6050 电机控制卡

使用手册

第一章 硬件指南 1

§1.1 **公司简介** 1
§1.2 **步进电机控制概述** 1
§1.3 **全数字伺服电机控制概述** 2
§1.4 6050 **系列电机控制卡性能指标** 4
§1.5 **控制卡的安装** 6
§1.6 **控制卡的硬件连线** 6
§1.6.1 ***电机控制信号*** *8*
§1.6.1.1 ***轴脉冲输出模式*** *8*
§1.6.1.2 ***轴方向*** *10*
§1.6.2 ***通用数字*** *IO* ***信号*** *10*
§1.6.3 ***编码器输入信号*** *12*
§1.6.4 PWM/DA ***输出信号*** *13*

第二章 软件指南 15

§2.1 **一般说明** 15
§2.2 **软件的安装** 15
§2.3 **编译与联接（**COMPILING AND LINKING**）** 16
§2.3.1 Microsoft Visual C++ *16*
§2.3.2 Microsoft Visual BASIC *16*
§2.3.3 Borland Delphi *17*
§2.3.4 Borland C++ Builder *17*
§2.4 **库函数介绍** 17
§2.4.1 ***初始化函数*** *17*
§2.4.2 ***单轴运动控制函数*** *18*
§2.4.3 ***轴随动运动控制函数*** *20*
§2.4.4 ***插补函数*** *21*
§2.4.5 ***插补疑问*** *23*
§2.4.5 ***其它说明*** *31*
§2.4.6 ***库函数一览表*** *35*

第三章 库函数详解 39

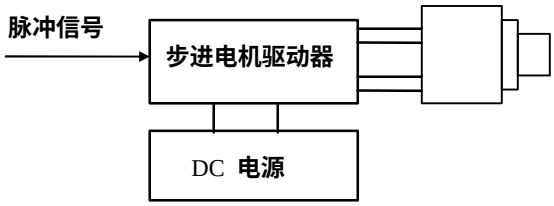
第一章 硬件指南

§1.1 公司简介

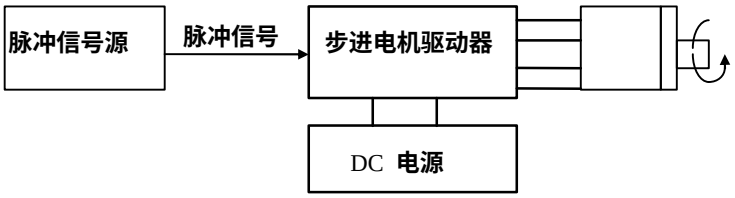
我公司是一家设计、制造和销售工业控制产品及系统的专业公司。我们的业务重点在于电机控制。通过我们的产品和设计，我们可帮助用户解决复杂的电机控制问题。从产品的概念设计到工程技术的具体实施，我们的工程师都能为广大用户提供非常专业的技术咨询、技术支持及技术服务。

§1.2 步进电机控制概述

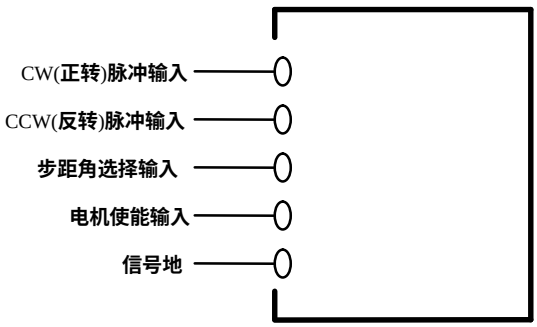
步进电机系统可分为两大部分，即步进电机和驱动器。



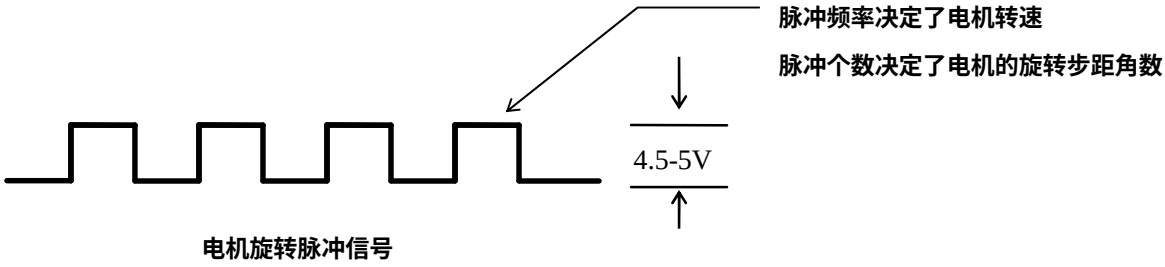
在上图中控制系统还不完整，还要有脉冲信号源整个步进电机系统才能运转起来。



在实际中，步进电机驱动器要求的控制信号要复杂一些，举例如下：

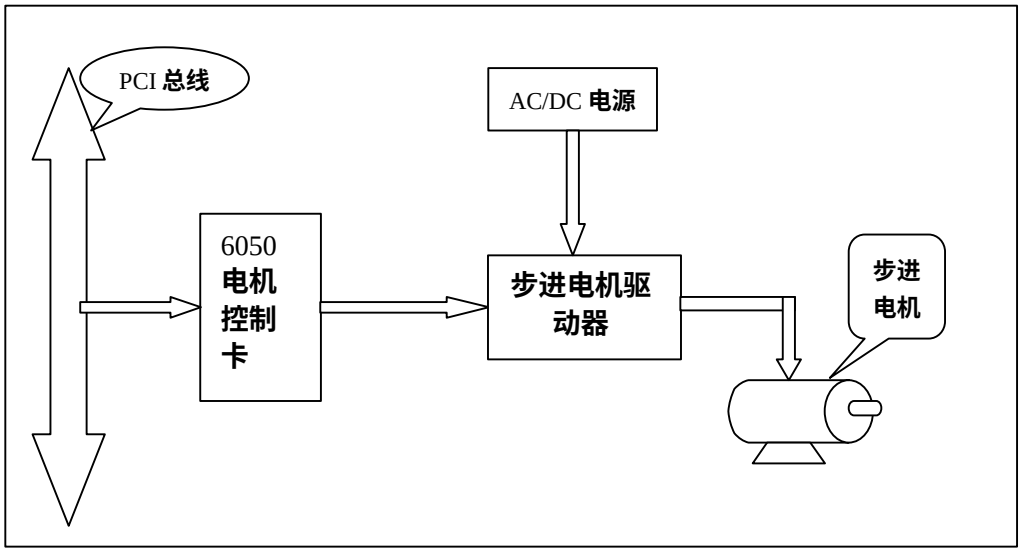


驱动器要求的脉冲信号一般为 TTL 电平兼容的方波信号，而步距角选择和电机使能信号为 TTL 电平信号。



用普通脉冲频率发生器并不能很好的控制步进电机，根据应用需要，用户可用 6050 **系列板卡组成更复杂的步进电机控制系统，如下图：**

步进电机控制



上位机：**上位机（**PC **机）通过控制软件对电机控制卡进行读写操作，可向控制卡发送位置、速度、加速度命令。**

步进电机控制卡：**控制卡根据上位机的命令产生脉冲序列，所产生的脉冲个数、频率及频率变化率即为为控制电机的位置、速度及加速度指令。**

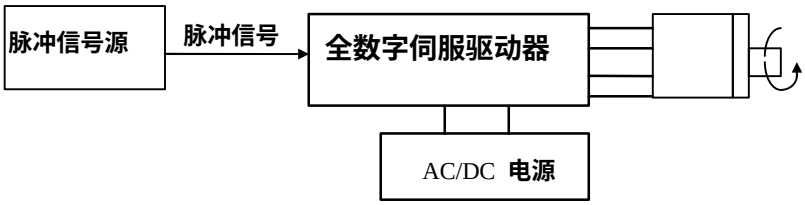
步进电机驱动器：**步进电机驱动器根据接收到的脉冲信号，产生多拍节脉冲驱动信号控制步进电机旋转。**

应用范围：**步进电机控制系统比伺服系统要经济，它适用于不需要位置反馈的场合，它主要应用在经济型系统中。**

§1.3 全数字伺服电机控制概述

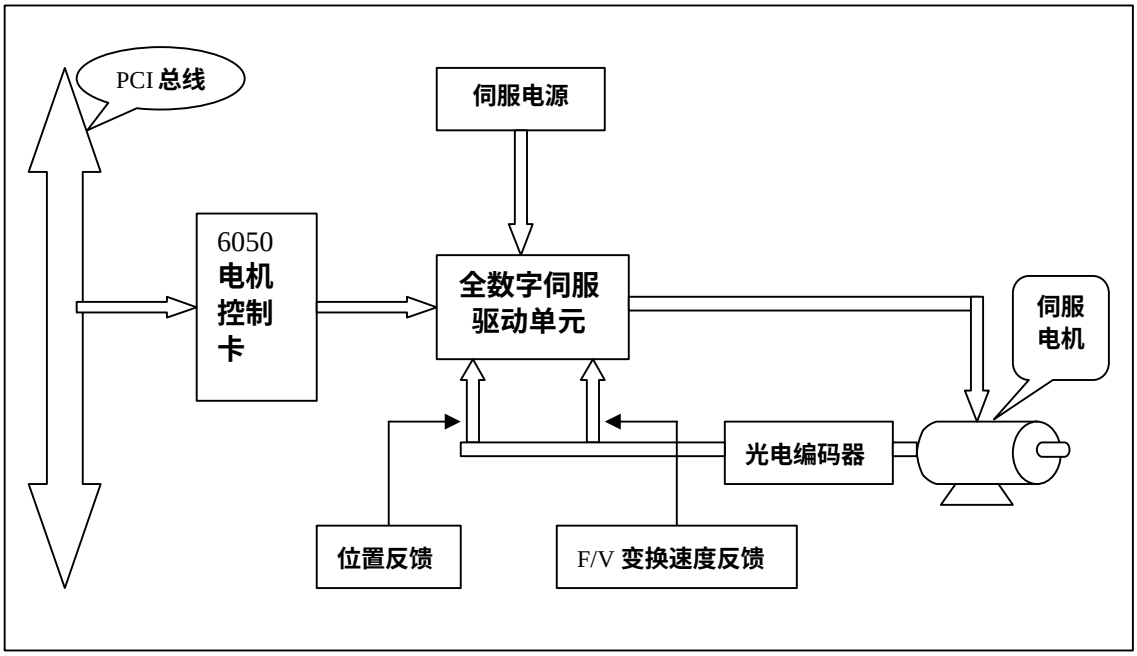
全数字伺服控制系统类似于步进电机控制系统，它也可分为两大部分，即：伺服电机和全数字伺服驱动器。

在上图中加上脉冲信号源整个控制系统才算完整：



用 PC 机控制全数字伺服电机框图如下：

伺服电机控制



上位机：**上位机（PC 机）通过控制软件对电机控制卡进行读写操作，并向控制卡发出位移、速度、加速度等命令。**

电机控制卡：**控制卡根据上位机的命令产生脉冲序列，所产生的脉冲个数、频率及频率变化率即为控制电机的位置、速度及加速度指令。**

伺服驱动单元：**伺服驱动单元根据控制卡的位置命令值减去位置反馈值来算出电机位置误差，位置误差值经过驱动单元的数字滤波器（PID 调节算法）产生电机速度控制信号，速度控制信号经驱动单元内的电流环等环节产生驱动电流，对伺服电机进行调速。**

伺服电机：**增量编码器是伺服电机的典型的反馈元件，它将电机的旋转角度转换为正交的电脉冲信号，伺服驱动单元根据该反馈信号就能跟踪电机的旋转位置，从而组成伺服电机的闭环控制系统。**

***应用范围：*伺服电机的应用范围非常广泛，它主要应用在数控机床、工业机器人、包装机械、纺织机械、印刷机械、橡胶机械等领域。总之，凡需要进行精确位置控制的场合一般都可用伺服电机实现。**

§1.4　6050 系列电机控制卡性能指标

6050 系列电机控制卡的特性列表如下：

主卡硬件结构

- FPAG + DSP 结构，FPGA 选用 Altera 公司 ACEX1K50，DSP 选用 TI 公司 TMS320LF/LC2812，DSP 主频：150MHz

主机界面

- 33M　PCI 总线

推荐工作环境

- 0-70℃
- %20-90 相对湿度，无凝结

电机控制

- 4 轴脉冲信号输出；频率范围：1-8,000,000 steps/sec.
- 高电平 4.5-5V,驱动电流最大 50mA

运动曲线

- 梯形包络线
- S 形加减速
- 速度曲线
- 电子齿轮
- 4 轴直线插补
- 4 轴圆弧插补（任意 2 轴圆弧+2 轴直线，即螺旋线插补）
- 轴位置按比例随动编码器（螺纹切削或硬攻丝）

运动范围

- 位置寄存器：32 位 -2147483648 至 2147483647
- 插补向量长度: <1073741823　　(直线长度和圆弧长度)
- 圆弧最大半径: <1048575(用户单位)

通用 IO 及其它

- 20 路光隔输入 DC12-24V
- 8 路光隔/达林顿放大输出，最大驱动电流 500mA
- 2 路继电器输出： 1.A 24 VDC / 120VAC
- 1 路 PWM 脉冲/12 位 DA 模拟量输出：PWM：频率范围：18-150000Hz, DA 输出：分辨率 12 位，-10 VDC 至 +10 VDC
- 1 路光电编码器输入： 差分/单端 A、B、Z 三相，4 倍频率。

继电器型号：HKE (汇港继电器)HRS2(H)

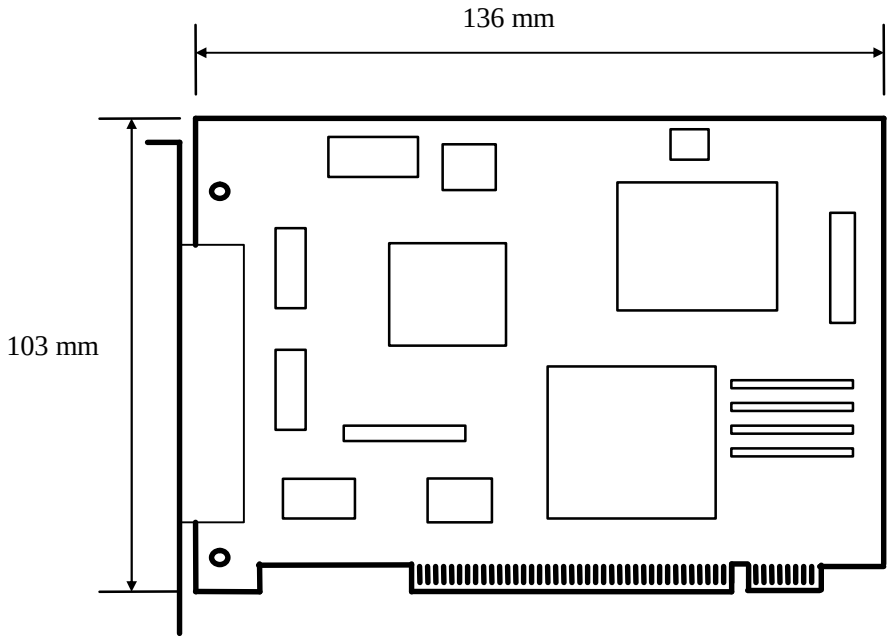
. 继电器耐久性(平均寿命)	电气寿命	100,000 operations at nominal load
		20,000,000 operations at 0.5A 120VAC

		500,000 operations at 1A 24VDC
	机械寿命	1,000,000 operations at 0.15A 48VDC
		10,000,000 operations

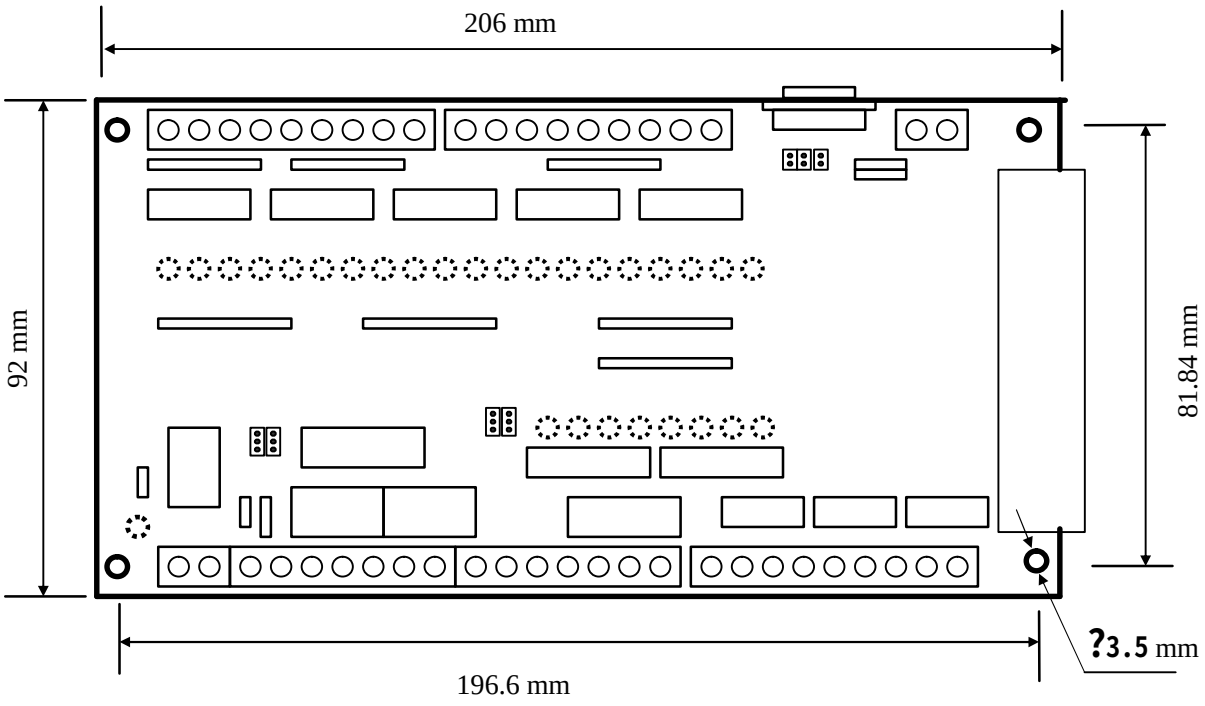
软件形式

·用户函数库，支持 VB、VC、Delphi **及** C++ Builder

6050 控制卡主板外形尺寸



端子板外形尺寸



§1.5 控制卡的安装

完整的 6050 卡它包含以下内容：

① 6050 PCI 主板一块
② 6050 端子板一块
③ 2 米长 68 芯电缆一根
④ 光盘一张
⑤ 6050 使用手册一本

在安装控制卡前，请确保计算机是处于关闭电源状态，然后再将 6050 卡插入计算机主板上某一空闲的 PCI 插槽中，上紧固定螺钉，通过 68 芯电缆正确连接 6050 卡与端子板。具体安装步骤如下：

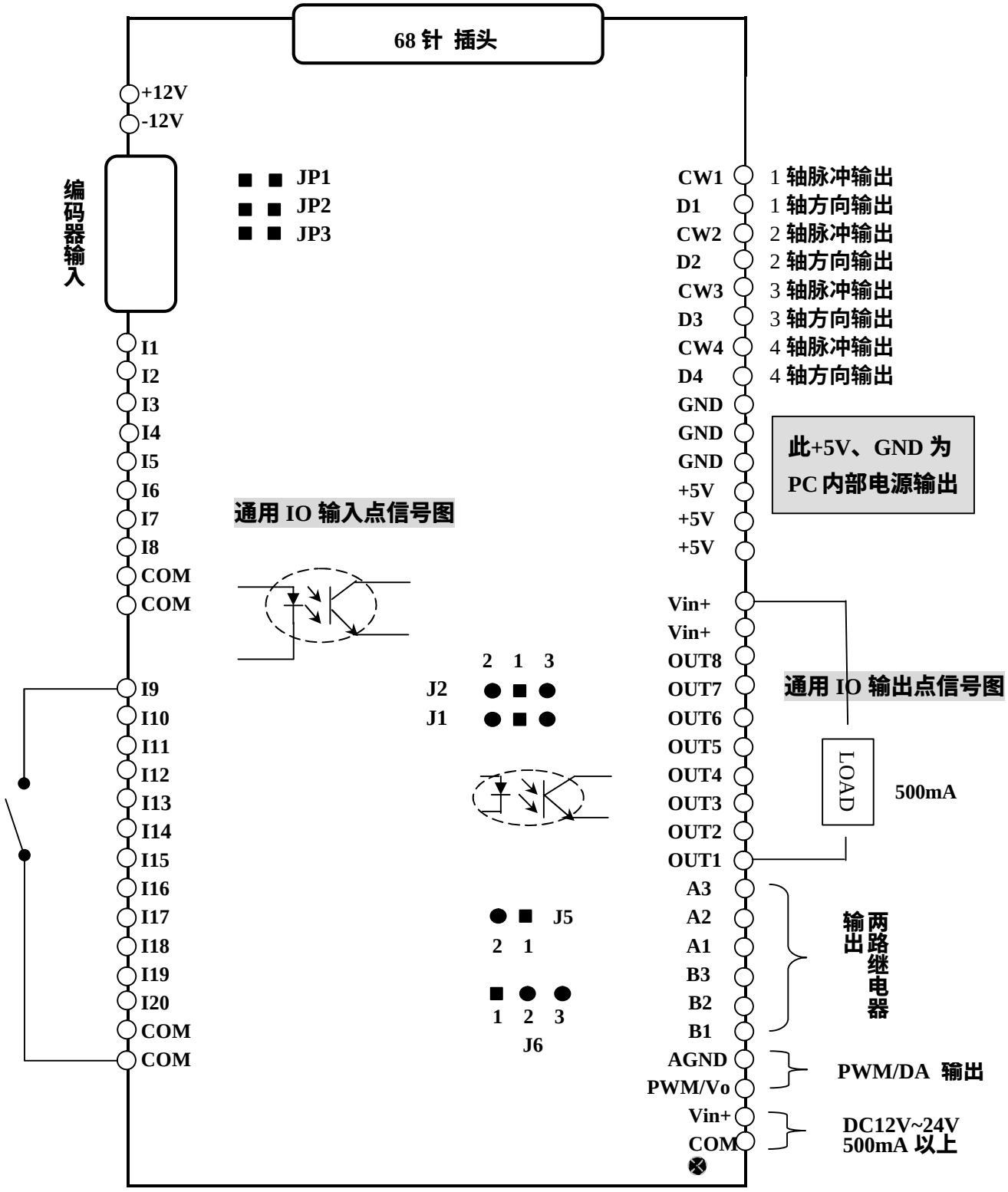
①关闭计算机电源
②打开计算机箱盖，将 6050 主板插入计算机某一空闲的 PCI 插槽中，上紧固定螺钉
③固定好端子板，用 68 芯电缆将 6050 主板与端子板连接牢靠。
④关闭计算机箱盖，打开计算机电源
⑤安装 6050 的驱动程序

6050 卡的驱动程序支持 Win9x、Win2000、WinXp、WinVista，在安装驱动程序时请在光盘的 6050\driver 文件夹中寻找。

§1.6 控制卡的硬件连线

6050 卡通过端子板展开其硬件连线，在本节中将着重介绍有关 6050 端子板硬件连线原理。

6050 端子板接线图



在上图中：

* COM 为外部电源的地（GND）
* Vin+ 接外部电源 12V-24V

* GND & +5v、+12、-12 、AGND　　是从 PC 机内部引出的电源
* CW1、D1、CW2、D2、CW3、D3、CW4、D4　是电机的控制信号,由 PC 机内部电源供电
* IN1 - IN20　　是 20 个通用 IO 输入点
* OUT1 - OUT8　　是 8 个通用 IO 输出点
* PWM/VO、AGND　　是 PWM 或 DA 模拟电压输出，由 PC 机内部电源供电

如果用户要用到 20 个通用 IO 输入点或 8 个通用 IO 输出点，必须在 COM 与 Vin+之间外接 12V-24V 电源。

§1.6.1 电机控制信号

表 1.　电机控制信号

管腿名称	定义	解释
CW1	0 轴脉冲输出（CW）	高电平 4.5-5V,驱动电流最大 50mA，可直接打开光电偶合器
D1	0 轴方向控制/或反向脉冲输出（CCW）	高电平 4.5-5V,驱动电流最大 50mA，可直接打开光电偶合器
CW2	1 轴脉冲输出（CW）*	（同上）
D2	1 轴方向控制/或反向脉冲输出（CCW）	（同上）
CW3	2 轴脉冲输出（CW）*	（同上）
D3	2 轴方向控制/或反向脉冲输出（CCW）	（同上）
CW4	3 轴脉冲输出（CW）*	（同上）
D4	3 轴方向控制/或反向脉冲输出（CCW）	（同上）

§1.6.1.1 轴脉冲输出模式

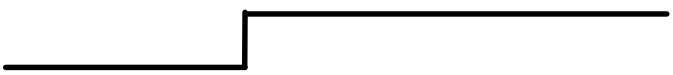
通过软件（函数 SetAxisOutMode_6050 ），每个轴的驱动信号可设两种输出方式为：

CW（正转）+ CCW（反转）　或
CW（正转）+ 方向控制信号

如设为 脉冲/方向 方式则：CW 端为脉冲输出，D 端为方向输出如下：

CW 端：

D 端：



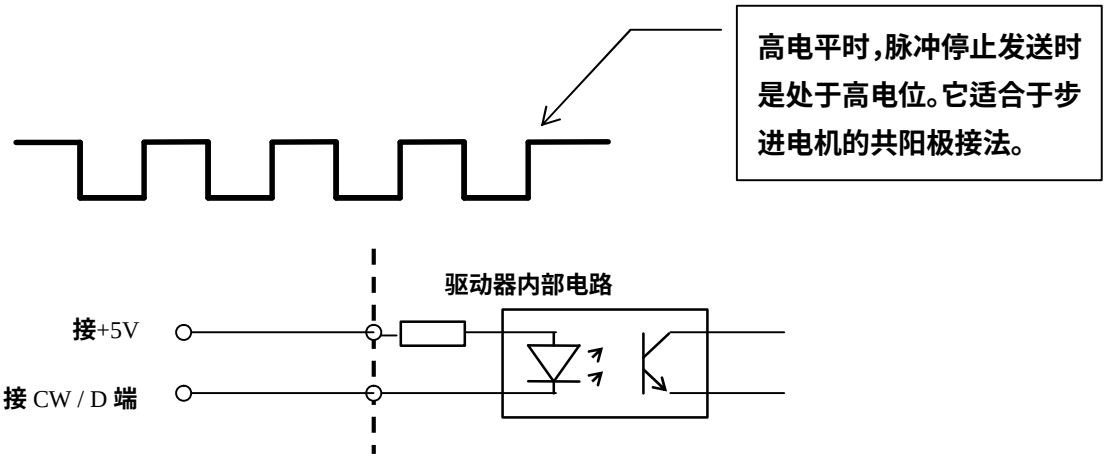
如设为 正向脉冲/反向脉冲 方式，则：CW **端为正向脉冲（**CW**）输出，**D **端为反向脉冲（**CCW**）输出如下：**

CW **端：**

D **端：**

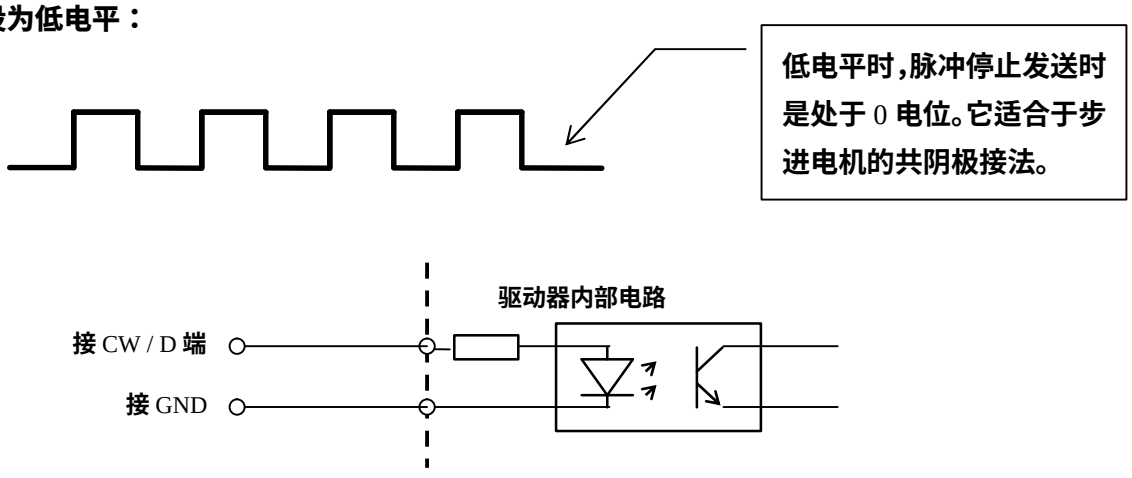
无论设为哪种方式，输出脉冲电平的高低也可用软件（函数 SetAxisOutMode_6050**）来设定。**

A．设为高电平：



共阳极接法

B．设为低电平：



共阴极接法

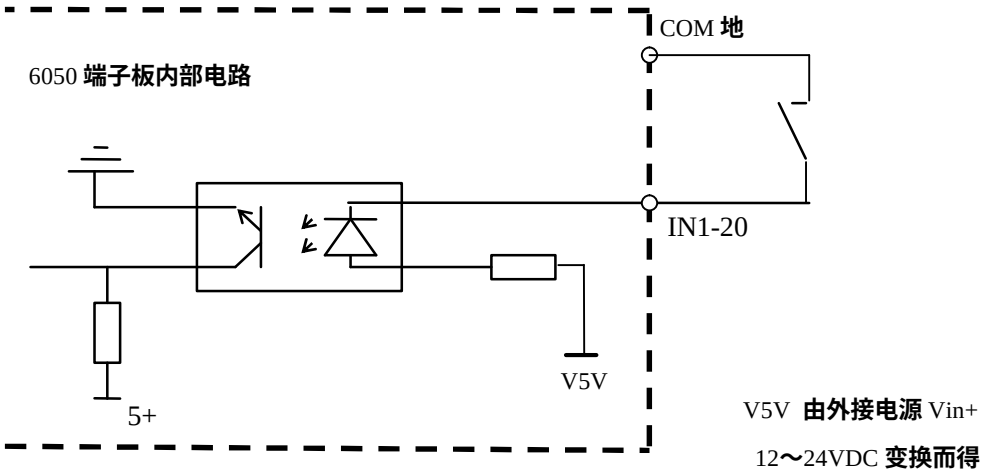
§1.6.1.2 轴方向

轴方向的改变（电机正转或反转），一种方法是改变硬件连线，还有就是通过软件（函数 SetAxisOutMode_6050 **），改变轴的方向：**

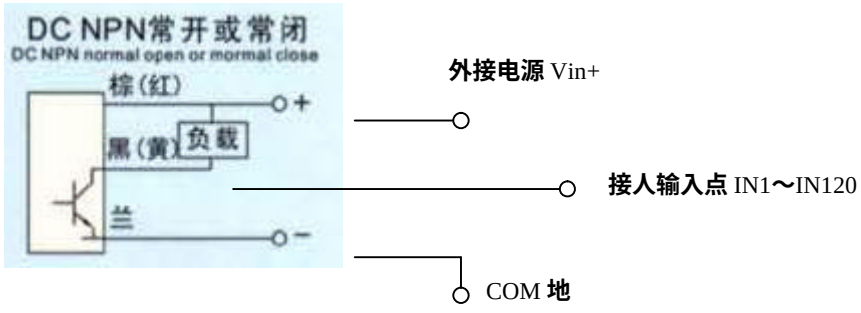
在轴脉冲输出模式为：CW（正转）+ **方向控制信号 时，它改变** ***方向控制信号*** **的高低电平；在轴脉冲输出模式为：**CW（正转）+ CCW（反转） **时，它改变** ***CW（正转）、CCW（反转）信号*** **的输出次序。**

§1.6.2 通用数字 IO 信号

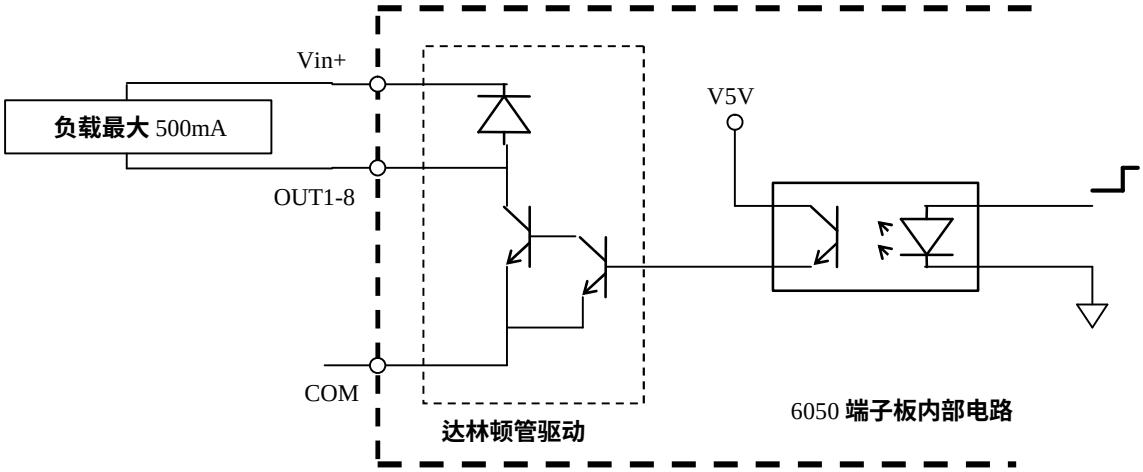
输入点原理图



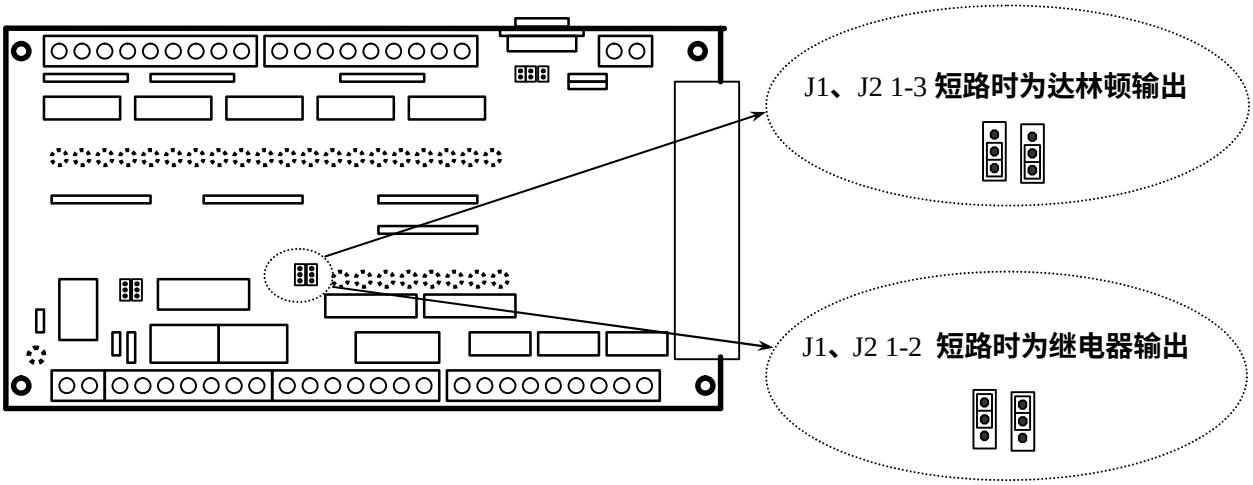
当输入点是 光电开关类型的传感器时，请选用 NPN **型。 接线原理图如下：**



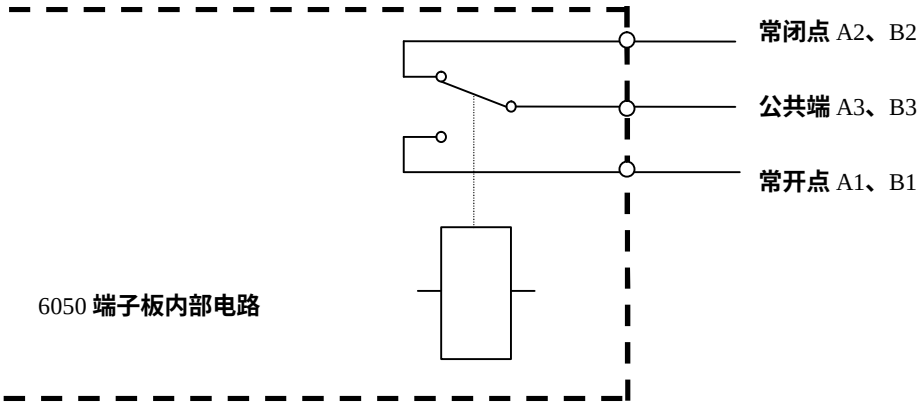
输出点原理图



在端子板上有 J1、J2 **跳线，当** J1、J2 **的** 1-2 **短路时，输出点** 1、2 **为 继电器输出；**1-3 **短路时为达林顿输出。如下图**:



继电器输出原理图：



有关 IO **操作的函数：**

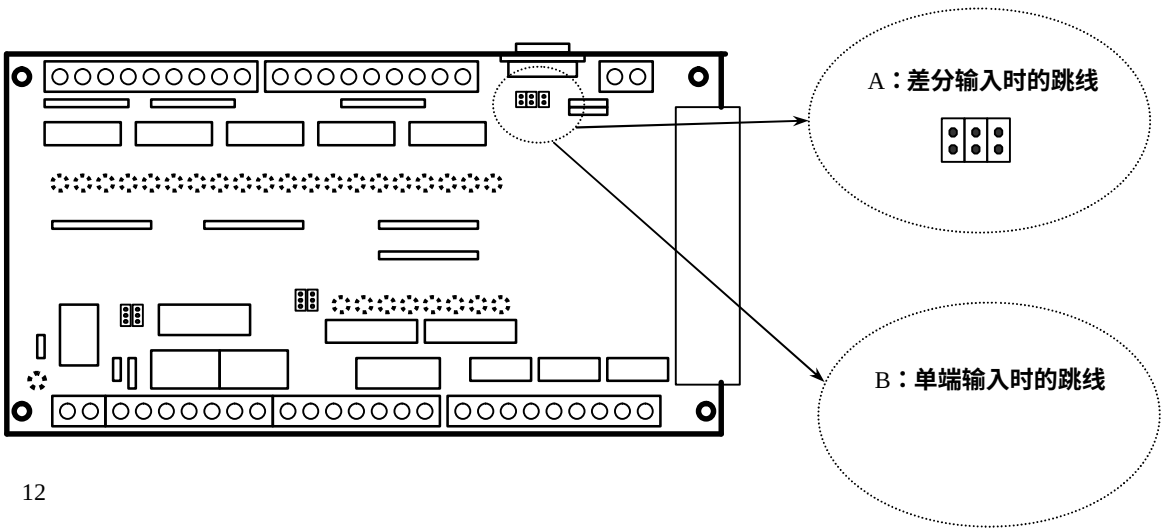
编号	函数名	描述
14	ReadIO_6050	读取通用输入点 I1-I20 状态
14	ReadIOBit_6050	按位读取通用输入点 I1-I20 状态
16	WriteIo_6050	设置通用输出点 O1-O8 状态
81	WriteIoBit_6050	按位设置通用输出点 O1-O8 状态
82	ReadOs_6050	读取输出点 O1-O8 状态
83	ReadOsBit_6050	按位读取输出点 O1-O8 状态
29	Set_Emergency_Stop_6050	设置 IO 急停信号
15	SetAxisIO_6050	设置轴限位、零位（Home 点）IO 是否有效
62	LM_IOOut_6050	插补模式的 IO 点输出
63	LM_Wait_I_6050	插补等待通用 IO 点输入

§1.6.3 编码器输入信号

表 2. 编码器输入接口 DB9 针连线表

管腿名称	定义	解释
1	A+	接编码器 A+
6	A-	接编码器 A- （编码器单端输入 可不接）
2	B+	接编码器 B+
7	B-	接编码器 B- （编码器单端输入 可不接）
3	Z+	接编码器 Z+ （可不接）
8	Z-	接编码器 Z- （可不接）
4	+5V	PC 机+5V 电源
5	+12V	PC 机+12V 电源-
9	GND	PC 机电源地

编码器信号是单端输入时，用户应该将端子板上 JP1、JP2、JP3 跳线跨接，



光电编码器的供电

当光电编码器是差分输入时，光电编码器的电源可由外部供电，也可由 6050 **板内供电，如果由** 6050 **板内供电可直接从** DB9 **针插头中引出**+5V **或**+12V **电源。**

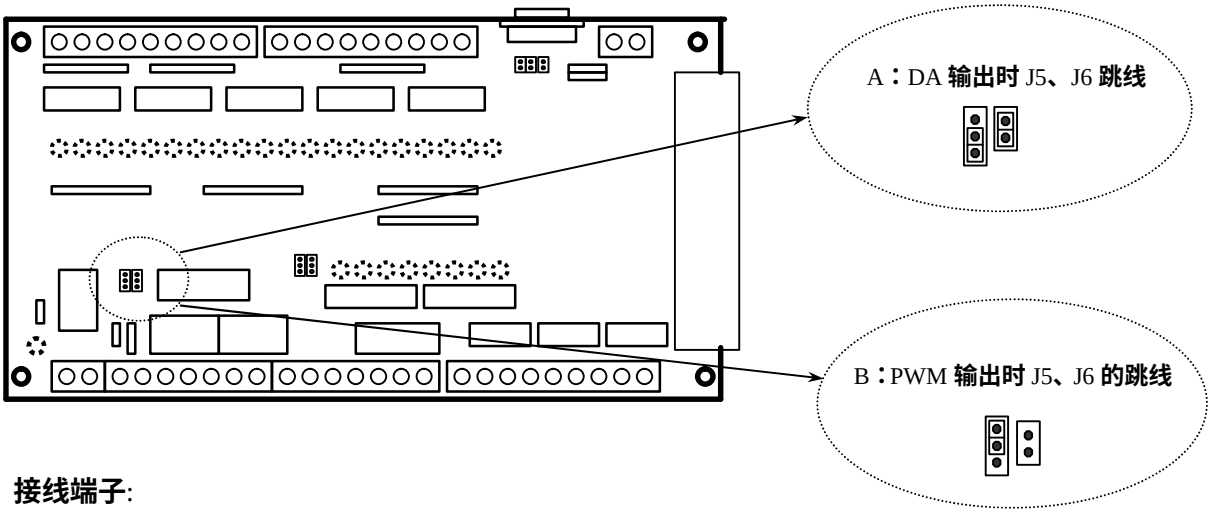
当光电编码器是单端输入时，光电编码器的电源只能由 6050 **板内电源供电。**

有关编码器的函数：

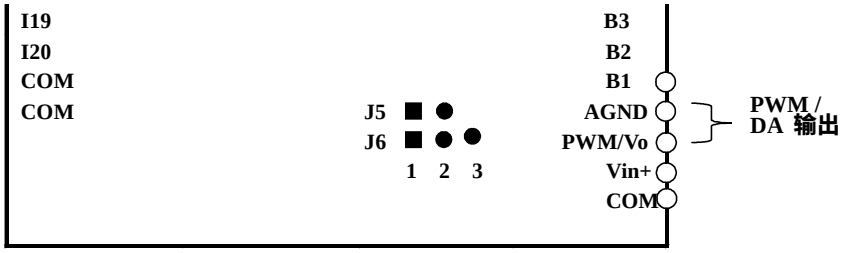
编号	函数名	描述
10	ReadEncoderPos_6050	读取编码器位置
11	HomeEncode_6050	复位编码器,编码器位置清零
74	SetAxisFE_6050	设置轴跟随编码器运动
77	LM_LineFE_6050	直线插补 跟随编码器位置
80	GetEnAuto0Flag_6050	获得编码器自动清零标志

§1.6.4 PWM/DA 输出信号

6050 **有一路** PWM **或** DA **模拟量输出，通过端子板上的跳线** J5、J6 **来选择是** PWM **输出还是** DA **输出。如下：**



接线端子:



PWM **信号由** TI2812DSP **内部产生，其产生原理是：**PWM **的载波频率由** DSP **的主频**

（150MHz）进行 1-128 分频，形成 PWM 的基频，PWM 发生器中有两个 16 位的计数器，这两个 16 位计数器再对基频进行分频，一个产生载波频率、一个产生占空比，所以本卡所产生的 PWM 脉冲最低频率为：150000000÷126÷65536=17.88≈18Hz。在 DA 模拟量输出模式下 PWM 的载波频率固定为 25KHz，所以本卡 DA 输出的分辨率为：1/6000，6000=150MHz/25KHz。

有关 PWM/DA 输出的函数：

编号	函数名	描述
12	PwmOut_6050	PWM 脉冲输出
12	PwmOut2_6050	PWM 脉冲输出
13	PwmStop_6050	PWM 脉冲停止输出
12	DAOut_6050	DA 模拟电压输出
61	LM_PWM_6050	插补模式的 PWM 脉冲输出

第二章 软件指南

§2.1 一般说明

在随 6050 卡提供给用户的光盘中可找到以下文件夹：

[光盘]：------6050
|
|-----Driver
|-----Setup
|-----WinCNC

在 Driver 中提供了 6050 的驱动程序，包括 Win9x、Win2000、WinXp 三种平台。

在 Setup 中提供了 Windows 32 位 动态连接库 DLLs，支持 Microsoft Visual BASIC 、Visual C++及 Borland Delphi、C++ Builder。还有一些测试及示例程序。

在 WinCNC 中提供了基于 6050 卡的 CNC 机床数控软件 WinCNC for 6050 Demo（演示版）。

§2.2 软件的安装

在光盘的 6050\Setup 文件夹中运行 setup 程序即开始安装软件。setup 程序将提示安装目标盘和安装目录，该程序的省却安装目录为：

C:\6050

在该目录下，分别有 6050\VB、6050\VC、6050\Delph、6050\Bcb 子目录，它们分别对应 Microsoft Visual BASIC、Microsoft Visual C++、Borland Delphi、Borland C++ Builder 等编程环境。在这些目录下有以下文件：

表 2.1 6050\VB 文件一览表

文件名	解释
Declare.bas	VB 窗体函数声明文件
6050Test.EXE	6050 卡测试软件
6050Test	6050 卡测试软件的 VB 源程序

表 2.2 6050\VC 文件一览表

文件名	解释
dfjzh6050dll.dll	VC 动态联接库
dfjzh6050dll.lib	VC 联接库
dfjzh6050dll.h	VC 函数声明文件
6050Demo.EXE	6050 卡演示软件
6050Demo	6050 卡演示软件源程序

表 2.3　6050\Delphi 文件一览表

文件名	解释
Dfjzh6050Api.DLL	Delphi 动态联接库
drv6050.pas	Delphi 函数声明文件

表 2.4　6050\Bcb 文件一览表

文件名	解释
dfjzh6050dll.dll	C++ Builder 动态联接库
dfjzh6050dll.lib	C++ Builder 联接库
dfjzh6050dll.h	C++ Builder 函数声明文件
dfjzh6050dyn.h	C++ Builder 函数声明文件，动态调用 Dll 的头文件
Example	C++ Builder 例子文件夹

§2.3 编译与联接（Compiling and Linking）

这一节将说明如何使用 6050 的函数库，该函数库是以 Windows 32 位动态连接库 DLLs 的形式提供给用户的。

§2.3.1　Microsoft　Visual C++

用户在用 VC 编写程序时，首先将 6050 动态联接库 dfjzh6050dll.dll 拷贝到 Windows 系统目录，比如 C:\ Windows\System，或拷贝到 VC 的工作目录。然后将 dfjzh6050dll.lib 及头文件 dfjzh6050dll.h 包含在工程项目中，这样用户就可以使用 6050 的库函数了。

§2.3.2　Microsoft　Visual BASIC

用户在用 VB 编写程序时，首先将 6050 动态联接库 Dfjzh6050Api.dll 拷贝到 Windows 系统目录，比如 C:\ Windows\System，或拷贝到 VB 的工作目录。然后在窗体的通用说明区对 6050 库函数进行说明，这样用户就可调用 6050 的库函数对 6050 卡进行编程。

§2.3.3 Borland Delphi

用户在用 Borland Delphi 编写程序时，可按照如下步骤操作：

1）把 dfjzh6050Api.dll 拷贝到 Windows 系统目录，比如 C:\ Windows\System，或拷贝到当前工作目录；

2）把 drv6050.pas 文件拷贝到工作目录；

3）打开 delphi 编辑环境，打开应用程序项目文件；

4）打开 delphi 编辑环境中的“Project”菜单，点击“Add to project...”，把 drv6050.pas 文件添加到应用程序的项目文件中；

5）调出应用程序项目文件中使用 6050 卡工作的单元，点击“File”菜单，用该菜单下的“use unit ...”把 drv6050.pas 单元包括进去；

6）然后就像使用一般的系统库函数一样使用 dfjzh6050Api.dll 中的函数即可。

§2.3.4 Borland C++ Builder

在我们提供的 C++ Builder 应用 6050 的例子中，采用动态调用 Dll 的方法，C++ Builder 动态调用 Dll 的步骤如下：

1）从新定义函数。 我们已做，用户只需包含 dfjzh6050dyn.h 头文件即可。

2）载入 Dll，用 LoadLibrary 函数。

3）获取函数地址，用 GetProcAssress 函数。

4）强制类型转换，即将所获取的函数地址强制转换为函数。

5）函数调用。

6）释放 Dll。

C++ Builder 动态调用 Dll 稍显复杂，更详细说明请看 C++ Builder 参考书（比如：《C++ Builder 编程技巧、经验与实例》人民邮电出版社）。

C++ Builder 静态调用 Dll 的步骤与 VC 类似，在此不再赘述。

§2.4 库函数介绍

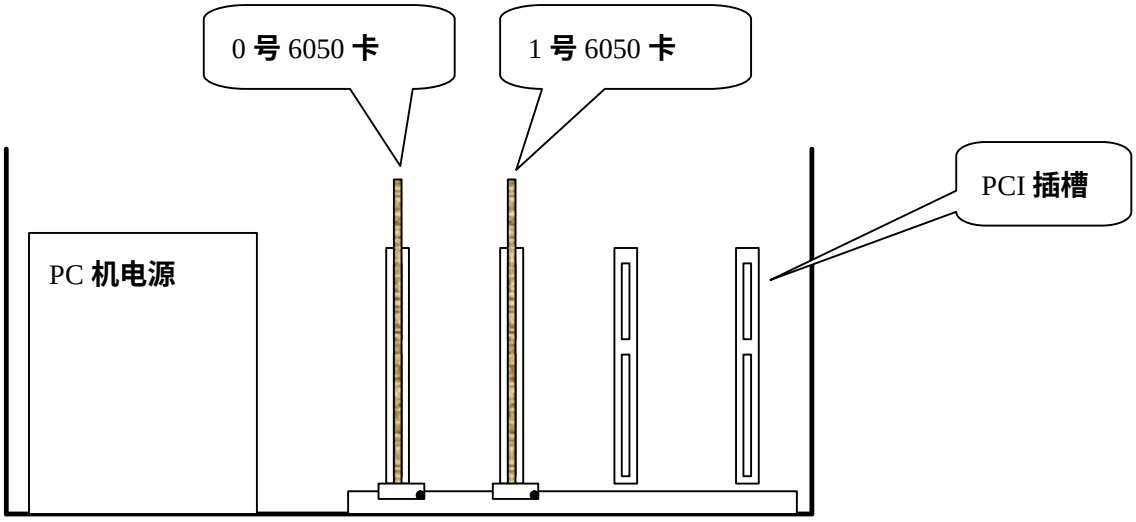
6050 驱动函数库中总共有 60 多个函数，这些函数可粗分类为初始化函数、单轴运动控制函数、检测函数及插补函数。每个函数的详细介绍在本书的第三章中给出。

在用户编制自己的程序时，我们建议用户参考我们所提供的示例程序，这些示例程序将使你省去很多例行工作。

§2.4.1 初始化函数

在调用函数库的各个函数之前必须对函数库进行初始化。初始化函数为 InitCrad_6050，该函数的第一个参数为板号，其它参数见第三章中详细说明。用户如果用到多个 6050 板，则

每个板都必须初始化。 6050 的驱动程序最多支持 16 个轴，用户最多可同时使用 4 块 6050 板，板号为：0-3。



当在一台计算机中用到多块 6050 卡时，6050 卡的板号由其插槽位置决定，越靠近电源的 6050 卡其板号越小，远离的卡其板号越大。

在用户程序退出前，必须调用 ExitCard_6050()函数，释放函数库。当用到多个板时要一一释放，其释放次序是：最先初始化的最后释放，最后初始化的最先释放。

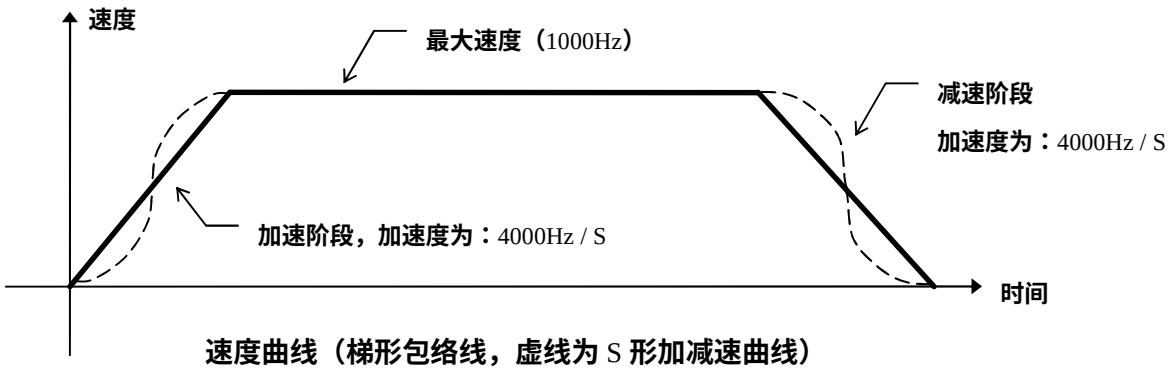
§2.4.2 单轴运动控制函数

6050 板提供了轴的两种基本运动形式，即轴的点到点的运动控制（位置模式）和速度控制（速度模式）。

位置模式时，当用户在程序中写下如下语句：

```
SetAxisAcc_6050(0,0,4000);
SetAxisVel_6050(0,0,1000);
SetAxisPos_6050(0,0,xPosSet);
StartAxis_6050(0,0);
```

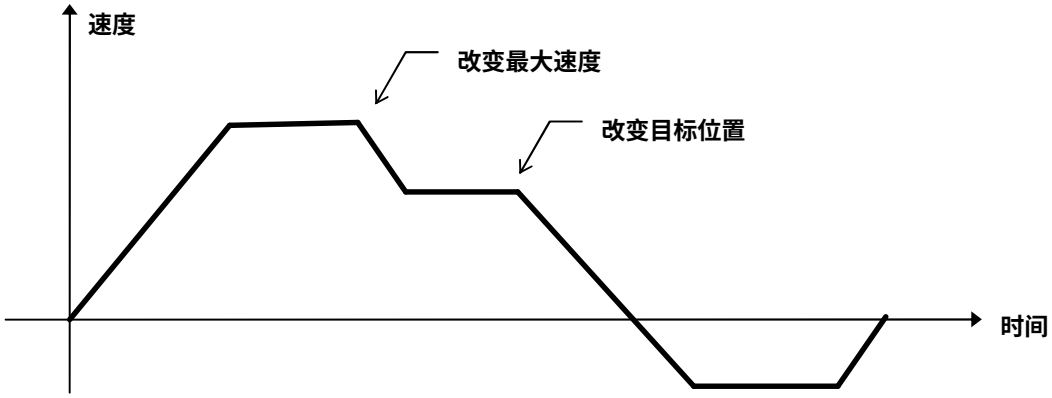
0 号轴将以最大 1000Hz 的频率发 2000 个脉冲，速度曲线如下：



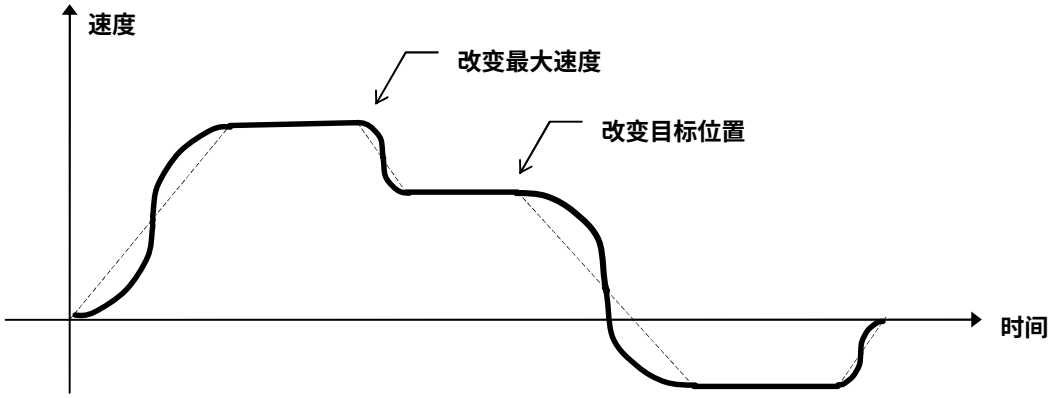
速度曲线（梯形包络线，虚线为 S 形加减速曲线）

上图为梯形包络线，它的加减速值相等。该梯形的面积就是电机转动的距离（2000 个步距角）。由于步进电机所带负载有一定的机械惯性质量，速度不可能立刻达到设定值，所以应有加减速阶段。

在电机运行时，可随时改变梯形包络线的参数，可走出更复杂的运动轨迹，如下图所示：



复杂的梯形包络线

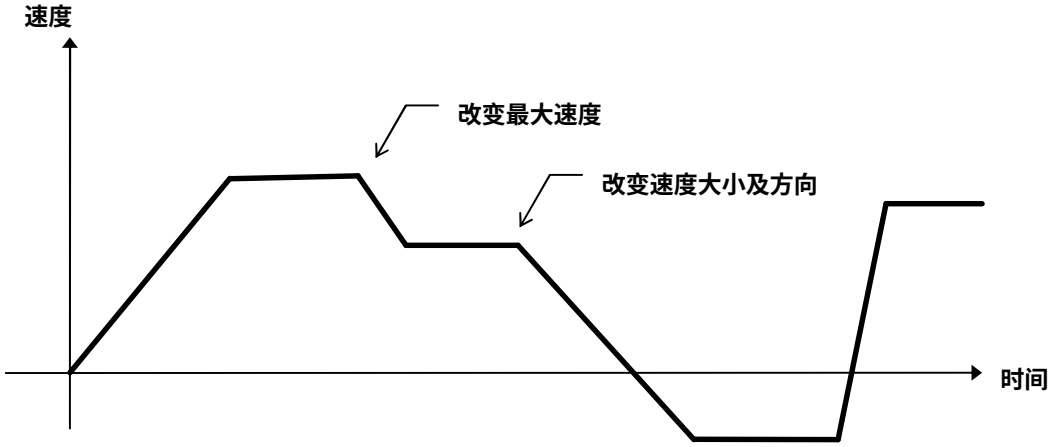


复杂的梯形包络线(S 形加减速)

用户也可以用速度模式来控制电机轴运动，速度模式函数为：

short StartAxisVel_6050 (unsigned short Board_NO,unsigned short Axis_No,long velocity);

该函数的第一个参数为卡号 0、1、2……，第二个参数为轴号 0、1、2……，第三个参数为速度参数，单位为脉冲频率（Hz 脉冲数/秒），可设正负。速度模式下的电机速度曲线如下：



速度模式曲线

速度模式的曲线类似于位置模式，但它没有终点值，用户可随时用轴停止命令函数（AbortAxis_6050、StopAxis_6050、CeaseAxis_6050）命令电机停止运行。在实际编程当中，用户可随时改变步进电机为位置模式或为速度模式。

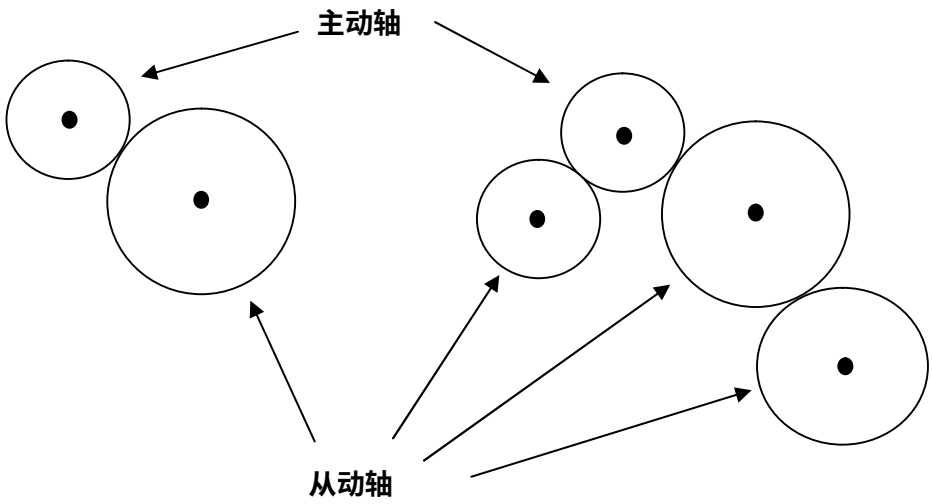
§2.4.3 轴随动运动控制函数

6050 板提供了轴的两种两种随动运动形式，即从动轴随动主动轴（电子齿轮）、轴随动编码器（编码器控制模式）。

在电子齿轮模式下，通过函数：

SetAxisEGear_6050(Board_NO, Axis_No,F_Axis_No,Rate,Kp)；

来设定主从关系、齿轮比等。主从关系可随意设定，只要不违反自己随自己或父子孙的伦理关系就可以。齿轮比很广，在 0.001 至 1000 之间，可正负。电子齿轮随动关系，举例如下图：



在编码器控制模式下，通过函数：

SetAxisFE_6050(Board_NO, Axis_No,Rate,Kp, Mode1, Mode2,Mode3)；

来设定轴跟随编码器，可以是速度方式或位置方式等。另外，在插补模式下也可设轴跟随编码器，这为用户编写带螺纹切削功能的数控系统提供方便。

§2.4.4 插补函数

在实际应用中，有可能需要几个电机轴按一定轨迹同时运动，最典型的是数控机床应用，比如：在数控铣床中的 XYZ **坐标系中，要求机床的** XYZ **轴运行一段直线或一段圆弧或一段空间螺旋线等，这种** ***几个电机轴按一定轨迹同时运动*** **的方式就是所谓的插补运动。**

在 6050 **卡的插补引擎中虚拟了四个轴，即** X、Y、Z、W **轴，组成一个** XYZW **坐标系。这四个虚拟轴如何对应卡上的四个实际轴完全由用户控制，而且它们的编程单位是按实际应用单位实现的（即：用户单位，由用户指定是 毫米、 还是角度或是角弧度等）。针对数控机床的实际应用，**6050 **卡对插补函数进行了如下优化：**

1. **虚拟坐标轴，用户可方便组成** 2 **轴、**3 **轴或** 4 **轴数控系统**
2. **用户单位编程，编程直观，中间单位转换的计算量最小。**
3. **绝对值编程，没有累计误差和计算误差**
4. **小线段连续插补算法先进，在曲线或曲面加工时有显著优势。**
5. **实现不同平面的圆弧插补和螺旋线插补。**
6. **实现** 4 **轴线性和** 4 **轴圆弧（**2 **轴直线；既螺旋线）插补。**
7. **实现插补模式下的轴位置跟随编码器（螺纹切削、硬攻丝）。**

6050 **插补函数库相对独立，所有插补函数均以字母** LM **打头，按作用可分为参数设定函数、状态检测函数、实际操作函数、插补顺序执行函数。它们的函数原型如下：**

参数设定函数：

设置插补引擎参数：short LM_SetMinVel_6050 (….);
设置插补引擎参数：short LM_SetMaxVel_6050 (….);
设置插补引擎参数：short LM_SetSpeedPri_6050 (….);
设置插补引擎参数：short LM_SetParaAngle_6050 (….);
设置插补引擎参数：short LM_SetSysPara_6050 (….);

设置某轴与插补 X **匹配：** short LM_SetXAxis_6050 (….);
撤消插补 X **匹配：** short LM_OffXAxis_6050 ();

设置某轴与插补 Y 匹配： short LM_SetYAxis_6050 (….);
撤消插补 Y 匹配： short LM_OffYAxis_6050 ();
设置某轴与插补 Z 匹配： short LM_SetZAxis_6050 (….);
撤消插补 Z 匹配： short LM_OffZAxis_6050 ();
设置某轴与插补 W 匹配： short LM_SetWAxis_6050 (….);
撤消插补 W 匹配： short LM_OffWAxis_6050 ();

设置插补加减速： short LM_SetACCDec_6050 (….);
设置插补 S 形加减速： short LM_SetSAccePower_6050 (….);

状态检测函数：

返回插补命令缓冲区长度： short LM_GetBuffLen_6050 (….);
返回插补状态： short LM_GetState_6050 (….);
返回插补测量状态： short LM_GetMeasureState_6050 (….);
返回当前插补行号： short LM_GetLineNO_6050 (….);
返回插补编码器跟随状态： short LM_GetFEnState_6050 (….);

实际操作函数：

清空插补缓冲区： short LM_CleanBuff_6050 (….);
暂停插补运算： short LM_Pause_6050 (….);
恢复插补运算： short LM_Resume_6050 (….);
变更插补速度： short LM_SetSpeedRate_6050 (….);
插补到当前行结束停止： short LM_LineEnd_6050(….);
插补开始： short LM_Start_6050(….);
获得插补开始位置
X **轴：** short LM_GetXStartPos_6050(….);
Y **轴：** short LM_GetYStartPos_6050(….);
Z **轴：** short LM_GetZStartPos_6050(….);
W **轴：** short LM_GetWStartPos_6050(….);

插补顺序执行函数：

插补结束命令： short LM_End_6050 (….);
线性插补命令： short LM_Line_6050 (….);
线性插补命令： short LM_LineMaxV_6050 (….);

线性插补测量命令： short LM_LineMeasure_6050 (….);
线性插补跟随编码器位置： short LM_LineFE_6050 (….);

圆弧/螺旋线插补命令(顺圆)：
XY **平面：**short LM_ArcCW_6050 (….);
ZX **平面：**short LM_ArcCW_ZX_6050 (….);
YZ **平面：**short LM_ArcCW_YZ_6050 (….);
圆弧/螺旋线插补命令(逆圆)：
XY **平面：**short LM_ArcCCW_6050 (….);
ZX **平面：**short LM_ArcCCW_ZX_6050 (….);
YZ **平面：**short LM_ArcCCW_YZ_6050 (….);
插补等待命令： short LM_Wait_6050 (….);
插补等待输入点命令： short LM_Wait_I_6050 (….);
插补输出 IO **点命令：** short LM_IOOut_6050 (….);
插补输出 PWM **命令：** short LM_PWM_6050 (….);

使用插补函数的一般步骤如下：

1．**设置插补引擎参数、设置某些轴与插补虚拟** XYZW **轴匹配、设置插补加速度和减速度。**
2．**获得插补轴开始位置（通过** LM_GetXStartPos_6050 **等命令）**
3．**写入插补顺序执行命令（通过 直线、圆弧等插补命令 写入插补缓冲区）。**
4．**插补开始（通过 插补开始命令** LM_Start_6050（….）;
5．**根据需要，循环或定时检测插补缓冲区，如果不满，可继续写入插补顺序执行命令。**

注意：在步骤 1 和 2 时，注意相关轴必须是停止状态。

在写入插补顺序执行命令时，是写入插补缓冲区中，该插补缓冲区的长度为 16 **行，即一次最多可写** 16 **行插补顺序执行命令。插补引擎会按照写入顺序一行一行执行，直到碰到插补结束命令（**LM_End_6050）**或插补缓冲区空。用户也可用轴停止命令（**AbortAxis_6050、StopAxis_6050、CeaseAxis_6050）**来终止插补引擎工作。在插补引擎工作时也可随时暂停插补轴运动（用** LM_Pause_6050 **命令）和恢复插补运动（用** LM_Resume_6050 **命令）。在插补轴运动时也可用** LM_SetSpeedRate_6050 **命令随时改变插补速度。**

§2.4.5 插补疑问

⑴插补运算的单位是什么

距离是：用户单位；速度是：用户单位/分钟；加速度是：用户单位/秒平方

在用户设置某轴与插补虚拟的 X、Y、Z、W **轴进行匹配时（用** LM_SetXAxis_6050 **及** YZW **各轴相关函数 ），函数** LM_SetXAxis_6050 **中将代入该轴的脉冲当量，脉冲当量就是：该轴的电信号发一个脉冲，对应该轴实际移动多少用户单位。**

比如，某一直线轴通过滚珠丝杠与电机直联，该滚珠丝杠的螺距是 5 **毫米，而电机转一圈需** 2000 **个脉冲，则：轴的脉冲当量**=5/2000=0.0025**；**0.0025 **的含义是：一个脉冲将使该轴移动** 0.0025 **毫米。**

轴的脉冲当量很重要，轴的脉冲当量确定后，则该轴的用户单位就确定，在上例中轴的用户单位就是毫米。

在实际插补中，插补速度是几个轴的合成速度，即矢量速度。这种情况稍微复杂，解释如下：

比如，X、Y **两轴要插补运动，**X、Y **联在一个十字滑台上，即它们都是直线运动轴，它们的脉冲当量有可能不一样，但它们的用户单位是一样的，都是毫米，这时它们的插补矢量单位也是毫米，毫不含糊。矢量速度就是：毫米/分钟。**

如果，X、W **两轴要插补运动，**X **联在一个直线轴，**W **联在一个旋转轴上，这时不管它们的脉冲当量是否一样，但它们的用户单位是不一样的，一个是毫米，一个是度，这时它们的插补矢量单位就很不直观，是毫米？还是度？不管怎样理解，它们的插补运动按设置的轨迹严格执行的。**

例如编程如下（C **语言）：**

```
LM_SetXAxis_6050(0,0,0.0025,0.01);      //0 轴与 X 轴匹配，脉冲当量是：0.0025,直线轴
LM_SetWAxis_6050(0,3,0.089,0.01);      //3 轴与 W 轴匹配，脉冲当量是：0.089,  旋转轴
LM_SetACCDec_6050 (0,5000,5000);    //设置插补加速度和减速度
LM_GetXStartPos_6050(0,NULL);         //获得 X 轴插补开始位置，假如是 0
LM_GetWStartPos_6050(0,NULL);         //获得 W 轴插补开始位置，假如是 0
LM_Line_6050 (0,100,0,0,360,1000,100);    //X 轴终点位置是 100，W 轴是 360，
                                          //速度 1000/分钟
LM_End_6050(0);                           //插补结束
LM_Start_6050(0,0);                       //开始插补
```

上例中，虽然插补矢量单位不明确，但并不影响 X、W **两轴准确的按线性比例的运行到** 100、360 **的位置。**

⑵在插补开始前，为什么要获得轴插补开始位置

6050 **卡的插补运算是在一个虚拟的** XYZW **坐标系中进行，其插补函数位置坐标全是绝对坐标，所以在插补开始前，首先要确定起点。函数** LM_GetXStartPos_6050**（及对应的** Y、Z、W **轴函数）指示** 6050 **卡，**X、Y、Z、W **轴的当前位置为插补起始点。在后面可以看到，**

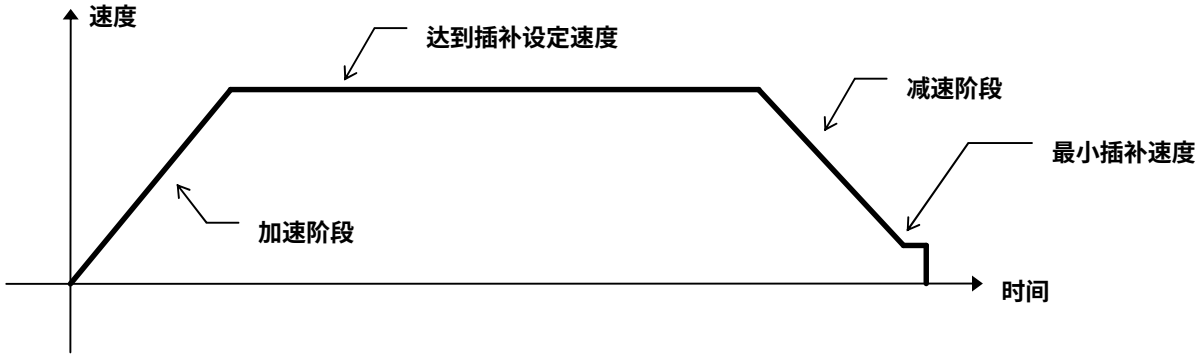
插补顺序执行函数所代入的轴位置参数都是轴的终点位置值，所以在插补开始前，显式指示 6050 卡获得插补开始位置（即：告诉 6050 卡，轴的当前位置是插补开始位置）就是很自然的事情。

目前在市场上流行的运动控制卡，绝大部分用相对坐标来进行插补运算，但在实际运用中（典型例子是 CNC 机床数控系统），绝大部分是用绝对坐标编程的，针对实际应用，6050 卡选择用绝对位置编程，且用 用户单位 编程。这样作有很多优点，它方便了用户编程，省去了很多中间计算环节，非常直观，且没有积累误差。

⑶6050 卡怎样处理插补加速度及插补速度

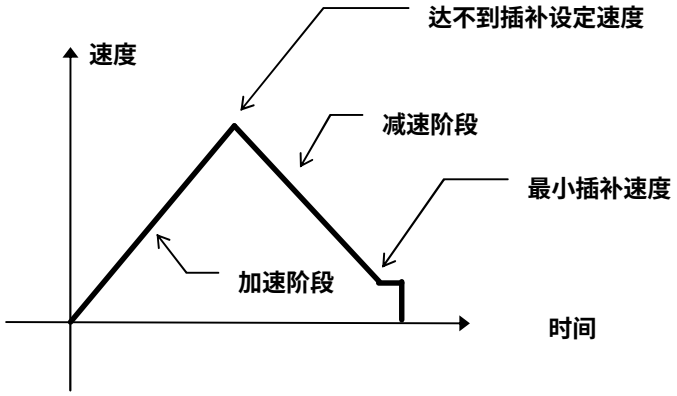
简单一条直线或圆弧，它们的加减速按直线处理，加速度和减速度可以不一样，如下：

直线形加速：



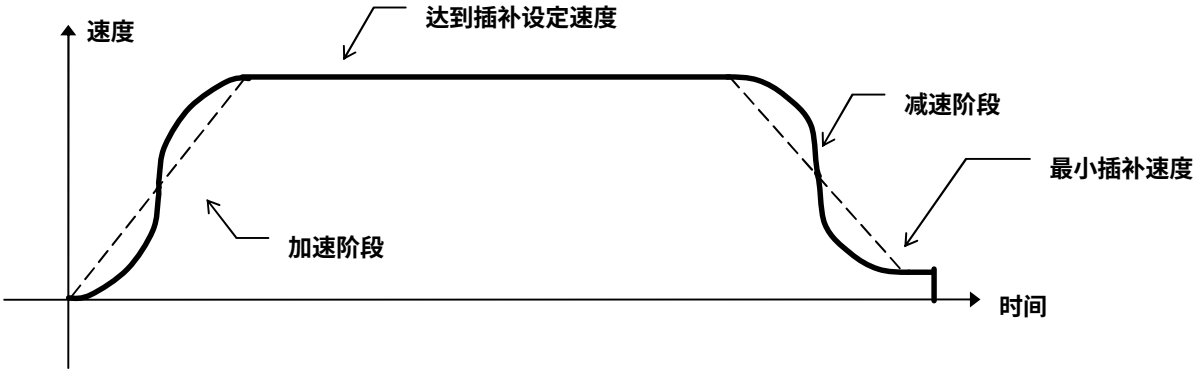
插补速度曲线

如果直线或圆弧很短，不够加减速，则插补速度曲线如下：

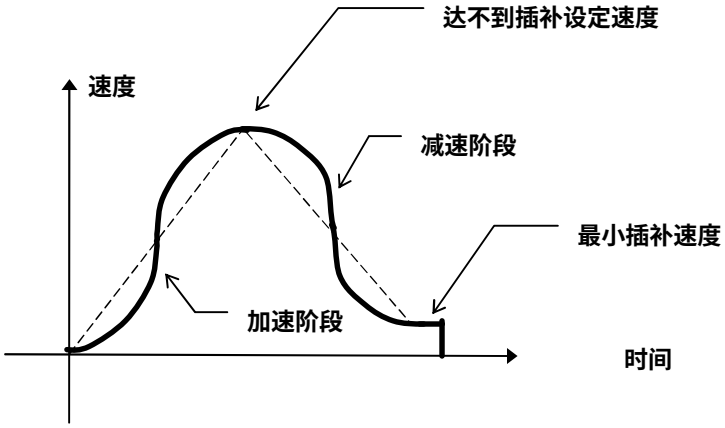


插补速度曲线

S 形加速：



插补速度曲线

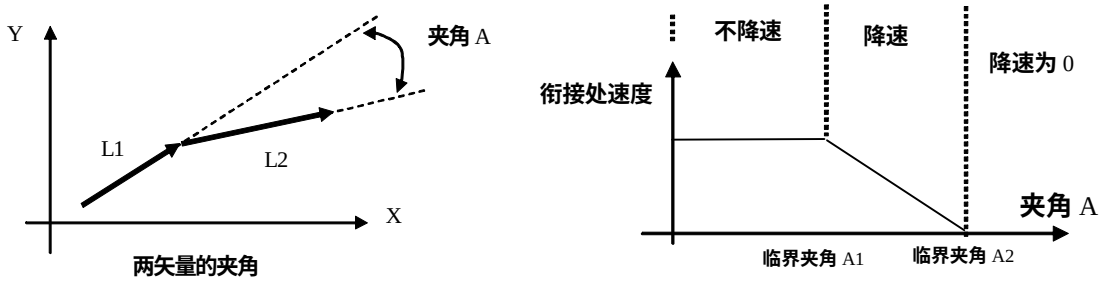


插补速度曲线

⑷两线段之间如何进行插补速度衔接

这里所谓“线段”是指直线或圆弧。6050 **卡在处理两线段之间速度衔接时，采用矢量夹角的方法进行处理，即根据两线段的矢量夹角来决定衔接处的插补速度。如下图所示：**

设线段 L1 **与线段** L2 **的插补速度是一样的，均为** F**，两线段的矢量夹角为** A**，由函数**



LM_SetParaAngle_6050（….）**设定了两个临界夹角：**A1 **和** A2**，则线段** L1 **终点处的速度** Fe **计算如下：**

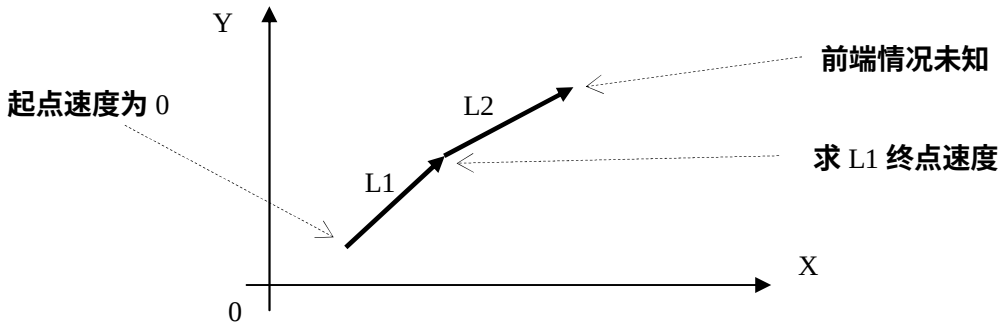
Fe=F； 当 A≤A1

Fe=0； 当 A≥A2

$$F_e = F \times \frac{A2 - A}{A2 - A1}$$ 当 A1≤A≤A2

⑸多个小线段连续插补时如何处理插补速度

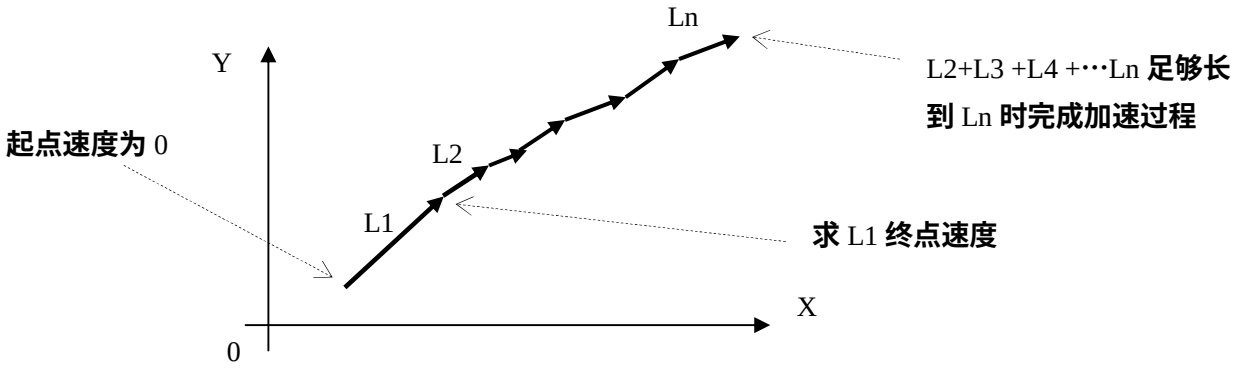
所谓小线段是指它的长度不足以按设定加减速完成加速过程或减速过程，加速过程是指从某一起始速度（比如 0 速）加速到指定速度，同理减速过程。通常插补引擎在运行到某一线段之前，要对它进行插补预处理，要预知它的起点速度和终点速度，线段首尾相连，本段的终点速度就是下一段的起点速度，所以在插补预处理中主要是计算线段的终点速度。当线段很短时，情况就变得复杂起来，举例如下：



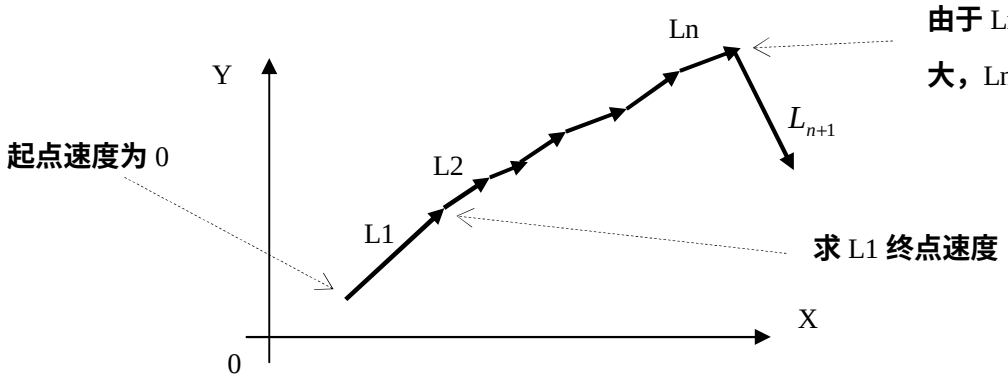
如上图，假如 L1 **、**L2 **矢量夹角很小，**L1 **是短线段（不足以完成加速过程）。按合理的推断** L1 **、**L2 **衔接处的速度（**L1 **终点速度）应该平滑过渡，假如** L2 **也很短（不足以完成加速过程），**L1 **终点速度到底是多少就很难推算。要推算** L1 **的终点速度，必须满足以下条件：**

1．L2 **足够长**

2．L2+L3 +L4 +…Ln **足够长，如下图：**



3．L2+L3 +L4 +…Ln **不足够长，但** Ln **的终点速度可知，如下图：**



不难看出，满足以上 3 种条件，就能计算出 L1 终点处的速度。

6050 卡的插补缓冲区有 16 行深度，当用户不断将插补数据写入缓冲区时，也就不断的激励 6050 卡进行插补预处理，由于 6050 卡采用以上所介绍的插补算法，使 6050 卡有能力进行多个小线段的连续高速插补。

在极端情况下，如果 16 行的小线段都不足以完成加减速过程，这时该如何处理呢？在 6050 卡的插补函数：LM_Start_6050(….)中有强制执行参数：ObligeFlag；当 ObligeFlag 设为 1 时，将假定第 16 行的（或最末端的）终点速度为 0，这样就能计算出各线段的终点速度，使插补引擎能够连续运行。

⑹什么时候发出插补开始命令 LM_Start_6050

1. 当插补缓冲区空时，用户然后写入插补顺序执行命令，这时发出插补开始命令 LM_Start_6050 将启动插补引擎。
2. 当插补引擎执行到 LM_End_6050 后，需要发出插补开始命令 LM_Start_6050 再次启动插补引擎。
3. 当发出 LM_LineEnd_6050 命令后，插补引擎运行到当前行结束并停止，需要发出插补开始命令 LM_Start_6050 再次启动插补引擎。
4. 当插补引擎被轴停止命令（StopAxis_6050、AbortAxis_6050、CeaseAxis_6050）终止后， 需要再次启动插补引擎，这时用户需要先清空插补缓冲区空，形成条件 1。

⑺在插补开始运行时，轴单独的控制命令如何运用

在相关轴进行插补运算时，除了轴停止命令（StopAxis_6050、AbortAxis_6050、CeaseAxis_6050），其它命令都不能执行。如果用户写入其它命令（比如 StartAxis_6050），系统将报错，用函数 GetErrorNo_6050 可以检测到错误号。

与插补不相关的轴（即：轴没有与插补虚拟轴 XYZW 进行匹配），可以正常操作。

⑻如何实现不同板间的的多轴插补

理论上不同板间的轴是不能实现插补的，因为插补运算是由板上的 DSP 完成的，举例说明，假如用户用到两块 6050 卡，要实现 8 个轴的线性插补，这时用户要自行计算两块卡上插

补矢量的分量，并分别写入各自板上的插补缓冲区，而后顺序启动两板的插补引擎，这时从微观来看，两块板上的插补引擎启动时间不能同步，最多相差在 3 毫秒之内（由 6050 卡的工作原理决定的）。如果这种误差能满足用户要求，也就能实现不同板间的插补。

还有一种方法，用函数 LM_IOOut_6050 (….)、LM_Wait_I_6050 (….)实现两板的同步。在两块卡的端子板上连一根线，比如板 0 的 1 号输出点连在板 1 的 1 号输入点上，在两卡的插补缓冲区中分别写入如下命令：

```
….
LM_IOOut_6050 (0，1，1，100)；//板 0 的 1 号输出点为 1
LM_Line_6050 (0，….);              //执行线性插补
….

LM_Wait_I_6050 (1，1，100)；     //等待板 1 的 1 号输入点为 1 时（实际为从板 0 发出）
LM_Line_6050 (1，….);              //执行线性插补
….
```

在上例中，两块板的线性插补同步时间在 1 毫秒之内。

⑼如何运用 PWM 输出与插补速度随动命令

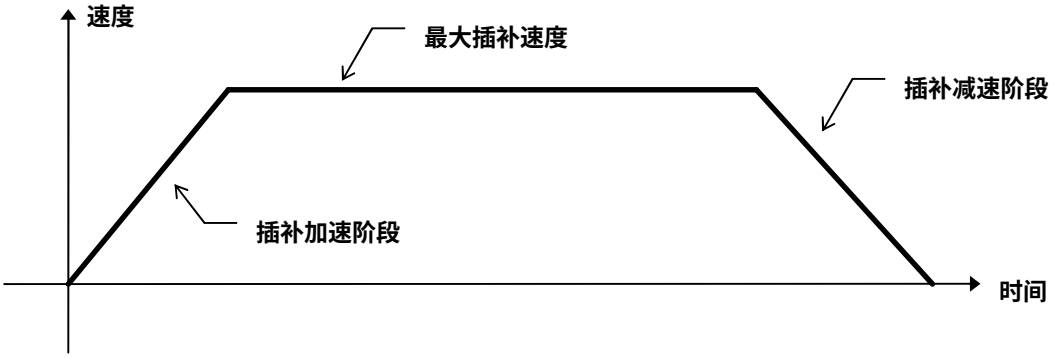
在激光加工中，PWM 信号一般用来控制激光功率，PWM 输出与插补速度随动命令（函数 LM_PWM_6050）用于实现这个目标。在程序前端写入 LM_PWM_6050 命令，往后 PWM 脉冲输出的占空比将随插补速度按比例线性变化，这些工作完全由板上的 DSP 完成，用户使用起来将非常方便。

⑽插补结束函数（LM_End_6050）起什么作用

插补结束函数（LM_End_6050）最主要的作用有两个：1、告诉 6050 卡，当执行到 插补结束函数（LM_End_6050）所在行时，插补速度为 0，即停止插补运动。2、告诉 6050 卡，何时开始新的插补运动，等待新的插补开始命令 LM_Start_6050（）。

如前所述，6050 卡对写入插补缓冲区的命令进行插补预处理，对各个插补线段的起点位置速度及终点位置速度进行预先计算，比如：当只执行一条很简单的直线插补命令 LM_Line_6050(0,1000,0,0,0,1000,100)时，其后也应该加入插补结束函数（LM_End_6050），这时 6050 明白只有一条直线插补，该线段终点位置的插补速度为 0，如下：

```
……
LM_Line_6050(0,1000,0,0,0,1000,100)；
LM_End_6050（0）；
LM_Start_6050（0,0）；
```



插补速度曲线

如果不写插补结束函数（LM_End_6050**），例如下：**

……

LM_Line_6050(0,1000,0,0,0,1000,100)；

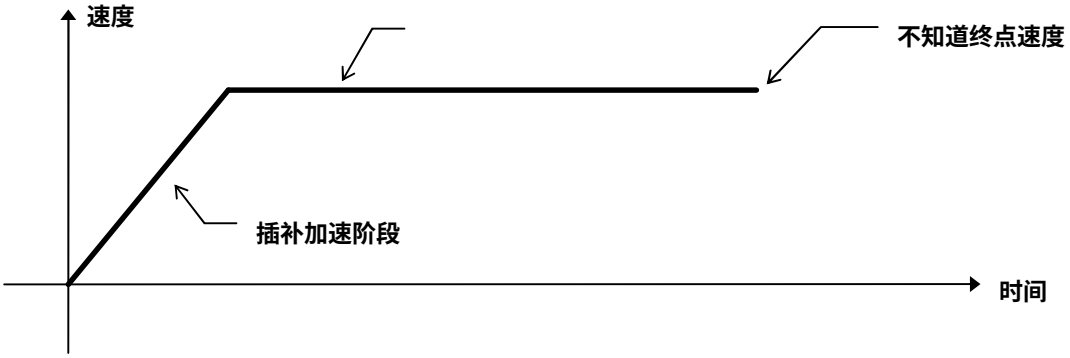
LM_Start_6050（0,0）；

由于不知道该线段终点位置的插补速度到底是多少，6050 **卡不能完成插补预处理（即不能完成该线段的速度规划），在上例中，在发出插补开始命令后，实际** 6050 **卡并不立即执行插补运动，除非插补开始命令代入强制执行参数，如下：**

LM_Line_6050(0,1000,0,0,0,1000,100)；

LM_Start_6050（0,1）； //**第** 2 **个参数为** 1**，表示强制执行（**6050 **卡认为**

//**该线段的终点速度为** 0）。



未完成的插补速度曲线

需要指出的是，能告知 6050 **卡插补速度为** 0 **的还有以下几个函数：**

插补等待命令： short LM_Wait_6050 (….);

插补等待输入点命令： short LM_Wait_I_6050 (….);

插补输出 IO **点命令：** short LM_IOOut_6050 (….);

插补输出 PWM **命令：** short LM_PWM_6050 (….);

⑾插补输出 IO 点函数 LM_IOOut_6050 与普通输出 IO 点函数 WriteIo_6050 的关系

在插补正在执行阶段（用函数 LM_GetState_6050 检测到的状态值为 7 ），函数 LM_IOOut_6050 的优先级高于普通输出 IO 点函数 WriteIo_6050，但低于按位输出 IO 点函数 WriteIoBit_6050。如，在插补序列中有函数 LM_IOOut_6050，它对第 1 号输出点置为 1，当插补运动执行到该行时，1 号输出点为 1，而这时再用 WriteIo_6050 函数对 IO 点置位时，第 1 号 IO 点的输出将不受影响，除非插补状态值不为 7。而按位输出 IO 点函数 WriteIoBit_6050 可随时控制输出点。

⑿函数 LM_GetState_6050 说明

函数 LM_GetState_6050 的原意是获得插补状态，其返回值 4 有两层意思：

1、插补缓冲区空

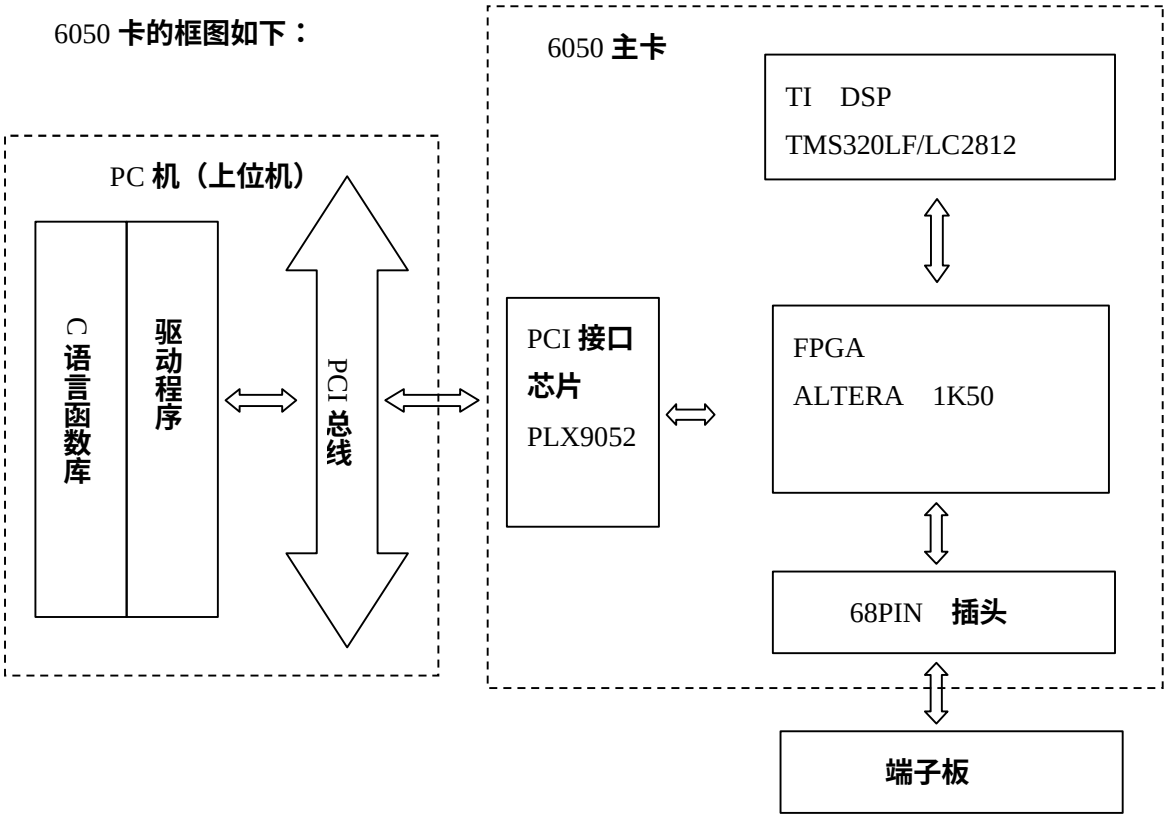
2、等待后续命令

当插补缓冲区还滞留有插补命令，而函数 LM_GetState_6050 返回值为 4 时（等待后续命令），其原因就是问题(10)中所阐述的现象，即 6050 卡未能完成插补预处理，没有完成该线段的插补速度规划。解决方法是加入插补结束函数（LM_End_6050）或 用 函数 LM_Start_6050（0,1）强制运行。

§2.4.5 其它说明

⑴6050 卡工作原理

6050 卡的框图如下：



如上图所示，在 FPGA **中开辟有** DSP **与** PC **机的数据交换区，**PC **需要读取的数据（轴位置、速度、轴状态、插补状态及** IO **等），**DSP **会每** 1 **毫秒更新一次；而** PC **机在写入数据和命令时，先将数据写入数据交换区，然后向** DSP **发出中断信号通知** DSP **读取或执行命令。**

基于以上机理，如果编程如下：

```
…. ….
CeaseAxis_6050（0，0）；          //0 轴立即停止
State= ReadAxisState_6050(0，0);   //读取 0 轴状态
…. ….
```

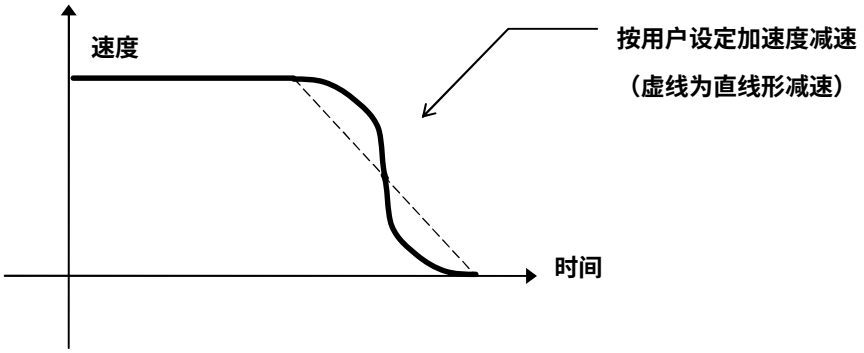
State **并不能马上返回**-5**（轴被** CeaseAxis_6050 **命令停止），一般按如下编程，**State **就能返回**-5 **值。**

```
…. ….
CeaseAxis_6050（0，0）；          //0 轴立即停止
Sleep（2）；                      //程序等待 2 毫秒
State= ReadAxisState_6050(0，0);   //读取 0 轴状态
…. ….
```

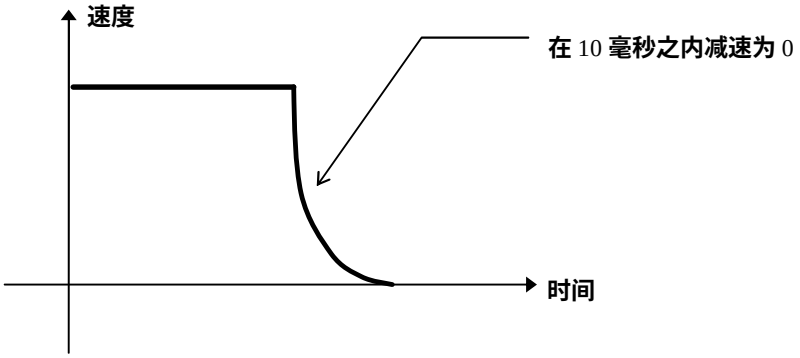
⑵轴停止命令

轴停止命令有三种，它们的速度曲线如下：

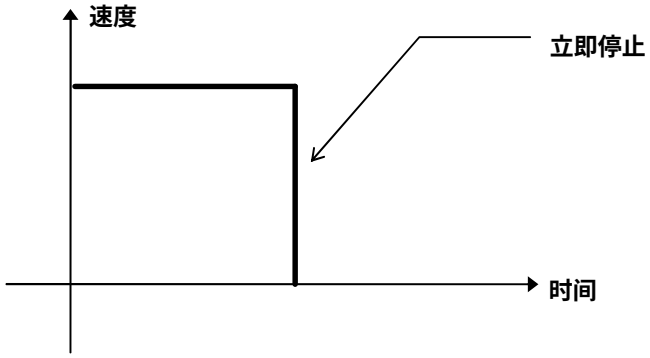
1． StopAxis_6050 **模式：**



2．AbortAxis_6050 **模式：**



3．CeaseAxis_6050 模式：



⑶通用 IO 点

6050 卡有 8 个输出点，20 个输入点，其中 20 个输入点中第 9 至第 20 点有双重目的，在缺省状态下，它们分别是 4 个轴的正限位输入点、0 位输入点、负限位输入点。可用函数 SetAxisIO_6050 来控制它们是否有效，无效的轴 IO 可以当普通 IO 来用。当轴 IO 有效时，当碰到限位时，输入点会以中断的方式通知 DSP，DSP 将以 AbortAxis_6050 模式使轴停止运动。6050 还可接入急停开关，用 Set_Emergency_Stop_6050 函数可以使急停开关有效并与第 1 至第 8 某一输入点对应。缺省时急停开关无效。当急停信号到来时，DSP 会以 CeaseAxis_6050 模式使全部轴停止。

⑷PWM/DA 输出与编码器输入

6050 卡用同一个输出信号来实现 PWM 或 DA 输出，当用户选择 DA 输出时，在信号前端还是 PWM 信号，它的频率是固定的 25K，6050 卡通过改变占空比来改变电压。

在实际应用中，PWM 输出可控制激光电源，DA 输出可控制主轴转速。编码器输出可接一个手摇轮。

⑸轴位置计数器溢出

6050 卡的轴位置计数器在主卡上的 FPGA 中，是一个 32 位可逆计数器，它真实地记录每个轴发出的脉冲个数，其有效值范围为：-2147483648 至 2147483647。当轴沿一个方向长时间运动时，轴位置计数器有可能溢出。每转 10000 线的伺服电机，按 3000 转/分钟旋转，当运行大概不到 72 分钟时，位置计数器就溢出了。

6050 卡中，如果轴位置计数器溢出，当轴以位置模式或插补模式运行时，6050 卡会以立即停止模式（CeaseAxis_6050 模式）让轴停止运行，同时报警（用 GetErrorNo_6050 函数可检测到）；如果是速度模式（用 StartAxisVel_6050 命令启动轴运动），则 6050 卡只会报警，不会将轴停止。

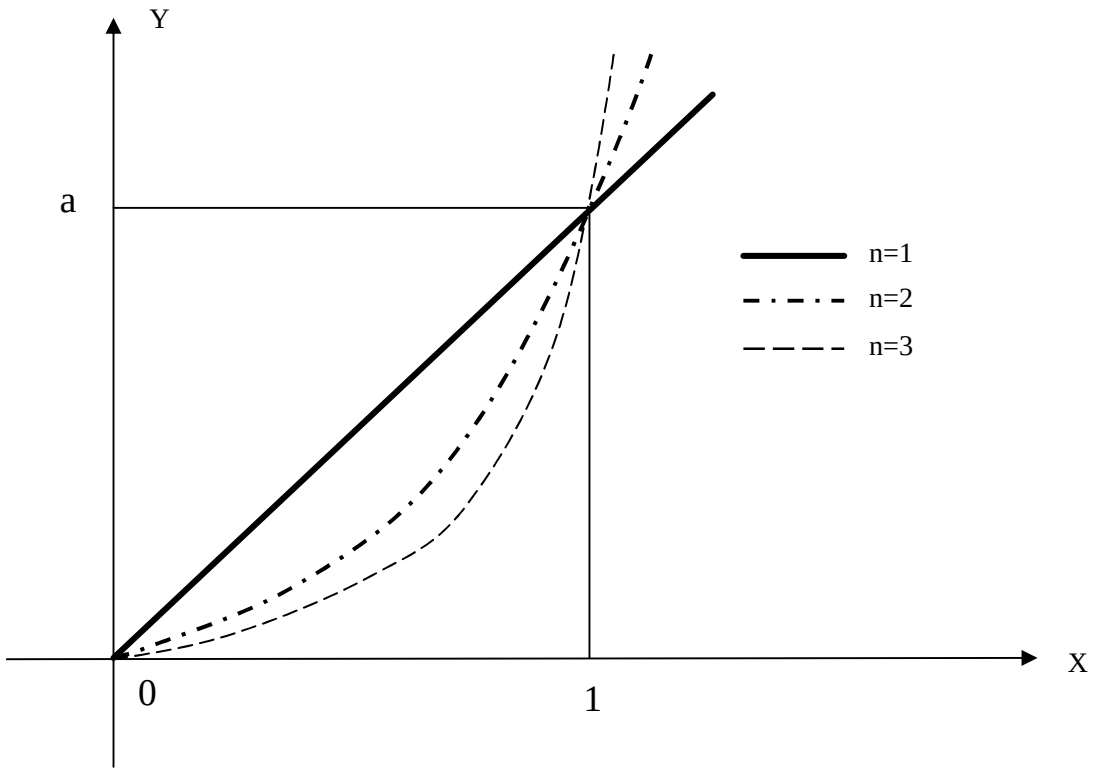
溢出时的轴位置计数器将从最大正值转到最小负值，或从 最小负值转到最大正值。

⑹S 形加减速说明

无论是单轴运动或是多轴插补运动，6050 均可采用直线形或 S 形加减速，原理说明如下：

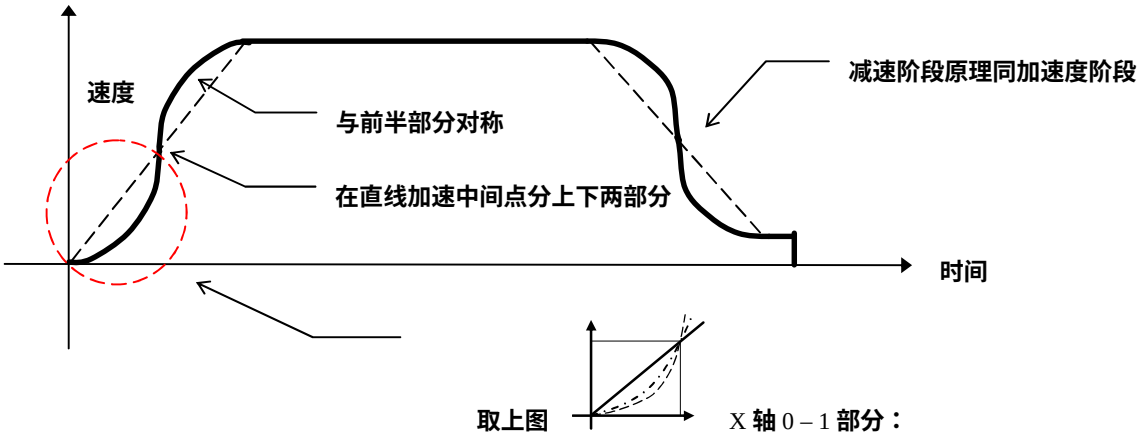
指数公式 ： $y = ax^n$

曲线如下：



从上图可以看出，如果 X **是时间轴，**Y **是速度轴，如果** n=1 **就是直线加速度。在** 6050 **卡中统一采用指数曲线形状加减速，将直线形加减速度当成指数曲线的一个特例。用户在设加速度参数时，一般设两个参数：即：加速度和指数，加速度就是上图** a**，指数对应上图的** n**。**

实际情况要更复杂，实际的算法优化如下图：



S **形加速度曲线**

§2.4.6 库函数一览表

表 2.5 库函数一览表 (C 语言方式)

编号	函数名	描述	详解页码
初始化函数			
1	InitCard_6050	初始化 6050 卡	39
23	ExitCard_6050	退出 6050 卡,并释放所占资源	40
单轴运动控制			
3	Home_6050	轴位置清零	56
4	SetAxisPos_6050	设置轴的位置	42
5	SetAxisVel_6050	设置轴的速度	43
6	SetAxisAcc_6050	设置轴的加速度	44
7	StartAxis_6050	轴开始运行,位置模式	46
8	StartAxisVel_6050	设置轴速度模式,并按所设速度开始运行	47
9	StopAxis_6050	轴停止,按所设加速度减速停止	54
15	SetAxisIO_6050	设置轴限位 IO 是否有效	77
17	SetAxisOutMode_6050	设置轴的输出模式	41
19	AbortAxis_6050	轴停止,在 10 毫秒之内减速停止	53
20	CeaseAxis_6050	轴停止,立即停止,无减速过程	55
24	GoHome_6050	轴回 Home 点	57
72	SetAxisSAcce_6050	设置轴 S 型加速度	45
73	SetAxisTEC_6050	设置轴螺距反向间隙补偿	61
74	SetAxisFE_6050	设置轴跟随编码器运动	50
75	SetAxisEGear_6050	设置轴电子齿轮运动	48
76	CancelAxisFEG_6050	取消轴跟随运动	52
84	SetAxisOffset_6050	设置轴位置偏移值	63
轴位置速度状态读取			
2	ReadAxisPos_6050	读取轴的当前位置	59
26	ReadAxisVel_6050	读取轴的当前速度	60
27	ReadAxisState_6050	读取轴的状态	61
编码器位置读取			
10	ReadEncoderPos_6050	读取编码器位置	64
11	HomeEncode_6050	复位编码器,编码器位置清零	65
80	GetEnAuto0Flag_6050	获得编码器自动清零标志	66
79	ResetEn0Flag_6050	编码器自动清零标志置 0	67
PWM 脉冲/DA 输出控制			

12	PwmOut_6050 PwmOut2_6050	PWM 脉冲输出	79 80
13	PwmStop_6050	PWM 脉冲停止输出	81
68	DAOut_6050	DA(数模转换)模拟量输出	82
读取 6050 卡有关版本信息			
28	ReadFirmwareVersion_6050	读取 6050 控制卡固件版本号	68
67	ReadDllVersion_6050	读取 6050 控制卡动态链接库版本号	69
70	GetDriverVersion_6050	读取 6050 卡的驱动版本号	70
I/O 读写控制			
29	Set_Emergency_Stop_6050	设置 IO 急停信号	78
14	ReadIO_6050	读取通用输入点 I1-I20 状态	71
14	ReadIOBit_6050	按位读取通用输入点 I1-I20 状态	72
16	WriteIo_6050	设置通用输出点 O1-O8 状态	73
81	WriteIoBit_6050	按位设置通用输出点 O1-O8 状态	74
82	ReadOs_6050	读取输出点 O1-O8 状态	75
83	ReadOsBit_6050	按位读取输出点 O1-O8 状态	76
插补函数			
插补初始化函数			
32	LM_SetXAxis_6050 LM_SetYAxis_6050 LM_SetZAxis_6050 LM_SetWAxis_6050	将实际轴与插补引擎的 X 轴相匹配	89
33	LM_OffXAxis_6050 LM_OffYAxis_6050 LM_OffZAxis_6050 LM_OffWAxis_6050	撤消插补引擎的 X 轴匹配	91
54	LM_GetXStartPos_6050 LM_GetYStartPos_6050 LM_GetZStartPos_6050 LM_GetWStartPos_6050 LM_GetAxisStartPos_6050	获得插补轴当前位置,也是轴插补的开始位置,在发送插补命令 (LM_Line_6050,IpolArc_6030)开始前调用	112 113
插补顺序执行函数			
42	LM_Line_6050	直线插补 该命令将把插补数据送入 6050 卡的插补缓存	94

		器中,6050 卡的插补缓存器总共有 16 行	
69	LM_LineMaxV_6050		96
65	LM_LineMeasure_6050	直线插补并测量输入点	97
66	LM_GetMeasureState_6050	获得插补测量状态	99
77	LM_LineFE_6050	直线插补 跟随编码器位置	100
43	LM_ArcCW_6050	圆弧插补,顺圆 该命令将把插补数据送入 6050 卡的插补缓存器中,6050 卡的插补缓存器总共有 16 行	104
	LM_ArcCW_YZ_6050		104
	LM_ArcCW_ZX_6050		104
44	LM_ArcCCW_6050	圆弧插补,逆圆 该命令将把插补数据送入 6050 卡的插补缓存器中,6050 卡的插补缓存器总共有 10 行	104
	LM_ArcCCW_YZ_6050		105
	LM_ArcCCW_ZX_6050		105
60	LM_Wait_6050	插补等待	108
61	LM_PWM_6050	插补模式的 PWM 输出，占空比随速度变化	83
62	LM_IOOut_6050	插补模式的 IO 点输出	109
63	LM_Wait_I_6050	插补等待通用 IO 点输入	110
45	LM_End_6050	插补结束 该命令将把插补数据送入 6050 卡的插补缓存器中,6050 卡的插补缓存器总共有 10 行 当插补引擎运行到该行时将停止运行,如要重新开始,则再送入插补数据命令(LM_Line_6050,LM_ArcCW_6050,LM_ArcCCW_6050),并发 LM_Start_6050 命令	111
插补控制函数			
40	LM_SetACCDec_6050	设置插补加速度和减速度	92
71	LM_SetSAccePower_6050	设置插补 S 型加速度(1,2,3,4)指数	93
41	LM_Start_6050	插补开始	114
46	LM_CleanBuff_6050	清除插补缓存	115
55	LM_GetBuffLen_6050	获得插补缓存剩余长度	116
47	LM_SetMinVel_6050	设插补最小速度	86
57	LM_SetMaxVel_6050	设插补最大速度	87
48	LM_SetParaAngle_6050	设插补参数,角度	88
49	LM_GetLineNO_6050	获得插补当前行号	118
50	LM_Pause_6050	插补暂停	119
51	LM_Resume_6050	恢复插补暂停	120

52	LM_SetSpeedRate_6050	**设插补速率**	121
53	LM_LineEnd_6050	**插补运行完当前行停止**	125
56	LM_GetState_6050	**获得插补状态**	117
58	LM_SetSpeedPri_6050	**设插补速度优先还是精度优先**	85
59	LM_SetSysPara_6050	**设插补系统参数**	84
78	LM_GetFEnState_6050	**获得轴编码器跟随已达到目标状态**	103
100	GetErrorNo_6050	**获得错误号**	122

第三章　库函数详解

InitCard_6050

初始化控制卡

函数编号:1

函数原型：

C: short InitCard_6050(short Board_NO,short Axis0,short Axis1,
short Axis2,short Axis3);

VB: Declare Function InitCard_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByVal Axis0 As Integer, ByVal Axis1 As Integer, ByVal Axis2 As Integer, ByVal Axis3 As Integer) As Integer

Delphi: function InitCard_6050(Board_NO : smallInt; Axis0 : smallInt; Axis1 : smallInt; Axis2 : smallInt; Axis3 : smallInt) : SmallInt; stdcall;

解释：

这个函数初始化电机控制卡，用户应在程序的初始化部分调用该函数，对每个板进行初始化。它将做以下工作：

①初始化板上芯片

②轴位置寄存器清零

③所有轴输出信号都设置 脉冲/方向 模式，脉冲信号端（CW）为高电平，方向信号端（D）为低电平，即轴控制信号设置为：脉冲/方向，共阳极模式。

④所有 IO 输出信号都设置为低电平（重新上电时）。如果不是重新上电，6050 卡将保持原来状态。

轴正在运行时是否可调用：否

参数：

board_NO: **板号，可能值** 0、1、2、3**；一台计算机中最多可以运行** 4 **块控制卡。**

Axis0,Axis1,Axis2,Axis3: 0 **或** 1, 0=**无效**,1=**有效**

返回码：

(0) **成功**

(-1、-2、-3) **参数不对，内存不够，或** 6050 **控制板不存在。**

参考函数：Exit6050Card

ExitCard_6050

释放控制卡驱动库函数

函数原型：

函数编号:23

C： short ExitCard_6050(short Board_NO);

VB: Public Declare Function ExitCard_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer) As Integer

Delphi: function ExitCard_6050(Board_NO : smallInt) : SmallInt; stdcall;

解释：

释放驱动函数库，这个函数释放 6050 **函数库所占用资源。在用户程序退出之前应调用该函数一次。**

轴正在运行时是否可调用：否

参数：

board_NO: **板号，可能值** 0、1、2、3；

返回码：

(0) **成功**

(-1) **非法操作**

参考函数：

InitCard_6050

SetAxisOutMode_6050

设置轴的输出模式

函数编号:17

函数原型：

C: short SetAxisOutMode_6050(short Board_NO,
unsigned short Axis_No,
short Mode_A,
short Mode_B,
short Mode_C);

VB: Declare Function SetAxisOutMode_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByVal Axis_NO As Integer, ByVal Mode_A As Integer, ByVal Mode_B As Integer, ByVal Mode_C As Integer) As Integer

Delphi: function SetAxisOutMode_6050(Board_NO : smallInt; Axis_No : Word; Mode_A : shortInt; Mode_B : shortInt; Mode_C : shortInt) : SmallInt; stdcall;

解释：

这个函数用来设置轴脉冲的输出模式及轴方向，6050 **板卡支持“脉冲/方向”和“正向脉冲/反向脉冲”两种输出模式。**

轴正在运行时是否可调用：否

参数：

Board_NO	**卡号，可能值** 0，1，2，3
Axis_No	**轴号，可能值** 0，1，2，3
Mode_A	0 **或** 1， 0=**共阳极输出**,1=**共阴极输出**
Mode_B	0 **或** 1， 0=**脉冲-方向模式输出**,1=**脉冲-脉冲模式输出**
Mode_C	0 **或** 1， 0=**轴方向正常**,1=**轴方向反转**

返回码：

(0) **成功**

(-1) **非法操作**

SetAxisPos_6050

设置梯形包络线的终点位置

函数编号:4

函数原型：

C： short SetAxisPos_6050(short Board_NO,unsigned short Axis_No,long position);

VB： Declare Function SetAxisPos_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByVal Axis_NO As Integer, ByVal position As Long) As Integer

Delphi: function SetAxisPos_6050(Board_NO : smallInt; Axis_No : Word; position : LongInt) : SmallInt; stdcall;

解释：

设置轴的终点位置参数，在调用该函数后，用户再调用 startAxis_6050 **函数，轴终点位置的新参数才有起作用。**

轴正在运行时是否可调用：是

参数：

Board_NO **卡号，可能值** 0，1，2，3

Axis_No **轴号，可能值** 0，1，2，3

position **轴的位置**; **范围**: **长整型数**; **单位**: **脉冲个数。**

返回码：

(0) **成功**

(-1) **非法参数**

参考函数：

SetAxisVel_6050

SetAxisAcc_6050

StartAxis_6050

SetAxisVel_6050

设置梯形包络线的最大速度

函数编号:5

函数原型：

C： short SetAxisVel_6050(short Board_NO,unsigned short Axis_No,long velocity);

VB： Declare Function SetAxisVel_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByVal Axis_NO As Integer, ByVal velocity As Long) As Integer

Delphi: function SetAxisVel_6050(Board_NO : smallInt; Axis_No : Word; velocity : LongInt) : SmallInt; stdcall;

解释：

设置轴的最大速度，在调用该函数后，用户再调用 StartAxis_6050 **函数，新参数才起作用。**

轴正在运行时是否可调用：是

参数：

Board_NO **卡号，可能值** 0，1，2，3

Axis_No **轴号，可能值** 0，1，2，3

Velocity **轴的速度， 范围**: **长整型数**; **单位**: Hz，**轴的脉冲频率**

返回码：

(0) **成功**

(-1) **非法参数**

参考函数：

SetAxisPos_6050

SetAxisAcc_6050

StartAxis_6050

SetAxisAcc_6050

设置梯形包络线的加速度，绝对值

函数编号:6

函数原型：

C： short SetAxisAcc_6050(short Board_NO,unsigned short Axis_No,
long acceleration);

VB： Declare Function SetAxisAcc_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByVal Axis_NO As Integer, ByVal acceleration As Long) As Integer

Delphi: function SetAxisAcc_6050(Board_NO : smallInt; Axis_No : Word; acceleration : LongInt) : SmallInt; stdcall;

解释：

设置轴的加速度参数。在控制卡初始化时，每个轴的加速度均为 0。

轴正在运行时是否可调用：否

参数：

Board_NO	**卡号，可能值 0，1，2，3**
Axis_No	**轴号，可能值 0，1，2，3**
Acceleration	**轴的加速度; 范围: 长整型数,必须是正值; 单位: 脉冲数 / 秒平方。**

返回码：

(0) **成功**

(-1) **非法参数**

参考函数：

SetAxisPos_6050

SetAxisVel_6050

StartAxis_6050

SetAxisSAcce_6050

SetAxisSAcce_6050

设置设置轴 S 型加速度

函数编号:72

函数原型：

C： short SetAxisSAcce_6050(short Board_NO,unsigned short Axis_No,
short PowerFlag);

VB： Declare Function SetAxisAcc_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByVal Axis_NO As Integer, ByVal acceleration As Long) As Integer

Delphi: function SetAxisSAcce_6050(Board_NO : smallInt; Axis_No : smallInt; PowerFlag : shortInt)shortInt; stdcall;

解释：

设置轴的加速度曲线参数。在控制卡初始化时，每个轴的加速度指数曲线为 2。S **形加速度理论请看§**2.4.5 **节有关内容。**

轴正在运行时是否可调用：否

参数：

Board_NO ： **卡号，可能值** 0，1，2，3

Axis_No ： **轴号，可能值** 0，1，2，3

PowerFlag ： 1-4 S **型加速度**(1,2,3,4)**指数，**1=**直线形加速度。系统缺省值：**2

返回码：

(0) **成功**

(-1) **非法参数**

参考函数：

SetAxisAcc_6050

StartAxis_6050

更新梯形包络线参数，轴开始运动

函数编号:7

函数原型：

C: short StartAxis_6050(short Board_NO,unsigned short Axis_No);

VB: Declare Function StartAxis_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByVal Axis_NO As Integer) As Integer

Delphi: function StartAxis_6050(Board_NO : smallInt; Axis_No : Word) : SmallInt; stdcall;

解释：

轴以位置模式开始运行，在这之前，必须设好轴的位置、速度和加速度。

轴正在运行时是否可调用：是

参数：

Board_NO **卡号，可能值** 0，1，2，3

Axis_No **轴号，可能值** 0，1，2，3

返回码：

(0) **成功**

(-1) **不成功**

参考函数：

SetAxisPos_6050

SetAxisVel_6050

SetAxisAcc_6050

StartAxisVel_6050

设置轴为速度模式

函数编号:8

函数原型：

C: short StartAxisVel_6050 (short Board_NO,unsigned short Axis_No,long velocity);

VB: Declare Function StartAxisVel_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByVal Axis_NO As Integer, ByVal velocity As Long) As Integer

Delphi: function StartAxisVel_6050(Board_NO : smallInt; Axis_No : Word; velocity : LongInt) : SmallInt; stdcall;

解释：

轴以速度模式开始运行，在这之前，必须设好轴的加速度。

轴正在运行时是否可调用：是

参数：

Board_NO **卡号，可能值** 0，1，2，3

Axis_No **轴号，可能值** 0，1，2，3

velocity: **轴的最大速度，可正负，单位：脉冲数 / 秒（**Hz**）。**

返回码：

(0) **成功**

(-1) **不成功**

参考函数：

SetAxisEGear_6050

设置轴电子齿轮运动

函数编号:75

函数原型：

C: short SetAxisEGear_6050 (short Board_NO,unsigned short Axis_No,
unsigned short F_Axis_No,double Rate,double Kp);

VB: Declare Function StartAxisVel_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByVal Axis_NO As Integer, ByVal velocity As Long) As Integer

Delphi: function SetAxisEGear_6050(Board_NO : smallInt; Axis_No : smallInt; F_Axis_No : smallInt; Rate : double; Kp : double)shortInt; stdcall;

解释：

该函数设定轴以电子齿轮模式工作，设定完后立即生效。主从关系、齿轮比可通过参数设定。主从关系可随意设定，只要不违反自己随自己或父子孙的伦理关系就可以。设定时要求主动轴、从动轴均在停止状态(6050 **卡会在两轴停止状态那一刻读取两轴的当前位置,以此为两轴的开始位置**)，**在设定完后从动轴为电子齿轮模式，而主动轮可以为其它任意模式**(**除了** Home **和** GoHome **命令**)。

轴正在运行时是否可调用：否

参数：

Board_NO **：卡号，可能值** 0，1，2，3

Axis_No **：**0－3, **从动轴号**

F_Axis_No **：**0－3, **主动轴号**

Rate **： 从动轴跟随比率：**

设：

A1=**从动轴开始位置，**A2=**从动轴实时位置，单位：脉冲数**

B1=**主动轴开始位置，**B2=**主动轴实时位置，单位：脉冲数**

则：

$$Rate = \frac{A2 - A1}{B2 - B1}$$

可正负；**范围**：**绝对值** 0.001-1000；　**单位**：**无**；　**分辨率**: 0.001

Kp　　：**跟随** PID **调节系数**；**范围**: 0.001-1000；　**单位**：**无**；　**分辨率**: 0.001，**一般设为** 0.1、0.2 **就可。**

返回码：

(0)　**成功**

(-1)　**不成功**

参考函数：

CancelAxisFEG_6050

SetAxisFE_6050

设置轴跟随编码器运动

函数编号:74

函数原型：

C: short SetAxisFE_6050 (short Board_NO,unsigned short Axis_No,
unsigned short F_Axis_No,double Rate,double Kp);

VB: Declare Function StartAxisVel_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByVal Axis_NO As Integer, ByVal velocity As Long) As Integer

Delphi: function SetAxisFE_6050(Board_NO : smallInt; Axis_No : smallInt; Rate : double; Kp : double; Mode1 : shortInt; Mode2 : shortInt; Mode3 : shortInt)shortInt; stdcall;

解释：

该函数设定轴以编码器控制模式工作，设定完后立即生效。跟随关系、跟随比可通过参数设定。在设定完后轴为编码器控制模式工作，这时编码器(或接手摇轮、或是某一轴的位置反馈)的运动将带动该轴运动，这当中编码器不能执行编码器位置清零命令
(HomeEncode_6050 **命令**)。

轴正在运行时是否可调用：根据参数决定

参数：

Board_NO **：卡号，可能值** 0，1，2，3

Axis_No **：**0－3, **轴号**

Rate **： 轴跟随比率：**

设：

A1=**从动轴开始位置，**A2=**从动轴实时位置，单位：脉冲数**

B1=**编码器开始位置，**B2=**编码器实时位置，单位：脉冲数**

则：

$$Rate = \frac{A2 - A1}{B2 - B1}$$

可正负; **范围**: **绝对值** 0.001-1000; **单位**: **无**; **分辨率**: 0.001

Kp **： 跟随** PID **调节系数**; **范围**: 0.001-1000; **单位**: **无**; **分辨率**: 0.001**，一般设为**

0.1、0.2 就可。

Mode1 ： 0=速度模式,1=位置模式

在速度模式下，轴的位置跟随不会严格按跟随比率来执行，运动效果更为柔和，而位置模式则会严格按跟随比率执行，运动柔和与否可用参数 Kp 调节。

Mode2 ： 0= 不自动清零， 1=自动清零

6050 卡的编码器计数器虽为 32 位，但为了防止编码器计数器溢出，可在开始时自动将编码器计数器清零。

Mode3 ： 0=停止状态跟随，1=可运动状态跟随

当轴的速度不为零时（运动状态）可立即转为编码器跟随模式。在实践中，比如数控铣床的攻丝、数控车床的切螺纹等操作，均是某一轴在运动状态下过渡到与编码器跟随。

返回码：

(0) 成功

(-1) 不成功

参考函数：

CancelAxisFEG_6050

CancelAxisFEG_6050

取消轴跟随运动

函数编号:76

函数原型：

C: short CancelAxisFEG_6050 (short Board_NO,unsigned short Axis_No);

VB: Declare Function StartAxisVel_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByVal Axis_NO As Integer, ByVal velocity As Long) As Integer

Delphi: function CancelAxisFEG_6050(Board_NO : smallInt; Axis_No : smallInt)shortInt; stdcall;

解释：

当轴以电子齿轮模式工作或以编码器控制模式工作时，要取消这种模式，则可用该函数解除轴的这两种工作模式。解除轴跟随模式只能用该函数或轴停止命令（StopAxis_6050、AbortAxis_6050、CeaseAxis_6050）。 (**插补模式下的编码器控制模式除外,看函数** LM_LineFE_6050)

轴正在运行时是否可调用：否

参数：

Board_NO **：卡号，可能值** 0，1，2，3

Axis_No **：**0－3, **轴号**

返回码：

(0) **成功**

(-1) **不成功**

参考函数：

AbortAxis_6050

轴停止,在 10 毫秒之内减速停止

函数编号:19

函数原型：

C: short AbortAxis_6050(short Board_NO,unsigned short Axis_No);

VB: Declare Function AbortAxis_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByVal Axis_NO As Integer) As Integer

Delphi: function AbortAxis_6050(Board_NO : smallInt; Axis_No : Word) : SmallInt; stdcall;

解释：

这个函数将使轴的速度在 10 毫秒之内降为零。

轴正在运行时是否可调用：是

参数：

Board_NO **卡号，可能值** 0，1，2，3

Axis_No **轴号，可能值** 0，1，2，3

返回码：

(0) **成功**

(-1) **不成功**

参考函数：

StopAxis_6050

CeaseAxis_6050

StopAxis_6050

轴停止,按所设加速度减速停止

函数编号:9

函数原型：

C: short StopAxis_6050(short Board_NO,unsigned short Axis_No);

VB: Declare Function StopAxis_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByVal Axis_NO As Integer) As Integer

Delphi: function StopAxis_6050(Board_NO : smallInt; Axis_No : Word) : SmallInt; stdcall;

解释：

这个函数将使轴的速度以用户设定的减速度方式降为零。减速度就是所设的轴加速度。

轴正在运行时是否可调用：是

参数：

Board_NO **卡号，可能值** 0，1，2，3

Axis_No **轴号，可能值** 0，1，2，3

返回码：

(0) **成功**

(-1) **不成功**

参考函数：

AbortAxis_6050

CeaseAxis_6050

CeaseAxis_6050

轴停止，立即停止，无减速过程

函数编号：20

函数原型：

C: short CeaseAxis_6050(unsigned short Board_NO,unsigned short Axis_No);

VB: Declare Function CeaseAxis_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByVal Axis_NO As Integer) As Integer

Delphi: function CeaseAxis_6050(Board_NO : smallInt; Axis_No : Word) : SmallInt; stdcall;

解释：

这个函数将使轴的速度立即降为零。

轴正在运行时是否可调用：是

参数：

Board_NO **卡号，可能值** 0，1，2，3

Axis_No **轴号，可能值** 0，1，2，3

返回码：

(0) **成功**

(-1) **不成功**

参考函数：

StopAxis_6050

AbortAxis_6050

Home_6050

轴位置寄存器清零

函数编号:3

函数原型：

C: short Home_6050(short Board_NO,unsigned short Axis_No);

VB: Declare Function Home_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByVal Axis_NO As Integer) As Integer

Delphi: function Home_6050 (Board_NO : smallInt; Axis_No : Word) : SmallInt; stdcall;

解释：

这个函数将把轴内部位置计数器清零，轴位置计数器存在于电机控制卡内部，用户只能通过 Home_6050、GoHome_6050 **和** ReadAxisPos_6050 **函数对其进行清零和读取。**

轴正在运行时是否可调用：否

参数：

Board_NO **卡号，可能值** 0，1，2，3

Axis_No **轴号，可能值** 0，1，2，3

返回码：

(0) **成功**

(-1) **不成功**

参考函数：

ReadAxisPos_6050

GoHome_6050

轴回 Home 点

函数编号:24

函数原型：

C: short GoHome_6050(unsigned short Board_NO,unsigned short Axis_No,
long goHomeVel,long LeaveHomeVel,long LeaveHomePos);

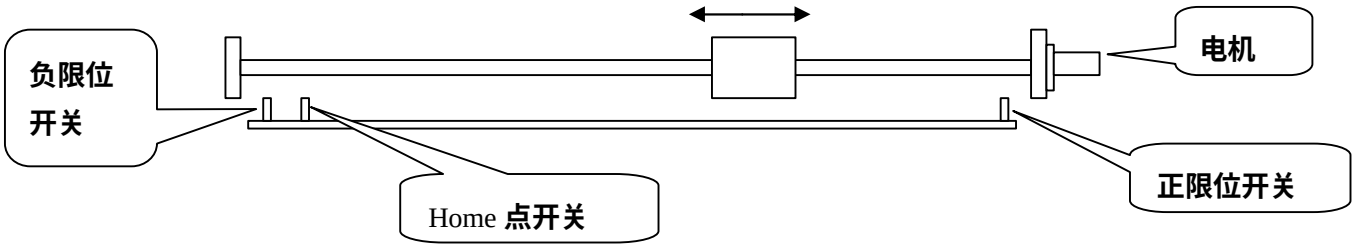
VB: Declare Function GoHome_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByVal Axis_NO As Integer, ByVal goHomeVel As Long, ByVal LeaveHomeVel As Long, ByVal LeaveHomePos As Long) As Integer

Delphi: function GoHome_6050(Board_NO : smallInt; Axis_No : Word; goHomeVel : LongInt; LeaveHomeVel : LongInt; LeaveHomePos : LongInt) : SmallInt; stdcall;

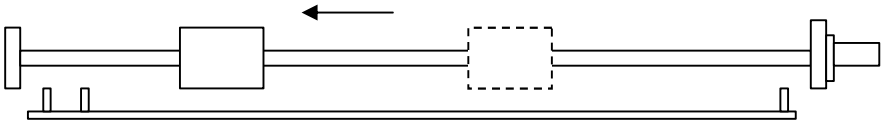
解释：

这个函数将使轴进行真实回 Home **点操作。回** Home **点过程示意如下：**

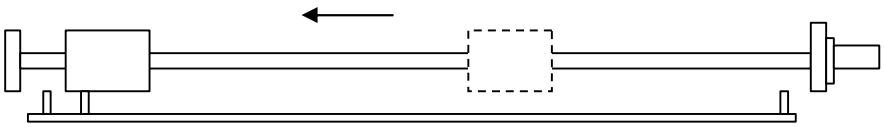
连线图示意：



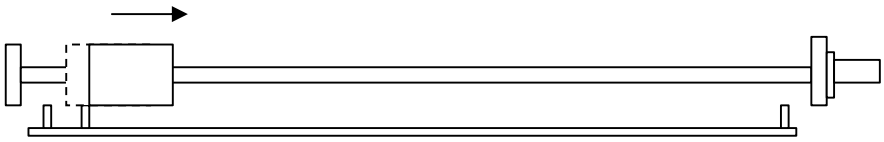
步骤 1**：从当前位置向** Home **点移动**



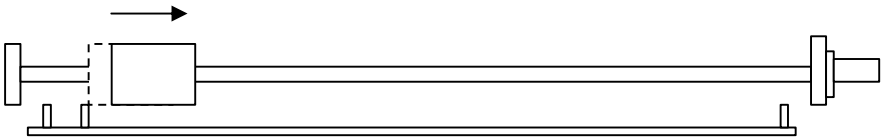
步骤 2**：压上** Home **点**



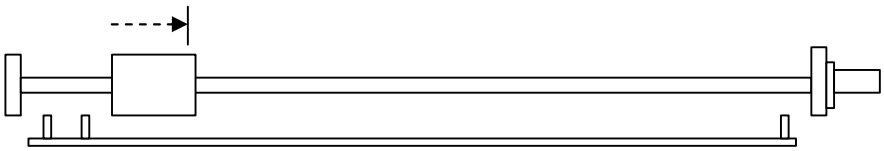
步骤 3：脱离 Home 点



步骤 4：脱离 Home 点 一段距离



步骤 5：停止运动，轴位置计数器清零，GoHome 完成



在回 Home 点时，回 Home 点方向、速度，脱离 Home 点方向、速度及距离均可由参数设定。只有在轴 Home 点有效时（由函数 SetAxisIO_6050 设定），GoHome 函数才有效。

注意: 在 GoHome_6050 命令之前，确保有 SetAxisAcc_6050 （设置轴加速度）命令

轴正在运行时是否可调用：否

参数：

Board_NO : 0－3, **板号**

Axis_No : 0－3, **轴号**

goHomeVel : **轴回 Home 点速度; 范围: 长整型数; 可正负； 单位: Hz，轴的脉冲频率。决定步骤 1 的方向和速度。**

LeaveHomeVel: **轴离开 Home 点速度; 范围: 长整型数; 可正负；单位: Hz，轴的脉冲频率。决定步骤 3、4 的方向和速度。该速度值一般要选择小一点。**

LeaveHomePos: **轴离开 Home 点距离; 范围: 长整型数,必须是正值; 单位: 脉冲个数。决定步骤 4 的距离。**

返回码：

(0) **成功**

(-1) **非法参数**

参考函数：

ReadAxisPos_6050、Home_6050、SetAxisIO_6050

ReadAxisPos_6050

返回轴当前实际位置

函数编号:2

函数原型：

C: long ReadAxisPos_6050(short Board_NO,unsigned short Axis_No);

VB: Declare Function ReadAxisPos_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByVal Axis_NO As Integer) As Long

Delphi: function ReadAxisPos_6050(Board_NO : smallInt; Axis_No : Word) : LongInt; stdcall;

解释：

返回轴的当前实际位置。例如：轴从零位正向旋转了 100 **个步距角，又负向旋转了** 40 **个步距角，则调用** ReadAxisPos_6050 **函数将返回** 60**（轴的当前位置）。**

轴正在运行时是否可调用：是

参数：

Board_NO **卡号，可能值** 0，1，2，3

Axis_No **轴号，可能值** 0，1，2，3

返回码：

轴的位置; 范围，长整型数； 单位，脉冲个数。

当返回轴位置= -2147483648 **时,表示不成功,有错误产生。**

参考函数：

ReadAxisVel_6050

ReadAxisVel_6050

返回轴的当前速度

函数编号:26

函数原型：

C: long ReadAxisVel_6050(short Board_NO,unsigned short Axis_No);

VB: Declare Function ReadAxisVel_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByVal Axis_NO As Integer) As Long

Delphi: function ReadAxisVel_6050(Board_NO : smallInt; Axis_No : Word) : LongInt; stdcall;

解释：

返回轴的当前实际速度。

轴正在运行时是否可调用：是

参数：

Board_NO **卡号，可能值** 0，1，2，3

Axis_No **轴号，可能值** 0，1，2，3

返回码：

轴的当前速度； 范围，长整型数；单位，Hz**，轴的脉冲频率。**

当返回轴位置= -2147483648 **时,表示不成功,有错误产生。**

参考函数：

ReadAxisPos_6050

ReadAxisState_6050

读取轴的状态

函数编号:27

函数原型：

C: short ReadAxisState_6050(short Board_NO,unsigned short Axis_No);

VB: Declare Function ReadAxisState_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByVal Axis_NO As Integer) As Integer

Delphi: function ReadAxisState_6050(Board_NO : smallInt; Axis_No : Word) : SmallInt; stdcall;

解释：

这个函数将返回轴的状态。

轴正在运行时是否可调用：是

参数：

Board_NO **卡号，可能值** 0，1，2，3

Axis_No **轴号，可能值** 0，1，2，3

返回码：

1: **轴在运动中**

0: **轴在停止状态,轴位置到达,或被清零**(Home_6050 **命令**)

-1: **轴** GoHome OK

-2: **轴在** GoHome **中暂时停止**

-3: **轴被** StopAxis_6050 **命令停止**

-4: **轴被** AbortAxis_6050 **命令停止**

-5: **轴被** CeaseAxis_6050 **命令停止**

-6: **轴被 插补 命令停止**

-7: **轴被正限位停止**

-8: **轴被负限位停止**

-9: **轴位置寄存器溢出被停止**

-10: **轴被外部** IO **急停停止**

-11: **轴在跟随模式下速度为** 0

255：**读错误**

参考函数：

SetAxisTEC_6050

设置轴螺距反向间隙补偿

函数编号:73

函数原型：

C: short SetAxisTEC_6050 (short Board_NO,unsigned short Axis_No,long ErrorV,short TimeNum);

VB: Declare Function ReadAxisState_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByVal Axis_NO As Integer) As Integer

Delphi: function SetAxisTEC_6050(Board_NO : smallInt; Axis_No : smallInt; ErrorV : LongInt; TimeNum : shortInt)shortInt; stdcall;

解释：

该函数将设轴螺距反向间隙补偿值。一般原则是当轴回零点完成后（Home_6050 **或** GoHome_6050 **命令之后），调用该函数来设轴的反向间隙补偿值。在调用该函数那一刻，**6050 **卡自动判断轴方向，当轴运动方向与该时刻方向相反时，**6050 **卡将开始进行螺距反向间隙补偿。注意：**Home_6050 **命令或** GoHome_6050 **命令自动取消轴螺距反向间隙补偿。**

轴正在运行时是否可调用：否

参数：

Board_NO : **卡号，可能值** 0，1，2，3

Axis_No : **轴号，可能值** 0，1，2，3

ErrorV : **轴螺距反向间隙补偿值**; **范围**: 0-32767; **单位**: **脉冲个数。**

TimeNum : **补偿所用时间**; **范围**: 1-20; **单位**: **毫秒。一般原则是反向间隙补偿值越大，所用时间应该越长。**

返回码：

0 **或**-1, -1=**不成功**,0=**成功**

参考函数：

SetAxisOffset_6050

设置轴位置偏移值

函数编号:84

函数原型：

C: short SetAxisOffset_6050 (short Board_NO,unsigned short Axis_No ,long Offset);

VB: Declare Function ReadAxisState_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByVal Axis_NO As Integer) As Integer

Delphi: function SetAxisOffset_6050(Board_NO : smallInt; Axis_No : smallInt; Offset : LongInt)shortInt; stdcall;

解释：

该函数将预设轴位置计数器的值。Home_6050 **命令或** GoHome_6050 **命令都不能将轴位置的偏移值（预设轴位置寄存器的值）清零，要清零，就设为** 0 **即可。**

在 6050 **卡内部，函数** ReadAxisPos_6050 **返回值将是：**

轴位置实际计数器值 + **轴位置的偏移值。**

所以当 轴位置实际计数器值 溢出（6050 **卡有报错），或两者之和溢出时（**6050 **卡不会报错），**ReadAxisPos_6050 **返回值将不正确。**

（所谓溢出指位置值超出长整型数 -2,147,483,647 **至** +2,147,483,647 **的范围）**

轴正在运行时是否可调用：否

参数：

Board_NO : **卡号，可能值** 0，1，2，3

Axis_No : **轴号，可能值** 0，1，2，3

offset : **轴位置的偏移值**; **范围**: **长整型数**; **单位**: **脉冲个数。**

返回码：

0 **或**-1, -1=**不成功**,0=**成功**

参考函数：

ReadEncoderPos_6050

读取编码器位置

函数编号:10

函数原型：

C: long ReadEncoderPos_6050(short Board_NO);

VB: Declare Function ReadEncoderPos_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer) As Integer

Delphi: function ReadEncoderPos_6050(Board_NO : smallInt) : SmallInt; stdcall;

解释：

该函数返回编码器计数器值

轴正在运行时是否可调用：是

参数：

卡号，可能值 0，1，2，3

返回码：

编码器位置； 范围；长整型数； 单位， 脉冲个数。

参考函数：

HomeEncoder_6050

HomeEncode_6050

编码器位置清零

函数编号:11

函数原型：

C: short HomeEncode_6050(short Board_NO);

VB: Declare Function HomeEncode_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer) As Integer

Delphi: function HomeEncode_6050(Board_NO : smallInt) : SmallInt; stdcall;

解释：

该函数将编码器位置计数器清零。可以通过调用函数 ReadEncoderPos_6050 **来得到编码器当前位置计数器。**

轴正在运行时是否可调用：当有轴跟随编码器时不能，其它情况下可以

参数：

卡号，可能值 0，1，2，3

返回码：

(0) 成功

(-1) 不成功

参考函数：

ReadEncoderPos_6050

GetEnAuto0Flag_6050

获得编码器自动清零标志

函数编号:80

函数原型：

C: short GetEnAuto0Flag_6050 (short Board_NO);

VB: Declare Function HomeEncode_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer) As Integer

Delphi: function GetEnAuto0Flag_6050(Board_NO : smallInt)shortInt; stdcall;

解释：

该函数将获得编码器位置计数器是否被 6050 **卡自动清零。在执行** LM_LineFE_6050 **命令时，为了防止编码器计数器溢出，**6050 **卡可能对编码器计数器清零。**

轴正在运行时是否可调用：是

参数：

卡号，可能值 0，1，2，3

返回码：

0:

1: 编码器计数器曾经被 6050 卡自动清零过

255: 命令不成功

参考函数：

ReadEncoderPos_6050

LM_LineFE_6050

ResetEn0Flag_6050

ResetEn0Flag_6050

编码器自动清零标志置 0

函数编号:79

函数原型：

C: short ResetEn0Flag_6050 (short Board_NO);

VB: Declare Function HomeEncode_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer) As Integer

Delphi: function ResetEn0Flag_6050(Board_NO : smallInt)shortInt; stdcall;;

解释：

当用函数 GetEnAuto0Flag_6050 **得到编码器位置计数器被自动清零（编码器自动清零标志位在** 6050 **卡中被置** 1**），用该函数将编码器自动清零标志位置** 0**。**

轴正在运行时是否可调用：是

参数：

卡号，可能值 0，1，2，3

返回码：

0 或-1， -1=不成功,0=成功

参考函数：

ReadEncoderPos_6050

LM_LineFE_6050

GetEnAuto0Flag_6050

ReadFirmwareVersion_6050

读取 6050 控制卡固件版本号

函数编号:28

函数原型：

C: short ReadFirmwareVersion_6050(short Board_NO);

VB: Declare Function ReadFirmwareVersion_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer) As Integer

Delphi: function ReadFirmwareVersion_6050(Board_NO : smallInt;) : SmallInt; stdcall;

解释：

该函数返回控制卡上的 DSP **固件程序版本。**

轴正在运行时是否可调用：是

参数：

卡号，可能值 0，1，2，3

返回码：

(0) 不成功

(>1) 成功。比如: 10 **即代表固件为** 1.0 **版本**

ReadDllVersion_6050

读取 6050 控制卡动态链接库版本号

函数编号:67

函数原型：

C: short ReadDllVersion_6050();

VB: Declare Function ReadDllVersion_6050 Lib "dfjzh6050dll.dll" () As Integer

Delphi: function ReadDllVersion_6050( ) : SmallInt; stdcall;

解释：

该函数返回控制卡动态链接库的版本号。

轴正在运行时是否可调用：是

参数：

（无）

返回码：

(0) 不成功

(>1) 成功。比如: 10 即代表动态链接库的版本为 1.0 版本

GetDriverVersion_6050

读取 6050 卡的驱动版本号

函数编号:70

函数原型：

C: int GetDriverVersion_6050 (unsigned short Board_NO);

VB: Declare Function GetDriverVersion_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer) As Integer

Delphi: function GetDriverVersion_6050 (Board_NO : smallInt;) : SmallInt; stdcall;

解释：

该函数返回控制卡动态链接库的版本号。

轴正在运行时是否可调用：是

参数：

Board_NO : 0－3, **板号**

返回码：

(0) **不成功**

(>1) **成功。比如**: 10 **即代表** 6050 **卡的硬件驱动版本为** 1.0 **版本**

ReadIO_6050

读板上数字 IO（输入）点信号状态

函数编号:14

函数原型：

C: long ReadIO_6050(unsigned short Board_NO);

VB: Declare Function ReadIO_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer) As Long

Delphi: function ReadIO_6050(Board_NO : smallInt;) : LongInt; stdcall;

解释：

读取通用输入点 I1-I20 **状态,返回数值转换为二进制时的** 0-19 **位有效。**

轴正在运行时是否可调用：是

参数：

Board_NO : 0－3, **板号**

返回码：

(>=0) 成功，0-19 **位为通用输入点** I1-I20 **状态**

(-1) 不成功

参考函数：

ReadIOBit_6050

WriteIo_6050

ReadIOBit_6050

按位读板上数字 IO（输入）点信号状态

函数编号:14

函数原型：

C: short ReadIOBit_6050(unsigned short Board_NO,short Index);

VB: Declare Function ReadIO_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer) As Long

Delphi: function ReadIOBit_6050(Board_NO : smallInt; Index : shortInt)shortInt; stdcall;

解释：

按位读取通用输入点 I1-I20 **状态。**

轴正在运行时是否可调用：是

参数：

Board_NO : 0－3, **板号**

Index : 1 - 20, **输入点索引号**

返回码：

0-1 输入点状态

(-1) 不成功

参考函数：

ReadIO_6050

WriteIo_6050

WriteIo_6050

置板上数字 IO（输出）点信号

函数编号:16

函数原型：

C: short WriteIo_6050(short Board_NO,unsigned short IO_V);

VB: Declare Function WriteIo_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByVal IO_V As Integer) As Integer

Delphi: function WriteIo_6050(Board_NO : smallInt; IO_V : Word;) : SmallInt; stdcall;

解释：

这个函数将置板上 8 路通用数字输出点状态。达林顿输出方式： 1：导通；0：断开。继电器方式：1：常开点闭合，常闭点断开；0：常开点断开，常闭点闭合。

轴正在运行时是否可调用：是

参数：

Board_NO **卡号，可能值** 0，1，2，3

IO_V **输出点值**,WORD **型，低** 8 **位有效,对应** O1-O8,8 **个输出点状态**

返回码：

(0) **成功**

(-1) **参数非法**

参考函数：

ReadIO_6050

WriteIoBit_6050

按位置板上数字 IO（输出）点信号

函数编号:81

函数原型：

C: short WriteIoBit_6050(short Board_NO,unsigned short IO_V,short Index);

VB: Declare Function WriteIo_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByVal IO_V As Integer) As Integer

Delphi: function WriteIoBit_6050(Board_NO : smallInt; IO_V: smallInt; Index : shortInt)shortInt; stdcall;

解释：

这个函数将置板上 8 路通用数字输出点状态。达林顿输出方式： 1：导通；0：断开。
继电器方式：1：常开点闭合，常闭点断开；0：常开点断开，常闭点闭合。

轴正在运行时是否可调用：是

参数：

Board_NO **：卡号，可能值** 0，1，2，3
IO_V **：** 0 - 1，**输出点值**
Index **：** 1 - 8, **输出点索引号**

返回码：

(0) **成功**
(-1) **参数非法**

参考函数：

ReadIO_6050

ReadOs_6050

读取输出点 O1-O8 状态

函数编号:82

函数原型：

C: short ReadOs_6050 (short Board_NO);

VB: Declare Function WriteIo_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByVal IO_V As Integer) As Integer

Delphi: function ReadOs_6050(Board_NO : smallInt)shortInt; stdcall;

解释：

读取通用输出点 O1-O8 **状态，返回数值转换为二进制时的** 0-7 **位有效。由于可以用插补模式来设置输出点（函数** LM_IOOut_6050**），而该模式的输出点状态输出不是立即生效的，所以可以用该函数来随时检测输出点状态。**

轴正在运行时是否可调用：是

参数：

Board_NO **：卡号，可能值** 0，1，2，3

返回码：

0-256 **低** 8 **位有效,对应** O1-O8,8 **个输出点状态**

(-1) **参数非法**

参考函数：

ReadOsBit_6050

ReadOsBit_6050

按位读取输出点 O1-O8 状态

函数编号:83

函数原型：

C: short ReadOsBit_6050 (short Board_NO,short Index);

VB: Declare Function WriteIo_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByVal IO_V As Integer) As Integer

Delphi: function ReadOsBit_6050(Board_NO : smallInt; Index : shortInt)shortInt; stdcall;

解释：

按位读取通用输出点 O1-O8 **状态。由于可以用插补模式来设置输出点，由于可以用插补模式来设置输出点（函数** LM_IOOut_6050**），而该模式的输出点状态输出不是立即生效的，所以可以用该函数来随时检测输出点状态。**

轴正在运行时是否可调用：是

参数：

Board_NO **：卡号，可能值** 0，1，2，3

Index **：** 1 - 8, **输出点索引号**

返回码：

0 - 1 **对应输出点状态**

(-1) **参数非法**

参考函数：

ReadOs_6050

SetAxisIO_6050

设置轴 IO 是否有效

函数编号:15

函数原型：

C: short SetAxisIO_6050(unsigned short Board_NO,unsigned short Axis_No, short PLimit,short NLimit,short Home);

VB: Declare Function SetAxisIO_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByVal Axis_NO As Integer, ByVal Plimit As Integer, ByVal Nlimit As Integer, ByVal Home As Integer) As Integer

Delphi: function SetAxisIO_6050(Board_NO : smallInt; Axis_No: Word; PLimit : shortInt; NLimit : shortInt; Home : shortInt) : SmallInt; stdcall;

解释：

设置轴 IO **是否有效**

轴正在运行时是否可调用：否

参数：

Board_NO : 0－3, **板号**

Axis_No : 0－3, **轴号**

PLimit : 0 **或** 1, 0=**该轴正限位无效**,1=**该轴正限位有效**

NLimit : 0 **或** 1, 0=**该轴负限位无效**,1=**该轴负限位有效**

Home : 0 **或** 1, 0=**该轴** Home **点无效**,1=**该轴** Home **点有效**

返回码：

(0) **成功**

(-1) **不成功**

参考函数：

io_read6030

Set_Emergency_Stop_6050

设置 IO 急停信号

函数编号:29

函数原型：

C: short Set_Emergency_Stop_6050(unsigned short Board_NO,unsigned short Mask, short Mode);

VB: Declare Function Set_Emergency_Stop_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByVal Mask As Integer, ByVal Mode As Integer) As Integer

Delphi: function Set_Emergency_Stop_6050(Board_NO : Word; Mask : Word; Mode : smallInt) : SmallInt; stdcall;

解释：

该函数设置 I1-I8 **中的某一个输入点为急停信号输入点。**6050 **卡在初始化后，急停信号输入点是无效的。**

轴正在运行时是否可调用：是

参数：

Board_NO : 0－3, **板号**

Mask : 0、1－8, **急停信号与通用输入点** I1-I8 **相联**,0 **表示无效**,6050 **卡上电时 缺省为无效。**

注意: 输入点必须是 1-8 点，因为 1-8 输入点支持中断模式。

Mode : (**保留**)

返回码：

(0) **成功**

(-1) **不成功**

参考函数：

PwmOut_6050

PWM 脉冲输出

函数编号:12

函数原型：

C: short PwmOut_6050(short Board_NO,long frequency,float Pulse_Highf);

VB: Declare Function PwmOut_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByVal frequency As Long, ByVal Pulse_Highf As Single) As Integer

Delphi: function PwmOut_6050(Board_NO : smallInt; frequency : LongInt; Pulse_Highf:Single) : SmallInt; stdcall;

解释：

控制卡上包含一路 PWM **输出，调用该函数即可使其输出设定的频率和占空比的** PWM **信号。**

轴正在运行时是否可调用：是

参数：

Board_NO : 0－3, **板号**

frequency: 18-1500000; PWM **输出频率**; **单位**: **赫兹**(Hz)

Pulse_Highf: **占空比**,**范围**:0.0-1.0; **精度**:**不小于** 1%

返回码：

(0) **成功**

(-1) **参数非法**

参考函数：

PwmOut2_6050

PwmStop_6050

DAOut_6050

LM_PWM_6050

PwmOut2_6050

PWM 脉冲输出

函数编号:12

函数原型：

C: short PwmOut2_6050(short Board_NO,long frequency,float Pulse_Highf);

VB: Declare Function PwmOut2_6050 Lib "dfjzh6050dll.dll" (Board_NO as integer , frequency as long , Pulse_Highf as single) as integer

Delphi: function PwmOut2_6050(Board_NO : smallInt; frequency : LongInt; Pulse_Highf: Single) : SmallInt; stdcall;

解释：

控制卡上包含一路 PWM **输出，调用该函数即可使其输出设定的频率和占空比的** PWM **信号。**

轴正在运行时是否可调用：是

参数：

Board_NO : 0－3, **板号**

frequency: 18-1500000; PWM **输出频率**; **单位**: **赫兹**(Hz)

Pulse_Highf: **高电平脉冲宽度**,**单位：毫秒**;

精度（分辨率）:0.00000667 **当** frequency=2288 **；**

0.000853 **当** frequency <2288

返回码：

(0) **成功**

(-1) **参数非法**

参考函数：

PwmOut_6050

PwmStop_6050

DAOut_6050

LM_PWM_6050

PwmStop_6050

PWM 输出停止

函数编号:13

函数原型：

C: short PwmStop_6050(short Board_NO);

VB: Declare Function PwmStop_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer) As Integer

Delphi: function PwmStop_6050(Board_NO : smallInt) : SmallInt; stdcall;

解释：

断开 PWM **输出。**

轴正在运行时是否可调用：是

参数：

Board_NO **卡号，可能值** 0，1，2，3

返回码：

(0) **成功**

(-1) **参数非法**

参考函数：

PwmOut_6050

PwmOut2_6050

DAOut_6050

LM_PWM_6050

DAOut_6050

DA(数模转换)模拟量输出

函数编号:68

函数原型：

C: short DAOut_6050(unsigned short Board_NO,double DA_Avlue);

VB: Declare Function DAOut_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByVal DA_Avlue As Double) As Integer

Delphi: function DAOut_6050(Board_NO : smallInt; DA_Avlue : Double) : SmallInt; stdcall;

解释：

轴正在运行时是否可调用：是

参数：

Board_NO : 0－3, **板号**

DA_Avlue : -10 - +10 V; **精度**: 1/6000; **即**: **精度大于** 12 **位**(4096),**不到** 13 **位**(8192).
该 DA **输出与** PWM **输出占用同一个硬件资源,通过** 6050 **端子板上的跳线选择是** DA **输出还是** PWM **输出**.

返回码：

(0) **成功**

(-1) **参数非法**

参考函数：

PwmOut_6050

PwmStop_6050

LM_PWM_6050

LM_PWM_6050

插补模式的 PWM 输出

函数编号:61

函数原型：

C: short LM_PWM_6050(unsigned short Board_NO,long frequency,
double Pulse_Highf,double Speed,long LineNO);

VB: Declare Function LM_PWM_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByVal frequency As Long, ByVal Pulse_Highf As Double, ByVal Speed As Double, ByVal LineNO As Long) As Integer

Delphi: function LM_PWM_6050(Board_NO : smallInt; frequency : LongInt;Pulse_Highf : Double; Speed : Double; LineNO: LongInt) : SmallInt; stdcall;

解释：

插补模式的 PWM **输出，占空比随速度变化。该命令将把插补数据送入** 6050 **卡的插补缓存器中**,6050 **卡的插补缓存器总共有** 16 **行。当插补引擎运行到该行时将根据参数决定** PWM **输出**.

轴正在运行时是否可调用：是

参数：

Board_NO : 0－3, **板号**

frequency :18-1500000; PWM **输出频率**; **单位**: **赫兹**(Hz)
当为 0 **时**,**停止** PWM **输出**

Pulse_Highf : **占空比**,**范围**:0.0-1.0; **精度**:**不小于** 1%
该值对应插补速度达到 Speed **所设速度时的最大占空比**,**当插补速度大于** Speed **所设速度时**,**也按该参数输出**.

Speed : **占空比随动的最大插补速度**,**单位**: **用户单位/分钟**
当该值为 0 **时**,**表明不是随动模式**,**当插补引擎运行到该行时将根据** frequency,Pulse_Highf **参数值立即** PWM **输出**.

LineNO : **插补行号**,**用户自由设定**

返回码：

(0) 成功

(-1) 不成功

LM_SetSysPara_6050

设插补系统参数

函数编号:59

函数原型：

C： short LM_SetSysPara_6050(unsigned short Board_NO,double MinLength, double MinSpeed,double ArcError);

VB： Declare Function LM_SetSysPara_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByVal MinLength As Double, ByVal MinSpeed As Double, ByVal ArcError As Double) As Integer

Delphi: function LM_SetSysPara_6050(Board_NO : Word; MinLength : Double; MinSpeed : Double; ArcError : Double) : SmallInt; stdcall;

解释：

设置插补运算中的最小长度、速度和圆弧误差值。

在插补引擎正在运行当中是否可调用：否

参数：

Board_NO : 0－3, **板号**

MinLength: **系统最小长度,单位: 用户单位, 缺省值**: 0.001;**在直线或圆弧插补时,直线或圆弧长度不能小于该设定值**

MinSpeed : **系统最小速度,单位: 用户单位/分钟, 缺省值**: 0.001;**在直线或圆弧插补时,直线或圆弧插补速度不能小于该设定值**

ArcError : **系统最大圆弧误差,单位: 用户单位, 缺省值**: 0.2;**在圆弧插补时,圆弧起点和终点半径误差不能大于该设定值**

返回码：

大于 0 **成功**

小于 0 **操作失败**

参考函数：

GetErrorNo_6050

LM_SetSpeedPri_6050

设置插补速度速度优先还是精度优先

函数编号 ： 58

函数原型：

C： short LM_SetSpeedPri_6050(unsigned short Board_NO,short PriorityFlag);

VB： Declare Function LM_SetSpeedPri_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByVal PriorityFlag As Integer) As Integer

Delphi: function LM_SetSpeedPri_6050(Board_NO : Word; PriorityFlag: smallInt) : SmallInt; stdcall;

解释：

该函数确定插补运算的两种效果，速度优先意味着运行相对更平滑，而精度优先意味着控制相对更精确

在插补引擎正在运行当中是否可调用：否

参数：

Board_NO : 0－3, **板号**

PriorityFlag : 0-1; 1:**插补速度优先**, 0:**插补精度优先。** **缺省值**: 1 **插补速度优先**

返回码：

(0) 成功

(-1) 不成功

参考函数：

LM_SetMinVel_6050

设置插补最小速度

函数编号:47

函数原型：

C： short LM_SetMinVel_6050(unsigned short Board_NO,
double MinLineVel,
double MinArcVel);

VB： Declare Function LM_SetMinVel_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByVal MinLineVel As Double, ByVal MinArcVel As Double) As Integer

Delphi: function LM_SetMinVel_6050(Board_NO : Word; MinLineVel : Double; MinArcVel : Double) : SmallInt; stdcall;

解释：

在插补引擎正在运行当中是否可调用：否

参数：

Board_NO : 0－3, **板号**

MinLineVel: **直线最小插补速度**, **单位**: **用户单位/分钟**, **缺省值**: 30

MinArcVel : **圆弧最小插补速度**, **单位**: **用户单位/分钟**, **缺省值**: 30

返回码：

(0) 成功

(-1) 不成功

参考函数：

LM_SetMaxVel_6050

LM_SetMaxVel_6050

设置插补最大速度

函数编号:57

函数原型：

C： short LM_SetMaxVel_6050(unsigned short Board_NO,
double MaxLineVel,
double MaxArcVel);

VB： Declare Function LM_SetMaxVel_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByVal MaxLineVel As Double, ByVal MaxArcVel As Double) As Integer

Delphi: function LM_SetMaxVel_6050(Board_NO : Word; MaxLineVel : Double; MaxArcVel : Double) : SmallInt; stdcall;

解释：

在插补引擎正在运行当中是否可调用：否

参数：

Board_NO : 0－3, **板号**

MaxLineVel: **直线最大插补速度**, **单位**: **用户单位/分钟**, **缺省值**: 40000

MaxArcVel : **圆弧最大插补速度**, **单位**: **用户单位/分钟**, **缺省值**: 40000

返回码：

(0) 成功

(-1) 不成功

参考函数：

LM_SetMinVel_6050

LM_SetParaAngle_6050

设置插补引擎的角度参数

函数编号:48

函数原型：

C： short LM_SetParaAngle_6050(short Board_NO,short angle1,short angle2);

VB： Declare Function LM_SetParaAngle_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByVal angle1 As Integer, ByVal angle2 As Integer) As Integer

Delphi: function LM_SetParaAngle_6050(Board_NO : Word; angle1: smallInt, angle2: smallInt) : SmallInt; stdcall;

解释：

该函数将设置插补引擎内部参数。这些参数将影响两两直线插补线段之间的运动平滑性。

在插补引擎正在运行当中是否可调用：否

参数：

Board_NO : 0－3, **板号**

ang1：: **值范围：**0-90，**且** ang1<ang2。**插补引擎初始化时该参数的却省值为** 20。

ang2：: **值范围：**0-90，**且** ang1<ang2。**插补引擎初始化时该参数的却省值为** 35。

返回码：

大于 0 **成功**

小于 0 **非法参数**

参考函数：

LM_SetXAxis_6050

将实际轴与插补引擎的 X 轴相匹配

函数编号:32

函数原型：

C： short LM_SetXAxis_6050(short Board_NO,short Axis_NO,
double factor_c_t,double delta);

VB： Declare Function LM_SetXAxis_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByVal Axis_NO As Integer, ByVal factor_c_t As Double, ByVal delta As Double) As Integer

Delphi: function LM_SetXAxis_6050(Board_NO : smallInt; Axis_NO : shortInt; factor_c_t : double; delta : double) : SmallInt; stdcall;

解释：

该函数将实际轴与插补引擎虚拟的 X 轴相匹配。插补引擎虚拟了一个 XYZW 坐标系，在执行插补运算前，用户必须将这个虚拟的 XYZW 坐标系实际化，可对虚拟的 XYZW 坐标的任意两轴直至四轴全部实际化。

同理：也有 YZW 轴的匹配函数：

short LM_SetYAxis_6050 (short Board_NO,short Axis_NO,
double factor_c_t,double delta);

short LM_SetZAxis_6050 (short Board_NO,short Axis_NO,
double factor_c_t,double delta);

short LM_SetWAxis_6050 (short Board_NO,short Axis_NO,
double factor_c_t,double delta);

在插补引擎正在运行当中是否可调用：否

参数：

Board_NO: **卡号，可能值 0，1，2，3**

axis_NO： **轴号，可能值为：0，1，2，3。**

factor_c_t: **轴的脉冲当量。即：电机转动一个脉冲的步距角时，该轴移动多少用户单位。**

例如，某一直线轴通过滚珠丝杠与电机直联，该滚珠丝杠的螺距是 5 毫米，而电机转一圈需 2000 个脉冲，则：轴的脉冲当量=5/2000=0.0025；
0.0025 的含义是：一个脉冲将使该轴移动 0.0025 毫米。
轴的脉冲当量很重要，轴的脉冲当量确定后，本书以下有关轴的直线、圆弧插补的距离单位就确定。上例中如果：factor_c_t=0.0025，则该轴的距离单位就是毫米。

delta： 该轴动态准确定位的误差值。单位为用户单位。
在插补运算时，在每行插补运动开始之前，系统将以该值为依据来判断轴位置误差是否超限，如果超越极限，插补引擎将停止工作并报警。报警号为 3（插补误差超限）。

返回码：

(0) 成功
(-1) 不成功

参考函数：

LM_OffXAxis_6050

LM_OffXAxis_6050

撤消插补引擎的 X 轴匹配

函数编号:33

函数原型：

C： short LM_OffXAxis_6050(short Board_NO);

VB： Declare Function LM_OffXAxis_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer) As Integer

Delphi: function LM_OffXAxis_6050(Board_NO : smallInt) : SmallInt; stdcall;

解释：

该函数将撤消插补引擎的 X **轴匹配。它是函数** LM_SetIpol_X_Axis6030 **的反操作。**

同理：也有 YZW **轴的撤消匹配函数：**

short LM_OffYAxis_6050(short Board_NO);

short LM_OffZAxis_6050(short Board_NO);

short LM_OffWAxis_6050(short Board_NO);

在插补引擎正在运行当中是否可调用：否

参数：

Board_NO **卡号，可能值** 0，1，2，3

返回码：

(0) 成功

(-1) 不成功

参考函数：

LM_SetIpol_X_Axis6030

LM_SetACCDec_6050

设置插补加速度和插补减速度

函数编号:40

函数原型：

C： short LM_SetACCDec_6050(short Board_NO,long acceleration,
long deceleration);

VB： Declare Function LM_SetACCDec_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByVal acceleration As Long, ByVal deceleration As Long) As Integer

Delphi: function LM_SetACCDec_6050(Board_NO : smallInt; acceleration : LongInt; deceleration : LongInt) : SmallInt; stdcall;

解释：

该函数将设置插补加速度和插补减速度。

在插补引擎正在运行当中是否可调用：否

参数：

Board_NO	0－3, **板号**
acceleration	**插补加速度，用户单位/秒平方。范围：**1-10000。**缺省值**: 500
Deceleration	**插补减速度，用户单位/秒平方。范围：**1-10000。**缺省值**: 500

返回码：

0	**成功**
(<0)	**不成功**

参考函数：

LM_SetSAccePower_6050

LM_SetSAccePower_6050

设置插补 S 型加速度(1,2,3,4)指数

函数编号:71

函数原型：

C： short LM_SetSAccePower_6050 (short Board_NO,short PowerFlag);

VB： Declare Function LM_SetACCDec_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByVal acceleration As Long, ByVal deceleration As Long) As Integer

Delphi: function LM_SetSAccePower_6050(Board_NO : smallInt; PowerFlag:shortInt)shortInt; stdcall;

解释：

该函数设置插补加速度和插补减速度曲线参数。在控制卡初始化时，插补加减速度的指数曲线为 2。S **形加速度理论请看§** 2.4.5 **节有关内容。**

在插补引擎正在运行当中是否可调用：否

参数：

Board_NO ： 0－3, **板号**

PowerFlag ： 1-4 S **型加速度**(1,2,3,4)**指数，**1=**直线形加速度。系统缺省值：**2

返回码：

0 **成功**

(<0) **不成功**

参考函数：

LM_SetACCDec_6050

LM_Line_6050

写入直线插补命令

函数编号:42

函数原型：

C： short LM_Line_6050(short Board_NO,
double xPos,double yPos,double zPos,double wPos,
double Speed,long LineNO);

VB： Declare Function LM_Line_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByVal xPos As Double, ByVal yPos As Double, ByVal zPos As Double, ByVal wPos As Double, ByVal Speed As Double, ByVal LineNO As Long) As Integer

Delphi: function LM_Line_6050(Board_NO : smallInt; xPos : double; yPos : double; zPos : double; wPos : double; Speed : double; LineNO : LongInt) : SmallInt; stdcall;

解释：

该函数将直线插补命令写入插补缓冲区。

在插补引擎正在运行当中是否可调用：是

参数：

Board_NO : 0－3, **板号**

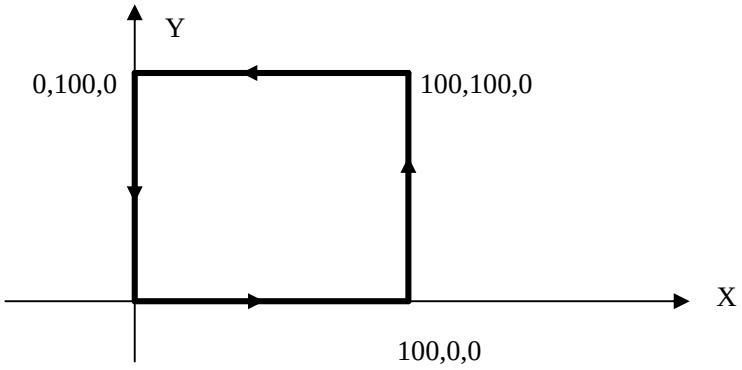
xPos、yPos、zPos、wPos： XYZW **轴的终点位置；绝对坐标，单位为用户单位。**

Speed： **插补速度，单位：用户单位/分钟。**

LineNO： **用户自定义行号。当插补引擎运行到该行时，用** LM_GetWorkLine6030 (long *LineNO)**函数可检测该行号值。**

例 1 **直线插补（**C **语言）：**

设 XYZ **轴从当前位置（**0，0，0，0**）直线移动到（**100，0，0，0**）位置，再直线移动到（**100，100，0，0**）位置，再直线移动到（**0，100，0，0**）位置，再直线移动到（**0，0，0，0**）位置；如下图：**



插补速度 F=200，**行号从** 100 **开始；**

则：

```
double F=200;
long    LineNO=100;
LM_Line_6050 (0, 100，0，0，0，F，LineNO)；
LM_Line_6050 (100，100，0，0，F，LineNO+1)；
LM_Line_6050 (0，100，0，0，F，LineNO+2)；
LM_Line_6050 (0，0，0，0，F，LineNO+3)；
LM_End_6050(0,LineNO+4);
LM_Start_6050(0, LineNO+5);
```

返回码：

(0)　　成功
(-1)　　不成功

参考函数：

LM_ArcCW_6050

LM_End_6050

LM_Start_6050

LM_LineMaxV_6050

直线插补

函数编号:69

函数原型：

C： short LM_LineMaxV_6050(unsigned short Board_NO,
double xPos,double yPos,double zPos,double wPos,
double Speed, double MaxSpeed, long LineNO);

VB： Declare Function LM_LineMaxV_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByVal xPos As Double, ByVal yPos As Double, ByVal zPos As Double, ByVal wPos As Double, ByVal Speed As Double, ByVal MaxSpeed As Double, ByVal LineNO As Long) As Integer

Delphi: function LM_LineMaxV_6050(Board_NO : Word; xPos : double; yPos : double; zPos : double; wPos : double; Speed : double; MaxSpeed : double; LineNO : LongInt) : SmallInt; stdcall;

解释：

该函数与 LM_Line_6050 **区别是：①可单独指定本行的最大速度，②执行到该行时不会有** PWM **信号输出（如果用户用到** LM_PWM_6050 **函数）。**

在插补引擎正在运行当中是否可调用：是

参数：

Board_NO : 0－3, **板号**

xPos : X **轴插补的终点位置(绝对值),单位为用户单位。**

yPos : y **轴插补的终点位置(绝对值),单位为用户单位。**

zPos : z **轴插补的终点位置(绝对值),单位为用户单位。**

wPos : w **轴插补的终点位置(绝对值),单位为用户单位。**

Speed : **插补速度,单位: 用户单位/分钟**

MaxSpeed : **本行插补最大速度,单位: 用户单位/分钟**

LineNO : **插补行号,用户自由设定**

返回码：

(0) 成功

(-1) 不成功

参考函数：

LM_LineMeasure_6050

直线插补及测量 IO 点输入

函数编号:65

函数原型：

C： short LM_LineMeasure_6050(unsigned short Board_NO,
double xPos,double yPos,double zPos,double wPos, short IO_Index,short Mode,
double Speed, long LineNO);

VB： Declare Function LM_LineMeasure_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByVal xPos As Double, ByVal yPos As Double, ByVal zPos As Double, ByVal wPos As Double, ByVal IO_Index As Integer, ByVal Mode As Integer, ByVal Speed As Double, ByVal LineNO As Long) As Integer

Delphi: function LM_LineMeasure_6050(Board_NO : Word; xPos : double; yPos : double; zPos : double; wPos : double; IO_Index : SmallInt; Mode: SmallInt;Speed : double; LineNO : LongInt) : SmallInt; stdcall;

解释：

直线插补并检测由参数 IO_Index **指示的输入点状态，如检测到输入点，就立即停止。该函数与** LM_Line_6050 **区别是：**

①本行的最大速度就是由 Speed **指定（函数** LM_SetSpeedRate_6050 **不能将其增速）。**
②执行到该行时不会有 PWM **信号输出（如果用户用到** LM_PWM_6050 **函数）。**
③当 6050 **卡检测到由** IO_Index **指定的输入点后立即停止（轴停止模式为** CeaseAxis_6050**）。**
④在本行执行完之前，不能再加入新的插补函数，如果加入，6050 **卡将报错，错误号为** 18**。**
⑤在本行执行当中，如果检测到输入点（由参数 IO_Index **指示），插补运动立即停止，这时插补状态（由函数** LM_GetState_6050 **获得）将为** 3**（被** LM_End_6050 **结束），函数** LM_GetMeasureState_6050 **将返回状态** 1**。**
⑥在本行执行当中，如果没有检测到输入点，在本行执行完后插补运动也将停止，这时插补状态（由函数 LM_GetState_6050 **获得）将为** 4**（被** LM_End_6050 **结束），函数** LM_GetMeasureState_6050 **将返回状态** 0**。**
⑦无论检测到输入点与否，在本行执行完后，如果还要进行插补运动，都要进行如下操作：

```
LM_CleanBuff_6050(…);          //清除插补缓存
LM_GetXStartPos_6050(…);       //重新获得插补开始位置
LM_GetYStartPos_6050(…);
```

LM_GetZStartPos_6050(…);

LM_GetWStartPos_6050(…);

在插补引擎正在运行当中是否可调用：是

参数：

Board_NO : 0－3, **板号**

xPos : X **轴插补的终点位置(绝对值),单位为用户单位。**

yPos : y **轴插补的终点位置(绝对值),单位为用户单位。**

zPos : z **轴插补的终点位置(绝对值),单位为用户单位。**

wPos : w **轴插补的终点位置(绝对值),单位为用户单位。**

IO_Index : 1-8; **通用** IO **点输入点序号**,

Mode : 1-4; (**目前** 2**、**4 **保留**)

1**：通用** IO **点输入点为** 1 **时，立即停止。**

3**：通用** IO **点输入点为** 0 **时，立即停止。**

Speed : **插补速度,单位**: **用户单位/分钟**

LineNO : **插补行号,用户自由设定**

返回码：

(0) 成功

(-1) 不成功

参考函数：

LM_GetMeasureState_6050

LM_GetMeasureState_6050

获得当前测量状态

函数编号:66

函数原型：

C： short LM_GetMeasureState_6050(short Board_NO);

VB： Declare Function LM_GetMeasureState_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer) As Integer

Delphi: function LM_GetMeasureState_6050(Board_NO : smallInt)shortInt; stdcall;

解释：

该函数将获得当前测量状态（由插补命令 LM_LineMeasure_6050 **产生）。**

在插补引擎正在运行当中是否可调用：是

参数：

Board_NO : 0－3, **板号**

返回码：

0: **没有测量到输入点。**

1: **检测到输入点**

255: **命令不成功**

注意：检测到输入点后，如果有新的直线插补测量运动开始执行，测量状态将自动转为 0 **状态。**

参考函数：

LM_LineFE_6050

插补模式的 轴跟随编码器位置命令

函数编号:77

函数原型：

C： short LM_LineFE_6050 (short Board_NO,
double xPos,double yPos,double zPos,double wPos,
double Rate,double Kp,short Mode,double Speed,long LineNO);

VB： Declare Function LM_Line_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByVal xPos As Double, ByVal yPos As Double, ByVal zPos As Double, ByVal wPos As Double, ByVal Speed As Double, ByVal LineNO As Long) As Integer

Delphi: function LM_LineFE_6050(Board_NO : smallInt; xPos : double; yPos : double; zPos :double; wPos : double; Rate : double; Kp : double; Mode : shortInt; Speed : double; LineNo : LongInt)shortInt; stdcall;

解释：

插补模式下的轴跟随编码器位置命令。该函数将 写入插补缓冲区。该函数的主要目的是为了实现螺纹切削。以数控铣床为例，编码器同轴安装在主轴上，用该函数可实现硬攻丝。

在插补引擎正在运行当中是否可调用：是

参数：

Board_NO : 0－3, **板号**

xPos、yPos、zPos、wPos： XYZW **轴的终点位置；绝对坐标，单位为用户单位。**
XYZW **轴中只有一个轴可以实现插补模式的编码器位置跟随，**6050 **卡根据各轴起点与终点位置之差是否为** 0**，自动判断那个轴跟随。如果有多于一个轴的位置之差不为** 0**，**6050 **卡将报错。**

Mode : 0,1; 0=**到达目标,立即停止跟随**; 0=**到达目标,不停止跟随,由后续命令决定**;
当 Mode=0 **时，跟随轴到达或超过终点位置时，跟随轴将立即停止（以** AbortAxis_6050 **模式）。当** Mode=1 **时，跟随轴到达或超过终点位置时，将由后续命令决定，细节如下：**

直线插补命令：同轴同方向，跟随轴将不停，自动过渡到直线插补；其它情况下，跟随轴将立即停止（以 AbortAxis_6050 **模式）。**
圆弧插补命令：跟随轴将立即停止（以 AbortAxis_6050 **模式）。**

其它插补命令：（包括 LM_End_6050 **命令）不停止跟随,再由后续命令决定。**

Rate ：**轴跟随比率：**

设：

A1=**跟随轴开始位置，**A2=**跟随轴实时位置，单位：用户单位**

B1=**编码器开始位置，**B2=**编码器实时位置，单位：脉冲数**

则：

$$Rate = \frac{A2 - A1}{B2 - B1}$$

可正负; **范围**: **绝对值**

0.001×**跟随轴脉冲当量 至** 1000×**跟随轴脉冲当量**;

单位: **无**; **分辨率**: 0.001×**跟随轴脉冲当量**

B1=**编码器开始位置：**

6050 **卡根据前面是否有未完成的*轴跟随编码器位置命令* 对编码器进行清零操作。如果没有，**6050 **卡将自动清零编码器计数器，防止计数器溢出。是否自动清零可由函数** GetEnAuto0Flag_6050 **探知。**

Kp ：**跟随** PID **调解系数**; **范围**: 0.001-1000; **单位**: **无**; **分辨率**: 0.001

Speed ：**插补预期速度,也应该是本行插补最大速度,单位**: **用户单位/分钟**

实际该行的插补速度将由 Rate **参数和编码器所反馈的速度决定。**

LineNO ：**插补行号,用户自由设定**

例 1 **用该函数实现数控铣床的硬攻丝（**C **语言）：**

设主轴编码器线数为 2500 **线，在** 6050 **卡中将四倍频，为：**10000 **线；螺距为** 0.8 **；插补速度** F=200**，行号从** 100 **开始；**

则：

```
double F=200;
long   LineNO=100;
double Rate=-1*0.8/10000;

LM_Line_6050 (0，0，0，0，0，F，LineNO)；
LM_Line_6050 (0，0，-5，0，F，LineNO+1)；
```

```
LM_LineFE_6050 (0，0，-25，0，1，Rete，0.1，F，LineNO+2)；//Z 轴-5 至-25 攻丝
LM_IOOut_6050 (0,……);                                    //控制主轴反转
LM_LineFE_6050 (0，0，-5，0，0，Rete，0.1，F，LineNO+4)； // Z 轴-25 至-5 回退
LM_End_6050(0,LineNO+5);
LM_Start_6050(0, LineNO+6);
```

返回码：

(0)　　成功
(-1)　　不成功

参考函数：

LM_ArcCW_6050
LM_End_6050
LM_Start_6050

LM_GetFEnState_6050

获得轴编码器跟随已达到目标状态

函数编号:78

函数原型：

C： short LM_GetFEnState_6050 (short Board_NO);

VB： Declare Function LM_GetMeasureState_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer) As Integer

Delphi: function LM_GetFEnState_6050(Board_NO : smallInt)shortInt; stdcall;

解释：

该函数将获得轴编码器跟随已达到或超过目标状态（由插补命令 LM_LineFE_6050 **产生）。**

在插补引擎正在运行当中是否可调用：是

参数：

Board_NO : 0－3, **板号**

返回码：

0: **跟随还没有达到目标**

1: **跟随已达到目标**

255: **命令不成功**

注意：跟随已达到目标，如果有新的插补模式下的轴跟随编码器位置命令开始执行，跟随状态将自动转为 0 **状态。**

参考函数：

LM_LineFE_6050

LM_ArcCW_6050

写入圆弧/螺旋线插补命令

函数编号:43、44

函数原型：

C
```
short LM_ArcCW_6050(short Board_NO,
double xPos,double  yPos,double zPos,double wPos,
double xcPos,double ycPos,
double Speed,
long  LineNO)
```

VB：
```
Declare Function LM_ArcCW_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByVal xPos As Double, ByVal yPos As Double, ByVal zPos As Double, ByVal wPos As Double, ByVal xcPos As Double, ByVal ycPos As Double, ByVal Speed As Double, ByVal LineNO As Long) As Integer
```

Delphi:
```
function LM_ArcCW_6050( Board_NO : smallInt; xPos : double; yPos : double; zPos : double; wPos : double; xcPos : double; ycPos : double; Speed : double;  LineNO : LongInt) : SmallInt; stdcall;
```

解释：

该函数将圆心在 xy **平面的顺时针圆弧/螺旋线插补命令写入插补缓冲区。同理，还有另外两个类似的顺时针插补命令：**

```
short LM_ArcCW_YZ_6050 (short Board_NO,
                        double xPos,double yPos,double zPos,double wPos,
                        double ycPos,double zcPos,
                        double Speed,long  LineNO);

short LM_ArcCW_ZX_6050 (short Board_NO,
                        double xPos,double yPos,double zPos,double wPos,
                        double zcPos,double xcPos,
                        double Speed,long  LineNO);
```

以及三个逆时针插补命令：

```
short LM_ArcCCW_6050(short Board_NO,
                     double xPos,double  yPos,double zPos,double wPos,
                     double xcPos,double ycPos,
                     double Speed,long  LineNO);
```

short LM_ArcCCW_YZ_6050 (short Board_NO,
double xPos,double yPos,double zPos,double wPos,
double ycPos,double zcPos,
double Speed,long LineNO);

short LM_ArcCCW_ZX_6050 (short Board_NO,
double xPos,double yPos,double zPos,double wPos,
double zcPos,double xcPos,
double Speed,long LineNO);

在插补引擎正在运行当中是否可调用：是

参数：

Board_NO： 0－3, **板号**

XY 平面：

xPos、yPos、zPos、wPos： XYZW **轴的终点位置；绝对坐标，单位为用户单位。**

xcPos : X **轴的圆心位置(绝对值),单位为用户单位。**

ycPos : Y **轴的圆心位置(绝对值),单位为用户单位。**

YZ 平面：

xPos、yPos、zPos、wPos： XYZW **轴的终点位置；绝对坐标，单位为用户单位。**

ycPos : Y **轴的圆心位置(绝对值),单位为用户单位。**

zcPos : Z **轴的圆心位置(绝对值),单位为用户单位。**

ZX 平面：

xPos、yPos、zPos、wPos： XYZW **轴的终点位置；绝对坐标，单位为用户单位。**

zcPos : Z **轴的圆心位置(绝对值),单位为用户单位。**

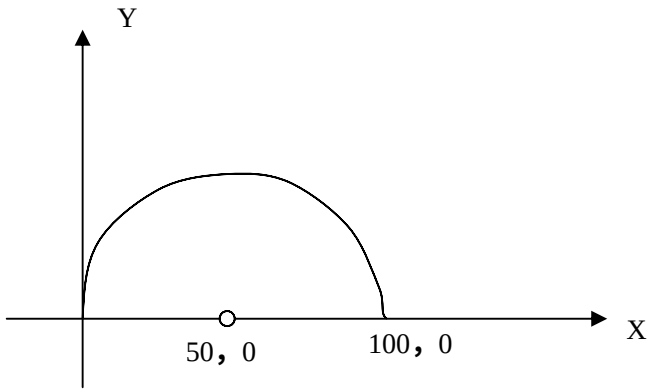
xcPos : X **轴的圆心位置(绝对值),单位为用户单位。**

Speed： **插补速度，单位：用户单位/分钟。**

LineNO： **用户自定义行号。**

例 1（C **语言**）：

设 XYZ **轴的当前位置为** 0、0、0、0**；要运行一个在** XY **平面的顺圆如下图：**

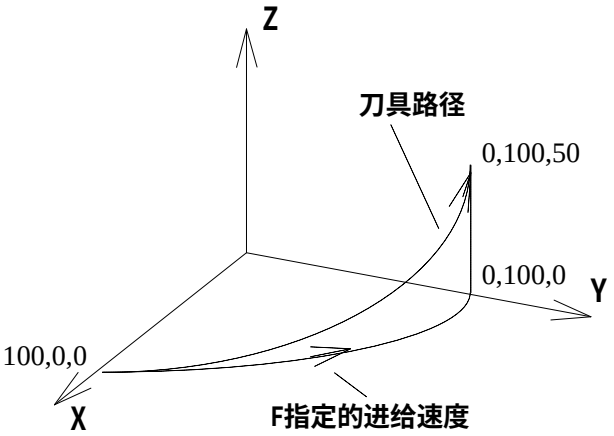


插补速度是：500，**行号为** 100；

则： LM_ArcCW_6050 (0，100，0，0，0，50，0，500，100)；

例 2 **螺旋线插补（**C **语言）：**

设 XYZW **轴从当前位置（**0，0，0，0）**直线移动到（**100，0，0，0）**位置，再以螺旋线轨迹移动到终点位置（**0，100，50）**；如下图：**



插补速度 F=200，**行号从** 100 **开始；**

则：

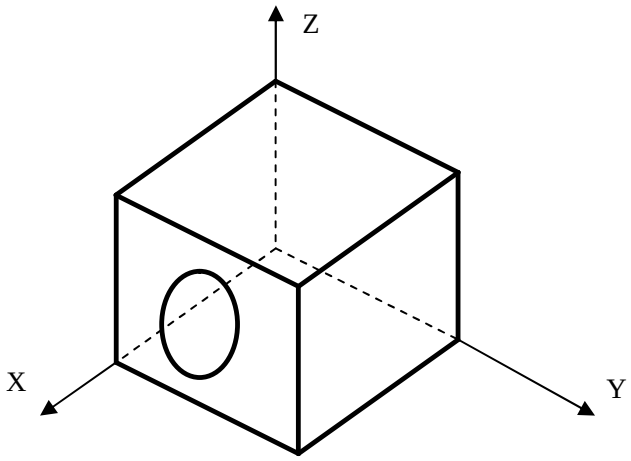
```
double F=200;
long   LineNO=100;
LM_Line_6050 (0，100，0，0，0，F，LineNO)；
LM_ArcCW_6050 (0，0，100，50，0，0，0，F，LineNO+1)；
LM_End_6050(0, LineNO+2);
    LM_Start_6050(0,0);
```

由F**值指定的插补速度是指沿着圆弧的进给速度，直线轴的进给速度为：**

F(**直线**)=F***直线轴的长度/圆弧的弧长。**

决定插补速度F值时要十分注意直线轴的进给率不要超过各种限制。

在上例中，圆弧插补均在 XY 平面完成，实际应用中有可能在 YZ 平面进行插补运动，比如在数控加工中心的机床上加工箱体零件，在箱体侧面铣一个圆孔，这时就会用到在不同平面的圆弧插补：



需要注意的是，XY 平面、YZ 平面、ZX 平面的圆弧插补函数，轴终点位置参数、圆心坐标参数，其代入次序各不相同，见参数说明。

返回码：

大于 0　**成功**

小于 0　**非法参数**

参考函数：

LM_Line_6050

LM_End_6050

LM_Start_6050

LM_Wait_6050

插补等待

函数编号:60

函数原型：

C： short LM_Wait_6050(unsigned short Board_NO,unsigned long Millisecond,long LineNO);

VB： Declare Function LM_Wait_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByVal Millisecond As Long, ByVal LineNO As Long) As Integer

Delphi: function LM_Wait_6050(Board_NO : Word; Millisecond: LongInt, LineNO : LongInt) : SmallInt; stdcall;

解释：

该命令将把插补数据送入 6050 卡的插补缓存器中,6050 卡的插补缓存器总共有 16 行.当插补引擎运行到该行时将暂停运行,在等待用户所设时间后,再重新开始.

在插补引擎正在运行当中是否可调用：是

参数：

Board_NO : 0－3, **板号**

Millisecond : **等待时间,毫秒**

LineNO : **插补行号,用户自由设定**

返回码：

(0) 成功

(-1) 不成功

参考函数：

LM_IOOut_6050

插补模式的 IO 点输出

函数编号:62

函数原型：

C： short LM_IOOut_6050(unsigned short Board_NO,short IO_Index,short IO_Value,
long LineNO);

VB： Declare Function LM_IOOut_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByVal IO_Index As Integer, ByVal IO_Value As Integer, ByVal LineNO As Long) As Integer

Delphi: function LM_IOOut_6050(Board_NO : Word; IO_Index: smallInt;short IO_Value: smallInt; LineNO : LongInt) : SmallInt; stdcall;

解释：

该命令将把插补数据送入 6050 卡的插补缓存器中,6050 卡的插补缓存器总共有 16 行． 当插补引擎运行到该行时将根据参数输出 IO 点．

在插补引擎正在运行当中是否可调用：是

参数：

Board_NO : 0－3, **板号**

IO_Index : 1-8; **通用 IO 点输出点序号**,

IO_Value : 0 - 1; **输出点值**

LineNO : **插补行号,用户自由设定**

返回码：

(0) 成功

(-1) 不成功

参考函数：

LM_Wait_I_6050

插补等待通用 I0 点输入

函数编号:63

函数原型：

C： short LM_Wait_I_6050 (unsigned short Board_NO,short IO_Index,long LineNO);

VB： Declare Function LM_Wait_I_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByVal IO_Index As Integer, ByVal LineNO As Long) As Integer

Delphi: function LM_Wait_I_6050(Board_NO : Word; IO_Index: smallInt; LineNO : LongInt) : SmallInt; stdcall;

解释：

该命令将把插补数据送入 6050 卡的插补缓存器中,6050 卡的插补缓存器总共有 16 行． 当插补引擎运行到该行时将等待，直到输入点为 1 时再执行下一行插补命令．

在插补引擎正在运行当中是否可调用：是

参数：

Board_NO : 0－3, **板号**

IO_Index : 1-8; **通用** IO **点输入点序号**,

LineNO : **插补行号,用户自由设定**

返回码：

(0) 成功

(-1) 不成功

参考函数：

LM_End_6050

写入插补结束命令

函数编号:45

函数原型：

C： short LM_End_6050(short Board_NO,long LineNO);

VB： Declare Function LM_End_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByVal LineNO As Long) As Integer

Delphi: function LM_End_6050(Board_NO : smallInt; LineNO : LongInt) : SmallInt; stdcall;

解释：

该命令将把插补数据送入 6050 **卡的插补缓存器中**,6050 **卡的插补缓存器总共有** 16 **行。当插补引擎运行到该行时将停止运行**,**如要重新开始**,**则再送入插补数据命令**(LM_Line_6050,LM_ArcCW_6050,LM_ArcCCW_6050),**并发出** LM_Start_6050 **命令**

在插补引擎正在运行当中是否可调用：是

参数：

Board_NO : 0－3, **板号**

LineNO : **用户自定义行号。当插补引擎运行到该行时，用** LM_GetLineNO_6050(short Board_NO);**函数可检测该行号值。**

返回码：

(0) 成功

(-1) 不成功

参考函数：

LM_Line_6050

LM_ArcCW_6050

LM_Start_6050

LM_GetXStartPos_6050

获得插补轴 X 当前位置

函数编号:54

函数原型：

C： short LM_GetXStartPos_6050(unsigned short Board_NO,double *Pos);

VB： Declare Function LM_GetXStartPos_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByRef Pos As Double) As Integer

Delphi: function LM_GetXStartPos_6050(Board_NO : smallInt; Pos : double) : SmallInt; stdcall;

解释：

指示 6050 **卡，轴的现在位置是轴插补的开始位置，并可获得轴当前位置（用户单位），在发送插补命令**(LM_Line_6050, LM_ArcCCW_6050)**开始前调用，类似的函数还有：**

short LM_GetYStartPos_6050(unsigned short Board_NO,double *Pos);

short LM_GetZStartPos_6050(unsigned short Board_NO,double *Pos);

short LM_GetWStartPos_6050(unsigned short Board_NO,double *Pos);

注意: 在该命令之前，先写入 LM_CleanBuff_6050 **（清空插补命令缓冲区）命令**

在插补引擎正在运行当中是否可调用：否

参数：

Board_NO : 0－3, **板号**

Pos : **获得 轴插补的开始位置,单位为用户单位。指针类参数,可以为** NULL(**空指针**) **，**

VB **用户可以直接设为** 0**，或用同等函数** LM_GetAxisStartPos_6050**。**

返回码：

(0) 成功

(-1) 不成功

参考函数：

LM_GetAxisStartPos_6050

LM_GetAxisStartPos_6050

获得插补轴当前位置

函数编号:54

函数原型：

C： double LM_GetAxisStartPos_6050(unsigned short Board_NO,short AxisFlag);

VB： Declare Function LM_GetAxisStartPos_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByRef AxisFlag As Integer) As Double

Delphi: function LM_GetAxisStartPos_6050(Board_NO : Word; AxisFlag : smallInt) : double; stdcall;

解释：

指示 6050 **卡，轴的现在位置是轴插补的开始位置，并获得轴当前位置（用户单位），在发送插补命令**(LM_Line_6050, LM_ArcCCW_6050)**开始前调用。**

注意: 如果在用户程序中多次用到上述命令，在第二次用该命令之前，先写入 LM_CleanBuff_6050 **（清空插补命令缓冲区）命令**

在插补引擎正在运行当中是否可调用：否

参数：

Board_NO : 0－3, **板号**

AxisFlag : 1- 4, **指示是那个轴；**1：X **轴，** 2：Y **轴，** 3：Z **轴，** 4：W **轴。**

返回码：

轴的当前位置。用户单位

当返回轴位置=-2147483648 **时,表示不成功,有错误产生。**

参考函数：

LM_GetXStartPos_6050

LM_Start_6050

启动插补引擎，插补开始

函数编号:41

函数原型：

C： short LM_Start_6050(short Board_NO,short ObligeFlag);

VB： Declare Function LM_Start_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer，ByVal **ObligeFlag** As Integer) As Integer

Delphi: function LM_Start_6050(Board_NO : smallInt; ObligeFlag : shortInt) : SmallInt; stdcall;

解释：

在用户写入一定数量的插补命令后，可调用该函数让插补引擎开始工作。

在插补引擎正在运行当中是否可调用：是

参数：

Board_NO ： **卡号，可能值** 0，1，2，3

ObligeFlag： **强制运行插补命令标志。**0=**不强制；**1=**强制。根据第** ***§2.4.4 插补疑问*** **节所介绍内容，当用户写入** 1 **或** 2 **行短线段插补命令时，可在随后的** LM_Start_6050 **命令中将该参数置为** 1**，强制运行插补命令。**

返回码：

(0) 成功

(-1) 不成功

参考函数：

LM_Line_6050

LM_ArcCW_6050

LM_End_6050

LM_CleanBuff_6050

清空插补命令缓冲区

函数编号:46

函数原型：

C： LM_CleanBuff_6050(short Board_NO);

VB： Declare Function LM_CleanBuff_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer) As Integer

Delphi: function LM_CleanBuff_6050(Board_NO : smallInt);SmallInt; stdcall;

解释：

这个函数将清空插补命令缓冲区。

在插补引擎正在运行当中是否可调用：否

参数：

Board_NO : 0－3, **板号**

返回码：

(0) 成功

(-1) 不成功

参考函数：

LM_GetBuffLen_6050

获得插补命令缓冲区空闲长度

函数编号:55

函数原型：

C： short LM_GetBuffLen_6050(short Board_NO);

VB： Declare Function LM_GetBuffLen_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer) As Integer

Delphi: function LM_GetBuffLen_6050(Board_NO : smallInt) : SmallInt; stdcall;

解释：

这个函数将返回插补缓冲区的空闲长度。控制卡有 16 **行命令缓冲区。**

在插补引擎正在运行当中是否可调用：是

参数：

返回码：

0-16 **成功，插补缓冲区的空闲长度**

(-1) 不成功

参考函数：

LM_GetState_6050

获得当前插补状态

函数编号:56

函数原型：

C： short LM_GetState_6050(short Board_NO);

VB： Declare Function LM_GetState_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer) As Integer

Delphi: function LM_GetState_6050(Board_NO : smallInt) : SmallInt; stdcall;

解释：

这个函数将获得当前插补引擎的状态。

在插补引擎正在运行当中是否可调用：是

参数：

Board_NO : 0－3, **板号**

返回码：

0: **停止状态**
1: **被轴停止命令结束** (**建议用** Abort_6050 **命令**)
2: **被行结束停止命令停止**
3: **被** LM_End_6050 **结束**
4: **插补缓冲区空 或 等待后续命令**
5: **被进给倍率为** 0 **暂停**
6: **被进给暂停挂起**
7: **插补正在进行**
255: **命令不成功**

注意：当返回码为：4 **或** 5、6、7 **时就是 插补引擎正在运行 状态。**

参考函数：

LM_GetLineNO_6050

获得当前插补行号

函数编号:49

函数原型：

C： long LM_GetLineNO_6050(short Board_NO);

VB： Declare Function LM_GetLineNO_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer) As Long

Delphi: function LM_GetLineNO_6050(Board_NO : smallInt) : LongInt; stdcall;

解释：

这个函数将获得插补引擎当前运行的插补行号，该行号是用户在写入插补命令时自定义的。

在插补引擎正在运行当中是否可调用：是

参数：

Board_NO : 0－3, **板号**

例（C 语言）：

```
long LineNO;
LineNO = LM_GetLineNO_6050 (0);
```

返回码：

当前行号; **范围**: **长整型数**; **用户设定值。当返回值**=-2147483648 **时,表示不成功,有错误产生。**

参考函数：

LM_Pause_6050

暂停插补运算

函数编号:50

函数原型：

C： short LM_Pause_6050(short Board_NO);

VB： Declare Function LM_Pause_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer) As Integer

Delphi: function LM_Pause_6050(Board_NO : smallInt) : SmallInt; stdcall;

解释：

这个函数将暂停插补运算，插补各轴在 10 毫秒之内减速停止。

可用 LM_Resume_6050(short Board_NO)**函数恢复。**

在插补引擎正在运行当中是否可调用：是

参数：

Board_NO : 0－3, **板号**

返回码：

(0) 成功

(-1) 不成功

参考函数：

LM_Resume_6050

LM_Resume_6050

恢复插补运算

函数编号:51

函数原型：

C： short LM_Resume_6050(short Board_NO);

VB： Declare Function LM_Resume_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer) As Integer

Delphi: function LM_Resume_6050(Board_NO : smallInt) : SmallInt; stdcall;

解释：

这个函数将恢复被函数（LM_Pause_6050**）暂停的插补运算。**

在插补引擎正在运行当中是否可调用：是

参数：

Board_NO : 0－3, **板号**

返回值:

(0) 成功

(-1) 不成功

参考函数：

LM_Pause_6050

LM_SetSpeedRate_6050

改变插补矢量速度

函数编号:52

函数原型：

C： short LM_SetSpeedRate_6050(short Board_NO,short Rate);

VB： Declare Function LM_SetSpeedRate_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByVal Rate As Integer) As Integer

Delphi: function LM_SetSpeedRate_6050(Board_NO : smallInt; Rate : smallInt) : SmallInt; stdcall;

解释：

这个函数将实时改变插补矢量速度。设：用户设定速度=Vu**；则：插补矢量速度**

$= \frac{V_u \times Rate}{100.0}$

在插补引擎正在运行当中是否可调用：是

参数：

Board_NO : 0－3, **板号**

Rate : 1-160, 100=100%,**即**:**按原设定速度执行**;

10=10%,**即**:**按原设定速度的百分之十执行**,

返回码：

(0) 成功

(-1) 不成功

参考函数：

GetErrorNo_6050

获取系统错误信息

函数编号:100

函数原型：

C： short GetErrorNo_6050(short Board_NO,short CleanFlag,
unsigned int *ErrorNo,unsigned int *FuncNo);

VB： Declare Function GetErrorNo_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer, ByVal CleanFlag As Integer, ByRef ErrorNo As Integer, ByRef FuncNo As Integer) As Integer

Delphi: function GetErrorNo_6050(Board_NO : smallInt; CleanFlag : smallInt; ErrorNo: Cardinal:FuncNo: Cardinal) : SmallInt; stdcall;

解释：

获得错误号 当调用函数的返回值 为"不成功"时,或电机卡不正常时,可调用该函数获得错误信息

在插补引擎正在运行当中是否可调用：是

参数：

Board_NO : 0－3, **板号**

CleanFlag : 0 **或** 1; 0=**不清除错误**; 1=**清除错误**;

*ErrorNo : **返回错误编号**; **指针类参数**,**可以为** NULL(**空指针**)，VB **用户可以直接设为** 0。

*FuncNo : **返回产生错误的函数编号**;**指针类参数**,**可以为** NULL(**空指针**)，VB **用户可以直接设为** 0。

返回码：

(0) 无错误

(大于 0) 有错误，错误编号。

* **错误编号表**:

1 **轴位置寄存器超限**

2 **轴在插补运动中**

3 **插补误差超限**

4 **轴在运动中**

5 **逻辑错误**

6 **插补轴压在限位上**

7 **内存不够**

8 **参数错误**

9 **插补缓存器满**

10 **与电机卡通讯失败**

11 **与电机卡通讯超时**

12 **插补数据小于系统最小速度**

13 **插补数据小于系统最小长度**

14 **插补数据大于系统圆弧半径误差**

15 **插补数据圆弧计算错误**

16 **设定值越界；**

系统值范围:

轴位置: -2147483648 **至** 2147483647

插补向量长度: <1073741823 (**直线长度和圆弧长度**)

圆弧最大半径: <1048575

插补圆心位置: -2147483648 **至** 2147483647

17 6050 **动态链接库版本与** 6050 **卡的固件版本不一致**

18 **有插补测量代码,后续不能再加入插补命令**

19 **轴在跟随运动**

20 **轴在停止过程**

参考函数：

LM_LineEnd_6050

插补运行完当前行停止

函数编号:53

函数原型：

C： short LM_LineEnd_6050(short Board_NO);

VB： Declare Function LM_LineEnd_6050 Lib "dfjzh6050dll.dll" (ByVal Board_NO As Integer) As Integer

Delphi: function LM_LineEnd_6050(Board_NO : smallInt;) : SmallInt; stdcall;

解释：

插补运行完当前行停止。该命令不进入插补缓存区,可在插补运行时,随时执行 当插补引擎运行完当前行停止时,如要重新开始,则再执行 LM_Start_6050 命令

在插补引擎正在运行当中是否可调用：是

参数：

Board_NO : 0－3, **板号**

返回码：

(0) 成功

(-1)不成功

参考函数：