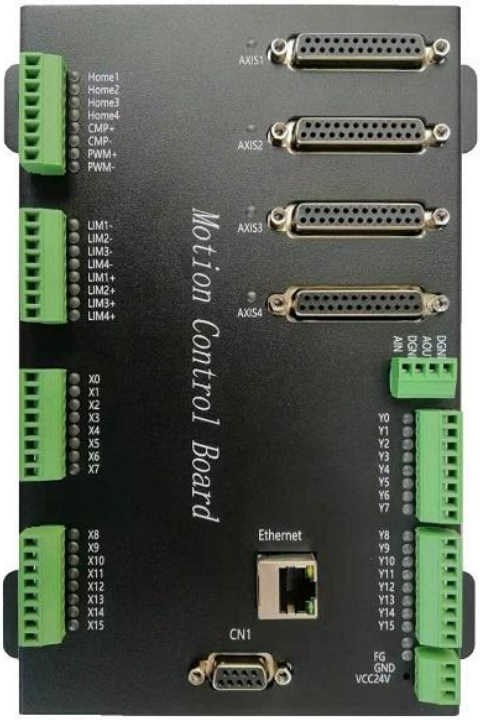
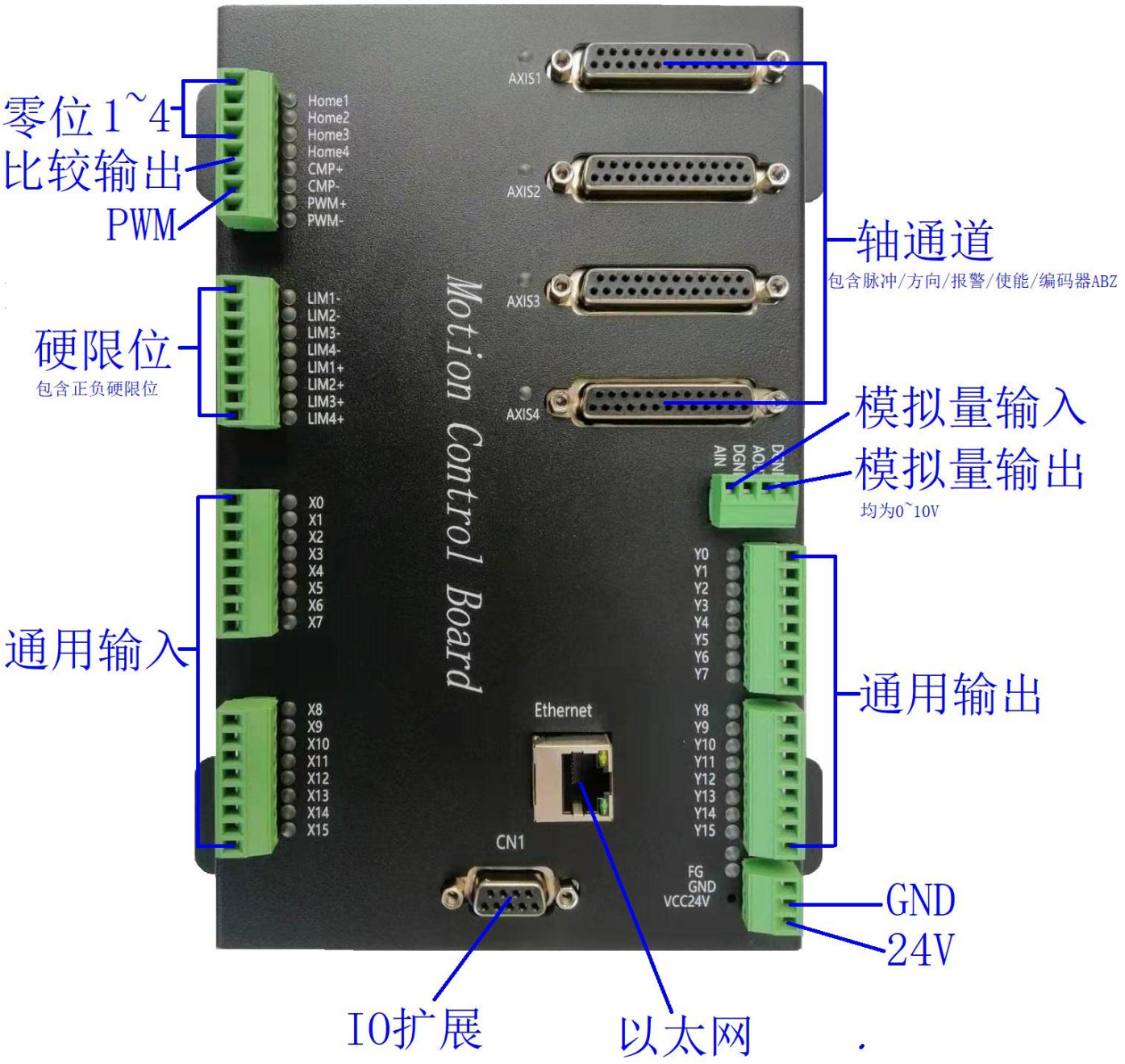
ETH_GAS_N 运动控制卡用户手册 V3.9

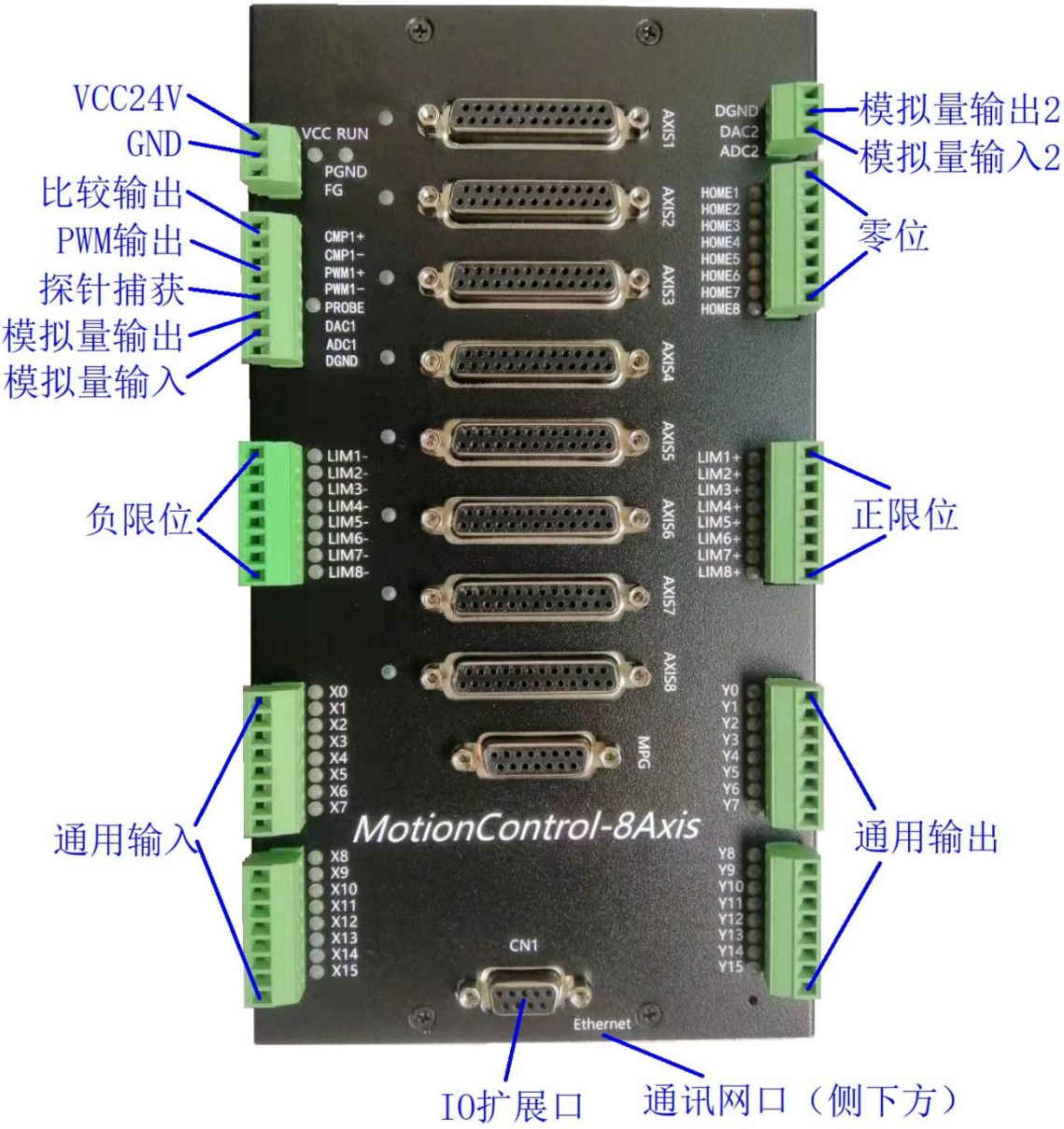


目录

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。

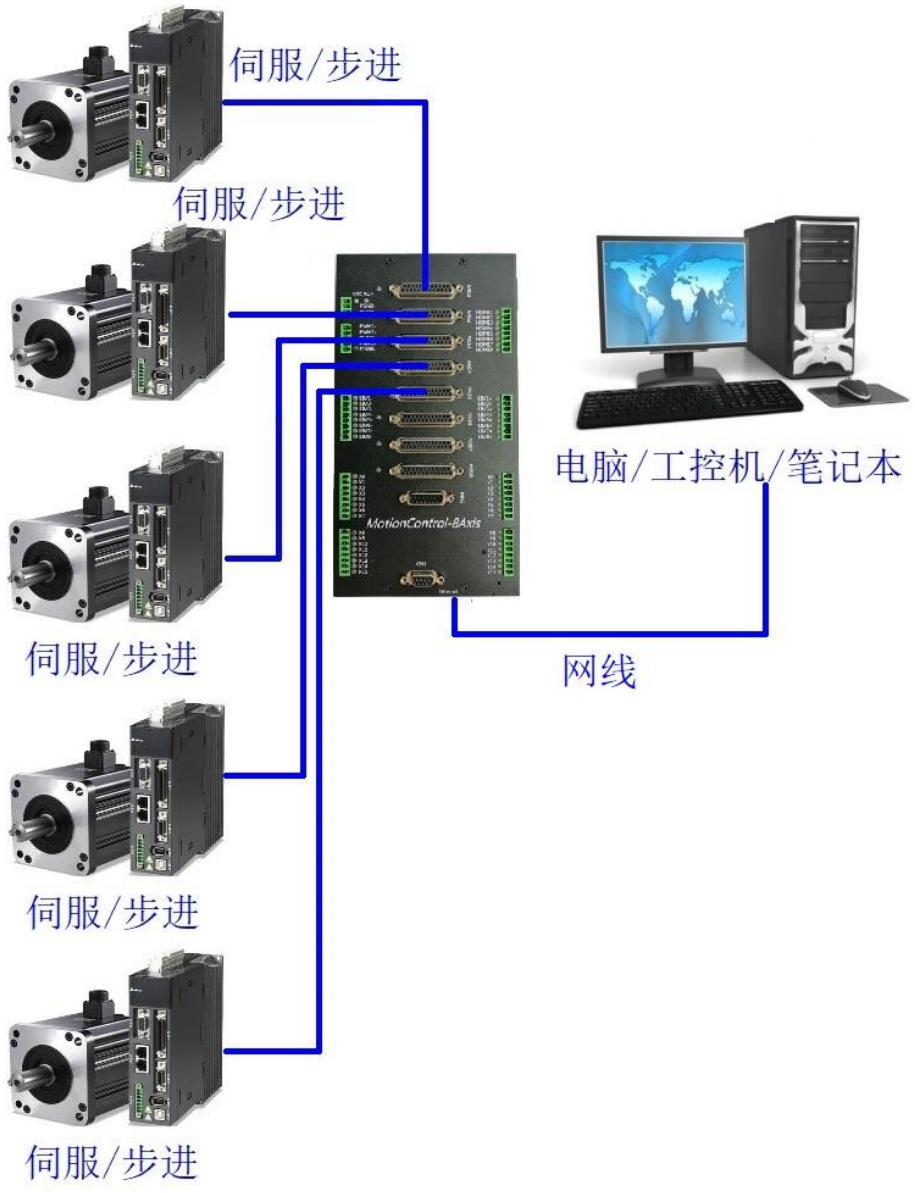

零位1~4
比较输出
PWM
硬限位
包含正负硬限位
通用输入
轴通道
包含脉冲/方向/报警/使能/编码器ABZ
模拟量输入
模拟量输出
均为0~10V
通用输出
GND
24V
IO扩展
以太网
Home1
Home2
Home3
Home4
CMP+
CMP-
PWM+
PWM-
LIM1-
LIM2-
LIM3-
LIM4-
LIM1+
LIM2+
LIM3+
LIM4+
X0
X1
X2
X3
X4
X5
X6
X7
X8
X9
X10
X11
X12
X13
X14
X15
AXIS1
AXIS2
AXIS3
AXIS4
AIN
DGND
AOUT
DIN
Y0
Y1
Y2
Y3
Y4
Y5
Y6
Y7
Y8
Y9
Y10
Y11
Y12
Y13
Y14
Y15
FG
GND
VCC24V
Ethernet
CN1
Motion Control Board


VCC24V
GND
比较输出
PWM输出
探针捕获
模拟量输出
模拟量输入
负限位
通用输入
IO扩展口
通讯网口（侧下方）
模拟量输出2
模拟量输入2
零位
正限位
通用输出
VCC RUN
PGND
FG
CMP1+
CMP1-
PWM1+
PWM1-
PROBE
DAC1
ADC1
DGND
AXIS1
AXIS2
AXIS3
AXIS4
AXIS5
AXIS6
AXIS7
AXIS8
MPG
DGND
DAC2
ADC2
HOME1
HOME2
HOME3
HOME4
HOME5
HOME6
HOME7
HOME8
LIM1-
LIM2-
LIM3-
LIM4-
LIM5-
LIM6-
LIM7-
LIM8-
LIM1+
LIM2+
LIM3+
LIM4+
LIM5+
LIM6+
LIM7+
LIM8+
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
MotionControl-8Axis
CN1
Ethernet

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、轴信号接口

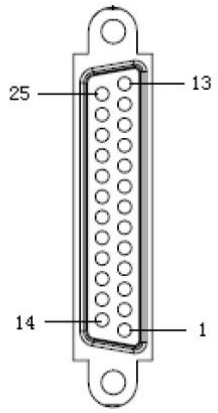
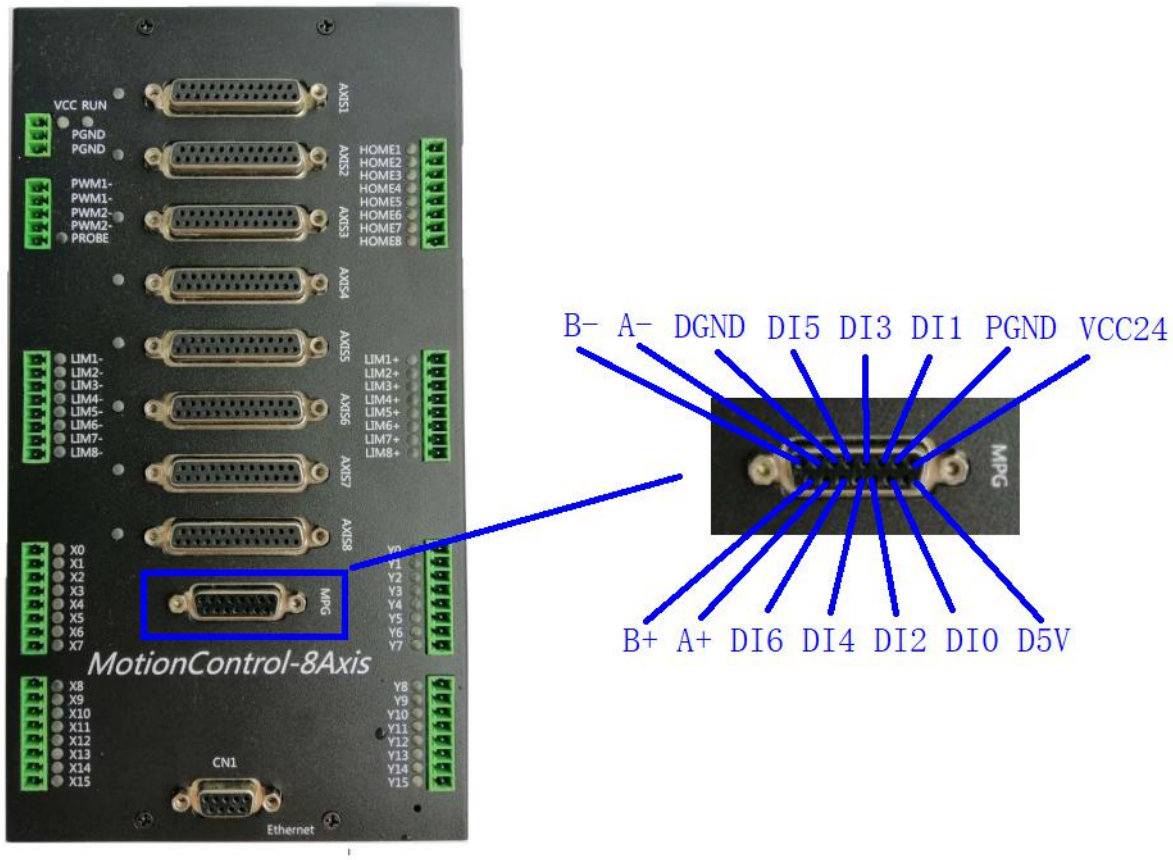


图 3-1：轴通道接口引脚说明

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+
10	GND	数字地
11	PLUSE-	脉冲输出-
12	NC	保留
13	GND	数字地
14	OVCC	+24V 输出
15	RESET	驱动报警复位
16	SERDY	电机到位
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-
23	PLUSE+	脉冲输出+
24	GND	数字地
25	NC	保留

3.2、手轮通道接口



控制卡	手轮
D5V	VCC 和 LED+
PGND	0V 和 LED-
A+	A
A-	A-
B+	B
B-	B-
DI0	X
DI1	Y
DI2	Z
DI3	A
DI4	B
DI5	X1
DI6	X10
DGND	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！！！

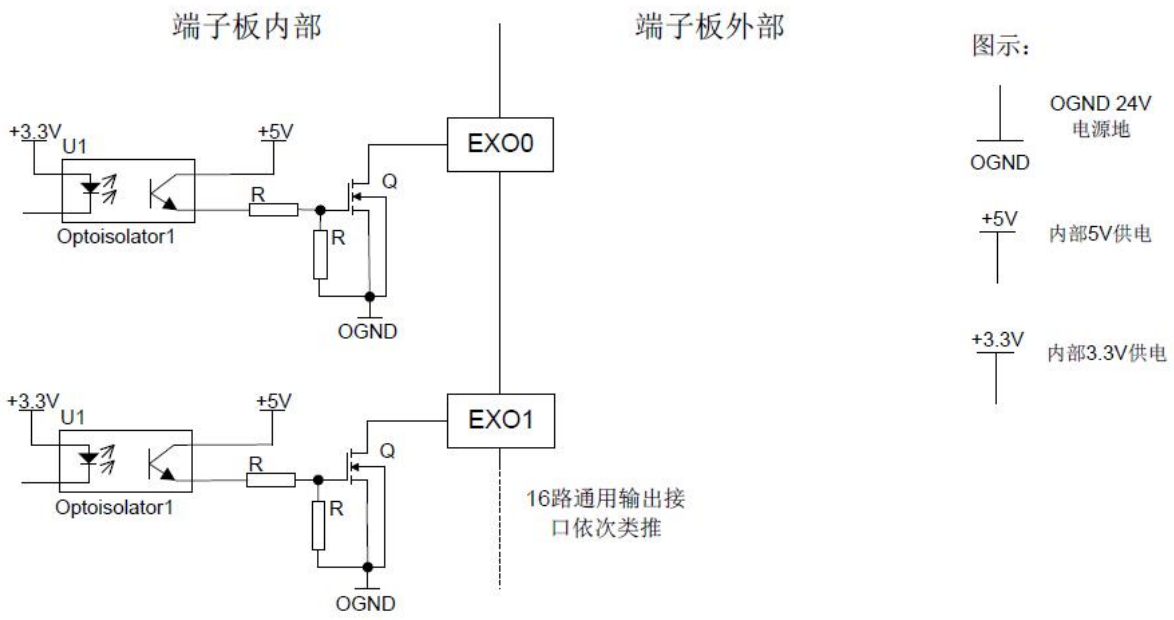
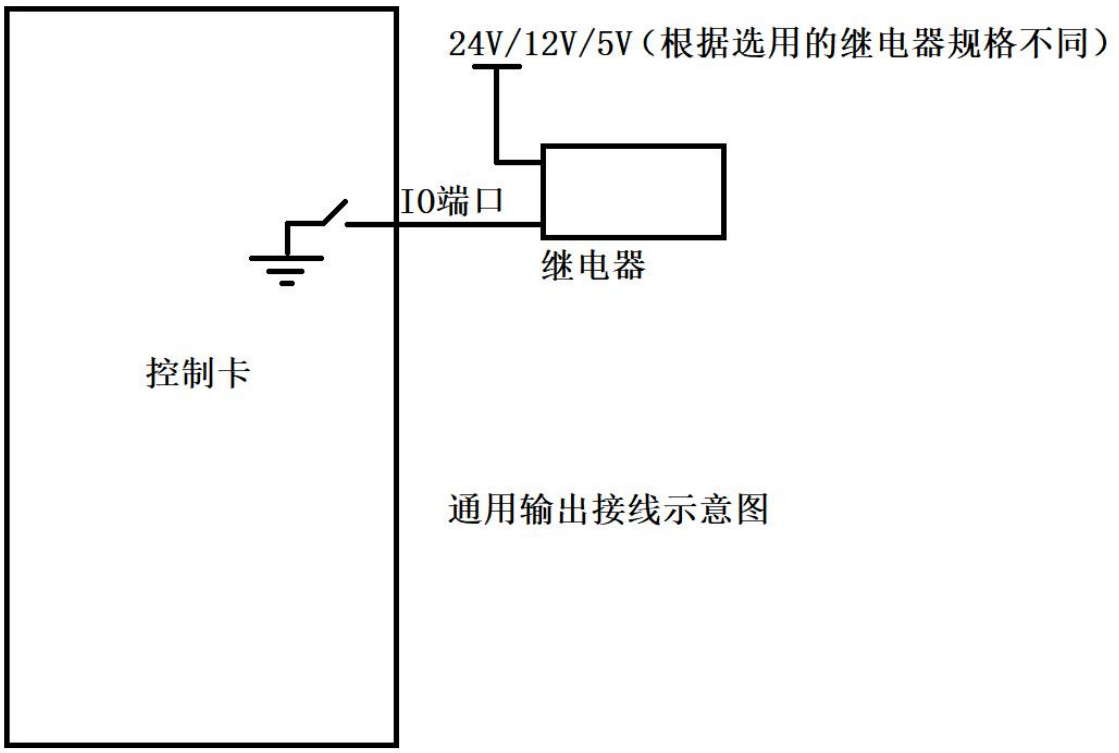


图 3-3：端子板通用数字输出信号内部电路示意图



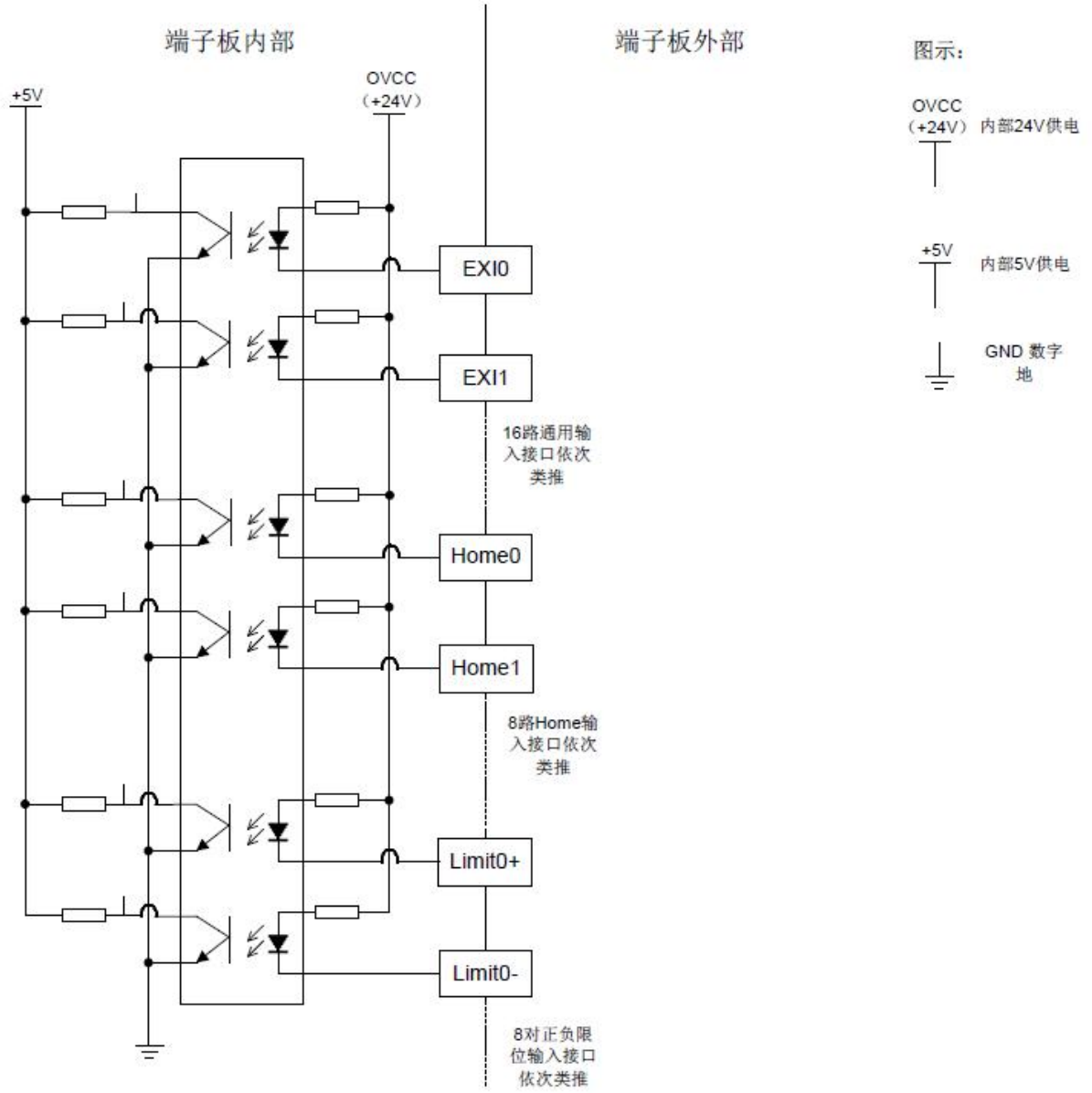
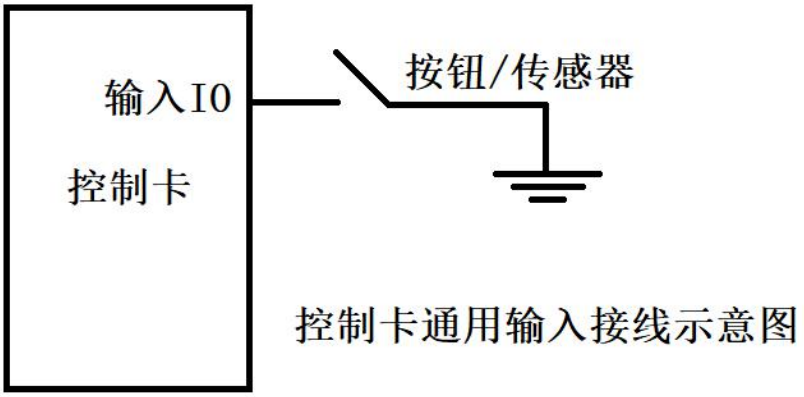


图 3-2：端子板通用输入，HOME 输入，LIMIT 输入信号内部电路示意图



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 GA_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
GA_SetCardNo	切换当前运动控制器卡号
GA_GetCardNo	读取当前运动控制器卡号
GA_Open	打开板卡
GA_Reset	复位板卡
GA_Close	关闭板卡

参数详细说明：

int GA_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 GA_GetCardNo(short *pCardNum)	
pCardNum	读取的当前运动控制器的卡号
int GA_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 GA_Reset()	
无参数	
int GA_Close()	
无参数	

示例代码：

```
int iRes = 0;
iRes += GA_SetCardNo(1);//切换到第 1 块板卡
iRes += GA_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 += GA_Reset();//复位板卡
iRes += GA_Close();//关闭板卡
```

5.2、板卡配置类 API

API	说明
GA_AlarmOn	设置轴驱动报警信号有效
GA_AlarmOff	设置轴驱动报警信号无效
GA_AlarmSns	设置运动控制器轴报警信号电平逻辑
GA_LmtsOn	设置轴限位信号有效
GA_LmtsOff	设置轴限位信号无效
GA_LmtSns	设置运动控制器各轴限位触发电平
GA_EncOn	设置为“外部编码器”计数方式
GA_EncOff	设置为“脉冲计数器”计数方式

GA_EncSns	设置编码器的计数方向
GA_StepSns	设置脉冲输出通道的方向
GA_HomeSns	设置运动控制器 HOME 输入的电平逻辑

参数详细说明：

int GA_AlarmOn(short nAxisNum)	
nAxisNum	控制轴号，取值范围：[1,AXIS_MAX_COUNT]
int GA_AlarmOff(short nAxisNum)	
nAxisNum	控制轴号，取值范围：[1,AXIS_MAX_COUNT]
int GA_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 GA_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)：需要将该轴的正限位有效 MC_LIMIT_NEGATIVE(该宏定义为)：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 GA_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)：需要将该轴的正限位无效 MC_LIMIT_NEGATIVE(该宏定义为)：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 GA_LmtSns(unsigned short nSense)	
nSense	按位标识轴的限位的触发电平状态，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，指令GA_LmtSns()的参数设置各轴正负限位开关的触发电平，当该参数的某个状态位时，表示将对应的限位开关设置为高电平触发，当某个状态位为时，表示将对应的限开关设置为低电平触发。 指令参数和各轴限位的对应关系如下所示： 状态位 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 限位开关 轴 8 轴 7 轴 6 轴 5 轴 4 轴 3 轴 2 轴 1
int GA_EncOn(short nEncoderNum)	
nEncoderNum	编码器通道号
int GA_EncOff(short nEncoderNum)	
nEncoderNum	编码器通道号
int GA_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 GA_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 GA_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 += GA_SetCardNo(1);//切换到第 1 块板卡
iRes += GA_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 += GA_Reset();//复位板卡
iRes += GA_AlarmOn(1);//设置轴 1 驱动报警信号有效
iRes += GA_AlarmOff(1);//设置轴 1 驱动报警信号无效
iRes += GA_Close();//关闭板卡
```

5.3、IO 操作 API

API	说明
GA_GetDiRaw	获取 IO 输入（包含主卡 IO、限位、零位）
GA_GetDiReverseCount	读取数字量输入信号的变化次数
GA_SetDiReverseCount	设置数字量输入信号的变化次数的初值
GA_SetExtDoValue	设置 IO 输出（包含主模块和扩展模块）
GA_GetExtDiValue	获取 IO 输入（包含主模块和扩展模块）
GA_GetExtDoValue	获取 IO 输出（包含主模块和扩展模块）
GA_SetExtDoBit	设置指定 IO 模块的指定位输出（包含主模块和扩展模块）
GA_GetExtDiBit	获取指定 IO 模块的指定位输入（包含主模块和扩展模块）
GA_GetExtDoBit	获取指定 IO 模块的指定位输出（包含主模块和扩展模块）

参数详细说明：

int GA_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 GA_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 GA_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 GA_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 GA_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 GA_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 GA_SetExtDoBit(short nCardIndex,short nBitIndex,unsigned short nValue)	
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)
nValue	IO 输出值(0/1)
int GA_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 GA_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

示例代码:

```
int iRes = 0;
iRes += GA_SetCardNo(1);//切换到第 1 块板卡
iRes += GA_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 += GA_Reset();//复位板卡
iRes += GA_SetExtDoBit(0,4,1);//设置板卡 1 端口 3IO 输出
iRes += GA_SetExtDoBit(0,4,0);//关闭板卡 1 端口 3IO 输出
```

```
iRes += GA_Close();//关闭板卡
```

5.4、点位运动 API

API	说明
GA_PrfTrap	设置指定轴为点位模式
GA_SetTrapPrm	设置点位模式运动参数
GA_SetTrapPrmSingle	设置点位模式运动参数（可替代 GA_SetTrapPrm）
GA_GetTrapPrm	读取点位模式运动参数
GA_GetTrapPrmSingle	读取点位模式运动参数（可替代 GA_GetTrapPrm）
GA_SetPos	设置目标位置
GA_SetVel	设置目标速度
GA_Update	启动点位运动

参数详细说明：

int GA_PrfTrap(short nAxisNum)	
iAxis	规划轴号
int GA_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 GA_SetTrapPrmSingle 函数替代本函数

int GA_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 GA_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 GA_GetTrapPrmSingle 函数替代本函数

int GA_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 GA_SetPos(short nAxisNum,long pos)	
iAxis	规划轴号
pos	设置目标位置，单位是脉冲
int GA_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 GA_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，.....

示例代码：

```
int iRes = 0;

iRes += GA_SetCardNo(1);//切换到第 1 块板卡
iRes += GA_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 += GA_Reset();//复位板卡
iRes += GA_AxisOn(1);//设置轴 1 使能
iRes += GA_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 += GA_SetTrapPrm(m_iAxisNum,&TrapPrm);
iRes += GA_SetVel(1,10);//设置轴 1 点位运动速度为 10 脉冲/毫秒
iRes += GA_SetPos(1,10000);//设置轴 1 目标位置为 10000
iRes += GA_Update(0XFF);//启动点位运动
```

5.5、JOG 运动 API

API	说明
GA_PrfJog	设置指定轴为 JOG 模式(速度模式)
GA_SetJogPrm	设置 JOG 模式运动参数
GA_SetJogPrmSingle	设置 JOG 模式运动参数（可替代 GA_SetJogPrm）
GA_GetJogPrm	读取 JOG 模式运动参数
GA_GetJogPrmSingle	读取 JOG 模式运动参数（可替代 GA_GetJogPrm）
GA_SetVel	设置目标速度
GA_Update	启动 JOG 运动

参数详细说明：

int GA_PrfJog(short nAxisNum)	
iAxis	iAxis
int GA_SetJogPrm(short nAxisNum,TJogPrm *pPrm)	
iAxis	iAxis

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 GA_SetJogPrmSingle 函数替代本函数

int GA_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 GA_GetJogPrm(short nAxisNum,TJogPrm *pPrm)	
iAxis	iAxis

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 GA_GetJogPrmSingle 函数替代本函数

int GA_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 GA_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 GA_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 += GA_SetCardNo(1);//切换到第 1 块板卡
iRes += GA_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 += GA_Reset();//复位板卡

iRes += GA_PrfJog(1);//设置轴 1 为 Jog 模式
iRes += GA_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 = GA_SetJogPrm(1,&JogPrm);//设置 JOG 运动参数

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

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

5.6、运动状态检测类 API

API	说明
GA_AxisOn	打开驱动器使能
GA_AxisOff	关闭驱动器使能
GA_Stop	停止一个或多个轴的规划运动，停止坐标系运动
GA_GetSts	读取轴状态
GA_ClrSts	清除驱动器报警标志、跟随误差越限标志、限位触发标志
GA_GetPrfPos	读取规划位置
GA_GetPrfVel	读取规划速度
GA_GetAxisEncPos	读取编码器位值
GA_GetAllSysStatus	获取所有板卡相关状态

参数详细说明：

int GA_AxisOn(short nAxisNum)	
nAxisNum	轴编号，取值范围：[1,AXIS_MAX_COUNT]
int GA_AxisOff(short nAxisNum)	
nAxisNum	轴编号，取值范围：[1,AXIS_MAX_COUNT]
int GA_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 GA_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 GA_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 GA_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 GA_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 GA_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 GA_GetAllSysStatus(TAllSysStatusData *pAllSysStatusData)	

pAllSysStatusData	typedef struct _AllSysStatusData
	{
	double dAxisEncPos[9];//轴编码器位置，包含一个手轮
	double dAxisPrfPos[8];//轴规划位置
	unsigned long lAxisStatus[8];//轴状态
	short nADCValue[2];//ADC值
	long lUserSegNum[2];//两个坐标系的用户段号
	long 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
	long lCrdSpace[2];//两个坐标系的剩余空间
	double dCrdVel[2];//两个坐标系的速度
	double dCrdPos[2][5];//两个坐标系的坐标
	long lLimitPosRaw;//正硬限位
	long lLimitNegRaw;//负硬限位
	long lAlarmRaw;//报警输入
	long lHomeRaw;//零位输入
	long lMPG;//手轮信号
	long lGpiRaw[4];//通用IO输入（主卡+3个扩展模块）

	}TAllSysStatusData;

示例代码：

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

iRes += GA_SetCardNo(1);//切换到第 1 块板卡
iRes += GA_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 += GA_Reset();//复位板卡

iRes += GA_AxisOn(1);//设置轴 1 使能
iRes += GA_AxisOff(1);//设置轴 1 断开使能

iRes += GA_Stop((0X0001 << (5-1)),0);//设置轴 5 停止运动
iRes += GA_GetSts(1,&lSts);//获取轴 1 状态
iRes += GA_ClrSts(1);//清除轴 1 运动状态
iRes += GA_GetPrfPos(1,&dPrfPos ,1,NULL)
iRes += GA_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 GA_LmtsOn 函数。

API	说明
GA_SetSoftLimit	设置软限位
GA_GetSoftLimit	获取软限位
GA_LmtsOn	设置轴限位信号有效
GA_LmtsOff	设置轴限位信号无效
GA_LmtSns	设置运动控制器各轴限位触发电平
GA_EStopSetIO	设置系统紧急停止 IO
GA_EStopOnOff	开启/关闭紧急停止功能
GA_EStopGetSts	获取紧急停止触发状态
GA_EStopClrSts	清除紧急停止触发状态

参数详细说明：

int GA_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 GA_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 GA_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 GA_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

<table>
<tr><td colspan="2">int GA_LmtSns(unsigned short nSense)</td></tr>
<tr><td>nSense</td><td>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
<table>
<tr><td rowspan="16">nSense
(16 位)</td><td>Bit0</td><td>轴 1 正限位逻辑电平（1 低电平触发，0 高电平触发）</td></tr>
<tr><td>Bit1</td><td>轴 1 负限位逻辑电平（1 低电平触发，0 高电平触发）</td></tr>
<tr><td>Bit2</td><td>轴 2 正限位逻辑电平（1 低电平触发，0 高电平触发）</td></tr>
<tr><td>Bit3</td><td>轴 2 负限位逻辑电平（1 低电平触发，0 高电平触发）</td></tr>
<tr><td>Bit4</td><td>轴 3 正限位逻辑电平（1 低电平触发，0 高电平触发）</td></tr>
<tr><td>Bit5</td><td>轴 3 负限位逻辑电平（1 低电平触发，0 高电平触发）</td></tr>
<tr><td>Bit6</td><td>轴 4 正限位逻辑电平（1 低电平触发，0 高电平触发）</td></tr>
<tr><td>Bit7</td><td>轴 4 负限位逻辑电平（1 低电平触发，0 高电平触发）</td></tr>
<tr><td>Bit8</td><td>轴 5 正限位逻辑电平（1 低电平触发，0 高电平触发）</td></tr>
<tr><td>Bit9</td><td>轴 5 负限位逻辑电平（1 低电平触发，0 高电平触发）</td></tr>
<tr><td>Bit10</td><td>轴 6 正限位逻辑电平（1 低电平触发，0 高电平触发）</td></tr>
<tr><td>Bit11</td><td>轴 6 负限位逻辑电平（1 低电平触发，0 高电平触发）</td></tr>
<tr><td>Bit12</td><td>轴 7 正限位逻辑电平（1 低电平触发，0 高电平触发）</td></tr>
<tr><td>Bit13</td><td>轴 7 负限位逻辑电平（1 低电平触发，0 高电平触发）</td></tr>
<tr><td>Bit14</td><td>轴 8 正限位逻辑电平（1 低电平触发，0 高电平触发）</td></tr>
<tr><td>Bit15</td><td>轴 8 负限位逻辑电平（1 低电平触发，0 高电平触发）</td></tr>
</table>

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

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

//将轴 1 和 2 的正负限位都设置为常开，低电平触发
GA_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 GA_LmtSnsEX，用法与本函数一样</td></tr>
<tr><td colspan="2">int GA_EStopSetIO(short nCardIndex,short nIOIndex,short nEStopSns,unsigned long lFilterTime)</td></tr>
<tr><td>nCardIndex</td><td>卡号，主卡为0，扩展IO卡依次为1、2、3、4.....</td></tr>
<tr><td>nIOIndex</td><td>IO索引，0~15</td></tr>
<tr><td>nEStopSns</td><td>电平逻辑，0不反转，1反转</td></tr>
<tr><td>lFilterTime</td><td>滤波时间，单位ms</td></tr>
<tr><td colspan="2">int GA_EStopOnOff(short nEStopOnOff)</td></tr>
<tr><td>nEStopOnOff</td><td>0 紧急停止功能关闭，1 紧急停止功能打开</td></tr>
<tr><td colspan="2">int GA_EStopGetSts(short *nStatus)</td></tr>
<tr><td>nStatus</td><td>0 紧急停止未触发，1 紧急停止触发</td></tr>
</table>

int GA_EStopClrSts()	

示例代码：

```
int iRes = 0;
long lSts = 0;
long lSoftLimPos= 0X7FFFFFFF;
long lSoftLimNeg=0X80000000

iRes += GA_SetCardNo(1);//切换到第 1 块板卡
iRes += GA_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 += GA_Reset();//复位板卡

iRes = GA_SetSoftLimit(1,lSoftLimPos,lSoftLimNeg);

iRes = GA_LmtsOn(1,-1);//将第一个轴的正限位和负限位都有效
iRes = GA_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 = GA_EStopOnOff(1);//紧急停止功能打开
iRes = GA_EStopGetSts(&nStatus);//获取紧急停止触发状态
iRes = GA_EStopClrSts();//清除紧急停止状态
```

5.8、其他指令 API

API	说明
GA_ZeroPos	清零轴的规划和编码器位置
GA_GetID	获取板卡唯一标识 ID
GA_SetAxisBand	设置轴到位误差带
GA_SetBacklash	设置反向间隙补偿的相关参数
GA_GetBacklash	读取反向间隙补偿的相关参数
GA_SetStopDec	设置平滑停止减速度和急停减速度

参数详细说明：

int GA_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 GA_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 GA_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 GA_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 GA_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

示例代码：

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

long lSoftLimPos= 0X7FFFFFFF;
long lSoftLimNeg=0X80000000

iRes += GA_SetCardNo(1);//切换到第 1 块板卡
iRes += GA_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 += GA_Reset();//复位板卡

iRes += GA_ZeroPos(1,8);//清零所有 8 个轴的位置
iRes += GA_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 = GA_SetBacklash(1,3,2,0);

iRes = GA_GetID(&ulID);
```

5.9、插补运动指令 API

API	说明
GA_SetCrdPrm	设置坐标系参数，确立坐标系映射，建立坐标系
GA_GetCrdPrm	查询坐标系参数
GA_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
GA_CrdClear	清除插补缓存区内的插补数据
GA_LnXY	缓存区指令，两维直线插补
GA_LnXYZ	缓存区指令，三维直线插补
GA_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）
GA_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）
GA_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）
GA_BufIO	缓存区指令，设置 IO 输出
GA_BufDelay	缓存区指令，延时一段时间
GA_BufMoveVel	在插补运动的过程中插入 BufferMove 轴的速度设定
GA_BufMoveAcc	在插补运动的过程中插入 BufferMove 轴的加速度设定
GA_BufMove	在插补运动的过程中插入阻塞和非阻塞的点位运动
GA_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
GA_CrdData	向插补缓存区增加插补数据
GA_CrdStart	启动插补运动
GA_SetOverride	设置插补运动目标合成速度倍率
GA_GetCrdPos	查询该坐标系的当前坐标位置值
GA_CrdSpace	读取插补缓存区中的剩余空间
GA_CrdStatus	查询插补运动坐标系状态
GA_SetUserSegNum	缓存区指令，设置自定义插补段段号
GA_GetUserSegNum	读取自定义插补段段号
GA_GetRemainderSegNum	读取未完成的插补段段数
GA_GetLookAheadSpace	获取前瞻缓冲区剩余空间

参数详细说明：

int GA_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：(0,32767)，单位：pulse/ms synAccMax：最大合成加速度。取值范围：(0,32767)，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 GA_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：(0,32767)，单位：pulse/ms synAccMax：该坐标系的最大合成加速度。取值范围：(0,32767)，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 GA_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 GA_CrdClear(short nCrdNum,short FifoIndex)	
nCrdNum	坐标系编号
FifoIndex	FifoIndex 索引
int GA_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：(0,32767)，单位：pulse/ms。
synAcc	插补段的合成加速度。取值范围：(0,32767)，单位：pulse/(ms*ms)。
velEnd	插补段的终点速度。取值范围：[0,32767)，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 GA_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：(0,32767)，单位：pulse/ms。
synAcc	插补段的合成加速度。取值范围：(0,32767)，单位：pulse/(ms*ms)。
velEnd	插补段的终点速度。取值范围：[0,32767)，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 GA_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：(0,32767)，单位：pulse/ms。
synAcc	插补段的合成加速度。取值范围：(0,32767)，单位：pulse/(ms*ms)。
velEnd	插补段的终点速度。取值范围：[0,32767)，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 GA_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：(0,32767)，单位：pulse/ms。
synAcc	插补段的合成加速度。取值范围：(0,32767)，单位：pulse/(ms*ms)。
velEnd	插补段的终点速度。取值范围：[0,32767)，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 GA_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：(0,32767)，单位：pulse/ms。
synAcc	插补段的合成加速度。取值范围：(0,32767)，单位：pulse/(ms*ms)。
velEnd	插补段的终点速度。取值范围：[0,32767)，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 GA_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
int GA_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 GA_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 GA_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 GA_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 GA_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 GA_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 GA_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。
int GA_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 GA_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 GA_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 GA_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 GA_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 GA_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 GA_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 GA_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 GA_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

5.10、硬件捕获类 API

API	说明
GA_SetCaptureMode	设置编码器捕获方式，并启动捕获
GA_GetCaptureMode	读取编码器捕获方式
GA_GetCaptureStatus	读取编码器捕获状态
GA_SetCaptureSense	设置捕获电平
GA_GetCaptureSense	获取捕获电平
GA_ClearCaptureStatus	清除捕获状态

参数详细说明：

int GA_SetCaptureMode(short nEncodeNum,short nMode)	
nEncodeNum	编码器轴号
nMode	编码器捕获模式 CAPTURE_HOME　Home 捕获 CAPTURE_INDEX　Index 捕获
int GA_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 GA_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 GA_SetCaptureSense(short nEncodeNum,short nMode ,short nSence)	
nEncodeNum	编码器起始轴号
nMode	捕获模式 CAPTURE_HOME　Home 捕获 CAPTURE_INDEX　Index 捕获
nSence	捕获电平 0：下降沿触发 1：上升沿触发
int GA_GetCaptureSense(short nEncodeNum,short nMode ,short *pSence)	
nEncodeNum	编码器号
nMode	捕获模式 CAPTURE_HOME　Home 捕获 CAPTURE_INDEX　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 GA_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/电子凸轮类 API

API	说明
GA_PrfGear	设置指定轴进入电子齿轮模式
GA_SetGearMaster	设置电子齿轮运动跟随主轴
GA_GetGearMaster	读取电子齿轮运动跟随主轴
GA_SetGearRatio	设置电子齿轮比
GA_GetGearRatio	获取电子齿轮比
GA_GearStart	启动电子齿轮运动
GA_GearStop	停止电子齿轮运动
GA_SetGearEvent	设置电子齿轮触发事件
GA_GetGearEvent	获取电子齿轮触发事件

参数详细说明：

int GA_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 GA_SetGearMaster(short nAxisNum,short nMasterAxisNum,short nMasterType=GEAR_MASTER_PROFILE)	
nAxisNum	轴号
nMasterAxisNum	主轴号
nMasterType	主轴类型，GEAR_MASTER_PROFILE，跟随规划，GEAR_MASTER_ENCODER,跟随编码器
int GA_GetGearMaster(short nAxisNum,short *pnMasterAxisNum,short *pMasterType=NULL)	
nAxisNum	轴号
pnMasterAxisNum	主轴号
pMasterType	主轴类型，GEAR_MASTER_PROFILE，跟随规划，GEAR_MASTER_ENCODER,跟随编码器
int GA_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 GA_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 GA_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 GA_GearStop(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 Bit7~0 对应 8~1 轴
int GA_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 GA_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

5.12、比较输出类 API

API	说明
GA_CmpPluse	设置比较器输出 IO 立即输出指定电平或者脉冲
GA_CmpBufSetChannel	设置比较缓冲区对应输出通道
GA_CmpBufData	向比较器缓冲区发送比较数据
GA_CmpBufSts	获取比较器缓冲区状态
GA_CmpBufStop	停止比较器缓冲区
GA_CmpRpt	设置比较器缓冲区等比输出
GA_CmpSetTriggerCount	设置比较器缓冲区触发计数初值
GA_CmpGetTriggerCount	获取比较器缓冲区触发计数初值

参数详细说明：

int GA_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 GA_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 GA_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 GA_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 GA_CmpBufStop(short nChannel)	

nChannel	bit0 代表通道 1，bit1 代表通道 2
int GA_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 GA_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 GA_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

5.13、自动回零相关 API

API	说明
GA_HomeStart	启动轴回零
GA_HomeStop	停止轴回零
GA_HomeSetPrm	设置回零参数
GA_HomeSetPrmSingle	设置回零参数（非结构体方式）
GA_HomeGetPrm	获取回零参数
GA_HomeGetPrmSingle	获取回零参数（非结构体方式）
GA_HomeGetSts	获取回零状态

参数详细说明：

int GA_HomeStart(short iAxisNum)	
nAxisNum	需要回零的轴号,取值范围：[1,AXIS_MAX_COUNT]
int GA_HomeStop(short iAxisNum)	
nAxisNum	需要停止回零的轴号,取值范围：[1,AXIS_MAX_COUNT]
int GA_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
	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 GA_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 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：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
int GA_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 GA_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：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
int GA_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;
AxisHomePrm.nHomeDir = 0;
AxisHomePrm.dHomeRapidVel = 5;
AxisHomePrm.dHomeLocatVel = 1;
AxisHomePrm.dHomeIndexVel = 1;
AxisHomePrm.dHomeAcc = 0.2;

AxisHomePrm.lOffset = 0;
//设置轴 1 回零参数
iRes = GA_HomeSetPrm(1,&AxisHomePrm);
//启动轴 1 回零
iRes = GA_HomeStart(1);
//获取轴 1 回零状态
iRes = GA_HomeGetSts(1,&nStatus);
```

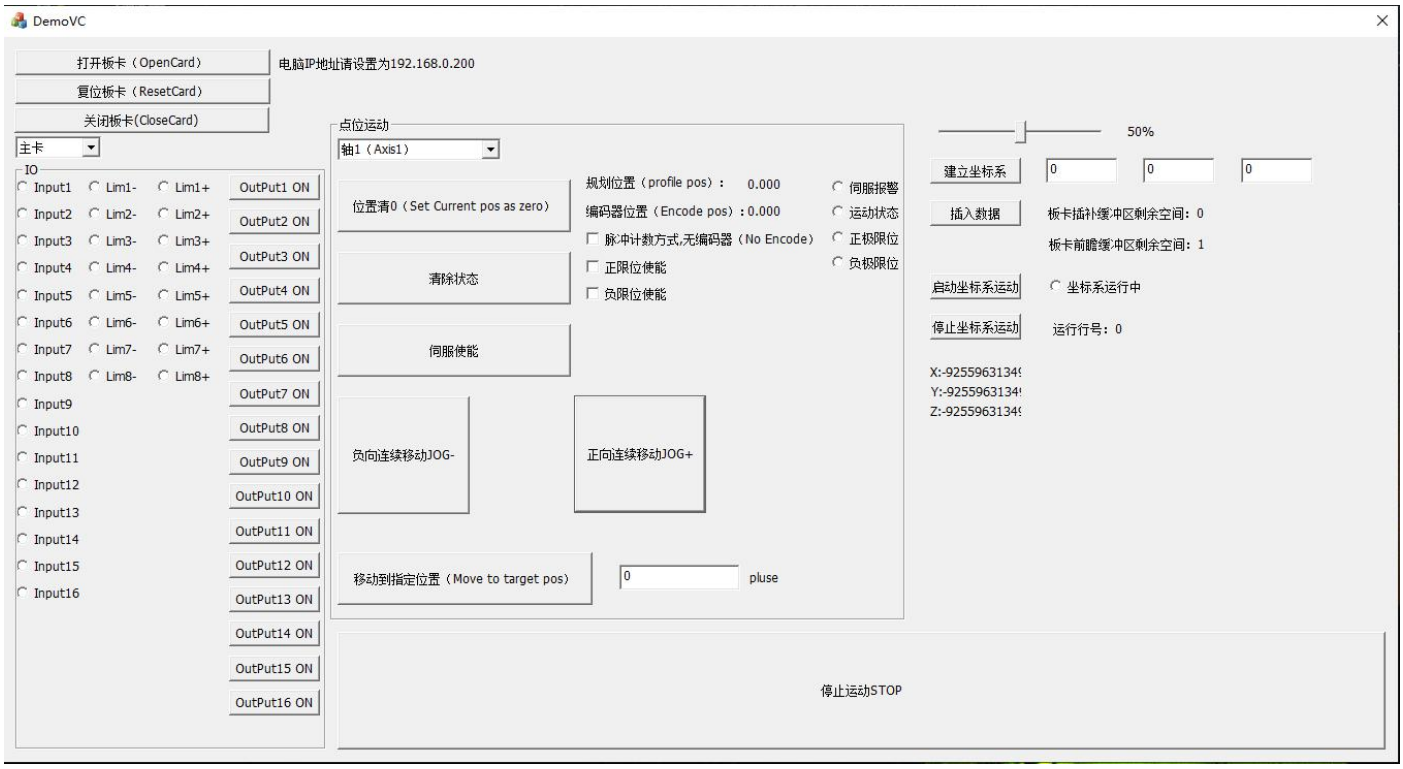
5.14、PT 模式相关 API

API	说明
GA_PrfPt	设置指定轴为 PT 模式
GA_PtSpace	读取指定轴的 PT 缓冲区空闲存储空间
GA_PtRemain	读取指定轴的 PT 存储空间尚未执行的数据长度
GA_PtData	向指定轴的 PT 缓冲区发送数据
GA_PtClear	清除指定轴 PT 缓冲区的数据
GA_PtStart	启动 PT 模式运动

参数详细说明：

int GA_PrfPt(short nAxisNum,short mode=PT_MODE_STATIC)	
nAxisNum	轴编号
mode	PT_MODE_STATIC 静态 PT_MODE_DYNAMIC 动态
int GA_PtSpace(short nAxisNum,long *pSpace,short nCount)	
nAxisNum	轴编号
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 GA_PtRemain(short nAxisNum,long *pRemainSpace,short nCount)	
nAxisNum	轴编号
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 GA_PtData(short nAxisNum,short* pData,long lLength,double dDataID)	
nAxisNum	轴编号
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 GA_PtClear(long lAxisMask)	
nAxisNum	轴编号
int GA_PtStart(long lAxisMask)	
nAxisNum	轴编号

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
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
负向连续移动JOG-
正向连续移动JOG+
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

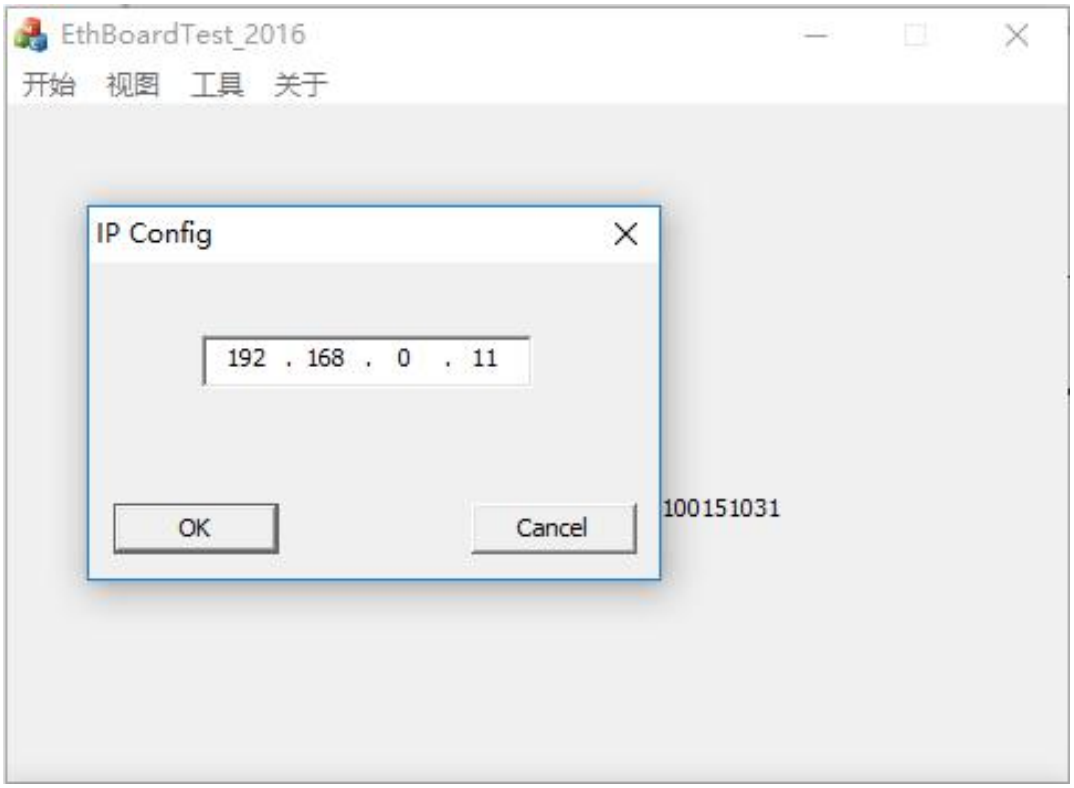
通过多个板卡并联的方案，一台电脑可以控制 16 轴、32 轴、48 轴....1024 轴

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：GA_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 += GA_SetCardNo(1);
iRes += GA_Open(0,"192.168.0.200");
iRes += GA_SetCardNo(2);
iRes += GA_Open(0,"192.168.0.200");
iRes += GA_SetCardNo(3);
iRes += GA_Open(0,"192.168.0.200");
```

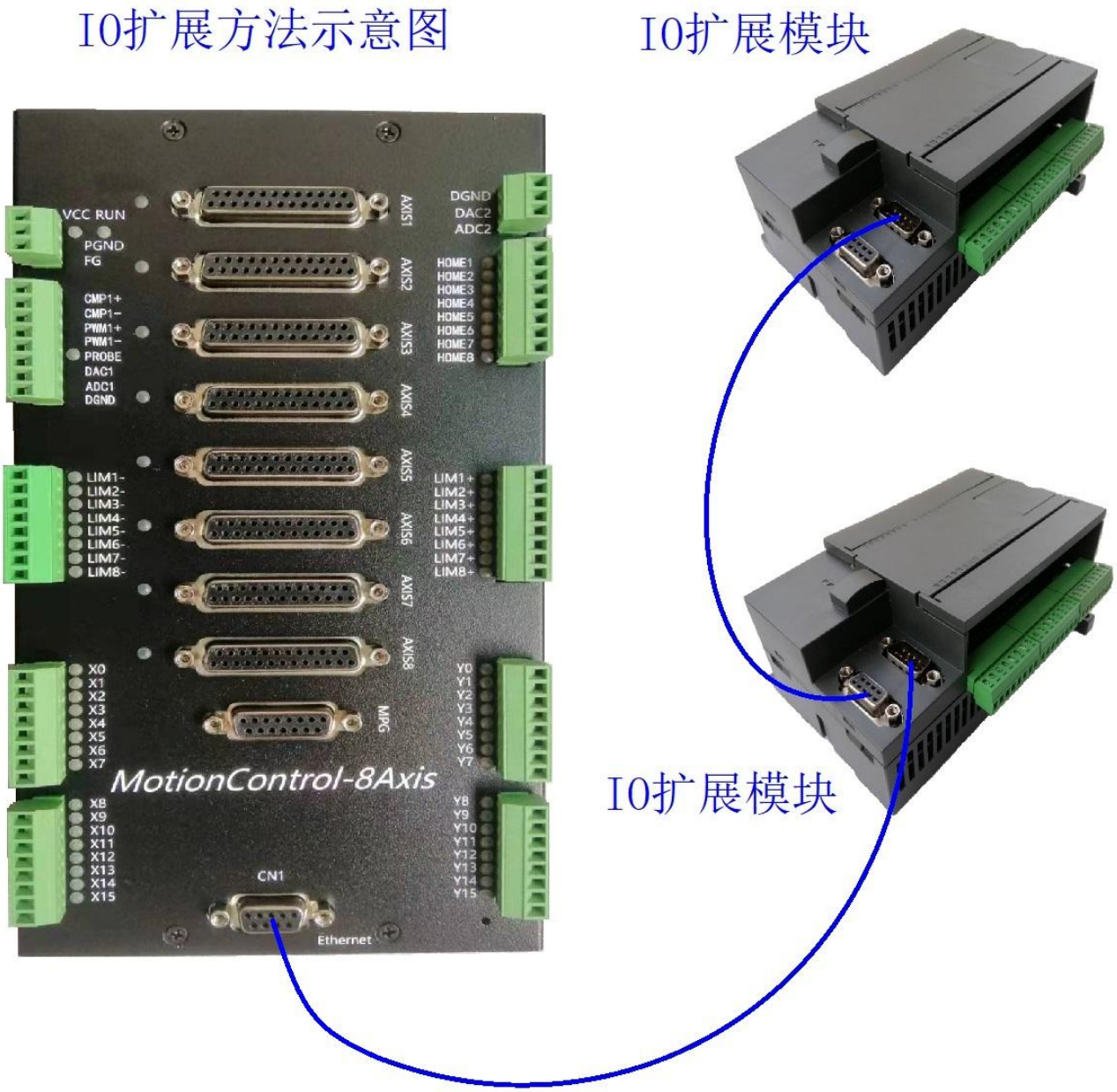
设置板卡 1 第 1 个 IO 输出，然后板卡 2 第 1 个 IO 输出，最后板卡 3 第 1 个 IO 输出代码为：

```
iRes += GA_SetCardNo(1);
iRes += GA_SetExtDoBit(0,0,1);
iRes += GA_SetCardNo(2);
iRes += GA_SetExtDoBit(0,0,1);
iRes += GA_SetCardNo(3);
iRes += GA_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。

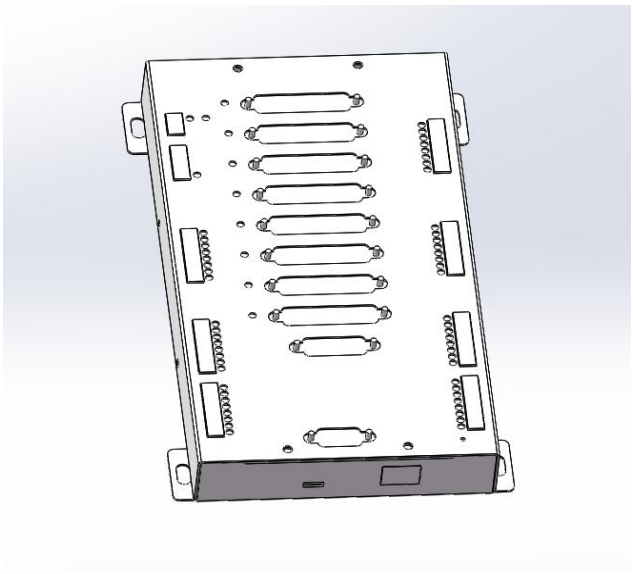
八、IO 扩展方法

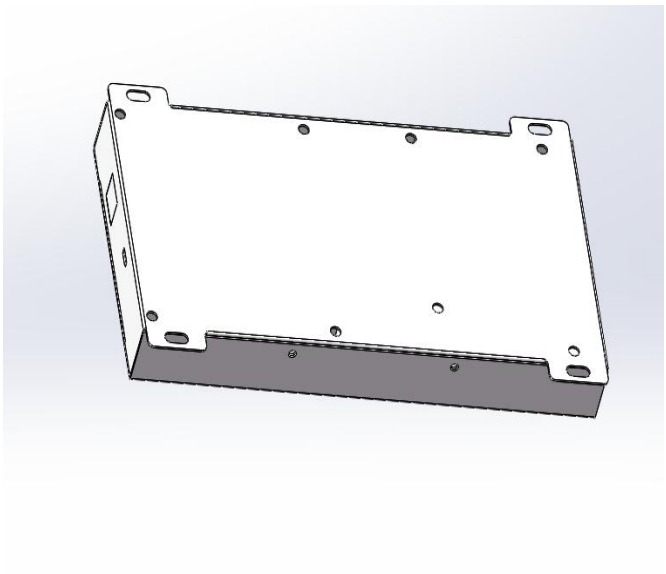
如果运动控制卡自身 IO 不够用，可选购我司 IO 扩展卡配套使用，通过一根 DB9 延长线串联即可，扩展协议为我司自定义总线协议，性能稳定，延迟小，且不占用电脑端口。

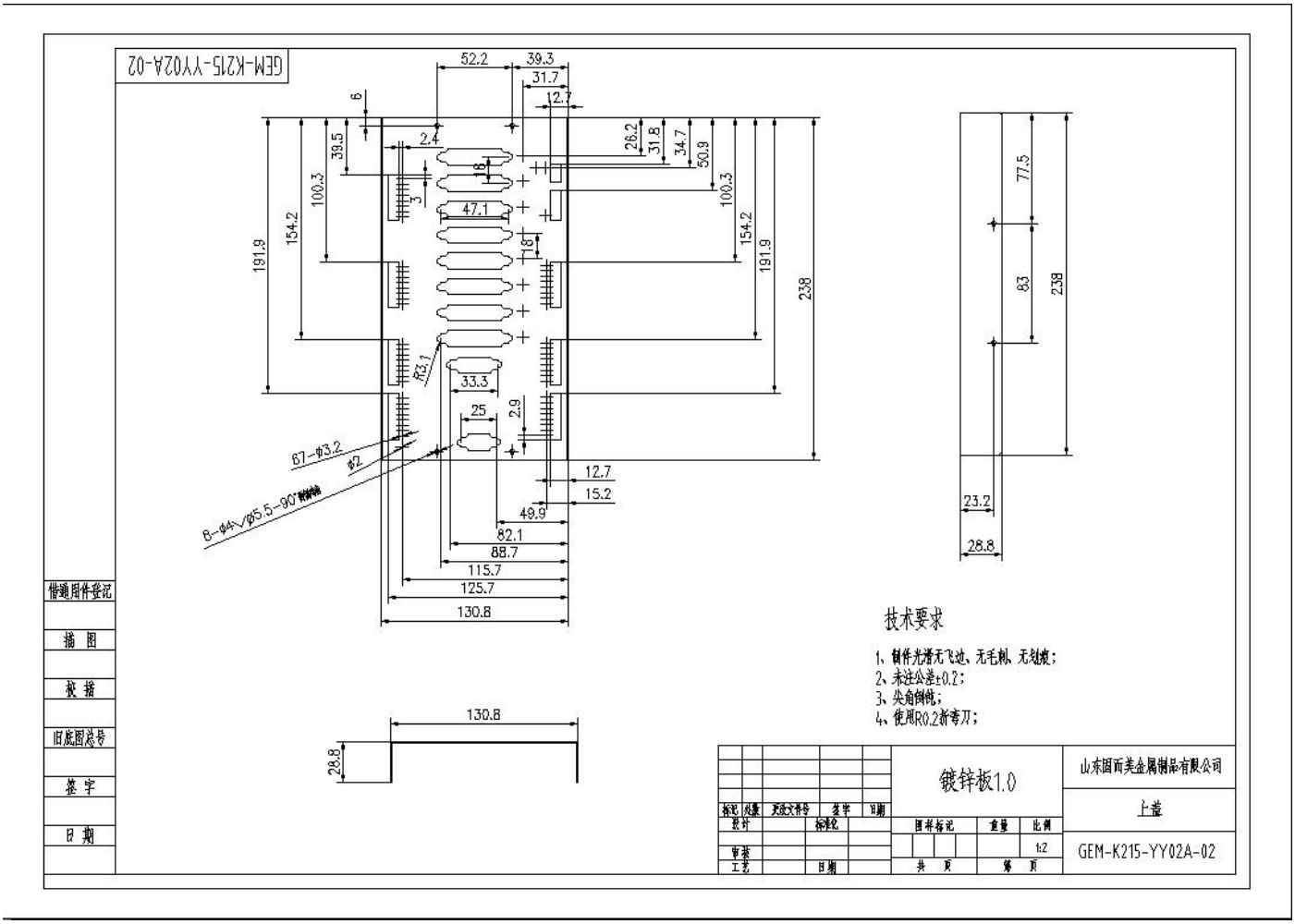


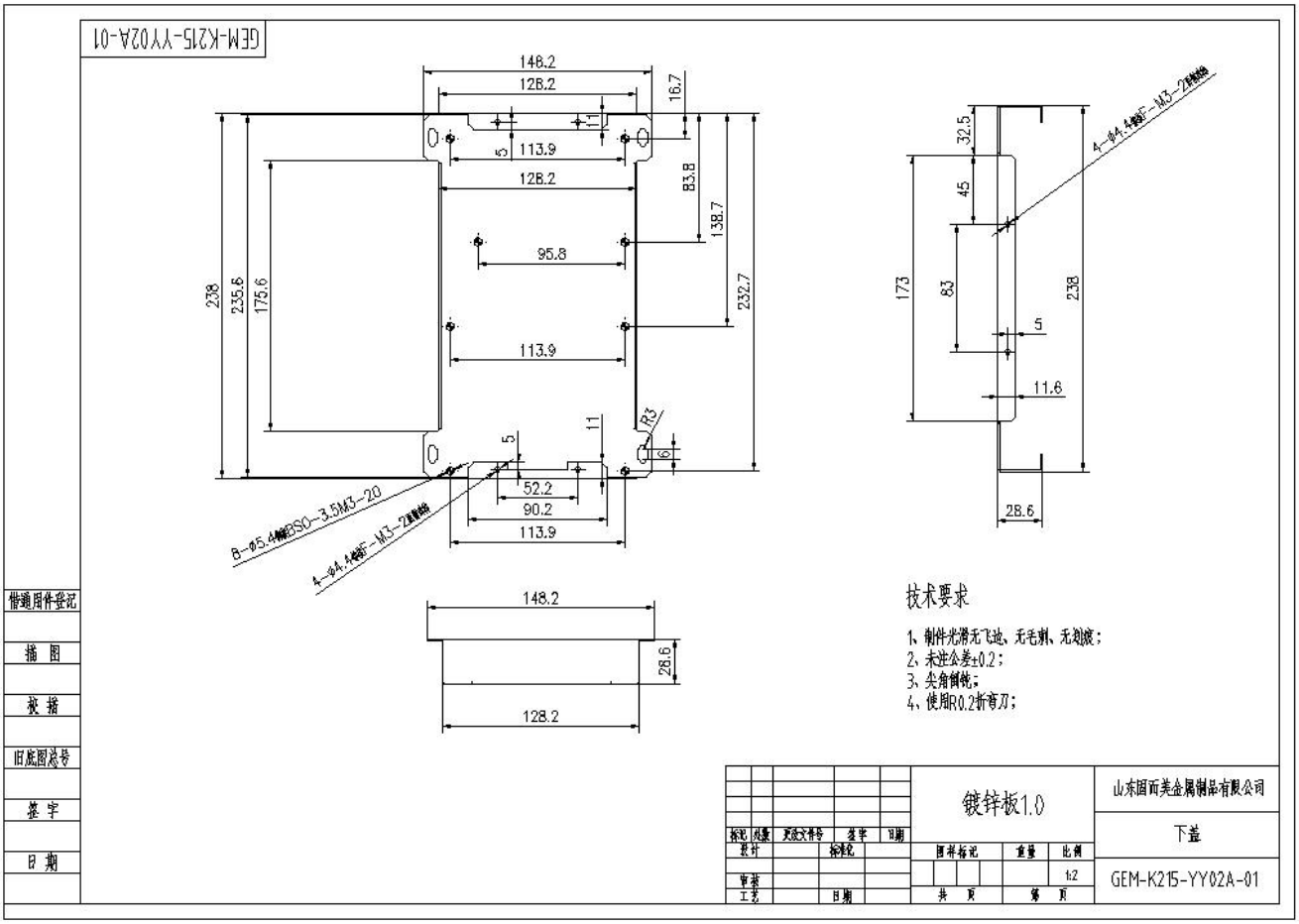
九、8 轴运动控制卡安装尺寸

外形尺寸（长*宽*高 mm）(该尺寸不含侧耳)	238*130.8*28.8
喷塑颜色	黑纱纹
配件	安装螺钉
板厚（mm）	1.0
其他事项	无






GEM-K215-YY02A-02
52.2
39.3
31.7
12.7
6
2.4
39.5
26.2
31.8
34.7
50.9
100.3
154.2
191.9
238
47.1
18
3
33.3
25
2.9
R3.7
67-φ3.2
φ2
8-φ4.5/φ5.5-90°
12.7
15.2
49.9
82.1
88.7
115.7
125.7
130.8
77.5
83
23.2
28.8
技术要求
1、钣件光滑无飞边、无毛刺、无划痕；
2、未注公差±0.2；
3、尖角倒钝；
4、使用R0.2折弯刀；
镀锌板1.0
山东固而美金属制品有限公司
上盖
GEM-K215-YY02A-02
1:2
借通用件登记
描 图
校 描
旧底图总号
签 字
日 期


GEM-K215-YY02A-01
148.2
128.2
16.7
11
5
113.9
128.2
83.8
138.7
95.8
232.7
238
235.6
175.6
113.9
R3
6
5
11
52.2
90.2
113.9
8-Φ5.4锪BSO-3.5M3-20
4-Φ4.4锪扩-M3-2深贯通
148.2
28.6
128.2
32.5
45
4-Φ4.4锪扩-M3-2深贯通
173
83
238
5
11.6
28.6
技术要求
1、制件光滑无飞边、无毛刺、无划痕；
2、未注公差±0.2；
3、尖角倒钝；
4、使用R0.2折弯刀；
普通用件登记
描 图
校 描
旧底图总号
签 字
日 期
镀锌板1.0
山东固而美金属制品有限公司
下盖
GEM-K215-YY02A-01
1:2

十、附录

附录一：API 一览

板卡打开关闭 API	
GA_SetCardNo	切换当前运动控制器卡号
GA_GetCardNo	读取当前运动控制器卡号
GA_Open	打开板卡
GA_Reset	复位板卡
GA_Close	关闭板卡
板卡配置类 API	
GA_AlarmOn	设置轴驱动报警信号有效
GA_AlarmOff	设置轴驱动报警信号无效
GA_AlarmSns	设置运动控制器轴报警信号电平逻辑
GA_LmtsOn	设置轴限位信号有效
GA_LmtsOff	设置轴限位信号无效
GA_LmtSns	设置运动控制器各轴限位触发电平
GA_EncOn	设置为“外部编码器”计数方式
GA_EncOff	设置为“脉冲计数器”计数方式
GA_EncSns	设置编码器的计数方向
GA_StepSns	设置脉冲输出通道的方向
IO 操作 API	
GA_GetDiRaw	读取数字 IO 输入状态的原始值
GA_GetDiReverseCount	读取数字量输入信号的变化次数
GA_SetDiReverseCount	设置数字量输入信号的变化次数的初值
GA_SetExtDoValue	设置 IO 输出（包含主模块和扩展模块）
GA_GetExtDiValue	获取 IO 输入（包含主模块和扩展模块）
GA_GetExtDoValue	获取 IO 输出（包含主模块和扩展模块）
GA_SetExtDoBit	设置指定 IO 模块的指定位输出（包含主模块和扩展模块）
GA_GetExtDiBit	获取指定 IO 模块的指定位输入（包含主模块和扩展模块）
GA_GetExtDoBit	获取指定 IO 模块的指定位输出（包含主模块和扩展模块）
轴点位运动 API	
GA_PrfTrap	设置指定轴为点位模式
GA_SetTrapPrm	设置点位模式运动参数
GA_SetTrapPrmSingle	设置点位模式运动参数（可替代 GA_SetTrapPrm）
GA_GetTrapPrm	读取点位模式运动参数
GA_GetTrapPrmSingle	读取点位模式运动参数（可替代 GA_GetTrapPrm）
GA_SetPos	设置目标位置
GA_SetVel	设置目标速度
GA_Update	启动点位运动
轴 JOG 运动 API	
GA_PrfJog	设置指定轴为 JOG 模式(速度模式)

GA_SetJogPrm	设置 JOG 模式运动参数
GA_SetJogPrmSingle	设置 JOG 模式运动参数（可替代 GA_SetJogPrm）
GA_GetJogPrm	读取 JOG 模式运动参数
GA_GetJogPrmSingle	读取 JOG 模式运动参数（可替代 GA_GetJogPrm）
GA_SetVel	设置目标速度
GA_Update	启动 JOG 运动
运动状态检测类 API	
GA_AxisOn	打开驱动器使能
GA_AxisOff	关闭驱动器使能
GA_Stop	停止一个或多个轴的规划运动，停止坐标系运动
GA_GetSts	读取轴状态
GA_ClrSts	清除驱动器报警标志、跟随误差越限标志、限位触发标志
GA_GetPrfPos	读取规划位置
GA_GetAxisEncPos	读取编码器位值
安全机制 API	
GA_SetSoftLimit	设置软限位
GA_GetSoftLimit	获取软限位
其他 API	
GA_ZeroPos	清零轴的规划和编码器位置
GA_GetID	获取板卡唯一标识 ID
GA_SetAxisBand	设置轴到位误差带
GA_SetBacklash	设置反向间隙补偿的相关参数
GA_GetBacklash	读取反向间隙补偿的相关参数
GA_SetStopDec	设置平滑停止减速度和急停减速度
插补运动指令 API	
GA_SetCrdPrm	设置坐标系参数，确立坐标系映射，建立坐标系
GA_GetCrdPrm	查询坐标系参数
GA_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
GA_CrdClear	清除插补缓存区内的插补数据
GA_LnXY	缓存区指令，两维直线插补
GA_LnXYZ	缓存区指令，三维直线插补
GA_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）
GA_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）
GA_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）
GA_BufIO	缓存区指令，设置 IO 输出
GA_BufDelay	缓存区指令，延时一段时间
GA_BufMoveVel	在插补运动的过程中插入 BufferMove 轴的速度设定
GA_BufMoveAcc	在插补运动的过程中插入 BufferMove 轴的加速度设定
GA_BufMove	在插补运动的过程中插入阻塞和非阻塞的点位运动
GA_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
GA_CrdData	向插补缓存区增加插补数据
GA_CrdStart	启动插补运动
GA_SetOverride	设置插补运动目标合成速度倍率
GA_GetCrdPos	查询该坐标系的当前坐标位置值

GA_CrdSpace	读取插补缓存区中的剩余空间
GA_CrdStatus	查询插补运动坐标系状态
GA_SetUserSegNum	缓存区指令，设置自定义插补段段号
GA_GetUserSegNum	读取自定义插补段段号
GA_GetRemainderSegNum	读取未完成的插补段段数
GA_GetLookAheadSpace	获取前瞻缓冲区剩余空间
硬件捕获类 API	
GA_SetCaptureMode	设置编码器捕获方式，并启动捕获
GA_GetCaptureMode	读取编码器捕获方式
GA_GetCaptureStatus	读取编码器捕获状态
GA_SetCaptureSense	设置捕获电平
GA_GetCaptureSense	获取捕获电平
GA_ClearCaptureStatus	清除捕获状态
Gear/电子齿轮/电子凸轮类 API	
GA_PrfGear	设置指定轴进入电子齿轮模式
GA_SetGearMaster	设置电子齿轮运动跟随主轴
GA_GetGearMaster	读取电子齿轮运动跟随主轴
GA_SetGearRatio	设置电子齿轮比
GA_GetGearRatio	获取电子齿轮比
GA_GearStart	启动电子齿轮运动
GA_GearStop	停止电子齿轮运动
GA_SetGearEvent	设置电子齿轮触发事件
GA_GetGearEvent	获取电子齿轮触发事件
比较输出类 API	
GA_CmpPluse	设置比较器输出 IO 立即输出指定电平或者脉冲
GA_CmpBufSetChannel	设置比较缓冲区对应输出通道
GA_CmpBufData	向比较器缓冲区发送比较数据
GA_CmpBufSts	获取比较器缓冲区状态
GA_CmpBufStop	停止比较器缓冲区
GA_CmpRpt	设置比较器缓冲区等比输出
GA_CmpSetTriggerCount	设置比较器缓冲区触发计数初值
GA_CmpGetTriggerCount	获取比较器缓冲区触发计数初值
PT 模式 API	
GA_PrfPt	设置指定轴为 PT 模式
GA_PtSpace	读取指定轴的 PT 缓冲区空闲存储空间
GA_PtRemain	读取指定轴的 PT 存储空间尚未执行的数据长度
GA_PtData	向指定轴的 PT 缓冲区发送数据
GA_PtClear	清除指定轴 PT 缓冲区的数据
GA_PtStart	启动 PT 模式运动

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 GA_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 GA_LmtsOn 这个函数，具体参见章节 5.7“安全机制 API”

11.5、调用 GA_Stop 函数停止加速度不够快，怎么调整？

GA_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”