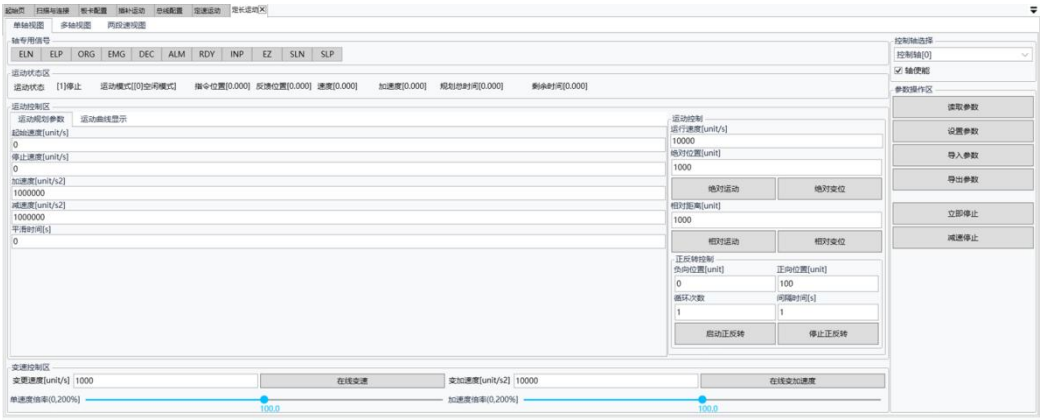
通过 Motion 软件，用户可以通过配置的方式，完成控制卡的各项功能。

支持的运动功能有回零运动、定长运动、定速运动等，其快捷操作按键如下图所示：

支持的其他功能有 IO 操作、PWM 操作、锁存操作等，其快捷操作按键如下图所示：

下面将以“定长运动”为例，描述其基本操作：

第一步：进入到如下图所示的“定长运动”操作界面：



第二步：配置轴参数

配置运动规划参数

运行速度：10000

加速度：10000

减速度：10000

参数操作区：

选择轴：控制轴 0

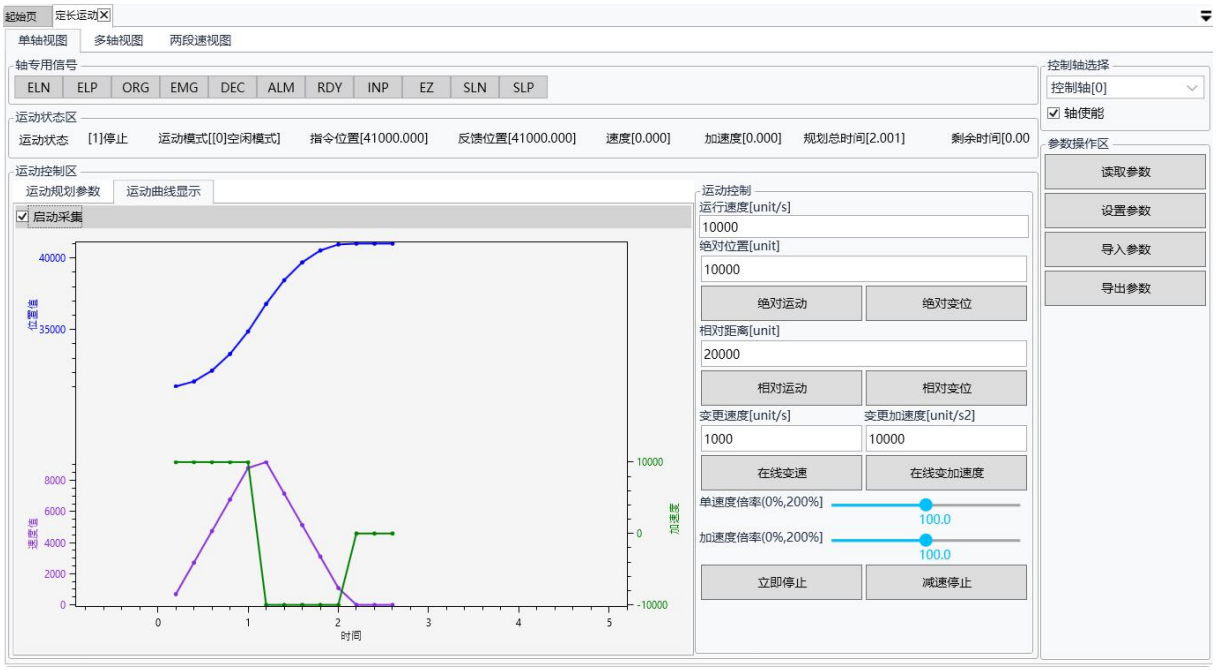
轴使能：勾选

第三步：启动运动

设置相对距离：10000；

点击“相对运动”按钮，启动运动。

在左侧的“运动曲线显示”中，绘制出轴的位置和速度曲线。



4.3 安全机制

本节介绍控制卡提供给用户的所有可用安全机制，包括：正负硬限位、正负软限位、驱动报警、跟随误差、圆形运动安全区等触发或者越界时采取的减速停止或者紧急停止等安全保护机制。

4.3.1 参数与状态配置

函数列表

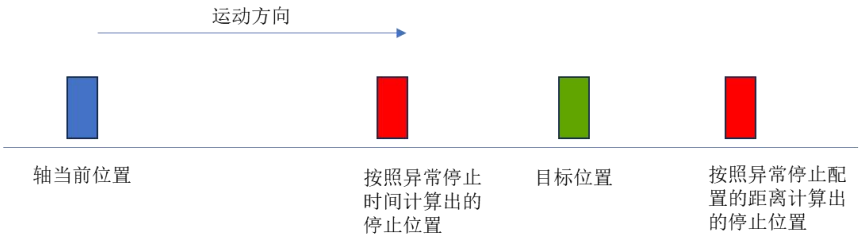
名称	功能
NV_AxisSetStopParam	设置轴的异常停止参数
NV_AxisGetStopParam	获取轴的异常停止参数
NV_AxisGetStopReason	获取轴运动停止的原因
NV_AxisClrStopReason	清除轴运动停止原因的记录

重点说明

（1）系统在正常运行中，有可能发生异常，如软限位减速停止、IO 触发减速停止、超出设置的跟随误差，那么轴就会执行异常停止。通过函数 NV_AxisSetStopParam 配置出现异常情况下，轴选择异常停止的最短距离。

参数 StopTime，用于配置出现异常后，需要停止的时间（系统内部自动换算停止的距离）；参数 StopDist，用于配置出现异常后，需要停止的距离。

因此，系统出现异常后，系统内部会计算出三个位置：设置的目标位置、按照异常停止时间计算出的停止位置、按照异常停止距离计算出的停止位置，在这三个位置中，系统自动选择距离当前位置最近的作为停止点，如下图，系统将会选择“按照停止时间计算出的停止位置”。



（2）系统异常停止后，通过函数 NV_AxisGetStopReason 可以读取停止原因。

4.3.2 软限位停止功能

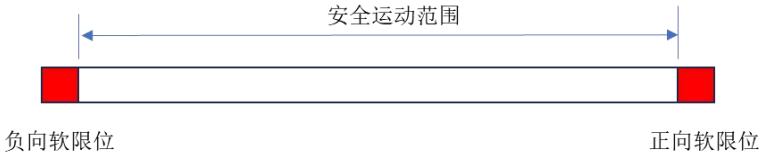
函数列表

名称	功能
NV_AxisSetSoftLimit	设置轴的软限位功能
NV_AxisGetSoftLimit	获取轴的软限位功能配置
NV_AxisSetCircleLimit	设置圆弧限位保护功能
NV_AxisGetCircleLimit	获取圆弧限位保护功能
NV_AxisSetErrorBand	设置超差停止功能
NV_AxisGetErrorBand	获取超差停止功能状态

重点说明

（1）软限位

控制卡能够通过设置软限位来限制各轴的运动范围，如下图所示：



限位存在情况下轴的安全运动范围

工作台规划位置超越软限位时，控制卡按照用户设置的紧急停止减速度紧急停止工作台的运动。限位触发以后，运动控制卡禁止向触发限位方向运动，同时该轴相应的限位触发方向状态置 1。轴运动到安全运动范围内后，需要调用指令清除轴的状态，才能使轴状态恢复到正常状态。

软限位的使用：

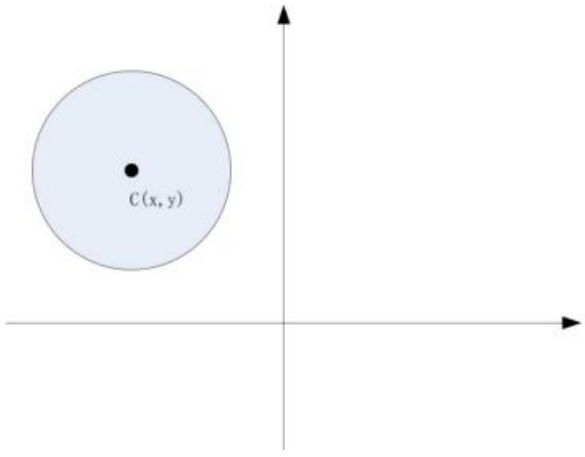
（a）回原点以后设置软限位。

（b）正向软限位必须大于负向软限位。

（c）软限位和限位开关可以同时使用，当软限位触发时也会置起限位触发标志。

（2）圆形区域限位，即运动仅在用户设定的圆形范围内进行，超出这个范围，运动会依据设定的减速模式停止，且禁止任何运动操作。仅支持平面圆形限位，不支持空间圆形限位。

圆形限位的使用：首先，用户需要绑定两个轴进而确定该功能所处的坐标系；然后，根据用户设置的圆心坐标及半径，确定该区域在坐标系中的位置；最后，运动过程中，对两轴的每一个位置 xi、yi 都判定与圆心的距离，若超过半径 R，则触发减速停或急停。使用注意事项：轴在运动中，不可设置圆形限位参数；



圆形限位区域

（3）超差停止：对于伺服控制系统而言，在某些异常情况下，例如电机故障、编码器 A、B 相信号接反或断线、机械摩擦太大或者机械故障造成电机堵转等，电机的实际位置可能与规划位置差距很大。为了及时检测这些情况，增强系统的安全性并延长设备使用寿命，控制卡提供跟随误差超限自动停止的安全保护机制。

控制卡在每个周期内都检查控制轴的实际位置与规划位置的误差是否超越所设定的跟随误差极限。如果位置误差超越所设定的跟随误差极限，控制卡自动紧急停止该轴的运动，同时该轴跟随误差越限标志置 1。

4.3.3 硬触发停止和命令停止功能

函数列表

名称	功能
NV_EmgStop	紧急停止所有轴
NV_AxisStop	指定轴停止运动

重点说明

（1）紧急停止

当系统发生比较严重的异常，通过函数 NV_EmgStop 可以紧急停止所有轴的运动，在任何情况下，调用该函数，轴运动都会做紧急停止处理。

（2）减速停止

停止指令 NV_AxisStop，只适应于单轴运动、同步运动、PVT 运动，不适用于多轴插补运动。

4.4 轴参数配置及状态读取

当用户连接好一整套系统后（包括运动控制卡，驱动器，电机），如何对轴进行配置以及查看这套系统的运动状态。控制卡可以帮助用户在应用程序中对轴进行配置、查看驱动器报警、当前运动位置、运动速度和加速度等一系列状态参数。本章将介绍用户如何配置轴及可以检测到哪些状态。

4.4.1 轴基础参数配置

函数列表

名称	功能
NV_AxisSetEnable	设置轴使能
NV_AxisGetEnable	获取轴使能状态
NV_AxisSetBusMode	设置总线轴控制模式
NV_AxisGetBusMode	获取总线轴控制模式
NV_AxisSetStepPulse	设置轴脉冲类型
NV_AxisGetStepPulse	获取轴脉冲类型
NV_AxisSetUnit	设置轴的当量
NV_AxisGetUnit	获取轴的当量
NV_AxisSetMaxVel	设置轴的最大速度
NV_AxisGetMaxVel	获取轴的最大速度
NV_AxisSetMaxAcc	设置轴的最大加速度
NV_AxisGetMaxAcc	获取轴的最大加速度
NV_AxisGetErrorStatus	读取轴的错误状态值

重点说明

（1）轴使能

轴在运动前，需要调用使能指令 NV_AxisSetEnable，打开指定控制轴所连电机的伺服使能信号，使指定控制轴进入控制状态。

（2）轴类型

在运动控制系统中，将运动控制的对象称为“轴”，运动控制系统中的一个电机控制的运动平台称为一个运动轴，每个运动轴只有一个自由度，可以做直线运动或旋转运动。在控制卡中，轴的分类如下：

轴类型编号	轴类型	说明
0	虚拟轴	控制卡内的虚拟轴，不与实际驱动器关联。可以用于同步控制的虚拟主轴，机械手算法中的笛卡尔坐标轴，仿真软件中轴。
1	差分脉冲轴(5V)	控制卡外部实际连接的伺服或者步进轴。

2	EtherCAT 总线轴	通过 EtherCAT 总线扩展的轴
3	单端脉冲轴(24V)	通过 24V 高速输出端口，采用单端方式控制的轴

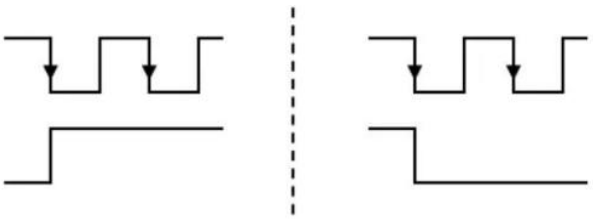
（3）脉冲轴

控制器卡具有多个脉冲/方向输出端口，并且部分高速输出端口可以复用为单端的脉冲输出口，单端脉冲输出端口接线方式，请参考对应产品的用户手册。

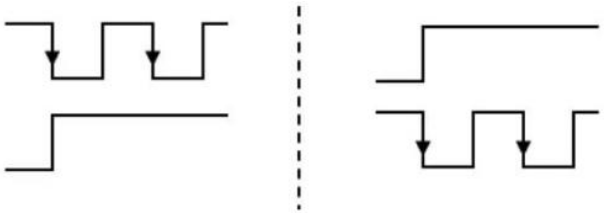
对于差分脉冲/方向输出端口，支持七种类型输出：脉冲方向类型(PUL/DIR)、双脉冲类型(CW/CCW)、正交脉冲类型(A/B 相)。

输出脉冲类型 OUTMODE	正方向脉冲		负方向脉冲	
	PULSE 输出端	DIR 输出端	PULSE 输出端	DIR 输出端
0		高电平		低电平
1		高电平		低电平
2		低电平		高电平
3		低电平		高电平
4		高电平	高电平	
5		低电平	低电平	
6	AB 相输出，A 超前 B 90°		AB 相输出，B 超前 A 90°	

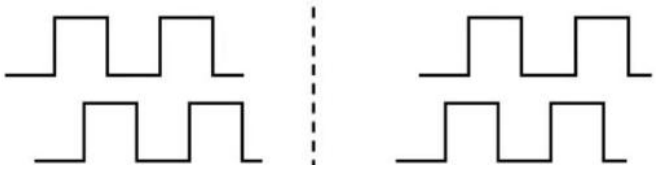
脉冲方向模式(PUL/DIR)：PUL+、PUL-脉冲线输出指令脉冲数，脉冲数对应电机运行距离，而脉冲频率对应电机运行速度。 DIR+、DIR-方向线输出方向信号，该信号的不同电平对应电机不同的转动方向。



双脉冲模式(CW/CCW)：两根线都输出脉冲信号，CW 为正方向脉冲信号，CCW 为反方向脉冲信号，通常都是差分方式输出，两信号相位相差角度，根据相位超前或滞后来决定。



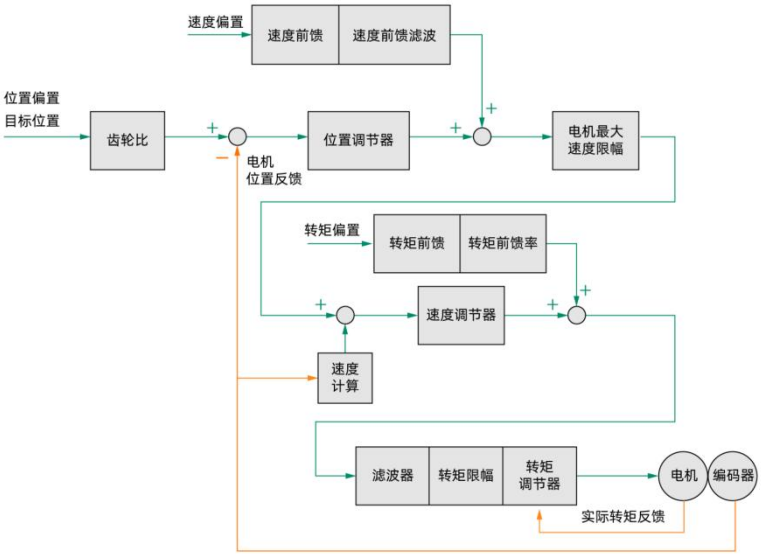
正交脉冲模式(A/B 相)：正交脉冲指两个相互独立的相同脉冲信号，正方向脉冲信号在负方向脉冲信号之前产生，二者相位相差 90 度，此时为正向旋转；负方向脉冲信号在正方向脉冲信号之前产生，二者相位相差 90 度，此时为负向旋转。 通过两个脉冲之间的相位差来达到计数或编码的作用。



（4）EtherCAT 总线轴

EtherCAT 总线轴常用的有三种控制模式，分别为 CSP 周期位置模式、CSV 周期速度模式、CST 周期转矩模式。 CSP、CSV、CST 模式的设置，如果通过 PDO 设置，就需要预先设置驱动器的 PDO 列表，PDO 包含对应的数据字典时，才可切换成该模式；如果通过 SDO 设置，直接调用模式配置函数即可完成。驱动器默认 PDO 列表内置的数据字典需要查看驱动器手册确定。

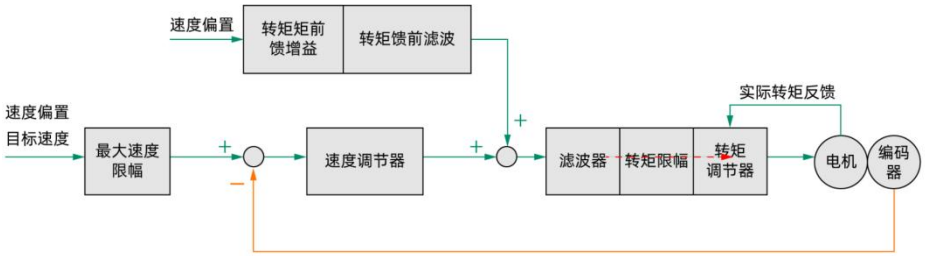
（a）当驱动器 PDO 列表包含数据字典 607Ah 时，配置为 CSP 模式，此时使用位置运动指令控制电机运动，并设置运动速度。CSP 模式下，控制卡完成位置指令规划，然后将规划好的目标位置周期性地发送给伺服驱动器，位置、速度、转矩控制由伺服驱动器内部完成。



驱动器 CSP 模式

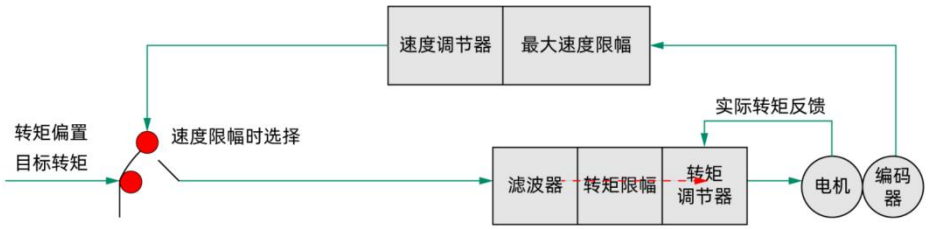
（b）当驱动器 PDO 列表包含数据字典 60FFh 时，配置为 CSV 模式，此时使用专

用的速度指令控制电机以设置值的转速运行。CSV 模式下，控制卡将计算好的目标速度周期性同步地发送给伺服驱动器，速度、转矩调节由伺服内部执行。



驱动器 CSV 模式

（c）当驱动器 PDO 列表包含数据字典 6071h 时，轴配置为 CST 模式，此时使用专用的转矩指令控制电机以设置值的转矩运行。CST 模式下，上位控制器将计算好的目标转矩周期性同步的发送给伺服驱动器，转矩调节由伺服内部执行



驱动器 CST 模式

（5）轴的当量 units

在运动控制装置中，当机械结构确定以后，电机和机械装置的传动关系也就固定了，电机每转一圈产生的机械位移量也就固定了。脉冲当量是每单位对应的脉冲数，可以是单位距离、单位角度等。 控制卡以脉冲当量作为基本单位，运动的目标位置、速度、加减速度等都是以脉冲当量作为基础单位来进行运算执行，脉冲当量修改后，目标位置、速度、加减速度等会随脉冲当量改变成比例变化。

假设脉冲型运动控制卡，控制卡发 1000 个脉冲，设备机构走 1mm,则把脉冲当量 units 设置为 1000 时，运动中 move 相关的指令，走 1mm，实际发出了 1000 个脉冲。

脉冲当量 units 参数值，必须大于 0。

（6）轴的最大速度和加速度

通过函数 NV_AxisSetMaxVel 和 NV_AxisSetMaxAcc 设置轴的最大速度和加速度。在系统中，通过这两参数限制轴运动的最大速度和最大加速度。

（7）设置总线轴控制模式

总线轴工作在 CSP、CSV、SCT 模式，通过函数 NV_AxisSetBusMode 进行设置。

总线伺服支持的模式，各厂商有所差异，请查阅对应的手册。

控制卡中轴控配置的模式编号，如下定义的枚举值

```
typedef enum
{
BUS_MODE_NULL  = 0,
BUS_MODE_PP= 1,
BUS_MODE_VL= 2,
BUS_MODE_PV= 3,
BUS_MODE_TQ= 4,
BUS_MODE_HM= 6,
BUS_MODE_IP= 7,
BUS_MODE_CSP   = 8,
BUS_MODE_CSV   = 9,
BUS_MODE_CST   = 10,
BUS_MODE_MAX   = 11
} BUS_MODE_TYPE;
```

4.4.2 轴状态配置及读取

函数列表

分类	名称	功能
轴规划状态	NV_AxisSetPrfPos	修改轴的当前规划位置
	NV_AxisGetPrfPos	获取轴的当前规划位置
	NV_AxisGetPrfVel	获取轴的当前规划速度
	NV_AxisGetPrfAcc	获取轴的当前加速度
	NV_AxisGetPrfMode	获取轴的运动模式

轴运动状态	NV_AxisCheckDone	检测轴状态
	NV_AxisGetStatus	获取轴运动的各种状态
	NV_AxisSetReached	设置轴到位检测
	NV_AxisGetReached	获取轴到位检测状态

重点说明

（1）轴规划状态

通过轴规划相关函数，可以读取轴当前规划状态，包括位置、速度。

（2）轴运动状态

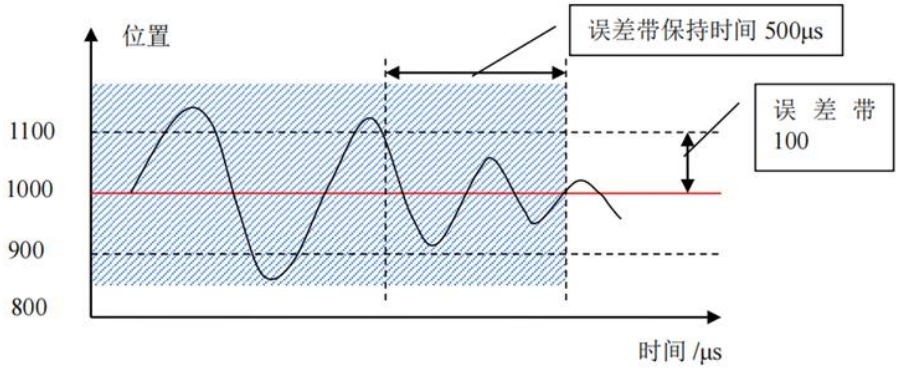
轴是否处于运动中：在调用一个运动指令后，如果需要判断当前运动是否已经执行完成，通过函数 NV_AxisCheckDone 的状态参数 pStatus 即可实现。NV_AxisCheckDone，只适应于单轴运动、同步运动、PVT 运动，不适用于多轴插补运动。

轴的状态：需要获取轴的状态，通过函数 NV_AxisGetStatus 指令读取，参数 pStatus 的各 Bit 位表示的意义如下表。

Bit 位	轴状态说明
0	idle，已停止
1	Run，运行中
2	Stopping，停止中
3	Homing，回零中
4	Homed，已完成回零动作
5	Inp，等待 inp 信号中
6	正在配置网络中
7	In Gantry，处于龙门状态
8	In Gear，处于电子齿轮的从轴状态
9	In Cam，处于电子凸轮的从轴状态
10	In Robot，处于各种专用机械结构中
11	In interpolate，在轨迹运动中
12	Error，轴处于错误中

轴到位检测：用户使用伺服电机时，由于伺服电机在运动的过程中会存在运动滞后，即出现规划停止，而实际位置并没有到位的情况。用户可以使用 NV_AxisSetReached 配置到位检测，并通过 NV_AxisGetReached 检测轴是否已经到位。检测电机到位标志可以保证系统的定位精度，应当根据机械系统的实际情况设置合适的到位误差带和误差带保持时间。如果到位误差带设置的太小，或者误差带保持时间太长，都会使到位时间增长，影响加工效率。

如下图所示，当轴启动电机到位检测功能，设置误差带为 100，误差带保持时间为 500us，当电机在 1000 位置处静止时，会有震荡。控制卡判断其震荡位于误差带内 500us 后，就将轴状态的第 1 位值。蓝色阴影区域表示电机到位标志还未置 1，无阴影区域表示已置 1。可以看出，误差带保持时间越短，误差带越大，控制卡会在越短时间内将电机到位标志置 1。



电机到位检测误差带

使用电机到位检测功能必须注意以下几点：

（a）axis 正确关联编码器，并且编码器方向和规划运动方向必须一致。

（b）正确设置到位误差带，默认情况下到位误差带无效。

4.4.3 轴编码器操作

函数列表

名称	功能
NV_AxisSetEncMapping	设置轴的编码器映射关系
NV_AxisGetEncMapping	获取轴的编码器映射关系
NV_AxisSetEncUnit	设置轴的编码器脉冲当量

NV_AxisGetEncUnit	获取轴的编码器脉冲当量
NV_AxisSetEncPos	修改轴的编码器位置
NV_AxisGetEncPos	获取轴的编码器位置

重点说明

控制卡内部为每个轴配置了反馈脉冲计数装置即编码器计数，默认的计数源是伺服驱动反馈回的脉冲，因此在接线时，需要将伺服驱动的脉冲反馈连接到轴对应的编码器上，通过函数 NV_AxisGetEncPos 读取编码器值；如果是步进驱动，那么就没有反馈信号，不需要连接，读取的计数值为 0。

调用 NV_AxisSetEncMapping 配置轴关联的编码器输入通道，配置完成后可通过轴号操作对应编码器计数位置值。

调用 NV_AxisSetEncPos 修改编码器位置的值。例如，将轴 1 的编码器位置设置为 500，则接下来的编码器计数从 500 开始；若设置为 1000，则编码器计数从 1000 开始。

调用 NV_AxisSetEncMapping 函数时，设置对应通道的编码器脉冲当量，设置该值后，读取的编码器位置也转换成当量值。如读取编码器值为 2000，表示编码器当前值为 2000unit。

4.4.4 轴 IO 操作

函数列表

名称	功能
NV_AxisIoSetMapping	设置轴 IO 映射关系
NV_AxisIoGetMapping	设置轴 IO 映射关系
NV_AxisIoSetEnable	设置轴 IO 专用使能
NV_AxisIoGetEnable	读取轴 IO 专用使能
NV_AxisIoSetDiLogic	设置轴专用输入信号的有效电平
NV_AxisIoGetDiLogic	读取轴专用输入信号的有效电平
NV_AxisIoGetDi	读取轴专用 IN 信号
NV_AxisIoGetDiStatus	读取轴专用 IN 信号
NV_AxisIoSetDoBit	设置轴专用输出信号

NV_AxisIoGetDoBit	读取轴专用输出信号
NV_AxisIoSetOrgLogic	设置轴原点输入信号的有效电平
NV_AxisIoGetOrgLogic	读取轴原点输入信号的有效电平
NV_AxisIoSetEzLogic	设置轴 EZ 输入信号的有效电平
NV_AxisIoGetEzLogic	读取轴 EZ 输入信号的有效电平
NV_AxisIoSetHardLimit	设置轴的硬件限位功能
NV_AxisIoGetHardLimit	获取轴的硬件限位功能状态
NV_AxisIoSetDecStop	设置轴的硬件减速停止功能
NV_AxisIoGetDecStop	获取轴的硬件减速停止功能状态
NV_AxisIoSetEmgStop	设置轴的硬件急停功能
NV_AxisIoGetEmgStop	获取轴的硬件急停功能状态
NV_AxisIoSetAlarm	设置伺服报警功能
NV_AxisIoGetAlarm	获取伺服报警功能状态
NV_AxisIoSetInp	设置伺服到位功能
NV_AxisIoGetInp	获取伺服到位功能状态
NV_AxisIoSetRdy	设置伺服准备好功能
NV_AxisIoGetRdy	获取伺服准备好功能状态

重点说明

（1）轴专用信号

轴的原点信号：在轴运动平台上，一般有一个原点传感器，它用于表示轴的零点位置，一般用 ORG 标识。

轴的正负限位信号：在运动平台上，正向行程和负向行程一般也会安装一个限位传感器，表示正向和负向的极限位置，一般用 EL+和 EL-标识。

伺服电机控制信号：设备中有交流伺服电机时，控制卡上的伺服电机控制信号 SEVON、RDY、ALM、INP 和 RESET 与伺服电机相连，通过这些信号控制伺服驱动。

SEVON 是控制卡输出给伺服电机驱动器的使能信号，当 SEVON 信号为无效状态时，伺服驱动器不使能，电机处于自由状态；当 SEVON 信号有效时，伺服驱动器使能，电机锁紧，等待指令脉冲信号。

RDY 是伺服电机驱动器发给控制卡的状态信号，当 RDY 信号为有效时，表示伺服

驱动器已经准备好，控制卡接收到该信号后就可以向伺服驱动器发出运动命令；如果RDY 信号无效， 表示伺服驱动器还未准备好，这时控制卡发出指令脉冲信号，伺服驱动器也不会运动。

ALM 信号是伺服报警信号，是伺服驱动器发送给控制卡的信号。控制卡提供专用的驱动报警信号输入接口，当检测到驱动器报警信号以后，控制卡将关闭该轴的伺服使能，立即停止运动规划，同时该轴报警触发标志置 1。

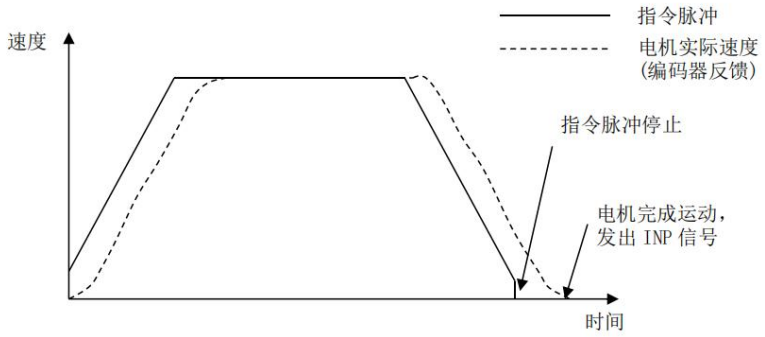
驱动器报警信号产生以后，应当执行以下操作：

（a）了解驱动器报警状态，确定引起驱动器报警的原因，解决驱动器报警；

（b）调用清除报警，控制卡的轴才可以正常运动。

是否启动伺服报警功能，通过函数 NV_AxisIoSetAlarm 设置。

INP 信号，是从伺服电机驱动器发给控制卡的状态信号，告知控制卡伺服电机已经停止。伺服电机驱动器内部有一个位置偏差计数器，记录指令脉冲和位置反馈脉冲之间的偏差；伺服电机驱动器将控制电机运动使位置偏差趋于 0，但是电机实际位置总是滞后于指令脉冲，所以当运动控制卡的指令脉冲发送完毕时，伺服电机并没有立即停止，而是继续运动，如下图所示，直到位置偏差趋于 0；这时驱动器将发出一个 INP 信号。



伺服定位完成时的 INP 信号

RESET 信号是控制卡输出给伺服电机驱动器的控制信号。当伺服驱动报警后，控制卡发送该信号，立即清除驱动器报警。

（2）轴专用 IO 信号在函数中 Bit 位定义

上述轴 IO 信号，在函数 NV_AxisIoSetEnable 、 NV_AxisIoGetEnable 、NV_AxisIoSetDiLogic、NV_AxisIoGetDiLogic、NV_AxisIoGetDi、NV_AxisIoGetDiStatus 中进行操作时，按照 Bit 位进行表示，如下表所示：

Bit 位	信号名称
0	ELN 负限位信号
1	ELP 正限位信号
2	ORG 原点信号
3	EMG 急停信号
4	DEC 减速停止信号
5	ALM 伺服报警信号
6	RDY 伺服准备信号
7	INP 伺服到位信号
8	EZ 编码器 Z 相信号

（3）轴专用 IN 的配置及读取

轴专用 IN 的有效电平，通过函数 NV_AxisIoSetLogic 设置；有效电平值 Value 的表示方式，按照 bit 位设置，高有效将该位置 1，低有效将该位置 0。

轴专用 IN 信号，通过函数 NV_AxisIoGetDi 读取，按照 Bit 位排布。

上述函数，按照实际硬件电路的物料端口名称读取信号。

轴专用 IN 信号有效电平，通过函数 NV_AxisIoGetStatus 读取，按照 Bit 位排布。

上述函数，读取的状态值，根据映射配置后引脚排布，有效电平由函数 NV_AxisIoSetLogic 配置。

如：轴 0，负限位信号，配置有效电平为 0，那么当该端口电平信号为低时。

函数 NV_AxisIoGetDi 读取的第 0 位，值为 0；

函数 NV_AxisIoGetStatus 读取的第 0 位，值为 1；

假设，负限位配置有效电平为 1，那么当该端口电平信号为低时。

函数 NV_AxisIoGetDi 读取的第 0 位，值为 0；

函数 NV_AxisIoGetStatus 读取的第 0 位，值为 0；

假设，负限位配置有效电平为 1，那么当该端口电平信号为高时。

函数 NV_AxisIoGetDi 读取的第 0 位，值为 1；

函数 NV_AxisIoGetStatus 读取的第 0 位，值为 1；

（4） 轴 IO 映射功能

控制卡支持轴 IO 映射配置功能，支持将轴专用 IO 信号配置到任意一个硬件输入口，如可将限位接口当原点信号。该功能可减少现场接线、换线的困难。

4.5 硬件资源

本章主要介绍如何通过指令访问硬件资源，硬件资源类型详细说明如下：

数字量输出资源：包括高速数字量输出、通用数字量输出、专用数字量输出（如伺服使能数字量输出、伺服报警清除数字量输出等）。

数字量输入资源：包括高速数字量输入、普通数字量输入、专用数字量输入（正限位数字量输入、负限位数字量输入、驱动报警数字量输入、原点信号数字量输入等）。

编码器计数资源：用来对外部编码器的脉冲输出进行计数。

模拟量输入资源：电压输入通道，用来获取外部模拟量输入信号。

模拟量输出资源：电压输出通道，输出模拟控制电压。

4.5.1 数字 IO 操作

函数列表

分类	名称	功能
通用 IO 操作函数	NV_IoGetDi	读取一批输入信号
	NV_IoGetDiBit	读取单个输入口状态
	NV_IoSetDiCountMode	设置输入的计数模式
	NV_IoGetDiCountMode	获取输入的计数模式
	NV_IoSetDiCountValue	修改当前输入的计数值
	NV_IoGetDiCountValue	获取当前输入的计数值
	NV_IoSetDiFilter	设置滤波时间
	NV_IoGetDiFilter	获取滤波时间
	NV_IoSetDo	设置批量输出
	NV_IoGetDo	获取批量输出
	NV_IoSetDoMask	有条件地设置批量输出

分类	名称	功能
	NV_IoSetDoBit	设置单个输出
	NV_IoGetDoBit	获取单个输出的电平
	NV_IoSetDoBitReverse	设置单个输出按时翻转
	NV_IoSetDoBitPulse	控制输出信号产生脉冲波形
	NV_IoClrDoBitPulse	中断输出信号产生脉冲波形
高速 IO 操作函数	NV_HSIoGetDi	高速 IO 读取一批输入信号
	NV_HSIoGetDiBit	高速 IO 读取单个输入口状态
	NV_HSIoSetDo	高速 IO 设置批量输出
	NV_HSIoSetDoMask	高速 IO 有条件地设置批量输出
	NV_HSIoGetDo	高速 IO 获取批量输出
	NV_HSIoSetDoBit	高速 IO 设置单个输出
	NV_HSIoGetDoBit	高速 IO 获取单个输出的电平
	NV_HSIoSetDiFilter	设置输入的滤波时间
	NV_HSIoGetDiFilter	获取输入的滤波时间

重点说明

控制卡上的数字 I/O 口，可以用于检测开关信号、传感器信号等输入信号，或者控制继电器、电磁阀等输出设备，或者与 PLC 系统交互 IO 信号。

普通 IO 函数既可以操作普通 IO，也可以操作高速 IO。

高速 IO 函数仅能操作高速 IO，一般可不用，直接使用普通 IO 操作函数即可。

对于 IO 端口，可以批量一次性操作，也可以按照 Bit 位，一个个操作。

通用输入端口支持计数功能，最大支持 16 个计数器，支持信号沿变化计数，计数模式支持上升沿、下降沿、任意沿，通过函数 NV_IoSetDiCountMode 进行配置；

CountMode 参数的计数模式如下：

计数模式	说明
0	禁用
1	上升沿计数
2	下降沿计数
3	任意沿计数

通用输出端口支持输出单个翻转脉冲，翻转电平时间通过参数配置，该功能通过 NV_IoSetDoBitReverse 指令实现；输出端口支持输出指定数量的脉冲，通过调用 NV_IoSetDoBitPulse 指令实现。

注意：在批量进行 IO 电平读取显示时，建议使用十六进制数进行赋值（尽量避免使用十进制数，特别是在不支持无符号变量的开发环境下），因为使用十六进制数比使用十进制数更加直观、方便。

例程

例程：普通 IO 操作，请参考工程“IOSample”。

IO 的读取：

```
UInt16[] pValue = new ushort[m_nButtonINCount];
ret = NV_IoGetDiBit(m_nConnectNo, 0, pValue, m_nButtonINCount);
```

IO 的输出：

```
ret = NV_IoSetDoBit(m_nConnectNo, nIndex, (ushort)(1 - pValue));
```

例程 2：高速 IO 操作，请参考工程“HSIOSample”。

高速 IO 的读取：

```
UInt16[] pValue = new ushort[m_nButtonINCount];
ret = NV_HSIoGetDiBit(m_nConnectNo, 0, pValue, m_nButtonINCount);
```

高速 IO 的输出：

```
ret = NV_HSIoSetDoBit(m_nConnectNo, nIndex, (ushort)(1 - pValue));
```

4.5.2 编码器操作

函数列表

名称	功能
NV_EncoderSetMode	设置编码器的工作模式

NV_EncoderGetMode	获取编码器的工作模式
NV_EncoderSetFilter	设置编码器的滤波参数
NV_EncoderGetFilter	读取编码器的滤波参数
NV_EncoderSetPulsePos	设置编码器的脉冲计数值
NV_EncoderGetPulsePos	读取编码器的脉冲计数值

重点说明

控制卡支持差分编码器输入，可以用于检测平台的位移或电机的转角。

编码器有 EA、EB 两个差分信号，脉冲计数信号由 EA 和 EB 端口输入，计数值为 32 位，计数范围-2^31~2^31。

在编码器计数前，需要设置编码器的工作方式，计数方向及分频数，分频支持 1、2、4 分频。

编码器的计数值为原始计数值，单位为脉冲数。计数值的读取和设置，通过函数 NV_EncoderGetPulsePos 和 NV_EncoderSetPulsePos 操作。

例程

例程：编码器设置，请参考工程“EncoderSample”

设置编码器工作模式：

```
ERROR_CODE_TYPE ret = 0;
UInt16 EncMode = 0;   //计数模式：0 - AB 相
UInt16 EncDir = 0;//计数方向: 0 -  正向计数
UInt16 EncDivision = 0;//计数分频：0，1 - 1 分频
ret=NV_EncoderSetMode(m_nConnectNo,m_nChannel,EncMode, EncDir, EncDivision);
```

设置编码器值：

```
ERROR_CODE_TYPE ret = 0;
Int32 Value = 1000; //设置编码器值为 1000
ret = NV_EncoderSetPulsePos(m_nConnectNo, m_nChannel, Value);
```

4.5.3 模拟量操作

函数列表

分类	名称	功能
模拟输入函数	NV_ADSetType	设置 AD 的采样类型
	NV_ADGetType	读取 AD 采样类型
	NV_ADGetValue	读取 AD 值
模拟输出函数	NV_DASetEnable	设置 DA 使能
	NV_DAGetEnable	读取 DA 使能
	NV_DASetValue	设置 DA 值
	NV_DAGetValue	读取 DA 值

重点说明

（1）模拟输入

控制卡有些型号支持模拟量输入，不同型号有些差异，具体的规格指标请查阅对应的用户手册。

对于 AD 支持电压和电流采样的控制卡，通过函数 NV_ADSetType 配置采样的类型，其类型编号如下：

编号	采样类型
0	-10-10V
1	- 5V～5V
2	1V～5V
3	0-10V
4	0-5V
5	-20-20MA
6	4-20MA
7	0-20MA

（2）模拟输出

控制卡有些型号支持模拟量输出，具体的规格指标请查阅对应的用户手册。

例程

例程：模拟量的读取和输出，请参考工程“ADCSample”。

AD 的读取：

```
ushort m_nChannelADC = 0;
Int32 pValue = 0;
ret = NV_ADGetValue(m_nConnectNo, m_nChannelADC, ref pValue);
```

DA 的使能和设置输出：

```
UInt16 Enable = 1;
UInt16 Value = 3;
ushort m_nChannelDAC = 0;
ret = NV_DASetEnable(m_nConnectNo, m_nChannelDAC, Enable);
if (ret != 0)
{
MessageBox.Show($"NV_DASetEnable 调用报错：{ret}");
return;
}

ret = NV_DASetValue(m_nConnectNo, m_nChannelDAC, Value);
if (ret != 0)
{
MessageBox.Show($"NV_DASetValue 调用报错：{ret}");
return;
}
```

4.5.4 手轮操作

函数列表

分类	名称	功能
手轮参数配置	NV_HandwheelSetEncMapping	设置手轮编码器映射关系
	NV_HandwheelGetEncMapping	获取手轮编码器映射关系
	NV_HandwheelSetEncoderPos	设置手轮编码器值
	NV_HandwheelGetEncoderPos	读取手轮编码器值
	NV_HandwheelGetHardInStatus	获取手轮上的输入信号状态
	NV_HandwheelSetHardAxisSel	设置手轮硬件轴选挡位对应的控制轴
	NV_HandwheelGetHardAxisSel	获取手轮硬件轴选挡位对应的控制轴
	NV_HandwheelSetHardRatioSel	设置手轮硬件倍选挡位对应的倍率值
	NV_HandwheelGetHardRatioSel	获取手轮硬件倍选挡位对应的倍率值
	NV_HandwheelSetSoftParam	设置手轮软件控制参数
	NV_HandwheelGetSoftParam	获取手轮软件控制参数
运动指令	NV_HandwheelMove	启动手轮运动
	NV_HandwheelStop	退出手轮运动

重点说明

控制卡可以支持手轮功能，对于输入编码器，可以配置（映射）为手轮编码器输入，通过函数 NV_HandwheelSetEncMapping 进行配置。

如果控制卡上有专用的手轮接口，通过手轮的专用函数 NV_HandwheelGetHardInStatus 可以读取手轮上的轴选信号、倍率信号、急停信号；通过函数 NV_HandwheelSetHardAxisSel 和 NV_HandwheelSetHardRatioSel 配置手轮硬件轴选和倍选对应的参数值。

如果控制卡上没有专用的手轮接口，轴号和倍率只能通过软件设置，通过函数 NV_HandwheelSetSoftParam 配置手轮参数。

函数 NV_HandwheelMove 启动手轮运动，通过 NV_HandwheelStop 函数退出手轮模式。

例程

例程：硬件控制手轮运动，请参考工程“HandwheelSample”。

设置手轮编码器映射：

```
ERROR_CODE_TYPE ret = 0;
UInt16 MapEncType = 3;  //映射编码器通道类型: 3 手轮编码器
UInt16 MapEncNo =  0;  //映射编码器通道编号:0 通道
Double SmoothTime =  0.50;//平滑时间:0.5s
ret=NV_HandwheelSetEncMapping(m_nConnectNo, MapEncType, MapEncNo, SmoothTime);
```

设置手轮的轴选：

```
for (UInt16 i = 0; i < 6; i++)
{
UInt16 AxisSel = i;
UInt16 AxisNo = (ushort)_AxisSel[AxisSel].SelectedIndex;
ret = NV_HandwheelSetHardAxisSel(m_nConnectNo, AxisSel, AxisNo);
if (ret != 0)
{
MessageBox.Show($"NV_HandwheelSetHardAxisSel 调用报错：{ret}");
return;
}
}
```

设置手轮的倍率选择：

```
for (UInt16 i = 0; i < 3; i++)
{
UInt16 RatioSel = i;
Double Ratio = (double)_RatioSel[RatioSel].Value;
```

```
ret = NV_HandwheelSetHardRatioSel(m_nConnectNo, RatioSel, Ratio);
if (ret != 0)
{
MessageBox.Show($"NV_HandwheelSetHardRatioSel 调用报错：{ret}");
return;
}
}
```

手轮运动：

```
ERROR_CODE_TYPE ret = 0;
UInt16 Mode = 0;   //手轮模式:0 硬件手轮
ret = NV_HandwheelMove(m_nConnectNo, Mode);
```

4.6 单轴基本运动

4.6.1 运动描述

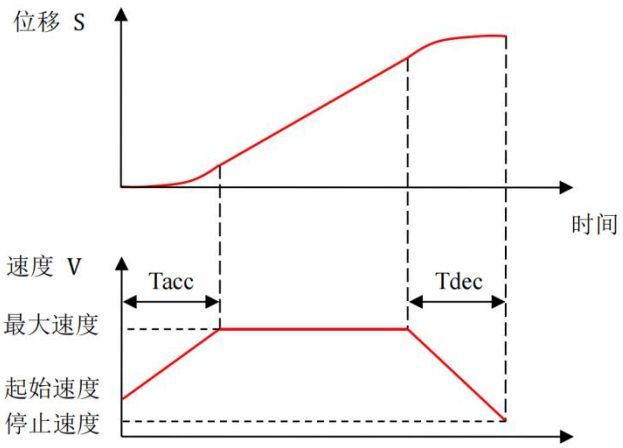
单轴运动是指规划一个轴运动的轨迹；常用的单轴基本运动包括单轴点位运动、连续运动。用户根据实际需求选择相应的运动模式。

单轴点位运动：指轴从当前位置开始，以设定的速度运动到指定位置后，准确地停止的运动，核心要素是停止在指定的目标位置，常用指令如 NV_PmoveAbsolute，NV_PmoveRelative。

连续运动：指轴从起始速度开始运行，加速至最大速度后连续运动；只有当接收到停止指令或外部停止信号后，才减速停止，常用指令如 NV_VmoveMove、NV_JogSoftMove。

在单轴运动下，各轴可以独立设置目标位置、目标速度、加速度、减速度、起始速度、平滑时间等运动参数，因此在运动前，需要对轴的这些参数进行设置。

为了让轴在运动过程中能平稳加速、准确停止，一般采用梯形速度曲线控制运动过程，如下图所示，即电机以起始速度开始运动，加速至最大速度后保持速度不变，在运动结束前减速至停止速度，最终停止在指定位置。



梯形速度曲线及对应的位移曲线

为改善轴运动的平稳性，控制卡还提供了 S 形速度控制曲线。在 S 形速度控制过程中，指令速度从一个内部设定的速度（运动算法内部计算）快速加速到起始速度，然后作 S 形加速运动；运动结束前，指令速度作 S 形减速运动到停止速度，然后再快速减速到一个内部设定的速度，这时运动停止，如下图所示。S 型速度的实现，通过设置平滑参数来完成。

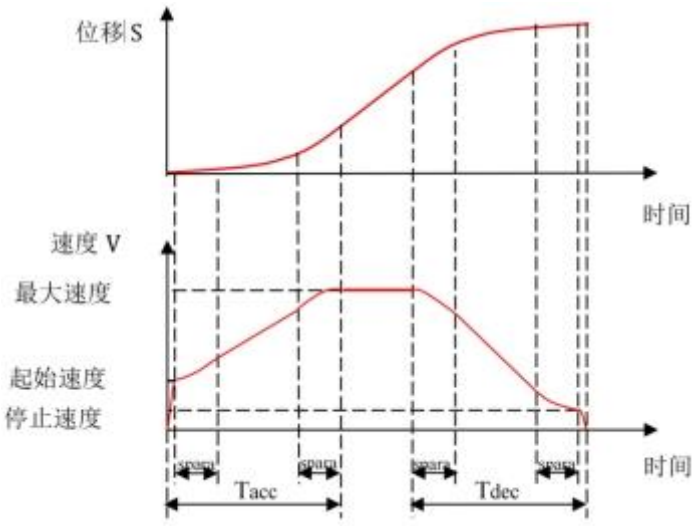


图 S 形速度曲线及对应的位移曲线

spara：S 段时间，Tacc：总加速时间， Tdec：总减速时间

4.6.2 回零运动

函数列表

名称	功能
NV_HomeSetProfile	设置回零的参数
NV_HomeGetProfile	获取回零的参数
NV_HomeStatus	获取回零的状态
NV_HomeError	获取回零失败的原因
NV_HomeMove	启动回零运动

重点说明

（1）回零功能说明

控制卡上有两种类型的轴，脉冲轴和总线轴，回零功能调用相同的函数，通过函数 NV_HomeSetProfile 中的 HomeWay 参数进行区分。

回零参数结构体，定义如下：

```
typedef struct HomeProfileStruct
{
    Int16 HomeWay;//回零方式：0 - 本地回零；1 - 总线回零
    Int16 HomeMode;    //回零模式，详细参考 DS402 回零说明
    Int16 OffsetMode;    //回零的偏移模式：0-不偏移,直接设置为偏移位置;
1-清零后偏移;
2-偏移后清零
    Double Offset;        //回零的偏移值
    Double LowSpeed;   //回零的最低速度
    Double HighSpeed;  //回零的最高速度
    Double HomeAcc;    //回零的加速度
}HomeProfileStruct_t
```

总线轴的回零，根据伺服驱动器支持的模式，配置好参数后，伺服自动回零。回零

动作执行前，需要特别注意以下几点：（1）驱动器的位置、速度和加速度参数的单位，不同驱动器这些参数单位可能不同，如果设置的速度和加速度太高或太低，会导致回零失败；（2）确保驱动器的限位、原点和 Index 信号的接线正确；

脉冲轴的回零，控制卡是按照 DS402 规则设计的回零动作，用户配置好参数后，控制卡控制电机回零。回零前需要特别注意选择的回零模式与接线一定要匹配，如步进驱动器（没有 Index 信号）选择有 Index 信号的回零方式，这种配置方式就不正确。

回零的状态通过 NV_HomeStatus 函数读取，回零动作处于哪个阶段由参数 pStep 表示，具体编号如下：

pStep 编号	说明
0	静止状态，未启动回零
1	前向快速寻找零点
2	触碰限位后退找零点，或在零点状态下回找零点边缘
3	前向快速寻找 EZ
4	后退快速寻找 EZ
5	前向快速寻找 LIMIT
6	后退快速寻找 LIMIT
7	慢速前向寻找零点边缘
8	慢速后退寻找零点边缘
9	慢速前向寻找 EZ 边缘
10	慢速后退寻找 EZ 边缘
11	慢速前向寻找 LIMIT 边缘
12	慢速后退寻找 LIMIT 边缘
13	定位到原点/限位锁存位置
14	定位到 EZ 锁存位置
15	定位到 POT/NOT 锁存位置
17	命令发送过来的减速停止
18	异常情况（限位）减速停止
19	处理 offset 的相关操作
20	完成回零过程，但不代表是回零成功

（2）脉冲轴支持的回零模式

脉冲轴支持多种回零模式，按照 DS402 规则执行，各模式详细描述请查阅“附件：回零模式”。

例程

例程：总线回零，请参考工程“HomeMoveSample”

总线回零参数设置：

```
m_sProfile.HomeWay = 1;//回零方式：1 - 总线回零
m_sProfile.HomeMode = 1;//回零模式:1
m_sProfile.OffsetMode = 0;//回零的偏移模式：0-不偏移
m_sProfile.Offset = 0;//回零的偏移值
m_sProfile.LowSpeed = 100;//回零的最低速度
m_sProfile.HighSpeed = 2000;//回零的最高速度
m_sProfile.HomeAcc = 20000;//回零的加速度
ret = NV_HomeSetProfile(m_nConnectNo, m_nAxisNo, m_sProfile);
if (ret != 0)
{
MessageBox.Show($"NV_HomeSetProfile 调用报错：{ret}");
return;
}
```

4.6.3 点位运动

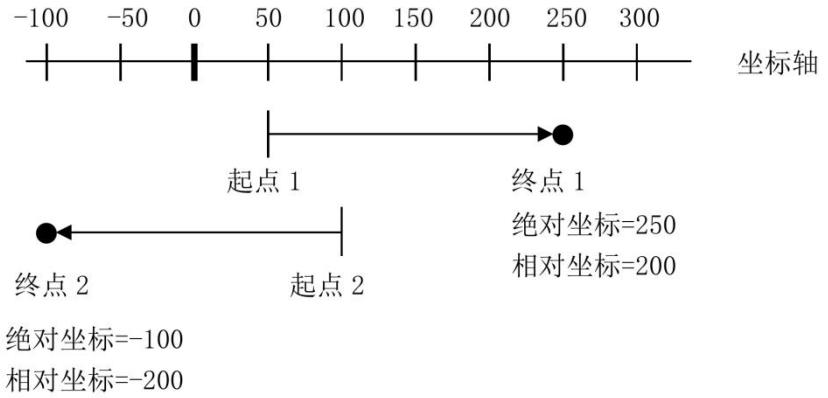
函数列表

分类	名称	功能
运动参数	NV_PmoveSetProfile	设置定长运动的参数
	NV_PmoveGetProfile	获取定长运动的参数
运动函数	NV_PmoveAbsolute	设置绝对位置定长参数并启动运动
	NV_PmoveRelative	设置相对位置定长参数并启动运动
	NV_PmoveTwice	设置两段位置并启动运动

在线变位	NV_PmoveChangePos	修改点位运动的目标位置或者在停止的时候进行新的一个运动
规划信息	NV_PmoveGetPlanInfo	获取规划信息

重点说明

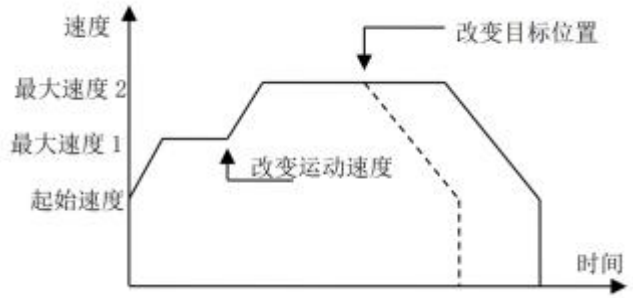
（1）控制卡在描述运动轨迹时可以用绝对坐标也可以用相对坐标，如下图所示。



相对模式与绝对模式，各有优点，如：在绝对坐标模式中用一系列坐标点定义一条曲线，如果要修改中间某点坐标时，不会影响后续点的坐标；在相对坐标模式中，用一系列坐标点定义一条曲线，用循环命令可以重复这条曲线轨迹多次。用户根据实际需要选择合适的模式即可。

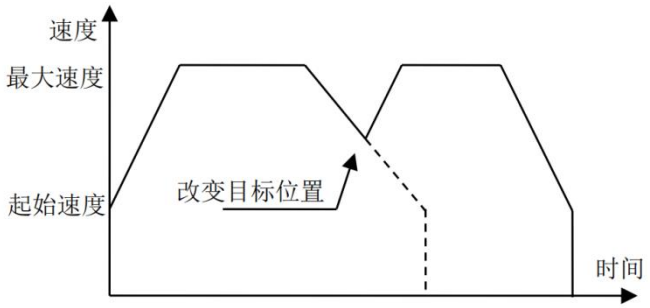
（2）控制卡的函数库中距离或位置的单位为 units，速度单位为 units/s，加速度/减速度单位为 unit/s^2。

（3）在点位运行过程中，最大速度 MaxVel 和目标位置 Dist 均可以实时改变，其典型速度时间曲线如下图所示。



改变最大速度及目标位置

（4）在减速阶段，也可以改变目标位置，电机的速度将如下图所示发生变化。



减速时改变目标位置

（5）单轴运动时，需要设置运动的参数，通过结构体 ProfileStruct 设置，其结构体参数如下：

```
typedef struct ProfileStruct
{
    UInt16 IsValid;// 参数是否有效
    Double StartSpeed; // 起始速度
    Double MaxSpeed;    // 最大速度
    Double StopSpeed;   // 停止速度
    Double Acceleraton;// 加速度
    Double Deceleraton;// 减速度
    Double SmoothTime; // 平滑时间
}ProfileStruct_
```

（6）单轴做两段运动时，通过 Profile2Struct_t 结构体设置参数，其参数如下：

```
typedef struct Profile2Struct
{
    Double Position1;    // 第一段运动的相对或者绝对位置（根据 Mode 决定）
    Double Position2;    // 第二段运动的相对或者绝对位置（根据 Mode 决定）
    Double StartSpeed;   // 起始速度
    Double MaxSpeed1;    // 第一段最大速度
```

Double LinkSpeed; // 第二段最大速度

Double MaxSpeed2; // 停止速度

Double StopSpeed; // 加速度

Double Acceleraton1;// 第一段运行加速度

Double Deceleraton1;// 第一段运行减速度

Double Acceleraton2;// 第二段运行加速度

Double Deceleraton2;// 第二段运行减速度

UInt16 PosMode; // 位置模式：0 - 绝对坐标模式；1 - 相对坐标模式

Double SmoothTime; // 平滑时间，单位 s，范围 0-1

}Profile2Struct_t;

例程

例程：单轴作点位运动，请参考工程“P2PMoveSample”。

设置轴运动参数：

```
ERROR_CODE_TYPE ret = 0;
m_sProfile.StartSpeed = 100;  // 起始速度
m_sProfile.MaxSpeed = 2000;  // 最大速度
m_sProfile.StopSpeed = 20;// 停止速度
m_sProfile.Acceleraton = 500; // 加速度
m_sProfile.Deceleraton = 500; // 减速度
m_sProfile.SmoothTime = 0.2;  // 平滑时间，单位 s，范围 0-1
ret = NV_PmoveSetProfile(m_nConnectNo, m_nAxisNo, m_sProfile);
```

运动到指定位置：

```
ERROR_CODE_TYPE ret = 0;
Double Position = 3500; // 运动目标位置
ret = NV_PmoveAbsolute(m_nConnectNo, m_nAxisNo, Position, m_sProfile);
```

相对运动一段距离：

```
ERROR_CODE_TYPE ret = 0;
Double Distance = 6500; //  相对运动一段距离
ret = NV_PmoveRelative(m_nConnectNo, m_nAxisNo, Distance, m_sProfile);
```

例程：改变速度、改变终点位置，请参考工程“P2PMoveSample”。

运动过程中改变速度：

```
ERROR_CODE_TYPE ret = 0;
Double NewSpeed = 200; //  改变速度
ret = NV_AxisChangeSpeed(m_nConnectNo, m_nAxisNo, NewSpeed);
```

运动过程中改变目标位置：

```
ERROR_CODE_TYPE ret = 0;
Double Position = 8600; //  改变目标位置
ret = NV_PmoveChangePos(m_nConnectNo, m_nAxisNo, Position);
```

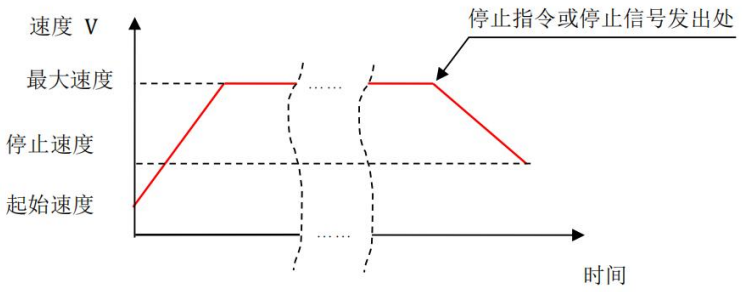
4.6.4 定速运动

函数列表

分类	名称	功能
运动参数	NV_VmoveSetProfile	设置定速运动的参数
	NV_VmoveGetProfile	获取定速运动的参数
运动函数	NV_VmoveMove	软件启动定速运动

重点说明

（1）定速运动指控制卡控制电机以梯形或 S 形速度曲线，按照指定的加速度，从起始速度加速至最大速度，然后以该速度连续运行；当遇到停止指令，或者异常停止，电机按照配置的减速度或者急停速度减速停止。连续运动的速度时间曲线如下图所示。

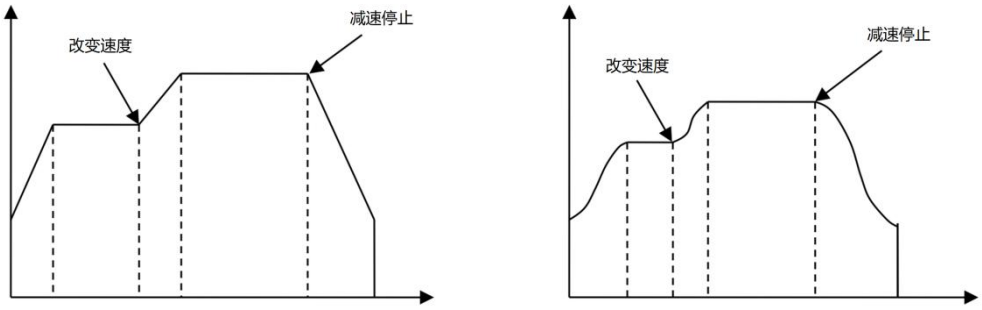


连续运动速度曲线

该功能的主要用途是：速度控制，如：传送带的速度、包装机连续送料速度等。

（2）在执行连续运动过程中，可以调用 NV_AxisChangeSpeed 实时改变速度。

注意：在以 S 形速度曲线连续运动时，改变最大速度最好在加速过程已经完成的恒速段进行。下图为梯形和 S 形速度曲线下连续运动中变速和减速停止过程的速度曲线。



梯形运动中变速　　S 形运动中变速

例程

例程：以 S 形速度曲线加速的连续运动及变速、停止运动，请参考工程“VmoveSample”。

设置定速运动的参数：

```
ERROR_CODE_TYPE ret = 0;
m_sProfile.StartSpeed = 10;   // 起始速度
m_sProfile.MaxSpeed = 200;    // 最大速度
m_sProfile.StopSpeed = 10;// 停止速度
m_sProfile.Acceleraton = 100; // 加速度
m_sProfile.Deceleraton = 100; // 减速度
```

m_sProfile.SmoothTime = 0.2; // 平滑时间，单位 s，范围 0-1

ret = NV_VmoveSetProfile(m_nConnectNo, m_nAxisNo, m_sProfile);

改变速度：

ERROR_CODE_TYPE ret = 0;

Double NewSpeed = 450; // 设置新的速度

ret = NV_AxisChangeSpeed(m_nConnectNo, m_nAxisNo, NewSpeed);

停止运动：

ERROR_CODE_TYPE ret = 0;

ret = NV_AxisStop(m_nConnectNo, m_nAxisNo, 0);

4.6.5 Jog 运动

函数列表

分类	名称	功能
配置运动参数	NV_JogSetProfile	设置 Jog 运动的参数
	NV_JogGetProfile	获取 Jog 运动的参数
配置硬件方式触发信号	NV_JogSetDiConfig	设置 Jog 硬件信号，并启动 Jog 运动
	NV_JogGetDiConfig	读取 Jog 硬件信号
启动 Jog 运动	NV_JogMove	启动 Jog 运动

重点说明

（1）Jog 运动参数配置：通过函数 NV_JogSetProfile 设置 Jog 运动参数。

（2）Jog 运动的启动，有两种方式：

软件启动方式，通过函数 NV_JogSoftMove 执行运动；

硬件输入端口触发方式，通过函数 NV_JogSetDiConfig 配置启动输入端口，设置完毕后立即生效，只要端口满足配置的电平条件，就触发运动。

（3）在运动过程中，可以调用 NV_AxisChangeSpeed 实时改变速度。

（4）Jog 运动的参数结构体如下：

```
typedef struct JogProfileStruct
{
    Double StartSpeed;   // 起始速度
    Double MaxSpeed;// 最大速度
    Double StopSpeed;    // 停止速度
    Double Acceleraton; // 加速度
    Double Deceleraton; // 减速度
    Double SmoothTime;   // 平滑时间，单位 s，范围 0-1
}JogProfileStruct_t
```

例程

例程：通过软件配置方式触发执行 Jog 运动，请参考工程“JOGSample”。

设置 Jog 运动参数：

```
ERROR_CODE_TYPE ret = 0;
m_sProfile.StartSpeed = 10;// 起始速度
m_sProfile.MaxSpeed = 200;//最大速度
m_sProfile.StopSpeed = 10;// 停止速度
m_sProfile.Acceleraton = 50;//加速度
m_sProfile.Deceleraton = 50;//减速度
m_sProfile.SmoothTime = 0;// 平滑时间，单位 s，范围 0-1
ret = NV_JogSetProfile(m_nConnectNo, m_nAxisNo, m_sProfile);
```

启动正向 Jog 运动：

```
ERROR_CODE_TYPE ret = 0;
ret = NV_JogMove(m_nConnectNo, m_nAxisNo, 1, 1, 0); //软件触发方式，正向运动
```

例程：通过硬件端口触发执行 Jog 运动，请参考工程“JOGSample”

设置硬件触发方式的硬件端口号及端口电平（IN4 电平为 0 时，正向运动；IN5 电平为 0 时，反向运动）：

UInt16 ForwardPin = 4;//正向时对应的通用输入口编号:4

UInt16 ForwardLevel = 0; //正向时对应的通用输入口有效电平:0

UInt16 ReversePin = 5;//反向时对应的通用输入口编号:5

UInt16 ReverseLevel = 0; //反向时对应的通用输入口有效电平:0

ret = NV_JogSetDiConfig(m_nConnectNo, m_nAxisNo, ForwardPin, ForwardLevel, ReversePin, ReverseLevel);

启动硬件触发 Jog：

ERROR_CODE_TYPE ret = 0;

ret = NV_JogMove(m_nConnectNo, m_nAxisNo, 0, 0, 0); //硬件触发方式，执行运动由配置的输入端口控制

4.6.6 变速运动

函数列表

分类	名称	功能
变速运动	NV_AxisChangeSpeed	修改点位运动的速度
	NV_AxisChangeSpeedByRate	修改点位运动的速度的比例
	NV_AxisChangeSpeedAcc	修改点位运动的速度、加速度、减速度
	NV_AxisChangeSpeedAccByRate	修改点位运动的速度、加速度比例

重点说明

变速函数对单轴运动如定长、定速、JOG 功能生效。

例程

例程：请参考工程“P2PMoveSample”和 “VmoveSample”。

4.6.7 CSV 模式

函数列表

分类	名称	功能
CSV 模式	NV_AxisSetTargetVelocity	设置控制目标速度，需要切换到对应的模式
	NV_AxisGetTargetVelocity	获取设置的目标速度
	NV_AxisGetActualVelocity	获取实时速度

重点说明

CSV 模式（周期同步速度模式）下，控制卡将计算好的目标速度周期性同步地发送给伺服驱动器， 速度、转矩调节由伺服内部执行；伺服内部的调节方式及配置方式，请参考各家伺服的手册。

伺服驱动模式的切换有两种方法可以实现：通过 SDO 的方式，调用函数 NV_AxisSetBusMode 实现；通过 PDO 的方式，将参数 6060h 配置到使用的 PDO，由总线自动更新。

在用户的应用程序中，通过函数 NV_AxisSetTargetVelocity 设置目标速度；通过函数 NV_AxisGetActualVelocity 读取当前实际的速度。

例程

例程：设置轴到 CSV 模式，输出速度 300。

设置轴工作模式为 CSV：

```
ERROR_CODE_TYPE ret = 0;
short Value = 9; //工作模式 CSV 模式：9
ret = NV_AxisSetBusMod(m_nConnectNo, m_nAxisNo, Value);
```

设置轴输出的目标速度：

```
ERROR_CODE_TYPE ret = 0;
short Value = 300; //目标速度
ret = NV_AxisSetTargetVelocity(m_nConnectNo, m_nAxisNo, Value);
```

4.6.8 CST 模式

函数列表

分类	名称	功能
CST 模式	NV_AxisSetTargetTorque	设置控制目标转矩，需要切换到对应的模式
	NV_AxisGetTargetTorque	获取设置的目标转矩
	NV_AxisGetActualTorque	获取转矩
	NV_AxisSetMaxTorque	设置最大转矩
	NV_AxisGetMaxTorque	获取最大转矩

重点说明

CST 模式（周期同步转矩模式）下，控制卡将计算好的目标转矩周期性同步的发送给伺服驱动器，转矩调节由伺服内部执行；伺服内部的调节方式及配置方式，请参考各家伺服的手册。

伺服驱动模式的切换有两种方法可以实现：通过 SDO 的方式，调用函数 NV_AxisSetBusMode 实现；通过 PDO 的方式，将参数 6060h 配置到使用的 PDO，由总线自动更新。

在用户的应用程序中，通过函数 NV_AxisSetTargetTorque 设置目标转矩；通过函数 NV_AxisGetActualTorque 读取当前实际的转矩。

例程

例程：设置轴到 CST 模式，输出力矩 50。

设置轴工作模式为 CST：

```
ERROR_CODE_TYPE ret = 0;
short Value = 10; //工作模式 CST 模式：10
ret = NV_AxisSetBusMod(m_nConnectNo, m_nAxisNo, Value);
```

设置最大力矩限制：

```
ERROR_CODE_TYPE ret = 0;
short Value = 10; //设置最大力矩限制：2000
```

ret = NV_AxisSetMaxTorque(m_nConnectNo, m_nAxisNo, Value);

设置轴输出的目标力矩：

ERROR_CODE_TYPE ret = 0;

short Value = 50; //目标力矩

ret = NV_AxisSetTargetTorque(m_nConnectNo, m_nAxisNo, Value);

4.7 插补运动

插补运动主要应用于需要多轴联动的连续轨迹加工场合，如金属加工、点胶等行业。控制卡的插补运动模式主要具有以下功能：

(1) 可以实现直线、圆弧、螺旋线等插补运动。

(2) 具有前瞻预处理功能，能够实现小线段高速平滑的连续轨迹运动。

(3) 每个坐标系含有一个缓存区，可以实现缓存区暂停、恢复、倍率等功能。

(4) 支持插补运动平滑功能。

(5) 具有缓存区延时、缓冲区数字量输出、缓冲区单轴运动等功能。

4.7.1 插补配置

函数列表

名称	功能
NV_CrdSetPrm	设置插补坐标系参数
NV_CrdGetPrm	获取插补坐标系参数
NV_CrdSetPrfMax	设置插补坐标系运动限制参数
NV_CrdGetPrfMax	获取插补坐标系运动限制参数
NV_CrdSetProfile	设置插补坐标系运动参数
NV_CrdGetProfile	获取插补坐标系运动参数
NV_CrdSetLinkMode	设置插补坐标系拐角参数
NV_CrdGetLinkMode	获取插补坐标系拐角参数
NV_CrdSetOverride	设置插补倍率
NV_CrdGetOverride	获取插补倍率
NV_CrdStatus	获取插补坐标系运动状态

NV_CrdPosition	获取插补坐标系各轴位置
NV_CrdOpen	插补开启
NV_CrdClose	关闭连续插补缓冲区
NV_CrdStart	连续插补启动
NV_CrdPause	插补暂停
NV_CrdStop	插补停止
NV_CrdGetStopReason	获取插补系停止原因
NV_CrdClrStopReason	清除插补系停止原因

重点说明

（1）插补运动的基本步骤

实现插补运动的基本步骤如下：

a）建立插补坐标系；

b）设置前瞻参数以实现高速平滑的连续轨迹运动；

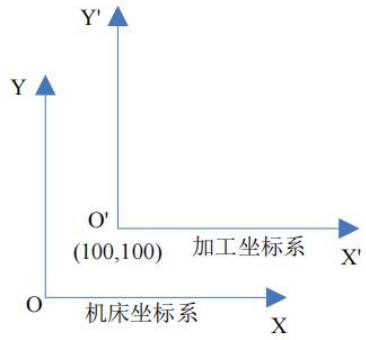
c）向插补缓存区压入直线、圆弧等插补运动指令；

d）启动运动，控制卡依次执行缓存区的插补数据，直到所有的插补数据全部运动完成。

a) 建立插补坐标系

在控制卡的初始状态下，复位之后或者还未使用过插补运动状态下，所有的规划轴都处于单轴 运动模式下，插补坐标系是无效的。所以，进行插补运动时，首先需要建立坐标系，将规划轴映射到相应的坐标系中。每个坐标系最多支持八轴，用户根据自己的需求，也可以使用二维(X-Y)、三维(X-Y-Z)坐标系描述运动轨迹。用户通过调用指令 NV_CrdSetPrm 将在坐标系内描述的运动通过映射关系映射到相应的规划轴上。运动控制卡根据坐标映射关系，控制各轴运动，实现要求的运动轨迹。调用指令 NV_CrdSetPrm 时，所映射的各规划轴必须处于静止状态。

建立坐标系示例： 建立一个二维坐标系，轴 1 对应为 x 轴，轴 2 对应为 y 轴，坐标系原点的规划位置是(100,100)，单位：unit，在此坐标系内运动的最大合成速度为 500unit/s，最大合成加速度和减速度为 5000unit/s2 ，如下图所示。



加工坐标系偏移量示意图

通过函数 NV_CrdSetPrm 建立坐标系，通过函数 NV_CrdSetProfile 设置运动参数。

b）初始化前瞻

前瞻初始化详细说明请参考“插补前瞻”内容。

c）往缓存区压入插补数据

控制卡插补运动模式采用缓存区运动方式，提供 5000 段的缓存区空间，用户需要向插补缓存区中传递插补数据，在启动插补运动后，控制卡会依次执行用户所传递的插补数据，直到所有的插补数据全部运动完成。

在插补段数据量比较大的时候，用户通过指令 NV_CrdStatus 查询插补缓存区的剩余空间，在有空间的时候再调用缓存区指令传递数据，如果插补缓存区已满，调用缓存区指令将会返回错误，说明该段插补数据没有输入成功，需要再次输入该段插补数据。

d）启动运动

当插补缓冲区已经有了一定量的插补数据时，就可以通过调用指令 NV_CrdStart 启动运动了。建议已经压入缓存区的插补数据足够运动一段时间再启动运动，如果数据段太少就启动，容易导致缓存区跑空。

（2）插补器的倍率

插补坐标系支持实时调整倍率，通过指令 NV_CrdSetOverride 配置。当倍率参数 VelRatio 值为 0 时，运动速度也将调整到 0，插补器处于暂停状态。

（3）插补器运动的暂停/停止

插补器有专用的暂停指令 NV_CrdPause 和停止指令 NV_CrdStop。

暂停以后，通过 NV_CrdStart 指令恢复运动。

（4）插补运动平滑

控制卡默认使用梯形速度规划模式，但在高加速度加工场合，由于加速度过大，很

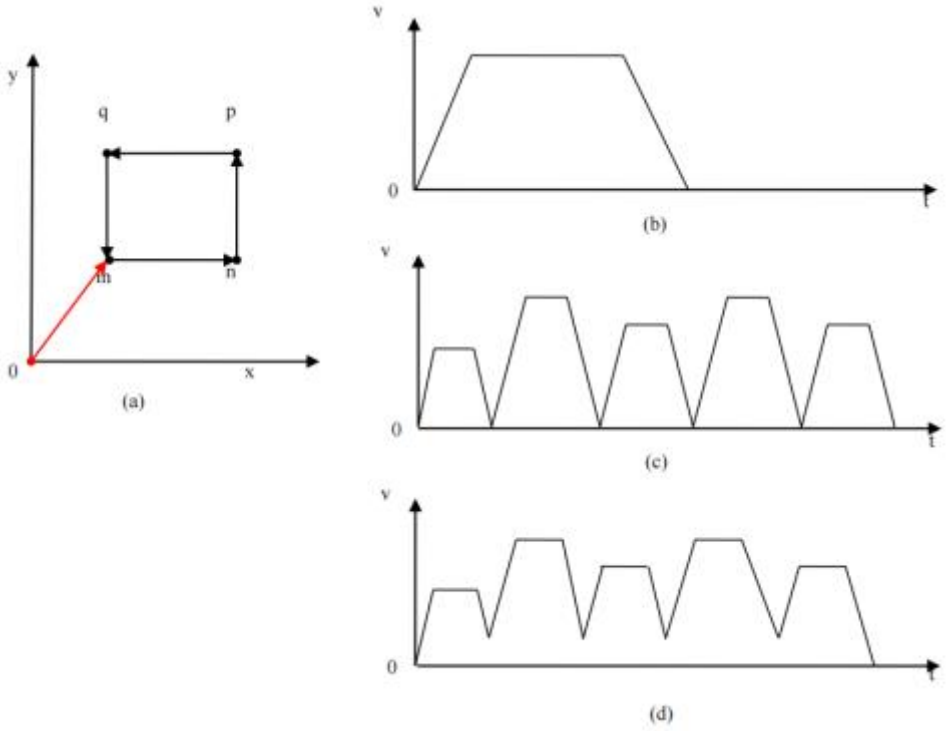
容易激发机台的振动，调用指令 CrdSetProfile 设置平滑时间，对坐标系运动的合成速度及合成加速度进行平滑，有效抑制机床的振动，并且不会影响轨迹精度。

在相同的加速度情况下，平滑时间越大，运动越平滑，效率越低，反之亦然。一般建议设置为 20~50ms。但由于机台的差别性，用户可以根据实际情况在参数允许范围内进行调整。

（5）插补前瞻模块（该功能 5000 系列卡才支持）

前瞻模块是一个预处理模块，通过前瞻多段轨迹信息，自动根据轨迹段的特征对插补运动速度进行优化和限制，以解决连续小线段轨迹加工时，由于频繁加减速或者不减速造成的加工效率低或者机械振动的问题，在保证运动平稳的前提下提高加工效率。

下面用一个实例来说明前瞻机制的优势，假设机床要加工一个长方形的零件，刀具所走的轨迹如下图（a）所示。



使用前瞻与不使用前瞻的速度规划区别示意图

如果按照图 (b)所示的速度规划，即在拐角处不减速，则加工精度一定会较低，而且可能在拐弯时对刀具和零件造成较大冲击。如果按照图 (c)所示的速度规划，即在拐角处减速为 0，可以最大限度保证加工精度，但加工速度就会慢下来。如果按照图 (d)所示的速度规划，在拐角处将速度减到一个合理值，既可以满足加工精度又能提高加工速度，就是一个好的速度规划。

为了实现类似图 (d)所示的好的速度规划，前瞻模块不仅要知道当前运动的位置参

数，还要提前知道后面若干段运动的位置参数，这就是所谓的前瞻。前瞻处理模块会依照用户设定的最大加速度值计算速度规划，使得在拐角处的速度不会超过允许的终点速度，防止对机械部分造成冲击，并且保障轨迹的误差。

控制卡中通过指令 NV_CrdSetLookAhead 开启前瞻功能，通过函数 NV_CrdSetLinkMode 设置拐角过渡连接方式和轨迹误差。

（5）坐标的运行状态

坐标系的状态通过函数 NV_CrdStatus 读取，其坐标系的运行状态值如下表：

状态值	状态	描述
0	LIST_IDLE	未使用，空闲状态
1	LIST_ACTIVE	打开 list,但没有 start
2	LIST_RUNNING	启动 list,正常运行状态
3	LIST_PAUSING	暂停中，正在减速到停止状态
4	LIST_PAUSED	已暂停，运动速度为 0
5	LIST_STOPPING	减速停止中
6	LIST_STOPPED	停止状态，下一个周期马上切换到 idle 状态

4.7.2 插补运动

函数列表

分类	名称	功能
直线插补	NV_CrdLnXY	两轴直线插补
	NV_CrdLnXYZ	三轴直线插补
	NV_CrdLnXYZA	四轴直线插补
	NV_CrdLnMove	多轴直线插补
圆弧插补	NV_CrdArcXYR	半径模式的 XY 平面圆弧插补
	NV_CrdArcXYC	圆心模式的 XY 平面圆弧插补
	NV_CrdArcYZR	半径模式的 YZ 平面圆弧插补
	NV_CrdArcYZC	圆心模式的 YZ 平面圆弧插补
	NV_CrdArcZXR	半径模式的 ZX 平面圆弧插补
	NV_CrdArcZXC	圆心模式的 ZX 平面圆弧插补
	NV_CrdArcXYZ	三点模式的空间圆弧插补
	NV_CrdHelixXYRZ	半径模式的 XY 平面螺旋线插补

	NV_CrdHelixXYCZ	圆心模式的 XY 平面螺旋线插补
	NV_CrdHelixYZRX	半径模式的 YZ 平面螺旋线插补
	NV_CrdHelixYZCX	圆心模式的 YZ 平面螺旋线插补
	NV_CrdHelixZXRY	半径模式的 ZX 平面螺旋线插补
	NV_CrdHelixZXCY	圆心模式的 ZX 平面螺旋线插补
	NV_CrdArcMoveC	圆心模式的多轴圆弧插补
	NV_CrdArcMoveR	半径模式的多轴圆弧插补
	NV_CrdArcMove3P	三点模式的多轴圆弧插补

重点说明

插补运动是为了实现轨迹控制，控制卡按照一定的控制策略控制多轴联动，使运动平台用微小直线段精确地逼近轨迹的理论曲线，保证运动平台从起点到终点上的所有轨迹点都控制在允许误差范围内。

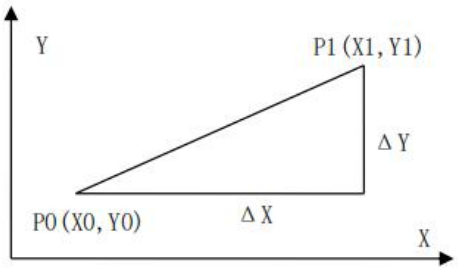
控制卡支持的插补模式包括：直线插补、平面圆弧插补、空间圆弧插补、螺旋线插补等。

(1) 直线插补

控制卡的插补模式支持在 2-8 轴的直线插补。用户输入直线的终点坐标、速度和加速度等，控制卡根据速度规划从起点沿着直线运动到终点。

两轴直线插补：如下图所示，2 轴直线插补从 P0 点运动至 P1 点，X、Y 轴同时启动，并同时到达终点；X、Y 轴的运动速度之比为ΔX：ΔY，二轴合成的矢量速度为：

$$\frac{\Delta P}{\Delta t}=\sqrt{\left(\frac{\Delta X}{\Delta t}\right)^2+\left(\frac{\Delta Y}{\Delta t}\right)^2}$$

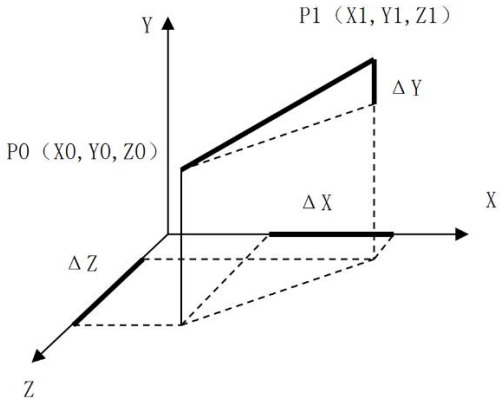


两轴直线插补

三轴直线插补：如下图所示，在 X、Y、Z 轴内直线插补，从 P0 点运动至 P1 点。

插补过程中 3 轴的速度比为ΔX：ΔY：ΔZ，三轴合成的矢量速度为：

$$\frac{\Delta P}{\Delta t}=\sqrt{\left(\frac{\Delta X}{\Delta t}\right)^2+\left(\frac{\Delta Y}{\Delta t}\right)^2+\left(\frac{\Delta Z}{\Delta t}\right)^2}$$



三轴直线插补

四轴直线插补：4 轴插补可以理解为在 4 维空间里的直线插补。一般情况是 3 个轴进行直线插补，另一个旋转轴也按照一定的比例关系和这条空间直线一起运动。其合成矢量速度为：

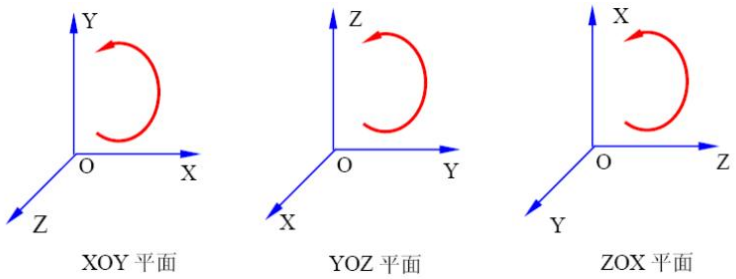
$$\frac{\Delta P}{\Delta t}=\sqrt{\left(\frac{\Delta X}{\Delta t}\right)^2+\left(\frac{\Delta Y}{\Delta t}\right)^2+\left(\frac{\Delta Z}{\Delta t}\right)^2+\left(\frac{\Delta U}{\Delta t}\right)^2}$$

调用直线插补函数时，用户需设置最大合成速度 MaxVel，如果该值为 0，将沿用上一段运动的最大速度。

（2）平面圆弧插补

控制卡的插补模式支持在 XY 平面、YZ 平面和 ZX 平面的圆弧插补。

圆弧插补的旋转方向按照右手螺旋定则定义为：从坐标平面的“上方”(即垂直于坐标平面的第三个轴的正方向)看，来确定逆时针方向和顺时针方向。可以这样简单记忆：将右手拇指前伸，其余四指握拳，拇指指向第三个轴的正方向，其余四指的方向即为逆时针方向。映射坐标系为二维坐标系(X-Y)时，XOY 坐标平面内的圆弧插补逆时针方向同样定义，如下图。

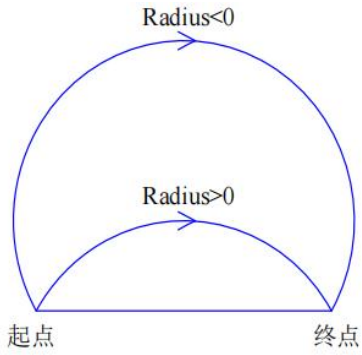


圆弧插补逆时针方向示意图

圆弧插补有两种描述方法：半径描述方法和圆心坐标描述方法，用户可以根据加工数据选择合适的描述方法来编程。所使用的描述方法遵循 G 代码的编程标准。

a) 半径描述方法

指令 NV_CrdArcXYR、NV_CrdArcYZR、NV_CrdArcZXR 是使用半径描述方法对圆弧进行描述。使用半径描述方法，用户需要输入圆弧终点坐标、圆弧半径、圆弧的旋转方向、坐标模式（相对坐标与绝对坐标）等。其中参数半径可为正值，也可为负值，其绝对值为圆弧的半径，正值表示圆弧的旋转角度≤180°，负值表示圆弧的旋转角度＞180°，如图 10-45 所示，半径描述方法无法描述 360°的整圆。



半径取正值/负值圆弧插补示意图

b) 圆心坐标描述方法

指令 NV_CrdArcXYC、NV_CrdArcYZC、NV_CrdArcZXC 是使用圆心坐标描述方法对圆弧进行描述。使用圆心描述方法，用户需要输入圆弧终点坐标、圆心位置、圆弧的旋转方向、坐标模式（相对坐标与绝对坐标）等。

（3）空间圆弧插补

控制卡的插补模式支持使用三点描述法描述空间圆弧插补。通过调用指令 NV_CrdArcXYZ，输入空间圆弧的终点坐标、中间点坐标（位于空间圆弧起点和终点之

间的某个点）、坐标模式（相对坐标与绝对坐标）等，根据起点、中间点和终点确定一个圆弧。

（4）螺旋线插补

控制卡的插补模式支持在 XY 平面、YZ 平面和 ZX 平面的螺旋线插补。螺旋线插补是平面圆弧插补的扩展，指的是在加工一个平面圆弧的同时，垂直于该平面的直线轴也进行线性插补运动。当圆弧从起点运动终点的同时，直线轴也由起点运动到终点。其中，螺旋线投影的圆弧的描述方式和平面圆弧插补的描述一致，同时输入直线轴的终点位置，即可描述一段螺旋线。

例程

例程 1：多轴直线插补，请参考工程“LinerMoveSample”。

插补器相关参数配置：

```
ERROR_CODE_TYPE ret = 0;
UInt16 Enable = 0;   //是否开启小线段前瞻功能：0：不开启
ret = NV_CrdSetLookAhead(m_nConnectNo, m_nCrdNo, Enable);
if (ret != 0)
{
MessageBox.Show($"NV_CrdSetLookAhead 调用报错：{ret}");
return;
}
UInt16 LinkMode = 0; //坐标系的拐角过渡连接方式:   0-直接过渡
Double PathError = 0;//坐标系的轨迹误差，单位 unit
ret = NV_CrdSetLinkMode(m_nConnectNo, m_nCrdNo, LinkMode, PathError);
if (ret != 0)
{
MessageBox.Show($"NV_CrdSetLinkMode 调用报错：{ret}");
return;
}
```

```
Double MaxVel = 2000;     //最大速度
Double MaxAcc = 10000;    //最大加速度
Double MaxDec = 10000;    //最大减速度
ret = NV_CrdSetPrfMax(m_nConnectNo, m_nCrdNo, MaxVel, MaxAcc, MaxDec);
if (ret != 0)
{
MessageBox.Show($"NV_CrdSetPrfMax 调用报错：{ret}");
return;
}
m_sProfile.StartSpeed = 10;// 起始速度
m_sProfile.MaxSpeed = 1000;// 最大速度
m_sProfile.StopSpeed = 10; // 停止速度
m_sProfile.Acceleraton = 2000; // 加速度
m_sProfile.Deceleraton = 2000; // 减速度
m_sProfile.SmoothTime = 0.3;     // 平滑时间，单位 s，范围 0-1
ret = NV_CrdSetProfile(m_nConnectNo, m_nCrdNo, m_sProfile);
if (ret != 0)
{
MessageBox.Show($"NV_CrdSetProfile 调用报错：{ret}");
return;
}
```

添加直线插补段：

```
double EndX, EndY, EndZ, EndA;
ushort PosMode;
uint SegNum;
```

EndX = 100;

EndY = 200;

EndZ = 300;

EndA = 100;

PosMode = 1; // 运动模式：1 - 相对坐标模式

SegNum = 0; //该段插补的段号；0 默认会自动递增

ret = NV_CrdLnXYZA(m_nConnectNo, m_nCrdNo, EndX, EndY, EndZ, EndA, PosMode, SegNum);

启动运动：

UInt16 RunMode = 1; //运行模式: 1-单步执行

ret = NV_CrdStart(m_nConnectNo, m_nCrdNo, RunMode);

if (ret != 0)

{

MessageBox.Show($"NV_CrdStart 调用报错：{ret}");

return;

}

停止运动：

ERROR_CODE_TYPE ret = 0;

ret = NV_CrdStop(m_nConnectNo, m_nCrdNo, 0);

暂停运动：

ERROR_CODE_TYPE ret = 0;

ret = NV_CrdPause(m_nConnectNo, m_nCrdNo);

例程 2：圆弧插补，请参考工程“LinerMoveSample”。

插补器相关参数配置同例程 1。

添加圆弧插补：

double EndX, EndY, Radius;

ushort ArcDir,PosMode;

uint SegNum;

int Cycles；

EndX = 100;

EndY = 200;

Radius = 350;

ArcDir = 0; // 圆弧的方向：0 - 顺时针方向

Cycles = 0; // 整圆的圈数: 0，没有完整的圈数

PosMode = 1; // 运动模式：1 - 相对坐标模式

SegNum = 0; //该段插补的段号；0 默认会自动递增

ret = NV_CrdArcXYR(m_nConnectNo, m_nCrdNo, EndX, EndY, Radius, ArcDir, Cycles, PosMode, SegNum);

例程 3：螺旋线插补，请参考工程“LinerMoveSample”。

插补器相关参数配置同例程 1。

添加螺旋线插补：

double EndX, EndY, Radius, EndZ ;

ushort ArcDir, PosMode;

uint SegNum;

int Cycles；

EndX = 100;

EndY = 200;

Radius = 350;

EndZ = 500;

ArcDir = 0; // 圆弧的方向：0 - 顺时针方向

Cycles = 5; // 整圆的圈数: 5

PosMode = 1; // 运动模式：1 - 相对坐标模式

SegNum = 0; //该段插补的段号；0 默认会自动递增

ret = NV_CrdHelixXYRZ(m_nConnectNo, m_nCrdNo, EndX, EndY, Radius, EndZ,

ArcDir, Cycles, PosMode, SegNum);

4.7.3 插补中其他操作

函数列表

名称	功能
NV_CrdBufDo	在插补中插入缓存按组 IO 输出
NV_CrdBufDoBit	在插补中插入缓存按位 IO 输出
NV_CrdBufDoBitDelayToStart	在插补中相对于轨迹起点 IO 滞后输出
NV_CrdBufDoBitDelayToStop	在插补中相对于轨迹终点 IO 滞后输出
NV_CrdBufDoBitAheadToStop	在插补中相对于轨迹终点 IO 提前输出
NV_CrdBufDoBitClearAction	在插补中清除段内未执行完的 IO
NV_CrdBufDoBitReverseOut	在插补中 IO 输出延时翻转
NV_CrdBufDoBitDelayOut	在插补中 IO 延时输出
NV_CrdDoSetPauseParam	配置在插补中暂停或者异常时 IO 的操作
NV_CrdDoGetPauseParam	读取在插补中暂停或者异常时 IO 的操作配置值
NV_CrdBufWaitIn	在插补中插入条件等待功能
NV_CrdBufSetDoBitPulse	在插补中插入输出 IO 波形功能
NV_CrdBufClrDoBitPulse	在插补中清除输出脉冲波形
NV_CrdBufDelay	在插补中插入延时等待功能
NV_CrdBufDA	在插补中插入 DA 输出功能
NV_CrdBufPWM	在插补中插入 PWM 输出功能
NV_CrdBufPWMAheadStop	在插补轨迹段终点 PWM 提前停止功能
NV_CrdBufPmove	在插补中启动坐标系外的轴进行定长运动
NV_CrdBufGear	在插补中跟随下一段轨迹运动一段距离
NV_CrdBufFollow	在插补中跟随下一段轨迹运动开始按比例跟随插补速度运动
NV_CrdBufVmove	在插补中开始定速运动
NV_CrdBufStop	在插补中停止坐标系外的轴

重点说明

（1）插补中 IO 控制

控制卡在插补运动中提供了 IO 操作功能。

函数 NV_CrdBufIO 可以在运动中设置输出，支持输出类型为普通输出和高速输出；通过该函数，只能将输出口输出高电平或者低电平操作，其他高级功能，如需要输出连续的 PWM 波形、高精度 OUT 控制，则需要其他函数才能实现。

对于输出操作，函数库中提供有相对于轨迹位置的提前/滞后操作，这些函数主要用于点胶类、激光类的工艺。

函数 NV_CrdBufWaitIn 用于读取输入 IN 信号，如果输入信号等于 DiValue 设置的状态，那么表示等待条件满足，运动继续往下执行；如果条件不满足，一直等待，直到时间超出 TimeOut；超时后，控制卡按照 Action 的配置，执行后续的响应。当函数 NV_CrdBufWaitIn 的 TimeOut 值设置为 0 时，运动控制卡将一直等待 IO 输入信号，超时时间为无限长。

运动过程中，遇到异常停止（如碰到 EMG 信号、限位信号）时，控制卡按照用户预先设置的异常停止处理。

当暂停、停止连续插补时，控制卡不会对 IO 执行操作

（2）插补延时等待

控制卡在插补运动中提供了延时等待功能，通过函数 NV_CrdBufDelay 配置等待时间。

（3）插补中输出 DA

控制卡在插补运动中提供了 DA 输出功能，通过函数 NV_CrdBufDA 配置。DA 输出值跟随合速度的变化而变化；DA 输出最大值和跟随速度最大值根据参数配置，DA 值与速度值为线性变换关系。

（4）插补中输出脉冲，可设置脉冲个数及高低电平时间

控制卡在插补运动中提供了脉冲输出功能，脉冲的输出特性通过 NV_CrdBufSetDoBitPulse 函数配置；支持输出的引脚为普通输出和高速输出口；当脉冲未输出完，需要终止输出是，调用 NV_CrdBufClrDoBitPulse 函数。

（5）插补中输出 PWM

控制卡在插补运动中提供了 PWM 输出功能，PWM 参数通过 NV_CrdBufPWM 函数配置；通过使能开关，可以打开和关闭输出；PWM 的参数可以实时修改，设置后即生效。

PWM 跟随模式通过 PwmMode 参数设置，0 表示按照 PWM 设置的占空比和频率输出；1 和 2 表示跟随速度的变化调整占空比或者频率；占空比或者频率的最大值与设置的最大速度成线性比列关系。

在轨迹的终点，可以提前关闭 PWM 输出，设置提前量的方式，可以设置时间或者距离，通过函数 NV_CrdBufPWMAheadStop 设置。

（6）插补中设置外部轴运动

控制卡在插补运动中提供了插补器外的轴做随动，随动方式支持定长距离、比例距离、比列速度、恒定速度。

通过 NV_CrdBufPmove 函数，在插补中启动坐标系外的另外一个轴进行定长运动；该轴有独立的运动位移、速度、加速等参数。

通过 NV_CrdBufGear 函数，启动坐标系外的另外一个轴移动一段距离 Distance；插补器运动的距离/速度与外部轴运动距离/速度为同步关系。

通过 NV_CrdBufFollow 函数，启动坐标系外的另外一个轴跟随插补器运动，速度跟随比列通过参数 Ratio 配置。

通过 NV_CrdBufVmove 函数，启动坐标系外的另外一个轴定速运动。

通过 NV_CrdBufStop，停止外部轴的运动。

例程

例程：插补中 OUT 控制，请参考工程“LinerMoveSample”。

配置插补运动请参考上一节例程。

插补中配置 OUT 口输出：

```
UInt16 DoType = 0; UInt16 DoIndex = 0;
UInt32 DoValue = 0; UInt32 DoMask = 0;
DoType = 0;   //输出类型：0-普通 OUT
DoIndex = 1; //输出口编号:1, 即 OUT1
DoValue = 1; //  输出的 IO 状态：  1
SegNum = 0; //该段插补的段号；0 默认会自动递增
ret = NV_CrdBufDoBit(m_nConnectNo, m_nCrdNo, DoType, DoIndex,
```

(ushort)DoValue, SegNum);

例程：插补中输出 PWM，请参考工程“LinerMoveSample”。

配置插补运动请参考上一节例程。

插补中配置 PWM 输出：

PwmNo = 0; //PWM 的通道序号

PwmEnable = 1; //PWM 的通道使能

PwmMode = 1;//PWM 的跟随模式: 2-跟随，频率自动调整

MaxVel = 2000; //最大合速度，单位：unit / s

Duty = 0.5; //占空比，范围：0-1

Frequancy = 10; //频率，范围：0 - 500KHz

SegNum = 0;

ret = NV_CrdBufPWM(m_nConnectNo, m_nCrdNo, PwmNo, PwmEnable, PwmMode, MaxVel, Duty, Frequancy, SegNum);

4.7.4 矩形插补

函数列表

名称	功能
NV_CrdRectangle	连续插补中矩形插补指令

重点说明

（1）支持两轴矩形插补运动，包括逐行模式与渐开线模式。

（2）矩形插补运动支持前瞻模式和非前瞻模式。前瞻模式矩形拐角会平滑处理变成圆弧，无法到达矩形端点；非前瞻模式则精确到达矩形端点。

（3）逐行模式的矩形插补运动。

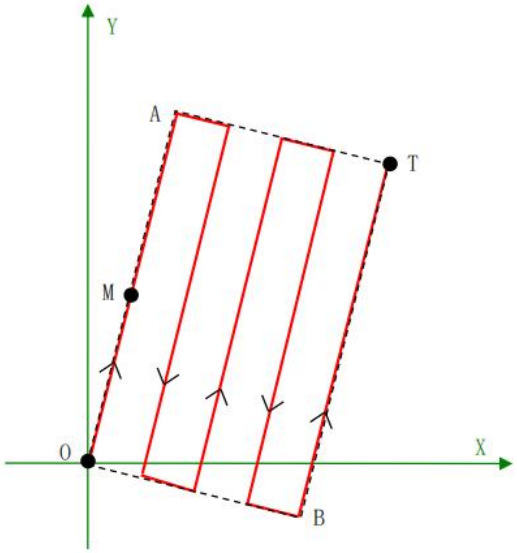
如下图所示，要进行一次逐行模式的矩形插补运动，必须确定以下三个参数：

（a）对角位置，如图中的 T 点，该点确定了矩形的对角点。

（b）矩形方向标记位置，如图中的 M 点。矩形方向标记位置确定了矩形插补运动

第一条边的运动方向，该方向为射线 OM 的方向。 此时，通过对角位置坐标及矩形边的方向，则可以确定该矩形区域的框架，如图中的矩形 OATB。

（c）行数，如图中矩形插补运动的行数为 5。通过行数参数，可以设置在指定矩形区域内进行矩形逐行插补运动的行数。



（4）渐开线模式的矩形插补运动。

如下图所示，要进行一次渐开线模式的矩形插补运动，必须确定以下三个参数：

（a）对角位置，如图中的 T 点，该点确定了矩形的对角点。

（b）矩形方向标记位置，如图中的 M 点。矩形方向标记位置确定了矩形插补运动第一条边的运动方向，该方向为射线 OM 的方向。此时，通过对角位置坐标及矩形边的方向，则可以确定该矩形区域的框架，如图中的矩形 OATB。

（c）圈数，如图中矩形插补运动的圈数为 3。通过圈数参数，可以设置在指定矩形区域内进行矩形区域渐开线插补运动的圈数。

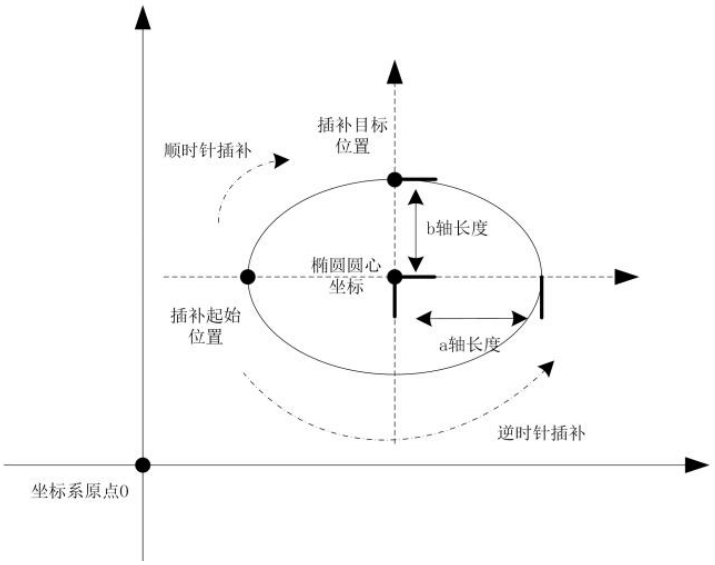
4.7.5 椭圆插补

函数列表

名称	功能
NV_CrdEllipse	启动椭圆插补运动

重点说明

（1）椭圆插补指令实现平面椭圆插补，如下图所示。需设置椭圆圆心坐标、a 轴长度、b 轴长度，以确定椭圆在坐标系中数学表达；设置插补目标位置，插补方向，以确定插补动作。



注意：

1）a 轴定义为平行于 x 轴的椭圆半轴长度，b 轴定义为平行于 y 轴的椭圆半轴长度；

2）插补起始位置为坐标系轴当前位置；

3）目前仅支持标准椭圆，即长轴和短轴与 XY 平行；

4.7.6 单段插补

函数列表

名称	功能
NV_LnMove	添加任意轴单段直线插补
NV_ArcMoveC	以圆心模式添加多轴单段圆弧运动
NV_CrdArcMoveR	以半径模式添加多轴单段圆弧运动
NV_ArcMove3P	以三点模式添加多轴单段圆弧运动

重点说明

（1）单段插补与前面开缓存区插补相比，差异在于：不用调用开缓存区函数，直

接调用插补函数，就可以执行插补运动。

4.8 补偿功能

4.8.2 反向间隙补偿

函数列表

名称	功能
NV_AxisSetBacklash	设置反向间隙功能
NV_AxisGetBacklash	获取反向间隙补偿功能状态

重点说明

反向间隙误差是指由于传动链中机械间隙的存在，执行部件在运动过程中从正向运动变为负向运动时，或者从负向运动变为正向运动时，执行部件的运动量与理论量存在误差，最后将反映为叠加至工件上的加工精度的误差。为了消除反向间隙误差，提高机器的加工精度和定位精度，控制卡提供了反向间隙误差补偿功能。

用户只要在初始化的时候，调用反向间隙误差补偿指令 NV_AxisSetBacklash 设置了相应的参数，反向间隙误差补偿功能将会生效；也可以通过该指令来关闭反向间隙误差补偿功能；每个周期的补偿量用户可以通过参数 CompChangeValue 配置。

补偿方向指的是，反向间隙误差补偿是沿正方向补偿还是沿负方向补偿。如果参数 CompDir 设置为 1，则只有电机从正方向转为负方向运动时，反向间隙补偿量生效，当电机向正方向运动时，反向间隙补偿量为 0；这种情况下，用户应该在回零之后，让工作台向正方向运动一定的距离，以保证正方向运动没有间隙存在。如果参数 CompDir 设置为-1，则只有电机从负方向转为正方向运动时，反向间隙补偿量生效，当电机向负方向运动时，反向间隙补偿量为 0；这种情况下，用户应该在回零之后，让工作台向负方向运动一定的距离，以保证负方向运动没有间隙存在。如果参数 CompDir 设置为 0，则电机往正/负方向运动时，反向间隙补偿量都生效。

反向间隙补偿量会直接叠加到控制卡的输出量上，当用户读取规划位置时，不会读到反向间隙补偿量。但是用户如果读取电机编码器的值，将会读到反向间隙的补偿量。

4.8.1 丝杠螺距补偿

函数列表

名称	功能
NV_AxisSetPitchErr	设置丝杆螺距补偿功能
NV_AxisGetPitchErr	获取丝杆螺距补偿功能状态

重点说明

螺距误差是指由于传动链中丝杆的制造误差或安装误差，导致电机旋转的角度和丝杆的直线位移成非线性关系，最终将导致实际和理论的直线位移存在误差。为了消除螺距误差，提高机器的加工精度，控制卡提供了螺距误差补偿功能。

使用螺距误差补偿注意事项：

（1）补偿值以表格形式来进行补偿，最大数量 1000 个；边界点不补偿；

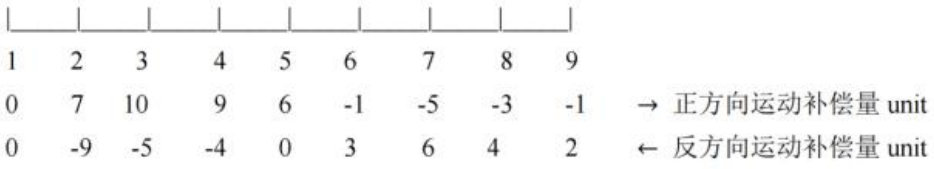
（2）正反两个方向有不同的补偿值；

（3）一般在回零完成之后使用该功能；

（4）读取位置有补偿前的原始位置和补偿后的位置；

（5）适用于所有运动指令；

螺距补偿数据表一般由激光干涉仪测量而来，假设有如下图所示 9 个位置（8 个段，每段必须保持等长距离进行测量），若 2~9 点经测量补偿数据如下图所示。



假设轴需补偿段的总长度为 8000unit，则每隔 1000unit 长度测量一次，起始点为 0，

往正方向运动时，各点需要补偿的 unit 为（0, 7, 10, 9, 6，-1，-5，-3, -1）；

往负方向运动时，各点需要补偿的 unit 为（0, -9，-5，-4，0, 3, 6, 4，2 ）；

轴往正方向运动时，将按照正方向运动补偿量进行补偿；往负方向运动时，将按照反方向运动补偿量进行补偿。

4.9 位置锁存功能

4.9.1 软件锁存

指令列表

名称	功能
NV_LatchSetMode	设置锁存功能参数
NV_LatchGetMode	获取锁存功能参数
NV_LatchSetSource	设置锁存的数值的来源
NV_LatchGetSource	获取锁存设定的数值的来源
NV_LatchClear	清除锁存数据
NV_LatchStatus	获取锁存的锁存状态
NV_LatchValue	获取锁存的数值

重点说明

控制卡支持 8 路软件锁存。

锁存的触发信号，支持多种，如通用输入、原点输入等，通过 NV_LatchSetMode 函数进行配置。

输入口类型编号	输入口类型
0	通用输入口
1	高速输入口
2	原点输入口
3	正限位输入口
4	负限位输入口
5	编码器 Z 相输入口

锁存的触发沿，支持多种方式：

锁存触发沿	说明
0	下降沿
1	上升沿
2	任意沿

锁存的数据对象有多种，如规划位置、编码器反馈位置、总线反馈位置、脉冲计数器位置、规划速度等，通过函数 NV_LatchSetSource 进行配置。

锁存数据源 SrcType 的类型：

锁存数据源	说明
0	编码器计数位置
1	轴指令位置（规划器位置）
2	轴反馈位置
3	脉冲计数器位置（内部参数）
10	轴指令速度
11	轴反馈速度
20	轴输入状态（排序：正限位，负限位，原点，急停、减速停，报警，ready，inp, ez）
21	轴输出状态（排序：伺服使能，伺服报警清除）

锁存器支持连续锁存，具有数据缓存，缓存区大小为 1024；当缓存区数据满后，数据会溢出，用户需要及时读取。

锁存执行完毕后，用户通过清除函数 NV_HSLatchClear，清除当前状态，回到初始状态。

例程

例程：软锁存，请参考工程“LatchSample”；

配置锁存模式：

```
ERROR_CODE_TYPE ret = 0;
UInt16 Mode = 0; UInt16 DiType = 0; UInt16 DiIndex = 0; UInt16 DiLogic = 0;
Mode = 2;//锁存方式：2-连续锁存
DiType = 0;   // 输入口类型: 0-DI_TYPE_COMMON
DiIndex = 2; //输入口编号:2
```

DiLogic = 1; //锁存的触发沿:1 - 上升沿

ret=NV_LatchSetMode(m_nConnectNo, m_nChannel,Mode, DiType, DiIndex, DiLogic);

配置锁存源（锁存源为编码器 0 的计数值）：

UInt16 SrcType = 0; UInt16 SrcIndex = 0;

SrcType = 0; //位置源类型:编码器计数位置,

SrcIndex = 0; //位置源编号:0

ret = NV_LatchSetSource(m_nConnectNo, m_nChannel, SrcType, SrcIndex);

4.9.2 高速锁存

函数列表

名称	功能
NV_HSLatchSetMode	设置锁存功能
NV_HSLatchGetMode	获取锁存功能状态
NV_HSLatchSetSource	设置锁存的数值的来源
NV_HSLatchGetSource	获取锁存设定的数值的来源
NV_HSLatchClear	清除锁存数据
NV_HSLatchStatus	获取锁存的状态
NV_HSLatchValue	获取锁存的数值

重点说明

控制卡支持 4 路高速锁存，高速锁存触发信号有三种，高速输入、原点输入、编码器 Z-index 信号；锁存通道与高速输入引脚的对应关系，通过函数 NV_HSLatchSetMode 进行配置。

触发信号类型	说明
1	高速输入
2	原点输入
5	编码器 Z 相输入

高速锁存的触发沿，支持多种方式：

锁存触发沿类型	说明
0	下降沿
1	上升沿
2	任意沿

高速锁存的信号源，支持两种：

信号源类型	说明
0	编码器计数位置
1	轴指令位置
2	轴反馈位置

高速锁存器支持连续锁存，具有数据缓存，缓存区大小为 1024；当缓存区数据满后，数据会溢出，用户需要及时读取。

锁存执行完毕后，用户通过清除函数 NV_HSLatchClear，清除当前状态，回到初始状态。

高速锁存操作：

（1）配置高速锁存参数，如锁存方式，锁存沿，触发引脚，通过指令 NV_HSLatchSetMode 配置；

（2）配置高速锁存源，通过指令 NV_HSLatchSetSource 设置；

（2）然后读取锁存状态。当有锁存值的时候，就可以读取锁存的位置；

（3）锁存执行完成后，需要通过清除函数 NV_HSLatchClear，清除当前状态，回到初始状态。

（4）配置完高速锁存，但是还没有执行完，需要取消，也是通过调用清除函数处理。

例程

例程 1：高速锁存 0 的使用，请参考工程“HSLatchSample”

配置高速锁存模式：

```
ERROR_CODE_TYPE ret = 0;
UInt16 Mode = 0; UInt16 DiType = 0; UInt16 DiIndex = 0; UInt16 DiLogic = 0;
Mode = 2; //高速锁存模式：2 - 连续锁存
DiType = 1; //输入口类型:1 高速输入
DiIndex = 0;//输入口编号 0
DiLogic = 0; //锁存的边沿，0 - 下降沿
m_nChannel = 0;
ret=NV_HSLatchSetMode(m_nConnectNo,m_nChannel,Mode,DiType,DiIndex,iLogic);
```

配置锁存源

```
UInt16 SrcType = 0; UInt16 SrcIndex = 0;
SrcType = 1;   //锁存源类型:1 编码器计数位置
SrcIndex = 0;//锁存源序号:0
ret = NV_HSLatchSetSource(m_nConnectNo, m_nChannel, SrcType, SrcIndex);
```

4.10 位置比较功能

4.10.1 软件一维位置比较

指令列表

名称	功能
NV_CmpSetEnable	设置软件一维位置比较器使能
NV_CmpGetEnable	读取软件一维位置比较器使能
NV_CmpSetSource	设置软件一维位置比较器
NV_CmpGetSource	读取软件一维位置比较器设置
NV_CmpClear	清除软件一维位置比较（比较点、比较状态等）
NV_CmpAddPoint	添加软件一维位置比较点
NV_CmpAddPoints	添加软件一维位置比较点（多个）
NV_CmpStatus	读取当前软件一维比较状态

重点说明

控制卡支持 12 个软件一维位置比较，每个比较器最多都可以添加 256 个比较点。软件一维位置比较的触发延时时间小于 1ms。

软件比较信号源 SrcType 的类型支持两种：

信号源类型	说明
0	编码器计数位置
1	轴指令位置
2	轴反馈位置

软件比较的比较模式，支持以下两种：

比较模式	说明
0	小于等于
1	大于等于

软件比较输出执行的动作，由指令 NV_CmpAddPoint 配置，参数 pAction 表示输出选择的功能，pActpara1 表示输出的对象，pActpara2 表示输出对象执行的动作，这三个参数组合如下

触发动作编号	功能描述	动作参数 1	动作参数 2
1	IO 置为高电平	IO 口编号	无
2	IO 置为低电平	IO 口编号	无
3	IO 反相输出	IO 口编号	无
4	输出高电平脉冲	IO 口编号	输出高电平时间 s
5	输出低电平脉冲	IO 口编号	输出低电平时间 s
10	停止指定轴	控制轴号	停止时间 s
11	在线变速	控制轴号	新的速度
12	在线变位	控制轴号	新的位置，绝对位置

例程

例程：启动一维比较 0，请参考工程“CmpSample”。

使能比较器：

```
ERROR_CODE_TYPE ret = 0;
UInt16 Enable = 1; //使能 1
m_nChannel = 0;
ret = NV_CmpSetEnable(m_nConnectNo, m_nChannel, Enable);
```

配置比较源：

```
UInt16 pSrcType = 0;
UInt16 pSrcIndex = 0;
pSrcType = 0;     //位置源类型:0 编码器反馈位置
pSrcIndex = 0;    //位置源编号:0
ret = NV_CmpSetSource(m_nConnectNo, m_nChannel, pSrcType, pSrcIndex);
```

添加比较点：

```
ERROR_CODE_TYPE ret = 0;
UInt16 Mode = 1; //比较模式:1 大于等于
Double Pos = 500;//比较位置：500
UInt16 Action = 4;     //比较点触发功能编号: 输出高电平脉冲
UInt32 Actpara1 = 5; //比较点触发功能参数 1: 输出 IO 口编号为 5，即 OUT5
Double Actpara2 = 0.3;//比较点触发功能参数 2：输出高电平时间 0.3s
ret = NV_CmpAddPoint(m_nConnectNo, m_nChannel, Mode, Pos, Action, Actpara1, Actpara2);
```

清空比较器：

```
ERROR_CODE_TYPE ret = 0;
ret = NV_CmpClear(m_nConnectNo, m_nChannel);
```

4.10.2 软件二维位置比较

指令列表

名称	功能
NV_Cmp2DSetEnable	设置软件二维位置比较器使能
NV_Cmp2DGeEnable	读取软件二维位置比较器使能
NV_Cmp2DSetSource	设置软件二维位置比较器
NV_Cmp2DGeSource	读取软件二维位置比较器设置
NV_Cmp2DClear	清除软件二维位置比较（比较点、比较状态等）
NV_Cmp2DAddPoint	添加软件二维位置比较点
NV_Cmp2DAddPoints	添加软件二维位置比较点（多个）
NV_Cmp2DStatus	读取当前软件二维比较状态

重点说明

控制卡支持 2 个软件二维位置比较，每个比较器最多都可以添加 256 个比较点。软件二维位置比较的触发延时时间小于 1ms。

软件比较信号源 SrcType 的类型支持两种：

信号源类型	说明
0	编码器计数位置
1	轴指令位置
2	轴反馈位置

软件比较的比较模式，支持以下两种：

比较模式	说明
0	小于等于
1	大于等于

软件比较输出执行的动作，由指令 NV_Cmp2DAddPoint 配置，参数 Action 表示输出选择的功能，Actpara1 表示输出的对象，Actpara2 表示输出对象执行的动作，这三个

参数组合如下:

Action	功能	Actpara1	Actpara2
1	IO 置为高电平	IO 号	无
2	IO 置为低电平	IO 号	无
3	取反 IO	IO 号	无
4	输出高电平脉冲	IO 号	输出高电平时间 s
5	输出低电平脉冲	IO 号	输出低电平时间 s
10	停止指定轴	轴号	停止时间 s
11	在线变速	轴号	新的速度
12	在线变位	轴号	新的位置（绝对位置）

例程

例程：二维比较使用，请参考工程“Cmp2DSample”。

二维比较使能：

```
ERROR_CODE_TYPE ret = 0;
UInt16 Enable = 1; //使能 1
ret = NV_Cmp2DSetEnable(m_nConnectNo, m_nChannel, Enable);
```

配置比较源：

```
UInt16[] pSrcType = new ushort[2];
UInt16[] pSrcIndex = new ushort[2];
pSrcType[0] = 0;//  位置源类型: 0 编码器反馈位置
pSrcIndex[0] = 0;//位置源编号:0
pSrcType[1] = 0;//  位置源类型: 0 编码器反馈位置
pSrcIndex[1] = 1;//位置源编号:1
ret = NV_Cmp2DSetSource(m_nConnectNo, m_nChannel, pSrcType, pSrcIndex);
```

添加比较点：

```
ERROR_CODE_TYPE ret = 0;
UInt16[] Mode = new UInt16[2];
```

Double[] Pos = new Double[2];

Mode[0] = 1; //比较模式:1 大于等于

Pos[0] = 400;//比较位置：400

Mode[1] = 1; //比较模式:1 大于等于

Pos[1] = 550;//比较位置：550

UInt16 Action = 5;//比较点触发功能编号: 输出低电平脉冲

UInt32 Actpara1 = 6; //比较点触发功能参数 1: 输出 IO 口编号为 6，即 OUT6

Double Actpara2 = 0.1;//比较点触发功能参数 2：输出低电平时间 0.1s

ret = NV_Cmp2DAddPoint(m_nConnectNo, m_nChannel, Mode, Pos, Action, Actpara1, Actpara2);

4.10.3 高速一维比较

函数列表

名称	功能
NV_HSCmpSetMode	设置高速一维比较模式
NV_HSCmpGetMode	读取高速一维比较模式
NV_HSCmpSetSource	设置高速一维比较位置源
NV_HSCmpGetSource	读取高速一维比较位置源
NV_HSCmpSetOuputMode	设置高速一维比较输出模式
NV_HSCmpGetOuputMode	读取高速一维比较输出模式
NV_HSCmpSetOutPulse	配置高速一维比较输出信号及波形
NV_HSCmpGetOutPulse	读取高速一维比较输出信号及波形
NV_HSCmpStartPulse	启动比较器的脉冲输出
NV_HSCmpStopPulse	停止比较器的脉冲输出
NV_HSCmpClear	清除已添加的所有高速位置比较点
NV_HSCmpSetPoint	设置高速一维比较位置
NV_HSCmpGetPoint	读取高速一维比较位置

NV_HSCmpAddPoints	设置高速一维比较的队列先入先出比较
NV_HSCmpSetLinear	设置高速一维比较线性模式参数
NV_HSCmpGetLinear	读取高速一维比较线性模式参数设置
NV_HSCmpSetOffset	设置高速一维比较比较点的整体偏移位置值
NV_HSCmpGetOffset	读取高速一维比较比较点的整体偏移位置值
NV_HSCmpTableStart	高速一维比较列表比较启动
NV_HSCmpStatus	获取高速一维比较的比较器的状态

重点说明

控制卡提供 4 路高速比较输出功能，可以用于控制相机拍照，即高速飞拍功能。

每个比较器均可配置 1 个高速输出接口，高速输出端口编号可从 HO0-7 中配置（不同型号控制卡有差异，请查阅相应的用户手册）。

比较器支持多种比较模式：小于等于、大于等于、队列、线性，用户可通过指令 NV_HSCmpSetMode 设置；

比较器的模式	说明
0	禁止
1	小于等于
2	大于等于
3	线性（FIFO）
4	队列

对于单次比较，支持小于等于、大于等于两种模式；

对于队列模式，提供 127 个比较点，遵循先添加先比较的原则，比较完后可追加比较点，也可一次性添加多个比较点；在函数库中，支持队列数据缓存，缓存区支持 25000 个点，用户可以在执行比较过程中实时检测缓存区的空余空间，当缓存区有空余空间时，可以往缓存区添加数据。

线性模式，适用于每个比较点位置都是按照线性增量依次增加的情况，用户只需要设置起始比较点位置以及线性增量值即可，最大支持 65535 次比较。

比较器支持用户配置位置源，支持两种位置源：

比较的位置源类型	说明
0	编码器计数位置
1	轴指令位置
2	轴反馈位置

比较器输出，支持输出电平或者脉冲，通过指令 NV_HSCmpSetOuputMode 进行配置；将 DoMode 设置为 0，输出口输出电平，将 DoMode 设置为 1，输出口输出脉冲，脉冲的高低电平和持续时间通过 DoLogic 和 DoTime 设置；如果需要输出多个 PWM 波，通过 NV_HSCmpSetOutPulse 指令设置。

在执行高速比较过程中，为了方便调试，在比较输出端口没有输出信号的情况下，通过函数 NV_HSCmpStartPulse 可以强制输出脉冲，脉冲特性由函数 NV_HSCmpSetOutPulse 设置的参数决定，函数 NV_HSCmpStopPulse 可以停止脉冲输出。

位置比较时间间隔最小可达几微秒，每个比较器的位置比较都是独立进行的；在队列及线性比较模式中，执行位置比较时，每个比较点的触发是按照添加的比较点顺序执行，如果有一个比较点没有被触发比较动作，那么后面的比较点都将不会被触发。

支持比较位置整体偏移，通过指令 HSCmpSetOffset 设置偏移值。如采用队列模式，将两个比较点 500 和 1000，已经配置到比较器中，通过 HSCmpSetOffset 设置了偏移值 50，那么实际比较位置将会是 550 和 1050；如采用线性模式，线性比较初始位置为 400，间隔 100，比较次数为 3 次，那么比较位置分别为 400,500,600，通过 HSCmpSetOffset 函数设置了偏移值 50 后，最终实际比较位置为 450,550,650。

通过清除函数，可以清除比较器比较点、比较器状态及设置的参数。

单次高速比较操作：

（1）通过函数 NV_HSCmpSetMode 配置比较模式；

（2）添加比较点

（3）读取比较器状态，确保比较完成。

队列模式和线性模式高速比较操作：

（1）通过函数 NV_HSCmpSetMode 配置比较模式；

（2）设置输出端口的电平状态及电平持续时间；

（3）添加比较点；

（4）读取比较器状态，如果缓存区有空余空间，继续添加比较点；如果不添加新的点，等待确保比较完成。

例程

例程 1：高速比较使用（单点比较），请参考工程“HSCmpSample”。

高速比较 0 配置模式：

```
ERROR_CODE_TYPE ret = 0;
UInt16 Mode = 0; UInt16 DoIndex = 0;
Mode = 2; //比较模式:2：大于等于
DoIndex = 1;  //高速比较输出口：1
m_nChannel = 0;
ret = NV_HSCmpSetMode(m_nConnectNo, m_nChannel, Mode, DoIndex);
```

配置比较源：

```
UInt16 pSrcType = 0;
UInt16 pSrcIndex = 0;
pSrcType = 0; //比较位置源:0 编码器计数位置
pSrcIndex = 0;  //比较源编号：0
ret = NV_HSCmpSetSource(m_nConnectNo, m_nChannel, pSrcType, pSrcIndex);
```

配置输出模式：

```
UInt16 DoMode = 0; UInt16 DoLogic = 0; Double DoTime = 0;
DoMode = 1;   //输出模式:1 - 输出 PWM 波形
DoLogic = 0;  //输出有效电平:0 - 低电平
DoTime = 1000;//输出脉冲宽度，单位：us，取值范围：1us - 134s
ret = NV_HSCmpSetOutputMode(m_nConnectNo, m_nChannel, DoMode, DoLogic, DoTime);
```

配置输出的脉冲：

UInt16 FirstLogic = 0; Double FirstTime = 0;

Double SecondTime = 0; UInt32 PulseNum = 0;

FirstLogic = 1; //比较生效时的起始电平:1 – 起始为高电平脉冲

FirstTime = 0; //比较生效时的起始电平维持时间

SecondTime = 8000; //比较生效时的相反电平维持时间

PulseNum = 2; //比较生效时的发生的波形个数:2

ret = NV_HSCmpSetOutPulse(m_nConnectNo, m_nChannel, FirstLogic, FirstTime, SecondTime, PulseNum);

添加比较点：

Double Value = 500; //添加比较点

ret = NV_HSCmpSetPoint(m_nConnectNo, m_nChannel, Value);

例程 2：高速比较使用（线性比较），请参考工程“HSCmpSample”。

高速比较 0 配置模式：

ERROR_CODE_TYPE ret = 0;

UInt16 Mode = 0; UInt16 DoIndex = 0;

Mode = 3; //比较模式: 3：线性比较

DoIndex = 1; //高速比较输出口：1

m_nChannel = 0;

ret = NV_HSCmpSetMode(m_nConnectNo, m_nChannel, Mode, DoIndex);

配置比较源和输出方式，如同上面的配置。

添加比较点：

Double StartPos = 0; Double Interval = 0; UInt32 Count = 0;

StartPos = 600; //线性比较起始位置:600

Interval = 400; //线性比较的间隔:400

Count = 5;//线性比较次数:5

ret = NV_HSCmpSetLinear(m_nConnectNo, m_nChannel, StartPos, Interval, Count);

4.10.4 高速二维位置比较

指令列表

名称	功能
NV_HSCmp2DSetMode	设置高速二维比较模式
NV_HSCmp2DGetMode	读取高速二维比较模式设置
NV_HSCmp2DSetPSOPara	设置高速二维 PSO 输出模式参数
NV_HSCmp2DGetPSOPara	读取高速二维 PSO 输出模式参数
NV_HSCmp2DSetSource	设置高速二维比较位置源
NV_HSCmp2DGetSource	读取高速二维比较器设置的位置源
NV_HSCmp2DSetErrorBand	设置高速二维比较器误差带
NV_HSCmp2DGetErrorBand	读取高速二维比较器误差带
NV_HSCmp2DSetOuputMode	设置高速二维比较器输出模式
NV_HSCmp2DGetOuputMode	读取高速二维比较器输出模式
NV_HSCmp2DSetOutPulse	设置高速二维比较器的输出信号及波形
NV_HSCmp2DGetOutPulse	读取高速二维比较器的输出信号及波形
NV_HSCmp2DStartPulse	强制启动比较器的脉冲输出
NV_HSCmp2DStopPulse	停止比较器的脉冲输出
NV_HSCmp2DClearPoints	清除已添加的所有高速位置比较点
NV_HSCmp2DAddPoints	设置高速二维比较器的队列先入先出比较点
NV_HSCmp2DStatus	读取高速二维比较器的状态

重点说明

控制卡提供 2 路高速二维比较输出功能。

每个比较器均可配置 1 个高速输出接口，高速输出端口编号可从 HO0-7 中配置（不同型号控制卡有差异，请查阅相应的用户手册）。。

比较器支持四种模式，如下表：

比较器的模式	说明

0	禁止
1	进入误差带触发
2	离开误差带触发
3	PSO 模式，立即触发
4	PSO 模式，进入误差带触发

误差带，通过指令 NV_HSCmp2DSetErrorBand 设置。

比较器支持三种位置源：

比较的位置源类型	说明
0	编码器计数位置
1	轴指令位置
2	轴反馈位置

高速二维比较器支持队列模式添加位置比较点，一次可以添加多个位置点，支持队列数据缓存，缓存区支持 1000 个点，用户可以在执行比较过程中实时检测缓存区的空余空间，当缓存区有空余空间时，可以往缓存区添加数据。

比较器输出，支持输出电平或者脉冲，通过指令 NV_HSCmp2DSetOuputMode 进行配置；将 DoMode 设置为 0，输出口输出电平，将 DoMode 设置为 1，输出口输出脉冲，脉冲的高低电平和持续时间通过 DoLogic 和 DoTime 设置；如果需要输出多个 PWM 波，通过 NV_ HSCmp2DSetOutPulse 指令设置。

位置比较时间间隔最小可达几微秒，每个比较器的位置比较都是独立进行的；在执行位置比较时，每个比较点的触发是按照添加的比较点顺序执行，如果有一个比较点没有被触发比较动作，那么后面的比较点都将不会被触发。

通过清除函数，可以清除比较器比较点、比较器状态及设置的参数。

高速二维比较操作：

（1）配置比较模式和比较位置源；

（2）通过函数 NV_HSCmp2DSetErrorBand 配置误差带；

（3）通过函数 NV_HSCmp2DSetOuputMode 配置输出方式；

（4）添加比较点

（4）读取比较器状态，确保比较完成。

PSO 输出操作：

（1）配置比较模式和比较位置源；

（2）通过函数 NV_HSCmp2D SetPSOPara 配置 PSO 输出，模式参数；

例程

例程：高速二维比较使用，请参考工程“HSCmp2DSample”。

配置高速二维比较模式：

```
ERROR_CODE_TYPE ret = 0;
UInt16 Mode = 0; UInt16 DoIndex = 0;
Mode = 1;//比较模式：1：进入误差带后触发
DoIndex = 1; //高速输出口编号:1
ret = NV_HSCmp2DSetMode(m_nConnectNo, m_nChannel, Mode, DoIndex);
```

配置比较源：

```
UInt16[] pSrcType = new ushort[2];
UInt16[] pSrcIndex = new ushort[2];
pSrcType[0] = 0;//比较位置源：0：编码器计数器
pSrcIndex[0] = 0;     //比较源编号:0
pSrcType[1] = 0;//比较位置源：0：编码器计数器
pSrcIndex[1] = 1;     //比较源编号:1
ret = NV_HSCmp2DSetSource(m_nConnectNo, m_nChannel, pSrcType, pSrcIndex);
```

配置误差带：

```
Double[] pErrorBand = new double[2];
pErrorBand[0] = 20;   //比较位置误差带:20
```

pErrorBand[1] = 20; //比较位置误差带:20

ret = NV_HSCmp2DSetErrorBand(m_nConnectNo, m_nChannel, pErrorBand);

配置输出模式：

UInt16 DoMode = 0; UInt16 DoLogic = 0; Double DoTime = 0;

DoMode = 1; //输出模式:1 - 输出 PWM 波形

DoLogic = 0;//输出有效电平:0 - 低电平

DoTime = 1200;//输出脉冲宽度，单位：us，取值范围：1us - 134s

ret = NV_HSCmp2DSetOutputMode(m_nConnectNo, m_nChannel, DoMode, DoLogic, DoTime);

配置输出的脉冲：

UInt16 FirstLogic = 0; Double FirstTime = 0;

Double SecondTime = 0; UInt32 PulseNum = 0;

FirstLogic = 1; //比较生效时的起始电平:1 – 起始为高电平脉冲

FirstTime = 0;//比较生效时的起始电平维持时间

SecondTime = 500; //比较生效时的相反电平维持时间

PulseNum = 2; //比较生效时的发生的波形个数:2

ret = NV_HSCmp2DSetOutPulse(m_nConnectNo, m_nChannel, FirstLogic, FirstTime, SecondTime, PulseNum);

添加比较点：

Double[] pValue = new Double[50];

Double[] pValue2 = new Double[50];

UInt16 Count = 0;

pValue[0] = 500;

pValue2[0] = 600;

```
pValue[1] = 800;
pValue2[1] = 1000;
pValue[1] = 1100;
pValue2[1] = 1400;
Count = 3;
ret = NV_HSCmp2DAddPoints(m_nConnectNo, m_nChannel, pValue, pValue2, Count);
```

清除比较器

```
ERROR_CODE_TYPE ret = 0;
ret = NV_HSCmp2DClear(m_nConnectNo, m_nChannel);
```

4.11 EtherCAT 总线操作

4.11.1 总线配置

函数列表

名称	功能
NV_EcatReset	复位 EtherCAT 总线
NV_EcatSetCycleTime	设置 EtherCAT 总线循环周期
NV_EcatGetCycleTime	读取 EtherCAT 总线循环周期
NV_EcatGetSlaveNum	获取 EtherCAT 从站总数
NV_EcatGetLostHeartbeatNodes	获取心跳报文信息
NV_EcatGetErrcode	获取总线错误码
NV_EcatClrErrcode	清除总线错误码
NV_EcatGetConsumeTime	读取 EtherCAT 总线平均周期时间、最大周期时间、执行周期数

重点说明

（1）总线初始化

控制卡支持 EtherCAT 总线通讯，通过总线可以扩驱动、模块等符合 ETG 标准的设

备。EtherCAT 总线驱动器线缆连接好之后，使用前需要进行总线初始化操作，并使用指令映射驱动器的轴号、打开驱动器使能等操作。

EtherCAT 总线相关的基本参数

（a）主站编号：控制卡有可能支持多路 EtherCAT 主站，编号从 0 开始。

（b）从站编号：默认根据 EtherCAT 设备接线顺序排列；在 EtherCAT 配置中，可以为每个设备设置从站编号。

（c）轴号：控制卡为每个 EtherCAT 轴配置的轴号；总线轴分配了轴号后，就可以将它当做本地轴使用。

（d）总线周期：默认 1000us；在 EtherCAT 配置中，用户可以配置总线的周期。

总线配置及初始化：

（a）通过配置工具 Motion 配置总线，并将配置文件下载到控制卡内；

（b）启动总线后，实时扫描总线状态，判断是否有错误发生；如果有错误，通过函数 NV_EcatGetErrcode 读取总线错误，并排除故障，然后再重新初始化。

（c）总线初始化过程中，总线驱动的状态机按照“初始化—>预运行—>安全运行—>运行”进行切换。

（d）总线初始化成功后，用户可使用 NV_EcatGetSlaveNum 指令获取当前所有从站个数。

（2）总线信息读取

在总线运行过程中，可以实时读取总线的错误码，查看总线是否出现错误；如果出现故障，排除故障后，需要调用 NV_EcatClrErrcode 函数，清除总线故障码。

在总线运行过程中，可以监听从站的心跳报文和紧急报文，时刻监控从站的在线状态。

通过函数 NV_EcatGetConsumeTime 可以读取总线执行周期数、平均周期时间、最大周期时间，用于帮助用户判断总线任务消耗的时间。原则上读出的“最大周期时间”不能大于配置的总线周期。

4.11.2 总线轴和模块操作

函数列表

分类	名称	功能
总线轴	NV_AxisGetErrcode	获取总线轴错误码
	NV_AxisClrErrcode	清除总线轴的错误码
总线模块	NV_EcatGetSlaveIoNum	获取总线节点的 IO 数量
	NV_EcatGetDiBit	获取总线节点的单个输入状态
	NV_EcatGetDi	获取总线节点的单组输入状态
	NV_EcatSetDoBit	设置总线节点的单个输出电平
	NV_EcatGetDoBit	获取总线节点的单个输出电平
	NV_EcatSetDo	设置总线节点的一组输出电平
	NV_EcatGetDo	获取总线节点的一组输出电平

重点说明

（1）驱动上自带的轴 IO 信号，通过 NV_AxisIoGetDi 和 NV_AxisIoSetDo 读取和设置；

驱动上自带的通用 IO 信号的读写操作，有两种方式：（1）在配置总线的时候，将其配置到 PDO 映射上；（2）通过操作从站节点的对象字典方式。

总线轴报错，通过函数 NV_EcatGetAxisErrcode 读取错误码，通过函数 NV_EcatClrAxisErrcode 清除轴错误。

（2）在使用 EtherCAT IO 模块、AD/DA 模块时，有专用函数读取和设置模块的输入输出信息；操作上按照模块的节点号（从站号）读取和设置输入输出。

4.11.3 对象字典和 PDO 操作

函数列表

名称	功能
NV_EcatSetSDO	设置总线节点的对象字典
NV_EcatGetSDO	获取总线节点的对象字典
NV_EcatSetPDO	设置总线节点的过程数据对象字典

NV_EcatGetPDO	获取总线节点的过程数据对象字典

重点说明

（1）控制卡通过 SDO 可以对从站的对象字典进行读写操作。SDO 的数据交互使 Mailbox 通信，因而数据交互的效率大大低于 PDO，需要多个总线周期。

（2）控制卡支持用户直接操作（读取和写入）PDO 数据，访问方式如同访问从站的对象字典（函数 NV_EcatGetSDO 从从站的对象字典中读取数据，NV_EcatGetPDO 从 PDO 对应参数位置中读取数据）。

4.12 PWM 功能

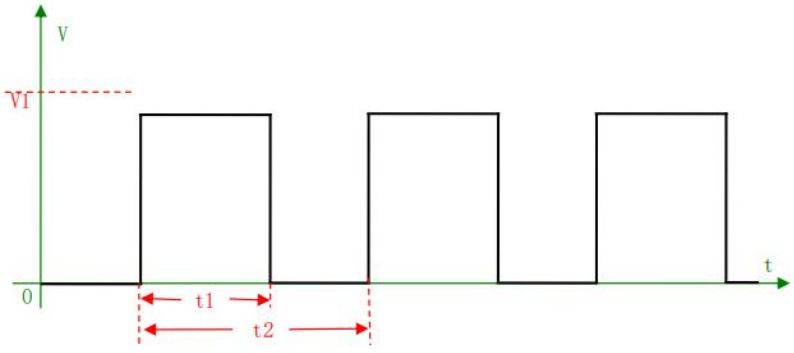
指令列表

名称	功能
NV_PWMSetEnable	设置 PWM 使能。
NV_PWMGetEnable	读取 PWM 使能
NV_PWMSetOutput	设置 PWM 输出
NV_PWMGetOutput	获取 PWM 输出

重点说明

控制卡支持 4 路 PWM 输出，输出通道复用高速输出端口，电平幅值为 24V。

PWM 波形如下图，周期为 t2（频率即为 1/t2），占空比为 t1/t2。通过指令 NV_PWMSetOutput 配置好 PWM 通道、频率、占空比，并通过指令 NV_PWMSetEnable 使能以后，对应的高速输出端口持续输出 PWM 波；将使能关闭，PWM 停止输出。



例程

例程：PWM 设置及输出，请参考工程“PWMSample”。

PWM 设置及使能：

```
ERROR_CODE_TYPE ret = 0;
UInt16 Enable = 1;  // 使能值： 1 - 使能
UInt16 DoIndex = 1; //高速输出口：1
Double Duty = 0.4; //占空比 0-1，0.4
Double Fre = 100;  //频率：1000hz
ret = NV_PWMSetOutput(m_nConnectNo, m_nChannel, Duty, Fre);
ret = NV_PWMSetEnable(m_nConnectNo, m_nChannel, Enable, DoIndex);
```

4.13 PVT 运动

指令列表

分类	名称	功能
PVT 数据表配置	NV_PvtAddTable	添加 PVT 列表数据
	NV_PtsAddTable	添加 PTS 列表数据
	NV_PvtsAddTable	添加 PVTS 列表数据
	NV_PttAddTable	添加 PTT 列表数据
PVT 运动指令	NV_PvtStart	同时启动多轴 PVT
	NV_PvtStatus	获取当前 PVT 状态

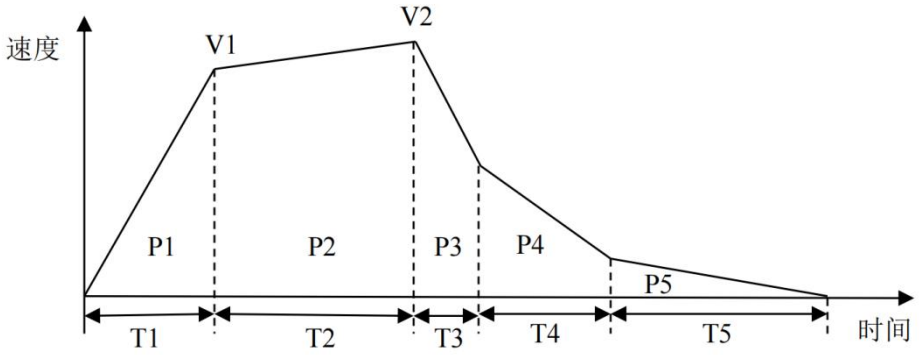
重点说明

控制卡共有四种 PVT 模式，分别为 PTT、PTS、PVT、PVTS 模式。其中 PTT、PTS 运动模式用于单轴速度规划；PVT、PVTS 运动模式则用于多轴轨迹规划。用户可以根据实际需求选择适合的 PVT 模式。

（1）PTT 运动模式

PTT 模式中第一个字母 P 表示位置（Position）、第二个字母 T 表示时间（Time），最后一个字母 T 表示梯形（Trapezoid）；PTT 模式表示在梯形速度曲线下规划点位运动。用户通过输入一系列位置和时间参数，自定义单轴的运动规律。

用户需要将待规划的速度曲线分割成若干段来进行描述，如下图所示，整个速度曲线被分割成 5 段。



PTT 运动速度曲线

第 1 段起点速度为 0；经过时间 T1 后，位移量为 P1，即速度曲线所围面积。因此第 1 段的终点速度为 V1=2×P1/T1；

第 2 段起点速度为 V1，经过时间 T2 后，位移量为 P2，即速度曲线所围面积。因此第 2 段的终点速度为 V2=2×P2/T2-V1；

第 3、4、5 段依此类推。

注意：在定义一段完整的 PTT 运动时，第 1 段的起点位置和时间被设为 0；各段的终点位置和时间都是相对于该段起点的相对值。位置单位为 unit，时间单位为秒。

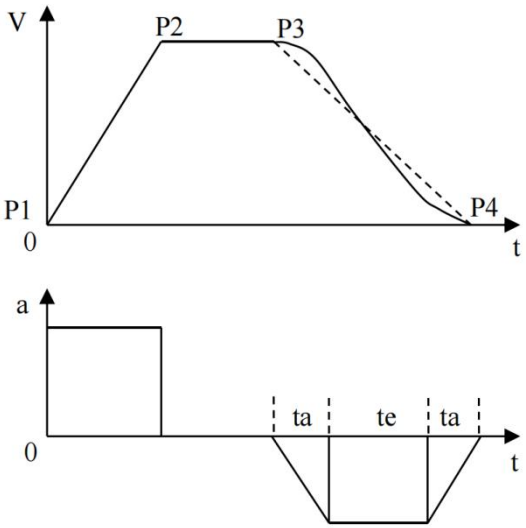
PT 模式在实现任意速度规划方面非常具有优势。用户将任意的速度规划曲线分割为足够密的小段，用户只需要给出每段所需时间和位置点，运动控制卡会计算段内各点的速度，生成一条连续的速度曲线。为了得到光滑的速度曲线，可以增加速度曲线的分割段数。

（2）PTS 运动模式

PTS 运动模式是 PTT 的扩展功能模式。PTS 的最后一个字母 S 表示 S 形速度曲线；PTS 模式是在 S 形速度曲线下规划点位运动；和 PTT 模式相比，其各段速度过渡更加平滑。

用户通过输入一系列位置、时间、百分比参数，自定义单轴的运动规律。其中位置、时间参数定义和 PTT 模式相同；“百分比”参数是指：相邻 2 个数据点之间加速度的变化时间占速度变化时间的百分比。

以下图为例说明，数据点 P2 和 P3 之间加速度不变，因此数据点 P2 的百分比为 0；数据点 P3 和 P4 之间加速度变化时间为 2×ta，速度变化时间为 2×ta+te，因此数据点 P3 的百分比为[2×ta/(2×ta+te)]×100%。



加速度变化时间百分比定义

调整百分比参数，可以改变 S 形速度曲线的形状。如上图所示，当数据点 P3 的百分比为 0 时，数据点 P3 和 P4 之间的速度曲线为直线（如图中虚线所示）；当数据点 P3 的百分比不为 0 时，数据点 P3 和 P4 之间的速度曲线为 S 形曲线（如图中实线所示）。“百分比”参数越大，S 段曲线越长。

（3）PVT 运动模式

PVT 模式是 PT 模式的进阶版本，其通过的位置、速度和时间这三种参数共同组成的数组来描述运动规律。与 PT 模式相同，PVT 模式同样要求用户将待规划的速度曲线通过分段的形式来进行描述，其中位置、速度和时间满足如下函数关系：

$$p = at^3 + bt^2 + ct + d$$

$$v = \frac{dp}{dt} = 3at^2 + 2bt + c$$

如果给定相邻 2 个数据点的“ 位置、速度、时间”参数，可以得到如下方程组：

$$at_1^3 + bt_1^2 + ct_1 + d = p_1$$

$$3at_1^2 + 2bt_1 + c = v_1$$

$$at_2^3 + bt_2^2 + ct_2 + d = p_2$$

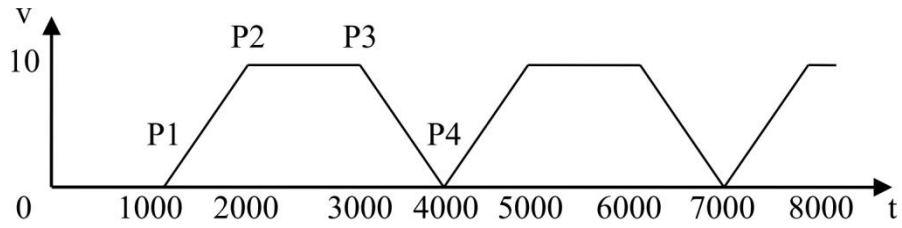
$$3at_2^2 + 2bt_2 + c = v_2$$

求解该方程组，可以得到 a、b、c、d，因此相邻 2 个数据点的运动规律就可以确定下来。

假设有如下所示的 4 个数据点，采用 PVT 方式进行描述。

数据点	时间（ms）	位置（unit）	速度（unit/ms）
P1	1000	0	0
P2	2000	5000	10
P3	3000	15000	10
P4	4000	20000	0

采用循环调用的方式，运动的速度曲线如下图所示：



循环执行数据表

（4）特别注意：下载的第一组（即起始点）数据中位置、时间必须为 0；数组中的数据都是以起始点的数据为参考点。

例程

例程 1：PTT 运动，请参考工程“PVTSample”。

添加 PTT 运动的位置点：

```
Double[] pTime = new double[50];
Double[] pPos = new double[50];
pTime[0] = 0;
pPos[0] = 0;
```

```
pTime[1] = 0.2;
pPos[1] = 300;
pTime[2] = 0.8;
pPos[2] = 1200;
pTime[3] = 1;
pPos[3] = 1500;
pTime[4] = 1.1;
pPos[4] = 1550;
Count = 5;
ret = NV_PttAddTable(m_nConnectNo, m_nAxisNo, pTime, pPos, Count);
```

启动运动：

```
UInt16 AxisNum = 0;
UInt16[] pAxisList = new ushort[1];
Double[] pDelayTime = new double[1];
AxisNum = 1;//要启动 PVT 的轴数量:1
pAxisList[0] = 1;     //要启动 PVT 的轴列表:Axis1
pDelayTime[0] = 0;    //各轴延时启动的时间
ret = NV_PvtStart(m_nConnectNo, AxisNum, pAxisList, pDelayTime);
```

例程 2：PVT 运动，请参考工程“PVTSample”。

添加 PVT 运动的位置点：

```
Double[] pTime = new double[50];
Double[] pPos = new double[50];
Double[] pVel = new double[50];
pTime[0] = 0;
```

```
pPos[0] = 0;
pVel[0] = 0;
pTime[1] = 2;
pPos[1] = 5000;
pVel[1] = 1000;
pTime[2] = 3;
pPos[2] = 15000;
pVel[2] = 1000;
pTime[3] = 4;
pPos[3] = 2000;
pVel[3] = 0;
Count = 4;
ret = NV_PvtAddTable(m_nConnectNo, m_nAxisNo, pTime, pPos, pVel, Count);
```

启动运动：

```
UInt16 AxisNum = 0;
UInt16[] pAxisList = new ushort[1];
Double[] pDelayTime = new double[1];
AxisNum = 1;//要启动 PVT 的轴数量:1
pAxisList[0] = 1;     //要启动 PVT 的轴列表:Axis1
pDelayTime[0] = 0;    //各轴延时启动的时间
ret = NV_PvtStart(m_nConnectNo, AxisNum, pAxisList, pDelayTime);
```

4.14 电子齿轮

指令列表

名称	功能

NV_GearSetParam	设置电子齿轮的参数
NV_GearGetParam	读取电子齿轮的参数
NV_GearIn	进入电子齿轮模式
NV_GearInPos	进入电子齿轮模式
NV_GearOut	退出电子齿轮模式

重点说明

（1）电子齿轮模式

电子齿轮模式能够将两轴或多轴联系起来，实现精确的同步运动，从而替代传统的机械齿轮连接，电子齿轮模式能够灵活的设置传动比，节省机械系统的安装时间。我们把被跟随的轴叫主轴，把跟随的轴叫从轴。电子齿轮模式下，1 个主轴能够驱动多个从轴，从轴可以跟随主轴的指令位置、反馈位置。

齿轮比：主轴位置与从轴位置的比例，例如主从轴连接比例为 1:5，主轴发送 1 个脉冲，此时对应给从轴发送 5 个脉冲；当主轴速度变化（位置也同时变化）时，从轴会根据设定好的传动比自动跟随变化；电子齿轮模式也能够在运动过程中修改传动比。

（2）直接啮合的方式

通过指定参数，进行齿轮动作；开始动作后，Slave（从轴）以 Master（主轴）速度乘以齿轮比得到的速度为目标速度，进行加减速动作。

该方式电子齿轮函数为 NV_GearIn。

（3）从指定位置同步的方式

同步开始到同步结束的过程本质为同步区间内从轴跟随主轴的一个电子凸轮；此时根据主轴范围（MasterSyncPosition- MasterStartDistance，MasterSyncPosition），从轴范围（当前位置， SlaveSyncPosition），指令会根据设置齿轮比以及上述三个参数自动设计一条凸轮曲线，执行同步时从轴跟随主轴完成啮合动作。

AvoidReversal：设置为 0（FALSE），如果从轴物理位置超前的情况下进行反转；

设置为 1 （TRUE）如果从轴物理上不能实现反转或者导致危险。

只在模态轴下适用。 如果反转不能被避免，那么轴将错误停止。

该方式电子齿轮函数为 NV_GearInPos。

（4）电子齿轮的解除

要解除耦合必须通过 GearOut 指令。

（5）电子齿轮的数据结构

```
typedef struct GearProfileStruct
{
    UInt16 MasterAxisNo;          // 主轴编号
    UInt16 MasterSrcType;         // 主轴位置源类型
    Int32 RatioNumerator;         // 电子齿轮比分子
    Int32 RatioDenominator;     // 电子齿轮比分母
    Double Acceleraton;         // 加速度
    Double Deceleraton;         // 减速度
    Double MasterSyncPosition; // 主轴同步位置，在这个位置前完成同步，[GearInPos]
    Double SlaveSyncPosition;   // 从轴同步位置，在这个位置前完成同步，[GearInPos]
    Double MasterStartDistance;// 主轴开始距离，在这个距离间进行同步，[GearInPos]
    UInt16 AvoidReversal;         // 禁止反转：0-允许反转；1-不允许反转，[GearInPos]
}GearProfileStruct_t,
```

例程

例程：单轴电子齿轮，请参考工程“GearSample”

配置电子齿轮参数

```
ERROR_CODE_TYPE ret = 0;
m_sProfile.MasterAxisNo = 1;     // 主轴编号:1
m_sProfile.MasterSrcType = 1;   //指令位置：1
m_sProfile.RatioNumerator = 3：// 电子齿轮比分子;
```

```
m_sProfile.RatioDenominator = 1;  // 电子齿轮比分母
m_sProfile.Acceleraton = 100;    // 加速度;
m_sProfile.Deceleraton = 100;  // 减速度
m_sProfile.AvoidReversal = 1;  //1-不允许反转
m_sProfile.MasterSyncPosition = 500; // 主轴同步位置
m_sProfile.SlaveSyncPosition = 30; // 从轴同步位置
m_sProfile.MasterStartDistance = 200;// 主轴开始距离
ret = NV_GearSetParam(m_nConnectNo, m_nSlaveAxisNo, m_sProfile);
if (ret != 0)
{
MessageBox.Show($"NV_GearSetParam 调用报错：{ret}");
return;
}
```

启动电子齿轮运动：

```
ret = NV_GearInPos(m_nConnectNo, m_nSlaveAxisNo);
if (ret != 0)
{
MessageBox.Show($"NV_GearInPos 调用报错：{ret}");
return;
}
```

脱离电子齿轮运动

```
ERROR_CODE_TYPE ret = 0;
ret = NV_GearOut(m_nConnectNo, m_nSlaveAxisNo);
if (ret != 0)
```

```
{
MessageBox.Show($"NV_GearOut 调用报错：{ret}");
return;
}
```

4.15 电子凸轮

指令列表

名称	功能
NV_CAMAddTable	将凸轮数据添加到 Table 表
NV_CAMGetTable	读取 Table 表数据
NV_CAMGetTableSize	读取 Table 表中数据个数
NV_CAM	电子凸轮运动（根据凸轮表运动）
NV_CAMInAddTable	将凸轮主从轴数据添加到 Table 表
NV_CAMIn	跟随凸轮表运动（根据凸轮表运动）
NV_CAMOut	退出电子凸轮运动

重点说明

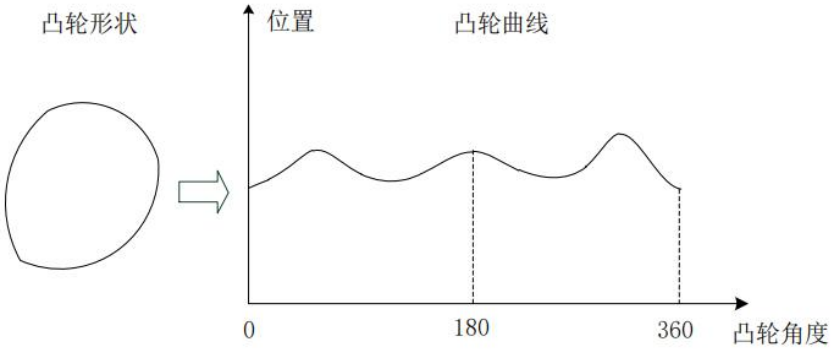
（1）凸轮组成

机械凸轮由主动件和从动件。主动件为在金属盘上加工轮廓曲线，一般为匀速旋转运动；从动件一般与凸轮轮廓接触，主动件在运动时，从动件往复运动，运动轨迹由凸轮轮廓决定。由于凸轮机构属于高副机构，故凸轮与从动件之间为点或线接触，不便润滑、易于磨损，有噪声，盘片的加工制造要求较高，维修复杂。因此在很多应用上，使用电子凸轮替代机械凸轮。

电子凸轮是利用构造的凸轮曲线来模拟机械凸轮，以达到机械凸轮系统相同的凸轮轴与主轴之间相对运动的软件系统，通过控制器控制伺服电机来模拟机械凸轮的功能，不需要另外安装机械结构。

（2）电子凸轮工作原理

电子凸轮属于多轴同步运动，这种运动是基于主轴外加一个或多个从轴系统，是在机械凸轮的基础上发展而来，电子凸轮多用于周期性的曲线运动场合。如下图，机械凸轮按照凸轮的轮廓可以得出一段转动角度与加工位置运动轨迹，此轨迹为弧线，将该段弧线分解成无数个直线轨迹，组合起来得到一串趋近于弧线运动轨迹，电子凸轮直接将此段轨迹运动参数装入运动指令，即可控制轴走出目标轨迹。



设置一段电子凸轮的运动程序装入控制器，通过编码器将位置信号反馈给控制器，控制器将接收到的位置信号进行处理后输出给伺服驱动器，伺服驱动器控制多个轴同步运动完成预设轨迹。

（3）凸轮功能

控制卡为用户提供了多种凸轮运动形式：

CAM：凸轮表运动；

CAMIN：跟随凸轮表运动；

CAMOut：退出凸轮运动。

（4）凸轮表运动

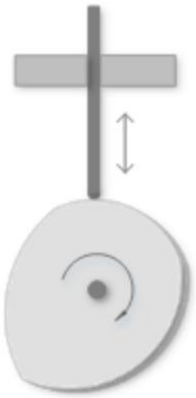
凸轮表运动，用户可以通过存储在 Table 表的位置信息，形成运动轨迹。

通过函数 NV_CAMAddTable 可以往 Table 表中添加数据，系统最大支持 4 个表。

通过函数 NV_CAM 调用表中的数据（数据表示轴移动的位移，单位为 units），执行运动；整个位移执行的时间，通过参数 Time 设置。

（5）跟随凸轮表运动

跟随凸轮表运动，指从轴跟随主轴运动，两轴之间的位置关系由凸轮表的位置信息决定。



机械凸轮示意图

参考机械凸轮示意图，跟随凸轮表运动中的主轴相当于旋转的金属盘，从轴相当于上下运动的连杆结构，凸轮表相当于金属盘的轮廓曲线。

主轴：可以为虚拟轴，也可以为实轴，通常为旋转结构，匀速运动；

从轴：一般为实轴，通常为直线结构；

函数 NV_CAMIN，根据存储在 Table 表中的数据来决定轴的运动。这些 Table 数据值对应运动轨迹的位置，是相对于运动起始点的距离。跟随轴的运动根据凸轮主轴的运动来进行。

凸轮运动的触发及运动方式，由参数 LinkOptions 设置，不同的 bit 位表示不同的意：

Bit 0 当输入信号事件触发时，主从轴开始连接运动。

Bit 1 当主轴运动到设定的绝对位置时，主从轴开始连接运动。

Bit 2 自动重复连续双向运行。

Bit 5 只有主轴正向运动才连接。

（a）当 Options 的 Bit0 置为 1 时，表示凸轮运动由输入信号触发，触发信号由下面两个参数配置：

DiNum 触发输入信号：普通输入 IN0-7

DiLogic 触发逻辑：0-下降沿，1-上升沿，2-任意沿

（b）当 Options 的 Bit1 置为 1 时，表示主轴到达设定的绝对位置时，主从轴开始连接运动，位置值由参数 MasterPos 设置，单位为 units。

例程

例程 1：使用 CAM 指令实现 Sin 曲线运动

设计 Sin 曲线，将轴 0 在 5 秒时间内完成一次设计的曲线运动

```
//填写凸轮表
//SIN 曲线的波峰为 50 个 units
Double sin_maxval=50.0;
Double TableVal[360];
for (int i = 0; i <360; i++)
{
double angle = i;
double radians = i * 2 * PI / 360 'convert degrees to radians;
TableVal[i] = SIN(radians) * sin_maxval;
}

//将数据放入 Table 表
ConnectNo = 0；
TableIndex = 0；
Numes = 360；
IsNew = 0；
NV_CAMAddTable（ConnectNo, TableIndex，Numes，TableVal ，IsNew）；


//执行凸轮表运动
AxisNo = 1；
Time = 5.0；
NV_CAM(ConnectNo, AxisNo, TableIndex, Time)；
```

例程 2：电子凸轮工程示例“CAMSamlpe”

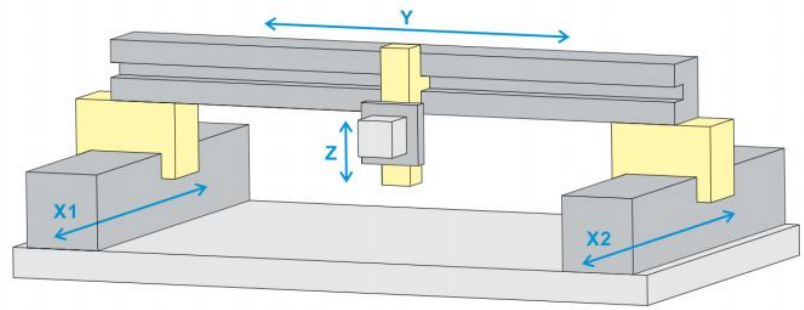
4.16 龙门功能

指令列表

名称	功能
NV_GantryIn	设置龙门的参数
NV_GantryOut	退出龙门模式
NV_GantryGetParam	设置龙门的参数
NV_GantrySetProtect	设置龙门误差保护参数
NV_GantryGetProtect	读取设置龙门误差保护参数

重点说明

龙门控制它是针对需要两个机床轴（x1、x2）共同推动横梁的设备结构而设计，如下图所示。



龙门结构示意图

龙门控制将 x1、x2 作为一个龙门组轴进行同步控制，使龙门组轴中的机床轴同步无偏差的运动。

龙门功能包括开环龙门模式、闭环龙门模式。开环龙门模式只保证龙门主轴和龙门从轴的规划位置是同步的。闭环龙门模式，龙门主轴和龙门从轴的规划位置不仅是同步的，控制卡还会根据龙门主轴和龙门从轴的实际位置进行同步控制，从而保证龙门主轴和龙门从轴的实际位置的绝对同步。

控制卡中同步源，可以由用户来选择，通过函数 NV_GantryIn 来进行配置。

使用龙门时需要注意：

（a）主从轴的配对关系：主轴可以配置多个从轴，但是从轴不能配置跟随多个主轴；

（b）龙门模式下：从轴的硬限位、软限位无效。

例程

例程：龙门功能，请参考工程“GantrySample”

设置误差保护：

```
ERROR_CODE_TYPE ret = 0;
UInt16 Enable = 1;  //启用误差保护功能
Double DecStopError = 50;  //触发减速停止时的误差值
Double ImdStopError = 100;  //触发立即停止时的误差值
UInt16 ErrorSrcType = 2; //误差比较来源：2 - 反馈位置
ret = NV_GantrySetProtect(m_nConnectNo, m_nSlaveAxisNo, Enable, DecStopError, ImdStopError, ErrorSrcType);
if (ret != 0)
{
MessageBox.Show($"NV_GantrySetProtect 调用报错：{ret}");
return;
}
```

龙门启动配置：

```
UInt16 MasterSrcType = 0; //主轴类型：2 - 反馈位置
UInt16 Direction = 0;//跟随方向：0 – 跟主轴同向
ret = NV_GantryIn(m_nConnectNo, m_nSlaveAxisNo);
if (ret != 0)
{
MessageBox.Show($"NV_GantryIn 调用报错：{ret}");
```

```
return;
}
```

5 函数详细说明

5.1 初始化函数

5.1.1 初始化控制卡

ERROR_CODE NV_ConnectPCIScan(ref ScanResult_t pScanResult);

功　能：扫描 PCI/PCIe 卡

参 数：pScanResult 返回扫描到的 PCI 卡信息

返回值：错误代码

ERROR_CODE NV_ConnectNETScan(ref ScanResult_t pScanResult);

功　能：扫描网络卡

参 数：pScanResult 返回扫描到的网络卡信息

返回值：错误代码

ERROR_CODE NV_ConnectScan(ref ScanResult_t pScanResult);

功　能：扫描工控机内所有卡。先扫描 PCI/PCIe 卡再扫描网络卡

参 数：pScanResult 返回扫描到的卡信息

返回值：错误代码

ERROR_CODE NV_ConnectOpenPCI(UInt16 ConnectNo)

功　能：打开 PCI/PCIe 卡（一个卡）

参　数：ConnectNo　　板卡连接编号[0,15]，范围 0-7

返回值：错误代码

ERROR_CODE NV_ConnectOpenPCIAll(ref ScanResult_t pScanResult)

功　能：打开所有 PCI/PCIe 卡

参　数：pScanResult　　PCI/PCIe 卡的信息结构体

返回值：错误代码

ERROR_CODE NV_ConnectOpenNET(UInt16 ConnectNo, string serverIP, UInt16 serverPort)

功　能：打开网络卡（一个卡）

参　数：ConnectNo　　板卡连接编号[0,15]，范围 0-7

serverIP 网络卡的 IP 地址

serverPort　网络卡的端口号

返回值：错误代码

ERROR_CODE NV_ConnectIsOnline(UInt16 ConnectNo,ref UInt16 pIsOnline)

功　能：控制卡连接是否在线

参　数：ConnectNo　　板卡连接编号[0,15]，范围 0-7

pIsOnline　　在线状态：0 - 不在线； 1 - 在线

返回值：错误代码

5.1.2 控制卡复位

ERROR_CODE NV_ConnectColdReset(UInt16 ConnectNo)

功　能：控制卡冷复位

参　数：ConnectNo　　板卡连接编号[0,15]，范围 0-7

返回值：错误代码

说明：相当于控制卡重新启动，里面所有的数据、状态都被清除。

执行复位操作后，必须关闭控制卡，等待 15 秒方可执行初始化控制卡，否则会出错。出错后必须重新执行复位操作，再等待 15 秒后执行初始化控制卡。

ERROR_CODE NV_ConnectWarmReset(UInt16 ConnectNo)

功　能：控制卡热复位

参　数：ConnectNo　　板卡连接编号[0,15]，范围 0-7

返回值：错误代码

说明：总线复位重新初始化、软件中的临时变量复位到初始值；但是不会改变用户已经配置的控制卡参数/功能，如 IO 的输出特性、高速锁存功能等。

执行总线热复位操作后，循环调用总线错误读取函数判断总线状态，如果总线状态返回正常，则可退出循环，总线热复位完成。

5.1.3 关闭控制卡

ERROR_CODE NV_ConnectClose()

功　能：关闭单张控制卡

返回值：错误代码

ERROR_CODE NV_ConnectCloseAll()

功　能：关闭所有控制卡

返回值：错误代码

5.1.4 配置恢复出厂设置

ERROR_CODE NV_ConfigRecovery(UInt16 ConnectNo)

功　能：恢复出厂配置

参　数：ConnectNo　　板卡连接编号[0,15]，范围 0-7

返回值：错误代码

说明：主要用于删除系统中的所有变量及文件信息。包括：eni 文件，eeprom_config 文件，全局变量文件等。调用该函数只是删除了文件信息，此时需要，再调用一次冷复位或者热复位重启总线才可以完全清除系统的临时缓存信息。执行初始复位操作后，等待 2 秒后再调用冷复位或热复位控制卡函数，否则会出错。

5.1.5 获取控制卡相关信息

ERROR_CODE NV_ConfigGetProductInfo(UInt16 ConnectNo,byte[] pProductName,byte[] pProductID)

功　能：获取产品信息

参　数：ConnectNo　板卡连接编号[0,15]

pProductName　返回最多 64 个字符的产品名称，比如“NC3000”

pProductID 返回最多 64 个字符的产品唯一编号，比如“FA942EDF44FSD”

返回值：错误代码

ERROR_CODE NV_ConfigGetDllVersion(byte[] pVersion,byte[] pVersionTime)

功　能：获取动态库版本及发布时间

参　数：ConnectNo　板卡连接编号[0,15]

pVersion　返回最多 64 个字符的固件版本，比如“1.0.0.1”

pVersionTime　返回最多 64 个字符的发布时间，比如“20250713”

返回值：错误代码

ERROR_CODE NV_ConfigGetSoftVersion(UInt16 ConnectNo,byte[] pVersion,byte[] pVersionTime)

功　能：获取固件版本及发布时间

参　数：ConnectNo　板卡连接编号[0,15]

pVersion　返回最多 64 个字符的固件版本，比如“1.0.0.1”

pVersionTime　返回最多 64 个字符的发布时间，比如“20250713”

返回值：错误代码

ERROR_CODE NV_ConfigGetHardVersion (UInt16 ConnectNo,ref string pVersion,ref string pVersionTime)

功　能：获取硬件版本及发布时间

参　数：ConnectNo　　板卡连接编号[0,15]

pVersion　　返回最多 64 个字符的固件版本，比如“1.0.0.1”

pVersionTime　返回最多 64 个字符的发布时间，比如“20250713”

返回值：错误代码

ERROR_CODE NV_ConfigGetAxisNum (UInt16 ConnectNo,ref UInt16 pAxisNum)

功　能：获取轴资源信息

参　数：ConnectNo 板卡连接编号[0,15]

pAxisNum　返回支持的轴数量

返回值：错误代码

ERROR_CODE NV_ConfigGetEncoderNum(UInt16 ConnectNo, ref UInt16 pEncoderNum);

功　能：获取编码器数量

参　数：ConnectNo　　板卡连接编号[0,15]

pEncoderNum 返回编码器数量

返回值：错误代码

ERROR_CODE NV_ConfigGetIoNum(UInt16 ConnectNo, ref UInt16 pInputNum,ref UInt16 pOutputNum)

功　能：获普通 IO 数量

参　数：ConnectNo　　板卡连接编号[0,15]

pInputNum　　返回支持的输入数量

pOutputNum　　返回支持的输出数量

返回值：错误代码

ERROR_CODE NV_ConfigGetHSIoNum(UInt16 ConnectNo, ref UInt16

pInputNum,ref UInt16 pOutputNum)

功　能：获高速 IO 数量

参　数：ConnectNo　板卡连接编号[0,15]

pInputNum　返回支持的高速输入数量

pOutputNum　返回支持的高速输出数量

返回值：错误代码

ERROR_CODE NV_ConfigGetCmpNum(UInt16 ConnectNo, ref UInt16 pCmpNum,ref UInt16 pHCmpNum)

功　能：获比较器数量

参　数：ConnectNo　板卡连接编号[0,15]

pCmpNum 返回支持的软件比较器数量

pHCmpNum 返回支持的高速比较器数量

返回值：错误代码

ERROR_CODE NV_ConfigGetLatchNum(UInt16 ConnectNo, ref UInt16 pLatchNum,ref UInt16 pHLatchNum)

功　能：获锁存器数量

参　数：ConnectNo　板卡连接编号[0,15]

pLatchNum　返回支持的软件锁存器数量

pHLatchNum　返回支持的高速锁存器数量

返回值：错误代码

ERROR_CODE NV_ConfigGetInputNumList(UInt16 ConnectNo, UInt16[] pInputNumList)

功　能：获取 IO 输入数量

参　数：ConnectNo 控制卡卡号

pInputNumList 返回各种输入端口的数量

返回值：错误代码

读取的各种输入端口数量，按照下表顺序排列：

值序号	输入端口类型
pInputNumList[0]	类型数量
pInputNumList[1]	通用输入端口
pInputNumList[2]	高速输入端口
pInputNumList[3]	原点输入端口
pInputNumList[4]	正限位输入端口
pInputNumList[5]	负限位输入端口
pInputNumList[6]	伺服报警输入端口
pInputNumList[7]	伺服到位输入端口
pInputNumList[8]	伺服准备输入端口
pInputNumList[9]	伺服 EZ 输入端口
pInputNumList[10]	手轮轴选输入端口
pInputNumList[11]	手轮倍选输入端口
pInputNumList[12]	手轮急停输入端口
pInputNumList[13]	模拟量输入端口

ERROR_CODE NV_ConfigGetOutputNumList(UInt16 ConnectNo, UInt16[] pOutputNumList)

功　能：获取 IO 输出数量

参　数：ConnectNo 控制卡卡号

pOutputNumList 返回各种输出端口的数量

返回值：错误代码

读取的各种输出端口数量，按照下表顺序排列：

值序号	输出端口类型

pOutputNumList[0]	类型数量
pOutputNumList[1]	通用输出端口
pOutputNumList[2]	高速输出端口
pOutputNumList[3]	伺服使能输出端口
pOutputNumList[4]	伺服清除输出端口
pOutputNumList[5]	PWM 输出端口
pOutputNumList[6]	模拟量输出端口

ERROR_CODE NV_ConfigGetAxisNumList(UInt16 ConnectNo, UInt16[] pAxisNumList)

功　能：获取各种类型的轴的数量

参　数：ConnectNo 控制卡卡号

pAxisNumList　返回各种类型的轴的数量

返回值：错误代码

读取的各种类型轴的数量，按照下表顺序排列：

值序号	轴的类型
pAxisNumList[0]	类型数量
pAxisNumList[1]	虚拟轴
pAxisNumList[2]	差分脉冲轴(5V)
pAxisNumList[3]	EtherCAT 总线轴
pAxisNumList[4]	单端脉冲轴(24V)

ERROR_CODE NV_ConfigGetEncoderNumList(UInt16 ConnectNo, UInt16[] pEncoderNumList)

功　能：获取各种类型的编码器的数量

参　数：ConnectNo　控制卡卡号

pEncoderNumList 返回各种类型的编码器的数量

返回值：错误代码

读取的各种类型编码器的数量，按照下表顺序排列：

值序号	编码器的类型
pEncoderNumList[0]	类型数量
pEncoderNumList[1]	差分编码器(5V)
pEncoderNumList[2]	单端编码器(24V)
pEncoderNumList[3]	手轮编码器

ERROR_CODE NV_ConfigGetFunNumList(UInt16 ConnectNo, UInt16[] pFunNumList)

功　能：获取各种功能通道的数量

参　数：ConnectNo 控制卡卡号

pFunNumList　返回各种功能通道的数量

返回值：错误代码

读取的各种功能通道的数量，按照下表顺序排列：

排列顺序序号	功能的类型
pFunNumList[0]	类型数量
pFunNumList[1]	IO 计数器
pFunNumList[2]	软件低速锁存器
pFunNumList[3]	软件低速比较器
pFunNumList[4]	软件低速二维比较器
pFunNumList[5]	硬件高速锁存器
pFunNumList[6]	硬件高速比较器
pFunNumList[7]	硬件高速二维比较器
pFunNumList[8]	插补器
pFunNumList[9]	插补器最大插补轴数
pFunNumList[10]	看门狗
pFunNumList[11]	凸轮表

5.2 系统配置函数

ERROR_CODE NV_FileDownloadMemfile(UInt16 ConnectNo, Byte[] pBuffer, UInt32 BuffSize, string pFileNameInControl, UInt16 FileType);

功　能：下载 PC 端的内存文件到控制卡的 FLASH

参　数：ConnectNo　　板卡连接编号[0,15]

pBuffer　　内存文件缓冲区

BuffSize　内存文件大小，单位：字节

pFileNameInControl 控制卡内文件名

FileType　文件类型：

2：参数

200：eni 文件，

201：ini 文件（总线配置文件）

返回值：错误代码

5.3 安全机制相关函数

5.3.1 参数与状态函数

ERROR_CODE NV_AxisSetStopParam(UInt16 ConnectNo,UInt16 AxisNo,UInt16 Mode, Double StopTime,Double StopDist)

功　能：设置轴的异常停止参数，按照最短距离停止，不超过原始目标位置。

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

Mode 停止方式：0-最短距离停止，1-按时间停止，2-按距离停止

StopTime　　异常停止的时间，单位 s，范围 0－60

StopDist　　异常停止的距离，单位 unit

返回值：错误代码

备注：最短距离停止方式：取“按时间计算的停止距离”以及“设定的停止距离”

当中的最短的距离停止。无论哪种停止方式，最终不超过设定的目标位置。

ERROR_CODE NV_AxisGetStopParam(UInt16 ConnectNo,UInt16 AxisNo,ref UInt16 pMode, ref Double pStopTime,ref Double pStopDist)

功　能：获取轴的异常停止参数

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

pMode　返回停止方式

pStopTime　返回异常停止的时间

pStopDist　返回异常停止的距离

返回值：错误代码

ERROR_CODE NV_AxisGetStopReason(UInt16 ConnectNo,UInt16 AxisNo,UInt32[] pReason,UInt16 Count)

功　能：获取轴运动停止的原因

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

pReason　返回的停止原因

Count　返回停止原因的数量

返回值：错误代码

ERROR_CODE NV_AxisClrStopReason(UInt16 ConnectNo,UInt16 AxisNo)

功　能：清除轴运动停止原因的记录

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

返回值：错误代码

5.3.2 软限位停止功能

ERROR_CODE NV_AxisSetSoftLimit(UInt16 ConnectNo,UInt16 AxisNo,UInt16 Enable,Double PositivePos,Double NegtivePos,UInt16 PosSource,UInt16 StopMode)

功　能：设置轴的软限位功能

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

Enable 设置是否启用软限位：0 – 不启动；1 – 启用

PositivePos 设置软限位最大位置值，脉冲范围 - 2147483647～2147483648

NegtivePos 设置软限位最小位置值，脉冲范围 - 2147483647～2147483648

PosSource 设置软限位的来源：1 – 轴指令位置；2 – 轴反馈位置

StopMode 设置软限位的停止方式：0 - 减速停止；1 - 立即停止

返回值：错误代码

ERROR_CODE NV_AxisGetSoftLimit(UInt16 ConnectNo,UInt16 AxisNo,ref UInt16 pEnable,ref Double pPositivePos,ref Double pNegtivePos,ref UInt16 pPosSource,ref UInt16 pStopMode)

功　能：获取轴的软限位功能配置

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

pEnable　返回是否启用软限位：0 - 不启动；1 - 启用

pPositivePos 返回软限位最大位置值

pNegtivePos　返回软限位最小位置值

pPosSource 返回软限位的来源：1 – 轴指令位置；2 – 轴反馈位置

pStopMode　返回软限位的停止方式：0 - 减速停止；1 - 立即停止

返回值：错误代码

ERROR_CODE NV_AxisSetCircleLimit(UInt16 ConnectNo, UInt16 Enable,UInt16[] pAxisList ,Double[] pCenter, Double Radius, UInt16 PosSource,UInt16 StopMode)

功　能：设置圆弧限位保护功能

参　数：ConnectNo　板卡连接编号[0,15]

Enable 使能：0 - 不启动；1 - 启用

pAxisList　轴序号列表，必须要有两个不同的轴

pCenter　两轴平面的圆心，单位 units

Radius　最大的半径范围，单位 units

PosSource 设置软限位的来源：1 – 轴指令位置；2 – 轴反馈位置

StopMode 设置软限位的停止方式：0 - 减速停止；1 - 立即停止

返回值：错误代码

ERROR_CODE NV_AxisGetCircleLimit(UInt16 ConnectNo,ref UInt16 pEnable,UInt16[] pAxisList,Double[] pCenter, ref Double pRadius, ref UInt16 pPosSource,ref UInt16 pStopMode)

功　能：获取圆弧限位保护功能

参　数：ConnectNo　板卡连接编号[0,15]

pEnable　获取使能

pAxisList 轴序号列表，必须要有两个不同的轴

pCenter　返回两轴平面的圆心

pRadius　返回最大的半径范围

pPosSource 返回软限位的来源：1 – 轴指令位置；2 – 轴反馈位置

pStopMode　返回软限位的停止方式：0 - 减速停止；1 - 立即停止

返回值：错误代码

ERROR_CODE NV_AxisSetErrorBand(UInt16 ConnectNo,UInt16 AxisNo,UInt16 Enable,Double ErrorBand,Double HoldTime)

功　能：设置超差停止功能。轴的指令位置和编码器反馈位置超过最大允许误差，系统停止轴运动，防止异常发生。

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

Enable 使能：0 - 不启动；1 - 启用

ErrorBand 差值范围

HoldTime　最少停留时间，单位 us

返回值：错误代码

ERROR_CODE NV_AxisGetErrorBand(UInt16 ConnectNo,UInt16 AxisNo,ref UInt16 pEnable,ref Double pErrorBand,ref Double pHoldTime)

功　能：获取超差停止功能状态

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

pEnable　返回使能状态

pErrorBand 返回差值范围

pHoldTime　返回最少停留时间

返回值：错误代码

5.3.3 命令触发停止功能

ERROR_CODE NV_EmgStop(UInt16 ConnectNo)

功　能：紧急停止所有轴

参　数：ConnectNo　板卡连接编号[0,15]

返回值：错误代码

注 意：此函数适用于所有运动模式

ERROR_CODE NV_AxisStop(UInt16 ConnectNo,UInt16 AxisNo, UInt16 StopMode)

功　能：指定轴停止运动

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

StopMode　停止方式，0：减速停止，1：紧急停止

返回值：错误代码

5.4 轴参数相关函数

5.4.1 轴基础参数

ERROR_CODE NV_AxisSetEnable(UInt16 ConnectNo,UInt16 AxisNo,UInt16 Enable)

功　能：设置轴使能

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

Enable　设置轴是否启用：0 - 不使能 ；1 - 使能

返回值：错误代码

ERROR_CODE NV_AxisGetEnable(UInt16 ConnectNo,UInt16 AxisNo,UInt16[] pEnable,UInt16 Count)

功　能：获取轴使能

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

pEnable　获取设置轴是否启用

Count　从 AxisNo 号开始一次性读取后续 Count 个轴的值

返回值：错误代码

ERROR_CODE NV_AxisSetBusMode(UInt16 ConnectNo,UInt16 AxisNo,UInt16

Value)

功　能：设置总线轴控制模式

参　数：ConnectNo　　板卡连接编号[0,15]

AxisNo　　轴序号

Value　　设置总线轴控制模式

返回值：错误代码

注意：总线轴控配置模式编号，如下定义的枚举值

typedef enum

{

BUS_MODE_NULL　= 0,

BUS_MODE_PP= 1,

BUS_MODE_VL= 2,

BUS_MODE_PV= 3,

BUS_MODE_TQ= 4,

BUS_MODE_HM= 6,

BUS_MODE_IP= 7,

BUS_MODE_CSP　= 8,

BUS_MODE_CSV　= 9,

BUS_MODE_CST　= 10,

BUS_MODE_MAX　= 11

} BUS_MODE_TYPE;

ERROR_CODE　NV_AxisGetBusMode(UInt16　ConnectNo,UInt16　AxisNo,UInt16[] pValue,UInt16 Count)

功　能：获取总线轴控制模式

参　数：ConnectNo　　板卡连接编号[0,15]

AxisNo　　轴序号

pValue　　获取总线轴控制模式

Count　从 AxisNo 号开始一次性读取后续 Count 个轴的值

返回值：错误代码

ERROR_CODE　NV_AxisSetStepPulse(UInt16　ConnectNo,UInt16　AxisNo,UInt16 StepPulse)

功　能：设置轴脉冲类型

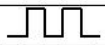
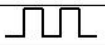
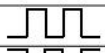
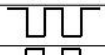
参　数：ConnectNo　　板卡连接编号[0,15]

AxisNo　　轴序号

StepPulse　设置轴对应的脉冲类型：0-6

返回值：错误代码

脉冲类型 StepPulse 如下：

输出脉冲类型 OUTMODE	正方向脉冲		负方向脉冲	
	PULSE 输出端	DIR 输出端	PULSE 输出端	DIR 输出端
0		高电平		低电平
1		高电平		低电平
2		低电平		高电平
3		低电平		高电平
4		高电平	高电平	
5		低电平	低电平	
6	AB 相输出，A 超前 B 90°		AB 相输出，B 超前 A 90°	

ERROR_CODE　NV_AxisGetStepPulse(UInt16　ConnectNo,UInt16　AxisNo,UInt16[] pStepPulse,UInt16 Count)

功　能：获取轴脉冲类型

参　数：ConnectNo　　板卡连接编号[0,15]

AxisNo　　轴序号

pStepPulse　获取轴对应的脉冲类型

Count　从 AxisNo 号开始一次性读取后续 Count 个轴的值

返回值：错误代码

ERROR_CODE NV_AxisSetUnit(UInt16 ConnectNo,UInt16 AxisNo,Double Value)

功　能：设置轴的当量

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

Value　设置轴的当量值，必须是正数。

返回值：错误代码

ERROR_CODE NV_AxisGetUnit(UInt16 ConnectNo,UInt16 AxisNo,Double[] pValue,UInt16 Count)

功　能：获取轴的当量

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

pValue 获取轴的当量值

Count　从 AxisNo 号开始一次性读取后续 Count 个轴的值

返回值：错误代码

ERROR_CODE NV_AxisSetMaxVel(UInt16 ConnectNo,UInt16 AxisNo,Double MaxVel)

功　能：设置轴的最大速度

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

MaxVel　设置轴的最大速度，必须是正数。

返回值：错误代码

ERROR_CODE NV_AxisGetMaxVel(UInt16 ConnectNo,UInt16 AxisNo,Double[] pMaxVel ,UInt16 Count)

功　能：获取轴的最大速度

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

PMaxVel　获取轴的最大速度

Count　从 AxisNo 号开始一次性读取后续 Count 个轴的值

返回值：错误代码

ERROR_CODE　NV_AxisSetMaxAcc(UInt16　ConnectNo,UInt16　AxisNo,Double MaxAcc)

功　能：设置轴的最大加速度

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

MaxAcc　设置轴的最大加速度，必须是正数。

返回值：错误代码

ERROR_CODE　NV_AxisGetMaxAcc(UInt16　ConnectNo,UInt16　AxisNo,Double[] pMaxAcc,UInt16 Count)

功　能：获取轴的最大加速度

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

pMaxAcc　获取轴的最大加速度

Count　从 AxisNo 号开始一次性读取后续 Count 个轴的值

返回值：错误代码

ERROR_CODE NV_AxisGetErrorStatus(UInt16 ConnectNo, UInt16 AxisNo, UInt32[] pValue, UInt16 Count)

功　能：获取轴的错误状态

参　数：ConnectNo　　板卡连接编号[0,15]

AxisNo　　轴序号

pValue　　获取轴的错误状态值

Count　从 AxisNo 号开始一次性读取后续 Count 个轴的值

返回值：错误代码

轴错误状态值表示的含义如下表：

轴错误状态值	表示的含义
-1	无错误
0	伺服报警
1	硬件正限位
2	硬件负限位
3	软件正限位
4	软件负限位
5	跟随误差超出设置范围报警
6	圆形限位报警
7	龙门超差报警

5.4.2 轴规划状态

ERROR_CODE NV_AxisSetPrfPos(UInt16 ConnectNo,UInt16 AxisNo,Double Value)

功　能：修改轴的当前规划位置

参　数：ConnectNo　　板卡连接编号[0,15]

AxisNo　　轴序号

Value　修改轴的当前位置

返回值：错误代码

ERROR_CODE NV_AxisGetPrfPos(UInt16 ConnectNo,UInt16 AxisNo,Double[] pValue,UInt16 Count)

功　能：获取轴的当前规划位置

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

pValue 获取轴的当前位置值

Count 从 AxisNo 号开始一次性读取后续 Count 个轴的值

返回值：错误代码

ERROR_CODE NV_AxisGetPrfVel(UInt16 ConnectNo,UInt16 AxisNo,Double[] pValue,UInt16 Count)

功　能：获取轴的当前规划速度

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

pValue 获取轴的当前速度值

Count 从 AxisNo 号开始一次性读取后续 Count 个轴的值

返回值：错误代码

ERROR_CODE NV_AxisGetPrfAcc(UInt16 ConnectNo,UInt16 AxisNo,Double[] pValue,UInt16 Count)

功　能：获取轴的当前加速度

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

pValue 获取轴的当前加速度值

Count 从 AxisNo 号开始一次性读取后续 Count 个轴的值

返回值：错误代码

ERROR_CODE NV_AxisGetPrfMode(UInt16 ConnectNo,UInt16 AxisNo,UInt16[] pValue,UInt16 Count)

功　能：获取轴的运动模式

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

pValue 获取轴的运动模式

Count　从 AxisNo 号开始一次性读取后续 Count 个轴的值

返回值：错误代码

5.4.3 轴运动状态

ERROR_CODE NV_AxisCheckDone (UInt16 ConnectNo,UInt16 AxisNo,UInt16[] pStatus,UInt16 Count)

功　能：检测轴状态

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

pStatus　状态： 0：指定轴正在运行，1：指定轴已停止

Count　返回轴运动状态的数量

返回值：错误代码

注　意：此函数适用于单轴、PVT 运动

ERROR_CODE NV_AxisGetStatus(UInt16 ConnectNo,UInt16 AxisNo,UInt32[] pStatus,UInt16 Count)

功　能：获取轴运动的各种状态

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

pStatus　返回的轴运动状态

Count　返回轴运动状态的数量

返回值：错误代码

轴状态 pStatus 各个 Bit 位的作用：

Bit	轴状态说明
0	stopped.已停止
1	inp.等待 inp 信号中
2	homed.已完成回零动作
3	wating reach.等待到位条件满足，指令位置与编码位置在误差范围。
4	In Gantry.处于龙门状态
5	In Gear.处于电子齿轮的从轴状态
6	In Cam.处于电子凸轮的从轴状态

ERROR_CODE NV_AxisSetReached(UInt16 ConnectNo,UInt16 AxisNo,UInt16 Enable,Double ErrorBand,Double HoldTime)

功　能：设置轴到位检测：规划停止后检测规划位置与编码器反馈位置的差值，误差范围之内认为到位

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

Enable 使能：0 – 禁止； 1 – 使能；

ErrorBand 差值范围，单位 unit

HoldTime　最少停留时间，单位 us

返回值：错误代码

ERROR_CODE NV_AxisGetReached(UInt16 ConnectNo,UInt16 AxisNo,ref UInt16 pEnable,ref Double pErrorBand,ref Double pHoldTime)

功　能：获取轴到位检测状态

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

pEnable　返回使能状态

pErrorBand 返回差值范围

pHoldTime　返回最少停留时间

返回值：错误代码

5.4.4 轴编码器参数

ERROR_CODE NV_AxisSetEncMapping(UInt16 ConnectNo,UInt16 AxisNo,UInt16 MapEncType,UInt16 MapEncNo)

功　能：设置轴编码器映射关系,实现误差带检测、超差保护等场合使用。

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

MapEncType　映射编码器通道类型

0：所有编码器；

1：差分编码器；

2：单端编码器；

3：手轮编码器；

4：内部脉冲计数器

MapEncNo 映射编码器通道号

返回值：错误代码

ERROR_CODE NV_AxisGetEncMapping(UInt16 ConnectNo,UInt16 AxisNo,ref UInt16 pMapEncType,ref UInt16 pMapEncNo)

功　能：获取轴编码器映射关系

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

pMapEncType　返回映射编码器通道类型

pMapEncNo　返回映射编码器通道号

返回值：错误代码

ERROR_CODE NV_AxisSetEncUnit(UInt16 ConnectNo,UInt16 AxisNo, Double

EncUnit)

功　能：设置轴编码器映射关系,实现误差带检测、超差保护等场合使用。

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

EncUnit 设置编码器的脉冲当量值

返回值：错误代码

ERROR_CODE NV_AxisGetEncUnit(UInt16 ConnectNo,UInt16 AxisNo,Double[] pMapEncType,UInt16 pMapEncNo)

功　能：获取轴编码器映射关系

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

pEncUnit 返回设置的编码器脉冲当量值

pMapEncNo　返回编码器的数量

返回值：错误代码

ERROR_CODE NV_AxisSetEncPos(UInt16 ConnectNo,UInt16 AxisNo,Double Value)

功　能：设置轴编码器位置值，便于直接采用轴号去操作编码器

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

Value　设置编码器位置值

返回值：错误代码

ERROR_CODE NV_AxisGetEncPos(UInt16 ConnectNo,UInt16 AxisNo,ref Double[] pValue,UInt16 Count)

功　能：获取轴编码器位置值，便于直接采用轴号去操作编码器

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　　轴序号

pValue 返回编码器位置值

Count　　需要获取的数量

返回值：错误代码

NV_AxisIoSetDoBit5.4.5 轴 IO 操作

ERROR_CODE NV_AxisIoSetMapping(UInt16 ConnectNo,UInt16 AxisNo, UInt16 DiType, UInt16 MapDiType, UInt16 MapDiIndex)

功　能：设置轴 IO 映射关系

参　数：ConnectNo　控制卡卡号

AxisNo　　轴序号

DiType 指定轴的 IO 信号类型：

0：负限位信号

1：正限位信号

2：原点信号

3：急停信号

4：减速停止信号

5：伺服报警信号

6：伺服准备信号

7：伺服到位信号

8：EZ 输入信号

MapDiType　　轴 IO 映射输入口类型：

0：通用输入端口

1：高速输入端口

2：原点输入端口

3：负限位输入端口

4：正限位输入端口

5：编码器 Z 相输入端口

6：伺服报警输入端口

7：伺服到位输入端口

8：伺服准备输入端口

65535(0xFFFF)：无效输入口，不关联任何输入口

MapDiIndex 轴 IO 映射输入口编号

返回值：错误代码

ERROR_CODE NV_AxisIoGetMapping(UInt16 ConnectNo,UInt16 AxisNo, UInt16 DiType, ref UInt16 pMapDiType, ref UInt16 pMapDiIndex)

功 能：设置轴 IO 映射关系

参 数：ConnectNo 板卡连接编号[0,15]

AxisNo 轴序号

DiType 轴 IO 信号类型

pMapDiType 返回轴 IO 映射类型

pMapDiIndex 返回轴 IO 映射索引号

返回值：错误代码

ERROR_CODE NV_AxisIoSetEnable(UInt16 ConnectNo, UInt16 AxisNo, UInt32 Value)

功 能：设置轴 IO 专用使能

参 数：ConnectNo 板卡连接编号[0,15]

AxisNo 轴序号

Value 轴 IO 使能信号，按照 bit 位设置，0 - 禁用，1 - 启用

返回值：错误代码

说明：函数 NV_AxisIoSetEnable、NV_AxisIoGetEnable、NV_AxisIoSetDiLogic、

NV_AxisIoGetDiLogic、NV_AxisIoGetDi、NV_AxisIoGetDiStatus 信号位定义，如下表所示：

Bit 位	信号名称
0	ELN 负限位信号
1	ELP 正限位信号
2	ORG 原点信号
3	EMG 急停信号
4	DEC 减速停止信号
5	ALM 伺服报警信号
6	RDY 伺服准备信号
7	INP 伺服到位信号
8	EZ 编码器 Z 相信号

ERROR_CODE NV_AxisIoGetEnable(UInt16 ConnectNo, UInt16 AxisNo, UInt32[] pValue, UInt16 Count)

功　能：读取轴 IO 专用使能

参　数：ConnectNo 板卡连接编号[0,15]

AxisNo　　轴序号

pValue　　返回轴 IO 信号使能值。

Count 返回信号状态的轴数

ERROR_CODE NV_AxisIoSetDiLogic(UInt16 ConnectNo, UInt16 AxisNo, UInt32 Value)

功　能：设置轴专用输入信号的有效电平

参　数：ConnectNo 板卡连接编号[0,15]

AxisNo　　轴序号

Value 轴专用输入信号有效电平，按照 bit 位设置每种输入信号的有效电

平，1 - 高电平，0 - 低电平

返回值：错误代码

ERROR_CODE NV_AxisIoGetDiLogic(UInt16 ConnectNo, UInt16 AxisNo, UInt32[] pValue, UInt16 Count)

功　能：读取轴专用输入信号的有效电平

参　数：ConnectNo 板卡连接编号[0,15]

AxisNo　　轴序号

pValue　返回轴专用输入信号有效电平

Count　　需要读取的轴数量

ERROR_CODE NV_AxisIoGetDi(UInt16 ConnectNo,UInt16 AxisNo, UInt32[] pValue,UInt16 Count)

功　能：读取轴专用输入信号（按照实际物理端口读取）

参　数：ConnectNo 板卡连接编号[0,15]

AxisNo　　轴序号

pValue　返回轴专用信号的输入电平状态：0 - 低电平；1 - 高电平

Count　需要读取的轴数量

返回值：错误代码

ERROR_CODE NV_AxisIoGetDiStatus(UInt16 ConnectNo,UInt16 AxisNo, UInt32[] pValue,UInt16 Count)

功　能：读取轴专用输入信号（按照 IO 映射后的端口读取）

参　数：ConnectNo 板卡连接编号[0,15]

AxisNo　　轴序号

pValue　返回轴专用信号的输入有效状态：1 - 有效；0 - 无效

Count　　需要读取的轴数量

返回值：错误代码

ERROR_CODE NV_AxisIoSetDoBit(UInt16 ConnectNo, UInt16 AxisNo, UInt16 DoType, UInt16 Value)

功　能：设置轴专用输出信号

参　数：ConnectNo　控制卡卡号[0,15]

AxisNo　轴序号

DoType　输出类型：0　伺服使能输出端口；1　伺服报警清除输出端口

Value　轴专用输出状态值

返回值：错误代码

ERROR_CODE NV_AxisIoGetDoBit(UInt16 ConnectNo, UInt16 AxisNo, UInt16 DoType, UInt16[] pValue, UInt16 Count)

功　能：读取轴专用输出信号

参　数：ConnectNo 板卡连接编号[0,15]

AxisNo　轴序号

DoType　输出类型：0 伺服使能输出端口；1 伺服报警清除输出端口

pValue　返回轴专用输出口状态值

Count 需要读取的轴数量

返回值：错误代码

ERROR_CODE NV_AxisIoSetOrgLogic(UInt16 ConnectNo, UInt16 AxisNo, UInt16 Value)

功　能：设置轴原点输入信号的有效电平

参　数：ConnectNo 板卡连接编号[0,15]

AxisNo　轴序号

Value 轴原点输入有效电平值

返回值：错误代码

ERROR_CODE NV_AxisIoGetOrgLogic(UInt16 ConnectNo, UInt16 AxisNo, UInt16[] pValue, UInt16 Count)

功　能：读取轴原点输入信号的有效电平

参　数：ConnectNo 板卡连接编号[0,15]

AxisNo　　轴序号

pValue　　返回轴原点输入有效电平

Count 需要读取的数量

返回值：错误代码

ERROR_CODE NV_AxisIoSetEzLogic(UInt16 ConnectNo, UInt16 AxisNo, UInt16 Value)

功　能：设置轴 EZ 输入信号的有效电平

参　数：ConnectNo 板卡连接编号[0,15]

AxisNo　　轴序号

Value 轴 EZ 输入有效电平值

返回值：错误代码

ERROR_CODE NV_AxisIoGetEzLogic(UInt16 ConnectNo, UInt16 AxisNo, UInt16[] pValue, UInt16 Count)

功　能：读取轴 EZ 输入有效电平

参　数：ConnectNo 板卡连接编号[0,15]

AxisNo　　轴序号

pValue　　返回轴 EZ 输入有效电平

Count 需要读取的数量

返回值：错误代码

ERROR_CODE NV_AxisIoSetHardLimit(UInt16 ConnectNo, UInt16 AxisNo, UInt16 Dir, UInt16 Enable, UInt16 Logic, UInt16 StopMode)

功　能：设置轴的硬件限位功能

参　数：ConnectNo 板卡连接编号[0,15]

AxisNo　　轴序号

Dir　方向：0-负向限位，1-正向限位

Enable　　设置是否启用

Logic 有效电平：0-低电平，1-高电平

StopMode 设置停止方式：0：减速停止；1：立即停止

返回值：错误代码

ERROR_CODE NV_AxisIoGetHardLimit(UInt16 ConnectNo, UInt16 AxisNo, UInt16 Dir, ref UInt16 pEnable, ref UInt16 Logic, ref UInt16 pStopMode)

功　能：获取轴的硬件限位功能状态

参　数：ConnectNo 板卡连接编号[0,15]

AxisNo　　轴序号

Dir　方向：0-负向限位，1-正向限位

pEnable　返回启用状态

Logic 返回有效电平：0-低电平，1-高电平

pStopMode 返回停止方式：0：减速停止；1：立即停止

返回值：错误代码

ERROR_CODE NV_AxisIoSetDecStop(UInt16 ConnectNo, UInt16 AxisNo, UInt16 Enable, UInt16 Logic)

功　能：设置轴的硬件减速停止功能

参　数：ConnectNo 板卡连接编号[0,15]

AxisNo　　轴序号

Enable　设置是否启用

Logic 有效电平：0-低电平，1-高电平

返回值：错误代码

ERROR_CODE NV_AxisIoGetDecStop(UInt16 ConnectNo, UInt16 AxisNo, ref UInt16 pEnable, ref UInt16 Logic)

功　能：获取轴的硬件减速停止功能状态

参　数：ConnectNo 板卡连接编号[0,15]

AxisNo　　轴序号

pEnable　返回启用状态

Logic 返回有效电平：0-低电平，1-高电平

返回值：错误代码

ERROR_CODE NV_AxisIoSetEmgStop(UInt16 ConnectNo, UInt16 AxisNo, UInt16 Enable, UInt16 Logic)

功　能：设置轴的硬件急停功能

参　数：ConnectNo 板卡连接编号[0,15]

AxisNo　　轴序号

Enable　设置是否启用

Logic 有效电平：0-低电平，1-高电平

返回值：错误代码

ERROR_CODE NV_AxisIoGetEmgStop(UInt16 ConnectNo, UInt16 AxisNo, ref UInt16 pEnable, ref UInt16 Logic)

功　能：获取轴的硬件急停功能状态

参　数：ConnectNo 板卡连接编号[0,15]

AxisNo　　轴序号

pEnable　返回启用状态

Logic 返回有效电平：0-低电平，1-高电平

返回值：错误代码

ERROR_CODE NV_AxisIoSetAlarm(UInt16 ConnectNo, UInt16 AxisNo, UInt16 Enable, UInt16 Logic)

功　能：设置伺服报警功能

参　数：ConnectNo 板卡连接编号[0,15]

AxisNo　　轴序号

Enable　设置是否启用

Logic 有效电平：0-低电平，1-高电平

返回值：错误代码

ERROR_CODE NV_AxisIoGetAlarm(UInt16 ConnectNo, UInt16 AxisNo, ref UInt16 pEnable, ref UInt16 Logic)

功　能：获取伺服报警功能状态

参　数：ConnectNo 板卡连接编号[0,15]

AxisNo　　轴序号

pEnable　返回启用状态

Logic 返回有效电平：0-低电平，1-高电平

返回值：错误代码

ERROR_CODE NV_AxisIoSetInp(UInt16 ConnectNo, UInt16 AxisNo, UInt16 Enable, UInt16 Logic)

功　能：设置伺服到位功能

参　数：ConnectNo 板卡连接编号[0,15]

AxisNo　　轴序号

Enable　　设置是否启用

Logic 有效电平：0-低电平，1-高电平

返回值：错误代码

ERROR_CODE NV_AxisIoGetInp(UInt16 ConnectNo, UInt16 AxisNo, ref UInt16 pEnable, ref UInt16 Logic)

功　能：获取伺服到位功能状态

参　数：ConnectNo 板卡连接编号[0,15]

AxisNo　　轴序号

pEnable　返回启用状态

Logic 返回有效电平：0-低电平，1-高电平

返回值：错误代码

ERROR_CODE NV_AxisIoSetRdy(UInt16 ConnectNo, UInt16 AxisNo, UInt16 Enable, UInt16 Logic)

功　能：设置伺服准备好功能

参　数：ConnectNo 板卡连接编号[0,15]

AxisNo　　轴序号

Enable　　设置是否启用

Logic 有效电平：0-低电平，1-高电平

返回值：错误代码

ERROR_CODE NV_AxisIoGetRdy(UInt16 ConnectNo, UInt16 AxisNo, ref UInt16 pEnable, ref UInt16 Logic)

功　能：获取伺服准备好功能状态

参　数：ConnectNo 板卡连接编号[0,15]

AxisNo　　轴序号

pEnable　返回启用状态

Logic 返回有效电平：0-低电平，1-高电平

返回值：错误代码

5.5 硬件资源相关函数

5.5.1 通用 IO 操作

通用数字输入

ERROR_CODE　NV_IoGetDi(UInt16　ConnectNo,UInt16　DiGrpIndex,UInt32[] pValue,UInt16 Count)

功　能：读取一批输入信号

参　数：ConnectNo　　板卡连接编号[0,15]

DiGrpIndex　输入口编号

pValue　　实际的读数

Count　　从该组开始读取的组数量

返回值：错误代码

ERROR_CODE　NV_IoGetDiBit(UInt16　ConnectNo,UInt16　DiIndex,UInt16[] pValue,UInt16 Count)

功　能：读取单个输入状态

参　数：ConnectNo　　板卡连接编号[0,15]

DiIndex　输入口编号

pValue　　实际的输入读数

Count 需要读取的数量

返回值：错误代码

ERROR_CODE NV_IoSetDiCountMode(UInt16 ConnectNo, UInt16 Channel, UInt16 DiIndex,UInt16 CountMode)

功 能：设置输入的计数模式。

参 数：ConnectNo 板卡连接编号[0,15]

Channel 计数器通道号：0-15

DiIndex 输入的序号

CountMode 计数模式：0-3

返回值：错误代码

计数模式 CountMode 的如下：

计数模式	说明
0	禁用
1	上升沿计数
2	下降沿计数
3	任意沿计数

ERROR_CODE NV_IoGetDiCountMode(UInt16 ConnectNo, UInt16 Channel,ref UInt16 pDiIndex,ref UInt16 pCountMode)

功 能：获取输入的计数模式。

参 数：ConnectNo 板卡连接编号[0,15]

Channel 计数器通道号：0-15

pDiIndex 输入的序号

pCountMode 返回计数模式

返回值：错误代码

ERROR_CODE NV_IoSetDiCountValue(UInt16 ConnectNo, UInt16 Channel,UInt32 CountValue)

功　能：修改当前输入的计数值。

参　数：ConnectNo　板卡连接编号[0,15]

Channel　计数器通道号：0-15

CountValue　要修改的计数值

返回值：错误代码

ERROR_CODE NV_IoGetDiCountValue(UInt16 ConnectNo, UInt16 Channel,UInt32[] pCountValue, UInt16 Count)

功　能：获取当前输入的计数值。

参　数：ConnectNo　板卡连接编号[0,15]

Channel　计数器通道号：0-15

pCountValue　返回当前的计数值

Count　读取的数量

返回值：错误代码

ERROR_CODE NV_IoSetDiFilter(UInt16 ConnectNo,Double FilterTime)

功　能：设置输入的滤波时间，用来消除毛刺电平。

参　数：ConnectNo　板卡连接编号[0,15]

FilterTime　滤波时间，单位：us，范围 0-10000

返回值：错误代码

ERROR_CODE NV_IoGetDiFilter(UInt16 ConnectNo,ref Double pFilterTime)

功　能：获取输入的滤波时间

参　数：ConnectNo　板卡连接编号[0,15]

pFilterTime　返回滤波时间，单位：us

返回值：错误代码

通用数字输出

ERROR_CODE NV_IoSetDo(UInt16 ConnectNo,UInt16 DoGrpIndex,UInt32 Value)

功　能：设置批量输出。

参数：ConnectNo 板卡连接编号[0,15]

DoGrpIndex 组下标

Value　　输出的数值

返回值：错误代码

ERROR_CODE NV_IoGetDo(UInt16 ConnectNo,UInt16 DoGrpIndex,UInt32[] pValue,UInt16 Count)

功　能：获取批量输出状态。

参数：ConnectNo　板卡连接编号[0,15]

DoGrpIndex 组下标

pValue　　输出的数值

Count　　从该组开始读取的组数量

返回值：错误代码

ERROR_CODE NV_IoSetDoMask(UInt16 ConnectNo,UInt16 DoGrpIndex,UInt32 Value,UInt32 DoMask)

功　能：有条件地设置批量输出。

参数：ConnectNo　板卡连接编号[0,15]

DoGrpIndex 组下标

Value　　输出的数值

DoMask　　设置输出的有效通道，bit 位为 1 则输出，0 则不改变

返回值：错误代码

ERROR_CODE NV_IoSetDoBit(UInt16 ConnectNo,UInt16 DoIndex,UInt16 Value)

功　能：设置单个输出。

参数：ConnectNo　板卡连接编号[0,15]

DoIndex　　输出的引脚序号

Value　　输出的电平

返回值：错误代码

ERROR_CODE NV_IoGetDoBit(UInt16 ConnectNo,UInt16 DoIndex,UInt16[] pValue,UInt16 Count)

功　能：获取单个输出状态。

参数：ConnectNo　板卡连接编号[0,15]

DoIndex　　输出的引脚序号

pValue　　输出的数值

Count　　从该输出开始读取的数量

返回值：错误代码

ERROR_CODE NV_IoSetDoBitReverse(UInt16 ConnectNo,UInt16 Channel,UInt16 DoIndex,UInt16 Value,Double ReverseTime)

功　能：设置单个输出按时翻转

参数：ConnectNo　板卡连接编号[0,15]

Channel 输出通道号：0-7

DoIndex　　输出口编号

Value　　输出电平：0-低电平，1-高电平，2-翻转电平

ReverseTime 翻转时间，单位 s，范围 0-10

返回值：错误代码

ERROR_CODE NV_IoSetDoBitPulse(UInt16 ConnectNo,UInt16 Channel,UInt16 DoIndex,UInt16 FirstLevel,Double FirstTime,Double SecondTime,UInt32 PulseNum)

功　能：控制输出信号产生脉冲波形

参数：ConnectNo　板卡连接编号[0,15]

Channel 输出通道号：0-7

DoIndex　　输出口编号

FirstLevel　输出起始电平：0-低电平，1-高电平，2-翻转电平

FirstTime 输出起始电平维持时间，单位：s，范围 0-10

SecondTime 输出相反电平维持时间，单位：s，范围 0-10

PulseNum　　输出脉冲的个数

返回值：错误代码

注意：电平维持时间，最短 0.001s 以上。

ERROR_CODE NV_IoClrDoBitPulse(UInt16 ConnectNo,UInt16 Channel)

功　能：停止输出信号产生脉冲波形

参数：ConnectNo　板卡连接编号[0,15]

Channel 输出通道号：0-7

返回值：错误代码

5.5.2 高速 IO 操作

ERROR_CODE NV_HSIoGetDi(UInt16 ConnectNo, ref UInt32 pValue)

功　能：读取高速 IN

参　数：ConnectNo　板卡连接编号[0,15]

pValue　　高速 IN 的值

返回值：错误代码

ERROR_CODE NV_HSIoGetDiBit(UInt16 ConnectNo, UInt16 HSDiIndex,UInt16[] pValue,UInt16 Count)

功　能：读取单个输入

参　数：ConnectNo　板卡连接编号[0,15]

HSDiIndex　高速输入的引脚序号

pValue　实际的高速输入读数

Count　要读取的数量

返回值：错误代码

高速输出

ERROR_CODE NV_HSIoSetDo(UInt16 ConnectNo, UInt32 Value)

功　能：设置高速输出的值。

参　数：ConnectNo 板卡连接编号[0,15]

Value　高速输出的值

返回值：错误代码

ERROR_CODE NV_HSIoSetDoMask(UInt16 ConnectNo, UInt32 Value， UInt32 DoMask)

功　能：设置高速输出的值。

参　数：ConnectNo 板卡连接编号[0,15]

Value　高速输出的值

DoMask　设置高速输出的有效通道，bit 位为 1 则输出，0 则不改变

返回值：错误代码

ERROR_CODE NV_HSIoGetDo (UInt16 ConnectNo, ref UInt32 pValue)

功　能：读取高速输出的值

参　数：ConnectNo 板卡连接编号[0,15]

pValue　　高速输出的值

返回值：错误代码

ERROR_CODE NV_HSIoSetDoBit(UInt16 ConnectNo, UInt16 HSDoIndex,UInt16 Value)

功　能：设置单个高速输出。

参　数：ConnectNo 板卡连接编号[0,15]

HSDoIndex 高速输出的引脚序号

Value　高速输出的电平

返回值：错误代码

ERROR_CODE NV_HSIoGetDoBit(UInt16 ConnectNo, UInt16 HSDoIndex,UInt16[] pValue,UInt16 Count)

功　能：获取单个高速输出状态。

参　数：ConnectNo　板卡连接编号[0,15]

HSDoIndex　高速输出的引脚序号

pValue　　输出的数值

Count　　从该输出开始读取的数量

返回值：错误代码

ERROR_CODE NV_HSIoSetDiFilter(UInt16 ConnectNo,Double FilterTime)

功　能：设置输入的滤波时间，用来消除毛刺电平。

参　数：ConnectNo　板卡连接编号[0,15]

FilterTime　滤波时间，单位：us，范围 0-10000

返回值：错误代码

ERROR_CODE NV_HSIoGetDiFilter(UInt16 ConnectNo,ref Double pFilterTime)

功　能：获取输入的滤波时间

参　数：ConnectNo　　板卡连接编号[0,15]

pFilterTime　　返回滤波时间，单位：us

返回值：错误代码

5.5.3 编码器操作

ERROR_CODE NV_EncoderSetMode(UInt16 ConnectNo,UInt16 Channel,UInt16 EncMode,UInt16 EncDir，UInt16 EncDivision)

功　能：设置编码器的工作模式

参　数：ConnectNo　　板卡连接编号[0,15]

Channel　　编码器序号

EncMode　计数模式：

0：AB 相

1：脉冲/方向

2：方向/脉冲

3：双脉冲方式

EncDir　　计数方向:

0：　正向计数；

1：　反向计数

EncDivision 计数分频：

0/1：　1 分频，原始计数

2：2 分频

4：4 分频

8：8 分频

返回值：错误代码

ERROR_CODE NV_EncoderGetMode(UInt16 ConnectNo,UInt16 Channel,ref UInt16 pEncMode,ref UInt16 pEncDir,ref UInt16 pEncDivision)

功　能：获取编码器的工作模式

参　数：ConnectNo　板卡连接编号[0,15]

Channel　编码器序号

pEncMode 返回计数模式

pEncDir　返回计数方向

pEncDivision 返回计数分频

返回值：错误代码

ERROR_CODE NV_EncoderSetFilter(UInt16 ConnectNo,UInt16 Channel,Double FilterTime)

功　能：设置编码器的滤波时间。

参　数：ConnectNo　板卡连接编号[0,15]

Channel　编码器序号

FilterTime　滤波时间，单位 us，范围 0-40

返回值：错误代码

ERROR_CODE NV_EncoderGetFilter(UInt16 ConnectNo,UInt16 Channel,Double[] pFilterTime, UInt16 Count)

功　能：获取编码器的滤波时间。

参　数：ConnectNo　板卡连接编号[0,15]

Channel　编码器序号

pFilterTime 返回滤波时间，单位 us，范围 0-40

Count　返回编码器滤波参数的数量

返回值：错误代码

ERROR_CODE NV_EncoderSetPulsePos(UInt16 ConnectNo,UInt16 Channel,Int32 Value)

功　能：设置编码器的计数值

参　数：ConnectNo　板卡连接编号[0,15]

Channel　编码器序号

Value 计数值（脉冲数）

返回值：错误代码

ERROR_CODE NV_EncoderGetPulsePos(UInt16 ConnectNo,UInt16 Channel,Int32[] pValue,UInt16 Count)

功　能：读取编码器的计数值

参　数：ConnectNo　板卡连接编号[0,15]

Channel　编码器序号

pValue　返回当前计数值

Count　获取值的数量

返回值：错误代码

5.5.4 模拟量操作

模拟输入

ERROR_CODE NV_ADGetValue(UInt16 ConnectNo,UInt16 Channel, Int32[] pValue, UInt16 Count)

功　能：读取单个通道多个 AD 值

参　数：ConnectNo 板卡连接编号[0,15]

Channel　AD 的通道号：0-3

pValue　返回读取的采样数据值

Count　连续读取 Count 个采样值

返回值：错误代码

模拟输出

ERROR_CODE NV_DASetEnable(UInt16 ConnectNo, UInt16 Channel, UInt16 Enable)

功　能：设置 DA 使能。

参　数：ConnectNo 板卡连接编号[0,15]

Channel　通道号：0-1

Enable 使能：0 - 关闭； 1 - 使能

返回值：错误代码

ERROR_CODE NV_DAGetEnable (UInt16 ConnectNo, UInt16 Channel, UInt16[] pEnable, UInt16 Count)

功　能：读取 DA 使能

参　数：ConnectNo　板卡连接编号[0,15]

Channel 通道号：0-1

pEnable　返回使能值

Count　读取 Count 个通道使能状态

返回值：错误代码

ERROR_CODE NV_DASetValue(UInt16 ConnectNo,UInt16 Channel, Int32 Value)

功　能：设置 DA 值。

参　数：ConnectNo　板卡连接编号[0,15]

Channel 通道号：0-1

Value　设置 DA 输出值

返回值：错误代码

ERROR_CODE NV_DAGetValue(UInt16 ConnectNo,UInt16 Channel, Int32[] pValue, UInt16 Count)

功　能：读取 DA 值

参　数：ConnectNo　板卡连接编号[0,15]

Channel 通道号：0-1

pValue　回读 DA 值

Count　连续读取 Count 个值

返回值：错误代码

5.5.5 手轮操作

ERROR_CODE NV_HandwheelSetEncMapping(UInt16 ConnectNo, UInt16 MapEncType, UInt16 MapEncNo, Double SmoothTime)

功　能：设置手轮编码器映射关系。

参　数：ConnectNo　板卡连接编号[0,15]

MapEncType　映射编码器通道类型

0：所有编码器（系统自动排序）；

1：差分编码器；

2：单端编码器；

3：手轮编码器；

4：内部脉冲计数器

MapEncNo 映射编码器通道号，编号从 0 开始

SmoothTime　平滑时间，单位 s，范围 0-1

返回值：错误代码

ERROR_CODE NV_HandwheelGetEncMapping(UInt16 ConnectNo, ref UInt16 pMapEncType, ref UInt16 pMapEncNo, ref Double pSmoothTime)

功　能：获取手轮编码器映射关系

参　数：ConnectNo　　板卡连接编号[0,15]

pMapEncType 返回映射编码器通道类型

pMapEncNo　返回映射编码器通道号

pSmoothTime　　平滑时间，单位 s，范围 0-1

返回值：错误代码

ERROR_CODE NV_HandwheelSetEncoderPos(UInt16 ConnectNo, Int32 Value)

功　能：设置手轮对应的编码器值

参　数：ConnectNo　　板卡连接编号[0,15]

Value 计数值

返回值：错误代码

ERROR_CODE NV_HandwheelGetEncoderPos(UInt16 ConnectNo, ref Int32 pValue)

功　能：读取手轮对应的编码器值

参　数：ConnectNo　　板卡连接编号[0,15]

pValue　　返回当前计数值

返回值：错误代码

ERROR_CODE NV_HandwheelGetHardInStatus (UInt16 ConnectNo, ref UInt16 pMode, ref UInt16 pAxisSelState, ref UInt16 pRatioSelState, ref UInt16 pEmgState)

功　能：获取手轮上的输入信号状态

参　数：ConnectNo　　板卡连接编号[0,15]

pMode　　控制模式：0 - 硬件控制；1 – 软件控制

pAxisSelState 轴选信号（DB26 端口中输入 1-6 信号），有效位值为 1

pRatioSelState　倍选信号（DB26 端口中输入 7-9 信号），有效位值为 1

pEmgState　　急停信号（DB26 端口中输入 10 信号），有效位值为 1

返回值：错误代码

ERROR_CODE NV_HandwheelSetHardAxisSel(UInt16 ConnectNo,UInt16 AxisSel, UInt16 AxisNo)

功　能：设置手轮硬件轴选参数

参　数：ConnectNo　板卡连接编号[0,15]

AxisSel　手轮轴选编号

AxisNo　控制轴号

返回值：错误代码

ERROR_CODE NV_HandwheelGetHardAxisSel(UInt16 ConnectNo,UInt16 AxisSel, UInt16[] pAxisNo,UInt16 Count)

功　能：获取手轮硬件轴选参数

参　数：ConnectNo　板卡连接编号[0,15]

AxisSel　手轮轴选编号

pAxisNo　返回控制轴号

Count　获取值的数量

返回值：错误代码

ERROR_CODE NV_HandwheelSetHardRatioSel(UInt16 ConnectNo,UInt16 AxisSel, Double pRatio)

功　能：设置手轮硬件倍选参数

参　数：ConnectNo　板卡连接编号[0,15]

RatioSel　手轮倍选编号

Ratio　三个手轮倍率，正数表示默认方向，负数表示与默认方向反向

返回值：错误代码

ERROR_CODE NV_HandwheelGetHardRatioSel(UInt16 ConnectNo,UInt16 RatioSel,

Double[] pRatio,UInt16 Count)

功　能：获取手轮硬件倍选参数

参　数：ConnectNo　板卡连接编号[0,15]

RatioSel　手轮倍选编号

pRatio　手轮倍率，正数表示默认方向，负数表示与默认方向反向

Count　获取值的数量

返回值：错误代码

手轮软件控制参数

ERROR_CODE NV_HandwheelSetSoftParam(UInt16 ConnectNo,UInt16 AxisNum, UInt16[] pAxisList, Double[] pRatioList)

功　能：设置手轮软件控制参数

参　数：ConnectNo　板卡连接编号[0,15]

AxisNum　轴数

pAxisList　控制轴列表

pRatioList　倍率列表

返回值：错误代码

ERROR_CODE NV_HandwheelGetSoftParam(UInt16 ConnectNo,ref UInt16 pAxisNum, UInt16[] pAxisList, Double[] pRatioList)

功　能：获取手轮软件控制参数

参　数：ConnectNo　板卡连接编号[0,15]

AxisNum　返回轴数

pAxisList　返回控制轴列表

pRatioList　返回倍率列表

返回值：错误代码

手轮运动

ERROR_CODE NV_HandwheelMove(UInt16 ConnectNo,UInt16 Mode)

功　能：启动手轮运动

参　数：ConnectNo　　板卡连接编号[0,15]

　　Mode　　控制方式：0-采用手轮硬件控制，1-采用软件控制

返回值：错误代码

注意：启动手轮运动前，需要判断轴处于正常的空闲状态（不能处于报警状态，运动状态），否则切换为手轮运动会失败。

ERROR_CODE NV_HandwheelStop(UInt16 ConnectNo)

功　能：退出手轮运动

参　数：ConnectNo　　板卡连接编号[0,15]

返回值：错误代码

5.6 单轴基本运动

5.6.1 回零运动

回零结构体：

```
typedef struct HomeProfileStruct
{
    Int16 HomeWay;//回零方式：0 - 本地回零；1 - 总线回零
    Int16 HomeMode;    //回零模式，详细参考 DS402 回零说明
    Int16 OffsetMode;    //回零的偏移模式：0-不偏移,直接设置为偏移位置;1-清零后偏移;2-偏移后清零
    Double Offset;        //回零的偏移值
    Double LowSpeed;   //回零的最低速度
    Double HighSpeed;   //回零的最高速度
```

```
    Double HomeAcc;    //回零的加速度
}HomeProfileStruct_t, * PHomeProfileStruct_t;
```

ERROR_CODE NV_HomeSetProfile(UInt16 ConnectNo,UInt16 AxisNo, HomeProfileStruct_t Profile)

功　能：设置回零的参数

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

Profile 回零参数

返回值：错误代码

ERROR_CODE NV_HomeGetProfile(UInt16 ConnectNo,UInt16 AxisNo, ref PHomeProfileStruct_t pProfile)

功　能：获取回零的参数

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

pProfile 返回回零参数

返回值：错误代码

ERROR_CODE NV_HomeStatus(UInt16 ConnectNo,UInt16 AxisNo,ref UInt16 pState,ref UInt16 pStep)

功　能：获取回零的状态

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

pState　返回本地回零状态

0：未回零完成

1：已回零完成

pStep　　返回当前的回零阶段

返回值：错误代码

回零阶段 pStep 的具体定义：

pStep 编号	说明
0	静止状态，未启动回零
1	前向快速寻找零点
2	触碰限位后退找零点，或在零点状态下回找零点边缘
3	前向快速寻找 EZ
4	后退快速寻找 EZ
5	前向快速寻找 LIMIT
6	后退快速寻找 LIMIT
7	慢速前向寻找零点边缘
8	慢速后退寻找零点边缘
9	慢速前向寻找 EZ 边缘
10	慢速后退寻找 EZ 边缘
11	慢速前向寻找 LIMIT 边缘
12	慢速后退寻找 LIMIT 边缘
13	定位到原点/限位锁存位置
14	定位到 EZ 锁存位置
15	定位到 POT/NOT 锁存位置
17	命令发送过来的减速停止
18	异常情况（限位）减速停止
19	处理 offset 的相关操作
20	完成回零过程，但不代表是回零成功

ERROR_CODE NV_HomeError(UInt16 ConnectNo,UInt16 AxisNo,ref UInt32 pErrCode,byte[] pErrString)

功　能：获取回零失败的原因

参　数：ConnectNo　　板卡连接编号[0,15]

AxisNo 轴序号

pErrCode 返回回零失败错误码

pErrString 返回错误提示字符串

返回值：错误代码

ERROR_CODE NV_HomeMove(UInt16 ConnectNo,UInt16 AxisNo)

功 能：启动回零运动

参 数：ConnectNo 板卡连接编号[0,15]

AxisNo 轴序号

返回值：错误代码

5.6.2 点位运动

点位速度参数结构体定义

```
typedef struct ProfileStruct_t
{
    UInt16 IsValid;   // 参数是否有效
    Double StartSpeed;    // 起始速度
    Double MaxSpeed; // 最大速度
    Double StopSpeed;// 停止速度
    Double Acceleraton;   // 加速度
    Double Deceleraton;   // 减速度
    Double SmoothTime;    // 平滑时间，单位 s，范围 0-1
}ProfileStruct_t, *PProfileStruct_t;
```

运动参数设置

ERROR_CODE NV_PmoveSetProfile(UInt16 ConnectNo,UInt16 AxisNo,ProfileStruct_t Profile)

功　能：设置定长运动的参数

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

Profile　规划参数

返回值：错误代码

ERROR_CODE NV_PmoveGetProfile(UInt16 ConnectNo,UInt16 AxisNo,ref PProfileStruct_t pProfile)

功　能：获取定长运动的参数

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

pProfile　返回规划参数

返回值：错误代码

ERROR_CODE NV_PmoveAbsolute(UInt16 ConnectNo, UInt16 AxisNo, Double Position, ProfileStruct_t Profile)

功　能：设置绝对位置定长参数并启动运动

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

Position　要到达的位置，单位 units

Profile　规划参数

返回值：错误代码

说明：可以在进行点位运动的状态下，调用此函数，可以进行强制变位功能；

ERROR_CODE NV_PmoveRelative(UInt16 ConnectNo, UInt16 AxisNo, Double Distance, ProfileStruct_t Profile)

功　能：设置相对位置定长参数并启动运动

参　数：ConnectNo　　板卡连接编号[0,15]

AxisNo　　轴序号

Position　要运动的距离

Profile　　规划参数

返回值：错误代码

两段运动参数结构体定义

```
typedef struct Profile2Struct_t
{
    Double Position1;    // 第一段运动的相对或者绝对位置（根据 Mode 决定）
    Double Position2;    // 第二段运动的相对或者绝对位置（根据 Mode 决定）
    Double StartSpeed;   // 起始速度
    Double MaxSpeed1;    // 第一段最大速度
    Double LinkSpeed;    // 第二段最大速度
    Double MaxSpeed2;    // 停止速度
    Double StopSpeed;    // 加速度
    Double Acceleraton1;// 第一段运行加速度
    Double Deceleraton1;// 第一段运行减速度
    Double Acceleraton2;// 第二段运行加速度
    Double Deceleraton2;// 第二段运行减速度
    UInt16 PosMode; // 位置模式：0 - 绝对坐标模式；1 - 相对坐标模式
Double SmoothTime;   // 平滑时间
}Profile2Struct_t, *PProfile2Struct_t;
```

ERROR_CODE NV_PmoveTwice(UInt16 ConnectNo,UInt16 AxisNo,Profile2Struct_t Profile)

功　能：设置相对位置定长参数并启动运动（运动类型为两段运动）。

参　数：ConnectNo　　板卡连接编号[0,15]

AxisNo　　轴序号

Profile　　规划参数

返回值：错误代码

说明：

（1）如果是相对位置模式，Position1 和 Position2 都是各自运动段的距离，他们需要同向；（2）如果是绝对位置模式，那么 Position1 必须处在当前位置和 Position2 区间的位置

在线修改运动

ERROR_CODE NV_PmoveChangePos(UInt16 ConnectNo,UInt16 AxisNo,Double Position,UInt16 Mode)

功　能：修改点位运动的目标位置或者在停止的时候进行新的一个运动

参　数：ConnectNo 板卡连接编号[0,15]

AxisNo　　轴序号

Position 根据 mode 决定要追加的运动距离或者位置

Mode　　改变的位置的模式：

0：绝对变位，变更运动到新的绝对位置，运动停止后也可执行。

1：相对变位，相对第一次开始运动的起始置位的距离。

运动停止后调用报错处理。

返回值：错误代码

运动状态

ERROR_CODE NV_PmoveGetPlanInfo(UInt16 ConnectNo,UInt16 AxisNo,ref Double pTotalTime, ref Double pRemainTime)

功　能：返回点位规划信息

参　数：ConnectNo　　板卡连接编号[0,15]

AxisNo 轴序号

pTotalTime 读取运动的总时间，单位 s

pRemainTime 读取运动的剩余时间，单位 s

返回值：错误代码

5.6.3 定速运动

ERROR_CODE NV_VmoveSetProfile(UInt16 ConnectNo,UInt16 AxisNo,ProfileStruct_t Profile)

功 能：设置定速运动的参数

参 数：ConnectNo 板卡连接编号[0,15]

AxisNo 轴序号

Profile 规划参数

返回值：错误代码

ERROR_CODE NV_VmoveGetProfile(UInt16 ConnectNo,UInt16 AxisNo,ref PProfileStruct_t pProfile)

功 能：获取定速运动的参数

参 数：ConnectNo 板卡连接编号[0,15]

AxisNo 轴序号

pProfile 返回规划参数

返回值：错误代码

ERROR_CODE NV_VmoveMove(UInt16 ConnectNo,UInt16 AxisNo,Int16 Direction)

功 能：软件启动定速运动

参 数：ConnectNo 板卡连接编号[0,15]

AxisNo 轴序号

Direction 定速运动的方向：

-1：负向，

0 ：暂停

1 ：正向

返回值：错误代码

说明：可以在运动中修改方向。

5.6.4 Jog 运动

Jog 运动参数结构体

```
typedef struct JogProfileStruct
{
    Double StartSpeed;   // 起始速度
    Double MaxSpeed;// 最大速度
    Double StopSpeed;    // 停止速度
    Double Acceleraton; // 加速度
    Double Deceleraton; // 减速度
    Double SmoothTime;   // 平滑时间，单位 s，范围 0-1
}JogProfileStruct_t, * PJogProfileStruct_t;
```

ERROR_CODE NV_JogSetProfile(UInt16 ConnectNo,UInt16 AxisNo,JogProfileStruct_t Profile)

功　能：设置 Jog 运动的参数

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

Profile　规划参数

返回值：错误代码

ERROR_CODE NV_JogGetProfile(UInt16 ConnectNo,UInt16 AxisNo,ref JogProfileStruct_t pProfile)

功　能：获取 Jog 运动的参数

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

pProfile　返回规划参数

返回值：错误代码

ERROR_CODE NV_JogSetDiConfig(UInt16 ConnectNo,UInt16 AxisNo,UInt16 ForwardPin, UInt16 ForwardLevel,UInt16 ReversePin,UInt16 ReverseLevel)

功　能：采用硬件触发方式，启动 Jog 运动

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

ForwardPin　正向时对应的通用输入口编号

ForwardLevel　正向时对应的通用输入口有效电平

ReversePin　反向时对应的通用输入口编号

ReverseLevel　反向时对应的通用输入口有效电平

返回值：错误代码

ERROR_CODE NV_JogGetDiConfig(UInt16 ConnectNo,UInt16 AxisNo,ref UInt16 pForwardPin, ref UInt16 pForwardLevel,ref UInt16 pReversePin,ref UInt16 pReverseLevel)

功　能：获取硬件触发方式，Jog 运动参数配置

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

ForwardPin　返回正向时对应的通用输入口编号

ForwardLevel　返回正向时对应的通用输入口有效电平

ReversePin　返回反向时对应的通用输入口编号

ReverseLevel 返回反向时对应的通用输入口有效电平

返回值：错误代码

说明：配置的输入端口电平状态为有效电平，则执行 Jog 运动，为无效电平停止 Jog 运动。

ERROR_CODE NV_JogMove(UInt16 ConnectNo,UInt16 AxisNo, UInt16 Mode, UInt16 Forward, UInt16 Reverse)

功 能：采用软件触发方式，启动 Jog 运动

参 数：ConnectNo 板卡连接编号[0,15]

AxisNo 轴序号

Mode 模式：0-硬件输入控制，1-软件控制

Forward 正转：0-停止，1-正向

Reverse 反转：0-停止，1-反向

返回值：错误代码

说明：Forward 与 Reverse 值相等的时候不转

5.6.5 变速运动

ERROR_CODE NV_AxisChangeSpeed(UInt16 ConnectNo,UInt16 AxisNo,Double NewSpeed)

功 能：修改点位运动的速度

参 数：ConnectNo 板卡连接编号[0,15]

AxisNo 轴序号

NewSpeed 要设置的新速度

返回值：错误代码

ERROR_CODE NV_AxisChangeSpeedByRate(UInt16 ConnectNo,UInt16 AxisNo,Double NewRate)

功 能：修改点位运动的速度的比例

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

NewRate　比列系数，单位：百分比

范围：0 – 200

返回值：错误代码

ERROR_CODE NV_AxisChangeSpeedAcc(UInt16 ConnectNo,UInt16 AxisNo,Double NewSpeed, Double NewAcc, Double NewDec)

功　能：修改点位运动的速度、加速度、减速度

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

NewSpeed　要修改的最大速度

NewAcc　要修改的加速度

NewDec　要修改的减速度

返回值：错误代码

ERROR_CODE NV_AxisChangeSpeedAccByRate(UInt16 ConnectNo,UInt16 AxisNo,Double NewRate)

功　能：修改点位运动的速度、加速度比例

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

NewRate　要修改的最大速度、加速度、减速度，单位：百分比

范围 0 – 200

返回值：错误代码

注意：可以多次修改，但是基数是启动运动时的速度、加速度和减速度。

5.6.6 CSV 模式

ERROR_CODE NV_AxisSetTargetVelocity(UInt16 ConnectNo, UInt16 AxisNo, Int32

Value)

功　能：设置控制目标速度，需要切换到对应的模式

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

Value　控制的速度值

返回值：错误代码

ERROR_CODE NV_AxisGetTargetVelocity (UInt16 ConnectNo, UInt16 AxisNo, Int32[] pValue, UInt16 Count)

功　能：获取设置的目标速度

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

pValue　获取当前的速度值

返回值：错误代码

ERROR_CODE NV_AxisGetActualVelocity (UInt16 ConnectNo, UInt16 AxisNo, Int32[] pValue, UInt16 Count)

功　能：获取实时速度

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

pValue　获取当前的速度值

返回值：错误代码

5.6.7 CST 模式

ERROR_CODE NV_AxisSetTargetTorque(UInt16 ConnectNo, UInt16 AxisNo, Int16 Value)

功　能：设置控制目标转矩，需要切换到对应的模式

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

Value　控制的转矩值

返回值：错误代码

ERROR_CODE NV_AxisGetTargetTorque (UInt16 ConnectNo, UInt16 AxisNo,Int16[] pValue, UInt16 Count)

功　能：获取设置的目标转矩

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

pValue　获取当前的转矩值

Count　返回多个连续轴的转矩值

返回值：错误代码

ERROR_CODE NV_AxisGetActualTorque (UInt16 ConnectNo, UInt16 AxisNo, Int16[] pValue, UInt16 Count)

功　能：获取转矩

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

pValue　获取当前的转矩值

Count　返回多个连续轴的转矩值

返回值：错误代码

ERROR_CODE NV_AxisSetMaxTorque(UInt16 ConnectNo, UInt16 AxisNo, Int16 Value)

功　能：设置最大转矩

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　　轴序号

Value　　控制的最大转矩值

返回值：错误代码

ERROR_CODE NV_AxisGetMaxTorque (UInt16 ConnectNo, UInt16 AxisNo, Int16[] pValue, UInt16 Count)

功　能：获取最大转矩

参　数：ConnectNo　　板卡连接编号[0,15]

AxisNo　　轴序号

pValue　　获取最大的转矩值

Count　　返回多个连续轴的转矩值

返回值：错误代码

5.7 插补运动

5.7.1 插补配置

ERROR_CODE NV_CrdSetLookAhead(UInt16 ConnectNo,UInt16 CrdNo,UInt16 Enable)

功　能：设置小线段前瞻功能

参　数：ConnectNo　　板卡连接编号[0,15]

CrdNo　　插补坐标系序号，范围 0-7

Enable　　是否开启小线段前瞻功能：

0：不开启；

1：开启

返回值：错误代码

ERROR_CODE NV_CrdGetLookAhead(UInt16 ConnectNo,UInt16 CrdNo,ref UInt16

pEnable)

功　能：获取小线段前瞻功能状态

参　数：ConnectNo　板卡连接编号[0,15]

CrdNo　插补坐标系序号，范围 0-7

pEnable　返回是否开启小线段前瞻功能

返回值：错误代码

ERROR_CODE NV_CrdSetPrm(UInt16 ConnectNo,UInt16 CrdNo,UInt16 AxisNum,UInt16[] pAxisList,Double[] pOffset)

功　能：设置插补坐标系参数

参　数：ConnectNo　板卡连接编号[0,15]

CrdNo　插补坐标系序号，范围 0-7

AxisNum　坐标系的轴数量，范围 1-8

pAxisList　坐标系的轴列表

pOffset　坐标系相对机械原点的偏移量

返回值：错误代码

ERROR_CODE NV_CrdGetPrm(UInt16 ConnectNo,UInt16 CrdNo,ref UInt16 pAxisNum,UInt16[] pAxisList,Double[] pOffset)

功　能：获取插补坐标系参数

数：ConnectNo　板卡连接编号[0,15]

CrdNo　插补坐标系序号，范围 0-7

pAxisNum　返回坐标系的轴数量，范围 1-8

pAxisList　返回坐标系的轴列表

pOffset　返回坐标系相对机械原点的偏移量

返回值：错误代码

ERROR_CODE NV_CrdSetPrfMax(UInt16 ConnectNo,UInt16 CrdNo,Double MaxVel,Double MaxAcc,Double MaxDec)

功　能：设置插补坐标系运动极限参数

参　数：ConnectNo　板卡连接编号[0,15]

CrdNo　插补坐标系序号，范围 0-7

MaxVel　坐标系的允许最大速度，范围：大于 0

MaxAcc　坐标系的允许最大加速度，范围：大于 0

MaxDec　坐标系的允许最大减速度，范围：大于 0

返回值：错误代码

ERROR_CODE NV_CrdGetPrfMax(UInt16 ConnectNo,UInt16 CrdNo,ref Double pMaxVel,ref Double pMaxAcc,ref Double pMaxDec)

功　能：获取插补坐标系运动极限参数

参　数：ConnectNo　板卡连接编号[0,15]

CrdNo　插补坐标系序号，范围 0-7

pMaxVel　返回坐标系的允许最大速度

pMaxAcc　返回坐标系的允许最大加速度

pMaxDec　返回坐标系的允许最大减速度

返回值：错误代码

ERROR_CODE NV_CrdSetProfile(UInt16 ConnectNo, UInt16 CrdNo, ProfileStruct_t Profile)

功　能：设置插补运动规划的参数

参　数：ConnectNo　板卡连接编号[0,15]

CrdNo　插补坐标系序号

Profile　规划参数

返回值：错误代码

ERROR_CODE NV_CrdGetProfile(UInt16 ConnectNo, UInt16 CrdNo, ref ProfileStruct_t pProfile)

功　能：获取插补运动规划的参数

参　数：ConnectNo　板卡连接编号[0,15]

CrdNo　插补坐标系序号

pProfile　返回规划参数

返回值：错误代码

ERROR_CODE NV_CrdSetLinkMode(UInt16 ConnectNo,UInt16 CrdNo,UInt16 LinkMode,Double PathError)

功　能：设置插补坐标系拐角参数

参　数：ConnectNo　板卡连接编号[0,15]

CrdNo　插补坐标系序号，范围 0-7

LinkMode 坐标系的拐角过渡连接方式

0：直接过渡，

1：圆弧过渡

PathError　坐标系的轨迹误差，单位 unit

返回值：错误代码

ERROR_CODE NV_CrdGetLinkMode(UInt16 ConnectNo,UInt16 CrdNo,ref UInt16 pLinkMode,ref Double pPathError)

功　能：获取插补坐标系拐角参数

参　数：ConnectNo　板卡连接编号[0,15]

CrdNo　插补坐标系序号，范围 0-7

pLinkMode　返回坐标系的拐角过渡连接方式

pPathError　返回坐标系的轨迹误差

返回值：错误代码

ERROR_CODE NV_CrdSetOverride(UInt16 ConnectNo,UInt16 CrdNo,Double VelRatio)

功　能：设置插补速度倍率

参　数：ConnectNo　板卡连接编号[0,15]

CrdNo　插补坐标系序号，范围 0-7

VelRatio　设置的倍率；速度倍率为 0 则为暂停

返回值：错误代码

ERROR_CODE NV_CrdGetOverride(UInt16 ConnectNo,UInt16 CrdNo,ref Double pVelRatio)

功　能：获取插补速度倍率

参　数：ConnectNo　板卡连接编号[0,15]

CrdNo　插补坐标系序号，范围 0-7

pVelRatio 获取已设置的倍率

返回值：错误代码

ERROR_CODE NV_CrdStatus(UInt16 ConnectNo,UInt16 CrdNo,ref UInt16 pRun,ref UInt32 pSegment, ref UInt16 pRemain,ref UInt16 pSegmentType，ref Double pCurVel)

功　能：获取插补坐标系运动状态

参　数：ConnectNo　板卡连接编号[0,15]

CrdNo　插补坐标系序号，范围 0-7

pRun　返回坐标系的运行状态

pSegment　返回坐标系的当前运行段号

pRemain　返回坐标系的剩余插补空间

pSegmentType 返回坐标系插补段类型

pCurVel 返回坐标系的当前插补速度

返回值：错误代码

坐标系的运行状态值如下：

状态值	状态	描述
0	LIST_IDLE	未使用，空闲状态
1	LIST_ACTIVE	打开 list,但没有 start
2	LIST_RUNNING	启动 list,正常运行状态
3	LIST_PAUSING	暂停中，正在减速到停止状态
4	LIST_PAUSED	已暂停，运动速度为 0
5	LIST_STOPPING	减速停止中
6	LIST_STOPPED	停止状态，下一个周期马上切换到 idle 状态

ERROR_CODE NV_CrdPosition(UInt16 ConnectNo,UInt16 CrdNo, Double[] pPos, ref UInt16 pCount)

功　能：获取插补坐标系各轴位置

参　数：ConnectNo 板卡连接编号[0,15]

CrdNo　插补坐标系序号，范围 0-7

pPos　返回当前插补系坐标位置 Xpos,Ypos,Zpos...

pCount　返回位置的数量

返回值：错误代码

ERROR_CODE NV_CrdGetStopReason(UInt16 ConnectNo,UInt16 CrdNo,UInt32[] pReason,UInt16 Count)

功　能：获取插补运动停止的原因

参　数：ConnectNo　板卡连接编号[0,15]

CrdNo　插补坐标系序号，范围 0-7

pReason　返回的停止原因

Count　　返回停止原因的数量

返回值：错误代码

ERROR_CODE NV_CrdClrStopReason(UInt16 ConnectNo,UInt16 CrdNo)

功　能：清除插补运动停止原因的记录

参　数：ConnectNo　　板卡连接编号[0,15]

CrdNo　　插补坐标系序号，范围 0-7

返回值：错误代码

ERROR_CODE NV_CrdOpen(UInt16 ConnectNo,UInt16 CrdNo)

功　能：打开连续插补缓冲区

参　数：ConnectNo　　板卡连接编号[0,15]

CrdNo　　插补坐标系序号，范围 0-7

返回值：错误代码

ERROR_CODE NV_CrdClose(UInt16 ConnectNo,UInt16 CrdNo)

功　能：关闭连续插补缓冲区

参　数：ConnectNo　　板卡连接编号[0,15]

CrdNo　　插补坐标系序号，范围 0-7

返回值：错误代码

ERROR_CODE NV_CrdStart(UInt16 ConnectNo,UInt16 CrdNo,UInt16 RunMode)

功　能：插补启动

参　数：ConnectNo　　板卡连接编号[0,15]

CrdNo　　插补坐标系序号，范围 0-7

RunMode　　运行模式

0：连续执行

1：单步执行

返回值：错误代码

ERROR_CODE NV_CrdPause(UInt16 ConnectNo,UInt16 CrdNo)

功　能：插补暂停

参　数：ConnectNo　板卡连接编号[0,15]

CrdNo　插补坐标系序号，范围 0-7

返回值：错误代码

ERROR_CODE NV_CrdStop(UInt16 ConnectNo,UInt16 CrdNo,UInt16 StopMode)

功　能：插补停止

参　数：ConnectNo　板卡连接编号[0,15]

CrdNo　插补坐标系序号，范围 0-7

StopMode 停止模式：

0：减速停止

1：立即停止

返回值：错误代码

5.7.2 插补运动

直线插补

ERROR_CODE NV_CrdLnXY(UInt16 ConnectNo,UInt16 CrdNo,Double EndX,Double EndY, UInt16 PosMode,UInt32 SegNum,Double MaxVel,Double LinkVel)

功　能：添加 XY 两轴直线插补

参　数：ConnectNo　板卡连接编号[0,15]

CrdNo　插补坐标系序号，范围 0-7

EndX　X 的段末位置

EndY　　Y 的段末位置

PosMode 运动模式：0 - 绝对坐标模式；1 - 相对坐标模式

SegNum　该段插补的段号；0 默认会自动递增

MaxVel　　该段的最大速度；-1 表示同上上段速度

LinkVel　　过渡速度；-1 表示按照默认设置过渡

返回值：错误代码

ERROR_CODE　NV_CrdLnXYZ(UInt16　ConnectNo,UInt16　CrdNo,Double EndX,Double EndY,Double EndZ,UInt16 PosMode,UInt32 SegNum,Double MaxVel,Double LinkVel)

功　能：添加 XYZ 三轴直线插补

参　数：ConnectNo　板卡连接编号[0,15]

CrdNo　插补坐标系序号，范围 0-7

EndX　X 的段末位置

EndY　Y 的段末位置

EndZ　Y 的段末位置

PosMode 运动模式：0 - 绝对坐标模式；1 - 相对坐标模式

SegNum　该段插补的段号；0 默认会自动递增

MaxVel　　该段的最大速度；-1 表示同上上段速度

LinkVel　　过渡速度；-1 表示按照默认设置过渡

返回值：错误代码

ERROR_CODE　NV_CrdLnXYZA(UInt16　ConnectNo,UInt16　CrdNo,Double EndX,Double EndY,Double EndZ,Double EndA, UInt16 PosMode,UInt32 SegNum,Double MaxVel,Double LinkVel)

功　能：添加 XYZA 四轴直线插补

参　数：ConnectNo　板卡连接编号[0,15]

CrdNo 插补坐标系序号，范围 0-7

EndX X 的段末位置

EndY Y 的段末位置

EndZ Z 的段末位置

EndA A 的段末位置

PosMode 运动模式：0 - 绝对坐标模式；1 - 相对坐标模式

SegNum 该段插补的段号；0 默认会自动递增

MaxVel 该段的最大速度；-1 表示同上上段速度

LinkVel 过渡速度；-1 表示按照默认设置过渡

返回值：错误代码

ERROR_CODE NV_CrdLnMove(UInt16 ConnectNo,UInt16 CrdNo,UInt16 AxisNum,UInt16[] pAxisList, Double[] pEndList, UInt16 PosMode,UInt32 SegNum,Double MaxVel,Double LinkVel)

功 能：添加任意轴直线插补

参 数：ConnectNo 板卡连接编号[0,15]

CrdNo 插补坐标系序号，范围 0-7

AxisNum 轴数量，范围 1-8

pAxisList 轴列表

pEndList 轴段末位置列表

PosMode 运动模式：0 - 绝对坐标模式；1 - 相对坐标模式

SegNum 该段插补的段号；0 默认会自动递增

MaxVel 该段的最大速度；-1 表示同上上段速度

LinkVel 过渡速度；-1 表示按照默认设置过渡

返回值：错误代码

注：这里的轴列表序号对应的是 0-X,1-Y,2-Z,3-A,4-B,5-C,6-U,7-V

圆弧插补

ERROR_CODE NV_CrdArcXYR(UInt16 ConnectNo,UInt16 CrdNo,Double EndX,Double EndY,Double Radius,UInt16 ArcDir, Int32 Cycles,UInt16 PosMode,UInt32 SegNum,Double MaxVel,Double LinkVel)

功　能：以半径模式添加 XY 平面的圆弧

参　数：ConnectNo　板卡连接编号[0,15]

CrdNo　插补坐标系序号，范围 0-7

EndX　X 的段末位置

EndY　Y 的段末位置

Radius　圆弧的半径

正数：圆弧的旋转角度≤180°

负数：圆弧的旋转角度＞180°

ArcDir　圆弧的方向：0 - 顺时针方向；1 - 逆时针方向

Cycles整圆的圈数，走完圆弧轨迹段后继续绕的整圆圈数。

PosMode 运动模式：0 - 绝对坐标模式；1 - 相对坐标模式

SegNum 该段插补的段号；0 默认会自动递增

MaxVel　该段的最大速度；-1 表示同上上段速度

LinkVel　过渡速度；-1 表示按照默认设置过渡

返回值：错误代码

ERROR_CODE NV_CrdArcXYC(UInt16 ConnectNo,UInt16 CrdNo,Double EndX,Double EndY,Double CenterX,Double CenterY,UInt16 ArcDir, Int32 Cycles,UInt16 PosMode,UInt32 SegNum,Double MaxVel,Double LinkVel)

功　能：以圆心模式添加 XY 平面的圆弧

参　数：ConnectNo　板卡连接编号[0,15]

CrdNo　插补坐标系序号，范围 0-7

EndX　X 的段末位置

EndY　Y 的段末位置

CenterX X 的圆心位置

CenterY Y 的圆心位置

ArcDir 圆弧的方向：0 - 顺时针方向；1 - 逆时针方向

Cycles 整圆的圈数，走完圆弧轨迹段后继续绕的整圆圈数。

PosMode 运动模式：0 - 绝对坐标模式；1 - 相对坐标模式

SegNum 该段插补的段号；0 默认会自动递增

MaxVel 该段的最大速度；-1 表示同上上段速度

LinkVel 过渡速度；-1 表示按照默认设置过渡

返回值：错误代码

ERROR_CODE NV_CrdArcYZR(UInt16 ConnectNo,UInt16 CrdNo,Double EndY,Double EndZ,Double Radius,UInt16 ArcDir, Int32 Cycles,UInt16 PosMode,UInt32 SegNum,Double MaxVel,Double LinkVel)

功 能：以半径模式添加 YZ 平面的圆弧

参 数：ConnectNo 板卡连接编号[0,15]

CrdNo 插补坐标系序号，范围 0-7

EndY Y 的段末位置

EndZ Z 的段末位置

Radius 圆弧的半径

正数：圆弧的旋转角度≤180°

负数：圆弧的旋转角度＞180°

ArcDir 圆弧的方向：0 - 顺时针方向；1 - 逆时针方向

Cycles整圆的圈数，走完圆弧轨迹段后继续绕的整圆圈数。

PosMode 运动模式：0 - 绝对坐标模式；1 - 相对坐标模式

SegNum 该段插补的段号；0 默认会自动递增

MaxVel 该段的最大速度；-1 表示同上上段速度

LinkVel 过渡速度；-1 表示按照默认设置过渡

返回值：错误代码

ERROR_CODE NV_CrdArcYZC(UInt16 ConnectNo,UInt16 CrdNo,Double EndY,Double EndZ,Double CenterY,Double CenterZ,UInt16 ArcDir, Int32 Cycles,UInt16 PosMode,UInt32 SegNum,Double MaxVel,Double LinkVel)

功　能：以圆心模式添加 YZ 平面的圆弧

参　数：ConnectNo　板卡连接编号[0,15]

CrdNo　插补坐标系序号，范围 0-7

EndY　Y 的段末位置

EndZ　Z 的段末位置

CenterY　Y 的圆心位置

CenterZ　Z 的圆心位置

ArcDir　圆弧的方向：0 - 顺时针方向；1 - 逆时针方向

Cycles　整圆的圈数，走完圆弧轨迹段后继续绕的整圆圈数。

PosMode　运动模式：0 - 绝对坐标模式；1 - 相对坐标模式

SegNum　该段插补的段号；0 默认会自动递增

MaxVel　该段的最大速度；-1 表示同上上段速度

LinkVel　过渡速度；-1 表示按照默认设置过渡

返回值：错误代码

ERROR_CODE NV_CrdArcZXR(UInt16 ConnectNo,UInt16 CrdNo,Double EndZ,Double EndX,Double Radius,UInt16 ArcDir, Int32 Cycles,UInt16 PosMode,UInt32 SegNum,Double MaxVel,Double LinkVel)

功　能：以半径模式添加 ZX 平面的圆弧

参　数：ConnectNo　板卡连接编号[0,15]

CrdNo　插补坐标系序号，范围 0-7

EndZ　Z 的段末位置

EndX　　X 的段末位置

Radius　　圆弧的半径

正数：圆弧的旋转角度≤180°

负数：圆弧的旋转角度＞180°

ArcDir　　圆弧的方向：0 - 顺时针方向；1 - 逆时针方向

Cycles整圆的圈数，走完圆弧轨迹段后继续绕的整圆圈数。

SegNum　该段插补的段号；0 默认会自动递增

MaxVel　　该段的最大速度；-1 表示同上上段速度

LinkVel　　过渡速度；-1 表示按照默认设置过渡

返回值：错误代码

ERROR_CODE　NV_CrdArcZXC(UInt16　ConnectNo,UInt16　CrdNo,Double EndZ,Double EndX,Double CenterZ,Double CenterX,UInt16 ArcDir, Int32 Cycles,UInt16 PosMode,UInt32 SegNum,Double MaxVel,Double LinkVel)

功　能：以圆心模式添加 ZX 平面的圆弧

参　数：ConnectNo　板卡连接编号[0,15]

CrdNo　插补坐标系序号，范围 0-7

EndZ　Z 的段末位置

EndX　X 的段末位置

CenterZ　　Z 的圆心位置

CenterX　　X 的圆心位置

ArcDir　　圆弧的方向：0 - 顺时针方向；1 - 逆时针方向

Cycles整圆的圈数，走完圆弧轨迹段后继续绕的整圆圈数。

PosMode 运动模式：0 - 绝对坐标模式；1 - 相对坐标模式

SegNum　该段插补的段号；0 默认会自动递增

MaxVel　　该段的最大速度；-1 表示同上上段速度

LinkVel　　过渡速度；-1 表示按照默认设置过渡

返回值：错误代码

ERROR_CODE NV_CrdArcXYZ(UInt16 ConnectNo,UInt16 CrdNo,Double EndX,Double EndY,Double EndZ,Double InterX,Double InterY,Double InterZ, Int32 Cycles,UInt16 PosMode,UInt32 SegNum,Double MaxVel,Double LinkVel)

功　能：以任意三点模式添加空间圆弧（XY 平面、YZ 平面、XZ 平面的三点圆弧，也采用该函数）

参　数：ConnectNo　板卡连接编号[0,15]

CrdNo　插补坐标系序号，范围 0-7

EndX　X 的段末位置

EndY　Y 的段末位置

EndZ　Z 的段末位置

InterX　X 的过渡点位置

InterY　Y 的过渡点位置

InterZ　Z 的过渡点位置

Cycles整圆的圈数，走完圆弧轨迹段后继续绕的整圆圈数。

PosMode 运动模式：0 - 绝对坐标模式；1 - 相对坐标模式

SegNum　该段插补的段号；0 默认会自动递增

MaxVel　该段的最大速度；-1 表示同上上段速度

LinkVel　过渡速度；-1 表示按照默认设置过渡

返回值：错误代码

ERROR_CODE NV_CrdHelixXYRZ(UInt16 ConnectNo,UInt16 CrdNo,Double EndX,Double EndY,Double EndZ,Double Radius,UInt16 ArcDir, Int32 Cycles,UInt16 PosMode,UInt32 SegNum,Double MaxVel,Double LinkVel)

功　能：以半径模式添加 XY 平面的圆弧并在 Z 轴移动的圆柱型运动

参　数：ConnectNo　板卡连接编号[0,15]

CrdNo 插补坐标系序号，范围 0-7

EndX X 的段末位置

EndY Y 的段末位置

EndZ Z 的段末位置

Radius 圆弧的半径

正数：圆弧的旋转角度≤180°

负数：圆弧的旋转角度＞180°

ArcDir 圆弧的方向：0 - 顺时针方向；1 - 逆时针方向

Cycles整圆的圈数，走完圆弧轨迹段后继续绕的整圆圈数。

PosMode 运动模式：0 - 绝对坐标模式；1 - 相对坐标模式

SegNum 该段插补的段号；0 默认会自动递增

MaxVel 该段的最大速度；-1 表示同上上段速度

LinkVel 过渡速度；-1 表示按照默认设置过渡

返回值：错误代码

ERROR_CODE NV_CrdHelixXYCZ(UInt16 ConnectNo,UInt16 CrdNo,Double EndX,Double EndY,Double EndZ,Double CenterX,Double CenterY,UInt16 ArcDir, Int32 Cycles,UInt16 PosMode,UInt32 SegNum,Double MaxVel,Double LinkVel)

功 能：以圆心模式添加 XY 平面的圆弧并在 Z 轴移动的圆柱型运动

参 数：ConnectNo 板卡连接编号[0,15]

CrdNo 插补坐标系序号，范围 0-7

EndX X 的段末位置

EndY Y 的段末位置

EndZ Z 的段末位置

CenterX X 的圆心位置

CenterY Y 的圆心位置

ArcDir 圆弧的方向：0 - 顺时针方向；1 - 逆时针方向

Cycles整圆的圈数，走完圆弧轨迹段后继续绕的整圆圈数。

PosMode 运动模式：0 - 绝对坐标模式；1 - 相对坐标模式

SegNum 该段插补的段号；0 默认会自动递增

MaxVel 该段的最大速度；-1 表示同上上段速度

LinkVel 过渡速度；-1 表示按照默认设置过渡

返回值：错误代码

ERROR_CODE NV_CrdHelixYZRX(UInt16 ConnectNo,UInt16 CrdNo,Double EndX,Double EndY,Double EndZ,Double Radius,UInt16 ArcDir, Int32 Cycles,UInt16 PosMode,UInt32 SegNum,Double MaxVel,Double LinkVel)

功 能：以半径模式添加 YZ 平面的圆弧并在 X 轴移动的圆柱型运动

参 数：ConnectNo 板卡连接编号[0,15]

CrdNo 插补坐标系序号，范围 0-7

EndX X 的段末位置

EndY Y 的段末位置

EndZ Z 的段末位置

Radius 圆弧的半径

正数：圆弧的旋转角度≤180°

负数：圆弧的旋转角度＞180°

ArcDir 圆弧的方向：0 - 顺时针方向；1 - 逆时针方向

Cycles整圆的圈数，走完圆弧轨迹段后继续绕的整圆圈数。

PosMode 运动模式：0 - 绝对坐标模式；1 - 相对坐标模式

SegNum 该段插补的段号；0 默认会自动递增

MaxVel 该段的最大速度；-1 表示同上上段速度

LinkVel 过渡速度；-1 表示按照默认设置过渡

返回值：错误代码

ERROR_CODE NV_CrdHelixYZCX(UInt16 ConnectNo,UInt16 CrdNo,Double EndX,Double EndY,Double EndZ,Double CenterY,Double CenterZ,UInt16 ArcDir, Int32 Cycles,UInt16 PosMode,UInt32 SegNum,Double MaxVel,Double LinkVel)

功　能：以圆心模式添加 YZ 平面的圆弧并在 X 轴移动的圆柱型运动

参　数：ConnectNo　板卡连接编号[0,15]

CrdNo　插补坐标系序号，范围 0-7

EndX　X 的段末位置

EndY　Y 的段末位置

EndZ　Z 的段末位置

CenterY　Y 的圆心位置

CenterZ　Z 的圆心位置

ArcDir　圆弧的方向：0 - 顺时针方向；1 - 逆时针方向

Cycles整圆的圈数，走完圆弧轨迹段后继续绕的整圆圈数。

PosMode 运动模式：0 - 绝对坐标模式；1 - 相对坐标模式

SegNum 该段插补的段号；0 默认会自动递增

MaxVel　该段的最大速度；-1 表示同上上段速度

LinkVel　过渡速度；-1 表示按照默认设置过渡

返回值：错误代码

ERROR_CODE NV_CrdHelixZXRY(UInt16 ConnectNo,UInt16 CrdNo,Double EndX,Double EndY,Double EndZ,Double Radius,UInt16 ArcDir, Int32 Cycles,UInt16 PosMode,UInt32 SegNum,Double MaxVel,Double LinkVel)

功　能：以半径模式添加 ZX 平面的圆弧并在 Y 轴移动的圆柱型运动

参　数：ConnectNo　板卡连接编号[0,15]

CrdNo　插补坐标系序号，范围 0-7

EndX　X 的段末位置

EndY　Y 的段末位置

EndZ　　Z 的段末位置

Radius　　圆弧的半径

正数：圆弧的旋转角度≤180°

负数：圆弧的旋转角度＞180°

ArcDir　　圆弧的方向：0 - 顺时针方向；1 - 逆时针方向

Cycles整圆的圈数，走完圆弧轨迹段后继续绕的整圆圈数。

PosMode 运动模式：0 - 绝对坐标模式；1 - 相对坐标模式

SegNum　该段插补的段号；0 默认会自动递增

MaxVel　该段的最大速度；-1 表示同上上段速度

LinkVel　过渡速度；-1 表示按照默认设置过渡

返回值：错误代码

ERROR_CODE NV_CrdHelixZXCY(UInt16 ConnectNo,UInt16 CrdNo,Double EndX,Double EndY,Double EndZ,Double CenterZ,Double CenterX,UInt16 ArcDir, Int32 Cycles,UInt16 PosMode,UInt32 SegNum,Double MaxVel,Double LinkVel)

功　能：以圆心模式添加 ZX 平面的圆弧并在 Y 轴移动的圆柱型运动

参　数：ConnectNo　　板卡连接编号[0,15］

CrdNo　　插补坐标系序号，范围 0-7

EndX　　X 的段末位置

EndY　　Y 的段末位置

EndZ　　Z 的段末位置

CenterZ　Z 的圆心位置

CenterX　X 的圆心位置

ArcDir　　圆弧的方向：0 - 顺时针方向；1 - 逆时针方向

Cycles整圆的圈数，走完圆弧轨迹段后继续绕的整圆圈数。

PosMode 运动模式：0 - 绝对坐标模式；1 - 相对坐标模式

SegNum　该段插补的段号；0 默认会自动递增

MaxVel 该段的最大速度；-1 表示同上上段速度

LinkVel 过渡速度；-1 表示按照默认设置过渡

返回值：错误代码

ERROR_CODE NV_CrdArcMoveC(UInt16 ConnectNo, UInt16 CrdNo, UInt16 AxisNum, UInt16[] pAxisList, Double[] pEndList, Double[] pCenterList, UInt16 ArcDir, Int32 Cycles, UInt16 PosMode, UInt32 SegNum, Double MaxVel, Double LinkVel);

功 能：以圆心模式添加多轴圆弧运动

参 数：ConnectNo 板卡连接编号[0,15]

CrdNo 插补坐标系序号

AxisNum 轴数量

pAxisList 轴列表

pEndList 目标位置列表

pCenterList 圆心位置列表

ArcDir 圆弧的方向, 0：顺时针方向，1：逆时针方向

Cycles整圆的圈数，走完圆弧轨迹段后继续绕的整圆圈数。

PosMode 位置模式：0 - 绝对坐标模式；1 - 相对坐标模式

SegNum 该段插补的段号；0 默认会自动递增

MaxVel 该段的最大速度；Vel = 0 默认上段速度

LinkVel 过渡速度，-1 表示按照默认设置执行

返回值：错误代码

ERROR_CODE NV_CrdArcMoveR(UInt16 ConnectNo, UInt16 CrdNo, UInt16 AxisNum, UInt16[] pAxisList, Double[] pEndList, Double Radius, UInt16 ArcDir, Int32 Cycles, UInt16 PosMode, UInt32 SegNum, Double MaxVel, Double LinkVel)

功 能：以半径模式添加多轴圆弧运动

参 数：ConnectNo 板卡连接编号[0,15]

CrdNo 插补坐标系序号

AxisNum 轴数量

pAxisList 轴列表

pEndList 目标位置列表

Radius 圆弧的半径

正数：圆弧的旋转角度≤180°

负数：圆弧的旋转角度＞180°

ArcDir 圆弧的方向, 0：顺时针方向，1：逆时针方向

Cycles整圆的圈数，走完圆弧轨迹段后继续绕的整圆圈数。

PosMode 位置模式：0 - 绝对坐标模式；1 - 相对坐标模式

SegNum 该段插补的段号；0 默认会自动递增

MaxVel 该段的最大速度；Vel = 0 默认上段速度

LinkVel 过渡速度，-1 表示按照默认设置执行

返回值：错误代码

ERROR_CODE NV_CrdArcMove3P(UInt16 ConnectNo, UInt16 CrdNo, UInt16 AxisNum, UInt16[] pAxisList, Double[] pEndList, Double[] pInterList, Int32 Cycles, UInt16 PosMode, UInt32 SegNum, Double MaxVel, Double LinkVel);

功 能：以三点模式添加多轴圆弧运动

参 数：ConnectNo 板卡连接编号[0,15]

CrdNo 插补坐标系序号

AxisNum 轴数量

pAxisList 轴列表

pEndList 目标位置列表

pInterList 圆弧经过点位置列表

Cycles整圆的圈数，走完圆弧轨迹段后继续绕的整圆圈数。

PosMode 位置模式：0 -绝对坐标模式；1 - 相对坐标模式

SegNum　　该段插补的段号；0 默认会自动递增

MaxVel　　该段的最大速度；Vel = 0 默认上段速度

LinkVel　　过渡速度，-1 表示按照默认设置执行

返回值：错误代码

5.7.3 插补中其他操作

ERROR_CODE　NV_CrdBufDo(UInt16　ConnectNo,UInt16　CrdNo,UInt16 DoType,UInt16 DoGrpIndex,UInt32 DoValue,UInt32 DoMask,UInt32 SegNum)

功　能：在插补中插入缓存按组 IO 输出

参　数：ConnectNo　板卡连接编号[0,15]

CrdNo　插补坐标系序号，范围 0-7

DoType　　输出类型：0-1

0：普通 OUT

1：高速 OUT

DoGrpIndex　输出组号，每 32 个输出口为 1 组，从 0 开始编号

DoValue　　输出的 IO 组状态

DoMask　　设置输出的有效通道，每 bit 为 1 则输出，0 则不改变

SegNum　　该段插补的段号；0 默认会自动递增

返回值：错误代码

ERROR_CODE　NV_CrdBufDoBit(UInt16　ConnectNo,UInt16　CrdNo,UInt16 DoType,UInt16 DoIndex,UInt16 DoValue,UInt32 SegNum)

功　能：在插补中插入缓存按位 IO 输出

参　数：ConnectNo　板卡连接编号[0,15]

CrdNo　插补坐标系序号，范围 0-7

DoType　　输出类型：0-1

0：普通 OUT

1：高速 OUT

DoIndex 输出口编号，从 0 开始编号

DoValue 输出的 IO 状态

SegNum 该段插补的段号；0 默认会自动递增

返回值：错误代码

ERROR_CODE NV_CrdBufDoBitDelayToStart(UInt16 ConnectNo, UInt16 CrdNo, UInt16 AccessType, UInt16 DoType, UInt16 DoIndex, UInt16 DoValue, UInt16 DelayMode, Double DelayValue, Double ReverseTime, UInt32 SegNum)

功 能：在插补中相对于轨迹起点 IO 滞后输出

参 数：ConnectNo 板卡连接编号[0,15]

CrdNo 插补坐标系序号，范围 0-7

AccessType 访问类型

0：下段轨迹

1：整个轨迹

DoType 输出类型：0-1

0：普通 OUT

1：高速 OUT

DoIndex 输出口编号，从 0 开始编号

DoValue 输出的 IO 状态

DelayMode 延迟输出的模式

0：延迟输出的时间，单位 s

1：延迟输出的距离，单位 unit

DelayValue 延迟输出的值

如果模式为 0，该值表示时间

如果模式为 1，该值表示距离

ReverseTime 电平输出后的延时翻转时间，单位 s

SegNum 该段插补的段号；0 默认会自动递增

返回值：错误代码

ERROR_CODE NV_CrdBufDoBitDelayToStop(UInt16 ConnectNo, UInt16 CrdNo, UInt16 AccessType, UInt16 DoType, UInt16 DoIndex, UInt16 DoValue, Double DelayTime, Double ReverseTime, UInt32 SegNum)

功 能：在插补中相对于轨迹终点 IO 滞后输出

参 数：ConnectNo 板卡连接编号[0,15]

CrdNo 插补坐标系序号，范围 0-7

AccessType 访问类型

0：下段轨迹

1：整个轨迹

DoType 输出类型：0-1

0：普通 OUT

1：高速 OUT

DoIndex 输出口编号，从 0 开始编号

DoValue 输出的 IO 状态

DelayTime 延迟输出时间

ReverseTime 电平输出后的延时翻转时间，单位 s

SegNum 该段插补的段号；0 默认会自动递增

返回值：错误代码

ERROR_CODE NV_CrdBufDoBitAheadToStop(UInt16 ConnectNo, UInt16 CrdNo, UInt16 AccessType, UInt16 DoType, UInt16 DoIndex, UInt16 DoValue, UInt16 AheadMode, Double AheadValue, Double ReverseTime, UInt32 SegNum)

功 能：在插补中相对于轨迹终点 IO 提前输出

参　数：ConnectNo　板卡连接编号[0,15]

CrdNo　插补坐标系序号，范围 0-7

AccessType　访问类型

0：下段轨迹

1：整个轨迹

DoType　输出类型：0-1

0：普通 OUT

1：高速 OUT

DoIndex　输出口编号，从 0 开始编号

DoValue　输出的 IO 状态

AheadMode　提前输出的模式

0：提前输出的时间，单位 s

1：提前输出的距离，单位 unit

AheadValue　提前输出的值

如果模式为 0，该值表示时间

如果模式为 1，该值表示距离

ReverseTime

SegNum　该段插补的段号；0 默认会自动递增

返回值：错误代码

ERROR_CODE NV_CrdBufDoBitClearAction(UInt16 ConnectNo, UInt16 CrdNo, UInt16 DoType, UInt16 DoGrpIndex, UInt32 IoMask, UInt32 SegNum)

功　能：在插补中清除段内未执行完的 IO

参　数：ConnectNo　板卡连接编号[0,15]

CrdNo　插补坐标系序号，范围 0-7

DoType　输出类型：0-1

0：普通 OUT

1：高速 OUT

DoGrpIndex　输出组号，每 32 个输出口为 1 组，从 0 开始编号

IoMask　设置需要清除的通道，bit 位为 1 则清除，0 则不清除

SegNum　该段插补的段号；0 默认会自动递增

返回值：错误代码

ERROR_CODE NV_CrdBufDoBitReverseOut(UInt16 ConnectNo, UInt16 CrdNo, UInt16 DoType, UInt16 DoIndex, Double ReverseTime, UInt32 SegNum)

功　能：在插补中 IO 输出延时翻转

参　数：ConnectNo　板卡连接编号[0,15]

CrdNo　插补坐标系序号，范围 0-7

DoType　输出类型：0-1

0：普通 OUT

1：高速 OUT

DoIndex　输出口编号，从 0 开始编号

ReverseTime 延时翻转时间，单位 s

SegNum　该段插补的段号；0 默认会自动递增

返回值：错误代码

ERROR_CODE NV_CrdBufDoBitDelayOut(UInt16 ConnectNo, UInt16 CrdNo, UInt16 DoType, UInt16 DoIndex, UInt16 DoValue, Double DelayTime, UInt32 SegNum)

功　能：在插补中 IO 延时输出

参　数：ConnectNo　板卡连接编号[0,15]

CrdNo　插补坐标系序号，范围 0-7

DoType　输出类型：0-1

0：普通 OUT

1：高速 OUT

DoIndex　输出口编号，从 0 开始编号

DoValue　　输出的 IO 状态

DelayTime　延时的时间，单位 s

SegNum　　该段插补的段号；0 默认会自动递增

返回值：错误代码

ERROR_CODE NV_CrdDoSetPauseParam(UInt16 ConnectNo, UInt16 CrdNo, UInt16 DoType, UInt16 DoGrpIndex, UInt32 DoValue, UInt32 DoMask, UInt16 Action)

功　能：配置在插补中暂停或者异常时 IO 的操作。

参　数：ConnectNo　板卡连接编号[0,15]

CrdNo　插补坐标系序号，范围 0-7

DoType　　输出类型：0-1

0：普通 OUT

1：高速 OUT

DoGrpIndex　输出组号，每 32 个输出口为 1 组，从 0 开始编号

DoValue　　输出的 IO 状态

DoMask　设置 IO 的通道，bit 位为 1 则设置动作，0 则不设置

Action　暂停下 IO 的动作

SegNum　　该段插补的段号；0 默认会自动递增

返回值：错误代码

说明：

（1）DoMask 选择输出端口标志：bit0~bit31 代表 Out0~Out31，位值为 1 时输出，位值为 0 不输出；

（2）DoValue 输出电平状态：bit0~bit31 代表 Out0~Out31，位值为 1 时输出高电平，位值为 0 时输出低电平

（3）在插补中，执行暂停，配置 IO 的动作，Action 的其枚举值如下

```
enum PAUSED_ACTION
{
    PAUSED_ACTION_NONE = 0,              //无动作
```

PAUSED_ACTION_PAUSE_ONLY = 1, //仅在暂停时操作

PAUSED_ACTION_PAUSE_RESUME = 2, //在暂停和恢复的进行操作

PAUSED_ACTION_PAUSE_RESUME_STOP = 3, //在暂停，恢复，停止时进行操作

PAUSED_ACTION_PAUSE_STOP = 4,//在暂停，停止时进行操作

PAUSED_ACTION_STOP_ONLY = 5 //仅在停止时进行操作

};

ERROR_CODE NV_CrdDoGetPauseParam(UInt16 ConnectNo, UInt16 CrdNo, UInt16 DoType, UInt16 DoGrpIndex, UInt32[] pDoValue, UInt32[] pDoMask, UInt16[] pAction)

功 能：读取配置在插补中暂停或者异常时其 IO 的操作。

参 数：ConnectNo 板卡连接编号[0,15]

CrdNo 插补坐标系序号，范围 0-7

DoType 输出类型：0-1

0：普通 OUT

1：高速 OUT

DoGrpIndex 输出组号，每 32 个输出口为 1 组，从 0 开始编号

pDoValue 返回配置的输出的 IO 状态

pDoMask 返回配置的 IO 端口

pAction 返回配置的 IO 动作值

SegNum 该段插补的段号；0 默认会自动递增

返回值：错误代码

ERROR_CODE NV_CrdBufWaitIn(UInt16 ConnectNo,UInt16 CrdNo,UInt16 DiType,UInt16 DiIndex,UInt16 DiValue,Double TimeOut,UInt32 Action,UInt32 SegNum)

功 能：在插补中插入条件等待功能

参 数：ConnectNo 板卡连接编号[0,15]

CrdNo　插补坐标系序号，范围 0-7

DiType　输入类型

0：普通输入口

1：高速输入口

DiIndex　输入编号

DiValue　需要满足的输入数值

TimeOut　最大等待时间，单位 s

0 代表一直等待

Action　超时后的响应：0-2

0：执行后续运动

1：暂停运动

2：报警停止

SegNum　该段插补的段号；0 默认会自动递增

返回值：错误代码

ERROR_CODE NV_CrdBufSetDoBitPulse(UInt16 ConnectNo,UInt16 CrdNo, UInt16 DoType, UInt16 DoIndex,, UInt16 FirstLevel, Double FirstTime, Double SecondTime, UInt32 PulseNum, UInt16 Mode,UInt32 SegNum)

功　能：在插补中插入输出 IO 波形功能，输出端口类型为普通输出。

参　数：ConnectNo　板卡连接编号[0,15]

CrdNo　插补坐标系序号，范围 0-7

DoType　输出口类型

DoIndex　输出口编号

FirstLevel　起始电平

0：起始为低电平

1：起始为高电平

FirstTime　起始电平维持时间，单位 s，不可为 0

SecondTime　相反电平维持时间，单位 s

PulseNum 输出脉冲的个数

Mode 模式：0 - 不中断运动； 1 - 输出完成之后继续执行

SegNum　　该段插补的段号；0 默认会自动递增

返回值：错误代码

注意：电平维持时间，最短 0.001s 以上。

ERROR_CODE NV_CrdBufClrDoBitPulse(UInt16 ConnectNo,UInt16 CrdNo, UInt16 DoType, UInt16 DoIndex, UInt32 SegNum)

功　能：在插补中清除输出脉冲波形

参　数：ConnectNo　板卡连接编号[0,15]

CrdNo　插补坐标系序号，范围 0-7

DoType　　输出口类型

0：普通输出口

1：高速输出口

DoIndex　输出口编号

SegNum　　该段插补的段号；0 默认会自动递增

返回值：错误代码

ERROR_CODE NV_CrdBufDelay(UInt16 ConnectNo,UInt16 CrdNo,Double DelayTime,UInt32 SegNum)

功　能：在插补中插入延时等待功能

参　数：ConnectNo　板卡连接编号[0,15]

CrdNo　插补坐标系序号，范围 0-7

DelayTime等待时间，单位 s

0：表示一直等待

非 0：表示等待的时间

SegNum　　该段插补的段号；0 默认会自动递增

返回值：错误代码

ERROR_CODE NV_CrdBufDA(UInt16 ConnectNo,UInt16 CrdNo,UInt16 DANo, UInt16 Enable, UInt16 Mode, Double MaxVel, Double Value,UInt32 SegNum)

功　能：在插补中插入 DA 输出功能

参　数：ConnectNo　板卡连接编号[0,15]

CrdNo　插补坐标系序号，范围 0-7

DANo　DA 的通道序号

Enable　DA 输出通道使能

Mode　DA 输出的模式

0：不跟随，保持设置值 DAValue 输出

1：跟随，DA 值自动调整，最大速度下输出值为 DAValue

MaxVel　DA 跟随最大速度，单位：unit/s

Value　　DA 输出值，单位 V

在输出模式 0 下，按照该值输出

在输出模式 1 下，该值为最大速度下输出值

SegNum　　该段插补的段号；0 默认会自动递增

返回值：错误代码

ERROR_CODE NV_CrdBufPWM(UInt16 ConnectNo,UInt16 CrdNo,UInt16 PwmNo,UInt16 Enable, UInt16 Mode, Double MaxVel, Double Duty, Double Frequancy,UInt32 SegNum)

功　能：在插补中插入 PWM 输出功能

参　数：ConnectNo　板卡连接编号[0,15]

CrdNo　插补坐标系序号，范围 0-7

PwmNo　　PWM 的通道序号

Enable　　PWM 的通道使能

Mode　　PWM 的跟随模式

0：不跟随，保持设置输出

1：跟随，占空比自动调整

2：跟随，频率自动调整

MaxVel　最大合速度，单位：unit/s

自动调整时，跟随的最大速度；

在达到这个最大速度时，PWM 占空比或者频率达到最大值

Duty　PWM 输出的波形占空比，范围：0-1

Frequancy PWM 输出的波形频率，范围：0-500KHz

SegNum　该段插补的段号；0 默认会自动递增

返回值：错误代码

ERROR_CODE NV_CrdBufPWMAheadStop(UInt16 ConnectNo,UInt16 CrdNo,UInt16 PwmNo,UInt16 AheadMode, Double AheadValue,UInt32 SegNum)

功　能：在插补轨迹段终点 PWM 提前停止功能

参　数：ConnectNo　　板卡连接编号[0,15]

CrdNo　　插补坐标系序号，范围 0-7

PwmNo　　PWM 的通道序号

AheadMode　PWM 提前关闭的模式

0：提前关闭的时间，单位 s

1：提前关闭的距离，单位 unit

AheadValue　PWM 提前关闭的值

如果模式为 0，该值表示时间

如果模式为 1，该值表示距离

SegNum　　该段插补的段号；0 默认会自动递增

返回值：错误代码

ERROR_CODE NV_CrdBufPmove(UInt16 ConnectNo,UInt16 CrdNo,UInt16 AxisNo,Double Distance,Double Vel,Double Acc,UInt16 PosMode,UInt32 SegNum)

功　能：在插补中启动坐标系外的轴进行定长运动

参　数：ConnectNo　板卡连接编号[0,15]

CrdNo　插补坐标系序号，范围 0-7

AxisNo　要启动的轴序号，注意不是坐标系内的轴

Distance　轴定长运动的目标位置或者距离

Vel　轴定长运动的最大速度

Acc　轴定长运动的加速度

PosMode 运动模式：0 - 绝对坐标模式；1 - 相对坐标模式

SegNum　该段插补的段号；0 默认会自动递增

返回值：错误代码

ERROR_CODE NV_CrdBufGear(UInt16 ConnectNo,UInt16 CrdNo,UInt16 AxisNo,Double Distance,UInt32 SegNum)

功　能：在插补中，启动坐标系外的轴跟随轨迹运动一段距离

参　数：ConnectNo　板卡连接编号[0,15]

CrdNo　插补坐标系序号，范围 0-7

AxisNo　要启动的轴序号，注意不是坐标系内的轴

Distance　要等比例运动的距离

SegNum　该段插补的段号；0 默认会自动递增

返回值：错误代码

注：要跟随的是要有实际运动的插补段，在这段运动完成的时候，该轴会移动 Distance 的距离；在插补器速度为 0 零，跟随轴速度也会为 0。该功能主要用于实现刀向跟随。

ERROR_CODE NV_CrdBufFollow(UInt16 ConnectNo,UInt16 CrdNo,UInt16 AxisNo,Double Ratio,UInt32 SegNum)

功　能：在插补中，启动坐标系外的轴跟随轨迹运动，按比例跟随插补速度运动

参　数：ConnectNo　板卡连接编号[0,15]

CrdNo　插补坐标系序号，范围 0-7

AxisNo　要启动的轴序号，注意不是坐标系内的轴

Ratio　插补速度的比例

SegNum　该段插补的段号；0 默认会自动递增

返回值：错误代码

ERROR_CODE NV_CrdBufVmove(UInt16 ConnectNo,UInt16 CrdNo,UInt16 AxisNo,Double Vel,UInt32 SegNum)

功　能：在插补中，启动坐标系外的轴开始定速运动

参　数：ConnectNo　板卡连接编号[0,15]

CrdNo　插补坐标系序号，范围 0-7

AxisNo　要启动的轴序号，注意不是坐标系内的轴

Vel　运动速度：

Vel>0，正向运动

Vel=0，暂停

Vel<0，负向运动

SegNum　该段插补的段号；0 默认会自动递增

返回值：错误代码

ERROR_CODE NV_CrdBufStop(UInt16 ConnectNo,UInt16 CrdNo,UInt16 AxisNo,UInt16 StopMode,UInt32 SegNum)

功　能：在插补中停止坐标系外的轴

参　数：ConnectNo　板卡连接编号[0,15]

CrdNo 插补坐标系序号，范围 0-7

AxisNo 要启动的轴序号，注意不是坐标系内的轴

StopMode 停止模式：0 - 减速停止；1 - 立即停止

SegNum 该段插补的段号；0 默认会自动递增

返回值：错误代码

5.7.4 矩形插补

ERROR_CODE NV_CrdRectangle(UInt16 ConnectNo, UInt16 CrdNo, UInt16 AxisNum, UInt16[] pAxisList, Double[] pEndList, Double[] pMarkList, UInt16 RectMode, Int32 Cycles,UInt16 PosMode, UInt32 SegNum, Double MaxVel, Double LinkVel)

功 能：连续插补中矩形插补指令

参 数：ConnectNo 板卡连接编号[0,15]

CrdNo 插补坐标系序号，范围 0-7

AxisNum 轴数量。保留参数，固定值为 2

pAxisList 轴列表

pEndList 对角位置数组，单位：unit

pMarkList 矩形方向标记位置数组，单位：unit

RectMode 矩形插补模式，0：逐行，1：渐开线

Cycles 行数/圈数

PosMode 运动模式：0 - 绝对坐标模式；1 - 相对坐标模式

SegNum 该段插补的段号；0 默认会自动递增

MaxVel 该段的最大速度；-1 表示同上上段速度

LinkVel 过渡速度；-1 表示按照默认设置过渡

返回值：错误代码

说明：

该指令只支持 2 轴矩形插补运动

5.7.5 椭圆插补

ERROR_CODE NV_CrdEllipse (UInt16 ConnectNo, UInt16 CrdNo, UInt16 AxisNum, UInt16[] pAxisList, Double[] pEndList, Double[] pCenterList, Double Alen, Double Blen, UInt16 ArcDir, UInt16 PosMode, UInt32 SegNum, Double MaxVel, Double LinkVel)

功 能：启动椭圆插补运动

参 数：ConnectNo 板卡连接编号[0,15]

CrdNo 插补坐标系序号，范围 0-7

AxisNum 轴数量。保留参数，固定值为 2

pAxisList 轴列表

pEndList 对角位置数组，单位：unit

pCenterList 椭圆圆心位置，单位：unit

Alen 长半轴长度，单位：unit

Blen 短半轴长度，单位：unit

ArcDir 插补方向：0 - 顺时针方向；1 - 逆时针方向

PosMode 运动模式：0 - 绝对坐标模式；1 - 相对坐标模式

SegNum 该段插补的段号；0 默认会自动递增

MaxVel 该段的最大速度；-1 表示同上上段速度

LinkVel 过渡速度；-1 表示按照默认设置过渡

返回值：错误代码

5.7.6 单段插补

直线插补

ERROR_CODE NV_LnMove(UInt16 ConnectNo, UInt16 CrdNo, UInt16 AxisNum, UInt16[] pAxisList, Double[] pEndList, UInt16 PosMode = 0, Double MaxVel = -1)

功 能：添加任意轴单段直线插补

参 数：ConnectNo 板卡连接编号[0,15]

CrdNo 插补坐标系序号，范围 0-7

AxisNum　　轴数量，范围 1-8

pAxisList 轴列表

pEndList　轴位置列表

PosMode 运动模式：0 - 绝对坐标模式；1 - 相对坐标模式

MaxVel　　该段的最大速度；-1 表示同上上段速度

返回值：错误代码

注：轴列表序号对应的是 0 - X, 1 - Y, 2 - Z, 3 - A, 4 - B, 5 - C；

以圆心模式单段圆弧插补

ERROR_CODE NV_ArcMoveC(UInt16 ConnectNo, UInt16 CrdNo, UInt16 AxisNum, UInt16[] pAxisList, Double[] pEndList, Double[] pCenterList, UInt16 ArcDir, Int32 Cycles, UInt16 PosMode = 0, Double MaxVel = -1)

功　能：以圆心模式添加多轴单段圆弧运动

参　数：ConnectNo　板卡连接编号[0,15]

CrdNo　插补坐标系序号，范围 0-7

AxisNum　　轴数量，范围 1-8

pAxisList 轴列表

pEndList　轴目标位置列表，单位：unit

pCenterList 轴圆心位置列表，单位：unit

ArcDir　　圆弧的方向：0 - 顺时针方向；1 - 逆时针方向

Cycles圈数，走完指令圆弧段后继续绕整圆的圈数。

PosMode 运动模式：0 - 绝对坐标模式；1 - 相对坐标模式

MaxVel　　该段的最大速度；-1 表示同上上段速度

返回值：错误代码

以半径模式单段圆弧插补

ERROR_CODE NV_CrdArcMoveR(UInt16 ConnectNo, UInt16 CrdNo, UInt16 AxisNum, UInt16[] pAxisList, Double[] pEndList, Double Radius, UInt16 ArcDir, Int32

Cycles, UInt16 PosMode = 0, Double MaxVel = -1)

功　能：以半径模式添加多轴单段圆弧运动

参　数：ConnectNo　板卡连接编号[0,15]

CrdNo　插补坐标系序号，范围 0-7

AxisNum　轴数量，范围 1-8

pAxisList 轴列表

pEndList 轴目标位置列表，单位：unit

Radius　圆弧的半径

正数：圆弧的旋转角度≤180°

负数：圆弧的旋转角度＞180°

ArcDir　圆弧的方向：0 - 顺时针方向；1 - 逆时针方向

Cycles圈数，走完指令圆弧段后继续绕整圆的圈数。

PosMode 运动模式：0 - 绝对坐标模式；1 - 相对坐标模式

MaxVel　该段的最大速度；-1 表示同上上段速度

返回值：错误代码

以三点模式多轴圆弧插补

ERROR_CODE NV_ArcMove3P(UInt16 ConnectNo, UInt16 CrdNo, UInt16 AxisNum, UInt16[] pAxisList, Double[] pEndList, Double[] pInterList, Int32 Cycles, UInt16 PosMode = 0, Double MaxVel = -1);

功　能：以半径模式添加多轴单段圆弧运动

参　数：ConnectNo　板卡连接编号[0,15]

CrdNo　插补坐标系序号

AxisNum　轴数量

pAxisList 轴列表

pEndList 目标位置列表，单位：unit

pInterList　圆弧经过点位置列表，单位：unit

Cycles圈数，走完指令圆弧段后继续绕整圆的圈数。

PosMode 位置模式：0 -绝对坐标模式；1 - 相对坐标模式

MaxVel 该段的最大速度；Vel = 0 默认上段速度

返回值：错误代码

5.8 补偿功能

5.8.1 反向间隙补偿

ERROR_CODE NV_AxisSetBacklash(UInt16 ConnectNo,UInt16 AxisNo,UInt16 Enable,Double CompValue,Double CompChangeValue,Int16 CompDir)

功 能：设置反向间隙补偿功能

参 数：ConnectNo 板卡连接编号[0,15]

AxisNo 轴序号

Enable 使能：0 – 关闭；1 - 启用

CompValue 总补偿量

CompChangeValue 单周期补偿量

CompDir 补偿方向

0：双向补偿

1：正向补偿

-1：负向补偿

返回值：错误代码

ERROR_CODE NV_AxisGetBacklash(UInt16 ConnectNo,UInt16 AxisNo, ref UInt16 pEnable,ref Double pCompValue,ref Double pCompChangeValue,ref Int16 pCompDir)

功 能：获取反向间隙补偿功能状态

参 数：ConnectNo 控制卡卡号

AxisNo 轴序号

pEnable 返回使能

pCompValue 返回总补偿量

pCompChangeValue 返回单周期补偿量

pCompDir 返回补偿方向

返回值：错误代码

5.8.2 丝杆螺距补偿

ERROR_CODE NV_AxisSetPitchErr(UInt16 ConnectNo,UInt16 AxisNo, Int16 CompDir,UInt16 Enable,Double StartPos,Double Distance,Double[] pCompValue,UInt16 Count)

功 能：设置丝杆螺距补偿功能

参 数：ConnectNo 板卡连接编号[0,15]

AxisNo 轴序号

CompDir 设置补偿方向：1 - 正方向；-1 – 负方向

Enable 使能：0 – 关闭；1 - 启用

StartPos 起始点

Distance 补偿的长度范围

pCompValue 各点位置需要补偿的当量值

Count 补偿点的数量（加上起点和终点的数量，必须大于 2）

范围 2-1000

返回值：错误代码

说明：补偿表正、负方向需要分别创建；正、负方向的起始点和补偿长度可以不一致。

ERROR_CODE NV_AxisGetPitchErr(UInt16 ConnectNo,UInt16 AxisNo, Int16 CompDir,ref UInt16 pEnable,ref Double pStartPos,ref Double pDistance,Double[] pComp,ref UInt16 pCount)

功 能：获取丝杆螺距补偿功能状态

参 数：ConnectNo 板卡连接编号[0,15]

AxisNo 轴序号

pCompDir 补偿方向：1 - 正方向；-1 – 负方向

pEnable 返回使能

pStartPos 返回起始点

pDistance 返回补偿的长度范围

pCompValue 返回对应方向运动各点位置需要补偿的当量值

pCount 返回补偿点的数量

返回值：错误代码

5.9 锁存功能

5.9.1 软件锁存

ERROR_CODE NV_LatchSetMode(UInt16 ConnectNo,UInt16 Channel,UInt16 Mode,UInt16 DiType,UInt16 DiIndex,UInt16 DiLogic)

功 能：设置锁存功能参数。

参 数：ConnectNo 板卡连接编号[0,15]

Channel 锁存通道号：0-7

Mode 锁存方式：0-禁用，1-锁存一次，2-连续锁存

DiType 输入口类型

DiIndex 输入口编号

DiLogic 锁存的边沿，0-下降沿，1-上升沿，2-任意沿

返回值：错误代码

锁存触发 DiType 的类型：

输入口类型编号	输入口类型说明
0	通用输入口
1	高速输入口
2	原点输入口
3	正限位输入口

4	负限位输入口
5	编码器 Z 相输入口

ERROR_CODE NV_LatchGetMode(UInt16 ConnectNo,UInt16 Channel,ref UInt16 pMode,ref UInt16 pDiType,ref UInt16 pDiIndex,ref UInt16 pDiLogic)

功　能：获取锁存功能参数

参　数：ConnectNo　板卡连接编号[0,15]

Channel　锁存通道号：0-7

pMode　返回锁存的模式

pDiType　返回输入口类型

pDiIndex　返回输入口编号

pDiLogic　返回锁存的边沿，0-下降沿，1-上升沿，2-任意沿

返回值：错误代码

ERROR_CODE NV_LatchSetSource(UInt16 ConnectNo,UInt16 Channel,UInt16 SrcType,UInt16 SrcIndex)

功　能：设置锁存的数值的来源

参　数：ConnectNo　板卡连接编号[0,15]

Channel　锁存通道号：0-7

SrcType　锁存数据源类型

SrcIndex　锁存源的轴序号

返回值：错误代码

锁存数据源 SrcType 的类型：

锁存数据源	说明
0	轴指令位置（规划器位置）
1	编码器计数位置
2	轴反馈位置
3	脉冲计数器位置（内部参数）

10	轴指令速度
11	轴反馈速度
20	轴输入状态（排序：负限位，正限位，原点，急停、减速停，报警，ready，inp, ez）
21	轴输出状态（排序：伺服使能，伺服报警清除）

ERROR_CODE NV_LatchGetSource(UInt16 ConnectNo,UInt16 Channel,ref UInt16 pSrcType,ref UInt16 pSrcIndex)

功　能：获取锁存设定的数值的来源

参　数：ConnectNo　板卡连接编号[0,15]

Channel　锁存通道号：0-7

pSrcType　返回锁存的来源类型

pSrcIndex　返回锁存源的轴序号

返回值：错误代码

ERROR_CODE NV_LatchClear(UInt16 ConnectNo,UInt16 Channel)

功　能：清除锁存数据

参　数：ConnectNo　板卡连接编号[0,15]

Channel　锁存通道号：0-7

返回值：错误代码

ERROR_CODE NV_LatchStatus(UInt16 ConnectNo,UInt16 Channel,ref UInt16 pNumber,ref UInt16 pRemain)

功　能：获取锁存的锁存状态

参　数：ConnectNo　板卡连接编号[0,15]

Channel　锁存通道号：0-7

pNumber　返回锁存个数

pRemain　返回剩余锁存空间个数

返回值：错误代码

ERROR_CODE NV_LatchValue(UInt16 ConnectNo,UInt16 Channel,Double[] pValue,ref UInt16 pValueNum,UInt16 Count)

功　能：获取锁存的数值

参　数：ConnectNo　板卡连接编号[0,15]

Channel　锁存通道号：0-7

pValue　返回锁存的数值的数组，最小定义为 Count 大小的数组

pValueNum　返回锁存的数值的个数

Count　连续读取 Count 个锁存值

返回值：错误代码

5.9.2 高速锁存

ERROR_CODE NV_HSLatchSetMode(UInt16 ConnectNo,UInt16 Channel,UInt16 Mode,UInt16 DiType,UInt16 DiIndex,UInt16 DiLogic)

功　能：设置锁存功能。

参　数：ConnectNo　板卡连接编号[0,15]

Channel　锁存通道号：0-3

Mode　高速锁存模式：0-禁用，1-锁存一次，2-连续锁存

DiType　输入口类型

DiIndex　输入口编号

DiLogic　锁存的边沿，0-下降沿，1-上升沿，2-任意沿

返回值：错误代码

高速锁存触发信号 DiType 的类型：

高速锁存触发信号类型	说明
1	高速输入
2	原点输入

5	编码器 Z 相输入

ERROR_CODE NV_HSLatchGetMode(UInt16 ConnectNo,UInt16 Channel,ref UInt16 pMode,ref UInt16 pDiType,ref UInt16 pDiIndex,ref UInt16 pDiLogic)

功　能：获取锁存功能状态

参　数：ConnectNo　板卡连接编号[0,15]

Channel　锁存通道号：0-3

pMode　返回锁存的模式

pDiType　返回输入口类型

pDiIndex　返回输入口编号

pDiLogic　返回锁存的边沿

返回值：错误代码

ERROR_CODE NV_HSLatchSetSource(UInt16 ConnectNo,UInt16 Channel,UInt16 SrcType,UInt16 SrcIndex)

功　能：设置锁存的数值的来源

参　数：ConnectNo　板卡连接编号[0,15]

Channel　锁存通道号：0-3

SrcType　锁存源类型

SrcIndex　锁存源序号

返回值：错误代码

高速锁存的信号源 SrcType 的类型：

高速锁存的信号源类型	说明
0	编码器计数位置
1	轴指令位置
2	轴反馈位置

ERROR_CODE NV_HSLatchGetSource(UInt16 ConnectNo,UInt16 Channel,ref UInt16 pSrcType,ref UInt16 pSrcIndex)

功　能：获取锁存设定的数值的来源

参　数：ConnectNo　　板卡连接编号[0,15]

Channel　锁存通道号：0-3

pSrcType　　返回锁存的来源类型

pSrcIndex　　返回锁存源的序号

返回值：错误代码

ERROR_CODE NV_HSLatchClear(UInt16 ConnectNo,UInt16 Channel)

功　能：清除锁存数据

参　数：ConnectNo　　板卡连接编号[0,15]

Channel　锁存通道号：0-3

返回值：错误代码

ERROR_CODE NV_HSLatchStatus(UInt16 ConnectNo,UInt16 Channel,ref UInt16 pNumber,ref UInt16 pRemain)

功　能：获取锁存的数值的状态

参　数：ConnectNo　　板卡连接编号[0,15]

Channel　锁存通道号：0-3

pNumber　　返回锁存个数

pRemain　　返回剩余锁存空间个数

返回值：错误代码

ERROR_CODE NV_HSLatchValue(UInt16 ConnectNo,UInt16 Channel,Double[] pValue,ref UInt16 pValueNum,UInt16 Count)

功　能：获取锁存的数值

参　数：ConnectNo　　板卡连接编号[0,15]

Channel　锁存通道号：0-3

pValue　返回锁存的数值的数组，最小定义为 Count 大小的数组

pValueNum　返回锁存的数值的个数

Count　连续读取 Count 个锁存值

返回值：错误代码

5.10 位置比较功能

5.10.1 软件一维位置比较

ERROR_CODE NV_CmpSetEnable(UInt16 ConnectNo,UInt16 Channel, UInt16 Enable)

功　能：设置软件一维位置比较器使能

参　数：ConnectNo 板卡连接编号[0,15]

Channel　比较通道号：0-11

Enable　比较功能状态，0：禁止，1：使能

返回值：错误代码

ERROR_CODE NV_CmpGetEnable(UInt16 ConnectNo,UInt16 Channel, ref UInt16 pEnable)

功　能：读取软件一维位置比较器使能

参　数：ConnectNo　板卡连接编号[0,15]

Channel　比较通道号：0-11

pEnable 返回比较功能状态

返回值：错误代码

ERROR_CODE NV_CmpSetSource(UInt16 ConnectNo,UInt16 Channel, UInt16 SrcType,UInt16 SrcIndex)

功　能：设置软件一维位置比较器

参　数：ConnectNo 板卡连接编号[0,15]

Channel　比较通道号：0-11

SrcType　比较源类型：0-1

SrcIndex　比较源序号

返回值：错误代码

软件比较信号源 SrcType 的类型：

0	编码器计数位置
1	轴指令位置
2	轴反馈位置

ERROR_CODE NV_CmpGetSource(UInt16 ConnectNo,UInt16 Channel, ref UInt16 pSrcType,ref UInt16 pSrcIndex)

功　能：读取软件一维位置比较器设置

参　数：ConnectNo　板卡连接编号[0,15]

Channel　比较通道号：0-11

pSrcType　返回比较源类型：0-1

pSrcIndex　返回比较源序号

返回值：错误代码

ERROR_CODE NV_CmpClear(UInt16 ConnectNo,UInt16 Channel)

功　能：清除已添加的所有软件一维位置比较点

参　数：ConnectNo　板卡连接编号[0,15]

Channel 比较通道号：0-11

返回值：错误代码

ERROR_CODE NV_CmpAddPoint(UInt16 ConnectNo, UInt16 Channel, UInt16 Mode,

Double Pos, UInt16 Action, UInt32 Actpara1, Double Actpara2)

功　能：单个添加软件一维位置比较点

参　数：ConnectNo　板卡连接编号[0,15]

Channel　　比较通道号：0-11

Mode　比较模式，0：小于等于，1：大于等于

Pos 比较位置，单位 units

Action 比较点触发功能编号，不在范围不执行

ActParam1　比较点触发功能参数，不在范围不执行

ActParam2　比较点触发功能参数，不在范围不执行

返回值：错误代码

函数 Action, Actpara 参数值

Action **触发动作编号**	**功能描述**	**ActParam1** **动作参数 1**	**ActParam2** **动作参数 2**
1	IO 置为高电平	IO 口编号	无
2	IO 置为低电平	IO 口编号	无
3	IO 反相输出	IO 口编号	无
4	输出高电平脉冲	IO 口编号	输出高电平时间 s
5	输出低电平脉冲	IO 口编号	输出低电平时间 s
10	停止指定轴	控制轴号	停止时间 s
11	在线变速	控制轴号	新的速度
12	在线变位	控制轴号	新的位置，绝对位置

ERROR_CODE NV_CmpAddPoints(UInt16 ConnectNo, UInt16 Channel, UInt16[] pMode, Double[] pPos, UInt16[] pAction, UInt32[] pActParam1, Double[] pActParam2, UInt16 Count)

功　能：批量添加软件一维位置比较点

参　数：ConnectNo　板卡连接编号[0,15]

Channel 比较通道号：0-11

pMode[40] 比较模式，0：小于等于，1：大于等于

pPos[40] 比较位置，单位 units

pAction[40]比较点触发动作功能编号，不在范围不执行

pActPara1[40] 比较点触发动作功能参数 1，不在范围不执行

pActParam2[40] 比较点触发动作功能参数 2，不在范围不执行

Count 比较点数量

返回值：错误代码

ERROR_CODE NV_CmpStatus(UInt16 ConnectNo, UInt16 Channel, ref UInt16 pCmpMode, ref Double pCmpPos, ref UInt32 pCmpNum, ref UInt32 pRemain, ref UInt32 pSpace)

功 能：读取当前软件一维比较状态

参 数：ConnectNo 板卡连接编号[0,15]

Channel 比较通道号：0-11

pCmpMode 返回比较的模式，0：小于等于，1：大于等于

pCmpPos 返回正在比较的位置值

pCmpNum 返回已比较完成的数量

pRemain 返回剩余未比较的数量

pSpace 返回可添加的比较数量

返回值：错误代码

5.10.2 软件二维位置比较

ERROR_CODE NV_Cmp2DSetEnable(UInt16 ConnectNo,UInt16 Channel, UInt16 Enable)

功 能：设置软件二维位置比较器使能

参 数：ConnectNo 板卡连接编号[0,15]

Channel 比较通道号：0-1

Enable 比较功能状态，0：禁止，1：使能

返回值：错误代码

ERROR_CODE NV_Cmp2DGetEnable(UInt16 ConnectNo,UInt16 Channel, ref UInt16 pEnable)

功 能：读取软件二维位置比较器使能

参 数：ConnectNo 板卡连接编号[0,15]

Channel 比较通道号：0-1

pEnable 返回比较功能状态

返回值：错误代码

ERROR_CODE NV_Cmp2DSetSource(UInt16 ConnectNo,UInt16 Channel, UInt16[] pSrcType,UInt16[] pSrcIndex)

功 能：设置软件二维位置比较器

参 数：ConnectNo 板卡连接编号[0,15]

Channel 比较通道号：0-1

pSrcType[2] 比较源类型：0-2

pSrcIndex[2] 比较源序号

返回值：错误代码

软件比较的位置源类型	说明
0	编码器计数位置
1	轴指令位置
2	轴反馈位置

ERROR_CODE NV_Cmp2DGetSource(UInt16 ConnectNo,UInt16 Channel, UInt16[] pSrcType,UInt16[] pSrcIndex)

功　能：读取软件二维位置比较器设置

参　数：ConnectNo　　板卡连接编号[0,15]

Channel　比较通道号：0-1

pSrcType[2] 返回比较源类型

pSrcIndex[2]返回比较源序号

返回值：错误代码

ERROR_CODE NV_Cmp2DClear(UInt16 ConnectNo,UInt16 Channel)

功　能：清除已添加的所有软件二维位置比较点

参　数：ConnectNo　　板卡连接编号[0,15]

Channel　比较通道号：0-1

返回值：错误代码

ERROR_CODE NV_Cmp2DAddPoint(UInt16 ConnectNo, UInt16 Channel, UInt16[] pMode, Double[] pPos, UInt16 pAction, UInt32 pActpara1, Double pActpara2)

功　能：单点添加软件二维位置比较点

参　数：ConnectNo　　板卡连接编号[0,15]

Channel　比较通道号：0-1

pMode[2]比较模式，0：小于等于，1：大于等于

pPos[2] 比较位置，单位 units

pAction 比较点触发功能编号，不在范围不执行

pActpara1　比较点触发功能参数，不在范围不执行

返回值：错误代码

函数 Action, Actpara1, Actpara2 参数值

Action	功能	Actpara1	Actpara2
1	IO 置为高电平	IO 号	无
2	IO 置为低电平	IO 号	无

3	取反 IO	IO 号	无
4	输出高电平脉冲	IO 号	输出高电平时间 s
5	输出低电平脉冲	IO 号	输出低电平时间 s
10	停止指定轴	轴号	停止时间 s
11	在线变速	轴号	新的速度
12	在线变位	轴号	新的位置（绝对位置）

ERROR_CODE NV_Cmp2DAddPoints(UInt16 ConnectNo, UInt16 Channel, UInt16[] pMode1, Double[] pPos1, UInt16[] pMode2, Double[] pPos2, UInt16[] pAction, UInt32[] pActpara1, Double[] pActpara2, UInt16 Count)

功　能：批量添加软件二维位置比较点

参　数：ConnectNo　板卡连接编号[0,15]

Channel　比较通道号：0-1

pMode1[20]　比较 X 方向模式，0：小于等于，1：大于等于

pPos1[20]　比较 X 方向位置，单位 units

pMode2[20]　比较 Y 方向模式，0：小于等于，1：大于等于

pPos2[20]　比较 Y 方向位置，单位 units

pAction[20] 比较点触发功能编号，不在范围不执行

pActpara1[20]　比较点触发功能参数，不在范围不执行

pActpara2[20]　比较点触发功能参数，不在范围不执行

Count　比较点数量

返回值：错误代码

ERROR_CODE NV_Cmp2DStatus(UInt16 ConnectNo, UInt16 Channel, UInt16[] pCmpMode, Double[] pCmpPos, ref UInt32 pCmpNum, ref UInt32 pRemain， ref UInt32 pSpace)

功　能：读取当前软件二维比较状态

参　数：ConnectNo　板卡连接编号[0,15]

Channel　比较通道号：0-1

pCmpMode[2] 返回比较模式，0：小于等于，1：大于等于

pCmpPos[2]　返回正在比较的位置值

pCmpNum　返回已比较完成的数量

pRemain　返回剩余未比较的数量

pSpace　返回可添加的比较数量

返回值：错误代码

5.10.3 高速一维位置比较

ERROR_CODE NV_HSCmpSetMode(UInt16 ConnectNo, UInt16 Channel, UInt16 Mode, UInt16 DoIndex)

功　能：设置高速一维比较模式

参　数：ConnectNo　控制卡卡号

Channel　比较通道号：0-3

Mode 比较模式：0-5

0：禁用（默认值）

1：小于等于

2：大于等于

3：线性比较，提供起始比较点，位置增量，比较次数

4：队列比较，采用先添加先比较，比较完可追加比较点，

也可一次性添加多个比较点

其他值：不处理

DoIndex　高速比较输出口：0-7

返回值：错误代码

ERROR_CODE NV_HSCmpGetMode(UInt16 ConnectNo, UInt16 Channel, ref UInt16

pMode, ref UInt16 pDoIndex)

功　能：读取高速一维比较模式

参　数：ConnectNo 板卡连接编号[0,15]

Channel　　比较通道号：0-3

pMode 返回比较模式设置

pDoIndex　返回高速比较输出口

返回值：错误代码

ERROR_CODE NV_HSCmpSetSource(UInt16 ConnectNo, UInt16 Channel, UInt16 SrcType, UInt16 SrcIndex)

功　能：设置高速一维比较位置源

参　数：ConnectNo 控制卡卡号

Channel　比较通道号：0-3

SrcType　比较位置源：0-2

SrcIndex 比较源索引（轴/编码器编号）

返回值：错误代码

比较的位置源类型	说明
0	编码器计数位置
1	轴指令位置
2	轴反馈位置

ERROR_CODE NV_HSCmpGetSource(UInt16 ConnectNo, UInt16 Channel, ref UInt16 pSrcType, ref UInt16 pSrcIndex)

功　能：读取高速一维比较位置源

参　数：ConnectNo 板卡连接编号[0,15]

Channel　比较通道号：0-3

pSrcType　比较位置源

pSrcIndex 比较源索引

返回值：错误代码

ERROR_CODE NV_HSCmpSetOuputMode(UInt16 ConnectNo, UInt16 Channel, UInt16 DoMode, UInt16 DoLogic, Double Time)

功　能：设置高速一维比较输出模式

参　数：ConnectNo　控制卡卡号

Channel　比较通道号：0-3

DoMode　输出模式：0-输出翻转脉冲； 1-输出 PWM 波形

DoLogic 有效电平：0-低电平，1-高电平

DoTime　脉冲宽度，单位：us，范围：1us-134s

返回值：错误代码

ERROR_CODE NV_HSCmpGetOuputMode(UInt16 ConnectNo, UInt16 Channel, ref UInt16　pDoMode,ref UInt16 pDoLogic, ref Double pTime)

功　能：读取高速一维比较输出模式

参　数：ConnectNo　控制卡卡号

Channel　比较通道号：0-3

pDoMode 返回输出模式

pDoLogic 返回有效电平设置

pDoTime 返回脉冲宽度设置

返回值：错误代码

ERROR_CODE NV_HSCmpSetOutPulse(UInt16 ConnectNo, UInt16 Channel, UInt16 FirstLogic, Double FirstTime, Double SecondTime, UInt32 PulseNum)

功　能：配置高速一维比较输出信号及波形

参　数：ConnectNo 控制卡卡号

Channel　比较通道号：0-3

FirstLogic 比较生效时的起始电平；

0：起始为低电平脉冲

1：起始为高电平脉冲

2：输出翻转

FirstTime 比较生效时的起始电平维持时间，单位 us，范围：1us-134s

SecondTime 比较生效时的相反电平维持时间，单位 us，范围：1us-134s

PulseNum 比较生效时输出的 PWM 波形个数

0：表示持续输出

非 0：表示输出的个数

返回值：错误代码

ERROR_CODE NV_HSCmpGetOutPulse(UInt16 ConnectNo, UInt16 Channel, ref UInt16 pFirstLogic, ref Double pFirstTime, ref Double pSecondTime, ref UInt32 pPulseNum)

功 能：读取高速一维比较输出信号及波形

参数：ConnectNo 板卡连接编号[0,15]

Channel 比较通道号：0-3

pFirstLogic 返回比较生效时的起始电平

pFirstTime 返回比较生效时的起始电平维持时间

pSecondTime 返回比较生效时的相反电平维持时间

pPulseNum 返回比较生效时发送的 PWM 波形个数

返回值：错误代码

ERROR_CODE NV_HSCmpStartPulse(UInt16 ConnectNo, UInt16 Channel)

功 能：强制启动比较器的脉冲输出

参 数：ConnectNo 板卡连接编号[0,15]

Channel 比较通道号：0-3

返回值：错误代码

ERROR_CODE NV_HSCmpStopPulse(UInt16 ConnectNo, UInt16 Channel)

功　能：停止比较器的脉冲输出

参数：ConnectNo　板卡连接编号[0,15]

　　Channel　比较通道号：0-3

返回值：错误代码

ERROR_CODE NV_HSCmpClear(UInt16 ConnectNo, UInt16 Channel)

功　能：清除已添加的所有高速位置比较点

参　数：ConnectNo　　控制卡卡号

　　Channel　比较通道号：0-3

返回值：错误代码

ERROR_CODE NV_HSCmpSetPoint(UInt16 ConnectNo, UInt16 Channel, Double CmpPos)

功　能：设置高速一维比较模式 1 和 2 的单点位置

参　数：ConnectNo　　控制卡卡号

　　Channel　比较通道号：0-3

　　CmpPos　添加比较位置，单位：units

返回值：错误代码

ERROR_CODE NV_HSCmpGetPoint(UInt16 ConnectNo, UInt16 Channel, ref Double pCmpPos)

功　能：读取高速一维比较模式 1 和 2 的单点位置

参　数：ConnectNo　　控制卡卡号

　　Channel　比较通道号：0-3

　　pCmpPos 返回比较位置，单位：units

返回值：错误代码

ERROR_CODE NV_HSCmpAddPoints(UInt16 ConnectNo, UInt16 Channel, Double[] pValue, UInt16 Count)

功　能：批量添加高速一维比较队列模式位置点

参数：ConnectNo　板卡连接编号[0,15]

Channel　　比较通道号：0-3

pValue[100]比较的位置数组

Count　添加的数量

返回值：错误代码

ERROR_CODE NV_HSCmpSetLinear(UInt16 ConnectNo, UInt16 Channel, Double StartPos，Double Increment, UInt32 Count)

功　能：设置高速一维比较线性模式参数

参　数：ConnectNo 控制卡卡号

Channel　　比较通道号：0-3

StartPos 线性比较的起始位置点

Increment 位置增量值，单位：units（正值表示位置递增，负值表示位置递减）

Count 比较次数，取值范围：1~65535

返回值：错误代码

ERROR_CODE NV_HSCmpGetLinear(UInt16 ConnectNo, UInt16 Channel, ref Double pStartPos，ref Double pIncrement, ref UInt32 pCount)

功　能：读取高速一维比较线性模式参数设置

参　数：ConnectNo 控制卡卡号

Channel　　比较通道号：0-3

pStartPos 返回起始位置点

pIncrement 返回位置增量值

pCount 返回比较次数

返回值：错误代码

ERROR_CODE NV_HSCmpSetOffset (UInt16 ConnectNo, UInt16 Channel, Double PosOffset)

功 能：设置高速一维比较比较点的整体偏移位置值，对所有点都有效。

参 数：ConnectNo 板卡连接编号[0,15]

Channel 比较通道号：0-3

PosOffset 比较位置的整体偏移值

返回值：错误代码

ERROR_CODE NV_HSCmpGetOffset (UInt16 ConnectNo, UInt16 Channel, ref Double pPosOffset)

功 能：读取高速一维比较比较点的整体偏移位置值

参 数：ConnectNo 板卡连接编号[0,15]

Channel 比较通道号：0-3

pPosOffset 比较位置的整体偏移值

返回值：错误代码

ERROR_CODE NV_HSCmpStatus(UInt16 ConnectNo, UInt16 Channel, ref UInt16 pCmpMode, ref Double pCmpPos, ref UInt32 pCmpNum, ref UInt32 pRemain, ref UInt32 pSpace)

功 能：获取高速一维比较的比较器的状态

参数：ConnectNo 板卡连接编号[0,15]

Channel 比较通道号：0-3

pCmpMode 返回比较的模式

pCmpPos 返回正在比较的位置值

pCmpNum返回已比较完成的数量

pRemain 返回剩余未比较的数量

pSpace 返回可添加的比较数量

返回值：错误代码

5.10.4 高速二维位置比较

ERROR_CODE NV_HSCmp2DSetMode(UInt16 ConnectNo, UInt16 Channel, UInt16 Mode, UInt16 DoIndex)

功 能：设置高速二维比较模式

参数：ConnectNo 控制卡卡号

Channel 比较通道号：0-1

Mode 比较模式：0-2

0：禁止（默认值）

1：进入误差带触发

2：离开误差带触发

3：PSO 模式，立即触发

4：PSO 模式，进入误差带触发

DoIndex 高速比较输出口：0-7

返回值：错误代码

ERROR_CODE NV_HSCmp2DSetPSOPara(UInt16 ConnectNo, UInt16 Channel, Double Distance)

功 能：设置高速二维 PSO 输出模式参数

参 数：ConnectNo 控制卡卡号

Channel 比较通道号：0-1

Distance　PSO 输出间距，大于 0，两轴合成距离，单位：units

返回值：错误代码

ERROR_CODE NV_HSCmp2DGetPSOPara(UInt16 ConnectNo, UInt16 Channel, ref Double pDistance)

功　能：读取高速二维 PSO 输出模式参数

参　数：ConnectNo　控制卡卡号

Channel　比较通道号：0-1

pDistance 返回 PSO 输出间距。单位：units

返回值：错误代码

ERROR_CODE NV_HSCmp2DGetMode(UInt16 ConnectNo, UInt16 Channel, ref UInt16 pMode, ref UInt16 pDoIndex)

功　能：读取高速二维比较模式设置

参数：ConnectNo 板卡连接编号[0,15]

Channel 比较通道号：0-1

pMode 返回比较模式设置

pDoIndex 返回高速比较输出口

返回值：错误代码

ERROR_CODE NV_HSCmp2DSetSource(UInt16 ConnectNo, UInt16 Channel, UInt16[] SrcType, UInt16[] SrcIndex)

功　能：设置高速二维比较位置源

参数：ConnectNo 控制卡卡号

Channel　比较通道号：0-1

SrcType　比较位置源：0-1

SrcIndex 比较源索引

返回值：错误代码

比较的位置源类型	说明
0	编码器计数位置
1	轴指令位置
2	轴反馈位置

ERROR_CODE NV_HSCmp2DGetSource(UInt16 ConnectNo, UInt16 Channel, UInt16[] pSrcType, UInt16[] pSrcIndex)

功　能：读取高速二维比较器设置的位置源

参数：ConnectNo 控制卡卡号

Channel　　比较通道号：0-1

pSrcType 比较位置源

pSrcIndex 比较源索引

返回值：错误代码

ERROR_CODE NV_HSCmp2DSetErrorBand(UInt16 ConnectNo, UInt16 Channel, Double[] pErrorBand)

功　能：设置高速二维比较器误差带

参数：ConnectNo　板卡连接编号[0,15]

Channel　比较通道号：0-1

pErrorBand[2]　比较误差带，单位 units

返回值：错误代码

ERROR_CODE NV_HSCmp2DGetErrorBand(UInt16 ConnectNo, UInt16 Channel, Double[] pErrorBand)

功　能：读取高速二维比较器误差带

参数：ConnectNo 板卡连接编号[0,15]

Channel　比较通道号：0-1

pErrorBand[2]比较误差带

返回值：错误代码

ERROR_CODE NV_HSCmp2DSetOuputMode(UInt16 ConnectNo, UInt16 Channel, UInt16 DoMode, UInt16 DoLogic, double Time)

功　能：设置高速二维比较器输出模式

参数：ConnectNo　控制卡卡号

Channel　比较通道号：0-1

DoMode　返回输出模式：0 - 输出翻转； 1 - 输出 PWM 波形

DoLogic 有效电平：0-低电平，1-高电平

DoTime　脉冲宽度，单位：us，取值范围：1us-134s

返回值：错误代码

ERROR_CODE NV_HSCmp2DGetOuputMode(UInt16 ConnectNo, UInt16 Channel, ref UInt16 pDoMode,ref UInt16 pDoLogic, ren double pTime)

功　能：读取高速二维比较器输出模式

参数：ConnectNo　控制卡卡号

Channel　比较通道号：0-1

pDoMode　返回输出模式

pDoLogic 返回有效电平设置

pDoTime 返回脉冲宽度设置

返回值：错误代码

ERROR_CODE NV_HSCmp2DSetOutPulse(UInt16 ConnectNo, UInt16 Channel,

UInt16 FirstLogic, Double FirstTime, Double SecondTime, UInt32 PulseNum)

功　能：设置高速二维比较器的输出信号及波形

参数：ConnectNo　板卡连接编号[0,15]

Channel　比较通道号：0-1

FirstLogic 比较生效时的起始电平：0-2

0：起始为低电平脉冲

1：起始为高电平脉冲

2：输出翻转

FirstTime　比较生效时的起始电平维持时间，单位 us，取值范围：1us-134s

SecondTime　比较生效时的相反电平维持时间，单位 us，取值范围：1us-134s

PulseNum 比较生效时的发送的 PWM 波形个数（0 – 持续输出）

返回值：错误代码

ERROR_CODE NV_HSCmp2DGetOutPulse(UInt16 ConnectNo, UInt16 Channel, ref UInt16 pFirstLogic, ref Double pFirstTime, ref Double pSecondTime, ref UInt32 pPulseNum)

功　能：读取高速二维比较器的输出信号及波形

参数：ConnectNo　板卡连接编号[0,15]

Channel　比较通道号：0-1

pFirstLogic　返回比较生效时的起始电平

pFirstTime 返回比较生效时的起始电平维持时间

pSecondTime　返回比较生效时的相反电平维持时间

pPulseNum 返回比较生效时的发生的 PWM 波形个数（0 – 持续输出）

返回值：错误代码

ERROR_CODE NV_HSCmp2DStartPulse(UInt16 ConnectNo, UInt16 Channel)

功　能：强制启动比较器的脉冲输出

参　数：ConnectNo 板卡连接编号[0,15]

Channel　　比较通道号：0-3

返回值：错误代码

ERROR_CODE NV_HSCmp2DStopPulse(UInt16 ConnectNo, UInt16 Channel)

功　能：停止比较器的脉冲输出

参数：ConnectNo　板卡连接编号[0,15]

Channel　比较通道号：0-3

返回值：错误代码

ERROR_CODE NV_HSCmp2DClearPoints(UInt16 ConnectNo, UInt16 Channel)

功　能：清除已添加的所有高速位置比较点

参数：ConnectNo　板卡连接编号[0,15]

Channel　比较通道号：0-1

返回值：错误代码

ERROR_CODE NV_HSCmp2DAddPoints(UInt16 ConnectNo, UInt16 Channel,Double[] pPosX, Double[] pPosY, UInt16 Count)

功　能：设置高速二维比较器的队列先入先出比较点

参数：ConnectNo 控制卡卡号

Channel　比较通道号：0-1

pPosX X 方向比较的位置数组

pPosY Y 方向比较的位置数组

Count 输入位置的数量

返回值：错误代码

ERROR_CODE NV_HSCmp2DStatus(UInt16 ConnectNo, UInt16 Channel, ref UInt16 pCmpMode,　Double[] pCmpPos, ref UInt32 pCmpNum, ref UInt32 pRemain, ref UInt32

pSpace)

功　能：读取高速二维比较器的状态

参数：ConnectNo　板卡连接编号[0,15]

Channel　比较器的通道序号

pCmpMode　返回比较的模式

pCmpPos[2]返回正在比较的位置值

pCmpNum返回已比较完成的数量

pRemain　返回剩余未比较的数量

pSpace　返回可添加的比较数量

返回值：错误代码

5.11 EtherCAT 总线操作

5.11.1 总线配置

ERROR_CODE NV_EcatReset(UInt16 ConnectNo，UInt16 MasterNo)

功　能：复位 EtherCAT 总线

参数：ConnectNo　板卡连接编号[0,15]

MasterNoEtherCAT 主站编号

返回值：错误代码

ERROR_CODE NV_EcatSetCycleTime(UInt16 ConnectNo, UInt16 MasterNo, Double CycleTime)

功 能：设置 EtherCAT 总线循环周期

参 数：ConnectNo 板卡连接编号[0,15]

MasterNo　EtherCAT 主站编号

CycleTime 总线循环周期，单位：us

可配置的周期：62.5,125,250,500,1000,2000,4000

返回值：错误代码

ERROR_CODE NV_EcatGetCycleTime (UInt16 ConnectNo, UInt16 MasterNo, ref Double pCycleTime)

功 能：读取 EtherCAT 总线循环周期

参 数：ConnectNo 板卡连接编号[0,15]

MasterNo EtherCAT 主站编号

pCycleTime 总线循环周期

返回值：错误代码

ERROR_CODE NV_EcatGetSlaveNum （ UInt16 ConnectNo, UInt16 MasterNo, ref UInt16 pSlaveNum)

功 能：获取 EtherCAT 从站总数

参 数：ConnectNo 板卡连接编号[0,15]

MasterNo EtherCAT 主站编号

pSlaveNum 从站总数

返回值：错误代码

ERROR_CODE NV_EcatGetLostHeartNodes (UInt16 ConnectNo, UInt16 MasterNo, UInt16[] pNodeID， ref UInt16 pNodeNum)

功 能：获取心跳报文信息

参数： ConnectNo 板卡连接编号[0,15]

MasterNo EtherCAT 主站编号

pNodeID 节点号

pNodeNum 节点个数

返回值：错误代码

ERROR_CODE NV_EcatGetErrcode(UInt16 ConnectNo, UInt16 MasterNo,ref UInt32 pValue)

功　能：获取总线错误码

参　数：ConnectNo　控制卡卡号

MasterNoEtherCAT 主站编号

pValue　总线错误码实际值

返回值：错误代码

ERROR_CODE NV_EcatClrErrcode(UInt16 ConnectNo, UInt16 MasterNo)

功　能：清除总线错误码

参　数：ConnectNo 板卡连接编号[0,15]

MasterNo EtherCAT 主站编号

返回值：错误代码

5.11.2 总线轴操作

ERROR_CODE NV_AxisGetErrcode(UInt16 ConnectNo,UInt16 AxisNo,UInt32[] pValue,, UInt16 Count)

功　能：获取总线轴错误码

参　数：ConnectNo　板卡连接编号[0,15]

MasterNoEtherCAT 主站编号

AxisNo　轴序号

pValue　返回该轴错误码

Count　返回错误码的数量

返回值：错误代码

ERROR_CODE NV_AxisClrErrcode(UInt16 ConnectNo,UInt16 AxisNo)

功　能：清除总线轴的错误码

参　数：ConnectNo　板卡连接编号[0,15]

MasterNoEtherCAT 主站编号

AxisNo　轴序号

返回值：错误代码

5.11.3 总线模块操作

ERROR_CODE NV_EcatGetSlaveIoNum(UInt16 ConnectNo, UInt16 MasterNo, UInt16 SlaveId, ref UInt16 pInputNum, ref UInt16 pOutputNum)

功　能：获取从站节点 IO 资源信息

参　数：ConnectNo　板卡连接编号[0,15]

MasterNoEtherCAT 主站编号

SlaveId　对应的总线节点号

pInputNum　返回支持的输入数量

pOutputNum　返回支持的输出数量

返回值：错误代码

ERROR_CODE NV_EcatGetDiBit(UInt16 ConnectNo, UInt16 MasterNo,UInt16 SlaveId,UInt16 DiIndex,UInt16[] pValue, UInt16 Count)

功　能：获取总线节点的单个输入状态

参　数：ConnectNo　板卡连接编号[0,15]

MasterNoEtherCAT 主站编号

SlaveId　对应的总线节点号

DiIndex　对应的输入 IO 序号

pValue　返回该输入的实际电平

Count　获取参数值个数

返回值：错误代码

ERROR_CODE NV_EcatGetDi(UInt16 ConnectNo, UInt16 MasterNo,UInt16 SlaveId,UInt16 DiGrpIndex,ref UInt32 pValue)

功　能：获取总线节点的单组输入状态

参　数：ConnectNo　板卡连接编号[0,15]

MasterNoEtherCAT 主站编号

SlaveId　对应的总线节点号

DiGrpIndex　对应的组序号。32 点为一组，序号从 0 开始

pValue　返回该组输入的实际电平

返回值：错误代码

ERROR_CODE NV_EcatSetDoBit(UInt16 ConnectNo, UInt16 MasterNo,UInt16 SlaveId,UInt16 DoIndex,UInt16 Value)

功　能：设置总线节点的单个输出电平

参　数：ConnectNo　板卡连接编号[0,15]

MasterNoEtherCAT 主站编号

SlaveId　对应的总线节点号

DoIndex　对应的输出 IO 序号。32 点为一组，序号从 0 开始

Value 设置该输出的实际电平

返回值：错误代码

ERROR_CODE NV_EcatGetDoBit(UInt16 ConnectNo, UInt16 MasterNo,UInt16 SlaveId,UInt16 DoIndex,UInt16[] pValue, UInt16 Count)

功　能：获取总线节点的单个输出电平

参　数：ConnectNo　板卡连接编号[0,15]

MasterNoEtherCAT 主站编号

SlaveId　对应的总线节点号

DoIndex 对应的输出 IO 序号。32 点为一组，序号从 0 开始

pValue 返回该输出的实际电平

Count 获取参数值个数

返回值：错误代码

ERROR_CODE NV_EcatSetDo(UInt16 ConnectNo, UInt16 MasterNo,UInt16 SlaveId,UInt16 DoGrpIndex,UInt32 Value)

功 能：设置总线节点的一组输出电平

参 数：ConnectNo 板卡连接编号[0,15]

MasterNoEtherCAT 主站编号

SlaveId 对应的总线节点号

DoGrpIndex 对应的组序号。32 点为一组，序号从 0 开始

Value 设置该组输出的实际电平

返回值：错误代码

ERROR_CODE NV_EcatGetDo(UInt16 ConnectNo, UInt16 MasterNo,UInt16 SlaveId,UInt16 DoGrpIndex,ref UInt32 pValue)

功 能：获取总线节点的一组输出电平

参 数：ConnectNo 板卡连接编号[0,15]

MasterNoEtherCAT 主站编号

SlaveId 对应的总线节点号

DoGrpIndex 对应的组序号。32 点为一组，序号从 0 开始

pValue 返回该组输出的实际电平

返回值：错误代码

5.11.4 总线对象字典操作

ERROR_CODE NV_EcatSetSDO(UInt16 ConnectNo, UInt16 MasterNo,UInt16 SlaveId,

UInt16 Index,UInt16 SubIndex,UInt16 ValLength,UInt32 Value)

功　能：设置总线节点的对象字典

参　数：ConnectNo　板卡连接编号[0,15]

MasterNoEtherCAT 主站编号

SlaveId　对应的总线节点号

Index　对象字典的索引

SubIndex　对象字典的子索引

ValLength　对象字典的长度

Value　要设置的值

返回值：错误代码

ERROR_CODE NV_EcatGetSDO(UInt16 ConnectNo, UInt16 MasterNo,UInt16 SlaveId, UInt16 Index,UInt16 SubIndex,UInt16 ValLength,ref UInt32 pValue)

功　能：获取总线节点的对象字典

参　数：ConnectNo　板卡连接编号[0,15]

MasterNoEtherCAT 主站编号

SlaveId　对应的总线节点号

Index　对象字典的索引

SubIndex　对象字典的子索引

ValLength　对象字典的长度

pValue　返回对象字典的实际值

返回值：错误代码

ERROR_CODE NV_EcatSetPDO(UInt16 ConnectNo, UInt16 MasterNo,UInt16 SlaveId, UInt16 Index,UInt16 SubIndex,UInt16 ValLength,UInt32 Value)

功　能：设置总线节点的过程数据对象字典

参　数：ConnectNo　板卡连接编号[0,15]

MasterNoEtherCAT 主站编号

SlaveId 对应的总线节点号

Index 对象字典的索引

SubIndex 对象字典的子索引

ValLength 对象字典的长度

Value 要设置的值

返回值：错误代码

ERROR_CODE NV_EcatGetPDO(UInt16 ConnectNo, UInt16 MasterNo,UInt16 SlaveId, UInt16 Index,UInt16 SubIndex,UInt16 ValLength,ref UInt32 pValue)

功 能：获取总线节点的过程数据对象字典

参 数：ConnectNo 板卡连接编号[0,15]

MasterNoEtherCAT 主站编号

SlaveId 对应的总线节点号

Index 对象字典的索引

SubIndex 对象字典的子索引

ValLength 对象字典的长度

pValue 返回对象字典的实际值

返回值：错误代码

5.12 PWM 功能

ERROR_CODE NV_PWMSetEnable(UInt16 ConnectNo, UInt16 Channel, UInt16 Enable, UInt16 DoIndex)

功 能：设置 PWM 使能。

参 数：ConnectNo 板卡连接编号[0,15]

Channel PWM 通道号：0-3

Enable 使能值：0 - 关闭； 1 - 使能

DoIndex 高速输出口：0-7

返回值：错误代码

ERROR_CODE NV_PWMGetEnable (UInt16 ConnectNo, UInt16 Channel, ref UInt16 pEnable, ref UInt16 pDoIndex)

功 能：读取 PWM 使能

参 数：ConnectNo 板卡连接编号[0,15]

Channel PWM 通道号：0-3

pEnable 返回使能值

pDoIndex 返回输出口编号

返回值：错误代码

ERROR_CODE NV_PWMSetOutput(UInt16 ConnectNo, UInt16 Channel, Double Duty, Double Fre)

功 能：设置 PWM 输出。

参 数：ConnectNo 板卡连接编号[0,15]

Channel PWM 通道号：0-3

Duty 占空比，取值范围：0~1

Fre 频率，取值范围：0-500KHz

返回值：错误代码

ERROR_CODE NV_PWMGetOutput(UInt16 ConnectNo, UInt16 Channel, ref Double pDuty, ref Double pFre)

功 能：获取 PWM 输出。

参 数：ConnectNo 板卡连接编号[0,15]

Channel PWM 通道号：0-3

pDuty 返回 PWM 占空比

pFre 返回 PWM 频率

返回值：错误代码

5.13 PVT 运动

ERROR_CODE NV_PvtAddTable(UInt16 ConnectNo,UInt16 AxisNo, Double[] pTime, Double[] pPos,Double[] pVel, UInt16 Count)

功 能：添加 PVT 列表数据

参 数：ConnectNo 板卡连接编号[0,15]

AxisNo 轴序号

pTime[50] 添加的时间数组，单位 s

pPos[50] 添加的位置数组，单位 units

pVel[50] 添加的速度数组

Count 添加的个数

返回值：错误代码

注意：下载的第一组（即起始点）数据中位置、时间必须为 0；数组中的数据都是以起始点的数据为参考点。

ERROR_CODE NV_PtsAddTable(UInt16 ConnectNo,UInt16 AxisNo, Double[] pTime,Double[] pPos, Double[] pPercent,UInt16 Count)

功 能：添加 PTS 列表数据

参 数：ConnectNo 板卡连接编号[0,15]

AxisNo 轴序号

pTime[50] 添加的时间数组，单位，s

pPos[50] 添加的位置数组

pPercent[50] 添加的百分比数组，范围：0~100；数组长度：Count

Count 添加的个数

返回值：错误代码

ERROR_CODE NV_PvtsAddTable(UInt16 ConnectNo,UInt16 AxisNo, Double[] pTime, Double[] pPos, Double StartVel, Double EndVel,UInt16 Count)

功　能：添加 PVTS 列表数据

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

pTime[50]　添加的时间数组，单位，s

pPos[50]　添加的位置数组

StartVel　添加的开始速度

EndVel　添加的结束速度

Count　添加的个数

返回值：错误代码

ERROR_CODE NV_PttAddTable(UInt16 ConnectNo,UInt16 AxisNo, Double[] pTime, Double[] pPos, UInt16 Count)

功　能：添加 PTT 列表数据

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo 轴序号

pTime[50]　添加的时间数组，单位，s

pPos[50]　添加的位置数组

Count　添加的个数。每个数据表具有 1000 个存储空间，每个数据点占用 1 个存储空间

返回值：错误代码

ERROR_CODE NV_PvtStart(UInt16 ConnectNo,UInt16 AxisNum,UInt16[] pAxisList, Double[] pDelayTime)

功　能：同时启动多轴 PVT

参　数：ConnectNo　板卡连接编号[0,15]

AxisNum　要启动 PVT 的轴数量

pAxisList　要启动 PVT 的轴列表

pDelayTime　各轴延时启动 PVT 的时间

返回值：错误代码

ERROR_CODE NV_PvtStatus(UInt16 ConnectNo, UInt16 AxisNo, ref UInt16 pIndex, ref UInt16　pTotal, ref UInt16 pMode)

功　能：获取当前 PVT 状态

参　数：ConnectNo　板卡连接编号[0,15]

AxisNo　轴序号

pIndex　返回运行的段号

pTotal　返回总运行段数

pMode　返回目前的 PVT 模式：

0：NULL

1：PVT

2：PTS

3：PVTS

4：PTT

返回值：错误代码

5.14 电子齿轮

电子齿轮参数结构体：

typedef struct GearProfileStruct

{

UInt16 MasterAxisNo; // 主轴编号

UInt16 MasterSrcType; // 主轴位置源类型

Int32 RatioNumerator; // 电子齿轮比分子

Int32 RatioDenominator; // 电子齿轮比分母

Double Acceleraton; // 加速度

Double Deceleraton; // 减速度

Double MasterSyncPosition; // 主轴同步位置，在这个位置前完成同步，[GearInPos]

Double SlaveSyncPosition; // 从轴同步位置，在这个位置前完成同步，[GearInPos]

Double MasterStartDistance;// 主轴开始距离，在这个距离间进行同步，[GearInPos]

UInt16 AvoidReversal; // 禁止反转：0-允许反转；1-不允许反转，[GearInPos]

}GearProfileStruct_t, * PGearProfileStruct_t;

函数

直接啮合的方式：

通过指定参数，进行齿轮动作；

开始动作后，Slave（从轴）以 Master（主轴）速度乘以齿轮比得到的速度为目标速度，进行加减速动作。

执行电子齿轮后要解除耦合必须通过 GearOut 指令。

位置单位：unit

ERROR_CODE NV_GearSetParam(UInt16 ConnectNo,UInt16 SlaveAxisNo, GearProfileStruct_t Profile)

功 能：设置电子齿轮的参数

参　数：ConnectNo　　板卡连接编号[0,15]

SlaveAxisNo　设置从轴编号

Profile 电子齿轮的参数

返回值：错误代码

ERROR_CODE NV_GearGetParam(UInt16 ConnectNo,UInt16 SlaveAxisNo, ref GearProfileStruct_t pProfile)

功　能：读取电子齿轮的参数

参　数：ConnectNo　　板卡连接编号[0,15]

SlaveAxisNo　　对应的轴编号

pProfile　　返回电子齿轮的参数

返回值：错误代码

ERROR_CODE NV_GearIn(UInt16 ConnectNo,UInt16 SlaveAxisNo)

功　能：进入电子齿轮模式

参　数：ConnectNo　　板卡连接编号[0,15]

SlaveAxisNo　　启动的从轴序号

返回值：错误代码

从指定位置同步的方式：

同步开始到同步结束的过程本质为同步区间内从轴跟随主轴的一个电子齿轮、此时根据主轴范围（MasterSyncPosition- MasterStartDistance，MasterSyncPosition），从轴范围（当前位置， SlaveSyncPosition），指令会根据设置齿轮比以及上述三个参数自动设计一条齿轮曲线，执行同步时从轴跟随主轴完成齿轮啮合动作。

AvoidReversal：设置为 0（FALSE），如果从轴物理位置超前的情况下进行反转。设置为 1 （TRUE）如果从轴物理上不能实现反转或者导致危险。只在模态轴下适用。 如果反转不能被避免，那么轴将错误停止。

ERROR_CODE NV_GearInPos(UInt16 ConnectNo,UInt16 SlaveAxisNo)

功　能：进入电子齿轮模式

参　数：ConnectNo　　板卡连接编号[0,15]

SlaveAxisNo　　启动的从轴序号

返回值：错误代码

ERROR_CODE NV_GearOut(UInt16 ConnectNo,UInt16 SlaveAxisNo)

功　能：退出电子齿轮模式

参　数：ConnectNo　　板卡连接编号[0,15]

SlaveAxisNo　　要退出的从轴序号

返回值：错误代码

5.15 电子凸轮

ERROR_CODE NV_CAMAddTable(UInt16 ConnectNo, UInt16 TableIndex, UInt16 Mode, Double[] pValue, UInt16 Count)

功　能：将凸轮数据添加到 Table 表

参数：ConnectNo　板卡连接编号[0,15]

TableIndex　　Table 表的编号

Mode　　添加方式：0 - 追加到末尾，1 - 清空后添加

pValue　　添加的数据

Count 添加的数据个数

返回值：错误代码

ERROR_CODE NV_CAMGetTable(UInt16 ConnectNo, UInt16 TableIndex, UInt16 DataIndex,　Double[] pValue, UInt16 Count)

功　能：读取 Table 表数据

参数：ConnectNo　板卡连接编号[0,15]

TableIndex　Table 表的编号

DataIndex　数据编号

pValue　读取的数据数组

Count 要连续读取的数据个数

返回值：错误代码

ERROR_CODE NV_CAMGetTableSize(UInt16 ConnectNo, UInt16 TableIndex, ref UInt32 pSize)

功　能：读取 Table 表中数据个数

参数：ConnectNo　板卡连接编号[0,15]

TableIndexTable 表的编号

pSize 读取 Table 表中数据个数

返回值：错误代码

ERROR_CODE NV_CAM(UInt16 ConnectNo, UInt16 SlaveAxisNo, UInt16 TableIndex, Double MoveTime)

功　能：电子凸轮运动

参数：ConnectNo　板卡连接编号[0,15]

SlaveAxisNo 从轴号（需要运行 Table 表数据的轴号）

TableIndex Table 表的编号

MoveTime　运动时间，单位：s

返回值：错误代码

ERROR_CODE NV_CAMBox(UInt16 ConnectNo, UInt16 SlaveAxisNo, UInt16 TableIndex, UInt16 LinkAxisNo, UInt16 LinkSrcType, UInt16 LinkOptions, UInt32

LinkDiIndex, UInt32 LinkDiLogic, Double LinkStartPos, Double LinkDistance)

功　能：电子凸轮运动，CAMBOX 指令根据存储在 TABLE 中的数据来决定轴的运动。 这些 Table 数据值对应运动轨迹的位置，是相对于运动起始点的距离。跟随轴的运动根据参考轴的运动来进行。

参数：ConnectNo　板卡连接编号[0,15]

SlaveAxisNo 从轴号（需要运行 Table 表数据的轴号）

TableIndex Table 表的编号

LinkAxisNo 主轴号（被连接对象）

LinkSrcType 主轴位置源：1 - 指令位置；2 - 反馈位置

LinkOptions 主从轴连接方式，根据 Bit 位配置

LinkDiIndex 触发输入编号：普通输入 IN0-7

LinkDiLogic 触发输入逻辑：0-下降沿，1-上升沿，2-任意沿

LinkStartPos　当 LinkOptions 参数 Bit1 置为 1 时，该参数表示连接开始启动的主轴绝对位置。　单位：units

LinkDistance　连接后跟随主轴运动的距离

返回值：错误代码

说明：

（1）运动的总时间由主轴的设置速度和 LinkDistance 参数决定，从轴的实际速度根据 Table 轨迹与时间自动匹配。

（2）LinkOptions 的 Bit 位置位后，表示的意义：

Bit 0 当输入信号事件触发时，主从轴开始连接运动。

Bit 1 当主轴运动到设定的绝对位置时，主从轴开始连接运动。

Bit 2 自动重复连续双向运行。

Bit 5 只有主轴正向运动才连接。

ERROR_CODE NV_CAMInAddTable(UInt16 ConnectNo, UInt16 TableIndex, UInt16 Mode, Double[] pMasterPos, Double[] pSlavePos, UInt16 Count)

功　能：将凸轮主从轴数据添加到 Table 表

参数：ConnectNo　板卡连接编号[0,15]

TableIndex　　Table 表的编号

Mode 添加方式：0 - 追加到末尾，1 - 清空后添加

pMasterPos 添加主轴位置数组

pSlavePos 添加从轴位置数组

Count 添加的数据个数

返回值：错误代码

ERROR_CODE NV_CAMIn(UInt16 ConnectNo, UInt16 SlaveAxisNo, UInt16 TableIndex, UInt16 LinkAxisNo, UInt16 LinkSrcType, UInt32 LinkOptions, UInt32 LinkDiIndex, UInt32 LinkDiLogic, Double LinkStartPos)

功　能：电子凸轮运动，CAMIn 指令根据存储在 TABLE 中的数据来决定轴的运动。这些 Table 数据值对应运动轨迹的位置，是相对于运动起始点的距离。跟随轴的运动根据参考轴的运动来进行。

参数：ConnectNo　板卡连接编号[0,15]

SlaveAxisNo 从轴号（需要运行 Table 表数据的轴号）

TableIndex Table 表的编号

LinkAxisNo 主轴号（被连接对象）

LinkSrcType 主轴位置源：1 - 指令位置；2 - 反馈位置

LinkOptions 主从轴连接方式，根据 Bit 位配置

LinkDiIndex 触发输入编号：普通输入 IN0-7

LinkDiLogic 触发输入逻辑：0-下降沿，1-上升沿，2-任意沿

LinkStartPos　当 LinkOptions 参数 Bit1 置为 1 时，该参数表示连接开始启动的主轴绝对位置。单位：units

返回值：错误代码

说明： LinkOptions 的 Bit 位置位后，表示的意义：

Bit 0 当输入信号事件触发时，主从轴开始连接运动。

Bit 1 当主轴运动到设定的绝对位置时，主从轴开始连接运动。

Bit 2 自动重复连续双向运行。

Bit 5 只有主轴正向运动才连接。

ERROR_CODE NV_CAMOut(UInt16 ConnectNo, UInt16 SlaveAxisNo)

功　能：退出电子凸轮运动

参数：ConnectNo　板卡连接编号[0,15]

SlaveAxisNo 要退出的从轴序号

返回值：错误代码

5.16 龙门功能

ERROR_CODE NV_GantryIn (UInt16 ConnectNo, UInt16 SlaveAxisNo)

功　能：设置龙门的参数

参　数：ConnectNo　　板卡连接编号[0,15]

SlaveAxisNo 从轴轴号

返回值：错误代码

ERROR_CODE NV_GantryGetParam (UInt16 ConnectNo,UInt16 SlaveAxisNo, ref UInt16 pMasterAxisNo, ref UInt16 pMasterSrcType, ref UInt16 pDirection)

功　能：设置龙门的参数

参　数：ConnectNo　　板卡连接编号[0,15]

SlaveAxisNo　　从轴轴号

pMasterAxisNo　返回主轴轴号

pMasterSrcType　返回主轴位置源

pDirection　　返回跟随方向

返回值：错误代码

ERROR_CODE NV_GantryOut (UInt16 ConnectNo,UInt16 SlaveAxisNo)

功　能：退出龙门模式

参　数：ConnectNo　　板卡连接编号[0,15]

SlaveAxisNo　　要退出的从轴序号

返回值：错误代码

ERROR_CODE NV_GantrySetProtect(UInt16 ConnectNo,UInt16 SlaveAxisNo,UInt16 Enable, Double DecStopError,Double ImdStopError, UInt16 ErrorSrcType)

功　能：设置龙门超差保护参数

参　数：ConnectNo　　板卡连接编号[0,15]

SlaveAxisNo　轴序号

Enable　　启用误差保护功能

DecStopError　触发减速停止时的误差值

ImdStopError　触发立即停止时的误差值

ErrorSrcType　误差比较来源：1 - 指令位置；2 - 反馈位置

返回值：错误代码

ERROR_CODE NV_GantryGetProtect(UInt16 ConnectNo,UInt16 SlaveAxisNo,ref UInt16 pEnable, ref Double pDecStopError,ref Double pHardError, ref UInt16 pErrorSrcType)

功　能：读取设置龙门超差保护参数

参　数：ConnectNo　　板卡连接编号[0,15]

SlaveAxisNo　轴序号

pEnable　返回启用误差保护功能状态

pDecStopError 返回触发减速停止时的误差值

pImdStopError 返回触发立即停止时的误差值

pErrorSrcType 返回误差比较来源

返回值：错误代码

5.17 看门狗功能

ERROR_CODE NV_WatchdogSetEnable(UInt16 ConnectNo,UInt16 Channel, UInt16 Enabled, Double FeedTime)

功 能：使能看门狗

参 数：ConnectNo 板卡连接编号[0,15]

Channel 对应的看门狗序号

Enabled 是否启用看门狗

FeedTime 喂狗时间

返回值：错误代码

ERROR_CODE NV_WatchdogGetEnable(UInt16 ConnectNo,UInt16 Channel, ref UInt16 pEnabled, ref Double pFeedTime)

功 能：获取看门狗使能配置

参 数：ConnectNo 板卡连接编号[0,15]

Channel 对应的看门狗序号

pEnabled 返回是否启用看门狗

pFeedTime 返回喂狗时间

返回值：错误代码

ERROR_CODE NV_WatchdogSetAction(UInt16 ConnectNo,UInt16 Channel, Int32[] pAction,Int32[] pActParam1, UInt32[] pActParam2, UInt16 ActionNum)

功 能：看门狗功能设置

参　数：ConnectNo　板卡连接编号[0,15]

Channel　对应的看门狗序号

pAction[8]　触发动作类型，最多 8 个

pActParam1[8]　触发动作参数 1 (IO 编号或者轴号，设置值为-1 时操作所有 IO　或轴)

pActParam2[8]　触发动作参数 2 (掩码值每 1 位代表 1 个 IO 口或 1 个轴）

ActionNum　触发动作数量

返回值：错误代码

备注：32 个 IO 一组或者 32 个轴一组，从 0 开始编号，掩码值的每 1 个位代表 1 个 IO 或者 1 个轴，位值 1 表示启用。

函数 Action, ActParam 参数值

Action	功能	ActParam1	ActParam2
1	IO 置为高电平	IO 组号	IO 掩码值
2	IO 置为低电平	IO 组号	IO 掩码值
3	取反 IO	IO 组号	IO 掩码值
10	停止指定轴	轴组号	轴掩码值

ERROR_CODE NV_WatchdogGetAction(UInt16 ConnectNo,UInt16 Channel, Int32[] pAction,Int32[] pActParam1, UInt32[] pActParam2, ref UInt16 pActionNum)

功　能：读取看门狗功能设置

参　数：ConnectNo　板卡连接编号[0,15]

Channel　对应的看门狗序号

pAction[8]　返回触发动作

pActParam1[8]　返回动作参数 1

pActParam2[8]　返回动作参数 2

pActionNum　返回动作数量

ERROR_CODE NV_WatchdogFeedDog(UInt16 ConnectNo,UInt16 Channel)

功　能：看门狗喂狗

参　数：ConnectNo　　板卡连接编号[0,15]

　　Channel　　对应的看门狗序号，最大数量 4 个

返回值：错误代码

5.18 密码管理

ERROR_CODE NV_PasswordLogin(UInt16 ConnectNo, string PassWord)

功　能：密码登录

参　数：ConnectNo 板卡连接编号[0,15]

　　PassWord 登录密码，长度不大于 255 个字符

返回值：错误代码

ERROR_CODE NV_PasswordModify(UInt16 ConnectNo, string PassWord)

功　能：密码修改，需要先登入

参　数：ConnectNo 板卡连接编号[0,15]

　　PassWord 新密码，长度不大于 255 个字符

返回值：错误代码

ERROR_CODE NV_PasswordCheck(UInt16 ConnectNo, string PassWord)

功　能：密码校验

参　数：ConnectNo　　板卡连接编号[0,15]

　　PassWord 校验密码，长度不大于 255 个字符

返回值：错误代码

注 意：1）用户可以在系统软件开启时加入密码校验动作，以此对系统软件进行加密

2）密码校验失败 5 次后，无法进行校验；若需再次校验，需将电脑关机，断电重启

附件

F1 错误码

当调用的函数出现语法错误或者参数范围错误等情况时，函数会返回错误码，详见下表。

错误编号	错误描述
无错误	
0	
轴错误码	
2001	序号超过最大值
2002	轴编码器序号超过最大值
2003	独立编码器序号超过最大值
2004	编码器当量无效
2005	没有该引脚，范围已经超过最大值
2006	没有该类型，范围已经超过最大值
2007	轴类型切换禁止
2008	轴节点地址无效
2009	轴类型为 ECAT 时禁止切换
比较器错误码	
3001	比较类型序号超过最大值
3002	序号超过最大值
3003	没有该引脚，范围已经超过最大值
3004	引脚已经被使用
3005	模式错误
3006	正在添加数据
3007	添加数据超过最大值
编码器错误码	
4001	序号超过最大值
4002	轴编码器序号超过最大值

4003	编码器分频无效
4004	编码器模式无效
高速比较器错误码	
5001	序号超过最大值
5002	锁存类型序号超过最大值
5003	没有该引脚，范围已经超过最大值
5004	模式错误
5005	正在添加数据
5006	添加数据超过最大值
5007	引脚已经被使用
高速锁存错误码	
6001	序号超过最大值
6002	锁存类型序号超过最大值
6003	没有该引脚，范围已经超过最大值
6004	模式错误
软件比较错误码	
6101	序号超过最大值
6102	锁存类型序号超过最大值
6103	没有该引脚，范围已经超过最大值
6104	锁存类型错误
IO 错误码	
7001	输出序号超过最大值
7002	输入序号超过最大值
7003	高速输出序号超过最大值
7004	高速输入序号超过最大值
7005	输出序号已经在 pwm 中使用，不能当作普通输出
7006	输出序号已经在 reverse 中使用，不能当作普通输出
7007	输出序号已经在 1 维比较中使用，不能当作普通输出
7008	输出序号已经在 2 维比较中使用，不能当作普通输出

7009	pwm 序号超过最大值
7010	reverse 序号超过最大值
7011	计数错误
7012	反转输出范围错误
PWM 错误码	
9001	序号超过最大值
9002	没有该引脚，范围已经超过最大值
9003	引脚已经被使用
系统错误码	
10001	无效的文件访问
10002	内存分配错误
10003	CRC 校验错误
10004	无效的文件类型
10005	无效的数据包索引
10006	无效的密码文件访问
10007	无效的密码
10008	权限不足
10009	失败次数超过最大值
10010	密码校验失败
10011	看门狗序号超过最大值
10012	时间参数错误
10013	时间错误
10014	设置系统时间失败
10015	设置硬件时钟失败
10016	获取硬件时钟失败
10017	文件包数量超过最大值
10018	文件包大小超过最大值
10019	文件包数据超过最大值
10020	文件包数据无效

10021	文件包数据校验和无效
10022	文件包数据 CRC 校验和无效
日志错误码	
10101	无效的实例
10102	缓冲区太小
10103	无效的参数
10104	格式错误
总线错误码	
11001	超过最大节点数
11002	没有该节点
11003	没有该轴
11004	没有该输入
11005	没有该输出
11006	没有该 ADC
11007	没有该 DAC
11008	没有该 SYNC
11009	没有 ENI 文件
11010	轴不是 ECAT 轴
11011	总线模式错误
11012	没有 ECAT
11013	没有该 PDO 入口
11014	PDO 不能设置
11015	总线模式错误
11016	总线错误
高速二维比较错误码	
12001	序号超过最大值
12002	锁存类型序号超过最大值
12003	没有该引脚，范围已经超过最大值
12004	模式错误

12005	正在添加数据
12006	添加数据超过最大值
模拟量错误码	
13001	序号超过最大值
13002	序号超过最大值
轴状态错误码	
14001	
PVT 错误码	
15001	轴号或点数超出范围
15002	PT 点索引超出范围
15003	PT 点数超出范围
15004	时间等于零
15005	速度/加速度/减速度小于等于零
15006	首段数据不等于零
15007	PVT 点索引超出范围
15008	PVT 点数超出范围
15009	轴数超出范围
15010	点数等于零
15011	位置与上一点位置相同
15012	方向与速度不匹配
15013	速度小于零
15014	PVT 模式已经激活
空指针错误	
15100	空指针错误
15101	内存分配失败
通用控制错误码	
15200	不支持该功能
15200	轴正在运动
15200	轴停止动作错误

JOG 错误码	
15301	JOG IO 索引错误
15302	JOG IO 配置重复
15303	在 JOG 硬件模式下
15304	轴正在 JOG 模式
轴错误码	
15401	轴正在运动
15402	轴使能关闭
15403	轴电源关闭
15404	轴正软限位触发
15405	轴负软限位触发
15406	运动方向参数错误
叠加运动错误码	
15501	叠加轴相同
15502	叠加轴相互叠加
圆弧限位错误码	
15601	圆形限位半径错误
15602	圆形限位指针错误
电子齿轮错误码	
15701	电子齿轮正在运行
15702	电子齿轮参数错误
15703	电子齿轮从轴与主轴号相同
电子凸轮错误码	
15801	无效的凸轮参数
15802	凸轮表已满
15803	凸轮表为空
15804	凸轮未运行
15805	凸轮表正在使用中
15806	凸轮表索引超出范围

15807	凸轮表超过最大数量
手轮错误码	
16000	轴不是手轮通道
16001	轴不在手轮模式
16002	手轮倍率为零
16003	轴处于手轮运动中，禁止编码器清零
16004	驱动器报警信号有效
16005	紧急停止信号有效
16006	轴未使能
16007	已经处于手轮模式
16008	总线配置状态
16009	轴号超出范围
16010	轴选超出范围
16011	倍选超出范围
16012	设置手轮编码器值为非零值
16013	设置手轮轴号重复
螺距补偿错误码	
16100	丝杠补偿操作成功
16101	丝杠补偿方向无效
16102	丝杠补偿使能状态无效
16103	丝杠补偿表大小无效
16104	丝杠补偿长度无效
16105	丝杠补偿指针为空
16106	丝杠补偿内部错误
单轴错误码	
16201	轴紧急停止信号有效
16202	轴正限位触发
16203	轴负限位触发
16204	轴停止动作错误

16205	轴驱动器停止信号有效
16206	轴正在回零
16207	倍率无效；JOG IO 配置重复
参数错误码	
16300	参数错误
16301	轴未就绪
16302	轴正在运动
16303	轴已经到达终点，不能进行相对变位
回零错误码	
16400	回零加速度为零
16401	回零搜索速度为零
16402	回零寻找速度为零
16403	回零方法错误
16404	同时触发正负限位
16405	触发正限位
16406	同时触发负限位与原点
16407	触发负限位
16408	同时触发正限位与原点
16409	回零状态与模式错误
16410	负限位与原点太近
16411	正限位与原点太近
16412	负限位与原点重叠
16413	正限位与原点重叠
16414	正负限位之间没有原点
16415	负限位与 Z 相太近
16416	正限位与 Z 相太近
16417	锁存过程中触发负限位
16418	锁存过程中触发正限位
16419	回零起始速度为负

16420	Z 相数量错误
16421	轴不是 EtherCAT 类型
缓冲区错误码	
15800	缓冲区正在运行
15801	缓冲区未运行
15802	缓冲区未暂停
15803	缓冲区为空
15804	缓冲区已满
15805	缓冲区类型无效
15806	缓冲区参数无效
龙门错误码	
17000	从轴已经被其他主轴使用
17001	主轴和从轴相同
17002	龙门模式已经启用
17003	龙门参数未配置
17004	龙门模式未启用
17005	无效的误差参数
数据采集错误码	
17100	采集通道超出范围
17101	无效的采集源类型
17102	采集模块未就绪
17103	采集内存分配错误
17104	无效的轴数量
17105	无效的 IO 数量
17106	空指针错误
前瞻错误码	
18000	list 号错误
18001	list 不在打开状态
18002	list 已经关闭

18003	参数不在有效范围
18004	list 已经打开
18005	主要的 list 没有初始化
18006	轴数不在有效范围
18007	轴映射表为空
18008	映射轴错误
18009	映射轴忙
18010	运动中不运行更改参数
18011	缓冲区已满
18012	缓冲区为空
18013	段被激活
18014	索引错误
18015	圆弧超出范围
18016	list 不能暂停
18017	半径为 0 或小于两点的距离的一半
18018	另外一个 list 已经启动
18019	半径方式起点和终点重回
18020	轴的使能信号关闭，不能启动
18021	起跳速度小于零
18022	最大速度小于等于零
18023	终点速度小于零
18024	规划长度小于等于零
18025	IO 数量超出范围
18026	固件不支持
18027	空指针错误
18028	空指针错误 1
18029	空指针错误
18030	插补维度超出范围
18031	轴索引在列表中重复

18032	轴不在打开的列表中
18033	参数错误
18034	插补维度超出范围
18035	轴不在打开的列表中
18036	圆弧轴不在 XYZ 平面内
动态库层错误码	
30000	连接驱动加载失败
30001	连接驱动未加载
30002	连接驱动扫描失败
30003	连接驱动没找到卡
30004	连接驱动开卡失败
30005	连接没打开
30006	链接号已经打开或被占用
30007	连接号超限
30008	连接号重复
30009	存在重复开卡错误
30010	连接类型错误
30011	检测到连续通信超时主动关闭连接
30012	连接无效
30013	连接 PCI 开卡失败
30014	连接 TCP 开卡失败
30015	连接 VIR 开卡失败
30016	TCP 扫描失败
30100	通讯链接号超限
30101	通讯等待信号量超时
30102	通讯应答超时
30103	通讯标签不匹配
30104	通讯 CRC 校验码不匹配
30105	接收到数据长度不够

30106	//接收到数据长度太长
30107	//标签没有找到
30180	卡端通讯接收到不支持的命令
30181	卡端通讯接收到 CRC 校验不匹配
30182	卡端通讯接收到数据长度不够
30183	卡端通讯接收到数据长度太长
30190	TCP 发送失败
30191	TCP 通道被服务端主动关闭或者通信异常断开
30200	功能不支持
30201	模式不匹配，不允许操作当前函数
30300	参数标签不存在
30301	参数当中存在给入空指针
30302	参数无效，参数值不在有效范围内
30303	参数 Count 个数不在有效范围
31000	//文件打开失败
31001	文件大小错误
31002	文件缓存大小不够
31003	文件 CRC32 校验不匹配

F2 常用 PDO 对象表

下表为 PDO 映射对象常用的对象字典：

主索引（HEX）	子索引（HEX）	对象名称	数据类型	PDO 映射
603F	0	错误码(Error Code)	UINT16	T_PDO
6040	0	控制字(Controlword)	UINT16	R_PDO
6041	0	状态字(Statuword)	UINT16	T_PDO
6060	0	操作模式(Modes of operation)	INT8	T_PDO
6061	0	操作模式显示(modes of	INT8	R_PDO

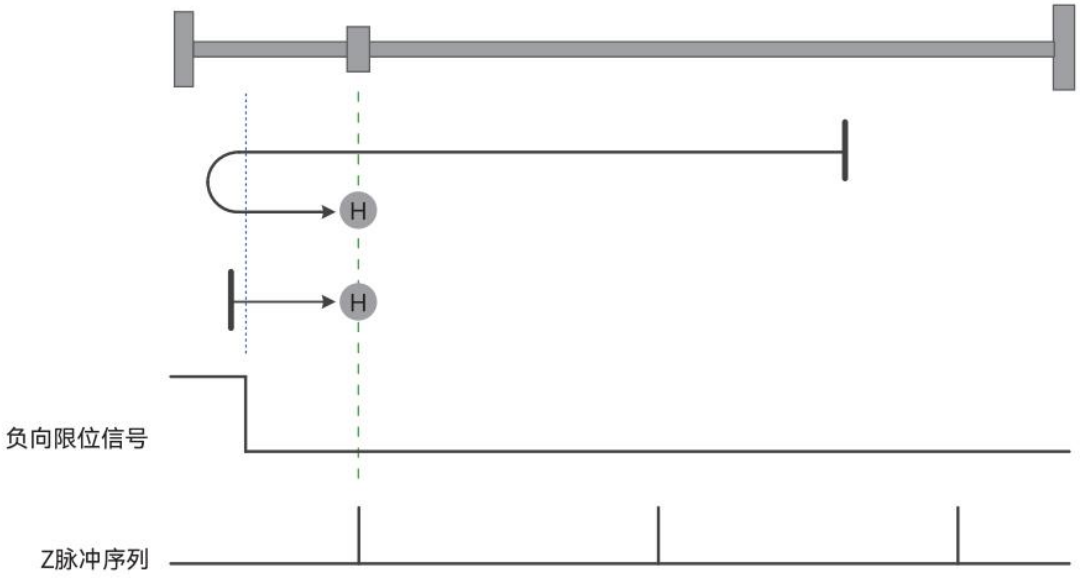
		operation display)		
6064	0	实际位置(position actual value)	INT32	T_PDO
606C	0	速度实际值 (Velocity actual value)	INT32	T_PDO
607A	0	目标位置(Target position)	INT32	R_PDO
60FD	0	数字输入(Digital inputs)	UINT32	T_PDO
60FE	1	物理输出开启(Physical outputs)	UINT32	R_PDO
60FE	2	物理输出使能(Bit mask)	UINT32	R_PDO
607C	0	原点偏移(Home offset)	INT32	NO
6098	0	回零方式(Homing method)	INT8	NO
6099	1	回零高速(Speed during search for switch)	UINT32	NO
6099	2	回零低速(Speed during search for zero)	UINT32	NO
609A	0	回零加速度(Homing acceleration)	UINT32	NO
6092	1	每转脉冲数(Feed)	UINT32	NO
6071	0	目标转矩(Target touque)	INT16	R_PDO
6072	0	最大转矩(Max touque)	UINT16	R_PDO
6077	0	转矩实际值(Touque actual value)	INT16	T_PDO
6080	0	最大电机速度(Max motor speed)	UINT32	R_PDO
6087	0	转矩变化率(Touque slope)	UINT32	R_PDO
60B2	0	转矩偏移值(Touque offset)	INT16	R_PDO

F3 回零模式

模式 1：寻找负限位和 Z 脉冲

机械原点：电机 Z 信号

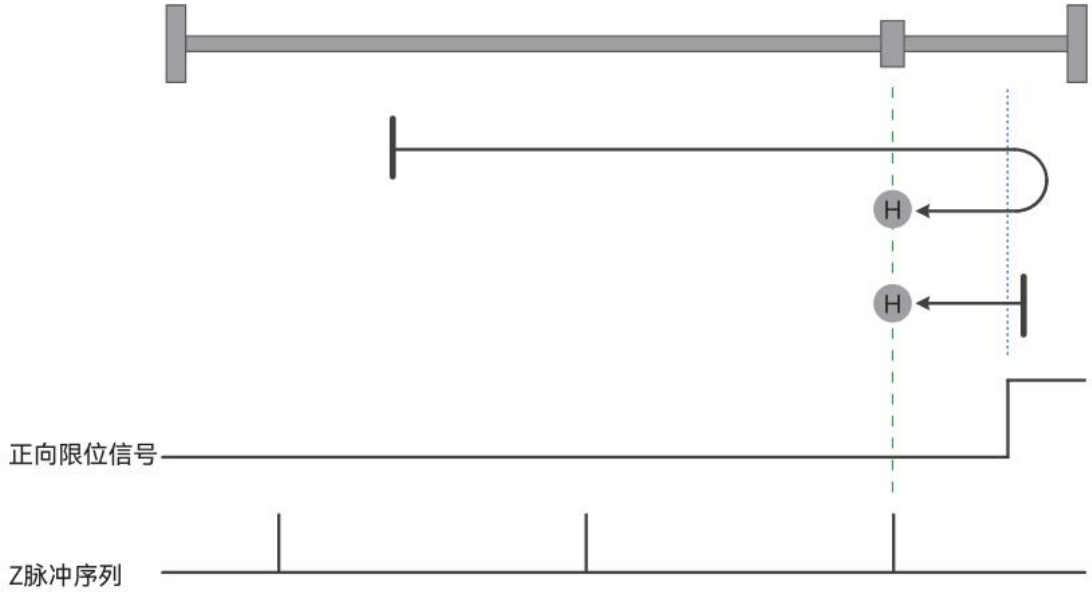
减速点：负向限位开关



模式 2：寻找正限位和 Z 脉冲

原点：Z 信号

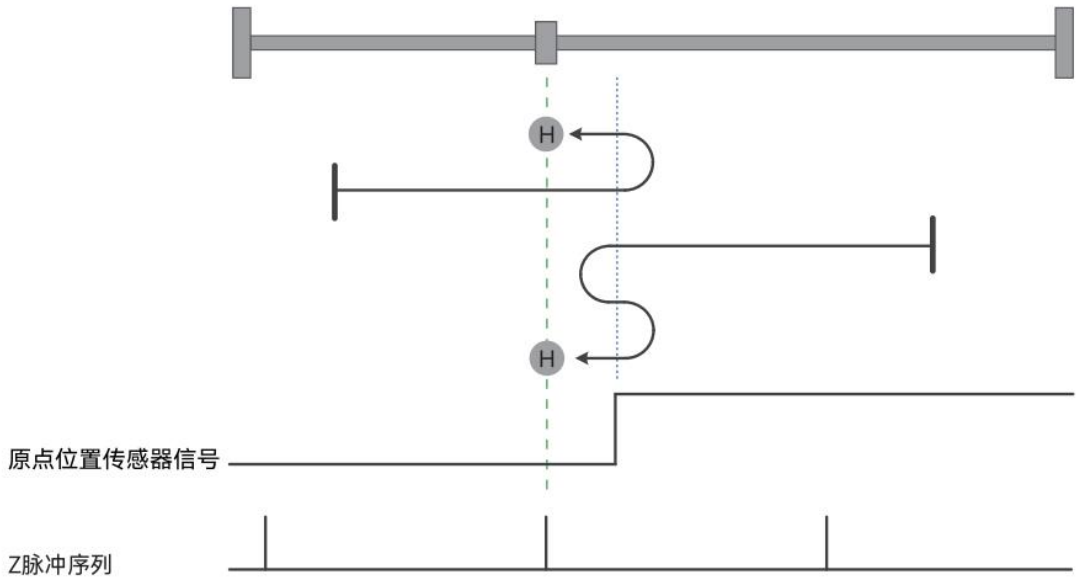
减速点：正向限位开关



模式 3：负向寻找原点开关和 Z 脉冲

原点：Z 信号

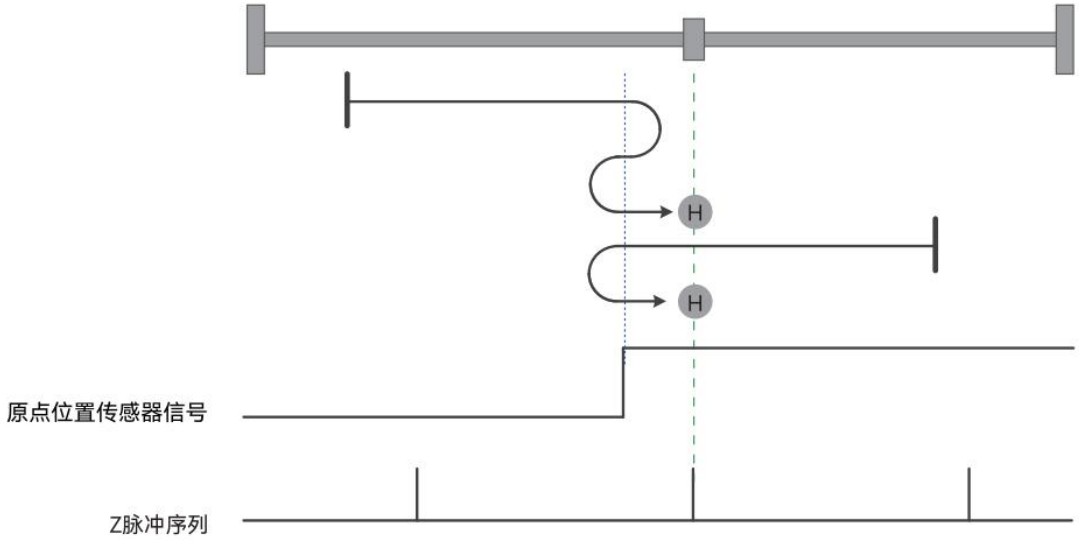
减速点：原点开关(HW)



模式 4：正向寻找原点开关和 Z 脉冲

原点：Z 信号

减速点：原点开关(HW)



模式 5：正向寻找原点开关和 Z 脉冲

原点：Z 信号

减速点：原点开关(HW)

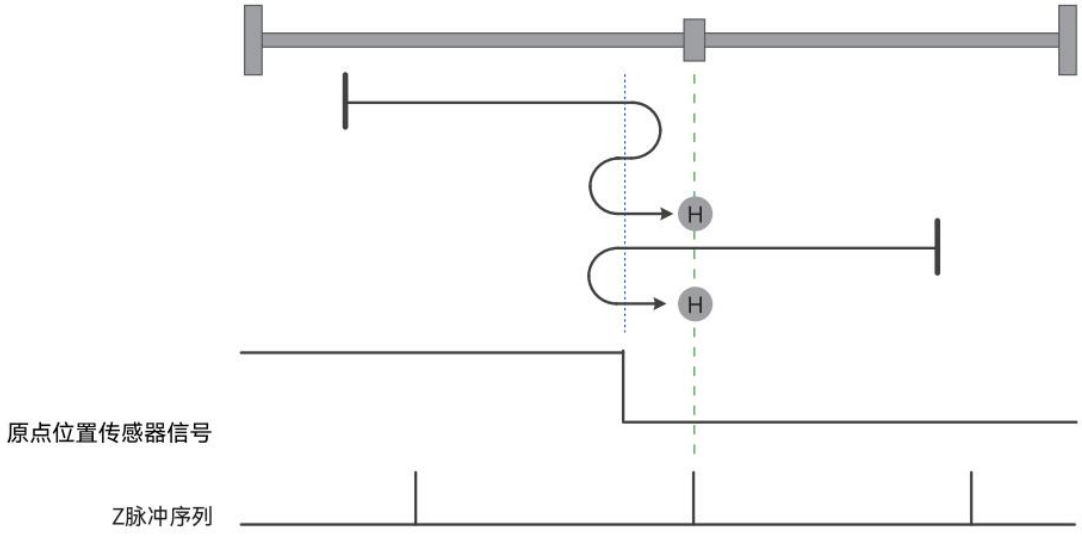
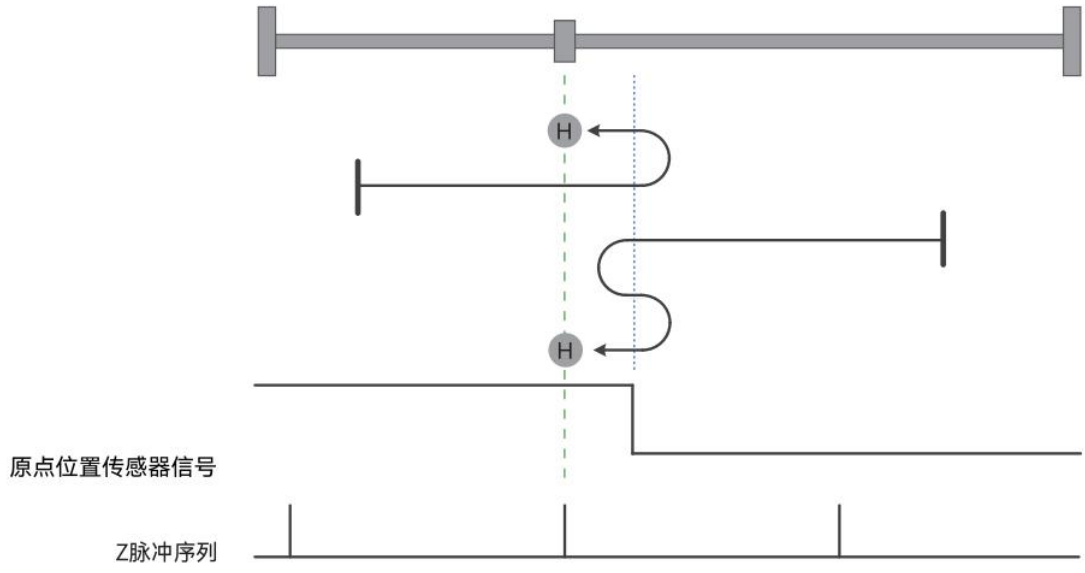


图 模式 5 原点回零电机运行曲线与转速说明

模式 6：负向寻找原点开关和 Z 脉冲

原点：Z 信号

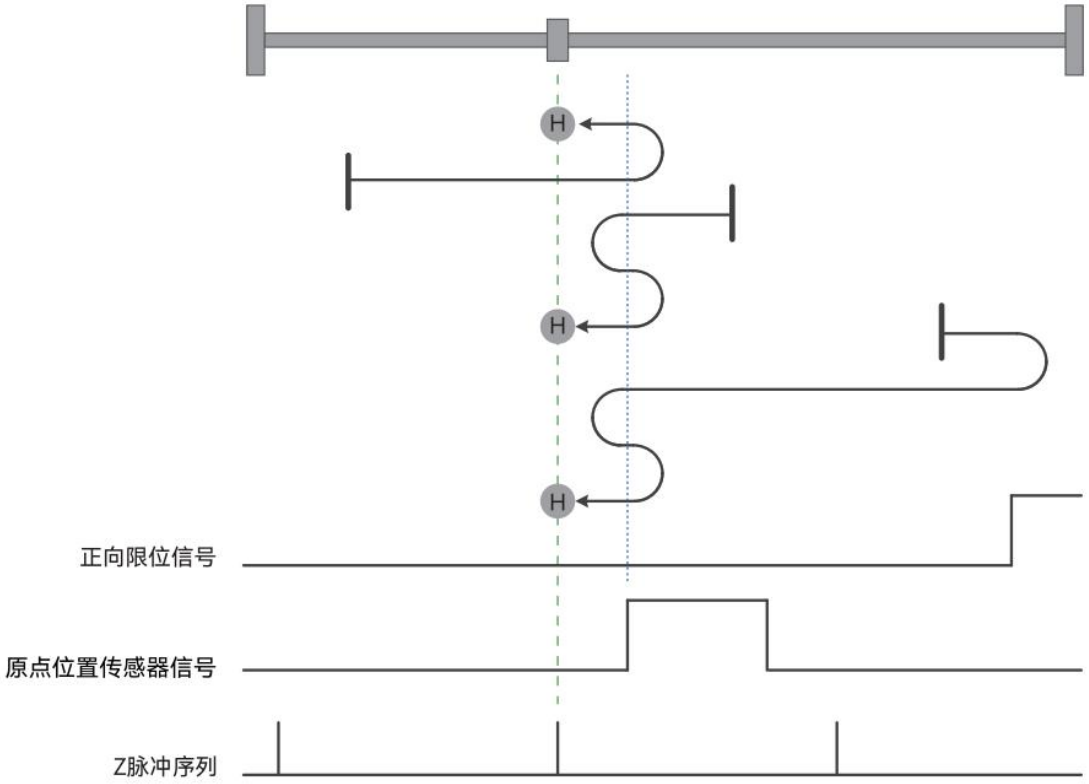
减速点：原点开关(HW)



模式 7：负向寻找原点开关 ON→OFF 位置和 Z 脉冲，正限位自动反向

原点：Z 信号

减速点：原点开关(HW)



模式 8：正向寻找原点开关 OFF→ON 位置和 Z 脉冲，正限位自动反向

原点：Z 信号

减速点：原点开关(HW)

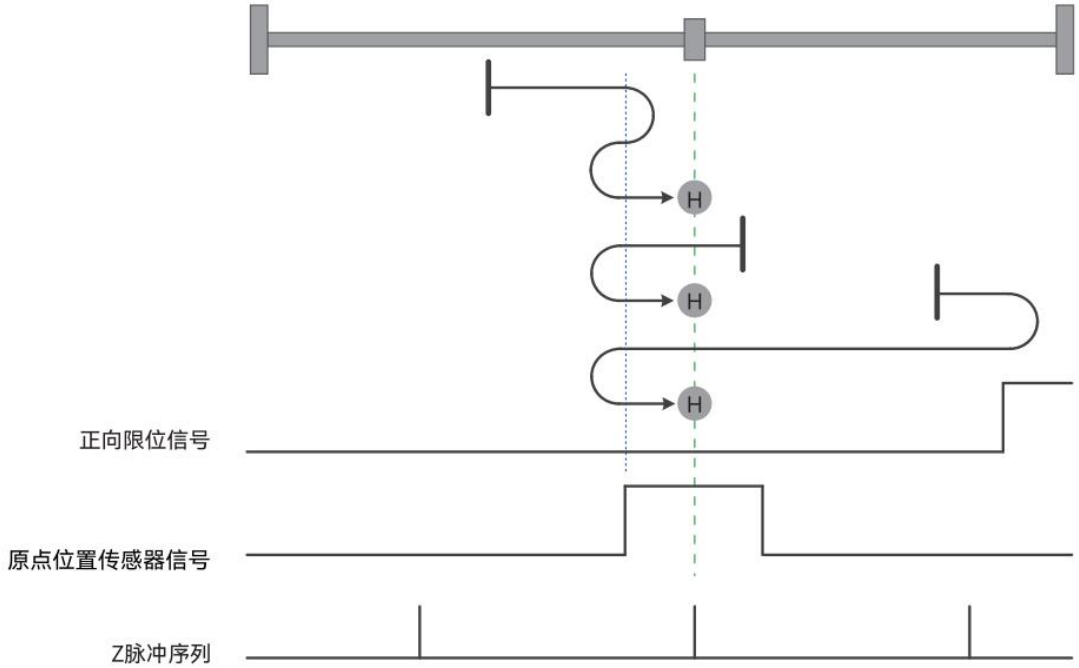


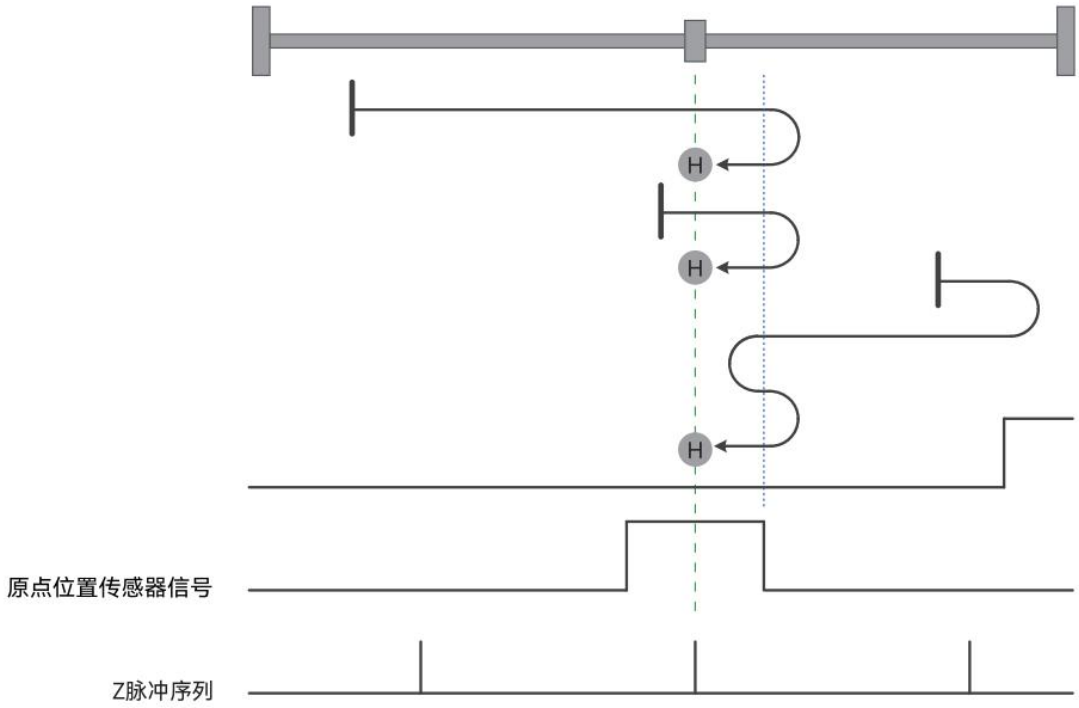
图 模式 8 原点回零电机运行曲线与转速说明

- 运动轨迹 1：回零启动时减速点信号无效，未遇到正向限位开关。
- 运动轨迹 2：回零启动时减速点信号无效，遇到正向限位开关。
- 运动轨迹 3：回零启动时减速点信号有效。

模式 9：负向寻找原点开关 OFF→ON 位置和 Z 脉冲，正限位自动反向

原点：Z 信号

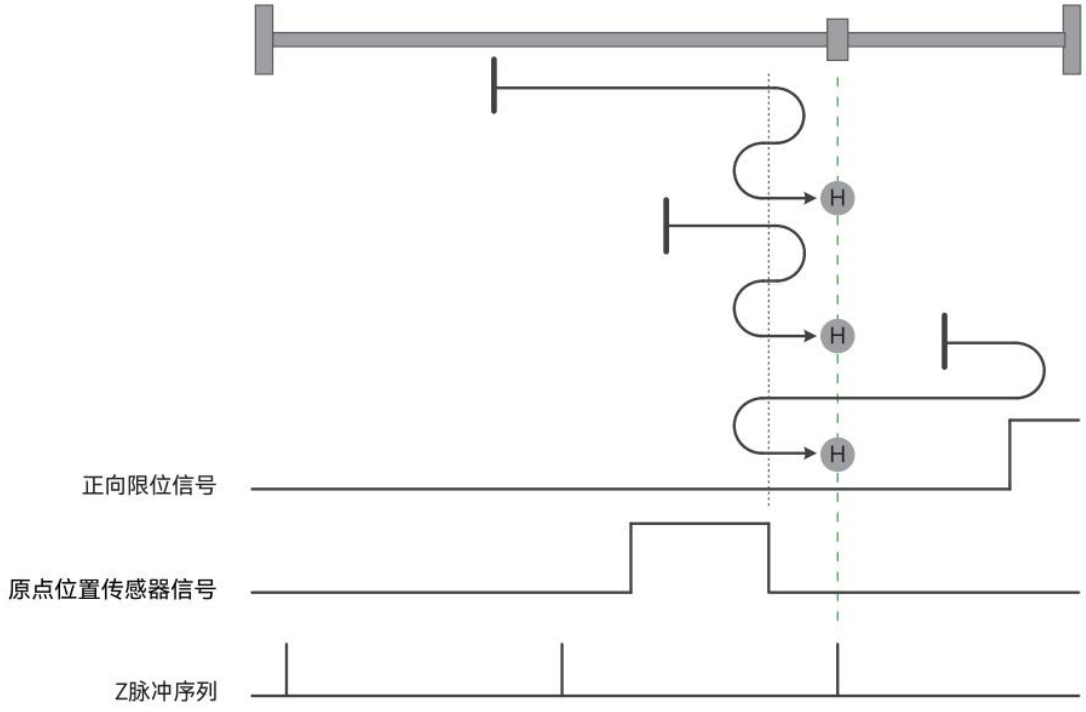
减速点：原点开关(HW)



模式 10：正向寻找原点开关 ON→OFF 位置和 Z 脉冲，正限位自动反向

原点：Z 信号

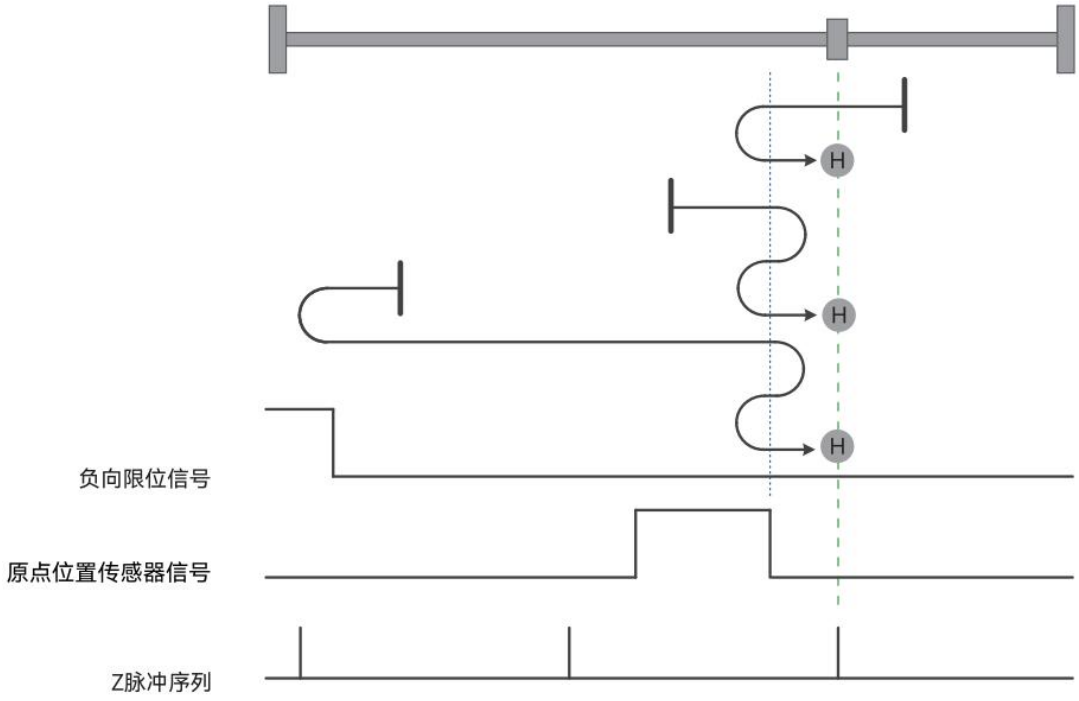
减速点：原点开关(HW)



模式 11：正向寻找原点开关 ON→OFF 位置和 Z 脉冲，负限位自动反向

原点：Z 信号

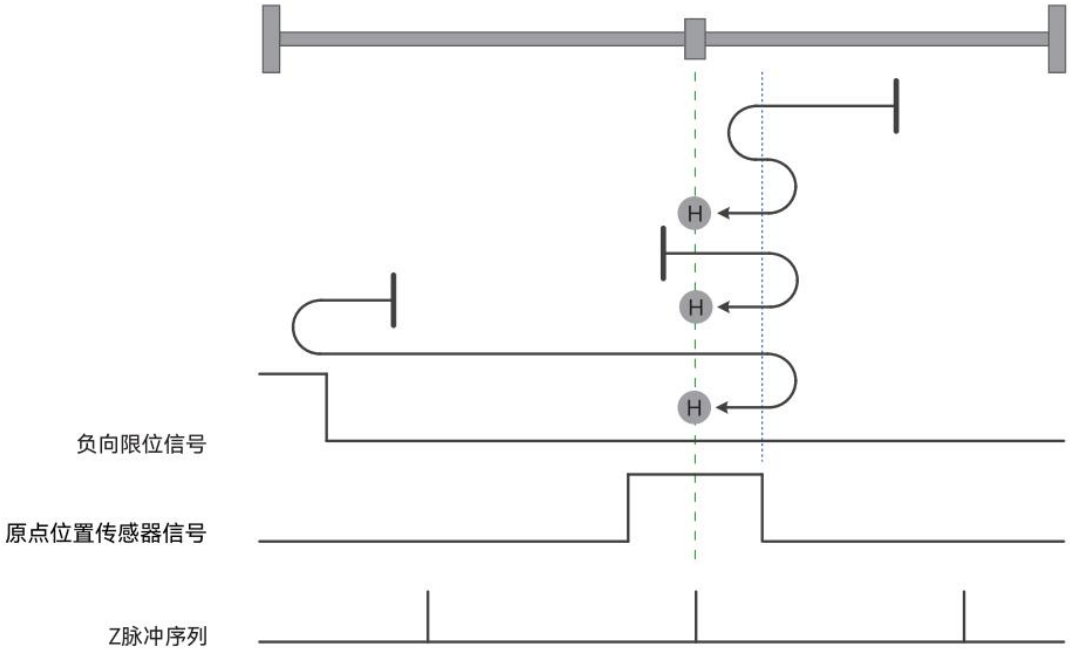
减速点：原点开关(HW)



模式 12：负向寻找原点开关 OFF→ON 位置和 Z 脉冲，负限位自动反向

原点：Z 信号

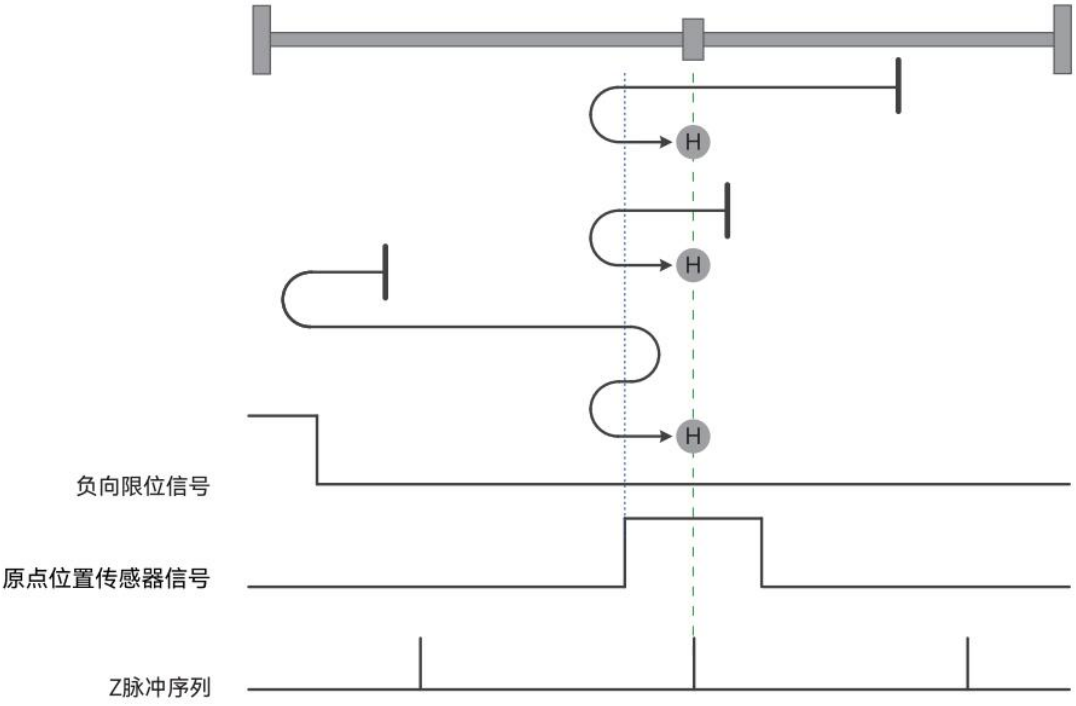
减速点：原点开关(HW)



模式 13：负向寻找原点开关 OFF→ON 位置和 Z 脉冲，负限位自动反向

原点：Z 信号

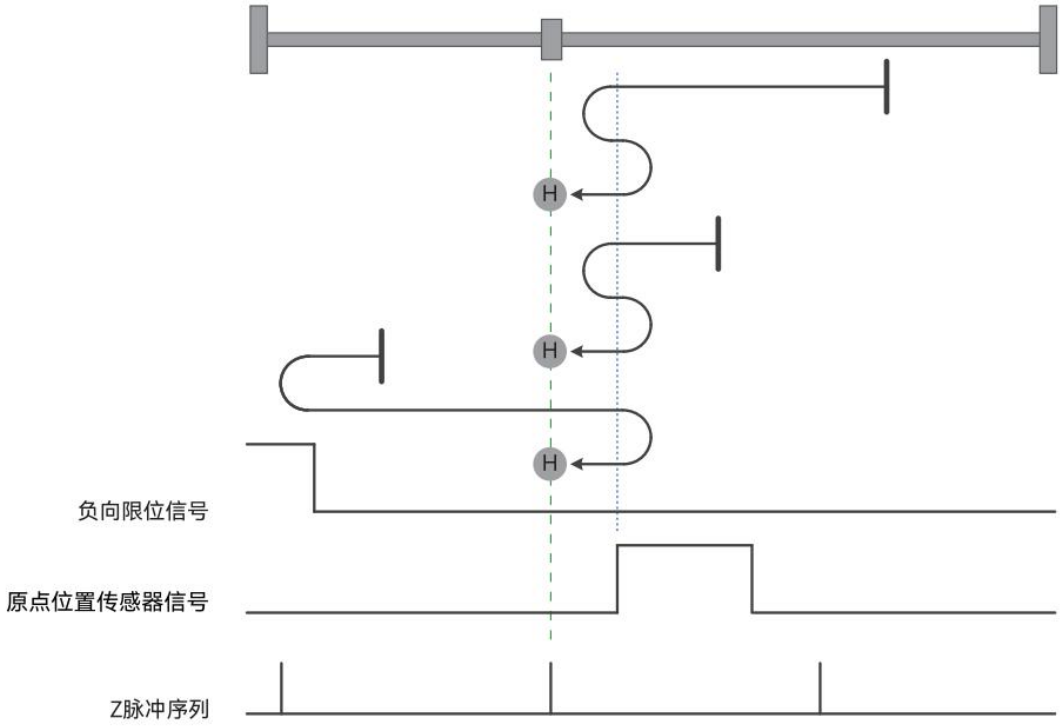
减速点：原点开关(HW)



模式 14：负向寻找原点开关 ON→OFF 位置和 Z 脉冲，负限位自动反向

原点：Z 信号

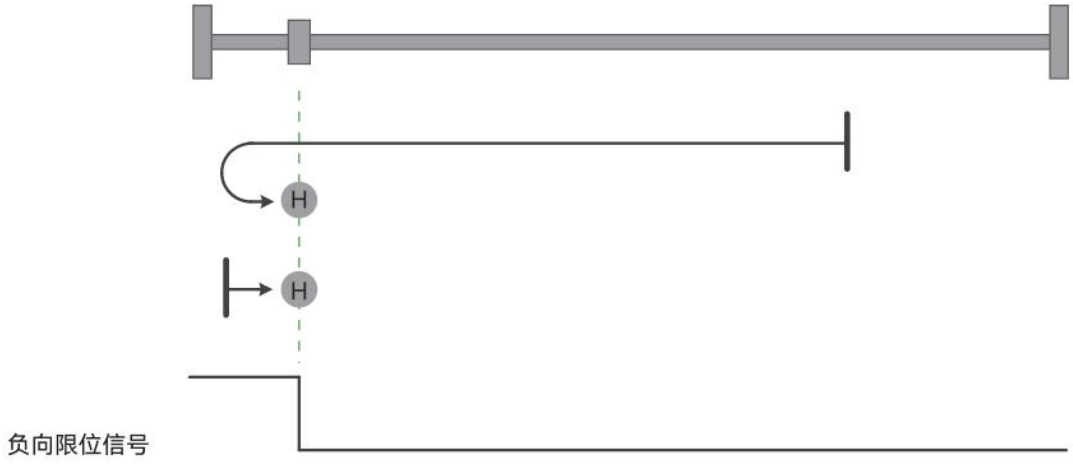
减速点：原点开关(HW)



模式 17：寻找负限位

机械原点：负向限位开关

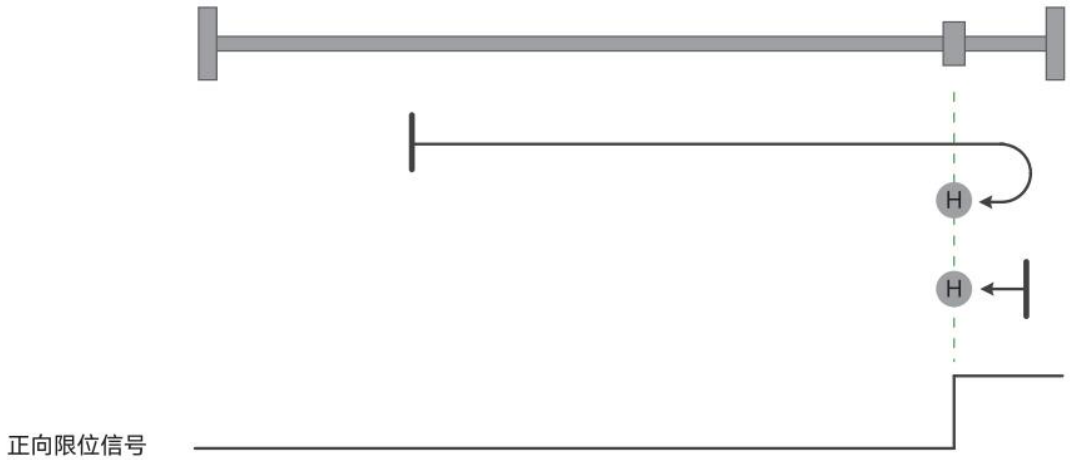
减速点：负向限位开关



模式 18：寻找正限位

原点：正向限位开关

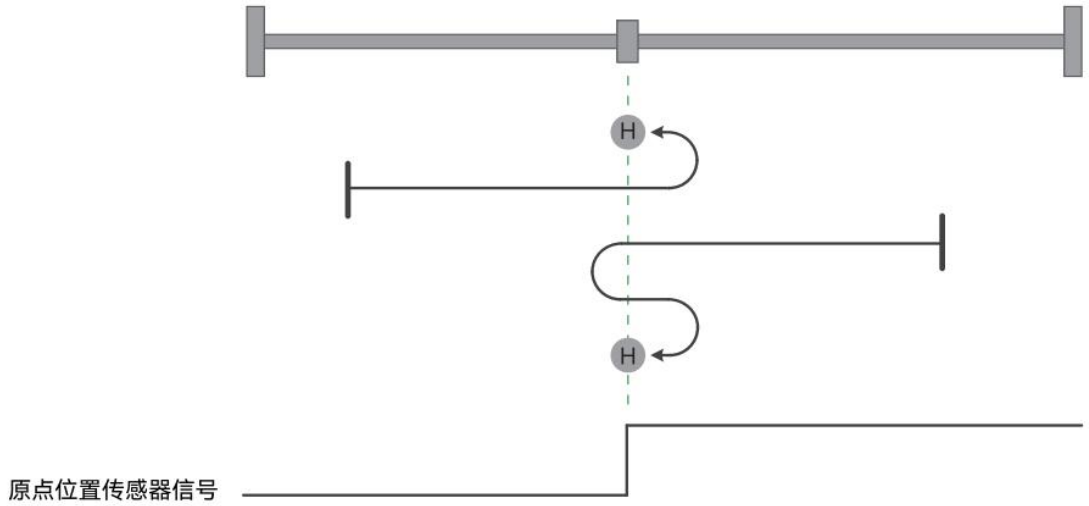
减速点：正向限位开关



模式 19：负向寻找原点开关 ON→OFF 位置

原点：原点开关(HW)

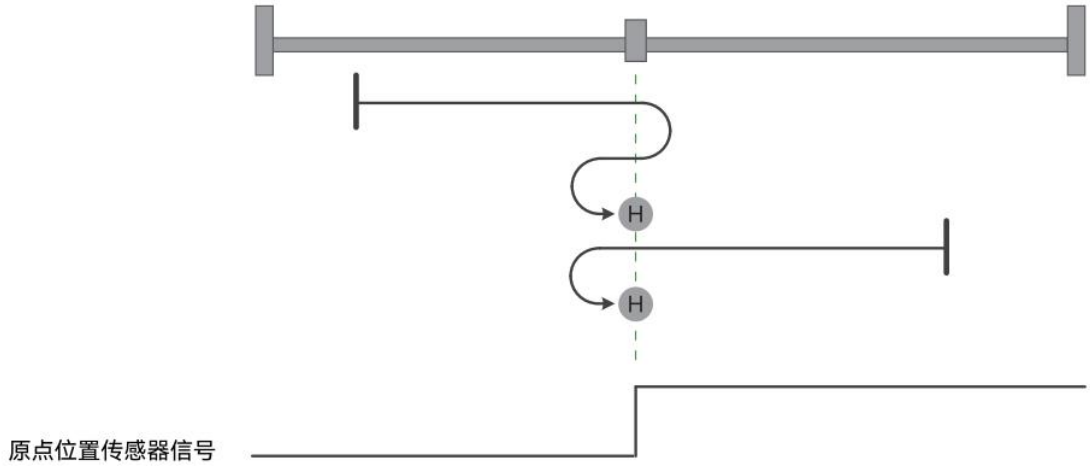
减速点：原点开关(HW)



模式 20：正向寻找原点开关 OFF→ON 位置

原点：原点开关(HW)

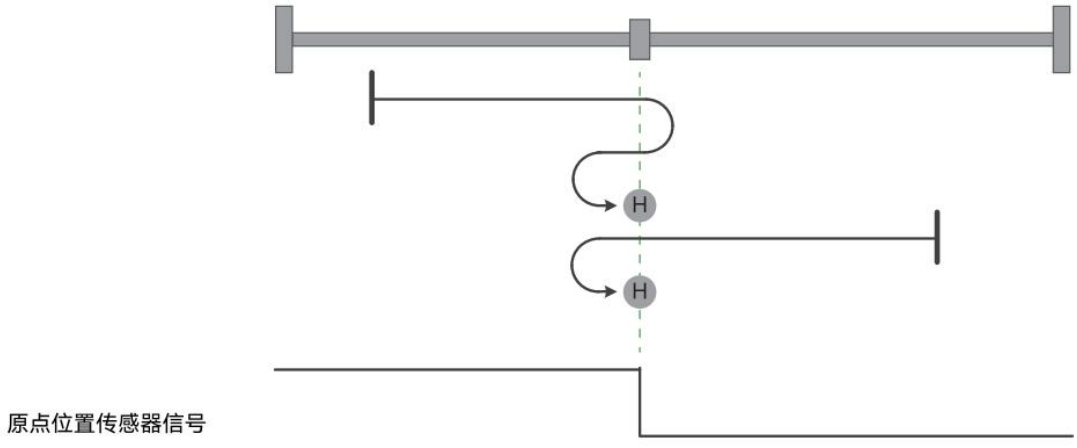
减速点：原点开关(HW)



模式 21：正向寻找原点开关 ON→OFF 位置

原点：原点开关(HW)

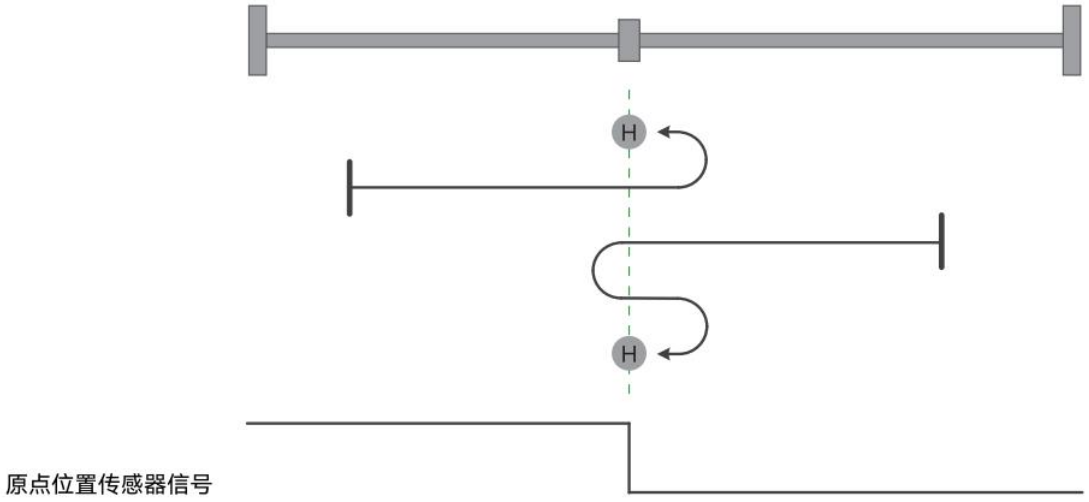
减速点：原点开关(HW)



模式 22：负向寻找原点开关 OFF→ON 位置

原点：原点开关(HW)

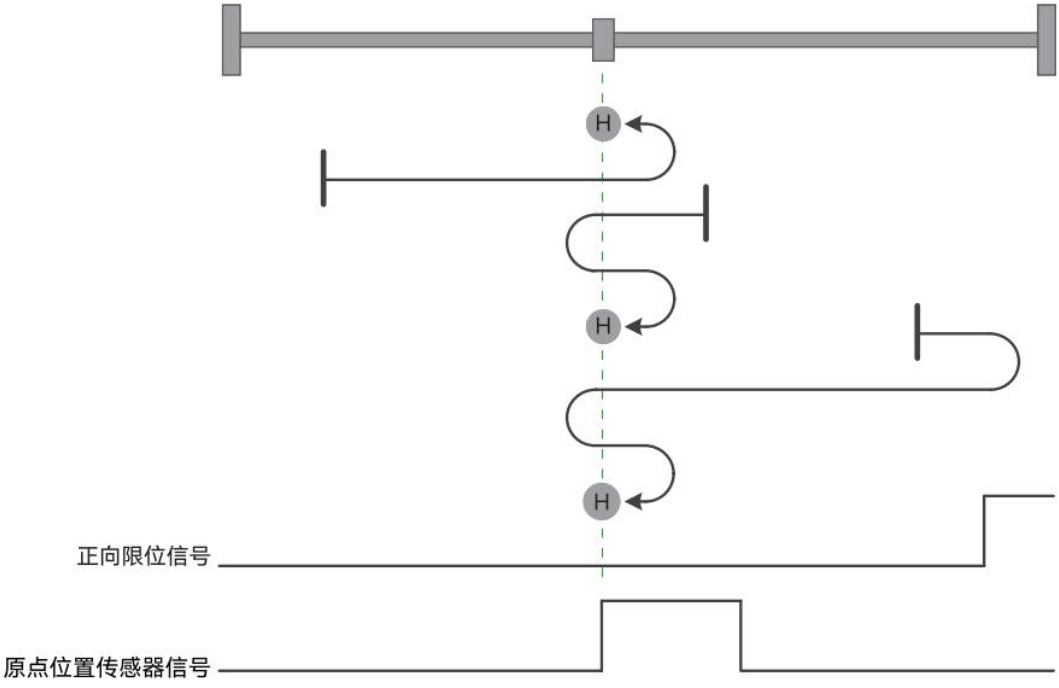
减速点：原点开关(HW)



模式 23：负向寻找原点开关 ON→OFF 位置，正限位自动反向

原点：原点开关(HW)

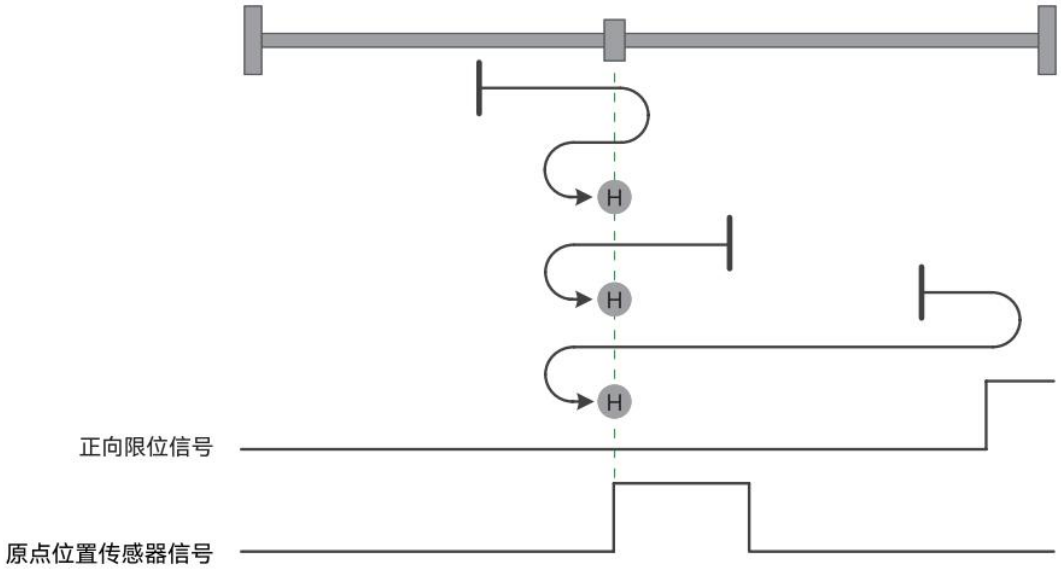
减速点：原点开关(HW)



模式 24：正向寻找原点开关 OFF→ON 位置，正限位自动反向

原点：原点开关(HW)

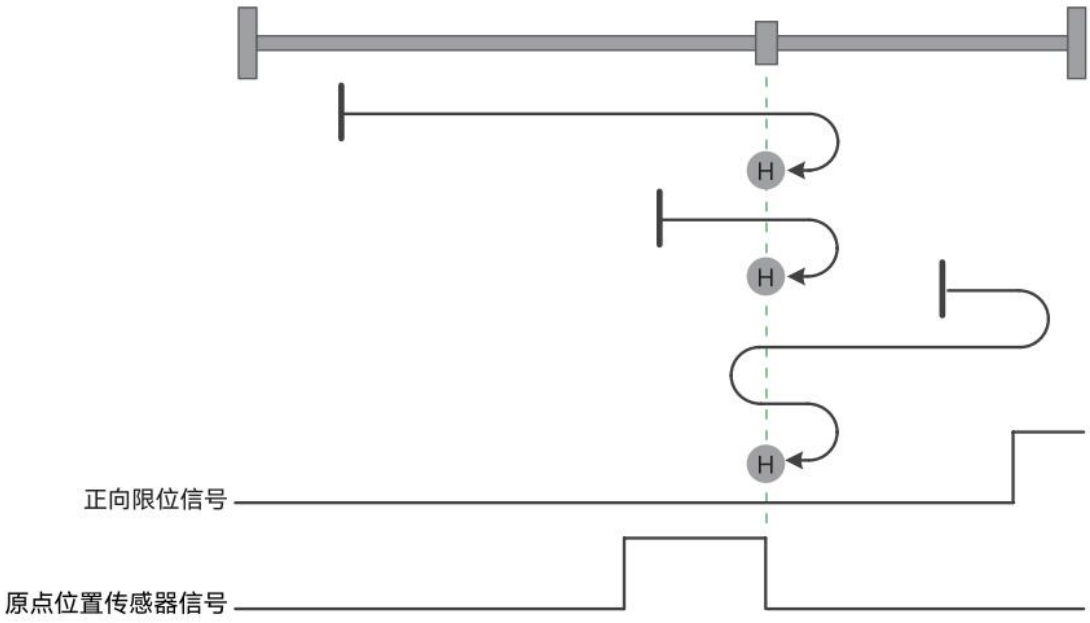
减速点：原点开关(HW)



模式 25：负向寻找原点开关 OFF→ON 位置，正限位自动反向

原点：原点开关(HW)

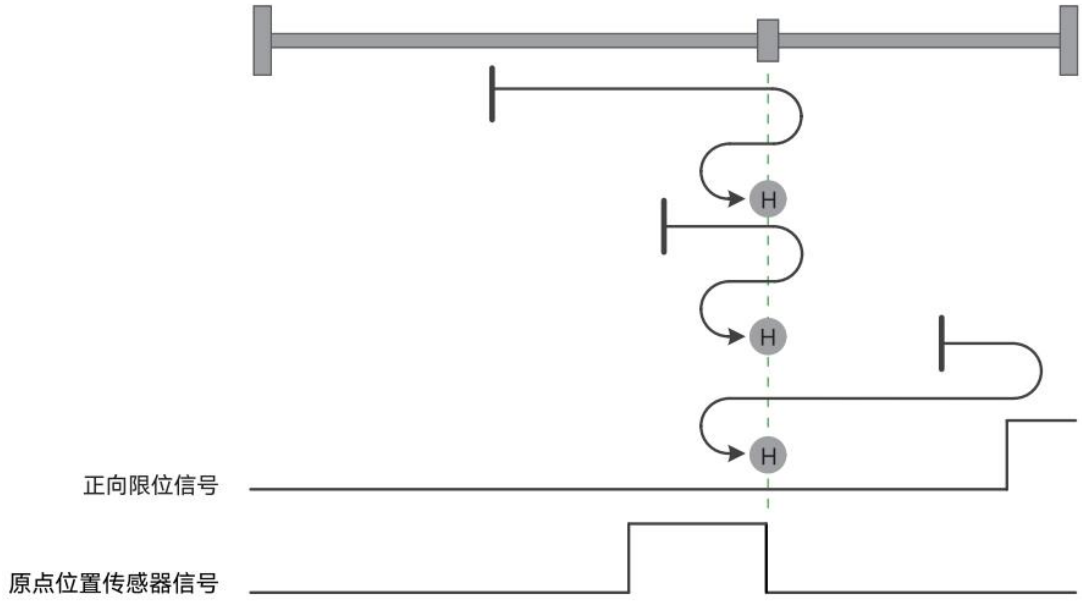
减速点：原点开关(HW)



模式 26：正向寻找原点开关 ON→OFF 位置，正限位自动反向

原点：原点开关(HW)

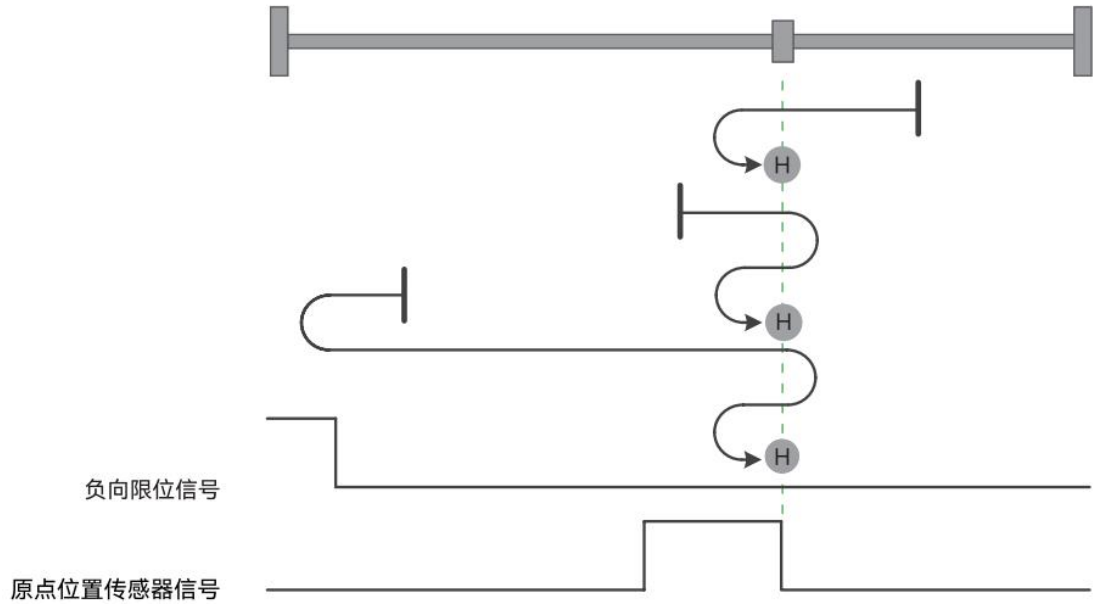
减速点：原点开关(HW)



模式 27：正向寻找原点开关 ON→OFF 位置，负限位自动反向

原点：原点开关(HW)

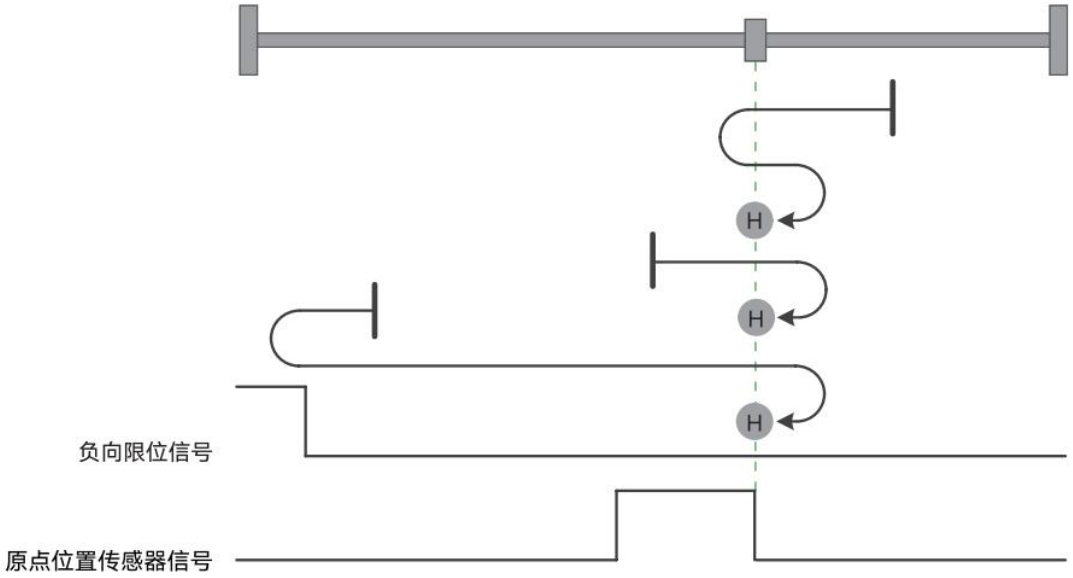
减速点：原点开关(HW)



模式 28：负向寻找原点开关 OFF→ON 位置，负限位自动反向

原点：原点开关(HW)

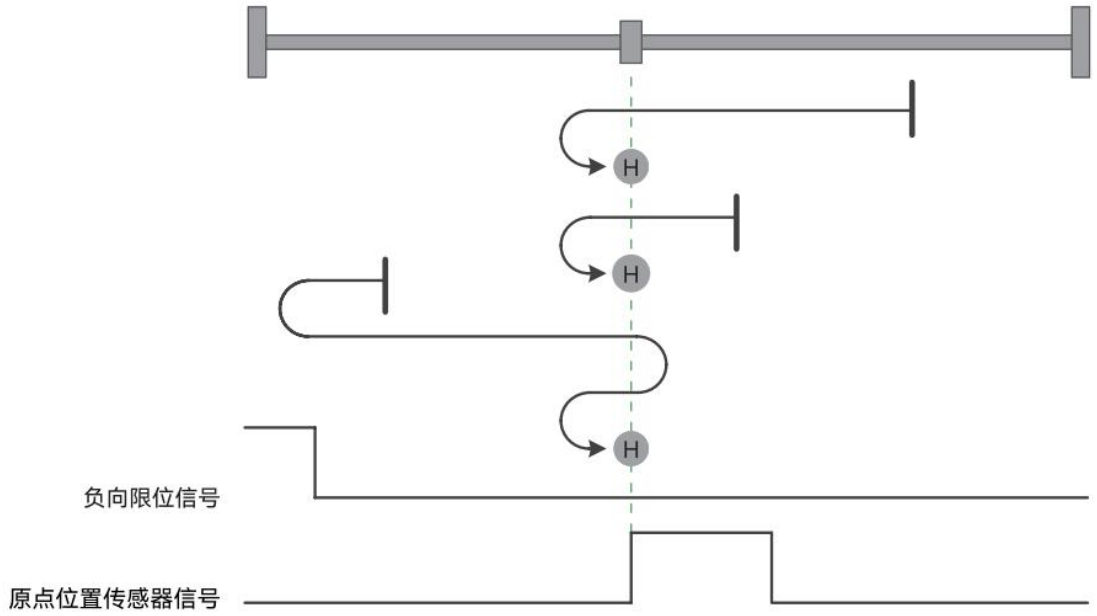
减速点：原点开关(HW)



模式 29：正向寻找原点开关 OFF→ON 位置，负限位自动反向

原点：原点开关(HW)

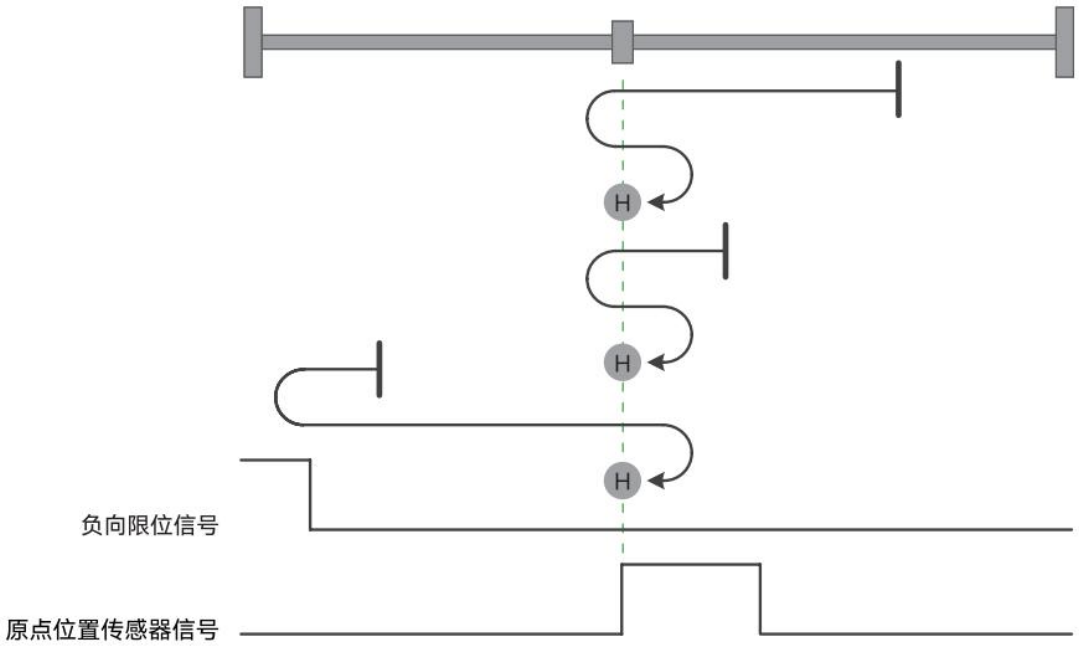
减速点：原点开关(HW)



模式 30：负向寻找原点开关 ON→OFF 位置，负限位自动反向

原点：原点开关(HW)

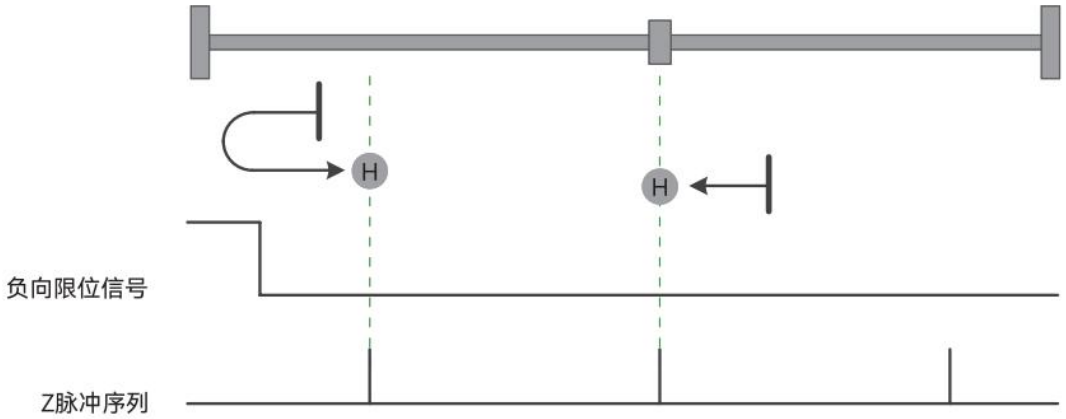
减速点：原点开关(HW)



模式 33：寻找负向运行时最近的 Z 脉冲

原点：Z 信号

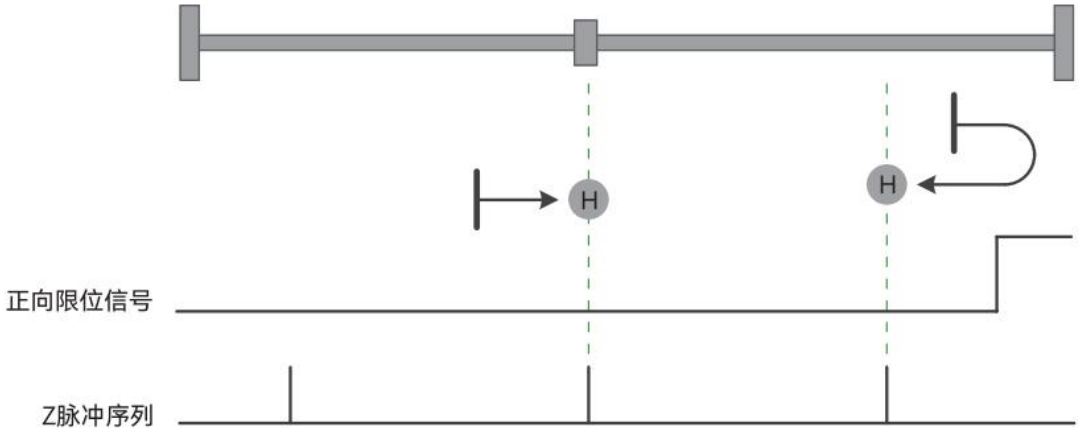
减速点：无



模式 34：寻找正向运行时最近的 Z 脉冲

原点：Z 信号

减速点：无



F4 信息源

enum PosSourceTypeEnum

{

[Description("[0]编码器计数位置")]

PosSource_EncoderPulse = 0,

[Description("[1]轴指令位置")]

PosSource_AxisCommand = 1,

[Description("[2]轴反馈位置")]

PosSource_AxisFeedback = 2,

[Description("[3]脉冲计数器位置")]

PosSource_AxisPulseCounter = 3,

[Description("[10]轴指令速度")]

PosSource_AxisCommandSpeed = 10,

[Description("[11]轴反馈速度")]

```
PosSource_AxisFeedbackSpeed = 11,
[Description("[20]轴输入状态")]
PosSource_AxisInputState = 20,
[Description("[21]轴输出状态")]
PosSource_AxisOutputState = 21,
}
```