# 多轴运动控制卡

# ADT-8960

## 说明书基本信息

本说明书由深圳众为兴技术股份有限公司组织编写。

本说明书主要编写人：艾小云。本说明书于 2008 年 6 月 3 日首次发布，当前版本号 16.06.25.17，项目号 YJ20061226。

## 版权声明

除此以外，本手册提到的产品及其资料仅供参考，内容如有更新，恕不另行通知。



深圳众为兴技术股份有限公司

# 目　　录

# 概述篇

# 第一章 产品简介

## 1.1 描述:

ADT-8960 卡是基于PCI 总线的高性能六轴伺服/步进控制卡，一个系统中可支持多达16 块控制卡，可控制96 路伺服/步进电机，支持即插即用。

脉冲输出方式可用单脉冲（脉冲+方向）或双脉冲（脉冲+脉冲）方式，最大脉冲频率2MHz，采用先进技术，保证在输出频率很高的时候，频率误差小于0.1%。

支持任意2-6 轴直线插补，最大插补速度1MHz。

位置管理采用两个加/减计数器，一个用于管理内部驱动脉冲输出的逻辑位置计数器，一个用于接收外部的输入，输入信号是A/B 相输入的编码器或光栅尺，作为实际位置计数器

计数器位数高达32 位，最大范围－2,147,483,648~+2,147,483,647。

提供DOS、WINDOWS95/98/NT/2000/XP/WINCE 开发库，可用VC++、VB、BC++、LabVIEW、Delphi、C++Builder 等进行软件开发。

# 第二章 主要性能

## 2.1 描述:

- 32 位 PCI 总线，即插即用
- 最大脉冲输出频率为 2MHz
- 6 轴步进/伺服电机控制，每轴可以独立控制，互不影响
- 任意 2-6 轴硬件直线插补，多轴连续运动
- 外部信号(手轮或通用输入信号)定量驱动、连续驱动
- 位置锁存，该功能用于位置测量十分准确、方便
- 手动减速功能
- 大容量硬件缓存功能

- 梯形加减速或 S 型加减速功能
- 6 轴增量编码器输入，32 位计数，频率高达 2MHz，最大计数范围 -2147483648~+2147493647
- 脉冲输出方式：脉冲/方向，脉冲/脉冲
- 带光耦隔离 32 路数字输入/16 路数字输出信号，包括每轴 2 个正负限位信号，另外还有 16 路可配置的输入输出信号可以调用，可以全部做输入信号也可以做输出信号。
- 运动中可实时改变速度及目标位置
- 6 轴可同动同停，具有硬件急停功能，安全性高
- 运动中可实时读取逻辑位置、实际位置、驱动速度、加速度、驱动状态
- 每轴有 2 个 STOP 信号，可用于原点搜寻、编码器 Z 相搜寻
- 可接受伺服驱动器的各种信号，如编码器 Z 相信号、到位信号、报警信号等
- 同一系统中支持多达 16 个控制卡，共 96 个轴
- 操作系统：DOS、WINDOWS95/98/NT/2000/XP、WINCE、WIN7
- 编程环境：C/BC++/VC/VB/C#/C++Builder/Delphi/LabVIEW/EVC

开放式 DOS 和 Windows 的应用示例

# 第三章 应用范围

## 3.1 应用范围:

- 机器视觉、自动检测设备、AOI；
- 生物、医学自动采样设备；
- 切割设备：钻石切割机、海绵切割机；
- 点胶行业；
- 半导体封装行业：固晶机；
- 广告行业：数控雕字机；
- 包装印刷设备：印刷机、移印机；
- 雕刻设备；
- 工业机器人设备；

- PCB 加工、SMT 等行业；
- 打螺丝机，螺母植入设备

# 硬件篇

# 第一章 硬件安装

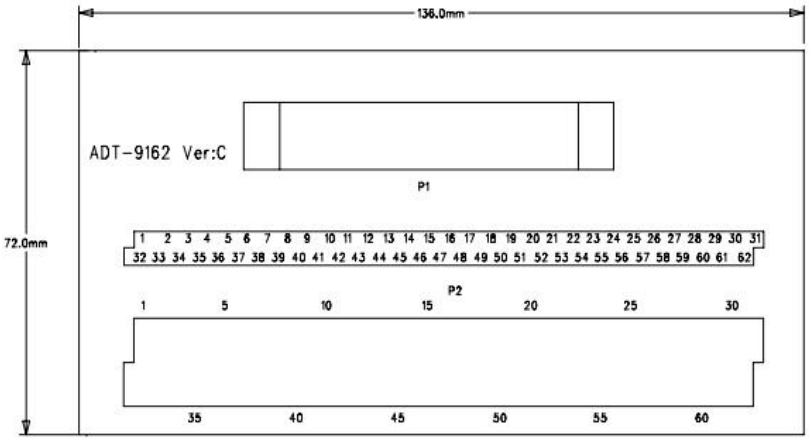
## 1.1 出货配置:

| ADT-8960 旧版配件 | | | |
|---|---|---|---|
| 序号 | 型号 | 规格 | 数量 |
| 1 | ADT-8960 | 用户手册（本手册） | 1 |
| 2 | ADT-8960 | 四轴 PCI 总线高性能运动控制卡 | 1 |
| 3 | ADT-8960 | 用户光盘 | 1 |
| 4 | D62GG | 62 芯屏蔽连接线 | 2 |
| 5 | DB64 | 64 芯扁平线 | 1 |

| | | | |
|---|---|---|---|
| 6 | ADT-9164 (VER:A) | 转接板，A 版母头 | 1 |
| 7 | ADT-9162 | 接线端子,62 芯母头(可与 ADT-9162A 通用) | 2 |
| 下面为 ADT-8960 新版防呆配件 | | | |
| 1 | ADT-8960 | 用户手册（本手册） | 1 |
| 2 | ADT-8960 | 四轴 PCI 总线高性能运动控制卡 | 1 |
| 3 | ADT-8960 | 用户光盘 | 1 |
| 4 | ADT-62GM | 62 芯屏蔽连接线,公对母 | 2 |
| 5 | DB64 | 64 芯扁平线 | 1 |
| 6 | ADT-9164 (VER:B) | 转接板，B 版公头 | 1 |
| 7 | ADT-9162A | 接线端子,62 芯母头(可与 ADT-9162 通用) | 1 |
| 8 | ADT-9162B | 接线端子,62 芯公头 | 1 |

**1.2 安装尺寸:**

**ADT-9162 尺寸图:**




## 1.3 安装步骤：

旧版配件连接方式：

1. 关闭电脑电源（注：ATX 电源需总电源关闭）。
2. 打开电脑机箱后盖。
3. 选择一条未占用的PCI 插槽,插入ADT-8960。
4. 确保 ADT-8960 的金手指完整插入 PCI 插槽,拧紧螺丝。
5. 将D62GG 连接线的一端和控制卡的J1 接口相连，另一端和ADT-9162 接线端子相连。
6. 根据用户情况决定是否安装J2 接口线。安装J2 的步骤：
   (1)将ADT-DB64 扁平线的一端和控制卡的J2 相连，另一端和ADT-DB64 转接板的P2 相连；
   (2)在机箱后面固定好ADT-DB64(VER:A)转接板；
   (3)将ADT-D62GG 分别和转接板的P2 和ADT-9162 接线端子相连。

新版配件连接方式：

1. 关闭电脑电源（注：ATX 电源需总电源关闭）。
2. 打开电脑机箱后盖。
3. 选择一条未占用的PCI 插槽,插入ADT-8960。

4. 确保 ADT-8960 的金手指完整插入 PCI 插槽,拧紧螺丝。
5. 将D62GM 连接线的一端和控制卡的J1 接口相连，另一端和ADT-9162B 接线端子相连。
6. 根据用户情况决定是否安装J2 接口线。安装J2 的步骤：

(1)将ADT-DB64 扁平线的一端和控制卡的J2 相连，另一端和ADT-DB64 转接板的P2 相连；

(2)在机箱后面固定好ADT_DB64(VER:B)转接板；

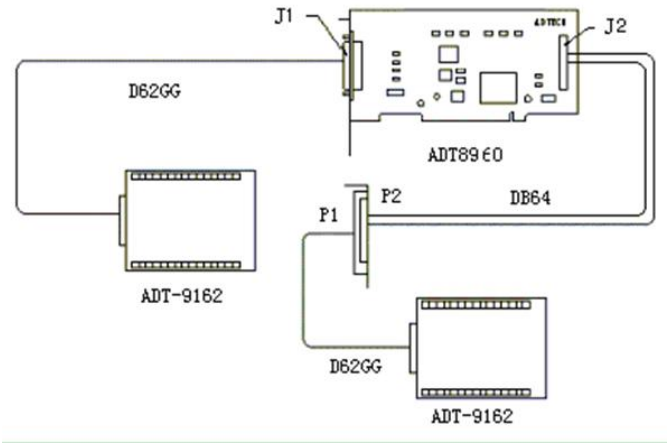
(3)将ADT-D62GM 分别和转接板的P2 和ADT-9162A 接线端子相连。

# 第二章 电气连接

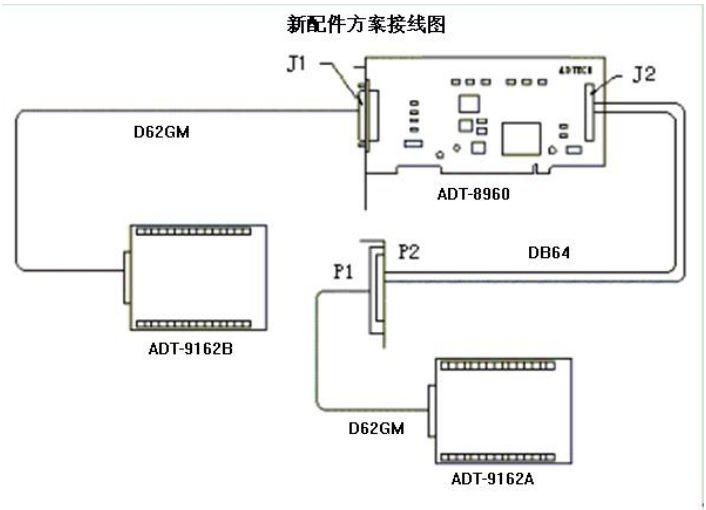
## 2.1 接线图：

旧配件方案接线图：

旧配件接线图



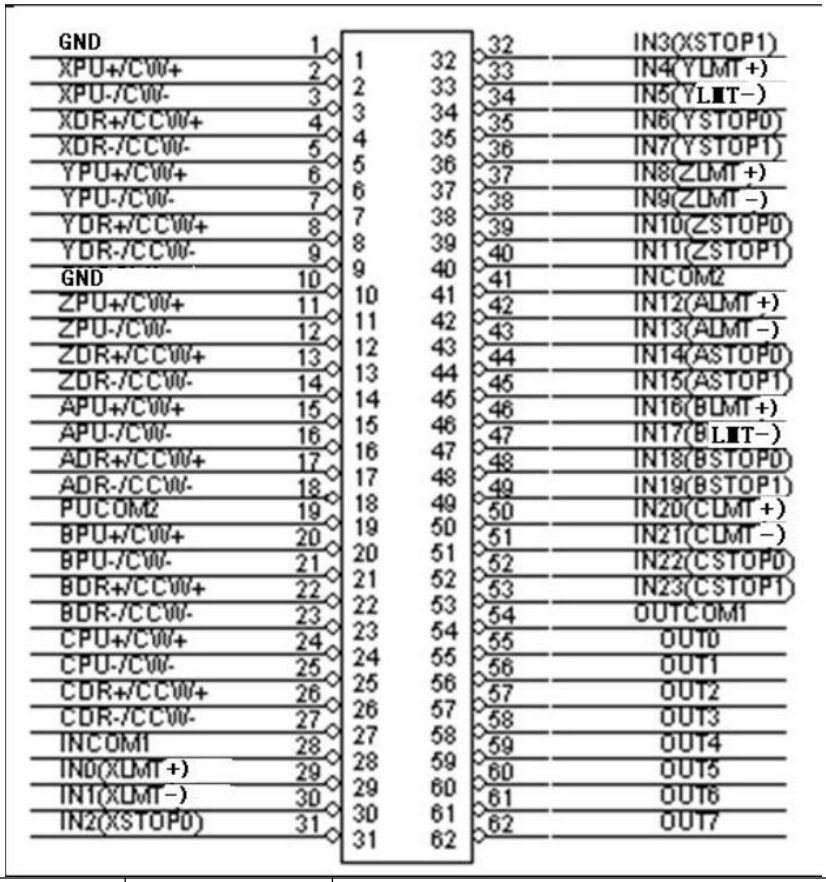

新配件方案接线图：




一块 ADT-8960 卡有二个输入/输出接口，其中 J1、J2 为 62 针插座， **J1** 为 X、Y、Z、A、B、C 轴的脉冲输出、开关量输入和开关量输出 OUT0-OUT7 的信号接线；**J2** 为 X、Y、Z、A、B、C 轴的编码器输入和开关量输入的信号接线，以及开关量输出 OUT8-OUT15 信号的接线；

信号定义如下：

## 2.2 J1 线号说明



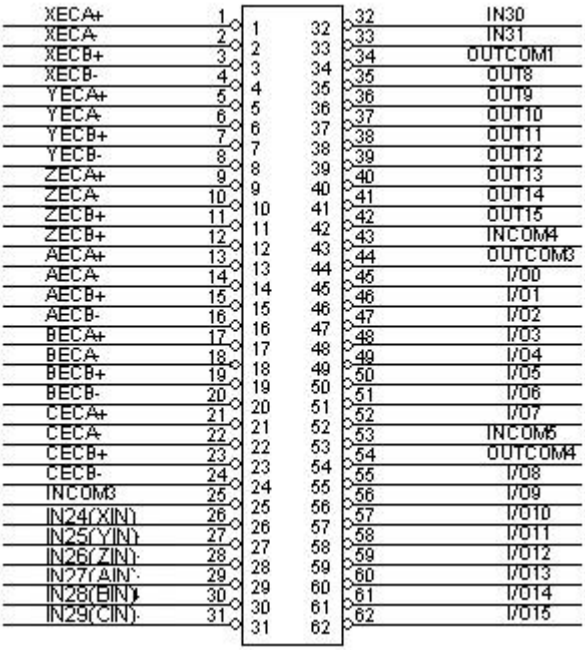

| 线 号 | 符 号 | 说 明 |
|---|---|---|
| 1 | GND | 内部电源地（注意：Ver:E 版本之后为内部电源地，之前的旧版为+5V 电源输出） |
| 2 | XPU+/CW+ | X 脉冲信号+ |
| 3 | XPU-/CW- | X 脉冲信号- |

| | | |
|---|---|---|
| 4 | XDR+/CCW+ | X 方向信号+ |
| 5 | XDR-/CCW- | X 方向信号- |
| 6 | YPU+/CW+ | Y 脉冲信号+ |
| 7 | YPU-/CW- | Y 脉冲信号- |
| 8 | YDR+/CCW+ | Y 方向信号+ |
| 9 | YDR-/CCW- | Y 方向信号- |
| 10 | GND | 内部电源地（注意：Ver:E 版本之后为内部电源地，之前的旧版为+5V 电源输出） |
| 11 | ZPU+/CW+ | Z 脉冲信号+ |
| 12 | ZPU-/CW- | Z 脉冲信号- |
| 13 | ZDR+/CCW+ | Z 方向信号+ |
| 14 | ZDR-/CCW- | Z 方向信号- |
| 15 | APU+/CW+ | A 脉冲信号+ |
| 16 | APU-/CW- | A 脉冲信号- |
| 17 | ADR+/CCW+ | A 方向信号+ |
| 18 | ADR-/CCW- | A 方向信号- |
| 19 | PUCOM | 用于单端输入的驱动器，脉冲共阳极接法使用，不可接外接电源 |
| 20 | BPU+/CW+ | B 脉冲信号+ |
| 21 | BPU-/CW- | B 脉冲信号- |
| 22 | BDR+/CCW+ | B 方向信号+ |
| 23 | BDR-/CCW- | B 方向信号- |
| 24 | CPU+/CW+ | C 脉冲信号+ |
| 25 | CPU-/CW- | C 脉冲信号- |
| 26 | CDR+/CCW+ | C 方向信号+ |
| 27 | CDR-/CCW- | C 方向信号- |
| 28 | INCOM1 | Ver:E 版本后为 IN0-5 开关量输入点公共端，之前的旧版本为 IN0-11 开关量输入点公共端 |
| 29 | IN0(XLMT+) | X 正向限位信号，可做通用输入信号 |

| | | |
|---|---|---|
| 30 | IN1(XLMT-) | X 反向限位信号，可做通用输入信号 |
| 31 | IN2(XSTOP0) | X 原点信号 0，手动减速原点，可做通用输入信号 |
| 32 | IN3(XSTOP1) | X 停止信号 1，可做通用输入信号 |
| 33 | IN4(YLMT+) | Y 正向限位信号，可做通用输入信号 |
| 34 | IN5(YMT-) | Y 反向限位信号，可做通用输入信号 |
| 35 | IN6(YSTOP0) | Y 原点信号 0，手动减速原点，可做通用输入信号 |
| 36 | IN7(YSTOP1) | Y 停止信号 1，可做通用输入信号 |
| 37 | IN8(ZLMT+) | Z 正向限位信号，可做通用输入信号 |
| 38 | IN9(ZLMT-) | Z 反向限位信号，可做通用输入信号 |
| 39 | IN10(ZSTOP0) | Z 原点信号 0，手动减速原点，可做通用输入信号 |
| 40 | IN11(ZSTOP1) | Z 停止信号 1，可做通用输入信号 |
| 41 | INCOM2 | Ver:E 版本后为 IN6-23 开关量输入点公共端，之前的旧版本为 IN12-23 开关量输入点公共端 |
| 42 | IN12(ALMT+) | A 正向限位信号，可做通用输入信号 |
| 43 | IN13(ALMT-) | A 反向限位信号，可做通用输入信号 |
| 44 | IN14(ASTOP0) | A 原点信号 0，手动减速原点，可做通用输入信号 |
| 45 | IN15(ASTOP1) | A 停止信号 1，可做通用输入信号 |
| 46 | IN16(BLMT+) | B 正向限位信号，可做通用输入信号 |
| 47 | IN17(BLMT-) | B 反向限位信号，可做通用输入信号 |
| 48 | IN18(BSTOP0) | B 原点信号 0，手动减速原点，可做通用输入信号 |
| 49 | IN19(BSTOP1) | B 停止信号 1，可做通用输入信号 |
| 50 | IN20(CLMT+) | C 正向限位信号，可做通用输入信号 |
| 51 | IN21(CLMT-) | C 反向限位信号，可做通用输入信号 |

| | | |
|---|---|---|
| 52 | IN22(CSTOP0) | C 原点信号 0，手动减速原点，可做通用输入信号 |
| 53 | IN23(CSTOP1) | C 停止信号 1，可做通用输入信号 |
| 54 | OUTCOM1 | OUT0-OUT7 的公共端 |
| 55 | OUT0 | 开关量输出点 |
| 56 | OUT1 | 开关量输出点 |
| 57 | OUT2 | 开关量输出点 |
| 58 | OUT3 | 开关量输出点 |
| 59 | OUT4 | 开关量输出点 |
| 60 | OUT5 | 开关量输出点 |
| 61 | OUT6 | 开关量输出点 |
| 62 | OUT7 | 开关量输出点 |

## 2.3 J2 线号说明



| 线号 | 符号 | 说明 |
|---|---|---|
| 1 | XECA+ | X 轴编码器 A 相输入+ |
| 2 | XECA- | X 轴编码器 A 相输入- |
| 3 | XECB+ | X 轴编码器 B 相输入+ |
| 4 | XECB- | X 轴编码器 B 相输入- |
| 5 | YECA+ | Y 轴编码器 A 相输入+ |
| 6 | YECA- | Y 轴编码器 A 相输入- |
| 7 | YECB+ | Y 轴编码器 B 相输入+ |
| 8 | YECB- | Y 轴编码器 B 相输入- |
| 9 | ZECA+ | Z 轴编码器 A 相输入+ |
| 10 | ZECA- | Z 轴编码器 A 相输入- |
| 11 | ZECB+ | Z 轴编码器 B 相输入+ |
| 12 | ZECB- | Z 轴编码器 B 相输入- |
| 13 | AECA+ | A 轴编码器 A 相输入+ |
| 14 | AECA- | A 轴编码器 A 相输入- |
| 15 | AECB+ | A 轴编码器 B 相输入+ |
| 16 | AECB- | A 轴编码器 B 相输入- |
| 17 | BECA+ | B 轴编码器 A 相输入+ |
| 18 | BECA- | B 轴编码器 A 相输入- |
| 19 | BECB+ | B 轴编码器 B 相输入+ |
| 20 | BECB- | B 轴编码器 B 相输入- |
| 21 | CECA+ | C 轴编码器 A 相输入+ |
| 22 | CECA- | C 轴编码器 A 相输入- |
| 23 | CECB+ | C 轴编码器 B 相输入+ |
| 24 | CECB- | C 轴编码器 B 相输入- |
| 25 | INCOM3 | IN24－IN31 的公共端 |
| 26 | IN24(XIN) | 开关量输入点（X 轴位置锁存） |
| 27 | IN25(YIN) | 开关量输入点（Y 轴位置锁存） |
| 28 | IN26(ZIN) | 开关量输入点（Z 轴位置锁存） |

| | | |
|---|---|---|
| 29 | IN27(AIN) | 开关量输入点（A 轴位置锁存） |
| 30 | IN28(BIN) | 开关量输入点（B 轴位置锁存） |
| 31 | IN29(CIN) | 开关量输入点（C 轴位置锁存） |
| 32 | IN30 | 开关量输入点 |
| 33 | IN31 | 开关量输入点（全轴急停） |
| 34 | OUTCOM2 | OUT8－OUT15 的公共端 |
| 35 | OUT8 | 开关量输出点 |
| 36 | OUT9 | 开关量输出点 |
| 37 | OUT10 | 开关量输出点 |
| 38 | OUT11 | 开关量输出点 |
| 39 | OUT12 | 开关量输出点 |
| 40 | OUT13 | 开关量输出点 |
| 41 | OUT14 | 开关量输出点 |
| 42 | OUT15 | 开关量输出点 |
| 43 | INCOM4 | I/O0-I/O7 作为输入的公共端 |
| 44 | OUTCOM3 | I/O0-I/O7 作为输出的公共端 |
| 45 | I/O0 | 可配置输入输出（X 轴手动正转），配置为输入是 IN44，配置为输出是 OUT16 |
| 46 | I/O1 | 可配置输入输出（X 轴手动反转），配置为输入是 IN45，配置为输出是 OUT17 |
| 47 | I/O2 | 可配置输入输出（Y 轴手动正转），配置为输入是 IN46，配置为输出是 OUT18 |
| 48 | I/O3 | 可配置输入输出（Y 轴手动反转），配置为输入是 IN47，配置为输出是 OUT19 |
| 49 | I/O4 | 可配置输入输出（Z 轴手动正转），配置为输入是 IN48，配置为输出是 OUT20 |
| 50 | I/O5 | 可配置输入输出（Z 轴手动反转），配置为输入是 IN49，配置为输出是 OUT21 |
| 51 | I/O6 | 可配置输入输出（A 轴手动正转），配置为输入是 IN50，配置为输出是 OUT22 |

| | | |
|---|---|---|
| 52 | I/O7 | 可配置输入输出（A 轴手动反转），配置为输入是 IN51，配置为输出是 OUT23 |
| 53 | INCOM5 | I/O8-I/O15 作为输入的公共端 |
| 54 | OUTCOM4 | I/O8-I/O15 作为输出的公共端 |
| 55 | I/O8 | 可配置输入输出（B 轴手动正转），配置为输入是 IN52，配置为输出是 OUT24 |
| 56 | I/O9 | 可配置输入输出（B 轴手动反转），配置为输入是 IN53，配置为输出是 OUT25 |
| 57 | I/O10 | 可配置输入输出（C 轴手动正转），配置为输入是 IN54，配置为输出是 OUT26 |
| 58 | I/O11 | 可配置输入输出（C 轴手动反转），配置为输入是 IN55，配置为输出是 OUT27 |
| 59 | I/O12 | 可配置输入输出，配置为输入是 IN56，配置为输出是 OUT28 |
| 60 | I/O13 | 可配置输入输出，配置为输入是 IN57，配置为输出是 OUT29 |
| 61 | I/O14 | 可配置输入输出，配置为输入是 IN58，配置为输出是 OUT30 |
| 62 | I/O15 | 可配置输入输出，配置为输入是 IN59，配置为输出是 OUT31 |

**说明：编码器用作通用输入信号时，XECA+、XECB+、YECA+、YECB+、ZECA+、ZECB+、AECA+、AECB+、BECA+、BECB+、CECA+、CECB+、分别用作对应输入信号的公共端。公共端电压只能用+5V，如果用外部+12V 电源，必须串 1K 电阻。具体接线方法参照下面数字输入连接部分。**

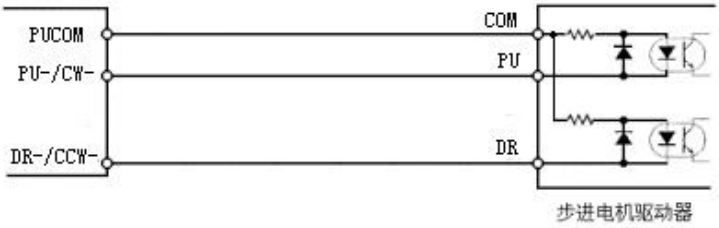
**可配置的 I/O 在使用时，输入输出公共端都需要接上，即：INCOM4，INCOM5 与 12V 或 24V 的电源正端相连，OUTCOM3 和 OUTCOM4 与外部电源负端(地线)相连。**

## 2.4 脉冲/方向输出信号的连接

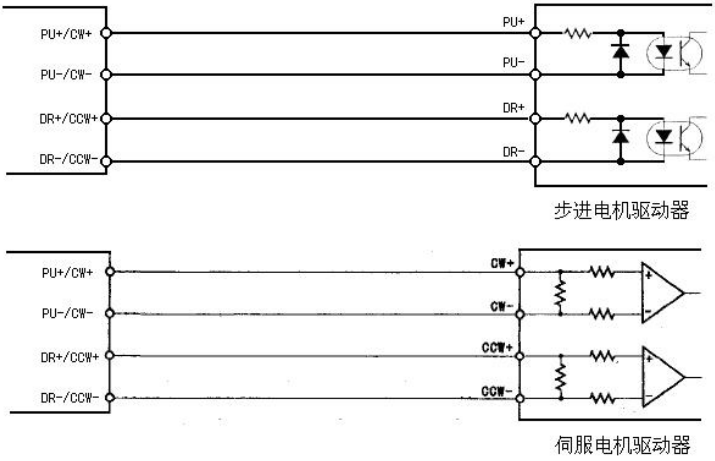
脉冲输出为差动输出方式

可与步进/伺服驱动器很方便的连接

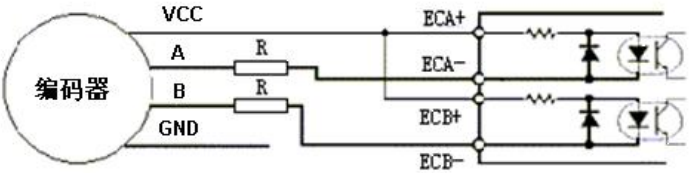
下图为脉冲与方向的阳极已连通的接法




下图为脉冲与方向信号独立的接法，建议采用此种方法，因为是差动接法，抗干扰性强。

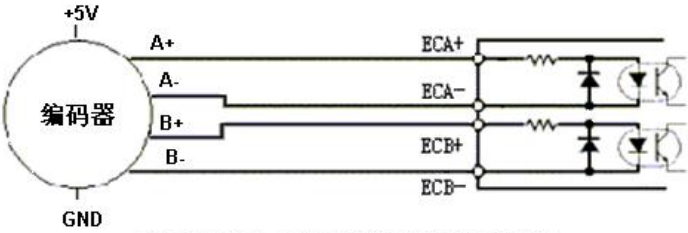


## 2.5 编码器输入信号的连接



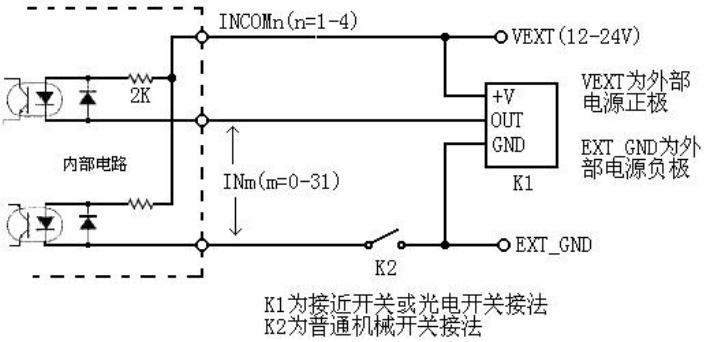

集电极开路(OPEN-COLLECT)输出型编码器接线图

+5V电源时 R可不用 +12V电源时 R=1KΩ +24V电源时 R=2KΩ




差分驱动(LINE DRIVER)输出型编码器接线图

## 2.6 数字输入的连接




说明：

(1) IN0-IN11 的公共端为 INCOM1

IN12-IN23 的公共端为 INCOM2

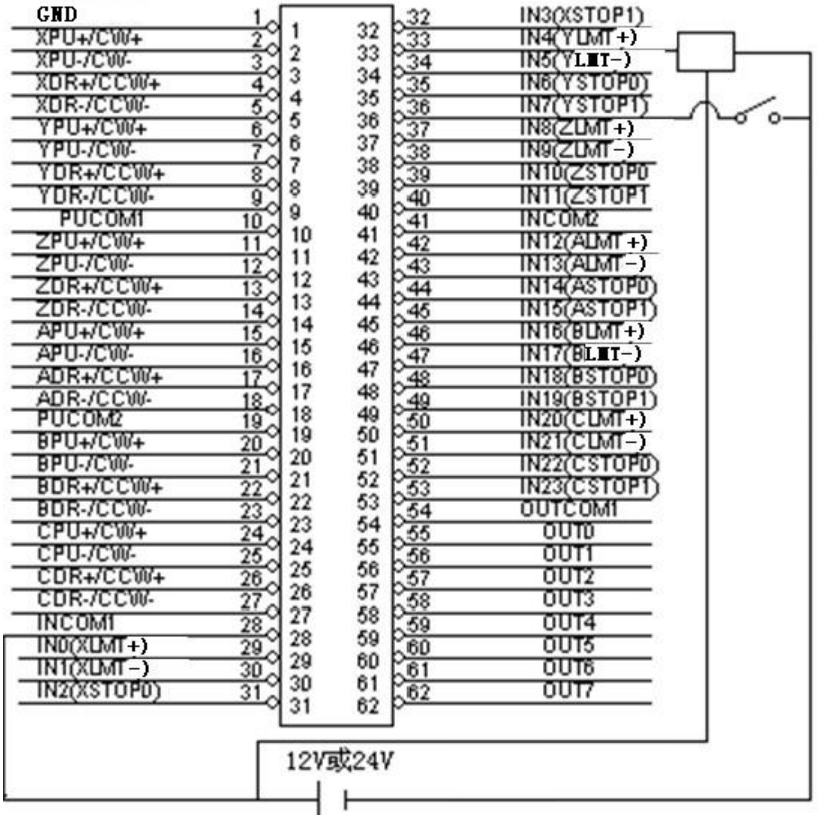
IN24-IN31 的公共端为 INCOM3

I/O0-I/O7 的公共端为 INCOM4

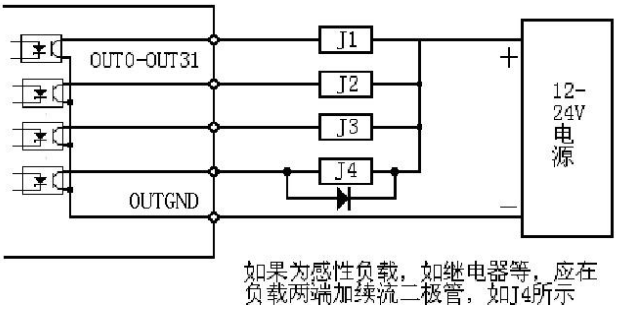
I/O8-I/O15 的公共端为 INCOM5

(2)为了使输入信号有效，首先必须确保对应输入信号的”光耦公共端”(INCOM1 或 INCOM2 或 INCOM3 或 INCOM4 或 INCOM5)已经和 12V 或 24V 的电源正端相连；其次普通开关的一端或接近开关的地线和电源负端(地线)相连；最后普通开关的另一端或接近开关的控制端必须和端子板对应的输入端相连。

(3)下面是普通开关和接近开关在使用外部电源给“光耦公共端”供电的实际接线图（以 J1 端子板为例）。



## 2.7 数字输出的连接

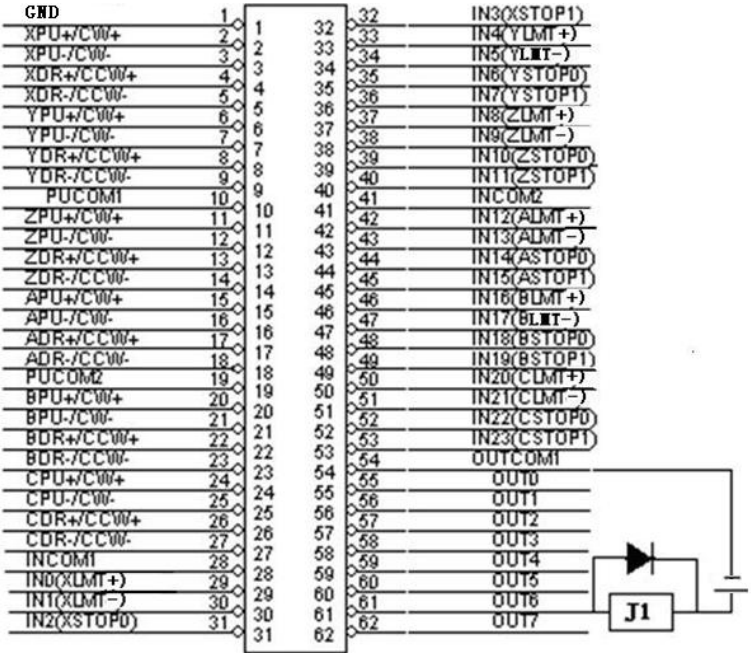



说明：

(1)OUT0-OUT7 的公共端是 OUTCOM1

(2)为了使输出信号有效，在使用外部电源时，必须确保输出公共端 OUTCOM1 和外部电源负端(地线)相连；在使用内部电源时，必须确保内部电源地线(GND)和地相连。继电器线圈的一端接电源正端；另一端接端子板对应的输出端。

（3）下图是继电器采用外部电源供电的实际接线图（以 J1 端子板为例）。



# 第三章 驱动安装

ADT8960 卡在 Win95/Win98/NT/Win2000/WinXP 下必须安装驱动程序才能使用，在 DOS 下则无须安装驱动程序。

以下以 Win98、 WinXP 为例，其余系统可参考。

控制卡驱动程序位于光盘上“开发包\驱动\控制卡驱动程序”文件夹下面，驱动程序文件名为 ADT8960.INF。

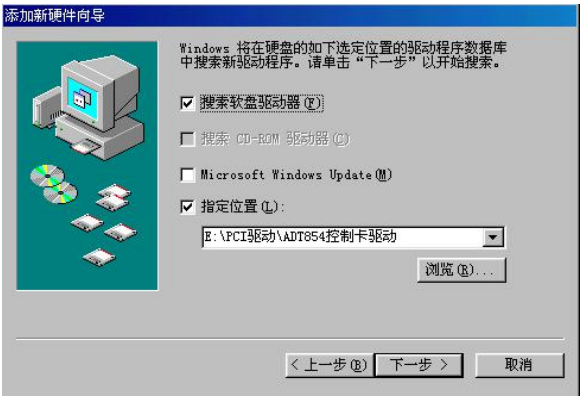
## 3.1 Win98 下驱动程序的安装

以下用Win98 Professional 中文版为例，说明驱动程序的安装，其余版本的 Win98 与此类似。

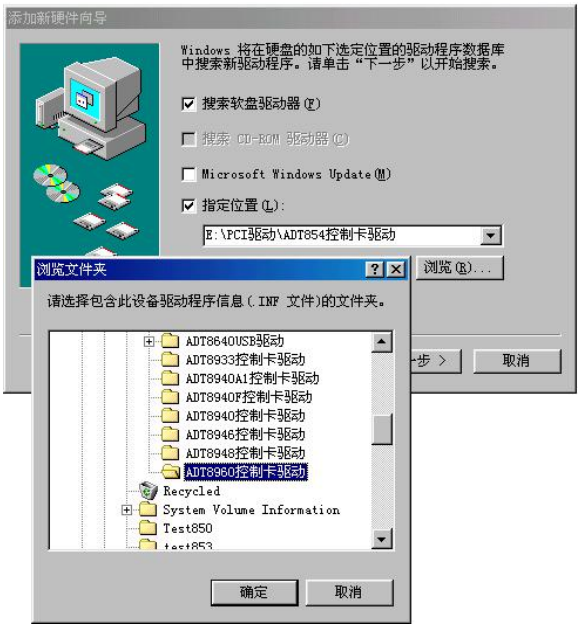
在将ADT8960 卡安装到电脑上的PCI 插槽后，开机时应以管理员身份登录，电脑开机后应发现新硬件，出现如下画面：


单击“下一步”后，再显示如下画面


按上图选择后，再单击“下一步”后，出现如下画面




再按上图选择“指定一个位置”，单击“下一步”，出现如下画面




点击“浏览”按钮，选择光盘“开发包\驱动\控制卡驱动程序”，即可找到 ADT8960.INF 文件的路径，点击“确定”，出现如下界面


单击”下一步”后出现如下画面


单击“完成”后，即完成 ADT8960 卡的安装

## 3.2 WinXP 下驱动程序的安装

WinXP 下的安装与上面类似，参考下图：


按上图选择后,出现如下画面


按上图选择后，再单击“下一步”后，出现如下画面


点击“浏览”按钮，选择光盘“开发包\驱动\控制卡驱动程序”，即可找到 ADT8960.INF 文件的路径，点击“下一步”，出现如下界面


点击“下一步”，出现如下界面

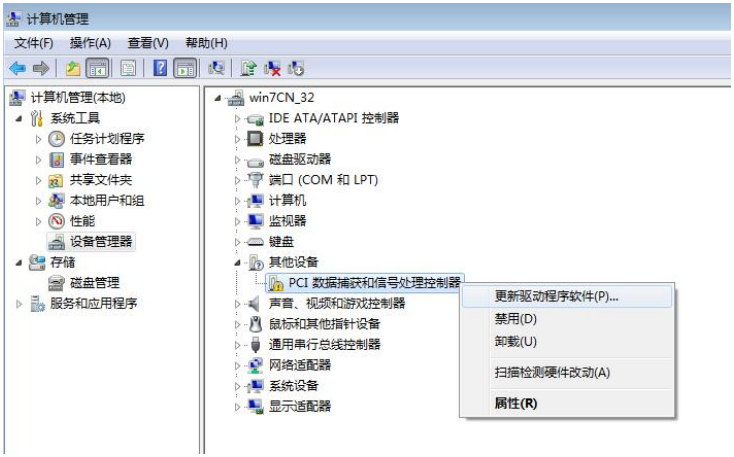

单击“完成”后，即完成 ADT8960 卡的安装

## 3.3 Win7 下驱动程序的安装

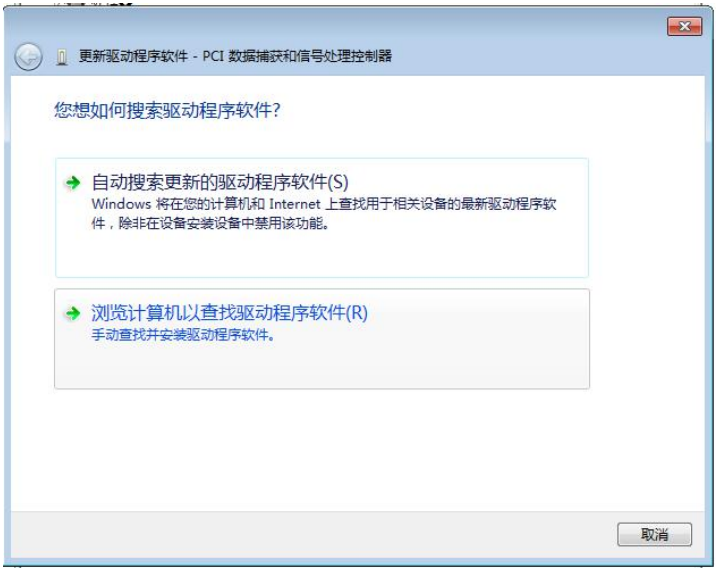
Win7 32 位系统下的安装步骤如下：

1、将控制卡插上 PCI 插槽后，通过“我的电脑”按鼠标右键选择“属性”，进入设备管理器，如图所示：

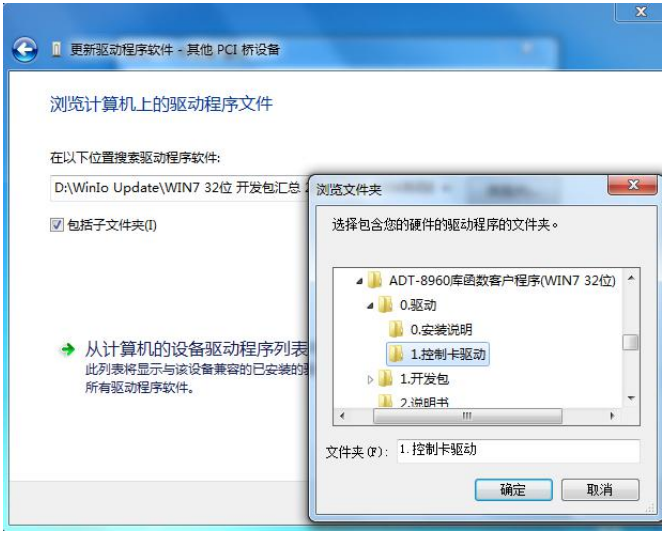

展开“其他设备”，选中“PCI 数据捕获和信号处理器”，按鼠标右键，如下图所示：



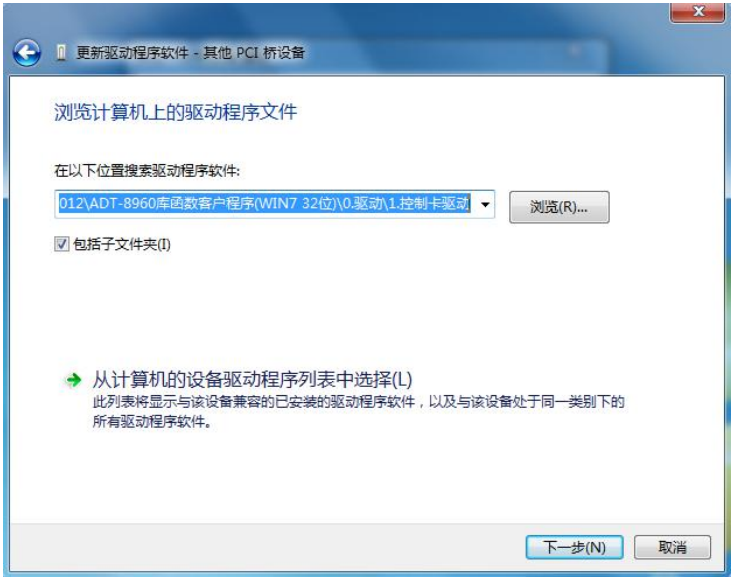

2、在弹出的对话框中，单击 “更新驱动程序软件(P)”，出现如下对话框：



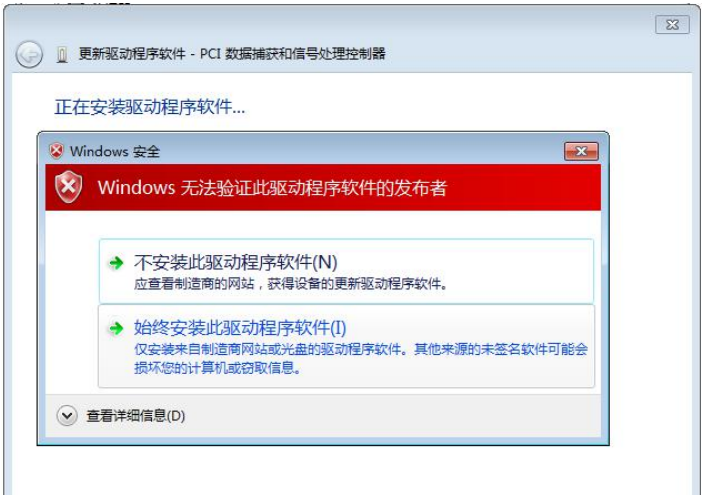

选择“浏览计算机以查找驱动程序软件(R)”选项，然后点击“浏览(R)”按钮，指定搜索的驱动所在的路径，如下图所示：



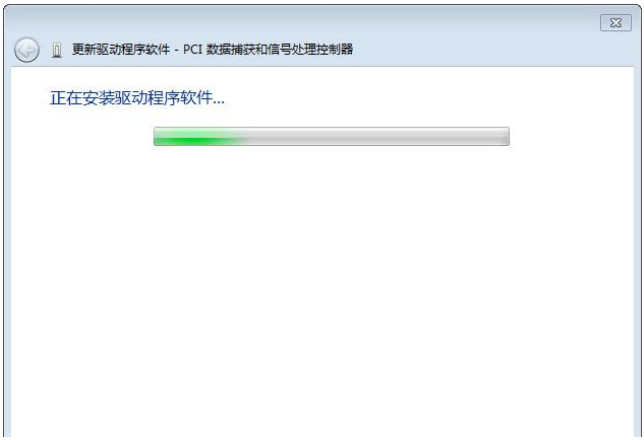
3、单击“确定”后，出现如下对话框：




4、单击“下一步”后，开始安装驱动程序后，出现如下界面：



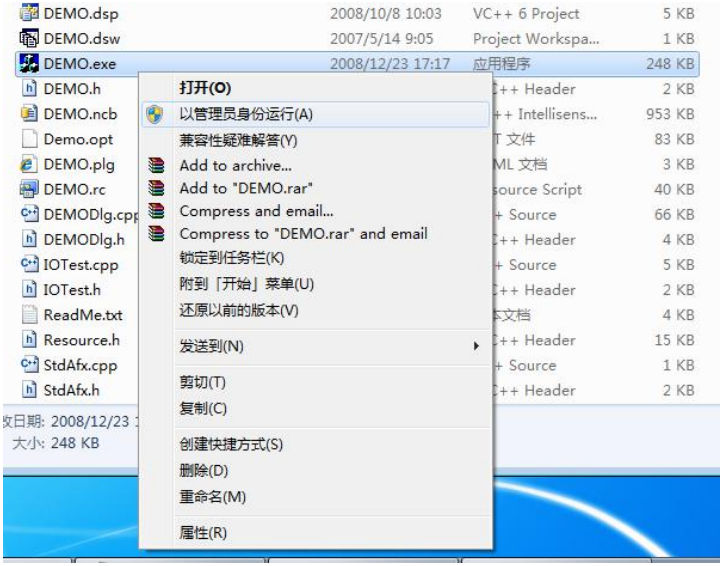
5、选择“始终安装此驱动程序软件(I)”后，出现如下界面




等待完成，出现如下对话框


即完成 ADT-8960 卡的安装。**注意：WIN7 系统需要管理员权限对 PCI 驱动进行加载，如果第一次运行控制卡应用程序直接双击，会导致控制卡初始化失败，所以在第一次安装完成后，必须对控制卡应用程序（比如 VC 示范程序“DEMO.EXE”）按鼠标**

右键，选择 “以管理员身份运行(A)”程序（如下图），之后启动应用程序就只用双击就可以正常运行。




# 第四章 电气规格

## 4.1 开关量输入：

通道：32 路，全部光耦隔离。
输入电压：12-24V
高电平>4.5V
低电平<1.0V
隔离电压：2500V DC

## 4.2 计数输入：

通道：6 路AB 相编码输入，全部光耦隔离。
最高计数频率：2MHz

输入电压：5-24V
高电平>4.5V
低电平<1.0V
隔离电压：2500V DC

**4.3 脉冲输出：**

通道：6 脉冲，6 方向，全部光耦隔离。
最高脉冲频率：2MHz
输出类型：5V 差动输出
输出方式：脉冲+方向 或 脉冲+脉冲

**4.4 开关量输出：**

输出通道：16 路，全部光耦隔离。
输出类型：NPN 集电极开路5-24VDC，最大电流100mA

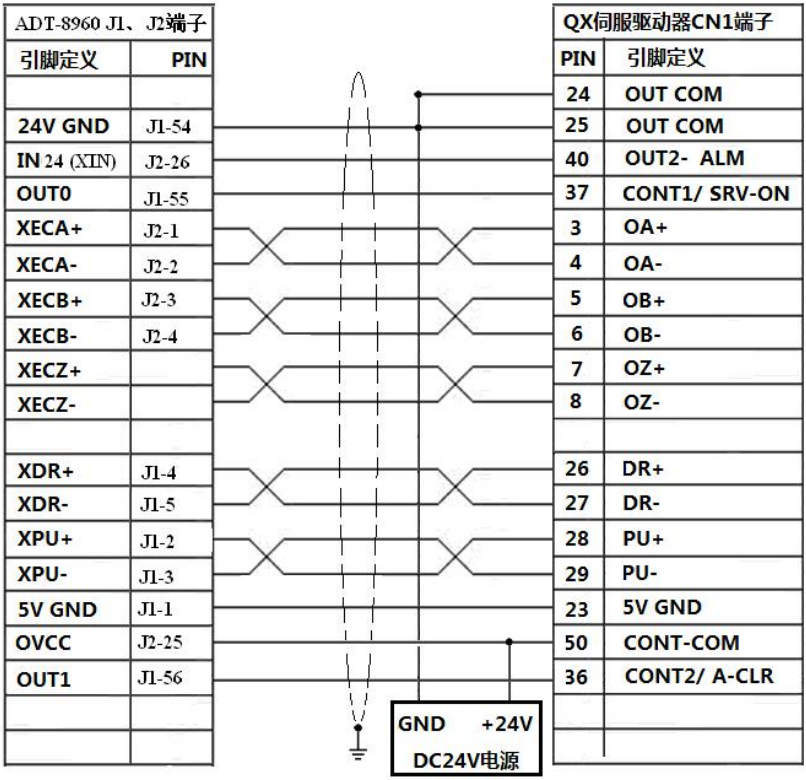
**4.4 电源输出：**

输出电压： +5V。
输出类型：直流源，最大电流500mA。

# 第五章 常见伺服接线图

**5.1 众为兴QX 系列驱动器接线图：**

**下图为控制卡接 QX 伺服驱动器接线：使用外部电源、外部使能，含报警信号。**

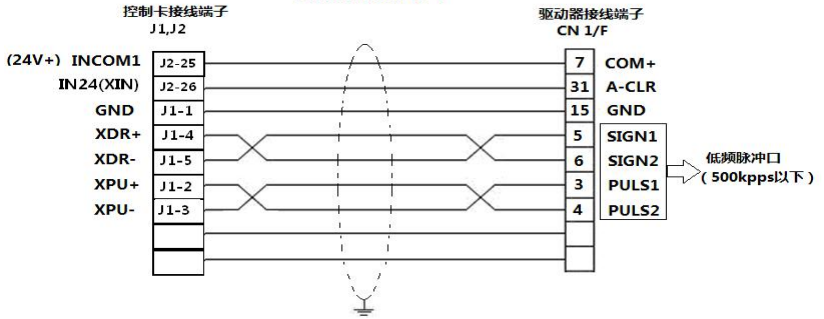



8960 Z相信号如果采用差分接法（Z+，Z-）需要进行电路转换，具体 请参考说明书附录A

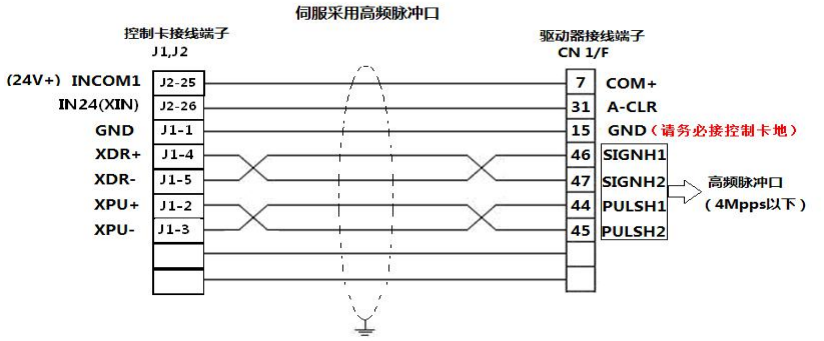
## 5.2 松下A5 系列驱动器接线图：

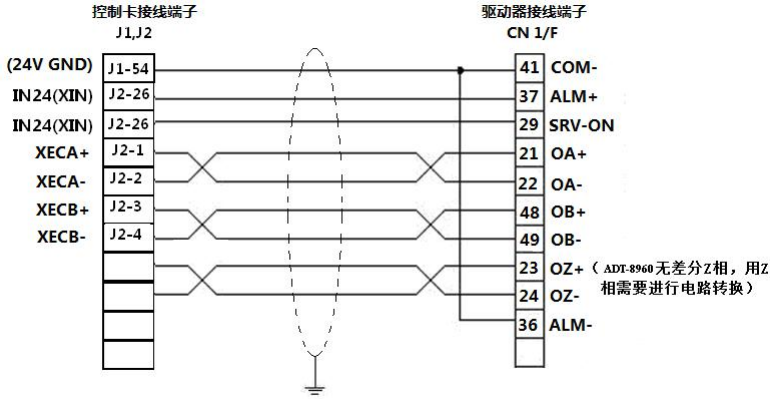
下图为控制卡接松下 A5 伺服驱动器接线：使用外部电源、外部使能，含报警信号。

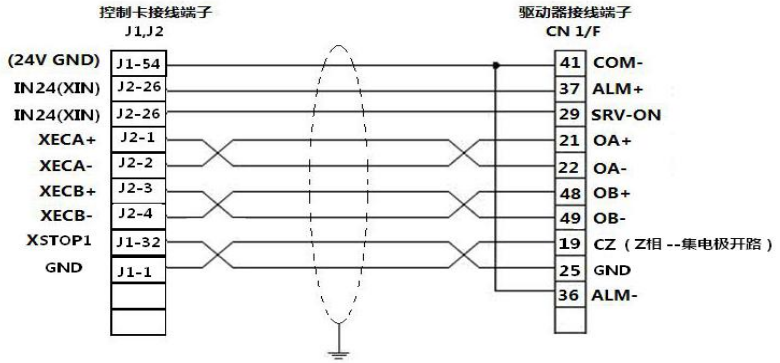
伺服采用低频脉冲口




伺服采用高频脉冲口

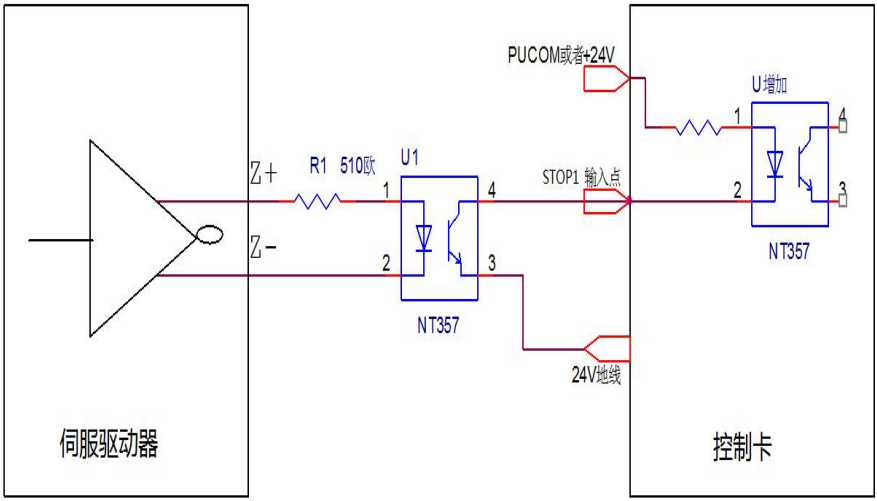





## 5.3 伺服差分Z 接控制卡时电路转换图：

差分信号转集电极开路输出



# 第六章 工作环境

## 6.1 工作温度:

工作温度： 0℃—60℃

## 6.2 储存温度:

储存温度： -20℃—80℃

## 6.3 工作湿度:

工作湿度： 20%—95%

## 6.4 储存湿度:

储存湿度： 0%—95%

# 软件编程篇

# 第一章 功能说明

## 1.1 脉冲输出方式

脉冲输出有独立2脉冲和1脉冲两种方式，采用独立2脉冲方式时，正方向驱动由PU/CW 输出驱动脉冲，负方向驱动由DR/CCW输出驱动脉冲；采用1脉冲方式时，由PU/CW输出驱动脉冲，由DR/CCW输出方向信号。

脉冲/方向都是正逻辑设定时

| 脉冲输出方式 | 驱动方向 | 输出信号波形 | |
|---|---|---|---|
| | | PU/CW 信号 | DR/CCW 信号 |
| 独立2脉冲方式 | +方向驱动输出 | | Low 电平 |
| | -方向驱动输出 | Low 电平 | |
| 1脉冲方式 | +方向驱动输出 | | Low 电平 |
| | -方向驱动输出 | | Hi 电平 |

## 1.2 硬件限制信号

硬件限制信号（LMT+，LMT-）是限定正方向和负方向驱动脉冲输出的输入信号，可设置成有效和无效，以及高低电平，并且正负限位可单独设置有效/无效。设置成无效时可作为普通输入点使用。

硬件限制信号（STOP0，STOP1）是可以实现硬件停止各轴驱动的输入信号，可设置为有效和无效，以及高低电平停止方式，设置为无效时可做一般输入点使用，另外STOP0,STOP1信号在插补驱动时仅仅对最低插补轴有效。

## 1.3 直线插补

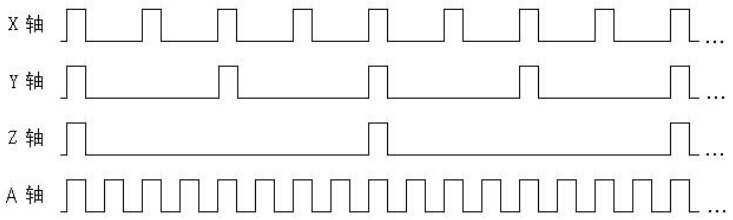
本卡可作任意2-6轴直线插补，采用改进的逐点比较法实现，可保证长轴的脉冲是均匀的，精度在一个脉冲以内。

首先取参与插补的轴中发出脉冲最多的轴，此轴即为长轴，其余轴按比例分配，速度控制只要控制长轴的速度即可。

举例说明如下(1-X轴，2-Y轴，3-Z轴，4-A轴)

做 4 轴直线插补

1 号轴发 1000 个脉冲，2 号轴发 500 个，3 号轴发 250 个，4 号轴发 2000 个。




从上图可知，A 轴为长轴，其余轴按脉冲比例分配输出。

关于插补的速度，虽然是设置参与插补的轴中，轴号小的轴的速度，实际是长轴以这个速度运动，如：在X-Y直线插补时，X移动距离大于Y移动距离时，X为长轴，X轴的驱动速度为设定的速度，X移动距离小于Y移动距离时，Y为长轴，Y轴的驱动速度为设定的速度。

例一:2、3 轴作两轴直线插补，2 轴正向发 10000 个脉冲，3 轴反向发 5000 个脉冲，即 2 号轴为长轴

```
set_startv(0,2,1000);
set_speed(0,2,1000);
inp_move2(0,2,3,10000,-5000);
```

执行如上程序后，2 号轴为长轴，以 1000/2=500Hz 的频率发出 10000 个脉冲，3 号轴的频率应为 500*5000/10000=250Hz。

例二:2、3 轴作两轴直线插补，2 轴正向发 5000 个脉冲，3 轴反向发 10000 个脉冲，即 2 号轴为长轴

```
set_startv(0,2,1000);
set_speed(0,2,1000);
inp_move2(0,2,3,5000,-10000);
```

执行如上程序后，3 号轴为长轴，以 1000/2=500Hz 的频率发出 10000 个脉冲，2 号轴的频率应为 500*5000/10000=250Hz。

上面以匀速为例为方便计算运行速度，插补也可按梯形加减速运动。

## 1.4 运动中变速

运动的过程中可以时时的改变运动速度，可以更灵活有效的控制运动速度和轨迹；速度的改变参考设置速度的相关函数 set_atartv、set_speed、set_acc 等。

## 1.5 外部信号驱动

外部信号驱动是指利用外部信号(手轮或开关)控制的运动,主要用于手动调试机器过程或示教等过程。当开启外部信号驱动使能后，就可以用手轮或机械开关等信号控制脉冲，从而控制运动；当不使用外部信号功能时，需要将外部信号驱动使能关闭。

## 1.6 位置锁存

利用每个轴上的指定的输入信号实现硬件位置锁存功能,利用一个锁存信号可以锁定所有轴的当前位置,锁定的位置可以是逻辑

位置,也可以实际位置.位置锁存的功能在测量系统中有重要的应用。

### 1.7 手动减速

利用每个轴上的指定的输入信号实现手动减速功能，每个轴可单独设定手动减速点，在运动到减速点后自动减速到指定的速度，以指定的速度搜索原点，原点信号需要外部触发。

### 1.8 硬件缓存

大容量硬件缓存功能，在运动过程前将插补数据提前存放在缓存区中，以便提前处理，保证脉冲输出的连续性，使运动过程平滑、连续，可有效提高加工精度；缓存空间高达 2M。

# 第二章 运动控制函数库使用导航

## 2.1 ADT8960 函数库概述

ADT8960 函数库实质是用户操作运动控制卡的接口，用户通过调用接口函数，即可控制运动控制卡完成相应的功能。

运动控制卡提供了 DOS 下的运动函数库和 Windows 下的动态链接库，下面分别介绍 DOS 和 Windows 下的函数库的调用方法。

## 2.2 Windows 下动态链接库的调用

Windows 下的动态链接库“adt8960.dll”利用 VC 编写而成，位于光盘“开发包\驱动\动态链接库”下，适用于 Window 下常用的编程语言工具：VB、VC、C++Builder、VB.NET、VC.NET、Delphi 和组态软件 LabVIEW 等。

### 2.2.1 VC 中的调用

(1) 新建一个项目；

(2) 将光盘“开发包\VC”下的“adt8960.lib”和“adt8960.h”文件拷贝到新建项目的路径下；

(3) 在新建项目”工作区”的”文件视图”中，右击鼠标，选择“Add Files to Project”，在插入文件对话框中，文件类型选择为“Library Files(.lib)”，搜索出“adt8960.lib”并且选择，点击“OK”，完成静态库的加载；

(4) 在源程序文件或头文件或全局头文件“StdAfx.h”的申明部分加上#include “adt8960.h“；

经过上述四步，用户即可调用动态链接库中的函数。

**说明：VC.NET 中的调用方法和 VC 相似。**

### 2.2.2 VB 中的调用

(1) 新建一个项目；

(2) 将光盘“开发包\VB”下的“adt8960.bas”文件拷贝到新建项目的路径下；

(3) 选择“工程\添加模块”菜单命令，选择对话框中的“现存”标签页，搜索出“adt8960.bas”模块文件，点击打开按钮；

经过上述三步，即可在程序中调用动态链接库的函数。

**说明：VB.NET 中的调用方法和 VB 相似。**

### 2.2.3 C++Builder 中的调用

(1) 新建一个项目；

(2) 将光盘中“开发包\C++Builder”中的“adt8960.lib”和“adt8960.h”拷贝到新建项目路径下；

(3) 选择“Project\Add to Project”菜单命令，在对话框中，文件类型选择为“Library files(*.lib)”，搜索出“adt8960.lib”文件，点击“打开”按钮；

(4) 在程序文件的申明部分加上#include “adt8960.h”；

经过上述四步，即可在程序中调用动态链接库。

### 2.2.4 LabVIEW 8 中的调用

(1) 新建一个 VI；

(2) 将光盘中“开发包\驱动\控制卡驱动”中“adt8960.dll”拷贝到新建路径下；

(3) 在需要调用库函数的地方，在程序框图的窗口中，在函数模板中选择“Connectivity\Libraries & Executables”下面的“Call Library Function Node”节点，添加到调用处；

(4) 双击节点，首先在“Call Library Function”对话框中选择“adt8960.dll”动态链接库，其次选择需要的库函数，最后配置好函数的返回值和参数属性；

经过上述四步，即可在程序中调用动态链接库。

## 2.3. DOS 下库函数的调用

DOS 下的函数库是利用 Borland C3.1 编译而成，存放在光盘“开发包\C++或 C”下，库函数分为大模式和巨模式两种，适用于标准 C 和 Borland C3.1 或以上版本。

Borland C 调用函数库的方法如下：

(1) 在 Borland C 的开发环境下，选择“Project\Open Project”命令新建一个项目；

(2) 将光盘中“开发包\C 或 C++”下面的“adt8960H.LIB”或“adt8960L.LIB”和“adt8960.H”文件拷贝到新建项目路径下；

(3) 选择“Project\Add Item”命令，在对话框中选择“adt8960H.LIB”或“adt8960L.LIB”，单击“Add”按钮；

(4) 在用户程序文件中增加#include “adt8960.h”申明；

经过上述四步，即可在程序中调用库函数。

## 2.4. 库函数返回值及其含义

为了保证用户在使用库函数时，正确掌控库函数的执行情况，函数库中的每个库函数都会在执行结束后，返回库函数的执行结果。用户依据返回值，可以很方便地判断出函数调用是否成功。

函数库中除“int adt8960_initial(void)”和“int read_bit(int cardno, int number)”的返回值特殊外，其他函数的返回值只有“0”和“1”两种情况，其中“0”表示调用正确，“1”表示调用失败。

下面以列表的形式介绍函数返回值的含义。

| 函数名 | 返回值 | 含义 |
|---|---|---|

| | | |
|---|---|---|
| adt8960_initial | -1 | 未安装相关服务 |
| | -2 | PCI 插槽故障 |
| | 0 | 没有安装控制卡 |
| | >0 | 代表控制卡的数量 |
| Read_bit | 0 | 低电平 |
| | 1 | 高电平 |
| | -1 | 代表卡号或输入点超限错误 |
| 其他所有函数 | 0 | 正确 |
| | 1 | 错误 |

**说明：返回值1错误，正常是由于调用库函数的过程中，传递的参数值cardno(卡号)或axis(轴号)错误引起的。卡号的值从0、1、2依次向上编号，所以在只用一块卡的情况下，卡号必须为0；轴号的值只能是1、2、3、4、5、6，其他的值都是错误的。**

# 第三章 运动控制开发要点

本卡在编程时常会遇到一些问题，其实，大部分问题是由于对本控制卡的原理不理解而产生的，以下就一些常见的、易产生误解的情况作一些说明。

## 3.1 卡的初始化

在程序的开始首先应调用adt8960_initial()函数，确认ADT8960卡的安装是否正确，然后设置脉冲输出的模式，限位开关的工作模式，以上参数应根据具体的机器来设置，一般只应在程序初始化时设置一次，以后不应再设置。

通常应根据可能使用的最大使用频率确定R值，除非最低频率不能满足使用要求，否则在使用过程中不应改变 R 值，

**说明：库函数“adt8960_initial”是通往ADT8960卡的“门户”，只有在调用该函数对运动控制卡初始化成功后，再调用其他函数才有意义。**

## 3.2 速度的设定

### 3.2.1 匀速运动

参数的设置很简单，只需要将驱动速度设置成等于起始速度，其余的参数不用设置。

相关函数：

set_startv

set_speed

### 3.2.2 S 曲线加减速

对于一些负载较重的方式，为了达到较好的加速效果，采用S曲线加速，此时加加速度的值需设置，加加速度的计算对S曲线的形状有很大的影响，可参考前面的例子。

相关函数：

set_startv

set_speed

set_acc

set_acac

set_ad_mode （设为S曲线加减速方式）

### 3.2.3 插补速度

ADT8960卡可以任意2轴作直线插补，任意3轴作直线插补，任意4轴直线插补，任意5轴直线插补以及6轴直线插补。

关于插补的速度，是使用最前面的一个轴的速度参数，作为长轴的速度，例如

inp_move2 (0,3,1,100,200)

是采用第一个轴的速度参数，即X轴，而与参数中的顺序无关。

inp_move3 (0,3,4,2,100,200,500)

是采用第二个轴的速度参数，即Y轴，而与参数中的顺序无关。

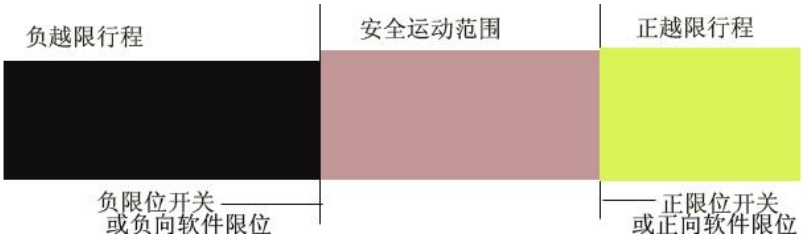
**说明：插补时的速度倍率是单轴运动时倍率的一半，即在同样的参数时，插补的速度只有单轴运动的一半。**

# 第四章 系统安全机制

## 4.1 错误信息监测：

使用get_stopdata()函数获取错误信息可获取轴的停止信息，包括由硬件限位引起的停止，原点信号引起的停止，正常停止，以及其他停止。

## 4.2 限位：




运动控制卡可使用限位开关或者软件限位控制轴的运动范围，负向限位开关或者负向软件限位触发后，只能朝正向运动，而不能再朝负向运动；正向限位开关或者正向软件限位触发后，只能朝负向运动，而不能再朝正向运动；注意，软件限位只能在回零成功后方可使用。

# 第五章 回零

## 5.1 回零运动：

### 5.1.1 所需函数列表：

| | |
|---|---|
| 单轴回原点 | home1 |
| 注意事项 | 见重点备注 |

以单轴回原点进行举例，多轴亦可用。

采用STOP0为原点信号

(1) 回原点分为四大步:

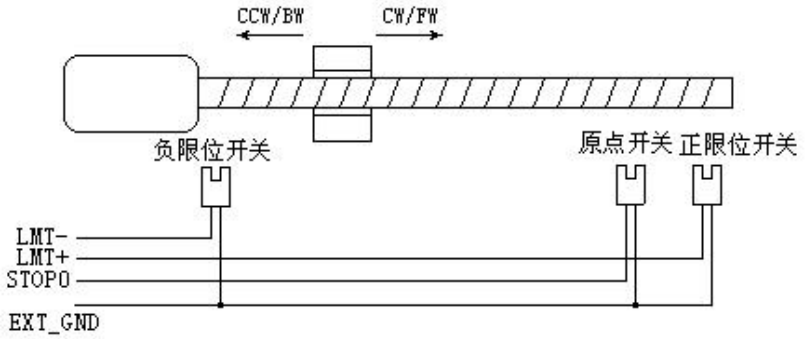
第一步:快速接近stop0(logical0原点设置)，找到stop0;

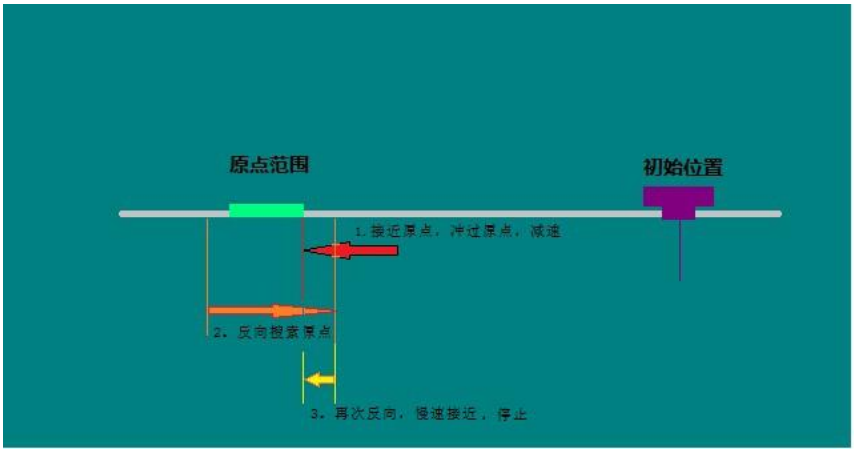
第二步:慢速反向离开stop0，反向移动指定原点范围脉冲数;

第三步:再次慢速接近stop0;

第四步:慢速接近stop1(logical1编码器Z相).

(2) 第四步可以选择是否执行,通过logical1来选择.



### 5.1.2 例程

```
void main()
{
    int retnX = -1;
    retnX = SetHomeMode_Ex(0, 1, 0, 0, 0, -1, 2000, 400, 100); //'设置回零模式
    retnX = SetHomeSpeed_Ex(0, 1, 100, 500, 200, 100, 200);    //'设置回零速度
    retnX = HomeProcess_Ex(0, 1) ;//'启动回零
    if (retnX < 0   || retnX > 20)
    {   // MessageBox ("回零失败");
        Return;
    }
    while(true)
    {
        //DoEvent();
        retnX = GetHomeStatus_Ex(0, 1);//查询回零状态
        // HomeStatus.Caption = "回原点状态：" + CStr(retnX)
        if (retnX < 0   || retnX > 20)
        { //MessageBox("回零失败");
          break;
        }

        if( retnX == 0 )
        {
            //MessageBox ("成功");
            break;
        }
    }
```

}

### 5.1.3 重点备注

回零注意参数设置，原点(STOP0)搜寻起始速度不能大于原点搜寻速度，如不需搜寻STOP1 信号，把此参数设为-1，需要时，请按照函数说明进行设置即可,上例是以X 轴为例，如需多轴回零，只需再增加函数调用即可，注意，此处举例并未搜寻Z 相信号，即STOP1 信号，如需，可进行设置即可调用。

### 5.1.3 重点备注

回零注意参数设置，原点(STOP0)搜寻起始速度不能大于原点搜寻速度，如不需搜寻STOP1 信号，把此参数设为-1，需要时，请按照函数说明进行设置即可,上例是以X 轴为例，如需多轴回零，只需再增加函数调用即可，注意，此处举例并未搜寻Z 相信号，即STOP1 信号，如需，可进行设置即可调用。

# 第六章 联动控制

联动控制包括单轴点位运动、多轴点位运动，在项目中具体实施，根据需要。

本章所有例程都相互独立，在实施项目的过程中卡的初始化只需设一次即可，即在程序系统初始化的过程中卡的初始化设置后，就无需设置.

## 6.1 单轴定量匀速运动：

### 6.1.1 所需函数列表

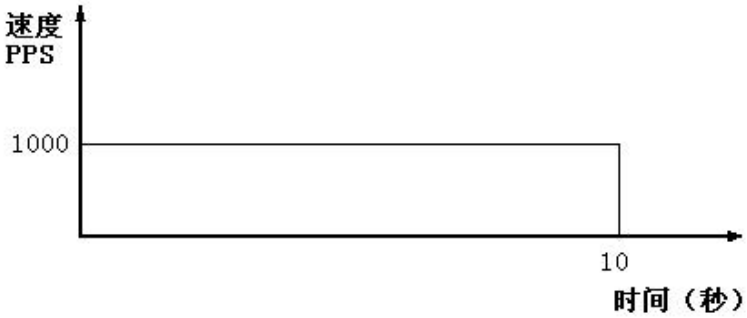
| | |
|---|---|
| 设置初始速度 | set_startv |
| 设置驱动速度 | set_speed |
| 驱动指令 | pmove |

| | |
|---|---|
| 读驱动状态 | get_status |
| 注意事项 | 见重点备注 |

### 6.1.2 例程

目的：

让X轴的步进电机以1000 pps的速度运动10000步：




程序如下：

```
#include “adt8960.h”
void main()
{
    int cardno;
    cardno= adt8960_initial();
    if(cardno<=0) return;              //未安装ADT8960卡
                                       //以下只对第一块卡X轴操作
也可对多个轴进行操作
                                       //如果有多块卡，即cardno>1
                                       //可修改卡号，操作其他卡
    set_pulse_mode(0,1,1,0,0);         //设置X轴为脉冲+方向
方式
    set_startv(0,1,1000);
    set_speed(0,1,1000);               //如果起始速度大于或等
```

```
于驱//动速度，则为匀速运动
    pmove(0,1,10000);                    //开始驱动
    int s;
    while(1)
    {
        get_status(0,1,&s);        //读驱动状态
        if(s==0)break;                   //驱动结束跳出
        ……                               //可执行读键盘，显示位
置等函数
    }
    return ;
    }
```

### 6.1.3 重点备注

本例程中涉及控制卡初始化函数，设置脉冲模式函数之前如果设置，就不需要再次设置，只需在程序初始化中设置一次即可。

## 6.2 单轴定量对称梯形加/减速运动：

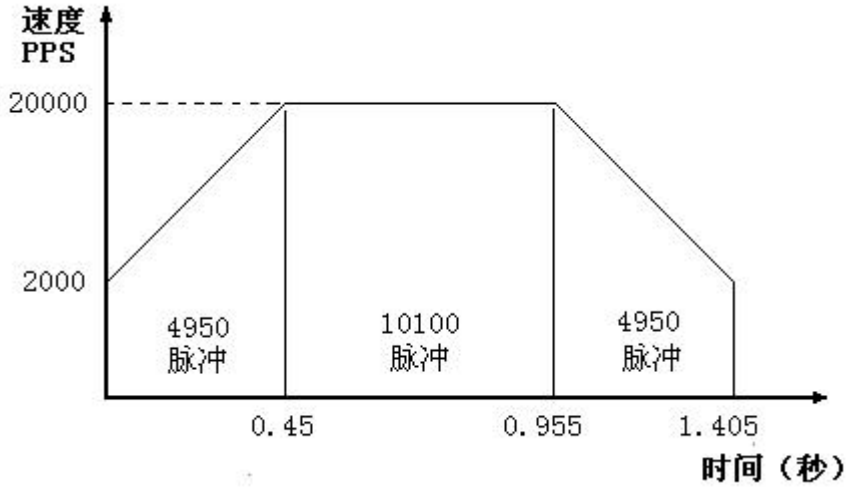
### 6.2.1 所需函数列表：

| | |
|---|---|
| 设置为梯形加减速 | set_ad_mode |
| 设置初始速度 | set_startv |
| 设置驱动速度 | set_speed |
| 设置加/减速度 | set_acc |
| 驱动指令 | pmove |
| 读驱动状态 | get_status |
| 注意事项 | 见重点备注 |

### 6.2.2 例程：

目的：

让X轴以下列速度运动20000步
起始速度：2000 pss
驱动速度：20000 pss
加/减速度：40000 pss




加速时间应为（20000-2000）/40000=0.45 秒
加速脉冲应为0.45*（20000+2000）/2=4950 个
减速与加速相同。

```
#include “adt8960.h”
void main()
{
    int cardno;
    cardno=adt8960_initial();
    if(cardno<=0) return;              //未安装ADT8960卡
                                       //以下只对第一块卡X轴操作
                                       //如果有多块卡，即cardno>1
                                       //可修改卡号，操作其他卡
```

```
        set_pulse_mode(0,1,1,0,0);      //设置X轴为脉冲+方向
    方式
        set_ad_mode(0,1,0);             //设置为梯形加减速
        set_startv(0,1,2000);           //起始速度   2000
        set_speed(0,1,20000);            //驱动速度   20000
        set_acc(0,1,320);               //加/减速度
40000/125=320
        pmove(0,1,20000);               //开始驱动
        int s;
        while(1)
        {
            get_status(0,1,&s);     //读驱动状态
            if(s==0)break;              //驱动结束跳出
            ……                          //可执行读键盘，显示位
    置等函数
        }
        return ;
    }
```

### 6.2.2 重点备注：

本例程中涉及控制卡初始化函数，设置脉冲模式函数之前如果设置，就不需要再次设置，只需在程序初始化中设置一次即可。

## 6.3 多轴运动

### 6.3.1 所需函数列表：

| | |
|---|---|
| 设置为梯形加减速 | set_ad_mode |
| 设置初始速度 | set_startv |
| 设置驱动速度 | set_speed |

| | |
|---|---|
| 设置加速度 | set_acc |
| 设置加加速度 | set_acac |
| 驱动指令 | pmove |
| 连续运动指令 | continue_move |
| 读驱动状态 | get_status |
| 注意事项 | 见重点备注 |

以上虽为单轴操作，但实际上可同时设置另外几轴的数据，互相之间并不影响，如在X轴驱动时，设置好Y轴的参数，然后驱动Y轴，对X轴的运动不会有任何影响，如此可独立操作四轴，

下面是一个简单的例子，X轴以匀速（1000pps）运动1000步，Y轴以直线加减速运动300000步（起始速度10000pps，驱动速度200000pps，加减时间 0.2秒），Z轴以完全S曲线加速连续运动（起始速度 100 pps，驱动速度4000 pss，加速时间1.2秒），W轴以匀速（300000 pps）连续运行，按‘s’键停止。

首先计算Y轴的加速度，为（200000-10000）/0.2=950000

然后计算Z轴的加速度及加速度的变化率，加速度计算方法为：首先在0.6秒内加速至2000 pps，然后0.6秒内再加至4000 pps，则加速度为2000*2/0.6=6667 pps/sec，加速度的变化率为6667/0.6=11111 pps/sec/sec。

### 6.3.2 例程：

程序如下：

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>
#include “ADT-8960.h”
```

```
void main()
{
    int cardno;
    cardno=ADT-8960_initial();
    if(cardno<=0) return;              //未安装ADT-854卡
                                       //以下只对第一块卡X轴操作
                                       //如果有多块卡，即cardno>1
                                       //可修改卡号，操作其他卡
    set_pulse_mode(0,1,1,0,0);         //设置X轴为脉冲+方向
方式
    set_pulse_mode(0,2,1,0,0);         //设置Y轴为脉冲+方向
方式
    set_pulse_mode(0,3,1,0,0);         //设置Z轴为脉冲+方向
方式
    set_pulse_mode(0,4,1,0,0);         //设置W轴为脉冲+方向
方式

    //Y轴加减速设置
    set_ad_mode(0,2,0);                //设置为梯形加减速
    //Z轴加减速设置
    set_ad_mode(0,3,1);                //设置为梯形加减速
    set_dec1_mode(0,3,0);              //对称加减速
    set_dec2_mode(0,3,0);              //采用自动减速

    //X轴
    set_startv(0,1,1000);              //起始速度   1000
    set_speed(0,1,1000);               //驱动速度   1000

    //Y轴
    set_startv(0,2,10000);             //起始速度  10000
```

```
set_speed(0,2,200000);              //驱动速度   200000
set_acc(0,2,7600);                  //加速度
                                        950000/125=7600

//Z轴
set_startv(0,3,100);
set_speed(0,3,4000);
set_acc(0,3,53);                    // 6667/125=53.3
set_acac(0,3,5625);                 // 62500000/11111=5625

//W轴
set_startv(0,4,300000);             // 300000
set_speed(0,4,300000);

pmove(0,1,1000);                //开始驱动
pmove(0,2,300000);
continue_move(0,3,0);
continue_move(0,4,0);

int s1,s2,s3,s4;
while(1)
{
    get_status(0,1,&s1);            //读X驱动状态
    get_status(0,2,&s2);            //读Y驱动状态
    get_status(0,3,&s3);            //读Z驱动状态
    get_status(0,4,&s4);            //读W驱动状态

    if(s1==0 && s2==0 && s3==0 && s4==0)break;
                                    //驱动结束跳出
    if(kbhit())
        key=getch();
```

```
            else
                key=-1;
            if(key==’s’)
            {
                dec_stop(0,3);
                dec_stop(0,4);
            }
        }
        return ;
    }
```

**6.3.3 重点备注：**

本例程中涉及控制卡初始化函数，脉冲模式函数之前如果设置，就不需要再次设置，只需在程序初始化中设置一次即可。

# 第七章 插补运动控制

本章所有例程都相互独立，在实施项目的过程中卡的初始化和倍率的设置只需设一次即可，即在程序系统初始化的过程中卡的初始化设置后，就无需设置。

## 7.1 两轴直线插补(匀速)

### 7.1.1 所需函数列表：

| | |
|---|---|
| 设置初始速度 | set_startv |
| 设置驱动速度 | set_speed |
| 驱动指令 | inp_move2 |
| 读插补状态 | get_inp_status |
| 注意事项 | 见重点备注 |

插补速度是以第一轴速度为基准，开始插补前只要设好第一轴的参数即可。下面是一个匀速直线插补的简单例子，圆弧插补与多轴直线插补的匀速驱动基本相同。

### 7.1.2 例程：

程序如下：

```
#include “adt8960.h”
void main()
{
    int cardno;
    cardno=adt8960_initial();
    if(cardno<=0) return;          //未安装ADT854卡
                                   //以下只对第一块卡X轴操作
                                   //如果有多块卡，即cardno>1
                                   //可修改卡号，操作其他卡

    set_pulse_mode(0,1,1,0,0); //设置X轴为脉冲+方向方式
    set_pulse_mode(0,2,1,0,0); //设置Y轴为脉冲+方向方式
    set_startv(0,1,1000);              //X起始速度   1000
    set_speed(0,1,1000);               //X驱动速度   1000

    inp_move2(0,1,2,10000,-20000);//X-Y开始插补
                                   //X正向移动10000步
                                   //Y反向移动20000步
    int s1;
    while(1)
    {
        get_inp_status(0,&s1); //读插补状态
        if(s1==0)break;    //插补结束跳出
```

```
            ……                    //可执行读键盘，显示位置等函数
    }
        return ;
    }
```

### 7.1.3 重点备注：

本例程中涉及控制卡初始化函数，设置脉冲模式函数之前如果设置，就不需要再次设置，只需在程序初始化中设置一次即可。

## 7.2 两轴直线插补(加减速)

### 7.2.1 所需函数列表：

| | |
|---|---|
| 设置为梯形加减速 | set_ad_mode |
| 设置初始速度 | set_startv |
| 设置驱动速度 | set_speed |
| 设置加速度 | set_acc |
| 插补指令 | inp_move2 |
| 读插补状态 | get_inp_status |
| 注意事项 | 见重点备注 |

两轴直线插补的加/减速驱动，只要将第一轴设置成直线加减速或S曲线加减速即可，注意，驱动开始之前须设成减速有效状态。

将上面的例子改成加减速驱动，X-Y直线加减速。

### 7.2.2 例程：

程序如下：

```
#include “adt8960.h”
```

```
void main()
{
    int cardno;
    cardno=adt8960_initial();
    if(cardno<=0) return;              //未安装ADT8960卡
                                       //
                                       //如果有多块卡，即cardno>1
                                       //可修改卡号，操作其他卡

    set_pulse_mode(0,1,1,0,0); //设置X轴为脉冲+方向方式
    set_pulse_mode(0,2,1,0,0); //设置Y轴为脉冲+方向方式
    set_ad_mode(0,1,0);
    set_startv(0,1,1000);              //X起始速度   1000
    set_speed(0,1,8000);               //X驱动速度   8000
    set_acc(0,1,1000);
    inp_move2(0,1,2,10000,-20000);//X-Y开始插补
                                       //X正向移动10000步
                                       //Y反向移动20000步
    int s1;
    while(1)
    {
        get_inp_status(0,&s1); //读插补状态
        if(s1==0)break;    //插补结束跳出
        ……                             //可执行读键盘，显示位
置等函数
    }
    return ;
}
```

### 7.2.3 重点备注：

本例程中涉及控制卡初始化函数，设置脉冲模式函数之前如果设置，就不需要再次设置，只需在程序初始化中设置一次即可。

## 7.3 三轴直线插补(加减速)

### 7.3.1 所需函数列表：

| | |
|---|---|
| 设置为梯形加减速 | set_ad_mode |
| 设置初始速度 | set_startv |
| 设置驱动速度 | set_speed |
| 设置加速度 | set_acc |
| 插补指令 | inp_move3 |
| 读插补状态 | get_inp_status |
| 注意事项 | 见重点备注 |

三轴直线插补可为任意三轴，以第一轴的速度为基准。

下面是一个直线加减速的简单的例子，其余加减速方式只须改一下设置即可。

### 7.3.2 例程：

```
#include “adt8960.h”
void main()
{
    int cardno;
    cardno=adt8960_initial();
```

```
    if(cardno<=0) return;              //未安装ADT8960卡
                                       //以下只对第一块卡X轴操作
                                       //如果有多块卡，即cardno>1
                                       //可修改卡号，操作其他卡

    set_pulse_mode(0,1,1,0,0);         //设置X轴为脉冲+方向
方式
    set_pulse_mode(0,2,1,0,0);         //设置Y轴为脉冲+方向
方式
    set_pulse_mode(0,3,1,0,0);         //设置Z轴为脉冲+方向
方式
    set_ad_mode(0,1,0);
    set_startv(0,1,1000);              //X起始速度   1000
    set_speed(0,1,8000);               //X驱动速度   8000
    set_acc(0,1,1000);

    inp_move3(0,1,2,3,5000,10000,-20000);  //X-Y-Z开始插
补
                                       //X正向移动5000步
                                       //Y正向移动10000步
                                       //Z反向移动20000步
    int s1;
    while(1)
    {
        get_inp_status(0,&s1);         //读插补状态
        if(s1==0 )break;                   //插补结束跳出
        ……                             //可执行读键盘，显示
位置等函数
    }
    return ;
}
```

### 7.3.3 重点备注：

本例程中涉及控制卡初始化函数，设置脉冲模式函数之前如果设置，就不需要再次设置，只需在程序初始化中设置一次即可。

# 第八章 轨迹运动控制

本章所有例程都相互独立，在实施项目的过程中卡的初始化设置只需设一次即可，即在程序系统初始化的过程中卡的初始化设置后，就无需设置.此处仅举例说明，实现圆弧可用缓存软件圆弧插补。

## 8.1 缓存插补

### 8.1.1 所需函数列表：

| | |
|---|---|
| 清除缓存 | reset_fifo |
| 读取缓存是否满 | read_fifo_full |
| 缓存插补 | fifo_inp_move2 |
| 注意事项 | 见重点备注 |

### 8.1.2 例程：

程序如下：

```
#include “adt8960.h”
void main()
{
    int cardno;
    int s1,s2;
    cardno=adt8960_initial();
```

```
    if(cardno<=0) return;                    //未安装adt8960卡
                                             //以下只对第一块卡X轴操作
                                             //如果有多块卡，即cardno>1
                                             //可修改卡号，操作其他卡

    set_pulse_mode(0,1,1,0,0);               //设置X轴为脉冲+方向
方式
    set_pulse_mode(0,2,1,0,0);               //设置Y轴为脉冲+方向
方式
     int m=reset_fifo(0);//清除缓存
    for (int i=0; i<50; )
    {
        //加工50段数据,每段为100个脉冲
         if (read_fifo_full(0)==0)//0:未满，1：满
         {
             //读取缓存是否满
              m=fifo_inp_move2(0,1,2,100,100,3000);
              i++;
         }
         else
         {
              //MessageBox("缓存满，稍后存放");
              break;
         }
    }


    return ;
}
```

### 8.1.3 重点备注：

本例程中涉及控制卡初始化函数，设置脉冲模式函数之前如果设置，就不需要再次设置，只需在程序初始化中设置一次即可。

# 第九章 通用数字量 I/O

运动控制卡为用户提供了一个通用的数字量输入/输出口，主机可通过指令的方式对输入/输出口进行操作。

### 9.1 输入口定义:具体端口定义见硬件篇第二章

### 9.2 输出口定义:具体端口定义见硬件篇第二章

### 9.3 输出口:

#### 9.3.1 所需函数列表：

| | |
|---|---|
| 输出函数 | write_bit |
| 注意事项 | 见重点备注 |

#### 9.3.2 例程：

程序如下：以1号端口为例，打开输出

```
#include “adt8960.h”
void main()
{
    Const int cardno = 0;
    write_bit(cardno,1,1);
    return ;
    }
```

### 9.3.3 重点备注：

输出端口根据需要定义端口号。

## 9.4 输入口：

### 9.4.1 所需函数列表：

| | |
|---|---|
| 读取输入点函数 | read_bit |
| 注意事项 | 见重点备注 |

### 9.4.2 例程：

程序如下：以1号端口为例，读取1号端口的电平，低电平有效

```
#include “adt8960.h”
void main()
{
    Const int cardno = 0;
      int value = -1；
    value    = read_bit(cardno,1);
      if(value == 0)
      {
          //执行其他相应操作
       }
    return ;
}
```

### 9.4.3 重点备注：

此函数根据端口的返回值调用。

# 第十章 复合运动控制

特别说明：复合运动的功能都可以上面都可用联动控制和插补控制章节中提到的函数完成，只是函数参数不一样，客户可根据自己需要选择。

本章所有例程都相互独立，在实施项目的过程中卡的初始化设置只需设一次即可，即在程序系统初始化的过程中卡的初始化设置后，就无需设置

## 10.1 单轴对称梯形相对运动：

### 10.1.1 所需函数列表：

| | |
|---|---|
| 驱动指令 | symmetry_relative_move |
| 读驱动状态 | get_status |
| 注意事项 | 见重点备注 |

### 10.1.2 例程：

目的：

让X轴以下列速度运动30000步

起始速度：2000 pss

驱动速度：20000 pss

```
#include “adt8960.h”
void main()
{
    int cardno;
    cardno=adt8960_initial();
    if(cardno<=0) return;                    //未安装ADT8960卡
                                             //以下只对第一块卡X轴操作
```

```
                                        //如果有多块卡，即cardno>1
                                        //可修改卡号，操作其他卡
    set_pulse_mode(0,1,1,0,0);          //设置X轴为脉冲+方向
方式

     symmetry_relative_move (0,1, 20000, 400, 4000, 0.1, 300,
0);
    int s;
    while(1)
    {
        get_status(0,1,&s);      //读驱动状态
        if(s==0)break;                  //驱动结束跳出
        ……                              //可执行读键盘，显示位
置等函数
    }
    return ;
}
```

**10.1.3 重点备注：**

复合运动控制函数根据需要使用。
本例程中涉及控制卡初始化函数，设置脉冲模式函数之前如果设置，就不需要再次设置，只需在程序初始化中设置一次即可。

**10.2 单轴对称梯形绝对运动：**

**10.2.1 所需函数列表：**

| | |
|---|---|
| 驱动指令 | symmetry_absolute_move |
| 读驱动状态 | get_status |

| 注意事项 | 见重点备注 |
| --- | --- |

### 10.2.2 例程：

目的：

让X轴以下列速度运动30000步

起始速度：2000 pss

驱动速度：20000 pss

```
#include “adt8960.h”
void main()
{
    int cardno;
    cardno=adt8960_initial();
    if(cardno<=0) return;              //未安装ADT8960卡
                                       //以下只对第一块卡X轴操作
                                       //如果有多块卡，即cardno>1
                                       //可修改卡号，操作其他卡
    set_pulse_mode(0,1,1,0,0);         //设置X轴为脉冲+方向
方式
      symmetry_absolute_move(0,1, 20000, 400, 4000, 0.1,
300, 0);
    int s;
    while(1)
    {
        get_status(0,1,&s);        //读驱动状态
        if(s==0)break;                 //驱动结束跳出
        ……                             //可执行读键盘，显示位
置等函数
    }
    return ;
```

```
}
```

### 10.2.3 重点备注：

复合运动控制函数根据需要使用。
本例程中涉及控制卡初始化函数，设置脉冲模式函数之前如果设置，就不需要再次设置，只需在程序初始化中设置一次即可。

## 10.3 对称直线插补相对运动：

以两轴直线插补为例

### 10.3.1 所需函数列表：

| | |
|---|---|
| 驱动指令 | symmetry_relative_line2 |
| 读驱动状态 | get_inp_status |
| 注意事项 | 见重点备注 |

### 10.3.2 例程：

程序如下：

```
#include “adt8960.h”
void main()
{
    int cardno;
    cardno=adt8960_initial();
    if(cardno<=0) return;              //未安装ADT8960卡
                                       //
                                       //如果有多块卡，即cardno>1
                                       //可修改卡号，操作其他卡

    set_pulse_mode(0,1,1,0,0);  //设置X轴为脉冲+方向方式
```

```
    set_pulse_mode(0,2,1,0,0);          //设置Y轴为脉冲+方向
方式
    symmetry_absolute_line2(0, 1, 2, 10000,- 20000, 1000,
                                        8000, 0.1, 300,0);
    //X-Y开始插补
    //X正向移动10000步
    //Y反向移动20000步
    int s1;
    while(1)
    {
        get_inp_status(0,&s1); //读插补状态
        if(s1==0)break;     //插补结束跳出
        ……                              //可执行读键盘，显示位
置等函数
    }
    return ;
}
```

**10.3.3 重点备注：**

复合运动控制函数根据需要使用。
三轴直线插补只需换成三轴直线插补指令即可。
本例程中涉及控制卡初始化函数，设置脉冲模式函数之前如果设置，就不需要再次设置，只需在程序初始化中设置一次即可。

**10.4 对称直线插补绝对运动：**

以两轴直线插补为例

**10.4.1 所需函数列表：**

| 驱动指令 | symmetry_absolute_line2 |
|---|---|

| | |
|---|---|
| 读驱动状态 | get_inp_status |
| 注意事项 | 见重点备注 |

10.4.2 例程：

程序如下：

```
#include “adt8960h”
void main()
{
    int cardno;
    cardno=adt8960_initial();
    if(cardno<=0) return;              //未安装ADT8960卡
                                       //
                                       //如果有多块卡，即cardno>1
                                       //可修改卡号，操作其他卡

    set_pulse_mode(0,1,1,0,0); //设置X轴为脉冲+方向方式
    set_pulse_mode(0,2,1,0,0);       //设置Y轴为脉冲+方向
方式
    symmetry_absolute_line2 (0, 1, 2, 10000,- 20000, 1000,
                                       8000, 0.1, 300,0);
    //X-Y开始插补
    //X正向移动10000步
    //Y反向移动20000步
    int s1;
    while(1)
    {
        get_inp_status(0,&s1); //读插补状态
        if(s1==0)break;    //插补结束跳出
        ……                          //可执行读键盘，显示位
置等函数
```

```
        }
        return ;
    }
```

### 10.4.3 重点备注：

复合运动控制函数根据需要使用。
三轴直线插补只需换成三轴直线插补指令即可。
本例程中涉及控制卡初始化函数，设置脉冲模式函数之前如果设置，就不需要再次设置，只需在程序初始化中设置一次即可。

# 第十一章 辅助控制

本章所有例程都相互独立，在实施项目的过程中卡的初始化和倍率的设置只需设一次即可，即在程序系统初始化的过程中卡的初始化设置后，就无需设置。

## 11.1 位置锁存：

### 11.1.1 所需函数列表：

| | |
|---|---|
| 锁存模式 | set_lock_position// |
| 捕捉位置锁存是否执行 | get_lock_status |
| 获取锁存位置 | get_lock_position |
| 注意事项 | 见重点备注 |

### 11.1.2 例程：

```
    目的：
    以X轴为例绑定锁存信号说明
#include “adt8960.h”
void main()
```

```
{
    int cardno;
    int status = -1;
    long pos = -1;
    cardno=adt8960_initial();
    if(cardno<=0) return;                    //未安装adt8960卡
                                             //以下只对第一块卡X轴操作
                                             //如果有多块卡，即cardno>1
                                             //可修改卡号，操作其他卡
    set_lock_position（0,1,0,0）;
      get_lock_status(0,    1, &status);
      if(status == 1)
    {
       get_lock_position(0, 1, &pos);
    }
    return ;
}
```

**11.1.3 重点备注：**

一个锁存信号绑定在一个轴上。

本例程中涉及控制卡初始化函数之前如果设置，就不需要再次设置，只需在程序初始化中设置一次即可。

**11.2 手动驱动：**

**11.2.1 所需函数列表：**

| | |
|---|---|
| 手动手脉 | manual_pmove |
| 连续手脉 | manual_continue |
| 关闭外部信号驱动使能 | manual_disable |
| 注意事项 | 见重点备注 |

### 11.2.2 例程：

目的：
让X轴以下列速度运动20000步
起始速度：2000 pss
驱动速度：20000 pss
加/减速度：40000 pss

```
#include “adt8960.h”
void main()
{
    int cardno;
    cardno=adt8960_initial();
    if(cardno<=0) return;              //未安装ADT8960卡
                                       //以下只对第一块卡X轴操作
                                       //如果有多块卡，即cardno>1
                                       //可修改卡号，操作其他卡
    set_pulse_mode(0,1,1,0,0);         //设置X轴为脉冲+方向
方式
    set_ad_mode(0,1,0);                //设置为梯形加减速
    set_startv(0,1,2000);              //起始速度   2000
    set_speed(0,1,20000);               //驱动速度   20000
    set_acc(0,1,320);                  //加/减速度   40000/125=320
    int res = manual_pmove(0,1,20000);
    if(res != 0)//
    {
      //关闭外部信号驱动使能
      manual_disable(0,1);
    }
    return ;
}
```

### 11.2.3 重点备注：

manual_continue 函数是连续发脉冲，manual_pmove 发定量脉冲。本例程中涉及控制卡初始化函数，设置脉冲模式函数之前如果设置，就不需要再次设置，只需在程序初始化中设置一次即可。

# 第十二章 运动控制开发编程示例

所有运动控制函数均为立即返回，当驱动命令发出后，运动过程由运动控制卡控制完成，此时用户的上位机软件既可以对整个运动过程进行实时监控，也可强制停止运动过程。

**说明：轴在运动过程中，不允许向运动轴发新的驱动指令，否则会放弃上次的驱动，而执行后面的驱动指令。**

尽管编程语言“五花八门”，种类繁多，但就其本质而言，最终可以

“九九归一”。概括起来就是“三大结构和一个思想”，其中“三大结构”是指所有编程语言中都强调的顺序结构、循环结构和分支结构，一个思想主要指完成设计任务时所用到的算法以及模块划分，这是整个程序设计的重点和难点。

为了保证程序具有通用性、规范性、可扩展性以及维护方便等特点，下面所有的示例从项目设计的角度着眼，将示例划分为以下几个模块：运动控制模块(对控制卡提供的库函数进一步进行封装)，功能实现模块(配合具体工艺的代码段)，监控模块和停止处理模块。

下面我们简单介绍ADT8960卡函数库在VB和VC编程语言中的应用,如果使用其它编程语言可参照VB和VC示例程序。

## 12.1 VB 编程示例

### 12.1.1 准备工作：

(1) 新建一个项目，保存为“test.vbp”;

**(2)** 按照前面讲述的方法，在项目中添加“adt8960.bas”模块；

### 12.1.2 运动控制模块：

(1) 在项目中添加一个新模块，保存为“ctrlcard.bas”；

(2) 在运动控制模块中首先自定义运动控制卡初始化函数，对需要封装到初始化函数中的库函数进行初始化；

(3) 继续自定义相关的运动控制函数，如：速度设定函数，单轴运动函数，差补运动函数等；

(4) ctrcard.bas 的源代码见示范程序：

## 12.2 VC 编程示例

### 12.2.1 准备工作：

(1) 新建一个项目,保存为“VCExample.dsw”；

**(2)** 根据前面讲述的方法，将静态库“adt8960.lib”加载到项目中；

### 12.2.2 运动控制模块：

(1) 在项目中添加一个新类，头文件保存为“CtrlCard.h”,源文件保存为“CtrlCard.cpp”；

(2) 在运动控制模块中首先自定义运动控制卡初始化函数，对需要封装到初始化函数中的库函数进行初始化；

(3) 继续自定义相关的运动控制函数，如：速度设定函数，单轴运动函数，差补运动函数等；

头文件 “CtrlCard.h”代码见示范程序。

# 第十三章 ADT-8960 基本库函数列表

### 13.1.1 库函数列表：

| 函数类别 | 函数名称 | 功能描述 | 章节 |
|---|---|---|---|
| 基本参数 | adt8960_initial | 初始化卡 | 14.1.1 |
| | set_pulse_mode | 脉冲模式 | 14.1.2 |
| | set_limit_mode | 限位模式 | 14.1.3 |
| | set_stop0_mode | 停止模式 | 14.1.4 |
| | set_stop1_mode | 停止模式 | 14.1.5 |
| | set_delay_time | 延时状态 | 14.1.6 |
| | set_suddenstop_mode | 硬件停止 | 14.1.7 |
| | set_ad_mode | 加减速方式 | 14.1.8 |
| 驱动状态检查 | get_status | 获取单轴驱动状态 | 14.2.1 |
| | get_inp_status | 获取插补驱动状态 | 14.2.2 |
| | get_delay_status | 延时状态 | 14.2.3 |
| | get_hardware_ver | 硬件版本 | 14.2.4 |
| 运动参数设定 | set_acc | 设定加速度 | 14.3.1 |
| | set_acac | 设定加速度变化率 | 14.3.2 |
| | set_startv | 设定初始速度 | 14.3.3 |
| | set_speed | 设定驱动速度 | 14.3.4 |
| | set_command_pos | 设定逻辑计数器 | 14.3.5 |
| | set_actual_pos | 设定实位计数器 | 14.3.6 |
| | set_symmetry_speed | 设定复合加速度 | 14.3.7 |
| | set_io_mode | 通用输入输出点设置 | 14.3.8 |
| 运动参数检查 | get_command_pos | 获取逻辑位置 | 14.4.1 |
| | get_actual_pos | 获取实际位置 | 14.4.2 |
| | get_speed | 获取驱动速度 | 14.4.3 |
| | get_out | 获取输出点 | 14.4.4 |
| | get_ad | 获取加速度 | 14.4.5 |

| | | | |
|---|---|---|---|
| 驱动类 | pmove | 单轴定量驱动 | 14.5.1 |
| | continue_move | 连续驱动 | 14.5.2 |
| | dec_stop | 减速停止 | 14.5.3 |
| | sudden_stop | 立即停止 | 14.5.4 |
| | inp_move2 | 两轴插补 | 14.5.5 |
| | inp_move3 | 三轴插补 | 14.5.6 |
| | inp_move4 | 四轴插补 | 14.5.7 |
| | inp_move5 | 五轴插补 | 14.5.8 |
| | inp_move6 | 六轴插补 | 14.5.9 |
| 开关量类 | read_bit | 读单个输入点 | 14.6.1 |
| | write_bit | 输出单点 | 14.6.2 |
| 复合驱动类 | symmetry_relative_move | 单轴相对运动 | 14.7.1 |
| | symmetry_absolute_move | 单轴绝对移动 | 14.7.2 |
| | symmetry_relative_line2 | 两轴直线插补相对移动 | 14.7.3 |
| | symmetry_absolute_line2 | 两轴直线插补绝对移动 | 14.7.4 |
| | symmetry_relative_line3 | 三轴直线插补相对运动 | 14.7.5 |
| | symmetry_absolute_line3 | 三轴直线插补绝对运动 | 14.7.6 |
| | symmetry_relative_line4 | 四轴直线插补相对运动 | 14.7.7 |
| | symmetry_absolute_line4 | 四轴直线插补绝对运动 | 14.7.8 |
| | symmetry_relative_line5 | 五轴直线插补相对运动 | 14.7.9 |
| | symmetry_absolute_line5 | 五轴直线插补绝对运动 | 14.7.10 |
| | symmetry_relative_line6 | 六轴直线插补相对运动 | 14.7.11 |
| | symmetry_absolute_line6 | 六轴直线插补绝对运动 | 14.7.12 |
| 外部脉冲驱动 | manual_pmove | 外部信号定量驱动 | 14.8.1 |
| | manual_continue | 外部信号连续驱动 | 14.8.2 |
| | manual_disable | 禁用外部信号驱动 | 14.8.3 |
| 位置锁存 | set_lock_position | 设置位置锁存工作模式 | 14.9.1 |
| | get_lock_status | 获取锁存状态 | 14.9.2 |
| | get_lock_position | 获取锁存位置 | 14.9.3 |

| | | | |
|---|---|---|---|
| | clr_lock_status | 清除锁存状态 | 14.9.4 |
| 硬件缓存类函数 | fifo_inp_move1 | 1 轴缓存 | 14.10.1 |
| | fifo_inp_move2 | 2 轴缓存 | 14.10.2 |
| | fifo_inp_move3 | 3 轴缓存 | 14.10.3 |
| | fifo_inp_move4 | 4 轴缓存 | 14.10.4 |
| | fifo_inp_move5 | 5 轴缓存 | 14.10.5 |
| | fifo_inp_move6 | 6 轴缓存 | 14.10.6 |
| | reset_fifo | 清除缓存 | 14.10.7 |
| | read_fifo_count | 缓存状态 | 14.10.8 |
| | read_fifo_empty | 缓存状态 | 14.10.9 |
| | read_fifo_full | 缓存状态 | 14.10.10 |
| 手动减速 | set_dec_mode | 设置手动减速模式 | 14.11.1 |
| | set_dec_pos1 | 设置手动减速点 | 14.11.2 |
| | set_dec_pos2 | 到减速点后剩余位置 | 14.11.3 |
| | clr_dec_status | 清除手动减速状态 | 14.11.4 |
| | get_dec_status | 获取手动减速状态 | 14.11.5 |
| | set_end_speed | 设置拖尾速度 | 14.11.6 |
| 自动回原点 | SetHomeMode_Ex | 设置回零参数 | 14.12.1 |
| | SetHomeSpeed_Ex | 回零速度参数 | 14.12.2 |
| | HomeProcess_Ex | 启动回零 | 14.12.3 |
| | GetHomeStatus_Ex | 获取回零状态 | 14.12.4 |

# 第十四章 ADT-8960 基本库函数详解

## 14.1 基本参数设置类

### 14.1.1 初始化卡：

int adt8960_initial(void)

**功能：**

初始化卡

**返回值:**

(1)返回值>0 时，表示ADT8960 卡的数量。如果为3，则下面的可用卡号分别为0、1、2；

(2)返回值=0 时，说明没有安装ADT8960 卡；

(3)返回值<0 时，-1 表示没有安装相关服务，-2 表示PCI 桥存在故障。

**注意：初始化函数是调用其它函数的前提，所以必须最先调用，以确认可使用的卡数以及初始化一些参数。**

## 14.1.2 设置输出脉冲工作方式：

int set_pulse_mode(int cardno, int axis, int value,int logic,int dir_logic)

**功能：**

设置输出脉冲的工作方式

**参数：**

cardno 卡号

axis 轴号(1-6)

value 0：脉冲+脉冲方式 1：脉冲+方向方式

脉冲/方向都是正逻辑设定时

| 脉冲输出方式 | 驱动方向 | 输出信号波形 | |
|---|---|---|---|
| | | PU/CW 信号 | DR/CCW 信号 |
| 独立2脉冲方式 | +方向驱动输出 | | Low 电平 |
| | -方向驱动输出 | Low 电平 | |
| 1脉冲方式 | +方向驱动输出 | | Low 电平 |
| | -方向驱动输出 | | Hi 电平 |

logic 0： 正逻辑脉冲 1： 负逻辑脉冲

正逻辑脉冲 ：  负逻辑脉冲 ： 

dir-logic 0：方向输出信号正逻辑 1：方向输出信号负逻辑

| dir_logic | 正方向脉冲输出时 | 负方向脉冲输出时 |
|---|---|---|
| 0 | Low | Hi |
| 1 | Hi | Low |

**返回值** 0：正确 1：错误

**默认模式为：脉冲+方向，正逻辑脉冲，方向输出信号正逻辑**

### 14.1.3 设定正/负方向限位输入 nLMT 信号的模式：

int set_limit_mode(int cardno, int axis, int v1,int v2,int logic)

**功能：**

设定正/负方向限位输入nLMT 信号的模式

**参数：**

| | | |
|---|---|---|
| cardno | 卡号 | |
| axis | 轴号(1-6) | |
| v1 | 0：正限位有效 | 1：正限位无效 |
| v2 | 0：负限位有效 | 1：负限位无效 |
| logic | 0：低电平有效 | 1：高电平有效 |

**返回值** 0：正确 1：错误

**默认模式为：正、负限位低电平有效**

### 14.1.4 设定 stop0 输入信号的模式：

int set_stop0_mode(int cardno, int axis, int v,int logic)

**功能：**

设定stop0输入信号的模式

**参数：**

| | | |
|---|---|---|
| cardno | 卡号 | |
| axis | 轴号(1-6) | |
| v | 0：无效 | 1：有效 |
| logic | 0：低电平有效 | 1：高电平有效 |

**返回值** 0：正确 1：错误

**默认模式为： 无效**

### 14.1.5 设定 stop1 输入信号的模式：

int set_stop1_mode(int cardno, int axis, int v,int logic)

**功能:**

设定stop1 输入信号的模式

**参数：**

cardno 卡号

axis 轴号(1-6)

v 0：无效 1：有效

logic 0：低电平有效 1：高电平有效

**返回值** 0：正确 1：错误

默认模式为：无效

### 14.1.6 延时时间：

int set_delay_time(int cardno,long time)

**功能:**

延时时间

**参数：**

cardno 卡号

time 延时时间

**返回值** 0：正确 1：错误

**说明：** 时间单位为1/8us，最大值为长整型的最大值。

### 14.1.7 硬件停止：

int set_suddenstop_mode(int cardno,int v,int logical)

**功能:**

硬件停止

**参数：**

| | | |
|---|---|---|
| cardno | 卡号 | |
| v | 0：无效 | 1：有效 |
| logical | 0：低电平 | 1：高电平 |
| **返回值** | 0：正确 | 1：错误 |

**说明：**硬件停止信号固定使用P2端子板33引脚(IN31)

### 14.1.8 加减速方式设定：

int set_ad_mode(int cardno,int axis,int mode)

**功能：**选择加减速模式，梯形或者 S 型

**参数：**

| | | |
|---|---|---|
| cardno | 卡号 | |
| axis | 轴号(1-6) | |
| mode | 0：直线加/减速 | 1：S 曲线加/减速 |
| **返回值** | 0：正确 | 1：错误 |

**默认模式为 ：**直线加/减速

## 14.2 驱动状态检查类

### 14.2.1 获取各轴驱动状态：

int get_status(int cardno,int axis,int *value)

**功能：**

获取各轴的驱动状态

**参数：**

| | | |
|---|---|---|
| cardno | 卡号 | |
| axis | 轴号(1-6) | |
| value | 驱动状态的指针 | |
| | 0：驱动结束 | 非 0：正在驱动 |
| **返回值** | 0：正确 | 1：错误 |

### 14.2.2 获取插补的驱动状态：

int get_inp_status(int cardno,int *value)

**功能：**

获取插补的驱动状态

**参数：**

cardno 卡号

value 插补状态的指针

**返回值** 0：插补结束 1：正在插补 0：正确 1：错误

### 14.2.3 延时状态：

int get_delay_status(int cardno)

**功能：**

延时状态

**参数：**

cardno 卡号

**返回值** 0：延时结束 1：延时进行中

### 14.2.4 硬件版本：

float get_hardware_ver(int cardno)

**功能：**

硬件版本

**参数：**

cardno 卡号

**返回值** 1.1:硬件版本为1.1 1.2:硬件版本1.2版

注：这里的返回值为多少就是多少，这里的1.1，1.2只是暂时说明用目前的硬件版本为1.3。

## 14.3 基本参数设置类

**注意：以下参数在初始化后值不确定，使用前必须设定**

### 14.3.1 运动参数设定类：

int set_acc(int cardno,int axis,long value)

**功能：**

加速度设定

**参数：**

cardno 卡号

axis 轴号

value 加速度(0-32000)

**返回值** 0：正确 1：错误

### 14.3.2 设定加速度变化率：

int set_acac(int cardno,int axis,long value)

**功能：**加/减速度的变化率设定

**参数：**

cardno 卡号

axis 轴号

value K值（1-65535）

**返回值** 0：正确 1：错误

### 14.3.3 初始速度设定：

int set_startv(int cardno,int axis,long value)

**功能：**

初始速度设定

**参数：**

cardno 卡号
axis 轴号
value 起始速度(1-2M)
**返回值** 0：正确 1：错误

### 14.3.4 驱动速度设定：

int set_speed(int cardno,int axis,long value)
**功能：**
驱动速度的设定
**参数：**
cardno 卡号
axis 轴号
value 速度(1-2M)
**返回值** 0：正确 1：错误

### 14.3.5 逻辑位置计数器设定：

int set_command_pos(int cardno,int axis,long value)
**功能：**
设定逻辑位置计数器的数值
**参数：**
cardno 卡号
axis 轴号
value 范围(-2147483648~+2147483647)
**返回值** 0：正确 1：错误
逻辑位置计数器任何时候都能进行读写操作

### 14.3.6 实际位置计数器设定：

int set_actual_pos(int cardno,int axis,long value)

**功能：**

设定实际位置计数器的数值

**参数：**

cardno 卡号

axis 轴号

value 范围(-2147483648~+2147483647)

**返回值** 0：正确 1：错误

实际位置计数器任何时候都能进行读写操作

### 14.3.7 设定复合加速度的值：

int set_symmetry_speed(int cardno,int axis,long lspd,long hspd,double tacc)

**功能:**

设定加速度

**参数:**

cardno 卡号

axis 轴号(1-6)

lspd 起步速度

hspd 驱动速度

tacc 加速时间

**返回值** 0:正确 1:错误

### 14.3.8 设定输入输出：

int set_io_mode(int cardno,int v1,int v2)

**功能：**

设定输入输出

**参数：**

cardno 卡号

v1 0:前面8 个点定义为输入 1:前面的8 个点定义为输出

v2 0:后面8个点定义为输入 1:后面的8个点定义为输出

返回值 0:正确 1:错误

**注：**前8个可配置的IO作为输入点时，对应的是IN44~IN51;

前8个可配置的IO作为输出点时，对应的是OUT16~OUT23

后8个可配置的IO作为输入点时，对应的是IN52~IN59;

后8个可配置的IO作为输出点时，对应的是OUT24~OUT31

## 14.4 运动参数检查类

***以下函数在任何时候均可调用***

### 14.4.1 获取各轴的逻辑位置：

int get_command_pos(int cardno,int axis,long *pos)

**功能：**

获取各轴的逻辑位置

**参数：**

cardno 卡号

axis 轴号

pos 逻辑位置值的指针

**返回值** 0：正确 1：错误

此函数可随时得到轴的逻辑位置，在电机未失步的情况下，pos 的值代表轴的当前位置。

### 14.4.2 获取各轴的实际位置（即编码器反馈输入值）：

int get_actual_pos(int cardno,int axis,long *pos)

**功能：**

获取各轴的实际位置

**参数：**

cardno 卡号

axis 轴号

pos 实际位置值的指针

**返回值** 0：正确 1：错误

此函数可随时得到轴的实际位置，在电机有失步的情况下，pos 的值依然代表轴的实际位置。

### 14.4.3 获取各轴的当前驱动速度：

int get_speed(int cardno,int axis,long *speed)

**功能：**

获取各轴当前的驱动速度

**参数：**

cardno 卡号

axis 轴号

speed 当前驱动速度的指针

**返回值** 0：正确 1：错误

数据的单位和驱动设定数值V一样。

此函数可随时得到轴的驱动速度。

### 14.4.4 获取输出点：

int get_out(int cardno, int number)

**功能**

获取输出状态

**参数:**

cardno 卡号

number 端口号

**返回值**:指定端口的当前状态,-1 表示参数错误

### 14.4.5 获取各轴的当前加速度：

int get_ad(int cardno,int axis,long *ad)

**功能：**获取各轴的当前加速度

**参数：**

| | |
|---|---|
| cardno | 卡号 |
| axis | 轴号 |
| ad | 当前加速度的指针 |

**返回值** 0：正确 1：错误

**说明：**数据的单位和驱动加速度设定数值 A 一样

## 14.5 驱动类

### 14.5.1 定量驱动：

int pmove(int cardno,int axis,long pulse)

**功能：**

定量驱动

**参数：**

| | |
|---|---|
| cardno | 卡号 |
| axis | 轴号 |
| pulse | 输出的脉冲数 |
| | >0：正方向移动 |
| | <0：负方向移动 |
| | 范围（-268435455~+268435455） |

**返回值** 0：正确 1：错误

**注意：写入驱动命令之前一定要正确地设定速度曲线所需的参数**

### 14.5.2 连续驱动：

int continue_move(int cardno,int axis,int dir)

**功能：**

连续驱动

**参数：**

cardno 卡号

axis 轴号

dir 方向0:正 1:负

**返回值** 0：正确 1：错误

### 14.5.3 驱动减速停止：

int dec_stop(int cardno,int axis)

**功能：**

驱动减速停止

**参数：**

cardno 卡号

axis 轴号

**返回值** 0：正确 1：错误

在驱动脉冲输出过程中，此命令作出减速停止，驱动速度比初始速度慢的时候也可以用本命令立即停止。

**注意：直线插补时，如需要减速停止，应当只对最前的插补轴使用此指令，否则可能不能达到预定的结果。**

### 14.5.4 驱动立即停止：

int sudden_stop(int cardno,int axis)

**功能：**

驱动立即停止

**参数：**

cardno 卡号

axis 轴号

**返回值** 0：正确 1：错误

立即停止正在驱动中的脉冲输出，在加/减速驱动中也立即停止。

**注意：直线插补时，如需要立即停止，应当只对最前的插补轴使用此指令，否则可能不能达到预定的结果。**

### 14.5.5 两轴直线插补：

int inp_move2(int cardno,int axis1,int axis2,long pulse1,long pulse2)

**功能：**

两轴直线插补

**参数：**

cardno 卡号

axis1 ,axis2 参与插补的轴号

pulse1,pulse2 移动的相对距离 范围（-8388608~+8388607）

**返回值** 0：正确 1：错误

### 14.5.6 三轴直线插补：

int inp_move3(int cardno,int axis1,int axis2,int axis3,long pulse1,long pulse2,long pulse3)

**功能：**

三轴直线插补

**参数：**

cardno 卡号

axis1 ,axis2,axis3 参与插补的轴号

pulse1,pulse2,pulse3 指定轴axis1,axis2,axis3移动的相对距离

范围（-8388608~+8388607）

**返回值** 0：正确 1：错误

### 14.5.7 四轴直线插补：

int inp_move4(int cardno, int axis1,int axis2,int axis3,int axis4,long pulse1,long pulse2,long pulse3,long pulse4)

**功能：**

四轴直线插补

**参数：**

cardno 卡号

axis1 ,axis2,axis3,axis4 参与插补的轴号

pulse1,pulse2,pulse3,pulse4四轴移动的相对距离

范围(-8388608~+8388607）

**返回值** 0：正确 1：错误

### 14.5.8 五轴直线插补：

int inp_move5(int cardno,int axis1,int axis2,int axis3,int axis4, int axis5,long pulse1,long pulse2,long pulse3,long pulse4,long pulse5)

**功能：**

五轴直线插补

**参数：**

cardno 卡号

axis1 ,axis2,axis3,axis4,axis5 参与插补的轴号

pulse1,pulse2,pulse3,pulse4,pulse5五轴移动的相对距离

范围（-8388608~+8388607）

**返回值** 0：正确 1：错误

### 14.5.9 六轴直线插补：

int inp_move6(int cardno,long pulse1,long pulse2,long pulse3,long pulse4,long pulse5,long pulse6)

**功能：**

六轴直线插补

**参数：**

cardno 卡号

pulse1,pulse2,pulse3,pulse4,pulse5,pulse6 四轴移动的相对距离

范围（-8388608~+8388607）

**返回值** 0：正确 1：错误

## 14.6 开关量输入输出类

### 14.6.1 读单个输入点：

int read_bit(int cardno,int number)

**功能：**

读取单个输入点

**参数：**

cardno 卡号

number 输入点（0-31）

**返回值** 0：低电平 1：高电平

-1：错误

### 14.6.2 输出单点：

int write_bit(int cardno,int number,int value)

**功能：**

输出单个点设定

**参数：**

cardno 卡号

number 输出点（0-15）

value 0：低 1：高

**返回值** 0：正确 1：错误

输出数number对应相应的输出号。

## 14.7 复合类型类

为了方便客户的使用，我们使用基本库函数封装了复合驱动类的函数，这部分函数主要是将速度模式设置、速度参数设置和运动函数进行了集成，

另外考虑了绝对运动和相对运动的概念。

### 14.7.1 单轴相对运动：

int symmetry_relative_move(int cardno, int axis, long pulse, long lspd ,long hspd, double tacc, long vacc, int mode)

**功能:**

参照当前位置,以加速进行定量移动

**参数:**

| | |
|---|---|
| cardno | 卡号 |
| axis | 轴号 |
| pulse | 脉冲 |
| lspd | 低速 |
| hspd | 高速 |
| tacc | 加速时间(单位:秒) |
| vacc | 加速度变化率 |
| mode | 模式（0:梯形，1:S曲线） |

**返回值** 0：正确 1：错误

### 14.7.2 单轴绝对运动：

int symmetry_absolute_move(int cardno, int axis, long pulse, long lspd ,long hspd, double tacc, long vacc, int mode)

**功能:**

参照零点位置,以加速进行定量移动

**参数:**

| | |
|---|---|
| cardno | 卡号 |
| axis | 轴号 |
| pulse | 脉冲 |
| lspd | 低速 |
| hspd | 高速 |
| tacc | 加速时间(单位:秒) |
| vacc | 加速度变化率 |
| mode | 模式（0:梯形，1:S曲线） |

**返回值** 0：正确 1：错误

### 14.7.3 二轴直线插补相对运动：

int symmetry_relative_line2(int cardno, int axis1, int axis2, long pulse1, long pulse2, long lspd ,long hspd, double tacc, long vacc, int mode)

**功能:**

参照当前位置,以加速进行直线插补

**参数:**

| | |
|---|---|
| cardno | 卡号 |
| axis1 | 轴号1 |
| axis2 | 轴号2 |
| pulse1 | 脉冲1 |
| pulse2 | 脉冲2 |
| lspd | 低速 |
| hspd | 高速 |
| tacc | 加速时间(单位:秒) |
| vacc | 加速度变化率 |
| mode | 模式（0:梯形，1:S曲线） |

**返回值** 0：正确 1：错误

### 14.7.4 二轴直线插补绝对运动：

int symmetry_absolute_line2(int cardno, int axis1, int axis2, long pulse1, long pulse2, long lspd ,long hspd, double tacc, long vacc, int mode)

**功能:**

参照零点位置,以加速进行直线插补

**参数:**

| | |
|---|---|
| cardno | 卡号 |
| axis1 | 轴号1 |
| axis2 | 轴号2 |
| pulse1 | 脉冲1 |
| pulse2 | 脉冲2 |
| lspd | 低速 |
| hspd | 高速 |
| tacc | 加速时间(单位:秒) |
| vacc | 加速度变化率 |
| mode | 模式（0:梯形，1:S曲线） |

**返回值** 0：正确 1：错误

### 14.7.5 三轴直线插补相对运动：

int symmetry_relative_line3(int cardno, int axis1, int axis2, int axis3, long pulse1, long pulse2, long pulse3, long lspd ,long hspd, double tacc, long vacc, int mode)

**功能:**

参照当前位置,以加速进行直线插补

**参数:**

| | |
|---|---|
| cardno | 卡号 |
| axis1 | 轴号1 |
| axis2 | 轴号2 |
| axis3 | 轴号3 |
| pulse1 | 脉冲1 |
| pulse2 | 脉冲2 |
| pulse3 | 脉冲3 |
| lspd | 低速 |
| hspd | 高速 |
| tacc | 加速时间(单位:秒) |
| vacc | 加速度变化率 |
| mode | 模式（0:梯形，1:S曲线） |

**返回值** 0：正确 1：错误

## 14.7.6 三轴直线插补绝对运动：

int symmetry_absolute_line3(int cardno, int axis1, int axis2, int axis3, long pulse1, long pulse2, long pulse3, long lspd ,long hspd, double tacc, long vacc, int mode)

**功能:**

参照零点位置,以加速进行直线插补

**参数:**

| | |
|---|---|
| cardno | 卡号 |
| axis1 | 轴号1 |
| axis2 | 轴号2 |
| axis3 | 轴号3 |
| pulse1 | 脉冲1 |
| pulse2 | 脉冲2 |
| pulse3 | 脉冲3 |
| lspd | 低速 |
| hspd | 高速 |
| tacc | 加速时间(单位:秒) |
| vacc | 加速度变化率 |

mode 模式（0:梯形，1:S曲线）

**返回值** 0：正确 1：错误

## 14.7.7 四轴直线插补相对运动：

int symmetry_relative_line4(int cardno,int axis1, int axis2, int axis3,int axis4,long pulse1, long pulse2, long pulse3, long pulse4,long lspd ,long hspd, double tacc, long vacc, int mode)

**功能:**

参照当前位置,以加速进行直线插补

**参数:**

cardno 卡号
axis1 轴号1
axis2 轴号2
axis3 轴号3
axis4 轴号4
pulse1 脉冲1
pulse2 脉冲2
pulse3 脉冲3
pulse4 脉冲4
lspd 低速
hspd 高速
tacc 加速时间(单位:秒)
vacc 加速度变化率
mode 模式（0:梯形，1:S曲线）

**返回值** 0：正确 1：错误

## 14.7.8 四轴直线插补绝对运动：

int symmetry_absolute_line4(int cardno, int axis1, int axis2, int axis3,int axis4,long pulse1, long pulse2, long pulse3, long pulse4,long lspd ,long hspd, double tacc, long vacc, int mode)

**功能:**

参照零点位置,以加速进行直线插补

**参数:**

cardno 卡号

axis1 轴号1
axis2 轴号2
axis3 轴号3
axis4 轴号4
pulse1 脉冲1
pulse2 脉冲2
pulse3 脉冲3
pulse4 脉冲4
lspd 低速
hspd 高速
tacc 加速时间(单位:秒)
vacc 加速度变化率
mode 模式（0:梯形，1:S曲线）

**返回值** 0：正确 1：错误

## 14.7.9 五轴直线插补相对运动：

int symmetry_relative_line5(int cardno,int axis1, int axis2, int axis3,int axis4,int axis5,long pulse1, long pulse2, long pulse3, long pulse4,long pulse5,long lspd ,long hspd, double tacc, long vacc, int mode)

**功能:**

参照当前位置,以加速进行直线插补

**参数:**

cardno 卡号
axis1 轴号1
axis2 轴号2
axis3 轴号3
axis4 轴号4
axis5 轴号5
pulse1 脉冲1
pulse2 脉冲2
pulse3 脉冲3
pulse4 脉冲4
pulse5 脉冲5
lspd 低速
hspd 高速
tacc 加速时间(单位:秒)
vacc 加速度变化率
mode 模式（0:梯形，1:S曲线）

**返回值** 0：正确 1：错误

## 14.7.10 五轴直线插补绝对运动：

int symmetry_absolute_line5(int cardno, int axis1, int axis2, int axis3,int axis4,int axis5,long pulse1, long pulse2, long pulse3, long pulse4,long pulse5,long lspd ,long hspd, double tacc, long vacc, int mode)

**功能:**

参照零点位置,以加速进行直线插补

**参数:**

| | |
|---|---|
| cardno | 卡号 |
| axis1 | 轴号1 |
| axis2 | 轴号2 |
| axis3 | 轴号3 |
| axis4 | 轴号4 |
| axis5 | 轴号5 |
| pulse1 | 脉冲1 |
| pulse2 | 脉冲2 |
| pulse3 | 脉冲3 |
| pulse4 | 脉冲4 |
| pulse5 | 脉冲5 |
| lspd | 低速 |
| hspd | 高速 |
| tacc | 加速时间(单位:秒) |
| vacc | 加速度变化率 |
| mode | 模式（0:梯形，1:S曲线） |

**返回值** 0：正确 1：错误

## 14.7.11 六轴直线插补相对运动：

int symmetry_relative_line6(int cardno,long pulse1, long pulse2, long pulse3, long pulse4,long pulse5, long pulse6,long lspd ,long hspd, double tacc, long vacc, int mode)

**功能:**

参照当前位置,以加速进行直线插补

**参数:**

| | |
|---|---|
| cardno | 卡号 |
| pulse1 | 脉冲1 |

| | |
|---|---|
| pulse2 | 脉冲2 |
| pulse3 | 脉冲3 |
| pulse4 | 脉冲4 |
| pulse5 | 脉冲5 |
| pulse6 | 脉冲6 |
| lspd | 低速 |
| hspd | 高速 |
| tacc | 加速时间(单位:秒) |
| vacc | 加速度变化率 |
| mode | 模式（0:梯形，1:S曲线） |

**返回值** 0：正确 1：错误

### 14.7.12 六轴直线插补绝对运动：

int symmetry_absolute_line6(int cardno,long pulse1, long pulse2, long pulse3, long pulse4,long pulse5, long pulse6,long lspd ,long hspd, double tacc, long vacc, int mode)

**功能:**

参照零点位置,以加速进行直线插补

**参数:**

| | |
|---|---|
| cardno | 卡号 |
| pulse1 | 脉冲1 |
| pulse2 | 脉冲2 |
| pulse3 | 脉冲3 |
| pulse4 | 脉冲4 |
| pulse5 | 脉冲5 |
| pulse6 | 脉冲6 |
| lspd | 低速 |
| hspd | 高速 |
| tacc | 加速时间(单位:秒) |
| vacc | 加速度变化率 |
| mode | 模式（0:梯形，1:S曲线） |

**返回值** 0：正确 1：错误

## 14.8 外部信号驱动

### 14.8.1 外部信号定量驱动：

int manual_pmove(int cardno, int axis, long pos)

**功能：**外部信号定量驱动

**参数：**

cardno 卡号

axis 轴号

返回值 0：正确 1：错误

**说明:**(1)发出定量脉冲，但驱动没有立即进行，需要等到外部信号电平发生变化

(2)可以使用普通按钮,也可以接手轮

### 14.8.2 外部信号连续驱动：

int manual_continue(int cardno, int axis)

**功能：**外部信号连续驱动

**参数：**

cardno 卡号

axis 轴号

**返回值** 0：正确 1：错误

**说明:**(1)发出连续的脉冲，但驱动没有立即进行，需要等到外部信号电平发生变化

(2)可以使用普通按钮,也可以接手轮

### 14.8.3 关闭外部信号驱动：

int manual_disable (int cardno, int axis)

**功能:**关闭外部信号驱动使能

**参数：**

cardno 卡号

axis 轴号(1-6)

**返回值** 0：正确 1：错误

**说明:**关闭外部信号驱动使能，注意在使用外部信号发脉冲的过程中，使用该功

能并不能停止脉冲，要停止脉冲使用停止命令。

## 14.9 位置锁存

### 14.9.1 位置锁存工作模式设置：

int set_lock_position(int cardno, int axis,int mode,int regi,int logical)

**功能:**设置到位信号功能,锁定所有轴的逻辑位置和实际位置

**参数:**

| | | | |
|---|---|---|---|
| axis | 参照轴 | | |
| mode | 锁存模式 | 0:无效 | 1:有效 |
| regi | 计数器模式 | 0:逻辑位置 | 1:实际位置 |
| logical | 电平信号 | 0:上升沿 | 1:下降沿 |

**返回值** 0：正确 1：错误

**说明:**使用指定轴 axis 的 IN 信号作为触发信号

### 14.9.2 获取锁存状态：

int get_lock_status(int cardno, int axis, int *v)

**功能:**获取锁存操作的状态

**参数:**

| | | |
|---|---|---|
| cardno | 卡号 | |
| axis | 轴号(1-6) | |
| v | 0：未执行锁存操作 | 1：执行过锁存操作 |

**返回值** 0：正确 1：错误

**说明:**利用该函数可以捕捉位置锁存是否执行

### 14.9.3 获取锁定的位置：

int get_lock_position(int cardno,int axis,long *pos)

**功能:**获取锁定的位置

**参数:**

cardno 卡号
axis 轴号(1-6)
pos 锁存的位置
**返回值** 0：正确 1：错误

### 14.9.4 清除锁定：

int _stdcall clr_lock_status(int cardno, int axis)
**功能:**清除锁定
**参数:**
cardno 卡号
axis 轴号(1-6)
**返回值** 0：正确 1：错误

## 14.10 硬件缓存

### 14.10.1 单轴缓存：

int fifo_inp_move1(int cardno,int axis1,long pulse1,long speed)
**功能:**单轴缓存
**参数:**
cardno 卡号
axis1 轴号(1-6)
pulse1 缓存的脉冲
speed 缓存的速度
**返回值** 0：正确 1：错误
**说明:**共有 2048 个缓存空间，每条单轴缓存指令占用 3 个空间，可缓存 682 条指令

### 14.10.2 两轴缓存：

int fifo_inp_move2(int cardno,int axis1,int axis2,long pulse1,long pulse2,long speed)

**功能:**两轴缓存

**参数:**

| | |
|---|---|
| cardno | 卡号 |
| axis1 | 轴号(1-6) |
| axis2 | 轴号(1-6) |
| pulse1 | 缓存的脉冲数 |
| pulse2 | 缓存的脉冲数 |
| speed | 缓存的速度 |

**返回值** 0：正确 1：错误

**说明:** 共有 2048 个缓存空间，每条两轴缓存指令占用 4 个空间，可缓存 512 条指令

### 14.10.3 三轴缓存：

int fifo_inp_move3(int cardno,int axis1,int axis2,int axis3,long pulse1,long pulse2,long pulse3,long speed)

**功能:**三轴缓存

**参数:**

| | |
|---|---|
| cardno | 卡号 |
| axis1 | 轴号(1-6) |
| axis2 | 轴号(1-6) |
| axis3 | 轴号(1-6) |
| pulse1 | 缓存的脉冲数 |
| pulse2 | 缓存的脉冲数 |
| pulse3 | 缓存的脉冲数 |
| speed | 缓存的速度 |

**返回值** 0：正确 1：错误

**说明:** 共有 2048 个缓存空间，每条三轴缓存指令占用 5 个空间，可缓存 409 条指令

### 14.10.4 四轴缓存：

int fifo_inp_move4(int cardno,int axis1,int axis2,int axis3,int axis4,long pulse1,long pulse2,long pulse3,long pulse4,long speed)

**功能:**四轴缓存

**参数:**

| | |
|---|---|
| cardno | 卡号 |
| axis1 | 轴号(1-6) |
| axis2 | 轴号(1-6) |
| axis3 | 轴号(1-6) |
| axis4 | 轴号(1-6) |
| pulse1 | 缓存的脉冲数 |
| pulse2 | 缓存的脉冲数 |
| pulse3 | 缓存的脉冲数 |
| pulse4 | 缓存的脉冲数 |
| speed | 缓存的速度 |

**返回值** 0：正确 1：错误

**说明:**共有 2048 个缓存空间，每条四轴缓存指令占用 6 个空间，可缓存 341 条指令

### 14.10.5 五轴缓存：

int fifo_inp_move5(int cardno,int axis1,int axis2,int axis3,int axis4,int axis5,long pulse1,long pulse2,long pulse3,long pulse4,long pulse5,long speed)

**功能:**五轴缓存

**参数:**

| | |
|---|---|
| cardno | 卡号 |
| axis1 | 轴号(1-6) |
| axis2 | 轴号(1-6) |
| axis3 | 轴号(1-6) |
| axis4 | 轴号(1-6) |
| axis5 | 轴号(1-6) |

| | |
|---|---|
| pulse1 | 缓存的脉冲数 |
| pulse2 | 缓存的脉冲数 |
| pulse3 | 缓存的脉冲数 |
| pulse4 | 缓存的脉冲数 |
| pulse5 | 缓存的脉冲数 |
| speed | 缓存的速度 |

**返回值** 0：正确 1：错误

**说明:** 共有 2048 个缓存空间，每条五轴缓存指令占用 7 个空间，可缓存 292 条指令

### 14.10.6 六轴缓存：

int fifo_inp_move6(int cardno,long pulse1,long pulse2,long pulse3,long pulse4,long pulse5,long pulse6,long speed)

**功能:**六轴缓存

**参数:**

| | |
|---|---|
| cardno | 卡号 |
| pulse1 | 缓存的脉冲数 |
| pulse2 | 缓存的脉冲数 |
| pulse3 | 缓存的脉冲数 |
| pulse4 | 缓存的脉冲数 |
| pulse5 | 缓存的脉冲数 |
| pulse6 | 缓存的脉冲数 |
| speed | 缓存的速度 |

**返回值** 0：正确 1：错误

**说明:** 共有 2048 个缓存空间，每条六轴缓存指令占用 8 个空间，可缓存 256 条指令

### 14.10.7 清除缓存：

int reset_fifo(int cardno)

**功能:**清除缓存

**参数:**

cardno 卡号

**返回值** 0：正确 1：错误

### 14.10.8 读取缓存：

int _stdcall read_fifo_count(int cardno,int *value)

**功能:**读取缓存数，存放进去的指令还剩多少未执行

**参数:**

cardno 卡号

value 返回缓存区中未执行指令的字节数

**返回值** 0：正确 1：错误

### 14.10.9 读取缓存是否为空：

int read_fifo_empty(int cardno)

**功能:**读取缓存是否为空

**参数:**

cardno 卡号

**返回值** 0：非空 1：空

### 14.10.10 读取缓存是否满：

int read_fifo_full(int cardno)

**功能:**读取缓存是否满了，满了之后将不能再存数据

**参数:**

cardno 卡号

**返回值** 0：未满 1：满

### 14.10.11 手动减速功能：

首先需要设置手动减速模式 ；当手动减速模式有效时，设置手动减速点 pos1 ，设置原点偏移量 pos2 ， 设置手动减速点后的结束速度(低速) endspeed 。

手动减速运动过程：运动到减速点 pos1 后自动减速到 endspeed 运行，以速度 endspeed 搜索原点信号，原点信号需要外部信号触发(stop0 的低电平触发)，当原点信号触发后，运动原点偏移量 pos2 后立即停止 ，当没有搜索到原点信号时以 endspeed 一直运动下去。

### 14.10.12 设置手动减速模式：

int set_dec_mode(int cardno,int axis,int mode)

**功能:**设置手动减速模式

**参数:**

| | |
|---|---|
| cardno | 卡号 |
| axis | 轴号(1-6) |
| mode | 手动减速模式 0 无效 1 有效 |
| **返回值** | 0：正确 1：错误 |

### 14.10.13 设置手动减速点：

int set_dec_pos1(int cardno, int axis,long pos)

**功能:**设置手动减速点

**参数:**

| | |
|---|---|
| cardno | 卡号 |
| axis | 轴号(1-6) |
| pos | 减速点 |

**返回值** 0：正确 1：错误

**说明：**到了该减速点后自动减速到指定的低速运动，寻找原点的触发信号，如果没有找到触发信号，将以此速度一直运动下去，直到运动结束

### 14.10.14 设置手动减速偏移量：

int set_dec_pos2(int cardno, int axis,long pos)

**功能:**原点的偏移量

**参数:**

cardno 卡号

axis 轴号(1-6)

pos 剩余位置

**返回值** 0：正确 1：错误

**说明：**运动该偏移量需要外部信号触发，触发信号为stop0的低电平

### 14.10.15 清除手动减速状态：

int clr_dec_status(int cardno, int axis)

**功能:**清除手动减速状态

**参数:**

cardno 卡号

axis 轴号(1-6)

**返回值** 0：正确 1：错误

### 14.10.16 获取手动减速状态：

int get_dec_status(int cardno,int axis,int *sta)

**功能:**获取手动减速状态

**参数:**

cardno 卡号

axis 轴号(1-6)

sta 减速状态

0:正在搜索

1:搜索完毕

2:运动已经停止，减速点没有发现

3: 没有达到实际的偏移

4:搜索减速点过程中伺服关闭

5:减速点已经发现，在向 offset 运动时没有到位（可能是限位触发等）

**返回值** 0：正确 1：错误

### 14.10.17 设定拖尾速度：

int set_end_speed(int cardno,int axis,long value)

**功能:**设置手动减速点后，低速寻找原点信号速度

**参数:**

| | |
|---|---|
| cardno | 卡号 |
| axis | 轴号(1-6) |
| value | 范围(1-2M) |

**返回值** 0：正确 1：错误

**说明：**如果在手动减速点pos1 之后没有找到触发信号(减速原点信号)，将以此速度一直运动下去

## 14.11 回原点功能

回原点动作说明:

(1) 回原点分为四大步:

第一步:快速接近stop0(logical0原点设置)，找到stop0;

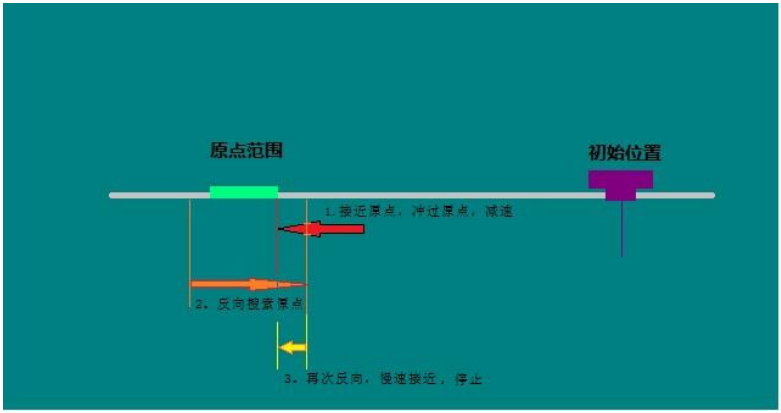
第二步:慢速反向离开stop0，反向移动指定原点范围脉冲数;

第三步:再次慢速接近stop0;

第四步:慢速接近stop1(logical1编码器Z相).

(2) 第四步可以选择是否执行,通过logical1来选择.

(3) 注意：调用该函数后如果回原点正在执行中，不会立即返回，若需多轴回原点,必须等待上一轴回原点结束后，才能执行下一轴的回原点动作.




## 14.11.1 设置回零模式：

int SetHomeMode_Ex(int m_nCardNum,int m_nAxisNum,int m_nHomeDir, int m_nStop0Active,int m_nLimitActive,int m_nStop1Active,long m_nBackRange,long m_nEncoderZRange,long m_nOffset);

**功能**：设置回零信号，步骤参数

**参数**：

int m_nCardNum 卡号

int m_nAxisNum 轴号

int m_nHomeDir 回零方向 0:负方向 1:正方向

int m_nStop0Active stop0 有效电平设置；0：低电平停止 1：高电平停止

int m_nLimitActive limit信号 有效电平设置；0：低电平停止 1：高电平停止

int m_nStop1Active stop1 有效电平设置；0：低电平停止 1：高电平停止 -1：无效，不启用stop1

long m_nBackRange 反向距离 >1

long m_nEncoderZRange 编码器Z相范围 >1

long m_nOffset 原点偏移量；==0不偏移，>0正方向偏移，<0负方向偏移

**返回值** 0：正确 -1至-8：错误类型,-x表示第x个参数错误

### 14.11.2 设置回零速度参数：

int _SetHomeSpeed_Ex(int m_nCardNum,int m_nAxisNum,long m_nStartSpeed,long m_nSearchSpeed,long m_nHomeSpeed,long m_nAcc,long m_nZPhaseSpeed);

**功能**：回零速度参数

**参数**：

int m_nCardNum 卡号

int m_nAxisNum 轴号

long m_nStartSpeed 原点(STOP0)搜寻起始速度

long m_nSearchSpeed 原点搜寻速度

long m_nHomeSpeed 低速接近原点速度

long m_nAcc 回原点过程中的加速度

long m_nZPhaseSpeed 编码器Z相(STOP1)搜寻速度

**返回值** 0：正确 -1至-7：错误类型 -x表示第x个参数错误

### 14.11.3 启动回零：

int HomeProcess_Ex(int m_nCardNum,int m_nAxisNum);

**功能**：启动回零

**参数**：

int m_nCardNum 卡号

int m_nAxisNum 轴号

**返回值** 0：正确 1：错误

说明 调用该函数时启动回零动作

### 14.11.4 获取回零状态：

int GetHomeStatus_Ex(int m_nCardNum,int m_nAxisNum);

**功能**：获取回零状态

**参数**：

int m_nCardNum 卡号

int m_nAxisNum 轴号

**返回值** 0:回零成功；-1表示参数1错误；-2表示参数2错误；-3表示回零未启动;

1-10表示执行的步骤，分别表示：

1 ：快速接近原点，搜索STOP0

2 ：检查STOP0是否找到

3 ：反向退出原点

4 ：检查反向退出原点是否完成

5 ：低速接近原点，搜索STOP0

6 ：检查STOP0搜索是否完成

7 ：低速接近Z相，搜索STOP1.如果STOP1设置为-1，则跳过7,8两步。

8 ：检查STOP1搜索是否完成

9 ：原点偏移

10 ：检查原点偏移

-100x：回零第x步出现异常，例如-1001表示回零第1步出现异常

-1020：回零被终止

备注：回零过程中需定时调用，直到回零成

# 第十五章 常见故障及解决方案

## 15.1 运动控制卡检测失败

在使用控制卡的过程中，如果遇到检测不到控制卡的现象，可以参照下面的方法逐步进行排查。

(1) 务必按照控制卡安装说明，分步安装好控制卡的驱动程序，确保在系统目录（system32或System）下有控制卡动态库文件；

(2) 检查运动控制卡和插槽接触是否良好。可以通过重插或更换插槽的方式测试，另可用橡皮擦对控制卡的金手指的污诟的清除再装上测试；

(3) 在系统设备管理器中，检查运动控制卡和其它硬件是否有冲突。使用PCI卡时，可以先取下其它板卡，如：声卡，网卡等；PC104卡可以调整拨码开关重新设定基地址，程序中卡初始化时使用的基地址必须和实际基地址相同；

(4) 检查操作系统是否存在问题，可以通过重新安装其他版本的操作系统进行测试；

(5) 按照上面的步骤检查后，如果依然找不到运动控制卡，可以通过更换运动控制卡，进一步进行检测，以便诊断运动控制卡是否已经损坏；

## 15.2 电机运行异常

在运动控制卡正常的前提下，电机出现异常现象时，可以参照下面的情况排除故障。

(1) 运动控制卡发出脉冲时，电机不运动

- 请检查控制卡和端子板的连接线是否接好；
- 电机驱动器的脉冲和方向信号线是否已经正确地连接到端子板；
- 伺服驱动器的外部电源是否已经连接好；
- 伺服、步进电机驱动器是否存在报警状态,如有报警则按报警对应代码检查原因所在；
- 伺服SON是否连接好，伺服电机是否有激磁状态等；
- 如果是伺服电机请检查驱动器的控制方式，本公司的控制卡支持“位置控制方式”；
- 电机、驱动器坏

(2) 步进电机运转时发出异常尖叫声，电机出现明显失步现象。

- 控制器的速度过快，请计算电机的速度，步进电机每秒10~15转

属于正常范围；

- 机械部卡死或机器的阻力太大；
- 电机的选型不够，请更换大力矩型号电机；
- 请检查驱动器的电流和电压，电流设为电机的额定电流的1.2倍，供电电压在驱动器的额定范围；
- 检查控制器起始速度，一般起始速度为0.5~1左右，加减速时间0.1秒以上；

(3) 伺服、步进电机在加工过程中出现明显振动或噪音现象

- 伺服驱动器的位置环增益和速度环增益太大，在定位精度允许的情况下降低服驱动器的位置环增益和速度环增益；
- 机器刚性太差，调整机器的结构；
- 步进电机选型不够，请更换大力矩型号电机；
- 步进电机的速度处于电机的共振区域，请避开此共振区或增大细分的办法来解决；

(4) 电机定位不准

- 请检查机械丝杠螺距和电机每转脉冲数与实际应用系统所设定的参数是否相符，即脉冲当量；
- 如果伺服电机，则增大位置环增益和速度环增益；
- 请检查机器的丝杠间隙，用千分表测试丝杠的反向间隙，如有间隙请调整丝杠；
- 如果是不定时、不定位置的定位不准，则要检查外部干扰信号；
- 电机选型不够在运动中出现抖动或失步现象；

(5) 电机没有方向

- 检查DR+ DR-接线有没有错误，是否接牢；
- 请确定控制卡采用的脉冲模式是否与实际驱动器模式相符，本控制卡支持“脉冲+方向”和“脉冲+脉冲模式”
- 步进电机要检查电机线有没断线、接触不良等现象；

## 15.3 开关量输入异常

在系统调试、运行过程中，某些输入信号检测异常，可以使用下面介绍的方法进行检查。

(1) 没有信号输入

- 据前面讲述的普通开关和接近开关的接线图，检查线路是否正确，确保输入信号的“光藕公共端”已经和内部或外部电源(+12V或24V)的正端相连；
- 本公司的I\O点的输入开关使用NPN型，如果没有请检查开关开

型号和接线方式；

- 检查光藕是否已经损坏。在线路正常的情况下，输入点在断开和闭合的情况下，输入状态不发生改变，可以利用万用表检测光藕是否已经被击穿，通过更换光藕可以解决光藕被击穿的问题；
- 检查开关电源的12V或24V是否正常；
- 开关损坏；

(2) 信号时有时无

- 检查是否存在干扰，可以在I\O测试画面检测信号的状态；如果是干扰情况则增加独石电容型号为104或采用屏蔽线等；
- 机械在正常运行过程中，出现明显的颤抖或异常停止现象，请检查限位开关信号是否存在干扰或限位开关性能是否可靠；
- 外部接线是否接触良好；

(3) 归零不准

- 速度太快，降低归零速度；
- 外部信号存在干扰，请检查干扰源；
- 归零方向错误；
- 归零开关安装位置不当或开关松动；

(4) 限位无效

- 在I\O测试下检测限位开关是否有效；
- 手动、自动加工时速度太快；
- 外部信号存在干扰，请检查干扰源；
- 手动方向错误；
- 限位开关安装位置不当或开关松动；

## 15.4 开关量输出异常

开关量输出异常，可以依据下面介绍的方法进行排查。

(1) 输出异常

- 依据前面讲述的输出点的接线图，检查线路是否正确，确保输出公共端(地线)和所用电源的地线相连；
- 检查输出器件是否已经损坏；
- 检查光藕是否已经损坏，利用万用表检测光藕是否已经被击穿，通过更换光藕可以解决光藕被击穿的问题；
- 安全要领。输出使用感性负载时一定要并联续流二级管，型号IN4007或IN4001；

(2) 输出不良判断方法

断开输出点上对外的接线，在输出点上接一10K左右的上拉电阻到电源端，此时输出的地线需接到电源的GND，并用万用表的红表笔点有12V的正极，黑表笔点在信号输出端同时用手点动测试画面的按钮看是否有电压输出，如果有则检查外围线路，否则检查板卡的公共端是否接好、内部光藕不良等；

## 15.5 编码器异常

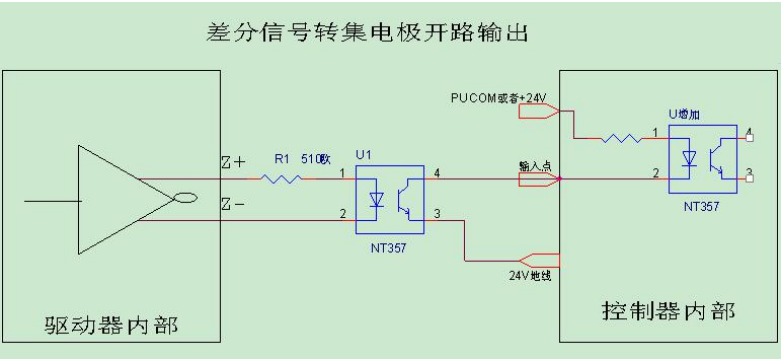
在使用编码器出现异常时，可以参照下面介绍的方法进行排查。

(1) 检查编码器接线。确保编码器的接线符合前面介绍的差动或集电极开路方式；
(2) 检测编码器电压。运动控制卡正常接受的是+5V的信号，如果选用的是+12V或+24V编码器，务必在编码器A、B相和端子板A、B相之间串联1K(+12V)电阻；
(3) 编码器计数不准。编码器的外围接线一定要采用屏蔽双绞线，编码器线不能跟强电等一些干扰较强的电线捆绑在一起，必须分开在30～50MM以上

# 附录

## A 伺服驱动器的Z 相信号差分和控制卡集电极转换

伺服驱动器（或编码器）为差分信号（比如：Z+,Z-），如何接控制卡集电极开路的STOP1信号（ADT-8504,ADT-8948A1,ADT-856, ADT-8960）



# 修改履历（一）

| | | | | | |
|---|---|---|---|---|---|
| 反馈人 | | 反馈日期 | | 当前版本/总页数 | |
| 问题描述 | | | | | |
| 工程师确认 | | | | | |
| 修订后版本 | | 修订后总页数 | | 修订人 | |