
PMC1000-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	2023/07/14

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

室 电话：+86 (0)755 29503086

传真：+86 (0)755 23327086

邮箱：sales@szruiotech.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

联系人：唐女士

电话：15202122728

邮箱：sales03@szruiotech.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@szruiotech.com

网址：www.szruiotech.com



目录

第 1 章	概述	6
1.1	产品型号说明.....	6
第 2 章	硬件及驱动程序安装.....	7
2.1	硬件安装步骤.....	7
2.1.1	拨码开关设置	7
2.1.2	硬件安装	8
2.2	驱动程序安装.....	9
2.2.1	Windows10.....	9
2.2.2	Windows7.....	13
第 3 章	软件演示.....	19
3.1	轴操作	19
3.2	IO 操作	20
第 4 章	应用程序开发	21
4.1	基于 Windows 平台的应用软件架构.....	21
4.2	Visual C++6.0	23
4.3	Visual Studio	24
4.4	Visual C#	26
第 5 章	运动功能详解及实现.....	28
5.1	初始化、关闭运动控制卡	28
5.2	设置脉冲输出模式	29
5.2.1	脉冲/方向模式.....	29
5.2.2	双脉冲模式	30
5.3	单轴位置和速度运动	31
5.3.1	梯形速度曲线运动.....	31
5.3.2	S 型速度曲线运动.....	32
5.3.3	速度运动	33



5.3.4	直线插补运动	33
5.3.5	回原地运动	36
5.3.6	脉冲指令计数	37
5.4	通用 I/O 控制	37
5.5	轴专用 IO 控制	41
5.6	软件限位	42
5.7	专用信号复用	43
第 6 章	函数库详解	45
6.1	函数列表	45
6.2	函数说明	47
6.2.1	连接控制器操作	47
6.2.2	控制器信息相关函数	48
6.2.3	单轴运动控制相关函数	51
6.2.4	轴状态和参数相关函数	56
6.2.5	通用 IO 操作相关函数	60
6.2.6	轴专用信号相关函数	64
第 7 章	附录	67
7.1	函数返回错误码	67

第1章 概述

PMC1000-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、6 轴、12 轴选择方案。其位置指令可用单路脉冲（脉冲+方向）或双路脉冲（CW+CCW 脉冲）方式输出；可以是差分式输出电路也可以是单端式输出电路。另外，除了通用输入输出信号接口外，还包括原点、限位、减速等专用信号接口，具有即插即用、多轴同时同步启停功能，并可选择梯形或 S 形速度曲线，同时具有软件直线插补功能。

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 PMC1000-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
PMC1008-P	P	P 表示普通脉冲款运动控制卡
	MC	MC 为我司控制卡产品代码
	1	1 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
	08	这两位表示轴数 04 表示 4 轴版本 06 表示 6 轴版本 08 表示 8 轴版本 12 表示 12 轴版本
	-P	-P 表示和 PC 的通讯方式为 PCI 总线

第2章 硬件及驱动程序安装

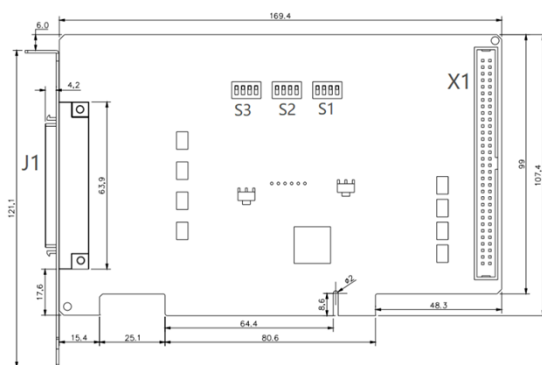
在拿到 PMC1000-P 系列运动控制卡之后需要对硬件进行如下检查：

- ◆ 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

2.1 硬件安装步骤

2.1.1 拨码开关设置

PMC1000-P 系列运动控制卡正面提供了 3 个拨码开关，分别是 S1、S2、S3。



其中 S1 提供控制卡号的拨码，拨码开关四个拨码采用的是 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：

表 2.1 拨码组合对应卡号表

S1-4	S1-3	S1-2	S1-1	控制卡号
OFF	OFF	OFF	OFF	0
OFF	OFF	OFF	ON	1
OFF	OFF	ON	OFF	2
OFF	OFF	ON	ON	3

OFF	ON	OFF	OFF	4
OFF	ON	OFF	ON	5
OFF	ON	ON	OFF	6
OFF	ON	ON	ON	7
ON	OFF	OFF	OFF	8
ON	OFF	OFF	ON	9
ON	OFF	ON	OFF	10
ON	OFF	ON	ON	11
ON	ON	OFF	OFF	12
ON	ON	OFF	ON	13
ON	ON	ON	OFF	14
ON	ON	ON	ON	15

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

拨码开关 S3 能够设置通用输出口的初始电平状态，拨至 ON 时输出初始电平为低，选择 OFF 时输出初始电平为高。拨码对应的输入口如下表所示：

表 2.2 PMC1000-P 输出口拨码开关表

拨码号	输出口号	高电平	低电平
S3-1	OUT1-OUT8	OFF(出厂设置)	ON
S3-2	OUT9-OUT16	OFF(出厂设置)	ON
S3-3	OUT17-OUT24	OFF(出厂设置)	ON
S3-4	OUT25-OUT32	OFF(出厂设置)	ON

2.1.2 硬件安装

具体安装步骤如下：

- 1) PC 机箱接地并关闭电源



- 2) 根据需求设置板卡拨码
- 3) 将控制卡插入主机机箱中的 PCI 插槽中
- 4) 拧紧螺丝，保证控制卡和 PCI 插槽接触良好稳定
- 5) 将接线板用电缆线与控制卡对应的插座连接，并确保连接牢固可靠。

注意：

- 1) 在使用第二个接线板的时候，需要将 2 个接线板的 5V 和 GND 连接在一起。
- 2) 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。
- 3) 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。

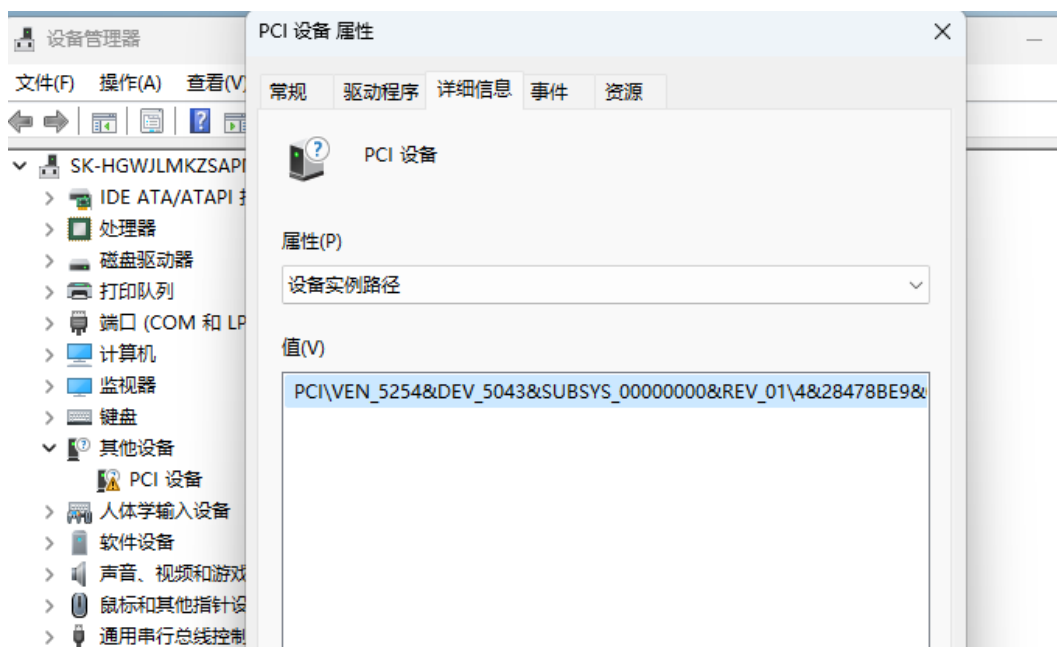
2.2 驱动程序安装

PMC1000-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。

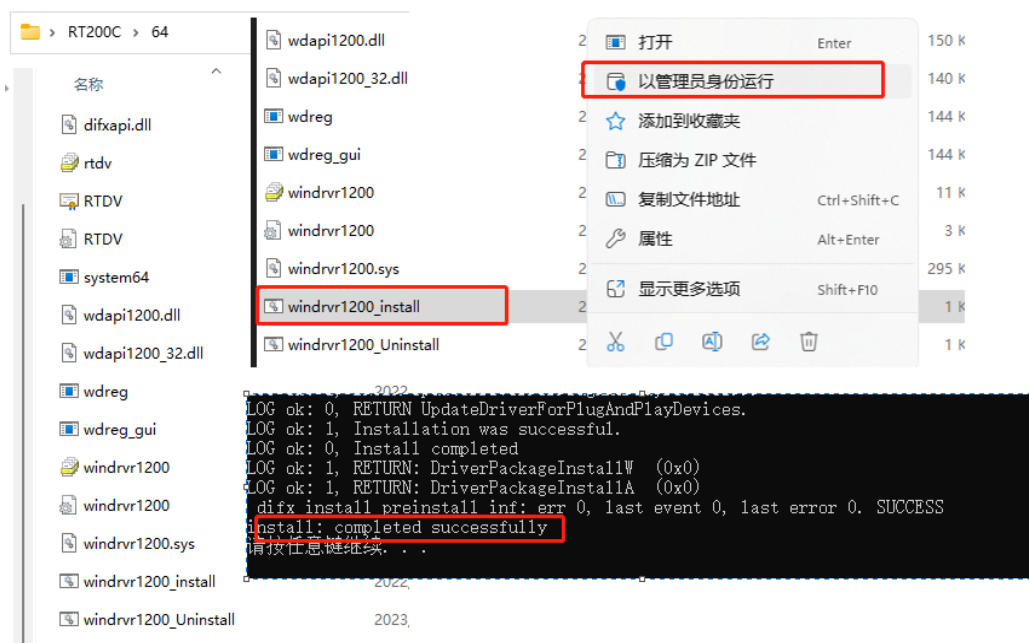
我司提供的开发套件文件夹名称使用 32/64 来区分不同系统位数的驱动文件。若存在驱动安装问题请与我司技术支持联系，我司将竭诚为您服务。

2.2.1 Windows10

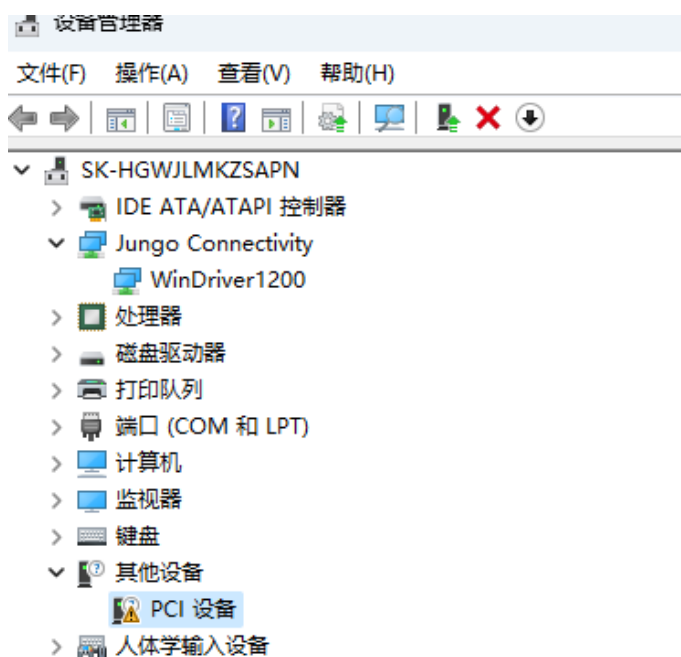
- 1) 在将板卡插入计算机后；鼠标右键单击“计算机”->选择“设备管理器”，在设备管理器中是否找得到黄色感叹号的“PCI 设备”选项，该设备的详细信息中 VEN_5254,DEV_5043 代表着该设备的厂商代码和设备代码。如下图所示。



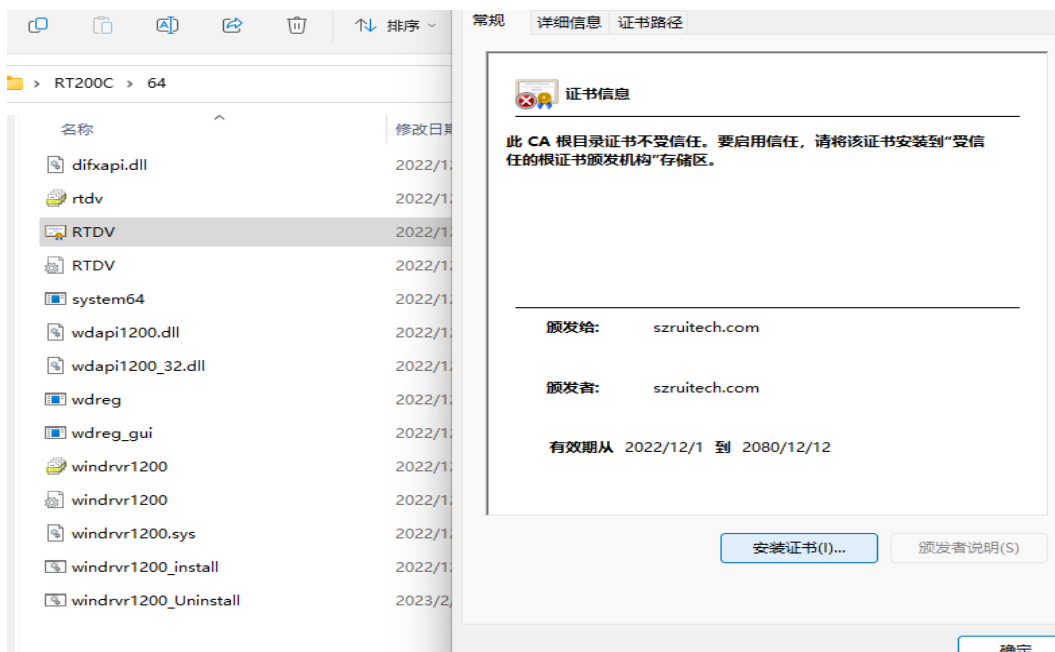
- 2) 找到提供的板卡驱动程序，根据系统位数进入相关的驱动安装文件夹；
找到文件“windrvr1200_install”；右键以管理员权限运行该批处理文件，安装成功会以“completed successfully”提示。



- 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

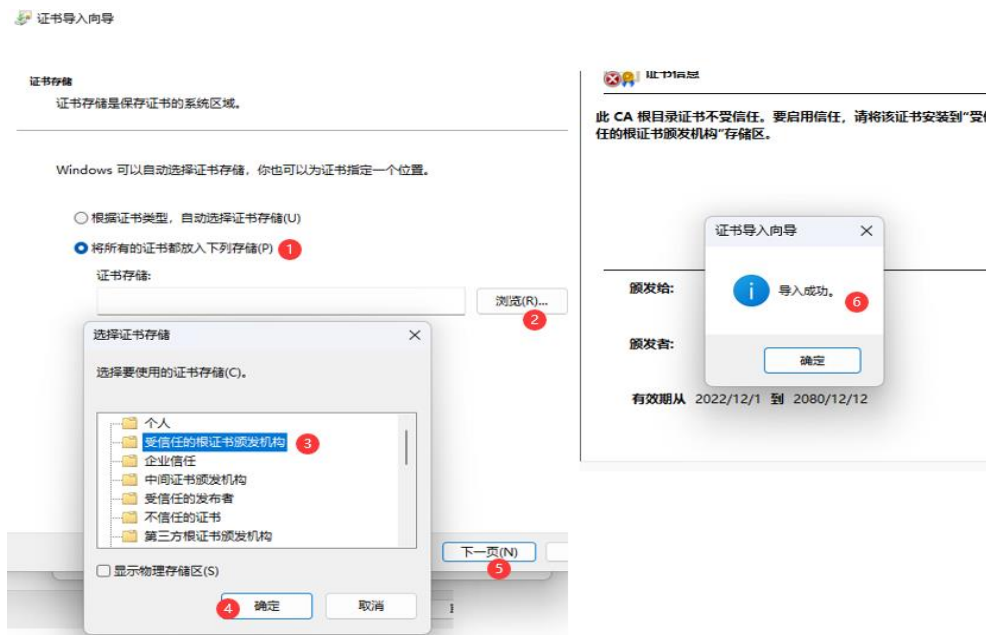
- ☐ 当前用户(C)
- ☒ 本地计算机(L)

单击“下一步”继续。

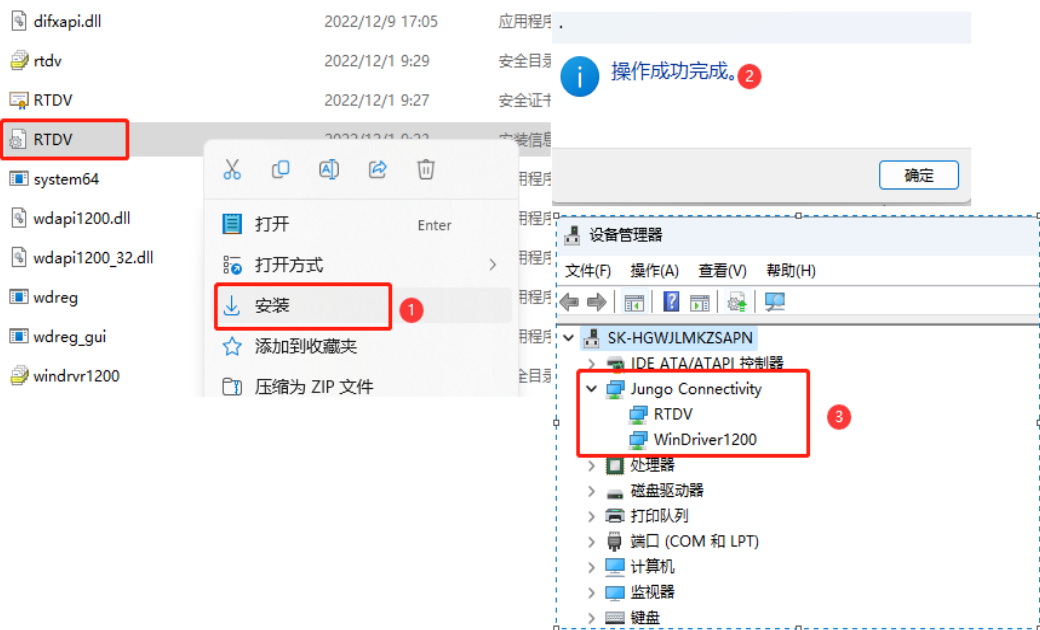
下一页(N)

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。

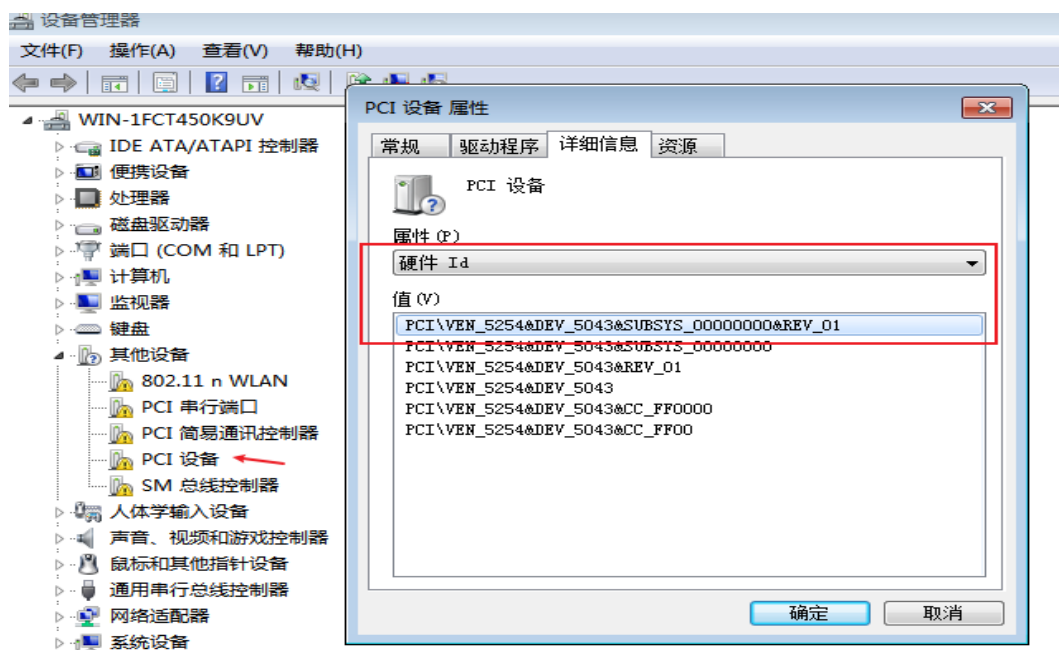


- 7) 在安装完证书之后，找到驱动文件，点击右键“安装”，选择“安装设备”；成功安装提示“操作成功完成”；这是在设备管理器中能够看到设备驱动已经成功安装，能够正常使用。

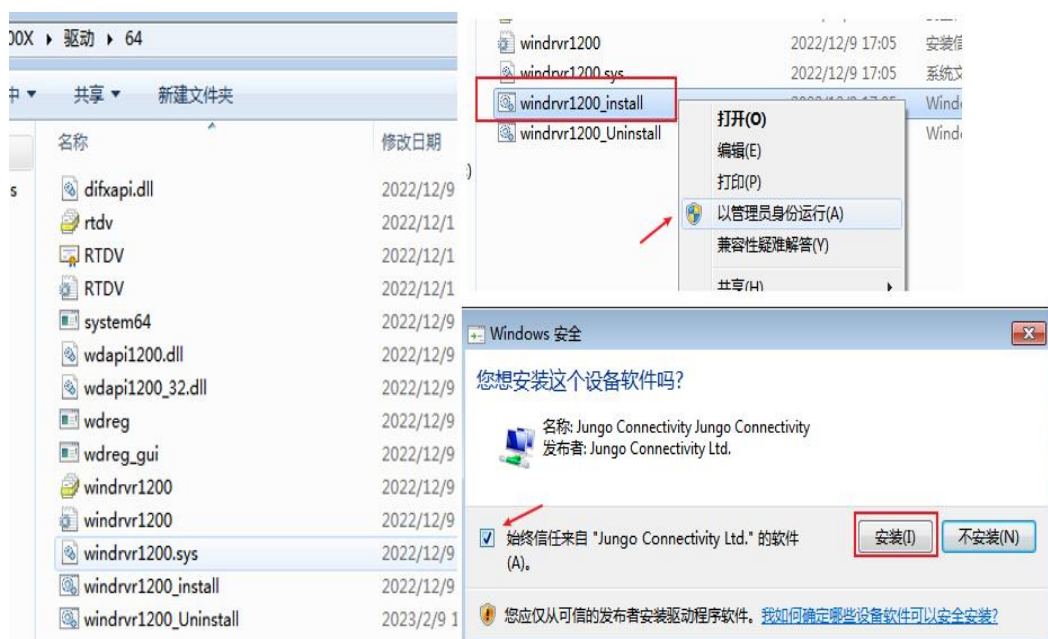


2.2.2 Windows7

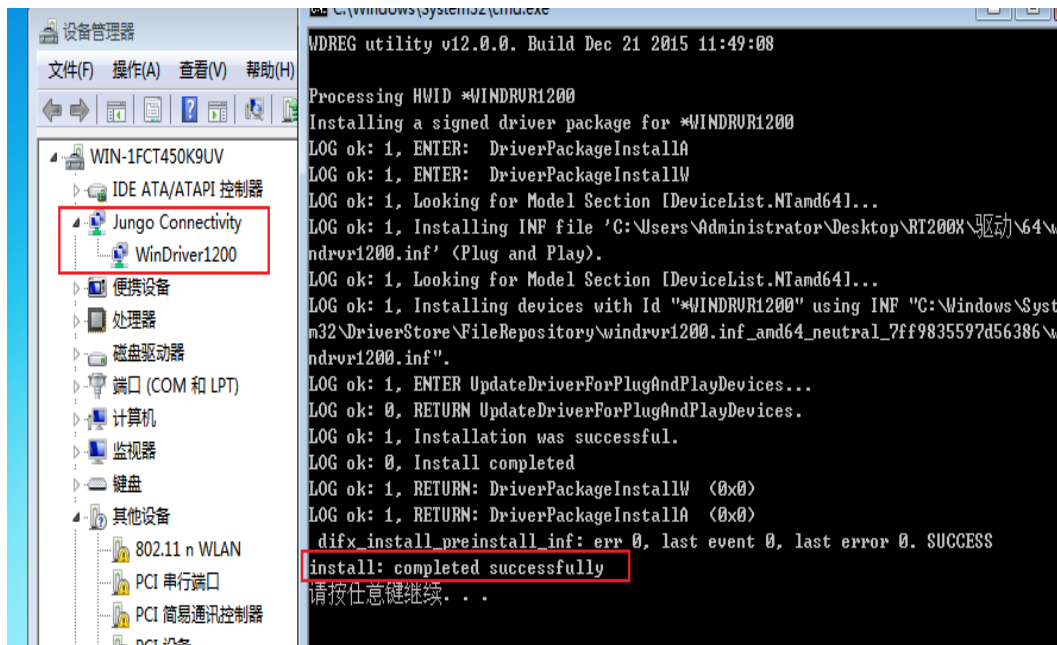
- 1) 在将板卡插入计算机后；鼠标右键单击“计算机”->选择“设备管理器”，在设备管理器中是否找得到黄色感叹号的“PCI 设备”选项，该设备的详细信息中 VEN_5254,DEV_5043 代表着该设备的厂商代码和设备代码。如下图所示。



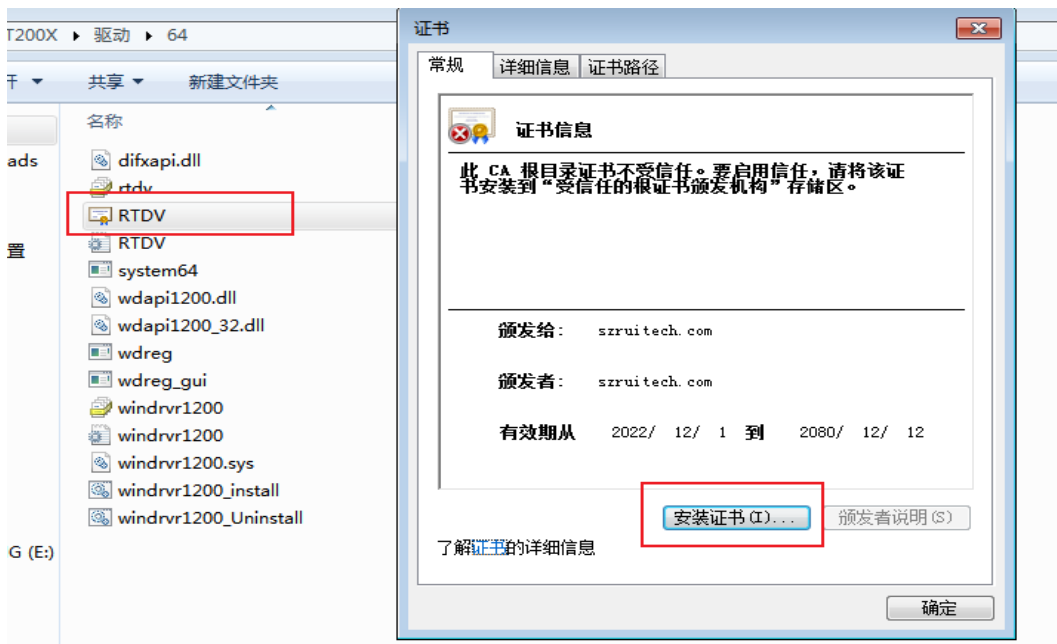
- 2) 在驱动文件路径下找到”windrvr1200_install”批处理文件，然后右键以管理员身份运行，在弹出的设备安装提示中选择“安装”。



- 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。



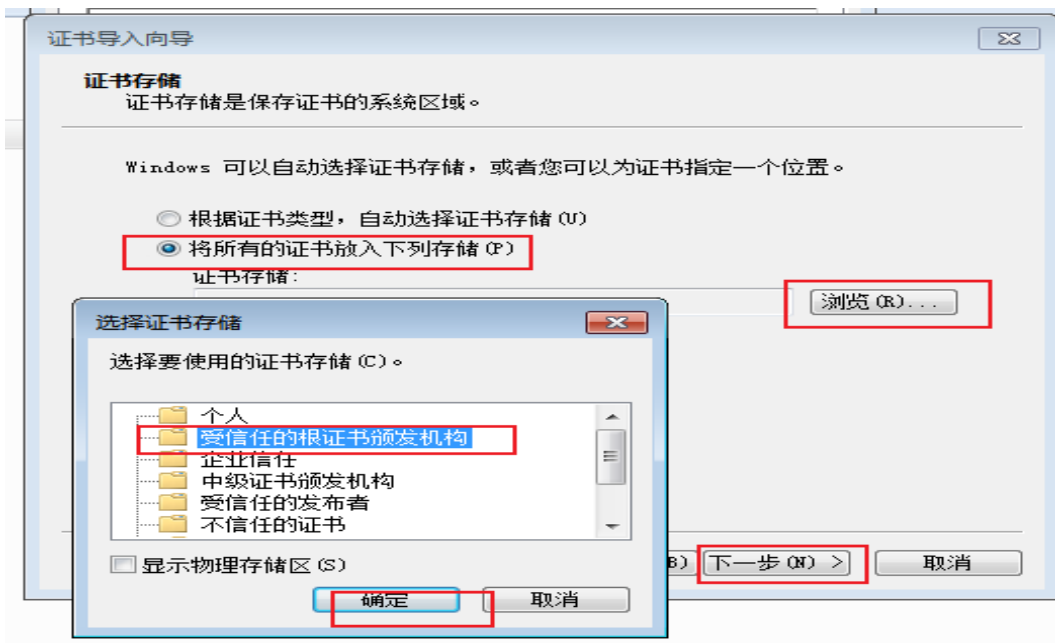
- 4) 在驱动文件路径下找到 RTDV 证书文件，打开之后点击“安装证书”。



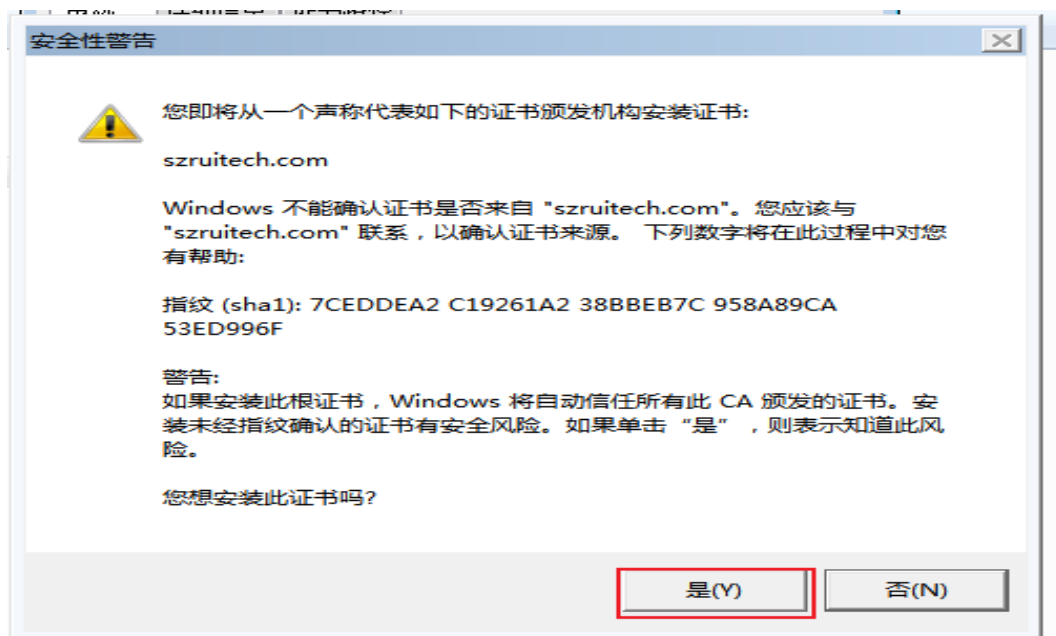
- 5) 在证书储存页面中，选择“将所有的证书放入下列储存”；然后点击“浏



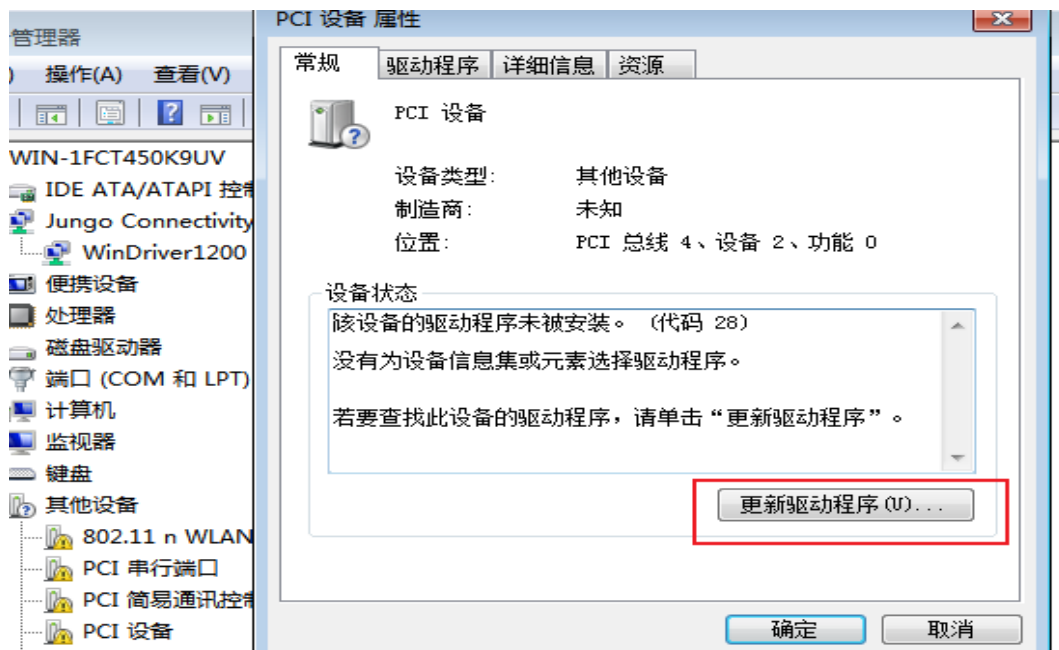
览”；选择要使用的证书储为“受信任的根证书颁发机构”；点击确定；再点击下一步。



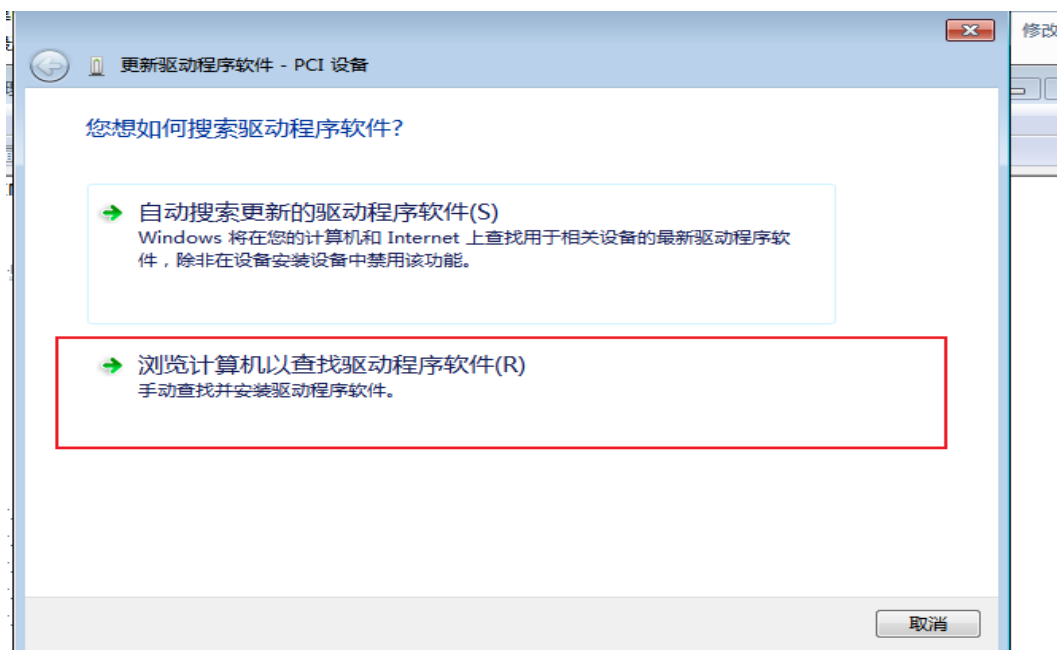
- 6) 在点击证书导入向导“完成”之后，出现证书的安全性警告，选择“是”。之后证书导入成功。



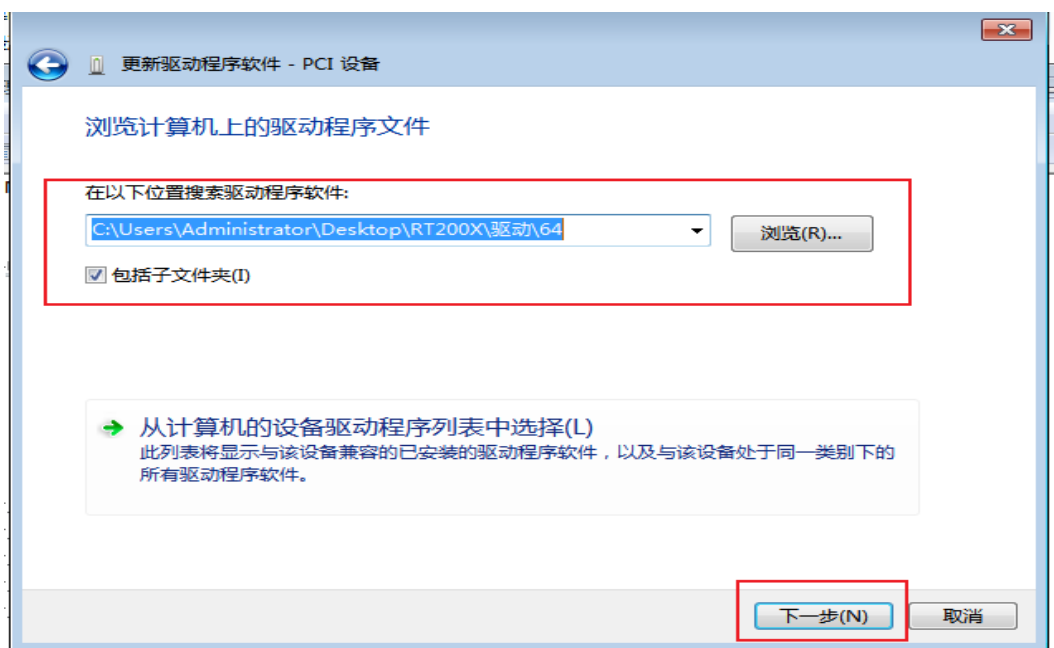
- 7) 在设备管理器中找到黄色感叹号的 PCI 设备；在设备属性中点击“更新驱动程序”。



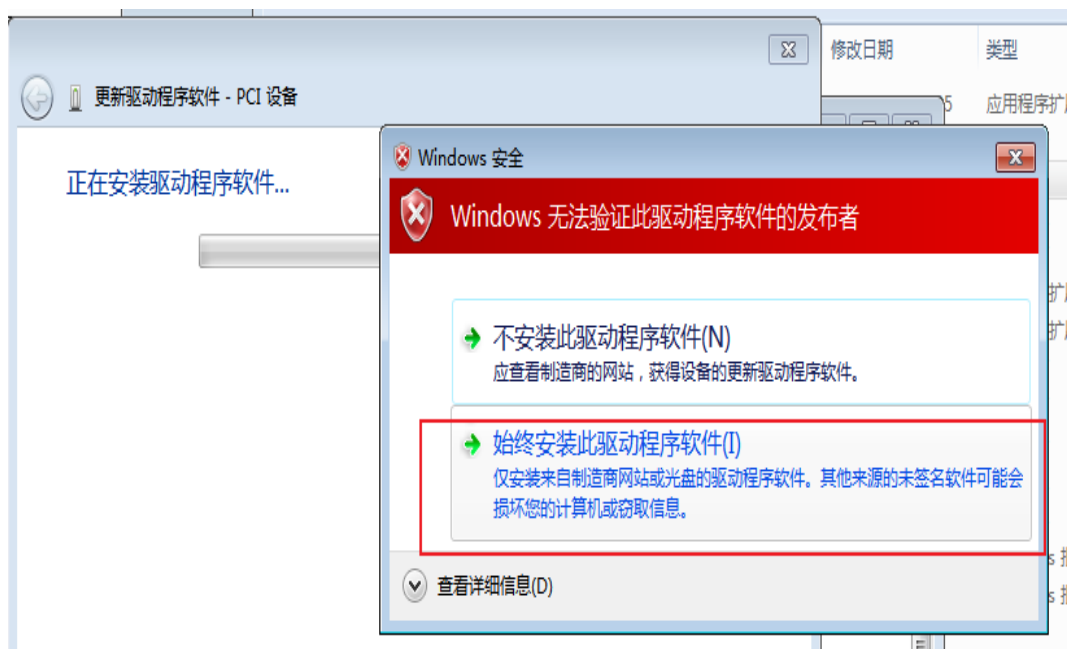
8) 选择“浏览计算机以查找驱动程序软件”。



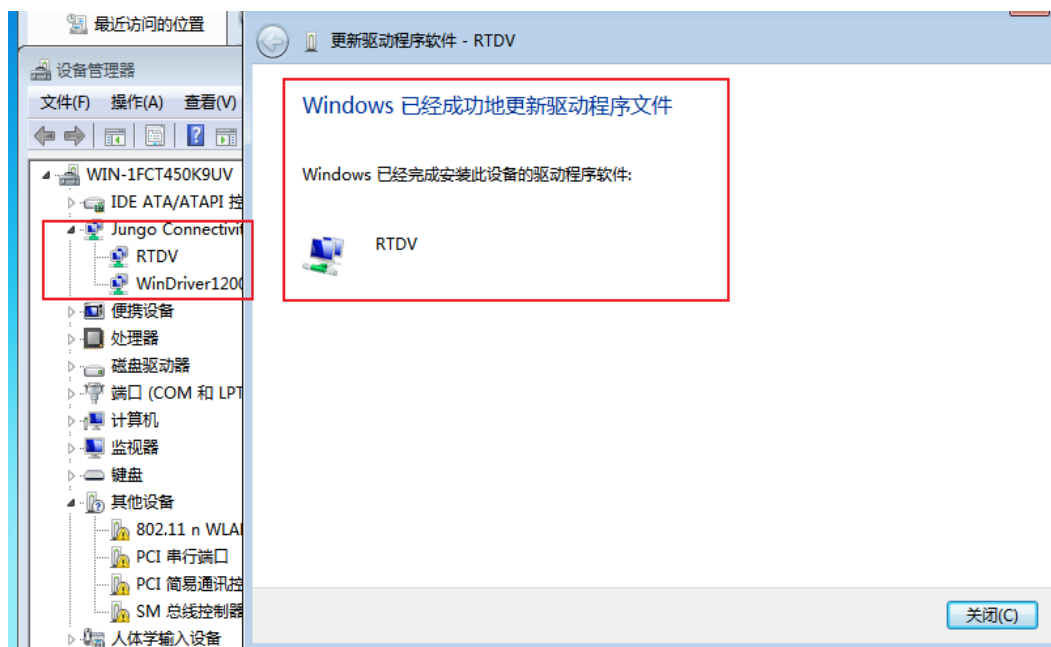
9) 在驱动文件路径中选择驱动文件路径，然后点击“下一步”。



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

3.1 轴操作

当板卡正确安装之后打开 Motion 演示软件如下图 3-1 所示：

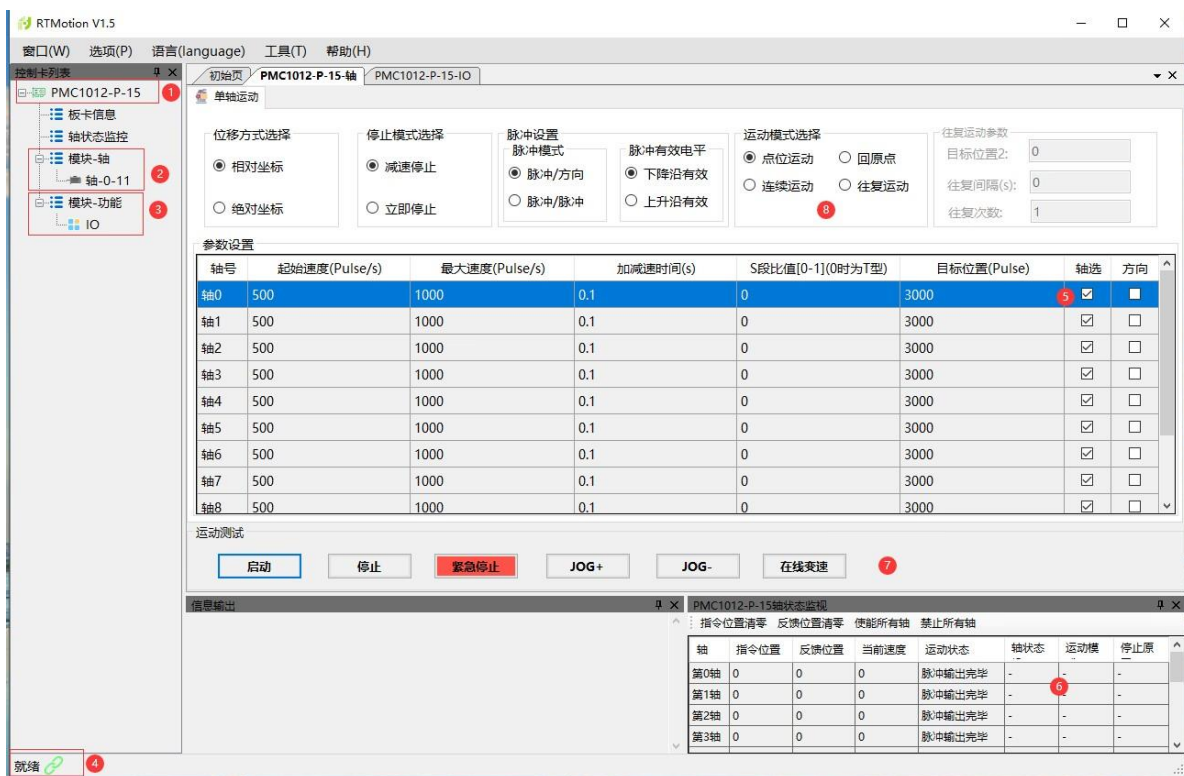


图 3-1 Motion 软件图

- 1) 打开 Motion 软件会自动扫描设备上的板卡并以列表的形式显示在“控制卡列表”中，其中-15 表示在控制卡拨码开关设置的板卡卡号。
- 2) Motion 软件会根据卡信号列出卡的功能模块，其中轴模块能够控制轴的运动停止、参数修改。
- 3) IO 模块中能够监测整个板卡的通用输入信号、通用输出信号、轴专用信号的状态。
- 4) 当 Motion 软件和控制板卡正常连接的情况下显示“就绪”并置成功连接图标；当无效连接时显示“离线”并置断开连接图标。
- 5) 在轴模块的页面打开中提供轴操作选择，只有勾选中的中才会应用用户



的各种操作。

- 6) 打开轴模块时会自动打开轴状态监控子模块，能够监控板卡所有轴的运动状态、当前位置、当前速度等。
- 7) 提供对轴的指令操作按钮。
- 8) 提供对轴运动参数设置选项。

3.2 IO 操作

IO 操作能够控制板卡的所有通用输入、通用输出、专用信号，界面打开如下图所示：

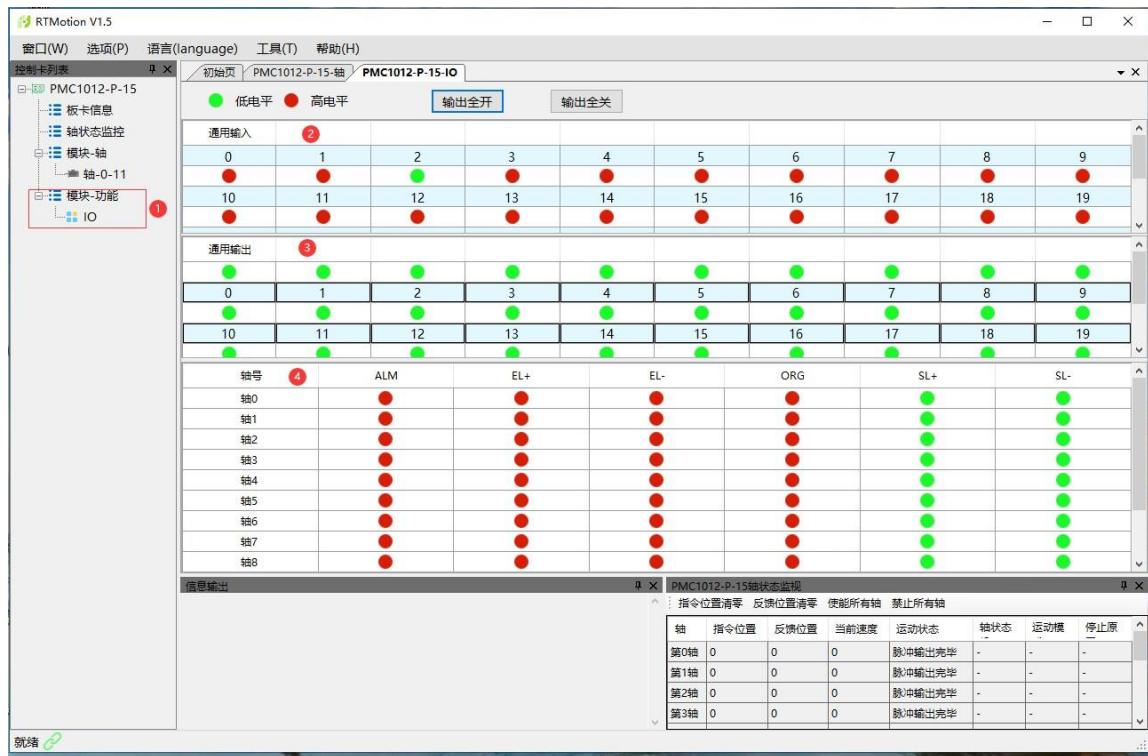


图 3-1 Motion IO 界面软件图

- 1) 功能模块提供板卡支持的所有功能
- 2) IO 模块界面能够实时监测通用输入的状态
- 3) IO 模块界面能够实时监测通用输出的状态，同时可以点击改变通用输出的状态。
- 4) IO 模块界面能够实时监测所有轴的专用信号状态。

第4章 应用程序开发

在我司提供的 PMC1000-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 基于 Windows 平台的应用软件架构

PMC1000-P 系列控制卡的机器控制系统框架如图 4-1 所示：

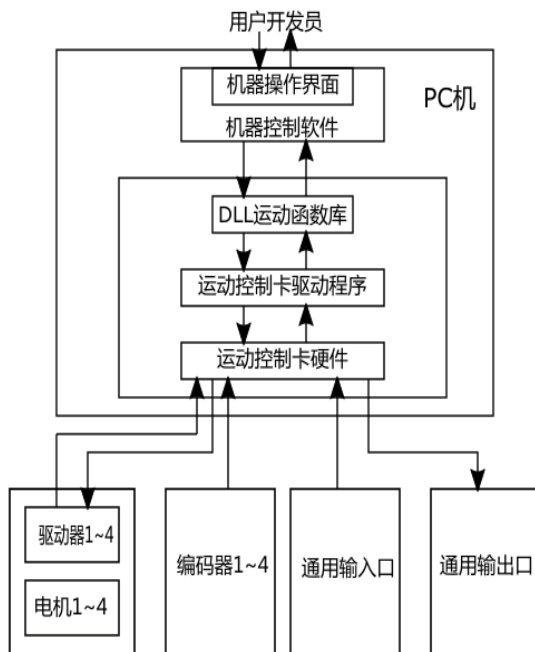


图 4-1 控制卡的机器控制系统框架

从上面的示意图可以看出，控制系统的工作原理可以概述为：

- 1) 操作员的操作信息通过操作接口（包括显示屏和键盘）传递给机器控制软件；
- 2) 机器控制软件将操作信息转化为运动参数并根据这些参数调用 DLL 库中运动函数；
- 3) 运动函数调用运动控制卡驱动程序发出控制指令给控制卡；
- 4) 运动控制卡再根据控制指令发出相应的控制信号(脉冲、方向信号) 给电机驱动器；
- 5) 电机驱动器根据控制信号来驱动电机运动；
- 6) 电机的运动通过机械传动机构转化为机器的动作。

用户在开发应用软件（即机器控制软件）的过程中所需要做的就是针对上面所说的第 1 步和第 2 步进行编程。锐特公司已提供支持 PMC1000-P 系列运动控制卡的硬件驱动程序和 DLL 运动函数库。这些函数提供了所有与运动控制相关的功能，使用极为方便。用户不需要更多了解硬件电路的细节以及运动和插补的计算细节，就能够使用 C、C++、Visual Basic 等程序语言调用这些函数来快速开

发出自己的应用软件。

4.2 Visual C++6.0

- 1) 启动 Visual C++6.0，新建一个 MFC 工程；

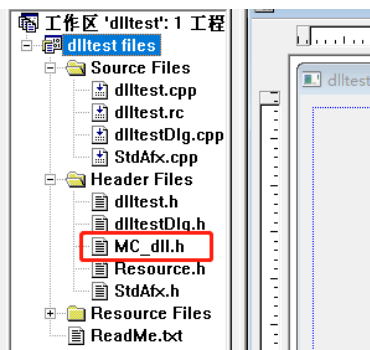


- 2) 根据项目平台不同选择 32 位或者 64 位的动态链接库文件，将"mcdll.dll"

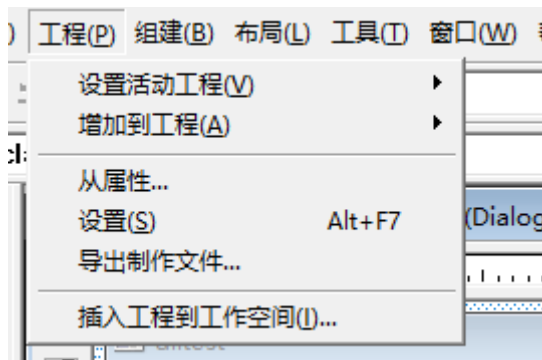
和"mcdll.lib"和 MC_dll.h 三个文件复制到工程文件夹中。

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

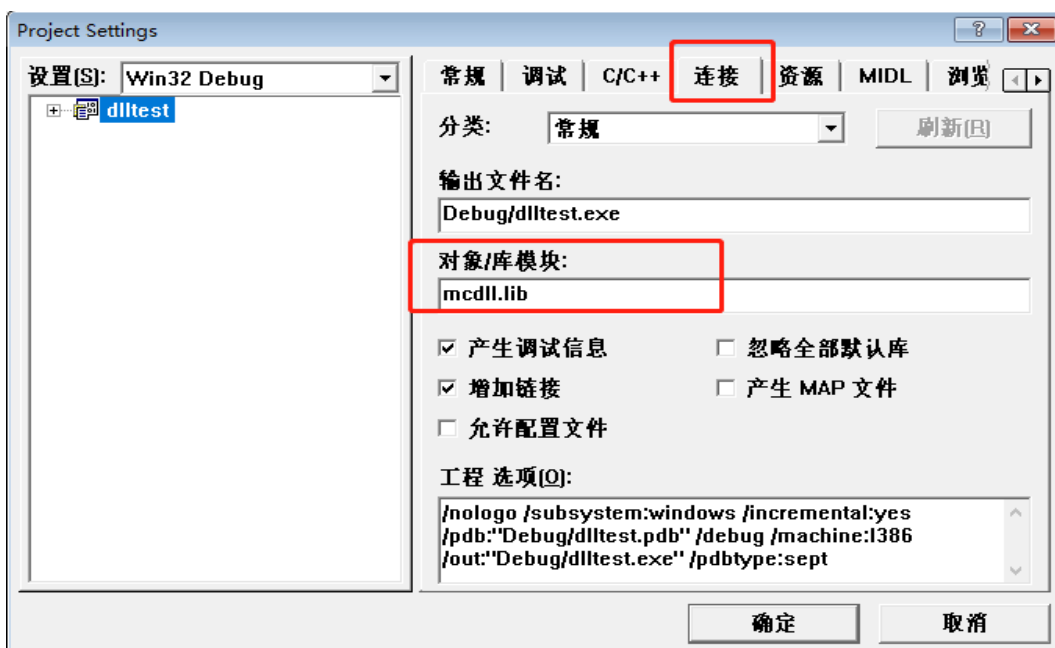
- 3) 在项目中右键项目"添加文件至工程"将 MC_dll.h 头文件添加至项目中。



- 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。

4.3 Visual Studio

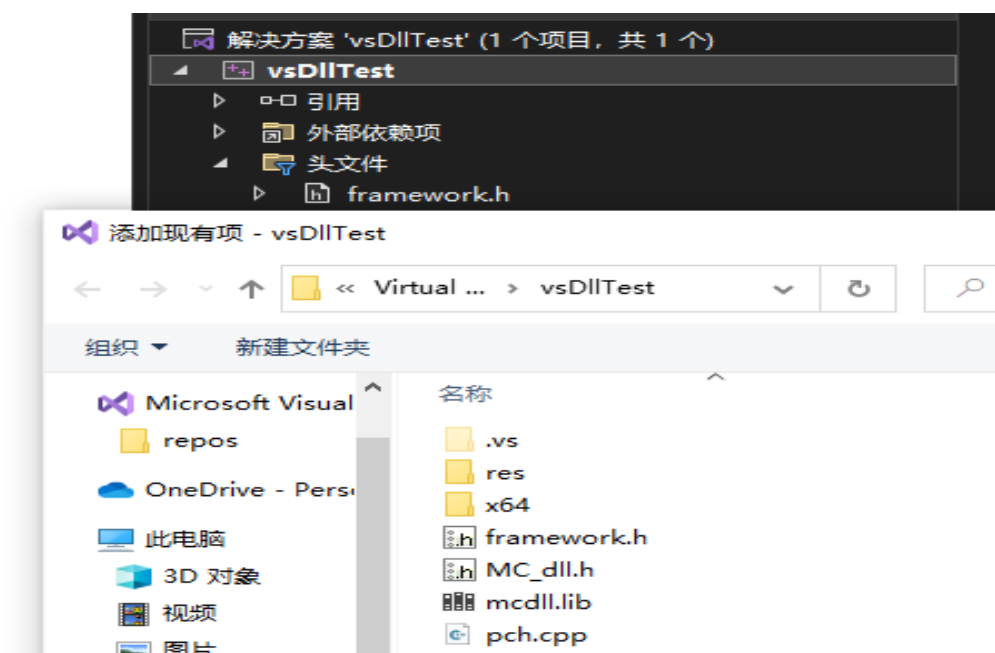
- 1) 启动 Visual Studio，点击新建 MFC 项目或者其他 C++ 项目。
- 2) 根据应用程序的编译位数选择合适的 mcdll.lib 文件，将 mcdll.lib 和



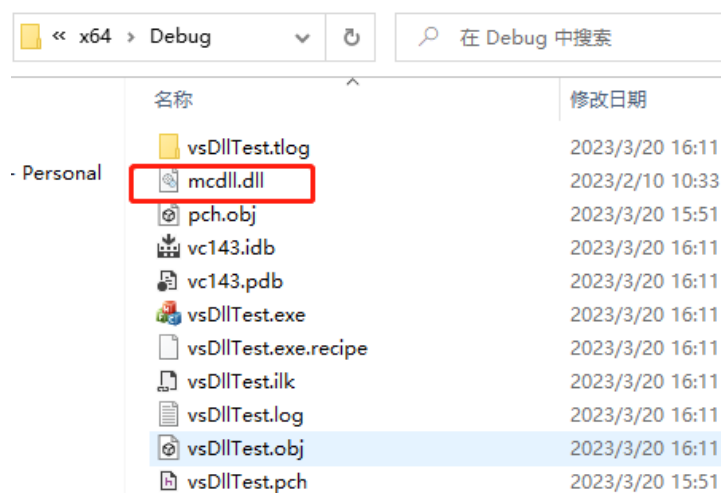
MC_dll.h 文件复制到项目路径下。

.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。



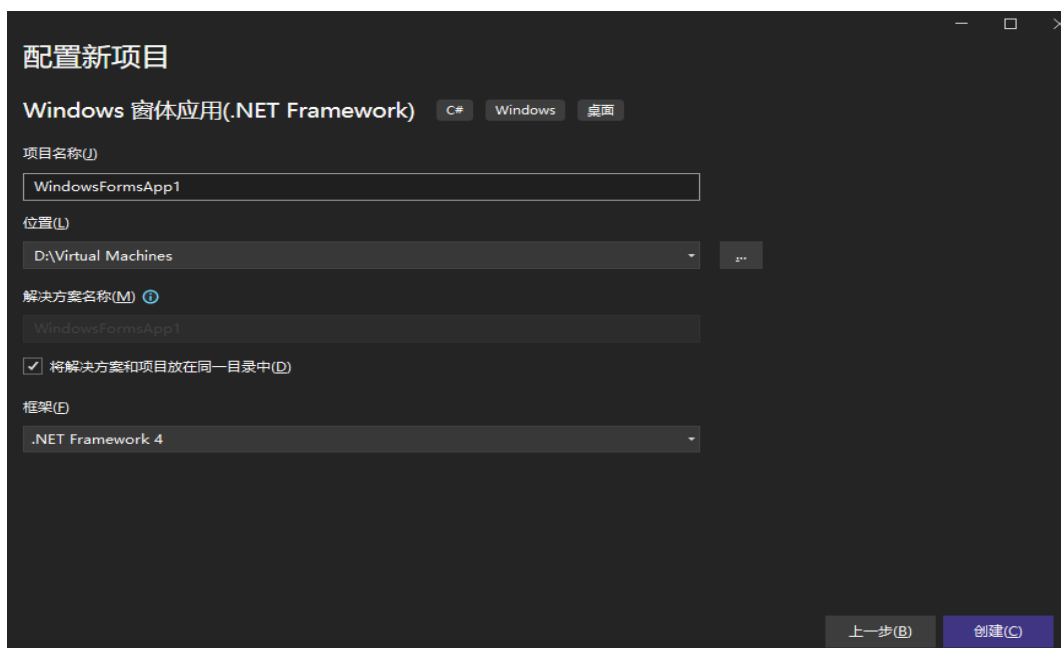


至此，用户可以调用函数库中的任何函数进行程序开发。

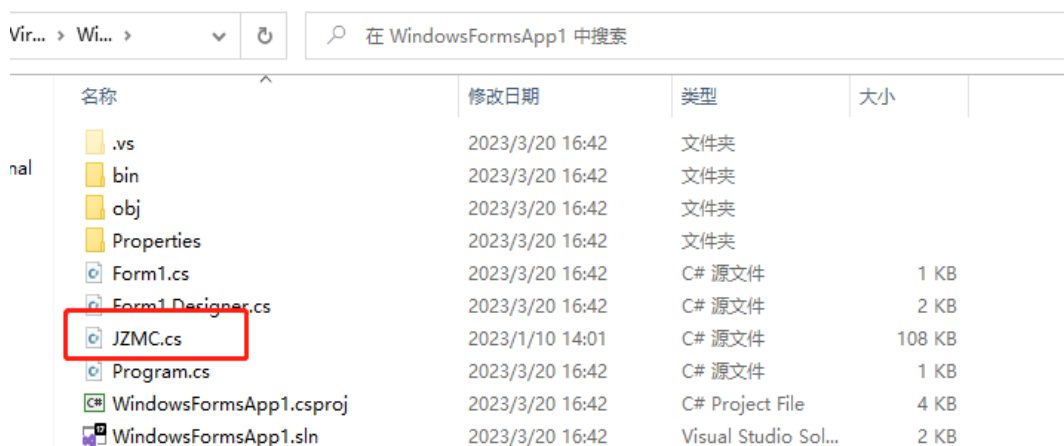
注意：必须将 mcdll.dll 文件放至执行程序.exe 文件路径下，否则执行程序会报缺失 xx.dll 文件的错误。

4.4 Visual C#

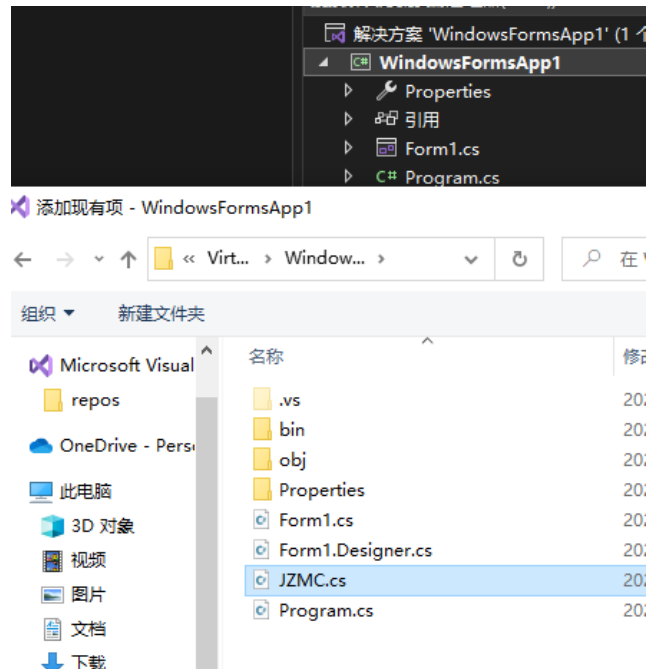
- 1) 启动 Visual Studio,选择建立 C#工程。



- 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 初始化、关闭运动控制卡

在操作 PMC1000-P 系列运动控制卡之前，必须调用控制卡初始化函数为运动控制卡分配资源。同样，当结束对运动控制卡的操作时，必须调用控制卡关闭函数释放运动控制卡所占用的系统资源，使得所占资源可被其它设备使用。具体相关函数和功能如表 5-1 所示：

表 5-1 初始、关闭控制卡函数说明

名称	功能	参考
MC_Open	初始化卡并分配系统资源	8.2.2.1
MC_Close	关闭卡并释放系统资源	8.2.2.1
注意：程序结束时，必须调用MC_Close ()函数释放系统资源。		

例程：初始化和关闭控制卡(以 C 语言为例说明，下同)

```
Int CardCount = 0;
Int CardIndex[15]={0};
MC_Open (0,2,"",& CardCount, CardIndex);
if(CardCount == 0)
{
    printf( "\n 没有发现运动控制卡" );
    getch();
    return;
}
//.....
For(int cardIndex = 0;cardIndex<CardCount;cardIndex++)
{
    MC_Close(CardIndex[cardIndex]);
```



}

5.2 设置脉冲输出模式

PMC1000-P 系列运动控制卡采用脉冲指令控制步进/伺服电机。针对不同型号的驱动器，在使用控制卡时必须对脉冲信号输出方式进行正确设定，系统才能正常工作。相关函数和功能如表 5-2 所示：

表 5-2 脉冲设置函数说明

名称	功能	参考
MC_SetPulseMode	设置指定轴的脉冲输出模式	
MC_GetPulseMode	获取指定轴的脉冲输出模式	
注意：在调用运动控制函数之前应先调用该函数来设置指令脉冲模式。		

指令脉冲包括两项基本信息：电机运转距离即脉冲数和电机转动方向。有两种基本指令模式：两种基本模式如表 5-3 所示：

表 5-3 脉冲指令模式表

模式	PULn-脚输出	DIRn-脚输出
方向脉冲（pulse/dir）	脉冲信号	方向信号（电平）
双脉冲（CW/CCW）	正向（CW）脉冲	反向（CCW）脉冲
注意：具体设置请参考MC_SetPulseMode函数具体说明。		

5.2.1 脉冲/方向模式

在此模式下，PULn-端子输出指令脉冲串，脉冲数对应电机运行的相应“距离”，而脉冲频率对应电机运行“速度”。DIRn-端子输出方向信号，该信号的输出电平对应电机的转动方向。此种模式在电机驱动器中应用最多。

脉冲信号可以设置为上升沿有效，即脉冲信号常态为低电平，当变为高电平时电机走一步；也可设置为下降沿有效，即脉冲信号常态为高电平，当变为低电平时电机走一步。所以实际上此种模式下有两种指令类型，如图 5-1 所示。

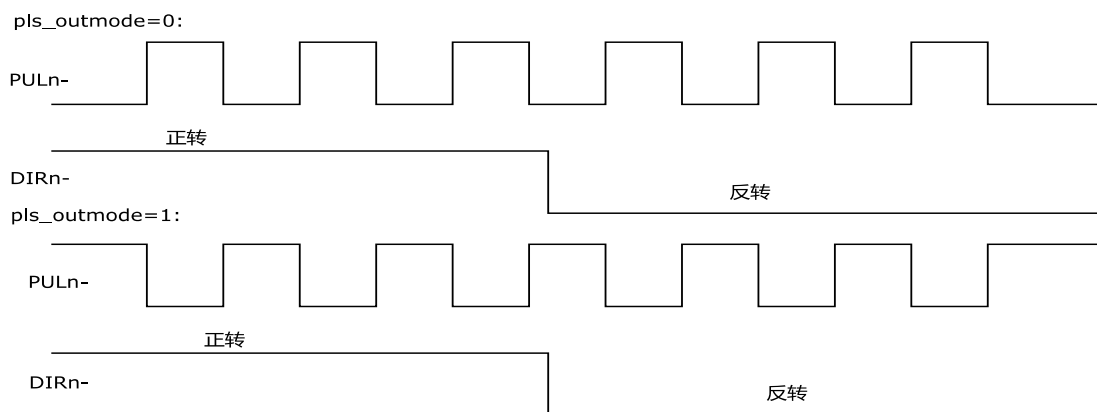


图 5-1 脉冲/方向信号图

5.2.2 双脉冲模式

在此模式下，PULn-和 DIRn-端子分别表示正向（CW）和反向（CCW）脉冲输出。从 PULn-端子输出的脉冲使电机朝正方向转动，而从 DIRn-端子输出的脉冲使电机朝负方向转动。脉冲信号有上升沿或下降沿有效的选择，所以该种模式下也有两种指令类型，如图 5-2 所示：

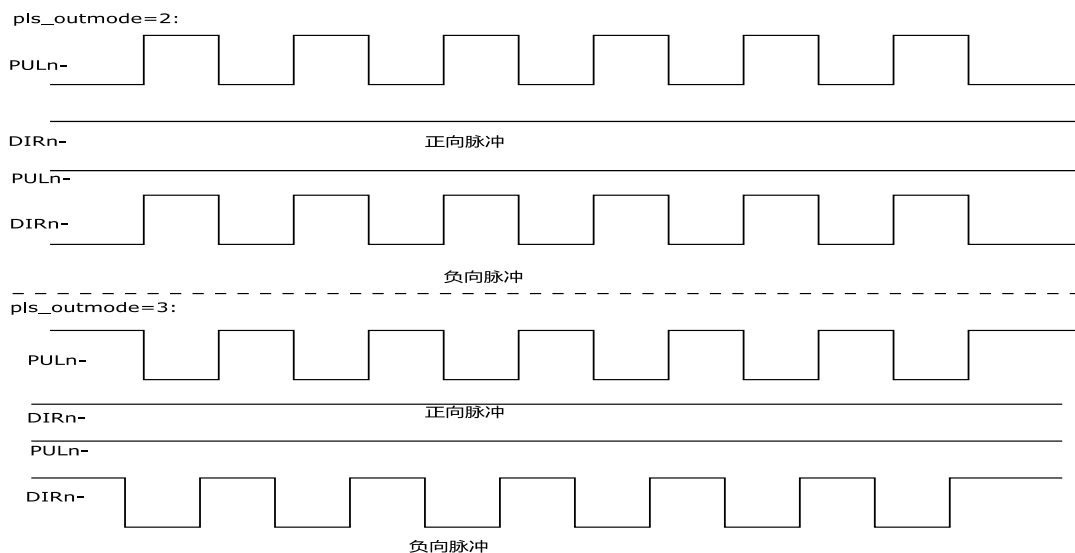


图 5-2 双脉冲信号图

例程：设置和获取单轴脉冲输出模式

```
Int pulseMode = 0;
```

```
/*
```

0 = pulse/dir 上升沿有效

1 = pulse/dir 下降沿有效

2 = CW/CCW 上升沿有效

3 = CW/CCW 下降沿有效

*/

`MC_SetPulseMode (0,0, pulseMode);` //表示设置卡 0 的轴 0 脉冲输出模式为脉冲方向模式，PULn-信号上升沿有效。

`MC_GetPulseMode(0,0, &pulseMode);`//获取卡 0 的轴 0 脉冲输出模式，获取至 pulseMode 变量。

5.3 单轴位置和速度运动

5.3.1 梯形速度曲线运动

位置控制采用这种速度控制模式较为常见。电机在运动一段指定距离时，加速减速过程中加速度的大小恒定不变，其运动速度按梯形曲线变化。运动速度之所以要按梯形曲线变化，是因为：电机转子和被拖动的物体具有惯性，不可能在瞬间达到指定速度，因此应该有一定的加速过程。减速时亦是类似，否则电机会因为瞬间力矩不足而出现丢步、过冲（步进电机系统）或振荡（伺服电机系统）现象。

实现梯形速度曲线控制的点位运动函数如表 5-4 所示：

表 5-4 梯形点位控制相关函数说明

名称	功能	参考
MC_MoveAbsolute	以绝对坐标启动一段点位运动，当S_Ratio参数为0时是梯形速度曲线	
MC_MoveRelative	以相对坐标启动一段点位运动，当S_Ratio参数为0时是梯形速度曲线	
注意：在PMC1000-P系列中加减速是对称的，所以加速时间等于减速时间		

5.3.2 S 型速度曲线运动

若加速度是线性变化，则加速和减速阶段的速度曲线将变为“S”形。这种速度曲线运动模式，运动过程更平稳，且有助于缩短加速过程、降低运动装置的振动和噪声以及延长机械传动部分的使用寿命。

设置 S 形速度曲线及其点位运动的函数如表 5-5 所示：

表 5-5 S 形速度控制相关函数说明

名称	功能	参考
MC_MoveAbsolute	以绝对坐标启动一段点位运动，当S_Ratio参数不为0时是S型速度曲线	
MC_MoveRelative	以相对坐标启动一段点位运动，当S_Ratio参数不为0时是S型速度曲线	
注意：在PMC1000-P系列中加减速是对称的，所以加速时间等于减速时间		

例程：S 形速度曲线作点位运动

Int connectNo = 0; //卡号，从开卡函数获得

Int axisNo = 0; //轴号

Double targetPos = 5000; //目标位置，5000Pulse

Double startVel = 500; //初始速度，500Pulse/s

Double stopVel = 0; //保留参数

Double maxVel = 1000; //最大运行速度

Double accTime = 0.1; //加减速时间,范围 0~1s

Double decTime = 0; //保留参数

Double sRatio = 0.1; //不为 0 时是 S 型速度曲线

Double maxAcc = 0; //保留参数

MC_MoveRelative (connectNo, axisNo, targetPos, startVel, stopVel, maxVel, accTime, decTime, sRatio, maxAcc); //设置卡 1 的轴 0 相对坐标下运动距离为 5000 个脉冲、起始速度为 500 脉冲/秒、运行速度为 1000 脉冲/秒、加减速时间为 0.1 秒、以 S 形速度曲线开始执行运动



5.3.3 速度运动

PMC1000-P 系列控制卡可以控制电机以梯形或 S 形速度曲线在指定加速时间内从初速度加速至运行速度，然后以该速度连续运行，直至调用停止指令或该轴遇到限位信号才会停止。速度运动的函数如表 5-6 所示：

表 5-6 速度运动相关函数说明

名称	功能	参考
MC_MoveVelocity	启动单轴速度运动，当S_Ratio参数等于0时是梯型速度曲线，不等于0时是S型速度曲线。	

例程：

例程：梯形速度曲线启动速度运动

Int connectNo = 0; //卡号，从开卡函数获得

Int axisNo = 0; //轴号

Double startVel = 500; //初始速度，500Pulse/s

Double stopVel = 0; //保留参数

Double maxVel = 1000; //最大运行速度

Double accTime = 0.1; //加减速时间,范围 0~1s

Double decTime = 0; //保留参数

Double sRatio = 0.1; //不为 0 时是 S 型速度曲线

Double maxAcc = 0; //保留参数

MC_MoveVelocity (connectNo, axisNo, startVel, stopVel, maxVel, accTime, decTime, sRatio, maxAcc); //设置卡 0 轴 0 起始速度为 500 脉冲/秒、运行速度为 1000 脉冲/秒、加减速时间为 0.1 秒、以 S 形速度曲线开始速度运动。

5.3.4 直线插补运动

PMC1000-P 系列控制卡可以指定同一卡上任意 2 轴、3 轴进行直线插补，插

补运动的计算是由专用函数完成的，用户只需调用相应的运动函数并设置运动速度、加速度、终点位置等参数，不需要介入插补过程的计算工作。

插补运动与多轴联动的区别：插补运动不但能保证起点、终点位置准确，而且各轴的脉冲是按照直线斜率成比例发出的。直线插补相关函数如表 5-7 所示：

表 5-7 直线插补运动相关函数说明

名称	功能	参考
MC_MoveLinearEx	启动多轴的直线插补运动	

二轴直线插补：

如图 4-7 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 $\Delta X : \Delta Y$ ，与合成的矢量速度 ΔP 的关系为：

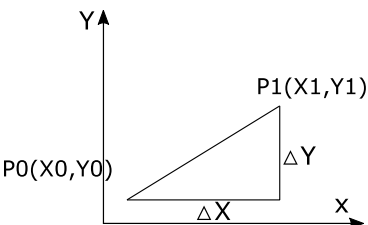
$$\frac{\Delta P}{\Delta t} = \sqrt{\left(\frac{\Delta X}{\Delta t}\right)^2 + \left(\frac{\Delta Y}{\Delta t}\right)^2}$$


图 5-6 两轴直线插补示意图

调用二轴直线插补函数时，需要设定插补矢量速度，包括其初始速度 StrVel 和运行速度 MaxVel 等参数。

例程：二轴直线插补

```
Short AxisArray[2];
```

```
Short DistArray[2];
```

```
AxisArray[0]=0; // 轴 0、轴 1 直线插补
```

```
AxisArray[1]=1;
```

```
DistArray[0]=1000; // ΔX=1000 脉冲
```

```
DistArray[1]=2000; // ΔY=2000 脉冲
```

`MC_MoveLinearEx (0,2,AxisArray, DistArray, 400,1000, 0.1,0);` //起始速度为 400 pps，运行速度为 1000 pps，加速时间为 0.1s，相对坐标模式。

三轴直线插补：

如图 5-7 示，XYZ 3 轴直线插补从 P0 点运动至 P1 点。插补过程中 3 轴的速度比为 $\Delta X:\Delta Y:\Delta Z$ ，与合成的矢量速度 ΔP 的关系为：

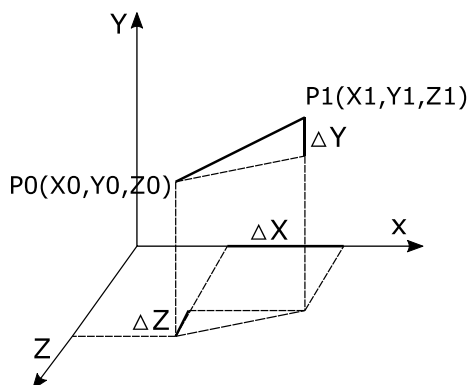


图 5-7 三轴直线插补

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

调用 3 轴直线插补时，需要设定插补矢量速度，包括初速度 StrVel 和运行速度 MaxVel 等参数。

3 轴直线插补例程：

```
Short AxisArray[3];
```

```
Short DistArray[3];
```

```
AxisArray[0]=0; //轴 0、轴 1、轴 2 直线差补
```

```
AxisArray[1]=1;
```

```
AxisArray[2]=2
```

```
DistArray[0]=1000; //ΔX=1000 脉冲
```

```
DistArray[1]=2000; //ΔY=2000 脉冲
```

```
DistArray[2]=3000; //ΔZ=3000 脉冲
```

```
MC_MoveLinearEx (0,3,AxisArray, DistArray, 400,1000, 0.1,0); //起始速度为
```

400pps，运行速度为 1000pps，加速时间为 0.1s,相对坐标模式。

5.3.5 回原地运动

在进行精确的位置控制之前，需要设定运动坐标系的原点。如图 5-8 所示为回原点运动的过程。

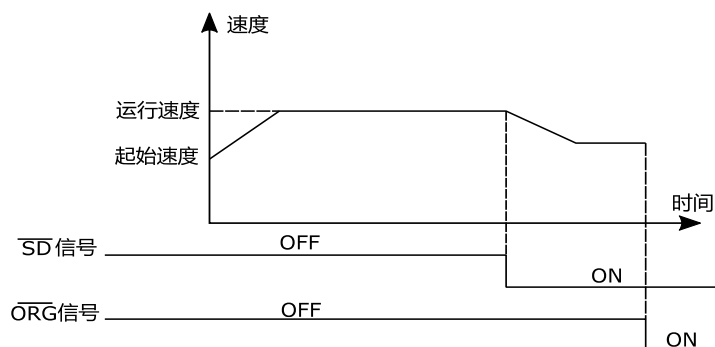


图 5-8 回原点过程示意图

从图 5-8 可以看出，启动回原点运动后，设备向原点方向运动；当碰到减速开关后，电机开始减速。当速度减至初速度后，继续以该速度运行，直到原点信号有效便立即停止，该位置即为原点位置。这种方式在高速回原点的运动中特别有用。具体回原点运动相关函数如表 5-8 所示：

表 5-8 回原点运动相关函数说明

名称	功能	参考
MC_HomeEx	启动指定轴进行回原点运动	

例程：回原点运动

`MC_HomeEx (0,0,500,5000,0.1);` // 设置卡 0 的 0 号轴以起始速度为 500pulse/s，运行速度为 5000pulse/s，加速时间为 0.1s 正向回原点运动

`UInt16 checkDone = 0;`

`MC_GetAxisCheckDone(0,0,& checkDone);`//获取卡 0 的 0 号轴运动状态

`while (checkDone ==0) //等待回原点动作完成`

`{`

`MC_GetAxisCheckDone(0,0,& checkDone);`

```
}
```

```
MC_SetAxisCmdPos (0,0,0); //将当前第 0 轴位置清零，即设为第 0 轴原点位置
```

5.3.6 脉冲指令计数

PMC1000-P 系列控制卡每轴都有一个 24 位的指令脉冲计数器，记录了当前对应轴的绝对指令脉冲位置值。可以通过调用 MC_GetAxisCmdPos 函数来读取该计数器的值，也可以通过调用 MC_SetAxisCmdPos 函数来设置该计数器的值。具体相关函数如表 5-9 所示：

表 5-9 指令脉冲计数相关函数说明

名称	功能	参考
MC_GetAxisCmdPos	读取指令位置计数器值	
MC_SetAxisCmdPos	设置指令位置计数器值	
注：在点位运动中，每次可以输出的最大脉冲个数为16777215。		

例程：指令位置操作

```
MC_SetAxisCmdPos (0,0,100); //设置卡 0 的轴 0 的脉冲位置为 100

Double curPos = 0;

MC_GetAxisCmdPos (0,0,& curPos); //读取卡 0 的轴 0 的当前位置值至变量 curPos
```

5.4 通用 I/O 控制

PMC1000-P 系列运动控制卡硬件上为用户提供了通用数字输入输出接口。用户可以使用这些数字 I/O 口用于检测开关信号、传感器信号等输入信号，或者输出继电器、指示灯等输出设备的控制信号。相关的函数如表 5-10 所示：

表 5-10 通用 IO 相关函数说明

名称	功能	参考
MC_GetInIoBit	按位读取控制卡的某一位输入口的电平状态	
MC_GetInIoPort	一起读取32位输入口的电平状态	
MC_GetInIoPtr	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_SetOutIoBit	按位设置通用输出口的电平状态	
MC_SetOutIoPort	一次设置32位通用输出口的电平状态	
MC_SetOutIoPtr	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_GetOutIoBit	按位获取通用输出口的电平状态	
MC_GetOutIoPort	一次获取32位通用输出口的电平状态	
MC_GetOutIoPtr	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_SetOutIoPort 或 MC_SetOutIoPort 对运动控制卡的输出口进行置位，使 MC_GetOutIoPort、MC_GetOutIoPtr、MC_GetInIoPort、MC_GetInIoPtr 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。

例程：通用输入输出

```
//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_GetOutIoBit(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_SetOutIoBit(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_SetOutloPtr(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_SetOutloPtr(connect_no,startIndex,0 ,(ulong)0x1);
    }

```




```
}
```

5.5 轴专用 IO 控制

PMC1000-P 系列运动控制卡为每个轴提均供了 6 个专用信号输入接口，分别为 1 个原点信号（ORG）输入接口、1 个报警信号(ORG)输入接口、2 个减速信号（SD+/SD-）输入接口以及 2 个限位信号（EL+/EL-）输入接口。各信号相关功能的实现均由控制卡硬件来完成，用户只需按照正确的接线方式接线，并调用相应功能的函数执行运动指令即可。具体相关函数如表 5-11 所示：

表 5-11 轴专用 IO 控制相关函数说明

名称	功能	参考
MC_GetAxisInIoState	获取轴专用信号状态	

例程：获取轴专用信号状态

```
UInt32 axisioState = 0;
```

```
MC_GetAxisInIoState(0,0,& axisioState);//获取卡 0 的 0 号轴专用信号状态,axisioState 的不同 bit 代表不同信号的状态。
```

```
/*
```

```
Bit0:EL-
```

```
Bit1:EL+
```

```
Bit2:ORG
```

```
Bit3:STP
```

```
Bit4:STA
```

```
Bit5:SD-
```

```
Bit6:SD+
```

```
Bit7:保留
```

```
Bit8:保留
```

```
Bit9:保留
```

```
Bit10:保留
```



Bit11:SL-

Bit12:SL+

Bit13:ALM

*/

5.6 软件限位

PMC1000-P 系列运动控制卡为用户提供了软件限位的功能。用户需要在软件上设置轴的软件限位使能还有轴的正向软件限位值和负向软件限位值，当设置完成后该轴的运动始终会被限制在负向软限位值到正向软限位值之间，当运动至软件限位值处后该轴运动会自动停止，且软限位标志触发。相关的函数如表 5-12 所示：

表 5-12 软件限位相关函数说明

名称	功能	参考
MC_SetAxisSoftELPrm	设置软件限位参数	
MC_GetAxisSoftELPrm	获取该轴软件限位设置	

例程：软件限位

```
MC_SetAxisSoftELPrm (0,0,1,0,50000,-50000,0); //设置卡 0 的轴 0 的软件限位使能，正向限位值为 50000，负向限位值为-50000
```

```
UInt32 axisState = 0;
```

```
MC_GetAxisInIoState (0,0,& axisState); //获取轴 0 的专用信号状态
```

```
If(GetBitStatus(axisState , 11) == 1) //这里获取 axisState 的 bit11 值，bit11 代表 SL-的状态
```

```
{
```

```
    //软件负限位处于触发状态
```

```
}  
  
Else if(GetBitStatus(axisState , 12) == 1)//这里获取 axisState 的 bit12 值，bit12  
代表 SL+的状态  
  
{  
    //软件正限位处于触发状态  
}
```

5.7 专用信号复用

PMC1000-P 系列运动控制卡提供了专用信号作普通输入使用的复用功能，目前支持将轴的硬件限位 EL+、EL-和轴的减速信号 SD+、SD-通过复用当作普通输入来使用，复用之后原有的限位和减速功能将失效。相关函数如表 4-12 所示：

表 4-12 专用信号复用相关函数说明

名称	功能	参考
MC_SetAxisInIoPrm	设置轴专用信号配置	
MC_GetAxisInIoPrm	获取轴专用信号配置	

例程：轴专用信号映射

```
MC_SetAxisInIoPrm (0, 0, 0,0,0,0); //设置卡 0 的轴 0 的硬件限位信号作为普通  
输入使用
```

```
UInt32 axisState = 0;
```

```
MC_GetAxisInIoState (0,0,& axisState);//获取轴 0 的专用信号状态
```

```
If(GetBitStatus(axisState , 0) == 1) //这里获取 axisState 的 bit0 值，bit0 代表  
EL-的状态,这里复用过后它只作普通输入用处
```

```
{  
    //负限位输入口处于触发状态  
}
```

```
Else if(GetBitStatus(axisState , 1) == 1)//这里获取 axisState 的 bit1 值，bit1 代
```



表 EL+的状态,这里复用过后它只作普通输入用处

```
{  
    //正限位输入口处于触发状态  
}
```

第6章 函数库详解

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
控制器信息	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_MoveAbsolute	开始一段绝对运动
	MC_MoveRelative	开始一段相对运动
	MC_MoveVelocity	单轴速度运动指令
	MC_MoveUpdateVel	在线变速指令
	MC_HomeEx	回零运动指令
	MC_StopAxis	轴停止指令
	MC_MoveLinearEx	多轴直线插补指令



轴状态和轴参数	MC_SetPulseMode	脉冲输出模式配置
	MC_GetPulseMode	获取脉冲输出模式配置
	MC_GetAxisCheckDone	判断轴是否在运动中
	MC_GetAxisCmdPos	获取轴的指令位置
	MC_GetAxisCmdSpeed	获取轴的指令速度
	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_GetOutIoBit	按位获取通用输出口的状态
	MC_GetOutIoPort	按端口获取通用输出口的状态
	MC_GetOutIoPtr	按起始索引和需要读取



		的数量读取通用输出口的状态
轴专用信号	MC_GetAxisInIoState	获取轴所有专用信号状态
	MC_SetAxisInIoPrm	设置轴专用信号配置
	MC_GetAxisInIoPrm	获取轴专用信号配置

6.2 函数说明

6.2.1 连接控制器操作

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

6.2.2 控制器信息相关函数

<pre>DWORD MC_GetTotalAxesAndIONum(WORD connect_no,WORD* total_axes, WORD* liner_num ,WORD* total_in_num,WORD* total_out_num, WORD* total_pwm_num);</pre>	
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

<pre>DWORD MC_GetTotalAdAndDaNum(WORD connect_no,WORD* total_ad_num, WORD* total_da_num);</pre>	
---	--



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_handle_wheel_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

<pre>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);</pre>	
单轴按绝对位置坐标运行至目标位置	
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	保留参数
Vel	最大运行速度(Pulse/s)
Acc	加减速时间(S)
Dec	保留参数
S_Ratio	S 段比例(为 0 时为梯形加减速曲线；0~1 时则为 S 形加减速曲线)



maxAcc	保留参数
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);	
单轴按相对位置坐标运行至目标位置	
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	保留参数
Vel	最大运行速度(Pulse/s)
Acc	加减速时间(S)
Dec	保留参数
S_Ratio	S 段比例(为 0 时为梯形加减速曲线； 0~1 时则为 S 形加减速曲线)
maxAcc	保留参数
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	保留参数
Vel	最大运行速度(Pulse/s)
Acc	加减速时间(S)
Dec	减速度
S_Ratio	S 段比例(为 0 时为梯形加减速曲线; 0~1 时则为 S 形加减速曲线)
maxAcc	保留参数
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_HomeEx(WORD connect_no,WORD axis,int32 startVel,int32 maxVel,double tAcc);	
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)
startVel	初始速度(Pulse/s)
maxVel	最大运行速度(Pulse/s)
tAcc	加减速时间(S)
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	保留参数
返回值	错误码

DWORD MC_MoveLinearEx(WORD ConnectNo ,WORD AxisNum,WORD* AxisList,double* DistArr,double startVel,double maxVel,double accTime,WORD pos_Mode);	
直线插补命令	
参数	说明
connect_no	连接号，成功开卡获得
AxisNum	插补轴数
AxisList	插补轴数组
DistArr	目标位置数组
startVel	初始速度(Pulse/s)
maxVel	最大运行速度(Pulse/s)
accTime	加减速时间(S)
pos_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

6.2.4 轴状态和参数相关函数

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 = pulse/dir 上升沿有效 1 = pulse/dir 下降沿有效 2 = CW/CCW 上升沿有效 3 = CW/CCW 下降沿有效
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_GetAxisCheckDone(WORD connect_no, WORD axis, WORD* Che	
---	--



ckDone);	
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 = 脉冲输出完毕 2 = 指令停止 3 = 碰到 EL 停止 4 = 碰到 ORG 停止 5 = 碰到软件负限位停止 6 = 碰到软件正限位停止
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(Pulse/s)
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	保留参数
max_pos	软件正限位位置
min_pos	软件负限位位置
stop_mode	保留参数
返回值	错误码

<pre>DWORD MC_GetAxisSoftELPrm(WORD connect_no, WORD axis, WORD* enable, WORD* pos_src, double* max_pos, double* min_pos, WORD* stop_mode);</pre>	
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	保留参数



max_pos	返回软件正限位位置
min_pos	返回软件负限位位置
stop_mode	保留参数
返回值	错误码

6.2.5 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 I	
--	--



oState);	
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	获取到的 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	获取到的 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 i o_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

6.2.6 轴专用信号相关函数

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 = EL- Bit1 = EL+ Bit2 = ORG Bit3 = STP Bit4 = STA Bit5 = SD- Bit6 = SD+ Bit7 = 保留 Bit8 = 保留 Bit9 = 保留 Bit10 = 保留 Bit11 = SL- Bit12 = SL+ Bit13 = ALM
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 = \pm EL 1 = \pm SL 2 = \pm SD 3 = ALM
enable	信号是否有效 0 = 禁止(当专用信号禁止时只当普通输入用) 1 = 使能(专用信号使能时具有专用信号功能)
io_logical	信号有效电平(该参数只对 ALM 设置有效) 0 = 低电平有效 1 = 高电平有效
stop_mode	保留参数
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 = \pm EL



	1 = \pm SL 2 = \pm SD 3 = ALM
enable	返回信号是否有效 0 = 禁止(当专用信号禁止时只当普通输入用) 1 = 使能(专用信号使能时具有专用信号功能)
io_logical	返回信号有效电平(该参数只对 ALM 有效) 0 = 低电平有效 1 = 高电平有效
stop_mode	保留参数
返回值	错误码

第7章 附录

7.1 函数返回错误码

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
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	spl 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
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
日志打印错误码	1701	配置 Log 打印错误