固高科技（深圳）有限公司

地　　址：深圳市高新技术产业园南区深港产学研基地西座二层 10＃

电　　话：0755-26970823　26970819　26970824

传　　真：0755-26970821

电子邮件：googol@googoltech.com

网　　址：http://www.googoltech.com.cn/

Googol Technology (HK) Ltd

Addr:　Room 4583,Annex Building

Hong Kong University of Science and Technology Hong Kong

Tel:　(852) 2358-1033

Fax:　(852) 2358-4931

E-mail:　info@googoltech.com

Web:　Http://www.googoltech.com

GT 系列运动控制器
编程手册

务必将此手册交给用户

- 非常感谢您选购 GT 系列运动控制器
- 在您使用之前，请仔细阅读此手册，确保正确使用。
- 请将此手册妥善保存，以备随时查阅。

版权申明

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

前言

感谢选用固高运动控制器

为回报客户，我们将以品质一流的运动控制器、完善的售后服务、高效的技术支持，帮助您建立自己的控制系统。

技术支持和售后服务

您可以通过以下途径获得我们的技术支持和售后服务：

- 电子邮件： 邮件地址 googol@googoltech.com；
- 电　　话： 0755-26970823、26970819、26970824
- 发　　函： 深圳市高新技术产业园南区深港产学研基地大楼
  西座二层 10＃
  固高科技（深圳）有限公司
  邮编：518057

编程手册的用途

用户通过阅读本手册，能够了解 GT 系列运动控制器的控制功能，掌握运动函数的用法，熟悉特定控制功能的编程实现。最终，用户可以根据自己特定的控制系统，编制上层应用程序，实现控制要求。

编程手册的使用对象

本编程手册适用于，具有 C 语言编程基础知识，有一定运动控制工作经验，对固高运动控制器的基本结构有一定了解的工程开发人员。

编程手册的主要内容

本手册分两部分。第一部分 编程说明，主要讲述 GT 系列运动控制器的各种控制功能以及相应的编程实现。第二部分 接口库函数说明，主要给出 GT 系列运动控制器接口函数的函数原型、函数说明及函数调用。

编程手册使用说明

本手册所有例程在除特别声明外，凡 DOS 例程全部使用 BorlandC3.1 编写，凡 Windows 例程全部使用 VC++编写。

相关文件

关于 GT 系列运动控制器的安装和调试，请参见随产品配套的《GT 系列运动控制器用户手册》。

目 录

目 录

第一部分

编程说明

第一章 运动控制器函数库的使用

运动控制器提供 DOS 下的运动函数库和 Windows 下的运动函数动态连接库。用户只要调用运动函数库中的函数，就可以实现运动控制器的各种功能。下面分别讲述 DOS、Windows 系统下函数库的使用。

1.1 DOS 系统下函数库的使用

DOS 下的运动命令函数库存放于产品配套光盘的 DOS\UserLib 下，共七个文件。分别为：

userlib.h	头文件
userlibt.lib	微模式的函数库
userlibs.lib	小模式下的函数库
userlibm.lib	中模式的函数库
userlibc.lib	紧凑模式的函数库
userlibl.lib	大模式的函数库
userlibh.lib	巨大模式的函数库

该函数库是用 BorlandC3.1 编译生成，开发者可在 BorlandC3.1 或更高版本的开发环境下链接函数库。

开发者使用函数库的方法如下：

1. BorlandC 开发环境下，选择菜单 Project——Open Project，建立工程文件；
2. 在用户开发的程序中加入：＃include “userlib.h”；
3. BorlandC 开发环境下，选择菜单 Project——Add Item，将所开发的 c 或 cpp 文件添加到工程文件中；
4. 将 userlib.h 及需要的库文件拷贝到工程文件目录下；
5. BorlandC 开发环境下，选择菜单 Project——Add Item，将库文件添加到工程文件中；
6. BorlandC 开发环境下，选择菜单 Option——Compiler——Code Generation 在 Model 栏中选择与库文件相应的编译模式；
7. 至此，可以调用运动命令函数库中的任何函数。具体函数的使用及定义，请参考本手册第二部分。

1.2 Windows 系统下动态连接库的使用

Windows 下的动态连接库存放于产品配套光盘的 Windows\Dll 下，共三个文件，分别为：Gtdll.h，GTDLL.lib，GTDLL.dll。这些文件是用 VC++编写的。对于目前程序员经常使用的高级编程语言：VC，VB，Delphi，下面将分别讲述各种语言下动态连接库的使用方法。

1.2.1 VC 中的使用

ISA 卡：

1. 在用户程序中加入：
   #include “Gtdll.h"
2. 在 VC 环境菜单中，
   选择 project--setting--link，在 Object/library modules 中输入 gtdll.lib
   然后用户即可在程序中调用动态联接库中的函数。

PCI 卡：

1. 在用户程序中加入：
   #include “Gt400.h"
   #include “gt400data.h”
2. 在 VC 环境菜单中，
   选择 project--setting--link，在 Object/library modules 中输入 gt400.lib
   然后用户即可在程序中调用动态联接库中的函数。

1.2.2 VB 中的使用

用户可根据板卡的总线类型，将光盘中 Windows\VB 目录下提供的 GTDeclarISA.bas 或 GTDeclarPCI.bas 加入用户的工程中，即可直接调用。

1.2.3 Delphi 中的使用

用户可将光盘中 Windows\Delphi 目录下提供的 GTFunc.pas 加入用户的工程中，即可直接调用。

第二章 命令返回值及其意义

2.1 命令(库函数)返回值

命令与库函数 运动控制器是按照主机发送的***运动控制命令***工作的。运动控制器配套有 C 语言函数库（DOS 环境）和动态连接库（Windows 环境）。用户通过在主机编制程序调用相应的***库函数***，也就是向运动控制器发出***运动控制命令***（简称***命令***）。

对于用户通过主机发送的命令，运动控制器在检查、校验后，会给出一个反馈。这个反馈就是***命令（库函数）的返回值***。返回值的定义如表 2-1。

表 2-1 返回值定义

返回值	意义	处理方法
-1	通讯出错	请参照《GT 系列运动控制器用户手册》中“故障处理”，检查运动控制器是否工作正常。
0	命令执行成功	继续
1	命令错误	调用 GT_GetCmdSts()命令，确定出错原因，予以改正。
2	圆弧插补半径不正确	该返回值只可能出现在 GT_ArcXY、GT_ArcYZ、GT_ArcZX 命令发送后。检查该命令的参数，改正后重发此命令。
3	直线插补长度为零或超出控制器处理范围	该返回值只可能出现在 GT_LnXY、GT_LnXYZ、GT_LnXYZA 命令发送后。检查该命令的参数，改正后重发此命令。
4	坐标运动模式下，（加）速度为零或负数或超出控制器处理范围。	该返回值只可能出现在 GT_SetSynVel、GT_SetSynAcc 命令发送后。检查该命令的参数，改正后重发此命令。
5	圆弧终点或半径描述错误	该返回值只可能出现在 GT_ArcXYP、GT_ArcYZP、GT_ArcZXP 命令发送后。检查该命令参数是否正确合理，改正后重发。
6	坐标映射失败	该返回值只可能出现在 GT_MapAxis 命令发送后。检查坐标映射是否正确，或者是否与以前已经执行的坐标映射指令相矛盾，改正后重发。
7	一般参数错误	检查该命令的参数是否合理或者是否超出限值，改正后重发此命令。

注意	*建议在用户主程序中，检测每个函数的返回值，以判断命令的执行状态。并建立必要的错误处理机制，保证程序安全可靠地运行。*

如果命令返回值为-1，而且重复调用该命令多次仍返回-1，说明运动控制器的通讯出现故障，运动控制器没有正确地接受主机的命令；或是运动控制器软件工作不正常，不能正确处理主机命令。此时应停止程序运行，参考《GT 系列运动控制器用户手册》的“附录 D 故障处理”，做相应的处理措施。

如果命令返回值为 1，表示主机发出的命令错误，运动控制器忽略这些非法运动命令。如果该命令是***多轴协调运动命令***出错，将引起多轴协调运动状态字的 bit3 置位；如果该命令是***单轴运动命令***出错，将引起当前轴状态字的 bit7 置位。同时，运动控制器的***命令状态寄存器***提供命令出错的详细原因，主机可通过调用命令 GT_GetCmdSts()得到命令出错的原因。

2.2 命令错误寄存器

运动控制器的命令状态寄存器提供命令出错的详细原因，主机可通过调用命令 GT_GetCmdSts()得到命令出错的原因。该寄存器为一个 16 位寄存器，（其中 bit0~bit11 主要针对面向控制轴的命令，bit12~bit15 则反映出面向坐标系命令出错的原因），各位定义见表 2-2：

下例为返回值处理函数，用户可参考该函数，根据自己的控制要求，编制相应的返回值处理函数。

例程 2-1 返回值处理函数

```
void error(short rtn)        //返回值处理函数，rtn 为函数的返回值
{
    switch (rtn)
    {
        case -1:
             printf("error: communciation error\n");   break;
        case 0:
             /*no error, continue */     break;
        case 1:
             printf("error: command error\n");    break;
        case 2:
        case 3:
        case 4:
        case 5:
        case 7:
             printf("error: parameter error\n");   break;
        case 6:
             printf("error: map is error\n");    break;
        default:
             break;
     }

    }
```

表 2-2 命令错误寄存器定义

位	定义
Bit0	1：表示命令输入的控制参数溢出，产生该出错标志的命令有：GT_SetPos，GT_SetVel，GT_SetAcc，GT_SetAtlPos 等。
Bit1	1：表示命令输入的控制参数非法，产生该出错标志的命令有：GT_SetVel，GT_SetAcc，GT_SetJerk，GT_SetMAcc，GT_SetMtrLmt，GT_SetKp，GT_SetKi，GT_SetKd，GT_SetKvff，GT_SetKaff，GT_SetILmt，GT_SetPosErr 等。
Bit2	1：表示主机发送 GT_MltiUpdt (value)命令而 value=0。
Bit3	1：表示 GT_DrvRst 命令使用非法。当前轴在伺服使能状态时，主机发送该命令将产生出错标志
Bit4	1：表示当控制器没有事件中断产生，而主机发送与中断有关的命令
Bit5	（空）
Bit6	1：表示当前轴正在运动时，主机发送修改当前轴的工作模式命令（当前轴处于电子齿轮模式时除外）
Bit7	1：表示当前轴状态寄存器的位置捕获完成标志为 1，或已经设定了 GT_CaptIndex（GT_CaptHome）命令但位置尚未捕获到时，而主机又发送了 GT_CaptIndex 命令
Bit8	1：表示当前轴状态寄存器的位置捕获完成标志为 1，或已经设定了 GT_CaptIndex（GT_CaptHome）命令但位置尚未捕获到时，而主机又发送了 GT_CaptHome 命令
Bit9	1：表示当前轴状态寄存器的驱动器报警标志为 1 时，主机发送 GT_AxisOn 命令
Bit10	（空）
Bit11	1：当前轴状态寄存器中“运动状态标志”为 1 时，主机发送 GT_ZeroPos 命令，该标志位置“1”；在当前轴运动模式为速度控制模式时，主机发送 GT_SynchPos 命令并使其生效，该标志位置“1”；当前轴的状态寄存器中“运动状态标志”为 1 时，用 GT_Update 或 GT_MltiUpdt 修改当前轴控制参数，而这些参数在当前运动模式下是不允许被修改的（如：在 S-曲线运动模式下，在电机运动中，主机发送 GT_SetVel 命令并发送 GT_Update 或 GT_MltiUpdt）
Bit12	1：表示面向坐标系的命令非法。包括： 建立坐标系时，所映射的物理轴正在运动； 缓冲区命令执行状态时，调用 GT_StrtMtn 命令； 缓冲区命令执行状态时，调用 GT_StrtList 命令； 缓冲区命令输入状态时，调用 GT_AddList 命令； 立即命令输入执行状态，运动未完成时，调用新的运动描述命令。
Bit13	1：表示调用 GT_MvXY、GT_MvXYZ、GT_MvXYZA 命令出错
Bit14	（空）
Bit15	1：表示缓冲区满。此时由于运动控制器缓冲区满，刚刚调用的运动描述命令没有被运动控制器接收，主机需要继续重复调用此命令，直至运动控制器接收该命令为止。

第三章 控制系统初始化

3.1 运动控制器初始化

3.1.1 函数列表

表 3-1 运动控制器初始化函数表

函数	说明
GT_Open()	打开运动控制器设备(除 ISA 卡用于 DOS 系统外的其它使用情况)
GT_SetAddr()	设置通讯基地址 (仅对于 ISA 卡用于 DOS 系统下)
GT_SwitchtoCardNo()	指定当前控制卡（仅用于 PCI 多卡控制系统）
GT_Reset()	复位运动控制器
GT_SetSmplTm()	设置控制周期

3.1.2 例程

例程 3-1 运控卡初始化函数(ISA 卡，用于 DOS 系统)

```
void GTInitial()                    //运动控制器初始化函数
{
    short rtn;
    unsigned long PortBase=0x300;  //赋值基地址

    rtn=GT_SetAddr(PortBase ); error(rtn); //设置基地址
    rtn=GT_Reset( );       error(rtn);      //复位运动控制器
    rtn=SetSmplTm(250);    error(rtn);     //设置控制周期为 250us
}
```

例程 3-2 运控卡初始化函数(ISA 卡，用于 Windows 系统)

```
void GTInitial()                    //运动控制器初始化函数
{
    short rtn;
    unsigned long PortBase=0x300;  //赋值基地址
    unsigned long irq=0;            //赋值中断号

    rtn=GT_Open(PortBase;irq );  error(rtn);  //打开运动控制器设备
    rtn=GT_Reset( );    error(rtn);           //复位运动控制器
    rtn=SetSmplTm(250);  error(rtn);      //设置控制周期为 250us
}
```

例程 3-3 运控卡初始化函数（PCI 卡）

```
void GTInitial()                    //运动控制器初始化函数
{
    short rtn;
    rtn=GT_Open( );   error(rtn);     //打开运动控制器设备
    rtn=GT_Reset( );  error(rtn);     //复位运动控制器
    /* 将 1 号卡设为当前卡（仅对于多卡系统，单卡系统可取消该行）*/
    rtn=GT_SwitchtoCardNo(1); error(rtn);
    rtn=SetSmplTm(250); error(rtn);       //设置控制周期为 250us
}
```

3.1.3 重点说明

设置主机与运动控制器的通讯基地址(仅对于 ISA 卡用于 DOS 系统)

运动控制器通过 PC 机的 ISA 总线与主机交换数据。而基地址其实就是向主机指明运动控制器所在的位置，使两者之间的通讯能够建立起来。

设置通讯基地址的函数为：short GT_SetAddr(unsigned short BaseAddr)。参数 BaseAddr 应与运控卡上“基地址跳线开关”的设置相对应（参见《GT 系列运动控制器用户手册》的***在运动控制卡上设置跳线***）。运控器默认通讯基地址为“0x300”。

如果该函数返回值为 0，表示主机与运控器成功建立通讯。如果该函数返回值为-1，说明主机与运控器建立通讯失败。

用户在初始化运控器时，必须首先调用该命令，以建立主机与运控器之间的通讯。

打开运动控制器设备(仅对于 ISA 卡，用于 Windows 系统)

short GT_Open(unsigned long PortBase, unsigned long irq)，打开运动控制器设备，用户程序开始时必须调用此函数。PortBase为运动控制器基地址，irq为运动控制器中断号。选择运动控制器基址和中断号时注意不要与其它设备冲突，否则该函数运行失败。函数返回0表示成功，非0 表示失败。

如果主机上 10～15 号中断都已经占满，或者用户不需要使用中断时，可设参数 irq 的值为 0，以表示无中断。

与 GT_Open 对应的函数是：short GT_Close(void)。关闭运动控制器设备，用户程序结束时必须调用此函数。函数返回 0 表示成功，非 0 表示失败。

设置控制周期

运动控制器以特定的控制周期刷新控制输出。这个控制周期允许用户根据系统要求设置。设置控制周期的命令为：short GT_SetSmplTm(double Timer)；参数 Timer 的单位是微秒。

因为运动控制器在控制周期内要完成必要的控制计算，控制周期不能太小，因此设定的范围为 48~1966.08 微秒。运动控制器默认的控制周期为 200 微秒，这

个控制周期对于普通的用户能够安全可靠地工作。

我们建议用户不要将控制周期设置成小于上述默认值。因为用户设定的控制周期小于控制器默认值时，可能会引起控制器工作不正常。如果由于具体系统的实际需要，必须减小控制周期时，在程序运行过程中必须常常调用 GT_GetCrdSts() 命令来获取控制周期设置是否合适的信息（*有关该命令的详细说明请参见 1.7 状态监测*），若其 bit2 置 1，说明控制周期过小，运动控制器工作不正常，此时应立即将运动控制器复位，恢复控制周期的默认值或将控制周期加大。

设置控制周期只在初始化过程中进行，建议用户在程序运行过程中最好不要修改控制周期。否则将导致不可预期的结果！

3.2 专用输入信号参数设置

3.2.1 函数列表

表 3-2 专用输入信号参数设置函数表

函数	说明
GT_LmtSns()	设置限位开关有效电平
GT_EncSns()	设置编码器计数方向（仅 SV 卡）

3.2.2 例程

例程 3-4 专用输入信号参数设置

```
void InputCfg()                    //专用输入信号参数设置
{
    short rtn;
    unsigned short LmtSense = 0;   //赋值限位开关参数为高电平触发
    unsigned int EncSense = 0xF;    /* 赋值 1～4 轴编码器反向，1、2 辅助编
                                       码器方向不变 */

    rtn=GT_LmtSns(LmtSense); error(rtn); /* 设置 1～4 轴正、负限位开关为
                                            高电平触发*/
    rtn=GT_EncSns(EncSense); error(rtn); //设置编码器方向
    }
```

3.2.3 重点说明

设置限位开关有效电平

运动控制器通过两个（正向、负向）限位开关自动地设定控制轴的运动范围。一旦限位开关被触发，运动控制器自动地禁止控制轴朝越限的方向运动。

运动控制器默认的限位开关为**常闭开关**。即正常工作时，限位开关信号为低电平；限位开关触发时，限位开关为高电平。如果用户使用常开开关，或是由于接线原因造成限位开关电平设置与上述默认状态相反，用户需通过命令改变限位开关有效电平，该命令为：short GT_LmtSns(unsigned short Sense)。

参数 Sense 表示各轴正向（或负向）限位开关有效电平状态，运控器默认值为全“0”。详细说明请参见该函数的定义。

设置编码器方向（仅 SV 卡）

运动控制器缺省设置认为控制电机旋转的正方向（即电机控制电压为正时电机的旋转方向）与编码器计数的正方向（即脉冲计数值增加的方向）相同。但是实际应用中，可能由于电机与编码器的设置不匹配、或是接线错误，造成电机与编码器旋转正方向相反，形成正反馈，导致运动控制器不能正常工作。此时用户可以设置编码器计数的正方向，相应命令为：short GT_EncSns(unsigned int Sense)。

参数 Sense 相应位表示是否修改相应轴编码器的计数方向，运控器默认值为全“0”。详细说明请参见该函数的定义。

3.3 运动控制轴初始化

3.3.1 函数列表

表 3-3 运动控制轴初始化函数表

函数		说明
GT_Axis()		设置当前轴
GT_ClrSts()		清除当前轴状态
GT_StepDir()		设置当前轴在脉冲输出模式下的输出方式为“脉冲/方向”方式
GT_StepPulse()		设置当前轴在脉冲输出模式下的输出方式为“正脉冲/负脉冲”方式
GT_AxisOn()		驱动使能
仅用于SV卡	GT_CtrlMode()	设置输出模拟量/脉冲量
	GT_CloseLoop()	设置为闭环控制
	GT_OpenLoop()	设置为开环控制
	GT_SetKp()	设置当前轴的伺服滤波器比例增益
	GT_SetKi()	设置当前轴的伺服滤波器积分增益
	GT_SetKd()	设置当前轴的伺服滤波器微分增益
	GT_SetKvff()	设置当前轴的伺服滤波器速度前馈增益
	GT_SetKaff()	设置当前轴的伺服滤波器加速度前馈增益
	GT_SetILmt()	设置当前轴的伺服滤波器误差积分饱和值
	GT_SetMtrLmt()	设置当前轴的伺服滤波器输出饱和值
	GT_SetMtrBias()	设置当前轴的伺服滤波器输出零点偏移值

3.3.2 例程

例程 3-5 控制轴初始化函数（SD、SE 、SG、SP 卡）

```
void AxisInitial()                    //控制轴初始化函数
{
    short rtn;
```

```
    for(int i=0;i<4;i++)
    {
      rtn=GT_Axis(i);       error(rtn);       //设置第 i 轴为当前轴
      rtn=GT_ClrSts();      error(rtn);       //清除当前轴不正确状态
      rtn=GT_StepPulse();   error(rtn);        //设置输出正负脉冲信号
      rtn=GT_AxisOn();      error(rtn);        //驱动使能
    }
}

void main()
{
  GTInitial();
  InputCfg();
  AxisInitial();
  }
```

例程 3-6　控制轴初始化函数（SV）

```
void AxisInitial()                  //控制轴初始化函数
{
    short rtn;

    rtn=GT_Axis(1);       error(rtn);       //设置第一轴为当前轴
    rtn=GT_ClrSts();      error(rtn);       //清除当前轴不正确状态
    rtn=GT_CtrlMode(0);   error(rtn);       //设置为输出模拟量
    rtn=GT_CloseLoop();   error(rtn);       //设置为闭环控制
        rtn=GT_SetKp(20);       error(rtn);     //设置比例增益 20
        rtn=GT_SetKi(0);        error(rtn);     //设置积分增益 0
        rtn=GT_SetKd(10);       error(rtn);      //设置微分增益 10
        rtn=GT_SetKvff(0);      error(rtn);     //设置速度前馈 0
        rtn=GT_SetKaff(0);      error(rtn);     //设置加速度前馈 0
        rtn=GT_SetMtrBias(10); error(rtn);     //设置输出零点偏移值为 10
        rtn=GT_Update();        error(rtn);      //参数刷新（参数生效）
   rtn=GT_AxisOn();        error(rtn);       //驱动使能

    rtn=GT_Axis(2);       error(rtn);       //设置第二轴为当前轴
    rtn=GT_ClrSts();       error(rtn);       //清除当前轴不正确状态
    rtn=GT_CtrlMode(0);   error(rtn);       //设置为输出模拟量
    rtn=GT_OpenLoop();    error(rtn);       //设置为开环控制
    rtn=GT_AxisOn();      error(rtn);       //驱动使能

    rtn=GT_Axis(3);       error(rtn);       //设置第三轴为当前轴
    rtn=GT_ClrSts();       error(rtn);       //清除当前轴不正确状态
    rtn=GT_CtrlMode(1);   error(rtn);       //设置输出脉冲量
    rtn=GT_StepDir();     error(rtn);       //设置输出脉冲+方向信号
    rtn=GT_AxisOn();      error(rtn);       //驱动使能


    rtn=GT_Axis(4);       error(rtn);       //设置第四轴为当前轴
    rtn=GT_ClrSts();       error(rtn);       //清除当前轴不正确状态
```

```
    rtn=GT_CtrlMode(1);   error(rtn);        //设置输出脉冲量
    rtn=GT_StepPulse();   error(rtn);        //设置输出正负脉冲信号
    rtn=GT_AxisOn();      error(rtn);        //驱动使能
  }


void main()
{
   GTInitial();
 InputCfg();
  AxisInitial();

  }
```

3.2.3 重点说明

设置当前轴

运动控制器可以同时控制四个控制轴，并且各控制轴可以独立设置参数。为了提高主机与运动控制器的通讯效率，运动控制器采用控制轴寻址的策略。

用户程序调用的单轴命令，都是作用于***当前轴***的。运动控制器默认的当前轴为第一轴。要想对其他轴发送命令，首先要调用***设置当前轴***命令：short GT_Axis(unsigned short num)。

GT_Axis()，将参数指定轴设置为当前轴。***此后调用的单轴命令都是针对当前轴，直到再次调用该函数将当前轴设置为另一个轴。***参数 num 表示指定的轴号，在 1、2、3、4 四个数中取值，分别代表第一、二、三、四轴。

设置输出“脉冲+方向”/“正负脉冲”

对于 SG、SE、SD 卡以及工作在脉冲输出模式下的 SV 卡，默认情况下，控制器输出“脉冲+方向”信号。用户可以调用函数 GT_StepPulse 设置控制器输出“正负脉冲”信号；调用函数 GT_StepDir 设置控制器输出“脉冲+方向”信号。

设置输出模拟量/脉冲量（以下说明仅对 SV 卡，非 SV 卡用户可跳过）

SV 控制器既可以输出模拟量，也可以输出脉冲量（不可同时输出）。控制器默认输出为模拟量。用户可以调用函数：short GT_CtrlMode(int mode)，设置输出模式。参数 Mode：0 表示为模拟量输出模式，1 表示为脉冲输出模式。

当输出设置为脉冲量时，可用 GT_StepDir 和 GT_StepPulse 设置输出方式，控制器默认输出为脉冲/方向模式。

设置为闭环/开环控制（仅 SV）：

SV 运控器有***闭环***和***开环***两种方式。

调用 GT_CloseLp()命令，当前轴工作在闭环方式，运动控制器将当前规划的运动位置、速度、加速度送入数字伺服滤波器，与反馈的实际位置进行比较获得控制输出信号。这种方式能够实现准确的位置控制。SV 运控器的默认控制方式为闭环控制方式。

调用 GT_OpenLp() 命令，当前轴工作在开环方式，允许主机通过 GT_SetMtrCmd()命令直接设置运动控制器的控制轴输出信号。这种方式主要用于只需转矩控制的运动或标定驱动器，运动控制器无法实现准确的位置控制。

设置数字伺服滤波参数（仅 SV）：

数字伺服滤波器用于计算控制输出信号。SV 运动控制器采用 ***PID*** 滤波器，外加速度和加速度前馈，即 PID+K_{vff}+K_{aff} 滤波器。通过调节各参数，该滤波器能对大多数系统实现精确而稳定的控制。伺服滤波器的参数可由主机设置。表 3-4 为数字伺服滤波器的参数定义。PID+K_{vff}+K_{aff} 滤波器原理如图 3-1 所示。

表 3-4 数字伺服滤波器的参数定义

符号	名称	参数 设置命令	参数 取值范围	参数 默认值
K_p	比例增益	GT_SetKp()	0 to 32,767	0
K_i	积分增益	GT_SetKi()	0 to 32,767	0
K_d	微分增益	GT_SetKd()	0 to 32,767	0
Lim	误差积分限	GT_SetILmt()	0 to 32,767	32,767
K_{vff}	速度前馈增益	GT_SetKvff()	0 to 32,767	0
K_{aff}	加速度前馈增益	GT_SetKaff()	0 to 32,767	0
B	输出静差补偿	GT_SetMtrBias()	-32768 to 32,767	0

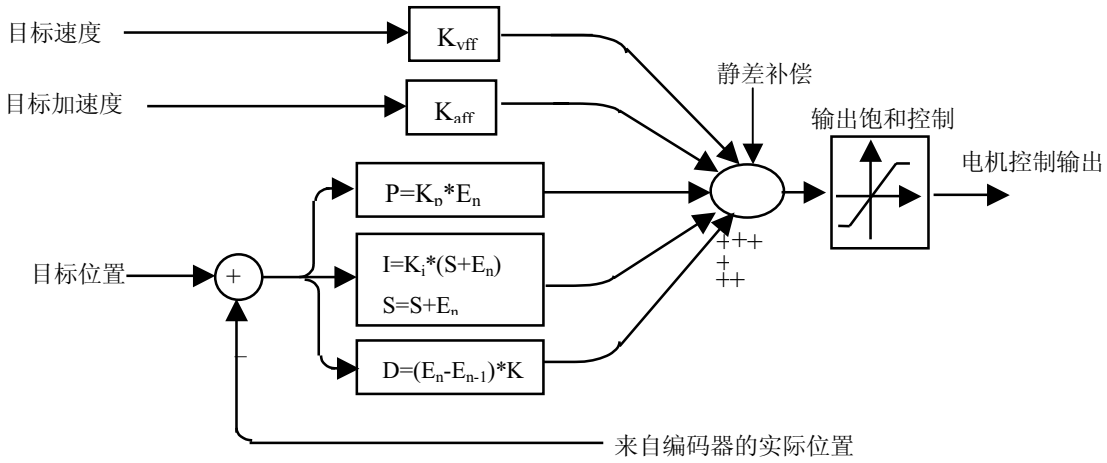

图 3-1 数字伺服滤波器原理图

数字伺服滤波器输出计算公式为：

$$E_n = \left(P_{target}\right)_n - \left(P_{actual}\right)_n$$

$$U_n = E_n K_p + \left(E_n - E_{n-1}\right)K_d + \left(\sum_n E_n\right)K_i / 256 + V_{target}K_{vff} + ACC_{target}K_{aff} + B$$

变量意义见表 3-5。

表 3-5 变量意义表

变量	意义
U_n	数字伺服滤波器输出值
E_n	第 n 个采样时刻的位置误差
P_{target}	第 n 个采样时刻的目标位置
P_{actual}	第 n 个采样时刻的实际位置
$\left(\sum_n E_n\right)$	第 n 个采样时刻的累积误差值
V_{target}	当前目标速度，单位：计数值/采样周期
ACC_{target}	当前目标加速度，单位：计数值/采样周期 ^{2}
B	电机静差补偿

第一次设定 Ki 增益时要小心。如果系统正在运行而积分项的值未知，把 Ki 设定为一非 0 值将引起突然的“跳跃”。为避免这种情况，需先把 LIM(积分限)设定为 0，Ki 设定为期望值，再设置 LIM 到期望的积分限。这样就清除了所有以前的积分值，从而使积分从前一个点开始平稳运算。

数字伺服滤波器的输出静差补偿主要用于补偿控制轴单方向外力的影响，如：机床垂直轴的重力。滤波器的输出静差补偿值可以通过主机命令设定，补偿值的范围为：-32768 到+32767。滤波器的输出静差补偿默认值为 0。

数字伺服滤波器输出的最大取值范围为+/-2^{15}，其对应的模拟量输出为+/-10V，而实际输出的取值范围受控制饱和值的限制。控制饱和值的大小（0-32767）决定了滤波器的有效输出范围。主机可以由 GT_SetMtrLmt()命令修改控制饱和值的大小，控制滤波器的有效输出范围，该值的默认值为 32767。

第四章 单轴运动

4.1 运动控制模式选择及相应运动参数设置

运动控制器针对单轴运动提供 4 种运动控制模式:***S-曲线模式***、***梯形曲线模式***、***速度控制模式***、***电子齿轮模式***。

某一种运动模式设定后，该轴将保持这种运动模式，直到设置新的运动模式为止。对于各模式之间的切换，除电子齿轮模式之外，其他模式必须是在当前轴运动完全停止的情况下进行。否则命令将被视为非法，控制器设置命令出错状态标志。控制器中不同的轴可以工作在不同的运动模式下。

对于不同的运动控制模式，需要设置不同的运动参数，具体使用参见以下的相应说明。

4.1.1 S-曲线模式

4.1.1.1 函数列表

表 4-1 S-曲线模式设置函数列表

函数	说明
GT_PrfIS()	设置当前轴的运动模式为 S-曲线模式
GT_SetJerk()	设置当前轴的加加速度
GT_SetMAcc()	设置当前轴的最大加速度
GT_SetVel()	设置当前轴的目标速度
GT_SetPos()	设置当前轴的目标位置

表 4-2 S-曲线模式参数设置范围 (ST：控制周期)

设置参数	数字范围	单 位
目标位置	-1,073,741,824 ~1,073,741,823	Pulse
加加速度	0~0.5（不含 0.5）	$Pulse/ST^3$
最大加速度	0~0.5（不含 0.5）	$Pulse/ST^2$
最大速度	0~16384	Pulse/ST

4.1.1.2 例程

例程 4-1 S-曲线模式运动

```
void SMotion()                          //S-曲线模式运动函数
{
```

```
short rtn;

rtn=GT_PrflS();       error(rtn);       // 设置运动模式为 S 曲线模式
rtn=GT_SetJerk(0.00000002);  error(rtn);    /* 设置最大加加速度为
                                              0.00000002 */
rtn=GT_SetMAcc(0.004); error(rtn);     // 设置最大加速度为 0.004
rtn=GT_SetVel(4);      error(rtn);     // 设置目标速度为 4
rtn=GT_SetPos(80000);  error(rtn);     // 设置目标位置为 80000
rtn=GT_Update();       error(rtn);     // 刷新参数
}

void main()
{
  GTInitial();
  InputCfg();
  AxisInitial();
  SMotion();
}
```

4.1.1.3 重点说明

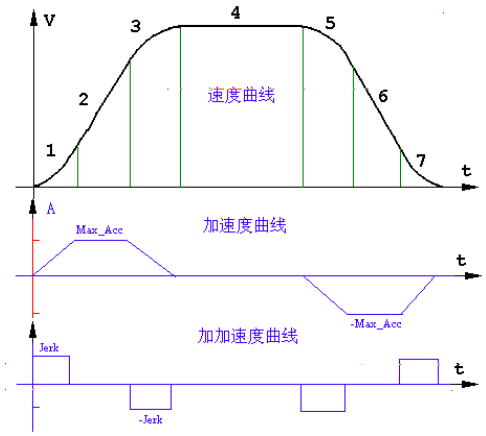

图 4-1　S-曲线模式的速度、加速度，加加速度曲线

图 4-1 所示为典型的 S-曲线模式的速度、加速度、加加速度的规划曲线，运动控制过程描述如下：

- 在开始的 1 区，加速度从零开始，以设定的最大加速度为目标，以加加速度 Jerk (单位时间内加速度的增量) 为增量递增，直到达到最大加速度为止；
- 在 2 区，加加速度为零，按已达到的最大加速度加速到第 3 区；
- 在第 3 区，按负的加加速度使加速度减为零值，使速度达到最大值。到此完成运动的加速过程；
- 第 4 段为匀速运行阶段，加速度和加加速度都为零；
- 在第 5、6、7 阶段与第 1、2、3 阶段类似，不同的是减速运行到速度为零。

在 S-曲线模式下，用户可以随时修改目标位置，其它参数在运动过程中不能修改。输入的最大速度、最大加速度和加加速度均是绝对值，控制轴的运动方向由目标位置决定。通常 S-曲线都是对称的，但可以缺少某些过程。如*图 4-2* 所示

的 S-曲线就缺少了第 4 段。

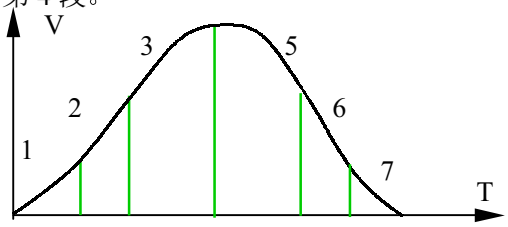

图 4-2 变形后的 S-曲线

4.1.2 梯形曲线模式

4.1.2.1 函数列表

表 4-3 梯形曲线模式设置函数列表

函数	说明
GT_PrflT()	设置当前轴的运动模式为梯形曲线模式
GT_SetAcc()	设置当前轴的加速度
GT_SetVel()	设置当前轴的最大速度
GT_SetPos()	设置当前轴的目标位置

表 4-4 梯形曲线模式参数设置范围(ST：控制周期)

设置参数	数字范围	单 位
加速度	0~16384	Pulse/ST ^{2}
最大速度	0~16384	Pulse/ST
目标位置	-1,073,741,824 ~1,073,741,823	Pulse

4.1.2.2 例程

例程 4-2 梯形曲线模式运动

```
void TMotion()                          //梯形曲线模式运动函数
{
  short rtn;
  rtn=GT_PrflT();       error(rtn); // 设置运动模式为梯形曲线模式
  rtn=GT_SetAcc(0.01); error(rtn); // 设置最大加速度为 0.01
  rtn=GT_SetVel(1);     error(rtn); // 设置目标速度为 1
  rtn=GT_SetPos(80000);error(rtn); // 设置目标位置为 80000
  rtn=GT_Update();      error(rtn); // 刷新参数
}

void main()
{
  GTInitial();
  InputCfg();
  AxisInitial();
  TMotion();
}
```

4.1.2.3 重点说明

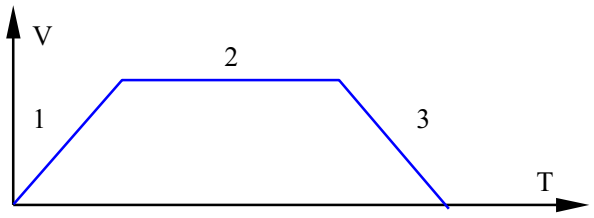

图 4-3 梯形曲线模式的速度曲线

图 4-3 描述梯形曲线模式的速度变化过程。一个典型的梯形速度曲线控制过程是：

- 第 1 阶段，速度按照设定的加速度值从零加速到最大速度；
- 第 2 阶段，加速度为零值，速度保持已达到的最大速度运行到第 3 段；
- 第 3 阶段，按设定的加速度减速到零，此时达到要求的目标位置。

在有些情况下，可能达不到最大设定速度就要减速，这样就没有第 2 阶段。在梯形曲线模式控制方式下，任意时刻都可改变速度和位置，此时速度曲线就如图 4-4 所示。

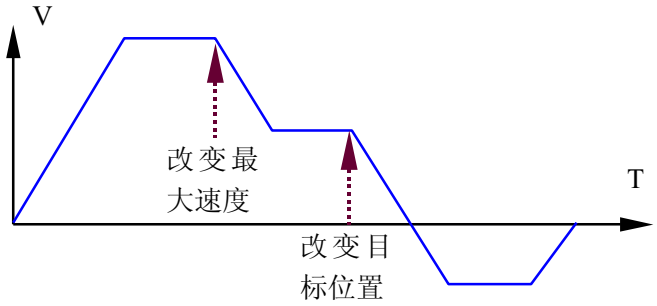

图 4-4 改变速度和位置后的梯形模式速度曲线

在调用 GT_PrflT()后，运动控制器将当前轴设定成梯形曲线模式。运动控制器按图 4-3 所示运动特征控制相应电机运动。用户需要设定目标位置、最大速度和加速度。相应函数见表 4-3。输入的最大速度和加速度均是绝对值，运动的方向由目标位置决定。梯形曲线模式中设定参数的范围见表 4-4 所示。

4.1.3 速度控制模式

4.1.3.1 函数列表

表 4-5 速度控制模式设置函数列表

函数	说明
GT_PrflV()	设置当前轴的运动模式为速度控制模式
GT_SetAcc()	设置当前轴的加速度
GT_SetVel()	设置当前轴的最大速度

表 4-6 速度控制模式参数设置范围(ST：控制周期)

设置参数	数字范围	单 位
最大速度	-16384~16384	Pulse/ST
加速度	0~16383	Pulse/ST ^{2}

4.1.3.2 例程

例程 4-3 速度曲线模式运动

```
void VMotion()                          //速度曲线模式运动函数
{
  short rtn;
  rtn=GT_PrflV();       error(rtn); // 设置运动模式为速度曲线模式
  rtn=GT_SetAcc(0.01); error(rtn); // 设置最大加速度为 0.01
  rtn=GT_SetVel(1);     error(rtn); // 设置目标速度为 1
  rtn=GT_Update();      error(rtn); // 刷新参数
}

void main()
{
  GTInitial();
  InputCfg();
  AxisInitial();
  VMotion();
}
```

4.1.3.3 重点说明

在调用 GT_PrflV()后，运动控制器将当前轴设定成速度控制模式。用户需要设定最大速度和加速度两个参数。在该模式下，开始运动时将以设定的加速度连续加速到设定的最大速度，运动方向由最大速度的符号确定，即正速度产生正向运动，而负速度则产生负向运动。在运动过程中，这两个运动参数可以随时修改。*表 4-5* 为速度控制模式的设置函数；*表 4-6* 为速度控制模式中设定参数的范围。

4.1.4 电子齿轮模式

4.1.4.1 函数列表

表 4-7 电子齿轮模式设置函数列表

函数	说明
GT_PrflG()	设置当前轴的运动模式为电子齿轮控制模式
GT_SetRatio()	设置当前轴的电子齿轮传动比

表 4-8 电子齿轮模式参数设置范围(ST：控制周期)

设置参数	数字范围	说明
主动轴号	1～6	1～4 为标准控制轴，5、6 轴为辅助编码器
电子齿轮传动比	-16384～16384	正表示与主动轴运动同向 负表示与主动轴运动反向

4.1.4.2 例程

例程 4-4　电子齿轮控制模式运动

```
void GMotion()                          //电子齿轮控制模式运动函数
{
   short rtn;
   rtn=GT_Axis(1);      error(rtn); //设置当前轴为第一轴
   rtn=GT_PrflG(2);     error(rtn); /* 设置运动模式为电子齿轮控制模式,
                                       主轴为 2 轴*/
  rtn=GT_SetRatio(-1); error(rtn); // 设置设置电子齿轮传动比为-1
   rtn=GT_Update();     error(rtn); // 刷新参数
}

void main()
{
   GTInitial();
  InputCfg();
   AxisInitial();
   GMotion();
}
```

4.1.4.3 重点说明

在调用 GT_PrflG()后，运动控制器将当前轴设定成电子齿轮模式（即当前轴为从轴），并指定跟随的主动轴轴号，能够设置其余三轴中的任意一个轴为电子齿轮的主动轴。也可以设置两个辅助编码器输入信号中的任何一个作为主动轴。

当前轴工作在电子齿轮模式下时，主机需设置一个参数，即电子齿轮传动比（从/主），其取值范围：-16384 至 16384。当前轴将按照这个速度比值，跟随主动轴运动。主动轴的运动模式可以是任何一种运动模式（如果主动轴处于坐标系运动模式，则当前轴将跟随主动轴的分运动，而不是坐标系合成运动）。当前轴运动位移增量等于与之相联系的主动轴的位移增量乘以电子齿轮传动比。

电子齿轮模式实际上是一个多轴联动模式，其运动效果与两个机械齿轮的啮合运动类似。

设定电子齿轮传动比的命令函数是：GT_SetRatio()。运动控制器允许一个主动轴带多个从动轴，或者从动轴作为主动轴再带从动轴运动的情况。如果任何一个从轴出现异常（如碰限位开关，驱动报警等），都将使主动轴运动停止。但是，如果主动轴是开环模式或者主动轴是辅助编码器，则从动轴异常只停止从动轴的运动，而不限制主动轴的运动。

注意	*当前轴进入电子齿轮模式以后，不论主轴有没有运动，当前轴都将始终处于正在运动状态（即当前轴状态寄存器中的 BIT10 位始终置位），直到用户将当前轴切换成别的运动模式，或者当前轴出现碰限位开关等异常情况。*

4.2 单轴运动停止

在某些情况下，为了安全起见，或为了实现某些特定的运动轨迹，需要在某些位置或某个时刻使运动停止。运动控制器对于单轴运动，提供两种方法实现这一功能：***急停***和***平滑停止***。

4.2.1 函数列表

表 4-9 单轴运动停止函数列表

函数	说明
GT_SmthStp()	平滑停止当前轴的运动
GT_AbptStp()	立即停止当前轴的运动

4.2.2 例程

例程 4-5 当检查到外部 IO 的 EXI15 口为高电平时，急停第一轴的运动。

```
void main()
{
    short rtn,ex_data;
    rtn=GT_ExInpt(&ex_data);   error(rtn); //读取输入端口的状态
    rtn=GT_Axis(1);            error(rtn); //设置第 1 轴为当前轴
     if(ex_data&0x8000)                    //判断 EXI15 是否为高电平
    {
        rtn=GT_AbptStp();      error(rtn); //急停运动
        }
}
```

4.2.3 重点说明

急停命令

GT_AbptStp()命令使单轴运动立即停止，同时将速度设为 0，加/减速控制器也停止工作。当不需要减速过程而希望紧急停止控制轴的运动时，通常采用该命令。

GT_AbptStp()命令在单轴运动的四种运动控制模式下都有效。

GT_AbptStp()命令时，应当非常小心，因为该命令会使电机突然停止，产生较大的冲击，会减少机构使用寿命。

平滑停止命令

GT_SmthStp()命令使单轴运动平滑终止。此时，控制轴运动以设定的加速度减速直到速度变为 0，减速过程与加速过程对称。*图 4-5* 为 S-曲线模式和梯形曲线模式下，平滑停止的速度曲线图。

GT_SmthStp()需与 GT_Update()或 GT_MltiUpdt()函数结合使用才能生效。GT_SmthStp()在电子齿轮模式下**无效**，在其他三种单轴运动控制模式下**有效**。

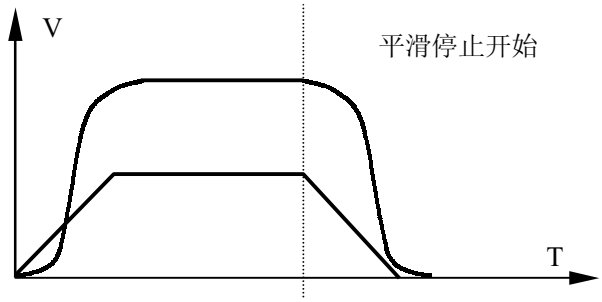

图 4-5 S-曲线模式和梯形曲线模式下平滑停止速度曲线图

4.3 单轴参数设置和刷新

在 *3.3 运动控制轴初始化*、 *4.1 单轴运动控制模式选择及相应运动参数设置* 以及 *4.2 单轴运动停止* 中都用到***参数刷新***，下面将详细讲述参数设置和刷新。参数刷新分为普通参数刷新和断点参数自动刷新。

为实现控制轴多个参数同时更新以及多个控制轴的参数同步更新，运动控制器采用***双缓冲机制***来完成单轴运动、控制参数的设置和生效。

双缓冲机制，当主机发出单轴运动、控制参数设置命令时，把这些令下载到运动控制器中。***参数刷新***之前，下载的参数和运动命令均不生效；***参数刷新***之后，控制器将在下一个控制周期将这些参数和命令复制到有效的寄存器中，使其同时生效。*表 4-10* 所列函数均采用双缓冲机制刷新。

表 4-10 双缓冲命令函数列表

函数	说明
GT_SetPos()	设置当前轴的目标位置
GT_SetBrkcn()	设置当前轴断点位置比较值
GT_SetVel()	设置当前轴的目标速度
GT_SetAcc()	设置当前轴的加速度
GT_SetMAcc()	设置当前轴的最大加速度
GT_SetJerk()	设置当前轴的加加速度
GT_SetRatio()	设置当前轴的电子齿轮比
GT_SetMtrLmt()	设置当前轴的伺服滤波器输出饱和值
GT_SetMtrBias()	设置当前轴的伺服滤波器输出零点偏移值
GT_SetKp()	设置当前轴的伺服滤波器比例增益
GT_SetKi()	设置当前轴的伺服滤波器积分增益
GT_SetKd()	设置当前轴的伺服滤波器微分增益
GT_SetKvff()	设置当前轴的伺服滤波器速度前馈增益
GT_SetKaff()	设置当前轴的伺服滤波器加速度前馈增益
GT_SetILmt()	设置当前轴的伺服滤波器误差积分饱和值
GT_SetPosErr()	设置当前轴的伺服滤波器位置误差极限
GT_SmthStp()	平滑停止当前轴的运动
GT_SynchPos()	立即停止当前轴的运动

4.3.1 普通参数刷新

4.3.1.1 函数列表

表 4-11 普通参数刷新函数列表

函数	说明
GT_SmthStp()	平滑停止当前轴的运动
GT_AbptStp()	立即停止当前轴的运动

4.3.1.2 例程

例程 4-6 普通参数刷新

```
void main()
{
  short rtn;
  GTInitial();
  InputCfg();
  AxisInitial();

  rtn=GT_Axis(1);  error(rtn);      //第 1 轴为当前轴
  VMotion();                        //速度控制模式运动
  delay(1000);                      //延时 1 秒
  rtn=GT_SmthStp(); error(rtn);     //平滑停止
  rtn=GT_Update();  error(rtn);     //参数刷新（平滑停止生效）

  rtn=GT_Axis(2);   error(rtn);      //第 2 轴为当前轴
  rtn=GT_SetAcc(0.01); error(rtn);  //加速度为 0.01 脉冲/控制周期^2
  rtn=GT_SetVel(1);    error(rtn);  //速度为 1 脉冲/控制周期
  rtn=GT_SetPos(20000); error(rtn); //目标位置 20000 脉冲

  rtn=GT_Axis(3);   error(rtn);      //第 3 轴为当前轴
  rtn=GT_SetAcc(0.01); error(rtn);  //加速度为 0.01 脉冲/控制周期^2
  rtn=GT_SetVel(1);    error(rtn);  //速度为 1 脉冲/控制周期
  rtn=GT_SetPos(20000); error(rtn); //目标位置 20000 脉冲

  rtn=GT_MltiUpdt(0x6); error(rtn);  /*多轴参数刷新（第 2、3 轴参数同
                                       时生效。2 轴和 3 轴实现运动同
                                       步。）*/
}
```

4.3.1.3 重点说明

GT_Update()命令实现当前轴立即刷新，使当前轴的运动参数或缓冲区命令生效。

GT_MltiUpdt()命令实现多轴运动参数（命令）立即刷新，即多个控制轴同步刷新参数（命令）。在希望多个控制轴的运动同步的情况下，主机可采用此命令来实现多控制轴运动同步。

4.3.2 断点参数自动刷新

运动控制器除提供了普通参数刷新命令外，还提供了断点参数自动刷新命令来实现控制器参数的自动刷新。主机可以设定某一条件（我们称为***断点***），当控制轴运动满足该条件时，控制器将自动刷新控制轴参数（命令）。设置断点参数自动刷新的函数见*表 4-12*：

表 4-12 断点参数刷新

函数	说明
GT_AuUpdtOn()	设置当前轴控制参数和命令自动更新
GT_AuUpdtOff()	关闭当前轴控制参数和命令自动更新
GT_GetBrkCn()	读取当前轴断点位置比较值
GT_PosBrk()	设置当前轴断点条件为大于断点位置
GT_NegBrk()	设置当前轴断点条件为小于断点位置
GT_ExtBrk()	设置当前轴原点信号触发断点模式
GT_MtnBrk()	设置当前轴运动到位触发断点模式
GT_BrkOff()	清除当前轴断点，并关闭断点模式

运动控制器默认***断点参数自动刷新***功能为***关闭***状态。主机可以通过GT_AuUpdtOn()、GT_AuUpdtOff()命令打开、关闭该功能。当前该功能处于打开还是关闭状态，用户可以调用函数 GT_GetMode()，读取轴模式寄存器的相应位来判断（详细解释参见 *4.5.2*）。

控制轴运动断点被触发一次后，控制器将清除断点。如需再次触发，主机须重新设置断点条件。同时，主机在设定断点后，在该断点未触发之前可以通过GT_BrkOff()命令清除该断点。

运动控制器提供了四种断点：正向目标位置、负向目标位置、轴运动完成事件、原点开关触发事件。下面将详细讲述四种断点参数自动刷新的使用方法。

4.3.2.1 正向目标位置断点

主机首先调用 GT_SetBrkCn()命令设定当前轴断点比较位置，并用 GT_Update()或 GT_MltiUpdt()命令刷新。然后主机发送 GT_PosBrk()命令设定当前轴断点为正向目标位置断点。当该轴的实际位置大于或等于断点比较值时，触发该断点并清除当前轴断点标志，然后查寻当前轴模式寄存器中的参数自动刷新标志，标志位为 1 则自动刷新轴参数，否则不刷新轴参数，同时无论是否允许参数自动刷新，都将当前轴状态寄存器中的断点到达标志位置起。

例程 4-7 正向目标位置断点设置及触发

本例程实现第一轴以速度 1(Pulse/ST)运动到为 20000 后，改变速度为4(Pulse/ST)运动到位置 50000。

```
void main()
{
  GTInitial();
  InputCfg();
```

```
  AxisInitial();

  rtn=GT_Axis(1);       error(rtn);  //设置当前轴为第 1 轴
  rtn=GT_ClrSts();      error(rtn);  //清除状态位
  rtn=GT_SetVel(1);     error(rtn);  //设置速度为 1(Pulse/ST)
  rtn=GT_SetAcc(0.1);   error(rtn);  //设置加速度为 0.1(Pulse/ST^2)
  rtn=GT_SetPos(50000); error(rtn);  //设定目标位置为 50000
   rtn=GT_AuUpdtOn();    error(rtn);   // 打开断点参数自动刷新功能
  rtn=GT_SetBrkCn(20000); error(rtn);  //设置断点位置
  rtn=GT_Update();      error(rtn);    //刷新参数（断点位置生效）
  rtn=GT_PosBrk();      error(rtn);    //设置条件为大于断点位置
  rtn=GT_SetVel(4);      error(rtn);  //设置速度为 4（等待自动刷新）
}
```

4.3.2.2 负向目标位置断点

主机首先调用 GT_SetBrkCn()命令设定当前轴断点目标位置，并用 GT_Update()或 GT_MltiUpdt()命令刷新。然后主机发送 GT_NegBrk()命令设定当前轴断点为负向位置断点。当该轴的实际位置小于或等于断点比较值时，触发该断点并清除当前轴断点标志，然后查寻当前轴模式寄存器中的参数自动刷新标志，标志位为 1 则自动刷新轴参数，否则不刷新轴参数，同时无论是否允许参数自动刷新，都将当前轴状态寄存器中的断点到达标志位置起。

例程 4-8　负向目标位置断点设置及触发

本例程实现第一轴以速度 1（Pulse/ST）运动到为-20000 后，改变速度为 4（Pulse/ST）运动到位置-50000。

```
void main()
{
  GTInitial();
  InputCfg();
  AxisInitial();

  rtn=GT_Axis(1);       error(rtn);  //设置当前轴为第 1 轴
  rtn=GT_ClrSts();      error(rtn);  //清除状态位
  rtn=GT_SetVel(1);     error(rtn);  //设置速度为 1(Pulse/ST)
  rtn=GT_SetAcc（0.1); error(rtn);  //设置加速度为 0.1（Pulse/ST^2）
  rtn=GT_SetPos(-50000); error(rtn); //设定目标位置为-50000
   rtn=GT_AuUpdtOn();    error(rtn);   // 打开断点参数自动刷新功能
  rtn=GT_SetBrkCn(-20000); error(rtn);  //设置断点位置
  rtn=GT_Update();      error(rtn);    //刷新参数（断点位置生效）
  rtn=GT_NegBrk();      error(rtn);    //设置条件为小于断点位置
  rtn=GT_SetVel(4);      error(rtn);  //设置速度为 4（等待自动刷新）
}
```

4.3.2.3 轴运动完成事件断点

主机发送 GT_MtnBrk()命令设定当前轴断点为运动完成断点。当该轴当前执行的运动完成，且状态寄存器中轴运动完成标志位置 1 时触发该断点并清除当前轴断点标志。然后查寻当前轴模式寄存器中的参数自动刷新标志，标志位为 1 则自动刷新轴参数，否则不刷新轴参数。同时无论是否允许参数自动刷新，都将当前

轴状态寄存器中的断点到达标志位置起。

例程 4-9　运动完成断点设置及触发

本例程实现第一轴以速度 1(Pulse/ST)运动到为 20000 后，改变速度为 4(Pulse/ST)运动到位置 0。

```
void main()
{
  GTInitial();
  InputCfg();
  AxisInitial();

  rtn=GT_Axis(1);       error(rtn);  //设置当前轴为第 1 轴
  rtn=GT_ClrSts();      error(rtn);  //清状态位
  rtn=GT_SetVel(1);     error(rtn);  //设置速度为 1(Pulse/ST)
  rtn=GT_SetAcc(0.1);   error (rtn) ;//设置加速度为 0.1 (Pulse/ST²)
  rtn=GT_SetPos(20000); error(rtn);  //设定目标位置为 20000
  rtn=GT_AuUpdtOn();    error(rtn);   // 打开断点参数自动刷新功能
  rtn=GT_SetMtnBrk();   error(rtn);   //设置运动完成断点模式
  rtn=GT_SetVel(4);     error(rtn);  //设置速度为 4（等待自动刷新）
  rtn=GT_SetPos(0);     error(rtn);  //设置目标位置 0（等待自动刷新）
}
```

4.3.2.4 原点开关触发事件断点

主机发送 GT_ExtBrk()命令设定当前轴断点事件为原点开关触发。主机发送原点捕获命令后，当该轴运动状态寄存器中 Index/Home 标志位置 1 时，触发该断点并清除当前轴断点标志，然后查寻当前轴模式寄存器中的参数自动刷新标志，标志位为 1 则自动刷新轴参数，否则不刷新轴参数。同时无论是否允许参数自动刷新，都将当前轴状态寄存器中的断点到达标志位置起。

例程 4-10　原点开关触发事件断点设置及触发

本程序将使第 1 轴在设定的运动中捕获原点事件，当控制轴的原点被捕获后，断点将触发控制轴平滑停止命令生效，使当前轴运动平滑停止。同时被捕获的原点位置将存入位置捕获寄存器。

```
void main()
{
  GTInitial();
  InputCfg();
  AxisInitial();
  rtn=GT_Axis(1);       error(rtn);  //设置当前轴为第 1 轴
  rtn=GT_ClrSts();      error(rtn);  //清除状态位 1
  rtn=GT_AuUpdtOn();    error(rtn);   // 打开断点参数自动刷新功能
  rtn=GT_CaptHome();    error(rtn);    //设置原点捕获
  rtn=GT_ExtBrk();      error(rtn);    //设置原点开关触发事件断点
  rtn=GT_PrflV();       error(rtn);    //设置为速度控制模式
  rtn=GT_SetVel(1);     error(rtn);    //设置速度为 1(Pulse/ST)
  rtn=GT_Update();      error(rtn);    //刷新参数
  rtn=GT_SmthStp();     error(rtn);  //平滑停止（等待自动刷新）
  }
```

4.4 单轴设置目标位置、实际位置

4.4.1 函数列表

表 4-13 单轴设置目标位置、实际位置函数列表

函数	说明
GT_SetPos()	设置当前轴的目标位置
GT_ZeroPos()	当前轴实际位置和目标位置清零
GT_SynchPos()	设置当前轴的目标位置等于实际位置
GT_SetAtlPos()	设置当前轴的实际位置

4.4.2 重点说明

设置当前轴的目标位置

调用函数 short GT_SetPos (long Pos)，设置在 S-曲线模式和梯形曲线模式下当前轴的目标位置。

参数 Pos 是需要设置的目标位置值，其取值范围为 –1073741824～1073741823。位置的单位为***脉冲数***。该函数设置的参数，需调用函数 GT_Update()或 GT_MltiUpdt()后才有效。

清零实际位置和目标位置

调用函数 short GT_ZeroPos()，将当前轴的实际位置寄存器和目标位置以及当前控制周期的规划位置寄存器设为零值。本函数只在当前轴运动停止有效，否则将被视为非法命令产生命令出错标识。本函数在当前轴处于电子齿轮运动模式时使用无效。

设置当前轴的目标位置等于实际位置

调用函数 short GT_SynchPos(void)，将当前控制轴的目标位置寄存器和当前伺服周期的规划位置寄存器设置为实际位置寄存器的位置值。该函数适应于 S-曲线和梯形曲线控制模式，可用于在运动控制产生了错误又需重新启动时，令目前位置控制寄存器的值等于实际位置寄存器的值。同时当前轴在进行驱动使能（关闭使能）或闭环（开环）状态切换时，也可以用该函数确保电机平稳过渡。该函数需与 GT_Update()或 GT_MltiUpdt()函数结合使用才能生效。本函数在当前轴处于电子齿轮运动模式时使用无效。

设置当前轴的实际位置

调用函数 short GT_SetAtlPos(long actl_pos)，将当前轴的实际位置和目标位置以及规划位置修改为设定值。该函数只在当前轴运动停止时有效，否则将被视为非法命令产生命令出错标识。该函数在当前轴处于电子齿轮运动模式时使用无效。参数 actl_pos 表示要设置的实际位置值。

4.5 单轴状态

4.5.1 轴状态寄存器

运动控制器为每一个控制轴提供一个 16 位的状态寄存器。在运动过程中，用户可以通过调用函数 GT_GetSts()查询这些状态全面了解当前轴的运动情况。

4.5.1.1 寄存器定义

表 4-14 轴状态寄存器定义

位	定义
0	运动完成标志位。如果控制轴运动完成，该位置 1。该标志在电子齿轮运动模式时无效。
1	电机伺服驱动器报警标志位。如果控制轴驱动器报警，该位置 1。
2	断点到达标志位。在设置断点条件下，断点条件满足，该位置 1。
3	Index/Home 标志位。在设置位置捕获命令后，控制器检测到要求的 Index/Home 捕获条件，该位置 1。
4	运动出错标志位。如果位置误差超过允许范围(参考 4.4.7.3 说明)，控制器将该位置 1。只有控制轴不再处于运动出错状态时，才能对它复位。
5	正向限位开关触发标志位。如果正向限位开关被触发，该位置 1。
6	负向限位开关触发标志位。如果负向限位开关被触发，该位置 1。
7	命令出错标志位。如果出现命令错误，控制器将该位置 1。
8	电机开环/闭环状态(1 表示闭环，0 表示开环)。
9	电机伺服使能/禁止状态(1 表示使能，0 表示禁止)。
10	运动状态标志位。它连续指示控制轴是否在运动。如果在运动，为 1；如果静止，为 0。
11	限位开关使能/禁止状态（1 表示使能，0 表示禁止)。
12 13	当前轴号标志(13bit=高位,12bit=低位)。 当前轴号的编码如下 Bit 13 Bit12 轴 0 0 1 0 1 2 1 0 3 1 1 4
14	设定 Home 开关信号捕获标志
15	设定 Index 信号捕获标志

4.5.1.2 说明

状态寄存器的各标志位定义如表 4-14 所示。其中标志位 8-15 指示控制轴的运行状态信息和编号，主机不能对它们进行复位，而 0-7 位表示控制轴不同的事件状态。这些事件一旦发生，相应的标志位被置 1，并一直保持。直到主机调用 GT_ClrSts()、GT_RstSts()函数，相应的标志位将被清除。而且，0-6 位同时能向主机提出中断申请。

为了简化运动控制系统的编程，控制轴状态寄存器中提供了两个标志位表征轴运动状态信息。其中一个标志位表征控制轴运动完成事件是否发生（第 0 位，运动完成标志位），另一标志位表征控制轴当前的运动状态（第 10 位，运动状态

标志位）。

主机和运动控制器均可修改控制器各控制轴的运动完成标志位。在某控制轴运动完成后，控制器设置该控制轴的运动完成标志位，此后主机可查询该标志位以确定运动是否完成；如果用户需要，也可通过主机编程使运动控制器在控制轴运动完成时自动向主机发出控制轴运动完成中断申请。在上述任何一种方式下，一旦主机识别到控制器某控制轴运动完成标志，主机应清除控制器相应控制轴的运动完成标志位，以便控制器标志下一次运动完成状态。运动控制器在下列几种情况下可产生运动完成状态标志：

- 运动到位。
- 在速度控制模式下，运动轴的实际速度和命令速度同时为 0。
- GT_SmthStp()命令生效后，控制轴运动速度到 0。
- 发出 GT_AbptStp()命令后。
- 在 GT_LmtsOn()的情况下限位开关动作。
- 在允许超差自动停方式下，控制轴运动出错时自动停止运动。
- 控制轴驱动器发生报警。
- 发出 GT_AxisOn()命令将当前轴驱动使能。

运动状态标志位与运动完成标志位类似。不同的是它仅连续的指示控制轴的状态，供主机查询而不能引发中断，也不能被主机修改。

控制轴运动状态标识位和运动完成标志位表征的只是控制器内部加/减速运动规划的状态，而不是控制轴的实际状态。即只表征控制器的加/减速控制是否已完成一次运动任务，到达目标位置。而控制轴是否到达实际位置还取决于实际系统的伺服滞后，稳定性及其它条件。

运动完成标志位在电子齿轮模式中无效。而运动状态标志位在一进入电子齿轮模式时就被置起，不管当前控制轴是否在运动。

4.5.2 轴模式寄存器

运动控制器提供一个模式寄存器表征其工作模式或工作条件。该寄存器可通过命令 GT_GetMode()查询其内容，其标志位定义如*表 4-15* 所示。

表 4-15 轴模式寄存器定义

位	定义
0-6	内部使用。
7	运动出错停止标志位。GT_AuStpOn() 和 GT_AuStpOff()命令可修改该标志位。标志位为 1 表示电机运动出错后，自动关闭电机伺服使能，电机运动停止。
8-9	保留
10	自动刷新标志位。GT_AuUpdtOn()，GT_AuUpdtOff()命令可修改该标志位。标志位为 1，表示控制器在断点条件满足后自动刷新控制轴参数。
11-13	控制轴运动模式标志，编码如下: Bit13 Bit12 Bit11 运动模式 0 0 0 梯形曲线模式 0 0 1 速度控制模式 0 1 0 S-曲线模式 0 1 1 电子齿轮模式 1 0 1 坐标运动模式
14-15	内部使用

第五章 多轴协调运动

运动控制器可以实现两种轨迹的多轴协调运动：直线插补、圆弧插补。描述复杂的多轴协调运动轨迹的最简单的方法是利用坐标系，在坐标系内能够方便地描述运动对象的运动轨迹。因此，在本手册**多轴协调运动**又称为**坐标系运动**；多轴协调运动模式又称为**坐标系运动控制模式**。

运动控制器通过**坐标映射**将控制轴由单轴运动控制模式转换为坐标系运动控制模式。在坐标系运动控制模式下，可以实现**单段轨迹运动**，**多段轨迹连续运动**。运动控制器开辟了底层**运动数据缓冲区**，可以实现多段轨迹快速、稳定的连续运动。

5.1 坐标映射

运动控制器利用一个四维坐标系（X Y Z A），描述直线、圆弧插补轨迹。其中在使用圆弧插补命令时，$X-Y-Z$ 三个轴构成图 5-1 所示的右手坐标系。用户也可以只利用二维（$X-Y$）、三维（$X-Y-Z$）坐标系描述运动轨迹。

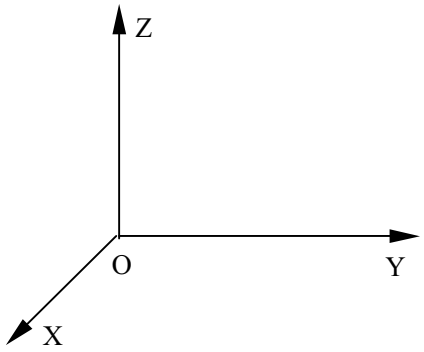

图 5-1　右手坐标系

用户通过调用 GT_MapAxis()命令将在坐标系内描述的运动通过映射关系映射到相应的轴上。从而建立各轴的运动和要求的运动轨迹之间的运动学传递关系。运动控制器根据坐标映射关系，控制各轴运动，实现要求的运动轨迹。调用 GT_MapAxis()命令时，所映射的各轴必须处于静止状态。

坐标映射命令函数原型是：short GT_MapAxis(short Axis_Num, double * map_count)。Axis_Num 为轴号（1、2、3 或 4），调用坐标映射命令以后，该轴工作于坐标运动模式。该轴的实际位置记为 $Axis\_N$，单位是脉冲。数组 map_count 包括五个元素，顺次记为 C_x、C_y、C_z、C_a、C，坐标轴 X、Y、Z、A 所对应的相应坐标记为 x、y、z、a。上述函数描述的映射关系能够简单地描述成下面的计算公式：

$$Axis\_N = C_x \times x + C_y \times y + C_z \times z + C_a \times a + C$$

由此可以看出被映射的控制轴的运动是坐标 X、Y、Z、A 的线性组合。

例程 5-1　最简单的坐标映射

在本例中实现最简单的坐标映射即：

1 号轴 = X
2 号轴 = Y
3 号轴 = Z
4 号轴 = A

```
void MapAxis()                          //坐标映射函数
{
    short rtn;
    double cnt1[5]={1,0,0,0,0};  /* 根据系统设置坐标映射数组 */
    double cnt2[5]={0,1,0,0,0};  /* 根据系统设置坐标映射数组 */
    double cnt3[5]={0,0,1,0,0};  /* 根据系统设置坐标映射数组 */
    double cnt4[5]={0,0,0,1,0};  /* 根据系统设置坐标映射数组 */
    rtn=GT_MapAxis(1,cnt1);  error(rtn);  /* 映射第 1 轴到 X 轴 */
    rtn=GT_MapAxis(2,cnt2);  error(rtn);  /* 映射第 2 轴到 Y 轴 */
    rtn=GT_MapAxis(3,cnt3);  error(rtn);  /* 映射第 3 轴到 Z 轴 */
    rtn=GT_MapAxis(4,cnt4);  error(rtn);  /* 映射第 4 轴到 A 轴 */
}

void main()
{
  GTInitial();
  InputCfg();
  AxisInitial();
  MapAxis();
  }
```

例程 5-2　单位换算坐标映射

该程序示例一个单位换算的坐标映射。已知电机每转脉冲数为 8000 脉冲，连接的丝杠螺距是 4 毫米。使用如下的坐标映射关系：

1 号轴 = 2000 X
2 号轴 = 2000 Y
3 号轴 = 2000 Z
4 号轴 = 2000 A

这样在描述坐标系下的各坐标轴运动时，单位是 1 毫米。由运动控制器自动实现单位换算。

```
void main()
{
  short rtn;
```

```
GTInitial();
InputCfg();
AxisInitial();
double cnt1[5]={2000,0,0,0,0};  /* 根据系统设置坐标映射数组 */
double cnt2[5]={0,2000,0,0,0};  /* 根据系统设置坐标映射数组 */
double cnt3[5]={0,0,2000,0,0};  /* 根据系统设置坐标映射数组 */
double cnt4[5]={0,0,0,2000,0};  /* 根据系统设置坐标映射数组 */
rtn=GT_MapAxis(1,cnt1);  error(rtn);/* 映射第 1 轴到 X 轴 */
rtn=GT_MapAxis(2,cnt2);  error(rtn);/* 映射第 2 轴到 Y 轴 */
rtn=GT_MapAxis(3,cnt3);  error(rtn);/* 映射第 3 轴到 Z 轴 */
rtn=GT_MapAxis(4,cnt4);  error(rtn);/* 映射第 4 轴到 A 轴 */
}
```

例程 5-3　倾斜补偿坐标映射

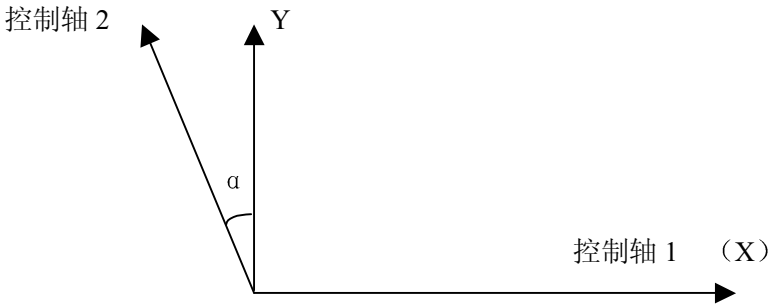

图 5-2 坐标映射示例

该程序示例坐标映射功能的一个应用。

如图 5-2 所示，如果控制轴 1 和控制轴 2 所对应的运动方向由于某种原因不垂直。为了利用直角坐标系 X-O-Y 描述运动轨迹，用户可以通过简单变换得到如下的坐标映射关系：

1 号轴 = X + Y tanα

2 号轴 = Y/cosα

使用上述映射关系，用户能够方便地直接在直角坐标系下描述运动轨迹。运动控制器自动地完成倾斜补偿运算。这个示例可以应用于补偿两垂直导轨存在安装误差的应用中。

```
void MapAxis()                          //坐标映射函数
{
    short rtn;
    double cnt1[5]={1,0,0,0,0};
    double cnt2[5]={0,0,0,0,0};
    cnt1[1]=tan(3),cnt2[1]=1/cos(3);    /* 这里倾斜角度为 3^0 */
    rtn=GT_MapAxis(1,cnt1);      error(rtn);
    rtn=GT_MapAxis(2,cnt2);      error(rtn);
}
```

```
void main()
{
  GTInitial();
 InputCfg();
  AxisInitial();
  MapAxis();
  }
```

通过上述 3 例用户可以了解坐标映射的基本原理。以次类推，坐标平移、坐标旋转、坐标比例变换（利用坐标比例变换能够方便地实现长度单位转换）等计算功能都能够通过坐标映射实现。

坐标系运动控制模式与***梯形曲线、S-曲线、速度控制、电子齿轮控制运动模式***的关系：

- 坐标系运动有其专用的运动命令函数，单轴运动命令函数对其无效；
- 5 种运动控制模式可以相互切换，坐标系运动控制模式在无坐标系轨迹运动（状态监测可知）时，可以切换到其他运动控制模式；
- 坐标系映射命令不修改面向控制轴命令的当前轴。

注意	*如果用户输入了矛盾的映射关系，例如将两个控制轴都映射到坐标系 X 轴上，运动控制器将根据控制轴编号小的映射关系计算坐标值。然后根据坐标映射关系，改变编号高的控制轴的实际位置，使之满足坐标映射关系。这将出现某个轴突然运动的情况。*

提示	*为防止这种矛盾的坐标映射引起电机突跳，建议用户输入的坐标映射关系最好满足独立不相关的条件。*

提示	*对于复杂的坐标映射关系，请与固高科技联系，我们能够根据用户的要求，为用户定制特殊的接口函数，满足特定的应用要求。*

5.2 坐标系运动合成速度、合成加速度设置

5.2.1 函数列表

表 5-1 设置坐标系运动合成速度、合成加速度函数

函数	说明
GT_SetSynVel()	设置坐标系运动合成速度
GT_SetSynAcc()	设置坐标系运动合成加速度

5.2.2 例程

例程 5-4 设置坐标系运动合成速度、合成加速度

该程序示例一个坐标系运动的合成速度、合成加速度的设置及单位说明。

本例利用例程 5-2 的单位换算坐标映射，将坐标系长度单位映射为 1mm。设置合成加速度为 3m/min，合成加速度 0.9m/min^2，控制周期为默认值 200us。

3m/min＝3000/300000(mm/ST)＝0.01mm/ST

0.9m/min^2＝900/(9x10^{10})(mm/ST^2)=1x10^{-8}(mm/ST^2)

```
void main()
{
  short rtn;
  GTInitial();
  InputCfg();
  AxisInitial();
  double cnt1[5]={2000,0,0,0,0};  /* 根据系统设置坐标映射数组 */
  double cnt2[5]={0,2000,0,0,0};  /* 根据系统设置坐标映射数组 */
  double cnt3[5]={0,0,2000,0,0};  /* 根据系统设置坐标映射数组 */
  double cnt4[5]={0,0,0,2000,0};  /* 根据系统设置坐标映射数组 */
  rtn=GT_MapAxis(1,cnt1);  error(rtn);/* 映射第 1 轴到 X 轴 */
  rtn=GT_MapAxis(2,cnt2);  error(rtn);/* 映射第 2 轴到 Y 轴 */
  rtn=GT_MapAxis(3,cnt3);  error(rtn);/* 映射第 3 轴到 Z 轴 */
  rtn=GT_MapAxis(4,cnt4);  error(rtn);/* 映射第 4 轴到 A 轴 */
  rtn=GT_SetSynAcc(0.00000001); error(rtn); /* 设置合成加速度为
                                          0.9(m/min2) */
  rtn=GT_SetSynVel(0.01); error(rtn); /*设置合成速度为 3(m/min) */
  }
```

5.2.3 重点说明

short GT_SetSynVel(double Vel); 该函数设置轨迹段的合成运动速度。参数 Vel 是设定的合成速度值，是坐标系各坐标轴分速度的矢量和（为正值）。其单位是**坐标系长度单位/控制周期**。该函数影响此后调用的所有直线插补和圆弧插补函数的速度，直到再次调用此函数为止。

合成速度：$V = \sqrt{V_x^2 + V_y^2 + V_z^2 + V_A^2}$

short GT_SetSynAcc(double Accel);该函数设置坐标系运动中轨迹段的合成加速度。参数 Accel 是设定的合成加速度值，是坐标系映射各轴分加速度的矢量和(均为正值)。其单位是**坐标系长度单位/控制周期** ^{2}。该函数影响此后调用的所有直线插补和圆弧插补函数的加速度，直到再次调用此函数为止。

合成加速度：$Acc = \sqrt{Acc_x^2 + Acc_y^2 + Acc_z^2 + Acc_A^2}$

*合成速度与加速度的单位**坐标系长度单位**（即 X,Y,Z,A 的长度单位），与坐标映射时设置的坐标映射参数有关。即合成速度、加速度设置是对应 X,Y,Z,A 的而不是轴 1,2,3,4 的。*

5.3 坐标系轨迹运动设置

5.3.1 函数列表

表 5-2 坐标系轨迹运动函数列表

函数	说明
GT_LnXY()	两维直线插补
GT_LnXYZ()	三维直线插补
GT_LnXYZA()	四维直线插补
GT_ArcXY()	XY 平面圆弧插补（以圆心位置和角度为输入参数）
GT_ArcXYP()	XY 平面圆弧插补（以终点位置和半径为输入参数）
GT_ArcYZ()	YZ 平面圆弧插补（以圆心位置和角度为输入参数）
GT_ArcYZP()	YZ 平面圆弧插补（以终点位置和半径为输入参数）
GT_ArcZX()	ZX 平面圆弧插补（以圆心位置和角度为输入参数）
GT_ArcZXP()	ZX 平面圆弧插补（以终点位置和半径为输入参数）

5.3.2 例程

例程 5-5 坐标系轨迹运动实现

本例在例程 5-4 的基础上加发轨迹设置命令，实现多轴协调运动立即执行。

```
void main()
{
  short rtn;
  GTInitial();
  InputCfg();
  AxisInitial();
  double cnt1[5]={2000,0,0,0,0};
  double cnt2[5]={0,2000,0,0,0};
  double cnt3[5]={0,0,2000,0,0};
  double cnt4[5]={0,0,0,2000,0};
  rtn=GT_MapAxis(1,cnt1);  error(rtn);
  rtn=GT_MapAxis(2,cnt2);  error(rtn);
  rtn=GT_MapAxis(3,cnt3);  error(rtn);
  rtn=GT_MapAxis(4,cnt4);  error(rtn);
  rtn=GT_SetSynAcc(0.00000001); error(rtn);
  rtn=GT_SetSynVel(0.01); error(rtn);
  //以上参考例程 5-4
  rtn=GT_LnXY(10,10);     //两轴直线插补运动到（10 毫米，10 毫米）
  }
```

5.3.3 重点说明

圆弧插补的旋转正方向按照右手螺旋定则定义为：从坐标平面的“上方”（即垂直于坐标平面的第三个轴的正方向）看，逆时针方向为正（图 1-9）。可以这样简单记忆：将右手拇指前伸，其余四指握拳，拇指指向第三个轴的正方向，其余四指的方向即为旋转的正方向。映射坐标系为二维坐标系（X-Y）时，XOY 坐标平面内的圆弧插补正方向同样定义。

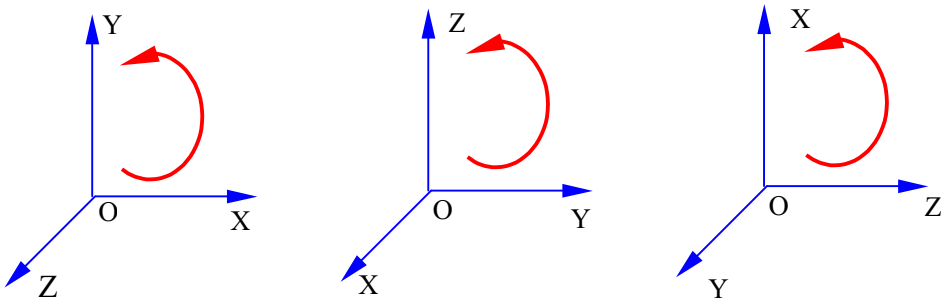

图 5-3 圆弧插补正方向

5.4 多段坐标系轨迹连续运动实现

在例程 5-5 中，如果用户希望实现多段轨迹连续运动，在最后一行 rtn=GT_LnXY(10,10)后，再加一条命令如“rtn=GT_LnXYZ(20,20,20)”。运行后用户会发现加的直线运动没有实现，查看后一条命令返回值为 1（错误命令）。这是因为运动控制器在“坐标系轨迹运动未完成”状态下，不接受新的坐标系轨迹运动命令。只有在前一轨迹运动结束后，才可以发送新的坐标系轨迹运动命令。

为了方便得实现多段坐标系轨迹连续运动，运动控制器提供一个大小为 4k 字的***缓冲区***。用户能够先将部分坐标系运动命令存放在这个控制器内部的循环队列命令缓冲区，然后发出执行命令。在运动控制器执行缓冲区内存放的运动命令的同时，主机能够继续向这个缓冲区内下载运动命令。这样就降低了对主机通讯实时性的要求，又提高了通讯效率。而运动控制器通过对缓冲区内的连续段运动轨迹信息的预处理，能够获得良好的运动特性。

下面分四部分讲述利用缓冲区实现连续坐标系轨迹运动：坐标系运动命令存入缓冲区；执行缓冲区中的坐标系运动。

5.4.1 坐标系运动命令存入缓冲区

5.4.1.1 函数列表

表 5-3 缓冲区管理命令函数列表

函数	说明
GT_StrtList()	打开并清空缓冲区
GT_MvXY()	定位缓冲区坐标起点（二维）
GT_MvXYZ()	定位缓冲区坐标起点（三维）
GT_MvXYZA()	定位缓冲区坐标起点（四维）
GT_AddList()	再次打开缓冲区
GT_EndList()	关闭缓冲区

可以放入缓冲区的命令，见表 5-4：

表 5-4 可以放入缓冲区的命令列表

函数	说明
GT_SetSynVel()	设置多轴协调运动轨迹切线速度
GT_SetSynAcc()	设置多轴协调运动轨迹切线加速度
GT_LnXY()	两维直线插补
GT_LnXYZ()	三维直线插补
GT_LnXYZA()	四维直线插补
GT_ArcXY()	XY 平面圆弧插补（以圆心位置和角度为输入参数）
GT_ArcXYP()	XY 平面圆弧插补（以终点位置和半径为输入参数）
GT_ArcYZ()	YZ 平面圆弧插补（以圆心位置和角度为输入参数）
GT_ArcYZP()	YZ 平面圆弧插补（以终点位置和半径为输入参数）
GT_ArcZX()	ZX 平面圆弧插补（以圆心位置和角度为输入参数）
GT_ArcZXP()	ZX 平面圆弧插补（以终点位置和半径为输入参数）

5.4.1.2 重点说明

打开并清空缓冲区、定位缓冲区坐标起点

GT_StrtList()，打开缓冲区并将缓冲区未执行的命令清空。当运动控制器上电后第一次采用坐标系运动模式使，若用户希望启动命令缓冲区策略，必须先调用GT_StrtList()命令进入缓冲区命令输入状态。

运动控制器规定，打开并清空缓冲区（GT_StrtList）后，**必须**紧接定位缓冲区坐标起点命令（GT_MvXY、GT_MvXYZ 或 GT_MvXYZA）。用户启动缓冲区中的命令后，运动控制器将从当前位置按照直线插补的方式移动到该命令指定位置，并将速度减速到零，然后顺序执行后续的缓冲区命令。

GT_MvXY()、GT_MvXYZ()、GT_MvXYZA()包含速度和加速度参数，这些参数作为此后输入缓冲区的命令的速度和加速度，直到设置新的速度和加速度命令为止。

关闭缓冲区

调用 GT_EndList()命令，关闭缓冲区，结束对缓冲区的操作。该命令在缓冲区处于打开状态下有效。

再次打开缓冲区

调用 GT_AddList()命令，打开已经关闭的缓冲区，接续原有的坐标系运动命令，增加新的坐标系运动命令。

在坐标系运动命令发送完毕后，可以再次调用 GT_EndList()命令，关闭缓冲区。用于增加缓冲区内的坐标系命令，用户可多次调用 GT_AddList()。

坐标系运动命令存入缓冲区的注意事项

用户能够在调用 GT_StrtList()之后的任意时刻调用 GT_EndList()，结束缓冲区命令输入状态；此后，可以调用 GT_AddList()继续缓冲区命令输入状态，输入的命令顺序放在已输入命令的后面；然后又可再次调用 GT_EndList()，结束缓冲区命令输入状态。GT_AddList()和 GT_EndList()构成的组合，可以使用任意多次。当运动轨迹描述完成时，一定要使用 GT_EndList()，通知运动控制器运动轨迹描述完成。

控制器对缓冲区中坐标系运动命令的处理机制

用户可以不断添加向缓冲区中发送坐标系运动命令，直到缓冲区满。

运动控制器内的缓冲区为 4096×16Bit 环形队列缓冲区。缓冲区满时，运动控制器拒绝接收用户输入的多轴协调运动描述命令，并返回缓冲区满的信息。在启动缓冲区中的命令后，随着命令的执行，缓冲区会有新的空间，用户可以继续发送更多的命令。

5.4.2 启动、停止缓冲区中的坐标系运动命令

5.4.2.1 函数列表

表 5-5 启动、停止命令列表

函数	说明
GT_StrtMtn()	启动缓冲区中坐标系命令执行
GT_StpMtn()	平滑停止坐标系运动
GT_EStpMtn()	紧急停止坐标系运动

5.4.2.2 例程

例程 1-21 多段坐标系轨迹连续运动实现

本例实现三段轨迹的连续运动，用户可参考实现大量轨迹段的连续运动。

```
void main()
{
  short rtn;
  GTInitial();
  InputCfg();
  AxisInitial();
  double cnt1[5]={2000,0,0,0,0};
  double cnt2[5]={0,2000,0,0,0};
  double cnt3[5]={0,0,2000,0,0};
  double cnt4[5]={0,0,0,2000,0};
```

```
    rtn=GT_MapAxis(1,cnt1);  error(rtn);
    rtn=GT_MapAxis(2,cnt2);  error(rtn);
    rtn=GT_MapAxis(3,cnt3);  error(rtn);
    rtn=GT_MapAxis(4,cnt4);  error(rtn);
    //以上参考例程 5-2
    rtn=GT_MvXYZA(0,0,0,0,0.01,0.00000001); error (rtn);
    /*设置缓冲区起点定位坐标（0mm,0mm,0mm,0mm），合成速度 3m/min,合成加
     速度 0.9m/min² */
  rtn=GT_LnXY(10,10);     error(rtn);  //运动到坐标（10mm，10mm）
  rtn=GT_ArcXY(0,0,123); error(rtn);
  // 以坐标（0，0）为圆心，以坐标（10，10）为起点，正向 123 度圆弧。
  rtn=GT_LnXYZA(0,0,10,12); error(rtn); //运动到坐标（0,0,10,12）
  rtn=GT_EndList();        error(rtn);   //关闭缓冲区
  rtn=GT_StrtMtn();        error(rtn);   //启动缓冲区的命令

    }
```

5.4.2.3 重点说明

启动缓冲区中命令

用户调用 GT_StrtMtn()命令，缓冲区中的命令将顺序执行。在启动缓冲区中命令之前，用户必须确定运动控制器的缓冲区内有运动描述命令可以被执行，也就是说在此之前主机已调用过 GT_StrtList()和 GT_MvXY（GT_MvXYZ、GT_MvXYZA）命令。

调用 GT_StrtMtn 命令后，不发关闭缓冲区命令 GT_EndList，运动控制器会一边将主机新发的坐标系运动命令加入缓冲区，一边执行缓冲区中的命令。

中断缓冲区中命令的执行

若缓冲区中的命令已经启动，用户希望中断缓冲区中命令的执行，可以调用 GT_StpMtn()和 GT_EStpMtn()命令。GT_StpMtn()命令类似单轴的 GT_SmthStp()命令，可以平滑停止整个坐标系合成运动；而 GT_EStpMtn()命令则类似单轴的 GT_AbptStp()命令，紧急停止整个坐标系合成运动。

GT_StpMtn()和 GT_EStpMtn()命令除了中断缓冲区中命令的执行，同时也关闭了缓冲区，相当于包含 GT_EndList()函数的功能。

在发送 GT_StpMtn()和 GT_EStpMtn()命令后，用户发送的坐标系运动命令将被立即执行。

在“坐标系轨迹运动已完成”情况下，用户可以调用 GT_StrtMtn()命令，再次启动缓冲区中的命令。这时运动控制器以当前缓冲区的合成速度从当前坐标位置以直线插补方式运动到 GT_StpMtn()或 GT_EStpMtn()命令中断处的位置，并减速到零，然后继续执行缓冲区中的命令。

用户不能在执行定位指令时发出中断命令，即运动到缓冲区定位点的过程不能被打断，这时发出中断命令会引起运动出错。

5.4.3 多轴协调运动轨迹速度规划策略

基于坐标系的多轴协调运动沿轨迹方向的合成速度采用梯形曲线加减速策略，函数 GT_SetSynVel()、GT_SetSynAcc()分别设置相应的最大速度和加速度。

对于缓冲区中命令（除 GT_MvXY()、GT_MvXYZ()、GT_MvXYZA()函数外），运动控制器在第一段直线插补（或圆弧插补）命令时合成速度由零加速到最大速度，并在最后一段目标位置处减速到零。而在轨迹中间的各直线插补段和圆弧插补段转接处，尽可能减少沿轨迹切线方向合成速度的变化，以获得稳定的运动特性。

但是此时需要考虑各控制轴的加速度特性，防止加速度过大造成轨迹失真。因此对各控制轴，运动控制器提供函数 GT_SetAccLmt()（该命令属于面向控制轴的命令），用户能够根据各控制轴的实际机电特性，设置各轴的加速度极限值，并采用下面的策略确定转接点合成速度：

策略 1：在插补轨迹段转接点，首先考虑两段轨迹速度的自然过渡，合成速度按照设定的速度和加速度加减速。同时确认各相关运动轴的加速度没有超过各轴设定的加速度极限。如果超过加速度极限，则采用策略 2。此时，为满足用户设定的运动速度要求，采用减速段发生在前一段轨迹，加速段发生在后一段轨迹的策略（*图 5-4(a)(b)(c)*）。这将保证在任一段运动轨迹上，运动的合成速度都不会超过用户设定的最大速度。

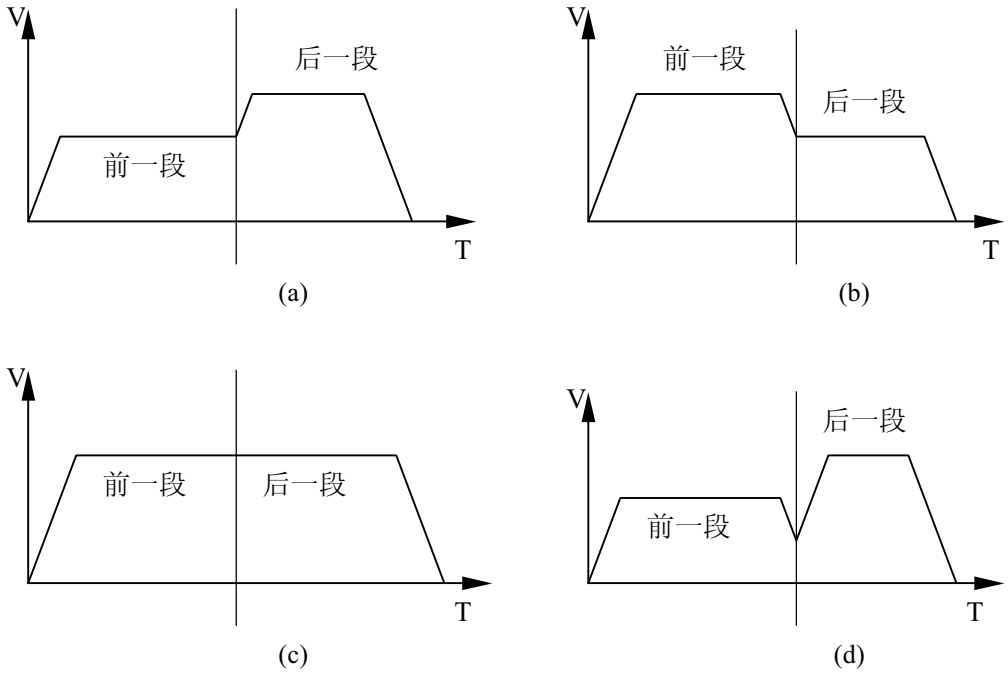

图 5-4 轨迹转接点速度策略

策略 2：在插补轨迹段转接点，如果采用策略 1 时，某一个运动轴的加速度超过了设定的加速度极限，则计算出适当的轨迹转接处的目标合成速度，在满足轨迹转接点处各轴加速度极限的前提下，合成速度下降尽可能少，尽量避免在轨迹段转接处产生较大的速度波动。此时同样采用减速段发生在前一段轨迹，加速段

发生在后一段轨迹的策略。（*图 5-4(d)*）

用户需要注意的是：在设置速度和加速度信息来进行轨迹规划时，应确保加速或者减速在一段轨迹之内完成，即能在一段轨迹之内达到目标速度，并能在一段轨迹之内减速到零。

5.4.4 连续轨迹运动中断点的信息

主机调用 GT_StpMtn()、GT_EStpMtn()命令后，能够通过调用 GT_GetBrkPnt()、GT_GetMtnNm()命令获得中断点的相应信息。GT_GetBrkPnt()获得中断点的坐标。GT_GetMtnNm()返回中断处正在执行的程序段编号。程序段编号根据下面的默认规则定义：

启动缓冲区中命令后，程序段编号随输入缓冲区段数递增，GT_MvXY()（或 GT_MvXYZ()、GT_MvXYZA()）不记段号，即该指令运动时段号为 0，以后段号依次增加；

当缓冲区内的程序段已经执行完毕，而用户又没有执行 GT_EndList()函数关闭缓冲区，那么控制器将认为坐标系运动还没有结束，这时坐标系状态字的 bit0 和 bit1 将不置位，同时程序段号将保持为当前执行完的轨迹段段号。

段号递增的规律是：

1．当前段运动结束时，如果缓冲区中还有命令令则段号指向下一轨迹段，若无则段号保持当前值；

2．缓冲区中命令执行过程中，使用 GT_StpMtn()或 GT_EStpMtn()函数停止坐标系运动时（将引起坐标系状态字的 Bit0 置位），段号保持当前值；

3．调用 GT_StrtList()命令，程序段号清零；

4．段号由 0 开始累加，当计数到最大值 65536 时段号将溢出并从 0 开始重新累计。

5.4.5 坐标系状态寄存器

运动控制器为用户提供一个坐标系运动规划的状态寄存器，用户可以用 GT_GetCrdSts()读取该 16 位寄存器，其中各标志位的定义见*表 5-6*。坐标系状态寄存器中的各个标志位只表征当前的坐标系运动状态，由控制器统一管理，用户程序不能干预该寄存器的状态。

表 5-6 坐标系状态寄存器位定义

位	定义
0	1＝坐标系轨迹运动全部完成或无坐标系轨迹运动
1	1＝缓冲区运动段编写结束（发出 GT_EndList()或 GT_StpMtn()、GT_EStpMtn()时，该位置位）
2	1＝伺服周期过小，引起运动错误（该位置 1 时，请立即修改伺服周期）
3	1＝面向坐标系的命令出错
4	1＝单段轨迹运动完成
5	1＝加速度超限报警（不论哪一个坐标轴的加速度分量超限，都将引起该位置位）
6	1＝异常自动停止允许（当坐标系中相关轴伺服报警或遇限位等异常状态时，若该位置位则将自动停止整个坐标系运动）
7	1＝处于立即命令输入状态；0=处于缓冲区命令输入或执行状态
8	保留
9	1＝坐标系中相关电机轴出现异常，并自动停止坐标系运动
10	1＝坐标系运动发生异常错误（正常情况下该标志不会置起，除非运动控制器运行出现了异常，如用户检测到该位，请立刻中断使用，并将控制器断电后再重新启动）
11～12	保留，默认状态为 0
13	1＝缓冲区空（当启动缓冲区运动后，缓冲区中没有运动描述命令时，该位置位）
15	保留，默认状态为 0

坐标系状态寄存器的 Bit6 置 1 时，映射到坐标系中的各控制轴中，如果某个轴的运动出现异常现象（包括伺服报警或触发限位开关），则运动控制器停止运动规划，自动地停止所有映射到坐标系中的控制轴的运动。这样能够保证整个运动轨迹的完整性。如果 Bit6 清 0，某个轴的运动出现异常现象时，运动控制器只停止该轴的运动，但是继续运动规划，其他映射到坐标系中的控制轴的运动照常进行。Bit6 的置 1 或清 0 由函数 GT_CrdAuStpOn()、GT_CrdAuStpOff()实现。

第六章 Home/Index 高速捕获

控制器为每个轴提供了一个高速位置捕获寄存器，用来保存外部触发信号动作时轴的当前实际位置。SV 允许使用增量式编码器的 C 相（Index）信号或原点（Home）开关信号作为捕获轴位置的触发信号。SD、SE、SG 允许原点（Home）开关信号作为捕获轴位置的触发信号。这两种触发信号的区别在于，前者来自增量式编码器 C 相（或 Z 相）信号。而 Home 信号来自于原点开关触发信号。控制器可以用 GT_CaptIndex()（仅 SV 卡）和 GT_CaptHome()命令来选择使用何种输入作为捕获触发信号。

运动控制器捕获到需要的 Index 或 Home 信号后，控制轴状态寄存器中 Index/Home 标志位置 1，并清除状态寄存器中 Index 捕获或 Home 捕获的设定标志位(15bit 或 14bit)。主机须重新设定位置捕获命令，并清除控制轴状态寄存器 Index/Home 捕捉到标志位才允许下次位置捕获发生。控制器捕获的位置是触发脉冲到来时刻该轴的实际位置，捕获位置精度为+/-1 个脉冲。GT-400-SV 运动控制器捕获位置采用硬件完成，因此控制轴的运动速度不会影响控制器的捕获精度。

控制器的这种高速位置捕获功能主要是用在对系统原点进行定位的场合，而对于重复定位精度要求很高的用户，可以采用 **Home+Index 的原点定位方式**，即在捕获到 Home 信号后再捕获最临近 Home 信号点的 Index 信号点。

例程 6-1 Home+Index 编程（仅对 SV 卡）

本例为 home+index 归零。

```
void home(long pos)
{
   unsigned short status;
   long actl_pos;
   rtn=GT_ClrSts();        error(rtn);  //清状态
   rtn=GT_CaptHome();      error(rtn);  //设置捕获 Home
   rtn=GT_PrflT();         error(rtn);  //设置梯形曲线
   rtn=GT_SetVel(4);       error(rtn);  //设置速度为 4 脉冲/控制周期
   rtn=GT_SetAcc(1);       error(rtn);  //设置加速度为 1
   rtn=GT_SetPos(pos);     error(rtn);  //设置目标位置
   rtn=GT_Update();        error(rtn);  //刷新参数
   rtn=GT_GetSts(&status); error(rtn);  //读取轴状态值
   while(!(status&0x8))                 //等待 Home 捕获
   {
     if(status&0x1)  return;  //如果运动已完成，但 Home 未触发,结束
     rtn=GT_GetSts(&status); error(rtn); //读取轴状态
    }
   rtn=GT_GetCapt(&pos);     error(rtn);  //读取捕获位置
```

```
    rtn=GT_SetPos(pos);         error(rtn);  //设置捕获位置为目标位置
    rtn=GT_Update();            error(rtn);  //刷新参数
    rtn=GT_ClrSts();            error(rtn);  //清状态
    rtn=GT_GetSts(&status);     error(rtn);  //读取状态值
    while(!(status&0x1))         //等待运动完成
    {
      rtn=GT_GetSts(&status); error(rtn);  //读取状态
      }
    rtn=GT_ClrSts();            error(rtn);  //清状态
    rtn=GT_CaptIndex();         error(rtn);  //设置捕获 Index
    pos=pos+8000;                            //捕获位置＋8000 脉冲
    rtn=GT_SetPos(pos);         error(rtn);  //从捕获位置运动 8000 脉冲
    rtn=GT_Update();            error(rtn);
    rtn=GT_GetSts(&status);     error(rtn);  //读取状态
    while(!(status&0x8))          //读取状态，等待捕获
    {
      rtn=GT_GetSts(&status); error(rtn);
    }
    rtn=GT_ClrSts();            error(rtn);   //清状态
    rtn=GT_GetCapt(&pos);       error(rtn);   //读取捕获位置
    rtn=GT_SetPos(pos);         error(rtn);   //设置运动到捕获位置
    rtn=GT_Update();            error(rtn);
    rtn=GT_GetSts(&status);     error(rtn);   //读取状态
    while(!(status&0x1))          //等待运动完成
    {
      rtn=GT_GetSts(&status); error(rtn);
     }
    rtn=GT_ClrSts();            error(rtn);   //清状态
    rtn=GT_ZeroPos();           error(rtn);   //清零
}

void main()
{
  short rtn;
  GTInitial();
  InputCfg();
  AxisInitial();
  rtn=GT_Axis(1); error(rtn);
  Home(200000);
  }
```

例程 6-2　Home 编程

本例为 home 归零。

```
void home(long pos)
{
   unsigned short status;
   long actl_pos;
   rtn=GT_ClrSts();          error(rtn);  //清状态
```

```
    rtn=GT_CaptHome();        error(rtn);  //设置捕获 Home
    rtn=GT_PrflT();           error(rtn);  //设置梯形曲线
    rtn=GT_SetVel(4);         error(rtn);  //设置速度为 4 脉冲/控制周期
    rtn=GT_SetAcc(1);         error(rtn);  //设置加速度为 1
    rtn=GT_SetPos(pos);       error(rtn);  //设置目标位置
    rtn=GT_Update();          error(rtn);  //刷新参数
    rtn=GT_GetSts(&status);   error(rtn);  //读取轴状态值
    while(!(status&0x8))                   //等待 Home 捕获
    {
      if(status&0x1)  return;  //如果运动已完成，但 Home 未触发,结束
      rtn=GT_GetSts(&status); error(rtn); //读取轴状态
     }
    rtn=GT_GetCapt(&pos);     error(rtn);  //读取捕获位置
    rtn=GT_SetPos(pos);       error(rtn);  //设置捕获位置为目标位置
    rtn=GT_Update();          error(rtn);  //刷新参数
    rtn=GT_ClrSts();          error(rtn);  //清状态
    rtn=GT_GetSts(&status);   error(rtn);  //读取状态值
    while(!(status&0x1))       //等待运动完成
    {
      rtn=GT_GetSts(&status); error(rtn);  //读取状态
      }
    rtn=GT_ZeroPos();         error(rtn);   //清零
}

void main()
{
  short rtn;
  GTInitial();
  InputCfg();
  AxisInitial();
  rtn=GT_Axis(1); error(rtn);
  Home(200000);
  }
```

第七章 安全机制及相应处理

7.1 控制轴运动错误监测及状态恢复

运动控制器为控制轴提供了运动出错监测功能。

对于闭环伺服控制而言，在某些情况下，电机的实际位置可能和期望位置差距很大。这时通常意味着存在一些危险情况，例如电机故障、光电编码器 A、B 信号接反或断线、机械摩擦太大或机械故障造成电机堵转等。为了能检测这种情况，增强系统的安全性和延长设备使用寿命，GT-400-SV 控制器中设置了可编程修改的控制轴位置误差极限值。

用GT_SetPosErr()命令设置位置误差极限值，用GT_GetPosErr()命令读取该值。

运动控制器在每一个伺服周期内都比较位置误差极限值与实际位置误差值以检测是否出现运动错误。如果控制轴的实际位置误差值超过了控制器设置的位置误差极限值，控制器就认为该轴出现了运动错误。

当控制轴运动错误发生时，控制器将产生下列几个事件：

- 状态寄存器运动出错标志位置 1；
- 如果模式寄存器中运动出错停止标志置 1，控制轴将停止运动，设定运动完成标志位为 1，并自动关闭电机伺服使能；否则只将控制轴状态寄存器中的运动出错标志位置 1 。主机可以用 GT_AuStpOn()和 GT_AuStpOff()命令来设定和清除该运动出错停止标志位。控制器默认标志为 0，即运动出错时电机轴不自动停。

从导致控制轴停止的运动出错状态恢复到正常运动状态，要执行以下操作：

- 确定引起运动出错的原因并加以改正；
- 清除状态寄存器中运动出错标识位；
- 使用 GT_SynchPos()或 GT_ZeroPos()，使控制轴实际位置与目标位置同步或实际位置与目标位置清零；
- 用 GT_AxisOn()命令重新使能该控制电机工作。
- 完成上述步骤后，该控制轴进入正常的伺服状态并等待下一次运动。

7.2 控制轴驱动器报警处理

某些电机驱动器为了电机和驱动器的安全和延长使用寿命，内部设定了系统故障（如驱动器输入超限，输入信号变化过大）检测和保护。当驱动器检测到工作异常或故障时，触发故障保护并输出报警信号。

控制器提供了专用的驱动器报警输入，当控制器检测到电机驱动器报警时，

控制器将控制轴状态寄存器中驱动器报警标志位置 1，运动完成标志位置 1，控制器关断驱动器使能信号并禁止该控制轴（效果等同于 GT_AxisOff()命令）。

从驱动器报警状态恢复到正常运动状态，要执行以下操作：

- 确定引起报警的原因并加以改正；
- 使用 GT_DrvRst()命令使驱动器复位；
- 用 GT_ClrSts()或 GT_RstSts()命令清除状态寄存器中驱动器报警标志位；
- 使用 GT_SynchPos()或 GT_ZeroPos()命令使控制轴实际位置与目标位置同步或实际位置与目标位置清零；
- 用 GT_AxisOn()命令重新使能控制电机工作。

完成上述步骤后，控制轴进入正常的伺服状态并等待下一次运动。

7.3 限位状态处理

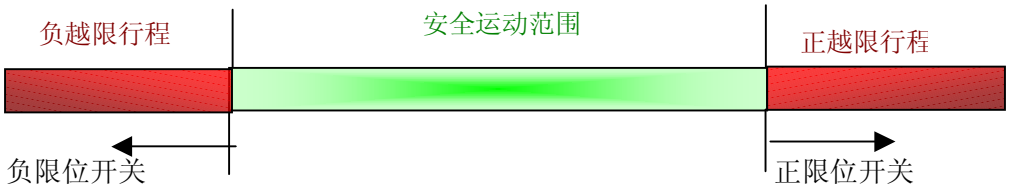

图 7-1 控制轴运动行程范围定义

运动控制器可使用限位开关自动标志控制轴的运动范围。装有限位开关的控制轴的“安全”运动范围如*图 7-1* 所示。

接收到限位开关后，提供以下两个操作：

- 控制器能自动设置控制轴状态寄存器中的相应限位开关触发标志位，通知主机采取适当措施来处理越限情况。
- 同时控制器将自动停止控制轴的运动，使之不再进一步朝越限区域运动。

一旦控制轴触发限位开关，该轴将只允许朝相反的方向运动。例如：假设控制轴的正限位开关被触发，控制器只允许该轴朝负方向运动，以便返回安全运动范围。

当控制轴返回安全运动范围后，主机必须使用 GT_ClrSts()或 GT_RstSts()命令清除其状态字中的相应状态标志位，使控制轴运动从越限状态恢复到正常状态 。

如果控制器的某个轴工作在电子齿轮模式，当从动轴碰到限位开关，主动轴如果在 GT_AxisOn()状态下，其运动方向同样受其从动轴限位开关的约束。例如：当从动轴的负向限位开关被触发时，从动轴只允许朝正方向运动，同其对应的主动轴也只允许朝某一方向运动。如果此时的电子齿轮比 ratio 为负，主动轴只允许朝负方向运动。这种互锁关系将一直保持，直到从动轴返回安全区为止。

第八章 中断

运动控制器可向主机发出中断请求，使主机能及时对各控制轴运动过程中出现的事件作出处理。这种中断方式通常比主机查询控制器的各种状态更为方便和有效。

运动控制器提供了两种中断方式，一种是**事件中断**，主要针对控制轴运动过程中出现的事件作出及时处理；另一种是**定时中断**，控制器以一定的周期向主机发出定时中断。用运动控制器构成系统时，该中断可作为系统定时器。

由于两个中断共享主机一根中断请求线，因此主机在系统中只允许通过GT_TmrIntr()和 GT_EvntIntr()命令来选择一种中断方式。GT_TmrIntr()命令设定控制器中断为定时中断，定时中断的周期由控制器的控制周期和 GT_SetIntrTm()命令的设定值共同确定。例如：控制器的控制周期为 200 微秒，主机用 GT_SetIntrTm()命令设定的值为 10，则定时中断的周期=10*200 微秒，即 2 毫秒。GT_EvntIntr()命令将关闭控制器定时中断，转向事件中断。控制器默认中断为事件中断。

表 4-14 控制轴状态寄存器中的 bit0—bit6 所对应的控制轴事件，都能够触发运动控制器的事件中断请求。对于每个控制轴，主机可以通过 GT_SetIntrMsk()命令设定控制器的控制轴中断屏蔽寄存器，确定各种事件能否向主机申请中断。

8.1 DOS 系统下中断处理

如果某个控制轴出现上述中断条件，激活中断请求信号，此时主机可根据实际情况响应中断，进行适当处理。主机完成中断处理后，应发出 GT_RstIntr()指令清除当前电机的中断请求条件，使下次中断可以发生，该命令带有一个“清除相应屏蔽标志”的参数。

主机一次只能响应一个控制轴的中断。例如，用户正在设置当前轴#1 的参数，而控制轴#3 向主机发出中断请求信号。此时，如果需要处理该中断，主机就必须把#3 设置为当前轴（使用 GT_AxisI()命令）。如果同时有多个控制轴提出中断申请，最小号的控制轴具有最高的中断优先级。

下面列出一个典型的中断条件处理序列以说明主机是如何响应中断的。假设控制轴#3(当前轴为#1)因“急停”引起瞬时运动错误并处于超限位状态，同时假设该控制轴中断屏蔽寄存器允许主机响应该电机发出的上述中断请求信号。此时，主机响应中断的操作过程如*表 8-1* 所示。在上述过程的最后，所有的状态位都被清除，主机中断请求信号恢复，没有中断等待处理。

表 8-1 主机响应中断处理

中断事件	主机响应
运动出错且超限位产生中断。	主机发出 GT_AxisI()命令。
控制器返回发出中断请求信号的控制轴状态，并把该轴设置为当前轴。	主机查询到运动出错和超限位标志位为 1， 先处理控制器运动出错。主机发出 GT_RstIntr(0xEF)指令清除运动出错标志位。
控制器清除运动出错标志位，并恢复主机中断源为低电平。	-
因为轴超限位中断仍然有效，控制器立即向主机发出超限位中断请求信号。	主机发出 GT_AxisI()指令。
控制器返回发出中断请求的控制轴状态，并把它设置为当前轴。	主机查询到超限位标志位为 1，执行相应的处理程序。主机发出 GT_RstIntr(00DF) 指令清除越位标识位。
控制器清除越位标志位，并恢复主机的中断源为低电平。	-

GT_RstIntr()指令和 GT_AxisI()指令只在中断存在时有效。如果没有中断，必须采用查询的命令如：GT_GetSts()命令查询轴的状态。

对于一个请求中断的控制轴而言，主机只能响应一个事件中断。当有多个事件同时申请中断时，并不要求像上面所举的例子中那样分别处理这些中断请求。控制器可以只向主机发出一次中断请求，主机可在中断响应处理过程中用 GT_GetIntr()命令读取中断轴的状态寄存器进行判断和作相应处理，然后用 GT_RstIntr()命令一次清除所有中断事件（GT_RstIntr()和 GT_AxisI()函数只能在中断服务程序中使用，而如果没有中断发生时调用 GT_GetIntr()函数，所返回的状态为当前轴状态）。

对于多轴同时申请中断的情形，主机也可以采用类似处理。在响应最小号的控制轴的中断时，用 GT_GetSts()命令查看其它控制轴的状态并作出相应处理，然后用 GT_RstSts()或 GT_ClrSts()命令清除相应的状态标识。

当某个轴处于轴伺服使能禁止时，除运动出错和驱动报警状态外，其它状态均不能引起事件中断，请用户使用时注意。

例程 8-1 事件中断例程（ISA 总线卡）：

```
void interrupt handler(…)
{
   disable();       //关闭中断
   GT_AxisI();   //将中断轴设置为当前轴
   GT_GetIntr(&event);   //读取中断轴状态寄存器
   If(event&0x8)            //判断是否为 HOME 触发中断
    (
      GT_GetCapt(&HomePos);     //读取 Home 捕获位置
      GT_SetPos(HomePos);         //设置目标位置为 Home 捕获位置
      GT_Update();                  //刷新参数
     )
   GT_RstIntr(0);                  //清除运动控制卡中断
```

```
    outportb(0x20,0x20);          //送 EOI 到主 8259
    outportb(0xa0,0x20);          //送 EOI 到从 8259
    enable();                     //打开中断
}
```

例程 8-2 时间中断例程(PCI 总线卡)：

```
#include <stdio.h>
#include <dos.h>
#include <conio.h>
#include "userlib.h"

GT_ISR oldisr;
int count;
void interrupt OnInterrupt(...)
{
    GT_ClearInt(0);          //清中断
    count++;                 //计数加 1，用户可设置自己的代码
    outportb(0x20,0x20);     //送 EOI 到主 8259
    outportb(0xa0,0x20);     //送 EOI 到从 8259
}
main()
{
    short nret=GT_Open();    //打开运动控制器
    if(nret)                 //检查返回值
    {
        printf("Open fail\n");
        return 0;
    }
    oldisr=GT_HookIsr(OnInterrupt);   //挂接中断
    nret=GT_SetIntrTm(500);           //设置时间中断周期 500*200 微妙
    if(nret)
    {
        printf("Set Interrupt Time fail\n");
        return 0;
    }
    nret=GT_TmrIntr();                //设置当前为时间中断
    while(!kbhit())
    {
        printf("Count:%d\n",count);   //打印计数值
        delay(500);
    }
    getch();
    nret=GT_EvntIntr();               //设置当前为事件中断
    GT_UnhookIsr(oldisr);             //取消挂接的中断，恢复原中断
    return 0;
}
```

8.2 WINDOWS98/2000/NT 系统下中断处理

在 DOS 环境开发程序时，用户可以直接修改中断向量，挂接自己的中断服务程序（Interrupt Service Routine 简为 ISR），从而实现对运动控制器产生的中断请求进行响应。但是在 WINDOWS 环境，操作系统将系统内核和应用程序进行隔离，

不再允许用户层程序修改设备的 ISR。基于此，GT-400-PCI 驱动程序独辟蹊径，提供两种联系用户程序和设备 ISR 的中断处理机制，我们称为***事件同步机制***和***中断预处理机制***。事件同步机制适用于事件中断和时间中断的处理，中断预处理机制只适用于事件中断的处理。这两种机制可以混合使用，也可以单独使用。

8.2.1 事件同步机制

其原理是，能够让设备 ISR 和上层用户程序共享一事件，实现用户程序和设备 ISR 同步，从而让用户程序感知硬件设备的中断请求。

具体实现方法是这样的，在用户程序中创建一同步事件，利用控制器驱动程序提供的 API 函数 GT_SetIntSyncEvent(HEVENT hIntEvent)向设备 ISR 设置同步事件。此后，用户程序和设备 ISR 就是两个共享同步事件的普通进程，一般情况用户程序为了不阻塞自己，会启动一个新的线程，在该新线程中调用 WaitForSingleObject()等待事件有效。一旦设备产生中断，设备 ISR 激活中断同步事件。此时用户线程从 WaitForSingleObject()处被激活,开始执行线程的后续部分。

要注意的是，用户程序在调用 CloseHandle()关闭事件、设备之前，必须调用 GT_SetIntSyncEvent(NULL), 通知设备 ISR 清除中断同步事件。

在这个流程中，用户必须有一个清醒的认识：用户线程只是被通知刚才中断发生了而并不是在中断处理过程中，实际上中断已经被内核清除了。而且受系统效率和中断频率的影响，并不保证每次中断都能激活用户线程，当中断频率很高，而操作系统线程调度慢时，有可能出现多次中断堆叠。一般要求中断频率<=10KHz。

详细编程可参考以下代码（所有 Windows 例程在 VC++环境下编写）：

例程 8-3 中断事件同步机制

```
//全局变量
HANDLE hSyncEvent;      //同步事件句柄
bool stopflag;          //线程结束标志

//主函数中代码
HANDLE hSubThread;
DWORD idSubThread;
//……
   //建立同步事件
   hSyncEvent=CreateEvent(NULL,true,false,NULL); //WIN32 API 函数
   if(hSyncEvent==INVALID_HANDLE_VALUE)
   {
        //..此处做有效性检查
   }
   //向设备 ISR 设置同步事件，从而实现事件共享
    nret=GT_SetIntSyncEvent(hSyncEvent);
    //GT400.DLL 提供的 API 函数
   if(nret)
```

```
    {
        //检查函数执行情况
    }
    stopflag=false;
    //为了不阻塞自己，创建并启动一个新的线程
    hSubThread=CreateThread(NULL,
                            0,
                            intproc, //  新线程函数
                            NULL,//新线程函数所用参数
                            0,
                            &idSubThread);
    //至此，主函数的任务完成

//下面是等待同步事件的线程函数
DWORD WINAPI intproc(LPVOID param)
{
    ResetEvent(hSyncEvent); //确保事件处于 nonsignaled 状态
    while(1)
    {
        //waiting for interrupt happen
        WaitForSingleObject(hSyncEvent,INFINITE);//等待事件
        if(stopflag)
            break;
        //add your code for handling intrerrupt event here
        //这里可添加你需要处理的代码
        //.....

        //reset event state
        ResetEvent(hSyncEvent);//复位同步事件
    }
    GT_SetIntSyncEvent(NULL);//通知设备 ISR 释放事件
    //close Synchronize Event Handle
    CloseHandle(hSyncEvent); //关闭同步事件句柄
    ExitThread(0); //退出线程
    return 0;
}
```

8.2.2 中断预处理机制

其原理是，在设备产生中断前，预先为设备设定一些命令（此处的命令指运动控制卡用的 GT 命令），当设备产生一指定中断时，按预设定的结构执行和该中断对应的命令。

其实现和使用相对简单，一般先分配一定内存空间用于存放要为设备设置的命令，按照指定的结构（在 gt400data.h 中定义）填充该内存，调用控制卡驱动程序提供的 API 函数 GT_SetBgCommandSet（）为设备设定命令，当设备产生中断

时，将自动搜索该数据结构，找到与中断对应的命令并执行，执行结果也保存在该结构中。用户层程序可以通过 GT_GetBgCommandResult（）获取命令执行结果。可用于后台命令集的函数参见表5－1，命令格式为在原函数前加Intr如：GT_PrflT()作为后台命令时命令字为 Intr_GT_PrflT。具体编程参见以下代码：

例程 8-4 中断中断预处理机制

```
//定义可能用到的最大内存空间的大小，
#define MAX_SIZE    500
//缓冲需要大小=4+4*(要设置命令的中断数量)+16*（所有命令的总数）
//如下面代码中，至小需要空间：4+4*2（两种中断）+16*（2+1）=60 字节。
   PBGCOMMANDSET pBgCmdSet;//定义一指针
   pBgCmdSet=(PBGCOMMANDSET)malloc(MAX_SIZE);//分配内存
   if(pBgCmdSet==NULL)
   {
       //做有效性检查
   }
   PBACKGROUND_COMMAND pBackCmd;
   PGENERAL_COMMAND pCmd;
   //指定要为多少种中断设置后台命令
   pBgCmdSet->Count=2;                  //要为两种中断设置后台命令
   //分别为每个中断设置命令数组，对同一个中断允许设多条命令
   pBackCmd=pBgCmdSet->BackgroundCommand;

   pBackCmd->InterruptMask=0x01;       //指定中断原因
   pBackCmd->CommandCount=2;           //产生本中断时要执行的命令数量
   //分别填充每一条命令
   //第一条命令
   pCmd=pBackCmd->GenCommand;
   pCmd->usCommand= Intr_GT_SetPos;       //设定命令类型字
   pCmd->OutputLength=2;
   //该命令执行时，需要向 DSP 输出多少个字长度的数据
   pCmd->InputLength=0;
   //该命令执行时，需要从 DSP 输入多少个字长度的数据
   pCmd->in.lData=20000;       //输入数据
   pCmd->out.lData=0;          //输出数据
   //填充下一条命令
   pCmd=PGENERAL_COMMAND((char*)pCmd+sizeof(GENERAL_COMMA
ND));  //指向下一个内存单元
   pCmd->usCommand= Intr_GT_Update;       //设定命令类型字
   pCmd->OutputLength=0;
   //该命令执行时，需要向 DSP 输出多少个字长度的数据
   pCmd->InputLength=0;
   //该命令执行时，需要从 DSP 输入多少个字长度的数据
   pCmd->in.lData=0;           //输入数据
   pCmd->out.lData=0;          //输出数据
```

```
    //为另一中断设置命令数组
    pBackCmd=
    PGENERAL_COMMAND( (char*)pCmd+sizeof(GENERAL_COMMAND));
    //指向下一内存单元

    pBackCmd->InterruptMask=0x02;//指定中断原因
    pBackCmd->CommandCount=1;//产生本中断时要执行的命令数量
    //填充第一条命令
    pCmd=pBackCmd->GenCommand;
    pCmd->usCommand= Intr_GT_SetKp;  //设定命令类型字
    pCmd->OutputLength=1;
    //该命令执行时，需要向 DSP 输出多少个字长度的数据
    pCmd->InputLength=0;
    //该命令执行时，需要从 DSP 输入多少个字长度的数据
    pCmd->in.lData=0;              //输入数据
    pCmd->out.lData=20;            //输出数据
    //将命令缓冲传给设备 ISR
    short nret=GT_SetBgCommandSet(pBgCmdSet,MAX_SIZE);
    if(nret)
{
    // 检查函数执行情况
}
//之后释放内存
free(pBgCmdSet);
```

第九章 通用数字量 I/O

运动控制器为用户提供了一个通用数字量输入/输出口。主机可以通过命令的方式对该输入/输出口进行操作。

其中，通用输入的 0 号断口（EXI0）可以作为探针输入信号，并通过相关命令设置捕获探针输入信号，当有探针输入信号时引起运动控制器捕获所有控制轴以及辅助编码器的实际位置。

16 位输出口：主机可以通过命令 GT_ExOpt(Data)设定该输出口的状态。Data 与控制器 CN2 接口的通用数字量输出端口 EXO0-EXO15 的对应关系为：

位一定义	位一定义	位一定义	位一定义
Bit0----EXO0	Bit1----EXO1	Bit2----EXO2	Bit3----EXO3
Bit4----EXO4	Bit5----EXO5	Bit6----EXO6	Bit7----EXO7
Bit8----EXO8	Bit9----EXO9	Bit10----EXO10	Bit11----EXO11
Bit12----EXO12	Bit13----EXO13	Bit14----EXO14	Bit15----EXO15

16 位输入口：主机可以通过命令 GT_ExInpt(&Data)返回该输入端口的状态。返回数据 Data 与 EXI0-EXI15 位定义对应关系为：

位一定义	位一定义	位一定义	位一定义
Bit0----EXI0	Bit1----EXI1	Bit2----EXI2	Bit3----EXI3
Bit4----EXI4	Bit5----EXI5	Bit6----EXI6	Bit7----EXI7
Bit8----EXI8	Bit9----EXI9	Bit10----EXI10	Bit11----EXI11
Bit12----EXI12	Bit13----EXI13	Bit14----EXI14	Bit15----EXI15

例程 9-1 若通用数字量输入 EXI5 为高电平，则设置通用数字量输出的 EXO0 为高电平。

```
void main()
{
    short rtn, ex_inp;
    rtn=GT_ExInpt(&ex_inp);        error(rtn);
    if(ex_inp&0x20)
    {
        rtn=GT_ExOpt(0x1);         error(rtn);
        }
}
```

第二部分

库函数说明

第十章 函数列表

函数	说明
GT_AbptStp()	立即停止当前轴的运动
GT_AddList()	追加命令缓冲区
GT_ArcXY()	XY 平面圆弧插补（以圆心位置和角度为输入参数）
GT_ArcXYP()	XY 平面圆弧插补（以终点位置和半径为输入参数）
GT_ArcYZ()	YZ 平面圆弧插补（以圆心位置和角度为输入参数）
GT_ArcYZP()	YZ 平面圆弧插补（以终点位置和半径为输入参数）
GT_ArcZX()	ZX 平面圆弧插补（以圆心位置和角度为输入参数）
GT_ArcZXP()	ZX 平面圆弧插补（以终点位置和半径为输入参数）
GT_AuStpOff()	设置运动出错自动停止无效
GT_AuStpOn()	设置运动出错自动停止有效
GT_AuUpdtOff()	关闭当前轴控制参数和命令自动更新
GT_AuUpdtOn()	设置当前轴控制参数和命令自动更新
GT_Axis()	设置某控制轴为当前轴
GT_AxisI()	设置发出中断请求的控制轴为当前轴（在 Windows 环境下禁用）
GT_AxisOff()	关闭当前轴
GT_AxisOn()	使当前轴处于工作状态
GT_BrkOff()	清除当前轴断点并关闭断点模式
GT_CaptHome()	设置允许当前轴捕获 HOME 信号
GT_CaptIndex()	设置允许当前轴捕获 INDEX 信号
GT_CaptProb()	设置探针捕获功能
GT_Close()	关闭运动控制器设备
GT_CloseLp()	设置当前轴为闭环伺服控制
GT_ClearInt	清除控制卡产生的中断
GT_ClrSts()	清除当前轴状态
GT_CrdAuStpOff()	关闭坐标系运动异常自动停止方式
GT_CrdAuStpOn()	打开坐标系运动异常自动停止方式
GT_CtrlMode()	设置当前轴控制输出为模拟量输出还是脉冲输出
GT_DrvRst()	使当前轴的伺服驱动器复位
GT_EncPos()	获取辅助编码器位置
GT_EncSns()	设置编码器的计数方向
GT_EncVel()	获取辅助编码器速度

函数	说明
GT_EndList()	关闭命令缓冲区
GT_EStpMtn()	紧急停止坐标系运动
GT_EvntIntr()	设置控制器对主机的中断为轴事件中断
GT_ExInpt()	读取通用输入端口的状态
GT_ExOpt()	设置通用输出端口的状态
GT_ExtBrk()	设置当前轴原点信号触发断点模式
GT_GetAcc()	读取当前轴的命令加速度
GT_GetAccLmt()	读取当前轴加速度极限
GT_GetAdc()	读取 AD 转换结果
GT_GetAddr()	读取运动控制器通讯基地址（在 Windows 环境下禁用）
GT_GetAtlErr()	读取当前轴的实际位置误差
GT_GetAtlPos()	读取当前轴的实际位置
GT_GetBgCommandResult()	获取控制器后台命令集执行结果
GT_GetBrkCn()	读取当前轴断点位置比较值
GT_GetBrkPnt()	读取缓冲区命令执行中断时的断点位置
GT_GetCapt()	读取当前轴 INDEX 或 HOME 捕获位置值
GT_GetCmdSts()	读取上一条控制命令的执行状态
GT_GetCrdSts()	读取坐标系状态字
GT_GetCurrentCardNo()	获取当前控制卡的卡号
GT_GetILmt()	读取当前轴的伺服滤波器积分误差饱和值
GT_GetIntgr()	读取当前轴的积分位置误差
GT_GetIntr()	读取当前轴中断事件（在 Windows 环境下禁用）
GT_GetIntrMsk()	读取当前轴中断屏蔽字
GT_GetIntrTm()	读取定时中断的时间常数
GT_GetJerk()	读取当前轴的命令加加速度
GT_GetKaff()	读取当前轴的伺服滤波器加速度前馈增益
GT_GetKd()	读取当前轴的伺服滤波器微分增益
GT_GetKi()	读取当前轴的伺服滤波器积分增益
GT_GetKp()	读取当前轴的伺服滤波器比例增益
GT_GetKvff()	读取当前轴的伺服滤波器速度前馈增益
GT_GetLmtSwt()	读取限位开关的状态
GT_GetMAcc()	读取当前轴的最大命令加速度
GT_GetMode()	读取当前轴模式字
GT_GetMtnNm()	读取缓冲区当前执行段号
GT_GetMtrBias()	读取当前轴的伺服滤波器输出零点偏移值
GT_GetMtrCmd()	开环方式下读取当前轴电机控制值
GT_GetMtrLmt()	读取当前轴的伺服滤波器输出饱和值
GT_GetPos()	读取当前轴的命令位置

函数	说明
GT_GetPosErr()	读取当前轴的伺服滤波器位置误差极限
GT_GetPrfPnt()	读取坐标系各轴坐标值
GT_GetRatio()	读取当前轴的电子齿轮比
GT_GetSmplTm()	读取控制器伺服采样周期
GT_GetSts()	读取当前轴状态字
GT_GetVel()	读取当前轴的命令速度
GT_HardRst()	硬件复位运动控制器
GT_HookIsr()	为控制器挂接中断服务程序
GT_LmtSns()	设置限位开关的有效电平
GT_LmtsOff()	关闭当前轴的限位开关
GT_LmtsOn()	使当前轴的限位开关有效
GT_LnXY()	两维直线插补
GT_LnXYZ()	三维直线插补
GT_LnXYZA()	四维直线插补
GT_MapAxis()	坐标系映射
GT_MltiUpdt()	多控制轴参数更新
GT_MtnBrk()	设置当前轴运动到位触发断点模式
GT_MvXY()	定位缓冲区命令起点（二维）
GT_MvXYZ()	定位缓冲区命令起点（三维）
GT_MvXYZA()	定位缓冲区命令起点（四维）
GT_NegBrk()	设置当前轴负向位置断点模式
GT_Open()	打开运动控制器设备
GT_OpenLp()	设置当前轴为开环控制
GT_PosBrk()	设置当前轴正向位置断点模式
GT_PrflG()	设置当前轴的运动模式为电子齿轮模式
GT_PrflS()	设置当前轴的运动模式为 S-曲线模式
GT_PrflT()	设置当前轴的运动模式为梯形曲线模式
GT_PrflV()	设置当前轴的运动模式为速度控制模式
GT_Reset()	复位运动控制器
GT_RstIntr()	清除当前轴中断事件（在 Windows 环境下禁用）
GT_RstSts()	清除当前轴状态
GT_SetAcc()	设置当前轴的加速度（梯形曲线模式、速度控制模式）
GT_SetAccLmt()	设置当前轴的加速度极限（坐标运动模式）
GT_SetAddr()	设置运动控制器通讯基地址（在 Windows 环境下禁用）
GT_SetAtlPos()	设置当前轴的实际位置
GT_SetBgCommandSet()	设置中断时后台执行命令集
GT_SetBrkCn()	设置当前轴断点位置比较值（与正向或负向位置断点模式一起使用）

函数	说明
GT_SetILmt()	设置当前轴的伺服滤波器误差积分饱和值
GT_SetIntSyncEvent()	为运动控制器中断设置同步事件
GT_SetIntrMsk()	设置当前轴中断屏蔽字
GT_SetIntrTm()	设置控制器定时中断的时间常数
GT_SetJerk()	设置当前轴的加加速度（S-曲线模式）
GT_SetKaff()	设置当前轴的伺服滤波器加速度前馈增益
GT_SetKd()	设置当前轴的伺服滤波器微分增益
GT_SetKi()	设置当前轴的伺服滤波器积分增益
GT_SetKp()	设置当前轴的伺服滤波器比例增益
GT_SetKvff()	设置当前轴的伺服滤波器速度前馈增益
GT_SetMAcc()	设置当前轴的最大加速度（S-曲线模式）
GT_SetMtrBias()	设置当前轴的伺服滤波器输出零点偏移值
GT_SetMtrCmd()	开环方式下设置当前轴电机控制值
GT_SetMtrLmt()	设置当前轴的伺服滤波器输出饱和值
GT_SetPos()	设置当前轴的目标位置（S-曲线模式、梯形曲线模式）
GT_SetPosErr()	设置当前轴的伺服滤波器位置误差极限
GT_SetRatio()	设置当前轴的电子齿轮比（电子齿轮模式）
GT_SetSmplTm()	设置控制器伺服采样周期
GT_SetSynAcc()	设置多轴协调运动轨迹切线加速度
GT_SetSynVel()	设置多轴协调运动轨迹切线速度
GT_SetVel()	设置当前轴的目标速度（S-曲线模式、梯形曲线模式、速度控制模式）
GT_SmthStp()	平滑停止当前轴的运动
GT_StepDir()	设置当前轴在脉冲输出模式下的输出方式为“脉冲＋方向”方式
GT_StepPulse()	设置当前轴在脉冲输出模式下的输出方式为“正负脉冲”方式
GT_StpMtn()	平滑停止坐标系运动
GT_StrtList()	打开命令缓冲区
GT_StrtMtn()	开始缓冲区命令执行
GT_SynchPos()	设置当前轴的目标位置等于实际位置
GT_SwitchtoCardNo()	切换当前卡
GT_TmrIntr()	设置控制器对主机的中断为定时中断
GT_UnhookIsr()	释放由 GT_HookIsr 函数为控制卡持接的 ISR，并恢复原 ISR。
GT_Update()	当前轴参数更新
GT_ZeroPos()	当前轴实际位置和目标位置清零

第十一章 函数说明

下面将各接口函数按照字母先后的顺序，逐一介绍。
其中的函数调用程序均在 BC3.1 语言环境下编写。
所有函数返回值参见第二章 命令（库函数）返回值。

GT_AbptStp

函数原型： short GT_AbptStp(void);

函数说明： 该函数用于立即停止当前轴的运动，将目标速度参数和实际速度设置为零值。该函数执行后立即生效，可用于急停处理。GT_AbptStp()命令在面向控制轴的四种运动控制模式下都有效。

系　　统： DOS，WINDOWS

适用板卡： 所有 GT 系列卡

相关函数： GT_SmthStp

函数调用： 下面的例子中当检查到外部 IO 的 EXI15 口为高电平时，急停第一轴的运动。

```
void main()
{
    short rtn,ex_data;
    rtn=GT_ExInpt(&ex_data);    error(rtn); //读取输入端口的状态
    rtn=GT_Axis(1);             error(rtn); //设置第 1 轴为当前轴
     if(ex_data&0x8000)                     //判断 EXI15 是否为高电平
    {
        rtn=GT_AbptStp();      error(rtn); //急停运动
        }
}
```

GT_AddList

函数原型： short GT_AddList(void);

函数说明： 在函数 GT_EndList()关闭缓冲区之后，如果用户需要继续增加坐标系轨迹段，可以调用本函数再次打开缓冲区。当缓冲区处于打开状态，该函数无效。

系　　统： DOS，WINDOWS

适用板卡： 所有 GT 系列卡

相关函数： GT_StrtList，GT_EndList

函数调用： 下面的例子中， 在用 GT_EndList()关闭缓冲区后，再次打开缓冲区。

```
void main()
{
    short rtn;
```

```
    rtn=GT_StrtList();                    error(rtn);
    rtn=GT_MvXYZA(0,0,0,0,5,0.1);   error(rtn);
        ……
    rtn=GT_EndList();                     error(rtn);
    rtn=GT_AddList();                     error(rtn);
    rtn=GT_LnXY(20000,30000);         error(rtn);
        ……
}
```

GT_ArcXY

函数原型：short GT_ArcXY(double X_center,double Y_center,double Angle);

函数说明：该函数实现 XOY 平面内的两轴圆弧插补。圆弧插补运动的起点坐标是前一段轨迹描述的终点坐标；如是第一条运动轨迹段，则起点是当前位置坐标。

函数参数：X_center、Y_center 是圆弧圆心坐标，坐标单位由用户通过坐标系映射函数自行定义；angle 是旋转角度，其单位是度，正负代表旋转方向，旋转角度的取值范围为-360～360 度。旋转方向参见 *5.3 坐标系轨迹运动设置*的相关说明。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

相关函数：GT_ArcYZ，GT_ArcZX

函数调用：举例——立即执行 XOY 平面圆弧插补命令。

```
void main()
{
    short rtn;
    rtn=GT_SetSynVel(5);                  error(rtn);
    rtn=GT_SetSynAcc(1);                  error(rtn);
    rtn=GT_ArcXY(40000,30000,180);   error(rtn);
  }
```

GT_ArcXYP

函数原型：short GT_ArcXYP(double X_end, double Y_end, double R, short Dir);

函数说明：该函数实现 XOY 平面内的两轴圆弧插补。圆弧插补运动的起点坐标是前一段轨迹描述的终点坐标；如是第一条运动轨迹段，则起点是当前位置坐标。

函数参数：X_end、Y_end 是圆弧终点坐标；R 是圆弧半径并带符号，其符号表示此段圆弧是优弧还是劣弧（正：劣弧，负：优弧），坐标和半径的单位由用户通过坐标系映射函数自行定义；Dir 是圆弧旋转方向，取值为 1 表示正向旋转，-1 表示负向旋转。旋转方向参见 *5.3 坐标系轨迹运动设置*的说明。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

相关函数：GT_ArcYZP，GT_ArcZXP

函数调用：举例——发送 XOY 平面圆弧插补命令到缓冲区。

```
void main()
{
    short rtn;
    rtn=GT_StrtList();     error(rtn);
    rtn=GT_MvXY(0,0, 5,0.1);    error(rtn);
```

```
    rtn=GT_ArcXYP(40000,0,20000,-1);    error(rtn);
    rtn=GT_EndList();    error(rtn);
}
```

GT_ArcYZ

函数原型：short GT_ArcYZ(double Y_center,double Z_center,double Angle);

函数说明：该函数实现 YOZ 平面内的两轴圆弧插补。圆弧插补运动的起点坐标是前一段轨迹描述的终点坐标；如是第一条运动轨迹段，则起点是当前位置坐标。

函数参数：Y_center、Z_center 是圆弧圆心坐标，坐标单位由用户通过坐标系映射函数自行定义；angle 是旋转角度，其单位是度，正负代表旋转方向，旋转角度的取值范围为-360～360 度。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

相关函数：GT_ArcXY，GT_ArcZX

函数调用：举例——立即执行 YOZ 平面圆弧插补命令。

```
void main()
{
    short rtn;
    rtn=GT_SetSynVel(5);    error(rtn);
    rtn=GT_SetSynAcc(1);    error(rtn);
    rtn=GT_ArcYZ(40000,30000,180);    error(rtn);
 }
```

GT_ArcYZP

函数原型：short GT_ArcYZP(double Y_end, double Z_end, double R, short Dir);

函数说明：该函数实现 YOZ 平面内的两轴圆弧插补。圆弧插补运动的起点坐标是前一段轨迹描述的终点坐标；如是第一条运动轨迹段，则起点是当前位置坐标。

函数参数：Y_end、Z_end 是圆弧终点坐标；R 是圆弧半径并带符号，其符号表示此段圆弧是优弧还是劣弧（正：劣弧，负：优弧），坐标和半径的单位由用户通过坐标系映射函数自行定义；Dir 是圆弧旋转方向，取值为 1 表示正向旋转，-1 表示负向旋转。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

相关函数：GT_ArcXYP，GT_ArcZXP

函数调用：举例——立即执行 YOZ 平面圆弧插补命令。

```
void main()
{
    short rtn;
    rtn=GT_StrtList();    error(rtn);
    rtn=GT_MvXYZ(0,0,0,5,0.1);    error(rtn);
    rtn=GT_ArcYZP(40000,0,20000,-1);    error(rtn);
    rtn=GT_EndList();    error(rtn);
    }
```

GT_ArcZX

函数原型： short GT_ArcZX(double Z_center,double X_center,double Angle);

函数说明： 该函数实现 ZOX 平面内的两轴圆弧插补。圆弧插补运动的起点坐标是前一段轨迹描述的终点坐标；如是第一条运动轨迹段，则起点是当前位置坐标。

函数参数： Z_center、X_center 是圆弧圆心坐标，坐标单位由用户通过坐标系映射函数自行定义；angle 是旋转角度，其单位是度，正负代表旋转方向，旋转角度的取值范围为-360～360 度。

系　　统： DOS，WINDOWS

适用板卡： 所有 GT 系列卡

相关函数： GT_ArcXY，GT_ArcYZ

函数调用： 举例——立即执行 ZOX 平面圆弧插补命令。

```
void main()
{
    short rtn;
    rtn=GT_SetSynVel(5);        error(rtn);
    rtn=GT_SetSynAcc(1);        error(rtn);
    rtn=GT_ArcZX(40000,30000,180);    error(rtn);
  }
```

GT_ArcZXP

函数原型： short GT_ArcZXP(double Z_end, double X_end, double R, short Dir);

函数说明： 该函数实现 ZOX 平面内的两轴圆弧插补。圆弧插补运动的起点坐标是前一段轨迹描述的终点坐标；如是第一条运动轨迹段，则起点是当前位置坐标。

函数参数： Z_end、X_end 是圆弧终点坐标；R 是圆弧半径并带符号，其符号表示此段圆弧是优弧还是劣弧（正：劣弧，负：优弧），坐标和半径的单位由用户通过坐标系映射函数自行定义；Dir 是圆弧旋转方向，取值为 1 表示正向旋转，-1 表示负向旋转。

系　　统： DOS，WINDOWS

适用板卡： 所有 GT 系列卡

相关函数： GT_ArcXYP，GT_ArcYZP

函数调用： 举例——发送 ZOX 平面圆弧插补命令到缓冲区。

```
void main()
{
    short rtn;
    rtn=GT_StrtList();        error(rtn);
    rtn=GT_MvXYZ(0,0,0,5,0.1);    error(rtn);
    rtn=GT_ArcZXP(40000,0,20000,-1);    error(rtn);
    rtn=GT_EndList();     error(rtn);
 }
```

GT_AuStpOff

函数原型： short GT_AuStpOff(void);

函数说明： 该函数清除当前轴运动出错时电机自动停止标识。如果当前轴在伺服控制过程中满足了事先设定的运动出错条件(即：实际位置误差超过了 GT_SetPosErr()函数设置值)，

控制器将只设置轴运动出错标识，而不自动停止该轴的伺服电机控制输出。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

相关函数：GT_AuStpOn

函数调用：举例——清除第四轴运动出错时电机自动停止标识。

```
void main()
{
    short rtn;
    rtn=GT_Axis(4);       error(rtn);
    rtn=GT_AuStpOff ();   error(rtn);
    }
```

GT_AuStpOn

函数原型：short GT_AuStpOn(void);

函数说明：该函数设置当前轴运动出错时电机自动停止标识。如果当前轴在伺服控制过程中满足了事先设定的运动出错条件（即：实际位置误差超过了 GT_SetPosErr()函数设置值），控制器将会自动停止该轴的伺服电机控制输出，使电机运动停止。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

相关函数：GT_AuStpOff

函数调用：举例——设置第三轴运动出错时电机自动停止标识。

```
void main()
{
    short rtn;
    rtn=GT_Axis(3);   error(rtn);
    rtn=GT_AuStpOff ();   error(rtn);
    }
```

GT_AuUpdtOff

函数原型：short GT_AuUpdtOff(void);

函数说明：该函数清除当前轴模式寄存器的自动刷新标志位。该函数生效后，已满足触发条件的断点将只设置断点标志而不刷新控制轴参数。该标志将一直保持到主机调用 GT_AuUpdtOn()函数为止。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

相关函数：GT_AuUpdtOn

函数调用：举例——清除第二轴模式寄存器的自动刷新标志位。

```
void main()
{
    short rtn;
    rtn=GT_Axis(2);     error(rtn);
    rtn=GT_AuUpdtOff ();   error(rtn);
   }
```

GT_AuUpdtOn

函数原型：short GT_AuUpdtOn(void);

函数说明：该函数使当前轴模式寄存器的自动刷新标志位置 1。该函数生效后，满足触发条件的断点将自动刷新控制轴参数。该标志将一直保持到主机调用 GT_AuUpdtOff()函数为止。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

相关函数：GT_AuUpdtOff

函数调用：举例——清除第一轴模式寄存器的自动刷新标志位。

```
void main()
{
    short rtn;
    rtn=GT_Axis(1);     error(rtn);
    rtn=GT_AuUpdtOn();   error(rtn);
    }
```

GT_Axis

函数原型：short GT_Axis(unsigned short num);

函数说明：该函数设置指定轴为当前轴。此后调用的面向控制轴的命令都针对指定轴，直到再次调用该控制轴寻址命令指向另一个轴。参数 num 表示指定的轴号，在 1、2、3、4 四个数中取值，分别代表第一、二、三、四轴。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

函数调用：举例——设置第二轴为当前轴，之后的命令都针对第二轴。

```
void main()
{
    short rtn;
    rtn=GT_Axis(2);     error(rtn);
    rtn=GT_Setpos(1000);     error(rtn);
    rtn=GT_SetVel(10);       error(rtn);
    rtn=GT_SetAcc(1);     error(rtn);
    }
```

GT_AxisI

函数原型：short GT_AxisI(void);

函数说明：当主机收到事件中断申请时，可以调用该函数设置发出中断申请的轴为当前轴。此后调用的面向控制轴的命令都针对产生中断申请的轴，直到调用其他控制轴寻址命令。

如果设置运动控制器为允许向主机申请事件中断方式，一旦运动控制器遇到满足中断的事件，立即向主机申请中断，主机应对其发出的中断申请进行处理。此时调用 GT_AxisI()命令，运动控制器自动将发出中断申请的控制轴设定为当前轴，不需要再设定每一个轴来检查是否发生中断，使主机快速读取中断状态进行相应处理。

系　　统：DOS

适用板卡：所有 GT 系列卡

函数调用：举例——在中断程序中处理产生中断的轴的正向限位开关事件。

```
void interrupt handler(...)
{
    short rtn;
    unsigned short intr_sts;
    long actl_pos, pos;
    disable();
    rtn=GT_AxisI();      if(rtn!=0) return;
    rtn=GT_GetIntr(&intr_sts);      if(rtn!=0) return;
    if (intr_sts & 0x20)    // positive limit switch error
      {
        rtn=GT_GetAtlPos(&actl_pos);    if(rtn!=0) return;
       pos=actl_pos-20000;
        rtn=GT_SetPos(pos);    if(rtn!=0) return;
       rtn=GT_Update();   if(rtn!=0) return;
        rtn=GT_RstIntr(0x9f);  if(rtn!=0) return;
    }
    enable();
    return;
}
```

GT_AxisOff

函数原型：short GT_AxisOff(void)；

函数说明：该函数使当前轴处于非控制状态并关闭轴驱动使能。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

相关函数：GT_AxisOn

函数调用：举例——使第二轴处于非控制状态并关闭轴驱动使能。

```
void main()
{
    short rtn;
    rtn=GT_Axis(2);   error(rtn);
    rtn=GT_AxisOff();      error(rtn);
   }
```

GT_AxisOn

函数原型：short GT_AxisOn(void)；

函数说明：该函数使当前轴处于控制状态，并使当前轴驱动器使能。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

相关函数：GT_AxisOff

函数调用：举例——使第二轴在设置了数字滤波参数后，处于控制状态，并使轴驱动器伺服使能。

```
void main()
{
    short rtn;
    rtn=GT_Axis(2);        error(rtn);
    rtn=GT_SetKp(10);      error(rtn);
    rtn=GT_Update();           error(rtn);
    rtn=GT_AxisOn();       error(rtn);
   }
```

GT_BrkOff

函数原型：short GT_BrkOff(void);

函数说明：该函数清除当前轴已设置的但未被触发的断点。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

相关函数：GT_BrkOn

函数调用：举例——清除第一轴已设置的但未被触发的断点。

```
void main()
{
    short rtn;
    rtn=GT_Axis(1); error(rtn);
    rtn=GT_BrkOff();        error(rtn);
    }
```

GT_CaptHome

函数原型：short GT_CaptHome(void);

函数说明：该函数设置 Home 信号捕获位置事件。调用该函数后，位置捕获寄存器将记录 Home 信号到来时的实际位置。执行一次 GT_CaptHome()只能捕获到一次 Home 位置信息。要想捕获下一个 Home 位置信息，必须清除对应的状态标志位，即调用 GT_ClrSts()或 GT_RstSts()函数，然后再设置 Home 信号捕获位置事件。捕获 Home 信号的逻辑采用硬件完成，与运动轴的速度无关。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

相关函数：GT_CaptIndex，GT_CaptProb

函数调用：举例——设置第三轴 Home 信号捕获位置事件。

```
void main()
{
    short rtn;
    rtn=GT_Axis(3);   error(rtn);
    rtn=GT_ClrSts();  error(rtn);
    rtn=GT_CaptHome();    error(rtn);
    }
```

GT_CaptIndex

函数原型：short GT_CaptIndex(void);

函数说明：该函数设置 Index 信号捕获位置事件。调用该函数后，位置捕获寄存器将记录编码器 Index 信号到来时的实际位置。执行一次 GT_CaptIndex()只能捕获到一次 Index 位置信息。要想捕获下一个 Index 位置信息，必须清除对应的状态标志位，即调用函数 GT_ClrSts()或 GT_RstSts()函数，然后再设置 Index 信号捕获位置事件。捕获到的实际位置可以作为运动轴的精确坐标原点。捕获 Index 信号的逻辑采用硬件完成，与运动轴的速度无关。

系　　统：DOS，WINDOWS

适用板卡：SV

相关函数：GT_CaptHome，GT_CaptProb

函数调用：举例——设置第三轴 Index 信号捕获位置事件。

```
void main()
{
    short rtn;
    rtn=GT_Axis(3);        error(rtn);
    rtn=GT_ClrSts();       error(rtn);
    rtn=GT_CaptIndex();  error(rtn);
    }
```

GT_CaptProb

函数原型：short GT_CaptProb(void);

函数说明：该函数设置捕获探针输入信号事件。本运动控制器以通用 IO 输入中的 0 号通道（EXI0）作为探针输入，调用该函数后，所有控制轴的位置捕获寄存器以及辅助编码器的位置捕获寄存器将记录探针信号到来时的实际位置。

当捕获事件发生时，所有控制轴状态字中表示有捕获发生的标志（即 bit3）置位。

执行一次 GT_CaptProb()只能捕获到一次探针输入信号的位置信息。要想捕获下一个位置信息，必须重新调用本函数设置捕获探针输入信号事件。捕获到的实际位置同样可以作为控制轴的精确坐标原点。捕获探针输入信号的逻辑采用硬件完成，与运动轴的速度无关。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

相关函数：GT_CaptHome，GT_CaptIndex

函数调用：举例——设置捕获探针输入信号事件，在运动过程中循环检测状态，在产生捕获时读取第二轴捕获到的实际位置值并打印。

```
void main()
{
    short rtn;
    unsigned short status;
    long actl_pos;
    rtn=GT_CaptProb();   error(rtn);
    ….
    rtn=GT_Axis(2); error(rtn);
    rtn=GT_GetSts(&status);   error(rtn);
    while(status&0x400)
```

```
        {
            if(status&0x4)
            {
                rtn=GT_GetAtlPos(&actl_pos);        error(rtn);
                printf("the capture pos of axis 2 is: %ld\n",actl_pos);
                break;
            }
            rtn=GT_GetSts(&status);       error(rtn);
        }
    }
```

GT_Close

函数原型：short GT_Close(void);
函数说明：关闭运动控制器设备，用户程序结束时必须调用此函数。
系　　统：DOS(PCI 总线)，WINDOWS
适用板卡：所有 GT 系列卡
相关函数：GT_Open

GT_CloseLp

函数原型：short GT_CloseLp(void);
函数说明：该函数设置当前轴为闭环控制状态。当系统上电时，各控制轴的缺省状态是闭环控制状态。如果当前轴已经使能（即调用了 GT_AxisOn()函数）而又处于开环状态，调用本函数有时会使电机突然转动。这是由于在此前的开环状态，电机可能产生转动或零漂，使电机的实际位置与目标位置产生较大的偏差。所以在当前轴已经伺服使能的情况下，在调用本函数之前，最好先禁止当前轴伺服使能（即调用 GT_AxisOff()函数），再调用 GT_SynchPos()函数令目标位置等于实际位置。
系　　统：DOS，WINDOWS
适用板卡：SV
相关函数：GT_OpenLp
函数调用：举例——在确保没有危险的情况下，设置第一轴为闭环控制状态。

```
    void main()
    {
        short rtn;
        rtn=GT_Axis(1);        error(rtn);
        rtn=GT_AxisOff();      error(rtn);
        rtn=GT_SynchPos();     error(rtn);
        rtn=GT_Update();            error(rtn);
        rtn=GT_ClosLp();            error(rtn);
        }
```

GT_ClearInt

函数原型：short GT_ClearInt(unsigned short CardNo)
函数说明：清除由控制卡产生的中断，在中断服务程序（ISR）结束前必须调用此函数以

解除控制卡产生的中断。

函数参数：CardNo 为对应的控制卡卡号，参看 GT_SwitchtoCardNo 中的说明。在使用单一卡时，此值永远设为 0。

函数返回：0 表示成功，-1 表示失败。

函数调用：参见 DOS 中断处理

适用板卡：PCI 总线卡

系　　统：DOS

函数调用：GT_ClearInt 只能用在中断服务程序中才有效，典型使用方式如下：

```
void interrupt   My_Isr(...) //用户将使用的中断服务程序
{
    ....
    ...//用户代码
    GT_ClearInt(0);
    outportb(0x20,0x20);
    outportb(0xa0,0x20); //向中断控制器声明中断服务结束。
}
```

GT_ClrSts

函数原型：short GT_ClrSts(void);

函数说明：该函数清除当前轴事件状态标志位，但是并不影响运动控制器事件触发的中断，仅仅是把当前轴状态字的事件标志位清零，使下一次新的事件发生时可以产生标志。关于状态字的标志位定义可参见*表 11-1*。

执行 GT_ClrSts()后，将清除 Bit0~Bit7，下一次再有新的事件发生后，对应位将会重新置为“1”。对于 bit8~bit15 位完全由运动控制器来管理的标志位，本函数将不起作用。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

函数调用：举例——清除第四轴事件状态标志位。

```
void main()
{
    short rtn;
    rtn=GT_Axis(4);   error(rtn);
    rtn=GT_ClrSts();    error(rtn);
    }
```

表 11-1 状态寄存器各标志位定义

位	定义
0	运动完成标志位。如果控制轴运动完成，该位置 1。该标志在电子齿轮运动模式时无效。
1	电机伺服驱动器报警标志位。如果控制轴驱动器报警，该位置 1。
2	断点到达标志位。在设置断点条件下，断点条件满足，该位置 1。
3	Index/Home 标志位。在设置位置捕获命令后，控制器检测到要求的 Index/Home 捕获条件，该位置 1。
4	运动出错标志位。如果位置误差超过允许范围(参考 4.4.7.3 说明)，控制

位	定义
	器将该位置 1。只有控制轴不再处于运动出错状态时，才能对它复位。
5	正向限位开关触发标志位。如果正向限位开关被触发，该位置 1。
6	负向限位开关触发标志位。如果负向限位开关被触发，该位置 1。
7	命令出错标志位。如果出现命令错误，控制器将该位置 1。
8	电机开环/闭环状态(1 表示闭环，0 表示开环)。
9	电机伺服使能/禁止状态(1 表示使能，0 表示禁止)。
10	运动状态标志位。它连续指示控制轴是否在运动。如果在运动，为 1；如果静止，为 0。
11	限位开关使能/禁止状态（1 表示使能，0 表示禁止)。
12 13	当前轴号标志(13bit=高位,12bit=低位)。 当前轴号的编码如下 Bit 13 Bit12 轴 0 0 1 0 1 2 1 0 3 1 1 4
14	设定 Home 开关信号捕获标志
15	设定 Index 信号捕获标志

GT_CrdAuStpOff

函数原型：short GT_CrdAuStpOff(void);

函数说明：该函数清除坐标系状态寄存器的轴异常自动停止允许标志。调用该函数之后，当坐标系映射的某一个控制轴运动出现异常时（例如伺服报警、限位到、轴禁止），运动控制器只是禁止该控制轴运动，其他映射到坐标系中的控制轴继续运动。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

相关函数：GT_CrdAuStpOn

函数调用：举例——清除坐标系状态寄存器中轴异常自动停止允许标志。

```
void main()
{
    short rtn;
    rtn=GT_CrdAuStpOff();    error(rtn);
    }
```

GT_CtrlMode

函数原型：short GT_CtrlMode(int mode);

函数说明：该函数设置当前轴的输出模式为模拟电压输出或脉冲输出。

函数参数：mode 表示输出模式，有两种：0 表示为模拟电压输出模式，1 表示为脉冲输出模式。运动控制器默认的状态为模拟电压输出模式。

系　　统：DOS，WINDOWS

适用板卡：SV

函数调用：举例——设置第一轴为脉冲输出模式，以控制步进电机。

```
void main()
{
    short rtn;
```

```
rtn=GT_Axis(1);   error(rtn);
rtn=GT_CtrlMode(1);    error(rtn);
}
```

GT_CrdAuStpOn

函数原型：short GT_CrdAuStpOn(void);

函数说明：该函数设置坐标系状态寄存器的轴异常自动停止允许标志。调用该函数之后，当坐标系映射的某一个控制轴运动出现异常时（例如伺服报警、限位到、轴禁止），运动控制器停止基于坐标系的运动规划。即映射到坐标系中的各轴的运动全部停止。控制器缺省情况是调用 GT_CrdAuStpOn()状态。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

相关函数：GT_CrdAuStpOff

函数调用：GT_CrdAuStpOn()

GT_DrvRst

函数原型：short GT_DrvRst(void);

函数说明：在当前轴驱动器发生故障报警时，调用该函数后，控制器向轴驱动器发出复位信号使驱动器复位。在调用该函数之前，当前轴需处于驱动禁止状态，否则,该命令无效。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

函数调用：举例——如果第四轴产生驱动报警信号，复位驱动器，并清除报警信号。

```
void main()
{
    short rtn;
    unsigned short status;
    rtn=GT_Axis(4);   error(rtn);
    rtn=GT_GetSts(&status);   error(rtn);
    if(status&0x2)
    {
        rtn=GT_DrvRst();      error(rtn);
        rtn=GT_RstSts(0xfffd);      error(rtn);
     }
}
```

GT_EncPos

函数原型：short GT_EncPos(short Enc_Num, long* Actl_pos);

函数说明：该函数用来读取辅助编码器实际位置值。

参数 Enc_Num 表示需要读取的辅助编码器号，运动控制器上带有两路辅助编码器，分别为一号和二号；参数*Actl_pos 取回指定辅助编码器的实际位置。

系　　统：DOS，WINDOWS

适用板卡：SV, SG, SP

相关函数：GT_EncVel

函数调用：举例——读取二号辅助编码器位置值并打印。

```
void main()
{
    short rtn;
    long actl_pos;
    rtn=GT_EncPos(2, &actl_pos);        error(rtn);
    printf("the actual position of assistant encoder 2 is: %ld\n", actl_pos);
}
```

GT_EncSns

函数原型：short GT_EncSns(unsigned int Sense);

函数说明：运动控制器要求控制轴（电机）运动的正方向和相应编码器计数的正方向一致，这样才能形成正确的负反馈。如果因接线错误或其他原因，使两者方向相反，用户可调用本函数使用软件方法校正编码器计数正方向。使编码器计数正方向与电机运动正方向保持一致。

参数 Sense 中 bit0 到 bit6 表示各个对应编码器计数方向是否需要反向，如表 2-3 所示。在表中对应位如果设置为零时，表示不修改对应轴编码器计数方向，反之，如果设置为“1”，表示修改对应轴编码器计数方向。寄存器默认值为全“0”。

系　　统：DOS，WINDOWS

适用板卡：SV

函数调用：举例——将第一和第二轴编码器反向，第三和第四轴编码器以及一、二号辅助编码器方向不变。

```
void main()
{
    short rtn;
    rtn=GT_EncSns(3);    error(rtn);
    }
```

表 11-2 编码器寄存器位定义表

Bit	说明	定义
6-15	没有使用，设置为零值	
5	2 号辅助编码器	0：不变　　1：取反
4	1 号辅助编码器	0：不变　　1：取反
3	Axis#4	0：不变　　1：取反
2	Axis#3	0：不变　　1：取反
1	Axis#2	0：不变　　1：取反
0	Axis#1	0：不变　　1：取反

GT_EncVel

函数原型：short GT_EncVel(short Enc_Num, double* Actl_vel);

函数说明：该函数用来读取辅助编码器实际位置值。

参数 Enc_Num 表示需要读取的辅助编码器号，运动控制器上带有两路辅助编码器，分别为一号和二号；参数*Actl_vel 取回指定辅助编码器的实际速度。

系　　统：DOS，WINDOWS
适用板卡：SV, SG, SP
相关函数：GT_EncPos
函数调用：举例——读取二号辅助编码器速度值并打印。

```
void main()
{
    short rtn;
    double actl_vel;
    rtn=GT_EncVel(2, &actl_vel);    error(rtn);
    printf("the actual velocity of assistant encoder 2 is: %f\n", actl_vel);
}
```

GT_EndList

函数原型：short GT_EndList(void);
函数说明：在缓冲区命令输入状态，调用本函数结束缓冲区命令输入状态。
系　　统：DOS，WINDOWS
适用板卡：所有 GT 系列卡
相关函数：GT_StrtList，GT_AddList
函数调用：举例——在缓冲区命令输入状态下，结束缓冲区命令输入状态。

```
void main()
{
    short rtn;
    rtn=GT_StrtList();              error(rtn);
    rtn=GT_MvXYZ(0,0,0,16,3.7);     error(rtn);
    rtn=GT_LnXYZ(1234,5678,9013);   error(rtn);
    rtn=GT_ArcXY(2345,6789,360);    error(rtn);
    rtn=GT_EndList();    error(rtn);
    }
```

GT_EStpMtn

函数原型：short GT_EStpMtn(void);
函数说明：该函数立即停止基于坐标系的多轴协调运动命令。该函数执行后没有减速过程，当前运动合成速度和目标合成速度立即为零。在缓冲区命令执行状态，运动停止后将保存运动停止时的必要信息，以便此后调用 GT_StrtMtn ()函数继续缓冲区命令执行。在立即命令输入执行状态，运动停止后将丢弃当前运动信息。

用户不能在执行定位指令时发出中断命令，即运动到定位点的过程不能被打断，这时发出中断命令会引起运动出错。

系　　统：DOS，WINDOWS
适用板卡：所有 GT 系列卡
相关函数：GT_StpMtn
函数调用：举例——缓冲区中命令执行过程中，急停坐标系运动。

```
void main()
{
    short rtn;
```

```
rtn=GT_StrtList();          error(rtn);
rtn=GT_MvXYZ(0,0,0,16,3.7);     error(rtn);
rtn=GT_LnXYZ(1234,5678,9013);   error(rtn);
rtn=GT_StrtMtn();           error(rtn);
rtn=GT_ArcXY(2345,6789,360);    error(rtn);
……
rtn=GT_EndList();    error(rtn);
rtn=GT_EStpMtn();    error(rtn);
}
```

GT_EvntIntr

函数原型： short GT_EvntIntr(void);
函数说明： 该函数设置运动控制器向主机申请的中断为轴事件中断。
系　　统： DOS，WINDOWS
适用板卡： 所有 GT 系列卡
相关函数： GT_TmrIntr
函数调用： GT_EvntIntr()

GT_ExInpt

函数原型： short GT_ExInpt (unsigned short * Data);
函数说明： 该函数获得运动控制器通用数字量输入的状态。
函数参数： *Data 返回该状态，其各位与通用数字量输入口的对应关系为：

位一定义	位一定义	位一定义	位一定义
Bit0----EXI0	Bit1----EXI1	Bit2----EXI2	Bit3----EXI3
Bit4----EXI4	Bit5----EXI5	Bit6----EXI6	Bit7----EXI7
Bit8----EXI8	Bit9----EXI9	Bit10----EXI10	Bit11----EXI11
Bit12----EXI12	Bit13----EXI13	Bit14----EXI14	Bit15----EXI15

系　　统： DOS，WINDOWS

适用板卡： 所有 GT 系列卡
相关函数： GT_ExOpt
函数调用： 举例——读取通用数字量输入的状态，若 EXI8 为高电平则停止坐标系运动。

```
void main()
{
    short rtn, ex_inp;
    rtn=GT_ExInpt(&ex_inp);    error(rtn);
    if(ex_inp&0x100)
    {
        rtn=GT_StpMtn();      error(rtn);
        }
}
```

GT_ExOpt

函数原型： short GT_ExOpt(unsigned short Data);

函数说明： 该函数设置运动控制器通用数字量输出的状态。

函数参数： Data 是所要设置的状态，其各位与通用数字输出口的对应关系为：

位一定义	位一定义	位一定义	位一定义
Bit0----EXO0	Bit1----EXO1	Bit2----EXO2	Bit3----EXO3
Bit4----EXO4	Bit5----EXO5	Bit6----EXO6	Bit7----EXO7
Bit8----EXO8	Bit9----EXO9	Bit10----EXO10	Bit11----EXO11
Bit12----EXO12	Bit13----EXO13	Bit14----EXO14	Bit15----EXO15

系　　统： DOS，WINDOWS

适用板卡： 所有 GT 系列卡

相关函数： GT_ExInpt

函数调用： 举例——若通用数字量输入 EXI5 为高电平，则设置通用数字量输出的 EXO0 为高电平。

```
void main()
{
    short rtn, ex_inp;
    rtn=GT_ExInpt(&ex_inp);        error(rtn);
    if(ex_inp&0x20)
    {
        rtn=GT_ExOpt(0x1);         error(rtn);
        }
}
```

GT_ExtBrk

函数原型： short GT_ExtBrk(void);

函数说明： 该函数设置当前轴的断点模式为原点开关触发断点模式。该函数调用后，控制器在检测到轴原点开关有效时，触发断点使轴状态寄存器的断点标识置 1 并清除该断点模式，即除非主机重新设置断点，控制器将不再触发该断点。同时，在轴模式状态寄存器的自动刷新标识置 1 的条件下，控制器将刷新当前轴的所有双缓冲结构参数和命令。

系　　统： DOS，WINDOWS

适用板卡： 所有 GT 系列卡

函数调用： 举例——设置第二轴的断点模式为原点开关触发断点模式，并在碰到原点开关时自动刷新平滑停止的命令。

```
void main()
{
    short rtn;
    rtn=GT_Axis(2);            error(rtn);
    rtn=GT_AuUpdtOn();         error(rtn);
    rtn=GT_CaptHome();         error(rtn);
    rtn=GT_ExtBrk();               error(rtn);
    rtn=GT_PrflT();            error(rtn);
```

```
    rtn=GT_SetPos(87654);     error(rtn);
    rtn=GT_SetVel(32);        error(rtn);
    rtn=GT_SetAcc(1.1);       error(rtn);
    rtn=GT_Update();              error(rtn);
    rtn=GT_SmthStp();         error(rtn);
  }
```

GT_GetAcc

函数原型： short GT_GetAcc(double * Acc);
函数说明： 该函数获得当前轴由 GT_SetAcc()函数设置的加速度值。
函数参数： *Acc 返回该加速度值。
系　　统： DOS，WINDOWS
适用板卡： 所有 GT 系列卡
相关函数： GT_SetAcc
函数调用： 举例——读取第三轴设定的加速度值。

```
void main()
{
    short rtn;
    double acc;
    rtn=GT_Axis(3);       error(rtn);
    rtn=GT_GetAcc(&acc);  error(rtn);
    }
```

GT_GetAccLmt

函数原型： short GT_GetAccLmt(unsigned long * Acclmt);
函数说明： 该函数获得 GT_SetAccLmt ()函数设置的当前轴允许加速度极限。
函数参数： *Acclmt 返回该加速度允许值。该极限值只在坐标系运动下起作用。
系　　统： DOS，WINDOWS
适用板卡： 所有 GT 系列卡
相关函数： GT_SetAccLmt
函数调用： 举例——读取第四轴设定的加速度极限值。

```
void main()
{
    short rtn;
    double acc_lmt;
    rtn=GT_Axis(4);       error(rtn);
    rtn=GT_GetAccLmt(&acc_lmt);error(rtn);
    }
```

GT_GetAdc

函数原型： short GT_GetAdc(short Channel, short* Adc_Data);
函数说明： 该函数读取 AD 转换结果。AD 转换的采样频率八路则为 770Hz。
函数参数： 参数 Channel 设置 AD 通道号，取值范围为：1～8；参数*Adc_Data 返回 AD

转换的结果，取值范围为：-2048～2047（表示电压：-10V～10V）。

系　　统：DOS，WINDOWS

适用板卡：订购了该功能的卡

函数调用：可在任何时刻调用该函数，举例——读取通道二的 AD 转换结果。

```
void main()
{
    short rtn;
    short Adc_Data;
    rtn=GT_GetAdc(2,&Adc_Data);
    if(rtn!=0) return;
}
```

GT_GetAddr

函数原型：short GT_GetAddr(unsigned short * Base_addr);

函数说明：该函数获得 GT_SetAddr()函数设置的主机与运动控制器之间的通讯基地址或上电时刻主机默认的基地址（0x300）。

函数参数：*Base_addr 返回目前通讯所使用的基地址，如果用户使用 GT_SetAddr()设置了基地址而该基地址有误，那么参数返回的也将是这个错误的基地址。

函数返回值：与别的函数不同的是：由于不与运动控制器发生数据交流，该函数的返回值永远是 0。

系　　统：DOS

适用板卡：ISA 总线卡

相关函数：GT_SetAddr

函数调用：举例——读取基地址，并打印。

```
void main()
{
    short rtn;
    unsigned shor base_addr;
    rtn=GT_GetAddr(&base_addr); error(rtn);
    printf("the base address is: %d\n",base_addr);
}
```

GT_GetAtlErr

函数原型：short GT_GetAtlErr (short * Aerr);

函数说明：该函数获得当前轴的实际位置误差。即当前规划位置与当前实际位置之差。该函数常被用来监测伺服控制位置误差大小。

函数参数：*Aerr 返回该实际位置误差。

系　　统：DOS，WINDOWS

适用板卡：SV

函数调用：举例——读取第二轴的实际位置误差，并打印。

```
void main()
{
    short rtn, actl_err;
    rtn=GT_Axis(2);        error(rtn);
```

```
        rtn=GT_GetAtlErr(&actl_err);          error(rtn);
        printf("the actual error is: %d\n",actl_err);
}
```

GT_GetAtlPos

函数原型： short GT_GetAtlPos(long * Apos);
函数说明： 该函数获得当前轴的实际位置值。
函数参数： *Apos 返回该实际位置值。
系　　统： DOS，WINDOWS
适用板卡： 所有 GT 系列卡
相关函数： GT_SetAtlPos
函数调用： 举例——读取第一轴的实际位置，并打印。

```
void main()
{
    short rtn;
    long actl_pos;
    rtn=GT_Axis(2); error(rtn);
    rtn=GT_GetAtlPos(&actl_pos);          error(rtn);
    printf("the actual pos is: %ld\n",actl_pos);
}
```

GT_GetBgCommandResult

函数原型： short WINAPI GT_GetBgCommandResult (PBGCOMMANDSET BgCmdset, ULONG CmdsetSize);
函数说明： 获取控制器后台命令集执行结果。
函数参数： pBgCmdset 为后台命令集起始地址。其中 GENERAL_COMMAND 结构的 result 变量保存了命令执行结果，0 表示命令执行成功，-1 表示相应命令执行失败。参数 CmdsetSize 为后台命令集的空间大小，字节为单位。
函数返回： 0 表示成功，-1 表示失败。
系　　统： WINDOWS
适用板卡： PCI 总线卡
相关函数： GT_SetBgCommandSet

GT_GetBrkCn

函数原型： short GT_GetBrkCn(long * Brk);
函数说明： 该函数获得当前轴由 GT_SetBrkCn()函数设定的断点位置。
函数参数： *Brk 返回该断点位置。
系　　统： DOS，WINDOWS
适用板卡： 所有 GT 系列卡
相关函数： GT_SetBrkCn
函数调用： GT_GetBrkCn()

GT_GetBrkPnt

函数原型： short GT_GetBrkPnt(double * Coord);

函数说明： 在进行基于坐标系的轨迹规划和运动时，主机能够使用 GT_EStpMtn()和GT_StpMtn()函数中断缓冲区命令执行过程。

函数参数： *Coord 返回中断点的坐标，坐标按照 X、Y、Z、A 的顺序存放在数组 coord 中。

系　　统： DOS，WINDOWS

适用板卡： 所有 GT 系列卡

函数调用： 举例——读取坐标系中断处的坐标位置，并打印。

```
void main()
{
    short rtn;
    double break_pos[4];
    rtn=GT_GetBrkPnt(break_pos); error(rtn);
    printf("the break pos in coordinate are: %f, %f, %f, %f \n", break_pos[0],
break_pos[1], break_pos[2], break_pos[3]);
}
```

GT_GetCapt

函数原型： short GT_GetCapt (long * Capt);

函数说明： 该函数获得当前轴捕获到的 Index 或 Home 信号出现时的位置值，该值在运动状态下一直保持不变，直到下一次捕获事件发生。

函数参数： *Capt 返回当前轴的该捕获位置值。

系　　统： DOS，WINDOWS

适用板卡： 所有 GT 系列卡

相关函数： GT_CaptHome，GT_CaptIndex，GT_CaptProb

函数调用： 举例——当第三轴产生 Home 信号捕获时，读取发生点的位置值，并将位置停在该点。

```
void main()
{
    short rtn;
    unsigned short status;
    long capt_pos;
    rtn=GT_GetSts(&status);    error(rtn);
    if(status&0x8)
    {
        rtn=GT_GetCapt(&capt_pos);    error(rtn);
        rtn=GT_SetPos(capt_pos);      error(rtn);
        }
}
```

GT_GetCmdSts

函数原型： short GT_GetCmdSts(unsigned short * Cstatus);

函数说明：该函数获得控制器命令状态寄存器的内容。

函数参数：*Cstatus 返回该寄存器的值。用户主程序中，应检测每个函数的返回值，以判断通讯及命令的执行状态。如果命令返回值为 1 时，指示主机发出的命令错误，运动控制器报告主机命令出错。一旦出现命令错误，运动控制器忽略这些非法运动命令。主机可通过调用本命令得到命令出错的原因。该寄存器为一 16 位寄存器，各位定义如下表：

位	定义
Bit0	1：表示命令输入的控制参数溢出，产生该出错标志的命令有：GT_SetPos()，GT_SetVel()，GT_SetAcc()，GT_SetAtlPos()等
Bit1	1：表示命令输入的控制参数非法，产生该出错标志的命令有：GT_SetVel()，GT_SetAcc()，GT_SetJerk()，GT_SetMAcc()，GT_SetMtrLmt()，GT_SetKp()，GT_SetKi()，GT_SetKd()，GT_SetKvff()，GT_SetKaff()，GT_SetILmt()，GT_SetPosErr()等
Bit2	1：表示主机发送 GT_MltiUpdt (value)命令而 value=0
Bit3	1：表示 GT_DrvRst()命令使用非法。当前轴在伺服使能状态时，主机发送该命令将产生出错标志
Bit4	1：表示当控制器没有事件中断产生，而主机发送与中断有关的命令
Bit5	（空）
Bit6	1：表示当前轴正在运动时，主机发送修改当前轴的工作模式命令（当前轴处于电子齿轮模式时除外）
Bit7	1：表示当前轴状态寄存器的位置捕获完成标志为 1，或已经设定了 GT_CaptIndex()（GT_CaptHome()）命令但位置尚未捕获到时，而主机又发送了 GT_CaptIndex()命令
Bit8	1：表示当前轴状态寄存器的位置捕获完成标志为 1，或已经设定了 GT_CaptIndex()（GT_CaptHome()）命令但位置尚未捕获到时，而主机又发送了 GT_CaptHome()命令
Bit9	1：表示当前轴状态寄存器的驱动器报警标志为 1 时，主机发送 GT_AxisOn()命令
Bit10	（空）
Bit11	1:当前轴状态寄存器中“运动状态标志”为 1 时,主机发送 GT_ZeroPos()命令，该标志位置“1”；在当前轴运动模式为速度控制模式时，主机发送 GT_SynchPos()命令并使其生效，该标志位置“1”；当前轴的状态寄存器中“运动状态标志”为 1 时，用 GT_Update()或 GT_MltiUpdt()修改当前轴控制参数,而这些参数在当前运动模式下是不允许被修改的（如：在 S-曲线运动模式下，在电机运动中，主机发送 GT_SetVel()命令并发送 GT_Update()或 GT_MltiUpdt()）
Bit12	1：表示面向坐标系的命令非法。包括： 建立坐标系时，所映射的物理轴正在运动； 缓冲区命令执行状态时，调用 GT_StrtMtn()命令； 缓冲区命令执行状态时，调用 GT_StrtList()命令； 缓冲区命令输入状态时，调用 GT_AddList ()命令； 立即命令输入执行状态，运动未完成时，调用新的运动描述命令。

位	定义
Bit13	1：表示调用 GT_MvXY ()、GT_MvXYZ()、GT_MvXYZA()命令出错
Bit14	（空）
Bit15	1：表示缓冲区满。此时由于运动控制器缓冲区满，刚刚调用的运动描述命令没有被运动控制器接收，主机需要继续重复调用此命令，直至运动控制器接收该命令为止。

系　　统：DOS，WINDOWS
适用板卡：所有 GT 系列卡
函数调用：可在任何时刻调用该函数。

GT_GetCrdSts

函数原型：short GT_GetCrdSts(unsigned short * Status);
函数说明：该函数获得基于坐标系运动规划的状态字。
函数参数：*Status 返回该状态字，意义参见下表：

位	定义
0	1＝坐标系轨迹运动全部完成或无坐标系轨迹运动
1	1＝缓冲区运动段编写结束（发出 GT_EndList()或 GT_StpMtn()、GT_EStpMtn()时，该位置位）
2	1＝伺服周期过小，引起运动错误（该位置 1 时，请立即修改伺服周期）
3	1＝面向坐标系的命令出错
4	1＝单段轨迹运动完成
5	1＝加速度超限报警（不论哪一个坐标轴的加速度分量超限，都将引起该位置位）
6	1＝异常自动停止允许（当坐标系中相关轴伺服报警或遇限位等异常状态时，若该位置位则将自动停止整个坐标系运动）
7	1＝处于立即命令输入状态；0=处于缓冲区命令输入或执行状态
8	保留
9	1＝坐标系中相关电机轴出现异常，并自动停止坐标系运动
10	1＝坐标系运动发生异常错误（正常情况下该标志不会置起，除非运动控制器运行出现了异常，如用户检测到该位，请立刻中断使用，并将控制器断电后再重新启动）
11～12	保留，默认状态为 0
13	1＝缓冲区空（当启动缓冲区运动后，缓冲区中没有运动描述命令时，该位置位）
15	保留，默认状态为 0

系　　统：DOS，WINDOWS
适用板卡：所有 GT 系列卡
函数调用：举例——读取坐标系运动规划的状态，当发现 bit2 位置位时，将所有轴伺服关断。

```
void main()
{
    short rtn;
```

```
        unsigned short status;
        rtn=GT_GetCrdSts(&status);     error(rtn);
        if(status&0x4)
        {
            rtn=GT_Axis(1);        error(rtn);
            rtn=GT_AxisOff();      error(rtn);
            rtn=GT_Axis(2);        error(rtn);
            rtn=GT_AxisOff();      error(rtn);
            rtn=GT_Axis(3);        error(rtn);
            rtn=GT_AxisOff();      error(rtn);
            rtn=GT_Axis(4);        error(rtn);
            rtn=GT_AxisOff();      error(rtn);
          }
}
```

GT_GetCurrentCardNo

函数原型： short GT_GetCurrentCardNo(void);
函数说明： 该函数获取当前控制卡的卡号。当前控制卡的卡号取值范围为 0--15。
系　　统： DOS、WINDOWS
适用板卡： PCI 总线卡
相关函数： GT_SwitchtoCardNo
函数调用： GT_GetCurrentCardNo (1)

GT_GetILmt

函数原型： short GT_GetILmt(unsigned short * Ilm);
函数说明： 该函数获得当前轴由 GT_SetILmt ()函数设置的误差积分限。
函数参数： *Ilm 返回该误差积分限。
系　　统： DOS，WINDOWS
适用板卡： SV
相关函数： GT_SetILmt

GT_GetIntgr

函数原型： short GT_GetIntgr(short * Intgr);
函数说明： 该函数获得当前轴伺服滤波器的积分误差。
函数参数： *Intgr 返回积分误差。
系　　统： DOS，WINDOWS
适用板卡： SV

GT_GetIntr

函数原型： short GT_GetIntr(unsigned * Status);
函数说明： 该函数获得产生中断申请轴的状态字。如果执行这个命令但运动控制器又没有产生中断申请时，GT_GetIntr()获得当前轴的状态字。
函数参数： *Status 返回该状态字，其意义参见 GT_ClrSts()函数的*表 11-1*。第 0－6 位表

示的是与中断相关的事件，当对应事件的中断屏蔽字设置为中断允许时，若中断条件满足，运动控制器就向主机发出中断申请，主机响应了这个中断后，就可以调用 GT_GetIntr()来查询中断事件类型。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

特别注意：在 WINDOWS 环境下，使用 GT_GetIntr 函数得到的是运动控制器上次中断产生时的中断状态字。Status 为 0：运动控制器没有产生过中断或上次产生的中断为时间中断；非 0：具体含义参看下表，注意 Status 的 Bit12, Bit13 标识的是上次事件中断的产生轴。在中断服务程序中可以调用该函数判断中断类型。

GT_GetIntrMsk

函数原型：short GT_GetIntrMsk (unsigned short * Mask);

函数说明：该函数获得 GT_SetIntrMsk ()函数设置的当前轴的中断屏蔽字。

函数参数：*Mask 返回该中断屏蔽字，其定义参考 GT_RstIntr()函数的位定义表及函数 GT_SetIntrMsk()的说明。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

相关函数：GT_SetIntrMsk

GT_GetIntrTm

函数原型：short GT_GetIntrTm(unsigned short * Timer);

函数说明：该函数获得 GT_SetIntrTm()函数设置的运动控制器定时中断的时间常数。

函数参数：*Timer 返回该时间常数。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

相关函数：GT_SetIntrTm

GT_GetJerk

函数原型： short GT_GetJerk (double * Jerk);

函数说明：该函数获得当前轴由 GT_SetJerk ()函数设置的加加速度值。

函数参数：双精度参数*Jerk 返回该加加速度值。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

相关函数：GT_SetJerk

GT_GetKaff

函数原型：short GT_GetKaff (unsigned short * Kaff);

函数说明：该函数获得当前轴由 GT_SetKaff()函数设置的加速度前馈增益。

函数参数：*Kaff 返回该加速度前馈增益。

系　　统：DOS，WINDOWS

适用板卡：SV

相关函数：GT_SetKaff

GT_GetKd

函数原型： short GT_GetKd(unsigned short * Kd);
函数说明： 该函数获得当前轴由 GT_SetKd()函数设置的微分增益。
函数参数： *Kd 返回该微分增益。
系　　统： DOS，WINDOWS
适用板卡： SV
相关函数： GT_SetKd

GT_GetKi

函数原型： short GT_GetKi (unsigned short * Ki);
函数说明： 该函数获得当前轴由 GT_SetKi ()函数设置的积分增益。
函数参数： *Ki 返回该积分增益。
系　　统： DOS，WINDOWS
适用板卡： SV
相关函数： GT_SetKi

GT_GetKp

函数原型： short GT_GetKp(unsigned short * Kp);
函数说明： 该函数获得当前轴由 GT_SetKp()函数设置的比例增益。
函数参数： *Kp 返回该比例增益。
系　　统： DOS，WINDOWS
适用板卡： SV
相关函数： GT_SetKp

GT_GetKvff

函数原型： short GT_GetKvff (unsigned short * Kvff);
函数说明： 该函数获得当前轴由 GT_SetKvff()函数设置的速度前馈增益。
函数参数： *Kvff 返回该速度前馈增益。
系　　统： DOS，WINDOWS
适用板卡： SV
相关函数： GT_SetKvff

GT_GetLmtSwt

函数原型： short GT_GetLmtSwt (unsigned short * Switch);
函数说明： 该函数获得当前限位开关实际状态。
函数参数： *Switch 返回限位开关输入信号电平，定义见下表。某位如果为 1，表示相应限位开关输入信号为高电平；某位如果为 0，表示相应限位开关输入信号为低电平。该函数与 GT_LmtSns()函数无关，只反应限位开关实际状态。

Bit	说明	定义
8-15	没有使用，设置为零值	
7	Axis #4 负向限位开关状态位	0：高电平　1：低电平

6	Axis #4 正向限位开关状态位	0：高电平 1：低电平
5	Axis #3 负向限位开关状态位	0：高电平 1：低电平
4	Axis #3 正向限位开关状态位	0：高电平 1：低电平
3	Axis #2 负向限位开关状态位	0：高电平 1：低电平
2	Axis #2 正向限位开关状态位	0：高电平 1：低电平
1	Axis #1 负向限位开关状态位	0：高电平 1：低电平
0	Axis #1 正向限位开关状态位	0：高电平 1：低电平

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

GT_GetMAcc

函数原型 short GT_GetMAcc(double * Macc);

函数说明：该函数获得当前轴由 GT_SetMAcc()函数设置的最大加速度值。

函数参数：双精度参数*Macc 是所需的最大加速度值。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

相关函数：GT_SetMAcc

GT_GetMode

函数原型：short GT_GetMode (unsigned short * Mode);

函数说明：该函数获得当前轴控制模式字。

函数参数：*Mode 返回该控制模式字，其各位的定义见下表。

位	定义
0-6	内部使用。
7	运动出错停止标志位。GT_AuStpOn() 和 GT_AuStpOff()命令可修改该标志位。标志位为 1 表示电机运动出错后，自动关闭电机伺服使能，电机运动停止。
8-9	保留
10	自动刷新标志位。GT_AuUpdtOn()，GT_AuUpdtOff()命令可修改该标志位。标志位为 1，表示控制器在断点条件满足后自动刷新控制轴参数。
11-13	控制轴运动模式标志。编码如下: Bit13 Bit12 Bit11 运动模式 0 0 0 梯形曲线模式 0 0 1 速度控制模式 0 1 0 S-曲线模式 0 1 1 电子齿轮模式 1 0 1 坐标运动模式
14-15	内部使用

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

GT_GetMtnNm

函数原型：short GT_GetMtnNm (unsigned short * Lnum);

函数说明：当主机使用 GT_EStpMtn()和 GT_StpMtn()函数中断缓冲区命令执行状态后，调用本函数，返回中断处程序段编号

函数参数：*Lnum 返回中断处程序段编号。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

GT_GetMtrBias

函数原型：short GT_GetMtrBias(unsigned short * Bias);

函数说明：该函数获得当前轴由 GT_SetMtrBias()函数设置的静差补偿。

函数参数：*Bias 返回该静差补偿。

系　　统：DOS，WINDOWS

适用板卡：SV

相关函数：GT_SetMtrBias

GT_GetMtrCmd

函数原型：short GT_GetMtrCmd (short * Mcmd);

函数说明：在开环状态下，该函数获得当前轴由 GT_SetMtrCmd()函数设置的电机控制输出值。

函数参数：*Mcmd 返回该电机控制输出值。在当前轴处于闭环状态时，参数*Mcmd 返回值无意义。

系　　统：DOS，WINDOWS

适用板卡：SV

相关函数：GT_SetMtrCmd

GT_GetMtrLmt

函数原型：short GT_GetMtrLmt (unsigned short * Mlmt);

函数说明：该函数获得当前轴由 GT_SetMtrLmt()函数设置的输出饱和极限。

函数参数：*Mlmt 返回该输出饱和极限。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

相关函数：GT_SetMtrLmt

GT_GetPos

函数原型：short GT_GetPos(long * Pos);

函数说明：该函数获得当前轴由 GT_SetPos()函数设置的位置值。

函数参数：*Pos 返回设定的目标位置值。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

相关函数：GT_SetPos

GT_GetPosErr

函数原型：short GT_GetPosErr(unsigned short * Perr);

函数说明：该函数获得当前轴由 GT_SetPosErr()函数设置的位置误差极限。
函数参数：*Perr 返回设定的位置误差极限。
适用板卡：SV
系　　统：DOS，WINDOWS
相关函数：GT_SetPosErr

GT_GetPrfPnt

函数原型： short GT_GetPrfPnt(double * Pnt);
函数说明：该函数获得坐标系各轴的坐标位置值。
函数参数：双精度参数*Pnt 作为指针，指向一个长度为 4 的数组，按 X、Y、Z、A 轴的顺序返回各坐标轴规划的坐标位置值，也就是虚拟坐标系中各轴的运动位置值。

用户在调用该函数，应自行定义一个长度为 4 的双精度型数组，并将数组首地址作为该函数的实参。

系　　统：DOS，WINDOWS
适用板卡：所有 GT 系列卡
函数调用：举例——获得当前坐标系各轴的坐标位置值，并打印。

```
void main()
{
    short rtn;
    double crd_prf_pos[4];
    rtn=GT_GetPrfPnt(crd_prf_pos);      error(rtn);
    printf("the coordinate pos are: %f, %f, %f, %f \n", crd_prf_pos[0],
            crd_prf_pos[1], crd_prf_pos[2], crd_prf_pos[3]);
}
```

GT_GetRatio

函数原型： short GT_GetRatio(double * Ratio);
函数说明：该函数获得在电子齿轮工作模式下，由 GT_SetRatio()函数设置的电子齿轮减速比。
函数参数：*Ratio 返回设定的电子齿轮减速比。
系　　统：DOS，WINDOWS
适用板卡：所有 GT 系列卡
相关函数：GT_SetRatio

GT_GetSmplTm

函数原型：short GT_GetSmplTm(double * Timer);
函数说明：该函数获得由 GT_SetSmplTm()函数设置的伺服周期。
函数参数：*Timer 返回设定的伺服周期，范围：48(us)~1966.08(us)。
系　　统：DOS，WINDOWS
适用板卡：所有 GT 系列卡
相关函数：GT_SetSmplTm

GT_GetSts

函数原型： short GT_GetSts (unsigned short * Status);
函数说明： 该函数获得当前轴的状态字。
函数参数： *Status 返回该状态字，其定义见 GT_ClrSts()函数的*表 11-1*。
系　　统： DOS，WINDOWS
适用板卡： 所有 GT 系列卡
函数调用： 举例——检测第一轴状态，如果轴没有运动，则给出新的位置值，并刷新。

```
void main()
{
    short rtn;
    unsigned short status;
    rtn=GT_GetSts(&status);    error(rtn);
    if(status&0x400) return;
    rtn=GT_SetPos(10000);      error(rtn);
    rtn=GT_Update(status);     error(rtn);
 }
```

GT_GetVel

函数原型： short GT_GetVel (double * Vel);
函数说明： 该函数获得当前轴由 GT_SetVel()函数设置的速度值。
函数参数： *Vel 返回该速度值。
系　　统： DOS，WINDOWS
适用板卡： 所有 GT 系列卡
相关函数： GT_SetVel

GT_HardRst

函数原型： void GT_HardRst (void);
函数说明： 该函数复位运动控制器，调用该函数后主机将输出一个复位信号，因此 GT_HardRst()可以在任何情况下复位运动控制器，重新设置所有的控制寄存器的初始化状态。控制寄存器初始状态如下：

[1] 所有运动参数寄存器设置为零
[2] 所有位置捕获寄存器设置为零
[3] 清除全部事件状态位
[4] 屏蔽所有中断申请位（清零）
[5] 设置位置控制模式为梯形曲线控制方式
[6] 各滤波器参数设置为零值
[7] 四个轴积分限定为 32767
[8] 四个轴位置误差极限值定为 32767
[9] 所有断点比较值设置为零
[10] 禁止自动参数刷新方式
[11] 四个轴设置为闭环模式
[12] 禁止四个轴运动出错自动停止
[13] 四个轴输出电压饱和值为 32767

[14] 四个轴加速度限值（用于坐标系运动）为 16384
[15] 允许坐标系运动异常自动停止
[16] 坐标系运动处于立即命令输入执行状态
[17] 坐标系与四个电机控制轴之间没有任何映射关系

系　　统：DOS，WINDOWS
适用板卡：所有 GT 系列卡
相关函数：GT_Reset
函数调用：通常在运动控制器发生不可恢复的错误时，才调用该函数。

GT_HookIsr

函数原型：GT_ISR GT_HookIsr(GT_ISR gtisr)
函数说明：为 PCI 控制卡中断挂接指定中断服务程序（ISR：Interrupt Service Routine）。
函数参数：gtisr 为中断服务程序起始地址。
函数返回：返回值为原中断服务程序起始地址，必须保留该值为 GT_UnhookIsr 函数所用。
函数调用：参见 DOS 中断处理
适用板卡：PCI 总线卡
系　　统：DOS
相关函数：GT_UnhookIsr

GT_LmtSns

函数原型： short GT_LmtSns(unsigned short Sense);
函数说明：该函数设置限位开关有效电平。
函数参数：Sense 的意义见下表。表中对应位如果设置为零，表示限位开关输入信号为高时触发限位；反之，如果设置为“1”时，表示限位开关输入信号为低时触发限位。控制器上电默认为全 “0”, 即控制器默认限位开关状态为高电平时触发。在初始化运动控制器时，必须正确给出限位开关有效电平，否则不能保证控制器正常工作或者导致所有轴正负限位开关标志置位。

Bit	说明	定义
8-15	没有使用，设置为零值	
7	Axis #4 负向限位开关状态位	0：高电平触发 1：低电平触发
6	Axis #4 正向限位开关状态位	0：高电平触发 1：低电平触发
5	Axis #3 负向限位开关状态位	0：高电平触发 1：低电平触发
4	Axis #3 正向限位开关状态位	0：高电平触发 1：低电平触发
3	Axis #2 负向限位开关状态位	0：高电平触发 1：低电平触发
2	Axis #2 正向限位开关状态位	0：高电平触发 1：低电平触发
1	Axis #1 负向限位开关状态位	0：高电平触发 1：低电平触发
0	Axis #1 正向限位开关状态位	0：高电平触发 1：低电平触发

系　　统：DOS，WINDOWS
适用板卡：所有 GT 系列卡
函数调用：举例——设置第三轴正负限位开关有效电平反向，并清除原来因限位开关有效电平不符，造成限位开关触发的错误状态。

```
void main()
```

```
{
    short rtn;
    rtn=GT_LmtSns(0x30);      error(rtn);
    rtn=GT_Axis(3);           error(rtn);
    rtn=GT_Rststs(0xff9f);    error(rtn);
 }
```

GT_LmtsOff

函数原型：short GT_LmtsOff (void);

函数说明：该函数将禁止运动控制器监视当前轴的限位开关状态。但限位开关的状态仍然反映在控制器限位开关状态寄存器。用户可以通过调用 GT_GetLmtSwt()来查询限位开关的状态，只是该状态的变化不引起控制器的任何操作。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

相关函数：GT_LmtsOn

GT_LmtsOn

函数原型：short GT_LmtsOn(void);

函数说明：该函数使运动控制器开始监视当前轴的限位开关状态，如果中断申请允许，不管是正向或负向限位开关被压下，运动控制器都会向主机发出中断申请，*表 11-1* 所示的限位开关对应位就会被置“1”；如果不允许中断，则可采用查询的办法，调用 GT_GetLmtSwt()或 GT_GetSts()命令来查询限位开关的状态。运动控制器默认状态为调用 GT_LmtsOn()。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

相关函数：GT_LmtsOff

GT_LnXY

函数原型：short GT_LnXY(double X, double Y);

函数说明：该函数实现二维直线插补运动。

参数 X、Y 分别为相应各坐标的终点坐标。直线插补运动的起点坐标是前一段轨迹描述的终点坐标；如是第一条轨迹，则是当前位置坐标。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

相关函数：GT_LnXYZ，GT_LnXYZA

函数调用：举例——二维直线插补命令。

```
void main()
{
    short rtn;
    rtn=GT_SetSynVel(30);           error(rtn);
    rtn=GT_SetSynAcc(1);            error(rtn);
    rtn=GT_LnXY(12345,67890);       error(rtn);
  }
```

GT_LnXYZ

函数原型：short GT_LnXYZ(double X, double Y, double Z);

函数说明：该函数实现三维直线插补运动。参数 X、Y、Z 分别为相应各坐标的终点坐标。直线插补运动的起点坐标是前一段轨迹描述的终点坐标；如是第一条轨迹，则是当前位置坐标。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

相关函数：GT_LnXY，GT_LnXYZA

函数调用：举例——三维直线插补命令。

```
void main()
{
    short rtn;
    rtn=GT_StrtList();          error(rtn);
    rtn=GT_MvXYZ(0,0,0,6,0.3);  error(rtn);
    rtn=GT_LnXYZ(12945.2,58372.83,65473.121);   error(rtn);
    rtn=GT_StrtMtn();   error(rtn);
    ……
}
```

GT_LnXYZA

函数原型：short GT_LnXYZA(long X,long Y,long Z,long A);

函数说明：该函数实现四维直线插补运动。

参数 X、Y、Z、A 分别为相应各坐标的终点坐标。直线插补运动的起点坐标是前一段轨迹描述的终点坐标；如是第一条轨迹，则是当前位置坐标。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

相关函数：GT_LnXY，GT_LnXYZ

函数调用：举例——四维直线插补命令。

```
void main()
{
    short rtn;
    rtn=GT_StrtList();          error(rtn);
    rtn=GT_MvXYZA(0,0,0,0,6,0.3);   error(rtn);
    rtn=GT_LnXYZA(12945.2,58372.83,65473.121,0); error(rtn);
     ……
}
```

GT_MapAxis

函数原型：short GT_MapAxis(unsigned short Axes_Num, double * map_array);

函数说明：GT_MapAxis()函数用来映射坐标系，即建立电机轴与坐标系之间的关系，其中就包含当量的转换。

函数参数：Axis_Num 为控制轴号（1、2、3、4），坐标映射函数调用以后，该控制轴工作于坐标运动模式。该轴的位置记为 *Axis _ N*，单位是脉冲。参数*map_array 作为指针，指

向一个包括五个元素的数组，顺次记为 C_x、C_y、C_z、C_a、C，坐标轴 X、Y、Z、A 所对应的相应坐标记为 x、y、z、a。上述函数描述的映射关系能够简单地描述成下面的计算公式：

$$Axis\_N = C_x \times x + C_y \times y + C_z \times z + C_a \times a + C$$

由此可以看出被映射的控制轴的运动可以是坐标轴 X、Y、Z、A 的线性组合。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

函数调用：举例——某个系统电机一（第一轴）对应坐标系 X 轴，电机一每转一圈 X 方向移动 4mm；电机二（第二轴）对应坐标系 Y 轴，电机一每转一圈 Y 方向移动 5mm；而两台电机四倍频后均为 8000pulse/圈。如果以 mm 为单位，则有以下方程：

$$Axis\_1 = 2000 \times x + 0 \times y + 0 \times z + 0 \times a + 0$$

$$Axis\_2 = 0 \times x + 1600 \times y + 0 \times z + 0 \times a + 0$$

```
void main()
{
    short rtn;
    double map_cnt0[5]={2000,0,0,0,0};
    double map_cnt1[5]={0,1600,0,0,0};
    rtn=GT_MapAxis(1,map_cnt0); error(rtn);
    rtn= GT_MapAxis(2,map_cnt1);error(rtn);
 }
```

GT_MltiUpdt

函数原型：short GT_MltiUpdt(unsigned short Mask);

函数说明：使多个轴设置的缓冲区命令和参数立即生效。与之不同，GT_Update()函数只使当前轴的设置参数和命令生效。

函数参数：Mask 各位的意义参见下表。如果不想让某个轴设置的参数立即生效时，可将相应位置“0”，反之置“1"。

Bit	定义
0	1 号轴
1	2 号轴
2	3 号轴
3	4 号轴
4-15	没有使用

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

相关函数：GT_Update

函数调用：举例——同时刷新第一和第三轴的参数 Kp。

```
void main()
{
    short rtn;
    rtn=GT_Axis(1);              error(rtn);
```

```
    rtn=GT_SetKp(10);          error(rtn);
    rtn=GT_Axis(3);            error(rtn);
    rtn=GT_SetKp(15);          error(rtn);
    rtn=GT_MltiUpdt(0x5);      error(rtn);
}
```

GT_MtnBrk

函数原型：short GT_MtnBrk(void);

函数说明：该函数设定当前轴的断点模式为运动完成模式。该函数调用后，控制器在轴状态寄存器运动完成标志位置 1 时触发断点，使轴状态寄存器的断点标志置 1 并清除该断点模式，即除非主机重新设置断点，控制器将不再触发该断点。同时，在轴模式状态寄存器的自动刷新标志置 1 的条件下，控制器将刷新轴的所有双缓冲结构参数和命令。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

函数调用：举例——设置第四轴的断点模式为运动完成触发断点模式，并在运动完成时自动刷新新的目标位置，从而引起反向运动。

```
void main()
{
    short rtn;
    rtn=GT_Axis(1);          error(rtn);
    rtn=GT_AuUpdtOn();       error(rtn);
    rtn=GT_MtnBrk();         error(rtn);
    rtn=GT_PrflT();          error(rtn);
    rtn=GT_ClrSts();         error(rtn);
    rtn=GT_SetPos(10000);    error(rtn);
    rtn=GT_SetVel(7);        error(rtn);
    rtn=GT_SetAcc(0.347);    error(rtn);
    rtn=GT_Update();         error(rtn);
    rtn=GT_SetPos(0);        error(rtn);
    }
```

GT_MvXY

函数原型：short GT_MvXY(double X,double Y,double Vel,double Accel);

函数说明：对于二维坐标系，该函数必须作为函数 GT_StrtList()后的第一条缓冲区命令，放在命令缓冲区中用来定义二维坐标系缓冲区命令的起点位置。

参数 X、Y 是起点坐标；参数 Vel 设定合成速度，单位是**坐标系长度单位/控制周期**；参数 Accel 设定合成加速度，单位是**坐标系长度单位/控制周期** ^{2}。此后调用 GT_StrtMtn()函数时，运动控制器控制坐标系各轴从当前位置以直线插补的方式运动到相应起点位置，然后顺序执行缓冲区命令。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

相关函数：GT_MvXYZ，GT_MvXYZA

函数调用：该函数只能在缓冲区打开状态下调用。举例——在缓冲区打开状态下，将缓冲区运动的起点定位在 X 轴 1000（坐标系长度单位单位），Y 轴 2000（坐标系长度单位单位）

的位置，并以速度为 0.25（坐标系长度单位单位/插补周期），加速度为 0.01（坐标系长度单位单位/插补周期 ^{2}）移动到定位点。

```
void main()
{
    short rtn;
    rtn=GT_StrtList();          error(rtn);
    rtn=GT_MvXY (1000,2000,0.25,0.01);    error(rtn);
  }
```

GT_MvXYZ

函数原型：short GT_MvXYZ (double X, double Y, double Z, double vel, double accel);

函数说明：对于三维坐标系，该函数必须作为函数 GT_StrtList ()后的第一条缓冲区命令，放在命令缓冲区中用来定义三维坐标系缓冲区命令的起点位置。

参数 X、Y、Z 是起点坐标；参数 Vel 设定合成速度，单位是**坐标系长度单位/控制周期**；参数 Accel 设定合成加速度，单位是**坐标系长度单位/控制周期 ^{2}**。此后调用 GT_StrtMtn()函数时，运动控制器控制坐标系各轴从当前位置以直线插补的方式运动到相应起点位置，然后顺序执行缓冲区命令。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

相关函数：GT_MvXY，GT_MvXYZA

GT_MvXYZA

函数原型：short GT_MvXYZA(double X, double Y, double Z, double A, double vel, double accel);

函数说明：对于四维坐标系，该函数必须作为函数 GT_StrtList()后的第一条缓冲区命令，放在命令缓冲区中用来定义四维坐标系缓冲区命令的起点位置。

参数 X、Y、Z、A 是起点坐标；参数 Vel 设定合成速度，单位是**坐标系长度单位/控制周期**；参数 Accel 设定合成加速度，单位是**坐标系长度单位/控制周期 ^{2}**。此后调用 GT_StrtMtn()函数时，运动控制器控制坐标系各轴从当前位置以直线插补的方式运动到相应起点位置，然后顺序执行缓冲区命令。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

相关函数：GT_MvXY，GT_MvXYZ

GT_NegBrk

函数原型：short GT_NegBrk(void);

函数说明：该函数设置当前轴的断点模式为小于目标位置断点。该函数调用前，先由函数 GT_SetBrkCn()设置断点位置，并调用 GT_Update()或 GT_MltiUpdt()使之有效。此后调用该函数，控制器将在每个伺服周期比较该轴的实际位置与断点位置。当轴的实际位置小于或等于断点位置时，轴状态寄存器的断点标识置 1 并清除该断点模式，即除非主机重新设置断点，控制器将不再触发该断点。同时，在轴模式寄存器的自动刷新标志置 1 的条件下，控制器将刷新轴的所有双缓冲结构参数和命令。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡
相关函数：GT_PosBrk

GT_OpenLp

函数原型：short GT_OpenLp(void);
函数说明：该函数设置当前轴为伺服开环控制状态。该函数主要在直接对电机控制时使用。一般情况下，应使伺服驱动器工作在闭环控制状态。
系　　统：DOS，WINDOWS
适用板卡：SV
相关函数：GT_CloseLp

GT_Open

函数原型：short GT_Open(unsigned long PortBase, unsigned long irq);
函数说明：打开运动控制器设备，用户程序开始时必须调用此函数。
函数参数：PortBase为运动控制器基地址，参数irq为运动控制器中断号（*如果主机上10～15号中断都已经占满，或者用户不需要使用中断时，可设参数irq的值为0，以表示无中断*）。选择运动控制器基址和中断号时注意不要与其它设备冲突，否则该函数运行失败。
系　　统：WINDOWS
适用板卡：ISA 总线卡
相关函数：GT_Close

GT_Open

函数原型：short GT_Open();
函数说明：打开运动控制器设备，用户程序开始时必须调用此函数。
系　　统：DOS，WINDOWS
适用板卡：PCI 总线卡
相关函数：GT_Close

GT_PosBrk

函数原型：short GT_PosBrk(void);
函数说明：该函数设置当前轴的断点模式为正向目标位置断点。该函数调用前，先由函数 GT_SetBrkCn()设置断点位置，并调用 GT_Update()或 GT_MltiUpdt()使之有效。此后调用该函数，控制器将在每个伺服周期比较该轴的实际位置与断点位置。当轴的实际位置大于或等于断点位置时，轴状态寄存器的断点标志置 1 并清除该断点模式，即主机需要重新设置断点，否则控制器将不再触发该断点。同时，在轴模式寄存器的自动刷新标志置 1 的条件下，控制器将刷新轴的所有双缓冲结构参数和命令。
系　　统：DOS，WINDOWS
适用板卡：所有 GT 系列卡
相关函数：GT_NegBrk

GT_PrflG

函数原型：short GT_PrflG(unsigned short Master);

函数说明：该函数设置当前轴的运动控制模式为电子齿轮模式，并指定跟随的主轴轴号。

参数 Master 可取 1、2、3、4，表示 1 到 4 轴为当前轴的主轴。也可取 5、6，表示指定两个辅助编码器中的一个作为主动轴（辅助编码器为可选功能）。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

相关函数：GT_PrflT，GT_PrflS，GT_PrflV

函数调用：当前轴没有运动时，可以调用该函数。举例——设置第一轴按照-1.5 的比例跟随第四轴的梯形曲线运动，而第三轴又按照 0.5 的比例跟随第一轴的运动。

```
void main()
{
    short rtn;
    rtn=GT_Axis(4);         error(rtn);
    rtn=GT_PrflT();         error(rtn);
    rtn=GT_Axis(1);         error(rtn);
    rtn=GT_PrflG(4);        error(rtn);
    rtn=GT_SetRatio(-1.5); error(rtn);
    rtn=GT_Axis(3);         error(rtn);
    rtn=GT_PrflG(1);        error(rtn);
    rtn=GT_SetRatio(0.5); error(rtn);
    rtn=GT_MltiUpdt(0x5); error(rtn);
 }
```

GT_PrflS

函数原型：short GT_PrflS(void);

函数说明：该函数设置当前轴的运动控制模式为 S-曲线模式。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

相关函数：GT_PrflT，GT_PrflG，GT_PrflV

函数调用：当前轴没有运动时，可以调用该函数。设置 S-曲线控制模式的过程如下：

```
void main()
{
    short rtn;
    rtn=GT_Axis(1);             error(rtn);
    rtn=GT_PrflS();             error(rtn);
    rtn=GT_SetPos(12345);       error(rtn);
    rtn=GT_SetVel(3.21);        error(rtn);
    rtn=GT_SetMAcc(0.345);      error(rtn);
    rtn=GT_Jerk(0.087);         error(rtn);
    rtn=GT_Update();                error(rtn);
    }
```

GT_PrflT

函数原型：short GT_PrflT(void);

函数说明：该函数设置当前轴的运动控制模式为梯形曲线模式。

系　　统：DOS，WINDOWS
适用板卡：所有 GT 系列卡
相关函数：GT_PrflS，GT_PrflG，GT_PrflV
函数调用：当前轴没有运动时，可以调用该函数。设置 T-曲线控制模式的过程如下：

```
void main()
{
    short rtn;
    rtn=GT_Axis(2);             error(rtn);
    rtn=GT_PrflT();             error(rtn);
    rtn=GT_SetPos(100000);      error(rtn);
    rtn=GT_SetVel(5.7);         error(rtn);
    rtn=GT_SetAcc(0.67);        error(rtn);
    rtn=GT_Update();                error(rtn);
 }
```

GT_PrflV

函数原型：short GT_PrflV(void);
函数说明：该函数设置当前轴的运动控制模式为速度控制模式。
系　　统：DOS，WINDOWS
适用板卡：所有 GT 系列卡
相关函数：GT_PrflS，GT_PrflG，GT_PrflT
函数调用：当前轴没有运动时，可以调用该函数。设置速度控制模式的过程如下：

```
void main()
{
    short rtn;
    rtn=GT_Axis(3);         error(rtn);
    rtn=GT_PrflV();         error(rtn);
    rtn=GT_SetVel(17);      error(rtn);
    rtn=GT_SetAcc(1.1);     error(rtn);
    rtn=GT_Update();            error(rtn);
    }
```

GT_Reset

函数原型：short GT_Reset (void)
函数说明：该函数是主机以命令的形式使运动控制器复位，其结果同 GT_HardRst()函数。
注意：调用该函数时，如果有电机轴处于控制状态（即使用 GT_AxisOn()将控制轴打开），该电机轴可能会有动作，这是由于复位指令在复位运动控制器的同时，也会使运动控制器的所有外部输出置于初始状态（包括关闭电机驱动使能），但由于电机本身对该关闭信号的滞后响应，使得运动控制器已经关闭控制时，电机还处于接收伺服控制命令的状态，从而使电机产生动作。所以在调用该函数之前，应使用 GT_AxisOff()指令将所有电机轴的控制状态关闭。
系　　统：DOS，WINDOWS
适用板卡：所有 GT 系列卡
相关函数：GT_HardRst

GT_RstIntr

函数原型： short GT_RstIntr(unsigned int Mask);

函数说明： 该函数清除当前轴的中断标志位。当控制器向主机申请中断后，主机应在中断服务程序结束时执行 GT_RstIntr ()函数清除控制器的中断标志，使运动控制器可以再次向主机申请中断。

函数参数： 屏蔽字 Mask 的定义见下表，当屏蔽字某一位设置为“1”时，允许该位标识的中断事件保留，如果设置为“0"时，则清除该中断标识的中断事件。

Bit	定义
7-15	没有使用,可设置为 0
6	正向限位开关
5	负向限位开关
4	运动出错
3	Index/Home 捕捉到
2	断点条件满足
1	轴驱动器报警
0	轴运动完成

系　　统： DOS

适用板卡： 所有 GT 系列卡

GT_RstSts

函数原型： short GT_RstSts(unsigned short Mask);

函数说明： 该函数根据屏蔽字 Mask 的定义清除当前轴的状态寄存器。

函数参数： 屏蔽字 Mask 的定义同 GT_ ()函数的*表 2-2* 中 Bit0~Bit7 的定义一致，当屏蔽字某一位设置为“1”时，允许该标志位保留，如果设置为“0”时，则清除该事件标志。

系　　统： DOS，WINDOWS

适用板卡： 所有 GT 系列卡

相关函数： GT_ ClrSts

GT_SetAcc

函数原型： short GT_SetAcc(double Acc);

函数说明： 该函数设置当前轴的加速度参数。该函数只在梯形速度模式和速度控制模式下有效。

函数参数： Acc 是所要设置的加速度值，其取值范围为：0～16384。加速度值的单位为**脉冲数/控制周期** ^{2}。需调用 GT_Update()或 GT_MltiUpdt()函数后，才能使新设置的参数有效。

系　　统： DOS，WINDOWS

适用板卡： 所有 GT 系列卡

相关函数： GT_GetAcc

GT_SetAccLmt

函数原型： short GT_SetAccLmt(double Acclmt);

函数说明： 该函数设置当前轴的允许加速度极限，从而影响缓冲区命令执行状态时轨迹

段转接处的轨迹速度规划策略。

函数参数：Acclmt 是所要设置的加速度极限，其取值范围为：0～16384。本函数所设的加速度极限值只在当前轴处于坐标系运动模式下缓冲区命令执行过程中起作用。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

相关函数：GT_GetAccLmt

GT_SetAddr

函数原型：short GT_SetAddr(unsigned short Address);

函数说明：该函数设置主机与运动控制器之间通讯的基地址。

函数参数：Address 是所要设置的基地址值，其取值范围为运动控制器允许的基地址设置范围。

系　　统：DOS

适用板卡：所有 GT 系列卡

相关函数：GT_GetAddr

GT_SetAtlPos

函数原型：short GT_SetAtlPos(long actl_pos);

函数说明：调用该函数将当前轴的实际位置和目标位置以及控制当前控制周期的规划位置修改为设定值。本函数只在当前轴运动停止时有效，否则将被视为非法命令产生命令出错标识。本函数在当前轴处于电子齿轮运动模式时使用无效。

函数参数：actl_pos 表示要设置的实际位置值。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

相关函数：GT_GetAtlPos，GT_SynchPos，GT_ZeroPos

函数调用：举例——当第四轴没有运动时，设置它的实际位置为 1000。

```
void main()
{
    short rtn;
    unsigned short status;
    rtn=GT_Axis(4);                error(rtn);
    rtn=GT_GetSts(&status);        error(rtn);
    if(status&0x400) return;
    rtn=GT_SetAtlPos(1000);        error(rtn);
    }
```

GT_SetBgCommandSet

函数原型：short WINAPI GT_SetBgCommandSet(PBGCOMMANDSET pBgCmdset, ULONG CmdsetSize);

函数说明：为运动控制卡设置中断时后台执行命令集，本函数会覆盖前次设置值。可以一次为多个中断设置后台命令，也可以为同一中断设置多条后台命令。

函数参数：pBgCmdset 为需要控制卡执行的命令集起始地址；CmdsetSize 为后台命令集所占内存空间大小，字节为单位。

函数返回值：0 表示成功，-1 表示失败。

运动控制器驱动程序允许用户为运动控制器指定它中断产生后要执行的 GT 命令集，由于这些命令是在控制器产生中断时自动执行，不直接与用户交互，所以我们称之为后台命令。

命令集结构如下：

```
typedef struct _BACKGROUND_COMMANDSET
{
USHORT    Count;                    //命令集的子项数目，即所要服务的中断类型数量
BACKGROUND_COMMAND BackgroundCommand[1];//某一指定中断的后台命令组
}BGCOMMANDSET,*PBGCOMMANDSET;

typedef struct _BACKGROUND_COMMAND
{
  USHORT              InterruptMask;//本命令数组服务的具体中断
  USHORT              CommandCount; //本命令组的命令数量
  GENERAL_COMMAND     GenCommand[1];   //命令数组
}BACKGROUND_COMMAND,*PBACKGROUND_COMMAND;

typedef struct _GENERAL_COMMAND //单个命令的结构
{
  USHORT      usCommand; //命令字
  USHORT      InputLength; //命令需要输入的数据长度，字为单位
  USHORT      OutputLength;//命令将返回的数据长度，字为单位
  USHORT      usResult; //命令执行结果

  union
  {
      USHORT      sData[2];
      ULONG       lData;
  }in; //命令输入数据

  union
  {
      USHORT      sData[2];
      ULONG       lData;
  }out; //命令返回数据

}GENERAL_COMMAND,*PGENERAL_COMMAND;
```

系　　统：WINDOWS

适用板卡：PCI 总线卡

相关函数：GT_SetIntSyncEvent

函数调用：参见 Windows 下中断处理

GT_SetBrkCn

函数原型： short GT_SetBrkCn(long Brk);

函数说明： 设置当前轴的断点位置比较值。

在调用 GT_PosBrk( )和 GT_NegBrk( )函数时，由于这两条命令会立即生效，因此必须在此前调用 GT_SetBrkCn( )函数，并调用 GT_Update()或 GT_MltiUpdt()函数使新设置的断点位置有效。

函数参数： Brk 表示要设置的当前轴的断点位置。其取值范围为–1073741824～1073741823。需调用 GT_Update()或 GT_MltiUpdt()函数后，才能使新设置的参数有效。

系　　统： DOS，WINDOWS

适用板卡： 所有 GT 系列卡

相关函数： GT_GetBrkCn

GT_SetILmt

函数原型： short GT_SetILmt(unsigned short　Ilm);

函数说明： 该函数设置当前轴伺服滤波器的误差积分限。

函数参数： Ilm 是需要设置的误差积分限，其取值范围为 0～32767。需调用 GT_Update()或 GT_MltiUpdt()函数后，才能使新设置的参数有效。

系　　统： DOS，WINDOWS

适用板卡： SV

相关函数： GT_GetILmt

GT_SetIntrMsk

函数原型： short GT_SetIntrMsk(unsigned short Mask);

函数说明： 该函数设置中断屏蔽字。

函数参数： Mask，其各位的意义参考 GT_RstIntr()函数的下表。当中断屏蔽字中某一位设置为“1”时，允许该位表示的中断事件向主机申请中断，如果设置为“0"时，不允许该位表示的中断事件向主机申请中断。

Bit	定义
7-15	没有使用,可设置为 0
6	正向限位开关
5	负向限位开关
4	运动出错
3	Index/Home 捕捉到
2	断点条件满足
1	轴驱动器报警
0	轴运动完成

系　　统： DOS，WINDOWS

适用板卡： 所有 GT 系列卡

相关函数： GT_GetIntrMsk

GT_SetIntrTm

函数原型： void GT_SetIntrTm(short Timer);

函数说明： 该函数设置控制器定时中断的时间常数。控制器定时中断时间由控制器的控制周期和该函数设置值 Timer 共同确定。

函数参数： Timer 为控制周期的倍数，范围为：0～32767。

例如：控制器的控制周期为 200 微秒，主机用 GT_SetIntrTm()函数设定的 Timer 值为 10，则定时中断的周期=10*200 微秒，即 2 毫秒。

系　　统： DOS，WINDOWS

适用板卡： 所有 GT 系列卡

相关函数： GT_GetIntrTm

GT_SetIntSyncEvent

函数原型： short WINAPI GT_SetIntSyncEvent(HANDLE hEvent);

函数说明： 为 PCI 控制卡设置中断同步事件，本函数会覆盖前次设置值。

函数参数： hEvent 为用于同步的事件名柄。当该值为 NULL 时，函数仅复位以前设置值。

该函数既可以处理运动控制器产生的时间中断，又可以处理运动控制器产生的事件中断。用户可以通过调用 GT_GetIntr(unsigned * Status)来判断当次中断为何种中断。Status 为 0：时间中断；非 0：具体含义请参考下表：

位	定义
15	设定 Index 信号捕获使能 （1 表示使能）
14	设定原点开关捕获使能 （1 表示使能）
12-13	Bit13 Bit12 中断产生轴 0 0 1 0 1 2 1 0 3 1 1 4
11	轴限位开关使能 （1 表示使能）
10	运动状态标志位 （1 表示在运动）
9	控制轴使能/关闭 (1 表示使能)
8	开环/闭环 (1 表示闭环)
7	主机命令出错 (1 表示出错)
6	负向限位开关动作 （1 表示动作）
5	正向限位开关动作 （1 表示动作）
4	运动出错 （1 表示出错）
3	Index/Home 位置捕捉到位 （1 表示到位）
2	断点到 （1 表示到位）
1	伺服驱动器报警 （1 表示报警）
0	运动完成标志位 （1 表示完成）

PCI 总线运动控制驱动程序可采用事件同步机制与用户程序同步，当控制卡产生中断时将设置 hEvent 代表的事件对象为置位状态。具体事件同步机制请参看相关进程同步机制开发

文档。

函数返回值：0 表示成功，-1 表示失败。

系　　统：WINDOWS

适用板卡：PCI 总线卡

相关函数：GT_SetBgCommandSet

函数调用：参见 Windows 下中断处理

GT_SetJerk

函数原型：short GT_SetJerk(double Jerk);

函数说明：该函数用于在 S-曲线模式下设定当前轴的加加速度。加加速度值的单位为**脉冲/控制周期**^{3}。

函数参数：Jerk 是所要设置的加加速度，其取值范围为：0～0.5。需调用 GT_Update()或 GT_MltiUpdt()函数后，才能使新设置的参数有效。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

相关函数：GT_GetJerk

GT_SetKaff

函数原型：short GT_SetKaff(unsigned short　Kaff);

函数说明：该函数设置当前轴伺服滤波器的加速度前馈增益。

函数参数：Kaff 是需要设置的加速度前馈增益，其取值范围为 0～32767。需调用 GT_Update()或 GT_MltiUpdt()函数后，才能使新设置的参数有效。

系　　统：DOS，WINDOWS

适用板卡：SV

相关函数：GT_GetKaff

GT_SetKd

函数原型：short GT_SetKd(unsigned short　Kd);

函数说明：该函数设置当前轴伺服滤波器的微分增益。

函数参数：Kd 是需要设置的微分增益，其取值范围为 0～32767。需调用 GT_Update()或 GT_MltiUpdt()函数后，才能使新设置的参数有效。

系　　统：DOS，WINDOWS

适用板卡：SV

相关函数：GT_GetKd

GT_SetKi

函数原型：short GT_SetKi(unsigned short　Ki);

函数说明：该函数设置当前轴伺服滤波器的积分增益。

函数参数：Ki 是需要设置的积分增益，其取值范围为 0～32767。需调用 GT_Update()或 GT_MltiUpdt()函数后，才能使新设置的参数有效。

系　　统：DOS，WINDOWS

适用板卡：SV

相关函数：GT_GetKi

GT_SetKp

函数原型：short GT_SetKp(unsigned short Kp);

函数说明：该函数设置当前轴伺服滤波器比例增益。

函数参数：Kp 是需要设置的比例增益，其取值范围为 0～32767。需调用 GT_Update()或 GT_MltiUpdt()函数后，才能使新设置的参数有效。

系　　统：DOS，WINDOWS

适用板卡：SV

相关函数：GT_GetKp

GT_SetKvff

函数原型：short GT_SetKvff (unsigned short Kvff);

函数说明：该函数设置当前轴伺服滤波器速度前馈增益。

函数参数：Kvff是需要设置的速度前馈增益，其取值范围为0～32767。需调用GT_Update()或 GT_MltiUpdt()函数后，才能使新设置的参数有效。

系　　统：DOS，WINDOWS

适用板卡：SV

相关函数：GT_GetKvff

GT_SetMAcc

函数原型：short GT_SetMAcc(double Macc);

函数说明：当控制模式为 S-曲线时，调用该函数设置当前轴最大加速度值。

函数参数：Macc 是所要设置的最大加速度值，其取值范围 0～0.5（不含 0.5），单位为**脉冲/控制周期**^{2}。需调用 GT_Update()或 GT_MltiUpdt()函数后，才能使新设置的参数有效。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

相关函数：GT_GetMAcc

GT_SetMtrBias

函数原型：short GT_SetMtrBias(short Bias);

函数说明：该函数设置当前轴伺服滤波器控制输出的静差补偿值。

函数参数：Bias 是需要设置的静差补偿值，其取值范围为-32768～32767。需调用 GT_Update()或 GT_MltiUpdt()函数后，才能使新设置的参数有效。该参数只对闭环控制有效，控制器默认值为 0。

系　　统：DOS，WINDOWS

适用板卡：SV

相关函数：GT_GetMtrBias

GT_SetMtrCmd

函数原型：short GT_SetMtrCmd(short Mcmd);

函数说明：该函数设置开环控制状态下的电压输出控制值。

函数参数：Mcmd 是需要设置的电机输出控制值，其取值范围为-32767～32767。这里-32767 表示最大的负电压值；32767 表示最大的正电压值。

系　　统：DOS，WINDOWS
适用板卡：SV
相关函数：GT_GetMtrCmd

GT_SetMtrLmt

函数原型：short GT_SetMtrLmt(unsigned short Mlmt)；
函数说明：该函数设置当前轴控制输出的饱和极限。
函数参数：Mlmt 是需要设置的饱和极限，其取值范围为 0～32767。需调用 GT_Update()或 GT_MltiUpdt()函数后，才能使新设置的参数有效。该参数只对闭环控制有效，控制器默认值为：32767。
系　　统：DOS，WINDOWS
适用板卡：SV
相关函数：GT_GetMtrLmt

GT_SetPos

函数原型：short GT_SetPos (long Pos)；
函数说明：该函数设置在 S-曲线模式和梯形曲线模式下当前轴的目标位置。
函数参数：Pos 是需要设置的目标位置值，其取值范围为 –1073741824～1073741823。该函数设置的参数，需调用函数 GT_Update()或 GT_MltiUpdt()后才有效。位置的单位为**脉冲数**。
系　　统：DOS，WINDOWS
适用板卡：所有 GT 系列卡
相关函数：GT_GetPos

GT_SetPosErr

函数原型：short GT_SetPosErr(unsigned short Perr)；
函数说明：该函数设置当前轴位置误差极限。
函数参数：Perr 是需要设置的位置误差极限，其取值范围为 0～32767。需调用 GT_Update()或 GT_MltiUpdt()函数后，才能使新设置的参数有效。
系　　统：DOS，WINDOWS
适用板卡：SV
相关函数：GT_GetPosErr

GT_SetRatio

函数原型：short GT_SetRatio(double Ratio)；
函数说明：在电子齿轮模式下，该函数设置当前轴的电子齿轮比，使主动轴与被动轴之间按设定比例关系运动。
函数参数：Ratio 为所设置的电子齿轮比，其取值范围是-16384~16384。当 Ratio 为正时，主动轴与被动轴同向运动；当 Ratio 的值为负时，主动轴与被动轴的转向相反。需调用 GT_Update()或 GT_MltiUpdt()函数后，才能使新设置的参数有效。
系　　统：DOS，WINDOWS
适用板卡：所有 GT 系列卡
相关函数：GT_GetRatio

GT_SetSmplTm

函数原型：short GT_SetSmplTm(double Timer);

函数说明：该函数设置运动控制器伺服周期。

参数 Timer 为所设置的伺服周期，其单位为微秒，设定值的范围为：48～1966.08 微秒。运动控制器推荐使用的伺服周期是 200 微秒。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

相关函数：GT_GetSmplTm

GT_SetSynAcc

函数原型：short GT_SetSynAcc(double Accel);

函数说明：该函数设置坐标系运动中轨迹段的合成加速度。

函数参数：Accel 是设定的合成加速度值，是坐标系映射各轴分加速度的矢量和（均为正值）。其单位是**坐标系长度单位/控制周期**^{2}。该函数影响此后调用的所有直线插补和圆弧插补函数的加速度，直到再次调用此函数为止。缓冲区的该函数，只影响缓冲区命令的加速度。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

GT_SetSynVel

函数原型：short GT_SetSynVel(double Vel);

函数说明：该函数设置轨迹段的合成运动速度。

函数参数：Vel 是设定的合成速度值，是坐标系各坐标轴分速度的矢量和（为正值）。其单位是**坐标系长度单位/控制周期**。该函数影响此后调用的所有直线插补和圆弧插补函数的速度，直到再次调用此函数为止。缓冲区的该函数，只影响缓冲区中命令的速度。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

GT_TmrIntr

函数原型：short GT_TmrIntr (void);

函数说明：该函数设置运动控制器向主机申请的中断为定时中断。定时中断的周期由控制器的伺服刷新周期和 GT_SetIntrTm()函数的设定值共同确定。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

相关函数：GT_EvntIntr

函数调用：举例——设置运动控制器向主机申请的中断为时间中断。

```
void main()
{
    short rtn;
    rtn=GT_TmrIntr();     error(rtn);
    }
```

GT_SetVel

函数原型：short GT_SetVel(double Vel);

函数说明：该函数设置当前轴的目标速度参数。

函数参数：Vel 是需要设施的目标速度值。在梯形曲线模式和 S-曲线模式下速度取值范围为：0 到 16384。在速度控制模式下，速度取值范围为：-16384～16384。速度值单位是**脉冲数/控制周期** 。需调用 GT_Update()或 GT_MltiUpdt()函数后，才能使新设置的参数有效。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

相关函数：GT_SetVel

GT_SmthStp

函数原型：short　GT_SmthStp(void);

函数说明：该函数用于停止当前轴的运动，与 GT_AbptStp()不同之处在于 GT_SmthStp()将按照设定的加速度参数减速停止当前轴的运动，而不象 GT_AbptStp()那样立即停止轴的运动。本函数需与 GT_Update()或 GT_MltiUpdt()函数结合使用才能生效。GT_SmthStp()用于除电子齿轮模式外的三种面向控制轴的运动控制模式。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

相关函数：GT_AbptStp

GT_StepDir

函数原型：short GT_StepDir (void);

函数说明：如果当前轴的控制输出模式为脉冲输出模式，调用该函数可将脉冲输出的方式设置为“脉冲＋方向”。

当用户调用 GT_CtrlMode()函数将控制输出模式设置为脉冲输出时，运动控制器默认的脉冲输出方式为“脉冲＋方向”。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

相关函数：GT_SetPulse

GT_StepPulse

函数原型：short GT_StepPulse (void);

函数说明：如果当前轴的控制输出模式为脉冲输出模式，调用该函数可将脉冲输出的方式设置为“正负脉冲”方式。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

相关函数：GT_SetDir

GT_StpMtn

函数原型：short GT_StpMtn (void);

函数说明：该函数按照当前的合成速度减速策略停止基于坐标系的多轴协调运动命令。在缓冲区命令执行过程中，运动停止后将保存运动停止时的必要信息，以便此后调用

GT_StrtMtn()函数继续缓冲区命令执行。

用户不能在执行定位指令时发出中断命令，即运动到定位点的过程不能被打断，这时发出中断命令会引起运动出错。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

相关函数：GT_EStpMtn

GT_StrtList

函数原型：short GT_StrtList (void);

函数说明：该函数清空运动控制器命令缓冲区，开始缓冲区命令输入状态。已经进入缓冲区命令输入状态后，该函数无效。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

相关函数：GT_AddList，GT_EndList

GT_StrtMtn

函数原型：short GT_StrtMtn (void);

函数说明：该函数启动缓冲区命令执行。运动控制器顺序执行存储在命令缓冲区中的运动命令。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

GT_SwitchtoCardNo

函数原型：short GT_SwitchtoCardNo(short card_no);

函数说明：该函数用于切换当前卡。当一 PC 系统使用多个控制卡时，本函数用于指定当前控制卡。当函数执行成功后，后随的所有 GT 函数将只操作在当前卡上。

函数参数：card_no 是将被设为当前卡的卡号值。取值范围为 0--15。

函数返回值：0 表示成功，-1 表示失败。

在多控制卡系统中，每个控制卡在操作系统启动时被分配一个卡号（0-15），该卡号在系统重新启动之前一直有效，用于区别各个控制卡。卡号分配原则遵循 PNP 规则，第一个被系统识别的卡永远为 0 号卡，所以在硬件配置没有改变的情况下，系统每次分配的卡号是一致的。

系　　统：DOS、WINDOWS

适用板卡：PCI 总线卡

相关函数：GT_GetCurrentCardNo

函数调用：GT_SwitchtoCardNo(1)

GT_SynchPos

函数原型：short GT_SynchPos(void)；

函数说明：该函数将当前控制轴的目标位置寄存器和当前伺服周期的规划位置寄存器设置为实际位置寄存器的位置值。该函数适应于 S-曲线和梯形曲线控制模式，可用于在运动控制产生了错误又需重新启动时，令目前伺服位置控制寄存器的值等于实际位置寄存器的值。同时当前轴在进行伺服使能（禁止使能）或闭环（开环）状态切换时，也可以用该函数确保

电机平稳过渡。该函数需与 GT_Update()或 GT_MltiUpdt()函数结合使用才能生效。本函数在当前轴处于电子齿轮运动模式时使用无效。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

相关函数：GT_ZeroPos，GT_SetAtlPos

GT_UnhookIsr

函数原型：short GT_UnhookIsr(GT_ISR old_isr)

函数说明：释放由 GT_HookIsr 函数为 GT400 控制卡持接的 ISR，并恢复原 ISR。

函数参数：参数 old_isr 为系统原用中断服务程序起始地址，由 GT_HookIsr 返回。

函数返回：0 表示成功，-1 表示失败。

适用板卡：PCI 总线卡

系　　统：DOS

相关函数：GT_HookIsr

函数调用：参见 DOS 中断处理。

注意：GT_HookIsr、GT_UnhookIsr 必须成对使用，即在使用了 GT_HookIsr 后，整个程序结束前必须调用 GT_UnhookIsr 恢复系统原有 ISR，否则系统将崩溃。

GT_Update

函数原型：short GT_Update(void);

函数说明：运动控制器的一些参数设置函数和一些命令函数，采用双缓冲寄存器方式工作。调用 GT_Update()函数后，针对当前轴的双缓冲参数和命令才能生效。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

相关函数：GT_MltiUpdt

GT_ZeroPos

函数原型：short GT_ZeroPos (void);

函数说明：调用该函数将当前控制轴的实际位置寄存器和目标位置以及当前伺服周期的规划位置寄存器设为零值。本函数只在当前轴运动停止有效，否则将被视为非法命令产生命令出错标识。本函数在当前轴处于电子齿轮运动模式时使用无效。

系　　统：DOS，WINDOWS

适用板卡：所有 GT 系列卡

相关函数：GT_SynchPos，GT_SetAtlPos