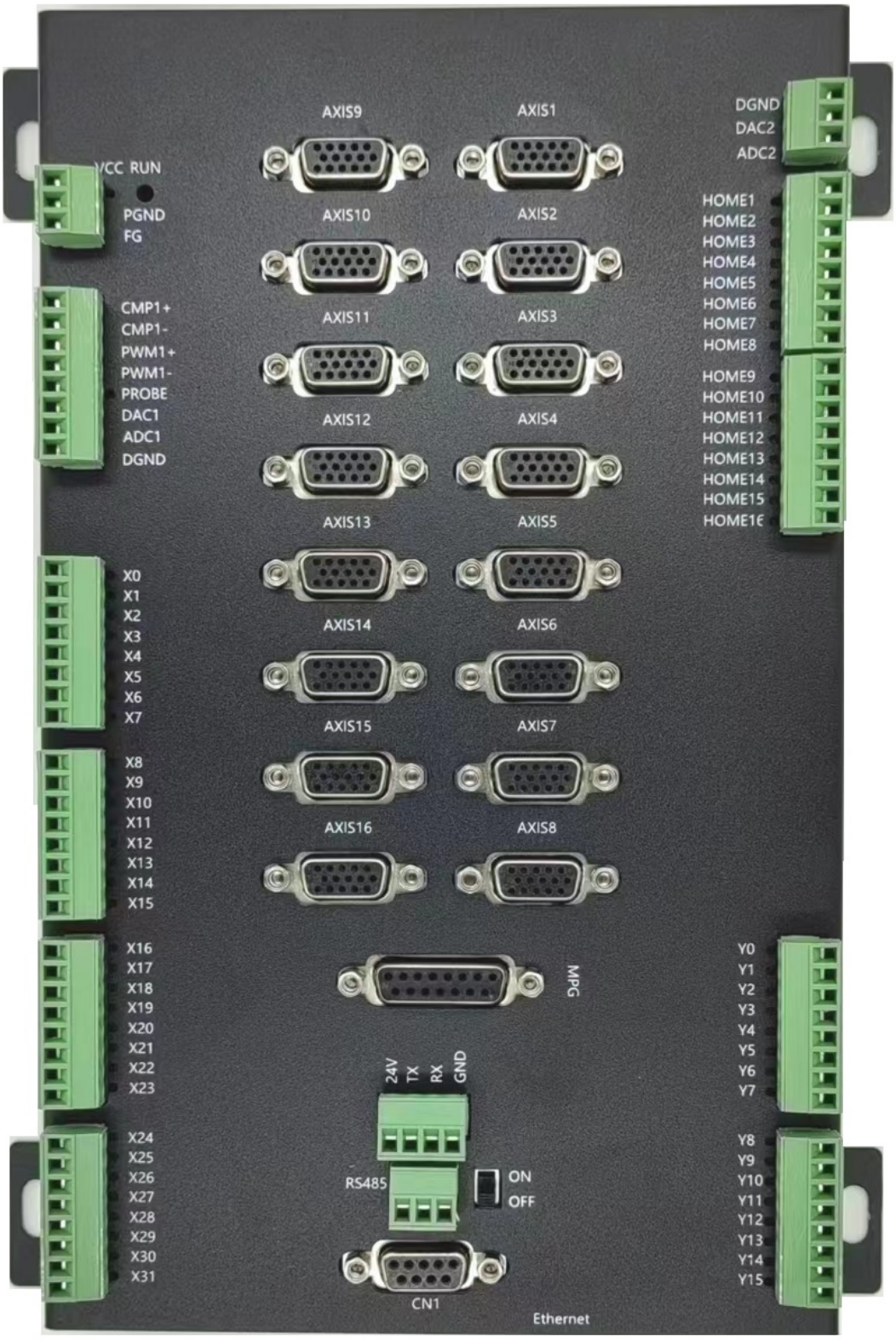
ETH_GAS_N 运动控制卡用户手册 V7.0

联系电话：13113186871

联系邮箱：manyikaimen@163.com



联系电话：13113186871

联系邮箱：manyikaimen@163.com

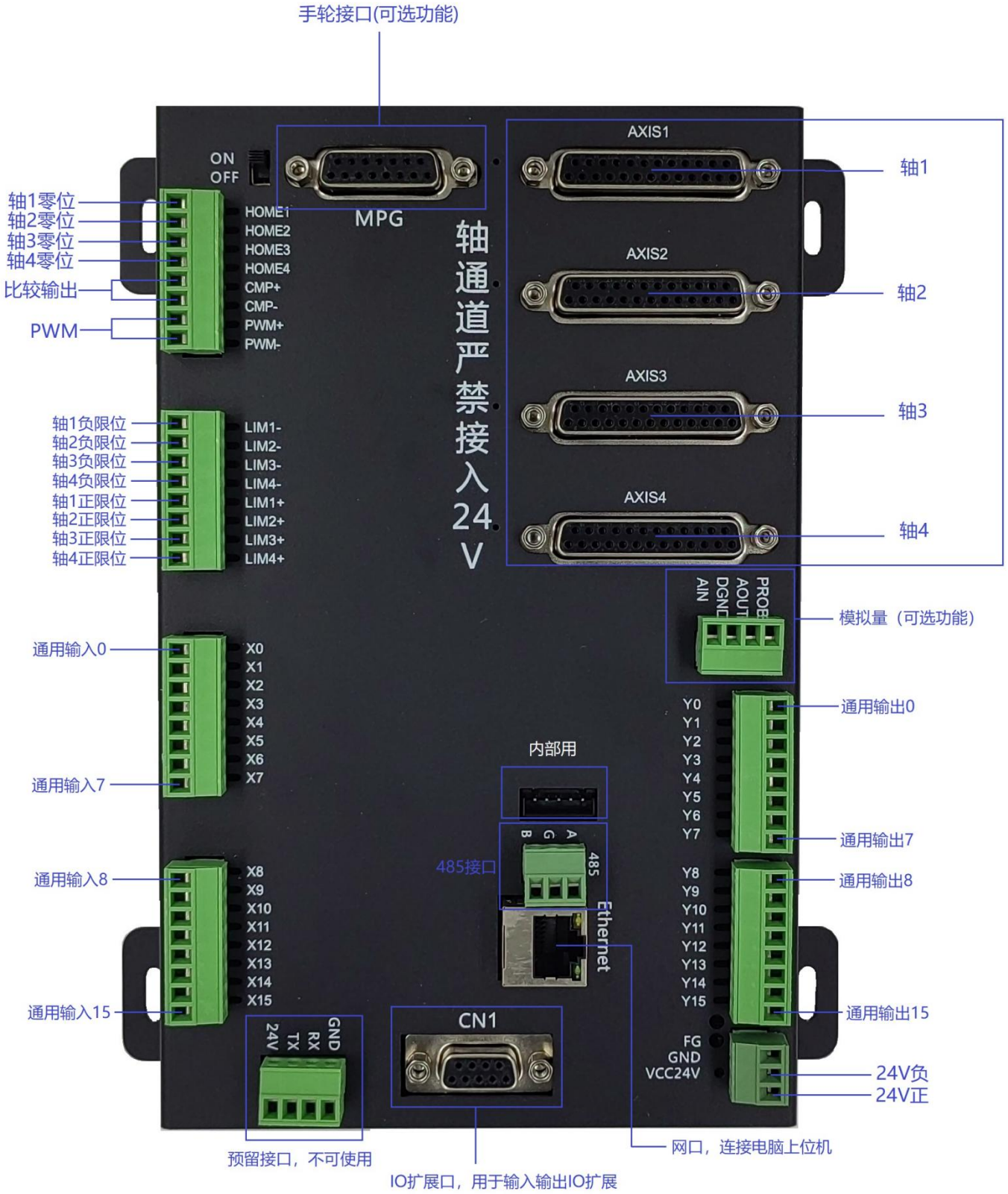
版权申明

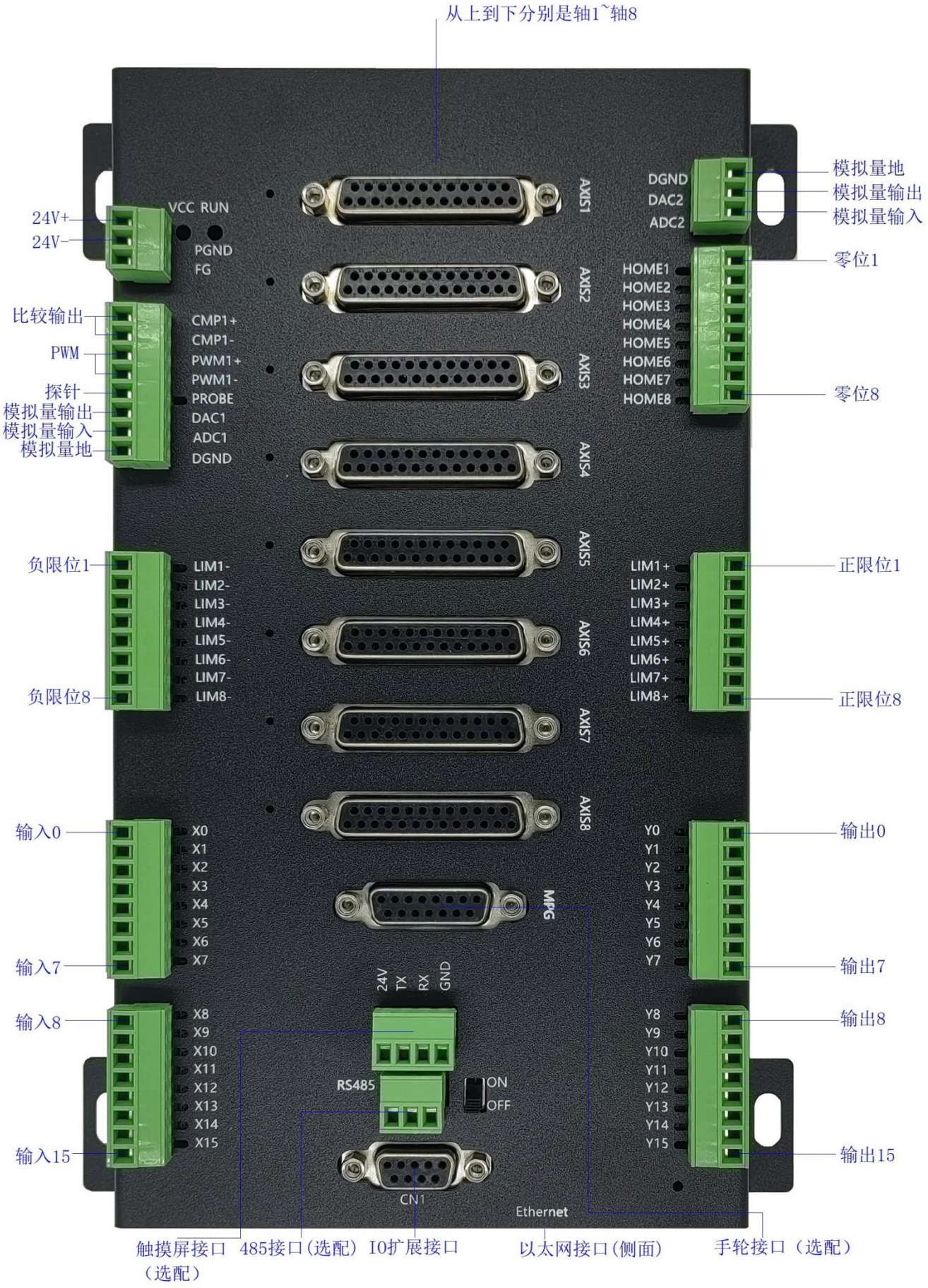
运动中的机器有危险！使用者有责任在机器中设计有效的出错处理和安全保护机制，博派科技没有义务或责任对由此造成的附带的或相应产生的损失负责。

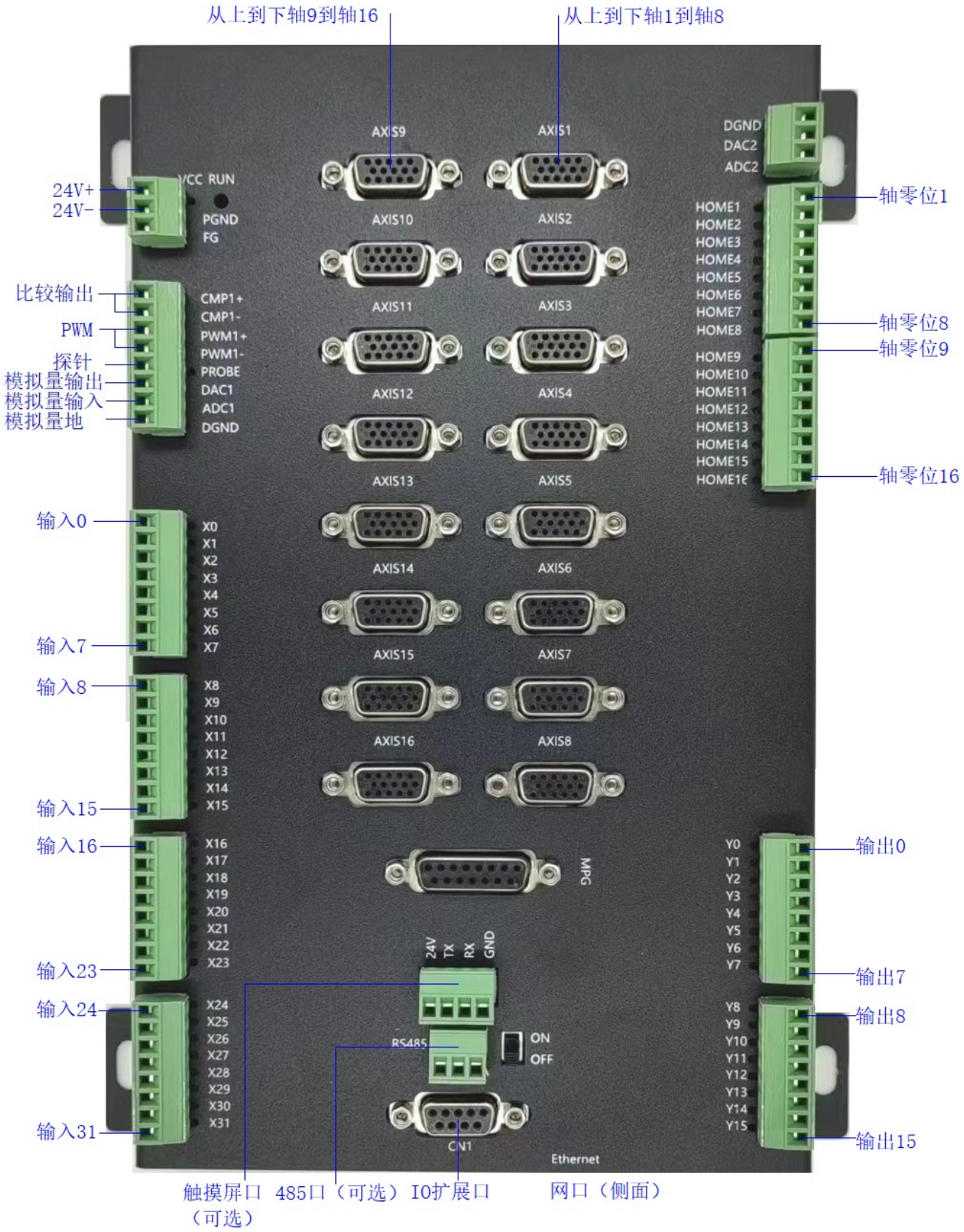
目录

一、硬件资源

1、板卡采用 24V 直流电源供电。
2、控制卡自身有 16 路通用输入，采用光耦隔离，抗干扰能力强。
3、控制卡自身有 16 路通用输出，可直接驱动继电器。
4、控制卡支持 IO 扩展，最大可扩展至 2048 输入/2048 输出。可满足所有应用场合。
5、控制卡有 8 路轴通道。每一路都包含脉冲、方向、正交编码器、Z 相索引、使能、报警、复位
6、控制卡有 8 路零位输入、8 路负硬限位输入、8 路正硬限位输入。
7、支持以太网或者串口编程。
8、脉冲输出最高频率达 2MHz
9、控制卡支持多个并联使用，最多可扩展至 2000 个轴，可满足所有应用场合。
10、控制卡支持点位运动、速度控制、直线、圆弧、连续轨迹插补，支持速度前瞻。硬件捕获、电子齿轮/电子凸轮、位置比较输出。支持 PT 模式与刀向跟随。


手轮接口(可选功能)
ON
OFF
MPG
轴通道严禁接入24V
AXIS1
AXIS2
AXIS3
AXIS4
轴1
轴2
轴3
轴4
轴1零位
轴2零位
轴3零位
轴4零位
比较输出
PWM
HOME1
HOME2
HOME3
HOME4
CMP+
CMP-
PWM+
PWM-
轴1负限位
轴2负限位
轴3负限位
轴4负限位
轴1正限位
轴2正限位
轴3正限位
轴4正限位
LIM1-
LIM2-
LIM3-
LIM4-
LIM1+
LIM2+
LIM3+
LIM4+
PROBE
AOUT
DGND
AIN
模拟量（可选功能）
通用输入0
通用输入7
通用输入8
通用输入15
X0
X1
X2
X3
X4
X5
X6
X7
X8
X9
X10
X11
X12
X13
X14
X15
内部用
485接口
B
G
A
485
Ethernet
Y0
Y1
Y2
Y3
Y4
Y5
Y6
Y7
Y8
Y9
Y10
Y11
Y12
Y13
Y14
Y15
通用输出0
通用输出7
通用输出8
通用输出15
FG
GND
VCC24V
24V负
24V正
24V
TX
RX
GND
CN1
预留接口，不可使用
IO扩展口，用于输入输出IO扩展
网口，连接电脑上位机

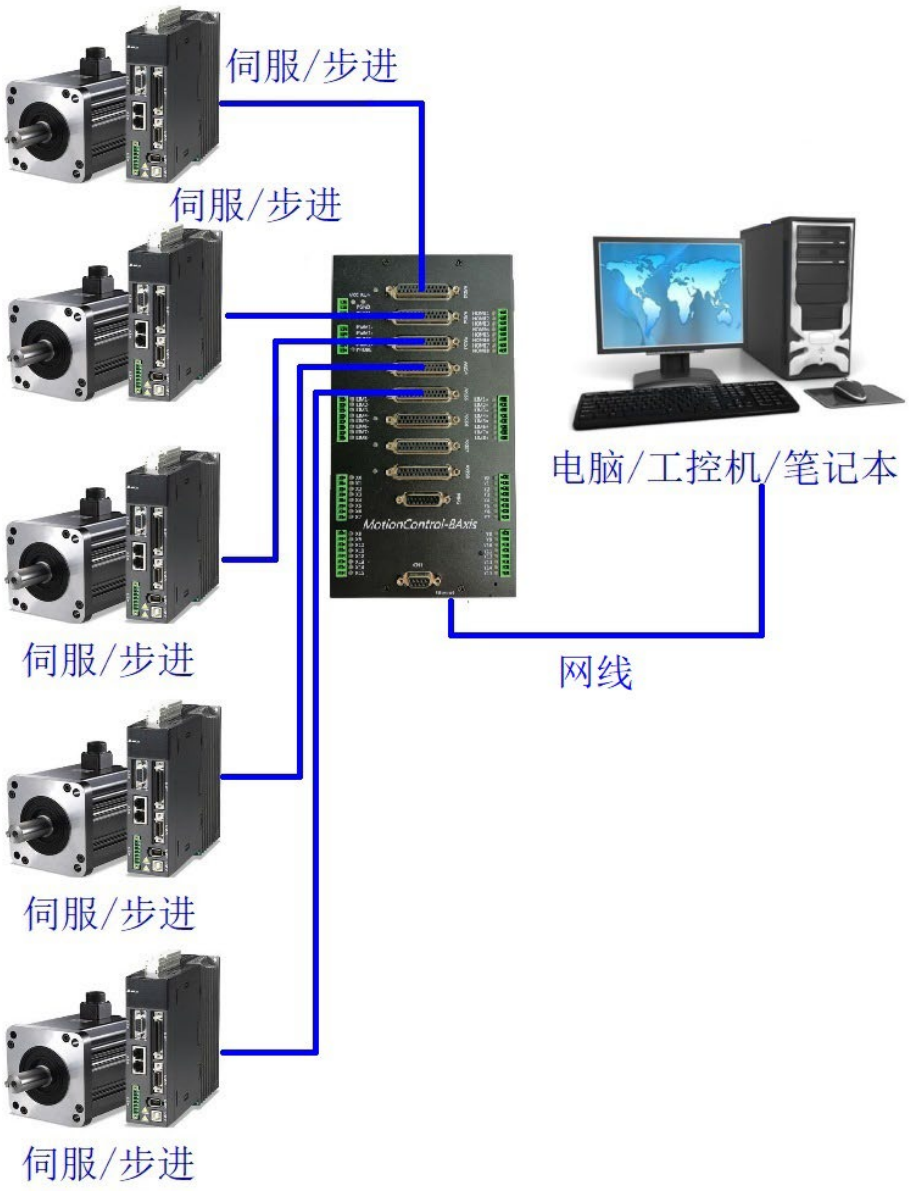

从上到下分别是轴1~轴8
AXIS1
AXIS2
AXIS3
AXIS4
AXIS5
AXIS6
AXIS7
AXIS8
MPG
24V+
24V-
VCC RUN
PGND
FG
比较输出
PWM
探针
模拟量输出
模拟量输入
模拟量地
CMP1+
CMP1-
PWM1+
PWM1-
PROBE
DAC1
ADC1
DGND
DGND
DAC2
ADC2
模拟量地
模拟量输出
模拟量输入
HOME1
HOME2
HOME3
HOME4
HOME5
HOME6
HOME7
HOME8
零位1
零位8
负限位1
负限位8
LIM1-
LIM2-
LIM3-
LIM4-
LIM5-
LIM6-
LIM7-
LIM8-
LIM1+
LIM2+
LIM3+
LIM4+
LIM5+
LIM6+
LIM7+
LIM8+
正限位1
正限位8
输入0
输入7
输入8
输入15
X0
X1
X2
X3
X4
X5
X6
X7
X8
X9
X10
X11
X12
X13
X14
X15
Y0
Y1
Y2
Y3
Y4
Y5
Y6
Y7
Y8
Y9
Y10
Y11
Y12
Y13
Y14
Y15
输出0
输出7
输出8
输出15
24V
TX
RX
GND
RS485
ON
OFF
CN1
Ethernet
触摸屏接口（选配）
485接口(选配)
IO扩展接口
以太网接口(侧面)
手轮接口（选配）


从上到下轴9到轴16
从上到下轴1到轴8
AXIS9
AXIS1
AXIS10
AXIS2
AXIS11
AXIS3
AXIS12
AXIS4
AXIS13
AXIS5
AXIS14
AXIS6
AXIS15
AXIS7
AXIS16
AXIS8
MPG
24V+
24V-
VCC RUN
PGND
FG
比较输出
PWM
探针
模拟量输出
模拟量输入
模拟量地
CMP1+
CMP1-
PWM1+
PWM1-
PROBE
DAC1
ADC1
DGND
输入0
输入7
输入8
输入15
输入16
输入23
输入24
输入31
X0
X1
X2
X3
X4
X5
X6
X7
X8
X9
X10
X11
X12
X13
X14
X15
X16
X17
X18
X19
X20
X21
X22
X23
X24
X25
X26
X27
X28
X29
X30
X31
24V
TX
RX
GND
RS485
ON
OFF
CN1
Ethernet
DGND
DAC2
ADC2
HOME1
HOME2
HOME3
HOME4
HOME5
HOME6
HOME7
HOME8
HOME9
HOME10
HOME11
HOME12
HOME13
HOME14
HOME15
HOME16
轴零位1
轴零位8
轴零位9
轴零位16
Y0
Y1
Y2
Y3
Y4
Y5
Y6
Y7
Y8
Y9
Y10
Y11
Y12
Y13
Y14
Y15
输出0
输出7
输出8
输出15
触摸屏口（可选）
485口（可选）
IO扩展口
网口（侧面）

二、软件资源

控制卡提供了 VC++及 C#和 Delphi 以及 VB 下的动态库，用户可利用动态库提供的 API 完成板卡打开、关闭、IO 输入输出、电机点位/速度/插补/硬件捕获/电子齿轮/比较输出等运动控制功能。Labview 下也可以通过调用 C++动态库的方式使用。同时板卡支持 Linux、Android、iOS、Wince、Python、QT 等开发环境及语言。

名称	修改日期	类型	大小
测试软件-EthBoardTest_2016	2019/1/23 17:44	文件夹	
开发例程源代码（基于VS2010）	2019/1/23 17:44	文件夹	
库文件及头文件	2019/1/23 17:44	文件夹	
8轴运动控制卡用户手册V2.0.pdf	2018/7/1 14:46	Adobe Acrobat ...	1,663 KB
开发工具软件VS2010下载地址（百度云盘链接）.txt	2018/7/24 20:55	文本文档	1 KB



三、硬件连接

3.1、8 轴及以下轴信号接口

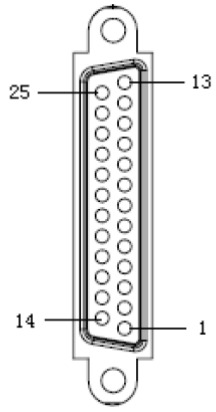
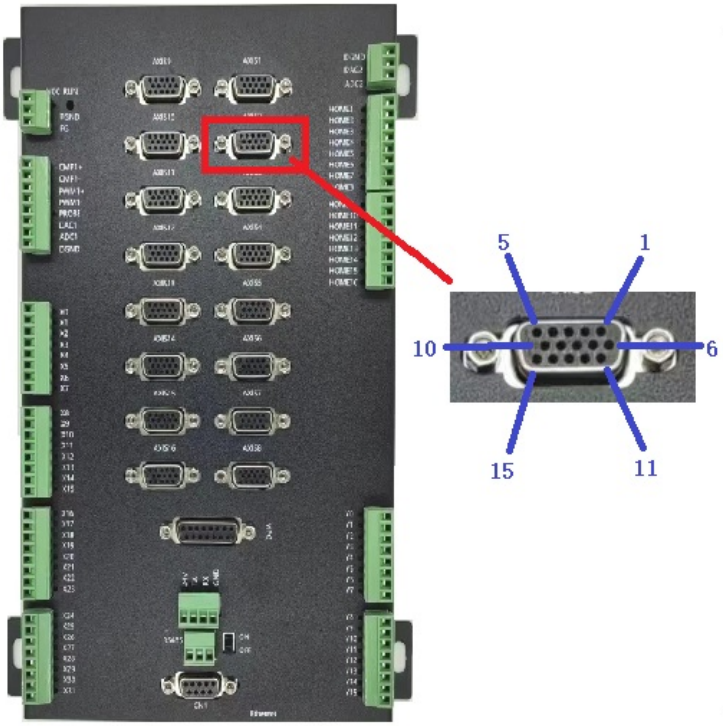


图 3-1：轴通道接口引脚说明

管脚序号	信号	管脚说明
1	OGND	外部电源地
2	ALM	驱动报警
3	ENABLE	驱动使能
4	A-	编码器输入 A-
5	B-	编码器输入 B-
6	C-	编码器输入 C-
7	+5V	电源输出
8	NC	保留
9	DIR+	方向+（5V 差分输出，严禁外接电源）
10	GND	数字地
11	PLUSE-	脉冲输出-（5V 差分输出，严禁外接电源）
12	NC	保留
13	GND	数字地
14	OVCC	+24V 输出(该输出不能用做步进电机动力电)
15	NC	保留
16	NC	保留
17	A+	编码器输入 A+
18	B+	编码器输入 B+
19	C+	编码器输入 C+
20	GND	数字地
21	GND	数字地
22	DIR-	方向-（5V 差分输出，严禁外接电源）
23	PLUSE+	脉冲输出+（5V 差分输出，严禁外接电源）
24	GND	数字地
25	NC	保留

3.2、10 轴及以上轴信号接口

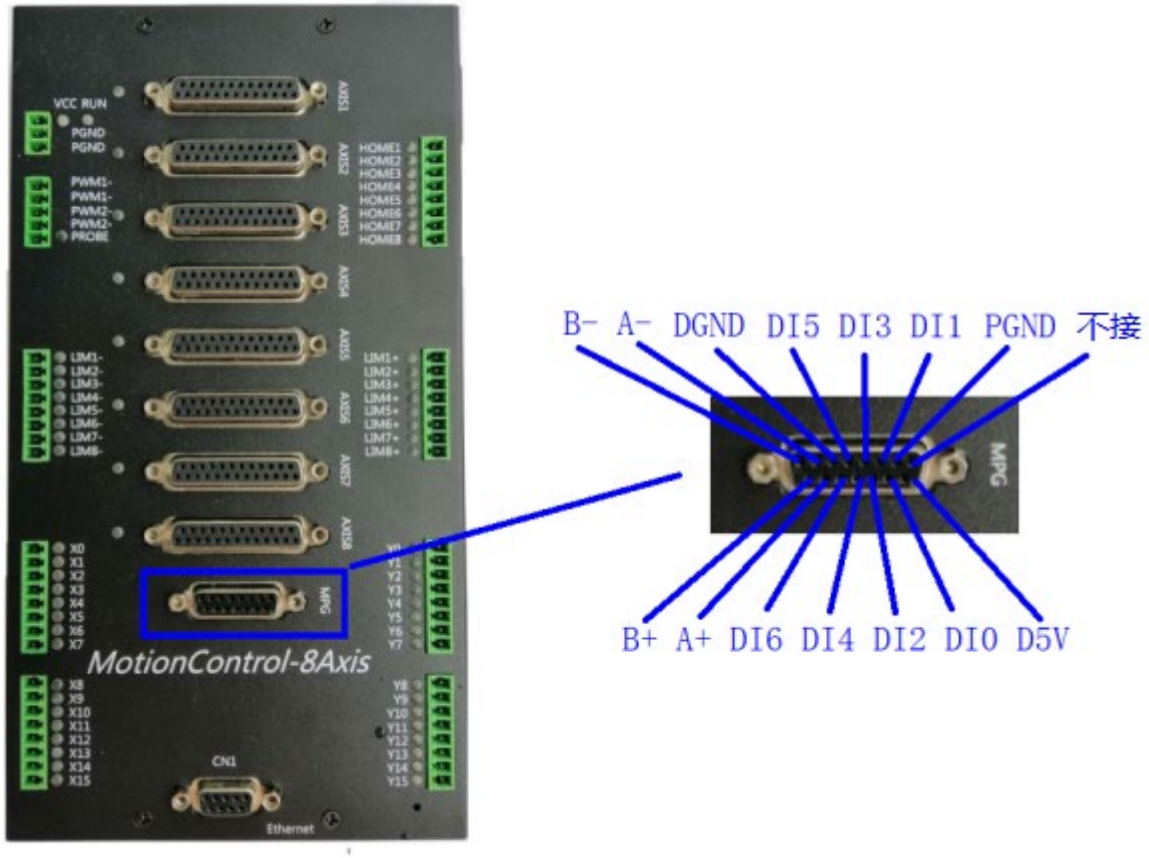


序号	信号	管脚说明
1	PLUSE+	脉冲输出+（5V 差分输出，严禁外接电源）
2	PLUSE-	脉冲输出-（5V 差分输出，严禁外接电源）
3	DIR+	方向+（5V 差分输出，严禁外接电源）
4	DIR-	方向-（5V 差分输出，严禁外接电源）
5	GND	地
6	OVCC	+24V 输出
7	C+	编码器输入 C+
8	C-	编码器输入 C-
9	NC	保留
10	ALM	驱动报警
11	A+	编码器输入 A+
12	A-	编码器输入 A-
13	B+	编码器输入 B+
14	B-	编码器输入 B-
15	ENABLE	驱动使能

图 3-1：轴通道接口引脚说明

序号	信号	管脚说明
1	PLUSE+	脉冲输出+（5V 差分输出，严禁外接电源）
2	PLUSE-	脉冲输出-（5V 差分输出，严禁外接电源）
3	DIR+	方向+（5V 差分输出，严禁外接电源）
4	DIR-	方向-（5V 差分输出，严禁外接电源）
5	GND	地
6	OVCC	+24V 输出(该输出不能用做步进电机动力电)
7	C+	编码器输入 C+
8	C-	编码器输入 C-
9	NC	保留
10	ALM	驱动报警
11	A+	编码器输入 A+
12	A-	编码器输入 A-
13	B+	编码器输入 B+
14	B-	编码器输入 B-
15	ENABLE	驱动使能

3.3、手轮通道接口



控制卡	引脚	手轮
D5V	9	VCC 和 LED+
PGND	2	0V 和 LED-
A+	14	A
A-	7	A-
B+	15	B
B-	8	B-
DI0	10	X
DI1	3	Y
DI2	11	Z
DI3	4	A
DI4	12	B
DI5	5	X1
DI6	13	X10
DGND	6	COM

重点说明：X100 不用接，当 X1 和 X10 没有触发的时候，就是 X100

3.3、通用数字输入输出信号、原点信号和限位信号接口

通用输入输出均为 NPN.

注:千万不可将 24V 直接接入输出 IO，可能导致输出端口短路，烧坏板卡！！！

VCC	外部电源 24V
PGND	外部电源地
X0	通用输入
X1	通用输入
X2	通用输入
X3	通用输入
X4	通用输入
X5	通用输入
X6	通用输入
X7	通用输入
X8	通用输入
X9	通用输入
X10	通用输入
X11	通用输入
X12	通用输入
X13	通用输入
X14	通用输入
X15	通用输入
Y0	通用输出
Y1	通用输出
Y2	通用输出
Y3	通用输出
Y4	通用输出
Y5	通用输出
Y6	通用输出
Y7	通用输出
Y8	通用输出
Y9	通用输出
Y10	通用输出
Y11	通用输出
Y12	通用输出
Y13	通用输出
Y14	通用输出
Y15	通用输出
LIM1-	1 轴负向限位
LIM2-	2 轴负向限位
LIM3-	3 轴负向限位
LIM4-	4 轴负向限位

LIM5-	5 轴负向限位
LIM6-	6 轴负向限位
LIM7-	7 轴负向限位
LIM8-	8 轴负向限位
LIM1+	1 轴正向限位
LIM2+	2 轴正向限位
LIM3+	3 轴正向限位
LIM4+	4 轴正向限位
LIM5+	5 轴正向限位
LIM6+	6 轴正向限位
LIM7+	7 轴正向限位
LIM8+	8 轴正向限位
HOME1	1 轴原点输入
HOME2	2 轴原点输入
HOME3	3 轴原点输入
HOME4	4 轴原点输入
HOME5	5 轴原点输入
HOME6	6 轴原点输入
HOME7	7 轴原点输入
HOME8	8 轴原点输入

注:千万不可将 24V 直接接入输出 IO，可能导致输出端口短路，烧坏板卡！！！

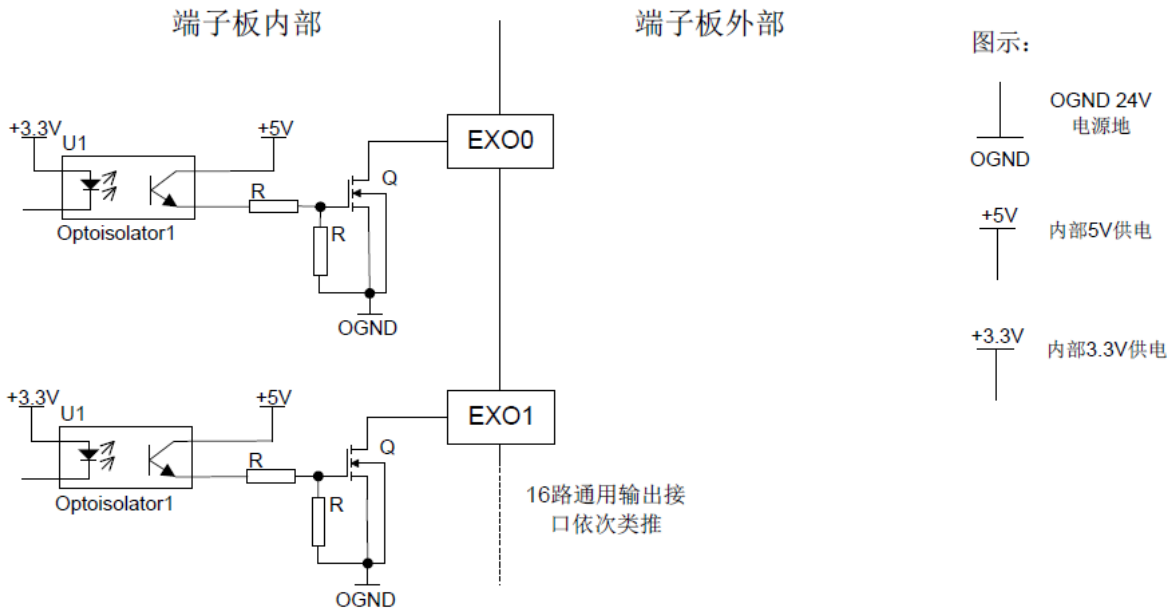
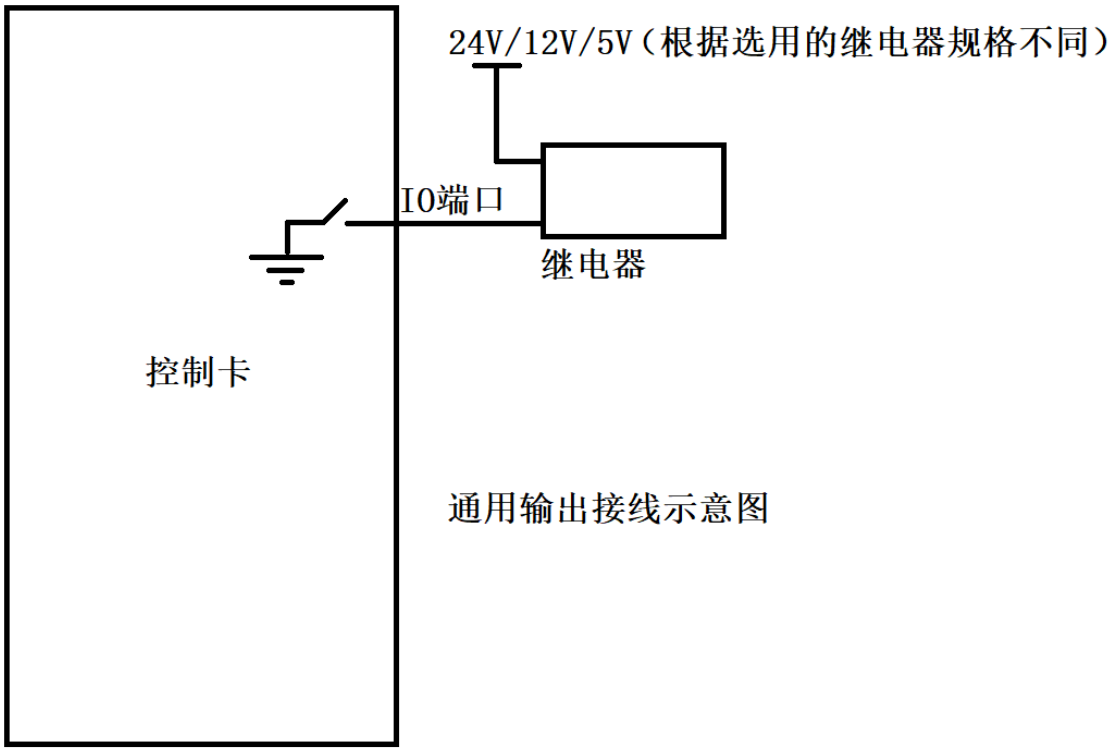


图 3-3：端子板通用数字输出信号内部电路示意图



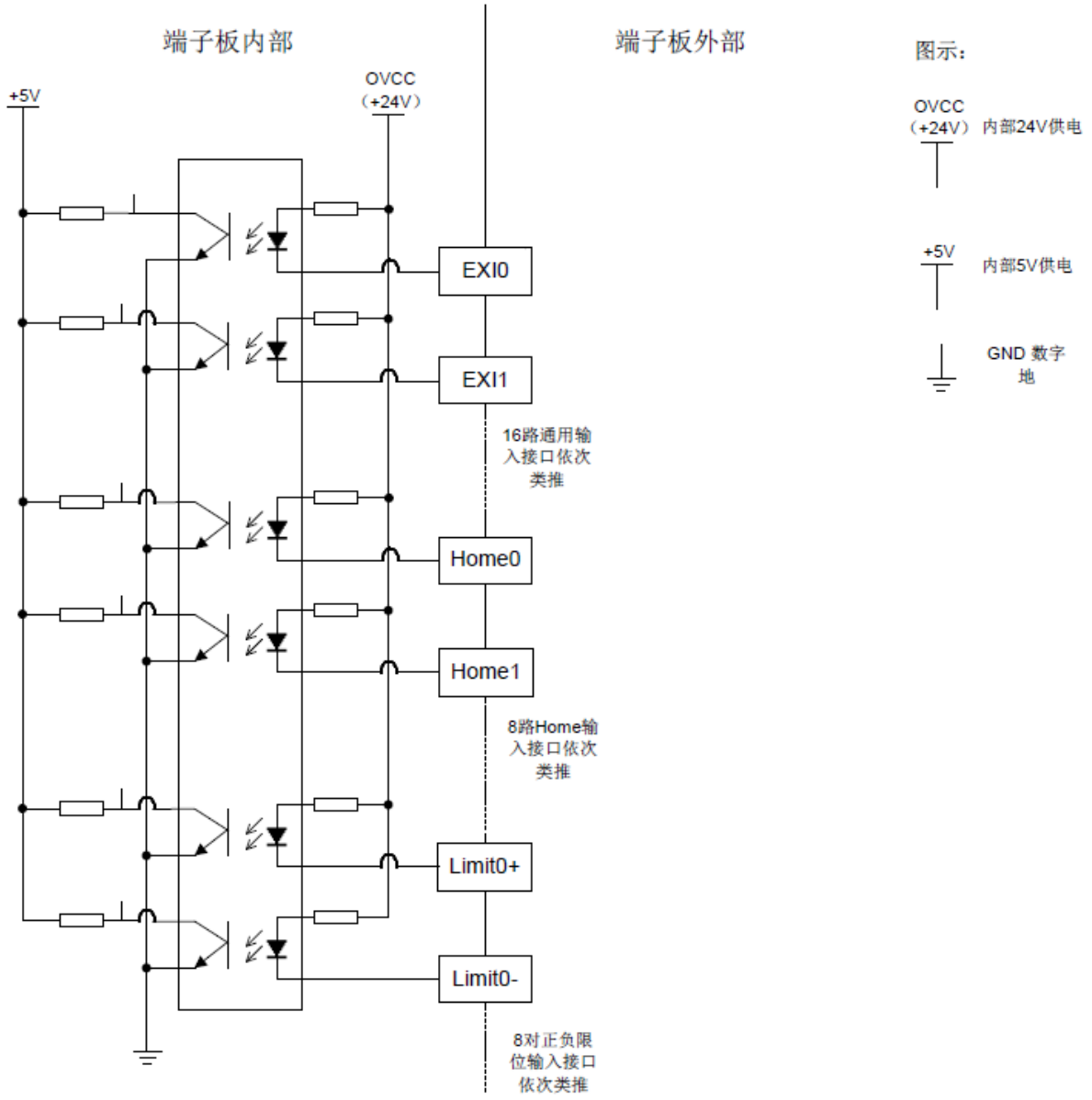
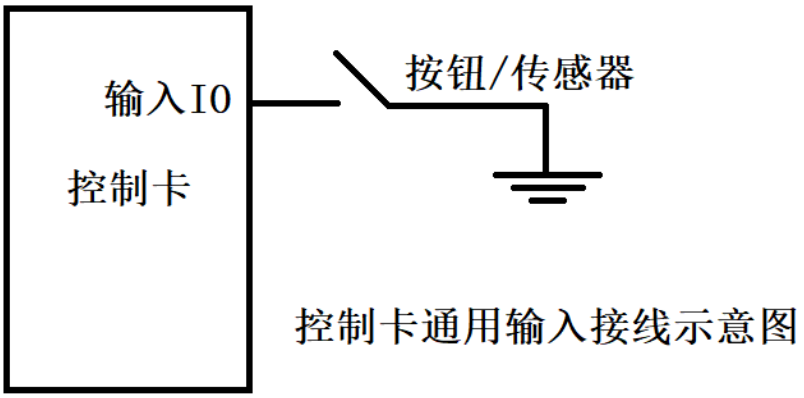


图 3-2：端子板通用输入，HOME 输入，LIMIT 输入信号内部电路示意图



控制卡通用输入接线示意图

四、API 返回值及其意义

返回值	意义	处理方法
0	执行成功	
1	执行失败	检测命令执行条件是否满足
2	版本不支持该 API	如有需要，联系厂家
7	参数错误	检测参数是否合理
-1	通讯失败	接线是否牢靠，更换板卡
-6	打开控制器失败	是否输入正确串口名，是否调用 2 次 MC_Open
-7	运动控制器无响应	检测运动控制器是否连接，是否打开。更换板卡

五、API 使用说明

5.1、板卡打开关闭 API

API	说明
MC_SetCardNo	切换当前运动控制器卡号
MC_GetCardNo	读取当前运动控制器卡号
MC_Open	打开板卡
MC_OpenByIP	打开板卡
MC_Reset	复位板卡
MC_Close	关闭板卡

参数详细说明：

int MC_SetCardNo(short iCardNum)	
iCardNum	将被设置为当前运动控制器的卡号，取值范围：[1,255]
int MC_GetCardNo(short *pCardNum)	
pCardNum	读取的当前运动控制器的卡号
int MC_Open(short iType=0,char* cName="COM1")	
iType	打开方式，0 网口，1 串口
cName	当 iType=0（网口方式打开）时，该参数代表 PC 端 IP 地址 当 iType=1（串口方式打开）时，该参数代表默认串口号
int MC_Open(short nCardNum,char* cPCEthernetIP,unsigned short nPCEthernetPort,char* cCardEthernetIP,unsigned short nCardEthernetPort)	
nCardNum	卡号，从 1 开始，第一个卡填 1，第二个卡填 2.....
cPCEthernetIP	电脑 IP 地址（需要和板卡在同一个网段）
nPCEthernetPort	电脑端口号，通常第一个卡填 60000，第二个卡填 60001.....
cCardEthernetIP	板卡 IP 地址（出厂默认 192.168.0.1）
nCardEthernetPort	板卡端口号（和电脑端口号保持一致即可）
int MC_OpenByIP(char* cPCIP,char* cCardIP,unsigned long ulID,unsigned short nRetryTime)	
cPCIP	电脑 IP
cCardIP	板卡 IP
ulID	固定为 0
nRetryTime	固定为 0
int MC_Reset()	
无参数	
int MC_Close()	
无参数	

示例代码：

```
int iRes = 0;
iRes += MC_SetCardNo(1);//切换到第 1 块板卡
iRes += MC_Open(0,"192.168.0.200");//打开板卡(通过网口，PC 端 IP 地址为 192.168.0.200)
iRes += MC_Reset();//复位板卡
iRes += MC_Close();//关闭板卡
```

5.2、板卡配置类 API

API	说明
MC_AlarmOn	设置轴驱动报警信号有效
MC_AlarmOff	设置轴驱动报警信号无效
MC_AlarmSns	设置运动控制器轴报警信号电平逻辑
MC_LmtsOn	设置轴限位信号有效
MC_LmtsOff	设置轴限位信号无效
MC_LmtSns	设置运动控制器各轴限位触发电平
MC_EncOn	设置为“外部编码器”计数方式
MC_EncOff	设置为“脉冲计数器”计数方式
MC_EncSns	设置编码器的计数方向
MC_StepSns	设置脉冲输出通道的方向
MC_HomeSns	设置运动控制器 HOME 输入的电平逻辑

参数详细说明：

int MC_AlarmOn(short nAxisNum)	
nAxisNum	控制轴号，取值范围：[1,AXIS_MAX_COUNT]
int MC_AlarmOff(short nAxisNum)	
nAxisNum	控制轴号，取值范围：[1,AXIS_MAX_COUNT]
int MC_AlarmSns(unsigned short nSense)	
nSense	按位表示各数量输入的电平逻辑，从 bit0~bit7，分别对应-8 轴的电平逻辑
int MC_LmtsOn(short nAxisNum,short limitType = -1)	
nAxisNum	控制轴号，取值范围：[1,AXIS_MAX_COUNT]
limitType	需要有效的限位类型 0：需要将该轴的正限位有效 1：需要将该轴的负限位有效 -1：需要将该轴的正限位和负限位都有效，默认为该值
int MC_LmtsOff(short nAxisNum,short limitType=-1)	
nAxisNum	控制轴号，取值范围：[1,AXIS_MAX_COUNT]
limitType	需要有效的限位类型 0：需要将该轴的正限位无效 1：需要将该轴的负限位无 -1：需要将该轴的正限位和负限位都无效，默认为该值
int MC_LmtSns(unsigned short nSense)	
nSense	1、此函数用于按位设置轴的限位的触发电平状态。 2、运动控制器默认的限位开关是常闭开关，即各轴处于正常工作状态时，其限位信号输入为低电平，当限位信号为高电平时，限位触发。 3、如果使用的传感器是常开，则需要调用本函数，将限位的逻辑电平反一下。 4、参数 nSense 一共 16 位，分别代表 8 个轴的正限位和负限位。 如下表所示

	Bit0	轴 1 正限位逻辑电平（1 低电平触发，0 高电平触发）
	Bit1	轴 1 负限位逻辑电平（1 低电平触发，0 高电平触发）
	Bit2	轴 2 正限位逻辑电平（1 低电平触发，0 高电平触发）

	nSense (16 位)	Bit3	轴 2 负限位逻辑电平（1 低电平触发，0 高电平触发）
		Bit4	轴 3 正限位逻辑电平（1 低电平触发，0 高电平触发）
		Bit5	轴 3 负限位逻辑电平（1 低电平触发，0 高电平触发）
		Bit6	轴 4 正限位逻辑电平（1 低电平触发，0 高电平触发）
		Bit7	轴 4 负限位逻辑电平（1 低电平触发，0 高电平触发）
		Bit8	轴 5 正限位逻辑电平（1 低电平触发，0 高电平触发）
		Bit9	轴 5 负限位逻辑电平（1 低电平触发，0 高电平触发）
		Bit10	轴 6 正限位逻辑电平（1 低电平触发，0 高电平触发）
		Bit11	轴 6 负限位逻辑电平（1 低电平触发，0 高电平触发）
		Bit12	轴 7 正限位逻辑电平（1 低电平触发，0 高电平触发）
		Bit13	轴 7 负限位逻辑电平（1 低电平触发，0 高电平触发）
		Bit14	轴 8 正限位逻辑电平（1 低电平触发，0 高电平触发）
		Bit15	轴 8 负限位逻辑电平（1 低电平触发，0 高电平触发）

```
//例程
//将轴 1 的正负限位都设置为常开，低电平触发
MC_LmtSns(0X03);

//将轴 2 的正负限位都设置为常开，低电平触发
MC_LmtSns(0X0C);

//将轴 1 和 2 的正负限位都设置为常开，低电平触发
MC_LmtSns(0X0F);

//注意：这个函数是一次性设置 8 个轴的限位电平
```

int MC_EncOn(short nEncoderNum)	
nEncoderNum	编码器通道号
int MC_EncOff(short nEncoderNum)	
nEncoderNum	编码器通道号
int MC_EncSns(unsigned short nSense)	
nSense	按位标识编码器的计数方向，bit0~bit7 依次对应编码器~8，bit8 对应辅助编码器：该编码器计数方向不取反
int MC_StepSns(unsigned short sense)	
sense	bit0 对应脉冲输出通道 1，bit1 对应脉冲输出通道 2，以此类推 对应位为 0 表示不反向，对应位为 1 表示反向
int MC_HomeSns(unsigned short sense)	
sense	sense：按位表示各数量输入的电平逻辑，从 bit0~bit7，分别对应轴 1-8 0：HOME 电平不取反 1：HOME 电平取反

示例代码：

```
int iRes = 0;

iRes += MC_SetCardNo(1);//切换到第 1 块板卡
iRes += MC_Open(0,"192.168.0.200");//打开板卡（通过网口，PC 端 IP 地址为 192.168.0.200）
iRes += MC_Reset();//复位板卡
```

```
iRes += MC_AlarmOn(1);//设置轴 1 驱动报警信号有效
iRes += MC_AlarmOff(1);//设置轴 1 驱动报警信号无效
iRes += MC_Close();//关闭板卡
```

5.3、IO/模拟量/PWM 常规操作 API

API	说明
MC_GetDiRaw	获取 IO 输入（包含主卡 IO、限位、零位）
MC_GetDiReverseCount	读取数字量输入信号的变化次数
MC_SetDiReverseCount	设置数字量输入信号的变化次数的初值
MC_SetExtDoValue	设置 IO 输出（包含主模块和扩展模块）
MC_GetExtDiValue	获取 IO 输入（包含主模块和扩展模块）
MC_GetExtDoValue	获取 IO 输出（包含主模块和扩展模块）
MC_SetExtDoBit	设置指定 IO 模块的指定位输出（包含主模块和扩展模块）
MC_GetExtDiBit	获取指定 IO 模块的指定位输入（包含主模块和扩展模块）
MC_GetExtDoBit	获取指定 IO 模块的指定位输出（包含主模块和扩展模块）
MC_SetDac（仅限模拟量版本）	设置 DAC 输出电压
MC_GetAdc（仅限模拟量版本）	读取 ADC 输入电压
MC_SetAdcFilter（仅限模拟量版本）	设置模拟量输入滤波时间
MC_SetAdcBias（仅限模拟量版本）	设置模拟量输入通道的零漂电压补偿值
MC_GetAdcBias（仅限模拟量版本）	读取模拟量输入通道的零漂电压补偿值
MC_SetPwm（仅限模拟量版本）	设置 PWM 输出频率以及占空比
MC_SetDoBitReverse	设置数字 IO 输出指定时间的单个脉冲

参数详细说明：

int MC_GetDiRaw(short nDiType,long *pValue)	
diType	指定数字 IO 类型 MC_LIMIT_POSITIVE(该宏定义为 0) 正限位 MC_LIMIT_NEGATIVE(该宏定义为 1) 负限位 MC_ALARM(该宏定义为 2) 驱动报警 MC_HOME(该宏定义为 3) 原点开关 MC_GPI(该宏定义为 4) 通用输入 MC_MPG(该宏定义为 7) 手轮 IO 输入
pValue	IO 输入值存放指针
int MC_GetDiReverseCount(short nDiType,short diIndex,unsigned long*pReserveCount,short nCount=1)	
nDiType	指定数字 IO 类型 MC_LIMIT_POSITIVE(该宏定义为 0) 正限位 MC_LIMIT_NEGATIVE(该宏定义为 1) 负限位 MC_ALARM(该宏定义为 2) 驱动报警 MC_HOME(该宏定义为 3) 原点开关 MC_GPI(该宏定义为 4) 通用输入 MC_ARRIVE(该宏定义为 5) 电机到位信号
diIndex	数字量输入的索引,取值范围： nDiType= MC_LIMIT_POSITIVE 时：[0,7] nDiType= MC_LIMIT_NEGATIVE 时：[0,7] nDiType= MC_ALARM 时：[0,0] nDiType= MC_HOME 时：[0,7] nDiType= MC_GPI 时：[0,15]

	nDiType= MC_ARRIVE 时：[0,7]
pReserveCount	读取的数字量输入的变化次数
nCount	读取变化次数的数字量输入的个数，默认为 1
int MC_SetDiReverseCount(short nDiType,short diIndex,unsigned long ReserveCount,short nCount)	
nDiType	指定数字 IO 类型 MC_LIMIT_POSITIVE(该宏定义为 0) 正限位 MC_LIMIT_NEGATIVE(该宏定义为 1) 负限位 MC_ALARM(该宏定义为 2) 驱动报警 MC_HOME(该宏定义为 3) 原点开关 MC_GPI(该宏定义为 4) 通用输入 MC_ARRIVE(该宏定义为 5) 电机到位信号
diIndex	数字量输入的索引,取值范围： nDiType= MC_LIMIT_POSITIVE 时：[0,7] nDiType= MC_LIMIT_NEGATIVE 时：[0,7] nDiType= MC_ALARM 时：[0,7] nDiType= MC_HOME 时：[0,7] nDiType= MC_GPI 时：[0,15] nDiType= MC_ARRIVE 时：[0,7]
ReserveCount	设置的数字量输入的变化次数
nCount	设置变化次数的数字量输入的个数，默认为 1
int MC_SetExtDoValue(short nCardIndex,unsigned long *value,short nCount=1)	
nCardIndex	起始板卡索引(0~63),0 是主模块，扩展模块从 1 开始
value	IO 输出值存放指针
nCount	本次设置的模块数量（1~64）
int MC_GetExtDiValue(short nCardIndex,unsigned long *pValue,short nCount=1)	
nCardIndex	起始板卡索引(0~63),0 是主模块，扩展模块从 1 开始
pValue	IO 输入值存放指针
nCount	本次获取的模块数量(1~64)
int MC_GetExtDoValue(short nCardIndex,unsigned long *pValue,short nCount=1)	
nCardIndex	起始板卡索引(0~63),0 是主模块，扩展模块从 1 开始
pValue	IO 输出值存放指针
nCount	本次获取的模块数量(1~64)
int MC_SetExtDoBit(short nCardIndex,short nBitIndex,unsigned short nValue)	
nCardIndex	起始板卡索引。0 代表主卡，1 代表扩展卡 1.....
nBitIndex	IO 位索引号.0~31
nValue	IO 输出值.1 代表打开输出口，0 代表关闭输出口

举例说明：
MC_SetExtDoBit(0,3,1)//打开主卡 Y3 输出
MC_SetExtDoBit(0,3,0)//关闭主卡 Y3 输出
 	
MC_SetExtDoBit(1,4,1)//打开第一个 IO 扩展卡 Y4 输出（带 IO 扩展卡时可以用）
MC_SetExtDoBit(1,4,0)//关闭第一个 IO 扩展卡 Y4 输出（带 IO 扩展卡时可以用）	

int MC_GetExtDiBit(short nCardIndex,short nBitIndex,unsigned short *pValue)	
nCardIndex	起始板卡索引(0~63),0 是主模块，扩展模块从 1 开始

nBitIndex	IO 位索引号(0~15)
pValue	IO 输入值存放指针
int MC_GetExtDoBit(short nCardIndex,short nBitIndex,unsigned short *pValue)	
nCardIndex	起始板卡索引(0~63),0 是主模块，扩展模块从 1 开始
nBitIndex	IO 位索引号(0~15)
pValue	IO 输入值存放指针
int MC_SetDac(short nDacNum,short* pValue,short nCount=1)	
nDacNum	DAC 起始通道号,1~N
Value	输出电压 0 对应 0V；10000 对应 10V
nCount	设置的通道数，默认为 1，1 次最多可以设置 2 路 DAC 输出
int MC_GetAdc(short nAdcNum,short *pValue,short nCount=1,unsigned long *pClock=NULL)	
nAdcNum	ADC 通道号，1~N
pValue	ADC 通道电压
nCount	读取的通道数，默认为 1，1 次最多可以读取 2 个 DAC 轴
int MC_SetAdcFilter(short nAdcNum,short nFilterTime)	
nAdcNum	模拟量输入通道号,取值范围 1~2
nFilterTime	滤波时间，8~200，单位毫秒。需为 8 的整数倍
int MC_SetAdcBias(short nAdcNum,short nBias)	
nAdcNum	模拟量输入通道号,取值范围 1~2
nBias	待设置的零漂补偿值，取值范围：[0~2000]对应 0~2.000V
int MC_GetAdcBias(short nAdcNum,short *pBias)	
nAdcNum	模拟量输入通道号,取值范围 1~2
pBias	读取到的零漂补偿值，取值范围：[0~2000]对应 0~2.000V
int MC_SetPwm(short nPwmNum ,double dFreq,double dDuty)	
nPwmNum	起始 PWM 通道，通常为 1
dFreq	PWM 频率，0~2000000
dDuty	占空比，0~100
int MC_SetDoBitReverse(short nDoType,short nDoNum,short nValue,short nReverseTime)	
nDoType	指定数字 IO 类型，通常为固定为 12
nDoNum	输出 IO 的编号 取值范围：[1,16]
nValue	nValue 设置数字 IO 输出电平。 1 表示高电平，0 表示低电平
nReverseTime	nReverseTime 维持 value 所设置电平的时间，取值范围：[0,32767]，单位：1ms

示例代码：

```
int iRes = 0;
iRes += MC_SetCardNo(1);//切换到第 1 块板卡
iRes += MC_Open(0,"192.168.0.200");//打开板卡（通过网口，PC 端 IP 地址为 192.168.0.200）
iRes += MC_Reset();//复位板卡
iRes += MC_SetExtDoBit(0,4,1);//设置板卡 1 端口 3IO 输出
iRes += MC_SetExtDoBit(0,4,0);//关闭板卡 1 端口 3IO 输出

//Y3 发出一个 100ms 时间宽度的负脉冲。
```

```
iRes += MC_SetDoBitReverse(12,4,0,100);

iRes += MC_Close();//关闭板卡
```

5.4、点位运动 API

API	说明
MC_PrfTrap	设置指定轴为点位模式
MC_SetTrapPrm	设置点位模式运动参数
MC_SetTrapPrmSingle	设置点位模式运动参数（可替代 MC_SetTrapPrm）
MC_GetTrapPrm	读取点位模式运动参数
MC_GetTrapPrmSingle	读取点位模式运动参数（可替代 MC_GetTrapPrm）
MC_SetPos	设置目标位置
MC_SetVel	设置目标速度
MC_Update	启动点位运动
MC_SetTrapPosAndUpdate	设置指定轴进行点位运动，可代替以上所有函数，效率高

参数详细说明：

int MC_PrfTrap(short nAxisNum)	
iAxis	规划轴号
int MC_SetTrapPrm(short nAxisNum,TTrapPrm *pPrm)	
iAxis	规划轴号

pPrm	设置点位模式运动参数
	
	//点位模式参数结构体
	typedef struct TrapPrm
	{
	double acc;//加速度
	double dec;//减速度
	double velStart;//起始速度
	short smoothTime;//平滑时间
	}TTrapPrm;
	
	Labview 下可用 MC_SetTrapPrmSingle 函数替代本函数

int MC_SetTrapPrmSingle(short nAxisNum,double dAcc,double dDec,double dVelStart,short dSmoothTime)	
nAxisNum	规划轴号
dAcc	加速度
dDec	减速度
dVelStart	启动速度
dSmoothTime	平滑时间
int MC_GetTrapPrm(short nAxisNum,TTrapPrm *pPrm)	
iAxis	规划轴号

pPrm	读取点位模式运动参数
	
	//点位模式参数结构体
	typedef struct TrapPrm
	{
	double acc;//加速度

	double dec;//减速度
	double velStart;//起始速度
	short smoothTime;//平滑时间
	}TTrapPrm;
	
	Labview 下可用 MC_GetTrapPrmSingle 函数替代本函数

int MC_GetTrapPrmSingle(short nAxisNum,double* dAcc,double* dDec,double* dVelStart,short* dSmoothTime)	
nAxisNum	规划轴号
dAcc	加速度存放指针
dDec	减速度存放指针
dVelStart	启动速度存放指针
dSmoothTime	平滑时间存放指针
int MC_SetPos(short nAxisNum,long pos)	
iAxis	规划轴号
pos	设置目标位置，单位是脉冲
int MC_SetVel(short nAxisNum,double vel)	
iAxis	规划轴号
vel	设置目标速度，单位是“脉冲/毫秒”
int MC_Update(long mask)	

mask	按位指示需要启动点位运动的轴号
	bit0 表示轴，bit1 表示 2 轴，……
	举例说明：
	MC_Update(1)//启动轴 1（Bit0 为 1）
	MC_Update(2)//启动轴 2（Bit1 为 1）
	MC_Update(7)//启动轴 1、2、3（Bit0、1、2 为 1）
	MC_Update(8)//启动轴 4（Bit3 为 1）

int MC_SetTrapPosAndUpdate(short nAxisNum,long long llPos,double dVel,double dAcc,double dDec,double dVelStart,short nSmoothTime,short nBlock);	
nAxisNum	轴号，从1开始
llPos	目标位置，单位脉冲
dVel	速度，单位脉冲/毫米，可以为小数
dAcc	加速度，单位脉冲/毫秒/毫秒，可以为小数
dDec	减速度，单位脉冲/毫秒/毫秒，可以为小数
dVelStart	固定为0
nSmoothTime	固定为0
nBlock	固定为0

示例代码：

```
int iRes = 0;

iRes += MC_SetCardNo(1);//切换到第 1 块板卡
iRes += MC_Open(0,"192.168.0.200");//打开板卡（通过网口，PC 端 IP 地址为 192.168.0.200）
iRes += MC_Reset();//复位板卡
iRes += MC_AxisOn(1);//设置轴 1 使能
```

```
iRes += MC_PrfTrap(1);//设置板卡轴 1 为点位模式

TTrapPrm TrapPrm;
TrapPrm.acc = 0.5;//设置点位运动加速度为 0.5 脉冲/毫秒^2
TrapPrm.dec = 0.5;//设置点位运动减速度为 0.5 脉冲/毫秒^2
TrapPrm.velStart = 0;//设置点位运动起始速度为 0 脉冲/毫秒
TrapPrm.smoothTime = 0;//设置点位运动平滑时间为 0

iRes += MC_SetTrapPrm(m_iAxisNum,&TrapPrm);
iRes += MC_SetVel(1,10);//设置轴 1 点位运动速度为 10 脉冲/毫秒
iRes += MC_SetPos(1,10000);//设置轴 1 目标位置为 10000
iRes += MC_Update(0XFF);//启动点位运动
```

5.5、JOG 运动 API

API	说明
MC_PrfJog	设置指定轴为 JOG 模式(速度模式)
MC_SetJogPrm	设置 JOG 模式运动参数
MC_SetJogPrmSingle	设置 JOG 模式运动参数（可替代 MC_SetJogPrm）
MC_GetJogPrm	读取 JOG 模式运动参数
MC_GetJogPrmSingle	读取 JOG 模式运动参数（可替代 MC_GetJogPrm）
MC_SetVel	设置目标速度
MC_Update	启动 JOG 运动

参数详细说明：

int MC_PrfJog(short nAxisNum)	
iAxis	轴号，从 1 开始
int MC_SetJogPrm(short nAxisNum,TJogPrm *pPrm)	
iAxis	轴号，从 1 开始

pPrm	设置 Jog 模式运动参数
	
	//JOG 模式参数结构体
	typedef struct JogPrm
	{
	double dAcc;//加速度
	double dDec;//减速度
	double dSmooth;//平滑时间
	}TJogPrm;
	
	Labview 下可用 MC_SetJogPrmSingle 函数替代本函数

int MC_SetJogPrmSingle(short nAxisNum,double dAcc,double dDec,double dSmooth)	
nAxisNum	规划轴号
dAcc	加速度
dDec	减速度

dSmooth	平滑时间
int MC_GetJogPrm(short nAxisNum,TJogPrm *pPrm)	
iAxis	轴号，从 1 开始

pPrm	获取 Jog 模式运动参数
	
	//JOG 模式参数结构体
	typedef struct JogPrm
	{
	double dAcc;//加速度
	double dDec;//减速度
	double dSmooth;//平滑时间
	}TJogPrm;
	
	Labview 下可用 MC_GetJogPrmSingle 函数替代本函数

int MC_GetJogPrmSingle(short nAxisNum,double* dAcc,double* dDec,double* dSmooth)	
nAxisNum	规划轴号
dAcc	加速度存放指针
dDec	减速度存放指针
dSmooth	平滑时间
int MC_SetVel(short nAxisNum,double vel)	
iAxis	轴号，从 1 开始
vel	设置目标速度，单位是“脉冲/毫秒”
int MC_Update(long mask)	
mask	按位指示需要启动JOG运动的轴号 bit0 表示轴，bit1 表示 2 轴，.....

示例代码：

```
int iRes = 0;

iRes += MC_SetCardNo(1);//切换到第 1 块板卡
iRes += MC_Open(0,"192.168.0.200");//打开板卡（通过网口，PC 端 IP 地址为 192.168.0.200）
iRes += MC_Reset();//复位板卡

iRes += MC_PrfJog(1);//设置轴 1 为 Jog 模式
iRes += MC_AxisOn(1);//设置轴 1 使能

TJogPrm JogPrm;
JogPrm.dSmooth = 0;
JogPrm.dAcc = 0.5;//设置 JOG 运动加速度为 0.5 脉冲/毫秒^2
JogPrm.dDec = 0.5;//设置 JOG 运动减速度为 0.5 脉冲/毫秒^2

//注意：如果轴当前模式不是 Jog 模式，设置 JOG 运动参数会失败，会返回 1
iRes = MC_SetJogPrm(1,&JogPrm);//设置 JOG 运动参数

iRes = MC_SetVel(1,-1*fabs(50));//设置 JOG 运动速度为 50 脉冲/毫秒
```

```
iRes += MC_Update(0XFF);//启动 JOG 运动
```

5.6、运动状态检测类 API

API	说明
MC_AxisOn	打开驱动器使能
MC_AxisOff	关闭驱动器使能
MC_Stop	停止一个或多个轴的规划运动，停止坐标系运动
MC_GetSts	读取轴状态
MC_ClrSts	清除驱动器报警标志、跟随误差越限标志、限位触发标志
MC_GetPrfPos	读取规划位置
MC_GetPrfVel	读取规划速度
MC_GetAxisEncPos	读取编码器位值
MC_GetAllSysStatus	获取所有板卡相关状态

参数详细说明：

int MC_AxisOn(short nAxisNum)	
nAxisNum	轴编号，取值范围：[1,AXIS_MAX_COUNT]
int MC_AxisOff(short nAxisNum)	
nAxisNum	轴编号，取值范围：[1,AXIS_MAX_COUNT]
int MC_Stop(long lMask,long lOption)	
lMask	按位指示需要停止运动的轴号或者坐标系号 bit0表示轴1，bit1表示轴2，„，bit7表示轴8 bit8表示坐标系1，bit9表示坐标系2 当 bit 位为 1 时表示停止对应的轴或者坐标系
lOption	按位指示停止方式 bit0表示轴1，bit1表示轴2，„，bit7表示轴8 bit8表示坐标系1，bit9表示坐标系2 当bit位为0时表示平滑停止对应的轴或坐标系 当bit位为1时表示紧急停止对应的轴或坐标系
int MC_GetSts(short nAxisNum,long *pSts,short nCount=1,unsigned long *pClock=NULL)	
nAxisNum	起始轴号

pSts	32位轴状态字，详细定义参见光盘头文件GAS_N.h的轴状态位定义部分
	//轴状态位定义
	#define AXIS_STATUS_ESTOP (0x00000001) //急停
	#define AXIS_STATUS_SV_ALARM (0x00000002) //驱动器报警标志
	#define AXIS_STATUS_POS_SOFT_LIMIT (0x00000004) //正软限位触发标志
	#define AXIS_STATUS_NEG_SOFT_LIMIT (0x00000008) //负软位触发标志
	#define AXIS_STATUS_FOLLOW_ERR (0x00000010) //规划位置和实际位置的误差过大时置 1
	#define AXIS_STATUS_POS_HARD_LIMIT (0x00000020) //正硬限位触发标志
	#define AXIS_STATUS_NEG_HARD_LIMIT (0x00000040) //负硬限位触发标志
	#define AXIS_STATUS_IO_SMS_STOP (0x00000080) //保留
	#define AXIS_STATUS_IO_EMG_STOP (0x00000100) //保留
	#define AXIS_STATUS_ENABLE (0x00000200) //电机使能标志
	#define AXIS_STATUS_RUNNING (0x00000400) //规划运动标志，规划器运动时置 1
	#define AXIS_STATUS_ARRIVE (0x00000800) //电机到位
	#define AXIS_STATUS_HOME_RUNNING (0x00001000) //正在回零

	#define AXIS_STATUS_HOME_SUCESS (0x00002000) //回零成功
	#define AXIS_STATUS_HOME_SWITCH (0x00004000) //零位信号
	#define AXIS_STATUS_INDEX (0x00008000) //z 索引信号
	#define AXIS_STATUS_GEAR_START (0x00010000) //电子齿轮开始啮合
	#define AXIS_STATUS_GEAR_FINISH (0x00020000) //电子齿轮完成啮合

nCount	读取的轴数，默认为 1次最多可以读取多个轴的状态
pClock	读取控制器时钟，默认为：NULL，即不用读取控制器时钟
int MC_ClrSts(short nAxisNum,short nCount)	
nAxisNum	起始轴号，取值范围：[1,AXIS_MAX_COUNT]
nCount	清除的轴数，默认为,1次最多可以清除个轴的异常状态
int MC_GetPrfPos(short nAxisNum,double *pValue,short nCount=1,unsigned long *pClock=NULL)	
nAxisNum	起始轴号，取值范围：[1,AXIS_MAX_COUNT]
pValue	规划位置
nCount	读取的规划轴数，默认为,1次最多可以读取多个轴的运动模式
pClock	读取控制器时钟，默认为：NULL，即不用读取控制器时钟
int MC_GetPrfVel(short nAxisNum,double *pValue,short nCount=1,unsigned long *pClock=NULL)	
nAxisNum	起始轴号，取值范围：[1,AXIS_MAX_COUNT]
pValue	轴的规划速度，单位脉冲/毫秒
nCount	读取的轴数，默认为,1次最多可以读取多个轴的编码器位置
pClock	读取控制器时钟，默认为：NULL，即不用读取控制器时钟
int MC_GetAxisEncPos(short nAxisNum,double *pValue,short nCount=1,unsignedlong *pClock=NULL);	
nAxisNum	起始轴号，取值范围：[1,AXIS_MAX_COUNT]
pValue	轴的编码器位置
nCount	读取的轴数，默认为,1次最多可以读取多个轴的编码器位置
pClock	读取控制器时钟，默认为：NULL，即不用读取控制器时钟
比如读取轴1~轴4的编码器位置，代码如下： double dEncPos[4]; MC_GetAxisEncPos(1,&dEncPos[0],4,NULL); 第一个参数1代表从轴1开始 第二个参数代表读取到的值要存放的变量地址,读取成功后，dEncPos[0]就是轴1编码器位置..类推 第三个参数4代表一次读取4个轴的 第四个参数无意义，固定为NULL	
int MC_GetAllSysStatusSX(TAllSysStatusDataSX *pAllSysStatusData)	

pAllSysStatusData	//16轴以内系统状态结构体
	typedef struct _AllSysStatusDataSX
	{
	long lAxisEncPos[16];//轴编码器位置
	long lAxisPrfPos[16];//轴规划位置
	unsigned long lAxisStatus[16];//轴状态
	short nADCValue[2];//ADC值
	long lUserSegNum[2];//两个坐标系的用户段号
	short lRemainderSegNum[2];//两个坐标系的剩余段号

	short nCrdRunStatus[2];//两个坐标系的坐标系状态
	short lCrdSpace[2];//两个坐标系的剩余空间
	float dCrdVel[2];//两个坐标系的速度
	long lCrdPos[2][5];//两个坐标系的坐标
	short lLimitPosRaw;//正硬限位
	short lLimitNegRaw;//负硬限位
	short lAlarmRaw;//报警输入
	short lHomeRaw;//零位输入
	long lMPGEncPos;//手轮编码器
	int lMPG;//手轮IO信号
	long lGpiRaw[8];//通用IO输入（除主卡外，最大支持7个扩展模块）
	long lGpoRaw[8];//通用IO输出（除主卡外，最大支持7个扩展模块）
	}TAllSysStatusDataSX;

示例代码：

```
int iRes = 0;
long lSts = 0;
double dPrfPos = 0;
Double dEncPos = 0;

iRes += MC_SetCardNo(1);//切换到第 1 块板卡
iRes += MC_Open(0,"192.168.0.200");//打开板卡（通过网口，PC 端 IP 地址为 192.168.0.200）
iRes += MC_Reset();//复位板卡

iRes += MC_AxisOn(1);//设置轴 1 使能
iRes += MC_AxisOff(1);//设置轴 1 断开使能

iRes += MC_Stop((0X0001 << (5-1)),0);//设置轴 5 停止运动
iRes += MC_GetSts(1,&lSts);//获取轴 1 状态
iRes += MC_ClrSts(1);//清除轴 1 运动状态
iRes += MC_GetPrfPos(1,&dPrfPos ,1,NULL)
iRes += MC_GetAxisEncPos(1,&dEncPos,1,NULL)
```

5.7、安全机制 API

重点说明：默认各轴软限位不使能。如果要启用硬限位，还要调用 MC_LmtsOn 函数。

API	说明
MC_SetSoftLimit	设置软限位
MC_GetSoftLimit	获取软限位
MC_LmtsOn	设置轴限位信号有效
MC_LmtsOff	设置轴限位信号无效
MC_LmtSns	设置运动控制器各轴限位触发电平
MC_SetLmtSnsSingle	设置运动控制器指定轴限位触发电平
MC_EStopConfig	配置急停触发时，模拟量值，IO 状态等
MC_EStopSetIO	设置系统紧急停止 IO
MC_EStopOnOff	开启/关闭紧急停止功能
MC_EStopGetSts	获取紧急停止触发状态
MC_EStopClrSts	清除紧急停止触发状态
MC_SetHardLimP（10 轴以上用）	设置正硬限位映射 IO
MC_SetHardLimN（10 轴以上用）	设置负硬限位映射 IO
MC_CrdHlimEnable	坐标系硬限位使能开启/关闭

参数详细说明：

int MC_SetSoftLimit(short nAxisNum,long lPositive ,long lNegative)	
axis	轴编号
positive	正限位，单位脉冲
negative	负限位，单位脉冲
int MC_GetSoftLimit(short nAxisNum,long *pPositive ,long *pNegative)	
axis	轴编号
pPositive	正限位存放指针，单位脉冲
pNegative	负限位存放指针，单位脉冲
int MC_LmtsOn(short nAxisNum,short limitType = -1)	
nAxisNum	控制轴号，取值范围：[1,AXIS_MAX_COUNT]
limitType	需要有效的限位类型 MC_LIMIT_POSITIVE(该宏定义为0)：需要将该轴的正限位有效 MC_LIMIT_NEGATIVE(该宏定义为1)：需要将该轴的负限位有效 -1：需要将该轴的正限位和负限位都有效，默认为该值
int MC_LmtsOff(short nAxisNum,short limitType=-1)	
nAxisNum	控制轴号，取值范围：[1,AXIS_MAX_COUNT]
limitType	需要有效的限位类型 MC_LIMIT_POSITIVE(该宏定义为0)：需要将该轴的正限位无效 MC_LIMIT_NEGATIVE(该宏定义为1)：需要将该轴的负限位无效 -1：需要将该轴的正限位和负限位都无效，默认为该值
int MC_LmtSns(unsigned short nSense)	
nSense	1、此函数用于按位设置轴的限位的触发电平状态。 2、运动控制器默认的限位开关是常闭开关，即各轴处于正常工作状态时，其限位信

号输入为低电平，当限位信号为高电平时，限位触发。
3、如果使用的传感器是常开，则需要调用本函数，将限位的逻辑电平反一下。
4、参数 nSense 一共 16 位，分别代表 8 个轴的正限位和负限位。
如下表所示

nSense (16 位)	Bit0	轴 1 正限位逻辑电平（1 低电平触发，0 高电平触发）
	Bit1	轴 1 负限位逻辑电平（1 低电平触发，0 高电平触发）
	Bit2	轴 2 正限位逻辑电平（1 低电平触发，0 高电平触发）
	Bit3	轴 2 负限位逻辑电平（1 低电平触发，0 高电平触发）
	Bit4	轴 3 正限位逻辑电平（1 低电平触发，0 高电平触发）
	Bit5	轴 3 负限位逻辑电平（1 低电平触发，0 高电平触发）
	Bit6	轴 4 正限位逻辑电平（1 低电平触发，0 高电平触发）
	Bit7	轴 4 负限位逻辑电平（1 低电平触发，0 高电平触发）
	Bit8	轴 5 正限位逻辑电平（1 低电平触发，0 高电平触发）
	Bit9	轴 5 负限位逻辑电平（1 低电平触发，0 高电平触发）
	Bit10	轴 6 正限位逻辑电平（1 低电平触发，0 高电平触发）
	Bit11	轴 6 负限位逻辑电平（1 低电平触发，0 高电平触发）
	Bit12	轴 7 正限位逻辑电平（1 低电平触发，0 高电平触发）
	Bit13	轴 7 负限位逻辑电平（1 低电平触发，0 高电平触发）
	Bit14	轴 8 正限位逻辑电平（1 低电平触发，0 高电平触发）
	Bit15	轴 8 负限位逻辑电平（1 低电平触发，0 高电平触发）

```
//例程
//将轴 1 的正负限位都设置为常开，低电平触发
MC_LmtSns(0X03);

//将轴 2 的正负限位都设置为常开，低电平触发
MC_LmtSns(0X0C);

//将轴 1 和 2 的正负限位都设置为常开，低电平触发
MC_LmtSns(0X0F);

//注意：这个函数是一次性设置 8 个轴的限位电平
//如果控制卡是 8 轴~16 轴，请使用函数 MC_LmtSnsEX，用法与本函数一样
```

int MC_SetLmtSnsSingle(short nAxisNum,short nPosSns,short nNegSns)

nAxisNum	轴号
nPosSns	电平逻辑，0 不反转，常开，1 反转，常闭
nNegSns	电平逻辑，0 不反转，常开，1 反转，常闭

int MC_EStopConfig(unsigned long ulEnableMask,unsigned long ulEnableValue,short nAdcMask,short nAdcValue,unsigned long ulIOMask,unsigned long ulIOValue)

ulEnableMask	急停触发时要设置使能状态的轴掩码，Bit0 代表轴 1，Bit1 代表轴 2，Bit2 代表轴 3.....
ulEnableValue	急停触发时，要设置的使能状态，Bit0 代表轴 1，Bit1 代表轴 2，Bit2 代表轴 3.....
nAdcMask	急停触发时，要设置的模拟量通道掩码，Bit0 代表通道 1，Bit1 代表通道 2.....
nAdcValue	急停触发时，要设置的模拟量值

ulIOMask	急停触发时要设置使能状态的 IO 掩码，Bit0 代表 Y0，Bit1 代表 Y1，Bit2 代表 Y2.....
ulIOValue	急停触发时要设置 IO 值，Bit0 代表 Y0，Bit1 代表 Y1，Bit2 代表 Y2.....
int MC_EStopSetIO(short nCardIndex,short nIOIndex,short nEStopSns,unsigned long lFilterTime)	
nCardIndex	卡号，主卡为0，扩展IO卡依次为1、2、3、4.....
nIOIndex	IO索引，0~15
nEStopSns	电平逻辑，0不反转，1反转
lFilterTime	滤波时间，单位ms
int MC_EStopOnOff(short nEStopOnOff)	
nEStopOnOff	0 紧急停止功能关闭，1 紧急停止功能打开
int MC_EStopGetSts(short *nStatus)	
nStatus	0 紧急停止未触发，1 紧急停止触发
int MC_EStopClrSts()	
int MC_SetHardLimP(short nAxisNum,short nType ,short nCardIndex,short nIOIndex)	
nAxisNum	轴编号
nType	-1:硬限位无效，0：用零位信号当限位，1：用通用输入 IO 当限位
nCardIndex	卡号，主卡为 0，扩展 IO 卡依次为 1、2、3、4.....
nIOIndex	IO 索引，0~31

例如将 Home0 作为轴 1 正限位：
MC_SetHardLimP(1,0,0,0)
//第一个 1 代表轴号，第二个 0 代表用零位当限位，第三个 0 代表主卡，第四个 0 代表 IO 索引
 	
例如将 Home8 作为轴 1 正限位：
MC_SetHardLimP(1,0,0,7)
//第一个 1 代表轴号，第二个 0 代表用零位当限位，第三个 0 代表主卡，第四个 7 代表 IO 索引
 	
例如将 X0 作为轴 3 正限位：
MC_SetHardLimP(3,1,0,0)
//第一个 3 代表轴号，第二个 1 代表用通用输入当限位，第三个 0 代表主卡，第四个 0 代表 IO 索引
 	
例如将 X8 作为轴 3 正限位：
MC_SetHardLimP(3,1,0,8)
//第一个 3 代表轴号，第二个 1 代表用通用输入当限位，第三个 0 代表主卡，第四个 8 代表 IO 索引	

int MC_SetHardLimN(short nAxisNum,short nType ,short nCardIndex,short nIOIndex)	
nAxisNum	轴编号
nType	-1:硬限位无效，0：用零位信号当限位，1：用通用输入 IO 当限位
nCardIndex	卡号，主卡为 0，扩展 IO 卡依次为 1、2、3、4.....
nIOIndex	IO 索引，0~31
例如将 Home0 作为轴 1 负限位： MC_SetHardLimN(1,0,0,0) //第一个 1 代表轴号，第二个 0 代表用零位当限位，第三个 0 代表主卡，第四个 0 代表 IO 索引 例如将 Home8 作为轴 1 负限位： MC_SetHardLimN(1,0,0,7)	

//第一个 1 代表轴号，第二个 0 代表用零位当限位，第三个 0 代表主卡，第四个 7 代表 IO 索引
 	
例如将 X0 作为轴 3 负限位：
MC_SetHardLimN(3,1,0,0)
//第一个 3 代表轴号，第二个 1 代表用通用输入当限位，第三个 0 代表主卡，第四个 0 代表 IO 索引
 	
例如将 X8 作为轴 3 负限位：
MC_SetHardLimN(3,1,0,8)
//第一个 3 代表轴号，第二个 1 代表用通用输入当限位，第三个 0 代表主卡，第四个 8 代表 IO 索引

int MC_CrdHlimEnable(short nCrdNum ,short nEnableFlag)	
nCrdNum	坐标系号
nEnableFlag	0 坐标系硬限位关闭 1 坐标系硬限位打开

示例代码：

```
int iRes = 0;
long lSts = 0;


//long lSoftLimPos= 0X7FFFFFFF;
long lSoftLimPos= 2147483647;

//long lSoftLimNeg=0X80000000;
long lSoftLimNeg=-2147483648;


iRes += MC_SetCardNo(1);//切换到第 1 块板卡
iRes += MC_Open(0,"192.168.0.200");//打开板卡（通过网口，PC 端 IP 地址为 192.168.0.200）
iRes += MC_Reset();//复位板卡

//iRes = MC_SetSoftLimit(1,lSoftLimPos,lSoftLimNeg);

//注意，板卡默认上电是没有软限位的
//设置轴 1 正软限位为 10000(单位脉冲)，负软限位为 10000(单位脉冲)
iRes = MC_SetSoftLimit(1,10000,-10000);

//这样就相当于关闭软限位
iRes = MC_SetSoftLimit(1,2147483647,-2147483648);

iRes = MC_LmtsOn(1,-1);//将第一个轴的正硬限位和负硬限位都有效
iRes = MC_EStopSetIO(0,0,0,10);//设置主卡通用输入 X0 为紧急停止 IO,滤波时间 10ms
iRes = MC_EStopOnOff(1);//紧急停止功能打开
iRes = MC_EStopGetSts(&nStatus);//获取紧急停止触发状态
iRes = MC_EStopClrSts();//清除紧急停止状态
```

5.8、其他指令 API

API	说明
MC_ZeroPos	清零轴的规划和编码器位置
MC_GetID	获取板卡唯一标识 ID
MC_SetAxisBand	设置轴到位误差带
MC_SetBacklash	设置反向间隙补偿的相关参数
MC_GetBacklash	读取反向间隙补偿的相关参数
MC_SetStopDec	设置平滑停止减速度和急停减速度
MC_WriteInterFlash	写板卡内部 Flash
MC_ReadInterFlash	读板卡内部 Flash
MC_DownPitchErrorTable	下载螺距误差补偿表
MC_ReadPitchErrorTable	读取螺距误差补偿表
MC_AxisErrPitchOn	打开指定螺距误差补偿
MC_AxisErrPitchOff	关闭指定轴螺距误差补偿
MC_SetPrfPos	修改规划（脉冲）位置，修改时，轴不能处于运动状态
MC_SetEncPos	修改编码器位置，修改时，轴不能处于运动状态

参数详细说明：

int MC_ZeroPos(short nAxisNum,short nCount=1)	
nAxisNum	需要位置清零的起始轴号,取值范围：[1,AXIS_MAX_COUNT]
nCount	需要位置清零的轴数
int MC_SetAxisBand(short nAxisNum,long lBand,long lTime)	
nAxisNum	轴号
band	误差带大小，单位：脉冲（默认值：3）
time	误差带保持时间，单位：ms（默认值：3）
int MC_SetBacklash(short nAxisNum,long lCompValue,double dCompChangeValue,long lCompDir)	
nAxisNum	需要进行反向间隙补偿的轴的编号，取值范围：[1,8]
lCompValue	反向间隙补偿值，当为 0 时表示没有使能反向间隙补偿功能，取值只能为正，范围：[0, 1073741824]，单位：脉冲
dCompChangeValue	反向间隙补偿的变化量，取值只能为正，范围：[0, 1073741824]，单位：脉冲/毫秒 当该参数的值为 0 或者大于等于 lCompValue 时，则反向间隙的补偿量将瞬间叠加在规划位置上，没有渐变的过程
lCompDir	反向间隙补偿方向 0：只补偿负方向，当电机向负方向运动时，将施加补偿量，电机向正方向运动时，不施加补偿量 1：只补偿正方向，当电机向正方向运动时，将施加补偿量，电机向负方向运动时，不施加补偿量
int MC_GetBacklash(short nAxisNum,long *pCompValue,double *pCompChangeValue,long *pCompDir)	
nAxisNum	nAxisNum 查询的轴号，取值范围：[1,AXIS_MAX_COUNT]
pCompValue	pCompValue 读取的反向间隙补偿值
pCompChangeValue	pCompChangeValue 读取的反向间隙补偿值的变化量

pCompDir	pCompDir 读取的反向间隙补偿的补偿方向
int MC_SetStopDec(short nAxisNum,double decSmoothStop,double decAbruptStop)	
nAxisNum	轴编号
DecSmoothStop	平滑停止减速度，单位：脉冲/毫秒/毫秒，建议范围 0.1~2，默认 0.5
DecAbruptStop	急停减速度，单位：脉冲/毫秒/毫秒，建议范围 1~20，默认 1
int MC_WriteInterFlash(unsigned char* pData,short nLength)	
pData	数据指针
nLength	数据长度，不能大于 500
int MC_ReadInterFlash(unsigned char*pData,short nLength)	
pData	数据指针
nLength	数据长度，不能大于 500
int MC_DownPitchErrorTable(short nTableNum,short nPointNum,long lStartPos,long lEndPos,short *pErrValue1,short *pErrValue2)	
nTableNum	表号，1~16
nPointNum	补偿点个数，2~1024
lStartPos	起始位置，单位脉冲
lEndPos	结束位置，单位脉冲
pErrValue1	StartPos-EndPos 补偿表，单位脉冲
pErrValue2	EndPos-StartPos 补偿表，单位脉冲
int MC_ReadPitchErrorTable(short nTableNum,short* pPointNum,long* pStartPos,long* pEndPos,short *pErrValue1,short *pErrValue2)	
nTableNum	表号，1~16
pPointNum	补偿点个数，2~1024
pStartPos	起始位置，单位脉冲
pEndPos	结束位置，单位脉冲
pErrValue1	StartPos-EndPos 补偿表，单位脉冲
pErrValue2	EndPos-StartPos 补偿表，单位脉冲
int MC_AxisErrPitchOn(short nAxisNum)	
nAxisNum	轴号
int MC_AxisErrPitchOff(short nAxisNum)	
nAxisNum	轴号
int MC_SetPrfPos(short nAxisNum,long lPrfPos)	
nAxisNum	轴编号
lPrfPos	规划(脉冲)位置
int MC_SetEncPos(short nEncodeNum,long lEncPos)	
nAxisNum	轴编号
lEncPos	编码器位置

示例代码：

```
int iRes = 0;
long lSts = 0;
```

```
unsigned long ulID = 0;

long lSoftLimPos= 0X7FFFFFFF;
long lSoftLimNeg=0X80000000

iRes += MC_SetCardNo(1);//切换到第 1 块板卡
iRes += MC_Open(0,"192.168.0.200");//打开板卡（通过网口，PC 端 IP 地址为 192.168.0.200）
iRes += MC_Reset();//复位板卡

iRes += MC_ZeroPos(1,8);//清零所有 8 个轴的位置
iRes += MC_SetAxisBand(1,3,5)//设置轴 1 到位误差带为 3 个脉冲，保持时间为 5ms

//设置轴 1 反向间隙为 3 个脉冲，补偿速率 2 脉冲/ms，反向运动时补偿
iRes = MC_SetBacklash(1,3,2,0);

iRes = MC_GetID(&ulID);


short ErrValue1[2000];
short ErrValue2[2000];

//任意初始化了一个螺距误差表(这里仅仅用于测试，实际应由激光干涉仪生成)
for(int i=0; i<=1000; i++)
{
    ErrValue1[i] = i;
    ErrValue2[i] = i;
}

//下载轴 2 的螺距误差表，点数量 1001，起始位置 10000，终止位置 20000
iRes = MC_DownPitchErrorTable(2,1001,10000,20000,&ErrValue1[0],&ErrValue2[0]);

//启用轴 2 螺距误差表
iRes = MC_AxisErrPitchOn(2);
```

5.9、插补运动指令 API

API	说明
MC_SetCrdPrm	设置坐标系参数，确立坐标系映射，建立坐标系
MC_SetCrdPrmSingleEX	用于代替 MC_SetCrdPrm 函数，方便不擅长结构体的客户使用。
MC_GetCrdPrm	查询坐标系参数
MC_InitLookAhead	配置指定坐标系指定 FifoIndex 前瞻缓冲区的拐弯速率，最大加速度，缓冲区深度，缓冲区指针等参数
MC_InitLookAheadSingle	用于替代 MC_InitLookAhead 函数，方便不擅长结构体的客户使用。
MC_InitLookAheadSingleEX	用于替代 MC_InitLookAhead 函数，方便不擅长结构体的客户使用。
MC_CrdClear	清除插补缓存区内的插补数据
MC_LnXY	缓存区指令，两维直线插补
MC_LnXYZ	缓存区指令，三维直线插补
MC_LnXYZA	缓存区指令，四维直线插补
MC_LnXYZAB	缓存区指令，五维直线插补
MC_LnXYZABC	缓存区指令，六维直线插补
MC_LnAll	缓存区指令，7 维~14 维直线插补
MC_ArcXYC	缓存区指令，XY 平面圆弧插补（以终点坐标和圆心位置为输入参数）
MC_ArcXZC	缓存区指令，XZ 平面圆弧插补（以终点坐标和圆心位置为输入参数）
MC_ArcYZC	缓存区指令，YZ 平面圆弧插补（以终点坐标和圆心位置为输入参数）
MC_HelixXYCZ	缓存区指令，XY 平面螺旋线插补（以终点坐标和圆心位置为输入参数）
MC_HelixXZCY	缓存区指令，XZ 平面螺旋线插补（以终点坐标和圆心位置为输入参数）
MC_HelixYZCX	缓存区指令，YZ 平面螺旋线插补（以终点坐标和圆心位置为输入参数）
MC_HelixXYCCount	缓存区指令，XY 平面螺旋线插补（以终点坐标和圆心位置为输入参数）
MC_HelixXZCCount	缓存区指令，XZ 平面螺旋线插补（以终点坐标和圆心位置为输入参数）
MC_HelixYZCCount	缓存区指令，YZ 平面螺旋线插补（以终点坐标和圆心位置为输入参数）
MC_BufPWM	缓存区指令，设置 PWM 频率及占空比
MC_BufIO	缓存区指令，设置 IO 输出
MC_BufIOReverse	缓存区指令，设置 IO 输出一个指定时间的脉冲
MC_BufWaitIO	缓存区指令，等待 IO 输入
MC_BufCmpPluse	缓存区指令，在插补运动中，插入立即比较输出
MC_BufDelay	缓存区指令，延时一段时间
MC_BufMoveVel	在插补运动的过程中插入 BufferMove 轴的速度设定
MC_BufMoveAcc	在插补运动的过程中插入 BufferMove 轴的加速度设定
MC_BufMove	在插补运动的过程中插入阻塞和非阻塞的点位运动。 重要提示：已经参与插补坐标系的轴不能再使用 BufMove
MC_BufGear	设定了脉冲输出的个数。它会保证与其后紧挨的指令同时启动，同时停止
MC_CrdData	向插补缓存区增加插补数据
MC_CrdStart	启动插补运动
MC_SetOverride	设置插补运动目标合成速度倍率
MC_GetCrdPos	查询该坐标系的当前坐标位置值
MC_CrdSpace	读取插补缓存区中的剩余空间
MC_CrdStatus	查询插补运动坐标系状态

MC_SetUserSegNum	缓存区指令，设置自定义插补段段号
MC_GetUserSegNum	读取自定义插补段段号
MC_GetRemainderSegNum	读取未完成的插补段段数
MC_GetLookAheadSpace	获取前瞻缓冲区剩余空间
MC_GetLookAheadSegCount	获取前瞻缓存区剩余段数
MC_GetCrdVel	查询该坐标系的当前合成速度值
MC_SetCrdStopDec	设置插补运动平滑停止和急停快慢

参数详细说明：

int MC_SetCrdPrm(short nCrdNum,TCrdPrm *pCrdPrm)	
nCrdNum	坐标系号，取值范围：[1,CRDSYS_MAX_COUNT]
pCrdPrm	typedef struct CrdPrm { short dimension; short profile[8]; double synVelMax; double synAccMax; short evenTime; short setOriginFlag; long originPos[8]; }TCrdPrm; dimension：坐标系的维数，取值范围：[1,4]。 Profile[8]：坐标系与规划器的映射关系，每个元素的取值范围：[0,4] synVelMax：最大合成速度。取值范围：(0,4000)，单位：pulse/ms synAccMax：最大合成加速度。取值范围：(0,4000)，单位：pulse/(ms*ms) evenTime：最小匀速段时间。取值范围：[0,32767)，单位：ms setOriginFlag：表示是否需要指定坐标系的原点坐标的规划位置 0：不需要指定原点坐标值，则坐标系的原点在当前规划位置上； 1：需要指定原点坐标值，坐标系的原点在 originPos 指定的规划位置上 originPos[8]：指定的坐标系原点的规划位置值
int MC_SetCrdPrmSingleEX(short nCrdNum,short dimension,short profile0,short profile1,short profile2,short profile3,short profile4,short profile5,short profile6,short profile7,double synVelMax,double synAccMax,short evenTime,short setOriginFlag,long originPos0,long originPos1,long originPos2,long originPos3,long originPos4,long originPos5,long originPos6,long originPos7)	
nCrdNum	nCrdNum:坐标系号，取值范围：[1,CRDSYS_MAX_COUNT]
dimension	dimension：坐标系的维数，取值范围：[1,5]。
profile0	坐标系 X 轴号，取值范围：[1,8]
Profile1	坐标系 Y 轴号，取值范围：[1,8]，用不到可以设置为 0
Profile2	坐标系 Z 轴号，取值范围：[1,8]，用不到可以设置为 0
Profile3	坐标系 A 轴号，取值范围：[1,8]，用不到可以设置为 0
Profile4	坐标系 B 轴号，取值范围：[1,8]，用不到可以设置为 0
Profile5	保留，固定为 0
Profile6	保留，固定为 0
Profile7	保留，固定为 0

synVelMax	该坐标系的最大合成速度。取值范围：(0,4000)，单位：pulse/ms
synAccMax	该坐标系的最大合成加速度。取值范围：(0,4000)，单位：pulse/(ms*ms)
evenTime	每个插补段的最小匀速段时间。取值范围：[0,32767)，单位：ms
setOriginFlag	表示是否需要指定坐标系的原点坐标的规划位置。 0：不需要指定原点坐标值，则坐标系的原点在当前规划位置上 1：需要指定原点坐标值，按照后面的 originPos 设定原点坐标值
originPos0	X 轴的原点坐标位置，单位脉冲
originPos1	Y 轴的原点坐标位置，单位脉冲
originPos2	Z 轴的原点坐标位置，单位脉冲
originPos3	A 轴的原点坐标位置，单位脉冲
originPos4	B 轴的原点坐标位置，单位脉冲
originPos5	保留，固定为 0
originPos6	保留，固定为 0
originPos7	保留，固定为 0
int MC_GetCrdPrm(short nCrdNum,TCrdPrm *pCrdPrm)	
nCrdNum	坐标系号，取值范围：[1,CRDSYS_MAX_COUNT]
pCrdPrm	pCrdPrm： typedef struct CrdPrm { short dimension; short profile[8]; double synVelMax; double synAccMax; short evenTime; short setOriginFlag; long originPos[8]; }TCrdPrm; dimension：坐标系的维数，取值范围：[1,4]。 Profile[8]：坐标系与规划器的映射关系，每个元素的取值范围：[0,4] synVelMax：该坐标系的最大合成速度。取值范围：(0,4000)，单位：pulse/ms synAccMax：该坐标系的最大合成加速度。取值范围：(0,4000)，单位：pulse/(ms*ms) evenTime：每个插补段的最小匀速段时间。取值范围：[0,32767)，单位：ms setOriginFlag：表示是否需要指定坐标系的原点坐标的规划位置： 0：不需要指定原点坐标值，则坐标系的原点在当前规划位置上； 1：需要指定原点坐标值，坐标系的原点在 originPos 指定的规划位置上 originPos[8]：指定的坐标系原点的规划位置值
int MC_InitLookAhead(short nCrdNum,short FifoIndex,TLookAheadPrm* plookAheadPara)	
nCrdNum	坐标系编号
FifoIndex	FifoIndex 索引

plookAheadPara	plookAheadPara：
	typedef struct _LookAheadPrm
	{
	int lookAheadNum; //前瞻段数
	TCrdData *pLookAheadBuf; //前瞻缓冲区指针

	double dSpeedMax[INTERPOLATION_AXIS_MAX]; //各轴的最大速度(p/ms)
	double dAccMax[INTERPOLATION_AXIS_MAX]; //各轴的最大加速度
	double dMaxStepSpeed[INTERPOLATION_AXIS_MAX];//各轴最大速度变化量（相当于启动速度）
	double dScale[INTERPOLATION_AXIS_MAX]; //各轴的脉冲当量
	}TLookAheadPrm;

int MC_InitLookAheadSingle(short nCrdNum,short FifoIndex,int lookAheadNum,double* dSpeedMax,double* dAccMax,double *dMaxStepSpeed,double *dScale)	
nCrdNum	坐标系编号，通常为 1
FifoIndex	FifoIndex 索引，通常为 0
lookAheadNum	前瞻段数，通常为 50~200
dSpeedMax	数组指针，数组长度为 6，存放各轴的最大速度(p/ms)
dAccMax	数组指针，数组长度为 6，存放各轴的最大加速度
dMaxStepSpeed	数组指针，数组长度为 6，各轴的最大速度变化量（相当于启动速度）
dScale	数组指针，数组长度为 6，存放各轴的脉冲当量（固定为 1）
int MC_InitLookAheadSingleEX(short nCrdNum,short FifoIndex,int lookAheadNum,double dSpeedMax0,double dSpeedMax1,double dSpeedMax2,double dSpeedMax3,double dSpeedMax4,double dAccMax0,double dAccMax1,double dAccMax2,double dAccMax3,double dAccMax4,double dMaxStepSpeed0,double dMaxStepSpeed1,double dMaxStepSpeed2,double dMaxStepSpeed3,double dMaxStepSpeed4,double dScale0,double dScale1,double dScale2,double dScale3,double dScale4)	
nCrdNum	坐标系号，通常为 1
FifoIndex	缓冲区索引，通常为 0
lookAheadNum	前瞻段数，通常为 200
dSpeedMax0	X 轴最大速度，单位脉冲/毫秒，通常为 1000
dSpeedMax1	Y 轴最大速度，单位脉冲/毫秒，通常为 1000
dSpeedMax2	Z 轴最大速度，单位脉冲/毫秒，通常为 1000
dSpeedMax3	A 轴最大速度，单位脉冲/毫秒，通常为 1000
dSpeedMax4	B 轴最大速度，单位脉冲/毫秒，通常为 1000
dAccMax0	X 轴最大加速度，单位脉冲/毫秒/毫秒，通常为 0.1~10，建议 1
dAccMax1	Y 轴最大加速度，单位脉冲/毫秒/毫秒，通常为 0.1~10，建议 1
dAccMax2	Z 轴最大加速度，单位脉冲/毫秒/毫秒，通常为 0.1~10，建议 1
dAccMax3	A 轴最大加速度，单位脉冲/毫秒/毫秒，通常为 0.1~10，建议 1
dAccMax4	B 轴最大加速度，单位脉冲/毫秒/毫秒，通常为 0.1~10，建议 1
dMaxStepSpeed0	X 轴最大速度变化量，单位脉冲/毫秒，通常为 0.1~20，建议 2
dMaxStepSpeed1	Y 轴最大速度变化量，单位脉冲/毫秒，通常为 0.1~20，建议 2
dMaxStepSpeed2	Z 轴最大速度变化量，单位脉冲/毫秒，通常为 0.1~20，建议 2
dMaxStepSpeed3	A 轴最大速度变化量，单位脉冲/毫秒，通常为 0.1~20，建议 2
dMaxStepSpeed4	B 轴最大速度变化量，单位脉冲/毫秒，通常为 0.1~20，建议 2
dScale0	X 轴分量比例，通常固定为 1
dScale1	Y 轴分量比例，通常固定为 1
dScale2	Z 轴分量比例，通常固定为 1
dScale3	A 轴分量比例，通常固定为 1
dScale4	B 轴分量比例，通常固定为 1

int MC_CrdClear(short nCrdNum,short FifoIndex)	
nCrdNum	坐标系编号
FifoIndex	FifoIndex 索引
int MC_LnXY(short nCrdNum,long x,long y,double synVel,double synAcc,double velEnd=0,short FifoIndex=0,long segNum = 0)	
nCrdNum	坐标系号，取值范围：[1,CRDSYS_MAX_COUNT]
x	插补段 x 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。
y	插补段 y 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。
synVel	插补段的目标合成速度。取值范围：(0,4000)，单位：pulse/ms。
synAcc	插补段的合成加速度。取值范围：(0,4000)，单位：pulse/(ms*ms)。
velEnd	插补段的终点速度。取值范围：[0,4000)，单位：pulse/ms。该值只有在没有使用前瞻预处理功能时才有意义，否则该值无效。默认为：0
FifoIndex	插补缓存区号，取值范围：[0,1]，默认为：0
segNum	用户自定义行号
int MC_LnXYZ(short nCrdNum,long x,long y,long z,double synVel,double synAcc,double velEnd=0,short FifoIndex=0,long segNum = 0)	
nCrdNum	坐标系号，取值范围：[1,CRDSYS_MAX_COUNT]
x	插补段 x 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。
y	插补段 y 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。
z	插补段 z 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。
synVel	插补段的目标合成速度。取值范围：(0,4000)，单位：pulse/ms。
synAcc	插补段的合成加速度。取值范围：(0, 4000)，单位：pulse/(ms*ms)。
velEnd	插补段的终点速度。取值范围：[0, 4000)，单位：pulse/ms。该值只有在没有使用前瞻预处理功能时才有意义，否则该值无效。默认为：0
FifoIndex	插补缓存区号，取值范围：[0,1]，默认为：0
segNum	用户自定义行号
int MC_LnXYZA(short nCrdNum,long x,long y,long z,long a,double synVel,double synAcc,double velEnd=0,short FifoIndex=0,long segNum = 0)	
nCrdNum	坐标系号，取值范围：[1,CRDSYS_MAX_COUNT]
x	插补段 x 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。
y	插补段 y 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。
z	插补段 z 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。
a	插补段 a 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。
synVel	插补段的目标合成速度。取值范围：(0, 4000)，单位：pulse/ms。
synAcc	插补段的合成加速度。取值范围：(0, 4000)，单位：pulse/(ms*ms)。
velEnd	插补段的终点速度。取值范围：[0, 4000)，单位：pulse/ms。该值只有在没有使用前瞻预处理功能时才有意义，否则该值无效。默认为：0
FifoIndex	插补缓存区号，取值范围：[0,1]，默认为：0
segNum	用户自定义行号
int MC_LnXYZAB(short nCrdNum,long x,long y,long z,long a,long b,double synVel,double synAcc,double velEnd=0,short FifoIndex=0,long segNum = 0)	
nCrdNum	坐标系号，取值范围：[1,CRDSYS_MAX_COUNT]
x	插补段 x 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。

y	插补段 y 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。
z	插补段 z 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。
a	插补段 a 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。
b	插补段 b 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。
synVel	插补段的目标合成速度。取值范围：(0, 4000)，单位：pulse/ms。
synAcc	插补段的合成加速度。取值范围：(0, 4000)，单位：pulse/(ms*ms)。
velEnd	插补段的终点速度。取值范围：[0, 4000)，单位：pulse/ms。该值只有在没有使用前瞻预处理功能时才有意义，否则该值无效。默认为：0
FifoIndex	插补缓存区号，取值范围：[0,1]，默认为：0
segNum	用户自定义行号
int MC_LnXYZABC(short nCrdNum,long x,long y,long z,long a,long b, long c,double synVel,double synAcc,double velEnd=0,short FifoIndex=0,long segNum = 0)	
nCrdNum	坐标系号，取值范围：[1,CRDSYS_MAX_COUNT]
x	插补段 x 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。
y	插补段 y 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。
z	插补段 z 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。
a	插补段 a 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。
b	插补段 b 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。
c	插补段 c 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。
synVel	插补段的目标合成速度。取值范围：(0, 4000)，单位：pulse/ms。
synAcc	插补段的合成加速度。取值范围：(0, 4000)，单位：pulse/(ms*ms)。
velEnd	插补段的终点速度。取值范围：[0, 4000)，单位：pulse/ms。该值只有在没有使用前瞻预处理功能时才有意义，否则该值无效。默认为：0
FifoIndex	插补缓存区号，取值范围：[0,1]，默认为：0
segNum	用户自定义行号
int MC_LnAll(short nCrdNum,long* pPos,short nDim, double synVel,double synAcc,double velEnd,short FifoIndex=0,long segNum = 0)	
nCrdNum	坐标系号，取值范围：[1,CRDSYS_MAX_COUNT]
pPos	插补各轴坐标。取值范围：[-1073741823, 1073741823]，单位：pulse。
nDim	参与插补轴数量
synVel	插补段的目标合成速度。取值范围：(0, 4000)，单位：pulse/ms。
synAcc	插补段的合成加速度。取值范围：(0, 4000)，单位：pulse/(ms*ms)。
velEnd	插补段的终点速度。取值范围：[0, 4000)，单位：pulse/ms。该值只有在没有使用前瞻预处理功能时才有意义，否则该值无效。默认为：0
FifoIndex	插补缓存区号，取值范围：[0,1]，默认为：0
segNum	用户自定义行号
int MC_ArcXYC(short nCrdNum,long x,long y,double xCenter,double yCenter,short circleDir,double synVel,double synAcc,double velEnd=0,short FifoIndex=0,long segNum = 0)	
nCrdNum	坐标系号，取值范围：[1,CRDSYS_MAX_COUNT]
x	圆弧插补 x 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。
y	圆弧插补 y 轴终点坐标值。取值范围：[-1073741823, 1073741823]，单位：pulse。
xCenter	圆弧插补的圆心 x 方向相对于起点位置的偏移量
yCenter	圆弧插补的圆心 Y 方向相对于起点位置的偏移量
circleDir	圆弧的旋转方向 0 顺时针圆弧 1 逆时针圆弧

synVel	插补段的目标合成速度。取值范围：(0, 4000)，单位：pulse/ms。
synAcc	插补段的合成加速度。取值范围：(0, 4000)，单位：pulse/(ms*ms)。
velEnd	插补段的终点速度。取值范围：[0, 4000)，单位：pulse/ms。该值只有在没有使用前瞻预处理功能时才有意义，否则该值无效。默认为：0
FifoIndex	插补缓存区号，取值范围：[0,1]，默认为：0
segNum	用户自定义行号
int MC_ArcXZC(short nCrdNum,long x,long z,double xCenter,double zCenter,short circleDir,double synVel,double synAcc,double velEnd=0,short FifoIndex=0,long segNum = 0)	
nCrdNum	坐标系号，取值范围：[1,CRDSYS_MAX_COUNT]
x	圆弧插补 x 轴终点坐标值。取值范围:[-1073741823, 1073741823],单位:pulse。
z	圆弧插补 z 轴终点坐标值。取值范围:[-1073741823, 1073741823],单位:pulse。
xCenter	圆弧插补的圆心 x 方向相对于起点位置的偏移量
zCenter	圆弧插补的圆心 z 方向相对于起点位置的偏移量
circleDir	圆弧的旋转方向 0 顺时针圆弧 1 逆时针圆弧
synVel	插补段的目标合成速度。取值范围：(0, 4000)，单位：pulse/ms。
synAcc	插补段的合成加速度。取值范围：(0, 4000)，单位：pulse/(ms*ms)。
velEnd	插补段的终点速度。取值范围：[0, 4000)，单位：pulse/ms。该值只有在没有使用前瞻预处理功能时才有意义，否则该值无效。默认为：0
FifoIndex	插补缓存区号，取值范围：[0,1]，默认为：0
segNum	用户自定义行号
int MC_ArcYZC(short nCrdNum,long y,long z,double yCenter,double zCenter,short circleDir,double synVel,double synAcc,double velEnd=0,short FifoIndex=0,long segNum = 0)	
nCrdNum	坐标系号，取值范围：[1,CRDSYS_MAX_COUNT]
y	圆弧插补 y 轴终点坐标值。取值范围:[-1073741823, 1073741823],单位:pulse。
z	圆弧插补 z 轴终点坐标值。取值范围:[-1073741823, 1073741823],单位:pulse。
yCenter	圆弧插补的圆心 y 方向相对于起点位置的偏移量
zCenter	圆弧插补的圆心 z 方向相对于起点位置的偏移量
circleDir	圆弧的旋转方向 0 顺时针圆弧 1 逆时针圆弧
synVel	插补段的目标合成速度。取值范围：(0, 4000)，单位：pulse/ms。
synAcc	插补段的合成加速度。取值范围：(0, 4000)，单位：pulse/(ms*ms)。
velEnd	插补段的终点速度。取值范围：[0, 4000)，单位：pulse/ms。该值只有在没有使用前瞻预处理功能时才有意义，否则该值无效。默认为：0
FifoIndex	插补缓存区号，取值范围：[0,1]，默认为：0
segNum	用户自定义行号
int MC_HelixXYCZ(short nCrdNum,long x,long y,long z,double xCenter,double yCenter,float k, short circleDir,double synVel,double synAcc,double velEnd=0,short FifoIndex=0,long segNum=-1)	
nCrdNum	坐标系号，取值范围：[1,CRDSYS_MAX_COUNT]
x	X 轴终点坐标
y	Y 轴终点坐标
z	Z 轴终点坐标
xCenter	圆弧插补的圆心 X 方向相对于起点位置的偏移量
yCenter	圆弧插补的圆心 Y 方向相对于起点位置的偏移量
k	螺距

circleDir	圆弧的旋转方向,0 顺时针圆弧,1 逆时针圆弧
synVel	插补段的目标合成速度。取值范围：(0, 4000)，单位：pulse/ms。
synAcc	插补段的合成加速度。取值范围：(0, 4000)，单位：pulse/(ms*ms)。
velEnd	插补段的终点速度。取值范围：[0, 4000)，单位：pulse/ms。该值只有在没有使用前瞻预处理功能时才有意义，否则该值无效。默认为：0
FifoIndex	插补缓存区号，取值范围：[0,1]，默认为：0
segNum	用户自定义段号
int MC_HelixXZCY(short nCrdNum,long x,long y,long z,double xCenter,double yCenter,float k, short circleDir,double synVel,double synAcc,double velEnd=0,short FifoIndex=0,long segNum=-1)	
nCrdNum	坐标系号，取值范围：[1,CRDSYS_MAX_COUNT]
x	X 轴终点坐标
y	Y 轴终点坐标
z	Z 轴终点坐标
xCenter	圆弧插补的圆心 X 方向相对于起点位置的偏移量
yCenter	圆弧插补的圆心 Y 方向相对于起点位置的偏移量
k	螺距
circleDir	圆弧的旋转方向,0 顺时针圆弧,1 逆时针圆弧
synVel	插补段的目标合成速度。取值范围：(0, 4000)，单位：pulse/ms。
synAcc	插补段的合成加速度。取值范围：(0, 4000)，单位：pulse/(ms*ms)。
velEnd	插补段的终点速度。取值范围：[0, 4000)，单位：pulse/ms。该值只有在没有使用前瞻预处理功能时才有意义，否则该值无效。默认为：0
FifoIndex	插补缓存区号，取值范围：[0,1]，默认为：0
segNum	用户自定义段号
int MC_HelixYZCX(short nCrdNum,long x,long y,long z,double xCenter,double yCenter,float k, short circleDir,double synVel,double synAcc,double velEnd=0,short FifoIndex=0,long segNum=-1)	
nCrdNum	坐标系号，取值范围：[1,CRDSYS_MAX_COUNT]
x	X 轴终点坐标
y	Y 轴终点坐标
z	Z 轴终点坐标
xCenter	圆弧插补的圆心 X 方向相对于起点位置的偏移量
yCenter	圆弧插补的圆心 Y 方向相对于起点位置的偏移量
k	螺距
circleDir	圆弧的旋转方向,0 顺时针圆弧,1 逆时针圆弧
synVel	插补段的目标合成速度。取值范围：(0, 4000)，单位：pulse/ms。
synAcc	插补段的合成加速度。取值范围：(0, 4000)，单位：pulse/(ms*ms)。
velEnd	插补段的终点速度。取值范围：[0, 4000)，单位：pulse/ms。该值只有在没有使用前瞻预处理功能时才有意义，否则该值无效。默认为：0
FifoIndex	插补缓存区号，取值范围：[0,1]，默认为：0
segNum	用户自定义段号
int MC_HelixXYCCount(short nCrdNum,double xCenter,double yCenter,float k,float CirlceCount, short circleDir,double synVel,double synAcc,double velEnd=0,short FifoIndex=0,long segNum=-1)	

nCrdNum	坐标系号，取值范围：[1,CRDSYS_MAX_COUNT]
xCenter	圆弧插补的圆心 X 方向相对于起点位置的偏移量，单位脉冲
yCenter	圆弧插补的圆心 Y 方向相对于起点位置的偏移量，单位脉冲
k	螺距，单位脉冲
CirlceCount	旋转圈数，可以为小数
circleDir	圆弧的旋转方向,0 顺时针圆弧,1 逆时针圆弧
synVel	插补段的目标合成速度。取值范围：(0, 4000)，单位：pulse/ms。
synAcc	插补段的合成加速度。取值范围：(0, 4000)，单位：pulse/(ms*ms)。
velEnd	插补段的终点速度。取值范围：[0, 4000)，单位：pulse/ms。该值只有在没有使用前瞻预处理功能时才有意义，否则该值无效。默认为：0
FifoIndex	插补缓存区号，取值范围：[0,1]，默认为：0
segNum	用户自定义段号
int MC_HelixXZCCount(short nCrdNum,double xCenter,double zCenter,float k,float CirlceCount, short circleDir,double synVel,double synAcc,double velEnd=0,short FifoIndex=0,long segNum=-1)	
nCrdNum	坐标系号，取值范围：[1,CRDSYS_MAX_COUNT]
xCenter	圆弧插补的圆心 X 方向相对于起点位置的偏移量，单位脉冲
zCenter	圆弧插补的圆心 Z 方向相对于起点位置的偏移量，单位脉冲
k	螺距，单位脉冲
CirlceCount	旋转圈数，可以为小数
circleDir	圆弧的旋转方向,0 顺时针圆弧,1 逆时针圆弧
synVel	插补段的目标合成速度。取值范围：(0, 4000)，单位：pulse/ms。
synAcc	插补段的合成加速度。取值范围：(0, 4000)，单位：pulse/(ms*ms)。
velEnd	插补段的终点速度。取值范围：[0, 4000)，单位：pulse/ms。该值只有在没有使用前瞻预处理功能时才有意义，否则该值无效。默认为：0
FifoIndex	插补缓存区号，取值范围：[0,1]，默认为：0
segNum	用户自定义段号
int MC_HelixYZCCount(short nCrdNum,double yCenter,double zCenter,float k,float CirlceCount, short circleDir,double synVel,double synAcc,double velEnd=0,short FifoIndex=0,long segNum=-1)	
nCrdNum	坐标系号，取值范围：[1,CRDSYS_MAX_COUNT]
yCenter	圆弧插补的圆心 Y 方向相对于起点位置的偏移量，单位脉冲
zCenter	圆弧插补的圆心 Z 方向相对于起点位置的偏移量，单位脉冲
k	螺距，单位脉冲
CirlceCount	旋转圈数，可以为小数
circleDir	圆弧的旋转方向,0 顺时针圆弧,1 逆时针圆弧
synVel	插补段的目标合成速度。取值范围：(0, 4000)，单位：pulse/ms。
synAcc	插补段的合成加速度。取值范围：(0, 4000)，单位：pulse/(ms*ms)。
velEnd	插补段的终点速度。取值范围：[0, 4000)，单位：pulse/ms。该值只有在没有使用前瞻预处理功能时才有意义，否则该值无效。默认为：0
FifoIndex	插补缓存区号，取值范围：[0,1]，默认为：0
segNum	用户自定义段号
int MC_BufPWM(short nCrdNum,short nPwmNum ,double dFreq,double dDuty,short nFifoIndex,long lUserSegNum=-1)	

nCrdNum	坐标系号，取值范围：[1,CRDSYS_MAX_COUNT]
nPwmNum	PWM 通道，从 1 开始
dFreq	频率，单位 HZ
dDuty	占空比，0~100
nFifoIndex	插补缓存区号，取值范围：[0,1]，默认为：0
lUserSegNum	用户自定义行号
int MC_BufIO(short nCrdNum,unsigned short doType,unsigned short nCardIndex,unsigned short doMask,unsigned short doValue,short FifoIndex=0,long segNum = 0)	
nCrdNum	坐标系号，取值范围：[1,CRDSYS_MAX_COUNT]
doType	指定数字 IO 类型 MC_GPO(该宏定义为 12) 通用输出
nCardIndex	板卡索引(0~63),0 是主模块，扩展模块从 1 开始
doMask	缓存区 IO 的输出控制掩码
doValue	缓存区 IO 的输出值
FifoIndex	插补缓存区号，取值范围：[0,1]，默认为：0
segNum	用户自定义行号

例程：
MC_BufIO(1,12,0,0X0001,0X0001,0,0);//打开 Y0
MC_BufIO(1,12,0,0X0001,0,0,0);//关闭 Y0
 	
//0X0008 的 Bit3 是 1，代表要对 Y3 进行操作
MC_BufIO(1,12,0,0X0008,0X0008,0,0);//打开 Y3
MC_BufIO(1,12,0,0X0008,0,0,0);//关闭 Y3
 	
//0X0009 的 Bit3 和 Bit0 是 1，代表要对 Y0 和 Y3 进行操作
MC_BufIO(1,12,0,0X0009,0X0009,0,0);//打开 Y3 和 Y0
MC_BufIO(1,12,0,0X0009,0,0,0);//关闭 Y3 和 Y0	

int MC_BufIOReverse(short nCrdNum,unsigned short nDoType,unsigned short nCardIndex,unsigned short doMask,unsigned short doValue,unsigned short nReverseTime,short FifoIndex=0,long segNum = 0)	
nCrdNum	坐标系号，取值范围：[1,CRDSYS_MAX_COUNT]
nDoType	固定为 12
nCardIndex	固定为 0
doMask	缓存区 IO 的输出控制掩码
doValue	缓存区 IO 的输出值
nReverseTime	持续时间，单位毫秒
FifoIndex	固定为 0
segNum	用户自定义行号
int MC_BufWaitIO(short nCrdNum,unsigned short nCardIndex,unsigned short nIOPortIndex,unsigned short nLevel,unsigned long lWaitTimeMS,unsigned short nFilterTime,short FifoIndex,long segNum)	
nCrdNum	坐标系号，取值范围：[1,CRDSYS_MAX_COUNT]
nCardIndex	板卡索引(0~63),0 是主模块，扩展模块从 1 开始
nIOPortIndex:	IO 索引，0~15

nLevel:	1 等待有信号输入，0 等待信号消失
lWaitTimeMS:	等待超时时间,单位 ms(超过该时间，会自动执行下一条命令，0 是一直等待
nFilterTime:	滤波时间，单位 ms。连续在这段时间检测到信号，才认为真正有信号
FifoIndex	插补缓存区号，取值范围：[0,1]，默认为：0
segNum:	用户自定义行号
int MC_BufCmpPluse(short nCrdNum,short nChannel,short nPluseType,short nTime,short nTimerFlag,short nFifoIndex,long lSegNum)	
nCrdNum	坐标系号，取值范围：[1,CRDSYS_MAX_COUNT]
nChannel	通道号，1 或者 2
nPluseType	通道输出类型，0 低电平，1 高电平，2 脉冲
nTime	脉冲时间
nTimerFlag	0us，1 毫秒
nFifoIndex	固定为 0
lSegNum	用户自定义行号
坐标系 1 插入一行指令，控制比较输出通道 2，输出一个 100us 的脉冲 MC_BufCmpPluse(1,2,2,100,0,0,0) 坐标系 1 插入一行指令，控制比较输出通道 1，输出一个 10ms 的脉冲 MC_BufCmpPluse(1,1,2,10,1,0,0)	
int MC_BufDelay(short nCrdNum,unsigned long ulDelayTime,short FifoIndex=0,long segNum)	
nCrdNum	坐标系号，取值范围：[1,CRDSYS_MAX_COUNT]
ulDelayTime	延时时间，单位 ms
FifoIndex	插补缓存区号，取值范围：[0,1]，默认为：0
segNum	用户自定义行号
int MC_BufMoveVel(short nCrdNum,short nAxisMask,float* pVel,short nFifoIndex,long lSegNum);	
nCrdNum	坐标系号。正整数，取值范围： [1, CRDSYS_MAX_COUNT]。
nAxisMask	需要进行点位运动的轴掩码，每一位代表一个轴，该位为 1 表示对应轴需要设定速度。该轴不能处于坐标系中。
pVel	点位运动的速度，单位： pulse/ms，这里传入的是一个数组长度为 8 的地址指针
nFifoIndex	插补缓存区号。正整数，取值范围： [0, 1]，默认值为： 0。
lSegNum	用户自定义段号
int MC_BufMoveAcc(short nCrdNum,short nAxisMask,float* pAcc,short nFifoIndex,long lSegNum);	
nCrdNum	坐标系号。正整数，取值范围： [1, CRDSYS_MAX_COUNT]。
nAxisMask	需要进行点位运动的轴掩码，每一位代表一个轴，该位为 1 表示对应轴需要设定加速度。该轴不能处于坐标系中。
pAcc	点位运动的加速度，单位： pulse/ms/ms，这里传入的是一个数组长度为 8 的地址指针
nFifoIndex	插补缓存区号。正整数，取值范围： [0, 1]，默认值为： 0。
lSegNum	用户自定义段号
int MC_BufMove(short nCrdNum,short nAxisMask,long* pPos,short nModalMask,short nFifoIndex,long lSegNum);	
nCrdNum	坐标系号。正整数，取值范围： [1, CRDSYS_MAX_COUNT]。

nAxisMask	需要进行点位运动的轴掩码，每一位代表一个轴，该位为 1 表示对应轴需要设定目标位置。该轴不能处于坐标系中。
pPos	点位运动目标位置，单位： pulse，这里传入的是一个数组长度为 8 的地址指针
nModalMask	点位运动的模式，每一位代表一个轴。 0:该指令为非阻塞指令，即不阻塞后续的插补缓存区指令的执行。 1:该指令为阻塞指令，将会阻塞后续的插补缓存区指令的执行。
nFifoIndex	插补缓存区号。正整数，取值范围： [0, 1]，默认值为： 0。
lSegNum	用户自定义段号
int MC_BufGear(short nCrdNum,short nAxisMask,long* pPos,short nFifoIndex,long lSegNum);	
nCrdNum	坐标系号。正整数，取值范围： [1, CRDSYS_MAX_COUNT]。
nAxisMask	需要进行点位运动的轴掩码，每一位代表一个轴，该位为 1 表示对应轴需要设定跟随量。该轴不能处于坐标系中。
pPos	跟随运动进给量，单位： pulse，这里传入的是一个数组长度为 8 的地址指针
nFifoIndex	插补缓存区号。正整数，取值范围： [0, 1]，默认值为： 0。
lSegNum	用户自定义段号
int MC_CrdData(short nCrdNum,void *pCrdData,short FifoIndex=0)	
nCrdNum	坐标系号，取值范围：[1,CRDSYS_MAX_COUNT]
pCrdData	插补数据
FifoIndex	插补缓存区号，取值范围：[0,1]，默认为：0
int MC_CrdStart(short mask,short option)	
mask	从 bit0~bit1 按位表示需要启动的坐标系，其中，bit0 对应坐标系 1，bit1 对应坐标系 2；0：不启动该坐标系，1：启动该坐标系。
option	从 bit0~bit1 按位表示坐标系需要启动的缓存区的编号，其中，bit0 对应坐标系 1,bit1 对应坐标系 2;0:启动坐标系中 FIFO0 的运动,1:启动坐标系中 FIFO1 的运动。
重点说明	函数返回值含义如下： MN，M 代表返回值十位，N 代表返回值个位 M 代表轴号 N 代表失败类型 N=1:轴 M 不在插补模式，无法启动，原因是未建立坐标系或者中途进入其他模式 N=2:轴 M 报警，无法启动坐标系 N=3:轴 M 急停，无法启动坐标系 N=4:轴 M 正软限位触发，无法启动坐标系 N=5:轴 M 正硬限位触发，无法启动坐标系 N=6:轴 M 负软限位触发，无法启动坐标系 N=7:轴 M 负硬限位触发，无法启动坐标系 N=8:轴 M 跟随误差超限，无法启动坐标系
int MC_SetOverride(short nCrdNum,double synVelRatio)	
nCrdNum	坐标系号，取值范围：[1,CRDSYS_MAX_COUNT]
synVelRatio	设置的插补目标速度倍率，取值范围：(0,1]，系统默认该值为：1。
int MC_GetCrdPos(short nCrdNum,double *pPos)	
nCrdNum	坐标系号，取值范围：[1,CRDSYS_MAX_COUNT]
pPos	读取的坐标系的坐标值，单位：pulse。该参数应该为一个数组首元素的指针，数组的元素个数取决于该坐标系的维数。
int MC_CrdSpace(short nCrdNum,long *pSpace,short FifoIndex=0)	

nCrdNum	坐标系号，取值范围：[1,CRDSYS_MAX_COUNT]
pSpace	读取插补缓存区中的剩余空间
FifoIndex	插补缓存区号，取值范围：[0,1]，默认为：0
int MC_CrdStatus(short nCrdNum,short *pCrdSts,long *pSegment,short FifoIndex=0)	
nCrdNum	坐标系号，取值范围：[1,CRDSYS_MAX_COUNT]

pCrdSts	读取插补运动状态
	//坐标系状态位定义（注意位判断用逻辑与，不要用逻辑等于）
	#define CRDSYS_STATUS_PROG_RUN (0x00000001)//启动中
	#define CRDSYS_STATUS_PROG_STOP (0x00000002)//平滑停止中
	#define CRDSYS_STATUS_PROG_ESTOP (0x00000004)//紧急停止中
	#define CRDSYS_STATUS_FIFO_FINISH_0 (0x00000010)//板卡 FIFO-0 数据已执行完毕的状态位
	#define CRDSYS_STATUS_FIFO_FINISH_1 (0x00000020)//板卡 FIFO-1 数据已执行完毕的状态位

pSegment	读取当前已经完成的插补段数，当重新建立坐标系或者调用 MC_CrdClear 指令后，该值会被清零
FifoIndex	所要查询运动状态的 FifoIndex 号，取值范围：[0,1]，默认为：0
int MC_SetUserSegNum(short nCrdNum,long segNum,short FifoIndex=0)	
nCrdNum	坐标系号，取值范围：[1,CRDSYS_MAX_COUNT]
segNum	设置用户自定义的插补段段号
FifoIndex	插补缓存区号，取值范围：[0,1]，默认为：0
int MC_GetUserSegNum(short nCrdNum,long *pSegment,short FifoIndex=0)	
nCrdNum	坐标系号，取值范围：[1,CRDSYS_MAX_COUNT]
pSegment	读取的用户自定义的插补段段号
FifoIndex	插补缓存区号，取值范围：[0,1]，默认为：0
int MC_GetRemainderSegNum(short nCrdNum,long *pSegment,short FifoIndex=0)	
nCrdNum	坐标系号，取值范围：[1,CRDSYS_MAX_COUNT]
pSegment	读取的剩余插补段的段数
FifoIndex	插补缓存区号，取值范围：[0,1]，默认为：0
int MC_GetLookAheadSpace(short nCrdNum,long *pSpace,short nFifoIndex=0)	
nCrdNum	坐标系编号
pSpace	读取前瞻缓存区中的剩余空间
nFifoIndex	nFifoIndex 索引
int MC_GetLookAheadSegCount(short nCrdNum,long *pSegCount,short nFifoIndex=0)	
nCrdNum	坐标系编号
pSegCount	前瞻缓存区剩余段数
nFifoIndex	FifoIndex 索引
int MC_GetCrdVel(short nCrdNum,double *pSynVel)	
nCrdNum	坐标系编号
pSynVel	速度，单位脉冲/毫秒
int MC_SetCrdStopDec(short nCrdNum,double decSmoothStop,double decAbruptStop)	
nCrdNum	坐标系编号，从 1 开始
decSmoothStop	坐标系平滑停止系数，取值范围 0.0001~1，数值越大，停止越快
decAbruptStop	坐标系紧急停止系数，取值范围 0.0001~1，数值越大，停止越快

5.10、硬件捕获类 API

API	说明
MC_SetCaptureMode	设置编码器捕获方式，并启动捕获
MC_GetCaptureMode	读取编码器捕获方式
MC_GetCaptureStatus	读取编码器捕获状态
MC_SetCaptureSense	设置捕获电平
MC_GetCaptureSense	获取捕获电平
MC_ClearCaptureStatus	清除捕获状态

参数详细说明：

int MC_SetCaptureMode(short nEncodeNum,short nMode)	
nEncodeNum	编码器轴号
nMode	编码器捕获模式 1：Home 捕获 2：Index 捕获 3：探针捕获
int MC_GetCaptureMode(short nEncodeNum,short *pMode,short nCount=1)	
nEncodeNum	编码器起始轴号
pMode	编码器捕获模式
nCount	读取的轴数，默认为 1 ，1 次最多可以读取 8 个编码器轴
int MC_GetCaptureStatus(short nEncodeNum,short *pStatus,long *pValue,short nCount=1,unsigned long *pClock=NULL)	
nEncodeNum	编码器起始轴号
pStatus	读取编码器捕获状态 为 1 时表示对应轴捕获触发
pValue	读取编码器捕获值，当捕获触发时，编码器捕获值会自动更新
nCount	读取的轴数，默认为 1 ，1 次最多可以读取 8 个编码器轴
pClock	读取控制器时钟
int MC_SetCaptureSense(short nEncodeNum,short nMode ,short nSence)	
nEncodeNum	编码器起始轴号
nMode	捕获模式 1：Home 捕获 2：Index 捕获
nSence	捕获电平 0：下降沿触发 1：上升沿触发
int MC_GetCaptureSense(short nEncodeNum,short nMode ,short *pSence)	
nEncodeNum	编码器号
nMode	捕获模式 1：Home 捕获 2：Index 捕获
pSence	捕获电平，0 或者 1 0：下降沿触发 1：上升沿触发
int MC_ClearCaptureStatus(short nEncodeNum)	

nEncodeNum	需要被清除捕获状态的编码器轴号

5.11、Gear/电子齿轮类 API

API	说明
MC_PrfGear	设置指定轴进入电子齿轮模式
MC_SetGearMaster	设置电子齿轮运动跟随主轴
MC_GetGearMaster	读取电子齿轮运动跟随主轴
MC_SetGearRatio	设置电子齿轮比
MC_GetGearRatio	获取电子齿轮比
MC_GearStart	启动电子齿轮运动
MC_GearStop	停止电子齿轮运动
MC_SetGearEvent	设置电子齿轮触发事件
MC_GetGearEvent	获取电子齿轮触发事件
MC_SetGearIntervalTime	设置电子齿轮平滑系数
MC_GetGearIntervalTime	获取电子齿轮平滑系数

参数详细说明：

int MC_PrfGear(short nAxisNum,short nDir=0)	
nAxisNum	轴号
nDir	跟随方向，0：双向跟随，1：正向跟随，-1：负向跟随
int MC_SetGearMaster(short nAxisNum,short nMasterAxisNum,short nMasterType=2)	
nAxisNum	轴号
nMasterAxisNum	主轴号
nMasterType	主轴类型，2，跟随规划，1,跟随编码器
int MC_GetGearMaster(short nAxisNum,short *pnMasterAxisNum,short *pMasterType=NULL)	
nAxisNum	轴号
pnMasterAxisNum	主轴号
pMasterType	主轴类型，2，跟随规划，1,跟随编码器
int MC_SetGearRatio(short nAxisNum,long lMasterEven,long lSlaveEven,long lMasterSlope=0,long lStopSmoothTime = 200)	
nAxisNum	轴号
lMasterEven	传动比系数,主轴位移，单位：pulse
lSlaveEven	传动比系数,从轴位移，单位：pulse
lMasterSlope	主轴离合区位移,单位： pulse 取值范围：不能小于 0 或者等于 1
lStopSmoothTime	脱离离合区缓冲时间，单位：ms
int MC_GetGearRatio(short nAxisNum,long *pMasterEven,long *pSlaveEven,long *pMasterSlope,long *pStopSmoothTime)	
nAxisNum	轴号
lMasterEven	传动比系数,主轴位移，单位：pulse
lSlaveEven	传动比系数,从轴位移，单位：pulse
lMasterSlope	主轴离合区位移,主轴离合区位移,主轴离合区位移,主轴离合区位移,单位：pulse
lStopSmoothTime	脱离离合区缓冲时间，单位：ms
int MC_GearStart(long lMask)	

lMask	按位指示需要启动 Gear 运动的轴号。当 bit 位为 1 时表示启动对应的轴 Bit7、6、5、4、3、2、1、0 对应 8 轴、7 轴、6 轴、5 轴、4 轴、3 轴、2 轴、1 轴
int MC_GearStop(long lAxisMask,long lEMGMask)	
lMask	按位指示需要停止 Gear 运动的轴号。 当 bit 位为 1 时表示停止对应的轴 Bit7~0 对应 8~1 轴
lEMGMask	按位指示需要立即停止 Gear 运动的轴号。 当 bit 位为 1 时表示立即停止对应的轴，忽略平滑停止时间 当 bit 位为 0 时表示平滑停止对应的轴，StopSmoothTime 时间内平滑停止 Bit7~0 对应 8~1 轴
int MC_SetGearEvent(short nAxisNum,short nEvent,double startPara0,double startPara1)	
nAxisNum	轴号
nEvent	事件，具体参见头文件 #define GEAR_EVENT_IMMED 1 #define GEAR_EVENT_BIG_EQU 2 #define GEAR_EVENT_SMALL_EQU 3 #define GEAR_EVENT_IO_ON 4 #define GEAR_EVENT_IO_OFF 5 GEAR_EVENT_IMMED ： 立即启动电子齿轮 GEAR_EVENT_BIG_EQU:主轴规划或者编码器位置大于等于指定数值时启动电子齿轮 GEAR_EVENT_SMALL_EQU:主轴规划或者编码器位置小于等于指定数值时启动电子齿轮 GEAR_EVENT_IO_ON：指定 IO 为 ON 时启动电子齿轮 GEAR_EVENT_IO_OFF：指定 IO 为 OFF 时启动电子齿轮
startPara0	startPara0：事件参数 0 GEAR_EVENT_BIG_EQU 和 GEAR_EVENT_SMALL_EQU 时代表编码器或者规划值 GEAR_EVENT_IO_ON 和 GEAR_EVENT_IO_OFF 时代表 IO 端口号
startPara1	保留
int MC_GetGearEvent(short nAxisNum,short *pEvent,double *pStartPara0,double *pStartPara1)	
nAxisNum	轴号
nEvent	事件，具体参见头文件 #define GEAR_EVENT_IMMED 1 #define GEAR_EVENT_BIG_EQU 2 #define GEAR_EVENT_SMALL_EQU 3 #define GEAR_EVENT_IO_ON 4 #define GEAR_EVENT_IO_OFF 5 GEAR_EVENT_IMMED ： 立即启动电子齿轮 GEAR_EVENT_BIG_EQU:主轴规划或者编码器位置大于等于指定数值时启动电子齿轮 GEAR_EVENT_SMALL_EQU:主轴规划或者编码器位置小于等于指定数值时启动电子齿轮 GEAR_EVENT_IO_ON：指定 IO 为 ON 时启动电子齿轮

	GEAR_EVENT_IO_OFF：指定 IO 为 OFF 时启动电子齿轮
startPara0	startPara0：事件参数 0 GEAR_EVENT_BIG_EQU 和 GEAR_EVENT_SMALL_EQU 时代表编码器或者规划值 GEAR_EVENT_IO_ON 和 GEAR_EVENT_IO_OFF 时代表 IO 端口号
startPara1	保留
int MC_SetGearIntervalTime(short nAxisNum,short nIntervalTime)	
nAxisNum	从轴轴号
nIntervalTime	滤波系数，2~32768，越大越平滑不抖动，越小跟随性越好误差小，需要平衡
int MC_GetGearIntervalTime(short nAxisNum,short *nIntervalTime)	
nAxisNum	从轴轴号
nIntervalTime	滤波系数，2~32768，越大越平滑不抖动，越小跟随性越好误差小，需要平衡

例程代码：

```
//设置轴 3 轴 4 进入电子齿轮模式，双向跟随
MC_PrfGear(3,0);
MC_PrfGear(4,0);

//设置轴 3 轴 4 跟随轴 1 的编码器
MC_SetGearMaster(3,1,1);
MC_SetGearMaster(4,1,1);

//设置轴 3 跟随比例为 1 比 1
MC_SetGearRatio(3,1,1,0,0);
//设置轴 4 跟随比例为 1 比 1
MC_SetGearRatio(4,1,1,0,0);

//设置轴 3 跟随触发方式为立即开始
MC_SetGearEvent(3,1,0,0)
//设置轴 4 跟随触发方式为立即开始
MC_SetGearEvent(4,1,0,0)

//启动轴 3 和轴 4 的跟随
MC_GearStart(0X0C);
```

5.12、电子凸轮类 API

API	说明
MC_PrfCam	设置指定轴进入电子凸轮模式
MC_SetCamMaster	设置电子凸轮运动跟随主轴
MC_GetCamMaster	读取电子凸轮运动跟随主轴
MC_SetCamEvent	设置电子凸轮触发事件
MC_GetCamEvent	获取电子凸轮触发事件
MC_SetCamIntervalTime	设置电子凸轮均值滤波时间
MC_GetCamIntervalTime	获取电子凸轮均值滤波时间
MC_SetUpCamTable	建立凸轮表
MC_DownCamTable	下载电子凸轮表到控制器
MC_CamStart	启动电子凸轮运动
MC_CamStop	停止电子凸轮运动

参数详细说明：

int MC_PrfCam(short nAxisNum,short nTableNum)	
nAxisNum	轴号
nTableNum	凸轮表编号，1~16
int MC_SetCamMaster(short nAxisNum,short nMasterAxisNum,short nMasterType)	
nAxisNum	轴号
nMasterAxisNum	主轴号
nMasterType	主轴类型，1：跟随编码器 2：跟随规划
int MC_GetCamMaster(short nAxisNum,short *pnMasterAxisNum,short *pMasterType=NULL)	
nAxisNum	轴号
pnMasterAxisNum	主轴号
pMasterType	主轴类型，1：跟随编码器 2：跟随规划
int MC_SetCamEvent(short nAxisNum,short nEvent,double startPara0,double startPara1)	
nAxisNum	轴号
nEvent	1：立即启动电子齿轮 2：主轴规划或者编码器位置大于等于指定数值时启动凸轮 3：主轴规划或者编码器位置小于等于指定数值时启动凸轮 4：指定 IO 为 ON 时启动电子齿轮 5：指定 IO 为 OFF 时启动电子齿轮
startPara0	事件参数 0 nEvent=2 或者 3 时代表编码器或者规划值 nEvent=4 或者 5 时代表 IO 端口号
startPara1	保留
int MC_GetCamEvent(short nAxisNum,short *pEvent,double *pStartPara0,double *pStartPara1)	
nAxisNum	轴号
pEvent	1：立即启动电子齿轮 2：主轴规划或者编码器位置大于等于指定数值时启动凸轮 3：主轴规划或者编码器位置小于等于指定数值时启动凸轮 4：指定 IO 为 ON 时启动电子齿轮 5：指定 IO 为 OFF 时启动电子齿轮

pStartPara0	事件参数 0 nEvent=2 或者 3 时代表编码器或者规划值 nEvent=4 或者 5 时代表 IO 端口号
pStartPara1	保留
int MC_SetCamIntervalTime(short nAxisNum,short nIntervalTime)	
nAxisNum	轴号
nIntervalTime	平滑时间，单位 ms
int MC_GetCamIntervalTime(short nAxisNum,short *nIntervalTime)	
nAxisNum	轴号
nIntervalTime	平滑时间，单位 ms
int MC_SetUpCamTable(short nCamTableNum,long lMasterValueMax, long *plSlaveCamData, long lCamTableLen)	
nCamTableNum	表号，1~16
lMasterValueMax	一个周期主轴最大值，单位脉冲
plSlaveCamData	从轴数据存放指针，数据单位：脉冲
lCamTableLen	表长度
int MC_DownCamTable(int *pProgress)	
pProgress	下载进度
int MC_CamStart(long lMask)	
LMask	lMask 按位指示需要启动 Cam 运动的轴号。当 bit 位为 1 时表示启动对应的轴 Bit7、6、5、4、3、2、1、0 对应 8 轴、7 轴、6 轴、5 轴、4 轴、3 轴、2 轴、1 轴
int MC_CamStop(long lAxisMask,long lEMGMask)	
lAxisMask	lMask 按位指示需要停止 Cam 运动的轴号。当 bit 位为 1 时表示启动对应的轴 Bit7、6、5、4、3、2、1、0 对应 8 轴、7 轴、6 轴、5 轴、4 轴、3 轴、2 轴、1 轴
lEMGMask	lEMGMask 按位指示需要停止 Cam 运动的轴号。当 bit 位为 1 时表示立即停止对应的轴 Bit7、6、5、4、3、2、1、0 对应 8 轴、7 轴、6 轴、5 轴、4 轴、3 轴、2 轴、1 轴

5.13、比较输出(飞拍)类 API

API	说明
MC_CmpPluse	设置比较器输出 IO 立即输出指定电平或者脉冲
MC_CmpBufSetChannel	设置比较缓冲区对应输出通道
MC_CmpBufData	向比较器缓冲区发送比较数据
MC_CmpBufSts	获取比较器缓冲区状态
MC_CmpBufStop	停止比较器缓冲区
MC_CmpRpt	设置比较器缓冲区等比输出
MC_CmpSetTriggerCount	设置比较器缓冲区触发计数初值
MC_CmpGetTriggerCount	获取比较器缓冲区触发计数初值

参数详细说明：

int MC_CmpPluse(short nChannelMask, short nPluseType1, short nPluseType2, short nTime1,short nTime2, short nTimeFlag1, short nTimeFlag2)	
nChannel	bit0 表示通道 1，bit1 表示通道 2
nPluseType1	通道 1 输出类型，0 低电平 1 高电平 2 脉冲
nPluseType2	通道 2 输出类型，0 低电平 1 高电平 2 脉冲
nTime1	通道 1 脉冲持续时间
nTime2	通道 2 脉冲持续时间
nTimeFlag1	比较器 1 的脉冲时间单位，0:us，1:ms
nTimeFlag2	比较器 2 的脉冲时间单位，0:us，1:ms
int MC_CmpBufSetChannel(short nBuf1ChannelNum,short nBuf2ChannelNum);	
nBuf1ChannelNum	比较缓冲区 1 对应输出通道号，默认为通道 1，可设置为 1 或者 2
nBuf2ChannelNum	比较缓冲区 2 对应输出通道号，默认为通道 2，可设置为 1 或者 2
int MC_CmpBufData(short nCmpEncodeNum, short nPluseType, short nStartLevel, short nTime, long *pBuf1, short nBufLen1, long *pBuf2, short nBufLen2,short nAbsPosFlag=0)	
nCmpEncodeNum	轴号
nPluseType	2 表示输出脉冲，1 表示反转电平。
nStartLevel	按位设置比较器输出的初始电平，bit0 比较器 11，bit1 比较器 2； 0：表示初始为低电平； 1：表示初始为高电平
nTime	输出脉冲时，该参数用来设定脉冲输出宽度，取值范围：[1, 65535]，单位：us，输出电平时，该参数无效
pBuf1	比较器 1 数据缓冲区，位置值为相对当前位置的距离
nBufLen1	比较器 1 数据缓冲区长度，0~128
pBuf2	比较器 2 数据缓冲区，位置值为相对当前位置的距离
nBufLen2	比较器 2 数据缓冲区长度，0~128
nAbsPosFlag	0：相对当前位置 1：绝对位置
int MC_CmpBufSts(short *pStatus,unsigned short *pCount1,unsigned short *pCount2)	
pStatus	按位指示比较器状态 bit0 代表比较器 1，bit1 代表比较器 2 0 代表板卡比较缓冲区数据已空，1 代表板卡比较缓冲区数据未完成
pCount1	比较器 1 板卡缓冲区剩余待比较数据
pCount2	比较器 2 板卡缓冲区剩余待比较数据
int MC_CmpBufStop(short nChannel)	
nChannel	bit0 代表通道 1，bit1 代表通道 2

int MC_CmpRpt(short nCmpEncodeNum,short nChannel,long lStartPos, long lRptTime, long lInterval, short nTime,short nPluseType,short nAbsPosFlag)	
nCmpEncodeNum	编码器通道
nChannel	bit0 代表比较缓冲区 1，bit1 代表比较缓冲区 2
lStartPos	起始位置,单位：脉冲
lRptTime	重复次数
lInterval	位置间隔，单位：脉冲
nTime	脉冲输出时，脉冲持续时间，单位 us
nPluseType	通道输出类型，0 低电平 1 高电平 2 脉冲
nAbsPosFlag	0：相对当前位置 1：绝对位置
int MC_CmpSetTriggerCount(unsigned long lTriggerCount1,unsigned long lTriggerCount2)	
lTriggerCount1	比较器 1 触发计数初值
lTriggerCount2	比较器 2 触发计数初值
int MC_CmpGetTriggerCount(unsigned long* plTriggerCount1,unsigned long* plTriggerCount2)	
plTriggerCount1	比较器 1 触发计数值
plTriggerCount2	比较器 2 触发计数值

示例代码：

```
int iRes = 0;

//下面代码控制比较输出端口输出高电平（手动测试用）。
//第 1 个参数为 1 代表通道 1
//第 2 个参数为 1，代表立即输出高电平
//第 3 个参数预留，固定为 1，无意义
//第 4 个参数代表脉冲时间，这里因为是输出高电平，并非脉冲，所以无意义
//第 5 个参数为预留，跟第 4 个参数相同即可
//第 6 个参数为时间单位，0 代表微秒，1 代表毫秒（这里因为是输出高电平，所以参数 6 无意义）
//第 7 个参数为预留，与第 6 个参数相同即可
iRes = MC_CmpPluse(1,1,1,100,100,0,0);

//下面代码控制比较输出端口输出低电平（手动测试用）。
//第 1 个参数为 1 代表通道 1
//第 2 个参数为 0，代表立即输出低电平
//第 3 个参数预留，固定为 1，无意义
//第 4 个参数代表脉冲时间，这里因为是输出高电平，并非脉冲，所以无意义
//第 5 个参数为预留，跟第 4 个参数相同即可
//第 6 个参数为时间单位，0 代表微秒，1 代表毫秒（这里因为是输出高电平，所以参数 6 无意义）
//第 7 个参数为预留，与第 6 个参数相同即可
iRes = MC_CmpPluse(1,0,1,100,100,0,0);

//下面代码控制比较输出端口输出一个 200ms 的脉冲（手动测试用）。
//第 1 个参数为 1 代表通道 1
//第 2 个参数为 0，代表立即输出低电平
//第 3 个参数预留，固定为 1，无意义
//第 4 个参数代表脉冲时间，这里因为是输出高电平，并非脉冲，所以无意义
```

```
//第 5 个参数为预留，跟第 4 个参数相同即可
//第 6 个参数为时间单位，0 代表微秒，1 代表毫秒
//第 7 个参数为预留，与第 6 个参数相同即可
iRes = MC_CmpPluse(1,2,1,200,200,1,1);

//下面代码控制比较输出端口 1 在相对轴 3 当前位置为 10000、20000、30000、40000、50000 时分表输出一个持续时间为 10ms 的脉冲（自动流程用）。
int iRes = 0;
//定义一个长度为 5 的数组，存储待比较数据点，如果只有一个位置要比较，就把数组长度设为 1
long lBufData[5] = {10000,20000,30000,40000,50000};

//第 1 个参数为 3 代表使用轴 3 的编码器位置进行比较
//第 2 个参数为 2 代表输出类型为脉冲(1 代表反转电平，2 代表脉冲)
//第 3 个参数为 0 代表初始电平为低电平（1 代表初始电平为高电平）
//第 4 个参数为 10 代表脉冲持续时间为 10 微秒或者 10 毫秒（取决于最后一个参数）
//第 5 个参数代表待比较数据点存放指针
//第 6 个参数为 5 代表待比较数据长度为 5 个数据点
//第 7 个参数固定为 NULL
//第 8 个参数固定为 0
//第 9 个参数为 0 代表坐标点为相对坐标（1 代表绝对坐标）
//第 10 个参数为 1 代表时间单位为毫秒（0 代表时间单位为微秒）
iRes = MC_CmpBufData(3,2,0,10,&lBufData[0],5,NULL,0,0,1);
```

5.14、自动回零相关 API

API	说明
MC_HomeStart	启动轴回零
MC_HomeStop	停止轴回零 注意，如果回零没有正常完成，必须调用该函数结束回零，否则轴不能运动
MC_HomeSetPrm	设置回零参数
MC_HomeSetPrmSingle	设置回零参数（非结构体方式，可以代替 MC_HomeSetPrm）
MC_HomeGetPrm	获取回零参数
MC_HomeGetPrmSingle	获取回零参数（非结构体方式，可以代替 MC_HomeGetPrm）
MC_HomeGetSts	获取回零状态

参数详细说明：

int MC_HomeStart(short iAxisNum)	
nAxisNum	需要回零的轴号,取值范围：[1,AXIS_MAX_COUNT]
int MC_HomeStop(short iAxisNum)	
nAxisNum	需要停止回零的轴号,取值范围：[1,AXIS_MAX_COUNT]
int MC_HomeSetPrm(short iAxisNum,TAxisHomePrm *pAxisHomePrm)	
nAxisNum	需要设置回零参数的轴号,取值范围：[1,AXIS_MAX_COUNT]

pAxisHomePrm	//轴回零参数
	typedef struct _AxisHomeParm{
	short nHomeMode; //1:HOME 回原点 2:HOME 加 Index 回原点 3:Index 回零
	short nHomeDir; //回零方向，1:正向回零，0:负向回零
	long lOffset; //回零偏移，回到零位后再走一个 Offset 作为零位
	double dHomeRapidVel; //回零快移速度，单位：Pluse/ms
	double dHomeLocatVel; //回零定位速度，单位：Pluse/ms
	double dHomeIndexVel; //回零寻找 INDEX 速度，单位：Pluse/ms
	double dHomeAcc; //回零使用的加速度，单位 Pluse/ms/ms
	}TAxisHomePrm;

int MC_HomeSetPrmSingle(short iAxisNum,short nHomeMode,short nHomeDir,long lOffset,double dHomeRapidVel,double dHomeLocatVel,double dHomeIndexVel,double dHomeAcc)	
nAxisNum	需要设置回零参数的轴号,取值范围：[1,AXIS_MAX_COUNT]
nHomeMode	1:HOME 回原点,此回零方式最常用 2:HOME 加 Index 回原点（仅支持带 Index 的驱动） 3:Index 回原点（仅支持带 Index 的驱动）
nHomeDir	回零方向，1:正向回零，0:负向回零
lOffset	回零偏移，回到零位后再走一个 Offset 作为零位,通常该参数为 0
dHomeRapidVel	回零快移速度，单位：Pluse/ms
dHomeLocatVel	回零定位速度，单位：Pluse/ms
dHomeIndexVel	回零寻找 INDEX 速度，单位：Pluse/ms
dHomeAcc	回零使用的加速度，单位 Pluse/ms/ms
int MC_HomeGetPrm(short iAxisNum,TAxisHomePrm *pAxisHomePrm)	
nAxisNum	需要获取回零参数的轴号,取值范围：[1,AXIS_MAX_COUNT]
pAxisHomePrm	//轴回零参数 typedef struct _AxisHomeParm{

	short nHomeMode; //1--HOME 回原点 2--HOME 加 Index 回原点
	short nHomeDir; //回零方向，1-正向回零，0-负向回零
	long lOffset; //回零偏移，回到零位后再走一个 Offset 作为零位
	double dHomeRapidVel; //回零快移速度，单位：Pluse/ms
	double dHomeLocatVel; //回零定位速度，单位：Pluse/ms
	double dHomeIndexVel; //回零寻找 INDEX 速度，单位：Pluse/ms
	double dHomeAcc; //回零使用的加速度
	}TAxisHomePrm;

int MC_HomeGetPrmSingle(short iAxisNum,short *nHomeMode,short *nHomeDir,long *lOffset,double* dHomeRapidVel,double* dHomeLocatVel,double* dHomeIndexVel,double* dHomeAcc)	
nAxisNum	需要获取回零参数的轴号,取值范围：[1,AXIS_MAX_COUNT]
nHomeMode	1--HOME 回原点 2--HOME 加 Index 回原点
nHomeDir	回零方向，1-正向回零，0-负向回零
lOffset	回零偏移，回到零位后再走一个 Offset 作为零位
dHomeRapidVel	回零快移速度，单位：脉冲/毫秒
dHomeLocatVel	回零定位速度，单位：脉冲/毫秒
dHomeIndexVel	回零寻找 INDEX 速度，单位：脉冲/毫秒
dHomeAcc	回零使用的加速度，单位：脉冲/毫秒/毫秒
int MC_HomeGetSts(short iAxisNum,unsigned short* pStatus)	
nAxisNum	需要获取回零状态的轴号,取值范围：[1,AXIS_MAX_COUNT]
pStatus	pStatus：指向回零状态的指针 0：尚未回零 1：回零中 2：回零成功

示例代码：

```
int iRes = 0;
short nStatus = 0;

TAxisHomePrm AxisHomePrm;

AxisHomePrm.nHomeMode = 1;//回零模式为 HOME 回原点
AxisHomePrm.nHomeDir = 0;//回零方向为负向回零
AxisHomePrm.dHomeRapidVel = 5;//回零快移速度，5 脉冲/毫秒
AxisHomePrm.dHomeLocatVel = 1;//回零定位速度，1 脉冲/毫秒
AxisHomePrm.dHomeAcc = 0.5;//回零使用的加速度，0.5 脉冲/毫秒/毫秒

AxisHomePrm.lOffset = 0;//回零偏移
//设置轴 1 回零参数
iRes = MC_HomeSetPrm(1,&AxisHomePrm);
//启动轴 1 回零
iRes = MC_HomeStart(1);
//获取轴 1 回零状态
iRes = MC_HomeGetSts(1,&nStatus);
```

5.15、PT 模式相关 API

API	说明
MC_PrfPt	设置指定轴为 PT 模式
MC_PtSpace	读取指定轴的 PT 缓冲区空闲存储空间
MC_PtRemain	读取指定轴的 PT 存储空间尚未执行的数据长度
MC_PtData	向指定轴的 PT 缓冲区发送数据
MC_PtClear	清除指定轴 PT 缓冲区的数据
MC_PtStart	启动 PT 模式运动

参数详细说明：

int MC_PrfPt(short nAxisNum,short mode=PT_MODE_STATIC)	
nAxisNum	轴编号，1、2、3、4.....
mode	PT_MODE_STATIC 静态 PT_MODE_DYNAMIC 动态
int MC_PtSpace(short nAxisNum,long *pSpace,short nCount)	
nAxisNum	轴编号，1、2、3、4.....
pSpace	空闲存储空间的数据长度存放指针
nCount	一次获取的 PT 空间个数（1~2）
int MC_PtRemain(short nAxisNum,long *pRemainSpace,short nCount)	
nAxisNum	轴编号，1、2、3、4.....
pRemainSpace	尚未执行的数据长度存放指针
nCount	一次获取的 PT 空间个数（1~2）
int MC_PtData(short nAxisNum,short* pData,long lLength,double dDataID)	
nAxisNum	轴编号，1、2、3、4.....
pData	数据存放指针
lLength	数据长度
nDataID	数据标识，每一包数据都应该和上一包不同，否则会被丢弃 1~4 循环。
int MC_PtClear(long lAxisMask)	
nAxisNum	轴掩码一个位代表一个轴，0X0001 代表轴 1，0X0007 表轴 1 轴 2 和轴 3
int MC_PtStart(long lAxisMask)	
nAxisNum	轴掩码一个位代表一个轴，0X0001 代表轴 1，0X0007 表轴 1 轴 2 和轴 3

示例代码：

```
int iRes = 0;int iAxisNum = 1;int g_dDataID = 1;
//声明一个长度为 100 的数组
short nData[100];

//设置轴 1 为 PT 模式
iRes = MC_PrfPt(iAxisNum );

//压入 PT 数据
iRes = MC_PtData(iAxisNum,nData,100,g_dDataID);g_dDataID++;

//启动 PT 运动
MC_PtStart(0X0001 << (iAxisNum-1));
```

5.16、手轮相关 API(支持手轮接口的型号可用)

API	说明
MC_GetDiRaw	获取手轮轴选和倍率 IO
MC_StartHandwheel	指定轴开始手轮模式
MC_EndHandwheel	指定轴结束手轮模式

参数详细说明：

int MC_GetDiRaw(short nDiType,long *pValue)	
nDiType 读取手轮 IO 时，该参数固定为 7	指定数字 IO 类型 MC_LIMIT_POSITIVE(该宏定义为 0) 正限位 MC_LIMIT_NEGATIVE(该宏定义为 1) 负限位 MC_ALARM(该宏定义为 2) 驱动报警 MC_HOME(该宏定义为 3) 原点开关 MC_GPI(该宏定义为 4) 通用输入 MC_MPG(该宏定义为 7) 手轮 IO 输入
pValue	IO 输入值存放指针

int MC_StartHandwheel(short nAxisNum,short nMasterAxisNum,long lMasterEven,long lSlaveEven,short nIntervalTime,double dAcc,double dDec,double dVel,short nStopWaitTime)	
nAxisNum	轴编号
nMasterAxisNum	跟随轴号，通常为 9
lMasterEven	跟随比例系数，可以先设定为 1，然后逐渐调整到合适比例。
lSlaveEven	跟随比例系数，可以先设定为 1，然后逐渐调整到合适比例。
nIntervalTime	固定为 0
dAcc	固定为 0.1
dDec	固定为 0.1
dVel	固定为 50
nStopWaitTime	固定为 0
int MC_EndHandwheel(short nAxisNum)	
nAxisNum	轴编号

重点说明：

1、手轮接线图参见章节 3.2 硬件接口说明

2、手轮 X/Y/Z/A/B/X1/X10/X100 的 IO 读取参见“5.3、IO 操作 API”章节。使用 MC_GetDiRaw 函数读取。

3、X100 通常不用接。默认 X1 和 X10 读不到的情况下就是 X100,这样可以节约 IO 端口。

4、X/Y/Z/A/B/X1/X10/X100 这些 IO 信号读取后，需要自行在代码里面处理哪些轴进入手轮模式以及比例。

5、假设 X 轴当前在手轮模式，程序检测到用户切换到 Y 轴了，那么 Y 轴进入手轮模式时，X 一定要退出手轮模式。

6、手轮倍率切换时，该轴必须先退出手轮模式才能修改倍率

5.17、串口/485 相关 API(可选项)

API	说明
MC_UartConfig	设置串口波特率、数据长度、校验和、停止位长度等通讯相关参数
MC_SendEthToUartString	通过以太网转串口发送数据
MC_ReadUartToEthString	读取串口转以太网数据

参数详细说明：

int MC_UartConfig(unsigned short nUartNum, unsigned long uLBaudRate,unsigned short nDataLength,unsigned short nVerifyType,unsigned short nStopBitLen)	
nUartNum	串口号（485 的串口号为 3）
uLBaudRate	波特率
nDataLength	数据长度，7 或者 8
nVerifyType	校验类型,0 无校验，1 奇校验，2 偶校验
nStopBitLen	停止位（该参数固定为 0，0 代表一个停止位）
int MC_SendEthToUartString(short nUartNum,unsigned char*pSendBuf, short nLength)	
nUartNum	串口号（485 的串口号为 3）
pSendBuf	待发送数据存储指针
nLength	收取到的数据长度，不会大于 256，0 代表未收到任何数据
int MC_ReadUartToEthString(short nUartNum,unsigned char* pRecvBuf, short* pLength)	
nUartNum	串口号（485 的串口号为 3）
pRecvBuf	待收取数据存储指针
pLength	收取到的数据长度，不会大于 256，0 代表未收到任何数据

5.18、坐标系跟随相关 API(仅高端款支持)

API	说明
MC_SetAddAxis	设置指定轴的累加运动轴。

参数详细说明：

int MC_SetAddAxis(short nAxisNum,short nAddAxisNum)	
nAxisNum	轴号，1~AXIS_MAX_COUNT
nAddAxisNum	累加运动轴轴号，1~(AXIS_MAX_COUNT)

坐标系跟随主要用于对运动中产品进行加工。

举例说明：

某产品在拉带上运动，需要对该产品进行加工，但在加工过程中，拉带不能停止。此时就需要用到坐标系跟随功能，需要将拉带的运动补偿到坐标系相关轴当中。

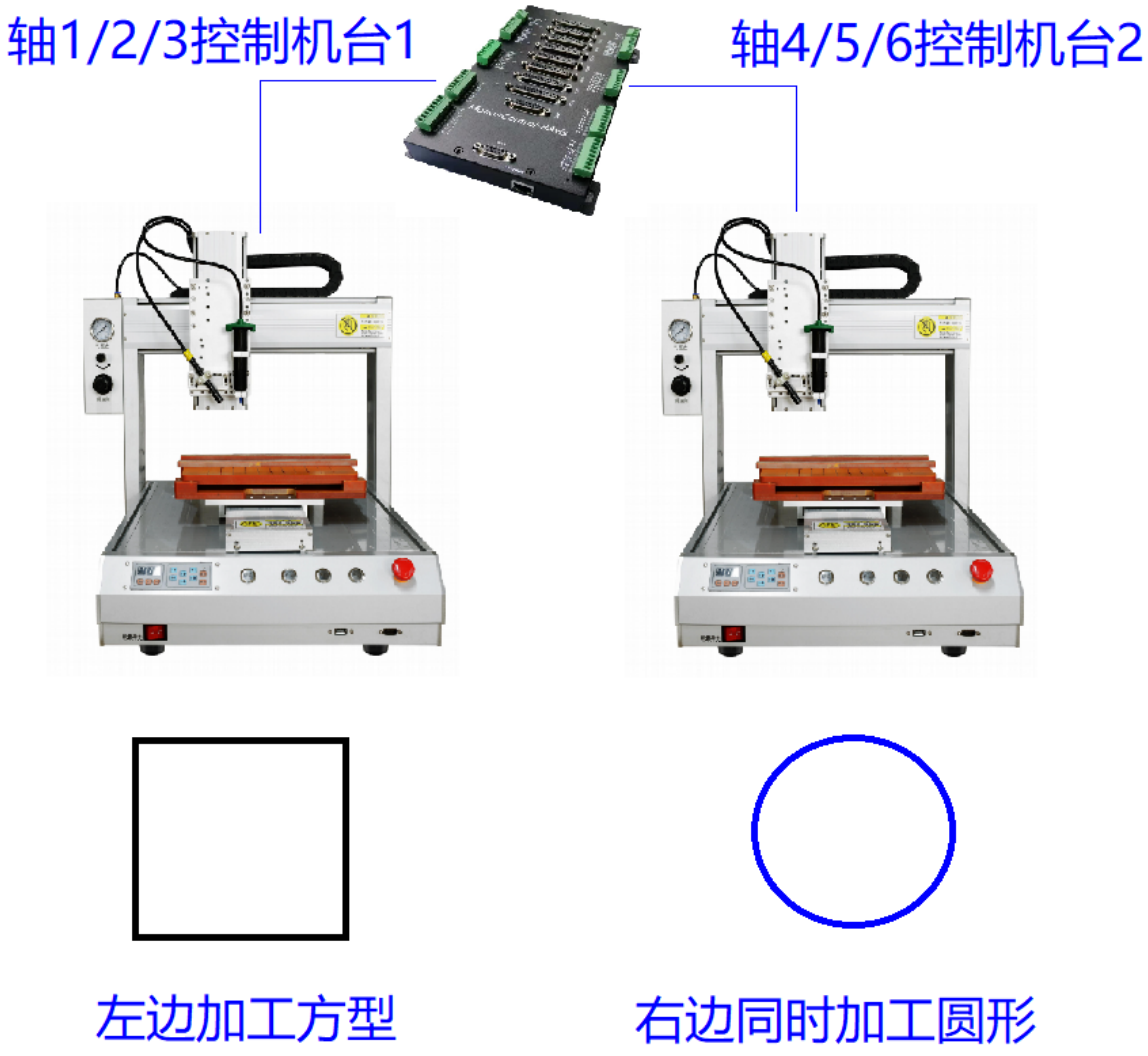
5.19、双通道相关使用说明(仅高端款支持)

标准款控制卡只能建立一个坐标系。但双通道即允许客户建立 2 个坐标系，2 个坐标系相互独立，互不干涉。

带双通道功能的控制卡，可以实现两个坐标系同时加工，比如 1/2/3 轴组成 XYZ 结构，4/5/6 轴组成另外一个 XYZ 结构。1/2/3 轴加工一个正方形的同时，4/5/6 轴可以同时画圆或者其他图形。

双通道函数和单通道函数是一模一样的，只是单通道坐标系号只能为 1，双通道的坐标系号取值范围是 1~2

具体函数参见“5.9、插补运动指令 API”



5.20、寄存器操作类 API（选配功能，PMC 系列支持，用于梯形图交互）

API	说明
MC_SetPLCShortD	以 short 的格式获取 PLC 的 D 寄存器
MC_GetPLCShortD	以 short 的格式获取 PLC 的 D 寄存器
MC_SetPLCLongD	以 long 的格式设置 PLC 的 D 寄存器
MC_GetPLCLongD	以 long 的格式获取 PLC 的 D 寄存器
MC_SetPLCFloatD	以 float 的格式设置 PLC 的 D 寄存器
MC_GetPLCFloatD	以 float 的格式获取 PLC 的 D 寄存器
MC_SetPLCM	设置 PLC 的 M 值
MC_GetPLCM	获取 PLC 的 M 值

参数详细说明：

int MC_SetPLCShortD(long lAdd,short *pData,short nCount)	
lAdd	D 寄存器首地址
pData	要设置的 D 寄存器存放地址
nCount	一次性设置 D 寄存器的个数
int MC_GetPLCShortD(long lAdd,short *pData,short nCount)	
lAdd	D 寄存器首地址
pData	读取到的 D 寄存器存放地址
nCount	一次性读取 D 寄存器的个数
int MC_SetPLCLongD(long lAdd,long *pData,short nCount)	
lAdd	D 寄存器首地址
pData	要设置的 D 寄存器存放地址
nCount	一次性设置 D 寄存器的个数
int MC_GetPLCLongD(long lAdd,long *pData,short nCount)	
lAdd	D 寄存器首地址
pData	读取到的 D 寄存器存放地址
nCount	一次性读取 D 寄存器的个数
int MC_SetPLCFloatD(long lAdd,float *pData,short nCount)	
lAdd	D 寄存器首地址
pData	要设置的 D 寄存器存放地址
nCount	一次性设置 D 寄存器的个数
int MC_GetPLCFloatD(long lAdd,float *pData,short nCount)	
lAdd	D 寄存器首地址
pData	读取到的 D 寄存器存放地址
nCount	一次性读取 D 寄存器的个数
int MC_SetPLCM(long lAdd,char *pData,short nCount)	
lAdd	M 寄存器首地址
pData	要设置的 M 寄存器存放地址
nCount	一次性设置 M 寄存器的个数
int MC_GetPLCM(long lAdd,char *pData,short nCount)	
lAdd	M 寄存器首地址
pData	读取到的 M 寄存器存放地址

nCount	一次性读取 M 寄存器的个数

5.21、机械臂操作类 API（选配功能，PMC 系列支持）

API	说明
MC_RobotSetPrm	设置机械手参数
MC_RobotSetPrmDelta40001	设置 Delta 机械手 40001 类型参数专用函数
MC_RobotSetForward	设置机械手进入关节运动
MC_RobotSetInverse	设置机械手进入坐标系运动

参数详细说明：

```
int MC_RobotSetPrm(unsigned short RobotID,unsigned long ulRobotType,short nJogAxisCount,short *pJogAxisList,short nVirAxisCount,short* pVirAxisList,void *RobotParm)
```

RobotID	机械手 ID，通常为 1
ulRobotType	机械手类型 40001 为标准 Delta 机械手 50008 为 XYZ+AC 双转台结构（也适用于 XYZ+单转台） 50011 为 XYZ+AC 双摆头结构（也适用于 XYZ+单摆头）
nJogAxisCount	关节轴数量
pJogAxisList	关节轴轴号列表
nVirAxisCount	虚拟轴数量
pVirAxisList	虚拟轴轴号列表

RobotParm	机械手参数，类型不同，参数结构体不同
	//Delta 参数
	typedef struct _DeltaParm{
	long lPlusePerCircle[3];//关节电机每圈脉冲数，单位脉冲
	double dRotateAngle[3];//关节平面相对 XZ 平面的旋转角度，单位度
	double dDisFixPlatform[3];//定平台中心点到连接点的长度，单位 mm
	double dLengthArm1[3];//活动关节 1 的臂长，单位 mm
	double dLengthArm2[3];//活动关节 2 的臂长，单位 mm
	double dDisMovPlatform[3];//动平台中心点到连接点的长度，单位 mm
	}DELTA_PARM;
	
	//XYZAC 双摇篮参数
	typedef struct _XYZAC_PRRA{
	double dCX; //C 轴旋转中心 X 坐标，单位毫米
	double dCY; //C 轴旋转中心 Y 坐标，单位毫米
	double dAY; //A 轴旋转中心 Y 坐标，单位毫米
	double dAZ; //A 轴旋转中心 Z 坐标，单位毫米
	long lPlusePerCircle[5]; //各轴每圈脉冲数量
	double dPitch[5]; //各轴螺距，单位毫米
	}XYZAC_PARM;

	//XYZTATC 双摆头参数
	typedef struct _XYZTATC_PRRA{
	double dDX; //C 在零度时，旋转关节中心相对 Z 轴的距离 X,单位毫米
	double dDY; //C 在零度时,旋转关节中心相对 Z 轴的距离 Y,单位毫米
	double dR; //末端旋转半径，单位毫米
	double dOrgAngleC; //回零完成后,C 轴角度,单位度,范围 0~360
	long lPlusePerCircle[5]; //各轴每圈脉冲数量
	double dPitch[5]; //各轴螺距，单位毫米
	}XYZTATC_PARM;

int MC_RobotSetPrmDelta40001(unsigned short RobotID,short nJogAxisCount,short JogAxis1,short JogAxis2,short JogAxis3,short nVirAxisCount,short VirAxisX,short VirAxisY,short VirAxisZ,long lPlusePerCircle,double dRotateAngle1,double dRotateAngle2,double dRotateAngle3,double dDisFixPlatform,double dLengthArm1,double dLengthArm2,double dDisMovPlatform)	
RobotID	RobotID:机械手 ID,通常为 1
nJogAxisCount	关节轴数量，固定为 3
JogAxis1	关节轴 1 轴号，通常为 1
JogAxis2	关节轴 2 轴号，通常为 2
JogAxis3	关节轴 3 轴号，通常为 3
nVirAxisCount	虚拟坐标轴数量，固定为 3
VirAxisX	虚拟坐标轴 X 轴号，通常为 11
VirAxisY	虚拟坐标轴 Y 轴号，通常为 12
VirAxisZ	虚拟坐标轴 Z 轴号，通常为 13
lPlusePerCircle	各轴每圈脉冲数量
dRotateAngle1	关节平面 1 相对 XZ 平面的旋转角度，单位度
dRotateAngle2	关节平面 2 相对 XZ 平面的旋转角度，单位度
dRotateAngle3	关节平面 3 相对 XZ 平面的旋转角度，单位度
dDisFixPlatform	定平台中心点到连接点的长度，单位 mm
dLengthArm1	活动关节 1 的臂长，单位 mm
dLengthArm2	活动关节 2 的臂长，单位 mm
dDisMovPlatform	动平台中心点到连接点的长度，单位 mm
int MC_RobotSetForward(unsigned short nRobotID)	
nRobotID	机械手 ID，通常为 1，调用该函数后，可以进行关节运动
int MC_RobotSetInverse(unsigned short nRobotID)	
nRobotID	机械手 ID，通常为 1，调用该函数后，可以进行坐标系 XYZ 运动

5.22、DXF 图形操作类 API

百度网盘分享的文件：dxf 文件读取解析例程
链接：https://pan.baidu.com/s/1XuTg-Ht584Zz38nI4hNgaw
提取码：6666

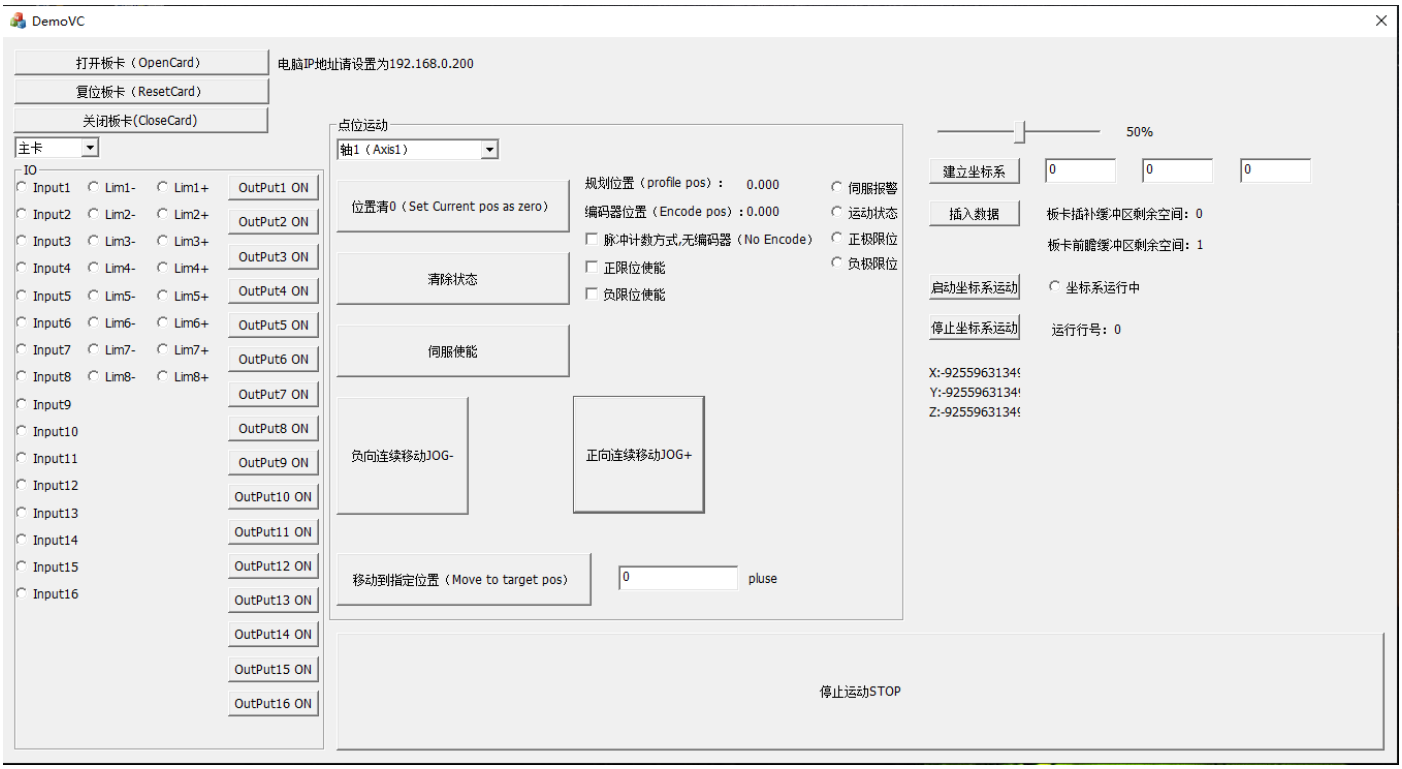
API	说明
MC_DxfLoadFile	加载 dxf 图形
MC_DxfGetCircleCount	获取 dxf 图形中圆形总数量
MC_DxfGetMultiLineCount	获取 dxf 图形中多线段总数量
MC_DxfGetCircleCenterR	获取 dxf 图形中第 N 个圆的圆心 XYZ 坐标和半径 R
MC_DxfGetCircleAllCenterR	获取 dxf 图形中所有圆的圆心坐标和半径
MC_DxfGetMultiLineInfo	获取 dxf 图形中多线段相关信息
MC_DxfGetMultiLinePoint	获取 dxf 图形中第 N 个多线段点数
MC_DxfGetMultiLineAllPoint	获取 dxf 图形中第 N 个多线段所有点的 XYZ 坐标

参数详细说明：

int MC_DxfLoadFile(char* pFile)	
PFile	Dxf 文件路径
int MC_DxfGetCircleCount(short nLayerIndex,long* pCount)	
nLayerIndex	固定为 0
PCount	
int MC_DxfGetMultiLineCount(short nLayerIndex,long* pCount)	
nLayerIndex	固定为 0
PCount	多线段总个数
int MC_DxfGetCircleCenterR(short nLayerIndex,long lCricleIndex,double* dX,double* dY,double* dZ,double* dR)	
nLayerIndex	固定为 0
lCricleIndex	圆弧索引，从 0 开始
dX	圆心 X
dY	圆心 Y
dZ	圆心 Z
dR	圆半径
int MC_DxfGetCircleAllCenterR(short nLayerIndex,double* dX,double* dY,double* dZ,double* dR,long* pCount)	
nLayerIndex	固定为 0
dX	圆心 X
dY	圆心 Y
dZ	圆心 Z
dR	圆半径
pCount	圆总数量
int MC_DxfGetMultiLineInfo(short nLayerIndex,long lMultiLineIndex,double* dSX,double* dSY,double* dSZ,double* dEX,double* dEY,double* dEZ,long* pCount)	
nLayerIndex	固定为 0
lMultiLineIndex	多线段索引，从 0 开始

dSX	起点 X
dSY	起点 Y
dSZ	起点 Z
dEX	终点 X
dEY	终点 Y
dEZ	终点 Z
pCount	总点数
int MC_DxfGetMultiLinePoint(short nLayerIndex,long lMultiLineIndex,long lPointIndex,double* dX,double* dY,double* dZ,short* nArcFlag,short* nArcDir,double* dCenterX,double* dCenterY,double* dR)	
nLayerIndex	固定为 0
lMultiLineIndex	多线段索引，从 0 开始
lPointIndex	点索引，从 0 开始
dX	X 坐标
dY	Y 坐标
dZ	Z 坐标
nArcFlag	0 端点，1 弧点
nArcDir	0 顺时针，1 逆时针（弧点时有效）
dCenterX	圆心坐标 X（弧点时有效）
dCenterY	圆心坐标 Y（弧点时有效）
dR	圆弧半径（弧点时有效）
int MC_DxfGetMultiLineAllPoint(short nLayerIndex,long lMultiLineIndex,double* dX,double* dY,double* dZ,short* nArcFlag,short* nArcDir,double* dCenterX,double* dCenterY,double* dR,long* pCount)	
nLayerIndex	固定为 0
lMultiLineIndex	多线段索引，从 0 开始
dX	X 坐标
dY	Y 坐标
dZ	Z 坐标
nArcFlag	0 端点，1 弧点
nArcDir	0 顺时针，1 逆时针（弧点时有效）
dCenterX	圆心坐标 X（弧点时有效）
dCenterY	圆心坐标 Y（弧点时有效）
dR	圆弧半径（弧点时有效）
pCount	总点数

六、测试软件


DemoVC
打开板卡（OpenCard）
电脑IP地址请设置为192.168.0.200
复位板卡（ResetCard）
关闭板卡(CloseCard)
主卡
IO
Input1 Lim1- Lim1+ OutPut1 ON
Input2 Lim2- Lim2+ OutPut2 ON
Input3 Lim3- Lim3+ OutPut3 ON
Input4 Lim4- Lim4+ OutPut4 ON
Input5 Lim5- Lim5+ OutPut5 ON
Input6 Lim6- Lim6+ OutPut6 ON
Input7 Lim7- Lim7+ OutPut7 ON
Input8 Lim8- Lim8+ OutPut8 ON
Input9 OutPut9 ON
Input10 OutPut10 ON
Input11 OutPut11 ON
Input12 OutPut12 ON
Input13 OutPut13 ON
Input14 OutPut14 ON
Input15 OutPut15 ON
Input16 OutPut16 ON
点位运动
轴1（Axis1）
位置清0（Set Current pos as zero）
清除状态
伺服使能
负向连续移动JOG-
正向连续移动JOG+
规划位置（profile pos）： 0.000
编码器位置（Encode pos）：0.000
脉冲计数方式,无编码器（No Encode）
正限位使能
负限位使能
伺服报警
运动状态
正极限位
负极限位
移动到指定位置（Move to target pos）
0
pluse
停止运动STOP
50%
建立坐标系
插入数据
板卡插补缓冲区剩余空间：0
板卡前瞻缓冲区剩余空间：1
启动坐标系运动
坐标系运行中
停止坐标系运动
运行行号：0
X:-92559631349
Y:-92559631349
Z:-92559631349

七、PC 端 IP 配置及多轴板卡并联实现方法

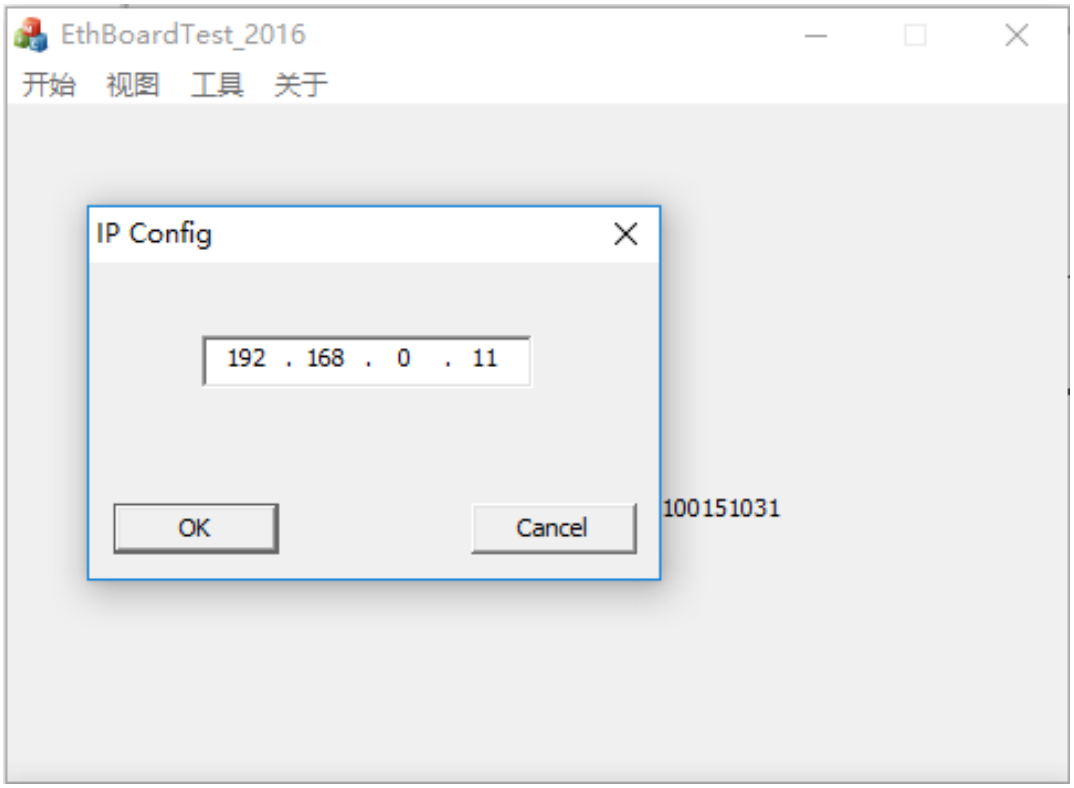
通过多个板卡并联的方案，一台电脑可以控制 16 轴、32 轴、48 轴....1024 轴

电脑 IP 地址需设置为 192.168.0.200

板卡出厂默认 IP 地址为 192.168.0.1，通常情况下无需做修改。

打开板卡代码为：MC_Open(0,"192.168.0.200");

用户如果需要多个轴板卡同时使用，可以用串口方式打开板卡，修改 IP 地址。如下图所示：



例如同时使用 3 个轴板卡，IP 地址分别为 192.168.0.1、192.168.0.2、192.168.0.3，则打开板卡代码为：

```
int iRes = 0;
iRes += MC_SetCardNo(1);
iRes += MC_Open(0,"192.168.0.200");
iRes += MC_SetCardNo(2);
iRes += MC_Open(0,"192.168.0.200");
iRes += MC_SetCardNo(3);
iRes += MC_Open(0,"192.168.0.200");
```

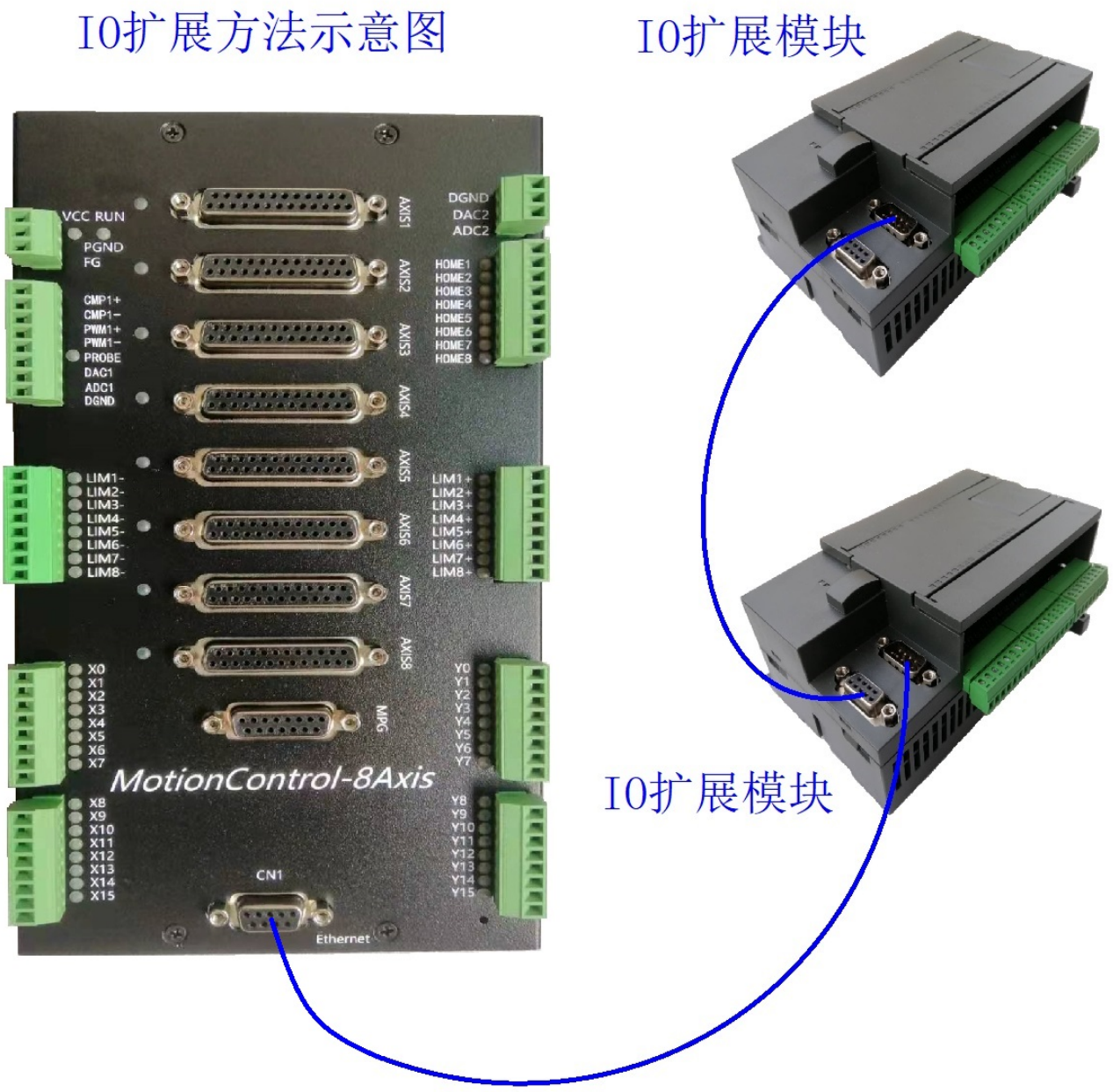
设置板卡 1 第 1 个 IO 输出，然后板卡 2 第 1 个 IO 输出，最后板卡 3 第 1 个 IO 输出代码为：

```
iRes += MC_SetCardNo(1);
iRes += MC_SetExtDoBit(0,0,1);
iRes += MC_SetCardNo(2);
iRes += MC_SetExtDoBit(0,0,1);
iRes += MC_SetCardNo(3);
iRes += MC_SetExtDoBit(0,0,1);
```

重点说明：如果使用交换机，通常建议板卡 IP 从 192.168.0.2 开始。

八、IO 扩展方法

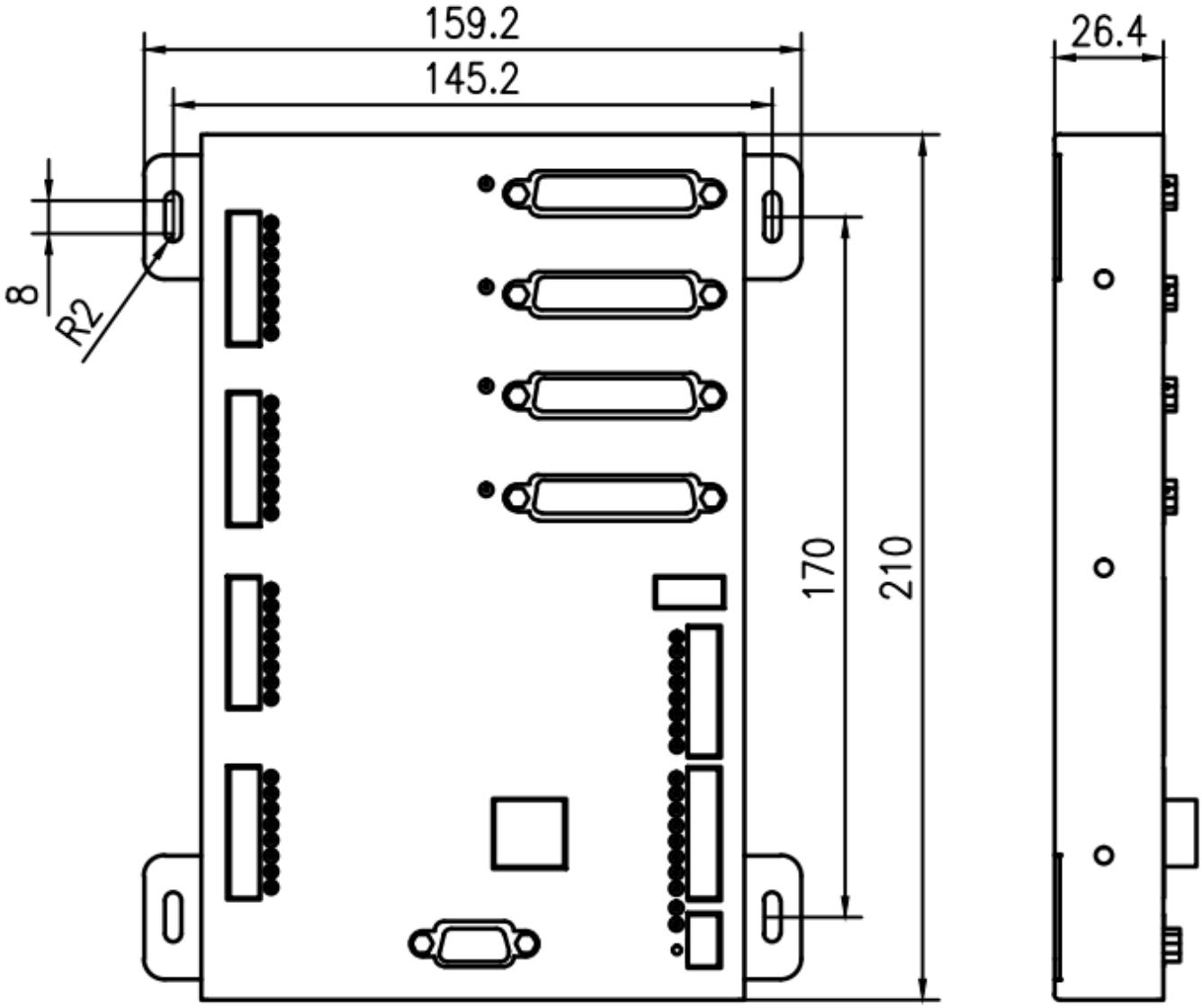
如果运动控制卡自身 IO 不够用，可选购我司 IO 扩展卡配套使用，通过一根 DB9 延长线串联即可，扩展协议为我司自定义总线协议，性能稳定，延迟小，且不占用电脑端口。



九、运动控制卡安装尺寸

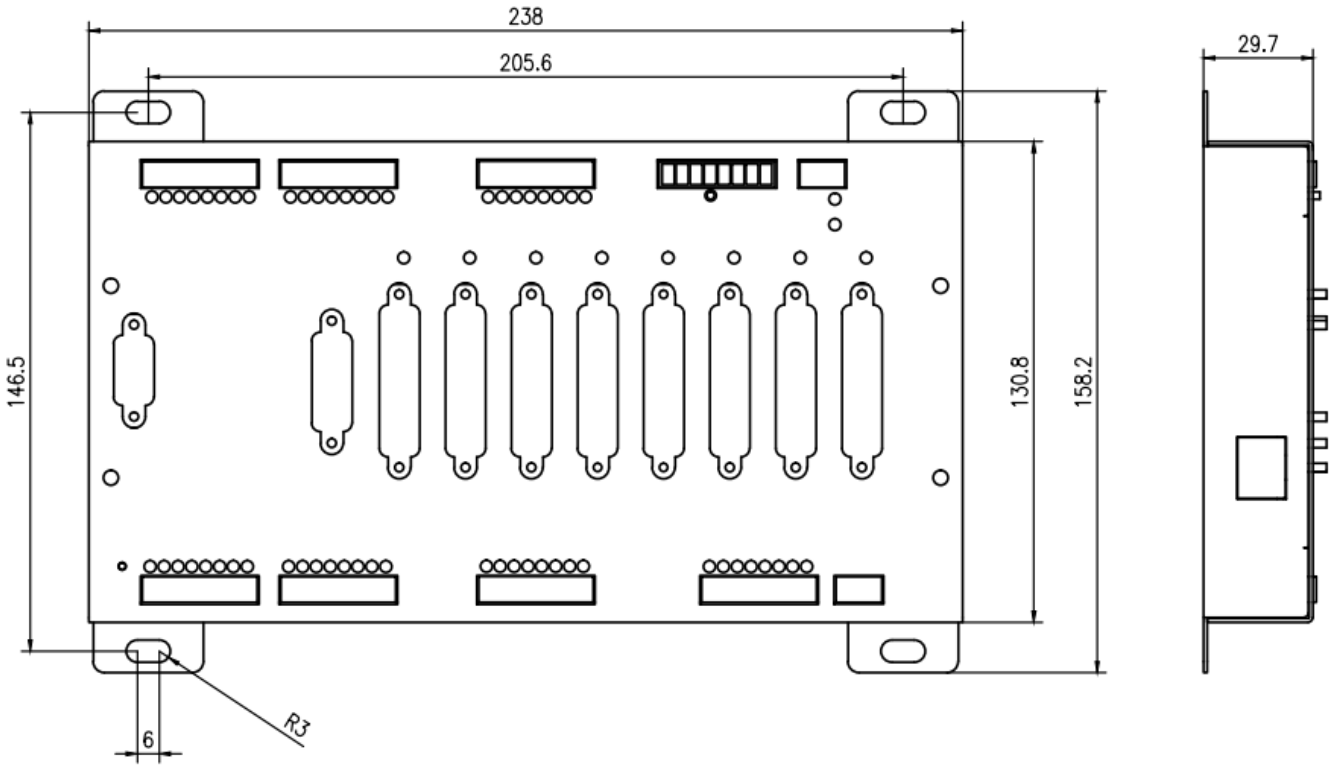
1、四轴运动控制卡安装尺寸

2~4 轴钣金 3D 模型下载链接：https://pan.baidu.com/s/1y5RfYDzl-zvReiIfnhOI3Q
提取码：6666



2、6 轴、8 轴运动控制卡安装尺寸

6~8 轴钣金 3D 模型下载链接：https://pan.baidu.com/s/137gJo0f9v3l8JkP1S6fYFg
提取码：6666



3、10 轴~16 轴运动控制卡安装尺寸

10~16 轴钣金 3D 模型下载链接：https://pan.baidu.com/s/1bEq2PB4o7e_i1rqpKZCfkw
提取码：6666

十、附录 API 一览

板卡打开关闭 API	
MC_SetCardNo	切换当前运动控制器卡号
MC_GetCardNo	读取当前运动控制器卡号
MC_Open	打开板卡
MC_Reset	复位板卡
MC_Close	关闭板卡
板卡配置类 API	
MC_AlarmOn	设置轴驱动报警信号有效
MC_AlarmOff	设置轴驱动报警信号无效
MC_AlarmSns	设置运动控制器轴报警信号电平逻辑
MC_LmtsOn	设置轴限位信号有效
MC_LmtsOff	设置轴限位信号无效
MC_LmtSns	设置运动控制器各轴限位触发电平
MC_EncOn	设置为“外部编码器”计数方式
MC_EncOff	设置为“脉冲计数器”计数方式
MC_EncSns	设置编码器的计数方向
MC_StepSns	设置脉冲输出通道的方向
IO 操作 API	
MC_GetDiRaw	读取数字 IO 输入状态的原始值
MC_GetDiReverseCount	读取数字量输入信号的变化次数
MC_SetDiReverseCount	设置数字量输入信号的变化次数的初值
MC_SetExtDoValue	设置 IO 输出（包含主模块和扩展模块）
MC_GetExtDiValue	获取 IO 输入（包含主模块和扩展模块）
MC_GetExtDoValue	获取 IO 输出（包含主模块和扩展模块）
MC_SetExtDoBit	设置指定 IO 模块的指定位输出（包含主模块和扩展模块）
MC_GetExtDiBit	获取指定 IO 模块的指定位输入（包含主模块和扩展模块）
MC_GetExtDoBit	获取指定 IO 模块的指定位输出（包含主模块和扩展模块）
MC_SetDac	设置 DAC 输出电压
MC_GetAdc	读取 ADC 输入电压
MC_SetAdcFilter	设置模拟量输入滤波时间
MC_SetAdcBias	设置模拟量输入通道的零漂电压补偿值
MC_GetAdcBias	读取模拟量输入通道的零漂电压补偿值
MC_SetPwm	设置 PWM 输出频率以及占空比
MC_SetDoBitReverse	设置数字 IO 输出指定时间的单个脉冲
轴点位运动 API	
MC_PrfTrap	设置指定轴为点位模式
MC_SetTrapPrm	设置点位模式运动参数
MC_SetTrapPrmSingle	设置点位模式运动参数（可替代 MC_SetTrapPrm）
MC_GetTrapPrm	读取点位模式运动参数
MC_GetTrapPrmSingle	读取点位模式运动参数（可替代 MC_GetTrapPrm）

MC_SetPos	设置目标位置
MC_SetVel	设置目标速度
MC_Update	启动点位运动
轴 JOG 运动 API	
MC_PrfJog	设置指定轴为 JOG 模式(速度模式)
MC_SetJogPrm	设置 JOG 模式运动参数
MC_SetJogPrmSingle	设置 JOG 模式运动参数（可替代 MC_SetJogPrm）
MC_GetJogPrm	读取 JOG 模式运动参数
MC_GetJogPrmSingle	读取 JOG 模式运动参数（可替代 MC_GetJogPrm）
MC_SetVel	设置目标速度
MC_Update	启动 JOG 运动
运动状态检测类 API	
MC_AxisOn	打开驱动器使能
MC_AxisOff	关闭驱动器使能
MC_Stop	停止一个或多个轴的规划运动，停止坐标系运动
MC_GetSts	读取轴状态
MC_ClrSts	清除驱动器报警标志、跟随误差越限标志、限位触发标志
MC_GetPrfPos	读取规划位置
MC_GetAxisEncPos	读取编码器位值
MC_GetPrfVel	读取规划速度
MC_GetAllSysStatusSX	获取所有板卡相关状态
安全机制 API	
MC_SetSoftLimit	设置软限位
MC_GetSoftLimit	获取软限位
MC_LmtsOn	设置轴限位信号有效
MC_LmtsOff	设置轴限位信号无效
MC_LmtSns	设置运动控制器各轴限位触发电平
MC_EStopSetIO	设置系统紧急停止 IO
MC_EStopOnOff	开启/关闭紧急停止功能
MC_EStopGetSts	获取紧急停止触发状态
MC_EStopClrSts	清除紧急停止触发状态
MC_SetHardLimP	设置正硬限位映射 IO（10 轴以上用）
MC_SetHardLimN	设置负硬限位映射 IO（10 轴以上用）
其他 API	
MC_ZeroPos	清零轴的规划和编码器位置
MC_GetID	获取板卡唯一标识 ID
MC_SetAxisBand	设置轴到位误差带
MC_SetBacklash	设置反向间隙补偿的相关参数
MC_GetBacklash	读取反向间隙补偿的相关参数
MC_SetStopDec	设置平滑停止减速度和急停减速度
MC_WriteInterFlash	写板卡内部 Flash
MC_ReadInterFlash	读板卡内部 Flash
MC_DownPitchErrorTable	下载螺距误差补偿表
MC_ReadPitchErrorTable	读取螺距误差补偿表

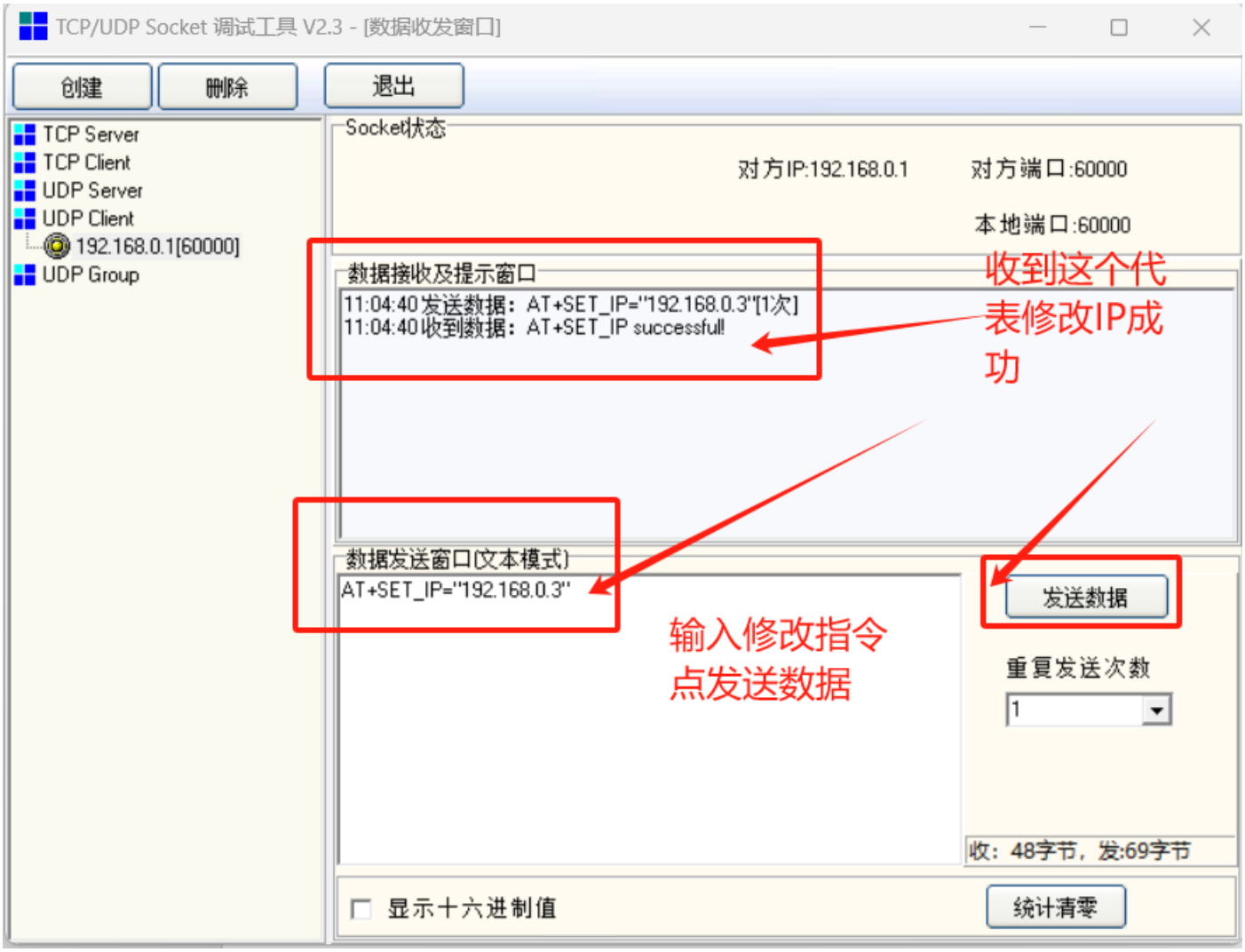
MC_AxisErrPitchOn	打开指定螺距误差补偿
MC_AxisErrPitchOff	关闭指定螺距误差补偿
插补运动指令 API	
MC_SetCrdPrm	设置坐标系参数，确立坐标系映射，建立坐标系
MC_GetCrdPrm	查询坐标系参数
MC_InitLookAhead	配置指定坐标系指定 FifoIndex 前瞻缓冲区的拐弯速率，最大加速度，缓冲区深度，缓冲区指针等参数
MC_InitLookAheadSingle	用于替代 MC_InitLookAhead 函数，方便不擅长结构体的客户使用。
MC_CrdClear	清除插补缓存区内的插补数据
MC_LnXY	缓存区指令，两维直线插补
MC_LnXYZ	缓存区指令，三维直线插补
MC_LnXYZA	缓存区指令，四维直线插补
MC_LnXYZAB	缓存区指令，五维直线插补
MC_LnXYZABC	缓存区指令，六维直线插补
MC_LnAll	缓存区指令，7~14 维直线插补
MC_ArcXYC	缓存区指令，XY 平面圆弧插补（以终点坐标和圆心位置为输入参数）
MC_ArcXZC	缓存区指令，XZ 平面圆弧插补（以终点坐标和圆心位置为输入参数）
MC_ArcYZC	缓存区指令，YZ 平面圆弧插补（以终点坐标和圆心位置为输入参数）
MC_HelixXYCZ	缓存区指令，XY 平面螺旋线插补（以终点坐标和圆心位置为输入参数）
MC_HelixXZCY	缓存区指令，XZ 平面螺旋线插补（以终点坐标和圆心位置为输入参数）
MC_HelixYZCX	缓存区指令，YZ 平面螺旋线插补（以终点坐标和圆心位置为输入参数）
MC_HelixXYCCount	缓存区指令，XY 平面螺旋线插补（以终点坐标和圆心位置为输入参数）
MC_HelixXZCCount	缓存区指令，XZ 平面螺旋线插补（以终点坐标和圆心位置为输入参数）
MC_HelixYZCCount	缓存区指令，YZ 平面螺旋线插补（以终点坐标和圆心位置为输入参数）
MC_BufPWM	缓存区指令，设置 PWM 频率及占空比
MC_BufIO	缓存区指令，设置 IO 输出
MC_BufIOReverse	缓存区指令，设置 IO 输出一个指定时间的脉冲
MC_BufDelay	缓存区指令，延时一段时间
MC_BufMoveVel	在插补运动的过程中插入 BufferMove 轴的速度设定
MC_BufMoveAcc	在插补运动的过程中插入 BufferMove 轴的加速度设定
MC_BufMove	在插补运动的过程中插入阻塞和非阻塞的点位运动
MC_BufGear	设定了脉冲输出的个数。它会保证与其后紧挨的指令同时启动，同时停止
MC_CrdData	向插补缓存区增加插补数据
MC_CrdStart	启动插补运动
MC_SetOverride	设置插补运动目标合成速度倍率
MC_GetCrdPos	查询该坐标系的当前坐标位置值
MC_CrdSpace	读取插补缓存区中的剩余空间
MC_CrdStatus	查询插补运动坐标系状态
MC_SetUserSegNum	缓存区指令，设置自定义插补段段号
MC_GetUserSegNum	读取自定义插补段段号
MC_GetRemainderSegNum	读取未完成的插补段段数
MC_GetLookAheadSpace	获取前瞻缓冲区剩余空间
MC_GetLookAheadSegCount	获取前瞻缓存区剩余段数
MC_GetCrdVel	查询该坐标系的当前合成速度值
硬件捕获类 API	

MC_SetCaptureMode	设置编码器捕获方式，并启动捕获
MC_GetCaptureMode	读取编码器捕获方式
MC_GetCaptureStatus	读取编码器捕获状态
MC_SetCaptureSense	设置捕获电平
MC_GetCaptureSense	获取捕获电平
MC_ClearCaptureStatus	清除捕获状态
Gear/电子齿轮/电子凸轮类 API	
MC_PrfGear	设置指定轴进入电子齿轮模式
MC_SetGearMaster	设置电子齿轮运动跟随主轴
MC_GetGearMaster	读取电子齿轮运动跟随主轴
MC_SetGearRatio	设置电子齿轮比
MC_GetGearRatio	获取电子齿轮比
MC_GearStart	启动电子齿轮运动
MC_GearStop	停止电子齿轮运动
MC_SetGearEvent	设置电子齿轮触发事件
MC_GetGearEvent	获取电子齿轮触发事件
比较输出类 API	
MC_CmpPluse	设置比较器输出 IO 立即输出指定电平或者脉冲
MC_CmpBufSetChannel	设置比较缓冲区对应输出通道
MC_CmpBufData	向比较器缓冲区发送比较数据
MC_CmpBufSts	获取比较器缓冲区状态
MC_CmpBufStop	停止比较器缓冲区
MC_CmpRpt	设置比较器缓冲区等比输出
MC_CmpSetTriggerCount	设置比较器缓冲区触发计数初值
MC_CmpGetTriggerCount	获取比较器缓冲区触发计数初值
PT 模式 API	
MC_PrfPt	设置指定轴为 PT 模式
MC_PtSpace	读取指定轴的 PT 缓冲区空闲存储空间
MC_PtRemain	读取指定轴的 PT 存储空间尚未执行的数据长度
MC_PtData	向指定轴的 PT 缓冲区发送数据
MC_PtClear	清除指定轴 PT 缓冲区的数据
MC_PtStart	启动 PT 模式运动
手轮相关 API（支持手轮接口型号可用）	
MC_StartHandwheel	设置指定轴进入手轮模式
MC_EndHandwheel	设置指定轴退出手轮模式

十一、常见问题解答

11.1、如何修改 IP 地址？

链接：https://pan.baidu.com/s/1f1gupFRGjVr8E4dL0PrsZg
提取码：pr6c
下载上面修改 IP 地址的小工具即可修改 IP 地址。



11.2、IP 地址忘记了怎么办？

上电状态下，长按面板上的复位按键就可以恢复出厂 IP，出厂 IP 地址为 192.168.0.1。

11.3、急停信号接哪里？

通用输入都可以映射为轴的急停输入，设置函数为 MC_EStopSetIO
关于这个函数的使用细节，参见章节 5.7“安全机制 API”

11.4、为什么碰到硬限位轴运动也不停止？

通常都是因为没有使能硬限位，使能函数为 MC_LmtsOn 这个函数，具体参见章节 5.7“安全机制 API”

11.5、调用 MC_Stop 函数停止加速度不够快，怎么调整？

MC_SetStopDec 函数可以修改缓停和急停的加速度，具体参见章节 5.7”其他指令 API”

11.6、点位运动如何判断电机到位？

判断电机到位，主要用 MC_GetSts 函数和 MC_GetPrfPos 函数,前者用于读取轴状态，后者用于读取轴当前脉冲位置。当轴状态里面的 AXIS_STATUS_RUNNING 位变为 0，同时规划脉冲位置与目标位置差值小于 1 时，认为电机到位。

11.7、日志开启方法

在 MC_Open 函数前面，调用 MC_StartDebugLog(1)，可以开启日志
程序运行后，日志在 RunTimeLog 文件夹