Zmotion®
正运动技术

做更好用的运动控制，智造美好生活
Better Motion Control, Smarter Life

Zmotion PC 函数库编程手册

V2.1.4

C# C++ LabMIEW VB MATLAB Qt
DELPHI Linux Microsoft .NET Framework python ROS













机器视觉运动控制一体机



运动控制器



运动控制卡



IO扩展模块



人机界面

版权说明

本手册介绍了正运动技术 PC 函数指令的使用及相关开发功能的简介。

为了更好的服务于广大受众，深圳市正运动技术有限公司，对所发布的信息（含文字、数据、图片等）作出以下声明

本手册版权归深圳市正运动技术有限公司所有，严禁任何媒体、网站、个人或组织以任何形式或出于任何目的在未经本公司书面授权的情况下抄袭、转载、摘编、修改本手册内容，或链接、转贴或以其他方式复制用于商业目的或发行，或稍作修改后作为他用，前述行为均将构成对本公司手册版权之侵犯，本司将依法追究其法律责任。

本手册中的信息资料仅供参考。由于改进设计和功能等原因，正运动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！



调试机器要注意安全！请务必在机器中设计有效的安全保护装置，并在软件中加入出错处理程序，否则所造成的损失，正运动公司没有义务或责任对此负责。

为了保证产品安全、正常、有效的使用，请您务必在安装、使用产品前仔细阅读本产品手册。

本手册所有例程源码均可通过以下百度网盘获取！！

网盘链接：https://pan.baidu.com/s/19z27imhdqwTDxycFwuenrA?pwd=qcgk

提取码：qcgk

修订记录

更新日期	版本号	版本（更改）说明	更新人
2022/10/15	V2.1.0	1. 新增 MOVEOP 例程（4.1.3） 2. 新增 HW2 相关内容及例程（4.1.3）	YJJ
2022/12/10	V2.1.1	1. 新增 MotionRt 的章节（4.1.2-2）及相关例程（4.1.3-5） 2. 新增实用封装例子（2.2.2） 3. 新增在线命令例子（12.1.3）	YJJ
2024/1/30	V2.1.2	1. 修改部分指令描述错误	ZXL
2025/8/26	V2.1.3	1. 修改部分指令描述错误	CWW
2025/12/9	V2.1.4	1. 手册图片优化 2. 更新 RTSys 相关内容	CWW

目录

Zmotion PC 编程手册·基本功能篇 1
第一章 产品描述说明 1
1.1 正运动控制器特点 1
1.2 产品架构 2
第二章 函数库的使用与封装 6
2.1 函数库调用 6
2.1.1 Windows 系统 6
2.1.2 LINUX 系统 21
2.1.3 Wince 系统 27
2.2 辅助库封装 30
2.2.1 辅助库封装方式 30
2.2.2 实用封装例子 33
第三章 指令一览表 46
第四章 板卡连接初始化 58
4.1 控制器连接 58
4.1.1 指令列表 58
4.1.2 重点说明 59
4.1.3 例程 80
4.2 控制器信息获取 91
4.2.1 指令列表 91
4.2.2 重点说明 91
4.2.3 例程 91
第五章 运动控制参数设置 94
5.1 运动基本参数 94
5.1.1 指令列表 94
5.1.2 重点说明 95
5.1.3 例程 103
5.2 轴其他参数 110
5.2.1 指令列表 110
5.2.2 重点说明 110
5.2.3 例程 115
5.3 轴参数指令表 129
5.3.1 指令列表 129
5.3.2 重点说明 129
5.3.3 例程 131
第六章 基本运动控制 135
6.1 运动缓冲机制 135
6.1.1 指令列表 135
6.1.2 重点说明 136
6.1.3 例程 137

6.2 单轴运动.....142
6.2.1 指令列表.....142
6.2.2 重点说明.....142
6.2.3 例程.....143
6.3 回零运动.....151
6.3.1 指令列表.....151
6.3.2 重点说明.....152
6.3.3 例程.....154
6.4 电子齿轮.....157
6.4.1 指令列表.....157
6.4.2 重点说明.....157
6.4.3 例程.....158
6.5 插补运动.....171
6.5.1 指令列表.....171
6.5.2 重点说明.....173
6.5.3 例程.....176
6.6 轴暂停/恢复/停止相关.....209
6.6.1 指令列表.....209
6.6.2 重点说明.....209
6.6.3 例程.....209
第七章 进阶运动控制.....212
7.1 坐标循环.....212
7.1.1 指令列表.....212
7.1.2 重点说明.....212
7.1.3 例程.....212
7.2 螺距补偿.....215
7.2.1 指令列表.....215
7.2.2 重点说明.....216
7.2.3 例程.....216
7.3 反向间隙补偿.....220
7.3.1 指令列表.....220
7.3.2 重点说明.....220
7.3.3 例程.....221
第八章 硬件接口访问与配置.....224
8.1 数字输入输出.....224
8.1.1 指令列表.....224
8.1.2 重点说明.....224
8.1.3 例程.....225
8.2 模拟量输入输出.....231
8.2.1 指令列表.....231
8.2.2 重点说明.....231
8.2.3 例程.....231

8.3 PWM 控制 233
8.3.1 指令列表 233
8.3.2 重点说明 233
8.3.3 例程 234
第九章 安全机制 237
9.1 轴软件限位函数 237
9.1.1 指令列表 237
9.1.2 重点说明 237
9.1.3 例程 238
9.2 轴硬件限位函数 241
9.2.1 指令列表 241
9.2.2 重点说明 241
9.2.3 例程 242
9.3 轴报警函数 245
9.3.1 指令列表 245
9.3.2 重点说明 245
9.3.3 例程 246
第十章 板卡数据交互 248
10.1 VR 寄存器 248
10.1.1 指令列表 248
10.1.2 重点说明 248
10.1.3 例程 248
10.2 Table 寄存器 250
10.2.1 指令列表 250
10.2.2 重点说明 250
10.2.3 例程 251
10.3 Modbus 寄存器 253
10.3.1 指令列表 253
10.3.2 重点说明 253
10.3.3 例程 254
10.4 Flash/文件读写 257
10.4.1 指令列表 257
10.4.2 重点说明 258
10.4.3 例程 259
第十一章 总线相关 267
11.1 总线初始化 267
11.1.1 指令列表 267
11.1.2 重点说明 267
11.1.3 例程 272
11.2 SDO/PDO 读写 277
11.2.1 指令列表 277
11.2.2 重点说明 278

11.2.3 例程.....278
11.3 总线转矩力矩.....281
11.3.1 指令列表.....281
11.3.2 重点说明.....282
11.3.3 例程.....282
第十二章 其他指令.....285
12.1 在线命令.....285
12.1.1 指令列表.....285
12.1.2 重点说明.....285
12.1.3 例程.....286
12.2 控制器 RTBasic 相关.....302
12.2.1 指令列表.....302
12.2.2 重点说明.....302
12.2.3 例程.....303
12.3 控制器自动上报.....306
12.3.1 指令列表.....306
12.3.2 重点说明.....307
12.3.3 例程.....307
12.4 调试相关.....312
12.4.1 指令列表.....312
12.4.2 重点说明.....312
12.4.3 例程.....319
12.5 RTSys 辅助调试.....329
12.5.1 RTSys 简介.....329
12.5.2 RTSys 如何获取.....329
12.5.3 RTSys 如何配合调试.....330
第十三章 指令详解.....345
第十四章 指令返回值.....550
14.1 指令返回值.....550
14.2 指令返回值详细说明.....550
14.3 其他：快速入门/常见问题.....560
14.3.1 快速入门.....560
14.3.2 常见问题.....569
Zmotion PC 编程手册・高级功能篇.....582
第一章 指令列表.....521
第二章 轴叠加/电子凸轮.....523
2.1 运动叠加.....523
2.1.1 指令列表.....523
2.1.2 重点说明.....523
2.1.3 例程.....523
2.2 电子凸轮同步追踪.....526
2.2.1 指令列表.....526

2.2.2 重点说明.....526
2.2.3 例程.....528
第三章 高速锁存应用.....547
3.1 高速锁存.....547
3.1.1 指令列表.....547
3.1.2 重点说明.....547
3.1.3 例程.....550
第四章 位置比较输出.....558
4.1 位置比较输出应用.....558
4.1.1 指令列表.....558
4.1.2 重点说明.....558
4.1.3 例程.....559
第五章 PT 运动.....584
5.1 PT 运动.....584
5.1.1 指令列表.....584
5.1.2 重点说明.....584
5.1.3 例程.....584
第六章 机械手控制运动应用.....593
6.1 机械手模型.....593
6.1.1 指令列表.....593
6.1.2 重点说明.....593
6.1.3 例程.....594
第七章 指令详解.....603

Zmotion PC 编程手册·基本功能篇

第一章 产品描述说明

1.1 正运动控制器特点

运动控制器支持 PC 直接在线控制，提供 DLL 函数库和 VC，VB，C#，PYTHON，LABVIEW 等例程。函数库同时提供 WINCE 和 LINUX 的支持。函数库针对于所有型号的控制器。



运动控制器在线控制相对 PCI 具有下面的优势：

1、不使用插槽，稳定性更好。

2、降低对 PC 的要求，不需要 PCI 插槽。

3、可以选用 MINI 电脑或 ARM 工控电脑，降低整体成本。

4、控制器直接做接线板使用，节省空间。

5、控制器上可以并行运行程序，与 PC 只需要简单交互，降低 PC 软件的复杂性。

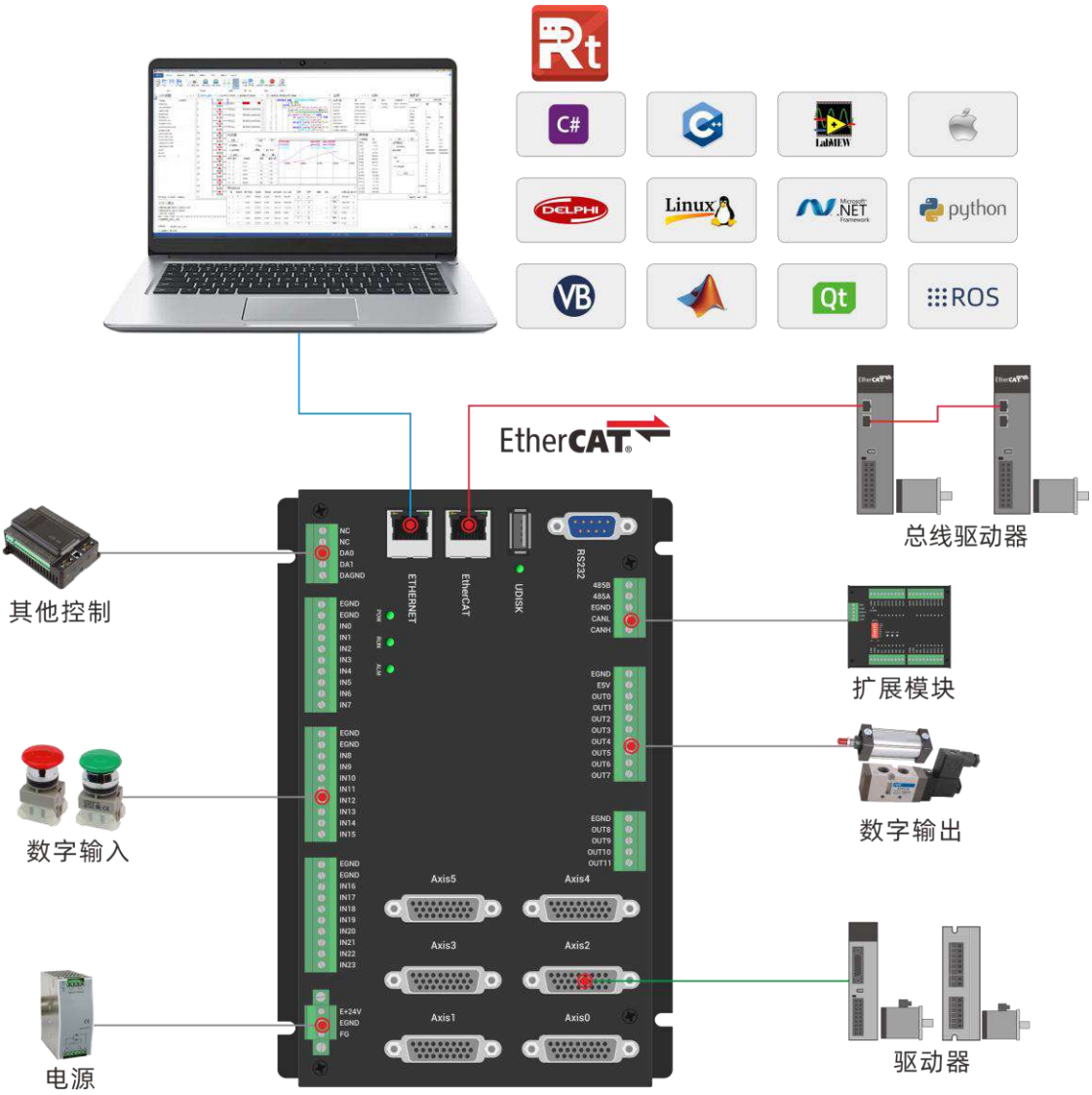
6、运动控制器就是控制电动机运行方式的专用控制器，控制结构模式一般是：控制器+驱动器+（步进或伺服）电机。

7、控制器支持速度前瞻、电子凸轮、电子齿轮、螺距补偿、固步跟踪、运动叠加、虚拟轴、脉冲闭环、硬件位置锁存、位置比较输出、连续插补、运动暂停等功能。

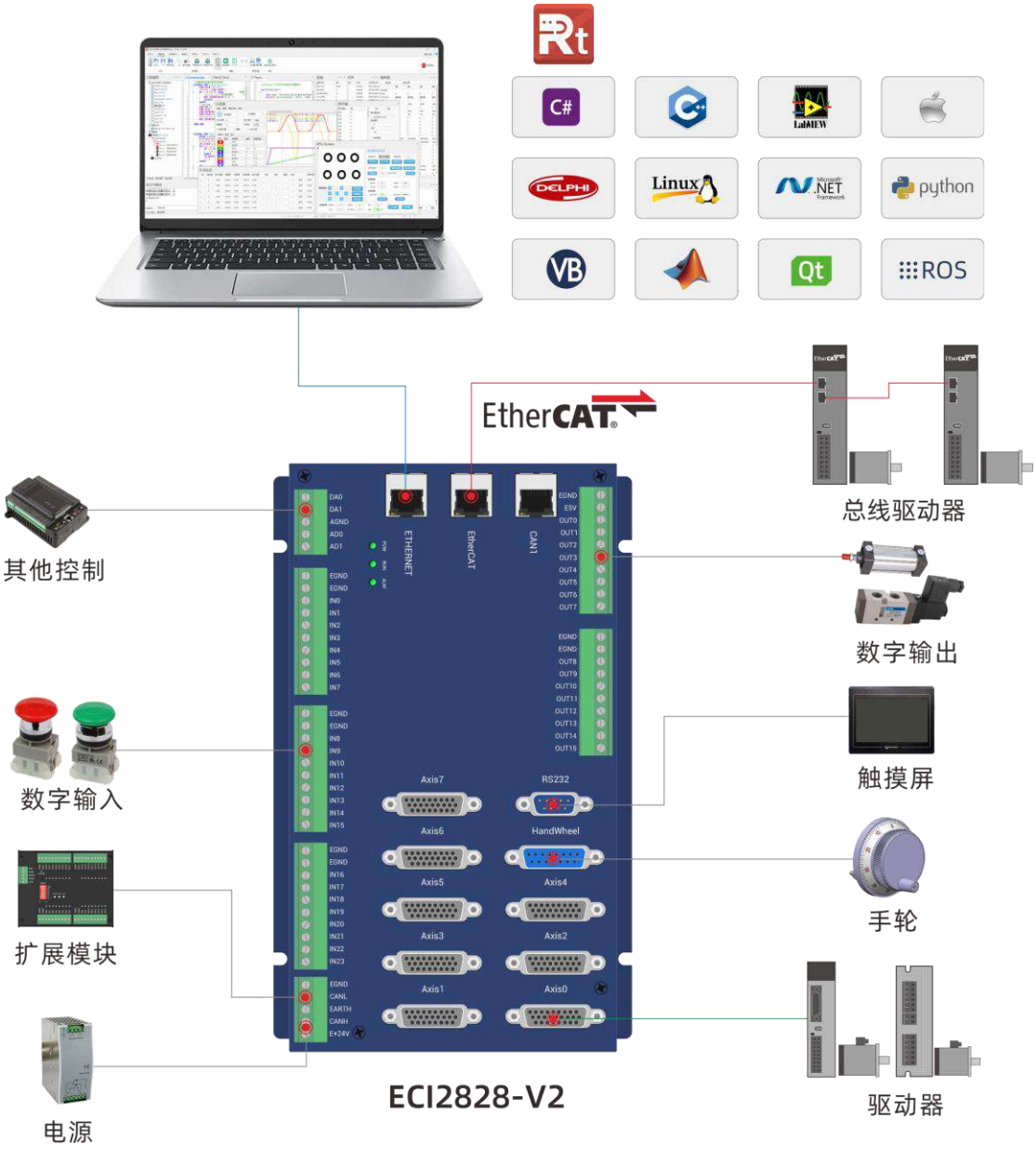
综上可以看出，选用以太网接口的运动控制器来代替 PCI 运动控制卡，可以节省空间，降低成本，优化程序，接线更方便，这也是越来越多的应用采用以太网的原因。

1.2 产品架构

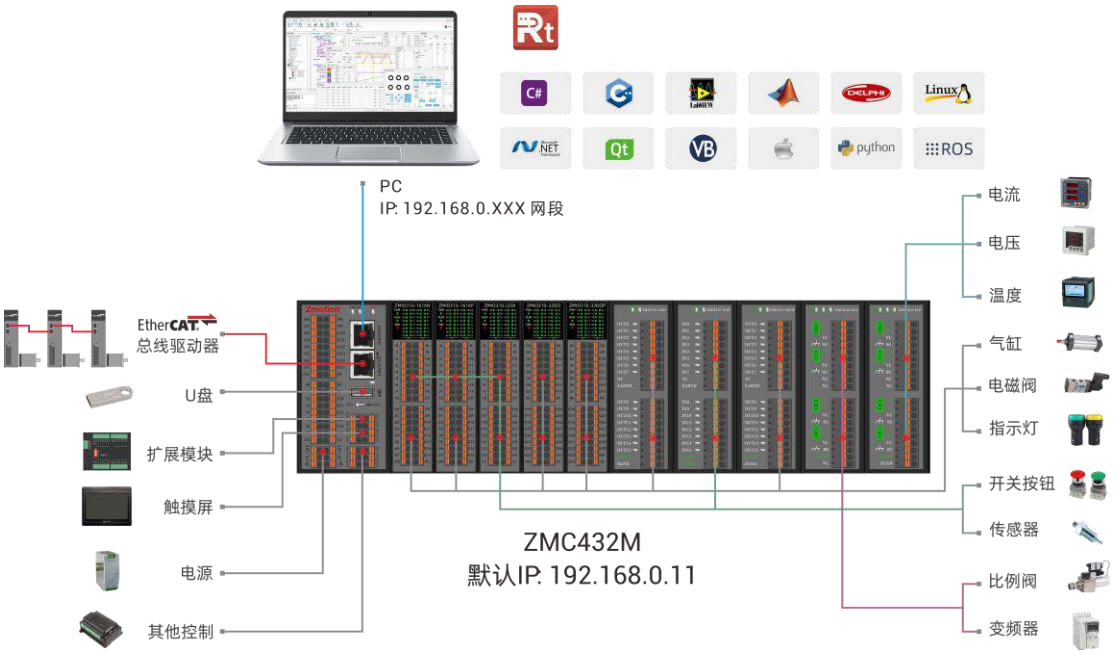
ZMC432CL-V2 架构图：



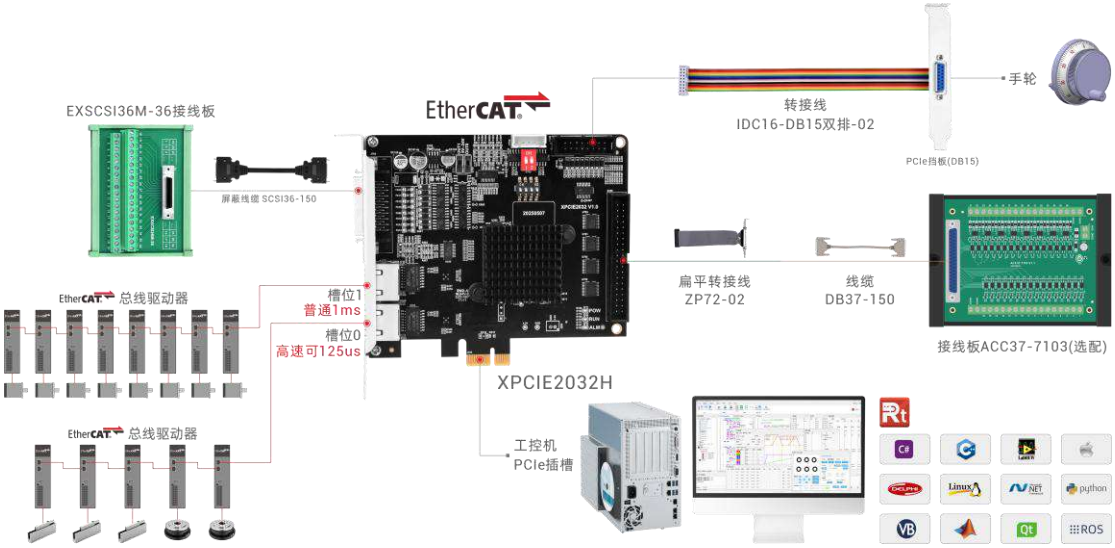
ECI2828-V2 架构图：



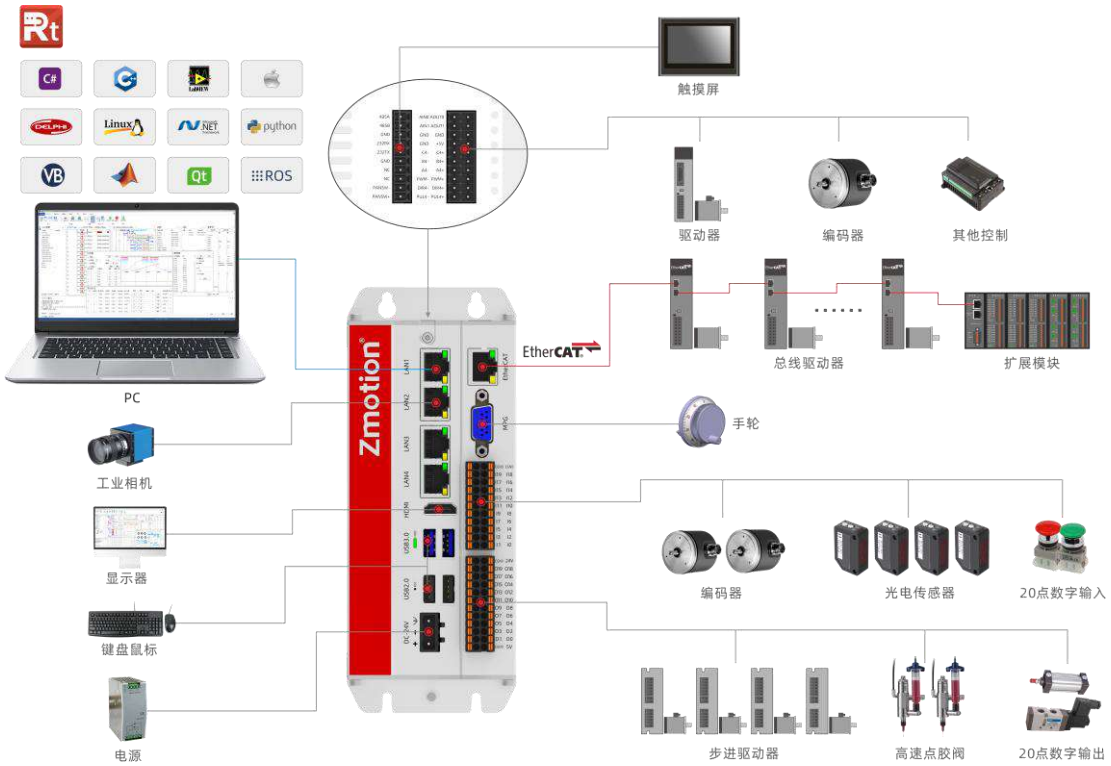
ZMC432M 架构图：

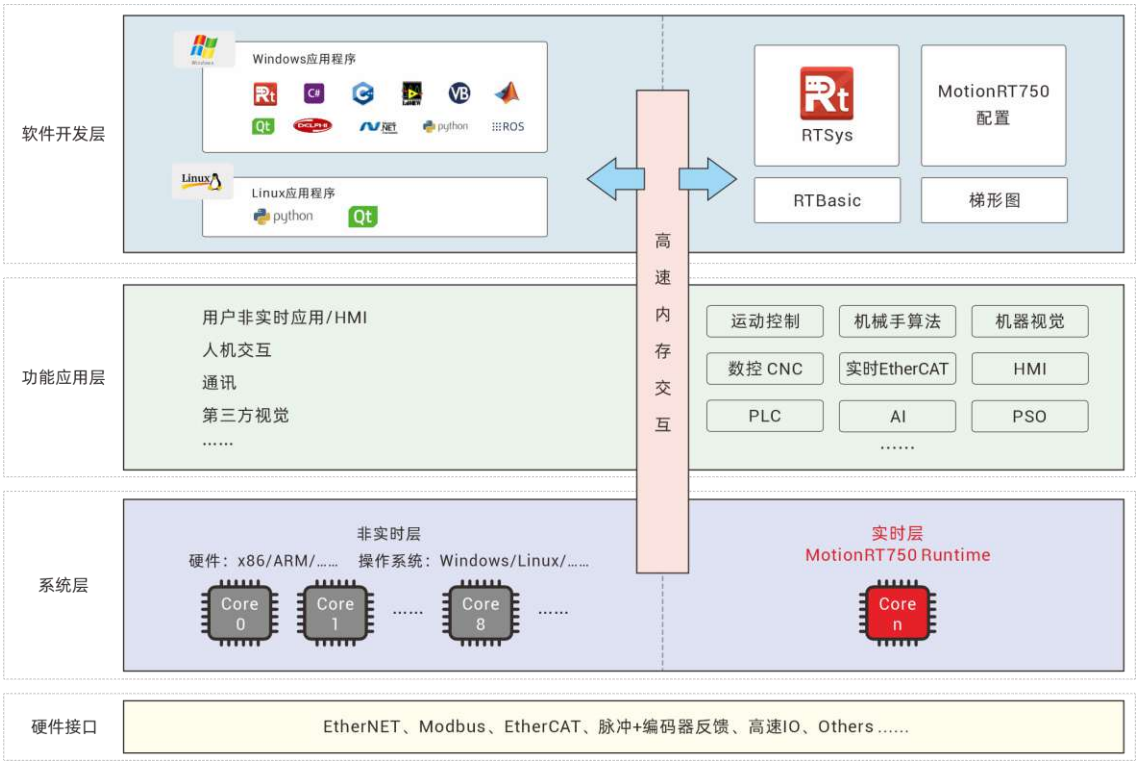


XPCIE2032H 架构图：



VPLC712 架构图：


触摸屏
驱动器
编码器
其他控制
总线驱动器
扩展模块
EtherCAT
手轮
PC
工业相机
显示器
键盘鼠标
电源
编码器
光电传感器
20点数字输入
步进驱动器
高速点胶阀
20点数字输出


软件开发层
Windows应用程序
Linux应用程序
RTSys
MotionRT750 配置
RTBasic
梯形图
高速内存交互
功能应用层
用户非实时应用/HMI
人机交互
通讯
第三方视觉
.....
运动控制
机械手算法
机器视觉
数控 CNC
实时EtherCAT
HMI
PLC
AI
PSO
.....
系统层
非实时层
硬件：x86/ARM/.....
操作系统：Windows/Linux/.....
Core 0
Core 1
.....
Core 8
.....
实时层
MotionRT750 Runtime
Core n
硬件接口
EtherNET、Modbus、EtherCAT、脉冲+编码器反馈、高速IO、Others .....

第二章 函数库的使用与封装

2.1 函数库调用

2.1.1 Windows 系统

2.1.1-1 VC++6.0

1) 打开 VC++6.0

2) 新建一个工程

3) 选择[MFC APPWizard(exe)]

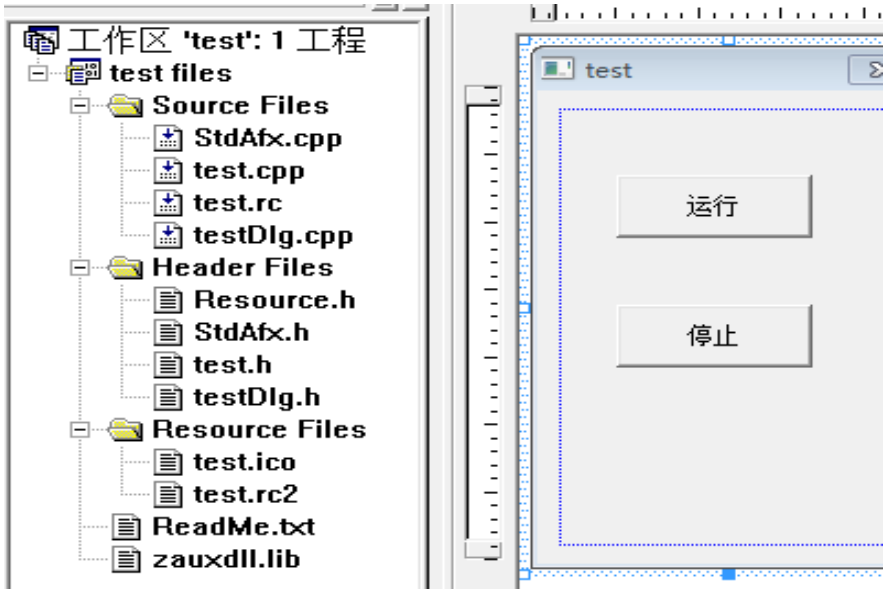
4) 选择工程保存路径

5) 设置工程名，选择[确定]



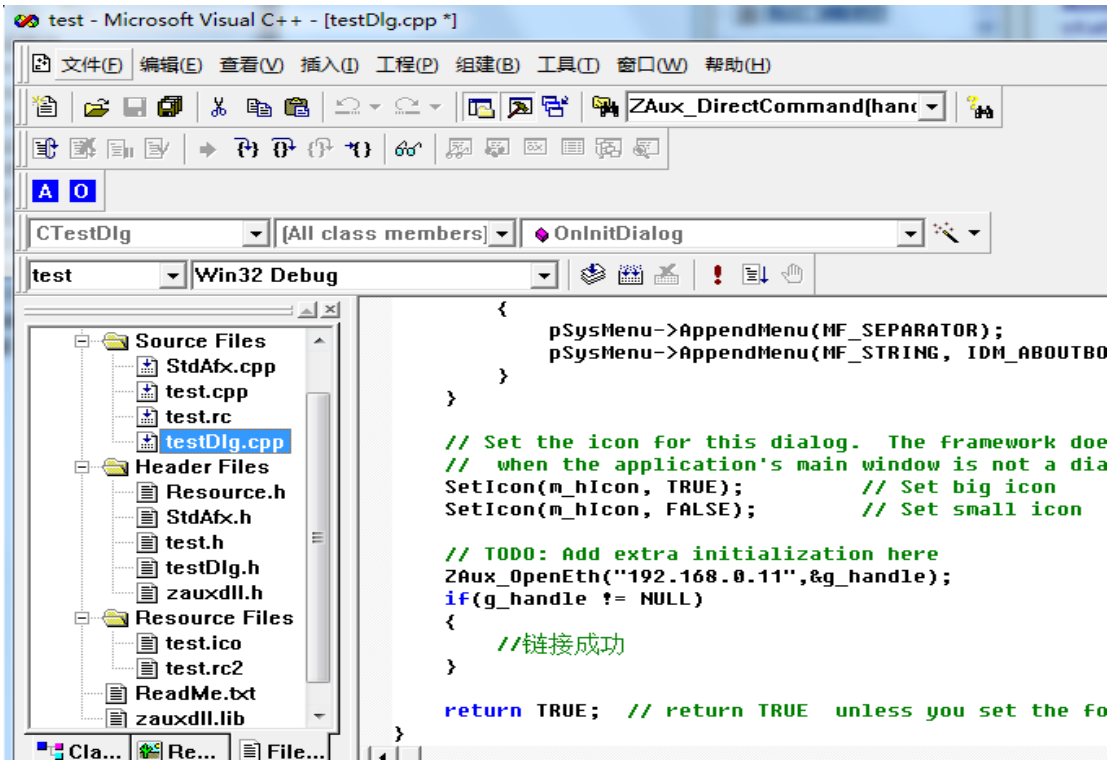
6) 在应用程序类型选择基本对话框。完成新建

7) 将提供的库函数 zauxdll.h、zauxdll.lib、zauxdll.dll 及 zmotion.dll 放到工程路径下（或者直接将通过 zauxdll.cpp 的源码加入工程）

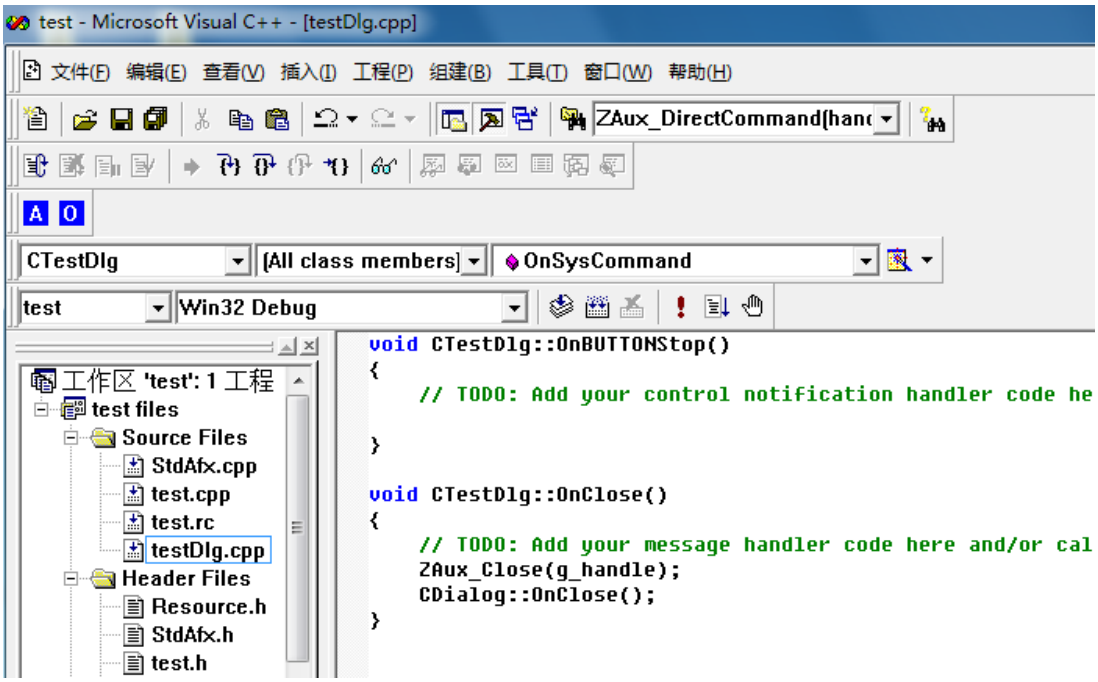


8) 在 testDlg.cpp 添加头文件#include “zauxdll.h”

9) 在工程文件中，找到 CTestDlg::OnInitDialog( )函数，在括号中添加连接控制器的代码：ZAux_OpenEth(“192.168.0.11”, &g_handle)



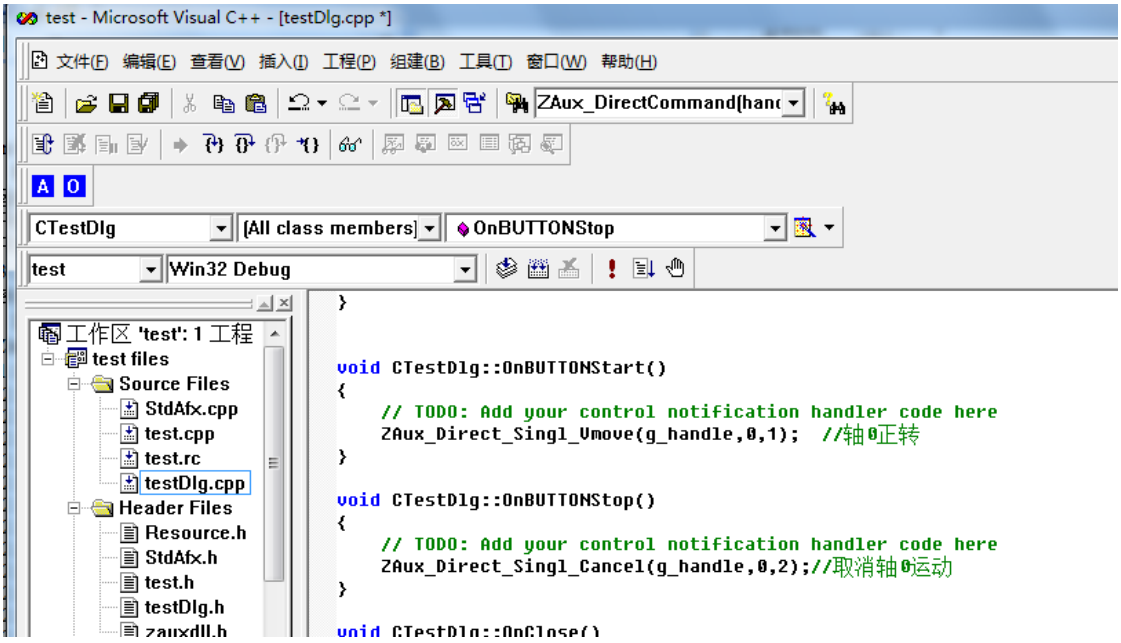
10) 在 CTestDlg 添加一个成员函数 CTestDlg::OnClose();用于关闭窗口断开控制器连接。在函数内添加 ZAux_Close(g_handle); 断开连接



11) 双击启动、停止在对应的事件函数下添加对应的代码：

ZAux_Direct_Single_Vmove(g_handle,0,1);

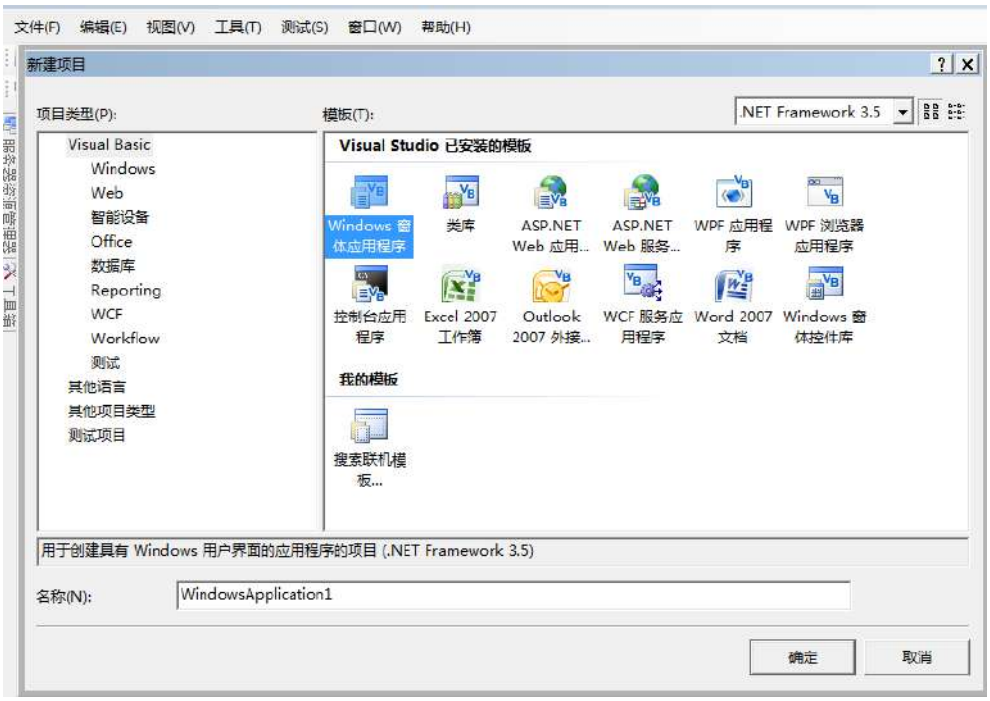
ZAux_Direct_Single_Cancel(g_handle,0,2);



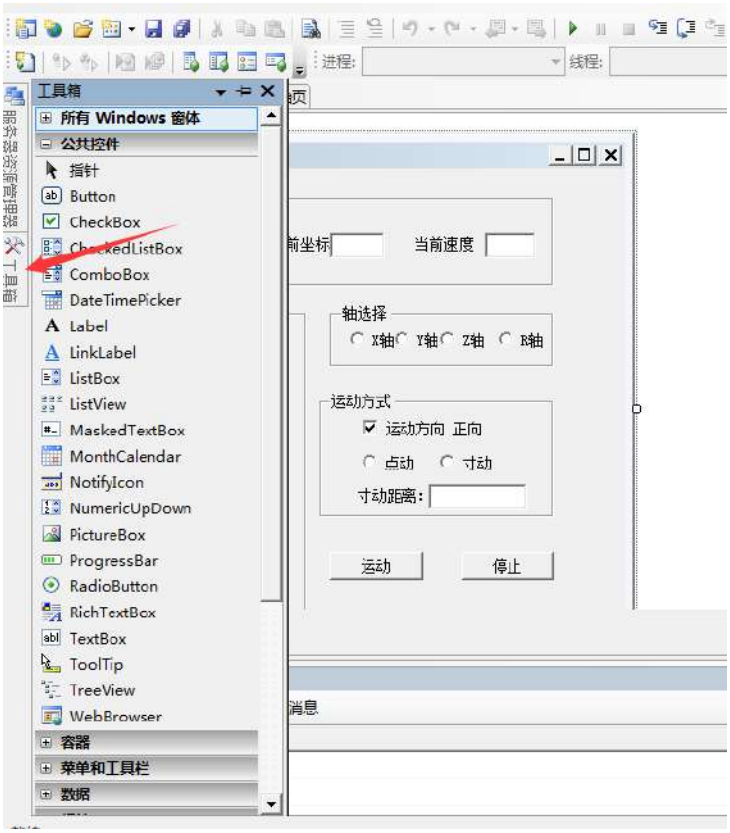
12) 编译程序后，运行程序，在界面下按下运行，则轴 0 会按初始化速度正向转动，按下停止，轴 0 停止运动。

2.1.1-2 VB.NET

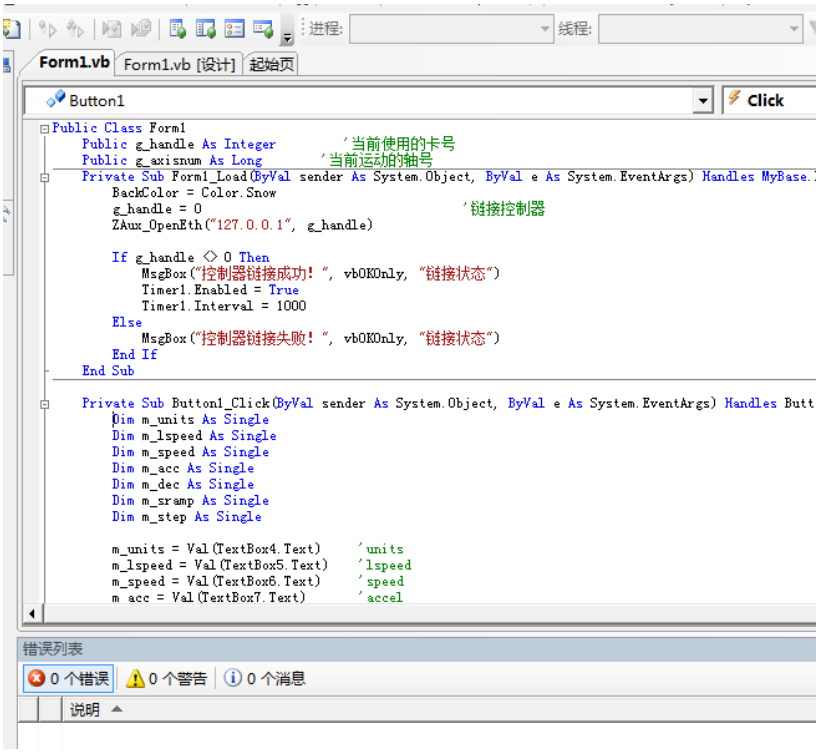
1) 建立项目，保存



2) 根据需要的功能和界面，选择相应的控件来完成操作界面



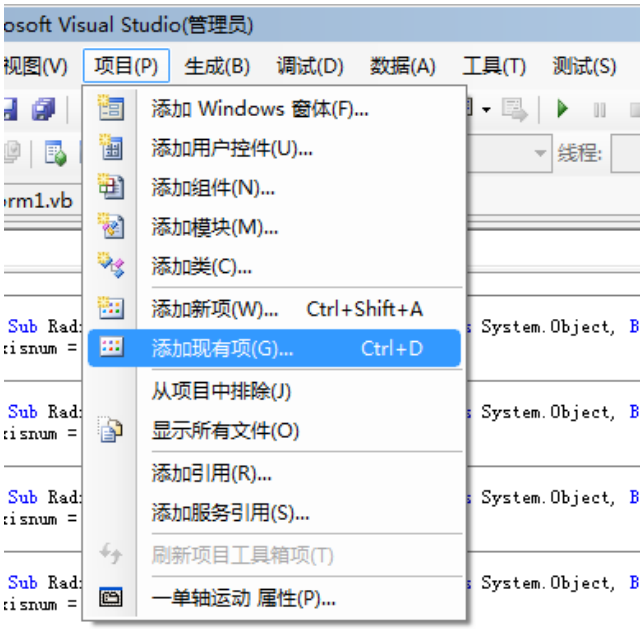
3) vb.net 是面向对象的编程语言，选择合理的触发事件（如 Button_Click、Checked、Load 等）来编写程序



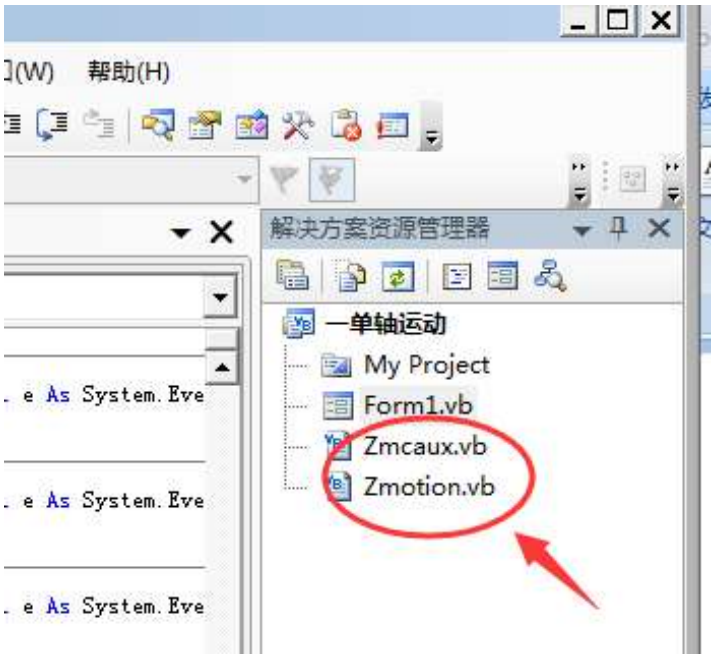
4) 添加 dll 文件和头文件，放在和项目文件同一文件夹内



5) 在项目内添加头文件（两个.vb 文件）

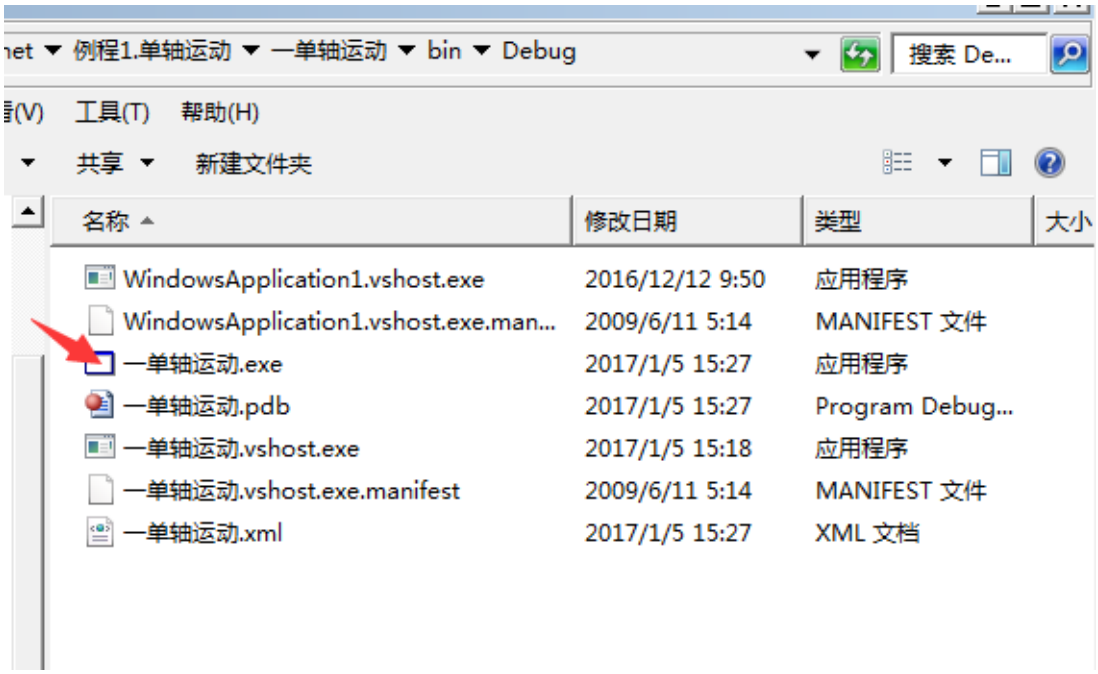


添加成功后如下图显示：



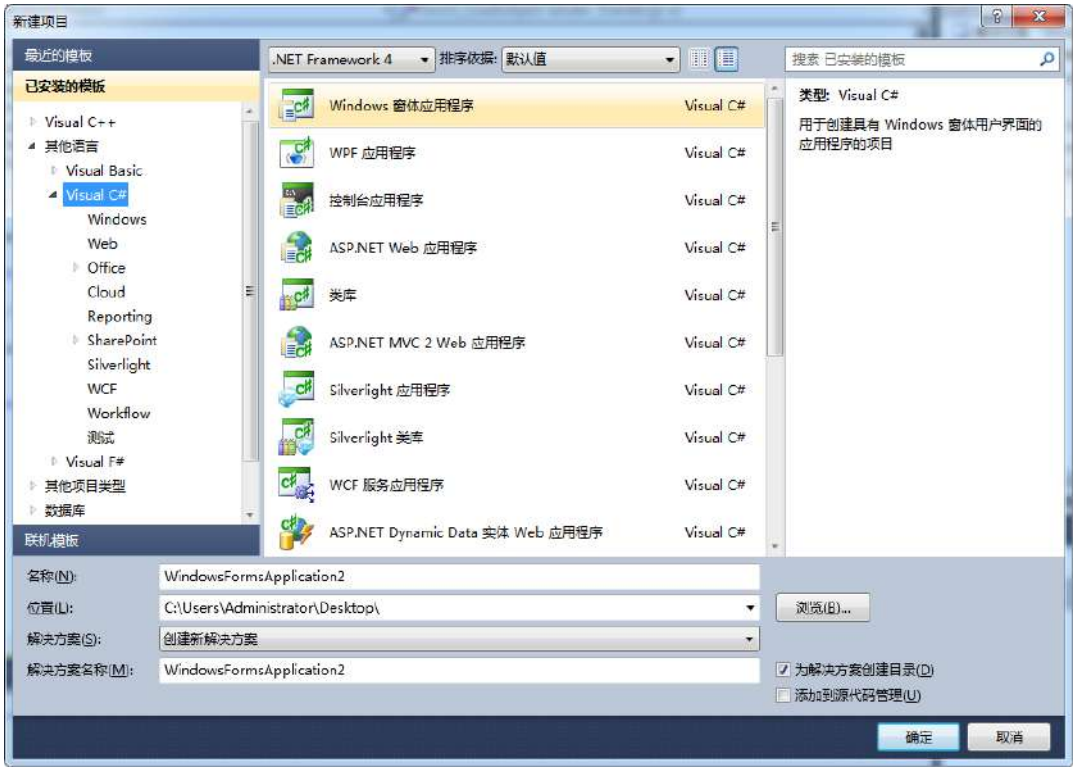
6) PC 连接上控制器后，设置好 IP，然后启动调试

7) 程序运行调试之后，在文件夹：“bin\Debug”里面有个 .exe 文件，可以直接运行工作

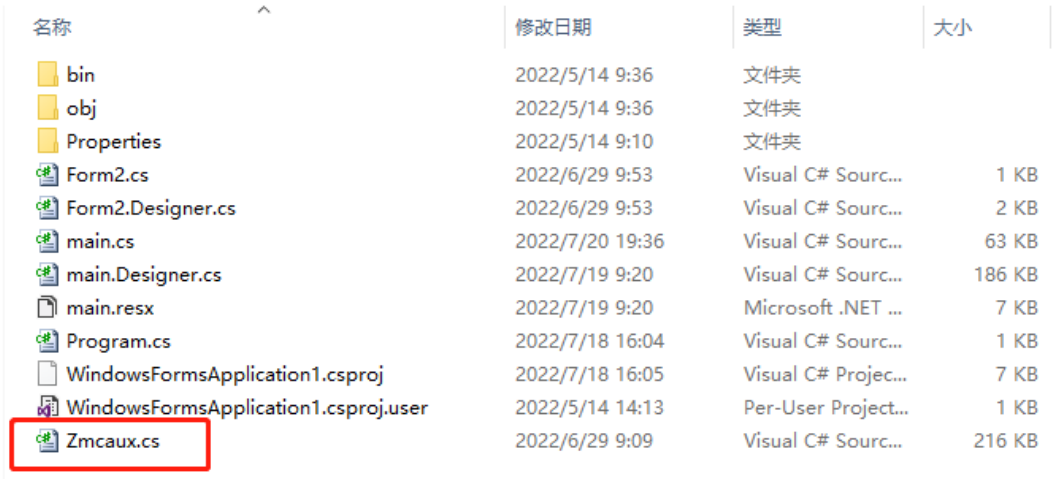


2.1.1-3 C#

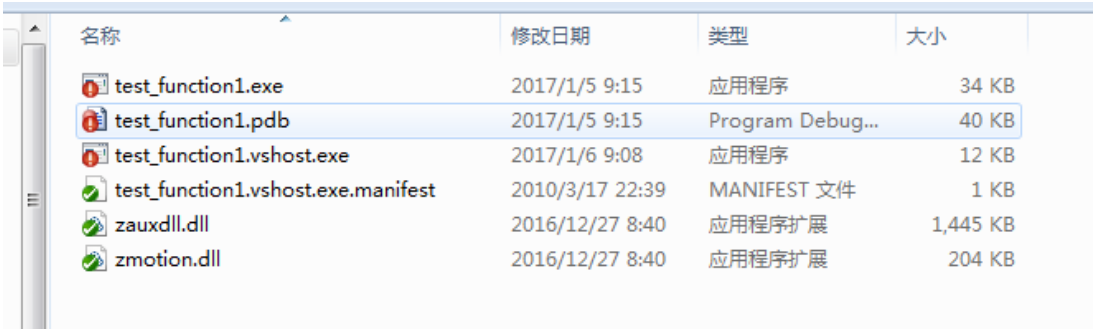
1) 打开 VS2012，点击[文件栏]的[新建]—[项目]，选择[windows 窗体应用程序]



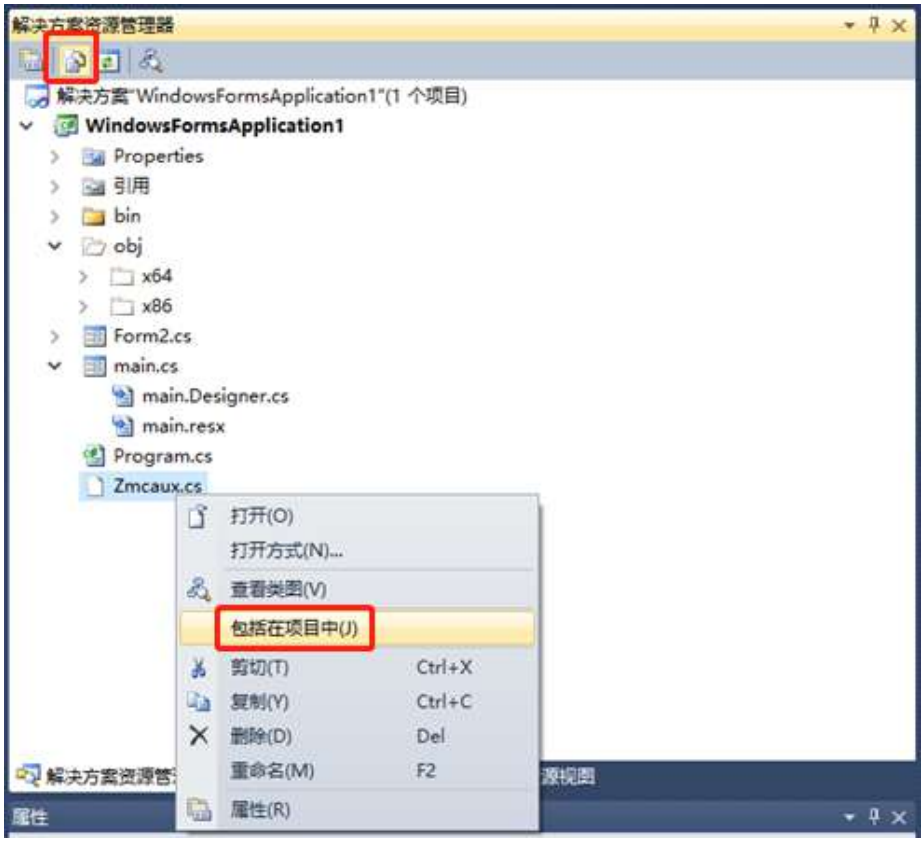
2) 找到新建项目的文件夹，将 zmcaux.cs 文件放入文件夹中



3) 将 zauxdll.dll 和 zmotion.dll 文件放入 debug 文件夹中



4) 用 VS 打开新建的项目文件，在右边的解决方案资源管理器中点击[显示所有文件]，然后鼠标右键点击[zmcaux.cs 文件]，点击[包括在项目中]



5) 双击[form1]，出现代码编辑界面，在文件开头写入 using cszmcaux

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Threading;                    //线程声明
using System.Text.RegularExpressions;//正则表达式声明

using cszmcaux;
namespace WindowsFormsApplication1
{
    public partial class main : Form
    {
        public IntPtr g_handle; //声明控制器连接句柄
```

6) 开始编程

2.1.1-4 LABVIEW

1. 找到厂家提供的光盘资料里面的 LabVIEW 的 VI 库文件，路径如下。

1) 进入厂商提供的光盘资料找到“04PC 函数”文件夹，并点击进入。



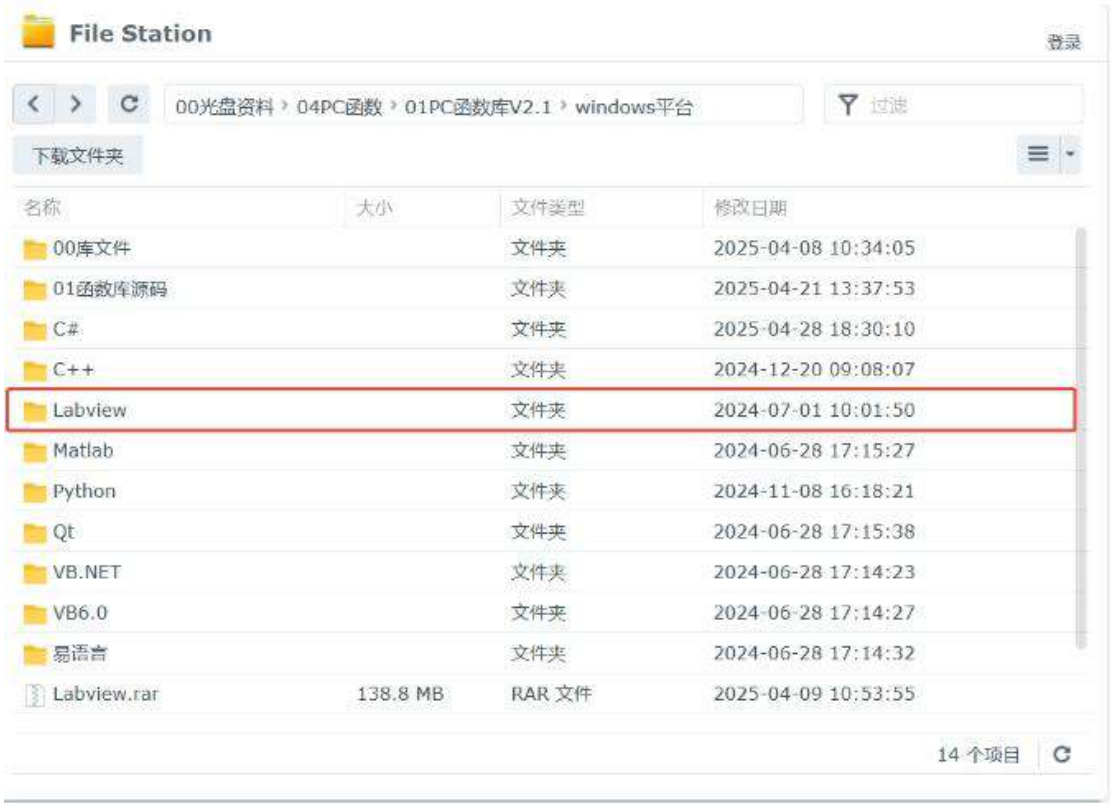
2) 选择“01PC 函数库 V2.1”文件夹。



3) 选择“Windows 平台”文件夹。



4) 选择“Labview”文件夹，里面有 32 位和 64 位的动态库和例程。





2. 将厂商提供的 LabVIEW 的 VI 库文件复制到 LabVIEW 安装路径下 LabVIEW/user.lib 的文件夹内。

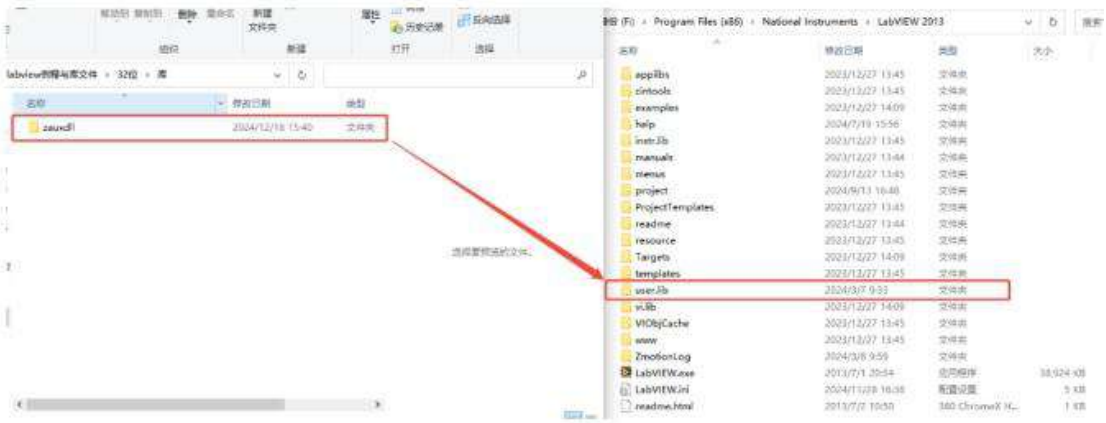
1) 找到解压的 VI 库文件夹（zaudll 是 32 位的 VI 库，64 的 VI 库是 newZauxDll64）。



2) 选中 LabVIEW 右击打开文件所在位置（这里以 LabVIEW2013-32 位为演示）。

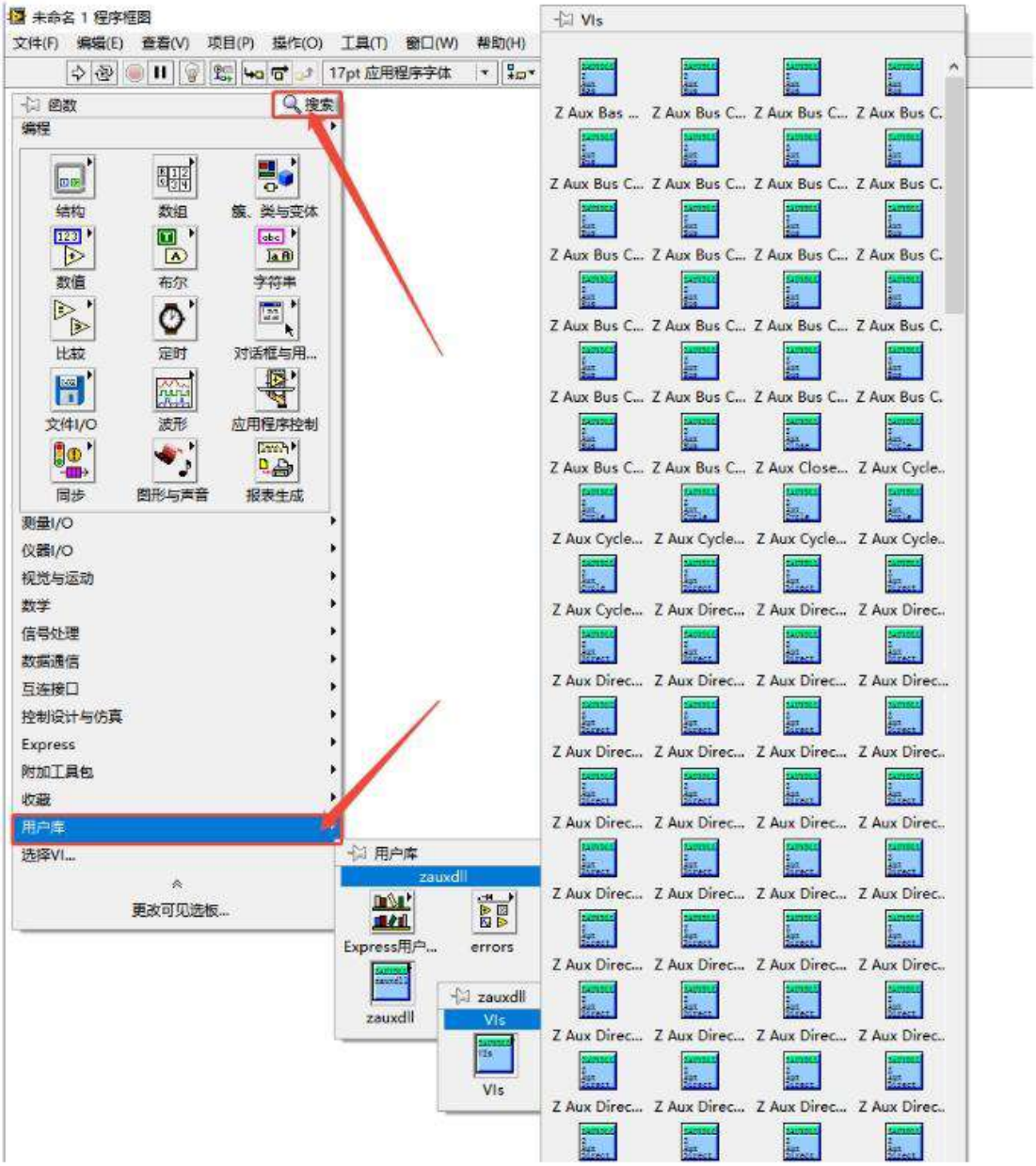


3) 将整个 VI 库文件夹直接复制放到 user.lib 该目录，然后关闭 LabVIEW 软件重新启动。



3. 重新启动 LabVIEW 后，选择新建 VI，然后右键点击 VI 程序框图空白处，选择用户库来找到添加的 VI 库，或者直接搜索 VI，最后将需要的函数直接拖到面板上。





2.1.1-5 Python

1. 导入动态库

1) python 中需要导入 platform 模块（以便获取前操作系统的类型)和 ctypes 模块(导入相关的连接库--动态库主要是 C/C++)

2) 根据不同的平台选择相应的动态库(C/C++)，例如 windowsX86、windowsX64、Linux 等平台的动态库。

```python
import platform
import ctypes

systype = platform.system()
if systype == 'Windows':
    if platform.architecture()[0] == '64bit':
        zauxdll = ctypes.WinDLL('./zauxdll64.dll')
        print('Windows x64')
    else:
        zauxdll = ctypes.WinDLL('./zauxdll.dll')
        print('Windows x86')
elif systype == 'Darwin':
    zmcdll = ctypes.CDLL('./zmotion.dylib')
    print("macOS")
elif systype == 'Linux':
    zmcdll = ctypes.CDLL('./zmotion.so')
    print("Linux")
else:
    print("OS Not Supported!!")

```

2. Python 函数封装

1) 进行函数封装，结合动态库使用

2) 根据需求去封装对应的函数

```python
class ZAUXWrapper:
    def __init__(self):
        self.handle = ctypes.c_void_p()
        self.sys_ip = ""
        self.sys_info = ""
        self.is_connected = False

    def search(self, console=[]):
        iplist = ctypes.create_string_buffer(b'', 1024)  # create_string_buffer创建的是一个 ANSI 标准的 C类型字符串
        zauxdll.ZAux_SearchEth(ctypes.byref(iplist), 1024, 200)
        s = iplist.value.decode()
        str_iplist = s.split()
        num = len(str_iplist)
        print(num, "Controller(s) Found:")
        print(*str_iplist, sep='\n')
        console.append("Searching...")
        console.append(str(num) + " Controller(s) Found:")
        return str_iplist, num

    def connect(self, ip, console=[]):
        if self.handle.value is not None:
            self.disconnect()
        ip_bytes = ip.encode('utf-8')
        p_ip = ctypes.c_char_p(ip_bytes)
        print("Connecting to", ip, "...")
        ret = zauxdll.ZAux_OpenEth(p_ip, ctypes.pointer(self.handle))
        msg = "Connected"
        if ret == 0:
            msg = ip + " Connected"
            self.sys_ip = ip
```

说明：对需要封装的函数进行封装处理，结合本文档进行处理即可。

3. python 函数自定义函数封装 basic 指令

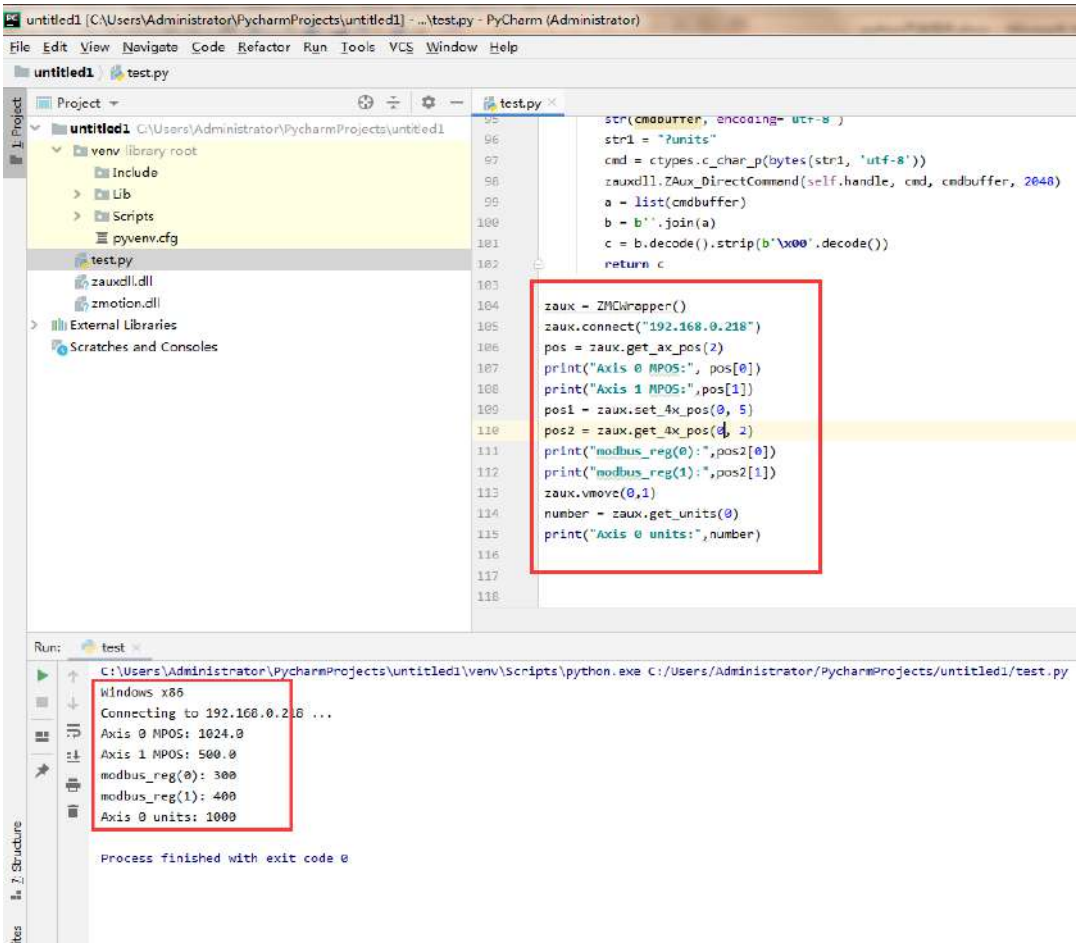
举例说明：

```
85      def vmove(self, iaxis, idir):
86          cmdbuffer = (ctypes.c_char * 2048)()
87          str1 = "VMOVE(%d) AXIS(%d)" % (idir, iaxis)
88          cmd = ctypes.c_char_p(bytes(str1, 'utf-8'))
89          zauxdll.ZAux_DirectCommand(self.handle, cmd, cmdbuffer, 2048)
90          return
91
92      def get_units(self, iaxis):
93          cmdbuffer = (ctypes.c_char * 2048)()
94          str(cmdbuffer, encoding='utf-8')
95          str1 = "?units"
96          cmd = ctypes.c_char_p(bytes(str1, 'utf-8'))
97          zauxdll.ZAux_DirectCommand(self.handle, cmd, cmdbuffer, 2048)
98          a = list(cmdbuffer)
99          b = b''.join(a)
100         c = b.decode().strip(b'\x00'.decode())
101         return c
```

说明：可以将想要一些功能结合 basic 指令组成字符串命令，通过字符串命令和 ZAux_DirectCommand 指令结合进行编程。

4. python 函数使用

举例说明：



说明：创建封装类的对象，然后进行相关函数调用。

2.1.2 LINUX 系统

2.1.2-1 QT

1) 启动 Qt Creator 2.6.1，新建一个工程

2) 将产品配套光盘“..\linux\函数库\x86”文件夹中的动态连接库 zmotion.so、头文件 zmotion.h 复制到工程文件夹中

3) 在 Qt Creator 界面中，打开.pro 文件，并添加

```
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets    (版本兼容问题)
SOURCES + = main.cpp\
            mainwindow.cpp\
            zaux.cpp
HEADERS + = mainwindow.h\
            zmotion.h\
            zaux.h
```

把 main.c 中的头文件改为#include <QApplication>

4) 添加动态库的方法

1>首先右键项目　　3>选择第二个外部库，点击[确定]

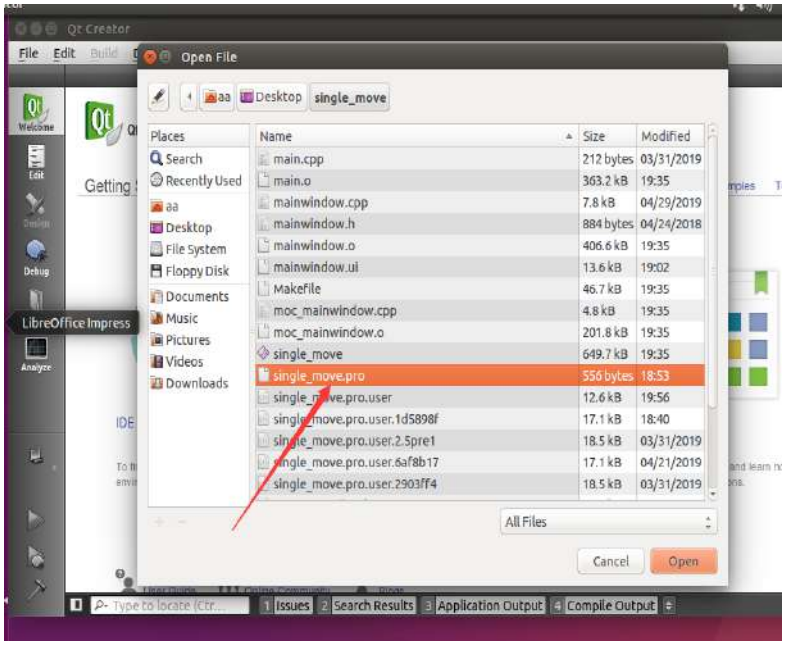
2>点击[添加库]　　4>点击[浏览库文件]

注意事项：

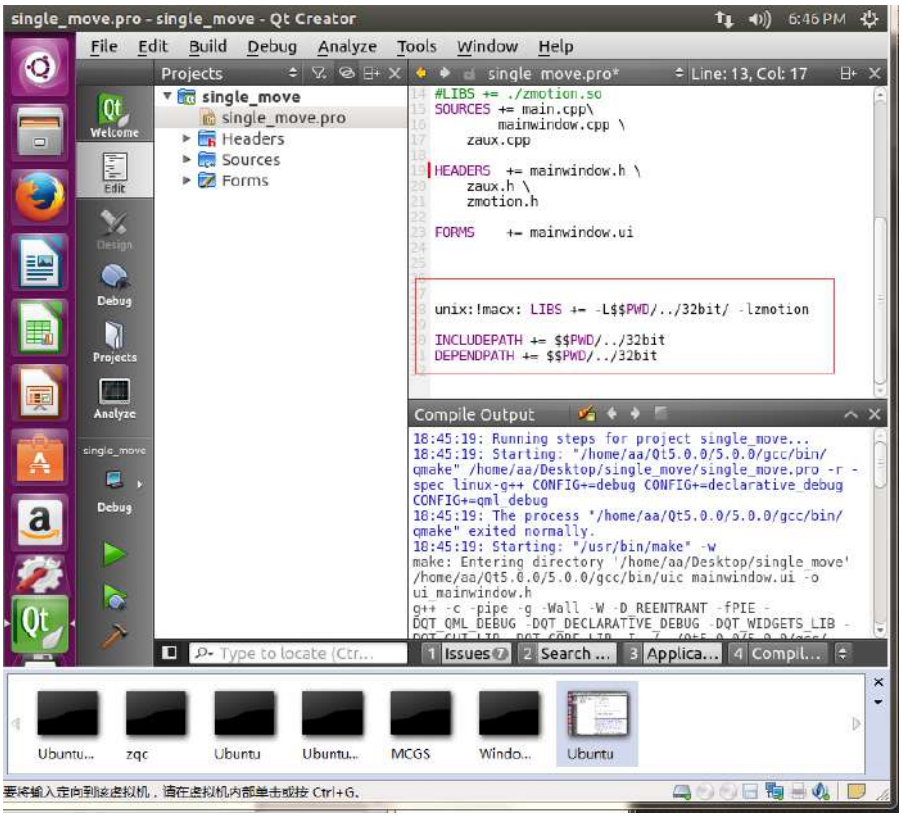
动态库的名字必须是 lib 开头才能识别，然后点击下一步，自动会把动态库的信息加入到.pro 中，下面便可以调用头文件中所包含的函数，可以运行程序。

5) 打开 QT 程序，点击[file]找到例程中的.pro 项目文件点击[open]

注意：QT 工程中不能出现中文路径



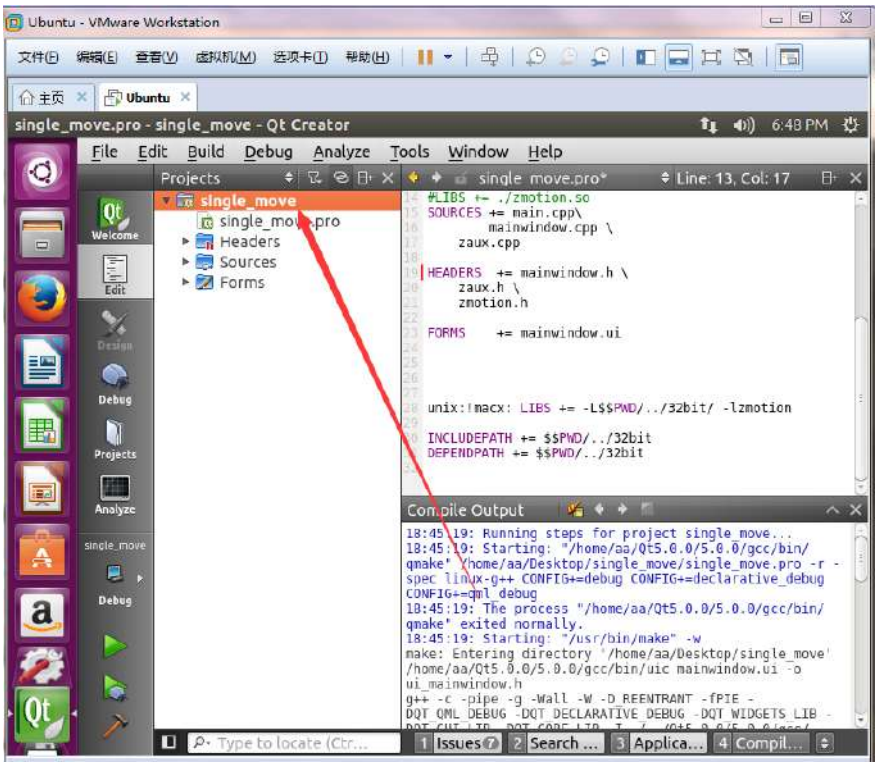
6) 将 QT 例程的.pro 中加入正确路径的动态库，双击[single_move.pro 文件]，查看到之前设置好的动态库路径，这时候要设置自己的动态库路径



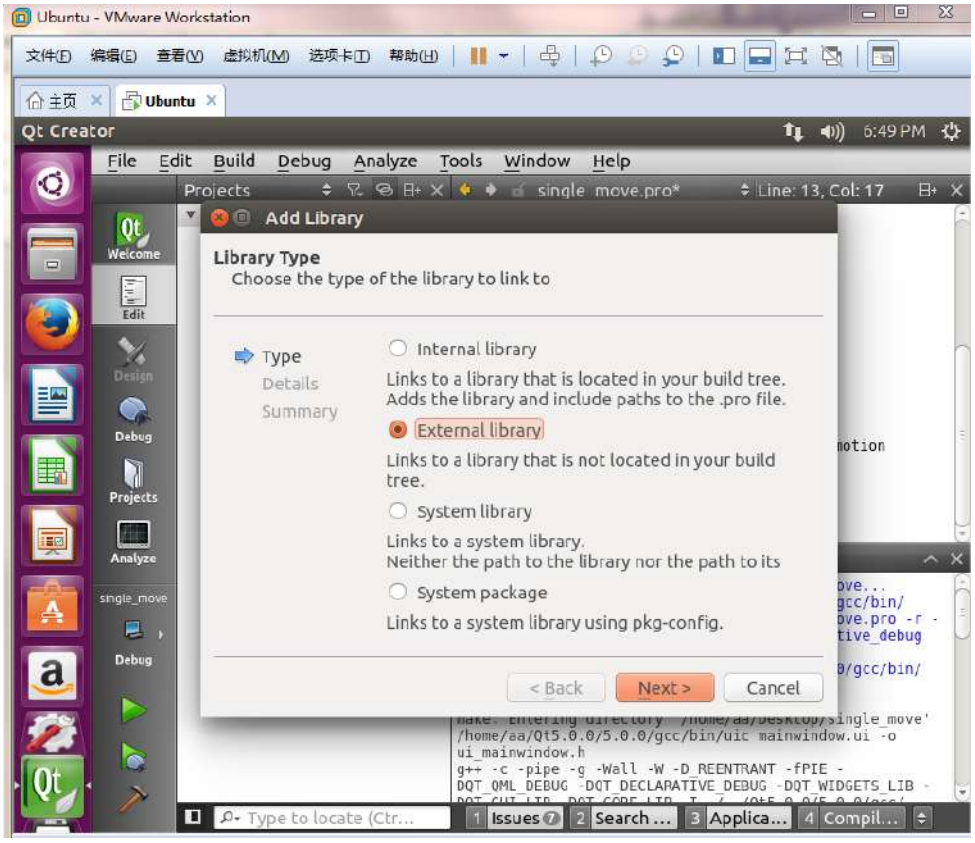
7) 将.pro 文件中的动态库路径进行删除

8) 将文件夹中 32 位的动态库放到指定的路径

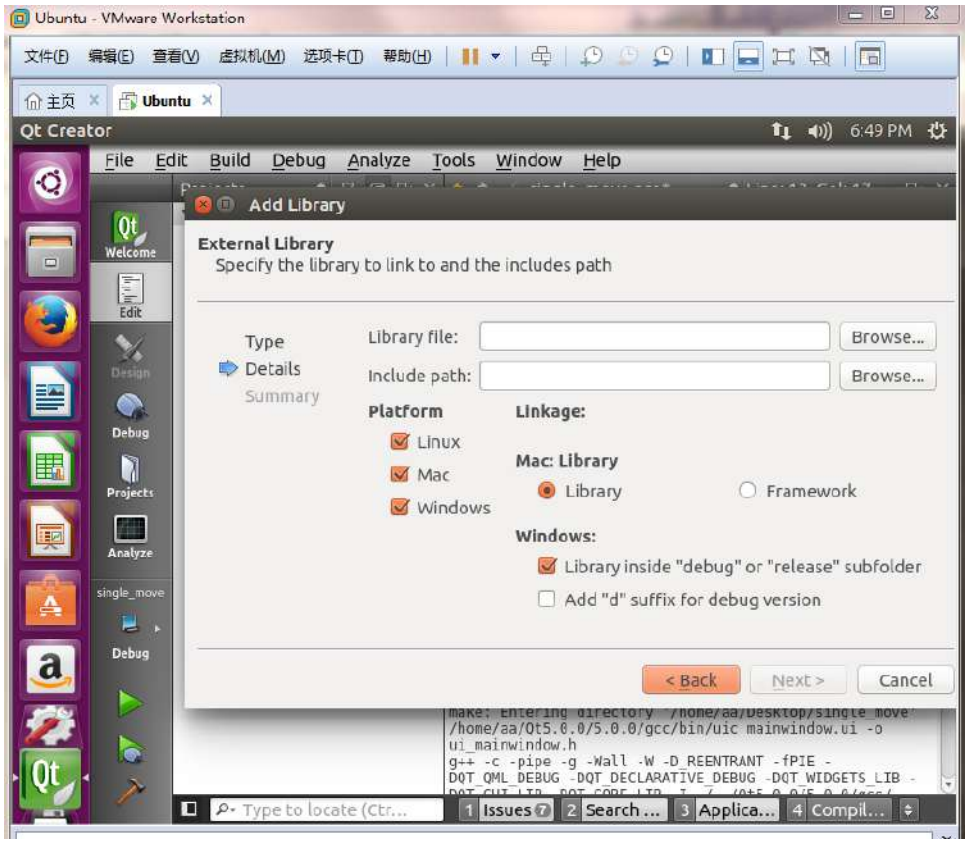
9) 加入自己的动态库路径，首先选中项目文件



10) 再进行右键，点击[add library 添加库]



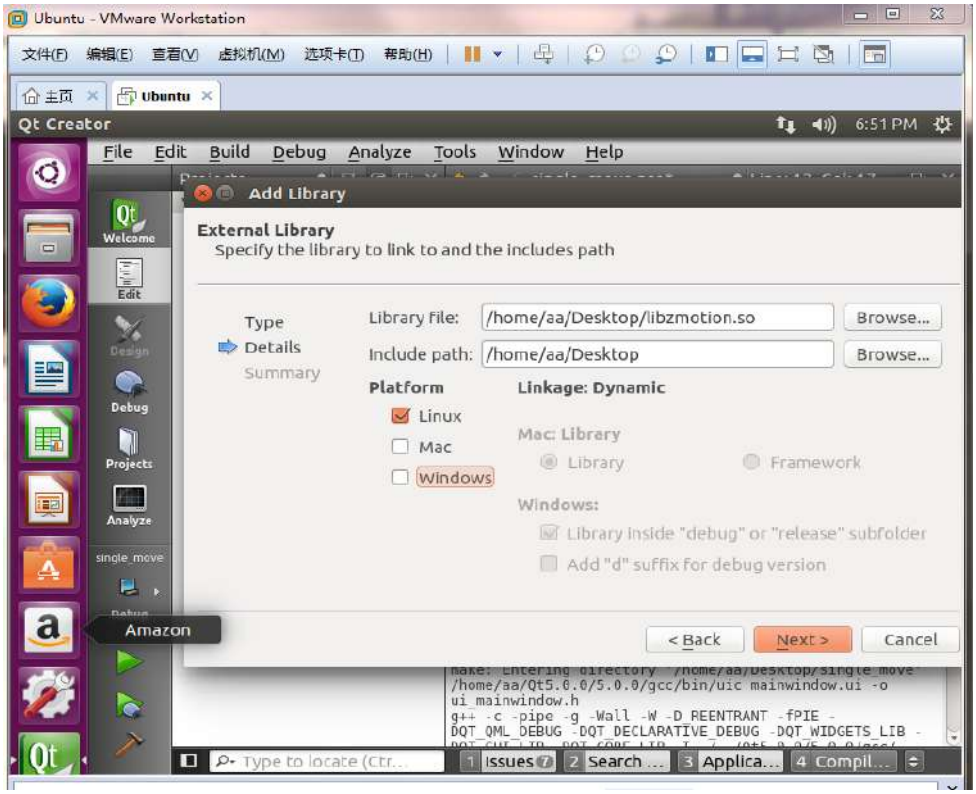
11) 选择第二个[Details]添加外部库



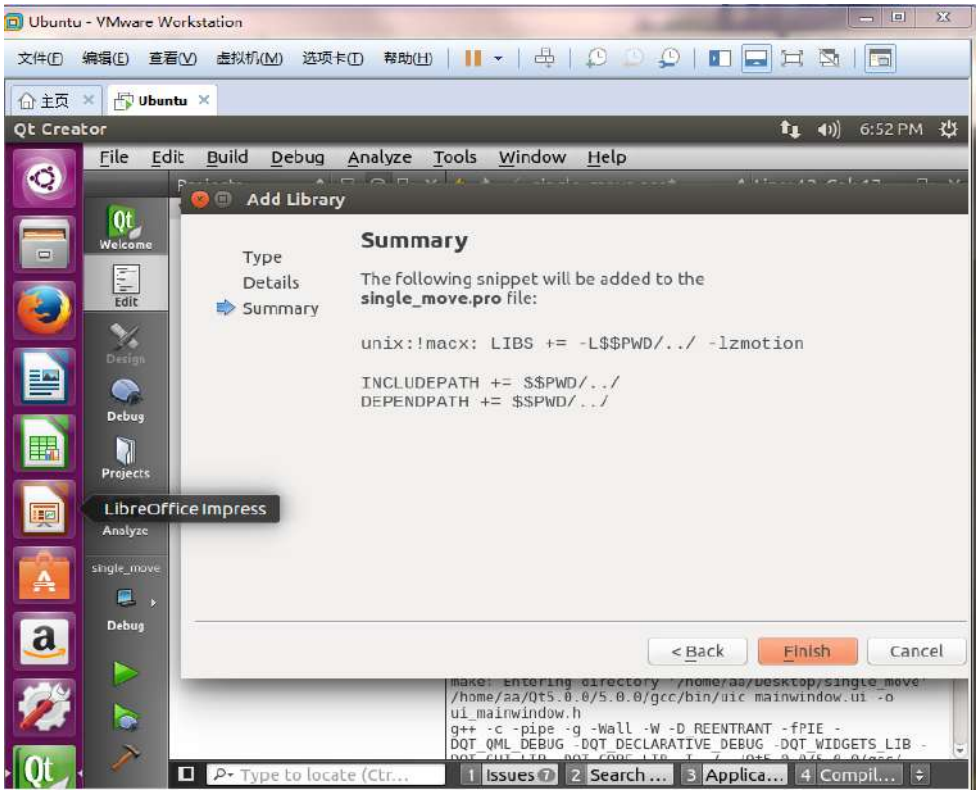
12) 这时候找到刚才放置动态库的路径

注意事项：

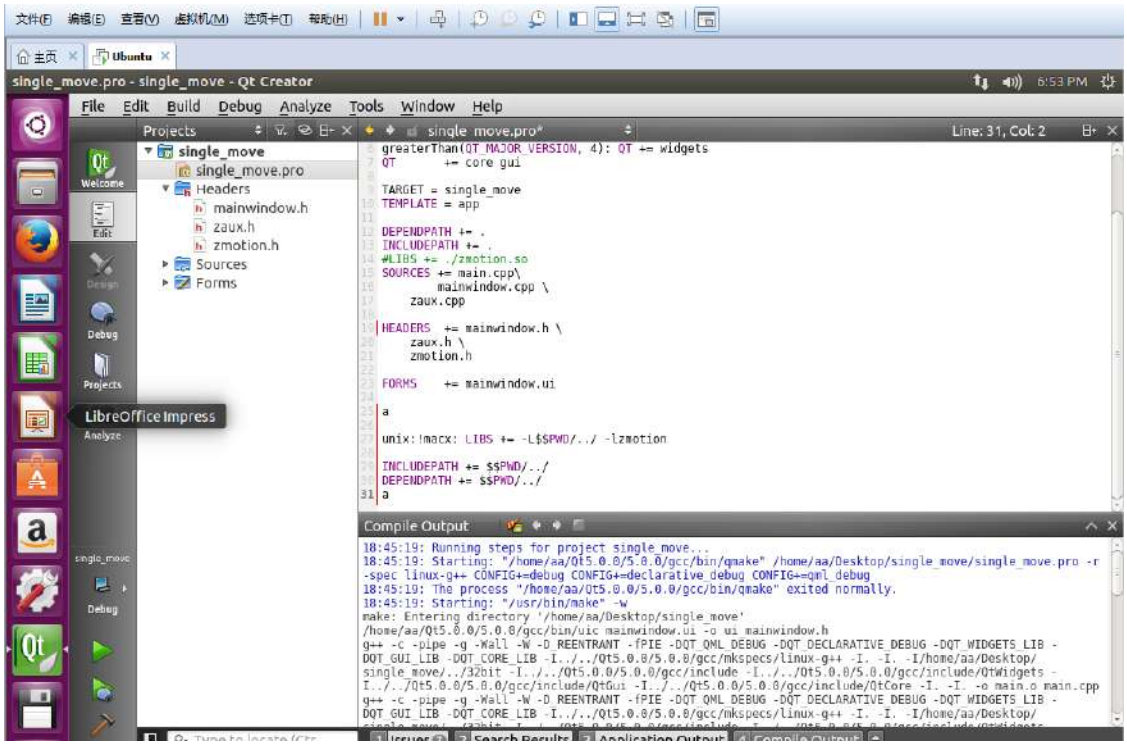
Linux 下面的动态库的名称的话是要 lib 开头才能识别，一般的话就不要轻易去更改文件的名称。



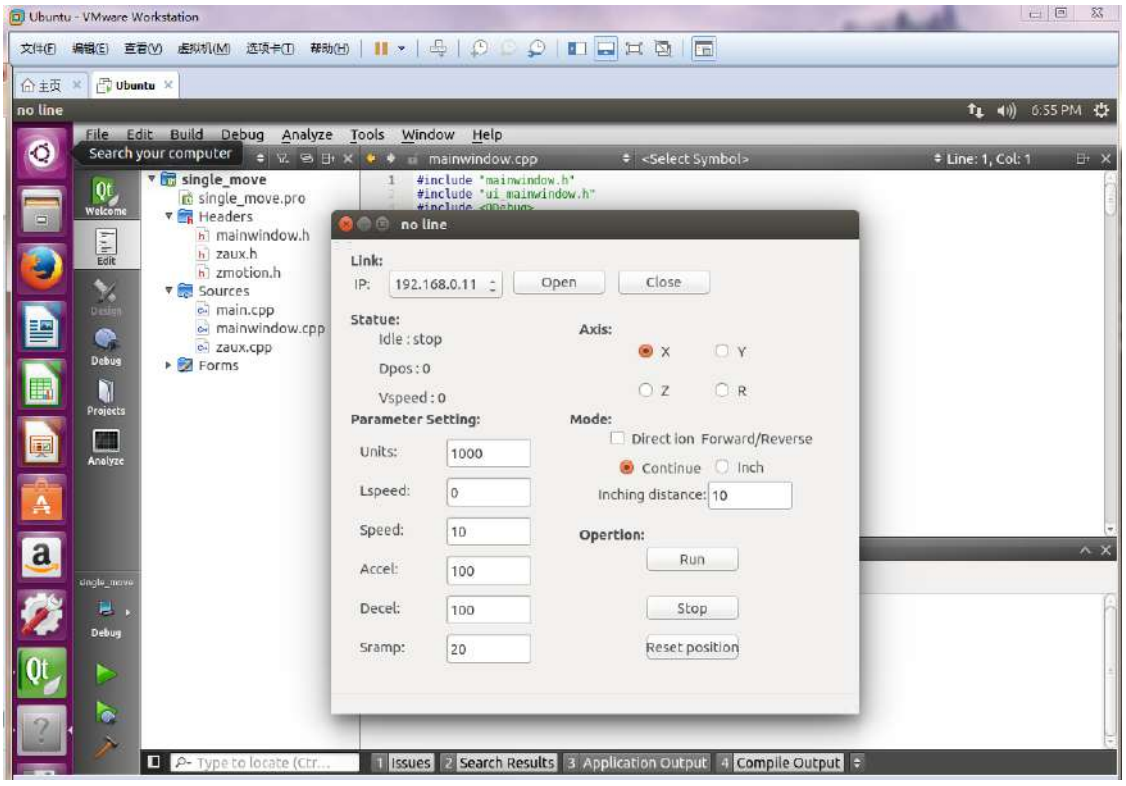
13) 点击[next]进入下一步



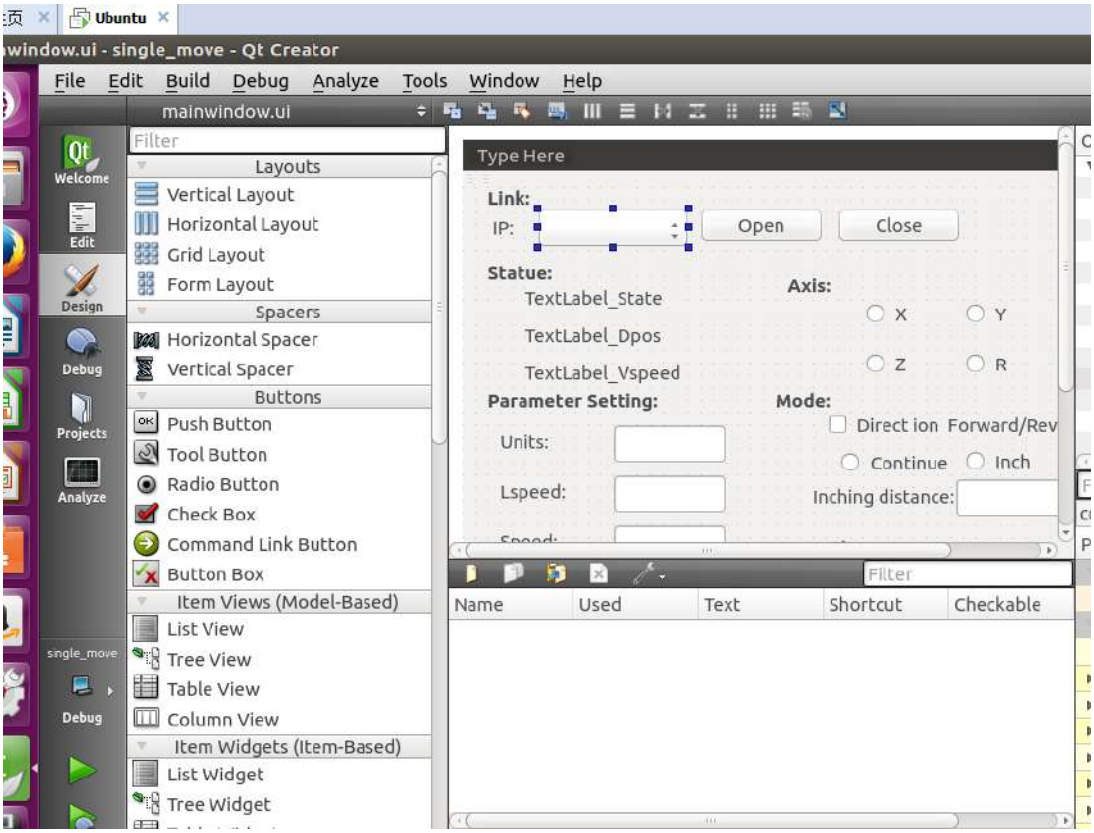
14) 查看.Pro 中生成动态库的路径

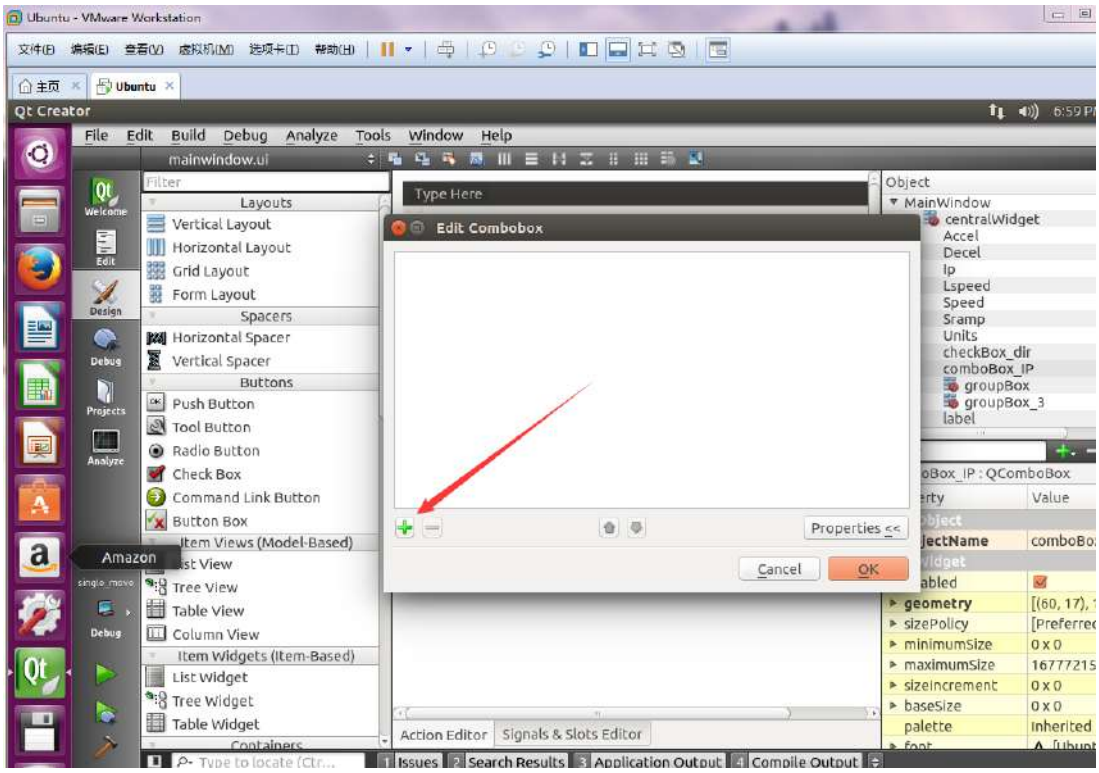


15) 点击运行之后就会出现程序



16) ip 的话可以手动添加，打开 ui 界面，双击点击按钮

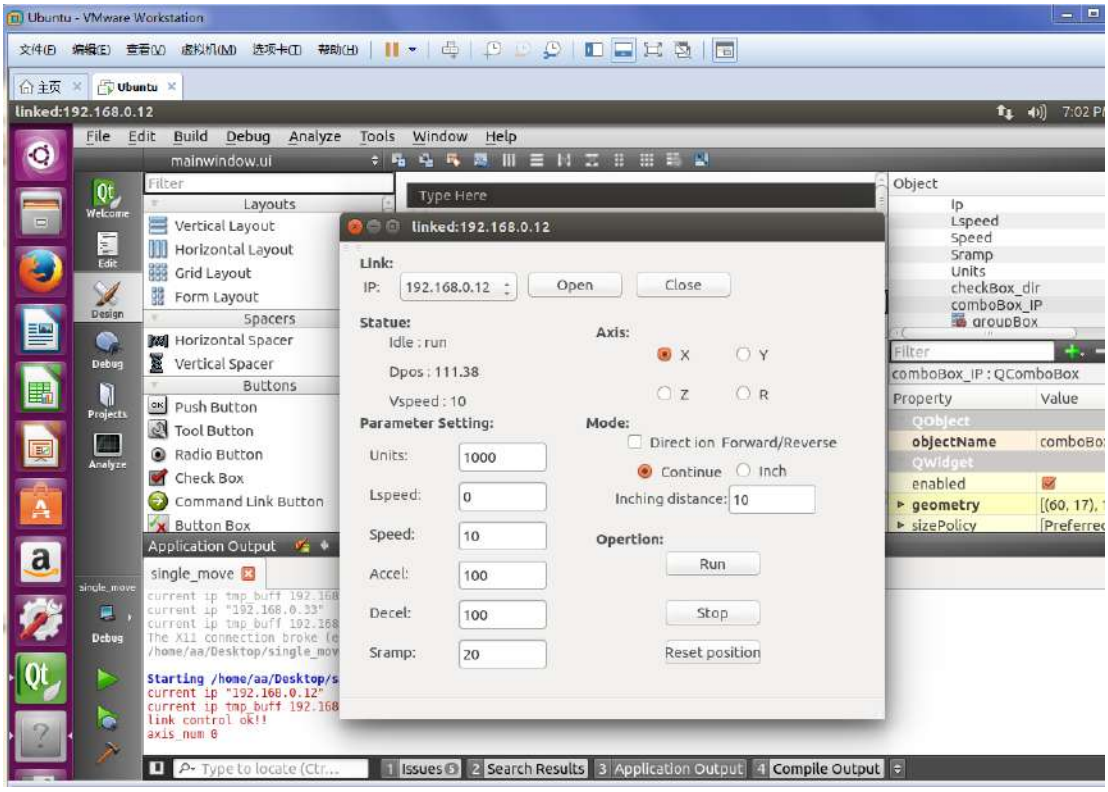




17) 控制器的 ip 网段 192.168.0.1

控制器的默认 ip 是 192.168.0.11

接下来就可以使用例程了，open 连接 ip，点击[run]



2.1.3 Wince 系统

2.1.3-1 C#

本例程用 wince6 移动设备作为外部连接设备，用 USB 作为连接主线。

1) 首先需要 PC 机安装同步设备驱动和 USB 驱动（不做说明）

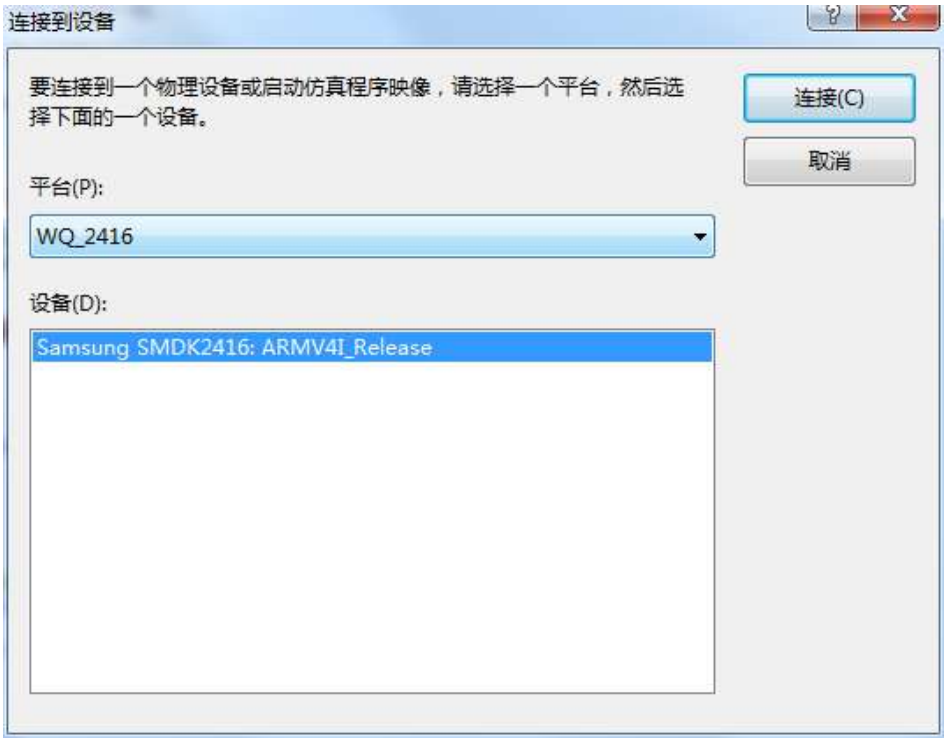
2) 用 USB 线和 PC 机连接，设备会自动与 PC 机连接，连接完成后同步软件会显示连接成功，如图所示



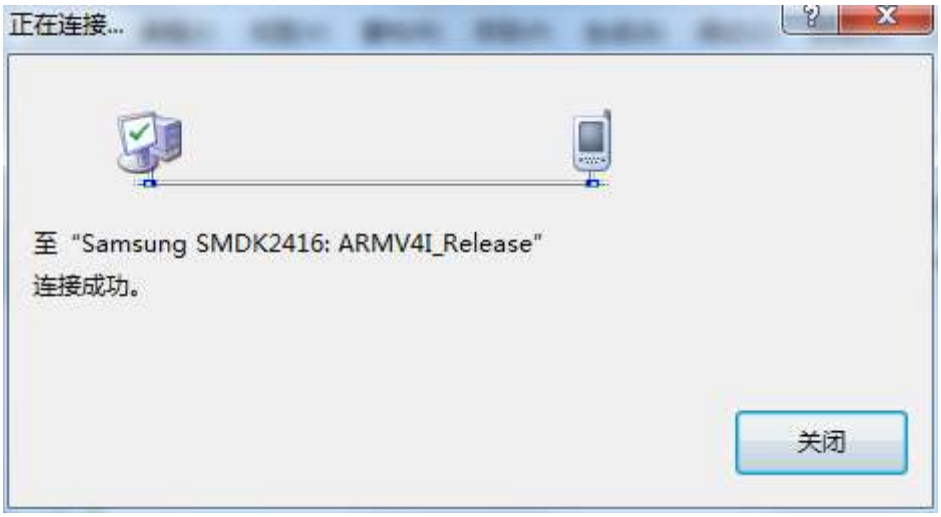
3) 设备同步软件连接完成后，将会与 VS2008 连接步骤如下：

1>点击[工具>>连接到设备]

2>打开如下窗口，同时选择图中平台/设备后点击[连接按钮]



3>连接成功后如下图所示：

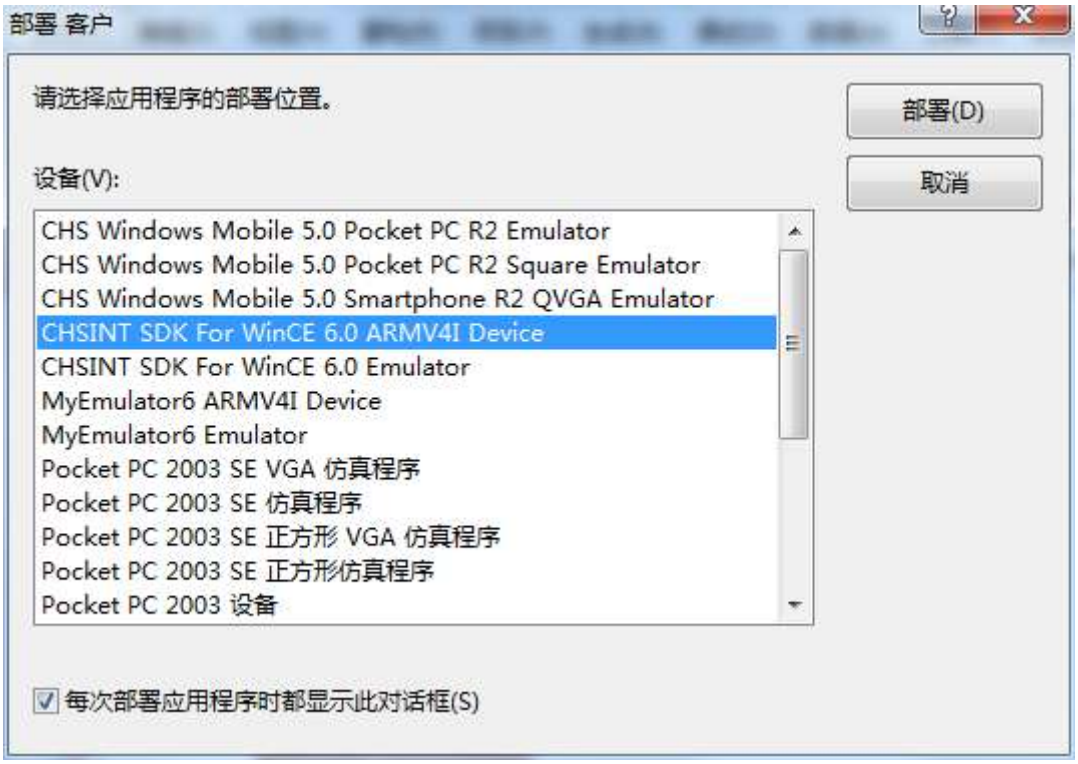


4>点击[关闭]按钮即可

4) 开始调试/运行程序（本说明使用调试）

5) 点击[调试]按钮后如图显示，部署平台

6) 选择图中平台，点击[部署]即可自动部署



7) 部署成功后会在设备上显示出设计窗口，可以开始调试运行



8) 完成窗口代码设计后，通过 USB 连接到 wince6 设备上后，调试运行即可在移动设备上进入此界面

9) 通过 wince6 设备界面可直接操作控制器

（**注：**设备 IP 和控制器 IP 以及 PC 机 IP 应当处于同一网段，且设备 IP 设置时，请注

意不要设置设备网关）

10) 点击[IP 检索]按钮会自动检索 IP 地址，然后点击下拉框按钮，会展现检索到的 IP 地址，需要 1~3 秒时间,然后在下拉框中选中需要连接的 IP 地址

11) 点击[连接]按钮，会自动连接下拉框中指定 IP 的控制器，如果连接失败会产生提示“控制器连接失败，请检查 IP 地址”。（连接成功后会显示“控制器连接成功”）

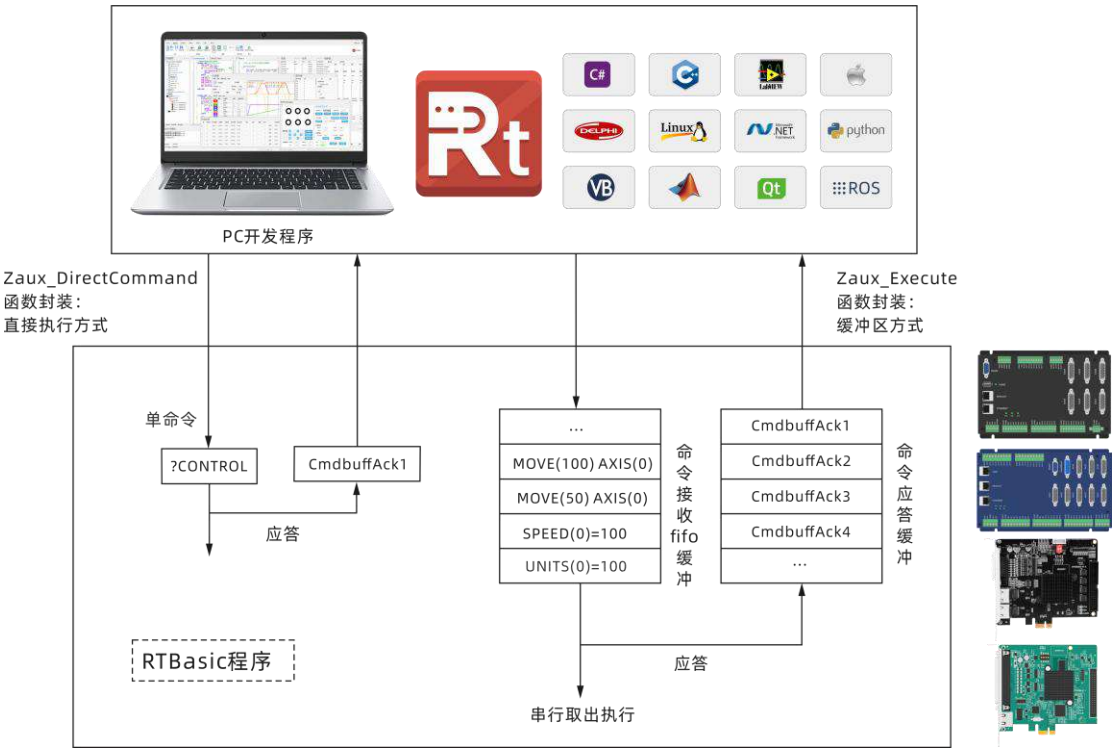
12) 分别点击“X,Y,Z,R 轴”可选定运动轴，分别对应控制器“0,1,2,3 轴”，然后选择运动方式“寸动/连续”，然后点击[运动]按钮，开始运动

13) “运动/停止/位置清零”控件，是整个例程的控制按键，可以控制程序运行、停止等。（当控制器未连接时，这些按键会提示“未连接控制器”）

14) 操作完成后可通过点击[断开]按钮，断开设备和控制器的连接

2.2 辅助库封装

2.2.1 辅助库封装方式



ZAux 库提供了运动和基本参数设置的封装，以开放源代码的方式提供，详细内容参见源码。

ZAux 库是直接利用 RTBASIC 命令通过 ZAux_Execute 方式或 ZAux_DirectCommand 方式发送到控制器上，相应函数可以参考 RTBASIC 手册对应的命令介绍。如果使用到没有封装的命令或者想封装自己的函数，可以通过 ZAux_Execute 或 ZAux_DirectCommand 发送，

或是参照已有代码修改增加相应的函数。

发送字符串命令有两种方式，缓冲方式和直接方式。

直接方式：直接执行单个变量/数组/参数相关命令，此时所有传递的参数必须是具体的数值，不能是表达式；参见 ZAux_DirectCommand 函数。ZAux_DirectCommand(控制器句柄, 命令字符串, 返回字符串, 返回字符长度)。

缓冲方式可以执行所有命令，并支持表达式作为参数，但是速度慢一些；参见 ZAux_Execute 函数。

ZAux_Execute(控制器句柄, 命令字符串, 返回字符串, 返回字符长度)确认函数的输入输出，参数的意义及限制。

例:如设置轴速度函数：ZAux_Direct_SetSpeed(ZMC_HANDLE handle, int iaxis, float fValue)相当与 BASIC 命令中的“SPEED(轴号) = 速度值”。则封装函数时命令字符串就是“speed(%d) =%f”。

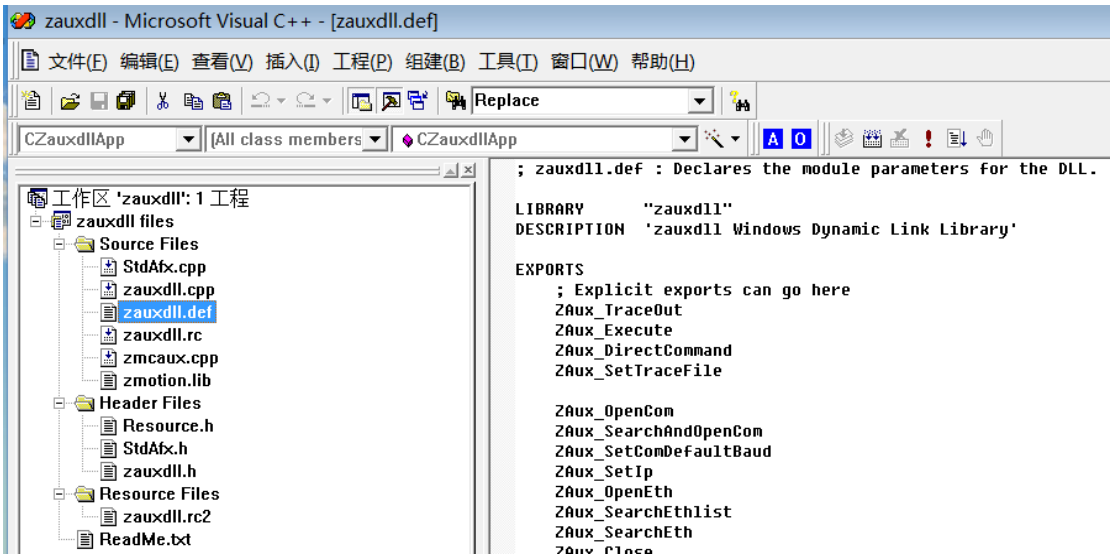
封装步骤：

1) 编写源代码：打开辅助库源码，在 zmcaux.cpp 中添加对应的函数

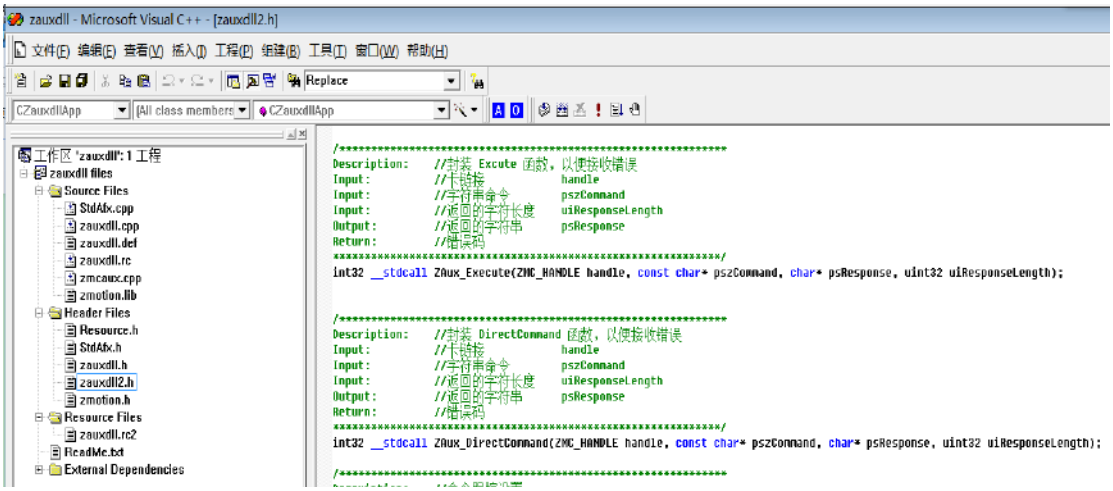
```
int32 __stdcall ZAux_Direct_SetSpeed(ZMC_HANDLE handle, int iaxis, float fValue)
{
    char  cmdbuff[2048];
    char  cmdbuffAck[2048];
    if( iaxis > MAX_AXIS_AUX)
    {
        return  ERR_AUX_PARAERR;
    }
    //生成命令
    sprintf(cmdbuff, "SPEED(%d)=%f", iaxis, fValue);

    //调用命令执行函数
    return ZAux_DirectCommand(handle, cmdbuff, cmdbuffAck, 2048);
}
```

2) 在 zauxdll.def 中定义新增加的函数定义



3) 在头文件中增加对应的声明



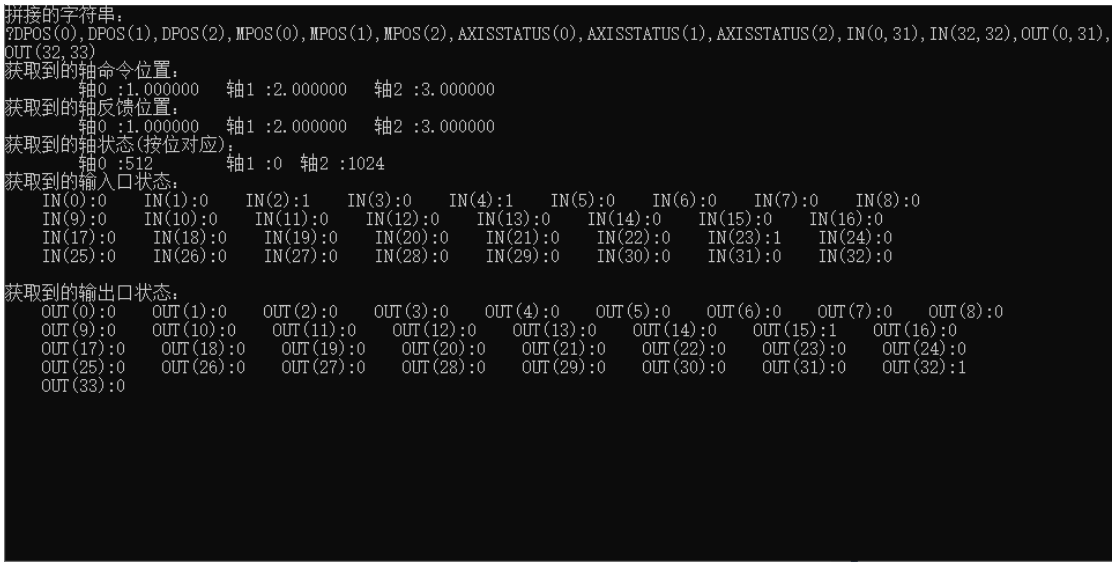
4) 编译生成新的 DLL

2.2.2 实用封装例子

2.2.2-1 直接获取多种类型数据

用户若想要获取多种数据，如轴的命令位置，轴的反馈位置，板卡上的 IO 点等等，往往都是通过多种单独独立的函数获取不同的数据，这样堆积，会导致读写次数的上位，导致程序的卡顿。为了提升一个上位程序的，往往可以通过自己定义一个自己的函数，通过一个函数，快速的把数据传输到上位程序上面来，而非是通过多次循环来获取到不同类型的数据。如下举例：

假设有一个简易的三轴平台，需要读取轴 0，轴 1，轴 2 的命令位置，反馈位置，以及控制器板卡上的输入口 0-输入口 32，输出口 0-输出口 33，以及三个轴的状态。获取数据程序如下：




```
// test1.cpp : 定义控制台应用程序的入口点。
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非0则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
    }
```

```
}

/************************************************************
Description:    //我的自定义直接获取数据函数
Input:          //handle            卡链接
                  iaxisNum          轴的总数量
                  iaxislist             轴号列表
                  fDposlist             输出的命令位置值
                  fMposlist             输出的反馈位置值
                  iAxisstatuslist       输出的轴状态位置值，按位对应
                  startIn               要获取起始的IN编号
                  endIn                 要获取结束的IN编号
                  iIn                   输出的IN状态,按位对应
                  startOut          要获取起始的OUT编号
                  endOut                要获取结束的OUT编号
                  iOut              输出的OUT状态,按位对应
Output:         //
Return:         //错误码
************************************************************/
int   Demo_Direct_MyGetData(ZMC_HANDLE handle,int iaxisNum, int* iaxislist, float* fDposlist,float* fMposlist,int32* iAxisstatuslist,int startIn , int endIn,int *iIn,int startOut , int endOut,int *iOut)
{
    char cmdbuff[2048];
    char  tempbuff[2048];
    char cmdbuffAck[20480];

    //若传进来的地址为空，则退出

    if(NULL == iaxislist || NULL == fDposlist || NULL == fMposlist ||  NULL == iAxisstatuslist || NULL == iIn || NULL == iOut)
    {
        return  ERR_AUX_PARAERR;
    }
    //若传进来的结束编号小于起始编码，则退出
    if ((endIn<startIn)  ||  (endOut<startOut))
    {
        return  ERR_AUX_PARAERR;
    }

    int ret=0;
    int i;
    //生成命令
```

```
    sprintf(cmdbuff, "?");
    //拼接DPOS
    for (i=0;i<iaxisNum;i++)
    {
        sprintf(tempbuff,"DPOS(%d),",iaxislist[i]);//生成对应命令的字符串

        strcat(cmdbuff, tempbuff);//字符串拼接

        if (strlen(cmdbuff)>1000)
        {
            return  ERR_AUX_PARAERR;  //参数错误，字符串拼接过长
        }
    }
    //拼接MPOS
    for (i=0;i<iaxisNum;i++)
    {
        sprintf(tempbuff,"MPOS(%d),",iaxislist[i]);//生成对应命令的字符串

        strcat(cmdbuff, tempbuff);//字符串拼接

        if (strlen(cmdbuff)>1000)
        {
            return  ERR_AUX_PARAERR;  //参数错误，字符串拼接过长
        }
    }
    //拼接AXISSTATUS
    for (i=0;i<iaxisNum;i++)
    {
        sprintf(tempbuff,"AXISSTATUS(%d),",iaxislist[i]);//生成对应命令的字符串

        strcat(cmdbuff, tempbuff);//字符串拼接

        if (strlen(cmdbuff)>1000)
        {
            return  ERR_AUX_PARAERR;  //参数错误，字符串拼接过长
        }
    }
    int32 ostart,istart,iend,oend;      //一次最多32个
    bool addflag;

    addflag=false;

    int32 temp;       //一次最多32个
    int32 temp2;      //一次最多32个
```

```
temp=endIn-startIn+1;
if (temp%32 == 0)
{
    temp=temp/32;
}
else
{
    temp=temp/32+1;
}

//拼接IN
for (i=0;i<temp;i++)
{
    istart = startIn+32*i;
    iend =istart+31;
    if (iend>endIn)
    {
        iend=endIn;
    }
    //生成命令
    sprintf(tempbuff, "IN(%d,%d),", istart,iend);
    strcat(cmdbuff, tempbuff);//字符串拼接
    if (strlen(cmdbuff)>1000)
    {
        return ERR_AUX_PARAERR ;  //参数错误，字符串拼接过长
    }

}

temp2=endOut-startOut+1;
if (temp2%32 == 0)
{
    temp2=temp2/32;
}
else
{
    temp2=temp2/32+1;
}
//拼接OUT
for (i=0;i<temp2;i++)
{
    ostart = startOut+32*i;
    oend =ostart+31;
```

```
        if (oend>endOut)
        {
            oend=endOut;
        }


        //生成命令
        sprintf(tempbuff, "OUT(%d,%d)", ostart,oend);

        strcat(cmdbuff, tempbuff);//字符串拼接
        if (i<temp-1)
        {
            strcat(cmdbuff, ",");//字符串拼接
        }

        if (strlen(cmdbuff)>1000)
        {
            return  ERR_AUX_PARAERR;   //参数错误，字符串拼接过长
        }

    }

    printf("拼接的字符串：\n",cmdbuff);
    printf("%s\n",cmdbuff);

    ret=ZAux_DirectCommand(handle,cmdbuff,cmdbuffAck,2048);
    if(ERR_OK != ret)
    {
        return ret;
    }
    //printf("%s\n",cmdbuffAck);
    //printf("%d\n",strlen(cmdbuffAck));
    //
    if(0 == strlen(cmdbuffAck))
    {
        return ERR_NOACK;
    }
    float ftempbuff[200];
    int itempbuff[200];

    ZAux_TransStringtoFloat(cmdbuffAck,iaxisNum*2,ftempbuff);//字符串转换为浮点数
    //DPOS输出
    for(i=0;i<iaxisNum;i++)
    {
```

```
        //printf("%f\n",ftempbuff[i]);
        fDposlist[i]=ftempbuff[i];

    }
    //MPOS输出
    for(i=0;i<iaxisNum;i++)
    {
        //printf("%f\n",ftempbuff[i+iaxisNum]);
        fMposlist[i]=ftempbuff[i+iaxisNum];
    }

    ZAux_TransStringtoInt(cmdbuffAck,iaxisNum*3+temp2+temp,itempbuff);//字符串转换为
整形
    //AXISSTATUS输出
    for(i=0;i<iaxisNum;i++)
    {
        //printf("%d\n",itempbuff[i+iaxisNum*2]);
        iAxisstatuslist[i]=itempbuff[i+iaxisNum*2];
    }
    //IN输出
    for(i=0;i<temp;i++)
    {
        //printf("%d\n",itempbuff[i+iaxisNum*3]);
        iIn[i]=itempbuff[i+iaxisNum*3];
    }
    //OUT输出
    for(i=0;i<temp2;i++)
    {
        //printf("%d\n",itempbuff[i+iaxisNum*3+temp]);
        iOut[i]=itempbuff[i+iaxisNum*3+temp];
    }
    return ERR_OK;
}
int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";                //控制器IP地址
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
```

```
    }
    printf("控制器连接成功！\n");

    int axis[4]={0,1,2,4};
    float d_dpos[4];
    float d_mpos[4];
    int32 d_axis_status[4];
    int d_in[10];
    int d_out[10];

    ret                                                                    =
Demo_Direct_MyGetData(handle,3,axis,d_dpos,d_mpos,d_axis_status,0,32,d_in,0,33,d_out)
;
    int i;
    printf("获取到的轴命令位置：\n");
    for (i=0;i<3;i++)
    {
        printf("\t轴%d :%f",i,d_dpos[i]);
    }
    printf("\n");
    printf("获取到的轴反馈位置：\n");
    for (i=0;i<3;i++)
    {
        printf("\t轴%d :%f",i,d_mpos[i]);
    }
    printf("\n");
    printf("获取到的轴状态(按位对应)：\n");
    for (i=0;i<3;i++)
    {
        printf("\t轴%d :%d",i,d_axis_status[i]);
    }
    printf("\n");

    printf("获取到的输入口状态：\n");
    int j=0;
    int tempval;
    for (i=0;i<=32;i++)
    {
        if (((i%32)==0)&&(i>0) )
        {
            j++;
        }
        //转换成位
        tempval=d_in[j]>>(i-32*j);
```

```
        printf("    IN(%d):%d",i,tempval &(0x01));
        if (((i%8)==0)&&(i>0) )
        {
            printf("\n");
        }
    }
    printf("\n");

    printf("获取到的输出口状态：\n");
    j=0;

    for (i=0;i<=33;i++)
    {
        if (((i%32)==0)&&(i>0) )
        {
            j++;
        }
        //转换成位
        tempval=d_out[j]>>(i-32*j);
        printf("    OUT(%d):%d",i,tempval &(0x01));
        if (((i%8)==0)&&(i>0) )
        {
            printf("\n");
        }
    }
    printf("\n");

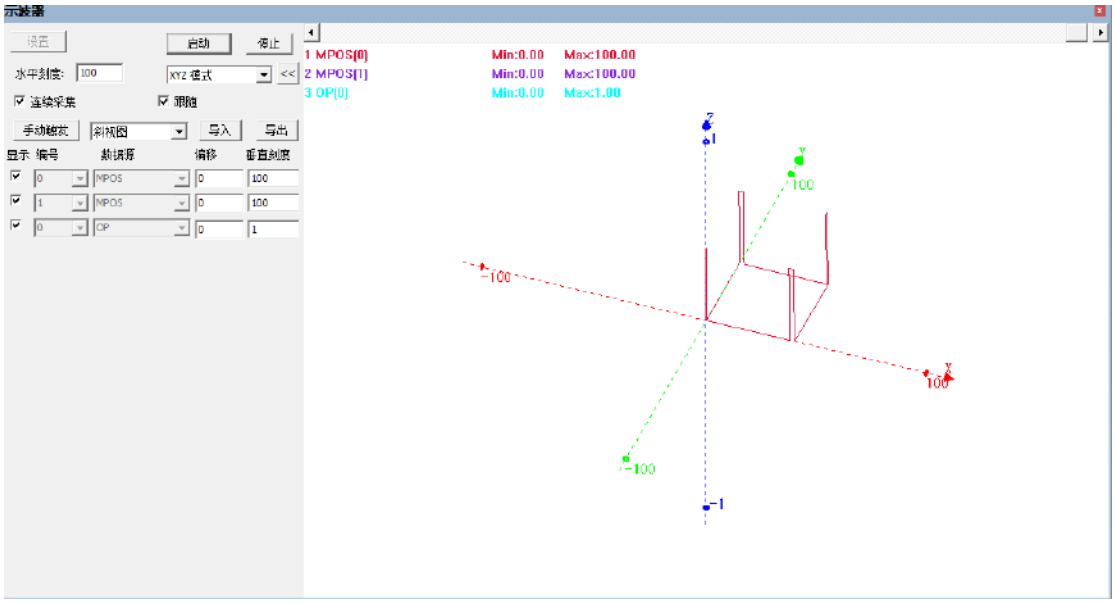
    Sleep(20000);
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");

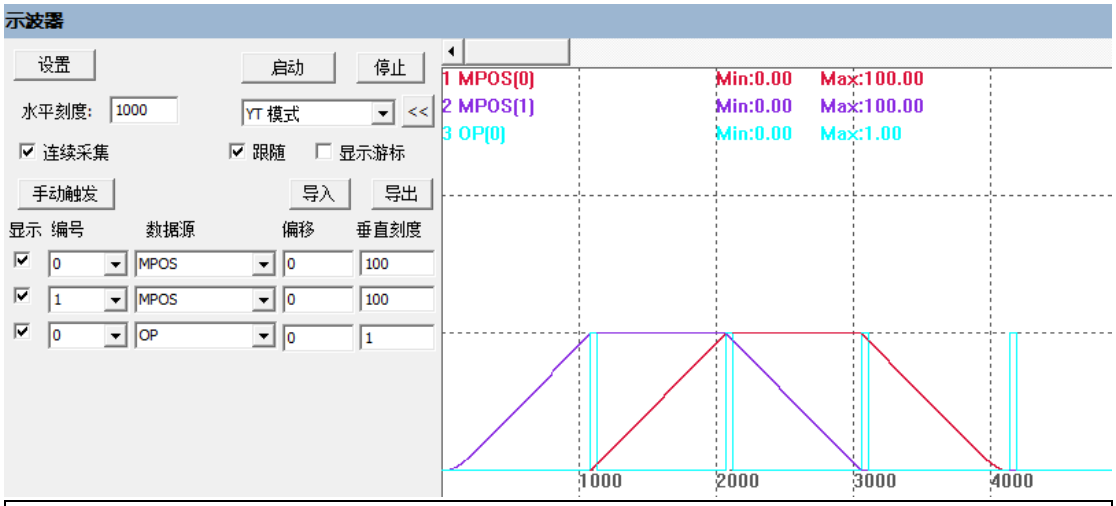
    handle = NULL;
    return 0;
}
```

2.2.2-2 一行命令执行多条不同类型缓冲指令

一般点胶行业，木工行业用的居多，都是用连续轨迹，连续轨迹之间有插入缓冲输出，如果把运动跟连续轨迹的运动跟缓冲输出分开来发的话，难免会有局限性，可以通过自己单独封装运动函数，来达到一行命令执行多个函数的效果，如下举例：

假设控制一个 XY 两轴平台，从坐标点（0,0）—>（100,0）（输出口 0 输出 50ms）—>（100,100）（输出口 0 输出 50ms）—>（0,100）（输出口 0 输出 50ms）—>（0,0）（输出口 0 输出 50ms）的轨迹，则可以通过自己封装，用一条函数，快速发送下去：






```
// test1.cpp : 定义控制台应用程序的入口点。
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非0则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
    }
```

```
}

/*************************************************************
Description:    //我的自定义运动函数
Input:          //handle            卡链接
                  iMoveLen          填写的运动长度
                  iaxisNum          参与运动总轴数
                  iaxislist             轴号列表
                  fPoslist          距离列表
                  iout              缓冲输出口
                  outlist               缓冲输出列表(每条运动，决定是否输出，0为不输出，1为在运动后输出)
                  outtime               缓冲输出时间

Output:         //
Return:         //错误码
*************************************************************/
int  Demo_Direct_MyMoveABS(ZMC_HANDLE handle,int iMoveLen,int iaxisNum, int* iaxislist, float* fPoslist,int iout,int *outlist,int outtime)
{
    char cmdbuff[2048];
    char  tempbuff[2048];
    char cmdbuffAck[20480];

    //若传进来的地址为空，则退出


    int ret=0;
    int i;
    //先读取剩余直线缓冲
    int iBuffLen = 0;
    ret = ZAux_Direct_GetRemain_LineBuffer(handle,iaxislist[0],&iBuffLen);
    if(iBuffLen <= iMoveLen*2)
    {
        return 1002;          //运动缓冲不够
    }

    //生成命令
    sprintf(cmdbuff, "BASE(");
    //拼接运动轴列表
    for (i=0;i<iaxisNum-1;i++)
    {
        sprintf(tempbuff,"%d,",iaxislist[i]);//生成对应命令的字符串
```

```
        strcat(cmdbuff, tempbuff);//字符串拼接

        if (strlen(cmdbuff)>1000)
        {
            return  ERR_AUX_PARAERR;   //参数错误，字符串拼接过长
        }
    }
    sprintf(tempbuff,"%d)\n",iaxislist[i]);//生成对应命令的字符串
    strcat(cmdbuff,tempbuff);

    //拼接运动
    for (i=0;i<iMoveLen;i++)
    {
        //printf("%d,%d\n",i*iaxisNum,i*iaxisNum+1);
        if (outlist[i]==0)     //该段运动不输出
        {
            strcat(cmdbuff, "MoveAbs(");

    sprintf(tempbuff,"%f,%f)\n",fPoslist[i*iaxisNum],fPoslist[i*iaxisNum+1]);//生成
对应命令的字符串
            strcat(cmdbuff, tempbuff);//字符串拼接

        }
        else if (outlist[i]==1)
        {
                strcat(cmdbuff, "MoveAbs(");

    sprintf(tempbuff,"%f,%f)\n",fPoslist[i*iaxisNum],fPoslist[i*iaxisNum+1]);//生成
对应命令的字符串
                strcat(cmdbuff, tempbuff);//字符串拼接

                strcat(cmdbuff, "Move_op2(");
                sprintf(tempbuff,"%d,%d,%d)\n",iout,1,outtime);//生成对应命令的字符
串
                strcat(cmdbuff, tempbuff);//字符串拼接
        }
        else
        {
            return  ERR_AUX_PARAERR;//参数错误
        }


    }
```

```
    printf("拼接的字符串：\n");
    printf("%s\n",cmdbuff);

    if (strlen(cmdbuff)>1000)
    {
        return  ERR_AUX_PARAERR;   //参数错误，字符串拼接过长
    }
    ret=ZAux_DirectCommand(handle,cmdbuff,cmdbuffAck,2048);

    return ret;

}
int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";                //控制器IP地址
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("控制器连接成功！\n");

    ret =ZAux_Direct_SetAtype(handle,0,1);//设置轴0轴类型为1
    commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetAtype(handle,1,1);//设置轴1轴类型为1
    commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetUnits(handle,0,100);//设置轴0脉冲当量为100
    commandCheckHandler("ZAux_Direct_SetUnits", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetUnits(handle,1,100);//设置轴1脉冲当量为100
    commandCheckHandler("ZAux_Direct_SetUnits", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetAccel(handle,0,500);//设置轴0加速度
    commandCheckHandler("ZAux_Direct_SetAccel", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetAccel(handle,1,500);//设置轴1加速度
    commandCheckHandler("ZAux_Direct_SetAccel", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetDecel(handle,0,500);//设置轴0减速度
```

```
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetDecel(handle,1,500);//设置轴1减速度
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetDpos(handle,0,0);//设置轴0 DPOS清0
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetDpos(handle,1,0);//设置轴1 DPOS清0
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetSpeed(handle,0,100);//设置轴0速度
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetSpeed(handle,1,100);//设置轴1速度
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetMerge(handle,0,1);//设置开启连续插补（开启主轴的即可，如轴
0，轴1插补，轴0为主轴，主轴号取决于连续插补运动指令轴列表的第一个轴号）



    int axis[2]={0,1};
    float POS[12]={0,0,0,100,100,100,100,0,0,0};
    int otlist[5]={0,1,1,1,1};

    ZAux_Trigger(handle);//触发示波器

    ret = Demo_Direct_MyMoveABS(handle,5,2,axis,POS,0,otlist,50);//
    commandCheckHandler("Demo_Direct_MyMoveABS", ret) ;//判断指令是否执行成功

    Sleep(20000);
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");

    handle = NULL;
    return 0;
}
```

第三章 指令一览表

注意：本手册中所有蓝色指令(如 ZAux_OpenCom)均带有超链接，点击即可跳转到指定的指令说明。

第四章 板卡连接初始化	
4.1 控制器连接	
ZAux_OpenCom	串口连接控制器
ZAux_SetComDefaultBaud	串口通讯参数设置
ZAux_OpenEth	以太网连接控制器
ZAux_OpenPci	PCI 卡连接
ZAux_Close	关闭控制器连接
ZAux_SetIp	设置控制 IP
ZAux_SearchEthlist	搜索当前网段下控制器的 IP
ZAux_SearchAndOpenCom	快速与控制器建立连接
ZAux_SearchEth	快速检索 IP 列表
ZAux_GetMaxPciCards	读取 PCI 的控制卡个数
ZAux_FastOpen	MotionRT7 连接 与控制器建立连接，指定连接的等待时间
ZAux_GetLocalRTState	获取 RT 状态
ZAux_LocalRTManage	启动停止 RT
4.2 控制器信息获取	
ZAux_GetControllerInfo	获取控制器卡信息
ZAux_GetSysSpecification	获取控制器最大规格数
ZAux_GetRtcTime	读取控制器 RTC 时间
ZAux_SetRtcTime	设置控制器 RTC 时间

第五章 运动控制参数设置	
5.1 运动基本参数	
ZAux_Direct_SetAtype	设置轴类型
ZAux_Direct_GetAtype	读取轴类型
ZAux_Direct_SetUnits	设置脉冲当量（units）
ZAux_Direct_GetUnits	读取脉冲当量（units）
ZAux_Direct_SetAccel	设置加速度，单位为 units /s/s
ZAux_Direct_GetAccel	读取加速度，单位为 units/s/s

ZAux_Direct_SetDecel	设置减速度，单位为 units/s/s
ZAux_Direct_GetDecel	读取减速度，单位为 units/s/s
ZAux_Direct_SetSpeed	设置轴速度，单位为 units/s
ZAux_Direct_GetSpeed	读取轴速度，单位为 units/s
ZAux_Direct_SetDpos	设置轴的规划位置，单位为 units
ZAux_Direct_GetDpos	读取轴的规划位置，单位为 units
ZAux_Direct_SetMpos	设置反馈位置，单位为 units
ZAux_Direct_GetMpos	读取反馈位置，单位为 units
ZAux_Direct_GetEncoder	获取控制器接收的脉冲数
ZAux_Direct_SetFastDec	设置快速减速度，单位为 units/s/s
ZAux_Direct_GetFastDec	读取快速减速度，单位为 units/s/s
ZAux_Direct_SetLspeed	设置轴起始速度，单位为 units/s
ZAux_Direct_GetLspeed	读取轴起始速度，单位为 units/s
ZAux_Direct_SetSramp	设置 S 曲线设置。0-梯形加减速
ZAux_Direct_GetSramp	读取 S 曲线设置。0-梯形加减速
ZAux_Direct_GetMspeed	读取反馈速度，单位为 units/s
ZAux_Direct_GetVpSpeed	读取当前轴运行的命令速度,单位为 units/s
ZAux_Direct_SetVpMode	加减速曲线
ZAux_Direct_GetIfIdle	读取轴是否运动结束
ZAux_Direct_GetAxisStatus	读取轴的告警状态
ZAux_Direct_GetAxisStatus2	读取轴的使能状态
ZAux_Direct_GetAxisStopReason	读取轴历史异常停止原因
5.2 轴其他参数	
ZAux_GetModbusDpos	快速读取多个当前轴的 DPOS（modbus）
ZAux_GetModbusMpos	快速读取多个当前轴的 MPOS（modbus）
ZAux_GetModbusCurSpeed	快速读取多个轴运动时的当前速度（modbus）
ZAux_Direct_SetAxisAddress	设置轴地址
ZAux_Direct_GetAxisAddress	读取轴地址
ZAux_Direct_SetInvertStep	设置脉冲输出模式
ZAux_Direct_GetInvertStep	读取脉冲输出模式
ZAux_Direct_SetMaxSpeed	设置脉冲输出最高频率
ZAux_Direct_GetMaxSpeed	读取脉冲输出最高频率
ZAux_Direct_GetMtype	读取当前正在运动指令类型
ZAux_Direct_GetNtype	读取当前正在运动的下一条运动指令类型
ZAux_Direct_SetOffpos	设置修改偏移相对位置
ZAux_Direct_GetOffpos	读取修改偏移相对位置

ZAux_Direct_Defpos	绝对位置偏移。定义 DPOS，不建议使用
ZAux_Direct_GetFe	读取随动误差
ZAux_Direct_SetFeLimit	设置最大允许的随动误差值(预留)
ZAux_Direct_GetFeLimit	读取最大允许的随动误差值(预留)
ZAux_Direct_SetFeRange	设置报警时随动误差值(预留)
ZAux_Direct_GetFeRange	获取报警时随动误差值(预留)
ZAux_Direct_SetAxisMode	设置 connect 运动保持参数
5.3 轴参数指令	
ZAux_Direct_GetAllAxisPara	浮点型读全部轴参数状态
ZAux_Direct_GetAllAxisInfo	整型读全部轴参数状态
ZAux_Direct_SetParam	通用的轴参数修改函数
ZAux_Direct_GetParam	通用的轴参数读取函数

第六章 基本运动控制	
6.1 运动缓冲机制	
ZAux_Direct_GetEndMove	读取轴当前运动的最终位置
ZAux_Direct_GetEndMoveBuffer	读取当前和缓冲中运动的最终位置
ZAux_Direct_GetLoaded	读取当前除了当前运动是否还有缓冲
ZAux_Direct_GetMovesBuffered	读取当前被缓冲起来的运动个数
ZAux_Direct_GetMoveCurmark	读取当前正在运动指令的 MOVE_MARK 标号
ZAux_Direct_SetMovemark	设置运动指令的 MOVE_MARK 标号
ZAux_Direct_GetRemain	读取返回轴当前运动还未完成的距离
ZAux_Direct_GetRemain_LineBuffer	读取轴剩余的直线缓冲，按直线段来计算
ZAux_Direct_GetRemain_Buffer	参数轴剩余的缓冲，按最复杂的空间圆弧来计算
ZAux_Direct_GetVectorBuffered	读取返回轴当前运动和缓冲运动还未完成的距离
ZAux_Direct_MoveOp	在运动缓冲中加入输出指令
ZAux_Direct_MoveOpAdist	在运动缓冲中设置提前输出距离
ZAux_Direct_MoveOpMulti	在运动缓冲中加入连续输出口输出指令
ZAux_Direct_MoveOpDelay	设置 op 口提前打开或关闭
ZAux_Direct_MoveOp2	在运动缓冲中加入输出指令，指定时间后输出状态翻转
ZAux_Direct_MoveAout	在运动缓冲中加入 AOUT 输出指令
ZAux_Direct_MoveDelay	在运动缓冲中加入延时指令
ZAux_Direct_MoveWait	BASE 轴运动缓冲加入一个可变的延时
ZAux_Direct_MoveTask	BASE 轴运动缓冲加入一个 TASK 任务
ZAux_Direct_MovePara	运动中修改参数

ZAux_Direct_MovePwm	运动中修改 PWM
ZAux_Direct_MoveSynmove	运动中同步其他轴的运动
ZAux_Direct_MoveASynmove	运动中触发其他轴的运动
ZAux_Direct_MoveTable	运动中修改 TABLE
ZAux_Direct_MoveCancel	运动中取消其他轴运动
ZAux_Direct_MoveModify	运动中修改结束位置
ZAux_Direct_MoveLimit	在当前的运动末尾位置增加速度限制，用于强制拐角减速
ZAux_Direct_MoveModify2	运动中修改结束位置 2
ZAux_Execute_GetMoveTime	读取对应运动剩余时间
ZAux_Execute_GetPreCalTime	预估点位运动时间，单位为 ms
6.2 点位运动	
ZAux_Direct_Single_Move	单轴相对运动
ZAux_Direct_Single_MoveAbs	单轴绝对运动
ZAux_Direct_Single_Vmove	单轴连续运动
ZAux_Direct_SetJogSpeed	设置 JOG 时速度
ZAux_Direct_GetJogSpeed	读取 JOG 时速度
ZAux_Direct_GetFastJog	读取 JOG 速度输入 IN，有输入时，速度由 SPEED 参数给出。如果没有输入，速度由 JOGSPEED 参数给出
ZAux_Direct_SetFastJog	设置 JOG 速度输入 IN，有输入时，速度由 SPEED 参数给出。如果没有输入，速度由 JOGSPEED 参数给出
ZAux_Direct_SetFwdJog	设置正向 JOG 输入对应的输入编号-1 无效。
ZAux_Direct_GetFwdJog	读取正向 JOG 输入对应的输入编号-1 无效。
ZAux_Direct_SetRevJog	设置负向 JOG 输入对应的输入编号-1 无效。
ZAux_Direct_GetRevJog	读取负向 JOG 输入对应的输入编号-1 无效。
6.3 回零运动	
ZAux_Direct_Single_Datum	控制器方式回零
ZAux_BusCmd_Datum	总线驱动器回零
ZAux_Direct_UserDatum	自定义二次回零
ZAux_Direct_SetHomeWait	设置回零反找等待时间
ZAux_Direct_GetHomeWait	读取回零反找等待时间
ZAux_BusCmd_SetDatumOffpos	设置回零偏移距离
ZAux_BusCmd_GetDatumOffpos	读取回零偏移距离
ZAux_Direct_SetCreep	设置回零爬行速度

ZAux_Direct_GetCreep	读取回零爬行速度
ZAux_Direct_GetHomeStatus	回零完成状态
ZAux_Direct_GetHomeStatus2	回零完成状态 2
ZAux_BusCmd_GetHomeStatus	驱动器回零完成状态
ZAux_BusCmd_GetHomeStatus2	驱动器回零完成状态 2
ZAux_Direct_SetDatumIn	设置回零原点信号
ZAux_Direct_GetDatumIn	读取回零原点信号
6.4 电子齿轮	
ZAux_Direct_SetClutchRate	设置连接速率
ZAux_Direct_GetClutchRate	获取连接速率
ZAux_Direct_EncoderRatio	编码器输入齿轮比，缺省(1,1)
ZAux_Direct_StepRatio	设置脉冲输出齿轮比，缺省(1,1)
ZAux_Direct_Connect	连接 同步运动指令 电子齿轮
ZAux_Direct_Connpath	连接 同步运动 2 指令 电子齿轮
ZAux_Direct_GetLinkax	读取当前连接运动的参考轴号
6.5 连续插补	
ZAux_Direct_Move	多轴相对直线插补
ZAux_Direct_MoveSp	相对多轴直线插补 SP 运动
ZAux_Direct_MoveAbs	绝对多轴直线插补
ZAux_Direct_MoveAbsSp	绝对多轴直线插补 SP 运动
ZAux_Direct_MoveCirc	相对圆心定圆弧插补运动
ZAux_Direct_MoveCircSp	相对圆心定圆弧插补运动 插补 SP 运动
ZAux_Direct_MoveCircAbs	绝对圆心圆弧插补运动
ZAux_Direct_MoveCircAbsSp	绝对圆心圆弧插补 SP 运动
ZAux_Direct_MoveCirc2	相对 3 点定圆弧插补运动
ZAux_Direct_MoveCirc2Abs	绝对 3 点定圆弧插补运动
ZAux_Direct_MoveCirc2Sp	相对 3 点定圆弧插补 SP 运动
ZAux_Direct_MoveCirc2AbsSp	绝对 3 点定圆弧插补 sp 运动
ZAux_Direct_MHelical	相对 3 轴 圆心画螺旋插补运动
ZAux_Direct_MHelicalAbs	绝对 3 轴 圆心画螺旋插补运动
ZAux_Direct_MHelicalSp	相对 3 轴 圆心画螺旋插补 SP 运动
ZAux_Direct_MHelicalAbsSp	绝对 3 轴 圆心画螺旋插补 SP 运动
ZAux_Direct_MHelical2	相对 3 轴 3 点画螺旋插补运动
ZAux_Direct_MHelical2Abs	绝对 3 轴 3 点画螺旋插补运动
ZAux_Direct_MHelical2Sp	相对 3 轴 3 点画螺旋插补 SP 运动
ZAux_Direct_MHelical2AbsSp	绝对 3 轴 3 点画螺旋插补 SP 运动

ZAux_Direct_MEclipse	相对椭圆插补运动
ZAux_Direct_MEclipseAbs	绝对椭圆插补运动
ZAux_Direct_MEclipseSp	相对椭圆插补 SP 运动
ZAux_Direct_MEclipseAbsSp	绝对椭圆插补 SP 运动
ZAux_Direct_MEclipseHelical	相对椭圆 + 螺旋插补运动
ZAux_Direct_MEclipseHelicalAbs	绝对椭圆 + 螺旋插补运动
ZAux_Direct_MEclipseHelicalSp	相对椭圆 + 螺旋插补 SP 运动
ZAux_Direct_MEclipseHelicalAbsSp	绝对椭圆 + 螺旋插补 SP 运动
ZAux_Direct_MSpherical	空间圆弧 + 螺旋插补运动
ZAux_Direct_MSphericalSp	空间圆弧 + 螺旋插补 SP 运动
ZAux_Direct_MSphericalABSSp	空间圆弧 + 螺旋插补 ABSSP 运动
ZAux_Direct_MoveSpiral	渐开线圆弧插补运动，相对移动方式
ZAux_Direct_MoveSpiralSp	渐开线圆弧插补 SP 运动，相对移动方式
ZAux_Direct_MoveSmooth	空间直线倒圆插补运动
ZAux_Direct_MoveSmoothSp	空间直线倒圆插补 SP 运动
ZAux_Direct_MoveTurnabs	旋转台直线插补运动
ZAux_Direct_McircTurnabs	旋转台圆弧+螺旋插补运动。
ZAux_Direct_MultiMove	多条相对多轴直线插补
ZAux_Direct_MultiMoveAbs	多条绝对多轴直线插补
ZAux_Direct_MultiLineN	多轴多段线直线连续插补
ZAux_Direct_SetFhspeed	设置轴保持速度
ZAux_Direct_GetFhspeed	读取轴保持速度
ZAux_Direct_SetEndMoveSpeed	设置 SP 运动的结束速度
ZAux_Direct_GetEndMoveSpeed	读取 SP 运动结束速度
ZAux_Direct_SetForceSpeed	设置 SP 运动的运行速度
ZAux_Direct_GetForceSpeed	读取 SP 运动的运行速度
ZAux_Direct_SetStartMoveSpeed	设置自定义速度的 SP 运动的起始速度
ZAux_Direct_GetStartMoveSpeed	读取自定义速度的 SP 运动的起始速度
ZAux_Direct_SetFholdIn	设置保持输入对应的输入点编号，-1 无效。
ZAux_Direct_GetFholdIn	读取保持输入对应的输入点编号，-1 无效。
ZAux_Direct_SetCornerMode	设置拐角减速模式
ZAux_Direct_GetCornerMode	读取拐角减速模式
ZAux_Direct_SetDecelAngle	设置拐角开始减速角度,单位为弧度
ZAux_Direct_GetDecelAngle	读取拐角开始减速角度,单位为弧度
ZAux_Direct_SetFullSpRadius	设置小圆限速最小半径
ZAux_Direct_GetFullSpRadius	读取小圆限速最小半径

ZAux_Direct_SetInterpFactor	设置插补时轴是否参与速度计算
ZAux_Direct_GetInterpFactor	读取插补时轴是否参与速度计算
ZAux_Direct_SetMerge	设置连续插补状态
ZAux_Direct_GetMerge	读取连续插补状态
ZAux_Direct_SetStopAngle	设置减速到最低的最小拐角 弧度制
ZAux_Direct_GetStopAngle	读取减速到最低的最小拐角 弧度制
ZAux_Direct_SetZsmooth	设置 减速倒角半径
ZAux_Direct_GetZsmooth	读取 减速倒角半径
6.6 轴暂停/恢复/停止相关	
ZAux_Direct_MovePause	运动暂停，插补运动暂停主轴
ZAux_Direct_MoveResume	取消运动暂停
ZAux_Direct_Rapidstop	所有轴立即停止
ZAux_Direct_CancelAxisList	多个轴运动停止
ZAux_Direct_Single_Cancel	单轴运动停止

第七章 进阶运动控制	
7.1 坐标循环	
ZAux_Direct_SetRepDist	设置循环范围
ZAux_Direct_GetRepDist	读取设置循环范围
ZAux_Direct_SetRepOption	设置坐标循环模式
ZAux_Direct_GetRepOption	读取坐标循环模式
7.2 螺距补偿	
ZAux_Direct_Pitchset	设置轴的螺距补偿，扩展轴无效
ZAux_Direct_Pitchset2	设置轴的螺距双向补偿，扩展轴无效。
ZAux_Direct_GetPitchStatus	设置轴的螺距补偿状态读取，扩展轴无效。
7.3 反向补偿	
ZAux_Direct_Backlash	设置轴的反向间隙补偿，扩展轴无效

第八章 硬件接口访问与配置	
8.1 数字输入输出	
ZAux_Direct_GetIn	读取单个输入信号
ZAux_Direct_SetOp	设置单个输出口信号
ZAux_Direct_GetOp	读取单个输出口状态
ZAux_GetModbusIn	快速读取多个输入(Modbus)
ZAux_GetModbusOut	快速读取多个当前的输出状态(Modbus)

ZAux_Direct_GetInMulti	读取多个输入信号
ZAux_Direct_SetOutMulti	IO 接口 设置多路输出
ZAux_Direct_GetOutMulti	IO 接口 获取多路输出状态
ZAux_Direct_SetInvertIn	设置输入口有效状态反转
ZAux_Direct_GetInvertIn	读取输入口反转状态
8.2 模拟量输入输出	
ZAux_Direct_GetAD	读取单个模拟量输入信号
ZAux_Direct_SetDA	打开单个模拟量输出信号
ZAux_Direct_GetDA	读取模拟输出口状态
8.3PWM 控制	
ZAux_Direct_SetPwmFreq	设置 pwm 频率
ZAux_Direct_GetPwmFreq	读取 pwm 频率
ZAux_Direct_SetPwmDuty	设置 pwm 占空比
ZAux_Direct_GetPwmDuty	读取 pwm 占空比

第九章 安全机制	
9.1 轴软件限位函数	
ZAux_Direct_SetFsLimit	设置正向软限位
ZAux_Direct_GetFsLimit	读取正向软限位
ZAux_Direct_SetRsLimit	设置负向软限位位置
ZAux_Direct_GetRsLimit	读取负向软限位位置
9.2 轴硬件限位函数	
ZAux_Direct_SetFwdIn	设置正向硬限位开关对应的输入点编号
ZAux_Direct_GetFwdIn	读取正向硬限位开关对应的输入点编号
ZAux_Direct_SetRevIn	设置负向硬件限位开关对应的输入点编号
ZAux_Direct_GetRevIn	读取负向硬件限位开关对应的输入点编号
9.3 轴报警函数	
ZAux_Direct_SetAlmIn	设置轴告警开关对应的输入点编号
ZAux_Direct_GetAlmIn	读取告警开关对应的输入点编号

第十章 板卡数据交互	
10.1 VR 寄存器	
ZAux_Direct_SetVrf	设置 VR 寄存器
ZAux_Direct_GetVrf	读取 VR 寄存器
ZAux_Direct_GetVrInt	整型方式读取 VR 寄存器

10.2 Table 寄存器	
ZAux_Direct_SetTable	设置系统 table 寄存器
ZAux_Direct_GetTable	读取系统 table 寄存器
ZAux_Direct_SetTableDouble	写入 Table
ZAux_Direct_GetTableDouble	读取 Table，可以一次读取多个
10.3 Modbus 寄存器	
ZAux_Modbus_Set0x	设置 modbus 位寄存器
ZAux_Modbus_Get0x	读取 modbus 位寄存器
ZAux_Modbus_Set4x	设置 modbus 寄存器（REG）
ZAux_Modbus_Get4x	读取 modbus 寄存器（REG）
ZAux_Modbus_Set4x_Long	Modbus4x 寄存器写操作(LONG)
ZAux_Modbus_Get4x_Long	Modbus4x 寄存器读操作(LONG)
ZAux_Modbus_Set4x_String	Modbus 字寄存器操作 （ASCII）
ZAux_Modbus_Get4x_String	Modbus 字寄存器操作 （ASCII）
ZAux_Modbus_Set4x_Float	设置 modbus 寄存器(float 类型)
ZAux_Modbus_Get4x_Float	读取 modbus 寄存器(float 类型)
10.4 Flash/文件读写	
ZAux_WriteUFile	向 U 盘中写文件
ZAux_ReadUFile	从 U 盘中读取文件
ZAux_FlashWritef	写控制器 FLASH 空间
ZAux_FlashReadf	读控制器 FLASH 空间
ZAux_DownMem3FileRam	打包字符串运动下发
ZAux_Run3FileRam	运行下发的 3 次文件
ZAux_Down3FileRam	下载 3 次文件到 RAM
ZAux_Down3FileRom	下载 3 次文件到 ROM
ZAux_3FileRamGetRemainSpace	动态下载 3 次文件，读取可写的空间
ZAux_3FileRamDownBegin	动态下载 3 次文件，启动下载
ZAux_3FileRamDownPart	动态下载 3 次文件
ZAux_3FileRamDownEnd	动态下载 3 次文件，结束下载
ZAux_Check3File	检测三次文件是否存在
ZAux_Delete3File	删除控制器上的文件
ZAux_FindFirst3File	查找控制器上的文件，文件名为空表示文件不不存在
ZAux_FindNext3File	查找控制器上的下个文件，文件名为空表示文件不不存在
ZAux_GetCur3File	获取控制器上的当前文件

ZAux_RemoveAll3Files	删除控制器上所有的三次的文件
ZAux_Up3File	上传三次文件
ZAux_Up3FileToMem	上传三次文件

第十一章 总线初始化	
11.1 总线初始化	
ZAux_BusCmd_GetNodeNum	读取卡槽上节点数量
ZAux_BusCmd_GetNodeInfo	读取节点上的信息
ZAux_BusCmd_GetNodeStatus	读取节点总线状态
ZAux_BusCmd_DriveClear	设置清除总线伺服报警
ZAux_Direct_SetErrormask	设置错误标记
ZAux_Direct_GetErrormask	读取错误标记
ZAux_Direct_SetAxisEnable	设置轴使能，只针对总线控制器轴使用有效
ZAux_Direct_GetAxisEnable	读取轴使能，只针对总线控制器轴使用有效
ZAux_Direct_GetBusAxisEnable	读取总线轴使能状态
11.2 SDO/PDO 读写	
ZAux_BusCmd_SDOWrite	通过设备号和槽位号进行 SDO 写入
ZAux_BusCmd_SDORead	通过设备号和槽位号进行 SDO 读取
ZAux_BusCmd_SDOWrite2	写入节点 SDO 参数信息
ZAux_BusCmd_SDORead2	读取节点 SDO 参数信息
ZAux_BusCmd_RtexRead	Rtex 读取参数信息
ZAux_BusCmd_RtexWrite	Rtex 写参数信息
ZAux_BusCmd_SDOReadAxis	读轴 SDO 参数信息
ZAux_BusCmd_SDOWriteAxis	写轴 SDO 参数信息
ZAux_BusCmd_NodePdoWrite	Pdo 写操作
ZAux_BusCmd_NodePdoRead	Pdo 读操作
ZAux_BusCmd_NodePdoWriteFloat	Pdo 写操作
ZAux_BusCmd_NodePdoReadFloat	Pdo 读操作
11.3 总线转矩力矩	
ZAux_BusCmd_GetDriveTorque	读取当前总线驱动当前力矩 需要设置对应的 DRIVE_PROFILE 类型
ZAux_BusCmd_SetMaxDriveTorque	设置当前总线驱动最大转矩 需要设置对应的 DRIVE_PROFILE 类型
ZAux_BusCmd_GetMaxDriveTorque	读取当前总线驱动最大转矩 需要设置对应的 DRIVE_PROFILE 类型
ZAux_Direct_SetDAC	设置模拟量输出 力矩、速度模式下可以 总线驱

	动需要设置对应 DRIVE_PROFILE 类型与 ATYPE
ZAux_Direct_GetDAC	读取模拟量输出 力矩、速度模式下可以 总线驱动需要设置对应 DRIVE_PROFILE 类型与 ATYPE

第十二章 其他指令	
12.1 在线命令	
ZAux_Execute	发送字符串（RTBasic）命令(缓冲方式)
ZAux_DirectCommand	发送字符串（RTBasic）命令(直接方式)
12.2 控制器 Basic 相关	
ZAux_Pause	暂停控制器内部运行的 BASIC 程序。
ZAux_Resume	恢复控制器内部运行的暂停的 BASIC 程序。
ZAux_BasDown	单 BAS 文件生成 ZAR 并且下载到控制器运行
ZAux_ZarDown	下载 ZAR 文件到控制器
ZAux_ZpjDown	下载 Zpj 工程文件到控制器
ZAux_Direct_GetVariablef	全局变量读取，也可以是轴参数等等，返回值为浮点型
ZAux_Direct_GetVariableInt	全局变量读取，也可以是轴参数等等，返回值为整型
ZAux_Direct_SetUserArray	设置 BASIC 自定义全局数组
ZAux_Direct_GetUserArray	读取设置 BASIC 自定义全局数组 可以一次读取多个
ZAux_Direct_SetUserVar	设置自定义全局变量
ZAux_Direct_GetUserVar	读取自定义全局变量
ZAux_SetTimeOut	命令的延时等待时间
ZAux_TransStringtoFloat	字符串转为 float
ZAux_TransStringtoInt	字符串转为 int
ZAux_TransStringtoDouble	字符串转为 double
12.3 控制器自动上报	
(*PZMCAutoUpCallBack)	主动上报的回调函数格式
ZAux_SetAutoUpCallBack	控制器自动上报
ZAux_CycleUpEnable	使能周期上报
ZAux_CycleUpDisable	去使能周期上报
ZAux_CycleUpGetRecvTimes	周期上报收到的包次数，超过溢出，调试使用.
ZAux_CycleUpForceOnce	强制上报一次,在运动指令后 idle 可能不准确的情

	况下调用
ZAux_CycleUpReadBuff	从周期上报里面读取内容（浮点型）
ZAux_CycleUpReadBuffInt	从周期上报里面读取内容（整型）
12.4 调试相关	
ZAux_SetTraceFile	命令跟踪设置
ZAux_Trigger	示波器触发函数
ZAux_Scope	示波器采样
ZAux_GetScopeLength	获取示波器采样长度

第四章 板卡连接初始化

4.1 控制器连接

4.1.1 指令列表

指令	说明
ZAux_OpenCom	串口连接控制器
ZAux_SetComDefaultBaud	串口通讯参数设置
ZAux_OpenEth	以太网连接控制器
ZAux_OpenPci	PCI 卡连接
ZAux_Close	关闭控制器连接
ZAux_SetIp	设置控制 IP
ZAux_SearchEthlist	搜索当前网段下控制器的 IP
ZAux_SearchAndOpenCom	快速与控制器建立连接
ZAux_SearchEth	快速检索 IP 列表
ZAux_GetMaxPciCards	读取 PCI 的控制卡个数
ZAux_FastOpen	MotionRT7 连接 与控制器建立连接，指定连接的等待时间
ZAux_GetLocalRTState	获取 RT 状态
ZAux_LocalRTManage	启动停止 RT
注意：使用 ZAux_FastOpen 或 ZAux_OpenCom 串口通讯连接控制器时，如果控制器的波特率不是默认的 38400，则在连接前就需要使用 ZAux_SetComDefaultBaud 将停止位设为 0，其他参数保持一致，才可以与控制器通讯上。	

4.1.2 重点说明

4.1.2-1 常规说明

串口



缺省情况下，控制器串口默认的参数如下：

波特率	数据位	停止位	校验位
38400	8	1	无

注意：电脑上面的串口参数需要和控制器上面的串口参数保持一致才能连接上。



站号：串口连接使用 485 的情况，可能会有多个从站，通过站号来选择,一般情况下填 0 就行。

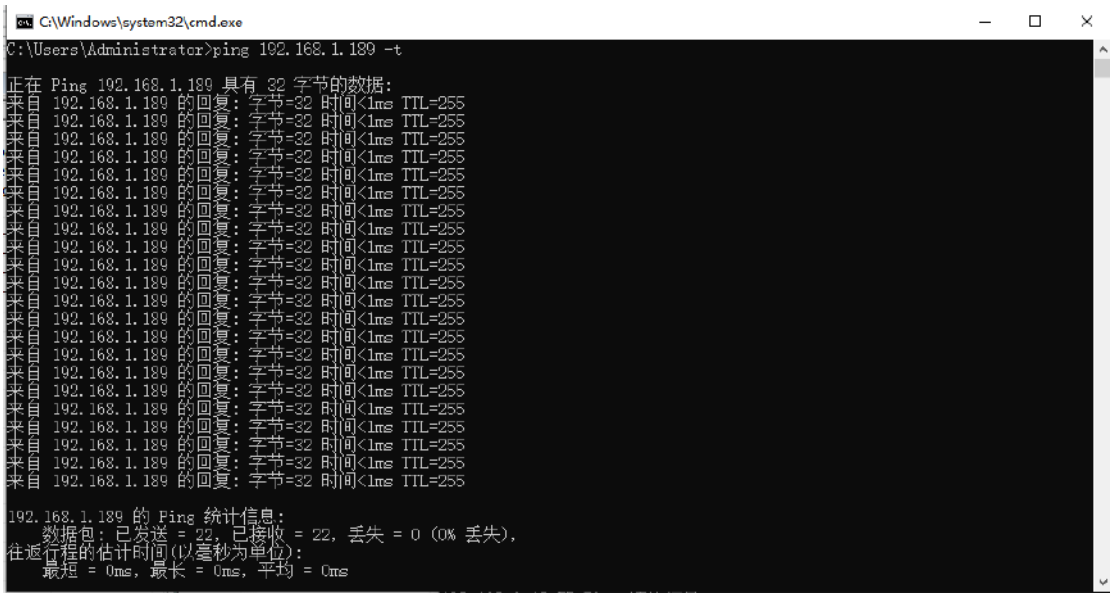
Ethernet 网口



网口采用 RJ45 标准网线接口，通讯速率为 100Mbit/s。控制器自带一个网口，但本身至少是支持 2 个网口通道的，可采用交换机扩展网口，?*port 打印通讯通道。

控制器出厂的 IP 地址为 192.168.0.11，端口号为 502，通讯时保证设备处于同一网段（前三段相同），IP 地址支持修改，网口常用于连接电脑或触摸屏，电脑连接控制器一般使用网口，如果忘记了控制器的 IP，可以使用串口连接上去查看或是修改。

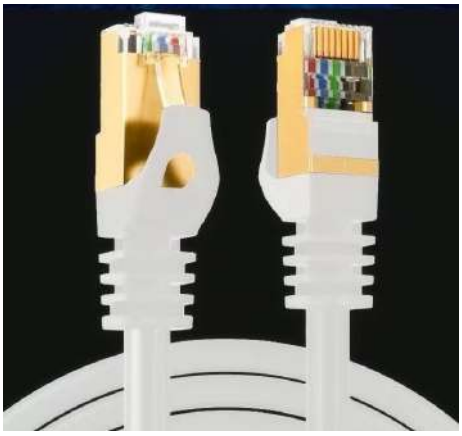
网口连接的能力与性能：



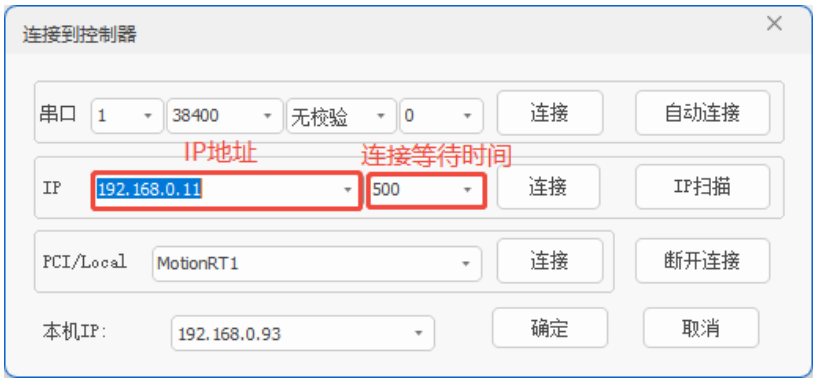
网线推荐

网线类型	传输频率	传输速率	传输距离	线缆类型
五类线	100Mhz	100Mbps	100 米	屏蔽/非屏蔽
超五类线	100Mhz	1000Mbps	100 米	屏蔽/非屏蔽
六类线	250Mhz	1Gbps	100 米	屏蔽/非屏蔽
超六类线	500Mhz	10Gbps	100 米	屏蔽/非屏蔽
七类线	600Mhz	10Gbps	100 米	双屏蔽
八类线	2000Mhz	40Gbps	30 米	双屏蔽

为保证通讯质量，在干扰较大的场合，最好采用七类以上的金属头双屏蔽层的网线，正常场合，使用超五类带屏蔽金属头的网线即可。如右图：网线带金属线的，可以起到接地保护作用。如果不采用金属头带屏蔽层的网线，在实际有干扰的现场，会出现丢包等不确定的事情。



检测函数返回值判断函数是否执行成功，返回 0 时则成功，非 0 失败。以太网口连接返回不成功时，检查传递的 IP 是否正确，并且保证电脑 IP 地址和控制器 IP 地址在同一个网段。

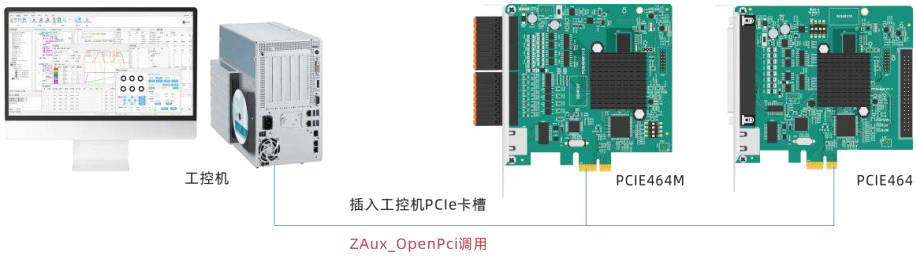




连接等待时间：网口连接有效。响应时间较短的话，可适当增大该值，延长响应时间。

起始 IP，结束 IP：根据自己设置的控制 IP 的网段填写对应的网段。

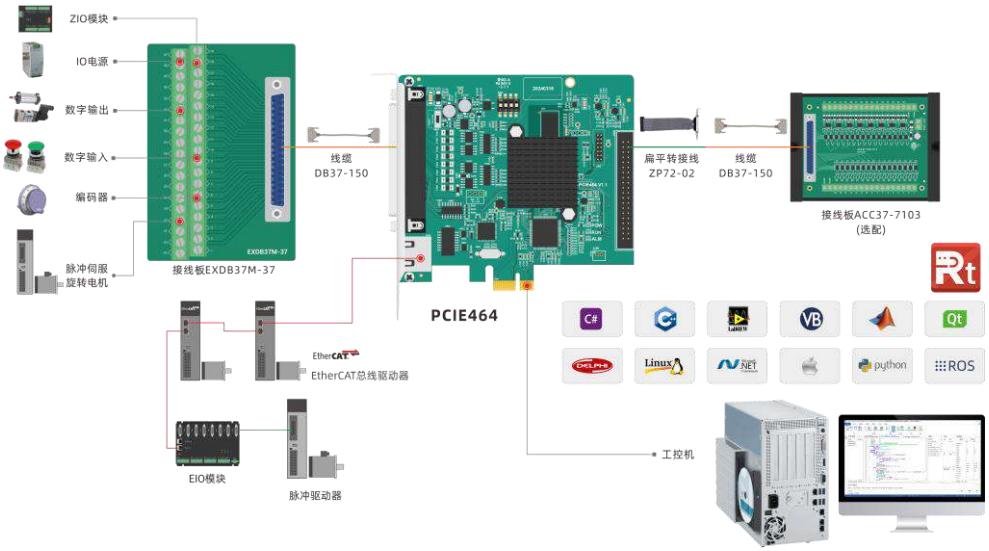
PCI 接口



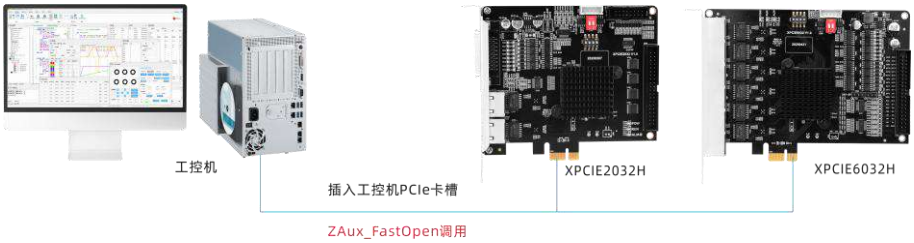


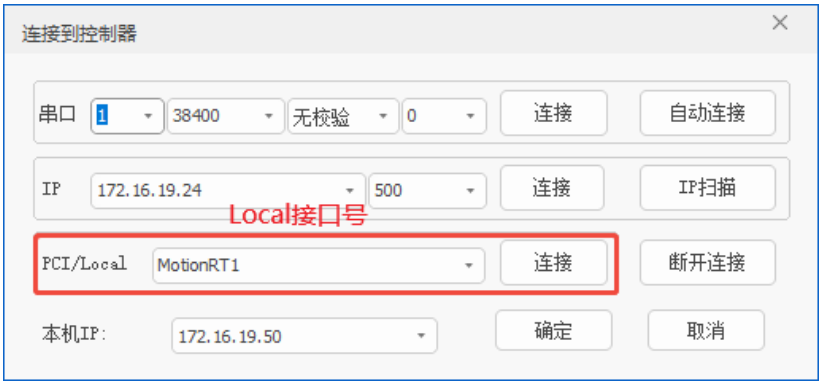
PCI 卡卡号：PCI 类型的控制卡插槽卡号(需安装驱动方可扫出)。

适用于 PCI、PCIe 系列控制卡，如 PCIe464，PCIe464M 等。

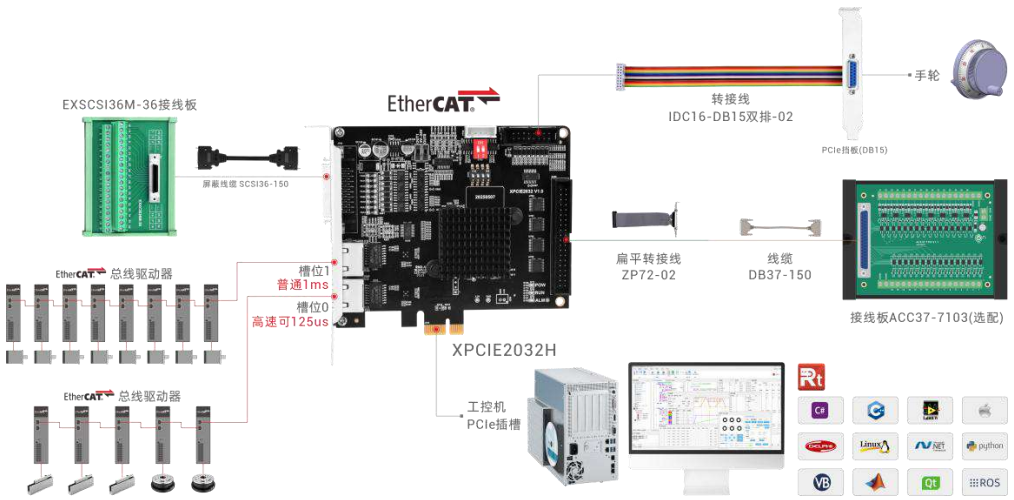


Local 接口





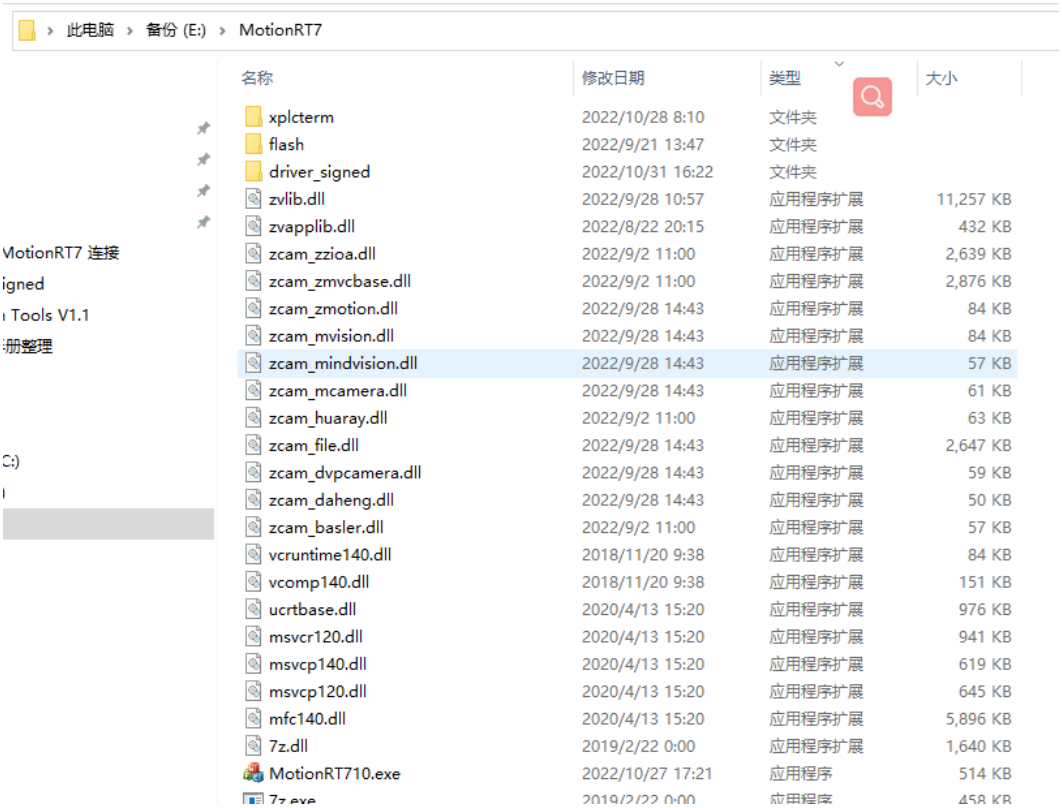
适用于 VPLC7 系列、XPCIe 系列，如 XPCIe2032H；并且需配合 MotionRT7 实时软核使用。



4.1.2-2 MotionRT7

MotionRT7 建议配合 XPCI/XPCIE 控制卡使用，能更大发挥性能，授权信息存储在控制卡上，每个控制卡具有唯一编号。

安装前准备好最新版本的 MotionRT7 安装包并解压。（MotionRT7 软件安装包获取路径可前往正运动技术官网自行下载：http://www.zmotion.com.cn/），下图为 MotionRT7 安装包解压后的文件内容。



1. Driver_signed：驱动文件夹，里面包含了 MotionRT7 驱动安装信息文件、ECAT 协议安装信息、安装向导软件，可能还会有安全目录文件、sys 系统文件和签名文件。



Dpinst_amd64.exe：安装向导软件；

ZMotionRT64.cat：驱动程序的数字签名文件；

ZMotionRT64.inf：MotionRT7 软件驱动安装信息，安装驱动时浏览文件夹选择该文件；

ZMotionRT64.sys：系统文件；

ZMotionRTPacket.inf：总线协议安装信息，安装 EACT 总线协议浏览文件夹选择该文件；

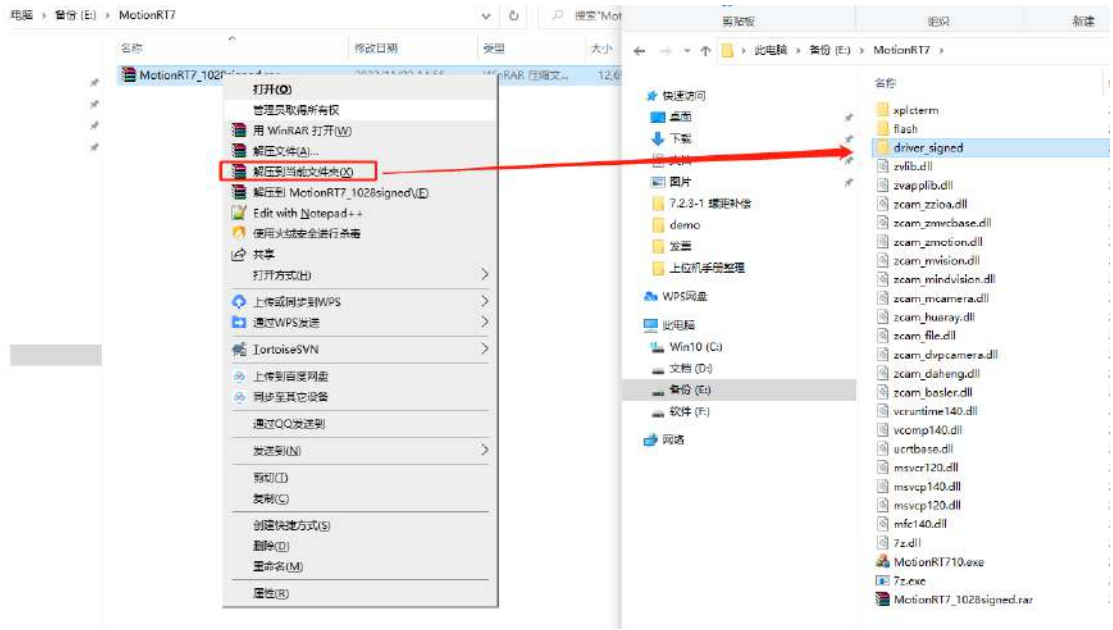
2. flash：控制器系统文件夹。

3. Udisk：U 盘文件夹。

4. Xplcterm：xplc 屏幕文件夹，内包含 xplcterm 软件，在使用 HMI 时作为屏幕显示。

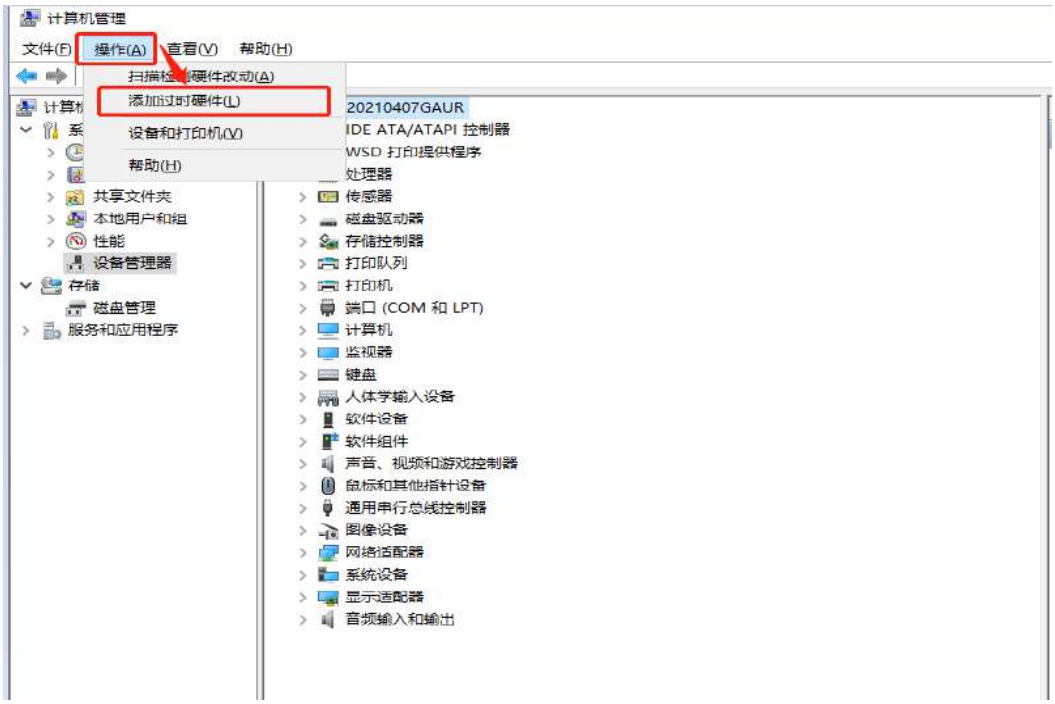
MotionRT7 驱动安装

获取到 MotionRT7 的安装包之后，自行把安装包解压到自己选择的位置，如下图：



[无卡操作]

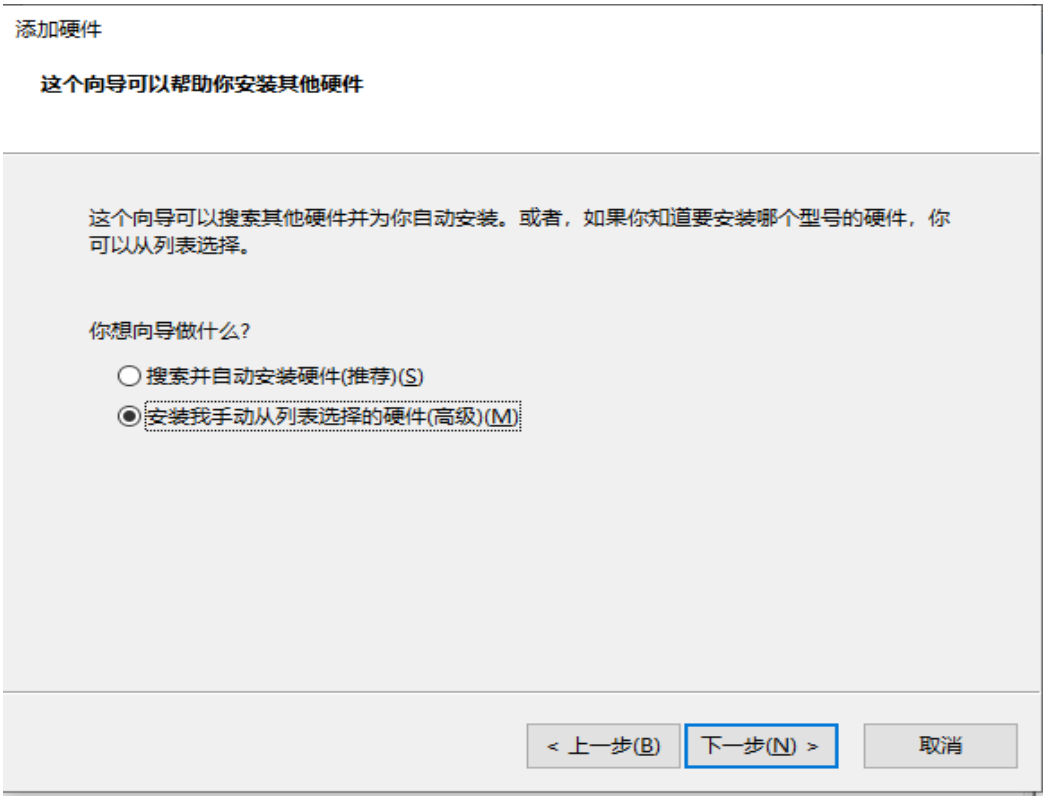
1. 无 XPCI/XPCIE 卡设备时，在设备管理器中，菜单：“操作”-“添加过时硬件”，如下图。若没有“操作”，就鼠标右击，找到“添加过时硬件”：



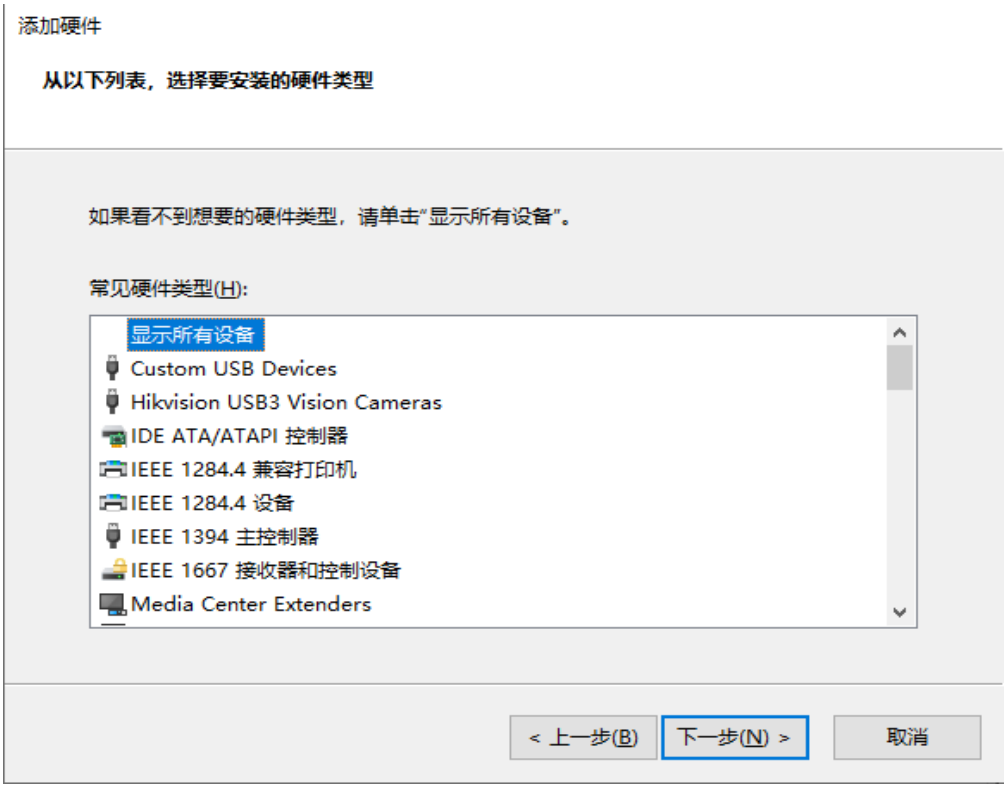
2. 点击“添加过时硬件”后，弹出添加硬件向导界面，单击“下一步”。



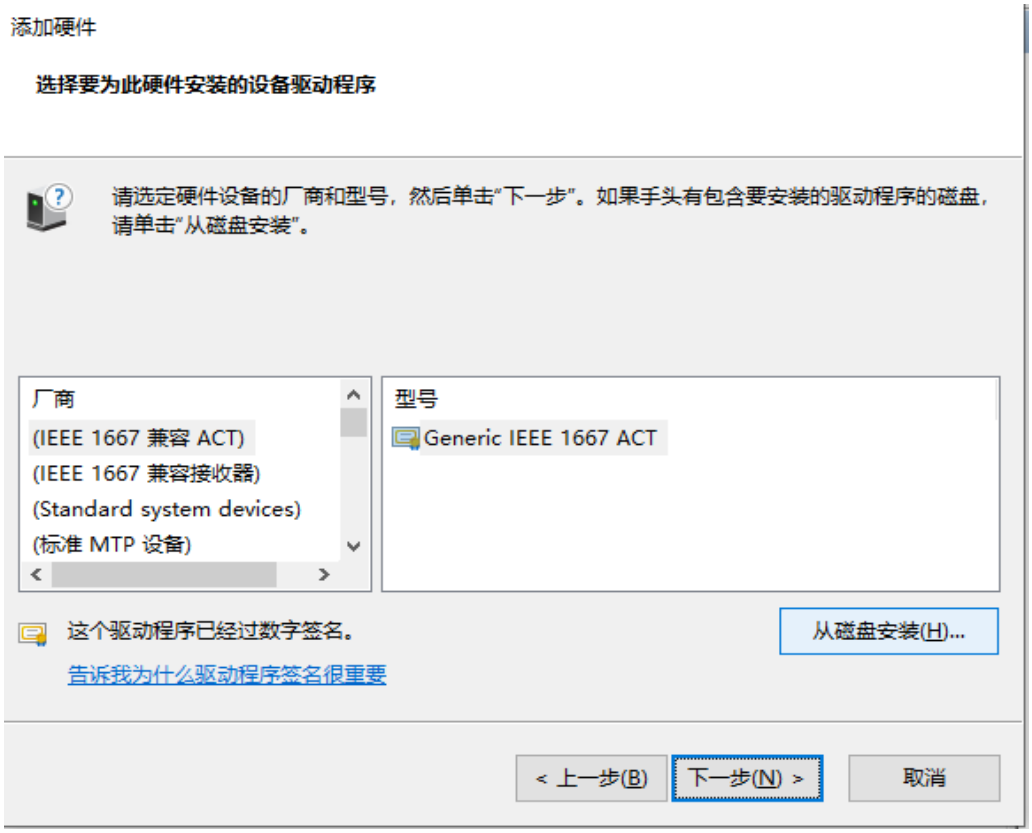
3. 继第 2 步操作，单击“下一步”之后，弹出以下窗口，选择“安装我手动从列表选择的硬件”后单击下一步。



4. 继第 3 步操作，单击“下一步”之后，弹出以下窗口，常见硬件类型选择“显示所有设备”后单击下一步。



5. 继第 4 步操作，单击“下一步”之后，弹出以下窗口，直接点击“从磁盘安装”。



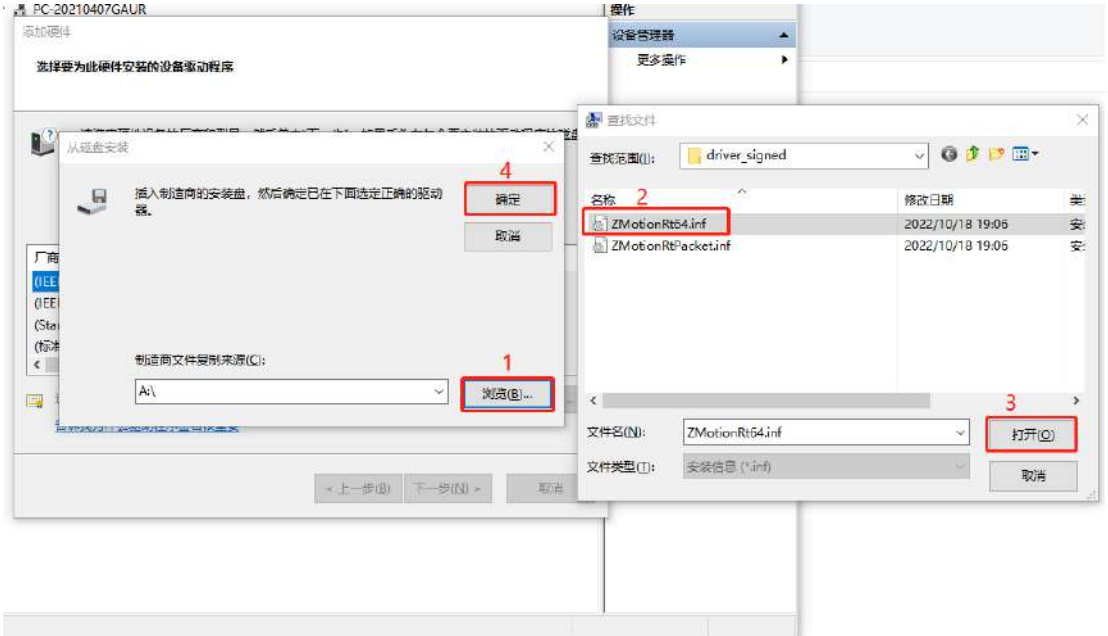
6. 继第 5 步操作，点击“从磁盘安装”之后，弹出以下窗口：

（1）点击“浏览”；

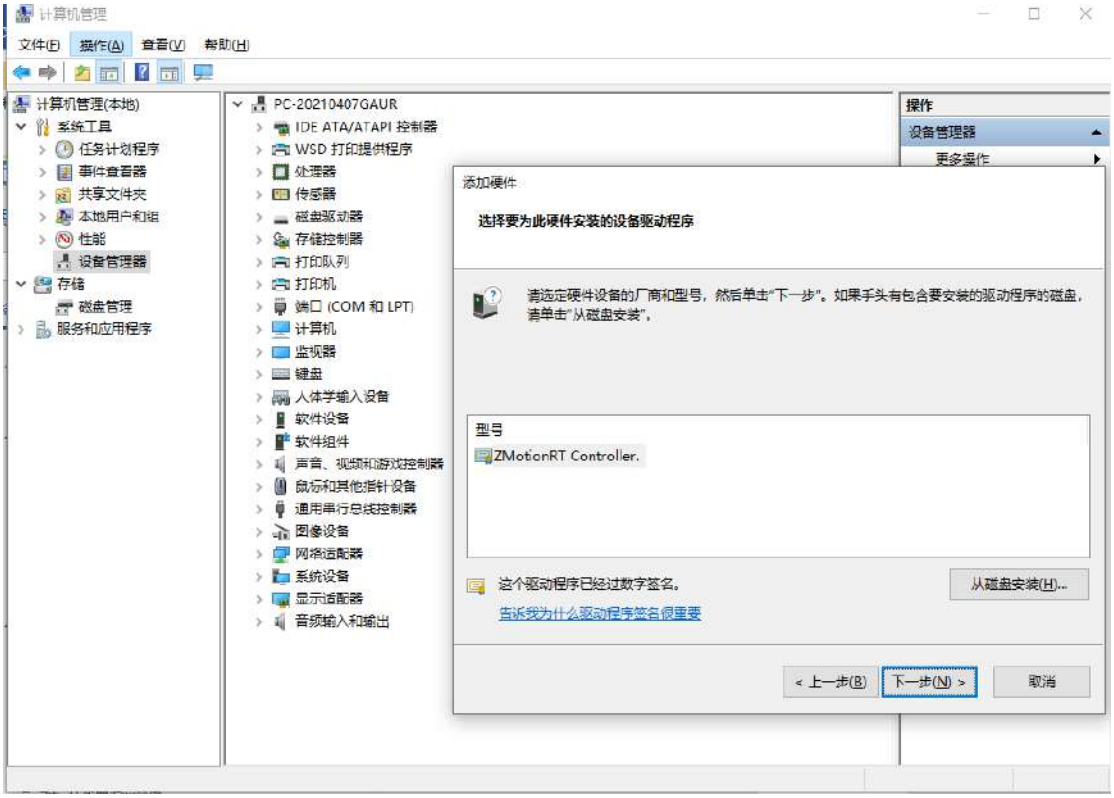
（2）打开 MotionRT7 压缩包下面的：MotionRT7\driver_signed\ZMotionRt64.inf 文件路径；

（3）打开 ZMotionRt64.inf 文件；

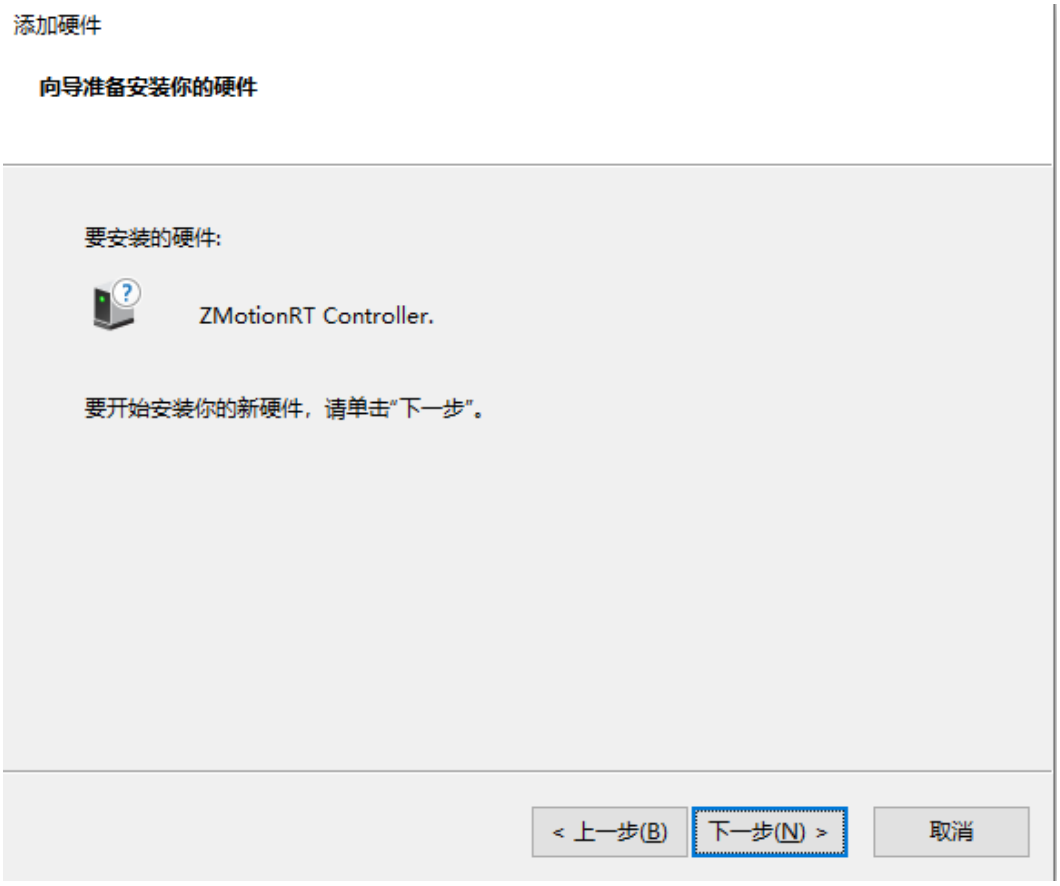
（4）点击确定。



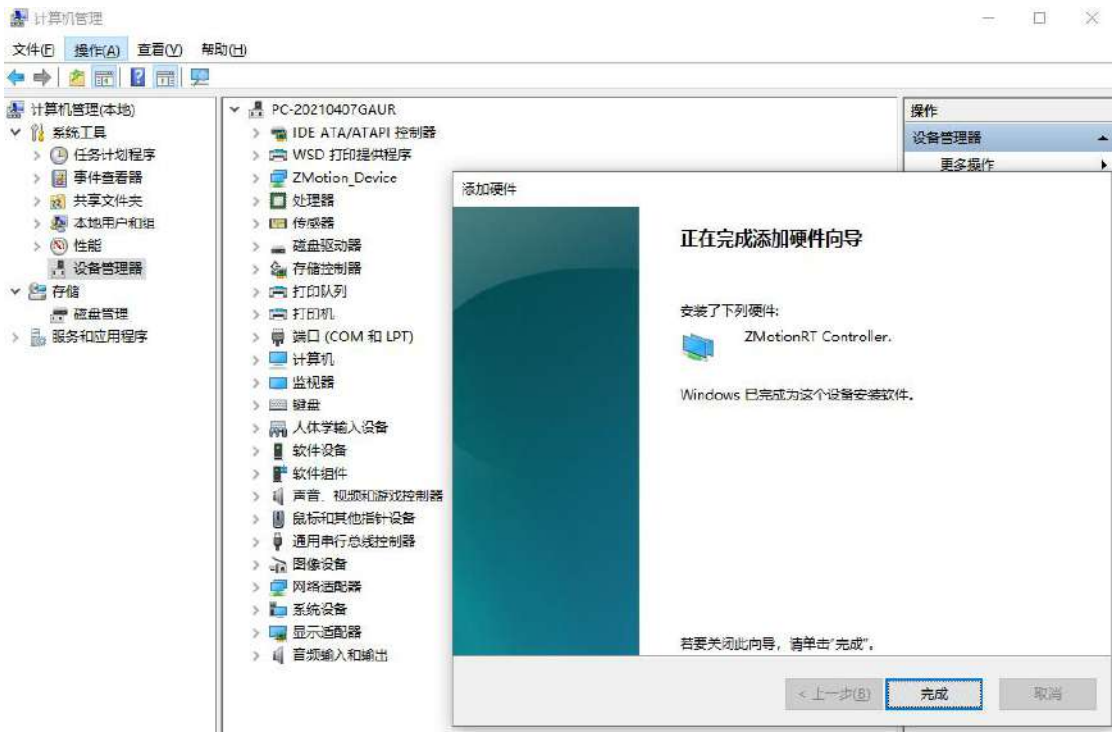
7. 继第 6 步操作，点击“确定”之后，弹出以下窗口，直接点击“下一步”；



8. 继第 7 步操作，点击“下一步”之后，弹出以下窗口，直接点击“下一步”；



9. 最后一步，步骤 8 点击“下一步”之后，弹出以下窗口，直接点击 “完成”，即可。设备管理器也会多出 ZMotion_Device 的设备。

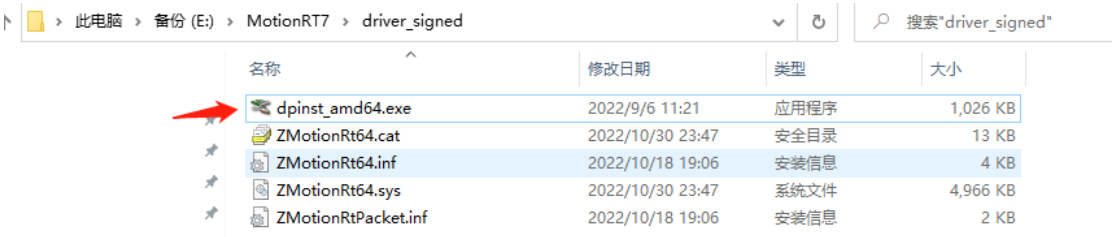


[有卡操作]

有 PCI 设备时，有两种方法。

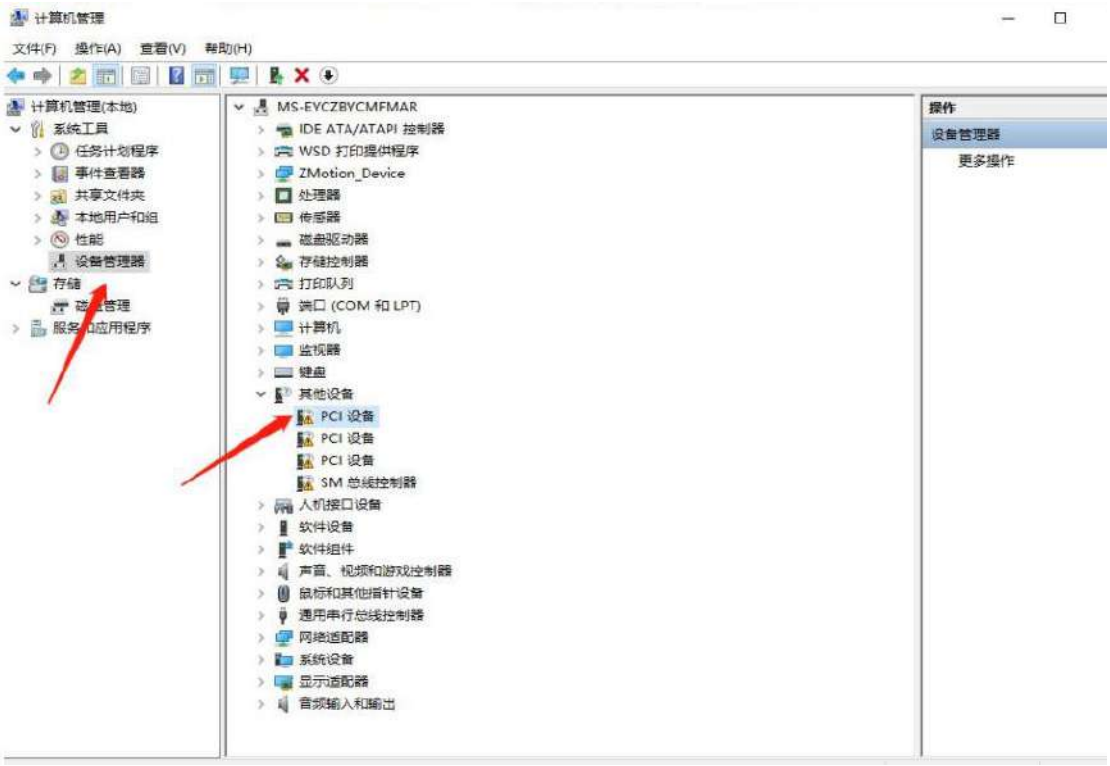
方法一：向导软件安装

打开 MotionRT7 压缩包下面的：MotionRT7 / driver_signed 目录中自带的安装向导软件 dpinst_amd64.exe 自动安装，具体操作按软件向导操作。

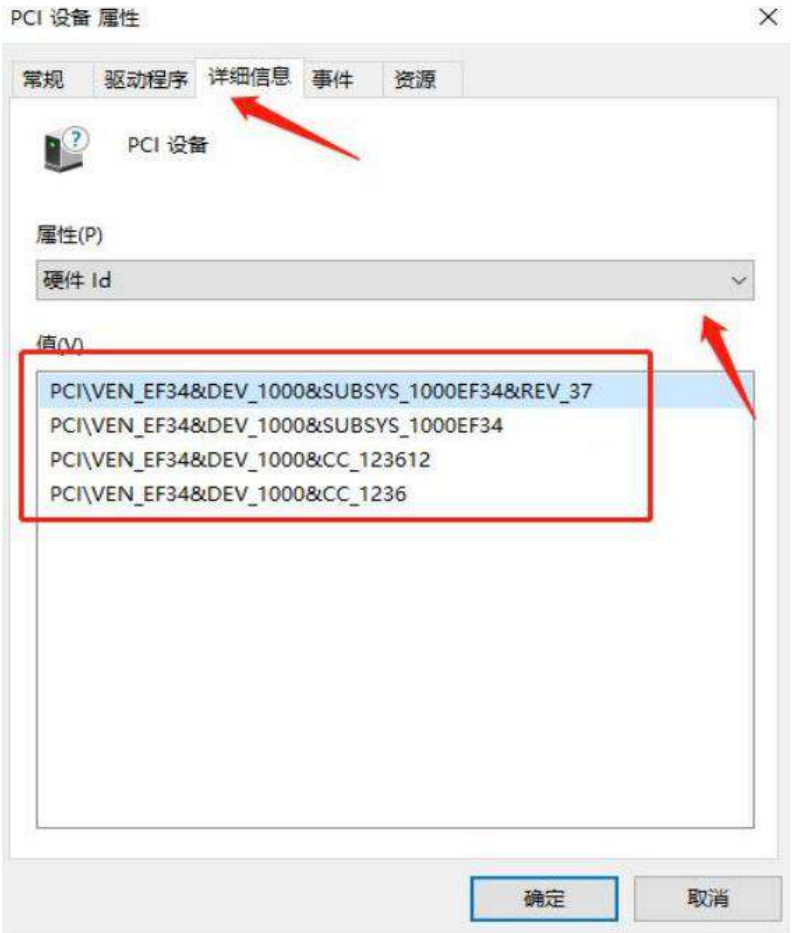


方法二：手动安装

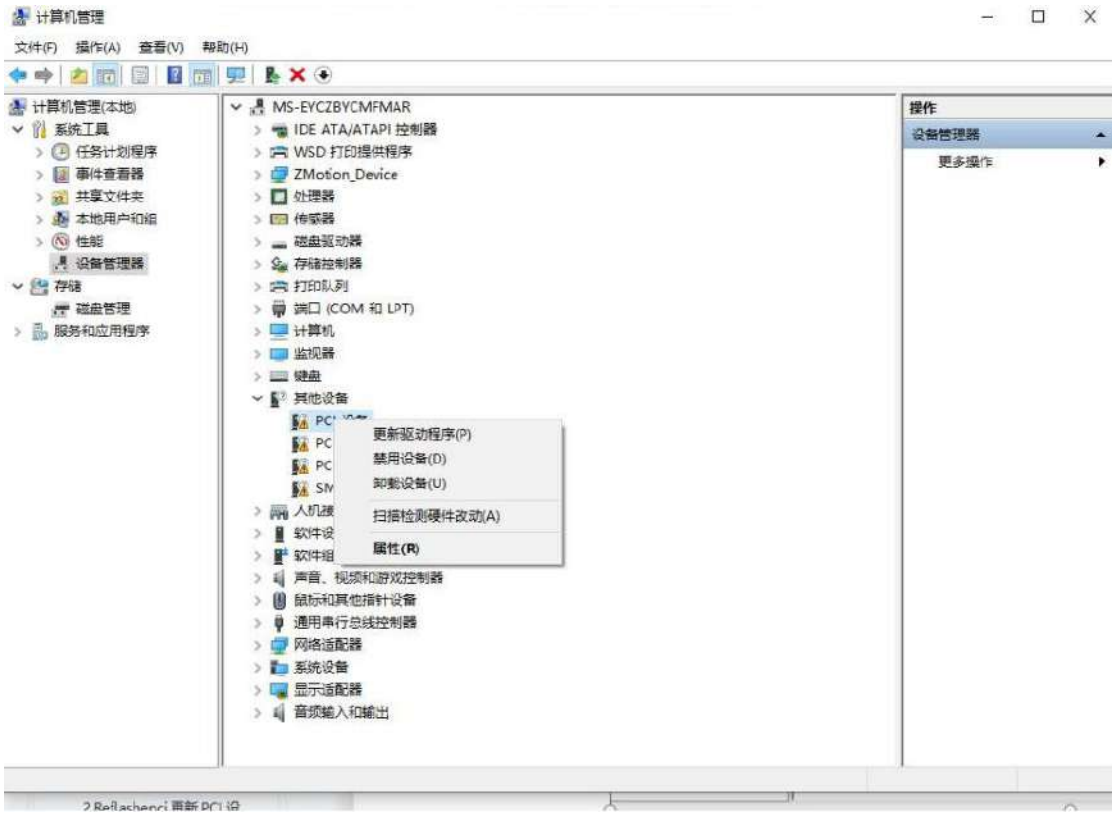
1. 打开设备管理器菜单选择其他设备中的 PCI 设备



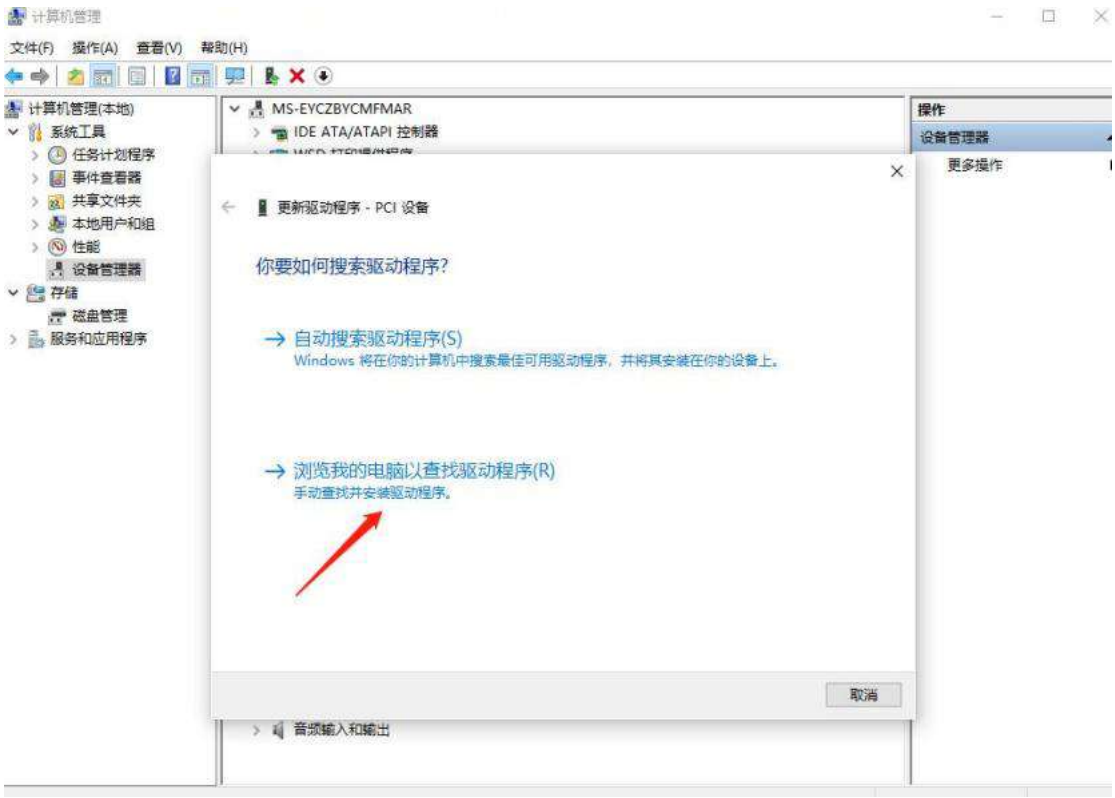
2. 若存在多个 PCI 设备时，鼠标右击“属性”查看详细信息，属性选择“硬件 ID”，确认为 PCI\VEN_EF34&DEV_1000&开头的 PCI 设备。



3. 找到 PCI 设备，右击选择“更新驱动程序”。



4. 选择“浏览我的电脑以查找驱动程序”。



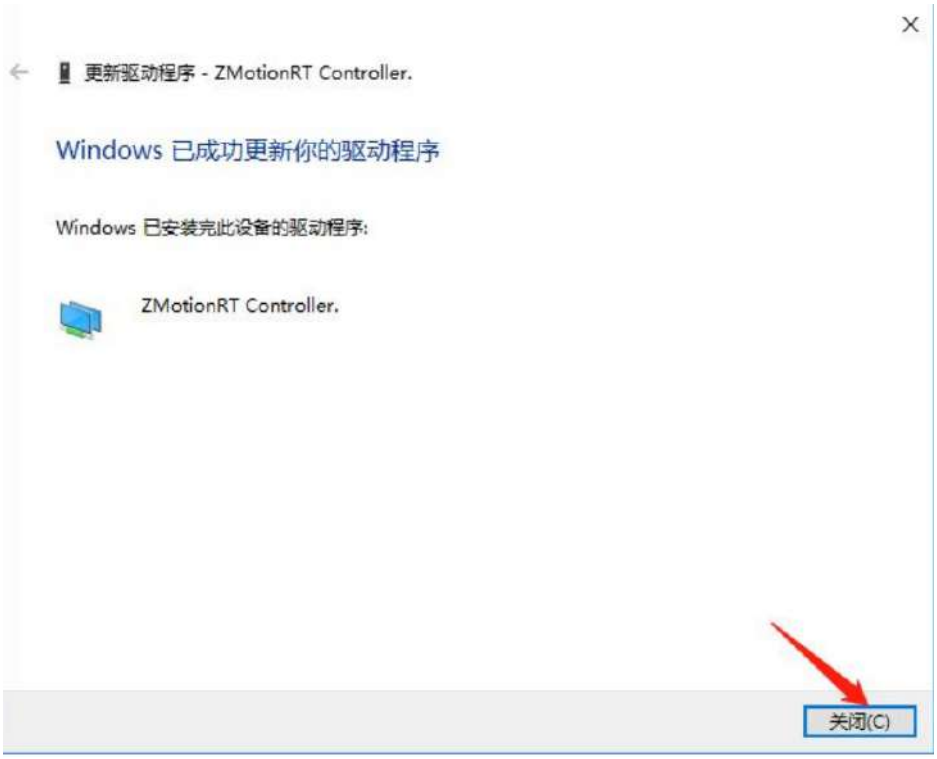
5. 点击浏览，选择驱动 driver 文件夹。



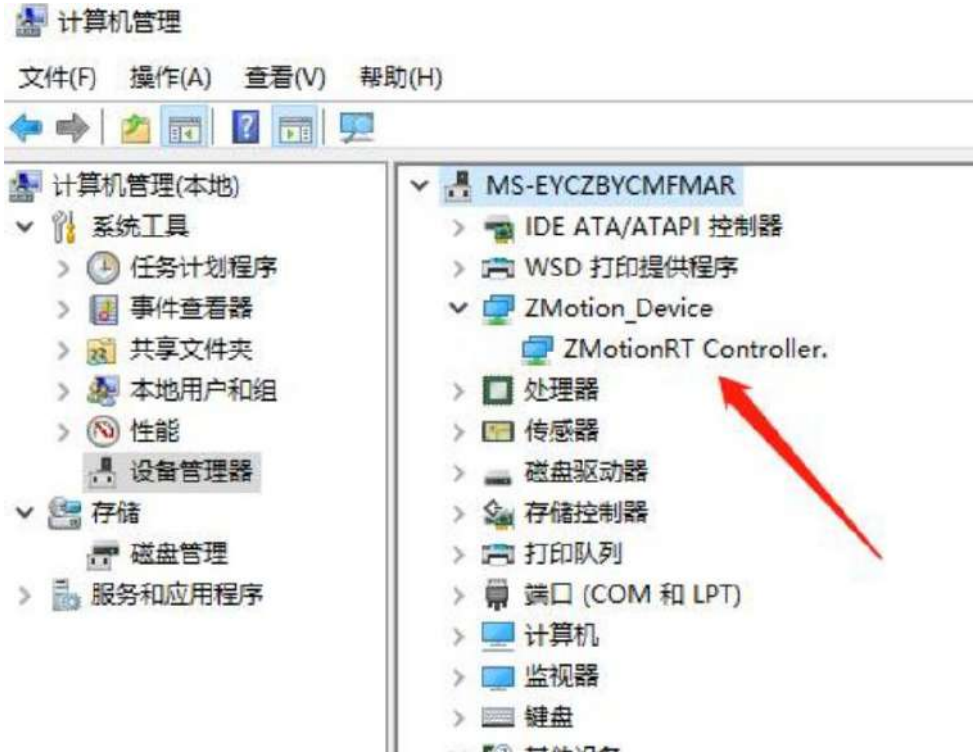
6. 点击下一步。



7. 等待安装完成，点击关闭。



8. 在设备管理器中有 ZMotionRT Controller 就是安装成功。

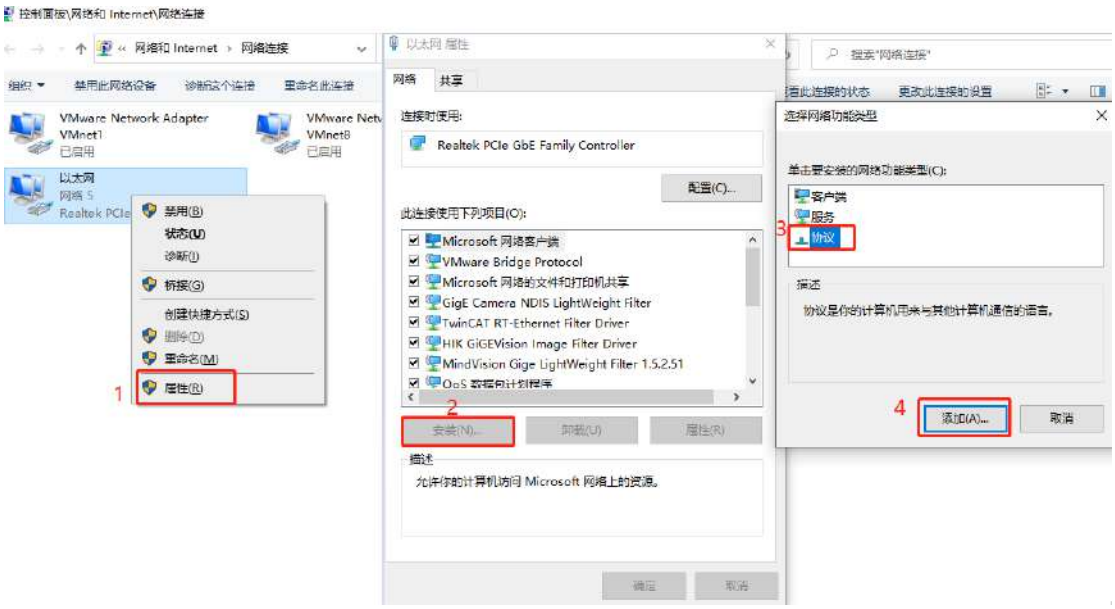


普通网卡安装 Ethercat 总线协议

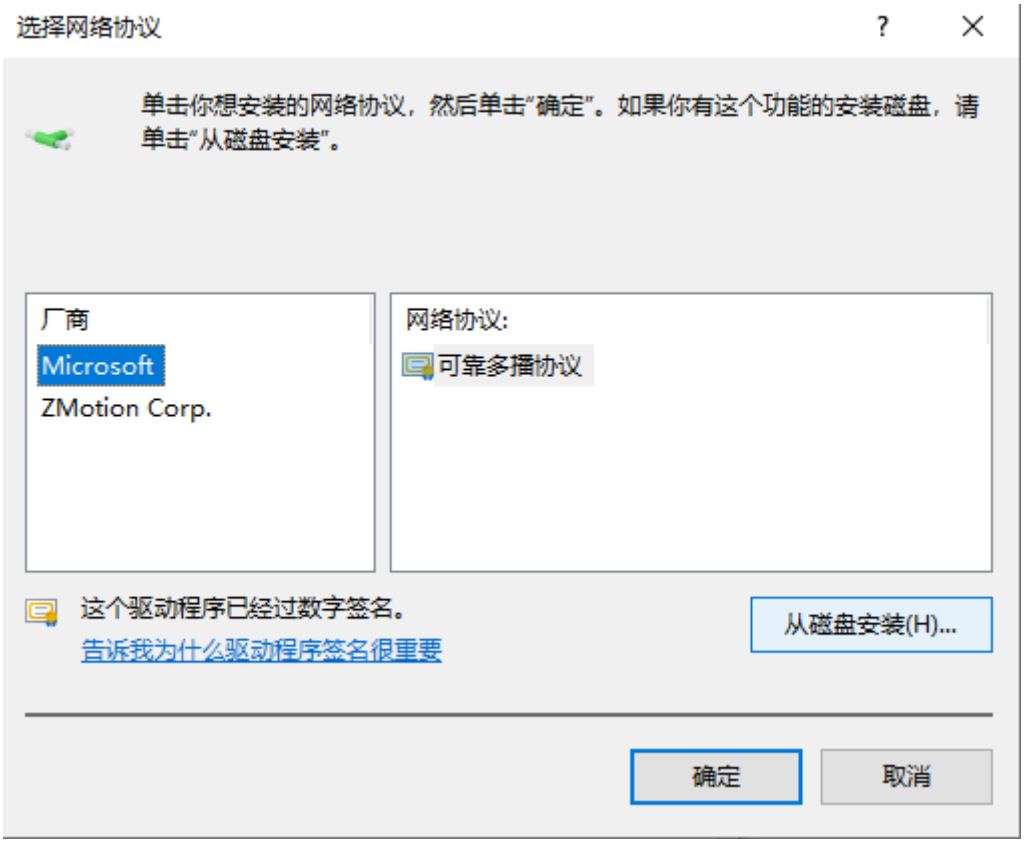
MotionRT710 支持 XPCI/XPCIE 的 ECAT 网口，也支持电脑普通网口作为 ECAT 使用。

1. 打开控制面板，在 Windows 网络连接界面，选择需要用作总线的网口，1 右击属性

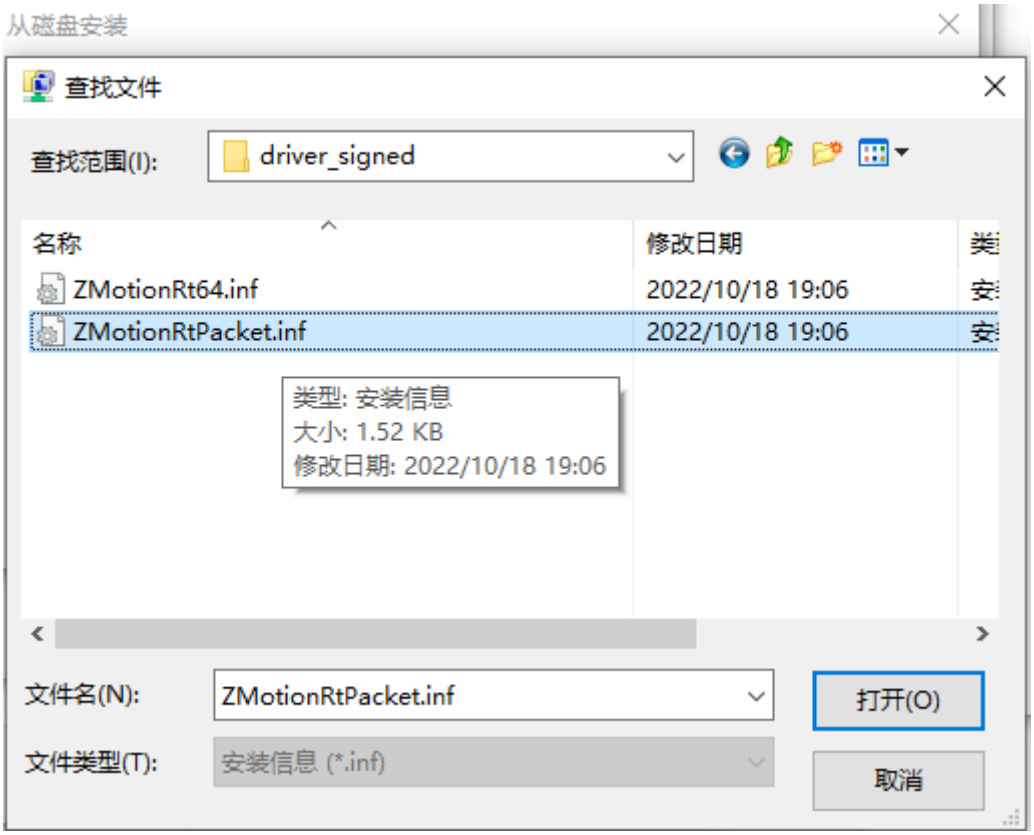
->2 安装->3 协议->4 添加。



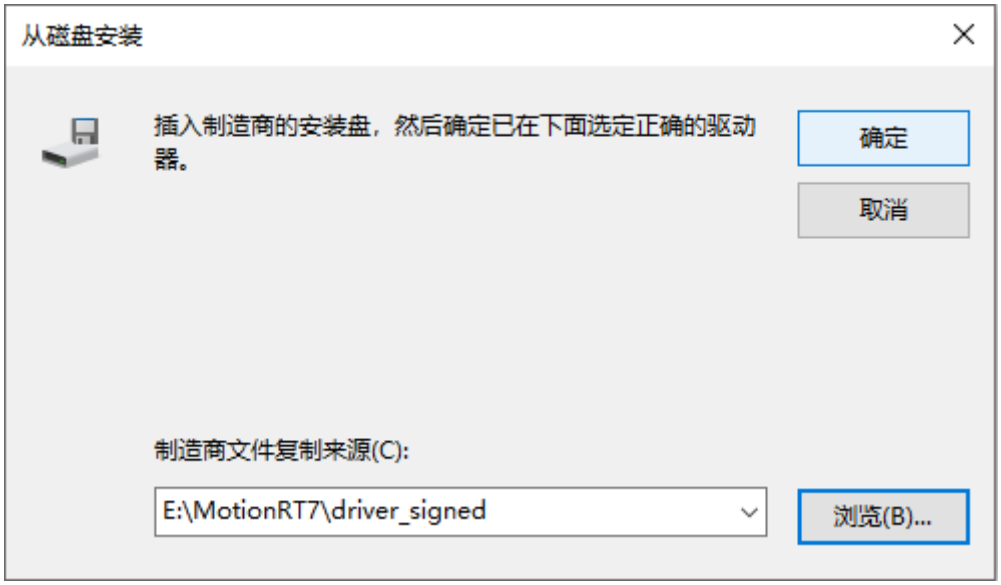
2. 直接单击“从磁盘安装”



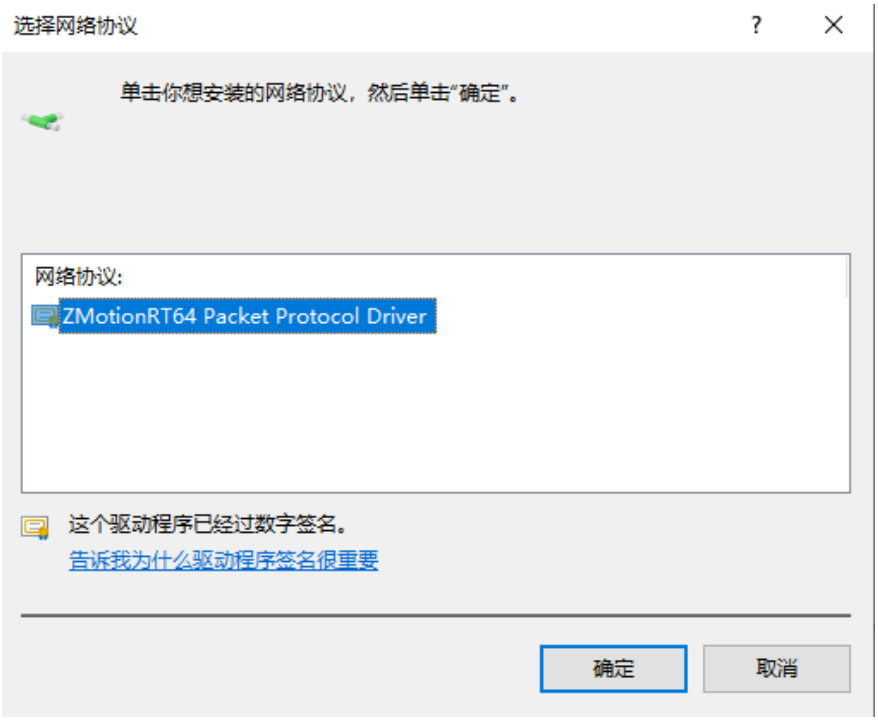
3. 浏览驱动位置，选择 ZMotionRtPacket.inf 文件，并打开。



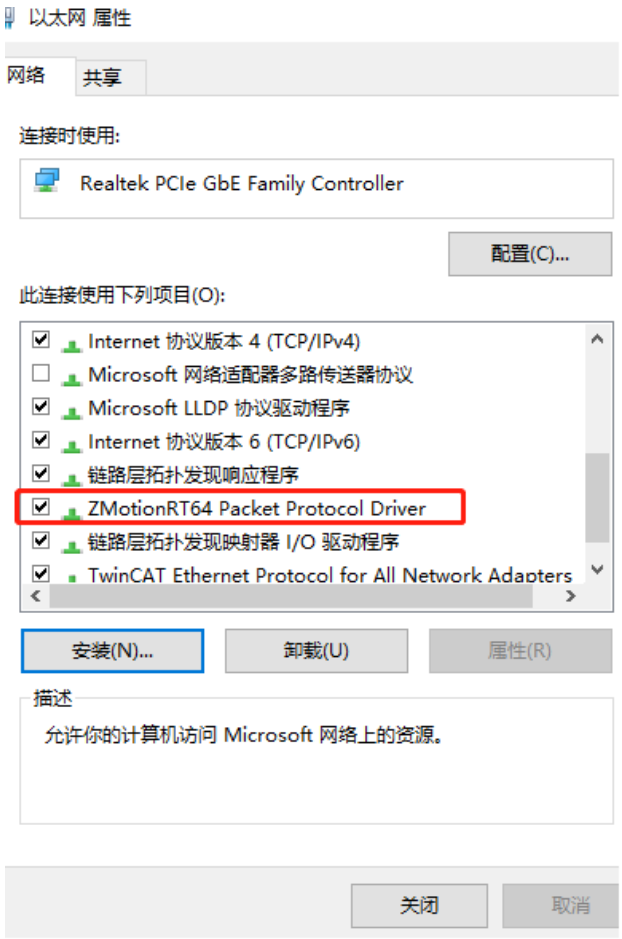
4. 继上一步选完文件后，直接点击确定。



5. 网络协议选中“ZMotionRT64 Packet Protocol Driver”后点击确定

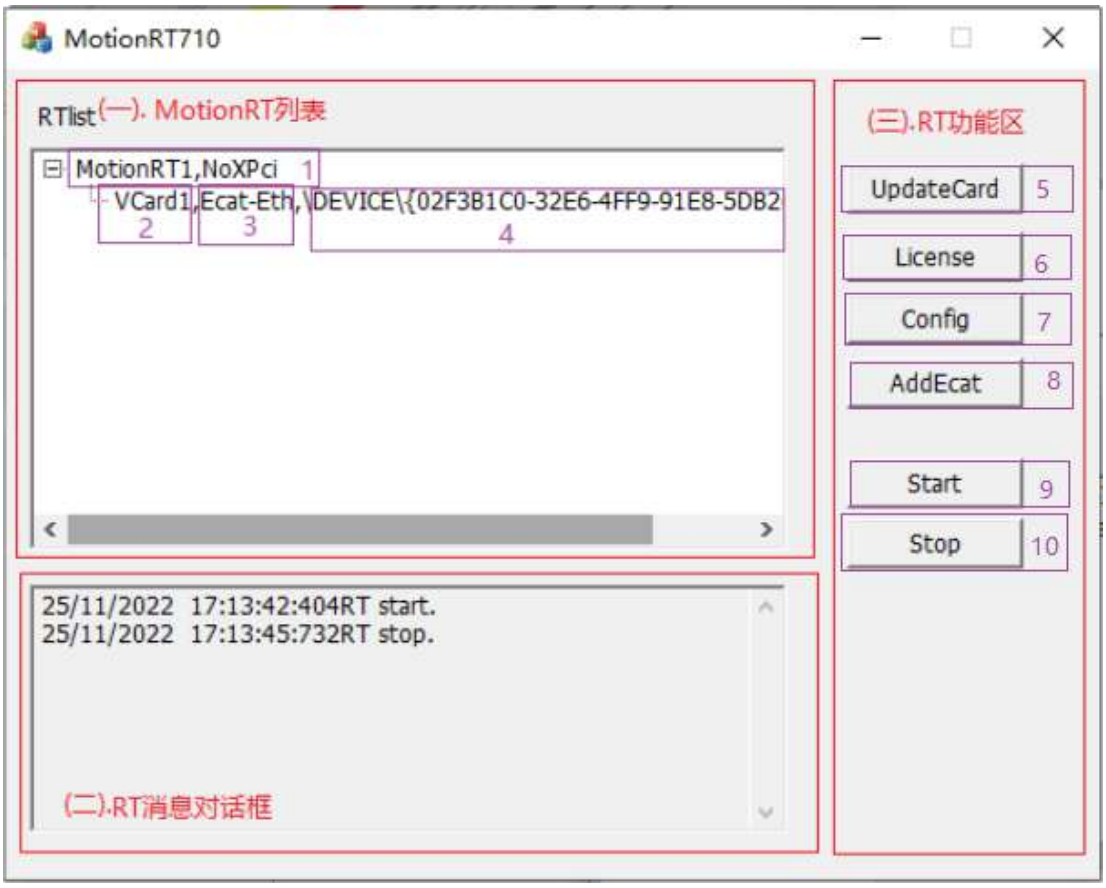


6. 完成后，以太网属性协议，会多出一条“ZMotionRT64 Packet Protocol Driver”，即代表安装成功。



MotionRT 控制台使用

【主界面】



（一）MotionRT 列表：

即当前电脑上存在的 PCI 卡列表

[1]: PCI 卡型号，NoXPci 指无卡

[2]: 卡的类型。

[3]: 卡的 ID。Ecat-Eth 指无卡时用的网口

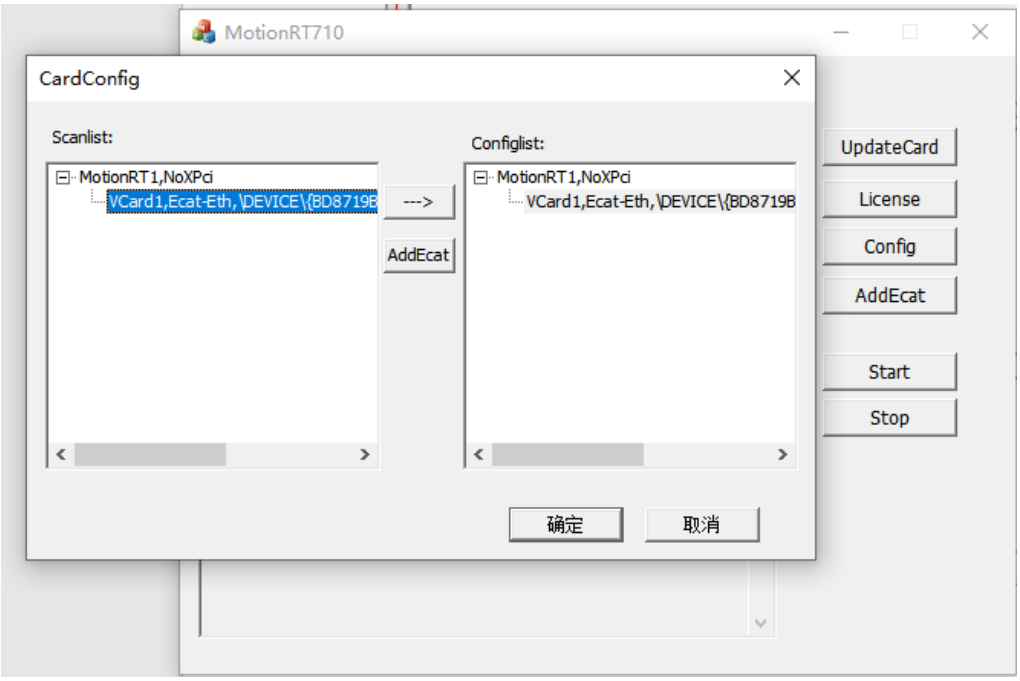
[4]: 卡的拨码地址。无卡时指：网卡的唯一编号

（二）RT 消息对话框

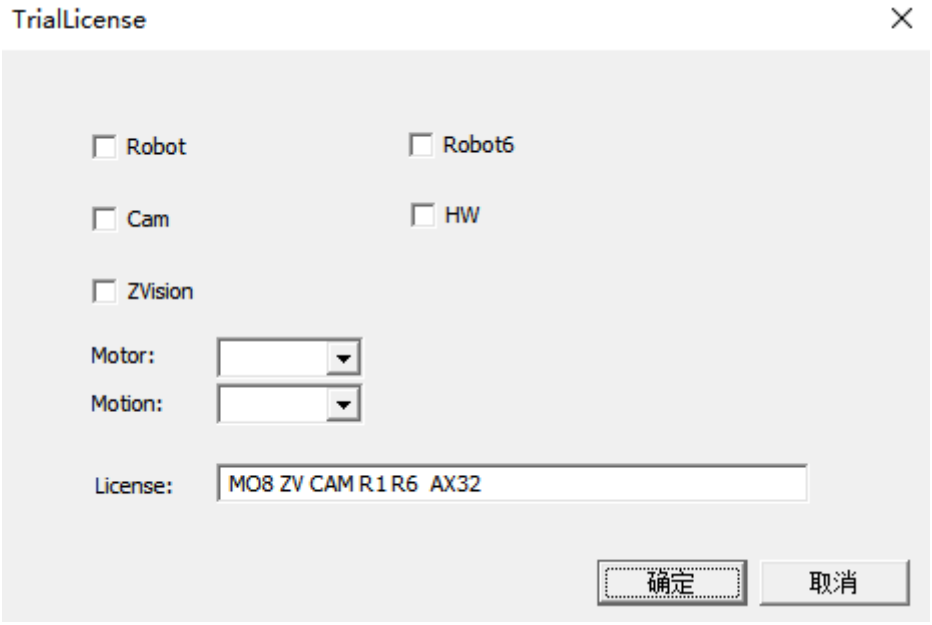
控制台跟各个卡之间交互的消息对话框。

（三）RT 功能区

[5]: UpdateCard: 刷新卡的配置信息，并保存，下次开启控制台的时候自动加载配置信息；首次使用时，需要手动 AddEcat 添加卡号信息，如下图：



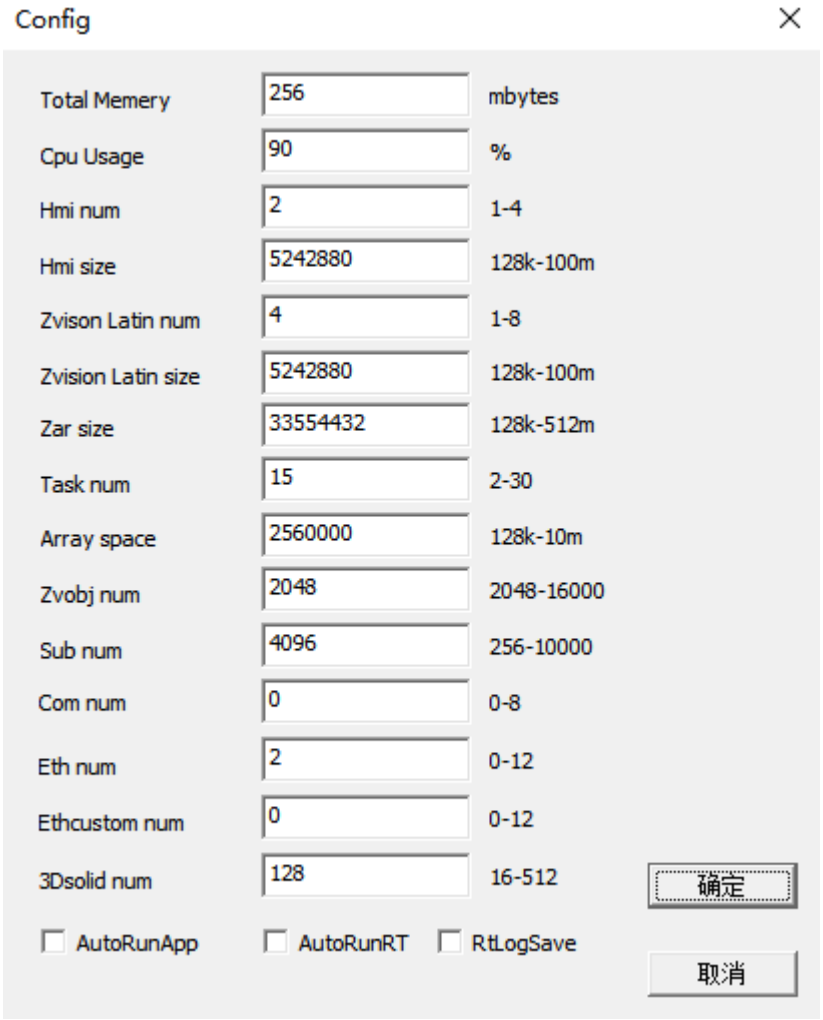
[6]: License 功能界面。可对所需的功能进行勾选配置，参见下表。



功能	描述
Robort	普通机械手
Robort6	六轴以及六轴以下的机械手
Cam	相机功能
HW	硬件位置比较输出功能
ZVision	时间指令功能

Motor	实际电机数量
Motion	支持的运动功能 MO0：点位运动 MO1：点位运动、电子凸轮 MO2：点位运动、电子凸轮、直线插补 MO6：点位运动、电子凸轮、直线插补、圆弧插补 MO8：点位运动、电子凸轮、直线插补、圆弧插补、连续插补
License	用来显示保存 License 参数配置的窗口

[7]: Config 配置界面



功能	描述
Total Memery	总内存，包括控制器内所有能够保存数据的空间所占内存，如数组空间、Zar 文件尺寸、通道大小、hmi 分辨率等等，最好设置值为 200 以上
Cpu Usage	CPU 使用限制

Hmi num	可用的 Hmi 个数
Hmi size	hmi 的分辨率大小
Zvision Latin num	视觉通道数量
Zvision Latin size	单个视觉通道尺寸
Zar size	Zar 文件尺寸大小
Task num	可执行的任务数量
Array Space	分配的数组空间大小
Zvobj num	Zvobject 的个数
Sub num	可 sub 最多的函数个数
Com num	串口个数
Eth num	网口 PORT 个数，设置值要小于最大值
Ethcustom num	自定义网络 PORT 数量
3Dsolid num	3D 固体个数
AutoRunApp	开机自动打开软件
AutoRunRT	开机自动运行 RT 软件
RtLogSave	保存日志到日志文件

[8]: AddEcat

添加可连接的卡

[9]: Start

启动 MotionRT 按钮，启动后，可连接控制卡

[10]: Stop

停止 MotionRT 按钮，停止后，不可连接控制卡

4.1.3 例程

4.1.3-1 网口连接控制器

本例程演示用网口去连接控制器，失败打印错误码。



```
// test1.cpp : 定义控制台应用程序的入口点。
//
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s return code is %d\n", command, ret);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";          //控制器 IP 地址 192.168.0.11,仿真器 IP
地址 127.0.0.1，仿真器连接需把仿真器打开
    ZMC_HANDLE handle = NULL;                     //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("connect controller failed!\n");
        handle = NULL;
        getchar();
        return -1;
    }
    printf("控制器连接成功！\n");

    //运动类型获取}
    Sleep(2000);
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
```

```
}
```

4.1.3-2 串口连接控制器




```
// test1.cpp : 定义控制台应用程序的入口点。
//
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s return code is %d\n", command, ret);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{

    ZMC_HANDLE handle = NULL;  //连接返回的句柄

    uint32 comid = 1; //连接的 COM 口

    int ret = ZAux_OpenCom(comid,&handle); //COM1 口连接控制器
    commandCheckHandler("ZAux_OpenCom", ret);
    if (ERR_SUCCESS != ret)
    {
```

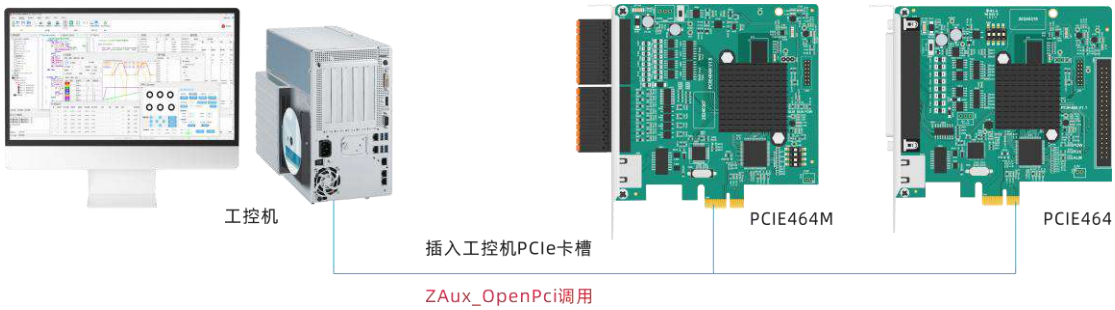
```
        printf("串口连接失败!\n");
        handle = NULL;
    }
    else
    {
        printf("串口连接成功!\n");
    }

    uint32 dwBaudRate = 38400; //波特率 38400
    uint32 dwByteSize = 8;     //8 位数据位
    uint32 dwParity = 0;       //无校验+
    uint32 dwStopBits = 0; ///停止位
    ret = ZAux_SetComDefaultBaud(dwBaudRate, dwByteSize, dwParity, dwStopBits); //设置串口通讯参数
    commandCheckHandler("ZAux_SetComDefaultBaud", ret);
    //do something...

    Sleep(2000);
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

4.1.3-3 PCI 方式连接控制器

连接 PCI 卡，如果失败，会打印错误码，直接返回。




```
// test1.cpp : 定义控制台应用程序的入口点。
//
```

```
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s return code is %d\n", command, ret);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    ZMC_HANDLE handle = NULL;  //连接返回的句柄
    uint32 cardnumcardnum = 0;   //PCI0

    int ret = ZAux_OpenPci(cardnumcardnum, &handle); ////以太网连接控制器以太网连接控制
器
    commandCheckHandler("ZAux_OpenPci", ret);
    if (ERR_SUCCESS != ret)
    {
        printf("PCI 连接失败!\n");
        Sleep(3000);
        handle = NULL;
        return 0;
    }
    else
    {
        printf("PCI 连接成功!\n");
    }

    //do something.....
    Sleep(3000);

    ret = ZAux_Close(handle);
```

```
    commandCheckHandler("ZAux_OpenPci", ret);
    return 0;
}
```

4.1.3-4 修改 IP 地址

```
// test1.cpp : 定义控制台应用程序的入口点。
//
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

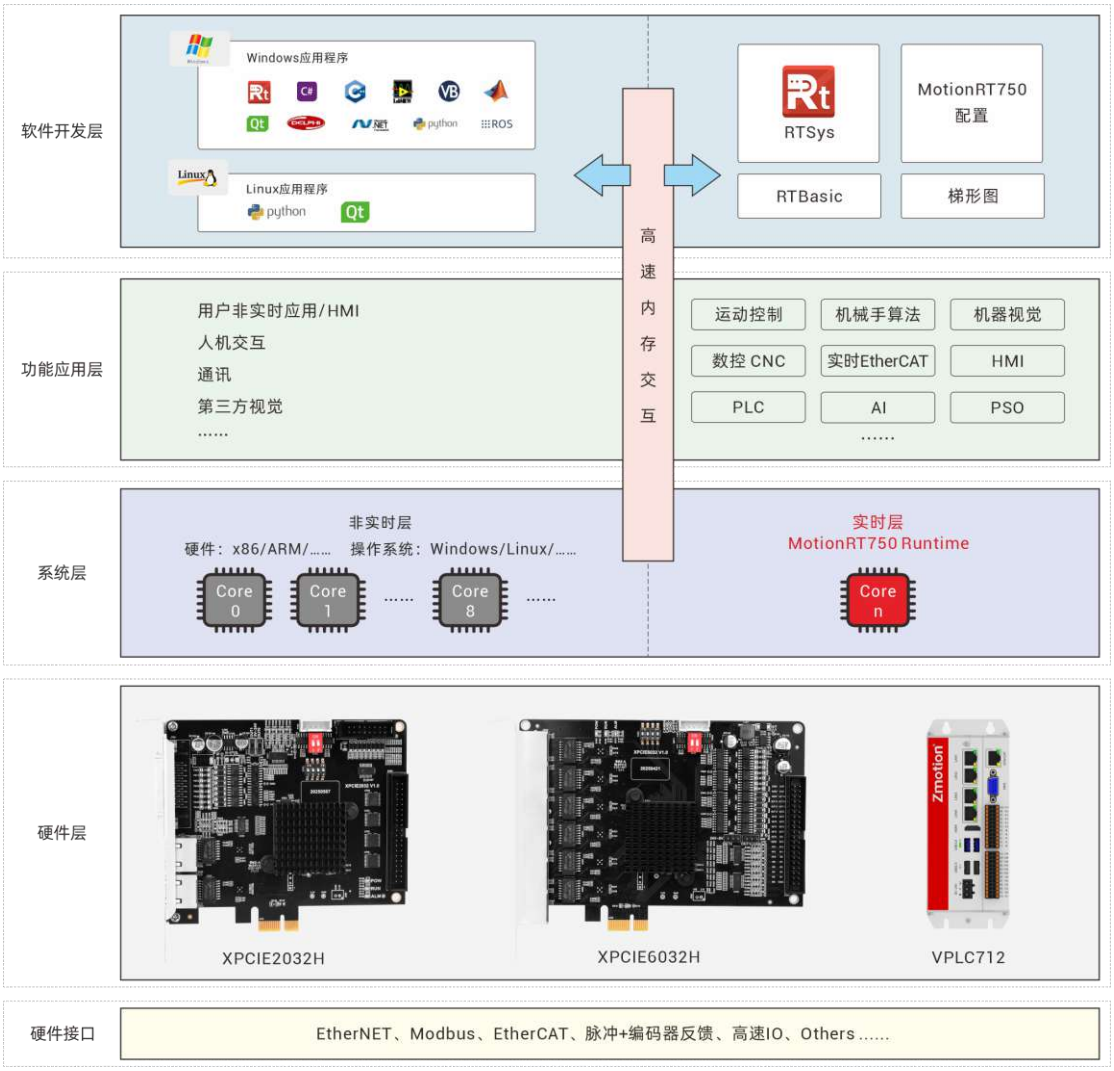
void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s return code is %d\n", command, ret);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";                //控制器 IP 地址,仿真器 IP 地址,
连接需把仿真器打开
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败! \n");
        handle = NULL;
        getchar();
        return -1;
    }
    printf("控制器连接成功! \n");
```

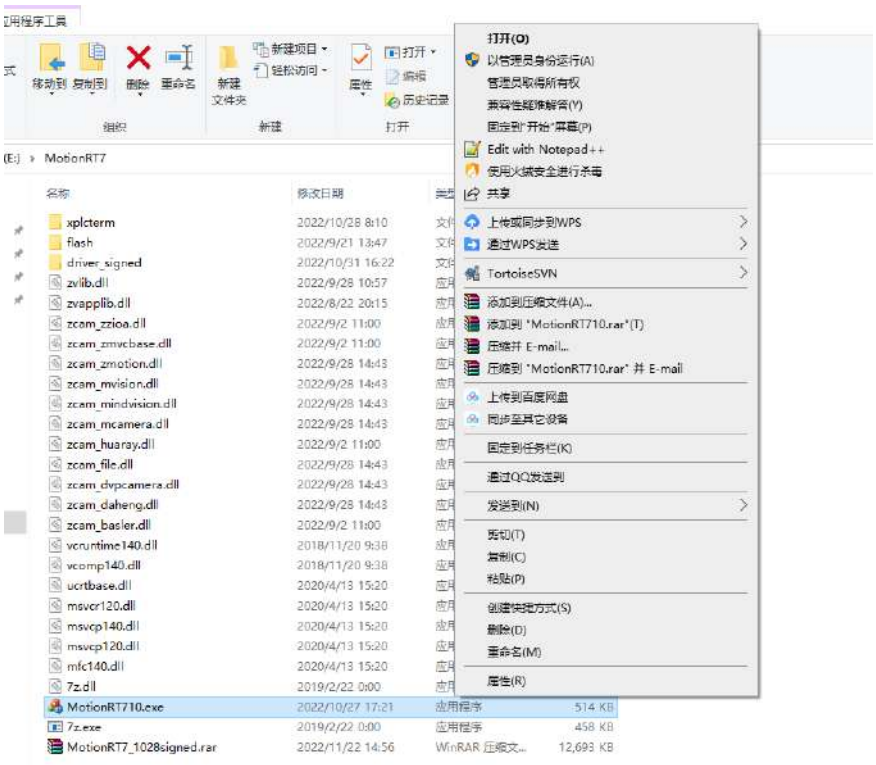
```
    ret = ZAux_SetIp( handle,(char *)"192.168.0.11");//仿真器 IP 无法修改，只能修改实物的，本例程为了示范，所以连接的仿真器
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功

    //运动类型获取}
    Sleep(2000);
    handle = NULL;
    return 0;
}
```

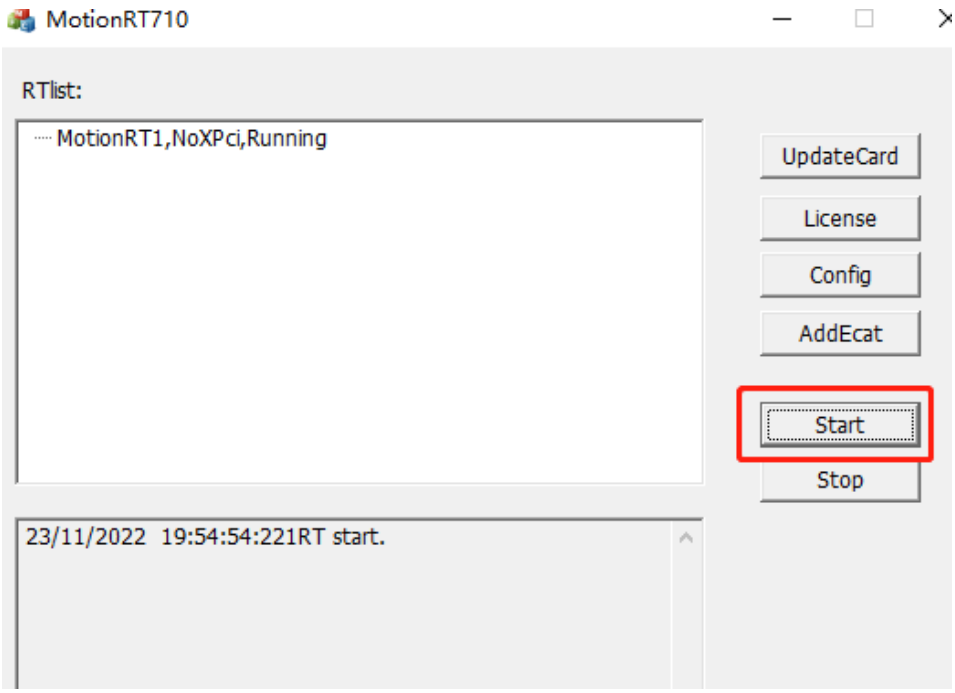
4.1.3-5 MotionRT 连接



连接 MotionRT，首先得先把 MotionRT 控制台打开：



打开控制台，配置好功能及参数后，点击 Start 启动后方可连接。



注意：zmotion.dll 版本要在 3.8.8.50 以上。方可使用 MotionRT。

```
// test1.cpp : 定义控制台应用程序的入口点。
//
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
```

```
void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非0则失败
    {
        printf("%s return code is %d\n", command, ret);
    }

}

int _tmain(int argc, _TCHAR* argv[])
{

    ZMC_HANDLE handle = NULL;                    //连接句柄

    char MotionID[32]= "";              //改模式当前对字符串无要求，可填空字符串
    int ret=ZAux_FastOpen(5, MotionID,1000 ,&handle);    //连接MotionRT

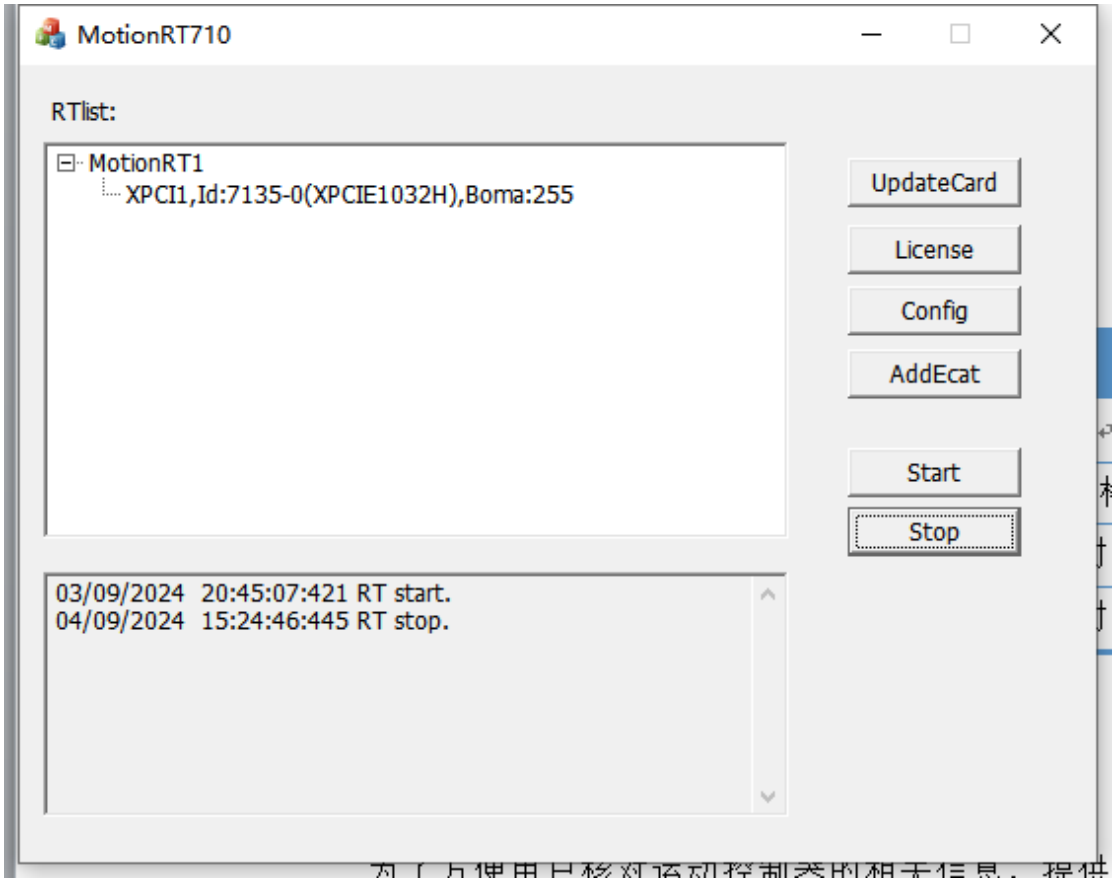
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败！\n");
        handle = NULL;
        getchar();
        return -1;
    }
    printf("控制器连接成功！\n");



    //运动类型获取}
    Sleep(2000);
    handle = NULL;
    return 0;
}
```

4.1.3-6 启停软核

连接 MotionRT, 首先得先把 MotionRT 控制台打开：




```
// test1.cpp : 定义控制台应用程序的入口点。
//
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非0则失败
    {
        printf("%s return code is %d\n", command, ret);
    }

}

int _tmain(int argc, _TCHAR* argv[])
{

    ZMC_HANDLE handle = NULL;                    //连接句柄
    int isel=0;
    int ret=0;
```

```
if (!ZAux_GetLocalRTState(isel))
{
    printf("软核控制台未启动！！\n");
    ret=ZAux_LocalRTManage(isel,1);
    if (ERR_SUCCESS != ret)
    {
        printf("控制台启动失败！\n");
        ZAux_Close(handle);
        handle = NULL;
        getchar();
        return -1;
    }

}else
{
    printf("软核控制台已启动！\n");
}



char MotionID[32]= "";              //改模式当前对字符串无要求，可填空字符串
 ret=ZAux_FastOpen(5, MotionID,1000 ,&handle);   //连接MotionRT

if (ERR_SUCCESS != ret)
{
    printf("控制器连接失败！\n");
    handle = NULL;
    getchar();
    return -1;
}
printf("控制器连接成功！\n");



//运动类型获取}
Sleep(2000);
ZAux_Close(handle);
handle = NULL;
ret=ZAux_LocalRTManage(isel,2);
if (ERR_SUCCESS != ret)
{
    printf("控制台停止失败！\n");
```

```
        getchar();
    }


    return 0;
}
```

4.2 控制器信息获取

4.2.1 指令列表

指令	说明
ZAux_GetControllerInfo	获取控制器卡信息
ZAux_GetSysSpecification	获取控制器最大规格数
ZAux_GetRtcTime	读取控制器 RTC 时间
ZAux_SetRtcTime	设置控制器 RTC 时间

4.2.2 重点说明

为了方便用户核对运动控制器的相关信息，提供指令 **ZAux_GetControllerInfo** 和 **ZAux_GetSysSpecification** 两个函数去获取控制器的相关信息。

可获取控制器型号类型，控制器软件版本（固件版本），控制器唯一 ID，控制器最大虚拟轴数，控制器最大电机数量，控制器最大 IN、OUT、AD、DA 数量，控制器系统日期，控制器系统时间等参数。若还想获取更多的控制器参数，可以通过 **ZAux_Execute** 指令在线发送，使用字符串“?*max”或者“?*set”获取更多的控制器参数。

4.2.3 例程

4.2.3-1 控制器信息

本例程主要演示了如何通过指令获取控制器的相关信息。

```
// test1.cpp : 定义控制台应用程序的入口点。
//
```

```
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s return code is %d\n", command, ret);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";                //控制器出厂默认 IP 地址
192.168.0.11,仿真器 IP 地址 127.0.0.1，仿真器连接需把仿真器打开
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败！\n");
        handle = NULL;
        getchar();
        return -1;
    }
    printf("控制器连接成功！\n");

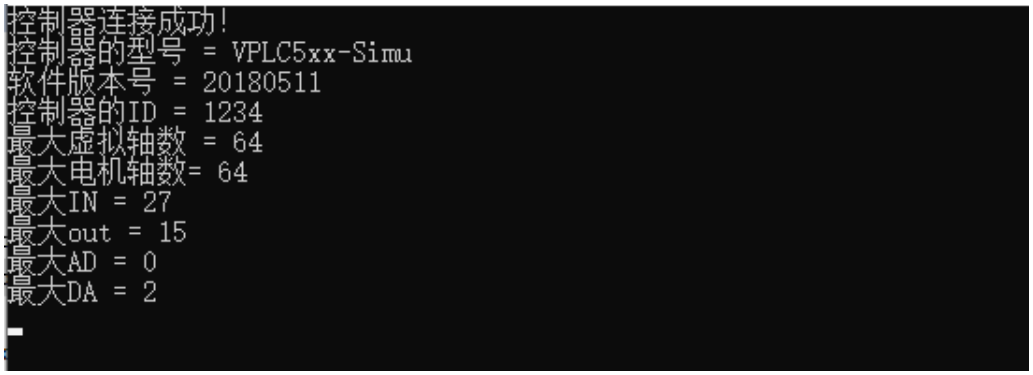
    char SoftTypes[20];
    char SoftVersions[20];
    char ControllerIDs[20];

    ret = ZAux_GetControllerInfo(handle,SoftTypes,SoftVersions,ControllerIDs);//控制卡
信息获取
    commandCheckHandler("ZAux_GetControllerInfo", ret) ;//判断指令是否执行成功
    printf("SoftType = %s\n", SoftTypes);          //打印控制器的型号
    printf("SoftVersion = %s\n", SoftVersions);    //打印软件版本号
```

```
    printf("ControllerId = %s\n", ControllerIDs); //打印控制器的 ID
    uint16 Max_VirtuAxises1;
    uint8 Max_motor1;
    uint8 Max_io1[4];
    ret = ZAux_GetSysSpecification(handle,&Max_VirtuAxises1,&Max_motor1,Max_io1);//获取
控制卡最大规格数
    commandCheckHandler("ZAux_GetSysSpecification", ret) ;//判断指令是否执行成功
    printf("Max_VirtuAxises1 = %u\n", Max_VirtuAxises1);          //最大虚拟轴数
    printf("Max_motor1 = %u\n", Max_motor1);   //最大电机轴数
    printf("Max_io1[0] = %u\n", Max_io1[0]); //最大 IN
    printf("Max_io1[1] = %u\n", Max_io1[1]); //最大 out
    printf("Max_io1[2] = %u\n", Max_io1[2]); //最大 AD
    printf("Max_io1[3] = %u\n", Max_io1[3]); //最大 DA

    Sleep(2000);
    ret = ZAux_Close( handle);//关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    handle = NULL;
    return 0;
}
```

运行结果如下：



第五章 运动控制参数设置

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

轴种类	描述
电机轴	使用控制器的脉冲轴接口、EtherCAT 总线或 RTEX 总线接口和驱动器连接，给设备分配轴号，将 1 台电机作为 1 个轴使用。
虚拟轴	运动控制器内的虚拟轴，不使用实际驱动器，或作为同步控制的虚拟主轴和作为机械手算法中的笛卡尔坐标轴使用。
编码器轴	使用控制器本地编码器轴接口，分配为实际编码器输入加以使用。

电机轴：主动运行，电机根据控制器发出的脉冲来运动，发送的脉冲数根据(运动参数变化量)*(**ZAux_Direct_SetUnits** 设置的 Units)确定，目标需求位置由 **ZAux_Direct_GetDpos** 的规划位置参数体现。

编码器轴：被动运行，编码器跟随电机转动，产生脉冲，反馈控制器，控制器接收的脉冲数调用 **ZAux_Direct_GetEncoder** 确定获取，编码器反馈位置由 **ZAux_Direct_GetMpos** 获取反馈位置参数体现。

当用户连接好一整套系统后（包括运动控制器，驱动器，电机），如何查看整个系统的状态？如何配置这一套系统？如何查看报警信息？如何进行速度，加减速的配置？本章节将介绍用户可以检测到哪些状态以及如何检测，如何配置运动控制参数。

5.1 运动基本参数

5.1.1 指令列表

基本轴参数：

指令	说明
ZAux_Direct_SetAtype	设置轴类型
ZAux_Direct_GetAtype	读取轴类型
ZAux_Direct_SetUnits	设置脉冲当量（units）
ZAux_Direct_GetUnits	读取脉冲当量（units）
ZAux_Direct_SetAccel	设置加速度，单位为 units /s^2
ZAux_Direct_GetAccel	读取加速度，单位为 units/s^2

ZAux_Direct_SetDecel	设置减速度，单位为 units/s^2
ZAux_Direct_GetDecel	读取减速度，单位为 units/s^2
ZAux_Direct_SetSpeed	设置轴速度，单位为 units/s
ZAux_Direct_GetSpeed	读取轴速度，单位为 units/s
ZAux_Direct_SetDpos	设置轴的规划位置，单位为 units
ZAux_Direct_GetDpos	读取轴的规划位置，单位为 units
ZAux_Direct_SetMpos	设置反馈位置，单位为 units
ZAux_Direct_GetMpos	读取反馈位置，单位为 units
ZAux_Direct_GetEncoder	获取控制器接收的脉冲数
ZAux_Direct_SetFastDec	设置快速减速度，单位为 units/s^2
ZAux_Direct_GetFastDec	读取快速减速度，单位为 units/s^2
ZAux_Direct_SetLspeed	设置轴起始速度，单位为 units/s
ZAux_Direct_GetLspeed	读取轴起始速度，单位为 units/s
ZAux_Direct_SetSramp	设置 S 曲线设置。0-梯形加减速
ZAux_Direct_GetSramp	读取 S 曲线设置。0-梯形加减速
ZAux_Direct_GetMspeed	读取反馈速度，单位为 units/s
ZAux_Direct_GetVpSpeed	读取当前轴运行的命令速度.单位为 units/s
ZAux_Direct_SetVpMode	加减速度曲线

轴基本状态：

指令	说明
ZAux_Direct_GetIfIdle	读取轴是否运动
ZAux_Direct_GetAxisStatus	读取轴的状态
ZAux_Direct_GetAxisStatus2	读取轴的使能状态
ZAux_Direct_GetAxisStopReason	读取轴历史异常停止原因

5.1.2 重点说明

5.1.2-1 轴状态

通过 **ZAux_Direct_GetAxisStatus** 指令获取轴的各种状态。按十进制显示数值，按二进制对应位判断状态，可同时发生多个错误。

轴状态字的定义如下：

位	说明	打印值

1	随动误差超限告警	2	2h
2	与远程轴通讯出错	4	4h
3	远程驱动器报错	8	8h
4	正向硬限位	16	10h
5	反向硬限位	32	20h
6	找原点中	64	40h
7	HOLD 速度保持信号输入	128	80h
8	随动误差超限出错	256	100h
9	超过正向软限位	512	200h
10	超过负向软限位	1024	400h
11	CANCEL 执行中	2048	800h
12	脉冲频率超过 MAX_SPEED 限制需要修改降速或修改 MAX_SPEED	4096	1000h
14	机械手指令坐标错误	16384	4000h
18	电源异常	262144	40000h
20	轴速度保护	1048576	100000h
21	运动中触发特殊运动指令失败	2097152	200000h
22	告警信号输入	4194304	400000h
23	轴进入了暂停状态	8388608	800000h

调用 **ZAux_Direct_GetAxisStopReason** 查看轴历史停止原因锁存，写 0 清除，按位锁存，锁存的是轴状态的信息。

调用 **ZAux_Direct_GetIfIdle** 判断加在轴上的运动指令是否完成，运动中返回 0，运动结束返回-1。

5.1.2-2 轴类型

轴类型调用 **ZAux_Direct_SetAtype/ZAux_Direct_GetAtype**（ATYPE）指令根据当前轴的特性去配置，在用户程序初始化时应第一时间完成轴类型的配置，类型不匹配会报错。

所有未分配的轴默认为虚拟轴，ATYPE 的值为 0。

控制器支持的轴类型如下：

ATYPE 类型	描述
0	虚拟轴
1	脉冲方向方式的步进或伺服
2	模拟信号控制方式的伺服(预留)
3	正交编码器
4	脉冲方向输出+正交编码器输入

5	脉冲方向输出+脉冲方向编码器输入
6	脉冲方向方式的编码器
7	脉冲方向方式步进或伺服+EZ 信号输入
8	ZCAN 扩展脉冲方向方式步进或伺服
9	ZCAN 扩展正交编码器
10	ZCAN 扩展脉冲方向方式的编码器
20	振镜类型，带振镜状态反馈 振镜连接不上 AXISSTATUS 的 bit2 会置位，ENCODER 返回原始的发送位置，脉冲单位 ZMC408SCAN 支持
21	振镜轴类型，需要控制器支持 缺省系统周期 250us，振镜刷新周期 50us，与固件有关 可以使用普通轴的所有运动控制指令，支持振镜轴与其它轴类型混合插补
22	振镜轴类型，带振镜位置反馈 振镜连接不上 AXISSTATUS 的 bit2 会置位，振镜报警 AXISSTATUS 的 bit3 会置位 MPOS 返回反馈位置，做了反矫正处理，ENCODER 返回原始的反馈位置脉冲单位 ZMC408SCAN 支持
24	远程编码器轴类型 ZHD500X 上手轮使用，需要控制器 5 系列 20180404 以上固件版本支持
50	RTEX 周期位置模式，需 RTEX 控制器
51	RTEX 周期速度模式，需 RTEX 控制器
52	RTEX 周期力矩模式，需 RTEX 控制器 请先关闭驱动器 2 自由度控制模式，并设置速度限制
65	EtherCAT 周期位置模式，需支持 EtherCAT
66	EtherCAT 周期速度模式，需支持 EtherCAT DRIVE_PROFILE 要设置为 20 或以上
67	EtherCAT 周期力矩模式，需支持 EtherCAT DRIVE_PROFILE 要设置为 30 或以上
70	EtherCAT 自定义操作，只读取编码器，需支持 EtherCAT

1. 轴类型(**ZAux_Direct_SetAtype**)=0:虚拟轴

多轴同步运动时可作为机器的主轴，从轴都跟随此虚拟主轴。

作为其他轴的叠加轴，给实际运动的轴叠加一个虚拟轴，这些虚拟轴可以用 ADDAX 指令（轴叠加）进行设置，然后将各个虚拟轴的运动叠加到实轴。

2. 轴类型(**ZAux_Direct_SetAtype**)=1 或 7:脉冲轴

轴的运动由控制器发脉冲控制，脉冲的方向决定电机的转向，根据发送脉冲的频率控制轴运动的快慢。

3. 轴类型(**ZAux_Direct_SetAtype**)=3 或 6:编码器轴

编码器单独占用一个轴号时，根据编码器类型，轴类型可选 3 或 6。

4. 轴类型(**ZAux_Direct_SetAtype**)=4 脉冲轴和编码器轴共用轴号

当前的脉冲轴带编码器反馈时，轴类型设为 4，脉冲输出和编码器输入信号在同一个轴上。

5. 轴类型(**ZAux_Direct_SetAtype**)=8:CAN 扩展轴

使用 CAN 总线扩展轴时，扩展的脉冲轴的轴类型设成 8，扩展轴上接入编码器轴的轴类型设为 9。

6. 轴类型(**ZAux_Direct_SetAtype**)=21:振镜轴号

接入激光振镜设备时，需要将振镜轴类型设为 21，激光振镜轴部分型号支持。

7. 轴类型(**ZAux_Direct_SetAtype**)=50,51,52 : RTEX 总线轴

使用 RTEX 总线驱动器时，轴类型只能在以上三个中选择，其中轴类型(**ZAux_Direct_SetAtype**)=50 是位置模式，使用运动指令控制电机运行；轴类型(**ZAux_Direct_SetAtype**)=51 速度模式，速度模式下使用 DAC 指令设置电机的运行速度，并持续运行；轴类型(**ZAux_Direct_SetAtype**)=52 力矩模式，力矩模式下使用 DAC 指令设置电机的力矩，并持续运行.

速度和力矩模式下若要切换模式，为防止事故，先将 DAC 置 0 后再使用 **ZAux_Direct_SetAtype** 指令切换。

注意：修改轴类型(ZAux_Direct_SetAtype)切换到力矩模式之前，请先将驱动器参数 Pr6.47 的第一位置为 0，关闭 2 自由度控制模式。再通过参数 Pr3.17 设置速度限制。当 Pr3.17(速度限制选择)的设定值是 0 时，通过 Pr3.21 设置速度限制，设定值是 1 时，可以通过 SL_SW 切换转矩控制时的速度限制值采用 Pr3.21 还是 Pr3.22。

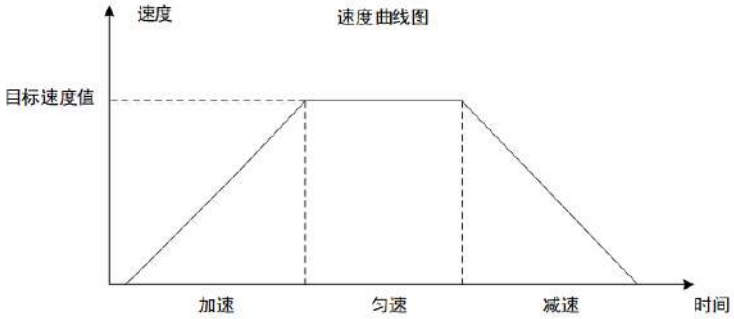
8. 轴类型(**ZAux_Direct_SetAtype**)=65,66,67 EtherCAT 总线轴号

使用 EtherCAT 总线驱动器时，轴类型只能在以上三个中选择，其中轴类型(**ZAux_Direct_SetAtype**)=65 是位置模式，使用运动指令控制电机运行；轴类型(**ZAux_Direct_SetAtype**)=66 速度模式，速度模式下使用 DAC 指令设置电机的运行速度，并持续运行，速度单位有两个，脉冲数/S 和 R/MIN，由驱动器确定；轴类型(**ZAux_Direct_SetAtype**)=67 力矩模式，力矩模式下使用 DAC 指令设置电机的力矩，并持续运行，力矩控制模式下 DAC 值范围 0-1000，对应 0-100%。

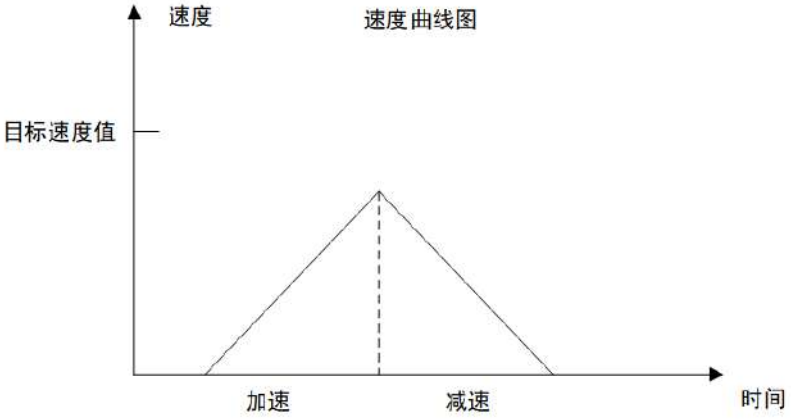
速度和力矩模式下若要切换模式，为防止事故，先将 DAC 置 0 后再使用轴类型(**ZAux_Direct_SetAtype**)指令切换。

5.1.2-3 轴速度

速度曲线一般分为三个阶段：加速阶段、匀速阶段、减速阶段，如下图所示。



当位移较短时，可能不存在匀速阶段，仅有加减速阶段，如下图所示。

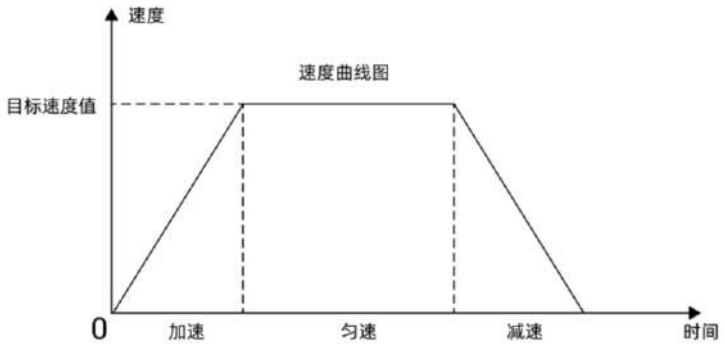


常用的速度指令有 **ZAux_Direct_SetSpeed** 设置运动速度，**ZAux_Direct_SetAccel** 加速度，**ZAux_Direct_SetDecel** 减速度，**ZAux_Direct_SetFastDec** 快减减速度（急停。报警、限位等情况下使用快减速速度快速停止运动）等，在轴参数初始化时设置完成，作为运动指令的速度参考。

梯形图曲线

不设置 SRAMP（令 SRAMP 等于 0。调用 **ZAux_Direct_SetSramp** 设置），速度曲线为梯形曲线，在此速度规划方式下，速度曲线按梯形曲线变化。保持速度、加减速度等参数值不变。

速度到达设定数值后匀速运动，若只设置加速度，减速度为 0 时，减速度会自动等于加速度的值。一般在运动前就设置好相应的加减速，运动过程中不要修改，运动中调整会导致运动轨迹变化。如图：S 曲线时间(**ZAux_Direct_SetSramp**)=0（加减速过程较快，速度变化对机床冲击较大。）



例：假设目标速度值为：100，加速度为 1000，减速度为：1000 则：

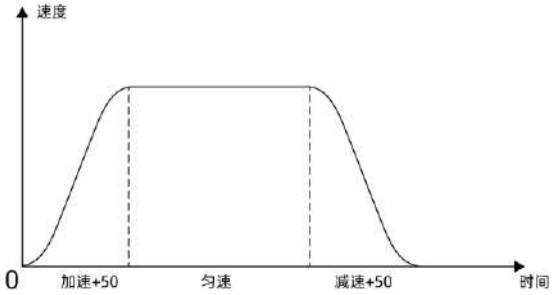
总加速时间=100/1000=0.1s

总减速时间=100/1000=0.1s

S 型曲线

通过调用 **ZAux_Direct_SetSramp** 来设定合适的加减速变化率，使得速度曲线平滑，在机械启停或加减速时减少抖动。S 曲线时间(**ZAux_Direct_SetSramp**)值的范围在 0 到梯形加速时间之间（单位 ms），设置后加减速过程会变长相应的时间，时间越长速度曲线越平滑。设置时间若超过梯形加速时间，按照梯形加速时间进行。

当 S 曲线时间(**ZAux_Direct_SetSramp**)=50 时，得到 S 型曲线如下：加减速时更柔和。



例：假设目标速度值为：100，加速度为 1000，减速度为：1000，S 曲线时间为 50ms，则：

总加速时间=100/1000 + 0.05=0.1s+0.05s=0.15s

总减速时间=100/1000 + 0.05=0.1s+0.05s=0.15s

5.1.2-4 脉冲当量

在运动控制装置中，当机械结构确定以后，电机和机械装置的传动关系也就固定了，电机每转一圈产生的机械位移量也就固定了。脉冲当量（**ZAux_Direct_SetUnits**）是每单位对应的脉冲数，可以是单位距离、单位角度等，支持 5 位小数精度。

控制器以脉冲当量（**ZAux_Direct_SetUnits**）作为基本单位，运动的目标位置、速度、加减速度等都是以脉冲当量（**ZAux_Direct_SetUnits**）作为基础单位来进行运算执行，脉冲

当量（**ZAux_Direct_SetUnits**）修改后，目标位置、速度、加减速度等会随脉冲当量（**ZAux_Direct_SetUnits**）改变成比例变化。

假设脉冲型运动控制器，控制器发 1000 个脉冲，设备机构走 1mm,则把脉冲当量 units 设置为 1000 时，运动中 move 相关的指令，走 1mm，实际发了 1000 个脉冲。速度设置为 100 个用户单位，则为 100mm/s，控制轴走 50 用户单位时则走 50mm。

语法：**ZAux_Direct_SetUnits**(控制器连接句柄，轴号，变量)

伺服脉冲当量设置：

减速比：即减速装置的传动比，是传动比的一种，是指减速机构中瞬时输入速度与输出速度的比值。

螺纹导程：是螺纹上任意一点沿同一条螺旋线转一周所移动的轴向距离，螺距指的是螺纹上相邻两牙对应点的轴向距离，代号是 P。单线螺纹的导程等于螺距，多线螺纹的导程等于头数（n）乘以螺距，即导程 S=nP。

若机构存在导程（单位 mm）以及减速机，则要把脉冲当量转换成 mm 为：

脉冲当量=电子齿轮比*导程/ 编码器分辨率*机械减速比

注：

1.电机转了 a 圈，减速机输出转了 b 圈，则减速比为：a:b

2.若电机直连丝杆，则机械减速比为 1：1

3.若轴为旋转轴，则导程为 360°

4.一圈脉冲数=编码器分辨率/电子齿轮比

步进脉冲当量设置：

步距角：一个脉冲，步进电机所转动的角度；

细分：把步距角细分成 N 等份，则一个脉冲，步进电机转动的角度为：步距角/细分

步进电机旋转一圈脉冲数为：360/（步距角/细分）=360*细分/步距角

减速比：即减速装置的传动比，是传动比的一种，是指减速机构中瞬时输入速度与输出速度的比值。

螺纹导程：是螺纹上任意一点沿同一条螺旋线转一周所移动的轴向距离，螺距指的是螺纹上相邻两牙对应点的轴向距离，代号是 P。单线螺纹的导程等于螺距，多线螺纹的导程等于头数（n）乘以螺距，即导程 S=nP。

若机构存在导程（单位 mm）以及减速机，则要把脉冲当量转换成 mm 为：

脉冲当量=导程/（（360/（步距角/细分））*减速比）=（导程*步距角）/（360*减速比*细分）

注：

1.电机转了 a 圈，减速机输出转了 b 圈，则减速比为：a:b

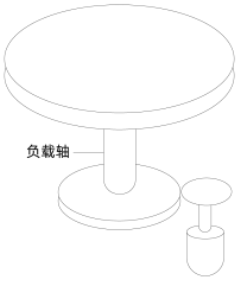
2.若电机直连丝杆，则机械减速比为 1：1

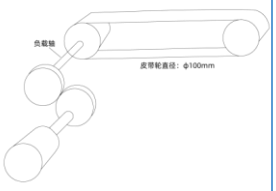
3.若轴为旋转轴，则导程为 360°

4.导程：电机旋转一周时，其推进面上任一点在轴向的前进距离。

5.步进电机默认是走一步转 1.8°的角度。则 1.8°就是步距角的值。所以步进电机默认是 360/1.8=200 个脉冲一转，但是在一些机构里面精度要求高，1.8°不适合，所以就有了驱动器上面的细分数设置，比如说驱动器上面有一个 5 的细分，即角度缩小了 5 倍即 1.8/5=0.36，就是指电机每走一步转动的角度为 0.36°，360/0.36=1000 个脉冲一转。

常见的几种机械传动的 UNITS（脉冲当量：**ZAux_Direct_SetUnits**）计算方式见下表。

机械结构	图例	机械规格	编码器分辨率	UNITS
滚珠丝杆		滚珠丝杆导程：10mm 减速比：1：1	驱动器电子齿轮比:1：1 编码器分辨率：10000 1 圈脉冲数：10000	1 圈脉冲数÷丝杆导程： UNITS=10000/10=1000 即一个 units 机构运动 1mm
圆台		1 圈旋转角度：360 减速比：1/5	驱动器电子齿轮比:1：1 编码器分辨率：10000 1 圈脉冲数：10000/1	如果圆台与电机直连 电机一度脉冲=1 圈 脉冲数/角度，则： UNITS=10000/360。 如果圆台通过减速机联结到伺服电机， 减速比 1：5 则电机旋 转 5°，机构旋转 1° 则有 1:5 减速结构的 UNITS=10000/360/5。

皮带+皮带轮		皮带轮直径：100mm 皮带轮周长：314mm 减速比:1/1	驱动器电子齿轮比:1：1 编码器分辨率：10000 1 圈脉冲数：10000/1	1 圈脉冲数/周长：UNITS=10000/314

5.1.3 例程

5.1.3-1 轴基本运动参数设置获取

```
// test1.cpp : 定义控制台应用程序的入口点。
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s return code is %d\n", command, ret);
    }

}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";                //控制器出厂默认 IP 地址
192.168.0.11,仿真器 IP 地址 127.0.0.1，仿真器连接需把仿真器打开
    ZMC_HANDLE handle = NULL;                        //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
```

```
        printf("connect controller failed!\n");
        handle = NULL;
        getchar();
        return -1;
    }
    printf("控制器连接成功! \n");

    ZAux_Direct_SetAtype(handle, 0, 1);   //设置轴 0 轴 类型为 1
    ZAux_Direct_SetUnits(handle, 0, 100); //设置轴 0 脉冲当量为 100
    ZAux_Direct_SetSpeed(handle, 0, 200); //设置轴 0 速度为 200units/s
    ZAux_Direct_SetAccel(handle, 0, 2000); //设置轴 0 加速度为 2000units/s/s
    ZAux_Direct_SetDecel(handle, 0, 2000); //设置轴 0 减速度为 2000units/s/s

    float dpos;
    int ifIdle;
    int status;
    float DposValue;

    int Atype;
    float Units;
    float Speed;
    float Accel;
    float Decel;
    ret =ZAux_Direct_GetAtype(handle, 0, &Atype);      //获取轴 0 轴类型
    commandCheckHandler("ZAux_Direct_GetAtype", ret);//判断指令是否执行成功
    printf("轴 0 轴类型 Atype = %d\n", Atype);

    ret =ZAux_Direct_GetUnits(handle, 0, &Units); //获取轴 0 脉冲当量
    commandCheckHandler("ZAux_Direct_GetUnits", ret);//判断指令是否执行成功
    printf("轴 0 脉冲当量 Units = %f\n", Units);

    ret =ZAux_Direct_GetSpeed(handle, 0, &Speed); //获取轴 0 速度
    commandCheckHandler("ZAux_Direct_GetSpeed", ret);//判断指令是否执行成功
    printf("轴 0 速度 Speed = %f\n", Speed);

    ret =ZAux_Direct_GetAccel(handle, 0, &Accel); //获取轴 0 加速度
    commandCheckHandler("ZAux_Direct_GetAccel", ret);//判断指令是否执行成功
```

```
printf("轴 0 加速度 Accel = %f\n", Accel);

ret =ZAux_Direct_GetDecel(handle, 0, &Decel); //获取轴 0 减速度
commandCheckHandler("ZAux_Direct_GetDecel", ret);//判断指令是否执行成功
printf("轴 0 减速度 Decel = %f\n", Decel);

ret = ZAux_Direct_SetDpos( handle,  0,  0);//轴指令位置清 0
commandCheckHandler("ZAux_Direct_SetDpos", ret);//判断指令是否执行成功
ret = ZAux_Direct_SetMpos( handle,  0,  0);//编码器反馈位置清 0
commandCheckHandler("ZAux_Direct_SetMpos", ret);//判断指令是否执行成功

ZAux_Trigger(handle);//示波器触发函数
ret =ZAux_Direct_Single_Move( handle,  0,  500);
printf("驱动轴 0 走 500units\n");
commandCheckHandler("ZAux_Direct_SetDpos", ret);//判断指令是否执行成功
Sleep(1000);

ret = ZAux_Direct_GetDpos(handle, 0, &dpos);        //获取 dpos
commandCheckHandler("ZAux_Direct_GetDpos", ret) ;//判断指令是否执行成功
printf("1 秒后轴 0 轴指令位置 dpos = %f\n", dpos);

ret = ZAux_Direct_GetMpos(handle, 0, &DposValue);
commandCheckHandler("ZAux_Direct_GetMpos", ret) ;//判断指令是否执行成功
printf("1 秒后轴 0 编码器反馈位置 Mpos = %f\n", DposValue);

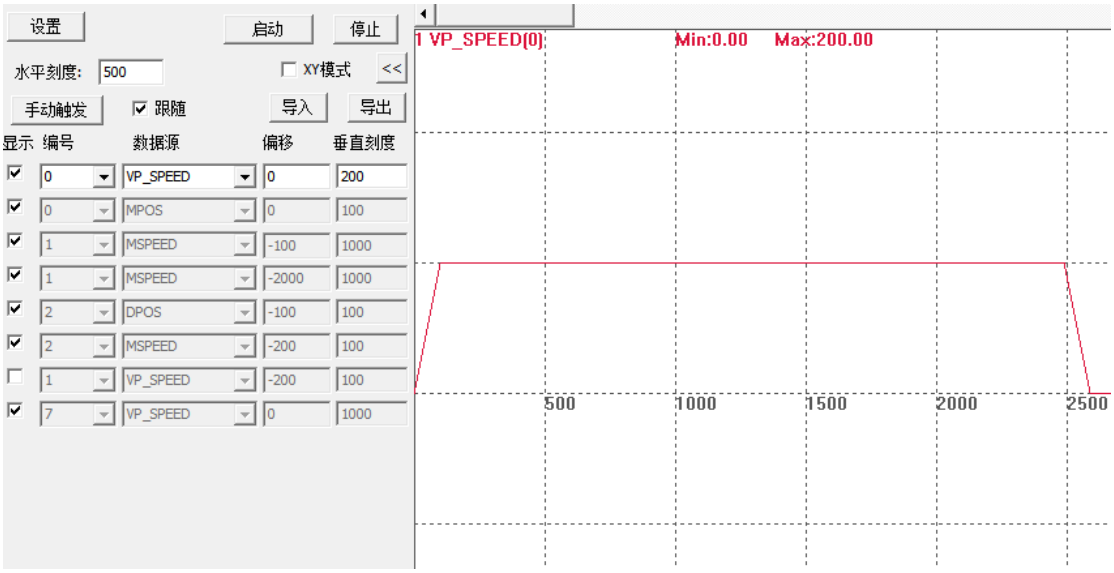
ret = ZAux_Direct_GetIfIdle(handle, 0, &ifIdle);    //获取当前轴是否空闲
commandCheckHandler("ZAux_Direct_GetIfIdle", ret) ;//判断指令是否执行成功
printf("1 秒后轴 0 运动状态 Idle: %s\n", ifIdle ? "停止" : "运动中");

ret = ZAux_Direct_GetAxisStatus(handle, 0, &status);  //获取轴状态
commandCheckHandler("ZAux_Direct_GetAxisStatus", ret) ;//判断指令是否执行成功
printf("轴状态 AXISSTATUS：%x h\n", status);

Sleep(2000);
ret = ZAux_Close(handle);     //关闭连接
commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
printf("connection closed!\n");
```

```
    handle = NULL;
    return 0;
}
```

输出波形图：



5.1.3-2 轴基本参数的设置获取（带起始速度与 S 曲线）

```
// test1.cpp : 定义控制台应用程序的入口点。
//
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s return code is %d\n", command, ret);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
```

```
char *ip_addr = (char *)"127.0.0.1";                //控制器 IP 地址
ZMC_HANDLE handle = NULL;                          //连接句柄
int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
if (ERR_SUCCESS != ret)
{

    printf("connect controller failed!\n");
    handle = NULL;
    getchar();
    return -1;
}
printf("控制器连接成功! \n");

ZAux_Direct_SetAtype(handle, 0, 1);   //设置轴 0 轴类型为 1
ZAux_Direct_SetUnits(handle, 0, 100); //设置轴 0 脉冲当量为 100
ZAux_Direct_SetSpeed(handle, 0, 200); //设置轴 0 速度为 200units/s
ZAux_Direct_SetAccel(handle, 0, 2000); //设置轴 0 加速度为 2000units/s/s
ZAux_Direct_SetDecel(handle, 0, 2000); //设置轴 0 减速度为 2000units/s/s
ZAux_Direct_SetLspeed(handle, 0, 100); //设置轴 0 起始速度为 20units/s
ZAux_Direct_SetSramp(handle, 0, 200);//设置轴 0 S 曲线时间为 200ms
float dpos;
int ifIdle;
int status;
float DposValue;

int Atype;
float Units;
float Speed;
float Accel;
float Decel;
float Lspeed;
float Sramp;

ret =ZAux_Direct_GetAtype(handle, 0, &Atype);      //获取轴 0 轴类型
commandCheckHandler("ZAux_Direct_GetAtype", ret);//判断指令是否执行成功
printf("轴 0 轴类型 Atype = %d\n", Atype);
```

```
ret =ZAux_Direct_GetUnits(handle, 0, &Units); //获取轴 0 脉冲当量
commandCheckHandler("ZAux_Direct_GetUnits", ret);//判断指令是否执行成功
printf("轴 0 脉冲当量 Units = %f\n", Units);

ret =ZAux_Direct_GetSpeed(handle, 0, &Speed); //获取轴 0 速度
commandCheckHandler("ZAux_Direct_GetSpeed", ret);//判断指令是否执行成功
printf("轴 0 速度 Speed = %f\n", Speed);

ret =ZAux_Direct_GetAccel(handle, 0, &Accel); //获取轴 0 加速度
commandCheckHandler("ZAux_Direct_GetAccel", ret);//判断指令是否执行成功
printf("轴 0 加速度 Accel = %f\n", Accel);

ret =ZAux_Direct_GetDecel(handle, 0, &Decel); //获取轴 0 减速度
commandCheckHandler("ZAux_Direct_GetDecel", ret);//判断指令是否执行成功
printf("轴 0 减速度 Decel = %f\n", Decel);

ret =ZAux_Direct_GetLspeed(handle, 0, &Lspeed); //获取轴 0 起始速度
commandCheckHandler("ZAux_Direct_GetLspeed", ret);//判断指令是否执行成功
printf("轴 0 起始速度 Lspeed = %f\n", Lspeed);

ret =ZAux_Direct_GetSramp(handle, 0, &Sramp); //获取轴 0 S 曲线时间
commandCheckHandler("ZAux_Direct_GetDecel", ret);//判断指令是否执行成功
printf("轴 0 S 曲线时间 Sramp = %f\n", Sramp);

ret = ZAux_Direct_SetDpos( handle,  0,  0);//轴指令位置清 0
commandCheckHandler("ZAux_Direct_SetDpos", ret);//判断指令是否执行成功
ret = ZAux_Direct_SetMpos( handle,  0,  0);//编码器反馈位置清 0
commandCheckHandler("ZAux_Direct_SetMpos", ret);//判断指令是否执行成功

ZAux_Trigger(handle);//示波器触发函数
ret =ZAux_Direct_Single_Move( handle,  0,  500);
printf("驱动轴 0 走 500units\n");
commandCheckHandler("ZAux_Direct_SetDpos", ret);//判断指令是否执行成功
Sleep(1000);

ret = ZAux_Direct_GetDpos(handle, 0, &dpos);        //获取 dpos
```

```
    commandCheckHandler("ZAux_Direct_GetDpos", ret) ;//判断指令是否执行成功
    printf("1 秒后轴 0 轴指令位置 dpos = %f\n", dpos);

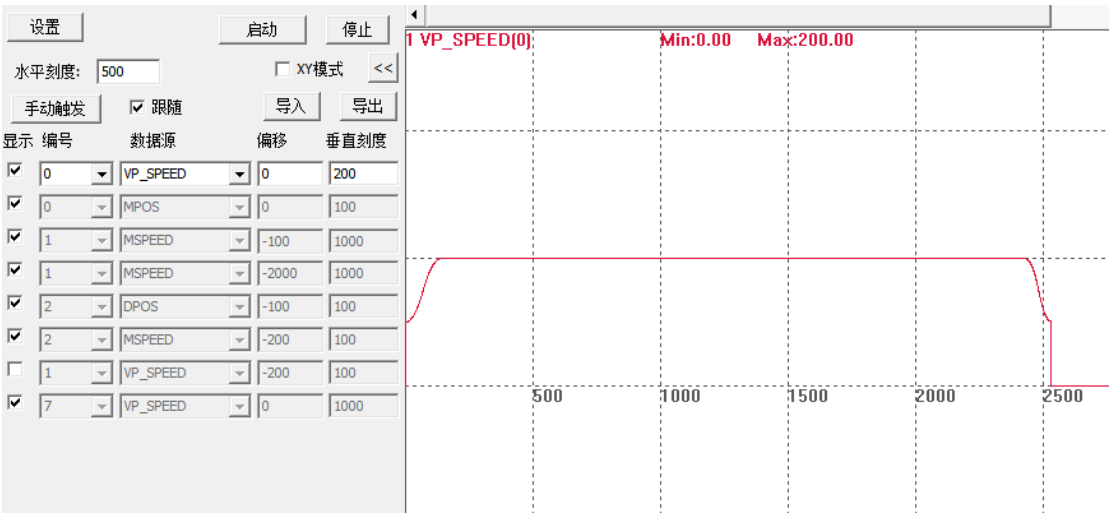
    ret = ZAux_Direct_GetMpos(handle, 0, &DposValue);
    commandCheckHandler("ZAux_Direct_GetMpos", ret) ;//判断指令是否执行成功
    printf("1 秒后轴 0 编码器反馈位置 Mpos = %f\n", DposValue);

    ret = ZAux_Direct_GetIfIdle(handle, 0, &ifIdle);   //获取当前轴是否空闲
    commandCheckHandler("ZAux_Direct_GetIfIdle", ret) ;//判断指令是否执行成功
    printf("1 秒后轴 0 运动状态 Idle: %s\n", ifIdle ? "停止" : "运动中");

    ret = ZAux_Direct_GetAxisStatus(handle, 0, &status);  //获取轴状态
    commandCheckHandler("ZAux_Direct_GetAxisStatus", ret) ;//判断指令是否执行成功
    printf("轴状态 AXISSTATUS：%x h\n", status);

    Sleep(2000);
    ret = ZAux_Close(handle);     //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

输出波形图：



5.2 轴其他参数

5.2.1 指令列表

轴参数：

指令	说明
ZAux_GetModbusDpos	快速读取多个当前轴的 DPOS（modbus）
ZAux_GetModbusMpos	快速读取多个当前轴的 MPOS（modbus）
ZAux_GetModbusCurSpeed	快速读取多个轴运动时的当前速度（modbus）
ZAux_Direct_SetAxisAddress	设置轴地址
ZAux_Direct_GetAxisAddress	读取轴地址
ZAux_Direct_SetInvertStep	设置脉冲输出模式
ZAux_Direct_GetInvertStep	读取脉冲输出模式
ZAux_Direct_SetMaxSpeed	设置脉冲输出最高频率
ZAux_Direct_GetMaxSpeed	读取脉冲输出最高频率
ZAux_Direct_GetMtype	读取当前正在运动指令类型
ZAux_Direct_GetNtype	读取当前正在运动的下一条运动指令类型
ZAux_Direct_SetOffpos	设置修改偏移相对位置
ZAux_Direct_GetOffpos	读取修改偏移相对位置
ZAux_Direct_Defpos	绝对位置偏移。定义 DPOS,不建议使用
ZAux_Direct_GetFe	读取随动误差
ZAux_Direct_SetFeLimit	设置最大允许的随动误差值(预留)
ZAux_Direct_GetFeLimit	读取最大允许的随动误差值(预留)
ZAux_Direct_SetFeRange	设置报警时随动误差值(预留)
ZAux_Direct_GetFeRange	读取报警时的随动误差值(预留)
ZAux_Direct_SetAxisMode	设置 connect 运动保持参数

5.2.2 重点说明

指令列表主要把其余的所有轴参数状态设置分类，规划分为参数类以及状态类，比较基本的参数状态获取，补足了各种不同情况下，获取相关信息。如 **ZAux_GetModbusDpos**，**ZAux_GetModbusMpos**，**ZAux_GetModbusCurSpeed** 三个函数获取的方式，是通过获取寄存器 Modbus_IEEE 的值来批量获取的。

5.2.2-1 轴地址说明

1. ZCAN 扩展轴时，扩展板上带 8 位拨码开关（V1.3 以上硬件版本）。

受总线带宽限制，ZCAN 扩展轴不要设置超过 2 个。

必须先用 **ZAux_Direct_SetAxisAddress** 设置轴地址，再用 **ZAux_Direct_SetAtype** 设置 ZCAN 扩展轴类型，修改后必须重新用 **ZAux_Direct_SetAtype** 设置轴类型。见例一。

位 1-4	CAN 地址拨码，组合值 0-15
位 5-6	CAN 速度拨码，不同组合值速度不同
位 7	特殊功能预留
位 8	120 欧姆电阻拨码，拨 ON 时电阻接通

规则：

ZAux_Direct_SetAxisAddress(控制器连接句柄, 轴号,(32*0)+ID)//扩展板的本地轴接口 0

ZAux_Direct_SetAxisAddress(控制器连接句柄, 轴号,(32*1)+ID)//扩展板的本地轴接口 1

2. 总线驱动器映射轴号，将连接的驱动器按编号一一映射。

驱动器编号根据接线顺序排列，编号从 0 开始到 EtherCAT 驱动器个数减 1。

驱动器编号与设备号不同，设备号包括 ECAT 接口所有连接的设备，驱动器编号只算连接的驱动器。

必须先设置轴地址，再设置 ECAT 轴的轴类型（**ZAux_Direct_SetAxisAddress**），修改后必须重新设置轴类型（**ZAux_Direct_SetAxisAddress**）。见例二。

位 0-15	驱动器编号加 1，0-自动指定
位 16-31	SLOT 编号（多槽位时）

规则：

ZAux_Direct_SetAxisAddress (控制器连接句柄, 轴号,(槽位号<<16)+驱动器编号+1)

3. 本地脉冲轴号重映射，4 系列控制器支持本地脉冲或编码器轴号重新映射，固件 160608 以上版本支持。

重新映射时注意先把原本地脉冲轴设置为虚拟轴。修改后必须重新设置轴类型（**ZAux_Direct_SetAtype**）。见例四。

位 0-15	映射的本地脉冲轴号
位 16-31	高 16 位全设为 1（等同于十进制下高 16 位= -1）

规则：

注：要把控制器物理接口轴 0 轴号重映射为轴 3，则轴 3 为目标轴号，轴 0 为重映射的轴号。

ZAux_Direct_SetParam(控制器连接句柄，“ATYPE”，重映射的轴号 0，0)

ZAux_Direct_SetParam(控制器连接句柄，“ATYPE”，目标轴号 3，0)

ZAux_Direct_SetAxisAddress (控制器连接句柄，目标轴号 3,(-1<<16) +重映射的轴号 0)

ZAux_Direct_SetParam(控制器连接句柄，“ATYPE”，目标轴号 3，1 或 7)

5.2.2-2 运动类型

在正运动控制卡中，每种运动都有不同运动类型，如走插补直线是一种运动类型，走圆心画弧又是另一种运动类型，这时候则可以通过调用 **ZAux_Direct_GetMtype** 函数来获取当前运动的运动类型，也可以通过调用 **ZAux_Direct_GetNtype** 函数来获取下条运动的运动类型。

运动类型表如下：

MTYPE	运动指令类型
0	IDLE(没有运动)
1	MOVE（单轴直线或者插补直线运动）
2	MOVEABS（绝对值单轴直线或者绝对插补直线运动）
3	MHELICAL（圆心螺旋运动）
4	MOVECIRC（圆弧插补）
5	MOVEMODIFY（修改运动位置）
6	MOVESP（SP 速度的单轴直线或者 SP 速度的插补直线运动）
7	MOVEABSSP（SP 速度的绝对值单轴直线/SP 速度的绝对插补直线运动）
8	MOVECIRCSP（SP 速度的圆弧插补）
9	MHELICALSP（SP 速度的圆心螺旋运动）
10	FORWARD，VMOVE(1)（正向持续运动）
11	REVERSE，VMOVE(-1)（负向持续运动）
12	DATUMING (回零运动中)
13	CAM (凸轮表运动)
14	FWD_JOG (映射正向 JOG 运动)
15	REV_JOG (映射负向 JOG 运动)
20	CAMBOX (跟随凸轮表运动)
21	CONNECT (同步运动)
22	MOVELINK (自动凸轮运动)
23	CONNPATH (同步运动 2，矢量类型的)
25	MOVESLINK（自动凸轮运动 2）
26	MOVESPIRAL(渐开线圆弧)
27	MECLIPSE，MECLIPSEABS，MECLIPSESP，MECLIPSEABSSP（椭圆运动）
28	MOVE_OP/MOVE_OP2 ， MOVE_TABLE ， MOVE_PWM ，MOVE_TASKMOVE_PARA，MOVE_ASYNMOVE，MOVE_AOUT（缓冲

	IO/缓冲寄存器操作等）
29	MOVE_DELAY，MOVE_WAIT，MOVE_SYNMOVE（缓冲延时）
31	MSPHERICAL，MSPHERICALSP（空间圆弧）
32	MOVE_PT（单位时间内运动距离）
33	CONNFRAME（机械手逆解运动）
34	CONNREFRAME（机械手正解运动）

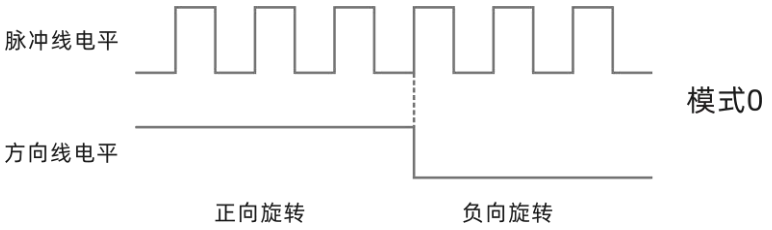
5.2.2-3 脉冲模式

脉冲方向模式(PUL/DIR):

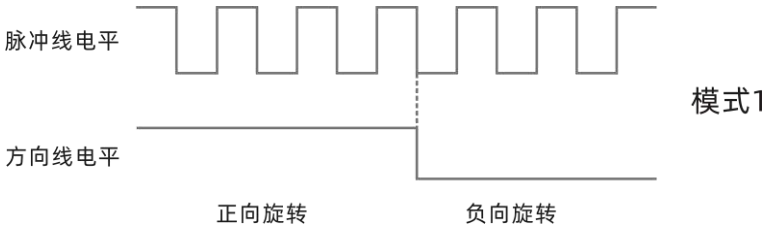
PUL+、PUL-脉冲线输出指令脉冲数，脉冲数对应电机运行距离，而脉冲频率对应电机运行速度。

DIR+、DIR-方向线输出方向信号，该信号的不同电平对应电机不同的转动方向。此种模式在驱动器中最多。

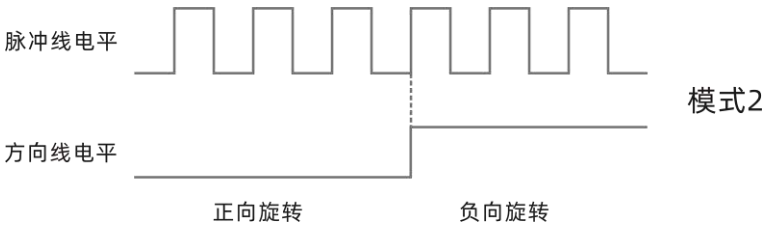
对应 **ZAux_Direct_SetInvertStep**(句柄,轴号,0)



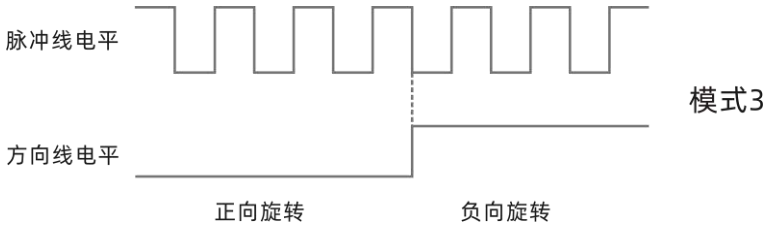
对应 **ZAux_Direct_SetInvertStep**(句柄,轴号,1)



对应 **ZAux_Direct_SetInvertStep**(句柄,轴号,2)



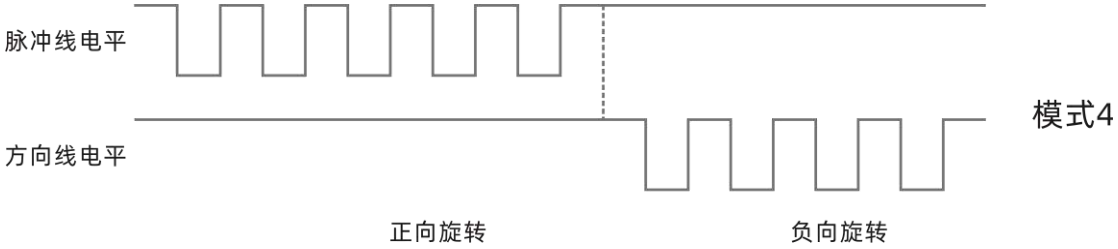
对应 **ZAux_Direct_SetInvertStep**(句柄,轴号,3)



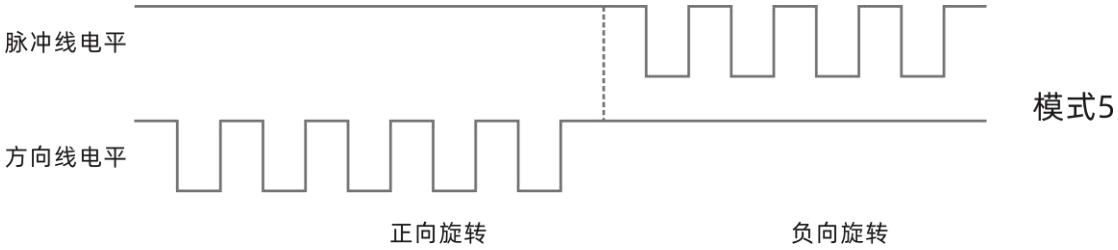
双脉冲模式(CW/CCW):

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

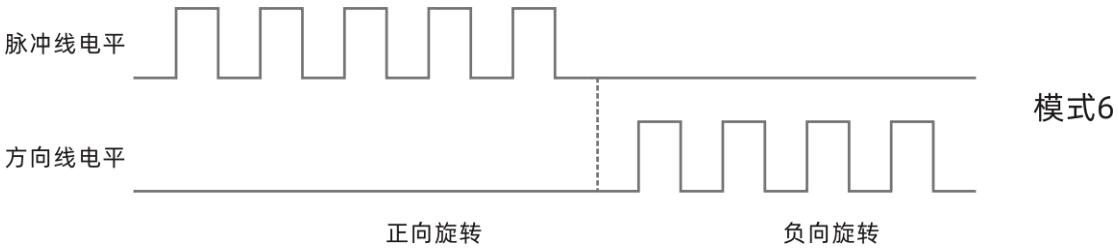
对应 **ZAux_Direct_SetInvertStep** (句柄, 轴号, 4)



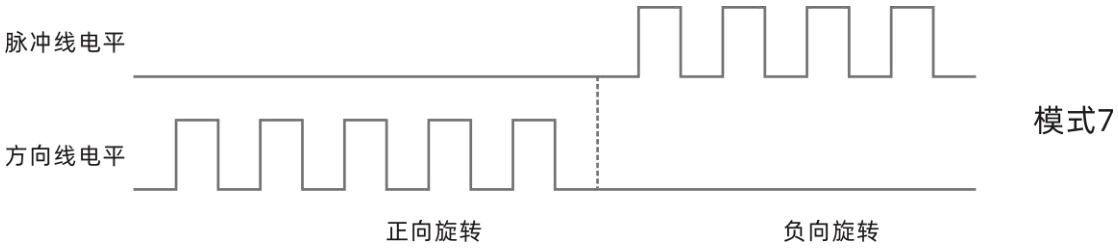
对应 **ZAux_Direct_SetInvertStep** (句柄, 轴号, 5)



对应 **ZAux_Direct_SetInvertStep** (句柄, 轴号, 6)



对应 **ZAux_Direct_SetInvertStep** (句柄, 轴号, 7)

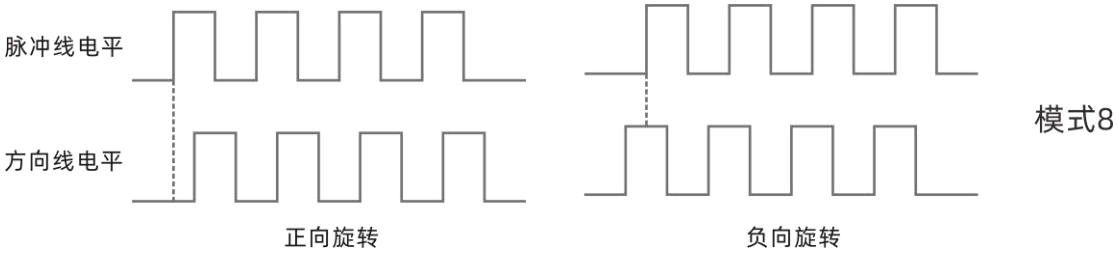


正交脉冲模式(A/B 相):

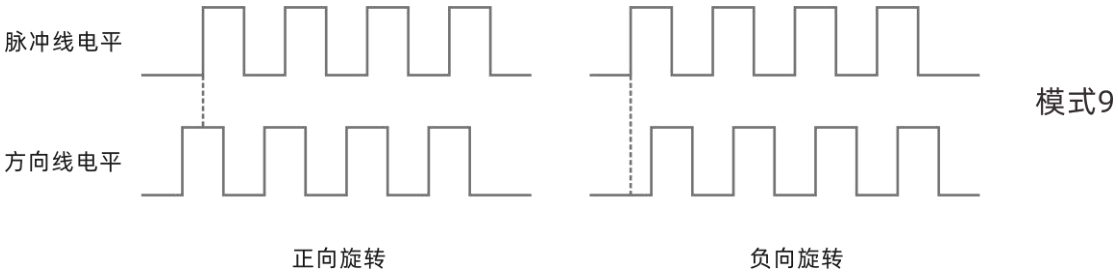
正交脉冲指两个相互独立的相同脉冲信号，正方向脉冲信号在负方向脉冲信号之前产生，二者相位相差 90 度，此时为正向旋转；负方向脉冲信号在正方向脉冲信号之前产生，二者相位相差 90 度，此时为负向旋转。

通过两个脉冲之间的相位差来达到计数或编码的作用。

对应 ZAux_Direct_SetInvertStep(句柄, 轴号,8)



对应 ZAux_Direct_SetInvertStep(句柄, 轴号,9)



5.2.3 例程

以下例程会把本章节所讲的指令全部有简单应用的示例。

5.2.3-1 本地脉冲轴号重映射

本例程，把轴 0 的轴号重映射成轴 3。

```
// test1.cpp : 定义控制台应用程序的入口点。
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s return code is %d\n", command, ret);
    }
}
```

```
int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"192.168.0.181";                //控制器 IP 地址，需要硬件接线才可跑出手册上的波形图，不可用仿真器
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("控制器连接成功! \n");
    int i=0;
    //轴地址清 0 复位
    for (i=0;i<10;i++)
    {
        ret =ZAux_Direct_SetAtype(handle,i,0);//设置轴地址之前需把 ATYPE 清 0
        commandCheckHandler("ZAux_Direct_SetParam", ret) ;//判断指令是否执行成功
        ret =ZAux_Direct_SetAxisAddress(handle,i,0);//设置轴地址 Axis_Address 清 0
        commandCheckHandler("ZAux_Direct_SetParam", ret) ;//判断指令是否执行成功
    }
    int axis[2]={0,3};
    for (i=0;i<2;i++)
    {
    ret = ZAux_Direct_SetParam(handle,"ATYPE",axis[i],1);//使用 SetParam 的方式设置轴列表，ATYPE=1
    commandCheckHandler("ZAux_Direct_SetParam", ret) ;//判断指令是否执行成功

    ret = ZAux_Direct_SetParam(handle,"DPOS",axis[i],0);//使用 SetParam 的方式设置轴列表，dpos 清 0
    commandCheckHandler("ZAux_Direct_SetParam", ret) ;//判断指令是否执行成功

    ret = ZAux_Direct_SetParam(handle,"ACCEL",axis[i],1000);//使用 SetParam 的方式设置轴列表，加速度为 1000
```

```
    commandCheckHandler("ZAux_Direct_SetParam", ret) ;//判断指令是否执行成功

    ret = ZAux_Direct_SetParam(handle,"DECEL",axis[i],1000);//使用 SetParam 的方式设置轴列表，减速度为 1000
    commandCheckHandler("ZAux_Direct_SetParam", ret) ;//判断指令是否执行成功

    ret = ZAux_Direct_SetParam(handle,"SPEED",axis[i],1000);//使用 SetParam 的方式设置轴列表，速度为 1000
    commandCheckHandler("ZAux_Direct_SetParam", ret) ;//判断指令是否执行成功

    }
    ret = ZAux_Direct_SetParam(handle,"DPOS",5,0);//使用 SetParam 的方式设置轴 5，dpos 清 0
    commandCheckHandler("ZAux_Direct_SetParam", ret) ;//判断指令是否执行成功


    ret = ZAux_Direct_SetParam(handle,"ATYPE",5,6);//使用 SetParam 的方式设置轴 5，轴类型设置为 6，编码器轴，也可以用 setAtype 函数
    commandCheckHandler("ZAux_Direct_SetParam", ret) ;//判断指令是否执行成功


    float len[2]={100,100};
    int AxisAddress0;
    int AxisAddress3;
    ret =ZAux_Direct_GetAxisAddress(handle,0,&AxisAddress0);//获取轴 0 地址
    commandCheckHandler("ZAux_Direct_GetAxisAddress", ret) ;//判断指令是否执行成功
    printf("修改前轴 0 轴地址为 Axis_Address：%d\n",AxisAddress0);

    ret =ZAux_Direct_GetAxisAddress(handle,3,&AxisAddress3);//获取轴 3 地址
    commandCheckHandler("ZAux_Direct_GetAxisAddress", ret) ;//判断指令是否执行成功
    printf("修改前轴 3 轴地址为 Axis_Address：%d\n",AxisAddress3);
    printf("*****************************************************\n");
    ZAux_Trigger(handle);//触发 RTSys 软件自带的示波器
    float IDLE;
    ZAux_Direct_Single_Move(handle,0,100);//轴 0 运动 100
    while (1)//等待轴 0 运动完成
    {
```

```
        Sleep(100);
        ZAux_Direct_GetParam(handle,"IDLE",0,&IDLE);
        if (IDLE<0)break;
    }
    Sleep(400);
    ZAux_Direct_Single_Move(handle,3,100);//轴 3 运动 100
    while (1)//等待轴 3 运动完成
    {
        Sleep(100);
        ZAux_Direct_GetParam(handle,"IDLE",3,&IDLE);
        if (IDLE<0)break;
    }

    //{要把物理接口轴 0 的轴号重映射为轴 3，则轴 3 为要修改的轴号，轴 0 为重映射的轴号。

    ret = ZAux_Direct_SetParam(handle,"ATYPE",0,0);//使用 SetParam 的方式重映射轴 0，
轴类型设置为虚拟轴，也可以用 setAtype 函数
    commandCheckHandler("ZAux_Direct_SetParam", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetParam(handle,"ATYPE",3,0);//使用 SetParam 的方式把目标轴号 3，
轴类型设置为虚拟轴，也可以用 setAtype 函数
    commandCheckHandler("ZAux_Direct_SetParam", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetAxisAddress(handle,3,(-1<<16)+0);//把轴 0 的轴号重新映射为轴 3
    commandCheckHandler("ZAux_Direct_SetAxisAddress", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetParam(handle,"ATYPE",3,1);//目标轴号轴类型设置为 1
    commandCheckHandler("ZAux_Direct_SetParam", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_GetAxisAddress(handle,0,&AxisAddress0);//获取轴 0 地址
    commandCheckHandler("ZAux_Direct_GetAxisAddress", ret) ;//判断指令是否执行成功
    printf("修改后轴 0 轴地址为 Axis_Address：%d\n",AxisAddress0);

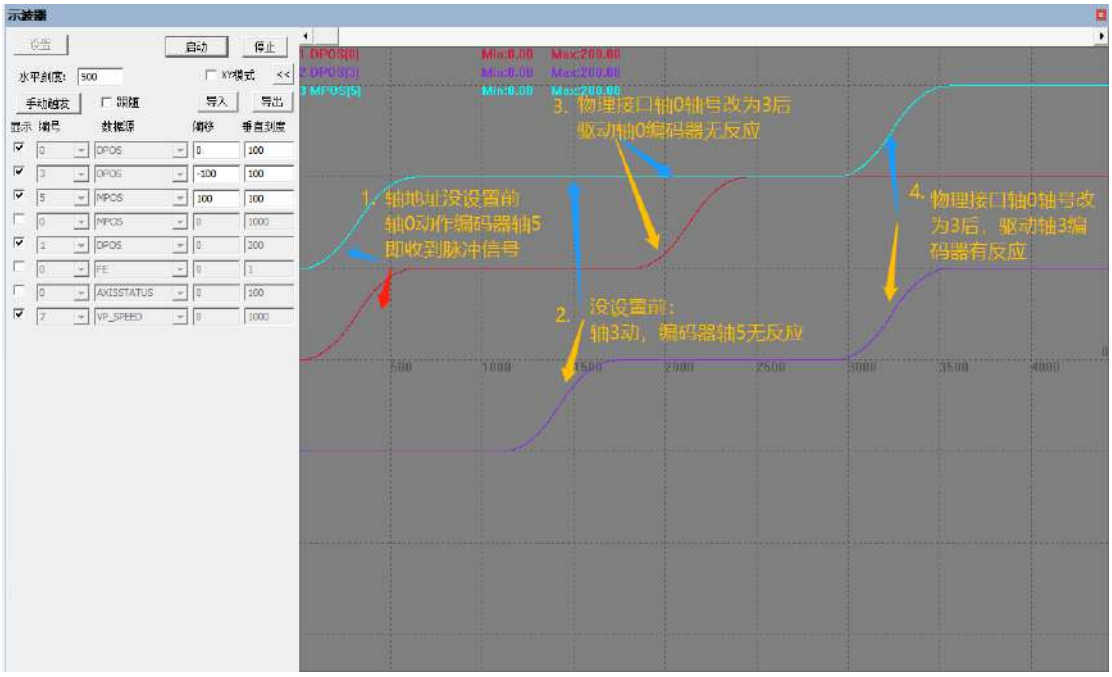
    ret =ZAux_Direct_GetAxisAddress(handle,3,&AxisAddress3);//获取轴 3 地址
    commandCheckHandler("ZAux_Direct_GetAxisAddress", ret) ;//判断指令是否执行成功
    printf("修改后轴 3 轴地址为 Axis_Address：%d\n",AxisAddress3);
```

```
    //例：要把物理接口轴 0 的轴号重映射为轴 3，则轴 3 为要修改的轴号，轴 0 为重映射的轴号。}

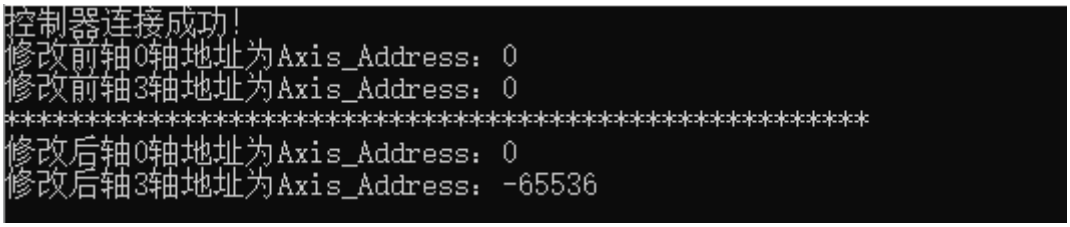
    ZAux_Direct_Single_Move(handle,0,100);//轴 0 运动 100
    while (1)
    {
        Sleep(100);
        ZAux_Direct_GetParam(handle,"IDLE",0,&IDLE);
        if (IDLE<0)break;
    }
    Sleep(400);
    ZAux_Direct_Single_Move(handle,3,100);//轴 3 运动 100

    Sleep(2000);
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

由如下运动波形图可观察出轴 0 的轴号变更为了 3。



程序结果输出：



5.2.3-2 参数获取及配置

```
// test1.cpp : 定义控制台应用程序的入口点。
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s return code is %d\n", command, ret);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";                //控制器 IP 地址
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("connect controller failed!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
```

```
printf("控制器连接成功！\n");

ZAux_Direct_SetDpos(handle, 1, 100); //设置轴 1 dpos=100
ZAux_Direct_SetDpos(handle, 2, 200); //设置轴 2 dpos=200
ZAux_Direct_SetSpeed(handle, 0, 300); //设置轴 0 速度为 300units/s
ZAux_Direct_SetSpeed(handle, 1, 400); //设置轴 1 速度为 400units/s
ZAux_Direct_SetSpeed(handle, 2, 500); //设置轴 2 速度为 500units/s

float Mpos[3]={0};
float Dpos[3]={0};
float Speed[3]={0};
int MaxSpeed;
int Invert_Step;
//{Invert_Step 用法，
//0-3 脉冲方向模式，脉冲线+方向线
//4-7 双脉冲方式（或称 CW/CCW），正向脉冲线+负向脉冲线
//8-9 AB 输出(部分控制器定制)
printf("************************************************\n");
ZAux_Direct_GetInvertStep(handle, 0, &Invert_Step);//获取轴 0 脉冲模式
commandCheckHandler("ZAux_GetModbusMpos", ret) ;//判断指令是否执行成功
printf("轴 0 脉冲模式 Invert_Step(0)：%d\n",Invert_Step);

ZAux_Direct_SetInvertStep(handle, 0, 1);//设置轴 0 脉冲模式为 1
commandCheckHandler("ZAux_GetModbusMpos", ret) ;//判断指令是否执行成功

ZAux_Direct_GetInvertStep(handle, 0, &Invert_Step);//获取轴 0 脉冲模式
commandCheckHandler("ZAux_GetModbusMpos", ret) ;//判断指令是否执行成功
printf("轴 0 脉冲模式 Invert_Step(0)（修改后）：%d\n",Invert_Step);

ZAux_Direct_SetInvertStep(handle, 0, 0);//设置轴 0 脉冲模式为 0（默认模式）
commandCheckHandler("ZAux_GetModbusMpos", ret) ;//判断指令是否执行成功
printf("************************************************\n");
//Invert_Step 用法}

//{max_speed 简单应用（speed 速度最大不能超过该值）
ret=ZAux_Direct_GetMaxSpeed(handle, 0, &MaxSpeed);//获取轴 0 最大脉冲输出频率
```

```
    commandCheckHandler("ZAux_Direct_GetMaxSpeed", ret) ;//判断指令是否执行成功
    printf("轴 0 最大脉冲输出频率 Maxspeed：轴 0=%d\n",MaxSpeed);
    ret=ZAux_Direct_SetMaxSpeed(handle, 0, MaxSpeed+100);//修改最大脉冲频率
    commandCheckHandler("ZAux_Direct_SetMaxSpeed", ret) ;//判断指令是否执行成功

    ret=ZAux_Direct_GetMaxSpeed(handle, 0, &MaxSpeed);
    commandCheckHandler("ZAux_Direct_GetMaxSpeed", ret) ;//判断指令是否执行成功
    printf("轴 0 最大脉冲输出频率 Maxspeed（修改后）：轴 0=%d\n",MaxSpeed);
    ret=ZAux_Direct_SetMaxSpeed(handle, 0, MaxSpeed-100);//改回最大脉冲频率
    commandCheckHandler("ZAux_Direct_SetMaxSpeed", ret) ;//判断指令是否执行成功
    //max_speed 简单应用（speed 速度最大不能超过该值）}

    //modbus 方式获取 DPOS,MPOS,VP_SPEED
    printf("**********************************************\n");
    ret=ZAux_GetModbusMpos( handle, 3, Mpos);//读取轴 0-轴 2 的 MPOS
    commandCheckHandler("ZAux_GetModbusMpos", ret) ;//判断指令是否执行成功
    printf("反馈位置 MPOS：轴 0=%f 轴 1=%f 轴 2=%f\n",Mpos[0],Mpos[1],Mpos[2]);

    ret=ZAux_GetModbusDpos(handle, 3, Dpos);
    commandCheckHandler("ZAux_GetModbusDpos", ret) ;//判断指令是否执行成功
    printf("规划位置 DPOS：轴 0=%f 轴 1=%f 轴 2=%f\n",Dpos[0],Dpos[1],Dpos[2]);

    ret=ZAux_GetModbusCurSpeed(handle, 3, Speed);//读取轴 0-轴 2 的 VP_SPEED（即轴运动
时，控制器发送的脉冲速度，不一定为理想
    commandCheckHandler("ZAux_GetModbusCurSpeed", ret) ;//判断指令是否执行成功
    printf("当前运动的规划速度 VP_SPEED（运动前）：轴 0=%f 轴 1=%f 轴
2=%f\n\n",Speed[0],Speed[1],Speed[2]);
    //modbus 方式获取 DPOS,MPOS,VP_SPEED}

    //{运动
    ret=ZAux_Direct_Single_Move(handle,0,200);//驱动轴 0 走 200
    commandCheckHandler("ZAux_Direct_Single_Move", ret) ;//判断指令是否执行成功
    ret=ZAux_Direct_Single_Move(handle,1,200);//驱动轴 1 走 200
    commandCheckHandler("ZAux_Direct_Single_Move", ret) ;//判断指令是否执行成功
    ret=ZAux_Direct_Single_Move(handle,2,200);//驱动轴 2 走 200
    commandCheckHandler("ZAux_Direct_Single_Move", ret) ;//判断指令是否执行成功
    Sleep(200);
```

```
    //运动}

    //{重新获取信息
    ret=ZAux_GetModbusMpos( handle, 3, Mpos);//读取轴 0-轴 2 的 MPOS
    commandCheckHandler("ZAux_GetModbusMpos", ret) ;//判断指令是否执行成功
    printf("反馈位置 MPOS（运动后）：轴 0=%f 轴 1=%f 轴
2=%f\n",Mpos[0],Mpos[1],Mpos[2]);

    ret=ZAux_GetModbusDpos(handle, 3, Dpos);
    commandCheckHandler("ZAux_GetModbusDpos", ret) ;//判断指令是否执行成功
    printf("规划位置 DPOS（运动后）：轴 0=%f 轴 1=%f 轴
2=%f\n",Dpos[0],Dpos[1],Dpos[2]);

    ret=ZAux_GetModbusCurSpeed(handle, 3, Speed);//读取轴 0-轴 2 的 VP_SPEED（即轴运动
时，控制器发送的脉冲速度
    commandCheckHandler("ZAux_GetModbusCurSpeed", ret) ;//判断指令是否执行成功
    printf("当前运动的规划速度 VP_SPEED（运动后）：轴 0=%f 轴 1=%f 轴
2=%f\n",Speed[0],Speed[1],Speed[2]);
    printf("************************************************\n");
    //重新获取信息}

    //{运动类型获取
    //值为 0：无运动，大于 0 有运动，具体说明查阅 PC 编程手册 5.2.2-2
    int Mtype=0;
    int Ntype=0;
    ret=ZAux_Direct_GetMtype(handle, 0,&Mtype);
    commandCheckHandler("ZAux_Direct_GetMtype", ret) ;//判断指令是否执行成功
    printf("轴 0 当前运动类型 Mtype：轴 0=%d \n",Mtype);

    ret=ZAux_Direct_GetNtype(handle, 0,&Ntype);
    commandCheckHandler("ZAux_Direct_GetNtype", ret) ;//判断指令是否执行成功
    printf("轴 0 当前正在运动的下一条运动指令类型 Ntype：=%d \n",Ntype);

    printf("************************************************\n");
    //运动类型获取}
    Sleep(2000);
    ret = ZAux_Close(handle);     //关闭连接
```

```
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

程序结果输出：

```
控制器连接成功!
************************************************
轴0脉冲模式Invert_Step(0)：0
轴0脉冲模式Invert_Step(0)（修改后）：1
************************************************
轴0最大脉冲输出频率Maxspeed：轴0=1000000
轴0最大脉冲输出频率Maxspeed（修改后）：轴0=1000100
************************************************
反馈位置MPOS：轴0=-900.000000 轴1=100.000000 轴2=200.000000
规划位置DPOS：轴0=-900.000000 轴1=100.000000 轴2=200.000000
当前运动的规划速度VP_SPEED（运动前）：轴0=0.000000 轴1=0.000000 轴2=0.000000

反馈位置MPOS（运动后）：轴0=-879.900024 轴1=120.099998 轴2=220.100006
规划位置DPOS（运动后）：轴0=-879.900024 轴1=120.099998 轴2=220.100006
当前运动的规划速度VP_SPEED（运动后）：轴0=202.000000 轴1=202.000000 轴2=202.000000
************************************************
轴0当前运动类型Mtype：轴0=1
轴0当前正在运动的下一条运动指令类型Ntype：=0
************************************************
```

5.2.3-3 轴坐标设置

```
// test1.cpp : 定义控制台应用程序的入口点。
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s return code is %d\n", command, ret);
    }
}
```

```
int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";                //控制器 IP 地址
    ZMC_HANDLE handle = NULL;                           //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("控制器连接成功! \n");

    ret =ZAux_Direct_SetAtype(handle,0,1);//设置轴类型
    commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetAccel(handle,0,500);//设置轴加速度
    commandCheckHandler("ZAux_Direct_SetAccel", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetDecel(handle,0,500);//设置轴减速度
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetSpeed(handle,0,200);//设置轴速度
    commandCheckHandler("ZAux_Direct_SetSpeed", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetDpos(handle,0,0);//设置轴 DPOS 清 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//判断指令是否执行成功
    float DPOS;
    float OFFPOS;
    ret =ZAux_Direct_GetDpos(handle,0,&DPOS);//获取轴 DPOS 值
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//判断指令是否执行成功
    printf("运动前轴 0 DPOS=%f\n",DPOS);

    float IDLE;
    ZAux_Direct_Single_Move(handle,0,200);//轴 0 运动 200
    while (1)//等待轴 0 运动完成
    {
        Sleep(100);
        ZAux_Direct_GetParam(handle,"IDLE",0,&IDLE);
        if (IDLE<0)break;
```

```
    }

    ret =ZAux_Direct_GetDpos(handle,0,&DPOS);//获取轴 0 的 DPOS 值
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//判断指令是否执行成功
    printf("运动完成轴 0 DPOS=%f\n",DPOS);

    ret =ZAux_Direct_SetOffpos(handle,0,800);//把轴 0 坐标相对偏移 800
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//判断指令是否执行成功

    //ret =ZAux_Direct_GetOffpos(handle,0,&OFFPOS);
    //commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//判断指令是否执行成功
    //printf("轴 0 偏移量 OFFPOS=%f\n",OFFPOS);

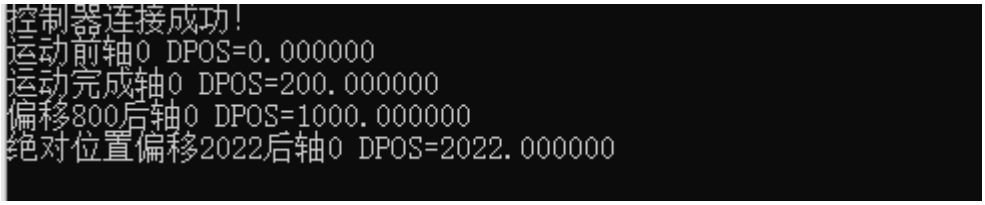
    ret =ZAux_Direct_GetDpos(handle,0,&DPOS);//获取轴 DPOS 值
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//判断指令是否执行成功
    printf("偏移 800 后轴 0 DPOS=%f\n",DPOS);

    ret =ZAux_Direct_Defpos(handle,0,2022);//把轴 0 坐标偏移为绝对位置 2022
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_GetDpos(handle,0,&DPOS);//获取轴 DPOS 值
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//判断指令是否执行成功
    printf("绝对位置偏移 2022 后轴 0 DPOS=%f\n",DPOS);

    Sleep(2000);
    ret = ZAux_Close(handle);     //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

程序结果输出：（只改变坐标，轴不动）



5.2.3-4 随动误差获取

对于伺服控制系统而言，在某些异常情况下，电机的实际位置可能与规划位置差距很大。这时通常存在一些危险情况，例如电机故障、编码器 A、B 相信号接反或断线、机械摩擦太大或者机械故障造成电机堵转等。为了及时检测这些情况，增强系统的安全性并延长设备使用寿命，运动控制器提供可以获取跟随误差超限的安全保护机制。

以下例程为获取轴 0 的随动误差。

```
// test1.cpp : 定义控制台应用程序的入口点。

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";                //控制器 IP 地址
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("控制器连接成功! \n");
```

```
    ret =ZAux_Direct_SetAtype(handle,0,5);//设置轴类型为 5，带编码器反馈的轴类型，因仿真器无编码器反馈接线，则会造成明显的随动误差
    commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetAccel(handle,0,500);//设置轴加速度
    commandCheckHandler("ZAux_Direct_SetAccel", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetDecel(handle,0,500);//设置轴减速度
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetSpeed(handle,0,200);//设置轴速度
    commandCheckHandler("ZAux_Direct_SetSpeed", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetDpos(handle,0,0);//设置轴 DPOS 清 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//判断指令是否执行成功

    float IDLE;
    ZAux_Direct_Single_Move(handle,0,200);//轴 0 运动 200
    while (1)//等待轴 0 运动完成
    {
        Sleep(100);
        ZAux_Direct_GetParam(handle,"IDLE",0,&IDLE);
        if (IDLE<0)break;
    }

    float FE;
    float DPOS;
    float MPOS;
    ret =ZAux_Direct_GetDpos(handle,0,&DPOS);//设置轴 DPOS 清 0
    commandCheckHandler("ZAux_Direct_GetDpos", ret) ;//判断指令是否执行成功
    printf("当前 DPOS=%f\n",DPOS);
    ret =ZAux_Direct_GetMpos(handle,0,&MPOS);//设置轴 DPOS 清 0
    commandCheckHandler("ZAux_Direct_GetMpos", ret) ;//判断指令是否执行成功
    printf("当前 MPOS=%f\n",MPOS);

    ret =ZAux_Direct_GetFe(handle,0,&FE);//设置轴 DPOS 清 0
    commandCheckHandler("ZAux_Direct_GetFe", ret) ;//判断指令是否执行成功
    printf("当前 FE=%f\n",FE);

    Sleep(2000);
    ret = ZAux_Close(handle);     //关闭连接
```

```
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

5.3 轴参数指令表

5.3.1 指令列表

指令	说明
ZAux_Direct_GetAllAxisPara	浮点型读全部轴参数状态
ZAux_Direct_GetAllAxisInfo	整型读全部轴参数状态
ZAux_Direct_SetParam	通用的轴参数修改函数
ZAux_Direct_GetParam	通用的轴参数读取函数

5.3.2 重点说明

运动控制参数表：

指令	描述	读写性	序号
Atype	轴类型	可读可写	0
Units	脉冲当量	可读可写	1
Accel	加速度	可读可写	2
Decel	减速度	可读可写	3
Speed	运行速度	可读可写	4
Creep	爬行速度	可读可写	5
Lspeed	起始速度	可读可写	6
Merge	连续插补开关	可读可写	7
Sramp	加减速曲线时间设置	可读可写	8
Dpos	轴指令规划位置	可读可写	9
Mpos	编码器反馈位置	可读可写	10
Endmove	当前运动目标位置	只读	11
Fs_Limit	正向软限位设置	可读可写	12
Rs_Limit	负向软限位设置	可读可写	13

Datum_In	映射原点输入	可读可写	14
Fwd_In	映射正限位输入	可读可写	15
Rev_In	映射负限位输入	可读可写	16
Idle	运动状态	只读	17
Loaded	缓冲是否为空状态位	只读	18
Mspeed	实际编码反馈速度	只读	19
Mtype	当前运动类型	只读	20
Ntype	下条运动类型	只读	21
Remain	当前运动剩余距离	只读	22
Vp_Speed	当前运动速度	只读	23
Axisstatus	轴状态	只读	24
Move_Mark	运动标号	可读可写	25
Move_Curmark	当前运动标号	只读	26
Vector_Buffered	当前运动剩余距离	只读	27
Axis_StopReason	轴停止原因	只读	28
Moves_Buffered	当前运动缓冲数	只读	29
Axis_Address	轴地址	可读可写	30
Axis_Enable	单轴使能	可读可写	31
Force_Speed	sp 运动速度	可读可写	32
Startmove_Speed	sp 运动开始速度	可读可写	33
Endmove_Speed	sp 运动结束速度	可读可写	34
Fastdec	快减减速度	可读可写	35
Addax_Axis	叠加轴号	只读	36
Link_Axis	连接轴号	只读	37
Corner_Mode	拐角模式	可读可写	38
Decel_Angle	拐角减速开始	可读可写	39
Stop_Angle	拐角减速结束	可读可写	40
Full_Sp_Radius	限速半径	可读可写	41
Splimit_Radius	限速值	可读可写	42
Zsmooth	倒角半径	可读可写	43
Vector_Moved	当前运动的距离	可读可写	44
Endmove_Buffer	缓冲最终位置	只读	45
Homewait	回零反找延时	可读可写	46
Fast_Jog	映射点动输入	可读可写	47
Fwd_Jog	映射正向 Jog 输入	可读可写	48
Rev_Jog	映射负向 Jog 输入	可读可写	49

Jogspeed	映射 Jog 速度	可读可写	50
Fhold_in	映射保持输入	可读可写	51
Fhspeed	保持速度	可读可写	52
Encoder	编码器原始值	只读	53
Encoder_Status	编码器状态	只读	54
PP_Step	编码器内部比例	可读可写	55
Reg_Inputs	锁存输入映射	可读可写	56
Mark	锁存触发	只读	57
MarkB	锁存 2 触发	只读	58
MarkC	锁存 3 触发	只读	59
MarkD	锁存 4 触发	只读	60
Reg_Pos	锁存位置	只读	61
Reg_PosB	锁存 2 位置	只读	62
Reg_PosC	锁存 3 位置	只读	63
Reg_PosD	锁存 4 位置	只读	64
Alm_In	映射报警输入	可读可写	65
Rep_Option	坐标循环模式	可读可写	66
Rep_Dist	坐标循环位置	可读可写	67
Invert_Step	脉冲模式设置	可读可写	68
Max_speed	脉冲频率限制	可读可写	69
Axis_Zset	精准输出设置	可读可写	70
Dac	总线轴模拟量控制	可读可写	71
ErrorMask	错误时操作	可读可写	72

以上运动参数列表状态的获取和设置，均可用指令 **ZAux_Direct_SetParam**、**ZAux_Direct_GetParam** 函数来设置或者获取，且设置获取时，字符串指令不区分大小写。

5.3.3 例程

5.3.3-1 字符串方式设置获取基本参数

```
// test1.cpp : 定义控制台应用程序的入口点。
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
```

```
void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s return code is %d\n", command, ret);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";                //控制器 IP 地址
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("connect controller failed!\n");
        handle = NULL;
        getchar();
        return -1;
    }
    printf("控制器连接成功! \n");

    ZAux_Direct_SetParam(handle,"ATYPE", 0, 1);    //设置轴 0 轴类型为 1,字符串指令不区分大小写
    ZAux_Direct_SetParam(handle,"Units", 0, 100); //设置轴 0 脉冲当量为 100,字符串指令不区分大小写
    ZAux_Direct_SetParam(handle,"SPEED", 0, 200); //设置轴 0 速度为 200units/s,字符串指令不区分大小写
    ZAux_Direct_SetParam(handle,"ACCEL",0, 2000); //设置轴 0 加速度为 2000units/s/s,字符串指令不区分大小写
    ZAux_Direct_SetParam(handle,"DECEL", 0, 2000); //设置轴 0 减速度为 2000units/s/s,字符串指令不区分大小写
    float dpos;
    float ifIdle;
    float status;
    float DposValue;
```

```
    float Atype;
    float Units;
    float Speed;
    float Accel;
    float Decel;

    ret =ZAux_Direct_GetParam(handle,"ATYPE", 0, &Atype);  //获取轴 0 轴类型为 1
    commandCheckHandler("ZAux_Direct_GetAtype", ret);//判断指令是否执行成功
    printf("轴 0 轴类型 Atype = %d\n", Atype);

    ret =ZAux_Direct_GetParam(handle, "Units",0, &Units); //获取轴 0 脉冲当量为 100
    commandCheckHandler("ZAux_Direct_GetUnits", ret);//判断指令是否执行成功
    printf("轴 0 脉冲当量 Units = %f\n", Units);

    ret =ZAux_Direct_GetParam(handle,"Speed", 0, &Speed); //获取轴 0 速度为
200units/s
    commandCheckHandler("ZAux_Direct_GetSpeed", ret);//判断指令是否执行成功
    printf("轴 0 速度 Speed = %f\n", Speed);

    ret =ZAux_Direct_GetParam(handle,"Accel", 0, &Accel); //获取轴 0 加速度为
2000units/s/s
    commandCheckHandler("ZAux_Direct_GetAccel", ret);//判断指令是否执行成功
    printf("轴 0 加速度 Accel = %f\n", Accel);

    ret =ZAux_Direct_GetParam(handle,"Decel", 0, &Decel); //获取轴 0 减速度为
2000units/s/s
    commandCheckHandler("ZAux_Direct_GetDecel", ret);//判断指令是否执行成功
    printf("轴 0 减速度 Decel = %f\n", Decel);

    ret = ZAux_Direct_SetParam( handle,"DPOS",  0,  0);//轴指令位置清 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret);//判断指令是否执行成功

    ret = ZAux_Direct_SetParam( handle, "MPOS", 0,  0);//编码器反馈位置位置清 0
    commandCheckHandler("ZAux_Direct_SetMpos", ret);//判断指令是否执行成功

    ZAux_Trigger(handle);//示波器触发函数
```

```
    ret =ZAux_Direct_Single_Move( handle,  0,  500);
    printf("驱动轴 0 走 500units\n");
    commandCheckHandler("ZAux_Direct_Single_Move", ret);//判断指令是否执行成功
    Sleep(1000);

    ret = ZAux_Direct_GetParam(handle,"DPOS", 0, &dpos);        //获取 dpos
    commandCheckHandler("ZAux_Direct_GetDpos", ret) ;//判断指令是否执行成功
    printf("1 秒后轴 0 轴指令位置 dpos = %f\n", dpos);

    ret = ZAux_Direct_GetParam(handle,"MPOS", 0, &DposValue);
    commandCheckHandler("ZAux_Direct_GetMpos", ret) ;//判断指令是否执行成功
    printf("1 秒后轴 0 编码器反馈位置 Mpos = %f\n", DposValue);

    ret = ZAux_Direct_GetParam(handle,"Idle", 0, &ifIdle);   //获取当前轴是否空闲
    commandCheckHandler("ZAux_Direct_GetIfIdle", ret) ;//判断指令是否执行成功
    printf("1 秒后轴 0 运动状态 Idle: %s\n", ifIdle ? "停止" : "运动中");

    ret = ZAux_Direct_GetParam(handle,"AxisStatus", 0, &status);  //获取轴状态
    commandCheckHandler("ZAux_Direct_GetAxisStatus", ret) ;//判断指令是否执行成功
    printf("轴状态 AXISSTATUS：%x h\n", status);

    Sleep(2000);
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

第六章 基本运动控制

6.1 运动缓冲机制

6.1.1 指令列表

指令	说明
ZAux_Direct_GetEndMove	读取轴当前运动的最终位置
ZAux_Direct_GetEndMoveBuffer	读取当前和缓冲中运动的最终位置
ZAux_Direct_GetLoaded	读取当前除了当前运动是否还有缓冲
ZAux_Direct_GetMovesBuffered	读取当前被缓冲起来的运动个数
ZAux_Direct_GetMoveCurmark	读取当前正在运动指令的 MOVE_MARK 标号
ZAux_Direct_SetMovemark	设置运动指令的 MOVE_MARK 标号
ZAux_Direct_GetRemain	读取返回轴当前运动还未完成的距离
ZAux_Direct_GetRemain_LineBuffer	读取轴剩余的直线缓冲，按直线段来计算
ZAux_Direct_GetRemain_Buffer	参数轴剩余的缓冲，按最复杂空间圆弧来计算
ZAux_Direct_GetVectorBuffered	读取返回轴当前运动和缓冲运动还未完成的距离
ZAux_Direct_MoveOp	在运动缓冲中加入输出指令
ZAux_Direct_MoveOpAdist	在运动缓冲中设置提前输出距离
ZAux_Direct_MoveOpMulti	在运动缓冲中加入连续输出口输出指令
ZAux_Direct_MoveOpDelay	设置 op 口提前打开或关闭
ZAux_Direct_MoveOp2	在运动缓冲中加入输出指令，指定时间后输出状态翻转
ZAux_Direct_MoveAout	在运动缓冲中加入 AOUT 输出指令
ZAux_Direct_MoveDelay	在运动缓冲中加入延时指令
ZAux_Direct_MoveWait	BASE 轴运动缓冲加入一个可变的延时
ZAux_Direct_MoveTask	BASE 轴运动缓冲开启一个 TASK 任务
ZAux_Direct_MovePara	运动中修改参数
ZAux_Direct_MovePwm	运动中修改 PWM
ZAux_Direct_MoveSynmove	运动中同步其他轴的运动
ZAux_Direct_MoveASynmove	运动中触发其他轴的运动
ZAux_Direct_MoveTable	运动中修改 TABLE
ZAux_Direct_MoveCancel	运动中取消其他轴运动

ZAux_Direct_MoveModify	运动中修改结束位置
ZAux_Direct_MoveLimit	在当前的运动末尾位置增加速度限制，用于强制拐角减速
ZAux_Direct_MoveModify2	运动中修改结束位置 2
ZAux_Execute_GetMoveTime	读取对应运动剩余时间
ZAux_Execute_GetPreCalTime	预估点位运动时间，单位为 ms

6.1.2 重点说明

运动指令用来实现点位运动、直线插补、圆弧插补、凸轮运动等，在运动前要设置好轴参数。

运动指令分为单轴运动指令和多轴运动指令，单轴运动指令一次只能操作一个轴的运动，轴号缺省状态下对主轴进行操作；多轴运动指令一次可以操作多个轴运动。

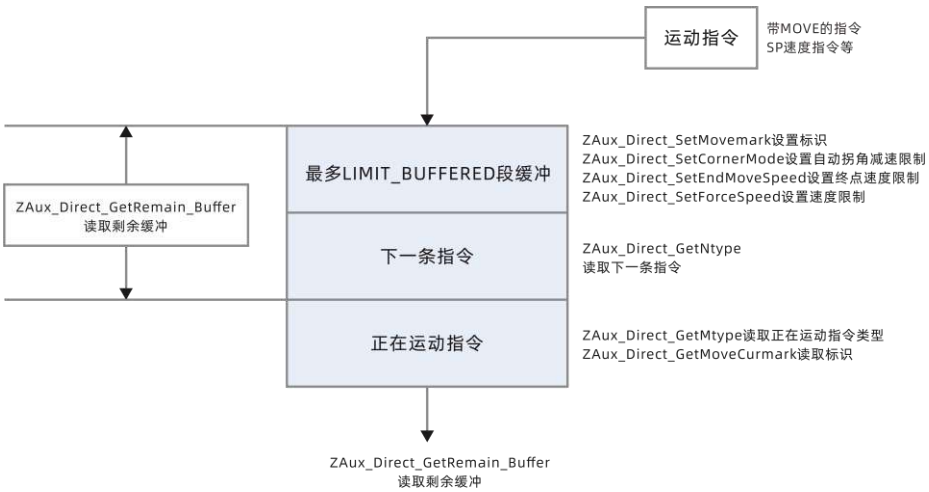
指令周期：取指令，分析指令，执行指令，执行完成取下一条，重复此过程。

ZMotion 运动控制器具有多级的缓冲，当前有运动指令正在执行时，后面调用的运动指令会填入缓冲，避免程序的阻塞。当多级运动缓冲区全部被塞满时,如果程序继续运行更多的运动指令，程序也会被阻塞。直到指令依次完成并退出，缓冲区有了空位，指令才会继续进入缓冲。

运动指令主要指直线、圆弧等各种插补运动指令以及带 MOVE 前缀的一些指令。

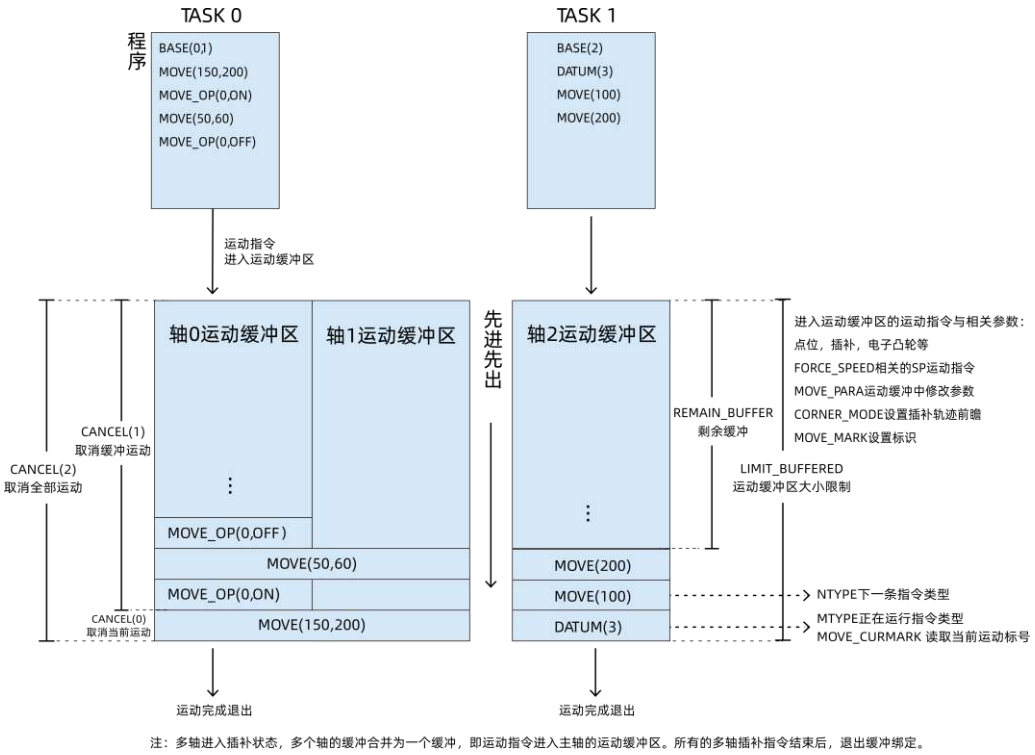
使用带 SP 函数时，**ZAux_Direct_SetForceSpeed**、**ZAux_Direct_SetEndMoveSpeed**、**ZAux_Direct_SetStartMoveSpeed** 会随 sp 运动指令写入运动缓存区。

可以用 **ZAux_Direct_MoveOp** 把 IO 操作指令填入缓冲，这样在运动指令之间自动加入 IO 输出。用 **ZAux_Direct_MoveDelay** 把延时指令填入缓冲，这样在运动指令之间自动延时。



每个轴的运动缓冲互不干扰，如下图。ZMC4 系列运动控制器每个轴可支持多达 4096 段运动缓冲（不同型号的控制器缓冲个数有差别，具体情况参见控制器手册说明）。

以正运动 ZMC406-V2 控制器为例，运动缓冲执行机制如下图：



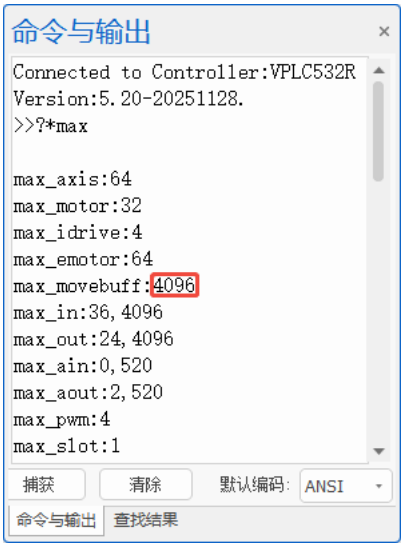
支持运动缓冲的指令有：如：ZAux_Direct_MoveSp，ZAux_Direct_MoveAbsSp，ZAux_Direct_MoveCircSp 等，即后缀带 SP 函数的指令。

不支持运动缓冲的指令：如 ZAux_Direct_MoveModify 等

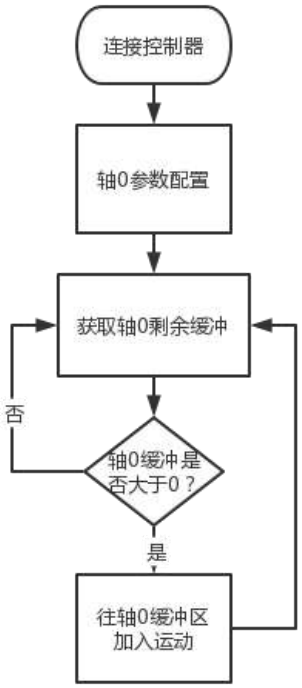
6.1.3 例程

6.1.3-1 获取剩余运动缓冲数量

在 RTSys 开发环境中可以通过在命令行输入?*max 在信息输出窗口来查看最大的运动缓冲区大小，例如 VPLC532R 运动缓冲区大小为 max_movebuff:4096。



本例程获取轴 0 缓冲区剩余缓冲数，根据剩余缓冲数是否大于 0，往缓冲区里面加入运动。




```
// test1.cpp : 定义控制台应用程序的入口点。
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s return code is %d\n", command, ret);
    }

}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";                //控制器 IP 地址
```

```
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("connect controller failed!\n");
        handle = NULL;
        getchar();
        return -1;
    }
    printf("控制器连接成功! \n");

    ZAux_Direct_SetAtype(handle, 0, 1);   //设置轴 0 轴类型为 1
    ZAux_Direct_SetUnits(handle, 0, 100); //设置轴 0 脉冲当量为 100
    ZAux_Direct_SetSpeed(handle, 0, 200); //设置轴 0 速度为 200units/s
    ZAux_Direct_SetAccel(handle, 0, 2000); //设置轴 0 加速度为 2000units/s/s
    ZAux_Direct_SetDecel(handle, 0, 2000); //设置轴 0 减速度为 2000units/s/s

    int MoveTime = 0;
    int GetValue;


    ret = ZAux_Direct_SetDpos( handle, 0 , 0);//DPOS 位置清 0
    commandCheckHandler("ZAux_Direct_SetDpos",ret);

    while (MoveTime < 100)
    {
        ret = ZAux_Direct_GetRemain_Buffer( handle, 0, &GetValue);//获取轴 0 剩余缓冲
数
        commandCheckHandler("ZAux_Direct_GetRemain_Buffer", ret);
        printf("Remain_Buffer = %d \n", GetValue);

        if (GetValue > 0)//若轴 0 缓冲数>0
        {
            ret = ZAux_Direct_Single_Move( handle, 0, 10);//给轴 0 缓冲加入运动
            commandCheckHandler("ZAux_Direct_Single_Move", ret);
            MoveTime += 1;
        }
```

```
    }


    Sleep(2000);
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

6.1.3-2 运动中触发输出（move_op2）

举例：假设在某个执行机构中，需要实现轴 0 运动 1mm 需要 100 个脉冲，轴 0 总共运动 2 米，每隔 100mm 触发一次输出口 0 用于点胶。

```
// test1.cpp : 定义控制台应用程序的入口点。
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非0则失败
    {
        printf("%s return code is %d\n", command, ret);
    }

}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";                //控制器IP地址
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("connect controller failed!\n");
        handle = NULL;
        getchar();
```

```
        return -1;
    }
    printf("控制器连接成功! \n");

    ZAux_Direct_SetAtype(handle, 0, 1);   //设置轴 0 轴类型为4
    ZAux_Direct_SetUnits(handle, 0, 100); //设置轴 0 脉冲当量为100,对应1mm

    ZAux_Direct_SetSpeed(handle, 0, 200); //设置轴 0 速度为 200units/s
    ZAux_Direct_SetAccel(handle, 0, 2000); //设置轴 0 加速度为 2000units/s/s
    ZAux_Direct_SetDecel(handle, 0, 2000); //设置轴 0 减速度为 2000units/s/s

    int MoveTime = 0;
    int GetValue;

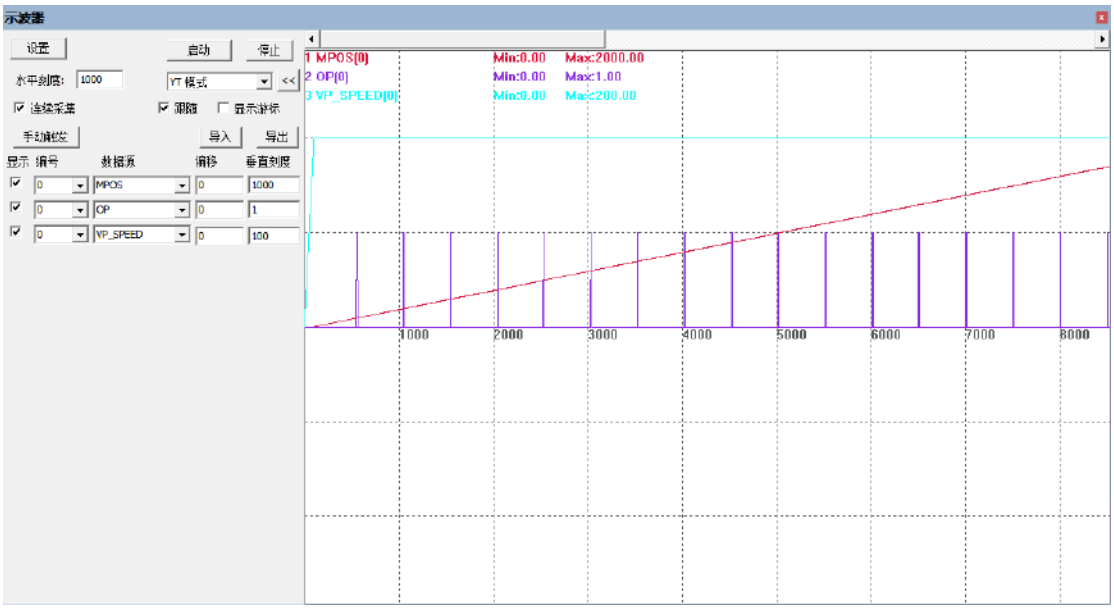

    ret = ZAux_Direct_SetDpos( handle, 0 , 0);//DPOS位置清0,
    commandCheckHandler("ZAux_Direct_SetDpos",ret);
    ret = ZAux_Direct_SetMpos( handle, 0 , 0);//MPOS位置清0,
    commandCheckHandler("ZAux_Direct_SetMpos",ret);
    ret = ZAux_Direct_SetMerge( handle, 0 , 1);//打开连续插补
    commandCheckHandler("ZAux_Direct_SetMerge",ret);

    ret =ZAux_Trigger(handle);
    commandCheckHandler("ZAux_Trigger",ret);

    for (int i=0;i<20;i++)
    {
        ret = ZAux_Direct_Single_Move(handle, 0 , 100);
        commandCheckHandler("ZAux_Direct_Single_Move", ret);
        ret = ZAux_Direct_MoveOp2(handle, 0 , 0,1,10);
        commandCheckHandler("ZAux_Direct_MoveOp2", ret);
    }

    Sleep(2000);
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

由下图示波器可观察出，在速度不掉的情况下，每隔 100 个脉冲当量的位置，OUT 0 输出一次：



6.2 单轴运动

6.2.1 指令列表

指令	说明
ZAux_Direct_Single_Move	单轴相对运动
ZAux_Direct_Single_MoveAbs	单轴绝对运动
ZAux_Direct_Single_Vmove	单轴连续运动
ZAux_Direct_SetJogSpeed	设置 JOG 时速度
ZAux_Direct_GetJogSpeed	读取 JOG 时速度
ZAux_Direct_GetFastJog	读取 JOG 速度输入 IN，有输入时，速度由 SPEED 参数给出。如果没有输入，速度由 JOGSPEED 参数给出
ZAux_Direct_SetFastJog	设置 JOG 速度输入 IN，有输入时，速度由 SPEED 参数给出。如果没有输入，速度由 JOGSPEED 参数给出
ZAux_Direct_SetFwdJog	设置正向 JOG 输入对应的输入编号-1 无效
ZAux_Direct_GetFwdJog	读取正向 JOG 输入对应的输入编号-1 无效
ZAux_Direct_SetRevJog	设置负向 JOG 输入对应的输入编号-1 无效
ZAux_Direct_GetRevJog	读取负向 JOG 输入对应的输入编号-1 无效

6.2.2 重点说明

控制器每个轴都有自己独立的轴参数，并且运动的参数值全局性的，即速度、加速度一

变则立刻生效，可以实现在线变速的功能。

当调用 ZAux_Direct_Single_Cancel 停止运动指令时，需通过使用减速度指令 ZAux_Direct_SetFastDec(缺省默认等于减速度的值)来设置的值减速，要实现急停时可以将快减减速度设置为驱动器的最大允许值。

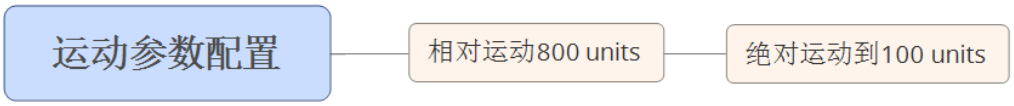
单轴持续运动（ZAux_Direct_Single_Vmove）连续往一个方向运动。若前面的单轴持续运动没有停止时，新的单轴持续运动指令会自动替换前面的单轴持续运动并修改方向，因此不需要调用 ZAux_Direct_Single_Cancel 停止前面的单轴持续运动。

6.2.3 例程

6.2.3-1 单轴点位运动

初始时命令位置为 0，脉冲当量设为 100，速度设为 200units/s，加速度 2000unit/s^2，减速度 2000units/s^2，S 形曲线加减速时间设为 200ms，先做相对运动，运动 800units，然后再做绝对运动，运动到绝对位置 100units 的位置。

Ps: S 形曲线加减速时间为运动加减速过程的变化时间，且为轨迹为 S 形，非梯形。




```
// test1.cpp : 定义控制台应用程序的入口点。
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s return code is %d\n", command, ret);
    }

}
```

```
int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";                //控制器 IP 地址
    ZMC_HANDLE handle = NULL;                           //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("connect controller failed!\n");
        handle = NULL;
        getchar();
        return -1;
    }
    printf("控制器连接成功！\n");

    ZAux_Direct_SetAtype(handle, 0, 1);   //设置轴 0 轴类型为 1
    ZAux_Direct_SetUnits(handle, 0, 100); //设置轴 0 脉冲当量为 100
    ZAux_Direct_SetSpeed(handle, 0, 200); //设置轴 0 速度为 200units/s
    ZAux_Direct_SetAccel(handle, 0, 2000); //设置轴 0 加速度为 2000units/s/s
    ZAux_Direct_SetDecel(handle, 0, 2000); //设置轴 0 减速度为 2000units/s/s
    ZAux_Direct_SetSramp(handle, 0, 200); //设置轴 0 S 曲线为 200ms

    ret = ZAux_Direct_SetDpos( handle,  0,  0);//轴指令位置清 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret);//判断指令是否执行成功
    ret = ZAux_Direct_SetMpos( handle,  0,  0);//编码器反馈位置清 0
    commandCheckHandler("ZAux_Direct_SetMpos", ret);//判断指令是否执行成功

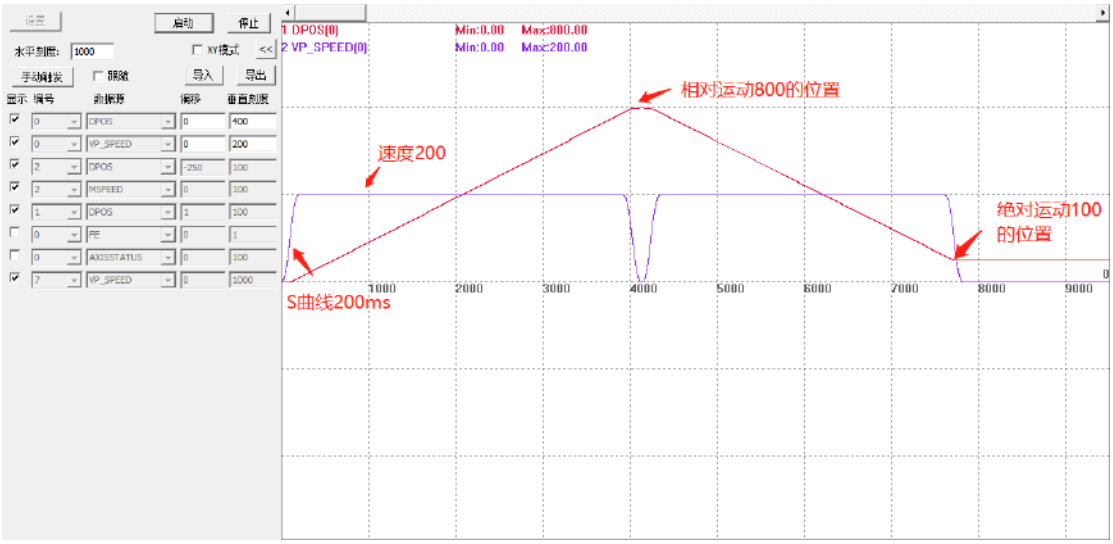
    ZAux_Trigger(handle);//示波器触发函数
    ret =ZAux_Direct_Single_Move( handle,  0,  800);
    commandCheckHandler("ZAux_Direct_SetDpos", ret);//判断指令是否执行成功

    float IDLE;
    while (1)//等待轴 0 运动完成
    {
        Sleep(100);
        ZAux_Direct_GetParam(handle,"IDLE",0,&IDLE);
        if (IDLE<0)break;
    }
```

```
    ret =ZAux_Direct_Single_MoveAbs( handle,  0, 100);
    commandCheckHandler("ZAux_Direct_SetDpos", ret);//判断指令是否执行成功
    while (1)//等待轴 0 运动完成
    {
        Sleep(100);
        ZAux_Direct_GetParam(handle,"IDLE",0,&IDLE);
        if (IDLE<0)break;
    }

    Sleep(2000);
    ret = ZAux_Close(handle);     //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

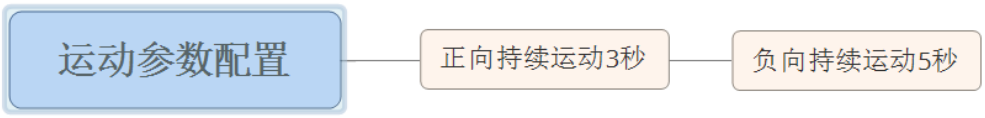
波形图：



6.2.3-2 单轴持续运动

初始时命令位置为 0，脉冲当量设为 100，速度设为 200units/s，加速度 2000unit/s^2，减速度 2000units/s^2，S 形曲线加减速时间设为 200ms，先做正向持续运动 3 秒中，然后再做负向持续运动 5 秒中，后停止运动。

Ps：S 形曲线加减速时间为运动加减速过程的变化时间，且为轨迹为 S 形，非梯形。




```
// test1.cpp : 定义控制台应用程序的入口点。
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s return code is %d\n", command, ret);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";                //控制器 IP 地址
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("connect controller failed!\n");
        handle = NULL;
        getchar();
        return -1;
    }
    printf("控制器连接成功! \n");

    ZAux_Direct_SetAtype(handle, 0, 1);   //设置轴 0 轴类型为 1
    ZAux_Direct_SetUnits(handle, 0, 100); //设置轴 0 脉冲当量为 100
    ZAux_Direct_SetSpeed(handle, 0, 200); //设置轴 0 速度为 200units/s
```

```
    ZAux_Direct_SetAccel(handle, 0, 2000); //设置轴 0 加速度为 2000units/s/s
    ZAux_Direct_SetDecel(handle, 0, 2000); //设置轴 0 减速度为 2000units/s/s
    ZAux_Direct_SetSramp(handle, 0, 200); //设置轴 0 S 曲线为 200ms

    ret = ZAux_Direct_SetDpos( handle,  0,  0);//轴指令位置清 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret);//判断指令是否执行成功
    ret = ZAux_Direct_SetMpos( handle,  0,  0);//编码器反馈位置清 0
    commandCheckHandler("ZAux_Direct_SetMpos", ret);//判断指令是否执行成功

    ZAux_Trigger(handle);//示波器触发函数
    ret =ZAux_Direct_Single_Vmove( handle,  0,  1);
    commandCheckHandler("ZAux_Direct_Single_Vmove", ret);//判断指令是否执行成功

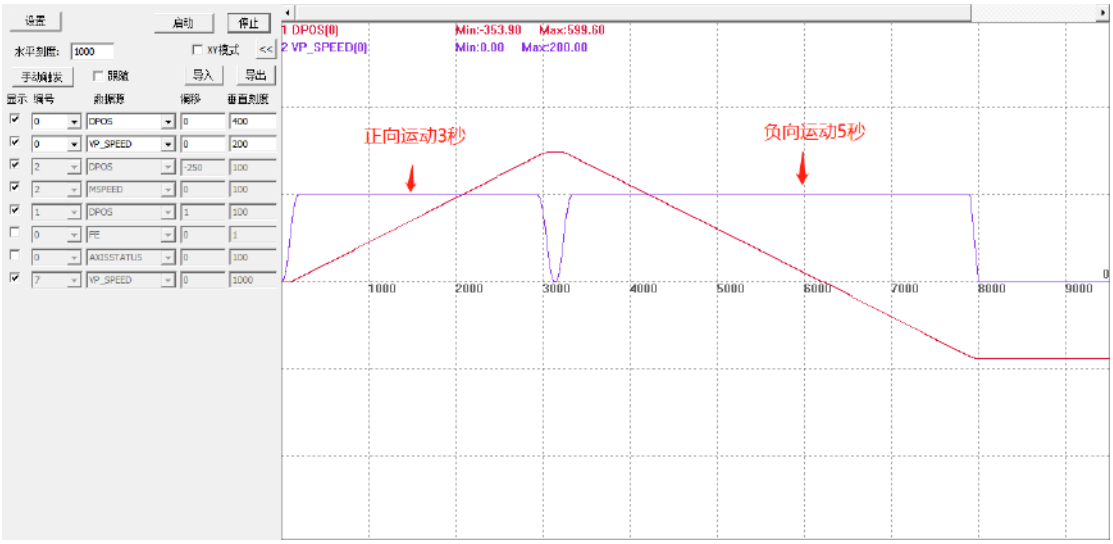
    Sleep(3000);//延时三秒

    ret =ZAux_Direct_Single_Vmove( handle,  0,  -1);
    commandCheckHandler("ZAux_Direct_SetDpos", ret);//判断指令是否执行成功



    Sleep(5000);//延时 5 秒
    ret = ZAux_Direct_Single_Cancel(handle,0,2);//停止运动
    commandCheckHandler("ZAux_Direct_Single_Cancel", ret) ;//判断指令是否执行成功
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

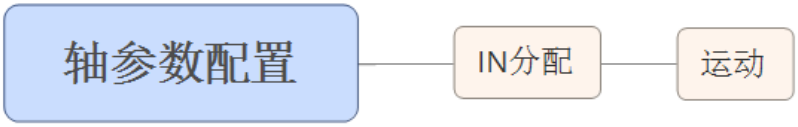
波形图：



6.2.3-3 快速点动（IN 控制）

初始时命令为 0，脉冲当量设为 100，速度设为 200units/s，加速度 2000unit/s^2，减速度 2000units/s^2，S 形曲线加减速时间设为 200ms。JOG 速度设置为 100。把输入口 IN0 映射为控制使用 JOG 速度和 SPEED 速度的输入口，把输入口 IN1 设置为正方向运动控制的输入口，把输入口 IN2 映射为负方向运动的控制的输入口。

Ps：S 形曲线加减速时间为运动加减速过程的变化时间，且为轨迹为 S 形，非梯形。




```
// test1.cpp : 定义控制台应用程序的入口点。
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s return code is %d\n", command, ret);
    }
```

```
}
int32 __stdcall ZAux_Direct_ZSIMU_IN(ZMC_HANDLE handle,int link_axis,int move_axis)
{


    char  cmdbuff[2048];
    char  cmdbuffAck[2048];
    //生成命令
    sprintf(cmdbuff, "ZSIMU_IN(%d,%d) ", link_axis,move_axis);


    //调用命令执行函数
    return ZAux_DirectCommand(handle, cmdbuff, cmdbuffAck, 2048);
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";                //控制器 IP 地址
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("connect controller failed!\n");
        handle = NULL;
        getchar();
        return -1;
    }
    printf("控制器连接成功！\n");

    ZAux_Direct_SetAtype(handle, 0, 1);   //设置轴 0 轴类型为 1
    ZAux_Direct_SetUnits(handle, 0, 100); //设置轴 0 脉冲当量为 100
    ZAux_Direct_SetSpeed(handle, 0, 200); //设置轴 0 速度为 200units/s
    ZAux_Direct_SetAccel(handle, 0, 2000); //设置轴 0 加速度为 2000units/s/s
    ZAux_Direct_SetDecel(handle, 0, 2000); //设置轴 0 减速度为 2000units/s/s
    ZAux_Direct_SetSramp(handle, 0, 200); //设置轴 0 S 曲线为 200ms
```

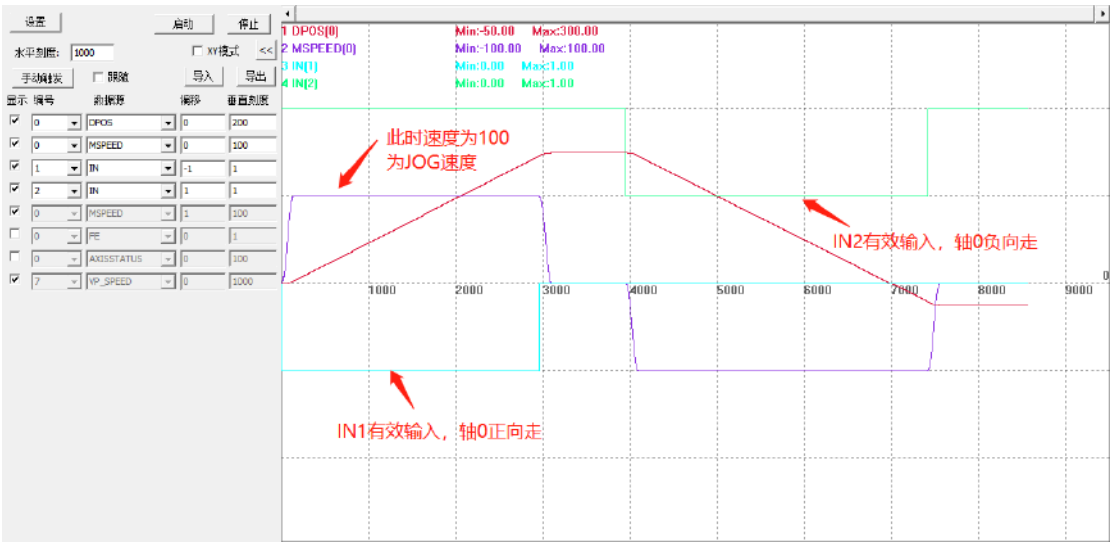
```
    ret = ZAux_Direct_SetDpos( handle,  0,  0);//轴指令位置清 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret);//判断指令是否执行成功
    ret = ZAux_Direct_SetMpos( handle,  0,  0);//编码器反馈位置清 0
    commandCheckHandler("ZAux_Direct_SetMpos", ret);//判断指令是否执行成功

    ZAux_Direct_SetJogSpeed(handle, 0, 100);//设置 JOG 速度为 100
    ZAux_Direct_SetFastJog(handle, 0, 0);//设置 JOG 速度是否启动控制 IN
    ZAux_Direct_SetFwdJog(handle, 0, 1);//设置 JOG 正方向控制 IN0
    ZAux_Direct_SetRevJog(handle, 0, 2);//设置 JOG 负方向控制 IN1

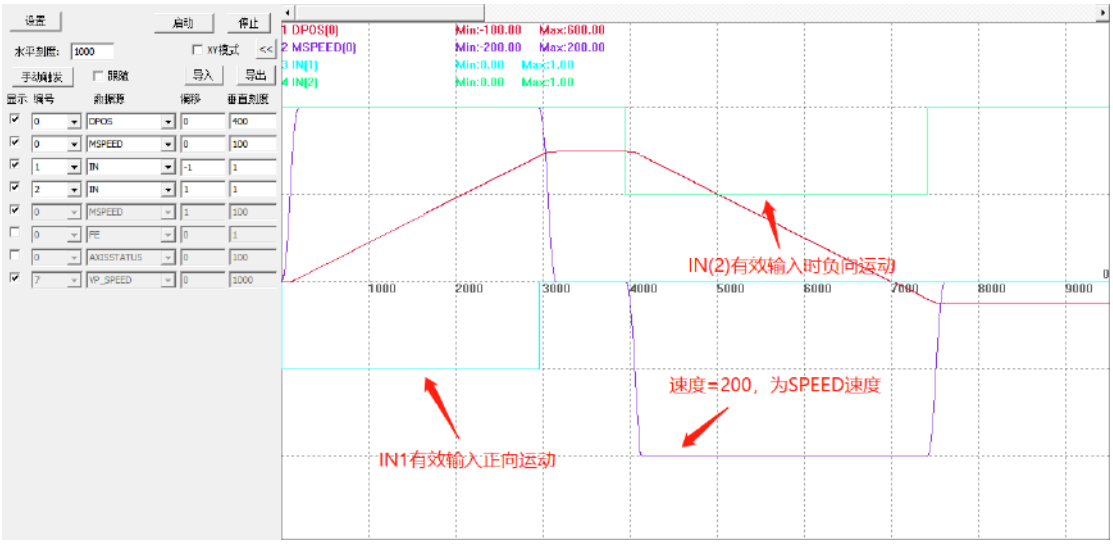
    ZAux_Direct_SetInvertIn(handle, 0, 1);//反转 IN 有效输入。ECI 系列的控制卡不用反转
    ZAux_Direct_SetInvertIn(handle, 1, 1);//反转 IN 有效输入。ECI 系列的控制卡不用反转
    ZAux_Direct_SetInvertIn(handle, 2, 1);//反转 IN 有效输入。ECI 系列的控制卡不用反转
    ZAux_Trigger(handle);//示波器触发函数

    Sleep(500);
    ret = ZAux_Close(handle);     //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

IN(0)未有效输入时：



IN(0)有效输入时：



6.3 回零运动

6.3.1 指令列表

指令	说明
ZAux_Direct_Single_Datum	控制器方式回零
ZAux_BusCmd_Datum	总线驱动器回零
ZAux_Direct_UserDatum	自定义二次回零
ZAux_Direct_SetHomeWait	设置回零反找等待时间
ZAux_Direct_GetHomeWait	读取回零反找等待时间
ZAux_BusCmd_SetDatumOffpos	设置回零偏移距离
ZAux_BusCmd_GetDatumOffpos	读取回零偏移距离
ZAux_Direct_SetCreep	设置回零爬行速度
ZAux_Direct_GetCreep	读取回零爬行速度
ZAux_Direct_GetHomeStatus	回零完成状态
ZAux_Direct_GetHomeStatus2	回零完成状态 2
ZAux_BusCmd_GetHomeStatus	驱动器回零完成状态
ZAux_BusCmd_GetHomeStatus2	驱动器回零完成状态 2
ZAux_Direct_SetDatumIn	设置回零原点信号对应输入口
ZAux_Direct_GetDatumIn	读取回零原点信号对应输入口
ZAux_Direct_SetInvertIn	设置输入口有效状态反转
ZAux_Direct_GetInvertIn	读取输入口反转状态

6.3.2 重点说明

原点开关的输入点由参数 **ZAux_Direct_SetDatumIn** 决定，正负限位开关通过 **ZAux_Direct_SetFwdIn**，**ZAux_Direct_SetRevIn** 设置。

ZMC 控制器为 0 触发有效，输入为 OFF 状态时，表示到达原点/限位，常开类型信号需要采用 **ZAux_Direct_SetInvertIn** 反转电平。

ECI 控制器为 1 触发有效，输入为 ON 状态时，表示到达原点/限位，常闭类型信号需要采用 **ZAux_Direct_SetInvertIn** 反转电平。

Z 信号回零必须配置为带 Z 信号轴类型。

多轴回零时，需要都每个轴都使用回零函数。

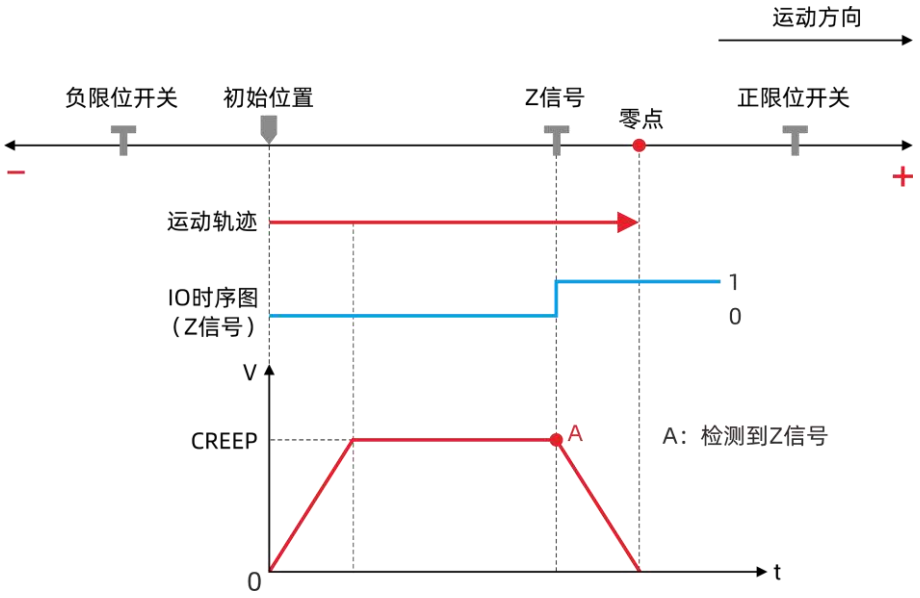
总线控制器使用控制器找原点模式完成后，需要手动清零反馈位置。

回零方式如下：

回零模式 1

如下图，**ZAux_Direct_Single_Datum** 模式 1：轴以 **ZAux_Direct_SetCreep** 速度正向运行，直到 Z 信号出现后开始减速，减速到 0 之后，此时将命令位置值重置为 0，停止后所处位置为原点，回零途中若碰到限位开关会直接停止。

回零模式 2 与模式 1 类型一致，找原点运动方向相反。



Z 相模式回零

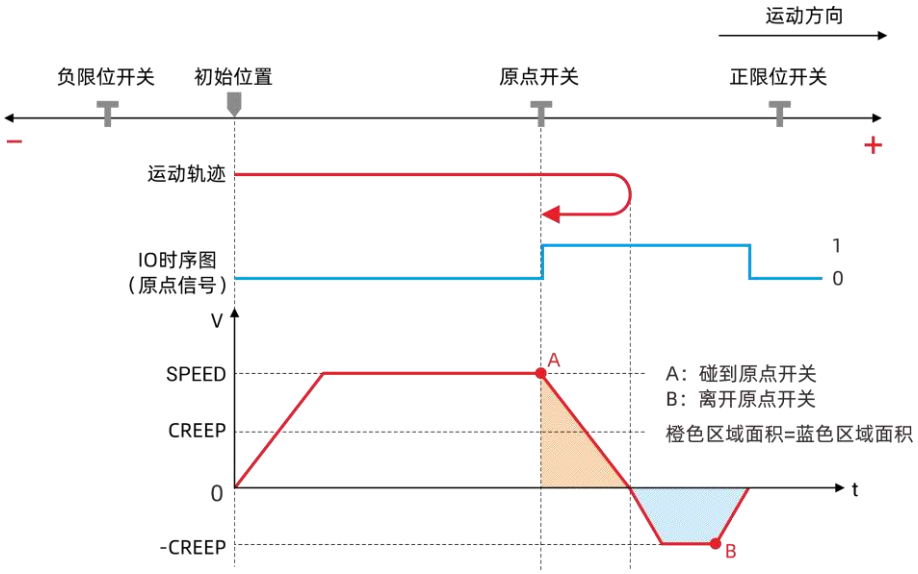
注：回零碰到 Z 相信号后，是用 Decel 的减速度，减速到 0 停止的。存在 1 小段误差，可以通过把 Creep 的速度减小，Decel 减速度加大来减小误差。若存在高精度的回零，则需要自己通过 Z 相锁存，手动做回零来达到高精度的 Z 相回零效果。

回零模式 3

如下图，**ZAux_Direct_Single_Datum** 模式 3：轴以 **ZAux_Direct_SetSpeed** 速度快速正向运行,直到碰到原点开关后开始减速，减速到 0 之后再反向以 **ZAux_Direct_SetCreep** 速度

离开原点，再次碰到原点之后减速停止，轴停止之后将位置值重置为 0，当前所处位置为原点，回零途中若碰到限位开关会直接停止。

回零模式 4 与模式 3 类型一致，找原点运动方向相反。

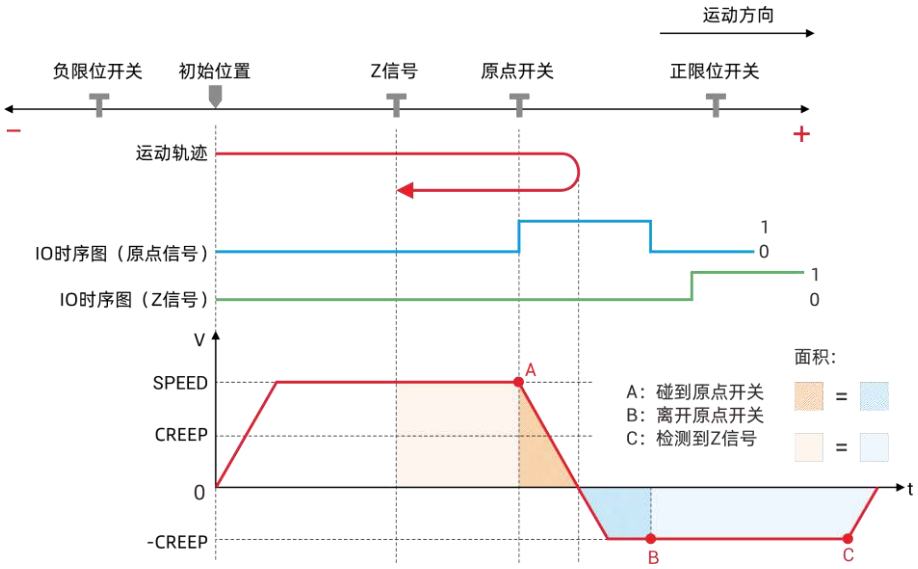


注：回零碰到原点信号后，是减速到 Creep 速度，反向运动遇到原点 IO 感应就停止，用减速到 0 停止的。存在从离开 IO 感应到减速为 0 之间的一小段误差，可以通过把 Creep 的速度减小，Decel 减速度加大来减小误差。若存在高精度的回零，则需要自己通过 IO 锁存，手动做回零来达到高精度的效果。

回零模式 5

如下图，**ZAux_Direct_Single_Datum** 模式 5：轴以 **ZAux_Direct_SetSpeed** 速度快速正向运行,直到碰到原点开关后开始减速，减速到 0 之后再反向以 **ZAux_Direct_SetCreep** 速度运动，直到 Z 信号出现之后减速停止，遇到 Z 信号便将位置值重置为 0，Z 信号出现的位置为原点，回零途中若碰到限位开关会直接停止。

回零模式 6 与模式 5 类型一致，找原点运动方向相反。



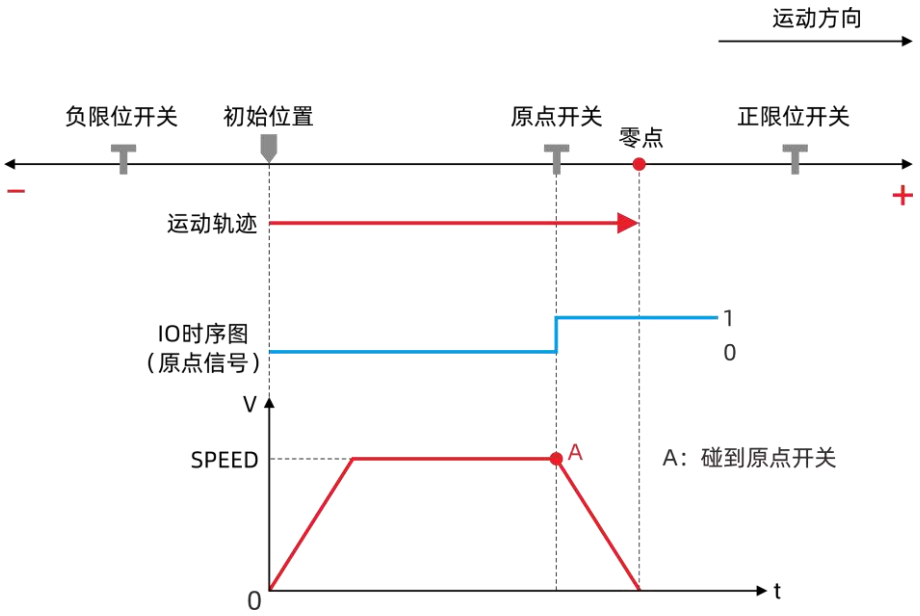
注：回零碰到原点信号后，是用 Decel 的减速度减速到 Creep 速度，反向运动，直到离

开原点 IO 感应，碰到 Z 相信号后，用 Decel 的减速度减速到 0 停止的。存在从离开 Z 信号感应到减速为 0 之间的一小段误差，可以通过把 Creep 的速度减小，Decel 减速度加大来减小误差。若存在高精度的回零，则需要自己通过 Z 相锁存，手动做回零来达到高精度的效果。

回零模式 8

如下图，**ZAux_Direct_Single_Datum** 模式 8：轴以 **ZAux_Direct_SetSpeed** 速度快速正向运行,直到碰到原点开关后开始减速，减速到 0 之后便将位置值重置为 0，停止后所处位置为原点，回零途中若碰到限位开关会直接停止。

回零模式 9 与模式 8 类型一致，找原点运动方向相反。



注：回零碰到原点信号后，是用 Decel 的减速度，减速到 0 停止的。存在一小段误差，可以通过把 speed 的速度减小，Decel 减速度加大来减小误差。若存在高精度的回零，则需要自己通过 IO 锁存，手动做回零来达到高精度的回零效果。

6.3.3 例程

6.3.3-1 单轴回零运动

本例程采用回零模式 3，轴以 **ZAux_Direct_SetSpeed** 速度正向运行，直到碰到原点开关。然后轴以 **ZAux_Direct_SetCreep** 速度反向运动直到离开原点开关。同时位置会自动清 0。例子里面以 IN0 作为原点开关，开启了电平反转。




```
// test1.cpp : 定义控制台应用程序的入口点。
```

```
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s return code is %d\n", command, ret);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";                //控制器 IP 地址
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("connect controller failed!\n");
        handle = NULL;
        getchar();
        return -1;
    }
    printf("控制器连接成功! \n");

    ZAux_Direct_SetAtype(handle, 0, 1);   //设置轴 0 轴类型为 1
    ZAux_Direct_SetUnits(handle, 0, 100); //设置轴 0 脉冲当量为 100
    ZAux_Direct_SetSpeed(handle, 0, 200); //设置轴 0 速度为 200units/s
    ZAux_Direct_SetAccel(handle, 0, 2000); //设置轴 0 加速度为 2000units/s/s
    ZAux_Direct_SetDecel(handle, 0, 2000); //设置轴 0 减速度为 2000units/s/s
    ZAux_Direct_SetSramp(handle, 0, 200); //设置轴 0 S 曲线时间为 200ms
    ret = ZAux_Direct_SetCreep(handle, 0, 50);        //设置回零时反向爬行速度
    commandCheckHandler("ZAux_Direct_SetCreep", 0);

    ret = ZAux_Direct_SetDatumIn(handle, 0, 0);       //设置原点点开关
```

```
    commandCheckHandler("ZAux_Direct_SetDatumIn", 0);

    ret = ZAux_Direct_SetHomeWait(handle, 0, 1000);  //设置回零等待时间
    commandCheckHandler("ZAux_Direct_SetHomeWait", 0);

    ret = ZAux_Direct_SetInvertIn(handle, 0, 1);     //设置输入口 0 电平反转
    commandCheckHandler("ZAux_Direct_SetInvertIn", 0);

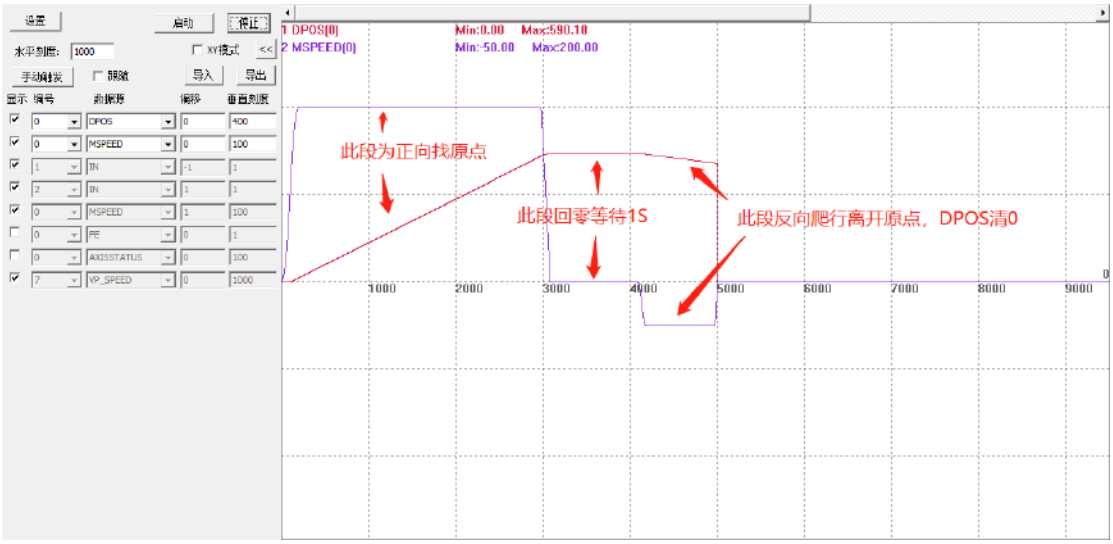
    ret = ZAux_Direct_SetDpos( handle,  0,  0);//轴指令位置清 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret);//判断指令是否执行成功
    ret = ZAux_Direct_SetMpos( handle,  0,  0);//编码器反馈位置清 0
    commandCheckHandler("ZAux_Direct_SetMpos", ret);//判断指令是否执行成功

    ZAux_Trigger(handle);//示波器触发函数
    ret = ZAux_Direct_Single_Datum(handle, 0, 3);    //回零，模式 3
    commandCheckHandler("ZAux_Direct_Single_Datum", 0);

    uint32 homestatus;
    while (1)//等待轴 0 回零运动完成
    {
        Sleep(100);
        ZAux_Direct_GetHomeStatus(handle,0,&homestatus);//获取回零运动完成状态
        if (homestatus==1)break;
    }

    Sleep(2000);
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

波形图：



6.4 电子齿轮

6.4.1 指令列表

指令	说明
ZAux_Direct_SetClutchRate	设置连接速率
ZAux_Direct_GetClutchRate	获取连接速率
ZAux_Direct_EncoderRatio	编码器输入齿轮比，缺省(1,1)
ZAux_Direct_StepRatio	设置脉冲输出齿轮比，缺省(1,1)
ZAux_Direct_Connect	连接 同步运动指令 电子齿轮
ZAux_Direct_Connpath	连接 同步运动 2 指令 电子齿轮（矢量方式）
ZAux_Direct_GetLinkax	读取当前连接运动的参考轴号

6.4.2 重点说明

电子齿轮功能用于两个轴的连接，将主轴与从轴按照一个常数齿轮比建立连接，不需要物理齿轮，使用指令直接设置电子齿轮的比值，由于是使用软件实现的，故电子齿轮比可以随时更改。

电子齿轮功能通过指令 **ZAux_Direct_Connect、ZAux_Direct_Connpath** 实现。将一个轴按照一定比例连接到另一个轴上做跟随运动，一条运动指令就能驱动两个电机的运行，通过对这两个电机轴移动量的检测，将位移偏差反馈到控制器并获得同步补偿，这样能使两个轴之间的位移偏差量控制在精度允许范围内。

电子齿轮连接的是脉冲个数，例如主从轴连接比例为 1:5，给主轴发送 1 个脉冲，此时对应给从轴发送 5 个脉冲。

同步指令连接 2 个轴后，取消时需要用 cancel 指令取消当前轴的同步运动，然后再调用别的运动，当切换同步倍率时可以直接调用 **ZAux_Direct_Connect** 来取代前面的 **ZAux_Direct_Connect** 指令。连接的是脉冲个数，要考虑不同轴 UNITS 的比例。

与凸轮不同，电子齿轮的连接是线性的。

电子齿轮功能通过指令 **ZAux_Direct_Connect**、**ZAux_Direct_Connpath** 实现，将一个轴按照一定比例连接到另一个轴上做跟随运动，一条运动指令就能驱动两个电机的运行，通过对这两个电机轴移动量的检测，将位移偏差反馈到控制器并获得同步补偿，这样能使两个轴之间的位移偏差量控制在精度允许范围内。

电子齿轮的作用：

1. 脉冲补偿，减少上位机负担（因为目前用的发送脉冲的元件，都有发送脉冲频率的限制）。

2. 匹配电机发出的脉冲数与机械最小移动量，可将指令输入 1 个脉冲对应的工件（或电机）移动量设定为任意值；可实现电机的无极变速，在电机启动和停止时，可以防止失步和过冲现象，这样就能充分发挥电机的潜能。

3. 传递同步运动信息，实现坐标的联动、运动形式之间的变换（旋转-旋转，旋转-直线，直线-直线）、简化控制等。

ZAux_Direct_Connpath 与 **ZAux_Direct_Connect** 的相同点：二者的使用语法相同，连接的都是脉冲个数，**ZAux_Direct_Connpath** 连接到单个轴的运动的效果与 **ZAux_Direct_Connect** 相同。

ZAux_Direct_Connpath 与 **ZAux_Direct_Connect** 的区别：**ZAux_Direct_Connect** 连接的是单个轴的目标位置。**ZAux_Direct_Connpath** 是连接的是插补轴的矢量长度，此时需要连接在插补运动的主轴上，连接到插补从轴上无法跟随插补运动。**ZAux_Direct_Connpath** 会跟踪 XY 轴插补的矢量长度变化，而不是跟踪单独的 X 轴或者 Y 轴。

6.4.3 例程

6.4.3-1 同步运动

本例程驱动轴 0 运动了 100 个脉冲当量的运动距离，把轴 1 轴 0 的轴参数设置一样，比较同步前与同步后的效果区别。

```
// test1.cpp : 定义控制台应用程序的入口点。
//


#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
```

```
#include "zauxdll2.h"



void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
    }

}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";  //控制器 IP 地址，使用仿真器查看时曲线可能会有误差，最好连实物控制器查看
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        getchar();
        return -1;
    }
    printf("控制器连接成功! \n");

    ret =ZAux_Direct_SetAtype(handle,0,1);//设置轴 0 轴类型为 1
    commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetAtype(handle,1,1);//设置轴 1 轴类型为 1
    commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetUnits(handle,0,100);//设置轴 0 脉冲当量为 100
    commandCheckHandler("ZAux_Direct_SetUnits", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetUnits(handle,1,100);//设置轴 1 脉冲当量为 100
    commandCheckHandler("ZAux_Direct_SetUnits", ret) ;//判断指令是否执行成功
```

```
ret =ZAux_Direct_SetAccel(handle,0,500);//设置轴 0 加速度
commandCheckHandler("ZAux_Direct_SetAccel", ret) ;//判断指令是否执行成功
ret =ZAux_Direct_SetAccel(handle,1,500);//设置轴 1 加速度
commandCheckHandler("ZAux_Direct_SetAccel", ret) ;//判断指令是否执行成功

ret =ZAux_Direct_SetDecel(handle,0,500);//设置轴 0 减速度
commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//判断指令是否执行成功
ret =ZAux_Direct_SetDecel(handle,1,500);//设置轴 1 减速度
commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//判断指令是否执行成功

ret =ZAux_Direct_SetDpos(handle,0,0);//设置轴 0 DPOS 清 0
commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//判断指令是否执行成功
ret =ZAux_Direct_SetDpos(handle,1,0);//设置轴 1 DPOS 清 0
commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//判断指令是否执行成功

ret =ZAux_Direct_SetSpeed(handle,0,100);//设置轴 0 速度
commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//判断指令是否执行成功

ret =ZAux_Direct_SetSpeed(handle,1,100);//设置轴 1 速度
commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//判断指令是否执行成功

ret =ZAux_Direct_Connect(handle,1,0,1);//设置开启同步连接，轴 1 跟着轴 0 走
commandCheckHandler("ZAux_Direct_Connect", ret) ;//判断指令是否执行成功

float IDLE;
ZAux_Trigger( handle);//触发示波器

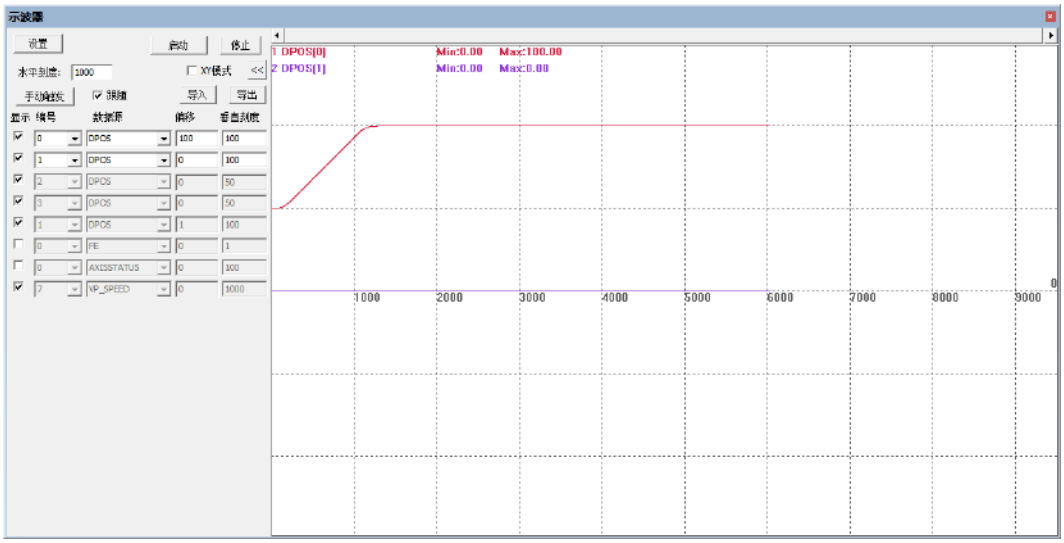
ret = ZAux_Direct_Single_Move(handle, 0,100.0); //轴 1 运动 100
commandCheckHandler("ZAux_Direct_Single_Move", ret) ;//判断指令是否执行成功


while (1)//等待轴 0 运动完成
{
    Sleep(100);
    ZAux_Direct_GetParam(handle,"IDLE",0,&IDLE);
    if (IDLE<0)break;
}
```

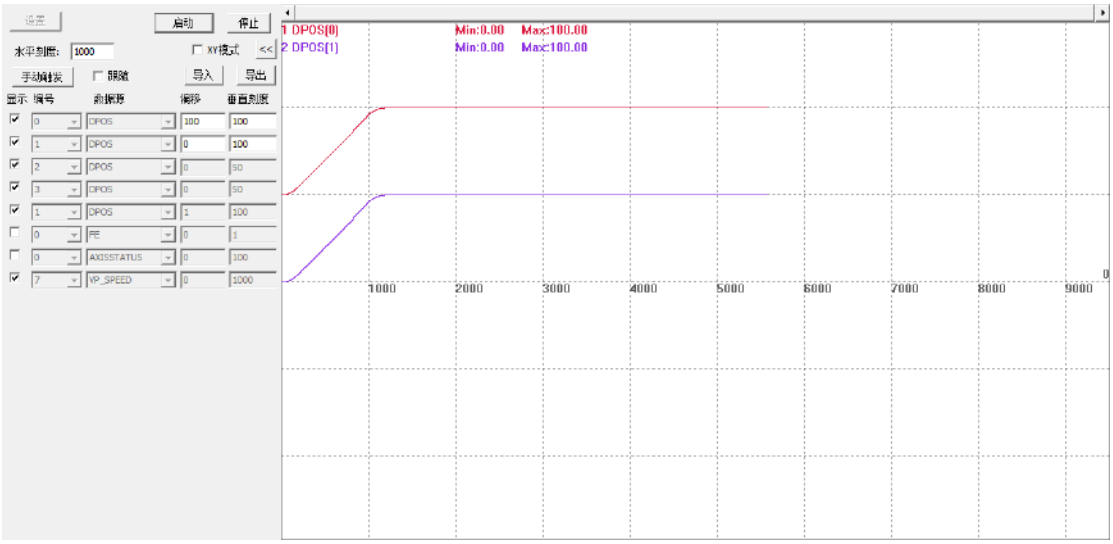
```
        ret =ZAux_Direct_Single_Cancel(handle,0,2);//取消同步运动
    commandCheckHandler("ZAux_Direct_Connect", ret) ;//判断指令是否执行成功


    getchar();
    ret = ZAux_Close(handle);     //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

未启动同步运动前：



启动同步运动后：ZAux_Direct_Connect（控制器连接句柄，1，0，1）



6.4.3-2 同步运动 2

本例程主要是驱动轴 2，跟随轴 0，轴 1 的插补合成矢量运动，同插补合成矢量进行同步运动。

```
// test1.cpp : 定义控制台应用程序的入口点。
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";  //控制器 IP 地址，使用仿真器查看时曲线可能会有误差，最好连实物控制器查看
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        getchar();
        return -1;
    }
    printf("控制器连接成功! \n");

    ret =ZAux_Direct_SetAtype(handle,0,1);//设置轴 0 轴类型为 1
    commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetAtype(handle,1,1);//设置轴 1 轴类型为 1
```

```
commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//判断指令是否执行成功
ret =ZAux_Direct_SetAtype(handle,2,1);//设置轴 2 轴类型为 1
commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//判断指令是否执行成功

ret =ZAux_Direct_SetUnits(handle,0,100);//设置轴 0 脉冲当量为 100
commandCheckHandler("ZAux_Direct_SetUnits", ret) ;//判断指令是否执行成功
ret =ZAux_Direct_SetUnits(handle,1,100);//设置轴 1 脉冲当量为 100
commandCheckHandler("ZAux_Direct_SetUnits", ret) ;//判断指令是否执行成功
ret =ZAux_Direct_SetUnits(handle,2,100);//设置轴 2 脉冲当量为 100
commandCheckHandler("ZAux_Direct_SetUnits", ret) ;//判断指令是否执行成功

ret =ZAux_Direct_SetAccel(handle,0,500);//设置轴 0 加速度
commandCheckHandler("ZAux_Direct_SetAccel", ret) ;//判断指令是否执行成功
ret =ZAux_Direct_SetAccel(handle,1,500);//设置轴 1 加速度
commandCheckHandler("ZAux_Direct_SetAccel", ret) ;//判断指令是否执行成功
ret =ZAux_Direct_SetAccel(handle,2,500);//设置轴 2 加速度
commandCheckHandler("ZAux_Direct_SetAccel", ret) ;//判断指令是否执行成功

ret =ZAux_Direct_SetDecel(handle,0,500);//设置轴 0 减速度
commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//判断指令是否执行成功
ret =ZAux_Direct_SetDecel(handle,1,500);//设置轴 1 减速度
commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//判断指令是否执行成功
ret =ZAux_Direct_SetDecel(handle,2,500);//设置轴 2 减速度
commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//判断指令是否执行成功

ret =ZAux_Direct_SetDpos(handle,0,0);//设置轴 0 DPOS 清 0
commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//判断指令是否执行成功
ret =ZAux_Direct_SetDpos(handle,1,0);//设置轴 1 DPOS 清 0
commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//判断指令是否执行成功
ret =ZAux_Direct_SetDpos(handle,2,0);//设置轴 2 DPOS 清 0
commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//判断指令是否执行成功

ret =ZAux_Direct_SetSpeed(handle,0,150);//设置轴 0 速度
commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//判断指令是否执行成功
ret =ZAux_Direct_SetSpeed(handle,1,100);//设置轴 1 速度
commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//判断指令是否执行成功
ret =ZAux_Direct_SetSpeed(handle,2,100);//设置轴 2 速度
```

```
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_Connpath(handle,1,0,2);//设置开启同步连接 2，轴 2 连接插补运动主
轴，轴 0
    commandCheckHandler("ZAux_Direct_Connect", ret) ;//判断指令是否执行成功

    float IDLE;
    ZAux_Trigger( handle);//触发示波器
    int axislist[2]={0,1};
    float dposlist[2]={300,400};
    ret = ZAux_Direct_Move(handle, 2,axislist,dposlist); //插补运动 300,400,
    commandCheckHandler("ZAux_Direct_Single_Move", ret) ;//判断指令是否执行成功

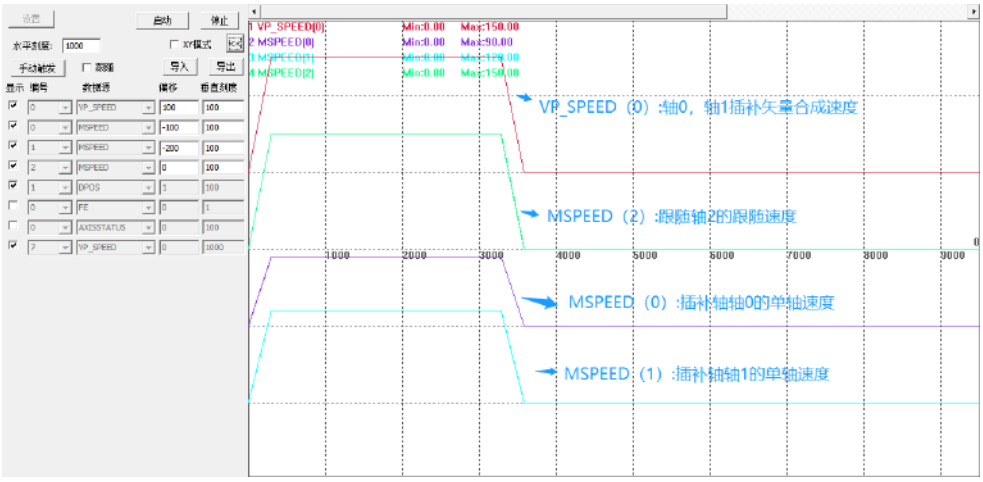
    while (1)//等待轴 0 运动完成
    {
        Sleep(100);
        ZAux_Direct_GetParam(handle,"IDLE",0,&IDLE);
        if (IDLE<0)break;
    }

    ret =ZAux_Direct_Single_Cancel(handle,0,2);//取消同步运动
    commandCheckHandler("ZAux_Direct_Connect", ret) ;//判断指令是否执行成功

    getchar();
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

插补速度曲线与跟随轴速度曲线波形图：

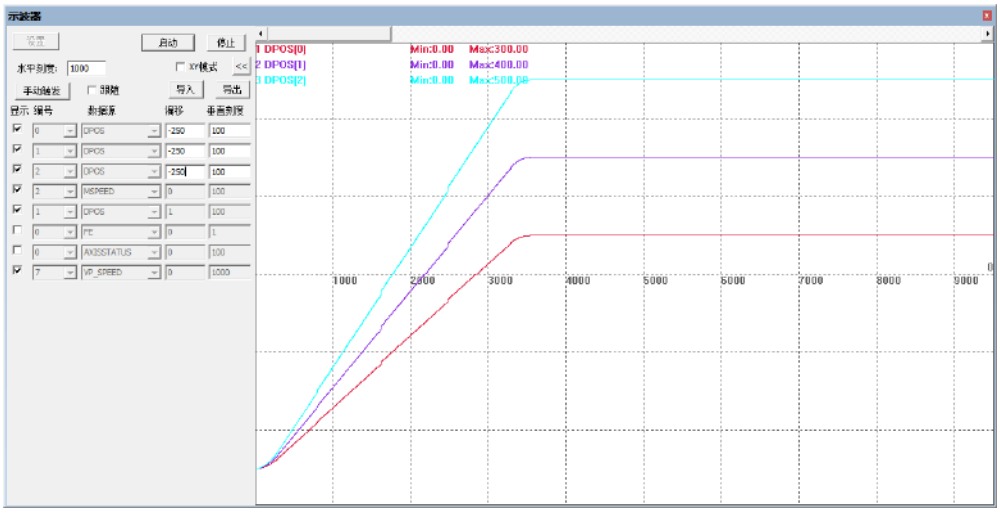
（由波形图可观察出，跟随轴是跟插补运动合成的矢量的速度变化而变化的）



插补运动与跟随轴运动轨迹曲线波形图：

（由波形图可观察出，跟随轴是跟插补运动合成的矢量的轨迹运动的。）

合成矢量距离=DPOS(2)=$\sqrt{300^2+400^2}=500$



6.4.3-3 脉冲输出电子齿轮比设置

本例程，在编码器齿轮比默认为 1 的情况下，修改脉冲输出齿轮比，把轴 0 的脉冲输出接入轴 5 的编码器输入，从而观察出轴 0 的脉冲输出的实际情况。

例程硬件接线：需把轴 0 脉冲输出与轴 5 编码器接上。

```
// test1.cpp : 定义控制台应用程序的入口点。
//
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
```

```
void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"192.168.0.181";  //控制器 IP 地址，该例程需要使用编码器查看对比，需要连接实物控制器
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        getchar();
        return -1;
    }
    printf("控制器连接成功! \n");

    ret =ZAux_Direct_SetAtype(handle,0,1);//设置轴 0 轴类型为 1
    commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetAtype(handle,5,6);//设置轴 5 轴类型脉冲编码器
    commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetUnits(handle,0,100);//设置轴 0 脉冲当量为 100
    commandCheckHandler("ZAux_Direct_SetUnits", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetUnits(handle,5,100);//设置轴 5 脉冲当量为 100
    commandCheckHandler("ZAux_Direct_SetUnits", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetAccel(handle,0,500);//设置轴 0 加速度
    commandCheckHandler("ZAux_Direct_SetAccel", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetDecel(handle,0,500);//设置轴 0 减速度
```

```
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetDpos(handle,0,0);//设置轴 0 DPOS 清 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetMpos(handle,5,0);//设置轴 5 MPOS 清 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetSpeed(handle,0,100);//设置轴 0 速度
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_StepRatio(handle,0,2,1);//设置轴 0 脉冲齿轮比为 2,即两倍脉冲输出
    commandCheckHandler("ZAux_Direct_StepRatio", ret) ;//判断指令是否执行成功
    float IDLE;
    ZAux_Trigger( handle);//触发示波器

    ret = ZAux_Direct_Single_Move(handle, 0,100); //轴 0 运动 100
    commandCheckHandler("ZAux_Direct_Single_Move", ret) ;//判断指令是否执行成功

    while (1)//等待轴 0 运动完成
    {
        Sleep(100);
        ZAux_Direct_GetParam(handle,"IDLE",0,&IDLE);
        if (IDLE<0)break;
    }

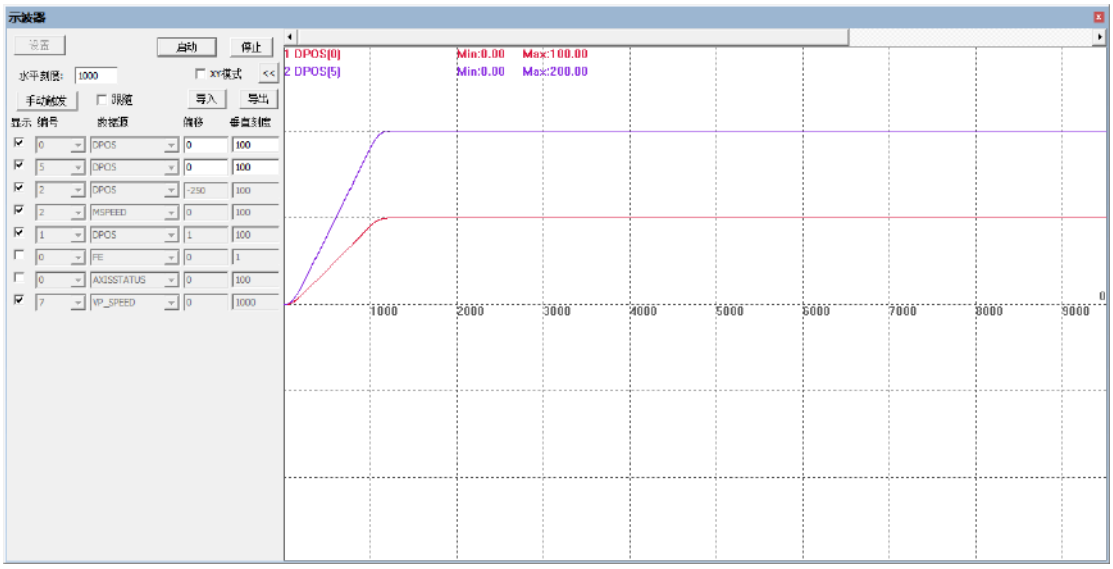
    ret =ZAux_Direct_StepRatio(handle,0,1,1);//恢复轴 0 脉冲齿轮比为 1
    commandCheckHandler("ZAux_Direct_StepRatio", ret) ;//判断指令是否执行成功

    getchar();
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

输出波形图：

由波形图可以观察出，编码器轴 5 接收到的脉冲为轴 0 的两倍，刚好符合设置脉冲输出

齿轮比。



6.4.3-4 编码器齿轮比设置

本例程，在脉冲输出齿轮比默认为 1 的情况下，修改编码器输入齿轮比，把轴 0 的脉冲输出接入轴 5 的编码器输入，从而观察出轴 5 的脉冲接收的实际情况。

例程硬件接线：需把轴 0 脉冲输出与轴 5 编码器接上。

```
// test1.cpp : 定义控制台应用程序的入口点。
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
    }

}

int _tmain(int argc, _TCHAR* argv[])
```

```
{
    char *ip_addr = (char *)"192.168.0.181";  //控制器 IP 地址，该例程需要使用编码器查看对比，需要连接实物控制器
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        getchar();
        return -1;
    }

    printf("控制器连接成功! \n");

    ret =ZAux_Direct_SetAtype(handle,0,1);//设置轴 0 轴类型为 1
    commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetAtype(handle,5,6);//设置轴 5 轴类型脉冲编码器
    commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetUnits(handle,0,100);//设置轴 0 脉冲当量为 100
    commandCheckHandler("ZAux_Direct_SetUnits", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetUnits(handle,5,100);//设置轴 5 脉冲当量为 100
    commandCheckHandler("ZAux_Direct_SetUnits", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetAccel(handle,0,500);//设置轴 0 加速度
    commandCheckHandler("ZAux_Direct_SetAccel", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetDecel(handle,0,500);//设置轴 0 减速度
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetDpos(handle,0,0);//设置轴 0 DPOS 清 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetMpos(handle,5,0);//设置轴 5 MPOS 清 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetSpeed(handle,0,100);//设置轴 0 速度
```

```
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//判断指令是否执行成功



    ret =ZAux_Direct_EncoderRatio(handle,5,2,1);//设置轴 5 脉冲齿轮比为 2,即两倍脉冲输
出
    commandCheckHandler("ZAux_Direct_EncoderRatio", ret) ;//判断指令是否执行成功

    float IDLE;
    ZAux_Trigger( handle);//触发示波器

    ret = ZAux_Direct_Single_Move(handle, 0,100); //轴 0 运动 100
    commandCheckHandler("ZAux_Direct_Single_Move", ret) ;//判断指令是否执行成功

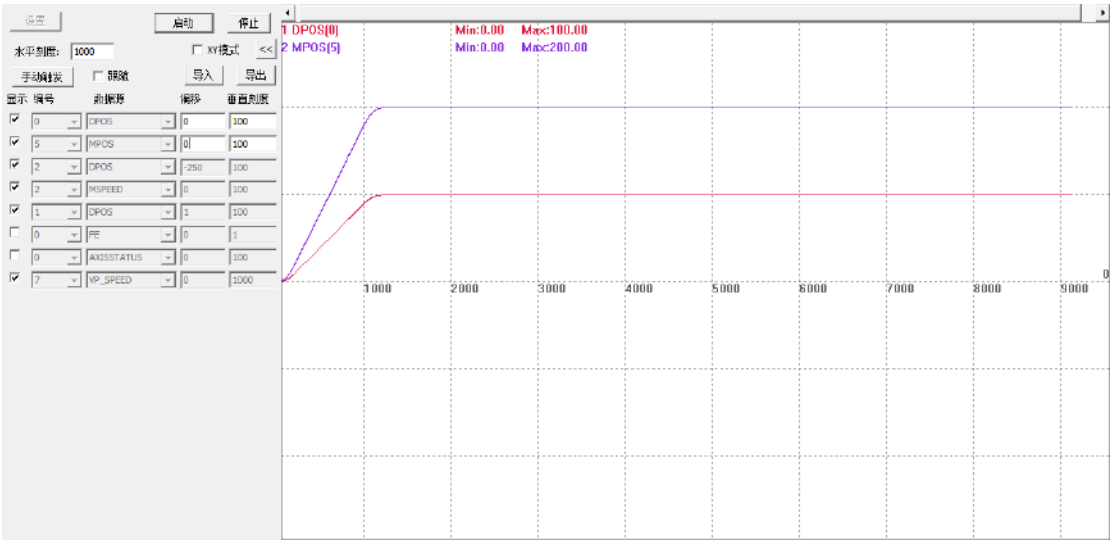
    while (1)//等待轴 0 运动完成
    {
        Sleep(100);
        ZAux_Direct_GetParam(handle,"IDLE",0,&IDLE);
        if (IDLE<0)break;
    }

    ret =ZAux_Direct_EncoderRatio(handle,5,1,1);//恢复轴 5 脉冲齿轮比为 1
    commandCheckHandler("ZAux_Direct_EncoderRatio", ret) ;//判断指令是否执行成功

    getchar();
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

输出波形图：

由波形图可以观察出，编码器轴 5 接收到的脉冲为轴 0 的两倍，刚好符合设置编码器输入齿轮比。



6.5 插补运动

6.5.1 指令列表

指令	说明
ZAux_Direct_Move	多轴相对直线插补
ZAux_Direct_MoveSp	相对多轴直线插补 SP 运动
ZAux_Direct_MoveAbs	绝对多轴直线插补
ZAux_Direct_MoveAbsSp	绝对多轴直线插补 SP 运动
ZAux_Direct_MoveCirc	相对圆心定圆弧插补运动
ZAux_Direct_MoveCircSp	相对圆心定圆弧插补运动 插补 SP 运动
ZAux_Direct_MoveCircAbs	绝对圆心圆弧插补运动
ZAux_Direct_MoveCircAbsSp	绝对圆心圆弧插补 SP 运动
ZAux_Direct_MoveCirc2	相对 3 点定圆弧插补运动
ZAux_Direct_MoveCirc2Abs	绝对 3 点定圆弧插补运动
ZAux_Direct_MoveCirc2Sp	相对 3 点定圆弧插补 SP 运动
ZAux_Direct_MoveCirc2AbsSp	绝对 3 点定圆弧插补 SP 运动
ZAux_Direct_MHelical	相对 3 轴 圆心画螺旋插补运动
ZAux_Direct_MHelicalAbs	绝对 3 轴 圆心画螺旋插补运动
ZAux_Direct_MHelicalSp	相对 3 轴 圆心画螺旋插补 SP 运动
ZAux_Direct_MHelicalAbsSp	绝对 3 轴 圆心画螺旋插补 SP 运动
ZAux_Direct_MHelical2	相对 3 轴 3 点画螺旋插补运动
ZAux_Direct_MHelical2Abs	绝对 3 轴 3 点画螺旋插补运动

ZAux_Direct_MHelical2Sp	相对 3 轴 3 点画螺旋插补 SP 运动
ZAux_Direct_MHelical2AbsSp	绝对 3 轴 3 点画螺旋插补 SP 运动
ZAux_Direct_MEclipse	相对椭圆插补运动
ZAux_Direct_MEclipseAbs	绝对椭圆插补运动
ZAux_Direct_MEclipseSp	相对椭圆插补 SP 运动
ZAux_Direct_MEclipseAbsSp	绝对椭圆插补 SP 运动
ZAux_Direct_MEclipseHelical	相对椭圆 + 螺旋插补运动
ZAux_Direct_MEclipseHelicalAbs	绝对椭圆 + 螺旋插补运动
ZAux_Direct_MEclipseHelicalSp	相对椭圆 + 螺旋插补 SP 运动
ZAux_Direct_MEclipseHelicalAbsSp	绝对椭圆 + 螺旋插补 SP 运动
ZAux_Direct_MSpherical	空间圆弧 + 螺旋插补运动
ZAux_Direct_MSphericalSp	空间圆弧 + 螺旋 插补 SP 运动
ZAux_Direct_MSphericalABSSp	空间圆弧 + 螺旋 插补 ABSSP 运动
ZAux_Direct_MoveSpiral	渐开线圆弧插补运动，相对移动方式
ZAux_Direct_MoveSpiralSp	渐开线圆弧插补 SP 运动，相对移动方式
ZAux_Direct_MoveSmooth	空间直线运动
ZAux_Direct_MoveSmoothSp	空间直线插补 SP 运动
ZAux_Direct_MoveTurnabs	旋转台直线插补运动
ZAux_Direct_McircTurnabs	旋转台圆弧 + 螺旋插补运动
ZAux_Direct_MultiMove	多条相对多轴直线插补
ZAux_Direct_MultiMoveAbs	多条绝对多轴直线插补
ZAux_Direct_MultiLineN	多轴多段线直线连续插补

指令	说明
ZAux_Direct_SetFhspeed	设置轴保持速度
ZAux_Direct_GetFhspeed	读取轴保持速度
ZAux_Direct_SetEndMoveSpeed	设置 SP 运动的结束速度
ZAux_Direct_GetEndMoveSpeed	读取 SP 运动结束速度
ZAux_Direct_SetForceSpeed	设置 SP 运动的运行速度
ZAux_Direct_GetForceSpeed	读取 SP 运动的运行速度
ZAux_Direct_SetStartMoveSpeed	设置自定义速度的 SP 运动的起始速度
ZAux_Direct_GetStartMoveSpeed	读取自定义速度的 SP 运动的起始速度
ZAux_Direct_SetFholdIn	设置保持输入对应的输入点编号，-1 无效。
ZAux_Direct_GetFholdIn	读取保持输入对应的输入点编号，-1 无效。

指令	说明
ZAux_Direct_SetCornerMode	设置拐角减速模式
ZAux_Direct_GetCornerMode	读取拐角减速模式
ZAux_Direct_SetDecelAngle	设置拐角开始减速角度,单位为弧度
ZAux_Direct_GetDecelAngle	读取拐角开始减速角度,单位为弧度
ZAux_Direct_SetFullSpRadius	设置小圆限速最小半径
ZAux_Direct_GetFullSpRadius	读取小圆限速最小半径
ZAux_Direct_SetInterpFactor	设置插补时轴是否参与速度计算
ZAux_Direct_GetInterpFactor	读取插补时轴是否参与速度计算
ZAux_Direct_SetMerge	设置连续插补状态
ZAux_Direct_GetMerge	读取连续插补状态
ZAux_Direct_SetStopAngle	设置减速到最低的最小拐角 弧度制
ZAux_Direct_GetStopAngle	读取减速到最低的最小拐角 弧度制
ZAux_Direct_SetZsmooth	设置自动倒角半径
ZAux_Direct_GetZsmooth	读取自动倒角半径

6.5.2 重点说明

6.5.2-1 SP 速度

SP 和非 SP 运动的区别

通过 **ZAux_Direct_SetSpeed** 设置轴速度，速度立即改变，可以实现动态变速，而所有的插补运动都有对应的自定义速度的 SP 运动，如 **ZAux_Direct_MoveSp**，**ZAux_Direct_MoveAbsSp** 等等。SP 运动的速度不再受全局速度 **ZAux_Direct_SetSpeed** 的控制，而是通过专门的速度指令设置对应的起始、运行、结束速度，这些速度也是随着运动一起进入运动缓冲的。

SP 速度详解

SP 速度,代表着可以进行速度控制的函数。正运动控制卡指令后缀带有 SP 的，即表述为该运动可以进行自定义的速度控制。

该章节函数参数设置主要应用于插补运动指令带 SP 后缀之后的函数，此时运动速度采用设置的 **ZAux_Direct_SetForceSpeed** 参数速度，而不是设置的 **ZAux_Direct_SetSpeed** 速度参数。SP 速度应用于插补运动指令带 SP 后缀之后的指令（例如 **ZAux_Direct_MoveSp**、**ZAux_Direct_MoveCircSp**）。

ZAux_Direct_SetStartMoveSpeed 设置的起始速度：自定义速度的 SP 运动的开始速度。

ZAux_Direct_SetEndMoveSpeed 设置的结束速度：自定义速度的 SP 运动的结束速度。

ZAux_Direct_SetForceSpeed 设置的强制速度：自定义速度的 SP 运动的强制速度。

注意：以上三个参数只有使用了带 SP 的运动指令才生效，参数都被带入运动缓冲。不使用时请将设置起始速度的指令 **ZAux_Direct_SetStartMoveSpeed** 和设置结束速度的指令 **ZAux_Direct_SetEndMoveSpeed** 设置较大值，否则下一段运动指令还会沿用此参数。

6.5.2-2 连续插补与轨迹前瞻

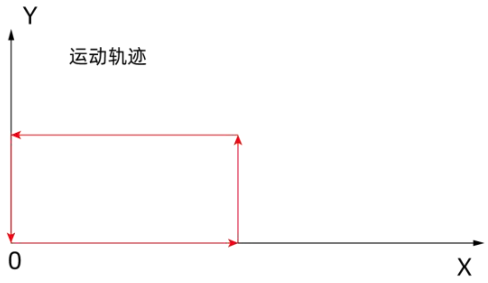
在实际加工过程中，为追求加工效率会开启连续插补，运动轨迹的拐角处若不减速，当拐角较大时，会对机台造成较大冲击，影响加工精度。若关闭连续插补，使拐角处减速为 0，虽然保护了机台，但是加工效率受到了较大影响，所以提供了前瞻指令，使在拐角处自动判断是否将拐角速度降到一个合理的值，既不会影响加工精度又能提高加工的速度，这就是轨迹前瞻功能的作用。

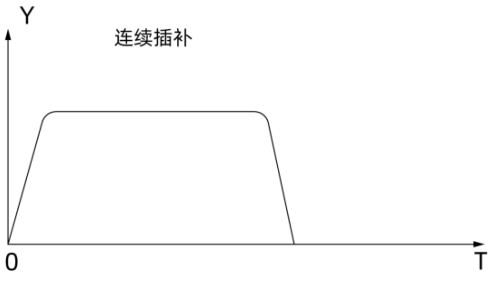
运动控制器的轨迹前瞻可以根据用户的运动路径自动计算出平滑的速度规划，减少机台的冲击，从而提高加工精度。自动分析在运动缓冲区的指令轨迹将会出现的拐点，并依据用户设置的拐角条件，自动计算拐角处的运动速度，也会依据用户设定的最大加速度值计算速度规划，使任何加减速过程中的速度值都不超过 **ZAux_Direct_SetAccel** 和 **ZAux_Direct_SetDecel** 的值，防止对机械部分产生破坏冲击力。

使用轨迹前瞻和不使用轨迹前瞻的速度规划情况：

假设运动轨迹如下左图，走一个长方形轨迹，分为四段直线插补运动。

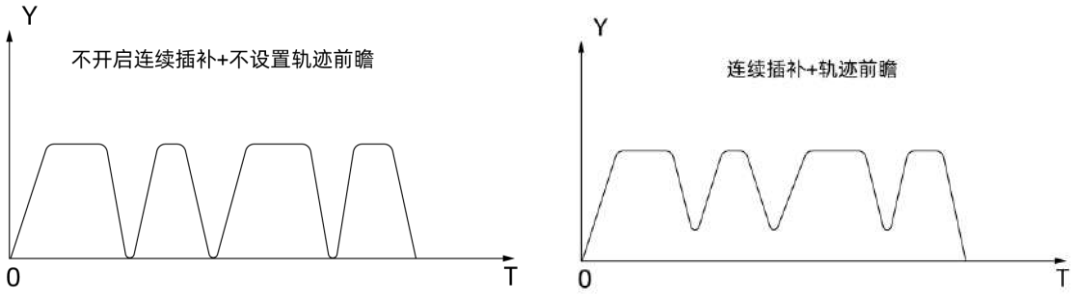
模式一：开启了连续插补后，得出的主轴速度随时间变化的曲线如下右图，主轴的速度是连续的，轨迹拐角处仍不减速，高速运行时拐角处冲击较大。





模式二：模式一条件下，关闭连续插补，得出的主轴速度随时间变化的曲线如下左图，每走完一段直线后便减速到 0 再开始第二段直线运动，加工效率不高。

模式三：模式一条件下，开启连续插补，并设置了轨迹前瞻参数，得出的主轴速度随时间变化的曲线如下右图，拐角处按照一定的比例减速，加工效率比模式二高。



以上模式若希望速度曲线更柔和，只需通过 **ZAux_Direct_SetSramp** 指令设置速度为 S 曲线。

轨迹前瞻的主要指令 **ZAux_Direct_SetCornerMode**，用于拐角处的速度规划，常用的三种模式，根据加工的轨迹的实际要求选择不同的模式。

此参数是在插补运动指令调用前生效，一般在参数初始化中设置拐角模式。因为前瞻运动参数会随着运动指令一并存入运动缓冲区，前瞻运动参数可以多次调用，不同模式也可以混合使用，例如 **ZAux_Direct_SetCornerMode**=2+8，表示同时使用自动拐角减速和小圆限速，根据轨迹段的要求设置合适的前瞻模式，在执行运动指令的时候自动优化轨迹。

注意：ZAux_Direct_SetCornerMode 模式一旦设置了参数就会存入控制器内，如果要取消设置的参数，就需要设置 **Zaux_Direct_SetCornerMode** =0 ,否则下次运行时，之前设置的 **Zaux_Direct_SetCornerMode** 仍会生效。

CORNER_MODE 指令参数说明：

位	值	描述
0	1	预留
1	2	自动拐角减速。 按 **ZAux_Direct_SetAccel 和 ZAux_Direct_SetDecel** 生效加减速度。 此参数是在运动函数调用前生效。 减速角度根**据 ZAux_Direct_SetDecelAngle** 和 **ZAux_Direct_SetStopAngle** 函数设定。 减速拐角参考速度以 FORCE_SPEED 速度为参考，一定要设置合理的 **ZAux_Direct_SetForceSpeed**。
2	4	预留
3	8	自动小圆限速，半径小于设置值时限速，大于限制值时不限速。 此参数在运动函数调用前修改生效。 限制速度与 **ZAux_Direct_SetForceSpeed** 有关。 限制速度 =**ZAux_Direct_SetForceSpeed** * 实际半径 /**ZAux_Direct_SetFullSpRadius** 限速半径 **ZAux_Direct_SetFullSpRadius** 设置。
4	16	预留

5	32	自动倒角设置。 此参数在直线运动函数调用前修改生效。 倒角的半径设置值，并不为实际值，只有 90 度时才是。 此直线运动自动和前面的直线运动做倒角处理，倒角半径参考 **ZAux_Direct_SetZsmooth**。

ZAux_Direct_SetCornerMode=2 拐角减速应用场合：不改变运动轨迹，仅在拐角处自动判断是否减速，一般用于改善机台抖动的问题，对轨迹精度有要求，对速度要求没那么快的场合。

ZAux_Direct_SetCornerMode=8 小圆限速应用场合：不改变运动轨迹，一般应用在圆弧加工，根据圆弧半径计算当前圆弧的限制速度。

ZAux_Direct_SetCornerMode=32 自动倒角应用场合：改变运动轨迹，不会降低速度，针对轨迹拐角较大的场合，倒角处运动轨迹做自动平滑处理，所以一般应用在对速度要求快，对轨迹精度要求不高的场合。为了方便说明每个位的功能，例程都是单独位设置，实际应用时可以多位同时使用。比如 **ZAux_Direct_SetCornerMode**=2+8，打开自动拐角减速和小圆限速。

6.5.3 例程

6.5.3-1 拐角减速

开启拐角减速后，若设置自定义速度（SP）运动指令的 STARTMOVE_SPEED(SP 指令起始速度 ZAux_Direct_SetStartMoveSpeed) 或 ENDMOVE_SPEED(SP 指令结束速度 ZAux_Direct_SetEndMoveSpeed)参数，则拐角减速失效，拐角处的速度按以上两个 SP 速度指令设置的大小运动。本例子开启了拐角减速模式 2 和连续插补，设置减速角度 15-45 度之间，三段直线插补轨迹。

```
// test1.cpp : 定义控制台应用程序的入口点。
//
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
#define PI 3.1415926


void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
```

```
    {
        printf("%s fail!return code is %d\n", command, ret);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"192.168.0.181";  //控制器 IP 地址，使用仿真器查看时曲线可能会有误差，最好连实物控制器查看
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        getchar();
        return -1;
    }
    printf("控制器连接成功! \n");

    ret =ZAux_Direct_SetAtype(handle,0,1);//设置轴 0 轴类型为 1
    commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetAtype(handle,1,1);//设置轴 1 轴类型为 1
    commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetUnits(handle,0,100);//设置轴 0 脉冲当量为 100
    commandCheckHandler("ZAux_Direct_SetUnits", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetUnits(handle,1,100);//设置轴 1 脉冲当量为 100
    commandCheckHandler("ZAux_Direct_SetUnits", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetAccel(handle,0,500);//设置轴 0 加速度
    commandCheckHandler("ZAux_Direct_SetAccel", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetAccel(handle,1,500);//设置轴 1 加速度
    commandCheckHandler("ZAux_Direct_SetAccel", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetDecel(handle,0,500);//设置轴 0 减速度
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//判断指令是否执行成功
```

```
    ret =ZAux_Direct_SetDecel(handle,1,500);//设置轴 1 减速度
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetDpos(handle,0,0);//设置轴 0 DPOS 清 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetDpos(handle,1,0);//设置轴 1 DPOS 清 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetSpeed(handle,0,100);//设置轴 0 速度
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetSpeed(handle,1,100);//设置轴 1 速度
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetMerge(handle,0,1);//设置开启连续插补（开启主轴的即可，如轴
0，轴 1 插补，轴 0 为主轴，主轴号取决于连续插补运动指令轴列表的第一个轴号）
    commandCheckHandler("ZAux_Direct_SetMerge", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetCornerMode(handle,0,2);//设置开启拐角模式（开启主轴的即可，
如轴 0，轴 1 插补，轴 0 为主轴，主轴号取决于连续插补运动指令轴列表的第一个轴号）
    commandCheckHandler("ZAux_Direct_SetCornerMode", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetDecelAngle(handle,0,15 * (PI/180));//设置开始减速角度（开启
主轴的即可，如轴 0，轴 1 插补，轴 0 为主轴，主轴号取决于连续插补运动指令轴列表的第一个
轴号）
    commandCheckHandler("ZAux_Direct_SetDecelAngle", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetStopAngle(handle,0,45 * (PI/180));//设置结束减速角度（开启主
轴的即可，如轴 0，轴 1 插补，轴 0 为主轴，主轴号取决于连续插补运动指令轴列表的第一个轴
号）
    commandCheckHandler("ZAux_Direct_SetStopAngle", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetForceSpeed(handle,0,100);//设置等比减速时的速度（开启主轴的
即可，如轴 0，轴 1 插补，轴 0 为主轴，主轴号取决于连续插补运动指令轴列表的第一个轴号）
    commandCheckHandler("ZAux_Direct_SetForceSpeed", ret) ;//判断指令是否执行成功

    float IDLE;
```

```
    float DposList[2]={0};
    int Axislist[2]={0,1};

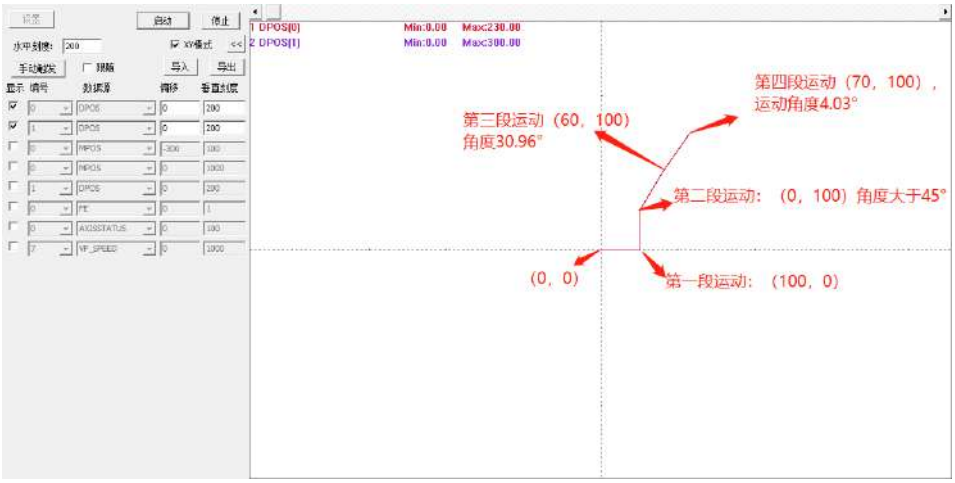
    ZAux_Trigger( handle);//触发示波器
    DposList[0]=100;
    DposList[1]=0;
    ret =ZAux_Direct_Move(handle,2,Axislist,DposList);//插补运动（100，0）
    commandCheckHandler("ZAux_Direct_SetForceSpeed", ret) ;//判断指令是否执行成功
    DposList[0]=0;
    DposList[1]=100;
    ret =ZAux_Direct_Move(handle,2,Axislist,DposList);//插补运动（0，100），运动角度
大于 45°，完全减速
    commandCheckHandler("ZAux_Direct_SetForceSpeed", ret) ;//判断指令是否执行成功
    DposList[0]=60;
    DposList[1]=100;
    ret =ZAux_Direct_Move(handle,2,Axislist,DposList);//插补运动（60，100），运动角
度 30.96° 处于 15° ~45°，等比减速
    commandCheckHandler("ZAux_Direct_SetForceSpeed", ret) ;//判断指令是否执行成功
    DposList[0]=70;
    DposList[1]=100;
    ret =ZAux_Direct_Move(handle,2,Axislist,DposList);//插补运动（70，100），运动角
度 4.03° 小于 15°，不减速
    commandCheckHandler("ZAux_Direct_SetForceSpeed", ret) ;//判断指令是否执行成功

    while (1)//等待轴 0 运动完成
    {
        Sleep(100);
        ZAux_Direct_GetParam(handle,"IDLE",0,&IDLE);
        if (IDLE<0)break;
    }

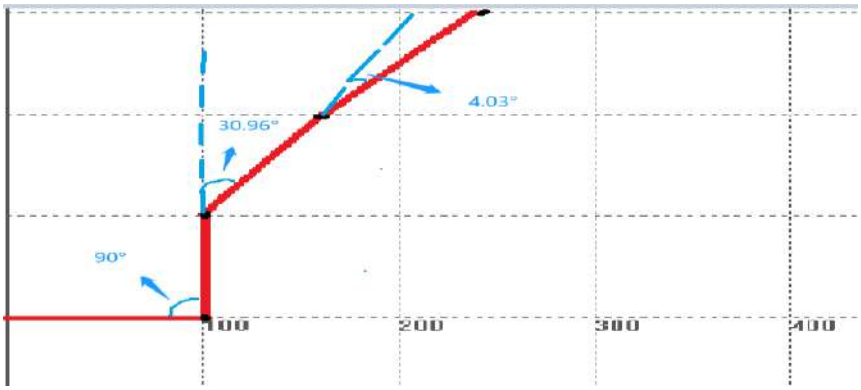
    getchar();
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret);    //判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
```

```
}
```

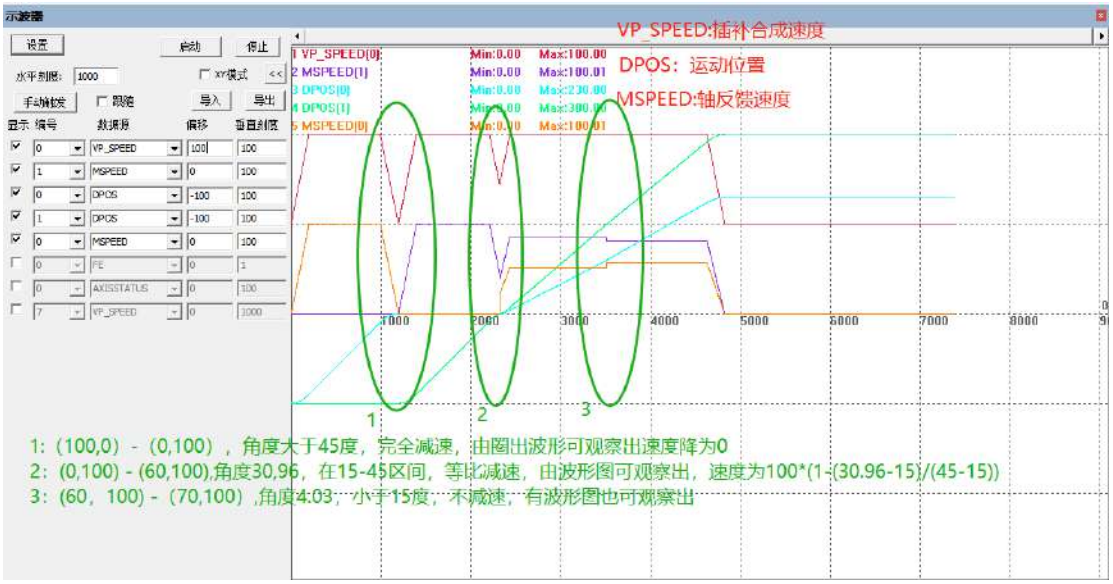
轨迹曲线图：



角度分析：



速度曲线图：



6.5.3-2 平面圆心画圆+小圆限速

小圆限制为 60，当半径大于或等于限制值时，不限制速度，按 SPEED 运行此时速度为 100。

```
// test1.cpp : 定义控制台应用程序的入口点。
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
#define PI 3.1415926

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"192.168.0.181";  //控制器 IP 地址，使用仿真器查看时曲线可能会有误差，最好连实物控制器查看
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        getchar();
        return -1;
    }
    printf("控制器连接成功! \n");

    ret =ZAux_Direct_SetAtype(handle,0,1);//设置轴 0 轴类型为 1
    commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//判断指令是否执行成功
```

```
    ret =ZAux_Direct_SetAtype(handle,1,1);//设置轴 1 轴类型为 1
    commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetUnits(handle,0,100);//设置轴 0 脉冲当量为 100
    commandCheckHandler("ZAux_Direct_SetUnits", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetUnits(handle,1,100);//设置轴 1 脉冲当量为 100
    commandCheckHandler("ZAux_Direct_SetUnits", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetAccel(handle,0,500);//设置轴 0 加速度
    commandCheckHandler("ZAux_Direct_SetAccel", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetAccel(handle,1,500);//设置轴 1 加速度
    commandCheckHandler("ZAux_Direct_SetAccel", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetDecel(handle,0,500);//设置轴 0 减速度
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetDecel(handle,1,500);//设置轴 1 减速度
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetDpos(handle,0,0);//设置轴 0 DPOS 清 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetDpos(handle,1,0);//设置轴 1 DPOS 清 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetSpeed(handle,0,100);//设置轴 0 速度
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetSpeed(handle,1,100);//设置轴 1 速度
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetMerge(handle,0,1);//设置开启连续插补（开启主轴的即可，如轴
0，轴 1 插补，轴 0 为主轴，主轴号取决于连续插补运动指令轴列表的第一个轴号）
    commandCheckHandler("ZAux_Direct_SetMerge", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetCornerMode(handle,0,8);//设置开启小圆限速（开启主轴的即可，
如轴 0，轴 1 插补，轴 0 为主轴，主轴号取决于连续插补运动指令轴列表的第一个轴号）
    commandCheckHandler("ZAux_Direct_SetCornerMode", ret) ;//判断指令是否执行成功
```

```
    ret =ZAux_Direct_SetFullSpRadius(handle,0,60);//设置开始减速角度（设置主轴的即可，如轴 0，轴 1 插补，轴 0 为主轴，主轴号取决于连续插补运动指令轴列表的第一个轴号）
    commandCheckHandler("ZAux_Direct_SetFullSpRadius", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetForceSpeed(handle,0,100);//设置等比减速时的速度（设置主轴的即可，如轴 0，轴 1 插补，轴 0 为主轴，主轴号取决于连续插补运动指令轴列表的第一个轴号）
    commandCheckHandler("ZAux_Direct_SetForceSpeed", ret) ;//判断指令是否执行成功

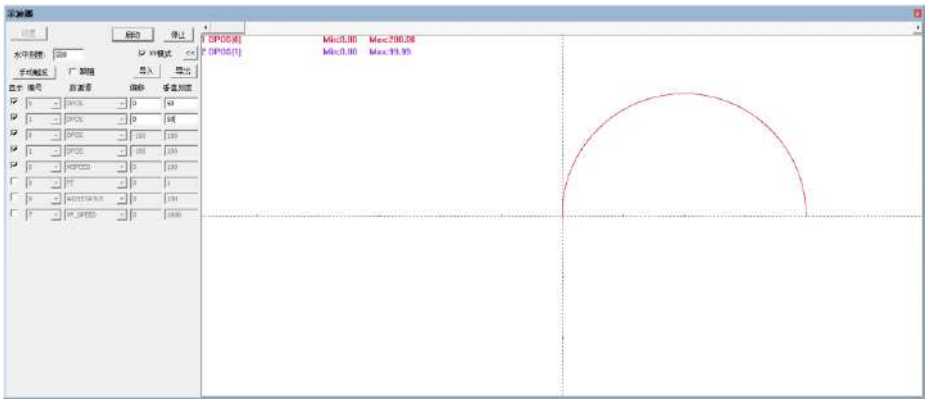
    float IDLE;
    int Axislist[2]={0,1};

    ZAux_Trigger( handle);//触发示波器
    //未运动前坐标为 0,0
    //插补轨迹终点坐标为 100,0
    //圆心坐标为 50,0，
    //由以上可知半径为 50
    ret =ZAux_Direct_MoveCirc(handle,2,Axislist,100,0,50,0,1);//圆弧插补插补运动
    commandCheckHandler("ZAux_Direct_MoveCirc", ret) ;//判断指令是否执行成功

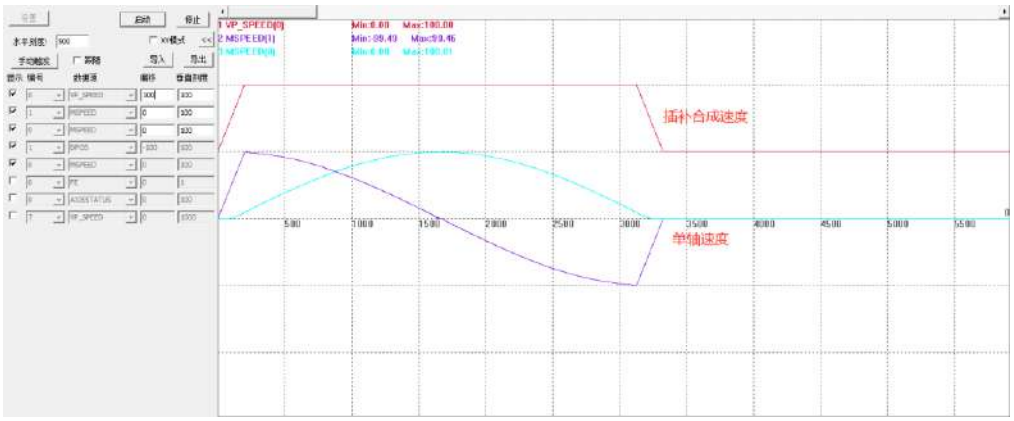
    while (1)//等待轴 0 运动完成
    {
        Sleep(100);
        ZAux_Direct_GetParam(handle,"IDLE",0,&IDLE);
        if (IDLE<0)break;
    }

    getchar();
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

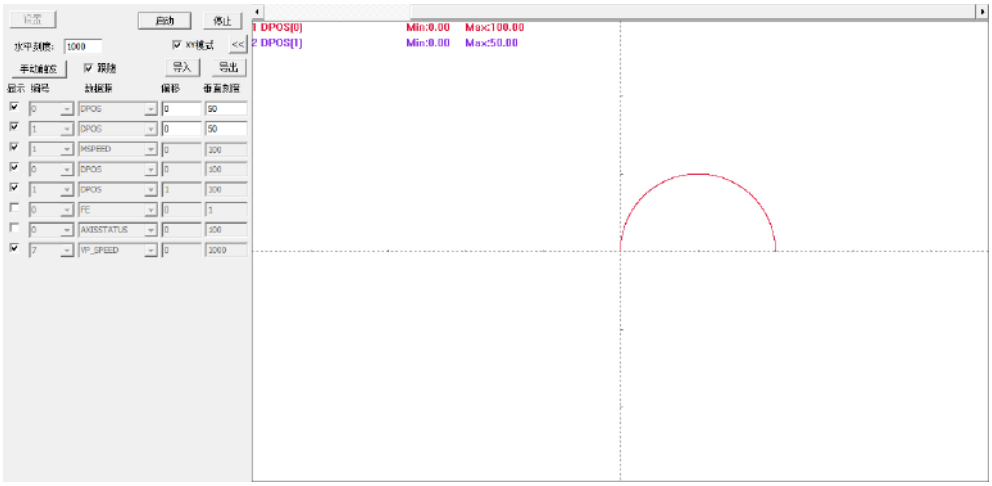
半径为 100，100>60，轨迹波形图：



设置半径为 100，100>60，速度波形图（完全不减速）：



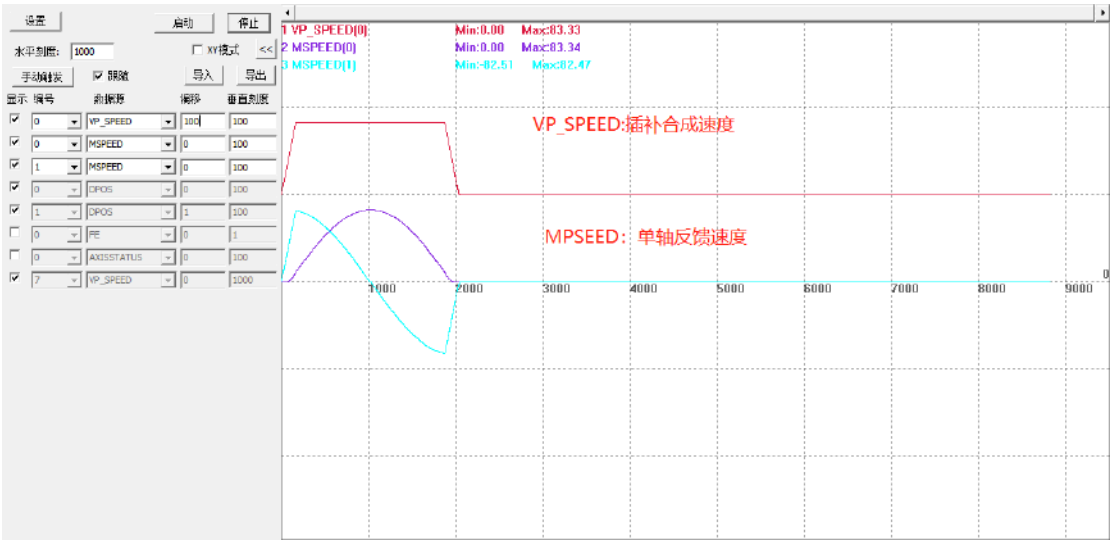
半径为 50，50<60，轨迹波形图：



半径为 50，50<60 速度波形图（小圆减速）：

//半径小于等于限制值时，限速=FORCE_SPEED * 实际半径/FULL_SP_RADIUS

//速度=100*50/60=83.33



6.5.3-3 倒角

倒角是可以用于直线、圆弧、螺旋等运动的，例程为了直观显示，只做了直线拐角。根据拐角角度自动计算实际拐角半径大小。超过路径时限制为 50%。(例:走两段位移,(100,0),(0,100)则倒角半径最大值为 50)

```
// test1.cpp : 定义控制台应用程序的入口点。
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
#define PI 3.1415926

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
    }

}

int _tmain(int argc, _TCHAR* argv[])
{
```

```
    char *ip_addr = (char *)"192.168.0.181";  //控制器 IP 地址，使用仿真器查看时曲线可能会有误差，最好连实物控制器查看
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        getchar();
        return -1;
    }
    printf("控制器连接成功! \n");

    ret =ZAux_Direct_SetAtype(handle,0,1);//设置轴 0 轴类型为 1
    commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetAtype(handle,1,1);//设置轴 1 轴类型为 1
    commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetUnits(handle,0,100);//设置轴 0 脉冲当量为 100
    commandCheckHandler("ZAux_Direct_SetUnits", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetUnits(handle,1,100);//设置轴 1 脉冲当量为 100
    commandCheckHandler("ZAux_Direct_SetUnits", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetAccel(handle,0,500);//设置轴 0 加速度
    commandCheckHandler("ZAux_Direct_SetAccel", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetAccel(handle,1,500);//设置轴 1 加速度
    commandCheckHandler("ZAux_Direct_SetAccel", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetDecel(handle,0,500);//设置轴 0 减速度
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetDecel(handle,1,500);//设置轴 1 减速度
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetDpos(handle,0,0);//设置轴 0 DPOS 清 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetDpos(handle,1,0);//设置轴 1 DPOS 清 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//判断指令是否执行成功
```

```
    ret =ZAux_Direct_SetSpeed(handle,0,100);//设置轴 0 速度
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetSpeed(handle,1,100);//设置轴 1 速度
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetMerge(handle,0,1);//设置开启连续插补（开启主轴的即可，如轴
0，轴 1 插补，轴 0 为主轴，主轴号取决于连续插补运动指令轴列表的第一个轴号）
    commandCheckHandler("ZAux_Direct_SetMerge", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetCornerMode(handle,0,32);//设置自动倒角（开启主轴的即可，如轴
0，轴 1 插补，轴 0 为主轴，主轴号取决于连续插补运动指令轴列表的第一个轴号）
    commandCheckHandler("ZAux_Direct_SetCornerMode", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetZsmooth(handle,0,50);//设置倒角半径（设置主轴的即可，如轴 0，
轴 1 插补，轴 0 为主轴，主轴号取决于连续插补运动指令轴列表的第一个轴号）
    commandCheckHandler("ZAux_Direct_SetZsmooth", ret) ;//判断指令是否执行成功

    float IDLE;
    int Axislist[2]={0,1};
    float Dposlist[2]={0};
    ZAux_Trigger( handle);//触发示波器

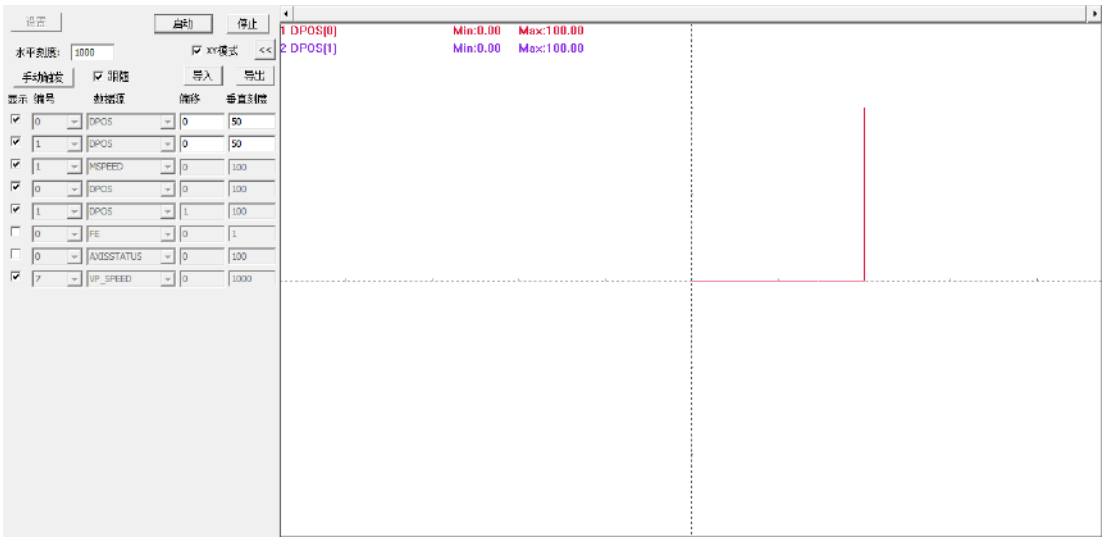
    Dposlist[0]=100;
    Dposlist[1]=0;
    ret =ZAux_Direct_Move(handle,2,Axislist,Dposlist);//直线插补运动
    commandCheckHandler("ZAux_Direct_MoveCirc", ret) ;//判断指令是否执行成功
    Dposlist[0]=0;
    Dposlist[1]=100;
    ret =ZAux_Direct_Move(handle,2,Axislist,Dposlist);//直线插补运动
    commandCheckHandler("ZAux_Direct_MoveCirc", ret) ;//判断指令是否执行成功

    while (1)//等待轴 0 运动完成
    {
        Sleep(100);
        ZAux_Direct_GetParam(handle,"IDLE",0,&IDLE);
```

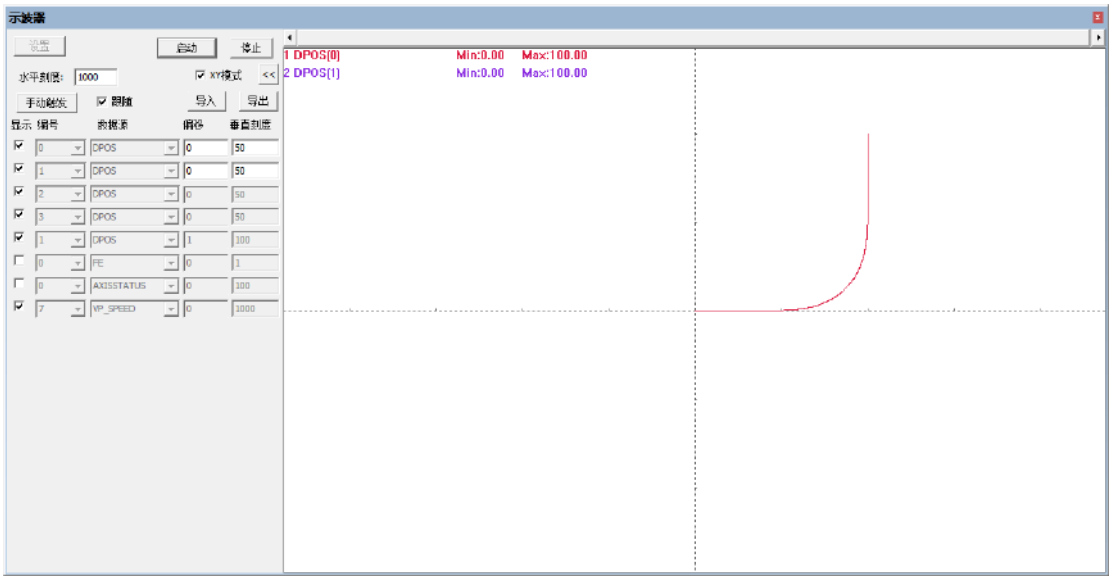
```
        if (IDLE<0)break;
    }


    getchar();
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

插补轨迹，未使用倒角前，轨迹波形图：



插补轨迹，使用倒角后，倒角半径设置为该轨迹的最大值 50，轨迹波形图：



6.5.3-4 轴保持输入应用

在轴 0 运动 1500 的一个 SP 点位运动，期间碰到 IN（0）的信号时，速度自动切换成保持速度。

```
// test1.cpp : 定义控制台应用程序的入口点。
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";                //控制器 IP 地址
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        getchar();
        return -1;
    }
    printf("控制器连接成功! \n");

    ret =ZAux_Direct_SetAtype(handle,0,1);//设置轴类型为 1
    commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetAccel(handle,0,500);//设置轴加速度
```

```
    commandCheckHandler("ZAux_Direct_SetAccel", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetDecel(handle,0,500);//设置轴减速度
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetDpos(handle,0,0);//设置轴 DPOS 清 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetForceSpeed(handle,0,200);//设置轴 SP 运行速度
    commandCheckHandler("ZAux_Direct_SetSpeed", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetFhspeed(handle,0,100);//设置保持速度为 100
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetFholdIn(handle,0,0);//设置保持速度映射 IN 为 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetInvertIn(handle,0,1);//设置保持速度映射 IN 电平反转
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//判断指令是否执行成功

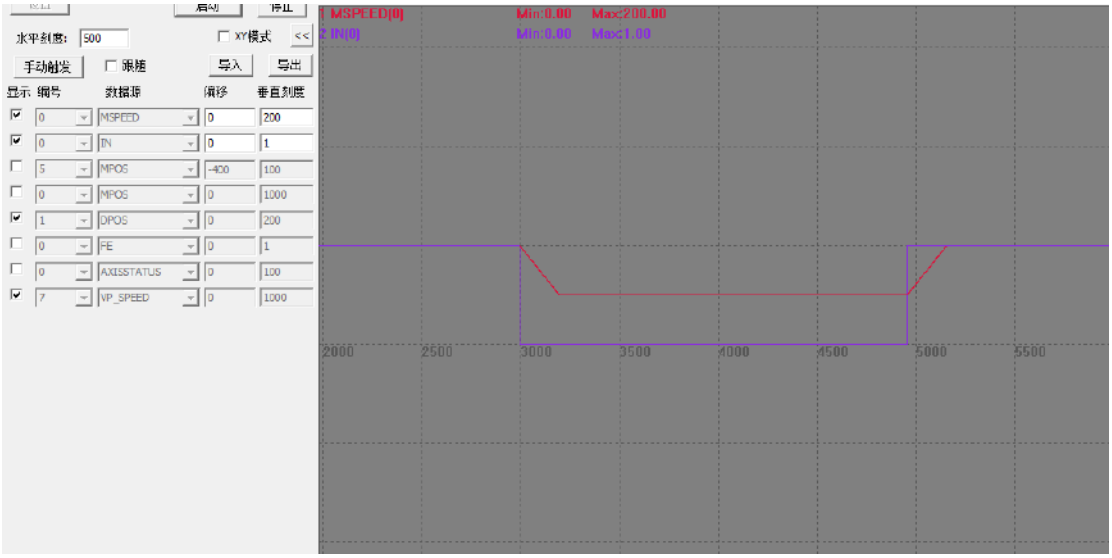
    float IDLE;
    int axislist=0;
    float dposlist=1500;
    ZAux_Trigger( handle);//触发示波器
    ZAux_Direct_MoveSp(handle,1,&axislist,&dposlist);//轴 0 运动 1500
    while (1)//等待轴 0 运动完成
    {
        Sleep(100);
        ZAux_Direct_GetParam(handle,"IDLE",0,&IDLE);
        if (IDLE<0)break;
    }

    getchar();
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

速度曲线图：

MSPEED（编码器反馈实时速度）：垂直刻度 200

IN（0）：垂直刻度 1



6.5.3-5 连续插补 SP 速度运动控制

```
// test1.cpp : 定义控制台应用程序的入口点。
//
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";                //控制器 IP 地址
    ZMC_HANDLE handle = NULL;                           //连接句柄
```

```
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器

    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        getchar();
        return -1;
    }
    printf("控制器连接成功! \n");

    ret =ZAux_Direct_SetAtype(handle,0,1);//设置轴类型为 1
    commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetAccel(handle,0,500);//设置轴加速度
    commandCheckHandler("ZAux_Direct_SetAccel", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetDecel(handle,0,500);//设置轴减速度
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetDpos(handle,0,0);//设置轴 DPOS 清 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetMerge(handle, 0,1);//设置连续插补
    commandCheckHandler("ZAux_Direct_SetMerge", ret) ;//判断指令是否执行成功

    float IDLE;
    int axislist=0;
    float dposlist=40;
    ZAux_Trigger( handle);//触发示波器

    //第一段

    ret =ZAux_Direct_SetForceSpeed(handle,0,30);//设置轴 SP 运行速度
    commandCheckHandler("ZAux_Direct_SetSpeed", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetStartMoveSpeed(handle,0,1000);//取消设置轴开始运行速度，设置
值大于 ForceSpeed 即可
    commandCheckHandler("ZAux_Direct_SetStartMoveSpeed", ret) ;//判断指令是否执行成
功
```

```
    ret =ZAux_Direct_SetEndMoveSpeed(handle,0,1000);//取消设置轴结束速度，设置值大于
ForceSpeed 即可
    commandCheckHandler("ZAux_Direct_SetEndMoveSpeed", ret) ;//判断指令是否执行成功

    ZAux_Direct_MoveSp(handle,1,&axislist,&dposlist);//轴 0 运动 40
    //第二段

    ret =ZAux_Direct_SetForceSpeed(handle,0,50);//设置轴 SP 运行速度
    commandCheckHandler("ZAux_Direct_SetSpeed", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetStartMoveSpeed(handle,0,20);//设置轴开始运行速度 20
    commandCheckHandler("ZAux_Direct_SetStartMoveSpeed", ret) ;//判断指令是否执行成
功

    ret =ZAux_Direct_SetEndMoveSpeed(handle,0,40);//设置轴结束运行速度 40
    commandCheckHandler("ZAux_Direct_SetEndMoveSpeed", ret) ;//判断指令是否执行成功

    dposlist=50;
    ZAux_Direct_MoveSp(handle,1,&axislist,&dposlist);//轴 0 运动 50
    //第三段

    ret =ZAux_Direct_SetForceSpeed(handle,0,60);//设置轴 SP 运行速度
    commandCheckHandler("ZAux_Direct_SetSpeed", ret) ;//判断指令是否执行成功

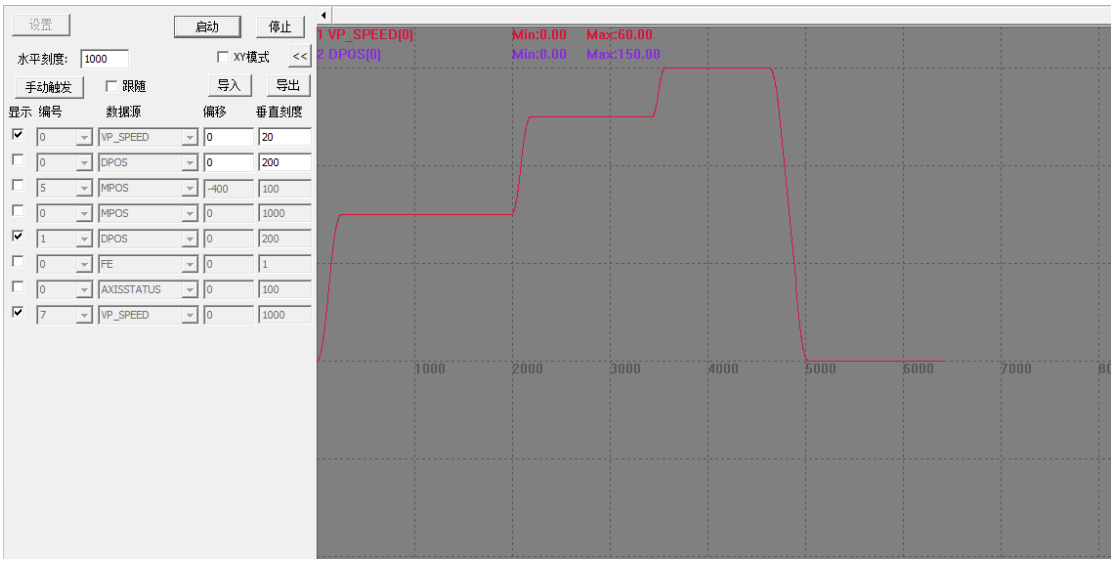
    ret =ZAux_Direct_SetStartMoveSpeed(handle,0,1000);//取消设置轴开始运行速度，设置
值大于 ForceSpeed 即可
    commandCheckHandler("ZAux_Direct_SetStartMoveSpeed", ret) ;//判断指令是否执行成
功

    ret =ZAux_Direct_SetEndMoveSpeed(handle,0,1000);//取消设置轴开始运行速度，设置值
大于 ForceSpeed 即可
    commandCheckHandler("ZAux_Direct_SetEndMoveSpeed", ret) ;//判断指令是否执行成功

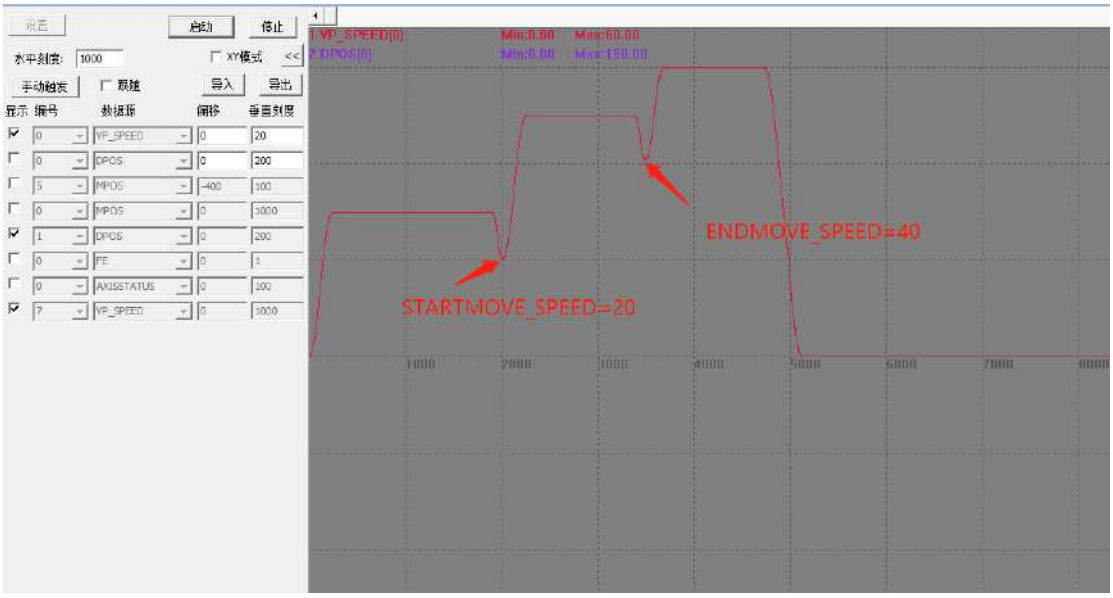
    dposlist=60;
    ZAux_Direct_MoveSp(handle,1,&axislist,&dposlist);//轴 0 运动 60
```

```
    getchar();
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

不使用 SP 速度时：



使用 SP 速度后：



6.5.3-6 多轴直线插补

```
// test1.cpp : 定义控制台应用程序的入口点。
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";                //仿真器 IP 地址
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("控制器连接成功! \n");

    ret =ZAux_Direct_SetAtype(handle,0,1);//设置轴 0 轴类型为 1
    commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetAtype(handle,1,1);//设置轴 1 轴类型为 1
    commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//判断指令是否执行成功
```

```
    ret =ZAux_Direct_SetAtype(handle,2,1);//设置轴 2 轴类型为 1
    commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//判断指令是否执行成功

    ret = ZAux_Direct_Single_Cancel(handle, 0, 0); //轴 0 运动停止
    commandCheckHandler("ZAux_Direct_Single_Cancel", ret);
    ret = ZAux_Direct_Single_Cancel(handle, 1, 0); //轴 1 运动停止
    commandCheckHandler("ZAux_Direct_Single_Cancel", ret);
    ret = ZAux_Direct_Single_Cancel(handle, 2, 0); //轴 2 运动停止
    commandCheckHandler("ZAux_Direct_Single_Cancel", ret);

    ret = ZAux_Direct_SetDpos(handle, 0, 0);         //设置轴 0 的 dpos
    commandCheckHandler("ZAux_Direct_SetDpos", ret);
    ret = ZAux_Direct_SetDpos(handle, 1, 0);         //设置轴 1 的 dpos
    commandCheckHandler("ZAux_Direct_SetDpos", ret);
    ret = ZAux_Direct_SetDpos(handle, 2, 0);         //设置轴 2 的 dpos
    commandCheckHandler("ZAux_Direct_SetDpos", ret);

    ret =ZAux_Direct_SetMerge(handle, 0,1);//设置连续插补
    commandCheckHandler("ZAux_Direct_SetMerge", ret) ;//判断指令是否执行成功

    int axislist[3] = { 1,0,2 };                      //运动轴列表，其中轴 1 为主轴
    float poslist[3] = { 100,200,300 };               //运动列表，轴 1-100，轴 0-200，
轴 2-300
    ZAux_Trigger( handle);//触发示波器

    ret = ZAux_Direct_SetSpeed(handle, axislist[0], 100);    //设置插补速度 100，设置
在主轴上
    commandCheckHandler("ZAux_Direct_SetSpeed", ret);

    ret = ZAux_Direct_SetAccel(handle, axislist[0], 500);    //设置插补加速度 500，设
置在主轴上
    commandCheckHandler("ZAux_Direct_SetAccel", ret);

    ret = ZAux_Direct_SetDecel(handle, axislist[0], 500);    //设置插补减速度 500，设
置在主轴上
    commandCheckHandler("ZAux_Direct_SetDecel", ret);
```

```
    ret = ZAux_Direct_MoveAbs(handle, 3, axislist, poslist); //调用运动，轴 0,1,2 走到对应的绝对位置
    commandCheckHandler("ZAux_Direct_MoveAbs", ret);

    int state[3];

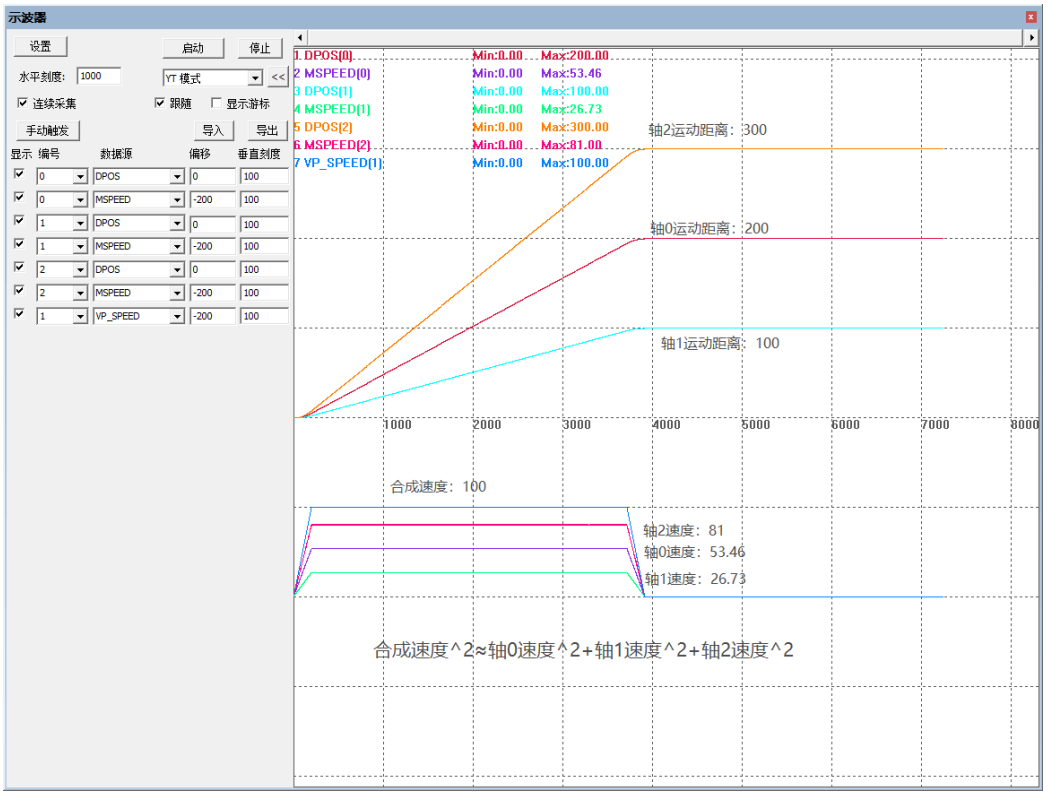
    do
    {
        ZAux_Direct_GetIfIdle(handle, axislist[0], state);
        ZAux_Direct_GetIfIdle(handle, axislist[1], state + 1);
        ZAux_Direct_GetIfIdle(handle, axislist[2], state + 2);

        if (!(state[0] || state[1] || state[2]))
        {
            Sleep(100);
        }
        else
        {
            break;
        }
    } while (true);//等待运动停止

    printf("Idle\n");

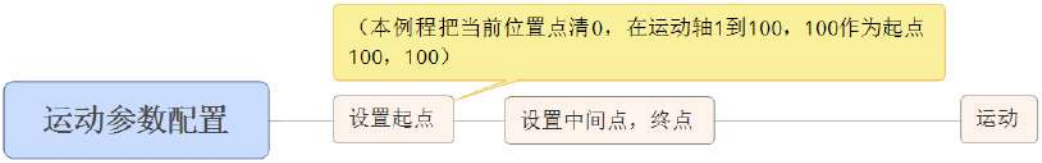
    Sleep(2000);
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

图为轴 0,1,2 的运动速度变化，线段表示意义如图所示。



6.5.3-7 平面三点画弧

本例程使用 2 个轴做平面圆弧运动，起点（100，100），终点(300,100)，中间点(200,200)，三点画弧。




```
// test1.cpp : 定义控制台应用程序的入口点。
//
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
```

```
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";                //仿真器 IP 地址
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("控制器连接成功! \n");

    float UnitsValue = 100;
    float DposValue = 0;
    float SpeedValue = 100;
    float AccelValue = 10000;
    float DecelValue = 10000;
    int AxisList[2] = { 0,1 };
    float DiatanceValue = 100;

    ret =ZAux_Direct_SetAtype(handle,0,1);//设置轴 0 轴类型为 1
    commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetAtype(handle,1,1);//设置轴 1 轴类型为 1
    commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//判断指令是否执行成功

    ret = ZAux_Direct_SetUnits(handle, AxisList[0], UnitsValue);//设置轴 0 脉冲当量
    commandCheckHandler("ZAux_Direct_SetUnits", ret);

    ret = ZAux_Direct_SetUnits(handle, AxisList[1], UnitsValue);//设置轴 1 脉冲当量
    commandCheckHandler("ZAux_Direct_SetUnits", ret);
```

```
ret = ZAux_Direct_SetSpeed(handle, AxisList[0], SpeedValue);//设置轴 0 速度
commandCheckHandler("ZAux_Direct_SetSpeed", ret);

ret = ZAux_Direct_SetSpeed(handle, AxisList[1], SpeedValue);//设置轴 1 速度
commandCheckHandler("ZAux_Direct_SetSpeed", ret);

ret = ZAux_Direct_SetAccel(handle, AxisList[0], AccelValue);//设置轴 0 加速度
commandCheckHandler("ZAux_Direct_SetAccel", ret);

ret = ZAux_Direct_SetAccel(handle, AxisList[1], AccelValue);//设置轴 1 加速度
commandCheckHandler("ZAux_Direct_SetAccel", ret);

ret = ZAux_Direct_SetDecel(handle, AxisList[0], DecelValue);//设置轴 0 减速度
commandCheckHandler("ZAux_Direct_SetDecel", ret);

ret = ZAux_Direct_SetDecel(handle, AxisList[1], DecelValue);//设置轴 1 减速度
commandCheckHandler("ZAux_Direct_SetDecel", ret);

ret = ZAux_Direct_SetDpos(handle, AxisList[0], 0);//设置轴 0 的 dpos 为 0
commandCheckHandler("ZAux_Direct_SetDpos", ret);

ret = ZAux_Direct_SetDpos(handle, AxisList[1], 0);//设置轴 1 的 dpos 为 0
commandCheckHandler("ZAux_Direct_SetDpos", ret);

ret = ZAux_Direct_Single_Cancel(handle, 0, 0); //轴 0 运动停止
commandCheckHandler("ZAux_Direct_Single_Cancel", ret);
ret = ZAux_Direct_Single_Cancel(handle, 1, 0); //轴 1 运动停止
commandCheckHandler("ZAux_Direct_Single_Cancel", ret);
ret = ZAux_Direct_Single_Cancel(handle, 2, 0); //轴 2 运动停止
commandCheckHandler("ZAux_Direct_Single_Cancel", ret);

int run_state = 0; //轴运动状态

ZAux_Trigger( handle);//触发示波器

//运动到起点位置（100，100）
ret = ZAux_Direct_Single_Move(handle, AxisList[0], DiatanceValue);
```

```
    commandCheckHandler("ZAux_Direct_Single_Move", ret);

    ret = ZAux_Direct_Single_Move(handle, AxisList[1], DiatanceValue);
    commandCheckHandler("ZAux_Direct_Single_Move", ret);
    //运动到起点位置（100，100）
    int state[2];
    do
    {
        ret = ZAux_Direct_GetIfIdle(handle, AxisList[0], state);
        commandCheckHandler("ZAux_Direct_GetIfIdle", ret);

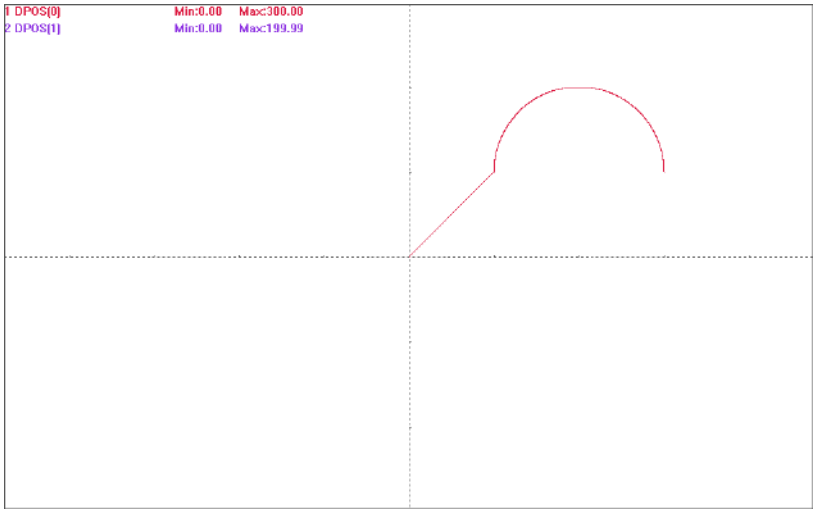
        ret = ZAux_Direct_GetIfIdle(handle, AxisList[1], state + 1);
        commandCheckHandler("ZAux_Direct_GetIfIdle", ret);

        if (!(state[0] || state[1] ))
        {
            Sleep(100);
        }
        else
        {
            break;
        }
    } while (true);//等待运动停止

    //三点画弧运动，起点（100，100），中间点（200，200），终点（300，100）
    ret = ZAux_Direct_MoveCirc2(handle, 2, AxisList, 200-100, 200-100, 300-100, 100-
100);
    commandCheckHandler("ZAux_Direct_MoveCirc2", ret);

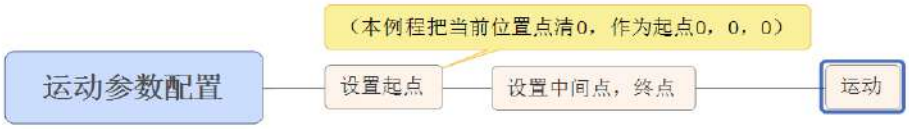
    Sleep(2000);
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

XY 的平面效果图



6.5.3-8 空间 3 点圆弧

本例程使用 3 个轴做空间圆弧运动，当前位置点，作为起点（0，0，0），终点(120,160,400)，中间点(240,320,300)，模式 0，三点画弧。




```
// test1.cpp : 定义控制台应用程序的入口点。
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
```

```
char *ip_addr = (char *)"127.0.0.1";                //仿真器 IP 地址
ZMC_HANDLE handle = NULL;                           //连接句柄
int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
if (ERR_SUCCESS != ret)
{
    printf("控制器连接失败!\n");
    handle = NULL;
    Sleep(2000);
    return -1;
}
printf("控制器连接成功! \n");

ret =ZAux_Direct_SetAtype(handle,0,1);//设置轴 0 轴类型为 1
commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//判断指令是否执行成功

ret =ZAux_Direct_SetAtype(handle,1,1);//设置轴 1 轴类型为 1
commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//判断指令是否执行成功

ret =ZAux_Direct_SetAtype(handle,2,1);//设置轴 2 轴类型为 1
commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//判断指令是否执行成功

ret = ZAux_Direct_SetSpeed(handle, 0, 200); //设置主轴 0 速度为 200units/s
commandCheckHandler("ZAux_Direct_SetSpeed", ret);
ret = ZAux_Direct_SetAccel(handle, 0, 2000); //设置主轴 0 加速度为 2000units/s/s
commandCheckHandler("ZAux_Direct_SetAccel", ret);
ret = ZAux_Direct_SetDecel(handle, 0, 2000); //设置主轴 0 减速度为 2000units/s/s
commandCheckHandler("ZAux_Direct_SetDecel", ret);

ret = ZAux_Direct_Single_Cancel(handle, 0, 0); //轴 0 运动停止
commandCheckHandler("ZAux_Direct_Single_Cancel", ret);
ret = ZAux_Direct_Single_Cancel(handle, 1, 0); //轴 1 运动停止
commandCheckHandler("ZAux_Direct_Single_Cancel", ret);
ret = ZAux_Direct_Single_Cancel(handle, 2, 0); //轴 2 运动停止
commandCheckHandler("ZAux_Direct_Single_Cancel", ret);

int run_state = 0; //轴运动状态
```

```
    ret = ZAux_Direct_SetDpos(handle, 0, 0);     //设置轴 0 的 dpos 为 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret);
    ret = ZAux_Direct_SetDpos(handle, 1, 0);     //设置轴 1 的 dpos 为 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret);
    ret = ZAux_Direct_SetDpos(handle, 2, 0);     //设置轴 2 的 dpos 为 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret);

    ZAux_Trigger( handle);//触发示波器

    //参与运动的轴列表
    int axisList[3] = { 0,1,2};

    //起点（0,0,0）终点(120,160,400)，中间点(240,320,300)，模式 0，三点画弧
    ret = ZAux_Direct_MSpherical(handle, 3, axisList, 120, 160, 400, 240, 320, 300,
0, 0, 0);
    commandCheckHandler("ZAux_Direct_MSpherical", ret);

    int state[3];

    do
    {
        ret = ZAux_Direct_GetIfIdle(handle, axisList[0], state);
        commandCheckHandler("ZAux_Direct_GetIfIdle", ret);

        ret = ZAux_Direct_GetIfIdle(handle, axisList[1], state + 1);
        commandCheckHandler("ZAux_Direct_GetIfIdle", ret);

        ret = ZAux_Direct_GetIfIdle(handle, axisList[2], state + 2);
        commandCheckHandler("ZAux_Direct_GetIfIdle", ret);

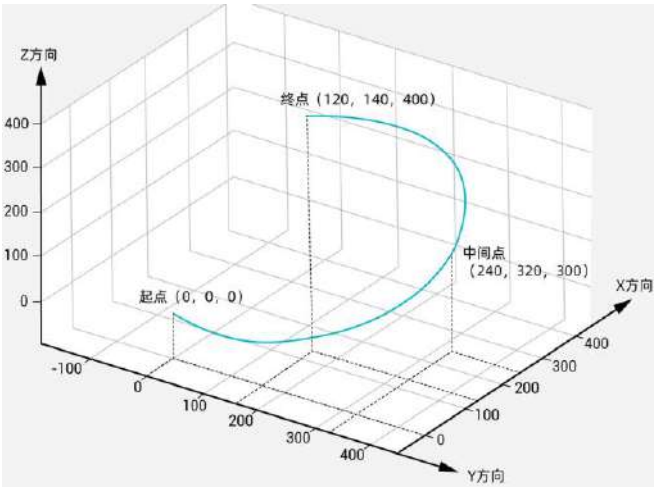
        if (!(state[0] || state[1] || state[2]))
        {
            Sleep(100);
        }
        else
        {
            break;
```

```
        }
    } while (true);//等待运动停止

    printf("Idle\n");

    Sleep(2000);
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

轨迹效果如下图所示：



6.5.3-9 螺旋曲线

本例程使用 3 个轴做空间圆弧运动，当前位置点，原点作为起点（0，0），终点(200，-200)，中间点(200,0），Z 轴参与运动，运动 100，作顺时针圆心螺旋运动。




```
// test1.cpp : 定义控制台应用程序的入口点。
//
#include "stdafx.h"
#include <windows.h>
```

```
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";                //仿真器 IP 地址
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("控制器连接成功! \n");

    ret =ZAux_Direct_SetAtype(handle,0,1);//设置轴 0 轴类型为 1
    commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetAtype(handle,1,1);//设置轴 1 轴类型为 1
    commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetAtype(handle,2,1);//设置轴 2 轴类型为 1
    commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//判断指令是否执行成功

    ret = ZAux_Direct_Single_Cancel(handle, 0, 0); //轴 0 运动停止
    commandCheckHandler("ZAux_Direct_Single_Cancel", ret);
    ret = ZAux_Direct_Single_Cancel(handle, 1, 0); //轴 1 运动停止
```

```
    commandCheckHandler("ZAux_Direct_Single_Cancel", ret);
    ret = ZAux_Direct_Single_Cancel(handle, 2, 0); //轴 2 运动停止
    commandCheckHandler("ZAux_Direct_Single_Cancel", ret);

    int run_state = 0; //轴运动状态

    ret = ZAux_Direct_SetDpos(handle, 0, 0);    //设置轴 0 的 dpos 为 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret);
    ret = ZAux_Direct_SetDpos(handle, 1, 0);    //设置轴 1 的 dpos 为 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret);
    ret = ZAux_Direct_SetDpos(handle, 2, 0);    //设置轴 1 的 dpos 为 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret);

    ret = ZAux_Direct_SetSpeed(handle, 0, 100); //设置主轴 轴 0 速度为 100units/s
    commandCheckHandler("ZAux_Direct_SetSpeed", ret);
    ret = ZAux_Direct_SetAccel(handle, 0, 10000); //设置主轴 轴 0 加速度为
10000units/s/s
    commandCheckHandler("ZAux_Direct_SetAccel", ret);
    ret = ZAux_Direct_SetDecel(handle, 0, 10000); //设置主轴 轴 0 减速度为
10000units/s/s
    commandCheckHandler("ZAux_Direct_SetDecel", ret);

    //参与运动的轴列表
    int axisList[3] = { 0,1,2 };
    ZAux_Trigger( handle);//触发示波器

    //原点作为起点，中心(200,0)，终点(200,-200)，顺时针，Z 轴参与速度计算，运动 100，
作圆心螺旋运动
    ret = ZAux_Direct_MHelical(handle, 3, axisList, 200,-200, 200, 0, 1, 100,0);
    commandCheckHandler("ZAux_Direct_MoveCirc", ret);

    int state[3];
    do
    {
        ret = ZAux_Direct_GetIfIdle(handle, axisList[0], state);
        commandCheckHandler("ZAux_Direct_GetIfIdle", ret);
```

```
        ret = ZAux_Direct_GetIfIdle(handle, axisList[1], state + 1);
        commandCheckHandler("ZAux_Direct_GetIfIdle", ret);

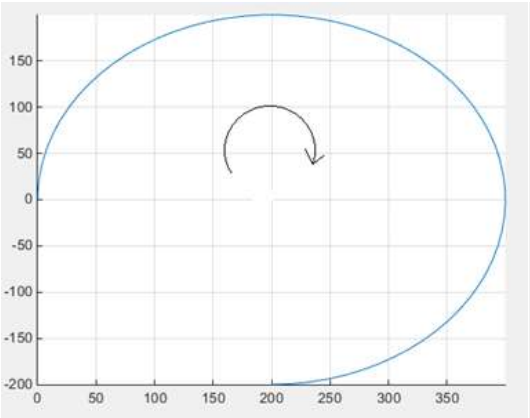
        ret = ZAux_Direct_GetIfIdle(handle, axisList[2], state + 2);
        commandCheckHandler("ZAux_Direct_GetIfIdle", ret);

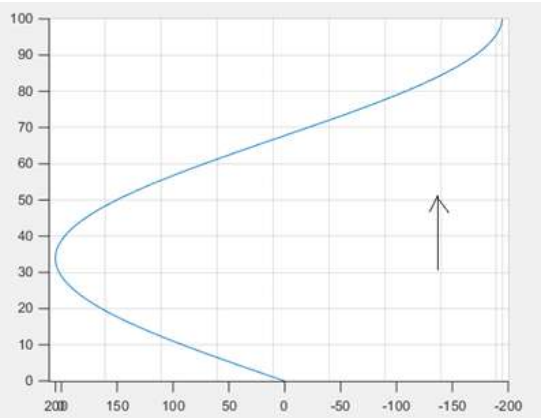
        if (!(state[0] || state[1] || state[2]))
        {
            Sleep(100);
        }
        else
        {
            break;
        }
    } while (true);//等待运动停止

    printf("Idle\n");

    Sleep(2000);
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

插补轨迹





6.6 轴暂停/恢复/停止相关

6.6.1 指令列表

指令	说明
ZAux_Direct_MovePause	运动暂停，插补运动暂停主轴
ZAux_Direct_MoveResume	取消运动暂停
ZAux_Direct_Rapidstop	所有轴立即停止
ZAux_Direct_CancelAxisList	多个轴运动停止
ZAux_Direct_Single_Cancel	单轴运动停止

6.6.2 重点说明

在单轴或多轴插补运动时，运动控制卡支持运动暂停，轴停止等功能。

注意：使用轴停止功能时，运动会以快减减速度（ZAux_Direct_SetFastDec）停止运动，若快减减速度未设置，则快减减速度会默认等于设置的减速度（ZAux_Direct_SetDecel），所以使用轴停止功能时必须设置减速度或者快减减速度。ZAux_Direct_Single_Cancel, 用模式 3 时，不能用于被叠加轴的停止, 否则可能导致叠加轴位置不一致。

6.6.3 例程

6.6.3-1 轴暂停与恢复

本例程使用轴 0 做单轴运动，正方向运动 2000 个 uintis，运动开始之后 1 秒后暂停，4 秒之后恢复运动。




```
// test1.cpp : 定义控制台应用程序的入口点。
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
```

```
void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s return code is %d\n", command, ret);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";                //控制器 IP 地址
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("connect controller failed!\n");
        handle = NULL;
        getchar();
        return -1;
    }
    printf("控制器连接成功! \n");

    ZAux_Direct_SetAtype(handle, 0, 1);   //设置轴 0 轴类型为 1
    ZAux_Direct_SetUnits(handle, 0, 100); //设置轴 0 脉冲当量为 100
    ZAux_Direct_SetSpeed(handle, 0, 200); //设置轴 0 速度为 200units/s
    ZAux_Direct_SetAccel(handle, 0, 2000); //设置轴 0 加速度为 2000units/s/s
    ZAux_Direct_SetDecel(handle, 0, 2000); //设置轴 0 减速度为 2000units/s/s
    ZAux_Direct_SetSramp(handle, 0, 200); //设置轴 0 S 曲线为 200ms

    ret = ZAux_Direct_SetDpos( handle,  0,  0);//轴指令位置清 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret);//判断指令是否执行成功
    ret = ZAux_Direct_SetMpos( handle,  0,  0);//编码器反馈位置清 0
    commandCheckHandler("ZAux_Direct_SetMpos", ret);//判断指令是否执行成功
    int GetVal;
    ret =ZAux_Direct_Single_Move( handle,  0,  2000);
    commandCheckHandler("ZAux_Direct_SetDpos", ret);//判断指令是否执行成功
```

```
    Sleep(1000);//等待 1S
    ZAux_Direct_MovePause(handle, 0,0);//暂停轴 0 当前运动

    Sleep(2000);//等待 2S

    ZAux_Direct_GetAxisStatus(handle, 0,&GetVal);//获取轴停止原因
    printf("当前轴状态：%x h\n",GetVal);

    Sleep(2000);//等待 2S
    ZAux_Direct_MoveResume(handle, 0);//恢复轴 0 运动

    float IDLE;
    while (1)//等待轴 0 运动完成
    {
        Sleep(100);
        ZAux_Direct_GetParam(handle,"IDLE",0,&IDLE);
        if (IDLE<0)break;
    }

    Sleep(2000);
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

第七章 进阶运动控制

7.1 坐标循环

7.1.1 指令列表

指令	说明
ZAux_Direct_SetRepDist	设置循环范围
ZAux_Direct_GetRepDist	读取设置循环范围
ZAux_Direct_SetRepOption	设置坐标循环模式
ZAux_Direct_GetRepOption	读取坐标循环模式

7.1.2 重点说明

本章节指令，可用于限制凸轮主轴的坐标循环范围，来实现多条凸轮曲线连续。

使用绝对运动模式时，如果目标位置处于坐标循环范围内，可以正确运动；如果处于坐标循环范围外，运动不正确。

相对运动不受影响。

ZAux_Direct_SetRepOption（坐标循环模式）：按位来表示不同的意义

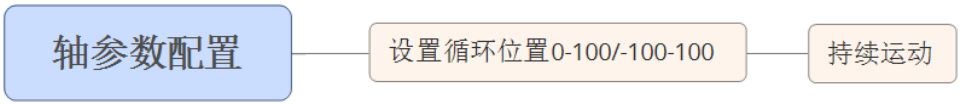
位	值	描述
0	1	0-循环范围：－坐标循环位置（**ZAux_Direct_SetRepDist**）到 +坐标循环位置（**ZAux_Direct_SetRepDist**） 1-循环范围：0 到 +坐标循环位置（**ZAux_Direct_SetRepDist**）
1	2	该位置 1 跟随凸轮表运动(**ZAux_Direct_Cambox**)和自动凸轮（**ZAux_Direct_Movelink**）的重复运动，禁止生效后还原为 0
2	4	预留
4	16	1-不使用 **ZAux_Direct_SetRepDist**，0-使用 **ZAux_Direct_SetRepDist**

7.1.3 例程

7.1.3-1 坐标循环

本例程使用轴 0 从原点位置开始，作-100 至 100 之间循环运动以及 0-100 直接的循环运

动。




```
// test1.cpp : 定义控制台应用程序的入口点。
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
    }

}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";                //控制器 IP 地址
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        getchar();
        return -1;
    }
    printf("控制器连接成功! \n");

    ret =ZAux_Direct_SetAtype(handle,0,1);//设置轴类型为 1
```

```
    commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetAccel(handle,0,500);//设置轴加速度
    commandCheckHandler("ZAux_Direct_SetAccel", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetDecel(handle,0,500);//设置轴减速度
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetDpos(handle,0,0);//设置轴 DPOS 清 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetSpeed(handle,0,250);//设置轴速度
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetRepDist(handle,0,100);//设置循环坐标为 100
    commandCheckHandler("ZAux_Direct_SetRepDist", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetRepOption(handle,0,1);//循环模式设置为 1
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//判断指令是否执行成功

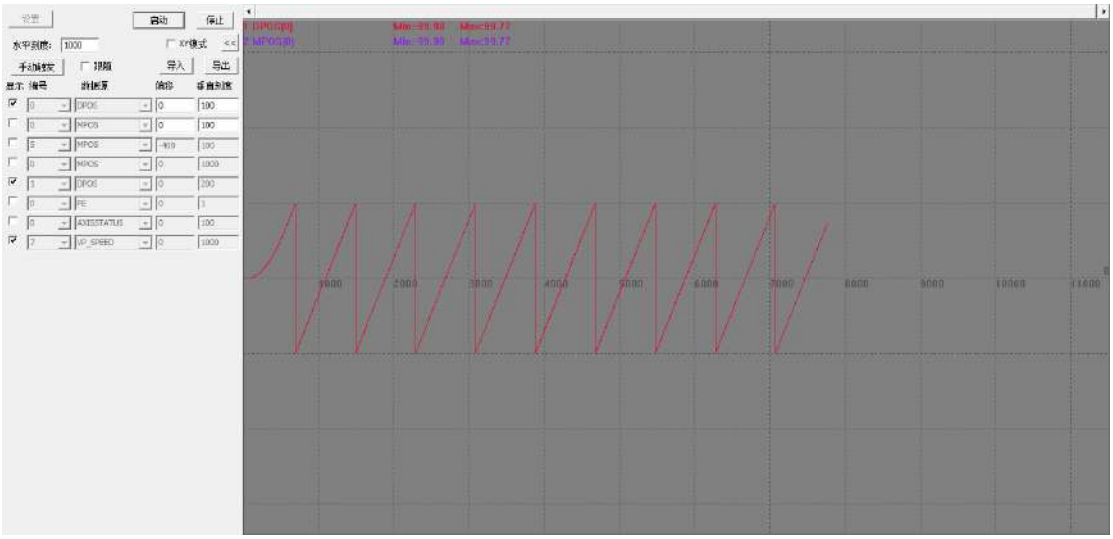
    float IDLE;
    int mode;
    int AxisStatus;
    ZAux_Trigger( handle);//触发示波器

    ZAux_Direct_Single_Vmove(handle,0,1);//轴 0 持续运动

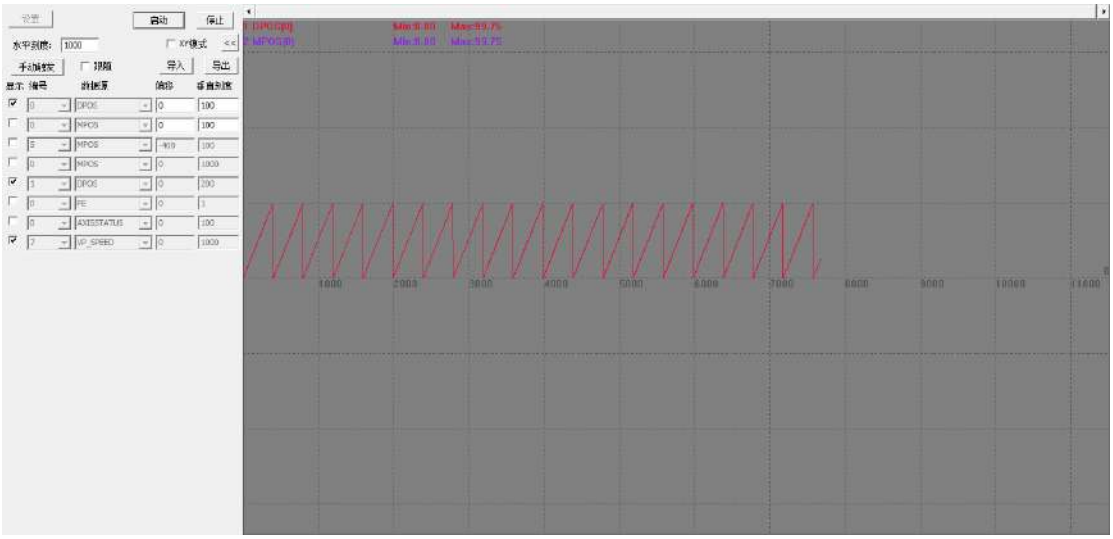
    while (1)//等待轴 0 运动完成
    {
        Sleep(100);
        ZAux_Direct_GetParam(handle,"IDLE",0,&IDLE);
        if (IDLE<0)break;
    }

    getchar();
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

ZAux_Direct_SetRepOption（坐标循环模式）=0，-100 至+100 之间循环



ZAux_Direct_SetRepOption（坐标循环模式）=1，0 至+100 之间循环



7.2 螺距补偿

7.2.1 指令列表

指令	说明
ZAux_Direct_Pitchset	设置轴的螺距补偿，扩展轴无效
ZAux_Direct_Pitchset2	设置轴的螺距双向补偿，扩展轴无效。
ZAux_Direct_GetPitchStatus	设置轴的螺距补偿状态读取，扩展轴无效。

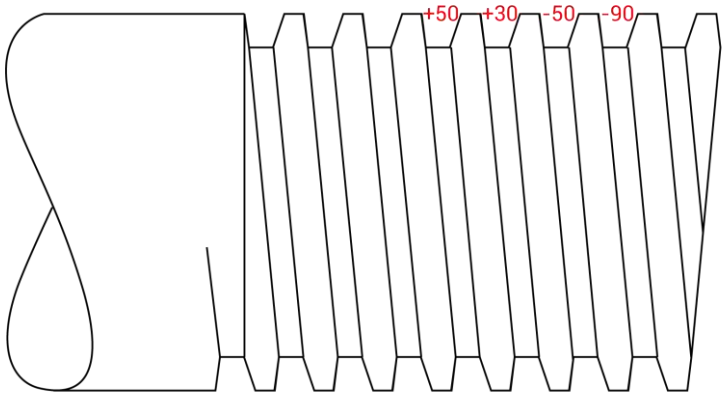
7.2.2 重点说明

开环、半闭环系统，其定位精度很大程度上受滚珠丝杠精度的影响，尽管滚珠丝杠的精度很高，但总是存在制造误差。要得到超过滚珠丝杠精度的运动精度，在系统中则必须采用螺距误差补偿功能，利用系统对误差进行补偿和修正，来保证加工精度。

本章节指令主要为螺距补偿的函数，螺距补偿主要应用于，当设备机构有制造误差时，可以使用螺距补偿，保证精度。

7.2.3 例程

7.2.3-1 螺距补偿



每一圈的螺距存在细微的误差，并不是严格均匀的，故精度高的场合需要螺距补偿来调整。本例程驱动轴 0 运动一段距离 700 的往返运动，需要在如下列表点补偿相应的脉冲数。

开始补偿点位置（MPOS）	补偿值（距离 100 时的脉冲个数）
100	0
200	50 个脉冲
300	30 个脉冲
400	-50 个脉冲
500	-90 个脉冲
600	0

```
// test1.cpp : 定义控制台应用程序的入口点。
//


#include "stdafx.h"
```

```
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"192.168.0.11";              //控制器 IP 地址
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        getchar();
        return -1;
    }
    printf("控制器连接成功! \n");

    ret =ZAux_Direct_SetAtype(handle,0,1);//设置轴类型为 1
    commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetAtype(handle,5,6);//设置轴类型为 6
    commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetAccel(handle,0,500);//设置轴加速度
    commandCheckHandler("ZAux_Direct_SetAccel", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetDecel(handle,0,500);//设置轴减速度
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetDpos(handle,0,0);//设置轴 DPOS 清 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//判断指令是否执行成功
```

```
ret =ZAux_Direct_SetMpos(handle,0,0);//设置轴 MPOS 清 0
commandCheckHandler("ZAux_Direct_SetMpos", ret) ;//判断指令是否执行成功

ret =ZAux_Direct_SetDpos(handle,5,0);//设置轴 DPOS 清 0
commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//判断指令是否执行成功
ret =ZAux_Direct_SetMpos(handle,5,0);//设置轴 MPOS 清 0
commandCheckHandler("ZAux_Direct_SetMpos", ret) ;//判断指令是否执行成功

ret =ZAux_Direct_SetSpeed(handle,0,250);//设置轴速度
commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//判断指令是否执行成功

float numlist[4]={50,30,-50,-90};

//开始补偿点位置（MPOS）   补偿值（距离 100 时的脉冲个数）
//  100                          0
//  200                          50 个脉冲
//  300                          30 个脉冲
//  400                          -50 个脉冲
//  500                          -90 个脉冲
//  600                          0
ret =ZAux_Direct_Pitchset(handle,0,1,100,4,100,0,numlist);//设置螺距补偿
commandCheckHandler("ZAux_Direct_Pitchset", ret) ;//判断指令是否执行成功

float IDLE;
int mode;
int AxisStatus;
ZAux_Trigger( handle);//触发示波器

ZAux_Direct_Single_Move(handle,0,700);//轴 0 运动 700

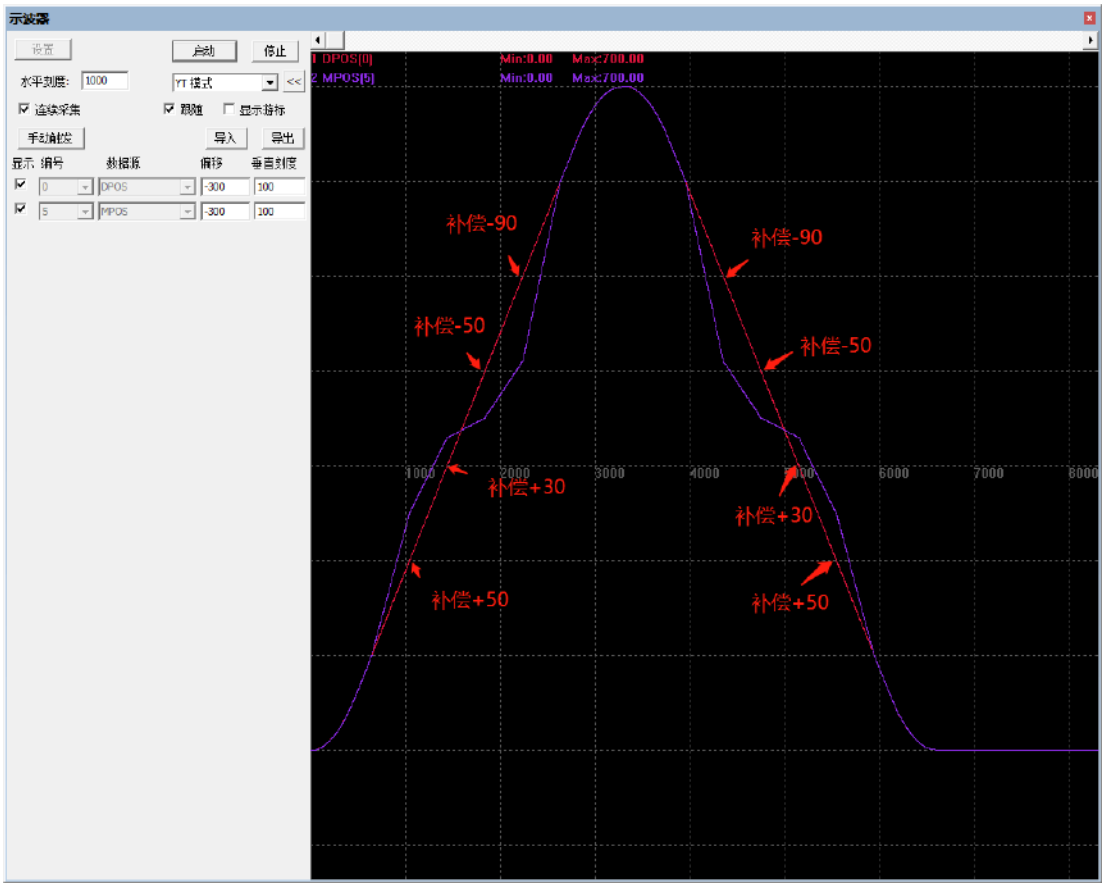
while (1)//等待轴 0 运动完成
{
    Sleep(100);
    ZAux_Direct_GetParam(handle,"IDLE",0,&IDLE);
    if (IDLE<0)break;
}
ZAux_Direct_Single_Move(handle,0,-700);//轴 0 运动-700
```

```
    while (1)//等待轴 0 运动完成
    {
        Sleep(100);
        ZAux_Direct_GetParam(handle,"IDLE",0,&IDLE);
        if (IDLE<0)break;
    }

    ret =ZAux_Direct_Pitchset(handle,0,0,100,4,100,0,numlist);//关闭螺距补偿
    commandCheckHandler("ZAux_Direct_Pitchset", ret) ;//判断指令是否执行成功

    getchar();
    ret = ZAux_Close(handle);     //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

波形图：（该波形图，为了更好的显示补偿效果，把轴 0 的脉冲输出，接入到了轴 5 的编码器输入上，方便观察规划位置以及实际位置的区别，以看出补偿效果）



7.3 反向间隙补偿

7.3.1 指令列表

指令	说明
ZAux_Direct_Backlash	设置轴的反向间隙补偿，扩展轴无效

7.3.2 重点说明

因为丝杠和螺母之间肯定存在一定的间隙，所以在正转后变换成反转的时候，在一定的角度内，尽管丝杠转动，但是螺母还要等间隙消除以后才能带动工作台运动，这个间隙就是反向间隙，但是要反映在丝杠的旋转角度上。

在设备上，给传动链的各环节中，比如齿轮传动、滚珠丝杠螺母副等都存在着反向间隙。反向间隙是影响机械加工精度的因素之一，当数控机床工作台在其运动方向上换向时，由于反向间隙的存在会导致伺服电动机空转而工作台无实际移动，称为失动。若反向间隙数值较小，对加工精度影响不大则不需要采取任何措施；若数值较大，则系统的稳定性明显下降，

加工精度明显降低，尤其是曲线加工，会影响到尺寸公差和曲线的一致性，此时必须进行反向间隙的消除或是补偿，以提高加工精度。

7.3.3 例程

7.3.3-1 反向间隙补偿

例：假设某机构，存在 10 脉冲当量的反向间隙，如下例子，即演示了该机构正向运动走 200 后在反向走 100，使用反向间隙补偿，把该间隙给自动补偿上去。

```
// test1.cpp : 定义控制台应用程序的入口点。
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"192.168.0.11";              //控制器 IP 地址
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
```

```
}
printf("控制器连接成功！\n");

ret =ZAux_Direct_SetAtype(handle,0,4);//设置轴类型为 4
commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//判断指令是否执行成功
ret =ZAux_Direct_SetUnits(handle,0,1);//设置脉冲当量为 1
commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//判断指令是否执行成功

ret =ZAux_Direct_SetAccel(handle,0,500);//设置轴加速度
commandCheckHandler("ZAux_Direct_SetAccel", ret) ;//判断指令是否执行成功
ret =ZAux_Direct_SetDecel(handle,0,500);//设置轴减速度
commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//判断指令是否执行成功
ret =ZAux_Direct_SetDpos(handle,0,0);//设置轴 DPOS 清 0
commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//判断指令是否执行成功
ret =ZAux_Direct_SetMpos(handle,0,0);//设置轴 MPOS 清 0
commandCheckHandler("ZAux_Direct_SetMpos", ret) ;//判断指令是否执行成功

ret =ZAux_Direct_SetSpeed(handle,0,1000);//设置轴速度
commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//判断指令是否执行成功

ret =ZAux_Direct_SetMerge(handle,0,1);//设置轴速度
commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//判断指令是否执行成功

ret =ZAux_Direct_Backlash(handle,0,1,10,100,100);//开启轴 0 反向间隙补偿，补偿 10
commandCheckHandler("ZAux_Direct_Backlash", ret) ;//判断指令是否执行成功

float IDLE;
int mode;
int AxisStatus;
ZAux_Trigger( handle);//触发示波器

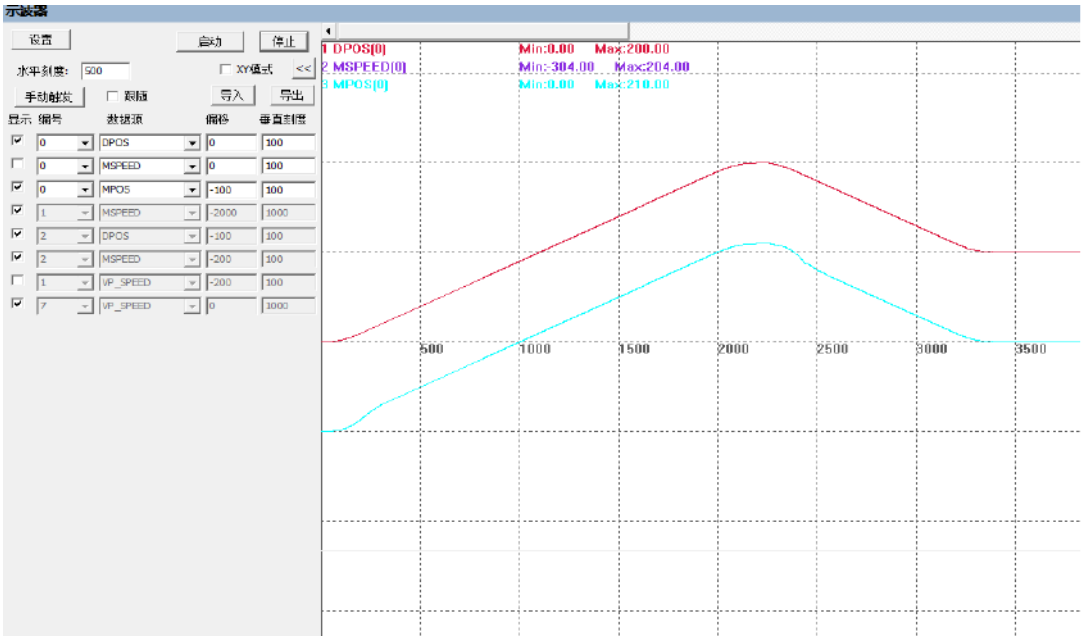
ZAux_Direct_Single_Move(handle,0,200);//轴 0 运动 200
ZAux_Direct_Single_Move(handle,0,-100);//轴 0 运动-100

while (1)//等待轴 0 运动完成
{
    Sleep(100);
```

```
        ZAux_Direct_GetParam(handle,"IDLE",0,&IDLE);
        if (IDLE<0)break;
    }
    ret =ZAux_Direct_Backlash(handle,0,0,0,100,100);//关闭轴 0 反向间隙补偿
    commandCheckHandler("ZAux_Direct_Backlash", ret) ;//判断指令是否执行成功

    Sleep(2000);
    ret = ZAux_Close(handle);     //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

运动波形图：(由波形图可以看出来，在运动换向时，蓝色实际位置曲线，比红色运动规划曲线多走了一点)



第八章 硬件接口访问与配置

8.1 数字输入输出

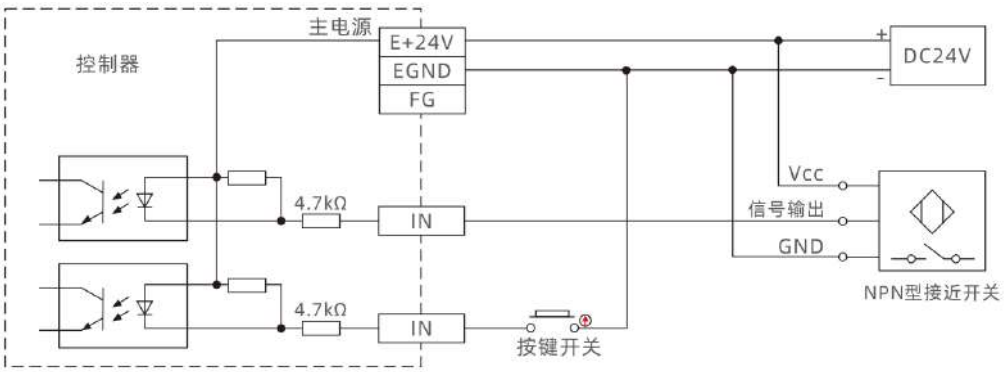
8.1.1 指令列表

指令	说明
ZAux_Direct_GetIn	读取单个输入信号
ZAux_Direct_SetOp	设置单个输出口信号
ZAux_Direct_GetOp	读取单个输出口状态
ZAux_GetModbusIn	快速读取多个输入(Modbus 方式)
ZAux_GetModbusOut	快速读取多个当前的输出状态(Modbus 方式)
ZAux_Direct_GetInMulti	读取多个输入信号
ZAux_Direct_SetOutMulti	IO 接口 设置多路输出
ZAux_Direct_GetOutMulti	IO 接口 获取多路输出状态
ZAux_Direct_SetInvertIn	设置输入口有效状态反转
ZAux_Direct_GetInvertIn	读取输入口反转状态的有无

8.1.2 重点说明

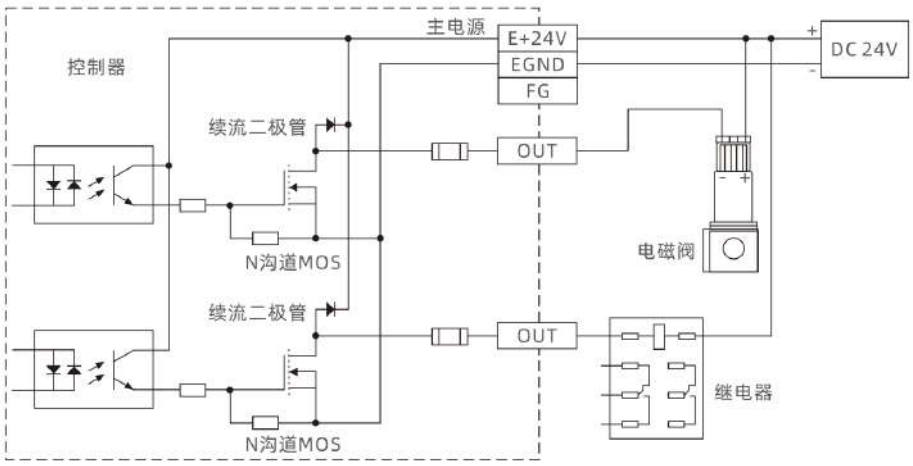
以 ZMC406-V2 控制器为例，控制器的输入输出口无需使用外部 IO 电源单独供电，连接扩展模块 IO 最多可扩展至 512 点。

控制器数字量输入口 IN 允许接入的信号类型为 NPN，24V 信号，内部电路图如下。IN 口电压 7V 以下为低电平。输入口分为高速输入口与普通输入口，一般高速 IN 口允许的输入信号频率为 100kHz，其他普通 IN 口允许输入的频率在 5kHz 左右。



控制器数字量输出口 OUT 信号类型为 NPN，0V 信号，输出电流 300mA，内部电路图如下。

高速输出口同样也有高低速之分，其中高速输出口输出最大频率为 400kHz，普通(低速)输出口输出频率为 8kHz 左右。



具体硬件参数参考对应控制器/控制卡硬件手册。

8.1.3 例程

8.1.3-1 IO 读取与设置

本例程获取仿真器 IN0 的输入状态，再获取 OP0 的状态，根据 OP0 的输入状态重新设置 OP0，后再读取 OP0 的状态。




```
// test1.cpp : 定义控制台应用程序的入口点。
//
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
```

```
        printf("%s fail!return code is %d\n", command, ret);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";                //控制器 IP 地址
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("控制器连接成功! \n");

    unsigned int in_value;
    unsigned int out_value;
    ret = ZAux_Direct_GetIn(handle, 0, &in_value);    //读取 IN0 状态
    commandCheckHandler("ZAux_Direct_GetIn", ret);
    printf("IN0 status: %d\n", in_value);
    ret = ZAux_Direct_GetOp(handle, 0, &out_value);   //读取 OP0 状态
    commandCheckHandler("ZAux_Direct_GetOp", 0);
    printf("OP0 status: %d\n", out_value);

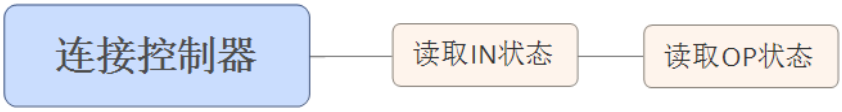
    if(out_value==0)
    {
        ret = ZAux_Direct_SetOp(handle, 0, 1);            //设置 OP0 状态
        commandCheckHandler("ZAux_Direct_SetOp", 0);
    }else
    {
        ret = ZAux_Direct_SetOp(handle, 0, 0);            //设置 OP0 状态
        commandCheckHandler("ZAux_Direct_SetOp", 0);
    }
    ret = ZAux_Direct_GetOp(handle, 0, &out_value);   //读取 OP0 状态
```

```
    commandCheckHandler("ZAux_Direct_GetOp", 0);
    printf("OP0 status: %d\n", out_value);

    Sleep(2000);
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

8.1.3-2 多个 IO 读取(GetModbusOut/GetModbusIn)

本例程获取仿真器 IN0 – IN24 的输入状态，再获取 OP0 - OP7 的状态。




```
// test1.cpp : 定义控制台应用程序的入口点。
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非0则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
    }

}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";                //控制器IP地址
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
```

```
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("控制器连接成功! \n");

    uint8 GetValue[8];                                  //读取设置的值

        ret = ZAux_GetModbusIn( handle, 0 , 24, GetValue);

        commandCheckHandler("ZAux_GetModbusIn", ret);

        for (int i = 0; i <= 24; i++)
        {
            if (i%8==0)
            {

                printf("\n");
            }
            //一次性取8位，每一位对应一个IO
            //(GetValue[(int)(i/8)] & (0x1 <<(i%8))：只保留对应位，其余全部置0
            // >> (i%8)):取出对应位状态
            printf("IN(%d) = %d  ", i, (GetValue[(int)(i/8)] & (0x1 <<(i%8))) >>
(i%8));

        }

    printf("\n**************************************************************************\n"
);
    ret = ZAux_GetModbusOut(handle, 0 , 7 , GetValue);
    commandCheckHandler("ZAux_GetModbusOut", ret);

    for (int i = 0; i <= 7; i++)
    {
        printf("OP(%d) = %d  ", i,(GetValue[(int)(i/8)] & (0x1 <<(i%8))) >> (i%8));
    }
    printf("\n");

    Sleep(20000);
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
```

```
    return 0;
}


 }
```

8.1.3-3 多个 IO 读取设置

本例程获取仿真器 IN 0-47 的输入状态，再获取 OP 0-OP7 的状态




```
// test1.cpp : 定义控制台应用程序的入口点。
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非0则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
    }

}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";                //控制器IP地址
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("控制器连接成功! \n");
```

```
    int32 GetValue[2];                                    //读取设置的值
    uint32 uValue;

    ret = ZAux_Direct_GetInMulti( handle, 0 , 48, GetValue);

    commandCheckHandler("ZAux_GetModbusIn", ret);

    for (int i = 0; i < 48; i++)
    {
        if (i%10==0)
        {

            printf("\n");
        }
        printf("IN(%d) = %d  ", i, (GetValue[(int)(i/32)] & (0x1 << (i%32))) >>
(i%32));
    }
    printf("\n**************************************************************************
\n");



    uValue=1;
    ret = ZAux_Direct_SetOutMulti(handle, 0 , 7 , &uValue);
    commandCheckHandler("ZAux_GetModbusOut", ret);

    ret = ZAux_Direct_GetOutMulti(handle, 0 , 7 , &uValue);
    commandCheckHandler("ZAux_GetModbusOut", ret);

    for (int i = 0; i <= 7; i++)
    {
        printf("OP(%d) = %d  ", i, (uValue & (0x1 << i)) >> i);
    }
    printf("\n");

    Sleep(20000);
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

8.2 模拟量输入输出

8.2.1 指令列表

指令	说明
ZAux_Direct_GetAD	读取单个模拟量输入信号
ZAux_Direct_SetDA	打开单个模拟量输出信号
ZAux_Direct_GetDA	读取模拟输出口状态

8.2.2 重点说明

控制器 AD/DA 采用了内部电源，如 ZMC416BE 模拟量的分辨率为 12 位，0-10V 电压型，对应刻度为 0-4095，控制器带 2 路 AD 和 2 路 DA，可采用模拟量扩展模块扩展更多接口，最多支持扩展 256 路 AD 和 128 路 DA，可选 12 位分辨率和 16 位分辨率的扩展模块。控制器默认位 12 位分辨率。

针脚号	名称	说明
1	DA0	0-10V 模拟输出口 0
2	DA1	0-10V 模拟输出口 1
3	AGND	模拟量口 GND
4	AD0	0-10V 模拟输入口 0
5	AD1	0-10V 模拟输入口 1

8.2.3 例程

8.2.3-1 DA/AD 的设置与读取

本例程获取仿真器 AD0 的输入状态，再获取 DA0 的状态，再设置 DA0 后在重新读取 DA0 的状态




```
// test1.cpp : 定义控制台应用程序的入口点。
//
```

```
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";                //控制器 IP 地址
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }

    printf("控制器连接成功! \n");

    float AD_value;
    float DA_value;
    ret = ZAux_Direct_GetAD(handle,0,&AD_value);//获取 AD0 状态
    commandCheckHandler("ZAux_Direct_GetAD", ret);
    printf("read AD_value: %f\n",AD_value);

    ret = ZAux_Direct_GetDA(handle,0,&DA_value);//获取 DA0 状态
    commandCheckHandler("ZAux_Direct_GetDA", ret);
```

```
    printf("read DA_value: %f\n",DA_value);
    Sleep(1000);
    ret = ZAux_Direct_SetDA(handle,0,3000);//设置 DA0 状态
    commandCheckHandler("ZAux_Direct_SetDA", ret);
    Sleep(1000);
    ret = ZAux_Direct_GetDA(handle,0,&DA_value);//获取 DA0 状态
    commandCheckHandler("ZAux_Direct_GetDA", ret);
    printf("read DA_value: %f\n",DA_value);

    Sleep(2000);
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

8.3 PWM 控制

8.3.1 指令列表

指令	说明
ZAux_Direct_SetPwmFreq	设置 pwm 频率
ZAux_Direct_GetPwmFreq	读取 pwm 频率
ZAux_Direct_SetPwmDuty	设置 pwm 占空比
ZAux_Direct_GetPwmDuty	读取 pwm 占空比
ZAux_Direct_MovePwm	运动中修改 pwm

8.3.2 重点说明

PWM 是英文“Pulse Width Modulation”的缩写，中文译为“脉冲宽度调制”，简称“脉宽调制”或“调制”，是利用微处理器的数字输出来对模拟电路进行控制的一种非常有效的技术。

PWM（脉冲宽度调制）就是通过调节占空比，来调节脉冲宽度。只有支持 PWM 功能的控制器才能使用

在调节激光切割的工艺参数时，经常会用到脉冲频率、占空比这两个，很多对激光切割工艺不熟悉的人员可能不太理解这两个参数的作用，接下来就给大家详细的介绍一下。

1 **周期信号**：从高电平到低电平再回到高电平为一个周期；

2 **脉冲频率**：在 1 秒钟内 PWM 有多少个周期；

3 **脉冲占空比**：一个周期内，高电平的时间与整个周期时间的比例；

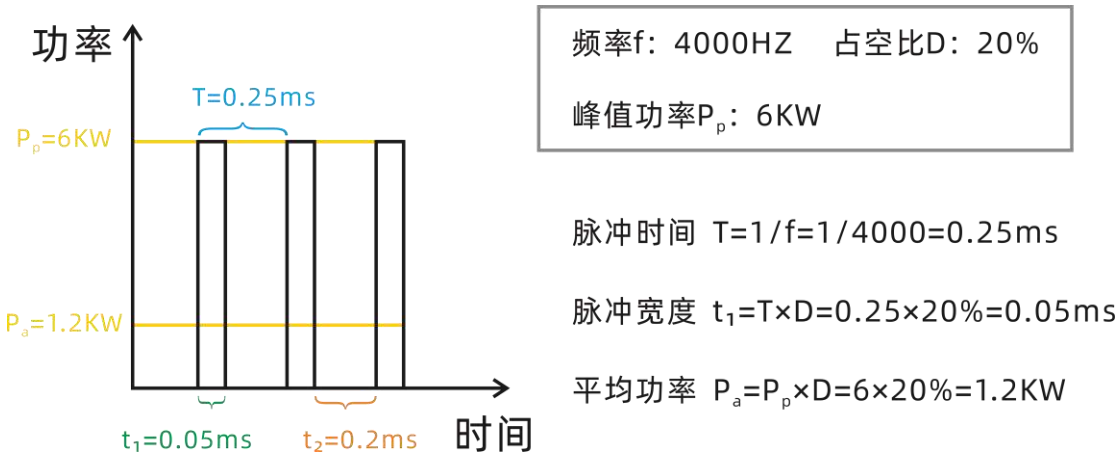
4 **脉冲时间**：一个周期所用的时间；

5 **脉冲宽度**：一个周期内，高电平所用的时间；

6 **激光峰值功率**：在一个脉冲宽度内输出的瞬时能量；

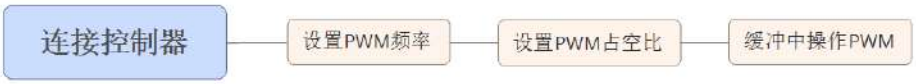
7 **平均功率**：激光在一个重复周期内输出的能量

接下来，我们用一个实例来说明如何计算 PWM 的各项参数。



8.3.3 例程

8.3.3-1 PWM 使用




```
// test1.cpp : 定义控制台应用程序的入口点。
//
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
```

```
    if (ret)//非 0 则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"192.168.0.11";                //控制器 IP 地址
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("控制器连接成功! \n");

    float freq_value;
    float duty_value;
    ret = ZAux_Direct_SetPwmFreq(handle, 0,20);          //设置 PWM 频率，必须在设置
占空比前设置频率，此处设置为 20，是为了能让 RTSys 示波器能抓取到波形
    commandCheckHandler("ZAux_Direct_SetPwmFreq", ret);
    ZAux_Trigger( handle);//触发示波器

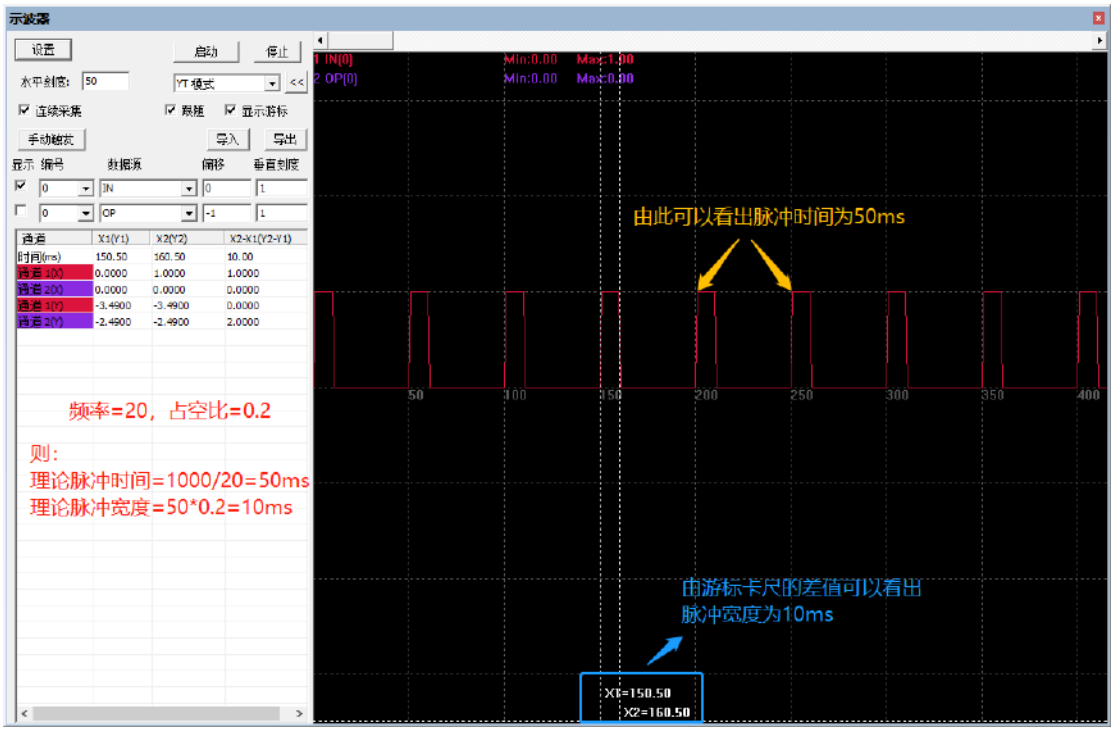
    ret = ZAux_Direct_SetPwmDuty(handle, 0, 0.2);          //设置 PWM 占空比，大于 0
即输出 PWM
    commandCheckHandler("ZAux_Direct_SetPwmDuty", ret);

    Sleep(1000);

    ret = ZAux_Direct_SetPwmDuty(handle, 0, 0);          //设置 PWM 占空比为 0，关闭
PWM 输出
    commandCheckHandler("ZAux_Direct_SetPwmDuty", ret);
```

```
    Sleep(2000);
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

示波器图形：



第九章 安全机制

9.1 轴软件限位函数

9.1.1 指令列表

指令	说明
ZAux_Direct_SetFsLimit	设置正向软限位
ZAux_Direct_GetFsLimit	读取正向软限位
ZAux_Direct_SetRsLimit	设置负向软限位位置
ZAux_Direct_GetRsLimit	读取负向软限位位置

9.1.2 重点说明

正向软限位、负向软限位都为控制器的限位功能，为软件限位，与硬件无关，限制的是命令位置的范围。当 **ZAux_Direct_SetFsLimit** (正向软限位)或者 **ZAux_Direct_SetRsLimit**（负向软限位）大于 **ZAux_Direct_SetRepDist**（坐标循环位置，详情查阅 7.1 章节）时，参数不起作用，正负向软限位被禁止。取消软限位时，建议不要去修改 **ZAux_Direct_SetRepDist**（坐标循环位置）的值，将 **ZAux_Direct_SetFsLimit**（正向软限位）或者 **ZAux_Direct_SetRsLimit**（负向软限位）设置一个较大值即可。**ZAux_Direct_SetFsLimit** (正向软限位)或者 **ZAux_Direct_SetRsLimit**（负向软限位）的值默认为 200000000/-200000000。

运动控制器能够通过安装限位开关或者设置软限位来限制各轴的运动范围。硬限位开关和软限位开关用于工艺对象轴的允许运动范围和工作范围。



与硬限位开关不同，软限位开关只通过软件程序设定来实现，而无需借助外部的开关元件。软限位开关将限制轴的“工作范围”，由指令直接设置限位位置，轴走到设置位置后立即采用快减减速度 STDEC（**ZAux_Direct_SetFastDec**）停止运动，且对应的轴状态位设置

为 1，它们应位于机床限制行进范围的相关硬限位开关的内侧。由于软限位开关的位置较为灵活，因此可根据当前的运行轨迹和具体要求调整轴的工作范围。

工作台碰到限位开关或者规划位置超越软限位时，运动控制器紧急停止工作台的运动。限位触发以后，轴无法继续运动，此时需要调整轴的位置，使其远离限位位置才能重新开始运动。

轴在碰到限位的时候才会产生停止信号，此时由于减速需要一定的时间，实际轴的位置会越过限位一定距离，假设停止时 SPEED（**ZAux_Direct_SetSpeed**）速度是 v_0，快减速 STDEC（**ZAux_Direct_SetFastDec**）为 a，计算公式：$v_t^2-v_0^2=2as$，带入下方数据：$0-100^2=2*(-1000)*s$，得出过冲的距离 s=5，由此可得，增大 STDEC（**ZAux_Direct_SetFastDec**）和减小 SPEED（**ZAux_Direct_SetSpeed**）都能达到减小过冲的目的。

9.1.3 例程

9.1.3-1 正负软限位设置



位	说明	打印值	
1	随动误差超限告警	2	2h
2	与远程轴通讯出错	4	4h
3	远程驱动器报错	8	8h
4	正向硬限位	16	10h
5	反向硬限位	32	20h
6	找原点中	64	40h
7	HOLD速度保持信号输入	128	80h
8	随动误差超限出错	256	100h
9	超过正向软限位	512	200h
10	超过负向软限位	1024	400h
11	CANCEL执行中	2048	800h
12	脉冲频率超过MAX_SPEED限制.需要修改降速或修改MAX_SPEED	4096	1000h
14	机械手指令坐标错误	16384	4000h
18	电源异常	262144	40000h
21	运动中触发特殊运动指令失败	2097152	200000h
22	告警信号输入	4194304	400000h
23	轴进入了暂停状态	8388608	800000h

```
// test1.cpp : 定义控制台应用程序的入口点。
//


#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"


void commandCheckHandler(const char *command, int ret)
{
```

```
    if (ret)//非 0 则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
    }

}

int _tmain(int argc, _TCHAR* argv[])
{

    char *ip_addr = (char *)"127.0.0.1";                //控制器 IP 地址
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        getchar();
        return -1;
    }
    printf("控制器连接成功! \n");

    ret =ZAux_Direct_SetAtype(handle,0,1);//设置轴类型为 1
    commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetAccel(handle,0,500);//设置轴加速度
    commandCheckHandler("ZAux_Direct_SetAccel", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetDecel(handle,0,500);//设置轴减速度
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetDpos(handle,0,0);//设置轴 DPOS 清 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//判断指令是否执行成功


    ret =ZAux_Direct_SetFsLimit(handle,0,500);//设置正向软限位为 500
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetRsLimit(handle,0,-500);//设置负向软限位为-500
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//判断指令是否执行成功
```

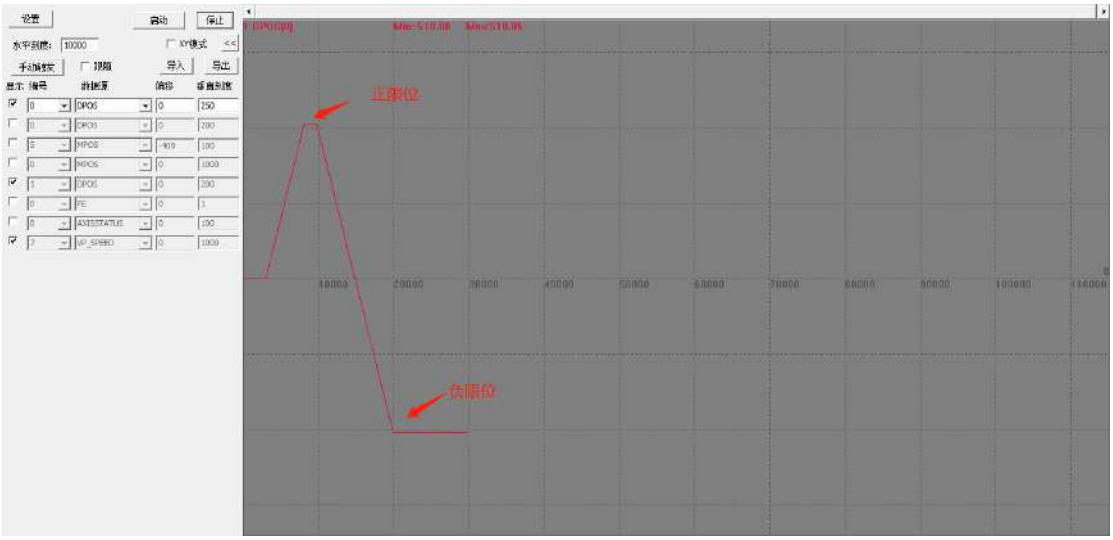
```
    float IDLE;
    int mode;
    int AxisStatus;
    ZAux_Trigger( handle);//触发示波器
    while(1)
    {
        printf("请输入运动模式，1 为正向运动，-1 为负向运动：\n");
        scanf("%d",&mode);

        if (mode!=1 && mode!=-1 && mode!=0)
        {
            printf("输入运动模式错误,%d\n",mode);
            Sleep(2000);
            return 0;
        }
        if (mode==0)
        {
            printf("退出\n");
            break;
        }
        ZAux_Direct_Single_Vmove(handle,0,mode);//轴 0 持续运动

        while (1)//等待轴 0 运动完成
        {
            Sleep(100);
            ZAux_Direct_GetParam(handle,"IDLE",0,&IDLE);
            if (IDLE<0)break;
        }
        ZAux_Direct_GetAxisStatus(handle,0,&AxisStatus);
        printf("AxisStatus 轴状态：%s %d：,\n",AxisStatus>0 ? "轴报警":"轴正常
",AxisStatus);
    }
    getchar();
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
```

```
    return 0;
}
```

输出波形图：



9.2 轴硬件限位函数

9.2.1 指令列表

指令	说明
ZAux_Direct_SetFwdIn	设置正向硬限位开关对应的输入点编号
ZAux_Direct_GetFwdIn	读取正向硬限位开关对应的输入点编号
ZAux_Direct_SetRevIn	设置负向硬件限位开关对应的输入点编号
ZAux_Direct_GetRevIn	读取负向硬件限位开关对应的输入点编号
ZAux_Direct_SetDatumIn	设置回零原点信号对应输入点编号
ZAux_Direct_GetDatumIn	读取回零原点信号对应输入点编号
ZAux_Direct_SetInvertIn	设置输入口有效状态反转
ZAux_Direct_GetInvertIn	读取输入口反转状态

9.2.2 重点说明

控制器为每个轴提供两个输入作为行程控制开关，需要安装硬件传感器，轴运动中碰到限位开关，立即报警停止：



注意 1：

如限位发送报警并报错 AXISSTATUS:10h（正限位报警），则继续操作轴时只能向负向运行。

如限位发送报警并报错 AXISSTATUS:20h（负限位报警），则继续操作轴时只能向正向运行。

如限位发送报警并报错 AXISSTATUS:30h（正限位报警+负限位报警），则需要检查光电开关等外设的有效电平是否与控制器设置相反。

注意 2：

切记设置快减减速度 STDEC（**ZAux_Direct_SetFastDec**）

把轴配置成正负限位有效后，如果轴运动超越了限位开关所在位置，限位开关触发，运动控制器禁止发出限位方向上运动，轴会紧急停止。此时由于减速需要一定的时间，实际轴的位置会越过限位一定距离，假设停止时 SPEED（**ZAux_Direct_SetSpeed**）速度是 v_0，快减速 STDEC（**ZAux_Direct_SetFastDec**）为 a，计算公式：$v_t^2-v_0^2=2as$，带入下方数据：$0-100^2=2*(-1000)*s$，得出过冲的距离 s=5，由此可得，增大 STDEC（**ZAux_Direct_SetFastDec**）和减小 SPEED（**ZAux_Direct_SetSpeed**）都能达到减小过冲的目的。

若未设置减速度，或者快减减速度，轴碰到限位，不会停止。

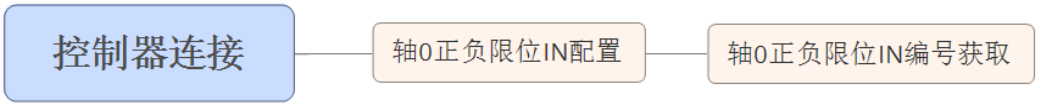
注意 3：

切记正负限位不要设置反。

碰到正限位，轴可正常往负方向跑，碰到负限位，轴可正常往正方向跑，若正负限位设置相反，则碰到正负限位，轴不可正常运动。

9.2.3 例程

9.2.3-1 正负硬限位设置




```
// test1.cpp : 定义控制台应用程序的入口点。
//
```

```
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";                //控制器 IP 地址
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("控制器连接成功! \n");

    int axislist[1] = { 0 };

    /*设置参数*/
    ret = ZAux_Direct_SetAtype(handle, axislist[0], 1);       //设置轴 0 的类型
    commandCheckHandler("ZAux_Direct_SetAtype", ret);
    ZAux_Direct_SetAtype(handle, axislist[0], 1);      //设置轴 0 轴类型为 1
    ZAux_Direct_SetUnits(handle, axislist[0], 100); //设置轴 0 脉冲当量为 100
    ZAux_Direct_SetSpeed(handle, axislist[0], 200); //设置轴 0 速度为 200units/s
    ZAux_Direct_SetAccel(handle, axislist[0], 2000); //设置轴 0 加速度为
```

```
2000units/s/s
    ZAux_Direct_SetDecel(handle, axislist[0], 2000); //设置轴 0 减速度为
2000units/s/s
    ZAux_Direct_SetFastDec(handle, axislist[0],3000);          //设置轴 0 的快减减速度
    float nvalue;
    ZAux_Direct_GetFastDec(handle, axislist[0],&nvalue);//获取轴 0 的快减减速度

    ret = ZAux_Direct_SetFwdIn(handle,axislist[0],1);          //映射正限位输入对应到
输入口 1
    commandCheckHandler("ZAux_Direct_SetFwdIn", ret);

    ZAux_Direct_SetRevIn(handle,axislist[0],2);                //映射负限位输入对应到
输入口 2
    commandCheckHandler("ZAux_Direct_SetRevIn", ret);

    ret = ZAux_Direct_SetInvertIn(handle,1, 0);                //设置原点信号表示低电
平有效
    commandCheckHandler("ZAux_Direct_SetInvertIn", ret);

    ret = ZAux_Direct_SetInvertIn(handle,2, 0);                //设置原点信号表示低电
平有效
    commandCheckHandler("ZAux_Direct_SetInvertIn", ret);

    /*获取参数*/
    int GetValue;

    ret = ZAux_Direct_GetAtype(handle, axislist[0], &GetValue);        //获取轴 0 的类
型
    printf("ZAux_Direct_GetAtype = %d \n", GetValue);

    ZAux_Direct_GetFwdIn(handle,axislist[0],&GetValue);        //获取设置正向限位输入
的 IN 编号
    printf("ZAux_Direct_GetFwdIn = %d \n", GetValue);

    ZAux_Direct_GetRevIn(handle,axislist[0],&GetValue);        //获取设置负向限位输入
的 IN 编号
    printf("ZAux_Direct_GetRevIn = %d \n", GetValue);
```

```
    Sleep(2000);

    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

9.3 轴报警函数

9.3.1 指令列表

指令	说明
ZAux_Direct_SetAlmIn	设置轴告警开关对应的输入点编号
ZAux_Direct_GetAlmIn	读取告警开关对应的输入点编号
ZAux_Direct_SetInvertIn	设置输入口有效状态反转
ZAux_Direct_GetInvertIn	读取输入口反转状态

9.3.2 重点说明

控制器提供专用的驱动报警信号输入接口。该接口应该与驱动器的报警输出信号接口相接。当检测到驱动器报警信号以后，运动控制器将关闭该轴的伺服使能。

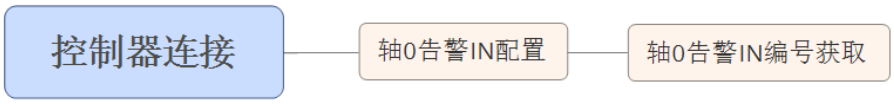
注意：

设置了告警输入口后，ZMC 控制器缺省为 OFF 有效，常开信号用 **ZAux_Direct_SetInvertIn** 反转电平;

ECI 控制器缺省为 ON 有效，常闭信号用 **ZAux_Direct_SetInvertIn** 反转电平。

9.3.3 例程

9.3.3-1 告警 IN 设置




```
// test1.cpp : 定义控制台应用程序的入口点。
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";                //控制器 IP 地址
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
```

```
    printf("控制器连接成功！\n");

    /*设置轴的相关参数*/
    ZAux_Direct_SetAlmIn(handle,0,0);//将轴 0 的告警信号定义到输入口 0
    commandCheckHandler("ZAux_Direct_SetAlmIn ",ret);

    ZAux_Direct_SetInvertIn(handle,0 ,0);//设置输入 0 低电平有效
    commandCheckHandler("ZAux_Direct_SetInvertIn ", ret);

    /*获取轴的相关参数*/
    int GetValue;

    ZAux_Direct_GetAlmIn(handle, 0, &GetValue);//获取轴 0 的告警信号输入口
    printf("ZAux_Direct_GetAlmIn = %d\n ", GetValue);

    Sleep(2000);
    ret = ZAux_Close(handle);     //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

第十章 板卡数据交互

10.1 VR 寄存器

10.1.1 指令列表

指令	说明
ZAux_Direct_SetVrf	设置 VR 寄存器
ZAux_Direct_GetVrf	读取 VR 寄存器
ZAux_Direct_GetVrInt	整型方式读取 VR 寄存器

10.1.2 重点说明

控制器具有电池保护寄存器或者铁电存储 VR，可以用于保存掉电的数据。ONPOWER 掉电中断可以用于触发记录掉电时的位置到 VR。

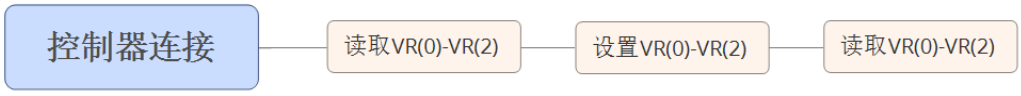
⚠只有 ZMC，XPLC 系列有 VR 掉电存储，ECI 系列有这个寄存器，实际不具备掉电存储功能。

⚠VR 寄存器以铁电的方式保存，最多可以保存 10 年左右。机器关键参数建议存储到 Flash。

⚠部分 ZMC，XPLC1/XPLC2 系列老产品是电池存储，正常使用几年没问题，可充电的，若长时间不开机会有影响。

10.1.3 例程

10.1.3-1 VR 寄存器的使用




```
// test1.cpp : 定义控制台应用程序的入口点。
//
```

```
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
    }

}
int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";                //控制器 IP 地址
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("控制器连接成功! \n");
    float getVrfData[3] = {0.0};

    ret = ZAux_Direct_GetVrf(handle, 0, 3, getVrfData);//读取 VR(0)-VR(2)
    commandCheckHandler("ZAux_Direct_GetVrf", ret);

    printf("getVrfData[0] = %f\n",getVrfData[0]);
    printf("getVrfData[1] = %f\n", getVrfData[1]);
    printf("getVrfData[2] = %f\n", getVrfData[2]);

    float setVrfData[3] = {1.1,2.2,3.3};
```

```
    //将数组中的值写入 vr
    ret = ZAux_Direct_SetVrf(handle, 0, 3, setVrfData);//设置 VR(0)-VR(2)
    commandCheckHandler("ZAux_Direct_SetVrf", ret);

    //取出刚才写入的值
    ret = ZAux_Direct_GetVrf(handle, 0, 3, getVrfData);//读取 VR(0)-VR(2)
    commandCheckHandler("ZAux_Direct_GetVrf", ret);

    printf("getVrfData[0] = %f\n",getVrfData[0]);
    printf("getVrfData[1] = %f\n", getVrfData[1]);
    printf("getVrfData[2] = %f\n", getVrfData[2]);

    Sleep(2000);
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

10.2 Table 寄存器

10.2.1 指令列表

指令	说明
ZAux_Direct_SetTable	设置系统 table 寄存器
ZAux_Direct_GetTable	读取系统 table 寄存器
ZAux_Direct_SetTableDouble	写入 Table
ZAux_Direct_GetTableDouble	读取 Table，可以一次读取多个

10.2.2 重点说明

系统数组 Table 寄存器

TABLE 是控制器自带的一个超大数组，数据类型为 32 位浮点型（4 系列及以上为 64 位浮点数），掉电不保存。编写程序时，TABLE 数组不需要再定义，可直接使用，索引下标

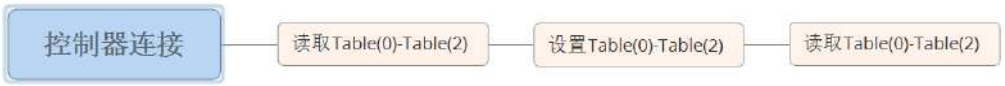
从 0 开始。系统缺省建立的全局数组，所有程序都可以访问。数据录波缓冲区，凸轮数据表，螺距补偿表，机械手参数，等等都是用 table 来存储。

RTBasic 的某些指令可以直接读取 TABLE 内的值作为参数，比如凸轮运动，凸轮表运动，同步运动，正逆解，位置比较输出等指令。

示波器采样的参数也存储在 TABLE 里。因此在开发应用中要注意多个 TABLE 区域的分配与使用，不要与示波器采样的数据存储区域重合。

10.2.3 例程

10.2.3-1 Table 寄存器的使用




```
// test1.cpp : 定义控制台应用程序的入口点。
//
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";                //控制器 IP 地址
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
```

```
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("控制器连接成功! \n");
    float getTableValue[3] = {0.0};
    //获取 table 的值
    ret = ZAux_Direct_GetTable(handle, 0, 3, getTableValue);
    commandCheckHandler("ZAux_Direct_GetTable", ret);
    //打印从 table 获取到的值
    printf("getTableValue[0] = %f\n",getTableValue[0]);
    printf("getTableValue[1] = %f\n", getTableValue[1]);
    printf("getTableValue[2] = %f\n", getTableValue[2]);

    float setTableValue[3] = {3.14,1.44,1.73};
    //设置 table
    ret = ZAux_Direct_SetTable(handle, 0, 3, setTableValue);
    commandCheckHandler("ZAux_Direct_SetTable", ret);

    //获取 table 的值
    ret = ZAux_Direct_GetTable(handle, 0, 3, getTableValue);
    commandCheckHandler("ZAux_Direct_GetTable", ret);

    //打印从 table 获取到的值
    printf("getTableValue[0] = %f\n",getTableValue[0]);
    printf("getTableValue[1] = %f\n", getTableValue[1]);
    printf("getTableValue[2] = %f\n", getTableValue[2]);

    Sleep(2000);
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

10.3 Modbus 寄存器

10.3.1 指令列表

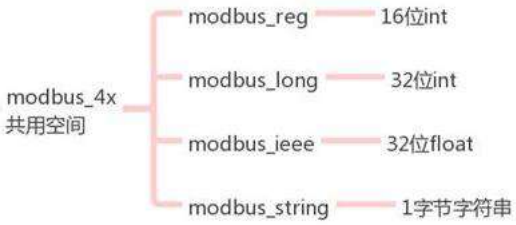
指令	说明
ZAux_Modbus_Set0x	设置 modbus 位寄存器
ZAux_Modbus_Get0x	读取 modbus 位寄存器
ZAux_Modbus_Set4x	设置 modbus 寄存器（REG）
ZAux_Modbus_Get4x	读取 modbus 寄存器（REG）
ZAux_Modbus_Set4x_Long	Modbus4x 寄存器写操作(LONG)
ZAux_Modbus_Get4x_Long	Modbus4x 寄存器读操作(LONG)
ZAux_Modbus_Set4x_String	Modbus 字寄存器操作（ASCII）
ZAux_Modbus_Get4x_String	Modbus 字寄存器操作（ASCII）
ZAux_Modbus_Set4x_Float	设置 modbus 寄存器(float 类型)
ZAux_Modbus_Get4x_Float	读取 modbus 寄存器(float 类型)

10.3.2 重点说明

MODBUS 寄存器符合 MODBUS 标准通讯协议，分为位寄存器和字寄存器两类。

位寄存器：MODBUS_BIT，触摸屏一般称为 MODBUS_0X，布尔型

字寄存器：MODBUS_REG、MODBUS_LONG、MODBUS_IEEE、MODBUS_STRING，触摸屏一般叫 MODBUS_4X，类型如下图。



控制器中字寄存器占用同一个存储空间，其中一个 LONG 占用两个 REG，使用时注意错开字寄存器编号。

MODBUS_LONG(0)占用 MODBUS_REG(0)与 MODBUS_REG(1)两个 REG 地址。

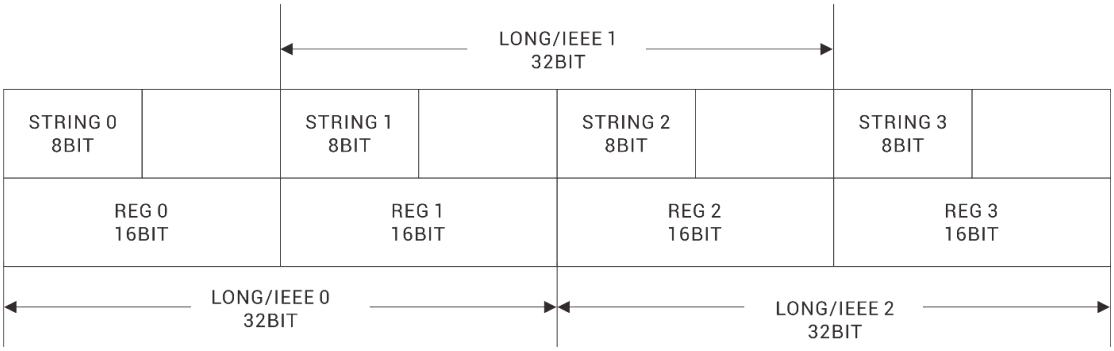
MODBUS_LONG(1)占用 MODBUS_REG(1)与 MODBUS_REG(2)两个 REG 地址。

MODBUS_IEEE(0)占用 MODBUS_REG(0)与 MODBUS_REG(1)两个 REG 地址。

MODBUS_IEEE(1)占用 MODBUS_REG(1)与 MODBUS_REG(2)两个 REG 地址。

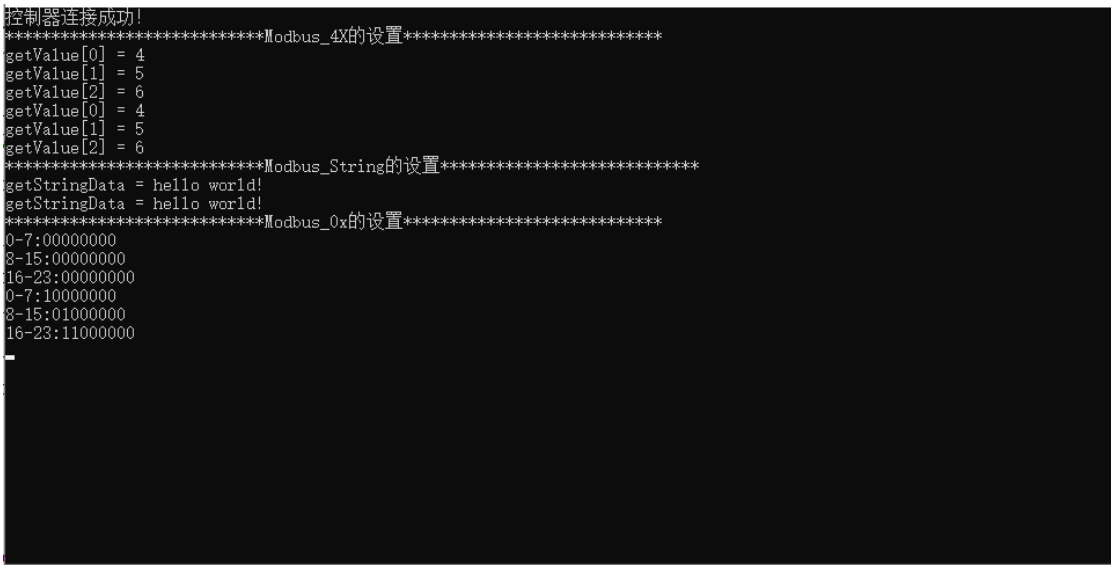
因此需注意 MODBUS_REG，MODBUS_LONG，MODBUS_IEEE 地址不能重叠。

4X 空间示意图



10.3.3 例程

10.3.3-1 Modbus 寄存器的使用




```
// test1.cpp : 定义控制台应用程序的入口点。
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非0则失败
    {
```

```
        printf("%s fail!return code is %d\n", command, ret);
    }

}


int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";                //控制器IP地址
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("控制器连接成功! \n");
    uint16 getValue[3];
    ZAux_Modbus_Get4x(handle, 0, 3, getValue);
    commandCheckHandler("ZAux_Modbus_Get4x", ret);
    printf("*****************************Modbus_4X的设置
*****************************\n");
    //打印获取到的值
    printf("getValue[0] = %d\n",getValue[0]);
    printf("getValue[1] = %d\n", getValue[1]);
    printf("getValue[2] = %d\n", getValue[2]);

    uint16 setValue[3] = {4,5,6};
    //设置modbus_4x数据，这里设置的是从0开始，共3个值
    //modbus_4x(0)设为4
    //modbus_4x(1)设为5
    //modbus_4x(2)设为6
    ret = ZAux_Modbus_Set4x(handle, 0, 3, setValue);
    commandCheckHandler("ZAux_Modbus_Set4x", ret);

    //获取modbus_4x的值，这里取是上面设置的值
    //getValue[0]的值是modbus_4x(0)中的值
    //getValue[1]的值是modbus_4x(1)中的值
    //getValue[2]的值是modbus_4x(2)中的值
    ZAux_Modbus_Get4x(handle, 0, 3, getValue);
    commandCheckHandler("ZAux_Modbus_Get4x", ret);
```

```
    //打印获取到的值
    printf("getValue[0] = %d\n",getValue[0]);
    printf("getValue[1] = %d\n", getValue[1]);
    printf("getValue[2] = %d\n", getValue[2]);
    printf("*****************************Modbus_String的设置
*****************************\n");
    char getStringData[13] = {0};
    //这里在modbus中取出"100-113"内容
    ZAux_Modbus_Get4x_String(handle,100,13,getStringData);
    commandCheckHandler("ZAux_Modbus_Get4x_String", 0);

    //打印获取到的字符串
    printf("getStringData = %s\n",getStringData);

    char setStringData[13] = "hello world!";
    //这里在modbus中写入一句"hello world!",第二个参数是开始写入的地址
    //字符串长度为13，所以第3个参数的值为13
    ret = ZAux_Modbus_Set4x_String(handle, 100, 13, setStringData);
    commandCheckHandler("ZAux_Modbus_Set4x_String", ret);



    //这里在modbus中取出"hello world!"
    ZAux_Modbus_Get4x_String(handle,100,13,getStringData);
    commandCheckHandler("ZAux_Modbus_Get4x_String", 0);

    //打印获取到的字符串
    printf("getStringData = %s\n",getStringData);
    uint8 GetValue[3];
    ret = ZAux_Modbus_Get0x( handle, 0, 24, GetValue);
    commandCheckHandler("ZAux_Modbus_Get0x" ,ret);

    printf("*****************************Modbus_0x的设置
*****************************\n");

    for (int j=0;j<=2;j++)
    {
        printf("%d-%d:",8*j,8*j+7 );
        for (int i=0;i<=7;i++)
        {
            putchar((GetValue[j]>>i & 1) ? '1' : '0');
        }
        printf("\n" );
    }
```

```
    uint8 SetValue[3] = {1, 2, 3};

    ret = ZAux_Modbus_Set0x( handle, 0, 24, SetValue);
    commandCheckHandler("ZAux_Modbus_Set0x",ret);

    ret = ZAux_Modbus_Get0x( handle, 0, 24, GetValue);
    commandCheckHandler("ZAux_Modbus_Get0x" ,ret);
    for (int j=0;j<=2;j++)
    {
        printf("%d-%d:",8*j,8*j+7 );
        for (int i=0;i<=7;i++)
        {
            putchar((GetValue[j]>>i & 1) ? '1' : '0');
        }
        printf("\n" );
    }

    Sleep(10000);
    ret = ZAux_Close(handle);     //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

10.4 Flash/文件读写

10.4.1 指令列表

指令	说明
ZAux_WriteUFile	向 U 盘中写文件
ZAux_ReadUFile	从 U 盘中读取文件
ZAux_FlashWritef	写控制器 FLASH 空间
ZAux_FlashReadf	读控制器 FLASH 空间
ZAux_DownMem3FileRam	打包字符串运动下发
ZAux_Run3FileRam	运行下发的 3 次文件
ZAux_Down3FileRam	下载 3 次文件到 RAM
ZAux_Down3FileRom	下载 3 次文件到 ROM

ZAux_3FileRamGetRemainSpace	动态下载 3 次文件，读取可写的空间
ZAux_3FileRamDownBegin	动态下载 3 次文件，启动下载
ZAux_3FileRamDownPart	动态下载 3 次文件
ZAux_3FileRamDownEnd	动态下载 3 次文件，结束下载
ZAux_Check3File	检测三次文件是否存在
ZAux_Delete3File	删除控制器上的文件
ZAux_FindFirst3File	查找控制器上的文件，文件名为空表示文件不不存在
ZAux_FindNext3File	查找控制器上的下个文件，文件名为空表示文件不不存在
ZAux_GetCur3File	获取控制器上的当前文件
ZAux_RemoveAll3Files	删除控制器上所有的三次的文件
ZAux_Up3File	上传三次文件
ZAux_Up3FileToMem	上传三次文件

10.4.2 重点说明

控制器上面有 U 盘接口和 flash。

用户可以通过 U 盘即可实现和控制器进行通讯，U 盘的文件格式和读写文件的格式/顺序必须保持一致。

存储 Flash 块(RTSys 在线命令?FLASH_SECTES 参数可以查看)，可以用于保存掉电的数据。

⚠ Flash 是有读写寿命限制，不可以无限次的擦写，经常改写的数据建议写入 VR。

U 盘接口

采用标准的 U 盘接口，用于插入 U 盘设备，支持与 U 盘数据交互。不同型号的控制器 U 盘接口的使用方法都是相同的，将 U 盘插在控制器上的 UDISK 端口即可，控制器上电后有 U 盘插入时，U 盘指示灯亮。

针脚号	名称	说明
1	V	内部+5V 电源
2	D-	差分数据 D-
3	D+	差分数据 D+
4	GND	内部电源地

在对 U 盘进行操作之前，先使用 U_STATE 指令判断 U 盘的状态，确保 U 盘能成功通讯，再使用 U 盘相关指令操作。（建议使用 2.0 的 U 盘）

U 盘接口主要有以下三个方面的用途：

1．程序升级

通过 U 盘口，下载打包好的 zar 加密程序包，方便客户更新系统程序。

2．加载三次文件

使用 U 盘指令加载 U 盘里保存的三次文件执行。

3．U 盘与寄存器数据交互

U 盘支持读写变量和数组。

⚠ 部分 U 盘格式不支持，U 盘读取文件系统支持 fat32 和 fat16，不支持 ntfs、exFAT 格式。

FLASH 数据拷贝：多个控制器中 FLASH 存储的数据可以通过 U 盘来相互传递。

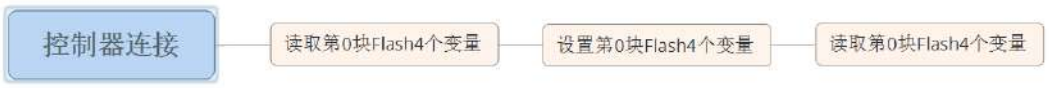
VR 寄存器、TABLE 寄存器与 U 盘里的数据互相传递。

读写文件类型为 SD(filenum).BIN 或 SD(filenum).CSV，不同的指令可操作的文件类型有所区别。

10.4.3 例程

10.4.3-1 Flash 的读写

下面的例子展示了 flash 读写的示例，先将数据写入控制器的第 0 块 flash，然后再读取 flash 中的数据。




```
// test1.cpp : 定义控制台应用程序的入口点。
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
```

```
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";                //控制器 IP 地址
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("控制器连接成功! \n");
    float readValues[4] = {0};
    uint32 readValuesNum = 0;
    //从 flash 的第 0 个块读取数据读取 flash 数据
    ret = ZAux_FlashReadf(handle, 0,4,readValues,&readValuesNum);
    commandCheckHandler("ZAux_FlashReadf", ret);

    //打印获取到的数据
    printf("readValues[0] = %f\n",readValues[0]);
    printf("readValues[1] = %f\n", readValues[1]);
    printf("readValues[2] = %f\n", readValues[2]);
    printf("readValues[3] = %f\n", readValues[3]);

    float writeValues[4] = { 1.0,2.0,3.0,4.0 };

    //将数据写入 flash 的第 0 个块
    ret = ZAux_FlashWritef(handle, 0, 4, writeValues);
    commandCheckHandler("ZAux_FlashWritef", ret);

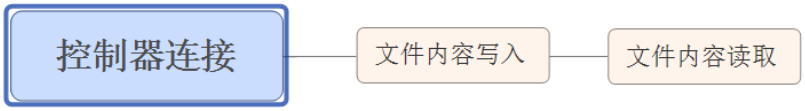
    //从 flash 的第 0 个块读取数据读取 flash 数据
    ret = ZAux_FlashReadf(handle, 0,4,readValues,&readValuesNum);
    commandCheckHandler("ZAux_FlashReadf", ret);
```

```
    //打印获取到的数据
    printf("readValues[0] = %f\n",readValues[0]);
    printf("readValues[1] = %f\n", readValues[1]);
    printf("readValues[2] = %f\n", readValues[2]);
    printf("readValues[3] = %f\n", readValues[3]);

    Sleep(2000);
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

10.4.3-2 U 盘文件的读写

将数据写入到文件，文件的格式须和 U 盘的文件格式保持一致，然后再读取写入的数据。




```
// test1.cpp : 定义控制台应用程序的入口点。
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
    }
}
```

```
int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";                //控制器 IP 地址
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("控制器连接成功! \n");

    //将数据存储到名为 test.txt 的文件，会在程序的相同路径下生成
    float writepvarlist[10] = {1.1,2.2,3.3,4.4,5.5,6.6,7.7,8.8,9.9,10.2};
    ret = ZAux_WriteUFile("test123.txt",writepvarlist,10);
    commandCheckHandler("ZAux_WriteUFile", ret);

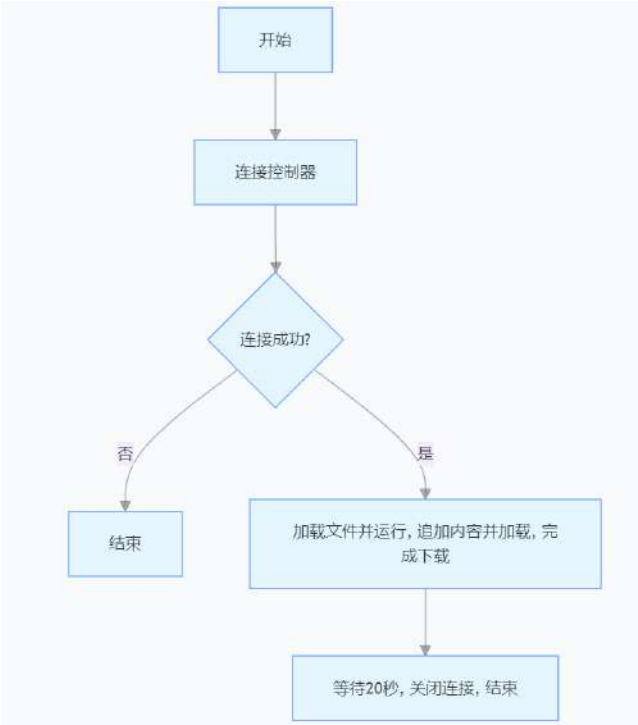
    float readpvarlist[10] = {0};
    int readnum = 0;
    //读取刚才存入文件的数据到浮点型数组
    ret = ZAux_ReadUFile("test123.txt", readpvarlist, &readnum);
    commandCheckHandler("ZAux_ReadUFile", ret);

    //打印读取到的结果
    printf("readpvarlist[0] = %f\n",readpvarlist[0]);
    printf("readpvarlist[1] = %f\n", readpvarlist[1]);
    printf("readpvarlist[2] = %f\n", readpvarlist[2]);
    printf("readpvarlist[3] = %f\n", readpvarlist[3]);
    printf("readpvarlist[4] = %f\n", readpvarlist[4]);
    printf("readpvarlist[5] = %f\n", readpvarlist[5]);
    printf("readpvarlist[6] = %f\n", readpvarlist[6]);
    printf("readpvarlist[7] = %f\n", readpvarlist[7]);
    printf("readpvarlist[8] = %f\n", readpvarlist[8]);
    printf("readpvarlist[9] = %f\n", readpvarlist[9]);
```

```
    Sleep(2000);
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

10.4.3-3 三次文件使用

三次文件，可以在多段运动之中，小线段过多，导致网口通信发送数据一次性发不完，多段发送跟不上运动速度执行效率，从而新增的一个三次文件功能。本例子使用仿真器，动态加载三次文件，动态追加字符串控制轴 0 的一个简易使用例子。




```
// test1.cpp : 定义控制台应用程序的入口点。
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非0则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
    }

}

int _tmain(int argc, _TCHAR* argv[])
```

```
{
    //步骤1：初始化连接控制器/仿真器
        char *ip_addr = (char *)"127.0.0.1";                //控制器IP地址
        ZMC_HANDLE handle = NULL;                           //连接句柄
        int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
        if (ERR_SUCCESS != ret)
        {
            printf("控制器连接失败!\n");
            handle = NULL;
            Sleep(2000);
            return -1;
        }
        printf("控制器连接成功! \n");

    //步骤2：开启3次文件动态加载
        uint32 iRemainSpace;
        uint32 nRemain;
        ret = ZAux_3FileRamDownBegin(handle, 0,  &iRemainSpace, "Zmc.z3p");
        commandCheckHandler("ZAux_3FileRamDownBegin",ret);
    //步骤3：编写文件内容字符串
        char  cmdbuff[262144];

        strcpy(cmdbuff, "SPEED(0)=100\n");
        strcat(cmdbuff, "ACCEL(0)=1000\n");
        strcat(cmdbuff, "DECEL(0)=1000\n");
        strcat(cmdbuff, "DPOS(0)=0\n");
        strcat(cmdbuff, "MERGE(0)=1\n");
        strcat(cmdbuff, "MOVE(100)axis(0)\n");
        strcat(cmdbuff, "MOVE(200)axis(0)\n");
        strcat(cmdbuff, "MOVE_RESUME\n");//首次结尾必须追加

        //若缓冲不足，等待3次文件缓冲足够下发
        ret = ZAux_3FileRamGetRemainSpace(handle, 0, &nRemain);
        commandCheckHandler("ZAux_3FileRamGetRemainSpace",ret);
        if (nRemain <= strlen(cmdbuff))
        {
            printf("拼接文件过大! \n");
            Sleep(5000);
            return 0;
        }
        //加载3次文件字符
        ret = ZAux_3FileRamDownPart(handle, 0, cmdbuff, (uint)strlen(cmdbuff),
&nRemain);
        commandCheckHandler("ZAux_3FileRamDownPart",ret);
```

```
        //运行3次文件
        ret = ZAux_Run3FileRam(handle, 0, 0);
        commandCheckHandler("ZAux_Run3FileRam",ret);



    //步骤3.5：动态追加加载
    //{若（文件/轨迹数据）不是超大，在第一步直接打包数据到位，运行跑完即可，无需动态追加，直接跳过步骤3.5，执行步骤4}

        strcpy(cmdbuff, "MOVE(1)axis(0)\n");
        for (int i=0;i<999;i++)
        {
            strcat(cmdbuff, "MOVE(1)axis(0)\n");
        }

        ret = ZAux_3FileRamGetRemainSpace(handle, 0, &nRemain);
        while (nRemain <= strlen(cmdbuff))
        {
            ret = ZAux_3FileRamGetRemainSpace(handle, 0, &nRemain);
            commandCheckHandler("ZAux_3FileRamGetRemainSpace",ret);
        }


        ret = ZAux_3FileRamDownPart(handle, 0, cmdbuff, (uint)strlen(cmdbuff),
&nRemain);//追加
        commandCheckHandler("ZAux_3FileRamDownPart",ret);


    //步骤4：全部下载完成
        ret = ZAux_3FileRamDownEnd(handle, 0);


        Sleep(20000);
    //步骤5：结束
        ret = ZAux_Close(handle);    //关闭连接
        commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
        printf("connection closed!\n");
        handle = NULL;
        return 0;
}
```

第十一章 总线相关

11.1 总线初始化

11.1.1 指令列表

指令	说明
ZAux_BusCmd_GetNodeNum	读取卡槽上节点数量
ZAux_BusCmd_GetNodeInfo	读取节点上的信息
ZAux_BusCmd_GetNodeStatus	读取节点总线状态
ZAux_BusCmd_DriveClear	设置清除总线伺服报警
ZAux_Direct_SetErrormask	设置错误标记
ZAux_Direct_GetErrormask	读取错误标记
ZAux_Direct_SetAxisEnable	设置轴使能，只针对总线控制器轴使用有效
ZAux_Direct_GetAxisEnable	读取轴使能状态，只针对总线控制器轴使用有效
ZAux_BusCmd_InitBus	总线初始化
ZAux_BusCmd_GetInitStatus	获取总线初始化状态
ZAux_Direct_GetBusAxisEnable	读取总线轴使能状态

11.1.2 重点说明

11.1.2-1 EtherCAT 协议描述

EtherCAT 是一项高性能、低成本、应用简易、拓扑灵活的工业以太网技术，可用于工业现场级的超高速 I/O 网络，使用标准的以太网物理层，传输媒体双绞线或光纤(100Base-TX 或 100Base-FX)。EtherCAT 系统由主站、从站组成。

总线卡 EtherCAT 通讯规格

项目		规格
通讯协议		IEC61800-7 CiA 402 Drive Profile
应用层	SDO	SDO 请求，SDO 应答
	PDO	可变 PDO 映射
	CiA 402	同步周期位置模式（CSP） 同步周期速度模式（CSV） 同步周期转矩模式（CST） HM(回零)

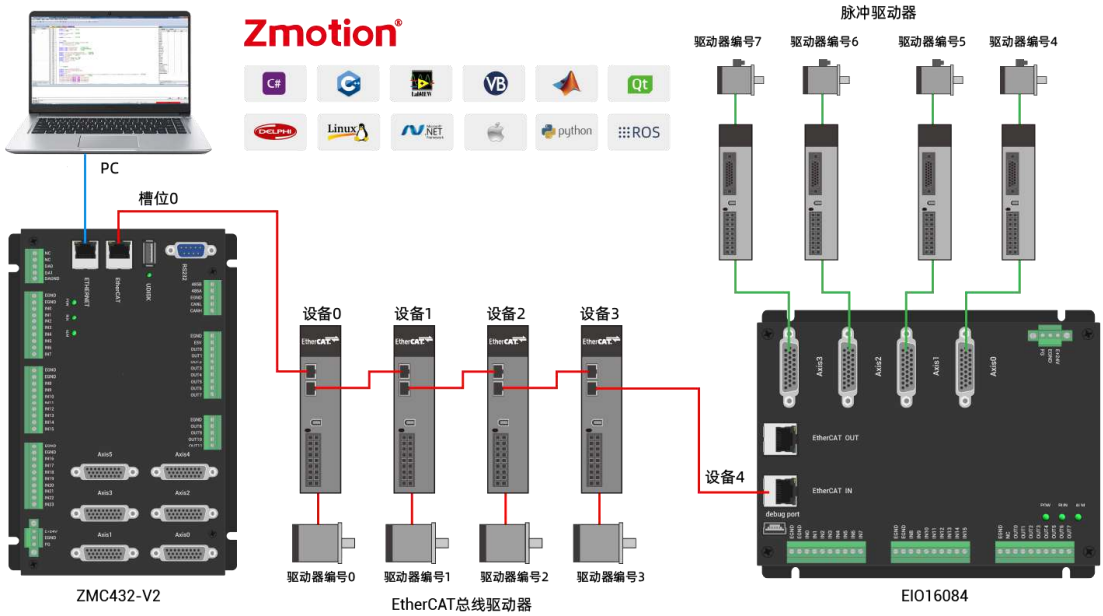
		TouchProbe(探针)
物理层	传输协议	100BASE-TX(IEEE802.3)
	最大距离	100M
	接口	RJ45(EtherCAT 口)

11.1.2-2 EtherCAT 总线接口

EtherCAT 总线接口与 EtherNET 网口一样采用 RJ45 标准网线接口，用于连接 EtherCAT 总线驱动器和扩展模块，部分控制器型号不支持 EtherCAT 总线。

EtherCAT 总线驱动器连接驱动器之后，使用前需要进行总线初始化操作，并使用指令映射驱动器的轴号、打开驱动器使能等，详情参见总线初始化参考程序。

支持总线轴和脉冲轴混合使用。EtherCAT 总线接线参考图如下。



基本概念：

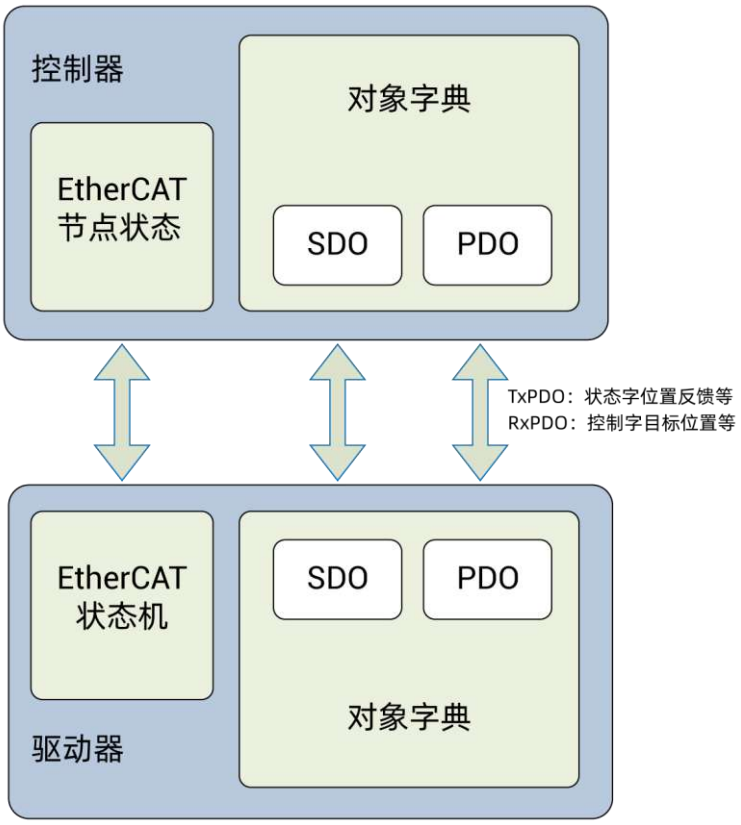
1.槽位号：槽位号即为控制器上的 EtherCAT 网口槽位号。默认只有一个 EtherCAT 的槽位，默认槽位号为 0。

2.节点编号：根据 EtherCAT 设备接线顺序排列，编号从 0 开始到（设备个数-1）。通过判断节点电机数，IO 数量来配置轴号与 IO 号等。

3.轴号：控制器对轴的编号，从 0 开始到（连接设备总轴数-1），通过 AXIS_ADDRESS 指令可以将轴号映射到任何一个连接的驱动器设备。

4.总线周期时间：默认 1ms。4 系列以上控制器支持修改总线同步周期时间 500us-4ms。

11.1.2-3 EtherCAT 通讯结构



结构图中，PDO 过程数据对象，包含了伺服驱动器运行过程中的实时数据，且以周期性地进行读写访问。SDO 邮箱通信，则以非周期性的对一些通信参数对象、PDO 过程数据对象，进行访问修改。

11.1.2-4 总线初始化

（1）脉冲控制器与总线控制器的区别

总线控制：总线控制是通过网线来进行发送命令规划位置值，总线控制器需要一段初始化程序才可以使能电机，进而再去控制电机。

脉冲控制：脉冲控制是直接打开对应的输出口使能电机，然后发送脉冲即可控制电机。

（2）总线初始化

总线初始化的过程（该过程为 basic 文件初始化过程）：

1. 节点扫描，扫描 EtherCAT 网络上所有的节点
2. 对 EtherCAT 节点，进行 PDO 预设置映射，如目标位置 607Ah 映射到轴的规划位置，6064h 反馈位置映射到轴的反馈位置等
3. 总线启动，切换驱动器状态机，“初始化—>预运行—>安全运行—>运行”

（3）总线回零

部分 EtherCAT 驱动器支持驱动器自身回零，并提供了多种回零方式。驱动器回零需要将原点、限位传感器开关直接接到驱动器对应 IO 端口中，（控制器方式回零时需要接到控制器 IO 口）。通过控制主站发送对应的回零命令即可完成回零动作。

11.1.2-5 总线模式

EtherCAT 总线常用的有三种控制模式，分别为 CSP 周期位置模式，CSV 周期速度模式，CST 周期转矩模式。

CSP、CSV、CST 模式的设置需要预先设置驱动器的 PDO 列表，PDO 包含对应的数据字典时，才可切换成该模式。驱动器默认 PDO 列表内置的数据字典需要查看驱动器手册确定。

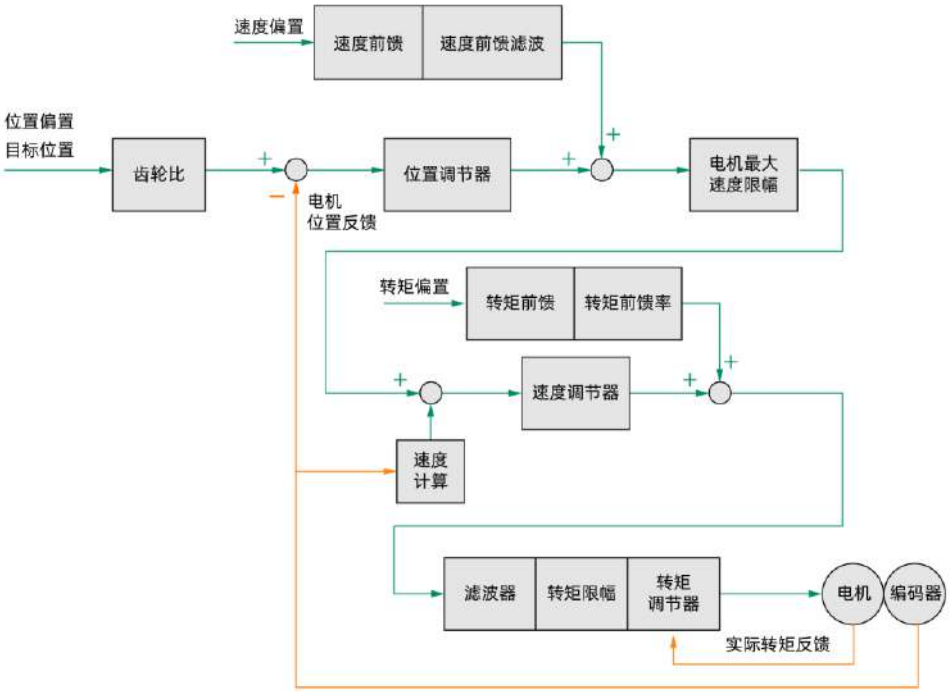
1. 当驱动器 PDO 列表包含数据字典 607Ah 时，轴类型可设置为 65，CSP 周期位置控制模式，此时使用位置运动指令控制电机运动，并设置运动速度。

2. 当驱动器 PDO 列表包含数据字典 60FFh 时，轴类型可设置为 66，CSV 周期速度控制模式，此时使用专用的速度指令控制电机以设置值的转速运行，不可使用位置运动指令。

3. 当驱动器 PDO 列表包含数据字典 6071h 时，轴类型可设置为 67，CST 周期转矩控制模式，此时使用专用的转矩指令控制电机以设置值的转矩运行，不可使用位置运动指令。

CSP 模式：

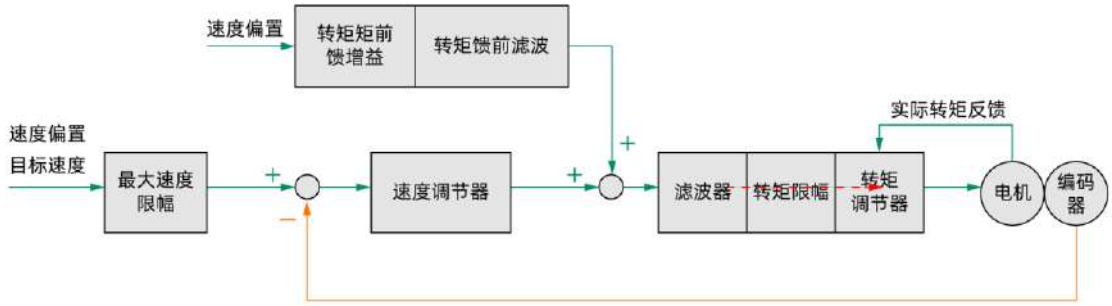
周期同步位置模式下，上位控制器完成位置指令规划，然后将规划好的目标位置周期性地发送给伺服驱动器，位置、速度、转矩控制由伺服驱动器内部完成。



CSP 模式图

CSV 模式：

周期同步速度模式下，上位控制器将计算好的目标速度周期性同步地发送给伺服驱动器，速度、转矩调节由伺服内部执行。

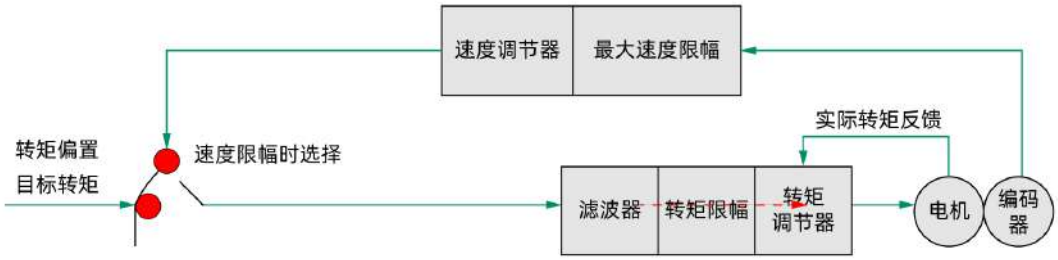


速度控制时需要看驱动器具体单位。

修改对应的 ATYPE 类型，然后在修改对应的 profile(速度控制设置为 20)即可，然后再通过在线命令或者程序中修改 dac 的值即可。

CST 模式:

周期同步转矩模式下，上位控制器将计算好的目标转矩周期性同步的发送给伺服驱动器，转矩调节由伺服内部执行。



EtherCAT 力矩控制需要设置对应的轴类型(67)，对应的将 profile 修改成 30 以上即可。

然后再通过在线命令或者程序中去修改 dac 的值去控制电机的转动(调整 dac 的值不易跨度过大)

RTEX 力矩模式:

需要修改对应的轴类型去修改对应的模式，然后再去修改对应的 profile 即可，详细请看 RTBasic 手册。

通过在线命令发送 dac 的值去控制电机的转动，也可以通过程序去控制电机的转动，力矩控制时 dac 的单位为千分之一的力矩，dac=1000 时表示 100%

指令
指令转矩 [大小]：带符号 32bit [单位]：0.1% [设定范围]：-电机最大转矩～电机最大转矩 * 最大转矩限制[%]＝100×Pr9.07/(Pr9.06×√2)

如果 dac(力矩)过小时，电机无法克服摩擦力，不能转动。

11.1.3 例程

11.1.3-1 EtherCAT 总线初始化（bas 方式）

（若不熟悉本公司的 RTBasic 语言，可以使用本公司免费提供的光盘资料里面的 basic 文件，“PC 函数库 V2.1\windows 平台\32 位库\C++\例程\例程 8-总线控制运动\总线初始化 BASIC 程序\ECAT 初始化.bas”）




```
// test1.cpp : 定义控制台应用程序的入口点。
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
```

```
    char *ip_addr = (char *)"192.168.0.11";                //控制器 IP 地址
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("控制器连接成功! \n");

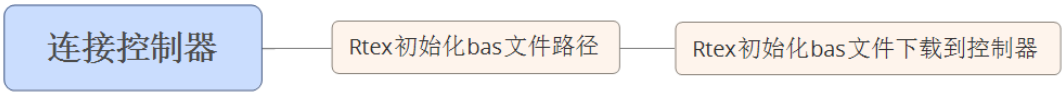
    const char * filename= ".\\ECAT 初始化.bas";

    //下载程序到 ROM 中  0-RAM 1-ROM
    ret = ZAux_BasDown(handle,filename,0);//把初始化 bas 文件下载到控制器里面进行初始
化
    commandCheckHandler("ZAux_BasDown",ret);

    Sleep(2000);
    ret = ZAux_Close(handle);     //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

11.1.3-2 Rtex 总线初始化（bas 方式）

（若不熟悉本公司的 RTBasic 语言，可以使用本公司免费提供的光盘资料里面的 basic 文件，“PC 函数库 V2.1\windows 平台\32 位库\C++\例程\例程 8-总线控制运动\总线初始化 BASIC 程序\RTEX 初始.bas”）



```
// test1.cpp : 定义控制台应用程序的入口点。
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"192.168.0.11";                //控制器 IP 地址
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("控制器连接成功! \n");
    //程序路径下
    const char * filename= ".\\RTEX 初始.bas";

    //下载程序到 ROM 中  0-RAM 1-ROM
    ret = ZAux_BasDown(handle,filename,0);//把初始化 bas 文件下载到控制器里面进行初始
化
    commandCheckHandler("ZAux_BasDown",ret);
```

```
    Sleep(2000);
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

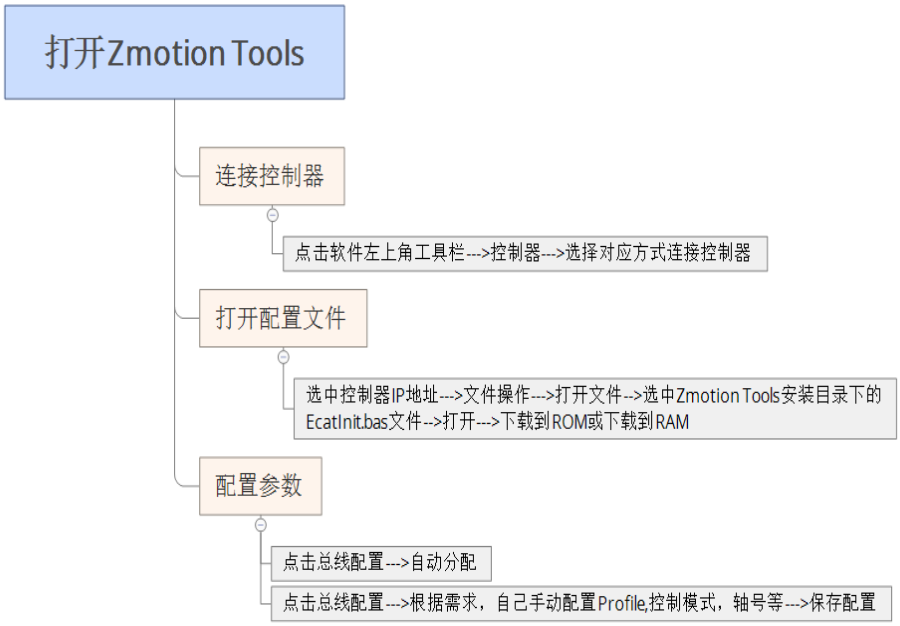
11.1.3-3 EtherCAT 总线轴的使能

在对 EtherCAT 总线轴进行使能前，需进行总线轴的初始化配置。该对 EtherCAT 总线进行初始化配置的方法有两种：方法 1.用本公司的 RTBasic 语言，编写一个对 EtherCAT 总线进行初始化配置的 bas 文件，在通过 ZAux_BasDown 函数把 Bas 文件下载到控制器中（若不熟悉本公司的 RTBasic 语言，可以使用本公司免费提供的光盘资料里面的 basic 文件，“PC 函数库 V2.1\windows 平台\32 位库\例程\例程 8-总线控制运动\总线初始化 BASIC 程序\ECAT 初始化.bas”），进行总线进行初始化配置；方法 2.通过使用本公司免费提供的 Zmotion Tools 小工具，对总线初始化配置；

该例程使用方法 2，通过 Zmotion Tools 工具，对总线进行初始化配置。

场景：当前控制器连接了一个 EtherCAT 总线电机，需对电机进行使能。

Zmotion Tools 工具配置流程框图：




```
// test1.cpp : 定义控制台应用程序的入口点。
//
```

```
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
    }

}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"192.168.0.11";                //控制器 IP 地址
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("控制器连接成功! \n");

    //ZAux_BusCmd_InitBus(handle);//总线初始化  （针对 Zmotion tools 工具软件配置过总线参数控制器使用有效）

     int GetValue;
     ret = ZAux_BusCmd_GetInitStatus(handle, &GetValue); //获取总线初始化状态 0：失败 1：成功（只针对 Zmotion tools 工具软件配置过总线参数控制器使用有效）
     commandCheckHandler("ZAux_BusCmd_GetInitStatus", ret);
     printf("总线初始化状态= %d \n", GetValue);
```

```
    int AxisEnable;
    ret = ZAux_Direct_GetAxisEnable(handle, 0, &AxisEnable);// 设置轴使能 0 表示关闭 1 表示打开
    commandCheckHandler("ZAux_BusCmd_GetInitStatus", ret);
    printf("轴 0 总线初始化使能状态为： %d \n", AxisEnable);

    if (AxisEnable<1)
    {
        ret = ZAux_Direct_SetAxisEnable(handle, 0, 1);// 设置轴使能 0 表示关闭 1 表示打开
        commandCheckHandler("ZAux_Direct_SetAxisEnable", ret);
        printf("打开使能\n");
    }
    else
    {
       ret = ZAux_Direct_SetAxisEnable(handle, 0, 0);// 设置轴使能 0 表示关闭 1 表示打开
        commandCheckHandler("ZAux_Direct_SetAxisEnable", ret);
        printf("关闭使能使能\n");
    }

   Sleep(2000);
   ret = ZAux_Close(handle);    //关闭连接
   commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
   printf("connection closed!\n");
   handle = NULL;
   return 0;
}
```

11.2 SDO/PDO 读写

11.2.1 指令列表

指令	说明
ZAux_BusCmd_SDOWrite	通过设备号和槽位号进行 SDO 写入
ZAux_BusCmd_SDORead	通过设备号和槽位号进行 SDO 读取

ZAux_BusCmd_SDOWrite2	写入节点 SDO 参数信息
ZAux_BusCmd_SDORead2	读取节点 SDO 参数信息
ZAux_BusCmd_RtexRead	Rtex 读取参数信息
ZAux_BusCmd_RtexWrite	Rtex 写参数信息
ZAux_BusCmd_SDOReadAxis	读轴 SDO 参数信息
ZAux_BusCmd_SDOWriteAxis	写轴 SDO 参数信息
ZAux_BusCmd_NodePdoWrite	Pdo 写操作
ZAux_BusCmd_NodePdoRead	Pdo 读操作
ZAux_BusCmd_NodePdoWriteFloat	Pdo 写操作
ZAux_BusCmd_NodePdoReadFloat	Pdo 读操作

11.2.2 重点说明

SDO 的数据交换使用 Mailbox 通信。因而请注意 SDO 的数据刷新时间变得不稳定。

主站侧在对象字典内的记录中读写数据，可进行对象设定以及从站的各种状态的监测。

注意：

SDO 的读写动作的响应需要花费时间。

ZAux_BusCmd_SDOWrite 和 ZAux_BusCmd_SDORead 总线指令，仅 EtherCAT 可以使用，通过设备号对槽位号进行 SDO 写入。

11.2.3 例程

11.2.3-1 EtherCAT 总线 SDO 读写

本例程是在总线初始化之后然后再新建工程之后再操作。




```
// test1.cpp : 定义控制台应用程序的入口点。
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
```

```
void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"192.168.0.11";                //控制器 IP 地址
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("控制器连接成功! \n");
    int slot = 0;
    int node = 0;
    int index = 0x6041;
    int subindex=0;
    int type = 6;
    int value;
    int setValue = 8;

    /*读取槽位 0 上面的状态字*/
    ret = ZAux_BusCmd_SDORead(handle,slot,node, index, subindex,type,&value);
    commandCheckHandler("ZAux_BusCmd_SDORead",ret);
    printf("Value=%d\n",value);

    /*设置槽位 0 上面的控制模式*/
    index = 0x6060;
```

```
    ret = ZAux_BusCmd_SDOWrite(handle,slot,node, index,subindex,2, setValue);
    commandCheckHandler("ZAux_BusCmd_SDOWrite", ret);

    /*读取槽位 0 上面的控制模式*/
    ret = ZAux_BusCmd_SDORead(handle,slot,node, index, subindex,type,&value);
    commandCheckHandler("ZAux_BusCmd_SDORead",ret);
    printf("Value=%d\n",value);

    Sleep(2000);
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

11.2.3-2 Rtex 参数相关信息

本例程是在总线初始化之后然后再新建工程之后再操作。

```
// test1.cpp : 定义控制台应用程序的入口点。
//
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
```

```
    char *ip_addr = (char *)"192.168.0.11";                //控制器 IP 地址
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("控制器连接成功! \n");
    float GetValue;
    int ipara = 7 * 256 + 11;
    /*设置 Pr7.11 的值*/
    ret = ZAux_BusCmd_RtexWrite(handle,0,ipara,500);

    /*读取 Pr7.11 的值*/
    ret = ZAux_BusCmd_RtexRead(handle,0, ipara,&GetValue);
    commandCheckHandler("ZAux_BusCmd_RtexRead", ret);
    printf("GetValue=%f\n", GetValue);

    Sleep(2000);
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

11.3 总线转矩力矩

11.3.1 指令列表

指令	说明
ZAux_BusCmd_GetDriveTorque	读取当前总线驱动当前力矩，需要设置对应的驱动器 PDO 类型

ZAux_BusCmd_SetMaxDriveTorque	设置当前总线驱动最大转矩，需要设置对应的驱动器 PDO 类型
ZAux_BusCmd_GetMaxDriveTorque	读取当前总线驱动最大转矩，需要设置对应的驱动器 PDO 类型
ZAux_Direct_SetDAC	模拟量控制。伺服轴 DA 直接控制，总线驱动速度或力矩模式下支持。
ZAux_Direct_GetDAC	模拟量控制。伺服轴 DA 直接控制，总线驱动速度或力矩模式下支持。

11.3.2 重点说明

电机转矩，简单的说，就是指转动的力量的大小。但电动机的转矩与旋转磁场的强弱和转子笼条中的电流成正比，和电源电压的平方成正比所以转矩是由电流和电压的因素所决定的。

电机扭矩和计算：T=9550 * P / n 。

P 是电机的额定（输出）功率单位是千瓦（KW）

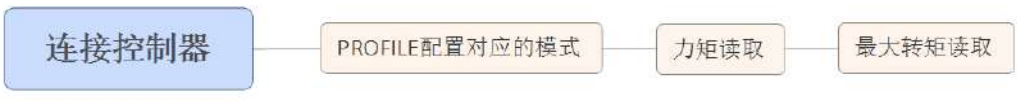
分母是额定转速 n 单位是转每分 (r/min)

P 和 n 可从 电机铭牌中直接查到。因为 P 和 n 都是电机的额定值，故 T 就是电机的额定转矩。一般来讲电动机带动机械转动的劲头就是电动机的转矩，电动机的转矩=皮带轮拖动皮带的力×皮带轮的半径。

注：必须设置正确的 ATYPE 与驱动器 PDO 以后才可以读取设置。

11.3.3 例程

11.3.3-1 获取当前总线驱动当前力矩




```
// test1.cpp : 定义控制台应用程序的入口点。
//
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
```

```
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"192.168.0.11";                //控制器 IP 地址
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("控制器连接成功! \n");

    int gValue;
    ret=ZAux_BusCmd_GetDriveTorque(handle,0,&gValue);//获取当前力矩，PROFILE 需配置对应的模式
    commandCheckHandler("ZAux_BusCmd_GetDriveTorque", ret) ;//判断指令是否执行成功
    printf("轴 0 当前力矩为：%d\n",gValue);

    ret=ZAux_BusCmd_GetMaxDriveTorque(handle,0,&gValue);//获取当前最大转矩，PROFILE 需配置对应的模式
    commandCheckHandler("ZAux_BusCmd_GetDriveTorque", ret) ;//判断指令是否执行成功
    printf("轴 0 当前最大转矩为：%d\n",gValue);

    Sleep(2000);
    ret = ZAux_Close(handle);    //关闭连接
```

```
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

第十二章 其他指令

12.1 在线命令

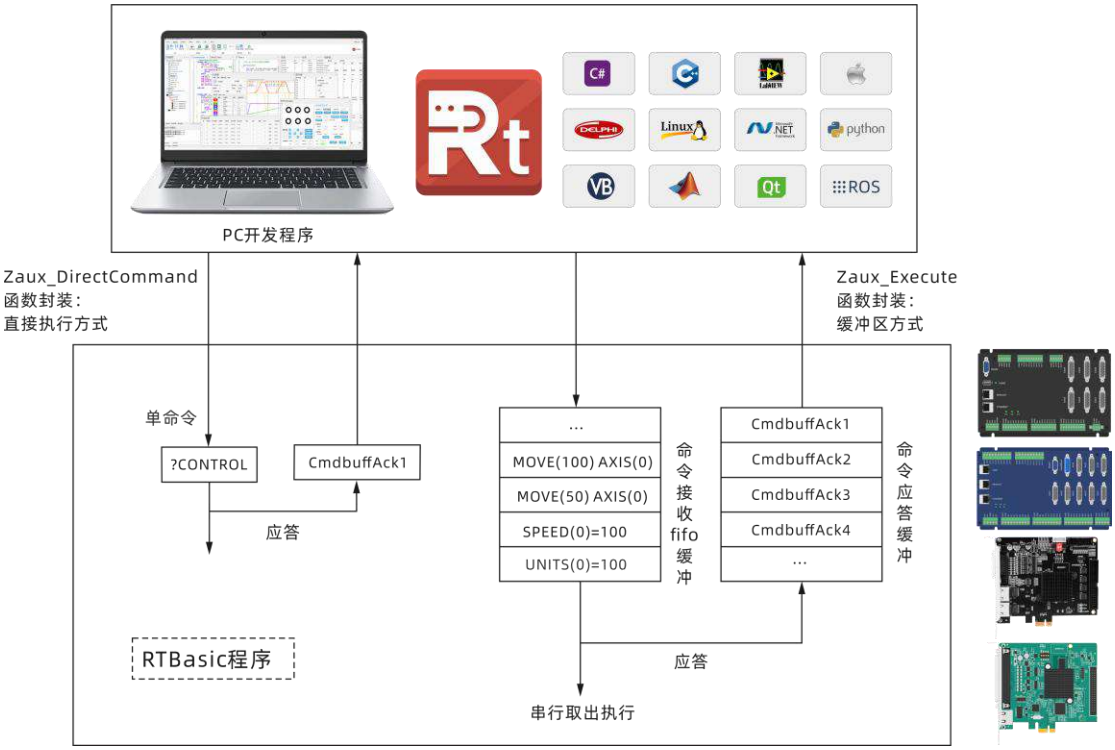
12.1.1 指令列表

指令	说明
ZAux_Execute	发送字符串（RTBasic）命令(缓冲方式)
ZAux_DirectCommand	发送字符串（RTBasic）命令(直接方式)

12.1.2 重点说明

ZAux_Execute 或 ZAux_DirectCommand 可对 basic 指令进行封装。如果使用到没有封装的命令或者想封装自己的函数，可以通过 ZAux_Execute 发送或 ZAux_DirectCommand，或是参照已有代码修改增加相应的函数。

发送字符串命令有两种方式，缓冲方式和直接方式。具体如图所示：



12.1.3 例程

12.1.3-1 在线命令函数的使用

本例程主要演示了如何使用 Execute 函数直接获取控制器的型号，如何使用 DirectCommand 函数封装设置速度功能指令示例。




```
// test1.cpp : 定义控制台应用程序的入口点。
//
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
    }
}
int  Test_Direct_SetSpeed(ZMC_HANDLE handle, int iaxis, float fValue)
{
    char cmdbuff[2048];
    char cmdbuffAck[2048];
    if (iaxis> MAX_AXIS_AUX)
    {
        return ERR_AUX_PARAERR;
    }
    sprintf(cmdbuff,"SPEED(%d)=%f",iaxis,fValue);//生成对应命令的字符串
    ZAux_DirectCommand(handle,cmdbuff,cmdbuffAck,2048);
}
/*参数：  mode      25, 26 2D 的比较模式
```

```
    Opnum      对应的输出口
    Opstate  第一个比较点的输出状态.
    maxerr    比较位置每个轴左右的脉冲偏差,  进入偏差范围后开始比较.
    num       TABLE 里面存储的比较点个数.
    tablepos 第一个比较点坐标所在 TABLE 编号

    与 hwtimer 并用时, 可以动态调整 hwtimer 参数.
    ModePara1:脉冲时间
    ModePara2:脉冲个数
    ModePara3:脉冲周期
    */
int32 __stdcall ZAux_Direct_HwPswitch2_2D(ZMC_HANDLE handle, int Axisnum,int Mode,int
Opnum , int Opstate,int maxerr,int num, int tablepos, float ModePara1=0, float
ModePara2=0,float ModePara3=0)
{
    if(0 > Axisnum || Axisnum > MAX_AXIS_AUX)
    {
        return  ERR_AUX_PARAERR;
    }
    char  cmdbuff[2048];
    char  cmdbuffAck[2048];
    //生成命令
    switch(Mode)
    {
    case 25:
        sprintf(cmdbuff, "HW_PSWITCH2(%d,%d,%d,%d,%d,%d) AXIS(%d)", Mode, Opnum,
Opstate, maxerr,num,tablepos,Axisnum);
        break;
    case 26:
        sprintf(cmdbuff, "HW_PSWITCH2(%d,%d,%d,%d,%d,%d,%f,%f,%f) AXIS(%d)", Mode,
Opnum, Opstate, maxerr,num,tablepos,ModePara1,ModePara2,ModePara3,Axisnum);
        break;

    }
    //调用命令执行函数
    return ZAux_Execute(handle, cmdbuff, cmdbuffAck,2048);
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";                //控制器 IP 地址
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
```

```
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("控制器连接成功! \n");

    const char * command="?control";            //发送获取控制器型号的命令
    char psRespons[20];                         //用于接受控制器型号的数组
    int length=20;                              //控制器型号的长度，一定要大于接受的字符串长度
    float GetValue1;
    ret = ZAux_Execute(handle, command, psRespons, length);   //直接通过 execute 直接获取控制的型号
    commandCheckHandler("ZAux_Execute",ret);

    printf("psRespons = %s\n", psRespons);  //打印返回的字符串
    printf("length = %d\n",length);

    ret=ZAux_Direct_GetSpeed(handle,0,&GetValue1);//获取设置速度前的速度值
    commandCheckHandler("ZAux_Direct_GetSpeed",ret);
    printf("设置速度前的速度值 = %f\n",GetValue1);

    ret=Test_Direct_SetSpeed(handle,0,1231);//使用自己封装的设置速度指令设置轴 0 速度
    commandCheckHandler("ZAux_Direct_GetSpeed",ret);

    ret=ZAux_Direct_GetSpeed(handle,0,&GetValue1);//获取设置速度后的速度值
    commandCheckHandler("ZAux_Direct_GetSpeed",ret);
    printf("设置速度后的速度值 = %f\n",GetValue1);
    Sleep(2000);
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");

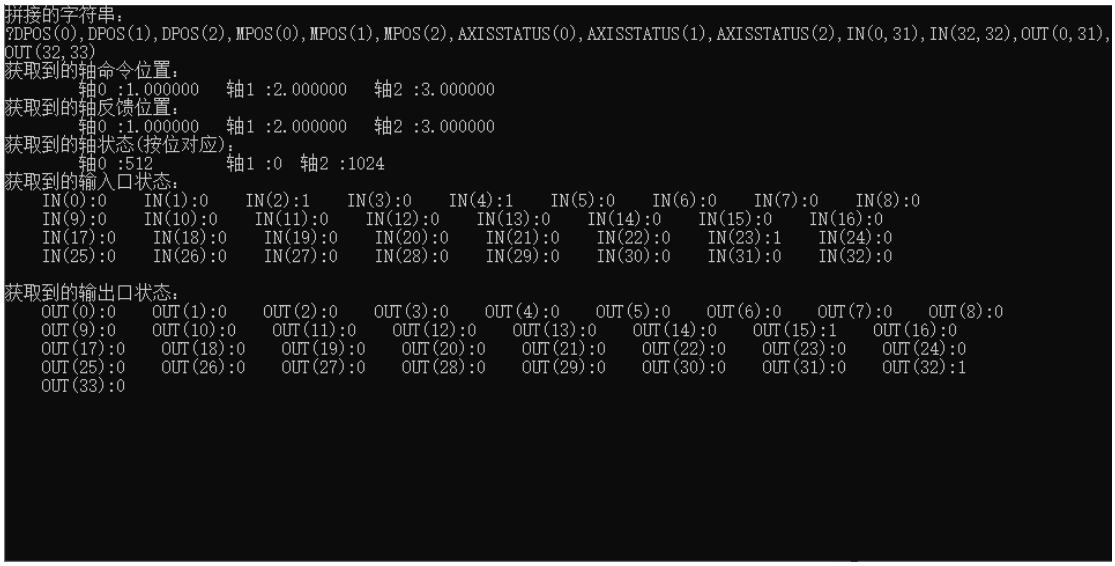
    handle = NULL;
```

```
    return 0;
}
```

12.1.3-2 在线命令函数的使用 2

用户若想要获取多种数据，如轴的命令位置，轴的反馈位置，板卡上的 IO 点等等，往往都是通过多种单独独立的函数获取不同的数据，这样堆积，会导致读写次数的上位，导致程序的卡顿。为了提升一个上位程序的，往往可以通过自己定义一个自己的函数，通过一个函数，快速的把数据传输到上位程序上面来，而非是通过多次循环来获取到不同类型的数据。如下举例：

假设有一个简易的三轴平台，需要读取轴 0，轴 1，轴 2 的命令位置，反馈位置，以及控制器板卡上的输入口 0-输入口 23,输出口 0-输出口，以及三个轴的状态。获取数据程序如下：




```
// test1.cpp : 定义控制台应用程序的入口点。
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非0则失败
    {
```

```
        printf("%s fail!return code is %d\n", command, ret);
    }

}

/************************************************************
Description:    //我的自定义直接获取数据函数
Input:          //handle            卡链接
                  iaxisNum          轴的总数量
                  iaxislist             轴号列表
                  fDposlist             输出的命令位置值
                  fMposlist             输出的反馈位置值
                  iAxisstatuslist       输出的轴状态位置值，按位对应
                  startIn               要获取起始的IN编号
                  endIn                 要获取结束的IN编号
                  iIn                   输出的IN状态,按位对应
                  startOut          要获取起始的OUT编号
                  endOut                要获取结束的OUT编号
                  iOut              输出的OUT状态,按位对应
Output:         //
Return:         //错误码
************************************************************/
int  Demo_Direct_MyGetData(ZMC_HANDLE handle,int iaxisNum, int* iaxislist, float*
fDposlist,float* fMposlist,int32* iAxisstatuslist,int startIn , int endIn,int
*iIn,int startOut , int endOut,int *iOut)
{
    char cmdbuff[2048];
    char  tempbuff[2048];
    char cmdbuffAck[20480];

    //若传进来的地址为空，则退出

    if(NULL == iaxislist || NULL == fDposlist || NULL == fMposlist ||  NULL ==
iAxisstatuslist || NULL == iIn || NULL == iOut)
    {
        return  ERR_AUX_PARAERR;
    }
    //若传进来的结束编号小于起始编码，则退出
    if ((endIn<startIn)  ||  (endOut<startOut))
    {
        return  ERR_AUX_PARAERR;
    }

    int ret=0;
```

```
int i;
//生成命令
sprintf(cmdbuff, "?");
//拼接DPOS
for (i=0;i<iaxisNum;i++)
{
    sprintf(tempbuff,"DPOS(%d),",iaxislist[i]);//生成对应命令的字符串

    strcat(cmdbuff, tempbuff);//字符串拼接

    if (strlen(cmdbuff)>1000)
    {
        return  ERR_AUX_PARAERR;  //参数错误，字符串拼接过长
    }
}
//拼接MPOS
for (i=0;i<iaxisNum;i++)
{
    sprintf(tempbuff,"MPOS(%d),",iaxislist[i]);//生成对应命令的字符串

    strcat(cmdbuff, tempbuff);//字符串拼接

    if (strlen(cmdbuff)>1000)
    {
        return  ERR_AUX_PARAERR;  //参数错误，字符串拼接过长
    }
}
//拼接AXISSTATUS
for (i=0;i<iaxisNum;i++)
{
    sprintf(tempbuff,"AXISSTATUS(%d),",iaxislist[i]);//生成对应命令的字符串

    strcat(cmdbuff, tempbuff);//字符串拼接

    if (strlen(cmdbuff)>1000)
    {
        return  ERR_AUX_PARAERR;  //参数错误，字符串拼接过长
    }
}
int32 ostart,istart,iend,oend;      //一次最多32个
bool addflag;

addflag=false;
```

```
int32 temp;       //一次最多32个
int32 temp2;      //一次最多32个

temp=endIn-startIn+1;
if (temp%32 == 0)
{
    temp=temp/32;
}
else
{
    temp=temp/32+1;
}

//拼接IN
for (i=0;i<temp;i++)
{
    istart = startIn+32*i;
    iend =istart+31;
    if (iend>endIn)
    {
        iend=endIn;
    }
    //生成命令
    sprintf(tempbuff, "IN(%d,%d),", istart,iend);
    strcat(cmdbuff, tempbuff);//字符串拼接
    if (strlen(cmdbuff)>1000)
    {
        return ERR_AUX_PARAERR ;  //参数错误，字符串拼接过长
    }

}

temp2=endOut-startOut+1;
if (temp2%32 == 0)
{
    temp2=temp2/32;
}
else
{
    temp2=temp2/32+1;
}
//拼接OUT
for (i=0;i<temp2;i++)
{
```

```
        ostart = startOut+32*i;
        oend =ostart+31;
        if (oend>endOut)
        {
            oend=endOut;
        }


        //生成命令
        sprintf(tempbuff, "OUT(%d,%d)", ostart,oend);

        strcat(cmdbuff, tempbuff);//字符串拼接
        if (i<temp-1)
        {
            strcat(cmdbuff, ",");//字符串拼接
        }

        if (strlen(cmdbuff)>1000)
        {
            return  ERR_AUX_PARAERR;   //参数错误，字符串拼接过长
        }

    }

    printf("拼接的字符串：\n",cmdbuff);
    printf("%s\n",cmdbuff);

    ret=ZAux_DirectCommand(handle,cmdbuff,cmdbuffAck,2048);
    if(ERR_OK != ret)
    {
        return ret;
    }
    //printf("%s\n",cmdbuffAck);
    //printf("%d\n",strlen(cmdbuffAck));
    //
    if(0 == strlen(cmdbuffAck))
    {
        return ERR_NOACK;
    }
    float ftempbuff[200];
    int itempbuff[200];

    ZAux_TransStringtoFloat(cmdbuffAck,iaxisNum*2,ftempbuff);//字符串转换为浮点数
    //DPOS输出
```

```
    for(i=0;i<iaxisNum;i++)
    {
        //printf("%f\n",ftempbuff[i]);
        fDposlist[i]=ftempbuff[i];

    }
    //MPOS输出
    for(i=0;i<iaxisNum;i++)
    {
        //printf("%f\n",ftempbuff[i+iaxisNum]);
        fMposlist[i]=ftempbuff[i+iaxisNum];
    }

    ZAux_TransStringtoInt(cmdbuffAck,iaxisNum*3+temp2+temp,itempbuff);//字符串转换为整形
    //AXISSTATUS输出
    for(i=0;i<iaxisNum;i++)
    {
        //printf("%d\n",itempbuff[i+iaxisNum*2]);
        iAxisstatuslist[i]=itempbuff[i+iaxisNum*2];
    }
    //IN输出
    for(i=0;i<temp;i++)
    {
        //printf("%d\n",itempbuff[i+iaxisNum*3]);
        iIn[i]=itempbuff[i+iaxisNum*3];
    }
    //OUT输出
    for(i=0;i<temp2;i++)
    {
        //printf("%d\n",itempbuff[i+iaxisNum*3+temp]);
        iOut[i]=itempbuff[i+iaxisNum*3+temp];
    }
    return ERR_OK;
}
int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";                //控制器IP地址
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
```

```
        Sleep(2000);
        return -1;
    }
    printf("控制器连接成功！\n");

    int axis[4]={0,1,2,4};
    float d_dpos[4];
    float d_mpos[4];
    int32 d_axis_status[4];
    int d_in[10];
    int d_out[10];

    ret =
Demo_Direct_MyGetData(handle,3,axis,d_dpos,d_mpos,d_axis_status,0,32,d_in,0,33,d_ou
t);
    int i;
    printf("获取到的轴命令位置：\n");
    for (i=0;i<3;i++)
    {
        printf("\t轴%d :%f",i,d_dpos[i]);
    }
    printf("\n");
    printf("获取到的轴反馈位置：\n");
    for (i=0;i<3;i++)
    {
        printf("\t轴%d :%f",i,d_mpos[i]);
    }
    printf("\n");
    printf("获取到的轴状态(按位对应)：\n");
    for (i=0;i<3;i++)
    {
        printf("\t轴%d :%d",i,d_axis_status[i]);
    }
    printf("\n");

    printf("获取到的输入口状态：\n");
    int j=0;
    int tempval;
    for (i=0;i<=32;i++)
    {
        if (((i%32)==0)&&(i>0) )
        {
            j++;
        }
```

```
        //转换成位
        tempval=d_in[j]>>(i-32*j);
        printf("    IN(%d):%d",i,tempval &(0x01));
        if (((i%8)==0)&&(i>0) )
        {
            printf("\n");
        }
    }
    printf("\n");

    printf("获取到的输出口状态：\n");
    j=0;

    for (i=0;i<=33;i++)
    {
        if (((i%32)==0)&&(i>0) )
        {
            j++;
        }
        //转换成位
        tempval=d_out[j]>>(i-32*j);
        printf("    OUT(%d):%d",i,tempval &(0x01));
        if (((i%8)==0)&&(i>0) )
        {
            printf("\n");
        }
    }
    printf("\n");

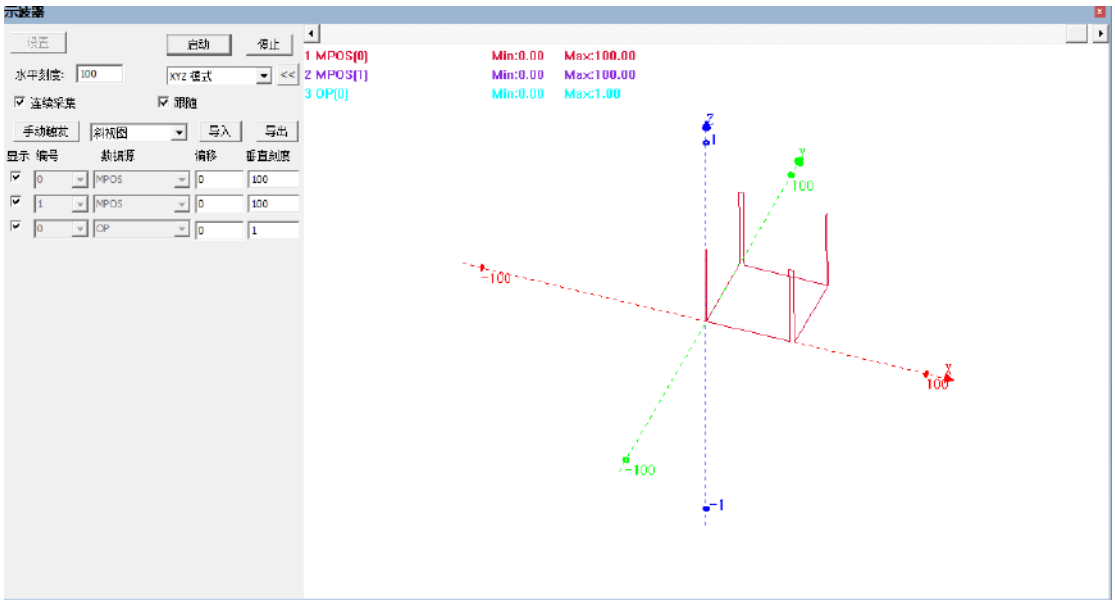
    Sleep(20000);
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");

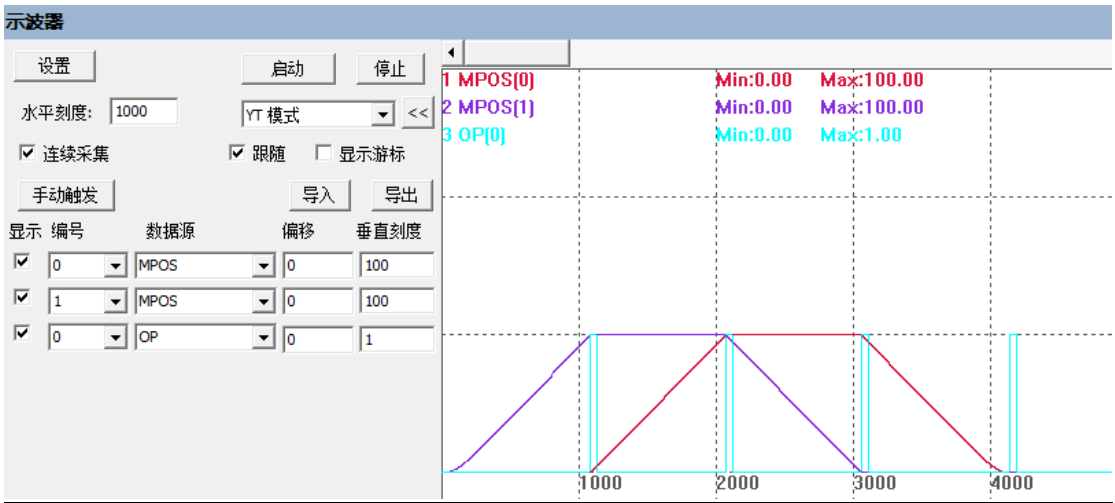
    handle = NULL;
    return 0;
}
```

12.1.3-3 在线命令函数的使用 3

一般点胶行业，木工行业用的居多，都是用连续轨迹，连续轨迹之间有插入缓冲输出，如果把运动跟连续轨迹的运动跟缓冲输出分开来发的话，难免会有局限性，可以通过自己单独封装运动函数，来达到一行命令执行多个函数的效果，如下举例：

假设控制一个 XY 两轴平台，从坐标点（0,0）—>（100,0）（输出口 0 输出 50ms）—>（100,100）（输出口 0 输出 50ms）—>（0,100）（输出口 0 输出 50ms）—>（0,0）（输出口 0 输出 50ms）的轨迹，则可以通过自己封装，用一条函数，快速发送下去：






```
// test1.cpp : 定义控制台应用程序的入口点。
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
```

```
void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非0则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
    }

}

/************************************************************
Description:    //我的自定义运动函数
Input:          //handle            卡链接
                  iMoveLen          填写的运动长度
                  iaxisNum          参与运动总轴数
                  iaxislist             轴号列表
                  fPoslist          距离列表
                  iout              缓冲输出口
                  outlist               缓冲输出列表(每条运动，决定是否输出，0为不输出，1为在运动后输出)
                  outtime               缓冲输出时间

Output:         //
Return:         //错误码
************************************************************/
int  Demo_Direct_MyMoveABS(ZMC_HANDLE handle,int iMoveLen,int iaxisNum, int* iaxislist, float* fPoslist,int iout,int *outlist,int outtime)
{
    char cmdbuff[2048];
    char  tempbuff[2048];
    char cmdbuffAck[20480];

    //若传进来的地址为空，则退出


    int ret=0;
    int i;
    //先读取剩余直线缓冲
    int iBuffLen = 0;
    ret = ZAux_Direct_GetRemain_LineBuffer(handle,iaxislist[0],&iBuffLen);
    if(iBuffLen <= iMoveLen*2)
    {
        return 1002;          //运动缓冲不够
    }
```

```
    //生成命令
    sprintf(cmdbuff, "BASE(");
    //拼接运动轴列表
    for (i=0;i<iaxisNum-1;i++)
    {
        sprintf(tempbuff,"%d,",iaxislist[i]);//生成对应命令的字符串

        strcat(cmdbuff, tempbuff);//字符串拼接

        if (strlen(cmdbuff)>1000)
        {
            return  ERR_AUX_PARAERR;  //参数错误，字符串拼接过长
        }
    }
    sprintf(tempbuff,"%d)\n",iaxislist[i]);//生成对应命令的字符串
    strcat(cmdbuff,tempbuff);

    //拼接运动
    for (i=0;i<iMoveLen;i++)
    {
        //printf("%d,%d\n",i*iaxisNum,i*iaxisNum+1);
        if (outlist[i]==0)    //该段运动不输出
        {
            strcat(cmdbuff, "MoveAbs(");

    sprintf(tempbuff,"%f,%f)\n",fPoslist[i*iaxisNum],fPoslist[i*iaxisNum+1]);//生成
对应命令的字符串
            strcat(cmdbuff, tempbuff);//字符串拼接

        }
        else if (outlist[i]==1)
        {
                strcat(cmdbuff, "MoveAbs(");

    sprintf(tempbuff,"%f,%f)\n",fPoslist[i*iaxisNum],fPoslist[i*iaxisNum+1]);//生成
对应命令的字符串
                strcat(cmdbuff, tempbuff);//字符串拼接

                strcat(cmdbuff, "Move_op2(");
                sprintf(tempbuff,"%d,%d,%d)\n",iout,1,outtime);//生成对应命令的字符
串
                strcat(cmdbuff, tempbuff);//字符串拼接
        }
```

```
        else
        {
            return  ERR_AUX_PARAERR;//参数错误
        }


    }


    printf("拼接的字符串：\n");
    printf("%s\n",cmdbuff);

    if (strlen(cmdbuff)>1000)
    {
        return  ERR_AUX_PARAERR;   //参数错误，字符串拼接过长
    }
    ret=ZAux_DirectCommand(handle,cmdbuff,cmdbuffAck,2048);

    return ret;


}
int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";                //控制器IP地址
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("控制器连接成功！\n");

    ret =ZAux_Direct_SetAtype(handle,0,1);//设置轴0轴类型为1
    commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetAtype(handle,1,1);//设置轴1轴类型为1
    commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetUnits(handle,0,100);//设置轴0脉冲当量为100
    commandCheckHandler("ZAux_Direct_SetUnits", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetUnits(handle,1,100);//设置轴1脉冲当量为100
```

```
    commandCheckHandler("ZAux_Direct_SetUnits", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetAccel(handle,0,500);//设置轴0加速度
    commandCheckHandler("ZAux_Direct_SetAccel", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetAccel(handle,1,500);//设置轴1加速度
    commandCheckHandler("ZAux_Direct_SetAccel", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetDecel(handle,0,500);//设置轴0减速度
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetDecel(handle,1,500);//设置轴1减速度
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetDpos(handle,0,0);//设置轴0 DPOS清0
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//判断指令是否执行成功
    ret =ZAux_Direct_SetDpos(handle,1,0);//设置轴1 DPOS清0
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetSpeed(handle,0,100);//设置轴0速度
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetSpeed(handle,1,100);//设置轴1速度
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetMerge(handle,0,1);//设置开启连续插补（开启主轴的即可，如轴
0，轴1插补，轴0为主轴，主轴号取决于连续插补运动指令轴列表的第一个轴号）



    int axis[2]={0,1};
    float POS[12]={0,0,0,100,100,100,100,0,0,0};
    int otlist[5]={0,1,1,1,1};

    ZAux_Trigger(handle);//触发示波器

    ret = Demo_Direct_MyMoveABS(handle,5,2,axis,POS,0,otlist,50);//
    commandCheckHandler("Demo_Direct_MyMoveABS", ret) ;//判断指令是否执行成功

    Sleep(20000);
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");

    handle = NULL;
    return 0;
}
```

12.2 控制器 RTBasic 相关

12.2.1 指令列表

指令	说明
ZAux_Pause	暂停控制器内部运行的 RTBasic 程序。
ZAux_Resume	恢复控制器内部运行的暂停的 RTBasic 程序。
ZAux_BasDown	单个.bas 文件生成 ZAR 并且下载到控制器运行
ZAux_ZarDown	下载 ZAR 文件到控制器
ZAux_ZpjDown	下载 Zpj 工程文件到控制器
ZAux_Direct_GetVariablef	全局变量读取，也可以是参数等，返回值为浮点型
ZAux_Direct_GetVariableInt	全局变量读取，也可以是参数等，返回值为整型
ZAux_Direct_SetUserArray	设置 RTBasic 自定义全局数组
ZAux_Direct_GetUserArray	读取 RTBasic 自定义全局数组可以一次读取多个
ZAux_Direct_SetUserVar	设置自定义全局变量
ZAux_Direct_GetUserVar	读取自定义全局变量
ZAux_SetTimeOut	命令的延时等待时间
ZAux_TransStringtoFloat	字符串转为 float
ZAux_TransStringtoInt	字符串转为 int
ZAux_TransStringtoDouble	字符串转为 double
ZAux_Direct_Base	指定控制器运动默认使用轴

12.2.2 重点说明

12.2.2-1 RTBasic 程序概述

RTBasic 所指的控制程序为下载在控制器中独立运行的程序。用户想使用控制程序，需要了解相应的语法及指令，以及下载步骤。针对不同系列控制器独立运行程序空间大小有较大差别，具体资源可查询对应型号产品手册。

独立控制程序优势：

独立性：控制程序能够脱离 PC 独立执行，不需占用 PC 任务资源，PC 只需要简单逻辑控制即可完成整个动作控制。

实时性：由于 WINDOWS 并不是一个实时系统，应用程序的任务调度完全受 PC 控制，应用程序与控制器之间需要不断的进行数据交互，所以往往在某些高速响应控制，通过 PC 的应用程序是无法及时响应的。

保密性：开发可以将不可导出的独立的控制程序提前下载或者生成下载包放在应用程序上且文件不可上载，并可加密为 zar 文件下载。可以通过对控制程序加密实现应用程序的加密，或者加密部分控制功能，保护开发人员利益。

12.2.2-2 PC 端如何使用 RTBasic

1. 控制程序编写

控制程序的编写需要在 xxx.bas 文件里面进行编写，语法格式为 RTBasic。（可以找对应的技术支持，或者光盘资料下载对应的 bas 文件）

2. 控制程序的下载

使用 ZAux_BasDown 指令将程序下载进去即可。(**注意：**xxx.bas 文件路径不能填错，文件路径也不适宜过长。(1000 个字节以内))

12.2.3 例程

12.2.3-1 控制程序的下载及编写

本例程主要演示了如何使用 PC 库函数将写好的 xxx.bas 程序下载到控制器当中去，首先编写控制程序，然后将编写好的程序保存到指定的路径，再使用 ZAux_BasDown 下载进去即可。

注意：文件路径一定要用“\\”去表示，不能用“/”或“\”。

例如：.bas 文件路径为“F:\111\Basic1.bas”，在 C 语言中表示路径中的“\”就要用“\\”去代替，即：“F:\\111\\Basic1.bas”。

例：本例程的 bas 文件放在当前路径下，则 C 语言中表示“\”就要用“.\\ECAT 初始化.bas”

本例程所用到的 bas 文件是一个 EtherCAT 总线轴的初始化程序，本例程的目的为：使用该 Bas 文件，以及 ZMC432 控制器，对连接该控制器的总线轴进行初始化配置。

本例程涉及到功能有：把“ECAT 初始化.bas”文件下载到控制器里面，并且对该 Bas 文件的全局变量 PUL_AxisStart（本地脉冲轴起始轴号）；PUL_AxisNum（本地脉冲轴数量）；Bus_AxisStart（总线轴起始轴号）三个变量，进行配置。

```
'*********************************************************ECAT总线初始化
global  BUS_TYPE            '总线类型。可用于上位机区分当前总线类型
global  Bus_Slot            '槽位号0（单总线控制器缺省0）
global  PUL_AxisStart       '本地脉冲轴起始轴号
global  PUL_AxisNum         '本地脉冲轴轴数量
global  Bus_AxisStart       '总线轴起始轴号
global  Bus_NodeNum         '总线配置节点数量,用于判断实际检测到的从站数量是否一致

BUS_TYPE        = 0     '总线类型。可用于上位机区分当前总线类型
Bus_Slot        = 0     '槽位号0（单总线控制器缺省0）
PUL_AxisStart   = 0     '本地脉冲轴起始轴号
PUL_AxisNum     = 0     '本地脉冲轴轴数量
Bus_AxisStart   = 0     '总线轴起始轴号
Bus_NodeNum     = 2     '总线配置节点数量,用于判断实际检测到的从站数量是否一致
```




```
// test1.cpp : 定义控制台应用程序的入口点。
//


#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"


void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非0则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
    }
}


int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"192.168.0.11";                //控制器IP地址
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
```

```
    printf("控制器连接成功！\n");

    const char * filename= ".\\ECAT 初始化.bas";

    //下载程序到 ROM 中  0-RAM 1-ROM
    ret = ZAux_BasDown(handle,filename,1);//把 ECAT 初始化 bas 文件下载到控制器里面进行初始化，该程序下载后，经过 7S 延时（等待驱动器上电预留时间）进行 Ethercat 总线初始化
    commandCheckHandler("ZAux_BasDown",ret);
    Sleep(500);//延时 500ms 等等程序下载到控制器
    //PUL_AxisStart，PUL_AxisNum，Bus_AxisStart 为 ECAT 初始化.bas 文件定义的三个全局变量

    float GetUserVar;
    ret = ZAux_Direct_GetUserVar(handle,"PUL_AxisStart",&GetUserVar);//获取修改前控制器本地脉冲轴起始轴号
    commandCheckHandler("ZAux_BasDown",ret);
    printf("设置前 PUL_AxisStart=%f\n",GetUserVar);
    ret = ZAux_Direct_GetUserVar(handle,"PUL_AxisNum",&GetUserVar);//获取修改前控制器本地脉冲轴总数
    commandCheckHandler("ZAux_BasDown",ret);
    printf("设置前 PUL_AxisNum=%f\n",GetUserVar);
    ret = ZAux_Direct_GetUserVar(handle,"Bus_AxisStart",&GetUserVar);//获取修改前控制器总线轴起始轴号
    commandCheckHandler("ZAux_BasDown",ret);
    printf("设置前 Bus_AxisStart=%f\n",GetUserVar);

    //因为本例程此次使用的是 ZMC432,该控制器本体有 6 个本地脉冲轴，所以把总线轴号设置为 6，
    //把 ECAT 初始化.bas 里面定义的变量做如下修改：
    //控制器本地脉冲轴起始轴号变量：PUL_AxisStart 修改为 0，默认为 0
    //控制器本地脉冲轴总数：PUL_AxisNum 修改为 6，默认为 0
    //控制器总线轴起始轴号：Bus_AxisStart 修改为 6，默认为 0
    ret = ZAux_Direct_SetUserVar(handle,"PUL_AxisStart",0);//设置控制器本地脉冲轴起始轴号
    commandCheckHandler("ZAux_BasDown",ret);
    ret = ZAux_Direct_SetUserVar(handle,"PUL_AxisNum",6);//设置控制器本地脉冲轴总数
    commandCheckHandler("ZAux_BasDown",ret);
```

```
    ret = ZAux_Direct_SetUserVar(handle,"Bus_AxisStart",6);//设置控制器总线轴起始轴号
    commandCheckHandler("ZAux_BasDown",ret);

    ret = ZAux_Direct_GetUserVar(handle,"PUL_AxisStart",&GetUserVar);//获取修改后控制器本地脉冲轴起始轴号
    commandCheckHandler("ZAux_BasDown",ret);
    printf("设置后 PUL_AxisStart=%f\n",GetUserVar);
    ret = ZAux_Direct_GetUserVar(handle,"PUL_AxisNum",&GetUserVar);//获取修改后控制器本地脉冲轴总数
    commandCheckHandler("ZAux_BasDown",ret);
    printf("设置后 PUL_AxisNum=%f\n",GetUserVar);
    ret = ZAux_Direct_GetUserVar(handle,"Bus_AxisStart",&GetUserVar);//获取修改后控制器总线轴起始轴号
    commandCheckHandler("ZAux_BasDown",ret);
    printf("设置后 Bus_AxisStart=%f\n",GetUserVar);

    Sleep(2000);
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

12.3 控制器自动上报

12.3.1 指令列表

指令	说明
(*PZMCAutoUpCallBack)	主动上报的回调函数格式
ZAux_SetAutoUpCallBack	控制器自动上报
ZAux_CycleUpEnable	打开使能周期上报
ZAux_CycleUpDisable	关闭使能周期上报
ZAux_CycleUpGetRecvTimes	周期上报收到的包次数，超过溢出，调试使用

ZAux_CycleUpForceOnce	强制上报一次，在运动指令后 idle 可能不准确的情况下调用
ZAux_CycleUpReadBuff	从周期上报里面读取内容（浮点型）
ZAux_CycleUpReadBuffInt	从周期上报里面读取内容（整型）

12.3.2 重点说明

主动上报：

1. 控制器主动上报功能。

2. 主动上报消息：控制器通过 SEND_AUTOUP 发消息给 PC 端。

3. 3.1.6.68 以后的 DLL 支持，如下图所示。



4. 查看哪些 PORT 支持主动上报可以通过使用 RTSys 在线命令?*IFAUTOUP_PORT 查看，设置为-1 为自动选择可上报的 PORT 上报。

周期上报：

1. 需要 zmotion.dll 与控制器固件同时支持。

2. 通过提前设置经常读取的参数主动周期上报，可以减少 PC 主动轮询的时间。

12.3.3 例程

12.3.3-1 主动上报

BASIC 程序：

SEND_AUTOUP(-1,2022,"Hello PC")

先将 PC 端的程序运行(自动会打开要上报的端口)，然后再通过 RTSys 执行 SEND_AUTOUP(-1,2022,"Hello PC")，每执行一次就上报一次。

流程图：




```
// test1.cpp : 定义控制台应用程序的入口点。
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
//该指令检查指令的返回值，如果返回值不为 0，则向屏幕打印返回值
void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
    }
}
//定义主动上报回调函数
void AutoUpCallBack(ZMC_HANDLE handle, int32 itypecode, int32 idatalength, uint8
*pdata)
{
    printf("上传数据码为：%d、上传数据长度为：%d、上报内容为：%s\n",
itypecode,idatalength,(char*)pdata);
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";                //控制器 IP 地址
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
```

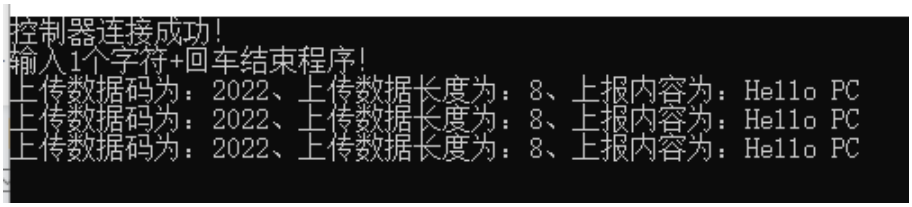
```
        return -1;
    }
    printf("控制器连接成功！\n");
    ret = ZAux_SetAutoUpCallBack( handle, AutoUpCallBack);
    commandCheckHandler("ZAux_SetAutoUpCallBack", ret);

    printf("输入 1 个字符+回车结束程序！\n");
    char a[128];
    scanf("%s",&a);

    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

示例图如下：

PC 端：

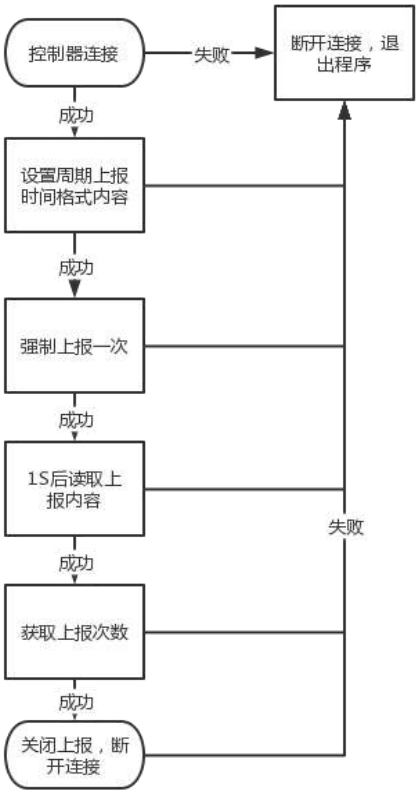


控制器端：



12.3.3-2 周期上报

周期上报，通过提前设置经常读取的参数主动周期上报，可以减少 PC 主动轮询的时间。具体上报流程如下图所示。




```
// test1.cpp : 定义控制台应用程序的入口点。
//
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
#include  <time.h>
//该指令检查指令的返回值，如果返回值不为 0，则向屏幕打印返回值
void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
        Sleep(2000);
        exit(0);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
```

```
    char *ip_addr = (char *)"127.0.0.1";                //控制器 IP 地址

    ZMC_HANDLE handle = NULL;                           //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("控制器连接成功! \n");

    ret = ZAux_CycleUpEnable(handle, 0, 700, "idle, mpos(0,3), dpos(1)");//周期上报
轴 0 的运动状态，轴 1 的 DPOS，轴 0-2 的 MPOS
    commandCheckHandler("ZAux_CycleUpEnable", ret) ;//判断指令是否执行成功

    ret = ZAux_CycleUpForceOnce(handle, 0);//强制上报一次，0 为通道号
    commandCheckHandler("ZAux_CycleUpForceOnce", ret) ;//判断指令是否执行成功

    double dtemp;//浮点型读取周期上报内容
    int32 itemp,addrnum;//整型读取周期上报内容
    addrnum=0;

    Sleep(1000);
    ret = ZAux_CycleUpReadBuff(handle, 0, "mpos", 0, &dtemp);//浮点型读取 MPOS（0）
    commandCheckHandler("ZAux_CycleUpReadBuff", ret) ;//判断指令是否执行成功

    ret = ZAux_CycleUpReadBuffInt(handle, 0, "dpos", 1, &itemp);//整型读取 DPOS（1）
    commandCheckHandler("ZAux_CycleUpReadBuff", ret) ;//判断指令是否执行成功
    printf("itemp=%d,mpos=%f\n",itemp,dtemp);

    addrnum = ZAux_CycleUpGetRecvTimes(handle, 0);//获取上报次数
    printf("上报次数=%d\n",addrnum);

    Sleep(2000);
```

```
    ret = ZAux_CycleUpDisable(handle, 0);//关闭周期上报
    commandCheckHandler("ZAux_CycleUpDisable", ret) ;//判断指令是否执行成功

    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

12.4 调试相关

12.4.1 指令列表

指令	说明
ZAux_SetTraceFile	命令跟踪设置
ZAux_Trigger	示波器触发函数
ZAux_Scope	示波器采样
ZAux_GetScopeLength	获取示波器采样长度

12.4.2 重点说明

12.4.2-1 调试打印信息

ZAux_SetTraceFile 指令需要结合 ZAux_Execute 指令结合使用，使用调试信息之后，会默认输出保存 ZAux_Execute 指令所发出的信息以及返回的内容，保存在当前程序路径下的\ZmotionLog\下，以 log 日志的方式保存。

打印输出模式：

0：表示关闭

1：表示输出错误命令

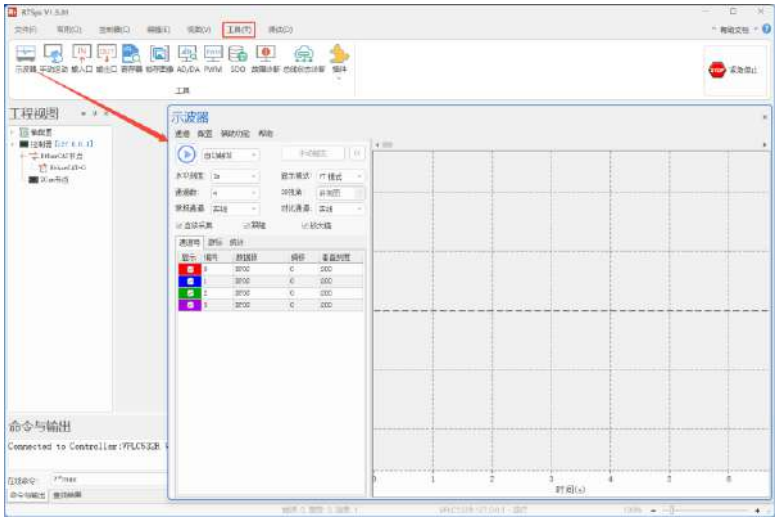
2：表示只输出运动与参数设置命令

3：输出全部的命令

注意：该指令只在调试的时候使用，不使用的时候，切记一定要关闭打印输出模式，否则会影响通讯效率。

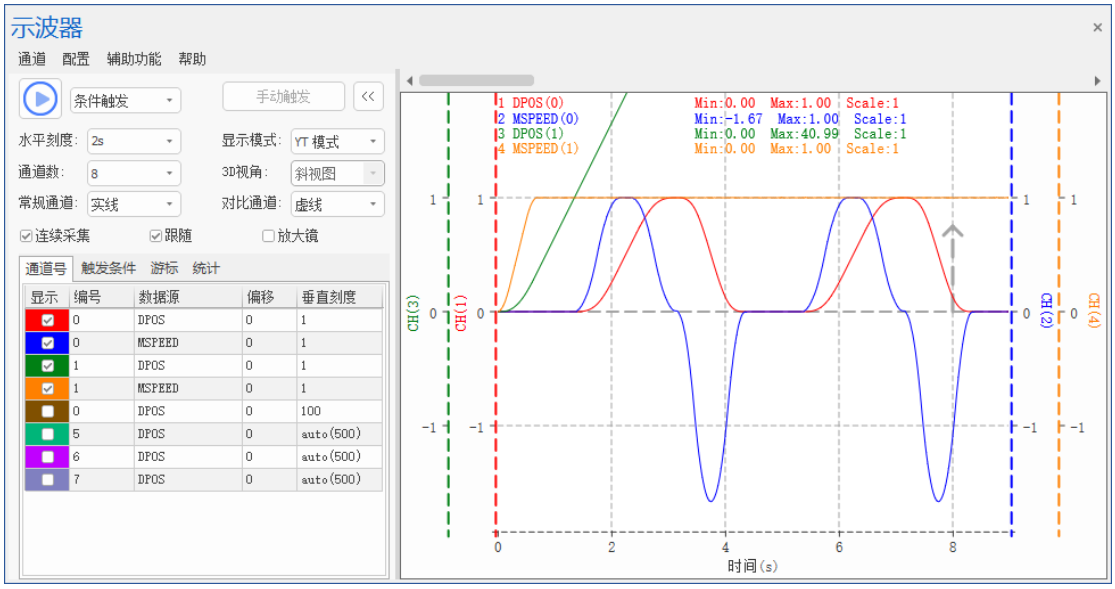
12.4.2-2 示波器

示波器属于程序调试与运行中极其重要的一个部分，用于把肉眼看不到的信号转换成图形，便于研究各种信号变化过程。示波器利用控制器内部处理的数据，把数据显示成图形，利用示波器可以显示各种不同的信号，如轴参数、轴状态等。在“工具”→“示波器”中，打开示波器窗口。



示波器必须先启动后触发才能成功采样。打开示波器，设置好相关参数之后点击“启动”，可手动触发采样，也可在程序里加入“ZAux_Trigger”指令自动触发示波器采样。

RTSys 支持 4 种采集波形模式，YT 模式（数值时间模式），XY 模式（平面模式），XYZ（三维模式），XYZD（四通道模式）。



示波器基础设置按钮功能：

按钮	功能
通道	选择的通道及叠加通道，对比通道不显示。

配置		选择进行示波器参数配置、观察器配置、数据源设计及配置的导入导出。
辅助功能		辅助观察波形，包括搜索波形、对比波形及导入导出波形。
帮助		显示鼠标操作指南界面，提示各个模式下鼠标的快捷操作内容。
		示波器启动开关。启动状态下显示为  ，但不触发示波器采样。
触发模式	自动触发	点击示波器“  ”启动按钮后，立即触发采集；
	手动触发	与“手动触发”按钮配合使用。点击“  ”启动后，点击“手动触发”按钮后下载至 RAM/ROM，或点击“  ”后直接下载至 RAM/ROM 后等待 Basic 程序触发“TRIGGER”指令，才可成功采样；
	条件触发	满足条件才触发示波器采样。须在采集开始前配置触发条件、预触发函数； 点击“  ”启动示波器后无法改变触发条件，下载程序等待条件满足则触发采集；（条件触发详细介绍参考下文）
手动触发		手动触发示波器采样按钮。
<<		按下隐藏通道名称和峰值，只显示通道编号。
水平刻度		横轴一格的刻度。下拉菜单选择，可手动输入数值和单位，默认输入单位为 ms，输入完成后自动转化为 s。将鼠标放置数值框，滚动鼠标可以缩放水平刻度。YT 模式有效，XYZ 模式和 XYZD 模式下变为灵敏度，表示鼠标左键操作的灵敏度。
显示模式	YT 模式	不同数据源随时间变化的曲线，每一个通道显示一条波形。
	XY 模式	XY 平面显示两个轴的插补合成轨迹，连续两个同类型通道为一组显示一条波形。 注：通道数设置小于 2 时不可使用该模式。
	XYZ 模式	XYZ 三维空间显示合成轨迹。依次选择通道作为 X、Y、Z 轴，三个同类型通道为一组显示一条波形(通道类型包括：常规通道、叠加通道、对比常规通道和对比叠加通道)，每种类型最多显示一条波形。 注：使用本模式时显卡的 OpenGL 版本须为 1.5 及以上。 通道数设置小于 3 时不可使用该模式。
	XYZD 模式	XYZD 四通道可视化显示轨迹，其中 XYZ 为三维空间合成轨迹显示，D 为以圆点形式显示的数据源。计算方式为：圆点直径大小=当前 D 数值÷D 基准数值×D 基准大小。参数修改位于“观察器配置”窗口。依次选择通道作为 X、Y、Z 轴和 D 数值取值通道，四个同类型通道为一组显示一条波形(通道类型包括：常规通道、叠加通道、对比常规通道和对比叠加通道)，每种类型最多显示一条波形。 当前 D 数值：当前位置的数据源值的大小。 注：使用本模式时显卡的 OpenGL 版本须为 1.5 及以上。 通道数设置小于 4 时不可使用该模式。

通道数		设置要采样的常规通道总数，启动状态下不可修改。当设置通道数大于控制器支持通道数时，会弹出提示信息：超过控制器支持的最大通道数。
3D 视角		可选斜视角、正视角、左视角和俯视角，默认斜视角。XYZ 模式和 XYZD 模式有效。
常规通道		常规通道的显示线形：点、实线、虚线。线形大小、颜色等属性可在“观察器配置”中设置。
对比通道		对比通道的显示线形：点、实线、虚线。线形大小、颜色等属性可在“观察器配置”中设置。
连续采集		不开启连续采集时，到达采样最大采集周期数后便停止采样；开启连续采集之后示波器会持续采样，到达采样最大采集周期数后仍继续采样，忽略设置的采样最大采集周期数，直到按下停止按钮后才会停止采样，采集的数据自动重新覆盖之前的数据。连续采集的所有波形采样数据均能导出。（使用串口时自动取消连续采集功能）
跟随		开启跟随后，横轴自动移动到实时采样处，跟随波形显示。
放大镜		勾选放大镜，鼠标移动到显示区域后自动在鼠标的右下方显示放大视图，放大视图跟随鼠标移动及刷新。放大镜参数修改位于“观察器配置”窗口。YT 模式有效。
通道号	显示	选择当前通道曲线是否显示。示波器有四种类型的通道，包含：常规通道 1~8，叠加通道 1~4，对比波形的常规通道 1~8 和对比波形的叠加通道 1~4。
	编号	选择需要采集的数据源编号，如：轴号、数字量 IO 编号，模拟量 IO 编号、TABLE 编号、VR 编号、MODBUS 编号等。编号设置范围为 0 到控制器的最大轴数，可手动输入编号。
	数据源	选择采集的数据源类型。点击鼠标左键可手动输入数据类型，或鼠标左键单击选择类型参数。可以在“数据源设计”窗口设置需要的参数类型。
	偏移	波形纵轴偏移量设置，下拉菜单选择偏移量，可手动输入。
	垂直刻度	纵轴一格的刻度。选择 auto 时表示自动刻度，示波器停止时可用，刻度值根据当前采集到的波形运动幅度自动变化，使波形能够完整的显示在当前示波器界面上。
↑		提示此处可能会出现丢点现象，与最大采集周期数有关。示波器开启连续采集后，在最大采集周期的 80%处会重新触发采集，此时 TABLE 数据开始重新覆写，在此过程中可能会出现丢点现象。手动触发模式下使用“TRIGGER”指令有效，在最大采集周期数的 80%处左右出现。
注意：若要设置示波器参数，如轴编号、数据源以及示波器“参数配置”窗口，要先停止示波器再设置。		

若要设置示波器参数，如轴编号、数据源以及启动示波器设置窗口，要先停止示波器再设置。除了窗口上方进度条可以控制曲线的显示，鼠标按住左键或右键也能拖动曲线。



示波器参数配置窗口：

参数	描述
基本参数	
采样周期(us)	示波器两次采样之间的时间间隔。不可修改。
间隔周期数	采样时间间隔，单位为系统周期，与控制器固件版本有关，一般默认 1ms，指令 SERVO_PERIOD 查看。（例：间隔周期数设置为 1，则表示 1 个周期采样一次；间隔周期数设置为 5，则表示 5 个周期采样一次；周期时间取决于控制器固件版本）一般来说，间隔周期数越小，采样数据越准确，单位时间内数据量越大。
最大采集周期数	总共采样的数据次数，数值越大采样范围越大。（即：一个通道采集的数据所需使用的 TABLE 大小）
自动使用 TABLE 数组末尾	抓取数据存放的位置，默认为 True。
TABLE 位置	设置抓取数据存放的位置。一般默认自动使用 TABLE 数据末尾空间，当“自动使用 TABLE 数组末尾”设置为 False 时可以自定义设置，但是设置时注意不要与程序使用的 TABLE 数据区域重合。 查看控制器 TABLE 空间大小有以下三种方法： a.使用 TSIZE 指令读取；

	b.在“控制器状态”窗口查看； c.在线命令?*max 打印查看。
叠加参数	
通道数	选择叠加的通道数量，下拉菜单选择。叠加通道最多支持 8 个。 注意：叠加的子通道必须为同一类型，叠加通道才有效，不能子通道 1 为常规数据源，子通道 2 为驱动器数据源。
叠加的第一/二个通道	用户选择进行叠加的通道号。
叠加方式	设置两条通道之间的叠加方式，下拉菜单选择相加或相减两种叠加方式。通道叠加后会生成新的叠加波形。（例：在通道号界面显示 CH:1-2 表示第一个通道和第二个通道以相减的形式进行叠加）
统计参数	
统计参数	设置示波器统计页面显示的参数信息，默认为 True。
超采样	
超采样长度	设置每个周期采样次数。

示波器数据导入导出：

导入：示波器必须在停止状态下才能导入数据，导入成功能将采样波形复现出来。

导入采样数据方法：点击“导入”，选择导入的数据文件为之前从示波器导出的文件类型后打开即可。

导出：导出参数包括示波器参数设置情况，以及各个通道的数据类型和每个采样点数据。如右图所示。

导出采样数据方法：先在设置里勾选“导出参数”，启动示波器采样，采样完成后点击“导出”，选择文件夹保存示波器数据，导出数据为文本文件。

示波器采样方法：

打开工程项目，连接控制器或仿真器，再打开示波器窗口（操作示波器窗口之前需要连接到控制器或仿真器才可以操作）。

在示波器窗口点击“设置”，选择采样通道数、采样深度、采样间隔、采样数据 TABLE 存储位置（一般来说自动使用 TABLE 数组末尾空间即可）和采样类型等，设置完成确认保存当前设置。

再选择采样数据编号和数据源，点击“启动”按钮。

将程序下载到控制器运行，程序里需要包含 **ZAux_Trigger** 自动触发示波器采样指令，此时示波器开始采样，显示出不同数据源的波形。可调整显示刻度和波形偏移，便于观察不同波形。

若波形精度不高或显示不完整，可点击“停止”按钮后再打开“设置”，调整好采样间隔和采样深度后重新执行上述采样过程。

若需要采样的时间较长，开启“连续采集”功能，此时采样时间与深度无关。

ZAux_Trigger 指令可以非常简洁与灵活的辅助波形查看与问题查找，示波器功能对查找问题非常有用，可以针对性的使用好该指令，进一步提高调试效率。

示波器使用注意事项：

示波器采样时间计算：

例如深度：10000，间隔：5

如果系统周期 SERVO_PERIOD=1000，也就是 1ms 轨迹规划周期，间隔 5 表示每 5ms 采集一个数据点，一共采集 10000 次数据，采集时间长度为 50s。

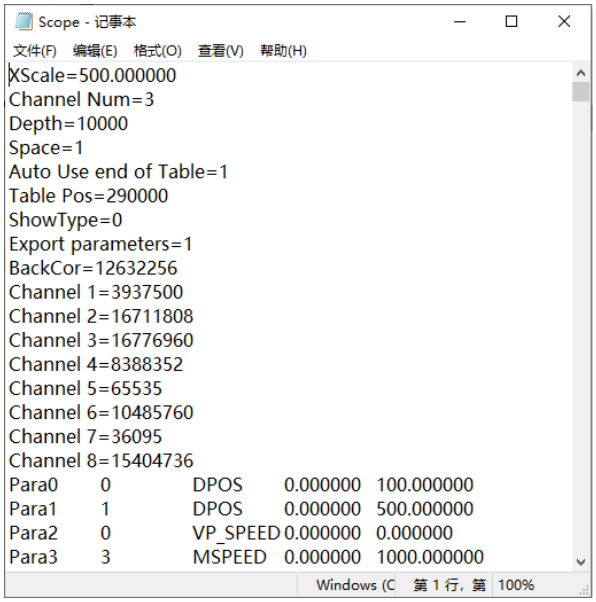
TABLE 数据末尾存储空间计算：

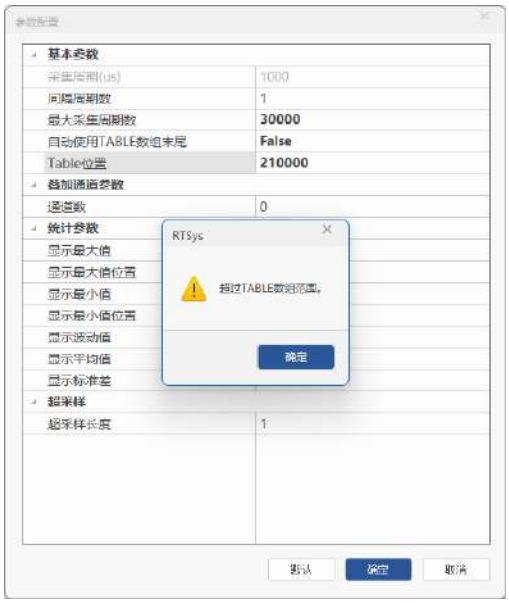
设置抓取数据存放的位置，一般默认自动使用 TABLE 数据末尾空间，此时根据采样数据占用空间大小自动计算起始空间地址。

计算方法：采样数据占用空间大小=通道数*深度

例：若控制器的 TABLE 空间大小为 320000，采样 4 个通道，深度为 30000，每个采样点占用一个 TABLE，所以会占用 4*30000＝120000 个 TABLE 位置，320000-120000=200000，此时 TABLE 的起始位置为 200000。

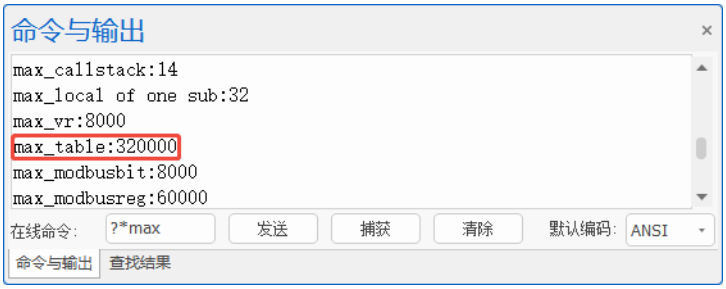
数据存放的位置也可以自定义配置，若按上面的通道数和深度，起始 TABLE 空间自定义时不能超过 200000，否则无法设置，如下图。





示波器采样数据占用的空间不要与程序使用的 TABLE 数据区域重合。

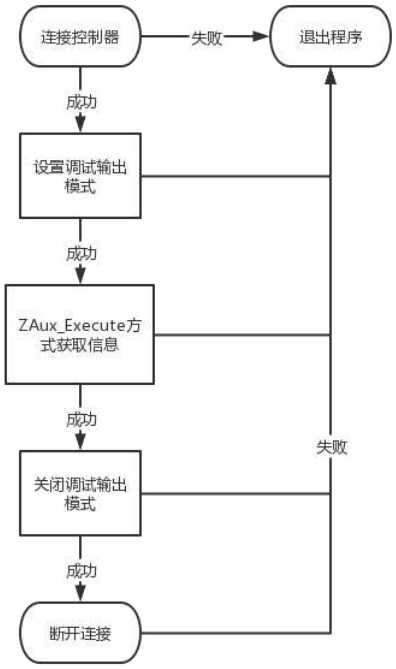
控制器 TABLE 空间大小可使用 TSIZE 指令读取、在“控制器状态”窗口查看或在线命令?*max 打印查看。



12.4.3 例程

12.4.3-1 调试打印信息

程序例程流程图：




```
// test1.cpp : 定义控制台应用程序的入口点。
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
#include  <time.h>
//该指令检查指令的返回值，如果返回值不为 0，则向屏幕打印返回值
void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
        Sleep(2000);
        exit(0);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    //char *ip_addr = (char *)"192.168.1.11";                //控制器 IP 地址
    char *ip_addr = (char *)"127.0.0.1";                //控制器 IP 地址
```

```
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("控制器连接成功! \n");

    //0 关闭  1-只输出错误命令  2-只输出运动与设置命令  3 输出全部命令
    int bifTofile = 3;     //设置调试打印输出模式为 3

    ret = ZAux_SetTraceFile(bifTofile, "预留");//设置打印输出模式
    commandCheckHandler("ZAux_SetTraceFile",ret);

    const char * command = "?control"; //发送获取控制器型号的命令

    const char * command1 = "?*max";    //发送控制器信息的命令

    char psRespons[1024];               //用于接受控制器型号的数组
    int length = 1024;                  //控制器型号的长度，一定要大于接受的字符串长
度

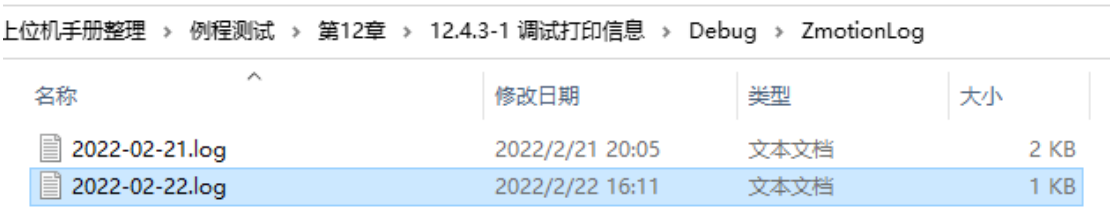
    ret = ZAux_Execute(handle,command, psRespons,length);//获取控制器型号
    commandCheckHandler("ZAux_Execute", ret);

    ret = ZAux_Execute(handle,command1, psRespons,length);//获取控制器型号
    commandCheckHandler("ZAux_Execute", ret);

    Sleep(2000);
    ret = ZAux_SetTraceFile(0, "预留");//关闭打印输出模式，调试完成之后必须关闭打印
输出模式
    commandCheckHandler("ZAux_Execute", ret);
    ret = ZAux_Close(handle);     //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
```

```
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

输出结果：





12.4.3-2 示波器的使用

本例程演示了，通过轴 0，轴 1，在平面点（0，0）位置走到平面点（100，100）位置，通过示波器抓取运动轨迹图以及速度曲线图。

上位机程序：

```
// test1.cpp : 定义控制台应用程序的入口点。
//


#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
```

```
#include "zauxdll2.h"
#include <time.h>
//该指令检查指令的返回值，如果返回值不为 0，则向屏幕打印返回值
void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
        Sleep(2000);
        exit(0);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    //char *ip_addr = (char *)"192.168.1.11";                //控制器 IP 地址
    char *ip_addr = (char *)"127.0.0.1";                //控制器 IP 地址
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("控制器连接成功! \n");

    ret =ZAux_Direct_SetAtype(handle,0,1);//设置轴 0 轴类型为 1
    commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//判断指令是否执行成功

    ret =ZAux_Direct_SetAtype(handle,1,1);//设置轴 1 轴类型为 1
    commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//判断指令是否执行成功
    commandCheckHandler("ZAux_Execute", ret);

    ret = ZAux_Direct_SetDpos(handle, 0, 0);         //设置轴 0 的 dpos
    commandCheckHandler("ZAux_Direct_SetDpos", ret);
```

```
    ret = ZAux_Direct_SetDpos(handle, 1, 0);         //设置轴 1 的 dpos
    commandCheckHandler("ZAux_Direct_SetDpos", ret);

    int axislist[2] = { 0,1 };                    //运动轴列表，其中轴 0 为主轴
    float poslist[2] = { 100,100};                //运动列表，轴 0-100，轴 1-100
    ZAux_Trigger( handle);//触发示波器

    ret = ZAux_Direct_SetSpeed(handle, axislist[0], 100);    //设置插补速度 100，设置
在主轴上
    commandCheckHandler("ZAux_Direct_SetSpeed", ret);

    ret = ZAux_Direct_SetAccel(handle, axislist[0], 500);    //设置插补加速度 500，设
置在主轴上
    commandCheckHandler("ZAux_Direct_SetAccel", ret);

    ret = ZAux_Direct_SetDecel(handle, axislist[0], 500);    //设置插补减速度 500，设
置在主轴上
    commandCheckHandler("ZAux_Direct_SetDecel", ret);

    ret = ZAux_Direct_MoveAbs(handle, 2, axislist, poslist); //调用运动，轴 0,1 走到
对应的 100,100 绝对位置
    commandCheckHandler("ZAux_Direct_MoveAbs", ret);

    int state[2];

    do
    {
        ZAux_Direct_GetIfIdle(handle, axislist[0], state);
        ZAux_Direct_GetIfIdle(handle, axislist[1], state + 1);

        if (!(state[0] || state[1]))
        {
            Sleep(100);
        }
        else
        {
            break;
```

```
        }
    } while (true);//等待运动停止


    Sleep(2000);
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

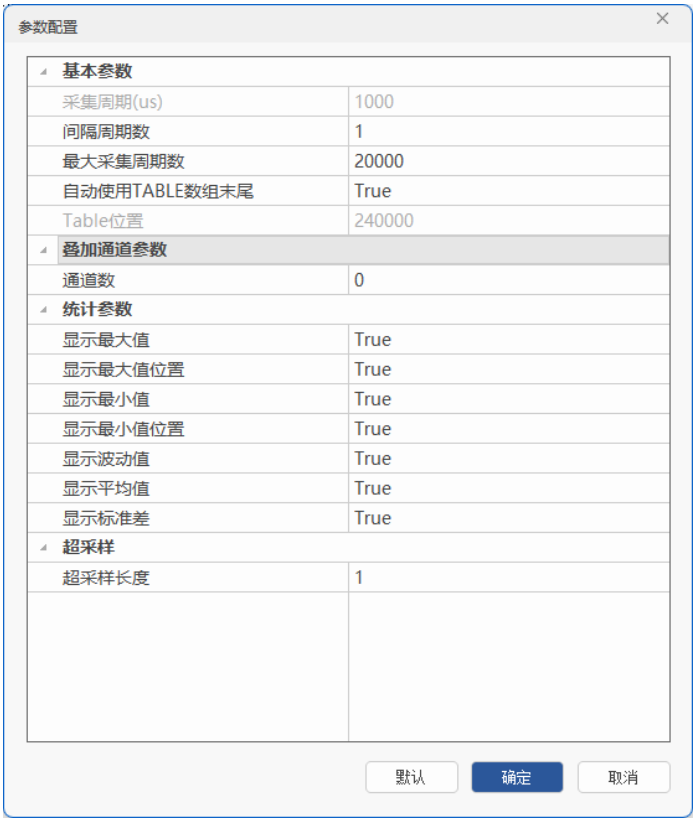
示波器打开：

步骤 1：打开 RTSys 软件。

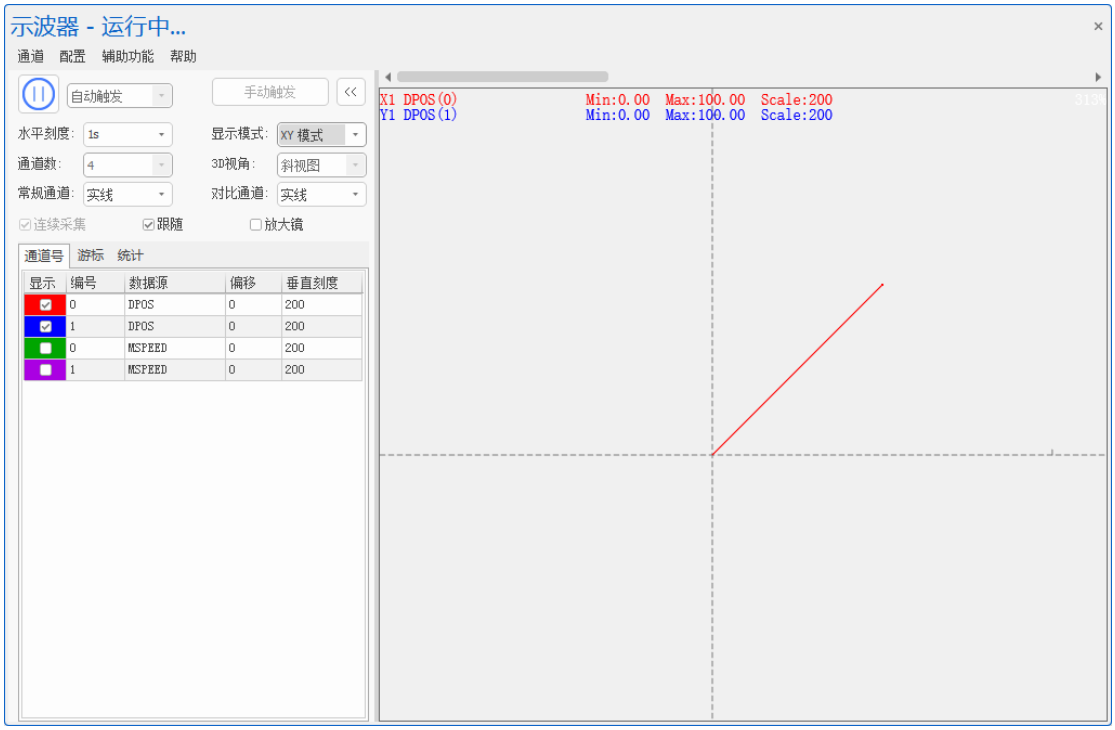
步骤 2：连接所使用的控制器。（菜单栏控制器—>连接—>连接控制器)。

步骤 3：（工具—>示波器）打开示波器。

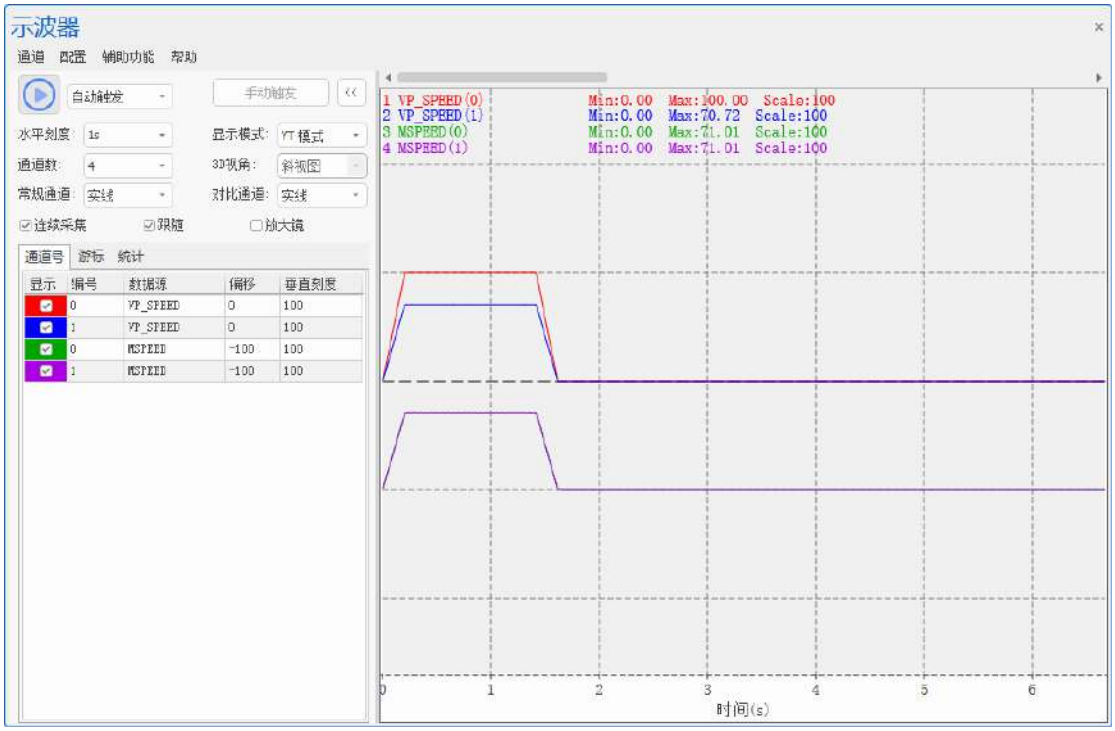
步骤 4：示波器设置。（按需设置示波器参数。设置参数详情请查看 12.4.2 章节，本例程使用参数设置如下图）



步骤 5：抓取运动轨迹图（编号，数据源，偏移，垂直刻度，水平刻度如下图，参数填完之后选择 XY 模式，选择自动触发，点击启动按钮，运行上位机程序）



步骤 6：抓取速度变化曲线图（编号，数据源，偏移，垂直刻度，水平刻度如下图，参数填完之后，显示模式选择 XT 模式，选择自动触发，点击启动按钮，运行上位机程序）



12.4.3-3 数据采样

本例子展示了数据采样，控制轴 0 运动了 10000 个单位，采样了每 10ms 采样一次，共采样 1000 次，轴 0 的命令规划位置以及命令速度，并打印采样信息。

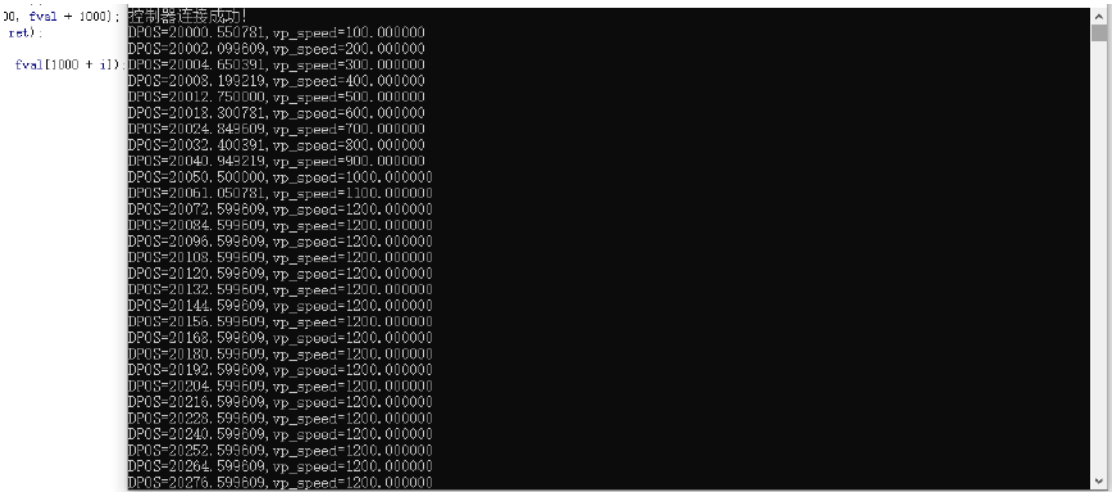
程序逻辑框图：



```
// test1.cpp : 定义控制台应用程序的入口点。
//
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
#include <time.h>
// 全局变量，存储控制器连接句柄
ZMC_HANDLE handle = NULL;
// 检查指令返回值，非零则打印错误信息并退出程序
// command: 指令名称
// ret: 指令返回码
void commandCheckHandler(const char *command, int ret) {
    if (ret) {
        printf("%s fail!return code is %d\n", command, ret);
        Sleep(5000);
        exit(0);
    }
}
int _tmain(int argc, _TCHAR* argv[]) {
    // 控制器IP地址，可按需修改
    char *ip_addr = (char *)"127.0.0.1";
    // 连接控制器
    int ret = ZAux_OpenEth(ip_addr, &handle);
    if (ERR_SUCCESS != ret) {
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("控制器连接成功! \n");
    // 配置示波器采样，采集轴0的DPOS和VP_SPEED
    ret = ZAux_Scope(handle, 1, 10, 1000, 3000, "DPOS(0),VP_SPEED(0)");
    commandCheckHandler("ZAux_Scope", ret);
    // 触发采样
    char ackbuff[2048];
    ret = ZAux_Execute(handle, "TRIGGER", ackbuff, 2048);
```

```
    commandCheckHandler("ZAux_Execute", ret);
    // 控制轴0单轴运动
    ret = ZAux_Direct_Single_Move(handle, 0, 10000);
    commandCheckHandler("ZAux_Direct_Single_Move", ret);
    // 等待采样完成
    int len = 0;
    float fval[2000] = {};
    Sleep(10000);
    // 获取采样数据长度
    ret = ZAux_GetScopeLength(handle, &len);
    commandCheckHandler("ZAux_GetScopeLength", ret);
    // 读取并打印采样数据
    if (len > 0) {
        ret = ZAux_Direct_GetTable(handle, 1000, 1000, fval);
        commandCheckHandler("ZAux_Direct_GetTable", ret);
        ret = ZAux_Direct_GetTable(handle, 2000, 1000, fval + 1000);
        commandCheckHandler("ZAux_Direct_GetTable", ret);
        for (int i = 0; i < len; i++) {
            printf("DPOS=%f,vp_speed=%f\n", fval[i], fval[1000 + i]);
        }
    }
    // 再次等待
    Sleep(10000);
    // 关闭连接
    ret = ZAux_Close(handle);
    commandCheckHandler("ZAux_Close", ret);
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

输出结果：

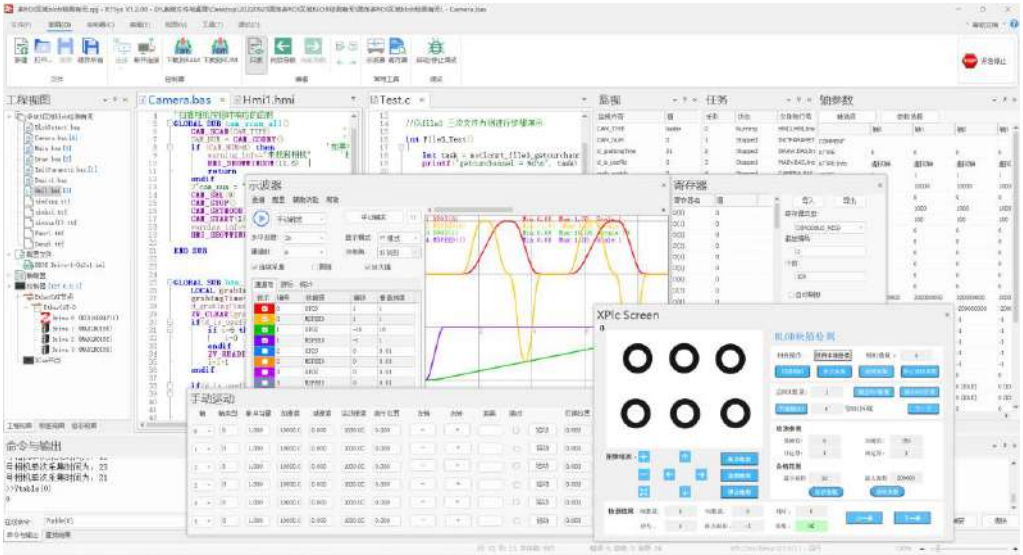


12.5 RTSys 辅助调试

12.5.1 RTSys 简介

RTSys 是正运动推出的集成运动控制+机器视觉功能的开发软件，支持 RTBasic、RTPlc 梯形图、RTHmi、C 语言、RTVision 机器视觉等二次开发，并可混合编程、实时仿真、在线跟踪以及诊断与调试，快速实现智能装备的视觉定位、测量、识别、检测和复杂的运动控制等系统的开发，RTSys 软件开发界面如下图所示。

用户可通过串口或网口连接 PC 与控制器，使用 RTSys 软件编写的程序可以直接下载到正运动控制器里脱机运行，也可以在 PC 平台仿真运行。



RTSys 软件支持四种编程方式：Basic、PLC 梯形图、HMI 组态、C 语言。使用 RTSys 进行 Basic 语言编程开发的优势在于：支持多个 Basic 任务运行；若使用 PLC 梯形图开发则支持一个 PLC 任务运行;若使用 HMI 组态编程开发则支持两个 HMI 任务运行。Basic 任务、PLC 梯形图任务和 HMI 组态任务、C 语言任务之间可以多任务运行。

RTSys 软件支持在线仿真调试，自带仿真器 ZMC Simulator 和组态程序仿真工具 xplc screen。

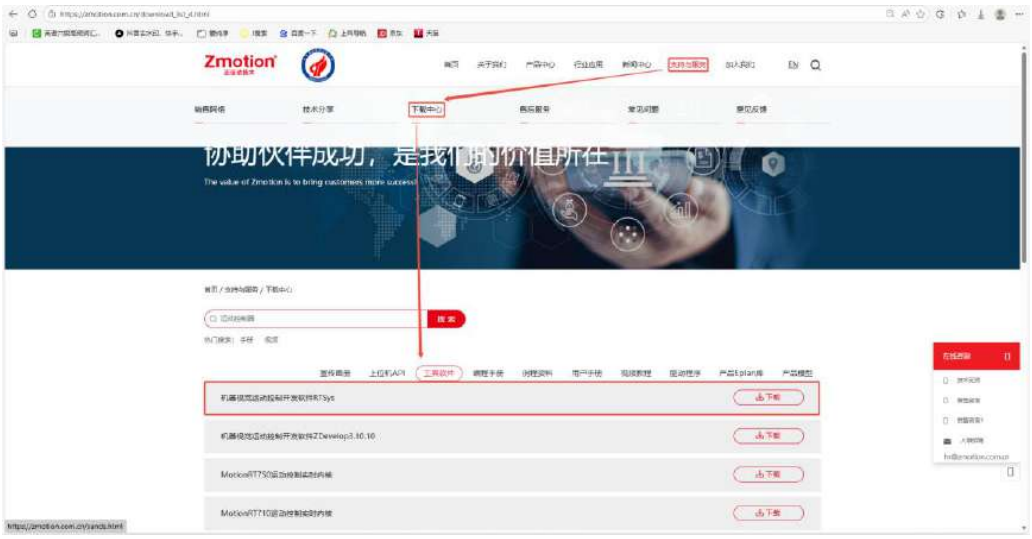
在 RTSys 创建的项目不建议在 Zdevelop 中打开！

使用 RTSys 软件 HMI 功能需要控制器固件支持才可使用！

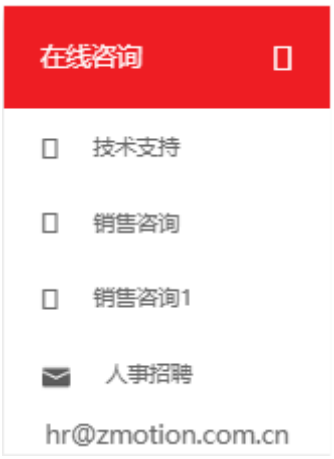
12.5.2 RTSys 如何获取

方法 1：通过正运动官网获取.

正运动官网资料下载：支持与服务 -> 下载中心 -> 工具软件；（http://www.zmotion.com.cn/download_list_4.html）



方法 2：咨询正运动技术人员获取。

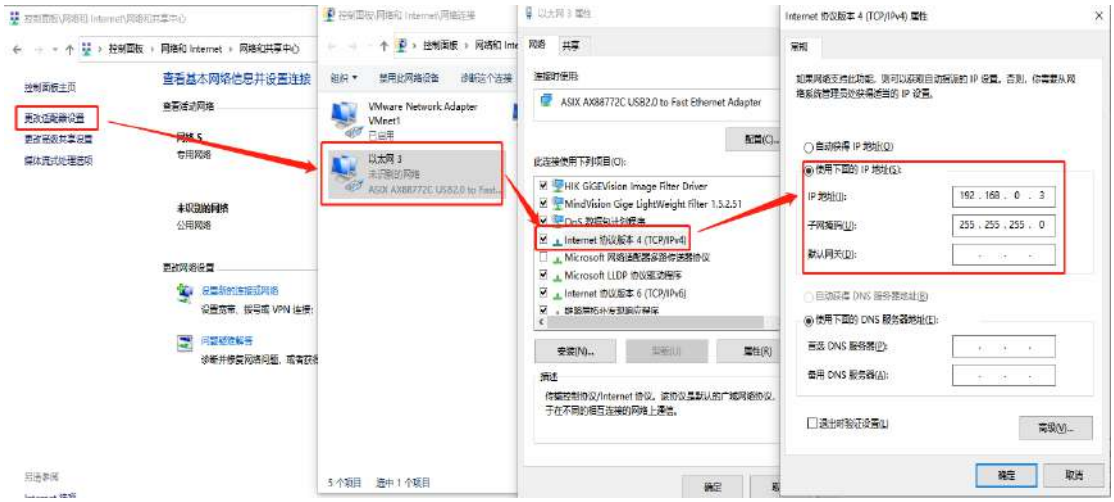


12.5.3 RTSys 如何配合调试

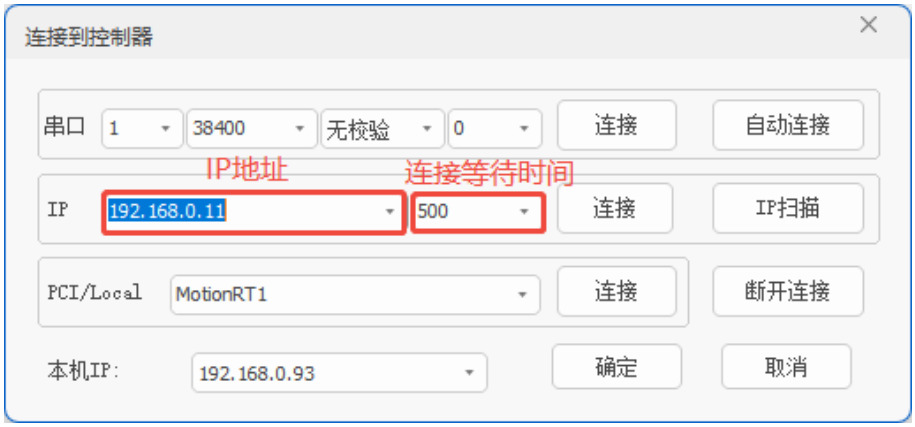
12.5.3-1 RTSys 连接控制器

网口连接：

正运动控制器，默认出厂 IP 为 192.168.0.11，在连接之前首先需要确认，IP 地址是否与控制器在同一个网段。如 IP 为：192.168.0.11。则连接的网口也需要为 0 网段（网口 IP 不可与控制器 IP 冲突）如下图（修改本机 IP 地址方法：控制面板->网络和 Internet->网络和共享中心->更改适配器设置）：



设置好网段之后，即可打开 RTSys，点击菜单栏：控制器->连接->控制器。在弹出的窗口，填写 IP 地址，点击连接。如下图：



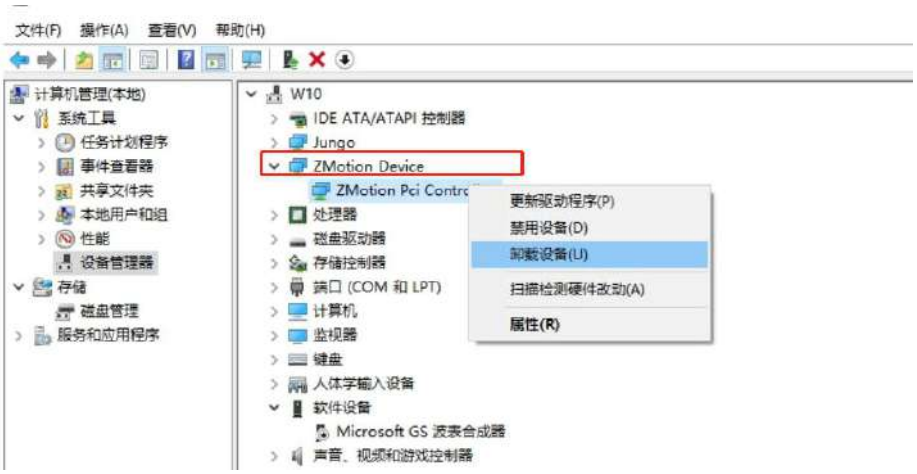
串口连接：

在电脑能识别 USB 转串口线，且在确保串口参数无误下，打开 RTSys，点击菜单栏：控制器->连接->控制器。在弹出的窗口，选择对应的串口号，点击连接。如下图：



PCI 连接：

先用厂家提供的 PCI 签名驱动，把 PCI 卡的驱动先装上。安装成功后，设备管理器里面会显示 Zmotion Device 如下图：

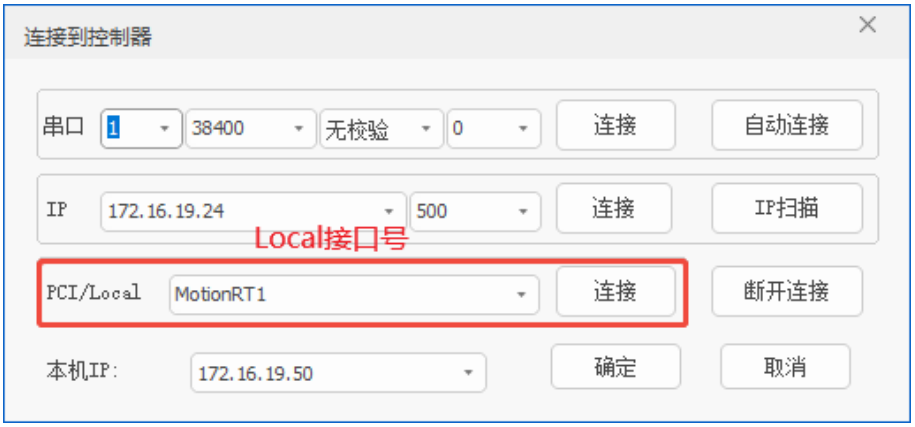


装完驱动后，打开打开 RTSys，点击菜单栏：控制器->连接->控制器。在弹出的窗口，选择对应的卡号，点击连接。如下图：



Local 连接：

该连接方式需将控制卡插入计算机/工控机卡槽中，根据对应用户手册安装 MotionRT 驱动后，打开 MotionRT 软件进行相应配置后点击启动。打开打开 RTSys，点击菜单栏：控制器->连接->控制器。在 RTSys 中 Local 栏即可扫描到 MotionRT。



12.5.3-2 RTSys 控制器状态

连接上控制器后，可通过控制器->控制器状态。在弹出的窗口，查看控制器相关信息。

基本信息：



控制器状态	说明
VirtualAxises	支持最大虚拟轴数
RealAxises	支持最大电机轴数
Taskes	最大任务数
Files/3Files	最大文件/三次文件数
Modbus0x Bits	Modbus 位寄存器用户可用空间大小
Modbus4x Regs	Modbus 字寄存器用户可用空间大小
VR Regs	VR 寄存器用户可用空间大小
TABLE Regs	TABLE 数组用户可用空间大小
RomSize	Rom 容量
FlashSize	Flash 容量
SoftType	软件型号
RTVersion	系统软件版本+固件版本
Build Date	固件版本生成时间
IpAddress	控制器 IP 地址
HardVersion	硬件版本
ControllerID	控制器唯一 ID
ZvlibVersion	视觉库文件版本
Axis features list	轴类型列表

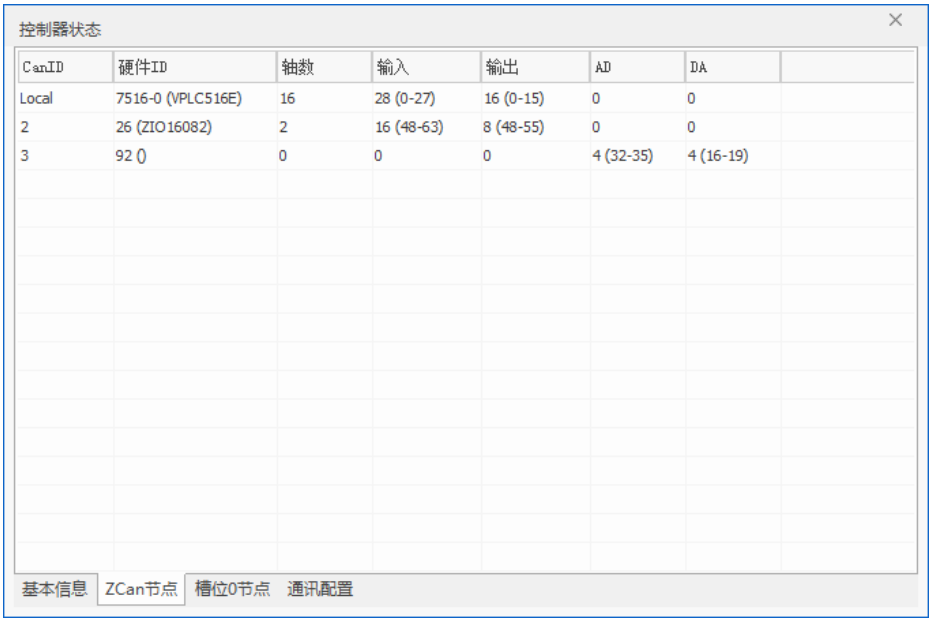
Axis features list（轴特点列表）	说明
0: Step Encoder	轴 0：脉冲+反馈
1: Step Encoder	轴 1：脉冲+反馈
2: Step Encoder	轴 2：脉冲+反馈

3: Step Encoder	轴 3：脉冲+反馈
4: Step Encoder	轴 4：脉冲+反馈
5: Step Encoder	轴 5：脉冲+反馈

CAN 节点信息：

在使用 ZCAN 扩展，扩展 IO 扩展卡卡之后，可以通过工具栏：控制器->控制器状态->ZCAN 节点观察到当前 ZCAN 各个节点信息，以及配置信息。如下图：

CANID 为 Loacl 是本体控制器资源，带输入 28 个，输出 16 个。为了避免 IO 地址冲突，所以扩展卡的起始地址需要设置为 28 之后，因为起始地址需要为 16 的整数倍（部分 8 个 IO 的扩展卡，则为 8 的整数倍），所以起始地址需要设置为 48。若扩展卡拨码不成功，对应节点信息不会显示，因此可以在本界面观察 ZCAN 扩展内容是否成功。



槽位 0 节点信息：

如果只有 EtherCAT 总线，槽位 0 是 EtherCAT 总线。

如果只有 RTEX 总线，槽位 0 是 RTEX 总线。

如果是 EtherCAT+RTEX 双总线，则槽位 0 是 EtherCAT 总线，槽位 1 是 RTEX 总线。

```
RAPIDSTOP(2)
WAIT IDLE(0)

SLOT_SCAN(0)
SLOT_START(0)

NODE_IO(0,0)=96
```

控制器状态

节点号	厂商ID	设备ID	拨码号	轴数	输入	输出	AD	DA
0	41bh	1ab0h	0	8	32 (32-63)	16 (32-47)	0	0

基本信息 ZCan节点 槽位0节点 通讯配置

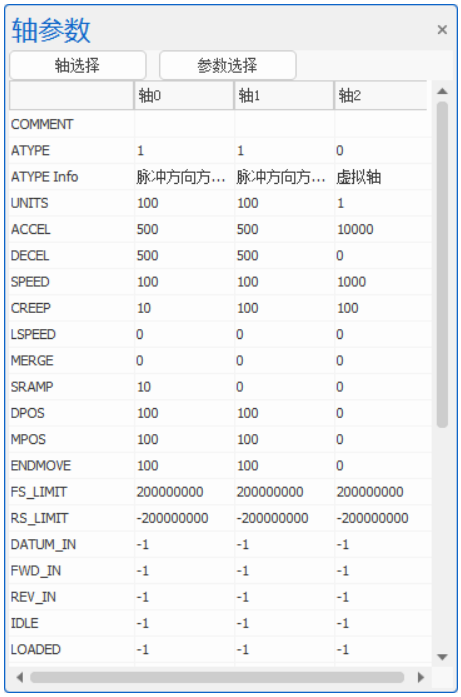
左边是一个最基本的 EtherCAT 初始化程序。

右边显示的第 1 至 N 行包含的是 EtherCAT 节点的设备信息。

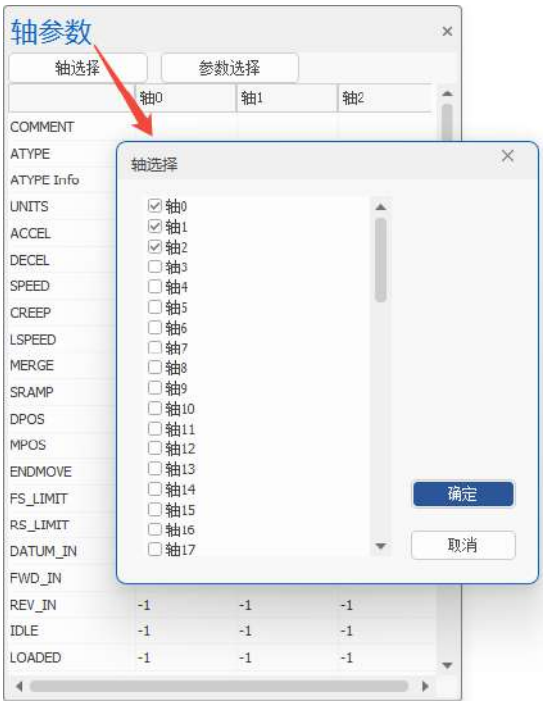
如上图，EtherCAT 节点 0 的硬件厂家 ID 为 41bh(正运动技术的厂商 ID)，硬件厂家设备 ID 为 1ab0h (正运动技术的产品 ID，对应产品为 EIO24088)，拨码号 0，8 个扩展轴，数字量输入 32 个，编号 IN(96)-IN(127)，数字量输出 16 个，编号 OP(96)-OP(111)，无 AD 和 DA。

12.5.3-3 RTSys 轴参数详解

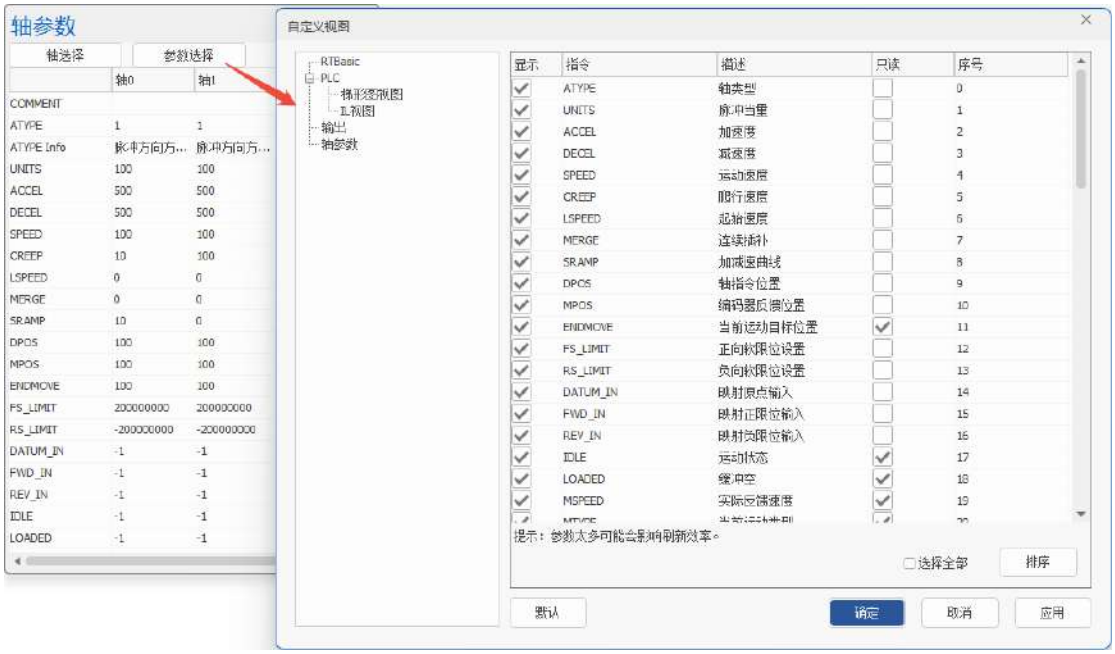
在对轴运动之前，需要对轴设置参数，方可进行运动。如设置（轴类型、脉冲当量、轴速度等），以及相关安全设置（正负硬件限位、正负软件限位、报警信号、急停减速度等）。RTSys 提供，直接对轴参数进行简易修改的操作界面。该界面是 RTSys 默认打开的，在整个界面的右则，如下图：（若不小心关闭了，也可在菜单栏：视图->轴参数界面打开）



其中选择要观察或者调试哪个的轴参数，直接点击轴选择，选择对应轴号即可。如图：



同样的，如果当前窗口显示的信息没有想要的，也可以勾选参数选择，选择对应的参数进行勾选。如下图：



各个参数对应的详细信息表：

指令	描述	读写性	序号
Atype	轴类型	可读可写	0
Units	脉冲当量	可读可写	1
Accel	加速度	可读可写	2
Decel	减速度	可读可写	3
Speed	运行速度	可读可写	4
Creep	爬行速度	可读可写	5
Lspeed	起始速度	可读可写	6
Merge	连续插补开关	可读可写	7
Sramp	加减速曲线时间设置	可读可写	8
Dpos	轴指令规划位置	可读可写	9
Mpos	编码器反馈位置	可读可写	10
Endmove	当前运动目标位置	只读	11
Fs_Limit	正向软限位设置	可读可写	12
Rs_Limit	负向软限位设置	可读可写	13
Datum_In	映射原点输入	可读可写	14
Fwd_In	映射正限位输入	可读可写	15
Rev_In	映射负限位输入	可读可写	16
Idle	运动状态	只读	17
Loaded	缓冲是否为空状态位	只读	18
Mspeed	实际编码反馈速度	只读	19
Mtype	当前运动类型	只读	20

Ntype	下条运动类型	只读	21
Remain	当前运动剩余距离	只读	22
Vp_Speed	当前运动速度	只读	23
Axisstatus	轴状态	只读	24
Move_Mark	运动标号	可读可写	25
Move_Curmark	当前运动标号	只读	26
Vector_Buffered	当前运动剩余距离	只读	27
Axis_StopReason	轴停止原因	只读	28
Moves_Buffered	当前运动缓冲数	只读	29
Axis_Address	轴地址	可读可写	30
Axis_Enable	单轴使能	可读可写	31
Force_Speed	sp 运动速度	可读可写	32
Startmove_Speed	sp 运动开始速度	可读可写	33
Endmove_Speed	sp 运动结束速度	可读可写	34
Fastdec	快减减速度	可读可写	35
Addax_Axis	叠加轴号	只读	36
Link_Axis	连接轴号	只读	37
Corner_Mode	拐角模式	可读可写	38
Decel_Angle	拐角减速开始	可读可写	39
Stop_Angle	拐角减速结束	可读可写	40
Full_Sp_Radius	限速半径	可读可写	41
Splimit_Radius	限速值	可读可写	42
Zsmooth	倒角半径	可读可写	43
Vector_Moved	当前运动的距离	可读可写	44
Endmove_Buffer	缓冲最终位置	只读	45
Homewait	回零反找延时	可读可写	46
Fast_Jog	映射点动输入	可读可写	47
Fwd_Jog	映射正向 Jog 输入	可读可写	48
Rev_Jog	映射负向 Jog 输入	可读可写	49
Jogspeed	映射 Jog 速度	可读可写	50
Fhold_in	映射保持输入	可读可写	51
Fhspeed	保持速度	可读可写	52
Encoder	编码器原始值	只读	53
Encoder_Status	编码器状态	只读	54
PP_Step	编码器内部比例	可读可写	55
Reg_Inputs	锁存输入映射	可读可写	56

Mark	锁存触发	只读	57
MarkB	锁存 2 触发	只读	58
MarkC	锁存 3 触发	只读	59
MarkD	锁存 4 触发	只读	60
Reg_Pos	锁存位置	只读	61
Reg_PosB	锁存 2 位置	只读	62
Reg_PosC	锁存 3 位置	只读	63
Reg_PosD	锁存 4 位置	只读	64
Alm_In	映射报警输入	可读可写	65
Rep_Option	坐标循环模式	可读可写	66
Rep_Dist	坐标循环位置	可读可写	67
Invert_Step	脉冲模式设置	可读可写	68
Max_speed	脉冲频率限制	可读可写	69
Axis_Zset	精准输出设置	可读可写	70
Dac	总线轴模拟量控制	可读可写	71
ErrorMask	错误时操作	可读可写	72

12.5.3-4 RTSys 手动运动

在运动之前，先确认几个基本参数的设置，即轴参数里面的轴类型(Atype)，脉冲当量(Units)，加减速度(Accel/Decel)，速度(Speed)。如下图：



1) 轴类型设置(ATYPE)：

a) 若是脉冲方式控制步进或者伺服，则轴类型设置为：

1（脉冲方向方式的步进或伺服）或者 7(脉冲方向方式的步进或伺服+EZ 信号输入)；

b) 若是脉冲方式控制步进或者伺服且需要编码器反馈，则轴类型设置为(常见设置为4)：4（脉冲方向输出+正交编码器输入）或者 5（脉冲方向输出+脉冲方向编码器输入）；

c) 若是单独一个编码器，则轴类型设置为（常见设置为 3）：

3（正交编码器）或者 6（脉冲方向方式的编码器）；

d) 若为总线类型，则无需设置轴类型，把厂家提供的总线初始化文件，下载到控制器里面即可，下载之后，轴类型会自动变成(常见为 65)：

55(RTEX 周期位置模式)，56(RTEX 周期速度模式)，57(RTEX 周期力矩模式)

65(ECAT 周期位置模式)，66(ECAT 周期速度模式)，67(ECAT 周期力矩模式)

2) 脉冲当量设置(Units)：

脉冲当量为控制器运动的基本单位，速度，加减速度都是以脉冲当量为单位的。

若电机 1000 个脉冲旋转 1 圈，丝杠一圈螺距 P=2mm，减速比 1：1

当脉冲当量为 1 时：控制轴走 1000，则轴走 1000 个脉冲当量，旋转了一圈，走了 2mm。

当脉冲当量为 1000/2 时：控制轴走 2,则轴走了 1000 个脉冲当量，旋转了一圈，走了 2mm。

3) 加减速度，速度设置（Accel/Dece/Speed）：

加减速度，速度以脉冲当量为单位，当加减速度，速度都为 0，控制轴运动，轴会是禁止状态，当加减速度，速度都为很大值时，控制轴运动，轴运动时，则会抖动或者报警过载，所以需要对加减速，速度设置合适值。

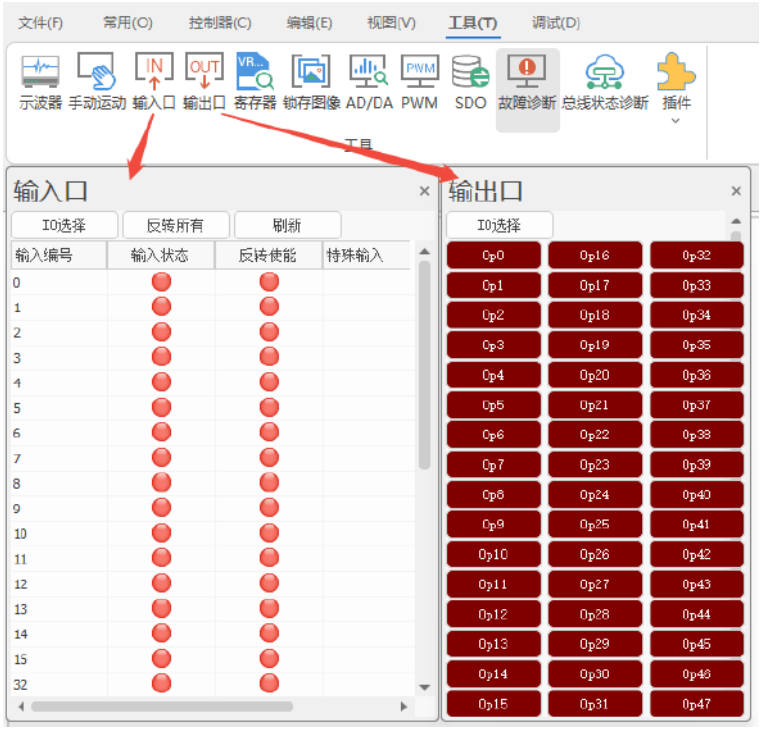
4) 操作运动

在以上参数设置完成之后，可在左上角菜单栏：工具->手动运动，弹出窗口，对轴进行手动运动。



12.5.3-5 RTSys IO 操作

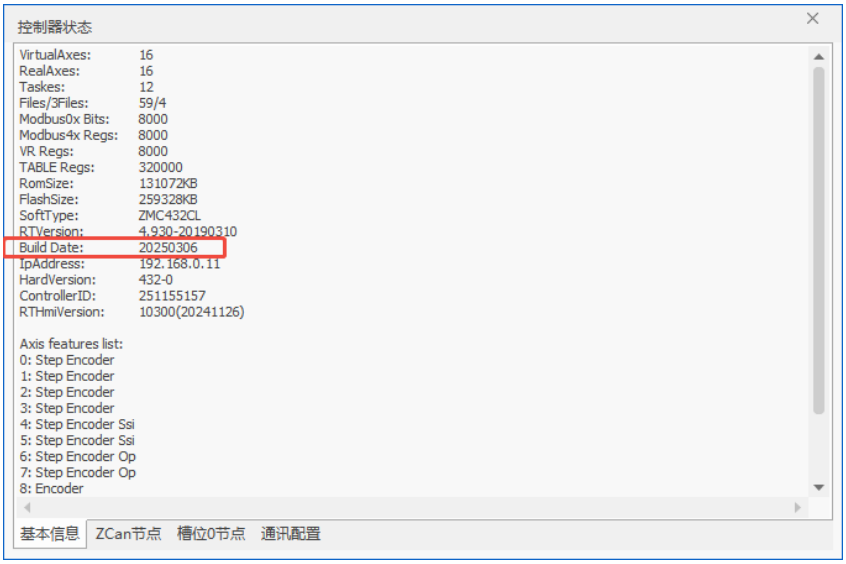
要观察对应的输入或者操作输出点，可以通过左上角菜单栏：工具->输入口/输出口，打开对应界面操作。如下图：



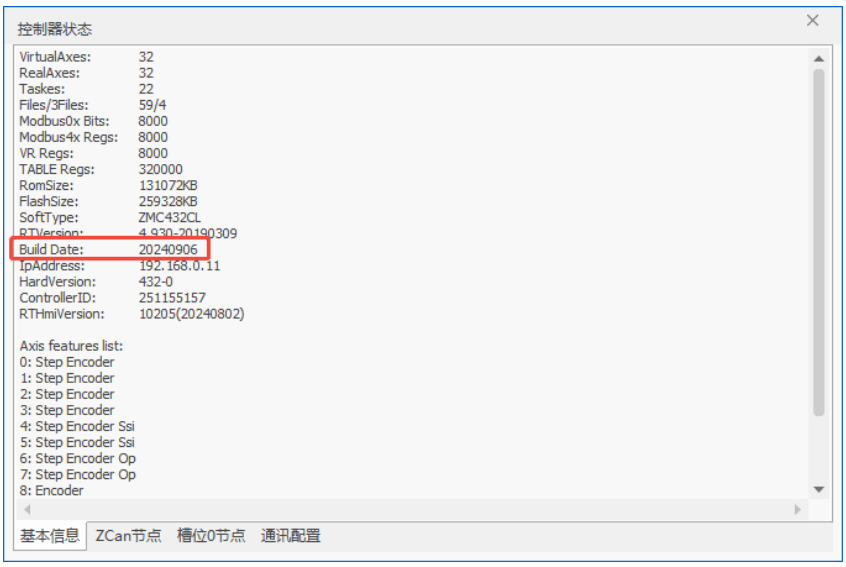
12.5.3-6 RTSys 固件升级

在使用控制器中，遇到一些问题，需要更新固件，可使用 RTSys 对控制器进行固件更新。

例：对 ZMC432CL-V2 控制器进行固件升级，更新固件前，固件版本：



目标固件版本：



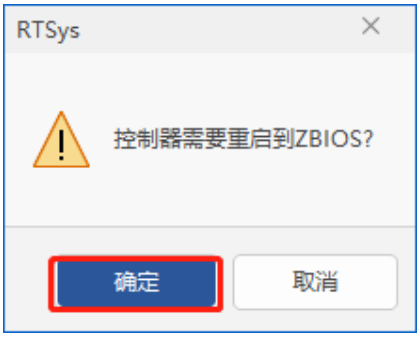
1. 打开 RTSys：左上角菜单栏->控制器->固件升级，弹出固件升级窗口：

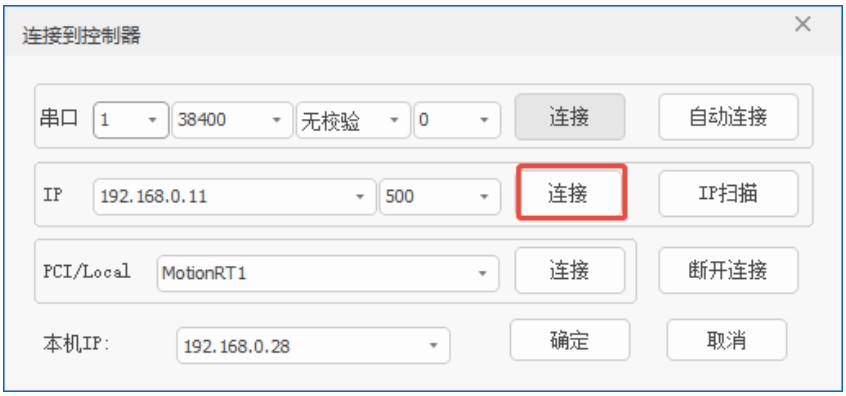


2. 点击浏览，选择保存的固件文件：

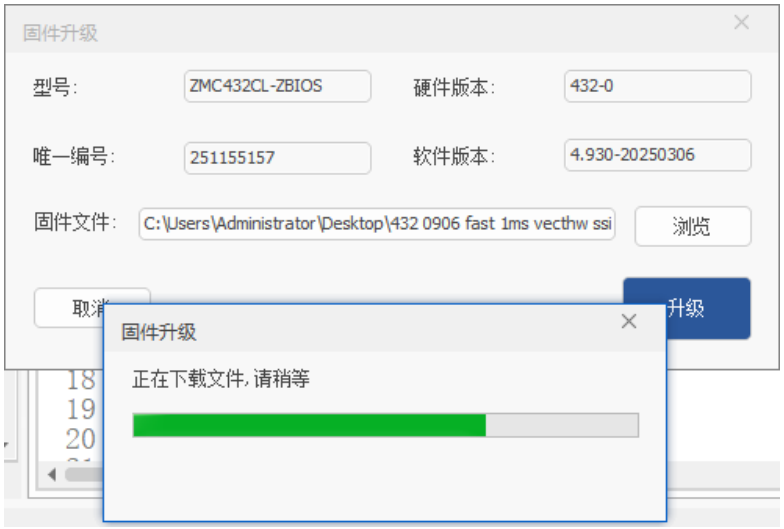


3. 选中固件后，点击打开，后点击升级。第一次点击升级，会弹出提示，进入 ZBIOS 状态，点击确定，后会弹出连接控制器窗口，点击连接，既进入 ZBIOS 状态：

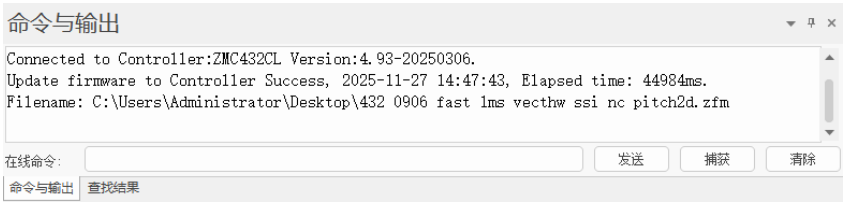




4. 点击连接之后，再次点击升级，会弹窗出现升级加载窗口界面：



5. 进度条满格后，“固件升级”界面消失，控制器输出框如下，显示固件升级成功：



12.5.3-7 RTSys 修改 IP 地址

修改 IP 地址，需要连接上控制器后方可修改 IP 地址；在打开 RTSys 连接上控制器后，点击菜单栏：控制器->修改 IP 地址，弹出窗口：



在新的 IP 地址填写新的 IP 地址，点击确认，重新连接控制器即可。

第十三章 指令详解

<table>
<tr><td>指令 1</td><td>ZAux_OpenCom</td></tr>
<tr><td>指令原型</td><td>int32 __stdcall ZAux_OpenCom(uint32 comid,ZMC_HANDLE *phandle)</td></tr>
<tr><td>指令说明</td><td>串口方式连接控制器。</td></tr>
<tr><td>输入参数</td><td><table><tr><td>参数名</td><td>描述</td></tr><tr><td>Comid</td><td>连接的 COM 口编号</td></tr></table></td></tr>
<tr><td>输出参数</td><td><table><tr><td>参数名</td><td>描述</td></tr><tr><td>Phandle</td><td>控制器连接句柄</td></tr></table></td></tr>
<tr><td>返回值</td><td>成功返回值为 0，非 0 详见错误码说明。</td></tr>
<tr><td>指令示例</td><td>串口连接控制器</td></tr>
<tr><td>详细说明</td><td>控制器提供多种方式连接，该指令为串口连接方式，较于网口连接，串口连接，通讯速率不高。
2.ZMC_HANDLE 类型：Zmotion 库中，专门用于控制卡连接数据定义类型；
3.控制器串口默认的参数<table><tr><td>波特率</td><td>数据位</td><td>停止位</td><td>校验位</td></tr><tr><td>38400</td><td>8</td><td>1</td><td>无</td></tr></table>4.如果控制器的波特率不是默认的 38400，在连接前就需要使用 ZAux_SetComDefaultBaud 将停止位设为 0，其他参数保持一致才可以与控制器通讯上。</td></tr>
</table>

<table>
<tr><td>指令 2</td><td>ZAux_SetComDefaultBaud</td></tr>
<tr><td>指令原型</td><td>int32 __stdcall ZAux_SetComDefaultBaud(uint32 dwBaudRate, uint32 dwByteSize, uint32 dwParity, uint32 dwStopBits)</td></tr>
<tr><td>指令说明</td><td>设置串口通讯参数。</td></tr>
<tr><td>输入参数</td><td><table><tr><td>参数名</td><td>描述</td></tr><tr><td>dwBaudRate</td><td>波特率缺省：38400</td></tr><tr><td>dwByteSize</td><td>数据位大小缺省：8 位</td></tr><tr><td>dwParity</td><td>校验位缺省：0 无校验</td></tr><tr><td>dwStopBits</td><td>停止位缺省：1 位</td></tr></table></td></tr>
<tr><td>输出参数</td><td>/</td></tr>
</table>

返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	串口连接控制器
详细说明	与 RTBasic 中的 SetCom 不一致，该函数只做串口参数设置。

指令 3	**ZAux_OpenEth**
指令原型	int32 __stdcall ZAux_OpenEth(char *ipaddr, ZMC_HANDLE * phandle)
指令说明	以太网连接控制器。
输入参数	参数名 / 描述 Ipaddr / 连接的 IP 地址
输出参数	参数名 / 描述 Phandle / 返回的连接句柄
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	网口连接控制器
详细说明	1.网口采用 RJ45 标准网线接口，通讯速率为 100Mbit/s。 2.控制器出厂的 IP 地址为 192.168.0.11，端口号为 502。对端通讯设备需与控制器处于同一网段,才可进行连接。 3.最常用的控制器连接方式。 4.ZMC_HANDLE 类型：Zmotion 库中，专门用于控制卡连接数据定义类型；

指令 4	**ZAux_OpenPci**
指令原型	int32 __stdcall ZAux_OpenPci(uint32 cardnum, ZMC_HANDLE *phandle)
指令说明	连接 PCI 卡。
输入参数	参数名 / 描述 Cardnum / PCI 卡号，缺省从 0 开始。
输出参数	参数名 / 描述 Phandle / 连接句柄
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	PCI 方式连接控制器
详细说明	1.PCI 连接，需要 PCI 类型的控制卡才可进行 PCI 连接，该类型连接通讯速率最高，但该连接需要工控机含有 PCI 卡槽接口。 2.ZMC_HANDLE 类型：Zmotion 库中，专门用于控制卡连接数据定义类型；

指令 5	ZAux_Close
指令原型	int32 __stdcall ZAux_Close(ZMC_HANDLE handle)
指令说明	关闭控制器连接。
输入参数	参数名 | 描述 Handle | 连接句柄。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	串口连接控制器
详细说明	/

指令 6	ZAux_SetIp
指令原型	int32 __stdcall ZAux_SetIp(ZMC_HANDLE handle, char * ipaddress)
指令说明	设置控制器 IP 地址。
输入参数	参数名 | 描述 Handle | 连接句柄 Ipaddress | 设置的 IP 地址值
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	修改 IP 地址
详细说明	若是使用网口连接的控制器，在修改 IP 地址成功后，控制器将会自动与 PC 断开连接，需把 PC 更换成对应的网段之后，在重新进行连接

指令 7	ZAux_SearchEthlist
指令原型	int32 __stdcall ZAux_SearchEthlist(char *ipaddrlist, uint32 addrbufflength, uint32 uims)
指令说明	搜索当前网段下的 IP 地址。
输入参数	参数名 | 描述 Addrbufflength | 搜索返回的 IP 地址总长度 Uims | 搜索超时时间。
输出参数	参数名 | 描述 Ipaddrlist | 搜索到的全部 IP 地址
返回值	成功返回值为 0，非 0 详见**错误码**说明。

指令示例	//自动搜索 IP 地址 char buffer[4096+256+1]; int32 iresult; iresult = ZAux_SearchEthlist(buffer, 4096+256+1, 100);//检索 IP 地址到 Buffer if(0!=iresult){return 0;}//失败退出 //从字符串转换过来 int ipos =0; const char * pstring; pstring = buffer; char buffer2[256]; buffer2[0] = '\0'; for(int j= 0; j< 20;j++) { //跳过空格 while(' ' == pstring[0]) { pstring++; } memset(buffer2, 0, sizeof(buffer2)); ipos = sscanf(pstring , "%s", &buffer2);//获取地址到 buffer2 if(-1 == ipos){break;} printf("%s\n",buffer2); //指到 IP 地址末尾 while((' ' != pstring[0]) && ('\t' != pstring[0]) && ('\0' != pstring[0])) { pstring++; } }
详细说明	控制器默认 IP 地址 192.168.0.11 若控制器与 PC 的 IP 地址不在通一个网段，则会扫不出 IP 地址

指令 8	**ZAux_SearchAndOpenCom**
指令原型	int32 __stdcall ZAux_SearchAndOpenCom(uint32 uimincomidfind,

	uint32 uimaxcomidfind, uint *pcomid, uint32 uims, ZMC_HANDLE *phandle)
指令说明	自动搜索串口并连接控制器。
输入参数	参数名 | 描述 uimincomidfind | 搜索的最小 COM 口编号 uimaxcomidfind | 搜索的最大 COM 口编号 uims | 连接响应时间 MS
输出参数	参数名 | 描述 Pcomid | 搜索到的 COM 口编号 phandle | 连接返回的句柄
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	/
详细说明	/

指令 9	**ZAux_SearchEth**
指令原型	int32 __stdcall ZAux_SearchEth(const char *ipaddress, uint32 uims)
指令说明	搜索当前 IP 地址是否在该网段。
输入参数	参数名 | 描述 ipaddress | 搜索判断的 IP 地址。 uims | 搜索超时时间。
输出参数	/
返回值	存在返回值为 0，非 0 详见**错误码**说明。
指令示例	/
详细说明	/

指令 10	**ZAux_GetMaxPciCards**
指令原型	int32 __stdcall ZAux_GetMaxPciCards( )
指令说明	搜索当前最大 PCI 卡数。
输入参数	/
输出参数	/
返回值	最大卡数量。
指令示例	Int Card; Card=ZAux_GetMaxPciCards();//获取最大卡的数量

详细说明	/

指令 11	**ZAux_FastOpen**
指令原型	int32 __stdcall ZAux_FastOpen(int type, char *pconnectstring, uint32 uims ,ZMC_HANDLE * phandle)
指令说明	与控制器建立连接, 可以指定连接的等待时间
输入参数	参数名 | 描述 type | 连接类型 type: 1-COM 2-ETH 4-PCI 5-MotionRT pconnectstring | 连接字符串: type=1：COM+串口号 type=2：IP 地址 type=4：Pci+卡号 type=5：LOCAL+卡号 uims | 连接超时时间 uims;单位 ms
输出参数	参数名 | 描述 phandle | 控制器连接句柄
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	串口连接:（注：若控制器的波特率不是默认的 38400，在连接前就需要使用 ZAux_SetComDefaultBaud 将停止位设为 0，其他参数保持一致才可以与控制器通讯上） ZMC_HANDLE phandle;//控制器连接句柄 Char comID[32]= "Com0";//串口 ID，不能直接写串口号，需要在串口号前加个 Com ZAux_FastOpen(1, comID,1000 ,&phandle); 网口连接例子： ZMC_HANDLE phandle;//控制器连接句柄 Char EthID[32]= "192.168.0.11";//网口 ID ZAux_FastOpen(2, EthID,1000 ,&phandle); PCI 连接例子： ZMC_HANDLE phandle;//控制器连接句柄 ZAux_FastOpen(4, “PCI1”,1000 ,&phandle); MotionRT 连接例子： ZMC_HANDLE phandle;//控制器连接句柄

	ZAux_FastOpen(5, “LOCAL1”,1000 ,&phandle);
详细说明	type 设置为 5,zmotion.dll 版本要在 3.8.8.50 以上。 若 type 设置为 5，字符串非 LOACL+卡号，则自动默认为 LOCAL1

指令 12	**ZAux_GetControllerInfo**
指令原型	int32 __stdcall ZAux_GetControllerInfo(ZMC_HANDLE handle ,char *SoftType ,char *SoftVersion ,char *ControllerId)
指令说明	读取控制器厂商信息。
输入参数	参数名 | 描述 handle | 控制器连接句柄
输出参数	参数名 | 描述 SoftType | 控制器软件型号。 SoftVersion | 控制器固件版本号。 ControllerId | 控制器唯一 ID。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	控制器信息
详细说明	/

指令 13	**ZAux_GetSysSpecification**
指令原型	int32 __stdcall ZAux_GetSysSpecification(ZMC_HANDLE handle, uint16 *Max_VirtuAxises,uint8 *Max_motor,uint8 *Max_io)
指令说明	读取控制器最大规格参数。
输入参数	参数名 | 描述 handle | 控制器连接句柄
输出参数	参数名 | 描述 Max_VirtuAxises | 控制器最大虚拟轴数。 Max_motor | 控制器实际轴数。 Max_io | IN,OUT,AD,DA 每个占一个 uint8 长度。
返回值	成功返回值为 0，非 0 详见**错误码**说明。

指令示例	控制器信息
详细说明	/

指令 14	ZAux_GetRtcTime
指令原型	int32 __stdcall ZAux_GetRtcTim(ZMC_HANDLE handle, char *RtcDate, char * RtcTime)
指令说明	读取控制器系统时间以及日期。默认从 2000 年 1 月 1 日起
输入参数	参数名：handle 描述：控制器连接句柄
输出参数	参数名：RtcDate 描述：控制器系统日期。格式:YYMMDD 或 YYYYMMDD 参数名：RtcTime 描述：控制器系统时间。格式:HHMMSS
返回值	详见**错误码**说明。
指令示例	char RtcDate[8]; char RtcTime[6]; ZAux_GetRtcTime(handle,RtcDate,RtcTime);
详细说明	YYMMDD 格式：如 220101(22 年 1 月 1 日) YYYYMMDD 格式：如 20220101(2022 年 1 月 1 日) HHMMSS 格式:如 131400(13 点 14 分 00 秒)

指令 15	ZAux_SetRtcTime
指令原型	int32 __stdcall ZAux_SetRtcTim(ZMC_HANDLE handle, char * RtcDate,char * RtcTime)
指令说明	设置控制器系统时间以及日期，掉电保存，默认从 2000 年 1 月 1 日起。仿真器时间日期无法修改

输入参数	<table><tr><td>参数名</td><td>描述</td></tr><tr><td>handle</td><td>控制器连接句柄</td></tr><tr><td>RtcDate</td><td>控制器系统日期。 格式:YYMMDD 或 YYYYMMDD</td></tr><tr><td>RtcTime</td><td>控制器系统时间。 格式:HHMMSS</td></tr></table>
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	char RtcDate[20]="20220101"; char RtcTime[20]="000000"; ZAux_SetRtcTime(handle,RtcDate,RtcTime);
详细说明	YYMMDD 格式：如 220101(22 年 1 月 1 日) YYYYMMDD 格式：如 20220101(2022 年 1 月 1 日) HHMMSS 格式:如 131400(13 点 14 分 00 秒)

指令 16	**ZAux_Direct_SetAtype**
指令原型	int32 __stdcall ZAux_Direct_SetAtype(ZMC_HANDLE handle, int iaxis, int iValue)
指令说明	设置指定轴的轴类型。
输入参数	<table><tr><td>参数名</td><td>描述</td></tr><tr><td>handle</td><td>控制器连接句柄</td></tr><tr><td>iaxis</td><td>轴号</td></tr><tr><td>iValue</td><td>轴类型（见下方详细说明 2）</td></tr></table>
输出参数	/
返回值	详见**错误码**说明。
指令示例	轴基本运动参数设置获取
详细说明 1	1.最好是在程序初始化的时候就设置好 ATYPE。 2.ZCAN 扩展轴要先设置 AXIS_ADDRESS，并在设置后延迟 2 个任务周期再调用运动指令，受总线带宽限制，ZCAN 扩展轴不要设置超过 2 个。 3.对部分产品型号带有独立的编码器，可以使用相应虚拟轴来做编码器轴使用，例如 ZMC206 的电机轴为 0-5 轴，编码器可以通过轴 6-11 来控制，详细可通过 RTSys 软件连接上控制器以后查看控制器状态。

详细说明 2

轴类型	描述
0	虚拟轴。
1	脉冲方向方式的步进或伺服 。
2	模拟信号控制方式的伺服 。
3	正交编码器 。
4	脉冲方向输出+正交编码器输入。
5	脉冲方向输出+脉冲方向编码器输入
6	脉冲方向方式的编码器，可用于手轮输入。
7	脉冲方向方式步进或伺服+EZ 信号输入。
8	ZCAN 扩展脉冲方向方式步进或伺服 。
9	ZCAN 扩展正交编码器。
10	ZCAN 扩展脉冲方向方式的编码器。
20	振镜类型，带振镜状态反馈；振镜连接不上 AXISSTATUS 的 bit2 会置位 ENCODER 返回原始的发送位置，脉冲单位；ZMC408SCAN 支持
21	振镜轴类型，需要控制器支持；缺省系统周期 250us，振镜刷新周期 50us，与固件有关；可以使用普通轴的所有运动控制指令，支持振镜轴与其它轴类型混合插补
22	振镜轴类型，带振镜位置反馈；振镜连接不上 AXISSTATUS 的 bit2 会置位，振镜报警 AXISSTATUS 的 bit3 会置位；MPOS 返回反馈位置，做了反矫正处理，ENCODER 返回原始的反馈位置脉冲单位；ZMC408SCAN 支持
24	远程编码器轴类型；ZHD500X 上手轮使用，需要控制器 5 系列 20180404 以上固件版本支持
50	RTEX 周期位置模式，需 Rtex 控制器
51	RTEX 周期速度模式，需 Rtex 控制器
52	RTEX 周期力矩模式，需 Rtex 控制器；请先关闭驱动器 2 自由度控制模式，并设置设置速度限制
65	ECAT 周期位置模式，需支持 EtherCAT
66	ECAT 周期速度模式，需支持 EtherCAT Profile 要设置为 20 或以上
67	ECAT 周期力矩模式，需支持 EtherCAT PROFILE 要设置为 30 或以上
70	ECAT 自定义操作，只读取编码器，需支持 EtherCAT

<table>
<tr><td>指令 17</td><td>ZAux_Direct_GetAtype</td></tr>
<tr><td>指令原型</td><td>int32 __stdcall ZAux_Direct_GetAtype(ZMC_HANDLE handle, int iaxis, int *piValue)</td></tr>
<tr><td>指令说明</td><td>读取指定轴的轴类型。</td></tr>
<tr><td>输入参数</td><td><table><tr><td>参数名</td><td>描述</td></tr><tr><td>handle</td><td>控制器连接句柄</td></tr><tr><td>iaxis</td><td>轴号</td></tr></table></td></tr>
<tr><td>输出参数</td><td><table><tr><td>参数名</td><td>描述</td></tr><tr><td>piValue</td><td>轴类型（见下方详细说明）</td></tr></table></td></tr>
<tr><td>返回值</td><td>成功返回值为 0，非 0 详见错误码说明。</td></tr>
<tr><td>指令示例</td><td>轴基本运动参数设置获取</td></tr>
</table>

详细说明

轴类型	描述
0	虚拟轴。
1	脉冲方向方式的步进或伺服。
2	模拟信号控制方式的伺服。
3	正交编码器。
4	步进+编码器。
6	脉冲方向方式的编码器，可用于手轮输入。
7	脉冲方向方式步进或伺服+EZ 信号输入。
8	ZCAN 扩展脉冲方向方式步进或伺服。
9	ZCAN 扩展正交编码器。
10	ZCAN 扩展脉冲方向方式的编码器。
20	振镜类型，带振镜状态反馈；振镜连接不上 AXISSTATUS 的 bit2 会置位 ENCODER 返回原始的发送位置，脉冲单位；ZMC408SCAN 支持
21	振镜轴类型，需要控制器支持；缺省系统周期 250us，振镜刷新周期 50us，与固件有关；可以使用普通轴的所有运动控制指令，支持振镜轴与其它轴类型混合插补
22	振镜轴类型，带振镜位置反馈；振镜连接不上 AXISSTATUS 的 bit2 会置位，振镜报警 AXISSTATUS 的 bit3 会置位；MPOS 返回反馈位置，做了反矫正处理，ENCODER 返回原始的反馈位置脉冲单位；ZMC408SCAN 支持
24	远程编码器轴类型；ZHD500X 上手轮使用，需要控制器 5 系列 20180404 以上固件版本支持
50	RTEX 周期位置模式，需 Rtex 控制器
51	RTEX 周期速度模式，需 Rtex 控制器
52	RTEX 周期力矩模式，需 Rtex 控制器；请先关闭驱动器 2 自由度控制模式，并设置设置速度限制
65	ECAT 周期位置模式，需支持 EtherCAT
66	ECAT 周期速度模式，需支持 EtherCAT Profile 要设置为 20 或以上
67	ECAT 周期力矩模式，需支持 EtherCAT PROFILE 要设置为 30 或以上
70	ECAT 自定义操作，只读取编码器，需支持 EtherCAT

指令 18	ZAux_Direct_SetUnits
指令原型	int32 __stdcall ZAux_Direct_SetUnits(ZMC_HANDLE handle,int iaxis, float fValue)
指令说明	设置脉冲当量，当设置为 1 时，表示单位为 1 个脉冲。 脉冲当量，控制器基本单位。指定每单位发送的脉冲数，支持 5 位小数精度。 控制器以脉冲当量作为基本单位，修改后坐标显示会随脉冲当量改变比例变化。
输入参数	参数名 | 描述 handle | 控制器连接句柄 iaxis | 轴号 fValue | 设置的脉冲当量
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	轴基本运动参数设置获取
详细说明	**如何设置** 假设电机 U=3600 脉冲转一圈，丝杠一圈螺距 P=2mm： 电机转 1° 对应的脉冲当量 UNITS： UNITS=U/360=3600/360=10；//此时 MOVE(1)，电机转 1° 工作台走 1mm 对应的脉冲当量 UNTIS： UNITS=U/P=3600/2=1800，//此时 MOVE(1)，工作台走 1mm 机台存在减速比时，要把减速比算上，假设减速比 i=2:1 UNITS=U*i/P=3600*2/2=3600

指令 19	ZAux_Direct_GetUnits
指令原型	int32 __stdcall ZAux_Direct_GetUnits(ZMC_HANDLE handle,int iaxis, float *pfValue)
指令说明	读取脉冲当量。 脉冲当量，控制器基本单位。指定每单位发送的脉冲数，支持 5 位小数精度 控制器以脉冲当量作为基本单位，修改后坐标显示会随脉冲当量改变比例变化。

输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。
输出参数	参数名 | 描述 pfValue | 返回轴的脉冲当量。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	轴基本运动参数设置获取
详细说明	/

指令 20	**ZAux_Direct_SetAccel**
指令原型	int32 __stdcall ZAux_Direct_SetAccel(ZMC_HANDLE handle,int iaxis, float fValue)
指令说明	设置加速度。单位为 units/s/s。
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。 pfValue | 设置的加速度。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	轴基本运动参数设置获取
详细说明	1.当多轴运动时，插补运动的加速度设置主轴加速度。 （主轴：轴列表数组里第 0 个数据所指定的轴号） 2.建议运动前设置好加速度和减速度，运动中不要修改，运动中调整会导致速度曲线变化。

指令 21	**ZAux_Direct_GetAccel**
指令原型	int32 __stdcall ZAux_Direct_GetAccel(ZMC_HANDLE handle,int iaxis, float *pfValue)
指令说明	读取加速度。单位为 units/s/s。
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。

输出参数	<table><tr><td>参数名</td><td>描述</td></tr><tr><td>pfValue</td><td>返回的加速度。</td></tr></table>
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	轴基本运动参数设置获取
详细说明	1.当多轴运动时，插补运动的加速度读取主轴加速度。 （主轴：轴列表数组里第 0 个数据所指定的轴号） 2.建议运动前设置好加速度和减速度，运动中不要修改，运动中调整会导致速度曲线变化。

指令 22	**ZAux_Direct_SetDecel**
指令原型	int32 __stdcall ZAux_Direct_SetDecel(ZMC_HANDLE handle,int iaxis, float fValue)
指令说明	设置减速度，单位 units/s/s。
输入参数	<table><tr><td>参数名</td><td>描述</td></tr><tr><td>handle</td><td>连接标识。</td></tr><tr><td>iaxis</td><td>轴号。</td></tr><tr><td>fValue</td><td>设置的减速度。</td></tr></table>
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	轴基本运动参数设置获取
详细说明	当多轴运动时，插补运动的加速度设置主轴减速度。 （主轴：轴列表数组里第 0 个数据所指定的轴号） 建议运动前设置好加速度和减速度，运动中不要修改，运动中调整会导致速度曲线变化。 当设置为 0 时，自动等于加速度 ACCEL 值，进行对称的加减速。

指令 23	**ZAux_Direct_GetDecel**
指令原型	int32 __stdcall ZAux_Direct_GetDecel(ZMC_HANDLE handle,int iaxis, float *pfValue);
指令说明	读取减速度，单位 units/s/s。
输入参数	<table><tr><td>参数名</td><td>描述</td></tr><tr><td>handle</td><td>连接标识。</td></tr><tr><td>iaxis</td><td>轴号。</td></tr></table>

输出参数	参数名 | 描述 pfValue | 获取减速度。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	轴基本运动参数设置获取
详细说明	1.当多轴运动时，插补运动的加速度读取主轴减速度。 （主轴：轴列表数组里第 0 个数据所指定的轴号） 2.建议运动前设置好加速度和减速度，运动中不要修改，运动中调整会导致速度曲线变化。 3.当设置为 0 时，自动等于加速度 ACCEL 值，进行对称的加减速。

指令 24	**ZAux_Direct_SetSpeed**
指令原型	int32 __stdcall ZAux_Direct_SetSpeed(ZMC_HANDLE handle,int iaxis, float fValue)
指令说明	设置轴速度，单位 units/s。
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。 fValue | 设置的轴速度。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	轴基本运动参数设置获取
详细说明	1.当多轴运动时，插补运动的速度设置主轴速度，作为插补运动的速度。 （主轴：轴列表数组里第 0 个数据所指定的轴号） 2.速度修改后，立刻生效，可以实现动态变速，但是改变瞬间会抖。希望平滑变速请使用 SPEED_RATIO（RTBasic 指令，PC 可以用这个函数：ZAux_Direct_SetParam(句柄, “SPEED_RATIO”,轴号, 当前速度的比例)）来代替掉速度函数，让他在线变速变得相对平滑。

指令 25	**ZAux_Direct_GetSpeed**
指令原型	int32 __stdcall ZAux_Direct_GetSpeed(ZMC_HANDLE handle,int iaxis, float *pfValue)
指令说明	读取轴速度。

输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。
输出参数	参数名 | 描述 pfValue | 获取轴速度。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	轴基本运动参数设置获取
详细说明	当多轴运动时，插补运动读取主轴速度，作为插补运动的速度。（主轴：轴列表数组里第 0 个数据所指定的轴号）

指令 26	**ZAux_Direct_SetDpos**
指令原型	int32 __stdcall ZAux_Direct_SetDpos(ZMC_HANDLE handle,int iaxis, float fValue)
指令说明	设置轴当前位置或称控制器发送的需求位置，单位 units。
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。 pfValue | 当前轴位置。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	轴基本运动参数设置获取
详细说明	写 DPOS 会自动转换为 OFFPOS 偏移，不会移动电机。

指令 27	**ZAux_Direct_GetDpos**
指令原型	int32 __stdcall ZAux_Direct_GetDpos(ZMC_HANDLE handle, int iaxis, float *pfValue)
指令说明	读取轴当前位置或称控制器发送的需求位置，单位 units。
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。
输出参数	参数名 | 描述 pfValue | 获取当前轴位置。

返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	轴基本运动参数设置获取
详细说明	/

指令 28	**ZAux_Direct_SetMpos**
指令原型	int32 __stdcall ZAux_Direct_SetMpos(ZMC_HANDLE handle, int iaxis, float fValue)
指令说明	设置轴测量反馈位置，单位 units。
输入参数	参数名：描述 handle：连接标识。 iaxis：轴号。 pfValue：设置当前轴测量反馈位置值。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	轴基本运动参数设置获取
详细说明	写 MPOS 会自动转换为 OFFPOS 偏移，不会移动电机。

指令 29	**ZAux_Direct_GetMpos**
指令原型	int32 __stdcall ZAux_Direct_GetMpos(ZMC_HANDLE handle, int iaxis, float *pfValue)
指令说明	读取轴测量反馈位置，单位 units。
输入参数	参数名：描述 handle：连接标识。 iaxis：轴号。
输出参数	参数名：描述 pfValue：获取当前轴测量反馈位置。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	轴基本运动参数设置获取
详细说明	/

指令 30	**ZAux_Direct_GetEncoder**
指令原型	int32 __stdcall ZAux_Direct_GetEncoder(ZMC_HANDLE handle,int iaxis, float *pfValue);

指令说明	读取内部编码器值。单位为脉冲
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。
输出参数	参数名 | 描述 pfValue | 返回当前轴脉冲为单位的反馈位置。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	float fValue; ret = ZAux_Direct_GetEncoder(handle, 0, &fValue); //读取轴 0 内部编码器值
详细说明	1.内部参数，只有配置为需要使用编码器的 ATYPE 时才可以读取。 2.驱动器有多圈绝对值编码器时，读取的就是多圈值。

指令 31	**ZAux_Direct_GetFastDec**
指令原型	int32 __stdcall ZAux_Direct_GetFastDec(ZMC_HANDLE handle, int iaxis, float * fValue)
指令说明	读取快速减速度，异常停止时自动采用的速度。单位为 units/s/s
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。
输出参数	参数名 | 描述 pfValue | 获取快减减速度的值。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	正负硬限位设置
详细说明	在 CANCEL,或异常停止时自动采用 当设置为 0 值时自动为 DECEL。

指令 32	**ZAux_Direct_SetFastDec**
指令原型	int32 __stdcall ZAux_Direct_SetFastDec(ZMC_HANDLE handle, int iaxis, float fValue)
指令说明	设置快速减速度，异常停止时自动采用的速度。单位为 units/s/s。

输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。 fValue | 设置的快减减速度的值。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	正负硬限位设置
详细说明	1.在 CANCEL,或异常停止时自动采用 2.当设置为 0 值时自动为 DECEL

指令 33	**ZAux_Direct_SetLspeed**
指令原型	int32 __stdcall ZAux_Direct_SetLspeed(ZMC_HANDLE handle, int iaxis, float fValue)
指令说明	设置起始速度，单位 units/s。
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。 fValue | 设置的起始速度。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	轴基本参数的设置获取（带起始速度与 S 曲线）
详细说明	1.当多轴运动时，设置主轴作为插补运动的起始速度。 2.当需要追求效率时，可以考虑设置起始速度。

指令 34	**ZAux_Direct_GetLspeed**
指令原型	int32 __stdcall ZAux_Direct_GetLspeed(ZMC_HANDLE handle, int iaxis, float *pfValue)
指令说明	读取起始速度，单位 units/s。
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。
输出参数	参数名 | 描述 pfValue | 返回的起始速度。

返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	轴基本参数的设置获取（带起始速度与 S 曲线）
详细说明	1.当多轴运动时，设置主轴作为插补运动的起始速度。 2.当需要追求效率时，可以考虑设置起始速度。

指令 35	ZAux_Direct_SetSramp
指令原型	int32 __stdcall ZAux_Direct_SetSramp(ZMC_HANDLE handle, int iaxis, float fValue)
指令说明	设置 S 曲线时间，单位 ms，0 时则表示梯形加减速。
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。 fValue | 设置的 S 曲线值。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	轴基本参数的设置获取（带起始速度与 S 曲线）
详细说明	设置后加减速过程会延长相应的时间 设置最大值为原来 T 型加速度的加速时间。

指令 36	ZAux_Direct_GetSramp
指令原型	int32 __stdcall ZAux_Direct_GetSramp(ZMC_HANDLE handle, int iaxis, float *pfValue)
指令说明	读取 S 曲线时间设置。
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。
输出参数	参数名 | 描述 pfValue | 返回的 S 曲线时间值。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	轴基本参数的设置获取（带起始速度与 S 曲线）
详细说明	1.设置后加减速过程会延长相应的时间 2.设置最大值为原来 T 型加速度的加速时间

指令 37	ZAux_Direct_GetMspeed
指令原型	int32 __stdcall ZAux_Direct_GetMspeed(ZMC_HANDLE handle, int iaxis, float *pfValue)
指令说明	读取当前实时的轴反馈速度，单位 units。
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。
输出参数	参数名 | 描述 pfValue | 返回的实时的轴反馈速度。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	float fValue; ret = ZAux_Direct_GetMspeed(handle, 0, &fValue); //读取轴 0 当前反馈速度。handle：控制器连续句柄
详细说明	/

指令 38	ZAux_Direct_GetVpSpeed
指令原型	int32 __stdcall ZAux_Direct_GetVpSpeed (ZMC_HANDLE handle, int iaxis, float *pfValue)
指令说明	读取控制器发送的轴实时运动速度，单位 units/s。
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。
输出参数	参数名 | 描述 pfValue | 返回的控制器发送的轴实时速度。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	float fValue; ret = ZAux_Direct_GetVpSpeed(handle, 0, &fValue); //读取轴 0 当前命令运动速度。handle：控制器连续句柄
详细说明	1.当多轴运动时，主轴返回的是插补运动的速度，不是主轴的分速度。 2.非主轴返回的是相应的分速度，与反馈速度效果一致。 3.规划速度在默认情况下是为显示多轴合成速度设计的，是没有负值的，除非把 SYSTEM_ZSET(RTBasic 指令，PC 可以用这个函数：ZAux_Direct_SetParam(句柄, “SYSTEM_ZSET”,轴号, 设置值)）的 bit0 的

	值设置为 0，就可以用来显示单轴的命令速度，可正可负。

指令 39	ZAux_Direct_GetIfIdle
指令原型	int32 __stdcall ZAux_Direct_GetIfIdle(ZMC_HANDLE handle, int iaxis, int *piValue)
指令说明	读取当前轴运动是否在运动中
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。
输出参数	参数名 | 描述 piValue | 返回的状态： 0 –运动中，-1-未运动
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	多轴直线插补
详细说明	当轴关联为机械手，CONNFRAME 逆解时，关节轴一直返回 0；CONNREFRAME 正解时，虚拟轴一直返回 0。

指令 40	ZAux_Direct_GetAxisStatus
指令原型	int32 __stdcall ZAux_Direct_GetAxisStatus(ZMC_HANDLE handle, int iaxis, int *piValue)
指令说明	读取当前轴的状态。 按十进制显示数值，按二进制对应位判断状态。
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。
输出参数	参数名 | 描述 pfValue | 返回状态值， 对应的位表示不同状态 （具体看详细说明）
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	轴基本参数的设置获取（带起始速度与 S 曲线）

详细说明

返回状态值，对应的位表示不同状态。

位	说明	打印值	
1	随动误差超限告警	2	2h
2	与远程轴通讯出错	4	4h
3	远程驱动器报错	8	8h
4	正向硬限位	16	10h
5	反向硬限位	32	20h
6	找原点中	64	40h
7	HOLD 速度保持信号输入	128	80h
8	随动误差超限出错	256	100h
9	超过正向软限位	512	200h
10	超过负向软限位	1024	400h
11	CANCEL 执行中	2048	800h
12	脉冲频率超过 MAX_SPEED 限制需要修改降速或修改 MAX_SPEED	4096	1000h
14	机械手指令坐标错误	16384	4000h
18	电源异常	262144	40000h
20	轴速度保护	1048576	100000h
21	运动中触发特殊运动指令失败	2097152	200000h
22	告警信号输入	4194304	400000h
23	轴进入了暂停状态	8388608	800000h

指令 41 ZAux_Direct_GetAxisStopReason

指令原型

int32 __stdcall ZAux_Direct_GetAxisStopReason(ZMC_HANDLE handle, int iaxis, int *piValue)

指令说明

读取轴历史状态。

输入参数

参数名	描述
handle	连接标识。
iaxis	轴号。

输出参数

参数名	描述
pfValue	返回状态值， 对应的位表示不同状态 （具体看详细说明）

返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	float fValue; ZAux_Direct_GetAxisStopReason(handle, 0, &fValue); //读取轴 0 轴停止的原因。handle：控制器连续句柄
详细说明	返回状态值，对应的位表示不同状态。

位	说明	打印值	
1	随动误差超限告警	2	2h
2	与远程轴通讯出错	4	4h
3	远程驱动器报错	8	8h
4	正向硬限位	16	10h
5	反向硬限位	32	20h
6	找原点中	64	40h
7	HOLD 速度保持信号输入	128	80h
8	随动误差超限出错	256	100h
9	超过正向软限位	512	200h
10	超过负向软限位	1024	400h
11	CANCEL 执行中	2048	800h
12	脉冲频率超过 MAX_SPEED 限制需要修改降速或修改 MAX_SPEED	4096	1000h
14	机械手指令坐标错误	16384	4000h
18	电源异常	262144	40000h
20	轴速度保护	1048576	100000h
21	运动中触发特殊运动指令失败	2097152	200000h
22	告警信号输入	4194304	400000h
23	轴进入了暂停状态	8388608	800000h

指令 42	**ZAux_GetModbusDpos**
指令原型	int32 __stdcall ZAux_GetModbusDpos (ZMC_HANDLE handle , int imaxaxises, float * pValueList);
指令说明	快速读取多个轴当前的 DPOS。Modbus 寄存器方式
输入参数	

参数名	描述
handle	连接标识。
imaxaxises	轴数量。

输出参数	参数名 | 描述 pValueList | 读取的 DPOS 坐标值，从轴 0 开始。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	参数获取及配置
详细说明	默认从轴 0 开始读取，读取速度最快的方式。

指令 43	**ZAux_GetModbusMpos**
指令原型	int32 __stdcall ZAux_GetModbusMpos (ZMC_HANDLE handle , int imaxaxises, float * pValueList);
指令说明	快速读取多个轴当前的 MPOS。Modbus 寄存器方式
输入参数	参数名 | 描述 handle | 连接标识。 imaxaxises | 轴数量。
输出参数	参数名 | 描述 pValueList | 读取的反馈坐标值，从轴 0 开始。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	参数获取及配置
详细说明	默认从轴 0 开始读取，读取速度最快的方式。

指令 44	**ZAux_GetModbusCurSpeed**
指令原型	int32 __stdcall ZAux_GetModbusCurSpeed (ZMC_HANDLE handle , int imaxaxises, float * pValueList);
指令说明	快速读取多个轴当前的速度。Modbus 寄存器方式
输入参数	参数名 | 描述 handle | 连接标识。 imaxaxises | 轴数量。
输出参数	参数名 | 描述 pValueList | 读取的当前速度，从轴 0 开始。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	参数获取及配置
详细说明	默认从轴 0 开始读取，读取速度最快的方式。

<table>
<tr><td>指令 45</td><td>ZAux_Direct_SetAxisAddress</td></tr>
<tr><td>指令原型</td><td>int32 __stdcall ZAux_Direct_SetAxisAddress (ZMC_HANDLE handle, int iaxis, int iValue)</td></tr>
<tr><td>指令说明</td><td>设置选择轴的轴地址。</td></tr>
<tr><td>输入参数</td><td><table><tr><td>参数名</td><td>描述</td></tr><tr><td>handle</td><td>连接标识。</td></tr><tr><td>iaxis</td><td>轴号。</td></tr><tr><td>iValue</td><td>配置的地址。</td></tr></table></td></tr>
<tr><td>输出参数</td><td>/</td></tr>
<tr><td>返回值</td><td>成功返回值为 0，非 0 详见错误码说明。</td></tr>
<tr><td>指令示例</td><td>本地脉冲轴号重映射</td></tr>
<tr><td>详细说明</td><td>1.ZCAN 扩展轴时，扩展板上带 8 位拨码开关（V1.3 以上硬件版本）。
受总线带宽限制，ZCAN 扩展轴不要设置超过 2 个。
必须先设置 AXIS_ADDRESS，再设置 ZCAN 扩展轴 ATYPE 类型，修改后必须重新设置 ATYPE。见例一。<table><tr><td>位 1-4</td><td>CAN 地址拨码，组合值 0-15</td></tr><tr><td>位 5-6</td><td>CAN 速度拨码，不同组合值速度不同</td></tr><tr><td>位 7</td><td>特殊功能预留</td></tr><tr><td>位 8</td><td>120 欧姆电阻拨码，拨 ON 时电阻接通</td></tr></table>规则：
ZAux_Direct_SetAxisAddress(handle，轴号，(32*0)+ID)
//扩展板的本地轴接口 0
ZAux_Direct_SetAxisAddress(handle，轴号，(32*1)+ID)
//扩展板的本地轴接口 1
2.总线驱动器映射轴号，将连接的驱动器按编号一一映射。
驱动器编号根据接线顺序排列，编号从 0 开始到 EtherCAT 驱动器个数减 1。
驱动器编号与设备号不同，设备号包括 ECAT 接口所有连接的设备，驱动器编号只算连接的驱动器。
必须先设置 AXIS_ADDRESS，再设置 ECAT 轴 ATYPE 类型，修改后必须重新设置 ATYPE。<table><tr><td>位 0-15</td><td>驱动器编号加 1，0-自动指定</td></tr><tr><td>位 16-31</td><td>SLOT 编号（多槽位时）</td></tr></table>规则：
ZAux_Direct_SetAxisAddress(handle，轴号，(槽位号<<16)+驱动器编号+1)
3.本地脉冲轴号重映射，4 系列控制器支持本地脉冲或编码器轴号重新映</td></tr>
</table>

	射，固件 160608 以上版本支持。 重新映射时注意先把原本地脉冲轴设置为虚拟轴。修改后必须重新设置 ATYPE。见指令示例：本地脉冲轴号重映射。 位 0-15：映射的本地脉冲轴号 位 16-31：高 16 位全设为 1（等同于十进制下高 16 位= -1） 规则： ZAux_Direct_SetAtype(handle，重映射的轴号,0) //设置轴类型为 0，低版本不设置会报错 ZAux_Direct_SetAtype(handle，要修改的本地脉冲轴号,0) //设置轴类型为 0，低版本不设置会报错 ZAux_Direct_SetAxisAddress(handle，重映射的轴号, (-1<<16) +要修改的本地脉冲轴号) ZAux_Direct_SetAtype(handle，重映射的轴号,1 或者 7)

指令 46	ZAux_Direct_GetAxisAddress
指令原型	int32 __stdcall ZAux_Direct_GetAxisAddress (ZMC_HANDLE handle, int iaxis, int *piValue)
指令说明	读取选择轴的轴地址。
输入参数	参数名：handle — 连接标识。 iaxis — 轴号。
输出参数	参数名：piValue — 返回轴地址。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	本地脉冲轴号重映射
详细说明	/

指令 47	ZAux_Direct_SetAxisEnable
指令原型	int32 __stdcall ZAux_Direct_SetAxisEnable(ZMC_HANDLE handle, int iaxis, int iValue);
指令说明	设置轴的使能。
输入参数	共有 2 个输入参数，见下方说明。
handle	连接句柄。
axis	槽位号

iValue	状态：0-关闭 1-打开
输出参数	一个参数
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	EthrCat 总线轴的使能

指令 48	ZAux_Direct_GetAxisEnable
指令原型	int32 __stdcall ZAux_Direct_GetAxisEnable (ZMC_HANDLE handle , int iaxis, int * piValue);
指令说明	读取 Ethrcat 轴的使能状态。
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。
输出参数	参数名 | 描述 piValue | 返回的使能状态。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	EthrCat 总线轴的使能
详细说明	/

指令 49	ZAux_Direct_SetInvertStep
指令原型	int32 __stdcall ZAux_Direct_SetInvertStep(ZMC_HANDLE handle, int iaxis, int iValue)
指令说明	设置脉冲输出模式。
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。 iValue | 模式选择，按位设置，缺省 0，（详看详细说明）
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	参数获取及配置
详细说明	模式选择，缺省 0 低 8 位（位 0-位 7）表示的模式值如下： 0-3 脉冲方向模式，脉冲线+方向线 4-7 双脉冲方式（或称 CW/CCW），正向脉冲线+负向脉冲线

8-9 AB 输出(部分控制器定制)

各个模式对应的电平如下：

模式值	描述	松下设置参考		三菱设置参考
		Pr0.06	Pr0.07	PA13
0	脉冲/方向（脉冲正逻辑）（正向）	0	3	××01h
1	脉冲/方向（脉冲负逻辑）（正向）	/	/	××11h
2	脉冲/方向（脉冲正逻辑）（负向）	1	3	××01h
3	脉冲/方向（脉冲负逻辑）（负向）	/	/	××11h
4	双脉冲（方向负逻辑）（正向）	/	/	××10h
5	双脉冲（方向负逻辑）（负向）	/	/	××10h
6	双脉冲（方向正逻辑）（正向）	1	1	××00h(默认)
7	双脉冲（方向正逻辑）（负向）	0(默认)	1(默认)	××00h(默认)

高 8 位（位 8-位 15）表示方向变化保护时间，单位微秒：0-255

常用模式为 0、2、6、7。

如果模式设定不正确，步进马达可能会在换向时丢失一步的位置，当不确定步进马达的设置时，可以设置 100 微秒左右的保护时间。

指令 50	**ZAux_Direct_GetInvertStep**
指令原型	int32 __stdcall ZAux_Direct_GetInvertStep(ZMC_HANDLE handle, int iaxis, int *piValue)
指令说明	读取脉冲输出模式。

输入参数

参数名	描述
handle	连接标识。
iaxis	轴号。

输出参数

参数名	描述
piValue	返回当前模式。

返回值：成功返回值为 0，非 0 详见**错误码**说明。

指令示例：参数获取及配置

详细说明

模式对应的值如下：

模式值	描述	松下设置参考		三菱设置参考
		Pr0.06	Pr0.07	PA13

0	脉冲/方向（脉冲正逻辑）（正向）	0	3	××01h
1	脉冲/方向（脉冲负逻辑）（正向）	/	/	××11h
2	脉冲/方向（脉冲正逻辑）（负向）	1	3	××01h
3	脉冲/方向（脉冲负逻辑）（负向）	/	/	××11h
4	双脉冲（方向负逻辑）（正向）	/	/	××10h
5	双脉冲（方向负逻辑）（负向）	/	/	××10h
6	双脉冲（方向正逻辑）（正向）	1	1	××00h (默认)
7	双脉冲（方向正逻辑）（负向）	0(默认)	1(默认)	××00h (默认)

高 8 位（位 8-位 15）表示方向变化保护时间，单位微秒：0-255

常用模式为 0、2、6、7。

如果模式设定不正确，步进马达可能会在换向时丢失一步的位置，当不确定步进马达的设置时，可以设置 100 微秒左右的保护时间。

指令 51	ZAux_Direct_GetMaxSpeed
指令原型	int32 __stdcall ZAux_Direct_GetMaxSpeed (ZMC_HANDLE handle , int iaxis, int * piValue);
指令说明	读取脉冲输出最高频率。

输入参数

参数名	描述
handle	连接标识。
iaxis	轴号。

输出参数

参数名	描述
piValue	返回的脉冲频率。

返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	参数获取及配置

详细说明

1.一旦发现脉冲频率超过此设置值会强制限制发送的脉冲频率不超过 MAX SPEED，并且产生脉冲超限报警。

2.对编码器轴，设置值低于 500K 时会启用编码器滤波，设置值高于 1M 时会取消编码器滤波设置。默认值为 1000000（老固件的默认脉冲频率为 500000）。

3.使用直线电机速度较快时，一般容易脉冲频率超限，可把数值适当设置大点。

指令 52	ZAux_Direct_SetMaxSpeed

指令原型	int32 __stdcall ZAux_Direct_SetMaxSpeed (ZMC_HANDLE handle , int iaxis, int iValue);
指令说明	设置的最高脉冲频率。
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。 iValue | 设置的最大脉冲频率
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	参数获取及配置
详细说明	1.一旦发现脉冲频率超过此设置值会强制限制发送的脉冲频率不超过 MAX SPEED，并且产生脉冲超限报警。 2.对编码器轴，设置值低于 500K 时会启用编码器滤波，设置值高于 1M 时会取消编码器滤波设置。默认值为 1000000（老固件的默认脉冲频率为 500000）。 3.使用直线电机速度较快时，一般容易脉冲频率超限，可把数值适当设置大点。

指令 53	ZAux_Direct_GetMtype
指令原型	int32 __stdcall ZAux_Direct_GetMtype(ZMC_HANDLE handle, int iaxis, int *piValue)
指令说明	读取当前运动的指令类型。
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。
输出参数	参数名 | 描述 piValue | 返回的运动指令类型
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	参数获取及配置
详细说明	运动指令类型值对应类型如下： MTYPE | 运动指令类型 0 | IDLE (没有运动) 1 | MOVE（单轴直线或者插补直线运动） 2 | MOVEABS（绝对值单轴直线或者绝对插补直线运动）

3	MHELICAL（圆心螺旋运动）
4	MOVECIRC（圆弧插补）
5	MOVEMODIFY（修改运动位置）
6	MOVESP（SP 速度的单轴直线或者 SP 速度的插补直线运动）
7	MOVEABSSP（SP 速度的绝对值单轴直线/SP 速度的绝对插补直线运动）
8	MOVECIRCSP（SP 速度的圆弧插补）
9	MHELICALSP（SP 速度的圆心螺旋运动）
10	FORWARD，VMOVE(1)（正向持续运动）
11	REVERSE，VMOVE(-1)（负向持续运动）
12	DATUMING (回零运动中)
13	CAM (凸轮表运动)
14	FWD_JOG (映射正向 JOG 运动)
15	REV_JOG (映射负向 JOG 运动)
20	CAMBOX (跟随凸轮表运动)
21	CONNECT (同步运动)
22	MOVELINK（自动凸轮运动）
23	CONNPATH (同步运动 2，矢量类型的)
25	MOVESLINK（自动凸轮运动 2）
26	MOVESPIRAL(渐开线圆弧)
27	MECLIPSE ， MECLIPSEABS ， MECLIPSESP ， MECLIPSEABSSP（椭圆运动）
28	MOVE_OP/MOVE_OP2，MOVE_TABLE，MOVE_PWM，MOVE_TASKMOVE_PARA ， MOVE_ASYNMOVE ，MOVE_AOUT（缓冲 IO/缓冲寄存器操作等）
29	MOVE_DELAY，MOVE_WAIT，MOVE_SYNMOVE（缓冲延时）
31	MSPHERICAL，MSPHERICALSP（空间圆弧）
32	MOVE_PT（单位时间内运动距离）
33	CONNFRAME（机械手逆解运动）
34	CONNREFRAME（机械手正解运动）

指令 54	**ZAux_Direct_GetNtype**
指令原型	int32 __stdcall ZAux_Direct_GetNtype(ZMC_HANDLE handle, int iaxis, int *piValue)

指令说明	读取当前运动的下一条运动指令类型。

输入参数

参数名	描述
handle	连接标识。
iaxis	轴号。

输出参数

参数名	描述
piValue	返回的运动指令类型

返回值：成功返回值为 0，非 0 详见**错误码**说明。

指令示例：参数获取及配置

详细说明

运动指令类型值对应类型如下：

MTYPE	运动指令类型
0	IDLE (没有运动)
1	MOVE（单轴直线或者插补直线运动）
2	MOVEABS（绝对值单轴直线或者绝对插补直线运动）
3	MHELICAL（圆心螺旋运动）
4	MOVECIRC（圆弧插补）
5	MOVEMODIFY（修改运动位置）
6	MOVESP（SP 速度的单轴直线或者 SP 速度的插补直线运动）
7	MOVEABSSP（SP 速度的绝对值单轴直线/SP 速度的绝对插补直线运动）
8	MOVECIRCSP（SP 速度的圆弧插补）
9	MHELICALSP（SP 速度的圆心螺旋运动）
10	FORWARD，VMOVE(1)（正向持续运动）
11	REVERSE，VMOVE(-1)（负向持续运动）
12	DATUMING (回零运动中)
13	CAM (凸轮表运动)
14	FWD_JOG (映射正向 JOG 运动)
15	REV_JOG (映射负向 JOG 运动)
20	CAMBOX (跟随凸轮表运动)
21	CONNECT (同步运动)
22	MOVELINK （自动凸轮运动）
23	CONNPATH (同步运动 2，矢量类型的)
25	MOVESLINK（自动凸轮运动 2）
26	MOVESPIRAL(渐开线圆弧)
27	MECLIPSE ， MECLIPSEABS ， MECLIPSESP ，

	MECLIPSEABSSP（椭圆运动）
28	MOVE_OP/MOVE_OP2，MOVE_TABLE，MOVE_PWM，MOVE_TASKMOVE_PARA ， MOVE_ASYNMOVE ，MOVE_AOUT（缓冲 IO/缓冲寄存器操作等）
29	MOVE_DELAY，MOVE_WAIT，MOVE_SYNMOVE（缓冲延时）
31	MSPHERICAL，MSPHERICALSP（空间圆弧）
32	MOVE_PT（单位时间内运动距离）
33	CONNFRAME（机械手逆解运动）
34	CONNREFRAME（机械手正解运动）

指令 55	**ZAux_Direct_SetOffpos**
指令原型	int32 __stdcall ZAux_Direct_SetOffpos (ZMC_HANDLE handle , int iaxis, int fValue);
指令说明	设置修改偏移位置。
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。 fValue | 设置的偏移值。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	轴坐标设置
详细说明	相对偏移修改所有的坐标，不会对已运行/进入缓冲区的运动产生影响。当修改完成后，OFFPOS 还原为 0。

指令 56	**ZAux_Direct_GetOffpos**
指令原型	int32 __stdcall ZAux_Direct_GetOffpos (ZMC_HANDLE handle , int iaxis, int *piValue);
指令说明	读取修改偏移位置。
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。

输出参数	参数名 | 描述 piValue | 获取的偏移值。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	轴坐标设置
详细说明	当修改完成后，OFFPOS 还原为 0。

指令 57	ZAux_Direct_Defpos
指令原型	int32 __stdcall ZAux_Direct_Defpos(ZMC_HANDLE handle, int iaxis, float pfDpos)
指令说明	根据定义当前位置为一个新的绝对位置值。
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。 pfDpos | 设置的距离坐标。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	轴坐标设置
详细说明	不会对已运行/进入缓冲区的运动产生影响。

指令 58	ZAux_Direct_GetFe
指令原型	int32 __stdcall ZAux_Direct_GetFe (ZMC_HANDLE handle , int iaxis, float*pfValue);
指令说明	读取随动误差。
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。
输出参数	参数名 | 描述 pfValue | 获取的偏差值。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	轴坐标设置
详细说明	随动误差=控制器规划位置-编码器反馈位置值

指令 59	ZAux_Direct_GetFeLimit
指令原型	int32 __stdcall ZAux_Direct_GetFeLimit (ZMC_HANDLE handle, int iaxis, float*pfValue);
指令说明	读取最大允许时的随动误差。（需使用闭环固件才生效，预留功能）
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。
输出参数	参数名 | 描述 pfValue | 最大允许的随动误差值。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	float pfValue; ZAux_Direct_GetFeLimit(handle, 0, &pfValue); //读取轴 0 最大允许时的随动误差。handle：控制器连续句柄
详细说明	/

指令 60	ZAux_Direct_SetFeLimit
指令原型	int32 __stdcall ZAux_Direct_SetFeLimit (ZMC_HANDLE handle , int iaxis, float fValue);
指令说明	设置最大允许的随动误差。（需使用闭环固件才生效，预留功能）
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。 fValue | 最大允许的随动误差值。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	float fValue=100; ZAux_Direct_SetFeLimit(handle, 0, fValue); //设置轴 0 最大允许时的随动误差=100。handle：控制器连续句柄
详细说明	/

指令 61	ZAux_Direct_GetFeRange
指令原型	int32 __stdcall ZAux_Direct_GetFeRange (ZMC_HANDLE handle , int iaxis, int*pfValue);
指令说明	读取报警时的随动误差。（需使用闭环固件才生效，预留功能）

输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。
输出参数	参数名 | 描述 pfValue | 报警时的随动误差值。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	Int iValue; ZAux_Direct_GetFeRanget(handle, 0, & iValue); //读取轴 0 报警时的随动误差。handle：控制器连续句柄
详细说明	/

指令 62	**ZAux_Direct_SetFeRange**
指令原型	int32 __stdcall ZAux_Direct_SetFeRange (ZMC_HANDLE handle , int iaxis, float fValue);
指令说明	设置报警时随动误差值。（需使用闭环固件才生效，预留功能）
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。 fValue | 报警时的随动误差值。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	Int iValue=100; ZAux_Direct_SetFeRanget(handle, 0, iValue); //设置轴 0 报警时随动误差值=100。handle：控制器连续句柄
详细说明	/

指令 63	**ZAux_Direct_GetAllAxisPara**
指令原型	int32 __stdcall ZAux_Direct_GetAllAxisPara(ZMC_HANDLE handle, const char *sParam,int imaxaxis,float *pfValue)
指令说明	一次性读取多个轴轴参数。

输入参数	<table><tr><td>参数名</td><td>描述</td></tr><tr><td>handle</td><td>连接标识。</td></tr><tr><td>sParam</td><td>Baisc 语法参数的字符串名称。</td></tr><tr><td>imaxaxis</td><td>轴数量。</td></tr></table>
输出参数	<table><tr><td>参数名</td><td>描述</td></tr><tr><td>pfValue</td><td>返回参数值。</td></tr></table>
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	float fAtype[8]; ZAux_Direct_GetAllAxisPara(handle,"ATYPE",8,fAtype); //获取 0-7 轴的轴类型
详细说明	/

指令 64	**ZAux_Direct_GetAllAxisInfo**
指令原型	int32 __stdcall ZAux_Direct_GetAllAxisInfo(ZMC_HANDLE handle,int imaxaxis,int *IdleStatus,float *DposStatus,float *MposStatus, int *AxisStatus)
指令说明	读取多个轴基本轴状态。 按十进制显示数值，按二进制对应位判断状态。
输入参数	<table><tr><td>参数名</td><td>描述</td></tr><tr><td>handle</td><td>连接标识。</td></tr><tr><td>imaxaxis</td><td>轴数量。</td></tr></table>
输出参数	<table><tr><td>参数名</td><td>描述</td></tr><tr><td>IdleStatus</td><td>轴运行状态。</td></tr><tr><td>DposStatus</td><td>轴命令坐标。</td></tr><tr><td>MposStatus</td><td>轴反馈坐标。</td></tr><tr><td>AxisStatus</td><td>轴状态。（详看详细说明）</td></tr></table>
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	int iIdle[8]; Float iDpos[8]; Float iMpos[8]; int iStatus[8]; ZAux_Direct_GetAllAxisPara(handle,,8,iIdle,iDpos,iMpos,iStatus); //获取 0-7 轴的轴基本轴状态
详细说明	轴状态，按位对应：

位	说明	打印值	
1	随动误差超限告警	2	2h
2	与远程轴通讯出错	4	4h
3	远程驱动器报错	8	8h
4	正向硬限位	16	10h
5	反向硬限位	32	20h
6	找原点中	64	40h
7	HOLD速度保持信号输入	128	80h
8	随动误差超限出错	256	100h
9	超过正向软限位	512	200h
10	超过负向软限位	1024	400h
11	CANCEL执行中	2048	800h
12	脉冲频率超过MAX_SPEED限制.需要修改降速或修改MAX_SPEED.	4096	1000h
14	机械手指令坐标错误	16384	4000h
18	电源异常	262144	40000h
21	运动中触发特殊运动指令失败	2097152	200000h
22	告警信号输入	4194304	400000h
23	轴进入了暂停状态	8388608	800000h

指令 65	ZAux_Direct_SetParam
指令原型	int32 __stdcall ZAux_Direct_SetParam(ZMC_HANDLE handle,const char *sParam,int iaxis, float fset)
指令说明	通用的参数修改函数 sParm 填写参数名称。
输入参数	参数名 / 描述 handle / 连接标识。 sParam / Baisc 语法参数的字符串名称。 iaxis / 轴号。 fset / 参数值。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	字符串方式设置获取基本参数
详细说明	上位机使用 Basic 指令方式获取控制器数值，常用于上位机未封装的 Basic 指令，使用该函数代替部分指令封装，可以快速获取对应值

指令 66	ZAux_Direct_GetParam
指令原型	int32 __stdcall ZAux_Direct_GetParam(ZMC_HANDLE handle,const

	char *sParam, int iaxis, float *pfValue)
指令说明	参数 通用的参数读取函数 sparam 填写参数名称
输入参数	参数名 | 描述 handle | 连接标识。 sParam | Baisc 语法参数的字符串名称。 iaxis | 轴号。
输出参数	参数名 | 描述 pfValue | 返回参数值。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	字符串方式设置获取基本参数
详细说明	上位机使用 Basic 指令方式获取控制器数值，常用于上位机未封装的 Basic 指令，使用该函数代替部分指令封装，可以快速获取对应值

指令 67	**ZAux_Direct_GetEndMove**
指令原型	int32 __stdcall ZAux_Direct_GetEndMove(ZMC_HANDLE handle, int iaxis, float *pfValue)
指令说明	读取当前运动的最终位置，单位 units。
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。
输出参数	参数名 | 描述 pfValue | 返回参数值。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	Float fValue; ZAux_Direct_GetEndMove(handle,0,&fValue); //获取轴 0 当前运动的最终位置.
详细说明	对 VMOVE，DATUM 等非固定距离的运动，ENDMOVE 并不准确，或时刻在变化。

指令 68	**ZAux_Direct_GetEndMoveBuffer**
指令原型	int32 __stdcall ZAux_Direct_GetEndMoveBuffer(ZMC_HANDLE handle, int iaxis, float *pfValue)

指令说明	读取当前和缓冲中运动的最终位置，可以用于相对绝对转换，单位 units。
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。
输出参数	参数名 | 描述 pfValue | 返回轴当前和缓冲的所有运动的最终目标位置。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	Float fValue; **ZAux_Direct_GetEndMoveBuffer**(handle,0,&fValue); //获取轴 0 缓冲中运动的最终位置
详细说明	对 VMOVE，DATUM 等非固定距离的运动，ENDMOVE_BUFFER 并不准确，或时刻在变化。 使用了循环坐标指令 REP_OPTION 后，ENDMOVE_BUFFER 根据 REP_OPTION 模式按 REP_DIST 设置值依次递减，即最小精度是 REP_DIST（模式 1）或 2 倍 REP_DIST（模式 0）。

指令 69	**ZAux_Direct_GetLoaded**
指令原型	int32 __stdcall ZAux_Direct_GetLoaded (ZMC_HANDLE handle , int iaxis,int *pfValue);
指令说明	读取是否还有运动缓冲，除当前运动
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。
输出参数	参数名 | 描述 pfValue | 返回值： -1：没有剩余运动缓冲 0：还有剩余运动缓冲
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	Int iValue; ZAux_Direct_GetLoaded (handle,0,&iValue); //获取轴 0 是否还有运动缓冲
详细说明	此指令无法判断轴是否停止，判断轴停止使用 IDLE。

指令 70	ZAux_Direct_GetMovesBuffered
指令原型	int32 __stdcall ZAux_Direct_GetMoveBuffered (ZMC_HANDLE handle , int iaxis, int * piValue);
指令说明	读取当前被缓冲起来的运动个数。
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。
输出参数	参数名 | 描述 pfValue | 缓冲运动数
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	Int iValue; ZAux_Direct_GetMoveBuffered(handle,0,&iValue); //获取轴 0 当前已缓冲数
详细说明	不要用总缓冲空间数减去 MOVES_BUFFERED 这个参数来判断是否有剩余的缓冲空间，特殊的运动指令可能会占用多个缓冲空间，采用 REMAIN_BUFFER 状态函数来判断更准确。

指令 71	ZAux_Direct_GetMoveCurmark
指令原型	int32 __stdcall ZAux_Direct_GetMoveCurmark (ZMC_HANDLE handle , int iaxis, int * piValue);
指令说明	读取当前正在运动指令的 MOVE_MARK 标号
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。
输出参数	参数名 | 描述 piValue | 当前 Mark 标号
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	Int iValue; ZAux_Direct_GetMoveCurmark(handle,0,&iValue); //获取轴 0 当前运动的 Mark 标号
详细说明	/

指令 72	ZAux_Direct_SetMoveMark
指令原型	int32 __stdcall ZAux_Direct_SetMoveMark (ZMC_HANDLE handle , int iaxis, int iValue);
指令说明	设置下一条要调用的运动指令的 MARK 标号。这个标号会和运动指令一起写入运动缓冲。每调用一条运动指令，MOVE_MARK 会自动加一。
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。 piValue | 设置的 MARK 值。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	ZAux_Direct_GetMoveCurmark(handle,0,1); //设置轴 0 下条运动的 Mark 标号为 1
详细说明	/

指令 73	ZAux_Direct_GetRemain
指令原型	int32 __stdcall ZAux_Direct_GetRemain(ZMC_HANDLE handle,int iaxis, float *pfValue)
指令说明	读取当前运动剩余距离，单位 units。
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。
输出参数	参数名 | 描述 pfValue | 返回的剩余距离
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	Float fValue; ZAux_Direct_GetRemain(handle,0,&fValue); //获取轴 0 当前运动剩余距离
详细说明	/

指令 74	ZAux_Direct_GetRemain_LineBuffer
指令原型	int32 __stdcall ZAux_Direct_GetRemain_LineBuffer (ZMC_HANDLE handle , int iaxis, int *piValue);

指令说明	轴剩余的缓冲，按直线段来计算，REMAIN_BUFFER 为唯一一个可以加 AXIS 并用 Zaux_DirectCommand 获取的。
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。
输出参数	参数名 | 描述 piValue | 剩余的直线缓冲数量
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	Int iValue; ZAux_Direct_GetRemain_LineBuffer(handle,0,&iValue); //获取轴 0 当前剩余的直线运动缓冲
详细说明	/

指令 75	**ZAux_Direct_GetRemain_Buffer**
指令原型	int32 __stdcall ZAux_Direct_GetRemain_Buffer(ZMC_HANDLE handle, int iaxis, int *piValue)
指令说明	读取轴剩余的运动缓冲，按剩余最复杂空间圆弧缓冲获取。
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。
输出参数	参数名 | 描述 piValue | 剩余的空间圆弧缓冲数量
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	获取剩余运动缓冲数量
详细说明	/

指令 76	**ZAux_Direct_GetVectorBuffered**
指令原型	int32 __stdcall ZAux_Direct_GetVectorBuffered(ZMC_HANDLE handle , int iaxis, float *pfValue);
指令说明	返回轴当前运动和缓冲运动还未完成的距离，units 单位。 对多轴插补是矢量距离。

输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。
输出参数	参数名 | 描述 piValue | 当前运动和缓冲运动已完成的距离
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	Float fValue; ZAux_Direct_GetMoveCurmark(handle,0,&fValue); //获取轴 0 当前运动和缓冲运动已完成的距离。
详细说明	对多轴插补是矢量距离。

指令 77	**ZAux_Direct_MoveOp**
指令原型	int32 __stdcall ZAux_Direct_MoveOp(ZMC_HANDLE handle, int iaxis, int ioutnum, int ivalue);
指令说明	把输出指定写到运动缓冲里面。
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。 ioutnum | IO 编号。 ivalue | IO 状态。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	例 1： ZAux_Direct_Single_Move(handle, 0, 100);//轴 0 正向运动 100 ZAux_Direct_MoveOp(handle, 0,1, 1);//轴 0 正向运动 100 后把 out1 的状态置为 1 例 2:连续位置锁存
详细说明	这个指令 LOAD 执行时不做任何运动，只操作输出口。语法同 OP 指令。普通模式，误差在一个扫描周期，所有控制器都可使用。精准位置输出模式，误差 1 微秒以内，4 系列产品，20170421 以上版本支持。 1.只有支持硬件比较输出的 OP 口才支持精准输出功能。 2.每个生效的精准输出 MOVE_OP 需要间隔 1 个周期时间才能继续生效，此间隔内新的 MOVE_OP 自动使用普通方式，超过此间隔，新 MOVE_OP 可继续生效。连续的 MOVE_OP，因为没有间隔时间，只有第一个生效。

	（某些控制器可以多个精准输出同时触发，具体查看控制器硬件手册说明，例如 ZMC420SCAN 前 8 个输出口支持精准输出，而且每个输出口可以同时使用精准输出） 3.在控制器支持 OP 口独立的情况下，不同 OP 口就算多个轴 MOVE_OP 精准输出也不冲突，在 OP 口精准输出功能不独立的情况，同时使用精准功能可能冲突。 4.MOVE_OP 精准功能基于 BASE 主轴，多个轴插补时，与 BASE 主轴 ATYPE 类型不一样的从轴，无法保证精准位置输出。

指令 78	ZAux_Direct_MoveOpMulti
指令原型	int32 __stdcall ZAux_Direct_MoveOpMulti(ZMC_HANDLE handle, int iaxis, int ioutnumfirst, int ioutnumend, int ivalue)
指令说明	把多个输出指令写到运动缓冲里面。
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。 ioutnumfirst | 要操作的第一个输出通道。 ioutnumend | 要操作的最后一个输出通道。 ivalue | IO 状态。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	ZAux_Direct_Single_Move(handle, 0, 100);//轴 0 正向运动 100 ZAux_Direct_MoveOpMulti((handle, 0,0,8, 1); //轴 0 正向运动 100 后把 out0-out7 的状态置为 1
详细说明	/

指令 79	ZAux_Direct_MoveOp2
指令原型	int32 __stdcall ZAux_Direct_MoveOp2(ZMC_HANDLE handle, int iaxis, int ioutnum, int ivalue, int iofftimems)
指令说明	把输出指令写到运动缓冲里面。iofftimems 时间后翻转，以产生脉冲输出的效果

输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。 ioutnum | IO 编号。 ivalue | IO 状态。 iofftimems | ms 时间后翻转，以产生脉冲输出的效果。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	飞剪应用
详细说明	这个指令 LOAD 执行时不做任何运动，只操作输出口。 单个轴同一时间只支持一个脉冲输出，第二个 MOVE_OP2 指令会自动关闭前面指令的脉冲。

指令 80	**ZAux_Direct_MoveAout**
指令原型	int32 __stdcall ZAux_Direct_MoveAout(ZMC_HANDLE handle, int iaxis, int ioutnum, float fvalue)
指令说明	在运动缓冲中加入模拟量输出指令。
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。 ioutnum | AOUT 编号。 fvalue | 输出的模拟量值。 12 位刻度值范围 0~4095 对应 0~10V 电压。 16 位刻度值范围 0~65536 对应 0~10V 电压。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	ZAux_Direct_Single_Move(handle, 0, 100);//轴 0 正向运动 100 ZAux_Direct_MoveAout(handle, 0,1, 2048);//轴 0 正向运动 100 后把 DA1 的状态置为 2048
详细说明	这个指令 LOAD 执行时不做任何运动，只修改 AOUT 值。此指令的类型与 MOVE_OP 一致。

指令 81	**ZAux_Direct_MoveDelay**
指令原型	int32 __stdcall ZAux_Direct_MoveDelay(ZMC_HANDLE handle, int

	iaxis, int itimems)
指令说明	运动缓冲里面写入延时。
输入参数	参数名 / 描述 handle / 连接标识。 iaxis / 轴号。 itimems / 延时时间 MS。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	例 1： ZAux_Direct_Single_Move(handle, 0, 100);//轴 0 正向运动 100 ZAux_Direct_MoveDelay(handle, 0,1000);//轴 0 正向运动 100 后延时 1s，才执行下一条缓冲 例 2：连续位置锁存
详细说明	前面的运动指令结束时速度会自动将为 0。

指令 82	**ZAux_Direct_MoveWait**
指令原型	int32 __stdcall ZAux_Direct_MoveWait(ZMC_HANDLE handle,uint32 base_axis,char *paraname,int inum,int Cmp_mode,float fvalue)
指令说明	运动缓冲加入一个条件判断的阻塞，固件 150802 以上版本，或 XPLC160405 以上版本支持。
输入参数	参数名 / 描述 handle / 连接标识。 base_axis / 运动轴轴号 paraname / 参数字符串名称：DPOS MPOS IN OP AIN VPSPEED MSPEED MODBUS_REG MODBUS_IEEE MODBUS_BIT TABLE VECTOR_BUFFERED REMAIN inum / 轴参数编号 Cmp_mode / 比较模式：1：大于等于 0：等于 -1：小于等于 Fvalue / 比较值

返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	例 1： ZAux_Direct_Single_Move(handle, 0, 100);//轴 0 正向运动 100 ZAux_Direct_MoveWait(handle, 0,"IN",1,0,1);//轴 0 正向运动 100 后，等到 IN1 有信号，才执行下一条缓冲 例 2:位置锁存
详细说明	这个指令 LOAD 执行时不做任何运动，只等待指定的条件满足，前面的运动指令结束时速度会自动降为 0。

指令 83	**ZAux_Direct_MoveTask**
指令原型	int32 __stdcall ZAux_Direct_MoveTask(ZMC_HANDLE handle,uint32 base_axis,uint32 tasknum,char *labelname)
指令说明	运动缓冲加入一个 TASK 任务当任务已经启动时，会报错，但不影响程序执行。部分控制卡支持该功能
输入参数	参数名 | 描述 handle | 连接标识。 base_axis | 运动轴轴号。 tasknum | 任务编号。 labelname | 触发的全局子 SUB 函数/标号。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	/
详细说明	这个指令 LOAD 执行时不做任何运动，只启动任务，此指令的类型与 MOVE_OP 一致。 当任务已经启动时，RTSys 内部会报错，但不影响程序执行。

指令 84	**ZAux_Direct_MovePara**
指令原型	int32 __stdcall ZAux_Direct_MovePara(ZMC_HANDLE handle,uint32 base_axis,char *paraname,uint32 iaxis,float fvalue)
指令说明	把参数设置写到运动缓冲里面，需要 20170503 版本以上固件支持。

输入参数	参数名 | 描述 handle | 连接标识。 base_axis | 运动轴轴号。 paraname | 参数名称字符串。 iaxis | 修改参数的轴号。 fvalue | 参数设定值。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	例：轴 0 总运动 300，在运动到 100 的时候，在线把速度修改为 100 后完成总运动。 ZAux_Direct_SetMerge(handle, 0,1);//打开连续插补 ZAux_Direct_Single_Move(handle, 0, 100);//轴 0 正向运动 100 ZAux_Direct_MovePara(handle, 0,"SPEED",0,100);//轴 0 正向运动 100 后，速度改为 100 ZAux_Direct_Single_Move(handle, 0, 200);//轴 0 正向运动 200
详细说明	这个指令 LOAD 执行时不做任何运动，只修改参数。此指令的类型与 MOVE_OP 一致。

指令 85	**ZAux_Direct_MovePwm**
指令原型	int32 __stdcall ZAux_Direct_MovePwm(ZMC_HANDLE handle,uint32 base_axis,uint32 pwm_num,float pwm_duty,float pwm_freq)
指令说明	把 PWM 输出写到运动缓冲里面，需要 20170503 版本以上固件支持。
输入参数	参数名 | 描述 handle | 连接标识。 base_axis | 运动轴轴号。 pwm_num | PWM 口编号。 pwm_duty | 设定的占空比。 pwm_freq | 设定的频率。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	PWM 使用
详细说明	这个指令 LOAD 执行时不做任何运动，只修改参数。此指令的类型与 MOVE_OP 一致。

指令 86	**ZAux_Direct_MoveSynmove**

指令原型	int32 __stdcall ZAux_Direct_MoveSynmove(ZMC_HANDLE handle,uint32 base_axis, uint32 iaxis, float fdist, uint32 ifsp)
指令说明	运动中触发其他轴运动写入缓冲，当前轴等待。
输入参数	参数名 / 描述 handle / 连接标识。 base_axis / 运动轴轴号。 iaxis / 同步的轴号。 fdist / 同步运动的距离。 ifsp / 同步运动是否为 SP 运动。 0：否 1：是
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	例：轴 0 总运动 300，在运动到 100 的时候，触发轴 1 运动 100，等待轴 1 运动完成后，轴 0 继续完成剩下的运动 ZAux_Direct_SetMerge(handle, 0,1);//打开连续插补 ZAux_Direct_Single_Move(handle, 0, 100);//轴 0 正向运动 100 ZAux_Direct_MoveSynmove(handle, 0,1,100,0);//轴 0 正向运动 100 后启动轴 1 运动 ZAux_Direct_Single_Move(handle, 0, 200);//轴 0 正向运动 200
详细说明	这个指令 LOAD 执行时不做任何运动，只修改参数。此指令的类型与 MOVE_OP 一致。

指令 87	**ZAux_Direct_MoveASynmove**
指令原型	int32 __stdcall ZAux_Direct_MoveASynmove(ZMC_HANDLE handle, uint32 base_axis, uint32 iaxis,float fdist,uint32 ifsp)
指令说明	运动中触发其他轴运动，当前轴不等待。需要 20170503 以上版本固件支持。
输入参数	参数名 / 描述 handle / 连接标识。 base_axis / 运动轴轴号。 iaxis / 触发的轴号。 fdist / 触发运动的距离。 ifsp / 触发运动是否为 SP 运动。
输出参数	/

返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	例：轴 0 总运动 300，在运动到 100 的时候，触发轴 1 运动 100，不会等待轴 1 运动完成，轴 0 继续完成剩下的运动 ZAux_Direct_SetMerge(handle, 0,1);//打开连续插补 ZAux_Direct_Single_Move(handle, 0, 100);//轴 0 正向运动 100 ZAux_Direct_MoveASynmove(handle, 0,1,100,0);//轴 0 正向运动 100 后启动轴 1 运动 ZAux_Direct_Single_Move(handle, 0, 200);//轴 0 正向运动 200
详细说明	这个指令 LOAD 执行时不做任何运动，只修改参数。此指令的类型与 MOVE_OP 一致。

指令 88	**ZAux_Direct_MoveTable**
指令原型	int32 __stdcall ZAux_Direct_MoveTable(ZMC_HANDLE handle,uint32 base_axis,uint32 table_num,float fvalue)
指令说明	在运动缓冲区中修改 Table 的值。以达到运动中修改 Table 的值
输入参数	参数名：描述 handle：连接标识。 base_axis：运动轴轴号。 table_num：TABLE 编号。 fvalue：设定的值。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	例：轴 0 总运动 300，在运动到 100 的时候，触发修改 table(1)=10. ZAux_Direct_Single_Move(handle, 0, 100);//轴 0 正向运动 100 ZAux_Direct_MoveTable(handle, 0,1,10);//轴 0 正向运动 100 后，等到 IN1 有信号，才执行下一条缓冲 ZAux_Direct_Single_Move(handle, 0,200);//轴 0 正向运动 200
详细说明	这个指令 LOAD 执行时不做任何运动，只修改参数。此指令的类型与 MOVE_OP 一致。

指令 89	**ZAux_Direct_MoveCancel**
指令原型	int32 __stdcall ZAux_Direct_MoveCancel(ZMC_HANDLE handle, int32 base_axis, int32 Cancel_Axis,int iMode)
指令说明	运动缓冲区中取消其他轴运动，以达到运动中取消其他轴运动的效果

<table>
<tr><td>输入参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>handle</td><td>连接标识。</td></tr>
<tr><td>Base_axis</td><td>运动轴轴号。</td></tr>
<tr><td>Cancel_axis</td><td>停止轴轴号。</td></tr>
<tr><td>iMode</td><td>停止模式
0：取消当前运动
1：取消缓冲的运动
2：取消当前运动和缓冲的运动，停止速度参考快减速 FASTDEC
3：立即中断脉冲发送
4：取消当前运动和缓冲运动，停止速度参考减速度 DECEL</td></tr>
</table>
CANCEL(4)为 4 系列控制器固件版本 170708 及以上支持。
CANCEL(3)不要对正在插补的从轴使用。</td></tr>
<tr><td>输出参数</td><td>/</td></tr>
<tr><td>返回值</td><td>成功返回值为 0，非 0 详见错误码说明。</td></tr>
<tr><td>指令示例</td><td>例：轴 0 总运动 300，在运动到 100 的时候，停止轴 1
ZAux_Direct_Single_Move(handle, 0, 100);//轴 0 正向运动 100
ZAux_Direct_MoveCancel(handle,0,1,0)//轴 0 缓冲里面插入停止轴 1 指令
ZAux_Direct_Single_Move(handle, 0,200);//轴 0 正向运动 200</td></tr>
<tr><td>详细说明</td><td>/</td></tr>
</table>

<table>
<tr><td>指令 90</td><td>ZAux_Direct_MoveModify</td></tr>
<tr><td>指令原型</td><td>int32 __stdcall ZAux_Direct_MoveModify(ZMC_HANDLE handle, int iaxis, float pfDisance)</td></tr>
<tr><td>指令说明</td><td>修改上一个运动的目标位置，前面没有运动时与 MOVEABS 效果一样。
连续插补时使用 MOVEMODIFY 会破坏速度连续性。</td></tr>
<tr><td>输入参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>handle</td><td>连接标识。</td></tr>
<tr><td>iaxis</td><td>轴号。</td></tr>
<tr><td>pfDisance</td><td>运动的距离。</td></tr>
</table>
</td></tr>
<tr><td>输出参数</td><td>/</td></tr>
<tr><td>返回值</td><td>成功返回值为 0，非 0 详见错误码说明。</td></tr>
<tr><td>指令示例</td><td>例：轴 0 以很慢的速度运动 300，1 秒后，把轴 0 总运动 300 修改为 200
ZAux_Direct_Single_Move(handle, 0, 300);//轴 0 正向运动 100</td></tr>
</table>

	Sleep(1000); ZAux_Direct_MoveModify(handle, 0,200);//修改轴 0 的目标位置
详细说明	前面没有运动时与 MOVEABS 效果一样，但不会进入运动缓冲 连续插补时使用 MOVEMODIFY 会破坏速度连续性。 MOVEMODIFY 同时对多轴运动时不一定为直线插补。

指令 91	**ZAux_Direct_MoveLimit**
指令原型	int32 __stdcall ZAux_Direct_MoveLimit(ZMC_HANDLE handle, int iaxis,float limitspeed)
指令说明	在当前的运动末尾位置增加速度限制，用于强制拐角减速。
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。 limitspeed | 限制到的速度。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	ZAux_Direct_SetMerge(handle, 0,1);//打开连续插补 ZAux_Direct_Single_Move(handle, 0, 200);//轴 0 正向运动 200 ZAux_Direct_MoveLimit(handle, 0,100);//两个运动之间降速到 100 ZAux_Direct_Single_Move(handle, 0, 200);//轴 0 正向运动 200
详细说明	/

指令 92	**ZAux_Direct_Single_Move**
指令原型	int32 __stdcall ZAux_Direct_Single_Move(ZMC_HANDLE handle, int iaxis, float fdistance)
指令说明	单轴相对运动。
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。 fdistance | 距离。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	单轴点位运动
详细说明	/

指令 93	ZAux_Direct_Single_MoveAbs
指令原型	int32 __stdcall ZAux_Direct_Single_MoveAbs(ZMC_HANDLE handle, int iaxis, float fdistance)
指令说明	单轴绝对运动。
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。 fdistance | 绝对距离。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	单轴点位运动
详细说明	/

指令 94	ZAux_Direct_Single_Vmove
指令原型	int32 __stdcall ZAux_Direct_Single_Vmove(ZMC_HANDLE handle, int iaxis, int idir)
指令说明	单轴连续运动指令，连续往一个方向运动。
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。 idir | 方向-1—负向 1—正向。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	单轴持续运动
详细说明	当前面的 VMOVE 运动没有停止时，此 VMOVE 指令会自动替换前面的 VMOVE 指令并修改方向，因此无需 CANCEL 前面的 VMOVE 指令。

指令 95	ZAux_Direct_GetJogSpeed
指令原型	int32 __stdcall ZAux_Direct_GetJogSpeed (ZMC_HANDLE handle , int iaxis, float *pfValue);
指令说明	读取 FastJog 运动时 Jog 速度。

输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。
输入参数	参数名 | 描述 pfValue | 返回的 JOG 速度值。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	快速点动（IN 控制）
详细说明	当 REV_JOG/FWD_JOG 被设置，对应输入点按下时，并保持当前输入状态，电机将以该函数设置的速度运动，输入点松开运动停止。

指令 96	**ZAux_Direct_SetJogSpeed**
指令原型	int32 __stdcall ZAux_Direct_SetJogSpeed (ZMC_HANDLE handle , int iaxis, float fValue);
指令说明	设置的 FastJog 运动时 Jog 速度。
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。 fValue | 设置的速度值
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	快速点动（IN 控制）
详细说明	当 REV_JOG/FWD_JOG 被设置，对应输入点按下时，并保持当前输入状态，电机将以该函数设置的速度运动，输入点松开运动停止。

指令 97	**ZAux_Direct_GetFastJog**
指令原型	int32 __stdcall ZAux_Direct_GetFastJog (ZMC_HANDLE handle , int iaxis, int*piValue);
指令说明	读取轴快速点动速度映射输入点。设置值为-1 时,不生效。
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。

输入参数	参数名 | 描述 piValue | 返回的 JOG 输入口编号。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	快速点动（IN 控制）
详细说明	如果设置快速点动输入，速度由 SPEED 参数给出。如果没有输入设置，速度由 JOGSPEED 参数给出

指令 98	**ZAux_Direct_SetFastJog**
指令原型	int32 __stdcall ZAux_Direct_SetFastJog (ZMC_HANDLE handle , int iaxis, int iValue);
指令说明	设置轴快速点动速度映射输入点。设置值为-1 时,不生效。
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。 iValue | 设置快速 JOG 输入口编号。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	快速点动（IN 控制）
详细说明	如果设置快速点动输入，速度由 SPEED 参数给出。如果没有输入设置，速度由 JOGSPEED 参数给出

指令 99	**ZAux_Direct_SetFwdJog**
指令原型	int32 __stdcall ZAux_Direct_SetFwdJog (ZMC_HANDLE handle , int iaxis, int iValue);
指令说明	设置正向 JOG 输入口编号。设置值为-1 时,不生效。
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。 iValue | 设置正向 JOG 输入口编号。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	快速点动（IN 控制）
详细说明	当有信号输入时，对应轴按照 JOGSPEED 速度正向运动

指令 100	ZAux_Direct_GetFwdJog
指令原型	int32 __stdcall ZAux_Direct_GetFwdJog(ZMC_HANDLE handle, int iaxis, int *piValue);
指令说明	读取正向 JOG 输入口编号。设置值为-1 时,不生效。
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。
输入参数	参数名 | 描述 piValue | 返回的正向 JOG 输入口编号
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	快速点动（IN 控制）
详细说明	当有信号输入时，对应轴按照 JOGSPEED 速度正向运动

指令 101	ZAux_Direct_GetRevJog
指令原型	int32 __stdcall ZAux_Direct_GetRevJog (ZMC_HANDLE handle, int iaxis, int * piValue);
指令说明	读取负向 JOG 输入对应的输入点编号，-1 无效。
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。
输入参数	参数名 | 描述 piValue | 返回的负向 JOG 输入口编号
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	快速点动（IN 控制）
详细说明	当有信号输入时，对应轴按照 JOGSPEED 速度负向运动 当 REV_JOG 和 FWD_JOG 同时有信号输入时，轴按照 FWD_JOG 运行。

指令 102	ZAux_Direct_SetRevJog
指令原型	int32 __stdcall ZAux_Direct_SetRevJog (ZMC_HANDLE handle, int iaxis, int iValue);
指令说明	设置负向 JOG 输入对应的输入点编号，-1 无效。

输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。 iValue | 设置负向 JOG 输入口编号。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	快速点动（IN 控制）
详细说明	当有信号输入时，对应轴按照 JOGSPEED 速度负向运动 当 REV_JOG 和 FWD_JOG 同时有信号输入时，轴按照 FWD_JOG 运行。

指令 103	**ZAux_Direct_Single_Datum**
指令原型	int32 __stdcall ZAux_Direct_Single_Datum(ZMC_HANDLE handle, int iaxis, int imode)
指令说明	单轴回零运动。
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。 imode | 找原点模式(详情看详细说明)
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	单轴回零运动

详细说明

mode：找原点模式，加 10 表示碰到限位后调头反找，不会碰到限位停止，例如 13=模式 3+限位调头反找，用于原点在正中间的情况。

值	描述
0	清除所有轴的错误状态。
1	轴以 CREEP 速度正向运行直到 Z 信号出现。碰到限位开关会直接停止。DPOS 值重置为 0 同时纠正 MPOS。
2	轴以 CREEP 速度反向运行直到 Z 信号出现。碰到限位开关会直接停止。DPOS 值重置为 0 同时纠正 MPOS。
3	轴以 SPEED 速度正向运行，直到碰到原点开关。然后轴以 CREEP 速度反向运动直到离开原点开关。找原点阶段碰到正限位开关会直接停止。爬行阶段碰到负限位开关会直接停止。DPOS 值重置为 0 同时纠正 MPOS
4	轴以 SPEED 速度反向运行，直到碰到原点开关。然后轴以 CREEP 速度正向运动直到离开原点开关。找原点阶段碰到负限位开关会直接停止。爬行阶段碰到正限位开关会直接停止。DPOS 值重置为 0 同时纠正 MPOS。
5	轴以 SPEED 速度正向运行，直到碰到原点开关。然后轴以 CREEP 速度反向运动直到离开原点开关，然后再继续以爬行速度反转直到碰到 Z 信号。碰到限位开关会直接停止。DPOS 值重置为 0 同时纠正 MPOS。
6	轴以 SPEED 速度反向运行，直到碰到原点开关。然后轴以 CREEP 速度正向运动直到离开原点开关，然后再继续以爬行速度正转直到碰到 Z 信号。碰到限位开关会直接停止。DPOS 值重置为 0 同时纠正 MPOS。
8	轴以 SPEED 速度正向运行，直到碰到原点开关。碰到限位开关会直接停止。
9	轴以 SPEED 速度反向运行，直到碰到原点开关。碰到限位开关会直接停止。

指令 104	ZAux_BusCmd_Datum
指令原型	int32 __stdcall ZAux_BusCmd_Datum (ZMC_HANDLE handle, uint32 iaxis , uint32 homemode);
指令说明	总线驱动器回零

输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。 Homemode | 回零模式，查看驱动器手册
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	/
详细说明	设置驱动器回零方式（6098h），缺省 0 表示使用驱动器当前的回零方式。 会使用轴的 SPEED, CREEP, ACCEL, DECEL，乘以 UNITS 后自动设置驱动器的 6099h，609Ah 动作时序： 6098 回零方式→6099 速度→609A 加速度→6060 切换当前模式

指令 105	**ZAux_Direct_UserDatum**
指令原型	int32 __stdcall ZAux_Direct_UserDatum(ZMC_HANDLE handle, int iaxis, int imode,float HighSpeed,float LowSpeed,float DatumOffset);
指令说明	自定义二次回零
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。 imode | 找原点模式(详情看详细说明) HihSpeed | 回零高速 LowSpeed | 回零低速 Datumoffset | 二次回零偏移距离
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	/

Imode

回零模式（默认为碰到限位反找）

值	描述
31	轴以 CREEP 速度正向运行直到 Z 信号出现。后以回零高速进行偏移，紧接着，轴再次以 CREEP 速度正向运行直到 Z 信号出现。 碰到限位开关会反找。DPOS 值重置为 0 同时纠正 MPOS。两次
32	轴以 CREEP 速度反向运行直到 Z 信号出现。后以回零高速进行偏移，紧接着，轴再次以 CREEP 速度反向运行直到 Z 信号出现。 碰到限位开关会反找。DPOS 值重置为 0 同时纠正 MPOS。两次
33	轴以 SPEED 速度正向运行，直到碰到原点开关。然后轴以 CREEP 速度反向运动直到离开原点开关。后以回零高速进行偏移，紧接着，轴再次以 SPEED 速度正向运行，直到碰到原点开关。然后轴以 CREEP 速度反向运动直到离开原点开关。 找原点阶段碰到正限位开关会反找。爬行阶段碰到负限位开关会直接停止。 DPOS 值重置为 0 同时纠正 MPOS。两次
34	轴以 SPEED 速度反向运行，直到碰到原点开关。然后轴以 CREEP 速度正向运动直到离开原点开关。后以回零高速进行偏移，紧接着，轴再次以 SPEED 速度反向运行，直到碰到原点开关。然后轴以 CREEP 速度正向运动直到离开原点开关。 找原点阶段碰到负限位开关会反找。爬行阶段碰到正限位开关会直接停止。DPOS 值重置为 0 同时纠正 MPOS。两次
35	轴以 SPEED 速度正向运行，直到碰到原点开关。然后轴以 CREEP 速度反向运动直到离开原点开关，然后再继续以爬行速度反转直到碰到 Z 信号。后以回零高速进行偏移，紧接着，轴再次以 SPEED 速度正向运行，直到碰到原点开关。然后轴以 CREEP 速度反向运动直到离开原点开关，然后再继续以爬行速度反转直到碰到 Z 信号。 碰到限位开关会直接反找。DPOS 值重置为 0 同时纠正 MPOS。两次

	36	轴以 SPEED 速度反向运行，直到碰到原点开关。然后轴以 CREEP 速度正向运动直到离开原点开关，然后再继续以爬行速度正转直到碰到 Z 信号。 后以回零高速进行偏移，紧接着，轴再次以 SPEED 速度反向运行，直到碰到原点开关。然后轴以 CREEP 速度正向运动直到离开原点开关，然后再继续以爬行速度正转直到碰到 Z 信号。 碰到限位开关会直接反找。DPOS 值重置为 0 同时纠正 MPOS。两次
	38	轴以 SPEED 速度正向运行，直到碰到原点开关。后以回零高速进行偏移，紧接着，轴再次以 SPEED 速度正向运行，直到碰到原点开关。 碰到限位开关会直接反找。两次
	39	轴以 SPEED 速度反向运行，直到碰到原点开关。后以回零高速进行偏移，紧接着，轴再次以 SPEED 速度反向运行，直到碰到原点开关。 碰到限位开关会直接反找。两次

指令 106	ZAux_Direct_SetHomeWait
指令原型	int32 __stdcall ZAux_Direct_SetHomeWait (ZMC_HANDLE handle, int iaxis, int fValue);
指令说明	设置回零返找等待时间。
输入参数	参数名 | 描述 handle | 连接句柄。 iaxis | 轴口编号。 fValue | 回零返找等待时间 MS
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	单轴回零运动
详细说明	对脉冲方式的伺服驱动器，回原点运动时，当反找时要等待一定时间。控制器默认值为 2ms。

指令 107	ZAux_Direct_GetHomeWait
指令原型	int32 __stdcall ZAux_Direct_GetHomeWait (ZMC_HANDLE handle, int iaxis, int *pfValue);
指令说明	读取回零返找等待时间。对脉冲方式的伺服驱动器，回原点运动时，当反

	找时要等待一定时间。控制器默认值为 2ms。
输入参数	参数名 | 描述 handle | 连接句柄。 iaxis | 轴口编号。
输出参数	参数名 | 描述 pfValue | 获取的回零返找等待时间 MS
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	Int iValue; ZAux_Direct_GetHomeWait (handle, 0, &iValue); //读取轴 0 的回零返找等待时间的设置值
详细说明	对脉冲方式的伺服驱动器，回原点运动时，当反找时要等待一定时间。控制器默认值为 2ms。

指令 108	**ZAux_BusCmd_SetDatumOffpos**
指令原型	int32 __stdcall ZAux_BusCmd_SetDatumOffpos (ZMC_HANDLE handle, int iaxis , float fValue);
指令说明	设置回零偏移距离。
输入参数	参数名 | 描述 handle | 连接句柄。 iaxis | 轴口号。 fvalue | 设置偏移距离。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	ZAux_BusCmd_SetDatumOffpos(handle, 0, 100); //设置轴 0 的回零完成后偏移 100
详细说明	主要针对总线电机，若为脉冲型电机，部分控制器支持，由固件决定。

指令 109	**ZAux_BusCmd_GetDatumOffpos**
指令原型	int32 __stdcall ZAux_BusCmd_GetDatumOffpos (ZMC_HANDLE handle, int iaxis , float *fValue);
指令说明	获取设置的回零偏移距离。

输入参数	参数名 | 描述 handle | 连接句柄。 iaxis | 轴口编号。
输出参数	参数名 | 描述 pfValue | 获取的回零偏移距离。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	float fValue; ZAux_BusCmd_GetDatumOffpos(handle, 0, &fValue); //读取轴 0 的回零偏移设置值
详细说明	主要针对总线电机，若为脉冲型电机，部分控制器支持，由固件决定，

指令 110	**ZAux_Direct_SetCreep**
指令原型	int32 __stdcall ZAux_Direct_SetCreep(ZMC_HANDLE handle, int iaxis, float fValue)
指令说明	设置回零爬行速度，用于原点搜寻。单位 units/s。
输入参数	参数名 | 描述 handle | 连接句柄。 iaxis | 轴口编号。 pfValue | 设置的爬行速度
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	单轴回零运动
详细说明	/

指令 111	**ZAux_Direct_GetHomeStatus**
指令原型	int32 __stdcall ZAux_Direct_GetHomeStatus (ZMC_HANDLE handle , uint32 iaxis, uint32 *homestatus);
指令说明	回零完成状态。
输入参数	参数名 | 描述 handle | 连接句柄。 iaxis | 轴号。

输出参数	参数名 | 描述 homestatue | 读取回零状态的值： 0-回零异常　1-回零正常。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	单轴回零运动
详细说明	/

指令 112	**ZAux_BusCmd_GetHomeStatus**
指令原型	int32 __stdcall ZAux_BusCmd_GetHomeStatus (ZMC_HANDLE handle, uint32 iaxis , uint32 * homestate);
指令说明	总线驱动器回零完成状态。
输入参数	参数名 | 描述 handle | 连接句柄。 iaxis | 轴号。
输出参数	参数名 | 描述 homestatue | 读取回零状态的值： 0-回零异常　1-回零正常。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	/
详细说明	/

指令 113	**ZAux_Direct_SetDatumIn**
指令原型	int32 __stdcall ZAux_Direct_SetDatumIn(ZMC_HANDLE handle, int iaxis , int iValue)
指令说明	设置轴映射原点 IN，-1 表示取消。
输入参数	参数名 | 描述 handle | 连接句柄。 iaxis | 轴口编号。 iValue | IO 口编号，-1 时为取消设置。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	单轴回零运动
详细说明	设置了原点开关后，ZMC 控制器输入 OFF 时认为有信号输入，要相反效

	果可以用 INVERT_IN 反转电平。(ECI 系列控制器相反)。

指令 114	ZAux_Direct_GetDatumIn
指令原型	int32 __stdcall ZAux_Direct_GetDatumIn(ZMC_HANDLE handle, int iaxis, int *piValue)
指令说明	读取轴映射原点 IN
输入参数	参数名 | 描述 handle | 连接句柄。 iaxis | 轴口编号。
输出参数	参数名 | 描述 piValue | 回对应的输入口编号，-1 表示没有设置。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	单轴回零运动
详细说明	设置了原点开关后，ZMC 控制器输入 OFF 时认为有信号输入，要相反效果可以用 INVERT_IN 反转电平。(ECI 系列控制器相反)。

指令 115	ZAux_Direct_SetClutchRate
指令原型	int32 __stdcall ZAux_Direct_SetClutchRate(ZMC_HANDLE handle, int iaxis, float fValue)
指令说明	connect 连接的速度，ratio/秒,默认值 1000000；
输入参数	参数名 | 描述 handle | 连接句柄。 iaxis | 轴口编号。 pfValue | 同步连接速率
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	机械手例程
详细说明	当设置为 0 时，根据轴的速度/加速度参数来跟踪连接，比较适合于手轮运动（当速度不够高时可能导致要持续运动一段时间才能结束）。 设置值如果不能远大于 CONNECT 连接比例的话，实际连接比例会减小 一般用于定义连结率从 0 到设置倍率的改变时间

指令 116	ZAux_Direct_GetClutchRate
指令原型	int32 __stdcall ZAux_Direct_GetClutchRate(ZMC_HANDLE handle, int

	iaxis, float *pfValue)
指令说明	读取连接速率。
输入参数	参数名 | 描述 handle | 连接句柄。 iaxis | 轴口编号。
输入参数	参数名 | 描述 pfValue | 当前轴的同步连接速率
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	/
详细说明	当设置为 0 时，根据轴的速度/加速度参数来跟踪连接，比较适合于手轮运动（当速度不够高时可能导致要持续运动一段时间才能结束）。 设置值如果不能远大于 CONNECT 连接比例的话，实际连接比例会减小 一般用于定义连结率从 0 到设置倍率的改变时间

指令 117	**ZAux_Direct_EncoderRatio**
指令原型	int32 __stdcall ZAux_Direct_EncoderRatio(ZMC_HANDLE handle, int iaxis,int output_count,int input_count)
指令说明	编码器输入齿轮比设置，缺省(1,1)，设置负值可切换方向。
输入参数	参数名 | 描述 handle | 连接句柄。 iaxis | 轴口编号。 Output_count | 分子。 Input_count | 分母。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	编码器齿轮比设置
详细说明	/

指令 118	**ZAux_Direct_StepRatio**
指令原型	int32 __stdcall ZAux_Direct_StepRatio(ZMC_HANDLE handle, int iaxis, int output_count,int input_count)
指令说明	设置步进输出齿轮比，缺省(1,1)。范围 1-65535。

输入参数	<table><tr><td>参数名</td><td>描述</td></tr><tr><td>handle</td><td>连接句柄。</td></tr><tr><td>iaxis</td><td>轴口编号。</td></tr><tr><td>Output_count</td><td>分子。</td></tr><tr><td>Input_count</td><td>分母。</td></tr></table>
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	脉冲输出电子齿轮比设置
详细说明	设置为负值可修改电机方向，但一般不建议。 脉冲电机可用 INVERT_STEP 指令；总线电机最好在驱动器上修改。 建议不要随便修改此参数，可以通过修改脉冲当量来达到相同的效果。

指令 119	**ZAux_Direct_Connect**
指令原型	int32 __stdcall ZAux_Direct_Connect(ZMC_HANDLE handle, float ratio, int link_axis, int move_axis)
指令说明	同步运动。电子齿轮，将当前轴的目标位置与 idring_axis 轴的插补矢量长度通过电子齿轮连接。
输入参数	<table><tr><td>参数名</td><td>描述</td></tr><tr><td>handle</td><td>连接句柄。</td></tr><tr><td>ratio</td><td>比率，可正可负，注意是脉冲个数的比例。</td></tr><tr><td>link_axis</td><td>连接轴的轴号（主轴）。</td></tr><tr><td>move_axis</td><td>运动轴的轴号。</td></tr></table>
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	同步运动
详细说明	1.连接的是脉冲个数，要考虑不同轴 UNITS 的比例。取消连接时用 CANCEL。 2.假设连接轴 0 的 UNIST 为 100，被连接轴 1 的 UNITS 为 10。 3.使用 ZAux_Direct_Connect 连接，比例 ratio 为 1，当轴 0 运动 s1=100 时，轴 1 运动=s1*UNITS(0)*ratio/UNITS(1)，此时运动 1000。 4.比率可以通过重复调用指令动态变化。 5.常用于手轮使用。

指令 120	**ZAux_Direct_Connpath**

指令原型	int32 __stdcall ZAux_Direct_Connpath(ZMC_HANDLE handle, float ratio, int link_axis, int move_axis)
指令说明	同步运动 2。电子齿轮，将当前轴的目标位置与 link_axis 轴的插补矢量长度通过电子齿轮连接。
输入参数	参数名 | 描述 handle | 连接句柄。 ratio | 比率，可正可负，注意是脉冲个数的比例。 link_axis | 连接轴的轴号（主轴）。 move_axis | 运动轴的轴号。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	同步运动 2
详细说明	1.需要连接到插补运动的主轴才能与插补矢量长度建立连接，连接到从轴的效果与 CONNECT 相同。 2.连接的是脉冲个数，要考虑不同轴 UNITS 的比例。取消连接时用 CANCEL。 3.比率可以通过重复调用指令动态变化

指令 121	**ZAux_Direct_GetLinkax**
指令原型	int32 __stdcall ZAux_Direct_GetLinkax (ZMC_HANDLE handle , int iaxis, int *piValue);
指令说明	读取当前连接参考的轴号。
输入参数	参数名 | 描述 handle | 连接句柄。 iaxis | 轴号。
输入参数	参数名 | 描述 piValue | 返回连接的参考轴号
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	Float fValue; ZAux_Direct_GetLinkax (handle ,0, &fValue); //查询轴 0 连接轴轴号
详细说明	返回当前连接运动的参考轴号，无连接时返回-1。

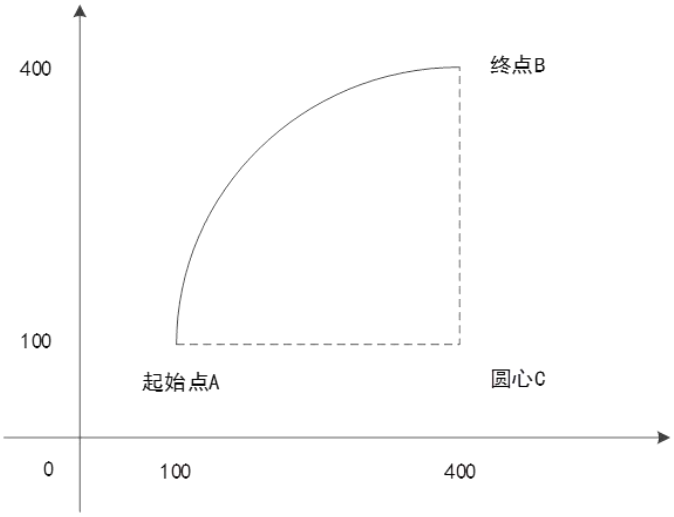
指令 122	ZAux_Direct_Move
指令原型	int32 __stdcall ZAux_Direct_Move(ZMC_HANDLE handle, int imaxaxises, int *piAxislist,float *pfDisancelist)
指令说明	多轴相对直线插补运动。
输入参数	参数名 / 描述 handle / 连接句柄。 imaxaxises / 运动轴数。 piAxislist / 轴列表数组。 pfDisancelist / 运动的距离列表。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	多轴直线插补
详细说明	插补运动距离 X=$\sqrt{X_0^2+X_1^2+X_2^2+....+X_n^2}$ 运动时间 T=X/主轴 SPEED 插补运动时只有主轴速度参数有效，主轴是轴列表数组的第一个轴，运动参照这个轴的参数。

指令 123	ZAux_Direct_MoveSp
指令原型	int32 __stdcall ZAux_Direct_MoveSp(ZMC_HANDLE handle, int imaxaxises , int *piAxislist, float *pfDisancelist)
指令说明	相对直线插补运动对应 SP 运动指令。
输入参数	参数名 / 描述 handle / 连接句柄。 imaxaxises / 运动轴数。 piAxislist / 轴列表数组。 pfDisancelist / 运动的距离列表。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	连续插补 SP 速度运动控制
详细说明	插补运动距离 X=$\sqrt{X_0^2+X_1^2+X_2^2+....+X_n^2}$ 运动时间 T=X/主轴 SPEED 插补运动时只有主轴速度参数有效，主轴是轴列表数组的第一个轴，运动参照这个轴的参数。

指令 124	ZAux_Direct_MoveAbs
指令原型	int32 __stdcall ZAux_Direct_MoveAbs(ZMC_HANDLE handle,int imaxaxises, int *piAxislist, float *pfDisancelist)
指令说明	绝对直线插补运动。
输入参数	参数名 / 描述 handle / 连接句柄。 imaxaxises / 运动轴数。 piAxislist / 轴列表数组。 pfDisancelist / 运动的绝对距离列表。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	多轴直线插补
详细说明	插补运动距离 X=$\sqrt{X_0^2+X_1^2+X_2^2+....+X_n^2}$ 运动时间 T=X/主轴 SPEED 插补运动时只有主轴速度参数有效，主轴是轴列表数组的第一个轴，运动参照这个轴的参数。

指令 125	ZAux_Direct_MoveAbsSp
指令原型	int32 __stdcall ZAux_Direct_MoveAbsSp(ZMC_HANDLE handle,int imaxaxises, int *piAxislist, float *pfDisancelist)
指令说明	绝对直线插补运动对应 SP 运动指令。
输入参数	参数名 / 描述 handle / 连接句柄。 imaxaxises / 运动轴数。 piAxislist / 轴列表数组。 pfDisancelist / 运动的绝对距离列表。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	/
详细说明	插补运动距离 X=$\sqrt{X_0^2+X_1^2+X_2^2+....+X_n^2}$ 运动时间 T=X/主轴 SPEED 插补运动时只有主轴速度参数有效，主轴是轴列表数组的第一个轴，运动

	参照这个轴的参数。

指令 126	ZAux_Direct_MoveCirc
指令原型	int32 __stdcall ZAux_Direct_MoveCirc(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fend1, float fend2, float fcenter1, float fcenter2, int idirection)
指令说明	圆心画弧。轴列表中第一轴和第二轴进行圆弧插补，相对移动方式。
输入参数	参数名 | 描述 handle | 连接标识。 imaxaxises | 总轴数。 piAxislist | 轴列表数组。 fend1 | 第一个轴运动终点坐标。 fend2 | 第二个轴运动终点坐标。 fcenter1 | 第一个轴运动圆心，相对于起始点。 fcenter2 | 第二个轴运动圆心，相对于起始点。 idirection | 0-逆时针，1-顺时针。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	平面圆心画圆+小圆限速
详细说明	第一轴和第二轴进行圆弧插补，相对移动方式，当终点距离为 0 时为整圆。 使用时需获得圆心和圆弧终点相对于起始点坐标。 坐标位置请确保正确，否则实际运动轨迹会错误 

	假设起始点 A 坐标为（100,100），圆心 C 坐标为（400,100），终点 B 坐标为（400,400） 圆心 C 相对于起始点 A 的坐标为（300,0），终点 B 相对于起始点 A 的坐标为（300,300）

指令 127	ZAux_Direct_MoveCircSp
指令原型	int32 __stdcall ZAux_Direct_MoveCircSp(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fend1, float fend2, float fcenter1, float fcenter2, int idirection)
指令说明	圆心画弧相对-SP 运动指令。
输入参数	参数名 | 描述 handle | 连接标识。 imaxaxises | 总轴数。 piAxislist | 轴列表数组。 fend1 | 第一个轴终点运动坐标。 fend2 | 第二个轴终点运动坐标。 fcenter1 | 第一个轴运动圆心，相对于起始点。 fcenter2 | 第二个轴运动圆心，相对于起始点。 idirection | 0-逆时针，1-顺时针。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	参考：ZAux_Direct_MoveCirc 示例
详细说明	参考：ZAux_Direct_MoveCirc 详细说明

指令 128	ZAux_Direct_MoveCircAbs
指令原型	int32 __stdcall ZAux_Direct_MoveCircAbs(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fend1, float fend2, float fcenter1, float fcenter2, int idirection)
指令说明	圆心画弧。轴列表中第一轴和第二轴进行圆弧插补，绝对移动方式。

输入参数	参数名 | 描述 handle | 连接标识。 imaxaxises | 总轴数。 piAxislist | 轴列表数组。 fend1 | 第一个轴终点运动坐标。 fend2 | 第二个轴终点运动坐标。 fcenter1 | 第一个轴运动圆心，绝对于起始点。 fcenter2 | 第二个轴运动圆心，绝对于起始点。 idirection | 0-逆时针，1-顺时针。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	参考：ZAux_Direct_MoveCirc 示例
详细说明	参考：ZAux_Direct_MoveCirc 详细说明

指令 129	**ZAux_Direct_MoveCircAbsSp**
指令原型	int32 __stdcall ZAux_Direct_MoveCircAbsSp(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fend1, float fend2, float fcenter1, float fcenter2, int idirection)
指令说明	圆心画弧。绝对圆弧运动对应 SP 运动指令。
输入参数	参数名 | 描述 handle | 连接标识。 imaxaxises | 总轴数。 piAxislist | 轴列表数组。 fend1 | 第一个轴终点运动坐标。 fend2 | 第二个轴终点运动坐标。 fcenter1 | 第一个轴运动圆心，绝对于起始点。 fcenter2 | 第二个轴运动圆心，绝对于起始点。 idirection | 0-逆时针，1-顺时针。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	参考：ZAux_Direct_MoveCirc 示例
详细说明	参考：ZAux_Direct_MoveCirc 详细说明

指令 130	**ZAux_Direct_MoveCirc2**

指令原型	int32 __stdcall ZAux_Direct_MoveCirc2(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fmid1, float fmid2, float fend1, float fend2)
指令说明	三点画弧。圆弧插补运动，轴列表第一轴和第二轴进行圆弧插补，相对移动方式。
输入参数	参数名 | 描述 handle | 连接标识。 imaxaxises | 总轴数。 piAxislist | 轴列表数组。 fmid1 | 第一个轴中间点，相对起始点距离。 fmid2 | 第二个轴中间点，相对起始点距离。 fend1 | 第一个轴结束点，相对起始点距离。 fend2 | 第二个轴结束点，相对起始点距离。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	平面三点画弧
详细说明	注意：不能用此指令进行整圆插补运动，整圆使用 MOVECIRC 相对圆弧，或连续使用两条此类指令。

指令 131	**ZAux_Direct_MoveCirc2Sp**
指令原型	int32 __stdcall ZAux_Direct_MoveCirc2Sp(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fmid1, float fmid2, float fend1, float fend2)
指令说明	三点画弧。圆弧插补运动，轴列表第一轴和第二轴进行圆弧插补，相对移动方式。相对运动对应的 SP 运动指令。
输入参数	参数名 | 描述 handle | 连接标识。 imaxaxises | 总轴数。 piAxislist | 轴列表数组。 fmid1 | 第一个轴中间点，相对起始点距离。 fmid2 | 第二个轴中间点，相对起始点距离。 fend1 | 第一个轴结束点，相对起始点距离。 fend2 | 第二个轴结束点，相对起始点距离。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。

指令示例	参考 ZAux_Direct_MoveCirc2 示例
详细说明	注意：不能用此指令进行整圆插补运动，整圆使用 MOVECIRC 相对圆弧，或连续使用两条此类指令。

指令 132	ZAux_Direct_MoveCirc2Abs
指令原型	int32 __stdcall ZAux_Direct_MoveCirc2Abs(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fmid1, float fmid2, float fend1, float fend2)
指令说明	三点画弧。圆弧插补运动，轴列表中第一轴和第二轴进行圆弧插补，绝对移动方式。
输入参数	参数名 | 描述 handle | 连接标识。 imaxaxises | 总轴数。 piAxislist | 轴列表数组。 fmid1 | 中间点第一个轴坐标，绝对位置。 fmid2 | 中间点第二个轴坐标，绝对位置。 fend1 | 结束点第一个轴坐标，绝对位置。 fend2 | 结束点第二个轴坐标，绝对位置。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	参考 ZAux_Direct_MoveCirc2 示例
详细说明	注意：不能用此指令进行整圆插补运动，整圆使用 MOVECIRC 相对圆弧，或连续使用两条此类指令。

指令 133	ZAux_Direct_MoveCirc2AbsSp
指令原型	int32 __stdcall ZAux_Direct_MoveCirc2AbsSp(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fmid1, float fmid2, float fend1, float fend2)
指令说明	圆弧插补运动，轴列表中第一轴和第二轴进行圆弧插补，绝对移动方式。对应的 SP 运动指令。

输入参数	参数名 | 描述 handle | 连接标识。 imaxaxises | 总轴数。 piAxislist | 轴列表数组。 fmid1 | 第一个轴中间点，绝对起始点距离。 fmid2 | 第二个轴中间点，绝对起始点距离。 fend1 | 第一个轴结束点，绝对起始点距离。 fend2 | 第二个轴结束点，绝对起始点距离。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	参考 ZAux_Direct_MoveCirc2 示例
详细说明	注意：不能用此指令进行整圆插补运动，整圆使用 MOVECIRC 相对圆弧，或连续使用两条此类指令。

指令 134	**ZAux_Direct_MHelical**
指令原型	int32 __stdcall ZAux_Direct_MHelical(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fend1, float fend2, float fcenter1, float fcenter2, int idirection, float fDistance3, int imode)
指令说明	圆心螺旋。轴列表中第一轴和第二轴进行圆弧插补，第三轴进行螺旋，为相对于起始点。
输入参数	参数名 | 描述 handle | 连接标识。 imaxaxises | 总轴数。 piAxislist | 轴列表数组。 fend1 | 结束点第一个轴运动坐标。 fend2 | 结束点第二个轴运动坐标。 fcentre1 | 第一个轴运动圆心，相对与起始点。 fcentre2 | 第二个轴运动圆心，相对与起始点。 idirection | 0-逆时针，1-顺时针。 fdistance3 | 第三个轴运动距离。 imode | 第三轴的速度计算： 0(缺省)：第三轴参与速度计算。 1：第三轴不参与速度计算。
输出参数	/

返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	螺旋曲线
详细说明	可完整螺旋一圈，从 Z 方向看为一整圆。 坐标位置请确保正确，否则实际运动轨迹会错误。

指令 135	**ZAux_Direct_MHelicalSp**
指令原型	int32 __stdcall ZAux_Direct_MHelicalSp(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fend1, float fend2, float fcenter1, float fcenter2, int idirection, float fDistance3, int imode)
指令说明	圆心螺旋运动对应 SP 运动指令。
输入参数	参数名：描述 handle：连接标识。 imaxaxises：总轴数。 piAxislist：轴列表数组。 fend1：第一个轴运动坐标。 fend2：第二个轴运动坐标。 fcentre1：第一个轴运动圆心，相对与起始点。 fcentre2：第二个轴运动圆心，相对与起始点。 idirection：0-逆时针，1-顺时针。 fdistance3：第三个轴运动距离。 imod：第三轴的速度计算： 0(缺省)：第三轴参与速度计算。 1：第三轴不参与速度计算。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	参考 ZAux_Direct_MHelical 示例
详细说明	可完整螺旋一圈，从 Z 方向看为一整圆。 坐标位置请确保正确，否则实际运动轨迹会错误。

指令 136	**ZAux_Direct_MHelicalAbs**
指令原型	int32 __stdcall ZAux_Direct_MHelicalAbs(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fend1, float fend2, float fcenter1, float fcenter2, int idirection, float fDistance3, int imode)
指令说明	圆心螺旋。轴列表中第一轴和第二轴进行圆弧插补，第三轴进行螺旋，绝对移动方式。

输入参数	参数名 / 描述 handle / 连接标识。 imaxaxises / 总轴数。 piAxislist / 轴列表数组。 fend1 / 第一个轴运动坐标。 fend2 / 第二个轴运动坐标。 fcentre1 / 第一个轴运动圆心，绝对坐标。 fcentre2 / 第二个轴运动圆心，绝对坐标。 idirection / 0-逆时针，1-顺时针。 fdistance3 / 第三个轴运动距离。 imod / 第三轴的速度计算： 0(缺省)：第三轴参与速度计算。 1：第三轴不参与速度计算。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	参考 ZAux_Direct_MHelical 示例
详细说明	可完整螺旋一圈，从 Z 方向看为一整圆。 坐标位置请确保正确，否则实际运动轨迹会错误。

指令 137	**ZAux_Direct_MHelicalAbsSp**
指令原型	int32 __stdcall ZAux_Direct_MHelicalAbsSp(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fend1, float fend2, float fcenter1, float fcenter2, int idirection, float fDistance3, int imode)
指令说明	圆心螺旋。轴列表中第一轴和第二轴进行圆弧插补，第三轴进行螺旋，绝对移动方式。对应 SP 运动指令。
输入参数	参数名 / 描述 handle / 连接标识。 imaxaxises / 总轴数。 piAxislist / 轴列表数组。 fend1 / 第一个轴运动坐标。 fend2 / 第二个轴运动坐标。 fcentre1 / 第一个轴运动圆心，绝对坐标。 fcentre2 / 第二个轴运动圆心，绝对坐标。 idirection / 0-逆时针，1-顺时针。

	fdistance3 | 第三个轴运动距离。 imode | 第三轴的速度计算： 0(缺省)：第三轴参与速度计算。 1：第三轴不参与速度计算。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	参考 ZAux_Direct_MHelical 示例
详细说明	可完整螺旋一圈，从 Z 方向看为一整圆。 坐标位置请确保正确，否则实际运动轨迹会错误。

指令 138	**ZAux_Direct_MHelical2Abs**
指令原型	int32 __stdcall ZAux_Direct_MHelical2Abs(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fmid1, float fmid2, float fend1, float fend2, float fDistance3, int imode)
指令说明	三点螺旋。轴列表中第一轴和第二轴进行圆弧插补，第三轴进行螺旋，绝对移动方式。
输入参数	参数名 | 描述 handle | 连接标识。 imaxaxises | 总轴数。 piAxislist | 轴列表数组。 fmid1 | 第一个轴中间点坐标，绝对坐标。 fmid2 | 第二个轴中间点坐标，绝对坐标。 fend1 | 第一个轴结束点坐标，绝对坐标。 fend2 | 第二个轴结束点坐标，绝对坐标。 fdistance3 | 第三个轴运动距离。 imode | 第三轴的速度计算： 0(缺省)：第三轴参与速度计算。 1：第三轴不参与速度计算。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	/
详细说明	无法螺旋一圈，需要螺旋一圈请使用 MHELICAL 或 MHELICALABS。

指令 139	ZAux_Direct_MHelical2AbsSp
指令原型	int32 __stdcall ZAux_Direct_MHelical2AbsSp(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fmid1, float fmid2, float fend1, float fend2, float fDistance3, int imode)
指令说明	函数功能：轴列表中第一轴和第二轴进行圆弧插补，第三轴进行螺旋，绝对移动方式。对应 SP 运动指令。
输入参数	参数名：描述 handle：连接标识。 imaxaxises：总轴数。 piAxislist：轴列表数组。 fmid1：第一个轴中间点坐标，绝对坐标。 fmid2：第二个轴中间点坐标，绝对坐标。 fend1：第一个轴结束点坐标，绝对坐标。 fend2：第二个轴结束点坐标，绝对坐标。 fdistance3：第三个轴运动距离。 imode：第三轴的速度计算： 0(缺省)：第三轴参与速度计算。 1：第三轴不参与速度计算。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	/
详细说明	无法螺旋一圈，需要螺旋一圈请使用 MHELICAL 或 MHELICALABS。

指令 140	ZAux_Direct_MHelical2
指令原型	int32 __stdcall ZAux_Direct_MHelical2(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fmid1, float fmid2, float fend1, float fend2, float fDistance3, int imode)
指令说明	轴列表中第一轴和第二轴进行圆弧插补，第三轴进行螺旋，相对移动方式。
输入参数	参数名：描述 handle：连接标识。 imaxaxises：总轴数。 piAxislist：轴列表数组。 fmid1：第一个轴中间点坐标，相对与起始点 fmid2：第二个轴中间点坐标，相对与起始点 fend1：第一个轴结束点坐标，相对与起始点

	fend2 第二个轴结束点坐标，相对与起始点 fdistance3 第三个轴运动距离。 imode 第三轴的速度计算： 0(缺省)：第三轴参与速度计算。 1：第三轴不参与速度计算。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	/
详细说明	无法螺旋一圈，需要螺旋一圈请使用 MHELICAL 或 MHELICALABS。

指令 141	**ZAux_Direct_MHelical2Sp**
指令原型	int32 __stdcall ZAux_Direct_MHelical2Sp(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fmid1, float fmid2, float fend1, float fend2, float fDistance3, int imode)
指令说明	轴列表中第一轴和第二轴进行圆弧插补，第三轴进行螺旋，相对移动方式。对应 SP 运动指令。
输入参数	参数名 描述 handle 连接标识。 imaxaxises 总轴数。 piAxislist 轴列表数组。 fmid1 第一个轴中间点坐标，相对与起始点 fmid2 第二个轴中间点坐标，相对与起始点 fend1 第一个轴结束点坐标，相对与起始点 fend2 第二个轴结束点坐标，相对与起始点 fdistance3 第三个轴运动距离。 imode 第三轴的速度计算： 0(缺省)：第三轴参与速度计算。 1：第三轴不参与速度计算。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	/
详细说明	无法螺旋一圈，需要螺旋一圈请使用 MHELICAL 或 MHELICALABS。

指令 142	**ZAux_Direct_MEclipse**

<table>
<tr><td>指令原型</td><td>int32 __stdcall ZAux_Direct_MEclipse(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fend1, float fend2, float fcenter1, float fcenter2, int idirection, float fADis, float fBDis)</td></tr>
<tr><td>指令说明</td><td>椭圆插补，中心画弧。轴列表中第一轴和第二轴进行椭圆插补，相对移动方式。</td></tr>
<tr><td>输入参数</td><td><table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>handle</td><td>连接标识。</td></tr>
<tr><td>imaxaxises</td><td>总轴数。</td></tr>
<tr><td>piAxislist</td><td>轴列表数组。</td></tr>
<tr><td>fend1</td><td>第一个轴运动坐标。</td></tr>
<tr><td>fend2</td><td>第二个轴运动坐标。</td></tr>
<tr><td>fcentre1</td><td>第一个轴运动圆心，相对与起始点。</td></tr>
<tr><td>fcentre2</td><td>第二个轴运动圆心，相对与起始点。</td></tr>
<tr><td>idirection</td><td>0-逆时针，1-顺时针。</td></tr>
<tr><td>fADis</td><td>第一轴的椭圆半径，半长轴或者半短轴都可。</td></tr>
<tr><td>fBDis</td><td>第二轴的椭圆半径，半长轴或者半短轴都可，AB 相等时自动为圆弧或螺旋。</td></tr>
</table></td></tr>
<tr><td>输出参数</td><td>/</td></tr>
<tr><td>返回值</td><td>成功返回值为 0，非 0 详见错误码说明。</td></tr>
<tr><td>指令示例</td><td>/</td></tr>
<tr><td>详细说明</td><td>可选第三个轴同步螺旋
可画整椭圆
只能画长轴（短轴）与 X 平行或垂直的椭圆</td></tr>
</table>

指令 143	**ZAux_Direct_MEclipseSp**
指令原型	int32 __stdcall ZAux_Direct_MEclipseSp(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fend1, float fend2, float fcenter1, float fcenter2, int idirection, float fADis, float fBDis)
指令说明	椭圆插补运动对应 SP 运动指令。

<table>
<tr><td>输入参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>handle</td><td>连接标识。</td></tr>
<tr><td>imaxaxises</td><td>总轴数。</td></tr>
<tr><td>piAxislist</td><td>轴列表数组。</td></tr>
<tr><td>fend1</td><td>第一个轴运动坐标。</td></tr>
<tr><td>fend2</td><td>第二个轴运动坐标。</td></tr>
<tr><td>fcentre1</td><td>第一个轴运动圆心，相对与起始点。</td></tr>
<tr><td>fcentre2</td><td>第二个轴运动圆心，相对与起始点。</td></tr>
<tr><td>idirection</td><td>0-逆时针，1-顺时针。</td></tr>
<tr><td>fADis</td><td>第一轴的椭圆半径，半长轴或者半短轴都可。</td></tr>
<tr><td>fBDis</td><td>第二轴的椭圆半径，半长轴或者半短轴都可，AB 相等时自动为圆弧或螺旋。</td></tr>
</table>
</td></tr>
<tr><td>输出参数</td><td>/</td></tr>
<tr><td>返回值</td><td>成功返回值为 0，非 0 详见错误码说明。</td></tr>
<tr><td>指令示例</td><td>/</td></tr>
<tr><td>详细说明</td><td>可选第三个轴同步螺旋
可画整椭圆
只能画长轴（短轴）与 X 平行或垂直的椭圆</td></tr>
</table>

指令 144	**ZAux_Direct_MEclipseAbs**
指令原型	int32 __stdcall ZAux_Direct_MEclipseAbs(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fend1, float fend2, float fcenter1, float fcenter2, int idirection, float fADis, float fBDis)
指令说明	轴列表中第一轴和第二轴进行椭圆插补，绝对移动方式。

<table>
<tr><td>输入参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>handle</td><td>连接标识。</td></tr>
<tr><td>imaxaxises</td><td>总轴数。</td></tr>
<tr><td>piAxislist</td><td>轴列表数组。</td></tr>
<tr><td>fend1</td><td>第一个轴运动坐标。</td></tr>
<tr><td>fend2</td><td>第二个轴运动坐标。</td></tr>
<tr><td>fcentre1</td><td>第一个轴运动圆心，绝对与起始点。</td></tr>
<tr><td>fcentre2</td><td>第二个轴运动圆心，绝对与起始点。</td></tr>
<tr><td>idirection</td><td>0-逆时针，1-顺时针。</td></tr>
<tr><td>fADis</td><td>第一轴的椭圆半径，半长轴或者半短轴都可。</td></tr>
<tr><td>fBDis</td><td>第二轴的椭圆半径，半长轴或者半短轴都可，AB 相等时自动为圆弧或螺旋。</td></tr>
</table>
</td></tr>
<tr><td>输出参数</td><td>/</td></tr>
<tr><td>返回值</td><td>成功返回值为 0，非 0 详见错误码说明。</td></tr>
<tr><td>指令示例</td><td>/</td></tr>
<tr><td>详细说明</td><td>可选第三个轴同步螺旋
可画整椭圆
只能画长轴（短轴）与 X 平行或垂直的椭圆</td></tr>
</table>

指令 145	**ZAux_Direct_MEclipseAbsSp**
指令原型	int32 __stdcall ZAux_Direct_MEclipseAbsSp(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fend1, float fend2, float fcenter1, float fcenter2, int idirection, float fADis, float fBDis)
指令说明	轴列表中第一轴和第二轴进行椭圆插补，绝对移动方式。对应的 SP 运动指令。

<table>
<tr><td>输入参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>handle</td><td>连接标识。</td></tr>
<tr><td>imaxaxises</td><td>总轴数。</td></tr>
<tr><td>piAxislist</td><td>轴列表数组。</td></tr>
<tr><td>fend1</td><td>第一个轴运动坐标。</td></tr>
<tr><td>fend2</td><td>第二个轴运动坐标。</td></tr>
<tr><td>fcentre1</td><td>第一个轴运动圆心，绝对与起始点。</td></tr>
<tr><td>fcentre2</td><td>第二个轴运动圆心，绝对与起始点。</td></tr>
<tr><td>idirection</td><td>0-逆时针，1-顺时针。</td></tr>
<tr><td>fADis</td><td>第一轴的椭圆半径，半长轴或者半短轴都可。</td></tr>
<tr><td>fBDis</td><td>第二轴的椭圆半径，半长轴或者半短轴都可，AB 相等时自动为圆弧或螺旋。</td></tr>
</table>
</td></tr>
<tr><td>输出参数</td><td>/</td></tr>
<tr><td>返回值</td><td>成功返回值为 0，非 0 详见错误码说明。</td></tr>
<tr><td>指令示例</td><td>/</td></tr>
<tr><td>详细说明</td><td>可选第三个轴同步螺旋
可画整椭圆
只能画长轴（短轴）与 X 平行或垂直的椭圆</td></tr>
</table>

指令 146	**ZAux_Direct_MEclipseHelical**
指令原型	int32 __stdcall ZAux_Direct_MEclipseHelical(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fend1, float fend2, float fcenter1, float fcenter2, int idirection, float fADis, float fBDis, float fDistance3)
指令说明	插补、椭圆+螺旋

输入参数	参数名 / 描述 handle / 连接标识。 imaxaxises / 总轴数。 piAxislist / 轴列表数组。 fend1 / 第一个轴运动坐标。 fend2 / 第二个轴运动坐标。 fcentre1 / 第一个轴运动圆心，相对与起始点。 fcentre2 / 第二个轴运动圆心，相对与起始点。 idirection / 0-逆时针，1-顺时针。 fADis / 第一轴的椭圆半径，半长轴或者半短轴都可 fBDis / 第二轴的椭圆半径，半长轴或者半短轴都可，AB 相等时自动为圆弧或螺旋。 fDistance3 / 第三个轴的运动坐标。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	/
详细说明	坐标位置请确保正确，否则实际运动轨迹会错误。

指令 147	**ZAux_Direct_MEclipseHelicalSp**
指令原型	int32 __stdcall ZAux_Direct_MEclipseHelicalSp(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fend1, float fend2, float fcenter1, float fcenter2, int idirection, float fADis, float fBDis, float fDistance3)
指令说明	插补、椭圆+螺旋。相对应的 SP 运动指令。
输入参数	参数名 / 描述 handle / 连接标识。 imaxaxises / 总轴数。 piAxislist / 轴列表数组。 fend1 / 第一个轴运动坐标。 fend2 / 第二个轴运动坐标。 fcentre1 / 第一个轴运动圆心，相对与起始点。 fcentre2 / 第二个轴运动圆心，相对与起始点。 idirection / 0-逆时针，1-顺时针。 fADis / 第一轴的椭圆半径，半长轴或者半短轴都可 fBDis / 第二轴的椭圆半径，半长轴或者半短轴都可，AB 相等时自动为圆弧或螺旋。

	fDistance3	第三个轴的运动坐标。
输出参数	/	
返回值	成功返回值为 0，非 0 详见**错误码**说明。	
指令示例	/	
详细说明	坐标位置请确保正确，否则实际运动轨迹会错误。	

指令 148	**ZAux_Direct_MEclipseHelicalAbs**	
指令原型	int32 __stdcall ZAux_Direct_MEclipseHelicalAbs(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fend1, float fend2, float fcenter1, float fcenter2, int idirection, float fADis, float fBDis,float fDistance3)	
指令说明	轴列表中第一轴和第二轴进行椭圆插补，绝对移动方式，第三个轴同步螺旋。绝对移动方式。	
输入参数	参数名	描述
	handle	连接标识。
	imaxaxises	总轴数。
	piAxislist	轴列表数组。
	fend1	第一个轴运动坐标。绝对坐标。
	fend2	第二个轴运动坐标。绝对坐标。
	fcentre1	第一个轴运动圆心，绝对于起始点。
	fcentre2	第二个轴运动圆心，绝对于起始点。
	idirection	0-逆时针，1-顺时针。
	fADis	第一轴的椭圆半径，半长轴或者半短轴都可以
	fBDis	第二轴的椭圆半径，半长轴或者半短轴都可以，AB 相等时自动为圆弧或螺旋。
	fDistance3	第三个轴的运动坐标。
输出参数	/	
返回值	成功返回值为 0，非 0 详见**错误码**说明。	
指令示例	/	
详细说明	坐标位置请确保正确，否则实际运动轨迹会错误。	

指令 149	**ZAux_Direct_MEclipseHelicalAbsSp**
指令原型	int32 __stdcall ZAux_Direct_MEclipseHelicalAbsSp(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fend1,float fend2, float fcenter1, float fcenter2, int idirection, float fADis, float fBDis,float fDistance3)

<table>
<tr><td>指令说明</td><td>轴列表中第一轴和第二轴进行椭圆插补，绝对移动方式，第三个轴同步螺旋。绝对移动方式。对应的 SP 运动指令。</td></tr>
<tr><td>输入参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>handle</td><td>连接标识。</td></tr>
<tr><td>imaxaxises</td><td>总轴数。</td></tr>
<tr><td>piAxislist</td><td>轴列表数组。</td></tr>
<tr><td>fend1</td><td>第一个轴运动坐标。绝对坐标。</td></tr>
<tr><td>fend2</td><td>第二个轴运动坐标。绝对坐标。</td></tr>
<tr><td>fcentre1</td><td>第一个轴运动圆心，绝对于起始点。</td></tr>
<tr><td>fcentre2</td><td>第二个轴运动圆心，绝对于起始点。</td></tr>
<tr><td>idirection</td><td>0-逆时针，1-顺时针。</td></tr>
<tr><td>fADis</td><td>第一轴的椭圆半径，半长轴或者半短轴都可以</td></tr>
<tr><td>fBDis</td><td>第二轴的椭圆半径，半长轴或者半短轴都可以，AB 相等时自动为圆弧或螺旋。</td></tr>
<tr><td>fDistance3</td><td>第三个轴的运动坐标。</td></tr>
</table>
</td></tr>
<tr><td>输出参数</td><td>/</td></tr>
<tr><td>返回值</td><td>成功返回值为 0，非 0 详见错误码说明。</td></tr>
<tr><td>指令示例</td><td>/</td></tr>
<tr><td>详细说明</td><td>坐标位置请确保正确，否则实际运动轨迹会错误。</td></tr>
</table>

指令 150	**ZAux_Direct_MSpherical**
指令原型	int32 __stdcall ZAux_Direct_MSpherical(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fend1, float fend2, float fend3, float fcenter1, float fcenter2, float fcenter3, int imode, float fcenter4, float fcenter5)
指令说明	空间圆弧插补运动，相对移动方式。

<table>
<tr><td>输入参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>handle</td><td>连接标识</td></tr>
<tr><td>imaxaxises</td><td>总轴数</td></tr>
<tr><td>piAxislist</td><td>轴列表数组。</td></tr>
<tr><td>fend1</td><td>第 1 个轴运动距离参数 1。</td></tr>
<tr><td>fend2</td><td>第 2 个轴运动距离参数 1。</td></tr>
<tr><td>fend3</td><td>第 3 个轴运动距离参数 1。</td></tr>
<tr><td>fcentre1</td><td>第 1 个轴运动距离参数 2。</td></tr>
<tr><td>fcentre2</td><td>第 2 个轴运动距离参数 2。</td></tr>
<tr><td>fcentre3</td><td>第 3 个轴运动距离参数 2。</td></tr>
<tr><td>imode</td><td>指定前面参数的意义
0：当前点，中间点，终点三点定圆弧:
距离参数 1：为终点距离，
距离参数 2：为中间点距离。
1：当前点，圆心，终点定圆弧,走最小的圆弧:
距离参数 1：为终点距离，
距离参数 2：为圆心的距离。
2：当前点，中间点，终点三点定圆:
距离参数 1：为终点距离，
距离参数 2：为中间点距离。
3：当前点，圆心，终点定圆,先走最小的圆弧，再继续走完整圆:
距离参数 1：为终点距离
距离参数 2：为圆心的距离</td></tr>
<tr><td>Fcentre4</td><td>第 4 个轴螺旋距离。</td></tr>
<tr><td>Fcentre5</td><td>第 5 个轴螺旋距离。</td></tr>
</table>
</td></tr>
<tr><td>输出参数</td><td>/</td></tr>
<tr><td>返回值</td><td>成功返回值为 0，非 0 详见错误码说明。</td></tr>
<tr><td>指令示例</td><td>空间圆弧</td></tr>
<tr><td>详细说明</td><td>坐标位置请确保正确，否则实际运动轨迹会错误。</td></tr>
</table>

指令 151	**ZAux_Direct_MSphericalSp**
指令原型	int32 __stdcall ZAux_Direct_MSphericalSp(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fend1, float fend2, float fend3, float fcenter1, float fcenter2, float fcenter3, int imode, float fcenter4, float

<table>
<tr><td></td><td>fcenter5)</td></tr>
<tr><td>指令说明</td><td>空间圆弧插补运动，相对移动方式。对应的 SP 运动指令。</td></tr>
<tr><td>输入参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>handle</td><td>连接标识</td></tr>
<tr><td>imaxaxises</td><td>总轴数</td></tr>
<tr><td>piAxislist</td><td>轴列表数组。</td></tr>
<tr><td>fend1</td><td>第 1 个轴运动距离参数 1。</td></tr>
<tr><td>fend2</td><td>第 2 个轴运动距离参数 1。</td></tr>
<tr><td>fend3</td><td>第 3 个轴运动距离参数 1。</td></tr>
<tr><td>fcentre1</td><td>第 1 个轴运动距离参数 2。</td></tr>
<tr><td>fcentre2</td><td>第 2 个轴运动距离参数 2。</td></tr>
<tr><td>fcentre3</td><td>第 3 个轴运动距离参数 2。</td></tr>
<tr><td>imode</td><td>指定前面参数的意义
0：当前点，中间点，终点三点定圆弧:
距离参数 1：为终点距离，
距离参数 2：为中间点距离。
1：当前点，圆心，终点定圆弧,走最小的圆弧:
距离参数 1：为终点距离，
距离参数 2：为圆心的距离。
2：当前点，中间点，终点三点定圆:
距离参数 1:为终点距离，
距离参数 2:为中间点距离。
3：当前点，圆心，终点定圆,先走最小的圆弧，再继续走完整圆:
距离参数 1:为终点距离，
距离参数 2:为圆心的距离</td></tr>
<tr><td>Fcentre4</td><td>第 4 个轴螺旋距离。</td></tr>
<tr><td>Fcentre5</td><td>第 5 个轴螺旋距离。</td></tr>
</table>
</td></tr>
<tr><td>输出参数</td><td>/</td></tr>
<tr><td>返回值</td><td>成功返回值为 0，非 0 详见错误码说明。</td></tr>
<tr><td>相关指令</td><td>参考 ZAux_Direct_MSpherical 例程</td></tr>
<tr><td>详细说明</td><td>坐标位置请确保正确，否则实际运动轨迹会错误。</td></tr>
</table>

指令 152	**ZAux_Direct_MoveSpiral**
指令原型	int32 __stdcall ZAux_Direct_MoveSpiral(ZMC_HANDLE handle, int

<table>
<tr><td></td><td>imaxaxises, int *piAxislist, float centre1, float centre2, float circles, float pitch,float distance3, float distance4)</td></tr>
<tr><td>指令说明</td><td>渐开线圆弧插补运动，相对移动方式，当起始半径 0 直接扩散时从 0 角度开始。</td></tr>
<tr><td>输入参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>handle</td><td>连接标识。</td></tr>
<tr><td>maxaxises</td><td>总轴数。</td></tr>
<tr><td>piAxislist</td><td>轴列表数组。</td></tr>
<tr><td>centre1</td><td>第 1 轴圆心的相对距离</td></tr>
<tr><td>centre2</td><td>第 2 轴圆心的相对距离</td></tr>
<tr><td>circles</td><td>要旋转的圈数，可以为小数圈，负数表示顺时针</td></tr>
<tr><td>pitch</td><td>每圈的扩散距离，可以为负。</td></tr>
<tr><td>distance3</td><td>第 3 轴螺旋的功能，指定第 3 轴的相对距离，此轴不参与速度计算。</td></tr>
<tr><td>distance4</td><td>第 4 轴螺旋的功能，指定第 4 轴的相对距离，此轴不参与速度计算。</td></tr>
</table>
</td></tr>
<tr><td>输出参数</td><td>/</td></tr>
<tr><td>返回值</td><td>成功返回值为 0，非 0 详见错误码说明。</td></tr>
<tr><td>相关指令</td><td>/</td></tr>
<tr><td>详细说明</td><td>当前点和圆心距离确定起始半径，当起始半径 0 时角度无法确定，直接从 0 角度开始。</td></tr>
</table>

指令 153	**ZAux_Direct_MoveSpiralSp**
指令原型	int32 __stdcall ZAux_Direct_MoveSpiralSp(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float centre1, float centre2, float circles, float pitch,float distance3, float distance4)
指令说明	渐开线圆弧插补运动，相对移动方式，当起始半径 0 直接扩散时从 0 角度开始。

输入参数	<table><tr><td>参数名</td><td>描述</td></tr><tr><td>handle</td><td>连接标识。</td></tr><tr><td>maxaxises</td><td>总轴数。</td></tr><tr><td>piAxislist</td><td>轴列表数组。</td></tr><tr><td>centre1</td><td>第 1 轴圆心的相对距离</td></tr><tr><td>centre2</td><td>第 2 轴圆心的相对距离</td></tr><tr><td>circles</td><td>要旋转的圈数，可以为小数圈，负数表示顺时针</td></tr><tr><td>pitch</td><td>每圈的扩散距离，可以为负。</td></tr><tr><td>distance3</td><td>第 3 轴螺旋的功能，指定第 3 轴的相对距离，此轴不参与速度计算。</td></tr><tr><td>distance4</td><td>第 4 轴螺旋的功能，指定第 4 轴的相对距离，此轴不参与速度计算。</td></tr></table>
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
相关指令	/
详细说明	当前点和圆心距离确定起始半径，当起始半径 0 时角度无法确定，直接从 0 角度开始。

指令 154	**ZAux_Direct_MoveSmooth**
指令原型	int32 __stdcall ZAux_Direct_MoveSmooth(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float end1, float end2, float end3, float next1, float next2, float next3, float radius)
指令说明	空间直线倒圆插补运动，根据下一个直线运动的绝对坐标在拐角自动插入圆弧。

输入参数	<table><tr><td>参数名</td><td>描述</td></tr><tr><td>handle</td><td>连接标识。</td></tr><tr><td>imaxaxises</td><td>运动轴数。</td></tr><tr><td>piAxislist</td><td>轴列表数组。</td></tr><tr><td>end1</td><td>第一个轴绝对坐标。</td></tr><tr><td>end2</td><td>第二个轴绝对坐标。</td></tr><tr><td>end3</td><td>第三个轴绝对坐标。</td></tr><tr><td>next1</td><td>第一个轴下个直线运动绝对坐标。</td></tr><tr><td>Next2</td><td>第二个轴下个直线运动绝对坐标。</td></tr><tr><td>Next3</td><td>第三个轴下个直线运动绝对坐标。</td></tr><tr><td>radius</td><td>插入圆弧的半径，过大时自动缩小。</td></tr></table>
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
相关指令	/
详细说明	此指令为早期开发指令，有轴数限制，建议使用 CORNER_MODE 指令，功能更多。 根据下一个直线运动的绝对坐标在拐角自动插入圆弧，加入圆弧后会使得运动的终点与直线的终点不一致，拐角过大时不会插入圆弧，当距离不够时会自动减小半径。

指令 155	**ZAux_Direct_MoveSmoothSp**
指令原型	int32 __stdcall ZAux_Direct_MoveSmoothSp(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float end1, float end2, float end3, float next1, float next2, float next3, float radius)
指令说明	空间直线倒圆插补运动，根据下一个直线运动的绝对坐标在拐角自动插入圆弧。

输入参数	参数名 / 描述 handle / 连接标识。 imaxaxises / 运动轴数。 piAxislist / 轴列表数组。 end1 / 第一个轴绝对坐标。 end2 / 第二个轴绝对坐标。 end3 / 第三个轴绝对坐标。 next1 / 第一个轴下个直线运动绝对坐标。 Next2 / 第二个轴下个直线运动绝对坐标。 Next3 / 第三个轴下个直线运动绝对坐标。 radius / 插入圆弧的半径，过大时自动缩小。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
相关指令	/
详细说明	此指令为早期开发指令，有轴数限制，建议使用 CORNER_MODE 指令，功能更多。 根据下一个直线运动的绝对坐标在拐角自动插入圆弧，加入圆弧后会使得运动的终点与直线的终点不一致，拐角过大时不会插入圆弧，当距离不够时会自动减小半径。

指令 156	**ZAux_Direct_McircTurnabs**
指令原型	int32 __stdcall ZAux_Direct_McircTurnabs (ZMC_HANDLE handle , int tablenum, float refpos1, float refpos2, int mode, float end1, float end2, int maxaxises, int *piAxislist, float * pfDistancelist);
指令说明	旋转台圆弧+螺旋插补运动。20130901 以后的控制器版本支持
输入参数	参数名 / 描述 handle / 连接句柄。 tablenum / 存储旋转参数的 TABLE 编号。 refpos1 / 第一个轴参考点，绝对位置。 refpos2 / 第二个轴参考点，绝对位置。 mode / 1-参考点是当前点前面，2-参考点是结束点后面，3-参考点在中间，采用三点定圆的方式。 end1 / 第一个轴结束点，绝对位置。 end2 / 第二个轴结束点，绝对位置。

	Maxaxises	参与运动轴数量。
	piAxislist	轴列表。
	pfDistancelist	旋转轴距离列表。
输出参数	/	
返回值	成功返回值为 0，非 0 详见**错误码**说明。	
相关指令	/	
详细说明	旋转功能是指工作平台在与 XY 平行的平面上旋转，旋转的正向与 XY 的正向要一致（右手法则）。 旋转的参数存储在 TABLE 表里面，依次存储 R 轴号，R 轴一圈脉冲数，X 轴号，Y 轴号，X 圆心，Y 圆心。	

指令 157	**ZAux_Direct_MoveTurnabs**	
指令原型	int32 __stdcall ZAux_Direct_MoveTurnabs (ZMC_HANDLE handle , int tablenum, int imaxaxises, int *piAxislist, float * pfDistancelist);	
指令说明	旋转台直线插补运动。20130901 以后的控制器版本支持	
输入参数	参数名	描述
	handle	连接句柄。
	tablenum	存储旋转参数的 TABLE 编号。
	imaxaxises	参与运动轴数量。
	piAxislist	轴号列表。
	pfDistancelist	距离列表。
输出参数	/	
返回值	成功返回值为 0，非 0 详见**错误码**说明。	
相关指令	/	
详细说明	旋转功能是指工作平台在与 XY 平行的平面上旋转，旋转的正向与 XY 的正向要一致（右手法则）。 旋转的参数存储在 TABLE 表里面，依次存储 R 轴号，R 轴一圈脉冲数，X 轴号，Y 轴号，X 圆心，Y 圆心。	

指令 158	**ZAux_Direct_MultiMove**
指令原型	int32 __stdcall ZAux_Direct_MultiMove(ZMC_HANDLE handle,int iMoveLen, int imaxaxises, int *piAxislist, float *pfDisancelist)
指令说明	多条相对多轴直线插补

输入参数	参数名 | 描述 handle | 连接句柄。 iMoveLen | 填写的运动长度 imaxaxises | 参与运动轴数量。 piAxislist | 轴号列表数组。 pfDistancelist | 距离列表数组。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	例：轴 0，1，2，3 一次性写入 5 组运动。 int axisList[4]={0,1,2,3}; float posList[20]; //位置赋予 for(int i=0;i<5;i++) { posList[0+i*4]=100; //轴 0 位置 posList[1+i*4]=200; //轴 1 位置 posList[2+i*4]=300; //轴 2 位置 posList[3+i*4]=400; //轴 3 位置 } //轴 0，1，2，3 一次性写入 5 组运动 ZAux_Direct_MultiMove(handle,5,4, axisList, posList);
详细说明	一次性发送多条多轴直线相对位置运动

指令 159	**ZAux_Direct_MultiMoveAbs**
指令原型	int32 __stdcall ZAux_Direct_MultiMoveAbs(ZMC_HANDLE handle,int iMoveLen, int imaxaxises, int *piAxislist, float *pfDisancelist)
指令说明	多条绝对多轴直线插补
输入参数	参数名 | 描述 handle | 连接句柄。 iMoveLen | 填写的运动长度 imaxaxises | 参与运动轴数量。 piAxislist | 轴号列表数组。 pfDistancelist | 距离列表数组。

输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	例：轴 0，1，2，3 一次性写入 5 组运动。 int axisList[4]={0,1,2,3}; float posList[20]; //位置赋予 for(int i=0;i<5;i++) { posList[0+i*4]=100+10*i; //轴 0 位置 posList[1+i*4]=200+20*i; //轴 1 位置 posList[2+i*4]=300+30*i; //轴 2 位置 posList[3+i*4]=400+40*i; //轴 3 位置 } //轴 0，1，2，3 一次性写入 5 组运动 ZAux_Direct_MultiMoveAbs (handle,5,4, axisList, posList);
详细说明	一次性发送多条多轴直线绝对位置运动

指令 160	**ZAux_Direct_MultiLineN**
指令原型	int32 __stdcall ZAux_Direct_MultiLineN(ZMC_HANDLE handle,int imode,int iMoveLen, int imaxaxises, int *piAxislist, float *pfDisancelist,int *iReBuffLen)
指令说明	多条相对多轴直线插补
输入参数	参数名 | 描述 handle | 连接句柄。 imode | bit0- bifabs 绝对值模式 bit1- bifsp 自定义速度模式 bit2- bifresume 恢复运动 bit3- bifmovescan 直线运动切换成振镜运动 iMoveLen | 填写的运动长度 imaxaxises | 参与运动轴数量。 piAxislist | 轴号列表数组。 pfDistancelist | 距离列表数组。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
相关指令	例：轴 0，1，2，3 一次性写入 5 组运动。

	int axisList[4]={0,1,2,3}; float posList[20]; //位置赋予 for(int i=0;i<5;i++) { posList[0+i*4]=100+10*i; //轴 0 位置 posList[1+i*4]=200+20*i; //轴 1 位置 posList[2+i*4]=300+30*i; //轴 2 位置 posList[3+i*4]=400+40*i; //轴 3 位置 } //轴 0，1，2，3 一次性写入 5 组运动 ZAux_Direct_MultiLineN (handle,1,5,4, axisList, posList);
详细说明	直接发送连续小线段插补。

指令 161	ZAux_Direct_SetFhspeed
指令原型	int32 __stdcall ZAux_Direct_SetFhspeed (ZMC_HANDLE handle , int iaxis, float fValue);
指令说明	设置轴保持速度。
输入参数	参数名 | 描述 handle | 连接句柄。 iaxis | 轴号。 fValue | 设置的速度值。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	轴保持输入应用
详细说明	对应的输入处于保持状态时才能一直以此速度运动。

指令 162	ZAux_Direct_GetFhspeed
指令原型	int32 __stdcall ZAux_Direct_GetFhspeed (ZMC_HANDLE handle , int iaxis, float *pfValue);
指令说明	读取轴保持速度。

输入参数	<table><tr><td>参数名</td><td>描述</td></tr><tr><td>handle</td><td>连接句柄。</td></tr><tr><td>iaxis</td><td>轴号。</td></tr><tr><td>fValue</td><td>设置的速度值。</td></tr></table>
输出参数	<table><tr><td>参数名</td><td>描述</td></tr><tr><td>fValue</td><td>读取的速度值。</td></tr></table>
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	参考：轴保持输入应用
详细说明	对应的输入处于保持状态时才能一直以此速度运动。

指令 163	**ZAux_Direct_SetEndMoveSpeed**
指令原型	int32 __stdcall ZAux_Direct_SetEndMoveSpeed(ZMC_HANDLE handle, int iaxis, float fValue);
指令说明	设置自定义速度的 SP 运动的结束速度。
输入参数	<table><tr><td>参数名</td><td>描述</td></tr><tr><td>handle</td><td>连接句柄。</td></tr><tr><td>iaxis</td><td>轴号。</td></tr><tr><td>fValue</td><td>设置的速度值。</td></tr></table>
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	连续插补 SP 运动
详细说明	只有使用了带 SP 的运动指令才生效。 不使用时请设置较大值。控制器默认值 1000。

指令 164	**ZAux_Direct_GetEndMoveSpeed**
指令原型	int32 __stdcall ZAux_Direct_GetEndMoveSpeed(ZMC_HANDLE handle, int iaxis, float *fValue);
指令说明	读取自定义速度的 SP 运动的结束速度。
输入参数	<table><tr><td>参数名</td><td>描述</td></tr><tr><td>handle</td><td>连接句柄。</td></tr><tr><td>iaxis</td><td>轴号。</td></tr></table>

输出参数	参数名 | 描述 fValue | 读取的速度值。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	参考：连续插补 SP 运动
详细说明	只有使用了带 SP 的运动指令才生效。 不使用时请设置较大值。控制器默认值 1000。

指令 165	**ZAux_Direct_GetForceSpeed**
指令原型	int32 __stdcall ZAux_Direct_GetForceSpeed (ZMC_HANDLE handle , int iaxis, float *pfValue);
指令说明	读取 SP 的运行速度。
输入参数	参数名 | 描述 handle | 连接句柄。 iaxis | 轴号。
输出参数	参数名 | 描述 fValue | 读取的 SP 运动速度值。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	参考：轴保持输入应用
详细说明	这个参数被带入运动缓冲。 当 FORCE_SPEED 大于 SPEED 时，SPEED 同时也会生效限制住最高速度(140716 以后版本 SPEED 不起作用)。 如果要求进入本段的时候 FORCE_SPEED 已经降为对应的速度，请设置 STARTMOVE_SPEED。

指令 166	**ZAux_Direct_SetForceSpeed**
指令原型	int32 __stdcall ZAux_Direct_SetForceSpeed (ZMC_HANDLE handle , int iaxis, float fValue);
指令说明	设置 SP 的运行速度。
输入参数	参数名 | 描述 handle | 连接句柄。 iaxis | 轴号。 fValue | 设置的速度值。

输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	参考：轴保持输入应用
详细说明	这个参数被带入运动缓冲。 当 FORCE_SPEED 大于 SPEED 时，SPEED 同时也会生效限制住最高速度(140716 以后版本 SPEED 不起作用)。 如果要求进入本段的时候 FORCE_SPEED 已经降为对应的速度，请设置 STARTMOVE_SPEED。

指令 167	**ZAux_Direct_SetStartMoveSpeed**
指令原型	int32 __stdcall ZAux_Direct_SetStartMoveSpeed(ZMC_HANDLE handle, int iaxis, float fValue);
指令说明	自定义速度的 SP 运动的起始速度，这个参数被带入运动缓冲， 单位是 units/s。参见软件手册里面的“STARTMOVE_SPEED”指令。
输入参数	参数名 | 描述 handle | 连接句柄。 iaxis | 轴号。 fValue | 设置的速度值。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	连续插补 SP 运动
详细说明	只有使用了带 SP 的运动指令才生效。 不使用时请设置较大值。控制器默认值 1000。

指令 168	**ZAux_Direct_GetStartMoveSpeed**
指令原型	int32 __stdcall ZAux_Direct_GetStartMoveSpeed(ZMC_HANDLE handle, int iaxis, float * pfValue);
指令说明	读取自定义速度的 SP 运动的起始速度。
输入参数	参数名 | 描述 handle | 连接句柄。 iaxis | 轴号。
输出参数	参数名 | 描述 fValue | 读取的 SP 运动起始速度值。
返回值	成功返回值为 0，非 0 详见**错误码**说明。

指令示例	参考：连续插补 SP 运动
详细说明	只有使用了带 SP 的运动指令才生效。 不使用时请设置较大值。控制器默认值 1000。

指令 169	ZAux_Direct_GetFholdIn
指令原型	int32 __stdcall ZAux_Direct_GetFholdIn (ZMC_HANDLE handle , int iaxis, into*piValue);
指令说明	读取保持输入。
输入参数	参数名 | 描述 handle | 连接句柄。 iaxis | 轴号。
输出参数	参数名 | 描述 piValue | 返回输入 FHOLDIN 输入口编号。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	参考：ZAux_Direct_SetFhspeed 示例
详细说明	如果有输入信号，运动轴的速度由 FHSPEED 参数设置。当前运动没有被取消。当取消输入，过程中的运动速度返回程序速度。 输入 OFF 时，认为有信号输入，要相反效果可以用 INVERT_IN 反转电平(ECI 系列控制器除外)。 此参数只适用速度控制方式(带 SP 后缀指令)。如果运动不是速度控制（CAMBOX，CONNECT，MOVELINK），不会起作用。

指令 170	ZAux_Direct_SetFholdIn
指令原型	int32 __stdcall ZAux_Direct_SetFholdIn (ZMC_HANDLE handle, int iaxis, int iValue);
指令说明	设置保持输入。保持输入对应的输入点编号，-1 无效。如果有输入信号，运动轴的速度由 FHSPEED 参数设置。当前运动没有被取消。当取消输入，过程中的运动速度返回程序速度。只对 SP 运动有效
输入参数	参数名 | 描述 handle | 连接句柄。 iaxis | 轴号。 iValue | 设置输入口的编号。
返回值	成功返回值为 0，非 0 详见**错误码**说明。

指令示例	轴保持输入应用
详细说明	如果有输入信号，运动轴的速度由 FHSPEED 参数设置。当前运动没有被取消。当取消输入，过程中的运动速度返回程序速度。 输入 OFF 时，认为有信号输入，要相反效果可以用 INVERT_IN 反转电平(ECI 系列控制器除外)。 此参数只适用速度控制方式(带 SP 后缀指令)。如果运动不是速度控制（CAMBOX，CONNECT，MOVELINK），不会起作用。

指令 171	**ZAux_Direct_SetCornerMode**
指令原型	int32 __stdcall ZAux_Direct_SetCornerMode(ZMC_HANDLE handle, int iaxis, int iValue);
指令说明	设置拐角模式。
输入参数	参数名：描述 handle：连接句柄。 iaxis：轴号。 iValue：模式设置 位 / 值 / 描述 0 / 1 / 预留 1 / 2 / 自动拐角减速 2 / 4 / 预留 3 / 8 / 自动小圆限速
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	拐角减速

详细说明

mode：不同的位代表不同的意义，位可以同时使用。

位	值	描述
0	1	预留
1	2	自动拐角减速 按 ACCEL,DECEL 加减速度，此参数是在 MOVE 函数调用前生效 减速角度定义查看 DECEL_ANGLE 和 STOP_ANGLE 指令 减速拐角参考速度以 FORCE_SPEED 速度为参考，一定要设置合理的 FORCE_SPEED
2	4	预留
3	8	自动小圆限速，半径小于设置值时限速，大于限制值时不限速 此参数在 MOVE 函数调用前修改生效 限制速度按 FORCE_SPEED 限速= FORCE_SPEED * 实际半径/FULL_SP_RADIUS 限速半径 FULL_SP_RADIUS 设置
4	16	预留
5	32	自动倒角设置 此参数在 MOVE 函数调用前修改生效 此 MOVE 运动自动和前面的 MOVE 运动做倒角处理，倒角半径参考 ZSMOOTH 此倒角针对插补的所有轴，20150701 以后固件支持

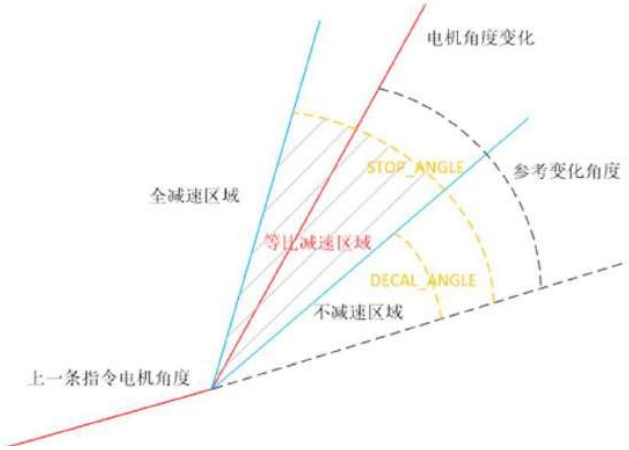
指令 172	**ZAux_Direct_GetCornerMode**
指令原型	int32 __stdcall ZAux_Direct_GetCornerMode (ZMC_HANDLE handle, int iaxis, int * piValue);
指令说明	读取拐角减速模式。

输入参数

参数名	描述
handle	连接句柄。
iaxis	轴号。

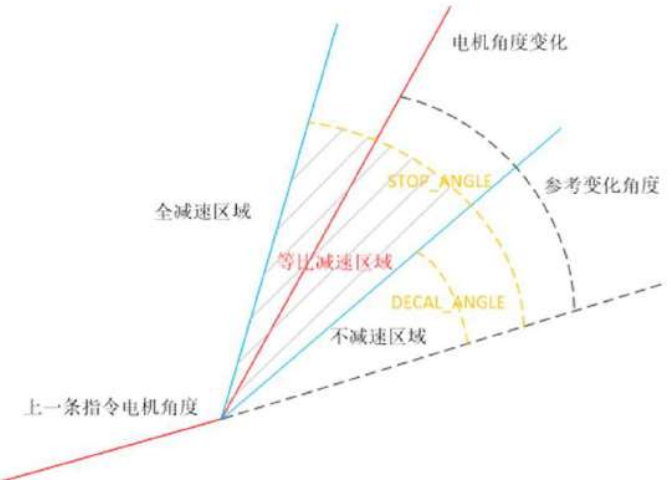
<table>
<tr><td>输出参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>iValue</td><td>模式设置。
<table>
<tr><td>位</td><td>值</td><td>描述</td></tr>
<tr><td>0</td><td>1</td><td>预留</td></tr>
<tr><td>1</td><td>2</td><td>自动拐角减速</td></tr>
<tr><td>2</td><td>4</td><td>预留</td></tr>
<tr><td>3</td><td>8</td><td>自动小圆限速</td></tr>
</table>
</td></tr>
</table>
</td></tr>
<tr><td>返回值</td><td>成功返回值为 0，非 0 详见错误码说明。</td></tr>
<tr><td>指令示例</td><td>拐角减速</td></tr>
<tr><td>详细说明</td><td>mode：不同的位代表不同的意义，位可以同时使用。
<table>
<tr><td>位</td><td>值</td><td>描述</td></tr>
<tr><td>0</td><td>1</td><td>预留</td></tr>
<tr><td>1</td><td>2</td><td>自动拐角减速
按 ACCEL,DECEL 加减速度，此参数是在 MOVE 函数调用前生效
减速角度定义查看 DECEL_ANGLE 和 STOP_ANGLE 指令
减速拐角参考速度以 FORCE_SPEED 速度为参考，一定要设置合理的 FORCE_SPEED</td></tr>
<tr><td>2</td><td>4</td><td>预留</td></tr>
<tr><td>3</td><td>8</td><td>自动小圆限速，半径小于设置值时限速，大于限制值时不限速
此参数在 MOVE 函数调用前修改生效
限制速度按 FORCE_SPEED
限速= FORCE_SPEED * 实际半径/FULL_SP_RADIUS
限速半径 FULL_SP_RADIUS 设置</td></tr>
<tr><td>4</td><td>16</td><td>预留</td></tr>
<tr><td>5</td><td>32</td><td>自动倒角设置
此参数在 MOVE 函数调用前修改生效
此 MOVE 运动自动和前面的 MOVE 运动做倒角处理，倒角半径参考 ZSMOOTH
此倒角针对插补的所有轴，20150701 以后固件支持</td></tr>
</table>
</td></tr>
</table>

指令 173	**ZAux_Direct_SetDecelAngle**
指令原型	int32 __stdcall ZAux_Direct_SetDecelAngle(ZMC_HANDLE handle, int iaxis, float fValue);
指令说明	设置拐角减速角度，开始减速角度，单位为弧度。

<table>
<tr><td>输入参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>handle</td><td>连接句柄。</td></tr>
<tr><td>iaxis</td><td>轴号。</td></tr>
<tr><td>fValue</td><td>设置的拐角减速角度。</td></tr>
</table>
</td></tr>
<tr><td>输出参数</td><td>/</td></tr>
<tr><td>返回值</td><td>成功返回值为 0，非 0 详见错误码说明。</td></tr>
<tr><td>指令示例</td><td>拐角减速</td></tr>
<tr><td>详细说明</td><td>减速拐角参考速度以 FORCE_SPEED 速度为参考，一定要设置合理的 FORCE_SPEED。
角度转换弧度公式：角度值*(PI/180)
减速角度是指电机的参考角度相对上一条运动的变化值。如下图。
此角度值不是实际轨迹的角度，是换算到电机变换的角度，此角度值仅为参考。
 

下一插补运动轨迹处于下方时取绝对值。
直线与圆弧相连时按照圆弧的切线方向计算角度。
与 STOP_ANGLE 一起使用，当实际运动的角度处于 DECEL_ANGLE（上限）与 STOP_ANGLE（下限）时才会减速。</td></tr>
</table>

<table>
<tr><td>指令 174</td><td>ZAux_Direct_GetDecelAngle</td></tr>
<tr><td>指令原型</td><td>int32 __stdcall ZAux_Direct_GetDecelAngle (ZMC_HANDLE handle, int iaxis, float *pfValue);</td></tr>
<tr><td>指令说明</td><td>读取拐角起始减速角度，单位为弧度。</td></tr>
<tr><td>输入参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>handle</td><td>连接句柄。</td></tr>
<tr><td>iaxis</td><td>轴号。</td></tr>
</table>
</td></tr>
</table>

输出参数	<table><tr><td>参数名</td><td>描述</td></tr><tr><td>fValue</td><td>读取的拐角减速角度。</td></tr></table>
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	参考：拐角减速
详细说明	减速拐角参考速度以 FORCE_SPEED 速度为参考，一定要设置合理的 FORCE_SPEED。 角度转换弧度公式：角度值*(PI/180) 减速角度是指电机的参考角度相对上一条运动的变化值。如下图。 此角度值不是实际轨迹的角度，是换算到电机变换的角度，此角度值仅为参考。  下一插补运动轨迹处于下方时取绝对值。 直线与圆弧相连时按照圆弧的切线方向计算角度。 与 STOP_ANGLE 一起使用，当实际运动的角度处于 DECEL_ANGLE（上限）与 STOP_ANGLE（下限）时才会减速。

指令 175	**ZAux_Direct_SetFullSpRadius**
指令原型	int32 __stdcall ZAux_Direct_SetFullSpRadius(ZMC_HANDLE handle, int iaxis, float fValue);
指令说明	设置小圆限速半径。
输入参数	<table><tr><td>参数名</td><td>描述</td></tr><tr><td>handle</td><td>连接句柄。</td></tr><tr><td>iaxis</td><td>轴号。</td></tr><tr><td>fValue</td><td>小圆半径。</td></tr></table>
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。

指令示例	平面圆心画圆+小圆限速
详细说明	当半径大于小圆限速半径时，速度是用户程序指定的速度值， 当半径小于小圆限速半径时，控制器会按比例减小速度。

指令 176	ZAux_Direct_GetFullSpRadius
指令原型	int32 __stdcall ZAux_Direct_GetFullSpRadius (ZMC_HANDLE handle , int iaxis, float *pfValue);
指令说明	读取小圆限速最小半径。
输入参数	参数名 | 描述 handle | 连接句柄。 iaxis | 轴号。
输出参数	参数名 | 描述 fValue | 返回的限速半径。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	平面圆心画圆+小圆限速
详细说明	当半径大于小圆限速半径时，速度是用户程序指定的速度值， 当半径小于小圆限速半径时，控制器会按比例减小速度。

指令 177	ZAux_Direct_GetInterpFactor
指令原型	int32 __stdcall ZAux_Direct_GetInterpFactor (ZMC_HANDLE handle , int iaxis, int *piValue);
指令说明	读取插补时轴是否参与速度计算,缺省参与(1)。此参数支队直线和螺旋的第三个轴起作用。
输入参数	参数名 | 描述 handle | 连接句柄。 iaxis | 轴号。
输出参数	参数名 | 描述 piValue | 返回的速度计算模式。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	/
详细说明	此参数只对直线的任意轴和螺旋的第三个轴起作用。运动后请及时还原，否则会导致后续运动不正确。 部分轴不参与速度计算时，先按参与计算的轴的插补计算出参与轴的分速

	度和运动总时间，再把不参与计算的轴的运动距离依次除以总时间，得到不参与计算的各轴的相应速度。 不能把插补的实际运动的所有轴都设置 0，会导致实际速度无穷大。

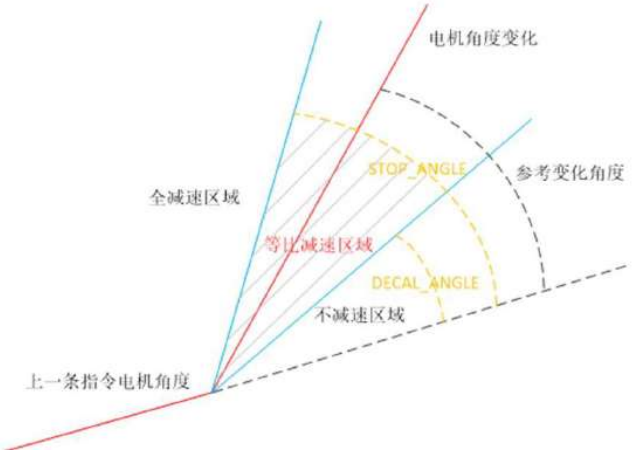
指令 178	ZAux_Direct_SetInterpFactor
指令原型	int32 __stdcall ZAux_Direct_SetInterpFactor (ZMC_HANDLE handle , int iaxis, int iValue);
指令说明	设置插补时轴是否参与速度计算，缺省参与(1)。此参数只对直线和螺旋的第三个轴起作用。
输入参数	参数名 | 描述 handle | 连接句柄。 iaxis | 轴号。 piValue | 模式：0-不参与 1-参与。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	/
详细说明	此参数只对直线的任意轴和螺旋的第三个轴起作用。运动后请及时还原，否则会导致后续运动不正确。 部分轴不参与速度计算时，先按参与计算的轴的插补计算出参与轴的分速度和运动总时间，再把不参与计算的轴的运动距离依次除以总时间，得到不参与计算的各轴的相应速度。 不能把插补的实际运动的所有轴都设置 0，会导致实际速度无穷大。

指令 179	ZAux_Direct_SetMerge
指令原型	int32 __stdcall ZAux_Direct_SetMerge(ZMC_HANDLE handle, int iaxis, int iValue)
指令说明	设置连续插补开关。
输入参数	参数名 | 描述 handle | 连接句柄。 iaxis | 轴号。 piValue | 0-关闭 1-打开。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	拐角减速

详细说明	当 MERGE 设置为 ON 时，多段插补间仍减速，可能原因如下： 1.可能 MERGE 并没有设置成功，可以打印查看。 2.控制器是点位运动型号，不支持连续插补，可联系厂家。 3.设置了 CORNER_MODE 拐角减速，打印确认。 4. 使用了带 SP 的运动指令，并设置了 ENDMOVE_SPEED，STARTMOVE_SPEED，此时速度由这两条指令确定。 5.多条插补间切换了主轴，主轴速度参数改变了。 6.多条插补间加入了 MOVE_DELAY 延时指令，即使延时写 0，也会导致减速。

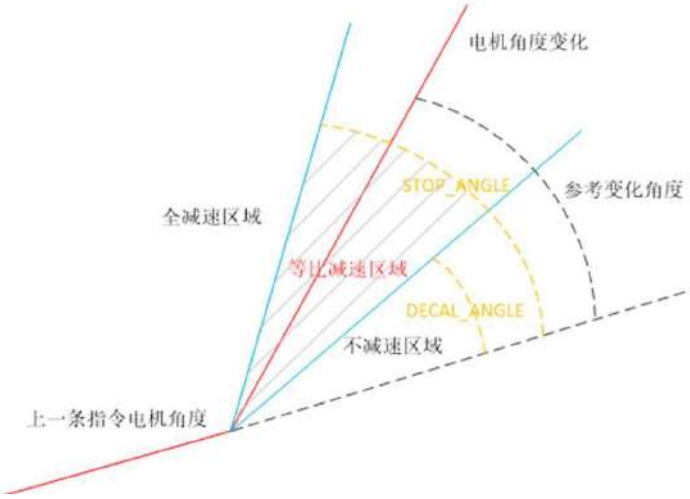
指令 180	**ZAux_Direct_GetMerge**
指令原型	int32 __stdcall ZAux_Direct_GetMerge(ZMC_HANDLE handle, int iaxis, int *piValue)
指令说明	读取连续插补开关状态。
输入参数	参数名：描述 handle：连接句柄。 iaxis：轴号。
输出参数	参数名：描述 piValue：获取的连续插补状态：0-关闭 1-打开。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	参考：拐角减速
详细说明	当 MERGE 设置为 ON 时，多段插补间仍减速，可能原因如下： 1.可能 MERGE 并没有设置成功，可以打印查看。 2.控制器是点位运动型号，不支持连续插补，可联系厂家。 3.设置了 CORNER_MODE 拐角减速，打印确认。 4. 使用了带 SP 的运动指令，并设置了 ENDMOVE_SPEED，STARTMOVE_SPEED，此时速度由这两条指令确定。 5.多条插补间切换了主轴，主轴速度参数改变了。 6.多条插补间加入了 MOVE_DELAY 延时指令，即使延时写 0，也会导致减速。

指令 181	**ZAux_Direct_SetStopAngle**
指令原型	int32 __stdcall ZAux_Direct_SetStopAngle(ZMC_HANDLE handle, int iaxis, float pfValue);

<table>
<tr><td>指令说明</td><td>设置停止减速角度。</td></tr>
<tr><td>输入参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>handle</td><td>连接句柄。</td></tr>
<tr><td>iaxis</td><td>轴号。</td></tr>
<tr><td>pfValue</td><td>停止减速角度。</td></tr>
</table>
</td></tr>
<tr><td>输出参数</td><td>/</td></tr>
<tr><td>返回值</td><td>成功返回值为 0，非 0 详见错误码说明。</td></tr>
<tr><td>指令示例</td><td>拐角减速</td></tr>
<tr><td>详细说明</td><td>减速拐角参考速度以 FORCE_SPEED 速度为参考，一定要设置合理的 FORCE_SPEED。
角度转换弧度公式：角度值*(PI/180)
减速角度是指电机的参考角度相对上一条运动的变化值。如下图。
此角度值不是实际轨迹的角度，是换算到电机变换的角度，此角度值仅为参考。
 

下一插补运动轨迹处于下方时取绝对值。
直线与圆弧相连时按照圆弧的切线方向计算角度。
与 STOP_ANGLE 一起使用，当实际运动的角度处于 DECEL_ANGLE（上限）与 STOP_ANGLE（下限）时才会减速。</td></tr>
</table>

指令 182	**ZAux_Direct_GetStopAngle**
指令原型	int32 __stdcall ZAux_Direct_GetStopAngle (ZMC_HANDLE handle , int iaxis, float * pfValue);
指令说明	拐角停止减速的角度，单位是弧度。减速拐角参考速度以 FORCE_SPEED 速度为参考，一定要设置合理的 FORCE_SPEED。

<table>
<tr><td>输入参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>handle</td><td>连接句柄。</td></tr>
<tr><td>iaxis</td><td>轴号。</td></tr>
</table>
</td></tr>
<tr><td>输出参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>pfValue</td><td>返回的拐角停止角度</td></tr>
</table>
</td></tr>
<tr><td>返回值</td><td>成功返回值为 0，非 0 详见错误码说明。</td></tr>
<tr><td>指令示例</td><td>参考：拐角减速</td></tr>
<tr><td>详细说明</td><td>减速拐角参考速度以 FORCE_SPEED 速度为参考，一定要设置合理的 FORCE_SPEED。
角度转换弧度公式：角度值*(PI/180)
减速角度是指电机的参考角度相对上一条运动的变化值。如下图。
此角度值不是实际轨迹的角度，是换算到电机变换的角度，此角度值仅为参考。
 

下一插补运动轨迹处于下方时取绝对值。
直线与圆弧相连时按照圆弧的切线方向计算角度。
与 STOP_ANGLE 一起使用，当实际运动的角度处于 DECEL_ANGLE（上限）与 STOP_ANGLE（下限）时才会减速。</td></tr>
</table>

指令 183	**ZAux_Direct_SetZsmooth**
指令原型	int32 __stdcall ZAux_Direct_SetZsmooth (ZMC_HANDLE handle, int iaxis, float fValue);
指令说明	设置减速倒角半径。

输入参数	<table><tr><td>参数名</td><td>描述</td></tr><tr><td>handle</td><td>连接句柄。</td></tr><tr><td>iaxis</td><td>轴号。</td></tr><tr><td>pfValue</td><td>设置的值。</td></tr></table>
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	倒角
详细说明	根据拐角角度自动计算实际拐角半径大小。超过路径时限制为 50%。 90°时拐角半径为设置值。

指令 184	**ZAux_Direct_GetZsmooth**
指令原型	int32 __stdcall ZAux_Direct_GetZsmooth(ZMC_HANDLE handle, int iaxis, float *pfValue);
指令说明	全局倒角半径读取。
输入参数	共有 2 个输入参数，见下方说明。
输入参数	<table><tr><td>参数名</td><td>描述</td></tr><tr><td>handle</td><td>连接句柄。</td></tr><tr><td>iaxis</td><td>轴号。</td></tr></table>
输出参数	<table><tr><td>参数名</td><td>描述</td></tr><tr><td>pfValue</td><td>返回的倒角半径。</td></tr></table>
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	参考：倒角
详细说明	根据拐角角度自动计算实际拐角半径大小。超过路径时限制为 50%。 90°时拐角半径为设置值。

指令 185	**ZAux_Direct_MovePause**
指令原型	int32 __stdcall ZAux_Direct_MovePause(ZMC_HANDLE handle,int iaxis, int imode)
指令说明	BASE 轴运动暂停，只有在单轴或多轴插补运动时有效，多轴联动时一起暂停。

<table>
<tr><td>输入参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>handle</td><td>连接句柄。</td></tr>
<tr><td>iaxis</td><td>轴号。</td></tr>
<tr><td>imode</td><td>暂停方式：
0 暂停当前运动。
1 在当前运动完成后正准备执行下一条运动指令时暂停。
2 在当前运动完成后正准备执行下一条运动指令时，并且两条指令的 MARK 标识不一样时暂停。这个模式可以用于一个动作由多个指令来实现时，可以在一整个动作完成后暂停。
3 强制暂停，IDLE 模式下也可以进入暂停状态 20170513 固件版本增加此功能</td></tr>
</table>
</td></tr>
<tr><td>输出参数</td><td>/</td></tr>
<tr><td>返回值</td><td>成功返回值为 0，非 0 详见错误码说明。</td></tr>
<tr><td>指令示例</td><td>轴暂停与恢复</td></tr>
<tr><td>详细说明</td><td>1.只有在单轴或多轴插补运动时有效，多轴联动时一起暂停。
2.可以通过 AXISSTATUS 来查询是否有暂停。
3.当轴已经暂停或不在运动中时，调用这个指令会有警告输出，但不影响程序运行。某些运动不支持暂停，如 VMOVE、同步运动指令等</td></tr>
</table>

<table>
<tr><td>指令 186</td><td>ZAux_Direct_Rapidstop</td></tr>
<tr><td>指令原型</td><td>int32 __stdcall ZAux_Direct_Rapidstop(ZMC_HANDLE handle, int imode)</td></tr>
<tr><td>指令说明</td><td>所有轴列表立即停止，如果轴参与插补，也停止插补运动。</td></tr>
<tr><td>输入参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>handle</td><td>连接句柄。</td></tr>
<tr><td>imode</td><td>模式
0 取消当前运动
1 取消缓冲的运动
2 取消当前运动和缓冲运动
3 立即中断脉冲发送</td></tr>
</table>
</td></tr>
<tr><td>输出参数</td><td>/</td></tr>
<tr><td>返回值</td><td>成功返回值为 0，非 0 详见错误码说明。</td></tr>
<tr><td>指令示例</td><td>/</td></tr>
<tr><td>详细说明</td><td>Mode0~2 减速度按 FASTDEC 和 DECEL 中最大的值。
RAPIDSTOP 后要调用绝对位置运动，必须先 WAIT IDLE 等待停止完成。</td></tr>
</table>

指令 187	ZAux_Direct_MoveResume
指令原型	int32 __stdcall ZAux_Direct_MoveResume (ZMC_HANDLE handle , int iaxis);
指令说明	取消运动暂停。
输入参数	参数名 | 描述 handle | 连接句柄。 iaxis | 轴号。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	轴暂停与恢复
详细说明	当 BASE 轴暂停暂停时，从暂停处继续运动。 可以通过 AXISSTATUS 来查询是否有暂停。

指令 188	ZAux_Direct_CancelAxisList
指令原型	int32 __stdcall ZAux_Direct_CancelAxisList(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, int imode)
指令说明	轴列表所有轴立即停止，如果轴参与插补，也停止插补运动。
输入参数	参数名 | 描述 handle | 连接标识。 imaxaxises | 运动的总轴数。 piAxislist | 轴列表数组。 imode | 模式 0 取消当前运动 1 取消缓冲的运动 2 取消当前运动和缓冲运动 3 立即停止
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	/
详细说明	Mode0~2 减速度按 FASTDEC 和 DECEL 中最大的值。

指令 189	ZAux_Direct_Single_Cancel
指令原型	int32 __stdcall ZAux_Direct_Single_Cancel (ZMC_HANDLE handle, int

	iaxis, int imode);
指令说明	单轴运动停止。
输入参数	参数名：handle — 连接标识。 iaxis — 轴号。 imode — 模式： 0 （缺省）取消当前运动 1 取消缓冲的运动 2 取消当前运动和缓冲运动 3 立即中断脉冲发送
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	单轴持续运动
详细说明	如果指定轴在插补运动轴列表中，无论主轴或者其他轴，都停止轴组的插补运动 MODE0~2 减速度按 FASTDEC 和 DECEL 中最大的值。 CANCEL 后要调用绝对位置运动，必须先等待停止完成。

指令 190	**ZAux_Direct_GetRepDist**
指令原型	int32 __stdcall ZAux_Direct_GetRepDist (ZMC_HANDLE handle , int iaxis, float *pfValue);
指令说明	读取根据 REP_OPTION 设置来自动循环轴 DPOS 和 MPOS 坐标。
输入参数	参数名：handle — 连接标识。 iaxis — 轴号。
输出参数	参数名：pfValue — 返回的循环坐标值。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	参考：坐标循环
详细说明	/

指令 191	**ZAux_Direct_SetRepDist**
指令原型	int32 __stdcall ZAux_Direct_SetRepDist (ZMC_HANDLE handle , int iaxis, float fValue);

指令说明	根据 REP_OPTION 设置来自动循环轴 DPOS 和 MPOS 坐标。
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。 pfValue | 设置的循环坐标值。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	参考：坐标循环
详细说明	/

指令 192	**ZAux_Direct_GetRepOption**
指令原型	int32 __stdcall ZAux_Direct_GetRepOption (ZMC_HANDLE handle , int iaxis, int *piValue);
指令说明	读取坐标重复设置。
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。
输出参数	参数名 | 描述 piValue | 返回的模式 位 | 值 | 描述 0 | 1 | 0:循环范围– REP_DIST 到 + REP_DIST 1:循环范围为 0 到 +REP_DIST 1 | 2 | 1: 禁止 CAMBOX 和 MOVELINK 的重复运动，禁止生效后还原为 0 2 | 4 | 预留 4 | 16 | 1:不使用 REP_DIST， 0:使用 REP_DIST
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	参考：坐标循环
详细说明	可用于限制凸轮主轴的坐标循环范围，来实现多条凸轮曲线连续。 使用绝对运动模式时，如果目标位置处于坐标循环范围内，可以正确运动； 如果处于坐标循环范围外，运动不正确。

	相对运动不受影响。

指令 193	ZAux_Direct_SetRepOption
指令原型	int32 __stdcall ZAux_Direct_SetRepOption (ZMC_HANDLE handle , int iaxis, int iValue);
指令说明	设置坐标循环模式。
输入参数	参数名：handle 描述：连接标识。 参数名：iaxis 描述：轴号。 参数名：iValue 描述：按位来表示不同的意义 位 0，值 1，描述：0:循环范围– REP_DIST 到 + REP_DIST 1:循环范围为 0 到 + REP_DIST 位 1，值 2，描述：1:禁止 CAMBOX 和 MOVELINK 的重复运动，禁止生效后还原为 0 位 2，值 4，描述：预留 位 4，值 16，描述：1:不使用 REP_DIST， 0:使用 REP_DIST
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	坐标循环
详细说明	可用于限制凸轮主轴的坐标循环范围，来实现多条凸轮曲线连续。 使用绝对运动模式时，如果目标位置处于坐标循环范围内，可以正确运动；如果处于坐标循环范围外，运动不正确。 相对运动不受影响。

指令 194	ZAux_Direct_Pitchset
指令原型	int32 __stdcall ZAux_Direct_Pitchset(ZMC_HANDLE handle,int iaxis,int iEnable,float StartPos,uint32 maxpoint,float DisOne ,uint32 TablNum,float * pfDisancelist)
指令说明	设置轴的螺距补偿，扩展轴无效。

输入参数	参数名 | 描述 handle | 连接句柄。 iaxis | 轴号 iEnable | 是否启用补偿，1：启动 0：关闭 StartPos | 起始补偿 MPOS 位置 maxpoint | 补偿区间总点数 DisOne | 每个补偿点间距 TablNum | 补偿坐标值填入 TABLE 系统数组起始引导地址 pfDisancelist | 区间补偿值列表
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	螺距补偿
详细说明	每点的补偿脉冲个数存储在 TABLE 表里面。

指令 195	**ZAux_Direct_Pitchset2**
指令原型	int32 __stdcall ZAux_Direct_Pitchset2(ZMC_HANDLE handle,int iaxis,int iEnable,float StartPos,uint32 maxpoint,float DisOne ,uint32 TablNum,float * pfDisancelist ,uint32 RevTablNum,float * RevpfDisancelist)
指令说明	设置轴的螺距补偿 2，扩展轴无效。
输入参数	参数名 | 描述 handle | 连接句柄。 iaxis | 轴号 iEnable | 是否启用补偿，1：启动 0：关闭 StartPos | 起始补偿 MPOS 位置 maxpoint | 补偿区间点数 DisOne | 每个补偿点间距 TablNum | 补偿坐标值填入 TABLE 系统数组起始引导地址 pfDisancelist | 区间补偿值列表 RevTablNum | 反向补偿坐标值填入 TABLE 系统数组起始引导地址 RevpfDisancelist | 反向区间补偿值列表 补偿数据方向于正向方向一致
输出参数	/

返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	/
详细说明	/

指令 196	**ZAux_Direct_GetPitchStatus**
指令原型	int32 __stdcall ZAux_Direct_GetPitchStatus(ZMC_HANDLE handle,int iaxis,int * IfEnable,float * PitchDist)
指令说明	设置轴的螺距补偿状态读取，扩展轴无效。
输入参数	参数名 | 描述 handle | 连接句柄。 iaxis | 轴号
输出参数	参数名 | 描述 iEnable | 是否启用补偿，1：启动 0：关闭 PitchDist | 补偿距离
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	/
详细说明	/

指令 197	**ZAux_Direct_GetIn**
指令原型	int32 __stdcall ZAux_Direct_GetIn(ZMC_HANDLE handle,int ionum,uint32 *piValue)
指令说明	读取输入口状态， 参见软件手册里面的“IN”指令。
输入参数	参数名 | 描述 handle | 连接句柄。 ionum | 输入口编号。
输出参数	参数名 | 描述 *piValue | 获取输入口的状态值。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	IO 读取与设置
详细说明	1.若设置了翻转，读取的是 INVERT_IN 翻转以后的状态。 2.ZIO 扩展板的 IO 通道号与拨码有关，起始值为（16+拨码组合值*16），EIO 总线扩展 IO 使用 NODE_IO 指令，只能设置为 8 的倍数，详细查看

	硬件手册。 注意：IO 映射编号要大于控制器自身最大的 IO 编号，不能与控制器的编号重合。

指令 198	ZAux_Direct_SetOp
指令原型	int32 __stdcall ZAux_Direct_SetOp(ZMC_HANDLE handle, int ionum,uint32 iValue);
指令说明	打开输出口，参见软件手册里面的“OP”指令。
输入参数	参数名 | 描述 handle | 连接句柄。 ionum | 输出口编号。 piValue | 设置的输出口的状态值。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	IO 读取与设置
详细说明	ZIO 扩展板的 IO 通道号与拨码有关，起始值为（16 +拨码组合值*16），EIO 总线扩展 IO 使用 NODE_IO 指令，只能设置为 8 的倍数，详细查看硬件手册。 注意 IO 映射编号要大于控制器自身最大的 IO 编号，不能与控制器的编号重合。

指令 199	ZAux_Direct_GetOp
指令原型	int32 __stdcall ZAux_Direct_GetOp(ZMC_HANDLE handle, int ionum, uint32 *piValue);
指令说明	读取输出口，参见软件手册里面的“OP”指令。
输入参数	参数名 | 描述 handle | 连接句柄。 ionum | 输入口编号。
输出参数	参数名 | 描述 piValue | 获取输出口的状态值。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	IO 读取与设置
*piValue	返回状态值
详细说明	ZIO 扩展板的 IO 通道号与拨码有关，起始值为（16 +拨码组合值*16），

	EIO 总线扩展 IO 使用 NODE_IO 指令，只能设置为 8 的倍数，详细查看硬件手册。 注意 IO 映射编号要大于控制器自身最大的 IO 编号，不能与控制器的编号重合。 最多可操作 32 个输出口。

指令 200	ZAux_GetModbusIn
指令原型	int32 __stdcall ZAux_GetModbusIn (ZMC_HANDLE handle,int ionumfirst, int ionumend, uint8 * pValueList);
指令说明	快速读取多个当前的输入状态。
输入参数	参数名：描述 handle：连接句柄。 ionumfirst：IN 起始编号。 ionumend：IN 结束编号。
输出参数	参数名：描述 pValueList：位状态，按位存储，一个 uint8 对应 8 个输入口状态，可读取 8 个及以上。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	多个 IO 读取
详细说明	Modbus 方式获取出来的状态是未反转前的状态。如果有用 INVERT_IN 反转，读取的状态可能就是不对的

指令 201	ZAux_GetModbusOut
指令原型	int32 __stdcall ZAux_GetModbusOut (ZMC_HANDLE handle,int ionumfirst, int ionumend, uint8 * pValueList);
指令说明	快速读取多个当前的输出状态。Modbus 方式获取出来的状态是未反转前的状态。
输入参数	参数名：描述 handle：连接句柄。 ionumfirst：OUT 起始编号。 ionumend：OUT 结束编号。

输出参数	参数名 | 描述 pValueList | 位状态，按位存储，一个 uint8 对应 8 个输出口状态，可读取 8 个及以上。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	多个 IO 读取
详细说明	

指令 202	**ZAux_Direct_SetOutMulti**
指令原型	int32 __stdcall ZAux_SetOutMulti (ZMC_HANDLE handle, uint16 iofirst, uint16 ioend, uint32 *istate);
指令说明	IO 接口设置多路输出。
输入参数	参数名 | 描述 handle | 连接句柄。 ionumfirst | OUT 起始编号。 ionumend | OUT 结束编号。 istate | 输出口状态，istate 按位设置，一个 UNIT 对应 32 个输出口状态。可设置 32 个及以上。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	多个 IO 读取设置
详细说明	/

指令 203	**ZAux_Direct_GetOutMulti**
指令原型	int32 __stdcall ZAux_direct_GetOutMulti (ZMC_HANDLE handle, uint16 iofirst, uint16 ioend, uint32 * istate);
指令说明	IO 接口获取多路输出状态。
输入参数	参数名 | 描述 handle | 连接句柄。 iofirst | IO 口起始编号 ioend | IO 口结束编号

输出参数	参数名 | 描述 istate | 按位获取 IO 口的状态。一个 UNIT 存储 32 个输出口状态。可读取 32 个及以上。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	多个 IO 读取设置
详细说明	/

指令 204	**ZAux_Direct_GetInMulti**
指令原型	int32 __stdcall ZAux_Direct_GetInMulti (ZMC_HANDLE handle, int startio , int endio, int32 *piValue);
指令说明	IO 接口读取多路输入。
输入参数	参数名 | 描述 handle | 连接句柄。 startio | IO 口起始编号 endio | IO 口结束编号
输出参数	参数名 | 描述 piValue | 按位获取 IO 口的状态。一个 UNIT 存储 32 个输入口状态。可读取 32 个及以上。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	多个 IO 读取设置
详细说明	/

指令 205	**ZAux_Direct_SetInvertIn**
指令原型	int32 __stdcall ZAux_Direct_SetInvertIn(ZMC_HANDLE handle, int ionum, int bifInvert);
指令说明	设置反转输入状态，参见软件手册里面的“ INVERT_IN”指令。
输入参数	参数名 | 描述 handle | 连接句柄。 ionum | 输入口编号。 BifInvert | 状态 0-反转关 1-反转开。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	正负硬限位设置

详细说明	ZMC 系列控制器输入 OFF 时认为有信号输入(ECI 系列控制器与之相反)

指令 206	ZAux_Direct_GetInvertIn
指令原型	int32 __stdcall ZAux_Direct_GetInvertIn(ZMC_HANDLE handle, int ionum, int * piValue);
指令说明	读取反转输入状态。
输入参数	参数名 | 描述 handle | 连接句柄。 ionum | 输入口编号。
输入参数	参数名 | 描述 piValue | 返回反转输入的状态。 0-反转关 1-反转开
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	参考：正负硬限位设置
详细说明	ZMC 系列控制器输入 OFF 时认为有信号输入(ECI 系列控制器与之相反)

指令 207	ZAux_Direct_GetAD
指令原型	int32 __stdcall ZAux_Direct_GetAD(ZMC_HANDLE handle, int ionum , float *pfValue);
指令说明	读取模拟量输入值，参见软件手册里面的“AIN”指令。
输入参数	参数名 | 描述 handle | 连接句柄。 ionum | AIN 口编号。
输入参数	参数名 | 描述 pfValue | 返回模拟量值
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	DA/AD 的设置与读取
详细说明	12 位刻度值范围 0~4095 对应 0~10V 电压。 16 位刻度值范围 0~65536 对应 0~10V 电压。 ZAIO 扩展板的 AD 通道号与拨码有关，起始值为（8 +拨码组合值*8），ZMIO 总线扩展 AD 使用 NODE_AIO 指令只能设置为 8 的倍数，详细查看硬件手册。 注意：AIO 映射编号要大于控制器自身最大的 AIO 编号，不能与控制器

	的编号重合。

指令 208	ZAux_Direct_SetDA
指令原型	int32 __stdcall ZAux_Direct_SetDA(ZMC_HANDLE handle, int ionum, float fValue);
指令说明	打开模拟量输出口，参见软件手册里面的“AOUT”指令。
输入参数	参数名 | 描述 handle | 连接句柄。 ionum | DA 输出口编号。 fValue | 设置值
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	DA/AD 的设置与读取
详细说明	12 位刻度值范围 0~4095 对应 0~10V 电压。 16 位刻度值范围 0~65536 对应 0~10V 电压。

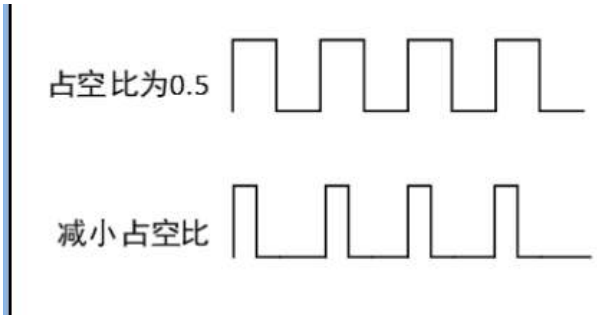
指令 209	ZAux_Direct_GetDA
指令原型	int32 __stdcall ZAux_Direct_GetDA(ZMC_HANDLE handle, int ionum, float *pfValue);
指令说明	读取模拟量输入口值，参见软件手册里面的“ AOUT”指令。
输入参数	参数名 | 描述 handle | 连接句柄。 ionum | DA 输出口编号。
输出参数	参数名 | 描述 pfValue | 返回设置的输出值
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	DA/AD 的设置与读取
详细说明	12 位刻度值范围 0~4095 对应 0~10V 电压。 16 位刻度值范围 0~65536 对应 0~10V 电压。

指令 210	ZAux_Direct_SetPwmFreq
指令原型	int32 __stdcall ZAux_Direct_SetPwmFreq(ZMC_HANDLE handle, int ionum, float fValue);
指令说明	pwm 频率设置，参见软件手册里面的“PWM_FREQ”指令。

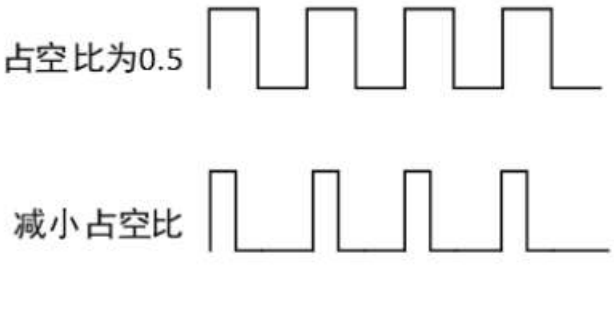
输入参数	<table><tr><td>参数名</td><td>描述</td></tr><tr><td>handle</td><td>连接句柄。</td></tr><tr><td>ionum</td><td>PWM 口编号。</td></tr><tr><td>fValue</td><td>设置值</td></tr></table>
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	PWM 使用
详细说明	PWM 只能通过设置占空比为 0 来关闭，不能通过设置 PWM 频率为 0 实现，不要将频率设为 0，PWM 频率一定要在 PWM 开关之前调整。

指令 211	**ZAux_Direct_GetPwmFreq**
指令原型	int32 __stdcall ZAux_Direct_GetPwmFreq(ZMC_HANDLE handle, int ionum, float *pfValue);
指令说明	pwm 频率读取。
输入参数	<table><tr><td>参数名</td><td>描述</td></tr><tr><td>handle</td><td>连接句柄。</td></tr><tr><td>ionum</td><td>PWM 口编号。</td></tr></table>
输出参数	<table><tr><td>参数名</td><td>描述</td></tr><tr><td>fValue</td><td>设置值</td></tr></table>
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	PWM 使用
详细说明	PWM 只能通过设置占空比为 0 来关闭，不能通过设置 PWM 频率为 0 实现，不要将频率设为 0，PWM 频率一定要在 PWM 开关之前调整。

指令 212	**ZAux_Direct_SetPwmDuty**
指令原型	int32 __stdcall ZAux_Direct_SetPwmDuty(ZMC_HANDLE handle, int ionum, float fValue);
指令说明	pwm 占空比设置，参见软件手册里面的“PWM_DUTY”指令。
输入参数	<table><tr><td>参数名</td><td>描述</td></tr><tr><td>handle</td><td>连接句柄。</td></tr><tr><td>ionum</td><td>PWM 口编号。</td></tr><tr><td>fValue</td><td>设置值</td></tr></table>
输出参数	/

返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	PWM 使用
详细说明	PWM 只能通过设置占空比为 0 来关闭，不能通过设置 PWM 频率为 0 实现，PWM 频率一定要在 PWM 开关之前调整。 占空比指有效电平占整个周期的比例。 一个周期中先输出有效电平，再输出无效电平。  PWM 的实际输出受输出口的控制，必须输出口打开，PWM 才能输出，否则输出被屏蔽掉。可以先开 PWM 功能，然后再开输出口，这样实现激光电源的首脉冲抑制功能。

指令 213	**ZAux_Direct_GetPwmDuty**
指令原型	int32 __stdcall ZAux_Direct_GetPwmDuty(ZMC_HANDLE handle, int ionum, float *pfValue);
指令说明	pwm 占空比读取。
输入参数	参数名 | 描述 handle | 连接句柄。 ionum | PWM 口编号。
输出参数	参数名 | 描述 fValue | 设置值
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	PWM 使用
详细说明	PWM 只能通过设置占空比为 0 来关闭，不能通过设置 PWM 频率为 0 实现，PWM 频率一定要在 PWM 开关之前调整。 占空比指有效电平占整个周期的比例。 一个周期中先输出有效电平，再输出无效电平。



PWM 的实际输出受输出口的控制，必须输出口打开，PWM 才能输出，否则输出被屏蔽掉。可以先开 PWM 功能，然后再开输出口，这样实现激光电源的首脉冲抑制功能。

指令 214	ZAux_Direct_SetFsLimit
指令原型	int32 __stdcall ZAux_Direct_SetFsLimit(ZMC_HANDLE handle, int iaxis, float fValue)
指令说明	设置轴正向软限位，取消软限位时设置一个较大的值即可。单位 units。
输入参数	参数名 / 描述 handle / 连接标识。 iaxis / 轴号。 pfValue / 设置的正向限位值。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	正负软限位设置
详细说明	当 FS_LIMIT 大于 REP_DIST 时，参数不起作用，正向软限位被禁止。 取消软限位时，建议不要去修改 REP_DIST 的值，将 FS_LIMIT 设置一个较大值即可。FS_LIMIT 的值默认为 200000000。 软限位无法作为 DATUM 回零时的限位信号参考。 注：设置限位前需检查有无设置对应的加减速度，以及快减减速度，若无设置，碰到限位无法停止

指令 215	ZAux_Direct_GetFsLimit
指令原型	int32 __stdcall ZAux_Direct_GetFsLimit(ZMC_HANDLE handle, int iaxis, float *fValue)
指令说明	读取正向软限位值。

输入参数	<table><tr><td>参数名</td><td>描述</td></tr><tr><td>handle</td><td>连接标识。</td></tr><tr><td>iaxis</td><td>轴号。</td></tr></table>
输出参数	<table><tr><td>参数名</td><td>描述</td></tr><tr><td>pfValue</td><td>返回的正向限位值。</td></tr></table>
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	正负软限位设置
详细说明	当 FS_LIMIT 大于 REP_DIST 时，参数不起作用，正向软限位被禁止。 取消软限位时，建议不要去修改 REP_DIST 的值，将 FS_LIMIT 设置一个较大值即可。FS_LIMIT 的值默认为 200000000。 软限位无法作为 DATUM 回零时的限位信号参考。 注：设置限位前需检查有无设置对应的加减速度，以及快减减速度，若无设置，碰到限位无法停止

指令 216	**ZAux_Direct_SetRsLimit**
指令原型	int32 __stdcall ZAux_Direct_SetRsLimit(ZMC_HANDLE handle, int iaxis, float fValue)
指令说明	设置轴负向软限位，取消软限位时设置一个较大的值即可。单位 units。
输入参数	<table><tr><td>参数名</td><td>描述</td></tr><tr><td>handle</td><td>连接标识。</td></tr><tr><td>iaxis</td><td>轴号。</td></tr><tr><td>pfValue</td><td>设置的负向限位值。</td></tr></table>
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	正负软限位设置
详细说明	当 RS_LIMIT 大于 REP_DIST 时，参数不起作用，负向软限位被禁止。 取消软限位时，建议不要去修改 REP_DIST 的值，将 RS_LIMIT 设置一个较大值即可。RS_LIMIT 的值默认为-200000000。 软限位无法作为 DATUM 回零时的限位信号参考。 注：设置限位前需检查有无设置对应的加减速度，以及快减减速度，若无设置，碰到限位无法停止

指令 217	**ZAux_Direct_GetRsLimit**
指令原型	int32 __stdcall ZAux_Direct_GetRsLimit(ZMC_HANDLE handle, int

	iaxis, float *fValue)
指令说明	读取负向软限位值。
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。
输出参数	参数名 | 描述 pfValue | 返回的负向限位值。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	正负软限位设置
详细说明	当 RS_LIMIT 大于 REP_DIST 时，参数不起作用，负向软限位被禁止。 取消软限位时，建议不要去修改 REP_DIST 的值，将 RS_LIMIT 设置一个较大值即可。RS_LIMIT 的值默认为-200000000。 软限位无法作为 DATUM 回零时的限位信号参考。 注：设置限位前需检查有无设置对应的加减速度，以及快减减速度，若无设置，碰到限位无法停止

指令 218	**ZAux_Direct_SetFwdIn**
指令原型	int32 __stdcall ZAux_Direct_SetFwdIn(ZMC_HANDLE handle, int iaxis,int iValue)
指令说明	设置正向限位信号，-1 表示取消，参见软件手册里面的“FWD_IN”指令。 注：设置限位前需检查有无设置对应的加减速度，以及快减减速度，若无设置，碰到限位无法停止
输入参数	参数名 | 描述 handle | 连接句柄。 iaxis | 轴号。 iValue | IO 口编号，-1 时为取消设置。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	正负硬限位设置
详细说明	1 设置限位前需检查有无设置对应的加减速度，以及快减减速度，若无设置，碰到限位无法停止 2 输入 OFF 时，认为有信号输入，要相反效果可以用 INVERT_IN 反转电平(ECI 系列控制器相反)。 3 正负限位切记勿要设置反，否则碰到限位还是会继续运动。

指令 219	ZAux_Direct_GetFwdIn
指令原型	int32 __stdcall ZAux_Direct_GetFwdIn (ZMC_HANDLE handle, int iaxis, int * piValue)
指令说明	读取选择轴的正限位 IO 号。注：设置限位前需检查有无设置对应的加减速度，以及快减减速度，若无设置，碰到限位无法停止
输入参数	参数名 | 描述 handle | 连接句柄。 iaxis | 轴号。
输入参数	参数名 | 描述 piValue | 返回对应的输入口编号，-1 表示没有设置。
piValue	返回对应的输入口编号，-1 表示没有设置。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	正负硬限位设置
详细说明	1 设置限位前需检查有无设置对应的加减速度，以及快减减速度，若无设置，碰到限位无法停止 2 输入 OFF 时，认为有信号输入，要相反效果可以用 INVERT_IN 反转电平(ECI 系列控制器相反)。 3 正负限位切记勿要设置反，否则碰到限位还是会继续运动。

指令 220	ZAux_Direct_SetRevIn
指令原型	int32 __stdcall ZAux_Direct_SetRevIn(ZMC_HANDLE handle, int iaxis,int iValue);
指令说明	设置负向限位信号，-1 表示取消，参见软件手册里面的“REV_IN”指令。注：设置限位前需检查有无设置对应的加减速度，以及快减减速度，若无设置，碰到限位无法停止
输入参数	参数名 | 描述 handle | 连接句柄。 iaxis | 轴号。 iValue | IO 口编号，-1 时为取消设置。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	正负硬限位设置
详细说明	1 设置限位前需检查有无设置对应的加减速度，以及快减减速度，若无设

	置，碰到限位无法停止 2 输入 OFF 时，认为有信号输入，要相反效果可以用 INVERT_IN 反转电平(ECI 系列控制器相反)。 3 正负限位切记勿要设置反，否则碰到限位还是会继续运动。

指令 221	ZAux_Direct_GetRevIn
指令原型	int32 __stdcall ZAux_Direct_GetRevIn(ZMC_HANDLE handle, int iaxis,int* piValue);
指令说明	读取选择轴的负限位 IO 号。注：设置限位前需检查有无设置对应的加减速度，以及快减减速度，若无设置，碰到限位无法停止
输入参数	参数名 | 描述 handle | 连接句柄。 iaxis | 轴号。
输入参数	参数名 | 描述 piValue | 返回对应的输入口编号，-1 表示没有设置。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	正负硬限位设置
详细说明	1 设置限位前需检查有无设置对应的加减速度，以及快减减速度，若无设置，碰到限位无法停止 2 输入 OFF 时，认为有信号输入，要相反效果可以用 INVERT_IN 反转电平(ECI 系列控制器相反)。 3 正负限位切记勿要设置反，否则碰到限位还是会继续运动。

指令 222	ZAux_Direct_SetAlmIn
指令原型	int32 __stdcall ZAux_Direct_SetAlmIn(ZMC_HANDLE handle, int iaxis,int iValue)
指令说明	设置轴报警信号，-1 表示取消，参见软件手册里面的“ALM_IN”指令。
输入参数	参数名 | 描述 handle | 连接句柄。 iaxis | 轴号。 iValue | IO 口编号，-1 时为取消设置。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	告警 IN 设置

详细说明	1 设置轴报警信号前需检查有无设置对应的加减速度，以及快减减速度，若无设置，碰到限位无法停止 2 输入 OFF 时，认为有信号输入，要相反效果可以用 INVERT_IN 反转电平(ECI 系列控制器相反)。

指令 223	ZAux_Direct_GetAlmIn
指令原型	int32 __stdcall ZAux_Direct_GetAlmIn(ZMC_HANDLE handle, int iaxis,int * piValue)
指令说明	读取选择轴的报警 IO 号。
输入参数	参数名 | 描述 handle | 连接句柄。 iaxis | 轴号。
输入参数	参数名 | 描述 piValue | 返回对应的输入口编号，-1 表示没有设置。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	告警 IN 设置
详细说明	1 设置轴报警信号前需检查有无设置对应的加减速度，以及快减减速度，若无设置，碰到限位无法停止 2 输入 OFF 时，认为有信号输入，要相反效果可以用 INVERT_IN 反转电平(ECI 系列控制器相反)。

指令 224	ZAux_Direct_SetVrf
指令原型	int32 __stdcall ZAux_Direct_SetVrf(ZMC_HANDLE handle,int vrstartnum, int numes, float *pfValue);
指令说明	设置 VR 中的数据。
输入参数	参数名 | 描述 handle | 连接句柄。 vrstartnum | 操作 VR 开始的起始编号 numes | 写的个数 pfValue | 数据列表
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	VR 寄存器的使用
详细说明	32 位浮点型。掉电存储寄存器

	注意不同型号控制器的数量有区别。 保存浮点数，与 VR_INT 和 VRSTRING 共用一个空间。

指令 225	ZAux_Direct_GetVrf
指令原型	int32 __stdcall ZAux_Direct_GetVrf(ZMC_HANDLE handle, int vrstartnum, int numes, float *pfValue)
指令说明	读取 VR 中的数据。
输入参数	参数名 | 描述 handle | 连接句柄。 vrstartnum | 操作 VR 开始的起始编号 numes | 读的个数
输出参数	参数名 | 描述 pfValue | 数据列表
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	VR 寄存器的使用
详细说明	32 位浮点型。 注意不同型号控制器的数量有区别。 保存浮点数，与 VR_INT 和 VRSTRING 共用一个空间。

指令 226	ZAux_Direct_GetVrInt
指令原型	int32 __stdcall ZAux_Direct_GetVrInt (ZMC_HANDLE handle ,int vrstartnum, int numes, int * piValue);
指令说明	VR 寄存器整型读取，必须 150401 以上版本才支持 VR_INT 的 DirectCommand 读取。
输入参数	参数名 | 描述 handle | 连接句柄。 vrstartnum | 操作 VR 开始的起始编号 numes | 读的个数
输出参数	参数名 | 描述 pfValue | 数据列表
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	/
详细说明	/

指令 227	ZAux_Direct_SetTable
指令原型	int32 __stdcall ZAux_Direct_SetTable(ZMC_HANDLE handle,int tabstart, int numes, float *pfValue)
指令说明	把数据写入到 TABLE 中。
输入参数	参数名 | 描述 handle | 连接句柄。 tabstart | 操作 TABLE 开始的起始编号 numes | 写的个数 pfValue | 数据列表
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	Table 寄存器的使用
详细说明	是控制器自带的一个超大数组，数据类型为 32 位浮点型（4 系列及以上为 64 位浮点数），掉电不保存。

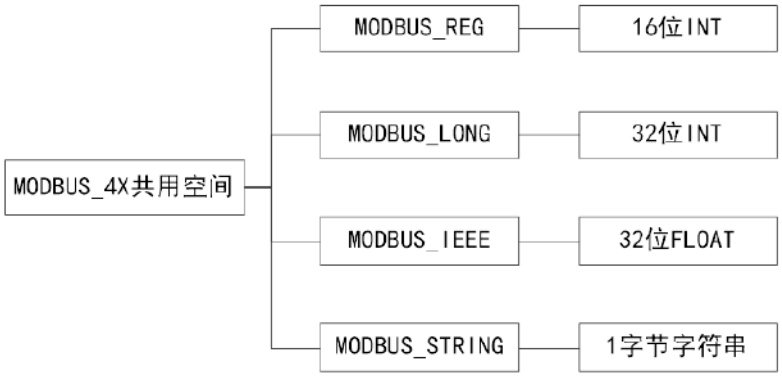
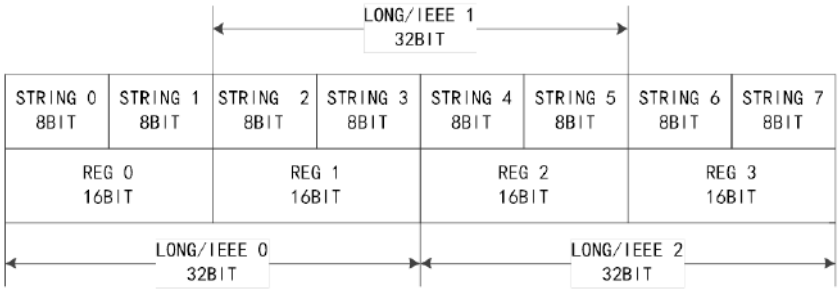
指令 228	ZAux_Direct_GetTable
指令原型	int32 __stdcall ZAux_Direct_GetTable(ZMC_HANDLE handle, int tabstart, int numes, float *pfValue)
指令说明	读取 TABLE 中的数据。
输入参数	参数名 | 描述 handle | 连接句柄。 tabstart | 读取 Table 寄存器的起始编号 numes | 读的个数
输出参数	参数名 | 描述 pfValue | 数据列表
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	Table 寄存器的使用
详细说明	是控制器自带的一个超大数组，数据类型为 32 位浮点型（4 系列及以上为 64 位浮点数），掉电不保存。

指令 229	ZAux_Modbus_Set0x
指令原型	int32 __stdcall ZAux_Modbus_Set0x(ZMC_HANDLE handle, uint16 start, uint16 inum, uint8* pdata)

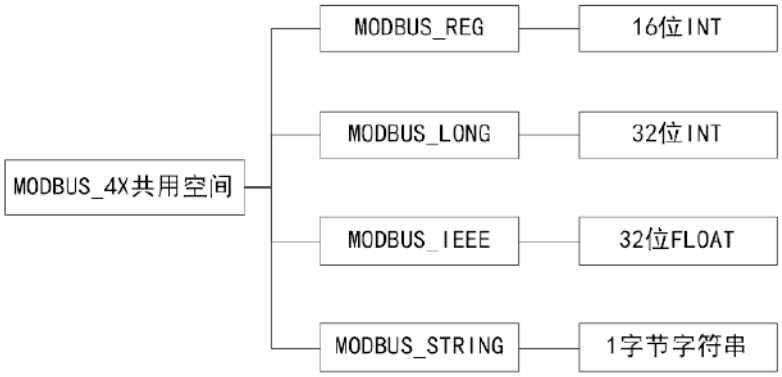
指令说明	设置 modbus 位寄存器。
输入参数	参数名 | 描述 handle | 连接句柄。 start | 操作 modbus_bit 起始编号 inum | 写的个数 pdata | 数据列表
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	Modbus 寄存器的使用
详细说明	/

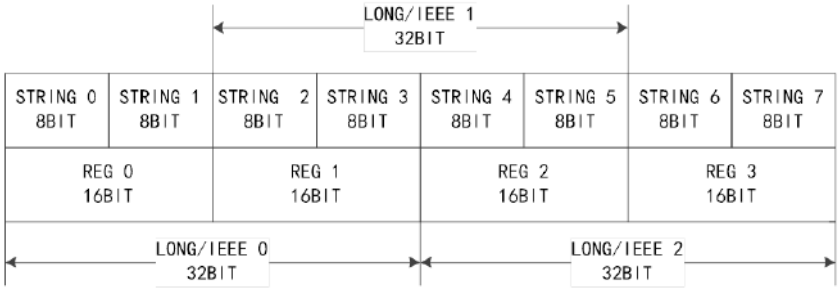
指令 230	**ZAux_Modbus_Get0x**
指令原型	int32 __stdcall ZAux_Modbus_Get0x(ZMC_HANDLE handle, uint16 start, uint16 inum, uint8* pdata)
指令说明	读取 modbus 位寄存器。
输入参数	参数名 | 描述 handle | 连接句柄。 start | 操作 modbus_bit 起始编号 inum | 读的个数
输出参数	参数名 | 描述 pdata | 数据列表
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	Modbus 寄存器的使用
详细说明	/

指令 231	**ZAux_Modbus_Set4x**
指令原型	int32 __stdcall ZAux_Modbus_Set4x(ZMC_HANDLE handle, uint16 start, uint16 inum, uint16* pdata)
指令说明	设置 modbus 字寄存器

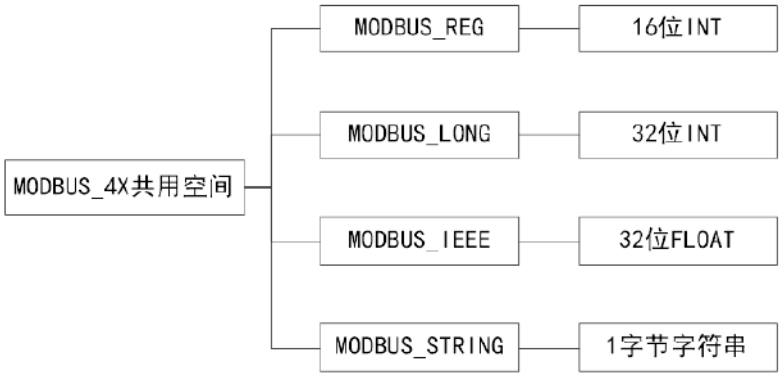
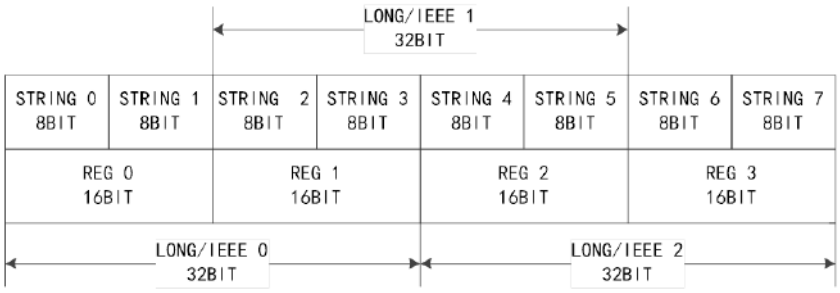
输入参数	<table><tr><td>参数名</td><td>描述</td></tr><tr><td>handle</td><td>连接句柄。</td></tr><tr><td>start</td><td>操作 modbus_reg 起始编号</td></tr><tr><td>inum</td><td>写的个数</td></tr><tr><td>pdata</td><td>数据列表</td></tr></table>
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	Modbus 寄存器的使用
详细说明	MODBUS 字寄存器，该函数对应 modbus_reg，与其他 modbus 寄存器公用空间：  4X 空间示意图： 

指令 232	**ZAux_Modbus_Get4x**
指令原型	int32 __stdcall ZAux_Modbus_Get4x(ZMC_HANDLE handle, uint16 start, uint16 inum, uint16* pdata)
指令说明	读取 modbus 字寄存器

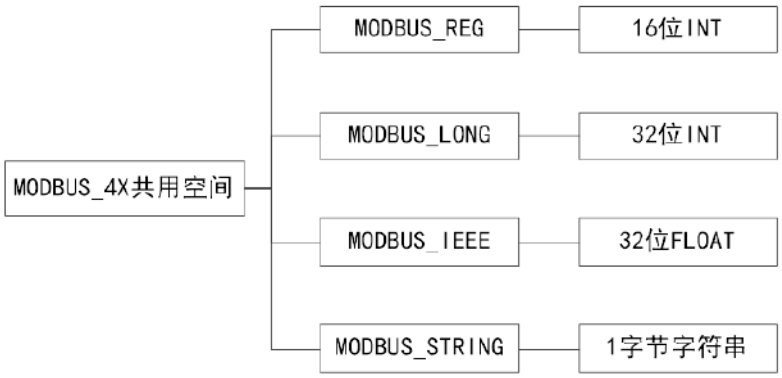
<table>
<tr><td>输入参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>handle</td><td>连接句柄。</td></tr>
<tr><td>start</td><td>操作 modbus_reg 起始编号</td></tr>
<tr><td>inum</td><td>读的个数</td></tr>
</table>
</td></tr>
<tr><td>输入参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>pdata</td><td>数据列表</td></tr>
</table>
</td></tr>
<tr><td>返回值</td><td>成功返回值为 0，非 0 详见错误码说明。</td></tr>
<tr><td>指令示例</td><td>Modbus 寄存器的使用</td></tr>
<tr><td>详细说明</td><td>MODBUS 字寄存器，该函数对应 modbus_reg，与其他 modbus 寄存器公用空间：
 

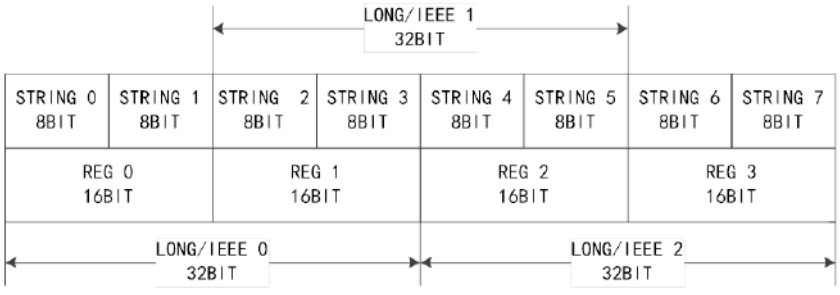
4X 空间示意图：
 
</td></tr>
</table>

指令 233	**ZAux_Modbus_Set4x_Long**
指令原型	int32 __stdcall ZAux_Modbus_Set4x_Long (ZMC_HANDLE handle, uint16 start, uint16 inum, int32 * pidata);
指令说明	Modbus 寄存器操作 MODBUS_LONG 设置。

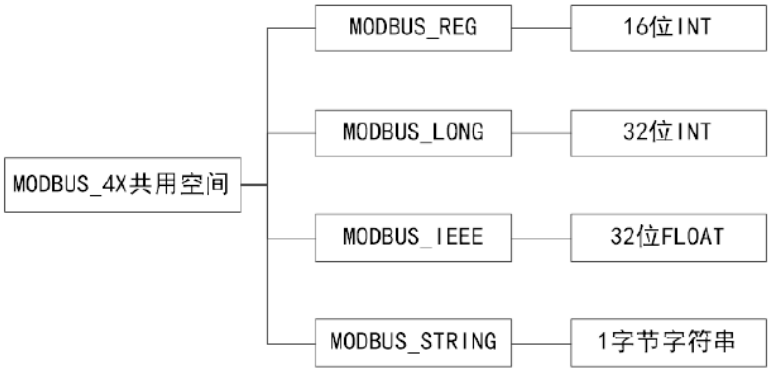
输入参数	参数名 | 描述 handle | 连接句柄。 start | 操作 modbus_long 起始编号 inum | 写的个数 pdata | 数据列表
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	/
详细说明	一个 MODBUS_LONG 占用两个 MODBUS_REG 地址，如：MODBUS_LONG(0)占用 MODBUS_REG(0)与 MODBUS_REG(1)两个 REG 地址。 MODBUS 字寄存器，该函数对应 modbus_long，与其他 modbus 寄存器公用空间：  4X 空间示意图： 

指令 234	**ZAux_Modbus_Get4x_Long**
指令原型	int32 __stdcall ZAux_Modbus_Get4x_Long (ZMC_HANDLE handle, uint16 start, uint16 inum, int32 * pidata);
指令说明	Modbus 寄存器操作 MODBUS_LONG 读取。

<table>
<tr><td>输入参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>handle</td><td>连接句柄。</td></tr>
<tr><td>start</td><td>操作 modbus_long 起始编号</td></tr>
<tr><td>inum</td><td>读的个数</td></tr>
</table>
</td></tr>
<tr><td>输入参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>pdata</td><td>数据列表</td></tr>
</table>
</td></tr>
<tr><td>返回值</td><td>成功返回值为 0，非 0 详见错误码说明。</td></tr>
<tr><td>指令示例</td><td>/</td></tr>
<tr><td>详细说明</td><td>一个 MODBUS_LONG 占用两个 MODBUS_REG 地址，如：MODBUS_LONG(0)占用 MODBUS_REG(0)与 MODBUS_REG(1)两个 REG 地址。
MODBUS 字寄存器，该函数对应 modbus_long，与其他 modbus 寄存器公用空间：
 

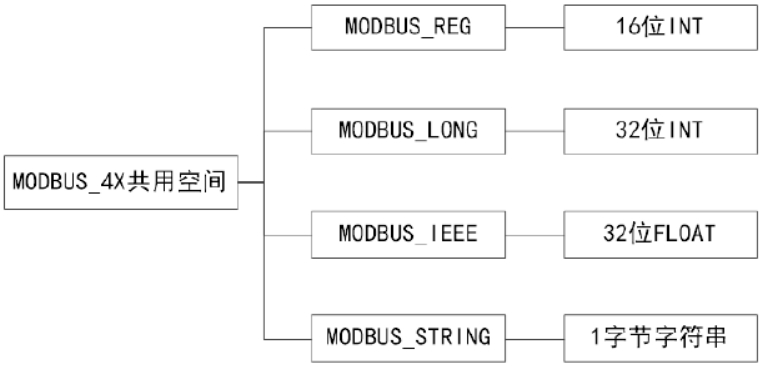
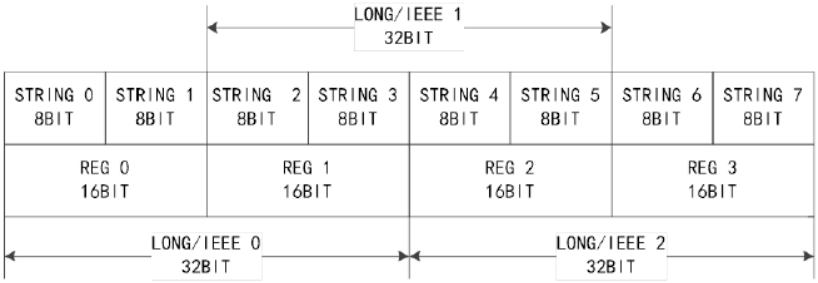
4X 空间示意图：
 
</td></tr>
</table>

指令 235	**ZAux_Modbus_Set4x_String**
指令原型	int32 __stdcall ZAux_Modbus_Set4x_String (ZMC_HANDLE handle , uint16 start,uint16 inum, char * pidata);
指令说明	设置 modbus_string

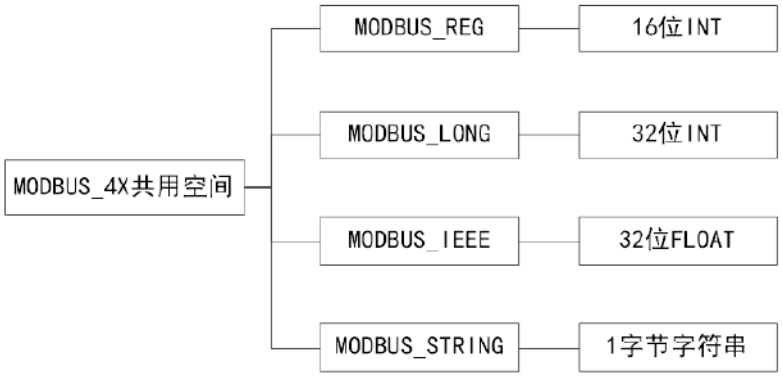
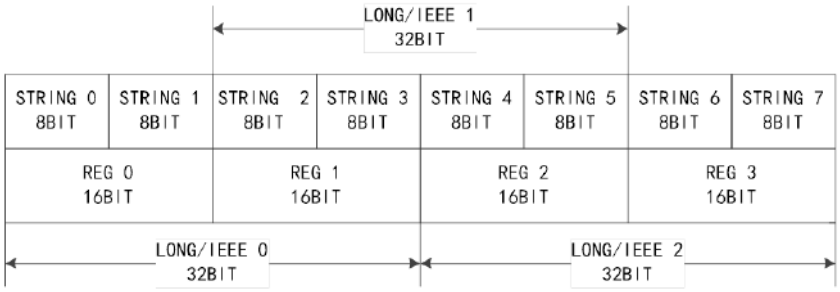
<table>
<tr><td>输入参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>handle</td><td>连接句柄。</td></tr>
<tr><td>start</td><td>Modbus 起始地址。</td></tr>
<tr><td>inum</td><td>长度。</td></tr>
<tr><td>pdata</td><td>写入的字符串。</td></tr>
</table>
</td></tr>
<tr><td>输出参数</td><td>/</td></tr>
<tr><td>返回值</td><td>成功返回值为 0，非 0 详见错误码说明。</td></tr>
<tr><td>指令示例</td><td>Modbus 寄存器的使用</td></tr>
<tr><td>详细说明</td><td>MODBUS 字寄存器，该函数对应 modbus_string，与其他 modbus 寄存器公用空间：
 

4X 空间示意图：
 
</td></tr>
</table>

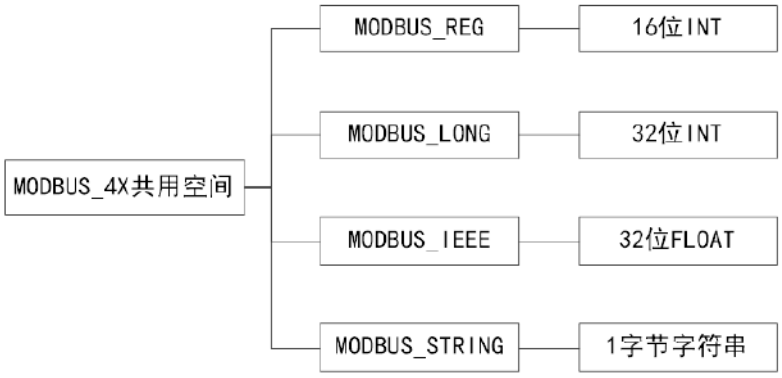
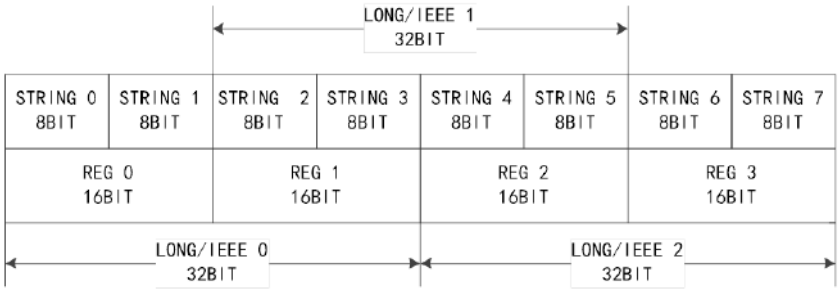
<table>
<tr><td>指令 236</td><td>ZAux_Modbus_Get4x_String</td></tr>
<tr><td>指令原型</td><td>int32 __stdcall ZAux_Modbus_Get4x_String (ZMC_HANDLE handle , uint16 start,uint16 inum, char * pidata);</td></tr>
<tr><td>指令说明</td><td>读取 modbus_string</td></tr>
<tr><td>输入参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>handle</td><td>连接句柄。</td></tr>
<tr><td>start</td><td>Modbus 起始地址。</td></tr>
<tr><td>inum</td><td>长度</td></tr>
</table>
</td></tr>
</table>

输出参数	<table><tr><td>参数名</td><td>描述</td></tr><tr><td>pdata</td><td>读取的字符串。</td></tr></table>
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	Modbus 寄存器的使用
详细说明	MODBUS 字寄存器，该函数对应 modbus_string，与其他 modbus 寄存器公用空间：  4X 空间示意图： 

指令 237	**ZAux_Modbus_Set4x_Float**
指令原型	int32 __stdcall ZAux_Modbus_Set4x_Float (ZMC_HANDLE handle, uint16 start, uint16 inum, float * pfdata);
指令说明	Modbus 寄存器操作 MODBUS_IEEE 设置。
输入参数	<table><tr><td>参数名</td><td>描述</td></tr><tr><td>handle</td><td>连接句柄。</td></tr><tr><td>start</td><td>操作 modbus_ieee 起始编号</td></tr><tr><td>inum</td><td>写的个数</td></tr><tr><td>pdata</td><td>数据列表</td></tr></table>
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。

指令示例	/
详细说明	一个 MODBUS_IEEE 占用两个 MODBUS_REG 地址，如：MODBUS_IEEE(0)占用 MODBUS_REG(0)与 MODBUS_REG(1)两个 REG 地址。 MODBUS 字寄存器，该函数对应 modbus_ieee，与其他 modbus 寄存器公用空间：  4X 空间示意图： 

指令 238	**ZAux_Modbus_Get4x_Float**
指令原型	int32 __stdcall ZAux_Modbus_Get4x_Float(ZMC_HANDLE handle, uint16 start, uint16 inum, float * pfdata);
指令说明	Modbus 寄存器操作 MODBUS_IEEE 读取.
输入参数	参数名 | 描述 handle | 连接句柄。 start | 操作 modbus_ieee 起始编号 inum | 读的个数
输出参数	参数名 | 描述 pdata | 数据列表
返回值	成功返回值为 0，非 0 详见**错误码**说明。

指令示例	/
详细说明	一个 MODBUS_IEEE 占用两个 MODBUS_REG 地址，如：MODBUS_IEEE(0)占用 MODBUS_REG(0)与 MODBUS_REG(1)两个 REG 地址。 MODBUS 字寄存器，该函数对应 modbus_ieee，与其他 modbus 寄存器公用空间：  4X 空间示意图： 

指令 239	**ZAux_WriteUFile**
指令原型	int32 __stdcall ZAux_WriteUFile(const char *sFilename, float *pVarlist, int inum)
指令说明	把 float 格式的变量列表存储到文件，与控制器的 U 盘文件格式一致。
输入参数	参数名：描述 sFilename：文件名。 pVarlist：数据列表。 inum：数据长度。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	文件读写测试

详细说明	/

指令 240	ZAux_ReadUFile
指令原型	int32 __stdcall ZAux_ReadUFile(const char *sFilename, float *pVarlist, int*pinum)
指令说明	读取 float 格式的变量列表，与控制器的 U 盘文件格式一致。
输入参数	参数名：描述 sFilename：文件名。 inum：数据长度。
输出参数	参数名：描述 pVarlist：数据列表。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	文件读写测试
详细说明	/

指令 241	ZAux_FlashWritef
指令原型	int32 __stdcall ZAux_FlashWritef(ZMC_HANDLE handle, uint16 uiflashid, uint32 uinumes, float *pfvlue)
指令说明	把 float 数据写入到 Flash 快中去。
输入参数	参数名：描述 handle：连接句柄。 uiflashid：FLASH 块号 uinumes：写的个数 pfvlue：数据列表
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	flash 读写测试
详细说明	内部 FLASH 采用顺序存储的方式，读取的顺序必须与存储时的顺序一致。 内部 FLASH 有存储次数限制，不要随意循环操作。 注意在运动中不要操作 FLASH，对运动执行会有影响

指令 242	ZAux_FlashReadf
指令原型	int32 __stdcall ZAux_FlashReadf(ZMC_HANDLE handle, uint16 uiflashid, uint32 uibuffnum, float *pfvlue, uint32* puinumesread)

指令说明	读取用户数据(float 类型)从 Flash 块中。
输入参数	参数名 | 描述 handle | 连接句柄。 uiflashid | FLASH 块号 uibuffnum | 缓冲变量个数
输出参数	参数名 | 描述 pfvlue | 数据列表。 puinumesread | 读取到的变量个数
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	flash 读写测试
详细说明	内部 FLASH 采用顺序存储的方式，读取的顺序必须与存储时的顺序一致。 内部 FLASH 有存储次数限制，不要随意循环操作。 注意在运动中不要操作 FLASH，对运动执行会有影响

指令 243	**ZAux_BusCmd_GetNodeNum**
指令原型	int32 __stdcall ZAux_BusCmd_GetNodeNum(ZMC_HANDLE handle,int slot,int *piValue);
指令说明	获取节点数量。
输入参数	参数名 | 描述 handle | 连接句柄。 slot | 槽位号
输出参数	参数名 | 描述 pivalue | 返回的节点数量。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	总线初始化（zomtiontools 方式）
详细说明	/

指令 244	**ZAux_BusCmd_GetNodeInfo**
指令原型	int32 __stdcall ZAux_BusCmd_GetNodeInfo(ZMC_HANDLE handle, int slot, int node, int sel, int *piValue);
指令说明	读取节点信息

输入参数	参数名 | 描述 handle | 连接句柄。 slot | 槽位号 node | 节点编号 sel | 信息编号
输出参数	参数名 | 描述 pivalue | 返回的参数信息。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	总线初始化（zomtiontools 方式）
详细说明	Sel 值 | 描述 0 | EtherCat 节点厂商编号 1 | EtherCat 节点设备编号 2 | EtherCat 节点版本 3 | EtherCat 节点 ALIAS 别名 4 | 预留 10 | EtherCat 节点 IN 个数 11 | EtherCat 节点 OUT 口个数 12 | EtherCat 节点 AIN 个数 13 | EtherCat 节点 Aout 个数 14 | 预留

指令 245	**ZAux_BusCmd_GetNodeStatus**
指令原型	int32 __stdcall ZAux_Direct_GetNodeStatus (ZMC_HANDLE handle, uint32 slot , uint32 node , uint32 *nodestatus);
指令说明	读取节点总线状态。
输入参数	参数名 | 描述 Handle | 连接句柄。 slot | 槽位号 node | 节点编号
输出参数	参数名 | 描述 nodestatus | nodestaus 按位处理 bit0-节点是否存在 bit1-通讯状态 bit2-节点状态。

返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	/
详细说明	Nodestaus： 值为 1 时，bit0 为 1，bit1 和 bit2 为 0，设备通讯正常 值为 3 时，bit0 和 bit1 为 1，bit2 为 0，设备通讯出错

BIT 位	意义
0	指明 node 是否存在；1-存在
1	通讯状态；1-出错；
2	节点状态；1-出错；

指令 246	**ZAux_BusCmd_DriveClear**
指令原型	int32 __stdcall ZAux_BusCmd_DriveClear(ZMC_HANDLE handle, int iaxis, uint32 mode);
指令说明	设置清除总线伺服报警。
输入参数	参数名：描述 handle：连接句柄。 iaxis：轴号 mode：0-清除当前 1-清除历史 2-清除外部输入警告
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	/
详细说明	驱动器没有错误时，使用指令控制器会警告 6015，不影响程序运行。

指令 247	**ZAux_BusCmd_InitBus**
指令原型	int32 __stdcall ZAux_BusCmd_InitBus(ZMC_HANDLE handle);
指令说明	总线初始化
输入参数	参数名：描述 handle：连接句柄。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	总线初始化（zomtiontools 方式）
详细说明	针对 Zmotion tools 工具软件配置过总线参数控制器使用有效，也可以通过 ZAux_BasDown 函数，把工具内的 basic 文件下载到控制器后使用

指令 248	ZAux_BusCmd_GetInitStatus
指令原型	int32 __stdcall ZAux_BusCmd_GetInitStatus(ZMC_HANDLE handle,int *piValue);
指令说明	获取总线初始化状态。
输入参数	参数名 | 描述 handle | 连接句柄。
输出参数	参数名 | 描述 pivalue | 返回的状态 0-初始化失败 1-初始化成功。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	总线初始化（zomtiontools 方式）
详细说明	针对 Zmotion tools 工具软件配置过总线参数控制器使用有效，也可以通过 ZAux_BasDown 函数，把工具内的 basic 文件下载到控制器后使用

指令 249	ZAux_Direct_SetErrormask
指令原型	int32 __stdcall ZAux_Direct_SetErrormask (ZMC_HANDLE handle, int iaxis , int iValue);
指令说明	设置错误标记
输入参数	参数名 | 描述 Handle | 连接句柄。 iaxis | 轴号 iValue | 设置值。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	/
详细说明	和 AXISSTATUS 做与运算来决定那些错误需要关闭 WDOG。

指令 250	ZAux_Direct_GetErrormask
指令原型	int32 __stdcall ZAux_Direct_GetErrormask (ZMC_HANDLE handle, int iaxis , int * piValue);
指令说明	获取错误标记，和 AXISSTATUS 做与运算来决定那些错误需要关闭 WDOG。

<table>
<tr><td>输入参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>Handle</td><td>连接句柄。</td></tr>
<tr><td>iaxis</td><td>轴号</td></tr>
<tr><td>iValue</td><td>设置值。</td></tr>
</table>
</td></tr>
<tr><td>输出参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>piValue</td><td>返回的标记值。</td></tr>
</table>
</td></tr>
<tr><td>返回值</td><td>成功返回值为 0，非 0 详见错误码说明。</td></tr>
<tr><td>指令示例</td><td>/</td></tr>
<tr><td>详细说明</td><td>和 AXISSTATUS 做与运算来决定那些错误需要关闭 WDOG。</td></tr>
</table>

<table>
<tr><td>指令 251</td><td>ZAux_BusCmd_SDOWrite</td></tr>
<tr><td>指令原型</td><td>int32 __stdcall ZAux_BusCmd_SDOWrite(ZMC_HANDLE handle, uint32 slot, uint32 node, uint32 index, uint32 subindex, uint32 type, int value);</td></tr>
<tr><td>指令说明</td><td>通过设备号和槽位号进行 SDO 写入</td></tr>
<tr><td>输入参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>handle</td><td>连接句柄。</td></tr>
<tr><td>slot</td><td>槽位号</td></tr>
<tr><td>node</td><td>节点编号</td></tr>
<tr><td>index</td><td>对象字典编号</td></tr>
<tr><td>subindex</td><td>对象字典子编号</td></tr>
<tr><td>type</td><td>数据类型：
<table>
<tr><td>1</td><td>boolean</td></tr>
<tr><td>2</td><td>integer 8</td></tr>
<tr><td>3</td><td>integer 16</td></tr>
<tr><td>4</td><td>integer 32</td></tr>
<tr><td>5</td><td>unsigned 8</td></tr>
<tr><td>6</td><td>unsigned 16</td></tr>
<tr><td>7</td><td>unsigned 32</td></tr>
</table>
</td></tr>
<tr><td>value</td><td>写入的数据值</td></tr>
</table>
</td></tr>
<tr><td>输出参数</td><td>/</td></tr>
<tr><td>返回值</td><td>成功返回值为 0，非 0 详见错误码说明。</td></tr>
<tr><td>指令示例</td><td>SDO 读写最大扭矩</td></tr>
<tr><td>详细说明</td><td>需连接好设备，扫描总线后才能执行。</td></tr>
</table>

	只有可写的数据字典才能写入。

指令 252	ZAux_BusCmd_SDORead
指令原型	int32 __stdcall ZAux_BusCmd_SDORead(ZMC_HANDLE handle, uint32 slot, uint32 node, uint32 index, uint32 subindex, uint32 type, int *value);
指令说明	通过设备号和槽位号进行 SDO 读取。
输入参数	参数名：描述 handle：连接句柄。 slot：槽位号 node：节点编号 index：对象字典编号 subindex：对象字典子编号 type：数据类型： 1 boolean 2 integer 8 3 integer 16 4 integer 32 5 unsigned 8 6 unsigned 16 7 unsigned 32
输出参数	参数名：描述 value：读取的数据值。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	SDO 读写最大扭矩
详细说明	需连接好设备，扫描总线后才能执行。

指令 253	ZAux_BusCmd_RtexWrite
指令原型	int32 __stdcall ZAux_BusCmd_RtexWrite(ZMC_HANDLE handle, uint32 iaxis, uint32 ipara, float value);
指令说明	RTEX 写参数信息。
输入参数	参数名：描述 handle：连接句柄。 iaxis：轴号。

<table>
<tr><td rowspan="3">输入参数</td><td>ipara</td><td>伺服参数编号。<table><tr><td>值</td><td>描述</td></tr><tr><td>1</td><td>参数分类*256 + 参数编号（Pr7.20=7*256+20）</td></tr><tr><td>2</td><td>特殊参数=128，当前参数写入 EEPROM（此时 value=1）</td></tr><tr><td>3</td><td>特殊参数=40000h + 回零箝位模式+ (设置值*256)，使用驱动器的回零箝位功能</td></tr></table></td></tr>
<tr><td>value</td><td>参数值</td></tr>
<tr><td colspan="2"></td></tr>
<tr><td>输出参数</td><td colspan="2">/</td></tr>
<tr><td>返回值</td><td colspan="2">成功返回值为 0，非 0 详见错误码说明。</td></tr>
<tr><td>指令示例</td><td colspan="2">RTEX 参数相关信息</td></tr>
<tr><td>详细说明</td><td colspan="2">1.伺服参数
<table>
<tr><td>伺服参数</td><td>功能</td><td>值</td></tr>
<tr><td>Pr0.00</td><td>设置电机转向</td><td>0：CW 1：CCW</td></tr>
<tr><td>Pr0.01</td><td>控制模式设置</td><td>一般设置为 0：半闭环控制</td></tr>
<tr><td>Pr0.08</td><td>电机一圈脉冲数</td><td>0-8388608（根据实际电机）</td></tr>
<tr><td>Pr0.09</td><td>齿轮比分子设定</td><td>0-1073741824</td></tr>
<tr><td>Pr0.10</td><td>齿轮比分母设定</td><td>1-1073741824</td></tr>
<tr><td>Pr4.01</td><td>正限位信号设置</td><td>常闭：00818181h（8487297）
常开：00010101h（65793）</td></tr>
<tr><td>Pr4.02</td><td>负限位信号设置</td><td>常闭：00828282h（8553090）
常开：00020202h（131586）</td></tr>
<tr><td>Pr4.03</td><td>Home 信号设置</td><td>常闭：00A2A2A2h（10658466）
常开：00222222h（2236962）</td></tr>
<tr><td>转矩相关</td><td colspan="2">/
/</td></tr>
<tr><td>Pr0.13</td><td>第一转矩限制</td><td>0~500%</td></tr>
<tr><td>Pr5.21</td><td>转矩限制选择
转矩控制时，固定为 Pr0.13（第 1 转矩限制）。</td><td>如下图所示</td></tr>
</table></td></tr>
</table>

<table>
<tr><td></td><td>
<table>
<tr><td rowspan="2">设定值</td><td colspan="2">TL_SW=0</td><td colspan="2">TL_SW=1</td></tr>
<tr><td>负方向</td><td>正方向</td><td>负方向</td><td>正方向</td></tr>
<tr><td>0，[1]</td><td colspan="4">Pr0.13</td></tr>
<tr><td>2</td><td>Pr5.22</td><td>Pr0.13</td><td>Pr5.22</td><td>Pr0.13</td></tr>
<tr><td>3</td><td colspan="2">Pr0.13</td><td colspan="2">Pr5.22</td></tr>
<tr><td>4</td><td>Pr5.22</td><td>Pr0.13</td><td>Pr5.22</td><td>Pr5.25</td></tr>
</table>
</td></tr>
<tr><td>Pr5.22</td><td>第二转矩限制</td><td>0~500%</td></tr>
<tr><td>Pr5.25</td><td>正方向转矩限制</td><td>0~500%</td></tr>
<tr><td>Pr5.26</td><td>负方向转矩限制</td><td>0~500%</td></tr>
<tr><td>速度相关</td><td>/</td><td>/</td></tr>
<tr><td>Pr3.12</td><td>加速时间设置</td><td>0~10000ms （达到 1000.r/min）</td></tr>
<tr><td>Pr3.13</td><td>减速时间设置</td><td>0~10000ms （达到 1000.r/min）</td></tr>
<tr><td>Pr3.14</td><td>S 加减速设置</td><td>0~1000ms</td></tr>
<tr><td>Pr3.17</td><td>速度限制选择
转矩控制时的速度限制值的方式选择</td><td>
<table>
<tr><td rowspan="2">设定值</td><td colspan="2">SL_SW</td></tr>
<tr><td>0</td><td>1</td></tr>
<tr><td>0</td><td colspan="2">Pr3.21</td></tr>
<tr><td>1</td><td>Pr3.21</td><td>Pr3.22</td></tr>
</table>
设置为 1 时，根据 RTEX 通信指令 SL_SW 的数值进行选择
</td></tr>
<tr><td>Pr3.21</td><td>速度限制值 1</td><td>0~20000r/min</td></tr>
<tr><td>Pr3.22</td><td>速度限制值 2</td><td>0~20000r/min</td></tr>
</table>

2.Rtex 通讯参数

Pr7.20	Rtex 通讯周期	-1：将 Pr7.91 的设定生效 3：0.5ms 6：1.0ms
Pr7.21	Rtex 指令更新周期比	1：1 倍 2：2 倍
Pr7.91	Rtex 通讯周期扩展	62500 ns 125000 ns 250000 ns 500000 ns 1000000 ns 2000000 ns

3.回零箝位模式

typecode	描述
$50	位置箝位状态监视器
$51	位置箝位 1 启动
$52	位置箝位 2 启动
$53	位置箝位 1,2 启动
$54	位置箝位 1 解除
$58	位置箝位 2 解除
$5c	位置箝位 1,2 解除

4.设置值

设置值=钳位 1 设置 +($10* 钳位 2 设置)

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
LATCH_SEL2				LATCH_SEL1			

箝位 1/2 设置	描述
$0	Z 相信号触发
$1	EXT1 的逻辑上升沿
$2	EXT2 的逻辑上升沿
$3	EXT3 的逻辑上升沿
$9	EXT1 的逻辑下降沿
$10	EXT2 的逻辑下降沿
$11	EXT3 的逻辑下降沿

指令 254	ZAux_BusCmd_RtexRead
指令原型	int32 __stdcall ZAux_BusCmd_RtexRead(ZMC_HANDLE handle, uint32 iaxis, uint32 ipara, float *value);
指令说明	RTEX 读参数信息。

输入参数

参数名	描述
handle	连接句柄。
iaxis	轴号。
ipara	伺服参数编号。 值 1：参数分类*256 + 参数编号（Pr7.20=7*256+20） 值 2：特殊参数=128，当前参数写入 EEPROM（此时 value=1） 值 3：特殊参数=40000h + **回零箝位模式**+ (设置值*256)，使用驱动器的回零箝位功能
value	参数值

输入参数

参数名	描述
value	读到的数据。

返回值

成功返回值为 0，非 0 详见**错误码**说明。

指令示例

RTEX 参数相关信息

详细说明

1.伺服参数

伺服参数	功能	值
Pr0.00	设置电机转向	0：CW　1：CCW
Pr0.01	控制模式设置	一般设置为 0：半闭环控制
Pr0.08	电机一圈脉冲数	0-8388608（根据实际电机）
Pr0.09	齿轮比分子设定	0-1073741824
Pr0.10	齿轮比分母设定	1-1073741824
Pr4.01	正限位信号设置	常闭：00818181h（8487297） 常开：00010101h（65793）
Pr4.02	负限位信号设置	常闭：00828282h（8553090） 常开：00020202h（131586）
Pr4.03	Home 信号设置	常闭：00A2A2A2h（10658466） 常开：00222222h（2236962）
转矩相关	/ /	

<table>
<tr><td>Pr0.13</td><td>第一转矩限制</td><td>0~500%</td></tr>
<tr><td rowspan="2">Pr5.21</td><td>转矩限制选择
转矩控制时，固定为 Pr0.13（第 1 转矩限制）。</td><td>如下图所示</td></tr>
<tr><td colspan="2"><table>
<tr><td rowspan="2">设定值</td><td colspan="2">TL_SW=0</td><td colspan="2">TL_SW=1</td></tr>
<tr><td>负方向</td><td>正方向</td><td>负方向</td><td>正方向</td></tr>
<tr><td>0，[1]</td><td colspan="4">Pr0.13</td></tr>
<tr><td>2</td><td>Pr5.22</td><td>Pr0.13</td><td>Pr5.22</td><td>Pr0.13</td></tr>
<tr><td>3</td><td colspan="2">Pr0.13</td><td colspan="2">Pr5.22</td></tr>
<tr><td>4</td><td>Pr5.22</td><td>Pr0.13</td><td>Pr5.22</td><td>Pr5.25</td></tr>
</table></td></tr>
<tr><td>Pr5.22</td><td>第二转矩限制</td><td>0~500%</td></tr>
<tr><td>Pr5.25</td><td>正方向转矩限制</td><td>0~500%</td></tr>
<tr><td>Pr5.26</td><td>负方向转矩限制</td><td>0~500%</td></tr>
<tr><td>速度相关</td><td>/</td><td>/</td></tr>
<tr><td>Pr3.12</td><td>加速时间设置</td><td>0~10000ms （达到 1000.r/min）</td></tr>
<tr><td>Pr3.13</td><td>减速时间设置</td><td>0~10000ms （达到 1000.r/min）</td></tr>
<tr><td>Pr3.14</td><td>S 加减速设置</td><td>0~1000ms</td></tr>
<tr><td>Pr3.17</td><td>速度限制选择
转矩控制时的速度限制值的方式选择</td><td><table>
<tr><td rowspan="2">设定值</td><td colspan="2">SL_SW</td></tr>
<tr><td>0</td><td>1</td></tr>
<tr><td>[0]</td><td colspan="2">Pr3.21</td></tr>
<tr><td>1</td><td>Pr3.21</td><td>Pr3.22</td></tr>
</table>设置为 1 时，根据 RTEX 通信指令 SL_SW 的数值进行选择</td></tr>
<tr><td>Pr3.21</td><td>速度限制值 1</td><td>0~20000r/min</td></tr>
<tr><td>Pr3.22</td><td>速度限制值 2</td><td>0~20000r/min</td></tr>
</table>

2.Rtex 通讯参数

Pr7.20	Rtex 通讯周期	-1：将 Pr7.91 的设定生效 3：0.5ms 6：1.0ms
Pr7.21	Rtex 指令更新周期比	1：1 倍 2：2 倍
Pr7.91	Rtex 通讯周期扩展	62500 ns 125000 ns

		250000 ns 500000 ns 1000000 ns 2000000 ns

3.回零箝位模式

typecode	描述
$50	位置箝位状态监视器
$51	位置箝位 1 启动
$52	位置箝位 2 启动
$53	位置箝位 1,2 启动
$54	位置箝位 1 解除
$58	位置箝位 2 解除
$5c	位置箝位 1,2 解除

4.设置值

设置值=钳位 1 设置 +($10* 钳位 2 设置)

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
LATCH_SEL2				LATCH_SEL1			

箝位 1/2 设置	描述
$0	Z 相信号触发
$1	EXT1 的逻辑上升沿
$2	EXT2 的逻辑上升沿
$3	EXT3 的逻辑上升沿
$9	EXT1 的逻辑下降沿
$10	EXT2 的逻辑下降沿
$11	EXT3 的逻辑下降沿

指令 255	ZAux_BusCmd_SDOReadAxis
指令原型	int32 __stdcall ZAux_BusCmd_SDOReadAxis (ZMC_HANDLE handle, uint32 iaxis,uint32 index,uint32 subindex ,uint32 type ,int32 * value)
指令说明	通过轴号进行 SDO 读取。

<table>
<tr><td>输入参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>handle</td><td>连接句柄。</td></tr>
<tr><td>iaxis</td><td>轴号。</td></tr>
<tr><td>index</td><td>对象字典编号。</td></tr>
<tr><td>subindex</td><td>对象字典子编号。</td></tr>
<tr><td>type</td><td>数据类型：
<table>
<tr><td>1</td><td>boolean</td></tr>
<tr><td>2</td><td>integer 8</td></tr>
<tr><td>3</td><td>integer 16</td></tr>
<tr><td>4</td><td>integer 32</td></tr>
<tr><td>5</td><td>unsigned 8</td></tr>
<tr><td>6</td><td>unsigned 16</td></tr>
<tr><td>7</td><td>unsigned 32</td></tr>
</table>
</td></tr>
<tr><td>type</td><td>数据类型。</td></tr>
</table>
</td></tr>
<tr><td>输出参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>value</td><td>读取的数据值。</td></tr>
</table>
</td></tr>
<tr><td>返回值</td><td>成功返回值为 0，非 0 详见错误码说明。</td></tr>
<tr><td>指令示例</td><td>Int32 iValue;
ZAux_BusCmd_SDOReadAxis(0, 24672,0,2,&iValue);// 24672=0x6061
//获取轴 0 的控制模式</td></tr>
<tr><td>详细说明</td><td>需连接好设备，扫描总线后才能执行。
只有可读的数据字典才能读取。</td></tr>
</table>

指令 256	**ZAux_BusCmd_SDOWriteAxis**
指令原型	int32 __stdcall ZAux_BusCmd_SDOWriteAxis (ZMC_HANDLE handle, uint32 iaxis, uint32 index,uint32 subindex ,uint32 type ,int32 value)
指令说明	通过轴号进行 SDO 写入。

<table>
<tr><td>输入参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>handle</td><td>连接句柄。</td></tr>
<tr><td>iaxis</td><td>轴号。</td></tr>
<tr><td>index</td><td>对象字典编号。</td></tr>
<tr><td>subindex</td><td>对象字典子编号。</td></tr>
<tr><td>type</td><td>数据类型：
<table>
<tr><td>1</td><td>boolean</td></tr>
<tr><td>2</td><td>integer 8</td></tr>
<tr><td>3</td><td>integer 16</td></tr>
<tr><td>4</td><td>integer 32</td></tr>
<tr><td>5</td><td>unsigned 8</td></tr>
<tr><td>6</td><td>unsigned 16</td></tr>
<tr><td>7</td><td>unsigned 32</td></tr>
</table>
</td></tr>
<tr><td>value</td><td>写入的数据值。</td></tr>
</table>
</td></tr>
<tr><td>输出参数</td><td>/</td></tr>
<tr><td>返回值</td><td>成功返回值为 0，非 0 详见错误码说明。</td></tr>
<tr><td>指令示例</td><td>ZAux_BusCmd_SDOWriteAxis(0,0x6060,0,2,8);
//轴 0 设备设置控制模式为 8，位置控制</td></tr>
<tr><td>详细说明</td><td>需连接好设备，扫描总线后才能执行。
只有可写的数据字典才能写入。</td></tr>
</table>

指令 257	**ZAux_BusCmd_NodePdoWrite**
指令原型	int32 __stdcall ZAux_BusCmd_NodePdoWrite (ZMC_HANDLE handle, uint32 inode,uint32 index,uint32 subindex ,uint32 type ,int value)
指令说明	PDO 写操作。

<table>
<tr><td>输入参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>handle</td><td>连接句柄。</td></tr>
<tr><td>inode</td><td>设备编号。</td></tr>
<tr><td>index</td><td>对象字典编号。</td></tr>
<tr><td>subindex</td><td>对象字典子编号。</td></tr>
<tr><td>type</td><td>数据类型：
<table>
<tr><td>1</td><td>boolean</td></tr>
<tr><td>2</td><td>integer 8</td></tr>
<tr><td>3</td><td>integer 16</td></tr>
<tr><td>4</td><td>integer 32</td></tr>
<tr><td>5</td><td>unsigned 8</td></tr>
<tr><td>6</td><td>unsigned 16</td></tr>
<tr><td>7</td><td>unsigned 32</td></tr>
</table>
</td></tr>
<tr><td>value</td><td>写入数据值。</td></tr>
</table>
</td></tr>
<tr><td>输出参数</td><td>/</td></tr>
<tr><td>返回值</td><td>成功返回值为 0，非 0 详见错误码说明。</td></tr>
<tr><td>指令示例</td><td>ZAux_BusCmd_NodePdoWrite(0,0x6040,0,3,15);
//设置设备 0 控制字为 15</td></tr>
<tr><td>详细说明</td><td>非轴和 IO 的设备，通过这个指令来读写 PDO，例如电源设备。
轴和 IO 类型的设备，已经可以通过轴参数和 IO 指令来访问 PDO，不能使用这个指令。
总线开启会自动提前读取当前的 PDO 列表，以及可写 PDO 的当前值。
可以在总线开启调用前通过 SDO 修改 PDO 列表，或相关数据字典的当前值。
需要启动以后才能修改，可以先用 SOD_START 启动到 SAFEOP，然后设置初始化 PDO 状态.，再启动到 OP。</td></tr>
</table>

<table>
<tr><td>指令 258</td><td>ZAux_BusCmd_NodePdoRead</td></tr>
<tr><td>指令原型</td><td>int32 __stdcall ZAux_BusCmd_NodePdoRead(ZMC_HANDLE handle,uint32 inode,uint32 index,uint32 subindex ,uint32 type ,int* ivalue)</td></tr>
<tr><td>指令说明</td><td>PDO 读操作。</td></tr>
<tr><td>输入参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>handle</td><td>连接句柄。</td></tr>
<tr><td>inode</td><td>设备编号。</td></tr>
<tr><td>index</td><td>对象字典编号。</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td></td><td>
<table>
<tr><td>subindex</td><td>对象字典子编号。</td></tr>
<tr><td>type</td><td>数据类型：
<table>
<tr><td>1</td><td>boolean</td></tr>
<tr><td>2</td><td>integer 8</td></tr>
<tr><td>3</td><td>integer 16</td></tr>
<tr><td>4</td><td>integer 32</td></tr>
<tr><td>5</td><td>unsigned 8</td></tr>
<tr><td>6</td><td>unsigned 16</td></tr>
<tr><td>7</td><td>unsigned 32</td></tr>
</table></td></tr>
<tr><td>value</td><td>写入数据值。</td></tr>
</table>
</td></tr>
<tr><td>输入参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>value</td><td>读取数据值。</td></tr>
</table>
</td></tr>
<tr><td>返回值</td><td>成功返回值为 0，非 0 详见错误码说明。</td></tr>
<tr><td>指令示例</td><td>Int32 iValue;
ZAux_BusCmd_NodePdoRead(0,0x6041,0,3,&iValue);
//获取设备 0 的状态字</td></tr>
<tr><td>详细说明</td><td>非轴和 IO 的设备，通过这个指令来读写 PDO，例如电源设备。
轴和 IO 类型的设备，已经可以通过轴参数和 IO 指令来访问 PDO，不能使用这个指令。</td></tr>
</table>

指令 259	**ZAux_BusCmd_GetDriveTorque**
指令原型	int32 __stdcall ZAux_BusCmd_GetDriveTorque (ZMC_HANDLE handle , int iaxis,int *piValue);
指令说明	读取当前总线力矩，需要设置对应的 DRIVE_PROFILE 类型
输入参数	参数名 / 描述 handle / 连接句柄。 iaxis / 轴号。
输出参数	参数名 / 描述 piValue / 当前转矩。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	获取当前总线驱动当前力矩
详细说明	必须设置正确的 ATYPE（设置为 65/66/67）与 DRIVE_PROFILE 以后才能读取。

指令 260	ZAux_BusCmd_SetMaxDriveTorque
指令原型	int32 __stdcall ZAux_BusCmd_SetMaxDriveTorque (ZMC_HANDLE handle, int iaxis , int piValue);
指令说明	设置当前总线驱动最大转矩 需要设置对应的 DRIVE_PROFILE 类型
输入参数	参数名 | 描述 handle | 连接句柄。 iaxis | 轴号。 fValue | 设置最大转矩限制。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	/
详细说明	必须设置正确的 ATYPE（设置为 65/66/67）与 DRIVE_PROFILE 以后才能读取。

指令 261	ZAux_BusCmd_GetMaxDriveTorque
指令原型	int32 __stdcall ZAux_BusCmd_GetMaxDriveTorque (ZMC_HANDLE handle, int iaxis , int *piValue);
指令说明	获取当前总线驱动最大转矩，需要设置对应的 DRIVE_PROFILE 类型
输入参数	参数名 | 描述 handle | 连接句柄。 iaxis | 轴号。
输出参数	参数名 | 描述 piValue | 获取的最大转矩
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	获取当前总线驱动当前力矩
详细说明	必须设置正确的 ATYPE（设置为 65/66/67）与 DRIVE_PROFILE 以后才能读取。

指令 262	ZAux_Direct_SetDAC
指令原型	int32 __stdcall ZAux_Direct_SetDAC (ZMC_HANDLE handle, int iaxis , float fValue);
指令说明	伺服轴 DA 直接控制力矩。ps:总线驱动需要设置对应 DRIVE_PROFILE 类型，ATYPE 需要设置为速度模式。

输入参数	参数名 | 描述 handle | 连接句柄。 iaxis | 轴号。 fValue | 设置模拟量输出值。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	/
详细说明	单位为 DA 模块的刻度，12 为或 16 位。 速度控制时要看驱动器具体单位。 力矩控制时单位为千分之一，等于 1000 时表示 100%力矩。

指令 263	**ZAux_Direct_GetDAC**
指令原型	int32 __stdcall ZAux_Direct_GetDAC (ZMC_HANDLE handle, int iaxis , float *fValue);
指令说明	伺服轴 DA 直接控制力矩。ps:总线驱动需要设置对应 DRIVE_PROFILE 类型，ATYPE 需要设置为速度模式。
输入参数	参数名 | 描述 handle | 连接句柄。 iaxis | 轴号。 fValue | 设置模拟量输出值。
输出参数	参数名 | 描述 fValue | 模拟量返回值。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	/
详细说明	单位为 DA 模块的刻度，12 为或 16 位。 速度控制时要看驱动器具体单位。 力矩控制时单位为千分之一，等于 1000 时表示 100%力矩。

指令 264	**ZAux_Execute**
指令原型	int32 __stdcall ZAux_Execute(ZMC_HANDLE handle, const char *pszCommand, char *psResponse, uint32 uiResponseLength)
指令说明	发送字符串命令到控制器，缓存方式(当控制器没有缓冲时自动阻塞)。 注意：ECI 等低系列控制器，在 bas 程序中添加打印，同时 PC 程序又调

	用了 ZAux_Execute 指令，会互相影响导致打印内容部分缺失。所以 ECI 不建议 bas 程序里添加打印，或者把两者调用的时间错开。
输入参数	参数名 | 描述 handle | 连接句柄。 pszCommand | 发送的命令字符串。 uiResponseLength | 返回的字符长度。
输出参数	参数名 | 描述 psResponse | 返回的字符串。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	在线命令函数的使用
详细说明	上位机调用上位机未封装的 Basic 指令功能

指令 265	**ZAux_DirectCommand**
指令原型	int32 __stdcall ZAux_DirectCommand(ZMC_HANDLE handle, const char *pszCommand,char *psResponse, uint32 uiResponseLength)
指令说明	发送字符串命令到控制器，直接方式（不进缓冲区,有少数命令，暂时不支持）。
输入参数	参数名 | 描述 handle | 连接句柄。 pszCommand | 发送的命令字符串。 uiResponseLength | 返回的字符长度。
输出参数	参数名 | 描述 psResponse | 返回的字符串。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	在线命令函数的使用
详细说明	上位机调用上位机未封装的 Basic 指令功能

指令 266	**ZAux_Pause**
指令原型	int32 __stdcall ZAux_Pause(ZMC_HANDLE handle)
指令说明	暂停控制器内部运行的 BASIC 程序。
输入参数	参数名 | 描述 handle | 连接句柄。
输出参数	/

返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	/
详细说明	暂停任务再恢复后，任务继续往下执行。

指令 267	**ZAux_Resume**
指令原型	int32 __stdcall ZAux_Resume(ZMC_HANDLE handle)
指令说明	继续控制器内部运行的 BASIC 程序。
输入参数	参数名 | 描述 handle | 连接句柄。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	/
详细说明	暂停任务再恢复后，任务继续往下执行。

指令 268	**ZAux_BasDown**
指令原型	int32 __stdcall ZAux_BasDown(ZMC_HANDLE handle,const char *Filename,uint32 run_mode)
指令说明	单个.bas 文件生成 ZAR 并且下载到控制器运行
输入参数	参数名 | 描述 handle | 连接句柄。 Filename | BAS 文件名带路径。 run_mode | 下载模式 0-RAM 1-ROM。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	控制程序的下载及编写
详细说明	.bas 文件，可在正运动控制卡直接运行的 Basic 语法文件

指令 269	**ZAux_ZarDown**
指令原型	int32 __stdcall ZAux_ZarDown(ZMC_HANDLE handle,const char *Filename,uint32 run_mode)
指令说明	下载 ZAR 程序到控制器运行

输入参数	参数名 | 描述 handle | 连接句柄。 Filename | ZAR 文件名带路径。 run_mode | 下载模式 0-RAM 1-ROM
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	/
详细说明	ZAR 文件，正运动控制卡直接运行的程序，经过加密后生成的文件

指令 270	**ZAux_Direct_GetVariableInt**
指令原型	int32 __stdcall ZAux_Direct_GetVariableInt (ZMC_HANDLE handle , const char * pname, int * piValue);
指令说明	Basic 整型全局变量读取，也可以是轴参数。
输入参数	参数名 | 描述 handle | 连接句柄。 pname | 全局变量名称/或者制定轴号的轴参数名称 DPOS(0)
输出参数	参数名 | 描述 piValue | 返回值
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	int piValue; ZAux_Direct_GetVariableInt (handle , "DPOS（0）", &piValue); //整型获取轴 0 DPOS 值
详细说明	全局变量，指的是需要在 RTSys 软件中进行设置全局变量的名称，然后再通过上位机去设置读取全局变量的值。 不支持 Basic 结构体的设置读取

指令 271	**ZAux_Direct_GetVariablef**
指令原型	int32 __stdcall ZAux_Direct_GetVariablef (ZMC_HANDLE handle , const char * pname, float* pfValue);
指令说明	basic 浮点型全局变量读取，也可以是参数等等。全局变量，需要在 RTSys 软件中进行设置全局变量的名称，然后再通过上位机去设置全变量的值。

输入参数	<table><tr><td>参数名</td><td>描述</td></tr><tr><td>handle</td><td>连接句柄。</td></tr><tr><td>pname</td><td>全局变量名称/或者制定轴号的轴参数名称 DPOS(0)</td></tr></table>
输出参数	<table><tr><td>参数名</td><td>描述</td></tr><tr><td>pfValue</td><td>返回值</td></tr></table>
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	Float pfValue; ZAux_Direct_GetVariablef (handle , "DPOS（0）", &pfValue); //浮点型获取轴 0 DPOS 值
详细说明	全局变量，指的是需要在 RTSys 软件中进行设置全局变量的名称，然后再通过上位机去设置读取全局变量的值。 不支持 Basic 结构体的设置读取

指令 272	**ZAux_Direct_SetUserArray**
指令原型	int32 __stdcall ZAux_Direct_SetUserArray(ZMC_HANDLE handle, char* arrayname, int arraystart, int numes, float * pfValue);
指令说明	Basic 全局数组的设置。
输入参数	<table><tr><td>参数名</td><td>描述</td></tr><tr><td>handle</td><td>连接标识。</td></tr><tr><td>arrayname</td><td>数组名称</td></tr><tr><td>arraystart</td><td>数组起始元素</td></tr><tr><td>numes</td><td>数组数量</td></tr><tr><td>pfValue</td><td>设置值</td></tr></table>
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	/
详细说明	全局数组，指的是需要在 RTSys 软件中进行设置全局变量的名称，然后再通过上位机去设置全局变量的值。 不支持 Basic 结构体的设置读取

指令 273	**ZAux_Direct_GetUserArray**
指令原型	int32 __stdcall ZAux_Direct_GetUserArray(ZMC_HANDLE handle, char* arrayname, int arraystart, int numes, float *pfValue);

指令说明	Basic 全局数组的读取。
输入参数	参数名 | 描述 handle | 连接标识。 arrayname | 数组名称 arraystart | 数组起始元素 numes | 数组数量
输出参数	参数名 | 描述 pfValue | 读取数组元素的值。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	/
详细说明	全局数组，指的是需要在 RTSys 软件中进行设置全局变量的名称，然后再通过上位机去设置全局变量的值。 不支持 Basic 结构体的设置读取

指令 274	**ZAux_Direct_SetUserVar**
指令原型	int32 __stdcall ZAux_Direct_SetUserVar(ZMC_HANDLE handle,char *varname , float pfValue);
指令说明	设置 Basic 自定义全局变量
输入参数	参数名 | 描述 handle | 连接句柄。 varname | 变量名 pfvalue | 设置值
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	控制程序的下载及编写
详细说明	自定义全局变量，指的是需要在 RTSys 软件中进行设置全局变量的名称，然后再通过上位机去设置变量的值。 不支持 Basic 结构体的设置读取

指令 275	**ZAux_Direct_GetUserVar**
指令原型	int32 __stdcall ZAux_Direct_GetUserVar(ZMC_HANDLE handle,char *varname , float *pfValue);
指令说明	读取 Basic 自定义全局变量。

输入参数	参数名 | 描述 handle | 连接句柄。 varname | 变量名
输出参数	参数名 | 描述 pfvalue | 返回值
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	控制程序的下载及编写
详细说明	自定义全局变量，指的是需要在 RTSys 软件中进行设置全局变量的名称，然后再通过上位机去设置变量的值。 不支持 Basic 结构体的设置读取

指令 276	**ZAux_SetTimeOut**
指令原型	int32 __stdcall ZAux_SetTimeOut(ZMC_HANDLE handle, uint32 timems)
指令说明	命令的延时等待时间。
输入参数	参数名 | 描述 handle | 连接标识。 timems | 等待时间 MS。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	/
详细说明	/

指令 277	**ZAux_TransStringtoInt**
指令原型	int32 __stdcall ZAux_TransStringtoInt (const char * pstringin, int inumes, int * pivlaue);
指令说明	字符串转换为 int
输入参数	参数名 | 描述 pstringin | 数据的字符串。 inumes | 转换数据个数。
输出参数	参数名 | 描述 pivlaue | 转换的数据。

返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	/
详细说明	/

指令 278	**ZAux_TransStringtoFloat**
指令原型	int32 __stdcall ZAux_TransStringtoFloat (const char * pstringin, int inues, int * pfvlaue);
指令说明	字符串转换为 float
输入参数	参数名 | 描述 pstringin | 数据的字符串。 inumes | 转换数据个数。
输出参数	参数名 | 描述 pfvlaue | 转换的数据。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	/
详细说明	/

指令 279	***PZMCAutoUpCallBack**
指令原型	typedef void (*PZMCAutoUpCallBack)(ZMC_HANDLE handle, int32 itypecode, int32 idatalength, uint8 *pdata);
指令说明	主动上报的回调函数格式
输入参数	参数名 | 描述 handle | 卡连接 itypecode | 上传类型码 idatalength | 数据长度 pdata | 数据指针
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	主动上报
详细说明	/

指令 280	**ZAux_SetAutoUpCallBack**
指令原型	int32 __stdcall ZAux_SetAutoUpCallBack (ZMC_HANDLE handle, PZMCAutoUpCallBack pcallback);

指令说明	控制器自动上报。
输入参数	参数名 | 描述 handle | 连接句柄。 pcallback | 函数格式。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	主动上报
详细说明	**主动上报：** 控制器主动上报功能 主动上报消息：控制器发消息给 PC 端

指令 281	**ZAux_CycleUpEnable**
指令原型	int32 __stdcall ZAux_CycleUpEnable(ZMC_HANDLE handle, uint32 cycleindex, float fintervalms, const char* psetesname);
指令说明	使能周期上报。
输入参数	参数名 | 描述 handle | 连接句柄。 cycleindex | 上报通道号，0-最大值（PS：ECI 产品最大值=0，ZMC4 系列，最大值=1） fintervalms | 上报间隔时间, ms 单位, 不能低于控制器总线通讯周期 psetesname | 上报参数选择，语法： 参数 1,例：IDLE 上报默认轴的 IDLE 参数 2(index),例：IDLE（1）上报轴 1 的 IDLE 参数 3(index,numes)例：IDLE（0,3）上报轴 0-轴 2 的 IDLE
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	周期上报
详细说明	（1）每个通道上传的字节数不能超过 1000 字节. （2）每个参数占用的空间有区别，一个 IO 只占一个 bit，4 系列以上产品缺省使用 double 数据类型，3 系列和 XPLC 产品缺省使用 float 数据类型。

	（3）只有网口连接才能周期上传 （4）需要 zmotion.dll 与控制器固件同时支持 （5）可以通过提前设置经常读取的参数主动周期上报，可以减少 PC 主动轮询的时间
详细说明 2	上报支持参数： VP_SPEED(命令当前速度)，MPOS(反馈位置)，DPOS（命令位置）， MSPEED（反馈当前速度)，AXISSTATUS（轴状态)，FE（随动误差）， MOVES_BUFFERED(当前缓冲数)，IDLE(运动状态)， MOVE_CURMARK(当前运动标号)，MARK(锁存触发)， VECTOR_BUFFERED(缓冲剩余距离)，MTYPE(当前运动类型)， DRIVER_FE(驱动器当前随动误差)， DRIVER_TORCH(驱动器当前力矩)， DRIVER_STATUS(驱动器状态)， IN(输入)，OUT(输出)，AIN(模拟量输入)，AOUT(模拟量输出)， TABLE(寄存器). 4 系列控制器增加参数：REMAIN_BUFFER1，剩余的直线运动缓冲数，MTYPE=1 的情况.

指令 282	**ZAux_CycleUpDisable**
指令原型	int32 __stdcall ZAux_CycleUpDisable(ZMC_HANDLE handle, uint32 cycleindex);
指令说明	关闭周期上报使能。
输入参数	参数名 | 描述 handle | 连接句柄。 cycleindex | 上报通道号, 0-最大值-1
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	周期上报
详细说明	需要 zmotion.dll 与控制器固件同时支持 通过提前设置经常读取的参数主动周期上报,可以减少 PC 主动轮询的时间

指令 283	**ZAux_CycleUpGetRecvTimes**
指令原型	uint32 __stdcall ZAux_CycleUpGetRecvTimes(ZMC_HANDLE handle, uint32 cycleindex);

指令说明	获取周期上报收到的包次数。
输入参数	参数名 | 描述 handle | 连接句柄。 cycleindex | 上报通道号，0-最大值-1
输出参数	/
返回值	周期上报收到的包次数
指令示例	周期上报
详细说明	需要 zmotion.dll 与控制器固件同时支持. 超过溢出 调试使用.

指令 284	ZAux_CycleUpForceOnce
指令原型	int32 __stdcall ZAux_CycleUpForceOnce(ZMC_HANDLE handle, uint32 cycleindex);
指令说明	强制上报一次。
输入参数	参数名 | 描述 handle | 连接句柄。 cycleindex | 上报通道号, 0-最大值-1
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	周期上报
详细说明	在运动指令后 idle 可能不准确的情况下调用

指令 285	ZAux_CycleUpReadBuff
指令原型	uint32 __stdcall ZAux_CycleUpReadBuff(ZMC_HANDLE handle, uint32 cycleindex, const char *psetname, uint32 isetindex, double *pvalue);
指令说明	从周期上报里面读取内容,内容获取数据类型为浮点型
输入参数	参数名 | 描述 handle | 连接句柄。 cycleindex | 上报通道号, 0-最大值-1 psetname | 参数名称 isetindex | 参数编号
输入参数	参数名 | 描述 pvalue | 周期上报里面读取的参数内容

返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	周期上报
详细说明	需要 zmotion.dll 与控制器固件同时支持. 通过提前设置经常读取的参数主动周期上报，可以减少 PC 主动轮询的时间.

指令 286	**ZAux_CycleUpReadBuffInt**
指令原型	uint32 __stdcall ZAux_CycleUpReadBuffInt(ZMC_HANDLE handle, uint32 cycleindex, const char *psetname, uint32 isetindex, int32*pvalue);
指令说明	从周期上报里面读取内容,内容获取数据类型为整型
输入参数	参数名 | 描述 handle | 连接句柄。 cycleindex | 上报通道号, 0-最大值-1 psetname | 参数名称 isetindex | 参数编号
输入参数	参数名 | 描述 pvalue | 周期上报里面读取的参数内容
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	周期上报
详细说明	需要 zmotion.dll 与控制器固件同时支持。 通过提前设置经常读取的参数主动周期上报，可以减少 PC 主动轮询的时间。

指令 287	**ZAux_SetTraceFile**
指令原型	int32 __stdcall ZAux_SetTraceFile(int bifTofile,const char *pFilePathName)
指令说明	打印输出调用的 PC 函数库命令到指定文件。

<table>
<tr><td>输入参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>bifTofile</td><td>打印输出模式
0- 关闭
1- 只输出错误命令
2- 只输出运动与参数设置命令
3- 输出所有命令</td></tr>
<tr><td>pFilePathName</td><td>文件名称带绝对路径。预留参数，该参数目前不生效，可任意填写字符串</td></tr>
</table>
</td></tr>
<tr><td>输出参数</td><td>/</td></tr>
<tr><td>返回值</td><td>成功返回值为 0，非 0 详见错误码说明。</td></tr>
<tr><td>指令示例</td><td>调试打印信息</td></tr>
<tr><td>详细说明</td><td>该文件保存在当前程序路径下的\ZmotionLog\下，以 log 日志的方式保存。
注意：该指令只在调试的时候使用，不使用的时候，切记一定要关闭打印输出模式，否则会影响通讯效率</td></tr>
</table>

<table>
<tr><td>指令 288</td><td>ZAux_Trigger</td></tr>
<tr><td>指令原型</td><td>int32 __stdcall ZAux_Trigger(ZMC_HANDLE handle)</td></tr>
<tr><td>指令说明</td><td>示波器触发函数（150723 以后固件版本支持）。</td></tr>
<tr><td>输入参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>handle</td><td>连接句柄。</td></tr>
</table>
</td></tr>
<tr><td>输出参数</td><td>/</td></tr>
<tr><td>返回值</td><td>成功返回值为 0，非 0 详见错误码说明。</td></tr>
<tr><td>指令示例</td><td>示波器的使用</td></tr>
<tr><td>详细说明</td><td>需使用 RTSys 软件打开示波器</td></tr>
</table>

<table>
<tr><td>指令 289</td><td>ZAux_Direct_GetCreep</td></tr>
<tr><td>指令原型</td><td>int32 __stdcall ZAux_Direct_GetCreep(ZMC_HANDLE handle, int iaxis,float *pfValue);</td></tr>
<tr><td>指令说明</td><td>读取 2 次回零爬行速度。</td></tr>
<tr><td>输入参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>handle</td><td>连接句柄。</td></tr>
<tr><td>iaxis</td><td>轴号。</td></tr>
</table>
</td></tr>
</table>

输入参数	参数名 | 描述 *piValue | 返回的爬行速度。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	float fValue; ZAux_Direct_GetCreep(handle, 0, &fValue); //读取轴 0 的回零爬行速度设置值
详细说明	回零运动中，找到原点后，离开原点时的爬行速度

指令 290	**ZAux_Direct_Base**
指令原型	int32 __stdcall ZAux_Direct_Base(ZMC_HANDLE handle, int imaxaxises, int *piAxislist)
指令说明	设置控制器运动的缺省轴号
输入参数	参数名 | 描述 handle | 连接句柄。 imaxaxises | 参与轴数 piAxislist | 轴号数组
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	/
详细说明	/

指令 291	**ZAux_Direct_Backlash**
指令原型	int32 __stdcall ZAux_Direct_Backlash(ZMC_HANDLE handle, int iaxis, bool enable,float dist,float speed,float accel)
指令说明	设置轴的反向间隙，扩展轴无效。
输入参数	参数名 | 描述 handle | 连接句柄。 iaxis | 参与轴号 enable | 使能开关：0 关闭；1 开启 dist | 补偿距离，单位：units speed | 反向间隙补偿速度 accel | 反向间隙补偿加速度
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。

指令示例	反向间隙补偿
详细说明	/

指令 292	ZAux_LocalRTManage
指令原型	uint32 __stdcall ZAux_LocalRTManage(int32 isel, int32 icode)
指令说明	Rt 软核控制台启停控制
输入参数	参数名 | 描述 isel, | 预留 icode | 1-启动 Rt,2-停止 Rt
输出参数	/
返回值	错误码
指令示例	启停软核
详细说明	/

指令 293	ZAux_LocalRTState
指令原型	uint32 __stdcall ZAux_GetLocalRTState(int32 isel)
指令说明	Rt 软核控制台启停控制
输入参数	参数名 | 描述 isel, | 预留
输出参数	0-没有运行，1-运行中，其他：错误码
返回值	其他：错误码
指令示例	启停软核
详细说明	/

指令 294	ZAux_Direct_MoveModify2
指令原型	int32 __stdcall ZAux_Direct_MoveModify2(ZMC_HANDLE handle,int imaxaxises, int* piAxislist, float* pfDisance)
指令说明	强制停止原来的运动，接着按原来的速度与加速度定位到新目标位置。前面没有运动时与 MOVEABS 效果一样，但各轴的运动是独立的，且不会进入运动缓冲。 连续插补时使用 MOVEMODIFY2 会破坏速度连续性。MOVEMODIFY2 同时对多轴运动时不一定为直线插补
输入参数	参数名 | 描述 handle | 预留 imaxaxises | 轴数量 piAxislist | 轴号列表 pfDisance | 绝对距离列表
输出参数	/
返回值	错误码
指令示例	/
详细说明	同 ZAux_Direct_MoveModify， ZAux_Direct_MoveModify2 为多轴 ZAux_Direct_MoveModify 为单轴。

指令 295	ZAux_Execute_GetMoveTime
指令原型	int32 __stdcall ZAux_Execute_GetMoveTime(ZMC_HANDLE handle, int iaxis, float *fValue)
指令说明	读取对应的运动剩余时间
输入参数	参数名 | 描述 handle | 链接句柄 iaxis | 轴号 fValue | 运动的剩余时间，单位 ms
输出参数	/
返回值	错误码
指令示例	ZAux_Direct_SetAxisMode(handle, 0, 2^6);//设置axismode ZAux_Direct_SetVpMode(handle,0,7); //设置vpmode ZAux_Direct_Single_MoveAbs(handle,0,1000); //设置运动1000 float fValue=0; ZAux_Execute_GetMoveTime(handle, 0, &fValue);//获取剩余时间

详细说明	Execute 封装命令，不要频繁获取 VpMode=7,AxisMode 的 bit6 为 1 的模式下生效

指令 296	ZAux_Execute_GetPreCalTime
指令原型	int32 __stdcall ZAux_Execute_GetPreCalTime(ZMC_HANDLE handle,float dis, float *fValue)
指令说明	预估点位运动时间，单位为 ms
输入参数	参数名 | 描述 handle | 链接句柄 dis | 轴的运动距离 fValue | 预估点位运动时间，单位为 ms
输出参数	/
返回值	错误码
指令示例	ZAux_Direct_SetAxisMode(handle, 0, 2^6);//设置axismode ZAux_Direct_SetVpMode(handle,0,7); //设置vpmode ZAux_Direct_Single_MoveAbs(handle,0,1000); //设置运动1000 float fValue=0; ZAux_Execute_ GetPreCalTime (handle, 0, &fValue);//获取预知时间
详细说明	Execute 封装命令，不要频繁获取 VpMode=7,AxisMode 的 bit6 为 1 的模式下生效

指令 297	ZAux_Direct_SetVpMode
指令原型	int32 __stdcall ZAux_Direct_SetVpMode(ZMC_HANDLE handle, int iaxis, int mode)
指令说明	加减速曲线类型选择。

<table>
<tr><td>输入参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>handle</td><td>链接句柄</td></tr>
<tr><td>iaxis</td><td>轴号</td></tr>
<tr><td>mode</td><td>加减速曲线类型选择
<table>
<tr><td>0</td><td>缺省值，使用 sramp 来设置 S 曲线</td></tr>
<tr><td>4</td><td>起步时最大加速度，达到最高速时加速度渐变为 0</td></tr>
<tr><td>6</td><td>新增类型 SS 曲线，加加速度连续的曲线类型，SS 模式比 T 形减速会增加 87%的减速时间。此模式加速时使用 0 模式，减速时才生效，方便连续小线段插补</td></tr>
<tr><td>7</td><td>新增类型 SS 曲线，加加速度连续的曲线类型。动态修改轴参数或连续插补可能导致加加速度无法连续，此时会切换到 0 模式，因此建议 SRAMP 也设置合适值</td></tr>
</table></td></tr>
</table></td></tr>
<tr><td>输出参数</td><td>/</td></tr>
<tr><td>返回值</td><td>错误码</td></tr>
<tr><td>指令示例</td><td></td></tr>
<tr><td>详细说明</td><td>/</td></tr>
</table>

<table>
<tr><td>指令 298</td><td>ZAux_Direct_SetAxisMode</td></tr>
<tr><td>指令原型</td><td>int32 __stdcall ZAux_Direct_SetAxisMode(ZMC_HANDLE handle, int iaxis, int mode)</td></tr>
<tr><td>指令说明</td><td>固件版本 20170616 及以上支持此功能。</td></tr>
<tr><td>输入参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>handle</td><td>链接句柄</td></tr>
<tr><td>iaxis</td><td>轴号</td></tr>
<tr><td>mode</td><td>模式设置直接填写位数数值即可
connect 运动保持
设置 BIT1=1，防止限位和软限位导致 CONNECT 运动退出。
BIT1=0，当碰到限位后，主从轴的 CONNECT 连接关系断开，消除限位报警后，此时操作主轴，从轴不再跟随；
BIT1=1，碰到限位后仍然保持连接关系，消</td></tr>
</table></td></tr>
</table>

		除报警后，从轴仍然保持关联。 BIT5 =0，缺省设置，主轴带编码器时优先跟踪 MPOS。 BIT5=1，设置在齿轮或凸轮运动的主轴上，强制指定跟踪主轴的 DPOS，涉及的指令：CAMBOX 、 CONNECT 、 MOVELINK 、 MOVESLINK 、 MOVESYNC 、 HW_PSWITCH2。 BIT6：缺省 0, 1-使用可以预知准确时间的算法, 目前 VP_MODE=7 下有效. BIT7：缺省 0, 1- MOVEMODIFY 时, 尽可能避免反向运行, 会使用更大的减速度.
输出参数	/	
返回值	错误码	
指令示例		
详细说明	/	

指令 299	ZAux_Direct_SetTableDouble	
指令原型	int32 __stdcall ZAux_Direct_SetTableDouble(ZMC_HANDLE handle,int tabstart, int numes, double *pfValue)	
指令说明	写入 table	
输入参数	参数名	描述
	handle	链接句柄
	tabstart	写入的 TABLE 起始编号
	numes	写入的数量
	pfValue	写入的数据值
输出参数	/	
返回值	错误码	
指令示例	Double value=1.01 ZAux_Direct_SetTableDouble(handle,0, 1, &value);	
详细说明	/	

指令 300	ZAux_Direct_GetTableDouble
指令原型	int32 __stdcall ZAux_Direct_GetTableDouble(ZMC_HANDLE handle, int tabstart, int numes, double *pfValue)

<table>
<tr><td>指令说明</td><td>读取 table，可以一次读取多个</td></tr>
<tr><td>输入参数</td><td><table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>handle</td><td>链接句柄</td></tr>
<tr><td>tabstart</td><td>读取 TABLE 起始编号</td></tr>
<tr><td>numes</td><td>读取的数量</td></tr>
<tr><td>pfValue</td><td>多个时必须分配空间</td></tr>
</table></td></tr>
<tr><td>输出参数</td><td>/</td></tr>
<tr><td>返回值</td><td>错误码</td></tr>
<tr><td>指令示例</td><td>Double value[1];
ZAux_Direct_GetTableDouble(handle,0, 1, value);</td></tr>
<tr><td>详细说明</td><td>/</td></tr>
</table>

指令 301	**ZAux_Direct_MSphericalABSSp**
指令原型	int32 __stdcall ZAux_Direct_MSphericalABSSp(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fend1, float fend2, float fend3, float fcenter1, float fcenter2, float fcenter3, int imode, float fcenter4, float fcenter5)
指令说明	空间圆弧 + 螺旋 插补 ABSSP 运动,20130901 以后的控制器版本支持

<table>
<tr><td>输入参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>handle</td><td>链接句柄</td></tr>
<tr><td>imaxaxises</td><td>参与运动总轴数</td></tr>
<tr><td>piAxislist</td><td>轴号列表</td></tr>
<tr><td>fend1</td><td>第 1 个轴运动距离参数 1，相对与起点</td></tr>
<tr><td>fend2</td><td>第 2 个轴运动距离参数 1，相对与起点</td></tr>
<tr><td>fend3</td><td>第 3 个轴运动距离参数 1，相对与起点</td></tr>
<tr><td>fcenter1</td><td>第 1 个轴运动距离参数 2，相对与起点</td></tr>
<tr><td>fcenter2</td><td>第 2 个轴运动距离参数 2，相对与起点</td></tr>
<tr><td>fcenter3</td><td>第 3 个轴运动距离参数 2，相对与起点</td></tr>
<tr><td>imode</td><td>指定前面参数的意义
<table>
<tr><td>值</td><td>描述</td></tr>
<tr><td>0</td><td>当前点，中间点，终点三点定圆弧，距离参数 1 为终点距离，距离参数 2 为中间点距离。</td></tr>
<tr><td>1</td><td>走最小的圆弧，距离参数 1 为终点距离，距离参数 2 为圆心的距离。</td></tr>
<tr><td>2</td><td>当前点，中间点，终点三点定圆，距离参数 1 为终点距离，距离参数 2 为中间点距离。</td></tr>
<tr><td>3</td><td>先走最小的圆弧，再继续走完整圆，距离参数 1 为终点距离，距离参数 2 为圆心的距离。</td></tr>
</table></td></tr>
<tr><td>fcenter4</td><td>第 4 个轴运动距离参数</td></tr>
<tr><td>fcenter5</td><td>第 5 个轴运动距离参数</td></tr>
</table>
</td></tr>
<tr><td>输出参数</td><td>/</td></tr>
<tr><td>返回值</td><td>错误码</td></tr>
<tr><td>指令示例</td><td>空间圆弧</td></tr>
<tr><td>详细说明</td><td>/</td></tr>
</table>

指令 302	**ZAux_Direct_GetBusAxisEnable**
指令原型	int32 __stdcall ZAux_Direct_GetBusAxisEnable(ZMC_HANDLE handle, int iaxis, int *piValue)
指令说明	读取总线轴使能状态

输入参数	<table><tr><td>参数名</td><td>描述</td></tr><tr><td>handle</td><td>链接句柄</td></tr><tr><td>iaxis</td><td>轴号</td></tr><tr><td>pfValue</td><td>返回的使能状态</td></tr></table>
输出参数	/
返回值	错误码
指令示例	int piValue=0; int ret=ZAux_Direct_GetBusAxisEnable(handle, 0, &piValue);
详细说明	直接获取的状态字的使能状态，而非是 axisEnable 的值

指令 303	**ZAux_Scope**
指令原型	int32 __stdcall ZAux_Scope(ZMC_HANDLE handle,int iEnable,int iPeriod,int iStartTab,int iStopTab, char * ParaNamep)
指令说明	数据采样
输入参数	<table><tr><td>参数名</td><td>描述</td></tr><tr><td>handle</td><td>链接句柄</td></tr><tr><td>iEnable</td><td>是否使能</td></tr><tr><td>iPeriod</td><td>采样周期，以控制器周期为单位</td></tr><tr><td>iStartTab</td><td>数据起始 TABLE 位置</td></tr><tr><td>iStopTab</td><td>数据停止 TABLE 位置</td></tr><tr><td>iParaNum</td><td>采集数据源数量，最大 8 个</td></tr><tr><td>ParaNamep</td><td>采集数据源字符名称</td></tr></table>
输出参数	/
返回值	错误码
指令示例	12.4.3-3 数据采样
详细说明	/

指令 304	**ZAux_GetScopeLength**
指令原型	int32 __stdcall ZAux_GetScopeLength(ZMC_HANDLE handle,int *ilen)
指令说明	获取示波器采样长度
输入参数	<table><tr><td>参数名</td><td>描述</td></tr><tr><td>handle</td><td>链接句柄</td></tr></table>
输出参数	<table><tr><td>参数名</td><td>描述</td></tr></table>

	ilen	数据长度	
返回值	错误码		
指令示例	12.4.3-3 数据采样		
详细说明	/		

指令 305	ZAux_Direct_MoveOpDelay
指令原型	int32 __stdcall ZAux_Direct_MoveOpDelay(ZMC_HANDLE handle, float uTime, int nAxis)
指令说明	设置 op 口提前打开或关闭
输入参数	参数名 | 描述 handle | 链接句柄 uTime | 延时时间，ms nAxis | 轴号
输出参数	/
返回值	错误码
指令示例	ZAux_Direct_SetParam(handle,” AXIS_ZSET”,0, 19);//启动精准输出 ZAux_Direct_MoveOpDelay(handle, 100, 0);//设置延后 100ms 输出 ZAux_Direct_Single_Move(handle, 0, 100)；//单轴运动 100 ZAux_Direct_MoveOp(handle, 0,0, 1);//轴 0 正向运动 100 后延时 100ms 把 out0 的状态置为 1 ZAux_Direct_Single_Move(handle, 0, 100)；//单轴运动 100
详细说明	为支持 HW 功能的控制器，且为对应的 HW 口，当 MOVE_OP 精准功能使用时，可以调整实际触发 OP 操作的时间，毫秒(ms)单位，支持小数，实际最长延时时间 100ms。 设置为负数，可以提前打开 OP，一般步进可以提前 2ms，伺服可以提前 20ms，点胶起胶可以使用。

指令 306	ZAux_DownMem3FileRam
指令原型	int32 __stdcall ZAux_DownMem3FileRam(ZMC_HANDLE handle, const char* pbuffer, uint32 buffsize,char* File3Name,int32 ifile3num)
指令说明	打包字符串运动下发

输入参数	参数名 | 描述 handle | 链接句柄 pbuffer | 运动字符串 buffsize | 字符串下载大小 File3Name | 下载生成的 3 次文件名称 ifile3num | 选择那个 3 次文件缓冲
输出参数	/
返回值	错误码
指令示例	三次文件的使用
详细说明	/

指令 307	ZAux_Run3FileRam
指令原型	int32 __stdcall ZAux_Run3FileRam(ZMC_HANDLE handle,int32 ifile3num ,int32 itasknum)
指令说明	运行下发的文件
输入参数	参数名 | 描述 handle | 链接句柄 ifile3num | 3 次文件编号 itasknum | 运动任务号
输出参数	/
返回值	错误码
指令示例	三次文件的使用
详细说明	/

指令 308	ZAux_Down3FileRam
指令原型	int32 __stdcall ZAux_Down3FileRam(ZMC_HANDLE handle, const char* pfilename, const char* pfilenameinControl,int32 ifile3num)
指令说明	下载 3 次文件到 RAM
输入参数	参数名 | 描述 handle | 链接句柄 pfilename | PC 文件名，PC 路径+文件名(Z3P 或者 NC) pfilenameinControl | 存储在控制器文件名(Z3P 或者 NC) ifile3num | 3 次文件编号
输出参数	/

返回值	错误码
指令示例	三次文件的使用
详细说明	/

指令 309	ZAux_3FileRamGetRemainSpace
指令原型	int32 __stdcall ZAux_3FileRamGetRemainSpace(ZMC_HANDLE handle,int32 ifile3num, uint32 *premainbyte)
指令说明	动态下载 3 次文件，读取可写的空间
输入参数	参数名：handle — 链接句柄；ifile3num — 3 次文件编号
输出参数	参数名：premainbyte — 字节数
返回值	错误码
指令示例	三次文件的使用
详细说明	/

指令 310	ZAux_3FileRamDownBegin
指令原型	int32 __stdcall ZAux_3FileRamDownBegin(ZMC_HANDLE handle, int32 ifile3num,uint32 *premainbyte,char*File3Name)
指令说明	动态下载 3 次文件，启动下载
输入参数	参数名：handle — 链接句柄；ifile3num — 3 次文件编号；File3Name — 设置 3 次文件名.3 次文件不会自动运行，需调用 ZAux_Run3FileRam 启动
输出参数	参数名：premainbyte — 剩余控件字节数
返回值	错误码
指令示例	三次文件的使用
详细说明	/

指令 311	ZAux_3FileRamDownPart
指令原型	int32 __stdcall ZAux_3FileRamDownPart(ZMC_HANDLE handle,int32

	ifile3num, const char* pbuffer, uint32 buffsize, uint32 *premainbyte)
指令说明	动态下载 3 次文件，一次下载的内容必须是完整的单行或多行,没有下载完成时，后面用\n 结尾，全部下载完成时，加\0 结尾 \0 会导致任务结束，需要再启动可以再次调用 ZMC_Run3FileRam
输入参数	参数名 | 描述 handle | 链接句柄 ifile3num | 3 次文件编号 File3Name | 设置 3 次文件名.3 次文件不会自动运行，需调用 ZAux_Run3FileRam 启动
输出参数	参数名 | 描述 premainbyte | 剩余控件字节数
返回值	错误码
指令示例	三次文件的使用
详细说明	/

指令 312	ZAux_3FileRamDownEnd
指令原型	int32 __stdcall ZAux_3FileRamDownEnd(ZMC_HANDLE handle,int32 ifile3num)
指令说明	动态下载 3 次文件，下载结束，自动加\0，并且调用后再次调用 ZMC_3FileRamDownPart 会返回错误
输入参数	参数名 | 描述 handle | 链接句柄 ifile3num | 三次文件编号
输出参数	/
返回值	错误码
指令示例	三次文件的使用
详细说明	/

指令 313	ZAux_Direct_MoveOpAdist
指令原型	int32 __stdcall ZAux_Direct_MoveOpAdist(ZMC_HANDLE handle, int iaxis,float distance)
指令说明	**提前关胶。** **针对 ZAux_Direct_MoveOp 提前一定距离输出设置，缺省 0 不生效，设置正数会比缺省输出位置提前指定矢量距离，设置负数则延后指定矢量距离。**

	只能对 ZAux_Direct_MoveOp 单个 OP 操作有效，对每个轴来说，操作的动作是顺序的，第一个改变位置的 ZAux_Direct_MoveOp 操作没有完成，后面的不会操作 MOVEOPADIST 带入缓冲后会自动清零，避免影响其它 ZAux_Direct_MoveOp。 对支持 HW 功能的控制器，可以由 AXIS_ZSET 来启动 HW 精准输出，同 ZAux_Direct_MoveOp 精准，此指令只支持 MOVE 插补类与 MOVESCAN 运动，凸轮、MOVE_PT 等不支持，支持 MOVE 指令跨小线段提前输出。
输入参数	参数名 | 描述 handle | 连接句柄 iaxis | 轴号 distance | 提前/延后关胶距离
输出参数	/
返回值	错误码
指令示例	ZAux_Direct_Single_Move(handle, 0, 100)；//单轴运动 100 ZAux_Direct_MoveOp(handle, 0,0, 1);//轴 0 正向运动 100 后把 out0 的状态置为 1 ZAux_Direct_Single_Move(handle, 0, 100)；//单轴运动 100 ZAux_Direct_MoveOpAdist(handle, 0,10.0);// 设置提前 10 个单位距离关闭 out0 ZAux_Direct_MoveOp(handle, 0,0, 0);//轴 0 正向运动 200 后把 out0 的状态置为 0
详细说明	4 系列控制器，fast 固件 190201 以后固件支持

指令 314	**ZAux_Direct_GetAxisStatus2**
指令原型	int32 __stdcall ZAux_Direct_GetAxisStatus2(ZMC_HANDLE handle, int iaxis,int homeIn ,int *piValue,int *iIdle,int *bHomestaus,int *BusEnableStaus)
指令说明	读取总线轴的状态
输入参数	参数名 | 描述 handle | 连接句柄 iaxis | 轴号 homeIn | 原点映射 IO 编号,（编号小于 0，则 homestatus 不生效）

输出参数	<table><tr><td>参数名</td><td>描述</td></tr><tr><td>piValue</td><td>轴状态返回值</td></tr><tr><td>iIdle</td><td>0 - 运动中 其他 - 无运动</td></tr><tr><td>bHomestaus</td><td>1 - 有，0-无</td></tr><tr><td>BusEnableStaus</td><td>总线轴使能状态</td></tr></table>
返回值	错误码
指令示例	int iaxis=0;//轴号 int homeIn=0;//原点编号 int cAxisStatus;//获取上来的轴状态 int cIdle;//获取上来的命令运动状态 int cHomestaus;//获取上来的原点状态 int cBusEnableStaus;//获取上来的总线轴状态 ZAux_Direct_GetAxisStatus2(handle, iaxis, homeIn,&cAxisStatus,&cIdle,& cHomestaus,& cBusEnableStaus);
详细说明	/

指令 315	**ZAux_Direct_GetHomeStatus2**
指令原型	int32 __stdcall ZAux_Direct_GetHomeStatus2(ZMC_HANDLE handle,uint32 iaxis,uint32 * homestatus)
指令说明	回零完成状态 2
输入参数	<table><tr><td>参数名</td><td>描述</td></tr><tr><td>handle</td><td>连接句柄</td></tr><tr><td>iaxis</td><td>轴号</td></tr></table>
输出参数	<table><tr><td>参数名</td><td>描述</td></tr><tr><td>homestatus</td><td>回零完成标志 0 – 回零异常；1 – 回零成功，2，回零中</td></tr></table>
返回值	错误码
指令示例	int ret=0; uint32 mHomeStatus=0; ret=ZAux_Direct_Single_Datum(handle,0,14);//控制轴 0 负限回零 sleep(1); ZAux_Direct_GetHomeStatus2(handle,0,&mHomeStatus); While(mHomeStatus==2) {

	ZAux_Direct_GetHomeStatus2(handle,0,&mHomeStatus); } If (mHomeStatu) { Printf(“home ok”); }else { Printf(home error); }
详细说明	/

指令 316	ZAux_BusCmd_GetHomeStatus2
指令原型	int32 __stdcall ZAux_BusCmd_GetHomeStatus2(ZMC_HANDLE handle,uint32 iaxis,uint32 * homestatus)
指令说明	回零完成状态 2
输入参数	参数名 | 描述 handle | 连接句柄 iaxis | 轴号
输出参数	参数名 | 描述 homestatus | 回零完成标志 0－ 回零未成功；1－ 回零成功；2－ 回零运动中
返回值	错误码
指令示例	int ret=0; uint32 mHomeStatus=0; ret=ZAux_BusCmd_Datum(handle,0,-1);//调用驱动器默认的回零方式 sleep(1); ZAux_BusCmd_GetHomeStatus2(handle,0,&mHomeStatus); While(mHomeStatus==2) { ZAux_BusCmd_GetHomeStatus2(handle,0,&mHomeStatus); } If (mHomeStatu) { Printf(“home ok”);

	}else { Printf(home error); }
详细说明	/

指令 317	ZAux_BusCmd_SDOWrite2
指令原型	int32 __stdcall ZAux_BusCmd_SDOWrite2(ZMC_HANDLE handle,uint32 slot,uint32 node,uint32 index,uint32 subindex ,uint32 type ,char * in_string)
指令说明	写节点 SDO 参数信息(兼容更新功能)
输入参数	参数名：描述 handle：连接句柄 slot：槽位号，缺省 0 node：节点编号 index：对象字典编号（注意函数为 10 进制数据） subindex：子编号（注意函数为 10 进制数据） type：数据类型 1 boolean 2 integer 8 3 integer 16 4 integer 32 5 unsigned 8 6 unsigned 16 7 unsigned 32 8 flot 9 string in_string：设定的数据值（以字符串格式写入）
输出参数	
返回值	错误码
指令示例	
详细说明	相比 ZAux_BusCmd_SDOWrite 多了 flot 与 string 功能

指令 318	ZAux_BusCmd_SDORead2
指令原型	int32 __stdcall ZAux_BusCmd_SDORead2(ZMC_HANDLE handle,uint32

<table>
<tr><td></td><td>slot,uint32 node,uint32 index,uint32 subindex ,uint32 type ,char * Out_string)</td></tr>
<tr><td>指令说明</td><td>读取节点 SDO 参数信息(兼容更新功能)</td></tr>
<tr><td>输入参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>handle</td><td>连接句柄</td></tr>
<tr><td>slot</td><td>槽位号，缺省 0</td></tr>
<tr><td>node</td><td>节点编号</td></tr>
<tr><td>index</td><td>对象字典编号（注意函数为 10 进制数据）</td></tr>
<tr><td>subindex</td><td>子编号（注意函数为 10 进制数据）</td></tr>
<tr><td>type</td><td>数据类型
<table>
<tr><td>1</td><td>boolean</td></tr>
<tr><td>2</td><td>integer 8</td></tr>
<tr><td>3</td><td>integer 16</td></tr>
<tr><td>4</td><td>integer 32</td></tr>
<tr><td>5</td><td>unsigned 8</td></tr>
<tr><td>6</td><td>unsigned 16</td></tr>
<tr><td>7</td><td>unsigned 32</td></tr>
<tr><td>8</td><td>float</td></tr>
<tr><td>9</td><td>string</td></tr>
</table></td></tr>
</table>
</td></tr>
<tr><td>输出参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>Out_string</td><td>返回值内容,以字符串形式输出</td></tr>
</table>
</td></tr>
<tr><td>返回值</td><td>错误码</td></tr>
<tr><td>指令示例</td><td></td></tr>
<tr><td>详细说明</td><td>/</td></tr>
</table>

<table>
<tr><td>指令 319</td><td>ZAux_Up3File</td></tr>
<tr><td>指令原型</td><td>int32 __stdcall ZAux_Up3File(ZMC_HANDLE handle, const char* pfilename, const char* pfilenameinControl)</td></tr>
<tr><td>指令说明</td><td>上传三次文件(直接将文件上传到本地)</td></tr>
<tr><td>输入参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>handle</td><td>连接句柄</td></tr>
<tr><td>pfilename</td><td>上传文件保存到本地的路径+文件名</td></tr>
<tr><td>pfilenameinControl</td><td>控制器里的三次文件名</td></tr>
</table>
</td></tr>
<tr><td>输出参数</td><td>/</td></tr>
<tr><td>返回值</td><td>错误码</td></tr>
</table>

指令示例	
详细说明	/

指令 320	ZAux_Up3FileToMem
指令原型	int32 __stdcall ZAux_Up3FileToMem(ZMC_HANDLE handle, char* pbuffer, uint32 buffsize, const char* pfilenameinControl, uint32* puifilesize)
指令说明	上传三次文件(将控制器文件内存上传到你提供的 buffer 中)
输入参数	参数名 / 描述 handle / 连接句柄 pbuffer / 指定上传的 buff 地址 buffsize / Buff 的长度 pfilenameinControl / 控制器的文件名 puifilesize / 文件大小
输出参数	/
返回值	错误码
指令示例	
详细说明	/

指令 321	ZAux_BusCmd_NodePdoReadFloat
指令原型	int32 __stdcall ZAux_BusCmd_NodePdoReadFloat(ZMC_HANDLE handle,uint32 inode,uint32 index,uint32 subindex , uint32 type,float* ivalue)
指令说明	PDO 读操作（flot 方式）。

<table>
<tr><td>输入参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>handle</td><td>连接句柄。</td></tr>
<tr><td>inode</td><td>设备编号。</td></tr>
<tr><td>index</td><td>对象字典编号。</td></tr>
<tr><td>subindex</td><td>对象字典子编号。</td></tr>
<tr><td>type</td><td>数据类型：
<table>
<tr><td>1</td><td>boolean</td></tr>
<tr><td>2</td><td>integer 8</td></tr>
<tr><td>3</td><td>integer 16</td></tr>
<tr><td>4</td><td>integer 32</td></tr>
<tr><td>5</td><td>unsigned 8</td></tr>
<tr><td>6</td><td>unsigned 16</td></tr>
<tr><td>7</td><td>unsigned 32</td></tr>
</table>
</td></tr>
</table>
</td></tr>
<tr><td>输出参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>value</td><td>读取数据值。</td></tr>
</table>
</td></tr>
<tr><td>返回值</td><td>成功返回值为 0，非 0 详见错误码说明。</td></tr>
<tr><td>指令示例</td><td></td></tr>
<tr><td>详细说明</td><td></td></tr>
</table>

指令 322	**ZAux_BusCmd_NodePdoWriteFloat**
指令原型	int32 __stdcall ZAux_BusCmd_NodePdoWriteFloat(ZMC_HANDLE handle,uint32 inode,uint32 index,uint32 subindex, uint32 type,float value)
指令说明	PDO 写操作（float 方式）。

<table>
<tr><td>输入参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>handle</td><td>连接句柄。</td></tr>
<tr><td>inode</td><td>设备编号。</td></tr>
<tr><td>index</td><td>对象字典编号。</td></tr>
<tr><td>subindex</td><td>对象字典子编号。</td></tr>
<tr><td>type</td><td>数据类型：
<table>
<tr><td>1</td><td>boolean</td></tr>
<tr><td>2</td><td>integer 8</td></tr>
<tr><td>3</td><td>integer 16</td></tr>
<tr><td>4</td><td>integer 32</td></tr>
<tr><td>5</td><td>unsigned 8</td></tr>
<tr><td>6</td><td>unsigned 16</td></tr>
<tr><td>7</td><td>unsigned 32</td></tr>
</table>
</td></tr>
<tr><td>value</td><td>写入数据值。</td></tr>
</table>
</td></tr>
<tr><td>输出参数</td><td>/</td></tr>
<tr><td>返回值</td><td>成功返回值为 0，非 0 详见错误码说明。</td></tr>
<tr><td>指令示例</td><td></td></tr>
<tr><td>详细说明</td><td></td></tr>
</table>

<table>
<tr><td>指令 323</td><td>ZAux_ZpjDown</td></tr>
<tr><td>指令原型</td><td>int32__stdcall ZAux_ZpjDown(ZMC_HANDLE handle,const char *ZpjName,const char *ZarName,const char *pPass, uint32 uid,uint32 run_mode)</td></tr>
<tr><td>指令说明</td><td>下载 ZAR 程序到控制器运行</td></tr>
<tr><td>输入参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>handle</td><td>连接句柄。</td></tr>
<tr><td>ZpjName</td><td>ZPJ 文件来源路径</td></tr>
<tr><td>ZarName</td><td>生成的 Zar 文件路径</td></tr>
<tr><td>pPass</td><td>软件密码,绑定 APP_PASS 没有密码时 pPass=NULL</td></tr>
<tr><td>uint32 uid</td><td>绑定控制器唯一 ID，0-不绑定</td></tr>
<tr><td>run_mode</td><td>下载模式 0-RAM 1-ROM</td></tr>
</table>
</td></tr>
<tr><td>输出参数</td><td>/</td></tr>
<tr><td>返回值</td><td>成功返回值为 0，非 0 详见错误码说明。</td></tr>
</table>

指令示例	Char * ZpjFileName= "C:\\Users\\Administrator\\Desktop\\Frame11Demo\\ZmcZpj.zpj"; //需要下载的 zpj 路径 Char *ZarFileName= "C:\\Users\\Administrator\\Desktop\\Frame11Demo\\ZmcZpj.zar"; //生成的 ZAR 路径名称 int iret=ZAux_ZpjDown(g_handle,ZpjFileName,m_ZarFileName,NULL,0,1); //zpj 下载
详细说明	ZAR 文件，正运动控制卡直接运行的程序，经过加密后生成的文件

指令 324	ZAux_Down3FileRom
指令原型	int32__stdcall ZAux_Down3FileRom(ZMC_HANDLE handle, const char* pfilename, const char* pfilenameinControl);
指令说明	运行下发的文件
输入参数	参数名：描述 handle：链接句柄 pfilename：3 次文件名称 pfilenameinControl：支持 NC 文件名，必须带名称，否则无法区分 bas 还是 NC 文件
输出参数	/
返回值	错误码
指令示例	/
详细说明	/

指令 325	ZAux_Check3File
指令原型	int32__stdcall ZAux_Check3File(ZMC_HANDLE handle, const char* pfilenameinControl, uint8 *pbIfExist, uint32 *pFileSize)
指令说明	检测控制器内三次文件是否存在
输入参数	参数名：描述 handle：连接句柄。 pfilenameinControl：三次文件名字 pbIfExist：是否存在标志 pFileSize：文件大小
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。

指令示例	/
详细说明	ZAR 文件，正运动控制卡直接运行的程序，经过加密后生成的文件

指令 326	ZAux_Delete3File
指令原型	int32__stdcall ZAux_Delete3File(ZMC_HANDLE handle, const char* pfilenameinControl)
指令说明	删除控制器上的 3 次文件
输入参数	参数名 / 描述 handle / 连接句柄。 pfilenameinControl / 三次文件名字 pbIfExist / 是否存在标志 pFileSize / 文件大小
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	/
详细说明	ZAR 文件，正运动控制卡直接运行的程序，经过加密后生成的文件

指令 327	ZAux_FindFirst3File
指令原型	int32__stdcall ZAux_FindFirst3File(ZMC_HANDLE handle, char* pfilenameinControl, uint32 *pFileSize)
指令说明	获取控制器上的首个文件， 文件名为空表示文件不不存在
输入参数	参数名 / 描述 handle / 连接句柄。 pfilenameinControl / 获取的三次文件名字，为空表示文件不不存在 pFileSize / 文件大小
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	/
详细说明	ZAR 文件，正运动控制卡直接运行的程序，经过加密后生成的文件

指令 328	ZAux_FindNext3File
指令原型	int32__stdcall ZAux_FindNext3File(ZMC_HANDLE handle, char* pfilenameinControl, uint32 *pFileSize)
指令说明	获取控制器上的下个文件， 文件名为空表示文件不不存在

输入参数	参数名	描述
	handle	连接句柄。
	pfilenameinControl	获取的三次文件名字，为空表示文件不不存在
	pFileSize	文件大小
输出参数	/	
返回值	成功返回值为 0，非 0 详见**错误码**说明。	
指令示例	/	
详细说明	ZAR 文件，正运动控制卡直接运行的程序，经过加密后生成的文件	

指令 329	**ZAux_GetCur3File**	
指令原型	int32__stdcall ZAux_GetCur3File(ZMC_HANDLE handle, char* pfilenameinControl, uint32 *pFileSize)	
指令说明	查找控制器上的当前文件	
输入参数	参数名	描述
	handle	连接句柄。
	pfilenameinControl	获取的三次文件名字，为空表示文件不不存在
	pFileSize	文件大小
输出参数	/	
返回值	成功返回值为 0，非 0 详见**错误码**说明。	
指令示例	/	
详细说明	ZAR 文件，正运动控制卡直接运行的程序，经过加密后生成的文件	

指令 330	**ZAux_RemoveAll3Files**	
指令原型	int32__stdcall ZAux_RemoveAll3Files(ZMC_HANDLE handle)	
指令说明	删除控制器上所有的三次的文件	
输入参数	参数名	描述
	handle	连接句柄。
输出参数	/	
返回值	成功返回值为 0，非 0 详见**错误码**说明。	
指令示例	/	
详细说明	ZAR 文件，正运动控制卡直接运行的程序，经过加密后生成的文件	

指令 331	ZAux_GetCur3File
指令原型	int32__stdcall ZAux_Up3File(ZMC_HANDLE handle, const char* pfilename, const char* pfilenameinControl)
指令说明	上传三次文件
输入参数	参数名 | 描述 handle | 连接句柄。 pfilename | 控制器内的三次文件名字 pfilenameinControl | PC 的三次文件地址+名字
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	/
详细说明	ZAR 文件，正运动控制卡直接运行的程序，经过加密后生成的文件

指令 332	ZAux_Up3File
指令原型	int32__stdcall ZAux_Up3File(ZMC_HANDLE handle, const char* pfilename, const char* pfilenameinControl)
指令说明	上传三次文件
输入参数	参数名 | 描述 handle | 连接句柄。 pfilename | 控制器内的三次文件名字 pfilenameinControl | PC 的三次文件地址+名字
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	/
详细说明	ZAR 文件，正运动控制卡直接运行的程序，经过加密后生成的文件

指令 333	ZAux_Up3FileToMem
指令原型	int32__stdcall ZAux_Up3FileToMem(ZMC_HANDLE handle, char* pbuffer, uint32 buffsize, const char* pfilenameinControl, uint32* puifilesize)
指令说明	上传三次文件,指定大小

<table>
<tr><td rowspan="7">输入参数</td><td>参数名</td><td>描述</td><td rowspan="7"></td></tr>
<tr><td>handle</td><td>连接句柄。</td></tr>
<tr><td>pfilename</td><td>控制器内的三次文件名字</td></tr>
<tr><td>Buffsize</td><td>文件大小</td></tr>
<tr><td>pfilenameinControl</td><td>PC 的三次文件地址+名字</td></tr>
<tr><td>puifilesize</td><td>文件大小</td></tr>
<tr><td colspan="3"></td></tr>
<tr><td>输出参数</td><td colspan="3">/</td></tr>
<tr><td>返回值</td><td colspan="3">成功返回值为 0，非 0 详见错误码说明。</td></tr>
<tr><td>指令示例</td><td colspan="3">/</td></tr>
<tr><td>详细说明</td><td colspan="3">ZAR 文件，正运动控制卡直接运行的程序，经过加密后生成的文件</td></tr>
</table>

第十四章 指令返回值

14.1 指令返回值

用户在程序中通过调用动态库封装的函数，向运动控制器发送指令，运动控制器根据收到的指令来运动。

运动控制器收到指令后，向发送指令的主机返回指令执行结果，指示指令是否正确执行。

14.2 指令返回值详细说明

错误码	意义	例子
210	文件过大	
212	状态错误	Resume 时为非暂停状态。
213	文件下载上传出错，丢包。	PC 函数调用返回此错误
214	下载文件的长度校验错误。	
215	缓冲长度不够	发送字符串命令过长时返回此错误。
217	控制器不支持或禁止的功能。	
218	调用传递的参数错误。	
219	下载冲突，同时启动了多个文件下载。	
220	文件名错误，有特殊字符。	
221	文件名错误，超过长度。	
222	文件不存在	
223	密码保护限制。	
224	密码保护限制 2。	
260	硬件错误	
261	磁盘没有格式化	
262	RTC 错误	
263	NORFLASH 错误	
264	RAM 错误	
265	NANDFLASH 错误	
266	U 盘错误	
267	FPGA 错误	

268	以太网硬件错误	
271	备份电源错误	
272	子卡不存在	
273	文件丢失	
274	系统文件错误	
275	无主控，子卡上产生	
276	程序文件校验错误	
277	程序文件错误导致不启动	
278	ZAR 校验 APPPASS 出错	
279	ZAR 校验 ID 出错	
280	BAS 文件超过最大数量	
281	子卡 ID 冲突，或多主冲突。	
282	不支持的功能	
284	zar 与控制器不匹配	
285	图片文件错误	
286	字体文件错误	
288	以上异常，导致下次开机报错。	
1000	运动模块返回错误偏移	
1002	无运动缓冲	
1004	从轴运动中	
1005	不支持的运动功能	
1006	圆弧位置错误	
1007	椭圆 AB 参数错误	
1008	运动模块输入参数错误	
1009	运动中，无法操作	
1010	暂停等重复操作	
1011	IDLE 无法做暂停等操作	
1012	当前运动不支持暂停	
1013	找不到暂停点	
1014	ATYPE 不支持	
1015	ZCAN 的 ATYPE 冲突	
1016	轴不支持的功能	
1017	FRAME 校正数据错误	
1018	FRAME 校正数据过少	
1019	FRAME 校正数据满足条件的数据过少	
1020	FRAME 校正数据辅助参数过少	

1021	FRAME 校正数据间隔过小，小于关节轴数	
1022	FRAME 的输入坐标错误	
1023	FRAME 状态下坐标不能强制修改	
1024	FRAME 逆解异常	
1025	不是 FRAME 状态	
1026	FRAME HAND 错误	
1027	姿态在插补中不能切换	
1028	特殊关节轴与虚拟轴当量要求一样。	
1030	CORNERMODE 7 位设置了但不支持此运动	
1031	CORNERMODE 7 位设置了但不是 FRAME 状态	
1032	AXIS_ADDRESS 错误	
2000	RTBasic 模块偏移	
2021	手动停止	
2022	因其他任务错误导致本任务停止	
2023	试图修改只读状态参数	
2024	数组越界	
2025	变量数超过控制器规格	
2026	数组数超过控制器规格	
2027	数组空间超过控制器规格	
2028	SUB 数超过控制器规格	
2029	标识符命名错误	
2030	标识符命名过长	
2031	没有右括号。	
2032	不认识的字符。	
2033	表达式中碰到不认识的名称。	
2034	SUB 不能在表达式中使用	
2043	不认识的命令标识符，当前行第一个标识名称。	
2044	堆栈溢出	
2045	数学表达式太复杂，不同控制器的规格不一样。	
2046	没有找到结束引用标号	
2047	指令没有返回值，不能用于表达式计算。	
2048	函数必须返回值，不用在一行的开头地方。	
2049	特殊指令必须单独一行	

2050	参数或数组需要索引。	
2051	变量不能使用索引	
2052	数组重定义且长度不一致。	
2053	数组定义长度参数错误，负数或过大。	
2054	标识符已经定义为 SUB 过程，不能再做其他用途。	
2055	标识符已经定义为参数，不能再做其他用途。	
2056	标识符预留，不能使用。	
2057	出现不能识别的字符	
2058	SUB 调用重复出栈。	
2060	语法格式错误	
2062	函数参数范围错误	包括任务号超过范围也返回这个错误。自动运行任务号出错也是这个错误码。
2063	函数参数过多	
2064	函数参数太少	
2065	缺少操作数	
2066	操作符后面缺少操作数	
2067	操作符前面缺少操作数	
2068	不认识的操作符	
2069	需要双目操作符	
2070	CALL 必须调用 SUB	
2072	需要赋值符号	
2073	空文件	
2074	SUB 定义的标识符名称冲突。	
2075	要启动的任务已经运行中。	
2076	多个参数要使用逗号隔开。	
2077	括号不配对，无左括号。	
2078	IF 判断的嵌套太多。	
2079	循环语句嵌套太多。	
2080	插补轴数太少。	
2081	CONST 常量，不能修改。	
2082	命令不能从 PC 在线发送。	
2083	SUB 定义的参数太多。	
2084	SUB 带参数，不能用于 GOTO 语句。	

2085	局部标识符定义太多。	
2086	LOCAL 变量名与文件变量名或其它标识符名称冲突。	
2087	LOCAL 不支持数组定义。	
2088	GSUB 定义的参数字母重复。	
2089	GSUB 定义的参数只能为单字母。	
2090	不能修改只读参数。	
2091	GSUB_IFPARA 函数使用场合错误。	
2092	除数为零	
2093	超过缓冲	
2094	在线命令阻赛时间过长。	
2095	参数重名	
2096	值没有初始化就使用了	
2097	轴号冲突	
2099	内部错误	
2100	SCANEDGE 个数过多	
2101	ZINDEX 类型不匹配	
2901	系统错误，定义的标识符过多。	
	包括变量，数组，过程，过程参数等等。	
3201	超过缓冲	
3202	文件非正常结束	
3204	内部状态错误	
3205	不支持的功能	
3206	内部调用参数错误	
3231	资源不够	
3242	os 错误	
3243	U 盘没有插入	
3244	文件重复打开	
3245	文件过大	
3248	文件名错误	
3249	文件名过长	
3250	文件不存在	
3301	圆弧的三点在一条线上。	
3302	两条直线平行，没有交点。	
3401	MODBUS 主端参数错误，一般长度超过	
3402	消息响应超时	

3407	Modbus 返回参数错误	
3408	Modbus 返回不支持	
3421	modbus 从端返回不支持的功能码	
3422	modbus 从端返回地址空间错误	
3423	modbus 从端返回数据长度不对	
3424	modbus 从端返回长度过长	
3501	ZCAN 返回无子卡	
3502	ZCAN 返回子卡无对应轴	
4000	4000-4500 PLC 模块的错误	
4002	参数错误	
4003	未知类型	
4004	未知函数	
4005	压栈太多 STL	
4006	压栈太多	
4007	程序太复杂， BLOCK 太多	
4008	没有压栈 BLOCK	
4009	没有压栈 STL	
4010	没有压栈	
4014	文件内容错误	
4015	RET 必须在 STL 的后面	
4016	超过范围	
4017	低于范围	
4018	L 没有定义	
4019	不支持 G 代码函数	
4020	不能 GOTO 跨 PLC 与 BASIC	
4021	PLC 主任务只有一个	
4022	语法错误	
4023	FOR NEXT 错误，不匹配	
4024	FOR NEXT 错误， 无 NEXT	
4026	FOR MC 混用	
4027	FOR STL 混用	
4030	必须 PLC 主任务中使用	
4031	必须中断中使用	
4032	参数个数少	
4033	参数个数多	
4034	要 8 的倍数	

4035	寄存器标识错误	
4036	寄存器类型错误	
4037	LV 个数超过	
4038	只读	
4500	4500-5000 PLC 上位机端错误	
4503	内存不够	
4504	回流到母线上	
4505	回流	
4506	AND 类型不能直接接母线	
4510	悬空	
4511	最右端必须是输出类型	
5000	5000-5500 HMI 模块的错误	
5000	LCD 号错误	
5002	LCD 号冲突	
5003	不支持对象	
5004	内存不够	
5005	控件层次错误	
5006	窗口号超过	
5007	无效窗口号	
5010	对象属性丢失	
5011	输入窗口有多个显示元件	
5012	ACTION 类型错误	
5013	事件过多	
5014	返回上个窗口失败	
5015	不能关闭基本窗口	
5016	字体中找不到对应字符	
5017	必须在 HMI 任务中使用	
5020	控件 ID 冲突	
5021	LCD 号错误	
5022	找不到可用 LCD	
5023	LCD 没有打开	
5024	LCD 无数据	
5025	程序复位	
5026	lcd 已经打开了	
5027	不是网络 LCD	
5028	不支持的压缩方式	

5029	颜色深度不支持	
5030	不支持的数据类型	
5031	设备号错误	
5032	LCD_SEL 不能使用	
5033	设置 REDRAW 不能再 DRAW 阶段	
5034	DRAW 函数只能在 DRAW 阶段	
5035	操作不能再 DRAW 阶段调用	
5036	内部 LCD 分辨率固定	
5037	LCD 分辨率超过	
5038	库文件名错误	
5039	字符过多	
5501-5599	PC 端 PLC 文件编译的错误	
5503	内存不够	
5504	回流到母线上	
5505	回流	
5506	AND 类型指令不能直接接母线	
5510	右边悬空，没有接输出指令	
5511	最右边不是输出类型指令	
5512	最右边不能连接在一起	
5513	输出类型指令必须在最右边	
5514	不支持的指令类型	
5517	寄存器没有值	
5518	DOT 值超过范围	
5519	索引寄存器超过范围	
5520	字符数过多	
5521	寄存器类型错误	
5522	寄存器值错误	
5523	寄存器个数过多	
5524	寄存器个数过少	
5525	STL 使用错误	
5526	RET 使用错误	
5527	重复 RET	
5528	END 或 LBL 的位置错误	
5529	函数不能直接接母线	
5530	出栈没有压栈	
5531	MPP 太多	

5532	寄存器类型使用错误	
5533	ANB 错误，块数不够	
5534	ORB 错误，块数不够	
5535	ANB 错误，输出操作后不能合并	
5536	ORB 错误，输出操作后不能合并	
5537	AND 直接接母线	
5538	OR 直接接母线	
5539	OR 不能在 OUT 指令的后面	
5540	STL 和 MC 不能共用	
5541	MC 不能直接接母线	
5542	_@寄存器要括号	
5543	注释错误	
5544	梯形图列数过多	
5545	输出类型不能直接接母线	
6000	ECAT 模块错误，slot 编号错误	
6001	内部错误， 功能不支持。	
6005	参数错误	
6006	支持的设备类型数超过限制	
6009	操作 node 超过从个数	
6012	资源不够	
6013	从设备反应超时	
6014	缓冲不够	
6015	应答包 wkc 错误	
6016	SDO 应答超长	
6017	SDO 应答错误	
6018	SDO 应答数据长度错误	
6019	WKC 超时	
6020	state 切换超时	
6021	SDO ABORT	
6023	NODE profile 错误	
6024	轴 profile 错误	
6025	轴数超过	
6029	PDO 列表个数超过	
6031	设备个数超过	
6042	设备不支持	
6045	邮箱超时	

6047	数据类型错误	
6049	模块不支持的子模块	
6050	模块子模块数量超过	
6051	模块不认识的子模块	
6208	RTEX 驱动 ID 冲突	
6209	扫描超时，一般为网线接触问题。	
6210	RTEX 初始化失败	
6211	RTEX 扫描结果错误	
20000	PC 端产生错误的偏移	
20002	参数错	
20003	超时	可能是 fifo 缓冲阻塞
20006	操作系统错误	
20007	串口打开失败	
20008	网络打开失败	可能同一台工控机多个网口 ip 和控制器 ip 在同网段
20009	句柄错误	
20010	发送错误	
20011	文件错误	
20012	文件长度错误	
20013	文件名过长	
20014	文件不存在	
20015	ZLB 库文件错误	
20016	文件没有编译	一般 PLC 文件
20020	固件文件不匹配	
20021	不支持的功能	
20030	输入缓冲长度不够	
20100	应答缓冲长度不够	
	30000 以上是 ZAUX 辅助库产生的错误码	辅助库源码直接搜索即可
30000	无应答	
30001	应答错误	
30002	参数错误	
30003	不支持	
30004	文件参数错误	
30005	OS 错误	
30006	单次发送数据超限	

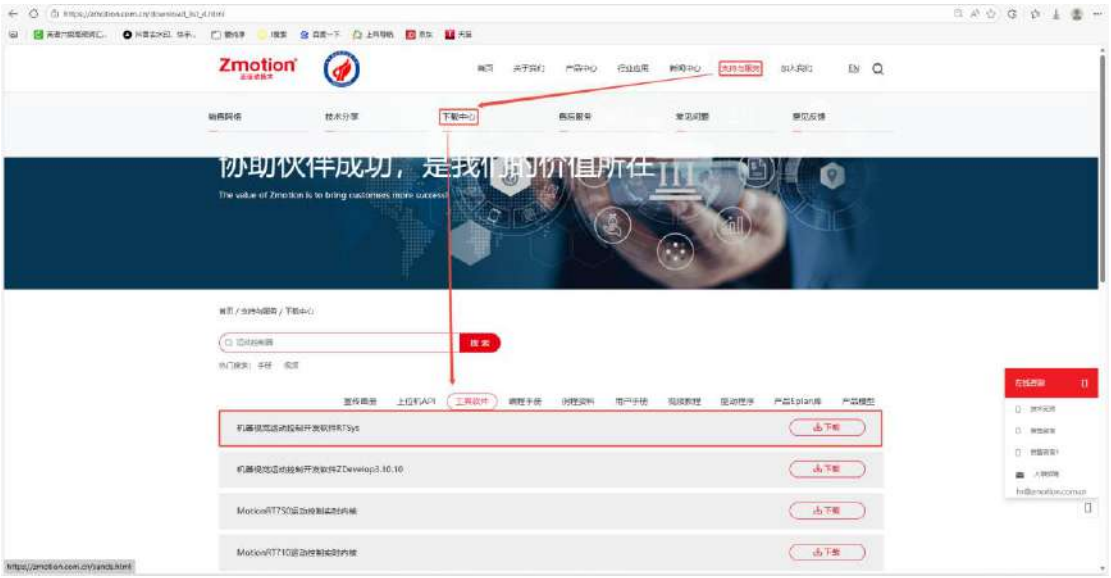
14.3 其他：快速入门/常见问题

14.3.1 快速入门

14.3.1-1 仿真器连接

步骤 1：

打开 RTSys 软件（若无 RTSys 软件，请到正运动官网资料下载：支持与服务->下载中心->工具软件；http://www.zmotion.com.cn/download_list_4.html）



步骤 2：

连接仿真器



步骤 3：

网口连接仿真器，IP 地址为：127.0.0.1（需要使用 RTSys 软件打开仿真器，才可连接仿真器）；

```
// test1.cpp :定义控制台应用程序的入口点。
```

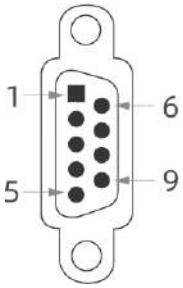
```
//
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s return code is %d\n", command, ret);
    }
}

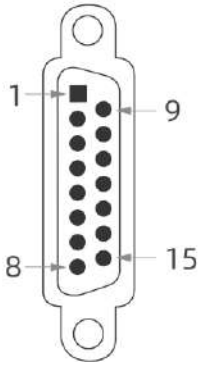
int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";                //控制器 IP 地址,仿真器 IP 地址,
连接需把仿真器打开
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("connect controller failed!\n");
        handle = NULL;
        getchar();
        return -1;
    }
    printf("控制器连接成功! \n");

    //运动类型获取}
    Sleep(2000);
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

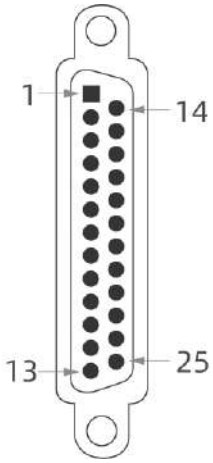
14.3.1-2 脉冲伺服驱动接线



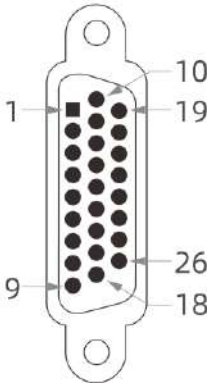
图：9PinDb 头



图：15PinDb 头



图：25PinDb 头



图：26PinDb 头

ZMC316

针脚号	信号
1	PUL+
2	PUL-
3	DIR+
4	DIR-
5	GND
6	ALM
7	SERVON
8	+5V
9	EGND
外部24V+	

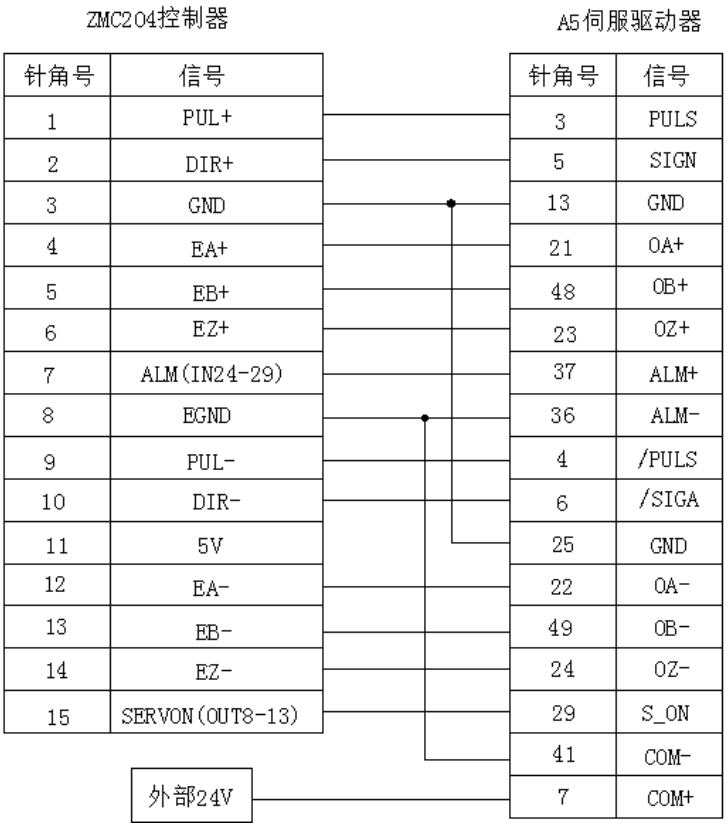
A5伺服驱动器

针脚号	信号
3	PULS
4	/PILS
5	SIGN
6	/SIGN
25	GND
37	ALM+
29	S-ON
36	ALM-
41	COM-
13	GND
21	OA+
48	OB+
23	OZ+
22	OA-
49	OB-
24	OZ-
7	COM+

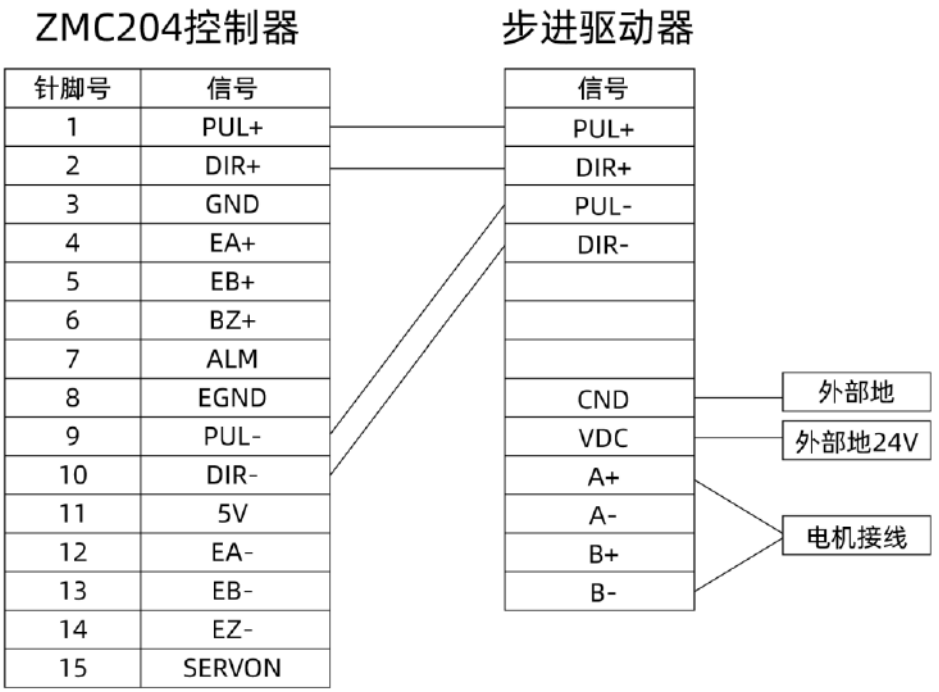
控制器 9PinDB 头与伺服驱动接线图



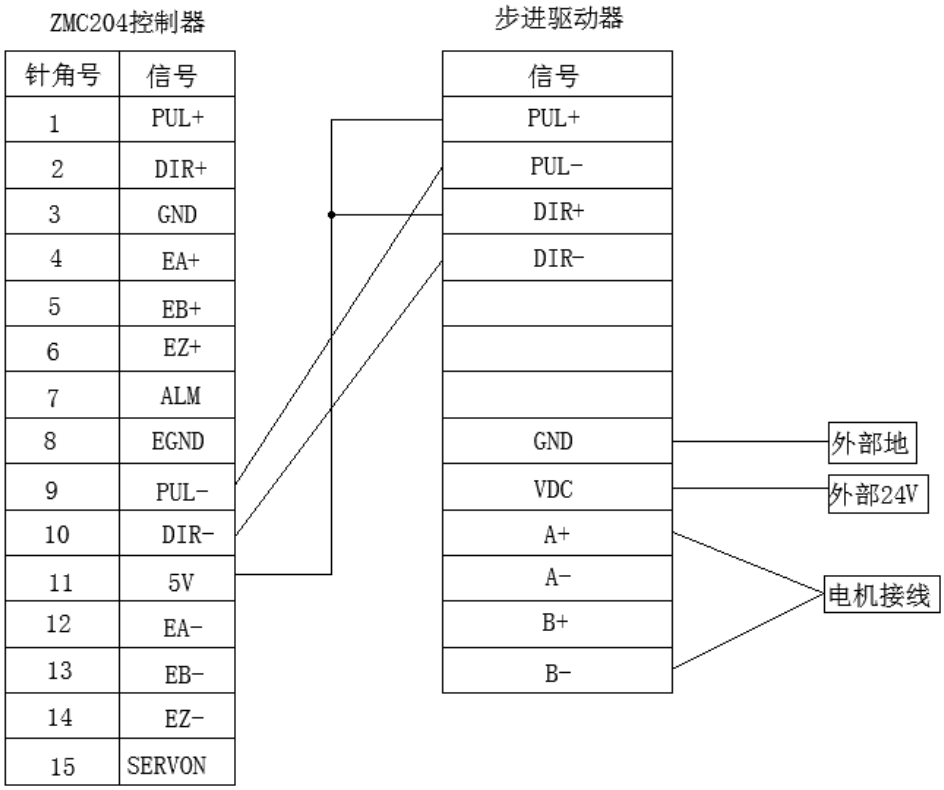
控制器 15PinDB 头与伺服驱动接线图（1）



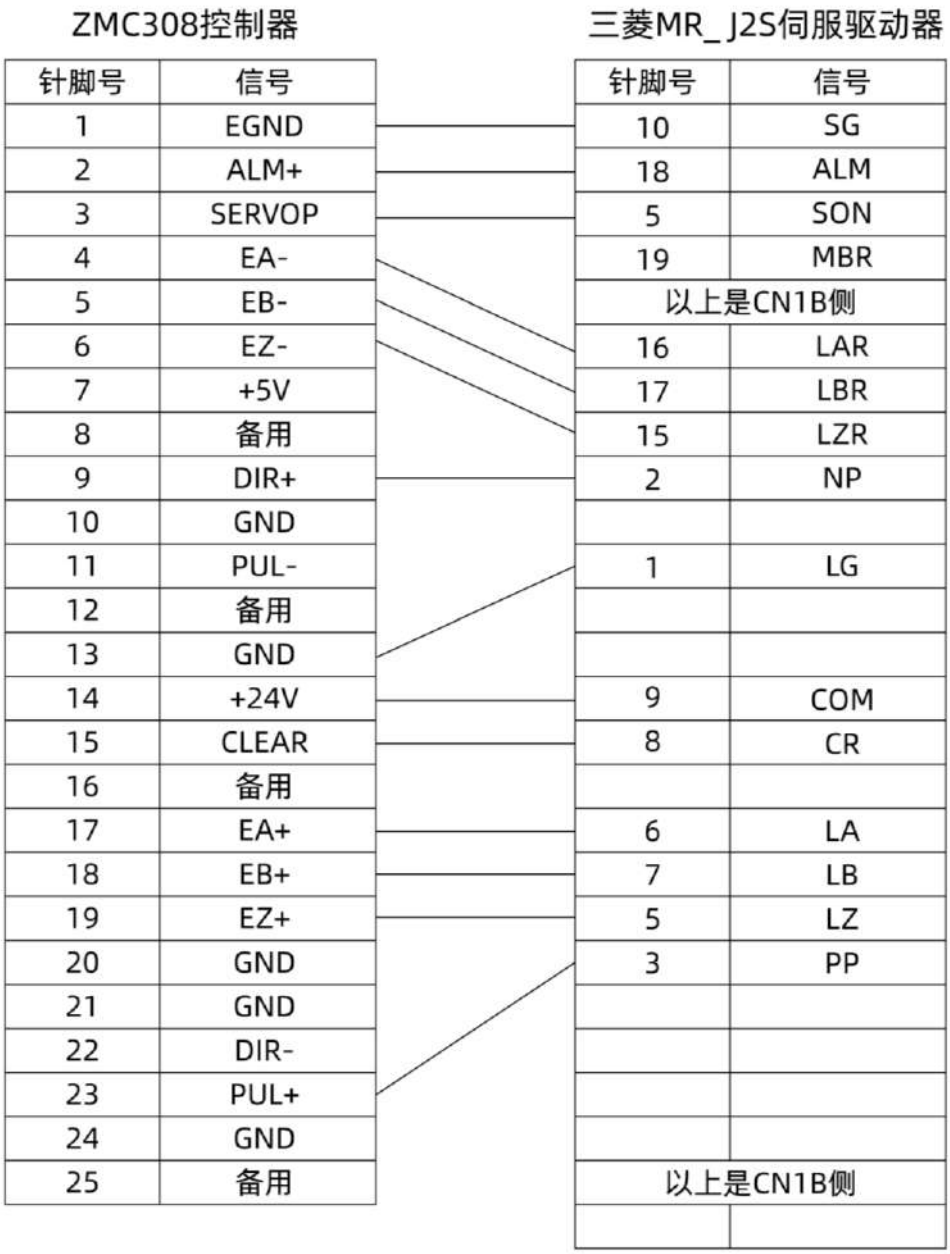
控制器 15PinDB 头与伺服驱动接线图（2）



控制器 15PinDB 头与步进驱动接线图（差分接法）



控制器 15PinDB 头与步进驱动接线图（单端接法）



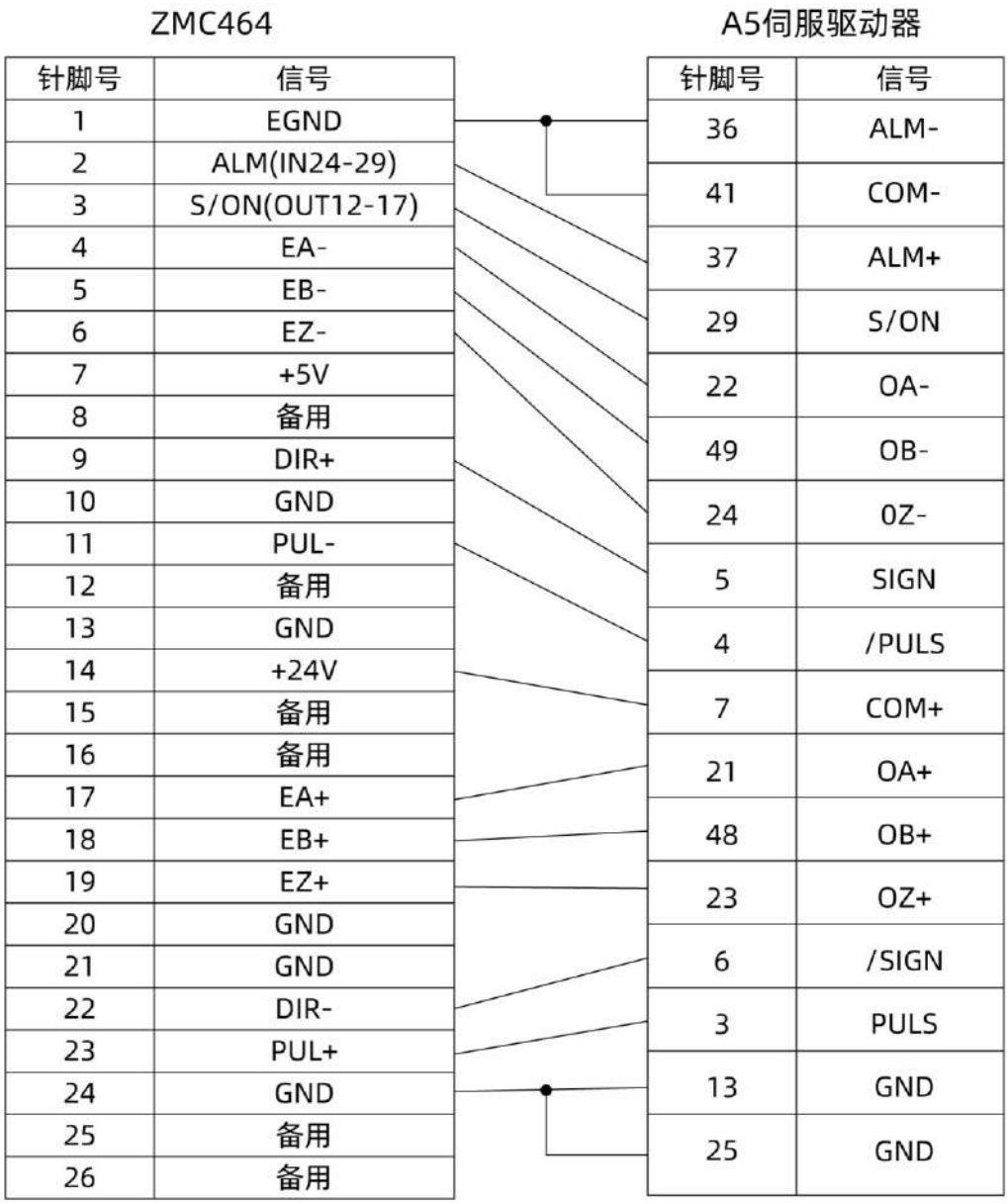
针脚号	信号
1	EGND
2	ALM+
3	SERVOP
4	EA-
5	EB-
6	EZ-
7	+5V
8	备用
9	DIR+
10	GND
11	PUL-
12	备用
13	GND
14	+24V
15	CLEAR
16	备用
17	EA+
18	EB+
19	EZ+
20	GND
21	GND
22	DIR-
23	PUL+
24	GND
25	备用

针脚号	信号
10	SG
18	ALM
5	SON
19	MBR
以上是CN1B侧	
16	LAR
17	LBR
15	LZR
2	NP
1	LG
9	COM
8	CR
6	LA
7	LB
5	LZ
3	PP
以上是CN1B侧	

控制器 25PinDB 头与伺服驱动接线图 1



控制器 25PinDB 头与伺服驱动接线图 2



控制器 26PinDB 头与伺服驱动接线图

14.3.1-3 编码器接线

差分接法

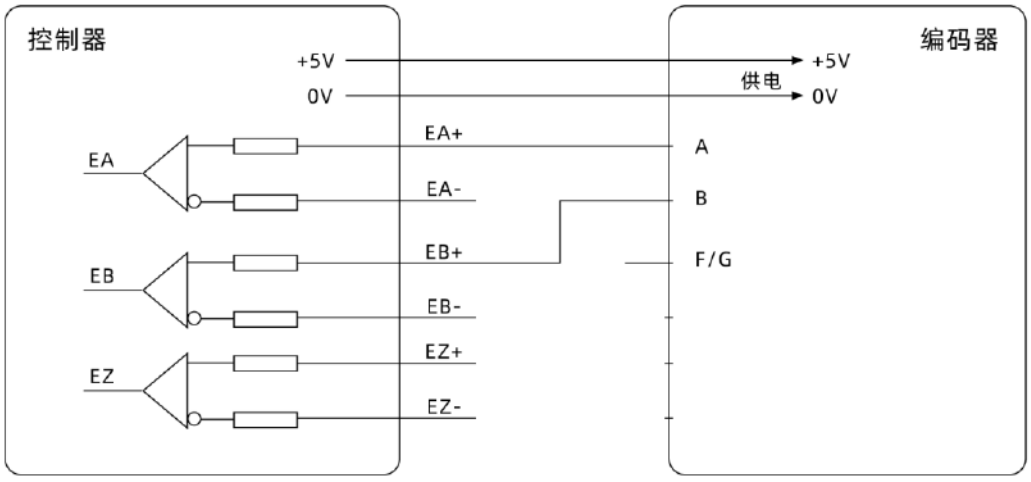
ZMC316公头

针脚号	信号
1	EA+
2	EA-
3	EB+
4	EB-
5	GND
6	EZ+
7	EZ-
8	+5V
9	

A5伺服驱动器

针脚号	信号
21	OA+
22	OA-
48	OB+
49	OB-
13	GND
23	OZ+
24	OZ-
3	PULS
5	SIGN
37	ALM+
36	ALM-
4	/PILS
6	/SIGN
25	GND
29	S-ON
41	COM-
7	COM+

单端接法（请尽量使用差分的编码器）：



14.3.2 常见问题

14.3.2-1 控制器连接不上

控制器的缺省 IP 地址为 192.168.0.11，请把电脑的 IP 地址修改为同一个网段，然后再连接。部分笔记本电脑的无线连接和网口有冲突，此时可以尝试先禁止无线连接。

（1）检查 PC 的 IP 地址，需要在同一网段。

（2）检查控制器 IP 地址，可以用串口连接上去查看。

（3）检查 PC 防火墙设置。

（4）Ping 一下控制器 IP，看是否能 ping 通控制器，若无法 Pin 通，检查物理接口，或者线

若 PC 无法同时连接上 internet 和控制器

方式 1：修改控制器 IP 地址与 PC 和网关位于同一个网段。

方式 2：修改 PC 的子网掩码，使得网关和控制器 IP 地址位于同一个网段，比如改为：255.255.128.0

14.3.2-2 调用运动指令，轴没用动

可能 1：

有限位或告警输入，或限位告警的电平配置错误。

特殊信号缺省是常闭的，国内部分产品是常开的。需要设置 INVERT_IN 把对应的口翻转过来。

可能 2：

轴的 atype 配置错误.

14.3.2-3 IO 扩展模块编号如何计算

根据模块的 ID 来确定，ID 通过拨码开关来定。

对 ZIO1608

ID0 – 输入 16 – 31 输出 16-23

ID1 – 输入 32 – 47 输出 32-39

依次类推。

14.3.2-4 是否所有的指令都可以从 PC 直接发送

WAIT,CAN 等部分指令限制从 PC 发送，因为会造成 PC 命令通道阻赛。
也不能通过 PC 命令来定义变量数组。

14.3.2-5 如何立刻停止运动？

不存在绝对的立刻停止， 通过设置 fastdel 参数为很大值， 可以使得限位或 rapidstop 时可以快速停止。

停止之后再次运动，最好加上 wait idle 的判断

rapidstop 无视 base，停止所有的轴

14.3.2-6 手轮如何实现？

采用 connect 指令来实现， 参见手轮例程。详情可以微信关注正运动小助手，搜索手轮关键字，里面有一篇关于手轮的文章

正运动小助手（学习园地）

14.3.2-7 如何通过 modbus 直接访问一些系统状态？

通过特殊的 modbus 寄存器可以访问 IO，位置等。
位 10000 开始表示输入，20000 开始表示输出。
字寄存器每个轴占两个字，float 格式

14.3.2-8 Cancel 后运动还在执行？

（1）cancel 缺省是取消当前运动，缓冲的运动不取消，带参数 2，cancel(2)后所有的都取消。
（2）检查加减速度有无设置，不存在绝对的立刻停止，通过设置 fastdel 参数为很大值，

可以使得限位或 rapidstop 时可以快速停止，若未设置减速度或者快减减速度，将停不下来。

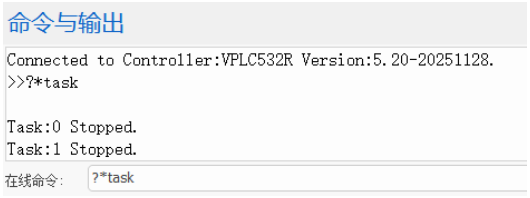
14.3.2-9 需要掉电能保存数据，怎么办？

VR 的数据是掉电保存的，如果其他数据要保存，可以写入 flash，上电时再读取出来，注意 flash 不要频繁读写,有读写次数限制。

14.3.2-10 程序运行中出错，如何查错？

不要重启，用 RTSys 工具连接上去（选择附加到当前程序的方式），即可查看程序运行内部状态和出错位置。

通过命令栏输入： ?*task 也可以参考出错原因和位置。



14.3.2-11 遇到限位的时候冲过去了？

1.提高 FASTDEC 的设置，或是加大限位信号的挡片感应范围。
2.检查正负限位有无设置反

14.3.2-12 设置了连续插补但是没效果？

1.检查 MERGE 是否设置在 MOVESP 指令的第一个 BASE 轴上？或者其他轴参数设置。
2.若运动指令为小线段，检测是否小线段运动时间太短，导致 PC 这边发运动的速度比实际运动速度跑得慢。

14.3.2-13 传感器信号跳动，一个信号受其他信号的影响.

1.检查是否 IO 传感器的电源与控制器的 IO 电源不是同一个，此时需要将两者的底线连接起来。
2.若驱动器 IO 映射，IO 信号跳动，可以检测 IO 映射是否重复

14.3.2-14 回零时本身在原点附近时出现慢速一直反找的现象.

控制器针对伺服类电机可能冲过原点的情况加了特别的保护，此时可能是控制器检测到原点减速停止，但是完全停止后又检测不到原点了，因此误以为轴已经冲过了原点感应范围。解决办法：LSPEED 在回零时设置为 0，让回零产生一定的减速距离。

14.3.2-15 触摸屏上 0X 系统输入状态寄存器为什么没有反转电平.

这个读取的是原始状态，反转指令只对 IN 读取有影响。

14.3.2-16 整圆插补有时不执行.

整圆插补必须使用相对圆弧插补，不能使用绝对圆弧。

14.3.2-17 ZAR 文件下载报错，如何定位问题？

通过最新的 RTSys 软件直接下载 zar 文件，调试窗口会显示出错信息。

14.3.2-18 PC 发送运动不运动的原因？

1.可能控制器 BASIC 程序不断的调用停止指令
2.检查接线
3.查看轴状态有无报错

14.3.2-19 上位机超时 3402

方法 1：硬件环境检查

1. 检查控制器是否启动未 RUN 状态，一般表现为控制器 RUN 灯没有亮起。没有亮起先排查控制器本身问题。
2. 检查网线是否松动接触不良，一般表现为网口 Link 灯和数据灯都没有亮起，可以更换网线尝试。
3. 检查网线信号是否互通，一般没有直观表现，可以通过 ping 检查或者万用表测量每个信号通路情况。
4. 如外接第三方交换机或转接设备，检查是否第三方设备影响，用直连的方式排除影响。
5. 检查控制器本体接口是否有故障，除 RUN 灯状态外，检查 ALM 灯和网口的 Link 灯和数据灯在不插网线时是否异常亮起，或用串口线尝试连接控制器，如有亮起则硬件有损坏，走返修流程，指示灯没有异常或使用串口线尝试连接能够连接可能为其他问题，串口线无法连接表示控制器本体有故障，排查控制器本身问题。

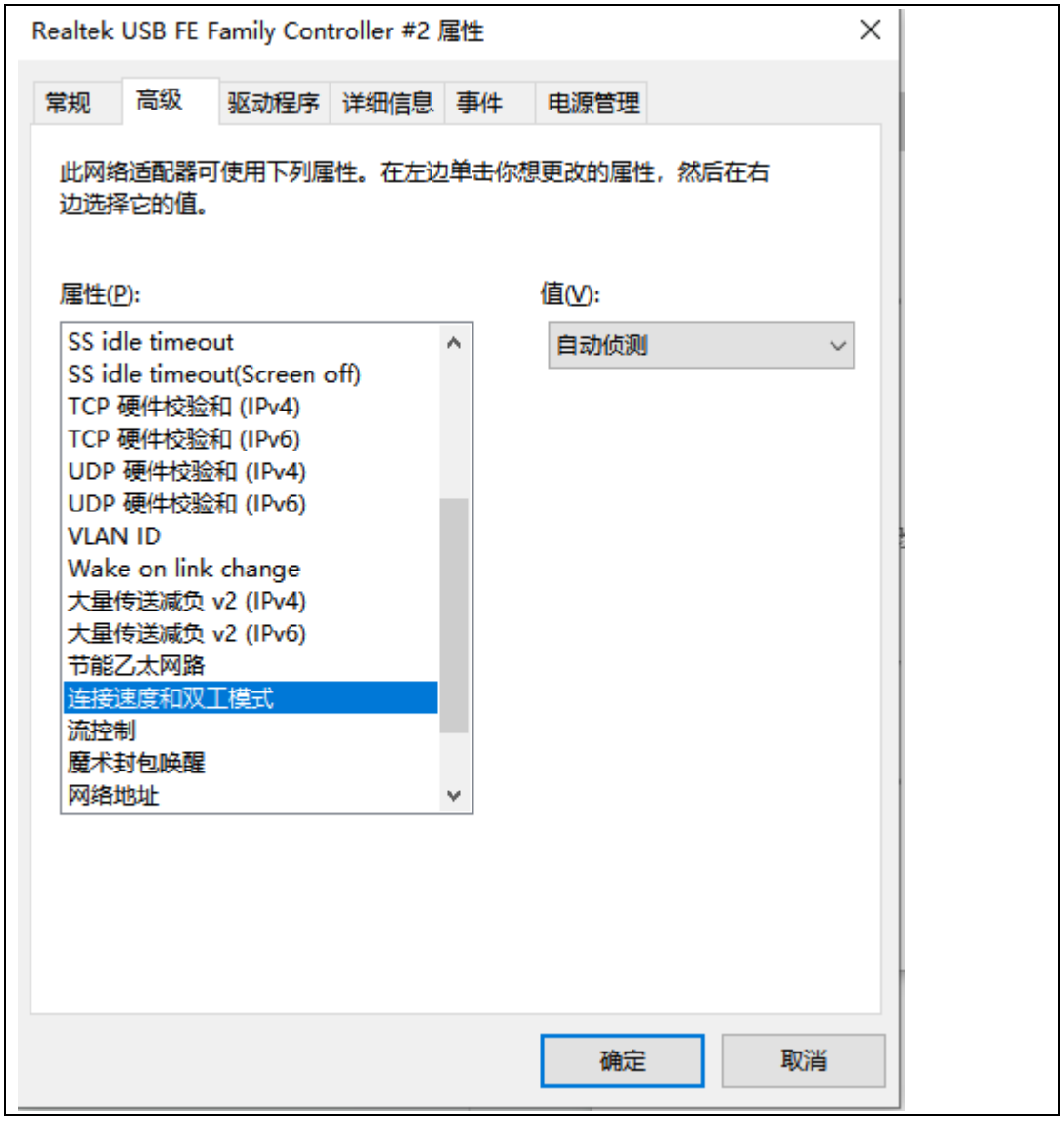
以上基本检查没有问题使用方法 2。

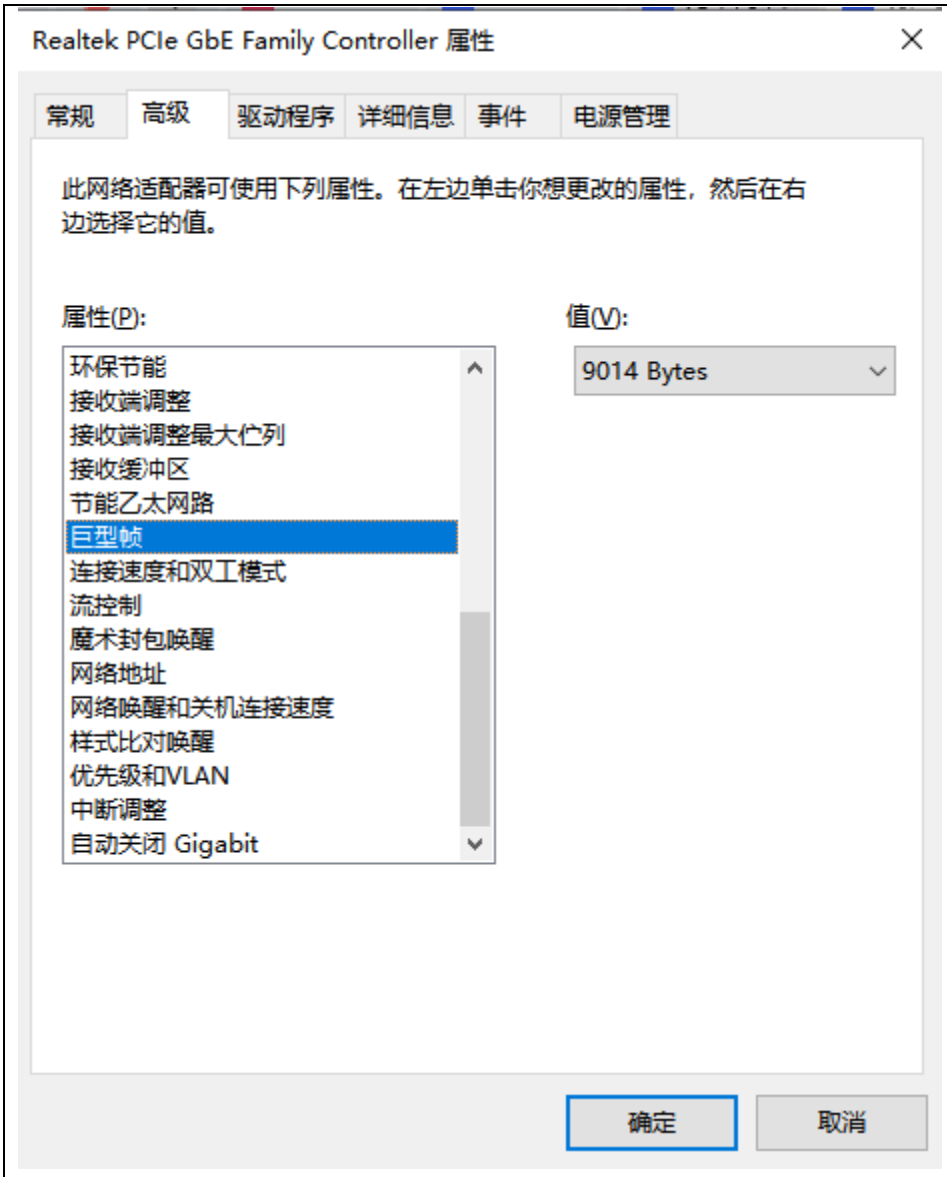
方法 2：客户端网络配置环境检查

1. 检查防火墙是否为关闭状态。不关闭可能出现单向通讯的情况，表现一般为客户端能 Ping 通控制器，但无法连接。如客户一定需要打开防火墙，请在入站规则配置处把正运动相关的软件均加入允许范围。



2. 检查高级网络配置项中的“连接速度和双工模式”“巨型帧配置”
   请选用自动适配选项，没有使用相机请不要打开巨型帧。
   有部分特别新型的网卡兼容性不好的不能适配老旧设备也会有比较特殊的配置，会有相关的字眼，通讯不成功可以尝试改动，没有相关字眼不要修改高级配置。

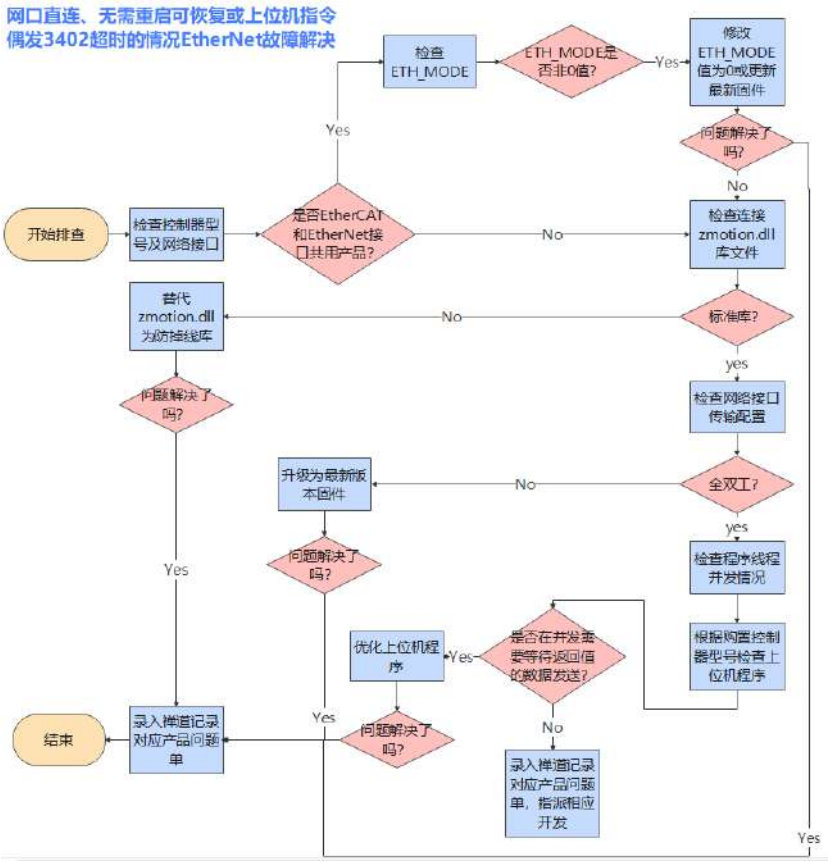

Realtek USB FE Family Controller #2 属性
常规
高级
驱动程序
详细信息
事件
电源管理
此网络适配器可使用下列属性。在左边单击你想更改的属性，然后在右边选择它的值。
属性(P):
值(V):
SS idle timeout
SS idle timeout(Screen off)
TCP 硬件校验和 (IPv4)
TCP 硬件校验和 (IPv6)
UDP 硬件校验和 (IPv4)
UDP 硬件校验和 (IPv6)
VLAN ID
Wake on link change
大量传送减负 v2 (IPv4)
大量传送减负 v2 (IPv6)
节能乙太网路
连接速度和双工模式
流控制
魔术封包唤醒
网络地址
自动侦测
确定
取消



3. 检查同一条链路内是否存在同一个 IP 的设备，排查 IP 冲突引起的故障。
4. 检查其他网卡（包含 Wifi）是否有同网段设备，如有请检查互通性及 IP 冲突的情况，排查 IP 冲突引起的故障。

方法 3：控制器接入链路检查

方法 3-1：不重启可恢复的情况检查



排查流程说明：

ETH_MODE 影响连接稳定性，0 模式为 ETHERNET,1 模式为 ETHERNET 单仅支持单个连接，提高抗干扰和稳定性，2 模式为自动切换，仅在 ETHERNET 和 ETHERCAT 接口共用时使用，其它控制器请不要设置这个模式，推荐使用 0 模式。

Zmotion.dll 选用规则：网卡型号多样兼容性较差，网络环境复杂，干扰防护低、的推荐使用防掉线库，网卡型号稳定、网络环境单一的、干扰防护高的推荐使用标准库，可在新版 RTsys 安装根目录替换，均有区分。

防掉线库使用弊端：兼容性好，在复杂网络和强干扰网络能够稳定性高，但环境过分复杂可能存在大量数据丢失情况，具体调整看应用情况。

标准库使用弊端：数据可靠性好，丢包重传，但在复杂环境可能加剧网络拥堵而更加容易引起掉线情况。

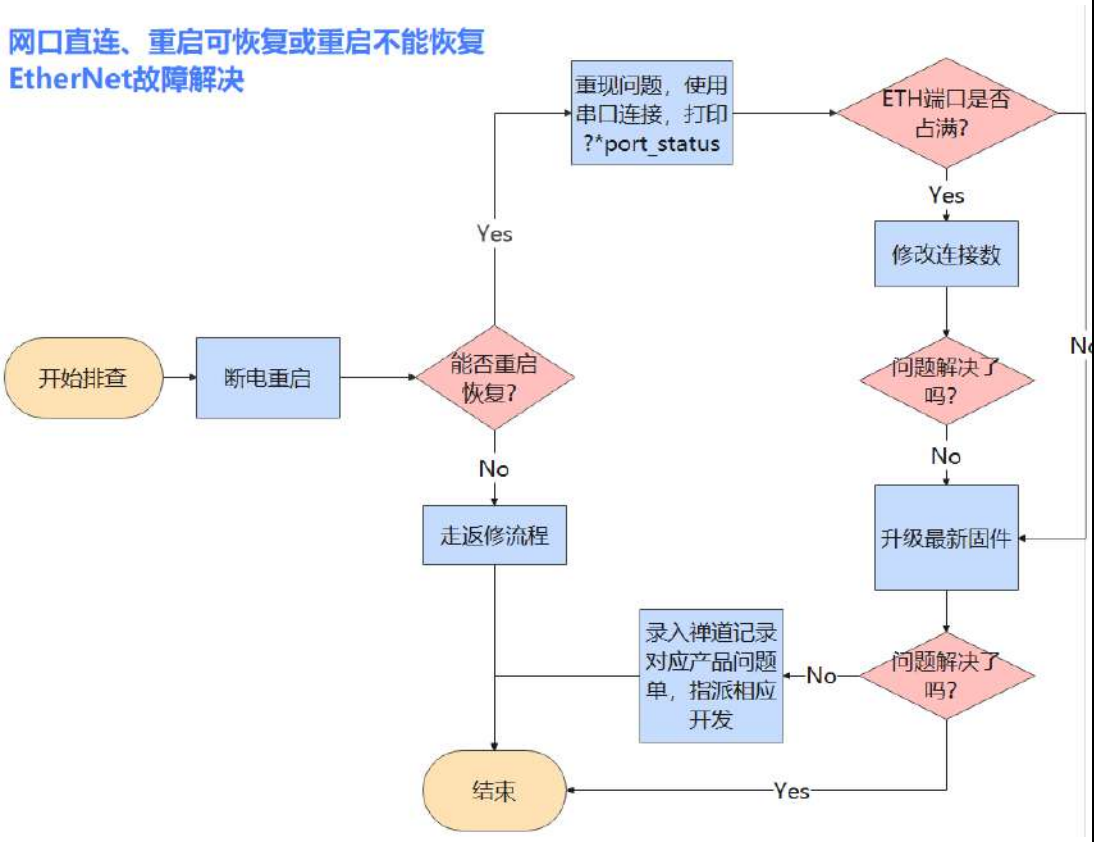
网卡传输配置检查方法，在 Powershell 界面输入以下命令（Win7 不支持）。

Get-NetAdapter | SELECT name, LinkSpeed, fullduplex | ft -autosize



上位机程序检查，主要检查线程并发情况和短时间内发送大量需要返回值的指令情况，较差性能控制器可能会频繁出现 3402 超时指令执行超时或连接超时现象。

方法 3-2：需要重启恢复或不能恢复的情况检查



方法 4：电气环境干扰处理

如怀疑现场干扰问题在前往现场时应该备好可用来实验的配件，验证是否存在干扰及判断干扰源位置。

技术人员备用现场实验设备：

磁环：参考链接，根据实际线横截面积选购

使用位置：

靠近控制器电源输入位置增加
信号线单端或者两端增加
动力线单端或两端增加
编码器线单端或两端增加

作用：降低高频干扰

隔离开关：220 转 24V 小隔离开关，参考链接，根据实际功率选购。

使用位置：

独立使用在控制器电源输入端

作用：一定程度提升电压稳定，抗一定程度雷击浪涌

滤波器：

使用位置：电源输入端

作用：一定程度提升电流稳定性，和消除共模差模干扰

万用表：

使用方法：将黑色表笔可靠地连接到已知的良好接地端，如建筑物的接地母线、接地棒等。这些接地体是作为参考零电位点。把红色表笔连接到要测量电压的地线上，例如设备外壳的接地端或者插座的地线插孔。注：避免用手作为参考地，地线电压过高可能导致触电风险，也不要选用智能万用表， 不当使用可能导致短路现象。

作用：测量地线是否干净，各接地位置电位差和导通性。

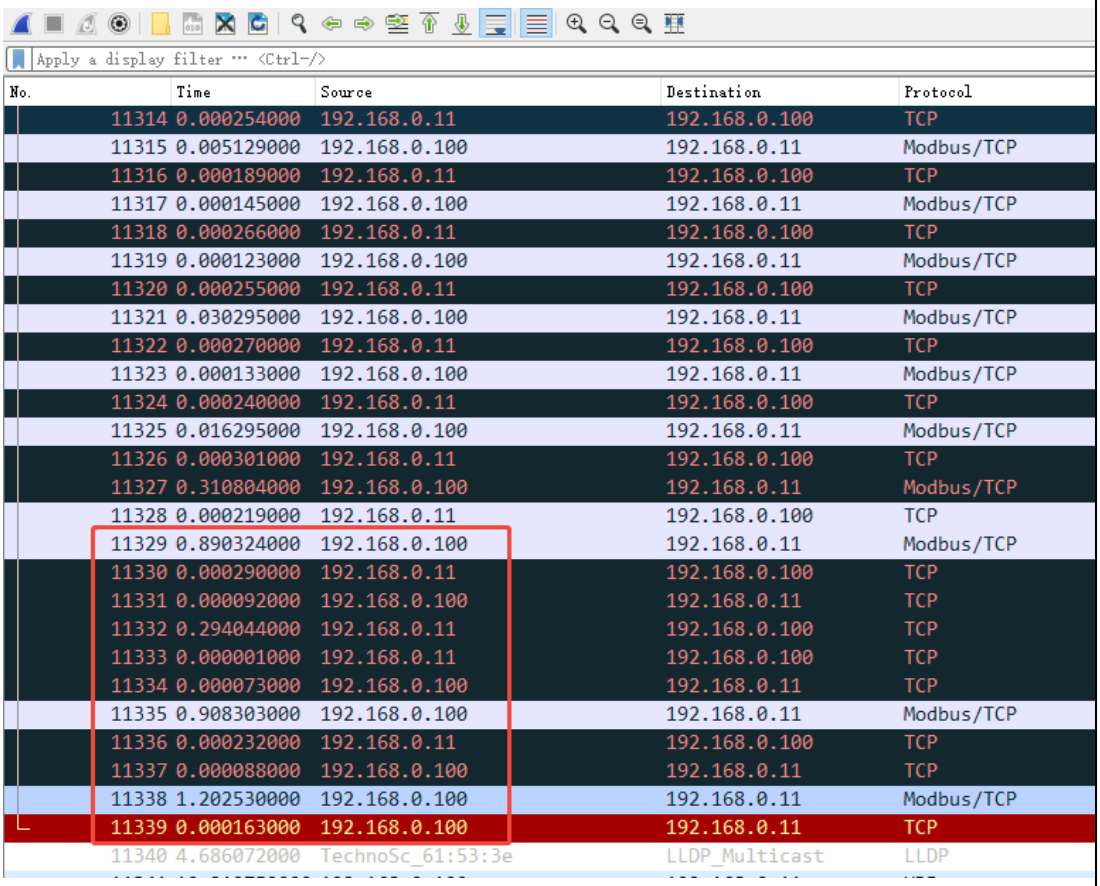
方法 5：Wireshark 抓包分析



PC机按照Wireshark抓包工具，直连控制器抓包即可，
如有交换机可接入交换机环境

TCP 掉线问题分析较简单

分析如图示例：



上图为干扰环境出现了控制器掉线问题，可以观察到包号 11329-11339 有超过 3s 没有数据交互，导致了控制器与 PC 机掉线。

TCP 标准协议超时时间一般为 3s，如果数据中断了 3s 以上，客户端会主动断开连接。

Zmotion PC 编程手册·高级功能篇

第一章 指令列表

注意：本手册中所有蓝色指令(如 ZAux_OpenCom)均带有超链接，点击即可跳转到指定的指令说明。

第二章 轴叠加/电子凸轮	
2.1 运动叠加	
ZAux_Direct_GetAddax	读取叠加轴
ZAux_Direct_Single_Addax	轴叠加运动
2.2 电子凸轮/同步追踪	
ZAux_Direct_Cam	电子凸轮 同步运动
ZAux_Direct_Cambox	跟随凸轮表 同步运动
ZAux_Direct_Movelink	特殊凸轮 同步运动
ZAux_Direct_Moveslink	特殊凸轮 2 同步运动

第三章 高速锁存应用	
3.1 高速锁存	
ZAux_Direct_Regist	位置锁存指令
ZAux_Direct_CycleRegist	连续位置锁存指令
ZAux_Direct_SetCloseWin	设置锁存触发的结束坐标范围点
ZAux_Direct_GetCloseWin	读取锁存触发的结束坐标范围点
ZAux_Direct_GetRegPos	读取返回锁存的测量反馈位置(MPOS)
ZAux_Direct_GetRegPosB	读取返回锁存的测量反馈位置(MPOS)
ZAux_Direct_SetOpenWin	设置锁存触发的开始坐标范围点
ZAux_Direct_GetOpenWin	读取锁存触发的开始坐标范围点
ZAux_Direct_GetMark	读取编码器锁存返回状态
ZAux_Direct_GetMarkB	读取编码器锁存 B 返回状态

第四章 位置比较输出	
4.1 位置比较输出应用	
ZAux_Direct_Pswitch	软件位置比较输出
ZAux_Direct_HwPswitch	硬件位置比较输出
ZAux_Direct_GetHwPswitchBuff	硬件位置比较输出剩余缓冲数
ZAux_Direct_HwPswitch2	总线硬件位置比较输出（脉冲轴和总线轴均支持此指令）

ZAux_Direct_HwPswitch2_2	硬件位置比较输出 2
ZAux_Direct_HwTimer	硬件定时器用于硬件比较输出后一段时间后还原电平
ZAux_Direct_MoveHwPswitch2	硬件位置比较输出，运动缓冲方式
ZAux_Direct_MoveHwTimer	硬件定时器用于硬件比较输出后一段时间后还原电平. ，运动缓冲方式, 4 系列产品支持
ZAux_Direct_SetHwPs2AxisNum	设置 PS2 轴号
ZAux_Direct_GetHwPs2AxisNum	获得 PS2 轴号
ZAux_Direct_GetHwPs2Counts	获得 PS2 比较的点数

第五章 PT 运动	
5.1 PT 运动	
ZAux_Direct_MultiMovePt	多条相对 PT 运动
ZAux_Direct_MultiMovePtAbs	多条绝对 PT 运动
ZAux_Direct_MultiMovePvt	多条相对 PvT 运动
ZAux_Direct_MultiMovePvtAbs	多条绝对 PvT 运动

第六章 机械手控制运动应用	
6.1 机械手模型	
ZAux_Direct_Connframe	机械手逆解指令
ZAux_Direct_Connreframe	机械手正解指令
ZAux_Direct_FrameRotate	机械手坐标系旋转
ZAux_Frame_Rotate	工件坐标系变换
ZAux_Frame_Rotate2	坐标系变换计算
ZAux_World_DPos	世界坐标系
ZAux_Direct_FrameStatus	机械手轴状态
ZAux_Direct_FrameTrans2	正逆解坐标转换
ZAux_Direct_FrameCal	参数矫正
ZAux_Direct_Mover_C	相对 + 关节轴直接走平面圆弧插补运动
ZAux_Direct_Mover_CABS	绝对 + 关节轴直接走平面圆弧插补运动
ZAux_Direct_Mover_C3	相对 + 关节轴直接走空间圆弧插补运动
ZAux_Direct_Mover_C3ABS	绝对 + 关节轴直接走空间圆弧插补运动
ZAux_Direct_Mover_L	相对 + 关节轴直线插补运动
ZAux_Direct_Mover_LAbs	绝对 + 关节轴直线插补运动

第二章 轴叠加/电子凸轮

2.1 运动叠加

2.1.1 指令列表

指令	说明
ZAux_Direct_GetAddax	读取叠加轴
ZAux_Direct_Single_Addax	轴叠加运动

2.1.2 重点说明

运动叠加，把一个轴的运动叠加到另一个轴。

ADDAX 指令叠加的是脉冲个数，而不是设置的脉冲当量。

转换关系：叠加轴运动距离*叠加轴脉冲当量/被叠加轴脉冲当量=被叠加轴运动距离。

假设轴 A 的脉冲当量是 100，轴 B 的脉冲当量是 50，叠加轴运动 100。

把轴 A 的运动叠加到轴 B，此时轴 A 显示运动了 100，轴 B 运动了 100*100/50=200。

把轴 B 的运动叠加到轴 A，此时轴 B 显示运动了 100，轴 A 运动了 100*50/100=50。

轴之间不能相互同时叠加，A 叠加到 B 后，B 不能再叠加到 A。

支持串联叠加，A 运动叠加到 B，B 在叠加到 C。

支持并联叠加，A 运动同时叠加到 B、C。

叠加时速度从被叠加轴开始变化，加减速按照叠加轴加减速及两轴 units 比例确定。

运动叠加，把一个轴的运动叠加到另一个轴。

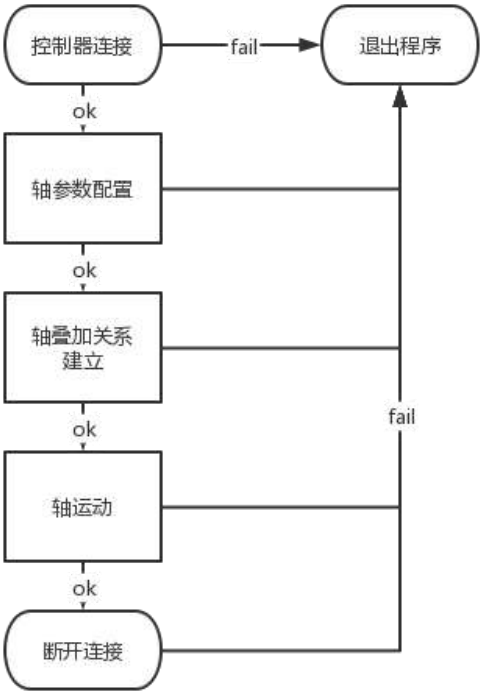
注：ZAux_Direct_Single_Addax 函数，在轴当前状态为正解状态或逆解状态的时候不起作用。

2.1.3 例程

2.1.3-1 运动叠加

轴 0 的脉冲当量是 100，轴 1 的脉冲当量是 50，把轴 0 的运动叠加到轴 1 上，轴 0 运动

100。




```
// test1.cpp : 定义控制台应用程序的入口点。
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非0则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
    }

}

int _tmain(int argc, _TCHAR* argv[])
{
```

```
char *ip_addr = (char *)"127.0.0.1";                //控制器 IP 地址
ZMC_HANDLE handle = NULL;                           //连接句柄
int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
if (ERR_SUCCESS != ret)
{
    printf("控制器连接失败!\n");
    handle = NULL;
    Sleep(2000);
    return -1;
}
printf("控制器连接成功! \n");

ZAux_Direct_SetAtype(handle, 0, 1);   //设置轴 0 轴类型为 1
ZAux_Direct_SetUnits(handle, 0, 100); //设置轴 0 脉冲当量为 100
ZAux_Direct_SetAtype(handle, 1, 1);   //设置轴 1 轴类型为 1
ZAux_Direct_SetUnits(handle, 1, 50); //设置轴 1 脉冲当量为 50

ZAux_Direct_SetSpeed(handle, 0, 200); //设置轴 0 速度为 200units/s
ZAux_Direct_SetAccel(handle, 0, 2000); //设置轴 0 加速度为 2000units/s/s
ZAux_Direct_SetDecel(handle, 0, 2000); //设置轴 0 减速度为 2000units/s/s
ZAux_Direct_SetSramp(handle, 0, 200); //设置轴 0 S 曲线时间为 200ms

ret = ZAux_Direct_SetDpos( handle,  0,  0);//轴指令位置清 0
commandCheckHandler("ZAux_Direct_SetDpos", ret);//判断指令是否执行成功
ret = ZAux_Direct_SetMpos( handle,  0,  0);//编码器反馈位置清 0
commandCheckHandler("ZAux_Direct_SetMpos", ret);//判断指令是否执行成功

ZAux_Direct_Single_Addax(handle,1,0);//轴 0 脉冲叠加到轴 1 上

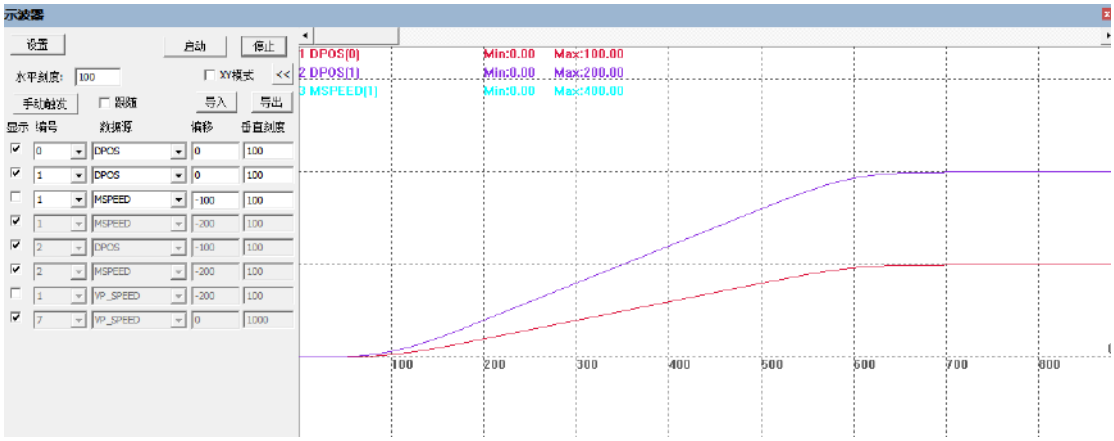
ZAux_Trigger(handle);
ZAux_Direct_Single_Move(handle,0,100);//轴 0 运动 100
float IDLE;
while (1)//等待轴 0 运动完成
{
    Sleep(100);
    ZAux_Direct_GetParam(handle,"IDLE",0,&IDLE);
    if (IDLE<0)break;
```

```
    }
    ZAux_Direct_Single_Addax(handle,1,-1);//取消叠加

    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");

    handle = NULL;
    return 0;
}
```

结果输出波形图：



2.2 电子凸轮同步追踪

2.2.1 指令列表

指令	说明
ZAux_Direct_Cam	电子凸轮（同步运动）
ZAux_Direct_Cambox	跟随凸轮表（同步运动）
ZAux_Direct_Movelink	特殊凸轮（同步运动）
ZAux_Direct_Moveslink	特殊凸轮 2（同步运动）
ZAux_Direct_MoveSync	同步运动，皮带上物体跟随

2.2.2 重点说明

（1）电子凸轮

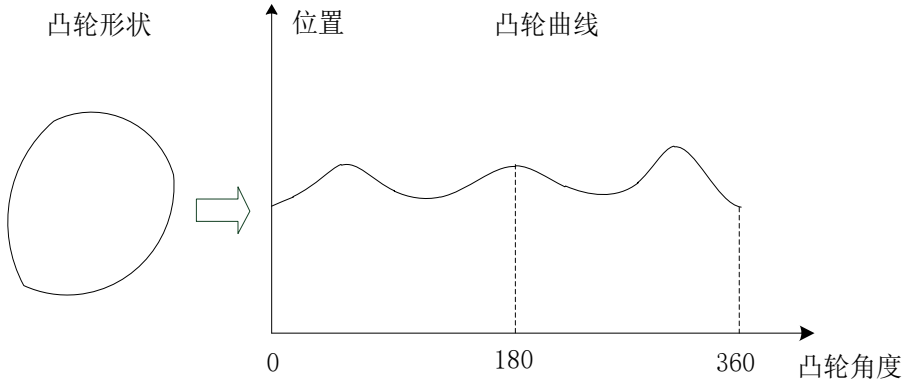
1>组成部分

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

2>工作原理

电子凸轮属于多轴同步运动，这种运动是基于主轴外加一个或多个从轴系统，是在机械凸轮的基础上发展而来，电子凸轮多用于周期性的曲线运动场合。

如下图，机械凸轮按照凸轮的轮廓可以得出一段转动角度与加工位置运动轨迹，此轨迹为弧线，将该段弧线分解成无数个直线轨迹，组合起来得到一串趋近于弧线运动轨迹，电子凸轮直接将此段轨迹运动参数装入运动指令，即可控制轴走出目标轨迹。



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

3>优点

电子凸轮的使用使机器的控制精度提高，控制距离加远，故障率降低，可靠性提高；简化了机构，使机构更加灵活，使调试和维修变得简便。电子凸轮用软件来控制信号，改变程序的相关运动参数就能改变运动曲线，应用灵活性高，工作可靠，操作简单，不需要额外安装机械构件，因而不存在磨损的情况。

（2）同步追踪

同步追踪是一种常见的凸轮应用，其动作与“追剪”类似，都是要求从轴（伺服）的位置在工作时与主轴同步，差别是“追剪”主要用于连续料的定长切割，而“同步追踪”则用于随机料（也就是物品出现的时机不固定），因此启动信号是根据感测器，触发后只追踪一次，设定的距离到达便自动回到起始位置，等下次触发信号收到才会再次追踪，常应用于流水线，例如同步喷漆，瓶罐注料，物品夹取等应用。

（3）自动凸轮

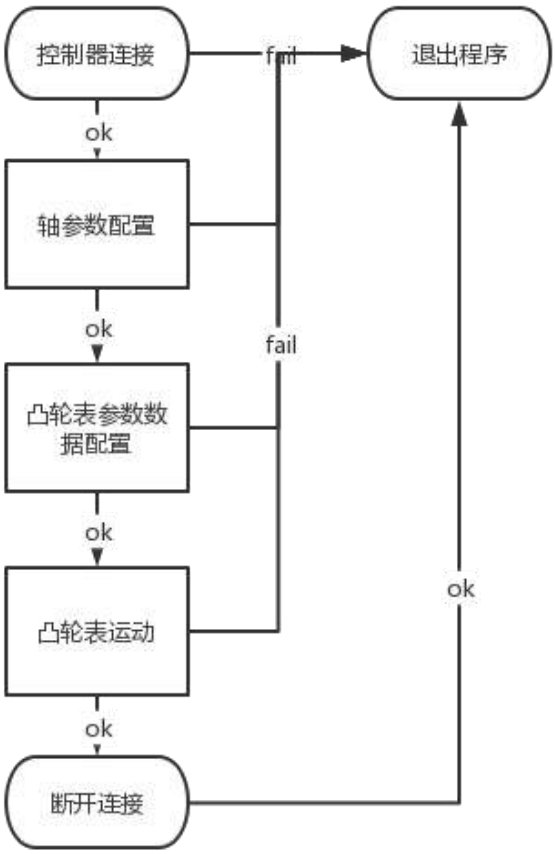
自动凸轮是凸轮运动的一种，主要针对两个轴之间的主从跟随运动，用户可以通过简单设置几个相关参数，便可以构建主轴与从轴之间的运动关系，位置关系不是存储于 TABLE 表中，靠指令参数设置每段跟随距离和变速过程，运动过程中从轴的速度自动计算以匹配主

轴，常见运动过程有跟随加速、减速、同步。

自动凸轮指令有 MOVELINK、MOVESLINK、FLEXLINK 等，常见应用场合有追剪、飞剪、轮切。

2.2.3 例程

2.2.3-1 凸轮表例程




```
// test1.cpp : 定义控制台应用程序的入口点。
//


#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
#include <math.h>
#define PI 3.1415
void commandCheckHandler(const char *command, int ret)
```

```
{
    if (ret)//非 0 则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
        Sleep(2000);
        exit(0);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";              //控制器 IP 地址
    //char *ip_addr = (char *)"192.168.1.11";               //控制器 IP 地址
    ZMC_HANDLE handle = NULL;                   //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);   //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("控制器连接成功! \n");

    ret =ZAux_Direct_SetAtype(handle, 0, 1);       //设置轴 0 轴类型为 1
    commandCheckHandler("ZAux_Direct_SetAtype", ret);//判断指令是否执行成功
    ret =ZAux_Direct_SetUnits(handle, 0, 100); //设置轴 0 脉冲当量为 100
    commandCheckHandler("ZAux_Direct_SetUnits", ret);//判断指令是否执行成功
    ret =ZAux_Direct_SetSpeed(handle, 0, 100); //设置轴 0 速度为 100units/s
    commandCheckHandler("ZAux_Direct_SetSpeed", ret);//判断指令是否执行成功
    ret =ZAux_Direct_SetAccel(handle, 0, 1000); //设置轴 0 加速度为 1000units/s/s
    commandCheckHandler("ZAux_Direct_SetAccel", ret);//判断指令是否执行成功
    ret =ZAux_Direct_SetDecel(handle, 0, 1000); //设置轴 0 减速度为 1000units/s/s
    commandCheckHandler("ZAux_Direct_SetDecel", ret);//判断指令是否执行成功

    ret = ZAux_Direct_SetDpos( handle,  0,  0);//轴指令位置清 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret);//判断指令是否执行成功
```

```
    ret = ZAux_Direct_SetMpos( handle,  0,  0);//编码器反馈位置清 0
    commandCheckHandler("ZAux_Direct_SetMpos", ret);//判断指令是否执行成功

    int TableStartPoint = 10;               //Table 的起始位置
    int TableEndPoint = 10+1024-1;              //Table 的结束位置
    float ftablemulti = 100;   //位置比例
    float ReferDistance = 1000;                 //参考移动的距离
    int i,j;

    float setTableValue[1024];//凸轮表数据

    //正弦曲线数据写入
    for ( j=0;j<4;j++)
    {
        for ( i=0;i<256;i++)
        {
            setTableValue[i+j*256]=(sin(((2*PI)/34)*i))*100;
        }
    }

    ZAux_Direct_SetTable(handle,TableStartPoint,1024,setTableValue);//把凸轮表数据写
入 Table 寄存器值

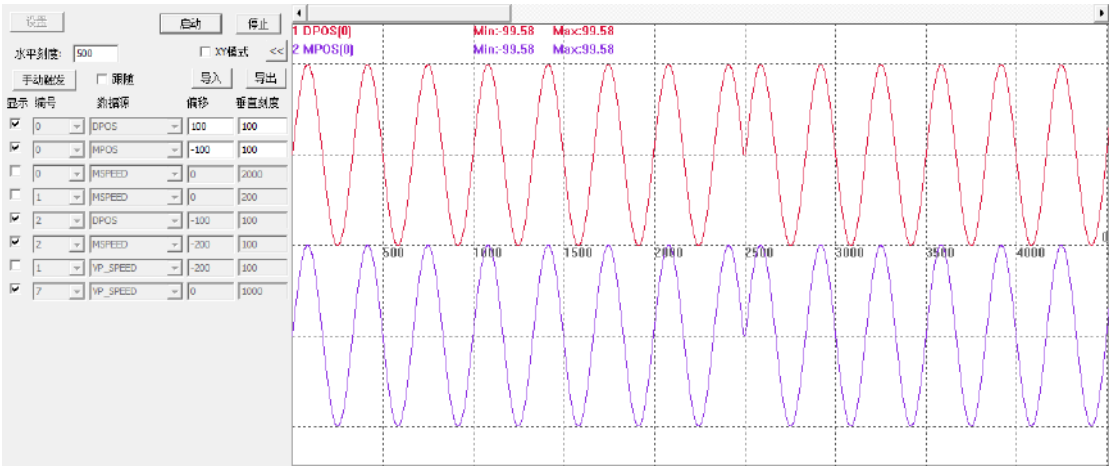
    ZAux_Trigger(handle);
    ret = ZAux_Direct_Cam( handle, 0, TableStartPoint, TableEndPoint, ftablemulti,
ReferDistance); //凸轮表运动
    commandCheckHandler("ZAux_Direct_Cam", ret);
    float IDLE;
    while (1)//等待轴 0 运动完成
    {
        Sleep(100);
        ZAux_Direct_GetParam(handle,"IDLE",0,&IDLE);
        if (IDLE<0)break;
    }

    Sleep(6000);
    ret = ZAux_Close(handle);    //关闭连接
```

```
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");

    handle = NULL;
    return 0;
}
```

输出波形：



2.2.3-2 飞剪应用

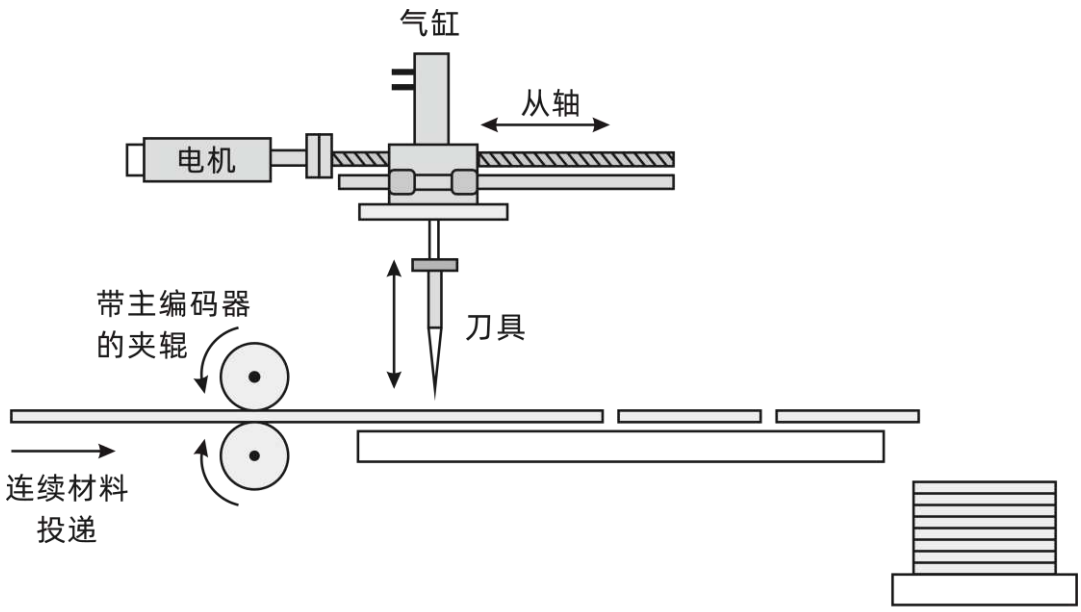
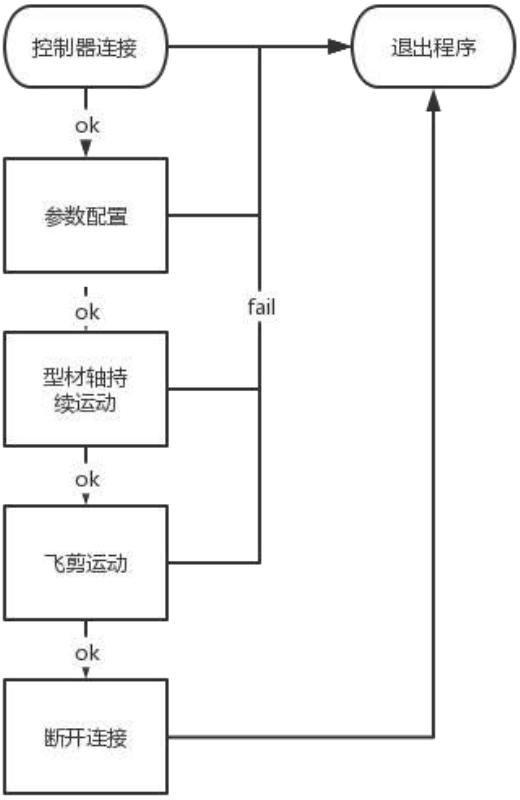


图1 将连续材料切割成定长的飞剪设备

型材持续运动，工作台先静止；直到型材持续运动了某段距离，工作台开始加速；待工作台速度与型材一致，然后开关工作刀具下剪，剪切完后刀具回升；工作台开始减速，然后退回起始点。重复过程，剪切得到设定长度的型材。

假设要切的型材长度为 4m，工作台运行距离 1m，轴 1 为基本轴（型材传送），轴 0 为跟随轴（追剪工作台），OUT0 口控制刀具，飞剪部分程序如下：




```
// test1.cpp : 定义控制台应用程序的入口点。
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
        Sleep(2000);
        exit(0);
    }
}
```

```
int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";              //控制器 IP 地址
    //char *ip_addr = (char *)"192.168.0.11";               //控制器 IP 地址
    ZMC_HANDLE handle = NULL;                   //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);   //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("控制器连接成功! \n");

    int FirstAxis = 0, SecondAxis = 1;                          //定义轴 0，1
    int AType = 1;                                                  //设置轴的类型
    float DposValue = 0;                                            //设置 DPOS 的值
    float SpeedValue = 3;                                       //设置速度的值,型材运行
速度 1m/s,60m/min
    float AccelValue = 10;                                      //设置加速度的值
    float DecelValue = 10;                                      //设置减速度的值
    float UnitsValue = 10000;                                       //设置 units 的值

    ret = ZAux_Direct_SetAtype(handle, FirstAxis, AType);           //设置轴 0 的类型
    commandCheckHandler("ZAux_Direct_SetAtype", ret);
    ret = ZAux_Direct_SetAtype(handle, SecondAxis, AType);          //设置轴 1 的类型
    commandCheckHandler("ZAux_Direct_SetAtype", ret);
    ret = ZAux_Direct_SetSpeed( handle, FirstAxis, SpeedValue);     //设置轴 0 的速度
    commandCheckHandler("ZAux_Direct_SetSpeed", ret);
    ret = ZAux_Direct_SetSpeed(handle, SecondAxis, SpeedValue);     //设置轴 1 的速度
    commandCheckHandler("ZAux_Direct_SetSpeed", ret);
    ret = ZAux_Direct_SetDpos( handle, FirstAxis, DposValue);       //设置轴 0 的 DPOS
    commandCheckHandler("ZAux_Direct_SetDpos", ret);
    ret = ZAux_Direct_SetDpos(handle, SecondAxis, DposValue);       //设置轴 1 的 DPOS
    commandCheckHandler("ZAux_Direct_SetDpos", ret);
    ret = ZAux_Direct_SetUnits(handle, FirstAxis, UnitsValue);      //设置轴 0 的脉冲当
```

```
量
    commandCheckHandler("ZAux_Direct_SetUnits", ret);
    ret = ZAux_Direct_SetUnits(handle, SecondAxis, UnitsValue);   //设置轴 1 的脉冲当
量
    commandCheckHandler("ZAux_Direct_SetUnits", ret);
    ret = ZAux_Direct_SetAccel(handle, FirstAxis, AccelValue);    //设置轴 0 的加速度
    commandCheckHandler("ZAux_Direct_SetAccel", ret);
    ret = ZAux_Direct_SetAccel(handle, SecondAxis, AccelValue);   //设置轴 1 的加速度
    commandCheckHandler("ZAux_Direct_SetAccel", ret);
    ret = ZAux_Direct_SetDecel(handle, FirstAxis, DecelValue);    //设置轴 0 的减速度
    commandCheckHandler("ZAux_Direct_SetDecel", ret);
    ret = ZAux_Direct_SetDecel(handle, SecondAxis, DecelValue);   //设置轴 1 的减速度
    commandCheckHandler("ZAux_Direct_SetDecel", ret);
    ret = ZAux_Direct_SetSramp(handle, FirstAxis, 400);  //设置轴 0 的 Sramp
    commandCheckHandler("ZAux_Direct_SetDecel", ret);
    ret = ZAux_Direct_SetSramp(handle, SecondAxis, 400); //设置轴 1 的 Sramp
    commandCheckHandler("ZAux_Direct_SetDecel", ret);

    ret = ZAux_Trigger( handle);                                  //触发示波器
    commandCheckHandler("ZAux_Direct_SetDecel", ret);

    ret = ZAux_Direct_Single_Vmove(handle, SecondAxis, 1);//型材持续运动
    commandCheckHandler("ZAux_Direct_Single_Vmove", ret);

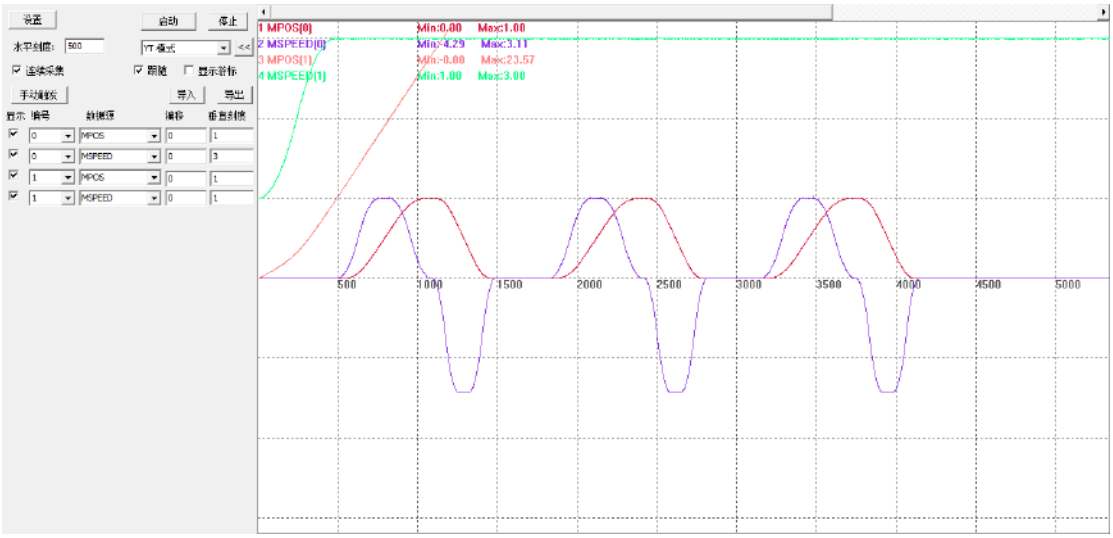
    int i;
    for (i=0;i<3;i++)//飞剪三次
    {
        ret = ZAux_Direct_Movelink( handle, FirstAxis, 0 , 1 , 0 , 0 , SecondAxis,8,
0);//型材运动 1m 前，工作台静止
        commandCheckHandler("ZAux_Direct_Movelink1", ret);
        ret = ZAux_Direct_Movelink( handle, FirstAxis, 0.4 , 0.8 , 0.8 , 0 ,
SecondAxis, 8, 0);//工作台加速阶段：型材运动 0.8m，工作台加速运动 0.4m
        commandCheckHandler("ZAux_Direct_Movelink2", ret);
        ret = ZAux_Direct_Movelink( handle, FirstAxis, 0.2 , 0.2 , 0 , 0 ,
SecondAxis, 8, 0);//同步跟随 0.2m
        commandCheckHandler("ZAux_Direct_Movelink3", ret);
        ZAux_Direct_MoveOp2(handle, FirstAxis,0,1,1000);//'操作 OP 扣，使刀具下剪，1s
```

```
后回升(时间要计算好)
        commandCheckHandler("ZAux_Direct_MoveOp2", ret);
        ret = ZAux_Direct_Movelink( handle, FirstAxis, 0.4 , 0.8 , 0 , 0.8,
SecondAxis, 8, 0);//工作台减速阶段：型材运动 0.8m，工作台减速运动 0.4m
        commandCheckHandler("ZAux_Direct_Movelink2", ret);
        ret = ZAux_Direct_Movelink( handle, FirstAxis, -1 , 1.2 , 0.5 , 0.5,
SecondAxis, 8, 0);//工作台回到起始点：型材运动 1.2m，工作台反向运动 1m
        commandCheckHandler("ZAux_Direct_Movelink2", ret);
    }

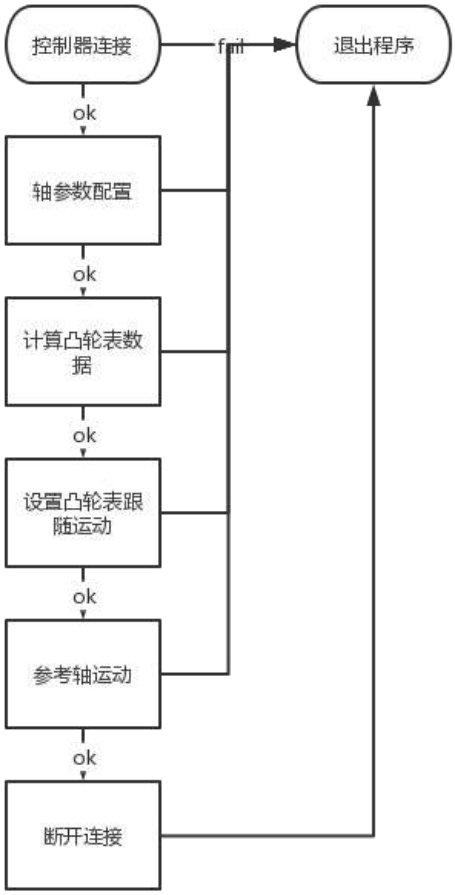
    Sleep(5000);
    ret = ZAux_Close(handle);     //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");

    handle = NULL;
    return 0;
}
```

波形图：



2.2.3-3 电子凸轮同步运动（跟随凸轮表）




```
// test1.cpp : 定义控制台应用程序的入口点。
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
#include <math.h>
void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非0则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
        Sleep(2000);
        exit(0);
    }
```

```
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";              //控制器 IP 地址
    //char *ip_addr = (char *)"192.168.0.11";               //控制器 IP 地址
    ZMC_HANDLE handle = NULL;                   //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);   //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("控制器连接成功! \n");

    int firstAxis = 0, secondAxis = 1;      //定义轴 0、1
    int AType = 1;                          //设置轴的类型
    int DposValue = 0;                      //设置 DPOS 的值
    int UnitValue = 100;                    //设置脉冲当量的值
    int SpeedValue = 200;                   //设置速度的值
    int AccelValue = 2000;                  //设置加速度的值
    int DecelValue = 2000;                  //设置减速度的值

    ret = ZAux_Direct_SetAtype(handle, firstAxis, AType);     //设置轴 0 的类型
    commandCheckHandler("ZAux_Direct_SetAtype", ret);
    ret = ZAux_Direct_SetAtype(handle, secondAxis, AType);    //设置轴 1 的类型
    commandCheckHandler("ZAux_Direct_SetAtype",ret);
    ret = ZAux_Direct_SetDpos(handle, firstAxis, DposValue); //设置轴 0 的 DPOS 的值
    commandCheckHandler("ZAux_Direct_SetDpos", ret);
    ret = ZAux_Direct_SetDpos(handle, secondAxis, DposValue);//设置轴 1 的 DPOS 的值
    commandCheckHandler("ZAux_Direct_SetDpos",ret);
    ZAux_Direct_SetUnits(handle, firstAxis, UnitValue);       //设置轴 0 的脉冲当量
    commandCheckHandler("ZAux_Direct_SetUnits", ret);
```

```
ZAux_Direct_SetUnits(handle, secondAxis, UnitValue);      //设置轴 1 的脉冲当量
commandCheckHandler("ZAux_Direct_SetUnits", ret);
ZAux_Direct_SetSpeed(handle, firstAxis,SpeedValue);       //设置轴 0 的速度的值
commandCheckHandler("ZAux_Direct_SetSpeed", ret);
ZAux_Direct_SetSpeed(handle, secondAxis, SpeedValue);     //设置轴 1 的速度的值
commandCheckHandler("ZAux_Direct_SetSpeed", ret);
ZAux_Direct_SetAccel(handle, firstAxis, AccelValue);      //设置轴 0 的加速度的值
commandCheckHandler("ZAux_Direct_SetAccel", ret);
ZAux_Direct_SetAccel(handle, secondAxis, AccelValue);     //设置轴 1 的加速度的值
commandCheckHandler("ZAux_Direct_SetAccel", ret);
ZAux_Direct_SetDecel(handle, firstAxis, DecelValue);      //设置轴 0 的减速度的值
commandCheckHandler("ZAux_Direct_SetDecel", ret);
ZAux_Direct_SetDecel(handle, secondAxis, DecelValue);     //设置轴 1 的减速度的值
commandCheckHandler("ZAux_Direct_SetDecel", ret);

//{计算 Table 的数据
float deg;
float rad;
float x;
int stepdeg = 2;
float PI = 3.1416;

uint32 IN_Status; //接受输入口的状态值
int piValue;      //读取轴 1 的运动状态
int i,j;
int TableStartPoint = 10;               //Table 的起始位置
int TableEndPoint = 10+1024-1;              //Table 的结束位置
float setTableValue[1024];//凸轮表数据

//正弦曲线数据写入
for ( j=0;j<4;j++)
{
    for ( i=0;i<256;i++)
    {
        setTableValue[i+j*256]=(sin(((2*PI)/34)*i))*100;
    }
}
```

```
    ZAux_Direct_SetTable(handle,TableStartPoint,1024,setTableValue);//把凸轮表数据写入 Table 寄存器值
    //计算 Table 的数据}

    ret = ZAux_Trigger(handle);
    commandCheckHandler("ZAux_Trigger", ret);

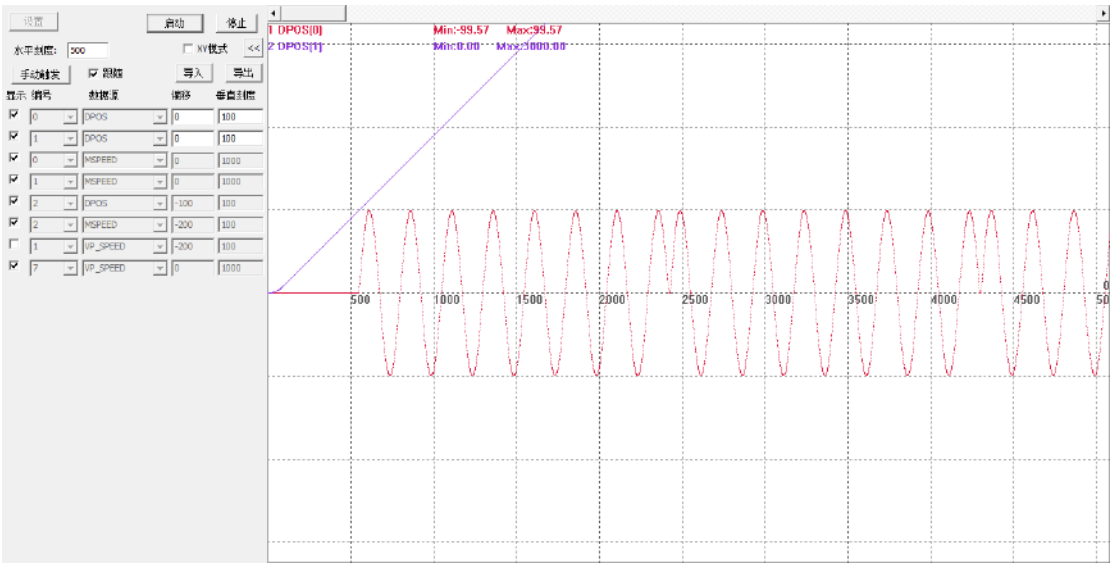
    ret = ZAux_Direct_Cambox(handle, firstAxis, TableStartPoint,
TableEndPoint,100,1500,1,2,100);//参考轴 1 运动到 100 位置时，跟随轴 0 启动凸轮表运动

    ret = ZAux_Direct_Single_Move(handle, secondAxis, 3000);
    commandCheckHandler("ZAux_Direct_Single_Move", ret);

    Sleep(5000);
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");

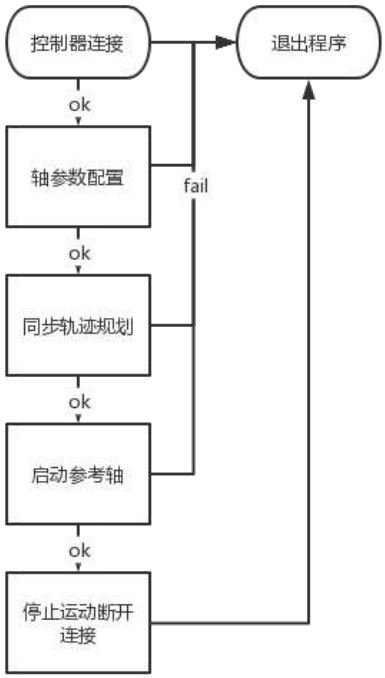
    handle = NULL;
    return 0;
}
```

波形图：



2.2.3-4 电子凸轮同步运动（moveslink）

程序逻辑框图：




```
// test1.cpp : 定义控制台应用程序的入口点。
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
#include <math.h>
void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
        Sleep(2000);
        exit(0);
    }

}
```

```
int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";              //控制器 IP 地址
    //char *ip_addr = (char *)"192.168.0.11";               //控制器 IP 地址
    ZMC_HANDLE handle = NULL;                   //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);   //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("控制器连接成功! \n");

    int FirstAxis = 0, SecondAxis = 1;                              //定义轴 0，1
    int AType = 1;                                                  //设置轴的类型
    float DposValue = 0;                                            //设置 DPOS 的值
    float SpeedValue = 100;                                         //设置速度的值
    float AccelValue = 2000;                                        //设置加速度的值
    float DecelValue = 2000;                                        //设置减速度的值
    float UnitsValue = 100;                                         //设置 units 的值

    char psRespons[1024];                                           //用于接受控制器型
号的数组
    int length = 1024;                                              //控制器型号的长
度，一定要大于接受的字符串长度
    float DistanceValue = 150;

    float fDistance = 50;
    float fLinkDis = 100;
    float startsp = 0;
    float endsp = 1;
    int iLinkaxis = SecondAxis;
    int ioption = 2;
    float flinkstartpos = 50;
```

```
    ret = ZAux_Direct_SetAtype(handle, FirstAxis, AType);          //设置轴 0 的类型
    commandCheckHandler("ZAux_Direct_SetAtype", ret);
    ret = ZAux_Direct_SetAtype(handle, SecondAxis, AType);         //设置轴 1 的类型
    commandCheckHandler("ZAux_Direct_SetAtype", ret);
    ret = ZAux_Direct_SetDpos( handle,  FirstAxis, DposValue);     //设置轴 0 的 DPOS
的值
    commandCheckHandler("ZAux_Direct_SetDpos", ret);
    ret = ZAux_Direct_SetDpos(handle, SecondAxis, DposValue);      //设置轴 1 的 DPOS
的值
    commandCheckHandler("ZAux_Direct_SetDpos", ret);
    ret = ZAux_Direct_SetUnits(handle, FirstAxis, UnitsValue);     //设置轴 0 的脉冲当
量
    commandCheckHandler("ZAux_Direct_SetUnits", ret);
    ret = ZAux_Direct_SetUnits(handle, SecondAxis, UnitsValue);    //设置轴 1 的脉冲当
量
    commandCheckHandler("ZAux_Direct_SetUnits", ret);
    ret = ZAux_Direct_SetAccel(handle, FirstAxis, AccelValue);     //设置轴 0 的加速度
    commandCheckHandler("ZAux_Direct_SetAccel", ret);
    ret = ZAux_Direct_SetAccel(handle, SecondAxis, AccelValue);    //设置轴 1 的加速度
    commandCheckHandler("ZAux_Direct_SetAccel", ret);
    ret = ZAux_Direct_SetDecel(handle, FirstAxis, DecelValue);     //设置轴 0 的减速度
    commandCheckHandler("ZAux_Direct_SetDecel", ret);
    ret = ZAux_Direct_SetDecel(handle, SecondAxis, DecelValue);    //设置轴 1 的减速度
    commandCheckHandler("ZAux_Direct_SetDecel", ret);

    ret = ZAux_Trigger(handle);                                    //触发示波器
    commandCheckHandler("ZAux_Trigger", ret);

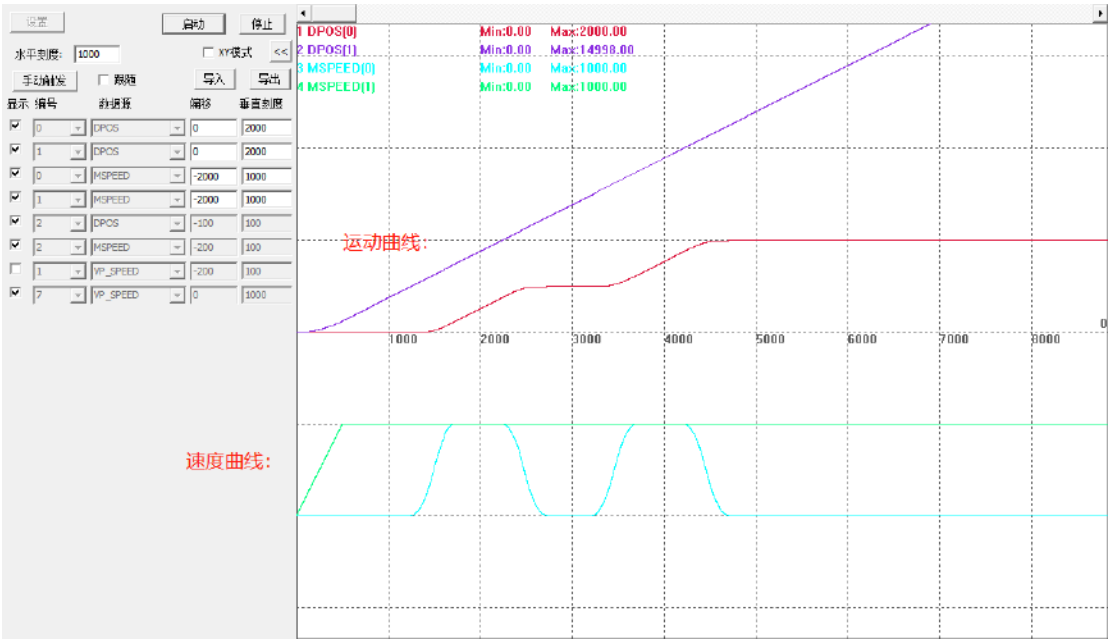
    ret = ZAux_Direct_Moveslink(handle, 0, 250, 500, 0, 1, 1, 2, 1000);//轴 1 位置
1000-1500 时，轴 0 加速与轴 1 同步，加速过程运动共 250
    commandCheckHandler("ZAux_Direct_Moveslink", ret);
    ret = ZAux_Direct_Moveslink(handle, 0, 500, 500, 1, 1, 1, 2, 1500);//轴 1 位置
1500-2000 时，轴 0 与轴 1 同步运动 500
    commandCheckHandler("ZAux_Direct_Moveslink", ret);
    ret = ZAux_Direct_Moveslink(handle, 0, 250, 500, 1, 0, 1, 2, 2000);//轴 1 位置
2000-2500 时，轴 0 与轴 1 减速为 0，减速过程共运动 250
    commandCheckHandler("ZAux_Direct_Moveslink", ret);
```

```
    ret = ZAux_Direct_Moveslink(handle, 0, 250, 500, 0, 1, 1, 2, 3000);//轴 1 位置
3000-3500 时，轴 0 加速与轴 1 同步，加速过程运动共 250
    commandCheckHandler("ZAux_Direct_Moveslink", ret);
    ret = ZAux_Direct_Moveslink(handle, 0, 500, 500, 1, 1, 1, 2, 3500);//轴 1 位置
3500-4000 时，轴 0 与轴 1 同步运动 500
    commandCheckHandler("ZAux_Direct_Moveslink", ret);
    ret = ZAux_Direct_Moveslink(handle, 0, 250, 500, 1, 0, 1, 2, 4000);//轴 1 位置
4000-4500 时，轴 0 与轴 1 减速为 0，减速过程共运动 250
    commandCheckHandler("ZAux_Direct_Moveslink", ret);

    ZAux_Direct_Single_Vmove(handle, 1,1);//启动轴 1
    Sleep(15000);
    ZAux_Direct_Single_Cancel(handle, 1,2);//停止轴 1
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");

    handle = NULL;
    return 0;
}
```

运动波形图：



2.2.3-5 皮带同步跟随运动（MOVESYNC）

```
// test1.cpp : 定义控制台应用程序的入口点。
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
#include <math.h>
void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
        Sleep(2000);
        exit(0);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";              //控制器 IP 地址
    //char *ip_addr = (char *)"192.168.0.11";               //控制器 IP 地址
    ZMC_HANDLE handle = NULL;                   //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);   //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("控制器连接成功! \n");

    int FirstAxis = 0, SecondAxis = 1;                          //定义轴 0，1
```

```
    int AType = 1;                                          //设置轴的类型
    float DposValue = 0;                                    //设置 DPOS 的值
    float SpeedValue = 100;                                 //设置速度的值
    float AccelValue = 2000;                                //设置加速度的值
    float DecelValue = 2000;                                //设置减速度的值
    float UnitsValue = 100;                                 //设置 units 的值

    ret = ZAux_Direct_SetAtype(handle, FirstAxis, AType);   //设置轴 0 的类型
    commandCheckHandler("ZAux_Direct_SetAtype", ret);
    ret = ZAux_Direct_SetAtype(handle, SecondAxis, AType);  //设置轴 1 的类型
    commandCheckHandler("ZAux_Direct_SetAtype", ret);
    ret = ZAux_Direct_SetDpos( handle,  FirstAxis, DposValue);  //设置轴 0 的 DPOS
的值
    commandCheckHandler("ZAux_Direct_SetDpos", ret);
    ret = ZAux_Direct_SetDpos(handle, SecondAxis, DposValue);   //设置轴 1 的 DPOS
的值
    commandCheckHandler("ZAux_Direct_SetDpos", ret);
    ret = ZAux_Direct_SetUnits(handle, FirstAxis, UnitsValue);  //设置轴 0 的脉冲当
量
    commandCheckHandler("ZAux_Direct_SetUnits", ret);
    ret = ZAux_Direct_SetUnits(handle, SecondAxis, UnitsValue); //设置轴 1 的脉冲当
量
    commandCheckHandler("ZAux_Direct_SetUnits", ret);
    ret = ZAux_Direct_SetAccel(handle, FirstAxis, AccelValue);  //设置轴 0 的加速度
    commandCheckHandler("ZAux_Direct_SetAccel", ret);
    ret = ZAux_Direct_SetAccel(handle, SecondAxis, AccelValue); //设置轴 1 的加速度
    commandCheckHandler("ZAux_Direct_SetAccel", ret);
    ret = ZAux_Direct_SetDecel(handle, FirstAxis, DecelValue);  //设置轴 0 的减速度
    commandCheckHandler("ZAux_Direct_SetDecel", ret);
    ret = ZAux_Direct_SetDecel(handle, SecondAxis, DecelValue); //设置轴 1 的减速度
    commandCheckHandler("ZAux_Direct_SetDecel", ret);

    ret = ZAux_Trigger(handle);                             //触发示波器
    commandCheckHandler("ZAux_Trigger", ret);

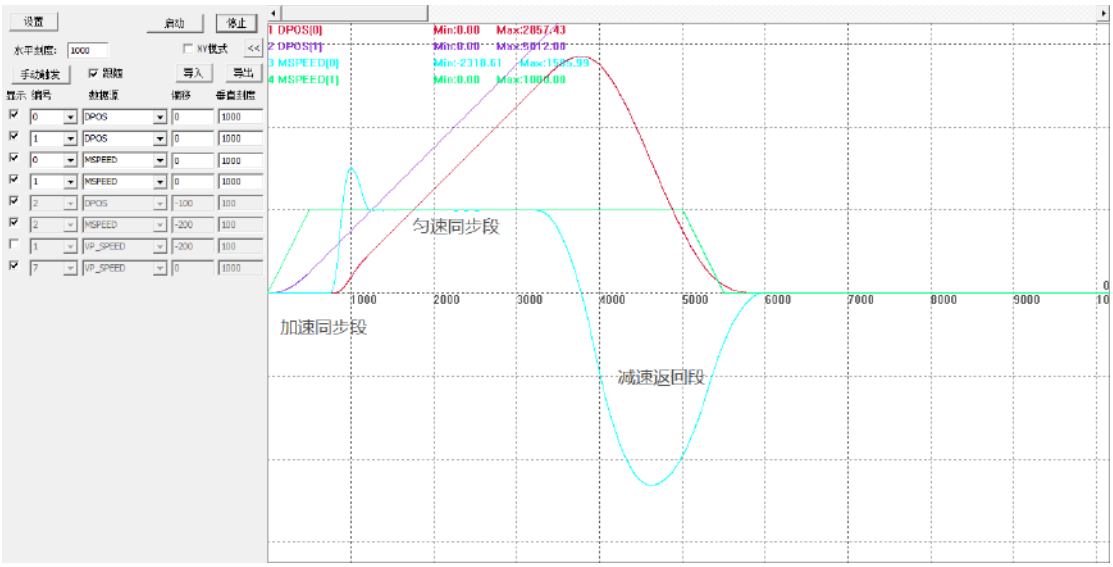
    ZAux_Direct_Single_Vmove(handle, 1,1);//启动轴 1,模拟皮带轴运动
```

```
    ZAux_Direct_MoveWait(handle,0,"MPOS",1,1,500);//等待轴 1 运动到 500 时，启动同步运动
    float fpos = 500;
    ZAux_Direct_MoveSync(handle,0,0,1000,1,1,&FirstAxis,&fpos);//皮带轴运动到 1000，跟随轴轴 0 运动到 500，且已同步速度，同步时间根据加减速度自动计算

    ZAux_Direct_MoveSync(handle,0,2000,1000,1,1,&FirstAxis,&fpos);//继续同步 2S
    fpos=0;
    ZAux_Direct_MoveSync(handle,-1,0,300,1,1,&FirstAxis,&fpos);//随轴 0 停止同步，回到初始位置

    Sleep(5000);
    ZAux_Direct_Single_Cancel(handle, 1,2);//停止轴 1
    ret = ZAux_Close(handle);     //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");

    handle = NULL;
    return 0;
}
```



第三章 高速锁存应用

3.1 高速锁存

3.1.1 指令列表

指令	说明
ZAux_Direct_Regist	位置锁存指令
ZAux_Direct_CycleRegist	连续位置锁存指令
ZAux_Direct_SetCloseWin	设置锁存触发的结束坐标范围点
ZAux_Direct_GetCloseWin	读取锁存触发的结束坐标范围点
ZAux_Direct_GetRegPos	读取返回锁存的测量反馈位置(MPOS)
ZAux_Direct_GetRegPosB	读取返回锁存的测量反馈位置(MPOS)
ZAux_Direct_SetOpenWin	设置锁存触发的开始坐标范围点
ZAux_Direct_GetOpenWin	读取锁存触发的开始坐标范围点
ZAux_Direct_GetMark	读取编码器锁存返回状态
ZAux_Direct_GetMarkB	读取编码器锁存 b 返回状态

3.1.2 重点说明

1. **ZAux_Direct_Regist** 用来锁存轴的测量反馈位置。

2. 支持编码器轴锁存，4 系列及以上控制器最新固件支持虚拟轴、脉冲轴锁存。

3. EtherCAT 支持驱动器锁存，此时使用驱动器 IO 点实现锁存。

4. Rtex 只支持控制器锁存。

5. 当锁存产生时，**ZAux_Direct_GetMark** 会被设置为 ON，同时锁存到的位置会被存储在参数 **ZAux_Direct_GetRegPos** 内。

6. 每个轴有输入信号 R0,R1,EZ 信号可以使用锁存功能。当使用两个信号锁存时，第二个信号锁存使用 **ZAux_Direct_GetMarkB** 和 **ZAux_Direct_GetRegPosB**。

支持编码器轴、总线轴锁存，不同型号的控制器支持锁存的轴类型不同，早期的控制器与运动控制卡只支持编码器类型的轴做锁存。4 系列及以上控制器最新固件支持虚拟轴、脉冲轴锁存。具体情况请详细查看控制器，控制卡相关的资料。

EtherCAT 支持驱动器锁存，此时使用驱动器 IO 点实现锁存，具体模式查看指令语法。

Rtex 只支持控制器锁存。

1.单次锁存模式：**ZAux_Direct_Regist**

输入信号 R0 在控制器中默认映射对应 IN0.(所用控制器，支持锁存输入口生效)

输入信号 R1 在控制器中默认映射对应 IN1.(所用控制器，支持锁存输入口生效)

输入信号 R2 在控制器中默认映射对应 IN2.(所用控制器，支持锁存输入口大于 2 生效)

输入信号 R3 在控制器中默认映射对应 IN3.(所用控制器，支持锁存输入口大于 2 生效)

值	描述
1	当 Z 脉冲上升沿时的绝对位置送到 REG_POS
2	当 Z 脉冲下降沿时的绝对位置送到 REG_POS
3	当输入信号 R0 上升沿的绝对位置送到 REG_POS
4	当输入信号 R0 下降沿的绝对位置送到 REG_POS
6	输入信号 R0 上升沿时的绝对位置送到 REG_POS，Z 信号上升沿时的绝对位置送到 REG_POSB
7	输入信号 R0 上升沿时的绝对位置送到 REG_POS，Z 信号下降沿时的绝对位置送到 REG_POSB
8	输入信号 R0 下降沿时的绝对位置送到 REG_POS，Z 信号上升沿时的绝对位置送到 REG_POSB
9	输入信号 R0 下降沿时的绝对位置送到 REG_POS，Z 信号下降沿时的绝对位置送到 REGPOSB
10	输入信号 R0 上升沿时的绝对位置送到 REG_POS，输入信号 R1 上升沿时的绝对位置送到 REG_POSB
11	输入信号 R0 上升沿时的绝对位置送到 REG_POS，输入信号 R1 下降沿时的绝对位置送到 REG_POSB
12	输入信号 R0 下降沿时的绝对位置送到 REG_POS，输入信号 R1 上升沿时的绝对位置送到 REG_POSB
13	输入信号 R0 下降沿时的绝对位置送到 REG_POS，输入信号 R1 下降沿时的绝对位置送到 REG_POSB
14	输入信号 R1 上升沿时的绝对位置送到 REG_POSB(14 以后 150804 以后版本支持，每个锁存通道独立，支持 4 通道锁存)
15	输入信号 R1 下降沿时的绝对位置送到 REG_POSB
16	Z 信号上升沿时的绝对位置送到 REG_POSB
17	Z 信号下降沿时的绝对位置送到 REG_POSB
18	输入信号 R2 上升沿时的绝对位置送到 REG_POSC
19	输入信号 R2 下降沿时的绝对位置送到 REG_POSC
20	输入信号 R3 上升沿时的绝对位置送到 REG_POSD
21	输入信号 R3 下降沿时的绝对位置送到 REG_POSD

脉冲轴类型一般采用 R0，R1，Z 脉冲这三种锁存；总线轴类型采用 R2，R3 锁存。

2. 连续锁存： **ZAux_Direct_CycleRegist**(ZMC_HANDLE handle,int iaxis, int imode,int iTabStart,int iTabNum)

参数	描述
handle	连接句柄
iaxis	控制轴
imode	锁存方式
iTabStart	连续锁存的内容存储的 table 位置，第一个 table 元素存储锁存的个数，后面存储锁存的坐标，最多保存个数=numes-1，溢出时循环写入。
iTabNum	占用的 table 个数。

通过把模式加 100 来支持连续锁存，锁存结果存储到 TABLE 寄存器里面。

分别对两个通道进行连续锁存，可以实现上下边沿的连续锁存。

ECI 控制器：20150829 以上固件支持

4 系列控制器：20170523 以上固件支持

mode：

输入信号 R0 在控制器中默认映射对应 IN0.(所用控制器，支持锁存输入口生效)

输入信号 R1 在控制器中默认映射对应 IN1.(所用控制器，支持锁存输入口生效)

输入信号 R2 在控制器中默认映射对应 IN2.(所用控制器，支持锁存输入口大于 2 生效)

输入信号 R3 在控制器中默认映射对应 IN3.(所用控制器，支持锁存输入口大于 2 生效)

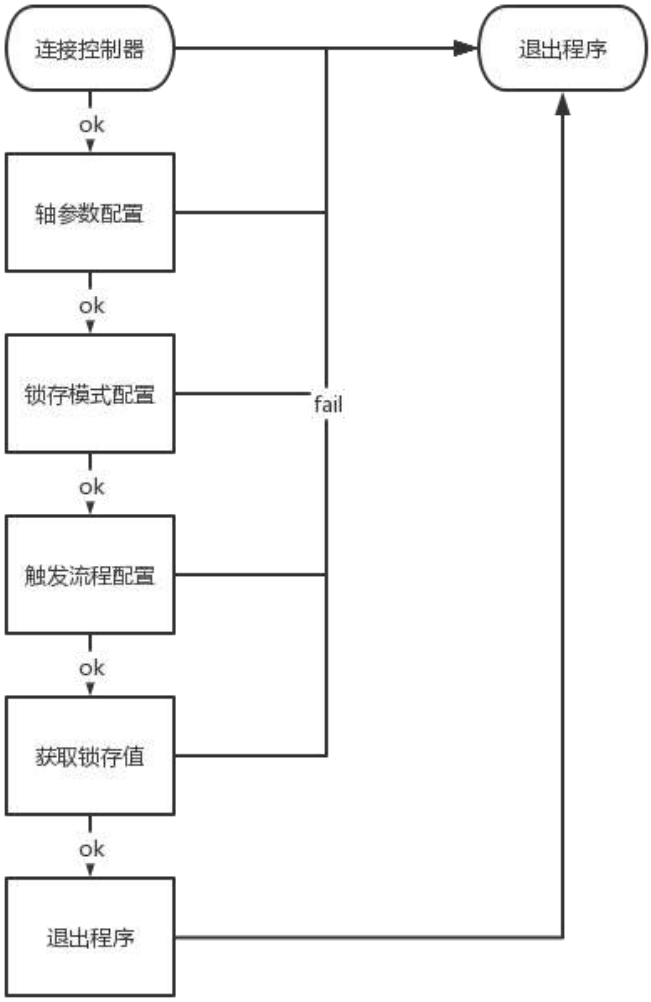
值	描述
101	当 Z 脉冲上升沿时的绝对位置送到 REG_POS
102	当 Z 脉冲下降沿时的绝对位置送到 REG_POS
103	当输入信号 R0 上升沿的绝对位置送到 REG_POS
104	当输入信号 R0 下降沿的绝对位置送到 REG_POS
114	输入信号 R1 上升沿时的绝对位置送到 REG_POSB
115	输入信号 R1 下降沿时的绝对位置送到 REG_POSB
116	Z 信号上升沿时的绝对位置送到 REG_POSB
117	Z 信号下降沿时的绝对位置送到 REG_POSB
123	当输入信号 R0 上升沿的绝对位置送到 REG_POSB
124	当输入信号 R0 下降沿的绝对位置送到 REG_POSB
133	当输入信号 R0 上升沿的绝对位置送到 REG_POS，下一次切换下降沿，轮流切换。
134	当输入信号 R0 下降沿的绝对位置送到 REG_POS，下一次切换上升沿，轮流切换。
135	当输入信号 R1 上升沿的绝对位置送到 REG_POSB，下一次切换下降沿，轮流切换。下一次切换下降沿，轮流切换。
136	当输入信号 R1 下降沿的绝对位置送到 REG_POSB，下一次切换上升沿，

	轮流切换。

3.1.3 例程

3.1.3-1 位置锁存

轴 0 在持续不断的运动，当 IN0 感应到信号，则高速锁存轴 0 编码器位置值。




```
// test1.cpp : 定义控制台应用程序的入口点。
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
```

```
#include <math.h>
void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
        Sleep(2000);
        exit(0);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    //char *ip_addr = (char *)"127.0.0.1";              //控制器 IP 地址
    char *ip_addr = (char *)"192.168.0.11";               //实物控制器 IP 地址默认
192.168.0.11
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("控制器连接成功! \n");

    int iaxis = 0;
    float DposValue = 0;
    float MposValue = 0;
    int mode = 3;     //等待 R0 上升沿的绝对位置
    int piValue;
    float pfValue;
    ret = ZAux_Direct_SetDpos( handle, iaxis, DposValue);//轴 0 Dpos 清 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret);

    ret = ZAux_Direct_SetMpos( handle, iaxis, MposValue);//轴 0 Mpos 清 0
```

```
    commandCheckHandler("ZAux_Direct_SetMpos", ret);

    ret = ZAux_Direct_Single_Vmove( handle, iaxis, 1);//轴 0 正向运动
    commandCheckHandler("ZAux_Direct_Single_Move", ret);

    ret = ZAux_Direct_Regist( handle, iaxis, mode);//设置单次锁存，锁存模式：IN0 上升沿触发锁存
    commandCheckHandler("ZAux_Direct_Regist", ret);

    ret = ZAux_Direct_MoveWait(handle, 1,"Mpos",iaxis,1,432);//轴 1 缓冲区里面加入等待轴 0 MPOS 大于 432 时才执行下一条指令的阻塞（这里占用了轴 1 的缓冲）
    commandCheckHandler("ZAux_Direct_Regist", ret);

    /*MPOS 等于 432 后，往轴 1 缓冲区里写入上升沿，本例程连接的控制器为 ZMC 系列，低电平为有效电平，
    所以把 IN(0)从 1 置为 0 才可以形成上升沿（若 ECI 系列控制卡，则需要把 IN(0)从 0 置为 1 才可以形成上升沿）*/

    ret = ZAux_Direct_MoveOp(handle, 1,0,1);//轴 1 缓冲区里面写入把 OUT0 置 1（这里使用的控制器，硬件接口上已经把 OUT0 直连 IN0 了）
    commandCheckHandler("ZAux_Direct_Regist", ret);
    ret = ZAux_Direct_MoveOp(handle, 1,0,0);////轴 1 缓冲区里面写入把 OUT0 置 0（这里使用的控制器，硬件接口上已经把 OUT0 直连 IN0 了）
    commandCheckHandler("ZAux_Direct_Regist", ret);

    while (1)
    {
        Sleep(2000);
        ZAux_Direct_GetMark(handle, iaxis, &piValue);

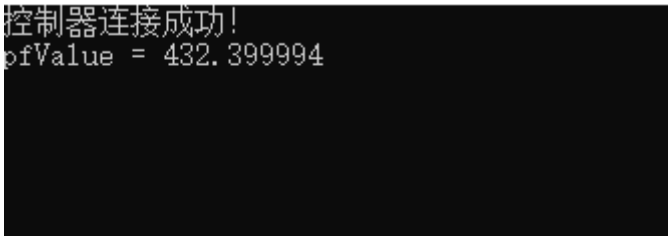
        if (piValue == -1)//触发锁存
        {
            ret =ZAux_Direct_GetRegPos(handle,0,&pfValue);
             commandCheckHandler("ZAux_Direct_GetRegPos", ret);
            printf("pfValue = %f\n", pfValue);
            break;
        }
```

```
    }


    ret = ZAux_Direct_Single_Cancel( handle, 0, 2);//停止运动
    commandCheckHandler("ZAux_Direct_Single_Cancel", ret);
    Sleep(5000);
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");

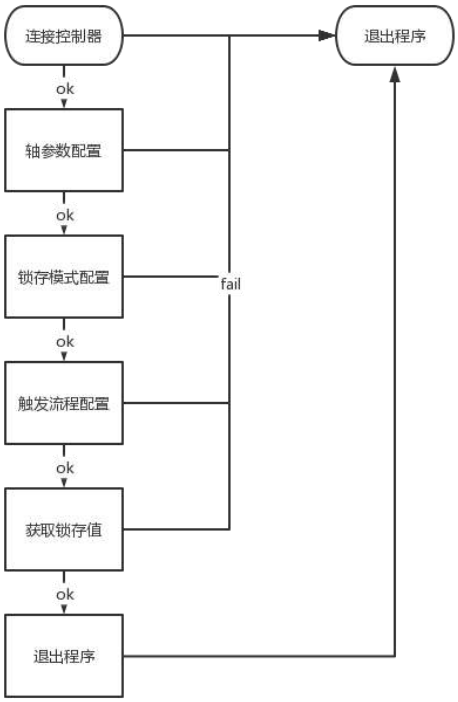

    handle = NULL;
    return 0;
}
```

输出结果：



3.1.3-2 连续位置锁存

轴 0 在持续不断的运动，当 IN0 感应到信号，则高速连续锁存轴 0 编码器位置值。



```
// test1.cpp : 定义控制台应用程序的入口点。
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
#include <math.h>
void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
        Sleep(2000);
        exit(0);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    //char *ip_addr = (char *)"127.0.0.1";              //控制器 IP 地址
    char *ip_addr = (char *)"192.168.0.11";               //实物控制器 IP 地址默认
192.168.0.11
    ZMC_HANDLE handle = NULL;                   //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);   //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("控制器连接成功! \n");

    int iaxis = 0;
    float DposValue = 0;
    float MposValue = 0;
```

```
    int mode = 103;        //等待 R0 上升沿的绝对位置
    float pfValues[10];
    float pfValue;
    int AType = 1;                          //设置轴的类型
    int UnitValue = 100;                    //设置脉冲当量的值
    int SpeedValue = 200;                   //设置速度的值
    int AccelValue = 2000;                  //设置加速度的值
    int DecelValue = 2000;                  //设置减速度的值

    ret = ZAux_Direct_SetAtype(handle, iaxis, AType);          //设置轴 0 的类型
    commandCheckHandler("ZAux_Direct_SetAtype", ret);

    ZAux_Direct_SetUnits(handle, iaxis, UnitValue);       //设置轴 0 的脉冲当量
    commandCheckHandler("ZAux_Direct_SetUnits", ret);

    ZAux_Direct_SetSpeed(handle, iaxis,SpeedValue);       //设置轴 0 的速度的值
    commandCheckHandler("ZAux_Direct_SetSpeed", ret);

    ZAux_Direct_SetAccel(handle, iaxis, AccelValue);      //设置轴 0 的加速度的值
    commandCheckHandler("ZAux_Direct_SetAccel", ret);

    ZAux_Direct_SetDecel(handle, iaxis, DecelValue);      //设置轴 0 的减速度的值
    commandCheckHandler("ZAux_Direct_SetDecel", ret);
    ret = ZAux_Direct_SetDpos( handle, iaxis, DposValue);//轴 0 Dpos 清 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret);

    ret = ZAux_Direct_SetMpos( handle, iaxis, MposValue);//轴 0 Mpos 清 0
    commandCheckHandler("ZAux_Direct_SetMpos", ret);

    ret = ZAux_Direct_Single_Vmove( handle, iaxis, 1);//轴 0 正向运动
    commandCheckHandler("ZAux_Direct_Single_Move", ret);

    ret = ZAux_Direct_CycleRegist( handle, iaxis, mode,0,100);//设置连续锁存，锁存模
式：IN0 上升沿触发锁存，锁存值存放 table(0)-table(99)
    commandCheckHandler("ZAux_Direct_CycleRegist", ret);

    ret = ZAux_Direct_MoveWait(handle, 1,"Mpos",iaxis,1,432);//轴 1 缓冲区里面加入等
```

```
待轴 0 MPOS 大于 432 时才执行下一条指令的阻塞（这里占用了轴 1 的缓冲）
    commandCheckHandler("ZAux_Direct_MoveWait", ret);

    /*MPOS 等于 432 后，往轴 1 缓冲区里写入上升沿，本例程连接的控制器为 ZMC 系列，低电平为有效电平，
    所以把 IN(0)从 1 置为 0 才可以形成上升沿（若 ECI 系列控制卡，则需要把 IN(0)从 0 置为 1 才可以形成上升沿）*/
    int i;

    for (i=0;i<10;i++)//每隔 1 秒触发 1 次锁存，触发 10 次
    {
        ret = ZAux_Direct_MoveOp(handle, 1,0,1);//轴 1 缓冲区里面写入把 OUT0 置 1（这里使用的控制器，硬件接口上已经把 OUT0 直连 IN0 了）
        commandCheckHandler("ZAux_Direct_MoveOp ", ret);
        ret = ZAux_Direct_MoveOp(handle, 1,0,0);////轴 1 缓冲区里面写入把 OUT0 置 0（这里使用的控制器，硬件接口上已经把 OUT0 直连 IN0 了）
        commandCheckHandler("ZAux_Direct_MoveOp ", ret);
        ret = ZAux_Direct_MoveDelay(handle, 1,1000);////轴 1 延时 1s
        commandCheckHandler("ZAux_Direct_MoveDelay", ret);
    }

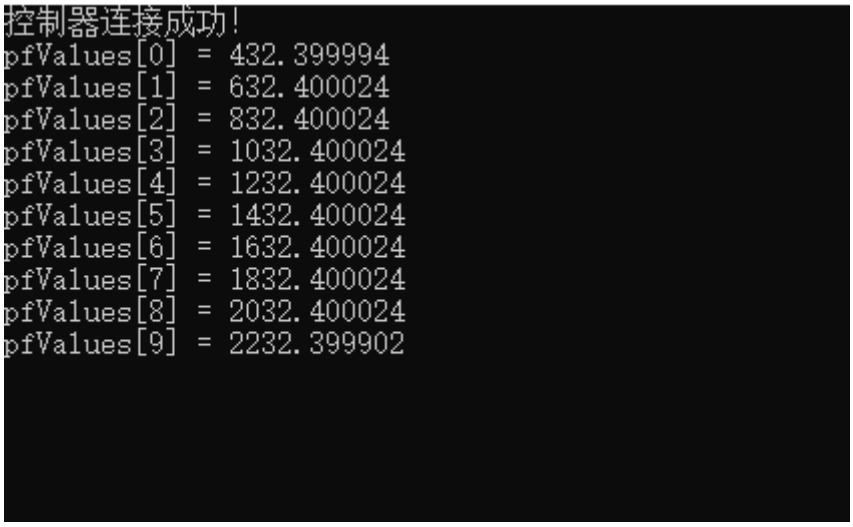
    while (1)
    {
        Sleep(2000);
        ZAux_Direct_GetTable(handle, 0, 1,&pfValue);

        if (pfValue == 10)//触发锁存 10 次退出循环
        {
            ret = ZAux_Direct_GetTable(handle, 1, 10,pfValues);
             commandCheckHandler("ZAux_Direct_GetTable", ret);
            for (i=0;i<10;i++)//每隔 1 秒触发 1 次锁存，触发 10 次
            {
                printf("pfValues[%d] = %f\n", i,pfValues[i]);
            }
            break;
        }
    }
```

```
    ret = ZAux_Direct_Single_Cancel( handle, 0, 2);//停止运动
    commandCheckHandler("ZAux_Direct_Single_Cancel", ret);
    Sleep(5000);
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");

    handle = NULL;
    return 0;
}
```

输出结果：



第四章 位置比较输出

4.1 位置比较输出应用

4.1.1 指令列表

指令	说明
ZAux_Direct_Pswitch	软件位置比较输出
ZAux_Direct_HwPswitch	硬件位置比较输出
ZAux_Direct_GetHwPswitchBuff	硬件位置比较输出剩余缓冲数
ZAux_Direct_HwPswitch2	总线硬件位置比较输出（脉冲轴和总线轴均支持此指令）
ZAux_Direct_HwPswitch2_2	硬件位置比较输出 2
ZAux_Direct_HwTimer	硬件定时器用于硬件比较输出后一段时间后还原电平
ZAux_Direct_MoveHwPswitch2	硬件位置比较输出，运动缓冲方式
ZAux_Direct_MoveHwTimer	硬件定时器用于硬件比较输出后一段时间后还原电平，运动缓冲方式,4 系列产品支持
ZAux_Direct_SetHwPs2AxisNum	设置 PS2 轴号
ZAux_Direct_GetHwPs2AxisNum	获得 PS2 轴号
ZAux_Direct_GetHwPs2Counts	获得 PS2 比较的点数

4.1.2 重点说明

1. 软件位置比较输出

根据轴的位置去操作对应的输出口，如果有多个 ZAux_Direct_Pswitch 操作同一个输出口时，需要按照编号顺序进行排列。

2. 精准位置输出

支持硬件比较输出功能的控制器才可使用精准输出功能，使用 ZAux_Direct_SetParam(handle, "AXIS_ZSET",轴号, 模式)

指令设置是否开启精准输出。普通的输出操作需要等待一个控制器周期才可执行。而精准输出操作，可在电机发出一个脉冲内响应，大大提高如点胶工艺的精度，同时可以使用 ZAux_Direct_SetParam(handle," MOVEOP_DELAY",轴号, 延时时间)函数调整响应时间（提前或延迟）。

模式：

Bit0	1-当前运动速度缺省使用插补速度，0-当前运动速度使用单轴的速度
Bit1	1-使用 **ZAux_Direct_MoveOp** 精确输出功能 0-**ZAux_Direct_MoveOp** 原来的方式
Bit4	1-对带编码器功能的轴，使用编码器位置的 **ZAux_Direct_MoveOp** 精准方式

多个轴插补时，对主轴设置

用法：

ZAux_Direct_SetParam(handle,"AXIS_ZSET",0, 19)//开启轴 0 精准输出

3. 硬件位置比较输出

通过设定反馈位置表，写到 Table 寄存器中，在运动中比较 table 中的值去操作对应的输出口。

硬件位置比较缓冲数为 1024 个，可以连续调用 1024 个硬件位置指令。

硬件位置指令调用后，不受后面的坐标修改功能的影响，硬件位置指令 TABLE 寄存器存储的坐标必须在调用时是正确的，因此尽量手动修改坐标，要规避硬件位置指令与坐标循环自动修改的随机冲突。

因为自动循环修改坐标不受程序控制，无法确定是在硬件位置的前面还是后面，这样 TABLE 寄存器里面的坐标就无法确定。总线轴请使用 **ZAux_Direct_HwPswitch2** 指令。

（**注**：4 系列及以上产品，最新固件）

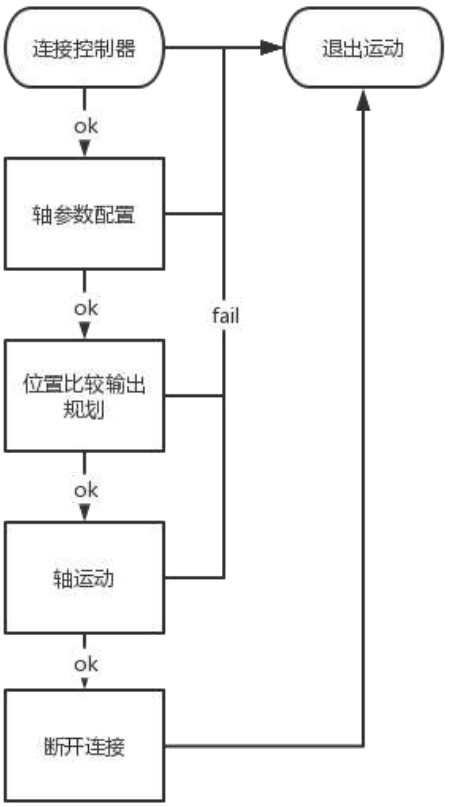
支持 PS0 功能型号具体可咨询正运动销售工程师。

4.1.3 例程

4.1.3-1 软件位置比较输出

轴 0 持续运动，在 100-200 的位置输出 OP0。

程序框图：




```
// test1.cpp : 定义控制台应用程序的入口点。
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
#include <math.h>
void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
        Sleep(2000);
        exit(0);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
```

```
    char *ip_addr = (char *)"127.0.0.1";              //仿真器 IP 地址
    //char *ip_addr = (char *)"192.168.0.11";               //实物控制器 IP 地址默认
192.168.0.11
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("控制器连接成功! \n");

    int iaxis = 0;
    float DposValue = 0;
    float MposValue = 0;
    int AType = 1;                          //设置轴的类型
    int UnitValue = 100;                    //设置脉冲当量的值
    int SpeedValue = 100;                   //设置速度的值
    int AccelValue = 2000;                  //设置加速度的值
    int DecelValue = 2000;                  //设置减速度的值

    ret = ZAux_Direct_SetAtype(handle, iaxis, AType);         //设置轴 0 的类型
    commandCheckHandler("ZAux_Direct_SetAtype", ret);
    ZAux_Direct_SetUnits(handle, iaxis, UnitValue);       //设置轴 0 的脉冲当量
    commandCheckHandler("ZAux_Direct_SetUnits", ret);
    ZAux_Direct_SetSpeed(handle, iaxis,SpeedValue);       //设置轴 0 的速度的值
    commandCheckHandler("ZAux_Direct_SetSpeed", ret);
    ZAux_Direct_SetAccel(handle, iaxis, AccelValue);      //设置轴 0 的加速度的值
    commandCheckHandler("ZAux_Direct_SetAccel", ret);
    ZAux_Direct_SetDecel(handle, iaxis, DecelValue);      //设置轴 0 的减速度的值
    commandCheckHandler("ZAux_Direct_SetDecel", ret);
    ret = ZAux_Direct_SetDpos( handle, iaxis, DposValue);//轴 0 Dpos 清 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret);
    ret = ZAux_Direct_SetMpos( handle, iaxis, MposValue);//轴 0 Mpos 清 0
    commandCheckHandler("ZAux_Direct_SetMpos", ret);
```

```
    float GetValue;
    uint32 piValue;
    ret = ZAux_Direct_Pswitch(handle,0,1,iaxis,0,1,100,200);  //轴 0 的 100-200 的 DPOS
范围内去操作输出口
    commandCheckHandler("ZAux_Direct_Pswitch",ret);

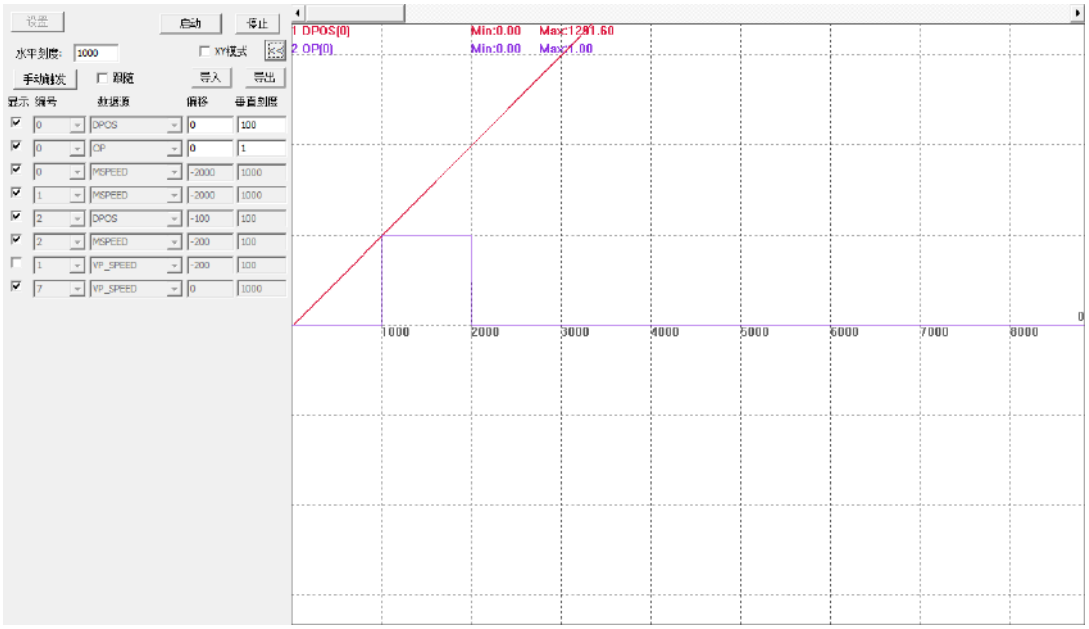
    ZAux_Trigger(handle);//启动示波器
    ret = ZAux_Direct_Single_Vmove( handle, iaxis, 1);//轴 0 正向运动
    commandCheckHandler("ZAux_Direct_Single_Move", ret);

    while (1)
    {
        Sleep(200);
        ret = ZAux_Direct_GetOp(handle,iaxis,&piValue);           //获取输出口 1 的状
态 1 表示打开 0 表示关闭
        commandCheckHandler("ZAux_Direct_GetDA",ret);
        printf("OP 状态 piValue=%d\n", piValue);
        ret = ZAux_Direct_GetDpos(handle,iaxis,&GetValue);            //获取当前位置
        printf("位置值 GetValue=%f\n", GetValue);
        if (GetValue>300)//位置大于 300 退出打印
        {
            break;
        }
    }

    ret = ZAux_Direct_Single_Cancel( handle, 0, 2);//停止运动
    commandCheckHandler("ZAux_Direct_Single_Cancel", ret);
    Sleep(5000);
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");

    handle = NULL;
    return 0;
}
```

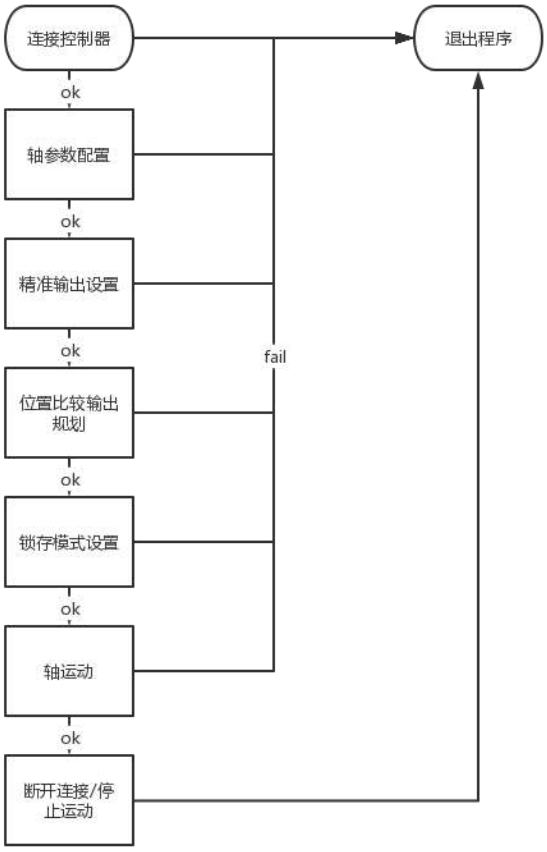
输出波形图：



4.1.3-2 飞拍应用（ZAux_Direct_HwPswitch）

假设轴 0 为持续运动的传送带轴，需要轴 0 在位置 100，150，200，250，300 时触发相机拍照，且锁存拍照时编码器的实际位置值。

程序框图：



```
// test1.cpp : 定义控制台应用程序的入口点。
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
#include <math.h>
void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
        Sleep(2000);
        exit(0);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    //char *ip_addr = (char *)"127.0.0.1";              //仿真器 IP 地址
    char *ip_addr = (char *)"192.168.0.11";               //实物控制器 IP 地址默认
192.168.0.11
    ZMC_HANDLE handle = NULL;                   //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);   //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("控制器连接成功! \n");

    int iaxis = 0;
    float DposValue = 0;
    float MposValue = 0;
```

```
    int AType = 1;                          //设置轴的类型
    int UnitValue = 100;                    //设置脉冲当量的值
    int SpeedValue = 50;                    //设置速度的值
    int AccelValue = 2000;                  //设置加速度的值
    int DecelValue = 2000;                  //设置减速度的值

    ret = ZAux_Direct_SetAtype(handle, iaxis, AType);         //设置轴 0 的类型
    commandCheckHandler("ZAux_Direct_SetAtype", ret);
    ZAux_Direct_SetUnits(handle, iaxis, UnitValue);       //设置轴 0 的脉冲当量
    commandCheckHandler("ZAux_Direct_SetUnits", ret);
    ZAux_Direct_SetSpeed(handle, iaxis,SpeedValue);       //设置轴 0 的速度的值
    commandCheckHandler("ZAux_Direct_SetSpeed", ret);
    ZAux_Direct_SetAccel(handle, iaxis, AccelValue);      //设置轴 0 的加速度的值
    commandCheckHandler("ZAux_Direct_SetAccel", ret);
    ZAux_Direct_SetDecel(handle, iaxis, DecelValue);      //设置轴 0 的减速度的值
    commandCheckHandler("ZAux_Direct_SetDecel", ret);
    ret = ZAux_Direct_SetDpos( handle, iaxis, DposValue);//轴 0 Dpos 清 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret);
    ret = ZAux_Direct_SetMpos( handle, iaxis, MposValue);//轴 0 Mpos 清 0
    commandCheckHandler("ZAux_Direct_SetMpos", ret);

    ret =ZAux_Direct_SetParam(handle,"AXIS_ZSET",0, 19);//开启轴 0 精准输出
    commandCheckHandler("ZAux_Direct_SetParam",ret);

    float TableValue[10]={100,101,150,151,200,201,250,251,300,301};//规划位数输出点
    ret =ZAux_Direct_SetTable(handle,0,10,TableValue );//设置 Table 值
    commandCheckHandler("ZAux_Direct_SetTable",ret);

    ret = ZAux_Direct_HwPswitch(handle,0,2,0,0,0,0);//停止并删除没有完成的比较点,预
防之前有未完成的比较点没删除
    commandCheckHandler("ZAux_Direct_HwPswitch",ret);
    ret = ZAux_Direct_HwPswitch(handle,0,1,1,0,0,10);//配置位置比较输出
    commandCheckHandler("ZAux_Direct_HwPswitch",ret);

    ret = ZAux_Direct_Regist( handle, iaxis, 4);//设置单次锁存，锁存模式：IN0 下降沿
触发锁存
    commandCheckHandler("ZAux_Direct_Regist", ret);
```

```
    ZAux_Trigger(handle);//启动示波器
    ret = ZAux_Direct_Single_Vmove( handle, iaxis, 1);//轴 0 正向运动
    commandCheckHandler("ZAux_Direct_Single_Move", ret);

    float GetValue,pfValue;
    int piValue;

    while (1)
    {
        Sleep(200);
        ZAux_Direct_GetMark(handle, iaxis, &piValue);//获取锁存是否触发

        if (piValue == -1)//触发锁存
        {
            ZAux_Direct_GetRegPos(handle,0,&pfValue);
            printf("pfValue = %f\n", pfValue);
            ret = ZAux_Direct_Regist( handle, iaxis, 4);//设置单次锁存，锁存模式：
IN0 下降沿触发锁存
            commandCheckHandler("ZAux_Direct_Regist", ret);
        }

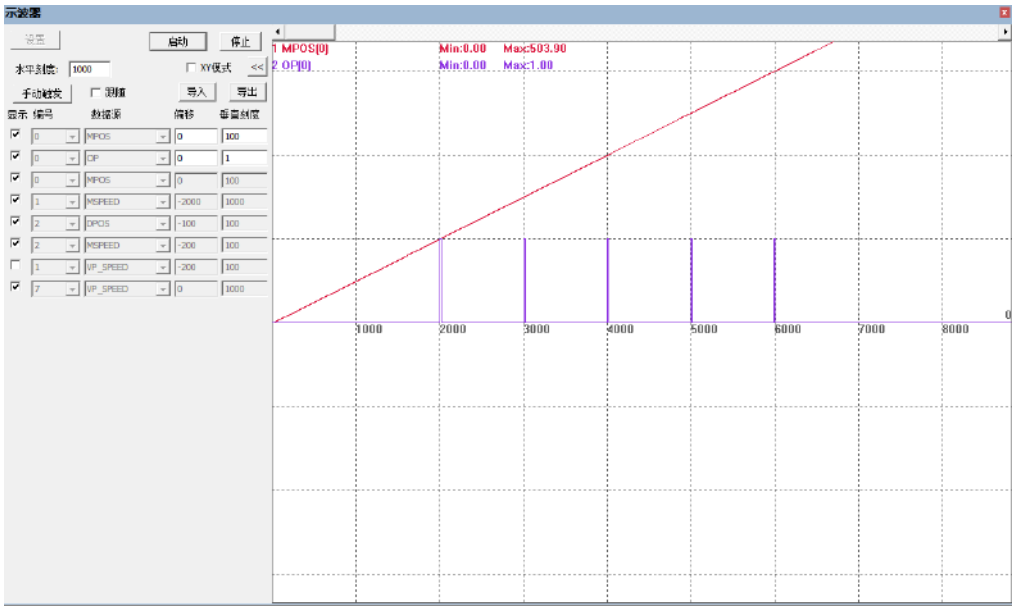
        ret = ZAux_Direct_GetDpos(handle,iaxis,&GetValue);              //获取位置值

        if (GetValue>500)//位置大于 500 退出
        {
            printf("位置值 GetValue=%f\n", GetValue);
            break;
        }
    }

    ret = ZAux_Direct_Single_Cancel( handle, 0, 2);//停止运动
    commandCheckHandler("ZAux_Direct_Single_Cancel", ret);
    Sleep(5000);
    ret = ZAux_Close(handle);     //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
```

```
    handle = NULL;
    return 0;
}
```

输出结果波形：



4.1.3-3 飞拍应用（多个相机绑定一个轴）（ZAux_Direct_HwPswitch2）

假设轴 0 为持续运动的传送带轴，需要轴 0 在位置 100，150，200，250 时触发相机 1，相机 2，相机 3，相机 4 拍照。相机 1，2，3，4 所对应的输出口分别是输出 0，1，2，3。

```
// test1.cpp : 定义控制台应用程序的入口点。
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
#include <math.h>
void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
        Sleep(2000);
        exit(0);
```

```
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    //char *ip_addr = (char *)"127.0.0.1";              //仿真器 IP 地址
    char *ip_addr = (char *)"192.168.0.11";               //实物控制器 IP 地址默认
192.168.0.11
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("控制器连接成功! \n");

    int iaxis = 0;
    float DposValue = 0;
    float MposValue = 0;
    int AType = 1;                          //设置轴的类型
    int UnitValue = 100;                    //设置脉冲当量的值
    int SpeedValue = 50;                    //设置速度的值
    int AccelValue = 2000;                  //设置加速度的值
    int DecelValue = 2000;                  //设置减速度的值

    ret = ZAux_Direct_SetAtype(handle, iaxis, AType);         //设置轴 0 的类型
    commandCheckHandler("ZAux_Direct_SetAtype", ret);
    ZAux_Direct_SetUnits(handle, iaxis, UnitValue);       //设置轴 0 的脉冲当量
    commandCheckHandler("ZAux_Direct_SetUnits", ret);
    ZAux_Direct_SetSpeed(handle, iaxis,SpeedValue);       //设置轴 0 的速度的值
    commandCheckHandler("ZAux_Direct_SetSpeed", ret);
    ZAux_Direct_SetAccel(handle, iaxis, AccelValue);      //设置轴 0 的加速度的值
    commandCheckHandler("ZAux_Direct_SetAccel", ret);
    ZAux_Direct_SetDecel(handle, iaxis, DecelValue);      //设置轴 0 的减速度的值
    commandCheckHandler("ZAux_Direct_SetDecel", ret);
    ret = ZAux_Direct_SetDpos( handle, iaxis, DposValue);//轴 0 Dpos 清 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret);
    ret = ZAux_Direct_SetMpos( handle, iaxis, MposValue);//轴 0 Mpos 清 0
    commandCheckHandler("ZAux_Direct_SetMpos", ret);

    ZAux_Direct_SetParam(handle,"HW_PS2AXISNUM",1,0);//使用轴 1 的缓冲区，比较轴 0 的
```

```
位置
    commandCheckHandler("ZAux_Direct_SetParam",ret);

    ZAux_Direct_SetParam(handle,"HW_PS2AXISNUM",2,0);//使用轴 2 的缓冲区，比较轴 0 的
位置
    commandCheckHandler("ZAux_Direct_SetParam",ret);

    ZAux_Direct_SetParam(handle,"HW_PS2AXISNUM",3,0);//使用轴 3 的缓冲区，比较轴 0 的
位置
    commandCheckHandler("ZAux_Direct_SetParam",ret);

    uint32 uValue=0;

    ZAux_Direct_SetOutMulti(handle,0,3,&uValue);//初始化输出口 0-3 的状态
    commandCheckHandler("ZAux_Direct_SetOutMulti",ret);

    float TableValue[8]={100,101,150,151,200,201,250,251};//规划位数输出点
    ret =ZAux_Direct_SetTable(handle,0,8,TableValue );//设置 Table 值
    commandCheckHandler("ZAux_Direct_SetTable",ret);

    ret = ZAux_Direct_HwPswitch2(handle,0,2,0,0,0,0,0,0);//预防之前有未完成的比较点
没删除,停止并删除轴 0 没有完成的比较点
    commandCheckHandler("ZAux_Direct_HwPswitch",ret);

    ret = ZAux_Direct_HwPswitch2(handle,0,1,0,1,0,1,-1,0);//配置位置比较输出,轴 0，模
式 1,输出口 0,输出 1，输出位置存在 table(0)-table(1),-1,不使用方向
    commandCheckHandler("ZAux_Direct_HwPswitch",ret);

    ret = ZAux_Direct_HwPswitch2(handle,1,2,0,0,0,0,0,0);//预防之前有未完成的比较点
没删除,停止并删除轴 1 没有完成的比较点
    commandCheckHandler("ZAux_Direct_HwPswitch",ret);

    ret = ZAux_Direct_HwPswitch2(handle,1,1,1,1,2,3,-1,0);//配置位置比较输出,轴 1，模
式 1,输出口 1,输出 1，输出位置存在 table(2)-table(3),-1,不使用方向
    commandCheckHandler("ZAux_Direct_HwPswitch",ret);

    ret = ZAux_Direct_HwPswitch2(handle,2,2,0,0,0,0,0,0);//预防之前有未完成的比较点
没删除,停止并删除轴 2 没有完成的比较点
    commandCheckHandler("ZAux_Direct_HwPswitch",ret);

    ret = ZAux_Direct_HwPswitch2(handle,2,1,2,1,4,5,-1,0);//配置位置比较输出,轴 2，模
式 1,输出口 2,输出 1，输出位置存在 table(4)-table(5),-1,不使用方向
    commandCheckHandler("ZAux_Direct_HwPswitch",ret);
```

```
    ret = ZAux_Direct_HwPswitch2(handle,3,2,0,0,0,0,0,0);//预防之前有未完成的比较点没删除,停止并删除轴 3 没有完成的比较点
    commandCheckHandler("ZAux_Direct_HwPswitch",ret);

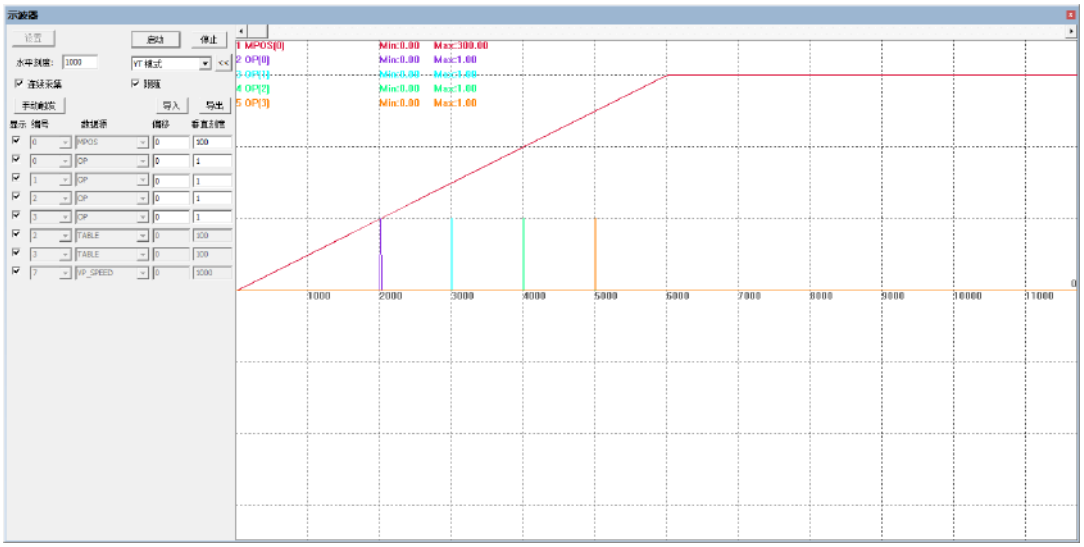
    ret = ZAux_Direct_HwPswitch2(handle,3,1,3,1,6,7,-1,0);//配置位置比较输出,轴 3，模式 1,输出口 1,输出 1，输出位置存在 table(6)-table(7),-1,不使用方向
    commandCheckHandler("ZAux_Direct_HwPswitch",ret);

    ZAux_Trigger(handle);//启动示波器
    ret = ZAux_Direct_Single_Move( handle, iaxis, 300);//轴 0 正向运动
    commandCheckHandler("ZAux_Direct_Single_Move", ret);

    Sleep(5000);
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");

    handle = NULL;
    return 0;
}
```

输出结果波形：



4.1.3-4 飞拍应用（2D 比较输出）

假设轴 0 轴 1 运动轴，需要轴 0，轴 1 构成的平面坐标系，在位置（100，100）（200，200），（300，300）时触发相机拍照。4 系列 170706 以后固件版本支持。

```
// test1.cpp : 定义控制台应用程序的入口点。
//

#include "stdafx.h"
```

```
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
#include <math.h>
void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
        Sleep(2000);
        exit(0);
    }
}
/*参数:  mode      25, 26 2D 的比较模式
    Opnum      对应的输出口
    Opstate  第一个比较点的输出状态.
    maxerr   比较位置每个轴左右的脉冲偏差,  进入偏差范围后开始比较.
    num      TABLE 里面存储的比较点个数.
    tablepos 第一个比较点坐标所在 TABLE 编号

    与 hwtimer 并用时, 可以动态调整 hwtimer 参数.
    ModePara1:脉冲时间
    ModePara2:脉冲个数
    ModePara3:脉冲周期
    */
int32 __stdcall ZAux_Direct_HwPswitch2_2D(ZMC_HANDLE handle, int Axisnum,int Mode,int
Opnum , int Opstate,int maxerr,int num, int tablepos, float ModePara1=0, float
ModePara2=0,float ModePara3=0)
{
    if(0 > Axisnum || Axisnum > MAX_AXIS_AUX)
    {
        return  ERR_AUX_PARAERR;
    }
    char  cmdbuff[2048];
    char  cmdbuffAck[2048];
    //生成命令
    switch(Mode)
    {
    case 25:
        sprintf(cmdbuff, "HW_PSWITCH2(%d,%d,%d,%d,%d,%d) AXIS(%d)", Mode, Opnum,
Opstate, maxerr,num,tablepos,Axisnum);
        break;
    case 26:
        sprintf(cmdbuff, "HW_PSWITCH2(%d,%d,%d,%d,%d,%d,%f,%f,%f) AXIS(%d)", Mode,
```

```
Opnum, Opstate, maxerr,num,tablepos,ModePara1,ModePara2,ModePara3,Axisnum);
        break;

    }
    //调用命令执行函数
    return ZAux_Execute(handle, cmdbuff, cmdbuffAck,2048);
}
int _tmain(int argc, _TCHAR* argv[])
{
    //char *ip_addr = (char *)"127.0.0.1";              //仿真器 IP 地址
    char *ip_addr = (char *)"192.168.0.11";               //实物控制器 IP 地址默认
192.168.0.11
    ZMC_HANDLE handle = NULL;                   //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);   //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("控制器连接成功! \n");

    int iaxis = 0;
    float DposValue = 0;
    float MposValue = 0;
    int AType = 1;                          //设置轴的类型
    int UnitValue = 100;                    //设置脉冲当量的值
    int SpeedValue = 50;                    //设置速度的值
    int AccelValue = 2000;                  //设置加速度的值
    int DecelValue = 2000;                  //设置减速度的值

    ret = ZAux_Direct_SetAtype(handle, iaxis, AType);       //设置轴 0 的类型
    commandCheckHandler("ZAux_Direct_SetAtype", ret);
    ZAux_Direct_SetUnits(handle, iaxis, UnitValue);     //设置轴 0 的脉冲当量
    commandCheckHandler("ZAux_Direct_SetUnits", ret);
    ZAux_Direct_SetSpeed(handle, iaxis,SpeedValue);     //设置轴 0 的速度的值
    commandCheckHandler("ZAux_Direct_SetSpeed", ret);
    ZAux_Direct_SetAccel(handle, iaxis, AccelValue);    //设置轴 0 的加速度的值
    commandCheckHandler("ZAux_Direct_SetAccel", ret);
    ZAux_Direct_SetDecel(handle, iaxis, DecelValue);    //设置轴 0 的减速度的值
    commandCheckHandler("ZAux_Direct_SetDecel", ret);
    ret = ZAux_Direct_SetDpos( handle, 0, 0);//轴 0 Dpos 清 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret);
```

```
    ret = ZAux_Direct_SetMpos( handle, 0, 0);//轴 0 Mpos 清 0
    commandCheckHandler("ZAux_Direct_SetMpos", ret);

    ret = ZAux_Direct_SetAtype(handle, 1, AType);          //设置轴 1 的类型
    commandCheckHandler("ZAux_Direct_SetAtype", ret);
    ZAux_Direct_SetUnits(handle, 1, UnitValue);        //设置轴 1 的脉冲当量
    commandCheckHandler("ZAux_Direct_SetUnits", ret);
    ZAux_Direct_SetSpeed(handle, 1,SpeedValue);        //设置轴 1 的速度的值
    commandCheckHandler("ZAux_Direct_SetSpeed", ret);
    ZAux_Direct_SetAccel(handle, 1, AccelValue);       //设置轴 1 的加速度的值
    commandCheckHandler("ZAux_Direct_SetAccel", ret);
    ZAux_Direct_SetDecel(handle, 1, DecelValue);       //设置轴 1 的减速度的值
    commandCheckHandler("ZAux_Direct_SetDecel", ret);
    ret = ZAux_Direct_SetDpos( handle, 1, DposValue);//轴 1 Dpos 清 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret);
    ret = ZAux_Direct_SetMpos( handle, 1, MposValue);//轴 1 Mpos 清 0
    commandCheckHandler("ZAux_Direct_SetMpos", ret);

    ZAux_Direct_SetOp(handle,0,0);//初始化输出口 0 的状态
    commandCheckHandler("ZAux_Direct_SetOp",ret);

    float TableValue[12]={100,100,102,102,200,200,202,202,300,300,302,302};//规划位
数输出点,每 2 个 table 存储一个点
    ret =ZAux_Direct_SetTable(handle,10,12,TableValue );//设置 Table 值
    commandCheckHandler("ZAux_Direct_SetTable",ret);

    ret = ZAux_Direct_HwPswitch2(handle,0,2,0,0,0,0,0,0);//停止并删除没有完成的比较
点,预防之前有未完成的比较点没删除
    commandCheckHandler("ZAux_Direct_HwPswitch2",ret);
    ret = ZAux_Direct_HwPswitch2(handle,1,2,0,0,0,0,0,0);//停止并删除没有完成的比较
点,预防之前有未完成的比较点没删除
    commandCheckHandler("ZAux_Direct_HwPswitch2",ret);

    ret = ZAux_Direct_HwPswitch2_2D(handle,0,25,0,1,10,6,10);
    commandCheckHandler("ZAux_Direct_HwPswitch2_2D",ret);

    int iaxislist[2]={0,1};
    float dposlist[2]={350,350};

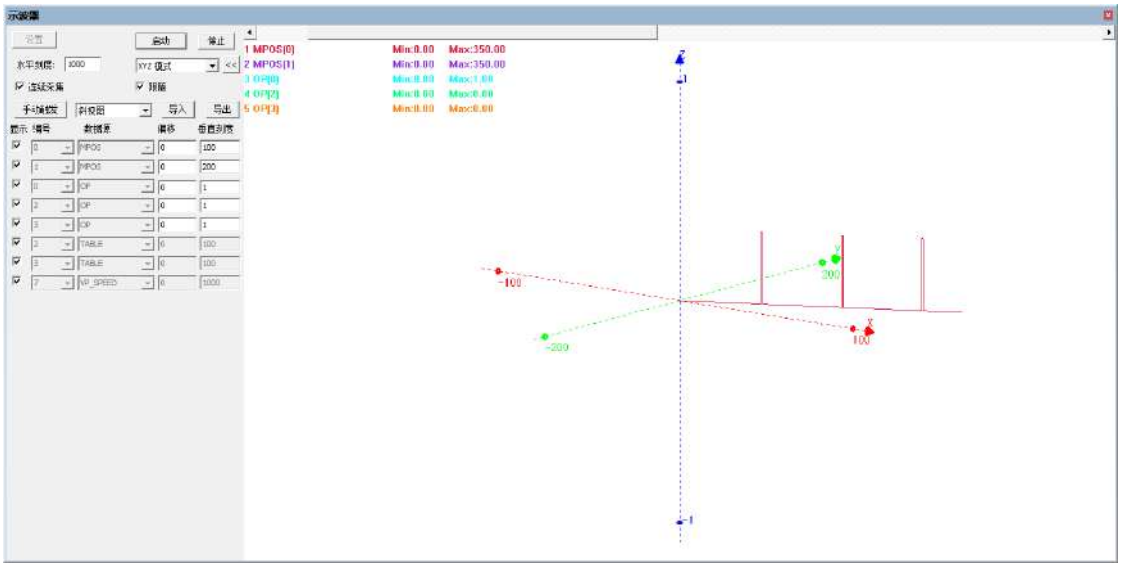
    ZAux_Trigger(handle);//启动示波器
    ret = ZAux_Direct_Move( handle,2,iaxislist , dposlist);//轴 0 正向运动
    commandCheckHandler("ZAux_Direct_Move", ret);

    Sleep(15000);
```

```
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");

    handle = NULL;
    return 0;
}
```

输出结果波形：



4.1.3-5 矢量位置比较输出

```
// test1.cpp : 定义控制台应用程序的入口点。
//需物理接线，把OUT0接入IN0方可显示出,例程波形

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
#include <math.h>
void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非0则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
        Sleep(2000);
        exit(0);
    }
```

```
}

int _tmain(int argc, _TCHAR* argv[])
{
    //char *ip_addr = (char *)"127.0.0.1";              //仿真器IP地址
    char *ip_addr = (char *)"192.168.0.147";              //实物控制器IP地址默认
192.168.0.11
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("控制器连接成功! \n");


    int iaxis[2] = {0,1};
    float DposValue = 0;
    float MposValue = 0;
    int AType = 1;                              //设置轴的类型
    int UnitValue = 100;                        //设置脉冲当量的值
    int SpeedValue = 50;                        //设置速度的值
    int AccelValue = 2000;                      //设置加速度的值
    int DecelValue = 2000;                      //设置减速度的值

    ret = ZAux_Direct_SetAtype(handle, 0, AType);         //设置轴0的类型
    commandCheckHandler("ZAux_Direct_SetAtype", ret);
    ZAux_Direct_SetUnits(handle, 0, UnitValue);       //设置轴0的脉冲当量
    commandCheckHandler("ZAux_Direct_SetUnits", ret);
    ZAux_Direct_SetSpeed(handle, 0,SpeedValue);       //设置轴0的速度的值
    commandCheckHandler("ZAux_Direct_SetSpeed", ret);
    ZAux_Direct_SetAccel(handle, 0, AccelValue);      //设置轴0的加速度的值
    commandCheckHandler("ZAux_Direct_SetAccel", ret);
    ZAux_Direct_SetDecel(handle, 0, DecelValue);      //设置轴0的减速度的值
    commandCheckHandler("ZAux_Direct_SetDecel", ret);
    ret = ZAux_Direct_SetDpos( handle, 0, DposValue);//轴0 Dpos清0
    commandCheckHandler("ZAux_Direct_SetDpos", ret);
    ret = ZAux_Direct_SetMpos( handle, 0, MposValue);//轴0 Mpos清0
    commandCheckHandler("ZAux_Direct_SetMpos", ret);

    ret = ZAux_Direct_SetAtype(handle, 1, AType);         //设置轴1的类型
```

```
    commandCheckHandler("ZAux_Direct_SetAtype", ret);
    ZAux_Direct_SetUnits(handle, 1, UnitValue);      //设置轴1的脉冲当量
    commandCheckHandler("ZAux_Direct_SetUnits", ret);
    ZAux_Direct_SetSpeed(handle, 1,SpeedValue);      //设置轴1的速度的值
    commandCheckHandler("ZAux_Direct_SetSpeed", ret);
    ZAux_Direct_SetAccel(handle, 1, AccelValue);     //设置轴1的加速度的值
    commandCheckHandler("ZAux_Direct_SetAccel", ret);
    ZAux_Direct_SetDecel(handle, 1, DecelValue);     //设置轴1的减速度的值
    commandCheckHandler("ZAux_Direct_SetDecel", ret);
    ret = ZAux_Direct_SetDpos( handle, 1, DposValue);//轴1 Dpos清0
    commandCheckHandler("ZAux_Direct_SetDpos", ret);
    ret = ZAux_Direct_SetMpos( handle, 1, MposValue);//轴1 Mpos清0
    commandCheckHandler("ZAux_Direct_SetMpos", ret);

    ret = ZAux_Direct_SetParam( handle, "VECTOR_MOVED",0,0);// '设置矢量起始位置
    commandCheckHandler("ZAux_Direct_SetParam", ret);



    float TableValue[10]={100,101,200,201,300,301,400,401,498,499};//规划位数输出点
    ret =ZAux_Direct_SetTable(handle,0,10,TableValue );//设置Table值
    commandCheckHandler("ZAux_Direct_SetTable",ret);

    ret = ZAux_Direct_HwPswitch2(handle,0,2,0,0,0,0,0,0);//停止并删除没有完成的比较
点,预防之前有未完成的比较点没删除
    commandCheckHandler("ZAux_Direct_HwPswitch2",ret);

    ret = ZAux_Direct_HwPswitch2(handle,0,3,0,1,0,10,0,0);//配置位置比较输出,主轴轴
0，模式3，输出口0，输出状态1
    commandCheckHandler("ZAux_Direct_HwPswitch2",ret);



    ZAux_Trigger(handle);//启动示波器
    float pos[2]={300,400};

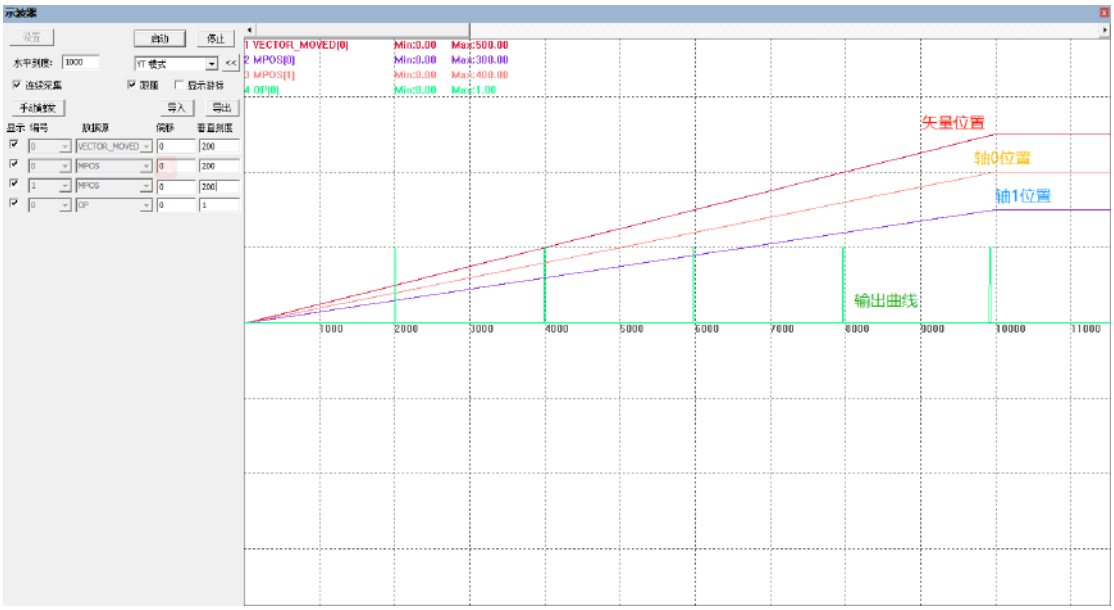
    ret = ZAux_Direct_Move( handle, 2,iaxis,pos);//轴 0 正向运动
    commandCheckHandler("ZAux_Direct_Move", ret);



    Sleep(10000);
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
```

```
    handle = NULL;
    return 0;
}
```

输出波形：



4.1.3-6 周期比较模式（距离复位）

```
// test1.cpp : 定义控制台应用程序的入口点。
// 本例程使用ZMC4系列控制器，且需物理接线，把OUT0接入IN0方可显示出,例程波形


#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
#include <math.h>
void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非0则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
        Sleep(2000);
        exit(0);
    }
```

```
}

int _tmain(int argc, _TCHAR* argv[])
{
    //char *ip_addr = (char *)"127.0.0.1";              //仿真器IP地址
    char *ip_addr = (char *)"192.168.0.11";               //实物控制器IP地址默认
192.168.0.11
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("控制器连接成功! \n");


    int iaxis[2] = {0,1};
    float DposValue = 0;
    float MposValue = 0;
    int AType = 1;                              //设置轴的类型
    int UnitValue = 100;                        //设置脉冲当量的值
    int SpeedValue = 50;                        //设置速度的值
    int AccelValue = 2000;                      //设置加速度的值
    int DecelValue = 2000;                      //设置减速度的值

    ret = ZAux_Direct_SetAtype(handle, 0, AType);         //设置轴0的类型
    commandCheckHandler("ZAux_Direct_SetAtype", ret);
    ZAux_Direct_SetUnits(handle, 0, UnitValue);       //设置轴0的脉冲当量
    commandCheckHandler("ZAux_Direct_SetUnits", ret);
    ZAux_Direct_SetSpeed(handle, 0,SpeedValue);       //设置轴0的速度的值
    commandCheckHandler("ZAux_Direct_SetSpeed", ret);
    ZAux_Direct_SetAccel(handle, 0, AccelValue);      //设置轴0的加速度的值
    commandCheckHandler("ZAux_Direct_SetAccel", ret);
    ZAux_Direct_SetDecel(handle, 0, DecelValue);      //设置轴0的减速度的值
    commandCheckHandler("ZAux_Direct_SetDecel", ret);
    ret = ZAux_Direct_SetDpos( handle, 0, DposValue);//轴0 Dpos清0
    commandCheckHandler("ZAux_Direct_SetDpos", ret);
    ret = ZAux_Direct_SetMpos( handle, 0, MposValue);//轴0 Mpos清0
    commandCheckHandler("ZAux_Direct_SetMpos", ret);

    ret = ZAux_Direct_SetAtype(handle, 1, AType);         //设置轴1的类型
```

```
    commandCheckHandler("ZAux_Direct_SetAtype", ret);
    ZAux_Direct_SetUnits(handle, 1, UnitValue);      //设置轴1的脉冲当量
    commandCheckHandler("ZAux_Direct_SetUnits", ret);
    ZAux_Direct_SetSpeed(handle, 1,SpeedValue);      //设置轴1的速度的值
    commandCheckHandler("ZAux_Direct_SetSpeed", ret);
    ZAux_Direct_SetAccel(handle, 1, AccelValue);     //设置轴1的加速度的值
    commandCheckHandler("ZAux_Direct_SetAccel", ret);
    ZAux_Direct_SetDecel(handle, 1, DecelValue);     //设置轴1的减速度的值
    commandCheckHandler("ZAux_Direct_SetDecel", ret);
    ret = ZAux_Direct_SetDpos( handle, 1, DposValue);//轴1 Dpos清0
    commandCheckHandler("ZAux_Direct_SetDpos", ret);
    ret = ZAux_Direct_SetMpos( handle, 1, MposValue);//轴1 Mpos清0
    commandCheckHandler("ZAux_Direct_SetMpos", ret);

    ret = ZAux_Direct_SetParam( handle, "VECTOR_MOVED",0,0);// '设置矢量起始位置
    commandCheckHandler("ZAux_Direct_SetParam", ret);


    ret = ZAux_Direct_HwPswitch2(handle,0,2,0,0,0,0,0,0);//停止并删除没有完成的比较点,预防之前有未完成的比较点没删除
    commandCheckHandler("ZAux_Direct_HwPswitch2",ret);
    //位置100开始比较，比较4次 周期距离100，输出有效距离20

    ret = ZAux_Direct_HwPswitch2(handle,0,5,0,1,100,4,100,20);//配置位置比较输出,主轴轴0，模式5
    commandCheckHandler("ZAux_Direct_HwPswitch2",ret);

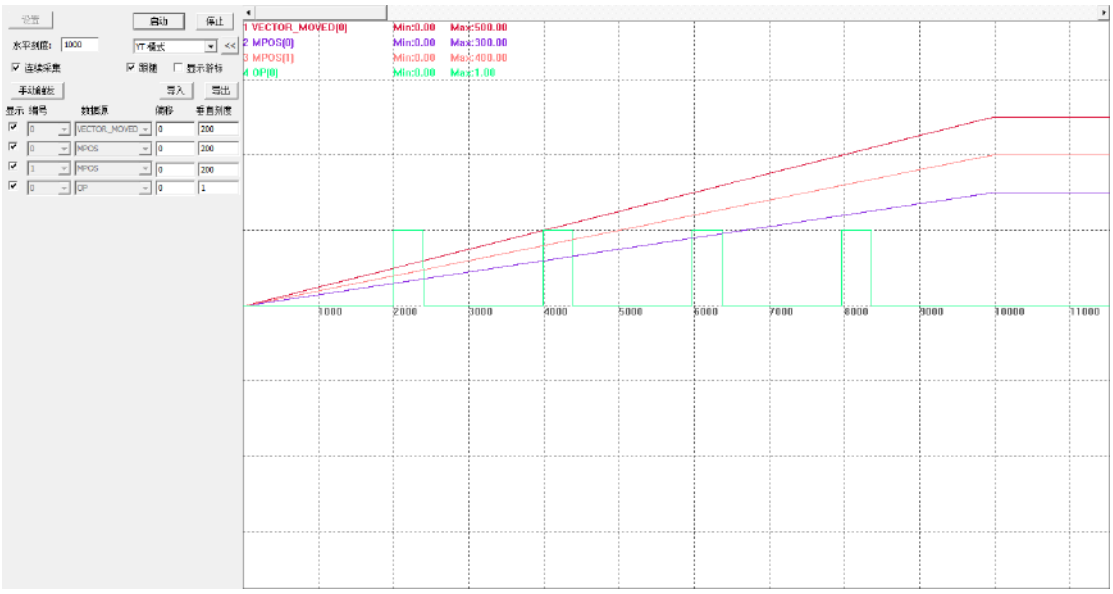

    ZAux_Trigger(handle);//启动示波器
    float pos[2]={300,400};

    ret = ZAux_Direct_Move( handle, 2,iaxis,pos);//轴 0 正向运动
    commandCheckHandler("ZAux_Direct_Move", ret);


    Sleep(10000);
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");

    handle = NULL;
    return 0;
}
```

输出波形：



4.1.3-7 周期比较模式（时间复位）

```
// test1.cpp : 定义控制台应用程序的入口点。
// 本例程使用ZMC4系列控制器，且需物理接线，把OUT0接入IN0方可显示出,例程波形

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
#include <math.h>
void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非0则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
        Sleep(2000);
        exit(0);
    }


}

int _tmain(int argc, _TCHAR* argv[])
{
```

```
    //char *ip_addr = (char *)"127.0.0.1";              //仿真器IP地址
    char *ip_addr = (char *)"192.168.0.147";              //实物控制器IP地址默认
192.168.0.11
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("控制器连接成功! \n");


    int iaxis[2] = {0,1};
    float DposValue = 0;
    float MposValue = 0;
    int AType = 1;                          //设置轴的类型
    int UnitValue = 100;                    //设置脉冲当量的值
    int SpeedValue = 50;                    //设置速度的值
    int AccelValue = 2000;                  //设置加速度的值
    int DecelValue = 2000;                  //设置减速度的值

    ret = ZAux_Direct_SetAtype(handle, 0, AType);         //设置轴0的类型
    commandCheckHandler("ZAux_Direct_SetAtype", ret);
    ZAux_Direct_SetUnits(handle, 0, UnitValue);       //设置轴0的脉冲当量
    commandCheckHandler("ZAux_Direct_SetUnits", ret);
    ZAux_Direct_SetSpeed(handle, 0,SpeedValue);       //设置轴0的速度的值
    commandCheckHandler("ZAux_Direct_SetSpeed", ret);
    ZAux_Direct_SetAccel(handle, 0, AccelValue);      //设置轴0的加速度的值
    commandCheckHandler("ZAux_Direct_SetAccel", ret);
    ZAux_Direct_SetDecel(handle, 0, DecelValue);      //设置轴0的减速度的值
    commandCheckHandler("ZAux_Direct_SetDecel", ret);
    ret = ZAux_Direct_SetDpos( handle, 0, DposValue);//轴0 Dpos清0
    commandCheckHandler("ZAux_Direct_SetDpos", ret);
    ret = ZAux_Direct_SetMpos( handle, 0, MposValue);//轴0 Mpos清0
    commandCheckHandler("ZAux_Direct_SetMpos", ret);

    ret = ZAux_Direct_SetAtype(handle, 1, AType);         //设置轴1的类型
    commandCheckHandler("ZAux_Direct_SetAtype", ret);
    ZAux_Direct_SetUnits(handle, 1, UnitValue);       //设置轴1的脉冲当量
    commandCheckHandler("ZAux_Direct_SetUnits", ret);
    ZAux_Direct_SetSpeed(handle, 1,SpeedValue);       //设置轴1的速度的值
```

```
    commandCheckHandler("ZAux_Direct_SetSpeed", ret);
    ZAux_Direct_SetAccel(handle, 1, AccelValue);     //设置轴1的加速度的值
    commandCheckHandler("ZAux_Direct_SetAccel", ret);
    ZAux_Direct_SetDecel(handle, 1, DecelValue);     //设置轴1的减速度的值
    commandCheckHandler("ZAux_Direct_SetDecel", ret);
    ret = ZAux_Direct_SetDpos( handle, 1, DposValue);//轴1 Dpos清0
    commandCheckHandler("ZAux_Direct_SetDpos", ret);
    ret = ZAux_Direct_SetMpos( handle, 1, MposValue);//轴1 Mpos清0
    commandCheckHandler("ZAux_Direct_SetMpos", ret);

    ret = ZAux_Direct_SetParam( handle, "VECTOR_MOVED",0,0);// '设置矢量起始位置
    commandCheckHandler("ZAux_Direct_SetParam", ret);



    float TableValue[10]={100,101,200,201,300,301,400,401,498,499};//规划位数输出点
    ret =ZAux_Direct_SetTable(handle,0,10,TableValue );//设置Table值
    commandCheckHandler("ZAux_Direct_SetTable",ret);

    ret = ZAux_Direct_HwPswitch2(handle,0,2,0,0,0,0,0,0);//停止并删除没有完成的比较点,预防之前有未完成的比较点没删除
    commandCheckHandler("ZAux_Direct_HwPswitch2",ret);

    ret = ZAux_Direct_HwPswitch2(handle,0,6,0,1,100,3,100,0);//配置位置比较输出,主轴0，模式6，输出口0，输出状态1,'位置100开始比较，比较3次周期距离100，输出有效时间由HW_TIMER指令确定
    commandCheckHandler("ZAux_Direct_HwPswitch2",ret);

    ret = ZAux_Direct_HwTimer(handle,2,1000000,500000,1,0,0);//周期1S，输出变为on后500ms变为off
    commandCheckHandler("ZAux_Direct_HwTimer",ret);

    ZAux_Trigger(handle);//启动示波器
    float pos[2]={300,400};

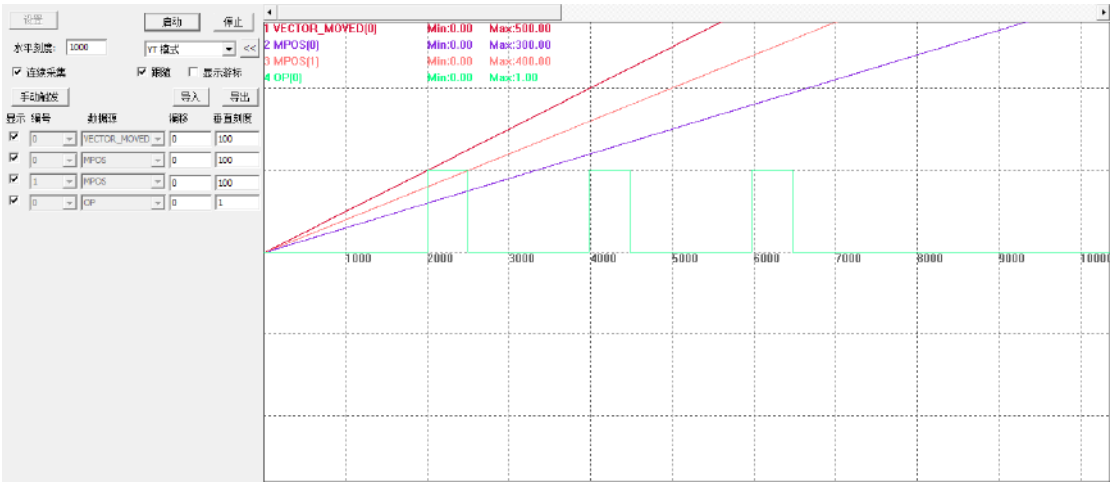


    ret = ZAux_Direct_Move( handle, 2,iaxis,pos);//矢量位置运动500
    commandCheckHandler("ZAux_Direct_Move", ret);

    Sleep(10000);
    ret = ZAux_Close(handle);     //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
```

```
    handle = NULL;
    return 0;
}
```

输出波形图：



第五章 PT 运动

5.1 PT 运动

5.1.1 指令列表

指令	说明
ZAux_Direct_MultiMovePt	多条相对 PT 运动
ZAux_Direct_MultiMovePtAbs	多条绝对 PT 运动
ZAux_Direct_MultiMovePvt	多条相对 PvT 运动
ZAux_Direct_MultiMovePvtAbs	多条绝对 PvT 运动

5.1.2 重点说明

1. 在一段时间内驱动电机运动设置的距离。

2. PT 运动时的加速度，速度，减速度都是根据所设置的时间以及位置所规划的

3. 一般是 PC 每个周期计算好对应的坐标，然后传给控制器。

4. 运动时的速度=(运动距离/时间长度)*1000 units/s。

5. 不要在极短时间运动大距离，脉冲频率会过高，电机堵转，可以分解成小段，重复发送。

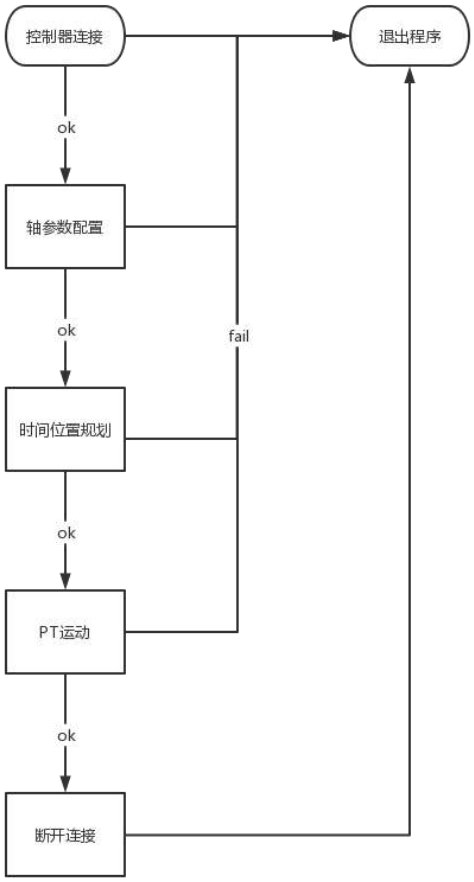
注意：使用该指令时，需记得配置快减减速度或者减速度，否则遇到异常，使用停止运动指令将不会停止。

5.1.3 例程

5.1.3-1 Pt 运动

轴 0 在前 1s 内从-100 运动到 0 的位置，在从 1s 开始轴 0 运动 100，轴 1 运动到-100，最后轴 0 与轴 1 回到绝对位置 0 处。

程序框图：




```
// test1.cpp : 定义控制台应用程序的入口点。
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
#include <math.h>
#define PI 3.1415
void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
        Sleep(2000);
        exit(0);
    }
}
```

```
int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";              //仿真器 IP 地址
    //char *ip_addr = (char *)"192.168.1.11";               //实物控制器 IP 地址默认
192.168.0.11
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("控制器连接成功! \n");

    int iaxis[2] = {0,1};

    float DposValue = 0;
    float MposValue = 0;
    int AType = 1;                           //设置轴的类型
    int UnitValue = 100;                     //设置脉冲当量的值

    ret = ZAux_Direct_SetAtype(handle, iaxis[0], AType);      //设置轴 0 的类型
    commandCheckHandler("ZAux_Direct_SetAtype", ret);
    ZAux_Direct_SetUnits(handle, iaxis[0], UnitValue);        //设置轴 0 的脉冲当量
    commandCheckHandler("ZAux_Direct_SetUnits", ret);
    ZAux_Direct_SetFastDec(handle, iaxis[0], 10000);      //设置快减减速度
    commandCheckHandler("ZAux_Direct_SetFastDec", ret);

    commandCheckHandler("ZAux_Direct_SetDecel", ret);
    ret = ZAux_Direct_SetDpos( handle, iaxis[0],0);  //轴 0 DPOS 设置为-100
    commandCheckHandler("ZAux_Direct_SetDpos", ret);
    ret = ZAux_Direct_SetMpos( handle, iaxis[0],0);  //轴 0 MPOS 设置为-100
    commandCheckHandler("ZAux_Direct_SetMpos12", ret);
    ret = ZAux_Direct_SetAtype(handle, iaxis[1], AType);      //设置轴 1 的类型
```

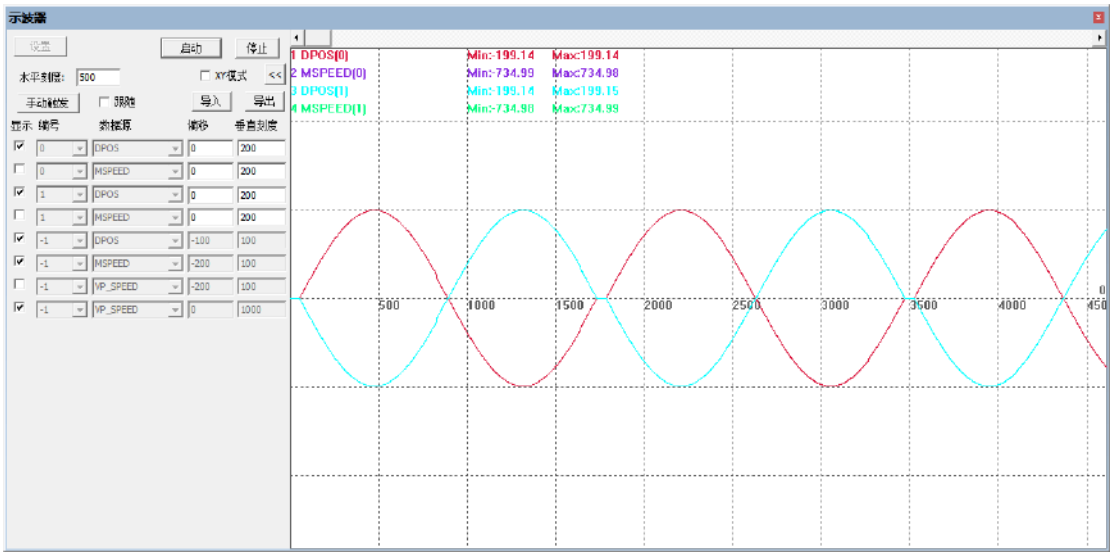
```
    commandCheckHandler("ZAux_Direct_SetAtype", ret);
    ZAux_Direct_SetUnits(handle, iaxis[1], UnitValue);        //设置轴 1 的脉冲当量
    commandCheckHandler("ZAux_Direct_SetUnits", ret);
    ZAux_Direct_SetFastDec(handle, iaxis[1], 10000);      //设置快减减速度
    commandCheckHandler("ZAux_Direct_SetFastDec", ret);

    ret = ZAux_Direct_SetDpos( handle, iaxis[1], DposValue);//轴 1 Dpos 清 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret);
    ret = ZAux_Direct_SetMpos( handle, iaxis[1], MposValue);//轴 1 Mpos 清 0
    commandCheckHandler("ZAux_Direct_SetMpos", ret);

    uint32 Tims[35];//相对绝对 PT 运动时间规划
    float DposList[35];//相对 PT 运动距离规划
    float DposList1[35];//相对 PT 运动距离规划
    int i;
    for ( i=0;i<35;i++)
    {
        Tims[i]=50;
        DposList[i]=(sin(((2*PI)/34)*i))*200;
        DposList1[i]=(cos(((2*PI)/34)*i+0.5*PI))*200;
    }
    printf("DposList[i]=%f\n",DposList[34]);
    ZAux_Trigger(handle);
    for ( i=0;i<3;i++)
    {
        ret =ZAux_Direct_MultiMovePtAbs(handle, 35,1,
iaxis,Tims,DposList);//35*50ms,走一个周期 y=sin(x)的正弦曲线
        commandCheckHandler("ZAux_Direct_MultiMovePtAbs", ret);
        ret =ZAux_Direct_MultiMovePtAbs(handle, 35,1,
&iaxis[1],Tims,DposList1);////35*50ms,走一个周期 y=cos(x+0.5pi)的正弦曲线
        commandCheckHandler("ZAux_Direct_SetMpos", ret);
        Sleep(34*50);//隔 34*50ms 发一次
    }
    Sleep(5000);
    ret = ZAux_Close(handle);     //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");
```

```
    handle = NULL;
    return 0;
}
```

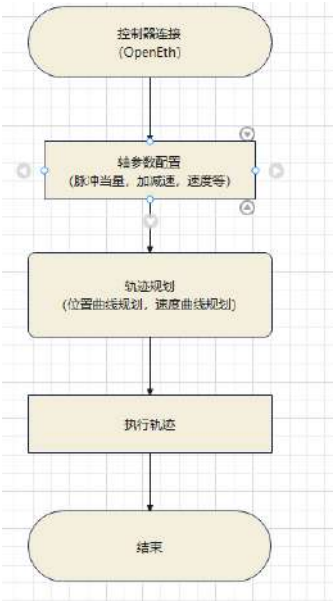
输出波形图：



5.1.3-2 Pvt 运动

例，控制轴 0 在 0-200 内做往复运动，运动位置曲线变化为：x=100*COS(X+π)+100；速度曲线变化为：v=-100*SIN(X+π)。

程序框图如下：




```
// test1.cpp : 定义控制台应用程序的入口点。
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
#include <math.h>
#define PI 3.141592654
void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非0则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
        Sleep(2000);
        exit(0);
    }

}

int _tmain(int argc, _TCHAR* argv[])
```

```
{
    char *ip_addr = (char *)"127.0.0.1";                //仿真器IP地址
    //char *ip_addr = (char *)"192.168.1.11";                //实物控制器IP地址默认
192.168.0.11
    ZMC_HANDLE handle = NULL;                    //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("控制器连接成功! \n");


    int iaxis[2] = {0,1};

    float DposValue = 0;
    float MposValue = 0;
    int AType = 1;                            //设置轴的类型
    int UnitValue = 100;                      //设置脉冲当量的值

    /*******************轴的基本参数配置*********************************************/
    ret = ZAux_Direct_SetAtype(handle, iaxis[0], AType);      //设置轴0的类型
    commandCheckHandler("ZAux_Direct_SetAtype", ret);

    ZAux_Direct_SetUnits(handle, iaxis[0], UnitValue);        //设置轴0的脉冲当量
    commandCheckHandler("ZAux_Direct_SetUnits", ret);

    ZAux_Direct_SetFastDec(handle, iaxis[0], 1000000);        //设置急停速度
    commandCheckHandler("ZAux_Direct_SetFastDec", ret);

    ret = ZAux_Direct_SetDpos( handle, iaxis[0],0); //初始化轴命令位置值为0
    commandCheckHandler("ZAux_Direct_SetDpos", ret);

    ret = ZAux_Direct_SetMpos( handle, iaxis[0],0); //初始化轴反馈位置值为0
    commandCheckHandler("ZAux_Direct_SetMpos12", ret);


    ZAux_Direct_SetMerge(handle,iaxis[0],1);
    commandCheckHandler("ZAux_Direct_SetMerge", ret);

    /*******************运动轨迹规划*********************************************/
```

```
    //  '周期=2π/|ω|
    //  'x=A*COS（ωx+ψ）+C
    //  '求导得斜率，即速度：
    //  'v=-A*ω*SIN（ωx+ψ）


    const int dlen=2*PI/0.01+1;    //根据周期拆分小线段

    uint32 timelist[dlen];
    float DposList[dlen];//Pvt运动距离规划
    float SpeedList[dlen];//Pvt运动速度规划
    int i;
    int temp=0;
    float x1=0.00;
    x1=0.00;
    //  例：假设位置曲线走一个 x=100*COS(X+π)+100的曲线，则速度曲线 v=-
100*SIN(X+π),周期则等于2π
    for ( i=0;i<dlen;i++)
    {

        DposList[i]=100*cos(x1+PI)+100;
        SpeedList[i]=-100*sin(x1+PI);
        timelist[i]=10;
        x1=x1+0.01;
    }
    i=0;

    ZAux_Trigger(handle);
    while(1)
    {
        if ((i+1)*25>dlen)
        {

            ret=ZAux_Direct_MultiMovePvtAbs(handle,dlen-
i*25,1,iaxis,timelist,DposList+25*i,SpeedList+25*i);
            commandCheckHandler("ZAux_Direct_MultiMovePvtAbs", ret);
            i=0;
            temp=temp+1;
            if (temp>2)
            {
                printf("三个周期执行完成！\n");
                break;
            }
        }else
```

```
        {

    ret=ZAux_Direct_MultiMovePvtAbs(handle,25,1,iaxis,timelist,DposList+25*i,SpeedLi
st+25*i);
            commandCheckHandler("ZAux_Direct_MultiMovePvtAbs", ret);
            i=i+1;
        }

    }


    Sleep(5000);
    ret = ZAux_Close(handle);     //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");

    handle = NULL;
    return 0;
}
```

输出效果如下：

位置曲线变化为：x=100*COS(X+π)+100。

速度曲线变化为：v=-100*SIN(X+π)。



第六章 机械手控制运动应用

6.1 机械手模型

6.1.1 指令列表

指令	说明
ZAux_Direct_Connframe	机械手逆解指令
ZAux_Direct_Connreframe	机械手正解指令
ZAux_Direct_FrameRotate	机械手坐标系旋转
ZAux_Frame_Rotate	工件坐标系变换
ZAux_Frame_Rotate2	坐标系变换计算
ZAux_World_DPos	世界坐标系
ZAux_Direct_FrameStatus	机械手轴状态
ZAux_Direct_FrameTrans2	正逆解坐标转换
ZAux_Direct_FrameCal	参数矫正
ZAux_Direct_Mover_C	相对 + 关节轴直接走平面圆弧插补运动
ZAux_Direct_Mover_CABS	绝对 + 关节轴直接走平面圆弧插补运动
ZAux_Direct_Mover_C3	相对 + 关节轴直接走空间圆弧插补运动
ZAux_Direct_Mover_C3ABS	绝对 + 关节轴直接走空间圆弧插补运动
ZAux_Direct_Mover_L	相对 + 关节轴直线插补运动
ZAux_Direct_Mover_LAbs	绝对 + 关节轴直线插补运动

6.1.2 重点说明

机械手有几个电机称为几关节机械手，控制实际电机运动的轴称为机械手的关节轴；控制机械手末端在空间坐标系下直线运动的轴称为虚拟轴。关节轴和虚拟轴通过 **ZAux_Direct_Connreframe** 或 **ZAux_Direct_Connframe** 来指明，只要轴数足够，控制器支持多个机械手。程序可以通过运动指令(所有的运动指令都可以使用)控制关节轴或虚拟轴运动，但同一时刻两种不同类型的轴不能同时运动。当关节轴运动时，虚拟轴需要位于 **ZAux_Direct_Connreframe** 模式，从而自动指向当前的空间坐标；当虚拟轴运动时，关节轴需要位于 **ZAux_Direct_Connframe** 模式，从而 **ZAux_Direct_Connreframe** 和 **ZAux_Direct_Connframe** 在控制器按连接指令对待，也具备运动类型(**ZAux_Direct_GetMtype**)，可以通过运动类型来查看特定轴是否位于对应的模式；通过

ZAux_Direct_Single_Cancel 函数可以取消机械手模式。

逆解运动：

ZAux_Direct_Connframe 对应的运动是逆解运动，此指令作用在关节轴上。此时虚拟轴可以做直线，圆弧，空间圆弧等运动，关节轴在逆解运动的作用下会自动运动到对应的关节轴的位置。

正解运动：

ZAux_Direct_Connreframe 对应的运动是正解运动，此指令作用在虚拟轴上。此时关节轴也可以做各种运动，但实际运动的轨迹不是直线圆弧，这种模式一般用于手动调整关节位置或上点位回零。

逆解运动的三种拐角模式：

逆解模式末端点运动轴的拐角模式 ZAux_Direct_SetCornerMode(ZMC_HANDLE handle, int iaxis, int iValue);	**关节轴的连接速度** ZAux_Direct_SetClutchRate	**运动模式描述**
iValue 值的 BIT7=1	无效	混合模式：虚拟轴的速度、加速度和关节轴的速度、加速度同时生效。 此模式下总是所有的虚拟轴自动参与运动，对虚拟轴的运动叠加无效。 ⚠150718 以上版本支持直线运动，不支持连续插补。 不支持运动中变化加减速度。
iValue 值的 BIT7=0	0	平滑模式：此模式下关节轴使用自己的速度、加速度 做速度规划，高速时轨迹会有变形。
	非 0	强制模式：此模式下关节轴完全按照虚拟轴的速度、加速度进行规划。

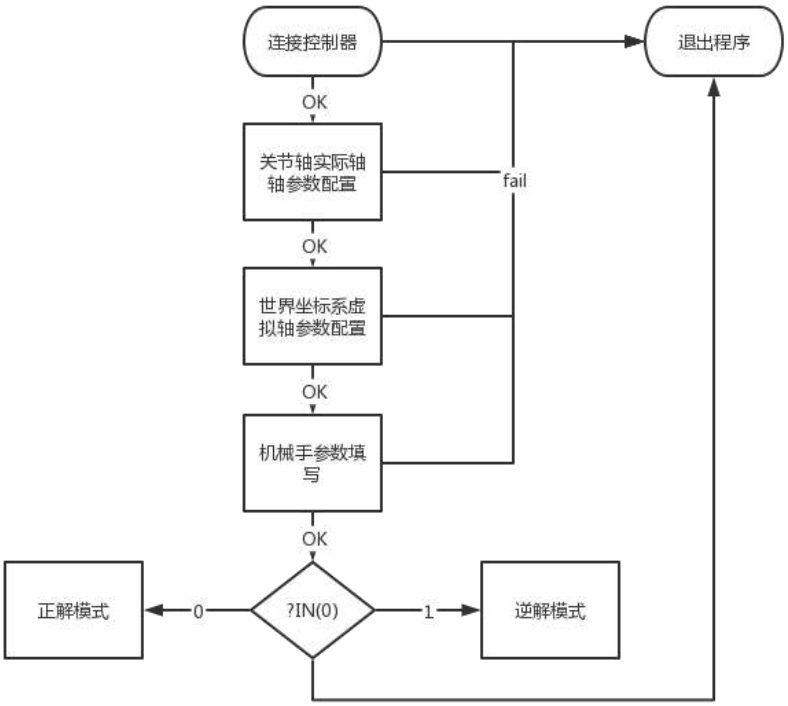
6.1.3 例程

6.1.3-1 机械手模型建立

本例程主要演示了标准 Scara 机械手的例程，通过机械手类型对应的值去修改机械手的类型，机械手的类型请参考手册即可。然后在 RTSys 软件中去手动运动操作对应的虚拟轴

即可。机械手仿真软件可从正运动官网处下载。

程序流程框图：




```
// test1.cpp : 定义控制台应用程序的入口点。
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
#include <math.h>
void commandCheckHandler(const char *command, int ret)
{
    if (ret)//非 0 则失败
    {
        printf("%s fail!return code is %d\n", command, ret);
        Sleep(2000);
        exit(0);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
```

```
    char *ip_addr = (char *)"127.0.0.1";                //仿真器 IP 地址
    //char *ip_addr = (char *)"192.168.1.11";                //实物控制器 IP 地址默认
192.168.0.11
    ZMC_HANDLE handle = NULL;                        //连接句柄
    int ret = ZAux_OpenEth(ip_addr, &handle);    //连接控制器
    if (ERR_SUCCESS != ret)
    {
        printf("控制器连接失败!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("控制器连接成功! \n");

    int JointAxisList[4] = { 0, 1, 2, 3 };//    //定义关节轴的类型

    //设置轴的类型
    ret = ZAux_Direct_SetAtype(handle, JointAxisList[0], 1);
    commandCheckHandler("ZAux_Direct_SetAtype", ret);

    ret = ZAux_Direct_SetAtype(handle, JointAxisList[1], 1);
    commandCheckHandler("ZAux_Direct_SetAtype", ret);

    ret = ZAux_Direct_SetAtype(handle, JointAxisList[2], 1);
    commandCheckHandler("ZAux_Direct_SetAtype", ret);

    ret = ZAux_Direct_SetAtype(handle, JointAxisList[3], 1);
    commandCheckHandler("ZAux_Direct_SetAtype", ret);

    //设置脉冲当量
    ret = ZAux_Direct_SetUnits(handle, JointAxisList[0], 3600 * 2 / 360);
    commandCheckHandler("ZAux_Direct_SetUnits", ret);

    ret = ZAux_Direct_SetUnits(handle, JointAxisList[1], 3600 * 2 / 360);
    commandCheckHandler("ZAux_Direct_SetUnits", ret);

    ret = ZAux_Direct_SetUnits(handle, JointAxisList[2], 3600 * 2 / 360);
```

```
commandCheckHandler("ZAux_Direct_SetUnits", ret);

ret = ZAux_Direct_SetUnits(handle, JointAxisList[3], 3600 * 2 / 1.5);
commandCheckHandler("ZAux_Direct_SetUnits", ret);

//设置 DPOS 的值
ret = ZAux_Direct_SetDpos(handle, JointAxisList[0], 0);
commandCheckHandler("ZAux_Direct_SetUnits", ret);

ret = ZAux_Direct_SetDpos(handle, JointAxisList[1], 0);
commandCheckHandler("ZAux_Direct_SetUnits", ret);

ret = ZAux_Direct_SetDpos(handle, JointAxisList[2], 0);
commandCheckHandler("ZAux_Direct_SetUnits", ret);

ret = ZAux_Direct_SetDpos(handle, JointAxisList[3], 0);
commandCheckHandler("ZAux_Direct_SetUnits", ret);

//设置加速度
ret = ZAux_Direct_SetAccel(handle, JointAxisList[0], 1000);
commandCheckHandler("ZAux_Direct_SetAccel", ret);

ret = ZAux_Direct_SetAccel(handle, JointAxisList[1], 1000);
commandCheckHandler("ZAux_Direct_SetAccel", ret);

ret = ZAux_Direct_SetAccel(handle, JointAxisList[2], 1000);
commandCheckHandler("ZAux_Direct_SetAccel", ret);

ret = ZAux_Direct_SetAccel(handle, JointAxisList[3], 1000);
commandCheckHandler("ZAux_Direct_SetAccel", ret);

//设置减速度
ret = ZAux_Direct_SetDecel(handle, JointAxisList[0], 1000);
commandCheckHandler("ZAux_Direct_SetDecel", ret);

ret = ZAux_Direct_SetDecel(handle, JointAxisList[1], 1000);
commandCheckHandler("ZAux_Direct_SetDecel", ret);
```

```
ret = ZAux_Direct_SetDecel(handle, JointAxisList[2], 1000);
commandCheckHandler("ZAux_Direct_SetDecel", ret);

ret = ZAux_Direct_SetDecel(handle, JointAxisList[3], 1000);
commandCheckHandler("ZAux_Direct_SetDecel", ret);

//设置连接速率
ret = ZAux_Direct_SetClutchRate(handle, JointAxisList[0], 0);
commandCheckHandler("ZAux_Direct_SetClutchRate", ret);

ret = ZAux_Direct_SetClutchRate(handle, JointAxisList[1], 0);
commandCheckHandler("ZAux_Direct_SetClutchRate", ret);

ret = ZAux_Direct_SetClutchRate(handle, JointAxisList[2], 0);
commandCheckHandler("ZAux_Direct_SetClutchRate", ret);

ret = ZAux_Direct_SetClutchRate(handle, JointAxisList[3], 0);
commandCheckHandler("ZAux_Direct_SetClutchRate", ret);

int VirtualAxisList[4] = { 6, 7, 8, 9 };//  //定义虚拟轴的类型
float TableList[7] = { 10, 10, 3600 * 2, 3600 * 2, 3600 * 2, 0, 0 };
float GetTablelist[8];
ret = ZAux_Direct_GetTable(handle, 0, 8, GetTablelist);

//设置轴的类型
ret = ZAux_Direct_SetAtype(handle, VirtualAxisList[0], 0);
commandCheckHandler("ZAux_Direct_SetAtype", ret);

ret = ZAux_Direct_SetAtype(handle, VirtualAxisList[1], 0);
commandCheckHandler("ZAux_Direct_SetAtype", ret);

ret = ZAux_Direct_SetAtype(handle, VirtualAxisList[2], 0);
commandCheckHandler("ZAux_Direct_SetAtype", ret);

ret = ZAux_Direct_SetAtype(handle, VirtualAxisList[3], 0);
commandCheckHandler("ZAux_Direct_SetAtype", ret);
```

```
//设置系统 table 的值
ret = ZAux_Direct_SetTable(handle, 0, 8, TableList);
commandCheckHandler("ZAux_Direct_SetAtype", ret);

//设置脉冲当量
ret = ZAux_Direct_SetUnits(handle, VirtualAxisList[0], 1000);
commandCheckHandler("ZAux_Direct_SetUnits", ret);

ret = ZAux_Direct_SetUnits(handle, VirtualAxisList[1], 1000);
commandCheckHandler("ZAux_Direct_SetUnits", ret);

ret = ZAux_Direct_SetUnits(handle, VirtualAxisList[2], 3600 * 2 / 360);
commandCheckHandler("ZAux_Direct_SetUnits", ret);

ret = ZAux_Direct_SetUnits(handle, VirtualAxisList[3], 1000);
commandCheckHandler("ZAux_Direct_SetUnits", ret);

//设置速度
ret = ZAux_Direct_SetSpeed(handle, VirtualAxisList[0], 200);
commandCheckHandler("ZAux_Direct_SetSpeed", ret);

ret = ZAux_Direct_SetSpeed(handle, VirtualAxisList[1], 200);
commandCheckHandler("ZAux_Direct_SetSpeed", ret);

ret = ZAux_Direct_SetSpeed(handle, VirtualAxisList[2], 200);
commandCheckHandler("ZAux_Direct_SetSpeed", ret);

ret = ZAux_Direct_SetSpeed(handle, VirtualAxisList[3], 200);
commandCheckHandler("ZAux_Direct_SetSpeed", ret);

//设置加速度
ret = ZAux_Direct_SetAccel(handle, VirtualAxisList[0], 1000);
commandCheckHandler("ZAux_Direct_SetAccel", ret);

ret = ZAux_Direct_SetAccel(handle, VirtualAxisList[1], 1000);
commandCheckHandler("ZAux_Direct_SetAccel", ret);
```

```
    ret = ZAux_Direct_SetAccel(handle, VirtualAxisList[2], 1000);
    commandCheckHandler("ZAux_Direct_SetAccel", ret);

    ret = ZAux_Direct_SetAccel(handle, VirtualAxisList[3], 1000);
    commandCheckHandler("ZAux_Direct_SetAccel", ret);

    //设置减速度
    ret = ZAux_Direct_SetDecel(handle, VirtualAxisList[0], 1000);
    commandCheckHandler("ZAux_Direct_SetDecel", ret);

    ret = ZAux_Direct_SetDecel(handle, VirtualAxisList[1], 1000);
    commandCheckHandler("ZAux_Direct_SetDecel", ret);

    ret = ZAux_Direct_SetDecel(handle, VirtualAxisList[2], 1000);
    commandCheckHandler("ZAux_Direct_SetDecel", ret);

    ret = ZAux_Direct_SetDecel(handle, VirtualAxisList[3], 1000);
    commandCheckHandler("ZAux_Direct_SetDecel", ret);

    uint32 GetValue = 0;  //1 表示 In 口打开   0 表示关闭
    int Jogmaxaxises = 4; //关节轴数量
    int frame = 1;        //机械手类型  1-标准 scara
    int Virmaxaxises = 4; //虚拟轴数量
    int  piValue;//定义获取运动缓冲的变量  -1 表示当前没有剩余的运动缓冲  0 表示还有剩余运动缓冲

    //uint32 GetValue = 0;//1 表示 In 口打开   0 表示关闭
    while (1)
    {
        //ZAux_Direct_GetIn(ZMC_HANDLE handle, int ionum, uint32 *piValue); 获取输入口的状态值
        ret = ZAux_Direct_GetIn(handle, 0, &GetValue);//1 表示打开输入口    0 表示输入口关闭
        commandCheckHandler("ZAux_Direct_GetIn", ret);
        //printf("GetValue 1 表示打开输入口  0 表示输入口关闭  GetValue = %d \n", GetValue);
```

```
        if (GetValue == 1)
        {
            ret = ZAux_Direct_Connframe(handle, Jogmaxaxises, JointAxisList, frame,
0, Virmaxaxises, VirtualAxisList);
            commandCheckHandler("ZAux_Direct_Connreframe", ret);
            printf("启动逆解模式\n");
            Sleep(1000);

            ret = ZAux_Direct_GetLoaded(handle, 0, &piValue);
            commandCheckHandler("ZAux_Direct_GetLoaded", ret);
            printf("piValue = %d\n", piValue);
            //printf("piValue 表示当前运动缓冲数 -1 表示没有运动缓冲数  0 表示还有剩
余缓冲数 piValue = %d\n", piValue);
            //0 表示没有运动   -1 表示有运动
            //等待运动缓冲区里面的任务数运行完成
            while (piValue == -1)
            {
                ret = ZAux_Direct_GetLoaded(handle, 0, &piValue);
                commandCheckHandler("ZAux_Direct_GetLoaded", ret);
                printf("piValue = %d\n", piValue);
                Sleep(1000);
                if (piValue == 0)
                {
                    printf("逆解---while 循环 piValue = %d\n ", piValue);
                    break;
                }
            }

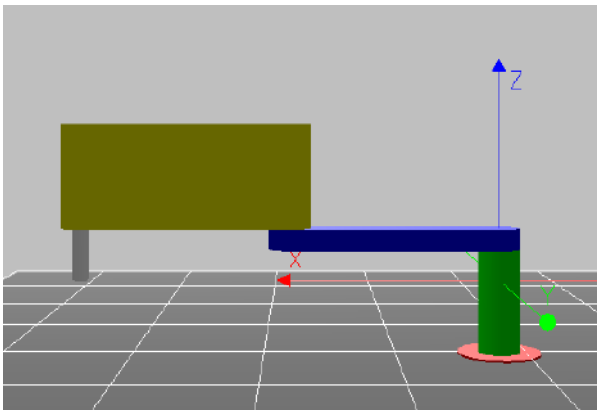
        }
        else if (GetValue == 0)
        {
            ret = ZAux_Direct_Connreframe(handle, Virmaxaxises, VirtualAxisList,
frame, 0, Jogmaxaxises, JointAxisList);
            commandCheckHandler("ZAux_Direct_Connreframe", ret);
            printf("启动正解模式\n");
            Sleep(1000);
```

```
            ret = ZAux_Direct_GetLoaded(handle, 0, &piValue);
            commandCheckHandler("ZAux_Direct_GetLoaded", ret);
            printf("piValue = %d\n", piValue);

            //等待运动缓冲区里面的任务数运行完成
            while (piValue == -1)
            {
                if (piValue == 0)
                {
                    printf("正解---while 循环 piValue = %d\n ", piValue);
                    break;
                }
            }
        }
    }
    ret = ZAux_Close(handle);    //关闭连接
    commandCheckHandler("ZAux_Close", ret) ;//判断指令是否执行成功
    printf("connection closed!\n");

    handle = NULL;
    return 0;
}
```

仿真器模型建立图：



第七章 指令详解

<table>
<tr><td>指令 1</td><td>ZAux_Direct_GetAddax</td></tr>
<tr><td>指令原型</td><td>int32 __stdcall ZAux_Direct_GetAddax (ZMC_HANDLE handle , int iaxis, int * piValue);</td></tr>
<tr><td>指令说明</td><td>读取叠加轴。</td></tr>
<tr><td>输入参数</td><td><table><tr><td>参数名</td><td>描述</td></tr><tr><td>handle</td><td>连接句柄。</td></tr><tr><td>iaxis</td><td>轴号。</td></tr></table></td></tr>
<tr><td>输入参数</td><td><table><tr><td>参数名</td><td>描述</td></tr><tr><td>piValue</td><td>读取轴的叠加轴号。无叠加轴时为-1</td></tr></table></td></tr>
<tr><td>返回值</td><td>成功返回值为 0，非 0 详见错误码说明。</td></tr>
<tr><td>指令示例</td><td>Int iValue;
ZAux_Direct_GetAddax(handle, 0, &iValue);
//读取轴 0 的叠加轴号</td></tr>
<tr><td>详细说明</td><td>转换关系：叠加轴运动距离*叠加轴 UNITS/被叠加轴 UNITS=被叠加轴运动距离。
假设轴 A 的 UNITS 是 100，轴 B 的 UNITS 是 50，叠加轴运动 100
把轴 A 的运动叠加到轴 B，此时轴 A 显示运动了 100，轴 B 运动了 100*100/50=200。
把轴 B 的运动叠加到轴 A，此时轴 B 显示运动了 100，轴 A 运动了 100*50/100=50。
轴之间不能相互同时叠加，A 叠加到 B 后，B 不能再叠加到 A。
支持串联叠加，A 运动叠加到 B，B 在叠加到 C。
支持并联叠加，A 运动同时叠加到 B、C。
叠加时速度从被叠加轴开始变化，加减速按照叠加轴加减速及两轴 units 比例确定。
ADDAX 在轴 MTYPE 为 FRAME 或 REFRAME 的时候不起作用。</td></tr>
</table>

<table>
<tr><td>指令 2</td><td>ZAux_Direct_Single_Addax</td></tr>
<tr><td>指令原型</td><td>int32 __stdcall ZAux_Direct_Single_Addax (ZMC_HANDLE handle , int iaxis, int iaddaxis);</td></tr>
<tr><td>指令说明</td><td>轴叠加运动，iaddaxis 运动叠加到 iaxis,ADDAX 指令叠加的是脉冲个数。</td></tr>
</table>

输入参数	参数名 | 描述 handle | 连接句柄。 iaxis | 被叠加轴号。 iaddaxis | 叠加轴轴号。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	运动叠加
详细说明	转换关系：叠加轴运动距离*叠加轴 UNITS/被叠加轴 UNITS=被叠加轴运动距离。 假设轴 A 的 UNITS 是 100，轴 B 的 UNITS 是 50，叠加轴运动 100 把轴 A 的运动叠加到轴 B，此时轴 A 显示运动了 100，轴 B 运动了 100*100/50=200。 把轴 B 的运动叠加到轴 A，此时轴 B 显示运动了 100，轴 A 运动了 100*50/100=50。 轴之间不能相互同时叠加，A 叠加到 B 后，B 不能再叠加到 A。 支持串联叠加，A 运动叠加到 B，B 在叠加到 C。 支持并联叠加，A 运动同时叠加到 B、C。 叠加时速度从被叠加轴开始变化，加减速按照叠加轴加减速及两轴 units 比例确定。 ADDAX 在轴 MTYPE 为 FRAME 或 REFRAME 的时候不起作用。

指令 3	**ZAux_Direct_Cam**
指令原型	int32 __stdcall ZAux_Direct_Cam(ZMC_HANDLE handle,int iaxis, int istartpoint, int iendpoint, float ftablemulti, float fDistance)
指令说明	电子凸轮，CAM 指令根据存储在 TABLE 中的数据来决定轴的运动。
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。 istartpoint | 起始点 TABLE 编号，存储第一个点的位置。 iendpoint | 结束点 TABLE 编号。 ftablemulti | 位置乘以这个比例，一般为脉冲当量。 fDistance | 参考运动的距离，用来计算总运动时间。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。

指令示例	凸轮表例程
详细说明	注：两个或多个 CAM 指令可以同时使用同一段 TABLE 数据区进行操作。运动的总时间由设置速度和第四个参数决定，运动的实际速度根据 TABLE 轨迹与时间自动匹配。 TBALE 数据需要手动设置，第一个数据为引导点，建议设为 0。 TABLE 数据*table multiplier 这个比例=发出的脉冲数。 请确保指令传递的距离参数*units 是整数个脉冲，否则出现浮点数会导致运动有细微误差。

指令 4	ZAux_Direct_Cambox
指令原型	int32 __stdcall ZAux_Direct_Cambox(ZMC_HANDLE handle,int iaxis,int istartpoint, int iendpoint, float ftablemulti, float fDistance, int ilinkaxis, int ioption, float flinkstartpos)
指令说明	电子凸轮，CAMBOX 指令根据存储在 TABLE 中的数据来决定轴的运动。这些 Table 数据值对应运动轨迹的位置，是相对于运动起始点的距离。跟随轴的运动根据参考轴的运动来进行。

输入参数

参数名	描述
handle	连接标识。
iaxis	轴号。
istartpoint	起始点 TABLE 编号，存储第一个点的位置。
iendpoint	结束点 TABLE 编号。
ftablemulti	位置乘以这个比例，一般为脉冲当量。
fDistance	参考运动的距离，用来计算总运动时间。
Ilinkaxis	参考轴。
ioption	与参考轴的连接方式，不同的二进制位代表不同的意义。 Bit 0 当参考轴锁存信号（Regist）事件触发时，当前轴与参考轴开始进行连接运动。 Bit 1 当参考轴运动到设定的绝对位置时，当前轴与参考轴开始连接运动。 Bit 2 自动重复连续双向运行。(通过设置 REP_OPTION=1，可以取消重复)。 Bit 5 只有参考轴的正向运动才连接。 Bit8 当参考轴 MARKB 信号事件触发时，当前轴与参考轴开始进行连接运动，锁存轴号为参考轴的轴号，需要最新固件支持

	Flinkstartpos	当 ioption 参数 bit 1 置为 1 时，该参数表示连接开始启动的绝对位置。
输出参数	/	
返回值	成功返回值为 0，非 0 详见**错误码**说明。	
指令示例	电子凸轮同步运动（跟随凸轮表）	
详细说明	注：两个或多个 CAMBOX 指令可以同时使用同一段 Table 数据区进行操作。 当前轴运动的总时间由参考轴的运动距离和轴速度确定，速度自动匹配。 Table 数据需要手动设置，第一个数据为引导点，建议设为 0。 Table 数据*table multiplier 这个比例=发出的脉冲数。 请确保指令传递的距离参数脉冲当量是整数个脉冲，否则出现浮点数会导致运动有细微误差。	

指令 5	**ZAux_Direct_Movelink**
指令原型	int32 __stdcall ZAux_Direct_Movelink(ZMC_HANDLE handle, int iaxis, float fDistance, float fLinkDis, float fLinkAcc, float fLinkDec,int iLinkaxis, int ioption, float flinkstartpos)
指令说明	自动凸轮。自定义的凸轮运动，该运动带有可设置的加减速阶段。

输入参数

参数名	描述
handle	连接标识。
iaxis	轴号。
fDistance	从连接开始到结束，当前轴移动的距离，采用用户单位。
fLinkDis	参考轴在连接的整个过程中移动的正向距离，用 units 单位。
fLinkAcc	在当前轴加速阶段，参考轴移动的正向距离，用 units 单位。
fLinkDec	在当前轴减速阶段，参考轴移动的正向距离，用 units 单位。
iLinkaxis	参考轴的轴号。
ioption	参考轴的轴号。 Ioption 连接模式选项，不同的二进制位代表不同的意义。 Bit 0，连接精确开始于参考轴上 regist 事件被触发的时刻。 Bit 1，连接开始于参考轴到达一个绝对位置时。

		Bit 2，当这位被设置时，MOVELINK 会自动重复执行并且可以反向。 Bit 3，当设置时,采用 S 曲线加减速。20170502 以上固件支持 Bit5，只有参考轴为正向运动才连接 Bit8，连接精确开始于参考轴上 MARKB 事件被触发的时刻，需要最新固件支持
	Flinkstartpos	当 ioption 参数 bit1 置为 1 时，该参数表示参考轴在该绝对位置值时，连接开始。
输出参数	/	
返回值	成功返回值为 0，非 0 详见**错误码**说明。	
指令示例	飞剪应用	
详细说明	1.被连接轴为参考轴，连接轴为跟随轴。 2.连接轴的距离分成 3 个阶段应用于参考轴的运动，分别是加速部分、匀速部分和减速部分。 3.在加速和减速阶段为了与速度匹配，基本轴运动距离必须是跟随轴运动距离的两倍。 4.请确保指令传递的距离参数脉冲当量是整数个脉冲，否则出现浮点数会导致运动有细微误差。	

指令 6	**ZAux_Direct_Moveslink**	
指令原型	int32 __stdcall ZAux_Direct_Moveslink(ZMC_HANDLE handle, int iaxis,float fDistance, float fLinkDis, float startsp, float endsp,int iLinkaxis, int ioption, float flinkstartpos)	
指令说明	自动凸轮 2。此指令用于自定义的凸轮运动，该运动自动规划中间曲线，不用计算凸轮表	
输入参数	参数名	描述
	handle	连接标识。
	iaxis	轴号。
	fDistance	从连接开始到结束，当前轴移动的距离，采用用户单位。
	fLinkDis	参考轴在连接的整个过程中移动的正向距离，用 units 单位。
	startsp	启动时跟随轴和参考轴的速度比例，units/units 单位，负数表示跟随轴负向运动
	endsp	结束时跟随轴和参考轴的速度比例，units/units 单位，

		负数表示跟随轴负向运动 注：当 start sp = end sp = fdistance/flink dist 时，匀速运动
	iLinkaxis	参考轴的轴号。
	Ioption	参考轴的轴号。 Ioption 连接模式选项，不同的二进制位代表不同的意义。 Bit0，连接精确开始于参考轴上 regist 事件被触发的时刻。 Bit1，连接开始于参考轴到达一个绝对位置时。 Bit2，当这位被设置时，MOVELINK 会自动重复执行并且可以反向。 Bit5，只有参考轴为正向运动才连接 Bit8，连接精确开始于参考轴上 MARKB 事件被触发的时刻，需要最新固件支持
	Flinkstartpos	当 ioption 参数 bit1 置为 1 时，该参数表示参考轴在该绝对位置值时，连接开始。
输出参数	/	
返回值	成功返回值为 0，非 0 详见**错误码**说明。	
指令示例	电子凸轮同步运动	
详细说明	1.被连接轴为参考轴，连接轴为跟随轴。 2.连接轴的距离分成 3 个阶段应用于参考轴的运动，分别是加速部分、匀速部分和减速部分。 3.在加速和减速阶段为了与速度匹配，基本轴运动距离必须是跟随轴运动距离的两倍。 4.请确保指令传递的距离参数脉冲当量是整数个脉冲，否则出现浮点数会导致运动有细微误差。	

指令 7	**ZAux_Direct_Regist**	
指令原型	int32 __stdcall ZAux_Direct_Regist(ZMC_HANDLE handle,int iaxis, int imode)	
指令说明	单次位置锁存指令。	
输入参数	参数名	描述
	handle	连接标识。
	iaxis	轴号。
	imode	锁存方式 值 描述

1	当 Z 脉冲上升沿时的绝对位置送到 **RegPos**
2	当 Z 脉冲下降沿时的绝对位置送到 **RegPos**
3	当输入信号 R0 上升沿的绝对位置送到 **RegPos**
4	当输入信号 R0 下降沿的绝对位置送到 **RegPos**
6	输入信号 R0 上升沿时的绝对位置送到 **RegPos**，Z 信号上升沿时的绝对位置送到 **RegPosB**
7	输入信号 R0 上升沿时的绝对位置送到 **RegPos**，Z 信号下降沿时的绝对位置送到 **RegPosB**
8	输入信号 R0 下降沿时的绝对位置送到 **RegPos**，Z 信号上升沿时的绝对位置送到 **RegPosB**
9	输入信号 R0 下降沿时的绝对位置送到 **RegPos**，Z 信号下降沿时的绝对位置送到 **RegPosB**
10	输入信号 R0 上升沿时的绝对位置送到 **RegPos**，输入信号 R1 上升沿时的绝对位置送到 **RegPosB**
11	输入信号 R0 上升沿时的绝对位置送到 **RegPos**，输入信号 R1 下降沿时的绝对位置送到 **RegPosB**
12	输入信号 R0 下降沿时的绝对位置送到 **RegPos**，输入信号 R1 上升沿时的绝对位置送到 **RegPosB**
13	输入信号 R0 下降沿时的绝对位置送到 **RegPos**，输入信号 R1 下降沿时的绝对位置送到 **RegPosB**
14	输入信号 R1 上升沿时的绝对位置送到 **RegPosB** (14 以后 150804 以后版本支持，每个锁存通道独立，支持 4 通道锁存)
15	输入信号 R1 下降沿时的绝对位置送到 **RegPosB**
16	Z 信号上升沿时的绝对位置送到 **RegPosB**
17	Z 信号下降沿时的绝对位置送到 **RegPosB**
18	输入信号 R2 上升沿时的绝对位置送到 RegPosC
19	输入信号 R2 下降沿时的绝对位置送到 RegPosC
20	输入信号 R3 上升沿时的绝对位置送到 RegPosD
21	输入信号 R3 下降沿时的绝对位置送到 RegPosD

输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	位置锁存
详细说明	（1）上升下降沿是以控制器内部状态而言。如果设置成上升沿触发，则锁存会在外部输入口由导通状态进入截止状态的一瞬间触发；如果设置成下降沿触发，则锁存会在外部输入口由截止状态进入导通状态的一瞬间触

发；具体要设置成上升沿还是下降沿触发，要根据外部输入口信号跳变的实际需求。

脉冲轴类型一般采用 R0，R1，Z 脉冲这三种锁存；总线轴类型采用 R2，R3 锁存。

（2）支持编码器轴锁存，4 系列及以上控制器最新固件支持虚拟轴、脉冲轴锁存。

（3）EtherCAT 支持驱动器锁存，此时使用驱动器 IO 点实现锁存，具体模式查看指令语法。

（4）Rtex 只支持控制器锁存。

（5）4 系列及以上控制器支持 4 锁存通道。

（6）锁存输入口 432：2 个，432N：4 个，412：8 个。

（7）支持 EtherCAT 驱动器锁存与控制器锁存同时使用，需要有 4 锁存通道功能。

（8）4 个通道指 MARK、MARKB、MARKC，MARKD，通过 REG_INPUTS 指定锁存输入口对应的锁存通道。

（9）当锁存产生时，轴状态 MARK 会被设置为 ON，同时锁存到的位置会被存储在参数 REG_POS 内。

（10）每个轴有输入信号 R0，R1，EZ 信号可以使用锁存功能。当使用两个信号锁存时，第二个信号锁存使用 MARKB 和 REG_POSB。

（11）R0，R1 输入一般对应到输入口 0 和 1，详细请查看控制器的硬件手册中通用输入章节。

指令 8 **ZAux_Direct_CycleRegist**

指令原型

int32 __stdcall ZAux_Direct_CycleRegist(ZMC_HANDLE handle,int iaxis, int imode,int iTabStart,int iTabNum)

指令说明

位置连续锁存。

输入参数

参数名	描述
handle	连接标识。
iaxis	轴号。
imode	见下表

值	描述
101	当 Z 脉冲上升沿时的绝对位置送到 **RegPos**
102	当 Z 脉冲下降沿时的绝对位置送到 **RegPos**
103	当输入信号 R0 上升沿的绝对位置送到 **RegPos**
104	当输入信号 R0 下降沿的绝对位置送到 **RegPos**
114	输入信号 R1 上升沿时的绝对位置送到 **RegPosB**
115	输入信号 R1 下降沿时的绝对位置送到 **RegPosB**

		116	Z 信号上升沿时的绝对位置送到 **RegPosB**
		117	Z 信号下降沿时的绝对位置送到 **RegPosB**
		118	输入信号 R2 上升沿时的绝对位置送到 **RegPosC**
		119	输入信号 R2 下降沿时的绝对位置送到 **RegPosC**
		120	输入信号 R3 上升沿时的绝对位置送到 RegPosD
		121	输入信号 R3 下降沿时的绝对位置送到 **RegPosD**
		123	当输入信号 R0 上升沿的绝对位置送到 **RegPosB**
		124	当输入信号 R0 下降沿的绝对位置送到 **RegPosB**
		133	当输入信号 R0 上升沿的绝对位置送到 **RegPos**，下一次切换下降沿，轮流切换。
		134	当输入信号 R0 下降沿的绝对位置送到 **RegPos**，下一次切换上升沿，轮流切换。
		135	当输入信号 R1 上升沿的绝对位置送到 **RegPosB**，下一次切换下降沿，轮流切换。下一次切换下降沿，轮流切换。
		136	当输入信号 R1 下降沿的绝对位置送到 **RegPosB**，下一次切换上升沿，轮流切换。
	iTabStart	连续锁存的内容存储的 table 位置，第一个 table 元素存储锁存的个数，后面存储锁存的坐标，最多保存个数= numes-1，溢出时循环写入	
	iTabNum	占用的 table 个数	
输出参数	/		
返回值	成功返回值为 0，非 0 详见**错误码**说明。		
指令示例	连续位置锁存		
详细说明	锁存结果存储到 TABLE 里面。 分别对两个通道进行连续锁存，可以实现上下边沿的连续锁存。 ECI：20150829 以上固件支持。 4 系列控制器：20170523 以上固件支持。		

指令 9	**ZAux_Direct_GetCloseWin**
指令原型	int32 __stdcall ZAux_Direct_GetCloseWin(ZMC_HANDLE handle,int iaxis, float * pfValue);
指令说明	读取锁存触发的结束坐标范围点。预留

输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。
输出参数	参数名 | 描述 pfValue | 返回的范围值。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	/
详细说明	预留的功能

指令 10	**ZAux_Direct_SetCloseWin**
指令原型	int32 __stdcall ZAux_Direct_SetCloseWin(ZMC_HANDLE handle,int iaxis, float fValue);
指令说明	设置锁存触发的结束坐标范围点。预留
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。 iValue | 设置的范围值。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	/
详细说明	预留的功能

指令 11	**ZAux_Direct_GetRegPosB**
指令原型	int32 __stdcall ZAux_Direct_GetRegPosB (ZMC_HANDLE handle, int iaxis, float *pfValue);
指令说明	返回锁存寄存器 2（**ZAux_Direct_GetMarkB**）的测量反馈位置(**ZAux_Direct_GetMpos**)，单位:脉冲当量。
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。
输出参数	参数名 | 描述 pfValue | 锁存的坐标位置。

返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	/
详细说明	默认只有编码器可锁存，四系列以上控制器脉冲与虚拟支持锁存

指令 12	**ZAux_Direct_GetRegPos**
指令原型	int32 __stdcall ZAux_Direct_GetRegPos (ZMC_HANDLE handle, int iaxis, float *pfValue);
指令说明	返回锁存寄存器 1（**ZAux_Direct_GetMark**）的测量反馈位置(**ZAux_Direct_GetMpos**)
输入参数	参数名：描述 handle：连接标识。 iaxis：轴号。
输出参数	参数名：描述 pfValue：锁存的坐标位置。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	位置锁存
详细说明	默认只有编码器可锁存，四系列以上控制器脉冲与虚拟支持锁存

指令 13	**ZAux_Direct_GetOpenWin**
指令原型	int32 __stdcall ZAux_Direct_GetOpenWin (ZMC_HANDLE handle, int iaxis, float *pfValue);
指令说明	读取锁存触发的开始坐标范围点。预留
输入参数	参数名：描述 handle：连接标识。 iaxis：轴号。
输出参数	参数名：描述 pfValue：返回的范围值。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	/
详细说明	预留的功能

指令 14	**ZAux_Direct_SetOpenWin**

指令原型	int32 __stdcall ZAux_Direct_SetOpenWin (ZMC_HANDLE handle, int iaxis, float fValue);
指令说明	设置锁存触发的开始坐标范围点。预留
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。 iValue | 设置的范围值。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	/
详细说明	预留的功能

指令 15	**ZAux_Direct_GetMark**
指令原型	iint32 __stdcall ZAux_Direct_GetMark(ZMC_HANDLE handle,int iaxis, int *piValue);
指令说明	读取锁存事件 0 返回状态。
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。
输出参数	参数名 | 描述 piValue | 模式设置 0–没发生 -1-发生
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	位置锁存
详细说明	1.锁存指令执行时，piValue 变真，返回-1。当执行完成时，piValue 变成假，返回 0。 2.对应锁存通道 R0. 3.控制器默认 IN0 对应锁存通道 R0，IN1 对应锁存通道 R1，

指令 16	**ZAux_Direct_GetMarkB**
指令原型	int32 __stdcall ZAux_Direct_GetMarkB(ZMC_HANDLE handle,int iaxis, int *piValue);
指令说明	读取锁存事件 1 返回状态。

输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号。
输出参数	参数名 | 描述 piValue | 模式设置 0 –没发生 -1-发生
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	/
详细说明	1.锁存指令执行时，piValue 变真，返回-1。当执行完成时，piValue 变成假，返回 0。 2.对应锁存通道 R1. 3.控制器默认 IN0 对应锁存通道 R0，IN1 对应锁存通道 R1，

指令 17	**ZAux_Direct_Pswitch**
指令原型	int32 __stdcall ZAux_Direct_Pswitch(ZMC_HANDLE handle,int num, int enable,int axisnum,int outnum,int outstate,float setpos,float resetpos)
指令说明	软件位置比较输出。
输入参数	参数名 | 描述 handle | 连接标识。 num | 软件比较器编号。不同型号，比较器数量不一样。如 ZMC1xx 系列 16 个。 enable | 比较器使能操作 0-关闭 1-打开。 axisnum | 比较输出的运动轴号。 outnum | 操作的输出口编号。 outstate | 输出口状态 1-打开 0-关闭。 setpos | 比较输出的起始位置。 resetpos | 比较输出的结束位置。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	软件位置比较
详细说明	如果多个 PSWITCH 操作同一个输出口，需要编号顺序排在一起。 使用脉冲型电机时只有 ATYPE 为 4 时才是比较反馈位置(MPOS)，默认出厂的 ATYPE 为 1 或 7 比较的是命令位置(DPOS)。

指令 18	**ZAux_Direct_HwPswitch**

<table>
<tr><td>指令原型</td><td>int32 __stdcall ZAux_Direct_HwPswitch(ZMC_HANDLE handle,int Axisnum,int Mode, int Direction, int Reserve, int Tablestart, int Tableend)</td></tr>
<tr><td>指令说明</td><td>硬件位置比较输出，ZMC4 系列产品支持脉冲轴与编码器轴。</td></tr>
<tr><td>输入参数</td><td><table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>handle</td><td>连接标识。</td></tr>
<tr><td>Axisnum</td><td>比较输出的轴号。</td></tr>
<tr><td>Mode</td><td>比较器操作 1-启动比较器 2-停止并删除未完成的点。</td></tr>
<tr><td>Direction</td><td>比较方向 0-负向 1-正向。</td></tr>
<tr><td>Reserve</td><td>预留。</td></tr>
<tr><td>Tablestart</td><td>TABLE 起始地址。</td></tr>
<tr><td>Tableend</td><td>TABLE 结束地址。</td></tr>
</table></td></tr>
<tr><td>输出参数</td><td>/</td></tr>
<tr><td>返回值</td><td>成功返回值为 0，非 0 详见错误码说明。</td></tr>
<tr><td>指令示例</td><td>飞拍应用</td></tr>
<tr><td>详细说明</td><td>（1）HW_PSWITCH 没有比较完所有点的话，一定要设置 mode 值为 2，通过 HW_PSWITCH(2)指令停止并删除没有完成的比较点，否则后面此输出通道会工作不正常。
（2）缺省轴 0-5 分别对应输出口 0 1 2 3 0 1。总共 4 个比较输出口。
（3）可以连续调用两个 HW_PSWITCH 指令，通过函数方式可以获取能调用的指令个数。
（4）每个比较点触发都会使得当前输出口电平翻转。
（5）HW 缓冲数 1024 个，可以连续调用 1024 个 HW 指令。
（6）HW 指令调用后,不受后面的坐标修改功能的影响，HW 指令 TABLE 存储的坐标必须在调用时是正确的，因此尽量手动修改坐标,要规避 HW 指令与坐标循环自动修改的随机冲突。
（7）因为自动循环修改坐标不受程序控制，无法确定是在 HW 的前面还是后面，这样 TABLE 里面的坐标就无法确定。
（8）此指令仅支持脉冲轴硬件位置比较输出，总线轴请使用 HW_PSWITCH2 指令。
（9）使用脉冲型电机时只有 ATYPE 为 4 时才是比较反馈位置(MPOS)，默认出厂的 ATYPE 为 1 或 7 比较的是命令位置(DPOS)。</td></tr>
<tr><td>详细说明 2</td><td>1.触发位置精度可以通过 MOVEOP_DELAY 调整偏差，最小到 us 级别
2.触发频率取决于位置触发点，2 次比较点触发必须要大于 1 个周期</td></tr>
</table>

<table>
<tr><td>指令 19</td><td colspan="2">ZAux_Direct_HwPswitch2</td></tr>
<tr><td>指令原型</td><td colspan="2">int32 __stdcall ZAux_Direct_HwPswitch2(ZMC_HANDLE handle,int Axisnum,int Mode, int Opnum, int Opstate, float ModeParal, float ModePara2, float ModePara3 , float ModePara4)</td></tr>
<tr><td>指令说明</td><td colspan="2">硬件位置比较输出 2，4 系列产品 20170513 以上版本支持。
ZMC306E/306N 支持。</td></tr>
<tr><td>输入参数</td><td colspan="2">
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>handle</td><td>连接标识。</td></tr>
<tr><td>Axisnum</td><td>比较输出的轴号。</td></tr>
<tr><td>Mode</td><td>模式 1-启动比较器
ModePara1 = 第一个比较点坐标所在 TABLE 编号
ModePara2 = 最后一个比较点坐标所在 TABLE 编号
ModePara3= 第一个点判断方向 0-坐标负向 1- 坐标正向 -1- 不使用方向
ModePara4 = 预留

模式 2-停止并删除没完成的比较点
ModePara1 = 预留
ModePara2 = 预留
ModePara3= 预留
ModePara4 = 预留

模式 3-矢量比较方式
ModePara1 = 主轴轴号
ModePara2 = 预留
ModePara3= 预留
ModePara4 = 预留

模式 4-矢量比较方式 单个比较点
ModePara1 = 比较点坐标
ModePara2 = 预留
ModePara3= 预留
ModePara4 = 预留

模式 5-矢量比较方式 周期脉冲模式
ModePara1= 比较点坐标</td></tr>
</table>
</td></tr>
</table>

		ModePara2= 重复周期，一个周期内比较两次，先输出有效状态，在输出无效状态 ModePara3= 周期距离，每个这个距离输出 Opstate,输出有效状态的距离(ModePara4)后还原为无效状态 ModePara4= 输出有效状态的距离，(ModePara1-ModePara4)位无效状态距离 模式 6-矢量比较方式 周期模式，这种模式一般与 HW_TIMER 一起使用 ModePara1 = 比较点坐标 ModePara2 = 重复周期，一个坐标只比较一次 ModePara3= 周期距离，每隔这个距离输出一次 ModePara4 = 预留 模式 8-启动比较器，单轴比较，同 Mode1，但是不翻转信号。可以和 HW_TIMER 搭配使用，模式 8 适配一些需要每个位置触发一定时间的单轴飞拍应用。 ModePara1 = 第一个比较点坐标所在 TABLE 编号 ModePara2 = 最后一个比较点坐标所在 TABLE 编号 ModePara3= 第一个点判断方向 0-坐标负向 1- 坐标正向 -1- 不使用方向 ModePara4 = 预留
	Opnum	输出口编号。4 系列 out 0-3 为硬件位置比较输出。
	Opstate	第一个比较点的输出状态。 0-关闭 1-打开
	ModeParal	多功能参数
	ModePara2	多功能参数。
	ModePara3	多功能参数
	ModePara4	多功能参数
输出参数	/	
返回值	成功返回值为 0，非 0 详见**错误码**说明。	
指令示例	飞拍应用（多个相机绑定一个轴）	
详细说明 1	（1）4 系列产品有 4 个比较输出口，可以选择不同的比较输出口，一般为 OUT0/1/2/3 口。 （2）比较主轴带编码器输入时，自动使用编码器位置来触发，可以使用 MOVEOP_DELAY 参数来调整输出准确时刻。 （3）不同的总线驱动器效果可能有差异，也可以通过 MOVEOP_DELAY	

	参数来调整。 （4）HW_PSWITCH2 与 MOVE_OP 精准使用同样的硬件资源，不建议在同一个通道同时使用，可以在不同的通道同时使用。 （5）每个系统周期内只能比较一次，系统周期通过 SERVO_PERIOD 查询。 （6）TABLE 位置数据在所有比较点完成前不要修改。 （7）脉冲轴和总线轴均支持此指令。 （8）使用脉冲型电机时只有 ATYPE 为 4 时才是比较反馈位置(MPOS)，默认出厂的 ATYPE 为 1 或 7 比较的是命令位置(DPOS)。
详细说明 2	1.触发位置精度可以通过 MOVEOP_DELAY 调整偏差，最小到 us 级别 2.触发频率取决于位置触发点，2 次比较点触发必须要大于 1 个周期

指令 20	ZAux_Direct_GetHwPswitchBuff
指令原型	int32 __stdcall ZAux_Direct_GetHwPswitchBuff(ZMC_HANDLE handle, int axisnum,int *buff)
指令说明	读取硬件位置比较输出剩余缓冲，4 系列产品脉冲轴与编码器轴支持。
输入参数	参数名 | 描述 handle | 连接标识。 Axisnum | 比较输出的轴号。
输出参数	参数名 | 描述 Buff | 剩余缓冲。
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	/
详细说明	/

指令 21	ZAux_Direct_HwTimer
指令原型	int32 __stdcall ZAux_Direct_HwTimer(ZMC_HANDLE handle,int mode, int cyclonetime, int optime, int reptimes, int opstate, int opnum)
指令说明	硬件定时器用于硬件比较输出后一段时间还原电平-4 系列产品支持

输入参数	参数名	描述
	handle	连接标识。
	mode	0 停止，2-启动
	cyclonetime	周期时间 us 单位
	optime	有效时间 us 单位
	reptime	重复次数
	opstate	输出口变为非此状态后开始计时
	opnum	必须能硬件比较输出的口
输出参数	/	
返回值	成功返回值为 0，非 0 详见**错误码**说明。	
指令示例	//周期调整为 2，输出两次 ZAux_Trigger(g_handle); ZAux_Direct_SetOp(g_handle, 0, 0); ZAux_Direct_HwTimer(g_handle, 2, 1000000, 500000, 2, 0, 0);//输出口 0 变为 on 后，硬件定时器触发开始计时，500ms 后切换为 off ZAux_Direct_SetOp(g_handle, 0, 1);	
详细说明	（1）HW_TIMER 只有 1 个，每次调用会强制停止之前的调用。 （2）ZMC420SCAN 每个输出口的 HW_TIMER 功能独立。 （3）ZMC3 系列部分、4 系列及以上产品支持此功能。 （4）OP 和 MOVE_OP 操作会关闭正在进行的 HW_TIMER 脉冲，这样可以使用 HW_TIMER 来实现类似 PWM 的功能，OP 输出打开脉冲输出，下一个 OP 输出关闭脉冲输出，当使用 MOVE_OP 精准输出时，可以实现精准的 PWM 输出无限脉冲功能。	

指令 22	**ZAux_Direct_MultiMovePt**
指令原型	int32 __stdcall ZAux_Direct_MultiMovePt(ZMC_HANDLE handle,int iMoveLen, int imaxaxises, int *piAxislist, uint32 *pTickslist, float *pfDisancelist)
指令说明	在规定 ticks 内驱动电机运动设置的距离。Ticks:时间的伺服周期数，时间=系统周期*ticks 注：SERVO_PERIOD 为 1ms 的控制器 TICKS 为 1ms(不同 SERVO_PERIOD 的 TICKS 时间不一样)

输入参数	参数名 | 描述 handle | 连接标识。 iMoveLen | 填写的运动数量 imaxaxises | 参与运动总轴数 piAxislist | 轴号列表 pTickslist | Ticks 时间列表 pfDisancelist | 运动距离列表
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	Pt 运动
详细说明	一般是 PC 每个周期计算好对应的坐标，然后传给控制器。运动时的速度=(运动距离/时间长度)*1000 units/s。不要在极短时间运动大距离，脉冲频率会过高，电机堵转，可以分解成小段，重复发送。

指令 23	ZAux_Direct_MultiMovePtAbs
指令原型	int32 __stdcall ZAux_Direct_MultiMovePtAbs(ZMC_HANDLE handle, int iMoveLen, int imaxaxises, int *piAxislist, uint32 *pTickslist, float *pfDisancelist)
指令说明	在规定 ticks 内驱动电机运动设置的绝对距离。Ticks:时间长度，大约为 1ms
输入参数	参数名 | 描述 handle | 连接标识。 iMoveLen | 填写的运动数量 imaxaxises | 参与运动总轴数 piAxislist | 轴号列表 pTickslist | Ticks 周期列表 pfDisancelist | 运动距离列表
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	Pt 运动
详细说明	一般是 PC 每个周期计算好对应的坐标，然后传给控制器。运动时的速度=(运动距离/时间长度)*1000 units/s。不要在极短时间运动大距离，脉冲频率会过高，电机堵转，可以分解成小段，重复发送。

指令 24	ZAux_Direct_Connframe

指令原型	int32 __stdcall ZAux_Direct_Connframe(ZMC_HANDLE handle, int Jogmaxaxises, int *JogAxislist, int frame, int tablenum , int Virmaxaxises , int *VirAxislist)
指令说明	机械手逆解指令。
输入参数	参数名 | 描述 handle | 连接标识。 Jogmaxaxises | 关节轴数。 JogAxislist | 关节轴列表。 frame | 机械手类型 1-scara 3-码垛 tablenum | 机械手相关参数 TABLE 存放起点。 Virmaxaxises | 关联的虚拟轴个数。 VirAxislist | 关联的虚拟轴列表。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	机械手的建立
详细说明	连接速度=0 时关节坐标系的运动速度和加速度受 SPEED 等参数的限制。 ⚠当关节轴告警等出错时，此运动会被 CANCEL。 ⚠不要在虚拟轴高速运动中 CANCEL 此运动，会引起轴高速运动中停止。 ⚠此命令 LOAD 时会自动修改虚拟轴的坐标，使得与关节轴一致，因此调用后需要 WAIT LOADED 后才开始运动。 ⚠连接过程中不要修改虚拟轴的坐标，或是调用 DATUM 等可能改坐标的运动，这样会导致关节轴快速运动到新的虚拟位置。 ⚠CONNFRAME 生效后，关节轴的 MTYPE 为 33，此时无法直接运动关节轴，必须通过运行虚拟轴来间接运动关节轴，当要直接移动关节轴时，先调用 CANCEL 取消 CONNFRAME，再运动关节轴。 ⚠当虚拟轴和实际轴都是旋转轴时，例如末端旋转轴，虚拟轴和实际轴的脉冲当量注意要一致。
FRAME 机械结构一览表	详细说明请查看《正运动机械手指令说明》文档。 其他特殊的机械结构如需支持请联系厂家。 frame | 结构类型 1 | 标准 SCARA 机械手 101 | SCARA+摆动，虚拟轴定义 4 个 105 | SCARA+摆动，虚拟轴定义 5 个 106 | 特殊 SCARA 107 | 特殊 SCARA

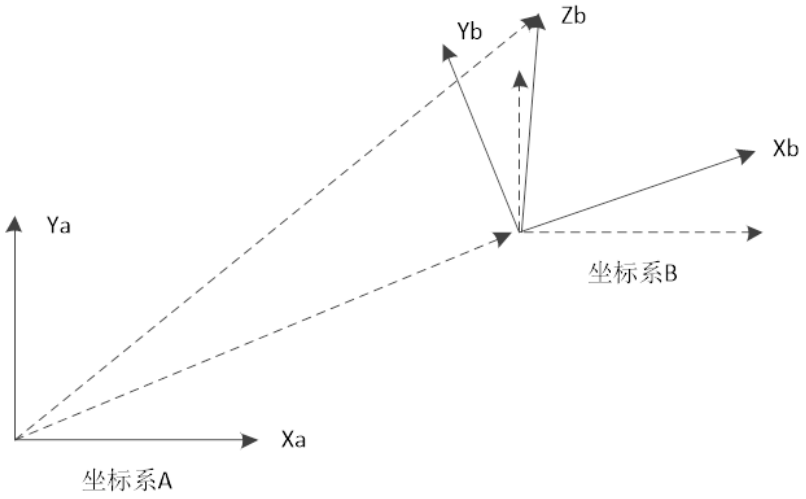
108	特殊 5 轴 SCARA
11	旋转台
17	双旋转台
18	偏移旋转台
19	偏移双旋转台
3	码垛机械手
103	码垛变形，喷涂机械手
5	旋转伸缩机械手
15	XY 滑台
102	2 轴 delta
2	3 轴 delta，R 类型控制器支持
12	4 轴 delta，R 类型控制器支持
13	3 轴垂直蜘蛛手，R 类型控制器支持
25	5 关节机械手
6	6 自由度机械手，R 类型控制器支持
26	特殊 6 自由度
36	特殊 6 自由度
100	XYZ+2 轴手腕，虚拟轴定义 3 个
104	XYZ+2 轴手腕，虚拟轴定义 5 个

指令 25	**ZAux_Direct_Connreframe**
指令原型	int32 __stdcall ZAux_Direct_Connreframe(ZMC_HANDLE handle, int Virmaxaxises, int *VirAxislist, int frame, int tablenum, int Jogmaxaxises, int *JogAxislist)
指令说明	机械手正解指令。
输入参数	参数名 | 描述 handle | 连接标识。 Virmaxaxises | 虚拟轴数量。 VirAxislist | 虚拟轴列表。 frame | 机械手类型 1-scara 3-码垛 tablenum | 机械手相关参数 TABLE 存放起点。 Jogmaxaxises | 关联的关节轴个数。 JogAxislist | 关联的关节轴列表。
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。

指令示例	机械手的建立
详细说明	此指令为逆解运动的反运动指令。 ⚠当虚拟轴 CONNREFRAME 运动 LOAD 时，关节轴的 CONNFRAME 运动会自动被 CANCEL。 ⚠当关节轴的 CONNFRAME 运动 LOAD 时，虚拟轴 CONNREFRAME 运动会自动被 CANCEL。
FRAME 机械结构一览表	详细说明请查看《正运动机械手指令说明》文档。 其他特殊的机械结构如需支持请联系厂家。

frame	结构类型
1	标准 SCARA 机械手
101	SCARA+摆动，虚拟轴定义 4 个
105	SCARA+摆动，虚拟轴定义 5 个
106	特殊 SCARA
107	特殊 SCARA
108	特殊 5 轴 SCARA
11	旋转台
17	双旋转台
18	偏移旋转台
19	偏移双旋转台
3	码垛机械手
103	码垛变形，喷涂机械手
5	旋转伸缩机械手
15	XY 滑台
102	2 轴 delta
2	3 轴 delta，R 类型控制器支持
12	4 轴 delta，R 类型控制器支持
13	3 轴垂直蜘蛛手，R 类型控制器支持
25	5 关节机械手
6	6 自由度机械手，R 类型控制器支持
26	特殊 6 自由度
36	特殊 6 自由度
100	XYZ+2 轴手腕，虚拟轴定义 3 个
104	XYZ+2 轴手腕，虚拟轴定义 5 个

指令 26	**ZAux_Direct_FrameRotate**
指令原型	int32 __stdcall ZAux_Direct_FrameRotate(ZMC_HANDLE handle,int iaxis,

<table>
<tr><td></td><td>float *pfRotatePara)</td></tr>
<tr><td>指令说明</td><td>机械手坐标系旋转，用于对工件坐标系的平移和旋转。</td></tr>
<tr><td>输入参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>handle</td><td>连接标识。</td></tr>
<tr><td>iaxis</td><td>轴号 关节轴/虚拟轴</td></tr>
<tr><td>pfRotateParat[0]</td><td>平移旋转参数: 坐标系 B 沿 X^的平移距离</td></tr>
<tr><td>pfRotateParat[1]</td><td>平移旋转参数: 坐标系 B 沿 Y^的平移距离</td></tr>
<tr><td>pfRotateParat[2]</td><td>平移旋转参数: 坐标系 B 沿 Z^的平移距离</td></tr>
<tr><td>pfRotateParat[4]</td><td>平移旋转参数: 坐标系 B 沿 X^的旋转弧度</td></tr>
<tr><td>pfRotateParat[5]</td><td>平移旋转参数: 坐标系 B 沿 Y^的旋转弧度</td></tr>
<tr><td>pfRotateParat[6]</td><td>平移旋转参数: 坐标系 B 沿 Z^的旋转弧度</td></tr>
</table>
</td></tr>
<tr><td>输出参数</td><td>/</td></tr>
<tr><td>返回值</td><td>成功返回值为 0，非 0 详见错误码说明。</td></tr>
<tr><td>指令示例</td><td>/</td></tr>
<tr><td>详细说明</td><td>目前只对 FRAME6 同时旋转姿态，其他具有 XYZ 虚拟轴的可以支持 XYZ 三个轴旋转，姿态轴不能旋转。
旋转后，虚拟轴 WORLD_DPOS 表示世界坐标不会改变，虚拟轴 DPOS 表示工件坐标会改变。使用时需控制器当前存在机械连接。
BASE 轴可以是虚拟轴与关节轴任意一个；BASE 轴无机械手连接时会报 1025 的错误。
多种机械手叠加的情况，根据 BASE 轴来识别是哪个机械手模式；如果 BASE（axis_1,axis_2）中的 axis_1 是模式 1 的机械手轴，axis_2 是模式 2 的机械手轴，那么结果是模式 1 的机械手进行坐标计算，即以 BASE 轴的顺序来计算。
 
</td></tr>
</table>

<table>
<tr><td>指令 27</td><td colspan="2">ZAux_Direct_MoveSync</td></tr>
<tr><td>指令原型</td><td colspan="2">int32 __stdcall ZAux_Direct_MoveSync(ZMC_HANDLE handle,float imode,int synctime, float syncposition, int syncaxis, int imaxaxises, int *piAxislist, float *pfDisancelist)</td></tr>
<tr><td>指令说明</td><td colspan="2">同步运动，皮带上物体跟随，此运动非插补运动，不保证运动轨迹为直线。</td></tr>
<tr><td>输入参数</td><td>参数名</td><td>描述</td></tr>
<tr><td></td><td>handle</td><td>连接标识。</td></tr>
<tr><td></td><td>imode</td><td>同步模式：
imode =-1:结束模式,运动到指定的绝对位置， 此模式运动如果后面紧接着其它 ZAux_Direct_MoveSync 指令，会被覆盖, 此模式下皮带轴无效。
imode =-2: 强制结束，调用时强制停止原来的 ZAux_Direct_MoveSync，运动到指定结束位置，此模式运动如果后面紧接着其它 ZAux_Direct_MoveSync 指令，会被覆盖, 此模式下皮带轴无效。
imode =0: 跟随轴轴列表第一个轴跟随
imode =10: 跟随轴轴列表第二个轴跟随
imode =20: 跟随轴轴列表第三个轴跟随
imode =0+angle：
皮带旋转角度。angle：皮带旋转角度，角度 = 皮带与跟随轴轴列表第 1/2 轴的正向旋转夹角。例如 imode = PI/4，皮带在 45 度的方向；imode =PI/2，皮带在 y 方向；imode =PI，皮带在 x 负向；imode =(PI*1.75)，皮带在-45 度的方向
imode=1000+tableindex， 轴号列表的跟踪比例依次存储 table 里面, 这样可以实现三维的皮带跟踪, 当皮带有倾斜的时候.(固件版本 20180209 增加此功能)(tableindex:table 寄存器起始编号)</td></tr>
<tr><td></td><td>synctime</td><td>同步时间，ms 单位.运动在指定时间内完成，完成时轴跟皮带轴上物体保持速度一致。0 表示根据运动轴的速度加速度来估计同步时间，可能不准确</td></tr>
<tr><td></td><td>syncposition</td><td>皮带轴物体被感应到时皮带轴的位置。此指令支持皮带轴坐标循环，但是在指令被调用时确保此参数位置和当前皮带轴位置之间没有发生坐标修改或循</td></tr>
</table>

		环操作，因此此指令调用时不要在坐标循环点附近
	syncaxis	皮带轴轴号
	imaxaxises	参与同步从轴总数
	piAxislist	跟随轴轴号列表
	pfDisancelist	皮带轴物体被感应到时从轴的绝对坐标位置
输出参数	/	
返回值	成功返回值为 0，非 0 详见**错误码**说明。	
指令示例	皮带轴同步运动跟随	
详细说明	/	

指令 28	**ZAux_Direct_MultiMovePvt**
指令原型	int32 __stdcall ZAux_Direct_MultiMovePvt(ZMC_HANDLE handle,int iMoveLen, int imaxaxises, int *piAxislist, uint32 *pTickslist, float *pfDisancelist,float *pSpeedlist)
指令说明	在一段时间内驱动电机运动设置的距离，可以指定结束速度，小段内速度会自动根据前面的速度与结束速度来自动规划，尽可能连续。
输入参数	参数名：描述 handle：连接标识。 iMoveLen：填写的运动数量 imaxaxises：参与运动总轴数 piAxislist：轴号列表 pTickslist：Ticks 时间列表 pfDisancelist：运动距离列表 pSpeedlist：对应距离结束后的速度
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	
详细说明	一般是 PC 每个周期计算好对应的坐标，然后传给控制器。不要在极短时间运动大距离，脉冲频率会过高，电机堵转，可以分解成小段，重复发送

指令 29	**ZAux_Frame_Rotate**
指令原型	int32 __stdcall ZAux_Frame_Rotate(ZMC_HANDLE handle,int imaxaxises,int *piAxislist,float distance0,float distance1,float distance2,float Radian0,float Radian1,float Radian2)
指令说明	工件坐标系变换

输入参数	参数名 | 描述 handle | 连接标识。 imaxaxises | 参与运动总轴数 piAxislist | 轴号列表 distance0 | 坐标系 B 沿 X^的平移距离 distance1 | 坐标系 B 沿 Y^的平移距离 distance2 | 坐标系 B 沿 Z^的平移距离 Radian0 | 坐标系 B 沿 X^的旋转的弧度 Radian1 | 坐标系 B 沿 Y^的旋转的弧度 Radian2 | 坐标系 B 沿 Z^的旋转的弧度
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	
详细说明	目前只对 FRAME6 同时旋转姿态，其他具有 XYZ 虚拟轴的可以支持 XYZ 三个轴旋转，姿态轴不能旋转。旋转后，虚拟轴 WORLD_DPOS 表示世界坐标不会改变，虚拟轴 DPOS 表示工件坐标会改变。 使用时需控制器当前存在机械链接。

指令 30	**ZAux_Frame_Rotate2**
指令原型	int32 __stdcall ZAux_Frame_Rotate2(ZMC_HANDLE handle,int *piAxislist,int imaxaxises,int tablein,int tableout,int dir,float *distance,float *radian)
指令说明	坐标系变换计算
输入参数	参数名 | 描述 handle | 连接标识。 piAxislist | 轴列表 imaxaxises | 轴数量 tablein | 转换前，填写的坐标存储 table 位置 tableout | 转换后，输出的坐标存储 table 位置 dir | 0：(DPOS 到 WORLD_DPOS)； 1：(WORLD_DPOS 到 DPOS) distance | 坐标系 B 沿 X^,Y^,Z^的平移距离 radian | 坐标系 B 沿 X^,Y^,Z^的旋转的弧度
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。

指令示例	
详细说明	手动计算坐标系旋转后的坐标值。 使用时需控制器当前存在机械链接。 base 轴，可以是虚拟轴与关节轴中的任意一个；base 轴无机械手链接，会报 1025 的错误。 多种机械手叠加的情况，根据 base 轴来识别是哪个机械手模式。如果 base（axis_1,axis_2）中的 axis_1 是模式 1 的机械手轴，axis_2 是模式 2 的机械手轴，那么结果是模式 1 的机械手进行坐标计算，即以 base 轴的顺序来计算。

指令 31	**ZAux_Direct_FrameStatus**
指令原型	int32 __stdcall ZAux_Direct_FrameStatus(ZMC_HANDLE handle,int axis,int piValue)
指令说明	机械手轴状态
输入参数	参数名 | 描述 handle | 连接标识。 axis | 轴号 piValue | 当前姿态
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	
详细说明	**指明当前的机械手姿态。** 不是机械手状态时返回-1，FRAME_TRANS2 指令会用到这个姿态。只对 SCARA、类 SCARA 和 6 自由度结构有多种姿态。 SCARA 右手姿态值为 0，左手姿态值为 1。

指令 32	**ZAux_Direct_FrameTrans2**
指令原型	int32 __stdcall ZAux_Direct_FrameTrans2(ZMC_HANDLE handle,int *piAxislist,int imaxaxises,int tablein,int tableout,int dirmode)
指令说明	正逆解坐标转换

<table>
<tr><td>输入参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>handle</td><td>连接标识。</td></tr>
<tr><td>piAxislist</td><td>轴列表</td></tr>
<tr><td>imaxaxises</td><td>轴数量</td></tr>
<tr><td>tablein</td><td>table 数组中的索引号，从这个索引开始连续存储数据，正解时输入关节坐标，逆解时输入虚拟轴坐标列表，最后附加姿态</td></tr>
<tr><td>tableout</td><td>table 这个索引开始存储数据，正解时输出虚拟坐标，最后附加姿态，逆解时输出关节坐标列表</td></tr>
<tr><td>dirmode</td><td>模式类型描述:
<table>
<tr><td>模式</td><td>类型</td><td>描述</td></tr>
<tr><td>0</td><td>逆解</td><td>虚拟轴到关节轴，无姿态，自动使用当前的姿态</td></tr>
<tr><td>1</td><td>正解</td><td>关节轴到虚拟轴，无姿态输出</td></tr>
<tr><td>2</td><td>逆解</td><td>输入虚拟轴坐标，最后加上姿态</td></tr>
<tr><td>3</td><td>正解</td><td>输出虚拟轴坐标，输出最后一个位置填姿态</td></tr>
</table>
</td></tr>
</table>
</td></tr>
<tr><td>输出参数</td><td>/</td></tr>
<tr><td>返回值</td><td>成功返回值为 0，非 0 详见错误码说明。</td></tr>
<tr><td>指令示例</td><td></td></tr>
<tr><td>详细说明</td><td>坐标转换函数
使用时必须在正逆解建立的情况下。
使用时必须 base 正确的轴号。逆解时，base 虚拟轴；正解时，base 关节轴。
按照正确的顺序填入相应个数的数据。填入的个数与数据，与?*frame 一致。</td></tr>
</table>

指令 33	**ZAux_Direct_FrameCal**
指令原型	int32 __stdcall ZAux_Direct_FrameCal(ZMC_HANDLE handle,int *piAxislist,int imaxaxises,int tablein,int space,int groups,int tableaux,int zeroout,int tableout2)
指令说明	参数矫正

<table>
<tr><td>输入参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>handle</td><td>连接标识。</td></tr>
<tr><td>piAxislist</td><td>轴列表</td></tr>
<tr><td>imaxaxises</td><td>轴数量</td></tr>
<tr><td>tablein</td><td>关节坐标存储的 table 起始编号，每个点的关节坐标按顺序存储，多个点之间间隔 space</td></tr>
<tr><td>space</td><td>每两个点之间间隔的 table 元素</td></tr>
<tr><td>groups</td><td>点数</td></tr>
<tr><td>tableaux</td><td>辅助参数的 table 编号，部分 frame 需要</td></tr>
<tr><td>zeroout</td><td>计算出来理论零点时关节轴的绝对坐标的 table 编号</td></tr>
<tr><td>tableout2</td><td>计算出来的机械手参数存储的位置，不填时直接存储原参数的位置</td></tr>
</table>
</td></tr>
<tr><td>输出参数</td><td>/</td></tr>
<tr><td>返回值</td><td>成功返回值为 0，非 0 详见错误码说明。</td></tr>
<tr><td>指令示例</td><td></td></tr>
<tr><td>详细说明</td><td>根据机械手关节示教的坐标和特点自动校正当前的机械手参数。

Tablein 中存储的关节轴坐标获取，用当前的零点位置与机械手参数建立机械手链接前提下，控制机械手末端点运动到校正点，取校正点的关节轴坐标。
校正当前零点位置与理论零点的偏差，计算出在理论零点的关节轴坐标。
校正机械手参数的值（校正部分参数），计算出理论的机械参数的值。
FRAME_CAL 只做计算，指令返回值为-1 计算成功，返回 0 计算失败。
FRAME_CAL 的 BASE 轴必须是在 FRAME 状态的轴。</td></tr>
</table>

指令 34	ZAux_Direct_MoveHwPswitch2
指令原型	int32 __stdcall ZAux_Direct_MoveHwPswitch2(ZMC_HANDLE handle,int Axisnum,int Mode, int Opnum , int Opstate, float ModePara1, float ModePara2,float ModePara3,float ModePara4,float HwTimerOpenTime,float HwTimerCycTime,int HwTimerReptimes)
指令说明	硬件位置比较输出，运动缓冲方式。

输入参数	参数名	描述
	handle	连接标识。
	Axisnum	比较输出的轴号。
	Mode	模式 1-启动比较器 ModePara1 = 第一个比较点坐标所在 TABLE 编号 ModePara2 = 最后一个比较点坐标所在 TABLE 编号 ModePara3= 第一个点判断方向 0-坐标负向 1-坐标正向 -1- 不使用方向 ModePara4 = 预留 模式 2-停止并删除没完成的比较点 ModePara1 = 预留 ModePara2 = 预留 ModePara3= 预留 ModePara4 = 预留 模式 3-矢量比较方式 ModePara1 = 主轴轴号 ModePara2 =预留 ModePara3= 预留 ModePara4 = 预留 模式 4-矢量比较方式 单个比较点 ModePara1 = 比较点坐标 ModePara2 = 预留 ModePara3= 预留 ModePara4 = 预留 模式 5-矢量比较方式 周期脉冲模式 ModePara1= 比较点坐标 ModePara2= 重复周期，一个周期内比较两次，先输出有效状态，在输出无效状态 ModePara3= 周期距离，每个这个距离输出 Opstate,输出有效状态的距离(ModePara4)后还原为无效状态 ModePara4=输出有效状态的距离，(ModePara3-

		ModePara4)为无效状态距离 模式 6-矢量比较方式 周期模式，这种模式一把与 HW_TIMER 一起使用 ModePara1 = 比较点坐标 ModePara2 = 重复周期，一个周期只比较一次 ModePara3= 周期距离，每隔这个距离输出一次 ModePara4 = 预留
	Opnum	输出口编号。4 系列 out 0-3 为硬件位置比较输出。
	Opstate	第一个比较点的输出状态。 0-关闭 1-打开
	ModeParal	多功能参数
	ModePara2	多功能参数。
	ModePara3	多功能参数
	ModePara4	多功能参数
	HwTimerOpenTime	有效时间，单位为 us
	HwTimerCycTime	周期时间，单位为 us
	HwTimerReptimes	重复次数
输出参数	/	
返回值	成功返回值为 0，非 0 详见**错误码**说明。	
指令示例		
详细说明		

指令 35	**ZAux_Direct_HwPswitch2_2**
指令原型	int32 ZAux_Direct_HwPswitch2_2(ZMC_HANDLE handle, int *AxisList,int Axisnum,int Mode,int Opnum , int Opstate,int maxerr,int TmodePara1, int TmodePara2,bool ModeFlag, float ModePara1, float ModePara2,float ModePara3)
指令说明	硬件位置比较输出 2，兼容 ZAux_Direct_HwPswitch2。4 系列产品，20170513 以上版本支持。与 hwtimer 并用时，可以动态调整 hwtimer 参数.

输入参数

参数名	描述
handle	连接标识。
AxisList	轴数组
Axisnum	轴数量
Mode	**2D 比较：25，26，每 2 个 table 存储一个点** **3D 比较：35，36，每 3 个 table 存储一个点** 多个点比较，每次输出翻转状态。 **注意：2D、3D 比较模式下 maxerr 偏差参数不能写 0。** **模式 7：与 HW_TIMER 一起使用** 说明：比较点写在 TABLE 中，坐标均参考矢量坐标，每到达一个 TABLE 比较矢量位置触发 OP，此时 OP 的脉冲宽度和每次触发的比较次数由 HW_TIMER 控制；到达下一个 TABLE 位置，OP 再次触发。 **模式 25：2D 比较** 说明：比较点写在 TABLE 中，两个连续的 TABLE 数据组成一个 2D 坐标，每到达一个比较位置 OP 反转一次。示意中蓝色段表示 OP 开启，各类常用插补运动均支持比较，比较点坐标一定的要准确，否则会影响后面点的比较。 **模式 26：2D 比较，与 HW_TIMER 复用** 说明：比较点写在 TABLE 中，两个连续的 TABLE 数据组成一个 2D 坐标，每到达一个比较位置触发 OP，每个比较点 OP 反转的次数和反转周期由 HW_TIMER 设置；到达下一个 TABLE 位置，OP 再次触发。类似模式 7 和模式 36。 **模式 35：3D 比较** 说明：比较点写在 TABLE 中，三个连续的 TABLE 数据组成一个 3D 坐标，每到达一个比较位置 OP 反转一次。示意中蓝色段表示 OP 开启，各类常用插补运动均支持比较，比较点坐标一定的要准确，否则会影响后面点的比较。

		模式 36：3D 比较，与 HW_TIMER 复用 说明：比较点写在 TABLE 中，三个连续的 TABLE 数据组成一个 3D 坐标，每到达一个比较位置触发 OP，每个比较点 OP 反转的次数和反转周期由 HW_TIMER 设置；到达下一个 TABLE 位置，OP 再次触发。类似模式 26 和模式 7
	Opnum	对应的输出口
	Opstate	第一个比较点的输出状态
	maxerr	比较位置每个轴左右的脉冲偏差，进入偏差范围后开始比较
	TmodePara1	7 模式：指第一个比较点 VECTOR_MOVED 坐标所在 TABLE 编号 25，26，35，36 模式：指 TABLE 里面存储的比较点个数
	TmodePara2	7 模式：指最后一个比较点 VECTOR_MOVED 坐标所在 TABLE 编号 25，26，35，36 模式：指第一个比较点坐标所在 TABLE 编号）
	ModeFlag	是否启用动态调整 hwtimer 参数
	ModePara1	脉冲时间
	ModePara2	脉冲个数
	ModePara3	脉冲周期
输出参数	/	
返回值	成功返回值为 0，非 0 详见**错误码**说明。	
指令示例		
详细说明		

指令 36	**ZAux_Direct_SetHwPs2AxisNum**
指令原型	int32 __stdcall ZAux_Direct_SetHwPs2AxisNum(ZMC_HANDLE handle,int iaxis1, int iaxis2)
指令说明	**设置 PS2 轴号** **设置 HW_PSWITCH2 指令实际操作的轴号，缺省-1 表示不修改。** 用于把没有使用的轴的 HW_PSWITCH2 缓冲重复利用起来，指向当前运动的主轴,，可以对当前的主轴同时做多个比较。

输入参数	参数名 | 描述 handle | 连接标识。 iaxis1 | 占用的缓冲轴号 iaxis2 | 绑定的实际操作的轴号
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	
详细说明	

指令 37	ZAux_Direct_GetHwPs2AxisNum
指令原型	int32 __stdcall ZAux_Direct_GetHwPs2AxisNum(ZMC_HANDLE handle,int iaxis1, int* iaxis2)
指令说明	**获得 PS2 轴号** **获取 HW_PSWITCH2 指令实际操作的轴号**
输入参数	参数名 | 描述 handle | 连接标识。 iaxis1 | 占用的缓冲轴号
输出参数	参数名 | 描述 iaxis2 | 绑定的实际操作的轴号
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	
详细说明	

指令 37	ZAux_Direct_GetHwPs2Counts
指令原型	int32 __stdcall ZAux_Direct_GetHwPs2Counts(ZMC_HANDLE handle,int iaxis, int* ivalue)
指令说明	**PS2 比较的点数** **HW_PSWITCH2 指令实际已经比较的点数，HW_PSWITCH2(2)时清 0。**
输入参数	参数名 | 描述 handle | 连接标识。 iaxis | 轴号

输出参数	参数名 | 描述 ivalue | 实际已经比较的点数
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	
详细说明	

指令 38	**ZAux_World_DPos**
指令原型	int32 __stdcall ZAux_World_DPos(ZMC_HANDLE handle,int axis,float*wdpos)
指令说明	**世界坐标系。** 虚拟轴参照世界坐标系的坐标值，没有旋转时与 DPOS 相同。
输入参数	参数名 | 描述 handle | 连接标识。 axis | 轴号
输出参数	参数名 | 描述 wdpos | 对应世界坐标系
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	
详细说明	

指令 39	**ZAux_Direct_Mover_C**
指令原型	int32 __stdcall ZAux_Direct_Mover_C(ZMC_HANDLE handle,int imaxaxises,int *piAxislist, float end1,float end2,float centre1,float centre2,int mode,float distance0, float distance1, float distance2)
指令说明	相对 + 关节轴直接走平面圆弧插补运动

<table>
<tr><td>输入参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>handle</td><td>预留</td></tr>
<tr><td>imaxaxises</td><td>参与运动总轴数</td></tr>
<tr><td>piAxislist</td><td>轴号列表</td></tr>
<tr><td>end1</td><td>第 1 个轴运动距离参数 1，相对坐标</td></tr>
<tr><td>end2</td><td>第 2 个轴运动距离参数 1，相对坐标</td></tr>
<tr><td>centre1</td><td>第 1 个轴运动距离参数 2，相对坐标</td></tr>
<tr><td>centre2</td><td>第 2 个轴运动距离参数 2，相对坐标</td></tr>
<tr><td>mode</td><td>模式
<table>
<tr><td>值</td><td>描述</td></tr>
<tr><td>0</td><td>当前点，中间点，终点三点定圆弧。
距离参数 1 为终点距离，距离参数 2 为中间点距离。</td></tr>
<tr><td>1</td><td>当前点，圆心，终点定圆弧。
走最短的圆弧。
距离参数 1 为终点距离，距离参数 2 为圆心的距离。</td></tr>
<tr><td>2</td><td>当前点，中间点，终点三点定圆。
距离参数 1 为终点距离，距离参数 2 为中间点距离。</td></tr>
<tr><td>3</td><td>当前点，圆心，终点定圆。
先走最短的圆弧，再继续走完整圆。
距离参数 1 为终点距离，距离参数 2 为圆心的距离。</td></tr>
</table></td></tr>
<tr><td>distance0</td><td>第 4 个轴运动距离参数</td></tr>
<tr><td>distance1</td><td>第 5 个轴运动距离参数</td></tr>
<tr><td>distance2</td><td>第 6 个轴运动距离参数</td></tr>
</table></td></tr>
<tr><td>输出参数</td><td>/</td></tr>
<tr><td>返回值</td><td>错误码</td></tr>
<tr><td>指令示例</td><td></td></tr>
<tr><td>详细说明</td><td>/</td></tr>
</table>

指令 40	ZAux_Direct_Mover_CABS
指令原型	int32 __stdcall ZAux_Direct_Mover_CABS(ZMC_HANDLE handle,int imaxaxises,int *piAxislist, float end1,float end2,float centre1,float centre2,int mode,float distance0, float distance1, float distance2)

<table>
<tr><td>指令说明</td><td>绝对 + 关节轴直接走平面圆弧插补运动</td></tr>
<tr><td>输入参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>handle</td><td>预留</td></tr>
<tr><td>imaxaxises</td><td>参与运动总轴数</td></tr>
<tr><td>piAxislist</td><td>轴号列表</td></tr>
<tr><td>end1</td><td>第 1 个轴运动距离参数 1，绝对坐标</td></tr>
<tr><td>end2</td><td>第 2 个轴运动距离参数 1，绝对坐标</td></tr>
<tr><td>centre1</td><td>第 1 个轴运动距离参数 2，绝对坐标</td></tr>
<tr><td>centre2</td><td>第 2 个轴运动距离参数 2，绝对坐标</td></tr>
<tr><td>mode</td><td>模式
<table>
<tr><td>值</td><td>描述</td></tr>
<tr><td>0</td><td>当前点，中间点，终点三点定圆弧。
距离参数 1 为终点距离，距离参数 2 为中间点距离。</td></tr>
<tr><td>1</td><td>当前点，圆心，终点定圆弧。
走最短的圆弧。
距离参数 1 为终点距离，距离参数 2 为圆心的距离。</td></tr>
<tr><td>2</td><td>当前点，中间点，终点三点定圆。
距离参数 1 为终点距离，距离参数 2 为中间点距离。</td></tr>
<tr><td>3</td><td>当前点，圆心，终点定圆。
先走最短的圆弧，再继续走完整圆。
距离参数 1 为终点距离，距离参数 2 为圆心的距离。</td></tr>
</table></td></tr>
<tr><td>distance0</td><td>第 4 个轴运动距离参数</td></tr>
<tr><td>distance1</td><td>第 5 个轴运动距离参数</td></tr>
<tr><td>distance2</td><td>第 6 个轴运动距离参数</td></tr>
</table></td></tr>
<tr><td>输出参数</td><td>/</td></tr>
<tr><td>返回值</td><td>错误码</td></tr>
<tr><td>指令示例</td><td></td></tr>
<tr><td>详细说明</td><td>/</td></tr>
</table>

指令 41	**ZAux_Direct_Mover_C3**
指令原型	int32 __stdcall ZAux_Direct_Mover_C3(ZMC_HANDLE handle,int imaxaxises,int *piAxislist,float end1,float end2,float end3,float centre1,float

<table>
<tr><td></td><td>centre2,float centre3,int mode, float distance0, float distance1, float distance2)</td></tr>
<tr><td>指令说明</td><td>相对 + 关节轴直接走空间圆弧插补运动</td></tr>
<tr><td>输入参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>handle</td><td>预留</td></tr>
<tr><td>imaxaxises</td><td>参与运动总轴数</td></tr>
<tr><td>piAxislist</td><td>轴号列表</td></tr>
<tr><td>end1</td><td>第 1 个轴运动距离参数 1，相对坐标</td></tr>
<tr><td>end2</td><td>第 2 个轴运动距离参数 1，相对坐标</td></tr>
<tr><td>end3</td><td>第 3 个轴运动距离参数 1，相对坐标</td></tr>
<tr><td>centre1</td><td>第 1 个轴运动距离参数 2，相对坐标</td></tr>
<tr><td>centre2</td><td>第 2 个轴运动距离参数 2，相对坐标</td></tr>
<tr><td>Centre3</td><td>第 3 个轴运动距离参数 2，相对坐标</td></tr>
<tr><td>mode</td><td>模式
<table>
<tr><td>值</td><td>描述</td></tr>
<tr><td>0</td><td>当前点，中间点，终点三点定圆弧。
距离参数 1 为终点距离，距离参数 2 为中间点距离。</td></tr>
<tr><td>1</td><td>当前点，圆心，终点定圆弧。
走最短的圆弧。
距离参数 1 为终点距离，距离参数 2 为圆心的距离。</td></tr>
<tr><td>2</td><td>当前点，中间点，终点三点定圆。
距离参数 1 为终点距离，距离参数 2 为中间点距离。</td></tr>
<tr><td>3</td><td>当前点，圆心，终点定圆。
先走最短的圆弧，再继续走完整圆。
距离参数 1 为终点距离，距离参数 2 为圆心的距离。</td></tr>
</table></td></tr>
<tr><td>distance0</td><td>第 4 个轴运动距离参数</td></tr>
<tr><td>distance1</td><td>第 5 个轴运动距离参数</td></tr>
<tr><td>distance2</td><td>第 6 个轴运动距离参数</td></tr>
</table></td></tr>
<tr><td>输出参数</td><td>/</td></tr>
<tr><td>返回值</td><td>错误码</td></tr>
<tr><td>指令示例</td><td></td></tr>
<tr><td>详细说明</td><td>/</td></tr>
</table>

<table>
<tr><td>指令 42</td><td>ZAux_Direct_Mover_C3ABS</td></tr>
<tr><td>指令原型</td><td>int32 __stdcall ZAux_Direct_Mover_C3ABS(ZMC_HANDLE handle,int imaxaxises,int *piAxislist,float end1,float end2,float end3,float centre1,float centre2,float centre3,int mode, float distance0, float distance1, float distance2)</td></tr>
<tr><td>指令说明</td><td>绝对 + 关节轴直接走空间圆弧插补运动</td></tr>
<tr><td>输入参数</td><td>
<table>
<tr><td>参数名</td><td>描述</td></tr>
<tr><td>handle</td><td>预留</td></tr>
<tr><td>imaxaxises</td><td>参与运动总轴数</td></tr>
<tr><td>piAxislist</td><td>轴号列表</td></tr>
<tr><td>end1</td><td>第 1 个轴运动距离参数 1，绝对坐标</td></tr>
<tr><td>end2</td><td>第 2 个轴运动距离参数 1，绝对坐标</td></tr>
<tr><td>end3</td><td>第 3 个轴运动距离参数 1，绝对坐标</td></tr>
<tr><td>centre1</td><td>第 1 个轴运动距离参数 2，绝对坐标</td></tr>
<tr><td>centre2</td><td>第 2 个轴运动距离参数 2，绝对坐标</td></tr>
<tr><td>Centre3</td><td>第 3 个轴运动距离参数 2，绝对坐标</td></tr>
<tr><td>mode</td><td>模式
<table>
<tr><td>值</td><td>描述</td></tr>
<tr><td>0</td><td>当前点，中间点，终点三点定圆弧。
距离参数 1 为终点距离，距离参数 2 为中间点距离。</td></tr>
<tr><td>1</td><td>当前点，圆心，终点定圆弧。
走最短的圆弧。
距离参数 1 为终点距离，距离参数 2 为圆心的距离。</td></tr>
<tr><td>2</td><td>当前点，中间点，终点三点定圆。
距离参数 1 为终点距离，距离参数 2 为中间点距离。</td></tr>
<tr><td>3</td><td>当前点，圆心，终点定圆。
先走最短的圆弧，再继续走完整圆。
距离参数 1 为终点距离，距离参数 2 为圆心的距离。</td></tr>
</table></td></tr>
<tr><td>distance0</td><td>第 4 个轴运动距离参数</td></tr>
<tr><td>distance1</td><td>第 5 个轴运动距离参数</td></tr>
<tr><td>distance2</td><td>第 6 个轴运动距离参数</td></tr>
</table>
</td></tr>
<tr><td>输出参数</td><td>/</td></tr>
</table>

返回值	错误码
指令示例	
详细说明	/

指令 43	ZAux_Direct_Mover_L
指令原型	int32 __stdcall ZAux_Direct_Mover_L(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float *pfDisancelist);
指令说明	相对 + 关节轴直线插补 机械手关节插补运动，机械手末端直线运动到指定坐标。 此指令正解模式下使用，直接操作关节轴时会有改变姿态的可能，所以要确保起点结束点姿态要保持一致，否则会报错。
输入参数	参数名 | 描述 handle | 预留 imaxaxises | 参与运动总轴数 piAxislist | 轴号列表 pfDisancelist | 距离列表
输出参数	/
返回值	错误码
指令示例	
详细说明	/

指令 44	ZAux_Direct_Mover_LAbs
指令原型	int32 __stdcall ZAux_Direct_Mover_LAbs(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float *pfDisancelist);
指令说明	绝对 + 关节轴直线插补 机械手关节插补运动，机械手末端直线运动到指定坐标。 此指令正解模式下使用，直接操作关节轴时会有改变姿态的可能，所以要确保起点结束点姿态要保持一致，否则会报错。
输入参数	参数名 | 描述 handle | 预留 imaxaxises | 参与运动总轴数 piAxislist | 轴号列表 pfDisancelist | 距离列表
输出参数	/
返回值	错误码

指令示例	
详细说明	/

指令 45	**ZAux_Direct_MoveHwTimer**
指令原型	int32__stdcall ZAux_Direct_MoveHwTimer(ZMC_HANDLE handle,int mode,int cyclonetime,int optime,int reptimes,int opstate,int opnum)
指令说明	硬件定时器用于硬件比较输出后一段时间还原电平-4 系列产品支持,-缓冲方式
输入参数	参数名 / 描述 handle / 连接标识。 mode / 0 停止，2-启动 cyclonetime / 周期时间 us 单位 optime / 有效时间 us 单位 reptime / 重复次数 opstate / 输出口变为非此状态后开始计时 opnum / 必须能硬件比较输出的口
输出参数	/
返回值	成功返回值为 0，非 0 详见**错误码**说明。
指令示例	/
详细说明	（1）HW_TIMER 只有 1 个，每次调用会强制停止之前的调用。 （2）ZMC420SCAN 每个输出口的 HW_TIMER 功能独立。 （3）ZMC3 系列部分、4 系列及以上产品支持此功能。 （4）OP 和 MOVE_OP 操作会关闭正在进行的 HW_TIMER 脉冲，这样可以使用 HW_TIMER 来实现类似 PWM 的功能，OP 输出打开脉冲输出，下一个 OP 输出关闭脉冲输出，当使用 MOVE_OP 精准输出时，可以实现精准的 PWM 输出无限脉冲功能。