深圳市雷赛控制技术有限公司
SHENZHEN LEADSHINE CONTROL TECHNOLOGY CO.,LTD

# 雷赛运动控制器
# API 函数编程手册

Version 1.5

2017.5.9

# 版 权 说 明



本手册中的信息资料仅供参考。由于改进设计和功能等原因，雷赛公司保留对本资料的最终解释权，内容如有更改，恕不另行通知。

调试机器要注意安全！用户必须在机器中设计有效的安全保护装置，在软件中加入出错处理程序。否则所造成的损失，雷赛公司没有义务或责任负责。

# 目录

# 文档版本

| 版本号 | 修订日期 | 备注 |
| --- | --- | --- |
| V 1.5 | 2017-5-9 | |
| | | |
| | | |

# 第一章 API 函数使用介绍

## 1.1 API 编程系统软件架构

雷赛控制器提供了动态链接库，用户可选择调用动态链接库提供的API函数来完成所需的各种功能。所谓API编程，即是指在PC机中编写应用程序代码，应用程序通过调用动态链接库提供的API函数来执行相关功能。如SMC606提供的动态链接库包含下面三个文件LTSMC.dll，LTSMC.h， LTSMC.lib，目前提供支持多种编程语言版本的动态库，包括MICROSOFT WINDOWS系统下C#、VB、VC、VB.NET、VC.NET、LABVIEW、DELPHI等，MAC系统下Xcode环境等。

SMC600系列控制器为不同的应用环境分别提供了动态链接库：

文件夹“WINCE_DLL”中的动态链接库适用于在WINCE环境下编程使用；

文件夹“WINDOWS_PC_DLL_32”中的动态链接库适用于在32位环境下编程使用；

文件夹“WINDOWS_PC_DLL_64”中的动态链接库适用于在64位环境下编程使用。

在Windows系统下，用户可以使用任何能够支持动态链接库的开发工具来开发应用程序。下面分别以Visual C＋＋、Visual Basic和C#为例讲解如何在这些开发工具中使用运动控制器的动态链接库。

在MAC系统下，用户可以在Xcode环境中编写应用程序。

## 1.2 基于 VC 6.0 软件开发的简介

下面以Visual C++ 6.0 环境下编写一个点位运动的应用软件为例，讲解用VC 开发应用软件的一般方法。

1）打开Visual C++ 6.0。

2）新建一个工程。

3）选择MFC APPWizard(exe)。

4）选择工程保存路径，如：E:\。

5）输入工程名,如：SMC_EXAMPLE。如图 1.1。

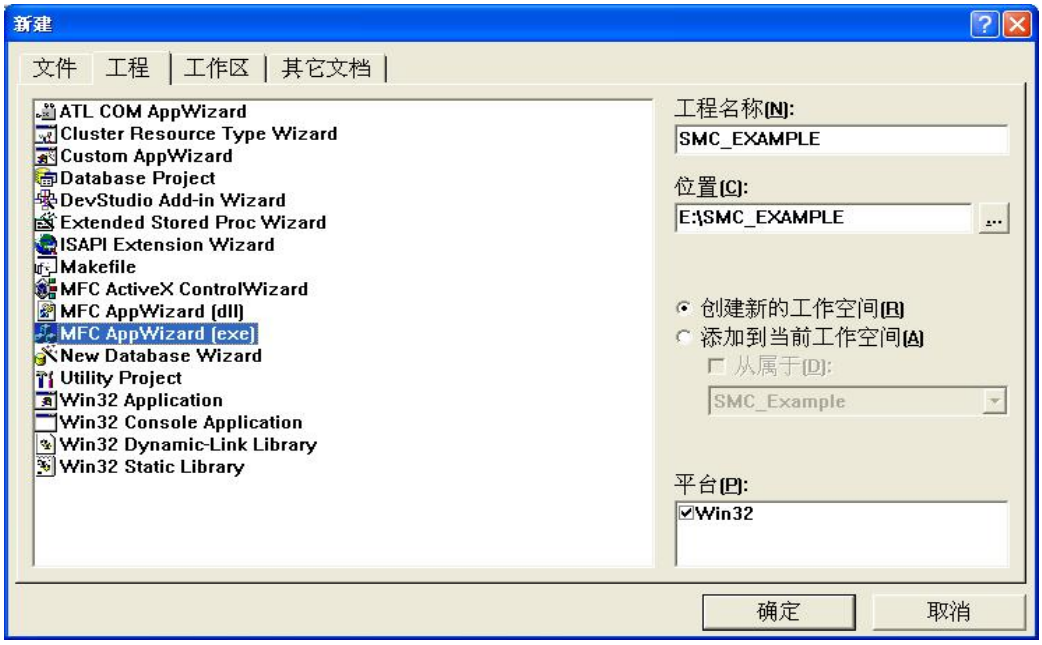



图 1.1 新建工程

6） 在应用程序类型中选择“基本对话框”，按“完成”键，建立工程。

7）从资料光盘动态库中的头文件目录下找到LTSMC.h、LTSMC.lib 和LTSMC.dll文件，拷贝到E:\SMC_EXAMPLE 目录。

8）在菜单中选择“工程”->“添加工程”->“文件”，选中LTSMC.lib文件和LTSMC.h文件加入到工程中。

9)打开SMC_EXAMPLE.cpp 文件、SMC_EXAMPLEDlg.cpp,在程序开始部分添加相应语句:#include “LTSMC.h”，如图1.2所示。

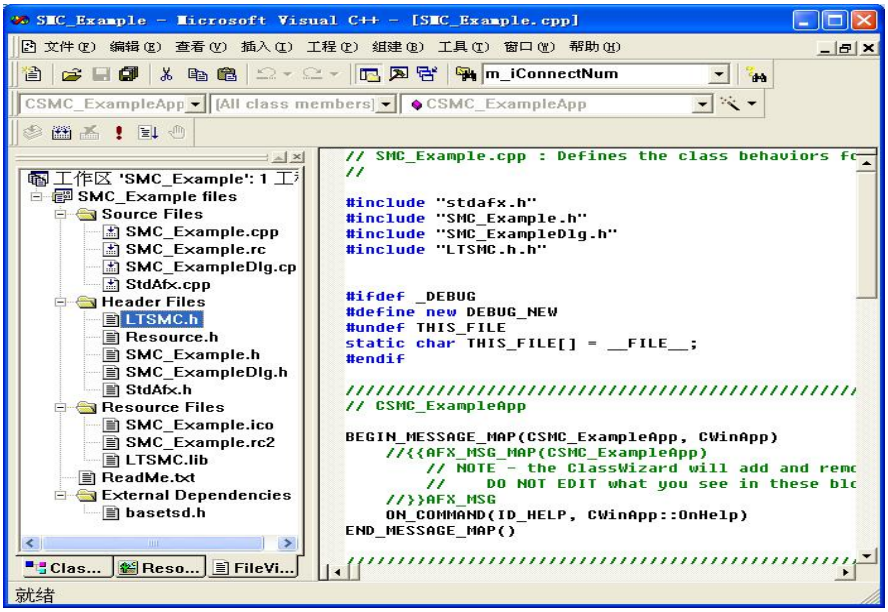



图1.2程序增加头文件

10）添加按钮“启动”和“停止”；并分别命名为“IDC_BUTTON_Start”和“IDC_BUTTON_Stop”，如图1.3所示。


图1.3 添加对话框

11）双击窗口界面，在CSMC_EXAMPLEDlg::CSMC_EXAMPLEDlg(CWnd* pParent /*=NULL*/):CDialog(CSMC_EXAMPLEDlg::IDD, pParent)函数中添加代码，用串口连接：

smc_board_init(0, 1,“com1”, 115200)

12）双击“启动”按键，在CSMC_EXAMPLEDlg::OnBUTTONStart()函数中添加代码：

```
smc_set_profile_unit(0,0,500,5000, 0.01,0.01,500);
smc_pmove_unit(0,0,200000,0);
```

13）双击“停止”按键在CSMC_EXAMPLEDlg::OnBUTTONStop()函数事件中输入代码：smc_stop(0,0,0);。如图1.4所示：

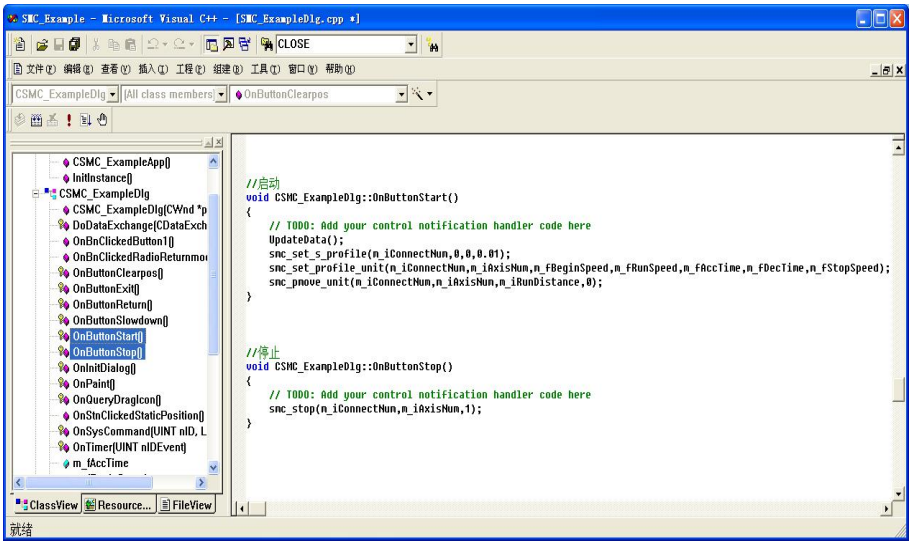



图1.4程序中调用运动控制器库函数

15）在CSMC_EXAMPLE Dlg 中添加一个成员函数OnCancel,在OnCancel 函数中添加代码如下：

```
CDialog::OnCancel();
```

```
smc_board_close(0);
```

16）编译程序后，运行程序，显示图1.5所示的界面。按下“启动”按钮，第0 轴就会输出长度为200000 的脉冲；运动中可以按下“停止”按钮便会减速停止脉冲输出。


图 1.5 程序运行界面

## 1.3 基于 VB 6.0 软件开发的简介

下面以Visual Basic6.0 环境下编写一个点位运动的应用软件为例，讲解用VB 开发应用软件的一般方法。

1）在磁盘上新建一个目录，如E:\test1

2）打开Visual Basic 6.0，新建一个“标注EXE”工程，在对话框上添加按钮“启动”和“停止”，并将其名称分别修改为“CB_Start”和“CB_Stop”，如图1.6所示。

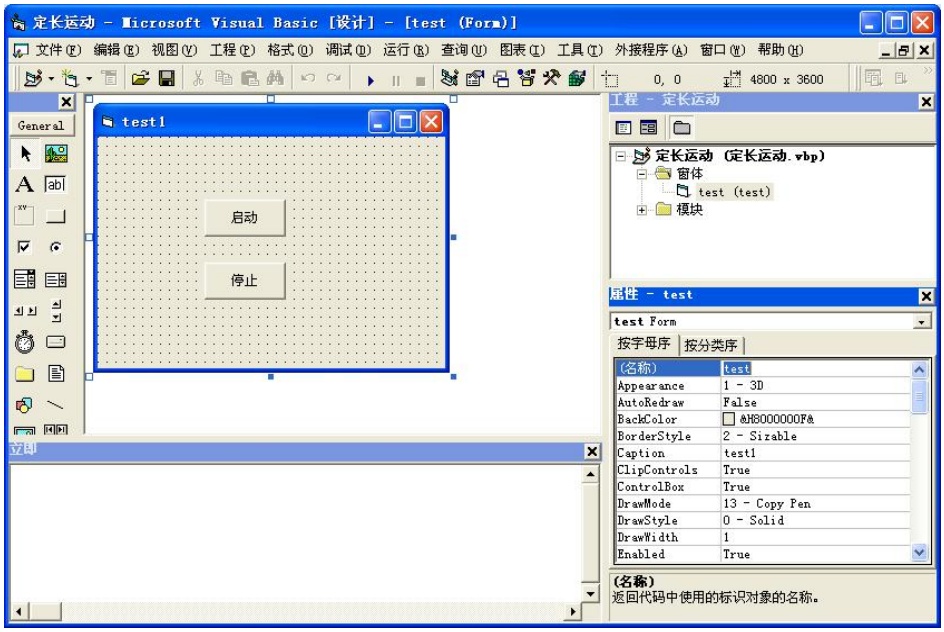



图1.6修改对话框

3）工程保存在E:\test1 目录下。

4）从资料光盘动态库中的头文件目录下找到LTSMC.bas、LTSMC.dll文件，拷贝到test1目录下。

5）菜单中选择“工程”->“添加模块”->“现存”，找到test1 目录下的LTSMC.bas 文件，添加到工程中，如图1.7所示。

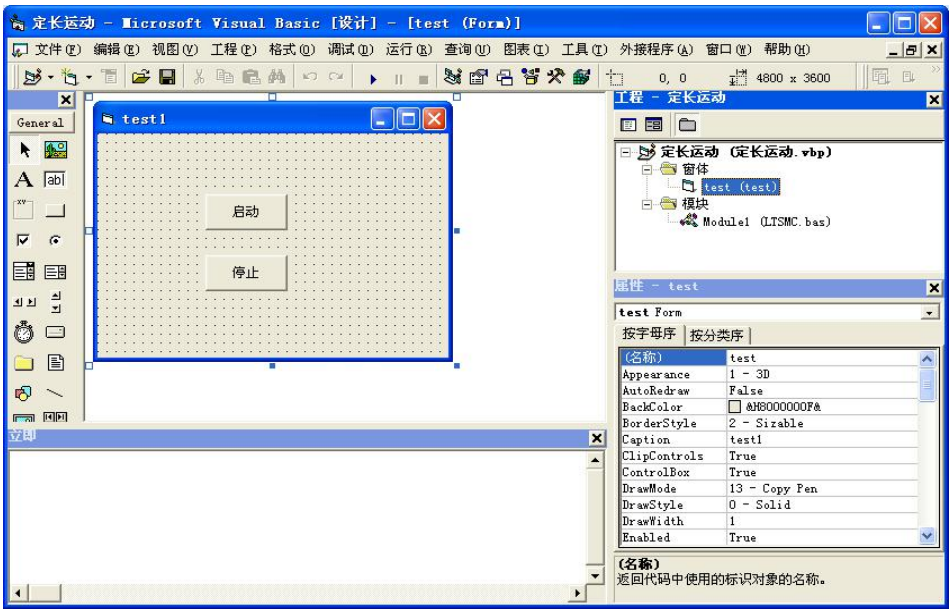



图1.7 添加头文件

6）用网口连接控制器。可参见图1.8中具体函数指令。

（1）双击窗口控件，在Form_Load 事件中添加代码

```
smc_board_init 0, 2, "192.168.5.11", 0
```

（2）双击“启动”按钮，在CB_Start_Click 事件中添加代码如下：

```
smc_set_profile_unit 0, 0, 500, 5000, 0.01, 0.01, 500
smc_pmove_unit 0, 0, 200000, 0
```

（3）双击“停止”按钮，在CB_Stop_Click()事件中添加代码如下：

```
smc_stop 0,0,0
```

（4）在Form_Unload 事件中添加代码

```
smc_board_close（0）
```

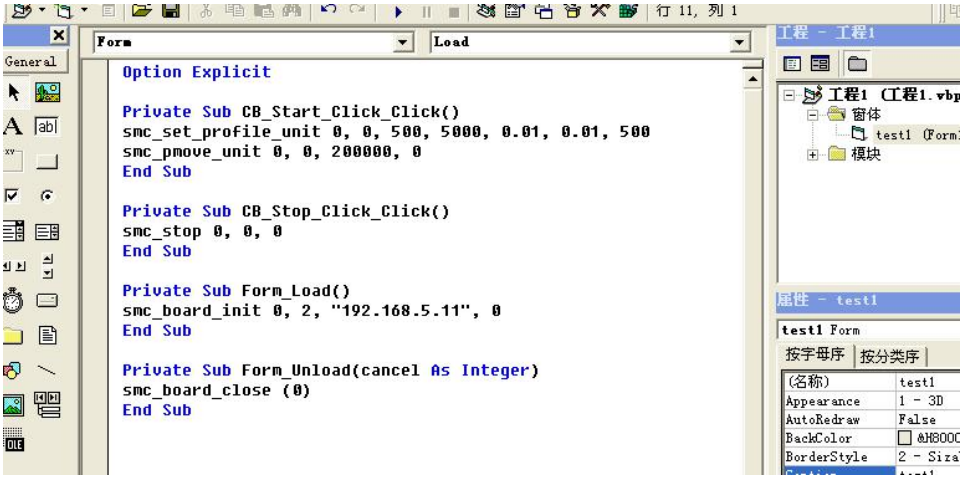



图1.8 程序中调用运动控制器库函数

7）程序编写完成。运行程序，显示界面如图1.9所示。按下“启动”按钮，第0 轴就会输出长度为200000 的脉冲；运动中可以按下“停止”按钮，便会减速停止脉冲输出。


图 1.9 程序运行界面（VB）

## 1.4 基于 C #软件开发的简介

下面以C#环境下编写一个点位运动的应用软件为例，讲解用C#开发应用软件的一般方法。

1）在磁盘上新建一个目录，如E:\ C_Sharp

2）打开C#2010，新建一个“windows窗体应用程序”工程，在对话框上添加按钮“启动”和“停止”，并将其名称分别修改为“CB_Start”和“CB_Stop”,如图1.10所示。

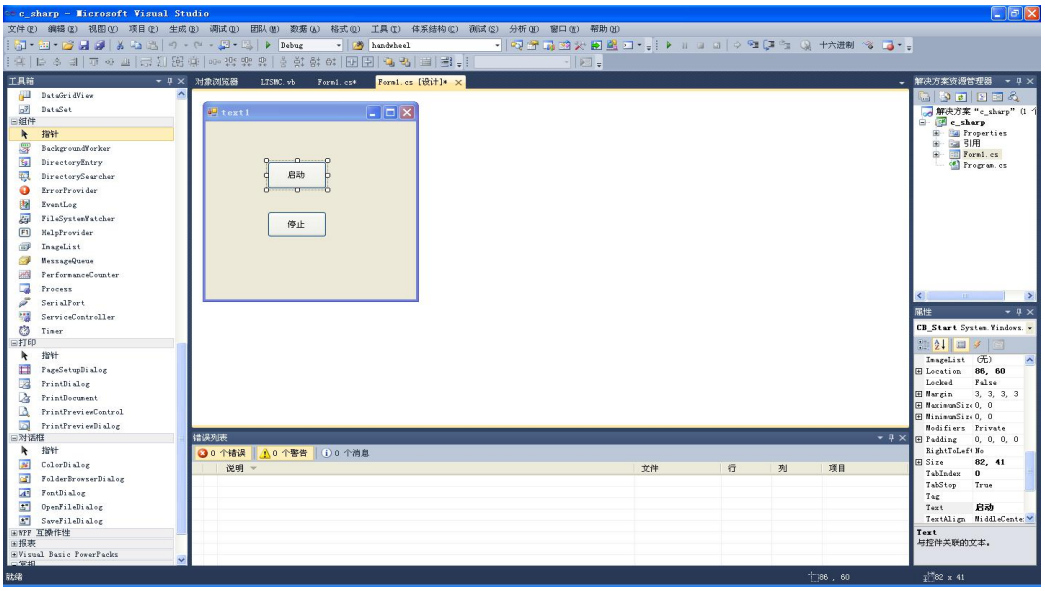


图 1.10 程序编辑界面

3）工程保存在E:\C_Sharp目录下。

4）在资料光盘动态库中的头文件目录下找到LTSMC.CS、LTSMC.dll 文件，拷贝到C_Sharp目录的..BIN/DEBUG下。

5）右击菜单中的工程名“C_Sharp” ->“添加”->“现有项”，找到test1 目录下的LTSMC.cs文件，添加到工程中。

6）添加头文件和事件；添加头文件 using Leadshine;（Leadshine为LTSMC.CS中的命名空间名），用网口连接控制器。

（1）双击FORM中的“启动”按键，在软件编辑器中写入代码

```
private void CB_Start_Click(object sender, EventArgs e)
{
    ushort CardNo = 0;                  //卡号
    ushort axis = 0;                    //运动轴号
    double start_speed = 0;             //启动速度
    double speed = 1000;                //最大运行速度
    double stop_speed = 0;              //停止速度
    double tacc = 0.1;                  //加速时间
    double tdec = 0.1;                  //减速时间
    double s_pare = 0.05;               //s形平滑系数
    double dist = 10000;                //运动距离
    LTSMC.smc_set_profile_unit(CardNo, axis, start_speed, speed, tacc, tdec, stop_speed);//设置速度参数
    LTSMC.smc_set_s_profile(CardNo, axis, 0, s_pare);   //设置S平滑系数
    LTSMC.smc_pmove_unit(CardNo, axis, dist, 0);        //启动定长运动
}
```

（2）双击FORM中的“停止”按键，在软件编辑器中写入代码

```
private void CB_Stop_Click(object sender, EventArgs e)
{
     ushort CardNo = 0; //卡号
    ushort axis = 0;    //运动轴
    ushort mode = 0;    //停止模式，0：减速停止，1：紧急停止
    LTSMC.smc_stop(CardNo, axis, mode);    //停止运动
}
```

（3）双击窗体form边框，在软件编辑器中写入代码，用网口连接控制器。

```
private void Form1_Load(object sender, EventArgs e)
{
    ushort CardNo = 0;
    //short res = LTSMC.smc_board_init(CardNo, 1, "COM1", 115200);//串口连接类型为1
    short res = LTSMC.smc_board_init(CardNo, 2, "192.168.5.11", 0);    //网口连接类型为2
    if (res != 0)
    {
        MessageBox.Show(string.Format("连接控制器失败,错误码:{0}", res), "错误");
    }
}
```

7）点击程序运行，或按F5键，程序自动运动，界面如下：点击启动控制器轴运行，按停止键运动停止，如图1.11。


图 1.11 程序运行界面（C#）

# 第二章 功能实现

## 2.1 参数设置

### 2.1.1 控制器初始化

在操作运动控制器之前，必须调用函数 smc_board_init 为运动控制器分配资源。同样，当程序结束对运动控制器的操作时，必须调用函数 smc_board_close 释放运动控制器所占用的 PC 系统资源，使得所占资源可被其它设备使用。

相关函数：

| 名称 | 功能 | 参考 |
|---|---|---|
| smc_board_init | 控制器初始化函数 | 3.1 节 |
| smc_board_close | 控制器关闭函数 | |

⚠注意：连接控制器之前都需要对控制器的初始化，连接方式有网口、串口等。

例程 1：假设电脑串口号为“COM1”，串口波特率为 115200、停止位 2、无校验 0

```
short iret =smc_board_init(0,1,"COM1",115200);      //串口默认连接方式 short iret =smc_board_init_ex (0,1,"COM1",115200,8,0,2);//串口高级连接
```

例程 2：假设控制器 IP 为“192.168.5.11”使用网口连接。

```
short iret =smc_board_init(0,2,"192.168.5.11", 0); //网口连接方式
```

### 2.1.2 脉冲输出模式设置

雷赛控制器采用指令脉冲控制步进/伺服电机。由于市面上的众多电机驱动器厂家信号接口要求各有不同（常用的有六种类型），所以在使用运动控制器控制具体的电机驱动器时，必须根据电机驱动器的接收信号类型，使用函数 smc_set_pulse_outmode 对运动控制器的脉冲输出模式进行正确的设定，电机才能正常工作。

相关函数：

| 名称 | 功能 | 参考 |
|---|---|---|
| smc_set_pulse_outmode | 设置指定轴的脉冲输出模式 | 3.2 节 |
| smc_get_pulse_outmode | 读取指定轴的脉冲输出模式 | |

### 2.1.3 脉冲当量设置

雷赛控制器提供了脉冲当量设置功能，该功能可以自定义位置（位移）单位；并且提供了相应的各种高级运动函数。

相关函数：

| 名称 | 功能 | 参考 |
|---|---|---|
| smc_set_equiv | 设置脉冲当量值 | 3.3 节 |
| smc_get_equiv | 读取脉冲当量值 | |

### 2.1.4 反向间隙补偿

雷赛控制器提供了反向间隙补偿功能，以降低机械传动反向间隙的影响。

相关函数：

| 名称 | 功能 | 参考 |
|---|---|---|
| smc_set_backlash_unit | 设置反向间隙值 | 3.4 节 |
| smc_get_backlash_unit | 读取反向间隙值 | |

### 2.1.5 轴 IO 映射

雷赛控制器支持轴 IO 映射配置功能，支持将轴专用 IO 信号配置到任意一个硬件输入口，如：可将限位接口当原点信号。该功能可减少现场接线、换线的困难。

轴 IO 映射相关函数：

| 名称 | 功能 | 参考 |
|---|---|---|
| smc_set_axis_io_map | 设置轴 IO 映射关系 | 3.26 节 |
| smc_get_axis_io_map | 读取轴 IO 映射关系设置 | |

例 1：设置第 2 轴原点接口作为第 0 轴的正限位信号.(例程为 c++编译环境)

```
int main(int argc, char* argv[])
{
/**********************变量定义*****************************/
    WORD CardNo = 0;            //连接号
    WORD Axis = 0;               //指定轴号：第 0 轴
    WORD IoType = 0;            //指定轴的 IO 信号类型为：正限位信号
    WORD MapIoType = 2;          //轴 IO 映射类型：原点信号
```

```
    WORD MapIoIndex = 2;          // 轴 IO 映射索引号：第 2 轴
    double filter = 0;           //滤波时间
    short ret;
/**********************函数调用执行**************************/
   //第一步、设置映射参数
   ret=smc_set_axis_io_map(CardNo,Axis,IoType,MapIoType,MapIoIndex,filter);
  //第二步、回读映射参数值
  ret=smc_get_axis_io_map(CardNo,Axis,IoType,&MapIoType,&MapIoIndex,&filter);
    printf("轴 IO 映射类型= %d\n ",MapIoType);
    printf("轴 IO 映射索引号= %d\n ",MapIoIndex);
    printf("滤波时间= %f\n ",filter);
}
```

### 2.1.6 参数设置例程

例程：控制器使用之前进行初始化（通过网口连接雷赛控制器），设置脉冲模式为模式 0，设置脉冲当量为 10，启动点位运动，运动结束后关闭控制器

```
   int main(int argc, char* argv[])
   {
   /**********************变量定义*****************************/
    short ret;   //错误码返回值
    WORD ConnectNo = 0; //连接号，可选 0--7
    WORD type = 2;       //Type 链接类型：1-串口，2-网口
    char *pconnectstring = "192.168.5.11";   //控制器 IP 地址
    DWORD baud=0;
    WORD axis = 0;          //运动轴
    WORD outmode = 0;       //脉冲输出模式
    double equiv = 10;      //脉冲当量
    double backlash = 10;    //反向间隙
    WORD posi_mode = 0;      //0:相对模式，1：绝对模式
    double read_pos;        //读取指令脉冲计数器值

   /**********************函数调用执行**************************/
   //第一步、控制器连接初始化，以网口连接控制器
    ret = smc_board_init(ConnectNo, type, pconnectstring, baud);
  //第二步、设置控制器初始参数值
  ret = smc_set_pulse_outmode(ConnectNo,axis,outmode);//设脉冲输出模式
  ret = smc_set_equiv(ConnectNo,axis, equiv);     //设置脉冲当量值
```

```
  ret = smc_set_backlash_unit(ConnectNo,axis,backlash); //设反向间隙值
 //第三步、控制器关闭后，释放系统资源
 ret = smc_board_close(ConnectNo);
}
```

## 2.2 运动功能

### 2.2.1 点位运动

主要包括定长运动、恒速运动、在线变速度、在线变位置运动等。

#### 2.2.1.1 参数设置

雷赛控制器在执行点位运动控制指令时，可使电机按照 T 形速度曲线或 S 形速度曲线进行点位运动。同时可以设置起始速度、停止速度，不同的加速时间、减速时间等速度参数。

相关函数：

| 名称 | 功能 | 参考 |
|---|---|---|
| smc_set_profile_unit | 设置单轴运动速度曲线 | 3.6 节 |
| smc_get_profile_unit | 读取单轴运动速度曲线 | |
| smc_set_s_profile | 设置单轴速度曲线 S 段平滑时间参数 | |
| smc_get_s_profile | 读取单轴速度曲线 S 段平滑时间参数 | |

⚠注意：（1）由于运动控制器的最大脉冲输出频率为 2MHz，故设置的最大速度与脉冲当量设置值的乘积必须小于 2MHz

（2）单轴速度曲线 S 平滑时间，若值为 0 则为 T 形曲线，不为 0 则为 S 平滑曲线，S 平滑时间值范围为 0-1S。

对于其速度曲线，根据加速度和减速段是否相同，分为对称和非对称；根据形状可分为 T 形和 S 形，见图 2.1、图 2.2.

“起始速度”：设置单轴运动的初始速度，单位：unit/s 。

“运行速度”：设置单轴运动的最大运行速度，单位：unit/s 。

“停止速度”：设置单轴运动的停止速度，单位：unit/s 。

“加速时间”：设置单轴运动时从起始速度加速到最大运行速度需要的时间（Tacc）

“减速时间”：设置单轴运动时从最大运行速度减速到停止速度需要的时间（Tdec）

“S 段时间”：设置单轴速度曲线 S 段的时间参数，见下图 spara。

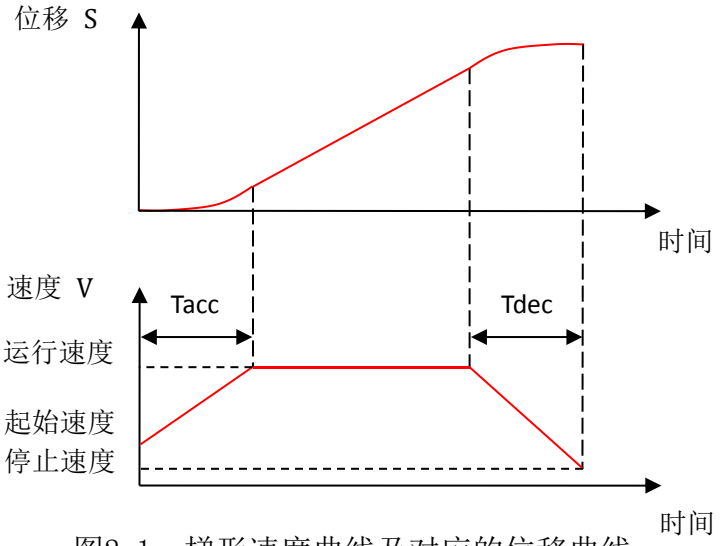



图2.1　梯形速度曲线及对应的位移曲线

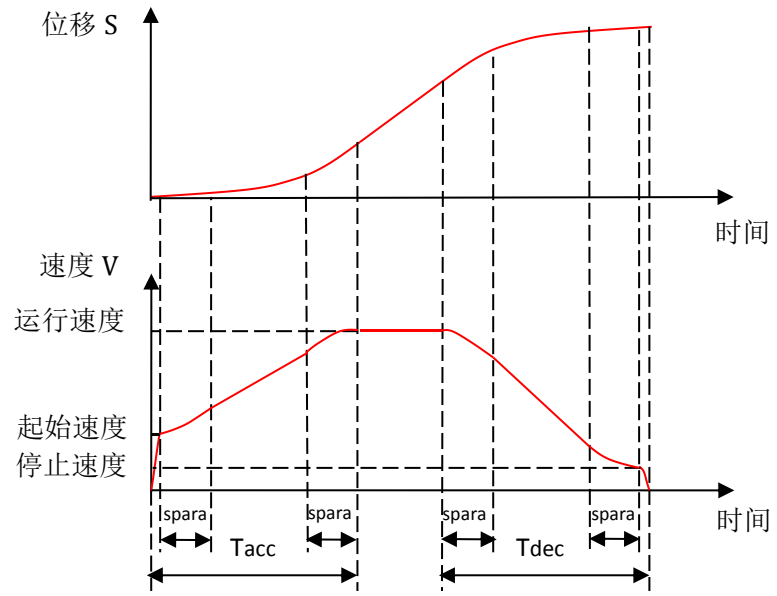



图2.2 S形速度曲线及对应的位移曲线

### 2.2.1.2 定长运动

定长运动是指：运动控制器控制运动平台从当前位置开始以设定的速度运动到指定位置后准确地停止。

在定长运动模式下，各轴可以独立设置目标位置、目标速度、加速时间、减速时间、起跳速度、停止速度、S 段参数值等运动参数，能够独立运动或停止。

相关函数：

| 名称 | 功能 | 参考 |
|---|---|---|

| | | |
|---|---|---|
| smc_pmove_unit | 点位运动 | 3.6 节 |

### 2.2.1.3 恒速运动

恒速运动模式中，控制器可以控制电机以梯形或 S 形速度曲线在指定的加速时间内从起始速度加速至最大速度，然后以该速度一直运行，直至调用停止指令或者该轴遇到限位信号才会按启动时的速度曲线减速停止。

相关函数：

| 名称 | 功能 | 参考 |
|---|---|---|
| smc_vmove | 指定轴恒速运动 | 3.6 节 |
| smc_stop | 指定轴停止运动 | |

⚠注意：恒速运动需手动停止，否则运动将一直持续。

### 2.2.1.4 在线变速变位运动

在点位运动过程中，速度和位置都是可以实时变化的。

相关函数：

| 名称 | 功能 | 参考 |
|---|---|---|
| smc_reset_target_position_unit | 在线改变指定轴的目标位置 | 3.6 节 |
| smc_update_target_position_unit | 强制改变指定轴的目标位置 | |
| smc_change_speed_unit | 在线改变指定轴的运动速度 | |

⚠注意：1）在线变位适用于点位运动；在线变速适用于点位和恒速运动。

2）在线变位后的目标位置为绝对坐标位置值，无论当前的运动模式为绝对坐标还是相对坐标模式。

3）在线变速可以设置变速时间，设置的变速时间是从当前速度变速到新速度的时间。此时控制器会重新计算起始速度加速到最高速度所需的时间以及最高速度减速到停止速度所需的时间，即加减速时间会被重新计算。变速一旦成立，该轴默认运行速度将会被改写为 New_Vel，加减速时间也会被控制器新计算的值所覆盖。

4）在线变位需运动状态处于末停止状态才会执行，而强制变位，不管当前轴是否在运动都可以改变位置。

例程 1：进行一段点位运动，运动距离是 10000unit，速度曲线为 S 形速度曲线，起始速度是 0，最大速度是 1000，停止速度是 0，加速时间 0.1S，减速时间 0.2S,运行一段时间后速度变为 2000 再运行一段时间后变位到 0。

```
int main(int argc, char* argv[])
{
/**********************变量定义*****************************/
        WORD ConnectNo=0;    //连接号，可选 0--7
        WORD ret=0 ;         //返回错误码
        WORD axis = 0;           //运动轴
        double Max_Vel = 1000;  //最大运行速度
        double Tacc = 0.1;      //加速度
        double Tdec = 0.2;      //减速度
       double Min_Vel = 0;     //起始速度
        double Stop_Vel = 0;     //停止速度
        double s_para = 0.1;     //S 形平滑系数
        double Dist = 10000;      //运动距离
        WORD posi_mode = 0;      //0:相对模式，1：绝对模式
       double New_Vel = 2000;  //在线变速后的速度值
       double Taccdec=0.1;  //在线变速后的加速时间值

/**********************函数调用执行*************************/
         //第一步、设置单轴运动速度曲线
        ret=smc_set_profile_unit(ConnectNo,axis,Min_Vel,Max_Vel,Tacc,Tdec,Stop_Vel)
        ;
        //第二步、设置单轴速度曲线平滑 S 段参数值
        ret =smc_set_s_profile(ConnectNo,axis,0,s_para);
        //第三步、启动定长运动
       ret =smc_pmove_unit(ConnectNo,axis,Dist,posi_mode);
      //第四步、启动在线变速
       Sleep(500);       //延时一段时间
      ret=smc_change_speed_unit(ConnectNo,axis,New_Vel,Taccdec);
      //第五步、启动在线变位，变目标位置到 0
      Sleep(500);       //延时一段时间
     ret =smc_reset_target_position_unit(ConnectNo, axis, 0);
     }
```

例程 2：进行一段负向恒速运动，运行一段时间后速度变为 2000，变速值为正值,运动方向将

往正向运动,再一段时间后运动停止。

```
int main(int argc, char* argv[])
{
/**********************变量定义*****************************/
        WORD ConnectNo=0;    //连接号，可选 0--7
        WORD ret=0 ;         //返回错误码
       WORD axis = 0;         //运动轴
       double Min_Vel = 0;    //起始速度
       double Max_Vel = 1000;  //最大运行速度
       double Tacc = 0.1;      //加速度
       double Tdec = 0.2;      //减速度
       double Stop_Vel = 0;     //停止速度
       double s_para = 0;      //S 形平滑系数
        double read_pos;        //读取指令脉冲计数器值
       WORD dir=0;            //负反向运动
       double New_Vel=2000   ;  //速度变化后值

/**********************函数调用执行*************************/
  //第一步、设置单轴运动速度曲线
   ret=smc_set_profile_unit(ConnectNo,axis,Min_Vel,Max_Vel,Tacc,Tdec,
    Stop_Vel);
  //第二步、设置单轴速度曲线平滑 S 段参数值
  ret =smc_set_s_profile(ConnectNo,axis,0,s_para);
  //第三步、启动恒速运动
  ret =smc_vmove(ConnectNo,axis,dir);
  //第四步、启动在线变速运动
  Sleep(500);                //延时一段时间
  ret =smc_change_speed_unit(ConnectNo,axis,New_Vel,0.1);
  //第五步、立即停止运动
  Sleep(500);       //延时一段时间
  ret =smc_emg_stop(ConnectNo);
}
```

### 2.2.2 回原点运动

在进行精确的运动控制之前，需要设定运动坐标系的原点。运动平台上都设有原点传感器（也称为原点开关），可用来原点信号的输入源。

雷赛控制器提供了 10 种回原点运动的方式：

方式 0：一次回零

该方式以设定速度回原点；适合于行程短、安全性要求高的场合。动作过程为：电机从初始位置以恒定速度向原点方向运动，当到达原点开关位置，原点信号被触发，电机立即停止（过程 0）；将停止位置设为原点位置，如图 2.9 所示。


图 2.9　一次回零方式示意图

方式 1：一次回零加回找

该方式先进行方式 1 运动，完成后再反向回找原点开关的边缘位置，当原点信号第一次无效时，电机立即停止；将停止位置设为原点位置如图 2.10 所示。


图 2.10　一次回零加回找方式示意图

方式 2：两次回零

如图 2.11 所示，该方式为方式 0 和方式 1 的组合。先进行方式 1 的回零加反找，完成后再进行方式 0 的一次回零。可参见方式 0 和方式 1 的说明。

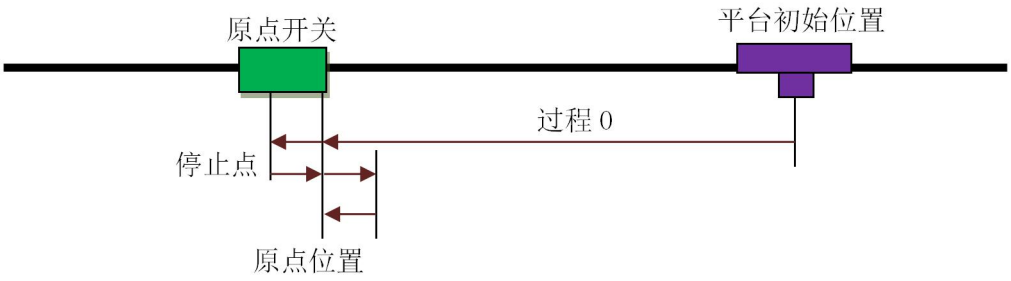


图 2.11 二次回零方式示意图

方式 3：一次回零后再找 EZ 信号

该方式在回原点运动过程中，当找到原点信号后，还要等待该轴的 EZ 信号出现，此时电机停止。回原点过程如图 2.12 所示。

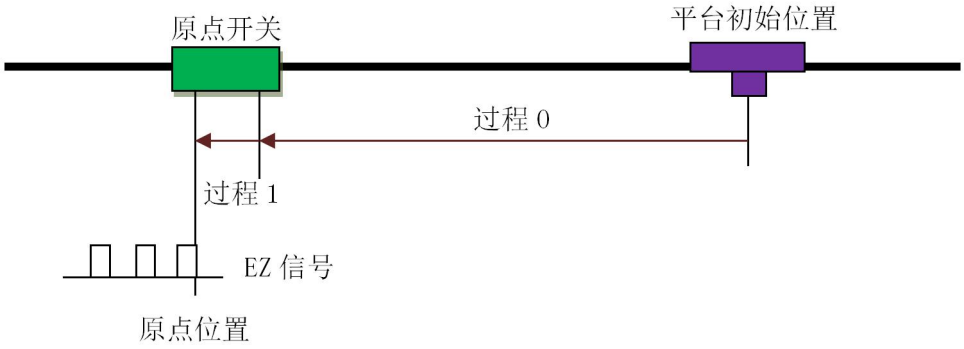



图 2.12 一次回零后再找 1 个 EZ 信号回零方式示意图

方式 4：记 1 个 EZ 信号回零

该方式在回原点运动过程中，当检测到该轴的 EZ 信号出现一次后，此时电机停止。回原点过程如图 2.13 所示。

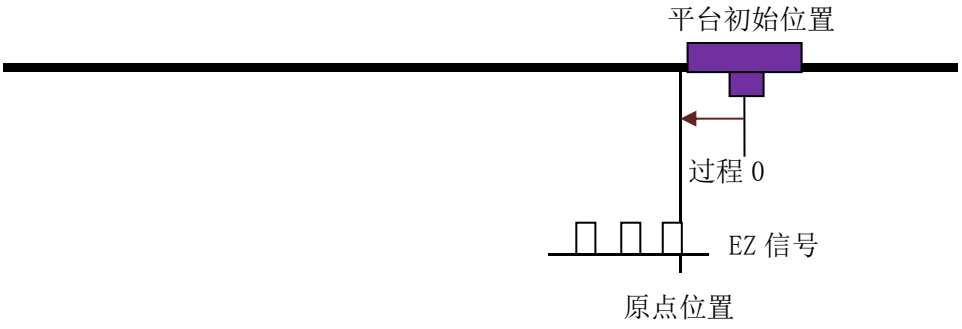



图 2.13 记 1 个 EZ 信号回零方式示意图

方式 5：一次回零再反找 EZ 信号

该方式在回原点运动过程中，当找到原点信号后，减速停止，然后以反找速度反向找到 EZ 生效此时电机停止。回原点过程如图 2.14 所示。

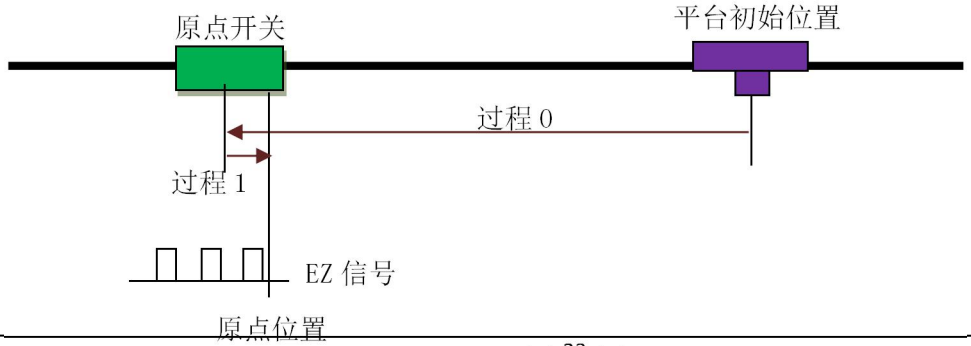


图 2.14 一次回零反找一个 EZ 进行回零

方式 6：原点锁存

如图 2.15 所示，电机先以设定速度回原点，当原点开关边沿触发时，将当前位置锁存下来，同时电机减速停止。电机减速停止完成后再反向回找锁存位置，运动到锁存位置，电机停止。


图 2.15 原点锁存回零方式示意图

方式 7：原点锁存加同向 EZ 锁存

此模式先以模式 6 的方式执行一次原点锁存回零，完成后继续沿设定回零方向运行到 EZ 信号产生，EZ 信号产生时锁存当前位置并执行减速停，电机减速停止之后再反向回找 EZ 的锁存位置，运动到锁存位置，电机停止。回原点过程如图 2.16 所示。

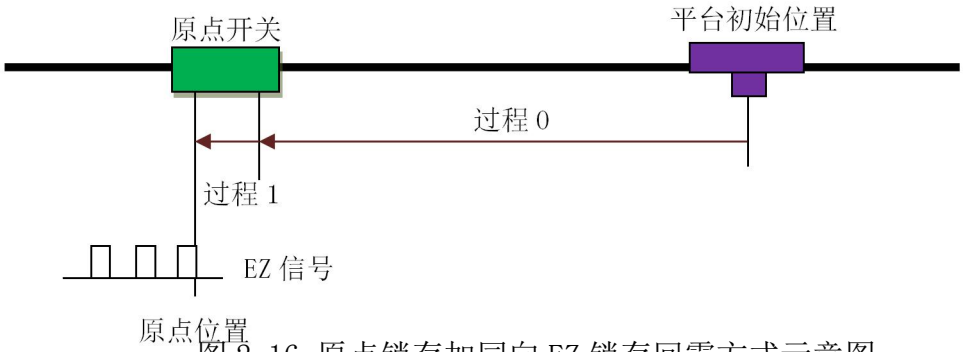



图 2.16 原点锁存加同向 EZ 锁存回零方式示意图

方式 8：单独记一个 EZ 锁存

在回零过程中检测到 EZ 有效边沿出现，锁存当前位置，执行减速停，电机减速停止之后再反向回找 EZ 的锁存位置，运动到锁存位置，电机停止。回原点过程如图 2.17 所示。

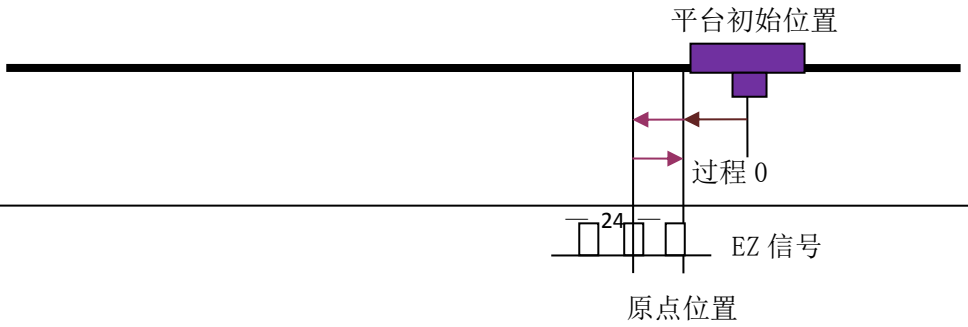


图 2.17　单独记一个 EZ 锁存回零方式示意图

方式 9：原点锁存加反向 EZ 锁存

此模式先以模式 6 的方式执行一次原点锁存回零，完成后以与设定回零方向相反的方向运行到 EZ 信号产生，EZ 信号产生时锁存当前位置并执行减速停，电机减速停止之后再反向回找 EZ 的锁存位置，运动到锁存位置，电机停止。回原点过程如图 2.18 所示。

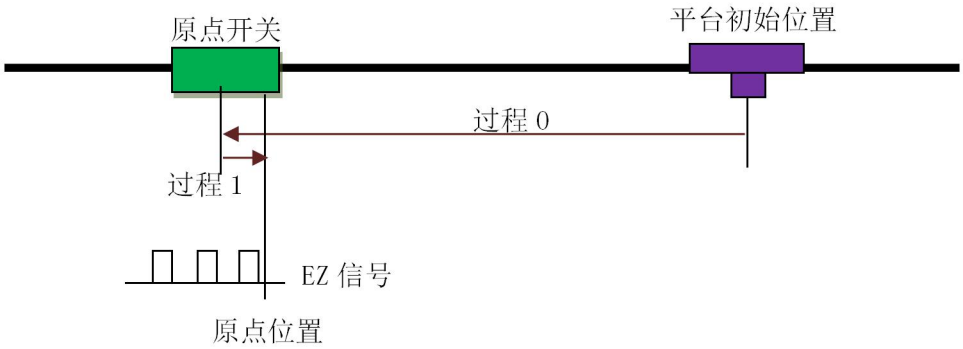



图 2.18 原点锁存加反向 EZ 锁存进行回零

回原点运动相关函数：

| 名称 | 功能 | 参考 |
|---|---|---|
| smc_set_home_pin_logic | 设置原点信号的有效电平 | 3.7 节 |
| smc_get_home_pin_logic | 读取原点原点信号设置 | |
| smc_set_homemode | 设置回原点模式 | |
| smc_get_homemode | 读取回原点模式 | |
| smc_set_ez_count | 设置回零 EZ 个数 | |
| smc_get_ez_count | 回读回零 EZ 个数 | |
| smc_set_home_position_unit | 设置回原点完成后的偏移位置值 | |
| smc_get_home_position_unit | 回读回原点完成后的偏移位置值 | |
| smc_set_home_profile_unit | 设置回原点速度参数 | |
| smc_get_home_profile_unit | 读取回原点速度参数 | |
| smc_set_el_home | 限位当原点切换函数 | |
| smc_home_move | 按指定的方向和速度方式开始回原 | |
| smc_get_home_result | 读取回原点状态 | |

例程：轴 0 运行回原点运动

```
int main(int argc, char* argv[])
{
/**********************变量定义*****************************/
      short ret = 0;           //错误返回
      short ConnectNo=0;        //链接号，范围：0~7
      WORD axis = 0;           //运动轴号，范围：0~最大轴数-1
      double Start_Vel = 1000;//回零起始速度，范围：0~2MHz 频率
      double Max_Vel = 1000; //回零运行速度，范围：0~2MHz 频率
      double Tacc = 0.1;      //加速时间，单位 s，范围：0.001~10s
      WORD org_logic=0;    //设置原点有效电平：0-低电平，1-高电平
      double filter=0;     //滤波时间为 0，保留参数，无意义
      WORD home_dir=1;    //设置回原点方向：0-负向、1-正向
      WORD mode =0;       //设置回原点模式为一次回原点
      WORD Source =0;    // 设置计数源为脉冲计数
      WORD enable =0;    // 设置完成回原点后计数使能
      double position =100;    // 设置完成回原点后计数值

/**********************函数调用执行*************************/
       //第一步、设置回原点电平参数
      ret=smc_set_home_pin_logic(ConnectNo,axis,org_logic,filter);
      //第二步、设置回原点模式
      ret=smc_set_homemode(ConnectNo,axis,home_dir,1,mode, Source);
       //第三步、设置回原点完成后计数位置值
     ret=smc_set_home_position_unit(ConnectNo,axis,enable,position);
      //第四步、设置回原点运动速度参数
      ret=smc_set_home_profile_unit(ConnectNo,axis,Start_Vel,Max_Vel,Tacc,0);
       //第五步、启动回原点运动
      ret=smc_home_move(ConnectNo,axis);
}
```

### 2.2.3 PVT 运动

雷赛控制器中提供了两种 PVT 模式来实现单轴任意速度规划的功能，分别为 PTT 运动模式和 PTS 运动模式。PTT 运动模式是用于单轴梯形速度的规划，而 PTS 运动模式则用于单轴 S 形速度的规划。

### 2.2.3.1 单轴速度规划功能

#### (1) PTT 运动模式

PTT 模式非常灵活，能够实现单轴任意速度规划。用户通过直接输入位置和时间参数描述运动规律。

相关函数：

| 名称 | 功能 | 参考 |
|---|---|---|
| smc_ptt_table_unit | 向指定数据表传送数据，采用 PTT 描述方式 | 3.8 节 |
| smc_pvt_move | 启动 PVT 运动 | |

例程：PTT 模式运动

规划如图 2.4 所示速度曲线，用 PTT 模式规划该速度曲线非常简单。

首先计算各段的位移量，即速度曲线和时间轴所围面积：P1=1500(unit)，P2=4000(unit)，P3=8500(unit)，P4=24000(unit)，P5=27000(unit)，P6=3000(unit)。

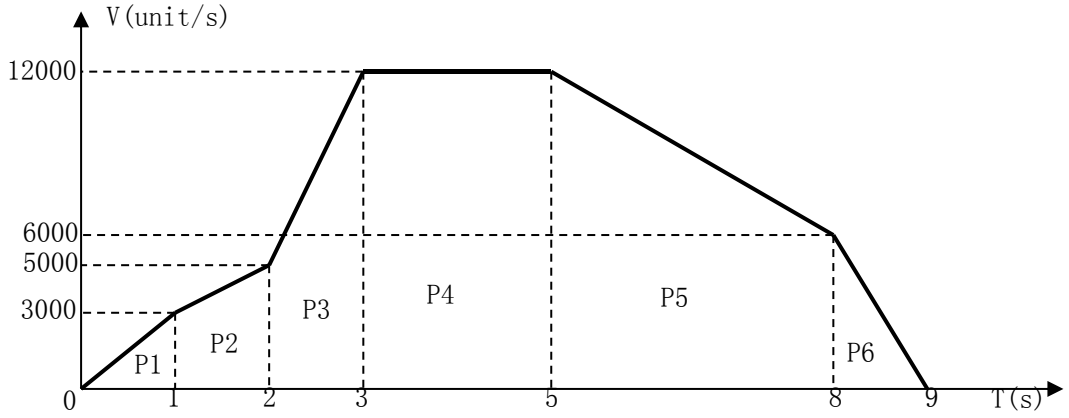



图 2.4　PTT 模式下的 V-T 曲线规划

将各段位移量累加，得到 PTT 模式下各点的位置和时间数据，如表 2.5 所示。

表 2.5 PTT 模式数组数据

| 序号 | 位置 P(unit) | 时间 T(s) |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 1500 | 1 |
| 2 | 5500 | 2 |
| 3 | 14000 | 3 |

| | | |
|---|---|---|
| 4 | 38000 | 5 |
| 5 | 65000 | 8 |
| 6 | 68000 | 9 |

编写程序如下：

```
int main(int argc, char* argv[])
{
/**********************变量定义*****************************/
        WORD MyCardNo =0;    //连接号，可选 0--7
        WORD ret=0 ;         //返回错误码
         WORD My_AxisList=0;     //运动轴
         WORD MyCount = 7;       //运动点数
         double MyPTime[7];     //运动点时间参数
         double MyPPos[7];     //运动点位置参数
         MyPTime[0] = 0;
         MyPPos[0] = 0;
         MyPTime[1] = 1;
         MyPPos[1] = 1500;
         MyPTime[2] = 2;
         MyPPos[2] = 5500;
         MyPTime[3] = 3;
         MyPPos[3] = 14000;
         MyPTime[4] = 5;
         MyPPos[4] = 38000;
         MyPTime[5] = 8;
         MyPPos[5] = 65000;
         MyPTime[6] = 9;
         MyPPos[6] = 68000;
         WORD MyAxisNum = 1;      //参与 PVT 运动的轴数为 1

/**********************函数调用执行*************************/
      //第一步、设置 PTT 运动参数
      ret=smc_ptt_table_unit(MyCardNo,My_AxisList,MyCount,MyPTime,MyPPos);
      //第二步、启动 PTT 运动
     ret=smc_pvt_move(MyCardNo,MyAxisNum,&My_AxisList);
 }
```

PTT 运动结果(位置-时间曲线、加速度-时间曲线)如图 2.6、2.7 所示。

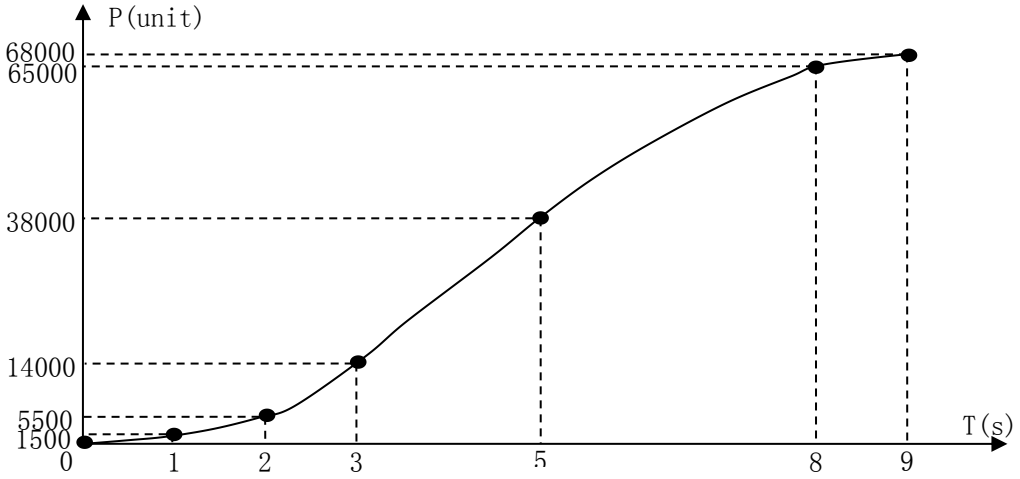



图 2.6 PTT 运动位移曲线

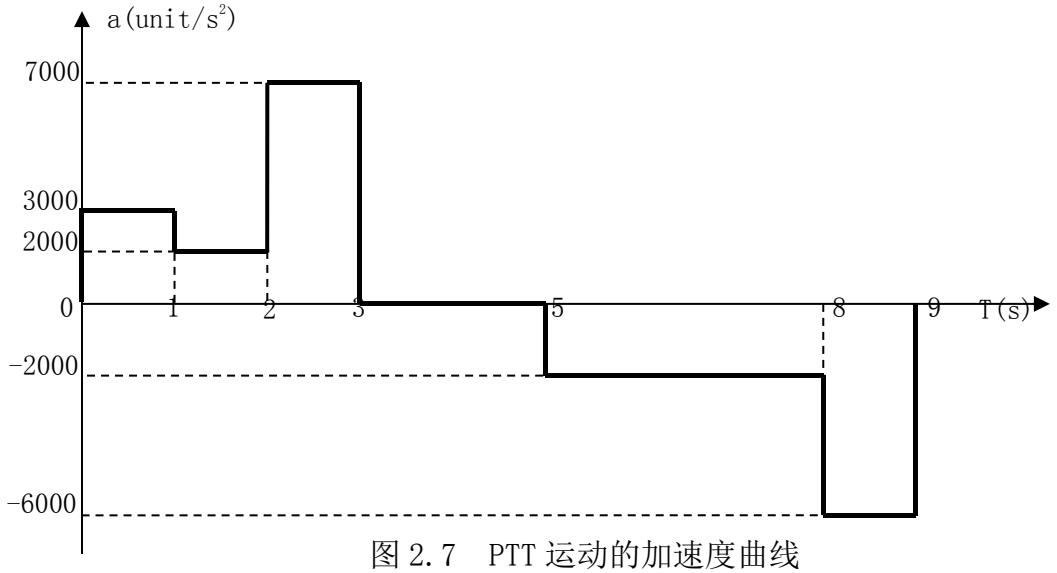



图 2.7　PTT 运动的加速度曲线

### (2) PTS 运动模式

PTS 运动模式是 PTT 的扩展功能模式，可以使各数据点的速度过渡更加平滑。用户通过输入位置、时间、百分比参数描述运动规律。数据点的百分比参数是指：相邻 2 个数据点之间加速度的变化时间占速度变化时间的百分比。

相关函数：

| 名称 | 功能 | 参考 |
|---|---|---|
| smc_pts_table_unit | 向指定数据表传送数据，采用 PTS 描述方式 | 3.8 节 |
| smc_pvt_move | 开始 PVT 运动 | |

例程：PTS 模式运动

由图 2.9 可知，例程中的加速度曲线存在突变，因此，运动过程中有冲击现象。如果要获得更加平滑的速度曲线，可以使用 PTS 模式运动，其速度曲线比 PTT 的速度曲线平滑。

设置各点的速度百分比参数如图 2.8 所示；其对应的加速度-时间曲线如图 2.9 所示，每段的加减速时间为该段时间值×速度百分比；每段的加速度最大值与 PTT 模式下的加速度值不一样，这是因为内部算法作了平滑处理。其对应的位置-时间曲线、速度-时间曲线如图 2.10、2.11 所示。

| 序号 | P(unit) | T(s) | Percent(%) |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 1500 | 1 | 20 |
| 2 | 5500 | 2 | 40 |
| 3 | 14000 | 3 | 60 |
| 4 | 38000 | 5 | 0 |
| 5 | 65000 | 8 | 20 |
| 6 | 68000 | 9 | 80 |

图 2.8 PTS 模式下的加速度曲线

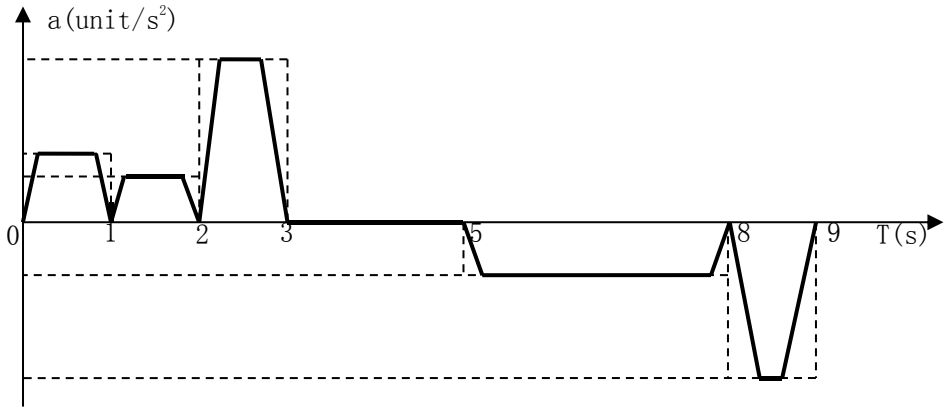



图 2.9 PTS 模式下的加速度曲线

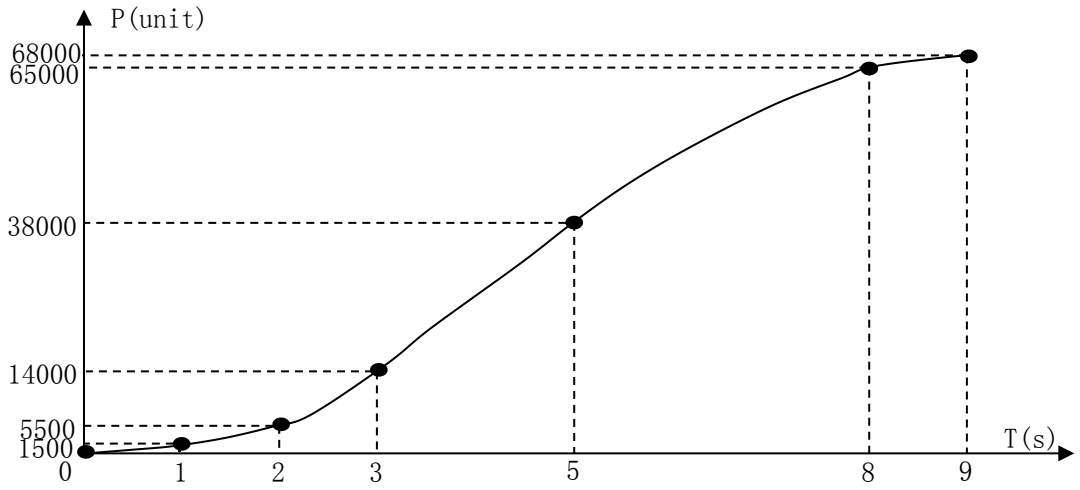



图 2.10　PTS 运动得到的位移曲线

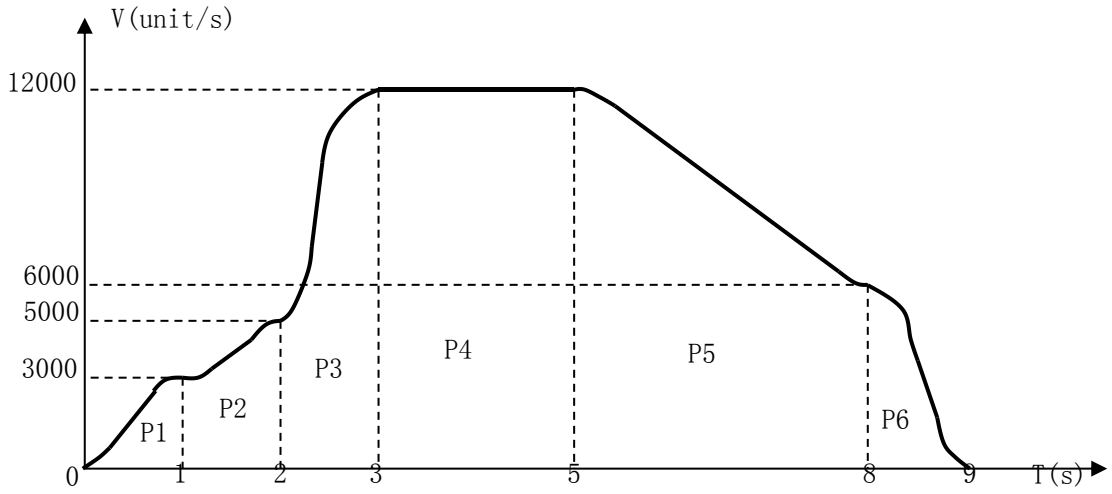



图 2.11 PTS 运动得到的速度曲线

编写程序如下：

```
int main(int argc, char* argv[])
{
/**********************变量定义*****************************/
        short ret = 0;            //错误返回
        short MyCardNo =0;       //控制器连接号
         WORD My_AxisList=0;     //运动轴
         WORD MyCount = 7;              //运动点数
         double MyPTime[6];     //运动点时间参数
         double MyPPos[6];     //运动点位置参数
         double MyPPer[6];     //运动加速度百分比
        MyPTime[0] = 0;
        MyPPos[0] = 0;
```

```
        MyPPer[0]=0;
        MyPTime[1] = 1;
        MyPPos[1] = 1500;
        MyPPer[1]=20;
        MyPTime[2] = 2;
        MyPPos[2] = 5500;
        MyPPer[2]=0;
        MyPTime[3] = 3;
        MyPPos[3] = 14000;
        MyPPer[3]=60;
        MyPTime[4] = 5;
        MyPPos[4] = 38000;
        MyPPer[4]=0;
        MyPTime[5] = 8;
        MyPPos[5] = 65000;
        MyPPer[5]=20;
        MyPTime[6] = 9;
        MyPPos[6] = 68000;
        MyPPer[6]=80;
        WORD MyAxisNum = 1;      //参与 PVT 运动的轴数为 1

/**********************函数调用执行**************************/
   //第一步、设置 PTS 运动参数
    ret=smc_pts_table_unit(MyCardNo,My_AxisList,MyCount,MyPTime,MyPPos, MyPPer);
  //第二步、启动 PTS 运动
  ret=smc_pvt_move(MyCardNo,MyAxisNum,&My_AxisList);
 }
```

### 2.2.3.2 多轴速度规划功能

雷赛控制器具有 PVT 高级运动曲线规划的功能，当用户需要规划一些特殊的运动轨迹而使用单轴运动及插补运动无法满足需求时，可以尝试使用 PVT 来规划自己的运动轨迹。

雷赛控制器共提供了两种 PVT 模式来实现多轴轨迹规划的功能，分别为 PVT、PVTS 运动模式。PVT 模式用于对各点的位置、时间、速度都有要求的轨迹规划，PVTS 模式用于只对各点的位置、时间有要求，而对各点的速度无太多要求的轨迹规划。

**（1）PVT 运动模式**

PVT 模式使用一系列数据点的位置、速度、时间参数描述运动规律。

相关函数：

| 名称 | 功能 | 参考 |
|---|---|---|
| smc_pvt_table_unit | 向指定数据表传送数据，采用 PVT 描述方式 | 3.8 节 |
| smc_pvt_move | 开始 PVT 运动 | |

⚠注意：当设置的各点 P、V、T 数据不合理时，很难得到理想的轨迹曲线。

理想轨迹上取点越多，实际轨迹越接近理想轨迹。

例程：PVT 模式运动

如图 2.12 所示，要控制运动平台按椭圆轨迹运动，但雷赛控制器中并没有提供椭圆插补函数，用户可以使用 PVT 函数自己设计该轨迹。

设该椭圆的长半轴长 9000unit，短半轴长 7000unit；椭圆轨迹的角速度ω恒定，轨迹运动的总时间为 10s。

显然该椭圆的方程为：

$$\begin{cases} x = 9000\cos(\theta) + 9000 \\ y = 7000\sin(\theta) \end{cases}$$

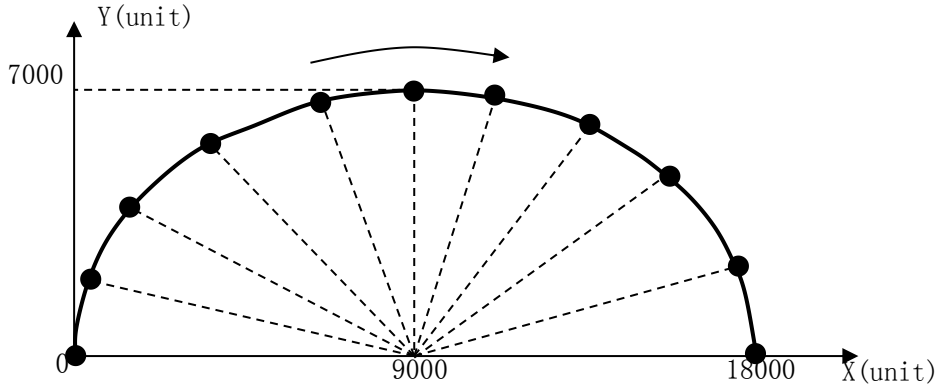



图 2.12　上半椭圆轨迹和分段

使用 PVT 模式运动上半椭圆轨迹的步骤为：

1、 将该轨迹分成圆弧角相等的十段轨迹；计算各点坐标值，即得 P 值。程序如下：

```
short ret;
WORD MyCount = 7;        //运动点数
double MyPPosX[11];      //定义数组，用于存储 PVT 的位置数据（X 轴）
```

```
double MyPPosY[11];     //定义数组, 用于存储 PVT 的位置数据（Y 轴）
WORD a, b, i ;
    a = 9000;     //定义椭圆长半轴
    b = 7000;      //短半轴长
const double PI = 3.14159265358979323846;
 for(i = 0;i< 11;i++)                  //计算各点的 X、Y 坐标
    {

        MyPPosX[i] = a * cos((10 - i) * PI / 10) + a;
         MyPPosY[i] = b * sin((10 - i) * PI / 10);
    }
    MyPPosX[0]=0;  //第一点位置为 0
    MyPPosY[0]=0;
```

2、根据各点坐标（即 P 值），计算出各点对应的速度的（V 值）和时间（T 值）。

对椭圆方程公式求导，可得 X、Y 轴方向的速度分量为：

$$\begin{cases} \dot{x} = -9000\sin(\theta)\dfrac{d\theta}{dt} \\ \dot{y} = 7000\cos(\theta)\dfrac{d\theta}{dt} \end{cases}$$

上式中 $\frac{d\theta}{dt}$ 即为角速度ω。

各点的 X、Y 轴方向的速度分量可由以下程序计算得出。程序如下：

```
double MyPVelX[11];   //定义数组,用于存储 PVT 中的速度数据（X 轴）
double MyPTimeX[11];  //定义数组,用于存储 PVT 中的时间数据（X 轴）
double MyPVelY[11];   //定义数组,用于存储 PVT 中的速度数据（Y 轴）
double MyPTimeY[11];   //定义数组，用于存储 PVT 中的时间数据（Y 轴）
double MyWVel;          // 定义角速度
 for(i = 0;i< 11;i++)
    {
        MyPTimeX[i] = i;                  // 存储 X 轴各点时间数据
        MyPTimeY[i] = i;                    // 存储 Y 轴各点时间数据
    }
 MyWVel = -PI/ 10;                 // 计算角速度
 MyPVelX[0] = MyPVelX[10]=0;      //起始点与终止点 X 轴速度设为 0
 MyPVelY[0] =MyPVelY[10]= 0;     //起始点与终止点Y轴速度设为0
 for(i = 0;i< 9;i++)
 {
```

```
        MyPVelX[i + 1] = -a * sin((10 - i - 1) * PI / 10) * MyWVel ;
                        //计算其他点 X 轴速度
        MyPVelY[i + 1] = b * cos((10 - i - 1) * PI / 10) * MyWVel ;
                        //计算其他点 Y 轴速度
    }
```

计算出各点 X、Y 轴的 P、V、T 数据如表 2.13 所示。

图 2.13　PVT 数据

| | X 轴 | | | Y 轴 | | |
|---|---|---|---|---|---|---|
| 序号 | P(unit) | V(unit/s) | T(s) | P(unit) | V(unit/s) | T(s) |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 440 | 873.731 | 1 | 2163 | 2091.479 | 1 |
| 2 | 1719 | 1661.927 | 2 | 4115 | 1779.117 | 2 |
| 3 | 3710 | 2287.443 | 3 | 5663 | 1292.603 | 3 |
| 4 | 6219 | 2689.048 | 4 | 6657 | 679.560 | 4 |
| 5 | 9000 | 2827.431 | 5 | 7000 | -0.003 | 5 |
| 6 | 11781 | 2689.046 | 6 | 6657 | -679.566 | 6 |
| 7 | 14290 | 2287.438 | 7 | 5663 | -1292.608 | 7 |
| 8 | 16281 | 1661.921 | 8 | 4114 | -1779.120 | 8 |
| 9 | 17560 | 873.724 | 9 | 2163 | -2091.481 | 9 |
| 10 | 18000 | 0 | 10 | 0 | 0 | 10 |

3、使用 smc_pvt_table_unit 函数向数据表传递数组数据。

程序如下：

```
WORD MyCardNo;
WORD My_AxisList[2];  //定义 PVT 运动轴列表变量
int MyCountX ;                  //定义 X 轴的 PVT 数据点编号变量
int MyCountY;             //定义 Y 轴的 PVT 数据点编号变量
MyCardNo = 0;                          // 连接号
My_AxisList[0] = 0;     // X、Y 轴)参与 PVT 运动
My_AxisList[1] = 1;
MyCountX =11;
MyCountY =11;
```

```
ret=smc_pvt_table_unit(MyCardNo, 0, MyCountX, MyPTimeX,MyPPosX, MyPVelX);    //
    以 PVT 描述方式向 X 轴传送 PVT 数据

ret=smc_pvt_table_unit(MyCardNo, 1, MyCountY, MyPTimeY, MyPPosY, MyPVelY);
    // 以 PVT 描述方式向 Y 轴传送 PVT 数据
```

4、使用 smc_pvt_move 函数执行 PVT 运动。

程序如下：

```
    WORD My_AxisNum = 2;                    //参与 PVT 运动的轴数为 2
ret=smc_pvt_move (MyCardNo, My_AxisNum, My_AxisList); //启动两轴 PVT 运动
```

执行完上述程序后，得到 PVT 运动轨迹如图 2.14 所示。

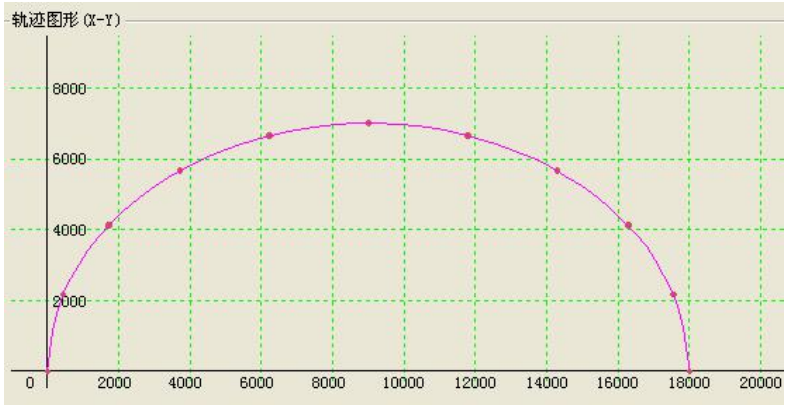



图 2.14　PVT 模式运动得到的上半椭圆轨迹

### (2) PVTS 运动模式

PVTS 运动模式只需要定义理想轨迹上数据点的位置和时间，以及起点速度和终点速度。运动控制器根据各数据点的位置、时间参数计算各点之间的轨迹位置和速度，确保各段轨迹的速度连续、加速度连续。相关函数：

| 名称 | 功能 | 参考 |
|---|---|---|
| smc_pvts_table_unit | 向指定数据表传送数据，采用 PVTS 描述方式 | 3.8 节 |
| smc_pvt_move | 启动 PVT 运动 | |

例程：PVTS 模式运动

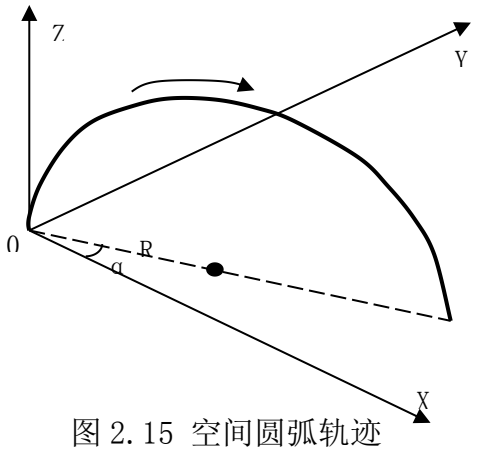



图 2.15 空间圆弧轨迹

如图 2.15 所示，设计空间圆弧轨迹。设其半径 R=15000unit，其映射在 XY 平面上的轨迹与 X 轴的夹角 α = π /6，轨迹运动总时间为 10s。

显然该空间圆弧的方程为：

$$\begin{cases} x = 15000\cos(\frac{\pi}{6})\cos(\theta) + 15000\cos(\frac{\pi}{6}) \\ y = 15000\sin(\frac{\pi}{6})\cos(\theta) + 15000\sin(\frac{\pi}{6}) \\ z = 15000\sin(\theta) \end{cases}$$

若对轨迹上各点速度无精确要求，采用 PVTS 模式设计该轨迹的步骤如下：

1、将该轨迹分成角度相等的十份，计算各点的位置坐标，即 P 值。

2、根据各点的 P 值，计算出各点的 T 值。设每段轨迹运动的时间相等。

3、设定起点速度与终点速度都为 0。

4、使用 smc_pvts_table_unit 函数向数据表传递数组数据。

5、使用 smc_pvt_move 函数执行 PVT 运动。

编写程序如下：

```
int main(int argc, char* argv[])
{
/**********************变量定义*****************************/
    short ret;          //返回错误码
    double MyPTimeX[11],MyPVelBeginX,MyPVelEndX;    //定义变量（X 轴）
    double MyPTimeY[11],MyPVelBeginY,MyPVelEndY;    //定义变量（Y 轴）
    double MyPTimeZ[11],MyPVelBeginZ,MyPVelEndZ;;    //定义变量（Z 轴）
    WORD MyCountX,MyCountY,MyCountZ,MyCardNo,My_AxisNum;
    double MyPPosX[11],MyPPosY[11] ,MyPPosZ[11];    //定义数组
    WORD My_AxisList[3];     //定义运动轴列表
   WORD R=15000;     //定义圆半径
```

```
WORD i;
const double pi = 3.14159265358979323846;   // 定义圆周率          MyCountX =
MyCountY =MyCountZ = 11;  //设置 X、Y、Z 轴的数据点数
MyPVelBeginX = MyPVelEndX = 0;     //X 轴的起点速度及终点速度为 0
MyPVelBeginY = MyPVelEndY = 0;      //Y 轴的起点速度及终点速度为 0
MyPVelBeginZ = MyPVelEndZ = 0;     //Z 轴的起点速度及终点速度为 0
MyCardNo = 0;                                  //连接号
My_AxisList[0]=0;        //0、1、2 号轴（即 X、Y、Z 轴）参加 PVT 运动
My_AxisList[1]=1;
My_AxisList[2]=2;
My_AxisNum = 3;          //3 个轴参与 PVT 运动

/*********************函数调用执行**************************/
//第一步、计算 X、Y、Z 轴的位置坐标值
for(i = 0;i<11;i++)
{
  MyPPosX[i] = R * cos(pi / 6) * cos((10 - i) * pi / 10) + R * cos(pi / 6);
       //计算 X 轴各点位置坐标
  MyPPosY[i] = R * sin(pi / 6) * cos((10 - i) * pi / 10) + R * sin(pi / 6);
       //计算 Y 轴各点位置坐标
  MyPPosZ[i] = R * sin((10 - i) * pi / 10); //计算 Z 轴各点位置坐标
    }
  MyPPosX[0]=MyPPosY[0]=MyPPosZ[0]=0;      //第一个点位位置为 0

  //第二步、计算 X、Y、Z 轴的各点时间值
for(i = 0;i<11;i++)
   {
   MyPTimeX[i] = i;              //计算 X 轴各点时间
   MyPTimeY[i] = i;              //计算 Y 轴各点时间
   MyPTimeZ[i] = i;              //计算 Z 轴各点时间
   }

//第三步、向 X、Y、Z 轴传送 PVTS 数据
ret=smc_pvts_table_unit(MyCardNo,0,MyCountX,MyPTimeX,MyPPosX,MyPVelBeginX,
MyPVelEndX);       //向 X 轴传送 PVTS 数据
ret=smc_pvts_table_unit(MyCardNo,1,MyCountY,MyPTimeY,  MyPPosY,  MyPVelBeginY,
MyPVelEndY);   //向 Y 轴传送 PVT 数据
ret=smc_pvts_table_unit(MyCardNo,2,MyCountZ,MyPTimeZ,
```

```
    MyPPosZ,MyPVelBeginZ,MyPVelEndZ);  //向 Z 轴传送 PVT 数据

    //第四步、启动 X、Y、Z 轴 PVTS 运动
    ret=smc_pvt_move(MyCardNo, My_AxisNum, My_AxisList);
}
```

以 PVTS 模式得到的空间圆弧轨迹如图 2.16 所示。

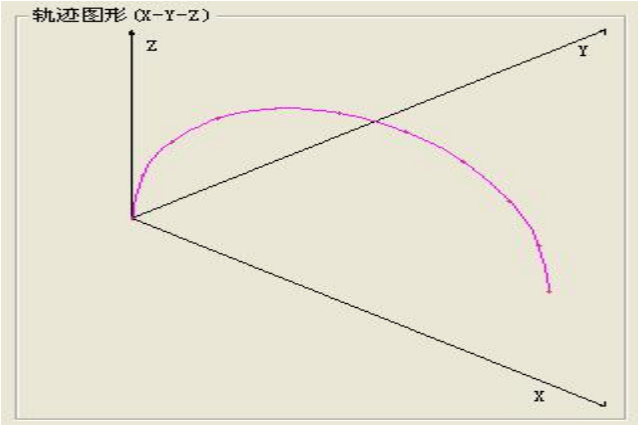



图 2.16 PVTS 模式运动得到的空间圆弧轨迹

## 2.2.4 插补运动

插补运动模式可以实现多轴的协调运动，从而完成一定的运动轨迹。插补运动模式又可以分为多轴插补与连续插补两种。

多轴插补可实现单段直线插补、单段圆弧插补、单段螺旋插补等运动。

连续插补可分为非前瞻和前瞻。可实现各种插补曲线连续运行，可实现速度的平滑过渡，减少机器振动，提高加工效率与加工精度。

### 2.2.4.1 参数设置

插补运动的速度、加减速时间、平滑 S 段时间等参数的设置，由不同的函数实现。

插补速度曲线有 T 形和 S 形，其速度曲线规划见图 2.17

“起始速度”：设置插补运动的起始速度。

“插补速度”：设置插补运动时的最大运行速度，为插补轴合速度值。

“终止速度”：设置插补运动的停止速度。

“加速时间”：设置插补运动时从起始速度加速到最大运行速度需要的时间 Tdec

“减速时间”：设置插补运动时从最大运行速度减速到停止速度需要的时间 Tacc

“S 段时间”：设置插补合速度曲线 S 段的时间参数（spara）。

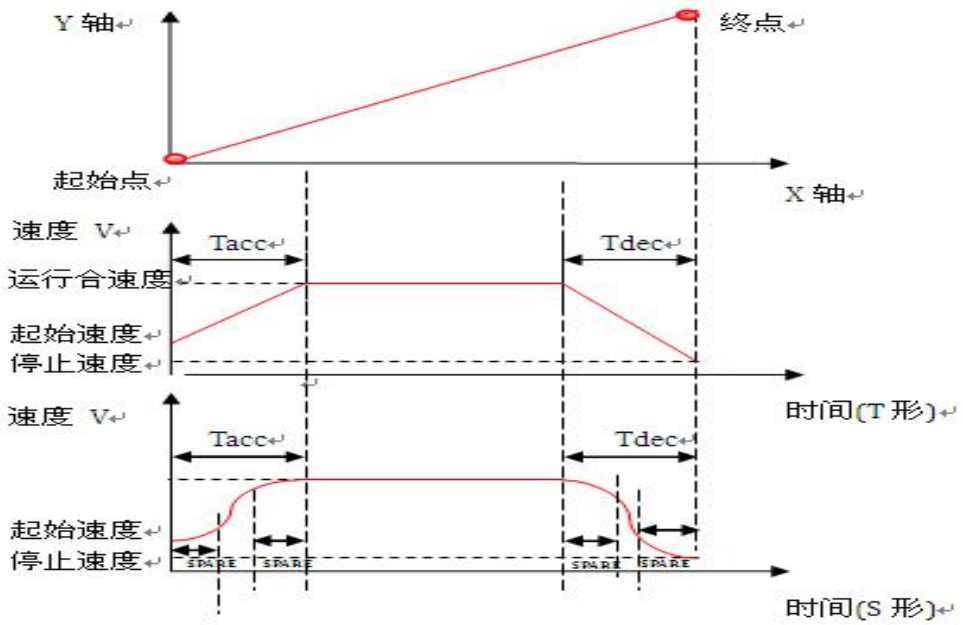



图 2.17 T 形、S 形速度曲线及对应位置图

相关函数：

| 名称 | 功能 | 参考 |
|---|---|---|
| smc_set_vector_profile_unit | 设置插补运动速度曲线 | 3.9 节 |
| smc_get_vector_profile_unit | 读取插补运动速度曲线 | |
| smc_set_vector_s_profile | 设置插补运动速度曲线的平滑时间 | |
| smc_get_vector_s_profile | 读取插补运动速度曲线的平滑时间 | |

### 2.2.4.2 单段插补

单段插补运动可以实现多轴的协调运动，如 2-6 轴直线插补，平面圆弧插补，空间圆弧插补，螺旋插补。当参与空间圆弧插补或螺旋插补的轴数多于 3 轴时，后面的轴自动跟随前三轴做线性运动。圆弧插补支持圆弧限速功能，即限制运行速度，确保加速度为设定值，可减少运行震动。

相关函数：

| 名称 | 功能 | 参考 |
|---|---|---|
| smc_line_unit | 直线插补运动 | 3.10 节 |
| smc_arc_move_center_unit | 圆心+终点模式的圆弧插补运动 | |
| smc_arc_move_radius_unit | 半径+终点模式的圆弧插补运动 | |

| 名称 | 功能 | 参考 |
| --- | --- | --- |
| smc_arc_move_3points_unit | 三点模式的圆弧插补运动 | |

例程：XY 轴直线插补

```
int main(int argc, char* argv[])
{
/**********************变量定义*****************************/
      short MyCardNo=0; //连接号
      short ret;       //返回错误码
      WORD Myposi_mode = 0;      //0:相对模式，1：绝对模式
      WORD MyCrd = 0;            //参与插补运动的坐标系
      WORD AxisArray[2];        //定义轴
      AxisArray[0] = 0;        //定义插补 0 轴为 X 轴
      AxisArray[1] = 1;        //定义插补 1 轴为 Y 轴
      double MyMin_Vel = 0;           //起始速度 0
      double MyMax_Vel = 3000;        //插补运动最大速度
      double MyTacc = 0.2;            //插补运动加速时间
      double MyTdec = 0.1;             //插补运动减速时间
      double MyStop_Vel = 0;          //插补运动停止速度
      WORD MySmode = 0;              //保留参数，固定值为 0
      double  MySpara = 0.05;          //平滑时间为 0.05s
      WORD MyaxisNum = 2;           //插补运动轴数为 2
      double Dist[2];
      Dist[0] = 10000 ;      //定义 X 轴运动距离
      Dist[1] = 8000 ;       //Y 轴运动距离

/**********************函数调用执行*************************/
   //第一步、设置插补运动速度参数
   ret=smc_set_vector_profile_unit(MyCardNo,MyCrd,MyMin_Vel,MyMax_Vel,MyTacc,MyTd
ec,MyStop_Vel);
   //第二步、设置插补运动平滑参数
   ret=smc_set_vector_s_profile( MyCardNo,MyCrd,MySmode,MySpara );
  //第三步、启动插补运动
 ret=smc_line_unit( MyCardNo,MyCrd,MyaxisNum,AxisArray,Dist, Myposi_mode);
 }
```

例程 2：XY 轴圆弧插补，圆心+终点模式

```
int main(int argc, char* argv[])
{
/**********************变量定义*****************************/
      short MyCardNo=0; //连接号
      short ret;         //返回错误码
       WORD Myposi_mode = 0;      //0:相对模式，1：绝对模式                    WORD
       MyCrd = 0;             //参与插补运动的坐标系
       WORD AxisArray[2];
       AxisArray[0] = 0;        //定义插补 0 轴为 X 轴
       AxisArray[1] = 1;        //定义插补 1 轴为 Y 轴
       double MyMin_Vel = 0;  //保留参数
      double MyMax_Vel = 3000;       //插补运动最大矢量速度 3000unit/s
       double MyTacc = 0.2;           //插补运动加速时间 0.2s
       double MyTdec = 0.1;             //运行减速时间
        double MyStop_Vel = 0;         //停止速度
       WORD MySmode = 0;             //保留参数，固定值为 0
       double  MySpara = 0.0;          //平滑时间为 0.05s
        WORD MyaxisNum = 2;           //插补运动轴数为 2
        WORD dir=0;           //圆弧方向，0：顺时针，1：逆时针
        long cic=0;           //圆弧圈数
        double cen[2];        //定义圆心坐标
        cen[0] = 10000 ;       //定义 X 轴圆心坐标
        cen[1] = 0 ;       //定义 Y 轴圆心坐标
        double Dist[2];       //定义终点坐标
        Dist[0] = 0 ;      //定义 X 轴运动终点
        Dist[1] = 0 ;       //定义 Y 轴运动终点

/**********************函数调用执行*************************/
   //第一步、设置插补运动速度参数
    ret=smc_set_vector_profile_unit(MyCardNo,MyCrd,MyMin_Vel,MyMax_Vel,MyTacc,MyTd
ec,MyStop_Vel);
  //第二步、设置 S 时间参数
  ret=smc_set_vector_s_profile( MyCardNo,MyCrd,MySmode,MySpara );
   //第三步、执行圆弧插补运动
  ret=smc_arc_move_center_unit(MyCardNo,MyCrd,MyaxisNum,AxisArray,Dist,cen,dir,ci
c,Myposi_mode);
}
```

### 2.2.4.3 连续插补

控制器提供了连续插补运动功能，包含前瞻（模式 1）和非前瞻运动（模式 0、模式 2，两种算法不一样）。连续插补指令支持直线插补，圆弧插补，螺旋线插补，IO 控制等，前瞻和非前瞻主要区别在于前瞻可很好应用于小线段轨迹，其轨迹连接处更平滑。

连续插补对于圆弧提供了圆弧限速功能功能，主要为限制运行速度，使加速度值不超出设定范围。 前瞻（模式 1）默认为圆弧限速启用，非前瞻模式 0 不支持圆弧限速，非前瞻模式 2 可自由设置圆弧限速功能。具体功能见下表：

| 连续插补功能 | 非前瞻(模式 0) | 前瞻(模式 1) | 非前瞻(模式 2) |
|---|---|---|---|
| 插补系 | 4 个 | 2 个 | 2 个 |
| 长线段轨迹 | 支持 | 支持 | 支持 |
| 小线段轨迹 | 不支持 | 支持 | 不支持 |
| 圆弧限速 | 不支持 | 支持 | 可自由设定 |
| Blend 功能 | 支持 | 不支持 | 不支持 |
| T\S 型曲线 | 支持 T、S 型 | 支持 T 型 | 支持 T、S 型 |
| 加、减速时间 | 可自由设定 | 对称曲线 | 可自由设定 |
| 起始、停止速度 | 可自由设定 | 不支持 | 可自由设定 |
| 速度倍率 | 支持(下一轨迹生效) | 支持(立即生效) | 支持(立即生效) |
| 插补延时 | 支持 | 支持 | 支持 |
| IO 等待输入 | 支持 | 支持 | 支持 |
| IO 立即输出 | 支持 | 支持 | 支持 |
| IO 超前、滞后输出 | 支持 | 支持 | 支持 |

此外，雷赛控制器支持两个坐标系，每个坐标系的连续缓冲区最多可缓存 5000 条指令。两个坐标系的速度可独立设置，执行连续插补时两个坐标系可独立进行连续插补运动，即可同时进行两组连续插补运动。

实现基本连续插补运动的一般步骤如下：

1） 如需要小线段前瞻功能，使用函数 smc_conti_set_lookahead_mode 设置连续插补前瞻模式及参数；

2） 使用函数 smc_conti_open_list 打开连续插补缓冲区；

3） 使用 smc_set_vector_speed_unit、smc_set_vector_s_profile 函数设置连续插补速度曲线；

4）编写连续插补运动指令；

5）使用函数 smc_conti_start_list 启动连续插补运动；

6）使用函数 smc_conti_close_list 关闭连续插补缓冲区。

需要注意的是 smc_conti_set_lookahead_mode 设置连续插补前瞻模式及参数指令必须在 smc_conti_open_list 打开连续插补缓冲区指令前调用。

下面列出了连续插补中需要调用的函数列表：

a. 连续插补初始化及状态检测函数如下表所示。

| 名称 | 功能 | 参考 |
|---|---|---|
| smc_conti_set_lookahead_mode | 设置连续插补前瞻模式及参数 | 3.11 节 |
| smc_conti_get_lookahead_mode | 回读连续插补前瞻模式及参数 | |
| smc_conti_open_list | 打开连续插补缓冲区 | |
| smc_conti_close_list | 关闭连续插补缓冲区 | |
| smc_conti_start_list | 开始连续插补 | |
| smc_conti_pause_list | 暂停连续插补 | |
| smc_conti_stop_list | 停止连续插补 | |
| smc_conti_remain_space | 查询插补缓冲区剩余插补空间 | |
| smc_conti_read_current_mark | 读连续插补缓冲区当前插补段 | |
| smc_conti_get_run_state | 读取连续插补运动状态 | |
| smc_check_done_multicoor | 检测连续插补运行状态 | |

b. 连续插补运动相关函数如下表所示。

| 名称 | 功能 | 参考 |
|---|---|---|
| smc_conti_line_unit | 连续插补中直线插补指令 | 3.11 节 |
| smc_conti_arc_move_center_unit | 基于圆心圆弧扩展的螺旋线连续插补运动 | |
| smc_conti_arc_move_radius_unit | 基于半径圆弧扩展的圆柱螺旋线连续插补运动 | |
| smc_conti_arc_move_3points_unit | 基于三点圆弧扩展的圆柱螺旋线连续插补运动 | |

c. 连续插补 IO 操作相关函数如下表所示。

| 名称 | 功能 | 参考 |
|---|---|---|
| smc_conti_set_pause_output | 设置连续插补暂停及异常停止时 IO 输出状态 | 3.13 节 |

| 名称 | 功能 | 参考 |
| --- | --- | --- |
| smc_conti_get_pause_output | 读取连续插补暂停及异常停止时 IO 输出状态 | |
| smc_conti_wait_input | 连续插补等待 IO 输入 | |
| smc_conti_delay_outbit_to_start | 连续插补中相对于轨迹段起点 IO 滞后输出 | |
| smc_conti_delay_outbit_to_stop | 连续插补中相对于轨迹段终点 IO 滞后输出 | |
| smc_conti_ahead_outbit_to_stop | 连续插补中相对于轨迹段终点 IO 提前输出 | |
| smc_conti_write_outbit | 连续插补中缓冲区立即 IO 输出 | |
| smc_conti_clear_io_action | 清除段内未执行完的 IO 动作 | |

d. 连续插补其它函数如下表所示。

| 名称 | 功能 | 参考 |
| --- | --- | --- |
| smc_conti_delay | 连续插补中暂停延时指令 | 3.11 节 |
| smc_conti_change_speed_ratio | 动态调整连续插补速度比例 | |
| smc_set_arc_limit | 设置圆弧插补功能 | |
| smc_get_arc_limit | 回读圆弧插补功能参数 | |
| smc_conti_set_blend | 设置连续插补 Blend 拐角过渡模式使能状态 | |
| smc_conti_get_blend | 读取连续插补 Blend 拐角过渡模式使能状态 | |

例程 1：连续插补运动，直线+圆弧（前瞻运动）

```
int main(int argc, char* argv[])
{
/**********************变量定义*****************************/
   short MyCardNo=0; //连接号
   short ret;     //返回错误码
    WORD Myposi_mode = 1;      //0:相对模式，1：绝对模式
    WORD MyCrd = 0;              //参与插补运动的坐标系
    WORD AxisArray[2];
```

```
    AxisArray[0] = 0;        //定义插补 0 轴为 X 轴
    AxisArray[1] = 1;       //定义插补 1 轴为 Y 轴
    double MyMin_Vel = 0;          //插补起始速度
    double MyMax_Vel = 30000;       //插补运动最大矢量速度 3000unit/s
    double MyTacc = 0.2;           //插补运动加速时间 0.2s
    double MyTdec = 0.1;          //插补运动减速时间 0.1s
   double MyStop_Vel = 0;         //插补停止速度
    WORD MySmode = 0;            //保留参数，固定值为 0
    double  MySpara = 0.0;         //平滑时间为 0.05s
   WORD MyaxisNum = 2;          //插补运动轴数为 2
   WORD enable=1;   //是否启用 Blend 功能，0 不使用，1 使用
    double Dist[2];          //定义终点坐标
     Dist[0] = 1000 ;        //定义 X 轴运动终点
     Dist[1] = 1000 ;       //定义 Y 轴运动终点
    WORD dir=0;           //圆弧方向，0：顺时针，1：逆时针
     long cic=0;          //圆弧圈数
     double cen[2];       //定义圆心坐标
     cen[0] = 10000 ;      //定义 X 轴圆心坐标
     cen[1] = 0 ;         //定义 Y 轴圆心坐标
      double Dist1[2];      //定义终点坐标
      Dist1[0] = 20000 ;      //定义 X 轴运动终点
      Dist1[1] = 0 ;       //定义Y轴运动终点
      WORD mode=1;           //定义连续插补使能参数，0 连续，1 前瞻运动
      long LookaheadSegment=200;    //定义插补段数:200 段
      double PathError=1;            //定义轨迹误差：1unit
      double LookaheadAcc=10000;  //定义拐弯加速度:10000unit/s2
      ArcLimit=1;     //使能圆弧限速，0：不使用，1 使能

/**********************函数调用执行**************************/
  //第一步、设置插补运动速度参数、S时间参数
  ret=smc_set_vector_profile_unit(MyCardNo,MyCrd,MyMin_Vel,MyMax_Vel,MyTacc,MyTdec,MyStop_Vel);
 ret=smc_set_vector_s_profile(MyCardNo,MyCrd,MySmode,MySpara );
  //第二步、设置圆弧限速功能使能
   ret=smc_set_arc_limit (MyCardNo,MyCrd, ArcLimit,0,0) ;
//第三步、设置前瞻参数
ret=smc_conti_set_lookahead_mode(MyCardNo,MyCrd,mode,LookaheadSegment ,PathError,LookaheadAcc);
//第四步、打开连续插补
```

```
ret=smc_conti_open_list(MyCardNo,MyCrd,MyaxisNum,AxisArray);
//第五步、开始连续插补
ret=smc_conti_start_list (MyCardNo,MyCrd);
//第六步、添加直线插补段
 ret=smc_conti_line_unit(MyCardNo,MyCrd,MyaxisNum,AxisArray,Dist,Myposi_mode,0);
//第七步、添加圆弧插补段
ret=smc_conti_arc_move_center_unit(MyCardNo,MyCrd,MyaxisNum,AxisArray ,Dist1,cen,
dir,cic,Myposi_mode,0);
//第八步、关闭连续插补缓冲区
ret=smc_conti_close_list (MyCardNo,MyCrd);
}
```

例程 2：连续插补中的 IO 功能 （非前瞻运动）

```
int main(int argc, char* argv[])
{
/**********************变量定义*****************************/
      short ret;         //错误返回
      short MyCardNo=0;  //连接号
      WORD Myposi_mode = 1;      //0:相对模式，1：绝对模式
      WORD MyCrd = 0;              //参与插补运动的坐标系
      WORD AxisArray[2];
      AxisArray[0] = 0;       //定义插补 0 轴为 X 轴
      AxisArray[1] = 1;       //定义插补 1 轴为 Y 轴
      double MyMin_Vel = 0;          //插补起始速度
      double MyMax_Vel = 3000;       //插补运动最大矢量速度 3000unit/s
      double MyTacc = 0.2;           //插补运动加速时间 0.2s
      double MyTdec = 0.1;          //插补运动减速时间 0.1s
       double MyStop_Vel = 0;         //插补停止速度
       WORD MySmode = 0;             //保留参数，固定值为 0
       double  MySpara = 0.0;         //平滑时间为 0.05s
      WORD MyaxisNum = 2;           //插补运动轴数为 2
      WORD enable=1;    //是否启用 Blend 功能，0 不使用，1 使用
      double Dist[2];          //定义终点坐标
       Dist[0] = 1000 ;        //定义 X 轴运动终点
       Dist[1] = 1000 ;       //定义 Y 轴运动终点
      WORD MyBitno = 1;  //’通用输出口 1
     WORD MyLevel = 0;  //’输出电平为低电平
     WORD MyDelayMode  =1;  //’滞后模式为滞后位置
```

```
    WORD MyDelayVal = 100;  //'滞后位置为 100 unit
    double MyRevTime = 0.5;  //'电平延时翻转时间为 0.5s
    WORD dir=0;          //圆弧方向，0：顺时针，1：逆时针
    long cic=0;          //圆弧圈数
    double cen[2];       //定义圆心坐标
    cen[0] = 10000 ;      //定义 X 轴圆心坐标
    cen[1] = 0 ;         //定义 Y 轴圆心坐标
    double Dist1[2];     //定义终点坐标
    Dist1[0] = 20000 ;     //定义 X 轴运动终点
    Dist1[1] = 0 ;       //定义Y轴运动终点
    WORD bitno=1;                  // '输入口 0
    WORD sta=0;                //  '输入口电平状态为低
    WORD TimeOut=2;            // '超时时间
    WORD mode=0;           //定义连续插补使能参数，0 连续，1 前瞻运动
    long LookaheadSegment=200;   //定义插补段数
    double PathError=0;           //定义轨迹误差
    double LookaheadAcc=1000000;  //定义拐弯加速度

/**********************函数调用执行**************************/
  //第一步、设置插补运动速度曲线
  ret=smc_set_vector_profile_unit(MyCardNo,MyCrd,MyMin_Vel,MyMax_Vel,MyTacc,MyTdec,MyStop_Vel);
 //第二步、设置插补速度曲线S段参数值
 ret=smc_set_vector_s_profile( MyCardNo,MyCrd,MySmode,MySpara );
//第三步、设置前瞻参数
ret=smc_conti_set_lookahead_mode(MyCardNo,MyCrd,mode,LookaheadSegment ,PathError,LookaheadAcc);
 //第四步、打开连续插补
 ret=smc_conti_open_list(MyCardNo,MyCrd,MyaxisNum,AxisArray);
 //第五步、开始连续插补
 ret=smc_conti_start_list (MyCardNo,MyCrd);
 //第六步、使能 Blend 功能
 ret=smc_conti_set_blend(MyCardNo,MyCrd,enable);
 //第七步、添加直线插补段
  ret=smc_conti_line_unit(MyCardNo,MyCrd,MyaxisNum,AxisArray,Dist,Myposi_mode,0);
 //第八步、相对于轨迹段起点,滞后 100unit，输出口 1 输出低电平，低电平持续时间为 0.5s
 ret=smc_conti_delay_outbit_to_start(MyCardNo,MyCrd,MyBitno,MyLevel,MyDelayVal,MyDelayMode,MyRevTime);     //
 //第十步、添加圆弧插补段
```

```
 ret=smc_conti_arc_move_center_unit(MyCardNo,MyCrd,MyaxisNum,AxisArray,Dist1,cen,
dir,cic,Myposi_mode,0);
//第九步、等待延时输入 IO
ret=smc_conti_wait_input(MyCardNo,MyCrd,bitno,sta,TimeOut,0);
//第十一步、关闭连续插补缓冲区
ret= smc_conti_close_list (MyCardNo,MyCrd);
}
```

### 2.2.5 手轮运动

雷赛控制器提供了一个强大的手轮脉冲控制功能，虽然接口只有一个，但是通过灵活配置，就能达到用户理想的手轮效果。控制器手轮功能提供了两种模式，一种默认模式，一种高级模式。

在默认模式下，手轮功能与手轮上的图标说明一一对应，例如 x,y,z,4,5,6 轴的单轴控制，3 个速率档位可选：1,10,100 倍的控制。

在高级模式下，轴档位不在是固定的单轴控制，而是可以通过函数来设定每一个轴档位关联的轴组，速率档位也不再是固定的 1,10,100 倍。举例说明：如果有三种情况需要用到手轮，1：x，y 轴同时运动，2：z，u 轴同时运动，3：4 轴同时运动。 那么就可以设定为 x 轴手轮信号关联 x,y 轴，同时可以单独设置 3 个档位的速率，不与其他轴手轮档位(y，z，u)的共用。y 轴手轮信号关联 z,u 轴，速率也可单独设置。z 轴则关联 x,y,z,u4 个轴，速率也单独设置。这样，当手轮轴选打到 x 轴开关时，可以同时控制 x,y 轴运动，打到 z 轴开关时，则可以控制 4 个轴同时运动。

不同的模式可以满足不同的使用情况，使得手轮功能大大增强，满足使用者的多种需求。手轮如下图 2.18.

图2.18 手持式手摇脉冲发生器外形

手轮运动相关函数：

| 名称 | 功能 | 参考 |
|---|---|---|
| smc_handwheel_set_axislist | 设置同一轴选档位下具体运动轴 | 3.17 节 |
| smc_handwheel_get_axislist | 读取同一轴选档位下具体运动轴 | |

| | | |
|---|---|---|
| smc_handwheel_set_ratiolist | 设置同一轴选下手轮倍率档位 | |
| smc_handwheel_get_ratiolist | 读取同一轴选下手轮倍率档位 | |
| smc_handwheel_set_mode | 设置手轮运动模式，硬件或软件 | |
| smc_handwheel_get_mode | 读取手轮运动模式，硬件或软件 | |
| smc_handwheel_set_index | 设置手轮运动轴选、倍率档位 | |
| smc_handwheel_get_index | 读取手轮运动轴选、倍率档位 | |
| smc_handwheel_move | 启动手轮运动 | |
| smc_handwheel_stop | 停止手轮运动 | |

例 1：手轮运动。运动轴选档位 0、倍选档位 0，运动轴 0、1、2 轴以 1 倍速度运行。5s 后换轴选档位 1、倍选档位 2，运动轴 4、5 以 100 倍速度运行。

```
int main(int argc, char* argv[])
{
/**********************变量定义*****************************/
    short ret = 0;           //错误返回
    short MyCardNo=0;
    WORD Myposi_mode = 1;      //0:相对模式，1：绝对模式
    WORD MyCrd = 0;             //参与插补运动的坐标系
    double MyMin_Vel = 0;          //插补起始速度
    double MyMax_Vel = 3000;       //插补运动最大矢量速度 3000unit/s
    double MyTacc = 0.2;           //插补运动加速时间 0.2s
    double MyTdec = 0.1;          //插补运动减速时间 0.1s
    double MyStop_Vel = 0;         //插补停止速度
   WORD MySmode = 0;            //保留参数，固定值为 0
   double  MySpara = 0.0;         //平滑时间为 0.05s
  WORD AxisSelIndex=0;     //手轮轴选档位 0
  WORD RatioSelIndex=0;    //倍选档位 0,1 倍速度
  WORD AxisSelIndex_1=1;     //手轮轴选档位 1
  WORD RatioSelIndex_1=2;    //倍选档位，100 倍速度
  WORD  InMode = 1;      //输入脉冲模式，0：脉冲+方向，1：AB 相脉冲
   WORD IfHardEnable = 0;    // 运行模式，0：软件控制，1：硬件控制
  WORD  AxisNum = 3;        //手轮运动轴为 3 轴
   WORD AxisArray[3];    //轴线档位 0 的运动轴为 3 轴
   AxisArray[0] = 0;     //运动轴 0
    AxisArray[1] = 1;    //运动轴 1
    AxisArray[2] = 2;    //运动轴 2
```

```
    WORD  AxisNum_1 = 2;          //轴线档位 1 的运动轴为 2 轴
     WORD AxisArray_1 [2];    //手轮运动轴列表
     AxisArray_1 [0] = 4;     //运动轴 4
      AxisArray_1 [1] = 5;    //运动轴 5
      WORD StartRatioIndex = 0;  //倍选起始值
       WORD    RatioSelNum = 3;     //’需设定倍率值 3
      double RatioList[3];    //’设定倍率列表
      RatioList[0] = 1;
      RatioList[1] = 10;
      RatioList[2] = 100;

  /**********************函数调用执行**************************/
  //第一步、设置插补运动速度曲线
  ret=smc_set_vector_profile_unit(MyCardNo,MyCrd,MyMin_Vel,MyMax_Vel,MyTacc,MyTde
  c,MyStop_Vel);
  //第二步、设置插补速度曲线的平滑时间
  ret=smc_set_vector_s_profile( MyCardNo,MyCrd,MySmode,MySpara );
  //第三步、设置轴选倍选档位0
  ret=smc_handwheel_set_index(MyCardNo,AxisSelIndex,RatioSelIndex);
  //第四步、设置手轮运动模式，硬件或软件模式
  ret=smc_handwheel_set_mode( MyCardNo,  InMode,  IfHardEnable );

  //第五步、设置手轮轴选档位及运动轴列表
  ret=smc_handwheel_set_axislist(MyCardNo,AxisSelIndex,AxisNum,AxisArray;
  ret=smc_handwheel_set_axislist(MyCardNo,AxisSelIndex_1,AxisNum_1,AxisArray_1);
//设置轴选档位1的具体运动轴

  //第六步、设置手轮倍率档位及对应轴的倍率值
  ret=smc_handwheel_set_ratiolist(MyCardNo,AxisSelIndex,StartRatioIndex,
  RatioSelNum, RatioList);
  ret=smc_handwheel_set_ratiolist(MyCardNo,AxisSelIndex_1,StartRatioIndex,
  RatioSelNum, RatioList);  //设置轴选档位 1 下，对应手轮倍率

  //第七步、启动手轮运动
  ret=smc_handwheel_move( MyCardNo,  0 );
  //第八步、5s 后手轮运动换成档位 1
 Sleep(5000);
 ret=smc_handwheel_set_index(MyCardNo,AxisSelIndex_1,RatioSelIndex_1);}
```

## 2.2.6 电子凸轮

雷赛控制器支持单轴或者多轴按凸轮关系表跟随主轴的指令位置或者编码器反馈位置运动。

电子凸轮数据表，采用相对位置模式，Mpos1 和 Spos1 固定为 0，主轴位置必须朝一个方向递增或者递减。

| 主轴位置 | Mpos1 | Mpos2 | ..... | Mposn-1 | MposN |
|---|---|---|---|---|---|
| 从轴位置 | Spos1 | Spos2 | ..... | Sposn-1 | SposN |

电子凸轮控制流程

1、下载凸轮数据表

2、启动从轴凸轮运动，从轴进入凸轮运动模式并为运动状态，当前点即为跟随起始点，最大跟随距离为凸轮表中主轴最大距离，主轴停止状态下方可启动从轴凸轮运动。

3、启动主轴运动，从轴开始跟随，超出跟随范围后从轴不在跟随。

4、停止从轴凸轮运动，从轴退出凸轮跟随模式。

相关指令：

| 名称 | 功能 | 参考 |
|---|---|---|
| smc_cam_table_unit | 设置凸轮表 | 3.17 节 |
| smc_cam_move | 启动电子凸轮运动 | |

例程：

```
WORD ConnectNo, MasterAxisNo,SlaveAxisNo,Count;
double MasterPos[100],SlavePos[100];
WORD SrcMode,MasterTargetPos;
ConnectNo = 0;          //链接号
MasterAxisNo = 0;       //主轴轴号
SlaveAxisNo = 1;        //从轴轴号
SrcMode = 0;            //主轴位置模式：0-指令位置，1-反馈位置
Count = 11;             //数据个数
//填充电子凸轮数据表，最大 1000 组，第一组数据保留，必须固定为（0，0）
for(i=0 ;i<Count;i++)
```

```
{
    MasterPos[i] = i*(-1000);
    SlavePos[i] = (-100)*(i % 2);
}
//添加电子凸轮表
iret =
smc_cam_table_unit(ConnectNo,MasterAxisNo,SlaveAxisNo,Count,MasterPos,SlavePos,Sr
cMode);
//启动从轴电子凸轮运动
iret = smc_cam_move(ConnectNo,SlaveAxisNo);
//控制主轴运动
iret = smc_pmove_unit(ConnectNo,MasterAxisNo,MasterPos[Count-1],0);
```

## 2.3 通用 IO 功能

### 2.3.1 通用 IO 控制

用户可以使用雷赛控制器上的数字 I/O 口用于检测开关信号、传感器信号等输入信号，或者控制继电器、电磁阀等输出设备。

雷赛控制器支持 I/O 延时翻转功能。该函数执行后，首先输出一个与当前电平相反的信号，延时设置的时间后，再自动翻转一次电平。

雷赛控制器支持输入 IO 计数功能。该功能允许用户设置输入 IO 作为计数器使用。

通用 IO 控制相关函数：

| 名称 | 功能 | 参考 |
|---|---|---|
| smc_read_inbit | 读取某一位输入口的电平状态 | 3.15 节 |
| smc_write_outbit | 设置某个端口输出电平 | |
| smc_read_outbit | 读取某一位输出口的电平状态 | |
| smc_read_inport | 读取全部输入口的电平状态 | |
| smc_read_outport | 读取全部输出口的电平状态 | |
| smc_write_outport | 设置全部输出口的电平状态 | |
| smc_reverse_outbit | IO 输出延时翻转 | |
| smc_set_io_count_mode | 设置 IO 计数模式 | |
| smc_get_io_count_mode | 读取 IO 计数模式 | |
| smc_set_io_count_value | 重置 IO 计数值 | |

| | | |
|---|---|---|
| smc_get_io_count_value | 读取 IO 计数值 | |

⚠注意：调用 smc_read_inbit（）函数可以读取某一位通用输入口的电平；调用 smc_read_inport（）函数可以一次性读入全部输入口的电平

### 2.3.2 虚拟 IO 映射

雷赛控制器支持虚拟 IO 映射功能。通过该功能函数的设置可以实现专用通用 IO 输入接口的滤波功能，并且可以通过专用函数读取该端口滤波后的电平状态。

虚拟 IO 映射相关函数：

| 名称 | 功能 | 参考 |
|---|---|---|
| smc_set_io_map_virtual | 设置虚拟 IO 映射关系 | 3.27 节 |
| smc_get_io_map_virtual | 读取虚拟 IO 映射关系设置 | |
| smc_read_inbit_virtual | 读取滤波后的虚拟IO口电平状态 | |

例 1：读取普通 IO 状态

```
int main(int argc, char* argv[])
{
/**********************变量定义*****************************/
      short ret = 0;            //错误返回
      WORD MyCardNo = 0;        //连接号
      WORD MyInport = 0;        //输入端口组号
      short vale;
      WORD bitno=0;    //IO 口 0
      WORD stat=0;    //低电平状态 0
      WORD portno=0;          //io 组号
      DWORD port_value=3;  //写入值 2，转换成 2 进制 11
      WORD mode=2;// IO 计数模式，0：禁用，1：上升沿计数，2：下降沿计数
      bitno=0;        //输出端口 0
      double filter_time=0;  //滤波时间，单位：s
      DWORD  CountValue=0;  //计数值
      double reverse_time=1;      //翻转时间，单位：s

/**********************函数调用执行说明*************************/
      //1、读输入 IO 电平状态返回值
      long MyInportValue=smc_read_inport(MyCardNo,MyInport);
      printf("输入 IO 电平状态返回值= %d\n ",MyInportValue);
```

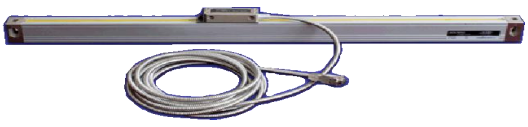
```
    //2、设置单个端口输出电平
    ret =smc_write_outbit(MyCardNo,bitno,stat);
    //3、读取单个端口电平状态
    vale=smc_read_outbit(MyCardNo,bitno);
    printf("读取端口电平状态= %d\n ",vale); //打印单个端口电平状态值
    //4、设置所有输出口电平状态
    smc_write_outport(MyCardNo,  portno, port_value);
    //5、读取所有输出口电平状态
    vale=smc_read_outport(MyCardNo,  portno);
    printf("所有输出口电平状态= %d\n ",vale); //打印所有输出口电平值
    //6、设置计数模式、读取计数值
    ret=smc_set_io_count_mode(MyCardNo,bitno,mode,filter_time);
    ret=smc_get_io_count_value(MyCardNo,bitno,& CountValue);
    printf("所有输出口电平状态= %d\n ", CountValue);  //打印计数值
    //7、设置电平翻转
    ret=smc_reverse_outbit(MyCardNo,bitno, reverse_time); //电平翻转
}
```

## 2.4 特殊 IO 功能

### 2.4.1 编码器检测

雷赛控制器每轴都有一个编码器输入接口用于检测平台的位移或电机的转角。编码器有 EA、EB、EZ 三个信号，脉冲计数信号由 EA 和 EB 端口输入；它可以接收两种类型的脉冲信号：正负脉冲输入或 A/B 相正交信号；EZ 信号是编码器零位信号。编码器外形如图 2.19 所示。

旋转编码器



光栅尺

图2.19　编码器外形

采用探针和编码器配合使用，雷赛控制器通过位置触发功能，可完成对工件的位置检测工作，如图 2.20 所示。即：当探针接触到工件时，产生一个触发信号，控制器接受该信号后，立即将编码器当前位置记录下来；通过记录工件的一系列数据，然后再通过软件处理，即可获得该工件的外形尺寸。

图2.20 被测工件与探针

编码器相关函数：

| 名称 | 功能 | 参考 |
|---|---|---|
| smc_set_counter_inmode | 设置编码器的计数方式 | 3.18 节 |
| smc_get_counter_inmode | 读取编码器的计数方式 | |
| smc_set_encoder_unit | 设置指定轴编码器脉冲计数值 | |
| smc_get_encoder_unit | 读取指定轴编码器脉冲计数值 | |
| smc_set_ez_mode | 设置指定轴的 EZ 信号电平 | |
| smc_get_ez_mode | 读取指定轴的 EZ 信号电平 | |
| smc_set_counter_reverse | 设置 AB 相计数值反相 | |
| smc_get_counter_reverse | 读取 AB 相计数值反相模式 | |

例程：编码器检测

```
int main(int argc, char* argv[])
{
/**********************变量定义*****************************/
       short ret = 0;           //错误返回
       short res = 0;
        WORD MyCardNo = 0;         // 连接号
        WORD Myaxis = 0;          //轴号
        WORD Mymode = 3;          //设置编码器的计数方式为 4 倍频，AB 相
        double Myencoder_value=0;
/**********************函数调用执行*************************/
   //第一步、设置 0 号轴的编码器计数方式
  ret=smc_set_counter_inmode(MyCardNo,Myaxis,Mymode);
   //第二步、设置 0 轴的编码值为 100
   res=smc_set_encoder_unit(MyCardNo,Myaxis,100);
   //第三步、读轴 0 轴的编码值
   res=smc_get_encoder_unit(MyCardNo,Myaxis,&Myencoder_value);
}
```

## 2.4.2 位置锁存

雷赛控制器支持多种高速位置锁存功能，可实现精确定位功能，其包括单次锁存、连续锁存。单次锁存在锁存信号有效后可锁存一次，再次锁存需复位锁存标志。

连续锁存可实现对多个位置依次进行锁存。在每次锁存后，不需要再复位锁存标志。锁存次数超过 1000 次，剔除最先的触发位置保存最新的触发位置。连续锁存间隔时间需大于 1ms。

高速位置锁存相关函数：

| 名称 | 功能 | 参考 |
|---|---|---|
| smc_set_ltc_mode | 设置指定轴的 LTC 信号 | 3.19 节 |
| smc_get_ltc_mode | 读取指定轴的 LTC 信号设置 | |
| smc_set_latch_mode | 设置锁存方式、单次连续锁存 | |
| smc_get_latch_mode | 读取锁存方式 | |
| smc_get_latch_value_unit | 从控制器内读取编码器锁存器的值 | |
| smc_get_latch_flag | 从控制器内读取指定轴的锁存次数 | |
| smc_reset_latch_flag | 复位指定控制器的锁存器的标志位 | |

原点锁存相关函数：

| 名称 | 功能 | 参考 |
|---|---|---|
| smc_set_homelatch_mode | 设置原点锁存模式 | 3.20 节 |
| smc_get_homelatch_mode | 读取原点锁存模式设置 | |
| smc_reset_homelatch_flag | 清除原点锁存标志 | |
| smc_get_homelatch_flag | 读取原点锁存标志 | |
| smc_get_homelatch_value_unit | 读取原点锁存值 | |

例 1：高速多次位置锁存

```
int main(int argc, char* argv[])
{
/**********************变量定义*****************************/
      WORD MyCardNo = 0;              // 连接号
      WORD Myaxis = 0;               //轴号
      short ret;       //返回错误码
      WORD Myltc_logic = 0;              //设置 LTC 触发方式为下降沿触发
      WORD Myltc_mode = 0;              //保留值 0
```

```
    double Myfilter = 0;              //保留参数
    WORD ltc_mode=2;                 //多次锁存模式 0 单次锁存，2 连续锁存
    WORD Mylatch_source = 0;          //设置锁存源为指令位置
    WORD triger=0;                   // 触发通道，固定值 0
     int i=0;

/**********************函数调用执行**************************/
   //第一步、设置 0 号轴的 LTC 信号，触发方式为下降沿触发
  ret=smc_set_ltc_mode( MyCardNo, Myaxis, Myltc_logic, Myltc_mode, Myfilter);
  //第二步、设置 0 号轴的锁存参数
  ret=smc_set_latch_mode(MyCardNo,Myaxis,ltc_mode,Mylatch_source,triger);
   //第三步、复位 0 号轴的锁存状态
   ret=smc_reset_latch_flag(MyCardNo,Myaxis);
   //第四步、设置速度参数
   ret=smc_set_profile_unit(MyCardNo,Myaxis,0,5000,0.1,0.1,0);
   //第五步、启动定长运动，等待运动停止
   ret=smc_pmove_unit(MyCardNo,Myaxis,10000,0);
   while(smc_check_done(MyCardNo, Myaxis)==0);  //等待运动停止
   //第六步、读取锁存个数
   int My_latch_flag =smc_get_latch_flag(MyCardNo,Myaxis);
   printf("锁存个数= %d\n ",My_latch_flag); //打印已锁存个数
   //第七步、读取锁存值
  if(My_latch_flag>0)
  {
    double *My_latch_Value=new double[My_latch_flag];
    for(i=0;i<=My_latch_flag;i++)
     {
          smc_get_latch_value_unit(MyCardNo,Myaxis,&My_latch_Value[i]);
          printf("My_latch_Value= %f\n ",My_latch_Value[i]);   //打印锁存值
      }
   }
}
```

例 2：原点锁存，当运动碰到原点信号有效，原点锁存标志有效

```
int main(int argc, char* argv[])
{
/**********************变量定义**************************/
     short ret = 0;           //错误返回
```

```
    WORD MyCardNo = 0;            // 连接号
    WORD Myaxis = 0;              //轴号
    WORD enable=1;               //原点锁存使能，0：禁止，1：允许
    WORD logic = 0;              //设置LTC触发方式为下降沿触发
    WORD source=0;               //设置锁存源,0为指令位置1为编码器位置
    double pos=0;               //读取原点锁存值
/**********************函数调用执行**************************/
   //第一步、设置锁存模式
  ret=smc_set_homelatch_mode(MyCardNo, Myaxis, enable,logic,source);
  //第二步、复位0号轴的锁存状态
  ret=smc_reset_homelatch_flag(MyCardNo,Myaxis);
  //第三步、设置速度参数
  ret=smc_set_profile_unit(MyCardNo,Myaxis,0,500,0.1,0.1,0);
  //第四步、启动定长运动
  ret=smc_pmove_unit(MyCardNo,Myaxis,10000,0);
  while(smc_check_done(MyCardNo, Myaxis)==0);    //等待运动停止
  //第五步、判断锁存标志，读取锁存值
  if ((smc_get_homelatch_flag(MyCardNo,Myaxis))>0)
   {
       smc_get_homelatch_value_unit(MyCardNo,Myaxis,&pos);
       printf("原点锁存值= %f\n ",pos);
   }
}
```

### 2.4.3 位置比较输出

雷赛控制器提供了位置触发输出信号的函数，包括单轴低速位置比较、单轴高速位置比较和一组二维低速位置比较。当电机运动到预先设置的位置时，自动触发特定的输出口。该功能在轨迹运动中用于控制点胶阀的开关、触发照相机快门等动作十分方便。一维低速位置比较相关函数如下表所示。

雷赛控制器提供了位置比较功能，位置比较的一般步骤是：

1、配置比较器；

2、清除比较器；

3、添加/更新比较位置点；

4、开始运动并查看比较状态。

一维低速位置比较相关函数：

| 名称 | 功能 | 参考 |
| --- | --- | --- |
| smc_compare_set_config | 设置一维位置比较器 | 3.22 节 |
| smc_compare_get_config | 读取一维位置比较器设置 | |
| smc_compare_clear_points | 清除一维位置比较点 | |
| smc_compare_add_point_unit | 添加一维位置比较点 | |
| smc_compare_get_current_point_unit | 读取当前一维比较点位置 | |
| smc_compare_get_points_runned | 查询已经比较过的一维比较点个数 | |
| smc_compare_get_points_remained | 查询可以加入的一维比较点个数 | |

⚠注意：（1）每轴的位置比较都是独立进行的。

（2）执行位置比较时，每个比较点的触发是按照添加的比较点顺序执行的，即如果有一个比较点没有被触发比较动作，那么后面的比较点是不会起作用的。

二维低速位置比较相关函数：

| 名称 | 功能 | 参考 |
| --- | --- | --- |
| smc_compare_set_config_extern | 设置二维位置比较器 | 3.22 节 |
| smc_compare_get_config_extern | 回读二维比较器参数 | |
| smc_compare_clear_points_extern | 清除二维位置比较点 | |
| smc_compare_add_point_extern_unit | 添加二维位置比较点 | |
| smc_compare_get_current_point_extern_unit | 读取当前二维位置比较点位置 | |
| smc_compare_get_points_runned_extern | 查询已经比较过的二维比较点个数 | |
| smc_compare_get_points_remained_extern | 查询可以加入的二维比较点个数 | |

一维高速位置比较相关函数：

| 名称 | 功能 | 参考 |
| --- | --- | --- |

| | | |
|---|---|---|
| smc_hcmp_set_mode | 设置高速比较模式 | 3.23 节 |
| smc_hcmp_get_mode | 读取高速比较模式设置 | |
| smc_hcmp_set_config | 配置高速比较器 | |
| smc_hcmp_get_config | 读取高速比较器配置 | |
| smc_hcmp_set_liner_unit | 设置高速比较线性模式参数 | |
| smc_hcmp_get_liner_unit | 读取高速比较线性模式参数 | |
| smc_hcmp_clear_points | 清除高速位置比较点 | |
| smc_hcmp_add_point_unit | 添加/更新高速比较位置 | |
| smc_hcmp_get_current_state_unit | 读取高速比较状态 | |
| smc_write_cmp_pin | 控制指定 CMP 端口的输出 | |
| smc_read_cmp_pin | 读取指定 CMP 端口的电平 | |

⚠注意：（1）每个比较器的位置比较都是独立进行的。如不再使用位置比较下比较口 OUT16 和 OUT17 时，我们需启用指令 SMCHcmpSetMode(Myhcmp,0)下禁止模式，释放比较端口，否则 OUT16、OUT17 单独作为输出口时，可能无法使用。

（2）在队列及线性比较模式中，执行位置比较时，每个比较点的触发是按照添加的比较点顺序执行的，即如果有一个比较点没有被触发比较动作，那么后面的比较点是不会被触发的。

例 1：一维低速位置比较

```
int main(int argc, char* argv[])
{
/**********************变量定义*****************************/
      short ret = 0;            //错误返回
      WORD MyCardNo = 0;          // 连接号
      WORD Myaxis = 0;            //轴号
      WORD enable=1;              //原点锁存使能，0：禁止，1：允许
      WORD source=0;               //设置锁存源,0 为指令位置 1 为编码器位
      double Mypos = 100;         //设置比较位置
      WORD Mydir = 1;             //比较模式，0：小于等于，1：大于等于
      WORD Myaction = 3;          //设置触发功能为 IO 电平取反
      WORD Myactpara = 0;         //设置输出 IO 端口 0 触发功能
      double pos=0;  //比较值
      long pointNum=0;  //比较过点个数
      long pointNum1=0;   //剩余比较点数量

/**********************函数调用执行*************************/
```

```
    //第一步、清除比较点
     ret=smc_compare_clear_points(MyCardNo,Myaxis);
     //第二步、配置比较器
     ret=smc_compare_set_config(MyCardNo,Myaxis, enable, source);
     //第三步、配置比较点参数
     ret=smc_compare_add_point_unit(MyCardNo,Myaxis,Mypos,Mydir,Myaction,Myactpa
ra);
     //第四步、设置运动速度参数
     ret=smc_set_profile_unit(MyCardNo,Myaxis,0,500,0.1,0.1,0);
     //第五步、启动运动
     ret=smc_pmove_unit(MyCardNo,Myaxis,1000,0);
     while(smc_check_done(MyCardNo, Myaxis)==0);    //等待运动停止
    //第六步、读取当前比较点位置值
     ret=smc_compare_get_current_point_unit(MyCardNo,Myaxis,&pos);
     printf("读取当前比较点= %f\n ",pos);  //打印比较点值
     //第七步、查询已比较过的点
     ret=smc_compare_get_points_runned(MyCardNo,Myaxis,&pointNum);
     printf("已经比较过的点= %d\n ",pointNum);
     //第八步、查询剩余的比较点数量
    ret=smc_compare_get_points_remained(MyCardNo,Myaxis,&pointNum1);
    printf("可以加入的比较点数量= %d\n ",pointNum1);
}
```

例 2：一维高速位置比较，队列模式

```
int main(int argc, char* argv[])
{
/**********************变量定义*****************************/
    short ret = 0;           //错误返回
    WORD MyCardNo = 0;         // 连接号
    WORD Myaxis = 0;          //轴号
    WORD hcmp=0;             //高速比较器
    WORD source=0;           //设置比较源,0 为指令位置 1 为编码器位置
    WORD Logic=0;            //低电平有效
    long MyTime=0;           //脉冲宽度，单位：us
    double cmp_pos=1000;    //添加比较点 0
    double cmp_pos1=2000;    //添加比较点 1
    double cmp_pos2=5000;   //添加比较点 2
    long remained_points=0;      //返回可添加比较点数
```

```
    double current_point=0;     //返回当前比较点位置，单位：unit
    long runned_points=0;        //返回已比较点数
/**********************函数调用执行**************************/
    //第一步、设置比较器参数
    ret=smc_hcmp_set_config(MyCardNo,hcmp,Myaxis, source,Logic,MyTime);
    //第二步、设置比较器模式
    WORD cmp_mode=4; //0：禁止（默认值,1 等于,2 小于,3 大于,4 队列,5 线性
    ret=smc_hcmp_set_mode(MyCardNo, hcmp,  cmp_mode);
    //第三步、清除已添加的所有高速位置比较点
    ret=smc_hcmp_clear_points(MyCardNo,hcmp);
    //第四步、添加位置比较点值
    ret=smc_hcmp_add_point_unit(MyCardNo, hcmp,cmp_pos);
    ret=smc_hcmp_add_point_unit(MyCardNo, hcmp,cmp_pos1);
    ret=smc_hcmp_add_point_unit(MyCardNo, hcmp,cmp_pos2);
    //第五步、设置速度参数
    ret=smc_set_profile_unit(MyCardNo,Myaxis,0,500,0.1,0.1,0);
    //第六步、启动运动
    ret=smc_pmove_unit(MyCardNo,Myaxis,1000,0);
    while(smc_check_done(MyCardNo, Myaxis)==0);    //等待运动停止
    //第七步、读取位置比较状态
    ret=smc_hcmp_get_current_state_unit(MyCardNo,hcmp,&remained_points,&current_p
    oint,&runned_points);
    printf("读取当前比较点= %f\n ",current_point);
    printf("已经比较过的点= %d\n ",runned_points);
    printf("可以加入的比较点数量= %d\n ",remained_points);
}
```

## 2.4.4 PWM 输出

雷赛控制器提供了 PWM 输出功能。如图 2.21 所示，雷赛控制器输出的 PWM 波形的周期为 t2（频率即为 1/t2），占空比为 t1/t2，幅值为 V1 = 5V。

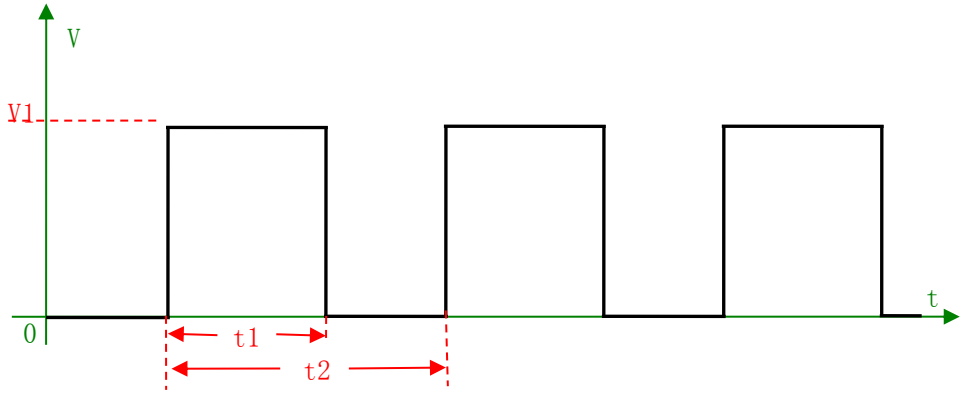


图 2.21　PWM 输出示意图

PWM（脉宽调制）基本原理就是对逆变电路开关器件的通断进行控制，使输出端得到一系列幅值相等的脉冲，用这些脉冲来代替正弦波或所需要的波形。

PWM 功能相关函数：

| 名称 | 功能 | 参考 |
| --- | --- | --- |
| smc_set_pwm_output | 设置 PWM 立即输出 | 3.14 节 |
| smc_get_pwm_output | 读取 PWM 当前输出状态 | |

例程：PWM 输出功能

```
int main(int argc, char* argv[])
{
/**********************变量定义*****************************/
    short ret = 0;            //错误返回
    WORD MyCardNo = 0;              // 连接号
    WORD MyPwmNo = 0;               //PWM 输出通道为 0 通道
    double MyfDuty = 0.5;              //PWM 输出占空比
    double MyfFre = 10000;            //PWM 输出频率
   double MyfDuty1;          //PWM 输出占空比
    double MyfFre1;          //PWM 输出频率

/**********************函数调用执行*************************/
    //第一步、设置 PWM 输出参数
    ret=smc_set_pwm_output(MyCardNo,MyPwmNo,MyfDuty,MyfFre);
    //第一步、回读且打印 PWM 输出参数
   ret=smc_get_pwm_output(MyCardNo,MyPwmNo,&MyfDuty1,&MyfFre1);
     printf("PWM输出占空比= %f\n ",MyfDuty1);
    printf("PWM输出频率= %f\n ",MyfFre1);
}
```

## 2.4.5 伺服专用功能

设备中有交流伺服电机时，使用雷赛控制器上的伺服电机控制信号 SEVON、RDY、ALM、INP 和 ERC 就显得十分方便。

SEVON 是控制器输出给伺服电机驱动器的控制信号，当 SEVON 信号为无效状态时，伺服驱动器不使能，电机处于自由状态；当 SEVON 信号有效时，伺服驱动器使能，电机锁紧；等待指令脉冲信号。

ALM 信号是从伺服电机驱动器发给控制器的状态信号，用来报告伺服驱动器或电机出错。控制器接收到 ALM 信号时，将立即停止发出脉冲，该过程是一个硬件处理过程。

INP 信号是从伺服电机驱动器发给控制器的状态信号，告知运动控制器伺服电机已经停止。

伺服电机驱动器通常有一个位置偏差计数器，记录指令脉冲和位置反馈脉冲之间的偏差。伺服电机驱动器将控制电机运动使位置偏差趋于 0，但是电机实际位置总是滞后于指令脉冲。所以当运动控制器的指令脉冲发送完毕时，伺服电机并没有立即停止，而是继续运动，如图 2.22 所示，直到位置偏差趋于 0；这时驱动器将发出一个 INP 信号。

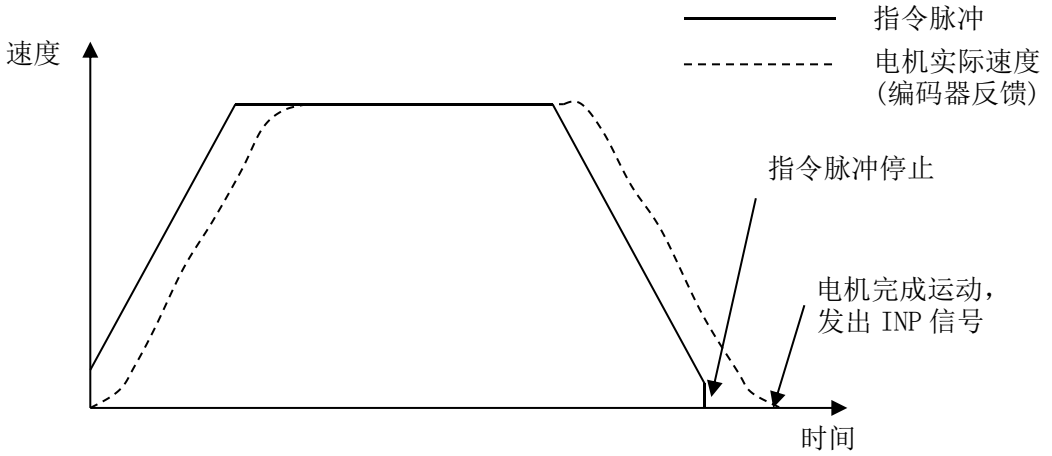



图 2.22　伺服定位完成时的 INP 信号

ERC 信号是控制器输出给伺服电机驱动器的控制信号。

伺服驱动器依赖电机目标位置（即要求电机达到的位置）和当前位置（即电机已经到达的位置）之间的误差来驱动电机运动，若这个误差为零电机将停止运动。当伺服驱动器接收到运动控制器发出的 ERC 信号后，会立即清除误差并停止电机运转。

专用 IO 相关函数：

| 名称 | 功能 | 参考 |
|---|---|---|
| smc_set_alm_mode | 设置指定轴的 ALM 信号 | 3.16 节 |
| smc_get_alm_mode | 读取指定轴的 ALM 信号设置 | |
| smc_read_alarm_pin | 读取指定轴的 ALARM 端口电平 | |
| smc_set_inp_mode | 设置指定轴的 INP 信号 | |

| | | |
|---|---|---|
| smc_get_inp_mode | 读取指定轴的 INP 信号设置 | |
| smc_read_inp_pin | 读取指定轴的 INP 端口电平 | |
| smc_read_rdy_pin | 读取指定轴的 RDY 端口的电平状态 | |
| smc_write_sevon_pin | 控制指定轴的伺服使能端口的输出 | |
| smc_read_sevon_pin | 读取指定轴的伺服使能端口的电平 | |
| smc_write_erc_pin | 控制指定轴的 ERC 信号输出 | |
| smc_read_erc_pin | 读取指定轴的 ERC 端口电平状态 | |
| smc_read_org_pin | 读取指定轴的 ORG 端口电平 | |
| smc_read_elp_pin | 读取指定轴的 ELP 端口电平 | |
| smc_read_eln_pin | 读取指定轴的 ELN 端口电平 | |
| smc_read_emg_pin | 读取指定轴的 EMG 端口电平 | |

⚠注意：（1）若控制器本身没有相应硬件接口，使用前需 IO 映射，如 SMC606 控制器的 INP 信号，使用前需要映射。

（2)某些函数轴号设为 255 时,所有轴类似参数都被设置。如函数 smc_set_el_mode 、smc_set_alarm_mode、smc_set_inp_mode、smc_set_home_pin_logic、smc_set_ez_mode、smc_set_io_dstp_mode。

## 2.4.6 限位功能

雷赛控制器提供了硬件和软件两种限位功能。用户可根据设备的硬件限位开关，来设置限位开关工作的有效电平，或直接以软件来设置软件限位位置值。

限位功能相关函数：

| 名称 | 功能 | 参考 |
|---|---|---|
| smc_set_el_mode | 设置限位开关信号 | 3.24 节 |
| smc_get_el_mode | 读取限位开关信号设置 | |
| smc_set_softlimit_unit | 设置软限位参数 | |
| smc_get_softlimit_unit | 读取软限位参数 | |

例程：设置限位开关参数

```
int main(int argc, char* argv[])
{
/**********************变量定义*****************************/
```

```
    WORD MyCardNo = 0;          //连接号
    short ret = 0;             //错误返回
    WORD Myaxis = 0;            //轴号
    WORD Myel_enable = 1;       //正负限位使能
    WORD Myel_logic = 0;        //正负限位低电平有效
    WORD Myel_mode = 0;         //正负限位停止方式为立即停止
    Myel_enable=1;          //使能状态，0：禁止，1：允许
    WORD source_sel=0; //计数器选择，0：指令位置计数器 1：编码器计数器
    WORD SL_action=1;     //限位停止方式，0：立即停止，1：减速停止
    double P_limit=1000;    //正限位位置，单位：unit
    double N_limit=-1000;
/**********************函数调用执行**************************/
    //第一步、设置 0 号轴硬件限位信号
   ret=smc_set_el_mode(MyCardNo,Myaxis,Myel_enable,Myel_logic,Myel_mode);
    //第二步、回读 0 号轴硬件限位信号参数
   ret=smc_get_el_mode(MyCardNo,Myaxis,&Myel_enable,&Myel_logic,&Myel_mode);
   printf("读取硬件限位设置参数= %d %d %d\n ", Myel_enable,Myel_logic,Myel_mode);
   //第三步、设置软限位参数
  ret=smc_set_softlimit_unit(MyCardNo,Myaxis,Myel_enable,    source_sel,SL_action,
     N_limit, P_limit);
  //第四步、回读软限位参数
  ret=smc_get_softlimit_unit(MyCardNo,Myaxis,&Myel_enable,&source_sel,&SL_action,
&N_limit, &P_limit);
  printf("软限位位置= %f %f\n ",N_limit,P_limit);  //打印软限位设定值
}
```

## 2.4.7 急停功能

雷赛控制器提供了运动急停功能，对应硬件接口电路进行接线，然后调用急停开关设置函数 smc_set_emg_mode 进行设置。

紧急停止相关函数：

| 名称 | 功能 | 参考 |
|---|---|---|
| smc_set_emg_mode | 设置 EMG 急停信号 | 3.25 节 |
| smc_get_emg_mode | 读取 EMG 急停信号设置 | |

⚠注意：若控制器本身没有急停硬件接口，使用前需 IO 映射，如 SMC606 控制器。

例程：设置通用输入口 0 接口为轴 0 的急停信号的映射入口，低电平有效

```
int main(int argc, char* argv[])
{
/**********************变量定义*****************************/
    WORD MyCardNo = 0;          //连接号
    WORD Myaxis = 0;            //轴号
   short ret = 0;             //错误返回
    WORD Myenable = 1;          //急停信号使能
    WORD Mylogic = 0;           //急停信号低电平有效

/**********************函数调用执行*************************/
    //第一步、设置轴 IO 映射，将通用输入 0 作为各轴的急停信号
    ret=smc_set_axis_io_map(MyCardNo, Myaxis, 3, 6, 0, 0);
    //第二步、设置 EMG 使能，低电平有效
    ret=smc_set_emg_mode(MyCardNo, Myaxis, Myenable,Mylogic);
    //第三步、回读 EMG 使能，低电平有效
    ret=smc_get_emg_mode(MyCardNo, Myaxis,&Myenable,&Mylogic);
    printf("急停信号参数,使能,有效电平= %d %d\n ",Myenable,Mylogic);
    //第四步、启动定长运动
    ret=smc_set_profile_unit(MyCardNo,Myaxis,0,1000,0.1,0.1,0);
    ret =smc_set_s_profile(MyCardNo,Myaxis,0,0.05);
    ret =smc_pmove_unit(MyCardNo,Myaxis,10000,1);
}
```

运动结果：运行程序后，当IN口0为低电平时，运动立即停止。

## 2.5 文件功能

雷赛控制器的文件功能，是控制器上传、下载文件（如 basic 文件、G 代码文件、参数文件等）。

文件相关函数：

| 名称 | 功能 | 参考 |
|---|---|---|
| smc_download_file | 下载本地文件到 FLASH | 3.29 节 |
| smc_download_memfile | 下载内存文件到 FLASH | |
| smc_upload_file | 上传 FLASH 文件到本地文件 | |
| smc_upload_memfile | 上传 FLASH 文件到内存文件 | |
| smc_download_file_to_ram | 下载本地文件到 RAM，掉电不保存 | |

| | | |
|---|---|---|
| smc_download_memfile_to_ram | 下载内存文件到 RAM，掉电不保存 | |
| smc_get_progress | 文件下载进度 | |

## 2.6 寄存器操作功能

雷赛控制器位寄存器BIT有10000位，寄存器REG有10000个字。它们的地址分配如下：

BIT00000 ~ BIT09999：客户自定义区

BIT10000 ~ BIT10299：数字输入端口电平。从 0 号端口开始按顺序排列,0-断开（高电平），1-导通（低电平）

BIT10300 ~ BIT10399：负向限位输入信号状态。0-正常，1-报警

BIT10400 ~ BIT10499：正向限位输入信号状态。0-正常，1-报警

BIT10500 ~ BIT10599：HOME 输入信号状态。0-正常，1-报警

BIT10600 ~ BIT10699：ALARM 输入信号状态。0-正常，1-报警

BIT11000 ~ BIT11399：数字输出端口电平。从 0 号端口开始按顺序排列,0-断开（高电平），1-导通（低电平）

REG00000 ~ REG09999：客户自定义区

REG10000 ~ REG10001：0 号轴当前位置，单位：pulse，类型：int32

REG10002 ~ REG10003：1 号轴当前位置，单位：pulse，类型：int32

REG10004 ~ REG10005：2 号轴当前位置，单位：pulse，类型：int32

REG10006 ~ REG10007：3 号轴当前位置，单位：pulse，类型：int32

REG10008 ~ REG10009：4 号轴当前位置，单位：pulse，类型：int32

REG10010 ~ REG10011：5 号轴当前位置，单位：pulse，类型：int32

REG10100 ~ REG10101：0 号轴当前位置，单位：unit，类型：float

REG10102 ~ REG10103：1 号轴当前位置，单位：unit，类型：float

REG10104 ~ REG10105：2 号轴当前位置，单位：unit，类型：float

REG10106 ~ REG10107：3 号轴当前位置，单位：unit，类型：float

REG10108 ~ REG10109：4 号轴当前位置，单位：unit，类型：float

REG10110 ~ REG10111：5 号轴当前位置，单位：unit，类型：float

REG10200 ~ REG10201：0 号轴当前速度，单位：pulse/s，类型：int32

REG10202 ~ REG10203：1 号轴当前速度，单位：pulse/s，类型：int32

REG10204 ~ REG10205：2 号轴当前速度，单位：pulse/s，类型：int32

REG10206 ~ REG10207：3 号轴当前速度，单位：pulse/s，类型：int32

REG10208 ~ REG10209：4 号轴当前速度，单位：pulse/s，类型：int32

REG10210 ~ REG10211：5 号轴当前速度，单位：pulse/s，类型：int32

REG10300 ~ REG10301：0 号轴当前速度，单位：unit/s，类型：float

REG10302 ~ REG10303：1 号轴当前速度，单位：unit/s，类型：float

REG10304 ~ REG10305：2 号轴当前速度，单位：unit/s，类型：float

REG10306 ~ REG10307：3 号轴当前速度，单位：unit/s，类型：float

REG10308 ~ REG10309：4 号轴当前速度，单位：unit/s，类型：float

REG10310 ~ REG10311：5 号轴当前速度，单位：unit/s，类型：float

REG10400 ~ REG10401：0 号编码器当前位置，单位：pulse，类型：int32

REG10402 ~ REG10403：1 号编码器当前位置，单位：pulse，类型：int32

REG10404 ~ REG10405：2 号编码器当前位置，单位：pulse，类型：int32

REG10406 ~ REG10407：3 号编码器当前位置，单位：pulse，类型：int32

REG10408 ~ REG10409：4 号编码器当前位置，单位：pulse，类型：int32

REG10410 ~ REG10411：5 号编码器当前位置，单位：pulse，类型：int32

REG10500 ~ REG10501：0 号编码器当前位置，单位：unit，类型：float

REG10502 ~ REG10503：1 号编码器当前位置，单位：unit，类型：float

REG10504 ~ REG10505：2 号编码器当前位置，单位：unit，类型：float

REG10506 ~ REG10507：3 号编码器当前位置，单位：unit，类型：float

REG10508 ~ REG10509：4 号编码器当前位置，单位：unit，类型：float

REG10510 ~ REG10511：5 号编码器当前位置，单位：unit，类型：float

寄存器操作相关函数：

| 名称 | 功能 | 参考 |
|---|---|---|
| smc_set_modbus_0x | 写位寄存器 | 3.30 节 |
| smc_get_modbus_0x | 读位寄存器 | |
| smc_set_modbus_4x | 写字寄存器 | |
| smc_get_modbus_4x | 读字寄存器 | |

例程：寄存器操作

```
int main(int argc, char* argv[])
{
/**********************变量定义*****************************/
      short ret = 0;           //错误返回
      WORD MyCardNo = 0;     // 连接号
      WORD start=0;        // 寄存器首地址
      WORD inum=3;        //寄存器个数
      char pdata=5;     //发送数据值
      char pdata1;
      WORD inum1=2;        //寄存器个数
      WORD pdata2[2];    //发送数据值
      pdata2[0]=1;
      pdata2[1]=2;
      WORD pdata3[2];

/**********************函数调用执行*************************/
      //第一步、写位寄存器，写寄存器 0、1、2 值分别为 1、0、1
      ret=smc_set_modbus_0x(MyCardNo,  start,  inum,& pdata );
      //第二步、读取位寄存器
      ret=smc_get_modbus_0x(MyCardNo, start, inum,& pdata1);
      //第三步、写字寄存器
      ret=smc_set_modbus_4x(MyCardNo,start, inum1,pdata2);
      //第四步、读字寄存器
      ret=smc_get_modbus_4x(MyCardNo, start, inum1, pdata3);
}
```

## 2.7 控制器组网

雷赛控制器通过交换机等设备可同时连接多个控制器，每个控制器可独立操作，如下图 2.22。

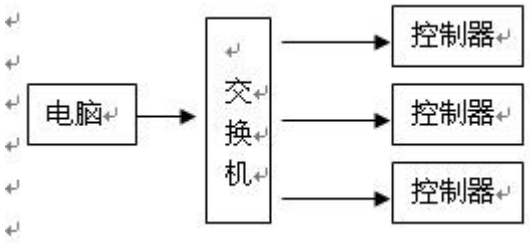



图 2.22 控制器组网示意图

如连接多个 SMC606 控制器，0 号控制器做点位运动，1 号控制器做直线插补运动。其实现步骤如下：

(1) 设置各控制器的 IP 地址在同一网段，但最后一位不同，如设定 0 号控制器 IP 为 192.168.5.11，设置 1 号控制器 IP 为 192.168.5.22

(2) 编写程序，连接 2 个控制器。

(3) 编辑指令可控制各控制器做不同的动作。

⚠注意：控制器连接前，各组网控制器的 IP 值（最后一位）要不相同。

例程：通过交换机，连接 2 台控制器。 0 号控制器做定长运动，运动完成后，对 1 号控制器做直线插补运动。

```
/**********************变量定义*****************************/
        WORD ConnectNo = 0; //0 控制器连接号，可选 0--7
        WORD ConnectNo2 = 1; //1 控制器连接号，可选 0--7
        WORD type = 2;       //Type 链接类型：1-串口，2-网口
        char *pconnectstring = "192.168.5.11";    //0 控制器 IP 地址
        char *pconnectstring2="192.168.5.22";     //1 控制器 IP 地址
        DWORD baud=0;
        WORD ret;                   //返回错误值
        WORD axis=0;                //定长运动轴
        double vel=2000;            //定长运动速度
        double dist=2000;           //定长运动距离
        WORD MyCrd=0;               //插补系 0
        double MyMax_Vel=1000;      //插补速度
        double MyTacc=0.1;          //加减速时间
        double MySpara=0.05;        //S 段时间
        WORD AxisArray[2];          //定义轴
        AxisArray[0] = 0;           //定义插补 0 轴为 X 轴
        AxisArray[1] = 1;           //定义插补 1 轴为 Y 轴
        double pos[2];
        pos[0] = 10000 ;            //定义 X 轴运动距离
        pos[1] = 8000 ;             //Y 轴运动距离
        WORD MyaxisNum=2;           //插补轴数
/**********************函数调用执行*************************/
    //第一步、连接 0、1 号控制器
        ret = smc_board_init(ConnectNo, type, pconnectstring, baud);    //0 控制器
```

```
链接初始化
if(ret!=0)
{   printf("0 控制器初始化失败：smc_board_init=%d\n",ret);}
else
{printf("0 控制器初始化成功\n");}
ret = smc_board_init(ConnectNo2, type, pconnectstring2, baud);
if(ret!=0)
{   printf("1 控制器初始化失败：smc_board_init=%d\n",ret);}
else
{   printf("1 控制器初始化成功\n"); }
//第二步、设置 0、1 号控制器轴 0 和轴 1 为 0
ret=smc_set_position_unit(ConnectNo,0,0);      //位置清零
ret=smc_set_position_unit(ConnectNo,1,0);
ret=smc_set_position_unit(ConnectNo2,0,0);
ret=smc_set_position_unit(ConnectNo2,1,0);
//第三步、设置 0 号控制器定长运动速度参数
ret=smc_set_profile_unit(ConnectNo,axis,0,vel,0.1,0.1,0);
ret=smc_set_s_profile(ConnectNo,axis,0,MySpara);
//第四步、设置 1 控制器插补运动速度参数
ret=smc_set_vector_profile_unit(ConnectNo2,MyCrd,0,MyMax_Vel,MyTacc,MyTacc,
0);
ret=smc_set_vector_s_profile( ConnectNo2,MyCrd,0,MySpara );
//第五步、启动 0 控制器定长运动
ret=smc_pmove_unit(ConnectNo,axis,dist,0);
//第六步、等待 0 控制器定长运动停止
while (smc_check_done(ConnectNo,axis)==0);
//第七步、启动插补运动
ret=smc_line_unit( ConnectNo2,MyCrd,MyaxisNum,AxisArray,pos, 0);
```

## 2.8 BASIC 程序控制功能

雷赛控制器的 BASIC 程序控制功能，用于对控制器中保存的 BASIC 程序进行读取、修改、运行、停止、调试等操作。

BASIC 程序控制相关函数：

| 名称 | 功能 | 参考 |
|---|---|---|
| smc_read_array | 按索引读数组值 | 3.32 节 |
| smc_modify_array | 按索引修改数组值 | |

| | | |
|---|---|---|
| smc_read_var | 读变量值 | |
| smc_modify_var | 修改变量值 | |
| smc_get_stringtype | 读取变量类型 | |
| smc_basic_run | 运行 | |
| smc_basic_stop | 停止 | |
| smc_basic_pause | 暂停 | |
| smc_basic_step_run | 单步运行 | |
| smc_basic_step_over | 运行到下一断点 | |
| smc_basic_continue_run | 继续运行 | |
| smc_basic_state | 当前状态 | |
| smc_basic_current_line | 当前执行行 | |
| smc_basic_break_info | 断点信息 | |
| smc_basic_message | 读取输出信息 | |
| smc_basic_command | 在线命令 | |

## 2.9 G 代码程序控制功能

雷赛控制器的 G 代码程序控制功能，用于对控制器中的 G 代码程序进行控制， 可对控制器中的 G 代码程序进行读取、删除、运行、停止、检查等操作。

G 代码程序控制相关函数：

| 名称 | 功能 | 参考 |
|---|---|---|
| smc_gcode_check_file | 检查文件是否存在 | 3.33 节 |
| smc_gcode_delete_file | 删除文件 | |
| smc_gcode_clear_file | 删除所有文件 | |
| smc_gcode_get_first_file | 读取第一个文件名 | |
| smc_gcode_get_next_file | 读取下一个文件名 | |
| smc_gcode_start | 启动 | |
| smc_gcode_stop | 停止 | |
| smc_gcode_pause | 暂停 | |
| smc_gcode_state | 读取当前状态 | |
| smc_gcode_set_current_file | 设置当前文件 | |
| smc_gcode_get_current_file | 读取当前文件名 | |
| smc_gcode_current_line | 读取当前运行行 | |
| smc_gcode_get_file_profile | 读取 G 代码运行文件属性 | |

# 2.10 总线控制功能

雷赛控制器支持总线功能，包括 CANopen 总线、EtherCAT 总线。总线控制和脉冲控制大部分的用法一样，但是存在一些区别，具体从以下几个部分来详细说明。

## 2.10.1 电机使能

在总线模式下，所有轴在运动之前都需要进行使能操作，不管是总线伺服还是总线步进。和脉冲控制器对电机使能即打开伺服使能信号不同，总控制器的轴使能，需要发送设置轴使能指令，并且在总线轴状态机（state_machine）变成”操作使能”状态后才完成。以下是具体代码部分：

注意：为更好说明该功能，该部分代码使用 VC++语言为例，仅供参考，请根据实际情况使用。

```
void CDMCd1Dlg::OnButtonEnable()        //轴使能操作函数
{
    // TODO: Add your control notification handler code here
    time_t t1,t2;                       //设置时间监控变量，在等待轴状态机变化时防止死循环使用
    unsigned long errcode=0;            //总线错误代码
    unsigned short statemachine=0;   //总线状态机
    nmcs_get_errcode(m_nConnectNo,2,&errcode);       //获取总线状态
    if(errcode==0)       //总线正常才允许使能操作
    {
        nmcs_set_axis_enable(m_nConnectNo,m_nAxis);//设置指定轴使能,此处 m_nAxis 即当前指定轴
        nmcs_get_axis_state_machine(m_nConnectNo,m_nAxis,&statemachine);//获取轴状态机
        t1=time(NULL);              //设置时间
        while(statemachine!=4) //监控轴状态机的值，该值等于 4 表示轴状态机处于准备好状态
        {
            t2=time(NULL);
            if(t2-t1 > 3)           //3 秒时间防止死循环
            {
                GetDlgItem(IDC_STATIC_BSState)->SetWindowText("使能超时，请检查设备");
                return;
            }
            nmcs_set_axis_enable(m_nConnectNo,m_nAxis);   //设置轴使能
            nmcs_get_axis_state_machine(m_nConnectNo,m_nAxis,&statemachine); //获取状态机
        }

    }
    else       //总线不正常状态下不响应使能操作
    {
```

```
        GetDlgItem(IDC_STATIC_BSState)->SetWindowText("总线错误，禁止操作！");
        return;
    }
}
```

### 2.10.2 电机复位

脉冲控制器的复位方式和总线控制器的复位方式有比较大的区别，脉冲控制器对复位的控制（读取原点限位信号，控制方式等）都在控制器部分，总线控制器对复位的控制只有发起复位和等待复位结束，中间的复位过程全部交由驱动器来处理。雷赛控制器支持标准的 34 种回零模式，具体请参考 EtherCAT 回零标准。具体由以下的示例说明。

注意：为更好说明该功能，该部分代码使用 VC++语言为例，仅供参考，请根据实际情况使用。

```
void CDMCd2Dlg::OnZero()          //轴回零操作
{
    // TODO: Add your control notification handler code here
    UpdateData(true);                       //刷新参数
    short iret = 0;                         //函数返回值，用来检测函数执行是否正确
    unsigned short statemachine=0;          //轴状态机标志
    unsigned long errcode=0;                //总线错误代码
    iret = nmcs_get_errcode(m_nConnectNo,2,&errcode);      //获取总线状态
    if(errcode!=0)                          //总线错误禁止回零
    {
        MessageBox("总线错误","错误");
        return;
    }
    iret = nmcs_get_axis_state_machine(m_nConnectNo,m_nAxis,&statemachine);
    if (statemachine!=4)                    //轴状态机错误，禁止回零
    {
        MessageBox("轴状态机错误","错误");
        return;
    }
    if (smc_check_done( m_nConnectNo,m_nAxis ) == 0) //轴在运动中，禁止操作
        return;
    //设置回零参数
    smc_set_home_profile_unit(m_nConnectNo,m_nAxis,m_nSpeedmin,m_nSpeedmax,m_nAcc,m_nDec);
    //设置回零方式
    iret = smc_set_homemode(m_nConnectNo,m_nAxis,m_nPositive,m_nLowspeed,m_nHome,1);
    //启动回零运动
    iret = smc_home_move(m_nConnectNo,m_nAxis);
    //判断当前轴状态
```

```
    while (smc_check_done(m_nConnectNo,m_nAxis)==0)
    {
        AfxGetApp()->PumpMessage();
        GetDlgItem(IDC_BUTTON1)->EnableWindow(false);
    }
    WORD state = 0;
    iret = smc_get_home_result(m_nConnectNo,m_nAxis,&state);
    if(state==1)//回零完成后指令清零
    {
        iret = smc_set_position_unit(m_nConnectNo,m_nAxis,0);
    }
    GetDlgItem(IDC_BUTTON1)->EnableWindow(true);
    UpdateData(false);
}
```

### 2.10.3 IO 控制及电机运动

总线控制器和脉冲控制器在 IO 控制以及轴运动方式上并无实质上的区别，控制函数两者都是一样的，以下以定长以及连续运动、位置清零、减速停止为例做说明。

```
//执行定长运动以及连续运动
void CDMCd1Dlg::OnButtonDo()
{
    // TODO: Add your control notification handler code here
    UpdateData(true);//刷新参数
    short iret = 0;
    unsigned short statemachine=0;
    unsigned long errcode=0;
    iret = nmcs_get_errcode(m_nConnectNo,2,&errcode);
    if(errcode!=0)
    {
        MessageBox("总线错误","错误");
        return;
    }
    iret = nmcs_get_axis_state_machine(m_nConnectNo,m_nAxis,&statemachine);
    if (statemachine!=4)
    {
        MessageBox("轴状态机错误","错误");
        return;
    }
    //注意：以上部分是总线控制方式，需要检测总线状态和轴状态机，以下部分总线控制方式和脉冲控制方式共用
    if (smc_check_done( m_nConnectNo,m_nAxis ) == 0) //已经在运动中
        return;
```

```
    iret = smc_set_equiv(m_nConnectNo, m_nAxis, 1);//设置脉冲当量

    //设定脉冲模式（此处脉冲模式固定为 P+D 方向：脉冲+方向）
    iret = smc_set_pulse_outmode(m_nConnectNo, m_nAxis, 0);
    //设定单轴运动速度参数
    smc_set_profile_unit(m_nConnectNo,m_nAxis,m_nSpeedMin,m_nSpeed,m_nAcc,m_nDec,m_nSpeedSt
op);
    //设定 S 段时间
    iret = smc_set_s_profile(m_nConnectNo,m_nAxis,0,m_nSPara);
    if( m_nActionst == 0 )
    {
        iret = smc_pmove_unit(m_nConnectNo, m_nAxis, m_nPulse*(m_bLogic?1:-1), 0);//相对定长运动
    }
    else
    {
        iret = smc_vmove(m_nConnectNo, m_nAxis, m_bLogic?1:0);     //恒速运动
    }

    UpdateData(false);
}
//执行清除指令位置，脉冲方式和总线方式一致
void CDMCd1Dlg::OnButtonClear()
{
    // TODO: Add your control notification handler code here
    for (int i=0;i<4;i++)
    {
        smc_set_position_unit(m_nConnectNo,i,0);            //指令位置清零
    }
}
//执行减速停止
void CDMCd1Dlg::OnButtonDecstop()
{
    // TODO: Add your control notification handler code here
    UpdateData(true);//刷新参数
    smc_set_dec_stop_time(m_nConnectNo,m_nAxis,m_nDec);//设置 10ms 减速停止时间
    smc_stop(m_nConnectNo,m_nAxis,0);     //减速停止
}
```

### 2.10.4 总线状态

总线控制器由于是主从结构，并且主从结构之间通过网线连接，所以需要在程序中实时扫描总线状态(一般采用定时器或者独立的任务来扫描)，以应对可能出现的异常。当总线状态正常时才能进行相应的操作，当总线状态异常时需要做相应的处理，如停止当前运行、提示报警等。如下例程所示：

```
//扫描总线状态
void CDMCd2Dlg::OnTimer(UINT nIDEvent)
{
    // TODO: Add your message handler code here and/or call default
    unsigned long TotalAxis=0;              //总线上连接的轴数
    unsigned long errcode=0;                //总线状态
    short iret=0;
    iret=nmcs_get_errcode(m_nConnectNo,2,&errcode);      //获取总线状态
    CString string;
    if(errcode==0)                          //状态为 0 表示总线正常，
    {
        nmcs_get_total_axes(m_nConnectNo,&TotalAxis);    //获取当前总线上的轴数
        string.Format("总线状态：正常   连接轴数：%ld", TotalAxis );
        GetDlgItem( IDC_STATIC_BUSState )->SetWindowText( string );
        m_nStatus = smc_check_done(m_nConnectNo,m_nAxis);            //判断当前轴状态
        GetDlgItem( IDC_EDIT_STATUS )->SetWindowText( m_nStatus?"静止":"运动");
        WORD state = 0;
        smc_get_home_result(m_nConnectNo,m_nAxis,&state);//获取回零运动结果
        GetDlgItem( IDC_EDIT_STATUS2 )->SetWindowText( state?"完成":"未完成");

    GetDlgItem( IDC_EDIT_HomeLogicState )->SetWindowText( smc_read_org_pin(m_nConnectNo,m_nAxis)?"高电平":"低电平" );

    GetDlgItem( IDC_EDIT_EZLogicState )->SetWindowText( smc_read_ez_pin(m_nConnectNo,m_nAxis)?"高电平":"低电平" );

            CString strpos;
            CString strspeed;
            smc_get_position_unit(m_nConnectNo,m_nAxis,&m_fUposition);      //获取当前轴位置
            smc_read_current_speed_unit(m_nConnectNo,m_nAxis,&m_NowSpe);    //获取当前轴速度

            strpos.Format("%.3lf",m_fUposition);
            strspeed.Format("%.3f",m_NowSpe);

            GetDlgItem( IDC_EDIT_XPOSITION )->SetWindowText( strpos );
            GetDlgItem( IDC_EDIT_MyNowSpe )->SetWindowText( strspeed );
    }
    Else      //总线异常需停止，然后报警提示
    {
```

```
        smc_emg_stop(m_nConnectNo);              //紧急停止所有轴
        string.Format("总线错误:%lu",errcode);
        GetDlgItem(IDC_STATIC_BUSState)->SetWindowText(string);
    }
    CDialog::OnTimer(nIDEvent);

}
```

总线控制器在使用过程中可能会遇到以下错误信息，有部分错误信息需要给控制器重新上电初始化总线才能消除，有些错误信息可以通过函数操作消除。

| 序号 | 总线错误 | 原因及解决方法 |
| --- | --- | --- |
| 1 | 000e | 总线初始化，若正常会自动消失，若一直报错，则需要检查线路。 |
| 2 | 001e | 增加或者丢失从站；需要重新扫描 |
| 3 | 0009 | 控制器与第一个从站丢失连接需要重新连接 |
| 4 | | |

**//清除总线错误操作**

```
void CDMCd2Dlg::OnButtonBusrst()
{
    // TODO: Add your control notification handler code here
    unsigned long errcode=0;
    nmcs_get_errcode(m_nConnectNo,2,&errcode);
    if(errcode!=0)
    {
        nmcs_clear_errcode(m_nConnectNo,0);      //清除总线错误

    }
    else
    {
        MessageBox("总线正常","错误");
        return;
    }
}
```

# 第三章 函数列表

## 3.1 通讯连接函数

short smc_board_init(WORD ConnectNo,WORD type, char *pconnectstring,DWORD baud)

功 能：控制器链接初始化函数，分配系统资源

参 数：ConnectNo 指定链接号（0-7），默认值 0

Type 链接类型：1-串口，2-网口

Pconnectstring 链接字符串，对应与控制器的 IP 地址或相应的 COM 口

Baud 波特率，默认值 115200

返回值：0： 链接成功，非 0： 链接失败错误码

适用范围：全系列控制器

⚠注意：使用 API 函数动态库时，数据位必须为 8 位。停止位、校验位默认参数下使用，参数发生改变可用函数 smc_board_init_ex。

例 1： 网口、串口初始化设置

short iret =smc_board_init(0,2，“192.168.5.11”，0) //网口初始化

short iret =smc_board_init(0,1，“COM1”，115200) //串口初始化

short smc_board_init_ex(WORD ConnectNo,WORD type, char *pconnectstring,DWORD dwBaudRate, DWORD dwByteSize, DWORD dwParity, DWORD dwStopBits)

功 能：控制器高级链接初始化函数，分配系统资源

参 数：ConnectNo 指定链接号（0-7），默认值 0

Type 链接类型：1-串口，2-网口

Pconnectstring 链接字符串，对应与控制器的 IP 地址或相应的 COM 口

dwBaudRate 波特率，默认值 115200

dwByteSize 8：数据为 8

dwParity 校验位，0：无校验、1：奇校验、2：偶校验

dwStopBits　　停止位，1：停止为 1,2：停止位 2

返回值：0：　　链接成功，非 0：　链接失败错误码

适用范围：全系列控制器

⚠注意：使用 API 函数动态库时，数据位必须为 8 位。一般被函数 smc_set_com 修改了停止位、校验位等情况下使用。

short smc_board_close(WORD ConnectNo)

功　能：控制器关闭函数，释放系统资源

参　数：ConnectNo 指定链接号（0-7）

返回值：错误代码

适用范围：全系列控制器

short smc_set_connect_timeout(DWORD timems);

功　能：网络链接超时时间

参　数：timems 超时时间，单位 ms，超时时间等于 0 或者没调用过该函数时默认为 5 秒钟

返回值：错误代码

适用范围：全系列控制器

short smc_get_release_version(WORD ConnectNo,char *ReleaseVersion)

功　能：读取发布版本号

参　数：ConnectNo　　指定链接号（0-7），默认值 0

ReleaseVersion　返回控制器发布版本号

返回值：　错误代码

适用范围：全系列控制器

short smc_get_card_version(WORD ConnectNo,DWORD *CardVersion)

功能：获取控制器硬件版本号

参　数：ConnectNo　　指定链接号：0-7,默认值 0

CardVersion　　返回控制器硬件版本号

返回值：错误代码

适用范围：全系列控制器

short smc_get_card_soft_version(WORD ConnectNo,DWORD *FirmID,DWORD *SubFirmID)

功能：获取控制器固件版本号

参　数：ConnectNo　指定链接号：0-7,默认值 0

FirmID　　返回控制器固件类型

SubFirmID　返回控制器固件版本号

返回值：错误代码

适用范围：全系列控制器

⚠注意：参数 FirmID 类型为十六进制

short smc_get_card_lib_version(DWORD *LibVer)

功　能：获取控制器动态库文件版本号

参　数：LibVer　　返回库版本号

返回值：错误代码

适用范围：全系列控制器

short smc_get_total_axes(WORD ConnectNo,DWORD *TotalAxis)

功能：获取当前控制器的轴数

参　数：ConnectNo　　指定链接号：0-7,默认值 0

TotalAxis　　返回当前控制器的轴数

返回值：错误代码

适用范围：全系列控制器

short smc_set_debug_mode(WORD mode,const char *FileName)

功　能：函数调用打印输出设置

参　数：mode　　打印输出使能状态，0：禁止，1：使能

FileName　　文件保存路径：

参数文件名+后缀：相对路径

完整描述参数文件的路径+文件名后缀：绝对路径

返回值：错误代码

适用范围：全系列控制器

⚠注意：使能打印输出后，可监控运动函数库的调用情况。在用户调用函数时，将输出相关信息，并保存在指定文件路径中

short smc_get_debug_mode(WORD mode,char *FileName)

功　能：读取函数调用打印输出设置

参　数：mode　　返回打印输出使能状态

FileName　　返回文件保存路径

返回值：错误代码

适用范围：全系列控制器

short smc_set_ipaddr( WORD ConnectNo,const char* IpAddr)

功能：设置控制器新 IP 地址

参　数：ConnectNo　指定链接号：0-7,默认值 0

IpAddr　　新 IP 地址字符串，如“192.168.5.11”

返回值：错误代码

适用范围：全系列控制器

short smc_get_ipaddr( WORD ConnectNo, char* IpAddr)

功能：读取控制器 IP 地址

参　数：ConnectNo　指定链接号：0-7,默认值 0

IpAddr　　返回 IP 地址字符串，如“192.168.5.11”

返回值：错误代码

适用范围：全系列控制器

short smc_set_com(WORD ConnectNo,WORD com, DWORD dwBaudRate, WORD wByteSize, WORD

wParity, WORD wStopBits);

功能：设置控制器 COM 口参数

参数：ConnectNo　　指定链接号：0-7,默认值 0

Com　　com 口:1-RS232、2-RS485

dwBaudRate　　波特率，如 9600、19200、　115200 等。

wByteSize　　数据位：7、8。默认值为 8

wParity　　较验位：0-无校验，1-奇校验，2-偶校验

wStopBits 停止位：1,2

返回值：错误代码

适用范围：全系列控制器

⚠注意：使用 API 函数动态库时，数据位必须为 8 位。

short smc_get_com(WORD ConnectNo,WORD com, DWORD* dwBaudRate, WORD* wByteSize, WORD* wParity, WORD*dwStopBits)

功能：读取控制器 COM 口参数

参 数：ConnectNo　　指定链接号：0-7,默认值 0

Com　　com 口:1-RS232、2-RS485

dwBaudRate　　返回波特率，如 9600、19200、　115200 等。

wByteSize　　返回数据位：7、8

wParity　　返回较验位：0-无校验，1-奇校验，2-偶校验

wStopBits　　返回停止位：1,2

返回值：错误代码

适用范围：全系列控制器

## 3.2　脉冲模式

short smc_set_pulse_outmode(WORD ConnectNo,WORD axis,WORD outmode)

功　能：设置指定轴的脉冲输出模式

参　数：ConnectNo　　指定链接号：0-7,默认值 0

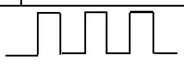
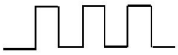
axis　　指定轴号，取值范围：0-控制器最大轴数-1

outmode　　脉冲输出方式选择，其值如表 3.1 所示

返回值：错误代码

适用范围:脉冲型全系列控制器

表 3.1 指令脉冲输出模式

| 脉冲输出模式 | 正方向脉冲 | | 负方向脉冲 | |
|---|---|---|---|---|
| | PULSE 输出端 | DIR 输出端 | PULSE 输出端 | DIR 输出端 |
| 0 |  | 高电平 |  | 低电平 |
| 1 |  | 高电平 |  | 低电平 |
| 2 |  | 低电平 |  | 高电平 |
| 3 |  | 低电平 |  | 高电平 |
| 4 |  | 高电平 | 高电平 |  |
| 5 |  | 低电平 | 低电平 |  |
| 6 | PULSE 输出端  DIR 输出端  | | PULSE 输出端  DIR 输出端  | |

⚠注意：1、在调用运动函数（如：smc_vmove 等）输出脉冲之前，一定要根据驱动器接收脉冲的模式调用 smc_set_pulse_outmode 设置控制器脉冲输出模式。

2、AB 相输出 300 系列、600 系列支持

short smc_get_pulse_outmode(WORD ConnectNo,WORD axis,WORD* outmode)

功　能：读取指定轴的脉冲输出模式设置

参　数：ConnectNo　　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

outmode　　返回脉冲输出方式

返回值：错误代码

适用范围:脉冲型全系列控制器

## 3.3　脉冲当量

short smc_set_equiv(WORD ConnectNo,WORD axis, double equiv)

功　能：设置脉冲当量值

参　数：ConnectNo　　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

equiv　　脉冲当量，单位：pulse/unit

返回值：错误代码

适用范围：全系列控制器

⚠注意：1）该函数适用于高级运动函数（包括点位、插补、连续插补运动）

2）当使用高级运动函数进行运动前，必须先使用该函数设置各运动轴的脉冲当量值，该值不能设置为 0

short smc_get_equiv(WORD ConnectNo,WORD axis, double *equiv)

功　能：返回脉冲当量值设置

参　数：ConnectNo　　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

equiv　　返回脉冲当量设置值

返回值：错误代码

适用范围：全系列控制器

## 3.4　反向间隙设置

short smc_set_backlash_unit(WORD ConnectNo,WORD axis,double backlash)

功　能：设置反向间隙值

参　数：ConnectNo　　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

backlash　　反向间隙值，单位：unit

返回值：错误代码

适用范围：脉冲型全系列控制器

short smc_get_backlash_unit(WORD ConnectNo,WORD axis,double *backlash)

功　能：读取反向间隙值设置

参　数：ConnectNo　　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

backlash　　返回反向间隙设置值

返回值：错误代码

适用范围：脉冲型全系列控制器

## 3.5　状态监控函数

short smc_check_done(WORD ConnectNo,WORD axis)

功　能：检测指定轴的运动状态

参　数：ConnectNo　　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

返回值：0：指定轴正在运行，1：指定轴已停止

适用范围：全系列控制器

⚠注意：此函数适用于单轴、PVT 运动

short smc_check_done_multicoor(WORD ConnectNo,WORD Crd)

功　能：检测坐标系的运动状态

参　数：ConnectNo　　指定链接号：0-7,默认值 0

Crd　　指定控制器上的坐标系号（取值范围：0~1）

返回值：坐标系状态，0：正在使用中，1：正常停止

适用范围：全系列控制器

⚠注意：此函数适用于插补运动

DWORD smc_axis_io_enable_status(WORD ConnectNo,WORD axis)

功　能：读取指定轴特殊信号的使能状态

参　数：ConnectNo　　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

返回值：见表 3.2。bit0 表示禁用，bit1 表示允许。

适用范围：脉冲型全系列控制器

⚠注意：此函数可用于读取各轴的特殊 IO 信号是否处于使能状态。其返回值为 10 进制，需

转换为 2 进制后，查看各 bit 值得高低。用法类似指令 smc_axis_io_status。

DWORD smc_axis_io_status(WORD ConnectNo,WORD axis)

功　能：读取指定轴有关运动信号的状态

参　数：ConnectNo　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

返回值：见表 3.2

适用范围：全系列控制器

表 3.2 轴的运动信号状态

| 位号 | 信号名称 | 描述 |
|---|---|---|
| 0 | ALM | 1：表示伺服报警信号 ALM 为 ON；　0：OFF |
| 1 | EL+ | 1：表示正硬限位信号 +EL 为 ON；　0：OFF |
| 2 | EL- | 1：表示负硬限位信号 - EL 为 ON；　0：OFF |
| 3 | EMG | 1：表示急停信号 EMG 为 ON；　0：OFF |
| 4 | ORG | 1：表示原点信号 ORG 为 ON；　0：OFF |
| 6 | SL+ | 1：表示正软限位信号+SL 为 ON；　0：OFF |
| 7 | SL- | 1：表示负软件限位信号-SL 为 ON；　0：OFF |
| 8 | INP | 1：表示伺服到位信号 INP 为 ON；　0：OFF |
| 9 | EZ | 1：表示 EZ 信号为 ON；　0：OFF |
| 10 | RDY 保留 | 1：表示伺服准备信号 RDY 为 ON（SMC100 控制器专用）； |
| 11 | DSTP | 1：表示减速停止信号 DSTP 为 ON（SMC100 控制器专用）； |
| 其他位 | 保留 | |

short smc_get_axis_run_mode(WORD ConnectNo, WORD axis,WORD* run_mode)

功　能：读取轴运动模式

参　数：ConnectNo　　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

run_mode　　返回运动模式：

0：空闲

1：Pmove

2：Vmove

3：Hmove

4：Handwheel

5：Ptt / Pts

6：Pvt / Pvts

7：Gear

8：Cam

9：Line

10：Continue

返回值：错误代码

适用范围：全系列控制器

short smc_set_position_unit(WORD ConnectNo,WORD axis, double pos)

功　能：设置当前指令位置计数器值

参　数：ConnectNo　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

Pos　　位置值，单位：unit

返回值：错误代码

适用范围：全系列控制器

short smc_get_position_unit(WORD ConnectNo,WORD axis, double *pos)

功　能：读取当前指令位置计数器值

参　数：　ConnectNo　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

pos　　返回当前位置值，单位：unit

返回值：错误代码

适用范围：全系列控制器

short smc_read_current_speed_unit(WORD ConnectNo,WORD axis, double *current_speed)

功　能：读取轴当前速度

参　数：ConnectNo　　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

current_speed 返回速度值，单位：unit/s。回读速度读取值带符号，正表示正向运动，负表示负向运动

返回值：错误代码

适用范围：全系列控制器

⚠注意：当执行插补及连续插补运动时，使用该函数读取的为矢量速度。

short smc_get_stop_reason(WORD ConnectNo,WORD axis,long* StopReason)

功　能：读取轴停止原因

参　数：ConnectNo　　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

StopReason　　停止原因：

0：正常停止

1：ALM 立即停止，IMD_STOP_AT_ALM

2：ALM 减速停止，DEC_STOP_AT_ALM

3：LTC 外部触发立即停止，IMD_STOP_AT_LTC

4：EMG 立即停止，IMD_STOP_AT_EMG

5：正硬限位立即停止，IMD_STOP_AT_ELP

6：负硬限位立即停止，IMD_STOP_AT_ELN

7：正硬限位减速停止，DEC_STOP_AT_ELP

8：负硬限位减速停止，DEC_STOP_AT_ELN

9：正软限位立即停止，IMD_STOP_AT_SOFT_ELP

10：负软限位立即停止，IMD_STOP_AT_SOFT_ELN

11：正软限位减速停止，DEC_STOP_AT_SOFT_ELP

12：负软限位减速停止，DEC_STOP_AT_SOFT_ELN

13：命令立即停止，IMD_STOP_AT_CMD

14：命令减速停止，DEC_STOP_AT_CMD

15：其它原因立即停止，IMD_STOP_AT_OTHER

16：其它原因减速停止，DEC_STOP_AT_OTHER

17：未知原因立即停止，IMD_STOP_AT_UNKOWN

18：未知原因减速停止，DEC_STOP_AT_UNKOWN

19：外部 IO 减速停止，DEC_STOP_AT_DEC

返回值：错误代码

适用范围：全系列控制器

short smc_clear_stop_reason(WORD ConnectNo,WORD axis)

功　能：清除轴停止原因

参　数：ConnectNo　　指定链接号：0-7,默认值 0

　　axis　　指定轴号，取值范围：0-控制器最大轴数-1

返回值：错误代码

适用范围：脉冲型全系列控制器

short smc_get_target_position_unit(WORD ConnectNo,WORD axis, double * pos)

功　能：读取当前目标位置

参　数：ConnectNo　　指定链接号：0-7,默认值 0

　　axis　　指定轴号，取值范围：0-控制器最大轴数-1

返回值：位置值，单位：unit

适用范围：全系列控制器

short smc_set_workpos_unit(WORD ConnectNo,WORD axis, double pos)

功　能：设置当前工件原点

参　数：ConnectNo　指定链接号：0-7,默认值 0

　　axis　　指定轴号，取值范围：0-控制器最大轴数-1

　　Pos　　设置位置值，单位：unit

返回值：错误代码

适用范围：全系列控制器

short smc_get_workpos_unit(WORD ConnectNo,WORD axis, double *pos)

功　能：读取当前工件原点

参　数：ConnectNo　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

Pos　　返回设置位置值，单位：unit

返回值：错误代码

适用范围：全系列控制器

## 3.6　点位运动函数

short smc_set_profile_unit(WORD ConnectNo,WORD axis,double Min_Vel,double Max_Vel,double Tacc,double Tdec,double Stop_Vel)

功　能：设置单轴运动速度曲线（时间模式）

参　数：ConnectNo　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

Min_Vel　　起始速度，单位：unit/s

Max_Vel　　最大速度，单位：unit/s

Tacc　　加速时间，单位：s

Tdec　　减速时间，单位：s

Stop_Vel　　停止速度，单位：unit/s

返回值：错误代码

适用范围：全系列控制器

⚠注意：由于运动控制器的最大脉冲输出频率为 2MHz，故设置的最大速度与脉冲当量设置值的乘积必须小于 2MHz。

short smc_set_profile_unit_acc(WORD ConnectNo,WORD axis,double Min_Vel,double Max_Vel,double acc,double dec,double Stop_Vel)

功　能：设置单轴运动速度曲线(加速度模式)

参　数：ConnectNo　　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

Min_Vel　　起始速度，单位：unit/s

Max_Vel　　最大速度，单位：unit/s

acc　　加速度，单位：unit/s^2

dec　　减速的，单位：unit/s^2

Stop_Vel　　停止速度，单位：unit/s

返回值：错误代码

适用范围：全系列控制器

⚠注意：由于运动控制器的最大脉冲输出频率为 2MHz，故设置的最大速度与脉冲当量设置值的乘积必须小于 2MHz。

short smc_get_profile_unit(WORD ConnectNo,WORD axis,double* Min_Vel,double* Max_Vel,double* Tacc,double* Tdec,double* Stop_Vel)

功　能：读取单轴运动速度曲线

参　数：ConnectNo　　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

Min_Vel　　返回起始速度设置

Max_Vel　　返回最大速度设置

Tacc　　返回加速时间设置

Tdec　　返回减速时间设置

Stop_Vel　　返回停止速度设置

返回值：错误代码

适用范围：全系列控制器

short smc_set_s_profile( WORD ConnectNo, WORD axis, WORD s_mode, double s_para)

功　能：设置单轴速度曲线 S 段参数值

参　数：ConnectNo　　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

s_mode　　保留参数，固定值为 0

s_para　　S 段时间，单位：s；范围：0~1 s

返回值：错误代码

适用范围：脉冲型全系列控制器

short smc_get_s_profile(WORD ConnectNo,WORD axis,WORD s_mode, double *s_para)

功　能：读取单轴速度曲线 S 段参数值

参　数：ConnectNo　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

s_mode　　保留参数 0

s_para　　返回设置的 S 段时间，单位：s;

返回值：错误代码

适用范围：脉冲型全系列控制器

short smc_pmove_unit(WORD ConnectNo,WORD axis,double Dist,WORD posi_mode)

功　能：定长运动

参　数：ConnectNo　　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

Dist　　目标位置，单位：unit

posi_mode　　运动模式，0：相对坐标模式，1：绝对坐标模式

返回值：错误代码

适用范围：全系列控制器

short smc_vmove(WORD ConnectNo,WORD axis,WORD dir)

功　能：指定轴连续运动

参　数：ConnectNo　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

dir　　运动方向，0：负方向，1：正方向

返回值：错误代码

适用范围：全系列控制器

short smc_reset_target_position_unit(WORD ConnectNo,WORD axis, double New_Pos)

功　能：在线改变指定轴的当前目标位置

参　数：ConnectNo　　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

New_Pos　　新目标位置，单位：unit

返回值：错误代码

⚠注意：1）该函数只适用于轴 PMOVE 运动运动状态下的变位。

2）参数 New_Pos 为绝对位置值，无论当前的运动模式为绝对还是相对坐标模式。

适用范围：全系列控制器

short smc_update_target_position_unit(WORD ConnectNo,WORD axis, double New_Pos)

功　能：强制改变指定轴的当前目标位置

参　数：ConnectNo　　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

New_Pos　　新目标位置，单位：unit

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

注意：1）该函数适用于轴处于 PMOVE 或者空闲运动状态下的变位。

2）参数 New_Pos 为绝对位置值，无论当前的运动模式为绝对坐标还是相对坐标模式。

short smc_change_speed_unit(WORD ConnectNo,WORD axis, double New_Vel,
double Taccdec)

功　能：在线改变指定轴的当前运动速度

参　数：ConnectNo　　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

New_Vel　　新的运行速度，单位：unit/s

Taccdec　　变速时间，单位：s

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

⚠注意：1）该函数适用于单轴运动中的变速。

2）设置的变速时间是从当前速度变速到新速度的时间。此时控制器会重新计算起始速度加速到最高速度所需的时间以及最高速度减速到停止速度所需的时间，即加减速时间会被重新计算。

3）变速一旦成立，该轴的默认运行速度将会被改写为 New_Vel，加减速时间也会被控制器新计算的值所覆盖，也即当调 smc_get_profile_unit 回读速度参数时会发生与 smc_set_profile_unit 所设置的值不一致的现象。

4）在点位运动中变速值 New_Vel 只允许正值，运动方向与变速值无关

5）恒速运动下变速值与其运行方向有关。 若起始运行方向为正，变速值为负值时，运动将往负方向运动。变速值为正时，则运动方向不变，速度都会以设定变速值运行。若起始运行方向为负，变速值为正值时，运动将往正方向运动，变速值为负时，则运动方向不变，速度都会以设定变速值运行。

## 3.7　回原点运动函数

short smc_set_home_pin_logic(WORD ConnectNo,WORD axis,WORD org_logic,
double filter)

功　能：设置 ORG 原点信号

参　数：ConnectNo　　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

org_logic　　ORG 信号有效电平，0：低有效，1：高有效

filter　　保留参数，固定值为 0

返回值：错误代码

适用范围：脉冲型全系列控制器

⚠注意：该函数轴号设为 255 时，所有轴回原点信号参数都被设置。

short smc_get_home_pin_logic(WORD ConnectNo,WORD axis,WORD *org_logic,
double *filter)

功　能：读取 ORG 原点信号设置

参　数：ConnectNo　　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

org_logic　　返回设置的 ORG 信号有效电平

filter　　保留参数

返回值：错误代码

适用范围：脉冲型全系列控制器

short smc_set_homemode(WORD ConnectNo,WORD axis,WORD home_dir,double vel_mode,WORD mode,WORD source)

功　能：设置回原点模式

参　数：ConnectNo　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

home_dir　　回原点方向，0：负向，1：正向

vel_mode　　回原点速度模式，默认值：1

Mode　　回原点模式：

① 总线型控制器回零模式参考驱动器设置

② SMC100 系列控制器支持的回零模式如下（其中 SMC102 只支持 0、1、2 三种模式）：

0：一次回零，即平台高速向原点传感器方向运动，当原点传感器 触发，电机立即停止

1：一次回零加回找，即平台高速向原点传感器方向运动，当原点 传感器触发后，电机低速反转；退出原点传感器触发区域后， 电机立即停止

2：二次回零，即平台高速向原点传感器方向运动，当原点传感器 触发后，电机低速反转；退出原点传感器触发区域后，再次以 低速向原点传感器方向运动；当原点传感器触发，电机立即停 止。

3：一次回零后再记一个 EZ 脉冲进行回零，即平台高速向原点传 感器方向运动，当原点传感器触发后，电机以低速继续向前运 动，当编码器上的 EZ 信号触发后，电机立即

停止

4：记一个 EZ 脉冲进行回零，EZ 信号的触发次数为 1。电机以低 速向前运动，当编码器上的 EZ 信号触发后，电机立即停止。

5：一次回零后反向再记一个 EZ 脉冲进行回零，即平台高速向原 点传感器方向运动，当原点传感器触发后，电机以低速反向运 动，当编码器上的 EZ 信号触发后，电机立即停止。（保留）

③ SMC300、SMC600 系列支持的回零模式如下：

0：一次回原点，即平台高速向原点传感器方向运动，当原点传感器触发，电机立即停止。

1：一次回原点加反找，即平台高速向原点传感器方向运动，当原点传感器触发后，电机低速反转；退出原点传感器触发区域后，电机立即停止。

2：二次回原点，即平台高速向原点传感器方向运动，当原点传感器触发后，电机低速反转；退出原点传感器触发区域后，再次以低速向原点传感器方向运动；当原点传感器触发，电机立即停止。

3：一次回原点 +EZ回原点方式，即平台高速向原点传感器方向运动，当原点传感器触发后，电机以低速继续向前运动，当编码器上的EZ信号触发后，电机立即停止。

4：EZ单独回原点。EZ信号的触发次数为1。电机以低速向前运动，当编码器上的EZ信号触发后，电机立即停止。

5：一次回零再反找 EZ。该方式在回原点运动过程中，当找到原点信号后，减速停止，然后以反找速度反向找到 EZ 生效此时电机停止

6：原点锁存。电机先以设定速度回原点，当原点开关边沿触发时，将当前位置锁存下来，同时电机减速停止。电机减速停止完成后再反向回找锁存位置，运动到锁存位置，电机停止。

7：原点锁存加同向 EZ 锁存。此模式先以模式 3 的方式执行一次原点锁存回零，完成后继续沿设定回零方向运行到 EZ 信号产生，EZ 信号产生时锁存当前位置并执行减速停，电机减速停止之后再反向回找 EZ 的锁存位置，运动到锁存位置，电机停止。

8：单独记一个 EZ 锁存。在回零过程中检测到 EZ 有效边沿出现，锁存当前位置，执行减速停，电机减速停止之后再反向回找 EZ 的锁存位置，运动到锁存位置，电机停止。

9：原点锁存加反向 EZ 锁存。此模式先以模式 3 的方式执行一次原点锁存回零，完成后以与设定回零方向相反的方向运行到 EZ 信号产生，EZ 信号产生时锁存当前位置并执行

减速停，电机减速停止之后再反向回找 EZ 的锁存位置，运动到锁存位置，电机停止

Source　回零计数源，0：指令位置计数器，1：编码器计数器

返回值：错误代码

适用范围：全系列控制器

⚠注意：当回原点模式 mode=4 时，回原点速度模式将固定为低速回原点。

short smc_get_homemode(WORD ConnectNo,WORD axis,WORD* home_dir, double* vel, WORD* mode, WORD* source)

功　能：读取回原点模式

参　数：ConnectNo　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

home_dir　返回回原点方向

vel　　返回回原点速度模式，默认值：1

mode　　返回回原点模式

Source　返回回零计数源，0：指令位置计数器，1：编码器计数器

返回值：错误代码

适用范围：全系列控制器

short smc_set_ez_count(WORD CardNo,WORD axis,WORD Count)

功　能：设置 EZ 个数

参　数：CardNo　　控制卡卡号

axis　　指定轴号，取值范围：0-控制器最大轴数-1

Count　　设置 EZ 个数值

返回值：错误代码

适用范围：SMC300、SMC600 系列

short smc_get_ez_count(WORD CardNo,WORD axis,WORD *Count)

功　能：设置 EZ 个数

参　数：CardNo　　控制卡卡号

axis　　指定轴号，取值范围：0-控制器最大轴数-1

Count 返回 EZ 个数值

返回值：错误代码

适用范围：SMC300、SMC600 系列

short smc_set_home_position_unit(WORD ConnectNo,WORD axis,WORD enable,double position);

功 能：回零完成后设置偏移位置值

参 数：ConnectNo 指定链接号：0-7,默认值 0

axis 指定轴号，取值范围：0-控制器最大轴数-1

enable 使能参数

0：禁止。

1：先清 0，然后运动到指定位置（相对位置）。

2：先运动到指定位置（相对位置），再清 0

position 设置回原点位置

返回值：错误代码

适用范围：SMC300、SMC600 系列

⚠注意：偏移位置，其运行速度、加减速时间为定长运动时设定值。

short smc_get_home_position_unit( WORD ConnectNo, WORD axis,WORD *enable, double* position);

功 能：读取回零完成后偏移位置值

参 数：ConnectNo 指定链接号：0-7,默认值 0

axis 指定轴号，取值范围：0-控制器最大轴数-1

enable 使能参数

0：禁止。

1：先清 0，然后运动到指定位置。

2：先运动到指定位置，再清 0

position 回读回原点位置设置值

返回值：错误代码

适用范围：SMC300、SMC600 系列

short smc_set_home_profile_unit(WORD ConnectNo,WORD axis,double Low_Vel,double High_Vel,double Tacc,double Tdec)

功　能：设置回原点速度参数

参　数：ConnectNo　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

Low_Vel　设置回原点起始速度

High_Vel　设置回原点运行速度

Tacc　　设置回原点加速、减速时间，单位：s

Tdec　　保留值 0

返回值：错误代码

适用范围：全系列控制器

short smc_get_home_profile_unit(WORD ConnectNo,WORD axis,double* Low_Vel,double* High_Vel,double* Tacc,double* Tdec);

功　能：回读回原点速度参数

参　数：ConnectNo　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

Low_Vel　回读回原点起始速度

High_Vel　回读回原点运行速度

Tacc　　回读回原点加减速时间，单位：s

Tdec　　保留值 0

返回值：错误代码

适用范围：全系列控制器

short smc_set_el_home(WORD ConnectNo,WORD axis,WORD mode)

功　能：限位当原点切换函数

参　数：ConnectNo　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

mode　　切换模式:0-不切换，1-正限位当原点，2-负限位当原点

返回值：错误代码

适用范围：脉冲型全系列控制器

short smc_home_move(WORD ConnectNo,WORD axis)

功　能：回原点运动

参　数：ConnectNo　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

返回值：错误代码

适用范围：全系列控制器

short smc_get_home_result(WORD ConnectNo,WORD axis,WORD* state);

功　能：读取回原点运动状态

参　数：ConnectNo　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

state　　返回回原点运动状态，0：未完成，1：完成

返回值：错误代码

适用范围：全系列控制器

## 3.8　PVT 运动函数

short smc_ptt_table_unit(WORD ConnectNo,WORD axis,DWORD count,
double*pTime,double*pPos)

功　能：向指定数据表传送数据，采用 PTT 模式

参　数：ConnectNo　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

count　　数据点个数，每个数据表具有 1000 个存储空间，每个数据点占用 1 个存储空间

pTime　　数据点时间数组，单位：s（精度：ms）；

pPos　　数据点位置数组，单位：unit；

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

⚠注意：（1）下载的第一组（即起始点）数据中位置、时间必须为 0；数组中的数据都是以起始点的数据为参考点

（2）调用该函数向数据表中传递数据时，会删除数据表中原先的数据，因此所有数据应当一次传送完毕。如果使用数据表的轴正在运动，禁止更新数据表

short smc_pts_table_unit (WORD ConnectNo, WORD axis,DWORD count, double* pTime, double* pPos,double* pPercent)

功　能：向指定数据表传送数据，采用 PTS 模式

参　数：ConnectNo　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

count　　数据点个数，每个数据表具有 1000 个存储空间，每个数据点占用 1 个存储空间

pTime　　数据点时间数组，单位：s（精度：ms）；

pPos　　数据点位置数组，单位：unit；

pPercent　数据点百分比数组，百分比的取值范围：[0,100]；

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

⚠注意：（1）下载的第一组（即起始点）数据中位置、时间必须为 0；数组中的数据都是以起始点的数据为参考点。

（2）调用该函数向数据表中传递数据时，会删除数据表中原有数据，因此所有数据应当一次传送完毕。如果使用数据表的轴正在运动，禁止更新数据表。

short smc_pvt_table_unit(WORD ConnectNo,WORD axis,DWORD count,double* pTime,double* pPos,double* pVel)

功　能：向指定数据表传送数据，采用 PVT 模式

参　数：ConnectNo　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

count　　数据点个数，每个数据表具有 5000 个存储空间，每个数据点占用 1 个存储空间

pTime　　数据点时间数组，单位：s（精度：ms）；

pPos　　数据点位置数组，单位：unit；

pVel　　数据点速度数组，单位：unit/s；

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

⚠注意：（1）下载的第一组（即起始点）数据中位置、时间、速度必须为 0；数组中的数据都是以起始点的数据为参考点。

（2） 调用该函数向数据表中传递数据时，会删除数据表中原有数据，因此所有数据应当一次传送完毕。如果使用数据表的轴正在运动，禁止更新数据表。

short smc_pvts_table_unit (WORD ConnectNo,WORD axis,DWORD count,double* pTime,double* pPos,double velBegin,double velEnd)

功　能：向指定数据表传送数据，采用 PVTS 模式

参　数：ConnectNo　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

count　　数据点个数，每个数据表具有 5000 个存储空间，每个数据点占用 1 个存储空间

pTime　　数据点时间数组，单位：s（精度：ms）；

pPos　　数据点位置数组，单位：unit；

velBegin　　设置的第一点的速度，单位：unit/s

velEnd　　设置的最后一点的速度，单位：unit/s

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

⚠注意：（1）下载的第一组（即起始点）数据中位置、时间、速度必须为 0；数组中的数据都是以起始点的数据为参考点。

（2） 调用该函数向数据表中传递数据时，会删除数据表中原有数据，因此所有数据

应当一次传送完毕。如果使用数据表的轴正在运动，禁止更新数据表。

short smc_pvt_move(WORD ConnectNo,WORD AxisNum,WORD* AxisList)

功　能：启动 PVT 运动

参　数：ConnectNo　　指定链接号：0-7，默认值 0

AxisNum　　轴数量，取值范围：1-控制器最大轴数

AxisList　　轴数列表，数组

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

## 3.9　插补运动参数函数

short smc_set_vector_profile_unit(WORD ConnectNo,WORD Crd,double Min_Vel,double Max_Vel,double Tacc,double Tdec,double Stop_Vel)

功　能：设置插补运动速度参数（时间模式）

参　数：ConnectNo　　指定链接号：0-7，默认值 0

Crd　　坐标系号，取值范围：0~1

Min_Vel　　起始速度，单位：unit/s

Max_Vel　　最大速度，单位：unit/s

Tacc　　加速时间，单位：s

Tdec　　减速时间，单位：s

Stop_Vel　　停止速度，单位：unit/s

返回值：错误代码

适用范围：脉冲型全系列控制器

short smc_set_vector_profile_unit_acc(WORD ConnectNo,WORD Crd,double Min_Vel,double Max_Vel,double acc,double dec,double Stop_Vel)

功　能：设置插补运动速度参数(加速度模式)

参　数：ConnectNo　　指定链接号：0-7，默认值 0

Crd　　坐标系号，取值范围：0~1

Min_Vel 起始速度，单位：unit/s

Max_Vel 最大速度，单位：unit/s

acc 加速时间，单位：unit/s^2

dec 减速时间，单位：unit/s^2

Stop_Vel 停止速度，单位：unit/s

返回值：错误代码

适用范围：脉冲型全系列控制器

short smc_get_vector_profile_unit(WORD ConnectNo,WORD Crd,double* Min_Vel,double* Max_Vel,double*Tacc,double*Tdec,double* Stop_Vel)

功 能：读取插补运动速度参数

参 数：ConnectNo 指定链接号：0-7，默认值 0

Crd 坐标系号，取值范围：0~1

Min_Vel 返回起始速度值，单位：unit/s

Max_Vel 返回最大速度值，单位：unit/s

Tacc 返回加速时间值，单位：s

Tdec 返回减速时间值，单位：s

Stop_Vel 返回停止速度值，单位：unit/s

返回值：错误代码

适用范围：脉冲型全系列控制器

short smc_set_vector_s_profile(WORD ConnectNo,WORD Crd,
WORD s_mode,double s_para)

功 能：设置插补运动速度曲线的平滑时间

参 数：ConnectNo 指定链接号：0-7,默认值 0

Crd 坐标系号，取值范围：0~1

s_mode 保留参数，固定值为 0

s_para 平滑时间，单位：s，范围：0~1

返回值：错误代码

适用范围：除 SMC100 系列外的控制器

short smc_get_vector_s_profile(WORD ConnectNo,WORD Crd,WORD s_mode,double *s_para)

功　能：读取设置的插补运动速度曲线平滑时间

参　数：ConnectNo　　指定链接号：0-7,默认值 0

Crd　　坐标系号，取值范围：0~1

s_mode　　保留参数，固定值为 0

s_para　　返回平滑时间设置

返回值：错误代码

适用范围：除 SMC100 系列外的控制器

short smc_set_vector_decstop_time(WORD ConnectNo,WORD Crd,double time)

功　能：设置插补异常减速停止时间参数

参　数：ConnectNo　　指定链接号：0-7,默认值 0

Crd 坐标系号，取值范围：0~1

time 减速停止时间，单位：s

返回值：错误代码

适用范围：除 SMC100 系列外的控制器

short smc_get_vector_decstop_time(WORD ConnectNo,WORD Crd,double* time)

功　能：读取插补异常减速停止时间参数

参　数：ConnectNo　　指定链接号：0-7,默认值 0

Crd 坐标系号，取值范围：0~1

time 返回减速停止时间，单位：s

返回值：错误代码

适用范围：除 SMC100 系列外的控制器

short smc_set_arc_limit(WORD ConnectNo,WORD Crd,WORD Enable,double MaxCenAcc,double MaxArcError)

功　能：设置圆弧插补限速功能

参　数：ConnectNo　　指定链接号：0-7,默认值 0

　　Crd　　坐标系号，取值范围：0~1

　　Enable　　使能参数，0：不限速，1：圆弧限速

　　MaxCenAcc　　保留参数 0

　　MaxArcError　　保留参数 0

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

⚠注意：圆弧限速只用于连续插补模式 1、和模式 2。

short smc_get_arc_limit(WORD ConnectNo,WORD Crd,WORD* Enable,double* MaxCenAcc,double* MaxArcError);

功　能：回读圆弧插补限速功能

参　数：ConnectNo　　指定链接号：0-7,默认值 0

　　Crd　　坐标系号，取值范围：0~1

　　Enable　　返回使能参数，0：不限速，1：圆弧限速

　　MaxCenAcc　　保留参数 0

　　MaxArcError　　保留参数 0

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

## 3.10 单段插补运动函数

short smc_line_unit(WORD ConnectNo,WORD Crd,WORD AxisNum,WORD* AxisList,double* Target_Pos,WORD posi_mode)

功　能：直线插补运动

参　数：ConnectNo　　指定链接号：0-7,默认值 0

　　Crd　　坐标系号，取值范围：0~1

　　AxisNum　　运动轴数，取值范围：2~控制器最大轴数

　　AxisList　　轴号列表

Target_Pos　　目标位置列表，单位：unit

posi_mode　　运动模式，0：相对坐标模式，1：绝对坐标模式

返回值：错误代码

适用范围：脉冲型全系列控制器

short smc_arc_move_center_unit(WORD ConnectNo,WORD crd,WORD AxisNum,WORD* AxisList,double *Target_Pos,double *Cen_Pos,WORD Arc_Dir,long Circle,WORD posi_mode)

功　能：圆心+圆弧终点模式扩展的螺旋线插补运动（可作两轴圆弧插补）

参　数：ConnectNo　　指定链接号：0-7,默认值 0

Crd　　坐标系号，取值范围：0~1

AxisNum　　运动轴数，取值范围：2~控制器最大轴数

AxisList　　轴号列表

Target_Pos　　目标位置数组，单位：unit

Cen_Pos　　圆心位置数组，单位：unit

Arc_Dir　　圆弧方向，0：顺时针，1：逆时针

Circle　　圈数：

负数：表示此时执行的为同心圆插补

该值的绝对值加 1 表示同心圆的圈数。如，-1 即表示 2 圈同心圆插补，-2 表示 3 圈同心圆插补…

自然数：表示此时执行的为螺旋线插补

该值表示螺旋线的圈数。如，0 即表示 0 圈螺旋线插补，1 表示 1 圈螺旋线插补…

posi_mode　　运动模式，0：相对坐标模式，1：绝对坐标模式

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

⚠注意：1）当轴数为 2 时，轴列表前两轴进行平面螺旋或同心圆插补。

2）当轴数为 3、运动轨迹为螺旋插补时，轴列表前两轴平面为基面，进行平面螺旋插补；同时，轴列表第三轴运动指定高度，该轴终点位置与该轴起点位置的差值

为螺旋线段相对于基面的高度。

3）当轴数大于 3、运动轨迹为螺旋插补时，列表前三轴进行螺旋插补的同时，后续轴做线性跟随运动，运动时间与前三轴的运动时间相等。

4）当运动轨迹为螺旋插补时：

轴列表前两轴组成的基面上，当起始点到终点的一半距离大于起始点到圆心的距离，为绽放螺旋线。

轴列表前两轴组成的基面上，当起始点到终点的一半距离小于起始点到圆心的距离，为收敛螺旋线。

轴列表前两轴组成的基面上，当起始点到终点的一半距离等于起始点到圆心的距离，为圆弧插补（插补轴数为 3 时则为圆柱螺旋线）。

short smc_arc_move_radius_unit(WORD ConnectNo,WORD crd,WORD AxisNum,WORD* AxisList,double *Target_Pos,double Arc_Radius,WORD Arc_Dir,long Circle,WORD posi_mode)

功　能：半径+圆弧终点模式扩展的螺旋线插补运动（可作两轴圆弧插补）

参　数：ConnectNo　　指定链接号：0-7,默认值 0

Crd　　坐标系号，取值范围：0~1

AxisNum　　运动轴数，取值范围：2~控制器最大轴数

AxisList　　轴号列表

Target_Pos　　目标位置数组，单位：unit

Arc_Radius　　圆弧半径值，单位：unit

Arc_Dir　　圆弧方向，0：顺时针，1：逆时针

Circle　　圈数，取值范围：大于等于 0

该值表示螺旋线的圈数。如，0 即表示 0 圈螺旋线插补，1 即表示 1 圈螺旋线插补…

posi_mode　　运动模式，0：相对坐标模式，1：绝对坐标模式

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

⚠注意：1）当轴数为 2 时，轴列表前两轴进行平面圆弧插补。

2）当轴数为 3 时，轴列表前两轴平面为基面，进行平面圆弧插补；同时，轴列表第三轴运动指定高度；该轴终点位置与该轴起点位置的差值为圆柱螺旋线段相对于基面的高度。

3）当轴数大于 3 时，轴列表前三轴进行圆柱螺旋插补的同时，后续轴做线性跟随运动，运动时间与前三轴的运动时间相等。

short smc_arc_move_3points_unit(WORD ConnectNo,WORD Crd,WORD AxisNum,WORD* AxisList,double *Target_Pos,double *Mid_Pos,long Circle,WORD posi_mode)

功　能：三点圆弧模式扩展的螺旋线插补运动（可作两轴及三轴空间圆弧插补）

参　数：ConnectNo　指定链接号：0-7,默认值 0

Crd　坐标系号，取值范围：0~1

AxisNum　运动轴数，取值范围：2~控制器最大轴数

AxisList　轴号列表

Target_Pos　目标位置数组，单位：unit

Mid_Pos　中间位置数组，单位：unit

Circle　圈数：

负数：表示此时执行的为空间圆弧插补

该值的绝对值减 1 表示空间圆弧的圈数。如，-1 即表示 0 圈空间圆弧，-2 即表示 1 圈空间圆弧…

自然数：表示此时执行的为圆柱螺旋线插补

该值表示螺旋线的圈数。如，0 即表示 0 圈螺旋线插补，1 即表示 1 圈螺旋线插补…

posi_mode　运动模式，0：相对坐标模式，1：绝对坐标模式

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

⚠注意：1）当轴数为 2 时，轴列表前两轴进行平面圆弧插补。

2）当轴数为 3、运动轨迹为圆柱螺旋插补时，轴列表前两轴平面为基面，进行平面圆弧插补；同时，轴列表第三轴运动指定高度；该轴终点位置与该轴起点位置的差值为圆柱螺旋线段相对于基面的高度。

3） 当轴数大于 3 时，轴列表前三轴进行圆柱螺旋插补或空间圆弧插补的同时，后续轴做线性跟随运动，运动时间与前三轴的运动时间相等。

## 3.11 连续插补运动函数

short smc_conti_set_lookahead_mode(WORD ConnectNo,WORD Crd,WORD mode, long LookaheadSegment, double PathError,double LookaheadAcc)

功　能：设置插补模式及小线段前瞻参数

参　数：ConnectNo　指定链接号：0-7,默认值 0

Crd　　坐标系号，取值范围：0~1

Mode　　插补模式：0-非前瞻模式 0，1-前瞻模式 1，2-非前瞻模式 2

LookaheadSegment　前瞻段数，即每次运行时内部计算段数

PathError　　轨迹误差，单位：unit

LookaheadAcc　拐弯加速度，单位 unit/s^2

返回值：错误代码

适用范围：SMC600 系列控制器

⚠注意：前瞻段数、轨迹误差、拐弯加速度都只有使能模式为前瞻运动时才有效。

short smc_conti_get_lookahead_mode(WORD ConnectNo,WORD Crd,WORD *mode, DWORD *LookaheadSegment, double* PathError,double* LookaheadAcc)

功　能：读取插补模式及小线段前瞻参数

参　数：ConnectNo 指定链接号：0-7,默认值 0

Crd　　坐标系号，取值范围：0~1

Mode　　返回插补模式：0-非前瞻模式 0，1-前瞻模式 1，2-非前瞻模式 2

LookaheadSegment　返回前瞻段数，即每次运行时内部计算段数

PathError　　返回轨迹误差

LookaheadAcc　返回拐弯加速度，单位 unit/s^2

返回值：错误代码

适用范围：SMC600 系列控制器

short smc_conti_set_blend(WORD ConnectNo,WORD Crd,WORD enable)

功　能：设置连续插补 Blend 拐角过渡模式使能状态

参　数：ConnectNo　　指定链接号：0-7,默认值 0

　　Crd　　坐标系号，取值范围：0~1

　　enable　　使能状态，0：禁用，1：使能

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

⚠注意：

1）当插补模式为 0 时起作用，用 smc_conti_set_lookahead_mode 设置

2）当使能 Blend 拐角平滑过渡后，各段运动轨迹之间的拐角将平滑过渡，从而得到更加平滑的速度曲线。

3）当计算机执行到此指令时，该函数设置将存入缓冲区，从该函数的下一条运动函数开始运动时起作用。

4）当使能拐角平滑后，除非再次调用该函数禁止拐角平滑过渡，否则后续的运动过程中拐角都将平滑过渡。

short smc_conti_get_blend(WORD ConnectNo,WORD Crd,WORD* enable)

功　能：读取连续插补 Blend 拐角过度模式使能状态设置

参　数：ConnectNo　　指定链接号：0-7,默认值 0

　　Crd　　坐标系号，取值范围：0~1

　　enable　　读取使能状态设置，0：禁用，1：使能

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

short smc_conti_open_list(WORD ConnectNo,WORD Crd,WORD AxisNum,WORD* AxisList)

功　能：打开连续插补缓冲区

参　数：ConnectNo　　指定链接号：0-7

　　Crd　　坐标系号，取值范围：0~1

　　AxisNum　　运动轴数，取值范围：2~控制器最大轴数

　　AxisList　　轴号列表：

AxisList[0]：X 轴

AxisList[1]：Y 轴

AxisList[2]：Z 轴

AxisList[3]：U 轴

AxisList[4]：V 轴

AxisList[5]：W 轴

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

⚠注意：1）连续缓冲区最多可缓存 5000 条指令。

2）在执行圆弧或螺旋线运动时，XYZ 为主动轴，UVW 为辅助轴；主动轴执行圆弧或螺旋线运动，辅助轴不执行圆弧及螺旋线运动，但跟随主动轴做线性运动。

3）当打开连续插补缓冲区后，则进入连续插补模式；此时，除非当执行完缓冲区中的指令或是调用停止连续插补指令 smc_conti_stop_list 后，参与连续插补的运动轴才能退出连续插补模式。

short smc_conti_start_list(WORD ConnectNo,WORD Crd)

功　能：开始连续插补

参　数：ConnectNo　指定链接号：0-7,默认值 0

Crd　　坐标系号，取值范围：0~1

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

short smc_conti_close_list(WORD ConnectNo,WORD Crd)

功　能：关闭连续插补缓冲区

参　数：ConnectNo　指定链接号：0-7,默认值 0

Crd　　坐标系号，取值范围：0~1

返回值：错误代码

short smc_conti_pause_list(WORD ConnectNo,WORD Crd)

功　能：暂停连续插补

参　数：ConnectNo　　指定链接号：0-7

　　Crd　　坐标系号，取值范围：0~1

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

⚠注意：当暂停连续插补后，连续插补运动将减速停止，当再次调用 smc_conti_start_list 指令时运动控制器将继续运行之前未完成的连续插补轨迹。

short smc_conti_stop_list(WORD ConnectNo,WORD Crd,WORD stop_mode)

功　能：停止连续插补

参　数：ConnectNo　　指定链接号：0-7,默认值 0

　　Crd　　坐标系号，取值范围：0~1

　　stop_mode　　停止模式，0：减速停止，1：立即停止

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

⚠注意：当正在执行连续插补运动时，通过此指令可以中止连续插补运动，并使参与连续插补的运动轴退出连续插补模式。

short smc_conti_change_speed_ratio (WORD ConnectNo,WORD Crd,double Percent)

功　能：动态调整连续插补速度比例

参　数：ConnectNo　　指定链接号：0-7,默认值 0

　　Crd　　坐标系号，取值范围：0~1

　　Percent　　速度比例，取值范围：0~2.0

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

⚠注意：当计算机执行到此指令时，运动控制器将下段轨迹调整连续插补速度比例。

short smc_conti_delay(WORD ConnectNo, WORD Crd, double delay_time,long mark)

功　能：连续插补中暂停延时指令

参　数：ConnectNo　　指定链接号：0-7,默认值 0

Crd　　坐标系号，取值范围：0~1

delay_time　　延时时间，单位：秒

mark　　标号，任意指定，0 表示自动编号

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

⚠注意：1）延时时间为运动停止时的等待时间。

2）当延时时间设置为 0 时，延时时间将无限长。

short smc_conti_line_unit(WORD ConnectNo,WORD Crd,WORD AxisNum,WORD* AxisList,double* Target_Pos,WORD posi_mode,long mark)

功　能：连续插补中直线插补指令

参　数：ConnectNo　　指定链接号：0-7,默认值 0

Crd　　坐标系号，取值范围：0~1

AxisNum　　运动轴数，取值范围：2-控制器最大轴数

AxisList　　轴号列表

Target_Pos　　目标位置数组，单位：unit

posi_mode　　运动模式，0：相对坐标模式，1：绝对坐标模式

mark　　标号，任意指定，0 表示自动编号

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

short smc_conti_arc_move_center_unit(WORD ConnectNo,WORD crd,WORD AxisNum,WORD* AxisList,double *Target_Pos,double *Cen_Pos,WORD Arc_Dir,long Circle,WORD posi_mode,long mark)

功　能：连续插补中基于圆心+终点圆弧扩展的螺旋线插补指令（可作两轴圆弧插补）

参　数：ConnectNo　　指定链接号：0-7,默认值 0

Crd　　坐标系号，取值范围：0~1

| | |
|---|---|
| AxisNum | 轴数，取值范围：2-控制器最大轴数 |
| AxisList | 轴号列表 |
| Target_Pos | 目标位置数组，单位：unit |
| Cen_Pos | 圆心位置数组，单位：unit |
| Arc_Dir | 圆弧方向，0：顺时针，1：逆时针 |
| Circle | 圈数：圈数，取值范围：大于等于 0<br>表示此时执行的为螺旋线插补<br>该值表示螺旋线的圈数。如，0 即表示 0 圈螺旋线插补，1 即表示 1 圈螺旋线插补… |
| posi_mode | 运动模式，0：相对坐标模式，1：绝对坐标模式 |
| mark | 标号，任意指定，0 表示自动编号 |

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

⚠注意：1）轴列表的前三轴必须为 XYZ 轴的组合；关于 XYZUVW 轴对应定义详见函数 smc_conti_open_list 的说明

2）当轴数为 2 时，轴列表前两轴进行平面螺旋或同心圆插补

3）当轴数为 3、运动轨迹为螺旋插补时，轴列表前两轴平面为基面，进行平面螺旋插补；同时，轴列表第三轴运动指定高度，该轴终点位置与该轴起点位置的差值为螺旋线段相对于基面的高度

4）当轴数大于 3、运动轨迹为螺旋插补时，主动轴进行螺旋插补的同时，辅助轴跟随主动轴做线性运动，运动时间与主动轴的总运动时间相等；关于主动轴及辅助轴对应定义详见函数 smc_conti_open_list 的说明

5）当运动轨迹为螺旋插补时：

轴列表前两轴组成的基面上，当起始点到终点的一半距离大于起始点到圆心的距离，为绽放螺旋线

轴列表前两轴组成的基面上，当起始点到终点的一半距离小于起始点到圆心的距离，为收敛螺旋线

轴列表前两轴组成的基面上，当起始点到终点的一半距离等于起始点到圆心的距离，为圆弧插补（插补轴数为 3 时则为圆柱螺旋线）

short smc_conti_arc_move_radius_unit(WORD ConnectNo,WORD crd,WORD AxisNum,WORD* AxisList,double *Target_Pos,double Arc_Radius,WORD Arc_Dir,long Circle,WORD posi_mode,long mark)

功　能：连续插补中基于半径+终点圆弧扩展的圆柱螺旋线插补指令（可作两轴圆弧插补）

参　数：ConnectNo　　指定链接号：0-7,默认值 0

Crd　　坐标系号，取值范围：0~1

AxisNum　　运动轴数，取值范围：2-控制器最大轴数

AxisList　　轴号列表

Target_Pos　　目标位置数组，单位：unit

Arc_Radius　　圆弧半径值，单位：unit

Arc_Dir　　圆弧方向，0：顺时针，1：逆时针

Circle　　圈数，取值范围：大于等于 0

该值表示螺旋线的圈数。如，0 即表示 0 圈螺旋线插补，1 即表示 1 圈螺旋线插补…

posi_mode　　运动模式，0：相对坐标模式，1：绝对坐标模式

mark　　标号，任意指定，0 表示自动编号

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

⚠注意：1）轴列表的前三轴必须为 XYZ 轴的组合；关于 XYZUVW 轴对应定义详见函数 smc_conti_open_list 的说明。

2）当轴数为 2 时，轴列表前两轴进行平面圆弧插补。

3）当轴数为 3 时，轴列表前两轴平面为基面，进行平面圆弧插补；同时，轴列表第三轴运动指定高度；该轴终点位置与该轴起点位置的差值为圆柱螺旋线段相对于基面的高度。

4）当轴数大于 3 时，主动轴进行圆柱螺旋插补的同时，辅助轴跟随主动轴做线性运动，运动时间与主动轴的总运动时间相等；关于主动轴及辅助轴对应定义详见函数 smc_conti_open_list 的说明。

short smc_conti_arc_move_3points_unit(WORD ConnectNo,WORD Crd,WORD AxisNum,WORD* AxisList,double *Target_Pos,double *Mid_Pos,long Circle,WORD posi_mode,long mark)

功　能：连续插补中基于三点圆弧扩展的圆柱螺旋线插补指令（可作两轴及三轴圆弧插补）

参　数：ConnectNo　指定链接号：0-7,默认值 0

Crd　坐标系号，取值范围：0~1

AxisNum　运动轴数，取值范围：2-控制器最大轴数

AxisList　轴号列表

Target_Pos　目标位置数组，单位：unit

Mid_Pos　中间位置数组，单位：unit

Circle　圈数

负数：表示此时执行的为空间圆弧插补

该值的绝对值减 1 表示空间圆弧的圈数。如，-1 即表示 0 圈空间圆弧，-2 即表示 1 圈空间圆弧…

自然数：表示此时执行的为圆柱螺旋线插补

该值表示螺旋线的圈数。如，0 即表示 0 圈螺旋线插补，1 即表示 1 圈螺旋线插补…

posi_mode　运动模式，0：相对坐标模式，1：绝对坐标模式

mark　标号，任意指定，0 表示自动编号

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

⚠注意：1）轴列表的前三轴必须为 XYZ 轴的组合；关于 XYZUVW 轴对应定义详见函数 smc_conti_open_list 的说明。

2）当轴数为 2 时，轴列表前两轴进行平面圆弧插补。

3）当轴数为 3、运动轨迹为圆柱螺旋插补时，轴列表前两轴平面为基面，进行平面圆弧插补；同时，轴列表第三轴运动指定高度；该轴终点位置与该轴起点位置的差值为圆柱螺旋线段相对于基面的高度。

4）当轴数大于 3 时，主动轴进行圆柱螺旋插补或空间圆弧插补的同时，辅助轴跟随主动轴做线性运动，运动时间与主动轴的总运动时间相等；关于主动轴及辅助轴对应定义详见函数 smc_conti_open_list 的说明。

short smc_conti_pmove_unit(WORD ConnectNo,WORD Crd,WORD axis,double dist,WORD posi_mode,WORD mode,long imark)

功　能：连续插补中控制指定其它轴运动指令

参　数：ConnectNo　　指定链接号：0-7,默认值 0

Crd　　坐标系号，取值范围：0~1

axis　　指定轴号，范围 0-控制器最大轴数-1

dist　　目标位置，单位：unit

posi_mode　　运动模式，0：相对坐标模式，1：绝对坐标模式

mode　　模式：

0：暂停启动（当缓冲区中的上一段插补运动结束后，执行此段定长运动；当本段定长运动结束后，再执行下一段插补运动）

1：直接启动（当缓冲区中的上一段插补运动结束后，执行此段定长运动，并且同时执行下一段插补运动）

mark　　标号，任意指定，0 表示自动编号

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

⚠注意：1）该指令可以实现在连续插补运动中，控制指定轴做定长运动。

2）该轴不能为参与连续插补的运动轴。

3）在使用该指令控制轴运动前，必须先使用函数 smc_set_profile_unit 设置该轴的运行速度。

## 3.12 连续插补状态检测函数

long smc_conti_remain_space(WORD ConnectNo,WORD Crd)

功　能：查询连续插补缓冲区剩余插补空间

参　数：ConnectNo　　指定链接号：0-7,默认值 0

Crd　　坐标系号，取值范围：0~1

返回值：连续插补缓冲区剩余大小

适用范围：SMC300、SMC600 系列控制器

long smc_conti_read_current_mark (WORD ConnectNo,WORD Crd)

功　能：读取连续插补缓冲区当前插补段号

参　数：ConnectNo　　指定链接号：0-7,默认值 0

　　Crd　　坐标系号，取值范围：0~1

返回值：当前连续插补段号

适用范围：SMC300、SMC600 系列控制器

short smc_conti_get_run_state(WORD ConnectNo,WORD crd)

功　能：读取连续插补运动状态

参　数：ConnectNo　　指定链接号：0-7,默认值 0

　　Crd　　坐标系号，取值范围：0~1

返回值：运动状态，0：运动中，1：暂停中，2：正常停止，3：未启动，4：空闲

　　5：异常减速停止中

适用范围：SMC300、SMC600 系列控制器

short smc_check_done_multicoor (WORD ConnectNo,WORD Crd)

功　能：检测连续插补运行状态

参　数：ConnectNo　　指定链接号：0-7,默认值 0

　　Crd　　坐标系号，取值范围：0~1

返回值：运动状态，0：运行中，1：停止

适用范围：SMC300、SMC600 系列控制器

## 3.13 连续插补 IO 控制函数

short smc_conti_set_pause_output(WORD ConnectNo,WORD crd,WORD action,long mask,long state)

功　能：设置连续插补暂停及异常停止时 IO 输出状态

参　数：ConnectNo　指定链接号：0-7,默认值 0

　　Crd　　坐标系号，取值范围：0~1

action　　激活模式：

0：保持原状

1：暂停连续插补时输出设定的 IO 状态，恢复运行时不恢复暂停前的 IO 状态

2：暂停连续插补时输出设定的 IO 状态，继续运行时恢复暂停前的 IO 状态

3：当暂停、停止连续插补，或遇到其他异常停止（如碰到 EMG 信号）时，输出设定的 IO 状态

mask　　选择输出端口标志：bit0~bit31 代表 Out0~Out31，位值为 1 时输出，位值为 0 不输出

state　　输出电平状态：bit0~bit31 代表 Out0~Out31，位值为 1 时输出高电平，位值为 0 时输出低电平

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

激活模式 3 的说明：1）暂停连续插补时，运动控制器输出设定的 IO 状态，继续运行时恢复暂停前的 IO 状态。

2）停止连续插补、或遇到其他异常停止时，运动控制器输出设定的 IO 状态，但是再次启动连续插补时不会恢复之前的 IO 状态。

short smc_conti_get_pause_output(WORD ConnectNo,WORD crd,WORD* action,long* mask,long* state)

功　能：读取连续插补暂停及异常停止时 IO 输出状态设置

参　数：ConnectNo　指定链接号：0-7,默认值 0

Crd　　坐标系号，取值范围：0~1

action　　返回激活状态设置

mask　　返回输出选择标志设置

state　　返回输出电平状态设置

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

short smc_conti_wait_input(WORD ConnectNo,WORD Crd,WORD bitno,WORD on_off,
double TimeOut,long mark)

功　能：连续插补等待 IO 输入

参　数：ConnectNo　指定链接号：0-7,默认值 0
Crd　坐标系号，取值范围：0~1
bitno　输入口号，取值范围：0~31
on_off　电平状态，0：低电平，1：高电平
TimeOut　超时时间，单位：s
mark　标号，任意指定，0 表示自动编号

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

⚠注意：1）当超时时间设为 0 时，运动控制器将一直等待 IO 输入信号，时间为无限长。
2）超时时间不为 0 时，在超时时间内，一旦有相应的 IO 响应，马上运行下段轨迹。若无 IO 响应，等待时间大于超时时间后，程序将自动运行下段轨迹。

short smc_conti_delay_outbit_to_start(WORD ConnectNo, WORD Crd, WORD bitno,WORD on_off,double delay_value,WORD delay_mode,double ReverseTime)

功　能：连续插补中相对于轨迹段起点 IO 滞后输出（段内执行）

参　数：ConnectNo　指定链接号：0-7,默认值 0
Crd　坐标系号，取值范围：0~1
bitno　输出口号，取值范围：0~31
on_off　电平状态，0：低电平，1：高电平
delay_value　滞后值，单位：s（滞后时间模式）或 unit（滞后距离模式）
delay_mode　滞后模式，0：滞后时间，1：滞后距离
ReverseTime　电平输出后的延时翻转时间，单位：s

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

⚠注意：1）设置的 IO 操作，将在该指令的下一条轨迹中起作用。

2）当 ReverseTime 参数设置为 0 时，相应 IO 端口电平将不会翻转，保持值不变。

3）当滞后模式选择为滞后距离时，位置源为指令位置计数器。

short smc_conti_delay_outbit_to_stop(WORD ConnectNo, WORD Crd, WORD bitno,WORD on_off,double delay_time,double ReverseTime)

功　能：连续插补中相对于轨迹段终点 IO 滞后输出

参　数：ConnectNo　指定链接号：0-7,默认值 0

Crd　　坐标系号，取值范围：0~1

bitno　　输出口号，取值范围：0~31

on_off　　电平状态，0：低电平，1：高电平

delay_time　滞后时间，单位：s

ReverseTime　保留参数，固定值为 0

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

⚠注意：1）设置的 IO 操作，将在该指令的下一条轨迹结束后起作用。

2）如果使用了 smc_conti_clear_io_action 函数清除段内未执行完的 IO 动作时，那么该指令将不会被执行。

short smc_conti_ahead_outbit_to_stop(WORD ConnectNo, WORD Crd, WORD bitno,WORD on_off,double ahead_value,WORD ahead_mode,double ReverseTime)

功　能：连续插补中相对于轨迹段终点 IO 提前输出（段内执行）

参　数：ConnectNo　指定链接号：0-7,默认值 0

Crd　　坐标系号，取值范围：0~1

bitno　　输出口号，取值范围：0~31

on_off　　电平状态，0：低电平，1：高电平

ahead_value　提前值，单位：s（提前时间模式）或 unit（提前距离模式）

ahead_mode　提前模式，0：提前时间，1：提前距离

ReverseTime　电平输出后的延时翻转时间，单位：s

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

⚠注意：1）设置的 IO 操作，将在该指令的下一条轨迹中起作用。

2）当 ReverseTime 参数设置为 0 时，相应 IO 端口电平将不会翻转，保持设置值不变

3）当滞后模式选择为滞后距离时，位置源为指令位置计数器。

short smc_conti_write_outbit(WORD ConnectNo, WORD Crd, WORD bitno,WORD on_off,double ReverseTime)

功　能：连续插补中缓冲区立即 IO 输出

参　数：ConnectNo　指定链接号：0-7

Crd　　坐标系号，取值范围：0~1

bitno　　输出口号，取值范围：0~31

on_off　　电平状态，0：低电平，1：高电平

ReverseTime　电平输出后的延时翻转时间，单位：s

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

⚠注意：1）当计算机执行到此指令时，将该指令存入缓冲区，当缓冲区中的上一段运动指令执行完毕时，该指令将执行。

2）该指令调用之后，前一段轨迹与下一段轨迹的速度曲线将不连续，即 Blend 平滑模式在这两段轨迹之间不起作用。

3）当 ReverseTime 参数设置为 0 时，相应 IO 端口电平将不会翻转，保持设置值不变。

short smc_conti_clear_io_action(WORD ConnectNo, WORD Crd, DWORD IoMask)

功　能：清除段内未执行完的 IO 动作

参　数：ConnectNo　指定链接号：0-7,默认值 0

Crd　　坐标系号，取值范围：0~1

IoMask　　清除标志：bit0~bit31 分别表示 Out0~Out31 输出口；位值：1：清除对应输出口段内未执行完的动作，比如翻转时间到达后 IO 翻转动作；0：不操作

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

⚠注意：该函数对 smc_conti_delay_outbit_to_start、smc_conti_ahead_outbit_to_stop、smc_conti_delay_outbit_to_stop 指令起作用。

## 3.14 PWM 立即输出函数

short smc_set_pwm_output(WORD ConnectNo,WORD pwm_no,double fDuty, double fFre)

功　能：设置 PWM 立即输出参数

参　数：ConnectNo　指定链接号：0-7,默认值 0

pwm_no　PWM 通道，取值范围：0~1

fDuty　占空比，取值范围：0~1

fFre　频率，取值范围：1HZ-500KHZ

返回值：错误代码

适用范围：除 SMC100 系列控制器外的控制器

short smc_get_pwm_output(WORD ConnectNo,WORD pwm_no,double* fDuty, double* fFre)

功　能：读取 PWM 当前输出参数

参　数：ConnectNo　指定链接号：0-7,默认值 0

pwm_no　PWM 通道，取值范围：0~1

fDuty　返回占空比设置值

fFre　返回频率设置值，取值范围：1HZ-500KHZ

返回值：错误代码

适用范围：除 SMC100 系列控制器外的控制器

## 3.15 通用 IO 接口函数

short smc_read_inbit(WORD ConnectNo, WORD bitno)

功　能：读取指定控制器的某个输入端口的电平

参　数：ConnectNo　指定链接号：0-7,默认值 0

bitno 输入端口号，取值范围： 0~控制器本机输入口数-1

返回值：指定的输入端口电平：0：低电平，1：高电平

适用范围：全系列控制器

short smc_write_outbit(WORD ConnectNo, WORD bitno,WORD on_off)

功　能：设置指定控制器的某个输出端口的电平

参　数：ConnectNo　指定链接号：0-7,默认值 0

bitno 输出端口号，取值范围： 0~控制器本机输出口数-1

on_off 输出电平，0：低电平，1：高电平

返回值：错误代码

适用范围：全系列控制器

short smc_read_outbit(WORD ConnectNo, WORD bitno)

功　能：读取指定控制器的某个输出端口的电平

参　数：ConnectNo　指定链接号：0-7,默认值 0

bitno 输出端口号，取值范围： 0~控制器本机输出口数-1

返回值：指定输出端口的电平，0：低电平，1：高电平

适用范围：全系列控制器

DWORD smc_read_inport(WORD ConnectNo,WORD portno)

功　能：读取指定控制器的全部输入端口的电平

参　数：ConnectNo　指定链接号：0-7,默认值 0

portno　　IO 组号，取值范围： SMC604A：0~1, 其它控制器为 0

返回值：各 IO 口的 bit 值

适用范围：全系列控制器

⚠注意：在 IO 口在 32 位内 PORT 号为 0.超出 32 位 PORT 为 1。返回值为 10 进制，转换为 2 进制后 bit 值对应各端口输入状态值

DWORD smc_read_outport(WORD ConnectNo,WORD portno)

功　能：读取指定控制器的全部输出口的电平

参　数：ConnectNo　指定链接号：0-7,默认值 0

portno　IO 组号，取值范围： SMC604A：0~1, 其它控制器为 0

返回值：各 IO 口的 bit 值

适用范围：全系列控制器

⚠注意：在 IO 口在 32 位内 PORT 号为 0.超出 32 位 PORT 为 1。返回值为 10 进制，转换为 2 进制后 bit 值对应各端口输出状态值。

short smc_write_outport(WORD ConnectNo, WORD portno,DWORD port_value)

功　能：设置指定控制器的全部输出口的电平

参　数：ConnectNo　指定链接号：0-7,默认值 0

portno　IO 组号，取值范围： SMC604A：0~1, 其它控制器为 0

port_value　输入端口电平数值

返回值：错误代码

适用范围：全系列控制器

short smc_reverse_outbit(WORD ConnectNo,WORD bitno,double reverse_time)

功　能：IO 输出延时翻转

参　数：ConnectNo　指定链接号：0-7,默认值 0

bitno　输出端口号，取值范围：0-控制器本机输出口数-1

reverse_time　延时翻转时间，单位：s

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

⚠注意：1）该函数只能对 OUT0-各控制器端口数值，进行操作

2）当延时翻转时间参数设置为 0 时，此时延时翻转时间将为无限大

short smc_set_io_count_mode(WORD ConnectNo,WORD bitno,WORD mode,double filter_time)

功　能：设置 IO 计数模式；

参　数：ConnectNo　指定链接号：0-7,默认值 0

bitno 输入端口号，取值范围：0-控制器本机输入口数-1
mode IO 计数模式，0：禁用，1：上升沿计数，2：下降沿计数
filter_time 滤波时间，单位：s

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

short smc_get_io_count_mode(WORD ConnectNo,WORD bitno,WORD *mode,double* filter_time)

功 能：读取 IO 计数模式设置；

参 数：ConnectNo 指定链接号：0-7,默认值 0
bitno 输入端口号，取值范围：0-控制器本机输入口数
mode 返回 IO 计数模式，0：禁用，1：上升沿计数，2：下降沿计数
filter_time 返回滤波时间，单位：s

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

short smc_set_io_count_value(WORD ConnectNo,WORD bitno,DWORD CountValue)

功 能：重置 IO 计数值

参 数：ConnectNo 指定链接号：0-7,默认值 0
bitno 输入端口号，取值范围：0-控制器本机输入口数-1
CountValue IO 计数值

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

short smc_get_io_count_value(WORD ConnectNo,WORD bitno,DWORD* CountValue)

功 能：读取 IO 计数值；

参 数：ConnectNo 指定链接号：0-7,默认值 0
bitno 输入端口号，取值范围：0-控制器本机输入口数-1
CountValue 返回 IO 计数值

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

## 3.16 专用 IO 接口函数

short smc_set_inp_mode(WORD ConnectNo,WORD axis,WORD enable,WORD inp_logic)

功　能：设置指定轴的 INP 信号

参　数：ConnectNo　　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

enable　　INP 信号使能，0：禁止，1：允许

inp_logic　　INP 信号的有效电平，0：低有效，1：高有效

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

⚠注意：（1）当使能 INP 信号功能后，只有在 INP 信号为有效状态时，对应的轴才能进行运动，否则此时检测轴的状态是正在运行（即对轴运动作限制），没有硬件接口 INP 的需先映射 INP IO 口。

（2）该函数轴号设为 255 时，所有轴 INP 信号参数都被设置。

short smc_get_inp_mode(WORD ConnectNo,WORD axis,WORD *enable,WORD *inp_logic)

功　能：读取指定轴的 INP 信号设置

参　数：ConnectNo　　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

enable　　返回 INP 信号使能状态

inp_logic　　返回设置的 INP 信号有效电平

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

short smc_set_alm_mode(WORD ConnectNo,WORD axis,WORD enable,WORD alm_logic,WORD alm_action)

功　能：设置指定轴的 ALM 信号

参　数：ConnectNo　　指定链接号：0-7,默认值 0
　　axis　　指定轴号，取值范围：0-控制器最大轴数-1
　　enableALM　　信号使能，0：禁止，1：允许
　　alm_logicALM　　信号的有效电平，0：低有效，1：高有效
　　alm_actionALM　　信号的制动方式，0：立即停止（只支持该方式）
返回值：错误代码
适用范围：脉冲型全系列控制器

⚠注意：该函数轴号设为 255 时，所有轴报警信号参数都被设置。

short smc_get_alm_mode(WORD ConnectNo,WORD axis,WORD *enable,WORD *alm_logic,WORD *alm_action)
功　能：读取指定轴的 ALM 信号设置
参　数：ConnectNo　指定链接号：0-7,默认值 0
　　axis　　指定轴号，取值范围：0-控制器最大轴数-1
　　enable　　返回 ALM 信号使能状态
　　alm_logic　返回设置的 ALM 信号有效电平
　　alm_action　返回 ALM 信号的制动方式
返回值：错误代码
适用范围：脉冲型全系列控制器

short smc_write_sevon_pin(WORD ConnectNo,WORD axis,WORD on_off)
功　能：控制指定轴的伺服使能端口的输出
参　数：ConnectNo　指定链接号：0-7,默认值 0
　　axis　　指定轴号，取值范围：0-控制器最大轴数-1
　　on_off　　设置伺服使能端口电平，0：低电平，1：高电平
返回值：错误代码
适用范围：SMC300、SMC600 系列控制器

short smc_read_sevon_pin(WORD ConnectNo,WORD axis)

功　能：读取指定轴的伺服使能端口的电平

参　数：ConnectNo　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

返回值：伺服使能端口电平，0：低电平，1：高电平

适用范围：SMC300、SMC600 系列控制器

short smc_write_erc_pin(WORD ConnectNo,WORD axis,WORD sel)

功　能：控制指定轴的 ERC 信号输出

参　数：ConnectNo　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

sel　　输出电平，0：低电平，1：高电平

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

short smc_read_erc_pin(WORD ConnectNo,WORD axis)

功　能：读取指定轴的 ERC 端口电平

参　数：ConnectNo　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

返回值：ERC 端口电平，0：低电平，1：高电平

适用范围：脉冲型 SMC300、SMC600 系列控制器

short smc_read_alarm_pin(WORD ConnectNo,WORD axis)

功　能：读取指定轴的 ALARM 端口电平

参　数：ConnectNo　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数

返回值：ALARM 端口电平，0：低电平，1：高电平

适用范围：脉冲型 SMC300、SMC600 系列控制器

short smc_read_inp_pin(WORD ConnectNo,WORD axis)

功　能：读取指定轴的 INP 端口电平

参　数：ConnectNo　　指定链接号：0-7,默认值 0

　　axis　　指定轴号，取值范围：0-控制器最大轴数-1

返回值：INP 端口电平，0：低电平，1：高电平

适用范围：脉冲型 SMC300、SMC600 系列控制器

short smc_read_org_pin(WORD ConnectNo,WORD axis)

功　能：读取指定轴的 ORG 端口电平

参　数：ConnectNo　　指定链接号：0-7,默认值 0

　　axis　　指定轴号，取值范围：0-控制器最大轴数-1

返回值：ORG 端口电平，0：低电平，1：高电平

适用范围：脉冲型 SMC300、SMC600 系列控制器

short smc_read_elp_pin(WORD ConnectNo,WORD axis)

功　能：读取指定轴的正硬限位端口电平

参　数：ConnectNo　指定链接号：0-7,默认值 0

　　axis　　指定轴号，取值范围：0-控制器最大轴数-1

返回值：EL+端口电平，0：低电平，1：高电平

适用范围：脉冲型 SMC300、SMC600 系列控制器

short smc_read_eln_pin(WORD ConnectNo,WORD axis)

功　能：读取指定轴的负硬限位端口电平

参　数：ConnectNo　指定链接号：0-7,默认值 0

　　axis　　指定轴号，取值范围：0-控制器最大轴数-1

返回值：EL-端口电平，0：低电平，1：高电平

适用范围：脉冲型 SMC300、SMC600 系列控制器

short smc_read_emg_pin(WORD ConnectNo,WORD axis)

功　能：读取指定轴的急停端口电平

参　数：ConnectNo　指定链接号：0-7,默认值 0

　　axis　　指定轴号，取值范围：0-控制器最大轴数-1

返回值：EMG 端口电平，0：低电平，1：高电平

适用范围：脉冲型 SMC300、SMC600 系列控制器

## 3.17 电子凸轮指令

short smc_cam_table_unit(WORD ConnectNo,WORD MasterAxisNo,WORD SlaveAxisNo,DWORD Count,double *pMasterPos,double *pSlavePos,WORD SrcMode)

功　能：添加电子凸轮表

参　数：MasterAxisNo　　主轴轴号

　　SlaveAxisNo　　从轴轴号

　　SrcMode　　主轴位置模式：0-指令位置，1-反馈位置

　　Count　　数据个数

返回值：错误代码

适用范围：SMC300、600 系列控制器

short smc_cam_move(WORD ConnectNo,WORD AxisNo)

功　能：启动从轴电子凸轮运动

参　数：SlaveAxisNo　　从轴轴号

返回值：错误代码

适用范围：SMC300、600 系列控制器

## 3.18 手轮功能函数

short smc_handwheel_set_index( WORD ConnectNo, WORD AxisSelIndex,

　　WORD RatioSelIndex )

功　能：选择或更换手轮运动轴选、倍率档位

参　数：ConnectNo　　指定链接号：0-7,默认值 0

　　AxisSelIndex　轴选档位，取值范围：SMC606:0-5

　　RatioSelIndex　倍率档位，取值范围：0、1、2，分别对应倍率 1、10、100

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

⚠注意：此指令用于启动手轮前选择档位和手轮运动过程中更换档位。一个轴选档位可对应多个运动轴（smc_handwheel_set_axislist）。同一个轴选档位下各个运动轴倍率值是一样的，但可以修改倍率值（smc_handwheel_set_ratiolist）。

short smc_handwheel_get_index( WORD ConnectNo, WORD* AxisSelIndex,WORD* RatioSelIndex )

功 能：读取手轮运动轴选、倍率档位

参 数：ConnectNo 指定链接号：0-7,默认值 0

AxisSelIndex 返回轴选档位，取值范围：SMC606:0-5

RatioSelIndex 倍率档位，取值范围：0、1、2，分别对应倍率 1、10、100

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

short smc_handwheel_set_axislist(WORD ConnectNo, WORD AxisSelIndex,WORD AxisNum,WORD* AxisList)

功 能：设置同一轴选档位下具体运动轴

参 数：ConnectNo 指定链接号：0-7,默认值 0

AxisSelIndex 轴选档位，取值范围：0-控制器最大轴数-1

AxisNum 运动轴数，取值范围：1~控制器最大轴数-1

AxisList 轴号列表：

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

short smc_handwheel_get_axislist( WORD ConnectNo,WORD AxisSelIndex, WORD* AxisNum, WORD* AxisList)

功 能：读取同一轴选档位下具体运动轴

参 数：ConnectNo 指定链接号：0-7,默认值 0

AxisSelIndex 轴选档位，取值范围：0-控制器最大轴数-1

AxisNum 运动轴数，取值范围：1~控制器最大轴数-1

AxisList 返回轴号列表：

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

short smc_handwheel_set_ratiolist( WORD ConnectNo, WORD AxisSelIndex, WORD StartRatioIndex, WORD RatioSelNum, double* RatioList)

功 能：设置同一轴选下手轮倍率档位

参 数：ConnectNo 指定链接号：0-7,默认值 0

AxisSelIndex 轴选档位，取值范围:0-控制器最大轴数-1

StartRatioIndex 倍选起始索引值，取值范围：0-2

RatioSelNum 倍率值数量，范围：1-3

RatioList 倍率列表，取值范围：[1,100]

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

⚠注意：此指令默认参数 StartRatioIndex=0，RatioSelNum=3，数组 RatioList[3]为 3 个分别对应 1、10、100。即同一轴选下，3 种倍率值都可以有。

short smc_handwheel_get_ratiolist( WORD ConnectNo,WORD AxisSelIndex, WORD StartRatioIndex, WORD RatioSelNum,double* RatioList )

功 能：读取手轮倍率档位及对应轴的倍率值

参 数： ConnectNo 指定链接号：0-7,默认值 0

AxisSelIndex 轴选档位，取值范围： 0-控制器最大轴数-1

StartRatioIndex 倍选起始索引值，取值范围：0-2

RatioSelNum 倍率值数目

RatioList 返回倍率列表，取值范围：[1,100]

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

short smc_handwheel_set_mode( WORD ConnectNo, WORD InMode, WORD IfHardEnable )

功　能：设置手轮运动模式，硬件、还是软件模式下运动

参　数：ConnectNo　　指定链接号：0-7,默认值 0

　　InMode　　输入脉冲模式，0：脉冲+方向，1：AB 相脉冲

　　IfHardEnable 运行模式，0：软件控制，1：硬件控制

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

⚠注意：硬件模式下运动，需配置好控制器上的硬件连接线等，软件模式下则可不用。

short smc_handwheel_get_mode ( WORD ConnectNo, WORD* InMode, WORD*  IfHardEnable )

功　能：读取手轮运动模式，硬件、还是软件模式下运动

参　数：ConnectNo　　指定链接号：0-7,默认值 0

　　InMode　　返回输入脉冲模式，0：脉冲+方向，1：AB 相脉冲

　　IfHardEnable　　返回运行模式，0：软件控制，1：硬件控制

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

short smc_handwheel_move( WORD ConnectNo, WORD ForceMove );

功　能：启动手轮运动

参　数：ConnectNo　　指定链接号：0-7,默认值 0

　　ForceMove　　参数保留，固定为 0

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

short smc_handwheel_stop ( WORD ConnectNo )

功　能：停止手轮运动

参　数：ConnectNo　指定链接号：0-7,默认值 0

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

## 3.19 编码器函数

short smc_set_counter_inmode(WORD ConnectNo,WORD axis,WORD mode)

功　能：设置编码器的计数方式

参　数：ConnectNo　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

mode　　编码器的计数方式：

0：非 A/B 相(脉冲/方向)

1：1×A/B

2：2×A/B

3：4×A/B

返回值：错误代码

适用范围：全系列控制器

short smc_get_counter_inmode(WORD ConnectNo,WORD axis,WORD *mode)

功　能：读取编码器的计数方式

参　数：ConnectNo　　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数

mode　　返回编码器的计数方式

返回值：错误代码

适用范围：全系列控制器

short smc_set_encoder_unit(WORD ConnectNo,WORD axis,double encoder_value)

功　能：设置指定轴编码器脉冲计数值

参　数：ConnectNo　　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

encoder_value 编码器计数值，单位：unit

返回值：错误代码

适用范围：全系列控制器

shortsmc_get_encoder_unit(WORD ConnectNo,WORD axis,double* pos)

功　能：读取指定轴编码器脉冲计数值

参　数：ConnectNo　　指定链接号：0-7,默认值 0

　　axis　　指定轴号，取值范围：0-控制器最大轴数-1

　　pos　　返回编码器位置值，单位：unit

返回值：错误代码

适用范围：全系列控制器

short smc_set_ez_mode(WORD ConnectNo,WORD axis,WORD ez_logic,WORD ez_mode,double filter)

功　能：设置指定轴的 EZ 信号电平

参　数：ConnectNo　　指定链接号：0-7,默认值 0

　　axis　　指定轴号，取值范围：0-控制器最大轴数-1

　　ez_logic　EZ　信号有效电平，0：低有效，1：高有效

　　ez_mode　　保留参数，固定值为 0

　　filter　　保留参数，固定值为 0

返回值：错误代码

适用范围：全系列控制器

⚠注意：该函数轴号设为 255 时，所有轴 EZ 信号参数都被设置。

short smc_get_ez_mode(WORD ConnectNo,WORD axis,WORD *ez_logic,WORD *ez_mode,double *filter)

功　能：读取指定轴的 EZ 信号电平

参　数：ConnectNo　指定链接号：0-7,默认值 0

　　axis　　指定轴号，取值范围：0-控制器最大轴数-1

　　ez_logic　　返回设置的 EZ 信号有效电平

　　ez_mode　　保留参数

Filter　　保留参数

返回值：错误代码

适用范围：全系列控制器

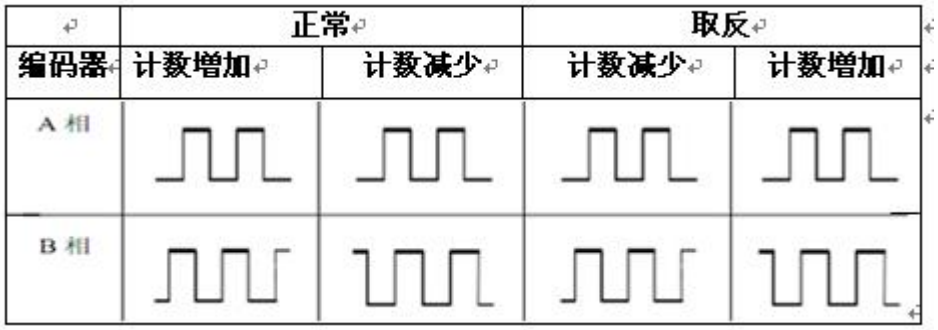
short smc_set_counter_reverse(WORD ConnectNo,WORD axis,WORD reverse);

功　能：设置 AB 相计数反相

参　数：ConnectNo　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

reverse　0-A 超前 B 增计数，1-B 超前 A 增计数




返回值：错误代码

适用范围：全系列控制器

⚠注意：该函数默认为模式 0.在编码器硬件连接线不变的情况下，我们将模式 0 变为 1，可使其编码计数为负值（取反）。

short smc_get_counter_reverse(WORD ConnectNo,WORD axis,WORD *reverse);

功　能：读取 AB 相计数反相值

参　数：ConnectNo　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

reverse　0-A 超前 B 增计数，1-B 超前 A 增计数

返回值：错误代码

适用范围：全系列控制器

## 3.20 高速位置锁存函数

short smc_set_ltc_mode(WORD ConnectNo,WORD axis,WORD ltc_logic,WORD ltc_mode,double filter)

功　能：设置指定轴的 LTC 信号

参　数：ConnectNo　　指定链接号：0-7,默认值 0

　　axis　　指定轴号，取值范围：0-控制器最大轴数-1

　　ltc_logic　　锁存信号的触发方式，0：下降沿锁存，1：上升沿锁存

　　ltc_mode　　保留值 0

　　filter　　保留参数，固定值为 0

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

short smc_get_ltc_mode(WORD ConnectNo,WORD axis,WORD *ltc_logic,WORD *ltc_mode,double *filter)

功　能：读取指定轴的 LTC 信号设置

参　数：ConnectNo　　指定链接号：0-7,默认值 0

　　axis　　指定轴号，取值范围：0-控制器最大轴数-1

　　ltc_logic　　返回 LTC 信号的触发方式，0：下降沿锁存，1：上升沿锁存

　　ltc_mode　　保留参数 0

　　filter　　保留参数

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

short smc_set_latch_mode(WORD ConnectNo,WORD axis,WORD ltc_mode,WORD latch_source,WORD triger_chunnel)

功　能：设置锁存方式

参　数：ConnectNo　　指定链接号：0-7,默认值 0

　　axis　　指定轴号，取值范围:0-控制器最大轴数-1

　　ltc_mode　　锁存方式：0：单次锁存，2：连续锁存

latch_source　　锁存源，0：指令位置计数器，1：编码器计数器

triger_chunnel　保留参数 0

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

short　smc_get_latch_mode(WORD　ConnectNo,WORD　axis,WORD*　all_enable,WORD* latch_source,WORD* triger_chunnel)

功　能：读取锁存方式

参　数：ConnectNo　　定链接号：0-7,默认值 0

axis　　指定轴号，取值范围:0-控制器最大轴数-1

ltc_mode　　返回锁存方式：0：单次锁存，2：连续锁存

latch_source　　返回锁存源，0：指令位置计数器，1：编码器计数器

triger_chunnel　保留参数 0

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

short smc_get_latch_value_unit(WORD ConnectNo,WORD axis,double* pos)

功　能：从控制器内读取锁存器的值

参　数：ConnectNo　　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

pos　　返回锁存值

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

⚠注意：当选择锁存方式为连续锁存时，用此函数成功读取锁存值后，锁存次数会减 1 并剔除该锁存值

short smc_get_latch_flag(WORD ConnectNo,WORD axis)

功　能：从控制器内读取指定轴的锁存次数

参　数：ConnectNo　　指定链接号：0-7,默认值 0

axis 指定轴号，取值范围：0-控制器最大轴数-1

返回值：有效锁存次数

适用范围：SMC300、SMC600 系列控制器

short smc_reset_latch_flag(WORD ConnectNo,WORD axis)

功 能：复位锁存器的标志位

参 数：ConnectNo 指定链接号：0-7,默认值 0

axis 指定轴号，取值范围：0-控制器最大轴数-1

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

⚠注意：当使用锁存功能前，必须先调用此函数复位锁存器的标志位

## 3.21 原点锁存函数

short smc_set_homelatch_mode(WORD ConnectNo,WORD axis,WORD enable,WORD logic,WORD source)

功 能：设置原点锁存模式

参 数：ConnectNo 指定链接号：0-7,默认值 0

axis 指定轴号，取值范围：0-控制器最大轴数-1

enable 原点锁存使能，0：禁止，1：允许

logic 触发方式，0：下降沿，1：上升沿

source 位置源选择，0：指令位置计数器，1：编码器计数器

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

short smc_get_homelatch_mode(WORD ConnectNo,WORD axis,WORD* enable,WORD* logic,WORD* source)

功 能：读取原点锁存模式设置

参 数：ConnectNo 指定链接号：0-7,默认值 0

axis 指定轴号，取值范围：0-控制器最大轴数-1

enable 返回原点锁存使能状态

logic 返回触发方式

source 返回位置源选择

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

short smc_reset_homelatch_flag(WORD ConnectNo,WORD axis)

功 能：清除原点锁存标志

参 数：ConnectNo 指定链接号：0-7,默认值 0

axis 指定轴号，取值范围：0-控制器最大轴数-1

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

long smc_get_homelatch_flag(WORD ConnectNo,WORD axis)

功 能：读取原点锁存标志

参 数：ConnectNo 指定链接号：0-7,默认值 0

axis 指定轴号，取值范围：0-控制器最大轴数

返回值：原点锁存标志，0：未锁存，1：锁存

适用范围：SMC300、SMC600 系列控制器

short smc_get_homelatch_value_unit(WORD ConnectNo,WORD axis,double* pos)

功 能：读取原点锁存值

参 数：ConnectNo 指定链接号：0-7,默认值 0

axis 指定轴号，取值范围：0-控制器最大轴数-1

pos 返回位置值

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

## 3.22 EZ 锁存函数

short smc_set_ezlatch_mode(WORD ConnectNo,WORD axis,WORD enable,WORD logic,WORD source)

功　能：设置 EZ 锁存模式

参　数：ConnectNo　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

enable　　Ez 锁存使能，0：禁止，1：允许

logic　　触发方式，0：下降沿，1：上升沿

source　　位置源选择，0：指令位置计数器，1：编码器计数器

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

short smc_get_ezlatch_mode(WORD ConnectNo,WORD axis,WORD* enable,WORD* logic,WORD* source)

功　能：读取 EZ 锁存模式设置

参　数：ConnectNo　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

enable　　返回 Ez 锁存使能状态

logic　　返回触发方式

source　　返回位置源选择

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

short smc_reset_ezlatch_flag(WORD ConnectNo,WORD axis)

功　能：清除原点锁存标志

参　数：ConnectNo　　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

long smc_get_ezlatch_flag(WORD ConnectNo,WORD axis)

功　能：读取 EZ 锁存标志

参　数：ConnectNo　　指定链接号：0-7,默认值 0

　　axis　　指定轴号，取值范围：0-控制器最大轴数

返回值：原点锁存标志，0：未锁存，1：锁存

适用范围：SMC300、SMC600 系列控制器

short smc_get_ezlatch_value_unit(WORD ConnectNo,WORD axis,double* pos)

功　能：读取 EZ 锁存值

参　数：ConnectNo　指定链接号：0-7,默认值 0

　　axis　　指定轴号，取值范围：0-控制器最大轴数-1

　　pos　　返回位置值

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

## 3.23 位置比较函数

short smc_compare_set_config(WORD ConnectNo, WORD axis,WORD enable, WORD cmp_source)

功　能：设置一维位置比较器

参　数：ConnectNo　　指定链接号：0-7,默认值 0

　　axis　　指定轴号，取值范围：0-控制器最大轴数-1

　　enable　　比较功能状态，0：禁止，1：使能

　　cmp_source　比较源，0：指令位置计数器，1：编码器计数器

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

short smc_compare_get_config(WORD ConnectNo, WORD axis,WORD* enable,WORD* cmp_source)

功　能：读取一维位置比较器设置

参　数：ConnectNo　指定链接号：0-7,默认值 0
　　axis　　指定轴号，取值范围：0-控制器最大轴数-1
　　enable　　返回比较功能状态
　　cmp_source 返回比较源
返回值：错误代码
适用范围：SMC300、SMC600 系列控制器

short smc_compare_clear_points(WORD ConnectNo,WORD axis)
功　能：清除已添加的所有一维位置比较点
参　数：ConnectNo　　指定链接号：0-7,默认值 0
　　axis　　指定轴号，取值范围：0-控制器最大轴数-1
返回值：错误代码
适用范围：SMC300、SMC600 系列控制器

short smc_compare_add_point_unit(WORD ConnectNo,WORD axis, double pos, WORD dir, WORD action, DWORD actpara)
功　能：添加一维位置比较点
参　数：ConnectNo　指定链接号：0-7,默认值 0
　　axis　　指定轴号，取值范围：0-控制器最大轴数-1
　　pos　　比较位置，单位：unit
　　dir　　比较模式，0：小于等于，1：大于等于
　　action　　比较点触发功能编号，见表 3.5
　　actpara　　比较点触发功能参数，见表 3.5
返回值：错误代码
适用范围：SMC300、SMC600 系列控制器

表 3.5 smc_compare_add_point_unit 函数 action, actpara 参数值

| action | actpara | 功能 |
|---|---|---|
| 1 | IO 号 | IO 置为高电平 |

| action | actpara | 功能 |
|---|---|---|
| 2 | IO 号 | IO 置为低电平 |
| 3 | IO 号 | 取反 IO |
| 5 | IO 号 | 输出 500us 脉冲 |
| 6 | IO 号 | 输出 1ms 脉冲 |
| 7 | IO 号 | 输出 10ms 脉冲 |
| 8 | IO 号 | 输出 100ms 脉冲 |
| 13 | 轴号 | 停止指定轴 |

short smc_compare_get_current_point_unit(WORD ConnectNo,WORD axis,double *pos)功能：读取当前比较点位置

参　数：ConnectNo　　指定链接号：0-7，默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

pos　　返回当前比较点位置，单位：unit

返回值：错误代码

适用范围：SMC300、SMC600系列控制器

short smc_compare_get_points_runned(WORD ConnectNo,WORD axis,long *pointNum);

功　能：读取已经比较过的点数量

参　数：ConnectNo　　指定链接号：0-7

axis　　指定轴号，取值范围：0-控制器最大轴数-1

pointNum　　返回当前已比较过点的数量

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

short smc_compare_get_points_remained(WORD ConnectNo,WORD axis,long *pointNum)

功　能：读取已经比较过的点数量

参　数：ConnectNo　　指定链接号：0-7

axis　　指定轴号，取值范围：0-控制器最大轴数-1

pointNum 返回当前可插入过点的数量，最大 256

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

short smc_compare_set_config_extern (WORD ConnectNo, WORD enable, WORD cmp_source)

功　能：设置二维位置比较器

参　数：ConnectNo　指定链接号：0-7,默认值 0

enable　二维位置比较功能状态，0：禁止，1：使能

cmp_source　二维位置比较源，0：指令位置计数器，1：编码器计数器

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

short smc_compare_get_config_extern(WORD ConnectNo, WORD* enable,WORD* cmp_source)

功　能：读取二维位置比较器设置

参　数：ConnectNo　指定链接号：0-7,默认值 0

Enable　返回比较功能状态

cmp_source 返回比较源

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

short smc_compare_clear_points_extern(WORD ConnectNo)

功　能：清除已添加的所有二维位置比较点

参　数：ConnectNo　指定链接号：0-7,默认值 0

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

short smc_compare_add_point_extern_unit(WORD ConnectNo,WORD* axis, double* pos, WORD* dir, WORD action, DWORD actpara)

功　能：添加二维位置比较点

参　数：ConnectNo　指定链接号：0-7,默认值 0

axis　指定控制器上的即将进行位置比较的轴列表(两个轴)

pos　二维位置比较位置列表，单位：unit

dir　比较模式列表，0：小于等于，1：大于等于

action　二维位置比较点触发功能编号，见表 3.6

actpara　二维位置比较点触发功能参数，见表 3.6

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

表 3.6 smc_compare_add_point_extern_unit 函数 action, actpara 参数值

| Action | actpara | 功能 |
|---|---|---|
| 1 | IO 号 | IO 置为高电平 |
| 2 | IO 号 | IO 置为低电平 |
| 3 | IO 号 | 取反 IO |
| 5 | IO 号 | 输出 500us 脉冲 |
| 6 | IO 号 | 输出 1ms 脉冲 |
| 7 | IO 号 | 输出 10ms 脉冲 |
| 8 | IO 号 | 输出 100ms 脉冲 |
| 13 | 轴号 | 停止指定轴 |

short smc_compare_get_current_point_extern_unit(WORD ConnectNo,double *pos)

功　能：读取当前二维位置比较点位置

参　数：ConnectNo　指定链接号：0-7,默认值 0

pos　返回当前二维位置比较点位置，单位：unit

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

short smc_compare_get_points_runned_extern(WORD ConnectNo,long *PointNum)

功　能：查询已经比较过的二维比较点个数

参　数：ConnectNo　指定链接号：0-7,默认值 0

PointNum　　返回已经比较过的二维位置比较点数

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

short smc_compare_get_points_remained_extern(WORD ConnectNo,long *PointNum)

功　能：查询可以加入的二维比较点个数

参　数：ConnectNo　指定链接号：0-7,默认值 0

PointNum　返回可以加入的二维位置比较点数

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

## 3.24 高速位置比较函数

short smc_hcmp_set_mode(WORD ConnectNo,WORD hcmp, WORD cmp_mode)

功　能：设置高速比较模式

参　数：ConnectNo　指定链接号：0-7,默认值 0

hcmp　　高速比较器，取值范围：0-1（对应硬件最后两个输出端口）

cmp_mode　比较模式：

0：禁止（默认值）

1：等于

2：小于

3：大于

4：队列，提供 127 个点比较空间，采用先添加先比较，比较完可追加比较点，也可一次性添加多个比较点

5：线性，提供起始比较点，位置增量，比较次数

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

⚠注意：1）当选择模式 1 时，只有当前位置等于比较位置时，CMP 端口才输出有效电平。

2）当选择模式 2 时，只要当前位置小于比较位置时，CMP 端口就一直保持有效电平。

3）当选择模式 3 时，只要当前位置大于比较位置时，CMP 端口就一直保持有效电平。

4）当选择模式 4 或 5 时，CMP 端口输出有效电平的时间通过 smc_hcmp_set_config 函数的 time 参数（脉冲宽度）设置。

short smc_hcmp_get_mode(WORD ConnectNo,WORD hcmp, WORD* cmp_mode)

功　能：读取高速比较模式设置

参　数：ConnectNo　指定链接号：0-7,默认值 0

hcmp　　高速比较器，取值范围：0-1（对应硬件最后两个输出端口）

cmp_mode　返回比较模式设置

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

short smc_hcmp_set_config(WORD ConnectNo,WORD hcmp,WORD axis, WORD cmp_source, WORD cmp_logic,long time)

功　能：配置高速比较器

参　数：ConnectNo　　指定链接号：0-7,默认值 0

hcmp　　高速比较器，取值范围：0-1（对应硬件最后两个输出端口）

axis　　关联轴号，取值范围：0-控制器最大轴数-1

cmp_source　比较位置源：0：指令位置计数器，1：编码器计数器

cmp_logic　有效电平：0：低电平，1：高电平

time　　脉冲宽度，单位：us，取值范围：1us~20s

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

⚠注意：该函数的 time 参数（脉冲宽度）只对队列和线性比较模式起作用 。

short smc_hcmp_get_config(WORD ConnectNo,WORD hcmp,WORD* axis, WORD* cmp_source, WORD* cmp_logic,long* time)

功　能：读取高速比较器配置

参　数：ConnectNo　　指定链接号：0-7,默认值 0

hcmp　　高速比较器，取值范围：0-1（对应硬件最后两个输出端口）

axis　　返回关联轴号设置，取值范围：0-控制器最大轴数-1

cmp_source　返回比较位置源设置

cmp_logic　返回有效电平设置

time　　返回脉冲宽度设置

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

short smc_hcmp_clear_points(WORD ConnectNo,WORD hcmp)

功　能：清除已添加的所有高速位置比较点

参　数：ConnectNo　指定链接号：0-7,默认值 0

hcmp　　高速比较器，取值范围：0-1（对应硬件最后两个输出端口）

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

short smc_hcmp_add_point_unit(WORD ConnectNo,WORD hcmp, double cmp_pos)

功　能：添加/更新高速比较位置

参　数：ConnectNo　指定链接号：0-7,默认值 0

hcmp　　高速比较器，取值范围：0-1（对应硬件最后两个输出端口）

cmp_pos　队列模式下：添加比较位置，单位：unit

线性模式下：更新起始比较位置，单位：unit

其他模式下：更新比较位置，单位：unit

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

注意：执行线性比较时该指令作为设置比较起始位置的同时也是启动命令，需在配置完比较器后调用执行。

short smc_hcmp_set_liner_unit(WORD ConnectNo,WORD hcmp,double Increment,long Count)

功　能：设置高速比较线性模式参数

参　数：ConnectNo　指定链接号：0-7,默认值 0

hcmp 高速比较器，取值范围：0-1（对应硬件最后两个输出端口）

Increment 位置增量值，单位：unit（正值表示位置递增，负值表示位置递减）

Count 比较次数，取值范围：1~65535

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

short smc_hcmp_get_liner_unit(WORD ConnectNo, WORD hcmp, double* Increment, long* Count)

功 能：读取高速比较线性模式参数设置

参 数：ConnectNo 指定链接号：0-7,默认值 0

hcmp 高速比较器，取值范围：0-1（对应硬件最后两个输出端口）

Increment 返回位置增量值设置

Count 返回比较次数设置

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

short smc_hcmp_get_current_state( WORD ConnectNo, WORD hcmp, long*remained_points, double *current_point, long *runned_points)

功 能：读取高速比较状态

参 数：ConnectNo 指定链接号：0-7,默认值 0

hcmp 高速比较器，取值范围：0-1（对应硬件最后两个输出端口）

remained_points 返回可添加比较点数

current_point 返回当前比较点位置，单位：unit

runned_points 返回已比较点数

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

short smc_write_cmp_pin( WORD ConnectNo, WORD hcmp, WORD on_off)

功 能：控制指定 CMP 端口的输出

参　数：ConnectNo　指定链接号：0-7,默认值 0

hcmp　　高速比较器，取值范围：0-1（对应硬件最后两个输出端口）

on_off　　设置 CMP 端口电平，0：低电平，1：高电平

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

⚠注意：该函数只在高速比较功能禁止的状态下起作用

short smc_read_cmp_pin(WORD ConnectNo,WORD hcmp)

功　能：读取指定 CMP 端口的电平

参　数：ConnectNo　指定链接号：0-7,默认值 0

hcmp　　高速比较器，取值范围：0-1（对应硬件最后两个输出端口）

返回值：CMP 端口电平

适用范围：SMC300、SMC600 系列控制器

## 3.25 软硬件限位函数

short smc_set_el_mode(WORD ConnectNo,WORD axis,WORD el_enable,WORD el_logic,WORD el_mode)

功　能：设置 EL 限位信号

参　数：ConnectNo　　指定链接号：0-7,默认值 0

Axis　　指定轴号，取值范围:0-控制器最大轴数-1

el_enable　　EL 信号的使能状态：

0：正负限位禁止

1：正负限位允许

2：正限位禁止、负限位允许

3：正限位允许、负限位禁止

el_logicEL 信号的有效电平：

0：正负限位低电平有效

1：正负限位高电平有效

2：正限位低有效，负限位高有效

3：正限位高有效，负限位低有效

el_modeEL 制动方式：

0：正负限位立即停止

1：正负限位减速停止

2：正限位立即停止，负限位减速停止

3：正限位减速停止，负限位立即停止

返回值：错误代码

适用范围：脉冲型全系列控制器

⚠注意：该函数轴号设为 255 时，所有轴限位信号参数都被设置。

short smc_get_el_mode(WORD ConnectNo,WORD axis,WORD *el_enable,WORD *el_logic,WORD *el_mode)

功　能：读取 EL 限位信号设置

参　数：ConnectNo　指定链接号：0-7,默认值 0

axis　指定轴号，取值范围：　0-控制器最大轴数-1

el_enable　返回设置的 EL 信号使能状态

el_logic　返回设置的 EL 信号有效电平

el_mode　返回 EL 制动方式

返回值：错误代码

适用范围：脉冲型全系列控制器

short smc_set_softlimit_unit(WORD ConnectNo,WORD axis,WORD enable, WORD source_sel,WORD SL_action, double N_limit, double P_limit)

功　能：设置软限位

参　数：ConnectNo　指定链接号：0-7,默认值 0

Axis　指定轴号，取值范围：　0-控制器最大轴数-1

Enable　使能状态，0：禁止，1：允许

source_sel　计数器选择，0：指令位置计数器，1：编码器计数器

SL_action　限位停止方式，0：立即停止，1：　减速停止

N_limit　　负限位位置，单位：unit

P_limit　　正限位位置，单位：unit

返回值：错误代码

适用范围：脉冲型全系列控制器

⚠注意：正负限位位置可为正数也可为负数，但正限位位置应大于负限位位置

short smc_get_softlimit_unit (WORD ConnectNo,WORD axis,WORD* enable, WORD* source_sel,WORD* SL_action, double * N_limit,double* P_limit)

功能：读取软限位设置

参　数：ConnectNo　　指定链接号：0-7,默认值 0

Axis　　指定轴号，取值范围： 0-控制器最大轴数-1

Enable　　返回使能状态

source_sel　　返回计数器选择

SL_action　　返回限位停止方式，0：立即停止，1： 减速停止

N_limit　　返回负限位脉冲数

P_limit　　返回正限位脉冲数

返回值：错误代码

适用范围：脉冲型全系列控制器

## 3.26 运动异常停止函数

short smc_stop(WORD ConnectNo,WORD axis,WORD stop_mode)

功　能：指定轴停止运动

参　数：ConnectNo　　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

stop_mode　　制动方式，0：减速停止，1：立即停止

返回值：错误代码

适用范围：全系列控制器

⚠注意：此函数适用于单轴、PVT 运动

short smc_stop_multicoor(WORD ConnectNo,WORD Crd,WORD stop_mode)

功　能：停止坐标系内所有轴的运动

参　数：ConnectNo　　指定链接号：0-7,默认值 0

　　Crd　　指定控制器上的坐标系号（取值范围：0~1）

　　stop_mode　　制动方式，0：减速停止，1：立即停止

返回值：错误代码

适用范围：脉冲型全系列控制器

⚠注意：此函数适用于插补运动

short smc_emg_stop(WORD ConnectNo)

功　能：紧急停止所有轴

参　数：ConnectNo　指定链接号：0-7,默认值 0

返回值：错误代码

适用范围：全系列控制器

⚠注意：此函数适用于所有运动模式

short smc_set_emg_mode(WORD ConnectNo,WORD axis,WORD enable,WORD emg_logic)

功　能：设置 EMG 急停信号

参　数：ConnectNo　指定链接号：0-7,默认值 0

　　Axis　　指定轴号，取值范围：　0-控制器最大轴数-1

　　Enable　　允许/禁止信号功能，0：禁止，1：允许

　　emg_logic　EMG 信号有效电平，0：低有效，1：高有效

返回值：错误代码

适用范围：脉冲型全系列控制器

short smc_get_emg_mode (WORD ConnectNo, WORD axis,WORD *enbale,WORD * logic)

功　能：读取 EMG 急停信号设置

参　数：ConnectNo　指定链接号：0-7,默认值 0

Axis　　指定轴号，取值范围：　0-控制器最大轴数-1

Enable　　返回 EMG 信号功能状态

Logic　　返回设置的 EMG 信号有效电平

返回值：错误代码

适用范围：脉冲型全系列控制器

short smc_set_io_dstp_mode(WORD ConnectNo,WORD axis,WORD enable,WORD logic)

功　能：设置 IO 触发减速停止电平信号；

参　数：ConnectNo　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

enable　　允许/禁止硬件信号功能，0：禁止，1：允许

logic　　外部减速停止信号有效电平，0：低有效，1：高有效

返回值：错误代码

适用范围：脉冲型全系列控制器

⚠注意：（1）减速停止信号（DSTP）的减速时间由函数 smc_set_dec_stop_time 设置。

（2）该函数轴号设为 255 时，所有轴停止信号参数都被设置。

short smc_get_io_dstp_mode(WORD ConnectNo,WORD axis,WORD *enable,WORD *logic)

功　能：读取 IO 触发减速停止电平信号

参　数：ConnectNo　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围：0-控制器最大轴数-1

enable　　返回 DSTP 硬件信号功能状态

logic　　返回设置的外部减速停止信号有效电平

返回值：错误代码

适用范围：脉冲型全系列控制器

short smc_set_dec_stop_time(WORD ConnectNo,WORD axis,double stop_time)

功　能：设置定长运动减速停止时间；

参　数：ConnectNo　　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围： 0-控制器最大轴数-1

stop_time　　减速时间，单位：s

返回值：错误代码

适用范围：脉冲型全系列控制器

⚠注意：当发生异常停止时，如：调用 smc_stop 函数、限位信号（软硬件）被触发、减速停止信号(DSTP)被触发等进行减速停止时，减速停止时间都为 smc_set_dec_stop_time 函数里设置的减速时间。

short smc_get_dec_stop_time(WORD ConnectNo,WORD axis,double *stop_time)

功　能：读取定长运动减速停止时间设置；

参　数：ConnectNo　　指定链接号：0-7,默认值 0

axis　　指定轴号，取值范围： 0-控制器最大轴数-1

stop_time　　返回设置的减速时间，单位：s

返回值：错误代码

适用范围：脉冲型全系列控制器

short smc_set_vector_dec_stop_time(WORD ConnectNo,WORD Crd,double time)功　能：设置插补运动减速停止时间；

参　数：ConnectNo　　指定链接号：0-7,默认值 0

Crd　　指定插补系，取值范围： 0-1

stop_time　　减速时间，单位：s

返回值：错误代码

适用范围：脉冲型全系列控制器

⚠注意：当发生异常停止时，如：调用 smc_stop_multicoor 函数、限位信号（软硬件）被触发、减速停止信号(DSTP)被触发等进行减速停止时，减速停止时间都为 smc_set_vector_dec_stop_time 函数里设置的减速时间。

short smc_get_vector_dec_stop_time(WORD ConnectNo,WORD Crd,double *time)功　能：读取插补运动减速停止时间设置；

参　数：ConnectNo　　指定链接号：0-7,默认值 0

Crd　指定插补系，取值范围：　0-1

stop_time　　返回设置的减速时间，单位：s

返回值：错误代码

适用范围：脉冲型全系列控制器

## 3.27 轴 IO 映射函数

short smc_set_axis_io_map(WORD ConnectNo,WORD Axis,WORD IoType,WORD MapIoType,WORD MapIoIndex,double filter_time)

功　能：设置轴 IO 映射参数

参　数：ConnectNo　　指定链接号：0-7,默认值 0

Axis　　指定轴号，取值范围：0-控制器最大轴数-1

IoType　　指定轴的 IO 信号类型：

0：正限位信号，AxisIoInMsg_PEL

1：负限位信号，AxisIoInMsg_NEL

2：原点信号，AxisIoInMsg_ORG

3：急停信号，AxisIoInMsg_EMG

4：减速停止信号，AxisIoInMsg_DSTP（保留）

5：伺服报警信号，AxisIoInMsg_ALM

6：伺服准备信号，AxisIoInMsg_RDY（保留）

7：伺服到位信号，AxisIoInMsg_INP

MapIoType 轴 IO 映射类型：

0：正限位输入端口，AxisIoInPort_PEL

1：负限位输入端口，AxisIoInPort_NEL

2：原点输入端口，AxisIoInPort_ORG

3：伺服报警输入端口，AxisIoInPort_ALM

4：伺服准备输入端口，AxisIoInPort_RDY

5：伺服到位输入端口，AxisIoInPort_INP

6：通用输入端口，AxisIoInPort_IO

MapIoIndex 轴 IO 映射索引号：

1）当轴 IO 映射类型设置为 6 时，此参数可设置为 0-17 整数，表示该映射对应的具体通用输入端口号。

2）当轴 IO 映射类型设置为 0-5 时，此参数可设置 0~7 整数，表示该映射所对应的具体轴号。

filter_time　轴 IO 信号滤波时间，单位：s

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

⚠注意：该函数可以实现对专用 IO 信号的硬件输入接口进行任意配置

short smc_get_axis_io_map(WORD ConnectNo,WORD Axis,WORD IoType,WORD* MapIoType,WORD* MapIoIndex,double* filter_time)

功　能：读取轴 IO 映射关系参数

参　数：ConnectNo　指定链接号：0-7,默认值 0

Axis　指定轴号，取值范围：0-控制器最大轴数-1

IoType　轴 IO 信号类型

MapIoType　返回轴 IO 映射类型

MapIoIndex　返回轴 IO 映射索引号

filter_time　返回轴 IO 信号滤波时间，单位：s

返回值：错误代码

适用范围：SMC300、SMC600 系列控制器

## 3.28 虚拟 IO 映射函数

short smc_set_io_map_virtual(WORD ConnectNo,WORD bitno,WORD MapIoType,WORDMapIoIndex,
double filter_time)

功　能：设置虚拟 IO 映射参数

参　数：ConnectNo　指定链接号：0-7,默认值 0

bitno　虚拟 IO 口号，取值范围：0-控制器最大输入 IO 口数-1

MapIoType　　虚拟 IO 映射类型：

0：正限位输入端口，AxisIoInPort_PEL

1：负限位输入端口，AxisIoInPort_NEL

2：原点输入端口，AxisIoInPort_ORG

3：伺服报警输入端口，AxisIoInPort_ALM

4：伺服准备输入端口，AxisIoInPort_RDY

5：伺服到位输入端口，AxisIoInPort_INP

6：通用输入端口，AxisIoInPort_IO

MapIoIndex　　虚拟 IO 映射索引号：

1）当虚拟 IO 映射类型设置为 6 时，此参数可设置为 0-17 整数，表示该映射对应的具体通用输入端口号

2）当虚拟 IO 映射类型设置为 0-5 时，此参数可设置 0~7 整数，表示该映射所对应的具体轴号

filter_time　　虚拟 IO 信号滤波时间，单位：s

返回值：错误代码

适用范围：除 SMC100 系列控制器外的控制器

⚠注意：该函数可以实现专用通用 IO 输入接口的滤波功能

short smc_get_io_map_virtual(WORD ConnectNo,WORD bitno,WORD* MapIoType,WORD* MapIoIndex,double* filter_time)

功　能：读取虚拟 IO 映射参数

参　数：ConnectNo　　指定链接号：0-7,默认值 0

bitno　　虚拟 IO 口号，取值范围：0-控制器最大输入 IO 口数-1

MapIoType　　返回虚拟 IO 映射类型

MapIoIndex　　返回虚拟 IO 映射索引号

filter_time　　返回虚拟 IO 信号滤波时间，单位：s

返回值：错误代码

适用范围：除 SMC100 系列控制器外的控制器

short smc_read_inbit_virtual(WORD ConnectNo,WORD bitno)

功　能：读取滤波后的虚拟 IO 口电平状态；

参　数：ConnectNo　　指定链接号：0-7,默认值 0

　　bitno　　虚拟 IO 口号，取值范围：0-控制器最大输入 IO 口数-1 返回值：指定的虚拟 IO 口电平：0：低电平，1：高电平

适用范围：除 SMC100 系列控制器外的控制器

⚠注意：1）此函数需要配合虚拟 IO 映射功能使用

2）smc_read_inbit_virtual 与 smc_read_inbit 函数的区别是：smc_read_inbit 函数是不经过滤波直接读取硬件端口的电平状态，smc_read_inbit_virtual 通过虚拟 IO 映射后读取相应端口滤波后的电平状态。

## 3.29 密码管理函数

short smc_write_sn(WORD ConnectNo, uint64 sn)

功　能：写入序列号

参　数：ConnectNo　指定链接号：0-7,默认值 0

　　Sn　　序列号

返回值：错误代码

适用范围：全系列控制器

short smc_read_sn(WORD ConnectNo, uint64* sn)

功　能：写入序列号

参　数：ConnectNo　指定链接号：0-7,默认值 0

　　Sn　　回读序列号

返回值：错误代码

适用范围：全系列控制器

short smc_write_password(WORD ConnectNo,const char* new_sn)

功　能：加密

参　数：ConnectNo　　指定链接号：0-7,默认值 0

new_sn　　密码，密码长度不大于 256 个字符

返回值：错误代码

适用范围：全系列控制器

short smc_check_password(WORD ConnectNo,const char* check_sn)

功　能：密码校验

参　数：ConnectNo　　指定链接号：0-7,默认值 0

check_sn　　旧密码，密码长度不大于 256 个字符

返回值：校验状态，0：失败，1：成功

适用范围：全系列控制器

⚠注意：1）密码校验失败 3 次后，无法再次校验

2）用户可以在系统软件开启时加入密码校验动作，以此对系统软件进行加 密。

short smc_enter_password(WORD ConnectNo, const char * str_pass)

功　能：登陆密码

参　数：ConnectNo　　指定链接号：0-7,默认值 0

str_pass　　密码，密码长度不大于 16 个字符

返回值：错误代码

适用范围：全系列控制器

short smc_modify_password(WORD ConnectNo, const char* spassold,
const char* spass)

功　能：修改登陆密码

参　数：ConnectNo　　指定链接号：0-7,默认值 0

Spassold　　旧密码，密码长度不大于 16 个字符

str_pass　　新密码，密码长度不大于 16 个字符

返回值：错误代码

适用范围：全系列控制器

⚠注意：smc_enter_password、smc_modify_password 主要为上传文件、参数初始化等内部

保护。

## 3.30 文件管理函数

short smc_download_file(WORD ConnectNo, const char* pfilename, const char* pfilenameinControl, WORD filetype)

功　能：下载本地文件到FLASH

参　数：ConnectNo　　指定链接号：0-7,默认值0

Pfilename　　本地文件名，包含完整路径

pfilenameinControl　　控制器内文件名

filetype　　文件类型：0-Basic,1-Gcode,2-参数，3-固件

返回值：错误代码

适用范围：全系列控制器

short smc_download_memfile(WORD ConnectNo, const char* pbuffer, uint32 buffsize, const char* pfilenameinControl, WORD filetype)

功　能：下载内存文件到FLASH

参　数：ConnectNo　　指定链接号：0-7,默认值0

Pbuffer　　内存文件缓冲区

Buffsize　　内存文件大小

pfilenameinControl　　控制器内文件名

filetype　　文件类型：0-Basic,1-Gcode,2-参数，3-固件

返回值：错误代码

short smc_upload_file (WORD ConnectNo, 0~7const char* pfilename, const char* pfilenameinControl, WORD filetype )

功　能：上传FLASH 文件到本地文件

参　数：ConnectNo　　指定链接号：0-7,默认值0

Pfilename　　本地文件名，包含完整路径

pfilenameinControl　　控制器内文件名

filetype 文件类型：0-Basic,1-Gcode,2-参数，3-固件

返回值：错误代码

适用范围：全系列控制器

short smc_upload_memfile(WORD ConnectNo, char* pbuffer, uint32 buffsize, const char* pfilenameinControl,uint32* puifilesize,WORD filetype)

功 能：上传FLASH 文件到内存文件

参 数：ConnectNo 指定链接号：0-7,默认值0

Pbuffer 内存文件缓冲区

buffsize 内存文件缓冲区大小

pfilenameinControl 控制器内文件名

puifilesize 控制器内文件大小

filetype 文件类型：0-Basic,1-Gcode,2-参数，3-固件

返回值：错误代码

short smc_download_file_to_ram(WORD ConnectNo,const char* pfilename, WORD filetype )

功 能：下载本地文件到RAM，掉电不保存

参 数：ConnectNo 指定链接号：0-7,默认值0

pfilename 本地文件名，包含完整路径

filetype 文件类型：0-Basic,1-Gcode,2-参数，3-固件

返回值：错误代码

适用范围：全系列控制器

short smc_download_memfile_to_ram(WORD ConnectNo,const char* pbuffer,uint32 buffsize,WORD filetype)

功 能：下载内存文件到RAM，掉电不保存

参 数：ConnectNo 指定链接号：0-7,默认值0

Pbuffer 内存文件缓冲区

Buffsize 内存文件大小

filetype 文件类型：0-Basic,1-Gcode,2-参数，3-固件

返回值：错误代码

适用范围：全系列控制器

short smc_get_progress(WORD ConnectNo,float* process)

功 能：文件下载进度

参 数：ConnectNo 指定链接号：0-7,默认值0

Process 返回文件下载进度值

返回值：错误代码

适用范围：全系列控制器

short smc_format_flash( WORD ConnectNo)

功 能：格式化FLASH

参 数：ConnectNo 指定链接号：0-7,默认值0

返回值：错误代码

适用范围：全系列控制器

## 3.31 寄存器操作

short smc_set_modbus_0x(WORD ConnectNo, WORD start, WORD inum, char* pdata )

功 能：写位寄存器

参 数：ConnectNo 指定链接号：0-7,默认值0

start 寄存器首地址：0~9999

inum 寄存器个数

pdata 字节数组，一个字节包括8个位寄存器。

返回值：错误代码

适用范围：全系列控制器

⚠注意：pdata，字节数组，一个字节为8个位寄存器。如pdata=5,则对应位寄存器0、1、2位值分别为1、0、1。当设定寄存器个数大于8个时，如设定9、10号寄存器位值为1、1，则设定Pdata[1]=3。

short smc_get_modbus_0x(WORD ConnectNo, WORD start, WORD inum, char* pdata )

功　能：读位寄存器

参　数：ConnectNo　　指定链接号：0-7,默认值0

start　　寄存器首地址：0~9999

inum　　寄存器个数

pdata　　返回字节数组，一个字节包括8个位寄存器。

返回值：错误代码

适用范围：全系列控制器

short smc_set_modbus_4x(WORD ConnectNo,WORD start,WORD inum,WORD* pdata)

功　能：写字寄存器

参　数：ConnectNo　　指定链接号：0-7,默认值0

start　　寄存器首地址：0~9999

inum　　寄存器个数

pdata　　字寄存器值数组

返回值：错误代码

适用范围：全系列控制器

short smc_get_modbus_4x(WORD ConnectNo,WORD start,WORD inum,WORD* pdata)

功　能：读字寄存器

参　数：ConnectNo　　指定链接号：0-7,默认值0

start　　寄存器首地址：0~9999

inum　　寄存器个数

pdata　　返回字寄存器值数组

返回值：错误代码

适用范围：全系列控制器

short smc_set_persistent_reg (WORD ConnectNo, DWORD start, DWORD inum, const char*

pdata);

功　能：设置掉电保存寄存器值

参　数：ConnectNo　　指定链接号：0-7,默认值0

start　　寄存器首地址：0~4096

inum　　寄存器个数，单次最大写入1024个字节

pdata　　寄存器值数组

返回值：错误代码

适用范围：SMC300、SMC600系列控制器

short smc_get_persistent_reg (WORD ConnectNo, DWORD start, DWORD inum, char* pdata)

功　能：回读掉电保存寄存器数值

参　数：ConnectNo　　指定链接号：0-7,默认值0

start　　寄存器首地址：0~4096

inum　　寄存器个数，单次最大读取1024个字节

pdata　　回读寄存器值数组

返回值：错误代码

适用范围：SMC300、SMC600系列控制器

## 3.32 模拟量操作函数

double　smc_get_ain(WORD ConnectNo,WORD channel)

功　能：读取模拟量电压值

参　数：ConnectNo　　指定链接号：0-7,默认值0

channel　　通道号：0,1

返回值：模拟电压：0-5V

适用范围:SMC104

short smc_set_ain_action(WORD ConnectNo,WORD channel,WORD mode,double fvoltage,WORD action,double actpara)

功　能：设置模拟量参数

参　数：ConnectNo　指定链接号：0-7,默认值0

channel　通道号：0,1

mode　触发模式：0-不触发，1-小于等于，2-大于等于

fvoltage　电压值：0-5V

action　触发动作，见下表

actpara　触发动作参数，见下表

返回值：错误代码

适用范围:SMC104

| 动作 | action | actpara |
|---|---|---|
| 单轴减速停止 | 0 | 轴号，SMC104(0-3) |
| 单轴立即停止 | 1 | 轴号，SMC104(0-3) |
| 多轴减速停止 | 2 | 轴号标志 |
| 多轴立即停止 | 3 | 轴号标志 |

short smc_get_ain_action(WORD ConnectNo,WORD channel,WORD* mode,double* fvoltage,WORD* action,double* actpara)

功　能：回读模拟量设定参数值

参　数：ConnectNo　指定链接号：0-7,默认值0

channel　通道号：0,1

mode　回读触发模式：0-不触发，1-小于等于，2-大于等于

fvoltage　回读电压值：0-5V

action　回读触发动作，见下表

actpara　回读触发动作参数，见下表

返回值：错误代码

适用范围:SMC104

| 动作 | action | actpara |
|---|---|---|
| 单轴减速停止 | 0 | 轴号，SMC104(0-3) |
| 单轴立即停止 | 1 | 轴号，SMC104(0-3) |
| 多轴减速停止 | 2 | 轴号标志 |

| | | |
|---|---|---|
| 多轴立即停止 | 3 | 轴号标志 |

short smc_set_ain_state(WORD ConnectNo,WORD channel)

功　能：置位模拟量输入触发状态

参　数：ConnectNo　指定链接号：0-7,默认值0

　　channel　　通道号：0,1

返回值：错误代码

适用范围:SMC104

short smc_get_ain_state(WORD ConnectNo,WORD channel)

功　能：读取模拟量输入触发状态

参　数：ConnectNo　指定链接号：0-7,默认值0

　　channel　　通道号：0,1

返回值：0-不触发,1-触发

适用范围:SMC104

⚠注意：目前有些控制器不具备此功能，如SMC606、SMC604等不具备此功能。SMC104则具有此功能。

## 3.33 BASIC 相关函数

short smc_write_array(WORD ConnectNo ,const char* name,uint32 startindex,int64* var,int32 num)

功　能：批量写数组值，从数组的startindex索引号开始，连续num个数组值

参　数：ConnectNo　　指定链接号：0-7,默认值0

　　name　　数组名，如array

　　startindex　　起始索引号，如启示索引号为5

　　var　　数组值列表数组，　如{array[5],array[6]}，每个值=实际值*4294967296(即实际值右移32位)

　　num　　数组值个数

返回值：错误代码

适用范围:全系列控制器

short smc_write_array_ex(WORD ConnectNo ,const char* name,uint32 startindex,double* var,int32 num)

功　能：批量写数组值，从数组的startindex索引号开始，连续num个数组值

参　数：ConnectNo　　指定链接号：0-7,默认值0

name　　数组名，如“array”

startindex　　起始索引号，如起始索引号为5

var　　数组值数组，如{array[5],array[6]}

num　　数组值个数,如数组值个数为2

返回值：错误代码

适用范围:全系列控制器

short smc_read_array( WORD ConnectNo , const char* name, uint32 index, int64* var, int32 *num )

功　能：按索引读数组值

参　数：ConnectNo　　指定链接号：0-7,默认值0

name　　返回数组名，多个之间加逗号，如“array0,array1”

index　　索引号，如索引号为1

var　　返回索引号对应数组值数组，如{array0[1],array1[1]},实际值=每个值/4294967296(或每个值左移32位)

num　　，如数组值个数为2

返回值：错误代码

适用范围:全系列控制器

short smc_read_array_ex( WORD ConnectNo , const char* name, uint32 index, double* var, int32 *num )

功　能：按索引读数组值

参　数：ConnectNo　　指定链接号：0-7,默认值0

name 返回数组名，多个之间加逗号，如“array0,array1”
index 索引号，如索引号为1
var 返回索引号对应数组值数组，如{array0[1],array1[1]}
num 返回数组值个数，如数组值个数为2

返回值：错误代码

适用范围:全系列控制器

short smc_modify_array(WORD ConnectNo ,const char* name,uint32 index，int64* var,int32 num)

功　能：按索引修改数组值

参　数：name 数组名，多个之间加逗号，如“array0,array1”
index 索引号，如索引号为1
var 索引号对应数组值数组,如{array0[1],array1[1]}，每个值=实际值*4294967296(或实际值右移32位)
num 数组个数，如数组个数为2

返回值：错误代码

适用范围:全系列控制器

short smc_modify_array_ex(WORD ConnectNo ,const char* name,uint32 index，double* var,int32 num)

功　能：按索引修改数组值

参　数：ConnectNo 指定链接号：0-7,默认值0
name 数组名，多个之间加逗号，如“array0,array1”
index 索引号，如索引号为1
var 索引号对应数组值数组,如{array0[1],array1[1]}
num 数组个数，如数组个数为2

返回值：错误代码

适用范围:全系列控制器

short smc_read_var(WORD ConnectNo,const char* name,int64* var,int32 *num)

功　能：读变量值

参　数：ConnectNo　指定链接号：0-7,默认值0

name　　变量名，多个之间加逗号，如“var0,var1”

var　　返回变量值数组，如{var0,var1},实际值=每个值/4294967296(或每个值左移32位)

num　　返回变量个数，如变量个数为2

返回值：错误代码

适用范围:全系列控制器

short smc_read_var_ex(WORD ConnectNo,const char* name,double* var,int32 *num)

功　能：读变量值

参　数：ConnectNo　指定链接号：0-7,默认值0

name　　变量名，多个之间加逗号，如“var0,var1”

var　　返回变量值数组，如{var0,var1}

num　　返回变量个数，如变量个数为2

返回值：错误代码

适用范围:全系列控制器

short smc_modify_var(WORD ConnectNo,const char* name,int64* var,int32 num)

功　能：修改变量值

参　数：ConnectNo　指定链接号：0-7,默认值0

name　　变量名，多个之间加逗号

var　　变量值

num　　变量个数

返回值：错误代码

适用范围:全系列控制器

short smc_modify_var_ex(WORD ConnectNo,const char* name,double* var,int32 num)

功　能：修改变量值

参　数：ConnectNo　　指定链接号：0-7,默认值0

　　name　　变量名，多个之间加逗号，如“var0,var1”

　　var　　变量值数组，如{var0,var1}

　　num　　变量个数，如变量个数为2

返回值：错误代码

适用范围:全系列控制器

short smc_get_stringtype(WORD ConnectNo,const char* varstring,int32* m_Type,int32* num)

功　能：读取变量类型

参　数：ConnectNo　指定链接号：0-7,默认值0

　　varstring　变量名

　　m_Typ　　变量类型，详见string_types 枚举

　　num　　数组变量长度

返回值：错误代码

适用范围:全系列控制器

```
enum string_types
{
STRING_USERSUB = 1,//SUB
STRING_VARIABLE = 2,//全局变量
STRING_ARRAY = 3,//全局数组
STRING_PARA = 4,//参数
STRING_CMD = 5,//命令
STRING_KEYWORD = 6, //保留关键字
STRING_VARIABLE_LOCAL = 7,//局部变量
STRING_ARRAY_LOCAL = 8,//局部数组
STRING_UNKOWN = 10,//未知变量
```

};

short smc_basic_delete_file(WORD ConnectNo)

功　能：删除BASIC程序

参　数：ConnectNo　指定链接号：0-7,默认值0

返回值：错误代码

适用范围:全系列控制器

short smc_basic_run(WORD ConnectNo)

功　能：运行

参　数：ConnectNo　指定链接号：0-7,默认值0

返回值：错误代码

适用范围:全系列控制器

short smc_basic_stop(WORD ConnectNo)

功　能：停止

参　数：ConnectNo　指定链接号：0-7,默认值0

返回值：错误代码

适用范围:全系列控制器

short smc_basic_pause(WORD ConnectNo)

功　能：暂停

参　数：ConnectNo　　链接号：0~7

返回值：错误代码

short smc_basic_step_run(WORD ConnectNo)

功　能：单步运行

参　数：ConnectNo　指定链接号：0-7,默认值0

返回值：错误代码

适用范围:全系列控制器

short smc_basic_step_over(WORD ConnectNo)

功　能：运行到下一断点

参　数：ConnectNo　指定链接号：0-7,默认值0

返回值：错误代码

适用范围:全系列控制器

short smc_basic_continue_run(WORD ConnectNo)

功　能：继续运行

参　数：ConnectNo　指定链接号：0-7,默认值0

返回值：错误代码

适用范围:全系列控制器

short smc_basic_state(WORD ConnectNo,WORD* State)

功　能：当前状态

参　数：ConnectNo　　指定链接号：0-7,默认值0

　　State　　运行状态：1-运行，2-暂停，3-停止，100-异常

返回值：错误代码

适用范围:全系列控制器

short smc_basic_current_line(WORD ConnectNo,uint32* line)

功　能：当前执行行

参　数：ConnectNo　　指定链接号：0-7,默认值0

　　line　　运行行号

返回值：错误代码

适用范围:全系列控制器

short smc_basic_break_info(WORD ConnectNo,uint32* line,uint32 linenum )

功　能：断点信息

参　数：ConnectNo　　指定链接号：0-7,默认值0
　　line　　断点行列表数组
　　linenum　　断点行数
返回值：错误代码
适用范围:全系列控制器

short smc_basic_message(WORD ConnectNo,char * pbuff,uint32 uimax,uint32 *puiread)
功　能：读取输出信息
参　数：ConnectNo　　指定链接号：0-7,默认值0
　　pbuff　　输出信息缓冲区
　　uimax　　缓冲区大小
　　puiread　　返回输出信息大小
返回值：错误代码
适用范围:全系列控制器

short smc_basic_command(WORD ConnectNo,const char* pszCommand,char*
psResponse,uint32 uiResponseLength)
功　能：在线命令
参　数：ConnectNo　　指定链接号：0-7,默认值0
　　pszCommand　　命令字符串
　　psResponse　　返回字符串
　　uiResponseLength　　返回字符串长度
返回值：错误代码
适用范围:全系列控制器

## 3.34 G 代码相关函数

short smc_gcode_check_file(WORD ConnectNo,const char* pfilenameinControl,uint8
*pbIfExist,uint32 *pFileSize)
功　能：检查文件是否存在

参　数：ConnectNo　　指定链接号：0-7,默认值0
　　pfilenameinControl　　控制器文件名
　　pbIfExist　　是否存在：0-不存在，1-存在
　　pFileSize　　文件大小，不存在返回（uint32)-1
返回值：错误代码
适用范围:SMC300、SMC600系列控制器

short smc_gcode_delete_file(WORD ConnectNo,const char* pfilenameinControl)
功　能：删除文件
参　数：ConnectNo　　指定链接号：0-7,默认值0
　　pfilenameinControl　　控制器文件名
返回值：错误代码
适用范围:SMC300、SMC600系列控制器

short smc_gcode_clear_file(WORD ConnectNo)
功　能：删除所有文件
参　数：ConnectNo　指定链接号：0-7,默认值0
返回值：错误代码
适用范围:SMC300、SMC600系列控制器

short smc_gcode_get_first_file(WORD ConnectNo,char* pfilenameinControl,uint32* pFileSize)
功　能：读取第一个文件名
参　数：ConnectNo　　指定链接号：0-7,默认值0
　　pfilenameinControl　　控制器文件名
　　pFileSize　　文件大小，不存在返回（uint32)-1
返回值：错误代码
适用范围:SMC300、SMC600系列控制器

short smc_gcode_get_next_file(WORD ConnectNo,char* pfilenameinControl,uint32* pFileSize )

功　能：读取下一个文件名

参　数：ConnectNo　　指定链接号：0-7,默认值0

　　pfilenameinControl　控制器文件名

　　pFileSize　　文件大小，不存在返回（uint32)-1

返回值：错误代码

适用范围:SMC300、SMC600系列控制器

short smc_gcode_start(WORD ConnectNo)

功　能：启动

参　数：ConnectNo　指定链接号：0-7,默认值0

返回值：错误代码

适用范围:SMC300、SMC600系列控制器

short smc_gcode_stop(WORD ConnectNo);

功　能：停止

参　数：ConnectNo　指定链接号：0-7,默认值0

返回值：错误代码

适用范围:SMC300、SMC600系列控制器

short smc_gcode_pause(WORD ConnectNo)

功　能：暂停

参　数：ConnectNo　指定链接号：0-7,默认值0

返回值：错误代码

适用范围:SMC300、SMC600系列控制器

short smc_gcode_state(WORD ConnectNo,WORD* State)

功　能：读取当前状态

参　数：ConnectNo　　指定链接号：0-7,默认值0

　　State　　当前G代码运行状态:1-运行，2-暂停，3-停止，4-异常

返回值：错误代码

适用范围:SMC300、SMC600系列控制器

short smc_gcode_set_current_file(WORD ConnectNo,const char* pfilenameinControl)

功　能：设置当前文件

参　数：ConnectNo　　指定链接号：0-7,默认值0

　　const char* pfilenameinControl　控制器文件名

返回值：错误代码

适用范围:SMC300、SMC600系列控制器

short smc_gcode_get_current_file(WORD ConnectNo,char* pfilenameinControl,WORD *fileid)

功　能：读取当前文件名

参　数：WORD ConnectNo　　链接号：0~7

　　pfilenameinControl　　返回控制器文件名

　　fileid　　返回当前文件号

返回值：错误代码

适用范围:SMC300、SMC600系列控制器

short smc_gcode_current_line(WORD ConnectNo,uint32* line,char* pCurLine)

功　能：读取当前运行行

参　数：ConnectNo　指定链接号：0-7,默认值0

　　uint32* line　　行号

　　char* pCurLine　　行字符串

返回值：错误代码

适用范围:SMC300、SMC600系列控制器

short smc_gcode_get_file_profile(WORD ConnectNo,uint32* maxfilenum,uint32* maxfilesize,uint32* savedfilenum);

功　能：读取G文件属性

参　数：ConnectNo　　指定链接号：0-7,默认值0

uint32* maxfilenum　　最大文件数量

uint32* maxfilesize　　文件容量最大值

uint32* savedfilenum　　已保存文件数量

返回值：错误代码

适用范围:SMC300、SMC600 系列控制器

## 3.35 总线相关函数

### 3.35.1 总线配置函数

short nmcs_reset_canopen(WORD ConnectNo)

功能：复位 CANopen 总线

参数：ConnectNo　　指定链接号：0-7,默认值 0

short nmcs_stop_etc(WORD ConnectNo,WORD* ETCState)

功　能：停止EtherCAT总线运行

参　数：ConnectNo　　指定链接号：0-7,默认值 0

ETCState　　0：停止 EtherCAT 总线成功，1：停止 EtherCAT 总线失败

返回值：错误代码

Short nmcs_axis_io_status(WORD ConnectNo,WORD axis)

功能：获取轴 IO 信息

参数：ConnectNo　　指定链接号:0-7,默认值 0

Axis　　轴号

Short nmcs_get_LostHeartbeat_Nodes(WORD ConnectNo,WORD PortNo,WORD* NodeID,WORD*

NodeNum)

功能：获取心跳报文信息，支持 CANopen、EtherCat

参数： ConnectNo 指定链接号：0-7，默认值 0

PortNo 端口号，固定为 2(其中 0、1 为 CANopen 端口)

NodeID 节点号

NodeNum 节点个数

Short nmcs_get_EmergeneyMessege_Nodes(WORD ConnectNo,WORD PortNo,DWORD* NodeMsg,WORD* MsgNum)

功能：获取紧急报文信息，支持 CANopen、EtherCAT

参数： ConnectNo 指定链接号：0-7，默认值 0
PortNo 端口号，固定为 2(其中 0、1 为 CANopen 端口)
NodeMsg
MsgNum

short nmcs_set_node_od(WORD ConnectNo,WORD PortNum,WORD NodeNum, int Index,int SubIndex,int ValLength,int Value)

功 能：设置从站对象字典

参 数：ConnectNo 指定链接号：0-7,默认值 0

NodeNum 节点号

PortNum 端口号（0-3）

Index 索引

SubIndex 子索引

ValLength 值长度（该参数只有三个值：8、16 和 32）

Value 从站值

返回值：错误代码

适用范围:全系列控制器

short nmcs_get_node_od(WORD ConnectNo,WORD PortNum,WORD NodeNum, int Index,int SubIndex,int* ValLength,int* Value)

功 能：获取从站对象字典

参　数：ConnectNo　　指定链接号：0-7,默认值 0
　　PortNum　　端口号（0-3）
　　NodeNum　　节点号
　　Index　　索引
　　SubIndex　　子索引
　　ValLength　　值长度（该参数只有三个值：8、16 和 32）
　　Value　　返回从站值

返回值：错误代码

适用范围:全系列控制器

short nmcs_SendNmtCommand(WORD ConnectNo,WORD PortNum,WORD NodeID,WORD NmtCommand)

功　能：发送 NMT 管理报文

参　数：ConnectNo　　指定链接号：0-7,默认值 0
　　PortNum　　端口号：
　　0（CANBUS_0）
　　1（CANBUS_1）
　　NodeID　　节点号
　　NmtCommand　　NmtCommandNMT 命令，该参数有五个值：
　　0x01-启动远程节点
　　0x02-停止远程节点
　　0x80-远程节点进入预操作模式
　　0x81-复位远程节点
　　0x82-远程节点复位通讯

返回值：错误代码

适用范围:全系列控制器

short nmcs_get_cycletime(WORD ConnectNo,WORD PortNum,DWORD* CycleTime)

功　能：读取EtherCAT总线循环周期

参　数：ConnectNo　　指定链接号：0-7,默认值 0

PortNum　　EtherCAT 端口号，固定为 2(其中 0、1 为 CANopen 端口)

CycleTime　　EtherCAT 总线循环周期，单位：us

返回值：错误代码

Short nmcs_write_rxpdo_extra(WORD ConnectNo,WORD PortNo,WORD address,WORD DataLen,DWORD Value)

功能：写扩展 rxpdo

参数： ConnectNo 链接号：0-7 号，默认值 0

PortNum：端口号，0,1 表示 CANOpen，2，3 表示 EtherCAT 端口

address：扩展 PDO 的首地址

DataLen：数据长度，按 16bit 计算，最大值为 2（表示 32bit 数据）

Value：数据值

Short nmcs_read_rxpdo_extra(WORD ConnectNo,WORD PortNo,WORD address,WORD DataLen,DWORD* Value)

功能：读扩展 rxpdo

参数： ConnectNo 链接号：0-7 号，默认值 0

PortNum：端口号，0,1 表示 CANOpen，2，3 表示 EtherCAT 端口

address：扩展 PDO 的首地址

DataLen：数据长度，按 16bit 计算，最大值为 2（表示 32bit 数据）

Value：数据值

Short nmcs_read_txpdo_extra(WORD ConnectNo,WORD PortNo,WORD address,WORD DataLen,DWORD* Value)

功能：读扩展 txpdo

参数： ConnectNo 链接号：0-7 号，默认值 0

PortNum：端口号，0,1 表示 CANOpen，2，3 表示 EtherCAT 端口

address：扩展 PDO 的首地址

DataLen：数据长度，按 16bit 计算，最大值为 2（表示 32bit 数据）

Value：数据值

### 3.35.2 总线 IO 及轴控制函数

short nmcs_get_axis_type(WORD ConnectNo,WORD axis, WORD* Axis_Type)

功 能：读取轴类型

参 数：ConnectNo 指定链接号：0-7,默认值 0

axis 轴号

Axis_Type 轴的类型，0：虚拟轴，1：EtherCAT 轴，2：CANopen 轴，3：脉冲轴，4：未知类型轴

返回值：错误代码

short nmcs_set_axis_enable(WORD ConnectNo,WORD axis)

功 能：设置 EtherCAT 总线驱动器使能

参 数：ConnectNo 指定链接号：0-7,默认值 0

axis EtherCAT 总线轴的轴号，255 表示使能所有 EtherCAT 轴

返回值：错误代码

short nmcs_set_axis_disable(WORD ConnectNo,WORD axis)

功 能：设置 EtherCAT 总线驱动器禁止使能

参 数：ConnectNo 指定链接号：0-7,默认值 0

axis EtherCAT 总线轴的轴号，255 表示失能所有 EtherCAT 轴

返回值：错误代码

short nmcs_syn_move(WORD ConnectNo,WORD AxisNum,WORD* AxisList,int* Position,WORD* PosiMode)

功 能：同步运动

参 数：ConnectNo 指定链接号：0-7,默认值 0

AxisNum 轴数(值范围 1-32，最大 32 个轴)

| | | |
|---|---|---|
| | AxisList | 轴列表 |
| | Position | 目标位置列表 |
| | PosiMode | 位置模式列表：<br>0-相对模式<br>1-绝对模式 |

返回值：错误代码

适用范围:全系列控制器

**注　意：**该运动的运动速度由 smc_set_profile_unit 设置

short nmcs_get_axis_state_machine(WORD ConnectNo,WORD axis, WORD* Axis_StateMachine);

功　能：读取EtherCAT总线轴状态机

| | | |
|---|---|---|
| 参　数： | ConnectNo | 指定链接号：0-7,默认值 0 |
| | axis | EtherCAT 总线轴轴号 |
| | Axis_StateMachine | EtherCAT 总线轴状态机<br>0：未启动状态<br>1：启动禁止状态<br>2：准备启动状态<br>3：启动状态<br>4：操作使能状态<br>5：停止状态<br>6：错误触发状态<br>7：错误状态 |

返回值：错误代码

short nmcs_get_axis_controlmode(WORD ConnectNo,WORD axis,long* contrlmode);

功　能：读取EtherCAT总线轴控制模式

| | | |
|---|---|---|
| 参　数： | ConnectNo | 指定链接号：0-7,默认值 0 |
| | axis | EtherCAT 总线轴轴号 |

contrlmode EtherCAT 总线轴控制模式，6：回零模式，8：CSP 模式

返回值：错误代码

short nmcs_set_home_profile(WORD ConnectNo,WORD PortNum,WORD axis,WORD home_mode,double High_Vel,double Low_Vel,double Tacc,double Tdec,double offsetpos)

功　能：设置 EtherCAT 总线轴回零参数

参　数：ConnectNo 指定链接号：0-7,默认值 0

PortNum EtherCAT 总线端口号，固定为 2(其中 0、1 为 CANopen 端口)

axis EtherCAT 总线轴轴号

home_mode EtherCAT 总线轴回零模式

High_Vel EtherCAT 总线轴回零高速

Low_Vel EtherCAT 总线轴回零低速

Tacc EtherCAT 总线轴回零加速时间

Tdec EtherCAT 总线轴回零减速时间

offsetpos EtherCAT 总线轴回零偏置

返回值：错误代码

short nmcs_get_total_axes(WORD ConnectNo,DWORD* TotalAxis)

功　能：读取 EtherCAT 总线轴和虚拟轴轴数

参　数：ConnectNo 指定链接号：0-7,默认值 0

TotalAxis EtherCAT 总线轴和虚拟轴轴数

返回值：错误代码

short nmcs_get_total_adcnum(WORD ConnectNo,WORD* TotalIn,WORD* TotalOut)

功　能：读取 EtherCAT 总线 AD/DA 输入输出口数

参　数：ConnectNo 指定链接号：0-7,默认值 0

TotalIn EtherCAT 总线 AD 输入数

TotalOut EtherCAT 总线 DA 输出数

返回值：错误代码

short nmcs_get_total_ionum(WORD ConnectNo,WORD *TotalIn,WORD *TotalOut)

功　能：读取 EtherCAT 总线 IO 输入输出口数

参　数：ConnectNo　　指定链接号：0-7,默认值 0

　　TotalIn　　EtherCAT 总线 IO 输入数

　　TotalOut　　EtherCAT 总线 IO 输出数

返回值：错误代码

short nmcs_get_total_adcnum(WORD CardNo,WORD *TotalIn,WORD *TotalOut)

功　能：获取 EtherCAT 总线模拟量 I/O 点数

参　数：CardNo　　控制卡卡号

　　TotalIn　　返回模拟量输入口点数

　　TotalOut　　返回模拟量输出口点数

返回值：错误代码

适用范围:全系列控制器

short nmcs_get_total_slaves(WORD ConnectNo,WORD PortNum,WORD* TotalSlaves)

功　能：获取 EtherCAT 从站总数

参　数：ConnectNo　　指定链接号：0-7,默认值 0

　　PortNum　　EtherCAT 端口号，固定为 2(其中 0、1 为 CANopen 端口)

　　TotalSlaves　　EtherCAT 从站总数

返回值：错误代码

short nmcs_get_axis_io_out(WORD ConnectNo,WORD axis,DWORD* iostate)

功　能：获取 EtherCAT 轴数字量输出 IO 状态

参　数：ConnectNo　　指定链接号：0-7,默认值 0

　　axis　　EtherCAT 轴号

　　iostate　　EtherCAT 轴数字量输出 IO 状态

返回值：错误代码

short nmcs_set_axis_io_out(WORD ConnectNo,WORD axis,DWORD iostate)

功　能：设置 EtherCAT 轴数字量输出 IO 状态

参　数：ConnectNo　　指定链接号：0-7,默认值 0

　　axis　　EtherCAT 轴号

　　iostate　　EtherCAT 轴数字量输出 IO 状态

返回值：错误代码

short nmcs_get_axis_io_in(WORD ConnectNo,WORD axis,DWORD* iostate)

功　能：获取 EtherCAT 轴数字量输入 IO 状态

参　数：ConnectNo　　指定链接号：0-7,默认值 0

　　axis　　EtherCAT 轴号

　　iostate　　EtherCAT 轴数字量输入 IO 状态

返回值：错误代码

short nmcs_write_outbit(WORD ConnectNo,WORD NoteID,WORD IoBit,WORD IoValue)

功　能：设置指定控制器或扩展模块的某个输出端口的电平

参　数：ConnectNo　　指定链接号：0-7,默认值 0

　　NodeID　　节点号

　　IoBit　　输出端口号

　　IoValue　　输出电平，　0：　低电平，　1：　高电平

返回值：错误代码

适用范围:全系列控制器

short nmcs_read_outbit(WORD ConnectNo,WORD NoteID,WORD IoBit,WORD *IoValue)

功　能：读取指定控制器或扩展模块的某个输出端口的电平

参　数：ConnectNo　　指定链接号：0-7,默认值 0

　　NodeID　　节点号

　　IoBit　　输出端口号

　　IoValue　　返回输出电平，　0：　低电平，　1：　高电平

返回值：错误代码

适用范围:全系列控制器

short nmcs_read_inbit(WORD ConnectNo,WORD NoteID,WORD IoBit,DWORD *IoValue)

功　能：读取指定控制器或扩展模块的某个输入端口的电平

参　数：ConnectNo　　指定链接号：0-7,默认值 0

NodeID　　节点号

IoBit　　输入端口号

IoValue　　输入端口电平，　0：　低电平，　1：　高电平

返回值：错误代码

适用范围:全系列控制器

short nmcs_write_outport(WORD ConnectNo,WORD NodeID,WORD PortNo,WORD IoValue)

功　能：设置指定 IO 组号的全部输出口的电平

参　数：ConnectNo　　指定链接号：0-7,默认值 0

NodeID　　节点号

PortNo　　IO 组号，最小值 0

IoValue　　输入端口电平，　0：　低电平，　1：　高电平

返回值：错误代码

适用范围:全系列控制器

**注　意：**IO 组号最小值为 0，每 32 点 IO 一组；IO 数满 32，组号加 1

short nmcs_read_outport(WORD ConnectNo,WORD NodeID,WORD PortNo,DWORD *IoValue)

功　能：读取指定 IO 组号的全部输出口的电平

参　数：ConnectNo　　指定链接号：0-7,默认值 0

NodeID　　节点号

PortNo　　IO 组号，最小值 0

IoValue　　输入端口电平，　0：　低电平，　1：　高电平

返回值：错误代码

适用范围:全系列控制器

**注　意：**IO 组号最小值为 0，每 32 点 IO 一组；IO 数满 32，组号加 1

short nmcs_read_inport(WORD ConnectNo,WORD NodeID,WORD PortNo,DWORD *IoValue)

功　能：读取指定 IO 组号的全部输入口的电平

参　数：ConnectNo　　指定链接号：0-7,默认值 0

NodeID　　节点号

PortNo　　IO 组号，最小值 0

IoValue　　输入端口电平，　0：　低电平，　1：　高电平

返回值：错误代码

适用范围:全系列控制器

**注　意：**IO 组号最小值为 0，每 32 点 IO 一组；IO 数满 32，组号加 1

### 3.35.3　总线错误代码函数

Short nmcs_set_alarm_clear(WORD ConnectNo,WORD PortNo,WORD NodeNo)

功能：清除报警信号,支持 CANopen、EtherCAT

参数: ConnectNo　　指定链接号：0-7,默认值 0

PortNo　　EtherCAT 总线端口号，固定为 2(其中 0、1 为 CANopen 端口)

NodeNo　　默认为 0

short　nmcs_get_errcode(WORD ConnectNo,WORD PortNum,WORD* errcode)

功　能：读取 EtherCAT 总线状态

参　数：ConnectNo　指定链接号：0-7,默认值 0

PortNum　EtherCAT 总线端口号，固定为 2(其中 0、1 为 CANopen 端口)

errcode　EtherCAT 总线状态，0 表示正常

返回值：错误代码

Short　nmcs_clear_errcode(WORD ConnectNo,WORD PortNo)

功能：清除总线错误

参数：ConnectNo　　指定链接号：0-7，默认值 0

PortNum　　EtherCAT 总线端口号，固定为 2(其中 0、1 为 CANopen 端口)

返回值：错误代码

short nmcs_get_card_errcode(WORD ConnectNo,WORD *Errcode);

功　能：获取指定链接号总线错误代码

参　数：ConnectNo　　指定链接号：0-7,默认值 0

　　Errcode　　总线错误代码

返回值：错误代码

适用范围:全系列控制器

short nmcs_clear_card_errcode(WORD ConnectNo);

功　能：清除指定链接号总线错误代码

参　数：ConnectNo　　指定链接号：0-7,默认值 0

返回值：错误代码

适用范围:全系列控制器

short　nmcs_get_axis_errcode(WORD ConnectNo,WORD axis,WORD *Errcode);

功　能：获取指定轴总线错误代码

参　数：ConnectNo　　指定链接号：0-7,默认值 0

　　axis　　指定轴号，取值范围：0-控制器最大轴数-1

　　Errcode　　总线错误代码

返回值：错误代码

适用范围:全系列控制器

short nmcs_clear_axis_errcode(WORD ConnectNo,WORD axis);

功　能：清除指定轴号总线错误代码

参　数：ConnectNo　　指定链接号：0-7,默认值 0

　　axis　　指定轴号，取值范围：0-控制器最大轴数-1

返回值：错误代码

适用范围:全系列控制器

# 表格 1：API 函数一览表

下表列出了所使用的 PC 端函数及说明。

| 功能 | 名称 | 功能说明 |
|---|---|---|
| 控制器配置函数 | smc_board_init | 控制器链接初始化函数，分配系统资源 |
| | smc_board_init_ex | 控制器高级链接初始化函数，分配系统资源 |
| | smc_board_close | 控制器关闭函数，释放系统资源 |
| | smc_set_connect_timeout | 网络链接超时时间 |
| | smc_get_release_version | 读取发布版本号 |
| | smc_get_card_version | 获取控制器硬件版本号 |
| | smc_get_card_soft_version | 获取控制器固件版本号 |
| | smc_get_card_lib_version | 获取控制器动态库文件版本号 |
| | smc_get_total_axes | 获取当前控制器的轴数 |
| | smc_set_debug_mode | 函数调用打印输出设置 |
| | smc_get_debug_mode | 读取函数调用打印输出设置 |
| | smc_format_flash | 格式化 FLASH |
| | smc_set_ipaddr | 设置控制器新 IP 地址 |
| | smc_get_ipaddr | 读取控制器 IP 地址 |
| | smc_set_com | 设置控制器 COM 口参数 |
| | smc_get_com | 读取控制器 COM 口参数 |
| 序列号 | smc_write_sn | 写入序列号 |
| | smc_read_sn | 读取序列号 |
| 脉冲模式设置 | smc_set_pulse_outmode | 设置指定轴的脉冲输出模式 |
| | smc_get_pulse_outmode | 读取指定轴的脉冲输出模式设置 |
| 脉冲当量设置 | smc_set_equiv | 设置脉冲当量值 |
| | smc_get_equiv | 返回脉冲当量值设置 |
| 反向间 | smc_set_backlash_unit | 设置反向间隙值 |

| | | |
|---|---|---|
| 隙设置 | smc_get_backlash_unit | 读取反向间隙值设置 |
| 状态监控 | smc_check_done | 检测指定轴的运动状态 |
| | smc_check_done_multicoor | 检测坐标系的运动状态 |
| | smc_axis_io_enable_status | 读取指定轴特殊信号的使能状态 |
| | smc_axis_io_status | 读取指定轴有关运动信号的状态 |
| | smc_get_axis_run_mode | 读取轴运动模式 |
| | smc_set_position_unit | 设置当前指令位置计数器值 |
| | smc_get_position_unit | 读取当前指令位置计数器值 |
| | smc_read_current_speed_unit | 读取轴当前速度 |
| | smc_get_stop_reason | 读取轴停止原因 |
| | smc_clear_stop_reason | 清除轴停止原因 |
| | smc_get_target_position_unit | 读取当前目标位置 |
| | smc_set_workpos_unit | 设置当前工件原点 |
| | smc_get_workpos_unit | 读取当前工件原点 |
| 单轴速度参数 | smc_set_profile_unit | 设置单轴运动速度曲线 |
| | smc_get_profile_unit | 读取单轴运动速度曲线 |
| | smc_set_s_profile | 设置单轴速度曲线 S 段参数值 |
| | smc_get_s_profile | 读取单轴速度曲线 S 段参数值 |
| 单轴运动 | smc_pmove_unit | 定长运动 |
| | smc_vmove | 指定轴连续运动 |
| | smc_reset_target_position_unit | 在线改变指定轴的当前目标位置 |
| | smc_update_target_position_unit | 强制改变指定轴的当前目标位置 |
| | smc_change_speed_unit | 在线改变指定轴的当前运动速度 |
| 回原点运动 | smc_set_home_pin_logic | 设置 ORG 原点信号 |
| | smc_get_home_pin_logic | 读取 ORG 原点信号设置 |
| | smc_set_homemode | 设置回原点模式 |
| | smc_get_homemode | 读取回原点模式 |
| | smc_set_ez_count | 设置回零 EZ 个数 |
| | smc_get_ez_count | 回读回零 EZ 个数 |

| | | |
|---|---|---|
| | smc_set_home_position_unit | 设置回原点完成后的偏移位置值 |
| | smc_get_home_position_unit | 回读回原点完成后的偏移位置值 |
| | smc_set_home_profile_unit | 设置回原点速度参数 |
| | smc_get_home_profile_unit | 读取回原点速度参数 |
| | smc_set_el_home | 限位当原点切换函数 |
| | smc_home_move | 回原点运动 |
| | smc_get_home_result | 读取回原点运动状态 |
| PVT 运动函数 | smc_ptt_table_unit | 向指定数据表传送数据，采用 PTT 模式 |
| | smc_pts_table_unit | 向指定数据表传送数据，采用 PTS 模式 |
| | smc_pvt_table_unit | 向指定数据表传送数据，采用 PVT 模式 |
| | smc_pvts_table_unit | 向指定数据表传送数据，采用 PVTS 模式 |
| | smc_pvt_move | 启动 PVT 运动 |
| 电子凸轮 | smc_cam_table_unit | 设置凸轮表 |
| | Smc_cam_move | 启动凸轮运动 |
| 插补运动参数 | smc_set_vector_profile_unit | 设置插补运动速度参数 |
| | smc_get_vector_profile_unit | 读取插补运动速度参数 |
| | smc_set_vector_s_profile | 设置插补运动速度曲线的平滑时间 |
| | smc_get_vector_s_profile | 读取设置的插补运动速度曲线平滑时间 |
| 单段插补运动 | smc_line_unit | 直线插补运动 |
| | smc_arc_move_center_unit | 圆心+圆弧终点模式扩展的螺旋线插补运动（可作两轴圆弧插补） |
| | smc_arc_move_radius_unit | 半径+圆弧终点模式扩展的螺旋线插补运动（可作两轴圆弧插补） |

| | | |
|---|---|---|
| | smc_arc_move_3points_unit | 三点圆弧模式扩展的螺旋线插补运动（可作两轴及三轴空间圆弧插补） |
| 连续插补运动 | smc_conti_open_list | 打开连续插补缓冲区 |
| | smc_conti_set_lookahead_mode | 设置连续插补前瞻模式及参数 |
| | smc_conti_get_lookahead_mode | 回读连续插补前瞻模式及参数 |
| | smc_conti_start_list | 开始连续插补 |
| | smc_conti_close_list | 关闭连续插补缓冲区 |
| | smc_conti_pause_list | 暂停连续插补 |
| | smc_conti_stop_list | 停止连续插补 |
| | smc_set_arc_limit | 设置圆弧限速功能 |
| | smc_get_arc_limit | 回读圆弧限速功能 |
| | smc_conti_set_blend | 设置连续插补 Blend 拐角过渡模式使能状态 |
| | smc_conti_get_blend | 读取连续插补 Blend 拐角过度模式使能状态设置 |
| | smc_conti_change_speed_ratio | 动态调整连续插补速度比例 |
| | smc_conti_delay | 连续插补中暂停延时指令 |
| | smc_conti_line_unit | 连续插补中直线插补指令 |
| | smc_conti_arc_move_center_unit | 连续插补中基于圆心+终点圆弧扩展的螺旋线插补指令（可作两轴圆弧插补） |
| | smc_conti_arc_move_radius_unit | 连续插补中基于半径+终点圆弧扩展的圆柱螺旋线插补指令（可作两轴圆弧插补） |
| | mc_conti_arc_move_3points_unit | 连续插补中基于三点圆弧扩展的圆柱螺旋线插补指令（可作两轴及三轴圆弧插补） |
| | smc_conti_pmove_unit | 连续插补中控制指定外轴运动指令 |

| | | |
|---|---|---|
| 连续插补状态检测 | smc_conti_remain_space | 查询连续插补缓冲区剩余插补空间 |
| | smc_conti_read_current_mark | 读取连续插补缓冲区当前插补段号 |
| | smc_conti_get_run_state | 读取连续插补运动状态 |
| | smc_check_done_multicoor | 检测连续插补运行状态 |
| 连续插补 IO | smc_conti_set_pause_output | 设置连续插补暂停及异常停止时 IO 输出状态 |
| | smc_conti_get_pause_output | 读取连续插补暂停及异常停止时 IO 输出状态设置 |
| | smc_conti_wait_input | 连续插补等待 IO 输入 |
| | smc_conti_delay_outbit_to_start | 连续插补中相对于轨迹段起点 IO 滞后输出（段内执行） |
| | smc_conti_delay_outbit_to_stop | 连续插补中相对于轨迹段终点 IO 滞后输出 |
| | smc_conti_ahead_outbit_to_stop | 连续插补中相对于轨迹段终点 IO 提前输出（段内执行） |
| | smc_conti_write_outbit | 连续插补中缓冲区立即 IO 输出 |
| | smc_conti_clear_io_action | 清除段内未执行完的 IO 动作 |
| PWM 控制输出 | smc_set_pwm_output | 设置 PWM 立即输出参数 |
| | smc_get_pwm_output | 读取 PWM 当前输出参数 |
| 通用 IO 操作 | smc_read_inbit | 读取指定控制器的某个输入端口的电平 |
| | smc_write_outbit | 设置指定控制器的某个输出端口的电平 |
| | smc_read_outbit | 读取指定控制器的某个输出端口的电平 |
| | smc_read_inport | 读取指定控制器的全部输入端口的电平 |

| | | |
|---|---|---|
| | smc_read_outport | 读取指定控制器的全部输出口的电平 |
| | smc_write_outport | 设置指定控制器的全部输出口的电平 |
| | smc_reverse_outbit | IO 输出延时翻转 |
| | smc_set_io_count_mode | 设置 IO 计数模式 |
| | smc_get_io_count_mode | 读取 IO 计数模式设置 |
| | smc_set_io_count_value | 重置 IO 计数值 |
| | smc_get_io_count_value | 读取 IO 计数值 |
| 专用 IO 操作 | smc_set_inp_mode | 设置指定轴的 INP 信号 |
| | smc_get_inp_mode | 读取指定轴的 INP 信号设置 |
| | smc_set_alm_mode | 设置指定轴的 ALM 信号 |
| | smc_get_alm_mode | 读取指定轴的 ALM 信号设置 |
| | smc_write_sevon_pin | 控制指定轴的伺服使能端口的输出 |
| | smc_read_sevon_pin | 读取指定轴的伺服使能端口的电平 |
| | smc_write_erc_pin | 控制指定轴的 ERC 信号输出 |
| | smc_read_erc_pin | 读取指定轴的 ERC 端口电平 |
| | smc_read_alarm_pin | 读取指定轴的 ALARM 端口电平 |
| | smc_read_inp_pin | 读取指定轴的 INP 端口电平 |
| | smc_read_org_pin | 读取指定轴的 ORG 端口电平 |
| | smc_read_elp_pin | 读取指定轴的 ELP 端口电平 |
| | smc_read_eln_pin | 读取指定轴的 ELN 端口电平 |
| | smc_read_emg_pin | 读取指定轴的 EMG 端口电平 |
| 手轮 | smc_handwheel_set_axislist | 设置同一轴选档位下具体运动轴 |
| | smc_handwheel_get_axislist | 读取同一轴选档位下具体运动轴 |
| | smc_handwheel_set_ratiolist | 设置同一轴选下手轮倍率档位 |
| | smc_handwheel_get_ratiolist | 读取同一轴选下手轮倍率档位 |

| | | |
|---|---|---|
| | smc_handwheel_set_mode | 设置手轮运动模式，硬件、还是软件模式下运动 |
| | smc_handwheel_get_mode | 读取手轮运动模式，硬件、还是软件模式下运动 |
| | smc_handwheel_set_index | 选择或更换手轮运动轴选、倍率档位 |
| | smc_handwheel_get_index | 读取选择或更换手轮运动轴选、倍率档位 |
| | smc_handwheel_move | 启动手轮运动 |
| | smc_handwheel_stop | 停止手轮运动 |
| 编码器 | smc_set_counter_inmode | 设置编码器的计数方式 |
| | smc_get_counter_inmode | 读取编码器的计数方式 |
| | smc_set_encoder_unit | 设置指定轴编码器脉冲计数值 |
| | smc_get_encoder_unit | 读取指定轴编码器脉冲计数值 |
| | smc_set_ez_mode | 设置指定轴的 EZ 信号电平 |
| | smc_get_ez_mode | 读取指定轴的 EZ 信号电平 |
| | smc_set_counter_reverse | 设置 AB 相计数值反相 |
| | smc_get_counter_reverse | 读取 AB 相计数值反相模式 |
| 高速位置锁存 | smc_set_ltc_mode | 设置指定轴的 LTC 信号 |
| | smc_get_ltc_mode | 读取指定轴的 LTC 信号设置 |
| | smc_set_latch_mode | 设置锁存方式 |
| | smc_get_latch_mode | 读取锁存方式 |
| | smc_get_latch_value_unit | 从控制器内读取锁存器的值 |
| | smc_get_latch_flag | 读取指定轴的锁存次数 |
| | smc_reset_latch_flag | 复位锁存器的标志位 |
| 原点锁存 | smc_set_homelatch_mode | 设置原点锁存模式 |
| | smc_get_homelatch_mode | 读取原点锁存模式设置 |
| | smc_reset_homelatch_flag | 清除原点锁存标志 |
| | smc_get_homelatch_flag | 读取原点锁存标志 |
| | smc_get_homelatch_value_unit | 读取原点锁存值 |

| | | |
|---|---|---|
| EZ 锁存 | smc_set_ezlatch_mode | 设置 ez 锁存模式 |
| | smc_get_ezlatch_mode | 读取 ez 锁存模式设置 |
| | smc_reset_ezlatch_flag | 清除 ez 锁存标志 |
| | smc_get_ezlatch_flag | 读取 ez 锁存标志 |
| | smc_get_ezlatch_value_unit | 读取 ez 锁存值 |
| 单轴位置比较 | smc_compare_set_config | 设置一维位置比较器 |
| | smc_compare_get_config | 读取一维位置比较器设置 |
| | smc_compare_clear_points | 清除已添加的所有一维位置比较点 |
| | smc_compare_add_point_unit | 添加一维位置比较点 |
| | smc_compare_get_current_point_unit | 读取当前比较点位置 |
| | smc_compare_get_points_runned | 读取已经比较过的点数量 |
| | smc_compare_get_points_remained | 读取已经比较过的点数量 |
| 二维位置比较 | smc_compare_set_config_extern | 设置二维位置比较器 |
| | smc_compare_get_config_extern | 读取二维位置比较器设置 |
| | smc_compare_clear_points_extern | 清除已添加的所有二维位置比较点 |
| | smc_compare_add_point_extern_unit | 添加二维位置比较点 |
| | smc_compare_get_current_point_extern_unit | 读取当前二维位置比较点位置 |
| | smc_compare_get_points_runned_extern | 查询已经比较过的二维比较点个数 |
| | smc_compare_get_points_remained_extern | 查询可以加入的二维比较点个数 |
| 高速位置比较 | smc_hcmp_set_mode | 设置高速比较模式 |
| | smc_hcmp_get_mode | 读取高速比较模式设置 |
| | smc_hcmp_set_config | 配置高速比较器 |

| | | |
|---|---|---|
| | smc_hcmp_get_config | 读取高速比较器配置 |
| | smc_hcmp_clear_points | 清除已添加的所有高速位置比较点 |
| | smc_hcmp_add_point_unit | 添加/更新高速比较位置 |
| | smc_hcmp_set_liner_unit | 设置高速比较线性模式参数 |
| | smc_hcmp_get_liner_unit | 读取高速比较线性模式参数设置 |
| | smc_hcmp_get_current_state | 读取高速比较状态 |
| | smc_write_cmp_pin | 控制指定 CMP 端口的输出 |
| | smc_read_cmp_pin | 读取指定 CMP 端口的电平 |
| 软件硬件限位 | smc_set_el_mode | 设置 EL 限位信号 |
| | smc_get_el_mode | 读取 EL 限位信号设置 |
| | smc_set_softlimit_unit | 设置软限位 |
| | smc_get_softlimit_unit | 读取软限位设置 |
| 运动异常停止函数 | smc_stop | 指定轴停止运动 |
| | smc_stop_multicoor | 停止坐标系内所有轴的运动 |
| | smc_emg_stop | 紧急停止所有轴 |
| | smc_set_emg_mode | 设置 EMG 急停信号 |
| | smc_get_emg_mode | 读取 EMG 急停信号设置 |
| | smc_set_io_dstp_mode | 设置 IO 触发减速停止电平信号 |
| | smc_get_io_dstp_mode | 读取 IO 触发减速停止参数 |
| | smc_set_dec_stop_time | 设置减速停止时间 |
| | smc_get_dec_stop_time | 读取减速停止时间设置 |
| 轴 IO 映射 | smc_set_axis_io_map | 设置轴 IO 映射参数 |
| | smc_get_axis_io_map | 读取轴 IO 映射参数 |
| 轴虚拟映射 | smc_set_io_map_virtual | 设置虚拟 IO 映射参数 |
| | smc_get_io_map_virtual | 读取虚拟 IO 映射参数 |
| | smc_read_inbit_virtual | 读取滤波后的虚拟 IO 口电平状态 |
| 密码管理 | smc_write_password | 加密 |
| | smc_check_password | 密码校验 |
| | smc_enter_password | 登陆密码 |

| | | |
|---|---|---|
| | smc_modify_password | 修改登陆密码 |
| 文件管理 | smc_download_file | 下载本地文件到 FLASH |
| | smc_download_memfile | 下载内存文件到 FLASH |
| | smc_upload_file | 上传FLASH 文件到本地文件 |
| | smc_upload_memfile | 上传 FLASH 文件到内存文件 |
| | smc_download_file_to_ram | 下载本地文件到 RAM，掉电不保存 |
| | smc_download_memfile_to_ram | 下载内存文件到RAM，掉电不保存 |
| | smc_get_progress | 文件下载进度 |
| 寄存器操作 | smc_set_modbus_0x | 写位寄存器 |
| | smc_get_modbus_0x | 读位寄存器 |
| | smc_set_modbus_4x | 写字寄存器 |
| | smc_get_modbus_4x | 读字寄存器 |
| 模拟量操作 | smc_get_ain | 读取模拟量电压值 |
| | smc_set_ain_action | 设置模拟量参数 |
| | smc_get_ain_action | 回读模拟量参数值 |
| | smc_get_ain_state | 读取模拟量输入触发状态值 |
| | smc_set_ain_state | 置位模拟量输入触发状态 |
| BASIC 相关函数 | smc_write_array | 按索引写数组值 |
| | smc_read_array | 按索引读数组值 |
| | smc_modify_array | 按索引修改数组值 |
| | smc_read_var | 读变量值 |
| | smc_modify_var | 修改变量值 |
| | smc_get_stringtype | 读取变量类型 |
| | smc_basic_delete_file | 删除BASIC程序 |
| | smc_basic_run | 运行 |
| | smc_basic_stop | 停止 |
| | smc_basic_pause | 暂停 |
| | smc_basic_step_run | 单步运行 |
| | smc_basic_step_over | 运行到下一断点 |

| | | |
|---|---|---|
| | smc_basic_continue_run | 继续运行 |
| | smc_basic_state | 当前状态 |
| | smc_basic_current_line | 当前执行行 |
| | smc_basic_break_info | 断点信息 |
| | smc_basic_message | 读取输出信息 |
| | smc_basic_command | 在线命令 |
| G 代码相关函数 | smc_gcode_check_file | 检查文件是否存在 |
| | smc_gcode_delete_file | 删除文件 |
| | smc_gcode_clear_file | 删除所有文件 |
| | smc_gcode_get_first_file | 读取第一个文件名 |
| | smc_gcode_get_next_file | 读取下一个文件名 |
| | smc_gcode_start | 启动 |
| | smc_gcode_stop | 停止 |
| | smc_gcode_pause | 暂停 |
| | smc_gcode_state | 读取当前状态 |
| | smc_gcode_set_current_file | 设置当前文件 |
| | smc_gcode_get_current_file | 读取当前文件名 |
| | smc_gcode_current_line | 读取当前运行行 |
| | smc_gcode_get_file_profile | 读取G文件属性 |
| 总线相关函数 | nmcs_reset_canopen | 复位CANopen总线 |
| | nmcs_stop_etc | 停止EtherCAT总线运行 |
| | nmcs_axis_io_status | 获取轴IO状态 |
| | nmcs_get_LostHeartbeat_Nodes | 获取心跳报文信息，支持 CANopen、EtherCat |
| | nmcs_get_EmergeneyMessege_Nodes | 获取紧急报文信息，支持 CANopen、EtherCAT |

| | | |
|---|---|---|
| 总线相关函数 | nmcs_set_node_od | 设置从站对象字典 |
| | nmcs_get_node_od | 获取从站对象字典 |
| | nmcs_SendNmtCommand | 发送NMT管理报文 |
| | nmcs_get_cycletime | 读取EtherCAT总线循环周期 |
| | nmcs_write_rxpdo_extra | |
| | nmcs_read_rxpdo_extra | |
| | nmcs_read_txpdo_extra | |
| | nmcs_get_axis_type | 读取总线轴类型 |
| | nmcs_set_axis_enable | 设置 EtherCAT 总线驱动器使能 |
| | nmcs_set_axis_disable | 设置EtherCAT总线驱动器禁止使能 |
| | nmcs_syn_move | 同步运动 |
| | nmcs_get_axis_state_machine | 读取EtherCAT总线轴状态机 |
| | nmcs_get_total_axes | 读取EtherCAT总线轴和虚拟轴轴数 |
| | nmcs_get_total_adcnum | 读取 EtherCAT 总线 AD/DA 输入输出口数 |
| | nmcs_get_total_ionum | 读取 EtherCAT 总线 IO 输入输出口数 |
| | nmcs_get_total_adcnum | 获取 EtherCAT 总线模拟量 I/O 点数 |
| | nmcs_get_total_slaves | 获取EtherCAT从站总数 |
| | nmcs_get_axis_io_out | 获取EtherCAT轴数字量输出IO状态 |
| | nmcs_set_axis_io_out | 设置EtherCAT轴数字量输出IO状态 |
| | nmcs_get_axis_io_in | 获取EtherCAT轴数字量输入IO状态 |
| | nmcs_write_outbit | 设置指定控制器或扩展模块的某个输出端口的电平 |

| | | |
|---|---|---|
| | nmcs_read_outbit | 读取指定控制器或扩展模块的某个输出端口的电平 |
| | nmcs_read_inbit | 读取指定控制器或扩展模块的某个输入端口的电平 |
| | nmcs_write_outport | 设置指定 IO 组号的全部输出口的电平 |
| | nmcs_read_outport | 读取指定 IO 组号的全部输出口的电平 |
| | nmcs_read_inport | 读取指定 IO 组号的全部输入口的电平 |
| | nmcs_set_alarm_clear | 清除报警信号 |
| | nmcs_get_errcode | 读取 EtherCAT 总线状态 |
| | nmcs_clear_errcode | 清除总线错误 |
| | nmcs_get_card_errcode | 获取指定链接号总线错误代码 |
| | nmcs_clear_card_errcode | 清除指定链接号总线错误代码 |
| | nmcs_get_axis_errcode | 获取指定轴总线错误代码 |
| | nmcs_clear_axis_errcode | 清除指定轴号总线错误代码 |
| | | |
| | | |
| | | |

# 表格 2：指令运行错误一览表

当BASIC程序出现语法错误或者参数范围错误等情况时，BASIC程序会被停止运行，同时会有错误的位置和信息输出。详见下表。

| 错误代码 | 错误号 | 可能错误原因 |
|---|---|---|
| Success | 0 | 操作成功 |
| Unknown Error | 1 | 未知错误 |
| Parameter Error | 2 | 参数错误 |
| Operate Timeout | 3 | 操作超时 |
| State Busy | 4 | 状态忙 |
| Too Many Connections | 5 | 链接太多 |
| Interpolation Error | 6 | 插补错误 |
| Connection Failure | 7 | 连接失败 |
| Cannot be connected | 8 | 无法连接 |
| Axis number is out of range | 9 | 卡号或轴号超出范围 |
| transport error | 10 | 数据传输错误 |
| firmware file error | 12 | 固件文件错误 |
| The firmware does not match | 14 | 固件文件不匹配 |
| firmware parameters error | 20 | 固件参数错误 |
| The current state of firmware is not allowed to operate | 22 | 固件当前状态不允许操作 |
| The feature is not supported | 24 | 不支持该功能 |
| password error | 25 | 密码错误 |
| The number of password inputs is limited | 26 | 密码输入次数受限 |
| ERR_AXIS_SEL_ERR | 30 | 手轮脉冲的轴档位选择超出范围 |

| | | |
|---|---|---|
| | | （软件控制模式） |
| ERR_HAND_AXIS_NUM_ERR | 31 | 手轮脉冲的轴映射数量超出范围 |
| ERR_AXIS_RATIO_ERR | 32 | 手轮脉冲的倍率档位选择超出范围（软件控制模式） |
| ERR_HANDWH_START | 33 | 已进入手轮脉冲模式，不能切换软硬件控制模式 |
| ERR_AXIS_BUSY_STATE | 34 | 轴已在运动，不能切换到手轮模式 |
| ERR_LIST_NUM_ERR | 50 | LIST号超出范围 |
| ERR_LIST_NOT_OEPN | 51 | LIST没有初始化 |
| ERR_PARA_NOT_VALID | 52 | 参数不在有效范围 |
| ERR_LIST_HAS_OPEN | 53 | LIST已经打开 |
| ERR_MAIN_LIST_NOT_OPEN | 54 | LIST没有初始化 |
| ERR_AXIS_NUM_ERR | 55 | 轴数不在有效范围 |
| ERR_AXIS_MAP_ARRAY_ERR | 56 | 轴映射表为空 |
| ERR_MAP_AXIS_ERR | 57 | 映射轴错误 |
| ERR_MAP_AXIS_BUSY | 58 | 映射轴忙 |
| ERR_PARA_SET_FORBIT | 59 | 运动中不允许更改参数 |
| ERR_FIFO_FULL | 60 | 缓冲区已满 |
| ERR_RADIUS_ERR | 61 | 半径为0或小于两点的距离的一半 |
| ERR_MAINLIST_HAS_START | 62 | LIST已经启动 |
| ERR_ACC_TIME_ZERO | 63 | 加减速时间为0 |
| ERR_MAINLIST_NOT_START | 64 | 主要LIST没有启动 |
| ERR_POINT_SAME_ON_RADIUS | 67 | 圆弧插补在半径模式下起点和终点不能重合 |
| MCVP_SMOOTH_TIME_SET_ERROR | 80 | s时间设置错误(小于等于0) |
| MCVP_START_VEL_SET_ERROR | 81 | 起始速度绝对值设置错误(小于0) |
| MCVP_STEADY_VEL_SET_ERROR | 82 | 最大速度绝对值设置错误(小于等于0) |
| MCVP_END_VEL_SET_ERROR | 83 | 停止速度绝对值设置错误(小于0) |

| | | |
|---|---|---|
| MCVP_TOTAL_LENGTH_SET_ERROR | 84 | 运动距离为0，无法运动 |
| ERR_AXIS_INDEX | 101 | 所选轴超出最大值 |
| ERR_SET_WHILE_MOVING | 102 | 轴正在运动，不能设置参数 |
| ERR_ENTER_WHILE_MOVING | 103 | 轴正在运动，不能进入该模式 |
| ERR_PEL_STATE | 104 | 轴处于正限位，不能正向运动 |
| ERR_NEL_STATE | 105 | 轴处于负限位，不能负向运动 |
| ERR_SOFT_PEL_STATE | 106 | 轴处于软正限位，不能正向运动 |
| ERR_SOFT_NEL_STATE | 107 | 轴处于软负限位，不能负向运动 |
| ERR_FORCE_IN_OTHER_MODE | 108 | 轴处于非点位模式，不能强制变位 |
| ERR_MAX_VEL_ZERO | 109 | 设置最大速度错误，不能为0 |
| ERR_EQU_ZERO | 110 | 设置轴当量错误，不能为0 |
| ERR_BACKLASH_NEG | 111 | 设置反向间隙错误，不能为负值 |
| ERR_MAX_PULSE | 112 | 设置位置错误，已超出允许范围 |
| ERR_CMP_EXCEED_MAX_AXISES | 121 | 所选比较轴超出范围 |
| ERR_CMP_EXCEED_MAX_INDEX | 122 | 比较点数已满，不能继续添加 |
| ERR_CMP_EXCEED_MAX_IO | 123 | 进行比较的IO超出范围 |
| ERR_CMP_EXCEED_MAX_CHAN | 124 | 选择的高速比较IO超出范围 |
| ERR_MAP_AXISIO_MAX_AXISES | 130 | 映射的轴超出范围 |
| PVT_ERROR_AXISES_OVER_RANGE | 140 | 所选轴超出范围 |
| PVT_ERROR_INDEX_OVER_RANGE | 141 | 控制点已满，不能继续添加 |
| PVT_ERROR_INDEX_EXCEED | 142 | 控制点已满，不能继续添加 |
| PVT_ERROR_TIME_EROOR | 143 | 插入段时间为0或者负数 |
| HOME_ERROR_AXISES_OVER_RANGE | 200 | 所选轴超出最大值 |
| HOME_ERROR_MAX_VEL | 202 | 设置的最大速度为0 |
| HOME_ERROR_MAX_ACC | 203 | 设置的加速度小于等于0 |
| HOME_ERROR_BOTH_LIMIT | 207 | 同时处于正、负限位,无法启动回零运动 |
| ERROR_ZSHELL_PARAERR | 1000 | 参数错误 |
| ERROR_ZSHELL_STOP_USER | 1040 | 用户手动停止 |

| | | |
|---|---|---|
| ERROR_ZSHELL_STOP_OTHERTASK | 1041 | 其他任务牵连停止 |
| ERROR_ZSHELL_PARA_CANNOT_MODIFIY | 1042 | 少数参数不让修改，SET扩展返回 |
| ERROR_ZSHELL_ARRAY_OVER | 1043 | 数组越界 |
| ERROR_ZSHELL_VAR_TOOMUCH | 1044 | 变量个数超过 |
| ERROR_ZSHELL_ARRAY_TOOMUCH | 1045 | 数组个数超过 |
| ERROR_ZSHELL_ARRAY_NOSPACE | 1046 | 数组没有空间 |
| ERROR_ZSHELL_SUB_TOOMUCH | 1047 | SUB过多 |
| ERROR_ZSHELL_LABEL_NAMEERR | 1048 | 标识符 命名错误 |
| ERROR_ZSHELL_LABEL_TOOMANYCHARES | 1049 | 标识符 命名过长 |
| ERROR_ZSHELL_NO_RIGHTBRACKET | 1050 | 没有右括号 |
| ERROR_ZSHELL_UNKOWNCHAR | 1051 | 不认识的字符 |
| ERROR_ZSHELL_UNKOWN_LABEL | 1052 | 不认识的名称，表达式中碰到 |
| ERROR_ZSHELL_STOP_INVALIDCMD | 1053 | 不认识的命令 |
| ERROR_ZSHELL_STOP_OVERSTACK | 1054 | 超过堆栈层数 |
| ERROR_ZSHELL_OVER_RECURSION | 1055 | 递归过多 |
| ERROR_ZSHELL_NO_QUOTEEND | 1056 | 引号没有结束 |
| ERROR_ZSHELL_CMD_CANNOTREAD | 1057 | 不能读取，不能在表达式中使用 |
| ERROR_ZSHELL_CMD_CANNOTRUN | 1058 | 函数之类不能出现在行首，只能在表达式中使用 |
| ERROR_ZSHELL_LINE_MUST_END | 1059 | 期望行结束，一些特殊指令需要 |
| ERROR_ZSHELL_ARRAY_NEEDINDEX | 1060 | 数组需要编号， 参数也使用 |
| ERROR_ZSHELL_NOTBRACKET_AFTERVAR | 1061 | 变量后不需要编号 |
| ERROR_ZSHELL_DIM_CONFLICT | 1062 | 数组重新定义冲突，长度不一致 |
| ERROR_ZSHELL_DIM_ARRAYLENGTHERR | 1063 | 数组长度错误 |
| ERROR_ZSHELL_DIM_LABEL_SUB | 1064 | 定义，名称 SUB |
| ERROR_ZSHELL_DIM_LABEL_PARA | 1065 | 定义，名称 PARA |
| ERROR_ZSHELL_DIM_LABEL_RESERVE | 1066 | 定义，名称 保留 |
| ERROR_ZSHELL_UNWANT_CHAR | 1067 | 不能出现的字符 |
| ERROR_ZSHELL_STACK_NOPUSH | 1068 | POP时没有压栈 |

| | | |
|---|---|---|
| ERROR_ZSHELL_FORMAT_ERR | 1070 | 格式错误 |
| ERROR_ZSHELL_SET_OVER | 1071 | 参数溢出，para(10) 10过大 |
| ERROR_ZSHELL_PARA_RANGEERR | 1072 | 一些函数和命令的参数范围错误 |
| ERROR_ZSHELL_PARA_TOOMANY | 1073 | 一些函数和命令的参数 过多 |
| ERROR_ZSHELL_PARA_TOOFEW | 1074 | 一些函数和命令的参数 过少 |
| ERROR_ZSHELL_NO_EXPR | 1075 | 读取不到表达式 |
| ERROR_ZSHELL_OPERNOPARA | 1076 | 操作符没有参数 |
| ERROR_ZSHELL_NOPARA_BEFOREOPER | 1077 | 操作符前面没有参数 |
| ERROR_ZSHELL_SIGNAL_CANNOTINEXPR | 1078 | 符号不能在表达式中使用 |
| ERROR_ZSHELL_NEED_OPER | 1079 | 需要双目操作符 |
| ERROR_ZSHELL_SUB_NOTSUB | 1080 | CALL 的不是SUB |
| ERROR_ZSHELL_NO_AUTO | 1081 | 没有AUTO所以不启动 |
| ERROR_ZSHELL_EQ_WANTED | 1082 | 赋值语句没有等号，变量或者参数等只能为赋值语句 |
| ERROR_ZSHELL_FILE_VAIN | 1083 | 程序文件空 |
| ERROR_ZSHELL_SUB_RENAME | 1084 | SUB重名，包括与其他的名称重了 |
| ERROR_ZSHELL_TASK_RUNNING | 1085 | 任务已经运行中 |
| ERROR_ZSHELL_NEED_COMMA_BETWEEN_PARA | 1086 | 操作数之间需要逗号 |
| ERROR_ZSHELL_NO_LEFTBRACKET | 1087 | 没有右括号 |
| ERROR_ZSHELL_TOOMANY_IFNESTED | 1088 | IF嵌套太多 |
| ERROR_ZSHELL_TOOMANY_LOOPNESTED | 1089 | LOOP嵌套太多 |
| ERROR_ZSHELL_MOVEAXISES_FEW | 1090 | 插补轴太少 |
| ERROR_ZSHELL_CONST_VAR | 1091 | 变量不能修改 |
| ERROR_ZSHELL_NOT_ONLINECMD | 1092 | 不能作为在线命令 |
| ERROR_ZSHELL_AXIS_OVER | 1093 | 轴号超出 |
| ERROR_ZSHELL_CRD_OVER | 1094 | 插补系超出 |
| ERROR_ZSHELL_STOP_UNKNOWN | 1099 | 未知错误，不可能出现的，一般是内部错误引起 |

| | | |
|---|---|---|
| ERROR_ZSHELL_DIVISION_BY_ZERO | 1200 | 除零错误，请排查除数是否为 0 |

深圳市雷赛控制技术有限公司
SHENZHEN LEADSHINE CONTROL TECHNOLOGY CO.,LTD

深 圳 市 雷 赛 控 制 技 术 有 限 公 司

地　址：深圳市深圳市南山区学苑大道1001号南山智园Ａ3栋9楼

邮　编：518052

电　话：0755-26415968

传　真：0755-26417609

Email：　info@szleadtech.com.cn

网　址：http://www.szleadtech.com.cn