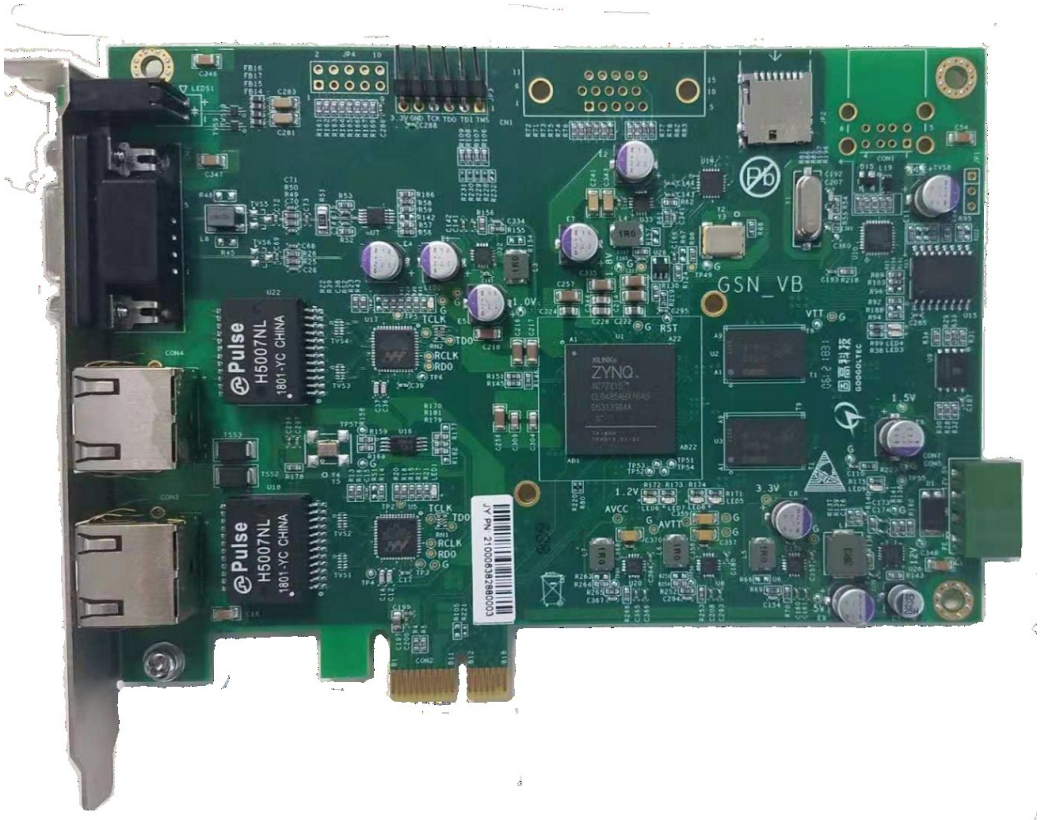
固高科技
GOOGOL TECHNOLOGY



GEN 系列运动控制器编程手册

基本功能

R1.3

2020 年 03 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（深圳）有限公司

地　　址：深圳市高新技术产业园南区深港产学研基地西座二楼 W211 室
电　　话：0755-26970817　26737236　26970824
传　　真：0755-26970821
电子邮件：support@googoltech.com
网　　址：http://www.googoltech.com.cn

固高科技（香港）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：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街 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	2018 年 10 月 25 日
1.1	2019 年 03 月 01 日
1.2	2019 年 09 月 19 日
1.3	2020 年 03 月 24 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（深圳）有限公司
邮编： 518057

编程手册的用途

用户通过阅读本手册，能够了解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

本手册由十三章内容组成，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《GEN 系列运动控制器用户手册》。
关于更复杂的控制器功能，请参见随产品配套的《GEN 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《GEN 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》。

目录

版权申明 1

联系我们 1

文档版本 2

前言 3

第 1 章　指令列表 7

第 2 章　运动控制器函数库的使用 12

2.1　Windows 系统下动态链接库的使用 12
2.1.1　Visual C++ 6.0 中的使用 12
2.1.2　Visual Basic 6.0 中的使用 12
2.1.3　Delphi 中的使用 12
2.1.4　Visual Basic 2008 中的使用 13
2.1.5　Visual C#中的使用 13

第 3 章　指令返回值及其意义 14

3.1　本章简介 14
3.2　指令返回值 14
3.3　例程 15

第 4 章　系统配置 16

4.1　本章简介 16
4.2　系统配置基本概念 16
4.2.1　硬件资源 16
4.2.2　软件资源 16
4.2.3　资源组合 16
4.3　系统配置工具 18
4.3.1　配置 axis 19
4.3.2　配置 profile 22
4.3.3　配置 encoder 22
4.4　配置文件生成和下载 24
4.5　配置信息修改指令 24
4.5.1　指令列表 24
4.5.2　重点说明 25
4.5.3　例程 25
4.6　控制器配置初始化状态 26

第 5 章　EtherCAT 指令说明 27

5.1　本章简介 27
5.2　EtherCAT 库指令 27
5.2.1　指令列表 27
5.2.2　重点说明 28
5.2.3　例程 32

第 6 章　运动状态检测 36

6.1　本章简介 36
6.2　指令列表 36
6.3　重点说明 36
6.3.1　轴状态定义 36
6.3.2　轴的运动参数 37
6.4　例程 38

第 7 章　运动模式 42

7.1　本章简介 42
7.2　点位运动模式 42
7.2.1　指令列表 42
7.2.2　重点说明 43
7.2.3　例程 43
7.3　Jog 运动模式 46
7.3.1　指令列表 46
7.3.2　重点说明 46
7.3.3　例程 46
7.4　电子齿轮（Gear）运动模式 49
7.4.1　指令列表 49
7.4.2　重点说明 49
7.4.3　例程 51
7.5　插补运动模式 53
7.5.1　指令列表 53
7.5.2　重点说明 54

第 8 章　访问硬件资源 81

8.1　本章简介 81
8.2　访问编码器 81
8.2.1　指令列表 81
8.2.2　例程 82
8.3　访问内部脉冲计数器 82
8.3.1　指令列表 82
8.3.2　重点说明 82
8.3.3　例程 83

第 9 章　安全机制 84

9.1　本章简介 84
9.2　限位 84
9.2.1　指令列表 84
9.2.2　重点说明 85
9.2.3　例程 85
9.3　报警 86
9.4　平滑停止和急停 86
9.5　掉电存储功能 87
9.5.1　指令列表 87

9.5.2 重点说明 87
9.5.3 例程 87

第 10 章 运动程序 88

10.1 本章简介 88
10.2 运动程序概述 88
10.3 运动程序的使用 89
10.3.1 指令列表 89
10.3.2 编写运动程序 89
10.3.3 编译 90
10.3.4 下载 90
10.3.5 绑定线程、函数和数据页 90
10.3.6 启动，停止，暂停线程 91
10.3.7 查询线程状态 91
10.3.8 例程 91
10.4 如何编写运动程序 93
10.4.1 语言元素 95
10.4.2 运算指令 96
10.4.3 流程控制与标准 C 语言的流程控制对比 97
10.5 可在运动程序中使用的指令 99

第 11 章 其它指令 101

11.1 本章简介 101
11.2 打开/关闭运动控制器 101
11.3 读取固件版本号 101
11.4 读取系统时钟 102
11.5 打开/关闭电机使能信号 102
11.6 维护位置值 102
11.7 电机到位检测 103
11.8 反向间隙补偿 106
11.9 螺距误差补偿 107
11.10 二维位置补偿 108
11.10.1 重点说明 108
11.10.2 例程 109

第 12 章 指令详细说明 111

第 13 章 索引 178

13.1 指令索引 178
13.2 例程索引 181
13.3 表格索引 182
13.4 图片索引 183

第1章 指令列表

注意	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　指令列表

第 4 章　系统配置	
4.4　配置文件生成和下载	
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
4.5　配置信息修改指令	
GTN_AlarmOff	控制轴驱动报警信号无效
GTN_AlarmOn	控制轴驱动报警信号有效
GTN_LmtsOn	控制轴限位信号有效
GTN_LmtsOff	控制轴限位信号无效
GTN_ProfileScale	设置控制轴的规划器当量变换值
GTN_EncScale	设置控制轴的编码器当量变换值
GTN_SetStopDec	设置平滑停止减速度和急停减速度
GTN_GetStopDec	读取平滑停止减速度和急停减速度
GTN_SetSense	设置输入输出资源的电平逻辑。
第 5 章　EtherCAT 指令说明	
GTN_InitEcatComm	EtherCAT 初始化
GTN_StartEcatComm	启动 DSP 总线运动控制
GTN_TerminateEcatComm	结束 EtherCAT 通讯
GTN_IsEcatReady	查询 GUC EtherCAT 通讯状态

GTN_EcatSDODownload	通用 SDO 下载（Service Data Object，参考 IEC 61800）
GTN_EcatSDOUpLoad	通用 SDO 上传（Service Data Object，参考 IEC 61800）
GTN_SetEcatHomingPrm	设置 EtherCAT 轴回零的参数
GTN_SetHomingMode	切换 EtherCAT 轴的回零模式
GTN_StartEcatHoming	启动 EtherCAT 轴回零
GTN_StopEcatHoming	停止 EtherCAT 轴回零
GTN_GetEcatHomingStatus	查询 EtherCAT 轴的回零状态
GTN_SetTouchProbeFunction	设置探针功能的参数（参数方式）
GTN_SetTouchProbeFunctionEx	设置探针功能的参数（功能描述方式）
GTN_GetTouchProbeStatus	查询 EtherCAT 轴的探针状态和捕获值
GTN_EcatIOReadInput	读取 EtherCAT IO 模块数字量输入
GTN_EcatIOWriteOutput	写入 EtherCAT IO 模块数字量输出
GTN_GetEcatAxisAI	读取 EtherCAT 轴的模拟量输入
GTN_GetEcatAxisDI	读取 EtherCAT 轴的数字量输入
GTN_SetEcatAxisDO	设置 EtherCAT 轴的数字量输出
GTN_GetEcatAxisDO	读取 EtherCAT 轴的数字量输出
GTN_SetPosScale	设置编码器倍率值
GTN_GetPosScale	读取编码器倍率值
GTN_GetEcatEncPos	读取 EtherCAT 轴的编码器位置
GTN_GetEcatEncVel	读取 EtherCAT 轴的编码器速度

GTN_GetEcatAxisPE	读取 EtherCAT 轴的规划和编码器误差
GTN_SetEcatAxisMode	设置 EtherCAT 轴的操作模式
GTN_GetEcatAxisMode	读取 EtherCAT 轴的操作模式
GTN_SetEcatAxisPT	设置 EtherCAT 轴的力矩
GTN_GetEcatAxisAtlTorque	读取 EtherCAT 轴的力矩值
GTN_GetEcatAxisAtlCurrent	读取 EtherCAT 轴的电流值
GTN_SetEcatAxisPV	设置 EtherCAT 轴的速度
GTN_GetEcatSlaves	读取 EtherCAT 总线的在线从站数目
GTN_GetMcEcatAxisNum	读取 EtherCAT 伺服轴数
GTN_GetEcatRawData	读取 EtherCAT 总线的 PDO 数据
GTN_SetEcatRawData	设置 EtherCAT 总线的 PDO 数据
第 6 章　运动状态检测	
GTN_GetSts	读取轴状态
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
GTN_GetPrfMode	读取轴运动模式
GTN_GetPrfPos	读取规划位置
GTN_GetPrfVel	读取规划速度
GTN_GetPrfAcc	读取规划加速度
GTN_GetAxisPrfPos	读取轴(axis)的规划位置值
GTN_GetAxisPrfVel	读取轴(axis)的规划速度值
GTN_GetAxisPrfAcc	读取轴(axis)的规划加速度值

GTN_GetAxisEncPos	读取轴(axis)的编码器位置值
GTN_GetAxisEncVel	读取轴(axis)的编码器速度值
GTN_GetAxisEncAcc	读取轴(axis)的编码器加速度值
GTN_GetAxisError	读取轴(axis)的规划位置值和编码器位置值的差值
GTN_Stop	停止一个或多个轴的规划运动，停止坐标系运动
第 7 章　运动模式	
7.2　点位运动模式	
GTN_PrfTrap	设置指定轴为点位运动模式
GTN_SetTrapPrm	设置点位运动模式下的运动参数
GTN_GetTrapPrm	读取点位运动模式下的运动参数
GTN_SetPos	设置目标位置
GTN_GetPos	读取目标位置
GTN_SetVel	设置目标速度
GTN_GetVel	读取目标速度
GTN_Update	启动点位运动
7.3 Jog 运动模式	
GTN_PrfJog	设置指定轴为 Jog 运动模式
GTN_SetJogPrm	设置 Jog 运动模式下的运动参数
GTN_GetJogPrm	读取 Jog 运动模式下的运动参数
GTN_SetVel	设置目标速度
GTN_GetVel	读取目标速度
GTN_Update	启动 Jog 运动
7.4　电子齿轮（Gear）运动模式	
GTN_PrfGear	设置指定轴为电子齿轮运动模式

GTN_SetGearMaster	设置电子齿轮运动跟随主轴
GTN_GetGearMaster	读取电子齿轮运动跟随主轴
GTN_SetGearRatio	设置电子齿轮比
GTN_GetGearRatio	读取电子齿轮比
GTN_GearStart	启动电子齿轮运动
7.5 插补运动模式	
GTN_SetCrdPrm	设置坐标系参数，确立坐标系映射，建立坐标系
GTN_GetCrdPrm	查询坐标系参数
GTN_CrdData	向插补缓存区增加插补数据
GTN_LnXY	缓存区指令，二维直线插补
GTN_LnXYZ	缓存区指令，三维直线插补
GTN_LnXYZA	缓存区指令，四维直线插补
GTN_LnXYG0	缓存区指令，二维直线插补(终点速度始终为 0)
GTN_LnXYZG0	缓存区指令，三维直线插补(终点速度始终为 0)
GTN_LnXYZAG0	缓存区指令，四维直线插补(终点速度始终为 0)
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)
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)
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)
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)
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)
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)
GTN_BufIO	缓存区指令，缓存区内数字量 IO 输出设置指令
GTN_BufDelay	缓存区指令，缓存区内延时设置指令

GTN_BufDA	缓存区指令，缓存区内输出 DA 值
GTN_BufLmtsOn	缓存区指令，缓存区内有效限位开关
GTN_BufLmtsOff	缓存区指令，缓存区内无效限位开关
GTN_BufSetStopIo	缓存区指令，缓存区内设置 axis 的停止 IO 信息
GTN_BufMove	缓存区指令，实现刀向跟随功能，启动某个轴点位运动
GTN_BufGear	缓存区指令，实现刀向跟随功能，启动某个轴跟随运动
GTN_CrdSpace	查询插补缓存区剩余空间
GTN_CrdClear	清除插补缓存区内的插补数据
GTN_CrdStart	启动插补运动
GTN_CrdStatus	查询插补运动坐标系状态
GTN_SetUserSegNum	缓存区指令，设置自定义插补段段号
GTN_GetUserSegNum	读取自定义插补段段号
GTN_GetRemainderSegNum	读取未完成的插补段段数
GTN_SetOverride	设置插补运动目标合成速度倍率
GTN_SetCrdStopDec	设置插补运动平滑停止、急停合成加速度
GTN_GetCrdStopDec	查询插补运动平滑停止、急停合成加速度
GTN_GetCrdPos	查询该坐标系的当前坐标位置值
GTN_GetCrdVel	查询该坐标系的合成速度值
GTN_InitLookAhead	初始化插补前瞻缓存区
GTN_CrdHsOn	开启 DMA 传输通道
GTN_CrdHsOff	关闭 DMA 传输通道
GTN_GetCrdHsPrm	读取插补坐标系 DMA 设置参数

8.2　访问编码器	
GTN_GetEncPos	读取轴编码器位置
GTN_GetEncVel	读取轴编码器速度
GTN_SetEncPos	修改轴编码器位置
GTN_SetSense	设置输入输出资源的电平逻辑。
GTN_GetSense	读取输入输出资源的电平逻辑
8.3　访问内部脉冲计数	
GTN_SetPlsPos	设置内部脉冲计数位置
GTN_GetPlsPos	读取内部脉冲计数位置
GTN_GetPlsVel	读取内部脉冲计数速度
第 9 章　安全机制	
9.2　限位	
GTN_SetSoftLimitMode	设置软限位模式
GTN_GetSoftLimitMode	读取软限位模式
GTN_GetLimitStatus	读取限位状态
GTN_SetSoftLimit	设置轴正向软限位和负向软限位
GTN_GetSoftLimit	读取轴正向软限位和负向软限位
9.5　掉电存储功能	
GTN_GetRetainValue	读取 MRAM 存储芯片数据
GTN_SetRetainValue	保存数据到 MRAM 存储芯片
第 10 章　运动程序	
GTN_Download	下载运动程序到运动控制器
GTN_GetFunId	读取运动程序中函数的标识
GTN_GetVarId	读取运动程序中变量的标识
GTN_Bind	绑定线程、函数、数据页

GTN_RunThread	启动线程
GTN_StopThread	停止正在运行的线程
GTN_PauseThread	暂停正在运行的线程
GTN_GetThreadSts	读取线程的状态
GTN_SetVarValue	设置运动程序中变量的值
GTN_GetVarValue	读取运动程序中变量的值
第 11 章　其它指令	
11.2　打开/关闭运动控制器	
GTN_Open	打开运动控制器
GTN_Close	关闭运动控制器
GTN_Reset	复位运动控制器
11.3　读取固件版本号	
GTN_GetVersion	读取运动控制器固件的版本号
GTN_GetVersionEx	读取动态库的版本号
11.4　读取系统时钟	
GTN_GetClock	读取运动控制器系统时钟
GTN_GetClockHighPrecision	读取运动控制器系统高精度时钟
11.5　打开/关闭电机使能信号	
GTN_AxisOn	打开驱动器使能
GTN_AxisOff	关闭驱动器使能
11.6　维护位置值	
GTN_SetPrfPos	修改指定轴的规划位置
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
GTN_ZeroPos	清零规划位置和实际位置，并进行零漂补偿
11.7　电机到位检测	

GTN_SetAxisBand	设置轴到位误差带
GTN_GetAxisBand	读取轴到位误差带
11.8　反向间隙补偿	
GTN_SetBacklash	设置反向间隙补偿的相关参数
GTN_GetBacklash	读取反向间隙补偿的相关参数
11.9　螺距误差补偿	
GTN_SetLeadScrewComp	加载补偿表
GTN_EnableLeadScrewComp	是否开启补偿
11.10　二维位置补偿	
GTN_SetCompensate2DTable	设置二维补偿表及数据
GTN_GetCompensate2DTable	获取二维补偿表参数
GTN_SetCompensate2D	设置二维补偿参数
GTN_GetCompensate2D	获取二维补偿参数
GTN_GetCompensate2DValue	获取补偿值

第2章 运动控制器函数库的使用

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。运动控制器的驱动程序存放在产品配套光盘的“Windows\Driver”文件夹下。

运动控制器指令函数动态链接库存放在产品配套光盘的“Windows”文件夹下。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。和网络配置相关的文件名为 RingNetDmaCfg.rndma 和 RingNetMapSUB6.rnmap。

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 Visual C++ 6.0 中的使用

(1) 启动Visual C++ 6.0，新建一个工程；

(2) 将产品配套光盘Windows\VC6文件夹中的动态链接库(gts.dll和gt_rn.dll)、头文件(gts.h)、lib文件(gts.lib)和网络初始化配置文件（RingNetDmaCfg.rndma和RingNetMapSUB6.rnmap）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。

2.1.2 Visual Basic 6.0 中的使用

(1) 启动Visual Basic，新建一个工程；

(2) 将产品配套光盘Windows\VB6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；

(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。

2.1.3 Delphi 中的使用

(1) 启动Delphi，新建一个工程；

(2) 将产品配套光盘Windows\Delphi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；

(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。

2.1.4　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)　将产品配套光盘Windows\VB2008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.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。

2.1.5　Visual C#中的使用

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

(2)　将产品配套光盘Windows\C#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 12 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	1. 检查当前指令的执行条件是否满足
2	license 不支持	1. 如果需要此功能，请与生产厂商联系。
7	指令参数错误	1. 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	1. 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
-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专用处理器是否工作正常
-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专用处理器是否工作正常
-15	动态库版本不匹配	1. 无法正常工作，需要更新 gt_rn.dll
15	动态库版本不匹配	1. 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	1. 固件版本比较老，建议更新专用处理器固件

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 VS2010 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 系统配置

4.1　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。

注意	本章表格中的“页码”即为此指令在“第 12 章 指令详细说明”中的对应页码。可以使用“超级链接”进行索引。 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。

4.2　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。

4.2.1　硬件资源

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。

4.2.2　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。

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。

4.2.3　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 4-1 所示。

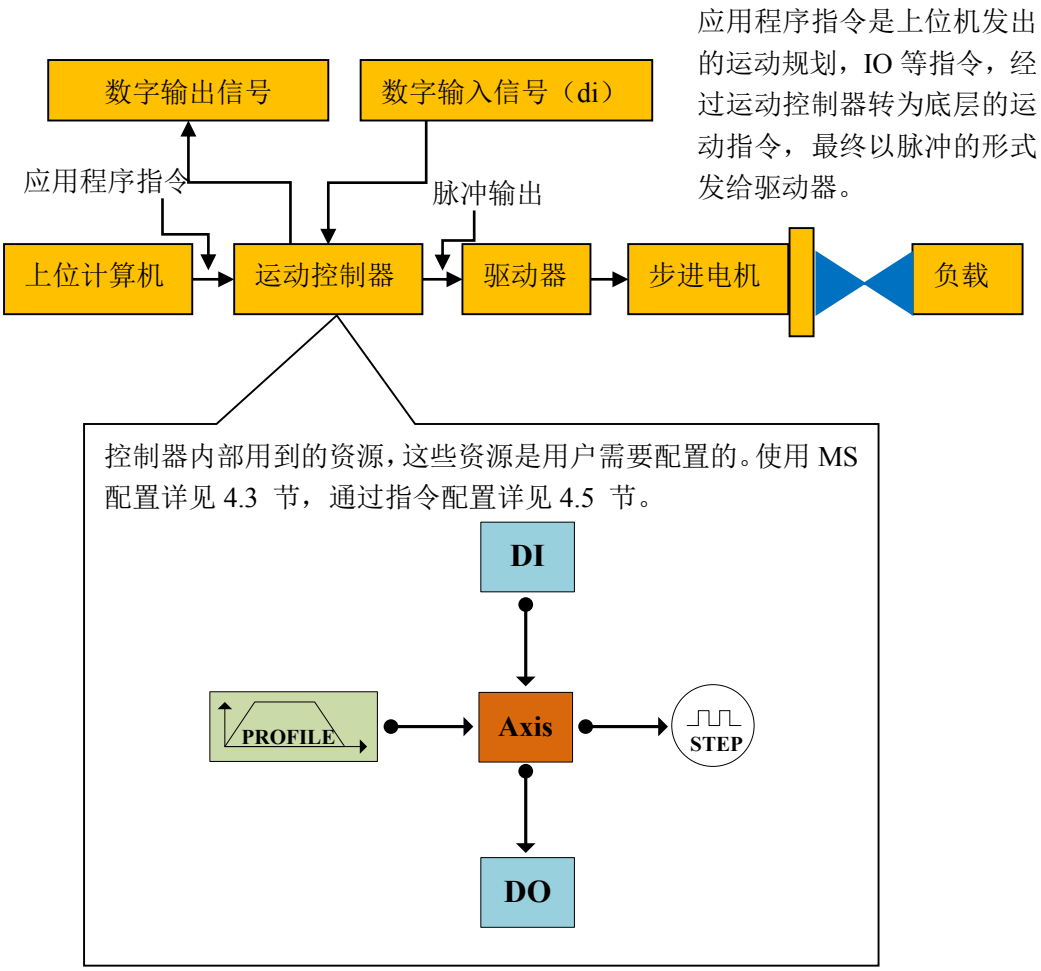


图 4-1　开环运动控制系统的配置

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 4-2 所示。

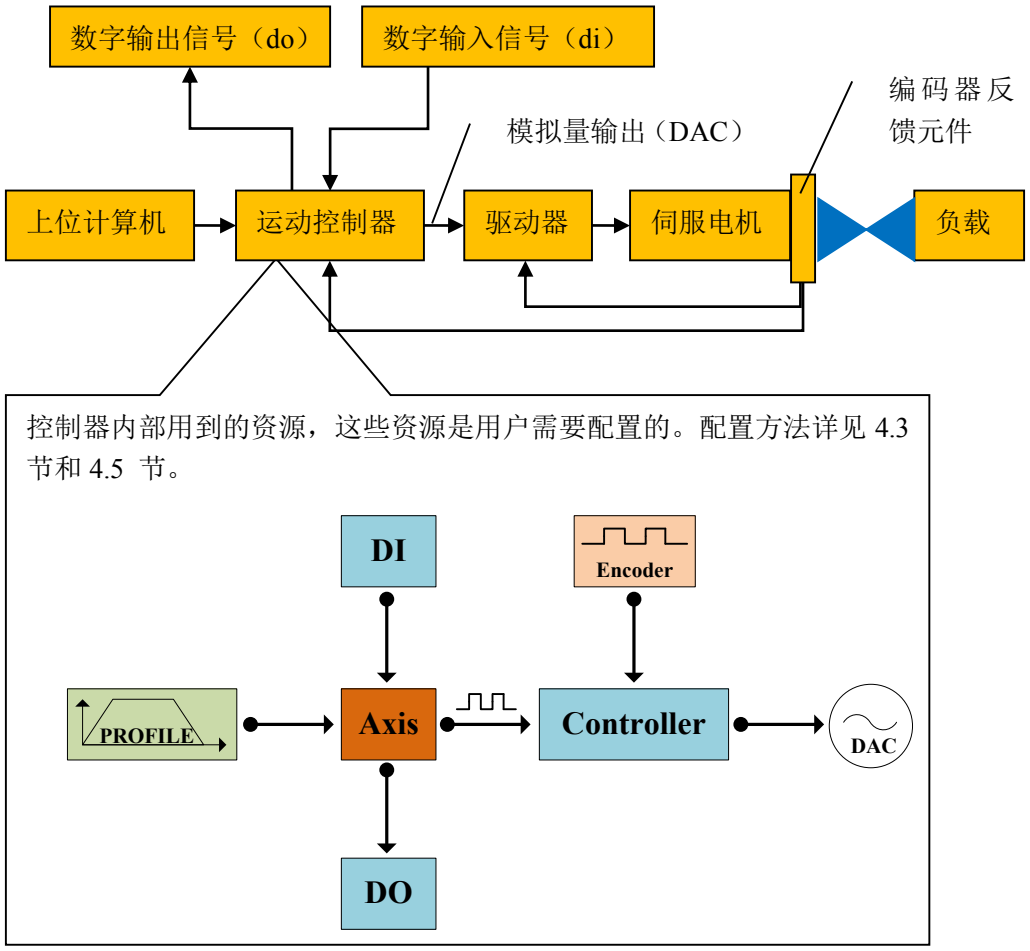


图 4-2　闭环运动控制系统的配置

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。

4.3　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 4-3 所示界面。

图 4-3　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 4-4 所示。

图 4-4　打开控制器配置

4.3.1　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 4-5 所示。

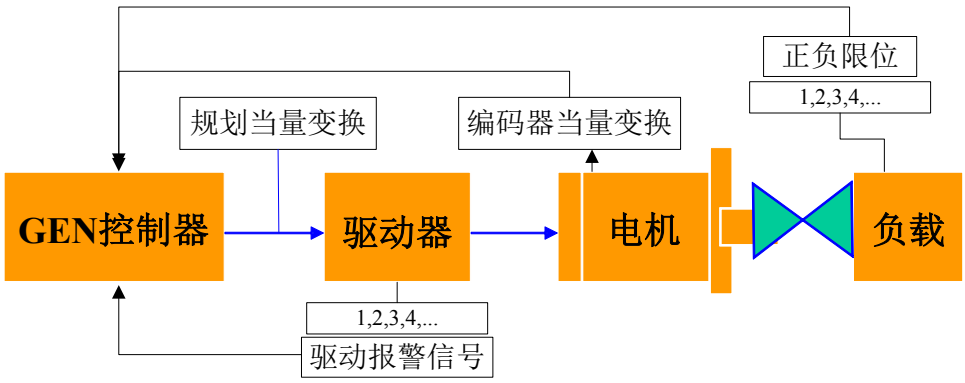


图 4-5　axis 配置对控制系统的影响

按照功能，MS 将 Axis 配置功能分为 2 个大模块，分别为报警功能和当量变换功能。界面如图 4-6~图 4-7 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。

图 4-6　axis 配置界面 1

(1)　驱动报警信号选择：驱动器报警信号由状态字（0x6041）的 Fault 位（bit3）解析而来，图 4-6“（驱动报警）类型”不可配置为轴报警以外的其它信号类型。“（驱动报警）索引号”下拉列表选择报警信号的编号，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)　正限位信号选择：正限位信号由数字量输入（0x60FD）的 bit1 解析而来，图 4-6“（正限位）类型”不可配置为正限位以外的其它信号类型。“（正限位）索引号”下拉列表选择正限位信号的编号，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_LmtsOff 设置，限位开关有效可以通过指令　GTN_LmtsOn 设置。

(3) 负限位信号选择：负限位信号由数字量输入（0x60FD）的 bit0 解析而来，图 4-6“（负限位）类型”不可配置为负限位以外的其它信号类型。“（负限位）索引号”下拉列表选择负限位信号的编号，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_LmtsOff 设置，限位开关有效可以通过指令　GTN_LmtsOn 设置。

注意	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 4-7　axis 配置界面 2

3. 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

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。

4. 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

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。

4.3.2　配置 profile

profile 配置界面如图 4-8 所示，配置平滑停止减速度和紧急停止减速度，该项可以通过指令 GTN_SetStopDec 设置。GEN 控制器没有本地 IO 资源，所以“xxx 类型”和“xxx 索引号”的配置无效。

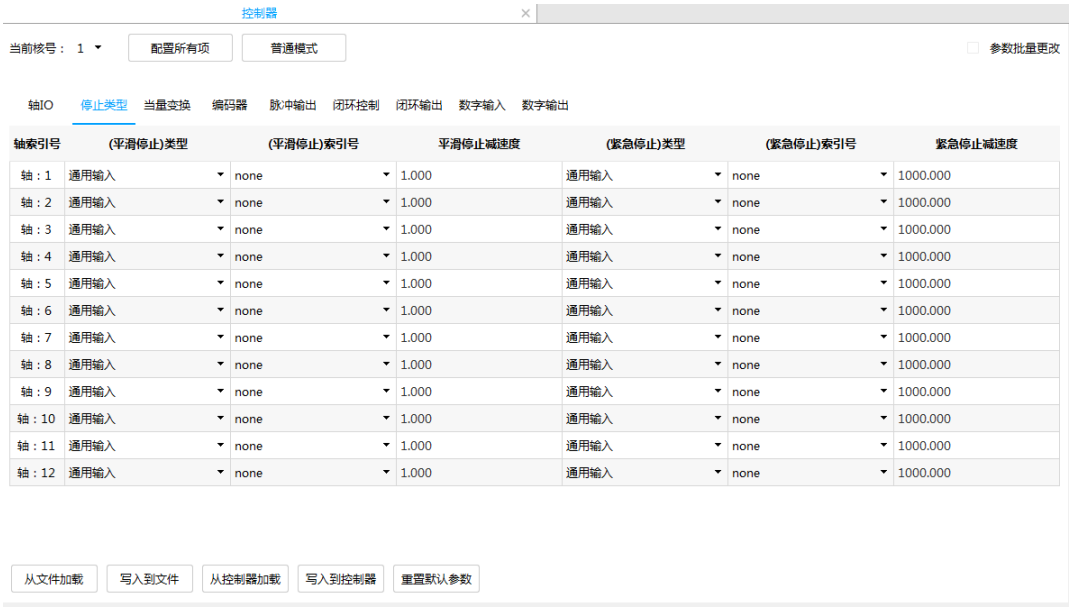


图 4-8　profile 配置界面

4.3.3　配置 encoder

Encoder 配置界面如图 4-9 所示。

图 4-9　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 4-10 所示。

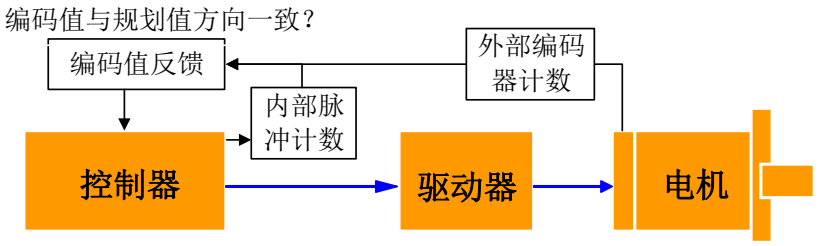


图 4-10　encoder 配置对控制系统的影响

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 4-11 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 4-11　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，因为 EtherCAT 总线发送的是网络报文，无法计算输出的脉冲个数，即没有脉冲计数器，必须设置成外部编码器计数。

4. 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. EtherCAT 控制器的 Home 和 Index 信号不直接给到控制器，所以配置无效。

4.4　配置文件生成和下载

表 4-1　下载配置文件指令

指令	说明
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

按照 4.3 节中的描述进行运动控制器的配置之后，如图 4-12 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）。

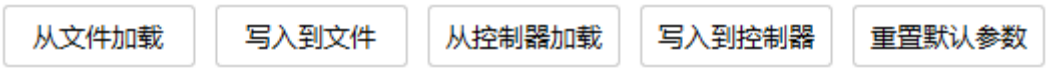


图 4-12　生成配置文件界面

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。

4.5　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。

4.5.1　指令列表

表 4-2　配置信息修改指令列表

指令	说明
GTN_AlarmOff	控制轴驱动报警信号无效
GTN_AlarmOn	控制轴驱动报警信号有效
GTN_LmtsOn	控制轴限位信号有效
GTN_LmtsOff	控制轴限位信号无效
GTN_ProfileScale	设置控制轴的规划器当量变换值
GTN_EncScale	设置控制轴的编码器当量变换值
GTN_SetStopDec	设置平滑停止减速度和急停减速度
GTN_GetStopDec	读取平滑停止减速度和急停减速度
GTN_SetSense	设置运动控制器编码器的电平逻辑

4.5.2　重点说明

(1)　打开运动控制器，　GTN_Open;

(2)　复位运动控制器，　GTN_Reset。

(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_LmtsOff，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，需通过驱动器调试软件修改；

(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。

4.5.3　例程

例程 4-1　初始化配置控制器

```
……
// 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);
// 设置编码器计数方向
sRtn =    GTN_SetSense (core, MC_ENCODE ,1,1);
// 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_LmtsOff(1,1,-1);
commandhandler("   GTN_LmtsOff", sRtn);
// 配置完成后要更新状态
sRtn =    GTN_ClrSts(1,1,1);
commandhandler("   GTN_ClrSts", sRtn);
// 轴1伺服使能
sRtn =    GTN_AxisOn(1,1);
commandhandler("   GTN_AxisOn", sRtn);
……
```

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。

4.6　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 4-3 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 4-3　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
	正负限位报警	有效，编号与轴号对应	GTN_LmtsOn GTN_LmtsOff
	驱动报警	有效，编号与轴号对应	GTN_AlarmOn GTN_AlarmOff
profile	该资源是否激活	激活	/
	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
encoder	该资源是否激活	激活	/
	输入脉冲反转	取反	GTN_SetSense

第5章 EtherCAT 指令说明

5.1　本章简介

GEN 控制器采用双核专用处理器(core1 和 core2)，目前只使用 core1 用于 EtherCAT 总线通讯，所以后续所有带 core 参数的指令的值都应设为 1。本章介绍 GEN 控制器使用 EtherCAT 总线相关的指令。

注意	本章表格中的“页码”即为此指令在“第 12 章 指令详细说明”中的对应页码。可以使用“超级链接”进行索引。 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　EtherCAT 库指令

5.2.1　指令列表

表 5-1　EtherCAT 库指令列表

指令	说明	页码
GTN_InitEcatComm	EtherCAT 初始化	150
GTN_StartEcatComm	启动 EtherCAT 通讯	173
GTN_TerminateEcatComm	结束 EtherCAT 通讯	176
GTN_IsEcatReady	查询 GUC EtherCAT 通讯状态	150
GTN_SetEcatHomingPrm	设置 EtherCAT 轴回零的参数	162
GTN_SetHomingMode	切换 EtherCAT 轴的回零模式	166
GTN_StartEcatHoming	启动 EtherCAT 轴回零	173
GTN_StopEcatHoming	停止 EtherCAT 轴回零	175
GTN_GetEcatHomingStatus	查询 EtherCAT 轴的回零状态	137
	设置探针功能的参数（参数方式）	170

GTN_SetTouchProbeFunction		
GTN_SetTouchProbeFunctionEx	设置探针功能的参数（功能描述方式）	171
GTN_GetTouchProbeStatus	查询 EtherCAT 轴的探针状态和捕获值	147
GTN_EcatIOReadInput	读取 EtherCAT IO 模块的输入值	125
GTN_EcatIOWriteOutput	写入 EtherCAT IO 模块的输出值	125
GTN_GetEcatAxisAI	读取 EtherCAT 轴的模拟量输入	134
GTN_GetEcatAxisDI	读取 EtherCAT 轴的数字量输入	134
GTN_SetEcatAxisDO	设置 EtherCAT 轴的数字量输出	161
GTN_GetEcatAxisDO	读取 EtherCAT 轴的数字量输出	134
GTN_SetPosScale	设置编码器倍率值	168
GTN_GetPosScale	读取编码器倍率值	142
GTN_GetEcatEncPos	读取 EtherCAT 轴的编码器位置	136
GTN_GetEcatEncVel	读取 EtherCAT 轴的编码器速度	137
GTN_GetEcatAxisPE	读取 EtherCAT 轴的规划和编码器误差	135
GTN_SetEcatAxisMode	设置 EtherCAT 轴的操作模式	162
GTN_GetEcatAxisMode	读取 EtherCAT 轴的操作模式	135
GTN_SetEcatAxisPT	设置 EtherCAT 轴的力矩	162
GTN_GetEcatAxisAtlTorque	读取 EtherCAT 轴的力矩值	134
	读取 EtherCAT 轴的电流值	134

GTN_GetEcatAxisAtlCurrent		
GTN_SetEcatAxisPV	设置 EtherCAT 轴的速度	162
GTN_GetEcatSlaves	读取 EtherCAT 总线的在线从站数目	137
GTN_GetMcEcatAxisNum	读取 EtherCAT 伺服轴数	140
GTN_GetEcatRawData	读取 EtherCAT 总线的 PDO（Process Data Object）	137
GTN_SetEcatRawData	设置 EtherCAT 总线的 PDO（Process Data Object）	164
GTN_EcatSDOUpLoad	读取 EtherCAT 总线的 SDO（Service Data Object）	126
GTN_EcatSDODownload	设置 EtherCAT 总线的 SDO（Service Data Object）	126

5.2.2　重点说明

运动控制器上电初始化之后，程序必须首先调用 GTN_IsEcatReady 指令，pStatus 返回值为 1 时表示 EtherCAT 从站和运动控制器实现正常通讯，返回其他值表示通讯未建立。

(1) **EtherCAT 通讯**

首先使用 EtherCATConfig 工具配置正确的 eni 文件，并将此 eni 文件存放在可执行文件相同目录中，具体配置方法详见 EtherCATConfigTool 相关资料。

使用 GTN_Open 打开控制器之后，调用 GTN_InitEcatComm 指令进行 EtherCAT 总线初始化，并使用 GTN_StartEcatComm 指令启动总线通讯。至此，可以通过相关指令对 EtherCAT 伺服和 EtherCAT IO 进行控制。

关闭 EtherCAT 通讯，可使用 GTN_TerminateEcatComm 指令。建议在退出程序时，先关闭 EtherCAT 通讯，再使用 GTN_Close 指令关闭控制器。

(2) **回零**

EtherCAT 轴的回零，控制器只对驱动器做设置回零参数、切换回零模式、启动回零、读取回零状态、停止回零操作，除此之外回零的整个过程由驱动器完成。

控制器可以通过 PDO 或 SDO 发送相关指令给驱动器。如果通过 PDO 方式发送指令，则驱动器的对象字 6060、6061 必须要支持 PDO 操作，即在配置驱动器 PDO 时，将 6060、6061 勾选上，如图 5-1 所示。如果驱动器的对象字 6060 和 6061 不能配置成 PDO，则按默认方式，即 SDO 通讯。

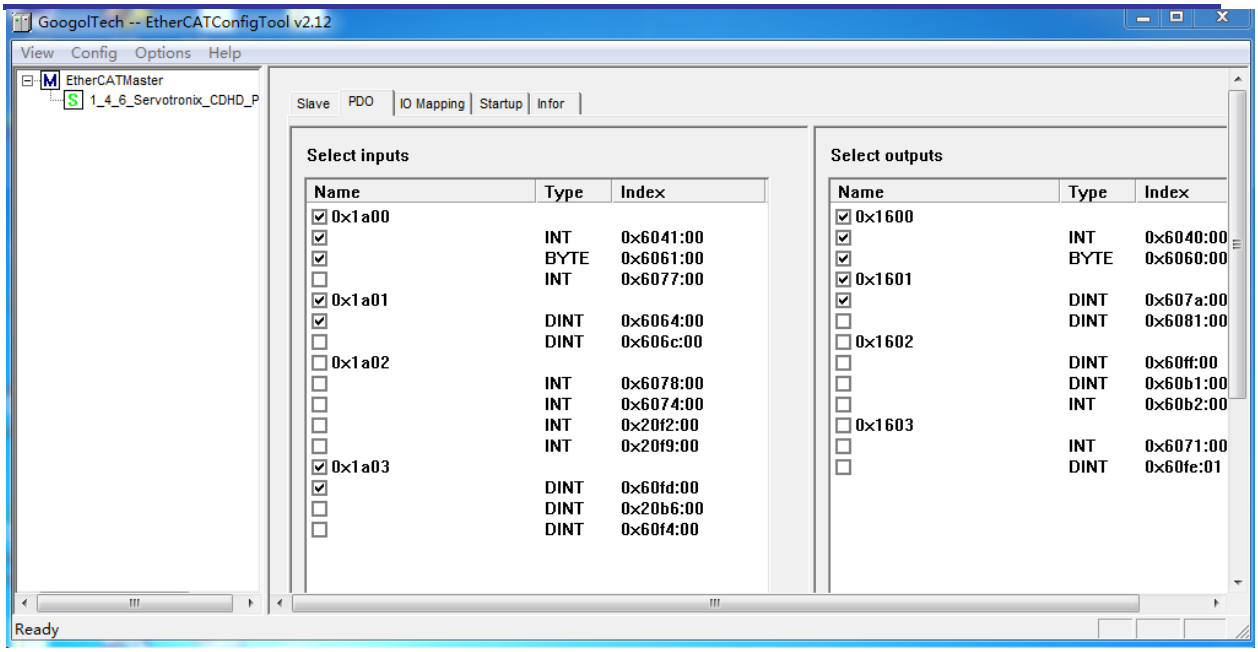


图 5-1　PDO 配置界面

GTN_SetEcatHomingPrm 指令设置回零参数，具体参数说明见指令详细列表，其中回零方式由驱动器决定，不同驱动器支持的回零方式不同，要根据具体驱动器来选择回零方式。其中的 probeFunction 参数只有在使用探针相关回零方式时才有效。

注意	启动回零的三个必要条件： 1. 目标驱动器处于伺服使能状态； 2. 目标驱动器的操作模式是回零模式； 3. 已成功设置回零参数。 如果以上三个条件之一未满足，指令 GTN_StartEcatHoming 将返回 1，并不做任何动作。

(3) 探针功能

控制器的探针捕获功能通过驱动器实现，控制器负责设置探针捕获方式和读取探针捕获状态。

控制器支持使用指令 GTN_SetTouchProbeFunction 和 GTN_SetTouchProbeFunctionEx 两种方式设置探针功能，区别只在于探针功能参数是按位表示还是具体描述方式。一般驱动器的探针功能支持单次捕获和连续捕获，单次即只记录一次位置值，连续捕获则自动刷新捕获位置值。捕获沿分为上升沿和下降沿，捕获的位置值分别存储于不同的参数中，使用指令 GTN_GetTouchProbeStatus 可读取捕获状态和捕获位置值。

注意	探针捕获状态置位后，需要关闭探针功能才能复位。因此，连续捕获模式下，需要借助用户自定义状态位来判断是否触发新的捕获，可查阅驱动器手册。例如，GTHD 将第 6 位作为上升沿捕获状态刷新标识。

(4) PDO 操作

控制器支持使用指令 GTN_GetEcatRawData 和 GTN_SetEcatRawData 直接对 EtherCAT 总线的 PDO 数据进行读写操作，但是要求用户熟知各对象字典的含义以及数据长度。PDO 偏移地址的计算方法如下：

1) 如图 5-2 所示连接 4 个 EtherCAT 从站，其中 Slave0 和 Slave1 是 GTHD 伺服从站，Slave2 和 Slave3 是 IO 从站。各从站的 PDO 信息可在【IO Mapping】项目中查看。

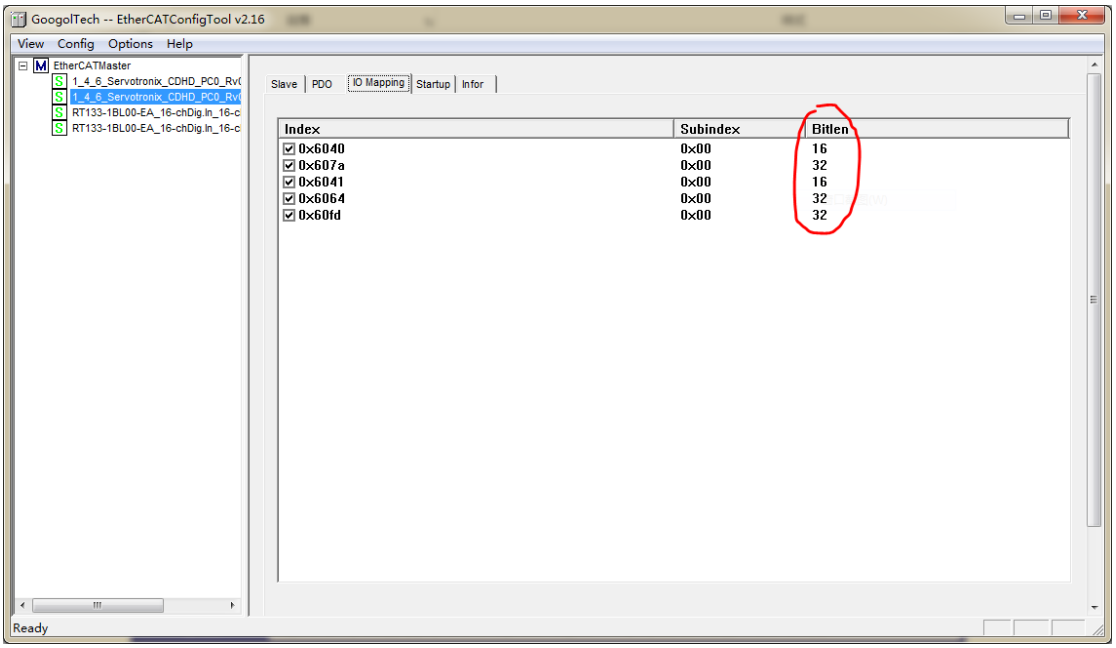


图 5-2　GTHD 默认配置之 PDO 信息

2) Slave0 的 PDO 首地址是 0，之后的 PDO 偏移地址依次增加。首先假设 Slave0 和 Slave1 的 PDO 配置相同，都如上图所示。则 Slave0 的 PDO 0x6040 的偏移地址是 0 byte；0x6041 的偏移地址是 6 bytes；Slave1 的 PDO 0x6040 的偏移地址是 16 bytes；0x6041 的偏移地址是 22 bytes；Slave2 的 PDO 首地址是 32 bytes；Slave3 的 PDO 首地址是 36 bytes。

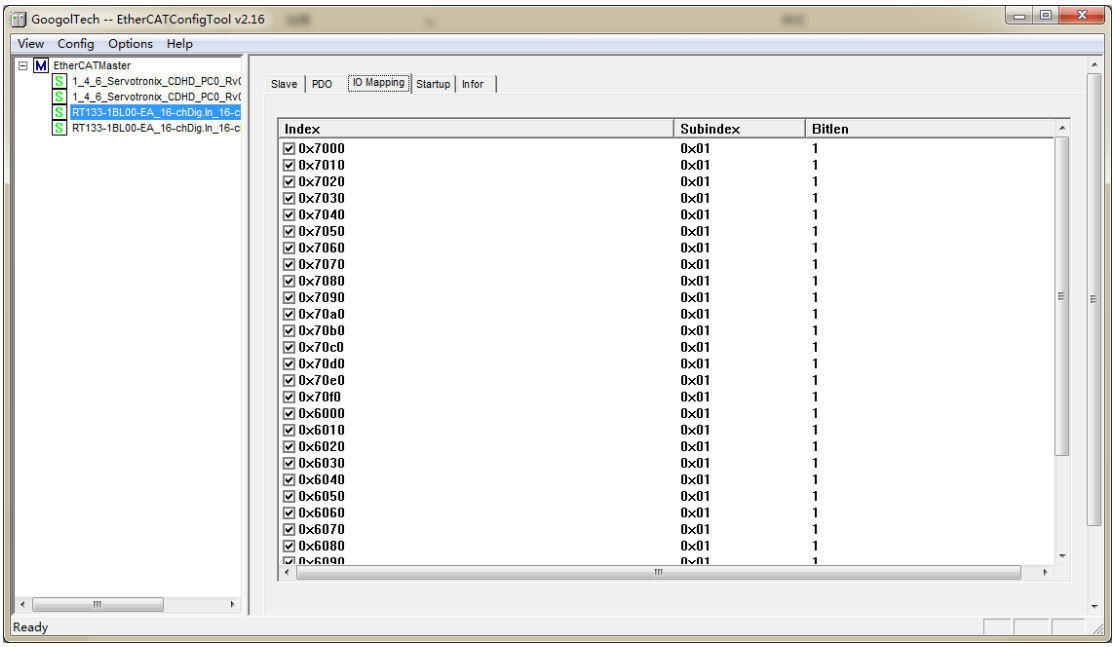


图 5-3　EtherCAT IO 默认配置之 PDO 信息

3) 读取 Slave2（第一个 IO 模块）的 16 位输入，可调用指令　GTN_GetEcatRawData(34, 2, &wInput)；设置 Slave2 的 16 位输出，可调用指令　GTN_SetEcatRawData(32, 2,

&wOutput)；读取 Slave3 的 16 位输入，可调用指令 GTN_GetEcatRawData(38, 2, &wInput)；设置 Slave2 的 16 位输出，可调用指令 GTN_SetEcatRawData(36, 2, &wOutput)。

(5) EtherCAT IO 的使用

EtherCAT IO 的数值可通过(4)章节描述的 PDO 的方式读取，也可以通过专用的 GTN_EcatIOReadInput 和 GTN_EcatIOWriteOutput 指令读取。以 EtherCAT 耦合器（IBM337-001-EC）、数字量输入模块（IBM311-160）、数字量输出模块（IBM321-160）、模拟量输入模块（IBM312-041）和模拟量输出模块（IBM322-041）顺序连接为例，PDO 配置信息如图 5-4 所示。可见耦合器连接的所有模块共同组成一个 EtherCAT 从站，并且从站内的 PDO 配置和实际连接的模块顺序相匹配。

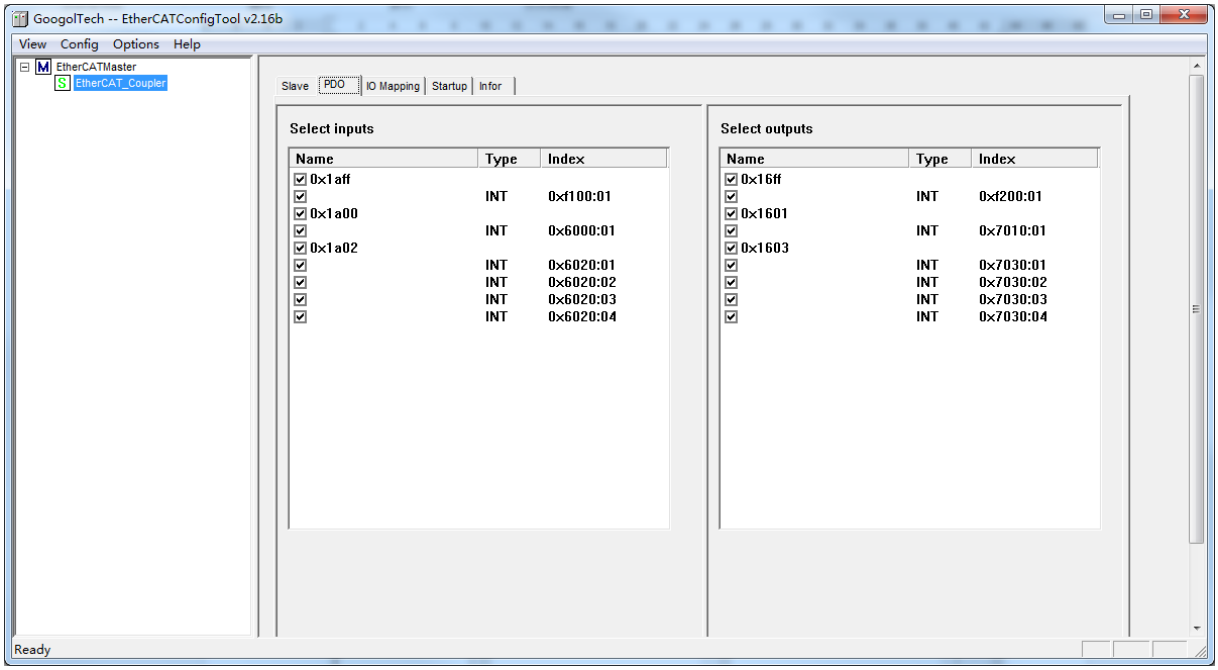


图 5-4　EtherCAT 耦合器和 IO 的 PDO 配置信息

首先，0x6000:01 和 0x6020:01~04 分别是数字量输入模块（IBM311-160）和模拟量输入模块（IBM312-041）的对象字典，0x7010:01 和 0x7030:01~04 分别是数字量输出模块（IBM321-160）和模拟量输出模块（IBM322-041）的对象字典。再看 IO Mapping 信息，如图 5-5 所示，所有的 output 数据都列在前面，input 数据都在后面，此时计算各自的偏移量就需要根据 IO Mapping 的信息进行。可见，0x7010:01 的偏移地址为 2bytes，0x7030:01~04 的偏移地址为 4bytes，0x6000:01 的偏移地址为 14bytes，0x6020:01~04 的偏移地址为 16bytes。

如果读取 IBM311-160 的输入值，调用指令 GTN_EcatIOReadInput(1, 0, 14, 2, &wDInput)；

如果写入 IBM321-160 的输出值，则调用指令 GTN_EcatIOWriteOutput(1, 0, 2, 2, &wDOutput)；

如果读取 IBM312-041 的第一个通道的模拟量输入值，调用指令 GTN_EcatIOReadInput(1, 0, 16, 2, &wAInput1)；如果读取 IBM312-041 的第二个通道的模拟量输入值，调用指令

GTN_EcatIOReadInput(1, 0, 18, 2, &wAInput2)；同时读取 IBM312-041 的第三和第四通道的模拟量输入值，调用指令　GTN_EcatIOReadInput(1, 0, 20, 4, &arrAInput34)；同时写入 IBM322-041 四个通道的模拟量输出值，调用指令　GTN_EcatIOWriteOutput(1, 0, 4, 8, &arrAOutput1234)。具体指令调用参考例程 5-4。

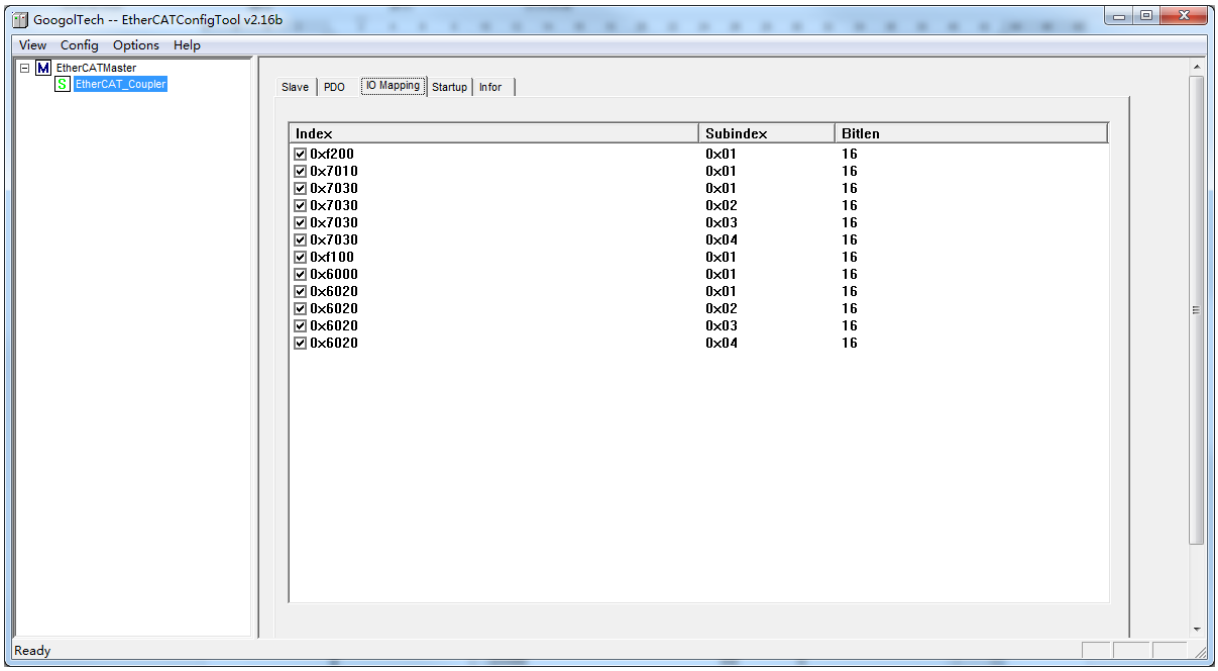


图 5-5　EtherCAT 耦合器和 IO 的 IO Mapping 信息

5.2.3　例程

例程 5-1　EtherCAT 初始化

```
#include <stdio.h>
#include "gts.h"
int main(int argc,char *argv[])
{
    short rtn;
    short status;

    rtn =    GTN_Open(CHANNEL_PCIE);

    if(rtn){
        printf("Not find ecat device\n");
        return -1;
    }

    rtn =    GTN_InitEcatComm(1);

    if(rtn){
        printf("EtherCAT communication error\n");
        return -1;
```

```
    }
    //*wait untill EtherCAT comminication OK*/
    do {
        rtn =    GTN_IsEcatReady(1, &status);
    }
    while(status != 1 || rtn != 0);

    rtn =    GTN_StartEcatComm(1);

    rtn =    GTN_Reset(1);

    rtn =    GTN_LoadConfig(1, "\\hard disk\\CPAC\\GTS800.cfg");
    //TODO: add other GT command below
    return 0;
}
```

注：程序的执行目录下需要包含 Gecat.eni 配置文件，此文件的生成请参考配置工具手册；同时 GTS800.cfg 可使用 MotionStudio 生成。

例程 5-2　采用 3 号回零方式

```
#define AXIS                1           // 定义轴号
int main(int argc, char* argv[])
{
    short sRtn;                         // 指令返回值变量
    long lAxisStatus;                   // 轴状态
    short sHomeSts;                     // 回零状态
    bool bStop = false;
    ..... .....

    sRtn =    GTN_ClrSts(1,AXIS);

    sRtn =    GTN_AxisOn(1,AXIS);

    do {
        sRtn =    GTN_GetSts(1, AXIS, &lAxisStatus, 1);
    } while(!lAxisStatus.9);
    // 必须处于伺服使能状态，切换到回零模式
    sRtn =    GTN_SetHomingMode(1, AXIS, 6);
    // 设置回零参数
    sRtn =    GTN_SetEcatHomingPrm(1, AXIS, 3, 5000, 3000, 100000, 0, 0);
    // 启动回零
```

```
    sRtn =    GTN_StartEcatHoming(1, AXIS);
    do {
        sRtn:=    GTN_GetEcatHomingStatus(1, AXIS, &sHomeSts);
        if (bStop)//中断、停止回零
        {
            sRtn:=    GTN_StopEcatHoming(1, AXIS);
            break;
        }
    } while(3 != sHomeSts);// 等待搜索原点完成
    sRtn =    GTN_SetHomingMode(1, AXIS, 8);  // 切换到位置控制模式
    return 0;
}
```

例程 5-3　探针 1 上升沿连续捕获（以 GTHD 为例）

```
#define AXIS                    1               // 定义轴号
int main(int argc, char* argv[])
{
    short sRtn;                                 // 指令返回值变量
    short sProbePrm;                            // 探针参数
    unsigned short usProbeSts;                                  // 探针捕获状态
    long lRiseValue1, lRiseValue2, lFallValue1, lFallValue2;   // 探针捕获值
    long lLastValue;
    bool bPreSts = false;
    bool bStopProbe = false;

    ..... .....
    sProbePrm = 0x0013;    // 探针 1 上升沿连续捕获，探针 2 无效
    sRtn =    GTN_SetTouchProbeFunction(1, AXIS, sProbePrm);
    do {
        sRtn =    GTN_GetTouchProbeStatus(1, AXIS, &usProbeSts,
            &lRiseValue1,    &lFallValue1, &lRiseValue2, &lFallValue2);
        if (usProbeSts.0)               // 探针 1 有效
        {
            if (usProbeSts.1)           // 探针 1 上升沿捕获触发
            {
                if (usProbeSts.6 != bPreSts)  // 再次捕获触发，实际判断条件请查阅驱动
                器相关手册。GTHD 将第 6 位作为上升沿捕获状态刷新标识
                {
                    lLastValue = lRiseValue1;
                    bPreSts = usProbeSts.6;
```

```
                    }
                }
                if (usProbeSts.2) {  // 探针 1 下降沿捕获触发
                }
            }
            if (usProbeSts.8) {        // 探针 2 有效
            }

            if (bStopProbe)
            {
                sProbePrm = 0x00; // 终止探针捕获
                sRtn =    GTN_SetTouchProbeFunction(1, AXIS, sProbePrm);
                break;
            }
        } while(true);
        return 0;
    }
```

例程 5-4　EtherCAT IO 的使用

```
    #define SLAVE                   0              // 定义轴号
    int main(int argc, char* argv[])
    {
        short sRtn;                                // 指令返回值变量
        unsigned short wDInput;
        unsigned short wDOutput;
        unsigned short wAInput1, wAInput2;
        unsigned short arrAInput34[2];
        unsigned short arrAOutput1234[4];

        ..... .....
        // 读取数字量模块的输入值
        sRtn =    GTN_EcatIOReadInput(1, SLAVE, 14, 2, &wDInput);
        // 读取模拟量模块的输入值
        sRtn =    GTN_EcatIOReadInput(1, SLAVE, 16, 2, &wAInput1);
        sRtn =    GTN_EcatIOReadInput(1, SLAVE, 18, 2, &wAInput2);
        sRtn =    GTN_EcatIOReadInput(1, SLAVE, 20, 4, &wAInput34);

        // 写入数字量模块的输出值
        sRtn =    GTN_EcatIOWriteOutput(1, SLAVE, 2, 2, &wDOutput);
        // 写入模拟量模块的输出值
```

```
    sRtn =    GTN_EcatIOWriteOutput(1, SLAVE, 4, 8, &wAOutput1234);
    return 0;
}
```

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。

注意	本章表格中的“页码”即为此指令在“第 12 章 指令详细说明”中的对应页码。可以使用“超级链接”进行索引。 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
GTN_GetSts	读取轴状态
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
GTN_GetPrfMode	读取轴运动模式
GTN_GetPrfPos	读取规划位置
GTN_GetPrfVel	读取规划速度
GTN_GetPrfAcc	读取规划加速度
GTN_GetAxisPrfPos	读取轴(axis)的规划位置值
GTN_GetAxisPrfV	读取轴(axis)的规划速度值

指令	说明
el	
GTN_GetAxisPrfAcc	读取轴(axis)的规划加速度值
GTN_GetAxisEncPos	读取轴(axis)的编码器位置值
GTN_GetAxisEncVel	读取轴(axis)的编码器速度值
GTN_GetAxisEncAcc	读取轴(axis)的编码器加速度值
GTN_GetAxisError	读取轴(axis)的规划位置值和编码器位置值的差值
GTN_Stop	停止一个或多个轴的规划运动，停止坐标系运动

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

位	定义
	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

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 11.7 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　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》。

“4.2.2 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。

“4.2.2 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“8.3 平滑停止和急停”。

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。

本章表格中的“页码”即为此指令在“第 12 章 指令详细说明”中的对应页码。可以使用“超级链接”进行索引。

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。

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	156
GTN_PrfJog	设置指定轴为 Jog 模式	156
GTN_PrfGear	设置指定轴为电子齿轮模式	156
GTN_GetPrfMode	查询指定轴的运动模式	142

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
GTN_PrfTrap	设置指定轴为点位运动模式
GTN_SetTrapPrm	设置点位运动模式下的运动参数
GTN_GetTrapPrm	读取点位运动模式下的运动参数
GTN_SetPos	设置目标位置
GTN_GetPos	读取目标位置
GTN_SetVel	设置目标速度
GTN_GetVel	读取目标速度
GTN_Update	启动点位运动

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。

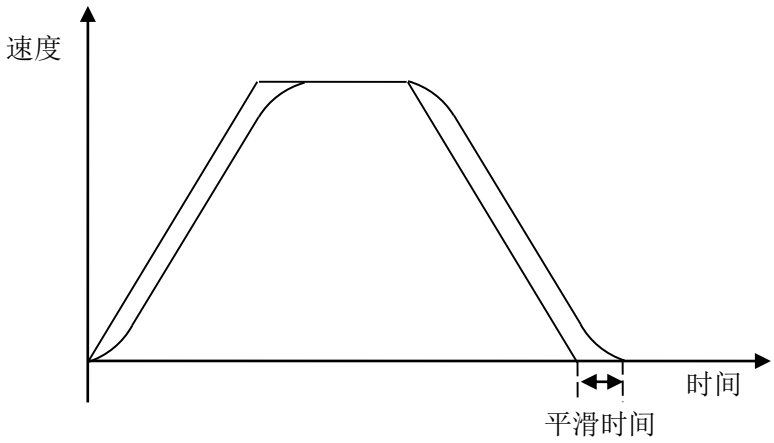


图 7-1　点位运动速度曲线

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。

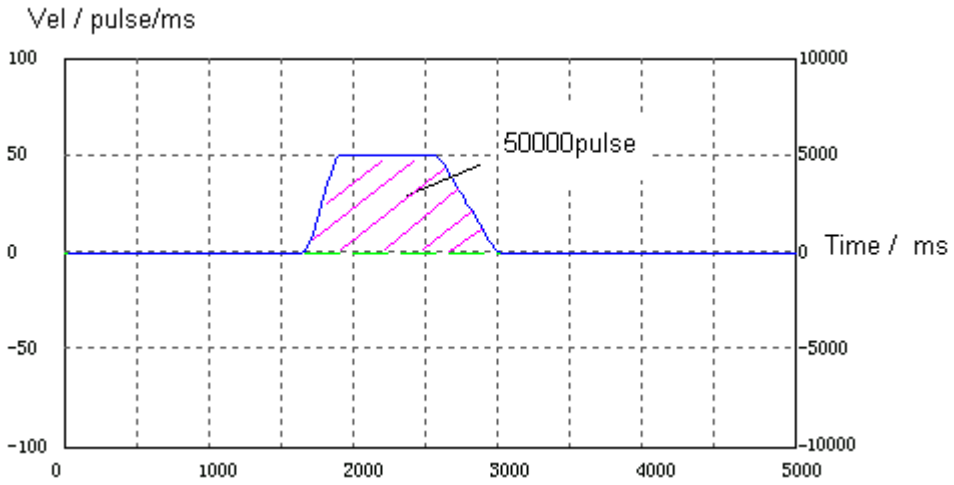


图 7-2　点位运动速度规划

```
#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
GTN_PrfJog	设置指定轴为 Jog 运动模式
GTN_SetJogPrm	设置 Jog 运动模式下的运动参数
GTN_GetJogPrm	读取 Jog 运动模式下的运动参数
GTN_SetVel	设置目标速度
GTN_GetVel	读取目标速度
GTN_Update	启动 Jog 运动

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。

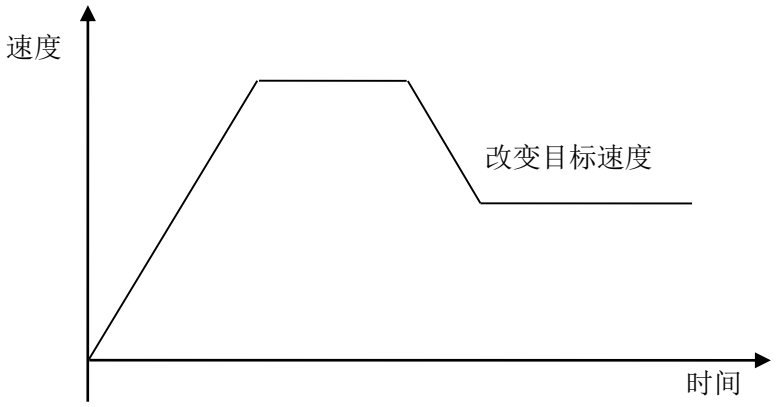


图 7-3　Jog 模式速度曲线

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。

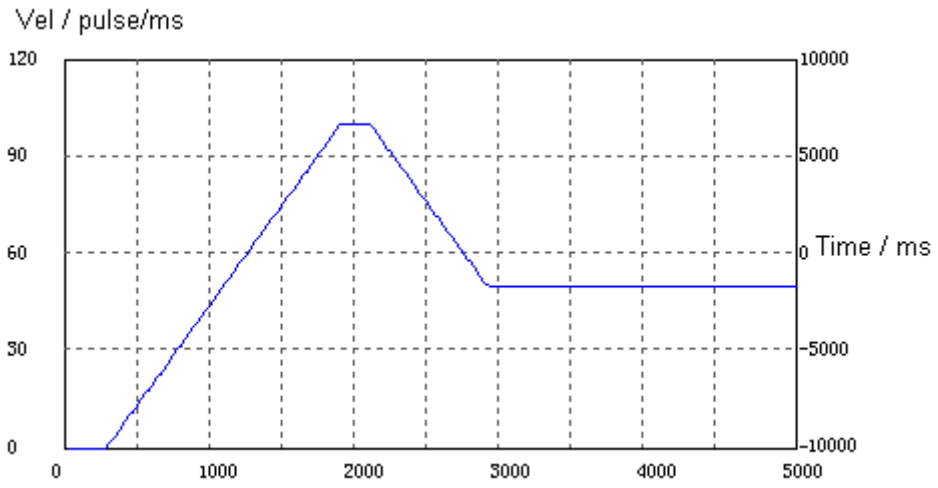


图 7-4　Jog 模式动态改变目标速度

```
#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
GTN_PrfGear	设置指定轴为电子齿轮运动模式
GTN_SetGearMaster	设置电子齿轮运动跟随主轴
GTN_GetGearMaster	读取电子齿轮运动跟随主轴
GTN_SetGearRatio	设置电子齿轮比
GTN_GetGearRatio	读取电子齿轮比
GTN_GearStart	启动电子齿轮运动

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。

注意	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

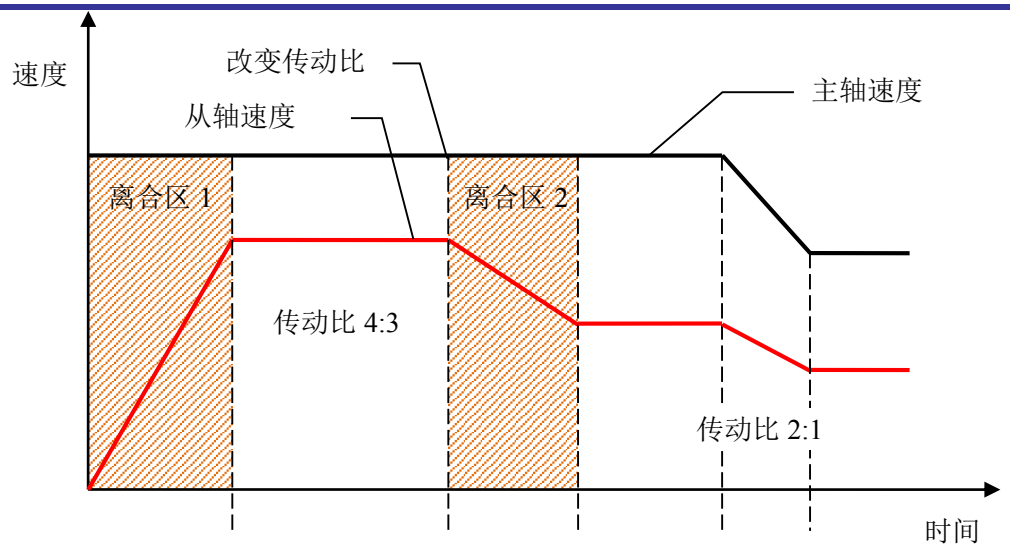


图 7-5　电子齿轮模式速度曲线

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(short profile, short dir)，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(short profile, short masterIndex, short masterType, short masterItem)。profile 为从轴轴号，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(short profile, long masterEven, long slaveEven, long masterSlope)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，应当调用以下指令：

GTN_SetGearRatio(slave, alpha, beta, slope);

(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 的工作模式请参考**错误!未找到引用源。**节内容

2）共享数据

通过指令 GTN_SetCoreShare(short core,short type,short index,short count)设置主轴共享资源。

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。

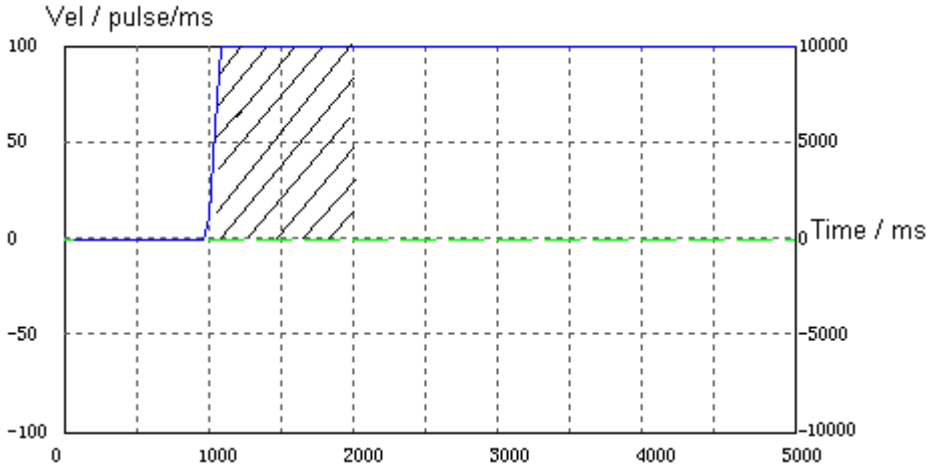


图 7-6　电子齿轮模式主轴速度规划

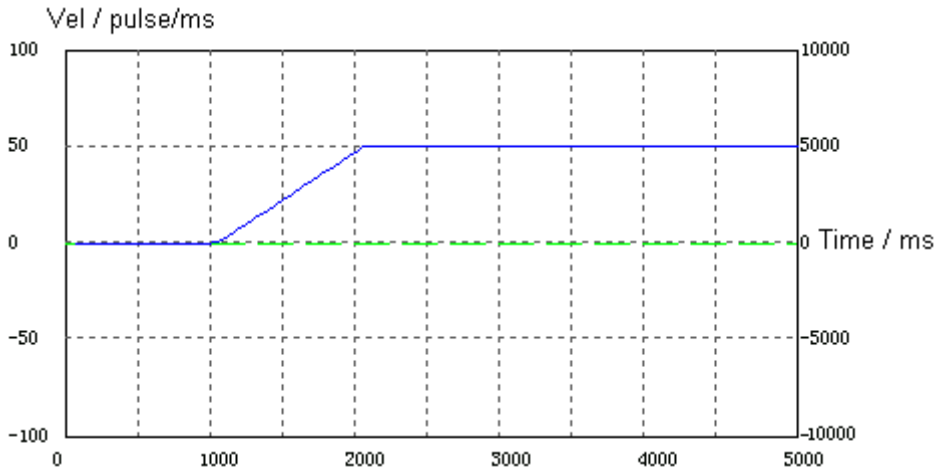


图 7-7　电子齿轮模式从轴速度规划

```
#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
GTN_SetCrdPrm	设置坐标系参数，确立坐标系映射，建立坐标系
GTN_GetCrdPrm	查询坐标系参数
GTN_CrdData	向插补缓存区增加插补数据
GTN_LnXY	缓存区指令，二维直线插补
GTN_LnXYZ	缓存区指令，三维直线插补
GTN_LnXYZA	缓存区指令，四维直线插补
GTN_LnXYG0	缓存区指令，二维直线插补(终点速度始终为 0)
GTN_LnXYZG0	缓存区指令，三维直线插补(终点速度始终为 0)
GTN_LnXYZAG0	缓存区指令，四维直线插补(终点速度始终为 0)

指令	说明
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)
GTN_ArcXYC	缓存区指令，XY 平面圆弧插补(以终点和圆心位置为输入参数)
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)
GTN_ArcYZC	缓存区指令，YZ 平面圆弧插补(以终点和圆心位置为输入参数)
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)
GTN_ArcZXC	缓存区指令，ZX 平面圆弧插补(以终点和圆心位置为输入参数)
GTN_BufIO	缓存区指令，缓存区内数字量 IO 输出设置指令
GTN_BufDelay	缓存区指令，缓存区内延时设置指令
GTN_BufDA	缓存区指令，缓存区内输出 DA 值
GTN_BufLmtsOn	缓存区指令，缓存区内有效限位开关
GTN_BufLmtsOff	缓存区指令，缓存区内无效限位开关
GTN_BufSetStopIo	缓存区指令，缓存区内设置 axis 的停止 IO 信息
GTN_BufMove	缓存区指令，实现刀向跟随功能，启动某个轴点位运动
GTN_BufGear	缓存区指令，实现刀向跟随功能，启动某个轴跟随运动
GTN_CrdSpace	查询插补缓存区剩余空间
GTN_CrdClear	清除插补缓存区内的插补数据
GTN_CrdStart	启动插补运动
GTN_CrdStatus	查询插补运动坐标系状态
GTN_SetUserSegNum	缓存区指令，设置自定义插补段段号
GTN_GetUserSegNum	读取自定义插补段段号
GTN_GetRemainderSegN	读取未完成的插补段段数

指令	说明
um	
GTN_SetOverride	设置插补运动目标合成速度倍率
GTN_SetCrdStopDec	设置插补运动平滑停止、急停合成加速度
GTN_GetCrdStopDec	查询插补运动平滑停止、急停合成加速度
GTN_GetCrdPos	查询该坐标系的当前坐标位置值
GTN_GetCrdVel	查询该坐标系的合成速度值
GTN_InitLookAhead	初始化插补前瞻缓存区
GTN_CrdHsOn	开启 DMA 传输通道
GTN_CrdHsOff	关闭 DMA 传输通道
GTN_GetCrdHsPrm	读取插补坐标系 DMA 设置参数

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。

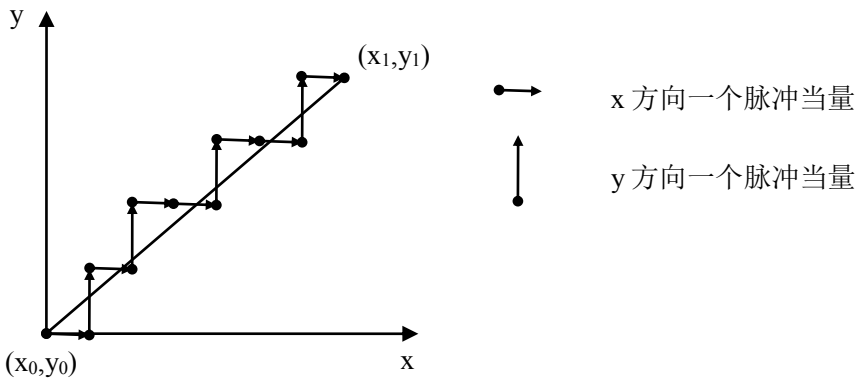


图 7-8　直线插补示意图

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。

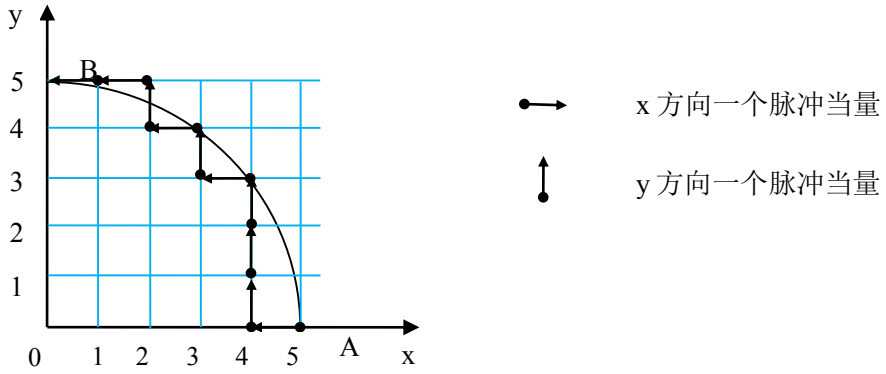


图 7-9　圆弧插补示意图

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。

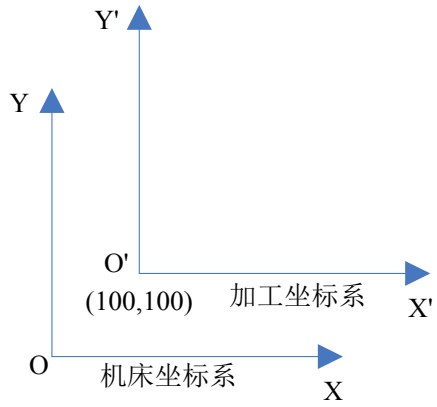
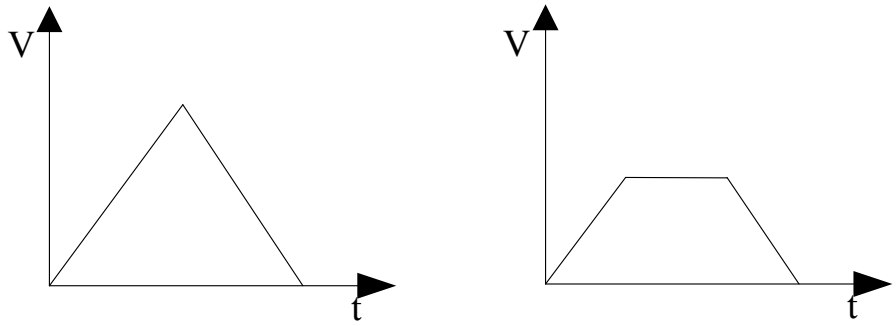


图 7-10　加工坐标系偏移量示意图

```
…… …… ……
// 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 GT_Ln 开头）和平面圆弧插补指令（以 GT_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。

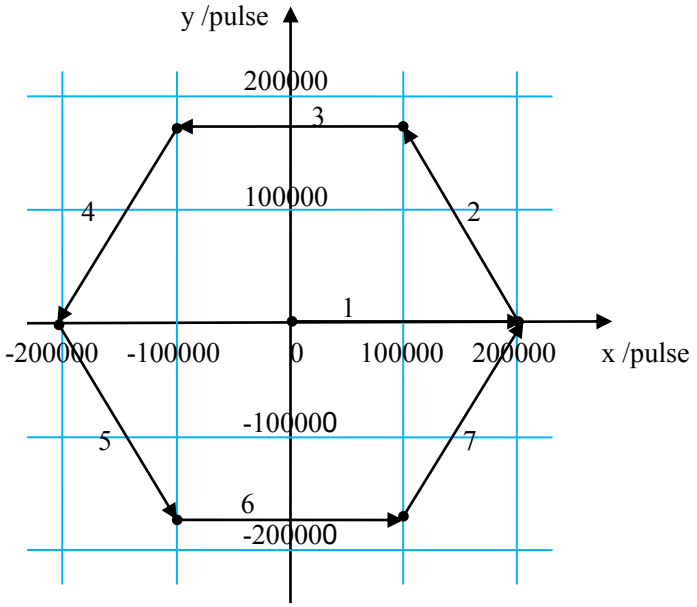


图 7-12　直线插补例程运动轨迹

```
…… …… ……
// 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。

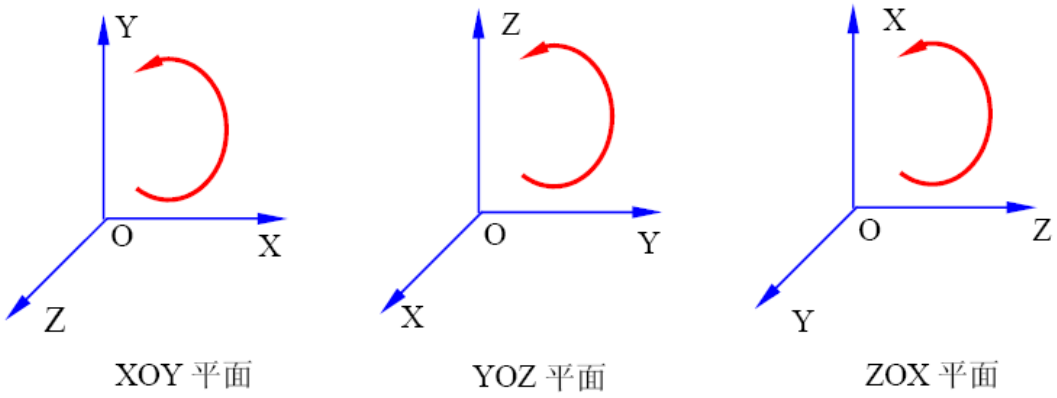


图 7-13　圆弧插补逆时针方向

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。

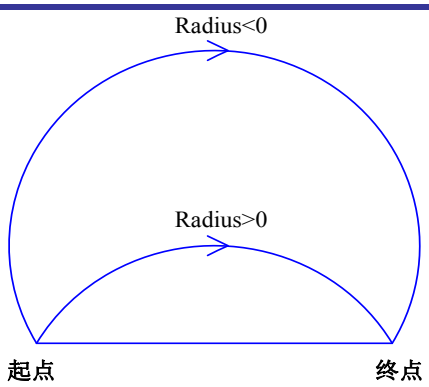


图 7-14　半径取正值/负值圆弧插补示意图

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。

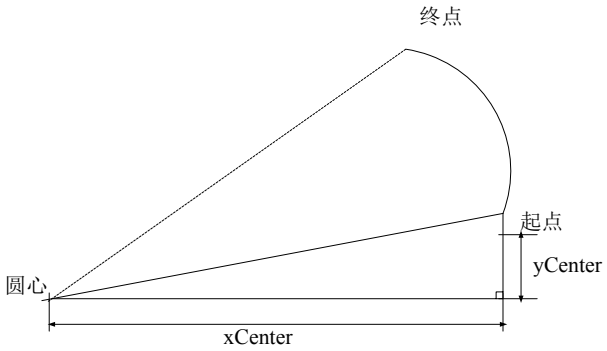


图 7-15　圆心坐标描述方法示意图

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。

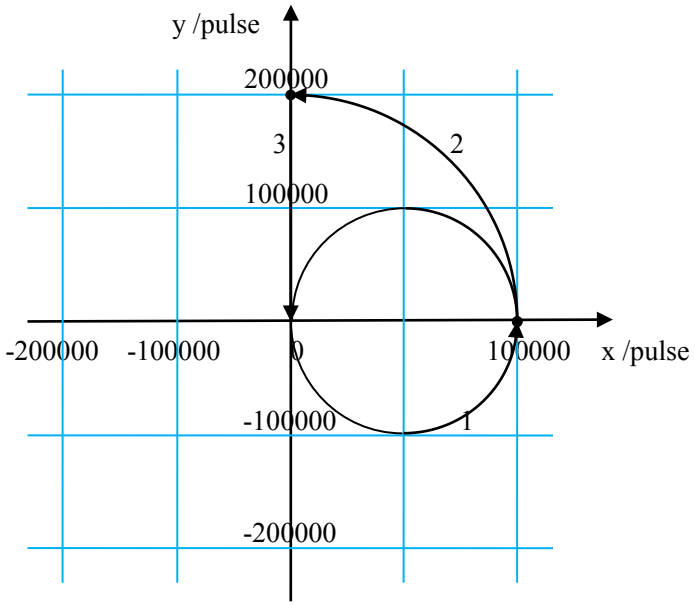


图 7-16　圆弧插补例程运动轨迹

```
…… …… ……
// 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. 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。

注意	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 7-17 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。

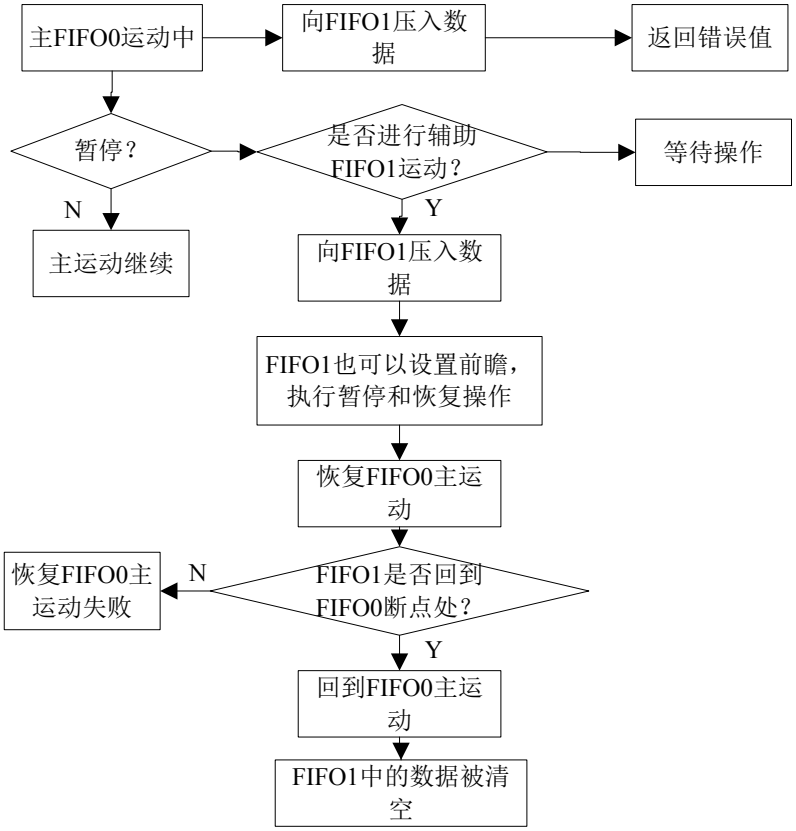


图 7-17　插补主运动与辅助运动流程

例程 7-7　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 7-18 所示。

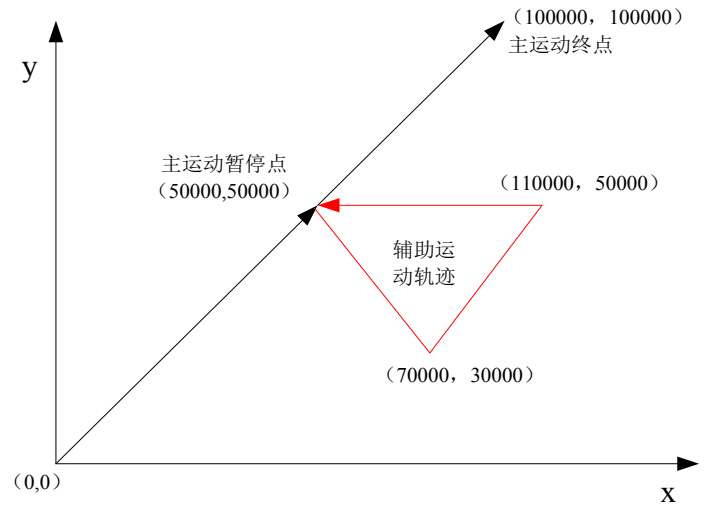


图 7-18　插补 FIFO 管理例程之换刀轨迹

```
…… …… ……
// 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);
```

6. 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 7-19 (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。

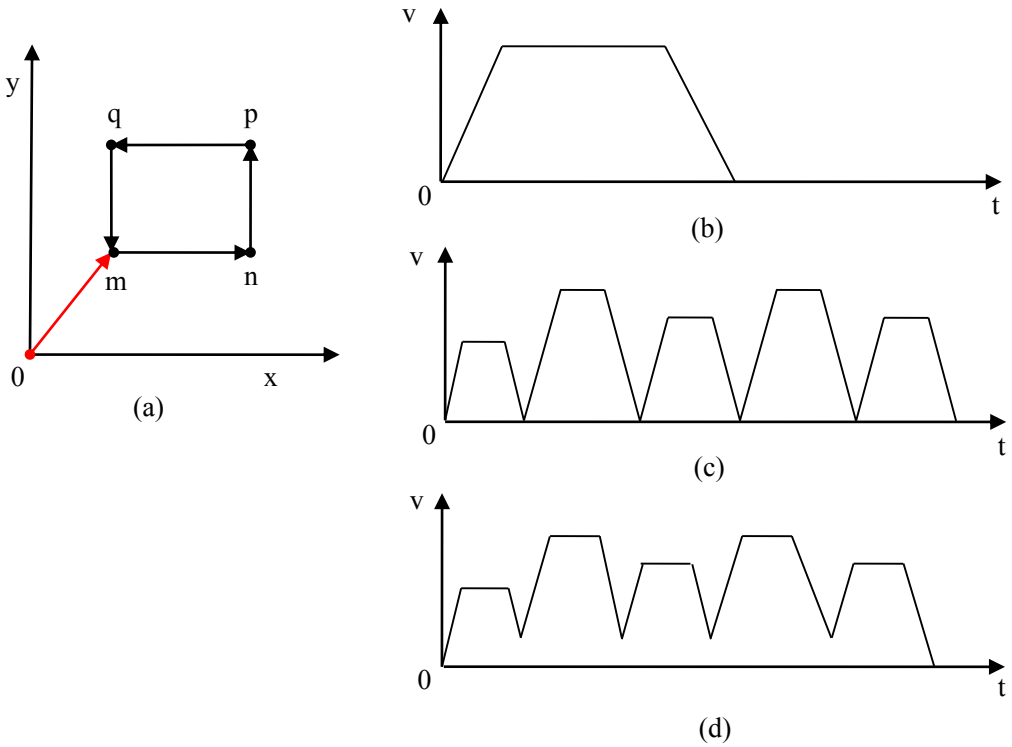


图 7-19　使用前瞻与不使用前瞻的速度规划区别

如果按照图 7-19 (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 7-19 (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 7-19(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 7-19 (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 7-19 (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 7-20 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 7-21 所示。

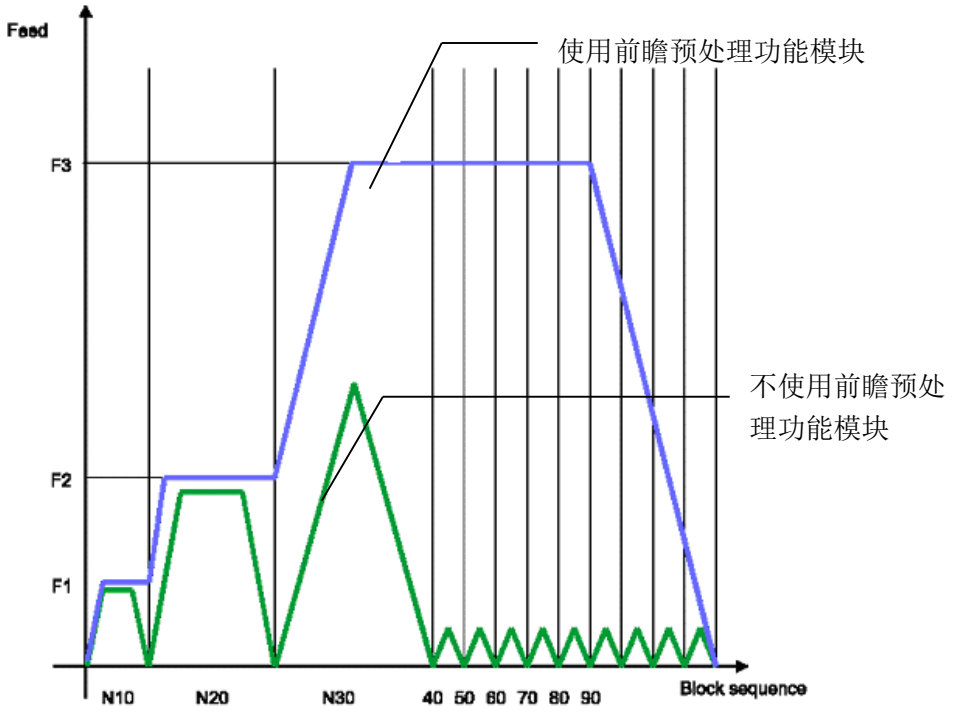


图 7-20　使用和不使用前瞻预处理功能模块的速度曲线对比图

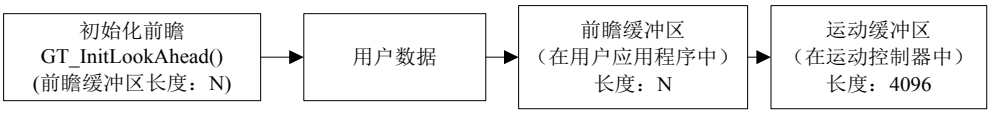


图 7-21　前瞻预处理流程图

(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 7-8　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 7-22 所示。红色线段为起始轨迹。

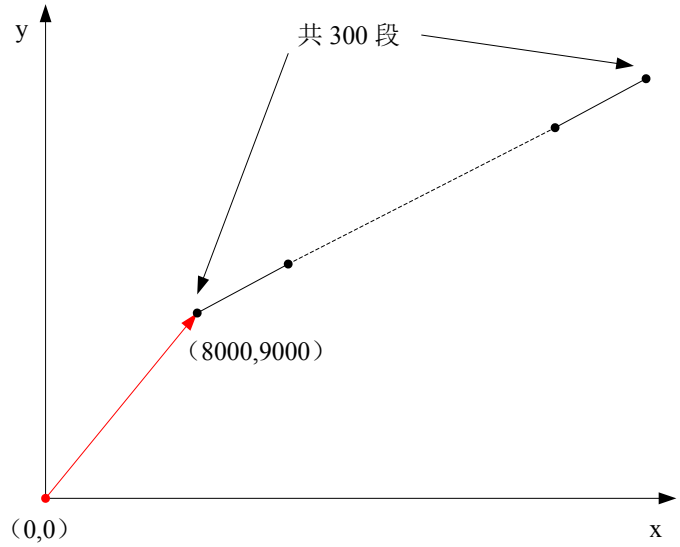


图 7-22　前瞻预处理例程之运动轨迹图

…… …… ……

// 指令返回值

```
short sRtn;
// 循环变量
int i;
// 定义前瞻缓存区内存区
TCrdDatacrdData[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 7-23 和图 7-24 所示。

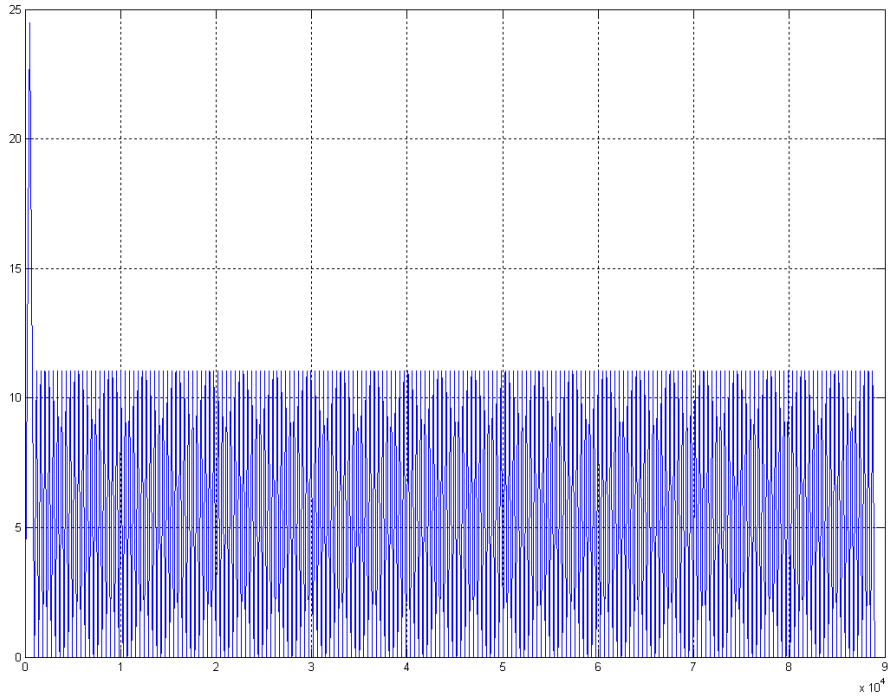


图 7-23　没有进行前瞻预处理的合成速度曲线

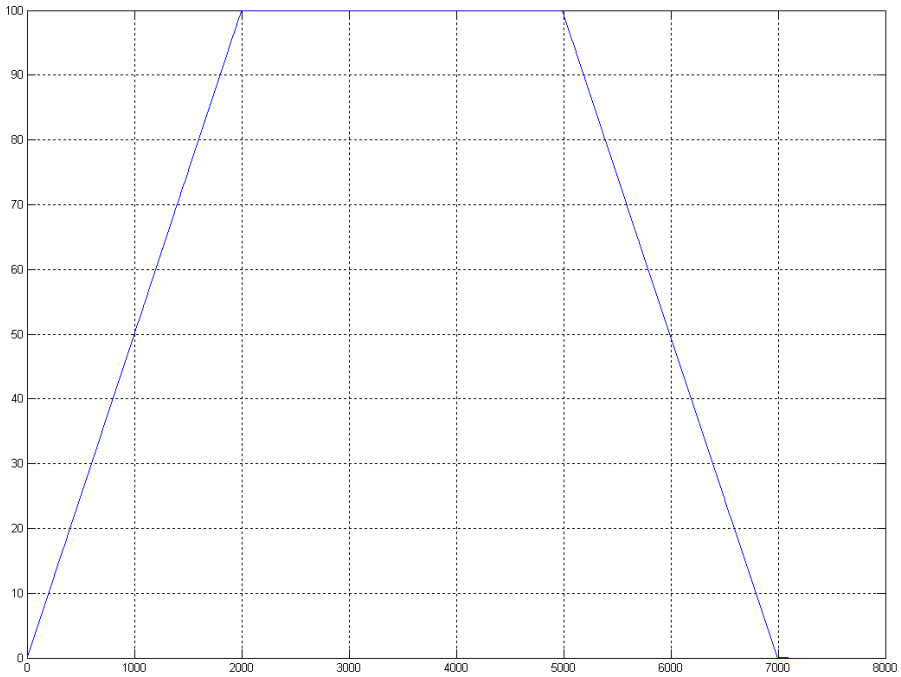


图 7-24　进行了前瞻预处理后的合成速度曲线

7.　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。

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 7-9　刀向跟随功能　GTN_BufMove

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 7-25 所示。

图 7-25　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
shor trun;
// 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

    // 启动运动

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

…… …… ……

例程插补合成速度和点位速度的运行结果如图 7-26 所示。

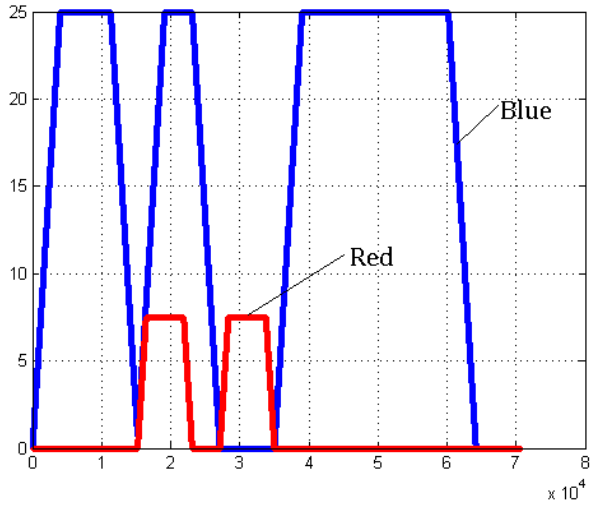


图 7-26　插补缓存区内的点位运动速度图

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 7-10　刀向跟随功能　GTN_BufGear

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 7-27 所示。

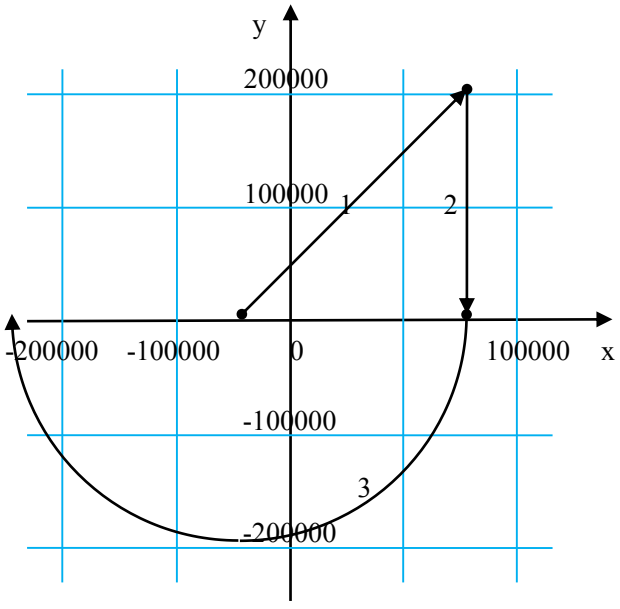


图 7-27　刀向跟随功能　GTN_BufGear 的运动轨迹

```
…… …… ……
// 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

    // 启动运动
    sRtn =    GTN_CrdStart(1,1, 0);
…… …… ……
```

例程的运行结果如图 7-28 所示。

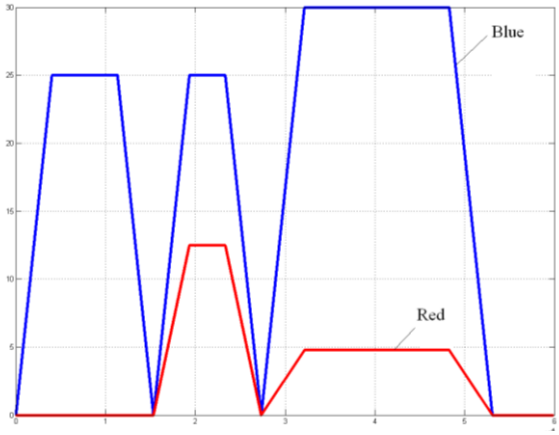


图 7-28　插补缓存区内的跟随运动速度图

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 7-11　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 7-29 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。

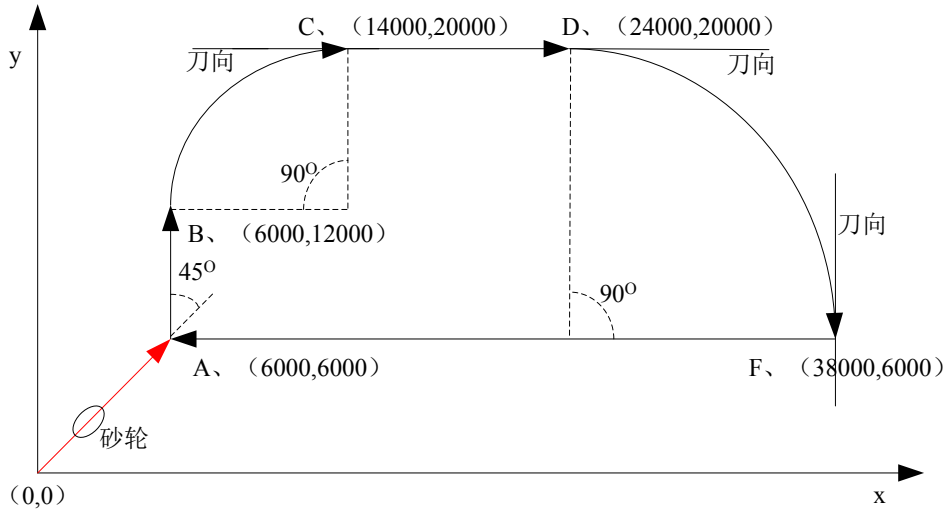


图 7-29　刀向跟随功能之工件尺寸和刀运动轨迹

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);
…… …… ……
```

8. 批量数据传输

为了提高插补指令的传输速率，增加 DMA 通道。

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;

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

    }
    do
    {

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

    } while (rtn == 1);

    // 关闭DMA功能

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

第8章 访问硬件资源

8.1　本章简介

本章表格中的“页码”即为此指令在“第 12 章 指令详细说明”中的对应页码。可以使用“超级链接”进行索引。

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
编码器	对外部编码器的脉冲输出进行计数
脉冲计数器	通过内部硬件计算输出的脉冲个数

本章介绍如何在应用程序中使用这些硬件资源。

8.2　访问编码器

8.2.1　指令列表

表 8-2　访问编码器指令列表

指令	说明
GTN_GetEncPos	读取轴编码器位置
GTN_GetEncVel	读取轴编码器速度
GTN_SetEncPos	修改轴编码器位置
GTN_SetSense	设置输入输出资源的电平逻辑。
GTN_GetSense	读取输入输出资源的电平逻辑。

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。如果用户正确的配置 EtherCAT 总线伺服，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 EtherCAT 伺服，或者使用步进电机时没有编码器反馈部件，如图 8-1 所示，则用户调用 GTN_GetEncPos 读取的编码器位置为 0。

图 8-1　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 4.3.3 ）。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。

8.2.2　例程

例程 8-1　读取轴编码器位置值

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

    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");
    }
    return 0;
}
```

8.3　访问内部脉冲计数器

8.3.1　指令列表

表 8-3　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	167
GTN_GetPlsPos	读取内部脉冲计数器位置	141
GTN_GetPlsVel	读取内部脉冲计数器速度	141

8.3.2　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 4.3.4 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。

8.3.3　例程

例程 8-2　访问内部脉冲计数器

```
#include "gts.h"
intmain(int argc, char* argv[])
{
    short sRtn, i;
    double plspos[12];

    sRtn =    GTN_Open();

    commandhandler("   GTN_Open", 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 安全机制

9.1　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。

注意	本章表格中的“页码”即为此指令在“第 12 章 指令详细说明”中的对应页码。可以使用“超级链接”进行索引。 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　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 9-1 所示。

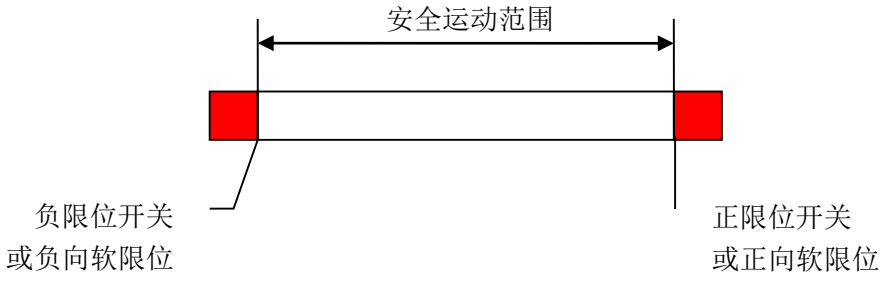


图 9-1　轴运动范围

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

9.2.1　指令列表

表 9-1　软限位指令列表

指令	说明
GTN_SetSoftLimitMode	设置软限位模式
GTN_GetSoftLimitMode	读取软限位模式
GTN_GetLimitStatus	读取限位状态
GTN_SetSoftLimit	设置轴正向软限位和负向软限位
GTN_GetSoftLimit	读取轴正向软限位和负向软限位

9.2.2　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，如何设置急停加速度请参见“4.5 配置信息修改指令”。

9.2.3　例程

例程 9-1　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，当运动超过正向软限位时，限位信号触发，运动停止。

```
#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);
// 轴使能
sRtn =    GTN_AxisOn(1,AXIS);
// 设置软限位
sRtn =    GTN_SetSoftLimit(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;
    }
```

9.3　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。

9.4　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。

9.5　掉电存储功能

9.5.1　指令列表

表 9-2　掉电功能指令列表

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

9.5.2　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。

9.5.3　例程

例程 9-2　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10章　运动程序

10.1　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。

注意	本章表格中的“页码”即为此指令在“第 12 章 指令详细说明”中的对应页码。可以使用“超级链接”进行索引。 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　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 10-1 所示。

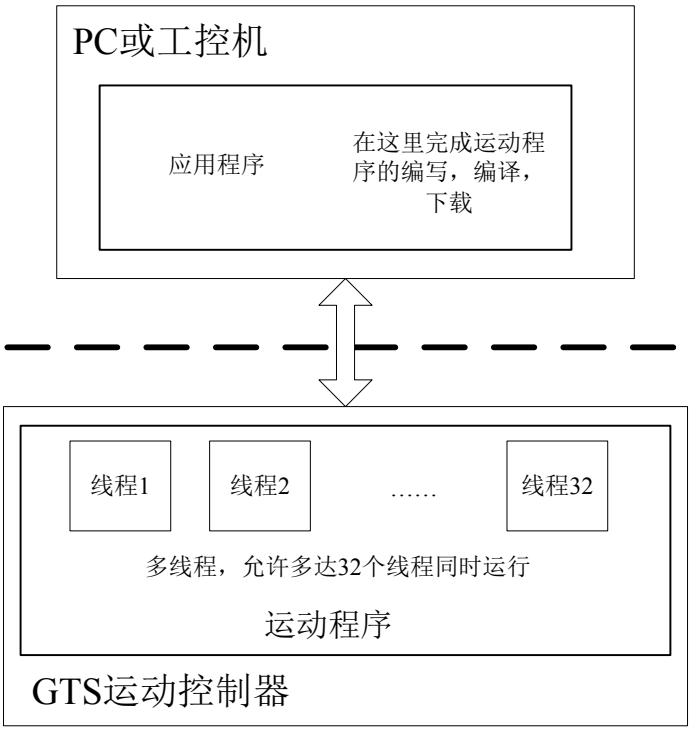


图 10-1　运动程序与应用程序的关系

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。

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。

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。

10.3　运动程序的使用

10.3.1 指令列表

运动程序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 10-1　运动程序指令列表

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

10.3.2 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“10.4.1 语言元素”、“10.4.2 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“10.5 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。

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);

10.3.3 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GTN_Compile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。

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。

执行 GTN_Compile 编译运动程序（*.c）之前，必须保证 error.ini 文件在相同目录下。

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

10.3.4 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。

10.3.5 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。

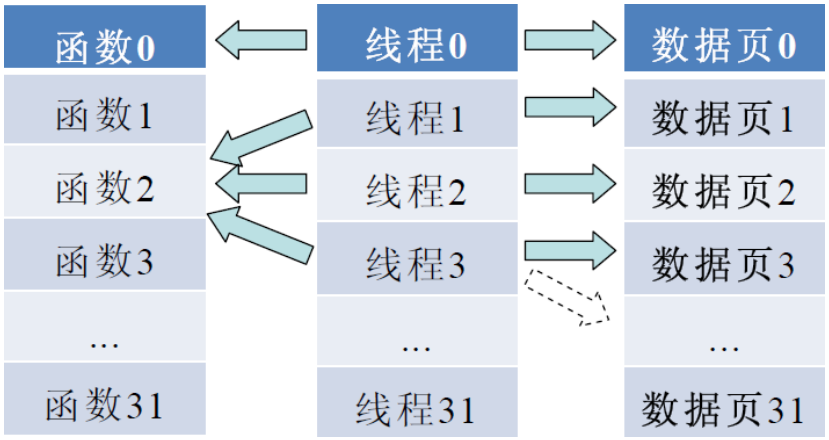


图 10-2　线程、函数和数据页的关系

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 10-2 所示。

10.3.6 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。

10.3.7 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。

10.3.8 例程

1. 单线程累加求和

例程 10-1　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;
    TCompileInfo compile;

    // 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);
// 编译运动程序sum.c,
// 编译成功以后生成sum.bin和sum.ini
// 必须保证error.ini文件位于工程文件夹中
rtn = GTN_Compile("sum.c", &compile);
printf("GTN_Compile ()=%d\n", rtn);
// 下载运动程序sum.bin,
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);
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 10-2　运动程序多线程累加求和

运动程序代码和例程 10-1 相同。

应用程序负责编译、下载、初始化、启动运动程序。和例程 10-1 不同之处在于启动 2 个线程完成累加运算任务。

10.4 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;
    TCompileInfo compile;

    // 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);
    // 编译运动程序sum.c,
    // 编译成功以后生成sum.bin和sum.ini
    // 必须保证error.ini文件位于工程文件夹中
    rtn = GTN_Compile("sum.c", &compile);
    printf("GTN_Compile ()=%d\n", rtn);
    // 下载运动程序sum.bin
    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;
}
```

10.4.1 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

10.4.2 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) return 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。

10.4.3 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;
```

标准 C 实现：

```
int i,sum;
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:
……
```

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

        cc = i<= 10;
        if(cc) 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;
}
```

(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;
```

标准 C 实现：

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

```
goto lbl_end;
lbl_Jump:
      a = b - a;
lbl_end:
 ……
```

```
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;
      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:
……
```

标准 C 实现：

```
int a;

a = 5;
switch(a)
{
case 1:
      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。

10.5　可在运动程序中使用的指令

表 10-2　可在运动程序中使用的指令

编号	指令	指令	指令
1	GT_Delay	GT_PrfTrap	GT_GetExtAdValue
4	GT_DelayHighPrecision	GT_SetTrapPrm	GT_GetExtAdVoltage
7	GT_SetDataPage	GT_GetTrapPrm	GT_SetExtDaValue

编号	指令	指令	指令
10	GT_SetDo	GT_PrfJog	GT_SetExtDaVoltage
13	GT_SetDoBit	GT_SetJogPrm	GT_GetStsExtMdl
16	GT_GetDo	GT_GetJogPrm	GT_AlarmOff
19	GT_GetDi	GT_PrfGear	GT_AlarmOn
22	GT_GetDiReverseCount	GT_SetGearMaster	GT_LmtsOn
25	GT_SetDiReverseCount	GT_GetGearMaster	GT_LmtsOff
28	GT_SetDac	GT_SetGearRatio	GT_ProfileScale
31	GT_GetDac	GT_GetGearRatio	GT_EncScale
34	GT_GetDacValue	GT_GearStart	GT_StepDir
37	GT_GetEncPos	GT_ZeroPos	GT_StepPulse
40	GT_SetTrigger	GT_SetExtIoValue	GT_SetMtrBias
43	GT_GetTriggerStatus	GT_GetExtIoValue	GT_GetMtrBias
46	GT_LinkCaptureOffset	GT_SetExtIoBit	GT_SetMtrLmt
49	GT_GetCaptureOffset	GT_GetExtIoBit	GT_GetMtrLmt
52	GT_GetClock	GT_SetPid	GT_EncSns
55	GT_GetSts	GT_GetPid	GT_SetPosErr
58	GT_ClrSts	GT_SetPos	GT_GetPosErr
61	GT_AxisOn	GT_GetPos	GT_SetStopDec
64	GT_AxisOff	GT_SetVel	GT_GetStopDec
67	GT_Update	GT_GetVel	GT_LmtSns
70	GT_Stop	GT_ZeroPos	GT_CtrlMode
73	GT_GetSoftLimit	GT_SetExtIoValue	GT_CrdSpace
76	GT_GetAxisBand	GT_GetExtIoValue	GT_CrdStart
79	GT_GetPrfVel	GT_SetExtIoBit	GT_SetOverride
82	GT_GetPrfMode	GT_GetExtIoBit	GT_GetCrdVel
85	GT_GetAxisPrfVel	GT_SetControlFilter	GT_CrdStatus
88	GT_GetAxisEncPos	GT_GetControlFilter	GT_SynchAxisPos
91	GT_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适应 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11章　其它指令

11.1　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。

注意	本章表格中的“页码”即为此指令在“第 12 章 指令详细说明”中的对应页码。可以使用“超级链接”进行索引。 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　打开/关闭运动控制器

表 11-1　打开/关闭运动控制器指令列表

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

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。

11.3　读取固件版本号

表 11-2　读取固件版本号指令列表

指令	说明
GTN_GetVersion	读取运动控制器固件的版本号
GTN_GetVersionEx	读取动态库的版本号

为了方便用户核对运动控制器固件版本，提供 GTN_GetVersion 指令来读取运动控制器固件版本号，版本号是一个含有 21 个字符的字符串：aaa bbbbbb ccc dddddd eee。具体的定义如表 11-3 所示。

表 11-3　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
eee	固件 3 的版本号

例程 11-1　读取运动控制器版本号

```
short rtn;
char *pVersion;                          // 定义指向版本号字符串的指针

rtn =    GTN_Open();

rtn =    GTN_GetVersion(1,&pVersion);          // 读取版本号

printf("%s\n", pVersion);

rtn =    GTN_Close()
```

11.4　读取系统时钟

表 11-4　读取系统时钟指令列表

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

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 125 微秒增加 1，调用　GTN_Reset 指令将会使计数时钟值清零。

11.5　打开/关闭电机使能信号

表 11-5　打开/关闭电机使能信号指令列表

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

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。

11.6　维护位置值

表 11-6　维护位置值指令列表

指令	说明
GTN_SetPrfPos	修改指定轴的规划位置
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
GTN_ZeroPos	清零规划位置和实际位置，并进行零漂补偿

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。

11.7　电机到位检测

表 11-7　电机到位检测指令列表

指令	说明
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
GTN_GetAxisBa	读取轴到位误差带

指令	说明
nd	

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 11-1 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。

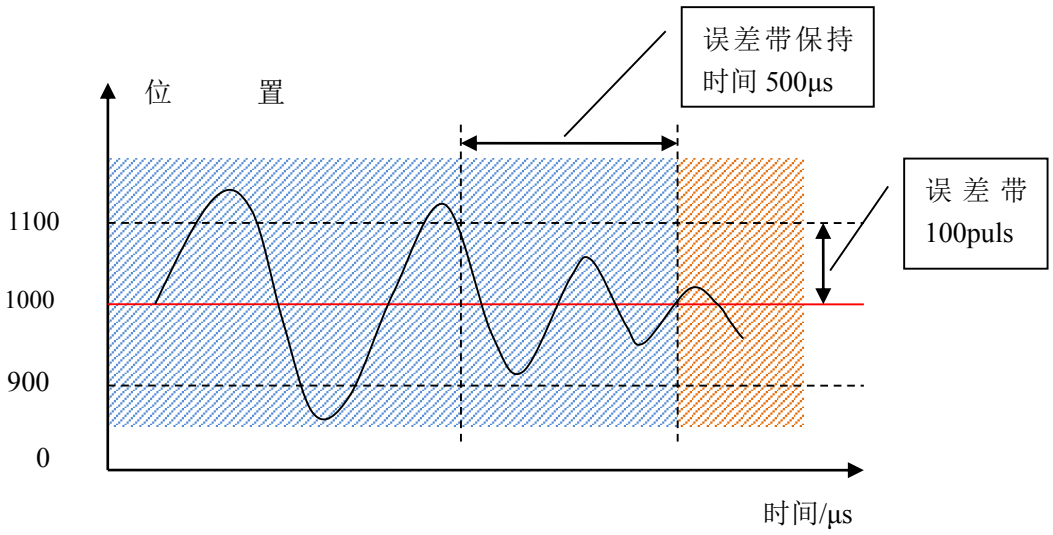


图 11-1　电机到位检测功能

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 11-2　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。

```
#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_ClrSts(1,1, 8);
    commandhandler("   GTN_ClrSts", 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;

}
```

11.8　反向间隙补偿

表 11-8　反向间隙补偿指令列表

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

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。

11.9　螺距误差补偿

表 11-9　螺距误差补偿指令列表

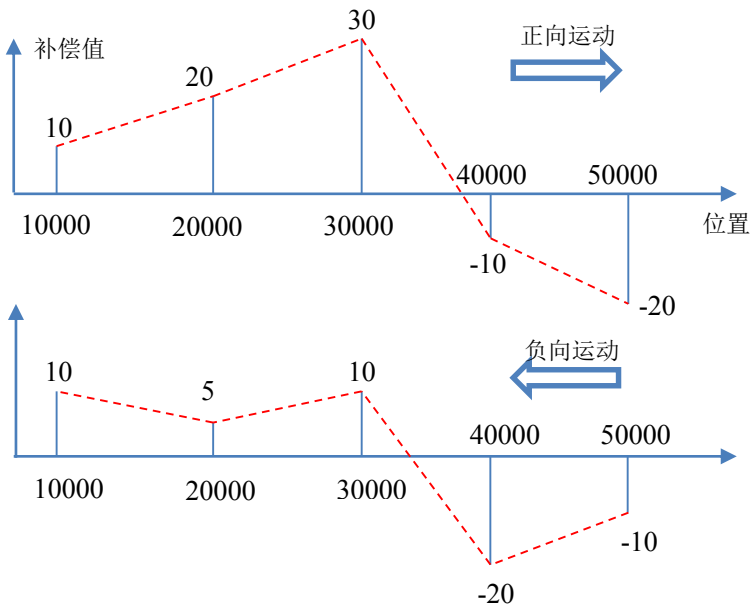
指令	说明
GTN_SetLeadScrewComp	加载补偿表
GTN_EnableLeadScrewComp	是否开启补偿

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）。



例程 11-3　螺距误差补偿

```
#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,compPos,compNeg);

//启动螺距误差补偿

sRtn =    GTN_EnableLeadScrewComp(1,1,1);
```

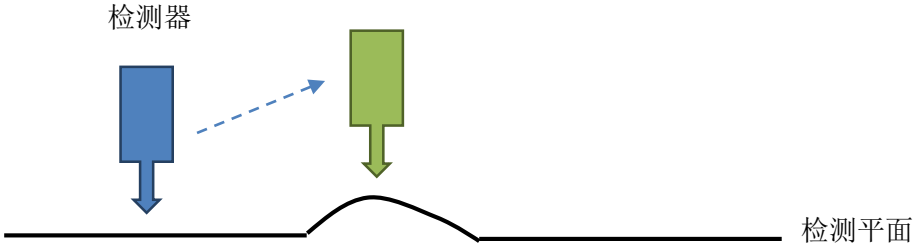
11.10 二维位置补偿

表 11-10　反螺距误差补偿指令列表

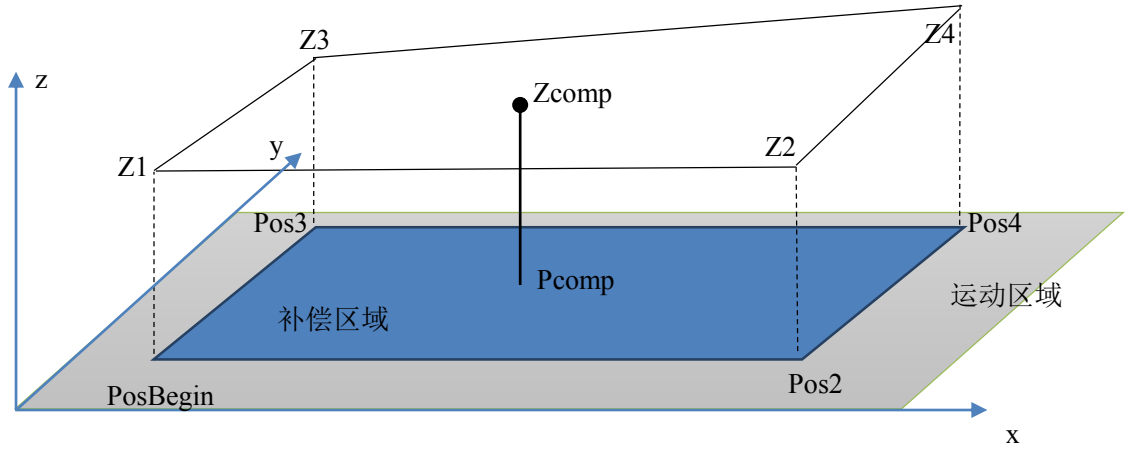
指令	说明
GTN_SetCompensate2DTable	设置二维补偿表及数据
GTN_GetCompensate2DTable	获取二维补偿表参数
GTN_SetCompensate2D	设置二维补偿参数
GTN_GetCompensate2D	获取二维补偿参数
GTN_GetCompensate2DValue	获取补偿值

11.10.1　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（如下图所示）。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。



下图所示，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。



11.10.2　例程

例程 11-4　二维位置补偿

```
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;
    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);   // 获取二维补偿表

    table.count[0] = 3;
    table.count[1] = 2;
    table.posBegin[0] = 0;          // 补偿起点从0开始
    table.posBegin[1] = 0;
    table.step[0] = 50000;          // 补偿步长为50000
    table.step[1] = 50000;

    sRtn =    GTN_SetCompensate2DTable(1,1,&table,&data[0][0]); // 写入补偿数据（不扩
展）
    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1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);
```

```
        double comp2DValue;
        sRtn =     GTN_GetCompensate2DValue(1,3,&comp2DValue);
        printf("comp2DValue=f%\n",comp2DValue );
    } while(run==1); //  等待插补运动停止
    return 0;
}
```

第12章　指令详细说明

注意

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。

以下表格中的所有指令的 core、axis、profile 和 encoder 参数的取值范围如下所示：

参数	取值范围（对应不同型号的 GEN 控制器）			
	008	016	032	064
core	1	1	1	[1, 2]
crd	[1, 4]	[1, 4]	[1, 4]	[1, 4]
axis	[1, 8]	[1, 16]	[1, 32]	[1, 32]
profile	[1, 8]	[1, 16]	[1, 32]	[1, 32]
encoder	[1, 8]	[1, 16]	[1, 32]	[1, 32]
pulse	[1, 8]	[1, 16]	[1, 32]	[1, 32]

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	24
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。		
axis	控制轴号，正整数。		
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	错误!未找到引用源。		

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	24
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。		
axis	控制轴号，正整数。		
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	错误!未找到引用源。		

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	53
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。		
crd	坐标系号，正整数		
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, 32767)，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。正整数，取值范围：[0, 1]，默认值为：0。		
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	53
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。		
crd	坐标系号，正整数		
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：[-4194304, 4194303]，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, 32767)，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。正整数，取值范围：[0, 1]，默认值为：0。
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	53
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。		
crd	坐标系号，正整数		
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, 32767)，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。正整数，取值范围：[0, 1]，默认值为：0。		
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	53
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。		
crd	坐标系号，正整数		
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：[-4194304,4194303]，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, 32767)，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。正整数，取值范围：[0, 1]，默认值为：0。		
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	53
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。		
crd	坐标系号，正整数		
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, 32767)，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。正整数，取值范围：[0, 1]，默认值为：0。
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	53
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。		
crd	坐标系号，正整数		
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：[-4194304, 4194303]，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, 32767)，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。正整数，取值范围：[0, 1]，默认值为：0。		
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	102
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。		
axis	关闭伺服使能的轴的编号。正整数。		
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	102
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。		
axis	打开伺服使能的轴的编号。正整数。		
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。 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	89
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。		
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 10-1　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	53
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。		
crd	坐标系号，正整数		
chn	模拟量输出的通道号。取值范围：[1, 8]。		
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。正整数，取值范围：[0, 1]，默认值为：0。		
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	53
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。		
crd	坐标系号，正整数		
delayTime	延时时间。取值范围：[0, 16383]，单位：ms。		
fifo	插补缓存区号。正整数，取值范围：[0, 1]，默认值为：0。		
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_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	53
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。
crd	坐标系号，正整数
gearAxis	需要进行跟随运动的轴号，取值范围：[1, 8]。该轴不能处于坐标系中。
pos	跟随运动的位移量，单位：pulse。
fifo	插补缓存区号。正整数，取值范围：[0, 1]，默认值为：0。
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 7-10　刀向跟随功能**

指令 15　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	53
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。		
crd	坐标系号，正整数		
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。正整数，取值范围：[0, 1]，默认值为：0。		
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 16　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	53
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。
crd	坐标系号，正整数
axis	需要将限位无效的轴的编号。
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。正整数，取值范围：[0, 1]，默认值为：0。
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 17　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	53
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。		
crd	坐标系号，正整数		
axis	需要将限位有效的轴的编号		
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。正整数，取值范围：[0, 1]，默认值为：0。		
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 18　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	53
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。		
crd	坐标系号，正整数		
moveAxis	需要进行点位运动的轴号，该轴不能处于坐标系中。		
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。正整数，取值范围：[0, 1]，默认值为：0。		
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 19　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	53
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。		
crd	坐标系号，正整数		
axis	需要设置停止 IO 信息的轴的编号。		
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 、MC_LIMIT_NEGATIVE、MC_ALARM、MC_HOME、MC_ARRIVE 时：取值范围同 axis 相同		

fifo	当 inputType= MC_GPI 时，取值范围：[1, 32]。 插补缓存区号。正整数，取值范围：[0, 1]，默认值为：0。
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 20　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	101
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	**例程 11-1　读取运动控制器版本号**		

指令 21　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	36
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。		
axis	起始轴号。正整数。		
count	读取的轴数，默认为 1。正整数。		
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 4-1　初始化配置控制器**		

指令 22　GTN_Compile

指令原型	short GTN_Compile(char *pFileName, TCompileInfo *pWrongInfo)		
指令说明	编译运动程序		
指令类型	立即指令，调用后立即生效。	章节页码	89
指令参数	该指令共有 2 个参数，参数的详细信息如下。		
pFileName	待编译的运动程序文件名，*.c文件		

pWrongInfo	编译信息，可通过此信息判断编译结果 typedef struct TCompileInfo { 　　char *pFileName;　//文件名 　　short lineNo　　//错误所在行号 　　char *pMessage;　//错误信息 }TCompileInfo;
指令返回值	若返回值为 2007： (1) 请检查文件名是否太长。 (2) 请检查需要编译的文件是否存在。 若返回值为 2008：表示编译失败。 (1) 请检查 error.ini 文件是否存在或已损坏。 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 10-1　运动程序单线程累加求和**

指令 23　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	53
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。		
crd	坐标系号，正整数		
fifo	所要清除的插补缓存区号。正整数，取值范围：[0, 1]，默认值为：0。		
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 24　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	53
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。		
crd	坐标系号，正整数		
pCrdData	只能设置为：NULL。		
fifo	插补缓存区号。正整数，取值范围：[0, 1]。默认值为：0。		
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 7-8　前瞻预处理例程**

指令 25　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	53
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。		
crd	坐标系号，正整数		
fifo	插补缓存区号。正整数，取值范围：[0, 1]。默认值为：0。		
link	数据缓存区序号，正整数，取值范围：[1, 4]		
threshold	数据缓存区大小，正整数 如果插补数据达到该数值，系统会自动将缓存区指令下发至DSP		
lookAheadInMc	前瞻下移功能标志位，默认为未下移		
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	GTN_CrdHsOff		
指令示例	**例程 7-12　开启 DMA 快速通道**		

指令 26　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	53
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。		
crd	坐标系号，正整数		
fifo	插补缓存区号，正整数，取值范围[0, 1]。		
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	GTN_CrdHsOn		
指令示例	**例程 7-12　开启 DMA 快速通道**		

指令 27　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	53
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。		
crd	坐标系号，正整数		
pSpace	读取插补缓存区中的剩余空间。		
fifo	插补缓存区号。正整数，取值范围：[0, 1]，默认值为：0。		
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 7-8　前瞻预处理例程**		

指令 28　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	53
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。		
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 29　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	53
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。		
crd	坐标系号，正整数		
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。正整数，取值范围：[0, 1]，默认值为：0。
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 30　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	89
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。		
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 10-1　运动程序单线程累加求和**		

指令 31　GTN_EcatIOReadInput

指令原型	short GTN_EcatIOReadInput(short core, unsigned short slave, unsigned short offset, unsigned short nSize, unsigned char *pValue)		
指令说明	读取EtherCAT IO模块的输入值。		
指令类型	立即指令，调用后立即生效。	章节页码	27
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。		
slave	EtherCAT 从站号，从 0 开始计数，包括伺服模块的排序。		
offset	输入的偏移地址，指从站内的 IO 地址偏移，越界将返回 7 参数错误。		
nSize	读取的字节数，不能超越最大的数据长度，越界将返回 7 参数错误。		
pValue	读取的数据指针。		
指令返回值	若返回值为 1：请检查相应从站是否已正确连接。 其他返回值：请参照指令返回值列表。		
相关指令	GTN_EcatIOWriteOutput		
指令示例	**例程 5-4　EtherCAT IO 的**		

指令 32　GTN_EcatIOWriteOutput

指令原型	short GTN_EcatIOWriteOutput(short core, unsigned short slave, unsigned short offset,

	unsigned short nSize, unsigned char *pValue)		
指令说明	写入EtherCAT IO模块的输出值。		
指令类型	立即指令，调用后立即生效。	章节页码	27
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。		
slave	EtherCAT 从站号，从 0 开始计数，包括伺服模块的排序。		
offset	输出的偏移地址，指从站内的 IO 地址偏移，越界将返回 7 参数错误。		
nSize	写入的字节数，不能超越最大的数据长度，越界将返回 7 参数错误。		
pValue	写入的数据指针。		
指令返回值	若返回值为 1：请检查相应从站是否已正确连接。 其他返回值：请参照指令返回值列表。		
相关指令	GTN_EcatIOReadInput		
指令示例	**例程 5-4　EtherCAT IO 的**		

指令 33　GTN_EcatSDODownload

指令原型	short GTN_EcatSDODownload(short core, unsigned short slave_position, unsigned short index, unsigned char subindex, unsigned char *pData, unsigned long data_size, unsigned long *pAbort_code)		
指令说明	通用 SDO 下载（Service Data Object，参考 IEC 61800）。		
指令类型	立即指令，调用后立即生效。	章节页码	27
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。		
slave_position	EtherCAT 从站号，从 0 开始计数		
index	SDO 的 Index (参考 IEC 61800)。		
subindex	SDO 的 Subindex (参考 IEC 61800)。		
pData	下载的数据指针。		
data_size	下载的数据量，按 Byte 计算。		
pAbort_code	异常退出代码。		
指令返回值	请参照指令返回值列表。		
相关指令	GTN_EcatSDOUpLoad		
指令示例	无		

指令 34　GTN_EcatSDOUpLoad

指令原型	short GTN_EcatSDOUpload(short core, unsigned short slave_position, unsigned short index, unsigned char subindex, unsigned char *pTarget, unsigned long target_size, unsigned long *pResult_size, unsigned long *pAbort_code)		
指令说明	通用 SDO 上传。		
指令类型	立即指令，调用后立即生效。	章节页码	27
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。		
slave_position	EtherCAT 从站号，从 0 开始计数		
index	SDO 的 Index (参考 IEC 61800)。		

subindex	SDO 的 Subindex (参考 IEC 61800)。
pTarget	上传的数据指针。
target_size	目标上传的数据量，按 Byte 计算。
pResult_size	实际上传的数据量。
pAbort_code	异常退出代码。
指令返回值	请参照指令返回值列表。
相关指令	GTN_EcatSDODownload
指令示例	无

指令 35　GTN_EnableLeadScrewComp

指令原型	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	107
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。		
axis	补偿轴轴号，正整数。		
mode	0-关闭；1-开启		
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 11-3　螺距误差补偿**		

指令 36　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	24
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。		
axis	控制轴号。正整数。		
alpha	编码器当量的alpha值，取值范围：[-32767, 0)和(0, 32767]。		
beta	编码器当量的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 37　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	49
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。
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。 当 core=1 时，对应关系如下： Bit | 31 | 30 | 29 | … | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 对应轴 | 32 轴 | 31 轴 | 30 轴 | … 轴 | 8 轴 | 7 轴 | 6 轴 | 5 轴 | 4 轴 | 3 轴 | 2 轴 | 1 轴 当 core=2 时，对应关系如下： Bit | 31 | 30 | 29 | … | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 对应轴 | 64 轴 | 63 轴 | 62 轴 | … 轴 | 40 轴 | 39 轴 | 38 轴 | 37 轴 | 36 轴 | 35 轴 | 34 轴 | 33 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**

指令 38　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**	103
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。		
axis	轴号。正整数。		
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 39　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**	36
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。		
axis	起始轴号。正整数。		
pValue	轴的编码器加速度。单位：pulse/ms^2。		

count	读取的轴数，默认为 1。 1 次最多可以读取 8 个编码器。
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 40　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	36
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。		
axis	起始轴号。正整数。		
pValue	轴的编码器位置。单位：pulse。		
count	读取的轴数，默认为 1。正整数。		
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 41　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	36
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。		
axis	起始轴号。正整数。		
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。 一次最多可以读取 8 个编码器。		
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 42　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	36

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。
axis	起始轴号。正整数。
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。 1 次最多可以读取 8 个编码器。
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 43　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	36
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。		
axis	起始轴号。正整数。		
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。 1 次最多可以读取 8 路。		
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 44　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	36
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。		
axis	起始轴号。正整数。		
pValue	轴的规划位置。单位：pulse。		
count	读取的轴数，默认为 1。正整数。 1 次最多可以读取 8 路。		
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_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	36
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。		
axis	起始轴号。正整数。		
pValue	轴的规划速度。单位：pulse/ms。		
count	读取的轴数，默认为 1。正整数。 1 次最多可以读取 8 路。		
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_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	103
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。		
axis	查询的轴号。正整数。		
pCompValue	读取的反向间隙补偿值。		
pCompChangeValue	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 47　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	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。		
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 48　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	102
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。		
pClock	读取的运动控制器的时钟，单位：250us。		
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_GetCompensate2D

指令原型	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	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。		
axis	需要补偿的轴号		
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	GTN_SetCompensate2D		
指令示例	**例程 11-4　二维位置补偿**		

指令 50　GTN_GetCompensate2DTable

指令原型	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	108
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。		
tableIndex	控制器内部的补偿表索引号，取值范围：[1]。		
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	GTN_SetCompensate2DTable		
指令示例	**例程 11-4　二维位置补偿**		

指令 51　GTN_GetCompensate2DValue

指令原型	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	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。		
axis	需要补偿的轴号		
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 11-4　二维位置补偿**		

指令 52　GTN_GetCrdHsPrm

指令原型	short GTN_GetCrdHsPrm(short core,short crd,short fifo,short *pEnable,short *pLink,unsigned short *pThreshold,short *pLookAheadInMc)		
适用板卡	GTN 系列产品		
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	53
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。		
crd	坐标系号。正整数		
fifo	插补缓存区号。正整数，取值范围：[0, 1]，默认值为：0。		
pEnable	使能标志位		
pLink	数据缓存区序号，正整数，取之范围[1,4]		
pThreshold	数据缓存区大小，正整数		
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		

指令 53　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	53
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。		
crd	坐标系号。正整数		
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 7-7　插补 FIFO 管理**		

指令 54　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	53
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。		
crd	坐标系号。正整数		
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 55　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	53
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。		
crd	坐标系号。正整数		
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 56　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	53
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。		
crd	坐标系号。正整数		
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 57　GTN_GetEcatAxisAI

指令原型	short GTN_GetEcatAxisAI(short core, short axis, short channel, short *pValue)
指令说明	读取EtherCAT轴模拟量输入。

指令类型	立即指令，调用后立即生效。	章节页码	27
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。		
axis	轴号		
channel	模拟量通道号，取值范围：[1, 2]。		
pValue	模拟量输入数值。		
指令返回值	若返回值为 1：请检查相应轴在 Gecat 配置文件中是否已经将相关对象（以 GTHD 为例，20F2h、20F9h）配置为 PDO。 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 58　GTN_GetEcatAxisAtlCurrent

指令原型	short GTN_GetEcatAxisAtlCurrent(short core, short axis, short * pCur)		
指令说明	读取 EtherCAT 轴的电流值。		
指令类型	立即指令，调用后立即生效。	章节页码	27
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。		
axis	轴号		
pCur	伺服电机电流值。		
指令返回值	若返回值为 1：请检查相应轴在 Gecat 配置文件中是否已经将相关对象（以 GTHD 为例，6078h）配置为 PDO。 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 59　GTN_GetEcatAxisAtlTorque

指令原型	short GTN_GetEcatAxisAtlTorque(short core, short axis, short * pTorque)		
指令说明	读取 EtherCAT 轴的力矩值。		
指令类型	立即指令，调用后立即生效。	章节页码	27
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。		
axis	轴号		
pTorque	伺服电机力矩值。		
指令返回值	若返回值为 1：请检查相应轴在 Gecat 配置文件中是否已经将相关对象（以 GTHD 为例，6077h）配置为 PDO。 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 60　GTN_GetEcatAxisDI

指令原型	short GTN_GetEcatAxisDI(short core, short axis, unsigned long *pDi)		
指令说明	读取EtherCAT轴的数字量输入值。		
指令类型	立即指令，调用后立即生效。	章节页码	27

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。
axis	轴号
pDi	数字 IO 输入状态，按位指示 IO 输入电平。
指令返回值	若返回值为 1：请检查相应轴在 Gecat 配置文件中是否已经将相关对象（以 GTHD 为例，60FDh）配置为 PDO。 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 61　GTN_GetEcatAxisDO

指令原型	short GTN_GetEcatAxisDO(short core, short axis, unsigned long *pDo)		
指令说明	读取EtherCAT轴的数字量输出值。		
指令类型	立即指令，调用后立即生效。	章节页码	27
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。		
axis	轴号		
pDo	数字 IO 输出状态，按位指示 IO 输出电平。		
指令返回值	若返回值为 1：请检查相应轴在 Gecat 配置文件中是否已经将相关对象（以 GTHD 为例，60FEh）配置为 PDO。 其他返回值：请参照指令返回值列表。		
相关指令	GTN_SetEcatAxisDO		
指令示例	无		

指令 62　GTN_GetEcatAxisMode

指令原型	short GTN_GetEcatAxisMode(short core, short axis, short *pMode)		
指令说明	读取 EtheCAT 轴的操作模式		
指令类型	立即指令，调用后立即生效。	章节页码	27
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。		
axis	轴号		
pMode	伺服操作模式（以 GTHD 为例）： 1：Profile position 3：Profile velocity 4：Profile torque 6：Homing 8：Cyclic sync position 9：Cyclic sync velocity 10：Cyclic sync torque		
指令返回值	若返回值为 1：请检查相应轴在 Gecat 配置文件中是否已经将相关对象（以 GTHD 为例，6061h）配置为 PDO。 其他返回值：请参照指令返回值列表。		

相关指令	GTN_SetEcatAxisMode
指令示例	无

指令 63　GTN_GetEcatAxisPE

指令原型	short GTN_GetEcatAxisPE(short core, short axis, long *pPosErr)		
指令说明	读取EtherCAT轴的位置误差。		
指令类型	立即指令，调用后立即生效。	章节页码	27
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。		
axis	轴号		
pPosErr	规划和编码器的位置差值。		
指令返回值	若返回值为 1：请检查相应轴在 Gecat 配置文件中是否已经将相关对象（以 GTHD 为例，60F4h）配置为 PDO。 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 64　GTN_GetEcatEncPos

指令原型	short GTN_GetEcatEncPos(short core, short axis, long *pEncPos)		
指令说明	读取 EtherCAT 轴的编码器位置。		
指令类型	立即指令，调用后立即生效。	章节页码	27
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。		
axis	轴号		
pEncPos	轴编码器实际位置。		
指令返回值	若返回值为 1：请检查相应轴在 Gecat 配置文件中是否已经将相关对象（以 GTHD 为例，6064h）配置为 PDO。 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 65　GTN_GetEcatEncVel

指令原型	short GTN_GetEcatEncVel(short core, short axis, long *pEncVel)		
指令说明	读取 EtherCAT 轴的编码器速度。		
指令类型	立即指令，调用后立即生效。	章节页码	27
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。		
axis	轴号		
pEncVel	编码器实际速度。		
指令返回值	若返回值为 1：请检查相应轴在 Gecat 配置文件中是否已经将相关对象（以 GTHD 为例，606Ch）配置为 PDO。 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 66　GTN_GetEcatHomingStatus

指令原型	short GTN_GetEcatHomingStatus(short core, short axis, unsigned short *pHomingStatus)		
指令说明	查询EtherCAT轴的回零状态。		
指令类型	立即指令，调用后立即生效。	章节页码	27
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。		
axis	轴号		
pHomeStatus	回零过程的状态值： BIT：状态 0：0：正在回零 1：回零完成 1：0：无意义 1：回零成功完成 2：0：无意义 1：回零过程出错		
指令返回值	若返回值为 1：请检查相应轴在 Gecat 配置文件中是否已经将相关对象（以 GTHD 为例，6041h）配置为 PDO。 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　采用 3 号回零方式**		

指令 67　GTN_GetEcatRawData

指令原型	short GTN_GetEcatRawData(short core, unsigned short offset, unsigned short size, unsigned char *pValue)		
指令说明	获取控制器 PDO 数据。		
指令类型	立即指令，调用后立即生效。	章节页码	27
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。		
offset	读取 PDO 数据偏移值，取值范围：0 ～ 当前 PDO 长度，单位：字节。		
size	读取数据长度，取值范围：[1, 56]，单位：字节。		
pValue	读取数据缓冲区地址		
指令返回值	若返回值为 1：请检查 EtherCAT 通讯是否正常。 其它返回值：请参照指令返回值列表。		
相关指令	GTN_SetEcatRawData		
指令示例	无		

指令 68　GTN_GetEcatSlaves

指令原型	short GTN_GetEcatSlaves(short core, short *pSlaveMotionCnt, short *pSlaveIOCnt)		
指令说明	读取EtherCAT总线的在线从站数目。		
指令类型	立即指令，调用后立即生效。	章节页码	27
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。		

pSlaveMotionCnt	在线的 EtherCAT 伺服从站数目。
pSlaveIOCnt	在线的 EtherCAT IO 从站数目。
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 69　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	81
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。		
encoder	编码器起始轴号。正整数		
pValue	编码器位置。		
count	读取的轴数。默认为 1。 1 次最多可以读取 8 个编码器轴。		
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 8-1　读取轴编码器位置值**		

指令 70　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	81
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。		
encoder	编码器起始轴号。正整数		
pValue	编码器速度。		
count	读取的轴数。默认为 1。 1 次最多可以读取 8 个编码器轴。		
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 8-1　读取轴编码器位置值**		

指令 71　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	89
指令参数	该指令共有 3 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 10-1　运动程序单线程累加求和**

指令 72　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	49
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。		
profile	规划轴号。正整数。		
pMasterIndex	读取主轴索引。正整数。		
pMasterType	读取主轴类型。 GEAR_MASTER_PROFILE（该宏定义为）表示跟随规划轴(profile)的输出值。 GEAR_MASTER_ENCODER（该宏定义为）表示跟随编码器(encoder)的输出值。 GEAR_MASTER_AXIS（该宏定义为）表示跟随轴(axis)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 73　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	49
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。		
profile	规划轴号。正整数。		
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 74　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	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。		
profile	规划轴号。正整数。		
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 75　GTN_GetMcEcatAxisNum

指令原型	short GTN_GetMcEcatAxisNum(short core, short *pNum)		
指令说明	读取 EtherCAT 伺服轴数。		
指令类型	立即指令，调用后立即生效。	章节页码	27
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。		
pPrm	EtherCAT 伺服轴数。		
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 76　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	82
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。		
pulse	脉冲计数器对应轴起始序号，正整数。		
pValue	内部脉冲计数器位置		
count	脉冲计数器对应轴个数 1 次最多可以读取 8 路。		
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_SetPls		
指令示例	**例程 8-2　访问内部脉冲计数器**		

指令 77　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	82
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。		
pulse	脉冲计数器对应轴起始序号，正整数。		
pValue	内部脉冲计数器速度		
count	脉冲计数器对应轴个数 1 次最多可以读取 8 路。		
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 78　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	42
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。		
profile	规划轴号。正整数。		
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 79　GTN_GetPosScale

指令原型	short GTN_GetPosScale(short core, short axis, unsigned short *pScale)		
指令说明	读取编码器倍率。		
指令类型	立即指令，调用后立即生效。	章节页码	27
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。		
axis	轴号		
pScale	编码器倍率，2 的幂运算的指数值。		
指令返回值	请参照指令返回值列表。		
相关指令	GTN_SetPosScale		
指令示例	无		

指令 80　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	36
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。		
profile	起始规划轴号。正整数。		
pValue	规划加速度。单位：pulse/ms^2。		
count	读取的轴数，默认为 1，正整数。 1 次最多可以读取 8 路		
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 81　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	36
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。		
profile	起始规划轴号，正整数。		
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。 1 次最多可以读取 8 路。
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 82　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**	36
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。		
profile	起始规划轴号，正整数。		
pValue	规划位置。单位：pulse。		
count	读取的轴数，默认为 1，正整数。 1 次最多可以读取 8 路。		
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 83　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**	36
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。		
profile	起始规划轴号，正整数。		
pValue	规划速度。单位：pulse/ms。		
count	读取的轴数，默认为 1，正整数。 1 次最多可以读取 8 路。		
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 84　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	53
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		
crd	坐标系号。正整数		
pSegment	读取的剩余插补段的段数。		
fifo	插补缓存区号。正整数，取值范围：[0, 1]，默认值为：0。		
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 85　GTN_GetRetainValue

指令原型	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	87
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。		
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[0,16369] 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	GTN_SetRetainValue		
指令示例	**例程 9-2　掉电存储操作**		

指令 86　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	81
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。		
dataType	输入输出资源类型 MC_ENCODE: (该宏定义为 23)：编码器。		
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_SetSense
指令示例	无。

指令 87　GTN_GetSoftLimit

指令原型	short GTN_GetSoftLimit (short core, short axis, long *pPositive, long *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	84
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。		
axis	轴号，正整数。		
pPositive	读取正向软限位。		
pNegative	读取负向软限位。		
指令返回值	请参照指令返回值列表。		
相关指令	GTN_SetSoftLimit		
指令示例	**例程 9-1　软限位使用**		

指令 88　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	84
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。		
axis	轴号，正整数		
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 9-1　软限位使用**		

指令 89　GTN_GetLimitStatus

指令原型	short GT_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	84
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。		
axis	轴号，正整数		
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 9-1　软限位使用**

指令 90　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	24
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。		
profile	规划器的编号，正整数。		
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 91　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	36
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。		
axis	起始轴号，正整数。		
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。 1 次最多可以读取 8 路。		
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 92　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	89
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。		
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 10-1　运动程序单线程累加求和**

指令 93　GTN_GetTouchProbeStatus

指令原型	short GTN_GetTouchProbeStatus(short core, short axis, unsigned short *probeStatus, long *probe1RiseValue, long *probe1FallValue, long *probe2RiseValue, long *probe2FallValue)
指令说明	查询EtherCAT轴的探针状态和捕获值。
指令类型	立即指令，调用后立即生效。　**章节页码**　27
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。
axis	轴号
probeStatus	探针状态： BIT: 描述 0: Probe1 开关: 0 OFF; 1 Enable 1: Probe1 上升沿: 0 未捕获; 1 捕获 2: Probe1 下降沿: 0 未捕获; 1 捕获 3-7: 保留 8: Probe2 开关: 0 OFF; 1 Enable 9: Probe2 上升沿: 0 未捕获; 1 捕获 10: Probe2 下降沿: 0 未捕获; 1 捕获 11-15: 保留
probe1RiseValue	探针 1 的上升沿捕获值。
probe1FallValue	探针 1 的下降沿捕获值。
probe2RiseValue	探针 2 的上升沿捕获值。
probe2FallValue	探针 2 的下降沿捕获值。
指令返回值	若返回值为 1：请检查相应轴在 Gecat 配置文件中是否已经将相关对象（以 GTHD 为例，60B9h、60BAh、60BBh）配置为 PDO。 其他返回值：请参照指令返回值列表。
相关指令	GTN_SetTouchProbeFunction、GTN_SetTouchProbeFunctionEx
指令示例	**例程 5-3　探针 1 上升沿连续捕获（以 GTHD 为例）**

指令 94　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	42
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。		
profile	规划轴号，正整数。		
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 95　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	53
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[1,1]		
crd	坐标系号。正整数		
pSegment	读取的用户自定义的插补段段号。		
fifo	插补缓存区号。正整数，取值范围：[0, 1]，默认值为：0。		
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 96　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	89
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 10-1　运动程序单线程累加求和**

指令 97　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	89
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。		
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 10-1　运动程序单线程累加求和**		

指令 98　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	42 / 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。		
profile	规划轴号，正整数。		
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 99　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	101

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。
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	**例程 11-1　读取运动控制器版本号**

指令 100　GTN_GetVersionEx

指令原型	short GTN_GetVersionEx(short core,short type,TVersion *pVersion);		
指令说明	读取动态库的版本号。		
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 3 个参数，参数的详细信息如下。		
core	内核，正整数。		
type	类型， type=0,读取gts.dll的版本信息 type=4,读取gt_rn.dll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	**例程 11-1　读取运动控制器版本号**		

指令 101　GTN_InitEcatComm

指令原型	short GTN_InitEcatComm(short core)		
指令说明	EtherCAT初始化。		
指令类型	立即指令，调用后立即生效。	章节页码	27
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。		
指令返回值	0 初始化成功；-8 配置错误，请检查 eni 配置文件和实际连接从站是否匹配；-9 未找到 eni 配置文件，请将 eni 文件放到可执行程序的相同目录；-10 未找到从站，请检查从站接线是否稳固		
相关指令	无		
指令示例	**例程 5-1　EtherCAT 初始化**		

指令 102　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	53
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。		
crd	坐标系号。正整数		
fifo	插补缓存区号。正整数，取值范围：[0, 1]，默认值为：0。		
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 7-8　前瞻预处理例程**		

指令 103　GTN_IsEcatReady

指令原型	short GTN_IsEcatReady(short core, short *pStatus)		
指令说明	查询EtherCAT通讯状态。		
指令类型	立即指令，调用后立即生效。	章节页码	27
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。		
pStatus	通讯状态： 0：通讯未完全建立；1：通讯完全建立。		
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 5-1　EtherCAT 初始化**		

指令 104　GTN_LmtsOff

指令原型	short GTN_LmtsOff(short core, short axis, short limitType=-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	24
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。		
axis	控制轴号，正整数。		
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、负限位和软限位都无效，默认为该值。		
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_LmtsOn		
指令示例	错误!未找到引用源。		

指令 105　GTN_LmtsOn

指令原型	short GTN_LmtsOn(short core, short axis, short limitType=-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	24
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。		
axis	控制轴号，正整数。		
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。		
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_LmtsOff		
指令示例	无。		

指令 106　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	53
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。		
crd	坐标系号。正整数		
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, 32767)，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。取值范围：[0, 1]，默认值为：0。		
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 107　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　53
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。
crd	坐标系号。正整数
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, 32767)，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。正整数，取值范围：[0, 1]，默认值为：0。
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 108　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　53
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。
crd	坐标系号。正整数
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, 32767)，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，取值范围：[0, 1]，默认值为：0。
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 109　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	53
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。		
crd	坐标系号。正整数		
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, 32767)，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。正整数，取值范围：[0, 1]，默认值为：0。		
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 110　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	53
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。		
crd	坐标系号。正整数		
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, 32767)，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。正整数，取值范围：[0, 1]，默认值为：0。		
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 111　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	53
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。		
crd	坐标系号。正整数		
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, 32767)，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。正整数，取值范围：[0, 1]，默认值为：0。		
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 112　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	24
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。		
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 113　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	101
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 #define CHANNEL_HOST (0) #define CHANNEL_UART (1) #define CHANNEL_SIM (2) #define CHANNEL_ETHER (3) #define CHANNEL_RS232 (4) #define CHANNEL_PCIE (5)		
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	**例程 11-1　读取运动控制器版本号**		

指令 114　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	89
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。		
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 115　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	49
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。		
profile	规划轴号，正整数。		
dir	设置跟随方式。		

指令返回值	0 表示双向跟随，1 表示正向跟随，-1 表示负向跟随。 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 116　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	46
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。		
profile	规划轴号，正整数。		
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 117　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	42
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。		
profile	规划轴号，正整数。		
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 118　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。		
指令类型	立即指令，调用后立即生效。	章节页码	24
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。		
axis	控制轴号，正整数。		
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 119　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	101
指令参数	无。		
指令返回值	请参照指令返回值列表。		
相关指令	无。		
指令示例	错误!未找到引用源。		

指令 120　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	89
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。		
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			
指令示例	**例程 10-1　运动程序单线程累加求和**		

指令 121　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	103
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。		
axis	轴号，正整数。		
band	误差带大小。单位：脉冲。		
time	误差带保持时间。单位：250 微秒。		
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 11-2　电机到位检测功能**

指令 122　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**	103
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。		
axis	需要进行反向间隙补偿的轴的编号，正整数。		
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 123　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**　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。
axis	需要补偿的轴号
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。 typedefstruct { short enable;　// 二维补偿使能标志 short tableIndex;　// 所使用的二维补偿表索引（取值为1） 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，

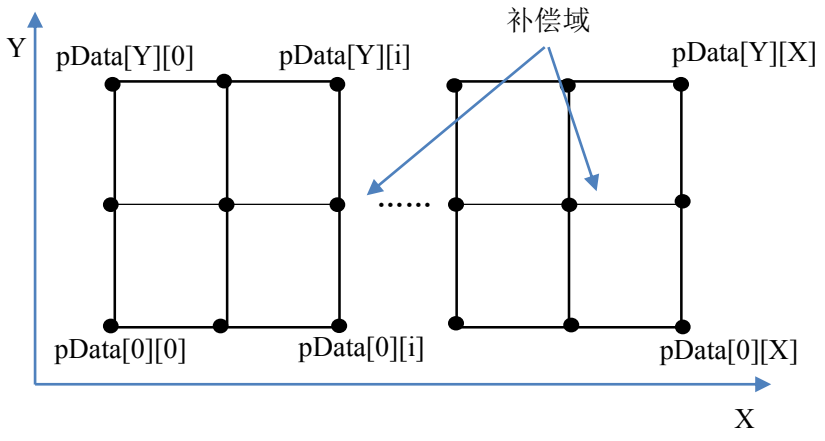
	MC_ENCODER：编码器位置
	short axisIndex[2]; //二维补偿X、Y方向运动所使用的轴号，axisType[0]：X方向的轴号，axisType[1]：Y方向的轴号
	} TCompensate2D;

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	GTN_GetCompensate2D
指令示例	**例程 11-4　二维位置补偿**

指令 124　GTN_SetCompensate2DTable

指令原型	short GTN_SetCompensate2DTable(short core, short tableIndex, TCompensate2DTable *pTable, long *pData, short externComp)		
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	108
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。		
tableIndex	控制器内部的补偿表索引号，取值范围：[1]。		

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：
	typedefstruct
	{
	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
	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 2000 即：X*Y<=2000。  		

externComp	是否自动扩展补偿区域。 0：不自动扩展补偿区域，按照设置的补偿区域进行补偿； 1：自动把设置的补偿区域向四周扩展 1 个步长。 如果不自动扩展补偿区域边界，则补偿区域的 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	GTN_GetCompensate2DTable
指令示例	**例程 11-4　二维位置补偿**

指令 125　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	53
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。		
crd	坐标系号，正整数。		
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, 4]。 **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, 32767)。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		

指令返回值	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。
	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 126　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	53
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。		
crd	坐标系号。正整数		
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 127　GTN_SetEcatAxisDO

指令原型	short GTN_SetEcatAxisDO(short core, short axis, unsigned long DoValue)		
指令说明	设置EtherCAT轴的数字量输出。		
指令类型	立即指令，调用后立即生效。	章节页码	27
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。		
axis	轴号		
DoValue	数字量输出值。		
指令返回值	若返回值为 1：请检查相应轴在 Gecat 配置文件中是否已经将相关对象（以 GTHD 为例，60FEh）配置为 PDO。 其他返回值：请参照指令返回值列表。		
相关指令	GTN_GetEcatAxisDO		
指令示例	无		

指令 128　GTN_SetEcatAxisMode

指令原型	short GTN_SetEcatAxisMode(short core, short axis, short mode)		
指令说明	设置 EtheCAT 轴的操作模式		
指令类型	立即指令，调用后立即生效。	章节页码	27
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。		
axis	轴号		
mode	伺服操作模式（以 GTHD 为例）： 1：Profile position 3：Profile velocity 4：Profile torque 6：Homing 8：Cyclic sync position 9：Cyclic sync velocity 10：Cyclic sync torque		
指令返回值	若返回值为 1：请检查相应轴在 Gecat 配置文件中是否已经将相关对象（以 GTHD 为例，6060h）配置为 PDO。 其他返回值：请参照指令返回值列表。		
相关指令	GTN_GetEcatAxisMode		
指令示例	无		

指令 129　GTN_SetEcatAxisPT

指令原型	short GTN_SetEcatAxisPT(short core, short axis, short torque)		
指令说明	设置 EtherCAT 轴的力矩。		
指令类型	立即指令，调用后立即生效。	章节页码	27
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。		
axis	轴号		
torque	力矩值		
指令返回值	若返回值为 1：请检查相应轴在 Gecat 配置文件中是否已经将相关对象（以 GTHD 为例，6071h）配置为 PDO。 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 130　GTN_SetEcatAxisPV

指令原型	short GTN_SetEcatAxisPV(short core, short axis, long velocity)		
指令说明	设置 EtherCAT 轴的速度		
指令类型	立即指令，调用后立即生效。	章节页码	27
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。		
axis	轴号		
velocity	速度值		
指令返回值	若返回值为 1：请检查相应轴在 Gecat 配置文件中是否已经将相关对象（以 GTHD		

	为例，60FFh）配置为 PDO。 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 131　GTN_SetEcatHomingPrm

指令原型	short GTN_SetEcatHomingPrm(short core, short axis, short method, double speed1, double speed2, double acc, long offset, unsigned short probeFunction)		
指令说明	设置EtherCAT轴的回零参数。		
指令类型	立即指令，调用后立即生效。	章节页码	27
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。		
axis	轴号		
method	回零方式（以 GTHD 为例）： 1：离开负限位后，第一个 index 标记回零 2：离开正限位后，第一个 index 标记回零 3：离开回零开关后，第一个 index 标记回零（正行程回零） 4：接触回零开关后，第一个 index 标记回零（正行程回零） 5：离开回零开关后，第一个 index 标记回零（负行程回零） 6：接触回零开关后，第一个 index 标记回零（负行程回零） 7：离开回零开关的负边沿后，第一个 index 标记回零（初始正方向运动） 8：接触回零开关的负边沿后，第一个 index 标记回零（初始正方向运动） 9：接触回零开关的正边沿后，第一个 index 标记回零（初始正方向运动） 10：离开回零开关的正边沿后，第一个 index 标记回零（初始正方向运动） 11：离开回零开关的正边沿后，第一个 index 标记回零（初始负方向运动） 12：接触回零开关的正边沿后，第一个 index 标记回零（初始负方向运动） 13：接触回零开关的负边沿后，第一个 index 标记回零（初始负方向运动） 14：离开回零开关的负边沿后，第一个 index 标记回零（初始负方向运动） 17：负限位下降沿回零 18：正限位下降沿回零 19：回零开关下降沿回零（正行程回零） 20：回零开关上升沿回零（正行程回零） 21：回零开关下降沿回零（负行程回零） 22：回零开关上升沿回零（负行程回零） 23：回零开关负边下降沿回零（初始正向运动） 24：回零开关负边上升沿回零（初始正向运动） 25：回零开关正边上升沿回零（初始正向运动） 26：回零开关正边下降沿回零（初始正向运动） 27：回零开关正边下降沿回零（初始负向运动） 28：回零开关正边上升沿回零（初始负向运动） 29：回零开关负边上升沿回零（初始负向运动） 30：回零开关负边下降沿回零（初始负向运动） 33：index 标记回零（负方向运动） 34：index 标记回零（正方向运动） 35：以当前位置为零位		

speed1	搜索开关速度，单位：驱动器设置的用户速度单位
speed2	搜索 index 标识速度，单位：驱动器设置的用户速度单位
acc	搜索加速度，单位：驱动器设置的用户加速度单位
offset	原点偏移量，单位：驱动器设置的用户位置单位
probeFunction	探针功能。
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　采用 3 号回零方式**

指令 132　GTN_SetEcatRawData

指令原型	short GTN_SetEcatRawData(short core, unsigned short offset, unsigned short size, unsigned char *pValue)		
指令说明	设置控制器 PDO 数据。		
指令类型	立即指令，调用后立即生效。	章节页码	27
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。		
offset	设置 PDO 数据偏移值，取值范围：0 ～ 当前 PDO 长度，单位：字节。		
size	设置数据长度，取值范围：[1, 56]，单位：字节。		
pValue	设置数据缓冲区地址。		
指令返回值	若返回值为 1：请检查 EtherCAT 通讯是否正常。 其它返回值：请参照指令返回值列表。		
相关指令	GTN_GetEcatRawData		
指令示例	无		

指令 133　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	81
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。		
encoder	编码器起始轴号，正整数。		
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 134　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	49
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。
profile	规划轴号，正整数。
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_PROFILE（该宏定义为 2）表示跟随规划轴(profile)的输出值。默认为该类型。 GEAR_MASTER_ENCODER（该宏定义为 1）表示跟随编码器(encoder)的输出值。 GEAR_MASTER_AXIS（该宏定义为 3）表示跟随轴(axis)的输出值。 GEAR_MASTER_AU_ENCODER(该宏定义为 4) 表示跟随辅助编码器 GEAR_MASTER_AXIS_OTHER(该宏定义为 103) 表示 core2 轴跟随 core1 中轴(axis)的输出值 GEAR_MASTER_ENCODER_OTHER(该宏定义为 101) 表示 core2 轴跟随 core1 中编码器(encoder)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。 1 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 135　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	49
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。		
profile	规划轴号，正整数。		
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 136　GTN_SetHomingMode

指令原型	short GTN_SetHomingMode(short core, short axis, short mode)		
指令说明	切换EtherCAT轴的回零模式。		
指令类型	立即指令，调用后立即生效。	章节页码	27
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。		
axis	轴号		
mode	模式选择。6：回零模式；8：周期同步位置模式（CSP）		
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 5-2　采用 3 号回零方式**		

指令 137　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　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。
profile	规划轴号，正整数。
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 138　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	107
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。		
axis	补偿轴轴号，正整数。		
n	补偿段数		
startPos	补偿开始位置		
lenPos	补偿总距离（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 11-3　螺距误差补偿**		

指令 139　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	53
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。		
crd	坐标系号。正整数		
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 140　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	82
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。		

encoder	脉冲计数器对应轴序号，正整数。
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 141　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	42
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。		
profile	规划轴号，正整数。		
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 142　GTN_SetPosScale

指令原型	short GTN_SetPosScale(short core, short axis, unsigned short scale)		
指令说明	设置编码器倍率，主要用于编码器分辨率较高的时候。		
指令类型	立即指令，调用后立即生效。	章节页码	27
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。		
axis	轴号		
scale	编码器倍率，2 的幂运算的指数值。 例如：编码器是 23 位，那么此值如果是 6，则运控换算变成 17 位。		
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_GetPosScale		
指令示例	无		

指令 143　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	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。
profile	规划轴编号，正整数。
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 144　GTN_SetRetainValue

指令原型	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	87
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。		
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[0,16369] 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	请参照指令返回值列表。		
相关指令	GTN_GetRetainValue		
指令示例	**例程 9-2　掉电存储操作**		

指令 145　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	81
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。		
dataType	输入输出资源类型 MC_ENCODE: (该宏定义为 23)：编码器。		
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 146　GTN_SetSoftLimit

指令原型	short GTN_SetSoftLimit (short core, short axis, long positive, long 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	84
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。		
axis	轴号，正整数。		
positive	正向软限位，当规划位置大于该值时，正限位触发。 默认值为：0x7fffffff，表示正向软限位无效。		
negative	负向软限位，当规划位置小于该值时，负限位触发。 默认值为：0x80000000，表示负向软限位无效。		
指令返回值	请参照指令返回值列表。		
相关指令	GTN_GetSoftLimit		
指令示例	**例程 9-1　软限位使用**		

指令 147　GTN_SetSoftLimitMode

指令原型	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	84
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。		
axis	轴号，正整数		
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 148　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	24
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。		
profile	规划器的编号，正整数。		
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 149　GTN_SetTouchProbeFunction

指令原型	short GTN_SetTouchProbeFunction(short core, short axis, short ProbePrm)

指令说明	设置EtherCAT轴的探针参数（参数方式）。		
指令类型	立即指令，调用后立即生效。	**章节页码**	27
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。		
axis	轴号		
ProbePrm	按位设置探针参数。		

BIT	描述
0	Probe1 开关　0:OFF　1:Enable
1	Probe1 触发类型 0：单次 1：连续
2-3	保留
4	上升沿触发 0：无效 1：有效
5	下降沿触发 0：无效 1：有效
6-7	保留
8	Probe2 开关　0:OFF　1:Enable
9	Probe2 触发类型 0：单次 1：连续
10-11	保留
12	上升沿触发 0：无效 1：有效
13	下降沿触发 0：无效 1：有效
14-15	保留

指令返回值	若返回值为 1：请检查相应轴在 Gecat 配置文件中是否已经将相关对象（以 GTHD 为例，60B8h）配置为 PDO。 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 5-3　探针 1 上升沿连续捕获（以 GTHD 为例）**		

指令 150　GTN_SetTouchProbeFunctionEx

指令原型	short GTN_SetTouchProbeFunctionEx(short core, short axis, short Probe1Enable, short Probe1TriggerType, short Probe1TriggerLevel, short Probe2Enable, short Probe2TriggerType, short Probe2TriggerLevel)		
指令说明	设置EtherCAT轴的探针参数（功能描述方式）。		
指令类型	立即指令，调用后立即生效。	**章节页码**	27
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。		
axis	轴号		
Probe1Enable	探针通道 1 使能状态 0：不使能 1：使能。		
Probe1TriggerType	探针 1 的触发类型： ECAT_PROBE_TRIGGER_TYPE_CONTINUES　连续触发； ECAT_PROBE_TRIGGER_TYPE_FIRST_EVENT 单次触发。		
Probe1TriggerLevel	探针 1 的触发电平： ECAT_PROBE_TRIGGER_LEVEL_POS　上升沿触发； ECAT_PROBE_TRIGGER_LEVEL_NEG　下降沿触发； ECAT_PROBE_TRIGGER_LEVEL_DUL　双沿触发。		
Probe2Enable	探针通道 2 使能状态 0：　不使能 1：使能。		
Probe2TriggerType	探针 2 的触发类型：		

Probe2TriggerLevel	ECAT_PROBE_TRIGGER_TYPE_CONTINUES　连续触发； ECAT_PROBE_TRIGGER_TYPE_FIRST_EVENT　单次触发。
	探针 2 的触发电平： ECAT_PROBE_TRIGGER_LEVEL_POS　上升沿触发； ECAT_PROBE_TRIGGER_LEVEL_NEG　下降沿触发； ECAT_PROBE_TRIGGER_LEVEL_DUL　双沿触发。
指令返回值	若返回值为 1：请检查相应轴在 Gecat 配置文件中是否已经将相关对象（以 GTHD 为例，60B8h）配置为 PDO。 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 151　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**	42
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。		
profile	规划轴号，正整数。		
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，取值范围：[0, 50]，单位 ms。平滑时间的数值越大，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 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 152　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	53
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		
crd	坐标系号。正整数		
segNum	设置用户自定义的插补段段号。		
fifo	插补缓存区号。正整数，取值范围：[0, 1]，默认值为：0。		
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 153　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	89
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。		
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 10-1　运动程序单线程累加求和**		

指令 154　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	42 / 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。		

profile	规划轴号，正整数。
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 155　GTN_StartEcatComm

指令原型	short GTN_StartEcatComm(short core)		
指令说明	启动EtherCAT通讯。成功调用这条指令后，才可以调用其他运动指令。		
指令类型	立即指令，调用后立即生效。	章节页码	27
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。		
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 5-1　EtherCAT 初始化**		

指令 156　GTN_StartEcatHoming

指令原型	short GTN_StartEcatHoming(short core, short axis)		
指令说明	启动EtherCAT轴回零。		
指令类型	立即指令，调用后立即生效。	章节页码	27
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。		
axis	轴号		
指令返回值	若返回值为 1：请检查相应轴是否满足以下条件： 1.　目标驱动器处于伺服使能状态； 2.　目标驱动器的操作模式是回零模式； 3.　已成功设置回零参数。 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　采用 3 号回零方式**		

指令 157　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。 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。		
指令类型	立即指令，调用后立即生效。	章节页码	36
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。		
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。		

<table>
<tr><td rowspan="4">option</td><td>当 core=1 时，对应关系如下：
<table>
<tr><td>Bit</td><td>31</td><td>30</td><td>29</td><td>…</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>对应轴</td><td>32 轴</td><td>31 轴</td><td>30 轴</td><td>… 轴</td><td>8 轴</td><td>7 轴</td><td>6 轴</td><td>5 轴</td><td>4 轴</td><td>3 轴</td><td>2 轴</td><td>1 轴</td></tr>
</table>
当 core=2 时，对应关系如下：
<table>
<tr><td>Bit</td><td>31</td><td>30</td><td>29</td><td>…</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>对应轴</td><td>64 轴</td><td>63 轴</td><td>62 轴</td><td>… 轴</td><td>40 轴</td><td>39 轴</td><td>38 轴</td><td>37 轴</td><td>36 轴</td><td>35 轴</td><td>34 轴</td><td>33 轴</td></tr>
</table></td></tr>
<tr><td>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。
当 core=1 时，对应关系如下：
<table>
<tr><td>Bit</td><td>31</td><td>30</td><td>29</td><td>…</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>对应轴</td><td>32 轴</td><td>31 轴</td><td>30 轴</td><td>… 轴</td><td>8 轴</td><td>7 轴</td><td>6 轴</td><td>5 轴</td><td>4 轴</td><td>3 轴</td><td>2 轴</td><td>1 轴</td></tr>
</table>
当 core=2 时，对应关系如下：
<table>
<tr><td>Bit</td><td>31</td><td>30</td><td>29</td><td>…</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>对应轴</td><td>64 轴</td><td>63 轴</td><td>62 轴</td><td>… 轴</td><td>40 轴</td><td>39 轴</td><td>38 轴</td><td>37 轴</td><td>36 轴</td><td>35 轴</td><td>34 轴</td><td>33 轴</td></tr>
</table></td></tr>
<tr><td>指令返回值</td><td>请参照指令返回值列表。</td></tr>
<tr><td>相关指令</td><td>无。</td></tr>
<tr><td>指令示例</td><td>无。</td></tr>
</table>

指令 158　GTN_StopEcatHoming

指令原型	short GTN_StopEcatHoming(short core, short axis)		
指令说明	停止EtherCAT轴回零。		
指令类型	立即指令，调用后立即生效。	章节页码	27
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。		
axis	轴号		
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 159　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	89
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。		
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 160　GTN_SynchAxisPos

指令原型	short GTN_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	102
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。		
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。 当 core=1 时，对应关系如下： Bit | 31 | 30 | 29 | … | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 对应轴 | 32 轴 | 31 轴 | 30 轴 | … 轴 | 8 轴 | 7 轴 | 6 轴 | 5 轴 | 4 轴 | 3 轴 | 2 轴 | 1 轴 当 core=2 时，对应关系如下： Bit | 31 | 30 | 29 | … | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 对应轴 | 64 轴 | 63 轴 | 62 轴 | … 轴 | 40 轴 | 39 轴 | 38 轴 | 37 轴 | 36 轴 | 35 轴 | 34 轴 | 33 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 161　GTN_TerminateEcatComm

指令原型	short GTN_TerminateEcatComm(short core)		
指令说明	结束EtherCAT通讯。		
指令类型	立即指令，调用后立即生效。	章节页码	27
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。		
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 162　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	42, 46
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。
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。 当 core=1 时，对应关系如下： Bit: 31 30 29 … 7 6 5 4 3 2 1 0 对应轴: 32轴 31轴 30轴 …轴 8轴 7轴 6轴 5轴 4轴 3轴 2轴 1轴 当 core=2 时，对应关系如下： Bit: 31 30 29 … 7 6 5 4 3 2 1 0 对应轴: 64轴 63轴 62轴 …轴 40轴 39轴 38轴 37轴 36轴 35轴 34轴 33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 163　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**	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。		
axis	需要位置清零的起始轴号，正整数		
count	需要位置清零的轴数。默认值为 1，取值范围同 axis。		
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**		

第13章　索引

13.1　指令索引

指令 1　GTN_AlarmOff....111
指令 2　GTN_AlarmOn....111
指令 3　GTN_ArcXYC....111
指令 4　GTN_ArcXYR....112
指令 5　GTN_ArcYZC....113
指令 6　GTN_ArcYZR....113
指令 7　GTN_ArcZXC....114
指令 8　GTN_ArcZXR....115
指令 9　GTN_AxisOff....116
指令 10　GTN_AxisOn....116
指令 11　GTN_Bind....116
指令 12　GTN_BufDA....116
指令 13　GTN_BufDelay....117
指令 14　GTN_BufGear....117
指令 15　GTN_BufIO....118
指令 16　GTN_BufLmtsOff....118
指令 17　GTN_BufLmtsOn....119
指令 18　GTN_BufMove....119
指令 19　GTN_BufSetStopIo....120
指令 20　GTN_Close....121
指令 21　GTN_ClrSts....121
指令 22　GTN_Compile....121
指令 23　GTN_CrdClear....122
指令 24　GTN_CrdData....122
指令 25　GTN_CrdHsOn....122
指令 26　GTN_CrdHsOff....123
指令 27　GTN_CrdSpace....123
指令 28　GTN_CrdStart....123
指令 29　GTN_CrdStatus....124
指令 30　GTN_Download....124
指令 31　GTN_EcatIOReadInput....125
指令 32　GTN_EcatIOWriteOutput....125
指令 33　GTN_EcatSDODownload....126
指令 34　GTN_EcatSDOUpLoad....126
指令 35　GTN_EnableLeadScrewComp....126
指令 36　GTN_EncScale....127
指令 37　GTN_GearStart....127
指令 38　GTN_GetAxisBand....128
指令 39　GTN_GetAxisEncAcc....128
指令 40　GTN_GetAxisEncPos....128

指令 41　GTN_GetAxisEncVel....129
指令 42　GTN_GetAxisError....129
指令 43　GTN_GetAxisPrfAcc....129
指令 44　GTN_GetAxisPrfPos....130
指令 45　GTN_GetAxisPrfVel....130
指令 46　GTN_GetBacklash....130
指令 47　GTN_GetClock....131
指令 48　GTN_GetClockHighPrecision....131
指令 49　GTN_GetCompensate2D....131
指令 50　GTN_GetCompensate2DTable....132
指令 51　GTN_GetCompensate2DValue....132
指令 52　GTN_GetCrdHsPrm....132
指令 53　GTN_GetCrdPos....133
指令 54　GTN_GetCrdPrm....133
指令 55　GTN_GetCrdStopDec....133
指令 56　GTN_GetCrdVel....134
指令 57　GTN_GetEcatAxisAI....134
指令 58　GTN_GetEcatAxisAtlCurrent....134
指令 59　GTN_GetEcatAxisAtlTorque....134
指令 60　GTN_GetEcatAxisDI....135
指令 61　GTN_GetEcatAxisDO....135
指令 62　GTN_GetEcatAxisMode....135
指令 63　GTN_GetEcatAxisPE....136
指令 64　GTN_GetEcatEncPos....136
指令 65　GTN_GetEcatEncVel....137
指令 66　GTN_GetEcatHomingStatus....137
指令 67　GTN_GetEcatRawData....137
指令 68　GTN_GetEcatSlaves....138
指令 69　GTN_GetEncPos....138
指令 70　GTN_GetEncVel....138
指令 71　GTN_GetFunId....139
指令 72　GTN_GetGearMaster....139
指令 73　GTN_GetGearRatio....139
指令 74　GTN_GetJogPrm....140
指令 75　GTN_GetMcEcatAxisNum....140
指令 76　GTN_GetPlsPos....141
指令 77　GTN_GetPlsVel....141
指令 78　GTN_GetPos....141
指令 79　GTN_GetPosScale....142
指令 80　GTN_GetPrfAcc....142
指令 81　GTN_GetPrfMode....142
指令 82　GTN_GetPrfPos....143
指令 83　GTN_GetPrfVel....143
指令 84　GTN_GetRemainderSegNum....143
指令 85　GTN_GetRetainValue....144
指令 86　GTN_GetSense....144

指令 87　GTN_GetSoftLimit 144
指令 88　GTN_GetSoftLimitMode 145
指令 89　GTN_GetLimitStatus 145
指令 90　GTN_GetStopDec 145
指令 91　GTN_GetSts 146
指令 92　GTN_GetThreadSts 146
指令 93　GTN_GetTouchProbeStatus 147
指令 94　GTN_GetTrapPrm 147
指令 95　GTN_GetUserSegNum 148
指令 96　GTN_GetVarId 148
指令 97　GTN_GetVarValue 148
指令 98　GTN_GetVel 149
指令 99　GTN_GetVersion 149
指令 100　GTN_GetVersionEx 149
指令 101　GTN_InitEcatComm 150
指令 102　GTN_InitLookAhead 150
指令 103　GTN_IsEcatReady 150
指令 104　GTN_LmtsOff 151
指令 105　GTN_LmtsOn 151
指令 106　GTN_LnXY 151
指令 107　GTN_LnXYG0 152
指令 108　GTN_LnXYZ 152
指令 109　GTN_LnXYZA 153
指令 110　GTN_LnXYZAG0 154
指令 111　GTN_LnXYZG0 154
指令 112　GTN_LoadConfig 155
指令 113　GTN_Open 155
指令 114　GTN_PauseThread 155
指令 115　GTN_PrfGear 156
指令 116　GTN_PrfJog 156
指令 117　GTN_PrfTrap 156
指令 118　GTN_ProfileScale 157
指令 119　GTN_Reset 157
指令 120　GTN_RunThread 157
指令 121　GTN_SetAxisBand 157
指令 122　GTN_SetBacklash 158
指令 123　GTN_SetCompensate2D 158
指令 124　GTN_SetCompensate2DTable 159
指令 125　GTN_SetCrdPrm 160
指令 126　GTN_SetCrdStopDec 161
指令 127　GTN_SetEcatAxisDO 161
指令 128　GTN_SetEcatAxisMode 162
指令 129　GTN_SetEcatAxisPT 162
指令 130　GTN_SetEcatAxisPV 162
指令 131　GTN_SetEcatHomingPrm 163
指令 132　GTN_SetEcatRawData 164

指令 133　GTN_SetEncPos 164
指令 134　GTN_SetGearMaster 164
指令 135　GTN_SetGearRatio 165
指令 136　GTN_SetHomingMode 166
指令 137　GTN_SetJogPrm 166
指令 138　GTN_SetLeadScrewComp 167
指令 139　GTN_SetOverride 167
指令 140　GTN_SetPlsPos 167
指令 141　GTN_SetPos 168
指令 142　GTN_SetPosScale 168
指令 143　GTN_SetPrfPos 168
指令 144　GTN_SetRetainValue 168
指令 145　GTN_SetSense 169
指令 146　GTN_SetSoftLimit 169
指令 147　GTN_SetSoftLimitMode 169
指令 148　GTN_SetStopDec 170
指令 149　GTN_SetTouchProbeFunction 170
指令 150　GTN_SetTouchProbeFunctionEx 171
指令 151　GTN_SetTrapPrm 171
指令 152　GTN_SetUserSegNum 172
指令 153　GTN_SetVarValue 173
指令 154　GTN_SetVel 173
指令 155　GTN_StartEcatComm 173
指令 156　GTN_StartEcatHoming 173
指令 157　GTN_Stop 174
指令 158　GTN_StopEcatHoming 175
指令 159　GTN_StopThread 175
指令 160　GTN_SynchAxisPos 175
指令 161　GTN_TerminateEcatComm 176
指令 162　GTN_Update 176
指令 163　GTN_ZeroPos 176

13.2　例程索引

例程 3-1　检测 GTN 指令是否正常执行 15
例程 4-1　初始化配置控制器 25
例程 5-1　EtherCAT 初始化 32
例程 5-2　采用 3 号回零方式 33
例程 5-3　探针 1 上升沿连续捕获（以 GTHD 为例） 33
例程 5-4　EtherCAT IO 的使用 34
例程 6-1　获取轴 1 的轴状态、运动模式、位置、速度和加速度 38
例程 7-1　点位运动 43
例程 7-2　Jog 运动 46
例程 7-3　电子齿轮跟随 51
例程 7-4　建立坐标系 56

例程 7-5　直线插补例程 57
例程 7-6　圆弧插补例程 61
例程 7-7　插补 FIFO 管理 64
例程 7-8　前瞻预处理例程 69
例程 7-9　刀向跟随功能 GTN_BufMove 72
例程 7-10　刀向跟随功能 GTN_BufGear 75
例程 7-11　刀向跟随功能——实际工件加工 77
例程 7-12　开启 DMA 快速通道 79
例程 8-1　读取轴编码器位置值 82
例程 8-2　访问内部脉冲计数器 83
例程 9-1　软限位使用 85
例程 9-2　掉电存储操作 87
例程 10-1　运动程序单线程累加求和 91
例程 10-2　运动程序多线程累加求和 93
例程 11-1　读取运动控制器版本号 102
例程 11-2　电机到位检测功能 104
例程 11-3　螺距误差补偿 108
例程 11-4　二维位置补偿 109

13.3　表格索引

表 1-1　指令列表 7
表 3-1　运动控制器指令返回值定义 14
表 4-1　下载配置文件指令 24
表 4-2　配置信息修改指令列表 24
表 4-3　控制器配置初始化状态 26
表 5-1　EtherCAT 库指令列表 27
表 6-1　运动状态检测指令列表 36
表 6-2　轴状态定义 37
表 7-1　设置运动模式指令列表 42
表 7-2　点位运动模式指令列表 42
表 7-3　Jog 运动模式指令列表 46
表 7-4　电子齿轮运动模式指令列表 49
表 7-5　插补运动模式指令列表 53
表 8-1　运动控制器硬件资源 81
表 8-2　访问编码器指令列表 81
表 8-3　访问内部脉冲计数输入指令列表 82
表 9-1　软限位指令列表 84
表 9-2　掉电功能指令列表 87
表 10-1　运动程序指令列表 89
表 10-2　可在运动程序中使用的指令 99
表 11-1　打开/关闭运动控制器指令列表 101
表 11-2　读取固件版本号指令列表 101
表 11-3　固件版本号的定义格式 101
表 11-4　读取系统时钟指令列表 102

表 11-5　打开/关闭电机使能信号指令列表 102
表 11-6　维护位置值指令列表 102
表 11-7　电机到位检测指令列表 103
表 11-8　反向间隙补偿指令列表 106
表 11-9　螺距误差补偿指令列表 107
表 11-10　反螺距误差补偿指令列表 108

13.4　图片索引

图 4-1　开环运动控制系统的配置 17
图 4-2　闭环运动控制系统的配置 18
图 4-3　MotioStudio 运动控制器管理软件界面 19
图 4-4　打开控制器配置 19
图 4-5　axis 配置对控制系统的影响 20
图 4-6　axis 配置界面 1 20
图 4-7　axis 配置界面 2 21
图 4-8　profile 配置界面 22
图 4-9　encoder 配置界面 23
图 4-10　encoder 配置对控制系统的影响 23
图 4-11　encoder 输入脉冲反转项的影响 23
图 4-12　生成配置文件界面 24
图 5-1　PDO 配置界面 29
图 5-2　GTHD 默认配置之 PDO 信息 30
图 5-3　EtherCAT IO 默认配置之 PDO 信息 30
图 5-4　EtherCAT 耦合器和 IO 的 PDO 配置信息 31
图 5-5　EtherCAT 耦合器和 IO 的 IO Mapping 信息 32
图 7-1　点位运动速度曲线 43
图 7-2　点位运动速度规划 44
图 7-3　Jog 模式速度曲线 46
图 7-4　Jog 模式动态改变目标速度 47
图 7-5　电子齿轮模式速度曲线 50
图 7-6　电子齿轮模式主轴速度规划 51
图 7-7　电子齿轮模式从轴速度规划 51
图 7-8　直线插补示意图 55
图 7-9　圆弧插补示意图 55
图 7-10　加工坐标系偏移量示意图 56
图 7-11　不同 evenTime 下的速度曲线 57
图 7-12　直线插补例程运动轨迹 58
图 7-13　圆弧插补逆时针方向 60
图 7-14　半径取正值/负值圆弧插补示意图 60
图 7-15　圆心坐标描述方法示意图 61
图 7-16　圆弧插补例程运动轨迹 61
图 7-17　插补主运动与辅助运动流程 64
图 7-18　插补 FIFO 管理例程之换刀轨迹 65
图 7-19　使用前瞻与不使用前瞻的速度规划区别 67

图 7-20　使用和不使用前瞻预处理功能模块的速度曲线对比图.....68
图 7-21　前瞻预处理流程图.....68
图 7-22　前瞻预处理例程之运动轨迹图.....69
图 7-23　没有进行前瞻预处理的合成速度曲线.....71
图 7-24　进行了前瞻预处理后的合成速度曲线.....71
图 7-25　刀向跟随功能 GTN_BufMove 的运动轨迹.....72
图 7-26　插补缓存区内的点位运动速度图.....74
图 7-27　刀向跟随功能 GTN_BufGear 的运动轨迹.....75
图 7-28　插补缓存区内的跟随运动速度图.....76
图 7-29　刀向跟随功能之工件尺寸和刀运动轨迹.....77
图 8-1　开环控制系统示意图.....81
图 9-1　轴运动范围.....84
图 10-1　运动程序与应用程序的关系.....88
图 10-2　线程、函数和数据页的关系.....90
图 11-1　电机到位检测功能.....103