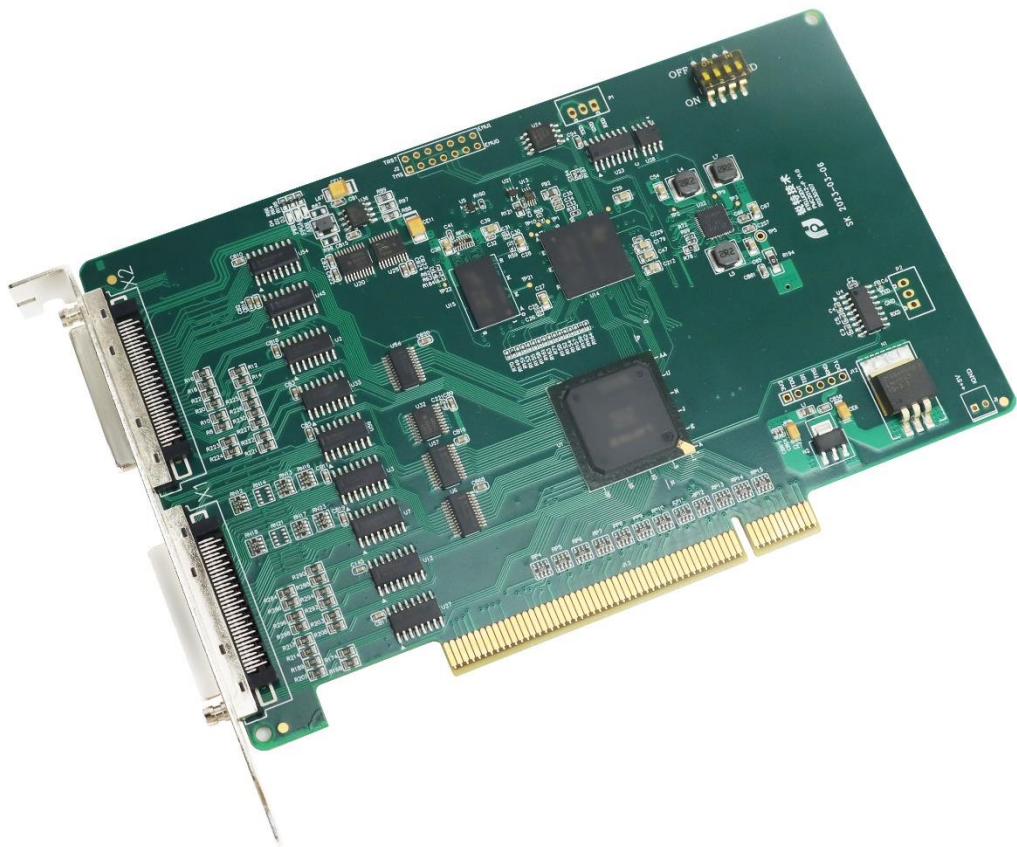

PMC2000-P 系列运动控制卡

使用手册





深圳锐特控制技术有限公司

手册版本

版本号	修改日期
V1.0	2023/03/16



版权申明

深圳市锐特控制技术有限公司

保留所有权力

本手册版权归深圳市锐特控制技术有限公司所有，未经公司书面许可，任何人不得翻印、翻译和抄袭本手册中的任何内容。

深圳锐特控制技术有限公司不承担由于使用本手册或本产品不当，所造成直接的、间接的、特殊的、附带的或相应产生的损失或责任。

本手册中的信息资料仅供参考。深圳市锐特控制技术有限公司保留对本资料的最终解释权，内容如有更改，恕不另行通知。

联系我们

深圳市锐特控制技术有限公司

地址：深圳市宝安区固戍南昌路庄边工业园

B 栋 3 楼

电话：+86 (0)755 29503086

传真：+86 (0)755 23327086

邮箱：sales@szruitech.com

华东办事处

地址：江苏省昆山市人民南路 888

号汇杰商务大厦 604 室

联系人：唐女士

电话：15202122728

邮箱：sales03@szruitech.com

山东办事处

地址：山东省济南市槐荫区西进时代中心

D 座六层 621

联系人：鹿先生

电话：13854109911

邮箱：sales06@szruitech.com

网址：www.szruitech.com



前言

本手册旨在帮助用户学习 PMC2000-P 系列控制卡的使用，用户通过阅读本手册，能够了解该系列运动控制卡的基本控制功能，掌握函数的用法，熟悉特定控制功能的编程实现，对函数错误返回的排查等。

若用户需要硬件相关参数及接线方法请查阅 PMC2000-P 系列控制卡的硬件手册。



目录

前言	4
第 1 章 概述	8
1.1 产品型号说明	8
1.2 典型应用	9
第 2 章 硬件及驱动程序安装	10
2.1 硬件安装步骤	10
2.2 驱动程序安装	12
第 3 章 软件演示	22
3.1 轴模块	22
3.2 插补模块	24
3.3 功能-IO	25
3.4 功能-IO 映射&复用	25
3.5 功能-手轮	26
3.6 功能-一维比较	26
3.7 功能-二维比较	27
3.8 功能-高速锁存	28
3.9 功能-PWM	28
第 4 章 应用程序开发	30
4.1 Visual C++ 6.0	31
4.2 Visual Studio	32
4.3 Visual C#	34



第 5 章	运动功能详解及实现.....	36
5.1	控制轴相关信号.....	36
5.2	坐标系当量配置.....	37
5.3	软件限位.....	38
5.4	轴状态获取.....	40
5.5	点位运动.....	44
5.6	速度运动.....	45
5.7	回零运动.....	46
5.8	直线插补.....	52
5.9	圆弧插补.....	53
5.10	空间圆弧插补.....	56
5.11	椭圆插补.....	59
5.12	通用 IO 控制.....	61
5.13	虚拟 IO 复用.....	65
5.14	轴专用信号映射.....	66
5.15	手轮.....	68
5.16	一维软件比较.....	70
5.17	一维高速比较.....	72
5.18	二维软件比较.....	74
5.19	高速锁存.....	76
第 6 章	附录.....	80



6.1	函数列表	80
6.2	函数说明	87
6.3	错误码	165



第1章 概述

PMC2000-P 系列运动控制卡是一款由深圳锐特控制技术有限公司自主研发的高性能、高可靠的 PCI 总线脉冲型运动控制卡，提供 4 轴、8 轴、12 轴选择方案。我们以完善的售后服务、高效的技术支持致力于帮助客户使用 PMC2000-P 系列运动控制卡建立自己的控制系统。

1.1 产品型号说明

产品型号	型号代码	说明
PMC2012-P	P	P 表示 PCI 总线
	MC	MC 为我司控制卡产品代码
	2	2 表示系列号，1 为我司点位系列控制卡 2 为我司高性能系列控制卡
	0	版本，0 为通用发布版本
	12	这两位表示轴数 4 表示 4 轴版本 6 表示 6 轴版本 8 表示 8 轴版本 12 表示 12 轴版本
	-P	-P 表示为脉冲款运动控制卡



1.2 典型应用

PMC2000-P 系列运动控制卡适用于各行各业自动化设备中。主要设备有：

- ◆ 电子产品装配、测量设备、台式点胶设备
- ◆ 固晶机、LCD 生产设备、激光加工设备
- ◆ 生物、医学自动采样、处理设备
- ◆ 机器视觉及自动检测设备
- ◆ 其它控制步进电机、伺服电机的自动化设备

第2章 硬件及驱动程序安装

在拿到 PMC2000-P 系列运动控制卡之后需要对硬件进行如下检查：

- ◆ 板卡主体是否有明显损坏
- ◆ 板卡金手指处是否完整,保证能够良好接触

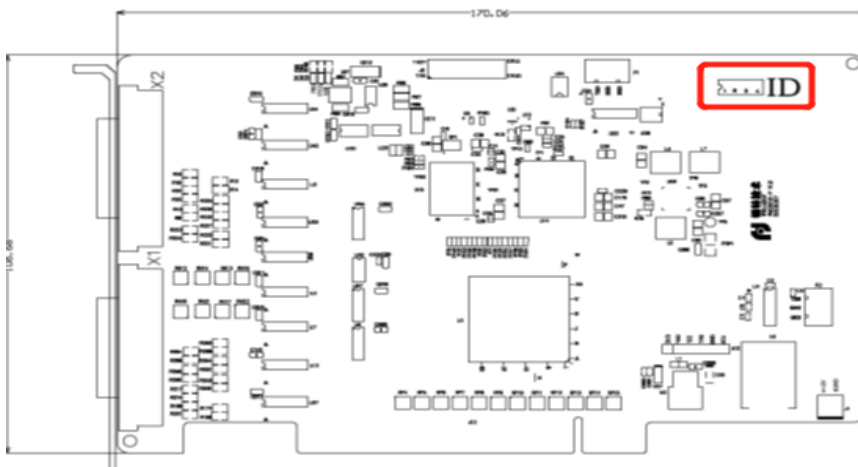
在安装前检查配件是否齐全一套 PMC2000-P 运动控制卡配件如下表所示：

产品名称	数量
控制卡主卡 PMC2000-P	1
控制卡接线盒 PMC2000EX	1
电缆线 SCSI68-2.0M-CN-VHDCI	2

2.1 硬件安装步骤

2.1.1 拨码开关设置

PMC2000-P 系列运动控制卡板卡正面提供拨码开关 S1 用于设置控制卡的控制卡号。拨码开关在板卡的位置如下图所示：



拨码开关 S1 的四个拨码采用的是 0/1 的二进制编码原理，拨到 ON 则该位



为 1，拨到 OFF 则该位为 0。拨码组合对应的卡号如下表所示：

S1-4	S1-3	S1-2	S1-1	控制卡号
保留	OFF	OFF	OFF	0
保留	OFF	OFF	ON	1
保留	OFF	ON	OFF	2
保留	OFF	ON	ON	3
保留	ON	OFF	OFF	4
保留	ON	OFF	ON	5
保留	ON	ON	OFF	6
保留	ON	ON	ON	7

注意：(1)拨码开关 S1 出厂默认设置为全 OFF，即默认卡号设置为 0；当电脑插入多张默认卡号为 0 的卡时会根据靠近 CPU 的顺序自动排序。

(2)除默认卡号 0 外，当多张卡存在相同的卡号时初始化函数将会返回错误码

2.1.2 硬件安装

- 1) PC 机箱接地并关闭电源
- 2) 将控制卡插入主机机箱中的 PCI 插槽中
- 3) 拧紧螺丝，保证控制卡和 PCI 插槽接触良好稳定
- 4) 通过 68pin 线连接控制卡与接线盒，控制卡的连接接口需要和接线盒上的接口位置对应
- 5) 主机上电开机，接线盒上电能够看到接线盒指示灯为绿色表示连接正常状态。

注意：

- (1) 在拔插控制卡时主机机箱必须完全断电，否则可能造成设备损坏。
- (2) 控制卡安装完成后必须拧紧螺丝，否则控制卡的 PCI 接口出现松动会造成软件运行的错误。



- (3) PC 机必须接地，否则可能产生干扰输入，造成脉冲毛刺。
- (4) 控制卡的 CN1 口连接接线盒必须和接线盒上的 C 口对应。当连接错误时接线盒上的状态指示灯为红色，调换两个接口的接线即可。接线盒正常通讯时状态指示灯为绿色。

2.2 驱动程序安装

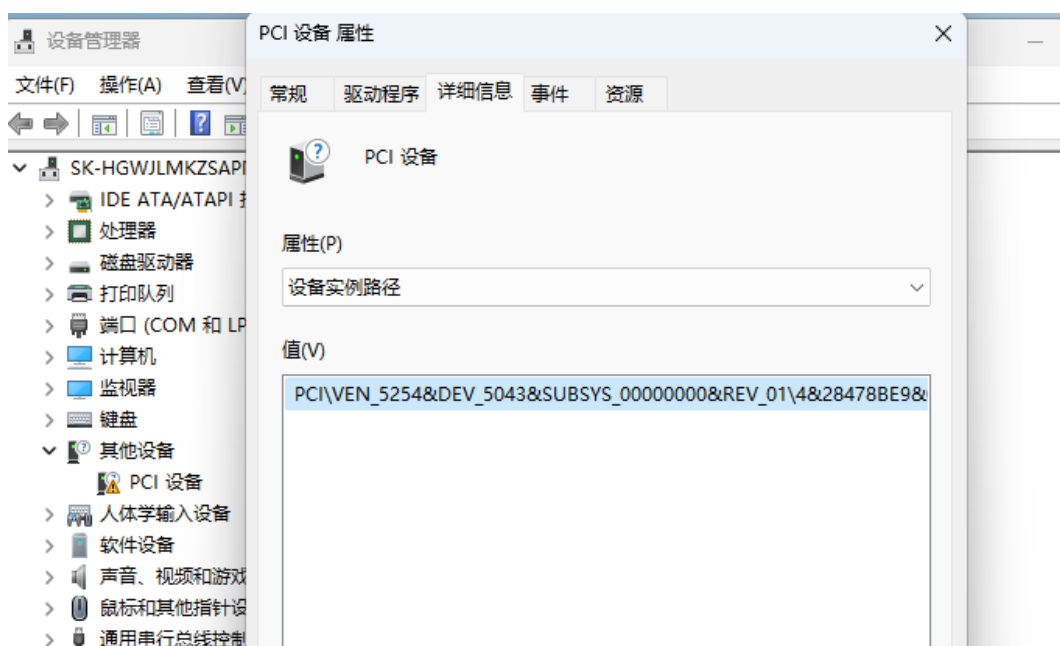
PMC2000-P 系列提供的开发套件中包括 32 位系统的驱动程序和 64 位系统的驱动程序，客户需要根据自身系统位数选择相应的驱动文件进行安装。

我司提供的开发套件文件夹名称使用 32/64 来区分不同系统位数的驱动文件。若存在驱动安装问题请与我司技术支持联系，我司将竭诚为您服务。

2.2.1 Windows10

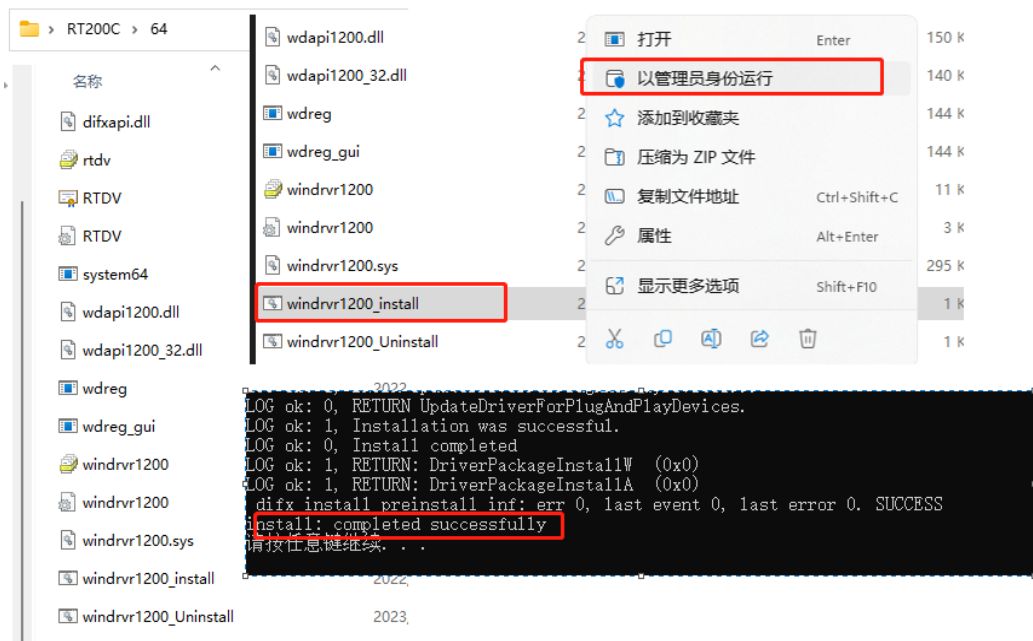
- 1) 在将板卡插入计算机后；鼠标右键单击“计算机”->选择“设备管理器”，在设备管理器中是否找得到黄色感叹号的“PCI 设备”选项，该设备的详细信息中 VEN_5254,DEV_5043 代表着该设备的厂商代码和设备代码。

如下图所示。





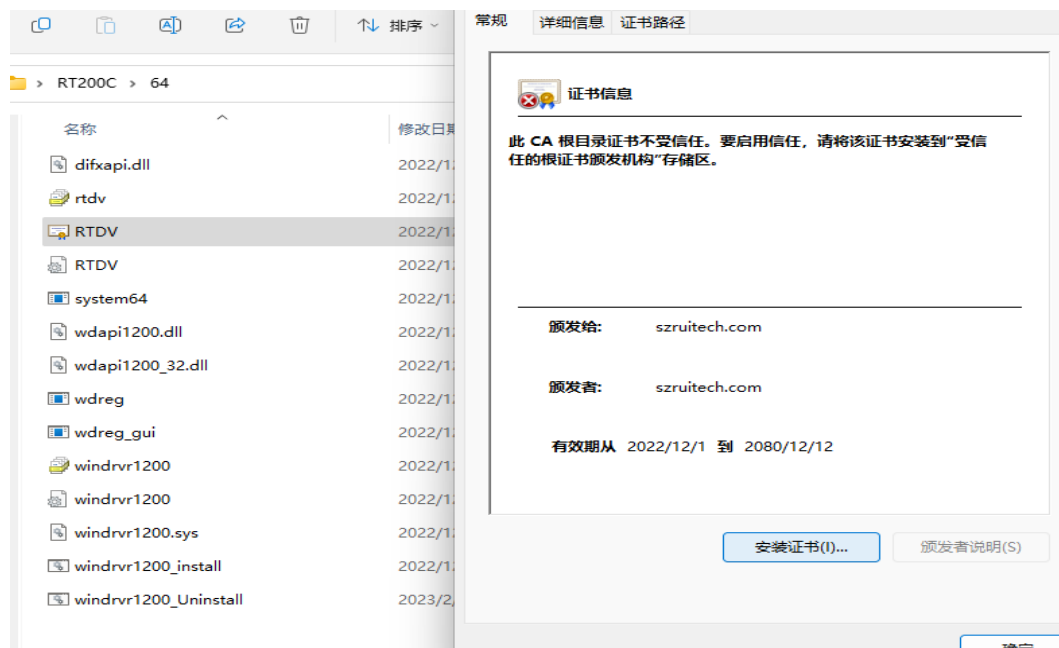
- 2) 找到提供的板卡驱动程序，根据系统位数进入相关的驱动安装文件夹；
找到文件"windrvr1200_install";右键以管理员权限运行该批处理文件，安装成功会以"completed successfully"提示。



- 3) 这时再看设备管理器会有一个 WinDriver1200 成功安装的设备



4) 找到驱动文件路径下的 RTDV 证书文件



5) 点击安装证书，证书储存位置选择"本地计算机"，选择下一页。

欢迎使用证书导入向导

该向导可帮助你将证书、证书信任列表和证书吊销列表从磁盘复制到证书存储。

由证书颁发机构颁发的证书是对你身份的确认，它包含用来保护数据或建立安全网络连接的信息。证书存储是保存证书的系统区域。

存储位置

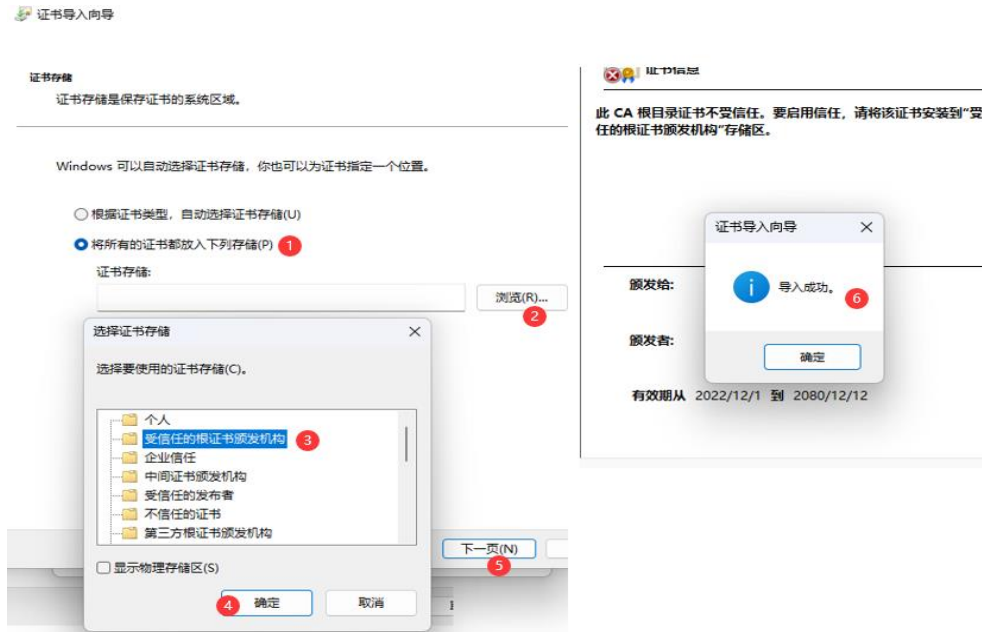
- ☐ 当前用户(C)
- ☒ 本地计算机(L)

单击"下一步"继续。

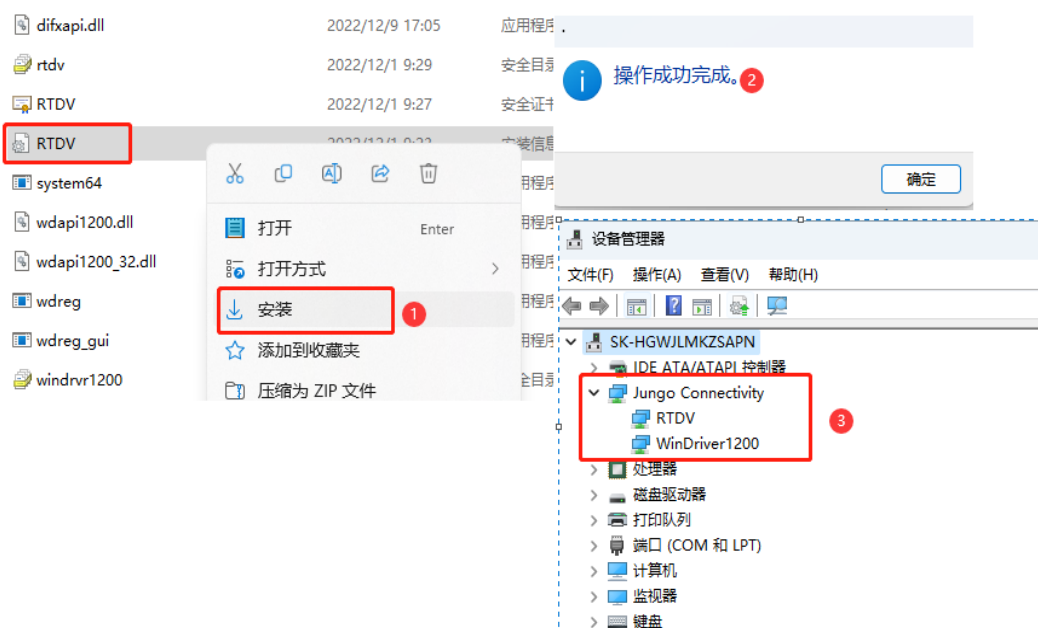
下一页(N)



- 6) 在证书储存中选择”将所有的证书都放在下列储存”，点击“浏览”，选择“受信任的根证书颁发机构”；点击“确定”；点击“下一页”；直到成功导入证书。

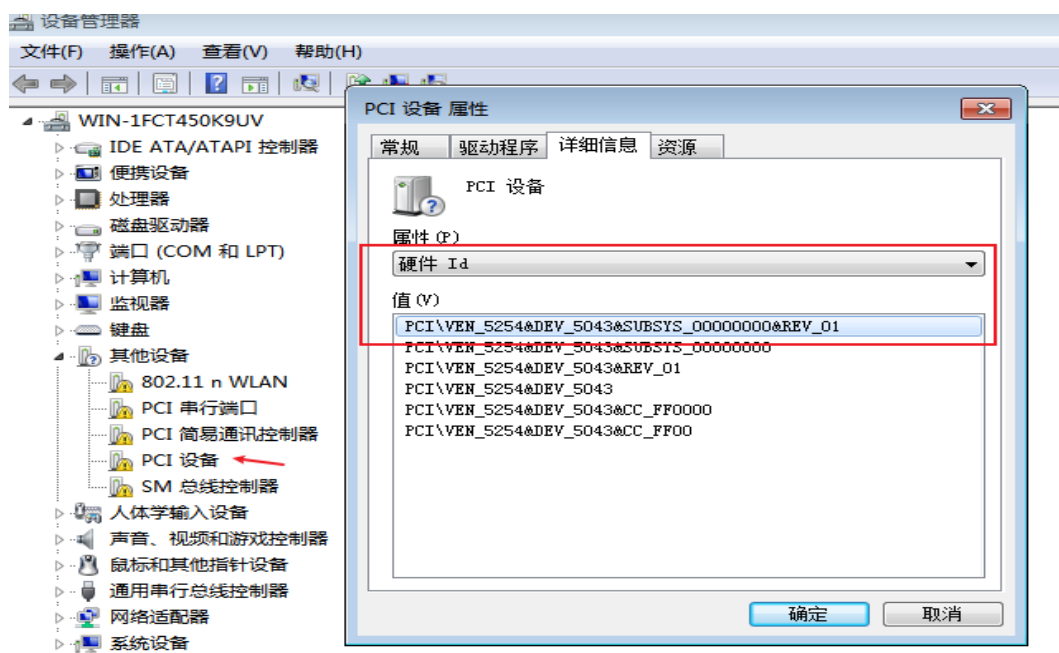


- 7) 在安装完证书之后，找到驱动文件，点击右键“安装”，选择“安装设备”；成功安装提示“操作成功完成”；这是在设备管理器中能够看到设备驱动已经成功安装，能够正常使用。



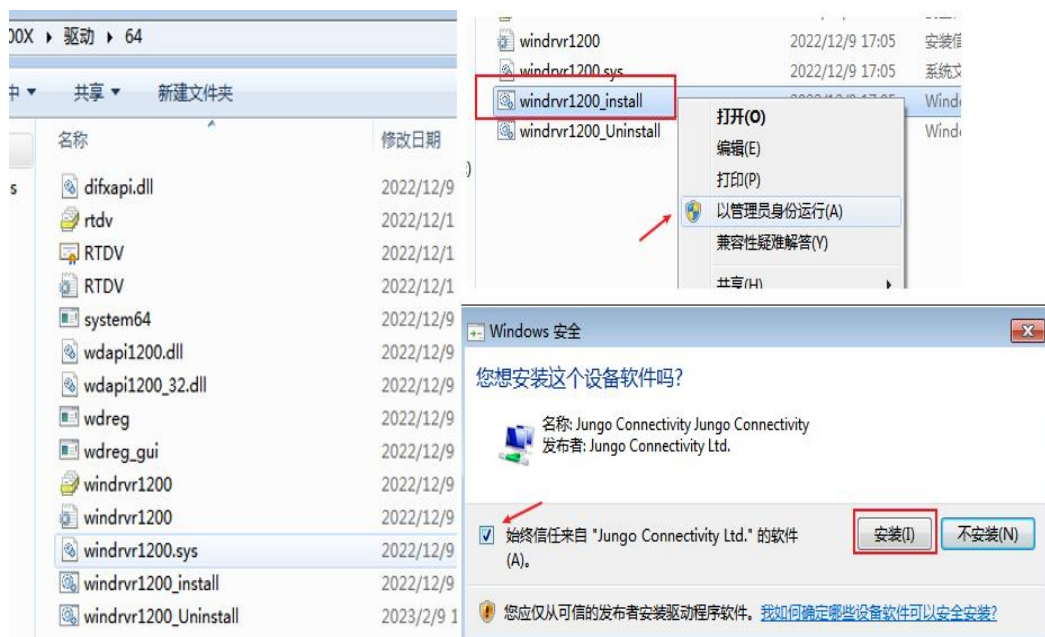
2.2.2 Windows7

- 1) 在将板卡插入计算机后；鼠标右键单击“计算机”->选择“设备管理器”，
在设备管理器中是否找得到黄色感叹号的“PCI 设备”选项，该设备的详细信息中 VEN_5254,DEV_5043 代表着该设备的厂商代码和设备代码。
如下图所示。

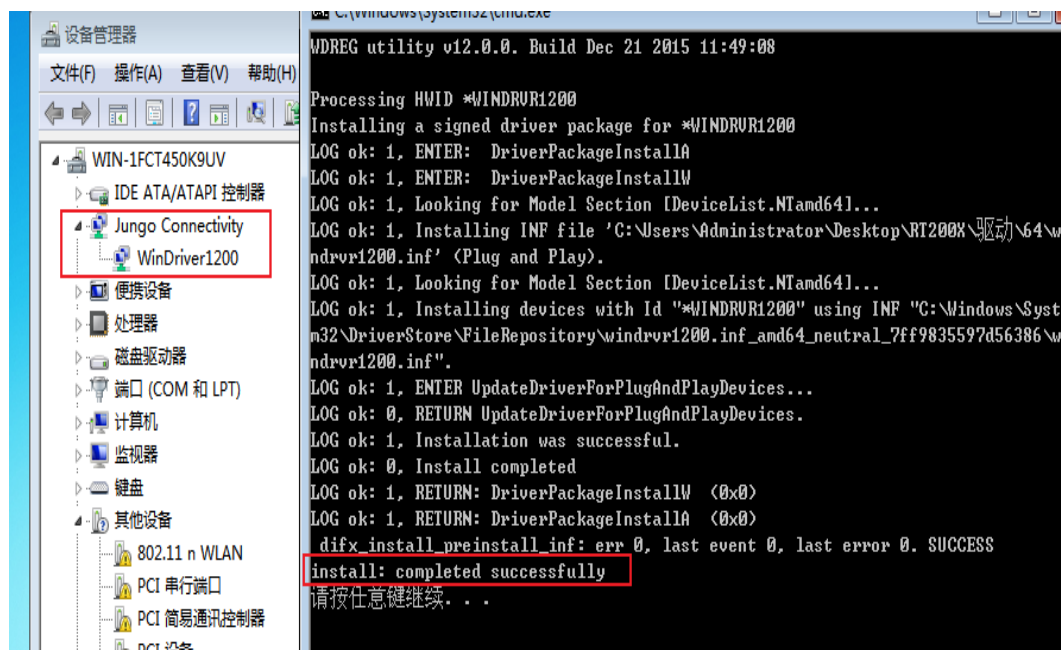




- 2) 在驱动文件路径下找到”windrvr1200_install”批处理文件，然后右键以管理员身份运行，在弹出的设备安装提示中选择“安装”。

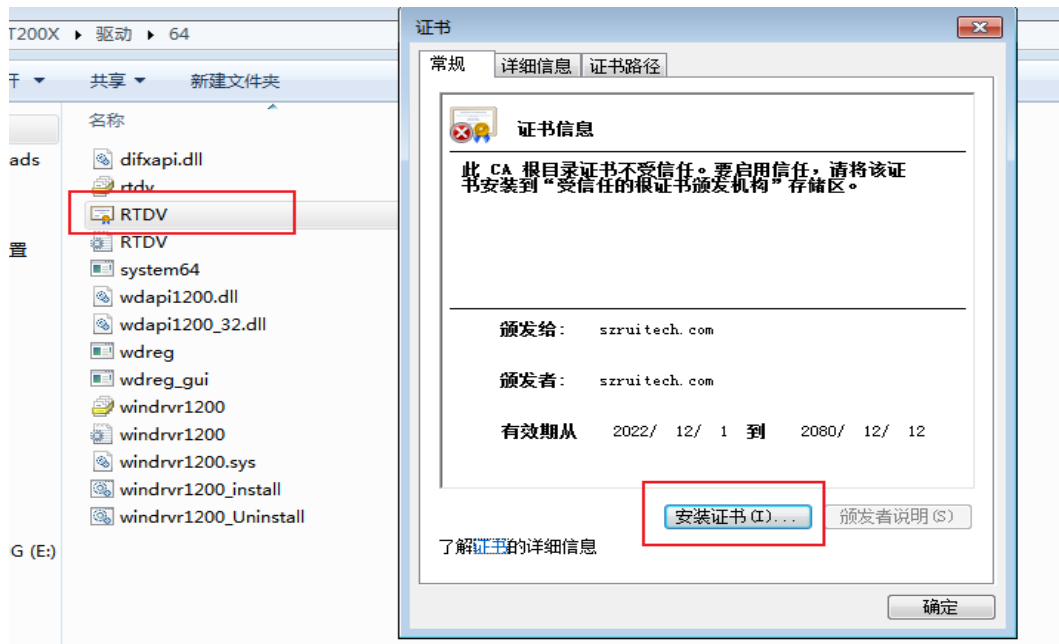


- 3) 在成功安装之后，会打印”completed successfully”等提示，同时设备管理器中新增 WinDriver1200 设备。

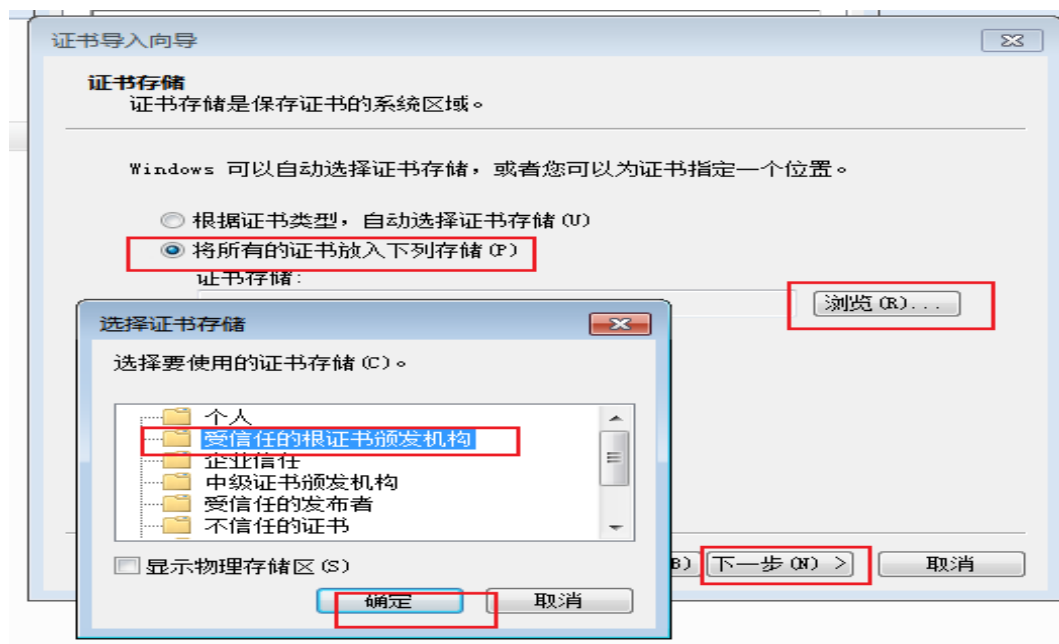




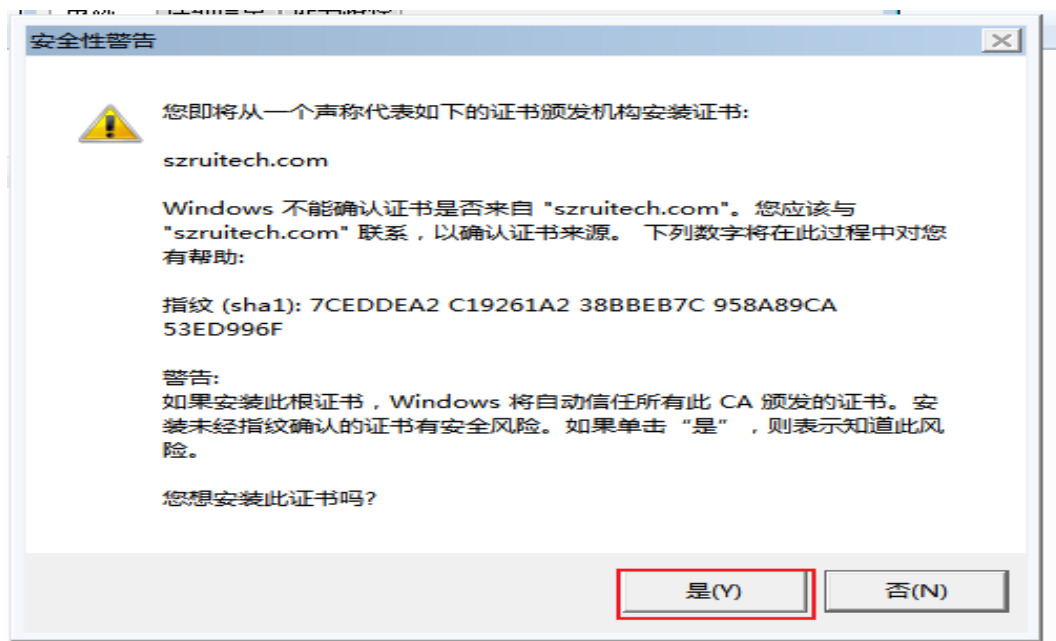
- 4) 在驱动文件路径下找到 RTDV 证书文件，打开之后点击“安装证书”。



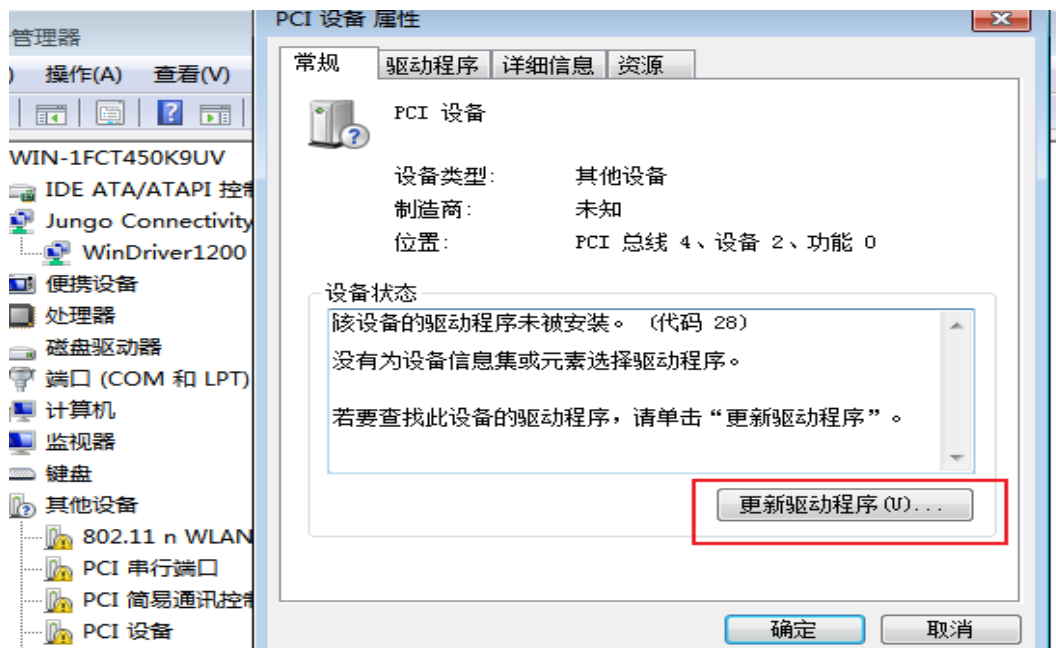
- 5) 在证书储存页面中，选择“将所有的证书放入下列储存”；然后点击“浏览”；选择要使用的证书储存为“受信任的根证书颁发机构”；点击确定；再点击下一步。



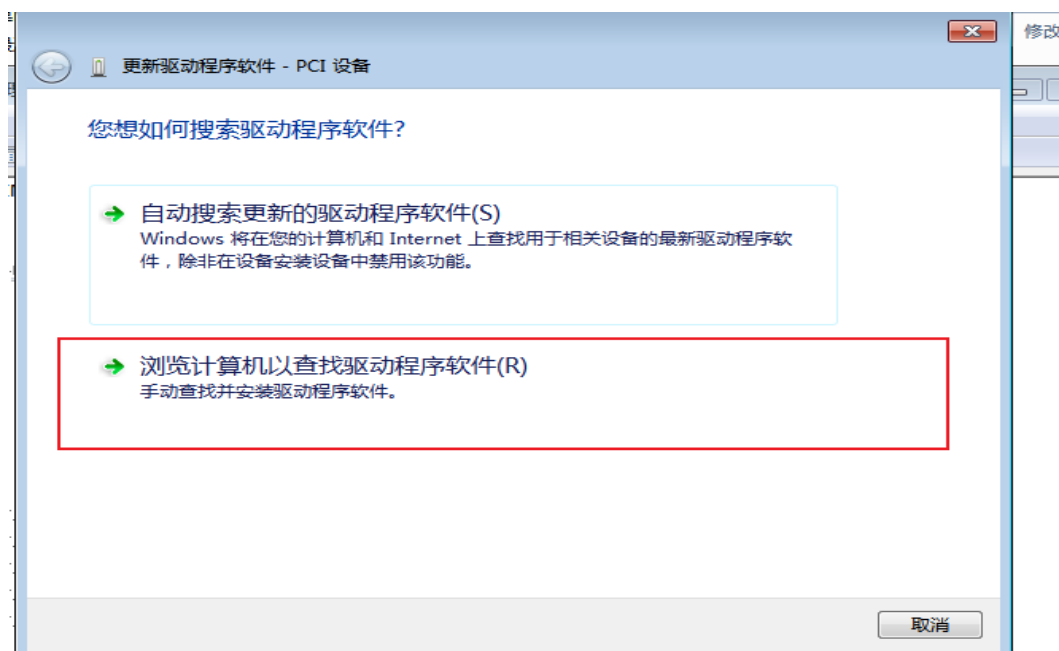
- 6) 在点击证书导入向导“完成”之后，出现证书的安全性警告，选择“是”。
之后证书导入成功。



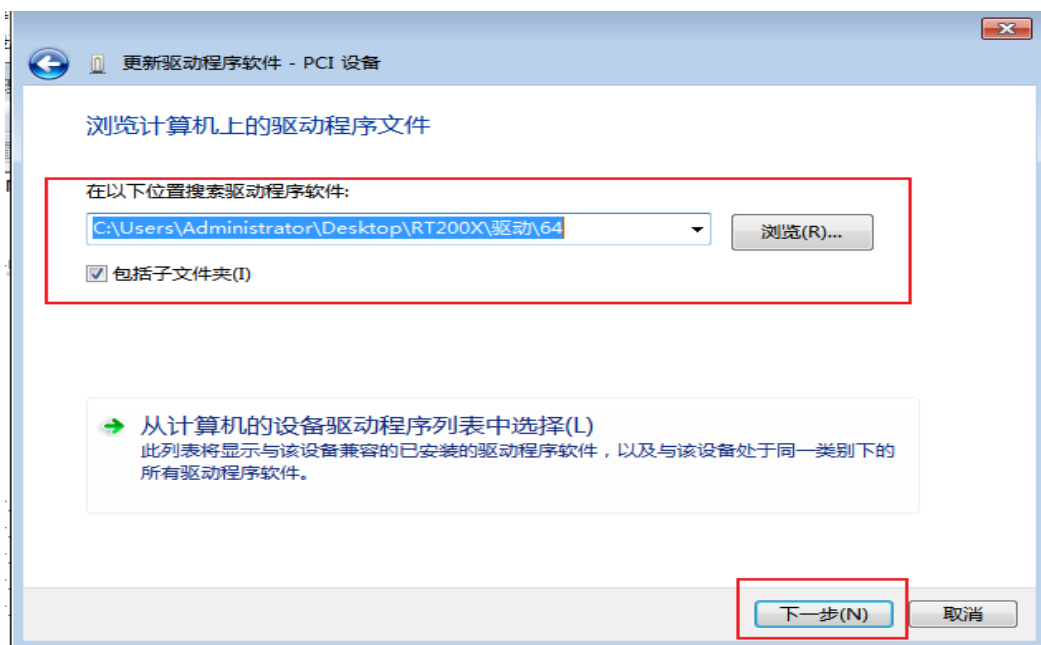
- 7) 在设备管理器中找到黄色感叹号的 PCI 设备；在设备属性中点击“更新驱动程序”。



- 8) 选择“浏览计算机以查找驱动程序软件”。

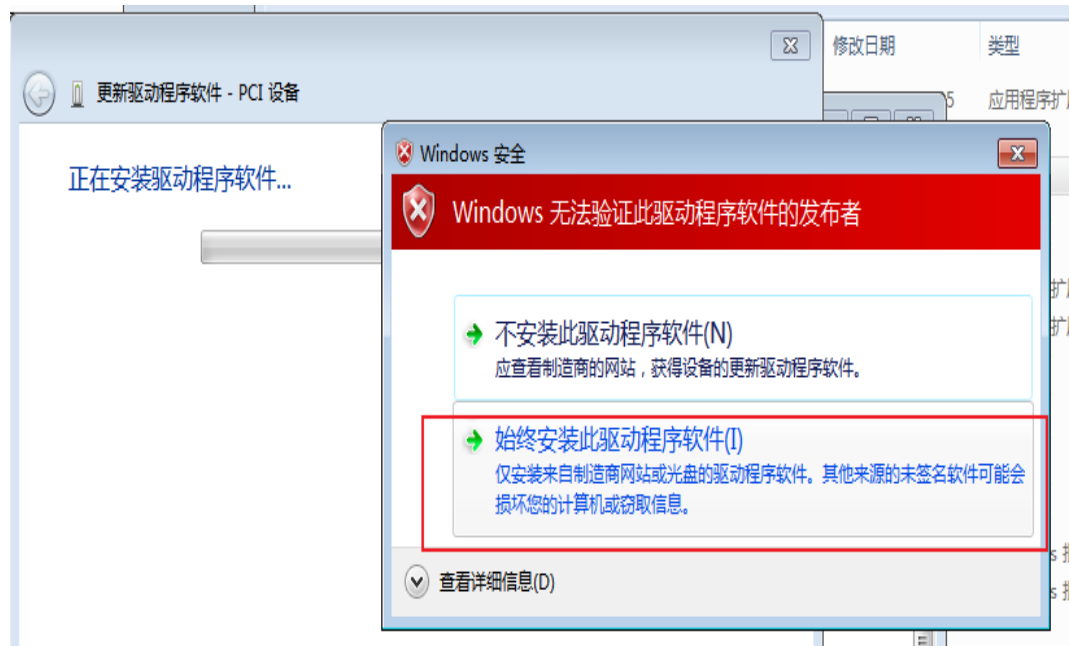


9) 在驱动文件路径中选择驱动文件路径，然后点击“下一步”。

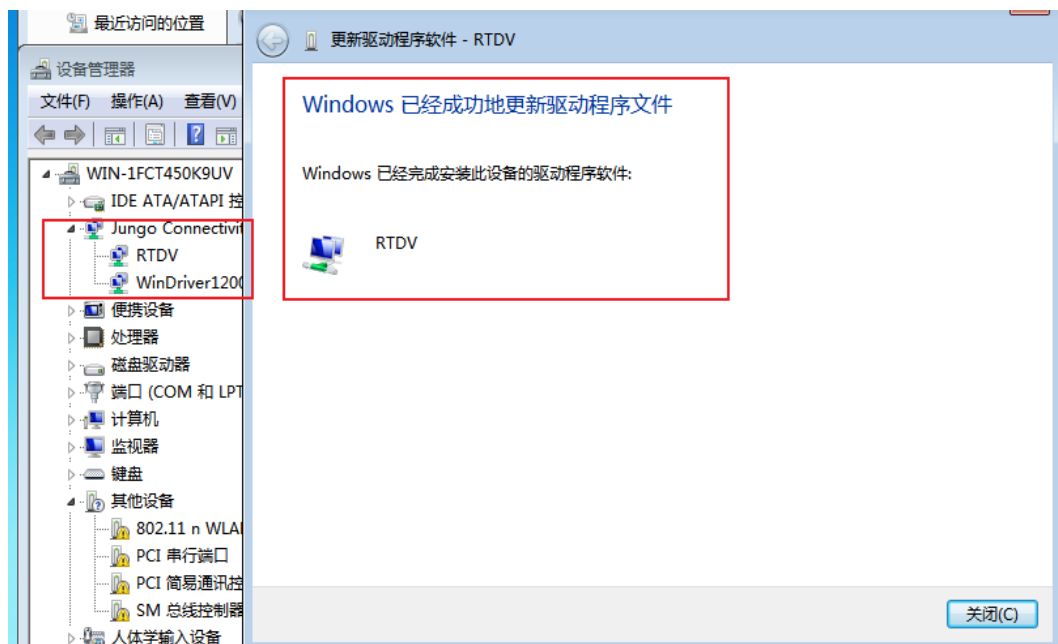




10) 在出现的安全提示中选择“始终安装此驱动程序软件”。



11) 驱动文件成功安装，在设备管理器中出现 RTDV 设备和 WinDriver1200 设备。至此板卡能够正常使用。



第3章 软件演示

在硬件和驱动都正确安装之后，可以使用我司提供的 Motion 软件对控制卡进行操作。测试软件打开时会自动进行开卡并对卡版本进行识别。软件打开界面如下图所示：



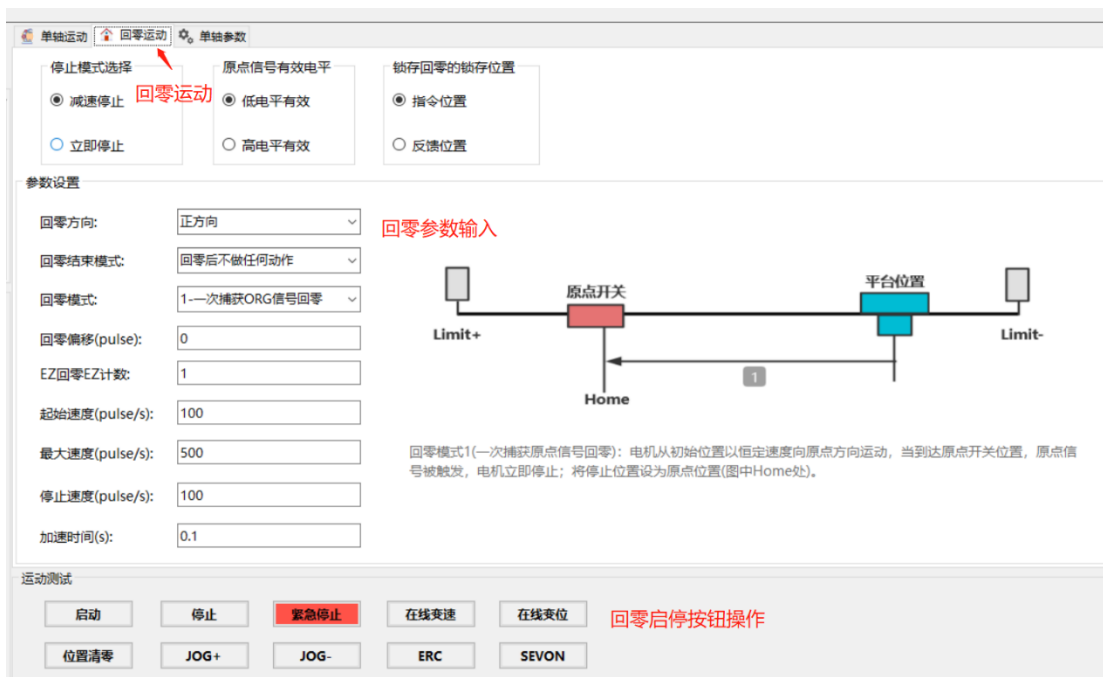
在测试软件成功开卡时状态栏底部会显示“就绪”的绿色成功连接提示表示成功开卡，控制卡列表会展开当前卡的功能模块信息。每个模块都是独立的界面，能够对控制卡的不同轴不同功能进行控制。

3.1 轴模块

测试软件的轴模块为控制卡的总轴数划分了控制界面，每个界面的操作相互独立，能够对轴的参数设置进行操作还有单轴运动的演示，打开轴界面如下图所示：



单轴测试界面为回零运动提供了单独的选项卡界面，在回零运动界面用户能够选择回零模式和回零参数进行回零演示，回零运动界面如下图所示：



单轴模块提供了对单轴的参数配置界面，主要配置轴的运动参数还有轴专用信号的配置，当打开单轴参数界面时会自动从控制卡上加载目前的参数，用户在修改了界面的参数后需要点击“下载至控制卡”按钮来将参数下发到轴才能生效，在下发后用户可以再次点击“从控制卡加载”按钮来验证修改的参数是否成功写

入控制卡的配置，同时单轴界面参数提供了参数配置一键应用到所有轴的选项，界面如下图所示：

单轴运动

回零运动

单轴参数

下载至控制卡

从控制卡加载

导出参数

导入参数

应用到所有轴

	参数值	说明
最大加减速限制		设置值小于运动配置的加减速速度值时，最大加减速速度设置为其运动加减速速度
脉冲输出模式		
脉冲输出模式	0_脉冲高+方向高	脉冲输出模式
编码器工作模式	0_脉冲方向计数	轴对应编码器的工作模式
编码器的工作方向	0_A在前为正	轴对应编码器的工作方向
软件限位		
软件限位使能	0_禁止	软件限位是否使能
软件限位位置源	0_指令位置	软件限位的限位位置源
软件限位最大位移		软件限位的最大运动距离
软件限位最小位移		软件限位的最小运动距离
软件限位发生时停止模式	0_减速停止	软件限位限位触发时轴的停止模式
硬件限位		
负限位使能(EL-)	0_禁止	EL-信号是否使能
负限位触发有效电平(EL-)	0_低电平有效	负限位触发的有效电平
负限位触发停止模式(EL-)	0_减速停止	负限位触发时停止模式，减速停止或立即停止
正限位使能(EL+)	0_禁止	EL+信号是否使能
正限位触发有效电平(EL+)	0_低电平有效	正限位触发的有效电平
正限位触发停止模式(EL+)	0_减速停止	正限位触发时停止模式，减速停止或立即停止
原点信号(ORG)		

3.2 插补模块

插补模块提供了对插补运动功能的演示，在控制卡成功开卡时会根据支持的插补系数个数来生成对应的插补模块数量。插补模块界面打开如下图所示：

初始页

PMC2012-P-0-坐标系0

运动参数

起始速度(pulse/s): 100

停止速度(pulse/s): 100

减速时间(s): 0.1

最大速度(pulse/s): 500

加速时间(s): 0.1

S段时间(s): 0

插补系运动参数

速度规划

最小匀速时间(s): 0.1

S型匀加速时间比值[0-1]: 0.1

速度平滑: ☒ 禁止 ☐ 启用

速度模式: ☒ T型 ☐ S型

最大加速度限制(s): 10000000

平滑时间(s): 0.1

平滑参数

插补数据

插补类型: 直线插补

增加插补数据

增加数据

删除数据

清空数据

选择插补类型

行号	类型	参数
----	----	----

操作

启动插补运动

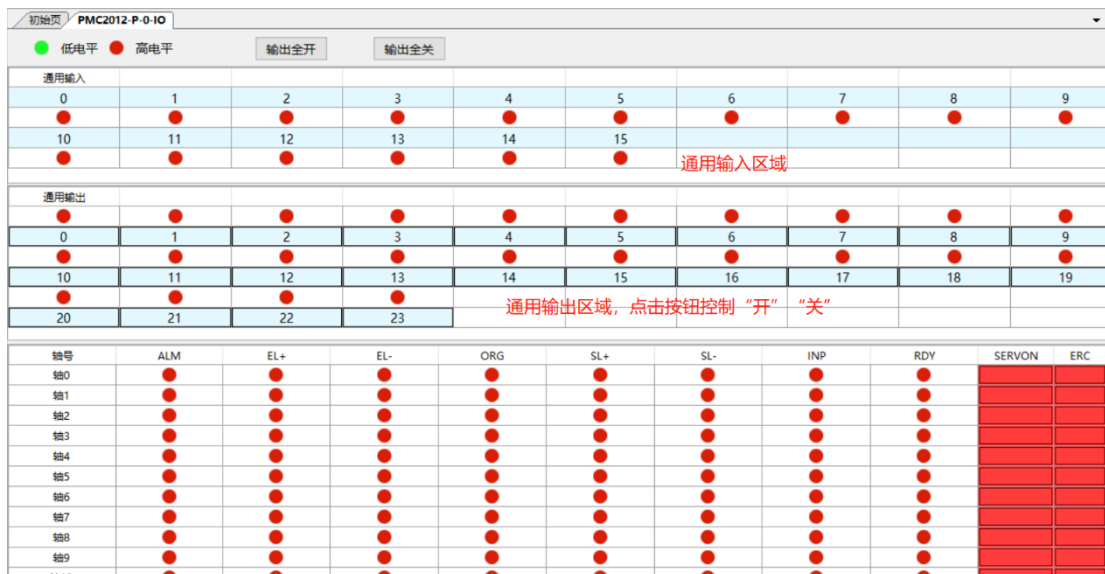
启动

停止

急停

3.3 功能-IO

在测试软件功能模块中 IO 界面能够监控板卡的所有 IO 信号，包括通用输入信号、通用输出信号和所有轴的专用信号，在成功开卡的情况下打开 IO 界面就自动开启监控，界面打开如下图所示：



3.4 功能-IO 映射&复用

在测试软件中提供了 IO 映射和 IO 复用功能的演示界面。

轴专用 IO 映射功能：该功能是通过将某个轴的专用信号映射到另一个轴的专用信号上或者某个通用输入上，通过映射可以达到硬件实际触发一个输入但是达到多个功能的效果，比如把轴 0 的负限位信号映射到轴 1 的负限位信号上此时当轴 1 的硬件负限位信号触发时相当于轴 0 的负限位也触发，轴 0 也会有负限位触发的效果。

虚拟 IO 复用功能：板卡内部提供了 32 位的虚拟 IO，通过虚拟 IO 复用能够共享硬件的接口；比如将虚拟输入信号 Bit0 映射到轴 0 不用的专用信号上，这

时就能够通过虚拟 IO 将专用信号当作普通输入来使用。

专用 IO 映射和虚拟 IO 复用功能的演示界面如下图所示：



3.5 功能-手轮

手轮功能能够通过辅助编码器达到视校机器的效果，手轮测试界面如下图所示：



3.6 功能-一维比较

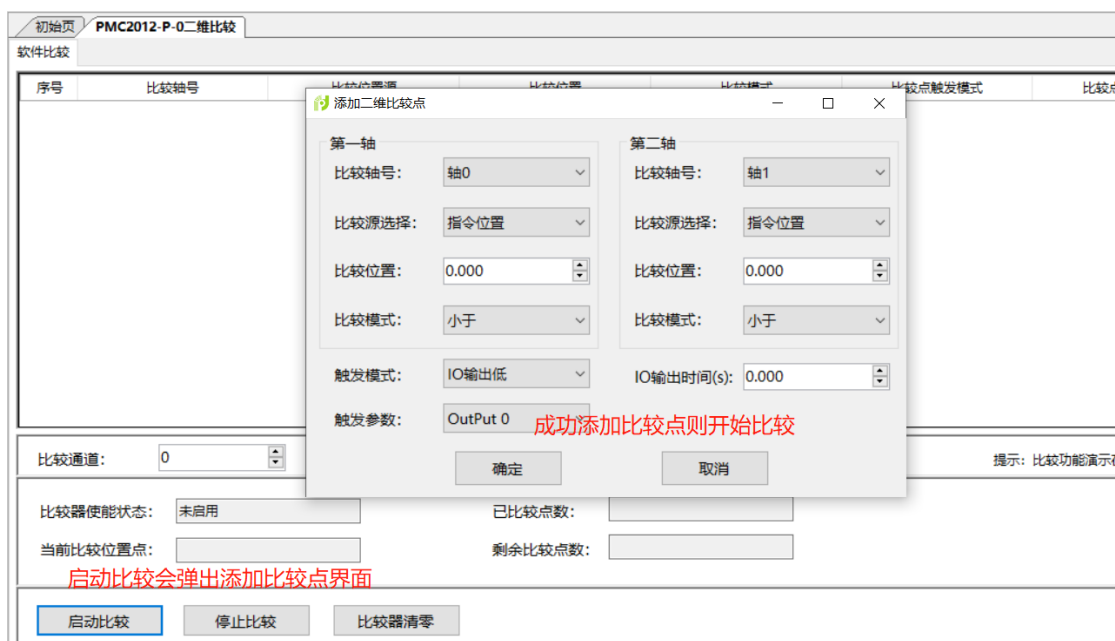
一维比较功能界面打开如下图所示：



每次启动比较都需要添加一个比较点，当比较点添加成功时开始比较，只有当前一个比较点完成比较后才会执行到后面添加的比较点。

3.7 功能-二维比较

二维比较相对于一维比较来说多了一个轴，只有当两个轴的比较条件同时满足时才会触发比较参数。二维比较的演示界面操作同一维比较相似，如下图所示：



3.8 功能-高速锁存

初始页 PMC2012-P-0-高速锁存

锁存参数

锁存对象

- ☐ 原点信号
- ☐ EZ信号
- ☒ 高速输入信号
- ☐ 通用输入信号

捕获通道

高速通道: Cap 0

通用IO口: Input 0

捕获轴

捕获轴: Axis 0

捕获方式

- ☒ 单次锁存
- ☐ 连续锁存

捕获逻辑

- ☐ 下降沿捕获
- ☒ 上升沿捕获
- ☐ 双边沿捕获

捕获位置源

- ☒ 指令源
- ☐ 编码器源

锁存对象 捕获轴 捕获通道 锁存位置

启动 停止 重启锁存器 获取捕获结果

提示: 捕获功能演示在启动捕获之后需要在轴

高速锁存界面在选择好锁存参数之后点击启动就能够根据信号的状态锁存位置值了，该演示界面并没有提供实时的锁存结果获取和显示，用户需要手动点击获取锁存结果来获取当前锁存到的位置值并显示在锁存结果表格中。

3.9 功能-PWM

PWM 输出控制界面如下图所示：

初始页 PMC2012-P-0-PWM

PWM

24V通道(对应output12-15) 0

5V通道 0

通道选择

- ☐ 5V
- ☒ 24V

频率(Hz) 100

占空比[0-1] 0.5

打开 关闭



选择不同的通道对应不同的通道参数，实际通道硬件接口请查看 PMC2000-P 相关信号的硬件手册。用户能够输入 PWM 的频率和占空比参数来达到自定义控制。“打开”和“关闭”对应着 PWM 的输出和关闭输出。



第4章 应用程序开发

在我司提供的 PMC2000-P 系列开发套件中提供有 Windows 系统下的动态链接库文件。在提供的开发套件中“动态库”文件夹下包含有 C 式头文件“MC_dll.h”和 C#语言下调用的头文件“JZMC.cs”,用户可以直接复制至项目中使用。

在“动态库”文件夹下根据文件夹命名分为 64 位系统下的动态库文件和 32 位系统下的动态库文件；用户需要根据具体软件运行的系统环境进行选择使用。

- ◆ 32bit 文件夹下为使用 32 位编译器生成的动态链接库文件“mcdll.dll”和“mcdll.lib”，32 位编译器编译的软件必须调用 32 位编译器生成的动态链接库；在 64 位系统下能够兼容运行 32 位编译器生成的软件。
- ◆ 64bit 文件夹下为使用 64 位编译器生成的动态链接库文件“mcdll.dll”和“mcdll.lib”，64 位编译器编译的软件必须调用 64 位编译器生成的动态链接库；在 32 位系统下无法兼容运行 64 位编译器生成的软件。

在进行应用程序开发之前，必须确保硬件和驱动程序的成功安装；使用开发套件中的动态链接库文件，用户可以使用任何能够支持动态链接库的开发工具来开发程序，下面分别以 Visual C++6.0 环境下 MFC 程序及 Visual Studio2022 环境下 WinForm 程序和普通 C++程序如何使用动态链接库进行简单讲解。



4.1 Visual C++6.0

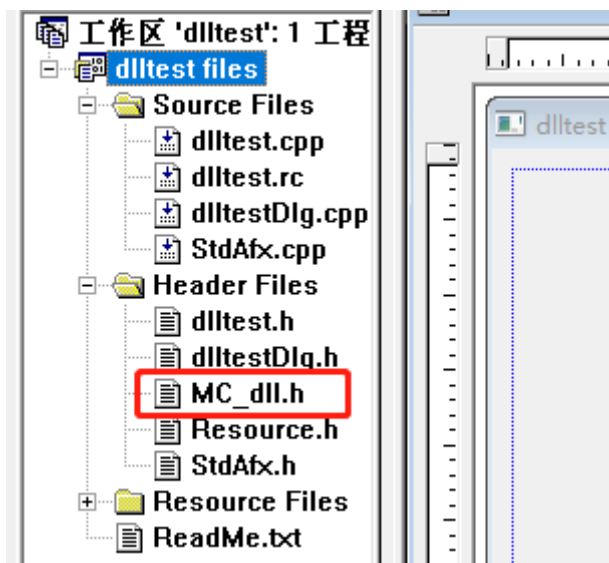
- 1) 启动 Visual C++6.0，新建一个 MFC 工程：



- 2) 根据项目平台不同选择 32 位或者 64 位的动态链接库文件，将"mcdll.dll"和"mcdll.lib"和 MC_dll.h 三个文件复制到工程文件夹中。

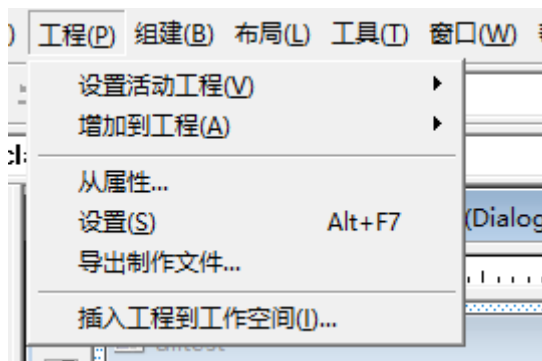
dlltestDlg.h	2023/3/20 15:04	C Header 源文件	2 KB
MC_dll.h	2023/3/6 9:42	C Header 源文件	79 KB
mcdll.dll	2023/3/3 17:28	应用程序扩展	400 KB
mcdll.lib	2023/3/3 17:28	Object File Library	33 KB
ReadMe.txt	2023/3/20 15:04	文本文档	4 KB

- 3) 在项目中右键项目"添加文件至工程"将 MC_dll.h 头文件添加至项目中。





- 4) 选择项目(Project)菜单下的设置(Settings)项。



- 5) 切换到连接(Link)标签页, 在对象/库模块(Object/library modules)栏中输入 lib 文件名 mcdll.lib。



- 6) 在应用程序文件中引入函数库头文件声明#include “MC_dll.h”。

至此, 用户就可以在程序中调用函数库中的函数进行应用开发了。

4.2 Visual Studio

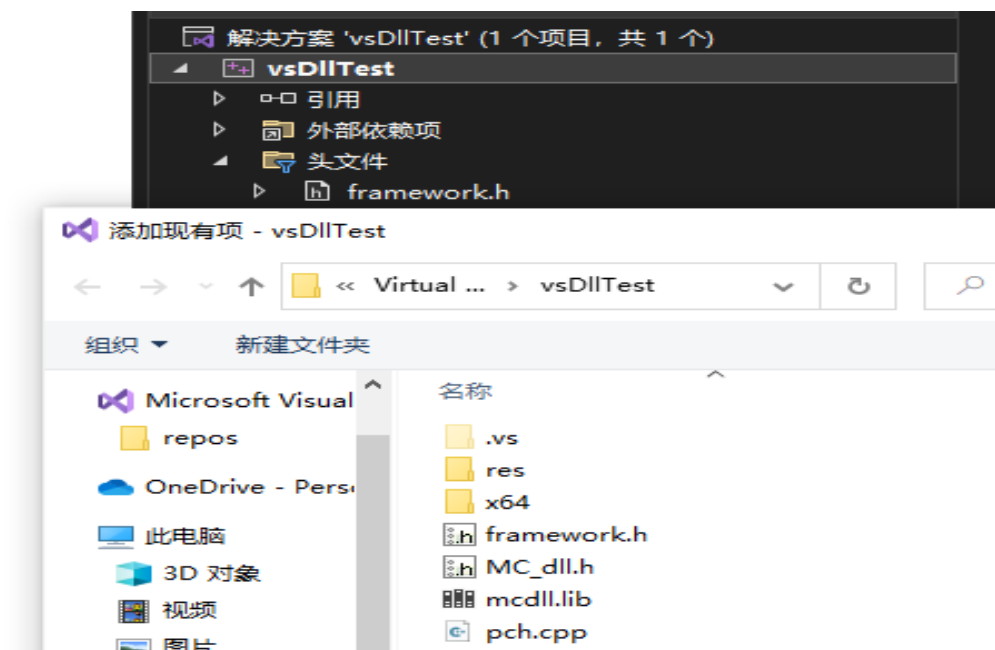
- 1) 启动 Visual Studio, 点击新建 MFC 项目或者其他 C++ 项目。
- 2) 根据应用程序的编译位数选择合适的 mcdll.lib 文件, 将 mcdll.lib 和



MC_dll.h 文件复制到项目路径下。

.vs	2023/3/20 15:45	文件夹	
res	2023/3/20 15:45	文件夹	
x64	2023/3/20 15:51	文件夹	
framework.h	2023/3/20 15:45	C Header 源文件	2 KB
MC_dll.h	2023/1/13 14:27	C Header 源文件	97 KB
mcdll.lib	2023/2/10 10:33	Object File Library	43 KB
pch.cpp	2023/3/20 15:45	C++ 源文件	1 KB
pch.h	2023/3/20 15:45	C Header 源文件	1 KB

- 3) 右键点击项目点击“添加现有项”将 MC_dll.h 头文件和 mcdll.lib 链接文件添加至项目中来。



- 4) 将 mcdll.dll 文件放至编译生成的.exe 文件路径下。

« x64 » Debug		在 Debug 中搜索
名称		修改日期
- Personal	vsDllTest.tlog	2023/3/20 16:11
	mcdll.dll	2023/2/10 10:33
	pch.obj	2023/3/20 15:51
	vc143.idb	2023/3/20 16:11
	vc143.pdb	2023/3/20 16:11
	vsDllTest.exe	2023/3/20 16:11
	vsDllTest.exe.recipe	2023/3/20 16:11
	vsDllTest.ilk	2023/3/20 16:11
	vsDllTest.log	2023/3/20 16:11
	vsDllTest.obj	2023/3/20 16:11
	vsDllTest.pch	2023/3/20 15:51

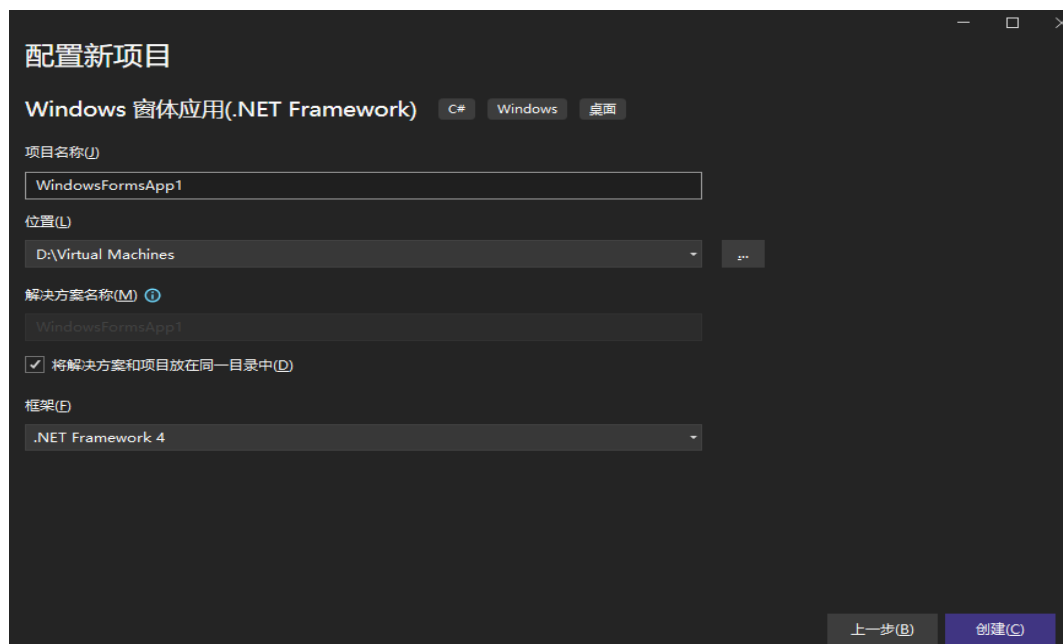


至此，用户可以调用函数库中的任何函数进行程序开发。

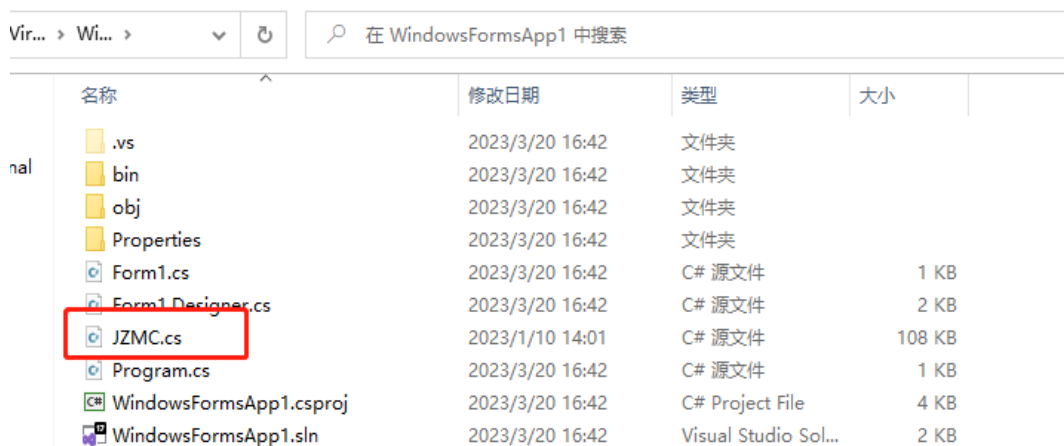
注意：必须将 mcdll.dll 文件放至执行程序.exe 文件路径下，否则执行程序会报缺失 xx.dll 文件的错误。

4.3 Visual C#

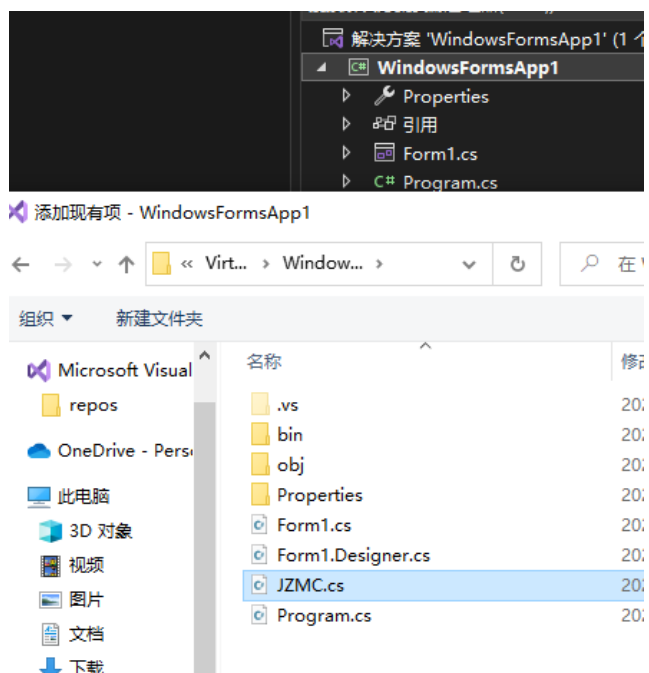
1) 启动 Visual Studio,选择建立 C#工程。



2) 将开发包中的 C#头文件"JZMC.cs"复制到项目的工程文件夹中。



3) 右键点击项目，选择添加现有项，将头文件添加至项目中来。



- 4) 根据程序开发的系统位数,将"mcdll.dll"动态库文件复制到程序编译生成的.exe 文件路径中。

bin > Debug		在 Debug 中搜索		
名称	修改日期	类型	大小	
mcdll.dll	2023/2/10 10:32	应用程序扩展	9,528 KB	
WindowsFormsApp1.exe	2023/3/20 16:51	应用程序	23 KB	
WindowsFormsApp1.pdb	2023/3/20 16:51	Program Debug...	32 KB	

至此,用户可以调用函数库中的任何函数进行程序开发。

注意: 必须将 mcdll.dll 文件放至执行程序.exe 文件路径下,否则执行程序会报缺失 xx.dll 文件的错误。



第5章 运动功能详解及实现

5.1 控制轴相关信号

- 1) 控制轴信号实现原理：根据配置的有效电平，在信号功能使能情况下，当输入电平与配置电平相同时，控制信号状态为有效。
- 2) PMC2000-P 系列针对每个脉冲轴提供多个专用信号，以 PMC2012-P 为例，具体功能如下：

PMC2012-P	轴号：0~7
控制轴的输入 IO	ALM(报警信号)
	EL+(硬件正限位信号)
	EL-(硬件负限位信号)
	ORG(原点信号)
	INP(伺服到信号)
	RDY(伺服准备好信号)
控制轴的输出 IO	SEVON(伺服使能)
	ERC(伺服报警清除)

PMC2012-P	轴号：8~11
控制轴的输入 IO	EL+(硬件正限位信号)
	EL-(硬件负限位信号)
	ORG(原点信号)



5.1.1 实现函数

PMC2000-P 系列出厂情况下未使能控制轴的专用信号，用户必须根据设备的硬件接线情况，设置专用信号的有效电平，相关函数如下表所示：

函数名称	功能	参考
MC_SetAxisInIoPrm	设置轴专用信号	函数库详解
MC_GetAxisInIoPrm	读取轴专用信号设置	
MC_SetAxisOutIo	设置轴专用输出口	
MC_GetAxisOutIo	读取轴专用输出口设置	

5.1.2 C#例程

```
ushort connect_no = 0; //卡号 0
ushort axis = 0; //轴号 0
ushort axis_io_type = 0; //专用信号类型，负限位
ushort enable = 1; //是否使能，使能
ushort io_logical = 0; //信号有效电平，低电平有效
ushort stopMode = 0; //信号触发轴停止模式，减速停止
MC_SetAxisInIoPrm(connect_no,axis,axis_io_type,enable,io_logical,stopMode
);
```

```
ushort connect_no = 0; //卡号 0
ushort axis = 0; //轴号 0
ushort axis_io_type = 0; //轴专用输出口类型，0 = 伺服使能
ushort io_level = 0; //输出电平，0 = 低电平
MC_SetAxisOutIo(connect_no,axis,axis_io_type,io_level); //设置轴 0 的伺服使
能输出低电平
```

5.2 坐标系当量配置

PMC2000-P 系列运动控制卡提供了脉冲当量设置功能，用户可以通过设置单轴的脉冲当量达到自定义坐标控制单位的目的。板卡出厂默认设置的脉冲当量



为 1，控制的单位为脉冲(Pulse)。

例如：某设备中运动控制卡执行点位运动目标 1000Pulse，其设备平台前进 1mm，用户可以通过设置脉冲当量功能将执行点位运动的目标位置单位设置为 mm，速度单位为 mm/s。将该轴的脉冲当量设置为 1000 时，执行点位运动指令的目标位置输入 1 则平台实际前进距离为 1mm。实际的轴脉冲输出为目标位置 * 当量设置。

5.2.1 实现函数

函数名称	功能	参考
MC_SetAxisEquvi	设置轴脉冲当量	函数库详解
MC_GetAxisEquvi	读取轴脉冲当量设置	

5.2.2 C#例程

```
ushort connect_no = 0; //卡号 0
ushort axisNo = 0; //轴号 0
double equvi = 1000; //轴当量设置 1000
MC_SetAxisEquvi(connect_no, axisNo, equvi); //设置 0 卡的轴 0 脉冲当量为 1000
```

5.3 软件限位

PMC2000-P 系列运动控制卡除了硬件限位外还可以通过设置软限位来限制轴的运动范围。

实现原理：

- 1) 通过设置轴的软限位位置和使能之后，当工作台的当前位置超越软限位时，该轴会根据设置的轴停止模式停止工作台的运动。



- 2) 当软限位触发以后，该轴无法再启动触发限位方向上的运动，同时该轴的软件限位为触发状态。此时该轴只能往限位相反的方向运动。

5.3.1 实现函数

控制卡出厂或者复位后软限位是默认禁止的，需要代码进行软限位的参数设置功能才能正常使用。相关函数如下表所示：

函数名称	功能	参考
MC_SetAxisSoftELPrm	设置轴软限位参数	函数库详解
MC_GetAxisSoftELPrm	获取轴软限位设置	
MC_GetAxisInloState	获取轴专用信号状态	

5.3.2 C#例程

```
ushort connect_no = 0; //卡号 0
ushort axisNo = 0; //轴号 0
ushort softEl_Enable = 1; //软限位使能
ushort posSrc = 0; //软限位的位置源，0 = 指令位置，1 = 反馈位置
double maxPos = 10000; //正限位
double minPos = -10000; //负限位
ushort stopMode = 0; //碰到限位时轴停止模式，0 = 减速停止，1 = 立即停止
//设置卡 0 轴 0 的软限位使能，软件正限位指令位置 10000，负限位指令位置-10000
//遇软件限位时减速停止
MC_SetAxisSoftELPrm(connect_no,axisNo, softEl_Enable, posSrc, maxPos, minPos,stopMode);

MC_Power(connect_no,axisNo, 1); //使能轴 0，执行运动前必须确保轴使能
//轴 0 以绝对运动模式运行至目标位置 10000
MC_MovePos(connect_no,axisNo,1,10000);
uint inloState = 0;
MC_GetAxisInloState(connect_no,axisNo, &inloState); //获取轴 0 的专用输入信号状态
//inloState 的 bit9 为软件负限位状态，该位为 1 表示有效
//inloState 的 bit10 为软件正限位状态，该位为 1 表示有效
```



5.4 轴状态获取

PMC2000-P 系列运动控制卡提供了多条指令来获取轴当前的状态，用户可以根据轴目前的状态来规划下一步的动作。

5.4.1 实现函数

函数名称	功能	参考
MC_GetAxisMachineState	判断轴运动状态	函数库详解
MC_GetAxisCheckDone	判断轴是否运动完成	
MC_GetAxisMotionMode	判断轴当前运动模式	
MC_GetAxisStopReason	获取轴停止原因	
MC_GetAxisCmdPos	获取轴当前指令位置	
MC_GetAxisCmdSpeed	获取轴当前指令速度	
MC_GetAxisFeedbackPos	获取轴当前反馈位置	
MC_GetHomeResult	获取轴回零结果	

5.4.2 C#例程

```
ushort connect_no = 0; //卡号 0
ushort axisNo = 0; //轴号 0
ushort axisState = 0; //用于获取状态
MC_GetAxisMachineState(connect_no, axisNo, out axisState);
/*
switch(axisState)
{
    case 0: 轴运动中
        break;
    case 1: 轴等待运动中
        break;
    case 2: 402 轴限位
        break;
```




```
        case 3: 轴暂停中
            break;
        case 4: 轴停止过程中
            break;
        case 5: 轴未使能
            break;
        case 6: 轴不存在
            break;
    }
    */
    ushort checkDone = 0; //用于获取装填
    MC_GetAxisCheckDone(connect_no, axisNo, out checkDone);
    /*
    switch(CheckDone)
    {
        case 0: 轴运行中
            break;
        case 1: 轴使能且空闲
            break;
    }
    */
    ushort motionMode = 0; //用于获取运动模式
    MC_GetAxisMotionMode(connect_no, axisNo, out motionMode);
    /*
    switch(motionMode)
    {
        case 0: 轴空闲
            break;
        case 1: 点位运动
            break;
        case 2: 速度运动
            break;
        case 3: 回零运动
            break;
        case 4: 手脉跟随运动
            break;
        case 129: 直线插补
            break;
        case 130: 圆弧插补
            break;
        case 131: 椭圆插补
            break;
```



```
}
*/

uint stopReason = 0;//用于获取停止原因
MC_GetAxisStopReason(connect_no,axisNo, out stopReason);
/*
switch(stopReason)
{
    case 0: 运动正常停止
    break;
    case 1: 调用指令减速停止
    break;
    case 2: Alarm 信号减速停止
    break;
    case 3: 硬件正限位减速停止
    break;
    case 4: 硬件负限位减速停止
    break;
    case 5: 软件正限位减速停止
    break;
    case 6: 软件负限位减速停止
    break;
    case 7: 区域限位引起停止运动
    break;
    case 8: IO 触发减速停止
    break;
    case 9: 减速暂停
    break;

    case 100: EMG 信号立即停止
    break;
    case 101: 调用指令立即停止
    break;
    case 102: Alarm 信号立即停止
    break;
    case 103: 硬件正限位立即停止
    break;
    case 104: 硬件负限位立即停止
    break;
    case 105: 软件正限位立即停止
    break;
    case 106: 软件负限位立即停止
```



```
        break;
    case 107: 区域限位引起停止运动
        break;
    case 108: IO 触发立即停止
        break;
    case 109: 运动中，掉使能
        break;
    case 110: 运动过大，异常停止
        break;
}
*/
double cmdPos = 0; //用于获取轴当前指令位置
MC_GetAxisCmdPos(connect_no, axisNo, out cmdPos); //获取当前轴指令位置
置

double cmdSpeed = 2; //用于获取轴当前指令速度
MC_GetAxisCmdSpeed(connect_no, axisNo, out cmdSpeed);

double feedbackPos = 0; //获取轴当前反馈位置
MC_GetAxisFeedbackPos(connect_no, axisNo, out feedbackPos);

ushort completeState = 0; //用于获取轴的回零结果
MC_GetHomeResult(connect_no, axisNo, out completeState);
/*
switch(completeState)
{
    case 0: 回零未完成
        break;
    case 1: 回零完成
        break;
    case 2: 回零出错，没有原点信号
        break;
    case 3: 回零中发现两个原点信号
        break;
    case 4: 回零中没有发现 EZ 信号
        break;
    case 5: 回零配置的方向错误
        break;
    case 6: 回零过程限位停止
        break;
    case 7: 回零外部停止
        break;
```



```
}  
*/
```

5.5 点位运动

点位运动是指从当前位置运动至指令下发的目标位置的一种运动模式。各轴可以独立设置目标位置、目标速度、加速时间、减速时间、起始速度和加减速模式等运动参数，能够独立运动和停止。

5.5.1 实现函数

- 1) 函数中距离或位置的单位为 Pulse，速度单位为 Pulse/s,加减速时间单位为 S.
- 2) 在设置单轴运动参数时，可以通过设置 S 段比值参数来控制电机是以梯形速度曲线或者 S 形速度曲线进行点位运动。

启动一段点位运动使用的相关函数如下表所示：

函数名称	功能	参考
MC_Power	轴使能，在使用轴之前需要先使能轴控制	函数库详解
MC_SetAxisMotionPrm	设置单轴运动参数	
MC_MovePos	启动一段点位运动	
MC_MoveAbsolute	以绝对位置启动一段点位运动，其中运动参数也在这一条指令下发	
MC_MoveRelative	以相对位置启动一段点位运动，其中运动参数也在这一条指令下发	



5.5.2 C#例程

```
ushort connect_no = 0; //卡号 0
ushort axis = 0; //轴号 0
MC_Power(connect_no,axis,1); //卡 0 的轴 0 使能

double startVel = 500; //启动速度, 500Pulse/s
double maxVel = 2000; //最大速度, 2000Pulse/s
double stopVel = 500; //停止速度, 500Pulse/s
double tAcc = 0.1; //加速时间, 0.1s
double tDec = 0.1; //减速时间, 0.1s
double sRatio = 0; //S 段比值, 0 表示 T 型加减速
double maxAcc = 1000000; //最大加减速限制
MC_SetAxisMotionPrm(connect_no,axis,startVel,maxVel,stopVel,tAcc,tDec,sRatio,maxAcc);

ushort pos_mode = 0; //位置模式, 0 = 相对位置模式
double pos = 10000; //运动距离, 10000Pulse
MC_MovePos(connect_no,axis,pos_mode,pos);
```

5.6 速度运动

速度运动是指各轴可以通过独立设置启动速度、运行速度、停止速度、加减速时间、加减速曲线等运动参数，使轴一直以运行速度运行，直到调用停止指令或者遇到限位信号才会停止的一种运动。

5.6.1 实现函数

在轴处于速度运动中时，可以通过调用 MC_MoveUpdateVel 函数实时改变当前轴的运行速度。启动速度运动指令使用的相关函数如下表所示：

函数名称	功能	参考
MC_Power	轴使能，在使用轴之前需	函数库详解



	要先使能轴控制	
MC_SetAxisMotionPrm	设置单轴运动参数	
MC_MoveVel	启动单轴速度运动	
MC_MoveVelocity	启动单轴速度运动，轴运动参数设置在同一条指令下发。	

5.6.2 C#例程

```
ushort connect_no = 0; //卡号 0
ushort axis = 0; //轴号 0
MC_Power(connect_no, axis, 1); //卡 0 的轴 0 使能

double startVel = 500; //启动速度, 500Pulse/s
double maxVel = 2000; //最大速度, 2000Pulse/s
double stopVel = 500; //停止速度, 500Pulse/s
double tAcc = 0.1; //加速时间, 0.1s
double tDec = 0.1; //减速时间, 0.1s
double sRatio = 0; //S 段比值, 0 表示 T 型加减速
double maxAcc = 1000000; //最大加减速限制
MC_SetAxisMotionPrm(connect_no, axis, startVel, maxVel, stopVel, tAcc, tDec, sRatio, maxAcc);

double vel = 1000; //运行速度, 1000Pulse/s, 大于 0 为正向运动
MC_MoveVel(connect_no, axis, vel);
//***
//***
//***

ushort stopMode = 0; //停止模式, 0 = 减速停止
double stop_time = 0.1; //减速停止的停止时间, 0.1s
MC_StopAxis(connect_no, axis, stopMode, stop_time);
```

5.7 回零运动

回零运动是帮助运动平台回到初始位置的一种运动模式。若运动平台上设有

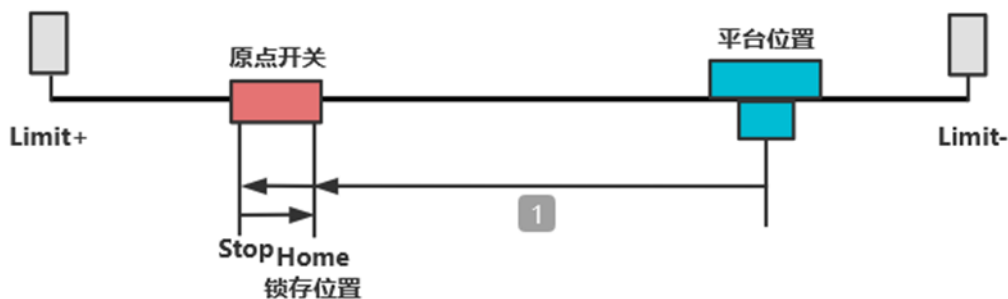


原点传感器，将原点传感器接入到对应轴的原点信号输入处。控制系统启动回零运动，平台搜寻原点开关的过程即为回零运动。

PMC2000-P 系列提供了 9 中回零模式，适应不同场合的回零需求：

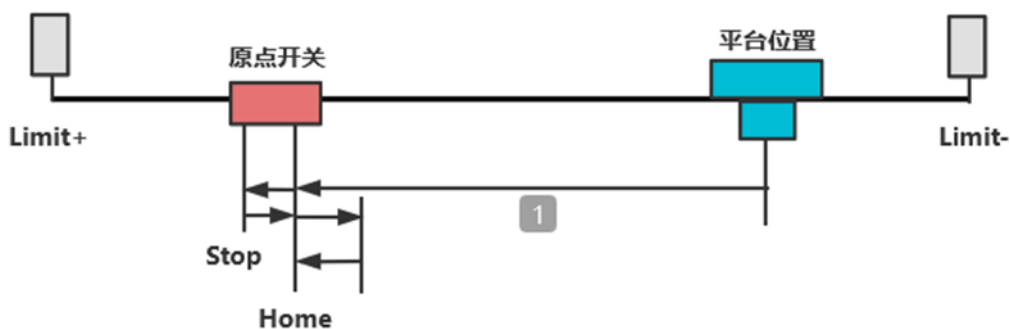
1) 模式 1：一次捕获 ORG 信号回零

电机先以设定速度回原点，当原点开关边沿触发时，将当前位置锁存下来，同时电机减速停止。电机减速停止完成后再反向回找锁存位置，运动到锁存位置，电机停止。回零过程如下图所示。



2) 模式 2：高速运动+反向一次捕获 ORG 信号回零

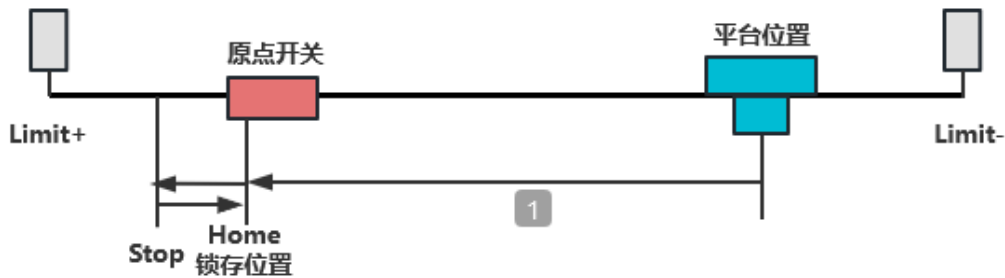
该方式以设定速度将平台运行到原点开关，当原点开关边沿触发时电机减速停止，然后再反向找原点开关边缘附近，当原点开关第一次无效时将当前位置锁存下来，电机减速停止完成后再反向回找锁存位置，运动到锁存位置，电机停止。回零过程如下图所示。



3) 模式 3：高速运动+同向低速捕获 ORG 信号回零



该方式先以设定速度将平台运动至原点开关附近，当原点开关边沿触发时将速度降至低速运行，当原点信号第一次无效时将该位置锁存，电机减速停止完成后反向回找锁存位置。回零过程如下图所示。



4) 模式 4: ORG+反向找 EZ 捕获回零

该方式在回原点运动过程中，当找到原点信号后，减速停止，然后以反找速度反向找到 EZ 生效，此时电机停止，将停止位置设为原点位置，回零过程如下图所示。



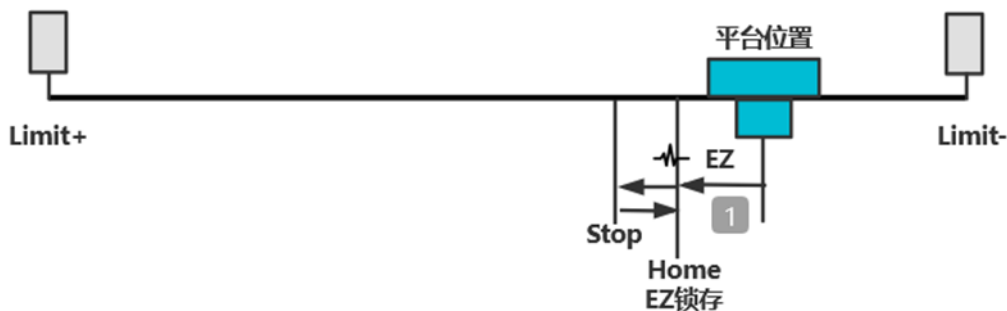
5) 模式 5: ORG+运动方向 EZ 捕获回零

该方式在回原点运动过程中，当找到原点信号后，还要等待该轴的 EZ 信号出现，锁存 EZ 信号出现的位置，电机减速停止完成后回到锁存位置，将停止位置设为原点位置，回零过程如下图所示。



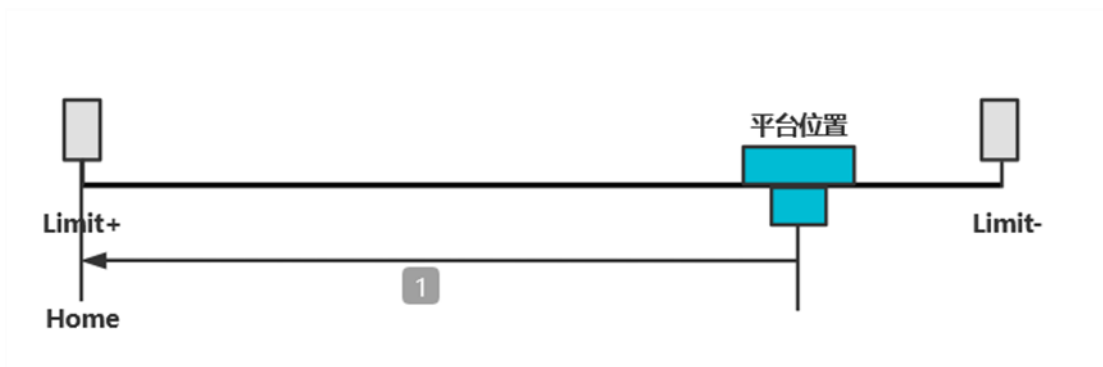
6) 模式 6：EZ 捕获回零

该方式在回原点运动过程中，当检测到该轴的 EZ 信号出现一次后，锁存 EZ 信号出现的位置，然后电机减速停止，电机减速停止完成后再回到锁存位置，将停止位置设为原点位置。回零过程如下图所示。



7) 模式 7：单次限位回零

该方式电机从初始位置以设定速度向限位方向运动，运动方向决定搜寻的限位开关，当到达限位开关位置，限位信号被触发，电机立即停止；将停止位置设



为原点位置。回零过程如下图所示。

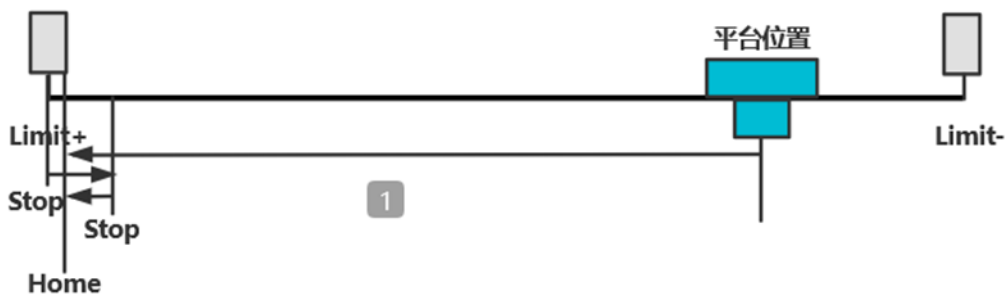
8) 模式 8：两次限位回零

该方式先进行回零模式 7 运动，完成后再反向回找限位开关的边缘位置，当限位信号第一次无效的时候，电机立即停止；将停止位置设为原点位置。回零过程如下图所示。



9) 模式 9：多次限位回零

该方式先进行回零模式 7 运动，电机从初始位置以设定速度向限位方向运动，当达到限位开关时，限位信号被触发，电机立即停止，停止后再反向回找限位开关的边缘位置，当找到限位开关边缘位置后电机减速停止，停止后再执行回零模式 7 运动，当到达限位开关位置，限位信号被触发，电机立即停止；将停止位置设为原点位置。回零过程如下图所示。





5.7.1 实现函数

回零运动的相关函数如下表所示：

函数名称	功能	参考
MC_Power	轴使能，在使用轴之前需要先使能轴控制	函数库详解
MC_Home	启动回零运动	
MC_GetHomePrm	获取回零参数	
MC_GetHomeResult	获取回零结果	
MC_GetAxisHomeState	获取原点信号输入状态	

5.7.2 C#例程

```
ushort connect_no = 0; //卡号 0
ushort axisNo = 0; //轴号 0
ushort posObject = 0; //针对捕获回零时捕获的位置源，0 = 指令位置，1 = 反馈位置
ushort homeMode = 1; //回零模式 1，一次捕获 ORG 信号回零
int homeDir = 1; //回零方向，1 = 正方向，-1 = 负方向
ushort homeLogic = 0; //回零逻辑电平，0 表示低电平有效，1 表示高电平有效
ushort ezCount = 1; //EZ 回零的 EZ 信号计数个数，最小值为 1
ushort offsetMode = 0; //保留参数
ushort offsetPos = 0; //保留参数

//在轴执行运动前，确保该轴使能
MC_Power(connect_no, axisNo, 1);

//启动轴 0 回零运动
MC_Home(connect_no, axisNo, posObject, homeMode,
homeDir, homeLogic, ezCount, offsetMode, offsetPos);

byte homeState = 0; //获取原点信号状态
MC_GetAxisHomeState(connect_no, axisNo, out homeState);

ushort completeState = 0; //获取回零结果
```



```
MC_GetHomeResult(connect_no,axisNo,out completeState);
```

5.8 直线插补

直线插补能够保证起点和终点位置的准确，同时插补轴的脉冲是按照直线斜率成比例发出的。

5.8.1 实现函数

PMC2000-P 系列提供 4 个插补系，每个插补系能够独立运动。直线插补的计算由控制卡的硬件执行，用户只需要将插补轴数、速度、加减速、终点位置等参数写入相关函数即可。相关函数如下表所示：

函数名称	功能	参考
MC_GetCrdCheckDone	判断插补系是否在运动中	函数库详解
MC_GetCrdCmdSpeed	读取当前插补器速度	
MC_SetCrdVelSmooth	设置插补器速度平滑	
MC_GetCrdVelSmooth	读取插补器速度平滑设置	
MC_SetCrdMotionPrm	设置插补器运动参数	
MC_GetCrdMotionPrm	读取插补器运动参数	
MC_MoveLinear	执行直线插补运动	

5.8.2 C#例程

```
ushort connect_no = 0; //卡号 0
ushort crd = 0; //插补系
for(int _axis = 0; _axis < 12; _axis++)
{
```



```
MC_Power(connect_no,_axis,1); //在轴运动之前确保轴已使能
}

//设置插补系的运动参数
MC_SetCrdMotionPrm(connect_no,
                    crd,
                    100, //起始速度
                    1000, //最大速度
                    100, //停止速度
                    0.1, //加速时间
                    0.1, //减速时间
                    0, //S 段时间，0 为 T 型
                    1000000); //最大加减速限制

ushort axisNum = 2; //轴数，2 轴直线插补
ushort [2]axisList = new ushort[]{0,1}; //轴 0 和轴 1 做直线插补运动
double [2]targetPos = new double[]{10000,5000}; //轴 0 和轴 1 的终点位置
ushort crdMode = 0; //运动模式，0 = 相对运动模式，1 = 绝对运动模式

ushort crdState = 0;
MC_GetCrdCheckDone(connect_no, crd, &crdState); //获取插补器的状态
if(crdState == 1) //如果处于使能并空闲状态
{
    //启动两轴直线插补
    MC_MoveLinear(connect_no,
                  crd,
                  axisNum,
                  axisList,
                  targetPos,
                  crdMode);
}
```

5.9 圆弧插补

PMC2000-P 系列运动控制卡提供有 3 种圆弧插补指令：

- 1) 终点圆心的圆弧插补函数：只需要根据起点，给定圆心位置和终点位置就能够计算出一个圆弧，如果插补轴数超过 3 轴的话其他轴作为跟随轴。
- 2) 终点半径圆弧插补函数：需要根据起点，目标位置和半径来确定一段圆



弧。

- 3) 三点圆弧插补函数：需要提供起点，目标位置和圆弧上的任意一点；程序会根据这三点来计算出一段圆弧。

5.9.1 实现函数

圆弧插补使用的相关函数如下表所示：

函数名称	功能	参考
MC_GetCrdCheckDone	判断插补器是否在运动中	函数库详解
MC_GetCrdCmdSpeed	获取插补器的指令速度	
MC_SetCrdVelSmooth	设置插补器速度平滑	
MC_GetCrdVelSmooth	获取插补器速度平滑设置	
MC_SetCrdMotionPrm	设置插补器的运动参数	
MC_GetCrdMotionPrm	获取插补器的运动参数	
MC_MovePlaneArcC	终点圆心的圆弧插补指令	
MC_MovePlaneArcR	终点半径的圆弧插补指令	
MC_MovePlaneArc3P	三点圆弧插补指令	

5.9.2 C#例程

```
ushort connect_no = 0; //卡号 0
ushort crd = 0; //插补系
for(int _axis = 0; _axis < 12; _axis++)
{
    MC_Power(connect_no, _axis, 1); //在轴运动之前确保轴已使能
}
```



//设置插补系的运动参数

```
MC_SetCrdMotionPrm(connect_no,
                    crd,
                    100, //起始速度
                    1000, //最大速度
                    100, //停止速度
                    0.1, //加速时间
                    0.1, //减速时间
                    0, //S 段时间，0 为 T 型
                    1000000); //最大加减速限制

ushort axisNum = 2; //轴数，2 轴直线插补
ushort [2]axisList = new ushort[]{0,1}; //轴 0 和轴 1 做直线插补运动
double [2]targetPos = new double[]{10000,0}; //轴 0 和轴 1 的终点位置
double [2]cenPos = new double[]{5000,0}; //圆心位置
ushort arcDir = 0; //圆弧插补方向，0 = 顺时针，1 = 逆时针
uint cirCle = 1; //插补圈数
ushort crdMode = 0; //运动模式，0 = 相对运动模式，1 = 绝对运动模式
```

```
ushort crdState = 0;
```

```
MC_GetCrdCheckDone(connect_no, crd, &crdState); //获取插补器的状态
```

```
if(crdState == 1) //如果处于使能并空闲状态
```

```
{
```

```
//终点圆心的圆弧插补
```

```
MC_MovePlaneArcC(connect_no,
                  crd,
                  axisNum,
                  axisList,
                  targetPos,
                  cenPos,
                  arcDir,
                  cirCle,
                  crdMode);
```

```
}
```

```
//.....
```

```
MC_GetCrdCheckDone(connect_no, crd, &crdState); //获取插补器的状态
```

```
if(crdState == 1) //如果处于使能并空闲状态
```

```
{
```

```
double arcRadius = 5000; //圆弧半径
```

```
//终点半径的圆弧插补
```



```
MC_MovePlaneArcR(connect_no,
                  crd,
                  axisNum,
                  axisList,
                  targetPos,
                  arcRadius,
                  arcDir,
                  cirCle,
                  crdMode);
}

//.....

MC_GetCrdCheckDone(connect_no, crd, &crdState); //获取插补器的状态
if(crdState == 1) //如果处于使能并空闲状态
{
double [2]midPos = new double[]{5000,5000}; //圆弧中点

//三点圆弧插补
MC_MovePlaneArc3P(connect_no,
                  crd,
                  axisNum,
                  axisList,
                  targetPos,
                  midPos,
                  cirCle,
                  crdMode);
}
```

5.10 空间圆弧插补

PMC2000-P 系列运动控制卡提供了两种空间型圆弧插补功能。在空间圆弧插补中，前两轴(XY 轴)作平面圆弧插补功能，第三轴(Z 轴)沿 Z 轴方向运动到指定高度。

当插补轴数大于 3 时，轴列表的前三轴进行空间圆弧插补的同时，后续轴做



线性跟随运动；跟随轴与作插补运动的轴同时运动、同时停止。

5.10.1 实现函数

插补的速度计算由控制卡硬件执行，用户只需要将插补运动的速度曲线参数及运动参数写入相关函数即可。空间圆弧插补使用的相关函数如下表所示：

函数名称	功能	参考
MC_GetCrdCheckDone	判断插补器是否在运动中	函数库详解
MC_GetCrdCmdSpeed	获取插补器的指令速度	
MC_SetCrdVelSmooth	设置插补器速度平滑	
MC_GetCrdVelSmooth	获取插补器速度平滑设置	
MC_SetCrdMotionPrm	设置插补器运动参数	
MC_GetCrdMotionPrm	获取插补器运动参数	
MC_MoveSpaceArcC	终点圆心的空间圆弧插补指令	
MC_MoveSpaceArc3P	三点圆弧空间插补指令	
MC_GetCrdStatus	获取插补器的工作状态	

5.10.2 C#例程

```
ushort connect_no = 0; //卡号 0
ushort crd = 0; //插补系
for(int _axis = 0; _axis < 12; _axis++)
{
    MC_Power(connect_no, _axis, 1); //在轴运动之前确保轴已使能
}

//设置插补系的运动参数
MC_SetCrdMotionPrm(connect_no,
```



```
        crd,
        100, //起始速度
        1000, //最大速度
        100, //停止速度
        0.1, //加速时间
        0.1, //减速时间
        0, //S 段时间, 0 为 T 型
        1000000); //最大加减速限制

ushort axisNum = 3; //轴数, 3 轴空间插补
ushort [3]axisList = new ushort[]{0,1,2}; //3 轴作空间插补运动
double [3]targetPos = new double[]{10000,0,10000}; //目标位置
double [3]cenPos = new double[]{5000,0,5000}; //圆心位置
ushort arcDir = 0; //圆弧插补方向, 0 = 顺时针, 1 = 逆时针
uint cirCle = 1; //插补圈数
ushort crdMode = 0; //运动模式, 0 = 相对运动模式, 1 = 绝对运动模式

ushort crdState = 0;
MC_GetCrdCheckDone(connect_no, crd, &crdState); //获取插补器的状态
if(crdState == 1) //如果处于使能并空闲状态
{
    //终点圆心的空间圆弧插补
    MC_MoveSpaceArcC(connect_no,
        crd,
        axisNum,
        axisList,
        targetPos,
        cenPos,
        arcDir,
        cirCle,
        crdMode);
}

//.....

MC_GetCrdCheckDone(connect_no, crd, &crdState); //获取插补器的状态
if(crdState == 1) //如果处于使能并空闲状态
{
    double [3]midPos = new double[]{5000,5000,5000}; //圆弧中点
    //三点圆弧空间插补
    MC_MoveSpaceArc3P(connect_no,
        crd,
        axisNum,
```



```
axisList,  
targetPos,  
midPos,  
cirCle,  
crdMode);  
  
}  
  
//.....
```

5.11 椭圆插补

PMC2000-P 系列运动控制卡提供椭圆插补功能，椭圆插补指令实现平面椭圆插补；

5.11.1 实现函数

执行椭圆插补指令时，用户需设置椭圆圆心坐标、椭圆长轴、椭圆短轴以确定椭圆在坐标系中的数学表达，同时设置插补轴的目标位置、椭圆插补方向以确定插补动作。椭圆插补使用的相关函数如下表所示：

函数名称	功能	参考
MC_GetCrdCheckDone	判断插补器是否在运动中	函数库详解
MC_GetCrdCmdSpeed	获取插补器的指令速度	
MC_SetCrdVelSmooth	设置插补器速度平滑	
MC_GetCrdVelSmooth	获取插补器速度平滑设置	
MC_SetCrdMotionPrm	设置插补器运动参数	
MC_GetCrdMotionPrm	获取插补器运动参数	
MC_MovePlaneEllipse	椭圆插补指令	
MC_GetCrdStatus	获取插补器的工作状态	



5.11.2 C#例程

```
ushort connect_no = 0; //卡号 0
ushort crd = 0; //插补系
for(int _axis = 0; _axis < 12; _axis++)
{
    MC_Power(connect_no, _axis, 1); //在轴运动之前确保轴已使能
}

//设置插补系的运动参数
MC_SetCrdMotionPrm(connect_no,
                    crd,
                    100, //起始速度
                    1000, //最大速度
                    100, //停止速度
                    0.1, //加速时间
                    0.1, //减速时间
                    0, //S 段时间, 0 为 T 型
                    1000000); //最大加减速限制

ushort axisNum = 2; //轴数, 2 轴作空间椭圆插补
ushort [2]axisList = new ushort[]{0, 1}; //2 轴作平面椭圆插补运动
double [2]targetPos = new double[]{0, 0}; //目标位置
double [2]cenPos = new double[]{5000, 0}; //椭圆圆心位置
double shortLen = 5000; //椭圆短轴
double longLen = 10000; //椭圆长轴
ushort arcDir = 0; //圆弧插补方向, 0 = 顺时针, 1 = 逆时针
uint cirCle = 1; //插补圈数
ushort crdMode = 0; //运动模式, 0 = 相对运动模式, 1 = 绝对运动模式

ushort crdState = 0;
MC_GetCrdCheckDone(connect_no, crd, &crdState); //获取插补器的状态
if(crdState == 1) //如果处于使能并空闲状态
{
    //椭圆插补
    MC_MovePlaneEllipse(connect_no,
                        crd,
                        axisNum,
                        axisList,
                        targetPos,
```



```
        cenPos,  
        shortLen,  
        longLen,  
        arcDir,  
        cirCle,  
        crdMode);  
    }
```

5.12 通用 IO 控制

用户可以使用 PMC2000-P 系列板卡提供的数字输入口用于检测开关信号、传感器信号等输入信号, 或者使用输出口控制继电器、电磁阀等输出设备的信号。

型号	通用输入口数量	通用输出口数量
PMC2012-P	16	16(简牛座为 24 个)

5.12.1 实现函数

函数名称	功能	参考
MC_GetInloBit	按位读取控制卡的某一位输入口的电平状态	函数库详解
MC_GetInloPort	一起读取 32 位输入口的电平状态	
MC_GetInloPtr	按照指定的起始索引和读取位数读取输入口的电平状态	
MC_SetOutloBit	按位设置通用输出口的电平状态	
MC_SetOutloPort	一次设置 32 位通用输出口的电平状态	
MC_SetOutloPtr	按照指定的起始索引和设置	



	位数设置通用输出口的电平状态	
MC_GetOutloBit	按位获取通用输出口的电平状态	
MC_GetOutloPort	一次获取 32 位通用输出口的电平状态	
MC_GetOutloPtr	按照指定的起始索引和读取位数读取通用输出口的电平状态	

注 意：在使用 MC_SetOutloPort 或 MC_GetOutloPort 对运动控制卡的输出口进行置位，使 MC_GetOutloPort、MC_GetOutloPtr、MC_GetInloPort、MC_GetInloPtr 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）。在对 IO 电平进行控制与读取时，使用十六进制数赋值远比使用十进制数赋值更加直观、方便。

通用输入输出的函数参数对应 IO 口关系如下表所示：

型号	按 bit 操作		按 port 操作	
PMC2012-P			Port_Index	实际 IO 口
	lo_Index	实际 IO 口	0	Bit0~Bit15
	0~15	0~15		对应 IO 口 0~15



5.12.2 C#例程

```
//对通用输入口进行操作，获取通用输入口 0 的电平状态
ushort connect_no = 0; //卡号 0

uint ioIndex = 0; //通用输入 0
byte ioState = 0; //用于接收实际状态

//方式 1，按位读取，通用输入 0 的实际状态
MC_GetInIoBit(connect_no, ioIndex, &ioState);
if(ioState == 0){
    //低电平
}
else{
    //高电平
}

uint portIndex = 0; //端口索引(一次读取 32 位，索引为需要读取的输入索引
/32)
uint ioState_32 = 0;
//方式 2，一次获取 32 位通用输入的实际状态
MC_GetInIoPort(connect_no, portIndex, &ioState_32);

bool input0State = false; //低电平
if((ioState_32 & (0x1 << 0)) == 0) input0State = false; //获取 32 位数中的 Bit0
状态，就是 Input0 的状态
else input0State = true; //高电平

uint startIndex = 0; //需要读取的输入口起始索引
uint readNum = 1; //需要读取的输入口数量(最多一次读取 64bit)
ulong ioState_64 = 0; //获取读取的输入口状态，最多 64bit
//方式 3，按照输入的起始索引和读取数量读取输入口状态
MC_GetInIoPtr(connect_no, startIndex, readNum, &ioState_64);
if((ioState_64 & ((ulong)(0x1 << 0))) == 0) input0State = false; //获取 32 位数
中的 Bit0 状态，就是 Input0 的状态
else input0State = true; //高电平

//对通用输出口进行操作。
ushort connect_no = 0; //卡号 0
```



```
uint ioIndex = 0; //通用输出 0
byte ioState = 0; //用于接收实际状态

//方式 1，按位读取设置通用输出 0 的实际状态
MC_GetOutloBit(connect_no, ioIndex, &ioState);
if(ioState == 0){
    //实际状态为低电平
    //手动使通用输出 0 输出高电平
    MC_SetOutloBit(connect_no, ioIndex, 1);
}
else{
    //实际状态为高电平
    //手动使通用输出 0 输出低电平
    MC_SetOutloBit(connect_no, ioIndex, 0);
}

uint portIndex = 0; //端口索引(一次读取 32 位，索引为需要读取的输入索引
/32)
uint ioState_32 = 0;
//方式 2，一次操作 32 位通用输出口的实际状态
MC_GetOutloPort(connect_no, portIndex, &ioState_32);
if((ioState_32 & (0x1 << 0)) == 0){
    //获取 32 位数中的 Bit0 状态，Input0 的实际状态为低电平
    //这里使用 MC_SetOutloPort 将 32 位输出口全部置为高电平
    //也可以单独对 ioState_32 的某位进行操作再写入
    MC_SetOutloPort(connect_no, portIndex, 0xFFFFFFFF);
}
else{
    //获取 32 位数中的 Bit0 状态，Input0 的实际状态为高电平
    //这里使用 MC_SetOutloPort 将 32 位输出口全部置为低电平
    //也可以单独对 ioState_32 的某位进行操作再写入
    MC_SetOutloPort(connect_no, portIndex, 0x0);
}

uint startIndex = 0; //需要读取的输出口起始索引
uint readNum = 1; //需要读取的输出口数量(最多一次读取 64bit)
ulong ioState_64 = 0; //获取读取的输出口状态，最多 64bit
//方式 3，按照输入的起始索引和读取数量读取输出口状态
MC_GetOutloPort(connect_no, startIndex, &ioState_64);
if((ioState_64 & ((ulong)(0x1 << 0))) == 0){
    //获取到 bit0 的实际电平为低电平，bit0 就是起始索引输出口的电平状
```

态



```
//将 bit0 的实际电平输出置为高电平
//其中 IoMask 参数 bit 位为零表示保持原来状态,为 1 表示写入 IoState
状态
MC_SetOutIoPtr(connect_no,startIndex,1 ,(ulong)0x1);
}
else{
//获取到 bit0 的实际电平为高电平, bit0 就是起始索引输出口的电平状
态
//将 bit0 的实际电平输出置为低电平
//其中 IoMask 参数 bit 位为零表示保持原来状态,为 1 表示写入 IoState
状态
MC_SetOutIoPtr(connect_no,startIndex,0 ,(ulong)0x1);
}
```

5.13 虚拟 IO 复用

PMC2000-P 系列提供了虚拟 IO 复用功能,该功能允许用户对虚拟 IO 口的硬件输入接口进行任意配置。

型号	虚拟 IO 口数量
PMC2012-P	32

5.13.1 实现函数

函数名称	功能	参考
MC_SetReuselIoMap	设置虚拟 IO 映射关系	函数库详解
MC_GetReuselIoMap	获取虚拟 IO 映射关系	
MC_GetReuseInIoBit	获取虚拟 IO 口的状态	



5.13.2 C#例程

```
ushort connect_no = 0; //卡号 0
ushort ioIndex = 0; //虚拟 IO 的索引号, (范围为 0~31)
ushort mapIoType = 0; //映射的实际 IO 信号类型
/*
typedef enum
{
    NEL_INDEX      = 0, //负限位
    PEL_INDEX      = 1, //正限位
    ORG_INDEX      = 2, //原点
    INP_INDEX      = 3, //伺服到位
    ALM_INDEX      = 5, //伺服报警
    RDY_INDEX      = 6, //伺服准备
    NORMAL_IO_TYPE = 9, //通用 IO 信号
} IN_TYPE_INDEX;
*/

uint mapindex = 0; //如果 mapIoType 为轴专用信号时, mapindex 为轴号;
否则为通用 IO 索引。
MC_SetReuselIoMap(connect_no, ioIndex, mapIoType, mapindex);

byte ioState = 0; //用来获取虚拟 IO 的状态
MC_GetReuselIoBit(connect_no, ioIndex, &ioState);
```

5.14 轴专用信号映射

PMC2000-P 系列运动控制卡提供了轴专用 IO 映射功能, 该功能允许用户对轴的专用 IO 信号的硬件输入接口进行任意配置。

例如:

- 1) 将所有轴的 EL-信号通过映射指令映射至通用输入口 0, 这样当通用输入口 0 触发时所有处于运动中的轴就相当于硬件限位触发。
- 2) 可以将轴 0 的报警信号映射到其他轴的报警信号, 当其他轴的报警信号



触发时亦相当于轴 0 的报警信号触发。

5.14.1 实现函数

函数名称	功能	参考
MC_SetAxisInIoMap	设置轴专用信号映射	函数库详解
MC_GetAxisInIoMap	获取轴专用信号映射	
MC_GetAxisInIoState	获取轴专用信号状态	

5.14.2 C#例程

```
//轴专用信号映射。(将所有轴的 EL-信号映射至通用输入 0)
ushort connect_no = 0; //卡号 0

for(ushort _axisNo = 0; _axisNo < 12; _axisNo++) //PMC2012-P, 设置 12 个轴
{
    //轴的专用信号在使用前需要进行配置
    //这里将轴的 EL-信号使能、有效电平为低电平有效、触发后轴的停止
    模式为立即停止
    MC_SetAxisInIoPrm(connect_no, _axisNo, 0, 1, 0, 1);
}

//将所有轴的 EL-信号映射至通用输入 0
for(ushort _axisNo = 0; _axisNo < 12; _axisNo++)
{
    //轴专用信号类型
    /*
        NEL_INDEX    = 0, //负限位
        PEL_INDEX    = 1, //正限位
        ORG_INDEX    = 2, //原点
        INP_INDEX    = 3, //伺服到位
        ALM_INDEX    = 5, //伺服报警
        RDY_INDEX    = 6, //伺服准备
    */
    ushort axisIoType = 0; //负限位
```



```
//映射信号类型
/*
    NEL_INDEX      = 0, //负限位
    PEL_INDEX      = 1, //正限位
    ORG_INDEX       = 2, //原点
    INP_INDEX       = 3, //伺服到位
    ALM_INDEX       = 5, //伺服报警
    RDY_INDEX       = 6, //伺服准备
    NORMAL_IO_TYPE  = 9, //通用 IO 信号
*/
ushort mapIoType = 9; //通用输入

//映射索引
//当 mapIoType = 9 为通用输入的索引号
//当 mapIoType 等于其他时为轴号
ushort mapIndex = 0;
```

```
MC_SetAxisInIoMap(connect_no, _axisNo, axisIoType, mapIoType, mapIndex);
}
//...
```

//映射完成后，通用输入 0 触发相当于所有轴的 EL-触发

5.15 手轮

PMC2000-P 系列运动控制卡具有一个辅助编码器通道，支持单轴手轮运动。

手轮运动多用于机器平台的视校，比如加工轨迹起点调整、刀具对位等场景。

5.15.1 实现函数

使用手轮功能需要设置辅助编码器的计数模式，同时输入手轮倍率启动运动。

使用的相关函数如下表所示：

函数名称	功能	参考
MC_SetHandleWheelPrm	手轮参数配置	函数库详解



MC_GetHandleWheelPrm	获取手轮参数配置	
MC_StartHandleWheel	启动手轮运动	
MC_SetAuxEncoderPrm	设置辅助编码器计数参数	
MC_GetAuxEncoderPrm	获取辅助编码器计数参数	
MC_SetAuxEncoderPositon	设置辅助编码器值	
MC_GetAuxEncoderPositon	获取辅助编码器值	

5.15.2 C#例程

//启动单轴手轮运动

ushort connect_no = 0; //卡号 0

ushort channel = 0; //辅助编码器通道，RT200C 系列只有 1 个辅助编码器通道

/*

Pulse_Dir_Mode_Count = 0, //脉冲方向计数

Pulse_A_B_Mode_Count_1 = 1, //AB 相计数，1 倍频计数

Pulse_A_B_Mode_Count_2 = 2, //AB 相计数，2 倍频计数

Pulse_A_B_Mode_Count_4 = 3 //AB 相计数，4 倍频计数

*/

ushort mode = 0; //辅助编码器计数模式

ushort dir = 0; //计数方向， 0 = 方向不取反，1 = 方向取反

MC_SetAuxEncoderPrm(connect_no, channel, mode, dir); //设置辅助编码器计数模式

MC_SetAuxEncoderPositon(connect_no, channel, 0); //辅助编码器值清零

ushort axis = 0; //轴号

double multi = 10; //手轮倍率输出脉冲，负值为反向

MC_SetHandleWheelPrm(connect_no, axis, channel, multi); //设置手轮参数



```
MC_Power(connect_no,axis,1); //在轴运动之前确保轴已使能
```

```
MC_StartHandleWheel(connect_no,axis); //启动轴 0 手轮运动
```

```
//...需要停止运动调用轴停止指令
```

5.16 一维软件比较

PMC2000-P 系列提供一维软件比较功能，可用于将指令位置或者反馈位置与设定的比较点位置进行比较，当反馈位置或指令位置达到比较点位置时，自动执行配置的比较点功能，操作 IO 口或者停止相关轴。

5.16.1 实现函数

一维软件比较的相关函数如下表所示：

函数名称	功能	参考
MC_GetSoftCmpPrm	获取软件比较配置参数	函数库详解
MC_GetSoftCmpStatus	获取软件比较状态信息	
MC_ClearSoftCmp	清除所有配置的软件比较点	
MC_StartSoft1DCmp	启动一维软件比较	
MC_StopSoftCmp	停止软件比较	

注意：使用 MC_StartSoft1DCmp 函数启动一维软件比较也相当于添加了一个比较点，可以添加多个比较点执行不同的操作，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的，即如果有一个比较点没有被触发比较动作，那么后面的比较点是不会起作用的。



5.16.2 C#例程

```
//一维软件比较
ushort connect_no = 0;//卡号 0
ushort cmpChan = 0;//比较通道，取值范围[0~3]
ushort cmpAxis = 0;//比较轴号，轴 0
ushort posSrc = 0;//比较的位置源，0 = 指令位置，1 = 反馈位置
ushort cmpCounts = 0;//保留参数
double [1]cmpPos = new double[]{10000}; //比较点位置
ushort cmpMode = 1;//比较模式
/*
    Less_Compare_Value = 0, //小于比较点模式
    Beyond_Compare_Value = 1, //大于比较点模式
    Linear_Compare_Value = 2, //等增量比较模式
    FIFO_Compare_Value = 3    //FIFO 点比较输出
*/

ushort triggerMode = 0;//触发模式
/*
    Trigger_IO_Low = 0, 输出口输出低
    Trigger_IO_High = 1, 输出口输出高
    Trigger_IO_Reverse = 2, 输出口电平翻转
    Trigger_IO_Pulse = 3, 输出脉冲
    Trigger_Axis_Imd_Stop = 4, 停止轴
*/

uint triggerPrm = 0;//触发参数
/*
    trigger_mode 为 0~3 时，该参数为 IO 序号；
    trigger_mode 为 4 时，该参数为轴序号；
*/

double outTime = 0;//输出时间，当 triggerMode = 3 时的输出脉冲时间，
单位是 S

//启动一维软件比较，相当于添加一个比较点
MC_StartSoft1DCmp(connect_no,
                   cmpChan,
                   cmpAxis,
                   posSrc,
                   cmpCounts,
```



```
cmpPos,  
cmpMode,  
triggerMode,  
triggerPrm,  
outTime);
```

上述例程中成功启动了一个一维软件比较点，启动比较之后再使比较轴进行相关运动，当比较轴的指令位置大于 10000Pulse 时，通用输出口 0 的电平将输出低电平。可以使用 MC_GetSoftCmpStatus 函数获取指定通道软件比较的状态。

5.17 一维高速比较

一维高速比较使用的硬件触发接口是高速光耦，相较于一维软件比较具有更高的精度，更快的反应时间。PMC2000-P 系列运动控制卡提供有高速比较通道，通道和高速比较接口的对应关系如下表所示：

型号	比较通道	对应高速比较口
PMC2012-P	0~3	Output12 ~ Output15

5.17.1 实现函数

一维高速比较使用的相关函数如下表所示：

函数名称	功能	参考
MC_StopHSCmp	停止高速比较口功能	函数库详解
MC_GetHSCmpPrm	获取高速比较参数	
MC_StartHS1DCmp	启动一维高速比较	
MC_GetHSCmpStatus	获取高速比较状态	
MC_ClearHSCmp	清除高速比较结果	



5.17.2 C#例程

```
//启动一个一维高速比较点
ushort connect_no = 0;//卡号 0
ushort cmp_hs_io = 0;//高速比较通道，取值范围 [0~3]，对应
Output12~Output15
ushort cmpAxis = 0;//比较轴号，轴 0
ushort posSrc = 0;//比较的位置源，0 = 指令位置，1 = 反馈位置
ushort cmpCounts = 1;//比较位置点数，在比较模式为 0~2 下，该值固定为
1
double [1]cmpPos = new double[]{10000}; //比较点位置
ushort cmpMode = 1;//比较模式
/*
    Less_Compare_Value = 0, //小于比较点模式
    Beyond_Compare_Value = 1, //大于比较点模式
    Linear_Compare_Value = 2, //等增量比较模式
    FIFO_Compare_Value = 3    //FIFO 点比较输出
*/

ushort triggerMode = 0;//触发模式
/*
    Trigger_IO_Low = 0, 输出口输出低
    Trigger_IO_High = 1, 输出口输出高
    Trigger_IO_Reverse = 2, 输出口电平翻转
    Trigger_IO_Pulse = 3, 输出脉冲
    Trigger_Axis_Imd_Stop = 4, 停止轴
*/

uint triggerPrm = 0;//触发参数
/*
    trigger_mode 为 0~3 时，对应 IO 序号为 Output12~Output15
    trigger_mode 为 4 时，该参数为轴序号；
*/

double outTime = 0;//输出时间，当 triggerMode = 3 时的输出脉冲时间，
单位是 S

//启动一维高速比较，相当于添加了一个比较点
MC_StartHS1DCmp(connect_no,
                 cmp_hs_io,
                 cmpAxis,
```



```
posSrc,  
cmpCounts,  
cmpPos,  
cmpMode,  
triggerMode,  
triggerPrm,  
outTime);
```

上述例程成功添加了一个一维高速比较点，在启动比较点之后运动相关比较轴，当相关轴指令位置运动大于 10000Pulse 时该比较点触发，通用输出口 12 将输出低电平。可以使用函数 MC_GetHSCmpStatus 获取指定通道高速比较的状态，也可以使用函数 MC_ClearHSCmp 函数清除已比较点和添加的比较信息。

5.18 二维软件比较

PMC2000-P 系列运动控制卡提供了二维软件比较功能。二维软件比较是通过添加两个轴的位置比较信息，当两个轴的比较条件都符合比较点配置的比较位置时触发比较动作。

5.18.1 实现函数

二维软件比较使用的相关函数如下表所示：

函数名称	功能	参考
MC_StopSoftCmp	停止软件比较	函数库详解
MC_GetSoftCmpPrm	获取软件比较参数	
MC_GetSoftCmpStatus	获取软件比较状态	
MC_ClearSoftCmp	清除所有配置的比较点	
MC_StartSoft2DCmp	添加并启动二维比较点	



5.18.2 C#例程

```
//添加二维比较点并启动

ushort connect_no = 0; //卡号 0
ushort cmpChan = 0; //比较通道, 取值范围[0~3]
ushort [2]cmpAxis = new ushort[]{0,1}; //比较轴号, 轴 0 和轴 1
ushort [2]posSrc = new ushort[]{0,0}; //比较的位置源, 0 = 指令位置, 1 = 反馈位置
ushort cmpCounts = 0; //保留参数
double [2]cmpPos = new double[]{10000,20000}; //比较点位置
ushort [2]cmpMode = new ushort[]{1,1}; //比较模式, 大于比较点
/*
    Less_Compare_Value = 0, //小于比较点模式
    Beyond_Compare_Value = 1, //大于比较点模式
    Linear_Compare_Value = 2, //等增量比较模式
    FIFO_Compare_Value = 3    //FIFO 点比较输出
*/

ushort triggerMode = 0; //触发模式
/*
    Trigger_IO_Low = 0, 输出口输出低
    Trigger_IO_High = 1, 输出口输出高
    Trigger_IO_Reverse = 2, 输出口电平翻转
    Trigger_IO_Pulse = 3, 输出脉冲
    Trigger_Axis_Imd_Stop = 4, 停止轴
*/

uint triggerPrm = 0; //触发参数
/*
    trigger_mode 为 0~3 时, 该参数为 IO 序号;
    trigger_mode 为 4 时, 该参数为轴序号;
*/

double outTime = 0; //输出时间, 当 triggerMode = 3 时的输出脉冲时间, 单位是 S

//启动一维软件比较, 相当于添加一个比较点
MC_StartSoft2DCmp(connect_no,
```



```
cmpChan,  
cmpAxis,  
posSrc,  
cmpCounts,  
cmpPos,  
cmpMode,  
triggerMode,  
triggerPrm,  
outTime);
```

上述例程添加了一个二维软件比较点并启动比较。当轴 0 的指令位置大于 10000 且轴 1 的指令位置大于 20000 时，将会触发比较点，通用输出口 0 将会输出低电平，至此一个比较点成功完成比较，可以通过函数 MC_GetSoftCmpStatus 获取比较状态，也可以通过 MC_ClearSoftCmp 清除所有配置的比较点。

5.19 高速锁存

PMC2000-P 系列运动控制卡提供有高速锁存功能，高速锁存是指当锁存信号触发时，能够准确记录触发时刻锁存轴的位置信息。PMC2000-P 系列控制卡提供多种锁存信号，如下表所示：

型号	信号	锁存范围			
PMC2012-P	原点(ORG)	轴 0~轴 11			
	EZ 信号	轴 0~轴 11			
	高速输入信号	高速输入口	硬件接口	锁存范围	
		LTC0	Input14	轴 0~轴 3	
		LTC1	Input15	轴 4~轴 7	



	通用输入信号	通用输入口	锁存范围
		0~13	轴 0~轴 11

5.19.1 实现函数

高速锁存使用的相关函数如下表所示：

函数名称	功能	参考
MC_StartCapture	启动锁存	函数库详解
MC_GetCapturePrm	获取指定锁存通道的配置参数	
MC_GetCaptureCount	获取指定通道的锁存状态	
MC_GetCapturePos	根据锁存状态获取锁存位置值	
MC_ReStartCapture	重启锁存	
MC_StopCapture	停止锁存	

注意：在单次锁存中，多次触发高速锁存口只锁存第一次触发时的位置，只有调用函数 MC_ReStartCapture 清除锁存状态方可再次锁存到位置值。

5.19.2 C#例程

```
ushort connect_no = 0; //卡号 0
ushort capObject = 0; //锁存对象，0 = 原点信号
/*
    CAPTURE_HOME    = 0, //原点信号
    CAPTURE_EZ = 1, //EZ 信号
    CAPTURE_HS_IN = 2, //高速 IN0 IN1 信号
    CAPTURE_GEN_IN = 3 //通用 IN 信号
*/
```



```
ushort capAxis = 0;//锁存轴号
//锁存通道
//当锁存对象为高速输入口时取值[0,1]代表 Input14、Input15
//当锁存对象为通用输入口时为输入口所有号
//锁存对象为 Home 和 EZ 时该变量无效
ushort capChan = 0;

ushort capIO = 0;//保留参数
ushort capMode = 0;//锁存模式，0 = 单次锁存；1 = 连续锁存
ushort capLogic = 0;//锁存信号的锁存逻辑
/*
CAPTURE_FALLING_EDGE   = 0, //下降沿捕获
CAPTURE_RISING_EDGE    = 1, //上升沿捕获
CAPTURE_DUAL_EDGE      = 2 //双边沿捕获， 注意：不支持 cap_object 对
象为 Home 和 EZ 时情况
*/

ushort capPosSrc = 0;//锁存位置源，0 = 指令位置；1 = 反馈位置
//启动锁存
MC_StartCapture(connect_no,
                capObject,
                capAxis,
                capChan,
                capIO,
                capMode,
                capLogic,
                capPosSrc);

MC_Power(connect_no,capAxis,1); //在轴运动之前确保轴已使能
//让锁存轴运动
MC_MoveVelocity(connect_no,capAxis,100,100, 500,0.1,0.1, 0, 1000000);
//...触发锁存轴的原点信号...

uint capCounts = 0;//用于获取以锁存点个数
//获取指定通道的锁存信息
MC_GetCaptureCount(connect_no,      capObject,capAxis,capChan      ,
&capCounts);

double [1000] capPos= new double[]; //用于获取锁存位置点的数组
ushort realCount = 0;//用于接收实际获取到的锁存点个数
MC_GetCapturePos(connect_no,  capObject,capAxis,capChan  ,capCounts,
```



&realCount, capPos);

```
for(int i = 0 ;i<realCount;i++)  
{  
    //打印锁存位置点 capPos[i]  
}
```

上述例程成功启动了轴 0 的原点锁存并且获取锁存位置。当已经获取一次锁存位置之后，可以使用 MC_ReStartCapture 函数重启锁存配置，还可以使用 MC_StopCapture 函数停止锁存功能。



第6章 附录

6.1 函数列表

功能分类	函数名称	函数说明
连接控制器操作	MC_Open	板卡连接函数
	MC_Close	关闭控制器
	MC_ColdReset	硬件复位，控制器所有信息复位
	MC_WarmReset	热复位，坐标系等信息复位
控制器信息	MC_GetConnectLineBoxState	获取板卡和接线盒的连接状态
	MC_GetTotalAxesAndIONum	获取控制器的轴数、IO口数量等相关信息
	MC_GetTotalAdAndDaNum	获取控制器的模拟量信息
	MC_GetControllerVersion	获取控制器的系统版本信息
	MC_GetProductID	获取控制器的产品型号
	MC_GetCapComNum	获取控制器的捕获通道数量
	MC_GetHandWheelEncNum	获取控制器的手轮和辅助编码器通道数量
单轴运动控制	MC_SetAxisVelSmooth	速度平滑参数设定
	MC_GetAxisVelSmooth	速度平滑参数获取



	MC_SetAxisMotionPrm	加速度按时间设置单轴的运动参数
	MC_GetAxisMotionPrm	加速度按时间单轴的运动参数获取
	MC_SetAxisMotionPrmAcc	按加速度单位设置单轴的运动参数
	MC_GetAxisMotionPrmAcc	按加速度单位获取单轴的运动参数
	MC_MoveAbsolute	开始一段绝对运动
	MC_MoveRelative	开始一段相对运动
	MC_MoveMultiPos	多段运动指令
	MC_MovePos	单轴定长运动指令
	MC_MovePosMultiAxes	多轴定长运动指令
	MC_MoveVelocity	单轴速度运动指令
	MC_MoveVel	单轴速度运动指令
	MC_MoveVelMultiAxes	多轴定速运动指令
	MC_MoveUpdateVel	在线变速指令
	MC_MoveUpateVelRatio	单轴改变倍率
	MC_Home	回零运动指令
	MC_GetHomePrm	获取回零参数
	MC_GetHomeResult	获取回零结果
坐标系运动控制	MC_StopCrd	停止插补系运动
	MC_GetCrdStopModeTime	获取插补系停止设置
	MC_SetCrdStopTime	设置插补系减速时间
	MC_GetCrdCheckDone	判断插补系是否在运动



		中
	MC_GetCrdCmdSpeed	获取当前插补系的指令速度
	MC_SetCrdVelSmooth	设置插补系速度平滑
	MC_GetCrdVelSmooth	获取插补系速度平滑设置
	MC_SetCrdMotionPrm	设置插补系运动参数
	MC_GetCrdMotionPrm	获取插补系运动参数
	MC_SetCrdMotionPrmAcc	以加减速单位设置插补系运动参数
	MC_GetCrdMotionPrmAcc	加减速以加速度单位获取插补系运动参数
	MC_MoveLinear	直线插补指令
	MC_MovePlaneArcC	终点圆心的圆弧插补指令
	MC_MovePlaneArcR	终点半径的圆弧插补指令
	MC_MovePlaneArc3P	三点圆弧插补指令
	MC_MoveSpaceArcC	终点圆心的空间圆弧插补指令
	MC_MoveSpaceArc3P	三点圆弧空间插补指令
	MC_MovePlaneEllipse	椭圆插补指令
	MC_GetCrdStatus	获取插补器的工作状态
	MC_GetCrdBufStatus	连续插补模式下，获取buffer的空间大小
	MC_MoveContiLinear	多段直线插补指令



	MC_SetCrdCmdWait	设置连续插补时，等待上位机指令启动运动
	MC_SetCrdCmdStart	设置连续插补时，立马执行运动
	MC_SetCrdLookAhead	配置连续前瞻插补的模式
	MC_GetCrdLookAhead	获取连续前瞻插补的模式
	MC_SetCrdCornerPrm	设置前瞻处理的拐角参数
	MC_GetCrdCornerPrm	获取前瞻处理的拐角参数
	MC_SetLimitArcPrm	设置圆弧位置限制参数
	MC_GetLimitArcPrm	获取圆弧位置限制参数
轴状态和轴参数	MC_Power	轴使能
	MC_SetPulseMode	脉冲输出模式配置
	MC_GetPulseMode	获取脉冲输出模式配置
	MC_SetEncodeMode	设置轴对应编码器的工作模式及方向
	MC_GetEncodeMode	获取轴对应编码器的工作模式及方向
	MC_GetAxisMachineState	判断轴是否运动完成
	MC_GetAxisCheckDone	判断轴是否在运动中
	MC_GetAxisMotionMode	获取轴当前的运动模式
	MC_SetAxisEquvi	设置轴当量
	MC_GetAxisEquvi	获取轴当量设置



	MC_StopAxis	停止轴
	MC_GetAxisStopModeTime	获取轴停止模式和停止时间
	MC_SetAxisStopTime	设置轴减速停止时间
	MC_StopEmgImd	紧急停止所有轴
	MC_GetAxisStopReason	获取轴停止原因
	MC_ClearAxisStopReason	清除轴的停止原因
	MC_GetAxisCmdPos	获取轴的指令位置
	MC_GetAxisCmdSpeed	获取轴的指令速度
	MC_GetAxisFeedbackPos	获取轴的反馈位置
	MC_SetAxisCmdPos	设置轴的指令位置
	MC_SetAxisSoftELPrm	设置轴的软件限位参数
	MC_GetAxisSoftELPrm	获取轴的软件限位参数
通用 IO	MC_GetInIoBit	按位获取通用输入口的状态
	MC_GetInIoPort	按端口获取通用输入口的状态
	MC_GetInIoPtr	按起始索引和需要读取的数量读取通用输入口的状态
	MC_SetOutIoBit	按位设置通用输出口状态
	MC_SetOutIoPort	按端口设置通用输出口的状态
	MC_SetOutIoPtr	按起始索引连续写入多个 IO 点状态



	MC_GetOutloBit	按位获取通用输出口的状态
	MC_GetOutloPort	按端口获取通用输出口的状态
	MC_GetOutloPtr	按起始索引和需要读取的数量读取通用输出口的状态
	MC_SetReuseloMap	设置虚拟 IO 口复用
	MC_GetReuseloMap	获取虚拟 IO 口复用配置
	MC_GetReuseInloBit	按位获取虚拟 IO 口状态
轴专用 IO	MC_SetAxisOutlo	设置轴的专用输出口电平
	MC_GetAxisOutlo	获取轴的专用输出口电平
	MC_SetAxisInloPrm	配置轴专用信号 IO 口逻辑
	MC_GetAxisInloPrm	获取轴专用信号 IO 口逻辑
	MC_SetAxisInloMap	配置轴专用信号映射
	MC_GetAxisInloMap	获取轴专用信号映射
	MC_GetAxisInloState	获取轴所有专用信号状态
	MC_GetAxisAlarmState	获取轴伺服报警信号状态



	MC_GetAxisHomeState	获取轴原点信号状态
	MC_GetAxisELState	获取轴限位信号状态
	MC_GetAxisInpState	获取轴到位信号状态
辅助编码器及手轮	MC_SetAuxEncoderPrm	设置辅助编码器计数参数
	MC_GetAuxEncoderPrm	获取辅助编码器计数参数
	MC_SetAuxEncoderPositon	设置辅助编码器值
	MC_GetAuxEncoderPositon	获取辅助编码器值
	MC_SetHandleWheelPrm	设置手轮参数
	MC_GetHandleWheelPrm	获取手轮参数配置
	MC_StartHandleWheel	启动手轮运动
锁存	MC_StartCapture	启动锁存指令
	MC_GetCapturePrm	获取锁存参数
	MC_GetCaptureCount	获取锁存状态
	MC_GetCapturePos	获取锁存位置
	MC_ReStartCapture	重启锁存器
	MC_StopCapture	停止锁存器
比较	MC_StopHSCmp	停止高速比较口功能
	MC_GetHSCmpPrm	获取高速比较参数
	MC_StartHS1DCmp	启动一维高速比较
	MC_GetHSCmpStatus	获取高速比较状态
	MC_ClearHSCmp	清除高速比较结果
	MC_StopSoftCmp	停止软件比较



	MC_GetSoftCmpPrm	获取软件比较参数
	MC_GetSoftCmpStatus	获取软件比较状态信息
	MC_ClearSoftCmp	清除所有配置的软件比较点
	MC_StartSoft1DCmp	启动一维软件比较指令
	MC_StartSoft2DCmp	启动二维软件比较指令
PWM	MC_StartPwm	启动 PWM 输出
	MC_GetPwmPrm	获取 PWM 通道参数配置

6.2 函数说明

对提供的函数接口进行详细说明。

6.2.1 控制器连接相关函数

DWORD MC_Open(WORD connect_no,WORD type, char *pconnectstring,WORD* PcieNum ,WORD* PcieIndex);	
开卡函数	
参数	说明
connect_no	连接号，PCI 连接的情况下默认为 0
type	连接类型：1 = 网络通讯；2 = PCI 通讯
pconnectstring	网络连接的 IP 地址；PCI 连接类型则为空
PcieNum	返回获取到的 PCI 通讯控制器数量
PcieIndex	返回 PCI 控制器的连接号，该连接号用于后续调用各函数的首参数



返回值	错误码
-----	-----

DWORD MC_Close(WORD connect_no)	
关闭控制器，释放资源	
参数	说明
connect_no	连接号，成功开卡获得
返回值	错误码

DWORD MC_ColdReset(WORD connect_no);	
硬件复位，控制器所有信息复位	
参数	说明
connect_no	连接号，成功开卡获得
返回值	错误码

DWORD MC_WarmReset(WORD connect_no);	
热复位，坐标系等信息复位	
参数	说明
connect_no	连接号，成功开卡获得
返回值	错误码



6.2.2 控制器信息相关函数

DWORD MC_GetConnectLineBoxState(WORD connect_no, WORD* conState);	
获取和接线盒的连接状态	
参数	说明
connect_no	连接号，成功开卡获得
conState	返回接线盒连接状态；接线盒没有连接 = 0； 接线盒上电 = 1； 接线盒连接初始化阶段 = 2； 接线盒连接正常运行阶段 = 3； 接线盒连接运行错误阶段 = 4；
返回值	错误码

DWORD MC_GetTotalAxesAndIONum(WORD connect_no,WORD* total_axes, WORD* liner_num ,WORD* total_in_num,WORD* total_out_num, WORD* total_pwm_num);	
获取控制器的轴、IO 口相关信息	
参数	说明
connect_no	连接号，成功开卡获得
total_axes	返回控制器的轴数
liner_num	返回控制器的插补系数个数
total_in_num	返回控制器的通用输入口数量
total_out_num	返回控制器的通用输出口数量
total_pwm_num	返回控制器的 PWM 通道数量
返回值	错误码



DWORD MC_GetTotalAdAndDaNum(WORD connect_no,WORD* total_ad_num,WORD* total_da_num);	
获取控制器的模拟量信息	
参数	说明
connect_no	连接号，成功开卡获得
total_ad_num	A/D 转换通道数量
total_da_num	D/A 转换通道数量
返回值	错误码

DWORD MC_GetControllerVersion(WORD connect_no,char* product_type,char* soft_version,DWORD* firmware_version,DWORD* dll_version);	
获取控制系统版本信息	
参数	说明
connect_no	连接号，成功开卡获得
product_type	返回产品类型，例如"PMC200C"
soft_version	返回软件版本，例如"pmc_200C_v1.0_soft"
firmware_version	返回硬件版本，例如"0x01000100"
dll_version	返回动态库版本，例如"0x10230113"
返回值	错误码



DWORD MC_GetProductID(WORD connect_no,DWORD* product_id);	
获取控制器产品型号	
参数	说明
connect_no	连接号，成功开卡获得
product_id	返回产品型号
返回值	错误码

DWORD MC_GetCapComNum(WORD connect_no,WORD* total_hs_cap_num,WORD* total_hs_com_num,WORD* total_soft_cap_num,WORD* total_soft_com_num,WORD* total_2d_hs_com_num,WORD* total_2d_soft_com_num);	
返回捕获通道和比较口的信息	
参数	说明
connect_no	连接号，成功开卡获得
total_hs_cap_num	返回硬件捕获通道数量
total_hs_com_num	返回硬件比较通道数量
total_soft_cap_num	返回软件捕获通道数量
total_soft_com_num	返回软件比较通道数量
total_2d_hs_com_num	返回二维硬件比较通道数量
total_2d_soft_com_num	返回二维软件比较通道数量
返回值	错误码



DWORD MC_GetHandWheelEncNum(WORD connect_no,WORD* total_handlewheel_num,WORD* total_enc_num);	
获取手轮通道及编码器通道数量	
参数	说明
connect_no	连接号，成功开卡获得
total_handlewheel_num	返回获取的手轮通道数量
total_enc_num	返回获取的编码器通道数量
返回值	错误码

6.2.3 单轴运动控制相关函数

DWORD MC_SetAxisVelSmooth(WORD connect_no, WORD axis, WORD ifEnable, double smTime);	
设置速度平滑参数	
参数	说明
connect_no	连接号，成功开卡获得
axis	轴号(从 0 开始)
ifEnable	是否使能平滑，0 表示禁止，1 表示使能
smTime	平滑时间，单位 s，最小值为 0.001s
返回值	错误码



DWORD MC_GetAxisVelSmooth(WORD connect_no, WORD axis, WORD* ifEnable, double* smTime);	
获取速度平滑参数	
参数	说明
connect_no	连接号，成功开卡获得
axis	轴号(从 0 开始)
ifEnable	返回是否使能平滑
smTime	返回平滑时间
返回值	错误码

DWORD MC_SetAxisMotionPrm(WORD connect_no,WORD axis,double start_vel,double max_vel,double stop_vel,double t_acc,double t_dec,double s_ratio,double maxAcc);	
加减速按照时间单位设置运动参数	
参数	说明
connect_no	连接号，成功开卡获得
axis	轴号(从 0 开始)
start_vel	启动速度(Pulse/s)
max_vel	最大运行速度(Pulse/s)



stop_vel	停止速度(Pulse/s)
t_acc	加速时间, 最大值为 100s, 最小值为 0
t_dec	减速时间, 最大值为 100s, 最小值为 0
s_ratio	S 型曲线参数配置, 最大值为 1, 最小值为 0; 该值为 0 时为 T 型速度曲线
maxAcc	最大加减速限制,
返回值	错误码

<pre> DWORD MC_GetAxisMotionPrm(WORD connect_no,WORD axis,double* start_ vel,double* max_vel,double* stop_vel,double* t_acc,double* t_dec,double *s_r atio,double* maxAcc); </pre>	
获取运动参数	
参数	说明
connect_no	连接号, 成功开卡获得
axis	轴号(从 0 开始)
start_vel	返回启动速度(Pulse/s)
max_vel	返回最大运行速度(Pulse/s)
stop_vel	返回停止速度(Pulse/s)
t_acc	返回加速时间, 最大值为 100s, 最小值为 0
t_dec	返回减速时间, 最大值为 100s, 最小值为 0
s_ratio	返回 S 型曲线参数配置, 最大值为 1, 最小值为 0; 该值为 0 时为 T 型速度曲线



maxAcc	返回最大加减速限制,
返回值	错误码

DWORD MC_SetAxisMotionPrmAcc(WORD connect_no,WORD axis,double start_vel,double max_vel,double stop_vel,double acc,double dec,double s_ratio,double maxAcc);	
按照加速度单位设置运动参数	
参数	说明
connect_no	连接号, 成功开卡获得
axis	轴号(从 0 开始)
start_vel	启动速度(Pulse/s)
max_vel	最大运行速度(Pulse/s)
stop_vel	停止速度(Pulse/s)
acc	加速度, 最大值为 2 的 32 次方, 最小值为 0
dec	减速度, 最大值为 2 的 32 次方, 最小值为 0
s_ratio	S 型曲线参数配置, 最大值为 1, 最小值为 0; 该值为 0 时为 T 型速度曲线
maxAcc	最大加减速限制,
返回值	错误码

DWORD MC_GetAxisMotionPrmAcc(WORD connect_no,WORD axis,double* st	
---	--



art_vel,double* max_vel,double* stop_vel,double* acc,double* dec,double *s_ratio,double* maxAcc);	
获取加速度单位的运动参数	
参数	说明
connect_no	连接号，成功开卡获得
axis	轴号(从 0 开始)
start_vel	返回启动速度(Pulse/s)
max_vel	返回最大运行速度(Pulse/s)
stop_vel	返回停止速度(Pulse/s)
acc	返回加速度，最大值为 2 的 32 次方，最小值为 0
dec	返回减速度，最大值为 2 的 32 次方，最小值为 0
s_ratio	返回 S 型曲线参数配置，最大值为 1，最小值为 0；该值为 0 时为 T 型速度曲线
maxAcc	返回最大加减速限制，
返回值	错误码

DWORD MC_MovePos(WORD connect_no,WORD axis,WORD pos_mode,double pos);	
单轴定长运动指令	
参数	说明
connect_no	连接号，成功开卡获得
axis	轴号(从 0 开始)



pos_mode	位置模式；0 表示相对运动，1 表示绝对运动
pos	目标位置
返回值	错误码

DWORD MC_MoveAbsolute(WORD connect_no,WORD axis,double pos,double start_V,double stop_V, double Vel,double Acc,double Dec, double S_Ratio, double maxAcc);	
单轴按绝对运动模式运行至目标位置	
参数	说明
connect_no	连接号，成功开卡获得
axis	轴号(从 0 开始)
pos	目标位置
start_V	启动速度(Pulse/s)
stop_V	停止速度(Pulse/s)
Vel	最大运行速度(Pulse/s)
Acc	加速度
Dec	减速度
S_Ratio	S 段比例
maxAcc	最大加减速限制
返回值	错误码



```
DWORD MC_MoveRelative(WORD connect_no,WORD axis,double pos,double start_V,double stop_V, double Vel,double Acc,double Dec, double S_Ratio, double maxAcc);
```

单轴按相对运动模式运行至目标位置

参数	说明
connect_no	连接号，成功开卡获得
axis	轴号(从 0 开始)
pos	目标位置
start_V	启动速度(Pulse/s)
stop_V	停止速度(Pulse/s)
Vel	最大运行速度(Pulse/s)
Acc	加速度
Dec	减速度
S_Ratio	S 段比例
maxAcc	最大加减速限制
返回值	错误码

```
DWORD MC_MoveMultiPos(WORD connect_no,WORD axis,WORD pos_mode, double first_pos,double next_max_v,double next_stop_v,double next_acc_t,double next_dec_t,double next_pos);
```



启动多段运动，目前支持两段；第一段运动完成以后，立马执行第二段，两段的衔接速度不会降到零	
参数	说明
connect_no	连接号，成功开卡获得
axis	轴号(从 0 开始)
pos_mode	保留参数，只支持绝对运动
first_pos	第一段的目标位置
next_max_v	第二段的最大速度
next_stop_v	第二段的停止速度
next_acc_t	第二段的加速时间
next_dec_t	第二段的减速时间
next_pos	第二段的目标位置（暂时不支持反向）
返回值	错误码

<pre>DWORD MC_MovePosMultiAxes(WORD connect_no, WORD axis_num,WORD* axis,WORD* pos_mode,double* pos);</pre>	
多轴定长运动，多个轴同时启动一段位置运动	
参数	说明
connect_no	连接号，成功开卡获得
axis_num	轴数量
axis	运动轴数组



pos_mode	运动模式数组
pos	目标位置数组
返回值	错误码

DWORD MC_MoveVelocity(WORD connect_no,WORD axis,double start_V,double stop_V, double Vel,double Acc,double Dec, double S_Ratio, double maxAcc);	
单轴启动速度运动	
参数	说明
connect_no	连接号, 成功开卡获得
axis	轴号(从 0 开始)
start_V	启动速度(Pulse/s)
stop_V	停止速度(Pulse/s)
Vel	最大运行速度(Pulse/s)
Acc	加速度
Dec	减速度
S_Ratio	S 段规划比例
maxAcc	最大加减速限制
返回值	错误码



DWORD MC_MoveVel(WORD connect_no,WORD axis, double vel);	
单轴启动定速运动	
参数	说明
connect_no	连接号, 成功开卡获得
axis	轴号(从 0 开始)
vel	最大运行速度, 大于 0 表示正向运动, 小于 0 表示反向运动
返回值	错误码

DWORD MC_MoveVelMultiAxes(WORD connect_no, WORD axis_num,WORD* axis,short* dir);	
多轴定速运动指令	
参数	说明
connect_no	连接号, 成功开卡获得
axis_num	轴数量
axis	运动轴数组
dir	运动方向数组
返回值	错误码

DWORD MC_MoveUpdateVel(WORD connect_no, WORD axis, double vel);	
单轴运动中改变速度	
参数	说明



connect_no	连接号，成功开卡获得
axis	轴号(从 0 开始)
vel	新速度，小于 0 为反向；位置运动中 vel 没有方向
返回值	错误码

DWORD MC_MoveUpateVelRatio(WORD connect_no,WORD axis, double ratio);	
单轴运动中改变倍率，当前运行速度乘以倍率得到新的运行速度	
参数	说明
connect_no	连接号，成功开卡获得
axis	轴号(从 0 开始)
ratio	倍率，0 表示暂停状态
返回值	错误码

DWORD MC_Home(WORD connect_no,WORD axis,WORD pos_object,WORD home_mode, int home_dir,WORD home_logic,WORD ez_count,WORD offset_mode,double pos_offset);	
启动回零运动	
参数	说明
connect_no	连接号，成功开卡获得



axis	轴号(从 0 开始)
pos_object	锁存回零的锁存位置; 0 表示指令位置, 1 表示反馈位置
home_mode	回零模式: 1 = 一次捕获 ORG 信号回零 2 = 高速运动+反向一次捕获 ORG 信号回零 3 = 高速运动+同向低速捕获 ORG 信号回零 4 = ORG+反向找 EZ 捕获回零 5 = ORG+运动方向 EZ 捕获回零 6 = EZ 捕获回零 7 = 单次限位回零 8 = 两次限位回零 9 = 多次限位回零
home_dir	回零方向, -1 表示负方向, 1 表示正方向
home_logic	原点信号的有效电平, 0 表示低电平有效, 1 表示高电平有效
ez_count	如果回零模式为 EZ 回零, 回零的 EZ 计数个数; 最小值为 1
offset_mode	保留参数
pos_offset	保留参数
返回值	错误码

<pre>DWORD MC_GetHomePrm(WORD connect_no,WORD axis,WORD* pos_object, WORD* home_mode, int* home_dir,WORD* home_logic,WORD* ez_count,WORD* offset_mode,double* pos_offset);</pre>	
获取回零运动参数	
参数	说明
connect_no	连接号, 成功开卡获得
axis	轴号(从 0 开始)



pos_object	返回锁存回零的锁存位置
home_mode	返回回零模式
home_dir	返回回零方向
home_logic	返回原点信号的有效电平
ez_count	返回 EZ 回零的 EZ 计数个数
offset_mode	返回回零偏移模式
pos_offset	返回偏移位置
返回值	错误码

DWORD MC_GetHomeResult(WORD connect_no,WORD axis,WORD* complete_state);	
获取回零结果	
参数	说明
connect_no	连接号，成功开卡获得
axis	轴号(从 0 开始)
complete_state	返回回零结果； 0 = 回零未完成 1 = 回零完成，找到原点位置 2 = 回零过程报错，没有原点信号 3 = 回零过程发现设备有两个原点信号 4 = 回零过程没有发现 EZ 信号 5 = 回零配置的方向错误 6 = 回零过程限位停止 7 = 超出软件限位范围 100 = 没有正常回零完成



返回值	错误码
-----	-----

6.2.4 坐标系运动控制相关函数

DWORD MC_StopCrd(WORD connect_no, WORD crd, WORD mode, double stop_time);	
停止坐标系运动	
参数	说明
connect_no	连接号，成功开卡获得
crd	坐标系号(从 0 开始)
mode	停止模式；0 表示减速停止，1 表示立即停止
stop_time	减速停止的停止时间(s)
返回值	错误码

DWORD MC_SetCrdStopTime(WORD connect_no, WORD crd, double stop_time);	
设置坐标系减速停止的停止时间	
参数	说明
connect_no	连接号，成功开卡获得
crd	坐标系号(从 0 开始)
stop_time	减速停止的停止时间(s)
返回值	错误码



```
DWORD MC_GetCrdStopModeTime(WORD connect_no, WORD crd, WORD* mode, double* stop_time);
```

获取坐标系减速停止的模式和停止时间

参数	说明
connect_no	连接号，成功开卡获得
crd	坐标系号(从 0 开始)
mode	返回坐标系的停止方式
stop_time	返回减速停止的停止时间(s)
返回值	错误码

```
DWORD MC_GetCrdCheckDone(WORD connect_no, WORD crd, WORD* crd_state);
```

获取坐标系运动状态

参数	说明
connect_no	连接号，成功开卡获得
crd	坐标系号(从 0 开始)
crd_state	运动状态；0 表示运行，1 表示使能并空闲状态
返回值	错误码

```
DWORD MC_GetCrdCmdSpeed(WORD connect_no, WORD crd, double* speed);
```



读取插补系当前速度	
参数	说明
connect_no	连接号，成功开卡获得
crd	坐标系号(从 0 开始)
speed	返回当前运行速度
返回值	错误码

DWORD MC_SetCrdVelSmooth(WORD connect_no, WORD crd, WORD ifEnable, double smTime);	
设置坐标系速度规划速度平滑	
参数	说明
connect_no	连接号，成功开卡获得
crd	坐标系号(从 0 开始)
ifEnable	是否使能速度平滑，0 表示禁止，1 表示使能
smTime	平滑时间
返回值	错误码

DWORD MC_GetCrdVelSmooth(WORD connect_no, WORD crd, WORD* ifEnable, double* smTime);	
获取坐标系速度规划速度平滑参数	
参数	说明



connect_no	连接号，成功开卡获得
crd	坐标系号(从 0 开始)
ifEnable	返回是否使能速度平滑
smTime	返回平滑时间
返回值	错误码

DWORD MC_SetCrdMotionPrm(WORD connect_no,WORD crd,double start_vel,double max_vel,double stop_vel,double t_acc,double t_dec,double s_ratio,double maxAcc);	
按加减速时间单位设置坐标系运动参数	
参数	说明
connect_no	连接号，成功开卡获得
crd	坐标系号(从 0 开始)
start_vel	启动速度(Pulse/s)
max_vel	最大速度(Pulse/s)
stop_vel	停止速度(Pulse/s)
t_acc	加速时间，最大值为 100s，最小值为 0
t_dec	减速时间，最大值为 100s，最小值为 0
s_ratio	S 型曲线参数，取值范围[0,1]，为 0 时是 T 型速度曲线
maxAcc	最大加减速限制



返回值	错误码
-----	-----

DWORD MC_GetCrdMotionPrm(WORD connect_no,WORD crd,double* start_vel,double* max_vel,double* stop_vel,double* t_acc,double* t_dec,double* s_ratio,double* maxAcc);	
获取按加减速时间单位坐标系运动参数	
参数	说明
connect_no	连接号，成功开卡获得
crd	坐标系号(从 0 开始)
start_vel	返回启动速度(Pulse/s)
max_vel	返回最大速度(Pulse/s)
stop_vel	返回停止速度(Pulse/s)
t_acc	返回加速时间
t_dec	返回减速时间
s_ratio	返回 S 型曲线参数
maxAcc	返回最大加减速限制
返回值	错误码

DWORD MC_SetCrdMotionPrmAcc(WORD connect_no,WORD crd,double start_vel,double max_vel,double stop_vel,double acc,double dec,double s_ratio,do	
--	--



able maxAcc);	
按加减速单位设置坐标系运动参数	
参数	说明
connect_no	连接号, 成功开卡获得
crd	坐标系号(从 0 开始)
start_vel	启动速度(Pulse/s)
max_vel	最大速度(Pulse/s)
stop_vel	停止速度(Pulse/s)
acc	加速度
dec	减速度
s_ratio	S 型曲线参数, 取值范围[0,1], 为 0 时是 T 型速度曲线
maxAcc	最大加减速限制
返回值	错误码

DWORD MC_GetCrdMotionPrmAcc(WORD connect_no,WORD crd,double* start_vel,double* max_vel,double* stop_vel,double* acc,double* dec,double* s_ratio,double* maxAcc);	
获取按加减速单位的坐标系运动参数	
参数	说明
connect_no	连接号, 成功开卡获得
crd	坐标系号(从 0 开始)



start_vel	返回启动速度(Pulse/s)
max_vel	返回最大速度(Pulse/s)
stop_vel	返回停止速度(Pulse/s)
acc	返回加速度
dec	返回减速度
s_ratio	返回 S 型曲线参数
maxAcc	返回最大加减速限制
返回值	错误码

DWORD MC_MoveLinear(WORD ConnectNo,WORD Crd,WORD AxisNum,WORD* AxisList,double* Dist,WORD Crd_Mode);	
直线插补命令	
参数	说明
connect_no	连接号，成功开卡获得
crd	坐标系号(从 0 开始)
AxisNum	插补轴数
AxisList	插补轴数组
Dist	目标位置数组
Crd_Mode	插补的位置模式；0 表示相对位置，1 表示绝对位置
返回值	错误码



DWORD MC_MovePlaneArcC(WORD ConnectNo,WORD Crd,WORD AxisNum,WORD* AxisList,double* Target_Pos,double* Cen_Pos,WORD Arc_Dir,DWORD Circle,WORD Crd_Mode);	
终点圆心的圆弧插补函数	
参数	说明
connect_no	连接号，成功开卡获得
crd	坐标系号(从 0 开始)
AxisNum	插补轴数
AxisList	插补轴数组
Target_Pos	目标位置数组
Cen_Pos	圆心位置数组
Arc_Dir	圆弧插补方向；0 表示顺时针，1 表示逆时针
Circle	插补圈数
Crd_Mode	插补的位置模式；0 表示相对位置，1 表示绝对位置
返回值	错误码

DWORD MC_MovePlaneArcR(WORD ConnectNo,WORD Crd,WORD AxisNum,WORD* AxisList,double* Target_Pos,double Arc_Radius,WORD Arc_Dir,DWORD Circle,WORD Crd_Mode);	
终点半径的圆弧插补函数	



参数	说明
connect_no	连接号，成功开卡获得
crd	坐标系号(从 0 开始)
AxisNum	插补轴数
AxisList	插补轴数组
Target_Pos	目标位置数组
Arc_Radius	圆弧半径
Arc_Dir	0 表示劣弧，1 表示优弧
Circle	插补圈数
Crd_Mode	插补的位置模式；0 表示相对位置，1 表示绝对位置
返回值	错误码

DWORD MC_MovePlaneArc3P(WORD ConnectNo,WORD Crd,WORD AxisNum, WORD* AxisList,double* Target_Pos,double* Mid_Pos,DWORD Circle,WORD Cr d_Mode);	
三点圆弧插补函数(起点、中点、目标位置点)	
参数	说明
connect_no	连接号，成功开卡获得
crd	坐标系号(从 0 开始)
AxisNum	插补轴数
AxisList	插补轴数组



Target_Pos	目标位置数组
Mid_Pos	中点位置数组
Circle	插补圈数
Crd_Mode	插补的位置模式；0 表示相对位置，1 表示绝对位置
返回值	错误码

DWORD MC_MoveSpaceArcC(WORD ConnectNo,WORD Crd,WORD AxisNum,WORD* AxisList,double* Target_Pos,double* Cen_Pos,WORD Arc_Dir,DWORD Circle,WORD Crd_Mode);	
终点圆心的空间圆弧插补函数	
参数	说明
connect_no	连接号，成功开卡获得
crd	坐标系号(从 0 开始)
AxisNum	插补轴数
AxisList	插补轴数组
Target_Pos	目标位置数组
Cen_Pos	圆心位置数组
Arc_Dir	圆弧插补方向，0 为顺时针，1 为逆时针
Circle	插补圈数
Crd_Mode	插补的位置模式；0 表示相对位置，1 表示绝对位置



返回值	错误码
-----	-----

DWORD MC_MoveSpaceArc3P(WORD ConnectNo,WORD Crd,WORD AxisNum,WORD* AxisList,double* Target_Pos,double* Mid_Pos,DWORD Circle,WORD Crd_Mode);	
三点圆弧空间插补函数(起点、中点、目标位置点)	
参数	说明
connect_no	连接号，成功开卡获得
crd	坐标系号(从 0 开始)
AxisNum	插补轴数
AxisList	插补轴数组
Target_Pos	目标位置数组
Mid_Pos	中点位置数组
Circle	插补圈数
Crd_Mode	插补的位置模式；0 表示相对位置，1 表示绝对位置
返回值	错误码

DWORD MC_MovePlaneEllipse(WORD ConnectNo,WORD Crd,WORD AxisNum,WORD* AxisList,double* Target_Pos,double* Cen_Pos, double short_len, double long_len,WORD Arc_Dir,DWORD Circle,WORD Crd_Mode);	
---	--



椭圆插补函数	
参数	说明
connect_no	连接号，成功开卡获得
crd	坐标系号(从 0 开始)
AxisNum	插补轴数
AxisList	插补轴数组
Target_Pos	目标位置数组
Cen_Pos	圆心位置数组
short_len	椭圆短轴
long_len	椭圆长轴
Arc_Dir	插补方向，0 为顺时针，1 为逆时针
Circle	插补圈数
Crd_Mode	插补的位置模式；0 表示相对位置，1 表示绝对位置
返回值	错误码

DWORD MC_GetCrdStatus(WORD ConnectNo,WORD Crd,WORD* runningState, DWORD* mark, WORD* runningMode);	
获取插补器的工作状态	
参数	说明
connect_no	连接号，成功开卡获得
crd	坐标系号(从 0 开始)



runningState	返回当前插补器的工作状态；0 表示运行，1 表示停止
mark	返回当前插补器的段号，如果是单段插补，返回 1
runningMode	返回当前插补器工作的模式
返回值	错误码

DWORD MC_GetCrdBufStatus(WORD ConnectNo,WORD Crd,WORD* curRunningMark,WORD* freeSpace, WORD* usedSpace);	
连续插补模式下，获取缓存区的空间大小	
参数	说明
connect_no	连接号，成功开卡获得
crd	坐标系号(从 0 开始)
curRunningMark	返回当前执行的插补段标记
freeSpace	返回剩余空间大小
usedSpace	返回已用空间大小
返回值	错误码

DWORD MC_MoveContiLinear(WORD ConnectNo,WORD Crd,WORD AxisNum,WORD* AxisList,double* Dist,WORD Crd_Mode, DWORD mark, WORD linkToLast, double feedVel, double endVel);	
连续直线插补指令	
参数	说明



connect_no	连接号，成功开卡获得
crd	坐标系号(从 0 开始)
AxisNum	插补轴数
AxisList	插补轴数组
Dist	目标位置数组
Crd_Mode	位置模式；0 表示相对运动，1 表示绝对运动
mark	设置的插补段号
linkToLast	是否连接上一段，0 表示不连接，1 表示连接。在前瞻模式下该配置有效，在连续插补模式下无效
feedVel	该段插补的最大速度
endVel	该段插补的最大停止速度
返回值	错误码

DWORD MC_SetCrdLookAhead(WORD ConnectNo,WORD Crd,WORD lookaheadMode);	
配置连续前瞻插补的模式	
参数	说明
connect_no	连接号，成功开卡获得
crd	坐标系号(从 0 开始)
lookaheadMode	0 表示连续插补，1 表示前瞻处理插补
返回值	错误码



DWORD MC_GetCrdLookAhead(WORD ConnectNo,WORD Crd,WORD* lookaheadMode);	
获取连续前瞻插补的模式	
参数	说明
connect_no	连接号，成功开卡获得
crd	坐标系号(从 0 开始)
lookaheadMode	返回连续前瞻插补模式
返回值	错误码

DWORD MC_SetCrdCmdWait(WORD ConnectNo,WORD Crd);	
设置连续插补时，等待上位机指令启动运动	
参数	说明
connect_no	连接号，成功开卡获得
crd	坐标系号(从 0 开始)
返回值	错误码

DWORD MC_SetCrdCmdStart(WORD ConnectNo,WORD Crd);	
与 MC_SetCrdCmdWait 指令配合使用，表示立马执行运动	
参数	说明
connect_no	连接号，成功开卡获得



crd	坐标系号(从 0 开始)
返回值	错误码

DWORD MC_SetCrdCornerPrm(WORD ConnectNo, WORD Crd, WORD corner Flag, double maxPathError, double maxVelLimit, double maxAccLimit);	
设置前瞻插补的拐角参数	
参数	说明
connect_no	连接号，成功开卡获得
crd	坐标系号(从 0 开始)
cornerFlag	拐角处理，0 表示处理，1 表示不处理
maxPathError	最大轨迹误差
maxVelLimit	最大速度限制
maxAccLimit	最大加速度限制
返回值	错误码

DWORD MC_GetCrdCornerPrm(WORD ConnectNo, WORD Crd, WORD* cornerFlag, double* maxPathError, double* maxVelLimit, double* maxAccLimit);	
获取前瞻插补的拐角参数	
参数	说明
connect_no	连接号，成功开卡获得



crd	坐标系号(从 0 开始)
cornerFlag	返回是否拐角处理
maxPathError	返回最大轨迹误差
maxVelLimit	返回最大速度限制
maxAccLimit	返回最大加速度限制
返回值	错误码

<pre>DWORD MC_SetLimitArcPrm(WORD connect_no, WORD enable, WORD* axis_list, double* arc_cen, double limit_radius, WORD pos_src , WORD stop_mode);</pre>	
设置圆弧位置限制参数	
参数	说明
connect_no	连接号，成功开卡获得
enable	1 表示使能，0 表示禁止
axis_list	轴列表
arc_cen	圆心位置
limit_radius	圆弧半径
pos_src	位置源，0 表示指令位置，1 表示反馈位置
stop_mode	停止模式，0 表示减速停止，1 表示立即停止
返回值	错误码



DWORD MC_GetLimitArcPrm(WORD connect_no, WORD* enable, WORD* axis_list, double* arc_cen, double* limit_radius, WORD* pos_src , WORD* stop_mode);	
获取圆弧位置限制参数	
参数	说明
connect_no	连接号，成功开卡获得
enable	返回是否使能
axis_list	返回轴列表
arc_cen	返回圆心位置
limit_radius	返回圆弧半径
pos_src	返回位置源，0 表示指令位置，1 表示反馈位置
stop_mode	返回停止模式，0 表示减速停止，1 表示立即停止
返回值	错误码

6.2.5 轴状态和参数相关函数

DWORD MC_Power(WORD connect_no, WORD axis, WORD ifEnable);	
轴使能，在使用轴之前需要先使能轴控制	
参数	说明
connect_no	连接号，成功开卡获得
axis	轴号(从 0 开始)
ifEnable	是否使能；0 表示禁止，1 表示使能



返回值	错误码
-----	-----

DWORD MC_SetPulseMode(WORD connect_no, WORD axis, WORD pulse_mode);	
轴脉冲输出模式设置	
参数	说明
connect_no	连接号, 成功开卡获得
axis	轴号(从 0 开始)
pulse_mode	脉冲输出模式 0 = 脉冲高+方向高 1 = 脉冲低+方向高 2 = 脉冲高+方向低 3 = 脉冲低+方向低 4 = 双脉冲高 5 = 双脉冲低
返回值	错误码

DWORD MC_GetPulseMode(WORD connect_no, WORD axis, WORD* pulse_mode);	
获取轴脉冲输出模式	
参数	说明
connect_no	连接号, 成功开卡获得
axis	轴号(从 0 开始)
pulse_mode	返回脉冲输出模式



返回值	错误码
-----	-----

DWORD MC_SetEncodeMode(WORD connect_no, WORD axis, WORD enc_mode, WORD enc_dir);	
设置轴对应编码器的工作模式及方向	
参数	说明
connect_no	连接号, 成功开卡获得
axis	轴号(从 0 开始)
enc_mode	编码器工作模式 0 = 脉冲方向计数 1 = AB 相计数, 1 倍频 2 = AB 相计数, 2 倍频 3 = AB 相计数, 4 倍频
enc_dir	0 = 计数方向不取反 1 = 计数方向取反
返回值	错误码

DWORD MC_GetEncodeMode(WORD connect_no, WORD axis, WORD* enc_mode, WORD* enc_dir);	
获取轴对应编码器的工作模式及方向	
参数	说明
connect_no	连接号, 成功开卡获得
axis	轴号(从 0 开始)
enc_mode	返回编码器工作模式



enc_dir	返回编码器的计数方向
返回值	错误码

DWORD MC_GetAxisMachineState(WORD connect_no, WORD axis, WORD* Axis_state);	
判断轴是否运动完成	
参数	说明
connect_no	连接号, 成功开卡获得
axis	轴号(从 0 开始)
Axis_state	获取轴的运动状态; 0 表示运行中, 1 表示空闲状态
返回值	错误码

DWORD MC_GetAxisCheckDone(WORD connect_no, WORD axis, WORD* CheckDone);	
判断轴是否运动完成	
参数	说明
connect_no	连接号, 成功开卡获得
axis	轴号(从 0 开始)
Axis_state	返回轴的运动状态 0 = 轴运动中 1 = 轴等待运动中 2 = 402 轴限位 3 = 轴暂停中



	4 = 轴停止过程中 5 = 轴未使能 6 = 轴不存在
返回值	错误码

DWORD MC_GetAxisMotionMode(WORD connect_no, WORD axis, WORD* Motion_Mode);	
获取轴当前的运动模式	
参数	说明
connect_no	连接号, 成功开卡获得
axis	轴号(从 0 开始)
Motion_Mode	获取轴的运动模式 0 = 轴空闲 1 = 点运动 2 = 速度控制运动 3 = 回零运动 4 = 手脉跟随运动 5 = 正弦振荡曲线模式 6 = PT_Move 7 = PVT_Move 129 = 直线插补 130 = 圆弧插补 131 = 椭圆插补 132 = 螺旋线插补 133 = 连续插补 134 = 前瞻插补 135 = 五轴+法向跟随
返回值	错误码



DWORD MC_SetAxisEquvi(WORD connect_no, WORD axis, double equvi);	
设置轴当量(运动目标位置*当量 = 实际脉冲数)	
参数	说明
connect_no	连接号, 成功开卡获得
axis	轴号(从 0 开始)
equvi	轴当量
返回值	错误码

DWORD MC_GetAxisEquvi(WORD connect_no, WORD axis, double* equvi);	
获取轴当量(运动目标位置*当量 = 实际脉冲数)	
参数	说明
connect_no	连接号, 成功开卡获得
axis	轴号(从 0 开始)
equvi	返回轴当量
返回值	错误码

DWORD MC_StopAxis(WORD connect_no, WORD axis, WORD mode, double stop_time);	
停止轴运动	
参数	说明
connect_no	连接号, 成功开卡获得



axis	轴号(从 0 开始)
mode	停止模式; 0 表示减速停止, 1 表示急停
stop_time	减速停止的停止时间
返回值	错误码

DWORD MC_SetAxisStopTime(WORD connect_no, WORD axis, double stop_time);	
设置轴减速停止时间, 单位为 s	
参数	说明
connect_no	连接号, 成功开卡获得
axis	轴号(从 0 开始)
stop_time	减速停止的停止时间
返回值	错误码

DWORD MC_GetAxisStopModeTime(WORD connect_no, WORD axis, WORD* mode, double* stop_time);	
获取轴的停止方式和停止时间	
参数	说明
connect_no	连接号, 成功开卡获得
axis	轴号(从 0 开始)



mode	返回轴停止方式
stop_time	返回轴减速停止的停止时间
返回值	错误码

DWORD MC_StopEmgImd(WORD connect_no);	
急停，所有轴停止运动	
参数	说明
connect_no	连接号，成功开卡获得
返回值	错误码

DWORD MC_GetAxisStopReason(WORD connect_no, WORD axis, DWORD* stop_reason);	
获取轴的停止原因	
参数	说明
connect_no	连接号，成功开卡获得
axis	轴号(从 0 开始)
stop_reason	停止原因 0 = 运动正常停止 1 = 客户调用指令减速停止 2 = Alarm 信号减速停止 3 = 硬件正限位减速停止 4 = 硬件负限位减速停止



	5 = 软件正限位减速停止 6 = 软件负限位减速停止 7 = 区域限位引起停止运动 8 = IO 触发减速停止 9 = 减速暂停 100 = EMG 信号立即停止 101 = 客户调用指令立即停止 102 = Alarm 信号立即停止 103 = 硬件正限位立即停止 104 = 硬件负限位立即停止 105 = 软件正限位立即停止 106 = 软件负限位立即停止 107 = 区域限位硬气停止运动 108 = IO 触发立即停止 109 = 运动中, 掉使能 110 = 运动过大, 异常停止
返回值	错误码

DWORD MC_ClearAxisStopReason(WORD connect_no, WORD axis);	
清除轴的停止原因	
参数	说明
connect_no	连接号, 成功开卡获得
axis	轴号(从 0 开始)
返回值	错误码



DWORD MC_GetAxisCmdPos(WORD connect_no, WORD axis, double* cmd_pos);	
读取轴当前指令位置	
参数	说明
connect_no	连接号, 成功开卡获得
axis	轴号(从 0 开始)
cmd_pos	返回当前指令位置
返回值	错误码

DWORD MC_GetAxisCmdSpeed(WORD connect_no, WORD axis, double* speed);	
返回轴当前运行速度	
参数	说明
connect_no	连接号, 成功开卡获得
axis	轴号(从 0 开始)
speed	返回当前运行速度
返回值	错误码

DWORD MC_GetEncoderPos(WORD connect_no, WORD axis, double* encode	
---	--



r_pos);	
返回编码器反馈位置	
参数	说明
connect_no	连接号, 成功开卡获得
axis	轴号(从 0 开始)
encoder_pos	返回当前编码器位置
返回值	错误码

DWORD MC_GetAxisFeedbackPos(WORD connect_no, WORD axis, double* feedback_pos);	
读取轴的反馈位置	
参数	说明
connect_no	连接号, 成功开卡获得
axis	轴号(从 0 开始)
feedback_pos	返回轴的反馈位置
返回值	错误码

DWORD MC_SetAxisCmdPos(WORD connect_no, WORD axis, double cmd_pos);	
设置轴当前的指令位置	
参数	说明



connect_no	连接号，成功开卡获得
axis	轴号(从 0 开始)
cmd_pos	指令位置
返回值	错误码

DWORD MC_SetAxisSoftELPrm(WORD connect_no, WORD axis, WORD enable, WORD pos_src, double max_pos, double min_pos, WORD stop_mode);	
设置轴的软件限位参数	
参数	说明
connect_no	连接号，成功开卡获得
axis	轴号(从 0 开始)
enable	软件限位是否使能，0 表示禁止，1 表示使能
pos_src	限位位置源；0 表示指令位置，1 表示反馈位置
max_pos	软件正限位位置
min_pos	软件负限位位置
stop_mode	限位发生时轴停止模式 0 = 减速停止 1 = 立即停止
返回值	错误码

DWORD MC_GetAxisSoftELPrm(WORD connect_no, WORD axis, WORD* enable)



le,WORD* pos_src, double* max_pos, double* min_pos, WORD* stop_mode);	
获取轴软件限位参数	
参数	说明
connect_no	连接号，成功开卡获得
axis	轴号(从 0 开始)
enable	返回软件限位是否使能，0 表示禁止，1 表示使能
pos_src	返回限位位置源；0 表示指令位置，1 表示反馈位置
max_pos	返回软件正限位位置
min_pos	返回软件负限位位置
stop_mode	返回限位发生时轴停止模式 0 = 减速停止 1 = 立即停止
返回值	错误码

6.2.6 通用 IO 操作相关函数

DWORD MC_GetInIoBit(WORD connect_no, DWORD io_index, byte* IoState);	
按位读取输入口状态	
参数	说明
connect_no	连接号，成功开卡获得
io_index	输入口索引(从 0 开始)
IoState	返回获取到的状态



返回值	错误码
-----	-----

DWORD MC_GetInIoPort(WORD connect_no, DWORD port_index, DWORD* IoState);	
一次读取 32 位输入口状态	
参数	说明
connect_no	连接号，成功开卡获得
port_index	输入口按 32 位分的索引(所要读取的 IO 口索引/32 取整)
IoState	返回获取到的输入口状态，一次获取 32 位
返回值	错误码

DWORD MC_GetInIoPtr(WORD connect_no, DWORD start_index, DWORD io_num, unsigned long long* IoState);	
按照起始地址读取 IO 口状态，最多一次读取 64 点	
参数	说明
connect_no	连接号，成功开卡获得
start_index	想要读取的输入口起始索引号
io_num	想要读取的输入口数量
IoState	返回获取到的输入口状态，根据你的数量按位分布
返回值	错误码



DWORD MC_SetOutIoBit(WORD connect_no, DWORD io_index, byte IoState);	
按位设置输出口的状态	
参数	说明
connect_no	连接号，成功开卡获得
io_index	输出口的索引(从 0 开始)
IoState	写入输出口的状态; 0 = 低电平 1 = 高电平
返回值	错误码

DWORD MC_SetOutIoPort(WORD connect_no, DWORD port_index, DWORD IoState);	
写通用输出口，一次写 32 位	
参数	说明
connect_no	连接号，成功开卡获得
port_index	输出口按 32 位分的索引，0 代表前 32 位输出口
IoState	写入输出口的状态；32 位数据，位号代表着相同索引的输出口状态
返回值	错误码

DWORD MC_SetOutIoPtr(WORD connect_no, DWORD start_index, byte IoState, unsigned long long IoMask);	
--	--



根据起始输出口索引，连续写多个输出口状态，最多连续写 64 点	
参数	说明
connect_no	连接号，成功开卡获得
start_index	起始索引
IoState	需要写入的 IO 口状态 0 = 低电平 1 = 高电平
IoMask	bit 位为零表示保持原来状态，为 1 表示写入 IoState 状态
返回值	错误码

DWORD MC_GetOutIoBit(WORD connect_no, DWORD io_index, byte* IoState);	
按位读取通用输出口的状态	
参数	说明
connect_no	连接号，成功开卡获得
io_index	输出口的索引(从 0 开始)
IoState	需要写入的 IO 口状态 0 = 低电平 1 = 高电平
返回值	错误码

DWORD MC_GetOutIoPort(WORD connect_no, DWORD port_index, DWORD* IoState);	
---	--



获取通用输出口的状态，一次获取 32 位	
参数	说明
connect_no	连接号，成功开卡获得
port_index	输出口按 32 位分的索引，0 代表前 32 位输出口
IoState	需要写入的 32 位输出口状态，位号对应输出口索引
返回值	错误码

DWORD MC_GetOutIoPtr(WORD connect_no, DWORD start_index, DWORD io_num, unsigned long long* IoState);	
按起始地址获取通用输出口状态，最多一次获取 64 位	
参数	说明
connect_no	连接号，成功开卡获得
start_index	起始索引
io_num	需要读取的数量
IoState	获取输出口状态，bit0 状态代表输出口索引为起始索引的状态
返回值	错误码

DWORD MC_SetReuselIoMap(WORD connect_no, WORD io_index,WORD map_io_type,DWORD map_index);	
复用输入 IO 映射，专用 IO 可通过虚拟 IO 映射，复用	
参数	说明



connect_no	连接号，成功开卡获得
io_index	需要映射的 IO 索引
map_io_type	映射 IO 类型 0 = 轴负限位 1 = 轴正限位 2 = 轴原点信号 3 = 轴伺服到位信号 5 = 轴伺服报警信号 6 = 轴伺服准备信号 9 = 通用 IO 信号
map_index	轴号，或者通用 IO 索引
返回值	错误码

DWORD MC_GetReuselIoMap(WORD connect_no, WORD io_index,WORD* map_io_type,DWORD* map_index);	
获取虚拟 IO 的配置参数	
参数	说明
connect_no	连接号，成功开卡获得
io_index	映射的 IO 索引
map_io_type	返回映射 IO 类型 0 = 轴负限位 1 = 轴正限位 2 = 轴原点信号 3 = 轴伺服到位信号 5 = 轴伺服报警信号 6 = 轴伺服准备信号 9 = 通用 IO 信号
map_index	返回映射索引；根据映射类型判断是轴号还是 IO 索引



返回值	错误码
-----	-----

DWORD MC_GetReuseInIoBit(WORD connect_no, DWORD io_index, byte* IoState);	
按位获取虚拟 IO 状态	
参数	说明
connect_no	连接号, 成功开卡获得
io_index	映射的 IO 索引
IoState	返回虚拟 IO 状态
返回值	错误码

6.2.7 轴控制 IO 信号相关函数

DWORD MC_SetAxisOutIo(WORD connect_no,WORD axis,WORD axis_io_type,WORD io_level);	
设置轴的专用输出口电平	
参数	说明
connect_no	连接号, 成功开卡获得
axis	轴号(从 0 开始)
axis_io_type	轴 IO 类型 0 = 伺服使能 1 = 伺服报警清除
io_level	输出电平 0 = 低电平



	1 = 高电平
返回值	错误码

DWORD MC_GetAxisOutIo(WORD connect_no,WORD axis,WORD axis_io_type,WORD* io_level);	
获取轴的专用输出口电平	
参数	说明
connect_no	连接号, 成功开卡获得
axis	轴号(从 0 开始)
axis_io_type	轴 IO 类型 0 = 伺服使能 1 = 伺服报警清除
io_level	返回获取的输出电平 0 = 低电平 1 = 高电平
返回值	错误码

DWORD MC_SetAxisInIoPrm(WORD connect_no,WORD axis,WORD axis_io_type,WORD enable, WORD io_logical, WORD stop_mode);	
设置轴专用信号逻辑	
参数	说明
connect_no	连接号, 成功开卡获得
axis	轴号(从 0 开始)



axis_io_type	轴 IO 类型 0 = 负限位 1 = 正限位 2 = 原点 3 = 伺服到位 5 = 伺服报警 6 = 伺服准备
enable	信号是否有效 0 = 禁止 1 = 使能
io_logical	信号有效电平 0 = 低电平有效 1 = 高电平有效
stop_mode	信号触发后的轴停止模式 0 = 减速停止 1 = 立即停止
返回值	错误码

DWORD MC_GetAxisInIoPrm(WORD connect_no,WORD axis,WORD axis_io_type,WORD* enable, WORD* io_logical, WORD* stop_mode);	
获取轴专用信号配置	
参数	说明
connect_no	连接号，成功开卡获得
axis	轴号(从 0 开始)
axis_io_type	轴 IO 类型 0 = 负限位 1 = 正限位 2 = 原点 3 = 伺服到位 5 = 伺服报警 6 = 伺服准备
enable	返回信号是否有效 0 = 禁止 1 = 使能



io_logical	返回信号有效电平 0 = 低电平有效 1 = 高电平有效
stop_mode	返回信号触发后的轴停止模式 0 = 减速停止 1 = 立即停止
返回值	错误码

DWORD MC_SetAxisInIoMap(WORD connect_no,WORD axis,WORD axis_io_type,WORD map_io_type,DWORD map_index);	
设置轴专用信号映射	
参数	说明
connect_no	连接号，成功开卡获得
axis	轴号(从 0 开始)
axis_io_type	轴 IO 类型 0 = 负限位 1 = 正限位 2 = 原点 3 = 伺服到位 5 = 伺服报警 6 = 伺服准备
map_io_type	映射 IO 的类型, 包括其他轴的专用信号或者通用 IO 等 0 = 轴负限位 1 = 轴正限位 2 = 轴原点 3 = 轴伺服到位 5 = 轴伺服报警 6 = 轴伺服准备 9 = 通用 IO 信号
map_index	映射索引，根据 map_io_type 判断是轴号还是 IO 口号
返回值	错误码



DWORD MC_GetAxisInIoMap(WORD connect_no, WORD axis,WORD axis_io_type,WORD* map_io_type,DWORD* map_index);	
获取轴专用信号映射	
参数	说明
connect_no	连接号，成功开卡获得
axis	轴号(从 0 开始)
axis_io_type	轴 IO 类型 0 = 负限位 1 = 正限位 2 = 原点 3 = 伺服到位 5 = 伺服报警 6 = 伺服准备
map_io_type	返回映射 IO 的类型 0 = 轴负限位 1 = 轴正限位 2 = 轴原点 3 = 轴伺服到位 5 = 轴伺服报警 6 = 轴伺服准备 9 = 通用 IO 信号
map_index	返回映射索引，根据 map_io_type 判断是轴号还是 IO 口号
返回值	错误码

DWORD MC_GetAxisInIoState(WORD connect_no, WORD axis, DWORD* in_io_state);	
获取轴所有专用信号的状态	



参数	说明
connect_no	连接号，成功开卡获得
axis	轴号(从 0 开始)
in_io_state	返回轴专用信号的状态 Bit0 = 负限位 Bit1 = 正限位 Bit2 = 原点 Bit3 = 伺服到位 Bit5 = 伺服报警 Bit6 = 伺服准备
返回值	错误码

DWORD MC_GetAxisAlarmState(WORD connect_no, WORD axis, byte* alarm_state);	
获取轴的报警信号输入状态	
参数	说明
connect_no	连接号，成功开卡获得
axis	轴号(从 0 开始)
alarm_state	返回轴报警信号的状态
返回值	错误码

DWORD MC_GetAxisHomeState(WORD connect_no, WORD axis, byte* home_state);	
获取轴的原点信号输入状态	



参数	说明
connect_no	连接号，成功开卡获得
axis	轴号(从 0 开始)
home_state	返回轴原点信号状态
返回值	错误码

DWORD MC_GetAxisELState(WORD connect_no, WORD axis, byte* el_minus_state, byte* el_plus_state);	
获取轴的硬件限位信号输入状态	
参数	说明
connect_no	连接号，成功开卡获得
axis	轴号(从 0 开始)
el_minus_state	返回轴硬件负限位状态
el_plus_state	返回轴硬件正限位状态
返回值	错误码

DWORD MC_GetAxisInpState(WORD connect_no, WORD axis, byte* inp_state);	
获取轴的伺服到位信号输入状态	
参数	说明



connect_no	连接号, 成功开卡获得
axis	轴号(从 0 开始)
inp_state	返回轴伺服到位信号状态
返回值	错误码

6.2.8 辅助编码器及手轮相关函数

DWORD MC_SetAuxEncoderPrm(WORD connect_no, WORD channel, WORD mode, WORD dir);	
设置辅助编码器计数参数	
参数	说明
connect_no	连接号, 成功开卡获得
channel	辅助编码器通道,目前只有 1 个通道, 默认 0
mode	辅助编码器计数模式 0 = 脉冲方向计数 1 = AB 相计数, 1 倍频 2 = AB 相计数, 2 倍频 3 = AB 相计数, 4 倍频
dir	辅助编码器计数方向 0 = 计数方向不取反 1 = 计数方向取反
返回值	错误码

DWORD MC_GetAuxEncoderPrm(WORD connect_no, WORD channel, WORD* mode, WORD* dir);	
--	--



获取辅助编码器计数参数	
参数	说明
connect_no	连接号，成功开卡获得
channel	辅助编码器通道,目前只有 1 个通道，默认 0
mode	返回辅助编码器计数模式 0 = 脉冲方向计数 1 = AB 相计数，1 倍频 2 = AB 相计数，2 倍频 3 = AB 相计数，4 倍频
dir	返回辅助编码器计数方向 0 = 计数方向不取反 1 = 计数方向取反
返回值	错误码

DWORD MC_SetAuxEncoderPositon(WORD connect_no, WORD channel, double CmdPos);	
设置辅助编码器位置	
参数	说明
connect_no	连接号，成功开卡获得
channel	辅助编码器通道,目前只有 1 个通道，默认 0
CmdPos	位置值
返回值	错误码

DWORD MC_GetAuxEncoderPositon(WORD connect_no, WORD channel, dou	
--	--



ble* CmdPos);	
获取辅助编码器位置	
参数	说明
connect_no	连接号，成功开卡获得
channel	辅助编码器通道,目前只有 1 个通道，默认 0
CmdPos	返回辅助编码器位置值
返回值	错误码

DWORD MC_SetHandleWheelPrm(WORD connect_no,WORD axis, WORD channel, double multi);	
手轮参数配置	
参数	说明
connect_no	连接号，成功开卡获得
axis	启动手轮运动的轴号(从 0 开始)
channel	使用的手轮通道，默认 0
multi	手轮的计数倍频，大于零为正向计数，小于零为负向计数
返回值	错误码

DWORD MC_GetHandleWheelPrm(WORD connect_no,WORD axis, WORD* channel, double* multi);	
--	--



获取手轮参数配置	
参数	说明
connect_no	连接号，成功开卡获得
axis	启动手轮运动的轴号(从 0 开始)
channel	返回使用的手轮通道
multi	返回手轮的计数倍频
返回值	错误码

DWORD MC_StartHandleWheel(WORD connect_no,WORD axis);	
启动手轮运动，退出手轮运动为调用停止指令	
参数	说明
connect_no	连接号，成功开卡获得
axis	启动手轮运动的轴号(从 0 开始)
返回值	错误码

6.2.9 锁存相关函数

DWORD MC_StartCapture(WORD connect_no, WORD cap_object, WORD cap_axis, WORD cap_chan, WORD cap_io, WORD cap_mode, WORD cap_logic, WORD cap_pos_src);	
启动锁存指令	



参数	说明
connect_no	连接号，成功开卡获得
cap_object	锁存对象 0 = 原点信号 1 = EZ 信号 2 = 高速输入口(IN14、IN15) 3 = 通用输入信号
cap_axis	锁存轴号(从 0 开始)
cap_chan	锁存通道（高速输入口则为 0, 1）,当 cap_object 对象为 Home 和 EZ 时该变量无效
cap_io	锁存触发的 IO 口号，针对 cap_object = 3
cap_mode	锁存模式 0 = 单次锁存 1 = 连续锁存
cap_logic	锁存逻辑 0 = 下降沿捕获 1 = 上升沿捕获 2 = 双边沿捕获
cap_pos_src	锁存位置源 0 = 指令位置 1 = 反馈位置
返回值	错误码

DWORD MC_GetCapturePrm(WORD connect_no, WORD cap_object, WORD cap_axis, WORD cap_chan , WORD* cap_io, WORD* cap_mode, WORD* cap_logic, WORD* cap_pos_src);	
获取锁存通道的配置参数	
参数	说明
connect_no	连接号，成功开卡获得



cap_object	锁存对象 0 = 原点信号 1 = EZ 信号 2 = 高速输入口(IN14、IN15) 3 = 通用输入信号
cap_axis	锁存轴号(从 0 开始)
cap_chan	锁存通道（高速输入口则为 0, 1）,当 cap_object 对象为 Home 和 EZ 时该变量无效
cap_io	返回通道配置的触发 IO 口
cap_mode	返回通道的锁存模式
cap_logic	返回通道的锁存逻辑
cap_pos_src	返回通道的锁存位置源
返回值	错误码

DWORD MC_GetCaptureCount(WORD connect_no, WORD cap_object,WORD cap_axis,WORD cap_chan , DWORD* cap_counts);	
获取锁存状态	
参数	说明
connect_no	连接号，成功开卡获得
cap_object	锁存对象 0 = 原点信号 1 = EZ 信号 2 = 高速输入口(IN14、IN15) 3 = 通用输入信号
cap_axis	锁存轴号(从 0 开始)
cap_chan	锁存通道（高速输入口则为 0, 1）,当 cap_object 对象为 Home 和 EZ 时该变量无效



cap_counts	返回锁存到的锁存点数
返回值	错误码

DWORD MC_GetCapturePos(WORD connect_no, WORD cap_object,WORD cap_axis, WORD cap_chan ,WORD read_cap_counts, WORD* real_counts, double* cap_pos);	
获取锁存位置点	
参数	说明
connect_no	连接号，成功开卡获得
cap_object	锁存对象 0 = 原点信号 1 = EZ 信号 2 = 高速输入口(IN14、IN15) 3 = 通用输入信号
cap_axis	锁存轴号(从 0 开始)
cap_chan	锁存通道（高速输入口则为 0, 1）,当 cap_object 对象为 Home 和 EZ 时该变量无效
read_cap_counts	需要读取的位置点个数
real_counts	返回实际读取到的位置点个数
cap_pos	返回位置点数组
返回值	错误码

DWORD MC_ReStartCapture(WORD connect_no, WORD cap_object, WORD c	
--	--



ap_axis, WORD cap_chan);	
重启锁存器	
参数	说明
connect_no	连接号，成功开卡获得
cap_object	锁存对象 0 = 原点信号 1 = EZ 信号 2 = 高速输入口(IN14、IN15) 3 = 通用输入信号
cap_axis	锁存轴号(从 0 开始)
cap_chan	锁存通道（高速输入口则为 0, 1）,当 cap_object 对象为 Home 和 EZ 时该变量无效
返回值	错误码

DWORD MC_StopCapture(WORD connect_no,WORD cap_object, WORD cap_axis, WORD cap_chan);	
停止锁存	
参数	说明
connect_no	连接号，成功开卡获得
cap_object	锁存对象 0 = 原点信号 1 = EZ 信号 2 = 高速输入口(IN14、IN15) 3 = 通用输入信号
cap_axis	锁存轴号(从 0 开始)
cap_chan	锁存通道（高速输入口则为 0, 1）,当 cap_object 对象为 Home 和 EZ 时该变量无效
返回值	错误码



6.2.10 比较功能相关函数

DWORD MC_StopHSCmp(WORD connect_no, WORD cmp_dim, WORD cmp_chan);	
停止高速比较口功能	
参数	说明
connect_no	连接号，成功开卡获得
cmp_dim	比较维数 1 = 一维比较 2 = 二维比较
cmp_chan	高速比较口 0~3;对应 Output12~15
返回值	错误码

DWORD MC_GetHSCmpPrm(WORD connect_no,WORD cmp_dim, WORD cmp_hs_io, WORD* cmp_mode, WORD* axis, WORD* cmp_source, WORD* trigger_mode, DWORD* trigger_Data);	
获取高速比较通道参数配置	
参数	说明
connect_no	连接号，成功开卡获得
cmp_dim	比较维数 1 = 一维比较 2 = 二维比较
cmp_hs_io	高速比较口 0~3;对应 Output12~15



cmp_mode	返回比较输出模式 0 = 小于比较点模式 1 = 大于比较点模式 2 = 等增量比较模式 3 = FIFO 点比较模式
axis	返回比较轴号
cmp_source	返回比较位置源 0 = 指令位置 1 = 反馈位置
trigger_mode	返回比较触发模式 0 = 低电平触发 1 = 高电平触发 3 = 输出脉冲 4 = 停止轴
trigger_Data	返回触发参数; 触发模式为 0~3 时对应 IO 序号 index 为 12~15 触发模式为 4 时, 该参数为轴号
返回值	错误码

DWORD MC_StartHS1DCmp(WORD connect_no, WORD cmp_hs_io, WORD cmp_axis, WORD pos_src, DWORD cmp_counts, double* cmp_pos, WORD cmp_mode, WORD trigger_mode, DWORD trigger_prm, double hs_out_time);	
启动高速比较口一维比较	
参数	说明
connect_no	连接号, 成功开卡获得
cmp_hs_io	高速比较口 0~3;对应 Output12~15
cmp_axis	比较轴号
pos_src	比较位置源 0 = 指令位置 1 = 反馈位置



cmp_counts	比较位置点数(模式 0~2 下, 该值为固定值 1)
cmp_pos	比较位置点数组(等增量比较模式下, 该数组长度为 2, 第一点为起点位置点; 第二点为增量值)
cmp_mode	比较输出模式 0 = 小于比较点模式 1 = 大于比较点模式 2 = 等增量比较模式 3 = FIFO 点比较输出
trigger_mode	比较触发模式 0 = 低电平触发 1 = 高电平触发 3 = 输出脉冲 4 = 停止轴
trigger_prm	触发参数; 触发模式为 0~3 时对应 IO 序号 index 为 12~15 触发模式为 4 时, 该参数为轴号
hs_out_time	当 trigger_mode 为 3 时的输出时间
返回值	错误码

DWORD MC_GetHSCmpStatus(WORD connect_no, WORD cmp_dim, WORD cmp_hs_io, WORD* enable, double* running_pos, DWORD* completed_points, DWORD* remained_points);	
获取高速比较状态	
参数	说明
connect_no	连接号, 成功开卡获得
cmp_dim	比较维数 1 = 一维比较 2 = 二维比较
cmp_hs_io	高速比较口 0~3;对应 Output12~15



enable	返回比较器使能状态 0 = 未启用 1 = 使能
running_pos	返回当前比较点的位置信息
completed_points	返回完成的比较点个数
remained_points	返回剩余比较点个数
返回值	错误码

DWORD MC_ClearHSCmp(WORD connect_no, WORD cmp_dim, WORD cmp_hs_io);	
清除比较结果	
参数	说明
connect_no	连接号，成功开卡获得
cmp_dim	比较维数 1 = 一维比较 2 = 二维比较
cmp_hs_io	高速比较口 0~3;对应 Output12~15
返回值	错误码

DWORD MC_StopSoftCmp(WORD connect_no,WORD cmp_dim, WORD cmp_chan);	
停止软件比较	
参数	说明



connect_no	连接号，成功开卡获得
cmp_dim	比较维数 1 = 一维比较 2 = 二维比较
cmp_chan	比较通道 一维软件比较取值范围[0,3] 二维软件比较取值范围[0,1]
返回值	错误码

DWORD MC_GetSoftCmpPrm(WORD connect_no,WORD cmp_dim, WORD cmp_hs_io, WORD* cmp_mode, WORD* axis, WORD* cmp_source, WORD* trigger_mode, DWORD* trigger_Data);	
获取软件比较参数	
参数	说明
connect_no	连接号，成功开卡获得
cmp_dim	比较维数 1 = 一维比较 2 = 二维比较
cmp_hs_io	比较通道 一维软件比较取值范围[0,3] 二维软件比较取值范围[0,1]
cmp_mode	返回比较输出模式 0 = 小于比较点模式 1 = 大于比较点模式 2 = 等增量比较模式 3 = FIFO 点比较输出模式
axis	返回比较轴，二维比较时数组长度为 2



cmp_source	返回比较位置源 0 = 指令位置 1 = 反馈位置
trigger_mode	返回比较触发模式 0 = 低电平 1 = 高电平 3 = 输出脉冲 4 = 停止轴
trigger_Data	返回触发参数
返回值	错误码

<pre> DWORD MC_GetSoftCmpStatus(WORD connect_no,WORD cmp_dim, WORD c mp_chan, WORD* enable, double* running_pos, DWORD* completed_points, DWORD* remained_points); </pre>	
获取软件比较状态信息	
参数	说明
connect_no	连接号，成功开卡获得
cmp_dim	比较维数 1 = 一维比较 2 = 二维比较
cmp_chan	比较通道 一维软件比较取值范围[0,3] 二维软件比较取值范围[0,1]
enable	返回比较器使能状态 0 表示未启用 1 表示使能
running_pos	返回当前比较位置点，二维比较则为两个轴位置
completed_points	返回已经完成比较点数



remained_points	返回剩余比较点数
返回值	错误码

DWORD MC_ClearSoftCmp(WORD connect_no,WORD cmp_dim, WORD cmp_chan);	
清除软件比较	
参数	说明
connect_no	连接号，成功开卡获得
cmp_dim	比较维数 1 = 一维比较 2 = 二维比较
cmp_chan	比较通道 一维软件比较取值范围[0,3] 二维软件比较取值范围[0,1]
返回值	错误码

DWORD MC_StartSoft1DCmp(WORD connect_no,WORD cmp_chan, WORD cmp_axis, WORD pos_src, DWORD cmp_counts, double* cmp_pos, WORD cmp_mode, WORD trigger_mode, DWORD trigger_prm, double out_time);	
启动一维软件比较	
参数	说明
connect_no	连接号，成功开卡获得
cmp_chan	比较通道 一维软件比较取值范围[0,3]



	二维软件比较取值范围[0,1]
cmp_axis	比较轴号
pos_src	比较位置源 0 = 指令位置 1 = 反馈位置
cmp_counts	保留参数，默认为 0
cmp_pos	比较点位置
cmp_mode	比较输出模式 0 = 小于比较点模式 1 = 大于比较点模式 2 = 等增量比较模式 3 = FIFO 点比较输出
trigger_mode	比较触发模式 0 = 低电平 1 = 高电平 2 = 电平翻转 3 = 输出脉冲 4 = 停止轴
trigger_prm	比较触发参数 trigger_mode 为 0~3 时，该参数为 IO 索引 trigger_mode 为 4 时，该参数为轴号
out_time	当 trigger_mode 为 3 时 IO 输出时间
返回值	错误码

DWORD MC_StartSoft2DCmp(WORD connect_no, WORD chan, WORD* axis, WORD* pos_src, DWORD cmp_count, double* pos, WORD* cmp_mode, WO RD trigger_mode, DWORD trigger_prm, double out_time);	
启动二维维软件比较	
参数	说明



connect_no	连接号，成功开卡获得
cmp_chan	比较通道 一维软件比较取值范围[0,3] 二维软件比较取值范围[0,1]
cmp_axis	比较轴号，数组长度为 2
pos_src	比较位置源，数组长度为 2 0 = 指令位置 1 = 反馈位置
cmp_counts	保留参数，默认为 0
pos	比较点位置，数组长度为 2
cmp_mode	比较输出模式，数组长度为 2 0 = 小于比较点模式 1 = 大于比较点模式 2 = 等增量比较模式 3 = FIFO 点比较输出
trigger_mode	比较触发模式 0 = 低电平 1 = 高电平 2 = 电平翻转 3 = 输出脉冲 4 = 停止轴
trigger_prm	比较触发参数 trigger_mode 为 0~3 时，该参数为 IO 索引 trigger_mode 为 4 时，该参数为轴号
out_time	当 trigger_mode 为 3 时 IO 输出时间
返回值	错误码

6.2.11 PWM 相关函数

```
DWORD MC_StartPwm(WORD connect_no, WORD pwm_chan_type, WORD p
```



wm_chan, WORD enable,WORD init_pin, double frequency, double duty);	
启动 PWM 输出	
参数	说明
connect_no	连接号，成功开卡获得
pwm_chan_type	保留参数，默认为 1
pwm_chan	输出通道，取值范围[0,3];对应 output12-15
enable	是否使能输出 0 = 禁止 1 = 使能
init_pin	初始电平 0 = 低电平 1 = 高电平
frequency	输出频率，单位 Hz
duty	占空比，取值范围[0,1];duty = (高电平时刻/脉冲总时刻)
返回值	错误码

DWORD MC_GetPwmPrm(WORD connect_no,WORD pwm_chan_type, WORD pwm_chan, WORD* enable, WORD* init_pin, double* frequency, double* duty);	
获取 PWM 输出参数	
参数	说明
connect_no	连接号，成功开卡获得
pwm_chan_type	保留参数，默认为 1
pwm_chan	输出通道，取值范围[0,3];对应 output12-15



enable	返回通道是否使能输出
init_pin	返回 PWM 的初始电平 0 = 低电平 1 = 高电平
frequency	返回输出频率，单位 Hz
duty	返回输出占空比
返回值	错误码

6.3 错误码

错误分类	错误码	描述
	0	函数执行成功
普通错误	10001	从卡上读回来的卡号超过最大限制
	10002	从卡上读回来的卡号有重复设置
	10003	参数错误
	10004	不支持这个功能
	10005	连接号没有配置参数
	10006	固件文件错误
	10007	文件名太长
	10008	产品不存在
	10009	固件文件不匹配
	10010	收到的数据包长度错误
	10011	发送的数据长度超过最大值限制



	10012	输入指针长度，与数据不符合
输入错误	10101	输入的轴数超过最大限制
	10102	输入参数的地址为空指针
	10103	输入参数的 vmove 的方向无效
	10104	输入的卡号超过最大或最小限制
	10105	搜索 EtherNET 时间超时
	10106	输入的输出 IO 数超过最大限制
	10107	输入的输入 IO 数超过最大限制
	10108	输入的控制器 ID 显示在运行中
	10109	仿真模式下，不支持该功能
	10110	坐标系维数错误
PCI 相关错误	10200	开卡失败，或者开卡不成功
	10201	开卡时打开互斥信号量失败
	10202	开卡时创建共享内存失败
	10203	关卡失败
	10204	超过最大传输数据长度
	10205	超过最大卡号限制
	10206	arm 端处理数据超时
	10207	发送数据超时，可能 PCI 的驱动安装不正确
	10208	需要发送的数据长度超过最大长度限制
	10209	读写双口 RAM 的数据



	10210	IO 模块通讯错误
圆弧限位相关 错误	101	圆弧限位轴数超过 2 个轴
	102	目标位置超过圆弧限位值
	103	运动中不允许设置参数
	104	限位圆弧半径太小
回零错误	201	没有回零信号存在
	202	限位回零时，运动方向不回来方向不匹配
	203	回零过程限位信号异常
	204	没有 EZ 信号存在
	206	回零异常停止
	208	回零模式参数不支持
单轴运动错误	301	单轴运动启动时，多线程异常
	302	多段运动启动时，两段位置运动方向不同
	303	多段运动启动时，有距离为 0 段
控制指令错误	403	轴运动中，不允许操作
	404	轴数超过最大值限制
	405	软件限位中
	406	硬件限位中
	407	报警信号有效
	408	运动阶段不支持操作
	409	运动模式不支持



	410	多个指令被执行
	411	高优先级线程抢占了绝大部分时间, 更新数据异常
	412	设置的位置超过最大值
	413	轴未使能
	414	两点的位置不在同一个运动方向
	415	两点的位置不支持相对运动的在线变速变位
	416	运动的启动模式错误
	417	功能不支持
	418	圆弧等的参数错误
	419	输入的轴数错误
	420	当量设置超过设置范围
	421	当量手轮倍频为零
轴控制错误	504	插补器错误最大值
	509	轴未是使能状态
	510	配置的切换状态机有问题
	511	配置的插补器类型错误
	512	轴不在插补器中
捕获功能相关 错误	600	捕获超过最大通道数
	601	捕获模式不存在
	602	捕获 buffer 数据已满
	603	捕获触发模式不存在



	604	配置捕获轴数为零
	605	捕获事件不存在
	606	捕获通道与轴不匹配
	607	捕获通道的轴不存在
比较功能相关 错误	700	比较模式超过最大通道数
	701	比较模式不存在
	702	比较 buffer 数据已满
	703	比较触发模式不存在
	704	错误配置比较轴号
通讯协议相关 错误	801	通讯协议不支持
	802	文件超过最大值
	803	该类文件下载不允许轴运动
	804	文件发送未完成
	805	下载文件类型错误
	806	文件下载数据包代号错误
	807	下载文件的总大小与分包后的总大小错误
	808	文件包的数据异常
	809	spi 的文件数据读写错误
多轴运动相关 错误	1001	多轴运动超出最大允许轴数
	1002	不支持的功能
	1003	插补运动未启动



	1004	轴不存在插补器中
多轴插补器指令错误	1101	插补器运动中
	1102	多轴插补器指令超出最大允许轴数
	1103	多轴插补器不支持的功能
多轴插补算法相关错误	1203	坐标系位置错误
	1204	坐标系维数错误
	1205	坐标系原点异常
	1206	坐标三点在一条直线
	1207	空间圆弧终点半径错误
	1208	输入参数不能构成圆弧
	1209	输入圆弧方向
	1210	某一个轴的半径为零，转为直线
多轴插补器运动错误	1308	多轴插补器运动超出最大允许轴数
手轮相关错误	1401	配置的手轮通道超过最大值
	1402	手轮配置的轴号超过最大值
	1403	配置的手轮倍频为零
	1404	通道中轴运动中
IO 控制相关错误	1501	out 口超过最大值
	1502	in 口超过最大值
	1503	io 映射参数错误



	1504	IO 计数未使能
	1505	fpga 通道为 0~7
	1506	配置的触发后动作不存在
	1507	配置的 IO 类型不存在
	1508	超过最大通道数
运动仿真相关 错误	1601	接收的数据长度不对
	1602	发送的数据长度不对
	1603	内存分配错误
日志打印错误 码	1701	配置 Log 打印错误