固高科技
GOOGOL TECHNOLOGY

GXN 系列运动控制器编程手册

基本功能

2022 年 04 月

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。

联系我们

固高科技股份有限公司

地址：深圳市高新技术产业园南区深港产学研基地西座二楼 W211 室

电话：0755-26970817　26737236　26970824

传真：0755-26970821

电子邮件：googol@googoltech.com

网址：http://www.googoltech.com.cn

固高科技（海外）有限公司

地址：香港九龍觀塘偉業街 108 號絲寶國際大廈 10 樓 1008-09 室

電話：+(852) 2358-1033

傳真：+(852) 2719-8399

電子郵件：sales@googoltech.com

info@googoltech.com

網址：http://www.googoltech.com

臺灣固高科技股份有限公司

地址：台中市西屯區福中二路 10 巷 22 號 2 樓

電話：+886-4-23588245

傳真：+886-4-23586495

電子郵件：twinfo@googoltech.com

文档版本

版本号	修订日期
1.0	2017 年 10 月 17 日
1.1	2017 年 12 月 14 日
1.2	2018 年 07 月 02 日
1.3	2019 年 05 月 22 日
1.4	2020 年 01 月 11 日
1.5	2020 年 04 月 08 日
1.6	2020 年 10 月 15 日
1.7	2020 年 12 月 31 日
1.8	2021 年 08 月 09 日

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。

固高产品的更多信息

固高科技的网址是 http://www.googoltech.com.cn。在我们的网页上可以得到更多关于公司和产品的信息，包括：公司简介、产品介绍、技术支持、产品最新发布等等。

您也可以通过电话（0755－26970817）咨询关于公司和产品的更多信息。

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：support@googoltech.com；
电话：0755－26970843
发函至：深圳市高新技术产业园南区园深港产学研基地西座二楼 W211 室
固高科技股份有限公司
邮编：518057

编程手册的用途

用户通过阅读本手册，能够了解运动控制器的基本控制功能，掌握函数的用法，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，编制用户应用程序，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语言编程基础或Windows环境下使用动态链接库的基础，同时具有一定运动控制工作经验，对伺服或步进控制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

本手册由十四章内容组成，详细介绍了运动控制器的基本控制功能及编程实现。

相关文件

关于控制器的调试和安装，请参见随产品配套的《GXN 系列运动控制器用户手册》。
关于网络型模块的硬件接口，请参见随产品配套的《GNM 系列端子板模块用户手册》。
关于更复杂的控制器功能，请参见随产品配套的《GXN 系列运动控制器编程手册之高级功能》
关于扩展模块的使用，请参见随产品配套的《GXN 扩展功能-扩展模块功能编程手册》和《gLink200 系列模块（500 协议）用户手册》。

产品相关手册及安装文件如驱动程序、dll 文件、例程、Demo 等，请登录固高科技公司网站下载，网址为：www.googoltech.com.cn/pro_view-53.html

目录

第1章 指令列表

提示	本手册中所有字体为蓝色的指令（如 GTN_PrfTrap）均带有超级链接，点击可跳转至指令说明。

表 1-1 指令汇总列表

GTN_GetCrdMPGMode	获取手轮引导参数	190

第2章 运动控制器函数库的使用

产品相关手册及安装文件如驱动程序、dll 文件、例程、Demo 等，请登录固高科技公司网站下载，网址为：http://www.googoltech.com.cn/product-9.html

2.1 Windows 系统下动态链接库的使用

在 Windows 系统下使用运动控制器，首先要安装驱动程序；运动控制器的动态链接库文件名为 gts.dll；网络初始化的动态链接库文件名为 gt_rn.dll。

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

2.1.1 头文件说明

运动控制器动态链接库中包括 gts.h，gts.h 作为基本功能必须包含。

2.1.2 Visual C++ 6.0 中的使用

(1) 启动Visual C++ 6.0，新建一个工程；

(2) 将VC环境的动态链接库(gts.dll和gt_rn.dll)、头文件(例如：gts.h)和lib文件(gts.lib)复制到工程文件夹中；

(3) 选择“Project”菜单下的“Settings…”菜单项；

(4) 切换到“Link”标签页，在“Object\library modules”栏中输入lib文件名，例如gts.lib；

(5) 在应用程序文件中加入函数库头文件的声明，例如：#include “gts.h”；

至此，用户就可以在Visual C++中调用函数库中的任何函数，开始编写应用程序。

2.1.3 Visual Basic 6.0 中的使用

(1) 启动Visual Basic，新建一个工程；

(2) 将VB6环境的动态链接库和函数声明文件复制到工程文件夹中；

(3) 选择“工程”菜单下的“添加模块”菜单项；

(4) 切换到“现存”标签页，选择函数声明文件，例如gts.bas，将其添加到工程当中；

至此，用户就可以在Visual Basic中调用函数库中的任何函数，开始编写应用程序。

2.1.4 Delphi 中的使用

(1) 启动Delphi，新建一个工程；

(2) 将Delphi环境的动态链接库和函数声明文件复制到工程文件夹中；

(3) 选择“Project”菜单下的“Add to Project…”菜单项；

(4) 将函数声明文件添加到工程当中；

(5) 在代码编辑窗口中，切换到用户的单元文件；

(6) 选择“File”菜单下的“Use Unit…”菜单项，添加对函数声明文件的引用；

至此，用户就可以在 Delphi 中调用函数库中的任何函数，开始编写应用程序。

2.1.5 Visual Basic 2008 中的使用

(1) 启动Visual Studio 2008，按照“File”->"New"，选择建立VB工程；

(2) 将VB2008环境的动态链接库和函数声明文件复制到工程文件夹中，注意：gts.dll应复制到"..\bin"文件夹中的debug或者release文件夹中；

(3) 选择“project”菜单下的“Add existing Item”菜单项，选择函数声明文件，如gts.vb，将其添加到工程当中；

至此，用户就可以在Visual Studio 2008中使用VB2008模块调用函数库中的任何函数，开始编写应用程序。

2.1.6 Visual C#中的使用

(1) 启动Visual Studio 2008，按照“File”->"New"，选择建立C#工程；

(2) 将VC#文件夹中的动态链接库和函数声明文件复制到工程文件夹中，注意：gts.dll应复制到"..\bin"文件夹中的debug或者release文件夹中；

(3) 选择“project”菜单下的“Add existing Item”菜单项，选择函数声明文件，如gts.cs，将其添加到工程当中；

至此，用户就可以在Visual Studio 2008中使用C#模块调用函数库中的任何函数，开始编写应用程序。

第3章 指令返回值及其意义

3.1 本章简介

本章主要介绍了运动控制器指令的所有返回值及其意义。在”第 13 章 指令详细说明”，每一条指令介绍的“指令返回值”一项中，都有更加详细的关于返回值意义和操作的介绍。

3.2 指令返回值

运动控制器按照主机发送的指令工作。运动控制器指令封装在动态链接库中。用户在编写应用程序时，通过调用运动控制器指令来操纵运动控制器。

运动控制器在接收到主机发送的指令时，将执行结果反馈到主机，指示当前指令是否正确执行。指令返回值的定义如表 3-1 所示。

表 3-1 运动控制器指令返回值定义

返回值	意义	处理方法
0	指令执行成功	无
1	指令执行错误	检查当前指令的执行条件是否满足
2	license 不支持	如果需要此功能，请与生产厂商联系。
7	指令参数错误	检查当前指令输入参数的取值
-1	主机和运动控制器通讯失败	1. 是否正确安装运动控制器驱动程序 2. 检查运动控制器是否接插牢靠 3. 更换主机 4. 更换控制器 5. 运动控制器的金手指是否干净
-6	打开控制器失败	1. 是否正确安装运动控制器驱动程序 2. 是否调用了 2 次 GTN_Open 指令 3. 其他程序是否已经打开运动控制器，或进程中是否还驻留着打开控制器的程序
-7	运动控制器没有响应	更换运动控制器
-10	主机和运动控制器通讯失败	1.PCI-E 松动，需要重新加载驱动 2.或者重启电脑、插紧主卡并拧上固定螺丝
-11	动态库加载失败	部分函数不支持，需要替换 gt_rn.dll
-13	编码器初始化失败	1.需要检测 core1 所连模块是否工作正常 2.对于拿云系列产品需要检测驱动 DSP 是否工作正常
-14	编码器初始化失败	1.需要检测 core2 所连模块是否工作正常 2.对于拿云系列产品需要检测驱动 DSP 是否工作正常
-15	动态库版本不匹配	无法正常工作，需要更新 gt_rn.dll
15	动态库版本不匹配	某些功能不能正常使用，需要更新 gt_rn.dll
16	不具备版本匹配功能	DSP 版本比较老，建议更新 DSP 固件

3.3 例程

例程 3-1 检测 GTN 指令是否正常执行

检测 GTN 指令是否正常执行，该程序在 VC 6.0 的 win32 console application 工程下运行。

```
#include "stdafx.h"
#include "windows.h"
#include "stdio.h"
#include "gts.h"

// 该函数检测某条GTN指令的执行结果，command为指令名称，error为指令执行返回值
void commandhandler(char *command, short error)
{
    // 如果指令执行返回值为非0，说明指令执行错误，向屏幕输出错误结果
    if(error)
    {
        printf("%s = %d\n", command, error);
    }
}

int main(int argc, char* argv[])
{
    // 指令返回值变量
    short sRtn;
    sRtn =  GTN_Open();
    // 指令返回值校验
    commandhandler(" GTN_Open", sRtn);
    return 0;
}
```

建议在用户程序中，检测每条指令的返回值，以判断指令的执行状态。并建立必要的错误处理机制，保证程序安全可靠地运行。

第4章 新手必读

4.1 本章简介

GXN 控制器是一款网络化控制器，目前包括 GTN 系列、GHN 系列、GSN 系列。主卡与网络模块、网络模块与网络模块都是通过千兆网通讯。主卡软件资源和网络模块硬件资源之间是完全独立的，通过相应的指令或配置文件进行配置(默认控制器会按照端子板连接顺序将主卡软件资源和硬件资源依次映射，详细的默认资源映射可参照章节 4.4)。

4.2 主卡软件资源

4.2.1 指令列表

表 4-1 设置软件资源指令

指令	说明	页码
GTN_SetResCount	设置主卡资源个数	250
GTN_GetResCount	读取主卡当前资源个数	207
GTN_GetResMax	读取主卡最大资源个数	207

4.2.2 重点说明

GXN 系列控制器采用多核微处理器，其中两个内核用于运动控制(简称 core1 和 core2)。默认状态下双核(core1 和 core2)之间软件资源是完全独立的。手册中所有 GTN_指令都是针对软件资源。

默认情况下，双核软件资源会根据所接模块类型和个数自适应。如果 core1 或 core2 所接模块对应的硬件资源总和（详见）小于等于最小值，则双核软件资源默认为最小值；如果硬件资源大于最小值时，双核会根据模块类型自动设置软件资源，但是资源个数不能超过最大值。

GTN 系列控制器按照软件资源分类，包括 4 中类型：GTN-016-G-CC、GTN-016-V-CC、GTN-016-G-CC、GTN-016-V-CC。

GSN 系列控制器目前有 4 种类型：GSN-024-G-00、GSN-024-V-00、GSN-024-GT-00、GSN-024-VT-00。

表 4-2 GXN 系列控制器软件资源列表

资源(注 1)	GTN-024-G(V)-CC(注 2)		GTN-016-G(V)-CC(注 2)	
	core1(注 3)	core2	core1(注 3)	core2
轴（AXIS）	8(12)	8(12)	8(8)	8(8)
规划器(Profile)	8(12)	8(12)	8(8)	8(8)
伺服控制器(Controller) (注 4)	8(12) (注 4)	8(12) (注 4)	8(8) (注 4)	8(8) (注 4)
脉冲输出(Step)	8(12)	8(12)	8(8)	8(8)
内部脉冲计数器(PULSE)	0(12)	0(12)	0(8)	0(8)
捕获（Trigger）	8(12)	8(12)	8(8)	8(8)
编码器（Encoder）	8(12)	8(12)	8(8)	8(8)
辅助编码器	0(6)	0(6)	0(4)	0(4)

资源(注 1)	GTN-024-G(V)-CC(注 2)		GTN-016-G(V)-CC(注 2)	
	core1(注 3)	**core2**	**core1**(注 3)	**core2**
MPG 编码器	0(3)	0(3)	0(2)	0(2)
模拟量输出(DAC)注 5	8(12)	8(12)	8(8)	8(8)
非轴模拟量输出(AuDAC)注 5	0(6)	0(6)	0(4)	0(4)
模拟量输入(ADC)注 6	0(0)	0(0)	0(0)	0(0)
非轴模拟量输入(AuADC)注 6	0(24)	0(24)	0(16)	0(16)
手脉输入（MPG 的 DI）	0(21)	0(21)	0(14)	0(14)
原点(HOME)	8(12)	8(12)	8(8)	8(8)
电机到位(ARRIVE)	8(12)	8(12)	8(8)	8(8)
轴报警(ALM)	8(12)	8(12)	8(8)	8(8)
轴正、负限位(LIMIT+、LIMIT-)	8(12)	8(12)	8(8)	8(8)
轴使能(ENABLE)	8(12)	8(12)	8(8)	8(8)
轴报警清除(CLEAR)	8(12)	8(12)	8(8)	8(8)
通用输入(GIN)	16(66)	16(66)	16(44)	16(44)
通用输出(OUT)	16(30)	16(30)	16(20)	16(20)
振镜(Scan)注 7	0(3)	0(0)	0(2)	0(0)
位置比较输出(PosCompare)	4(6)	4(6)	4(4)	4(4)
扩展模块数字量输入输出(EXT-DI/EXT-DO)注 8	2048/2048	2048/2048	2048/2048	2048/2048
扩展模块模拟量输入输出(EXT-AI/EXT-AO)注 9	384/384	384/384	384/384	384/384

表 4-2 （续）GXN 系列控制器软件资源列表

资源(注 1)	GSN-024-G(V)(T)-00	
	core1(注 3)	**core2**
轴（AXIS）	8(12)	8(12)
规划器(Profile)	8(12)	8(12)
伺服控制器(Controller) (注 4)	8(12) (注 4)	8(12) (注 4)
脉冲输出(Step)	8(12)	8(12)
内部脉冲计数器(PULSE)	8(12)	8(12)
捕获（Trigger）	8(12)	8(12)
编码器（Encoder）	8(12)	8(12)
辅助编码器	0(6)	0(6)
MPG 编码器	0(3)	0(3)
模拟量输出(DAC)注 5	8(12)	8(12)
非轴模拟量输出(AuDAC)注 5	0(6)	0(6)
模拟量输入(ADC)注 6	0(8)	0(8)
非轴模拟量输入(AuADC)注 6	0(24)	0(24)
手脉输入（MPG 的 DI）	0(21)	0(21)
原点(HOME)	8(12)	8(12)
电机到位(ARRIVE)	8(12)	8(12)
轴报警(ALM)	8(12)	8(12)
轴正、负限位(LIMIT+、LIMIT-)	8(12)	8(12)
轴使能(ENABLE)	8(12)	8(12)

资源(注 1)	GSN-024-G(V)(T)-00	
	core1(注 3)	**core2**
轴报警清除(CLEAR)	8(12)	8(12)
通用输入(GIN)	16(100)	16(100)
通用输出(OUT)	16(64)	16(64)
振镜(Scan)注 7	0(3)	0(0)
位置比较输出(PosCompare)	4(6)	4(6)
扩展模块数字量输入输出(EXT-DI/EXT-DO)注 8	2048/2048	2048/2048
扩展模块模拟量输入输出(EXT-AI/EXT-AO)注 9	384/384	384/384

表 4-2 （续）GXN 系列控制器软件资源列表

资源(注 1)	GSN-048-G-01	
	core1(注 3)	**core2**
轴（AXIS）	8(24)	8(24)
规划器(Profile)	8(24)	8(24)
伺服控制器(Controller) (注 4)	0(0) (注 4)	0(0) (注 4)
脉冲输出(Step)	8(24)	8(24)
内部脉冲计数器(PULSE)	8(24)	8(24)
捕获（Trigger）	8(24)	8(24)
编码器（Encoder）	8(24)	8(24)
辅助编码器	0(18)	0(18)
MPG 编码器	0(3)	0(3)
模拟量输出(DAC)注 5	8(24)	8(24)
非轴模拟量输出(AuDAC)注 5	0(12)	0(12)
模拟量输入(ADC)注 6	0(24)	0(24)
非轴模拟量输入(AuADC)注 6	0(48)	0(48)
手脉输入（MPG 的 DI）	0(21)	0(21)
原点(HOME)	8(24)	8(24)
电机到位(ARRIVE)	8(24)	8(24)
轴报警(ALM)	8(24)	8(24)
轴正、负限位(LIMIT+、LIMIT-)	8(24)	8(24)
轴使能(ENABLE)	8(24)	8(24)
轴报警清除(CLEAR)	8(24)	8(24)
通用输入(GIN)	16(100)	16(100)
通用输出(OUT)	16(40)	16(40)
振镜(Scan)注 7	0(4)	0(4)
位置比较输出(PosCompare)	4(8)	4(8)
扩展模块数字量输入输出(EXT-DI/EXT-DO)注 8	2048/2048	2048/2048
扩展模块模拟量输入输出(EXT-AI/EXT-AO)注 9	384/384	384/384

表 4-2 （续）R688C 系列控制器软件资源列表

资源(注 1)	R688C-024-G(V)(T)-00
	core1(注 3)
轴（AXIS）	8(24)

资源(注 1)	R688C-024-G(V)(T)-00
	core1(注 3)
规划器(Profile)	8(24)
伺服控制器(Controller) (注 4)	8(24) (注 4)
脉冲输出(Step)	8(24)
内部脉冲计数器(PULSE)	8(24)
捕获（Trigger）	8(24)
编码器（Encoder）	8(24)
辅助编码器	2(6)
MPG 编码器	1(3)
模拟量输出(DAC)注 5	8(24)
非轴模拟量输出(AuDAC)注 5	0(6)
模拟量输入(ADC)注 6	0(24)
非轴模拟量输入(AuADC)注 6	0(24)
手脉输入（MPG 的 DI）	7(28)
原点(HOME)	8(24)
电机到位(ARRIVE)	8(24)
轴报警(ALM)	8(24)
轴正、负限位(LIMIT+、LIMIT-)	8(24)
轴使能(ENABLE)	8(24)
轴报警清除(CLEAR)	8(24)
通用输入(GIN)	19(100)
通用输出(OUT)	30(40)
振镜(Scan)注 7	0(4)
位置比较输出(PosCompare)	4(8)
扩展模块数字量输入输出(EXT-DI/EXT-DO)注 8	2048/2048
扩展模块模拟量输入输出(EXT-AI/EXT-AO)注 9	384/384

注 1：本列中轴（AXIS），轴代表资源名称，(AXIS)代表该资源对应端子板硬件通道名
注 2：GTN 主卡型号，不同型号产品资源不一样，具体可参考《GTN 系列运动控制器用户手册》
注 3：本列中 8(12)，8 代表对应端口默认的资源个数，(12)代表对应核最大的资源个数
注 4：对于 GSN-0XX-G-CC/GTN-0XX-G-CC 系列控制器，伺服控制器 controller 资源为 0.
注 5，注 6：模拟量输入和输出只有 4 轴模块有硬件通道，非轴模拟量指硬件接口不在轴接口里面。
注 7：振镜功能只针对振镜模块
注 8，注 9：扩展模块数字量输入和输出资源对应核的最大资源分别为 2048 路，不可以设置

4.2.3 例程

例程 4-1　设置主卡软件资源

```
// 通过指令设置 Core1 中和轴相关的软件资源为 12

#include "stdafx.h"
#include "windows.h"
#include "stdio.h"
#include "gts.h"
```

```
int main(int argc, char* argv[])
{
    // 指令返回值变量
    short sRtn;
    short presCnt;
    sRtn =  GTN_Open();
    sRtn = GTN_Reset(1);
    sRtn = GTN_Reset(2);
    sRtn = GTN_GetResMax(1,MC_AXIS,&presCnt);
    sRtn = GTN_SetResCount(1,MC_AXIS,12);
    sRtn = GTN_GetResCount(1,MC_AXIS,&presCnt);
    return 0;
}
```

4.3 端子板模块

端子板模块按照轴的硬件资源分类有 2 种类型： 6 轴端子板和 4 轴端子板。6 轴和 4 轴类型又包括多种不同的型号。端子板型号和接口定义详细介绍具体可参考《GNM 系列端子板模块用户手册》。

4.3.1 指令列表

表 4-3 读取端子板状态指令

指令	说明	页码
GTN_GetTerminalStatus	读取模块通讯状态	211

4.3.2 重点说明

端子板模块的通讯端口的工作状态可以通过指令 GTN_GetTerminalStatus 获取。

4.3.3 例程

例程 4-2 读取模块状态

```
//通过指令读取所接模块个状态
#include "stdafx.h"
#include "windows.h"
#include "stdio.h"
#include "gts.h"
int main(int argc, char* argv[])
{
    // 指令返回值变量
    short sRtn,terminalCnt;
    short terminalType[4], terminalCon[4];
    short terminalLinkSts;
    int i;
    sRtn = GTN_Open();
```

```
        sRtn= GTN_Reset(1);
        sRtn= GTN_Reset(2);
    //读取模块个数
        sRtn= GTN_GetResCount(1,MC_TERMINAL,&terminalCnt);
        for(i=1;i<=terminalCnt;i++)
        {
            sRtn =   GTN_GetTerminalStatus(1,i,&terminalLinkSts);
        }
        return 0;
    }
```

4.4 资源映射

主卡软件逻辑资源和端子板硬件资源是相互独立的。在使用运动控制器进行各种操作之前，需要对运动控制器进行配置，使运动控制器的软件资源能控制对应的端子板硬件通道，这个过程，叫做映射。资源映射包括端子板模块映射和软件资源映射。

图 4-1　映射示意图

4.4.1 映射

默认情况下，控制器会按照端子板的类型和连接顺序自动映射，分配原则为优先考虑软件资源中的轴资源。扩展模块的资源映射请参考《GXN 扩展功能-扩展模块功能编程手册》。以 GTN-024-G(V)-CC 主卡类型为例介绍模块连接情况：

情况 1：硬件轴（Axis）总资源小于或等于 12（即 core1 的最大轴资源个数）

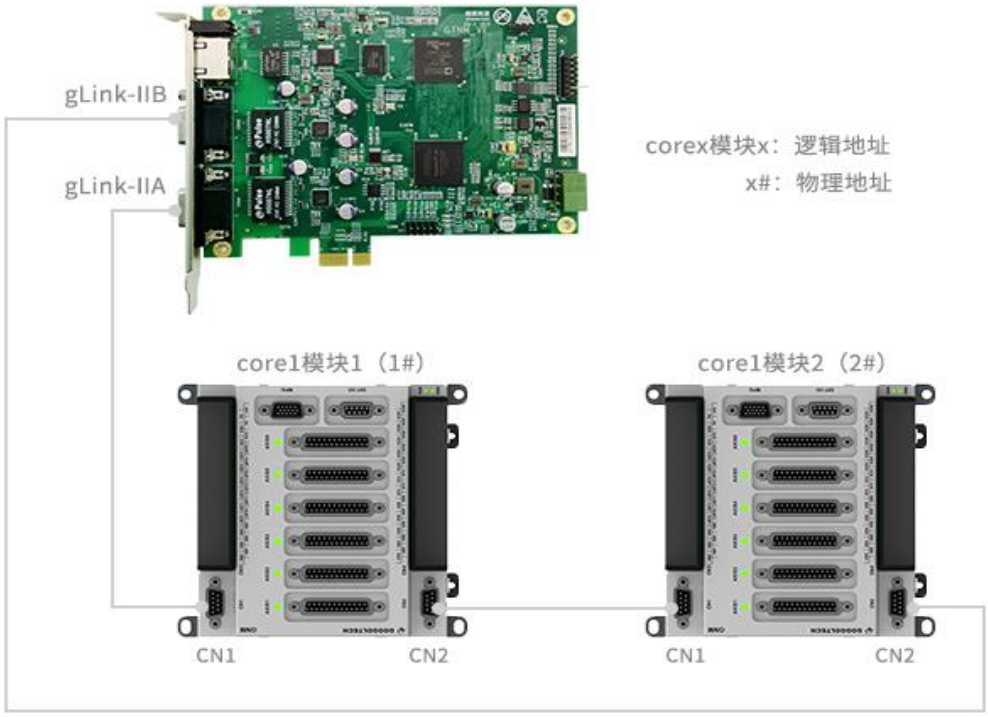


图 4-2　　端子板分配图 1

如图 4-2 所示，端子板模块 1 和端子板模块 2 都分配至控制器的 core1。主卡软件资源和端子板硬件物理资源的映射关系如表 4-4 所示。

表 4-4　　core1 资源映射列表（轴资源个数≤12）

软件和硬件资源映射			
资源名称	软件资源	硬件资源	
		端子板模块序号	端子板模块丝印
轴	1～6	core1 模块 1(1#)	AXIS1~AXIS6
	7~ 12	core1 模块 2(2#)	AXIS1~AXIS6
通用输入	1~16	core1 模块 1(1#)	DI00~DI15
	17~32	core1 模块 2(2#)	DI00~DI15
通用输出	1~10	core1 模块 1(1#)	DO00~DO09
	11~20	core1 模块 2(2#)	DO00~DO09
MPG	MPG_Encoder：1	core1 模块 1(1#)	MPG
	MPG_DI：0~7		
	MPG_Encoder：2	core1 模块 2(2#)	MPG
	MPG_DI：8~13		

情况 2:物理轴（Axis）总资源大于 12 个（即 core1 的最大轴资源个数），小于等于 24（即 core1 和 core2 的最大轴资源总个数）

默认优先按照 core1 中的软件资源分配，然后再按照 core2 的软件资源分配（注意：同一个轴模块硬件中的物理资源不能同时分配至 core1 和 core2。对于这种情况，如果 core1 的逻辑资源不足，将模块分配在 core1 会导致资源的浪费，则直接将该模块分配至 core2）。

最终资源分配结果是：前 2 个端子板模块分配为核 1，后两个分配为核 2。如图 4-3 所示。资源映射详见表 4-5 和

表 4-6。

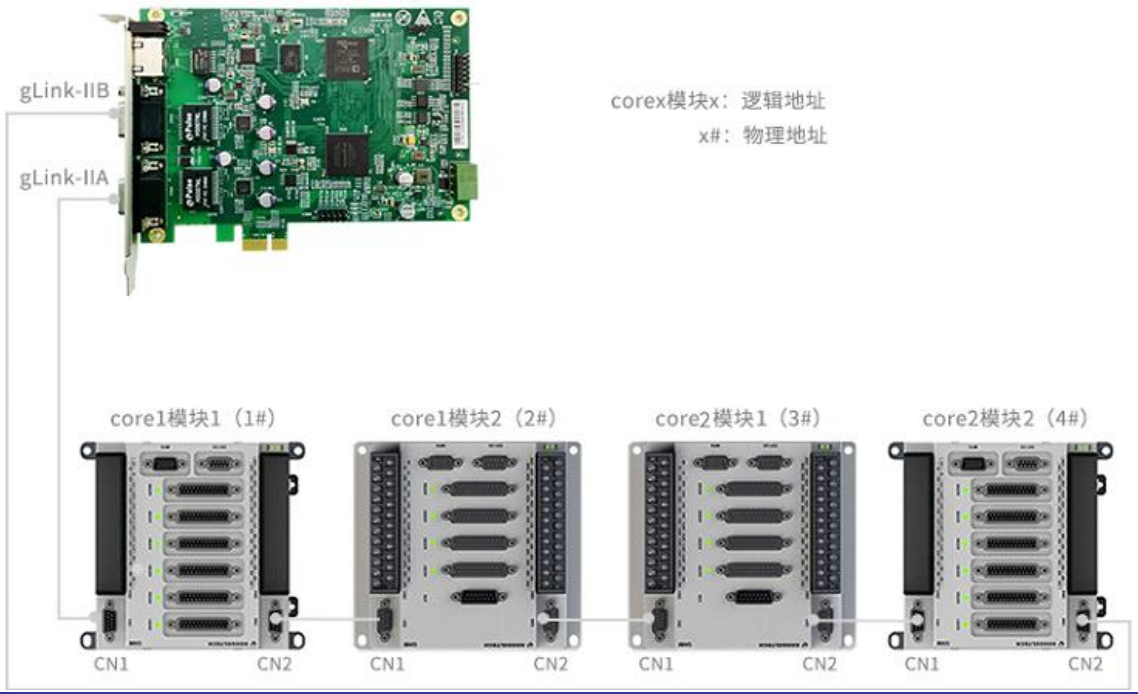


图 4-3　端子板分配图 2

表 4-5　core1 资源映射列表（12＜轴资源个数≤24）

软件和硬件资源映射			
资源名称	软件资源	硬件资源	
		端子板模块序号	端子板模块丝印
轴	1～6	core1 模块 1(1#)	AXIS1～AXIS6
	7～10	core1 模块 2(2#)	AXIS1～AXIS4
通用输入	1～16	core1 模块 1(1#)	DI00～DI15
	17～38	core1 模块 2(2#)	DI00～DI21
通用输出	1～10	core1 模块 1(1#)	DO00～DO09
	11～20	core1 模块 2(2#)	DO00～DO09
MPG	MPG_Encoder：1	core1 模块 1(1#)	MPG
	MPG_DI：0～6		
	MPG_Encoder：2	core1 模块 2(2#)	MPG
	MPG_DI：7～13		
非轴模拟量输入	1～8	core1 模块 2(2#)	AIN

表 4-6　core2 资源映射列表（12＜轴资源个数≤24）

软件和硬件资源映射			
资源名称	软件资源	硬件资源	
		端子板模块序号	端子板模块丝印
轴	1～4	core2 模块 1(3#)	AXIS1～AXIS4
	5～10	core2 模块 2(4#)	AXIS1～AXIS6
通用输入	1～22	core2 模块 1(3#)	DI00～DI21
	23～39	core2 模块 2(4#)	DI00～DI15
通用输出	1～10	core2 模块 1(3#)	DO00～DO09
	11～20	core2 模块 2(4#)	DO00～DO09
MPG	MPG_Encoder：1	core2 模块 1(3#)	MPG
	MPG_DI：0～6		
	MPG_Encoder：2	core2 模块 2(4#)	MPG
	MPG_DI：7～13		
非轴模拟量输入	1～8	core2 模块 1(3#)	AIN

第5章 系统配置

5.1 本章简介

在使用运动控制器进行各种操作之前，需要对运动控制器中进行配置，使运动控制器的状态和各种工作模式能够满足客户的要求。这个过程，叫做系统配置。

在运动控制器管理软件 MotionStudio（以下简称 MS）中包括一个系统配置的组件，用户可以利用该组件来对运动控制器进行配置，配置完成之后，生成相应的配置文件*.cfg，用户在编程时，调用相关的指令，将配置信息传递给运动控制器，即可完成整个运动控制器的配置工作。用户也可以利用相关的指令完成运动控制器的配置。

本手册中所有字体为蓝色的指令（如 GTN_PrfTrap）均带有超级链接，点击可跳转至指令说明。

5.2 系统配置基本概念

运动控制器内部包含了各种软硬件资源，各种软硬件资源之间相互组合，即可实现运动控制器的各种应用。

5.2.1 端子板模块映射

由于主卡包含 2 个完全独立的网络连接端口（gLinkII-A 和 gLinkII-B）,可以通过 MotionStudio 读取和配置每个端口所连接的端子板模块个数和类型。默认情况下，用户调用 GTN_Open 会根据主卡每个 core 的软件资源中的轴资源自动配置对应的关系。

5.2.2 硬件资源

数字量输出资源(do)：包括伺服使能数字量输出、伺服报警清除数字量输出、通用数字量输出。

数字量输入资源(di)：包括正限位数字量输入、负限位数字量输入、驱动报警数字量输入、原点信号数字量输入、通用数字量输入。

编码器计数资源(encoder)：用来对外部编码器的脉冲输出进行计数。

脉冲输出资源(step)：脉冲输出通道，可以输出“脉冲+方向”或者“CCW/CW”控制脉冲。

电压输出资源(dac)：电压输出通道，输出-10V~+10V 的控制电压。

5.2.3 软件资源

规划器资源(profile)：根据运动模式和运动参数实时计算规划位置和规划速度，生成所需的速度曲线，实时地输出规划位置。

伺服控制器资源(control)：根据伺服控制算法、控制参数、跟随误差实时地计算控制量。

轴资源(axis)：将软件资源、硬件资源进行组合，作为整体进行操作。其中包括驱动报警信号、正限位

信号、负限位信号、平滑停止信号、紧急停止信号的管理；规划器输出的规划位置的当量变换；编码器计数位置的当量变换等功能。

5.2.4 资源组合

系统配置就是将上述的硬件资源和软件资源相互组合，并对各个资源的基本属性进行配置的过程。下面的两个例子描述了资源组合的基本概念。

1. 开环控制模式（脉冲控制）

使用步进电机，或使用伺服电机的位置控制模式的运动控制系统，其基本配置如图 5-1 所示。

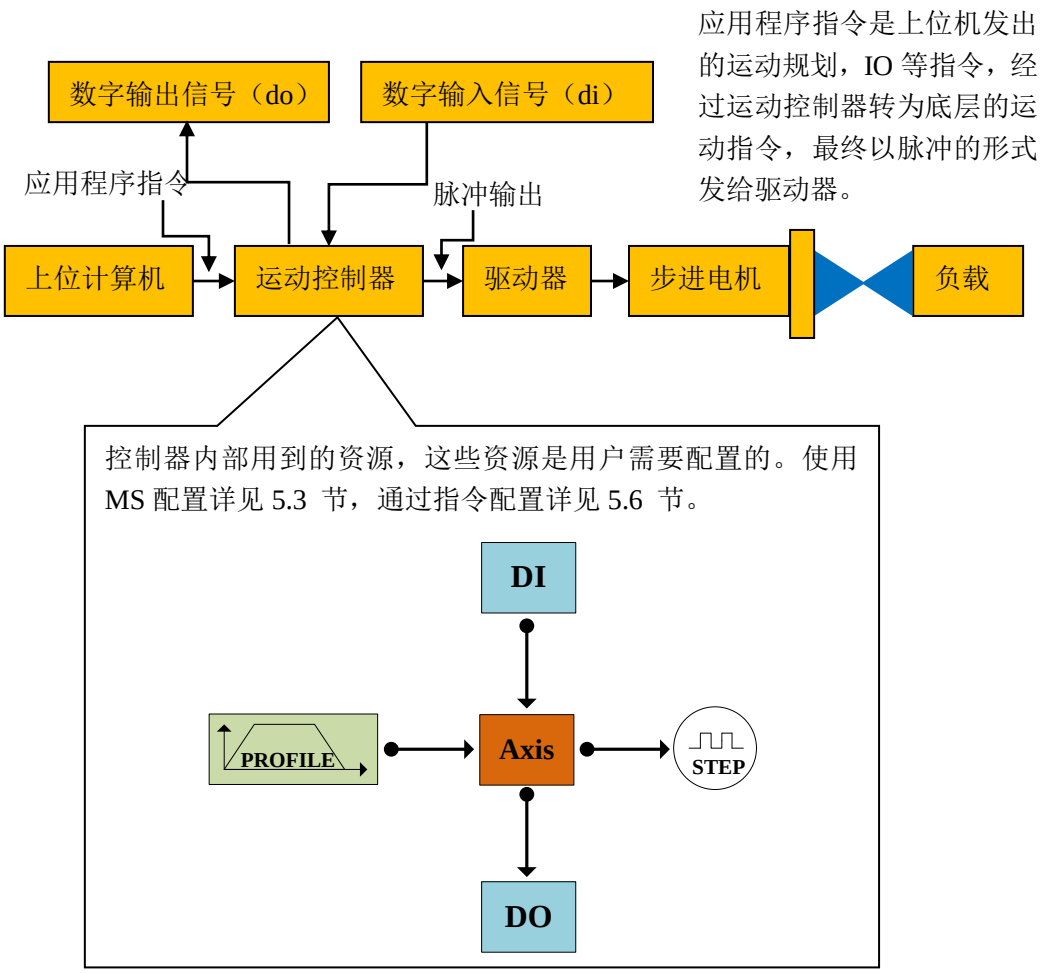


图 5-1 开环运动控制系统的配置

该实例中，profile 输出的规划位置进入 axis 中，在 axis 中进行当量变换的处理后，输出到 step，由 step 产生控制脉冲，驱动电机运动。axis 需要驱动报警、正向限位信号、负向限位信号、平滑停止信号、紧急停止信号等一些数字量输入信号来对运动进行管理；同时，axis 需要输出伺服使能信号，让电机使能。

2. 闭环控制模式（模拟量控制）

一个闭环伺服电机运动控制系统的常见组成部分如图 5-2 所示。

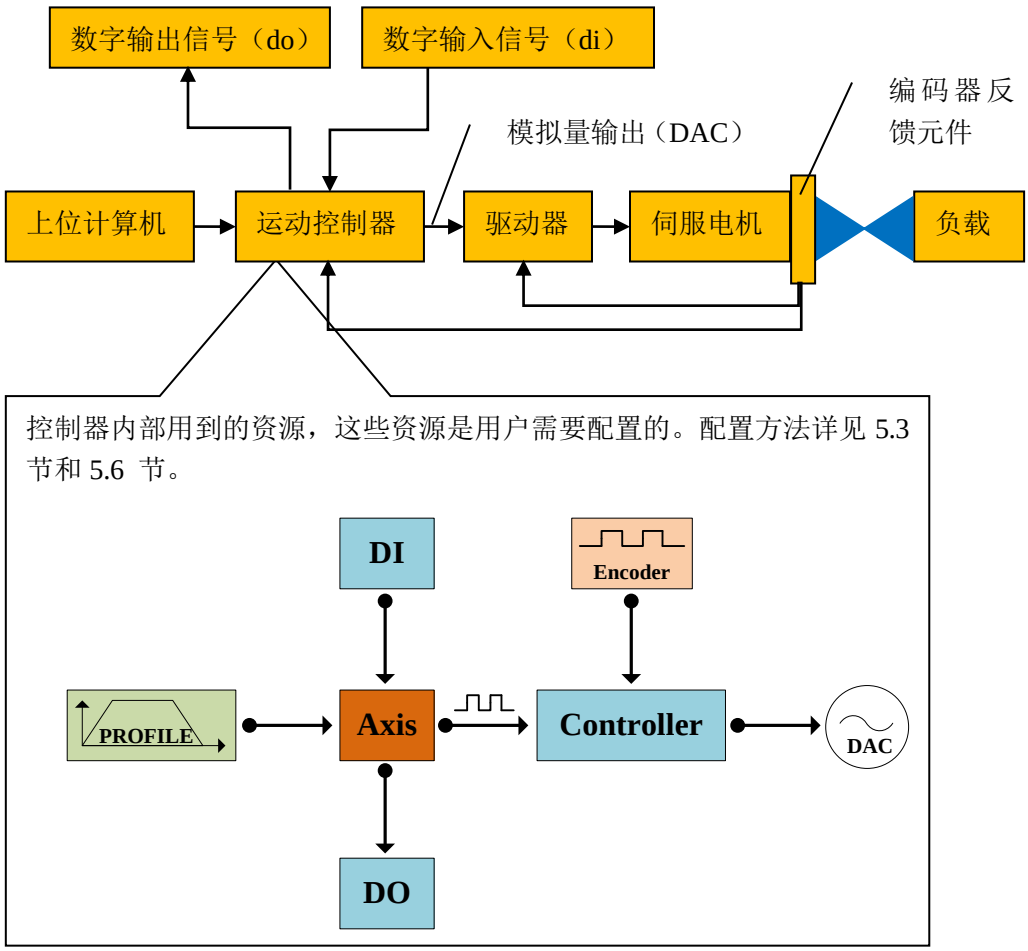


图 5-2　闭环运动控制系统的配置

该实例中，profile 输出的规划位置进入 axis 中，在 axis 中进行当量变换的处理后，输出到伺服控制器中，伺服控制器将规划位置与 encoder 的计数位置进行比较，获得跟随误差，并通过一定的伺服控制算法，得到实时的控制量，将控制量传递给 dac，由 dac 转换成控制电压来控制电机的运动。axis 需要驱动报警、正向限位信号、负向限位信号、平滑停止信号、紧急停止信号等一些数字量输入信号来对运动进行管理；同时，axis 需要输出伺服使能信号，让电机使能。

5.3 系统配置工具

使用固高科技提供的 MotionStudio 运动控制器管理软件能够方便地对系统进行配置，启动软件以后显示如图 5-3 所示界面。

图 5-3 MotioStudio 运动控制器管理软件界面

选择“工具”菜单，点击“控制器配置”，打开运动控制器配置面板就可以对系统进行配置。如图 5-4 所示。

图 5-4 打开控制器配置

5.3.1 配置 axis

配置说明：“axis”选项主要用来配置轴控制的相关信息。配置后对控制系统可能产生的影响如图 5-5 所示。

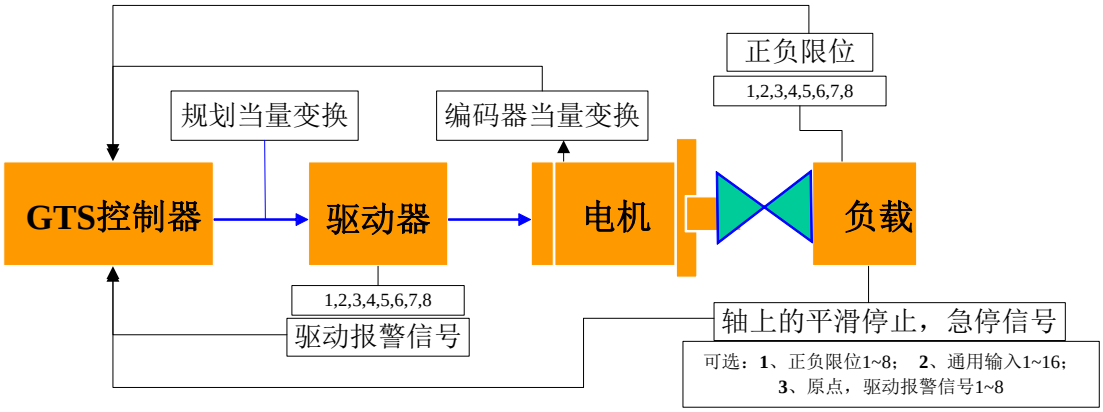


图 5-5 axis 配置对控制系统的影响

按照功能，MS 将 Axis 配置功能分为 3 个大模块，分别为报警功能、停止功能和当量变换功能。界面如图 5-6~图 5-8 所示。

1. 报警功能

一般情况下，驱动报警、正负限位信号均保持默认设置状态，即“驱动报警”，“正限位”和“负限位”的“编号”同“轴号”相对应。为了帮助用户提高配置的灵活性，可选用其他配置，如将轴 2 的限位信号配给轴 1 使用。用户设置时，一般情况下，该选项保持默认设置，除非出现特殊需求。

轴索引号	(驱动报警)类型	(驱动报警)索引号	(正限位)类型	(正限位)索引号	(负限位)类型	(负限位)索引号
轴：1	轴报警	1	正限位	1	负限位	1
轴：2	轴报警	2	正限位	2	负限位	2
轴：3	轴报警	3	正限位	3	负限位	3
轴：4	轴报警	4	正限位	4	负限位	4
轴：5	轴报警	5	正限位	5	负限位	5
轴：6	轴报警	6	正限位	6	负限位	6
轴：7	轴报警	7	正限位	7	负限位	7
轴：8	轴报警	8	正限位	8	负限位	8
轴：9	轴报警	9	正限位	9	负限位	9
轴：10	轴报警	10	正限位	10	负限位	10
轴：11	轴报警	11	正限位	11	负限位	11
轴：12	轴报警	12	正限位	12	负限位	12

图 5-6 axis 配置界面 1

(1) 驱动报警信号数字量输入选择：选择驱动报警信号的数字量输入的来源，运动控制器支持将任何数字量输入信号配置为驱动报警信号，增加用户进行硬件接线的自由性。图 5-6”（驱动报警）索引号”下拉列表选择数字量输入的类型，默认为选择驱动报警数字量输入；”（驱动报警）索引号”下拉列表选择数字量输入的索引号，在列表中如果选择“none”，则表示该 axis 的驱动报警信号无效。驱动报警无效可以通过指令 GTN_AlarmOff 设置，驱动报警有效可以通过指令 GTN_AlarmOn 设置。

(2) 正限位信号数字量输入选择：选择正限位信号的数字量输入的来源，运动控制器支持将任何数字量输入信号配置为正限位信号，增加用户进行硬件接线的自由性。图 5-6”（正限位）类型”下拉

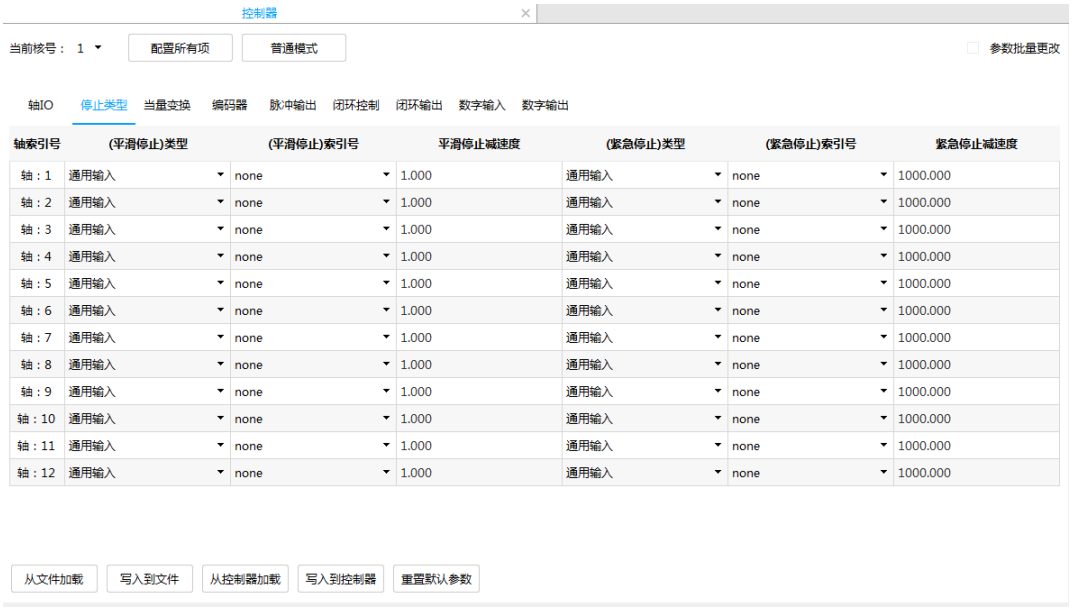
列表选择数字量输入的类型，默认为选择正限位数字量输入；”（正限位）索引号”下拉列表选择数字量输入的编号，在列表中如果选择“none”，则表示该 axis 的正限位信号无效。限位开关无效可以通过指令 GTN_LmtsOffEx 设置，限位开关有效可以通过指令 GTN_LmtsOnEx 设置。

(3) 负限位信号数字量输入选择：选择负限位信号的数字量输入的来源，运动控制器支持将任何数字量输入信号配置为负限位信号，增加用户进行硬件接线的自由性。图 5-6”（负限位）类型”下拉列表选择数字量输入的类型，默认为选择负限位数字量输入；”（负限位）索引号”下拉列表选择数字量输入的编号，在下拉列表中如果选择“none”，则表示该 axis 的负限位信号无效。限位开关无效可以通过指令 GTN_LmtsOffEx 设置，限位开关有效可以通过指令 GTN_LmtsOnEx 设置。

驱动报警信号、正负限位信号在控制器复位状态下，都默认各轴对应相应的报警和限位信号，如 1 轴的驱动报警以及正负限位的编号都是 1 号。

但若有特殊情况，可以通过设置将不同轴上的限位和报警信号挂接到本轴上，如将 1 轴的正限位设置为 2 号。这种情况下，若 2 号正限位触发了，该信号将传递给 1 轴。

2. 配置停止类型和停止减速度



轴索引号	(平滑停止)类型	(平滑停止)索引号	平滑停止减速度	(紧急停止)类型	(紧急停止)索引号	紧急停止减速度
轴：1	通用输入	none	1.000	通用输入	none	1000.000
轴：2	通用输入	none	1.000	通用输入	none	1000.000
轴：3	通用输入	none	1.000	通用输入	none	1000.000
轴：4	通用输入	none	1.000	通用输入	none	1000.000
轴：5	通用输入	none	1.000	通用输入	none	1000.000
轴：6	通用输入	none	1.000	通用输入	none	1000.000
轴：7	通用输入	none	1.000	通用输入	none	1000.000
轴：8	通用输入	none	1.000	通用输入	none	1000.000
轴：9	通用输入	none	1.000	通用输入	none	1000.000
轴：10	通用输入	none	1.000	通用输入	none	1000.000
轴：11	通用输入	none	1.000	通用输入	none	1000.000
轴：12	通用输入	none	1.000	通用输入	none	1000.000

图 5-7 axis 配置界面 2

(1) 平滑停止信号数字量输入选择：选择平滑停止信号的数字量输入的来源，运动控制器支持将任何数字量输入信号配置为平滑停止信号，增加用户进行硬件接线的自由性。图 5-7”（平滑停止）类型”下拉列表选择数字量输入的类型，默认为没有平滑停止信号；“（平滑停止）索引号”下拉列表选择数字量输入的编号，下拉列表中如果选择“none”，则表示该 axis 没有平滑停止信号。平滑停止信号数字量输入选择可以通过指令 GTN_SetStopIo 设置。

(2) 紧急停止信号数字量输入选择：选择紧急停止信号的数字量输入的来源，运动控制器支持将任何数字量输入信号配置为紧急停止信号，增加用户进行硬件接线的自由性。图 5-7”（紧急停止）类型”下拉列表选择数字量输入的类型，默认为没有紧急停止信号；”（紧急停止）索引号”下拉列表选择数字量输入的编号，下拉列表中如果选择“none”，则表示该 axis 没有紧急停止信号。紧急停止信号数字量输入选择可以通过指令 GTN_SetStopIo 设置。

3. 配置轴脉冲当量

轴索引号	规划器	规划器Alpha	规划器Beta	编码器	编码器Alpha	编码器Beta
轴：1	1	1	1	1	1	1
轴：2	2	1	1	2	1	1
轴：3	3	1	1	3	1	1
轴：4	4	1	1	4	1	1
轴：5	5	1	1	5	1	1
轴：6	6	1	1	6	1	1
轴：7	7	1	1	7	1	1
轴：8	8	1	1	8	1	1
轴：9	9	1	1	9	1	1
轴：10	10	1	1	10	1	1
轴：11	11	1	1	11	1	1
轴：12	12	1	1	12	1	1

图 5-8　axis 配置界面 3

(1) 规划器当量变换参数：如果需要在 axis 中对规划器输出的规划位置进行当量变换，则可以对该项的参数进行设置，当量变换的关系如下：

$$\frac{\Delta P_{profile}}{\Delta P_{axis}} = \frac{Alpha}{Beta}$$

其中：

$\Delta P_{profile}$ ——规划器输出的规划位置的变化量

ΔP_{axis} ——axis 输出的规划位置的变化量

(3) 系统默认的 Alpha 和 Beta 都为 1，因此，规划器输出的规划位置在经过 axis 之后没有经过任何变化。Alpha 的取值范围：(-32767, 0)和(0, 32767)；Beta 的取值范围：(-32767, 0)和(0, 32767)。该项可以通过指令 GTN_ProfileScale 来设置。

(2) 编码器当量变换参数：如果需要在 axis 中对编码器计数的位置值进行当量变换，则可以对该项的参数进行设置，当量变换的关系如下：

$$\frac{\Delta E_{enc}}{\Delta E_{axis}} = \frac{Alpha}{Beta}$$

其中：

ΔE_{enc} ——编码器计数的位置值的变化量

ΔE_{axis} ——axis 输出的编码器位置值的变化量

(4) 系统默认的 Alpha 和 Beta 都为 1，因此，编码器计数的位置值在经过 axis 之后没有经过任何变化。Alpha 的取值范围：(-32767, 0)和(0, 32767)；Beta 的取值范围：(-32767, 0)和(0, 32767)。该项可以

通过指令 GTN_EncScale 来设置。

注意	规划器当量和编码器当量的设置参数要满足 Alpha≥Beta。

5.3.2 配置 step

Step 配置界面如图 5-9 所示。

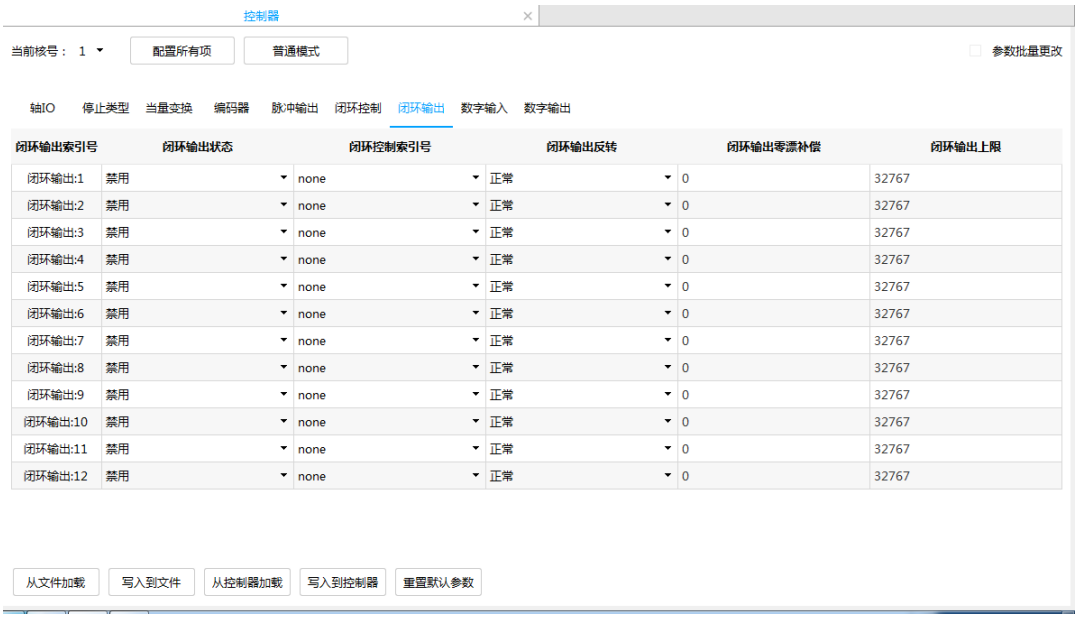
图 5-9 step 配置界面

配置说明：“step”选项主要用于配置脉冲控制模式。若采用的脉冲控制，而且是“脉冲+方向”的方式，该选项保持默认设置。如果轴 1 不使用脉冲控制模式，应当将“脉冲输出状态”下拉列表选择”禁用”。以此类推。

1. 脉冲输出索引号：表示需要进行配置脉冲输出的编号，不可编辑。
2. 脉冲输出模式选择：可以选择脉冲输出通道的脉冲输出模式，可以为“脉冲+方向”或者“CCW/CW”，默认为“脉冲+方向”。设置为“脉冲+方向”模式，可以调用指令 GTN_StepDir 来实现；设置为“CCW/CW”模式，可以调用指令 GTN_StepPulse 来实现。
3. 脉冲输出状态：如果该列表对应的状态为”禁用”，则该脉冲输出通道将不可用，不会输出脉冲。默认脉冲输出都是”启用”的。但是如果没有用到某个 step，则可以把该”脉冲输出”设置为”禁用”，这样可以节约运动控制器的处理资源。

5.3.3 配置 dac

Dac 配置界面如图 5-10 所示。



闭环输出索引号	闭环输出状态	闭环控制索引号	闭环输出反转	闭环输出零漂补偿	闭环输出上限
闭环输出:1	禁用	none	正常	0	32767
闭环输出:2	禁用	none	正常	0	32767
闭环输出:3	禁用	none	正常	0	32767
闭环输出:4	禁用	none	正常	0	32767
闭环输出:5	禁用	none	正常	0	32767
闭环输出:6	禁用	none	正常	0	32767
闭环输出:7	禁用	none	正常	0	32767
闭环输出:8	禁用	none	正常	0	32767
闭环输出:9	禁用	none	正常	0	32767
闭环输出:10	禁用	none	正常	0	32767
闭环输出:11	禁用	none	正常	0	32767
闭环输出:12	禁用	none	正常	0	32767

图 5-10 dac 配置界面

1. 配置说明：“闭环输出”选项主要用于设置 dac 的输出状态。dac 的输出在闭环控制时，作为伺服电压输出控制，用户不可控制伺服电压输出大小。但在开环状态下，可作为通用电压输出控制，用户可调用 GTN_SetDac 设置输出电压的大小。

2. 闭环输出索引号：需要进行配置的 dac 的编号，不可编辑。注意：当电机采用闭环控制模式时，默认是将轴号和 Dac 编号相对应。如轴 1 的电压控制的通道为 Dac1。

3. 闭环输出电压反转：选择是否需要将 dac 的输出电压取反，如果为“正常”，则向 dac 中写入正值时，dac 输出正电压，向 dac 中写入负值时，dac 输出负电压；如果为“取反”，则反之。

4. 闭环输出零漂补偿：如果需要对 dac 进行零漂补偿时，在这里设置具体的零漂补偿值。取值范围：[-32768, 32767]。该项可以通过指令 GTN_SetMtrBias 来设置。

5. 闭环输出上限：该项设置 dac 能够输出的最大电压绝对值。取值范围：(0, 32767]。换算关系如下：已知允许输出的电压范围为-aV~aV，则设置的值为 32767*a/10，取整数。例如，如果设置为 32767，则允许输出的电压范围为：[-10V, +10V]，设置为 16384，则允许输出的电压范围为：[-5V, +5V]。如果 control 输出的控制量绝对值，或者用户使用 GTN_SetDac 指令设置的电压值的绝对值超过设定值时，将会按照该项设置的参数被限制在指定电压范围之内。该项可以通过指令 GTN_SetMtrLmt 来设置。

6. 闭环输出状态：如果下拉列表选择”禁用”，则该电压输出通道将不可用，不会输出电压值。默认“禁用”的。但是如果没有用到某个 dac，则可以不把该 dac 启用，这样可以节约运动控制器的处理资源。

5.3.4 配置 encoder

Encoder 配置界面如图 5-11 所示。

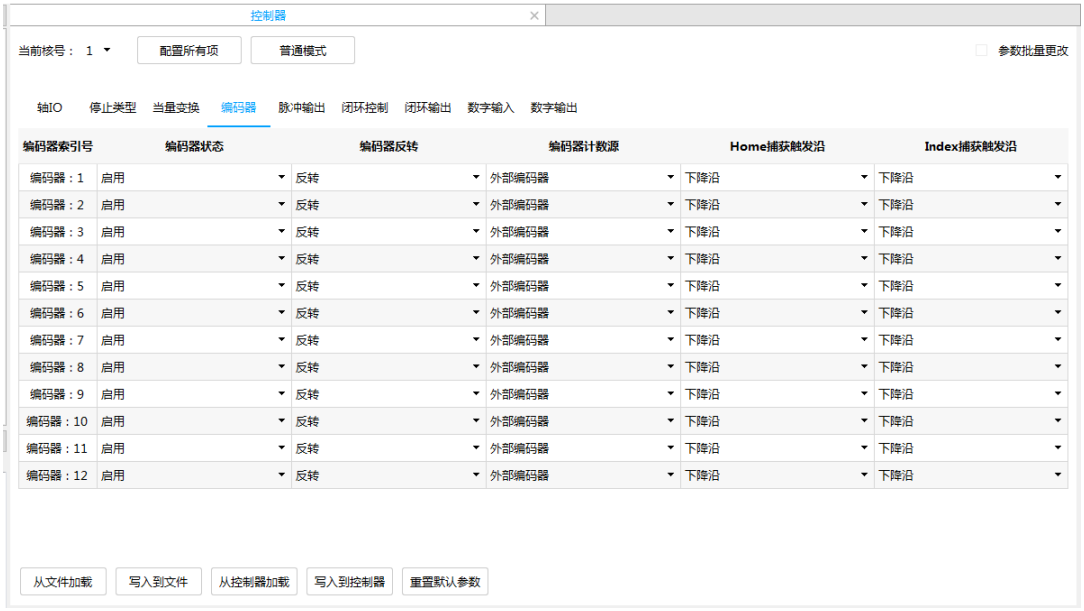


图 5-11 encoder 配置界面

配置说明：“编码器”选项主要配置与编码器有关的参数。配置后对控制系统可能产生的影响如图 5-12 所示。

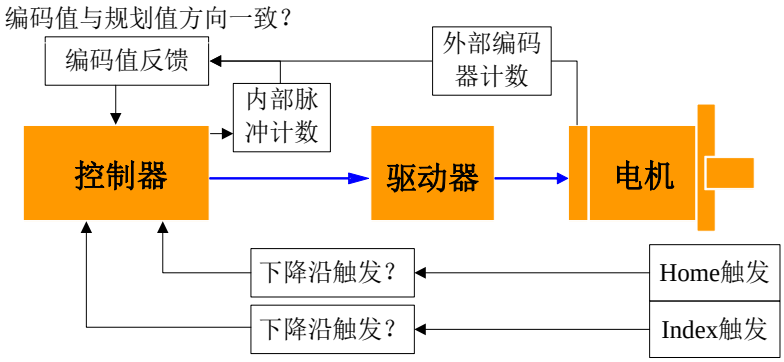


图 5-12 encoder 配置对控制系统的影响

当编码器值与规划值方向相反时，可以通过修改“输入脉冲反转”来校正。在闭环控制模式下，若出现“飞车”现象，也可通过修改该选项来校正。脉冲计数源的设置，一般情况下，保持默认设置。若没安装编码器，则可通过设置该选项为“脉冲计数器”。对于“Home 触发沿”和“Index 触发沿”的设置，取决于传感器的安装，若不能触发，可以通过修改这部分。

1. 编码器索引号：需要进行配置的编码器的编号，不可编辑。

2. 编码器反转：运动控制器可以接收正交编码器信号，该项选项与反馈脉冲方向以及编码器计数方向的关系如图 5-13 所示，该项可以通过指令 GTN_SetSense 来修改。

	正常		取反	
A 相				
B 相				
编码器	计数增加	计数减少	计数增加	计数减少

图 5-13 encoder 输入脉冲反转项的影响

3. 编码器计数源：表示编码器计数来源，默认情况下是外部编码器计数。如果没有外接编码器，则可以将其设置为脉冲计数器，encoder 将会对 step 输出的脉冲个数进行计数。设置为外部编码器，可以调用指令 GTN_EncOn 来实现；设置为脉冲计数器，可以调用指令 GTN_EncOff 来实现。

   设置为外部编码器是指通过外部安装的编码器计数值来计算，而脉冲计数器则是指通过控制器内部硬件来计算发出去的脉冲个数。在闭环控制方式下，必须设置成外部编码器计数方式。

4. Home 捕获触发沿：用来设置 Home 捕获的触发沿，默认为下降沿触发。如果选择了常闭开关，可以将捕获沿设置为上升沿触发。该项可以通过指令 GTN_SetTriggerEx 来修改。

5. Index 捕获触发沿：用来设置 Index 捕获的触发沿，默认为下降沿触发。该项可以通过指令 GTN_SetTriggerEx 来修改。

6. 编码器状态：如果编码器不被激活，则将不会对输入脉冲进行计数。默认编码器都是激活的。但是如果没有用到某个编码器，则可以不把该编码器激活，这样可以节约运动控制器的处理资源。

5.3.5 配置 control

Control 配置界面如图 5-14 所示。

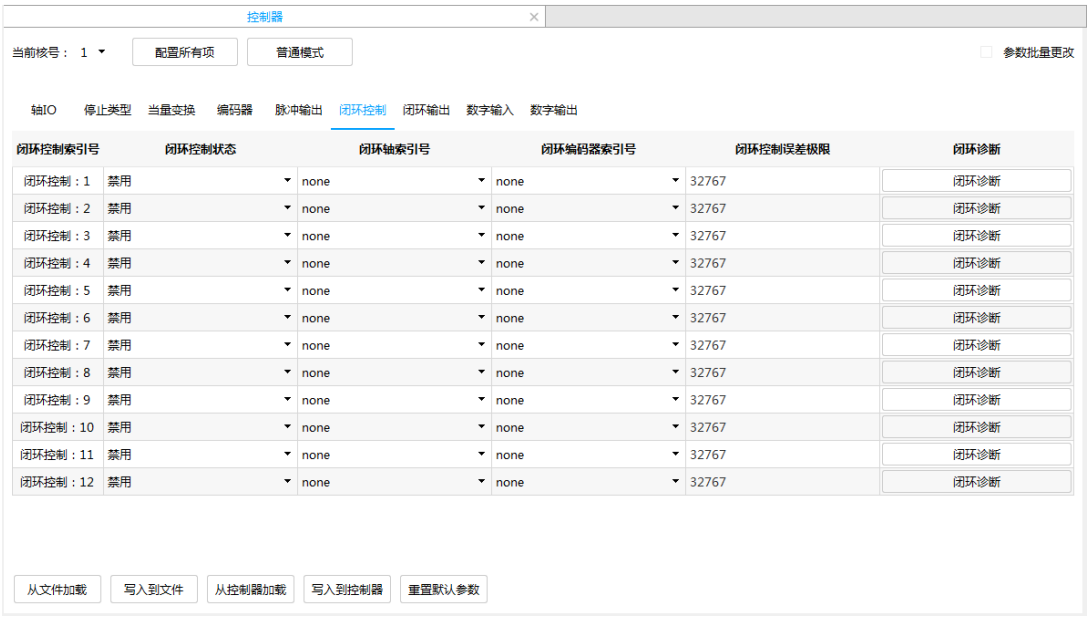


图 5-14 control 配置界面

配置说明：“闭环控制”选项主要设置闭环控制的参数。其中跟随误差表示如图 5-15 所示。

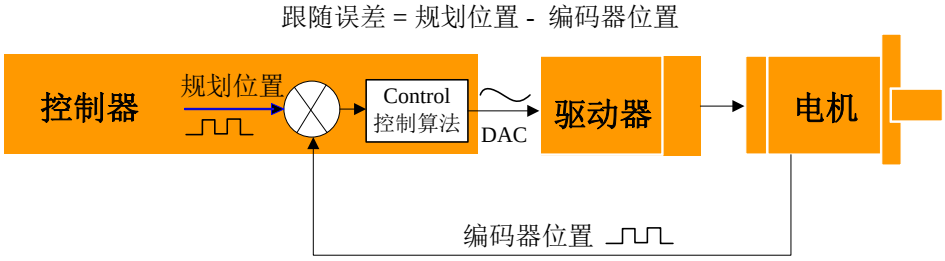


图 5-15 跟随误差解释图

当需要将相应轴设置成闭环（模拟量）控制方式时，必须将“闭环控制”配置选项中的“闭环控制状态”设置为”启用”。默认情况下是开环控制（“闭环控制状态”选择”禁用”），即开环（脉冲）控制方式。

1. 闭环控制索引号：需要进行配置的 control 的编号，不可编辑。
2. 闭环控制误差极限：表示规划位置和实际位置的误差的极限。当跟随误差超过设定的极限时，自动关闭 axis 的驱动器使能信号。默认值为：32767，单位：pulse。该项可以通过指令 GTN_SetPosErr 来设置。
3. 闭环轴索引号：如果需要伺服闭环控制模式，应该使 control 与 axis 关联。
4. 注意：默认为”none”，即开环（脉冲）控制方式。关联之后，运动控制器会将相应编号的 encoder、dac、axis、control 关联在一起，如图 5-14 所示，此时，dac 的值将不能独立设置，如果调用 GTN_SetDac 指令将会无效。切换开环方式和闭环方式可以通过指令 GTN_CtrlMode 来实现。

5.3.6 配置 di

Di 配置界面如图 5-16 所示。

图 5-16 di 配置界面

1. 输入信号类型：需要配置的 di 的类型，包括：伺服报警、正限位、负限位、原点、到位、通用输入。
2. 输入信号索引：选择需要配置的 di 的编号。
3. 输入信号反转：表示 di 的输入逻辑取反。默认情况下，0 表示输入低电平，1 表示输入高电平；输入反转后，0 表示输入高电平，1 表示输入低电平。该项可以通过指令 GTN_SetSense 设置。
4. 输入信号滤波：表示 di 输入信号维持设定时间才有效，默认滤波时间为 3，单位：伺服周期。

注意	设置该滤波参数需根据具体情况进行调整，比如，若信号维持时间很短，但又需要获取到，此时需要将滤波时间设置 0。

5.3.7 配置 do

Do 配置界面如图 5-17 所示。

图 5-17 do 配置界面

1. Do 类型选择：选择需要配置的 do 的类型，包括：伺服使能、报警清除、通用输出。

注意	报警清除信号并不是所有驱动器上都有的。即使驱动器上有此信号，此信号也不能清除所有的驱动报警信号。例如，在编码器接线不良引起驱动报警时，就无法使用清除报警信号来清除。用户可以调用指令 GTN_SetDo 来发出清除报警信号，但是当报警无法清除时，用户需要查询驱动器所提示的报警信号，然后查阅驱动器相关手册，来确定驱动器报警信号的清除方法。

2. 输出信号索引：需要配置的 do 的编号，不可编辑。

3. 输出信号反转：表示 do 的输出逻辑取反。默认值为：正常，0 表示输出低电平，1 表示输出高电平。输出反转后，0 表示输出高电平，1 表示输出低电平。

输出信号关联轴：表示该 do 关联到指定 axis 的驱动器使能。默认情况下，各轴都具有独立的驱动器使能 do 作为输出，当调用 GTN_AxisOn 指令时，该 do 置 1。如果驱动器是低电平使能，必须把“输出反转”设置为取反。

注意	Do 一旦和 axis 关联，就不能调用 GTN_SetDo 或 GTN_SetDoBit 指令直接设置伺服使能输出电平。如果不需要驱动器使能信号，可以取消和驱动器使能 do 的关联。取消关联以后，驱动器使能 do 就可以作为普通 do 来使用，可以调用 GTN_SetDo 或 GTN_SetDoBit 直接设置其输出电平。

5.4 配置文件生成和下载

表 5-1 下载配置文件指令

指令	说明	页码
GTN_LoadConfig	下载配置信息到运动控制器，调用该指令后需再调用 GTN_ClrSts 才能使该指令生效。	224

按照 5.3 节中的描述进行运动控制器的配置之后，如图 5-18 所示，在“控制器配置”界面中的正下方，点击“写入到文件”，即可对配置信息进行保存，生成配置文件（*.cfg）。

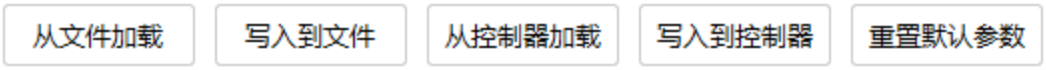


图 5-18 生成配置文件界面

用户可以调用 GTN_LoadConfig 指令将配置文件里的配置信息下载到运动控制器中，需要注意的是，如果配置文件与可执行文件不在同一目录下，在调用 GTN_LoadConfig 指令时，参数需要包含配置文件的绝对路径。

注意	GTN_LoadConfig 中的文件名长度不允许超过 125 个字，否则调用指令将会失败。

5.5 使用 MotionStudio 系统配置

本节将以 5.2.4 节的两个例子为基础，介绍在使用 MS 配置的流程。

注意	在调试阶段会频繁使用本流程。提醒用户，调试时请不要将电机与机械本体连接，否则可能会因操作失误而造成机械本体的严重损坏。

5.5.1 开环控制模式

1. 首先，通过 MS 调试软件生成配置文件(*.cfg)，具体操作请参考《MotioStudio 软件使用帮助》中的“怎样将控制器配置成开环模式？”。
2. 用户调用 GTN_LoadConfig 指令将配置文件里的配置信息下载到运动控制器中，需要注意的是，如果配置文件与可执行文件不在同一目录下，在调用 GTN_LoadConfig 指令时，参数需要包含配置文件的绝对路径。此时控制器配置完成。

5.5.2 闭环控制模式

1. 首先，通过 MS 调试软件生成配置文件(*.cfg)，具体操作请参考《MotioStudio 软件使用帮助》中的“怎样将控制器配置成闭环模式？”。
2. 用户调用 GTN_LoadConfig 指令将配置文件里的配置信息下载到运动控制器中，需要注意的是，如果配置文件与可执行文件不在同一目录下，在调用 GTN_LoadConfig 指令时，参数需要包含配置文件的绝对路径。此时控制器配置完成。

5.6 配置信息修改指令

用户除了可以使用上面所述的配置文件的方式实现运动控制器的初始化配置外，还可以使用指令的方式来实现初始化配置。

5.6.1 指令列表

表 5-2 配置信息修改指令列表

指令	说明	页码
GTN_AlarmOff	控制轴驱动报警信号无效	164
GTN_AlarmOn	控制轴驱动报警信号有效	164

指令	说明	页码
GTN_LmtsOnEx	控制轴限位信号有效	220
GTN_LmtsOffEx	控制轴限位信号无效	219
GTN_ProfileScale	设置控制轴的规划器当量变换值	231
GTN_EncScale	设置控制轴的编码器当量变换值	181
GTN_StepDir	将脉冲输出通道的脉冲输出模式设置为“脉冲+方向”	256
GTN_StepPulse	将脉冲输出通道的脉冲输出模式设置为“CCW/CW”	257
GTN_SetMtrBias	设置模拟量输出通道的零漂电压补偿值	244
GTN_GetMtrBias	读取模拟量输出通道的零漂电压补偿值	200
GTN_SetMtrLmt	设置模拟量输出通道的输出电压饱和极限值	244
GTN_GetMtrLmt	读取模拟量输出通道的输出电压饱和极限值	201
GTN_EncOn	设置为“外部编码器”计数方式	181
GTN_EncOff	设置为“脉冲计数器”计数方式	181
GTN_SetPosErr	设置跟随误差极限值	249
GTN_GetPosErr	读取跟随误差极限值	204
GTN_SetStopDec	设置平滑停止减速度和急停减速度	252
GTN_GetStopDec	读取平滑停止减速度和急停减速度	209
GTN_CtrlMode	设置控制轴为模拟量输出或脉冲输出	179
GTN_SetStopIo	设置平滑停止和紧急停止数字量输入的信息	252
GTN_SetSense	设置运动控制器数字量输入、编码器的电平逻辑	251

5.6.2 重点说明

1. 编码器计数方向设置

指令 GTN_SetSense 可以修改运动控制器各编码器的计数方向，当指令参数的某个状态位为 1 时，将所对应的控制轴的编码器计数方向取反。

例程 5-1 修改编码器计数方向

修改 core1 中的编码器 1 计数方向取反。

```
……
// 返回值变量
short sRtn;
// 设置编码器计数方向
sRtn =  GTN_SetSense (1, MC_ENCODE ,1,1);
……
```

2. 设置限位开关触发电平

运动控制器默认的限位开关为常闭开关，即各轴处于正常工作状态时，其限位开关信号输入为低电平；当限位开关信号输入为高电平时，与其对应轴的限位状态将被触发。如果使用常开开关，需要通过调用指令 GTN_SetSense 改变限位开关触发电平。

指令 GTN_SetSense 的参数设置各轴正负限位开关的触发电平，当该参数的某个状态位为 0 时，表示将对应的限位开关设置为高电平触发，当某个状态位为 1 时，表示将对应的限位开关设置为低电平触发。

```
……
// 返回值变量
```

```
short sRtn;
// 状态值变量
// 设置限位开关触发点平
sRtn = GTN_SetSense (1, MC_LIMIT_POSITIVE ,1,1);
……
```

5.6.3 例程

1. 开环（脉冲）控制模式

(1) 打开运动控制器， GTN_Open;

(2) 复位运动控制器， GTN_Reset。复位后默认的控制模式为“脉冲+方向”的脉冲控制方式。若不是采用“脉冲+方向”的控制方式，则可调用 GTN_StepPulse 修改；当需要还原为“脉冲+方向”的控制方式时，则可调用 GTN_StepDir 指令；

(3) 检查相关轴驱动器报警信号有没有连接。（一般若采用步进电机，可能没有驱动器报警信号），若没有连接，则应该调用 GTN_AlarmOff 指令，使驱动器报警无效，默认是有效的；

(4) 检查相关轴的限位开关，若没有连接，则需要通过调用 GTN_LmtsOffGTN_LmtsOffEx，使限位无效，默认是有效的；若有连接，则要检查触发电平是否设置正确，可通过 GTN_SetSense 指令修改；

(5) 在确认前面两步操作之后，调用 GTN_ClrSts，更新设置的状态；

(6) 调用 GTN_AxisOn，使能驱动器，这样相应的电机便能工作了；

(7) 使轴运动，运动后，若出现编码器位置和规划位置方向不一致，则可通过调用 GTN_SetSense 改变编码器的计数方向。

例程 5-2 设置第 1 轴为脉冲控制“脉冲+方向”方式

```
……
// 指令返回值变量
short sRtn;
// 打开运动控制器
sRtn=  GTN_Open();
// 指令返回值检测，请查阅例3-1
commandhandler(" GTN_Open", sRtn);
// 系统复位
sRtn=  GTN_Reset(1);
commandhandler(" GTN_Reset", sRtn);
// 配置轴1脉冲输出方式为脉冲+方向
sRtn=  GTN_StepDir(1,1);
commandhandler(" GTN_StepDir", sRtn);
// 配置轴1报警输出无效
sRtn=  GTN_AlarmOff(1,1);
commandhandler(" GTN_AlarmOff", sRtn);
```

```
// 配置轴1正负限位无效
sRtn= GTN_LmtsOffEx(1,1, -1, LIMIT_MODE_SOFT);
commandhandler("GTN_LmtsOffEx", sRtn);
// 配置完成后要更新状态
sRtn= GTN_ClrSts(1,1);
commandhandler(" GTN_ClrSts", sRtn);
// 轴1伺服使能
sRtn= GTN_AxisOn(1,1);
commandhandler(" GTN_AxisOn", sRtn);
……
```

2. 闭环（模拟量）控制模式

(1) 打开运动控制器， GTN_Open,

(2) 复位运动控制器， GTN_Reset。复位后的控制模式为“脉冲+方向”的脉冲控制方式，因此需要调用 GTN_CtrlMode 来修改相应轴的控制模式；

(3) 检查相关轴驱动器报警信号有没有连接。（一般若采用步进电机，可能没有驱动器报警信号），若没有连接，则应该调用 GTN_AlarmOff 指令，使驱动器报警无效，默认是有效的；

(4) 检查相关轴的限位开关，若没有连接，则需要通过调用 GTN_LmtsOffGTN_LmtsOffEx，使限位无效，默认是有效的；若有连接，则要检查触发电平是否设置正确，可通过 GTN_SetSense 指令修改；

(5) 在确认前面两步操作之后，调用 GTN_ClrSts，更新设置的状态；

(6) 设置 Pid 参数，通过调用 GTN_SetPid，将相应轴的 Pid 参数中的 Kp 设置为大于 0 的数，如 Kp=1；

(7) 调用 GTN_AxisOn，使能驱动器，若出现飞车现象（**注意:调试状态下不要接任何机械本体**），则可通过调用 GTN_SetSense 改变编码器的计数方向，最后再使能驱动器，这样相应的电机便能工作了。

例程 5-3 设置第 1 轴为闭环控制方式

```
……
// 指令返回值变量
short sRtn;
// PID结构体变量
TPid pid;
// 打开运动控制器
sRtn =  GTN_Open();
// 指令返回值检测，请查阅例3-1
commandhandler(" GTN_Open", sRtn);
// 系统复位
sRtn=  GTN_Reset(1);
commandhandler(" GTN_Reset", sRtn);
// 配置轴1为模拟量输出模式
sRtn=  GTN_CtrlMode(1,1, 0);
```

```
commandhandler(" GTN_CtrlMode", sRtn);
// 配置轴1报警输出无效
sRtn=  GTN_AlarmOff(1,1);
commandhandler(" GTN_AlarmOff", sRtn);
// 配置轴1正负限位无效
sRtn=  GTN_LmtsOffGTN_LmtsOffEx(1,1, -1, LIMIT_MODE_SOFT);
commandhandler("GTN_LmtsOffEx", sRtn);
// 配置完成后要更新状态
sRtn= GTN_ClrSts(1,1);
commandhandler(" GTN_ClrSts", sRtn);
// 获取当前pid参数
sRtn= GTN_GetPid(1,1, 1, &pid);
commandhandler(" GTN_GetPid", sRtn);
// 调试阶段，只需修改一下kp到一个较小的值
pid.kp = 1;
// 设置PID参数
sRtn=  GTN_SetPid(1,1, 1, &pid);
commandhandler(" GTN_SetPid", sRtn);
// 轴1伺服使能
sRtn= GTN_AxisOn(1,1);
commandhandler(" GTN_AxisOn", sRtn);
// 伺服使能后若出现飞车，则需要通过调用 GTN_SetSense 指令来修改编码器计数方向
// 等待伺服稳定
Sleep(200);
……
```

通过指令配置控制器之后，用户必须调用指令 GTN_ClrSts 更新状态，使得配置生效。很多初次使用的客户会容易忘记这一点。

5.7 控制器配置初始化状态

控制器初始化状态，是指调用指令 GTN_Open 成功后，调用指令 GTN_Reset 复位后的状态。此时所有的状态都为默认状态。想要让控制器回到初始状态，只要调用 GTN_Reset 指令即可。

表 5-3 所示为复位后，控制器配置选项的默认状态，调用第四栏“相关指令”中的指令可以修改对应选项的状态。

表 5-3 控制器配置初始化状态

资源	配置选项	默认状态	相关指令
axis	该资源是否激活	启用	/
	规划器当量	Alpha 和 Beta 都为 1	GTN_ProfileScale
	编码器当量	Alpha 和 Beta 都为 1	GTN_EncScale
	平滑停止和急停数字输入信号	不配置	GTN_SetStopIo
	平滑停止减速度	1 pulse/ms^2	GTN_SetStopDec
	急停的减速度	1000 pulse/ms^2	GTN_SetStopDec

资源	配置选项	默认状态	相关指令
	正负限位输入	有效，编号与轴号对应	GTN_LmtsOnEx GTN_LmtsOffEx
	驱动报警输出	有效，编号与轴号对应	GTN_AlarmOn GTN_AlarmOff
step	该资源是否激活	启用	/
	脉冲输出方式	脉冲+方向	GTN_StepDir GTN_StepPulse
dac	该资源是否激活	禁用	-
	DAC 输出电压零漂补偿	0	GTN_SetMtrBias
	DAC 输出电压反转	正常	/
	DAC 输出电压饱和极限	32767	GTN_SetMtrLmt
encoder	该资源是否激活	激活	/
	脉冲计数源	外部编码器	GTN_EncOn GTN_EncOff
	输入脉冲反转	取反	GTN_SetSense
	Home 及 Index 捕获触发沿	下降沿	GTN_SetTriggerEx
control	该资源是否与轴关联	不关联	/
	跟随误差极限	32767	GTN_SetPosErr
di	该资源是否激活	激活	/
	限位开关	常闭开关，常为低电平，高电平限位触发	GTN_SetSense
	所有 DI 输入反转	正常	GTN_SetSense
	所有 DI 滤波时间	3 个伺服周期	/
	所有 DI 状态	常为低电平	/
do	该资源是否激活	激活	/
	所有 DO 输出反转	正常，1 代表输出高电平，0 代表输出低电平	/
	驱动报警触发电平	1 代表输出高电平，0 代表输出低电平	/

第6章 运动状态检测

6.1 本章简介

当用户连接好一整套系统后（包括运动控制器，驱动器，电机），如何查看这套系统的运动状态？控制器可以帮助用户在应用程序中查看驱动器报警，当前运动位置，运动速度和加速度等一系列状态参数。本章将介绍用户可以检测到哪些状态以及如何检测。

提示	本手册中所有字体为蓝色的指令（如 GTN_PrfTrap）均带有超级链接，点击可跳转至指令说明。

6.2 指令列表

表 6-1 运动状态检测指令列表

指令	说明	页码
GTN_GetSts	读取轴状态	210
GTN_ClrSts	清除驱动器报警标志、跟随误差越限标志、限位触发标志 1. 只有当驱动器没有报警时才能清除轴状态字的报警标志 2. 只有当跟随误差正常以后，才能清除跟随误差越限标志 3. 只有当离开限位开关，或者规划位置在软限位行程以内时才能清除轴状态字的限位触发标志	176
GTN_GetPrfSts	读取规划器状态。	206
GTN_GetPrfMode	读取轴运动模式	205
GTN_GetPrfPos	读取规划位置	206
GTN_GetPrfVel	读取规划速度	206
GTN_GetPrfAcc	读取规划加速度	205
GTN_GetAxisPrfPos	读取轴(axis)的规划位置值	187
GTN_GetAxisPrfVel	读取轴(axis)的规划速度值	187
GTN_GetAxisPrfAcc	读取轴(axis)的规划加速度值	186
GTN_GetAxisEncPos	读取轴(axis)的编码器位置值	185
GTN_GetAxisEncVel	读取轴(axis)的编码器速度值	185
GTN_GetAxisEncAcc	读取轴(axis)的编码器加速度值	185
GTN_GetAxisError	读取轴(axis)的规划位置值和编码器位置值的差值	186
GTN_Stop	停止一个或多个轴的规划运动，停止坐标系运动	257

6.3 重点说明

6.3.1 轴状态定义

用户可以从控制器的运动状态寄存器中读取轴状态。当调用 GTN_GetSts 指令时，将返回一个 32 位的轴状态字，该轴状态字的定义如表 6-2 所示。

表 6-2 轴状态定义

位	定义
0	保留
1	驱动器报警标志 控制轴连接的驱动器报警时置 1
2	保留
3	保留
4	跟随误差越限标志 控制轴规划位置和实际位置的误差大于设定极限时置 1
5	正限位触发标志 正限位开关电平状态为限位触发电平时置 1 规划位置大于正向软限位时置 1
6	负限位触发标志 负限位开关电平状态为限位触发电平时置 1 规划位置小于负向软限位时置 1
7	IO 平滑停止触发标志 如果轴设置了平滑停止 IO，当其输入为触发电平时置 1，并自动平滑停止该轴
8	IO 急停触发标志 如果轴设置了急停 IO，当其输入为触发电平时置 1，并自动急停该轴
9	电机使能标志 电机使能时置 1
10	规划运动标志 规划器运动时置 1
11	电机到位标志 规划器静止，规划位置和实际位置的误差小于设定误差带，并且在误差带内保持设定时间后，置起到位标志
12~29	保留
30	编码器线数设置标志位
31	伺服周期是否 BOOT 成功

驱动器报警标志、限位触发标志、IO 停止、跟随误差越限标志触发以后，不会自动清 0。只有当产生异常的原因已经消除以后，调用 GTN_ClrSts 指令才能清除相应的异常标志。

规划运动状态(bit10)只表示理论上的运动状态。置 1 表示处于规划运动状态，清 0 表示处于规划静止状态。由于电机跟随滞后、机械系统震荡等原因，一般在规划静止一段时间以后，机械系统才能完全停止。

电机到位标志(bit11)表示实际到位状态。该标志位是电机到位检测功能的一部分。控制器复位后，默认该功能未开启。若要开启该功能，请参考 12.8 节的详细说明。该标志位置 1 表示已经处于规划静止状态(bit10=0)，并且规划位置和编码器位置的误差在设定的误差带内保持了设定时间。当规划运动或者规划位置和编码器位置的误差超出误差带时立即清 0。检测电机到位标志可以保证系统的定位精度，应当根据机械系统的实际情况设置合适的到位误差带和误差带保持时间。如果到位误差带设置的太小，或者误差带保持时间太长，都会使到位时间增长，影响加工效率。

6.3.2 规划器状态定义

用户可以从控制器的运动状态寄存器中读取规划器状态。当调用指令 GTN_GetPrfSts 时，将返回一个 32 位的规划器状态字，该轴状态字的定义如表 6-3 所示。

表 6-3 规划器状态定义

位	定义
0	保留
1	驱动器报警标志 控制轴连接的驱动器报警时置 1
2	保留
3	保留
4	保留
5	正限位触发标志 正限位开关电平状态为限位触发电平时置 1 规划位置大于正向软限位时置 1
6	负限位触发标志 负限位开关电平状态为限位触发电平时置 1 规划位置小于负向软限位时置 1
7	IO 平滑停止触发标志 如果轴设置了平滑停止 IO，当其输入为触发电平时置 1，并自动平滑停止该轴
8	IO 急停触发标志 如果轴设置了急停 IO，当其输入为触发电平时置 1，并自动急停该轴
9	保留
10	规划运动标志 规划器运动时置 1
11-29	保留
30	编码器线数设置标志位
31	伺服周期是否 BOOT 成功

6.3.3 轴的运动参数

调用 GTN_GetPrfMode 可以读取当前轴的运动模式。运动模式将在“**第 7 章 运动模式**”中详细介绍。控制器有如下几种运动模式：点位运动模式、Jog 模式、PT 模式、电子齿轮模式、Follow 模式、插补运动模式、PVT 模式，其中 PT 模式、Follow 模式和 PVT 模式详见《运动控制器编程手册之高级功能》。

“**5.2.3 软件资源**”中介绍过规划器的概念。规划器是位于控制器内部的，是根据用户所设置的运动模式和运动参数计算规划位置和规划速度的软件资源。它不代表电机系统的位置和速度。调用 GTN_GetPrfPos， GTN_GetPrfVel， GTN_GetPrfAcc 可以读取规划器当前的位置，速度和加速度。

“**5.2.3 软件资源**”中介绍过轴的概念。轴亦是位于控制器内部的，是将规划器和编码器硬件资源整合起来的软件资源。轴的规划位置是规划器位置经过当量变换后的值，轴的编码器位置也是编码器硬件的位置经过当量变换后的值。默认情况下，规划器位置的当量与轴的规划位置的当量之比是 1:1，轴的编码器位置当量与编码器硬件的位置当量之比也是 1:1。调用 GTN_GetAxisPrfPos， GTN_GetAxisPrfVel，GTN_GetAxisPrfAcc 可以读取轴的规划器当前的位置，速度和加速度。调用 GTN_GetAxisEncPos，GTN_GetAxisEncVel， GTN_GetAxisEncAcc 可以读取轴的编码器当前的位置，速度和加速度。调用 GTN_GetAxisError 读取轴的规划位置和轴的编码器位置的差值。

调用 GTN_Stop 停止正在运动的轴。停止方式可以分为急停和平滑停止。具体介绍请参考 10.4 。

6.4 例程

例程 6-1 获取轴 1 的轴状态、运动模式、位置、速度和加速度

```
#define AXIS                    1               // 定义轴号
int main(int argc, char* argv[])
{
    short bFlagAlarm = FALSE;       // 伺服报警标志
    short bFlagMError = FALSE;      // 跟随误差越限标志
    short bFlagPosLimit = FALSE;        // 正限位触发标志
    short bFlagNegLimit = FALSE;        // 负限位触发标志
    short bFlagSmoothStop = FALSE;// 平滑停止标志
    short bFlagAbruptStop = FALSE; // 急停标志
    short bFlagServoOn = FALSE;         // 伺服使能标志
    short bFlagMotion = FALSE;      // 规划器运动标志
    double dPrfPos;                     // 规划位置
    double dPrfVel;                     // 规划速度
    double dPrfAcc;                     // 规划加速度
    long lPrfMode;                      // 运动模式
    char chPrfMode[20];                 // 运动模式，字符串变量
    short sRtn;                         // 指令返回值变量
    long lAxisStatus;                   // 轴状态
    short core=1;
    // 打开运动控制器
    sRtn =   GTN_Open();
    // 指令返回值检测，请查阅例3-1
    commandhandler(" GTN_Open", sRtn);
    // 系统复位
    sRtn =   GTN_Reset(1);
    commandhandler(" GTN_Reset", sRtn);
    // 读取轴状态
    sRtn =   GTN_GetSts(1,AXIS, &lAxisStatus);
    commandhandler(" GTN_GetSts", sRtn);

    // 伺服报警标志
    if (lAxisStatus& 0x2)
    {
        bFlagAlarm = TRUE;
        printf("伺服报警\n");
    }
    else
    {
        bFlagAlarm = FALSE;
        printf("伺服正常\n");
    }
    // 跟随误差越限标志
```

```
if (lAxisStatus& 0x10)
{
    bFlagMError = TRUE;
    printf("运动出错\n");
}
else
{
    bFlagMError = FALSE;
    printf("运动正常\n");
}
// 正向限位
if (lAxisStatus& 0x20)
{
    bFlagPosLimit = TRUE;
    printf("正限位触发\n");
}
else
{
    bFlagPosLimit = FALSE;
    printf("正限位未触发\n");
}
// 负向限位
if (lAxisStatus& 0x40)
{
    bFlagNegLimit = TRUE;
    printf("负限位触发\n");
}
else
{
    bFlagNegLimit = FALSE;
    printf("负限位未触发\n");
}
// 平滑停止
if (lAxisStatus& 0x80)
{
    bFlagSmoothStop = TRUE;
    printf("平滑停止触发\n");
}
else
{
    bFlagSmoothStop = FALSE;
    printf("平滑停止未触发\n");
}
// 急停标志
if (lAxisStatus& 0x100)
{
```

```
        bFlagAbruptStop = TRUE;
        printf("急停触发\n");
    }
    else
    {
        bFlagAbruptStop = FALSE;
        printf("急停未触发\n");
    }
    // 伺服使能标志
    if(lAxisStatus& 0x200)
    {
        bFlagServoOn = TRUE;
        printf("伺服使能\n");
    }
    else
    {
        bFlagServoOn = FALSE;
        printf("伺服关闭\n");
    }
    // 规划器正在运动标志
    if (lAxisStatus& 0x400)
    {
        bFlagMotion = TRUE;
        printf("规划器正在运动\n");
    }
    else
    {
        bFlagMotion = FALSE;
        printf("规划器已停止\n");
    }
    // 读取运动数据
    sRtn =  GTN_GetPrfPos(1,AXIS, &dPrfPos);
    commandhandler(" GTN_GetPrfPos", sRtn);
    printf("规划位置  %8.2f\n", dPrfPos);
    sRtn =  GTN_GetPrfVel(1,AXIS, &dPrfVel);
    commandhandler(" GTN_GetPrfVel", sRtn);
    printf("规划速度  %8.2f\n", dPrfVel);
    sRtn= GTN_GetPrfAcc(1,AXIS, &dPrfAcc);
    commandhandler(" GTN_GetPrfAcc", sRtn);
    printf("规划加速度  %8.2f\n", dPrfAcc);

    // 读取运动模式
    sRtn =  GTN_GetPrfMode(1,AXIS, &lPrfMode);
    commandhandler(" GTN_GetPrfMode", sRtn);
    // 清空字符串
    memset( chPrfMode, '\0', 20);
```

```
    switch(lPrfMode)
    {
        case 0:
            sprintf(chPrfMode, "Trap");
        break;
        case 1:
            sprintf(chPrfMode, "Jog");
        break;
        case 2:
            sprintf(chPrfMode, "PT");
        break;
        case 3:
            sprintf(chPrfMode, "Gear");
        break;
        case 4:
            sprintf(chPrfMode, "Follow");
        break;
        case 5:
            sprintf(chPrfMode, "Interpolation");
        break;
        case 6:
            sprintf(chPrfMode, "PVT");
        break;
        default:
        break;
    }
    printf("运动模式  %s\n", chPrfMode);
    returnTRUE;
}
```

第7章 运动模式

7.1 本章简介

运动模式是指规划一个或多个轴运动的方式。运动控制器支持的运动模式有点位运动模式、Jog 运动模式、电子齿轮（即 Gear）运动模式和插补运动模式。

提示	本手册中所有字体为蓝色的指令（如 GTN_PrfTrap）均带有超级链接，点击可跳转至指令说明。

注意	用户需注意，每一个轴在任一时刻只能处在一种运动模式下。

表 7-1 设置运动模式指令列表

指令	说明	页码
GTN_PrfTrap	设置指定轴为点位模式	230
GTN_PrfJog	设置指定轴为 Jog 模式	230
GTN_PrfGear	设置指定轴为电子齿轮模式	230
GTN_GetPrfMode	查询指定轴的运动模式	205

注意	在设置或切换运动模式时，要保证轴处于规划静止状态。如果轴正在运动，请先调用 GTN_Stop 指令停止一个或多个轴的运动，然后再调用上表中的指令切换到想要的运动模式下。

7.2 点位运动模式

7.2.1 指令列表

表 7-2 点位运动模式指令列表

指令	说明	页码
GTN_PrfTrap	设置指定轴为点位运动模式	230
GTN_SetTrapPrm	设置点位运动模式下的运动参数	254
GTN_GetTrapPrm	读取点位运动模式下的运动参数	213
GTN_SetPos	设置目标位置	246
GTN_GetPos	读取目标位置	202
GTN_SetVel	设置目标速度	256
GTN_GetVel	读取目标速度	216
GTN_Update	启动点位运动	258

7.2.2 重点说明

每一个轴在规划静止时都可以设置为点位运动。在点位运动模式下，各轴可以独立设置目标位置、目标速度、加速度、减速度、起跳速度、平滑时间等运动参数，能够独立运动或停止。

调用 GTN_Update 指令启动点位运动以后，控制器根据设定的运动参数自动生成相应的梯形曲线速度规划，并且在运动过程中可以随时修改目标位置和目标速度。

设定平滑时间能够得到平滑的速度曲线，从而使加减速过程更加平稳，如图 7-1 所示。

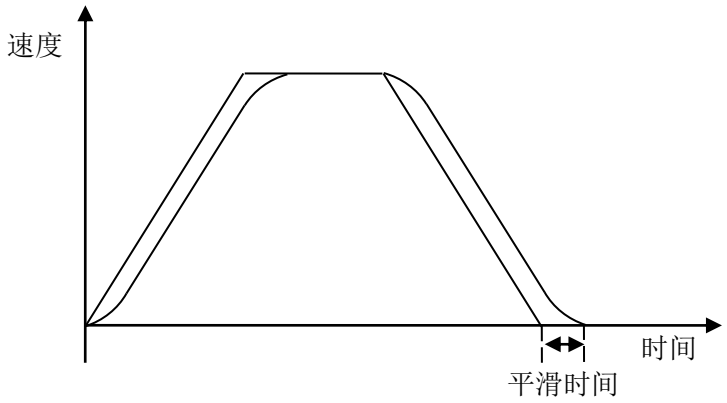


图 7-1　点位运动速度曲线

平滑时间是指加速度的变化时间，单位：ms，取值范围：[0, 50]。

7.2.3 例程

例程 7-1　点位运动

将第 1 轴设定为点位运动模式，并且以速度 50pulse/ms，加速度 0.25pulse/ms^2，减速度 0.125pulse/ms^2，平滑时间为 25ms 的运动参数正向运动 50000 个脉冲。

该例程生成一段梯形曲线速度规划，如图 7-2 所示。

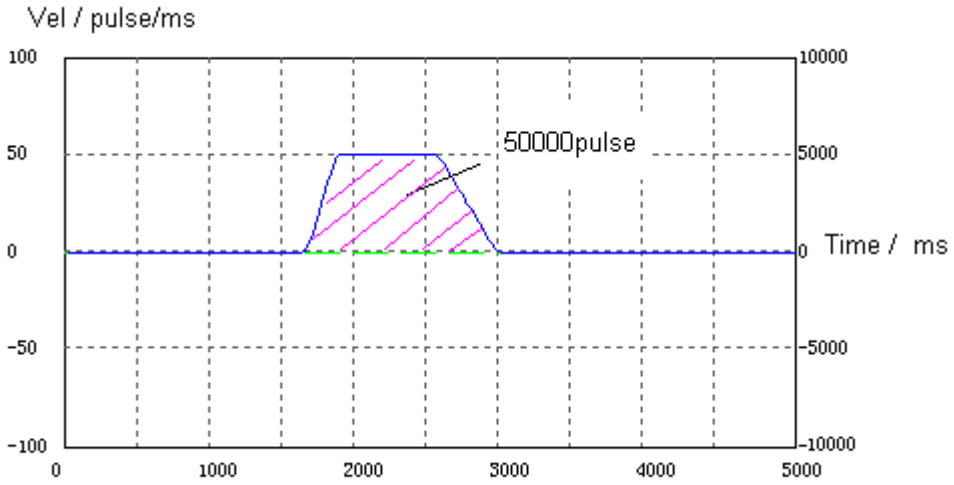


图 7-2　点位运动速度规划

```
#include "stdafx.h"
```

```
#include "conio.h"
#include "gts.h"

// 定义轴号
#define AXIS        1

int main (int argc, char* argv[])
{
    short sRtn;
    TTrapPrm trap;
    long sts;
    double prfPos;

    // 打开运动控制器
sRtn = GTN_Open();
    // 指令返回值检测，请查阅例3-1
commandhandler(" GTN_Open", sRtn);
// 复位运动控制器
sRtn =  GTN_Reset(1);
commandhandler(" GTN_Reset", sRtn);
    // 配置运动控制器
    // 注意：配置文件取消了各轴的报警和限位
sRtn =  GTN_LoadConfig(1,"test.cfg");
commandhandler(" GTN_LoadConfig", sRtn);
    // 清除各轴的报警和限位
sRtn = GTN_ClrSts(1,1, 8);
commandhandler(" GTN_ClrSts", sRtn);
    // 伺服使能
sRtn = GTN_AxisOn(1,AXIS);
commandhandler(" GTN_AxisOn", sRtn);

    // 位置清零
sRtn =  GTN_ZeroPos(1,AXIS);
commandhandler(" GTN_ZeroPos", sRtn);
    // AXIS轴规划位置清零
sRtn =  GTN_SetPrfPos(1,AXIS, 0);
commandhandler(" GTN_SetPrfPos", sRtn);
// 将 AXIS 轴设为点位模式
sRtn =  GTN_PrfTrap(1,AXIS);
commandhandler(" GTN_PrfTrap", sRtn);
// 读取点位运动参数(需要读取所有运动参数到上位机变量)
sRtn =  GTN_GetTrapPrm(1,AXIS, &trap);
commandhandler(" GTN_GetTrapPrm", sRtn);
// 设置需要修改的运动参数
trap.acc = 0.25;
trap.dec = 0.125;
```

```
trap.smoothTime = 25;
// 设置点位运动参数
sRtn =  GTN_SetTrapPrm(1,AXIS, &trap);
commandhandler(" GTN_SetTrapPrm", sRtn);
// 设置 AXIS 轴的目标位置
sRtn =  GTN_SetPos(1,AXIS, 50000L);
commandhandler(" GTN_SetPos", sRtn);
    // 设置AXIS轴的目标速度
sRtn =  GTN_SetVel(1,AXIS, 50);
commandhandler(" GTN_SetVel", sRtn);
    // 启动AXIS轴的运动
sRtn =  GTN_Update(1,1<<(AXIS-1));
commandhandler(" GTN_Update", sRtn);

    do
    {
        // 读取AXIS轴的状态
        sRtn =  GTN_GetSts(1,AXIS, &sts);
        // 读取AXIS轴的规划位置
        sRtn =  GTN_GetPrfPos(1,AXIS, &prfPos);
        printf("sts=0x%-10lxprfPos=%-10.1lf\r", sts, prfPos);
    }while(sts&0x400);// 等待AXIS轴规划停止

// 伺服关闭
    sRtn =  GTN_AxisOff(1,AXIS);
    printf("\n GTN_AxisOff()=%d\n", sRtn);
    getch();
    return 0;
}
```

7.3 Jog 运动模式

7.3.1 指令列表

表 7-3 Jog 运动模式指令列表

指令	说明	页码
GTN_PrfJog	设置指定轴为 Jog 运动模式	230
GTN_SetJogPrm	设置 Jog 运动模式下的运动参数	243
GTN_GetJogPrm	读取 Jog 运动模式下的运动参数	200
GTN_SetVel	设置目标速度	256
GTN_GetVel	读取目标速度	216
GTN_Update	启动 Jog 运动	258

7.3.2 重点说明

在 Jog 运动模式下，各轴可以独立设置目标速度、加速度、减速度、平滑系数等运动参数，能够独立运动或停止。

调用 GTN_Update 指令启动 Jog 运动以后，按照设定的加速度加速到目标速度后保持匀速运动，在运动过程中可以随时修改目标速度，如图 7-3 所示。

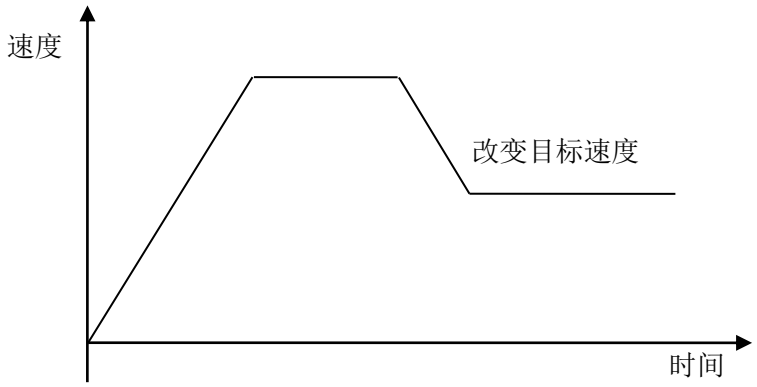


图 7-3 Jog 模式速度曲线

设定平滑系数能够得到平滑的速度曲线，从而使加减速过程更加平稳。平滑系数的取值范围是[0, 1)，越接近 1，加速度变化越平稳。

7.3.3 例程

例程 7-2 Jog 运动

轴 1 运动在 Jog 模式下，初始目标速度为 100pulse/ms。动态改变目标速度，当规划位置超过 100000pulse 时，修改目标速度为 50 pulse/ms。如图 7-4 所示。

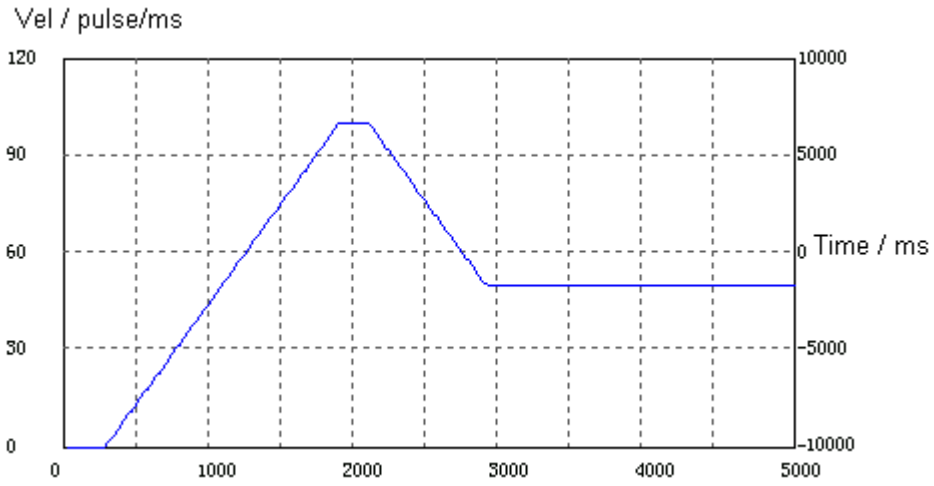


图 7-4 Jog 模式动态改变目标速度

```
#include "stdafx.h"
#include "conio.h"
#include "gts.h"

// 定义轴号
```

```
#define AXIS                    1

int main(int argc, char* argv[])
{
    short sRtn;
    TJogPrm jog;
    long sts;
    double prfPos, prfVel;

// 打开运动控制器
sRtn = GTN_Open();
// 指令返回值检测，请查阅例程 3-1
commandhandler(" GTN_Open", sRtn);
// 复位运动控制器
sRtn =  GTN_Reset(1);
commandhandler(" GTN_Reset", sRtn);
// 配置运动控制器
// 注意：配置文件取消了各轴的报警和限位
sRtn =  GTN_LoadConfig(1,"test.cfg");
commandhandler(" GTN_LoadConfig", sRtn);
// 清除各轴的报警和限位
sRtn = GTN_ClrSts(1,1, 8);
commandhandler(" GTN_ClrSts", sRtn);
// 伺服使能
sRtn = GTN_AxisOn(1,AXIS);
commandhandler(" GTN_AxisOn", sRtn);
// 位置清零
sRtn = GTN_ZeroPos(1,AXIS);
commandhandler(" GTN_ZeroPos", sRtn);
// 将 AXIS 轴设为 Jog 模式
sRtn = GTN_PrfJog(1,AXIS);
commandhandler(" GTN_PrfJog", sRtn);
// 读取 Jog 运动参数(需要读取全部运动参数到上位机变量)
sRtn = GTN_GetJogPrm(1,AXIS, &jog);
commandhandler(" GTN_GetJogPrm", sRtn);
//设置需要修改的运动参数
jog.acc = 0.0625;
jog.dec = 0.0625;
// 设置 Jog 运动参数
sRtn = GTN_SetJogPrm(1,AXIS, &jog);
commandhandler(" GTN_SetJogPrm", sRtn);
// 设置 AXIS 轴的目标速度
sRtn = GTN_SetVel(1,AXIS, 100);
commandhandler(" GTN_SetVel", sRtn);
// 启动 AXIS 轴的运动
sRtn = GTN_Update(1,1<<(AXIS-1));
```

```
commandhandler(" GTN_Update", sRtn);

    while(1)
    {
        // 读取AXIS轴的状态
        sRtn = GTN_GetSts(1,AXIS, &sts);
        // 读取AXIS轴的规划位置
        sRtn = GTN_GetPrfPos(1,AXIS, &prfPos);
        // 读取AXIS轴的规划速度
        sRtn = GTN_GetPrfVel(1,AXIS, &prfVel);
        printf("sts=0x%-10lxprfPos=%-10.1lfprfVel=%-10.1lf\r", sts, prfPos, prfVel);
        if(prfPos>= 100000)
        {
            // 设置AXIS轴新的目标速度
            sRtn = GTN_SetVel(1,AXIS, 50);
            commandhandler(" GTN_SetVel", sRtn);
            // AXIS轴新的目标速度生效
            sRtn = GTN_Update (1,1<<(AXIS-1));
            commandhandler(" GTN_Update", sRtn);
            break;
        }
    }
    while(!kbhit())
    {
        // 读取AXIS轴的状态
        sRtn = GTN_GetSts(1,AXIS, &sts);
        // 读取AXIS轴的规划位置
        sRtn = GTN_GetPrfPos(1,AXIS, &prfPos);
        // 读取AXIS轴的规划速度
        sRtn = GTN_GetPrfVel(1,AXIS, &prfVel);
        printf("sts=0x%-10lxprfPos=%-10.1lfprfVel=%-10.1lf\r", sts, prfPos, prfVel);
    }
// 伺服关闭
    sRtn = GTN_AxisOff (1,AXIS);
    printf("\n GTN_AxisOff()=%d\n", sRtn);
    getch();
    return 0;
}
```

7.4 电子齿轮（Gear）运动模式

7.4.1 指令列表

表 7-4 电子齿轮运动模式指令列表

指令	说明	页码
GTN_PrfGear	设置指定轴为电子齿轮运动模式	230
GTN_SetGearMaster	设置电子齿轮运动跟随主轴	241
GTN_GetGearMaster	读取电子齿轮运动跟随主轴	197
GTN_SetGearRatio	设置电子齿轮比	243
GTN_GetGearRatio	读取电子齿轮比	198
GTN_GearStart	启动电子齿轮运动	182

7.4.2 重点说明

电子齿轮模式能够将两轴或多轴联系起来，实现精确的同步运动，从而替代传统的机械齿轮连接。

我们把被跟随的轴叫主轴，把跟随的轴叫从轴。电子齿轮模式下，1 个主轴能够驱动多个从轴，从轴可以跟随主轴的规划位置、编码器位置。

传动比：主轴速度与从轴速度的比例。电子齿轮模式能够灵活的设置传动比，节省机械系统的安装时间。当主轴速度变化时，从轴会根据设定好的传动比自动改变速度。电子齿轮模式也能够在运动过程中修改传动比。

离合区：当改变传动比时，可以设置离合区，实现平滑变速，如图所示，阴影区域为离合区。电子齿轮模式速度曲线如图 7-5 所示。

离合区位移是指从轴平滑变速过程中主轴运动的位移。不要计算成从轴变速时走过的位移

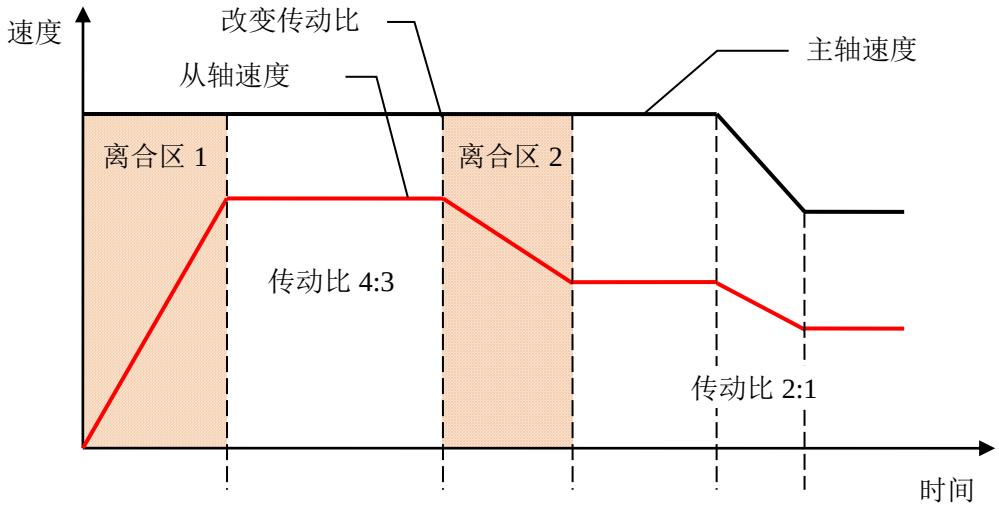


图 7-5 电子齿轮模式速度曲线

主轴匀速运动，从轴为电子齿轮模式。在离合区 1 从轴速度从 0 逐渐增大，直到到达传动比 4:3。当改变传动比至 2:1 时，在离合区 2 从轴速度逐渐变化直到满足新的传动比。离合区越大，从轴传动比的变

化过程越平稳。当主轴速度变化时，从轴速度也随着变化，保持固定的传动比。

(1) 如何切换到电子齿轮模式？

用户必须要调用 GTN_PrfGear，才能将指定轴设定为 Gear 模式。应将从轴设定为 Gear 模式。

(2) 如何设置主轴？

调用 GTN_SetGearMaster。参数 profile 为从轴轴号，参数 mastIndex 为主轴轴号。

注意	1、为了减少跟随滞后，从轴的轴号应当大于主轴的轴号。 2、默认情况下，主轴和跟随轴必须在同一 core

(3) 如何设定传动比和离合区？

1) 调用 GTN_SetGearRatio 指令来设置传动比和离合区。profile 是从轴轴号；masterEven 是主轴位移，slaveEven 是从轴位移，masterEven/slaveEven 等于传动比；slope 是主轴离合区位移，即主轴在离合区内走过的位移，用户应自行计算出来。

2) 如果从轴轴号为 slave，当主轴位移 alpha 时从轴位移 beta，主轴运动 slope 位移后从轴到达设定传动比，应当调用指令 GTN_SetGearRatio。

(4) 如何在不同 core 实现跟随

GTN 系列产品支持主轴和从轴在不同 core，但是需要将跟随轴所在 core 工作模式设置为协同模式并且将主轴相关的参数（规划位置或者编码器位置）共享。详细步骤如下所示：

1) 设置核 core 的工作模式；

2) 共享数据，具体操作请联系我司工程师。

7.4.3 例程

例程 7-3 电子齿轮跟随

主轴为 Jog 模式，从轴为电子齿轮模式，传动比为主轴速度：从轴速度=2:1，主轴运动离合区位移后（图中阴影部分的区域），从轴达到设定的传动比，如图 7-6 和图 7-7 所示。

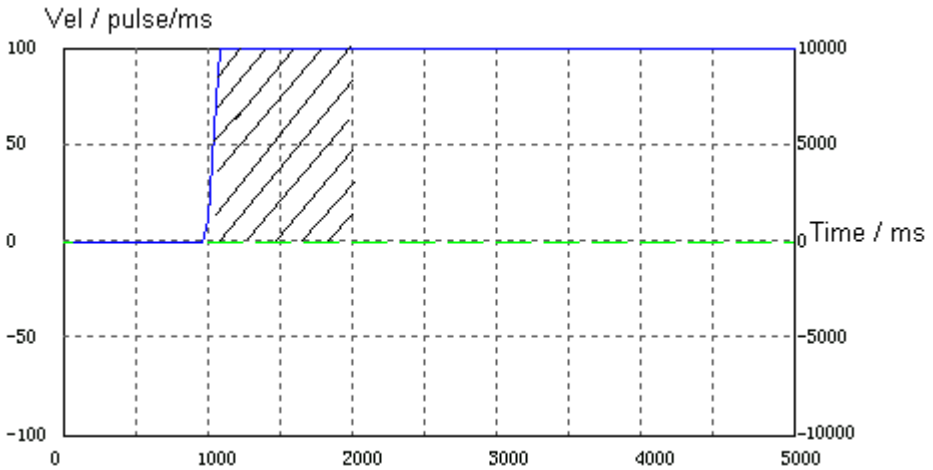


图 7-6 电子齿轮模式主轴速度规划

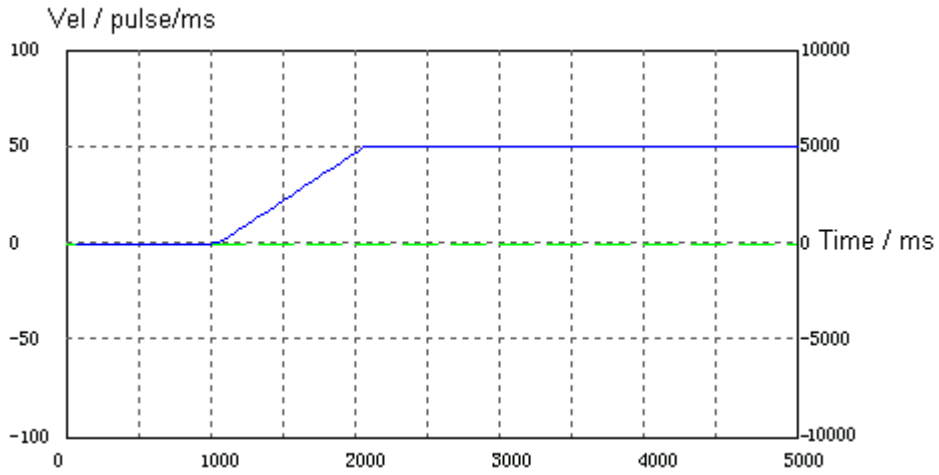


图 7-7　电子齿轮模式从轴速度规划

```
#include "stdafx.h"
#include "conio.h"
#include "gts.h"

#define MASTER              1
#define SLAVE               2
int main(int argc, char* argv[])
{
    short sRtn;
    double prfVel[8];
    TJogPrm jog;

    // 打开运动控制器
    sRtn = GTN_Open();
    // 指令返回值检测，请查阅例3-1
    commandhandler(" GTN_Open", sRtn);
    // 复位运动控制器
    sRtn =  GTN_Reset(1);
    commandhandler(" GTN_Reset", sRtn);
    // 配置运动控制器
    // 注意：配置文件 test.cfg 取消了各轴的报警和限位
    sRtn =  GTN_LoadConfig(1,"test.cfg");
    commandhandler(" GTN_LoadConfig", sRtn);
    // 清除各轴的报警和限位
    sRtn = GTN_ClrSts(1,1, 8);
    commandhandler(" GTN_ClrSts", sRtn);
    // 伺服使能
    sRtn =  GTN_AxisOn(1,MASTER);
    commandhandler(" GTN_AxisOn", sRtn);
    sRtn =  GTN_AxisOn(1,SLAVE);
    commandhandler(" GTN_AxisOn", sRtn);
    // 位置清零
```

```
    sRtn = GTN_ZeroPos(1,MASTER);
    commandhandler(" GTN_ZeroPos", sRtn);
    sRtn = GTN_ZeroPos (1,SLAVE);
    commandhandler(" GTN_ZeroPos", sRtn);
    // 将主轴设为 Jog 模式
    sRtn = GTN_PrfJog(1,MASTER);
    commandhandler(" GTN_PrfJog", sRtn);
    // 设置主轴运动参数
    sRtn = GTN_GetJogPrm(1,MASTER, &jog);
    commandhandler(" GTN_GetJogPrm", sRtn);
    jog.acc = 1;
    sRtn =   GTN_SetJogPrm(1,MASTER, &jog);
    commandhandler(" GTN_SetJogPrm", sRtn);
    sRtn = GTN_SetVel(1,MASTER, 100);
    commandhandler(" GTN_SetVel", sRtn);
    // 启动主轴
    sRtn = GTN_Update(1,1<<(MASTER-1));
    commandhandler(" GTN_Update", sRtn);
    // 将从轴设为 Gear 模式
    sRtn = GTN_PrfGear(1,SLAVE);
    commandhandler(" GTN_PrfGear", sRtn);
    // 设置主轴，默认跟随主轴规划位置
    sRtn = GTN_SetGearMaster(1,SLAVE, MASTER);
    commandhandler(" GTN_SetGearMaster", sRtn);
    // 设置从轴的传动比和离合区
    sRtn = GTN_SetGearRatio(1,SLAVE, 2, 1, 100000);
    commandhandler(" GTN_SetGearRatio", sRtn);
    // 启动从轴
    sRtn = GTN_GearStart(1,1<<(SLAVE-1));
    commandhandler(" GTN_GearStart",sRtn);

    while(!kbhit())
    {
        // 查询各轴的规划速度
        sRtn = GTN_GetPrfVel(1,1, prfVel, 8);
        printf("master vel=%-10.2lf\tslave vel=%-10.2lf\r",
        prfVel[MASTER-1], prfVel[SLAVE-1]);
    }

// 伺服关闭
    sRtn = GTN_AxisOff (1,MASTER);
    printf("\n GTN_AxisOff()=%d, Axis:%d\n", sRtn, MASTER);
    sRtn =   GTN_AxisOff (1,SLAVE);
    printf("\n GTN_AxisOff()=%d, Axis:%d\n", sRtn, SLAVE);
    getch();
    return 0;
```

}

7.5 插补运动模式

7.5.1 指令列表

表 7-5 插补运动模式指令列表

指令	说明	页码
GTN_SetCrdPrm	设置坐标系参数，确立坐标系映射，建立坐标系	235
GTN_GetCrdPrm	查询坐标系参数	191
GTN_CrdData	向插补缓存区增加插补数据	177
GTN_LnXY	缓存区指令，二维直线插补	220
GTN_LnXYZ	缓存区指令，三维直线插补	221
GTN_LnXYZA	缓存区指令，四维直线插补	222
GTN_LnXYG0	缓存区指令，二维直线插补(终点速度始终为 0)	221
GTN_LnXYZG0	缓存区指令，三维直线插补(终点速度始终为 0)	223
GTN_LnXYZAG0	缓存区指令，四维直线插补(终点速度始终为 0)	223
GTN_ArcXYR	缓存区指令，XY 平面圆弧插补(以终点位置和半径为输入参数)	165
GTN_ArcXYC	缓存区指令，XY 平面圆弧插补(以终点位置和圆心位置为输入参数)	165
GTN_ArcYZR	缓存区指令，YZ 平面圆弧插补(以终点位置和半径为输入参数)	167
GTN_ArcYZC	缓存区指令，YZ 平面圆弧插补(以终点位置和圆心位置为输入参数)	166
GTN_ArcZXR	缓存区指令，ZX 平面圆弧插补(以终点位置和半径为输入参数)	168
GTN_ArcZXC	缓存区指令，ZX 平面圆弧插补(以终点位置和圆心位置为输入参数)	167
GTN_BufIO	缓存区指令，缓存区内数字量 IO 输出设置指令	172
GTN_BufDelay	缓存区指令，缓存区内延时设置指令	170
GTN_BufDA	缓存区指令，缓存区内输出 DA 值	170
GTN_BufLmtsOn	缓存区指令，缓存区内有效限位开关	173
GTN_BufLmtsOff	缓存区指令，缓存区内无效限位开关	173
GTN_BufSetStopIo	缓存区指令，缓存区内设置 axis 的停止 IO 信息	174
GTN_BufMove	缓存区指令，实现刀向跟随功能，启动某个轴点位运动	174
GTN_BufGear	缓存区指令，实现刀向跟随功能，启动某个轴跟随运动	171
GTN_CrdSpace	查询插补缓存区剩余空间	178
GTN_CrdClear	清除插补缓存区内的插补数据	176
GTN_CrdStart	启动插补运动	178
GTN_CrdStatus	查询插补运动坐标系状态	179
GTN_SetUserSegNum	缓存区指令，设置自定义插补段段号	255
GTN_GetUserSegNum	读取自定义插补段段号	215
GTN_GetRemainderSegNum	读取未完成的插补段段数	206
GTN_SetOverride	设置插补运动目标合成速度倍率	245
GTN_SetCrdStopDec	设置插补运动平滑停止、急停合成加速度	238
GTN_GetCrdStopDec	查询插补运动平滑停止、急停合成加速度	191
GTN_GetCrdPos	查询该坐标系的当前坐标位置值	190
GTN_GetCrdVel	查询该坐标系的合成速度值	192

指令	说明	页码
GTN_InitLookAhead	初始化插补前瞻缓存区	219
GTN_CrdHsOn	开启插补 DMA 传输通道	177
GTN_CrdHsOff	关闭插补 DMA 传输通道	177
GTN_GetCrdHsPrm	读取插补坐标系 DMA 设置参数	189
GTN_SetCrdMapBase	设置插补坐标系基础映射规划轴	236
GTN_GetCrdMapBase	读取插补坐标系基础映射规划轴	190
GTN_SetG0Mode	设置插补 G0 指令模式。	241
GTN_GetG0Mode	读取插补 G0 指令模式。	197

7.5.2 重点说明

1. 直线插补与圆弧插补

插补运动在数控机床，切削加工工艺等数控装置中应用广泛。它可以实现多轴的协调运动，将数据段所描述的曲线的起点、终点之间的空间进行数据密化，从而形成要求的轮廓轨迹，根据密化后的数据向各个坐标发出进给脉冲，对应每个脉冲，机床在相应的坐标方向上移动一个脉冲当量的距离，从而将工件加工出所需要的轮廓形状。

插补最常见的两种方式是直线插补和圆弧插补。

直线插补方式中，两点间的插补沿着直线的点群来逼近。首先假设在实际轮廓起始点处沿 x 方向走一小段（如一个脉冲当量），发现终点在实际轮廓的下方，则下一条线段沿 y 方向走一小段，此时如果线段终点还在实际轮廓下方，则继续沿 y 方向走一小段，直到在实际轮廓上方以后，再向 x 方向走一小段。依次循环类推。直到到达轮廓终点为止。这样实际轮廓是由一段段的折线拼接而成，虽然是折线，如果我们每一段走刀线段都在精度允许范围内，那么此段折线还是可以近似看做一条直线段。这就是直线插补。假设某数控机床刀具在 xy 平面上从点(x_0, y_0)运动到点 (x_1, y_1)，其直线插补的加工过程如图 7-8 所示。

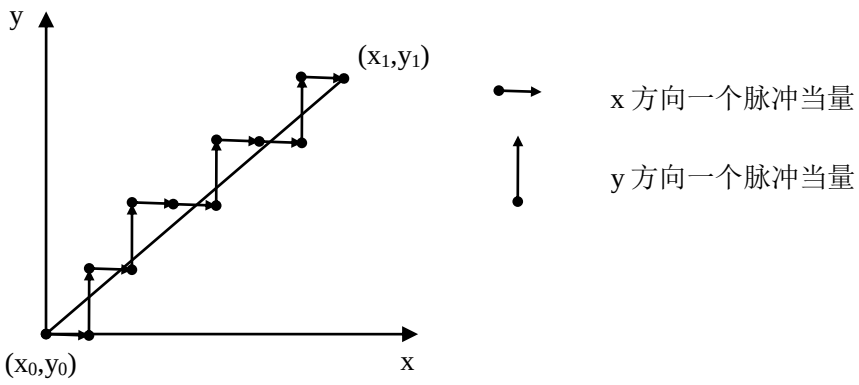


图 7-8 直线插补示意图

圆弧插补是给出两端点间的插补数字信息，以一定的算法计算出逼近实际圆弧的点群，控制刀具沿这些点运动，加工出圆弧曲线。圆弧插补只能在某一平面进行。假设某数控机床刀具在 xy 平面第一象限走一段逆圆弧，圆心为原点，半径为 5，起点 A(5, 0)，终点 B(0, 5)，其圆弧插补的加工过程如图 7-9 所示。

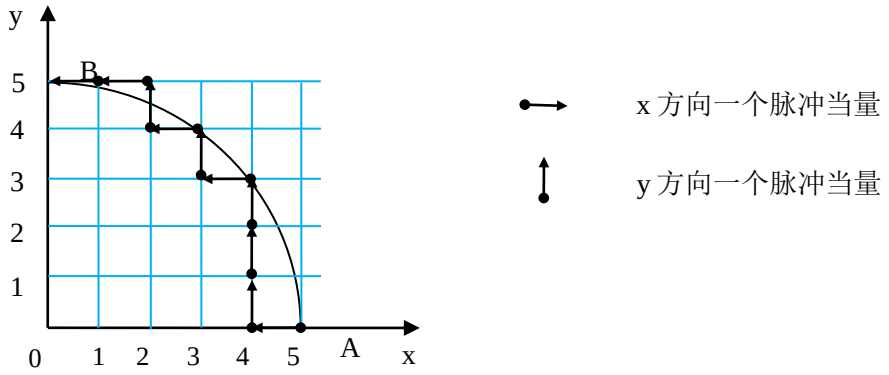


图 7-9　圆弧插补示意图

2. 运动控制器的插补模式

运动控制器的插补运动模式具有以下功能：

(1) 可以实现直线插补和圆弧插补；

(2) 可以同时有两个坐标系进行插补运动；

(3) 每个坐标系含有两个缓存区，可以实现缓存区暂停、恢复等功能；

(4) 具有缓存区延时和缓存区数字量输出的功能；

(5) 具有前瞻预处理功能，能够实现小线段高速平滑的连续轨迹运动。

3. 使用插补模式的步骤

使用插补模式需要至少两步操作：建立坐标系和向缓存区存入数据。下面分别就这两个步骤分别进行详细讲解。

(1) 建立坐标系

在运动控制器的初始状态下，复位之后或者还未使用过插补运动状态下，所有的规划轴都处于单轴运动模式下，两个坐标系也是无效的。所以，进行插补运动时，首先需要建立坐标系，将规划轴映射到相应的坐标系中。每个坐标系最多支持四维(X-Y-Z-A)，用户根据自己的需求，也可以利用二维(X-Y)、三维(X-Y-Z)坐标系描述运动轨迹。

用户通过调用指令 GTN_SetCrdPrm 指令将在坐标系内描述的运动通过映射关系映射到相应的规划轴上。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。调用指令 GTN_SetCrdPrm 时，所映射的各规划轴必须处于静止状态。

例程 7-4　建立坐标系

建立了一个二维坐标系，规划轴 1 对应为 x 轴，规划轴 2 对应为 y 轴，坐标系原点的规划位置是(100, 100)，单位：pulse，在此坐标系内运动的最大合成速度为 500pulse/ms，最大合成加速度为 1pulse/ms^2，最小匀速时间为 50ms。如图 7-10 所示。

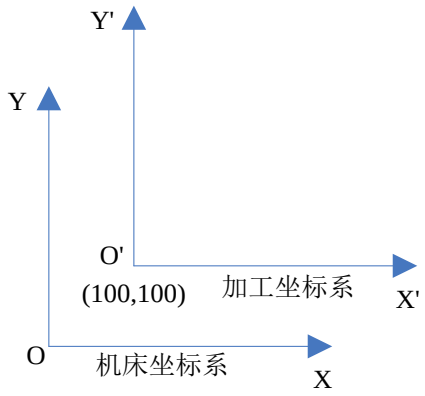


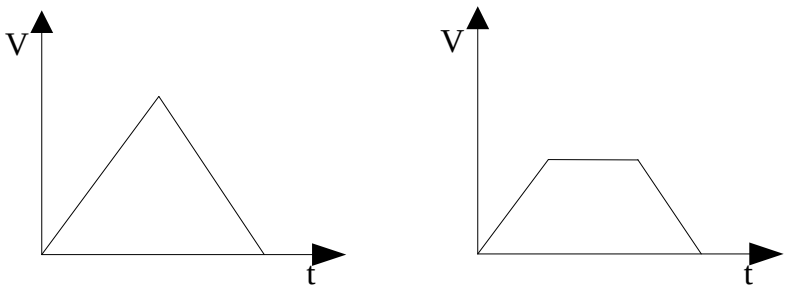
图 7-10 加工坐标系偏移量示意图

```
…… …… ……
// 指令返回值变量
short sRtn;
// TCrdPrm结构体变量，该结构体定义了坐标系
TCrdPrm crdPrm;

// 将结构体变量初始化为0
memset(&crdPrm, 0, sizeof(crdPrm));
// 为结构体赋值
crdPrm.dimension=2;              // 坐标系为二维坐标系
crdPrm.synVelMax=500;            // 最大合成速度：500pulse/ms
crdPrm.synAccMax=1;              // 最大加速度：1pulse/ms^2
crdPrm.evenTime = 50;            // 最小匀速时间：50ms
crdPrm.profile[0] = 1;           // 规划器1对应到X轴
crdPrm.profile[1] = 2;           // 规划器2对应到Y轴
crdPrm.setOriginFlag = 1;        // 表示需要指定坐标系的原点坐标的规划位置
crdPrm.originPos[0] = 100;       // 坐标系的原点坐标的规划位置为（100, 100）
crdPrm.originPos[1] = 100;

// 建立1号坐标系，设置坐标系参数

sRtn =   GTN_SetCrdPrm(1,1, &crdPrm);

…… …… ……
```

例程说明：

- dimension：表示所建立的坐标系的维数，取值范围：[1, 4]，该例程中所建立的坐标系是二维，即 X-Y 坐标系。
- synVelMax：表示该坐标系所能承受的最大合成速度，如果用户在输入插补段的时候所设置的目标速度大于了该速度，则将会被限制为该速度。
- synAccMax：表示该坐标系所能承受的最大合成加速度，如果用户在输入插补段的时候所设置的加速度大于了该加速度，则将会被限制为该加速度。
- evenTime：每个插补段的最小匀速时间。当用户设置的插补段比较短时，而该插补段的目标速度又设置的比较大，则会造成合成速度的曲线如图 7-11 (a)所示，只有加速段和减速段，形成一个

速度尖角，加速度在尖角处瞬间由正值变为了负值，造成较大的冲击；设置了 evenTime 之后，可以减小目标速度，使速度曲线如图 7-11 (b)所示，减小了加速度突变的冲击。



(a) evenTime=0　　(b) evenTime>0

图 7-11　不同 evenTime 下的速度曲线

(2) 向缓存区存入数据

运动控制器插补运动模式采用缓存区运动方式，即用户需要向插补缓存区中传递插补数据，然后，启动插补运动，运动控制器则会依次执行用户所传递的插补数据，直到所有的插补数据全部运动完成。

向缓存区存入数据的指令分直线插补（以 GTN_Ln 开头）和平面圆弧插补指令（以 GTN_Arc 开头）两种。

例程 7-5　直线插补例程

假设某数控机床刀具在 xy 平面从原点出发，走一段如图 7-12 所示的正六边形轨迹。一共需要走七段轨迹，图中标号已标出。每走完一段轨迹会输出一次 IO 信号，并且暂停 400ms，其直线插补的例程如下，直线插补例程运动轨迹如图 7-12 所示。

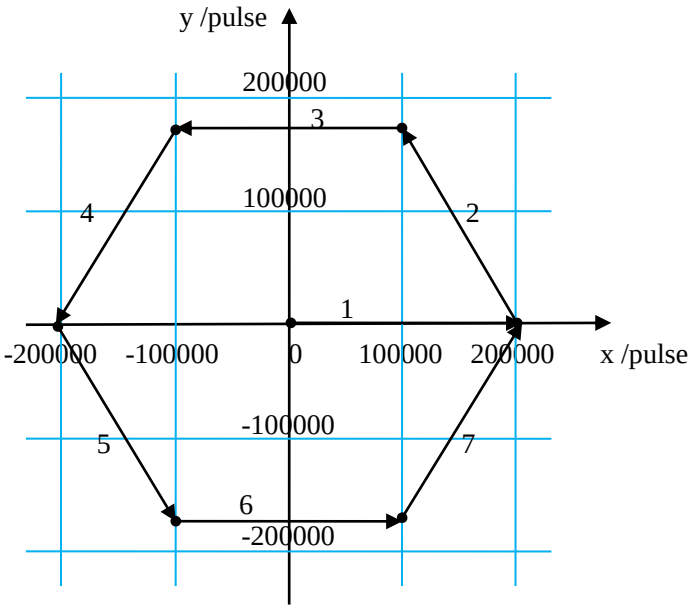


图 7-12　直线插补例程运动轨迹

```
…… …… ……
// 指令返回值变量
short sRtn;
// 坐标系运动状态查询变量
short run;
```

```
// 坐标系运动完成段查询变量
long segment;
// 坐标系的缓存区剩余空间查询变量
long space;

// 即将把数据存入坐标系1的FIFO0中，所以要首先清除此缓存区中的数据
sRtn = GTN_CrdClear(1,1, 0);
// 向缓存区写入第一段插补数据
sRtn = GTN_LnXY(
    1,
    1,              // 该插补段的坐标系是坐标系1
    200000, 0,      // 该插补段的终点坐标(200000, 0)
    100,            // 该插补段的目标速度：100pulse/ms
    0.1,            // 插补段的加速度：0.1pulse/ms^2
    0,              // 终点速度为0
    0);             // 向坐标系1的FIFO0缓存区传递该直线插补数据

// 向缓存区写入第二段插补数据
sRtn = GTN_LnXY(1,1, 100000, 173205, 100, 0.1, 0, 0);
// 缓存区数字量输出
sRtn = GTN_BufIO(
    1,
    1,              // 坐标系是坐标系1
    MC_GPO,         // 数字量输出类型：通用输出
    0xffff,         // bit0~bit15全部都输出
    0x55,           // 输出的数值为0x55
    0);             // 向坐标系1的FIFO0缓存区传递该数字量输出
// 第三段插补数据
sRtn = GTN_LnXY(1,1, -100000, 173205, 100, 0.1, 0, 0);
// 缓存区数字量输出
sRtn = GTN_BufIO(1,1, MC_GPO, 0xffff, 0xaa, 0);
// 第四段插补数据
sRtn = GTN_LnXY(1,1, -200000, 0, 100, 0.1, 0, 0);
// 缓存区延时指令
sRtn = GTN_BufDelay(
    1,
    1,              // 坐标系是坐标系1
    400,            // 延时400ms
```

```
        0);                     // 向坐标系1的FIFO0缓存区传递该延时
    // 第五段插补数据
    sRtn = GTN_LnXY(1,1, -100000, -173205, 100, 0.1, 0, 0);
    // 缓存区数字量输出
    sRtn = GTN_BufIO(1,1, MC_GPO, 0xffff, 0x55, 0);
    // 缓存区延时指令
    sRtn = GTN_BufDelay(1,1, 100, 0);
    // 第六段插补数据
    sRtn = GTN_LnXY(1,1, 100000, -173205, 100, 0.1, 0, 0);
    // 第七段插补数据
    sRtn = GTN_LnXY(1,1, 200000, 0, 100, 0.1, 0, 0);
    // 查询坐标系1的FIFO0所剩余的空间
    sRtn = GTN_CrdSpace(1,1, &space, 0);
    // 启动坐标系1的FIFO0的插补运动
    sRtn = GTN_CrdStart(1,1, 0);
    // 等待运动完成
    sRtn = GTN_CrdStatus(1,1, &run, &segment, 0);
    do
    {
        // 查询坐标系1的FIFO的插补运动状态
        sRtn = GTN_CrdStatus(
            1,
            1,              // 坐标系是坐标系1
            &run,           // 读取插补运动状态
            &segment,       // 读取当前已经完成的插补段数
            0);             // 查询坐标系1的FIFO0缓存区
    // 坐标系在运动，查询到的run的值为1
    }while(run == 1);
…… …… ……
```

(3) 圆弧插补

运动控制器的插补模式支持在 XY 平面、YZ 平面和 ZX 平面的圆弧插补。其中圆弧插补的旋转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，来确定逆时针方向和顺时针方向。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逆时针方向。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逆时针方向同样定义，如图 7-13 示。

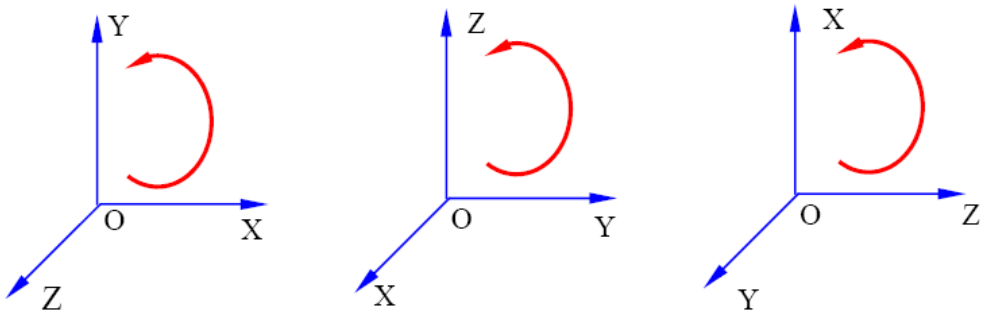


图 7-13 圆弧插补逆时针方向

圆弧插补有两种描述方法：半径描述方法和圆心坐标描述方法，用户可以根据加工数据选择合适的描述方法来编程。所使用的描述方法遵循 G 代码的编程标准。

a) 半径描述方法

调用指令 GTN_ArcXYR、 GTN_ArcYZR、 GTN_ArcZXR 是使用半径描述方法对圆弧进行描述。使用半径描述方法，用户需要输入圆弧终点坐标、圆弧半径、圆弧的旋转方向、速度和加速度等。其中参数半径可为正值，也可为负值，其绝对值为圆弧的半径，正值表示圆弧的旋转角度≤180°，负值表示圆弧的旋转角度＞180°，如图 7-14 所示，半径描述方法无法描述 360°的整圆。

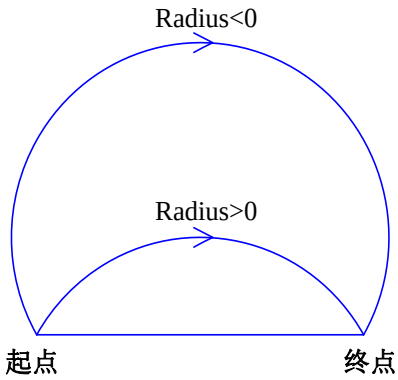


图 7-14 半径取正值/负值圆弧插补示意图

b) 圆心坐标描述方法

调用指令 GTN_ArcXYC、GTN_ArcYZC、GTN_ArcZXC 是使用圆心坐标描述方法对圆弧进行描述。使用圆心描述方法，用户需要输入圆弧终点坐标、圆心相对于起点坐标的相对位置值、圆弧的旋转方向、速度和加速度等。其中，圆心位置值参数的定义如图 7-15 所示。

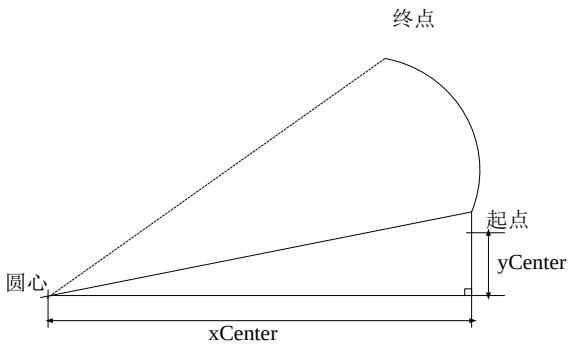


图 7-15 圆心坐标描述方法示意图

圆心参数为相对于起点位置的增量值，带正负号，如果起点的坐标为(xStart, yStart)，用户设置的圆心

参数为(xCenter, yCenter)，则圆心坐标值为(xStart+xCenter, yStart+yCenter)。用户设置的起点坐标和终点坐标重合时，则表示将要进行一个整圆的运动。

注意	用户应该保证圆弧描述指令可以正确描述一段圆弧，如果用户所设置的参数不能生成一段正确的圆弧，指令会返回 7(参数错误)。

例程 7-6　圆弧插补例程

假设某数控机床刀具在 xy 平面走一段如图 7-16 所示的圆弧。该例程使用圆心坐标描述方法描述一个整圆，使用半径描述方法描述一个 1/4 圆弧，最后回到原点位置。一共需要走三段轨迹，图中用标号标出。整圆的圆心坐标为(100000, 0)，半径 100000pulse。圆弧的圆心坐标为原点(0, 0)，半径 200000pulse。

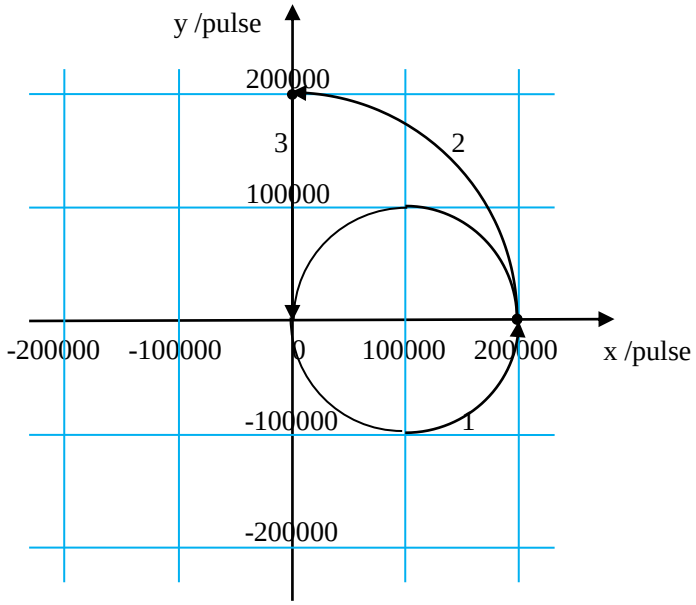


图 7-16　圆弧插补例程运动轨迹

```
…… …… ……
// 指令返回值变量
short sRtn;
// 坐标系运动状态查询变量
short run;
// 坐标系运动完成段查询变量
long segment;
// 坐标系的缓存区剩余空间查询变量
long space;

// 即将把数据存入坐标系1的FIFO0中，所以要首先清除此缓存区中的数据

sRtn =  GTN_CrdClear(1,1, 0);

// 向缓存区写入第一段插补数据

sRtn =  GTN_LnXY(

    1,
    1,                  // 该插补段的坐标系是坐标系1
    200000, 0,          // 该插补段的终点坐标(200000, 0)
```

```
        100,                    // 该插补段的目标速度：100pulse/ms
        0.1,                    // 插补段的加速度：0.1pulse/ms^2
        0,                      // 终点速度为0
        0);                     // 向坐标系1的FIFO0缓存区传递该直线插补数据

// 向缓存区写入第二段插补数据，该段数据是以圆心描述方法描述了一个整圆

sRtn =  GTN_ArcXYC(1,

        1,                      // 坐标系是坐标系1
        200000, 0,              // 该圆弧的终点坐标(200000, 0)
        -100000, 0,             // 圆弧插补的圆心相对于起点位置的偏移量(-100000, 0)
        0,                      // 该圆弧是顺时针圆弧
        100,                    // 该插补段的目标速度：100pulse/ms
        0.1,                    // 该插补段的加速度：0.1pulse/ms^2
        0,                      // 终点速度为0
        0);                     // 向坐标系1的FIFO0缓存区传递该直线插补数据

// 向缓存区写入第三段插补数据，该段数据是以半径描述方法描述了一个1/4圆弧

sRtn =  GTN_ArcXYR(1,

        1,                      // 坐标系是坐标系1
        0, 200000,              // 该圆弧的终点坐标(0, 200000)
        200000,                 // 半径：200000pulse
        1,                      // 该圆弧是逆时针圆弧
        100,                    // 该插补段的目标速度：100pulse/ms
        0.1,                    // 该插补段的加速度：0.1pulse/ms^2
        0,                      // 终点速度为0
        0);                     // 向坐标系1的FIFO0缓存区传递该直线插补数据

// 向缓存区写入第四段插补数据，回到原点位置

sRtn =  GTN_LnXY(1,1, 0, 0, 100, 0.1, 0, 0);

// 查询坐标系1的FIFO0所剩余的空间

sRtn =  GTN_CrdSpace(1,1, &space, 0);

// 启动坐标系1的FIFO0的插补运动

sRtn =  GTN_CrdStart(1,1, 0);

// 等待运动完成

sRtn =  GTN_CrdStatus(1,1, &run, &segment, 0);

do
{
    // 查询坐标系1的FIFO的插补运动状态

    sRtn =  GTN_CrdStatus(1,
```

```
                1,              // 坐标系是坐标系1
                &run,           // 读取插补运动状态
                &segment,       // 读取当前已经完成的插补段数
                0);             // 查询坐标系1的FIFO0缓存区
    // 坐标系在运动，查询到的run的值为1
    }while(run == 1);

    …… …… ……
```

4. 以 GTN_Ln 开头 G0 结尾的指令

这一类指令包括 GTN_LnXYG0、 GTN_LnXYZG0 和 GTN_LnXYZAG0。这类运动指令将会完成一个完整的加减速过程，即每段运动的合成速度都是从 0 开始，结束的时候也是 0。

这类指令与其他插补运动指令(包括直线插补指令和圆弧插补指令)的区别在于：如果使用了前瞻预处理功能，调用其他插补运动指令，则用户设置的终点速度将会无效，实际的终点速度是前瞻预处理模块根据用户设置的前瞻预处理参数和运动轨迹计算出来的一个合理的终点速度；但如果调用 GTN_LnXYG0 系列指令，终点速度仍然是 0，控制器不会优化或改变这类指令的终点速度。

5. 前瞻预处理

在数控加工等应用中，要求数控系统对机床进行平滑的控制，以防止较大的冲击影响零件的加工质量。运动控制器的前瞻预处理功能可以根据用户的运动路径计算出平滑的速度规划，减少机床的冲击，从而提高加工精度。

下面用一个实例来说明前瞻预处理的机制优势。假设机床要加工一个长方形的零件，刀具所走的轨迹如图 7-17 (a)所示。假设 m 点到 n 点距离 3000 个单位长度，有 30 段规划。n 点到 p 点距离 2000 个单位长度，有 20 段规划。每段规划 100 个单位长度。

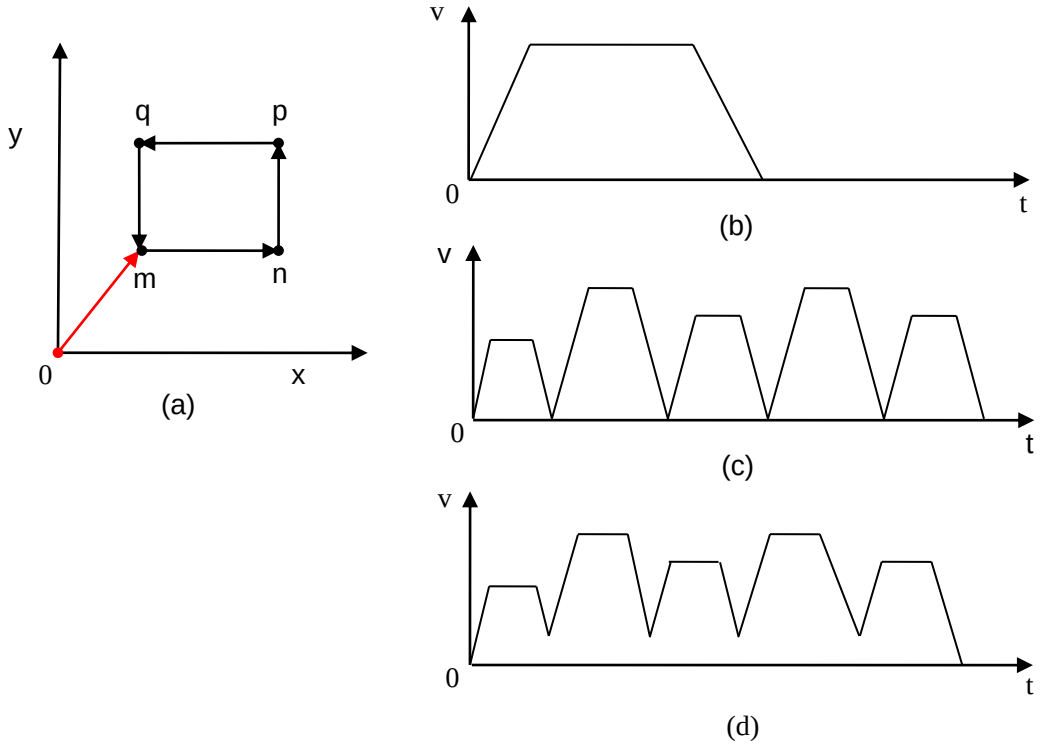


图 7-17　使用前瞻与不使用前瞻的速度规划区别

如果按照图 7-17 (b)所示的速度规划，即在拐角处不减速，则加工精度一定会较低，而且可能在拐弯时对刀具和零件造成较大冲击。如果按照图 7-17 (c)所示的速度规划，即在拐角处减速为 0，可以最大限度保证加工精度，但加工速度就会慢下来。如果按照图 7-17(d)所示的速度规划，在拐角处将速度减到一个合理值，既可以满足加工精度又能提高加工速度，就是一个好的速度规划。

为了实现类似图 7-17 (d)所示的好的速度规划，前瞻预处理模块不仅要知道当前运动的位置参数，还要提前知道后面若干段运动的位置参数，这就是所谓的前瞻。例如在对图 7-17 (a)中的轨迹做前瞻预处理时，我们设定控制器预先读取 50 段运动轨迹到缓存区中，则它会自动分析出在第 30 段将会出现拐点，并依据用户设定的拐弯时间计算在拐弯处的终点速度。前瞻预处理模块也会依照用户设定的最大加速度值计算速度规划，使任何加减速过程都不会超过这个值，防止对机械部分产生破坏性冲击力。

从下图 7-18 可以直观地了解，使用前瞻预处理功能模块来规划速度，在小线段加工过程中的对速度的显著提升。前瞻预处理流程图如图 7-19 所示。

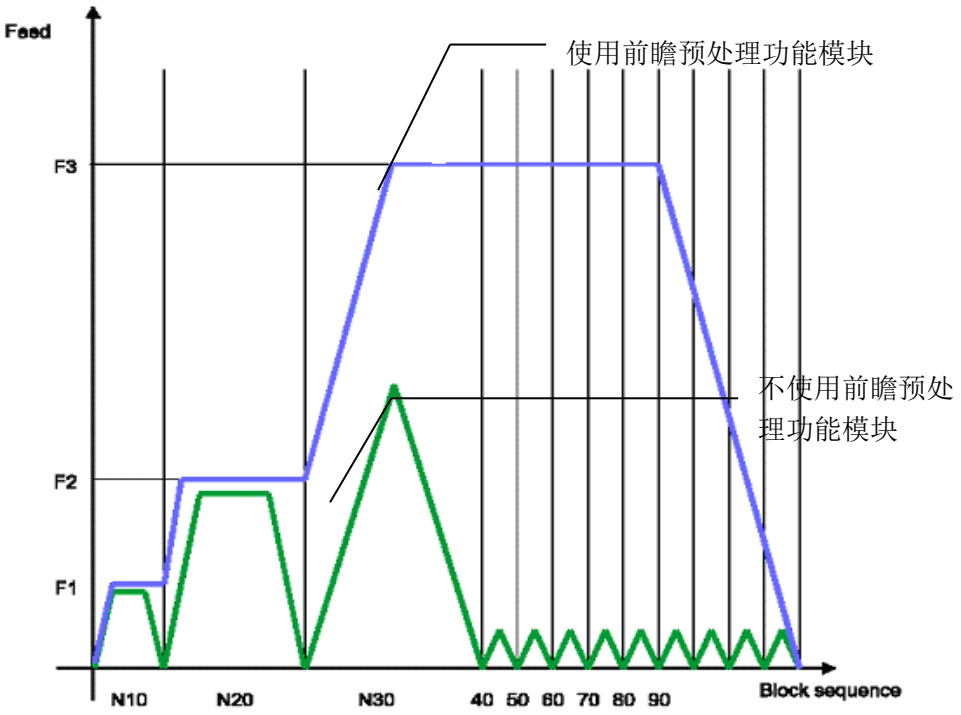


图 7-18　使用和不使用前瞻预处理功能模块的速度曲线对比图

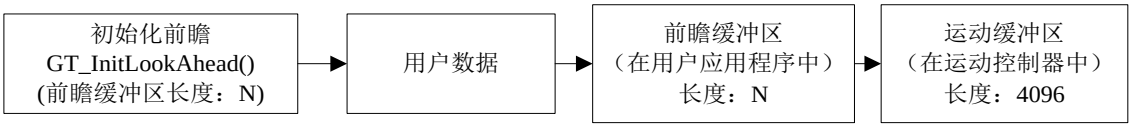


图 7-19　前瞻预处理流程图

(1) 用户在应用程序中定义一个结构体 TCrdData 的数组作为前瞻缓存区，数组大小用户自己设定，比如数组大小为 200，那么前瞻缓存区长度 N=200 段；然后调用指令 GTN_InitLookAhead 作初始化前瞻。

(2) 用户调用如 GTN_LnXY 等直线插补指令和圆弧插补指令将数据段输入缓存区。这时，插补数据先流入开辟的前瞻缓存区，当流入前瞻缓存区的数据段数大于了 N 之后，才能逐段流入运动缓存区。运动缓存区是控制器内部资源，大小 4096 段，每一段可以存放一条指令。控制器只能执行压入运动缓存区中的数据，所以用户一定要确保前瞻缓存区的数据进入运动缓存区。分不同情况分析：

1) 假设用户数据只有 190 段，则当用户调用完后，数据会一直停留在前瞻缓存区，因此，用户需要调用 GTN_CrdData(1, NULL, 0)来将前瞻缓存区数据压入运动缓存区。

2) 假设用户数据只有 300 段，则当用户调用完后，有 100 段数据已经流入运动缓存区，但还有 200 段留在前瞻缓存区，同样，用户需要调用 GTN_CrdData(1, NULL, 0)来将前瞻缓存区数据压入运动缓存区。

3) 假设用户有数据 5000 段，则在用户调用插补指令过程中，会出现前瞻缓存区和运动缓存区都被压满的情况，因此，需要注意，若一直压数据，没有启动插补运动，当压入第 4297 段时（前瞻缓存区和运动缓存区一共 4296 段大小），由于两缓存区都满了，所以调用指令会返回 1 。此时需要调用 GTN_CrdSpace 查询运动缓存区的空间，只有当查询到当前运动缓存区有空间时，才能继续调用插补指令压入剩下的数据。同样，最后用户需要调用 GTN_CrdData(1, NULL, 0)来将前瞻缓存区数据压入运动缓存区。

前瞻预处理功能只支持 3 轴或者 3 轴以下的插补运动，如果建立的坐标系大于 3 轴，则在使用前瞻预处理功能时，会返回错误值，调用缓存区指令时，会返回 7(参数错误)。

例程 7-7　前瞻预处理例程

假设机床加工过程中，需要走一长直线，该直线由 300 条小直线段组成，现对这段路径进行前瞻预处理。其轨迹如图 7-20 所示。红色线段为起始轨迹。

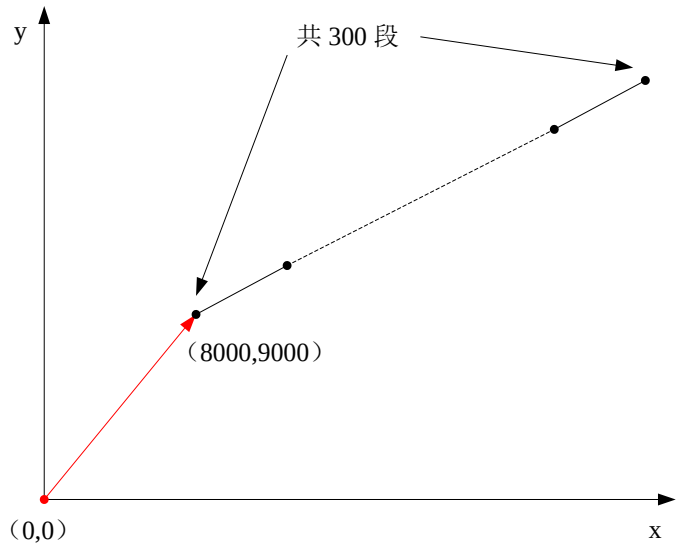


图 7-20　前瞻预处理例程之运动轨迹图

```
…… …… ……
// 指令返回值
short sRtn;
// 循环变量
int i;
// 定义前瞻缓存区内存区
TCrdData crdData[200];
long posTest[2];
long space;

// 初始化坐标系1的FIFO0的前瞻模块

sRtn =  GTN_InitLookAhead(1,1, 0, 5, 1, 200, crdData);

// 压插补数据：小线段加工
```

```
posTest[0] = 0;
posTest[1] = 0;
for(i=0;i<300;++i)
{
    sRtn =  GTN_LnXY(1,1, 8000+posTest[0], 9000+posTest[1], 100, 0.8, 0, 0);
    // 查询返回值是否成功
    if(0 != sRtn)
    {
        do
        {
            // 查询运动缓存区空间，直至空间不为0
            sRtn =  GTN_CrdSpace(1,1, &space, 0);
        }while(0 == space);
        // 重新调用上次失败的插补指令
        sRtn =  GTN_LnXY(1,1, 8000+posTest[0], 9000+posTest[1], 100, 0.8, 0, 0);
    }
    posTest[0] += 1600;
    posTest[1] += 1852;
}

// 将前瞻缓存区中的数据压入控制器
sRtn =  GTN_CrdData(1,1, NULL, 0);
// 启动运动
sRtn =  GTN_CrdStart(1,1, 0);
…… …… ……
```

例程说明：

- 拐弯时间(T)： GTN_InitLookAhead 指令的第三个参数，单位：ms。T 的经验范围是：1ms~10ms，T 越大，计算出来的终点速度越大，但却降低了加工精度；反之，提高了加工的精度，但计算出的终点速度偏低。因此要合理选择 T 值。
- 最大加速度(accMax)： GTN_InitLookAhead 指令的第四个参数，单位：pulse/ms2。系统能承受的最大加速度，根据不同的机械系统和电机驱动器取值不同。
- 前瞻缓存区(pLookAheadBuf)：前瞻缓存区是用户在应用程序中自己定义的，用于存放描述运动轨迹的数组。用户应根据自己的需要以及计算机的条件定义合适的缓存区大小，并且要在 GTN_InitLookAhead 指令的第五个参数中说明数组的大小。

调用 GTN_InitLookAhead 指令之后，在进行前瞻预处理的过程中，用户不能再对前瞻缓存区进行任何操作，否则将会破坏前瞻缓存区中的数据，造成数据的错误。

- 运动缓存区：插补缓存区是运动控制器内部专门用于插补运动的缓存区资源，大小 4096 段，每

一段可以放一条指令。

当前瞻缓存区的段数不为 0 时，用户调用缓存区指令传递的插补数据先进入前瞻缓存区，当前瞻缓存区放满之后，如果再有新的数据传入，最先进入前瞻缓存区的数据，则会进入插补缓存区。

如果用户所有的插补数据已经输入完毕，前瞻缓存区中还有数据没有进入插补缓存区，这时，需要调用 GTN_CrdData(1, NULL, 0)，运动控制器会将前瞻缓存区的数据依次传递给插补缓存区，直到前瞻缓存区被清空为止。

在数据量比较大的时候，用户需要配合 GTN_CrdSpace 指令查询插补缓存区的剩余空间，在有空间的时候再调用缓存区指令传递数据，如果插补缓存区已满，调用缓存区指令将会返回错误，说明该段插补数据没有输入成功，需要再次输入该段插补数据。

- 没有前瞻预处理和经过前瞻预处理的区别如图 7-21 和图 7-22 所示。

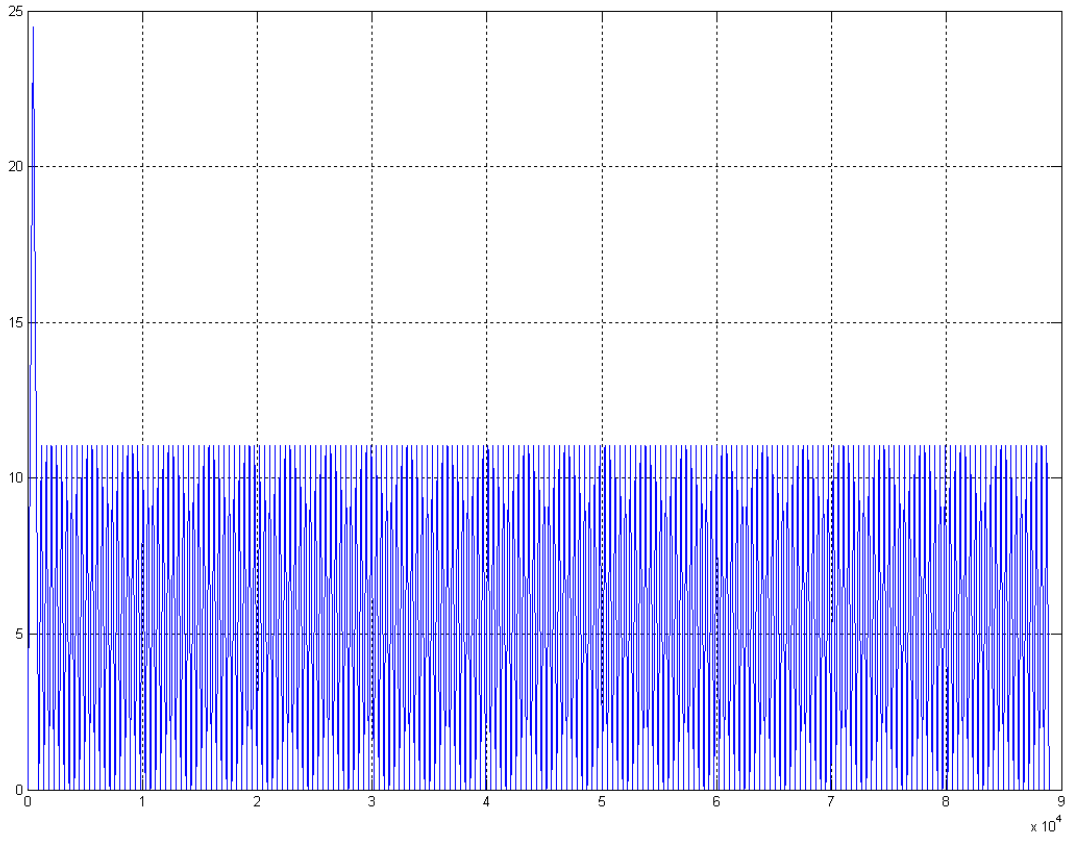


图 7-21　没有进行前瞻预处理的合成速度曲线

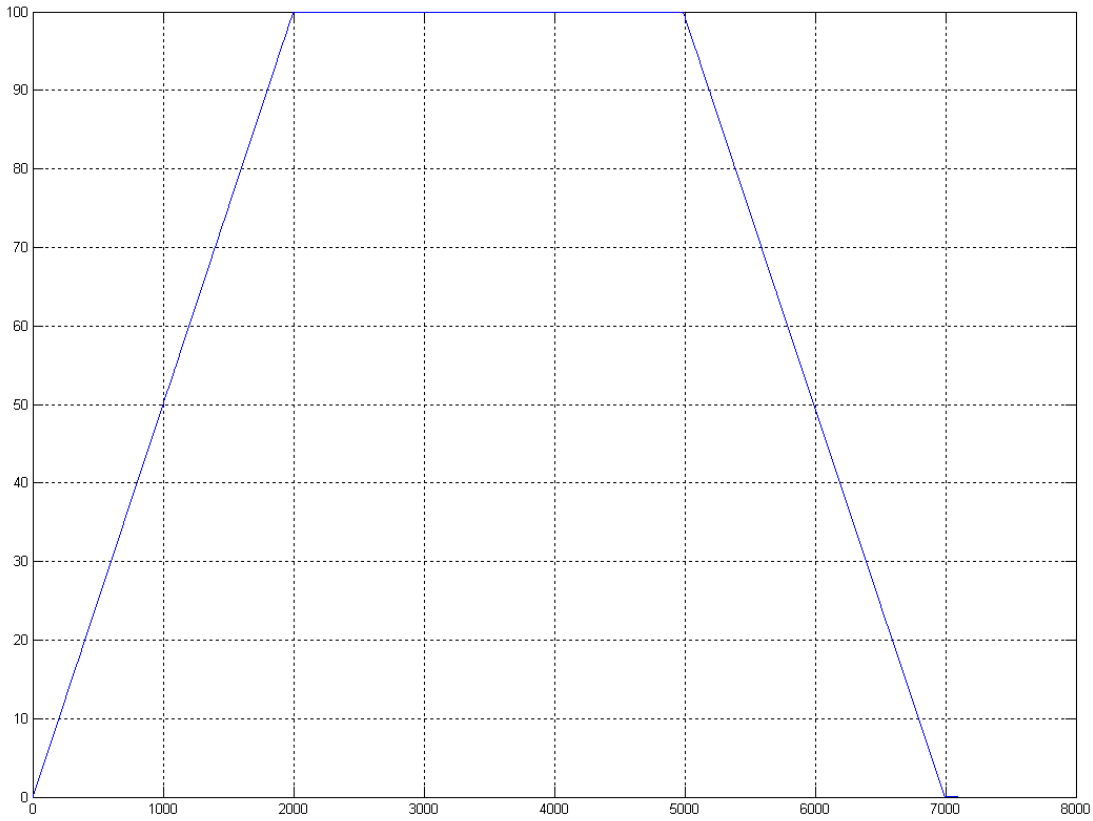


图 7-22　进行了前瞻预处理后的合成速度曲线

6. 刀向跟随功能

刀向跟随，就是在插补运动的过程中，使非插补轴随着插补运动的合成位移的变化而变化，从而实现在加工过程中，刀具始终处于合适的加工方向和位置的工艺。在本控制器的插补模块中有两条指令来实现该工艺： GTN_BufMove， GTN_BufMove 和 GTN_BufGear。

(1) GTN_BufMove

在插补运动过程中，如果需要坐标系以外的轴进行点位运动，则可以通过在缓存区中压入 GTN_BufMove 指令来实现。

GTN_BufMove 可以在插补运动的过程中插入模态和非模态的点位运动。**模态指令**的意义是，在进行该点位运动时，后续的插补缓存区中的指令将会被暂停执行，直到该指令执行完毕后，才执行下一条指令；**非模态指令**的意义是，该指令启动了一个轴的点位运动后，立即取下一条缓存区中的指令执行，不会等待点位运动的结束。

该指令的第二个参数是需要进行点位运动的轴号。

注意	需要进行点位运动的轴必须是坐标系外的轴，该轴的运动模式必须是点位运动，且该轴必须处于静止状态，否则该指令不能正常执行。

该指令的第三个参数是点位运动的目标位置，该位置值是相对于机床原点的绝对位置。该指令的第四个参数是点位运动的目标速度，该值必须为正值。该指令的第五个参数是点位运动的加速度，该值必须为正值。该指令的第六个参数表示该点位运动是模态的还是非模态的。

例程 7-8　刀向跟随功能

假设一个机床加工环境，有一个 XY 的二维坐标系，和一个不在坐标系内的轴 A。在 XY 坐标系内，刀具先从(0, 0)运动到(200000, 200000)（第 1 段轨迹）。然后，在 XY 方向从(200000, 200000)运动到(200000, 0)的同时，在 A 轴方向以 30 pulse/ms 的速度运动到 50000pulse 的位置（第 2 段轨迹）。这里用到 GTN_BufMove 的非模态方式。然后，在 A 轴方向以 30 pulse/ms 的速度运动 100000 的位置。等到 A 轴方向运动完成，在 XY 坐标内画一个圆心为原点，半径 200000，位于三四象限的半圆弧（第 3 段轨迹）。这里用到 GTN_BufMove 的模态方式，刀向跟随功能 GTN_BufMove 的运动轨迹如图 7-23 所示。

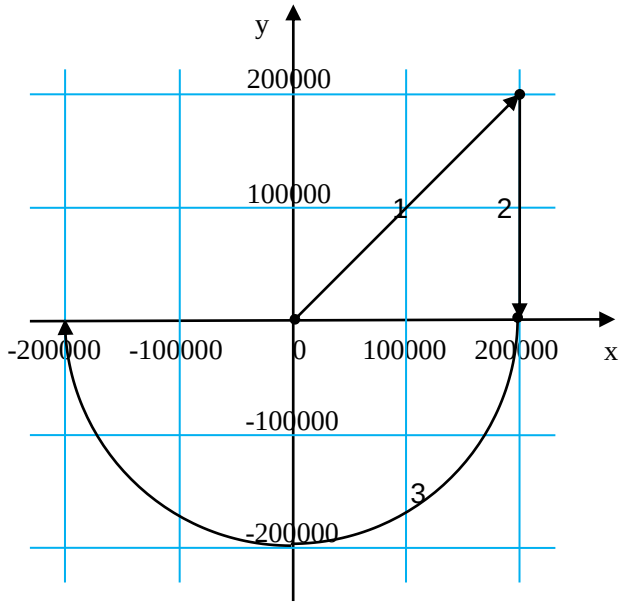


图 7-23　刀向跟随功能 GTN_BufMove 的运动轨迹

```
…… …… ……
// 定义指令返回值变量
short sRtn;
// 定义坐标系运动状态查询变量
short run;
// 定义坐标系运动完成段查询变量
long segment;

// 清除坐标系1的FIFO0中的数据

sRtn = GTN_CrdClear(1,1, 0);

// 向FIFO0缓存区写入一段直线插补数据

sRtn = GTN_LnXY(1,

    1,                  // 该插补段的坐标系是坐标系1
    200000, 200000,     // 该插补段的终点坐标(200000, 200000)
    100,                // 该插补段的目标速度：100pulse/ms
    0.1,                // 插补段的加速度：0.1pulse/ms^2
    0,                  // 终点速度为0
    0);                 // 向坐标系1的FIFO0缓存区传递该直线插补数据
// 向FIFO0缓存区写入一段非模态点位运动数据

sRtn = GTN_BufMove(1,
```

```
        1,                      // 该插补段的坐标系是坐标系1
        4,                      // 点位运动的轴号：第4轴
        50000,                  // 点位运动的目标位置：50000 pulse
        30,                     // 点位运动的目标速度：30 pulse/ms
        0.1,                    // 点位运动的目标加速度：0.1 pulse/(ms*ms)
        0,                      // 该点位运动是非模态指令
        0);                     // 向坐标系1的FIFO0缓存区传递该直线插补数据
    // 向FIFO0缓存区写入一段直线插补数据

    sRtn = GTN_LnXY(1,1, 200000, 0, 100, 0.1, 0, 0);     // 直线插补指令

    // 向FIFO0缓存区写入一段模态点位运动数据

    sRtn = GTN_BufMove(1,

        1,                      // 该插补段的坐标系是坐标系1
        4,                      // 点位运动的轴号：第4轴
        100000,                 // 点位运动的目标位置：100000 pulse
        30,                     // 点位运动的目标速度：30 pulse/ms
        0.1,                    // 点位运动的目标加速度：0.1 pulse/(ms*ms)
        1,                      // 该点位运动是模态指令
        0);                     // 向坐标系1的FIFO0缓存区传递该直线插补数据
    // 向缓存区写入一段圆弧插补数据，该段数据是以圆心描述方法描述了一个半圆

    sRtn = GTN_ArcXYC(1,

        1,                      // 坐标系是坐标系 1
        -200000, 0,             // 该圆弧的终点坐标(-200000, 0)
        -200000, 0,             // 圆弧插补的圆心相对于起点位置的偏移量(-200000, 0)
        0,                      // 该圆弧是顺时针圆弧
        100,                    // 该插补段的目标速度：100pulse/ms
        0.1,                    // 该插补段的加速度：0.1pulse/ms^2
        0,                      // 终点速度为0
        0);                     // 向坐标系1的FIFO0缓存区传递该直线插补数据


    do
    {
        // 坐标系在运动，查询到的run的值为1

        sRtn = GTN_CrdStatus(1,1, &run, &segment, 0);

    }while(run == 1);
    …… …… ……
```

例程插补合成速度和点位速度的运行结果如图 7-24 所示。

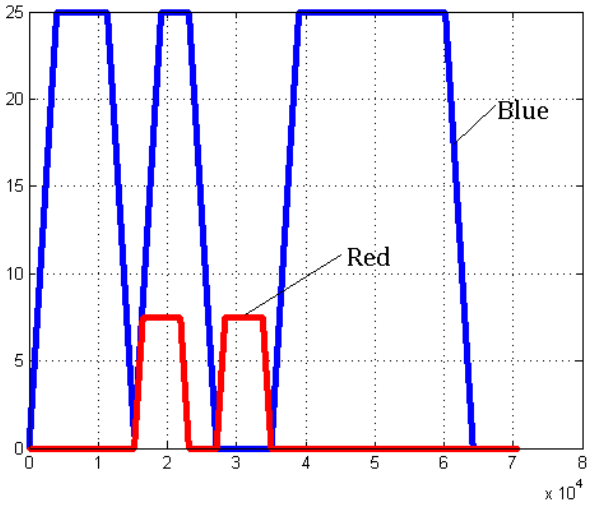


图 7-24 插补缓存区内的点位运动速度图

其中蓝色为插补运动的合成速度，红色为点位运动轴的速度值，可以看出第一个点位运动是非模态指令，与插补运动同时运动，而第二个点位运动是模态指令，会阻塞插补运动，只有点位运动结束之后，才能进行插补运动。

注意	在使用插补缓存区中的点位运动功能时，需要注意以下内容： 1. 点位运动的目标位置是相对于机床原点的绝对位置。 2. 如果在上一次的缓存区点位运动没有运动完成，又发送了新的点位运动，则会按照新的点位运动指令进行规划，即可以在插补缓存区中修改点位运动的目标位置和目标速度。 3. 如果在运动过程中停止插补缓存区的运动，则点位运动将不会停止，如果需要停止点位运动，则需要调用 GTN_Stop 指令停止响应轴的运动。恢复缓存区运动时，用户需自行保证之前点位运动的轴在合适的位置上。 4. 当在模态点位运动的过程中，进行点位运动的轴由于触发限位等异常原因停止时，插补缓存区将不会再继续运行，此时用户需排查异常情况，重新设置相应参数，使系统正常后才可以工作。

(2) GTN_BufGear

在插补过程中，需要坐标系外某一轴跟随坐标系插补运动的时候，可以通过在缓存区中压入 GTN_BufGear 指令来实现，该指令的第二个参数是需要进行跟随运动的轴号。

注意	需要进行跟随运动的轴不能是坐标系中的轴；如果在发送跟随指令 GTN_BufGear 时该轴正在运动，该指令将不能正常执行。

这里需要注意的是，该指令的第三个参数是跟随运动的位移量，该位移量是相对值，即下一段插补段运动过程中，跟随轴需要运动的位移量。使用的具体例程如下：

注意	GTN_BufMove 中的位置参数是相对于机床原点的绝对位置，而 GTN_BufGear 中的位置参数是相对位置。

例程 7-9 刀向跟随功能

假设一个机床加工环境，有一个 XY 的二维坐标系，和一个不在坐标系内的轴 A。在 XY 坐标系内，刀具先从(0, 0)运动到(200000, 200000)（第 1 段轨迹）。然后，在 XY 方向从(200000, 200000)运动到(200000,

0)的同时，在 A 轴方向运动到 50000pulse 的位置，两者将同时到达各自指定的规划位置（第 2 段轨迹）。然后，在 XY 坐标内画一个圆心为原点，半径 200000，位于三四象限的半圆弧，同时在 A 轴方向运动 100000 的位置，两者将同时到达各自指定的规划位置（第 3 段轨迹），刀向跟随功能 GTN_BufGear 的运动轨迹如图 7-25 所示。

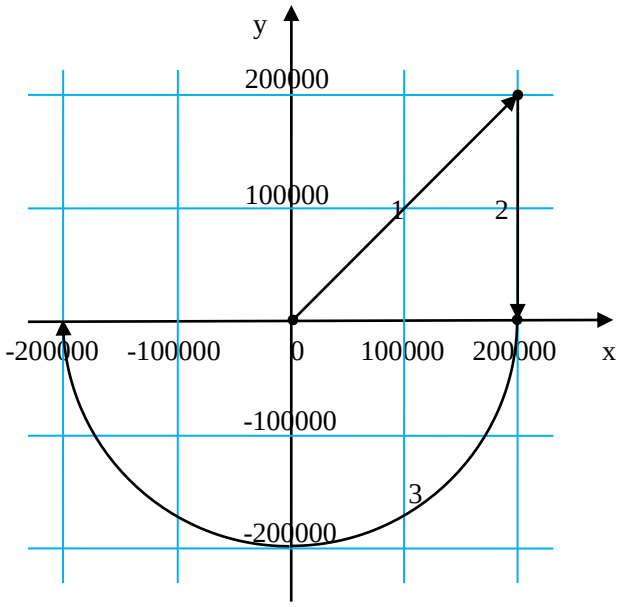


图 7-25 刀向跟随功能 GTN_BufGear 的运动轨迹

```
…… …… ……
// 定义指令返回值变量
short sRtn;
// 定义坐标系运动状态查询变量
short run;
// 定义坐标系运动完成段查询变量
long segment;

// 清除坐标系1的FIFO0中的数据

sRtn =  GTN_CrdClear(1,1, 0);

// 向FIFO0缓存区写入一段直线插补数据

sRtn =  GTN_LnXY(1,

    1,                  // 该插补段的坐标系是坐标系1
    200000, 200000,     // 该插补段的终点坐标(200000, 200000)
    100,                // 该插补段的目标速度：100pulse/ms
    0.1,                // 插补段的加速度：0.1pulse/ms^2
    0,                  // 终点速度为0
    0);                 // 向坐标系1的FIFO0缓存区传递该直线插补数据
// 向FIFO0缓存区写入一段跟随运动数据，

// GTN_BufGear()指令需要在所要跟随的插补段前

sRtn = GTN_BufGear(1,

    1,                  // 该插补段的坐标系是坐标系1
```

```
        4,                  // 跟随运动的轴号：第4轴
        50000,              // 跟随运动的位移量：50000 pulse。这里的位置参数是相对位置。
        0);                 // 向坐标系1的FIFO0缓存区传递该直线插补数据
    // 向FIFO0缓存区写入一段直线插补数据

    sRtn = GTN_LnXY(1,1, 200000, 0, 100, 0.1, 0, 0);

    // 向FIFO0缓存区写入一段跟随运动数据

    sRtn = GTN_BufGear(1,

        1,                  // 该插补段的坐标系是坐标系1
        4,                  // 跟随运动的轴号：第4轴
        50000,              // 跟随运动的位移量：50000 pulse。这里的位置参数是相对位置。
        0);                 // 向坐标系1的FIFO0缓存区传递该直线插补数据
    // 向缓存区写入一段圆弧插补数据，该段数据是以圆心描述方法描述了一个半圆

    sRtn = GTN_ArcXYC(1,

        1,                  // 坐标系是坐标系 1
        -200000, 0,         // 该圆弧的终点坐标(-200000, 0)
        -200000, 0,         // 圆弧插补的圆心相对于起点位置的偏移量(-200000, 0)
        0,                  // 该圆弧是顺时针圆弧
        100,                // 该插补段的目标速度：100pulse/ms
        0.1,                // 该插补段的加速度：0.1pulse/ms^2
        0,                  // 终点速度为0
        0);                 // 向坐标系1的FIFO0缓存区传递该直线插补数据


    do
    {
        // 坐标系在运动，查询到的run的值为1

        sRtn = GTN_CrdStatus(1,1, &run, &segment, 0);

    }while(1 == run);
…… …… ……
```

例程的运行结果如图 7-26 所示。

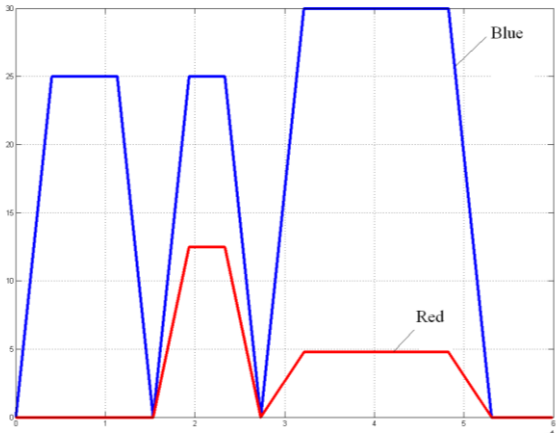


图 7-26　插补缓存区内的跟随运动速度图

其中蓝色为插补运动的合成速度，红色为跟随运动轴的速度值，跟随轴的速度跟随插补运动的合成速度的变化而变化。

在使用插补缓存区中的跟随运动功能时，需要注意以下内容：

1) GTN_BufGear 指令需要在所要跟随的插补段前，不要间隔其他种类的指令，可以同时调用多个 GTN_BufGear 指令来实现多个轴跟随插补运动。

2) 当暂停坐标系运动时，插补的合成速度减速到 0，跟随轴的速度也会为 0，如果希望重新恢复坐标系运动时，跟随轴仍能够实现跟随，不要调用 GTN_Stop 指令停止跟随轴的运动，否则重新启动插补缓存区的运动时，跟随轴将无法完成正在进行的跟随运动。

例程 7-10 刀向跟随功能——实际工件加工

假设机床加工过程中，机床需要加工一个工件，XY 平面的尺寸如图 7-27 所示，工件的厚度为 500（单位均为：pulse），采用的工艺是用砂轮磨削工件外轮廓，加工时不但要调整刀具的 Z 方向高度，同时需要调整砂轮的 C 向角度（假设砂轮旋转一周的脉冲总数是 10000）。在加工工艺中，首先为避免撞到工件，需要将砂轮抬到一定的高度:2000（假设安全高度为 2000），从原点空走到 A(6000, 6000)位置，之后调整砂轮逆时针转 45°（对应为 1250 pulse），使之与工件的加工切向方向一致，然后降低砂轮高度到加工位置:-100。接着，沿直线方向加工到 B(6000, 12000)位置，从 B 到 C 为一圆弧，需要实时更新砂轮的方向，使之保持与工件的加工切向方向一致，所以需要调用 GTN_BufGear 指令，跟随的位移量是 2500（90°）。随之后面的加工与之前类似，直至到加工点 F(36000, 6000)位置，此时需要抬刀使砂轮改变 90°，使之与 FA 段的加工方向一致，之后再放下砂轮至加工位置，直至加工完工件。

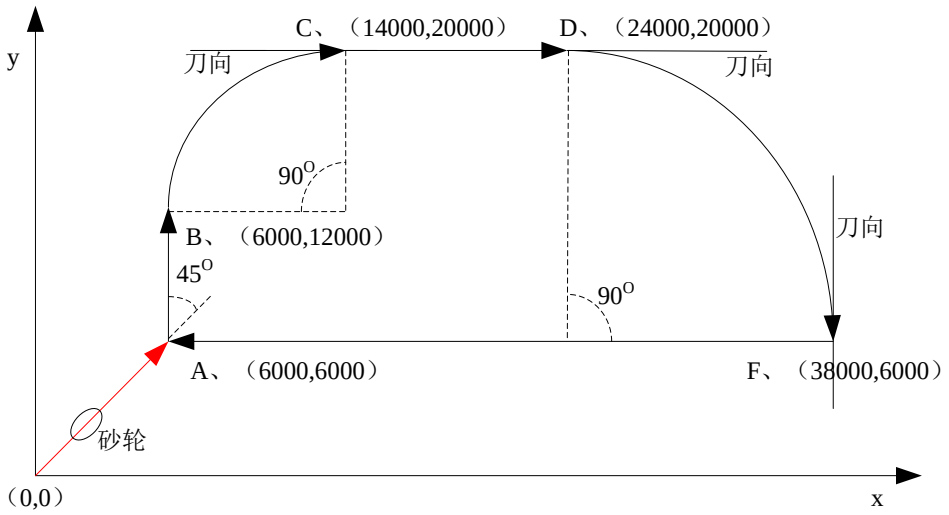


图 7-27 刀向跟随功能之工件尺寸和刀运动轨迹

假设 1 和 2 轴为 XY 轴，3 轴为 Z 轴，4 轴为 C 轴，并且 C 向初始角度为 0°，加工程序代码实现如下：

```
…… …… ……
// 定义指令返回值变量
short sRtn;
// 定义坐标系运动状态查询变量
short run;
// 定义坐标系运动完成段查询变量
long segment;
// 清除坐标系1的FIFO0中的数据

sRtn = GTN_CrdClear(1,1, 0);
```

```
// 向FIFO0缓存区写入一段直线插补数据
sRtn = GTN_BufMove(1,
    1,          // 该插补段的坐标系是坐标系1
    3,          // 点位运动的轴号：第3轴
    500,        // 点位运动的目标位置：500 pulse
    10,         // 点位运动的目标速度：10 pulse/ms
    0.1,        // 点位运动的目标加速度：0.1 pulse/(ms*ms)
    1,          // 该点位运动是模态指令，等砂轮抬高后才执行下面的指令
    0);         // 向坐标系1的FIFO0缓存区传递该直线插补数据
// 直线插补指令，到达A点
sRtn = GTN_LnXY(1,1, 6000, 6000, 50, 0.1, 0, 0);
sRtn = GTN_BufMove(1,
    1,          // 该插补段的坐标系是坐标系1
    4,          // 点位运动的轴号：第4轴
    -1250,      // 使其逆时针旋转45 °与BA方向一致
    10,         // 点位运动的目标速度：10 pulse/ms
    0.1,        // 点位运动的目标加速度：0.1 pulse/(ms*ms)
    1,          // 该点位运动是模态指令，等角度到位才后才执行下面的指令
    0);         // 向坐标系1的FIFO0缓存区传递该直线插补数据
// 模态指令，将砂轮放下到加工高度位置
sRtn = GTN_BufMove(1,1, 3, -100, 10, 0.1, 1, 0);
// 直线插补指令，到达B点
sRtn = GTN_LnXY(1,1, 6000, 12000, 50, 0.1, 0, 0);
sRtn = GTN_BufGear(1,
    1,          // 该插补段的坐标系是坐标系1
    4,          // 跟随运动的轴号：第4轴
    2500,       // 跟随运动的位移量：2500 pulse（相应为90 °）
    0);         // 向坐标系1的FIFO0缓存区传递该直线插补数据
// 到达C点位置
sRtn = GTN_ArcXYR(1,
    1,          // 坐标系是坐标系1
    0, 14000,   // 该圆弧的终点坐标(0, 200000)
    8000,       // 半径：8000pulse
    0,          // 该圆弧是顺时针圆弧
    50,         // 该插补段的目标速度：50pulse/ms
    0.1,        // 该插补段的加速度：0.1pulse/ms^2
    0,          // 终点速度为0
    0);         // 向坐标系1的FIFO0缓存区传递该直线插补数据
// 直线插补指令，到达D点
sRtn = GTN_LnXY(1,1, 24000, 20000, 50, 0.1, 0, 0);
```

```
// 跟随运动的位移量：2500 pulse（相应为90°）
sRtn = GTN_BufGear(1,1, 4, 2500, 0);
// 到达F点
sRtn = GTN_ArcXYR(1,1, 38000, 6000, 14000, 0, 50, 0.1, 0, 0);
// 模态指令，将砂轮抬高到安全高度位置
sRtn = GTN_BufMove(1,1, 3, 500, 10, 0.1, 1, 0);
// 模态指令，将砂轮抬旋转至与FA方向一致
//该位置相对初始位置为225°，即目标位置6250
sRtn = GTN_BufMove(1,1, 4, 6250, 10, 0.1, 1, 0);
// 模态指令，将砂轮放下到加工高度位置
sRtn = GTN_BufMove(1,1, 3, -100, 10, 0.1, 1, 0);
// 直线插补指令，到达A点
sRtn = GTN_LnXY(1,1, 6000, 6000, 50, 0.1, 0, 0);
// 启动运动
sRtn = GTN_CrdStart(1,1, 0);
do
{
    sRtn = GTN_CrdStatus(1,1, &run, &segment, 0);
// 坐标系在运动，查询到的 run 的值为 1
    }while(run == 1);
…… …… ……
```

7. 缓存区 FIFO 的管理（运动暂停与恢复）

每个坐标系包含两个缓存区(FIFO)：FIFO0 和 FIFO1，其中 FIFO0 为主要运动 FIFO，FIFO1 为辅助运动 FIFO，每个 FIFO 都含有 4096 段插补数据的空间。

缓存区的释放：用户如果不使用插补模式，直接切换到其他运动模式，插补模式的缓存区就自动释放了。如调用指令 GTN_PrfJog 即可。

在运动控制器的插补模式下，不能随意切换到其他运动模式，否则会导致插补坐标系破坏，并且原来压入插补缓存区的数据会丢失。但是在实际应用中，经常会有类似下面例子描述的情况。假如机床在走一段轨迹的途中需要暂停下来，更换路径换刀或者移到安全的地方以查看加工效果，然后再回到暂停时的坐标继续完成剩余的轨迹。为了实现上述操作，应利用每个坐标系提供的两个缓存区 FIFO0 和 FIFO1。

FIFO0 是主运动 FIFO，用户的主体插补运动的插补数据应该放在 FIFO0 中。FIFO0 的插补运动可以被中断（通过调用 GTN_Stop 指令），中断后可以进行辅助 FIFO1 的插补运动，辅助 FIFO1 的插补运动完成后，需要将坐标系位置恢复到 FIFO0 主运动被打断的位置，之后 FIFO0 可从断点处继续恢复原来的运动（恢复运动调用 GTN_CrdStart 指令）。

FIFO1 是辅助运动 FIFO，用户的辅助插补运动的插补数据可以放在 FIFO1 中，FIFO1 的插补数据必须在 FIFO0 的运动停止的情况下才能输入，如果 FIFO0 在运动，向 FIFO1 中传递插补数据，将会提示错误。FIFO1 的插补运动也可以暂停、恢复，但是，在 FIFO1 暂停时不可以进行 FIFO0 的插补运动，否则，FIFO1 缓存区将会被清空，不可以再恢复 FIFO1 的运动，插补主运动与辅助运动流程如图 7-28 所示。

注意	在主运动 FIFO0 的运动暂停之后，又进行了 FIFO1 的运动，如果用户希望在 FIFO1 运动结束之后，继续进行 FIFO0 的运动，则用户必须保证，FIFO1 运动结束后，坐标位置值与 FIFO0 停止时的坐标位置值(断点位置)相同，否则，FIFO0 将不接受启动运动指令，调用 GTN_CrdStart 指令恢复启动 FIFO0 的运动，将会提示错误。

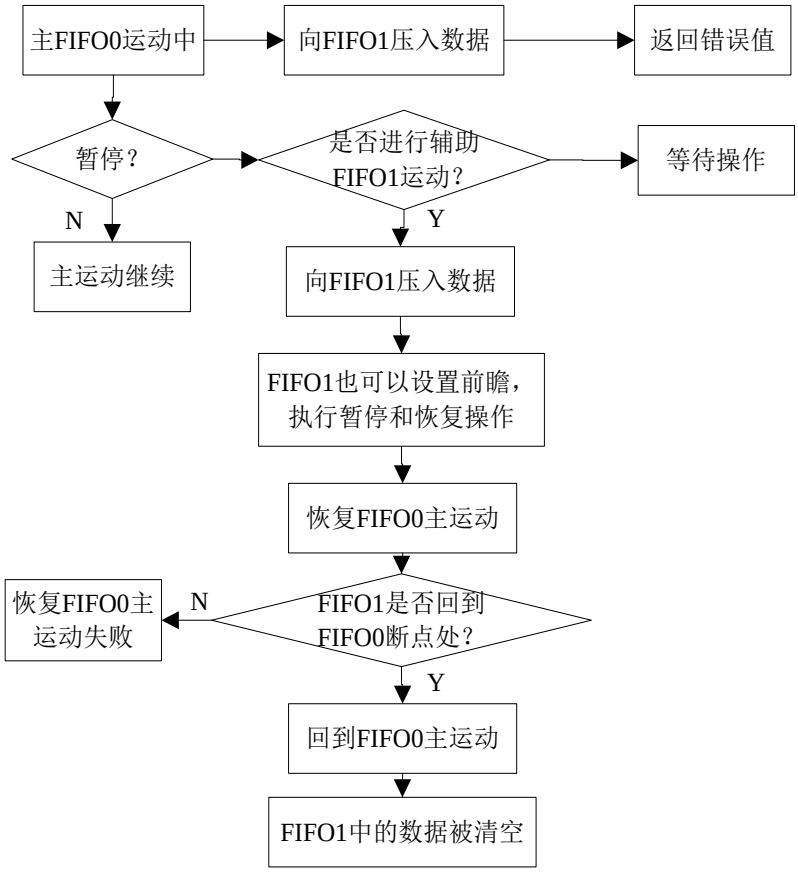


图 7-28　插补主运动与辅助运动流程

例程 7-11　插补 FIFO 管理

假设机床需要做运动：主加工运动需要从(0, 0)运动到(100000, 100000)位置，中途在(50000, 50000)附近出现了断刀，需要暂停运动去换刀。由于主加工运动已经将预定的轨迹数据压入了缓存区，若重新压入新的数据则会破坏原来的运动，因此需要启动辅助运动来协助完成。

假设换刀轨迹是先走到(70000, 30000)，然后走到(110000, 50000)位置完成换刀动作。但为了能继续主加工运动，需要回到暂停点，如图 7-29 所示。

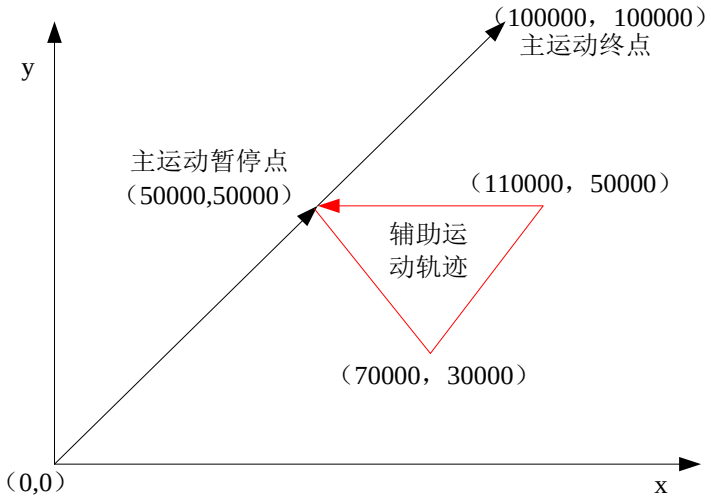


图 7-29　插补 FIFO 管理例程之换刀轨迹

```
…… …… ……
// 函数返回值
short sRtn;
// 循环变量
short i;
// 查询坐标系运行标志
short run;
// 查询坐标系运行的段数
long seg;
// 本程序局部运行标志，主运动启动时为1，停止时为0
short flag;
// 坐标系规划位置
double crdPos[2];
// 坐标系暂停位置
double crdstoppos[2];
// 坐标系结构体
TCrdPrm crdPrm;

// 清除fifo数据

sRtn = GTN_CrdClear(1,1, 0);

sRtn = GTN_CrdClear(1,1, 1);


// 向FIFO0缓存区写入主运动插补数据

sRtn = GTN_LnXY(1,

    1,                      // 该插补段的坐标系是坐标系1
    100000, 100000,         // 该插补段的终点坐标(100000, 100000)
    10,                     // 该插补段的目标速度：10pulse/ms
    1,                      // 插补段的加速度：1pulse/ms^2
    0,                      // 终点速度为0
    0);                     // 向坐标系1的FIFO0缓存区传递该直线插补数据
```

```
// 启动主运动，并将标志flag置1
sRtn = GTN_CrdStart(1,1, 0);
flag = 1;

do
{
    // 查询插补坐标位置
    sRtn = GTN_GetCrdPos(1,1, crdPos);
    // 当X轴的坐标位置大于50000时，暂停
    if(crdPos[0] > 50000.0)
    {
        // 停止坐标系1的运动
        sRtn = GTN_Stop(1,1, 1);
        // 注意:要等坐标系真正停止下来去获取当前的位置
        do
        {
            // 读取坐标系的状态，查看是否停止
            sRtn = GTN_CrdStatus(1,1, &run, &seg, 0);
        }while(run == 1);
        // 获取暂停后当前的位置，并将标志 flag 置 0，退出 while 循环
        sRtn = GTN_GetCrdPos(1,1, crdstoppos);
        flag = 0;
    }
    printf("crdPos[0]=%-10.1lf      crdPos[1]=%-10.1lf\r", crdPos[0], crdPos[1]);
}while(flag == 1);

// 向FIFO1缓存区写入辅助运动插补数据
sRtn = GTN_LnXY(1,1, 70000, 30000, 10, 1, 0, 1);
sRtn = GTN_LnXY(1,1, 110000, 50000, 10, 1, 0, 1);
// 启动坐标系1的FIFO1中运动
sRtn = GTN_CrdStart(1,1, 1);
do
{
    // 查询插补规划位置
    sRtn = GTN_GetCrdPos(1,1, crdPos);
    // 等待辅助运动完毕
    sRtn = GTN_CrdStatus(1,1, &run, &seg, 1);
```

```
        printf("crdPos[0]=%-10.1lf      crdPos[1]=%-10.1lf\r", crdPos[0], crdPos[1]);
    }while(1 == run);

    // 向FIFO1压入暂停后的位置点
    sRtn = GTN_LnXY(1,1, (long)crdstoppos[0], (long)crdstoppos[1], 10, 1, 0, 1);
    // 启动回暂停点运动
    sRtn = GTN_CrdStart(1,1, 1);
    do
    {
        // 查询插补规划位置
        sRtn = GTN_GetCrdPos(1,1, crdPos);
        // 等待回到暂停点位置运动完毕
        sRtn = GTN_CrdStatus(1,1, &run, &seg, 1);
        printf("crdPos[0]=%-10.1lf      crdPos[1]=%-10.1lf\r", crdPos[0], crdPos[1]);
    }while(1 == run);

    // 恢复主运动
    sRtn = GTN_CrdStart(1,1, 0);
    do
    {
        // 获取当前的位置
        sRtn = GTN_GetCrdPos(1,1, crdPos);
        // 等待主运动完毕
        sRtn = GTN_CrdStatus(1,1, &run, &seg, 0);
        printf("crdPos[0]=%-10.1lf      crdPos[1]=%-10.1lf\r", crdPos[0], crdPos[1]);
    }while(1 == run);
```

8. 插补指令快速传输功能

为了提高插补指令的传输速率，增加 DMA 通道。

1. DMA 功能为数据段批次压入功能，每次压入的数据段的数量由用户设定，压完数据段之后，再调用 GTN_CrdData 指令将剩余数据段压入缓冲区。

2. 开启 DMA 功能后，GTN_CrdSpace 将查询的为 PC 端的剩余数据段空间（共有 4096 段空间）。

3. 插补、振镜和位置比较输出等功能如果需要同时开启 DMA 功能时，需要选择不同的 DMA 缓存区序号。

例程 7-12 开启 DMA 快速通道

```
    // 指令返回值变量
    short sRtn;
    long pos[2] = {0,0};
    // TCrdPrm结构体变量，该结构体定义了坐标系
    TCrdPrm crdPrm;
    // 将结构体变量初始化为0
    memset(&crdPrm, 0, sizeof(crdPrm));
    // 为结构体赋值
    crdPrm.dimension=2;          // 坐标系为二维坐标系
    crdPrm.synVelMax=500;            // 最大合成速度：500pulse/ms
    crdPrm.synAccMax=1;          // 最大加速度：1pulse/ms^2
    crdPrm.evenTime = 50;        // 最小匀速时间：50ms
    crdPrm.profile[0] = 1;               // 规划器1对应到X轴
    crdPrm.profile[1] = 2;               // 规划器2对应到Y轴
    crdPrm.setOriginFlag = 1;            // 表示需要指定坐标系的原点坐标的规划位置
    crdPrm.originPos[0] = 100;           // 坐标系的原点坐标的规划位置为（100, 100）
    crdPrm.originPos[1] = 100;
    // 建立1号坐标系，设置坐标系参数

    sRtn =   GTN_SetCrdPrm(1,1, &crdPrm);

    …… …… ……

//使能DMA功能

sRtn = GTN_CrdHsOn(1,1,0,1,200,0);

//压入数据
for(int i = 0;i<221;i++)
{
    pos[0] += 100 ;
    pos[1] += 100;

    sRtn = GTN_LnXY(1,1,pos[0],pos[1],100,0.1,0);

}
do
{

    sRtn =   GTN_CrdData(1,1,NULL,0);

} while (rtn == 1);
```

第8章 访问硬件资源

8.1 本章简介

提示	本手册中所有字体为蓝色的指令（如 GTN_PrfTrap）均带有超级链接，点击可跳转至指令说明。

运动控制器包含的硬件资源分为如下几类：

表 8-1 运动控制器硬件资源

资源	说明
数字量输入	正负限位、驱动报警、原点、通用输入
数字量输入	电机到位信号输入（arrive）
数字量输出	报警清除、伺服使能、通用输出
编码器	对外部编码器的脉冲输出进行计数
模拟量输出（DAC）	电压输出
模拟量输入（ADC）	电压输入

本章介绍如何在应用程序中使用这些硬件资源。

8.2 访问数字 IO

8.2.1 指令列表

表 8-2 访问数字 IO 指令列表

指令	说明	页码
GTN_GetDi	读取数字 IO 输入状态	192
GTN_GetDiBit	按位读取数字 IO 输入状态	193
GTN_GetDiEx	读取多组数字 IO 输入状态	193
GTN_GetDiRaw	读取数字 IO 输入状态的原始值	194
GTN_GetDiReverseCount	读取数字量输入信号的变化次数	194
GTN_SetDiReverseCount	设置数字量输入信号的变化次数的初值	239
GTN_SetDo	设置数字 IO 输出状态	239
GTN_SetDoEx	设置多组数字 IO 输出状态	241
GTN_SetDoBit	按位设置数字 IO 输出状态	240
GTN_SetDoBitReverse	使数字量输出信号输出定时脉冲信号	240
GTN_GetDo	读取数字 IO 输出状态	195
GTN_SetSense	设置输入输出资源的电平逻辑	251
GTN_GetSense	读取输入输出资源的电平逻辑	208

8.2.2 重点说明

调用 GTN_GetDi 指令可以读取限位、驱动报警、原点、通用输入、手轮接口这些数字量输入接口的输入电平状态。其中，手轮接口为 5V 电平输入，其余 IO 为 24V 电平输入。

调用 GTN_SetDo 指令可以设置伺服使能、报警清除、通用输出这些数字量输出接口的输出电平状态。

调用 GTN_GetDiEx 和 GTN_SetDoEx 指令可以读取和设置多路 IO 状态。

调用 GTN_GetDiRaw 指令可以读取数字量输入接口的原始电平状态。

调用 GTN_GetDiReverseCount 指令可以读取数字量输入的变化次数，当数字量输入由 0 变为 1，或者由 1 变为 0，该次数就会增加一次。 GTN_SetDiReverseCount 指令用来设置数字量变化次数计数器的初值。

调用 GTN_SetDoBitReverse 指令可以使数字量输出信号输出一个定时的脉冲，例如，假设当前通用数字量输出信号 1 是高电平，当调用指令 GTN_SetDoBitReverse(MC_GPO, 1, 0, 100);则该数字量信号将会发出一个 100×period(中断周期)时间宽度的负脉冲。

下面详细说明专用数字量 IO 的含义和用途。

1. 正负限位（di）

控制器为每个轴提供两个输入作为行程控制开关。如图 8-1 所示。

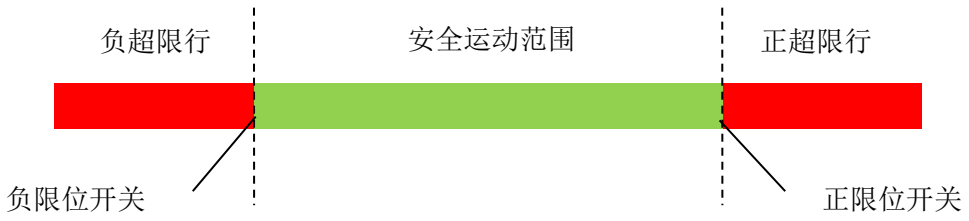


图 8-1 限位开关示意图

把轴配置成正负限位有效后，如果轴运动超越了限位开关所在位置，限位开关触发，运动控制器禁止触发限位方向上运动，同时该轴的限位触发状态置 1（通过调用 GTN_GetSts 可以查看此状态标志位）。离开限位回到安全运动范围以后，需要调用指令 GTN_ClrSts 清除限位触发状态，才能使控制轴回到正常运动状态。

限位触发后，轴紧急停止，调用 GTN_GetSts 读取状态，会发现正限位触发标志位或负限位触发标志位置 1。用户应按如下步骤操作：

1. 调用指令让轴向反方向运动到安全范围内停下。即如果正限位触发就让轴往负方向运动，负限位触发就让轴往正方向运动。
2. 调用 GTN_ClrSts 清除限位触发状态。
3. 限位触发后的紧急停止可能会因为运动过冲而引入位置误差，建议用户重新回零。回零方法详见 GTN_GoHome。

如果用户不使用限位开关，可以通过控制器配置，将轴的正负限位配置为无效。

限位开关默认为高电平触发。通过调用 GTN_SetSense 指令可以修改限位开关的触发电平。

2. 驱动器报警信号（di）

控制器提供专用的驱动报警信号输入接口。该接口应该与驱动器的报警输出信号接口相接。当检测到驱动器报警信号以后，运动控制器将关闭该轴的伺服使能，急停运动规划，同时该轴报警触发标志置 1（通过调用 GTN_GetSts 可以查看此状态标志位）。

驱动报警信号是低电平触发。

驱动报警后，轴紧急停止，调用 GTN_GetSts 读取状态，会发现驱动器报警标志位置 1。用户应当执行以下操作：

1. 确定引起驱动器报警的原因，并加以改正。
2. 复位驱动器。
3. 调用 GTN_ClrSts 清除报警，重新回机床原点。

3. 原点信号（di）

原点信号又叫 Home 信号，即电机回到原点时控制它停止的信号。用户负责在应用程序中捕获到 Home 信号，然后控制电机停止，具体操作请参考“第 8 章 访问硬件资源”。

原点信号默认为下降沿触发，通过调用 GTN_SetCaptureSense 指令可以修改原点开关的触发沿。

4. 伺服使能（do）

伺服使能信号默认与 axis 关联，用户不能直接调用 GTN_SetDo 或 GTN_SetDoBit 对伺服使能设置输出电平。默认情况下，调用 GTN_AxisOn，伺服使能输出低电平，该 do 置 0。如果驱动器是低电平使能，则必须在控制器配置将伺服使能输出设置为输出电平取反，具体操作请参考配置 do。

伺服使能信号的硬件接口在端子板 25pin 轴接口中可以找到（请参考硬件说明书）。如果用户使用的驱动器不含伺服使能输入信号，或者用户不需要使用此信号，可以取消伺服使能 do 与 axis 的关联（操作请参考配置 do），这样用户可以直接调用 GTN_SetDo 或 GTN_SetDoBit 对伺服使能设置输出电平。

5. 报警清除（do）

某些驱动器上有报警清除数字输入接口，控制器提供报警清除数字量输出接口，方便用户使用此功能。但是此信号不能清除所有的驱动报警情况。例如，在编码器接线不良引起驱动报警时，就无法使用清除报警信号来清除。因此，当报警无法清除时，用户需要具体问题具体分析。用户可以调用 GTN_SetDo 或 GTN_SetDoBit 来设置清除报警信号的电平。

8.2.3 例程

例程 8-1 访问数字 IO

有一个按键连接端子板上通用输入 EXI3，常为低电平，按下按键，输入高电平。端子板通用输出 EXO6 上连接一个指示灯，当 EXO6 输出低电平时指示灯亮，输出高电平时指示灯灭。开始指示灯一直灭，用户希望不断读取轴 1 正负限位的数字量输入变化次数，并将结果输出到显示器。在正负限位变化次数都超过 10 次之后归零重新计数，并且使指示灯一直亮起来。用户可以随时按键停止这种检测。

```
……
// 指令返回值
short sRtn;
// 通用输入读取值
long lGpiValue;
```

```
// 正限位输入变化次数
unsigned long lPosLmtReverseCount;
// 负限位输入变化次数
unsigned long lNegLmtReverseCount;

// 初始化变化次数为0
lPosLmtReverseCount = 0;
lNegLmtReverseCount = 0;

// 读取EXI3输入值

sRtn =  GTN_GetDi(1,MC_GPI, &lGpiValue);

// 此函数为检测指令返回值函数，参看例程3-1检测GT指令是否正常执行

commandhandler(" GTN_GetDi", sRtn);

// 如果为高电平，说明按键正在被按下，则不检测，返回 1
if( lGpiValue& (1<<3))
    return 1;
// EXO6输出高电平，使指示灯灭

sRtn =  GTN_SetDo(1,MC_GPO, 1<<6);

commandhandler(" GTN_SetDo", sRtn);

// 按键没有被按下，循环。如果EXI3输入值为高电平，即按键按下，则退出循环
while( ! (lGpiValue& (1<<3)))
{
    // 读取正限位输入变化次数

    sRtn = GTN_GetDiReverseCount(1

        MC_LIMIT_POSITIVE,      // 指定数字IO类型是正限位
        1,                      // 指定正限位1
        &lPosLmtReverseCount,   // 读取的值
        1);                     // 一次读取一个数字量

    commandhandler(" GTN_GetDiReverseCount", sRtn);

    // 读取负限位输入变化次数

    sRtn = GTN_GetDiReverseCount(1

        MC_LIMIT_NEGATIVE,      // 指定数字IO类型是负限位
        1,                      // 指定负限位1
        &lNegLmtReverseCount,   // 读取的值
        1);                     //一次读取一个数字量

    commandhandler(" GTN_GetDiReverseCount", sRtn);

    // 将结果输出到显示器
    printf("PosLmtReverseCount = %d\n NegLmtReverseCount = %d\n",
        lPosLmtReverseCount, lNegLmtReverseCount);
```

```
    // 如果正负限位的输入变化次数都超过10次
    if(( lPosLmtReverseCount>= 10) &&( lNegLmtReverseCount>= 10) )
    {
        // 重新归为0
        lPosLmtReverseCount = 0;
        lNegLmtReverseCount = 0;
        // 设置正限位输入变化次数

        sRtn = GTN_SetDiReverseCount(1,

            MC_LIMIT_POSITIVE,      // 指定数字IO类型是正限位
            1,                      // 指定正限位1
            &lPosLmtReverseCount,   // 读取的值
            1);                     // 一次读取一个轴

        commandhandler(" GTN_SetDiReverseCount", sRtn);

        // 设置负限位输入变化次数

        sRtn = GTN_SetDiReverseCount(1,

            MC_LIMIT_NEGATIVE,      // 指定数字IO类型是负限位
            1,                      // 指定负限位1
            &lNegLmtReverseCount,   // 读取的值
            1);                     // 一次读取一个轴

        commandhandler(" GTN_SetDiReverseCount", sRtn);

        // EXO6输出高电平，使指示灯亮

        sRtn = GTN_SetDoBit(1,

            MC_GPO,                 // 指定数字IO类型是通用输出
            7,                      // 指定第7个通用输出，即EXO6
            0);                     // 输出低电平

        commandhandler(" GTN_SetDoBit", sRtn);

    }
    // 不断读取通用输入，已检测EXI3的电平状态

    sRtn = GTN_GetDi(1,MC_GPI, &lGpiValue);

    commandhandler(" GTN_GetDi" , sRtn);

}
……
```

8.3 访问编码器

8.3.1 指令列表

表 8-3 访问编码器指令列表

指令	说明	页码
GTN_GetEncPos	读取轴编码器位置	195
GTN_GetEncVel	读取轴编码器速度	196
GTN_SetEncPos	修改轴编码器位置	241
GTN_ReadAuEncPos	读取辅助编码器位置	231
GTN_ReadMpgInfo	读取手轮信息	231
GTN_WriteAuEncPos	设置辅助编码器位置	259
GTN_WriteMpgPos	设置手轮编码器位置	259

8.3.2 重点说明

控制器编码器包括轴编码器、辅助编码器和手轮。轴编码器指端子板模块 25pin 轴接口对应的编码器，辅助编码器指端子板模块上的 ENCX（例如 ENC1）接口，手轮指端子板模块上的 MPGX（例如：MPG1）接口。

1、 轴编码器

控制器内部为每个轴配置了脉冲计数装置。控制器默认的脉冲计数源是外部编码器。如果用户在接线时将外部编码器的信号与端子板 25pin 轴接口的编码器信号接在一起，就可以调用上述指令读取外部编码器的值。如果用户没有接外部编码器反馈信号，例如，使用步进电机时没有编码器反馈部件，如图 8-2 所示，则用户调用 GTN_GetEncPos 读取的编码器位置为 0。

图 8-2 开环控制系统示意图

控制器还可以配置脉冲计数源是脉冲计数器（操作详见配置 step）。调用 GTN_GetEncPos 读取的将是运动控制器向驱动器发出的脉冲个数。因此，即使不接反馈部件，也可以读取变化的位置值。

调用 GTN_SetEncPos 修改编码器位置的值。例如，设置轴 1 的编码器位置为 0，则接下来的编码器计数从 0 开始。若设置为 1000，则从 1000 开始。

2、 辅助编码器

调用指令 GTN_ReadAuEncPos 和 GTN_WriteAuEncPos 读取和设置辅助编码器位置。

3、 手轮

调用指令 GTN_ReadMpgInfo 读取手轮编码器位置、编码器速度和 DI。调用指令 GTN_WriteMpgPos 设置手轮编码器位置。

8.3.3 例程

例程 8-2 读取 8 个轴编码器位置值，1 个辅助编码器位置和 1 个手轮信息

```
#include "gts.h"
int main(int argc, char* argv[])
{
    short sRtn, i;
    double enc[8];
    double auEnc;
    TMpgInfo mpgInfo;

    sRtn =  GTN_Open();

    commandhandler(" GTN_Open", sRtn);

    while(1)
    {
        // 读取8个编码的位置
        sRtn =  GTN_GetEncPos(1,1, &enc[0], 8);
        for(i=0;i<8;++i)
        {
            printf("%8.0lf", enc[i]);
        }
        printf("\r");
      //读取第1路辅助编码器位置
        sRtn =  GTN_ReadAuEncPos(1,1,&auEnc);
        printf("%8.0lf", auEnc);
        printf("\r");
      //读取第1路MPG编码器位置和MPG-DI输入状态
        sRtn =  GTN_ReadMpgInfo(1,1,&mpgInfo);
        printf("%8.0lf,%d", mpgInfo.pos,mpgInfo.di);
    }
    return 0;
}
```

8.4 访问 DAC

8.4.1 指令列表

表 8-4 访问 DAC 指令列表

指令	说明	页码
GTN_SetDac	设置 DAC 输出电压	238
GTN_GetDac	读取 DAC 输出电压	192
GTN_SetAuDac	设置非轴模拟量输出(AUDAC)输出电压	232
GTN_GetAuDac	读取非轴模拟量输出(AUDAC)输出电压	184

8.4.2 重点说明

控制器的模拟量输出通道在闭环控制模式下，作为伺服电压输出控制通道，与驱动器连接，是不允许用户操作的。在开环控制模式下，可以作为通用的电压输出通道。上电复位后，控制器默认为脉冲（开环）控制模式，用户可以调用 DAC 指令列表中的指令来设置和读取 dac 输出电压。模块上轴接口对应的模拟电压值与指令读取数值对应关系如表 8-5 所示。

表 8-5 轴控模拟电压值与指令读取数值对应关系

输入电压值（V）	指令读取数值
-10	-32768
0	0
10	32767

例程 8-3 访问 DAC

轴 4 的 dac 输出通道输出 5V 电压，设置完后也可以读取该通道电压。

```
……
// 指令返回值
short sRtn;
// 电压值
short sSetValue;
short sGetValue;
// 控制器复位，所有轴都为脉冲控制模式
sRtn = GTN_Reset(1);
// 计算轴4的电压输出值
sSetValue = (short) 32767*5/10;
// 设置轴4的输出电压
sRtn = GTN_SetDac(1,4, &sSetValue, 1);
// 读取轴4的输出电压值
sRtn = GTN_GetDac(1,4, &sGetValue, 1);
……
```

8.5 访问模拟量输入(仅适用于带模拟量功能模块)

8.5.1 指令列表

表 8-6 访问模拟量输入指令列表

指令	说明	页码
GTN_GetAdc	读取模拟量输入的电压值	182
GTN_GetAdcValue	读取模拟量输入的数字转换值	183
GTN_GetAuAdc	读取非轴模拟量输入的电压值	183
GTN_GetAuAdcValue	读取非轴模拟量输入的数字转换值	184
GTN_SetAdcFilterPrm	设置模拟量输入的滤波参数	232
GTN_GetAdcFilterPrm	读取模拟量输入的滤波参数	183

8.5.2 重点说明

模拟量输入分为模拟量（硬件通道和轴 25Pin 接口在一起）和非轴模拟量（独立的硬件接口）。

调用指令 GTN_GetAdc 读取指定轴通道的外部输入模拟电压值，指令 GTN_GetAuAdc 读取指定非轴通道的外部输入模拟电压值。模拟电压值与指令读取数值对应关系如表 8-7 所示。

表 8-7 模拟电压值与指令读取数值对应关系

输入电压值（V）	指令读取数值
-10	-32768
0	0
10	32767

8.5.3 例程

例程 8-4 访问 ADC

将 5V 电源并联接入 4 个 adc 输入通道，读取该输入通道的电压值和数字转换值。

```
……
// 指令返回值
short sRtn;
// 电压值
double dGetVoltageValue[4];
// 数字转换值
short sGetDigitalValue[4];
// 读取4个通道的输入电压
sRtn = GTN_GetAdc(1,1, &dGetVoltageValue[0], 4);
// 读取4个通道输入电压的数字转换值
sRtn =  GTN_GetAdcValue(1,1, &sGetDigitalValue[0], 4);
……
```

8.6 访问内部脉冲计数

8.6.1 指令列表

表 8-8 访问内部脉冲计数输入指令列表

指令	说明	页码
GTN_SetPlsPos	设置内部脉冲计数器位置	246
GTN_GetPlsPos	读取内部脉冲计数器位置	201
GTN_GetPlsVel	读取内部脉冲计数器速度	202

8.6.2 重点说明

脉冲计数器是通过内部硬件计算输出的脉冲个数，调用指令 GTN_GetPlsPos 读取指定轴通道的输出脉冲个数。和 GTN_EncOff 中将脉冲计数源设置为脉冲计数器读取的位置一致，但是不需要占用编码器读

取通道，即可以通过相应的指令同时读取编码器位置和脉冲计数器位置，便于调试。默认状态下，脉冲计数器资源是关闭的，需要通过指令 GTN_SetResCount 打开相应的软件资源。

8.6.3 例程

例程 8-5 访问内部脉冲计数器

```
#include "gts.h"
int main(int argc, char* argv[])
{
    short sRtn, i;
    double plspos[12];
    sRtn = GTN_Open();
    commandhandler(" GTN_Open", sRtn);
    sRtn= GTN_SetResCount(1,MC_PULSE,12);
    commandhandler(" GTN_SetResCount", sRtn);
    while(1)
    {
        // 读取12个脉冲计数的位置
        sRtn =  GTN_GetPlsPos(1,1, &plspos[0],8);
        commandhandler(" GTN_GetPlsPos", sRtn);
        sRtn =  GTN_GetPlsPos(1,9, &plspos[8],4);
        commandhandler(" GTN_GetPlsPos", sRtn);
        for(i=0;i<12;++i)
        {
            printf("%8.0lf",plspos[i]);
        }
        printf("\r");
    }
    return 0;
}
```

第9章 高速硬件捕获

9.1 本章简介

捕获即当某一种信号触发时，运动控制器能准确记录触发时刻轴的位置信息。控制器提供三种捕获方式，Home 捕获，Index 捕获、探针(Probe)捕获。本章将详细介绍这三种捕获方式以及如何在应用程序中实现。

提示	本手册中所有字体为蓝色的指令（如 GTN_PrfTrap）均带有超级链接，点击可跳转至指令说明。

9.2 Trigger 硬件捕获

9.2.1 指令列表

表 9-1 高速硬件捕获指令列表

指令	说明	页码
GTN_SetTriggerEx	设置 Trigger 捕获方式，并启动捕获	254
GTN_GetTriggerEx	读取 Trigger 捕获方式	213
GTN_GetTriggerStatusEx	读取 Trigger 捕获状态	214
GTN_ClearTriggerStatus	清除捕获状态	175

9.2.2 重点说明

主卡两个核之间的 Trigger 互相独立，每个 Trigger 均可以设置为不同的捕获模式，可以锁存不同的编码器，相关功能详见指令详细说明。

注意事项：

1. Trigger、捕获源和捕获编码器需要在同一个模块；
2. 每个 Trigger 之间是相互独立的；
3. 默认每个核的 Trigger 个数配置情况和轴配置一致(6 轴模块对应 6 个 Trigger，4 轴模块对应 4 个)，按照模块连接顺序一一配置。

9.2.3 例程

例程 9-1 Home/Index 捕获

该例程描述了怎样设置 Trigger 捕获。用户如直接使用此例程仍无法实现整个捕获过程，则需要控制端子板上 home 数字量输入电平使 home 信号触发。

```
    #define ENCODER             1

// 该函数检测某条GT指令的执行结果，command为指令名称，error为指令执行返回值
void commandhandler(char *command, short error)
{
    // 如果指令执行返回值为非，说明指令执行错误，向屏幕输出错误结果
    if(error)
    {
        printf("%s = %d\n", command, error);
    }
}

void TriggerIndex(void)
{
    short sRtn;
    double encPos;
    TTriggerEx trigger;
    TTriggerStatusEx status;

    // 启动Index捕获
    sRtn = GTN_GetTriggerEx(1, 1, &trigger);
    commandhandler("GTN_GetTriggerEx", sRtn);

    trigger.latchType = MC_ENCODER;
    trigger.latchIndex = ENCODER;
    trigger.firstPosition = 0;
    trigger.lastPosition = 0;
    trigger.loop = 1;
    trigger.offset = 0;
    trigger.probeIndex = 1;
    trigger.probeType = CAPTURE_INDEX;
    trigger.sense = 0;
    trigger.windowOnly = 0;

    sRtn = GTN_SetTriggerEx(1,
                            1,   // Trigger序号
                            &trigger);
    commandhandler("GTN_SetTriggerEx", sRtn);

    do
    {
        // 查询捕获状态
        sRtn = GTN_GetTriggerStatusEx(1, ENCODER, &status, 1);
        // 读取编码器位置
        // 该指令和捕获无关，仅用于显示编码器位置
        sRtn = GTN_GetEncPos(1, ENCODER, &encPos);
```

```
            // 显示捕获状态和编码器位置
            printf("execute = %d done = %d position = %lx loop = %d enc = %-8.0lf\r", status.execute, status.done,
status.position, status.loopCount, encPos);
            // 当指定轴捕获触发时退出循环
        }while(!status.done);
        // 显示捕获位置
        printf("\ncapture = %-8ld\n", status.position);
}

void TriggerHome(void)
{
        short sRtn;
        double encPos;
        TTriggerEx trigger;
        TTriggerStatusEx status;

        // 启动Home捕获
        sRtn = GTN_GetTriggerEx(1, 1, &trigger);
        commandhandler("GTN_GetTriggerEx", sRtn);

        trigger.latchType = MC_ENCODER;
        trigger.latchIndex = ENCODER;
        trigger.firstPosition = 0;
        trigger.lastPosition = 0;
        trigger.loop = 1;
        trigger.offset = 0;
        trigger.probeIndex = 1;
        trigger.probeType = CAPTURE_HOME;
        trigger.sense = 0;
        trigger.windowOnly = 1;

        sRtn = GTN_SetTriggerEx(1,
                                1,   // Trigger序号
                                &trigger);
        commandhandler("GTN_SetTriggerEx", sRtn);

        do
        {
            // 查询捕获状态
            GTN_GetTriggerStatusEx(1, ENCODER, &status, 1);
            // 读取编码器位置
            // 该指令和捕获无关，仅用于显示编码器位置
            GTN_GetEncPos(1, ENCODER, &encPos);
            // 显示捕获状态和编码器位置
            printf("execute = %d done = %d position = %lx loop = %d enc = %-8.0lf\r", status.execute, status.done,
status.position, status.loopCount, encPos);
```

```
        // 当指定轴捕获触发时退出循环
    }while(!status.done);
    // 显示捕获位置
    printf("\ncapture = %-8ld\n", status.position);
}
```

9.3 AuTrigger 硬件捕获

9.3.1 指令列表

表 9-2 高速硬件捕获指令列表

指令	说明	页码
GTN_SetAuTrigger	设置 AuTrigger 捕获方式，并启动捕获	254
GTN_GetAuTrigger	读取 AuTrigger 捕获方式	213
GTN_GetAuTriggerStatus	读取 AuTrigger 捕获状态	214
GTN_ClearAuTriggerStatus	清除 AuTrigger 捕获状态	175

9.3.2 重点说明

主卡两个核之间的 Trigger 互相独立，每个 AuTrigger 均可以设置为不同的捕获模式，可以锁存不同的编码器，相关功能详见指令详细说明。

注意事项：

1. AuTrigger、捕获源和捕获编码器需要在同一个模块；
2. 每个 AuTrigger 之间是相互独立的，其和 Trigger 之间也是独立的；
3. 每个模块上 Trigger 捕获的数量和模块的轴数相同，而 AuTrigger 则是模块上“多出来”的捕获资源。

目前轴模块上没有 AuTrigger 资源。GSHD 伺服驱动器上有 1 路 Trigger，5 路 AuTrigger。

按照下图的连接方式：

第一个 GSHD 上的 Trigger 的索引为 1，第二个 GSHD 上的 Trigger 的索引为 2；

第一个 GSHD 上的 AuTrigger 的索引为[1, 5]，第二个 GSHD 上的 AuTrigger 的索引为[6, 10]；

9.3.3 例程

例程 9-2

9.4 回零功能

9.4.1 指令列表

表 9-3 Smart Home 功能指令列表

指令	说明	页码
GTN_GoHome	启动 Smart Home 实现各种方式回原点	218
GTN_GetHomePrm	读取设置到控制器的 Smart Home 回原点参数	198
GTN_GetHomeStatus	获取 Smart Home 回原点的状态	199

9.4.2 重点说明

表 9-4 术语解释表

术语、缩写	解释
Home	原点
Index	编码器转动一圈产生的 Z 相信号（也称为 C 相信号）

Smart Home 是运动控制器的“Home/Index 回原点”和“自动回原点”的优化和扩展。Smart Home 仍然采用高速硬件捕获机制实现回原点，把原来较为繁琐的回零过程固化到控制器，只需要调用简单指令就能够实现回原点，简化了用户程序。此外，Smart Home 汲取了工控界较为常见的回原点方式，集成到控制器给予实现，具体包括以下回原点方式：

1、 限位回原点

（1） **电机位置在限位信号外：**调用回原点指令，电机位置在限位信号外，电机从所在位置以较高的速度往限位方向运动，如果碰到限位，则反方向运动，脱离限位后再以较低的速度往限位方向运动，触发限位后停止运动，此处即为原点。（这种回原点方式没有用到高速硬件捕获功能，适用于对回原点精度要求不高或者不易于安装 Home 开关的场合）。

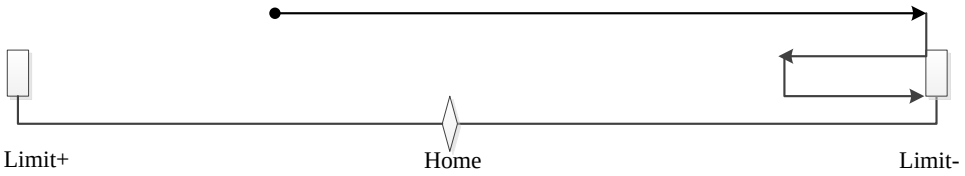


图 9-1-1 限位回原点示意图（当前位置不在限位上）

（2） **电机位置在限位信号上：**调用回原点指令，电机位置在限位信号上，电机先进行脱离限位的运动，电机从所在位置以较高的速度，目标位置为用户设置的脱离步长，moveDir 的反方向运动，尝试脱离限位，如果运动结束，限位信号没有置起来，则再次从所在位置以较高的速度，目标位置为用户设置的脱离步长，moveDir 的反方向运动，直至脱开限位，否则一直运动；

如果运动结束，已经脱开限位，则开始正常的回零运动，以较高的速度往 moveDir 方向运动，触发限位后停止运动，再以较高的速度，目标位置为用户设置的脱离步长，moveDir 的反方向运动，如果运动结束，已经脱开限位，则以较低的速度往 moveDir 方向运动，触发限位后停止运动，如果 homeOffset 为 0，则停止运动，此处即为原点，如果 homeOffset 不为 0，则以较低的速度运动到用户设置的 homeOffset 处，此处即为原点。（这种回原点方式没有用到高速硬件捕获功能，适用于对回原点精度要求不高或者不易于安装 Home 开关的场合）。

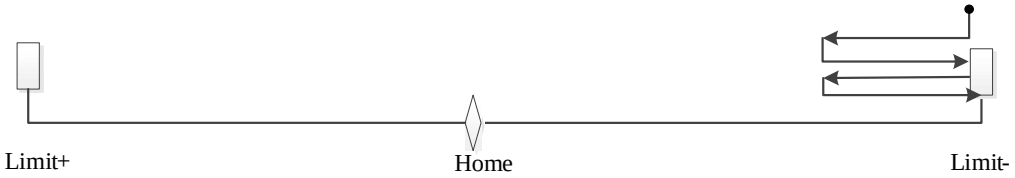


图 9-2-2 限位回原点示意图（当前位置在限位上）

2、 限位+Home 回原点

（1） 电机位置在限位信号外：调用回原点指令，电机位置在限位信号外，电机从所在位置以较高的速度往限位方向运动，如果碰到限位，则反方向运动并启动高速硬件捕获，在设定的搜索范围内搜索 Home，当触发 Home 开关后，电机会以较低的速度运动到捕获的位置处（即 Home 开关）。

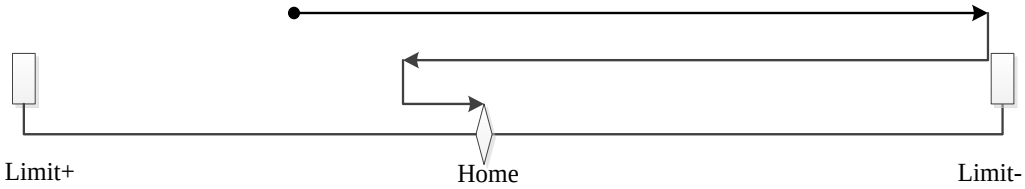


图 9-3-1 限位+Home 回原点示意图（当前位置不在限位上）

（2） 电机位置在限位信号上：调用回原点指令，电机位置在限位信号上，电机先进行脱离限位的运动，电机从所在位置以较高的速度，目标位置为用户设置的脱离步长，moveDir 的反方向运动，尝试脱离限位，如果运动结束，限位信号没有置起来，则再次从所在位置以较高的速度，目标位置为用户设置的脱离步长，moveDir 的反方向运动，直至脱开限位，否则一直运动。

如果运动结束，已经脱开限位，则以较高的速度往 moveDir 方向运动，触发限位后停止运动，再反方向运动并启动高速硬件捕获，在设定的搜索范围内搜索 Home，当触发 Home 开关后，电机会以较低的速度运动到捕获的位置处（即 Home 开关）。

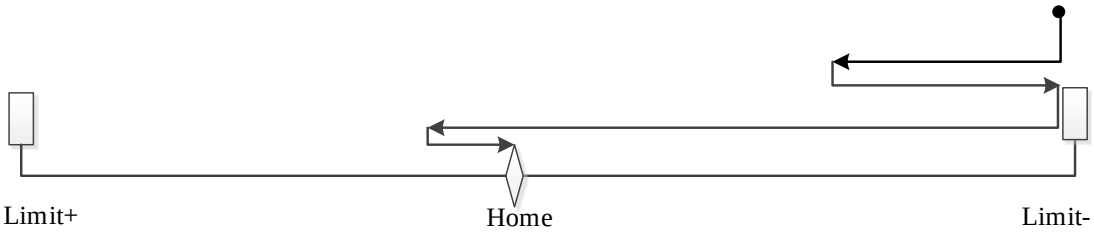


图 9-4-2 限位+Home 回原点示意图（当前位置在限位上）

3、 限位+Index 回原点

（1） 电机位置在限位信号外：调用回原点指令，电机从所在位置以较高的速度往限位方向运动，触发限位后再反方向以较低的速度运动并启动高速硬件捕获寻找 Index，捕获到编码器的 Index 信号后，电机会运动到捕获的位置处（编码器 Index 位置）。

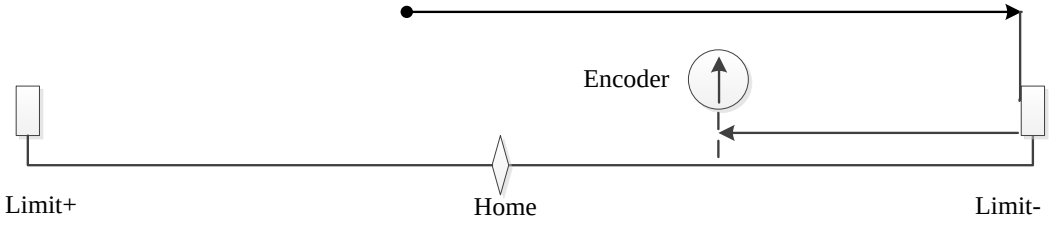


图 9-5-1 限位+Index 回原点示意图（当前位置不在限位上）

（2） **电机位置在限位信号上：**调用回原点指令，电机位置在限位信号上，电机先进行脱离限位的运动，电机从所在位置以较高的速度，目标位置为用户设置的脱离步长，moveDir 的反方向运动，如果运动结束，限位信号没有置起来，则再次从所在位置以较高的速度，目标位置为用户设置的脱离步长，moveDir 的反方向运动，直至脱开限位，否则一直运动。

如果运动结束，已经脱开限位，则以较高的速度往 moveDir 方向运动，触发限位后停止运动，再以较低的速度反方向运动并启动高速硬件捕获，在设定的搜索范围内寻找 Index，捕获到编码器的 Index 信号后，电机会运动到捕获的位置处（编码器 Index 位置）。

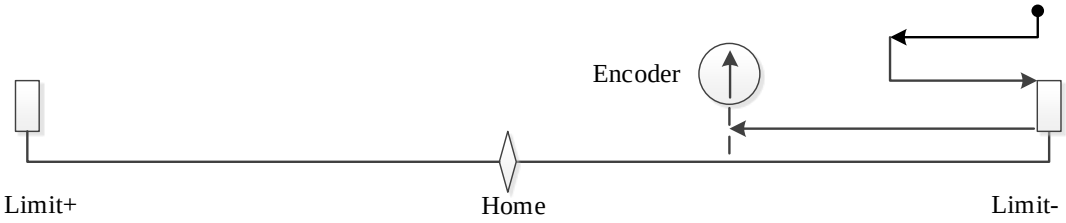


图 9-6-2 限位+Index 回原点示意图（当前位置在限位上）

4、 限位+Home+Index 回原点

（1） **电机位置在限位信号外：**调用回原点指令，电机位置在限位信号外，电机从所在位置以较高的速度往限位方向运动，如果碰到限位，则反方向运动并启动高速硬件捕获，在设定是范围内搜索 Home，当触发 Home 开关后，如果设定的搜索 Index 方向与搜索 Home 方向相同，则电机会直接以较低的速度在设定的搜索范围内寻找 Index；反之，电机会以较低的速度运动到捕获的 Home 位置处，之后再次启动运动，在设定的搜索范围内寻找 Index。捕获到编码器的 Index 信号后，电机运动到 Index 处的位置停止。

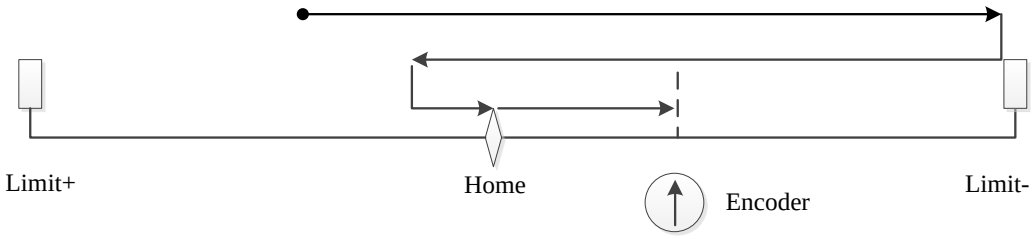


图 9-7-1 限位+Home+Index 回原点示意图（当前位置不在限位上）

（2） **电机位置在限位信号上：**调用回原点指令，电机位置在限位信号上，电机先进行脱离限位的运动，电机从所在位置以较高的速度，目标位置为用户设置的脱离步长，moveDir 的反方向运动，如果运动结束，限位信号没有置起来，则再次从所在位置以较高的速度，目标位置为用户设置的脱离步长，moveDir 的反方向运动，直至脱开限位，否则一直运动；

如果运动结束，已经脱开限位，则以较高的速度往 moveDir 方向运动，触发限位后停止运动，再反方向运动并启动高速硬件捕获，在设定是范围内搜索 Home，当触发 Home 开关后，如果设定的搜索 Index 方向与搜索 Home 方向相同，则电机会直接以较低的速度在设定的搜索范围内寻找 Index；反之，电机会以较低的速度运动到捕获的 Home 位置处，之后再次启动运动，在设定的搜索范围内寻找 Index。捕获到编码器的 Index 信号后，电机运动到 Index 处的位置停止。

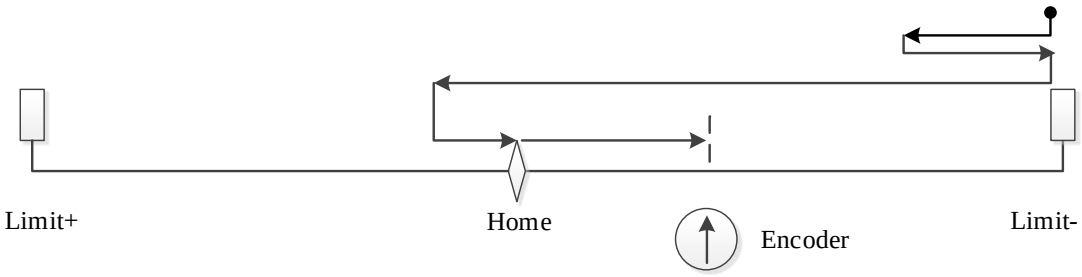


图 9-8-2 限位+Home+Index 回原点示意图（当前位置在限位上）

5、 Home 回原点

（1） 电机位置在 Home 信号外：调用回原点指令，电机位置在 Home 信号外，电机先进行脱离 Home 的运动，电机从所在位置以较高的速度运动并启动高速硬件捕获，在设定的搜索范围内寻找 Home，当触发 Home 开关后，电机会以较低的速度运动到捕获的位置处。

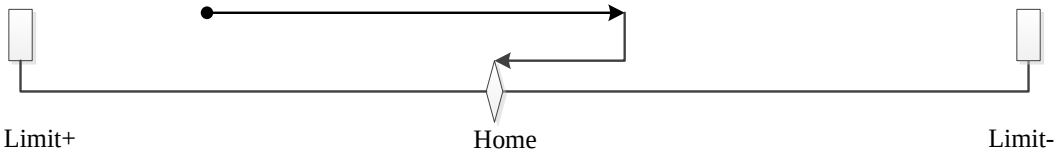


图 9-9-1 Home 回原点示意图（当前位置不在 Home 上）

（2） 电机位置在 Home 信号上：调用回原点指令，电机位置在 Home 信号上，电机从所在位置以较高的速度，目标位置为用户设置的脱离步长，moveDir 的反方向运动，如果运动结束，home 信号没有置起来，则再次从所在位置以较高的速度，目标位置为用户设置的脱离步长，moveDir 的反方向运动，直至脱开 Home，否则一直运动；如果运动过程结束，已经脱开 Home，电机从所在位置以较高的速度运动并启动高速硬件捕获，在设定的搜索范围内寻找 Home，当触发 Home 开关后，电机会以较低的速度运动到捕获的位置处。

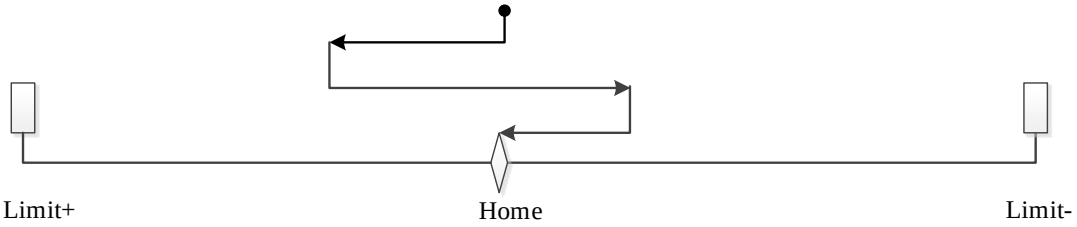


图 9-10-2 Home 回原点示意图（当前位置在 Home 上）

6、 Home+Index 回原点

（1） 电机位置在 Home 信号外，且 Home 信号设置的捕获方向和 Index 设置的捕获方向不一致：当 Home 信号设置的捕获方向和 Index 设置的捕获方向不一致时，调用回原点指令，电机从所在位置以较高的速度运动并启动高速硬件捕获，当触发 Home 开关后，电机会以较低的速度运动到捕获的位置处，之后再次启动运动，在设定的搜索范围内寻找 Index，捕获到编码器的 Index 信号后，电机运动到 Index 处的位置停止。

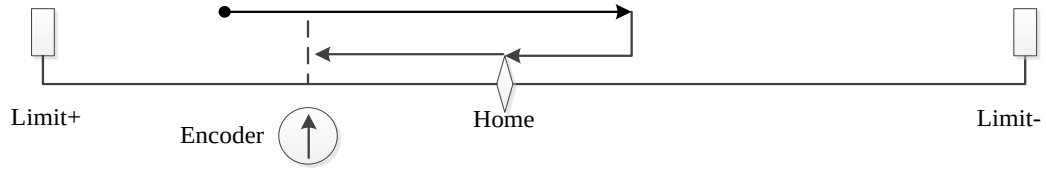


图 9-11-1 Home+Index 回原点示意图（情况 1）

（2） **电机位置在 Home 信号外，且 Home 信号设置的捕获方向和 Index 设置的捕获方向一致：**当 Home 信号设置的捕获方向和 Index 设置的捕获方向一致时，调用回原点指令，电机从所在位置以较高的速度运动并启动高速硬件捕获，当触发 Home 开关后，启动高速硬件捕获 Index，在设定的搜索范围内以较低的速度寻找 Index，捕获到编码器的 Index 信号后，停止到用户设定的位置。

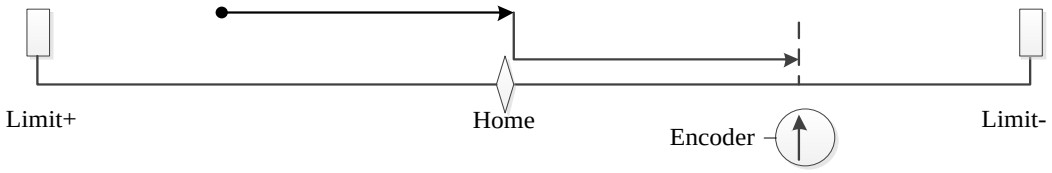


图 9-12-2 Home+Index 回原点示意图（情况 2）

（3） **电机位置在 Home 信号上，且 Home 信号设置的捕获方向和 Index 设置的捕获方向不一致：**调用回原点指令，电机位置在 Home 信号上，电机先进行脱离 Home 的运动，电机从所在位置以较高的速度，目标位置为用户设置的脱离步长，moveDir 的反方向运动，如果运动结束，home 信号没有置起来，则再次从所在位置以较高的速度，目标位置为用户设置的脱离步长，moveDir 的反方向运动，直至脱开 Home，否则一直运动。

如果运动过程结束，已经脱开 Home，电机从所在位置以较高的速度运动并启动高速硬件捕获，在设定的范围内搜索 Home，当触发 Home 开关后，往设定的搜索 Index 的方向搜索 Index 信号。在设定的搜索范围内寻找 Index。捕获到编码器的 Index 信号后，电机运动到 Index 处的位置停止。

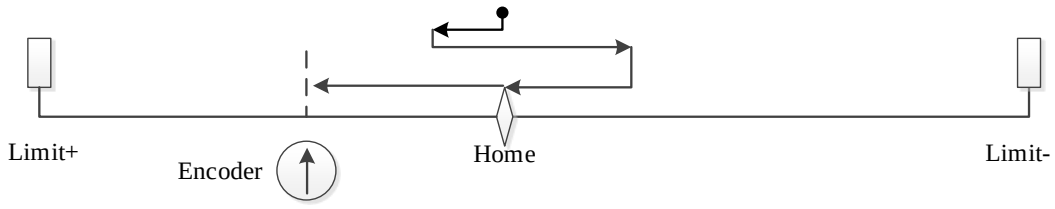


图 9-13-3Home+Index 回原点示意图（情况 3）

（4） **电机位置在 Home 信号上，且 Home 信号设置的捕获方向和 Index 设置的捕获方向一致：**调用回原点指令，电机位置在 Home 信号上，电机先进行脱离 Home 的运动，电机从所在位置以较高的速度，目标位置为用户设置的脱离步长，moveDir 的反方向运动，如果运动结束，home 信号没有置起来，则再次从所在位置以较高的速度，目标位置为用户设置的脱离步长，moveDir 的反方向运动，直至脱开 Home，否则一直运动。

如果运动过程结束，已经脱开 Home，电机从所在位置以较高的速度运动并启动高速硬件捕获，在设定的范围内搜索 Home，当触发 Home 开关后，往设定的搜索 Index 的方向搜索 Index

信号。在设定的搜索范围内寻找 Index。捕获到编码器的 Index 信号后，电机运动到 Index 处的位置停止。

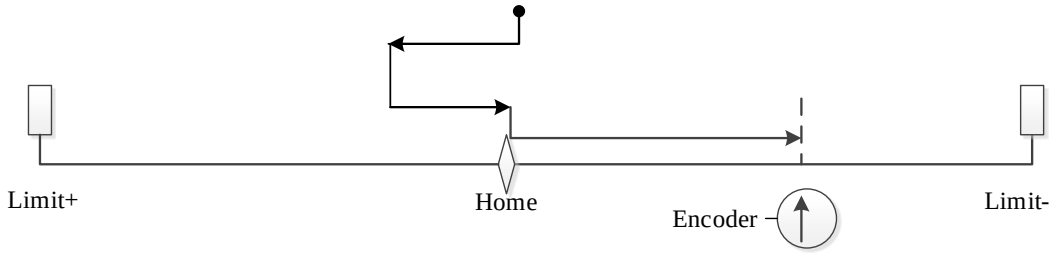


图 9-14-4Home+Index 回原点示意图（情况 4）

7、 Index 回原点

调用回原点指令，电机从所在位置以低速运动并启动高速硬件捕获，捕获到编码器的 Index 信号后，电机运动到 Index 处的位置停止。

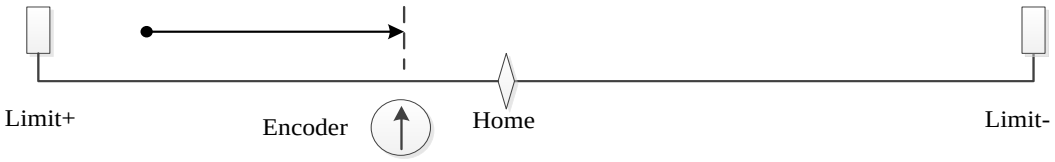


图 9-15 Index 回原点示意图

上面只是简单介绍了几种回原点方式的基本概况，每一种方式都包含正向和负向两种方法，正向指规划位置为正数的方向，负方向指规划位置为负数的方向，例如“限位+Home 回原点”即包括了“正限位+Home 回原点”和“负限位+Home 回原点”；每一种回原点方式，都可以通过设置偏移量使得最终电机停止的位置离原点位置有一个偏移量；每种回原点方式可能由于设定的搜索距离、电机意外停止等因素而找不到原点，大部分异常情况都可以通过查看回原点状态来进行辨识。此外，Smart Home 功能支持各轴单独回原点，互不影响，即可以实现多轴同时回原点。

9.4.3 例程

例程 9-3 回零功能

```
int main(int argc, char* argv[])
{
    short sRtn;
    sRtn= GTN_Open();
    sRtn= GTN_AlarmOff(1,gAxis);
    sRtn=GTN_LmtsOnEx(1,gAxis);
    sRtn= GTN_SetSense (1, MC_LIMIT_POSITIVE, 1, 0);   //限位为低电平触发
    sRtn= GTN_SetSense (1, MC_LIMIT_NEGATIVE, 1, 0);  //限位为低电平触发
    sRtn= GTN_SetSense(1, ENCODE,1,0);      //编码器不取反
    sRtn= GTN_ClrSts(1,gAxis);
    sRtn= GTN_ZeroPos(1,gAxis);
    //设置Smart Home回原点参数
    THomePrmt HomePrm;
```

```
        sRtn= GTN_GetHomePrm (1,1,&tHomePrm);
        tHomePrm.mode=20;
        tHomePrm.moveDir=1;
        tHomePrm.indexDir=1;
        tHomePrm.edge=0;
        tHomePrm.velHigh=5;
        tHomePrm.velLow=1;
        tHomePrm.acc=0.1;
        tHomePrm.dec=0.1;
        tHomePrm.searchHomeDistance=200000;
        tHomePrm.searchIndexDistance=30000;
        tHomePrm.escapeStep=1000;
        sRtn= GTN_GoHome(1,1,&tHomePrm);//启动Smart Home回原点
        do
        {
            THomeStatust HomeSts;
            sRtn= GTN_GetHomeStatus(1,1,&tHomeSts);//获取回原点状态
        } while(tHomeSts.run); //  等待搜索原点停止
        return 0;
    }
```

9.5 位置比较输出

9.5.1 指令列表

表 9-5　位置比较输出指令汇总列表

指令	说明	页码
GTN_SetPosCompareMode	设置位置比较输出模式	247
GTN_GetPosCompareMode	读取位置比较输出模式	204
GTN_PosCompareStatus	读取位置比较输出对应的状态	229
GTN_PosCompareClear	清除位置比较输出缓存区数据	225
GTN_PosCompareSpace	读取位置比较剩余空间指令	228
GTN_PosCompareStop	停止位置比较输出功能	229
GTN_PosCompareStart	开始位置比较输出	228
GTN_PosCompareInfo	读取位置比较输出相关信息	227
GTN_PosCompareData	设置一维位置比较输出数据（FIFO 模式）	226
GTN_SetPosCompareLinear	设置一维线性比较输出参数（线性模式）	246
GTN_GetPosCompareLinear	读取一维线性比较输出参数（线性模式）	203
GTN_PosCompareData2D	设置二维比较输出数据（FIFO 模式）	226
GTN_SetPosComparePsoPrm	设置位置同步比较输出	248
GTN_GetPosComparePsoPrm	读取位置同步比较输出	204
GTN_PosCompareHsOn	打开位置比较输出功能 DMA 通道	227
GTN_PosCompareHsOff	关闭位置比较输出功能 DMA 通道	227
GTN_SetPosCompareFifoMode	设置保存实际比较位置点的缓冲区模式	246

9.5.2 重点说明

1. 硬件输出通道

网络端子板模块包括 2 种不同输出电压的硬件输出通道，即 5V 输出（HSOXX±，例如：HSO00±）和 24V 输出（GPOXX，例如：GPO00）。HSOXX 和 GPOXX 硬件通道可以根据不同需求和不同的软件功能模块绑定，具体的软件功能可参考指令 GTN_SetTerminalPermitEx 中的参数 permit 解释说明。

默认状态下，GNM-402-00 和 GNM-403-00 等模块的 HSO00±、HSO01±分别配置为激光功能的开关信号（Laser+和 Laser-）和激光能量通道（即使能 PWM 信号输出（PWM+和 PWM-）），具体可参考《GNM 系列端子板模块用户手册》。此时，只能通过激光相关功能指令控制该硬件通道的输出，调用其他软件功能模块的指令对此硬件通道无效，并且这些指令会因为执行条件不满足而执行失败，返回值为 1。默认状态下，GNM-402-00 模块的 HSO02±、HSO03±分别设置为第一路和第二路位置比较输出软件功能，可以通过位置比较输出功能控制该硬件通道的输出。获取所有 HSOXX±硬件通道的控制权可以调用指令 GTN_GetTerminalPermitEx。

默认状态下，所有模块的 GPOXX 作为通用 IO 指令输出，可以通过表 8-2 访问。获取所有 GPOXX± 硬件通道的控制权可以调用指令 GTN_GetTerminalPermitEx。

如果用户需要通过非默认功能模块控制 HSOXX±或者 GPOXX 通道的输出，则可以调用指令 GTN_SetTerminalPermitEx 设置控制权。在该指令中，可以通过参数 dataType 设置控制权的硬件资源类型，包括 MC_GPO（通用数字量输出，对应硬件的 DO）和 MC_HSO（高速 IO 输出，对应硬件的 HSO）。

2. 资源映射

不同类型的主卡位置比较输出软件资源个数不一样，详细请参考表 4-2 中的”位置比较输出”对应的资源。各种端子板模块的位置比较输出资源和硬件输出类型详见表 9-6。

表 9-6 端子板那模块位置比较输出资源

模块类型	位置比较输出资源（个）	5V 位置比较输出（路）	24V 位置比较输出（路）
601/602/401	2	无	10
402	2	4	10
403	2	2	10

位置比较输出的软件资源和端子板模块的硬件资源映射关系和表 4-4~

表 4-6 其他资源映射关系一样。以默认优先按照 core1 中的软件资源分配，然后再按照 core2 的软件资源分配（注意：同一个轴模块硬件中的物理资源不能同时分配至 core1 和 core2。对于这种情况，如果 core1 的逻辑资源不足，将模块分配在 core1 会导致资源的浪费，则直接将该模块分配至 core2）。

最终资源分配结果是：前 2 个端子板模块分配为核 1，后两个分配为核 2。如图 4-3 所示。资源映射详见表 4-5 和

表 4-6。

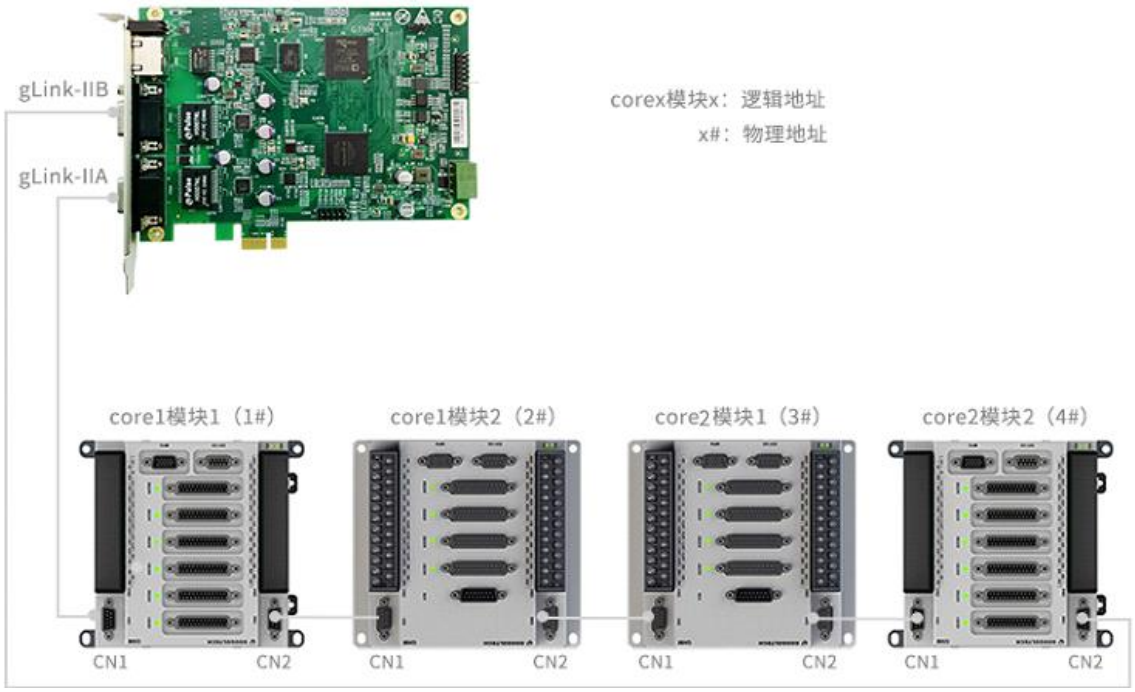


图 4-3 连接方式为例，双核关于位置比较输出软件资源和硬件资源的映射关系见

表 9-7。

表 9-7　位置比较输出资源映射

core1 软件和硬件资源映射			
资源名称	软件资源序号 Index	硬件资源	
		端子板模块序号	端子板模块通道(FIFO)
位置比较输出	1~2	core1 模块 1(1#)	1~2
	3~4	core1 模块 2(2#)	1~2
core2 软件和硬件资源映射 **(core2 目前不支持位置比较功能)**			
位置比较输出			

9.5.3 位置比较输出模式设置

1. 指令列表

表 9-8　位置比较输出模式设置指令列表

指令	说明	页码
GTN_SetPosCompareMode	设置位置比较输出模式	247
GTN_GetPosCompareMode	读取位置比较输出模式	204
GTN_PosCompareStatus	读取位置比较输出对应的状态	229
GTN_PosCompareSpace	读取位置比较剩余空间指令	228
GTN_PosCompareClear	清除位置比较输出缓存区数据	225
GTN_PosCompareStop	停止位置比较输出功能	229
GTN_PosCompareStart	开始位置比较输出	228

GTN_PosCompareInfo	读取位置比较输出相关信息	227

2. 例程

例程 9-4 设置位置比较输出通道 1 为一维 FIFO 模式

```
//读取位置比较输出模式
TPosCompareMode mode;
rtn= GTN_GetPosCompareMode(1,1,&mode);
mode.dimension= 1;        //一维
mode.mode=0;              //FIFO模式
mode.outputMode=0;        // 0-脉冲，1-电平，2-电平自动翻转
mode.outputPulseWidth=10;     //设置脉冲宽度
mode.sourceMode=0;        //0-编码器，1-冲计数器
mode.sourceX= 1;          //x轴对应的实际轴为轴1
//设置位置比较输出模式
rtn= GTN_SetPosCompareMode(1,1,&mode);
…… …… ……
```

例程 9-5 设置位置比较输出通道 1 为一维线性模式

```
TPosCompareMode mode;
rtn= GTN_GetPosCompareMode(1,1,&mode);   //读取位置比较输出模式
mode.dimension= 1;                       //一维
mode.mode=1;                             //线性模式
mode.outputMode=2;                       // 0-脉冲，1-电平，2-电平自动翻转
mode.outputPulseWidth=10;                //设置脉冲宽度
mode.sourceMode=0;                       //0-编码器，1-脉冲计数器
mode.sourceX= 1;                         //x轴对应的实际轴为轴1
rtn= GTN_SetPosCompareMode (1,1,&mode);  //设置位置比较输出模式
…… …… ……
```

例程 9-6 设置位置比较输出为二维 FIFO 模式

```
TPosCompareMode mode;
rtn= GTN_GetPosCompareMode (1,1,&mode);  //读取位置比较输出模式
mode.dimension=2;                        //二维
mode.errorBand=10;                       //误差限位10Pulse
mode.mode=0;                             //FIFO模式
mode.outputMode=0;                       // 0-脉冲，1-电平，2-电平自动翻转
mode.outputPulseWidth=10;                //设置脉冲宽度
mode.sourceMode=0;                       //0-编码器，1-冲计数器
mode.sourceX= 1;                         //x轴对应的实际轴为轴1
mode.sourceY= 2;                         //y轴对应的实际轴为轴2
rtn= GTN_SetPosCompareMode (1,1,&mode);  //设置位置比较输出模式
…… …… ……
```

9.5.4 一维位置比较输出功能

1. 指令列表

表 9-9 一维位置比较输出功能指令列表

指令	说明	页码
GTN_PosCompareData	设置一维位置比较输出数据（FIFO 模式）	226
GTN_SetPosCompareLinear	设置一维线性比较输出参数（线性模式）	246
GTN_GetPosCompareLinear	读取一维线性比较输出参数（线性模式）	203

2. 重点说明

通过指令 GTN_SetPosCompareMode 设置位置比较输出模式， GTN_PosCompareData 增加位置比较输出数据。对应的轴准确运动到设置位置时，对应的位置比较输出通道会输出电平或脉冲。

3. 例程

例程 9-7 通用一维位置比较输出（FIFO 模式）

```
/*
功能描述：第一通道位置比较输出，压入 10 组位置比较数据，位置点为[2,1002，2002…….,8002，9002],
到达每个位置点后第 1 路通用输出 DO00 和第 1 路 HSO±同时输出 1 个脉冲。
*/
//设置位置比较输出为一维 FIFO 模式,见例程 9-4
…… …… ……
//
int m_count=10;
int i=0;
short rtn;
short permit;
TPosCompareData comparedata;
rtn = GTN_PosCompareClear(1,1) ;
//设置第 1 路 GPO 和 HSO 为第 1 路位置比较输出模式
permit = 0x2;
rtn = GTN_SetTerminalPermitEx(1,1,MC_GPO,&permit,1,1);
rtn =   GTN_SetTerminalPermitEx(1,1,MC_HSO,&permit,1,1);
//读取位置比较输出模式
TPosCompareMode mode;
rtn = GTN_GetPosCompareMode(1,1,&mode);
mode.dimension= 1;        //一维
mode.mode=0;              //FIFO模式
mode.outputMode=0;        // 0-脉冲，1-电平，2-电平自动翻转
mode.outputPulseWidth=10;     //设置脉冲宽度
mode.sourceMode=0;        //0-编码器，1-冲计数器
mode.sourceX= 1;          //x轴对应的实际轴为轴1
//设置位置比较输出模式
rtn= GTN_SetPosCompareMode(1,1,&mode);
//设置位置比较输出数据
```

```
for ( i=1;i<=m_count;i++)
{
    comparedata.hso=0x1;
    comparedata.gpo=0x1;
    comparedata.pos=2+(i-1)*1000;
    comparedata.segmentNumber=i;
    rtn= GTN_PosCompareData (1,1,&comparedata);      //增加10组数据
}
//启动位置比较输出
rtn = GTN_PosCompareStart(1,1);
…… …… ……
```

例程 9-8　通用一维位置比较输出（Linear 模式）

```
/*
功能描述：第二通道位置比较输出，压入 10 组位置比较数据，位置点为[2,1002，2002…….,8002，9002],
到达每个位置点后第 2 路通用输出 DO00 和第 2 路 HSO±同时输出 1 个脉冲。
*/
//设置位置比较输出为一维线性模式,见例程 9-8
…… …… ……
int m_count=10;
int i=0;
short permit;
TPosCompareData comparedata;
rtn= GTN_PosCompareClear(1,2);
//设置第 2 路 GPO 和 HSO 为第 2 路位置比较输出模式
permit = 0x4;
rtn = GTN_SetTerminalPermitEx(1,1,MC_GPO,&permit,2,1);
rtn = GTN_SetTerminalPermitEx (1,1,MC_HSO, &permit,2,1);
//读取位置比较输出模式
TPosCompareMode mode;
rtn= GTN_GetPosCompareMode(1,1,&mode);
mode.dimension= 1;                    //一维
mode.mode=1;                          //线性模式
mode.outputMode=2;                    // 0-脉冲，1-电平，2-电平自动翻转
mode.outputPulseWidth=10;             //设置脉冲宽度
mode.sourceMode=0;                    //0-编码器，1-脉冲计数器
mode.sourceX= 1;                      //x轴对应的实际轴为轴1
//设置位置比较输出模式
rtn= GTN_SetPosCompareMode (1,1,&mode);
//设置一维线性比较输出参数
TPosCompareLinear pLinear;
rtn = GTN_GetPosCompareLinear(1,1,&pLinear);
pLinear.count = 100;
pLinear.hso = 0x2;
pLinear.gpo = 0x2;
pLinear.startPos = 0;
```

```
pLinear.interval = 1000;
rtn = GTN_SetPosCompareLinear(1,1,& pLinear);
//启动位置比较输出
rtn = GTN_PosCompareStart(1,1);
…… …… ……
```

9.5.5 二维位置比较输出功能

1. 指令列表

表 9-10 二维位置比较输出指令列表

指令	说明	页码
GTN_PosCompareData2D	设置二维比较输出数据（FIFO 模式）	226

2. 重点说明

通过指令 GTN_SetPosCompareMode 设置位置比较输出模式， GTN_PosCompareData 增加位置比较输出数据（如图 1 中圆点 T[Tx,Ty]为其中一个数据点）。对于二维位置点 T[Tx,Ty]，运动位置到达图 1 的蓝色矩形内，对应的位置比较输出通道会输出电平或脉冲，其中 Mx=My= errorBand。

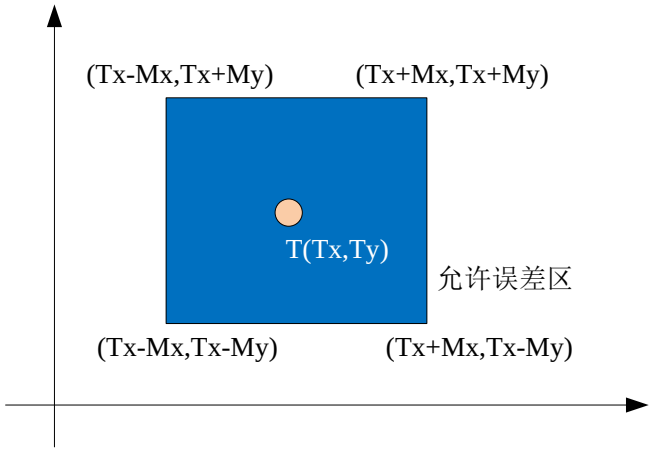


图 9-16 二维位置比较输出范围（errorband>0）

3. 例程

例程 9-9 二维位置比较输出（FIFO 模式）

```
/*
功能描述：第一通道位置比较输出，压入 10 组位置比较数据，位置点为[2,1002，2002…….,8002，9002],
到达每个位置点后第 1 路通用输出 DO00 和第 1 路 HSO±同时输出 1 个脉冲。
*/
//设置位置比较输出为二维 FIFO 模式,请参考例程 9-6
…… …… ……
int m_count=10;
int i=0;
short permit;
TPosCompareData2D comparedata2d;
rtn= GTN_PosCompareClear(1,1);
```

```
//设置第 1 路 GPO 和 HSO 为第 1 路位置比较输出模式
permit = 0x2;
rtn = GTN_SetTerminalPermitEx(1,1,MC_GPO,&permit,1,1);
rtn =  GTN_SetTerminalPermitEx(1,1,MC_HSO,&permit,1,1);
//读取位置比较输出模式
rtn= GTN_GetPosCompareMode (1,1,&mode);
mode.dimension=2;                    //二维
mode.errorBand=10;                   //误差限位10Pulse
mode.mode=0;                         //FIFO模式
mode.outputMode=0;                   // 0-脉冲，1-电平，2-电平自动翻转
mode.outputPulseWidth=10;            //设置脉冲宽度
mode.sourceMode=0;                   //0-编码器，1-冲计数器
mode.sourceX= 1;                     //x轴对应的实际轴为轴1
mode.sourceY= 2;                     //y轴对应的实际轴为轴2
//设置位置比较输出模式
rtn= GTN_SetPosCompareMode (1,1,&mode);
//设置位置比较输出数据
for ( i=1;i<=m_count;i++)
{
    comparedata2d.hso=0x1;
    comparedata2d.gpo=0x1;
    comparedata2d.posX=2+(i-1)*1000;
    comparedata2d.posY=2+(i-1)*1000;
    comparedata2d.segmentNumber=i;
    rtn=  GTN_PosCompareData2D(1,1,&comparedata);//增加10组数据
}
//启动位置比较输出
rtn = GTN_PosCompareStart(1,1);
…… …… ……
```

例程 9-10　获取实际的位置比较点

在二维位置比较中，两个轴由于跟随特性差异，无法同时达到理论点进行位置比较，而是在设定的误差带范围内就触发比较，此时如果需要知道实际的比较点位置，那么需要通过获取实际位置比较点功能来实现。

以下例程设置了二维位置比较FIFO模式，压入1000个点，启动插补运动，并获取实际比较点。

```
short rtn;
short core = 1;
//设置第一路位置比较
short posCompareIndex = 1;
short permit;
rtn = GTN_GetTerminalPermitEx(core,1,MC_HSO,&permit, posCompareIndex,1);
permit = 0x2;
rtn = GTN_SetTerminalPermitEx(core,1,MC_HSO,&permit, posCompareIndex,1);
```

```
//设置存储实际位置比较点缓冲区模式（此处静态模式）
rtn = GTN_SetPosCompareFifoMode(core, posCompareIndex,1);

//二维位置比较 FIFO 模式
TPosCompareMode mode;
rtn = GTN_GetPosCompareMode(core, posCompareIndex,&mode);
mode.dimension = 2;
mode.mode = 0;                  //FIFO 模式
mode.outputMode = 0; //0 脉冲，电平
mode.outputPulseWidth = 10;
mode.sourceMode = 1; //0 编码器，脉冲计数器
mode.sourceX = 1;
mode.sourceY = 2;
rtn = GTN_SetPosCompareMode(core, posCompareIndex,&mode);
if (rtn)
{
    printf("Error GTN_SetPosCompareMode=%d\n",rtn);
}

rtn = GTN_PosCompareClear(core, posCompareIndex);
if (rtn)
{
    printf("Error GTN_PosCompareClear=%d\n",rtn);
}
//压入二维位置比较点
unsigned short space;
TPosCompareData2D compareData;
for (int i=0;i<1000;i++)
{
    rtn = GTN_PosCompareSpace(core, posCompareIndex,&space);
    if (rtn)
    {
        printf("Error GTN_PosCompareSpace=%d\n",rtn);
    }

    compareData.gpo = 0xffff;
    compareData.hso = 0xffff;
    compareData.posX = (i+1) * 10;
    compareData.posY = (i+1) * 0;
    compareData.segmentNumber = i;
    rtn = GTN_PosCompareData2D(core, posCompareIndex,&compareData);
    if (rtn)
    {
        printf("Error GTN_PosCompareData2D=%d\n",rtn);
    }
}
```

```
//启动二维位置比较运动
rtn = GTN_PosCompareStart(core, posCompareIndex);
if (rtn)
{
    printf("Error GTN_PosCompareStart=%d\n",rtn);
}
//启动插补运动
short fifo = 0;
TCrdPrm crdPrm;
rtn = GTN_GetCrdPrm(core,crd,&crdPrm);
crdPrm.dimension = 2;
crdPrm.synVelMax = 500;
crdPrm.synAccMax = 1;
crdPrm.evenTime = 50;
crdPrm.profile[0] = 1;
crdPrm.profile[1] = 2;
rtn = GTN_SetCrdPrm(core,crd,&crdPrm);

rtn =GTN_CrdClear(core,crd,fifo);

rtn = GTN_LnXY(core,crd,20000,0,5,1,0,fifo);
rtn = GTN_CrdStart(core,1<<(crd-1),0);

//等待运动结束
short crdRun;
long crdSegment;
double prfPos[4];
double encPos[4];
TPosCompareInfo posCompareInfo;
TPosCompareStatus posCompareStatus;
do
{
    rtn = GTN_GetPrfPos(core,1,prfPos,4);
    rtn = GTN_PosCompareStatus(core, posCompareIndex,&posCompareStatus);
    rtn = GTN_PosCompareInfo(core, posCompareIndex,&posCompareInfo);
    printf("prfPos1=%0.1lf,space=%d,pulse=%ld,seg=%ld,hso=%d,gpo=%d\n",
        prfPos[0],posCompareStatus.space,posCompareStatus.pulseCount,posCompareStatus.
        segmentNumber,posCompareStatus.hso,posCompareStatus.gpo);

    //运动停止则关闭位置比较
    rtn = GTN_CrdStatus(core,1,&crdRun,&crdSegment,0);
    if (0 == crdRun)
    {
        rtn = GTN_PosCompareStop(core, posCompareIndex);
        if (rtn)
        {
```

```
                printf("Error GTN_PosCompareStop=%d\n",rtn);
            }
        }

    } while (1 == posCompareStatus.run);

    //获取实际位置比较点
    long posDataX[2000];
    long posDataY[2000];
    long actCount;
    TLatchValueInfo latchInfo;
    rtn =GTN_GetPosCompareLatchValue(core, posCompareIndex,100,&posDataX[0],&posDataY[0],
        &actCount,&latchInfo);
```

9.5.6 位置比较指令快速传输功能

1. 指令说明

表 9-11 位置比较指令快速传输功能指令列表

指令	说明	页码
GTN_PosCompareHsOn	打开位置比较输出功能 DMA 通道	227
GTN_PosCompareHsOff	关闭位置比较输出功能 DMA 通道	227

2. 重点说明

为了提高位置比较输出指令的传输速率，增加 DMA 通道。

a) DMA 功能为数据段批次压入运动控制器的功能，可以提高指令的传输速率。每次压入运动控制器的数据段数量由用户设定，默认为 200 段。开启 DMA 后，数据段会先被压入 DMA 的缓存区中，当数据段数到达阈值，如 200 段后，系统会自动将缓存区中的指令下发到运动控制器。压完所有的数据段之后，再循环调用 GTN_PosCompareData 或 GTN_PosCompareData2D 指令将 DMA 缓存区中的剩余数据段（剩余的段数未达到阈值）压入运动控制器，直到指令返回值为 0。

b) 开启 DMA 功能后， GTN_PosCompareStatus 将查询的为 PC 端的剩余数据段空间（共有 1000 段空间）。

c) 插补、振镜和位置比较输出等功能如果需要同时开启 DMA 功能时，需要选择不同的 DMA 缓存区序号。

3. 例程

例程 9-11 位置比较输出 DMA 通道

```
short i;
//设置第一路位置比较
short posCompareIndex = 1;

//开启位置比较输出 DMA 功能，设置 DMA 缓存区阈值为 200
rtn = GTN_PosCompareHsOn(1,1,1,200);
//设置第 1 路 GPO 和 HSO 为第 1 路位置比较输出模式
```

```
short permit;
permit = 0x2;
rtn = GTN_SetTerminalPermitEx(1,1,MC_GPO,&permit,1,1);
rtn = GTN_SetTerminalPermitEx(1,1,MC_HSO,&permit,1,1);
TPosCompareMode mode;
rtn= GTN_GetPosCompareMode (1,1,&mode);//读取位置比较输出模式
mode.dimension=1;// 一维
mode.errorBand=0;
mode.mode=0;// FIFO模式
mode.outputMode=0;//0-脉冲，1电平
mode.outputPulseWidth=10;//设置脉冲宽度
mode.sourceMode=0//0编码器，1冲计数器
mode.sourceX= 1;//x轴对应的实际轴为轴1
rtn= GTN_SetPosCompareMode (1,1,&mode);//设置位置比较输出模式
rtn= GTN_PosCompareClear(1,1);
/*
压入 1000 段位置比较输出数据
*/
TPosCompareData posCompareData;
for(i=0;i<1000;++i)
{
    posCompareData.segmentNumber = i+1;
    posCompareData.pos += 100;
    posCompareData.hso = 0;
    posCompareData.gpo = 1;
    rtn = GTN_PosCompareData(port, posCompareIndex,&posCompareData);
    if( CMD_SUCCESS != rtn )
    {
        return;
    }
}
/*将剩余的数据压入DSP缓存区*/
do
{
    rtn = GTN_PosCompareData(port, posCompareIndex,NULL);
} while (1 == rtn);

/*启动位置比较输出 */
rtn = GTN_PosCompareStart(core, posCompareIndex);
```

第10章 安全机制

10.1 本章简介

本章介绍运动控制器提供给用户的所有可用安全机制，包括：限位、报警、平滑停止、紧急停止、跟随误差极限停止和掉电存储功能。

提示	本手册中所有字体为蓝色的指令（如 GTN_PrfTrap）均带有超级链接，点击可跳转至指令说明。

10.2 限位

运动控制器能够通过安装限位开关或者设置软限位来限制各轴的运动范围，如图 10-1 所示。

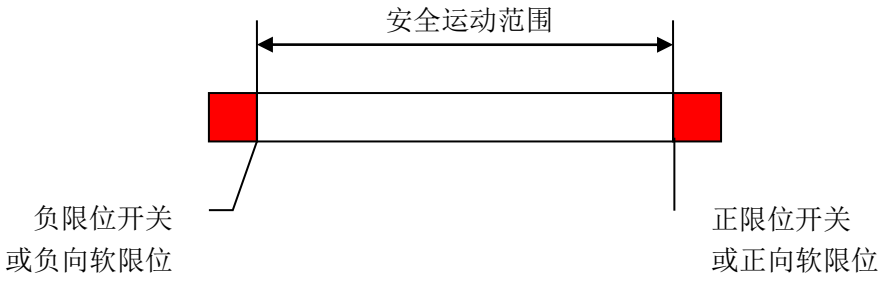


图 10-1 轴运动范围

工作台碰到限位开关或者规划位置超越软限位时，运动控制器紧急停止工作台的运动。限位触发以后，运动控制器禁止触发限位方向上运动，同时该轴的限位触发状态置 1。离开限位回到安全运动范围以后，需要调用指令 GTN_ClrSts 清除限位触发状态，才能使控制轴回到正常运动状态。

控制器复位后，默认软限位是无效的，没有触发的。

注意	限位标志为模态标志，一旦置起，需在离开限位回到安全运动范围以后，调用指令 GTN_ClrSts 清除限位触发状态

10.2.1 指令列表

表 10-1 软限位指令列表

指令	说明	页码
GTN_SetSoftLimitMode	设置软限位模式	252
GTN_GetSoftLimitMode	读取软限位模式	209
GTN_GetLimitStatus	读取限位状态	200
GTN_SetSoftLimitEx	设置轴正向软限位和负向软限位	251
GTN_GetSoftLimitEx	读取轴正向软限位和负向软限位	209

10.2.2 重点说明

应当在回原点以后再设置软限位。正向软限位必须大于负向软限位。软限位和限位开关可以同时使用，当软限位触发时也会置起限位触发标志。

限位触发以后使用急停加速度紧急停止。默认急停加速度为 1000pulse/ms^2。

10.2.3 例程

例程 10-1 软限位使用

该例程设置了第一轴的软限位，正向在 20000 处，负向在-20000 处。启动第一轴的点位运动后，通过下图 10-2 可以看到，当运动超过正向软限位时，限位信号触发，运动停止。

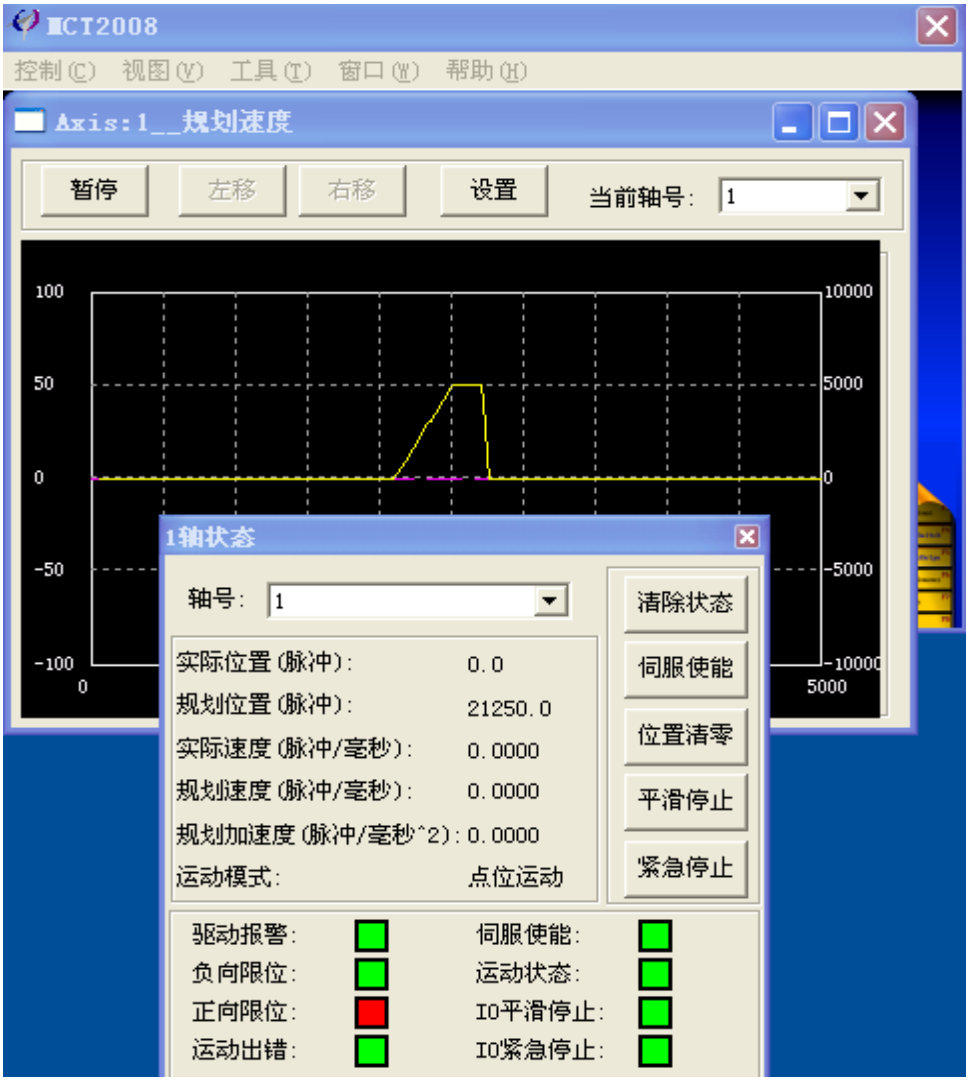


图 10-2 软限位触发

```
#include "stdafx.h"
#include "conio.h"
#include "gts.h"

#define AXIS                    1
int main(int argc, char* argv[])
```

```
{
    // 指令返回值
    short sRtn;
    // 轴状态变量
    long sts;
    // 点位运动结构体变量
    TTrapPrm trap;
    // 规划位置
    double prfPos;

    // 打开运动控制器
    sRtn = GTN_Open();
    // 复位
    sRtn = GTN_Reset(1);
    // 控制器配置
    sRtn = GTN_LoadConfig(1,"test.cfg");
    // 清除轴状态
    sRtn = GTN_ClrSts(1,1, 8);
    sRtn = GTN_LmtsOnEx(1, AXIS);
    // 设置软限位
    sRtn = GTN_SetSoftLimitEx(1,AXIS, 20000, -20000);
    // 将第一轴设置为点位运动模式
    sRtn = GTN_PrfTrap(1,AXIS);
    // 设置点位运动参数
    sRtn = GTN_GetTrapPrm(1,AXIS, &trap);
    trap.acc = 0.125;
    trap.dec = 0.125;
    sRtn = GTN_SetTrapPrm(1,AXIS, &trap);
    // 设置点位运动目标速度
    sRtn = GTN_SetVel(1,AXIS, 50);
    // 设置点位运动目标位置
    sRtn = GTN_SetPos(1,AXIS, 1000000L);
    // 启动点位运动
    sRtn = GTN_Update(1,1<<(AXIS-1));

    while(!kbhit())
    {
        // 读取第一轴轴状态
        sRtn = GTN_GetSts(1,AXIS, &sts);
        // 读取第一轴规划位置
        sRtn = GTN_GetPrfPos(1,AXIS, &prfPos);
        printf("sts=0x%-8lx prfPos=%-10.2lf\r", sts, prfPos);
    }
    return 0;
}
```

10.3 报警

运动控制器提供专用的驱动报警信号输入接口。当检测到驱动器报警信号以后，运动控制器将关闭该轴的伺服使能，急停运动规划，同时该轴报警触发标志置 1。

驱动器报警信号产生以后，应当执行以下操作：

(1) 确定引起驱动器报警的原因，并加以改正；

(2) 复位驱动器；

(3) 调用 GTN_ClrSts清除报警，重新回机床原点。

10.4 平滑停止和急停

运动控制器的每个轴都可以定义平滑停止 IO 和急停 IO。

当平滑停止 IO 输入为触发电平时（触发电平可以设置），运动控制器自动平滑停止所关联的控制轴，并将轴状态字（bit7）置 1。

当急停 IO 输入为触发电平时（触发电平可以设置），运动控制器自动紧急停止所关联的控制轴，并将轴状态字（bit8）置 1。

IO 平滑停止或者 IO 急停完成以后，必须调用 GTN_ClrSts 指令清除停止标志位（bit7 和 bit8），才能继续运动。

10.5 跟随误差极限

对于伺服控制系统而言，在某些异常情况下，电机的实际位置可能与规划位置差距很大。这时通常存在一些危险情况，例如电机故障、编码器 A、B 相信号接反或断线、机械摩擦太大或者机械故障造成电机堵转等。为了及时检测这些情况，增强系统的安全性并延长设备使用寿命，运动控制器提供跟随误差超限自动停止的安全保护机制。

运动控制器在每个控制采样周期内都检查控制轴的实际位置与规划位置的误差是否超越所设定的跟随误差极限。图 5-15 跟随误差解释图为跟随误差示意图，如果位置误差超越所设定的跟随误差极限，运动控制器自动紧急停止该轴的运动，同时该轴跟随误差越限标志置 1。

如何设置跟随误差极限请参考 5.3.5 节。

10.6 掉电存储功能

10.6.1 指令列表

表 10-2 掉电功能指令列表

指令	说明	页码
GTN_SetRetainValue	保存数据到 MRAM 存储芯片	250
GTN_GetRetainValue	读取 MRAM 存储芯片数据	208

10.6.2 重点说明

掉电存储功能可以保存用户的重要数据，防止在非正常情况下（例如非人为的断电）丢失一些重要的数据。目前提供的 MRAM 芯片最大存储资源为 16384 个 words，用户可以根据自己需求随意使用，详细的使用方式详见下文。

10.6.3 例程

例程 10-2 掉电存储操作

```
int main(int argc, char* argv[])
{
    short sRtn;

    sRtn = GTN_Open();

    short pwrite[16], pread[16];
    unsigned long address = 16368;
    for (int I = 0; I < 16 ;i++)
    {
        pdata[i] = i;
    }

    sRtn = GTN_GetRetainValue(1, address, 16, &pread[0]);

    sRtn = GTN_SetRetainValue(1, address, 16, &pwrite [0]);

    sRtn = GTN_GetRetainValue(1, address, 16, &pread[0]);

    return 0;
}
```

第11章　运动程序

11.1 本章简介

本章将介绍下载在控制器中运行的程序——运动程序。用户想要使用运动程序，需要了解相应的语法，以及下载步骤。本章将一一介绍。

提示	本手册中所有字体为蓝色的指令（如 GTN_PrfTrap）均带有超级链接，点击可跳转至指令说明。

11.2 运动程序概述

为了表述方便，直接在PC机上调用动态链接库发送指令访问控制器的程序称为“应用程序”，下载到运动控制器上执行的程序称为“运动程序”。运动程序与应用程序的关系如图11-1所示。

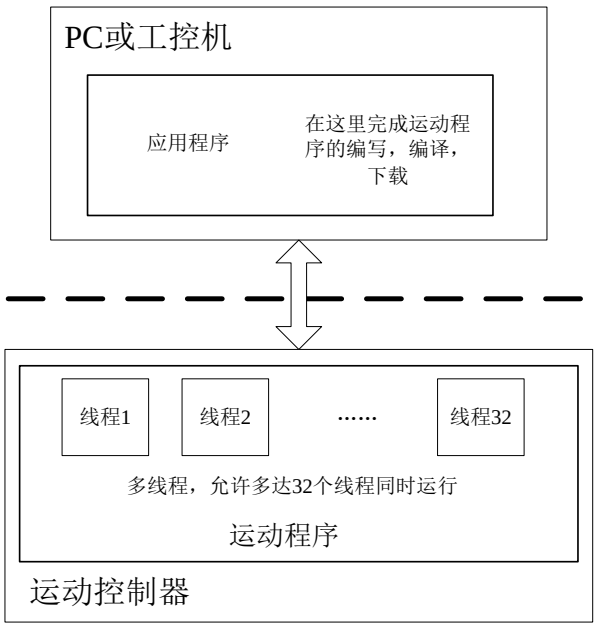


图 11-1　运动程序与应用程序的关系

运动程序的三个特点：独立性，实时性，并行性。

独立性：运动程序能够脱离主机在运动控制器上独立执行，主机能够将CPU资源分配给其它任务，从而将主机从繁琐的运动逻辑管理中解放出来。当然，如果需要，主机仍然可以在任何时候向控制器发送指令，即使运动控制器上的运动程序正在执行。

注意	当主机指令和运动控制器上的运动程序控制相同的轴时，需仔细设计运动逻辑，以免造成混乱。

实时性：运动控制器上执行运动程序由于不需要通过总线和主机进行频繁的数据交换，因此具有更高

的实时性。和在主机上执行的应用程序不同之处在于，运动程序对GT指令调用不必再通过PC总线，因此具有更高的执行效率。平均执行速度约为100指令/毫秒，是PC机执行API指令速度的5倍。

并行性：支持多任务，允许多达32个运动程序在运动控制器上同时执行。

注意	在多线程环境下，一个线程中连续的 2 条指令在执行时有可能被插入其它线程的指令。当启动多个线程并行执行时，应当仔细考虑线程之间是否会相互影响。

11.3 运动程序的使用

11.3.1 编写运动程序

运动程序可以使用C语言编写。但是一些编写规则和C语言略有不同。用户编写运动程序时应遵照“11.4.1 语言元素”、“11.4.2 运算指令”中的说明，否则有可能编译不通过。

运动程序可以和应用程序一样调用GT指令。用户可查阅“11.5 可在运动程序中使用的指令”知道哪些指令可以在运动程序中调用。

注意	运动程序中，调用 GT 指令必须完整描述函数的每一个参数。 例如：short GT_GetClock(unsigned long *pClock, unsigned long *pLoop=NULL)。 在应用程序中调用，可以写成如下形式，使用 VC 可以编译通过： long lClock; GT_GetClock(&lClock); 在运动程序中调用，必须写成如下形式，否则编译不通过： long lClock, lLoop; GT_GetClock (&lClock, &lLoop);

11.3.2 编译

为了让运动控制器能够执行用户用C语言编写的运动程序，必须对运动程序进行编译。使用MCT2008编译运动程序，生成目标程序文件（*.bin）和符号文件（*.ini）。目标文件用来下载到运动控制器。符号文件用来保存运动程序编译信息。应用程序必须使用这2个文件才能正确下载和启动运动程序，访问运动程序的变量。

关于如何使用MCT2008编译运动程序，请参考“MCT2008使用帮助”。具体操作：启动MCT2008，点击主界面菜单“帮助”-->“MCT2008使用帮助”，将会弹出MCT2008使用帮助对话框。在对话框左侧“目录”一页点击“工具”-->“运动程序编译器.mht”，可以查看详细的如何编译运动程序的说明。如图11-2所示。

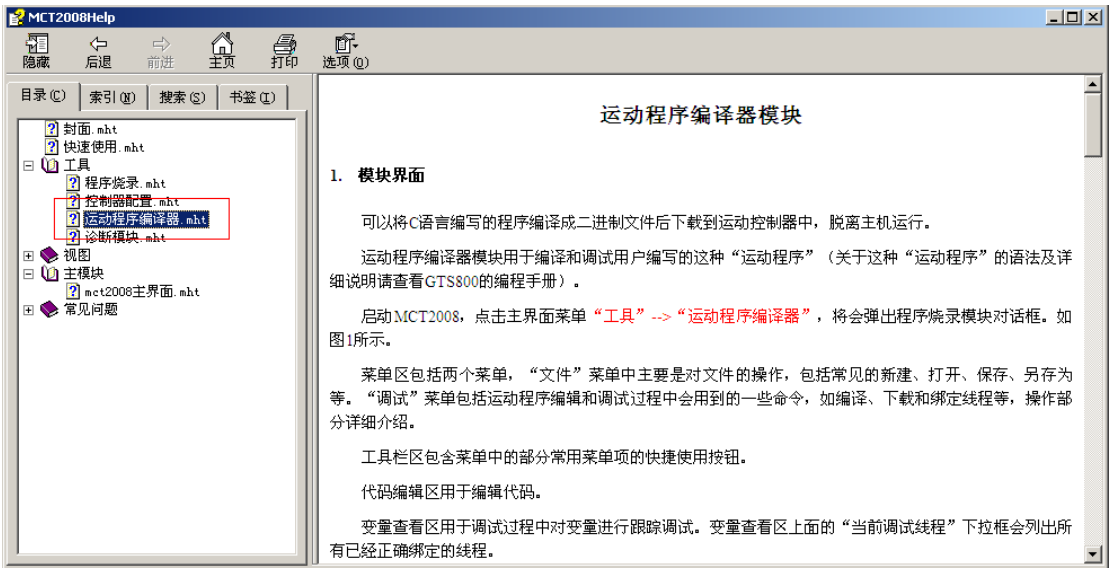


图 11-2　MCT2008 运动程序编译说明界面

编译成功后，目标程序文件（*.bin）和符号文件（*.ini）会出现在运动程序文件（*.c）的同一目录下。

注意	如果编译不成功，则无法进行下一步操作。用户应按照编译器的提示仔细检查错误原因。

11.3.3 指令列表

后面的操作需要用户在应用程序中调用相关的运动程序指令来完成。

表 11-1　运动程序指令列表

指令	说明	页码
GTN_Download	下载运动程序到运动控制器	180
GTN_GetFunId	读取运动程序中函数的标识	196
GTN_GetVarId	读取运动程序中变量的标识	215
GTN_Bind	绑定线程、函数、数据页	169
GTN_RunThread	启动线程	232
GTN_StopThread	停止正在运行的线程	258
GTN_PauseThread	暂停正在运行的线程	225
GTN_GetThreadSts	读取线程的状态	212
GTN_SetVarValue	设置运动程序中变量的值	255
GTN_GetVarValue	读取运动程序中变量的值	215

11.3.4 下载

编译成功后，用户需要在应用程序中调用 GTN_Download 指令将目标文件（*.bin）下载到运动控制器的 SDRAM 中。用户应保证目标文件和符号文件与应用程序在同一目录下。

当下载新的运动程序时会覆盖原有的运动程序。运动控制器每次上电以后需要重新下载运动程序。

11.3.5 绑定线程、函数和数据页

运动程序下载到运动控制器后，还不能立即执行。运动控制器会自动为运动程序中的每个函数，全局变量和局部变量定义好 ID，调用 GTN_GetFunId 读取函数 ID，调用 GTN_GetVarId 读取变量 ID。

为了执行运动程序，必须调用 GTN_Bind 指令绑定线程、函数和数据页。

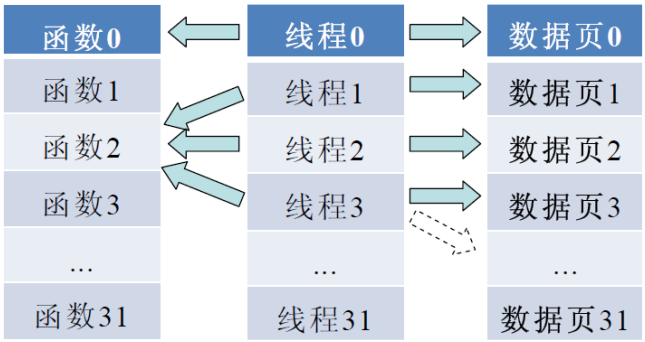


图 11-3　线程、函数和数据页的关系

运动控制器支持 32 个线程同时运行，一个线程只能分配一个函数，但是一个函数可以分配给多个线程同时执行，例如多轴回零时，可以让多个线程绑定同一个回零函数，然后同时启动这些线程就可以实现多轴同时回零。在线程执行过程中不允许绑定新的函数，除非线程执行完毕。

各函数的局部变量放在相互独立的数据页中。运动控制器提供 32 个数据页。在绑定线程和函数时，必须指明所使用的数据页。一个数据页只能分配给一个线程，但是一个线程可以使用多个数据页。线程在执行过程中可以切换数据页。其关系示意如图 11-3 所示。

11.3.6 启动，停止，暂停线程

将线程、函数和数据页绑定好后，调用 GTN_RunThread 指令就可以启动某一个线程。调用 GTN_StopThread 停止某个正在运行的线程，调用 GTN_PauseThread 暂停线程。

11.3.7 查询线程状态

应用程序可以随时调用 GTN_GetThreadSts 指令查询线程的执行状态。包括该线程是否正在运行，线程绑定的函数的返回值，当前正在执行的指令所在行数，当前正在执行的指令的返回值。

应用程序可以随时调用 GTN_SetVarValue 指令更新运动程序中所有变量的值。

应用程序可以随时调用 GTN_GetVarValue 指令查询运动程序中所有变量的值。

11.3.8 例程

1. 单线程累加求和

例程 11-1　运动程序单线程累加求和

运动程序完成累加求和任务。定义了全局变量 sum 用于保存累加和，局部变量 begin 用于保存累加起点，局部变量 end 用于保存累加终点。累加完成以后程序结束。

```
// ---------------------------------------------
```

```
    // 累加求和
    // begin  累加起点
    // end    累加终点
    // ---------------------------------------------
int sum;
int add(int begin, int end)
{
    int i;
    int cc;
    i =begin;
    lbl_loop:
        cc = i>end;
        if(cc)   goto lbl_end;
        sum = sum + i;
        i = i + 1;
        goto lbl_loop;
    lbl_end:
        return sum;
}
```

应用程序负责编译、下载、初始化、启动运动程序。

```
#include"stdafx.h"
#include"conio.h"
#include"gts.h"

int main(int argc, char* argv[])
{
    short rtn;
    short funId;
    TVarInfo sum, begin, end;
    double value;
    TThreadSts thread;

    // 打开运动控制器
    rtn = GTN_Open();
    printf(" GTN_Open()=%d\n", rtn);
    // 复位运动控制器
    rtn = GTN_Reset(1);
    printf(" GTN_Reset()=%d\n", rtn);
    // 下载运动程序sum.bin,
    // 必须保证sum.bin文件位于工程文件夹中
    rtn = GTN_Download(1,"sum.bin");
    printf(" GTN_Download()=%d\n", rtn);
    // 获取函数ID
    rtn = GTN_GetFunId("add", &funId);
    printf(" GTN_GetFunId()=%d\n", rtn);
```

```
    // 获取全局变量sum的ID
    rtn = GTN_GetFunId(NULL, "sum", &sum);
    printf(" GTN_GetFunId()=%d\n", rtn);
    // 获取局部变量begin的ID
    rtn = GTN_GetVarId("add", "begin", &begin);
    printf(" GTN_GetVarId()=%d\n", rtn);
    // 获取局部变量end的ID
    rtn = GTN_GetVarId("add", "end", &end);
    printf(" GTN_GetVarId()=%d\n", rtn);
    // 绑定线程，函数，数据页
    rtn = GTN_Bind(1,0, funId, 0);
    printf(" GTN_Bind()=%d\n", rtn);
    value = 0;
    // 初始化运动程序的全局变量sum
    rtn = GTN_SetVarValue(1,-1, &sum, &value);
    printf(" GTN_SetVarValue()=%d\n", rtn);
    value = 1;
    // 初始化运动程序的局部变量begin
    rtn = GTN_SetVarValue(1,0, &begin, &value);
    printf(" GTN_SetVarValue()=%d\n", rtn);
    value = 100;
    // 初始化运动程序的局部变量end
    rtn = GTN_SetVarValue(1,0, &end, &value);
    printf(" GTN_SetVarValue()=%d\n", rtn);
    // 启动线程
    rtn = GTN_RunThread(1,0);
    printf(" GTN_RunThread()=%d\n", rtn);

    do
    {
        // 查询线程状态
        rtn =  GTN_GetThreadSts(1,0, &thread);
        // 查询全局变量sum的值
        rtn =  GTN_GetVarValue(1,-1, &sum, &value);
        printf("run=%d sum=%-10.0lf\n", thread.run, value);
    }while( 1 == thread.run );    // 等待线程运行结束

    getch();
    return 0;
}
```

2. 多线程累加求和

例程 11-2　运动程序多线程累加求和

运动程序代码和例程 11-1 相同。

应用程序负责编译、下载、初始化、启动运动程序。和例程 11-1 不同之处在于启动 2 个线程完成累加

运算任务。

11.4 如何编写运动程序

```
#include"stdafx.h"
#include"conio.h"
#include"gts.h"

int main(int argc, char* argv[])
{
    short rtn;
    short funId;
    TVarInfo sum, begin, end;
    double value;
    TThreadSts thread;

    // 打开运动控制器
    rtn = GTN_Open();
    printf(" GTN_Open()=%d\n", rtn);
    // 复位运动控制器
    rtn = GTN_Reset(1);
    printf(" GTN_Reset()=%d\n", rtn);
    // 下载运动程序sum.bin
    // 必须保证sum.bin文件位于工程文件夹中
    rtn = GTN_Download(1,"sum.bin");
    printf(" GTN_Download()=%d\n", rtn);
    // 获取函数ID
    rtn = GTN_GetFunId("add", &funId);
    printf(" GTN_GetFunId()=%d\n", rtn);
    // 获取全局变量sum的ID
    rtn =   GTN_GetVarId(NULL, "sum", &sum);
    printf(" GTN_GetVarId()=%d\n", rtn);
    // 获取局部变量begin的ID
    rtn = GTN_GetVarId("add", "begin", &begin);
    printf(" GTN_GetVarId()=%d\n", rtn);
    // 获取局部变量end的ID
    rtn = GTN_GetVarId("add", "end", &end);
    printf(" GTN_GetVarId()=%d\n", rtn);
    // 绑定线程，函数，数据页
    rtn = GTN_Bind(1,0, funId, 0);
    printf(" GTN_Bind()=%d\n", rtn);
    // 绑定线程，函数，数据页
    rtn = GTN_Bind(1,1, funId, 1);
    printf(" GTN_Bind()=%d\n", rtn);
    value = 0;
    // 初始化运动程序的全局变量sum
```

```
    rtn = GTN_SetVarValue(1,-1, &sum, &value);
    printf(" GTN_SetVarValue()=%d\n", rtn);
    value = 1;
    // 初始化运动程序的局部变量begin
    rtn = GTN_SetVarValue(1,0, &begin, &value);
    printf(" GTN_SetVarValue()=%d\n", rtn);
    value = 50;
    // 初始化运动程序的局部变量end
    rtn = GTN_SetVarValue(1,0, &end, &value);
    printf(" GTN_SetVarValue()=%d\n", rtn);
    value = 51;
    // 初始化运动程序的局部变量begin
    rtn = GTN_SetVarValue(1,1, &begin, &value);
    printf(" GTN_SetVarValue()=%d\n", rtn);
    value = 100;
    // 初始化运动程序的局部变量end
    rtn = GTN_SetVarValue(1,1, &end, &value);
    printf(" GTN_SetVarValue()=%d\n", rtn);
    // 启动线程
    rtn = GTN_RunThread(1,0);
    printf(" GTN_RunThread()=%d\n", rtn);
    // 启动线程
    rtn = GTN_RunThread(1,1);
    printf(" GTN_RunThread()=%d\n", rtn);

    do
    {
        // 查询线程状态
        rtn = GTN_GetThreadSts(1,0, &thread);
    }while( 1 == thread.run );      // 等待线程运行结束
    do
    {
        // 查询线程状态
        rtn = GTN_GetThreadSts(1,1, &thread);
    }while( 1 == thread.run );      // 等待线程运行结束

    // 查询全局变量sum的值
    rtn = GTN_GetVarValue(1,-1, &sum, &value);
    printf("sum=%-10.0lf", value);

    getch();
    return 0;
}
```

11.4.1 语言元素

1. 数据类型

支持整型和浮点型 2 种数据类型。

- 整型 32 位，取值范围是-2, 147, 483, 648 ~ 2, 147, 483, 647。
- 浮点型采用定点格式，32 位整数，16 位小数。所能表示的最小精度为 (1/2)^16=0.0000152587890625。

2. 常量

可以在程序中直接使用立即数和宏。立即数可以是 10 进制整数、16 进制整数和浮点数。

3. 变量

可以声明局部变量和全局变量。每个函数最多可声明 1024 个局部变量。全局变量最多可声明 1024 个。整型类型说明符为 int。浮点型类型说明符为 double。

4. 数组

支持一维数组，支持常量下标索引和变量下标索引。

不支持多维数组，不支持用数组元素进行下标索引。

5. 函数

函数可以定义返回值类型和输入形参类型。

不支持在函数中调用自定义函数，但是可以调用 GT 运动控制指令。

6. 数据类型转换

支持强制数据类型转换。强制数据类型转换符有 int, double。

- 数据类型转换符必须加括号，如 a=(int)b
- 数据类型转换不会改变变量本身的数据类型定义

11.4.2 运算指令

支持算术运算、逻辑运算、关系运算、位运算，语法规则和 C 语言相同，但是不支持复杂表达式，只能使用 2 个操作数进行运算，而且这 2 个操作数的数据类型必须相同。

注意：由于运动程序中的浮点数据类型只有 16 位小数精度，请不要在运动程序中进行高精度浮点运算。

1. 算术运算

用于各类数值运算。包括加(+)、减(-)、乘(*)、除(/)、求余(或称模运算，%)共五种。

2. 逻辑运算

逻辑运算包括与(&&)、或(||)、非(!)三种。参数可以是整型变量、或者整型常数。

3. 关系运算

关系运算符用于比较运算。包括大于(>)、小于(<)、等于(==)、大于等于(>=)、小于等于(<=)和不等于(!=)六种。参与比较的参数类型必须一致。

4. 位运算

参与运算的量，按二进制位进行运算。包括位与(&)、位或(|)、位非(~)、位异或(^)、左移(<<)、右移(>>)六种。

5. 流程控制

支持条件跳转、条件返回，语法规则和 C 语言相同。

1) 条件跳转如下所示：

```
if(var) goto label;
```

当条件变量 var 非 0 时，跳转到标记为 label 的指令。

不支持表达式作为判断条件。

2) 条件返回如下所示：

```
if(var) goto value;
```

当条件变量 var 非 0 时，程序返回，返回值为 value。

不支持表达式作为判断条件。

11.4.3 流程控制与标准 C 语言的流程控制对比

(1) While 语句

运动程序实现：

```
int i, sum, cc;
i = 1;
sum = 0;
lbl_loop:
    cc = i> 10;
    if(cc)   goto lbl_end;
    sum = sum + i;
    i = i + 1;
    goto lbl_loop;
lbl_end:
……
```

标准 C 实现：

```
int i, sum;
i = 1;
sum = 0;
while(i<= 10)
{
    sum = sum + i;
    i = i + 1;
}
```

以上程序等同于：

运动程序实现：

```
int i, sum, cc;
i = 1;
sum = 0;
lbl_loop:
```

标准 C 实现：

```
int i, sum;
i = 1;
sum = 0;
while(i<= 10)
```

```
        sum = sum + i;
        i = i + 1;

    cc = i<= 10;
    if(cc) goto lbl_loop;
lbl_end:
……
```

```
{
    sum = sum + i;
    i = i + 1;
}
```

(2) for 语句

运动程序实现：

```
int i, sum, cc;
i = 1;
sum = 0;
lbl_loop:
    cc = i> 10;
    if(cc)  goto lbl_end;
    sum = sum + i;
    i = i + 1;
    goto lbl_loop;
lbl_end:
……
```

标准 C 实现：

```
int i, sum;
sum = 0;
for(i=1;i<= 10;i++)
{
    sum = sum + i;
}
```

(3) if… else 语句

运动程序实现：

```
int a, b;
int cc;
a = 5;
b = 10;
cc = a<=b;
    if(cc)  goto lbl_Jump;
a = a - b;
goto lbl_end;
lbl_Jump:
    a = b - a;
lbl_end:
  ……
```

标准 C 实现：

```
int a, b;
a = 5;
b = 10;
if(a>b)
{
    a = a - b;
}
else
{
    a = b - a;
}
```

(4) switch…case

运动程序实现：

```
#define MC_GPO 12   //注意：该宏定义应放在函数体外
int a;
int cc;
a = 5;
cc = a==1;
```

标准 C 实现：

```
int a;

a = 5;
switch(a)
{
case 1:
```

```
    if(cc)   goto lbl_1;
cc = a==2;
    if(cc)   goto lbl_2;
cc = a==5;
    if(cc)   goto lbl_3;
goto lbl_end;
lbl_1:
    GT_SetDo (MC_GPO,0x01);
    goto lbl_end;
lbl_2:
    GT_SetDo (MC_GPO,0x02);
    goto lbl_end;
lbl_3:
    GT_SetDo(MC_GPO,0x04);
    goto lbl_end;
lbl_end:
……
```

```
        GT_SetDo(MC_GPO,0x01);
        break;
    case 2:
        GT_SetDo (MC_GPO,0x02);
        break;
    case 5:
        GT_SetDo(MC_GPO,0x04);
break;
    default:
        break;
}
```

编写运动程序时：

1. 对于 GT 指令，其指令的参数个数必须写全；
2. GT 指令中的参数宏定义，在运动程序当中不支持，需要重新定义。

11.5 可在运动程序中使用的指令

表 11-2 可在运动程序中使用的指令

编号	指令	指令	指令
1	GT_AlarmOff	GT_GetExtIoValue	GT_SetDataPage
4	GT_AlarmOn	GT_GetGearMaster	GT_SetDiReverseCount
7	GT_AxisOn	GT_GetGearRatio	GT_SetDo
10	GT_CrdSpace	GT_GetJogPrm	GT_SetDoBit
13	GT_CrdStart	GT_GetMtrBias	GT_SetExtIoBit
16	GT_CrdStatus	GT_GetMtrLmt	GT_SetExtIoValue
19	GT_CtrlMode	GT_GetPid	GT_SetGearMaster
22	GT_Delay	GT_GetPos	GT_SetGearRatio
25	GT_DelayHighPrecision	GT_GetPosErr	GT_SetJogPrm
28	GT_EncOff	GT_GetPrfMode	GT_SetMtrBias
31	GT_EncOn	GT_GetPrfVel	GT_SetMtrLmt
34	GT_EncScale	GT_GetSoftLimit	GT_SetOverride
37	GT_GearStart	GT_GetStopDec	GT_SetPid
40	GT_GetAxisBand	GT_GetSts	GT_SetPos
43	GT_GetAxisEncPos	GT_GetTrapPrm	GT_SetPosErr
46	GT_GetAxisPrfVel	GT_GetTriggerStatus	GT_SetStopDec
49	GT_GetCaptureOffset	GT_GetVel	GT_SetTrapPrm
52	GT_GetClock	GT_LinkCaptureOffset	GT_SetTrigger

编号	指令	指令	指令
55	GT_GetControlFilter	GT_LmtSns	GT_SetVel
58	GT_GetCrdVel	GT_LmtsOff	GT_StepDir
61	GT_GetDac	GT_LmtsOn	GT_StepPulse
64	GT_GetDacValue	GT_PrfGear	GT_Stop
67	GT_GetDi	GT_PrfJog	GT_SynchAxisPos
70	GT_GetDiReverseCount	GT_PrfTrap	GT_Update
73	GT_GetDo	GT_ProfileScale	GT_ZeroPos
76	GT_GetEncPos	GT_SetControlFilter	GT_ZeroPos
79	GT_GetExtIoBit	GT_SetDac	GTN_GetAxisEncAcc

使运动程序功能必须注意以下几点：

1. 在编写运动程序过程中必须使用 GT 指令，GT 指令请参照 GTS 相关的编程手册
2. 如果想直接使用 GTN 指令，请联系技服人员，使用 DLM 功能模块

第12章 其它指令

12.1 本章简介

本章介绍运动控制器为用户提供的其他指令。包括：打开/关闭运动控制器、读取固件版本号、读取系统时钟、打开/关闭电机使能信号、维护位置值、电机到位检测、设置 PID 参数、反向间隙补偿、自动回原点、位置比较输出。

提示	本手册中所有字体为蓝色的指令（如 GTN_PrfTrap）均带有超级链接，点击可跳转至指令说明。

12.2 打开/关闭运动控制器

表 12-1 打开/关闭运动控制器指令列表

指令	说明	页码
GTN_Open	打开运动控制器(按照默认配置)	224
GTN_Close	关闭运动控制器	175
GTN_Reset	复位运动控制器	232

在使用运动控制器之前，首先需要使用 GTN_Open 指令打开运动控制器，和运动控制器建立通讯；在使用运动控制器结束之后，退出应用程序时，应当调用 GTN_Close 指令关闭运动控制器。

用户不应当在程序中反复地调用 GTN_Open 和 GTN_Close 指令，去频繁地打开和关闭运动控制器。只要在应用程序初始化阶段调用一次 GTN_Open 在应用程序退出时调用一次 GTN_Close 即可。

调用 GTN_Reset 指令将使运动控制器的所有寄存器恢复到默认状态，一般在打开运动控制器之后调用该指令。

12.3 读取固件、动态链接库版本号

表 12-2 读取固件版本号指令列表

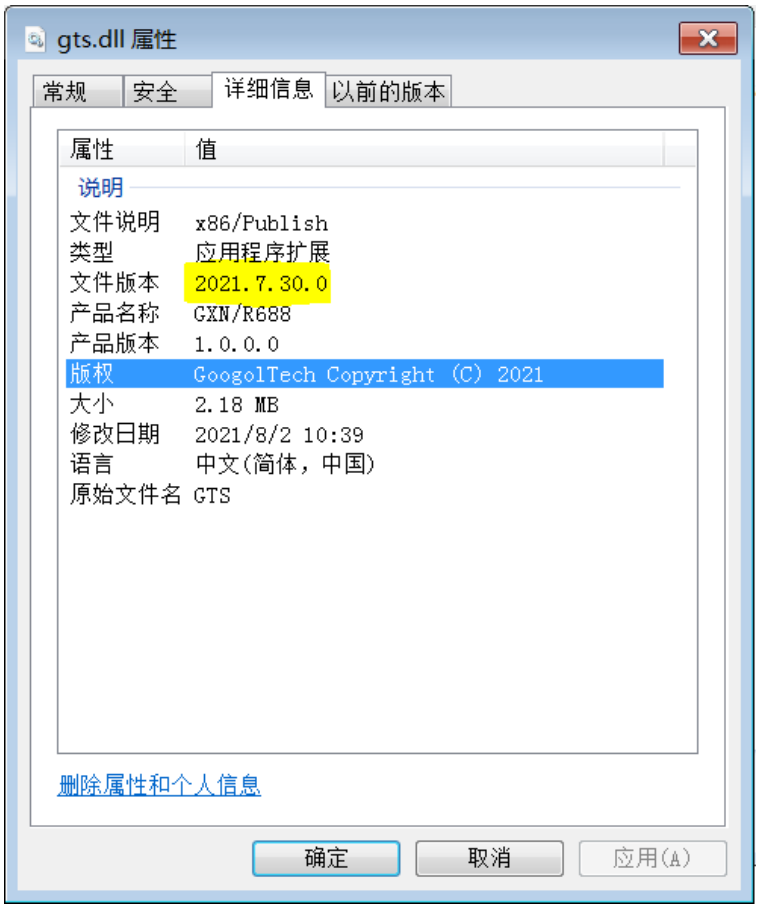
指令	说明	页码
GTN_GetVersionEx	读取运动控制器固件的版本号	216
GTN_GetTerminalVersion	读取端子板固件的版本号	211

为了方便用户核对运动控制器固件版本，提供 GTN_GetVersionEx 指令来读取运动控制器固件版本号，版本号是一个含有 18 个字符的字符串：aaa bbbbbb ccc dddddd。具体的定义如表 12-3 所示。

表 12-3 固件版本号的定义格式

aaa	固件 1 的版本号，如 100，即表示版本号为：1.00
bbbbbb	固件 1 的版本号的生成时间，如 090908，即表示该版本生成于：2009 年 9 月 8 日
ccc	固件 2 的版本号
dddddd	固件 2 的版本号的生成时间

用户读取动态链接库的版本，点击 gts.dll，右键->属性->详细信息处可以查看动态链接库的版本。



文件说明	动态链接库的说明，x86/Publish 表示是 32 位的发布版本的动态链接库。
文件版本	动态链接库的版本日期，如 2021.7.30.0，表示该动态链接库发布于：2021 年 7 月 30 日

例程 12-1 读取运动控制器版本号

```
short rtn;
TVersion version;

rtn =  GTN_Open(5, 2);

rtn = GTN_GetVersionEx(1, 1, &version);        // 读取固件版本号
rtn = GTN_Close()
```

12.4 读取系统时钟

表 12-4 读取系统时钟指令列表

指令	说明	页码
GTN_GetClock	读取运动控制器系统时钟	188
GTN_GetClockHighPrecision	读取运动控制器系统高精度时钟	188

运动控制器上电初始化之后，内部计数时钟从 0 开始计数，每 1 毫秒增加 1，通过 GTN_GetClock 指令可以读取该计数时钟的值。 GTN_GetClockHighPrecision 读取的时钟每个伺服周期增加 1，调用 GTN_Reset 指令将会使计数时钟值清零。

12.5 打开/关闭电机使能信号

表 12-5 打开/关闭电机使能信号指令列表

指令	说明	页码
GTN_AxisOn	打开驱动器使能	169
GTN_AxisOff	关闭驱动器使能	169

调用 GTN_AxisOn 指令将打开指定控制轴所连电机的伺服使能信号，使指定控制轴进入控制状态。如果在系统配置时，没有数字量输出与该 axis 关联，则该指令将会无效(5.3.7)。如果运动控制器被配置成了伺服控制方式(第 4 章)，那么必须首先设置指定轴的位置环的 PID 参数。

12.6 维护位置值

表 12-6 维护位置值指令列表

指令	说明	页码
GTN_SetPrfPos	修改指定轴的规划位置	249
GTN_SynchAxisPos	axis 合成规划位置和所关联的 profile 同步 axis 合成编码器位置和所关联的 encoder 同步	258
GTN_ZeroPos	清零规划位置和实际位置，并进行零漂补偿	259

第三章系统配置里提到，axis 含有 profile 和 encoder 的当量变换的功能，如果调用了 GTN_SetPrfPos 指令或者 GTN_SetEncPos 指令之后，profile 的输出值或者 encoder 的输出发生了变化，如果需要将 axis 当量变换之后的值与 profile 或者 encoder 的值同步，需要调用 GTN_SynchAxisPos 指令。

12.7 设置 PID 参数

表 12-7 设置 PID 参数指令列表

指令	说明	页码
GTN_SetControlFilter	设定 PID 索引，支持 3 组 PID 参数	235
GTN_GetControlFilter	读取当前 PID 索引	189
GTN_SetPid	设置 PID 参数	245
GTN_GetPid	读取 PID 参数	201

PID 参数说明如下：

(1) kp：比例增益；该系数的作用是改变控制系统的动态响应速度。

(2) ki：积分增益；该系数的作用是消除控制系统的稳态误差。

(3) kd：微分增益；该系数的作用是改善控制系统的动态性能。其作用与输出的偏差变化速度成比例，能够预测偏差的变化，产生超前控制作用，以阻止偏差的变化。一般不需要设置。

(4) kvff：速度前馈系数；该参数的作用是减小跟随误差。跟随误差即某一时刻的规划值与编码反馈值的差值。

(5) kaff：加速度前馈系数；一般不需要调节。

(6) integralLimit：积分饱和极限；该参数一般默认设置。

(7) derivativeLimit：微分饱和极限；该参数一般默认设置。

(8) limit：控制量输出饱和极限；该参数与驱动器的电压接收范围有关，默认是 32767，即对应范围是[-10V, 10V]。若驱动器的接收电压范围是[-5V, 5V]，则该参数为 16384，如果第一次调 PID 参数可以把此值设小，以防止发生正反馈，保护设备。当确定反馈正常时，需把此值设成对应的电压值，如很多驱动器默认的接收的范围是[-10V, 10V]。

调节 PID 的相关说明：

(1) 首先调节 kp，当电机很容易被移动时，说明 kp 太小，应慢慢加大 kp，但不能将电机调节的“过硬”，kp 太大了可能会引起高频振荡，这时就需要减小该参数。一般可以从 kp = 1 缓慢增大调试，利用 MotionStudio 的 TUNING 功能，查看响应曲线，一般如果控制旋转电机，走点位运动，只调此参数即可。

(2) 在响应速度满足的情况下，若跟随误差较大，则需要调节速度前馈系数，以减小跟随误差。此参数在连续轨迹运动的场合使用较多；但在点位运动，如果用直线电机，调整此参数，对电机快速响应也有一定的效果。

(3) 如果按照第一、第二步调试，效果还不能满足客户的需求，请调整驱动器的相关参数，如果驱动器的相关参数已调整好，请重复第一到第三步的操作，直到效果能满足用户的要求。

(4) 若需要消除稳态误差，调节一般先将 DA 零漂清除，建议先不要调节 ki 值。清除零漂方法：

1) GTN_Open 之后，调用 GTN_Reset，然后将相应轴设置成闭环控制模式（注意：设置 PID 参数时，只需要设置 kp=1，其他 PID 参数保持默认），之后调用 GTN_AxisOn。

2) 待电机稳定之后，调用 GTN_GetEncPos 获取对应轴的位置编码值，接着调用 GTN_SetMtrBias 来设置 DA 零漂值，该零漂值为编码值的负值。设置完之后，延时一段时间，然后再调用 GTN_GetEncPos，若此时的值还不为 0，则再次用此次的编码值与上次的编码值相加，将其和的负值再次设置 DA 零漂。

运动控制器支持设置 3 组 PID 参数，并且支持运动时在各组 PID 参数之间进行切换，通过调用 GTN_SetControlFilter 指令来切换 PID 参数。

12.8 电机到位检测

表 12-8 电机到位检测指令列表

指令	说明	页码
GTN_SetAxisBand	设置轴到位误差带 规划器静止，规划位置和实际位置的误差小于设定误差带，并且在误差带内保持设定时间后，轴状态的电机到位标志位置起到位标志	232
GTN_GetAxisBand	读取轴到位误差带	184

用户使用伺服电机时，由于伺服电机在运动的过程中可能会存在运动滞后，会出现规划停止，而实际位置并没有到位的情况。用户可以使用运动控制器的运动到位检测功能来判断电机是否实际到位。运动控制器默认该功能是无效的，当调用 GTN_SetAxisBand 指令设置了相应的误差带和保持时间之后，该功能生效。

该功能生效后，当规划器处于静止状态，即轴状态寄存器 bit10 为 0(参见 6.3.1)，并且规划位置和编码器位置的误差在设定的误差带内保持了设定时间，轴状态寄存器 bit11 将被置 1(参见 6.3.1)。当规划器运动，或者规划位置和编码器位置的误差超出误差带时立即清 0。检测电机到位标志可以保证系统的定位精度，应当根据机械系统的实际情况设置合适的到位误差带和误差带保持时间。如果到位误差带设置的太小，或者误差带保持时间太长，都会使到位时间增长，影响加工效率。

如图 12-1 所示。当轴 1 启动电机到位检测功能，设置误差带为 100pulse，误差带保持时间为 500μs，当电机在 1000pulse 位置处静止时，会有震荡。控制器判断其震荡位于误差带内 500μs 后，就将轴状态的 bit11 电机到位标志置 1。蓝色阴影区域表示电机到位标志还未置 1，橙色阴影区域表示已置 1。可以看出，误差带保持时间越短，误差带越大，控制器会在越短时间内将电机到位标志置 1，但是并不是越短越好。电机是否连接机械本体，也会影响控制器判断电机到位所需的时间。

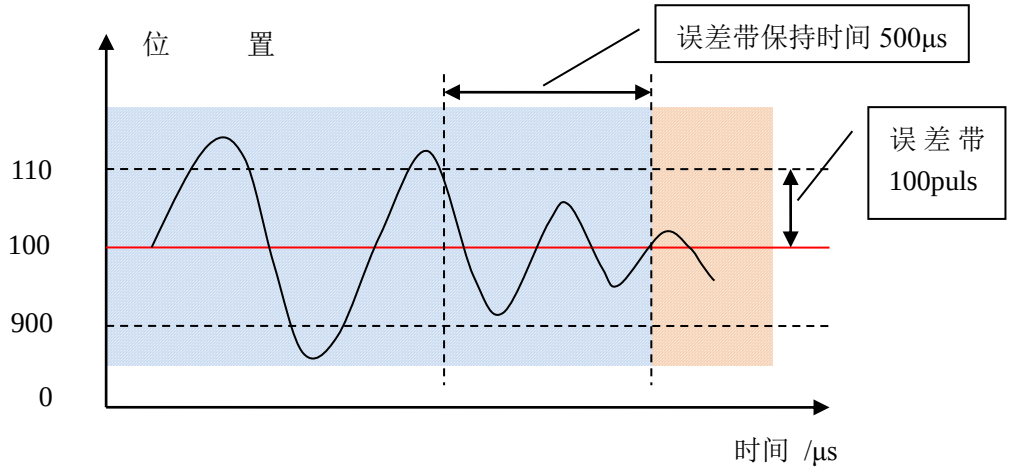


图 12-1　电机到位检测功能

注意	使用电机到位检测功能必须注意以下几点： 1. axis 正确关联编码器，并且编码器方向和规划运动方向必须一致。 2. 正确设置到位误差带，默认情况下到位误差带无效 3. 调用 GTN_ZeroPos 进行对位置进行清零，同时进行自动零漂补偿。

例程 12-2　电机到位检测功能

下面这个例程示例电机到位检测的使用方法。一个轴运动到位以后，启动另一个轴的运动。电机到位的运动状态检测如图 12-2 所示。

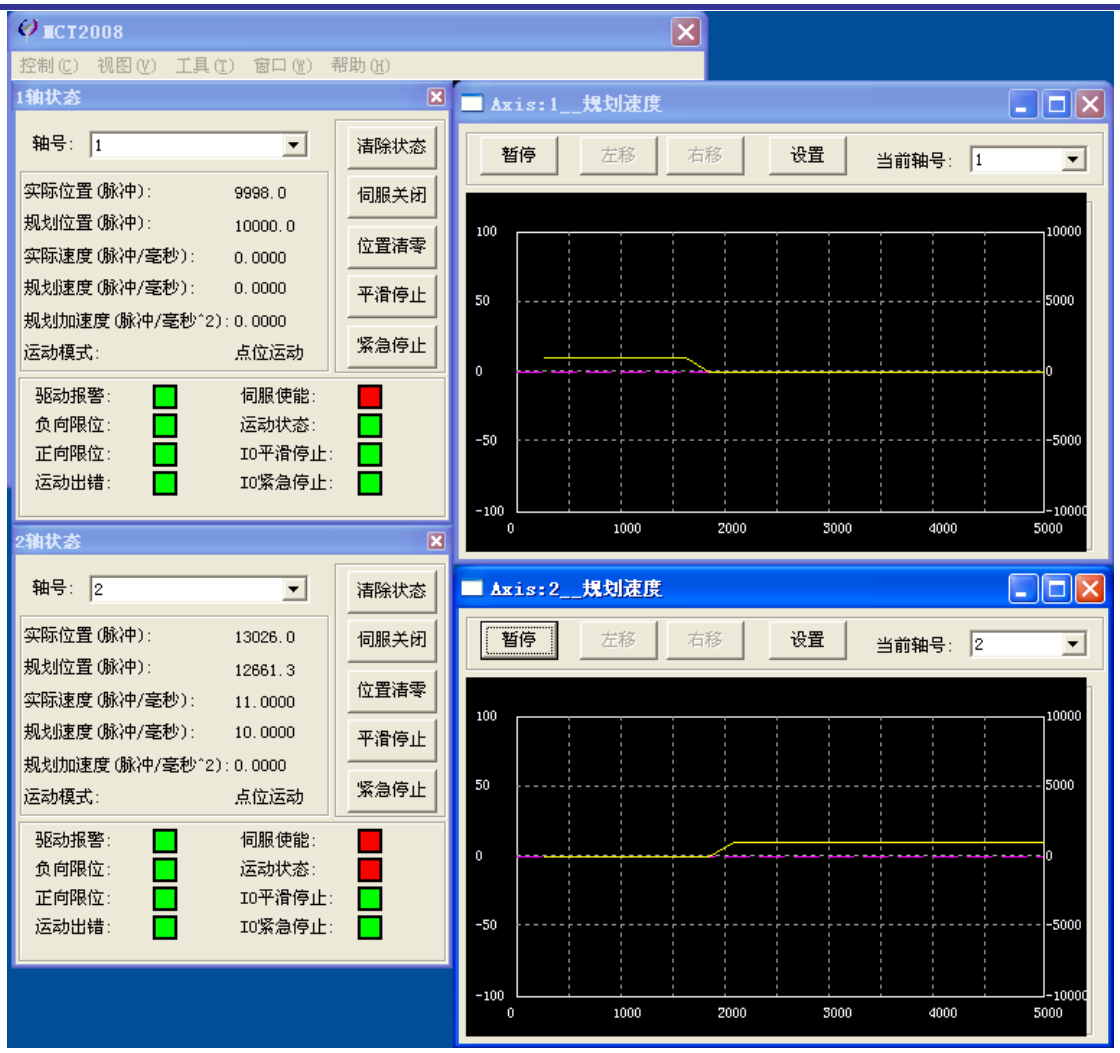


图 12-2 电机到位的运动状态检测

```
#include "stdafx.h"
#include "conio.h"
#include "windows.h"
#include "gts.h"

#define AXIS_X                  1
#define AXIS_Y                  2

int main(int argc, char* argv[])
{
    short sRtn;
    TPid pid;
    TTrapPrm trap;
    long sts;
    long posX, posY;
    double prfPos, prfVel;

    // 打开运动控制器
    sRtn = GTN_Open();
    commandhandler(" GTN_Open", sRtn);
```

```
// 复位控制器
sRtn = GTN_Reset(1);
commandhandler(" GTN_Reset", sRtn);
// 配置运动控制器为伺服模式
sRtn = GTN_LoadConfig(1,"servo.cfg");
commandhandler(" GTN_LoadConfig", sRtn);
// 延时一段时间
Sleep(100);
// 清除各轴的报警和限位
sRtn = GTN_SetSense (1,MC_ENCODER,AXIS_X, 0);
commandhandler(" GTN_SetSense", sRtn);
sRtn = GTN_SetSense (1,MC_ENCODER,AXIS_Y, 0);
commandhandler(" GTN_SetSense", sRtn);
sRtn = GTN_ClrSts(1,1, 8);
commandhandler(" GTN_ClrSts", sRtn);
// 读取X轴PID参数
sRtn = GTN_GetPid(1,AXIS_X, 1, &pid);
commandhandler(" GTN_GetPid", sRtn);
pid.kp = 10;
// 更新X轴PID参数
sRtn = GTN_SetPid(1,AXIS_X, 1, &pid);
commandhandler(" GTN_SetPid", sRtn);
// 读取Y轴PID参数
sRtn = GTN_GetPid(1,AXIS_Y, 1, &pid);
commandhandler(" GTN_GetPid", sRtn);
pid.kp = 10;
// 更新Y轴PID参数
sRtn = GTN_SetPid(1,AXIS_Y, 1, &pid);
commandhandler(" GTN_SetPid", sRtn);
// X轴伺服使能
sRtn = GTN_AxisOn(1,AXIS_X);
commandhandler(" GTN_AxisOn", sRtn);
// Y轴伺服使能
sRtn = GTN_AxisOn(1,AXIS_Y);
commandhandler(" GTN_AxisOn", sRtn);
// 延时一段时间，等待伺服稳定
Sleep(200);
// 位置清零，并进行自动零漂补偿
sRtn = GTN_ZeroPos (1,AXIS_X);
commandhandler(" GTN_ZeroPos", sRtn);
// 设置X轴到位误差带
sRtn = GTN_SetAxisBand(1,AXIS_X, 20, 5);
commandhandler(" GTN_SetAxisBand", sRtn);
// 位置清零，并进行自动零漂补偿
sRtn= GTN_ZeroPos(1,AXIS_Y);
commandhandler(" GTN_ZeroPos", sRtn);
```

```
// 设置Y轴到位误差带
sRtn = GTN_SetAxisBand(1,AXIS_Y, 20, 5);
commandhandler(" GTN_SetAxisBand", sRtn);
// X轴设为点位模式
sRtn = GTN_PrfTrap(1,AXIS_X);
commandhandler(" GTN_PrfTrap", sRtn);
// 读取X轴点位运动参数
sRtn = GTN_GetTrapPrm(1,AXIS_X, &trap);
commandhandler(" GTN_GetTrapPrm", sRtn);
trap.acc = 1;
trap.dec = 0.5;
// 设置X轴点位运动参数
sRtn = GTN_SetTrapPrm(1,AXIS_X, &trap);
commandhandler(" GTN_SetTrapPrm", sRtn);
// 设置X轴的目标速度
sRtn = GTN_SetVel(1,AXIS_X, 10);
commandhandler(" GTN_SetVel", sRtn);
// Y轴设为点位模式
sRtn = GTN_PrfTrap(1,AXIS_Y);
commandhandler(" GTN_PrfTrap", sRtn);
// 读取Y轴点位运动参数
sRtn = GTN_GetTrapPrm(1,AXIS_Y, &trap);
commandhandler(" GTN_GetTrapPrm", sRtn);
trap.acc = 1;
trap.dec = 0.5;
// 设置Y轴点位运动参数
sRtn = GTN_SetTrapPrm(1,AXIS_Y, &trap);
commandhandler(" GTN_SetTrapPrm", sRtn);
// 设置Y轴的目标速度
sRtn = GTN_SetVel(1,AXIS_Y, 10);
commandhandler(" GTN_SetVel", sRtn);

posX = 10000;
posY = 20000;
while(!kbhit())
{
    // 设置X轴目标位置
    sRtn = GTN_SetPos(1,AXIS_X, posX);
    commandhandler(" GTN_SetPos", sRtn);
    // 启动X轴的运动
    sRtn = GTN_Update(1,1<<(AXIS_X-1));
    commandhandler(" GTN_Update", sRtn);
    posX = - posX;
    // 等待X轴进入误差带
    do
    {
```

```
            GTN_GetSts(1,AXIS_X, &sts);
            GTN_GetPrfPos(1,AXIS_X, &prfPos);
            GTN_GetPrfVel(1,AXIS_X, &prfVel);
          printf("x pos=%-10.2lf vel=%-6.2lf\r", prfPos, prfVel);
        }while( 0x800 != ( sts& 0x800 ) );

        printf("\n");
        // 设置Y轴目标位置
        sRtn = GTN_SetPos(1,AXIS_Y, posY);
        commandhandler(" GTN_SetPos", sRtn);
        // 启动Y轴的运动
        sRtn = GTN_Update (1,1<<(AXIS_Y-1));
        commandhandler(" GTN_Update", sRtn);
        posY = - posY;
        // 等待Y轴进入误差带
        do
        {
            GTN_GetSts(1,AXIS_Y, &sts);
            GTN_GetPrfPos(1,AXIS_Y, &prfPos);
            GTN_GetPrfVel(1,AXIS_Y, &prfVel);
          printf("y pos=%-10.2lf vel=%-6.2lf\r", prfPos, prfVel);
        }while( 0x800 != ( sts& 0x800 ) );
        printf("\n");
    }
    // 伺服关闭
    sRtn = GTN_AxisOff (1,AXIS_X);
    printf("\n GTN_AxisOff()=%d, Axis:%d\n", sRtn, AXIS_X);
    sRtn = GTN_AxisOff (1,AXIS_Y);
    printf("\n GTN_AxisOff()=%d, Axis:%d\n", sRtn, AXIS_Y);
    return 0;

}
```

12.9 反向间隙补偿

表 12-9 反向间隙补偿指令列表

指令	说明	页码
GTN_SetBacklash	设置反向间隙补偿的相关参数	234
GTN_GetBacklash	读取反向间隙补偿的相关参数	187

反向间隙误差是指由于传动链中机械间隙的存在，执行部件在运动过程中，从正向运动变为负向运动时，或者从负向运动变为正向运动时，执行部件的运动量与理论量存在误差，最后将反映为叠加至工件上的加工精度的误差。为了消除反向间隙误差，提高机器的加工精度和定位精度，该控制器提供了反向间隙误差补偿功能。用户只要在初始化的时候调用相应的反向间隙误差补偿功能指令 GTN_SetBacklash 设置了相应的参数，反向间隙误差补偿功能将会生效；也可以通过指令 GTN_SetBacklash 来关闭反向间隙误差补

偿功能。

用户可以设置反向间隙误差补偿量的叠加速度，可以瞬间(一个控制周期内)叠加到输出量上，也可以选择以一定的速度叠加到输出量上。通过设置指令 GTN_SetBacklash 的 compChangeValue 参数来实现，当 compChangeValue 的值为 0 或者大于等于 compValue 的值时，则表示误差补偿量将瞬间叠加到输出量上，当为其他值时，表示误差补偿量的叠加速度，单位是：pulse/ms。

反向间隙误差补偿方向指的是，反向间隙误差补偿是沿正方向补偿还是沿负方向补偿。如果指令 GTN_SetBacklash 的参数 compDir 参数设置为 0 时，则只有电机从正方向转为负方向运动时，反向间隙补偿量生效，当电机向正方向运动时，反向间隙补偿量为 0。如果用户设置了补偿量的变化速度，则从正方向转为负方向时，补偿量以 compChangeValue 的速度叠加到 compValue 的值，当从负方向转为正方向时，补偿量从 compValue 以 compChangeValue 的速度减小为 0。这种情况下，用户应该在回零之后，让工作台向正方向运动一定的距离，以保证正方向运动没有间隙存在。

当指令 GTN_SetBacklash 的参数 compDir 参数设置为 1 时，则只有电机从负方向转为正方向运动时，反向间隙补偿量生效，当电机向负方向运动时，反向间隙补偿量为 0。如果用户设置了补偿量的变化速度，则从负方向转为正方向时，补偿量以 compChangeValue 的速度叠加到 compValue 的值，当从正方向转为负方向时，补偿量从 compValue 以 compChangeValue 的速度减小为 0。这种情况下，用户应该在回零之后，让工作台向负方向运动一定的距离，以保证负方向运动没有间隙存在。

反向间隙补偿量会直接叠加到运动控制器的输出量上，当用户读取规划位置时，不会读到反向间隙补偿量。但是用户如果读取电机编码器的值，将会读到反向间隙的补偿量。

12.10 螺距误差补偿

表 12-10 螺距误差补偿指令列表

指令	说明	页码
GTN_SetLeadScrewComp	加载补偿表	244
GTN_EnableLeadScrewComp	是否开启补偿	180

螺距误差是指由于传动链中丝杆的制造误差或安装误差，导致电机旋转的角度和丝杆的直线位移成非线性关系，最终将导致实际和理论的直线位移存在误差。为了消除螺距误差，提高机器的加工精度，该控制器提供了螺距误差补偿功能。

用户只要在初始化的时候调用相应的螺距误差补偿功能指令 GTN_SetLeadScrewComp 加载相应的补偿表，然后调用指令 GTN_EnableLeadScrewComp 开启螺距误差补偿功能，也可以通过指令 GTN_EnableLeadScrewComp 来关闭螺距误差补偿功能。

螺距误差补偿表需要根据实际机器测量获取。

如下图所示，补偿起始位置为 10000，补偿总长为 40000，设置补偿节点数为 5，因此补偿节点位置为{10000,20000,30000,40000,50000}，正向运动时从 10000 开始运动 40000 个脉冲，测量到达上述补偿节点位置时，规划位置和实际位置的差值（编码器位置减去规划位置）依次为 10->20->30->(-10)->(-20)；负向运动时从 50000 开始运动 40000 个脉冲，测量达到上述补偿节点位置时，规划位置和实际位置的差值依次为(-10)->(-20)->10->5->10。则{10,20,30,-10,-20}作为正向补偿值，{10,5,10,-20,-10}作为负向补偿值（注意负向补偿值的顺序）。

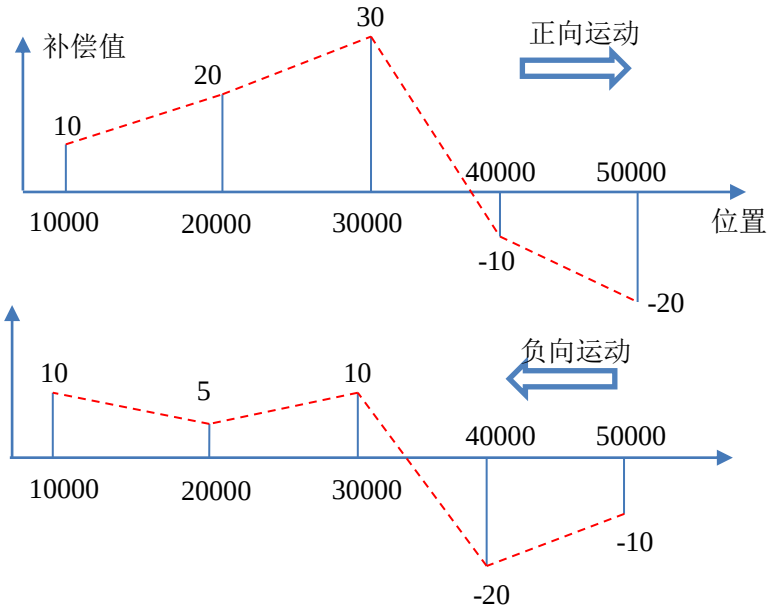


图 12-3 螺距误差补偿示意图

例程 12-3 螺距误差补偿例程

```
#define COMP_AXIS          1
#define LEAD_SCREW_N       5
#define CORE       1
/*-------------------------螺距误差补偿-----------------------------------------------*/
short sRtn;

long comPos[LEAD_SCREW_N] = {10,20,30,-10,-20};//正向补偿表

long comNeg[LEAD_SCREW_N] = {10,5,10,-20,-10};//负向补偿表

//设置螺距误差补偿表
sRtn = GTN_SetLeadScrewComp(1,1,5,10000,40000,comPos,comNeg);
//启动螺距误差补偿
sRtn = GTN_EnableLeadScrewComp(1,1,true);
```

12.11 二维位置补偿

表 12-11 二维位置补偿指令列表

指令	说明	页码
GTN_SetCompensate2DTable	设置二维补偿表及数据	235
GTN_GetCompensate2DTable	获取二维补偿表参数	188
GTN_SetCompensate2DtableRotationAngle	设置二维补偿表旋转参数。	235
GTN_GetCompensate2DtableRotationAngle	获取二维补偿表旋转参数。	189
GTN_SetCompensate2D	设置二维补偿参数	234
GTN_GetCompensate2D	获取二维补偿参数	188

指令	说明	页码
GTN_GetCompensate2DValue	获取补偿值	189

12.11.1 重点说明

二维补偿适用于这么一种场景：X、Y 轴执行插补对平面进行检测，当遇到凸起或凹陷区域，需要适当调整 Z 轴的高度以适应表面的凹凸不平（如图 12-4 所示）。二维补偿功能正是为上述场景或类似工艺场合开发的补偿校正功能。用户需要事先划出平面上的需要补偿的区域，并给出补偿区域内若干 X、Y 位置上 Z 方向的补偿值，控制器自动通过适当的算法计算出每个 X、Y 位置的 Z 方向补偿值，并在运动过程中自动给予补偿。

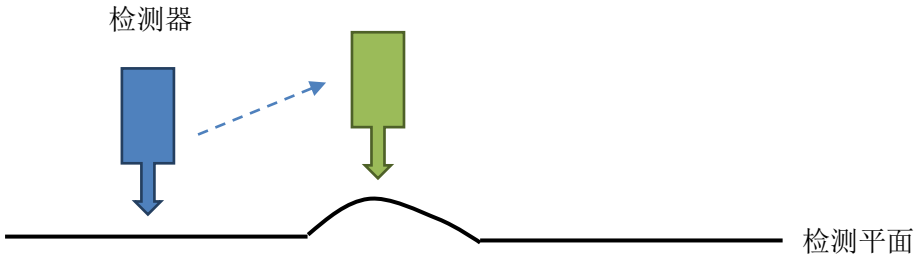


图 12-4　二维位置补偿示意图

如图 12-5 所示，灰色区域为平面运动区域，蓝色区域为需要进行 Z 方向校正的补偿区域，PosBegin 为补偿区域的起始点，给定该补偿区域 Z 方向的 4 个补偿值（即当运动到 PosBegin 的位置 Z 方向需要补偿运动 Z1，运动到 Pos2 位置 Z 方向需要补偿 Z2……），把这些已知的补偿值设置到控制器，则启动运动后，在补偿区域内到达某个位置则在 Z 方向补偿相应的值。

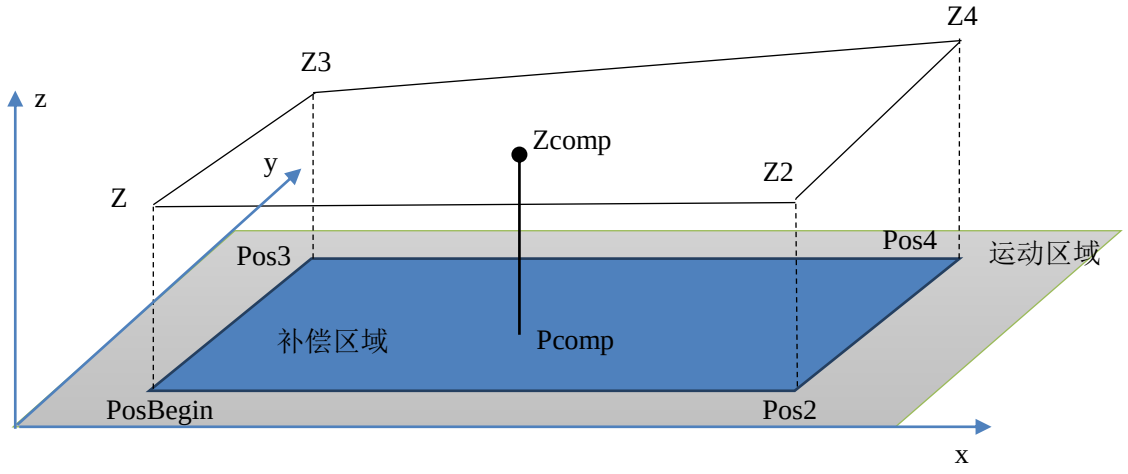


图 12-5　Z 向补偿示意图

12.11.2 例程

例程 12-4　二维位置补偿例程

```
int main(int argc, char* argv[])
{
    short sRtn;
    short run;
    long segment;
    //初始化控制器
```

```
sRtn= GTN_Open();                //打开运动卡
sRtn= GTN_Reset(1);              //复位
sRtn= GTN_ClrSts(1,1,4);         //清除轴状态
sRtn= GTN_ZeroPos(1,1,4); //位置清零
sRtn= GTN_AxisOn(1,1);           //伺服使能
sRtn= GTN_AxisOn(1,2);
sRtn= GTN_AxisOn(1,3);           //（必须先使能补偿轴）
//设置二维补偿表及参数
TCompensate2DTable table;
short pExternComp;
long data[2][3];
data[0][0] = 10;
data[0][1] = 20;
data[0][2] = 30;
data[1][0] = 100;
data[1][1] = 200;
data[1][2] = 300;
sRtn = GTN_GetCompensate2DTable(1,1,&table,& pExternComp);      //获取二维补偿表

table.count[0] = 3;
table.count[1] = 2;
table.posBegin[0] = 0;        //补偿起点从开始
table.posBegin[1] = 0;
table.step[0] = 50000;               //补偿步长为
table.step[1] = 50000;
pExternComp = 0;                     //是否自动扩展补偿区域。
sRtn = GTN_SetCompensate2DTable(1,1,&table,&data[0][0],pExternComp);      //写入补偿
TCompensate2D comp2D;
sRtn = GTN_GetCompensate2D(1,3,&comp2D);      //获取二维补偿参数
comp2D.enable = 1;
comp2D.tableIndex = 1;
comp2D.axisType[0] = MC_PROFILE;                //查表所使用的 X 位置为规划位置
comp2D.axisIndex[0] = 1;                        //1 轴作为二维补偿运动的 X 轴
comp2D.axisType[1] = MC_PROFILE;
comp2D.axisIndex[1] = 2;                        //2 轴作为二维补偿运动的 Y 轴
sRtn = GTN_SetCompensate2D(1,3,&comp2D);      //设置补偿参数（补偿轴必须先使能）
TCrdPrm crdPrm;                                 //设置插补运动
memset(&crdPrm, 0, sizeof(crdPrm));
crdPrm.dimension=2;
crdPrm.synVelMax=500;
crdPrm.synAccMax=1;
crdPrm.evenTime = 50;
crdPrm.profile[0] = 1;
crdPrm.profile[1] = 2;
sRtn = GTN_SetCrdPrm(1,1, &crdPrm);          //建立号坐标系，设置坐标系参数
sRtn = GTN_CrdClear(1,1, 0);                 //清除此缓存区
```

```
    sRtn = GTN_LnXY(1,1, 50000, 50000, 50, 0.1, 0, 0);      //插补数据
    sRtn = GTN_CrdStart(1,1, 0);                            //启动插补运动
    do
    {
        sRtn = GTN_CrdStatus(1,1,&run,&segment,0);
        doublecomp2DValue;
        sRtn = GTN_GetCompensate2DValue(1,3,&comp2DValue);
        printf("comp2DValue= % lf\r",comp2DValue );
    } while(run==1);              // 等待插补运动停止
    return 0;
}
```

12.12 手轮功能

12.12.1 单轴手轮

表 12-12 单轴手轮功能指令列表

指令	说明	页码
GTN_HandwheelInit	初始化单轴手轮功能	219
GTN_StartHandwheel	启动单轴手轮	256

1. 重点说明

单轴手轮能够实现以下功能：

（1） 轴停止时可以进入和退出手轮模式及设置手轮参数

（2） 手轮单格摇动时从轴可以准确运动设定的距离

（3） 手轮快速摇动时，从轴可以平稳运动

（4） 手轮不摇时，从轴能够根据设置的参数快速停止

2. 示例

该程序以 MPG1 作为主轴，轴 1 作为从轴，使能单轴手轮功能

```
short sRtn;
short core = 1;
short master = 1;
short slave = 1;
sRtn = GTN_AxisOn(core, slave);      //使能从轴，使能前保证从轴无限位报警、无轴报警
sRtn = GTN_HandwheelInit(core, MC_MPG);   //使用 MPG 作为主轴的源
sRtn = GTN_StartHandwheel(core, slave, master,1,100,1,0.5,0.5,100,200);

//如果要退出单轴手轮，可以按以下操作
sRtn = GTN_Stop(core,(1<<(slave-1)),0x0);
```

12.12.2 手轮引导

表 12-13 手轮引导功能指令列表

指令	说明	页码
GTN_SetCrdMPGMode	设置手轮引导参数	236
GTN_GetCrdMPGMode	获取手轮引导参数	190

1. 重点说明

手轮脉冲发生器作为主轴，能够引导插补运动。实现以下功能：

（1） 手轮速度的快慢决定了插补运动的速度

（2） 手轮摇动方向决定插补运动的方向

（3） 可以在自动加工和手轮引导模式之间相互切换（注意需要插补处于停止模式）

（4） 可以设置手轮速度和插补运动速度的比例关系

（5） 在插补缓冲区启动的点位运动也支持手轮引导

（6） 回退时遇到缓冲区 IO 指令停止回退

下图是自动加工与手轮引导模式相互切换的状态流程：

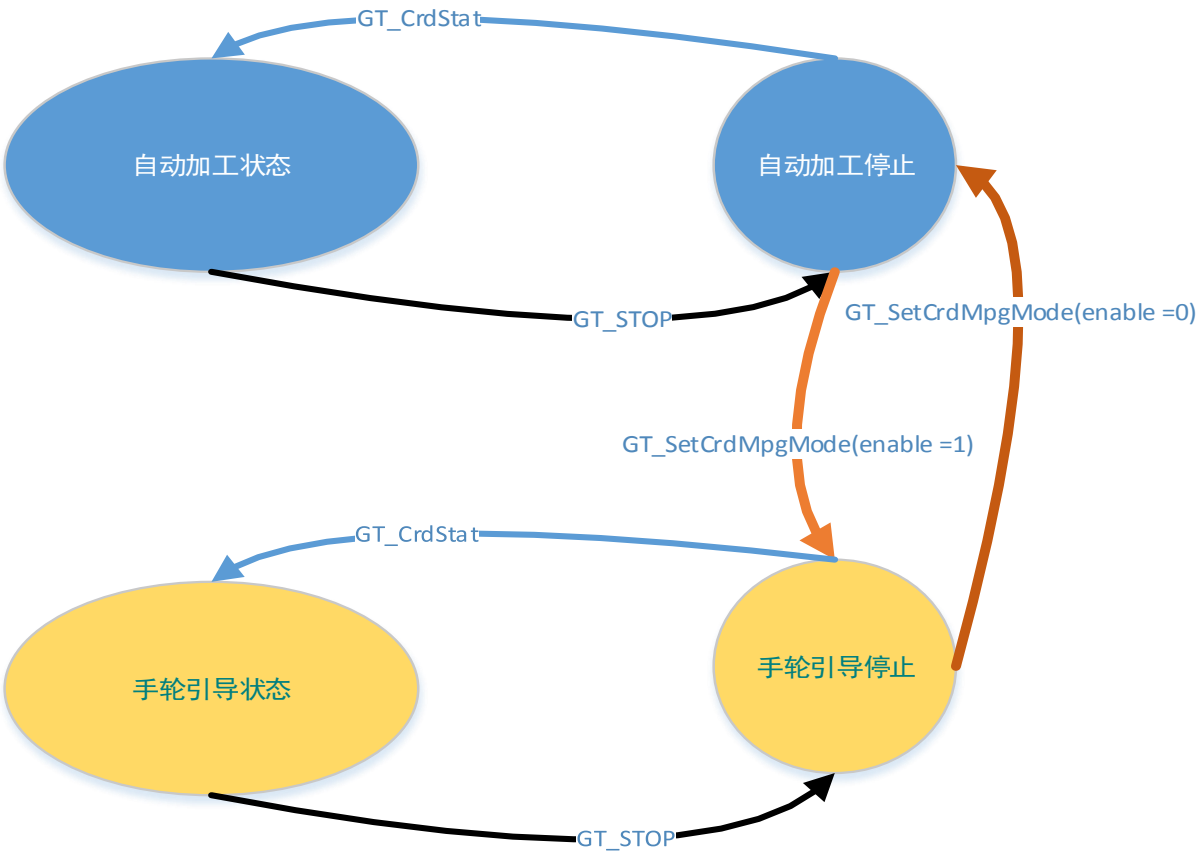


图 12-6 自动加工与手轮引导模式相互切换的状态流程

GTN_SetCrdMPGMode 指令设置的 materEven 和 slaveEven 按下面的关系设置：

插补速度倍率=手轮速度*齿轮比，齿轮比=slaveEven/masterEven。

图 12-7 手轮摇动速度与坐标系速度关系图

2. 示例

该例程创建了一个二维插补坐标系，把 MPG1 作为主轴引导插补坐标系，先设置手轮引导参数，接着压入一段圆弧和一段直线，启动插补运动，摇动手轮实现插补运动的前进、后退。最后，停止坐标系，退出手轮引导。

```
short rtn;
short core = 1;
//初始化插补坐标系
TCrdPrm crdPrm;
rtn = GTN_GetCrdPrm(core,1,&crdPrm);
crdPrm.dimension=2; // 坐标系为二维坐标系
crdPrm.synVelMax=500; // 最大合成速度：pulse/ms
crdPrm.synAccMax=1; // 最大加速度：pulse/ms^2
crdPrm.evenTime = 50; // 最小匀速时间：ms
crdPrm.profile[0] = 1; // 规划器对应到 X 轴
crdPrm.profile[1] = 2; // 规划器对应到 Y 轴
rtn = GTN_SetCrdPrm(core,1, &crdPrm);//建立坐标系，设置坐标系参数

//使能手轮引导
short crd = 1;
short master = 1;
short mode = 2;
long masterEven = 1;
long slaveEven = 1;
short filterTime = 10;
enable = 1;
rtn = GTN_SetCrdMPGMode(core,crd,enable,master,masterEven,slaveEven,filterTime,mode);
//压入插补数据
```

```
rtn = GTN_LnXY(core,crd,x,y,synVel,synAcc,0,0);
rtn = GTN_ArcXYC(core,crd,20,0,-10,0,0,synVel,synAcc,0,0);

//启动插补运动
rtn = GTN_CrdStart(core,crd,0x1);     //启动坐标系的 FIFO0

//摇动手轮即可实现插补坐标系前进、后退

//退出手轮引导
rtn = GTN_Stop(core,0x3,0x0); //停止规划轴和轴，则会停止插补坐标系
rtn = GTN_SetCrdMPGMode(core,crd,0,master,masterEven,slaveEven,filterTime,mode);
```

第13章 指令详细说明

提示

以下表格中的“章节页码”即为此指令在章节中的位置。可以使用“超级链接”进行索引。“指令示例”即为与此指令相关的例程，可以使用“超级链接”进行索引。

13.1 指令参数范围列表

表 13-1 GXN 产品指令参数范围

参数名称	GTN-024-BB-CC		GTN-016-BB-CC	
内核	[1, 32]		[1, 32]	
内核序号 MC_CORE	(card[1]-1)*2+1	(card-1)*2+2	(card-1)*2+1	(card-1)*2+2
轴 MC_AXIS	[1,12]	[1,12]	[1,8]	[1,8]
插补坐标系序号 MC_CRD	[1, 2]	[1, 2]	[1, 2]	[1, 2]
插补缓存区序号	[0, 1]	[0, 1]	[0, 1]	[0, 1]
网络模块个数 MC_TERMINAL	[1,3]	[1,3]	[1,2]	[1,2]
参数个数[2]	[1,8]	[1,8]	[1,8]	[1,8]
伺服控制器[3] MC_CONTROL	[1,12]	[1,12]	[1,8]	[1,8]
非轴模拟量输出 MC_AU_DAC	[1,6]	[1,6]	[1,6]	[1,6]
非轴模拟量输入 MC_AU_ADC	[1,24]	[1,24]	[1,16]	[1,16]
通用输入 MC_GPI	[1,66]	[1,66]	[1,44]	[1,44]
通用输出 MC_GPO	[1,30]	[1,30]	[1,20]	[1,20]
位置比较输出 MC_POS_COMPARE	[1,8]	[1,8]	[1,8]	[1,8]
辅助编码器（包含手轮） MC_AU_ENCODER	[1,6]	[1,6]	[1,4]	[1,4]
手轮 DI MC_MPG	[1,21]	[1,21]	[1,14]	[1,14]

[1]：card 指主卡个数，例：第 1 张卡对应 card=1，core 为 1，2

[2]：参数个数是指部分读取指令允许一次性读取参数的有效个数，个数设置超过有效个数最大值不会报错，但是读取的值无效

[3]：对于 GTN-016-G-BB 和 GTN-024-G-BB 这两款主卡类型，伺服控制器资源个数为 0

表 13-1 （续）GXN 产品指令参数范围

参数名称		GSN-024-AA-BB		GSN-048-G-01		GSN-008-LT
内核		[1,32]		[1,32]		[1,32]
内核序号 MC_CORE		(card-1)*2+1	(card-1)*2+2	(card-1)*2+1	(card-1)*2+2	(card-1)*2+1
轴 MC_AXIS		[1,12]	[1,12]	[1,24]	[1,24]	[1,8]
插补坐标系序号 MC_CRD	250us	[1, 2]	[1, 2]	[1, 2]	[1, 2]	[1, 2]
	500us	[1,4]	[1,4]	[1,4]	[1,4]	[1,4]
插补缓存区序号		[0, 1]	[0, 1]	[0, 1]	[0, 1]	[0, 1]
网络模块个数[1] MC_TERMINAL		[1, 12]	[1, 12]	[1, 24]	[1, 24]	[1, 8]
参数个数		[1,8]	[1,8]	[1,8]	[1,8]	[1,8]
伺服控制器[2] MC_CONTROL		[1,12]	[1,12]	[0,0]	[0,0]	[1,12]
非轴模拟量输出 MC_AU_DAC		[1,6]	[1,6]	[1,12]	[1,12]	[1,6]
非轴模拟量输入 MC_AU_ADC		[1,24]	[1,24]	[1,48]	[1,48]	[1,24]
通用输入 MC_GPI		[1,100]	[1,100]	[1,100]	[1,100]	[1,100]
通用输出 MC_GPO		[1,64]	[1,64]	[1,64]	[1,64]	[1,64]
位置比较输出 MC_POS_COMPARE		[1,8]	[1,8]	[1,8]	[1,8]	[1,8]
辅助编码器（包含手轮） MC_AU_ENCODER		[1,12]	[1,12]	[1,12]	[1,12]	[1,12]
辅助编码器 MC_AU_ENCODER_EX		[1,8]	[1,8]	[1,8]	[1,8]	[1,8]
手轮 MC_MPG_ENCODER		[1, 4]	[1, 4]	[1, 4]	[1, 4]	[1, 4]
手轮 DI MC_MPG		[1,28]	[1,28]	[1,28]	[1,28]	[1,28]

[1]：网络模块个数指的是数量。而网络模块序号代表排序顺序。GTN 运动控制器采用 GTN_Open(5,1)开卡方式，网络模块序号的取值范围是上表所示。GSN 运动控制器采用 GTN_Open(5,2)开卡方式，网络模块序号的取值范围为[0，单核最大轴数]，例如对于 GSN-024-AA-BB，则为[0,12]。

其中 0 号站代表运动控制器（主站）以及主站上挂接的扩展模块的站号索引。

[2]：对于 GSN-0XX-G-BB 和 GSN-0XX-GT-BB 这两款主卡类型，伺服控制器资源个数为 0

表 13-2 R688C 产品指令参数范围

参数名称	R688C-024-AA-BB
内核	1
内核序号 MC_CORE	1
轴 MC_AXIS	[1,24]
插补坐标系序号 MC_CRD	[1, 2]
插补缓存区序号	[0, 1]
网络模块个数 MC_TERMINAL	[1,24]
参数个数	[1,8]
伺服控制器 MC_CONTROL	[1,24]
非轴模拟量输出 MC_AU_DAC	[1,6]
非轴模拟量输入 MC_AU_ADC	[1,24]
通用输入 MC_GPI	[1,100]
通用输出 MC_GPO	[1,64]
位置比较输出 MC_POS_COMPARE	[1,8]
辅助编码器（包含手轮） MC_AU_ENCODER	[1,9]
辅助编码器 MC_AU_ENCODER_EX	[1,6]
手轮 MC_MPG_ENCODER	[1,3]
手轮 DI MC_MPG	[1,28]

表 13-3 R688S 产品指令参数范围

参数名称	R688S-016-VT-08 (GNS-EtherCAT)	R688S-032-VT-08 (GNS-EtherCAT)
内核	1	1
内核序号 MC_CORE	1	1
轴 MC_AXIS	[1, 16]	[1, 32]
插补坐标系序号 MC_CRD	[1, 4]	[1, 4]
插补缓存区序号	[0, 1]	[0, 1]

参数名称	R688S-016-VT-08 (GNS-EtherCAT)	R688S-032-VT-08 (GNS-EtherCAT)
参数个数	[1, 8]	[1, 8]
伺服控制器 MC_CONTROL	[1, 16]	[1, 32]
非轴模拟量输出 MC_AU_DAC	[1, 12]	[1, 12]
非轴模拟量输入 MC_AU_ADC	[1, 12]	[1, 12]
通用输入 MC_GPI	[1, 64]	[1, 64]
通用输出 MC_GPO	[1, 64]	[1, 64]
辅助编码器（包含手轮） MC_AU_ENCODER	[1, 8]	[1, 8]
辅助编码器 MC_AU_ENCODER_EX	[1, 5]	[1, 5]
手轮 MC_MPG_ENCODER	[1, 3]	[1, 3]
手轮 DI MC_MPG	[1, 21]	[1, 21]
扩展模块数字量输入输出 EXT-DI / EXT-DO	[1, 2048]	[1, 2048]
扩展模块模拟量输入输出 EXT-AI / EXT-AO	[1, 384]	[1, 384]

13.2 指令详细说明

指令 1 GTN_AlarmOff

指令原型	short GTN_AlarmOff(short core, short axis)		
指令说明	控制相应轴驱动报警信号无效。		
指令类型	立即指令，调用后立即生效。	**章节页码**	41
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
axis	控制轴号，正整数，取值范围请参照表13-1中的“轴”一栏		
指令返回值	若返回值为 1：请检查相应轴在配置文件中是否已经配置了报警无效。 其他返回值：请参照指令返回值列表。		
相关指令	GTN_AlarmOn		
指令示例	无		

指令 2 GTN_AlarmOn

指令原型	short GTN_AlarmOn(short core, short axis)
指令说明	控制轴驱动报警信号有效。

指令类型	立即指令，调用后立即生效。	章节页码	41
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
axis	控制轴号，正整数，取值范围请参照表13-1中的“轴”一栏		
指令返回值	若返回值为 1：请检查相应轴在配置文件中是否已经配置了报警无效。 其他返回值：请参照指令返回值列表。		
相关指令	GTN_AlarmOff		
指令示例	无		

指令 3 GTN_ArcXYC

指令原型	short GTN_ArcXYC(short core, short crd, long x, long y, double xCenter, double yCenter, short circleDir, double synVel, double synAcc, double velEnd=0, short fifo=0)		
指令说明	XY 平面圆弧插补。使用圆心描述方法描述圆弧。		
指令类型	缓存区指令。	章节页码	65
指令参数	该指令共有 11 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
crd	坐标系号。正整数，取值范围请参照表 13-1 中的“插补坐标系号”一栏		
x	圆弧插补 x 轴的终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。		
y	圆弧插补 y 轴的终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。		
xCenter	圆弧插补的圆心 x 方向相对于起点位置的偏移量。		
yCenter	圆弧插补的圆心 y 方向相对于起点位置的偏移量。		
circleDir	圆弧的旋转方向。 0：顺时针圆弧。 1：逆时针圆弧。		
synVel	插补段的目标合成速度。取值范围：(0, 65536]，单位：pulse/ms。		
synAcc	插补段的合成加速度。取值范围：(0, 32767]，单位：pulse/ms^2。		
velEnd	插补段的终点速度。取值范围：[0, 32767]，单位：pulse/ms。该值只有在没有使用前瞻预处理功能时才有意义，否则该值无效。默认值为：0。		
fifo	插补缓存区号。默认值为：0，取值范围请参照表 13-1 中的“插补缓存区序号”一栏		
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	**例程 7-6 圆弧插补例程**		

指令 4 GTN_ArcXYR

指令原型	short GTN_ArcXYR (short core, short crd, long x, long y, double radius, short circleDir, double synVel, double synAcc, double velEnd=0, short fifo=0)		
指令说明	XY 平面圆弧插补。以终点位置和半径为输入参数。		
指令类型	缓存区指令。	章节页码	65
指令参数	该指令共有 10 个参数，参数的详细信息如下。		

core	内核，正整数，取值范围请参照表13-1中的“内核”一栏
crd	坐标系号，取值范围请参照表 13-1 中的“插补坐标系号”一栏
x	圆弧插补 x 轴的终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。
y	圆弧插补 y 轴的终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。
radius	圆弧插补的圆弧半径值。取值范围：[-2^32, 2^32]，单位：pulse。 半径为正时，表示圆弧为小于等于 180 °圆弧。 半径为负时，表示圆弧为大于 180 °圆弧。 半径描述方式不能用来描述整圆。
circleDir	圆弧的旋转方向。 0：顺时针圆弧。 1：逆时针圆弧。
synVel	插补段的目标合成速度。取值范围：(0, 65536]，单位：pulse/ms。
synAcc	插补段的合成加速度。取值范围：(0, 32767]，单位：pulse/ms^2。
velEnd	插补段的终点速度。取值范围：[0, 32767]，单位：pulse/ms。该值只有在没有使用前瞻预处理功能时才有意义，否则该值无效。默认值为：0
fifo	插补缓存区号。默认值为：0，取值范围请参照表 13-1 中的“插补缓存区序号”一栏
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。
相关指令	无。
指令示例	**例程 7-6 圆弧插补例程**

指令 5 GTN_ArcYZC

指令原型	short GTN_ArcYZC (short core, short crd, long y, long z, double yCenter, double zCenter, short circleDir, double synVel, double synAcc, double velEnd=0, short fifo=0)		
指令说明	YZ 平面圆弧插补。以终点位置和圆心位置为输入参数。		
指令类型	缓存区指令。	章节页码	65
指令参数	该指令共有 11 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
crd	坐标系号，取值范围请参照表 13-1 中的“插补坐标系号”一栏		
y	圆弧插补 y 轴的终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。		
z	圆弧插补 z 轴的终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。		
yCenter	圆弧插补的圆心 y 方向相对于起点位置的偏移量。		
zCenter	圆弧插补的圆心 z 方向相对于起点位置的偏移量。		
circleDir	圆弧的旋转方向。 0：顺时针圆弧。 1：逆时针圆弧。		
synVel	插补段的目标合成速度。取值范围：(0, 65536]，单位：pulse/ms。		
synAcc	插补段的合成加速度。取值范围：(0, 32767]，单位：pulse/ms^2。		
velEnd	插补段的终点速度。取值范围：[0, 32767]，单位：pulse/ms。该值只有在没有使用前瞻预处理功能时才有意义，否则该值无效。默认值为：0。		

fifo	插补缓存区号。默认值为：0，取值范围请参照表 13-1 中的“插补缓存区序号”一栏
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。
相关指令	无。
指令示例	**例程 7-6 圆弧插补例程**

指令 6 GTN_ArcYZR

指令原型	short GTN_ArcYZR (short core, short crd, long y, long z, double radius, short circleDir, double synVel, double synAcc, double velEnd=0, short fifo=0)		
指令说明	YZ 平面圆弧插补。以终点位置和半径为输入参数。		
指令类型	缓存区指令。	**章节页码**	65
指令参数	该指令共有 10 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
crd	坐标系号，取值范围请参照表 13-1 中的“插补坐标系号”一栏		
y	圆弧插补 y 轴的终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。		
z	圆弧插补 z 轴的终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。		
radius	圆弧插补的圆弧半径值。取值范围：[-2^32,2^32]，单位：pulse。 半径为正时，表示圆弧为小于等于 180°圆弧。 半径为负时，表示圆弧为大于 180°圆弧。 半径描述方式不能用来描述整圆。		
circleDir	圆弧的旋转方向。 0：顺时针圆弧。 1：逆时针圆弧。		
synVel	插补段的目标合成速度。取值范围：(0, 65536]，单位：pulse/ms。		
synAcc	插补段的合成加速度。取值范围：(0, 32767]，单位：pulse/ms^2。		
velEnd	插补段的终点速度。取值范围：[0, 32767]，单位：pulse/ms。该值只有在没有使用前瞻预处理功能时才有意义，否则该值无效。默认值为：0。		
fifo	插补缓存区号。默认值为：0，取值范围请参照表 13-1 中的“插补缓存区序号”一栏		
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	**例程 7-6 圆弧插补例程**		

指令 7 GTN_ArcZXC

指令原型	short GTN_ArcZXC (short core, short crd, long z, long x, double zCenter, double xCenter, short circleDir, double synVel, double synAcc, double velEnd=0, short fifo=0)

指令说明	ZX 平面圆弧插补。以终点位置和圆心位置为输入参数。		
指令类型	缓存区指令。	章节页码	65
指令参数	该指令共有 11 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
crd	坐标系号，取值范围请参照表 13-1 中的“插补坐标系号”一栏		
z	圆弧插补 z 轴的终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。		
x	圆弧插补 x 轴的终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。		
zCenter	圆弧插补的圆心 z 方向相对于起点位置的偏移量。		
xCenter	圆弧插补的圆心 x 方向相对于起点位置的偏移量。		
circleDir	圆弧的旋转方向。 0：顺时针圆弧。 1：逆时针圆弧。		
synVel	插补段的目标合成速度。取值范围：(0, 65536]，单位：pulse/ms。		
synAcc	插补段的合成加速度。取值范围：(0, 32767]，单位：pulse/ms^2。		
velEnd	插补段的终点速度。取值范围：[0, 32767]，单位：pulse/ms。该值只有在没有使用前瞻预处理功能时才有意义，否则该值无效。默认值为：0。		
fifo	插补缓存区号。默认值为：0，取值范围请参照表 13-1 中的“插补缓存区序号”一栏		
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	**例程 7-6 圆弧插补例程**		

指令 8 GTN_ArcZXR

指令原型	short GTN_ArcZXR (short core, short crd, long z, long x, double radius, short circleDir, double synVel, double synAcc, double velEnd=0, short fifo=0)		
指令说明	ZX 平面圆弧插补。以终点位置和半径为输入参数。		
指令类型	缓存区指令。	章节页码	65
指令参数	该指令共有 10 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
crd	坐标系号，取值范围请参照表 13-1 中的“插补坐标系号”一栏		
z	圆弧插补 z 轴的终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。		
x	圆弧插补 x 轴的终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。		
radius	圆弧插补的圆弧半径值。取值范围：[-2^32,2^32]，单位：pulse。 半径为正时，表示圆弧为小于等于 180 °圆弧。 半径为负时，表示圆弧为大于 180 °圆弧。 半径描述方式不能用来描述整圆。		
circleDir	圆弧的旋转方向。 0：顺时针圆弧。 1：逆时针圆弧。		
synVel	插补段的目标合成速度。取值范围：(0, 65536]，单位：pulse/ms。		

synAcc	插补段的合成加速度。取值范围：(0, 32767]，单位：pulse/ms^2。
velEnd	插补段的终点速度。取值范围：[0, 32767]，单位：pulse/ms。该值只有在没有使用前瞻预处理功能时才有意义，否则该值无效。默认值为：0。
fifo	插补缓存区号。默认值为：0，取值范围请参照表 13-1 中的“插补缓存区序号”一栏
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。
相关指令	无。
指令示例	**例程 7-6 圆弧插补例程**

指令 9 GTN_AxisOff

指令原型	short GTN_AxisOff(short core, short axis)		
指令说明	关闭驱动器使能。		
指令类型	立即指令，调用后立即生效。	**章节页码**	146
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
axis	关闭伺服使能的轴的编号，正整数，取值范围请参照表 13-1 中的“轴”一栏		
指令返回值	请参照指令返回值列表。		
相关指令	GTN_AxisOn		
指令示例	**例程 7-1 点位运动**		

指令 10 GTN_AxisOn

指令原型	short GTN_AxisOn(short core, short axis)		
指令说明	打开驱动器使能。		
指令类型	立即指令，调用后立即生效。	**章节页码**	146
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
axis	打开伺服使能的轴的编号，正整数，取值范围请参照表 13-1 中的“轴”一栏		
指令返回值	若返回值为 1： (1) 若在配置文件中报警有效，请检查驱动器是否有报警。 (2) 若当前轴在规划运动，请调用 GTN_Stop 停止运动再调用该指令。 (3) 在闭环控制模式下，采用 MotioStudio 对控制器配置，请检查配置 control 一项中是否选择了关联，若没有，选择关联。若已选择，请检查配置 encoder 一项是否选择激活，若没有，选择激活。若已选择，则请检查 PID 参数中的 Kp 是否设置为 0，Kp 必须大于 0，调试阶段可设为一个较小值 5。 其他返回值：请参照指令返回值列表。		
相关指令	GTN_AxisOff		
指令示例	**例程 7-1 点位运动**		

指令 11 GTN_Bind

指令原型	short GTN_Bind(short core, short thread, short funId, short page)

指令说明	绑定线程、函数、数据页。		
指令类型	立即指令，调用后立即生效。	章节页码	133
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
thread	线程编号，取值范围：[0, 31]。		
funId	函数标识，可以调用 GTN_GetFunId 查询。		
page	数据页编号，取值范围：[0, 31]。		
指令返回值	若返回值为 1：请检查绑定的线程号是否已经有线程绑定并正在运行。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	**例程 11-1　　运动程序单线程累加求和**		

指令 12 GTN_BufDA

指令原型	short GTN_BufDA (short core, short crd, short chn, short daValue, short fifo=0)		
指令说明	缓存区内输出 DA 值。		
指令类型	缓存区指令。	章节页码	65
指令参数	该指令共有 5 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
crd	坐标系号，取值范围请参照表 13-1 中的“插补坐标系号”一栏		
chn	模拟量输出的通道号		
daValue	模拟量输出的值。取值范围：[-32768, 32767]，其中：-32768 对应-10V，32767 对应+10V。		
fifo	插补缓存区号。默认值为：0，取值范围请参照表 13-1 中的“插补缓存区序号”一栏		
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 13 GTN_BufDelay

指令原型	short GTN_BufDelay (short core, short crd, unsigned short delayTime, short fifo=0)		
指令说明	缓存区内延时设置指令。		
指令类型	缓存区指令。	章节页码	65
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
crd	坐标系号，取值范围请参照表 13-1 中的“插补坐标系号”一栏		
delayTime	延时时间。取值范围：[0, 16383]，单位：ms。		
fifo	插补缓存区号。默认值为：0，取值范围请参照表 13-1 中的“插补缓存区序号”一栏		
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。		

	(3) 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。
相关指令	无。
指令示例	**例程 7-5 直线插补例程**

指令 14 GTN_BufDisableDoBitPulse

指令原型	short GTN_BufDisableDoBitPulse(short core,short crd,short doType,short doIndex,short fifo)
指令说明	关闭缓冲区数字量脉冲输出。
指令类型	立即指令。 章节页码
指令参数	该指令共有 2 个参数，参数的详细信息如下。
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏
crd	插补坐标系，取值[1,2]。
doType	输出类型，取值：MC_ENABLE(使能输出口)，MC_CLEAR(清除报警输出口)，MC_GPO(通用输出口)。
doIndex	输出口索引号：取值：使能输出口[1,8]，清除报警输出口[1,8]，通用输出口[1,16]。
fifo	插补缓冲区，取值：[0,1]。
返回值	
相关指令	
指令示例	

指令 15 GTN_BufEnableDoBitPulse

指令原型	short GTN_BufEnableDoBitPulse(short core,short crd,short doType,short doIndex,unsigned short highLevelTime,unsigned short lowLevelTime,long pulseNum,short firstLevel,short fifo)
指令说明	使能缓冲区数字量脉冲输出。
指令类型	立即指令。 章节页码
指令参数	该指令共有 9 个参数，参数的详细信息如下。
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏
crd	插补坐标系，取值[1,2]。
doType	输出类型，取值：MC_ENABLE(使能输出口)，MC_CLEAR(清除报警输出口)，MC_GPO(通用输出口)。
doIndex	输出口索引号：取值：使能输出口[1,8]，清除报警输出口[1,8]，通用输出口[1,16]。
highLevelTime	高电平持续时间:highLevelTime,单位:ms。
lowLevelTime	低电平持续时间:lowLevelTime,单位:ms。
pulseNum	脉冲个数:pulseNum,取值范围:非负数,当为 0 时,表示无限输出脉冲,直到遇到关闭指令。
firstLevel	首先出现的电平状态,firstLevel,0:低电平;1:高电平。
fifo	插补缓冲区，取值：[0,1]。
返回值	
相关指令	
指令示例	

指令 16 GTN_BufGear

指令原型	short GTN_BufGear (short core, short crd, short gearAxis, long pos, short fifo=0)		
指令说明	实现刀向跟随功能，启动某个轴跟随运动。		
指令类型	缓存区指令。	章节页码	65
指令参数	该指令共有 5 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
crd	坐标系号，取值范围请参照表 13-1 中的“插补坐标系号”一栏		
gearAxis	需要进行跟随运动的轴号，该轴不能处于坐标系中。正整数，取值范围请参照表 13-1 中的“轴”一栏		
pos	跟随运动的位移量，单位：pulse。		
fifo	插补缓存区号。默认值为：0，取值范围请参照表 13-1 中的“插补缓存区序号”一栏		
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	**例程 7-9 刀向跟随功能**		

指令 17 GTN_BufIO

指令原型	short GTN_BufIO(short core, short crd, unsigned short doType, unsigned short doMask, unsigned short doValue, short fifo=0)		
指令说明	缓存区内数字量 IO 输出设置指令。		
指令类型	缓存区指令。	章节页码	65
指令参数	该指令共有 6 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
crd	坐标系号，取值范围请参照表 13-1 中的“插补坐标系号”一栏		
doType	数字量输出的类型。 MC_ENABLE(该宏定义为 10)：输出驱动器使能。 MC_CLEAR(该宏定义为 11)：输出驱动器报警清除。 MC_GPO(该宏定义为 12)：输出通用输出。		
doMask	从 bit0~bit15 按位表示指定的数字量输出是否有操作。 0：该路数字量输出无操作。1：该路数字量输出有操作。		
doValue	从 bit0~bit15 按位表示指定的数字量输出的值。		
fifo	插补缓存区号。默认值为：0，取值范围请参照表 13-1 中的“插补缓存区序号”一栏		
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。		
相关指令	无。		

指令示例	例程 7-5 直线插补例程

指令 18 GTN_BufLmtsOff

指令原型	short GTN_BufLmtsOff(short core, short crd, short axis, short limitType, short fifo=0)		
指令说明	缓存区内无效限位开关。		
指令类型	缓存区指令。	章节页码	65
指令参数	该指令共有 5 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
crd	坐标系号，取值范围请参照表 13-1 中的“插补坐标系号”一栏		
axis	需要将限位无效的轴的编号。正整数，取值范围请参照表 13-1 中的“轴”一栏		
limitType	需要无效的限位类型。 MC_LIMIT_POSITIVE(该宏定义为 0)：需要将该轴的正限位无效。 MC_LIMIT_NEGATIVE(该宏定义为 1)：需要将该轴的负限位无效。 -1：需要将该轴的正限位和负限位都无效，默认为该值。		
fifo	插补缓存区号。默认值为：0，取值范围请参照表 13-1 中的“插补缓存区序号”一栏		
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。		
相关指令	GTN_BufLmtsOn		
指令示例	无。		

指令 19 GTN_BufLmtsOn

指令原型	short GTN_BufLmtsOn(short core, short crd, short axis, short limitType, short fifo=0)		
指令说明	缓存区内有效限位开关。		
指令类型	缓存区指令。	章节页码	65
指令参数	该指令共有 5 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
crd	坐标系号，取值范围请参照表 13-1 中的“插补坐标系号”一栏		
axis	需要将限位有效的轴的编号。正整数，取值范围请参照表 13-1 中的“轴”一栏		
limitType	需要有效的限位类型。 MC_LIMIT_POSITIVE(该宏定义为 0)：需要将该轴的正限位设置为有效。 MC_LIMIT_NEGATIVE(该宏定义为 1)：需要将该轴的负限位设置为有效。 -1：需要将该轴的正限位和负限位都设置为有效，默认为该值。		
fifo	插补缓存区号。默认值为：0，取值范围请参照表 13-1 中的“插补缓存区序号”一栏		
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。		
相关指令	GTN_BufLmtsOff		

指令示例	无。

指令 20 GTN_BufMove

指令原型	short GTN_BufMove (short core, short crd, short moveAxis, long pos, double vel, double acc, short modal, short fifo=0)		
指令说明	实现刀向跟随功能，启动某个轴点位运动。		
指令类型	缓存区指令。	章节页码	65
指令参数	该指令共有 8 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
crd	坐标系号，取值范围请参照表 13-1 中的“插补坐标系号”一栏		
moveAxis	需要进行点位运动的轴号，该轴不能处于坐标系中。正整数，取值范围请参照表 13-1 中的“轴”一栏		
pos	点位运动的目标位置，单位：pulse。		
vel	点位运动的目标速度，单位：pulse/ms。		
acc	点位运动的加速度，单位：pulse/ms^2。		
modal	点位运动的模式。 0：该指令为非模态指令，即不阻塞后续的插补缓存区指令的执行。 1：该指令为模态指令，将会阻塞后续的插补缓存区指令的执行。		
fifo	插补缓存区号。默认值为：0，取值范围请参照表 13-1 中的“插补缓存区序号”一栏		
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	**例程 7-8 刀向跟随功能**		

指令 21 GTN_BufSetStopIo

指令原型	short GTN_BufSetStopIo (short core, short crd, short axis, short stopType, short inputType, short inputIndex, short fifo=0)		
指令说明	缓存区内设置 axis 的停止 IO 信息。		
指令类型	缓存区指令。	章节页码	65
指令参数	该指令共有 7 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
crd	坐标系号，取值范围请参照表 13-1 中的“插补坐标系号”一栏		
axis	需要设置停止 IO 信息的轴的编号。正整数，取值范围请参照表 13-1 中的“轴”一栏		
stopType	需要设置停止 IO 信息的停止类型。 0：紧急停止类型。 1：平滑停止类型。		
inputType	设置的数字量输入的类型。 MC_LIMIT_POSITIVE(该宏定义为 0)：正限位。 MC_LIMIT_NEGATIVE(该宏定义为 1)：负限位。 MC_ALARM(该宏定义为 2)：驱动报警。		

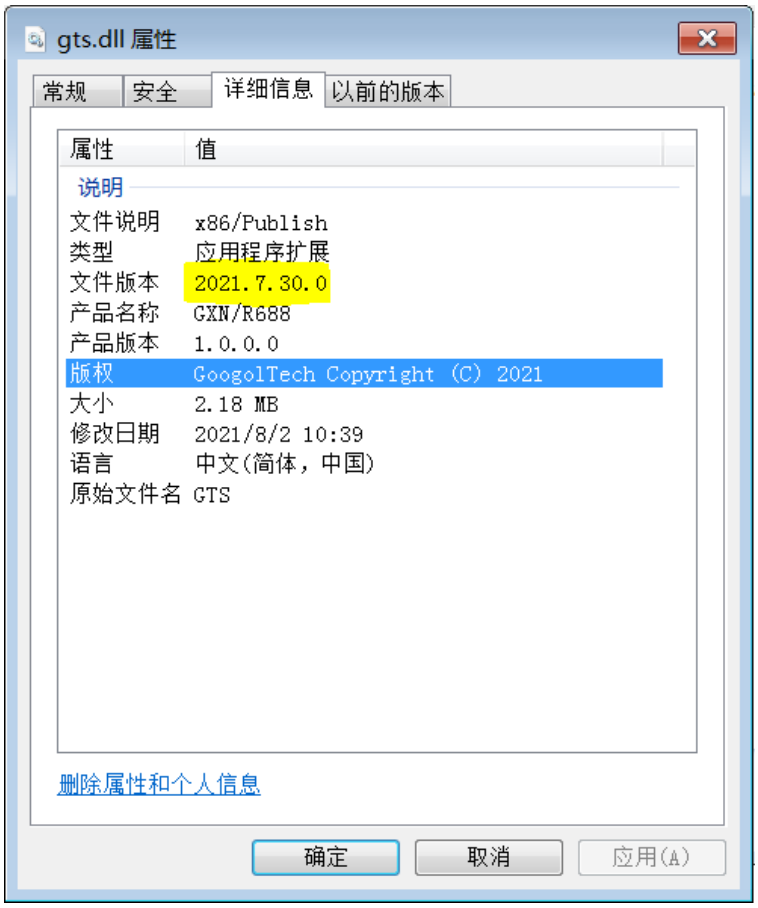
	MC_HOME(该宏定义为 3)：原点开关。 MC_GPI(该宏定义为 4)：通用输入。 MC_ARRIVE(该宏定义为 5)：电机到位信号。
inputIndex	设置的数字量输入的索引号，取值范围根据 inputType 的取值而定。请参考 inputType 对应的输入取值范围
fifo	插补缓存区号。默认值为：0，取值范围请参照表 13-1 中的“插补缓存区序号”一栏
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。
相关指令	无。
指令示例	无。

指令 22 GTN_ClearTriggerStatus

指令原型	short GTN_ClearTriggerStatus(short core,short index)		
指令说明	清除捕获状态。		
指令类型	立即指令，调用后立即生效。	**章节页码**	104
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
index	捕获序号，正整数，取值范围请参照表 13-1 中的“轴”一栏		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 23 GTN_Close

指令原型	short GTN_Close()		
指令说明	关闭运动控制器。		
指令类型	立即指令，调用后立即生效。	**章节页码**	144
指令参数	无。		
指令返回值	请参照指令返回值列表。		
相关指令	GTN_Open		
指令示例	**用户读取动态链接库的版本，点击 gts.dll，右键->**属性->详细信息处可以查看动态链接库的版本。		



文件说明	动态链接库的说明，x86/Publish 表示是 32 位的发布版本的动态链接库。
文件版本	动态链接库的版本日期，如 2021.7.30.0，表示该动态链接库发布于：2021 年 7 月 30 日

例程 12-1 读取运动控制器版本号

指令 24 GTN_ClrSts

指令原型	short GTN_ClrSts(short core, short axis, short count=1)		
指令说明	清除驱动器报警标志、跟随误差越限标志、限位触发标志。 1. 只有当驱动器没有报警时才能清除轴状态字的报警标志； 2. 只有当跟随误差正常以后，才能清除跟随误差越限标志； 3. 只有当离开限位开关，或者规划位置在软限位行程以内时才能清除轴状态字的限位触发标志。		
指令类型	立即指令，调用后立即生效。	章节页码	47
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
axis	轴序号，正整数，取值范围请参照表 13-1 中的“轴”一栏		
count	轴数，默认为 1，正整数，即从轴序号 axis 开始需要进行操作的轴数量，取值范围请参照表 13-1 中的“参数个数”一栏		
指令返回值	请参照指令返回值列表。		
相关指令	GTN_GetSts		
指令示例	**例程 5-1 修改编码器计数方向**		

指令 25 GTN_CrdClear

指令原型	short GTN_CrdClear(short core, short crd, short fifo)		
指令说明	清除插补缓存区内的插补数据。		
指令类型	立即指令，调用后立即生效。	章节页码	65
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
crd	坐标系号，取值范围请参照表 13-1 中的“插补坐标系号”一栏		
fifo	插补缓存区号。默认值为：0，取值范围请参照表 13-1 中的“插补缓存区序号”一栏		
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	**例程 7-5 直线插补例程**		

指令 26 GTN_CrdData

指令原型	short GTN_CrdData (short core, short crd, TCrdData *pCrdData, short fifo=0)		
指令说明	用于在使用前瞻时。调用该指令表示后续没有新的数据，将会一次性把前瞻缓存区的数据压入运动缓存区。		
指令类型	立即指令，调用后立即生效。	章节页码	65
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
crd	坐标系号，取值范围请参照表 13-1 中的“插补坐标系号”一栏		
pCrdData	只能设置为：NULL。		
fifo	插补缓存区号。默认值为：0，取值范围请参照表 13-1 中的“插补缓存区序号”一栏		
指令返回值	若返回值为非零值，说明前瞻缓存区还有数据没有被压入运动缓存区，而运动缓存区没有空间了。此时需要检查运动缓存区的空间（调用 GTN_CrdSpace 检查）。当检查运动缓存区有空间时，再次调用 GTN_CrdData 指令，直至返回值为 0 时，前瞻缓存区的数据才被完全送入运动缓存区。		
相关指令	无。		
指令示例	**例程 7-7 前瞻预处理例程**		

指令 27 GTN_CrdHsOff

指令原型	short GTN_CrdHsOff(short core,short crd,short fifo)		
指令说明	关闭 DMA 传输通道		
指令类型	立即指令，调用后立即生效。	章节页码	65
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
crd	坐标系号，取值范围请参照表 13-1 中的“插补坐标系号”一栏		
fifo	插补缓存区号。默认值为：0，取值范围请参照表 13-1 中的“插补缓存区序号”一栏		
指令返回值	若返回值为 1：请停止各轴规划运动后再设置。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	无		

指令 28 GTN_CrdHsOn

指令原型	short GTN_CrdHsOn(short core,short crd,short fifo,short link=1,unsigned short threshold=200, short lookAheadInMc=0)		
指令说明	开启 DMA 传输通道		
指令类型	立即指令，调用后立即生效。	章节页码	65
指令参数	该指令共有 6 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
crd	坐标系号，取值范围请参照表 13-1 中的“插补坐标系号”一栏		
fifo	插补缓存区号。默认值为：0，取值范围请参照表 13-1 中的“插补缓存区序号”一栏		
link	默认取1，取值范围[1,4]，如果同时使用振镜DMA，则振镜DMA指令对应的link应该为不同于插补DMA指令的link		
threshold	阈值， PC机指令达到该阈值时自动启动DMA发送数据，正整数，取值范围[1,4096]。		
lookAheadInMc	前瞻下移功能标志位，默认取0		
指令返回值	若返回值为 1：请停止各轴规划运动后再设置。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	无		

指令 29 GTN_CrdSpace

指令原型	short GTN_CrdSpace (short core, short crd, long *pSpace, short fifo=0)		
指令说明	查询插补缓存区剩余空间。		
指令类型	立即指令，调用后立即生效。	章节页码	65
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
crd	坐标系号，取值范围请参照表 13-1 中的“插补坐标系号”一栏		
pSpace	读取插补缓存区中的剩余空间。		
fifo	插补缓存区号。默认值为：0，取值范围请参照表 13-1 中的“插补缓存区序号”一栏		
指令返回值	若返回值为 1：检查当前坐标系是否映射了相关轴。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	**例程 7-7 前瞻预处理例程**		

指令 30 GTN_CrdStart

指令原型	short GTN_CrdStart (short core, short mask, short option)		
指令说明	启动插补运动。		
指令类型	立即指令，调用后立即生效。	章节页码	65
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
mask	从 bit0~bit1 按位表示需要启动的坐标系。 bit0 对应坐标系 1，bit1 对应坐标系 2。 0：不启动该坐标系，1：启动该坐标系。		
option	从 bit0~bit1 按位表示坐标系需要启动的缓存区的编号。		

	bit0 对应坐标系 1，bit1 对应坐标系 2。 0：启动坐标系中 FIFO0 的运动，1：启动坐标系中 FIFO1 的运动。
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 若使用了辅助 fifo1 运动，检查当前坐标系位置没有恢复到 fifo0 断点坐标系位置。 (3) 检查参数设置是否启动了坐标系。 (4) 检查坐标系是否在运动。 其他返回值：请参照指令返回值列表。
相关指令	无。
指令示例	**例程 7-5 直线插补例程**

指令 31 GTN_CrdStatus

指令原型	short GTN_CrdStatus (short core, short crd, short *pRun, long *pSegment, short fifo=0)		
指令说明	查询插补运动坐标系状态。		
指令类型	立即指令，调用后立即生效。	章节页码	65
指令参数	该指令共有 5 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
crd	坐标系号，取值范围请参照表 13-1 中的“插补坐标系号”一栏		
pRun	读取插补运动状态。0：该坐标系的该 FIFO 没有在运动；1：该坐标系的该 FIFO 正在进行插补运动。		
pSegment	读取当前已经完成的插补段数。当重新建立坐标系或者调用 GTN_CrdClear 指令后，该值会被清零。		
fifo	所要查询运动状态的插补缓存区号。默认值为：0，取值范围请参照表 13-1 中的“插补缓存区序号”一栏		
指令返回值	若返回值为 1：检查当前坐标系是否映射了相关轴。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	**例程 7-5 直线插补例程**		

指令 32 GTN_CtrlMode

指令原型	short GTN_CtrlMode(short core, short axis, short mode)		
指令说明	设置控制轴为模拟量输出或脉冲输出。		
指令类型	立即指令，调用后立即生效。	章节页码	41
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
axis	坐标系号，取值范围请参照表 13-1 中的“轴”一栏		
mode	切换的模式。 0：将指定轴切换为闭环控制模式(即电压控制方式)。 1：将指定轴切换为开环控制模式(即脉冲控制方式)。 非零值均表示 mode1 开环控制模式。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	**例程 5-3 设置第 1 轴为闭环控制方式**		

指令 33 GTN_DisableDoBitPulse

指令原型	short GTN_DisableDoBitPulse(short core,short doType,short doIndex)		
指令说明	关闭数字量脉冲输出。		
指令类型	立即指令。	章节页码	
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
doType	输出类型，取值：MC_ENABLE(使能输出口)，MC_CLEAR(清除报警输出口)，MC_GPO(通用输出口)。		
doIndex	输出口索引号：取值：使能输出口[1,8]，清除报警输出口[1,8]，通用输出口[1,16]。		
返回值			
相关指令			
指令示例	rtn = GTN_DisableDoBitPulse(core,MC_GPO,1); //关闭 GPO1 输出脉冲		

指令 34 GTN_Download

指令原型	short GTN_Download(short core, char *pFileName)		
指令说明	下载运动程序到运动控制器。注：文件包含的线程数不能大于 32。		
指令类型	立即指令，调用后立即生效。	章节页码	133
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
pFileName	下载到运动控制器的运动程序文件名		
指令返回值	若返回值为 2007： (1) 请检查文件名是否太长。 (2) 请检查需要下载的文件是否存在。 若返回值为 2008：表示打开文件失败。 (1) 请检查文件是否损坏，编译是否成功。 (2) 请检查 ini 和 bin 文件是否在同一根目录。 若返回值为 7：请检查线程数量是否大于 32。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	**例程 11-1 运动程序单线程累加求和**		

指令 35 GTN_EnableDoBitPulse

指令原型	short GTN_EnableDoBitPulse(short croe,short doType,short doIndex,unsigned short highLevelTime,unsigned short lowLevelTime,long pulseNum,short firstLevel)		
指令说明	使能数字量脉冲输出。		
指令类型	立即指令。	章节页码	
指令参数	该指令共有 7 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
doType	输出类型，取值：MC_ENABLE(使能输出口)，MC_CLEAR(清除报警输出口)，MC_GPO(通用输出口)。		
doIndex	输出口索引号：取值：使能输出口[1,8]，清除报警输出口[1,8]，通用输出口[1,16]。		
highLevelTime	高电平持续时间:highLevelTime,单位:ms。		
lowLevelTime	低电平持续时间:lowLevelTime,单位:ms。		

pulseNum	脉冲个数:pulseNum,取值范围:非负数,当为 0 时,表示无限输出脉冲,直到遇到关闭指令。
firstLevel	首先出现的电平状态,firstLevel,0:低电平;1:高电平。
返回值	
相关指令	
指令示例	rtn = GTN_EnableDoBitPulse(core,MC_GPO,1,1000,2000,10,0); //GPO1 输出高电平持续 1000ms 低电平持续 2000ms 的脉冲。

指令 36 GTN_EnableLeadScrewComp

指令原型	short GTN_EnableLeadScrewComp(short core,short axis,short mode)		
指令说明	是否开启补偿		
指令类型	立即指令，调用后立即生效。	**章节页码**	153
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
axis	补偿轴轴号，正整数，取值范围请参照表 13-1 中的“轴”一栏		
mode	0-关闭；1-开启		
指令返回值	请参照指令返回值及其意义。		
相关指令	无。		
指令示例	**例程 12-3 螺距误差补偿例程**		

指令 37 GTN_EncOff

指令原型	short GTN_EncOff(short core, short encoder)		
指令说明	设置为“脉冲计数器”计数方式。		
指令类型	立即指令，调用后立即生效。	**章节页码**	41
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
encoder	编码器通道号，正整数，取值范围请参照表 13-1 中的“轴”一栏		
指令返回值	请参照指令返回值列表		
相关指令	GTN_EncOn		
指令示例	无。		

指令 38 GTN_EncOn

指令原型	short GTN_EncOn(short core, short encoder)		
指令说明	设置为“外部编码器”计数方式。		
指令类型	立即指令，调用后立即生效。	**章节页码**	41
指令参数	该指令共有 2 个参数，参数的详细信息如下：		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
encoder	编码器通道号，正整数，取值范围请参照表 13-1 中的“轴”一栏		
指令返回值	请参照指令返回值列表		
相关指令	GTN_EncOff		
指令示例	无。		

指令 39 GTN_EncScale

指令原型	short GTN_EncScale(short core, short axis, short alpha, short beta)		
指令说明	设置控制轴的编码器当量变换值。		
指令类型	立即指令，调用后立即生效。	章节页码	41
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
axis	控制器轴号，正整数，取值范围请参照表 13-1 中的“轴”一栏		
alpha	规划器当量的alpha值，取值范围：[-32767, 0)和(0, 32767]。		
beta	规划器当量的beta值，取值范围：[-32767, 0)和(0, 32767]。		
指令返回值	若返回值为 1：若当前轴在规划运动，请调用 GTN_Stop 停止运动再调用该指令。 其他返回值：请参照指令返回值列表。		
相关指令	GTN_ProfileScale		
指令示例	无。		

指令 40 GTN_GearStart

指令原型	short GTN_GearStart(short core, long mask)
指令说明	启动电子齿轮运动。
指令类型	立即指令，调用后立即生效。 章节页码 61
指令参数	该指令共有 2 个参数，参数的详细信息如下。
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏
mask	按位指示需要启动 Gear 运动的轴号。当 bit 位为 1 时表示启动对应的轴。
指令返回值	若返回值为 1： (1) 请检查当前轴是否为电子齿轮模式，若不是，请先调用 GTN_PrfGear 将当前轴设置为电子齿轮模式。 (2) 请检查主轴是否已设置。 (3) 请检查传动比是否已设置。 其他返回值：请参照指令返回值列表。
相关指令	无。
指令示例	**例程 7-3 电子齿轮跟随**

Bit	11	10	9	8	7	6	5	4	3	2	1	0
对应轴	12 轴	11 轴	10 轴	9 轴	8 轴	7 轴	6 轴	5 轴	4 轴	3 轴	2 轴	1 轴
Bit	31	……										12
对应轴	32 轴	……										13 轴

指令 41 GTN_GetAdc

指令原型	short GTN_GetAdc (short core, short adc, double *pValue, short count=1, unsigned long *pClock=NULL)
指令说明	读取模拟量输入的电压值。
指令类型	立即指令，调用后立即生效。 章节页码 101
指令参数	该指令共有 5 个参数，参数的详细信息如下。
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏
adc	模拟量输入起始通道号，正整数，取值范围请参照表 13-1 中的“轴”一栏
pValue	读取的输入电压值。单位：伏特。

count	读取的通道数，默认为 1，正整数，取值范围请参照表 13-1 中的“参数个数”一栏
pClock	读取控制器时钟，默认值为：NULL，即不用读取控制器时钟。
指令返回值	请参照指令返回值列表。
相关指令	GTN_GetAdcValue
指令示例	**例程 8-4 访问 ADC**

指令 42 GTN_GetAdcFilterPrm

指令原型	short GTN_GetAdcFilterPrm(short core,short adc,double *pk)		
指令说明	读取模拟量输入的滤波参数。		
指令类型	立即指令，调用后立即生效。	章节页码	99
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
adc	位置比较模块编号，正整数，取值范围请参照表 13-1 中的“轴”一栏		
pK	滤波系数，取值范围[0,1)。越接近 1 效果越强，但滞后越大。		
指令返回值	请参照指令返回值列表。		
相关指令			
指令示例			

指令 43 GTN_GetAdcValue

指令原型	short GTN_GetAdcValue (short core, short adc, short *pValue, short count=1, unsigned long *pClock=NULL)		
指令说明	读取模拟量输入的数字转换值。		
指令类型	立即指令，调用后立即生效。	章节页码	101
指令参数	该指令共有 5 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
adc	模拟量输入起始通道号，正整数，取值范围请参照表 13-1 中的“轴”一栏		
pValue	读取的输入电压数值。单位：bit，取值范围：[-32768, 32767]，对应的电压值为[-10, 10]伏特。		
count	读取的通道数，默认为 1，正整数，取值范围请参照表 13-1 中的“参数个数”一栏		
pClock	读取控制器时钟，默认值为：NULL，即不用读取控制器时钟。		
指令返回值	请参照指令返回值列表。		
相关指令	GTN_GetAdc		
指令示例	**例程 8-4 访问 ADC**		

指令 44 GTN_GetAuAdc

指令原型	short GTN_GetAuAdc (short core, short adc, double *pValue, short count=1, unsigned long *pClock=NULL)		
指令说明	读取非轴模拟量输入的电压值。		
指令类型	立即指令，调用后立即生效。	章节页码	101
指令参数	该指令共有 5 个参数，参数的详细信息如下。		
core	读取的通道数，正整数，取值范围请参照表 13-1 中的“内核”一栏		
adc	auadc 起始通道号，正整数，取值范围请参照表 13-1 中的“非轴模拟量输入”一栏		
pValue	读取非轴通道的输入电压值。单位：伏特。		

count	读取的通道数，默认为 1。 正整数，取值范围请参照表 13-1 中的“参数个数”一栏。
pClock	读取控制器时钟，默认值为：NULL，即不用读取控制器时钟。
指令返回值	请参照指令返回值列表。
相关指令	无
指令示例	无

指令 45 GTN_GetAuAdcValue

指令原型	short GTN_GetAuAdcValue (short core, short adc, short *pValue, short count=1, unsigned long *pClock=NULL)		
指令说明	读取非轴模拟量输入的数字转换值。		
指令类型	立即指令，调用后立即生效。	章节页码	101
指令参数	该指令共有 5 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
adc	adc 起始通道号，正整数，取值范围请参照表 13-1 中的“非轴模拟量输入”一栏		
pValue	读取的输入电压数值。单位：bit，取值范围：[-32768, 32767]，对应的电压值为[-10, 10]伏特。		
count	读取的通道数。默认为 1。 正整数，取值范围请参照表 13-1 中的“参数个数”一栏。		
pClock	读取控制器时钟，默认值为：NULL，即不用读取控制器时钟。		
指令返回值	请参照指令返回值列表。		
相关指令	无		
指令示例	无		

指令 46 GTN_GetAuDac

指令原型	short GTN_GetAuDac(short core,short dac,short *pValue,short count,unsigned long *pClock=NULL)		
指令说明	读取非轴模拟量输出的数字转换值。		
指令类型	立即指令，调用后立即生效。	章节页码	100
指令参数	该指令共有 5 个参数，参数的详细信息如下。		
core	读取的通道数，正整数，取值范围请参照表 13-1 中的“内核”一栏		
dac	AUDAC 的起始通道号，正整数，取值范围请参照表 13-1 中的“非轴模拟量输出”一栏		
pValue	读取的非轴模拟量输出的电压数值。单位：bit，取值范围：[0, 32767]，对应的电压值为[0, 10]伏特。		
count	读取的通道数，默认为 1。 正整数，取值范围请参照表 13-1 中的“参数个数”一栏。		
pClock	读取控制器时钟，默认值为：NULL，即不用读取控制器时钟。		
指令返回值	请参照指令返回值列表。		
相关指令	无		

指令 47 GTN_GetAxisBand

指令原型	short GTN_GetAxisBand(short core, short axis, long *pBand, long *pTime)		
指令说明	读取轴到位误差带。		
指令类型	立即指令，调用后立即生效。	章节页码	147

指令参数	该指令共有 4 个参数，参数的详细信息如下。
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏
axis	轴号。正整数，取值范围请参照表 13-1 中的“轴”一栏
pBand	读取误差带大小。
pTime	读取误差带保持时间。
指令返回值	请参照指令返回值列表。
相关指令	GTN_SetAxisBand
指令示例	无。

指令 48 GTN_GetAxisEncAcc

指令原型	short GTN_GetAxisEncAcc(short core, short axis, double *pValue, short count=1, unsigned long *pClock=NULL)		
指令说明	读取 encoder 输出值经过当量变换之后的编码器加速度值。		
指令类型	立即指令，调用后立即生效。	章节页码	47
指令参数	该指令共有 5 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
axis	起始轴号。正整数，取值范围请参照表 13-1 中的“轴”一栏		
pValue	轴的编码器加速度。单位：pulse/ms^2。		
count	读取的轴数，默认为 1。 正整数，取值范围请参照表 13-1 中的“参数个数”一栏		
pClock	读取控制器时钟，默认值为：NULL，即不用读取控制器时钟。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 49 GTN_GetAxisEncPos

指令原型	short GTN_GetAxisEncPos(short core, short axis, double *pValue, short count=1, unsigned long *pClock=NULL)		
指令说明	读取 encoder 输出值经过当量变换之后的编码器位置值。		
指令类型	立即指令，调用后立即生效。	章节页码	47
指令参数	该指令共有 5 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
axis	起始轴号。正整数，取值范围请参照表 13-1 中的“轴”一栏		
pValue	轴的编码器位置。单位：pulse。		
count	读取的轴数，默认为 1，正整数，取值范围请参照表 13-1 中的“参数个数”一栏		
pClock	读取控制器时钟，默认值为：NULL，即不用读取控制器时钟。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 50 GTN_GetAxisEncVel

指令原型	short GTN_GetAxisEncVel(short core, short axis, double *pValue, short count=1, unsigned long *pClock=NULL)

指令说明	读取 encoder 输出值经过当量变换之后的编码器速度值。		
指令类型	立即指令，调用后立即生效。	章节页码	47
指令参数	该指令共有 5 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
axis	起始轴号。正整数，取值范围请参照表 13-1 中的“轴”一栏		
pValue	轴的编码器速度。单位：pulse/ms。		
count	读取的轴数，默认为 1 正整数，取值范围请参照表 13-1 中的“参数个数”一栏		
pClock	读取控制器时钟，默认值为：NULL，即不用读取控制器时钟。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 51 GTN_GetAxisError

指令原型	short GTN_GetAxisError(short core, short axis, double *pValue, short count=1, unsigned long *pClock=NULL)		
指令说明	读取 profile 经过当量变换之后的规划位置与 encoder 经过当量变换之后的编码器位置的差值。		
指令类型	立即指令，调用后立即生效。	章节页码	47
指令参数	该指令共有 5 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
axis	起始轴号。正整数，取值范围请参照表 13-1 中的“轴”一栏		
pValue	轴的规划位置与编码器位置的差值。单位：pulse。		
count	读取的轴数，默认为 1。 正整数，取值范围请参照表 13-1 中的“参数个数”一栏		
pClock	读取控制器时钟，默认值为：NULL，即不用读取控制器时钟。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 52 GTN_GetAxisPrfAcc

指令原型	short GTN_GetAxisPrfAcc(short core, short axis, double *pValue, short count=1, unsigned long *pClock=NULL)		
指令说明	读取 profile 输出值经过当量变换之后的规划加速度。		
指令类型	立即指令，调用后立即生效。	章节页码	47
指令参数	该指令共有 5 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
axis	起始轴号。正整数，取值范围请参照表 13-1 中的“轴”一栏		
pValue	轴的规划加速度。单位：pulse/ms^2。		
count	读取的轴数，默认为 1。正整数，取值范围请参照表 13-1 中的“参数个数”一栏		
pClock	读取控制器时钟，默认值为：NULL，即不用读取控制器时钟。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 53 GTN_GetAxisPrfPos

指令原型	short GTN_GetAxisPrfPos(short core, short axis, double *pValue, short count=1, unsigned long *pClock=NULL)		
指令说明	读取 profile 输出值经过当量变换之后的规划位置。		
指令类型	立即指令，调用后立即生效。	章节页码	47
指令参数	该指令共有 5 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
axis	起始轴号。正整数，取值范围请参照表 13-1 中的“轴”一栏		
pValue	轴的规划位置。单位：pulse。		
count	读取的轴数，默认为 1。正整数，取值范围请参照表 13-1 中的“参数个数”一栏		
pClock	读取控制器时钟，默认值为：NULL，即不用读取控制器时钟。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	无		

指令 54 GTN_GetAxisPrfVel

指令原型	short GTN_GetAxisPrfVel(short core, short axis, double *pValue, short count=1, unsigned long *pClock=NULL)		
指令说明	读取 profile 输出值经过当量变换之后的规划速度。		
指令类型	立即指令，调用后立即生效。	章节页码	47
指令参数	该指令共有 5 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
axis	起始轴号。正整数，取值范围请参照表 13-1 中的“轴”一栏		
pValue	轴的规划速度。单位：pulse/ms。		
count	读取的轴数，默认为 1。正整数，取值范围请参照表 13-1 中的“参数个数”一栏		
pClock	读取控制器时钟，默认值为：NULL，即不用读取控制器时钟。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 55 GTN_GetBacklash

指令原型	short GTN_GetBacklash (short core, short axis, long *pCompValue, double *pCompChangeValue, long *pCompDir)		
指令说明	读取反向间隙补偿的相关参数。		
指令类型	立即指令，调用后立即生效。	章节页码	147
指令参数	该指令共有 5 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
axis	查询的轴号。正整数，取值范围请参照表 13-1 中的“轴”一栏		
pCompValue	读取的反向间隙补偿值。		
pCompChange Value	读取的反向间隙补偿值的变化量。		
pCompDir	读取的反向间隙补偿的补偿方向。		
指令返回值	请参照指令返回值列表。		

相关指令	GTN_SetBacklash
指令示例	无。

指令 56 GTN_GetClock

指令原型	short GTN_GetClock(short core, unsigned long *pClock, unsigned long *pLoop=NULL)		
指令说明	读取运动控制器系统时钟。		
指令类型	立即指令，调用后立即生效。	章节页码	145
指令参数	该指令共有 3 个参数，参数的详细信息如下：		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
pClock	读取的运动控制器的时钟，单位：ms		
pLoop	内部使用，默认值为：NULL，即不读取该值		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 57 GTN_GetClockHighPrecision

指令原型	shortGTN_GetClockHighPrecision(short core, unsigned long *pClock)		
指令说明	读取运动控制器系统高精度时钟。		
指令类型	立即指令，调用后立即生效。	章节页码	145
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
pClock	读取的运动控制器的时钟，单位：中断周期。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 58 GTN_GetCompensate2D

指令原型	short GTN_GetCompensate2D(short core, short axis,TCompensate2D *pComp2d)		
指令说明	读取二维补偿参数。		
指令类型	立即指令，调用后立即生效。	章节页码	155
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
axis	需要补偿的轴号,正整数，取值范围请参照表 13-1 中的“轴”一栏		
pComp2d	设置二维补偿参数，该参数为一结构体，详细参数定义及说明请参照指令GTN_SetCompensate2D中参数3		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	**例程 12-4　　二维位置补偿例程**		

指令 59 GTN_GetCompensate2DTable

指令原型	short GTN_GetCompensate2DTable(short core,short tableIndex, TCompensate2DTable *pTable,short *pExternComp=NULL)
指令说明	获取二维补偿表参数。

指令类型	立即指令，调用后立即生效。	章节页码	154
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
tableIndex	控制器内部的补偿表索引号，取值范围：[1,4]。		
pTable	获取二维补偿表及数据的参数，该参数为一结构体，详细参数定义及说明请参照指令GTN_SetCompensate2DTable的参数3。		
pExternComp	获取是否自动扩展了补偿区域。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	**例程 12-4　　二维位置补偿例程**		

指令 60 GTN_GetCompensate2DtableRotationAngle

指令原型	short GTN_GetCompensate2DTableRotationAngle(short core,short tableIndex, short enable,double rotationAngle)		
指令说明	获取二维补偿表旋转参数。		
指令类型	立即指令，调用后立即生效。	章节页码	154
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
tableIndex	控制器内部的补偿表索引号，取值范围：[1,4]。		
pEnable	补偿表旋转功能使能，0：不使能，1：使能。		
pRotationAngle	补偿表以表起点位置相对于x坐标轴逆时针旋转角度，单位：度。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	**无。**		

指令 61 GTN_GetCompensate2DValue

指令原型	short GTN_GetCompensate2DValue(short core,short axis,double *pValue)		
指令说明	获取补偿值。		
指令类型	立即指令，调用后立即生效。	章节页码	154
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
axis	需要补偿的轴号,正整数，取值范围请参照表 13-1 中的“轴”一栏		
pValue	补偿值		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	**例程 12-4　　二维位置补偿例程**		

指令 62 GTN_GetControlFilter

指令原型	short GTN_GetControlFilter(short core, short control, short *pIndex)		
指令说明	读取当前 PID 索引。		
指令类型	立即指令，调用后立即生效。	章节页码	146
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		

control	伺服控制器编号，正整数，取值范围请参照表 13-1 中的“伺服控制器”一栏
pIndex	读取的伺服控制参数的索引号。
指令返回值	请参照指令返回值列表。
相关指令	GTN_SetControlFilter
指令示例	无。

指令 63 GTN_GetCrdHsPrm

指令原型	short GTN_GetCrdHsPrm(short core,short crd,short fifo,short *pEnable,short *pDmaBuf,unsigned short *pThreshold,short *pLookAheadInMc)		
指令说明	设置各个资源的配置信息		
指令类型	立即指令	章节页码	65
指令参数	该指令共有 7 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
crd	插补坐标系，正整数，取值范围请参照表13-1中的“插补坐标系序号”一栏		
fifo	插补缓存区，正整数，取值范围请参照表13-1中的“插补缓存区序号”一栏		
pEnable	使能标志位		
pDmaBuf	数据缓存区序号，正整数，取值范围请参照表13-1中的“DMA数据缓存区号”一栏		
pThreshold	数据缓存区大小，正整数，取值范围请参照表13-1中的“DMA指令数”一栏		
lookAheadInMc	前瞻下移功能标志位		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	无		
应用说明	无		

指令 64 GTN_GetCrdMapBase

指令原型	short GTN_GetCrdMapBase(short core,short crd,short *pBase)		
指令说明	获取插补坐标系映射基础规划轴。		
指令类型	立即指令，调用后立即生效。	章节页码	99
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
crd	坐标系编号，正整数，取值范围请参照表 13-1 中的“坐标系”一栏		
*pBase	基础规划轴，正整数，取值范围请参照表 13-1 中的“轴”一栏。 控制器默认为 1，即建立插补坐标系只支持前 8 个规划器。		
指令返回值	请参照指令返回值列表。		
相关指令			
指令示例			

指令 65 GTN_GetCrdMPGMode

指令原型	short GTN_GetCrdMPGMode(short core,short crd,short *pEnable,short *pMaster,long *pMasterEven,long *pSlaveEven,short *pFilterTime,short *pMode,short *pFifoEnd)		
指令说明	获取插补模式下的手轮引导功能参数		
指令类型	立即指令，调用后立即生效。	章节页码	56
指令参数	该指令共有 9 个参数，参数的详细信息如下。		

core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏
crd	坐标系索引，正整数，取值范围请参照表 13-1 中的“插补坐标系序号”一栏
pEnable	获取参数 1：开启手轮引导功能；0：关闭手轮引导功能
pMaster	获取参数：主动轴编码器
pMasterEven	获取参数：主动轴齿轮，传动比系数，主轴位移
pSlaveEven	获取参数：传动比系数，主轴位移
pFilterTime	获取参数：主动轴编码器滤波时间常数
pMode	获取参数：引导脉冲的模式
pfifoEnd	获取参数：返回插补缓冲区的指令行状态 0：未到达插补缓冲区端点，或非手轮引导模式； 1：正向到达插补缓冲区端点； -1：反向达到插补缓冲区端点；
指令返回值	7：参数取值超过范围限制
相关指令	
指令示例	**参考 12.12.2 例程**

指令 66 GTN_GetCrdPos

指令原型	short GTN_GetCrdPos (short core, short crd, double *pPos)		
指令说明	查询该坐标系的当前坐标位置值。获取的坐标值可能和规划位置不一致，取决于建立坐标系的原点是否为零。		
指令类型	立即指令，调用后立即生效。	**章节页码**	65
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
crd	坐标系号。正整数，取值范围请参照表 13-1 中的“插补缓存区序号”一栏		
pPos	读取的坐标系的坐标值。单位：pulse。该参数应该为一个数组首元素的指针，数组的元素个数取决于该坐标系的维数。		
指令返回值	请参照指令返回值列表		
相关指令	无。		
指令示例	**例程 7-11　　插补 FIFO 管理**		

指令 67 GTN_GetCrdPrm

指令原型	short GTN_GetCrdPrm (short core, short crd, TCrdPrm *pCrdPrm)		
指令说明	查询坐标系参数。		
指令类型	立即指令，调用后立即生效。	**章节页码**	65
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
crd	坐标系号。正整数，取值范围请参照表 13-1 中的“插补坐标系序号”一栏		
pCrdPrm	读取坐标系的相关参数 结构体的成员含义参照 GTN_SetCrdPrm 指令说明。		
指令返回值	请参照指令返回值列表.		
相关指令	GTN_SetCrdPrm		
指令示例	无。		

指令 68 GTN_GetCrdStopDec

指令原型	short GTN_GetCrdStopDec (short core, short crd, double *pDecSmoothStop, double *pDecAbruptStop)		
指令说明	查询插补运动平滑停止、急停合成加速度。		
指令类型	立即指令，调用后立即生效。	章节页码	65
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
crd	坐标系号。正整数，取值范围请参照表 13-1 中的“插补坐标系序号”一栏。		
pDecSmoothStop	查询坐标系合成平滑停止加速度，单位：pulse/ms^2。		
pDecAbruptStop	查询坐标系合成急停加速度，单位：pulse/ms^2。		
指令返回值	若返回值为 1：检查当前坐标系是否映射了相关轴。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 69 GTN_GetCrdVel

指令原型	short GTN_GetCrdVel(short core, short crd, double *pSynVel)		
指令说明	查询该坐标系的当前坐标速度值。		
指令类型	立即指令，调用后立即生效。	章节页码	65
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
crd	坐标系号。正整数，取值范围请参照表 13-1 中的“插补坐标系序号”一栏		
pSynVel	读取的坐标系的合成速度值，单位：pulse/ms。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 70 GTN_GetDac

指令原型	short GTN_GetDac(short core, short dac, short *pValue, short count=1, unsigned long *pClock=NULL)		
指令说明	读取 dac 输出电压。		
指令类型	立即指令，调用后立即生效。	章节页码	100
指令参数	该指令共有 5 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
dac	dac 起始轴号。正整数，取值范围请参照表 13-1 中的“轴”一栏		
pValue	输出电压，正整数，取值范围：[-32768, 32767]，对应的电压值为[-10, 10]伏特。		
count	读取的通道数。默认为 1。正整数，取值范围请参照表 13-1 中的“参数个数”一栏		
pClock	读取控制器时钟。		
指令返回值	请参照指令返回值列表。		
相关指令	GTN_SetDac		
指令示例	**例程 8-3 访问 DAC**		

指令 71 GTN_GetDi

指令原型	short GTN_GetDi(short core, short diType, long *pValue)

指令说明	读取数字 IO 输入状态。		
指令类型	立即指令，调用后立即生效。	章节页码	94
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
diType	指定数字 IO 类型。 MC_LIMIT_POSITIVE：正限位。 MC_LIMIT_NEGATIVE：负限位。 MC_ALARM：驱动报警。 MC_HOME：原点开关。 MC_GPI：通用输入。 MC_ARRIVE：电机到位信号。 MC_MPG：手轮 MPG 轴选和倍率信号（5V 电平输入）。		
pValue	数字 IO 输入状态，按位指示 IO 输入电平(根据配置工具 di 的 reverse 值不同而不同)。 当 reverse=0 时，1 表示高电平，0 表示低电平。 当 reverse=1 时，1 表示低电平，0 表示高电平。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	**例程 8-1 访问数字 IO**		

指令 72 GTN_GetDiBit

指令原型	short GTN_GetDiBit(short core,short diType,short diIndex,short *pValue)		
指令说明	读取数字 IO 输入状态。		
指令类型	立即指令，调用后立即生效。	章节页码	94
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
diType	指定数字 IO 类型。 MC_LIMIT_POSITIVE：正限位。 MC_LIMIT_NEGATIVE：负限位。 MC_ALARM：驱动报警。 MC_HOME：原点开关。 MC_GPI：通用输入。 MC_ARRIVE：电机到位信号。 MC_MPG：手轮 MPG 轴选和倍率信号（5V 电平输入）。		
diIndex	数字 IO 序号，取值范围请参照表 13-1 中的“内核”一栏		
pValue	数字 IO 输入状态，按位指示 IO 输入电平(根据配置工具 di 的 reverse 值不同而不同)。 当 reverse=0 时，1 表示高电平，0 表示低电平。 当 reverse=1 时，1 表示低电平，0 表示高电平。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	**例程 8-1 访问数字 IO**		

指令 73 GTN_GetDiEx

指令原型	short GTN_GetDiEx(short core,short diType,long *pValue,short arraySize=1)
指令说明	读取数字 IO 输入状态。

指令类型	立即指令，调用后立即生效。	章节页码	94
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
diType	指定数字 IO 类型。 MC_LIMIT_POSITIVE：正限位。 MC_LIMIT_NEGATIVE：负限位。 MC_ALARM：驱动报警。 MC_HOME：原点开关。 MC_GPI：通用输入。 MC_ARRIVE：电机到位信号。 MC_MPG：手轮 MPG 轴选和倍率信号（5V 电平输入）。		
pValue	数字 IO 输入状态，按位指示 IO 输入电平(根据配置工具 di 的 reverse 值不同而不同)。 当 reverse=0 时，1 表示高电平，0 表示低电平。 当 reverse=1 时，1 表示低电平，0 表示高电平。		
arraySize	数组大小，对应 pValue 变量个数。按照 32 路为一组，因此总的 DI 个数=32*arraySize。 正整数，取值范围请参照表 13-1 中的“参数个数”一栏		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	**例程 8-1 访问数字 IO**		

指令 74 GTN_GetDiRaw

指令原型	short GTN_GetDiRaw(short core, short diType, long *pValue)		
指令说明	读取数字 IO 输入状态的原始值。		
指令类型	立即指令，调用后立即生效。	章节页码	94
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
diType	指定数字 IO 类型。 MC_LIMIT_POSITIVE(该宏定义为 0)：正限位。 MC_LIMIT_NEGATIVE(该宏定义为 1)：负限位。 MC_ALARM(该宏定义为 2)：驱动报警。 MC_HOME(该宏定义为 3)：原点开关。 MC_GPI(该宏定义为 4)：通用输入。 MC_ARRIVE(该宏定义为 5)：电机到位信号。		
pValue	数字 IO 输入状态的原始值，按位指示 IO 输入电平。 1 表示高电平，0 表示低电平。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 75 GTN_GetDiReverseCount

指令原型	short GTN_GetDiReverseCount (short core, short diType, short diIndex, unsigned long *pReverseCount, short count)		
指令说明	取数字量输入信号的变化次数。		
指令类型	立即指令，调用后立即生效。	章节页码	94

指令参数	该指令共有 5 个参数，参数的详细信息如下。
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏
diType	指定数字 IO 类型。 MC_LIMIT_POSITIVE(该宏定义为 0)：正限位。 MC_LIMIT_NEGATIVE(该宏定义为 1)：负限位。 MC_ALARM(该宏定义为 2)：驱动报警。 MC_HOME(该宏定义为 3)：原点开关。 MC_GPI(该宏定义为 4)：通用输入。 MC_ARRIVE(该宏定义为 5)：电机到位信号。
diIndex	数字量输入的索引。 取值范围： 当 diType= MC_LIMIT_POSITIVE、MC_LIMIT_NEGATIVE、MC_ALARM、MC_HOME、MC_ARRIVE 时：正整数，取值范围请参照表 13-1 中的“轴”一栏 diType= MC_GPI 时：正整数，取值范围请参照表 13-1 中的“通用输入”一栏
pReverseCount	读取的数字量输入的变化次数。
count	读取变化次数的数字量输入的个数，默认为 1。 正整数，取值范围请参照表 13-1 中的“参数个数”一栏
指令返回值	请参照指令返回值列表。
相关指令	无。
指令示例	**例程 8-1 访问数字 IO**

指令 76 GTN_GetDo

指令原型	short GTN_GetDo (short core, short doType, long *pValue)		
指令说明	读取数字 IO 输出状态		
指令类型	立即指令，调用后立即生效。	章节页码	94
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
doType	指定数字 IO 类型。 MC_ENABLE(该宏定义为 10)：驱动器使能。 MC_CLEAR(该宏定义为 11)：报警清除。 MC_GPO(该宏定义为 12)：通用输出。		
pValue	数字 IO 输出状态，按位指示 IO 输出电平。 默认情况下，1 表示高电平，0 表示低电平。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 77 GTN_GetDoEx

指令原型	short GTN_GetDoEx (short core, short doType, long *pValue ,short arraySize)		
指令说明	读取数字 IO 输出状态		
指令类型	立即指令，调用后立即生效。	章节页码	94
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
doType	指定数字 IO 类型。		

	MC_ENABLE(该宏定义为 10)：驱动器使能。 MC_CLEAR(该宏定义为 11)：报警清除。 MC_GPO(该宏定义为 12)：通用输出。
pValue	数字 IO 输出状态，按位指示 IO 输出电平。 默认情况下，1 表示高电平，0 表示低电平。
arraySize	数组大小，对应 pValue 变量个数。按照 32 路为一组，因此总的 DI 个数=32*arraySize。 正整数，取值范围请参照表 13-1 中的“参数个数”一栏
指令返回值	请参照指令返回值列表。
相关指令	无。
指令示例	无。

指令 78 GTN_GetEncPos

指令原型	short GTN_GetEncPos (short core, short encoder, double *pValue, short count=1, unsigned long *pClock=NULL)		
指令说明	读取编码器位置。		
指令类型	立即指令，调用后立即生效。	**章节页码**	99
指令参数	该指令共有 5 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
encoder	编码器起始轴号。正整数，取值范围请参照表 13-1 中的“轴”一栏		
pValue	编码器位置。		
count	读取的轴数。默认为 1，正整数，取值范围请参照表 13-1 中的“参数个数”一栏		
pClock	读取控制器时钟。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	**例程 8-2 读取 8 个轴编码器位置值**		

指令 79 GTN_GetEncVel

指令原型	short GTN_GetEncVel (short core, short encoder, double *pValue, short count=1, unsigned long *pClock=NULL)		
指令说明	读取编码器速度		
指令类型	立即指令，调用后立即生效。	**章节页码**	99
指令参数	该指令共有 5 个参数，参数的详细信息如下：		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
encoder	编码器起始轴号。正整数，取值范围请参照表 13-1 中的“轴”一栏。		
pValue	编码器速度。		
count	读取的轴数。默认为 1，正整数，取值范围请参照表 13-1 中的“参数个数”一栏		
pClock	读取控制器时钟。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	**例程 8-2 读取 8 个轴编码器位置值**		

指令 80 GTN_GetFunId

指令原型	short GTN_GetFunId (char *pFunName, short *pFunId)

指令说明	读取运动程序中函数的标识。		
指令类型	立即指令，调用后立即生效。	章节页码	133
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
pFunName	运动程序函数名称。		
pFunId	根据运动程序函数名称查询函数标识。		
指令返回值	若返回值为 1，2007 或者 2008： 请检查重新检查 GTN_Download 是否调用成功。 若失败，请根据 GTN_Download 返回值提示操作，直至成功。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	**例程 11-1 运动程序单线程累加求和**		

指令 81 GTN_GetG0Mode

指令原型	short GTN_GetG0Mode(short core,short crd,short *pMode)		
指令说明	读取插补 G0 指令模式。		
指令类型	立即指令。	章节页码	
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
crd	坐标系号。正整数，取值范围请参照表 13-1 中的“插补坐标系号”一栏		
pMode	插补 G0 指令模式。 0：按照坐标系规划定位指令，定位轨迹为直线； 1：按照单轴规划定位指令，各个轴单独规划，加速度与速度相同，距离短的轴先到位，距离长的轴后到位，定位轨迹可能为折线。		
返回值			
相关指令			
指令示例			

指令 82 GTN_GetGearMaster

指令原型	short GTN_GetGearMaster (short core, short profile, short *pMasterIndex, short *pMasterType, short *pMasterItem)		
指令说明	读取电子齿轮运动跟随主轴。		
指令类型	立即指令，调用后立即生效。	章节页码	61
指令参数	该指令共有 5 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
profile	规划轴号。正整数，取值范围请参照表 13-1 中的“轴”一栏		
pMasterIndex	读取主轴索引。正整数。		
pMasterType	读取主轴类型。 GEAR_MASTER_ENCODER (1) :表示跟随编码器(encoder)的输出值。 GEAR_MASTER_PROFILE (2): 表示跟随规划轴(profile)的输出值。默认为该类型。 GEAR_MASTER_AXIS (3): 表示跟随轴(axis)的输出值。 GEAR_MASTER_AU_ENCODER (4): 表示跟随辅助编码器的输出值。 GEAR_MASTER_MPG_ENCODER (5): 表示跟随 MPG 的输出值。 GEAR_MASTER_ENCODER_OTHER (101): 表示 core2 轴跟随 core1 中编码器(encoder)的输出值		

	GEAR_MASTER_AXIS_OTHER (103): 表示 core2 轴跟随 core1 中轴(axis)的输出值 GEAR_MASTER _AU_ENCODER_OTHER (104): 表示 core2 轴跟随 core1 中辅助编码器的输出值 GEAR_MASTER_MPG_ENCODER_OTHER (105): 表示 core2 轴跟随 core1 中 MPG 的输出值。
pMasterItem	读取输出位置类型。当 masterType=GEAR_MASTER_AXIS 时起作用。 0 表示 axis 的规划位置输出值，默认为该值。 1 表示 axis 的编码器位置输出值。
指令返回值	若返回值为 1：请检查当前轴是否为电子齿轮模式，若不是，请先调用 GTN_PrfGear 将当前轴设置为电子齿轮模式。 其他返回值：请参照指令返回值列表。
相关指令	GTN_SetGearMaster
指令示例	无。

指令 83 GTN_GetGearRatio

指令原型	short GTN_GetGearRatio(short core, short profile, long *pMasterEven, long *pSlaveEven, long *pMasterSlope)		
指令说明	读取电子齿轮比。		
指令类型	立即指令，调用后立即生效。	**章节页码**	61
指令参数	该指令共有 5 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
profile	规划轴号。正整数，取值范围请参照表 13-1 中的“轴”一栏		
pMasterEven	读取传动比系数，主轴位移。单位：pulse。		
pSlaveEven	读取传动比系数，从轴位移。单位：pulse。		
pMasterSlope	读取主轴离合区位移。单位：pulse。 注意：该值只有在运动过程中才能正确读取，在非运动过程中读取的数值均为 0		
指令返回值	若返回值为 1：请检查当前轴是否为电子齿轮模式，若不是，请先调用 GTN_PrfGear 将当前轴设置为电子齿轮模式。 其他返回值：请参照指令返回值列表。		
相关指令	GTN_SetGearRatio		
指令示例	无。		

指令 84 GTN_GetHomePrm

指令原型	short GTN_GetHomePrm(short core, short axis, THomePrm *pHomePrm)		
指令说明	读取设置到控制器的Smart Home回原点参数。		
指令类型	立即指令，调用后立即生效。	**章节页码**	108
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
axis	进行回原点的轴号，正整数，取值范围请参照表 13-1 中的“轴”一栏		
pHomePrm	获取Smart Home回原点的参数，该参数为一结构体，详细参数定义及说明请参照结构体 THomePrm。		
指令返回值	请参照《GTN 系列运动控制器之基本功能》第 3 章 指令返回值及其意义。		
相关指令	无。		
指令示例	**例程 9-3 回零功能**		

指令 85 GTN_GetHomeStatus

指令原型	short GTN_GetHomeStatus(short core, short axis, THomeStatus *pHomeStatus)		
指令说明	获取Smart Home回原点的状态。		
指令类型	立即指令，调用后立即生效。	**章节页码**	108
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
axis	进行回原点的轴号，正整数，取值范围请参照表 13-1 中的“轴”一栏		

pHomeStatus	获取Smart Home回原点的状态参数，该参数为一结构体，详细参数定义及说明请参照结构体THomeStatus。
	typedef struct
	{
	short run; // 是正在进行回原点，0—已停止运动，1-正在回原点
	short stage; // 回原点运动的阶段
	short error; // 回原点过程的发生的错误
	short pad1; // 保留（无具体含义）
	long capturePos; // 捕获到Home或Index时刻的编码器位置
	long targetPos; // 需要运动到的目标位置（原点位置或者原点位置+偏移量）
	} THomeStatus;
	
	回原点运动的阶段宏定义：
	HOME_STAGE_IDLE(0)：未启动Smart Home回原点
	HOME_STAGE_START(1)：启动Smart Home回原点
	HOME_STAGE_ON_HOME_LIMIT_ESCAPE(2)：正在脱离原点/限位
	HOME_STAGE_SEARCH_LIMIT(10)：正在寻找限位
	HOME_STAGE_SEARCH_LIMIT_STOP(11)：触发限位停止
	HOME_STAGE_SEARCH_LIMIT_ESCAPE (13)：反方向运动脱离限位
	HOME_STAGE_SEARCH_LIMIT_RETURN (15)：重新回到限位
	HOME_STAGE_SEARCH_LIMIT_RETURN_STOP(16)：重新回到限位停止
	HOME_STAGE_SEARCH_HOME(20)：正在搜索Home
	HOME_STAGE_SEARCH_HOME_STOP(22)：搜索岛Home后停止
	HOME_STAGE_SEARCH_HOME_RETURN (25)：搜索到Home后运动到捕获的Home位置
	HOME_STAGE_SEARCH_INDEX(30)：正在搜索Index
	HOME_STAGE_SEARCH_GPI(40)：正在搜索GPI
	HOME_STAGE_SEARCH_GPI_RETURN(45)：搜索到GPI后运动到捕获GPI的位置
	HOME_STAGE_GO_HOME(80)：正在执行回原点过程
	HOME_STAGE_END(100)：回原点结束
	回原点过程的发生的错误宏定义：
	HOME_ERROR_NONE (0)：未发生错误
	HOME_ERROR_NOT_TRAP_MODE(1)：执行Smart Home回原点的轴不是处于点位运动模式
	HOME_ERROR_DISABLE (2)：执行Smart Home回原点的轴未使能
	HOME_ERROR_ALARM (3)：执行Smart Home回原点的轴驱动报警
	HOME_ERROR_STOP (4)：未完成回原点，轴停止运动（例如搜索距离太短）		

	HOME_ERROR_STAGE (5)：回原点阶段错误 HOME_ERROR_HOME_MODE (6)：模式错误（例，轴已经启动Smart Home，再重复调用回原点指令，则报错） HOME_ERROR_SET_CAPTURE_HOME (7)：设置Home捕获模式失败 HOME_ERROR_NO_HOME (8)：未找到Home HOME_ERROR_SET_CAPTURE_INDEX (9)：设置Index捕获模式失败 HOME_ERROR_NO_INDEX (10)：位找到Index
指令返回值	请参照《GTN 系列运动控制器之基本功能》第 3 章 指令返回值及其意义。
相关指令	无。
指令示例	**例程 9-3 回零功能**

指令 86 GTN_GetJogPrm

指令原型	short GTN_GetJogPrm(short core, short profile, TJogPrm *pPrm)		
指令说明	读取 Jog 运动模式下的运动参数。		
指令类型	立即指令，调用后立即生效。	章节页码	57
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
profile	规划轴号。正整数，取值范围请参照表 13-1 中的“轴”一栏		
pPrm	设置 Jog 模式运动参数。该参数为一个结构体，包含三个参数，详细的参数定义及说明请参照 GTN_SetJogPrm 指令说明。		
指令返回值	若返回值为 1： (1) 若当前轴在规划运动，请调用 GTN_Stop 停止运动再调用该指令。 (2) 请检查当前轴是否为 Jog 模式，若不是，请先调用 GTN_PrfJog 将当前轴设置为 Jog 模式。 其他返回值：请参照指令返回值列表。		
相关指令	GTN_SetJogPrm		
指令示例	**例程 7-2 Jog 运动**		

指令 87 GTN_GetLimitStatus

指令原型	short GTN_GetLimitStatus (short core, short axis, short *pLimitPositive, short *pLimitNegative)		
指令说明	读取软限位状态		
指令类型	立即指令，调用后立即生效。	章节页码	126
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
axis	轴号,正整数，取值范围请参照表 13-1 中的“轴”一栏		
pLimitPositive	1) 硬件正限位触发时，置起限位状态的bit0； 2) 软件正限位触发时，置起限位状态的bit1；		
pLimitNegative	1) 硬件负限位触发时，置起限位状态的bit0； 2) 软件负限位触发时，置起限位状态的bit1；		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	**例程 10-1 软限位使用**		

指令 88 GTN_GetMtrBias

指令原型	short GTN_GetMtrBias(short core, short dac, short *pBias)		
指令说明	读取模拟量输出通道的零漂电压补偿值。		
指令类型	立即指令，调用后立即生效。	章节页码	41
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
dac	模拟量输出通道号，正整数，取值范围请参照表13-1中的“轴”一栏		
pBias	读取的零漂补偿值。		
指令返回值	请参照指令返回值列表。		
相关指令	GTN_SetMtrBias		
指令示例	无。		

指令 89 GTN_GetMtrLmt

指令原型	short GTN_GetMtrLmt(short core, short dac, short *pLimit)		
指令说明	读取模拟量输出通道的输出电压饱和极限值。		
指令类型	立即指令，调用后立即生效。	章节页码	41
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
dac	模拟量输出通道号。正整数，取值范围请参照表13-1中的“轴”一栏		
pLlimit	读取的输出电压饱和极限值。		
指令返回值	请参照指令返回值列表。		
相关指令	GTN_SetMtrLmt		
指令示例	无。		

指令 90 GTN_GetPid

指令原型	short GTN_GetPid(short core, short control, short index, TPid *pPid)		
指令说明	读取 PID 参数。		
指令类型	立即指令，调用后立即生效。	章节页码	146
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
control	伺服控制器编号。正整数，取值范围请参照表 13-1 中的“轴”一栏		
index	伺服控制参数的索引号。取值范围：[1, 3]。		
pPid	读取 PID 参数。		
指令返回值	请参照指令返回值列表。		
相关指令	GTN_SetPid		
指令示例	**例程 5-3 设置第 1 轴为闭环控制方式**		

指令 91 GTN_GetPlsPos

指令原型	short GTN_GetPlsPos(short core,short pulse,double *pValue,short count=1,unsigned long *pClock=NULL)		
指令说明	读取内部脉冲计数器位置。		
指令类型	立即指令，调用后立即生效。	章节页码	102

指令参数	该指令共有 5 个参数，参数的详细信息如下。
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏
pulse	脉冲计数器对应轴起始序号，正整数，取值范围请参照表 13-1 中的“轴”一栏
pValue	内部脉冲计数器位置
count	脉冲计数器对应轴个数，默认为 1。 正整数，取值范围请参照表 13-1 中的“参数个数”一栏
pClock	读取控制器时钟。
指令返回值	请参照指令返回值
相关指令	GTN_SetPlsPos
指令示例	**例程 8-5 访问内部脉冲计数器**

指令 92 GTN_GetPlsVel

指令原型	short GTN_GetPlsVel(short core,short pulse,double *pValue,short count=1,unsigned long *pClock=NULL)		
指令说明	读取内部脉冲计数器速度。		
指令类型	立即指令，调用后立即生效。	章节页码	102
指令参数	该指令共有 5 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
pulse	脉冲计数器对应轴起始序号，正整数，取值范围请参照表 13-1 中的“轴”一栏		
pValue	内部脉冲计数器速度		
count	脉冲计数器对应轴个数，默认为 1。 正整数，取值范围请参照表 13-1 中的“参数个数”一栏		
pClock	读取控制器时钟。		
指令返回值	请参照指令返回值		
相关指令	无。		
指令示例	无		

指令 93 GTN_GetPos

指令原型	short GTN_GetPos(short core, short profile, long *pPos)		
指令说明	读取目标位置。		
指令类型	立即指令，调用后立即生效。	章节页码	54
指令参数	该指令共有 3 个参数，参数的详细信息如下：		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
profile	规划轴号。正整数，取值范围请参照表 13-1 中的“轴”一栏		
pos	读取目标位置，单位：pulse。		
指令返回值	若返回值为 1：请检查当前轴是否为 Trap 模式，若不是，请先调用 GTN_PrfTrap 将当前轴设置为 Trap 模式。 其他返回值：请参照指令返回值列表。		
相关指令	GTN_SetPos		
指令示例	无。		

指令 94 GTN_GetPosCompareFifoMode

指令原型	short GTN_GetPosCompareFifoMode(short core, short posCompareIndex, short *pMode)		
指令说明	获取存储实际位置比较点的缓冲区模式（大小 2048）。		
指令类型	立即指令，调用后立即生效。	章节页码	99
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏 **核 2 目前不支持位置比较功能**		
posCompareIndex	位置比较模块编号，正整数，取值范围请参照表 13-1 中的“轴”一栏		
pMode	缓冲区模式。		
指令返回值	请参照指令返回值列表。		
相关指令			
指令示例			

指令 95 GTN_GetPosCompareLatchValue

指令原型	short GTN_GetPosCompareLatchValue(short core,short posCompareIndex,long count,long *pDataX,long *pDataY,long *pCount,TLatchValueInfo *pInfo)		
指令说明	获取存储实际位置比较点数据。		
指令类型	立即指令，调用后立即生效。	章节页码	99
指令参数	该指令共有 7 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏 **核 2 目前不支持位置比较功能**		
posCompareIndex	位置比较模块编号，正整数，取值范围请参照表 13-1 中的“轴”一栏		
count	本次需要采集的数据量。		
pDataX	存储 X 方向位置点的数组，大小应该大于 count。		
pDataY	存储 Y 方向位置点的数组，大小应该大于 count。		
pCount	实际获取的实际位置比较点个数，如果缓冲区已经无数据，那么获取此值为 0。		

pInfo	typedef struct
	{
	short fifoFull; //缓冲区空间满了
	short pad1[3];
	double pad2[2];
	}TLatchValueInfo;
	如果缓冲区满了，需要重新调用 GTN_SetPosCompareFifoMode 设置模式，才能清除。		

指令返回值	请参照指令返回值列表。		
相关指令			
指令示例			

指令 96 GTN_GetPosCompareLinear

指令原型	short GTN_GetPosCompareLinear(short core, short posCompareIndex, TPosCompareLinear *pLinear)
指令说明	读取一维线性比较输出参数

指令类型	缓存区指令	章节页码	114
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏 **核2目前不支持位置比较功能**		
posCompareIndex	缓存区 FIFO，正整数，取值范围请参照表 13-1 中的“位置比较输出”一栏		
pLinear	位置比较输出相关参数，详见指令 GTN_SetPosCompareLinear 中的参数 3		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	无		

指令 97 GTN_GetPosCompareMode

指令原型	shortGTN_GetPosCompareMode(short core, short posCompareIndex, TPosCompareMode *pMode);		
指令说明	读取位置比较输出模式		
指令类型	立即指令，调用后立即生效。	章节页码	114
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏 **核 2 目前不支持位置比较功能**		
posCompareIndex	位置比较索引，正整数，取值范围[1,8]。		
pMode	读取输出模式信息，详细信息请参照指令 GTN_SetPosCompareMode 中的参数 3		
指令返回值	请参照指令返回值列表。		
相关指令	GTN_SetPosCompareMode		
指令示例	**例程 9-4 设置位置比较输出通道 1 为一维 FIFO 模式**		

指令98 GTN_GetPosComparePsoPrm

指令原型	shortGTN_GetPosComparePsoPrm(short core, short posCompareIndex, TPosComparePsoPrm *pPrm)
指令说明	读取位置同步比较输出
指令类型	立即指令 章节页码 114
指令参数	该指令共有 3 个参数，参数的详细信息如下。
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏 **核 2 目前不支持位置比较功能**
posCompareIndex	缓存区 FIFO，正整数，取值范围请参照表 13-1 中的“位置比较输出”一栏

pPrm	typedef struct
	{
	unsigned long count; //保留
	unsigned short hso; //保留
	unsigned short gpo; //保留
	long startPosX; //保留
	long startPosY; //保留
	long syncPos; //同步位置增量

	long time; //保留
	short reserve[20]; //保留
	} TPosComparePsoPrm;

指令返回值	请参照指令返回值
相关指令	无。
指令示例	**无。**

指令 99 GTN_GetPosErr

指令原型	short GTN_GetPosErr(short core, short control, long *pError)		
指令说明	读取跟随误差极限值。		
指令类型	立即指令，调用后立即生效。	章节页码	41
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
control	伺服控制器编号。正整数，取值范围请参照表13-1中的“伺服控制器”一栏		
pError	读取的跟随误差极限值。单位：pulse。		
指令返回值	请参照指令返回值列表。		
相关指令	GTN_SetPosErr		
指令示例	无。		

指令 100 GTN_GetPrfAcc

指令原型	short GTN_GetPrfAcc(short core, short profile, double *pValue, short count=1, unsigned long *pClock=NULL)		
指令说明	读取规划加速度。		
指令类型	立即指令，调用后立即生效。	章节页码	47
指令参数	该指令共有 5 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
profile	起始规划轴号。正整数，取值范围请参照表 13-1 中的“轴”一栏		
pValue	规划加速度。单位：pulse/ms^2。		
count	读取的轴数，默认为 1，正整数，取值范围请参照表 13-1 中的“参数个数”一栏		
pClock	读取控制器时钟，默认值为：NULL，即不用读取控制器时钟。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	**例程 6-1 获取轴 1 的轴状态、运动模式、位置、速度和加速度**		

指令 101 GTN_GetPrfMode

指令原型	short GTN_GetPrfMode(short core, short profile, long *pValue, short count=1, unsigned long *pClock=NULL)		
指令说明	读取轴运动模式。		
指令类型	立即指令，调用后立即生效。	章节页码	47
指令参数	该指令共有 5 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
profile	起始规划轴号，正整数，取值范围请参照表 13-1 中的“轴”一栏		
pValue	轴运动模式。		

	0：点位运动，控制器上电后默认为该模式； 1：Jog 模式； 2：PT 模式； 3：电子齿轮模式； 4：Follow 模式； 5：插补模式； 6：Pvt 模式。
count	读取的轴数，默认为 1，正整数，取值范围请参照表 13-1 中的“参数个数”一栏
pClock	读取控制器时钟，默认值为：NULL，即不用读取控制器时钟。
指令返回值	请参照指令返回值列表。
相关指令	无。
指令示例	**例程 6-1 获取轴 1 的轴状态、运动模式、位置、速度和加速度**

指令 102 GTN_GetPrfPos

指令原型	short GTN_GetPrfPos(short core, short profile, double *pValue, short count=1, unsigned long *pClock=NULL)		
指令说明	读取规划位置。		
指令类型	立即指令，调用后立即生效。	**章节页码**	47
指令参数	该指令共有 5 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
profile	起始规划轴号，正整数，取值范围请参照表 13-1 中的“轴”一栏		
pValue	规划位置。单位：pulse。		
count	读取的轴数，默认为 1，正整数，取值范围请参照表 13-1 中的“参数个数”一栏		
pClock	读取控制器时钟，默认值为：NULL，即不用读取控制器时钟。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	**例程 6-1 获取轴 1 的轴状态、运动模式、位置、速度和加速度**		

指令 103 GTN_GetPrfVel

指令原型	short GTN_GetPrfVel(short core, short profile, double *pValue, short count=1, unsigned long *pClock=NULL)		
指令说明	读取规划速度。		
指令类型	立即指令，调用后立即生效。	**章节页码**	47
指令参数	该指令共有 5 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
profile	起始规划轴号，正整数，取值范围请参照表 13-1 中的“轴”一栏		
pValue	规划速度。单位：pulse/ms。		
count	读取的轴数，默认为 1，正整数，取值范围请参照表 13-1 中的“参数个数”一栏		
pClock	读取控制器时钟，默认值为：NULL，即不用读取控制器时钟。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	**例程 6-1 获取轴 1 的轴状态、运动模式、位置、速度和加速度**		

指令 104 GTN_GetPrfSts

指令原型	short GTN_GetPrfSts(short core, short profile, long *pSts, short count=1, unsigned long *pClock=NULL)		
指令说明	读取规划器状态。		
指令类型	立即指令，调用后立即生效。	章节页码	47
指令参数	该指令共有 5 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
profile	起始轴号，正整数，取值范围请参照表 13-1 中的“轴”一栏		
pSts	32 位轴状态字，详细定义参见表 6-2 轴状态定义。		
count	读取的轴数，默认为 1，正整数，取值范围请参照表 13-1 中的“参数个数”一栏		
pClock	读取控制器时钟，默认值为：NULL，即不用读取控制器时钟。		
指令返回值	请参照指令返回值列表。		
相关指令	GTN_ClrSts		
指令示例			

指令 105 GTN_GetRemainderSegNum

指令原型	short GTN_GetRemainderSegNum(short core, short crd, long *pSegment, short fifo=0)		
指令说明	读取未完成的插补段段数。		
指令类型	立即指令，调用后立即生效。	章节页码	65
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
crd	坐标系号。正整数，取值范围请参照表 13-1 中的“插补坐标系序号”一栏		
pSegment	读取的剩余插补段的段数。		
fifo	插补缓存区号。默认为 0，正整数，取值范围请参照表 13-1 中的“插补缓存区序号”一栏。		
指令返回值	若返回值为 1：检查当前坐标系是否映射了相关轴。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 106 GTN_GetResCount

指令原型	short GTN_GetResCount(short core,short type,short *pCount)		
指令说明	读取主卡资源个数		
指令类型	立即指令，调用后立即生效。	章节页码	20
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
type	资源类型 MC_TERMINAL (宏定义为 50)　　//模块个数 其他类型请参照 GTN_SetResCount 的参数 2		
pCount	读取数据个数		
指令返回值	请参照指令返回值列表。		
相关指令	GTN_SetResCount		
指令示例	**例程 4-1 设置主卡软件资源**		

指令 107 GTN_GetResMax

指令原型	short GTN_GetResMax (short core,short type,short *pCount)		
指令说明	读取主卡资源最大个数		
指令类型	立即指令，调用后立即生效。	章节页码	20
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
type	资源类型 参照指令 GTN_GetResCount 指令的参数 2		
pCount	读取数据个数 最大资源个数请参照表4-2		
指令返回值	请参照指令返回值列表。		
相关指令	GTN_GetResCount		
指令示例	**例程 4-1 设置主卡软件资源**		

指令 108 GTN_GetRetainValue

指令原型	short GTN_GetRetainValue (short core,unsigned long address,short count,short *pData)		
指令说明	读取MRAM存储芯片数据		
指令类型	立即指令，调用后立即生效。	章节页码	129
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
address	起始地址，正整数，取值范围[0,16383] 当 count=1,取值范围为[0,16383] ... 当 count=16,取值范围为[0,16368] 注意：address+count 小于等于 16384		
count	读取数据个数，正整数，取值范围[1,16]		
pData	读取数据的数值		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	**例程 10-2 掉电存储操作**		

指令 109 GTN_GetSense

指令原型	short GTN_GetSense(short core,short dataType,short dataIndex,short *pValue)		
指令说明	输入输出资源的电平逻辑。		
指令类型	立即指令，调用后立即生效。	章节页码	94
指令参数	该指令共有 4 个参数，参数的详细信息如下：		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
dataType	输入输出资源类型 MC_ENCODE: (该宏定义为 23)：编码器。 MC_LIMIT_POSITIVE(该宏定义为 0)：正限位。 MC_LIMIT_NEGATIVE(该宏定义为 1)：负限位。 MC_ALARM(该宏定义为 2)：驱动报警。 MC_HOME(该宏定义为 3)：原点开关。 MC_GPI(该宏定义为 4)：通用输入。		

	MC_GPO(该宏定义为 12)：通用输出。 MC_ENABLE(该宏定义为 10)：使能信号。 MC_CLEAR(该宏定义为 11)：报警清除信号。
dataIndex	资源序号，正整数， (1)当 dataType 为 MC_ENCODE、MC_LIMIT_POSITIVE、MC_LIMIT_NEGATIVE、MC_ALARM、MC_HOME、MC_ENABLE、MC_CLEAR 时，取值范围请参照表 13-1 中的“轴”一栏 (2) 当 dataType 为 MC_GPI 时，取值范围请参照表 13-1 中的“通用输入”一栏 (3) 当 dataType 为 MC_GPO 时，取值范围请参照表 13-1 中的“通用输出”一栏
value	0：信号不取反， 1：信号取反。
指令返回值	请参照指令返回值列表。
相关指令	GTN_SetSense
指令示例	无。

指令 110 GTN_GetSoftLimitEx

指令原型	short GTN_GetSoftLimitEx (short core, short axis, double *pPositive, double *pNegative)		
指令说明	读取轴正向软限位和负向软限位。		
指令类型	立即指令，调用后立即生效。	**章节页码**	126
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
axis	轴号，正整数，取值范围请参照表 13-1 中的“轴”一栏		
pPositive	读取正向软限位。单位：脉冲		
pNegative	读取负向软限位。单位：脉冲		
指令返回值	请参照指令返回值列表。		
相关指令	GTN_SetSoftLimitEx		
指令示例	**例程 10-1 软限位使用**		

指令 111 GTN_GetSoftLimitMode

指令原型	short GTN_GetSoftLimitMode(short core,short axis,short *pMode)		
指令说明	读取软限位模式		
指令类型	立即指令，调用后立即生效。	**章节页码**	126
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
axis	轴号,正整数，取值范围请参照表 13-1 中的“轴”一栏		
mode	0 SOFT_LIMIT_MODE_STOP// 超越软限位位置后开始减速停止 1 SOFT_LIMIT_MODE_LIMIT // 限制在软限位范围之内		
指令返回值	请参照指令返回值列表。		
相关指令	GTN_GetSoftLimitMode		
指令示例	**例程 10-1 软限位使用**		

指令 112 GTN_GetStopDec

指令原型	short GTN_GetStopDec(short core, short profile, double *pDecSmoothStop, double *pDecAbruptStop)
指令说明	读取平滑停止减速度和急停减速度。

指令类型	立即指令，调用后立即生效。	章节页码	41
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
profile	规划器的编号，正整数，取值范围请参照表13-1中的“轴”一栏		
pDecSmoothStop	读取的平滑停止减速度。单位：pulse/ms^2。		
pDecAbruptStop	读取的急停减速度。单位：pulse/ms^2。		
指令返回值	请参照指令返回值列表。		
相关指令	GTN_SetStopDec		
指令示例	无。		

指令 113 GTN_GetSts

指令原型	short GTN_GetSts(short core, short axis, long *pSts, short count=1, unsigned long *pClock=NULL)		
指令说明	读取轴状态。		
指令类型	立即指令，调用后立即生效。	章节页码	47
指令参数	该指令共有 5 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
axis	起始轴号，正整数，取值范围请参照表 13-1 中的“轴”一栏		
pSts	32 位轴状态字，详细定义参见表 6-2 轴状态定义。		
count	读取的轴数，默认为 1，正整数，取值范围请参照表 13-1 中的“参数个数”一栏		
pClock	读取控制器时钟，默认值为：NULL，即不用读取控制器时钟。		
指令返回值	请参照指令返回值列表。		
相关指令	GTN_ClrSts		
指令示例	**例程 6-1 获取轴 1 的轴状态、运动模式、位置、速度和加速度**		

指令 114 GTN_GetTerminalPermitEx

指令原型	short GTN_GetTerminalPermitEx(short core, short station, short dataType, short *pPermit, short index=1, short count=1)		
指令说明	读取轴状态。		
指令类型	立即指令，调用后立即生效。	章节页码	
指令参数	该指令共有 6 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
station	模块序号，正整数，取值范围请参照表 13-1 中的“网络端子板模块序号”一栏		
dataType	设置输出功能类型的数据类型： MC_GPO（12）：通用数字量输出，对应硬件的 DO（参见硬件手册） MC_HSO（18）：高速 IO 输出，对应硬件的 HSO（参见硬件手册）		
pPermit	按位设置硬件输出通道信号输出的类型 从 bit0-bit15 按位表示对应信号类型输出，1：控制权打开；0：控制权关闭 Bit0: 通用 IO 指令输出（当 dataType 为 MC_HSO 时，该值无效） Bit1: 第一路位置比较输出 Bit2: 第二路位置比较输出 Bit3: 使能激光开关光输出（当 dataType 为 MC_GPO 时，该值无效） Bit4: 使能 PWM 信号输出（当 dataType 为 MC_GPO 时，该值无效） Bit5-bit15 对于 403 模块保留		

index	需要设置控制权的起始硬件通道序号，默认值为 1 取值范围请参考
count	需要设置控制权的硬件通道个数，默认值为 1，取值范围为[1，20]
指令返回值	请参照指令返回值列表。
相关指令	GTN_SetTerminalPermitEx
指令示例	

指令 115 GTN_GetTerminalStatus

指令原型	short GTN_GetTerminalStatus(short core,short index,TTerminalStatus *pTerminalStatus)		
指令说明	读取模块通讯状态		
指令类型	立即指令，调用后立即生效。	章节页码	46
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
index	模块序号，正整数，取值范围请参照表 13-1 中的“网络端子板模块序号”一栏		

pTerminalStatus	typedefstruct
	{
	unsigned short type; // 类型
	short id; // id,模块物理序号（即物理ID）
	long status; // 状态，bit0：0通讯错误,1通讯正常；
	//bit1:0模块未使能1：模块使能
	unsignedlong synchCount; // 保留
	unsigned long ringNetType; // bit0:网络状态0：未连接1：连接(网络初始化成功后该位置1，否则置0),bit1:网络类型0：环网，1：非环网
	unsigned long portStatus; // 模块端口状态，bit13: w0，portB Work Ready；bit12：portB物理连接情况，bit9: portA Work Ready，bit8：portA 物理连接情况，bit7：safe_status_n。0:进入安全模式，1：正常运行，bit3~2：dc_status同步状态。1正常
	unsigned long sportDropCount; // 模块 SPORT 包丢失个数
	unsigned long reserve[7]; // 保留
	} TTerminalStatus;		

指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例			

指令 116 GTN_GetTerminalVersion

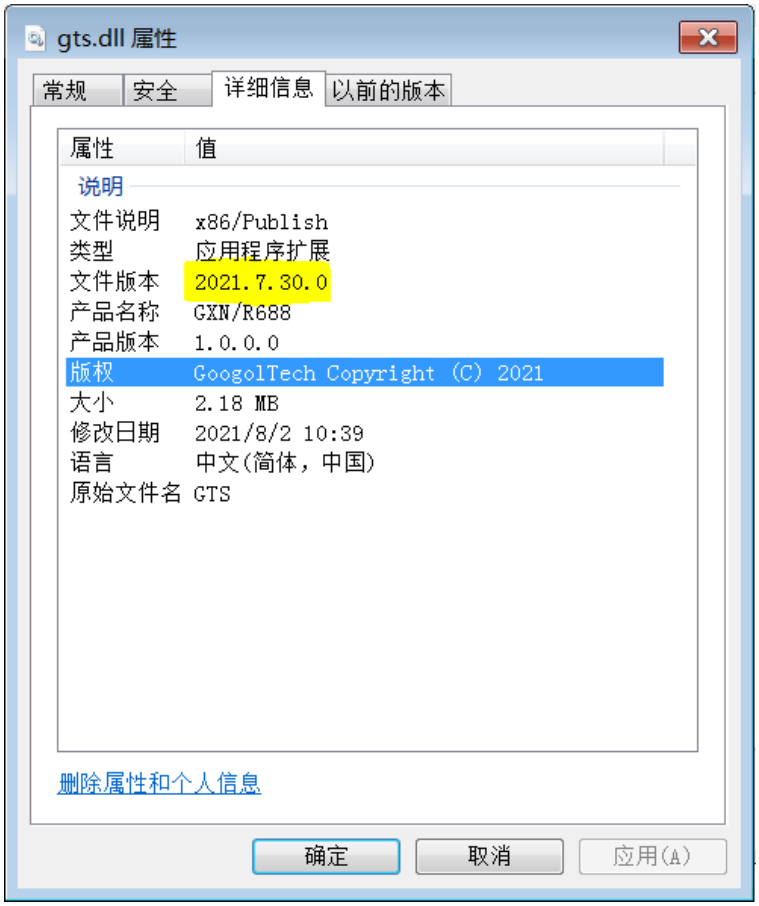
指令原型	shortGTN_GetTerminalVersion(shortcore,shortindex,TVersion*pTerminalVersion)		
指令说明	读取端子板版本号		
指令类型	立即指令，调用后立即生效。	章节页码	144
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
index	模块序号，正整数，取值范围请参照表13-1中的“网络端子板模块序号”一栏		
pTerminalVersion	读取端子板的固件版本号。 typedef struct		

```
{
    short year;
    short month;
    short day;
    short version;
    short user;
    short reserve1;
    short reserve2;
    short reserve3;
} TVersion;
```

指令返回值	请参照指令返回值列表。
相关指令	无。

用户读取动态链接库的版本，点击 gts.dll，右键->属性->详细信息处可以查看动态链接库的版本。

指令示例



文件说明	动态链接库的说明，x86/Publish 表示是 32 位的发布版本的动态链接库。
文件版本	动态链接库的版本日期，如 2021.7.30.0，表示该动态链接库发布于：2021 年 7 月 30 日

例程 12-1 读取运动控制器版本号

指令 117 GTN_GetThreadSts

指令原型	short GTN_GetThreadSts(short core, short thread, TThreadSts *pThreadSts)		
指令类型	立即指令，调用后立即生效。	章节页码	133
指令参数	该指令共有 3 个参数，参数的详细信息如下。		

core	内核，正整数，取值范围请参照表13-1中的“内核”一栏
thread	线程编号。取值范围：[0, 31]。

pThreadSts	读取线程状态
	typedef struct ThreadSts
	{
	short run; // 运行状态
	short error; // 当前执行的指令返回值
	double result; // 函数返回值
	short line; // 当前执行的指令行号
	} TThreadSts;

指令返回值	请参照指令返回值列表。
相关指令	无。
指令示例	**例程 11-1 运动程序单线程累加求和**

指令 118 GTN_GetTrapPrm

指令原型	short GTN_GetTrapPrm(short core, short profile, TTrapPrm *pPrm)		
指令说明	读取点位运动模式下的运动参数。		
指令类型	立即指令，调用后立即生效。	章节页码	54
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
profile	规划轴号，正整数，取值范围请参照表 13-1 中的“轴”一栏		
pPrm	读取点位运动模式运动参数，该参数为一个结构体，包含四个参数，详细的参数定义及说明请参照 GTN_SetTrapPrm 指令说明。		
指令返回值	若返回值为 1：请检查当前轴是否为 Trap 模式，若不是，请先调用 GTN_PrfTrap 将当前轴设置为 Trap 模式。 其他返回值：请参照指令返回值列表。		
相关指令	GTN_SetTrapPrm		
指令示例	**例程 7-1 点位运动**		

指令 119 GTN_GetTriggerEx

指令原型	shortGTN_GetTriggerEx(short core, short index, TTriggerEx *pTrigger)		
指令说明	读取 Trigger 捕获方式。		
指令类型	立即指令，调用后立即生效。	章节页码	104
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
index	Trigger 序号，正整数，取值范围请参照表 13-1 中的“轴”一栏		

pTrigger	读取 Trigger 触发模式信息
	typedef struct Trigger
	{
	short latchType; //锁存类型：MC_ENCODER 23
	//MC_AU_ENCODER 26
	short latchIndex; //编码器序号，取值范围请参考
	short probeType; // 捕获类型，取值范围[1,3]
	// CAPTURE_HOME 1		

	// CAPTURE_INDEX2
	// CAPTURE_PROBE 3
	short probeIndex; // 捕获类型对应的DI序号
	short sense; // 捕获沿,0:下降沿，1:上升沿
	long offset; // 捕获位置偏置值，设置为0，只有GTM模块支持offset为非0值
	unsigned long loop; // 捕获循环测试，0：无限循环，其他代表循环次数
	short windowOnly; // 捕获窗使能，0：不开启捕获窗，1：开启捕获窗
	long firstPosition; // windowOnly=1时生效，触发捕获位置的起点
	long lastPosition; // windowOnly=1时生效，触发捕获位置的终点
	}TTriggerEx;

指令返回值	请参照指令返回值列表。
相关指令	GTN_SetTriggerEx
指令示例	**例程 9-1 Home/Index 捕获**

指令 120 GTN_GetTriggerLatchValue

指令原型	short GTN_GetTriggerLatchValue(short core,short index,long count,long *pValue,long *pCount,TLatchValueInfo *pInfo)		
指令说明	读取 Trigger 捕获的值。		
指令类型	立即指令。	章节页码	
指令参数	该指令共有 6 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
index	控制器存储的捕获值索引，取值从 1 开始。		
count	捕获值个数。		
pValue	保存捕获值的数组。		
pCount	实际读取的捕获值个数。		

pInfo	读取控制器存储捕获值的信息。
	typedef struct
	{
	short fifoFull; // 标识存储捕获值的缓冲区是否满，0-没有满，1-满了
	short pad1[3];
	double pad2[2];
	}TLatchValueInfo;		

返回值			
相关指令			
指令示例			

指令 121 GTN_GetTriggerStatusEx

指令原型	shortGTN_GetTriggerStatusEx(short core,short index, TTriggerStatusEx *pTriggerStatus,short count=1)		
指令说明	读取 Trigger 捕获状态。		
指令类型	立即指令，调用后立即生效。	章节页码	104
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
i	Trigger 序号，正整数，取值范围请参照表 13-1 中的“轴”一栏		

pTriggerStatus	typedef struct TriggerStatusEx
	{
	short execute;//捕获运行状态，1:使能捕获，0：未使能捕获
	short done;//捕获状态，1：捕获完成，0：捕获中
	long position;//捕获位置
	unsigned long clock; //保留
	unsigned long loopCount; //重复捕获的第几次捕获
	}TTriggerStatusEx;

count	读取Trigger个数，取值范围[1,4]
指令返回值	请参照指令返回值列表。
相关指令	无。
指令示例	**例程 9-1 Home/Index 捕获**

指令 122 GTN_GetUserSegNum

指令原型	short GTN_GetUserSegNum (short core, short crd, long *pSegment, short fifo=0)		
指令说明	读取自定义插补段段号。		
指令类型	立即指令，调用后立即生效。	**章节页码**	65
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
crd	坐标系号。正整数，取值范围请参照表 13-1 中的“插补坐标系序号”一栏		
pSegment	读取的用户自定义的插补段段号。		
fifo	插补缓存区号。默认值为：0，取值范围请参照表 13-1 中的“插补缓存区序号”一栏		
指令返回值	若返回值为 1：检查当前坐标系是否映射了相关轴。 其他返回值：请参照指令返回值列表。		
相关指令	GTN_SetUserSegNum		
指令示例	无。		

指令 123 GTN_GetVarId

指令原型	short GTN_GetVarId(char *pFunName, char *pVarName, TVarInfo *pVarInfo)		
指令说明	读取运动程序中变量的标识。		
指令类型	立即指令，调用后立即生效。	**章节页码**	133
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
pFunName	全局变量输入 NULL。 局部变量所在函数的名称。		
pVarName	运动程序变量名称。		
pVarInfo	根据运动程序函数名称和变量名称查询变量标识。		
指令返回值	若返回值为 1，2007 或者 2008： 请检查重新检查 GTN_Download 是否调用成功。 若失败，请根据 GTN_Download 返回值提示操作，直至成功。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	**例程 11-1 运动程序单线程累加求和**		

指令 124 GTN_GetVarValue

指令原型	short GTN_GetVarValue(short core, short page, TVarInfo *pVarInfo, double *pValue, short count=1)		
指令说明	读取运动程序中变量的值。		
指令类型	立即指令，调用后立即生效。	章节页码	133
指令参数	该指令共有 5 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
page	数据页编号。 全局变量为-1。 局部变量取值范围：[0, 31]。		
pVarInfo	需要访问的变量标识。		
pValue	需要读取的变量值。		
count	需要读取的变量值的数量，取值范围：[1, 8]。		
指令返回值	请参照指令返回值列表。		
相关指令	GTN_SetVarValue		
指令示例	**例程 11-1　　运动程序单线程累加求和**		

指令 125 GTN_GetVel

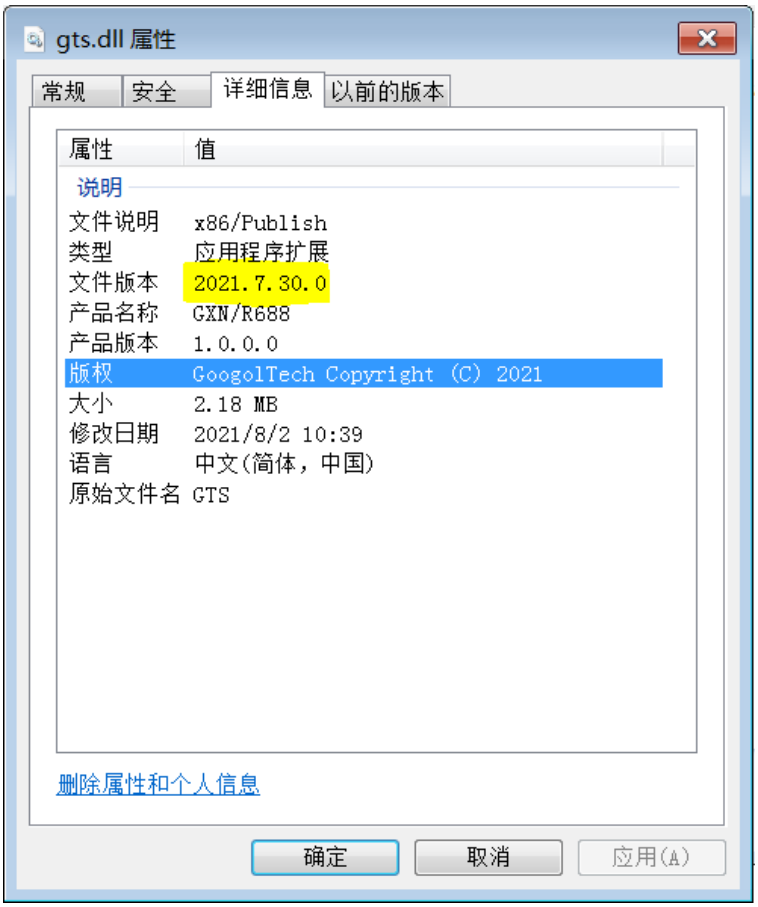
指令原型	short GTN_GetVel(short core, short profile, double *pVel)		
指令说明	读取目标速度。		
指令类型	立即指令，调用后立即生效。	章节页码	54, 57
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
profile	规划轴号，正整数，取值范围请参照表 13-1 中的“轴”一栏		
pVel	读取目标速度。单位：pulse/ms。		
指令返回值	若返回值为 1：请检查当前轴是否为 Trap 模式，若不是，请先调用 GTN_PrfTrap 将当前轴设置为 Trap 模式。 其他返回值：请参照指令返回值列表。		
相关指令	GTN_SetVel		
指令示例	无。		

指令 126 GTN_GetVersion

指令原型	short GTN_GetVersion(short core, char **pVersion)		
指令说明	读取运动控制器固件的版本号。		
指令类型	立即指令，调用后立即生效。	章节页码	144
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
pVersion	读取的运动控制器的固件版本号字符串。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	无		

指令 127 GTN_GetVersionEx

指令原型	short GTN_GetVersionEx(short core, short type, TVersion *pVersion);

指令说明	读取固件的版本号。		
指令类型	立即指令，调用后立即生效。	**章节页码**	144
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
type	类型参数 type=1，读取控制器的版本信息		
pVersion	读取动态库版本号。 typedef struct { short year; short month; short day; short version; short user; short reserve1; short reserve2; short reserve3; } TVersion;		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	**用户读取动态链接库的版本，点击 gts.dll，右键**->属性->详细信息处可以查看动态链接库的版本。  		
文件说明	动态链接库的说明，x86/Publish 表示是 32 位的发布版本的动态链接库。		

文件版本	动态链接库的版本日期，如 2021.7.30.0，表示该动态链接库发布于：2021 年 7 月 30 日

例程 12-1　　读取运动控制器版本号

指令 128 GTN_GoHome

指令原型	short GTN_GoHome(short core,shortaxis,THomePrm *pHomePrm)		
指令说明	启动 Smart Home 实现各种方式回原点		
指令类型	立即指令，调用后立即生效。	章节页码	108
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
axis	需要进行回原点的轴号，正整数，取值范围请参照表 13-1 中的“轴”一栏		

pHomePrm

设置 Smart Home 回原点的参数，该参数为一结构体，详细参数定义及说明请参照结构体 THomePrm：

```
typedef struct
{
    short mode;          // 回原点模式，参考下面的回原点模式宏定义
    short moveDir;       // 设置启动搜索原点时的运动方向：非正数-负方向，正整数-正方向
    short indexDir;      // 设置搜索Index的运动方向：非正数-负方向，正整数-正方向
    short edge;          // 设置捕获沿：0-下降沿，非0值-上升沿
    short triggerIndex; // 用于设置触发器：取值-1和[1,8]，-1表示使用的触发器
                           和轴号对应，其它值表示使用其它轴的触发器，触发器
                           用于实现高速硬件捕获，默认设置为-1即可
    short pad1[3]; // 保留（不需要设置）
    double velHigh;      // 搜索Home速度（单位：pulse/ms）
    double velLow;       // 搜索Index速度（单位：pulse/ms）
    double acc;          // 回原点运动的加速度（单位：pulse/ms^2）
    double dec;          // 回原点运动的减速度（单位：pulse/ms^2）
    short smoothTime; // 回原点运动的平滑时间：取值[0,1)，具体含义与GTS系列控制器点位运动相似
    short pad2[3];       // 保留（不需要设置）
    long homeOffset;     // 最终停止的位置相对于原点的偏移量
    long searchHomeDistance;    // Home最大搜索距离，
                                   //0表示不限制搜索距离，默认为805306368或-805306368
    long searchIndexDistance;    // Index最大搜索距离，
                                //0表示不限制搜索距离，默认为805306368或-805306368
    long escapeStep;    // 反方向离开限位的脱离步长，当移动一次设定的步长，仍未脱离限位时，会再次移动设定的步长，直到脱离限位
    long pad3[2];        // 保留（不需要设置）
} THomePrm;
```

回原点模式宏定义：

HOME_MODE_LIMIT	(10)：限位回原点
HOME_MODE_LIMIT_HOME	(11)：限位+Home回原点
HOME_MODE_LIMIT_INDEX	(12)：限位+Index回原点

	HOME_MODE_LIMIT_HOME_INDEX (13)：限位+Home+Index回原点 HOME_MODE_HOME (20)：Home回原点 HOME_MODE_HOME_INDEX (22)：Home+Index回原点 HOME_MODE_INDEX (30)：Index回原点
指令返回值	请参照指令返回值列表。
相关指令	无。
指令示例	**例程 9-3 回零功能**

指令 129 GTN_HandwheelInit

指令原型	GTN_HandwheelInit(short core, short mode)		
指令说明	初始化单轴手轮功能。		
指令类型	立即指令，调用后立即生效。	章节页码	157
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
mode	选择主轴的源。 #define MC_MPG 9 //使用MPG #define MC_AU_ENCODER 26 //使用辅助编码器 #define MC_ENCODER 23 //使用主轴编码器		
指令返回值	7：参数取值超过范围限制。		
相关指令			
指令示例	**参考 12.12.1 例程**		

指令 130 GTN_InitLookAhead

指令原型	short GTN_InitLookAhead (short core, short crd, short fifo, double T, double accMax, short n, TCrdData *pLookAheadBuf)		
指令说明	初始化插补前瞻缓存区。		
指令类型	立即指令，调用后立即生效。	章节页码	65
指令参数	该指令共有 7 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
crd	坐标系号。正整数，取值范围请参照表 13-1 中的“插补坐标系序号”一栏		
fifo	插补缓存区号。默认值为 0，取值范围请参照表 13-1 中的“插补缓存区序号”一栏		
T	拐弯时间。单位：ms。		
accMax	最大加速度。单位：pulse/ms^2。		
n	前瞻缓存区大小。取值范围：[0, 32767]。		
pLookAheadBuf	前瞻缓存区内存区指针。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	**例程 7-7 前瞻预处理例程**		

指令 131 GTN_LmtsOffEx

指令原型	short GTN_LmtsOffEx (short core, short axis, short limitType = -1, short limitMode = -1)
指令说明	控制轴限位信号无效。

指令类型	立即指令，调用后立即生效。	章节页码	41
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
Core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
axis	控制轴号，正整数，取值范围请参照表13-1中的“轴”一栏		
limitType	需要有效的限位类型。 MC_LIMIT_POSITIVE(该宏定义为0)：将该轴的正限位无效。 MC_LIMIT_NEGATIVE(该宏定义为1)：将该轴的负限位无效。 -1 ：将该轴的正限位、负限位都无效，默认为该值。		
limitMode	LIMIT_MODE_EXTERMAL (该宏定义为 0) ：设置该轴的硬限位 LIMIT_MODE_SOFT (该宏定义为 1) ：设置该轴的软限位 -1 ：设置该轴的硬限位和软限位		
指令返回值	若返回值为 1：请检查相应轴限位报警，配置文件是否已经配置了限位无效。 其他返回值：请参照指令返回值列表。		
相关指令	GTN_LmtsOnEx		
指令示例	**例程 5-2 设置第 1 轴为脉冲控制“脉冲+方向”方式**		

指令 132 GTN_LmtsOnEx

指令原型	short GTN_LmtsOnEx (short core, short axis, short limitType= -1, short limitMode= -1)		
指令说明	控制轴限位信号有效。		
指令类型	立即指令，调用后立即生效。	章节页码	41
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
axis	控制轴号，正整数，取值范围请参照表13-1中的“轴”一栏		
limitType	需要有效的限位类型。 MC_LIMIT_POSITIVE(该宏定义为0)：将该轴的正限位有效。 MC_LIMIT_NEGATIVE(该宏定义为1)：将该轴的负限位有效。 -1：将该轴的正限位和负限位都有效，默认为该值。		
limitMode	LIMIT_MODE_EXTERMAL (该宏定义为 0) ：设置该轴的硬限位 LIMIT_MODE_SOFT (该宏定义为 1) ：设置该轴的软限位 -1 ：设置该轴的硬限位和软限位		
指令返回值	若返回值为 1：请检查相应轴限位报警，配置文件是否已经配置了限位无效。 其他返回值：请参照指令返回值列表。		
相关指令	GTN_LmtsOffEx		
指令示例			

指令 133 GTN_LnXY

指令原型	short GTN_LnXY (short core, short crd, long x, long y, double synVel, double synAcc, double velEnd=0, short fifo=0)		
指令说明	XY 平面二维直线插补。		
指令类型	缓存区指令。	章节页码	65
指令参数	该指令共有 8 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		

crd	坐标系号。正整数，取值范围请参照表 13-1 中的“插补坐标系序号”一栏
x	插补段 x 轴终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。
y	插补段 y 轴终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。
synVel	插补段的目标合成速度。取值范围：(0, 65536]，单位：pulse/ms。
synAcc	插补段的合成加速度。取值范围：(0, 32767]，单位：pulse/ms^2。
velEnd	插补段的终点速度。取值范围：[0, 32767]，单位：pulse/ms。该值只有在没有使用前瞻预处理功能时才有意义，否则该值无效。默认值为：0。
fifo	插补缓存区号。默认值为 0，取值范围请参照表 13-1 中的“插补缓存区序号”一栏
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。
相关指令	无。
指令示例	**例程 7-5 直线插补例程**

指令 134 GTN_LnXYG0

指令原型	short GTN_LnXYG0 (short core, short crd, long x, long y, double synVel, double synAcc, short fifo=0)		
指令说明	二维直线插补，且终点速度始终为 0。		
指令类型	缓存区指令。	章节页码	65
指令参数	该指令共有 7 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
crd	坐标系号。正整数，取值范围请参照表 13-1 中的“插补坐标系序号”一栏		
x	插补段 x 轴终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。		
y	插补段 y 轴终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。		
synVel	插补段的目标合成速度。取值范围：(0, 65536]，单位：pulse/ms。		
synAcc	插补段的合成加速度。取值范围：(0, 32767]，单位：pulse/ms^2。		
fifo	插补缓存区号。默认值为 0，取值范围请参照表 13-1 中的“插补缓存区序号”一栏		
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 135 GTN_LnXYZ

指令原型	short GTN_LnXYZ (short core, short crd, long x, long y, long z, double synVel, double synAcc, double velEnd=0, short fifo=0)		
指令说明	三维直线插补。		
指令类型	缓存区指令。	章节页码	65

指令参数	该指令共有 9 个参数，参数的详细信息如下。
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏
crd	坐标系号。正整数，取值范围请参照表 13-1 中的“插补坐标系序号”一栏
x	插补段 x 轴终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。
y	插补段 y 轴终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。
z	插补段 z 轴终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。
synVel	插补段的目标合成速度。取值范围：(0, 65536]，单位：pulse/ms。
synAcc	插补段的合成加速度。取值范围：(0, 32767]，单位：pulse/ms^2。
velEnd	插补段的终点速度。取值范围：[0, 32767]，单位：pulse/ms。该值只有在没有使用前瞻预处理功能时才有意义，否则该值无效。默认值为：0。
fifo	插补缓存区号，默认值为 0，取值范围请参照表 13-1 中的“插补缓存区序号”一栏
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。
相关指令	无。
指令示例	无。

指令 136 GTN_LnXYZA

指令原型	short GTN_LnXYZA (short core, short crd, long x, long y, long z, long a, double synVel, double synAcc, double velEnd=0, short fifo=0)		
指令说明	四维直线插补。		
指令类型	缓存区指令。	章节页码	65
指令参数	该指令共有 10 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
crd	坐标系号。正整数，取值范围请参照表 13-1 中的“插补坐标系序号”一栏		
x	插补段 x 轴终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。		
y	插补段 y 轴终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。		
z	插补段 z 轴终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。		
a	插补段 a 轴终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。		
synVel	插补段的目标合成速度。取值范围：(0, 65536]，单位：pulse/ms。		
synAcc	插补段的合成加速度。取值范围：(0, 32767]，单位：pulse/ms^2。		
velEnd	插补段的终点速度。取值范围：[0, 32767]，单位：pulse/ms。该值只有在没有使用前瞻预处理功能时才有意义，否则该值无效。默认值为：0。		
fifo	插补缓存区号。默认值为 0，取值范围请参照表 13-1 中的“插补缓存区序号”一栏		
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。		
相关指令	无。		

指令示例	无。

指令 137 GTN_LnXYZAG0

指令原型	short GTN_LnXYZAG0 (short core, short crd, long x, long y, long z, long a, double synVel, double synAcc, short fifo=0)		
指令说明	四维直线插补，且终点速度始终为 0。		
指令类型	缓存区指令。	章节页码	65
指令参数	该指令共有 9 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
crd	坐标系号。正整数，取值范围请参照表 13-1 中的“插补坐标系序号”一栏		
x	插补段 x 轴终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。		
y	插补段 y 轴终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。		
z	插补段 z 轴终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。		
a	插补段 a 轴终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。		
synVel	插补段的目标合成速度。取值范围：(0, 65536]，单位：pulse/ms。		
synAcc	插补段的合成加速度。取值范围：(0, 32767]，单位：pulse/ms^2。		
fifo	插补缓存区号。默认值为 0，取值范围请参照表 13-1 中的“插补缓存区序号”一栏		
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 138 GTN_LnXYZG0

指令原型	short GTN_LnXYZG0 (short core, short crd, long x, long y, long z, double synVel, double synAcc, short fifo=0)		
指令说明	三维直线插补，且终点速度始终为 0。		
指令类型	缓存区指令。	章节页码	65
指令参数	该指令共有 8 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
crd	坐标系号。正整数，取值范围请参照表 13-1 中的“插补坐标系序号”一栏		
x	插补段 x 轴终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。		
y	插补段 y 轴终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。		
z	插补段 z 轴终点坐标值。取值范围：[-1073741824, 1073741823]，单位：pulse。		
synVel	插补段的目标合成速度。取值范围：(0, 65536]，单位：pulse/ms。		
synAcc	插补段的合成加速度。取值范围：(0, 32767]，单位：pulse/ms^2。		
fifo	插补缓存区号。默认值为 0，取值范围请参照表 13-1 中的“插补缓存区序号”一栏		
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。		

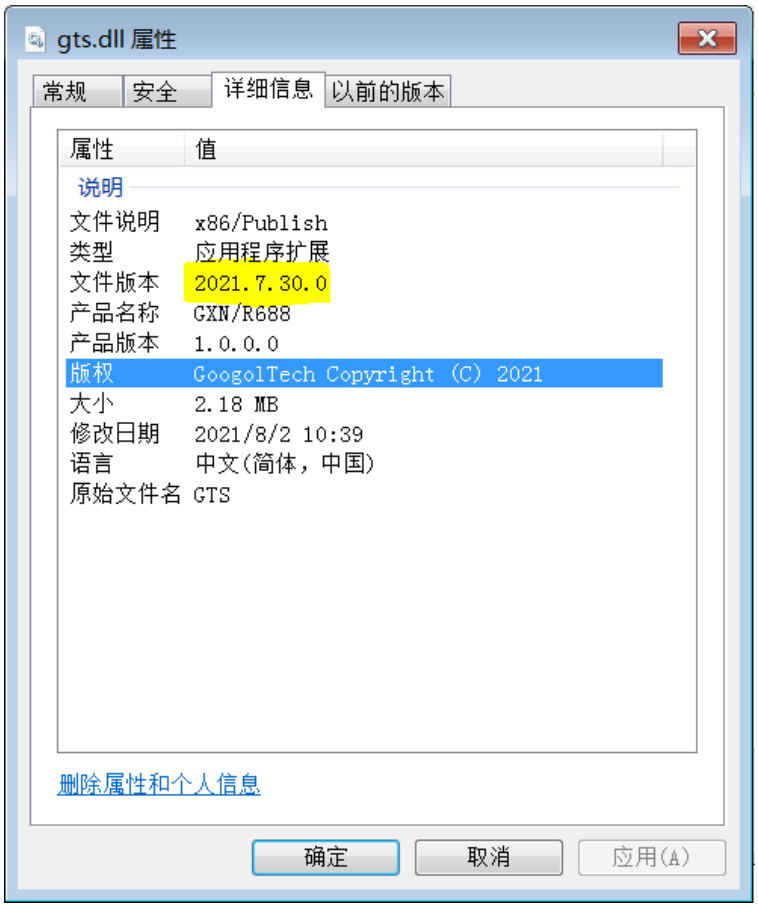
	(3) 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。
相关指令	无。
指令示例	无。

指令 139 GTN_LoadConfig

指令原型	short GTN_LoadConfig(short core, char *pFile)		
指令说明	下载配置信息到运动控制器，调用该指令后需再调用 GTN_ClrSts才能使该指令生效。		
指令类型	立即指令，调用后立即生效。	章节页码	40
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
pFile	配置文件的文件名。文件名格式：*.cfg 或*.CFG。用户可根据自己的需求，使用运动控制器管理软件 MotionStudio 生成此配置文件。		
指令返回值	若返回值为 1：请停止各轴规划运动后再设置。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	**例程 7-1 点位运动**		

指令 140 GTN_Open

指令原型	short GTN_Open (short channel=5, short param=1)		
指令说明	打开运动控制器,并按照默认配置初始化网络。		
指令类型	立即指令，调用后立即生效。	章节页码	144
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
channel	打开运动控制器的方式		
param	保留		
指令返回值	请参照指令返回值列表。 注意：在初始化网络之后，建议调用 GTN_Reset		
相关指令	GTN_Close		
指令示例	**用户读取动态链接库的版本，点击 gts.dll，右键->**属性->详细信息处可以查看动态链接库的版本。		



文件说明	动态链接库的说明，x86/Publish 表示是 32 位的发布版本的动态链接库。
文件版本	动态链接库的版本日期，如 2021.7.30.0，表示该动态链接库发布于：2021 年 7 月 30 日

例程 12-1　　读取运动控制器版本号

指令 141 GTN_PauseThread

指令原型	short GTN_PauseThread(short core, short thread)		
指令说明	暂停正在运行的线程。		
指令类型	立即指令，调用后立即生效。	**章节页码**	133
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
thread	线程编号。取值范围：[0, 31]。		
指令返回值	若返回值为 1： (1) 请检查线程号是否已经绑定。 (2) 请检查相应的数据页中是否超出范围。 其他返回值：请参照指令返回值列表。		
相关指令	GTN_RunThread; GTN_StopThread		
指令示例	无。		

指令 142 GTN_PosCompareClear

指令原型	short GTN_PosCompareClear(short core,short posCompareIndex)		
指令说明	清除位置比较输出缓存区数据		
指令类型	缓存区指令	章节页码	114
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏 **核 2 目前不支持位置比较功能**		
posCompareIndex	缓存区 FIFO，正整数，取值范围请参照表 13-1 中的“位置比较输出”一栏		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	**无。**		

指令 143 GTN_PosCompareData

指令原型	short GTN_PosCompareData(short core, short posCompareIndex, TPosCompareData *pData)		
指令说明	设置一维位置比较输出数据（FIFO 模式）		
指令类型	缓存区指令	章节页码	114
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏 **核 2 目前不支持位置比较功能**		
posCompareIndex	缓存区 FIFO，正整数，取值范围请参照表 13-1 中的“位置比较输出”一栏		

pData	typedefstruct
	{
	long pos; //位置比较输出的位置
	unsigned short hso; //位置比较输出hso通道的输出数值,按位表示HSObit0-bit9对应HSO0-HSO9，bit15:表示逻辑位。在激光功能和位置比较输出功能复用时生效；脉冲模式：0表示无输出，1表示输出脉冲；电平模式：0表示无输出，1表示有输出。
	unsigned short gpo; //通用GPO通道的输出数值，按位表示GPO，bit0-bit15分别对应GPO0-GPO15。脉冲模式：0表示无输出，1表示输出脉冲；电平模式：0表示拉低，1表示拉高。
	unsigned long segmentNumber; //设置数据段号
	} TPosCompareData;
	注意：如果该参数为NULL，则表示将剩余数据传入DSP缓存区，详见9.5.6		

指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	**例程 9-7 通用一维位置比较输出（FIFO 模式）**		

指令 144 GTN_PosCompareData2D

指令原型	short GTN_PosCompareData2D(short core, short posCompareIndex, TPosCompareData2D *pData);		
指令说明	设置二维位置比较输出数据（FIFO 模式）		
指令类型	缓存区指令。	章节页码	114
指令参数	该指令共有 3 个参数，参数的详细信息如下。		

core	内核，正整数，取值范围请参照表13-1中的“内核”一栏 核2目前不支持位置比较功能
posCompareIndex	缓存区FIFO，正整数，取值范围请参照表13-1中的“位置比较输出”一栏

pData	typedefstruct
	{
	long posX; //X轴位置比较输出的位置
	long posY; //Y轴位置比较输出的位置
	unsigned short hso; //位置比较输出hso通道的输出数值,按位表示HSObit0-bit9对应HSO0-HSO9，bit15:表示逻辑位。在激光功能和位置比较输出功能复用时生效；脉冲模式：0表示无输出，1表示输出脉冲；电平模式：0表示无输出，1表示有输出。
	unsigned short gpo; //通用GPO通道的输出数值，按位表示GPO，bit0-bit15分别对应GPO0-GPO15。脉冲模式：0表示无输出，1表示输出脉冲；电平模式：0表示拉低，1表示拉高。
	unsigned long segmentNumber; //设置数据段号
	} TPosCompareData2D;
	注意：如果该参数为NULL，则表示将剩余数据传入DSP缓存区，详见9.5.6

指令返回值	请参照指令返回值列表。
相关指令	无。
指令示例	**例程 9-9 二维位置比较输出（FIFO 模式）**

指令 145 GTN_PosCompareHsOff

指令原型	short GTN_PosCompareHsOff(short core, short posCompareIndex = 1)		
指令说明	关闭振镜 DMA 通道。		
指令类型	立即指令，调用后立即生效。	章节页码	114
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏 核2目前不支持位置比较功能		
posCompareIndex	缓存区FIFO，正整数，取值范围请参照表13-1中的“位置比较输出”一栏		
指令返回值	请参照指令返回值列表。		
相关指令	GTN_PosCompareHsOn		
指令示例	无		

指令 146 GTN_PosCompareHsOn

指令原型	short GTN_PosCompareHsOn(short core, short posCompareIndex=1, short dmaBuf=1, unsigned short threshold=200)		
指令说明	打开位置比较输出的 DMA 通道。		
指令类型	立即指令，调用后立即生效。	章节页码	114
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏 核2目前不支持位置比较功能		
posCompareInd	缓存区FIFO，正整数，取值范围请参照表13-1中的“位置比较输出”一栏		

ex	
dmaBuf	DMA 数据缓存区序号，正整数，默认值为 1.。取值范围请参照表 13-1 中的“DMA 数据缓存区号”一栏
threshold	DMA 指令数，正整数，默认值为 200。取值范围请参照表 13-1 中的“DMA 指令数”一栏 如果指令数达到该阈值，系统会自动将缓存区指令下发至运动控制器
指令返回值	请参照指令返回值列表。
相关指令	GTN_PosCompareHsOff
指令示例	无

指令 147 GTN_PosCompareInfo

指令原型	short GTN_PosCompareInfo(short core,short posCompareIndex,TPosCompareInfo *pInfo)		
指令说明	读取位置比较输出相关信息		
指令类型	缓存区指令	章节页码	114
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏 **核2目前不支持位置比较功能**		
posCompareIndex	缓存区 FIFO，正整数，取值范围请参照表 13-1 中的“位置比较输出”一栏		

pInfo	typedefstruct
	{
	unsigned short config; //保留
	unsigned short fifoEmpty; //fifo跑空标志，1：表示曾经跑空，清
	//空FPGA后才能清0,其他位保留
	unsigned short head; //保留
	unsigned short tail; //保留
	unsigned long commandReceive; //接收指令数量
	unsigned long commandSend; //发送指令数量
	long posX; // 在FIFO模式下，表示用户触发的x轴位置；在PSO模式下，表示用户的开关PSO位置，其他模式下无意义。
	long posY; // 在FIFO模式下，表示用户触发的y轴位置；在PSO模式下，表示用户的开关PSO位置，其他模式下无意义。
	} TPosCompareInfo;		

指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	无		

指令 148 GTN_PosCompareSpace

指令原型	short GTN_PosCompareSpace(short core,short posCompareIndex,unsigned short *pSpace);		
指令说明	读取位置比较剩余空间指令		
指令类型	立即指令	章节页码	114
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏 **核 2 目前不支持位置比较功能**		
posCompareIn	位置比较索引，正整数，取值范围[1,8]。		

dex	
pSpace	位置比较剩余空间，最大为 1000
指令返回值	请参照指令返回值列表。
相关指令	无
指令示例	**例程 9-10　　获取实际的位置比较点**

指令 149 GTN_PosCompareStart

指令原型	short GTN_PosCompareStart(short core,short posCompareIndex);		
指令说明	开始位置比较输出		
指令类型	缓存区指令	章节页码	114
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏 **核2目前不支持位置比较功能**		
posCompareIndex	缓存区 FIFO，正整数，取值范围请参照表 13-1 中的“位置比较输出”一栏		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例			

指令 150 GTN_PosCompareStatus

指令原型	short GTN_PosCompareStatus(short core,short posCompareIndex,TPosCompareStatus *pStatus);		
指令说明	读取位置比较输输出功能状态		
指令类型	立即指令	章节页码	114
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏 **核2目前不支持位置比较功能**		
posCompareIndex	缓存区FIFO，正整数，取值范围请参照表13-1中的“位置比较输出”一栏		

pStatus	typedefstruct
	{
	unsigned short mode; // 0:FIFO模式；1：Linear模式
	unsigned short run; //1:开始，0：停止
	unsigned short space; //FIFO剩余空间
	unsigned long pulseCount; //输出脉冲个数
	unsigned short hso; //位置比较输出hso通道的输出数值
	unsigned short gpo; //通用GPO通道的输出数值
	unsigned long segment; //执行的段号
	} TPosCompareStatus;		

指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	**无。**		

指令 151 GTN_PosCompareStop

指令原型	short GTN_PosCompareStop(short core,short posCompareIndex);		
指令说明	停止位置比较输出		
指令类型	缓存区指令	章节页码	114
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏 **核2目前不支持位置比较功能**		
posCompareIndex	缓存区 FIFO，正整数，取值范围请参照表 13-1 中的“位置比较输出”一栏		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	**无。**		

指令 152 GTN_PrfGear

指令原型	short GTN_PrfGear (short core, short profile, short dir)		
指令说明	设置指定轴为电子齿轮运动模式。		
指令类型	立即指令，调用后立即生效。	章节页码	54, 61
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
profile	规划轴号，正整数，取值范围请参照表 13-1 中的“轴”一栏		
dir	设置跟随方式。 0 表示双向跟随，正整数表示正向跟随，负整数表示负向跟随。		
指令返回值	若返回值为 1： (1) 若当前轴在规划运动，请调用 GTN_Stop 停止运动再调用该指令。 (2) 当前已经是电子齿轮模式，但再次设置的 dir 与当前的 dir 不一致。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	**例程 7-3 电子齿轮跟随**		

指令 153 GTN_PrfJog

指令原型	short GTN_PrfJog(short core, short profile)		
指令说明	设置指定轴为 Jog 运动模式。		
指令类型	立即指令，调用后立即生效。	章节页码	54, 57
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
profile	规划轴号，正整数，取值范围请参照表 13-1 中的“轴”一栏		
指令返回值	若返回值为 1：若当前轴在规划运动，请调用 GTN_Stop 停止运动再调用该指令。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	**例程 7-2 Jog 运动**		

指令 154 GTN_PrfTrap

指令原型	short GTN_PrfTrap(short core, short profile)		
指令说明	设置指定轴为点位运动模式。		
指令类型	立即指令，调用后立即生效。	章节页码	54,

指令参数	该指令共有 2 个参数，参数的详细信息如下。
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏
profile	规划轴号，正整数，取值范围请参照表 13-1 中的“轴”一栏
指令返回值	若返回值为 1：若当前轴在规划运动，请调用 GTN_Stop 停止运动再调用该指令。 其他返回值：请参照指令返回值列表。
相关指令	无。
指令示例	**例程 7-1 点位运动**

指令 155 GTN_ProfileScale

指令原型	short GTN_ProfileScale(short core, short axis, short alpha, short beta)		
指令说明	设置控制轴的规划器当量变换值。注意：alpha的绝对值要大于beta的绝对值。		
指令类型	立即指令，调用后立即生效。	章节页码	41
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
axis	控制轴号，正整数，取值范围请参照表13-1中的“轴”一栏		
alpha	规划器当量的alpha值，取值范围：[-32768, 0)和(0, 32767]。		
beta	规划器当量的beta值，取值范围：[-32767, 0)和(0, 32767]。		
指令返回值	若返回值为 1：若当前轴再规划运动，请调用 GTN_Stop 停止运动在调用该指令。 其他返回值：请参照指令返回值列表。		
相关指令	GTN_EncScale		
指令示例	无。		

指令 156 GTN_ReadAuEncPos

指令原型	short GTN_ReadAuEncPos (short core, short encoder, double *pValue, short count=1)		
指令说明	读取辅助编码器位置。		
指令类型	立即指令，调用后立即生效。	章节页码	99
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
encoder	辅助编码器起始轴号。正整数，取值范围请参照表 13-1 中的“辅助编码器”一栏		
pValue	编码器位置。		
count	读取的辅编个数。默认为 1，正整数，取值范围为[1,6]		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	**例程 8-2 读取 8 个轴编码器位置值**		

指令 157 GTN_ReadMpgInfo

指令原型	short GTN_ReadMpgInfo (short core, short mpg,TMpgInfo *pMpgInfo)		
指令说明	读取手轮相关信息		
指令类型	立即指令，调用后立即生效。	章节页码	99
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
mpg	手轮编号。正整数，取值范围请参照表 13-1 中的“手轮”一栏		
pMpgInfo	手轮相关信息		

	typedef struct
	{
	double pos; // 手轮编码器位置
	double vel; // 手轮速度
	double reserve[2]; //保留
	long di; //手轮接口对应的DI（MC_MPG）
	long reserve1[3]; //保留
	} TMpgInfo;

指令返回值	请参照指令返回值列表。
相关指令	无。
指令示例	**例程 8-2 读取 8 个轴编码器位置值**

指令 158 GTN_Reset

指令原型	short GTN_Reset(short core)		
指令说明	复位运动控制器。		
指令类型	立即指令，调用后立即生效。	章节页码	144
指令参数	该指令共有 1 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	**例程 5-2 设置第 1 轴为脉冲控制“脉冲+方向”方式**		

指令 159 GTN_RunThread

指令原型	short GTN_RunThread(short core, short thread)		
指令说明	启动线程。		
指令类型	立即指令，调用后立即生效。	章节页码	133
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
thread	线程编号，取值范围：[0, 31]。		
指令返回值	若返回值为 1： (1) 请检查线程号是否已经绑定。 (2) 请检查相应的数据页中是否超出范围。 其他返回值：请参照指令返回值列表。		
相关指令	无		
指令示例	**例程 11-1 运动程序单线程累加求和**		

指令 160 GTN_SetAdcFilterPrm

指令原型	short GTN_SetAdcFilterPrm(short core,short adc,double k)		
指令说明	设置模拟量输入的滤波参数。		
指令类型	立即指令，调用后立即生效。	章节页码	99
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
adc	位置比较模块编号，正整数，取值范围请参照表 13-1 中的“轴”一栏		
k	滤波系数，取值范围[0,1)。越接近 1 效果越强，但滞后越大。		

指令返回值	请参照指令返回值列表。
相关指令	
指令示例	

指令 161 GTN_SetAuDac

指令原型	short GTN_SetAuDac(short core,short dac,short *pValue,short count)		
指令说明	设置 AUDAC 的输出电压对应的数值		
指令类型	立即指令，调用后立即生效。	章节页码	100
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
dac	AUDAC 的起始序号。正整数，取值范围请参照表 13-1 中的“非轴模拟量输出”一栏		
pValue	输出电压。0 对应 0V，32767 对应+10V。		
count	设置的通道数。默认为 1。 正整数，取值范围请参照表 13-1 中的“参数个数”一栏		
指令返回值	请参照指令返回值列表。		
相关指令	GTN_GetAuDac		
指令示例			

指令 162 GTN_SetAxisBand

指令原型	short GTN_SetAxisBand(short core, short axis, long band, long time)		
指令说明	设置轴到位误差带。 规划器静止，规划位置和实际位置的误差小于设定误差带，并且在误差带内保持设定时间后，置起到位标志。		
指令类型	立即指令，调用后立即生效。	章节页码	147
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
axis	轴号，正整数，取值范围请参照表 13-1 中的“轴”一栏		
band	误差带大小。单位：脉冲。		
time	误差带保持时间。单位：中断周期。		
指令返回值	请参照指令返回值列表。		
相关指令	GTN_GetAxisBand		
指令示例	**例程 12-2　　电机到位检测功能**		

指令 163 GTN_SetAxisPrfVelFilter

指令原型	short GTN_SetAxisPrfVelFilter(short core, short axis, short filterNumExp)		
指令说明	设置轴规划速度滤波系数		
指令类型	立即指令，调用后立即生效。	章节页码	无
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
axis	轴号，正整数，取值范围请参照表 13-1 中的“轴”一栏		
filterNumExp	滤波算法阶数		
指令返回值	请参照指令返回值列表。		
相关指令	无		

指令示例	无

指令 164 GTN_SetBacklash

指令原型	short GTN_SetBacklash (short core, short axis, long compValue, double compChangeValue, long compDir)		
指令说明	设置反向间隙补偿的相关参数。		
指令类型	立即指令，调用后立即生效。	章节页码	152
指令参数	该指令共有 5 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
axis	需要进行反向间隙补偿的轴的编号，正整数，取值范围请参照表 13-1 中的“内核”一栏		
compValue	反向间隙补偿值，当为 0 时表示没有使能反向间隙补偿功能。取值范围：[0, 32767]，单位：脉冲。		
compChangeValue	反向间隙补偿的变化量。取值范围：[0, 32767]，单位：pulse/ms。 当该参数的值为 0 或者大于等于 compValue 时，则反向间隙的补偿量将瞬间叠加在规划位置上，没有渐变的过程。		
compDir	反向间隙补偿方向。 0：只补偿负方向，当电机向负方向运动时，将施加补偿量，当电机向正方向运动时，不施加补偿量。 1：只补偿正方向，当电机向正方向运动时，将施加补偿量，当电机向负方向运动时，不施加补偿量。		
指令返回值	若返回值为 1：若当前轴在规划运动，请调用 GTN_Stop 停止运动再调用该指令。 其他返回值：请参照指令返回值列表。		
相关指令	GTN_GetBacklash		
指令示例	无。		

指令 165 GTN_SetCompensate2D

指令原型	short GTN_SetCompensate2D(short core, short axis,TCompensate2D *pComp2d)		
指令说明	设置二维补偿参数（**调用前需要先使能补偿轴**）。		
指令类型	立即指令，调用后立即生效。	章节页码	154
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
axis	需要补偿的轴号,正整数，取值范围请参照表 13-1 中的“轴”一栏		

pComp2d	设置二维补偿参数，该参数为一结构体，详细参数定义及说明请参照结构体 TCompensate2D。
	typedef struct
	{
	short enable; // 二维补偿使能标志，为0：不使能，非0值：使能。
	short tableIndex; // 所使用的二维补偿表索引（取值范围为[1, 4]）
	short axisType[2]; // 查询二维补偿表所使用的X、Y方向位置类型, axisType[0]：X方向的位置类型，axisType[1]：Y方向的位置类型，取值为MC_PROFILE：规划位置，MC_ENCODER：编码器位置
	short axisIndex[2]; // 二维补偿X、Y方向运动所使用的轴号,axisIndex[0]：X方向的轴号，axisIndex[1]：Y方向的轴号
	} TCompensate2D;		

指令返回值	1：（1）补偿轴未使能（2）未使能补偿表索引或未设置补偿表及数据等于 0。
相关指令	无。
指令示例	无。

指令 166 GTN_SetCompensate2DTable

指令原型	GTN_SetCompensate2DTable(short core, short tableIndex, TCompensate2DTable *pTable, long *pData, short externComp=0)		
指令说明	设置二维补偿表及数据。		
指令类型	立即指令，调用后立即生效。	章节页码	154
指令参数	该指令共有 5 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
tableIndex	控制器内部的补偿表索引号，取值范围：[1，4]。		

pTable	设置二维补偿表及数据的参数，该参数为一结构体，详细参数定义及说明请参照结构体 TCompensate2DTable：
	typedef struct
	{
	short count[2]; // 补偿表的数据点数量，count[0]：X方向数据点数量count[1]：Y方向数据点数量，最小值为2。注意：count[0] * count[1]<=40000
	long posBegin[2];// 补偿区域起始点，posBegin[0]：X 方向位置, posBegin[1]：Y 方向位置
	long step[2]; // 补偿区域的步长(即补偿区域内，每两个补偿点间的距离，step[0]：X方向的间距，step[1]：Y方向的间距（根据补偿起点、补偿数据点和补偿间距，控制器自动计算出补偿区域）
	} TCompensate2DTable;		

pData	补偿数据，应该为一个二维数组，例如 pData[Y][X]，X、Y 分别为补偿表 X、Y 方向的数据点数量（**注意数组的行列**）。最大数据点数为 40000 即：X*Y<=40000。		
externComp	是否自动扩展补偿区域。 0：不生效，不自动扩展补偿区域，按照设置的补偿区域进行补偿； 非 0 值：生效，自动把设置的补偿区域向四周扩展 1 个步长，边界值补偿值为 0。 如果不自动扩展补偿区域边界，建议补偿区域的 X、Y 终止边界上的点的补偿值为 0； 如果不自动扩展补偿区域边界，且补偿轴处于静止状态，则第一个补偿点的补偿值不能超过 180（否则会由于速度突变太大而导致输出脉冲丢失，这个参数主要和驱动器的最大传输频率决定）。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 167 GTN_SetCompensate2DtableRotationAngle

指令原型	GTN_SetCompensate2DTableRotationAngle(short core,short tableIndex, short enable, double rotationAngle)		
指令说明	设置二维补偿表旋转参数。		
指令类型	立即指令，调用后立即生效。	章节页码	154
指令参数	该指令共有 4 个参数，参数的详细信息如下。		

core	内核，正整数，取值范围请参照表13-1中的“内核”一栏
tableIndex	控制器内部的补偿表索引号，取值范围：[1，4]。
enable	补偿表旋转功能使能，0：不使能，1：使能。
rotationAngle	补偿表以表起点位置相对于x坐标轴逆时针旋转角度，单位：度。
指令返回值	请参照指令返回值列表。
相关指令	无。
指令示例	无。

指令 168 GTN_SetControlFilter

指令原型	short GTN_SetControlFilter(short core, short control, short index)		
指令说明	设定 PID 索引，支持 3 组 PID 参数。		
指令类型	立即指令，调用后立即生效。	章节页码	146
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
control	伺服控制器编号，正整数，取值范围请参照表 13-1 中的“轴”一栏		
index	伺服控制参数的索引号。取值范围：[1, 3]。		
指令返回值	请参照指令返回值列表。		
相关指令	GTN_GetControlFilter		
指令示例	无。		

指令 169 GTN_SetCrdMapBase

指令原型	short GTN_SetCrdMapBase(short core,short crd,short base)		
指令说明	设置插补坐标系映射基础规划轴。该指令的功能在于使得 GTN_SetCrdPrm 建立坐标系时，能够支持第 8 个规划器以后的轴。		
指令类型	立即指令，调用后立即生效。	章节页码	99
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
crd	坐标系编号，正整数，取值范围请参照表 13-1 中的“坐标系”一栏		
base	基础规划轴，正整数，取值范围请参照表 13-1 中的“轴”一栏。 控制器默认为 1，即建立插补坐标系只支持前 8 个规划器。		
指令返回值	请参照指令返回值列表。		

相关指令	//以下例子用于把规划器~12作为插补坐标系轴
	rtn = GTN_SetCrdMapBase(1,1,9); //第个规划器作为插补坐标系映射基础轴
	TCrdPrm crdPrm;
	rtn = GTN_GetCrdPrm(1,1,&crdPrm);
	crdPrm.dimension = 4;
	crdPrm.profile[0]=1; //规划器作为插补坐标系的X轴
	crdPrm.profile[1]=2; //规划器作为插补坐标系的Y轴
	crdPrm.profile[2]=3; //规划器作为插补坐标系的Z轴
	crdPrm.profile[3]=4; //规划器作为插补坐标系的A轴
	rtn = GTN_SetCrdPrm(1,1,&crdPrm);		

指令示例			

指令 170 GTN_SetCrdMPGMode

指令原型	short GTN_SetCrdMPGMode(short core,short crd,short enable,short master,long masterEven,long slaveEven,short filterTime,short mode)		
指令说明	设置、开启插补模式下的手轮引导功能		
指令类型	立即指令，调用后立即生效。	章节页码	55
指令参数	该指令共有 8 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
crd	坐标系索引，正整数，取值范围请参照表 13-1 中的“插补坐标系序号”一栏		
enable	1：开启手轮引导功能；0：关闭手轮引导功能		
master	主动轴编码器，正整数，取值范围请参照表 13-1 中的“手轮”一栏。		
masterEven	传动比系数，主轴位移。 正整数，单位：pulse。		
slaveEven	传动比系数，从轴位移。 正整数，单位：pulse。		
filterTime	主动轴编码器滤波时间常数，正整数，单位：ms		
mode	手轮引导模式，0：静态模式，2：动态模式 静态模式：最多只能压入 4096 条插补指令，且可以后退到第一条插补指令，可以前进到最后一条插补指令。 动态模式：无指令数量限制，但最多只能回退 2048 段（如果执行段数不足 2048，按实际段数回退）。		
指令返回值	1：插补坐标系未停止 7：参数取值超过范围限制		
相关指令			
指令示例	**参考 12.12.2 例程**		

指令 171 GTN_SetCrdPrm

指令原型	short GTN_SetCrdPrm(short core, short crd, TCrdPrm *pCrdPrm)		
指令说明	设置坐标系参数，确立坐标系映射，建立坐标系。		
指令类型	立即指令，调用后立即生效。	章节页码	65
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
crd	坐标系号，正整数，取值范围请参照表 13-1 中的“插补坐标系序号”一栏		
pCrdPrm	设置坐标系的相关参数。 typedef struct CrdPrm { short dimension; short profile[8]; double synVelMax; double synAccMax; short evenTime; short setOriginFlag; long originPos[8]; }TCrdPrm; **dimension**：坐标系的维数。取值范围：[1, 8]。 **Profile[8]**：坐标系与规划器的映射关系。Profile[0..7]对应规划轴 1~8，如果规划轴没有对		

	应到该坐标系，则 profile[x]的值为 0；如果对应到了 X 轴，则 profile[x]为 1，Y 轴对应为 2，Z 轴对应为 3，A 轴对应为 4。不允许多个规划轴映射到相同坐标系的相同坐标轴，也不允许把相同规划轴对应到不同的坐标系，否则该指令将会返回错误值。每个元素的取值范围：[0, 4]。 **synVelMax**：该坐标系的最大合成速度。如果用户在输入插补段的时候所设置的目标速度大于了该速度，则将会被限制为该速度。取值范围：(0, 65536]。单位：pulse/ms。 **synAccMax**：该坐标系的最大合成加速度。如果用户在输入插补段的时候所设置的加速度大于了该加速度，则将会被限制为该加速度。取值范围：(0, 32767]。单位：pulse/ms^2。 **evenTime**：每个插补段的最小匀速段时间。取值范围：[0, 32767]。单位：ms。 **setOriginFlag**：表示是否需要指定坐标系的原点坐标的规划位置，该参数可以方便用户建立区别于机床坐标系的加工坐标系。0：不需要指定原点坐标值，则坐标系的原点在当前规划位置上。1：需要指定原点坐标值，坐标系的原点在 originPos 指定的规划位置上。 **originPos[8]**：指定的坐标系原点的规划位置值。
指令返回值	若返回值为 1： (1) 若坐标系下各轴在规划运动，请调用 GTN_Stop 停止运动再调用该指令。 (2) 请检查映射到 Profile 有没被激活，若无，则返回错误。 (3) 请见检查相应轴是否在坐标系下。 其他返回值：请参照指令返回值列表。
相关指令	GTN_GetCrdPrm
指令示例	**例程 7-4 建立坐标系**

指令 172 GTN_SetCrdStopDec

指令原型	short GTN_SetCrdStopDec (short core, short crd, double decSmoothStop, double decAbruptStop)		
指令说明	设置插补运动平滑停止、急停合成加速度		
指令类型	立即指令，调用后立即生效。	**章节页码**	65
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
crd	坐标系号。正整数，取值范围请参照表 13-1 中的“插补坐标系序号”一栏		
decSmoothStop	设置的坐标系合成平滑停止加速度。取值范围：(0, 32767)，单位：pulse/ms^2。		
decAbruptStop	设置的坐标系合成急停加速度。取值范围：(0, 32767)，单位：pulse/ms^2。		
指令返回值	若返回值为 1：检查当前坐标系是否映射了相关轴。 其他返回值：请参照指令返回值列表。		
相关指令	GTN_GetCrdStopDec		
指令示例	无。		

指令 173 GTN_SetDac

指令原型	short GTN_SetDac(short core, short dac, short *pValue, short count)		
指令说明	设置 dac 输出电压。当闭环模式下，da 输出通道与轴挂接时，用户不能调用该指令直接输出电压。		
指令类型	立即指令，调用后立即生效。	**章节页码**	100
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
dac	dac 起始轴号。正整数，取值范围请参照表 13-1 中的“轴”一栏		

pValue	输出电压。8 路轴控接口 -32768 对应-10V，32767 对应+10V。
count	设置的通道数。默认为 1。 正整数，取值范围请参照表 13-1 中的“参数个数”一栏
指令返回值	请参照指令返回值列表。
相关指令	GTN_GetDac
指令示例	**例程 8-3 访问 DAC**

指令 174 GTN_SetDiReverseCount

指令原型	short GTN_SetDiReverseCount(short core, short diType, short diIndex, unsigned long *pReverseCount, short count)		
指令说明	设置数字量输入信号的变化次数的初值。		
指令类型	立即指令，调用后立即生效。	章节页码	94
指令参数	该指令共有 5 个参数，参数的详细信息如下：		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
diType	指定数字 IO 类型。 MC_LIMIT_POSITIVE(该宏定义为 0)：正限位。 MC_LIMIT_NEGATIVE(该宏定义为 1)：负限位。 MC_ALARM(该宏定义为 2)：驱动报警。 MC_HOME(该宏定义为 3)：原点开关。 MC_GPI(该宏定义为 4)：通用输入。 MC_ARRIVE(该宏定义为 5)：电机到位信号。		
diIndex	数字量输入的索引。 diType= MC_LIMIT_POSITIVE、MC_LIMIT_NEGATIVE、MC_ALARM、MC_HOME 、MC_ARRIVE 时 正整数，取值范围请参照表 13-1 中的“轴”一栏 diType= MC_GPI 时：正整数，取值范围请参照表 13-1 中的“通用输入”一栏。		
pReverseCount	设置的数字量输入的变化次数的初值。		
count	设置变化次数初值的数字量输入的个数。默认为 1。 1 次最多可以设置 4 个数字量输入的变化次数的初值。		
指令返回值	请参照指令返回值列表。		
相关指令	GTN_GetDiReverseCount		
指令示例	**例程 8-1 访问数字 IO**		

指令 175 GTN_SetDo

指令原型	short GTN_SetDo(short core, short doType, long value)		
指令说明	设置数字 IO 输出状态。若 do 有挂接轴，则对应的不能直接输出。默认驱动器使能与轴挂接，所以用户不能调用该指令设置驱动器使能输出的电平。		
指令类型	立即指令，调用后立即生效。	章节页码	94
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“轴”一栏		
doType	指定数字 IO 类型。 MC_ENABLE(该宏定义为 10)：驱动器使能。 MC_CLEAR(该宏定义为 11)：报警清除。 MC_GPO(该宏定义为 12)：通用输出。		

value	按位指示数字 IO 输出电平。 默认情况下，1 表示高电平，0 表示低电平。
指令返回值	请参照指令返回值列表。
相关指令	无。
指令示例	**例程 8-1 访问数字 IO**

指令 176 GTN_SetDoBit

指令原型	short GTN_SetDoBit(short core, short doType, short doIndex, short value)		
指令说明	按位设置数字 IO 输出状态		
指令类型	立即指令，调用后立即生效。	章节页码	94
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“轴”一栏		
doType	指定数字 IO 类型 MC_ENABLE(该宏定义为 10)：驱动器使能。 MC_CLEAR(该宏定义为 11)：报警清除。 MC_GPO(该宏定义为 12)：通用输出。		
doIndex	输出 IO 的索引。 取值范围： doType=MC_ENABLE 时： doType=MC_CLEAR 时 正整数，取值范围请参照表 13-1 中的“轴”一栏 doType=MC_GPO 时 正整数，取值范围请参照表 13-1 中的“通用输出”一栏		
value	设置数字 IO 输出电平。 默认情况下，1 表示高电平，0 表示低电平。		
指令返回值	若返回值为 1：检查设置输出的 bit 是否挂接轴，若挂接，则不能直接输出。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	**例程 8-1 访问数字 IO**		

指令 177 GTN_SetDoBitReverse

指令原型	short GTN_SetDoBitReverse (short core, short doType, short doIndex, short value, short reverseTime)		
指令说明	使数字量输出信号输出定时脉冲信号。		
指令类型	立即指令，调用后立即生效。	章节页码	94
指令参数	该指令共有 5 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“轴”一栏		
doType	指定数字 IO 类型。 MC_ENABLE(该宏定义为 10)：驱动器使能。 MC_CLEAR(该宏定义为 11)：报警清除。 MC_GPO(该宏定义为 12)：通用输出。		
doIndex	输出 IO 的索引。 doType=MC_ENABLE 时：[1, 12]。 doType=MC_CLEAR 时：[1, 12]。		

	正整数，取值范围请参照表 13-1 中的“轴”一栏 doType=MC_GPO 时 正整数，取值范围请参照表 13-1 中的“通用输出”一栏
value	设置数字 IO 输出电平。 默认情况下，1 表示高电平，0 表示低电平。
reverseTime	维持 value 所设置电平的时间，取值范围：（0, 32767]，单位：中断周期。
指令返回值	若返回值为 1：检查设置输出的 bit 是否挂接轴，若挂接，则不能直接输出。 其他返回值：请参照指令返回值列表。
相关指令	无。
指令示例	无。

指令 178 GTN_SetDoEx

指令原型	short GTN_SetDoEx(short core,short doType,long *pValue,short arraySize)		
指令说明	设置数字 IO 输出状态。若 do 有挂接轴，则对应的不能直接输出。默认驱动器使能与轴挂接，所以用户不能调用该指令设置驱动器使能输出的电平。		
指令类型	立即指令，调用后立即生效。	章节页码	94
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“轴”一栏		
doType	指定数字 IO 类型。 MC_ENABLE(该宏定义为 10)：驱动器使能。 MC_CLEAR(该宏定义为 11)：报警清除。 MC_GPO(该宏定义为 12)：通用输出。		
pValue	按位指示数字 IO 输出电平。 默认情况下，1 表示高电平，0 表示低电平。		
arraySize	数组大小，对应 pValue 变量个数。按照 32 路为一组，因此总 DO 个数=32*arraySize。正整数，取值范围请参照表 13-1 中的“参数个数”一栏		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	**例程 8-1 访问数字 IO**		

指令 179 GTN_SetEncPos

指令原型	short GTN_SetEncPos (short core, short encoder, long encPos)		
指令说明	修改编码器位置。		
指令类型	立即指令，调用后立即生效。	章节页码	99
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
encoder	编码器起始轴号，正整数，取值范围请参照表 13-1 中的“轴”一栏		
encPos	编码器位置。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 180 GTN_SetG0Mode

指令原型	short GTN_SetG0Mode(short core,short crd,short mode)

指令说明	设置插补 G0 指令模式。		
指令类型	立即指令。	章节页码	
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
crd	坐标系号。正整数，取值范围请参照表 13-1 中的“插补坐标系号”一栏		
mode	插补 G0 指令模式。 0：按照坐标系规划定位指令，定位轨迹为直线； 1：按照单轴规划定位指令，各个轴单独规划，加速度与速度相同，距离短的轴先到位，距离长的轴后到位，定位轨迹可能为折线。		
返回值			
相关指令			
指令示例			

指令 181 GTN_SetGearMaster

指令原型	short GTN_SetGearMaster(short core, short profile, short masterIndex, short masterType, short masterItem)		
指令说明	设置电子齿轮运动跟随主轴。		
指令类型	立即指令，调用后立即生效。	章节页码	61
指令参数	该指令共有 5 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
profile	规划轴号，正整数，取值范围请参照表 13-1 中的“轴”一栏		
masterIndex	主轴索引，正整数。 主轴索引不能与规划轴号相同，最好主轴索引号小于规划轴号，如主轴索引为 1 轴，规划轴号为 2 轴。		
masterType	主轴类型。 GEAR_MASTER_ENCODER (1) :表示跟随编码器(encoder)的输出值。 GEAR_MASTER_PROFILE (2): 表示跟随规划轴(profile)的输出值。默认为该类型。 GEAR_MASTER_AXIS (3): 表示跟随轴(axis)的输出值。 GEAR_MASTER_AU_ENCODER (4): 表示跟随辅助编码器的输出值。 GEAR_MASTER_MPG_ENCODER (5): 表示跟随 MPG 的输出值。 GEAR_MASTER_ENCODER_OTHER (101): 表示 core2 轴跟随 core1 中编码器(encoder)的输出值 GEAR_MASTER_AXIS_OTHER (103): 表示 core2 轴跟随 core1 中轴(axis)的输出值 GEAR_MASTER _AU_ENCODER_OTHER (104): 表示 core2 轴跟随 core1 中辅助编码器的输出值 GEAR_MASTER_MPG_ENCODER_OTHER (105): 表示 core2 轴跟随 core1 中 MPG 的输出值。		
masterItem	轴类型，当 masterType=GEAR_MASTER_AXIS 时起作用。 0 表示 axis 的规划位置输出值。默认为该值。 非 0 值表示 axis 的编码器位置输出值。		
指令返回值	若返回值为 1： (1) 若当前轴在规划运动，请调用 GTN_Stop 停止运动再调用该指令。 (2) 请检查当前轴是否为电子齿轮模式，若不是，请先调用 GTN_PrfGear 将当前轴设置为电子齿轮模式。 其他返回值：请参照指令返回值列表。		

相关指令	GTN_GetGearMaster
指令示例	**例程 7-3 电子齿轮跟随**

指令 182 GTN_SetGearRatio

指令原型	short GTN_SetGearRatio(short core, short profile, long masterEven, long slaveEven, long masterSlope)		
指令说明	设置电子齿轮比。		
指令类型	立即指令，调用后立即生效。	章节页码	61
指令参数	该指令共有 5 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
profile	规划轴号，正整数，取值范围请参照表 13-1 中的“轴”一栏		
masterEven	传动比系数，主轴位移。 正整数，单位：pulse。		
slaveEven	传动比系数，从轴位移。 单位：pulse。		
masterSlope	主轴离合区位移。 单位：pulse。取值范围：不能小于 0 或者等于 1。		
指令返回值	若返回值为 1：请检查当前轴是否为电子齿轮模式，若不是，请先调用 GTN_PrfGear 将当前轴设置为电子齿轮模式。 其他返回值：请参照指令返回值列表。		
相关指令	GTN_GetGearRatio		
指令示例	**例程 7-3 电子齿轮跟随**		

指令 183 GTN_SetJogPrm

指令原型	short GTN_SetJogPrm(short core, short profile, TJogPrm *pPrm)		
指令说明	设置 Jog 运动模式下的运动参数。		
指令类型	立即指令，调用后立即生效。	章节页码	57
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
profile	规划轴号，正整数，取值范围请参照表 13-1 中的“轴”一栏		
pPrm	设置 Jog 模式运动参数。该参数为一个结构体，包含三个参数，详细的参数定义及说明如下： typedef struct JogPrm { double acc; double dec; double smooth; } TJogPrm; **acc:**点位运动的加速度。正数，单位：pulse/ms^2。 **dec:**点位运动的减速度。正数，单位：pulse/ms^2。未设置减速度时，默认减速度和加速度相同。 **smooth：**平滑系数。取值范围：[0, 1)。平滑系数的数值越大，加减速过程越平稳。		
指令返回值	若返回值为 1： (1) 若当前轴在规划运动，请调用 GTN_Stop 停止运动再调用该指令。		

	(2) 请检查当前轴是否为 Jog 模式，若不是，请先调用 GTN_PrfJog 将当前轴设置为 Jog 模式。 其他返回值：请参照指令返回值列表。
相关指令	GTN_GetJogPrm
指令示例	**例程 7-2 Jog 运动**

指令 184 GTN_SetLeadScrewComp

指令原型	GTN_SetLeadScrewComp(short core,short axis,short n,long startPos,long lenPos,long *pPositive,long *pNegative)		
指令说明	加载补偿表		
指令类型	立即指令，调用后立即生效。	章节页码	147
指令参数	该指令共有 7 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
axis	补偿轴轴号,正整数，取值范围请参照表 13-1 中的“轴”一栏		
n	补偿段数		
startPos	补偿起始点的规划位置，单位：pulse。		
lenPos	补偿区域总长，单位：pulse。补偿起始点和补偿总长一起构成了补偿区域，超出该规划位置范围，补偿无效。补偿区域总长不能为负数。（每个补偿区间长度为：lenPos/(n-1)）		
pCompPos	正向补偿表数组地址		
pCompNeg	负向补偿表数组地址		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	**例程 12-3 螺距误差补偿例程**		

指令 185 GTN_SetMtrBias

指令原型	short GTN_SetMtrBias(short core, short dac, short bias)		
指令说明	设置模拟量输出通道的零漂电压补偿值。		
指令类型	立即指令，调用后立即生效。	章节页码	41
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
dac	模拟量输出通道号，正整数，取值范围请参照表13-1中的“轴”一栏		
bias	零漂补偿值，取值范围：[-32768, 32767]。		
指令返回值	请参照指令返回值列表。		
相关指令	GTN_GetMtrBias		
指令示例	无。		

指令 186 GTN_SetMtrLmt

指令原型	short GTN_SetMtrLmt(short core, short dac, short limit)		
指令说明	设置模拟量输出通道的输出电压饱和极限值。		
指令类型	立即指令，调用后立即生效。	章节页码	41
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
dac	模拟量输出通道号，正整数，取值范围请参照表13-1中的“轴”一栏		

limit	输出电压饱和极限值，取值范围：(0, 32767]。若设置为32767，则限制允许输出的电压范围为：-10V~+10V；若设置为 16384，则限制允许输出的电压范围为：-5V~+5V。
指令返回值	请参照指令返回值列表。
相关指令	GTN_GetMtrLmt
指令示例	无。

指令 187 GTN_SetOverride

指令原型	short GTN_SetOverride (short core, short crd, double synVelRatio)		
指令说明	设置插补运动目标合成速度倍率。		
指令类型	立即指令，调用后立即生效。	章节页码	65
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
crd	坐标系号。正整数，取值范围请参照表 13-1 中的“插补坐标系序号”一栏		
synVelRatio	设置的插补目标速度倍率，取值范围：[0, 10]，系统默认该值为：1。		
指令返回值	若返回值为 1：检查当前坐标系是否映射了相关轴。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 188 GTN_SetPid

指令原型	short GTN_SetPid(short core, short control, short index, TPid *pPid)		
指令说明	设置 PID 参数。		
指令类型	立即指令，调用后立即生效。	章节页码	146
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
control	伺服控制器编号，正整数，取值范围请参照表 13-1 中的“伺服控制器”一栏		
index	伺服控制参数的索引号，取值范围：[1, 3]。		
pPid	设置 PID 参数 typedef struct Pid { double kp; double ki; double kd; double kvff; double kaff; long integralLimit; long derivativeLimit; short limit; }TPid; **kp**：比例增益，该值取值非负数； **ki**：积分增益，该值取值非负数； **kd**：微分增益，该值取值非负数； **kvff**：速度前馈系数，该值取值非负数； **kaff**：加速度前馈系数，该值取值非负数；		

	integralLimit：积分饱和极限，该值取值非负数； **derivativeLimit**：微分饱和极限，该值取值非负数； **limit**：控制量输出饱和极限，该值需要大于 0；
指令返回值	请参照指令返回值列表。
相关指令	GTN_GetPid
指令示例	**例程 5-3 设置第 1 轴为闭环控制方式**

指令 189 GTN_SetPlsPos

指令原型	short GTN_SetPlsPos(short core,short encoder,long encPos)		
指令说明	设置内部脉冲计数器位置。		
指令类型	立即指令，调用后立即生效。	章节页码	102
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
encoder	脉冲计数器对应轴序号，正整数，取值范围请参照表 13-1 中的“轴”一栏		
encPos	内部脉冲计数器位置		
指令返回值	请参照指令返回值		
相关指令	无。		
指令示例	**无**		

指令 190 GTN_SetPos

指令原型	short GTN_SetPos(short core, short profile, long pos)		
指令说明	设置目标位置。		
指令类型	立即指令，调用后立即生效。	章节页码	54
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
profile	规划轴号，正整数，取值范围请参照表 13-1 中的“轴”一栏		
pos	设置目标位置，单位：pulse。取值范围：[-1073741824, 1073741823]。		
指令返回值	若返回值为 1：请检查当前轴是否为 Trap 模式，若不是，请先调用 GTN_PrfTrap 将当前轴设置为 Trap 模式。 其他返回值：请参照指令返回值列表。		
相关指令	GTN_GetPos		
指令示例	**例程 7-1 点位运动**		

指令 191 GTN_SetPosCompareFifoMode

指令原型	short GTN_SetPosCompareFifoMode(short core,short index,short mode)		
指令说明	设置存储实际位置比较点的缓冲区模式（大小 2048）。		
指令类型	立即指令，调用后立即生效。	章节页码	99
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏 **核 2 目前不支持位置比较功能**		
index	位置比较模块编号，正整数，取值范围请参照表 13-1 中的“轴”一栏		

mode	缓冲区模式。
	#define POS_COMPARE_RESULT_FIFO_MODE_STATIC (0) //静态模式
	#define POS_COMPARE_RESULT_FIFO_MODE_LOOP (1) //动态模式		

	静态模式：位置比较点存满整个缓冲区后，停止继续保存新的位置比较点。 动态模式：位置比较点存满整个缓冲区后，覆盖最先进入缓冲区的数据，继续保存。 两种模式，都需要及时读取，否则可能会丢失数据。
指令返回值	请参照指令返回值列表。
相关指令	
指令示例	

指令 192 GTN_SetPosCompareLinear

指令原型	short GTN_SetPosCompareLinear(short core,short posCompareIndex, TPosCompareLinear *pLinear);		
指令说明	设置一维线性比较输出参数		
指令类型	缓存区指令	章节页码	114
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏 **核 2 目前不支持位置比较功能**		
posCompareIndex	缓存区 FIFO，正整数，取值范围请参照表 13-1 中的“位置比较输出”一栏		

pLinear	typedefstruct
	{
	unsigned long count; //位置比较输出位置个数
	unsigned short hso; //位置比较输出hso通道的输出数值，按位表示
	//脉冲模式：0 表示无输出，1 表示输出脉冲
	unsigned short gpo; //通用GPO通道的输出数值，按位表示
	//脉冲模式：0 表示无输出，1 表示输出脉冲
	long startPos; //线性比较输出的起点位置
	long interval; //位置比较输出位置间隔
	} TPosCompareLinear;		

指令返回值	请参照指令返回值列表。 注意：线性模式的输出电平按照 pLinear 中的输出值输出，不会反转。建议使用脉冲模式		
相关指令	无。		
指令示例	**无。**		

指令 193 GTN_SetPosCompareMode

指令原型	short GTN_SetPosCompareMode(short core, short posCompareIndex, TPosCompareMode *pMode);		
指令说明	设置位置比较输出模式		
指令类型	缓存区指令	章节页码	114
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏 **核 2 目前不支持位置比较功能**		
posCompareIndex	位置比较索引，正整数，取值范围[1,8]。		
pMode	读取位置比较输出模式 typedef struct		

```
{
    short mode;                    // 0：FIFO 模式，1：liner 模式，2：PSO 立即
                                      模式，3：PSO 等待到位触发模式
    short dimension;               //1：1D，2：2D
    short sourceMode;              // 0：编码器；  1：脉冲计数器
    short sourceX;                 // X 轴比较源[1,12]
    short sourceY;                 // Y 轴比较源[1,12]
    short outputMode;              //输出模式：0:脉冲  1:电平 2：电平自动翻转
    short outputCounter;           //  保留
    unsigned short outputPulseWidth; //输出脉冲宽度,单位为 1us，电平模式该参数无效
    unsigned short errorBand;      //  二维位置比较输出误差带
} TPosCompareMode;
```

- 对于PSO立即模式（mode=2），PSO的开关立即执行；PSO等待到位触发模式（mode=3），PSO的开关会根据设置的误差带，以及当前的编码器/脉冲计数器是否到位来决定是否执行PSO开关操作。对于PSO立即模式，PSO开关立即执行，对于伺服滞后系统，电机还没有到位就执行PSO开关，会影响加工效果；对于PSO等待到位触发模式，能保证电机到位才执行PSO开关，能给保证加工效果。
- fifo模式下：输出模式可选为：脉冲、电平
- liner模式下：输出模式可选为：脉冲、电平自动翻转
- PSO立即模式下：输出模式可选为：脉冲
- PSO等待到位触发模式：输出模式可选为：脉冲

指令返回值	请参照指令返回值列表。 注意事项：比较源的方向在该指令后修改无效。、
相关指令	GTN_GetPosCompareMode
指令示例	**例程 9-4 设置位置比较输出通道 1 为一维 FIFO 模式**

指令194 GTN_SetPosComparePsoPrm

指令原型	shortGTN_SetPosComparePsoPrm(short core, short posCompareIndex, TPosComparePsoPrm *pPrm)
指令说明	设置位置同步比较输出
指令类型	立即指令 章节页码
指令参数	该指令共有 3 个参数，参数的详细信息如下。
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏 **核 2 目前不支持位置比较功能**
posCompareIndex	缓存区 FIFO，正整数，取值范围请参照表 13-1 中的“位置比较输出”一栏

pPrm	typedef struct
	{
	unsigned long count; //保留
	unsigned short hso; //保留
	unsigned short gpo; //保留
	long startPosX; //保留

	long startPosY; //保留
	long syncPos; //同步位置增量
	long time; //保留
	short reserve[20]; //保留
	} TPosComparePsoPrm;

指令返回值	请参照指令返回值
相关指令	无。
指令示例	**无。**

指令 195 GTN_SetPosCompareStartLevel

指令原型	short GTN_SetPosCompareStartLevel(short core, short posCompareIndex, short type, short startLevel = 0);		
指令说明	设置位置比较输出模式的初始输出电平		
指令类型	立即指令	章节页码	114
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
posCompareIndex	缓存区 FIFO，正整数，取值范围请参照表 13-1 中的“位置比较输出”一栏		
type	位置比较输出类型，MC_GPO 和 MC_HSO		
startLevel	位置比较输出的初始电平， 0：和硬件初始电平一致，默认为高电平 1：和硬件初始电平相反		
指令返回值	请参照指令返回值列表。 注意事项：控制权为位置比较输出才能生效		
相关指令			
指令示例			

指令 196 GTN_SetPosErr

指令原型	short GTN_SetPosErr(short core, short control, long error)		
指令说明	设置跟随误差极限值。		
指令类型	立即指令，调用后立即生效。	章节页码	41
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
control	伺服控制器编号，正整数，取值范围请参照表13-1中的“伺服控制器”一栏		
error	跟随误差极限值，取值范围：(0, 2147483647]。单位：pulse。		
指令返回值	请参照指令返回值列表。		
相关指令	GTN_GetPosErr		
指令示例	无。		

指令 197 GTN_SetPrfPos

指令原型	short GTN_SetPrfPos(short core, short profile, long prfPos)		
指令说明	修改指定轴的规划位置。禁止在运动状态下修改规划位置。		
指令类型	立即指令，调用后立即生效。	章节页码	146
指令参数	该指令共有 3 个参数，参数的详细信息如下：		

core	内核，正整数，取值范围请参照表13-1中的“内核”一栏
profile	规划轴编号，正整数，取值范围请参照表 13-1 中的“轴”一栏
prfPos	设置的规划位置的值。
指令返回值	请参照指令返回值列表。
相关指令	GTN_GetPrfPos
指令示例	**例程 7-1 点位运动**

指令 198 GTN_SetResCount

指令原型	short GTN_SetResCount(short core,short type,short count)		
指令说明	设置主卡资源个数		
指令类型	立即指令，调用后立即生效。	**章节页码**	20
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
type	资源类型 MC_LIMIT_POSITIVE // 正限位 MC_LIMIT_NEGATIVE // 负限位 MC_ALARM // 驱动报警 MC_HOME // 回零 MC_GPI // 通用输入 MC_ARRIVE // 电机到位 MC_MPG // 手轮倍率输入 MC_ENABLE // 使能 MC_CLEAR // 报警清除 MC_GPO // 通用输出 MC_AU_ADC // 非轴模拟量输入 MC_DAC // 模拟量输出 MC_STEP // 脉冲输出 MC_PULSE // 内部脉冲计数器 MC_ENCODER // 编码器 MC_ADC // 模拟量输入 MC_AXIS // 轴资源 MC_PROFILE // 规划器 MC_CONTROL // 闭环资源 MC_TRIGGER // trigger捕获 MC_SCAN_CRD // 振镜 MC_POS_COMPARE // 位置比较输出 MC_AU_ENCODER_EX // 辅助编码器 MC_MPG_ENCODER // 手轮对应的编码器		
count	设置资源个数，正整数，请参考表4-1		
指令返回值	请参照第 3 章 指令返回值及其意义。		
相关指令	GTN_GetResCount		
指令示例	**例程 4-1 设置主卡软件资源**		

指令 199 GTN_SetRetainValue

指令原型	short GTN_SetRetainValue(short core,unsigned long address,short count,short *pData)		
指令说明	保存数据到 MRAM 存储芯片		
指令类型	立即指令，调用后立即生效。	章节页码	129
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
address	起始地址，正整数， 当 count=1,取值范围为[0,16383] ... 当 count=16,取值范围为[0,16368] 注意：address+count 小于等于 16384		
count	保存数据个数，正整数，取值范围[1,16]		
pData	保存数据的数值		
指令返回值	请参照《GTN 系列运动控制器之基本功能》第 3 章 指令返回值及其意义。		
相关指令	GTN_GetRetainValue		
指令示例	**例程 10-2 掉电存储操作**		

指令 200 GTN_SetSense

指令原型	short GTN_SetSense(short core,short dataType,short dataIndex,short value)		
指令说明	设置输入输出资源的电平逻辑。		
指令类型	立即指令，调用后立即生效。	章节页码	94
指令参数	该指令共有 4 个参数，参数的详细信息如下：		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
dataType	输入输出资源类型 MC_ENCODE: (该宏定义为 23)：编码器。 MC_LIMIT_POSITIVE(该宏定义为 0)：正限位。 MC_LIMIT_NEGATIVE(该宏定义为 1)：负限位。 MC_ALARM(该宏定义为 2)：驱动报警。 MC_HOME(该宏定义为 3)：原点开关。 MC_GPI(该宏定义为 4)：通用输入。 MC_GPO(该宏定义为 12)：通用输出。 MC_ENABLE(该宏定义为 10)：使能信号。 MC_CLEAR(该宏定义为 11)：报警清除信号。		
dataIndex	资源序号，		
value	0：信号不取反， 1：信号取反。		
指令返回值	请参照指令返回值列表。		
相关指令	GTN_GetSense		
指令示例	无。		

指令 201 GTN_SetSoftLimitEx

指令原型	short GTN_SetSoftLimitEx (short core, short axis, double positive, double negative)		
指令说明	设置轴正向软限位和负向软限位。		
指令类型	立即指令，调用后立即生效。	章节页码	126
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		

axis	轴号，正整数。正整数，取值范围请参照表 13-1 中的“轴”一栏
positive	正向软限位，当规划位置大于该值时，正限位触发。单位：脉冲
negative	负向软限位，当规划位置小于该值时，负限位触发。单位：脉冲
指令返回值	请参照指令返回值列表。
相关指令	GTN_GetSoftLimitEx
指令示例	**例程 10-1　　软限位使用**

指令 202 GTN_SetSoftLimitMode

指令原型	short GTN_SetSoftLimitMode(short core,short axis,short mode)		
指令说明	设置软限位模式		
指令类型	立即指令，调用后立即生效。	**章节页码**	126
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
axis	轴号,正整数，取值范围请参照表 13-1 中的“轴”一栏		
mode	0 SOFT_LIMIT_MODE_STOP　//超越软限位位置后开始减速停止 1 SOFT_LIMIT_MODE_LIMIT　//限制在软限位范围之内		
指令返回值	请参照指令返回值列表。		
相关指令	GTN_GetSoftLimitMode		
指令示例	**无**		

指令 203 GTN_SetStopDec

指令原型	short GTN_SetStopDec(short core, short profile, double decSmoothStop, double decAbruptStop)		
指令说明	设置平滑停止减速度和急停减速度。		
指令类型	立即指令，调用后立即生效。	**章节页码**	41
指令类型	立即指令，调用后立即生效。		
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
profile	规划器的编号，正整数，取值范围请参照表13-1中的“轴”一栏		
decSmoothStop	平滑停止减速度，取值范围：(0, 32767]。单位：pulse/ms^2。		
decAbruptStop	急停减速度，取值范围：(0, 32767]。单位：pulse/ms^2。		
指令返回值	请参照指令返回值列表。		
相关指令	GTN_GetStopDec		
指令示例	无。		

指令 204 GTN_SetStopIo

指令原型	short GTN_SetStopIo(short core, short axis, short stopType, short inputType, short inputIndex)		
指令说明	设置平滑停止和紧急停止数字量输入的信息。		
指令类型	立即指令，调用后立即生效。	**章节页码**	41
指令参数	该指令共有 5 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
axis	需要设置停止IO信息的轴的编号，正整数，取值范围请参照表13-1中的“轴”一栏		
stopType	需要设置停止 IO 信息的停止类型。		

	0：紧急停止类型。 1：平滑停止类型。
inputType	设置的数字量输入的类型。 MC_LIMIT_POSITIVE(该宏定义为 0)，正限位。 MC_LIMIT_NEGATIVE(该宏定义为 1)，负限位。 MC_ALARM(该宏定义为 2)，驱动报警。 MC_HOME(该宏定义为 3)，原点开关。 MC_GPI(该宏定义为 4)，通用输入。 MC_ARRIVE(该宏定义为 5)，电机到位信号。
inputIndex	设置的数字量输入的索引号，取值范围根据 inputType 的取值而定。 当 inputType= MC_LIMIT_POSITIVE 时， 当 inputType= MC_LIMIT_NEGATIVE 时， 当 inputType= MC_ALARM 时， 当 inputType= MC_HOME 时， 正整数，取值范围请参照表 13-1 中的“轴”一栏 当 inputType= MC_GPI 时， 正整数，取值范围请参照表 13-1 中的“通用输入”一栏
指令返回值	请参照指令返回值列表。
相关指令	无。
指令示例	无。

指令 205 GTN_SetTerminalPermitEx

指令原型	short GTN_SetTerminalPermitEx(short core, short station, short dataType, short *permit, short index=1, short count=1)
指令说明	设置硬件通道输出信号类型
指令类型	立即指令，调用后立即生效。 **章节页码**
指令参数	该指令共有 6 个参数，参数的详细信息如下。
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏
station	模块序号，正整数，取值范围请参照表 13-1 中的“网络端子板模块”一栏
dataType	设置输出功能类型的数据类型： MC_GPO（12）：通用数字量输出，对应硬件的 DO（参见硬件手册） MC_HSO（18）：高速IO输出，对应硬件的HSO（参见硬件手册）
permit	按位设置硬件输出通道信号输出的类型 从 bit0-bit15 按位表示对应信号类型输出，1：控制权打开；0：控制权关闭 Bit0: 通用 IO 指令输出（当 dataType 为 MC_HSO 时，该值无效） Bit1: 第一路位置比较输出 Bit2: 第二路位置比较输出 Bit3: 使能激光开关光输出（当 dataType 为 MC_GPO 时，该值无效） Bit4: 使能 PWM 信号输出（当 dataType 为 MC_GPO 时，该值无效） Bit5-bit15 对于 403 模块保留
index	需要设置控制权的起始硬件通道序号，默认值为 1
count	需要设置控制权的硬件通道个数，默认值为 1,最大值为 20
指令返回值	请参照《GTN 系列运动控制器编程手册之基本功能》中的第 3 章。
相关指令	
指令示例	

指令 206 GTN_SetTrapPrm

指令原型	short GTN_SetTrapPrm(short core, short profile, TTrapPrm *pPrm)		
指令说明	设置点位模式运动下的运动参数。		
指令类型	立即指令，调用后立即生效。	章节页码	54
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
profile	规划轴号，正整数，取值范围请参照表 13-1 中的“轴”一栏		
pPrm	设置点位运动模式运动参数，该参数为一个结构体，包含四个参数，详细的参数定义及说明如下： typedef struct TrapPrm { double acc; double dec; double velStart; short smoothTime; }TTrapPrm; **acc:**点位运动的加速度。正数，单位：pulse/ms^2。 **dec:**点位运动的减速度。正数，单位：pulse/ms^2。未设置减速度时，默认减速度和加速度相同。 **velStart：**起跳速度。正数，单位：pulse/ms。默认值为 0。 **smoothTime：**平滑时间。正整数，单位 ms。平滑时间的数值越大，加减速过程越平稳。平滑时间取值范围根据控制周期变化，例如： 250us 控制周期，平滑时间取值范围为：[0, 50]，单位 ms。 500us 控制周期，平滑时间取值范围为：[0, 100]，单位 ms。 1ms 控制周期，平滑时间取值范围为：[0, 200]，单位 ms。		
指令返回值	若返回值为 1： (1) 若当前轴在规划运动，请调用 GTN_Stop 停止运动再调用该指令。 (2) 请检查当前轴是否为 Trap 模式，若不是，请先调用 GTN_PrfTrap 将当前轴设置为 Trap 模式。 其他返回值：请参照指令返回值列表。		
相关指令	GTN_GetTrapPrm		
指令示例	**例程 7-1 点位运动**		

指令 207 GTN_SetTriggerEx

指令原型	short GTN_SetTriggerEx(short core,short index,TTriggerEx *pTrigger)		
指令说明	设置 Trigger 捕获触发模式		
指令类型	立即指令，调用后立即生效。	章节页码	104
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
index	Trigger 序号，正整数，取值范围请参照表 13-1 中的“轴”一栏		

pMode	设置 Trigger 捕获模式
	typedef struct Trigger
	{
	short latchType; //锁存类型：MC_ENCODER 23		

	//MC_AU_ENCODER 26
	short latchIndex; //编码器序号，取值范围请参考
	short probeType; // 捕获类型，取值范围[1,3]
	// CAPTURE_HOME 1
	// CAPTURE_INDEX2
	// CAPTURE_PROBE 3
	short probeIndex; // 捕获类型对应的DI序号
	short sense; // 捕获沿,0:下降沿，1:上升沿
	long offset; // 捕获位置偏置值，设置为0，只有GTM模块支持offset为非0值
	unsigned long loop; // 捕获循环测试，0：无限循环，其他代表循环次数
	short windowOnly; // 捕获窗使能，0：不开启捕获窗，1：开启捕获窗
	long firstPosition; // windowOnly=1时生效，触发捕获位置的起点
	long lastPosition; // windowOnly=1时生效，触发捕获位置的终点
	}TTriggerEx;

指令返回值	请参照指令返回值列表。
相关指令	GTN_GetTriggerEx
指令示例	**例程 9-1 Home/Index 捕获**

指令 208 GTN_SetUserSegNum

指令原型	short GTN_SetUserSegNum(short core, short crd, long segNum, short fifo=0)		
指令说明	设置自定义插补段段号。		
指令类型	缓存区指令。	章节页码	65
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
crd	坐标系号。正整数，取值范围请参照表 13-1 中的“插补坐标系序号”一栏		
segNum	设置用户自定义的插补段段号。		
fifo	插补缓存区号。默认值为：0，取值范围请参照表 13-1 中的“插补缓存区序号”一栏		
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。		
相关指令	GTN_GetUserSegNum		
指令示例	无。		

指令 209 GTN_SetVarValue

指令原型	short GTN_SetVarValue (short core, short page, TVarInfo *pVarInfo, double *pValue, short count=1)		
指令说明	设置运动程序中变量的值。		
指令类型	立即指令，调用后立即生效。	章节页码	133
指令参数	该指令共有 5 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
page	数据页编号。		

	全局变量为-1。 局部变量取值范围：[0, 31]。
pVarInfo	需要访问的变量标识。
pValue	需要写入的变量值。
count	需要写入的变量值的数量，取值范围：[1, 8]。
指令返回值	请参照指令返回值列表。
相关指令	GTN_GetVarValue
指令示例	**例程 11-1　　运动程序单线程累加求和**

指令 210 GTN_SetVel

指令原型	short GTN_SetVel(short core, short profile, double vel)		
指令说明	设置目标速度。		
指令类型	立即指令，调用后立即生效。	章节页码	54, 57
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
profile	规划轴号，正整数，取值范围请参照表 13-1 中的“轴”一栏		
vel	设置目标速度。单位：pulse/ms。		
指令返回值	若返回值为 1：请检查当前轴是否为 Trap 模式，若不是，请先调用 GTN_PrfTrap 将当前轴设置为 Trap 模式。 其他返回值：请参照指令返回值列表。		
相关指令	GTN_GetVel		
指令示例	**例程 7-1 点位运动**		

指令 211 GTN_StartHandwheel

指令原型	GTN_ StartHandwheel (short core,short slave,short master,short masterEven,short slaveEven,short intervalTime,double acc,double dec,double vel,short stopWaitTime)		
指令说明	初始化电子齿轮跟随功能。		
指令类型	立即指令，调用后立即生效。	章节页码	
指令参数	该指令共有 10 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
slave	从轴轴号，正整数，取值范围请参照表 13-1 中的“轴”一栏		
master	主轴轴号，正整数，取值范围请参照表 13-1 中的“辅助编码器”一栏。		
masterEven	主轴传动比系数，主轴的位移，取值[-32768,32767]。		
slaveEven	从轴传动比系数，从轴的位移。（两个系数的比值就是齿轮比）		
intervalTime	主轴位移采样时间间隔，取值[1,32767]，单位：规划周期。		
acc	跟随过程中从轴的加速度，正值。		
dec	跟随过程中从轴的减速度，正值。		
vel	跟随过程中从轴的最大速度，正值。		
stopWaitTime	判断停止时间，取值[1,32767]，单位：规划周期。高速跟随时，主轴急停，从轴经过多少时间后才停止，防止从轴跟随冲击过大。		
指令返回值	1：从轴正在运动 7：参数取值超过范围限制		
相关指令			
指令示例	**参考 12.12.1 例程**		

指令 212 GTN_StepDir

指令原型	short GTN_StepDir(short core, short step)		
指令说明	将脉冲输出通道的脉冲输出模式设置为“脉冲+方向”。		
指令类型	立即指令，调用后立即生效。	章节页码	41
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
step	脉冲输出通道号，正整数，取值范围请参照表13-1中的“轴”一栏		
指令返回值	若返回值为 1： (1) 若当前轴在规划运动，请调用 GTN_Stop 停止运动再调用该指令； (2) 若当前轴伺服使能，请调用 GTN_AxisOff 停止使能再调用该指令。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	**例程 5-2 设置第 1 轴为脉冲控制“脉冲+方向”方式**		

指令 213 GTN_StepPulse

指令原型	short GTN_StepPulse(short core, short step)		
指令说明	将脉冲输出通道的脉冲输出模式设置为“CCW/CW”。		
指令类型	立即指令，调用后立即生效。	章节页码	41
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
step	脉冲输出通道号，正整数，取值范围请参照表13-1中的“轴”一栏		
指令返回值	若返回值为 1： (1) 若当前轴在规划运动，请调用 GTN_Stop 停止运动再调用该指令。 (2) 若当前轴伺服使能，请调用 GTN_AxisOff 停止使能再调用该指令。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 214 GTN_Stop

指令原型	short GTN_Stop(short core, long mask, long option)
指令说明	停止一个或多个轴的规划运动，停止坐标系运动。
指令类型	立即指令，调用后立即生效。 章节页码 47
指令参数	该指令共有 3 个参数，参数的详细信息如下。
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏
mask	按位指示需要停止运动的轴号或者坐标系号。当 bit 位为 1 时表示停止对应的轴或者坐标系。
option	按位指示停止方式。当 bit 位为 0 时表示平滑停止对应的轴或坐标系，当 bit 位为 1 时表示急停对应的轴或坐标系。

mask 对应表：

Bit	31	30	29	28	……	3	2	1	0
对应轴或坐标系	32 轴	31 轴	30 轴	29 轴	……	4 轴	3 轴	2 轴	1 轴

option 对应表：

Bit	31	30	29	28	……	3	2	1	0
对应轴或坐	32	31	30	29	……	4	3	2	1

	标系 | 轴 | 轴 | 轴 | 轴 | | 轴 | 轴 | 轴 | 轴
指令返回值	请参照指令返回值列表。 注意：如果需要停止坐标系，则停止对应坐标系中的任何一个轴即可。
相关指令	无。
指令示例	无。

指令 215 GTN_StopThread

指令原型	short GTN_StopThread(short core, short thread)		
指令说明	停止正在运行的线程。		
指令类型	立即指令，调用后立即生效。	章节页码	133
指令类型	立即指令，调用后立即生效。		
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
thread	线程编号，取值范围：[0, 31]。		
指令返回值	若返回值为 1： (1) 请检查线程号是否已经绑定。 (2) 请检查相应的数据页中是否超出范围。 其他返回值：请参照指令返回值列表。		
相关指令	GTN_RunThread; GTN_PauseThread		
指令示例	无。		

指令 216 GTN_SynchAxisPos

指令原型	shortGTN_SynchAxisPos(short core, long mask)
指令说明	axis 合成规划位置和所关联的 profile 同步。 axis 合成编码器位置和所关联的 encoder 同步。
指令类型	立即指令，调用后立即生效。 章节页码 146
指令参数	该指令共有 2 个参数，参数的详细信息如下。
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏
mask	按位标识需要进行位置同步的轴号。0：表示不需要进行位置同步，1：需要进行位置同步。 Bit | 31 | 30 | 29 | 28 | …… | 3 | 2 | 1 | 0 对应轴 | 32 轴 | 31 轴 | 30 轴 | 29 轴 | …… | 4 轴 | 3 轴 | 2 轴 | 1 轴
指令返回值	若返回值为 1：请检查参数 mask 是否设置为 0。 其他返回值：请参照指令返回值列表。
相关指令	无。
指令示例	无。

指令 217 GTN_Update

指令原型	short GTN_Update(short core, long mask)		
指令说明	启动点位运动或 Jog 运动。		
指令类型	立即指令，调用后立即生效。	章节页码	54, 57
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		

mask	按位指示需要启动点位运动或 Jog 运动的轴号。当 bit 位为 1 时表示启动对应的轴。 Bit 31 30 29 28 …… 3 2 1 0 对应轴 32轴 31轴 30轴 29轴 …… 4轴 3轴 2轴 1轴
指令返回值	请参照指令返回值列表.
相关指令	无。
指令示例	**例程 7-1 点位运动**

指令 218 GTN_WriteAuEncPos

指令原型	short GTN_WriteAuEncPos (short core, short encoder,double *pPos,short count)		
指令说明	修改辅助编码器位置。		
指令类型	立即指令，调用后立即生效。	章节页码	99
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
encoder	编码器起始轴号，正整数，取值范围请参照表 13-1 中的“轴”一栏		
encPos	编码器位置。		
count	辅助编码器个数，正整数，取之范围为[1,6]		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	无。		

指令 219 GTN_WriteMpgPos

指令原型	short GTN_WriteMpgPos (short core, short mpg,double *pPos,short count)		
指令说明	修改手轮编码器位置。		
指令类型	立即指令，调用后立即生效。	章节页码	99
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
mpg	手轮起始编号，正整数，取值范围请参照表 13-1 中的“轴”一栏		
pPos	编码器位置。		
count	手轮个数，正整数，取之范围为[1,6]		
指令返回值	请参照指令返回值列表。		

指令 220 GTN_ZeroPos

指令原型	short GTN_ZeroPos(short core, short axis, short count=1)		
指令说明	清零规划位置和实际位置，并进行零漂补偿。		
指令类型	立即指令，调用后立即生效。	章节页码	146
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，内核，正整数，取值范围请参照表13-1中的“内核”一栏		
axis	需要位置清零的起始轴号，内核，正整数，取值范围请参照表 13-1 中的“轴”一栏		
count	需要位置清零的轴数。 默认值为 1，正整数，取值范围请参照表 13-1 中的“轴”一栏		
指令返回值	若返回值为 1：若当前轴在规划运动，请调用 GTN_Stop 停止运动再调用该指令。 其他返回值：请参照指令返回值列表。		

相关指令	无。
指令示例	无。

指令 221 GTN_BufDoBit

指令原型	short GTN_BufDoBit(short core, short crd, unsigned short doType, unsigned short index, short value, short fifo)		
指令说明	缓存区内数字量输出设置指令（可以设置大于 16 路数字量的输出）。		
指令类型	缓存区指令。	章节页码	65
指令参数	该指令共有 6 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表13-1中的“内核”一栏		
crd	坐标系号，取值范围请参照表 13-1 中的“插补坐标系号”一栏		
doType	数字量输出的类型。 MC_ENABLE(该宏定义为 10)：输出驱动器使能。 MC_CLEAR(该宏定义为 11)：输出驱动器报警清除。 MC_GPO(该宏定义为 12)：输出通用输出。		
index	数字量输出的索引。		
value	设置数字量输出的值。 默认情况下，1 表示高电平，0 表示低电平。		
fifo	插补缓存区号。默认值为：0，取值范围请参照表 13-1 中的“插补缓存区序号”一栏		
指令返回值	若返回值为 1： (1) 检查当前坐标系是否映射了相关轴。 (2) 检查是否向 fifo1 中传递数据，若是，则检查 fifo0 是否使用并运动，若运动，则返回错误。 (3) 检查相应的 fifo 是否已满。 其他返回值：请参照指令返回值列表。		
相关指令	无。		
指令示例	无		

指令 222 GTN_SetAuTrigger

指令原型	short GTN_SetAuTrigger(short core,short index,TTriggerEx *pTrigger)		
指令说明	设置 Trigger 捕获触发模式		
指令类型	立即指令，调用后立即生效。	章节页码	104
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
index	AuTrigger 序号，正整数，取值范围请参照表 13-1 中的“轴”一栏		

pTrigger	设置 Trigger 捕获模式
	typedef struct Trigger
	{
	short latchType; //锁存类型：MC_ENCODER 23
	//MC_AU_ENCODER 26
	short latchIndex; //编码器序号，取值范围请参考
	short probeType; // 捕获类型，取值范围[1,3]
	// CAPTURE_HOME 1		

	// CAPTURE_INDEX2
	// CAPTURE_PROBE 3
	short probeIndex; // 捕获类型对应的DI序号
	short sense; // 捕获沿,0:下降沿，1:上升沿
	long offset; // 捕获位置偏置值，设置为0，只有GTM模块支持offset为非0值
	unsigned long loop; // 捕获循环测试，0：无限循环，其他代表循环次数
	short windowOnly; // 捕获窗使能
	long firstPosition; // 触发捕获位置的起点
	long lastPosition; // 触发捕获位置的终点
	}TTriggerEx;

指令返回值	请参照指令返回值列表。
相关指令	
指令示例	

指令 223 GTN_GetAuTrigger

指令原型	short GTN_GetAuTrigger(short core,short index,TTriggerEx *pTrigger)		
指令说明	设置 Trigger 捕获触发模式		
指令类型	立即指令，调用后立即生效。	章节页码	104
指令参数	该指令共有 3 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
index	AuTrigger 序号，正整数，取值范围请参照表 13-1 中的“轴”一栏		

pTrigger	设置 Trigger 捕获模式
	typedef struct Trigger
	{
	short latchType; //锁存类型：MC_ENCODER 23
	//MC_AU_ENCODER 26
	short latchIndex; //编码器序号，取值范围请参考
	short probeType; // 捕获类型，取值范围[1,3]
	// CAPTURE_HOME 1
	// CAPTURE_INDEX2
	// CAPTURE_PROBE 3
	short probeIndex; // 捕获类型对应的DI序号
	short sense; // 捕获沿,0:下降沿，1:上升沿
	long offset; // 捕获位置偏置值，设置为0，只有GTM模块支持offset为非0值
	unsigned long loop; // 捕获循环测试，0：无限循环，其他代表循环次数
	short windowOnly; // 捕获窗使能
	long firstPosition; // 触发捕获位置的起点
	long lastPosition; // 触发捕获位置的终点
	}TTriggerEx;		

指令返回值	请参照指令返回值列表。		
相关指令			
指令示例			

指令 224 GTN_GetAuTriggerStatus

指令原型	short GTN_GetAuTriggerStatus(short core, short index, TTriggerStatusEx *pTriggerStatus, short count=1)		
指令说明	读取 Trigger 捕获状态。		
指令类型	立即指令，调用后立即生效。	**章节页码**	104
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
index	AuTrigger 序号，正整数，取值范围请参照表 13-1 中的“轴”一栏		

pTriggerStatus	typedef struct TriggerStatusEx
	{
	short execute; //捕获运行状态，1:使能捕获，0：未使能捕获
	short done; //捕获状态，1：捕获完成，0：捕获中
	long position; //捕获位置
	unsigned long clock; //保留
	unsigned long loopCount; //重复捕获的第几次捕获
	}TTriggerStatusEx;		

count	读取Trigger个数，取值范围[1,4]		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例			

指令 225 GTN_ClearAuTriggerStatus

指令原型	short GTN_ClearAuTriggerStatus(short core, short index)		
指令说明	读取 Trigger 捕获状态。		
指令类型	立即指令，调用后立即生效。	**章节页码**	104
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
index	AuTrigger 序号，正整数，取值范围请参照表 13-1 中的“轴”一栏		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例			

指令 226 GTN_ClearAlarm

指令原型	short GTN_ClearAlarm(short core, short axis, short count, unsigned short delayTime)		
指令说明	清除轴的报警信号		
指令类型	立即指令，调用后立即生效。	**章节页码**	104
指令参数	该指令共有 4 个参数，参数的详细信息如下。		
core	内核，正整数，取值范围请参照表 13-1 中的“内核”一栏		
axis	轴序号，正整数，取值范围请参照表 13-1 中的“轴”一栏		
count	轴数，默认为 1，正整数，即从轴序号 axis 开始需要进行操作的轴数量，取值范围请参照表 13-1 中的“参数个数”一栏		

delayTime	设置的轴报警信号低电平保持时间
指令返回值	请参照指令返回值列表。
相关指令	无。
指令示例	

第14章 索引

14.1 指令索引

14.2 例程索引

14.3 表格索引

14.4 图片索引